JCST Journal of Computer Science & Technology

Indexed in: SCIE, EI, INSPEC, CBST, DBLP, etc.

Sponsored by: ICT, CAS & CCF

Enhancing Storage Efficiency and Performance: A Survey of Data Partitioning Techniques

Liu Peng-Ju, Li Cui-Ping, Chen Hong

View online: http://doi.org/10.1007/s11390-024-3538-1

Articles you may be interested in

IMPULP: A Hardware Approach for In-Process Memory Protection via User-Level Partitioning

Yang-Yang Zhao, Ming-Yu Chen, Yu-Hang Liu, Zong-Hao Yang, Xiao-Jing Zhu, Zong-Hui Hong, Yun-Ge Guo

Journal of Computer Science and Technology. 2020, 35(2): 418-432 http://doi.org/10.1007/s11390-020-9703-2

Improved Task and Resource Partitioning Under the Resource-Oriented Partitioned Scheduling

Ze-Wei Chen, Hang Lei, Mao-Lin Yang, Yong Liao, Jia-Li Yu

Journal of Computer Science and Technology. 2019, 34(4): 839-853 http://doi.org/10.1007/s11390-019-1945-5

A Novel Hardware/Software Partitioning Method Based on Position Disturbed Particle Swarm Optimization with Invasive Weed Optimization

Xiao-Hu Yan, Fa-Zhi He, Yi-Lin Chen

Journal of Computer Science and Technology. 2017, 32(2): 340-355 http://doi.org/10.1007/s11390-017-1714-2

HybridTune: Spatio-Temporal Performance Data Correlation for Performance Diagnosis of Big Data Systems

Rui Ren, Jiechao Cheng, Xi-Wen He, Lei Wang, Jian-Feng Zhan, Wan-Ling Gao, Chun-Jie Luo

Journal of Computer Science and Technology. 2019, 34(6): 1167-1184 http://doi.org/10.1007/s11390-019-1968-y

A Cost-Efficient Approach to Storing Users' Data for Online Social Networks

Jing-Ya Zhou, Jian-Xi Fan, Cheng-Kuan Lin, Bao-Lei Cheng

Journal of Computer Science and Technology. 2019, 34(1): 234-252 http://doi.org/10.1007/s11390-019-1907-y

Bigflow: A General Optimization Layer for Distributed Computing Frameworks

Yun-Cong Zhang, Xiao-Yang Wang, Cong Wang, Yao Xu, Jian-Wei Zhang, Xiao-Dong Lin, Guang-Yu Sun, Gong-Lin Zheng, Shan-Hui Yin, Xian-Jin Ye, Li Li, Zhan Song, Dong-Dong Miao

Journal of Computer Science and Technology. 2020, 35(2): 453-467 http://doi.org/10.1007/s11390-020-9702-3

JCST Official

JCST WeChat

JCST Homepage: https://jcst.ict.ac.cn
SPRINGER Homepage: https://www.springer.com/journal/11390
E-mail: jcst@ict.ac.cn
Online Submission: https://mc03.manuscriptcentral.com/jcst
Twitter: JCST_Journal

Liu PJ, Li CP, Chen H. Enhancing storage efficiency and performance: A survey of data partitioning techniques. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(2): 346-368 Mar. 2024. DOI: 10.1007/s11390-024-3538-1

Enhancing Storage Efficiency and Performance: A Survey of Data Partitioning Techniques

Peng-Ju Liu (刘鹏举), Cui-Ping Li* (李翠平), *Distinguished Member, CCF*
and Hong Chen (陈　红), *Distinguished Member, CCF*

School of Information, Renmin University of China, Beijing 100872, China
Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education, Beijing 100872, China

E-mail: liupengju@ruc.edu.cn; licuiping@ruc.edu.cn; chong@ruc.edu.cn

Received June 21, 2023; accepted February 29, 2024.

Abstract Data partitioning techniques are pivotal for optimal data placement across storage devices, thereby enhancing resource utilization and overall system throughput. However, the design of effective partition schemes faces multiple challenges, including considerations of the cluster environment, storage device characteristics, optimization objectives, and the balance between partition quality and computational efficiency. Furthermore, dynamic environments necessitate robust partition detection mechanisms. This paper presents a comprehensive survey structured around partition deployment environments, outlining the distinguishing features and applicability of various partitioning strategies while delving into how these challenges are addressed. We discuss partitioning features pertaining to database schema, table data, workload, and runtime metrics. We then delve into the partition generation process, segmenting it into initialization and optimization stages. A comparative analysis of partition generation and update algorithms is provided, emphasizing their suitability for different scenarios and optimization objectives. Additionally, we illustrate the applications of partitioning in prevalent database products and suggest potential future research directions and solutions. This survey aims to foster the implementation, deployment, and updating of high-quality partitions for specific system scenarios.

Keywords data partitioning, survey, partitioning feature, partition generation, partition update

1 Introduction

In the era of big data, effectively processing massive data has emerged as a critical issue. Database partitioning, a fundamental yet challenging task, simplifies data manipulations by breaking down large datasets into smaller, easy-to-manage partitions based on specified criteria and storing them separately across multiple data blocks. A well-designed partition scheme significantly impacts system performance, resource utilization, and manageability, making it an indispensable strategy for database administrators (DBAs). Partitioning is often optimized for specific purposes in various database management systems (DBMSs). In multi-disk databases, it distributes data across various disks to facilitate better disk collaboration and accelerate read/write operations. In distributed databases, partitioning effectively mitigates machine node imbalances caused by overloading data and queries. Moreover, distributing large-scale datasets to multiple nodes, inclusive of replicas, can boost system availability and scalability. In parallel databases, it enables multiple processing units or cores to work on different parts of the data simultaneously. In NoSQL databases, driven by new data types and data storage/retrieval mechanisms, partitioning is crafted to better manage large volumes of unstructured or semi-structured data.

From the perspective of physical characteristics, partitioning can be broadly classified into three types:

Survey

The work was supported by the National Key Research and Development Program of China under Grant No. 2023YFB4503603, the National Natural Science Foundation of China under Grant Nos. 62072460, 62076245, and 62172424, and the Beijing Natural Science Foundation under Grant No. 4212022.

*Corresponding Author

©Institute of Computing Technology, Chinese Academy of Sciences 2024

horizontal partitioning (HP), vertical partitioning (VP), and irregular partitioning (IP), as detailed in Table 1. HP operates on a row-wise basis, keeping complete tuples within each partition, whereas VP functions column-wise, allowing incomplete yet consistent column data. IP, on the other hand, focuses on the data itself, without imposing strict restrictions on how it is partitioned. Thus, in terms of partition shape, both HP and VP divide the table space into rectangular areas, whereas IP allows partitions of arbitrary shapes, including rectangles. IP designs partition shapes tailored to query access patterns to achieve optimal query efficiency, ideal for online analytical processing (OLAP) and hybrid transactional/analytical processing (HTAP) applications. HP and VP also take into account partial record integrity to facilitate online transaction processing (OLTP), thereby making them suitable for any load scenario.

Data partitioning can be designed based on the database schema, data and load distribution, or a combination of these features. Schema-driven approaches examine the join relationships among tables to centrally allocate tuples involved in join operations. Data-driven approaches commonly employ domain and hash values of column values to create partitions. Query-driven approaches concentrate on mining nested filtering rules from queries to ensure each tuple is assigned to the most appropriate partition.

Other physical designs also significantly impact query latency, disk space usage, and more. To elucidate the role of partitioning, we next briefly describe how it differs from other design strategies.

Partition vs Storage Structure. Partitioning specifies which data should be stored in the same block file, while storage structure solves how the data is organized within a block. For example, Parquet[1], a widely adopted column-store file format in HDFS (Hadoop Distributed File System)①, provides efficient data compression and encoding schemes to enhance the performance of read-intensive queries.

Partition vs Index. Index is an auxiliary data structure designed for quickly locating and retrieving tuples, such as 1-dimensional indexes (B-tree[2]), and n-dimensional indexes (KD-tree[3], R-tree[4]). However, its performance tends to degrade when handling high-dimensional data or certain types of queries. In contrast, partitioning performs well in these scenarios.

Partition vs Materialized View. Materialized view techniques[5, 6] adopt a space-for-time strategy, creating views separating queried data copies from raw data and routing relevant queries to the most suitable view for faster execution. However, copying the complete query results requires additional storage space.

We present a detailed partitioning workflow and review a wide spectrum of existing partitioning studies. Some studies[7, 8] share a similar topic to ours; however, their focus lies on data-driven horizontal partitioning for specific environments (e.g., Hadoop cluster②). Our survey, in contrast, considers a broader range of generalized scenarios. We explore various partition types and place greater emphasis on partitioning requirements, design details, and the implementation process. We further delve into the feature extraction and cost model design before partitioning, along with addressing the data and load update issues after partitioning.

This paper is organized as follows: Section 2 provides an overview of data partitioning, including its four-stage workflow and core modules. Sections 3–5 explore the development trajectory of partitioning, incorporating classical approaches to horizontal, vertical, and irregular partitioning, respectively. Section 6 summarizes the support for partitioning in industry-leading database products. Section 7 gives open problems in this field and potential solutions. Finally, we conclude the survey in Section 8.

2 Data Partitioning Overview

The partitioning workflow typically comprises four stages, as depicted in Fig.1. Stage 1, feature extraction, addresses the issue of what to use for partition-

Table 1. Comparison of Three Common Partition Types

Type	Partition Strategy	Partition Shape	Scenario		
			OLTP	OLAP	HTAP
HP	Row-wise	Rectangular	✔	✔	✔
VP	Column-wise	Rectangular	✔	✔	✔
IP	Data-wise	Arbitrary	✘	✔	✔

①https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, Mar. 2024.
②https://hadoop.apache.org/docs/stable/index.html, Mar. 2024.

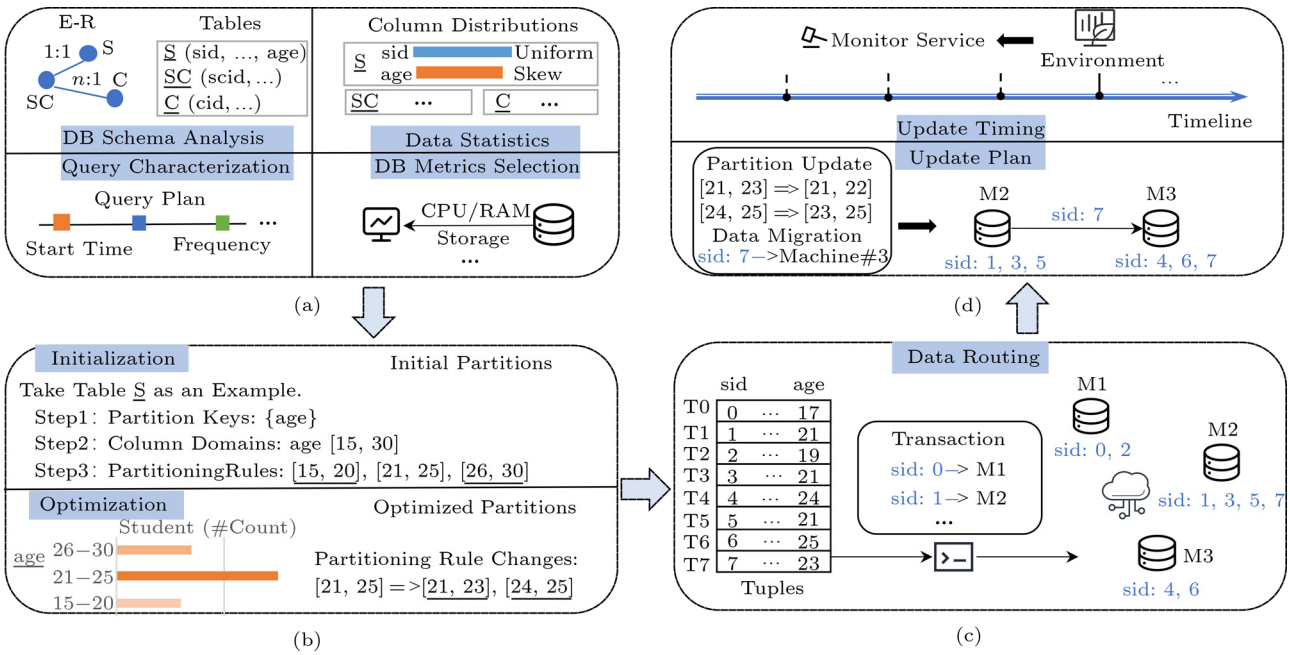

Fig.1. Data partitioning workflow. (a) Feature extraction. (b) Partition generation. (c) Partition deployment. (d) Partition update.

ing. This stage entails analyzing the database (DB) schema, parsing representative queries, conducting column data statistics, and selecting system optimization metrics. Stage 2, partition generation, includes two subtasks: partition initialization, which quickly establishes initial partitions using a low-complexity algorithm, and partition optimization, where the initial solution is iteratively refined based on predefined cost models. Stage 3, partition deployment, involves routing data to partition files via automated write transactions based on created partition structures. Stage 4, automatic partition update, timely adjusts partitions to sustain stable system performance amid data, load, and hardware resource uncertainties, which includes deciding update timings and formulating detailed update plans accordingly.

Consider a teaching system comprising three tables: student (S), course (C), student course (SC). Before partitioning a table (e.g., S), we first analyze its entity-relationship (E-R) graph and common column data distributions, gathering query information and system metrics as necessary. Assuming the age column has been selected as the partition key, initial partitioning rules are derived from its value domain, and skew partitions are further split according to the column histogram statistics. With the partitions, eight given tuples (T0, . . . ,T7) are distributed across three machines (M1, M2, M3). Subsequently, a service is established to continuously monitor the environment. When detecting an overload on M2, the partition boundaries for M2 ([21, 23] ⇒ [21, 22]) and M3 ([24, 25] ⇒ [23, 25]) are promptly adjusted, and a data migration plan is devised to move tuple T7 from M2 to M3.

Fig.2 displays a framework comprising five key modules used in the partitioning workflow. This survey concentrates on the modules highlighted in green.

1) *Deployment Scenario.* Partitioning optimization objectives, such as performance, manageability, and device costs, are greatly affected by system environments, user requirements, and the storage devices used. For instance, in a distributed database, partitioning tasks are more complex, necessitating the considerations of factors like multi-node clusters, node replicas, and network latency, to ensure uniform partition access and reduce cross-node operations. Tables 2 and 3 offer categorizations and symbolic representations of common optimization objectives and database environments, respectively.

2) *Partition Type.* Before designing partitions, it is necessary to choose the partition type to use based on the given scenario, as shown in Table 1.

3) *Cost Model.* After identifying the deployment scenario and deciding the partition type, a cost model is created to assess the given partition scheme and its associated update plan. There are three types of cost estimations: optimizer-based models, simplifying cost design at the expense of accuracy; network-based learning models, offering high precision but requiring sufficient metric samples and extensive training over-

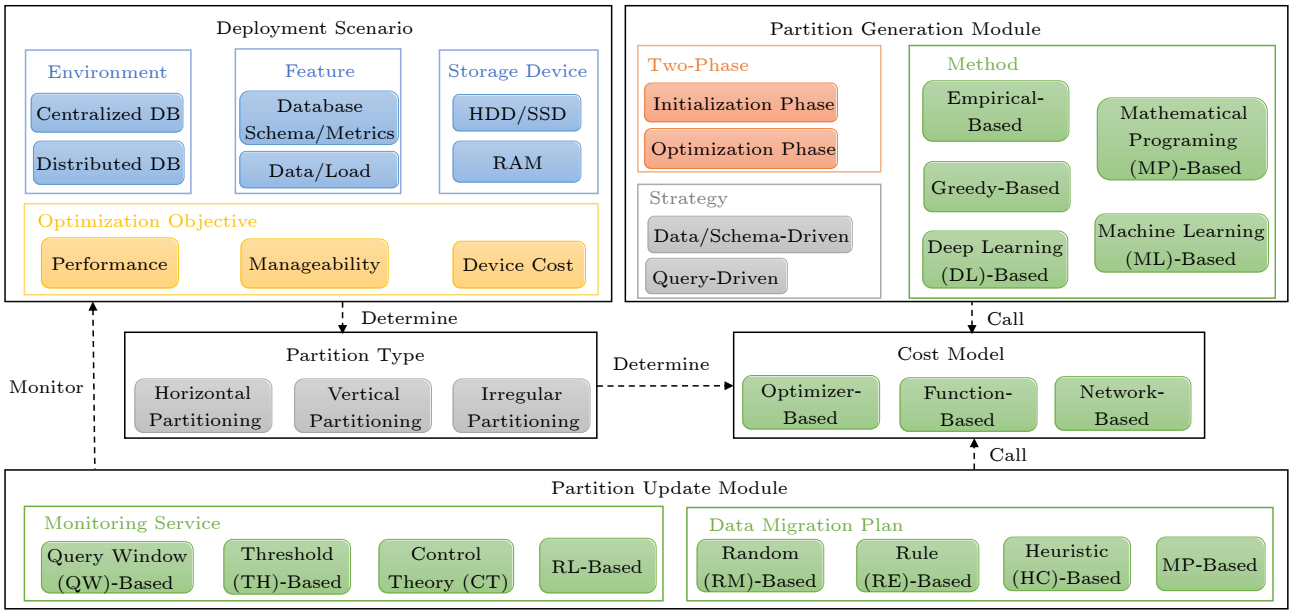

Fig.2. Overview of a modular framework for data partitioning technologies.

Table 2. Classification of Optimization Objectives

Symbol	Description
O1	Data balancing
O2	Load balancing
O3	Query cost estimation
O4	Query latency or system throughput
O5	Data transfer overhead
O6	Number of distributed transactions
O7	Coordination cost between machines
O8	Layout generation time
O9	Memory footprint
O10	Cache utilization
O11	Device cost
O12	Storage space

Table 3. Classification of System Environments

Symbol	Description
E-CH/S	Centralized database (HDD or SSD)
E-DH/S	Distributed database (HDD or SSD)
E-CM	Centralized database (RAM)
E-DM	Distributed database (RAM)

head; and function-based models, being more balanced due to their flexible and comprehensive design.

4) *Partition Generation Module.* The partition generation process, encompassing both initialization and optimization phases, is shared across most techniques and is guided by specifically designed cost models. Here, we categorize existing methods according to their algorithm types, as detailed in Table 4.

5) *Partition Update Module.* This module timely updates inefficient partitions, especially those that are query-driven and become fragile under new loads. It

Table 4. Classification of Partitioning Generation Methods

Symbol	Description
Greedy-based	Making partitioning decisions at each step based on predefined heuristic rules
Empirical-based	Developing schemes manually based on observation and experience
ML-based	Using traditional machine learning models
DL-based	Using deep learning algorithms
MP-based	Constructing mathematical programming equations with objectives and constraints

features two core components. a) The monitoring service uses five optional mechanisms (refer to Table 5) to determine repartition times. The term "window" in M-QW denotes a container for monitoring queries, cleared in time after each repartitioning. M-TH sets threshold conditions as feature boundaries like fixed time intervals and minimum query latency. b) Once repartitioning is triggered, a data migration plan is strictly made using a specific strategy (refer to Table 6), with considerations for data transfer overhead and potential benefits of new partitions. These plans are

Table 5. Classification of Monitoring Methods

Symbol	Description
M-QW	Monitoring whether a window is filled as new queries arrive
M-TH	Monitoring whether threshold conditions are met
M-CT	A feedback mechanism to determine the repartition timing
M-RL	Training an agent to automatically take repartition action based on the environmental feedback
M-SD	Service on demand

Table 6. Classification of Data Migration Plans

Symbol	Description
D-RM	One partition is randomly selected from those that need adjustment for partial reorganization
D-RE	Partitions are swapped using predefined rules, operators, structures, and algorithms
D-HC	Using heuristic information such as evaluation functions and metrics to guide data migration
D-MP	Converting the data migration into mathematical optimization problems, e.g., dynamic programing (DP), integer linear programming (ILP)

executed in the background, uniformly controlled by the master node, to ensure minimal impact on normal transaction execution.

3 Horizontal Partitioning

In Subsection 3.1, we define the static and dynamic horizontal partitioning (HP) problems, followed by an introduction to the extracted features in HP (Subsection 3.2). We then describe and summarize the research development of HP methods (Subsections 3.3 and 3.4), and the design of the HP-wise cost models (Subsection 3.5), based on different system environments. Fig.3 depicts the timeline of HP methods.

3.1 Formalization

Definition 1 (Static Horizontal Partitioning). *Static horizontal partitioning aims to find a classifier $\phi(\cdot)$ for a table with m tuples $D=(e_1,\ e_2,\ \ldots,\ e_m)$ and n collected queries $Q=(q_1,\ q_2,\ \ldots,\ q_n)$. When a new tuple arrives, the classifier ϕ assigns it to the specified partition P in time, i.e., $P=\phi(e),\ \forall e\in D$. The classifier partitions all tuples into k distinct partitions, represented as $\mathbb{P}=(P_1,\ P_2,\ \ldots,\ P_k)$, to achieve optimal system objectives such as low query latency and high system throughput. The total cost of processing Q over $\mathbb{P}$ is denoted by $C(\phi,\ Q)$.*

$$\phi^*=\arg\min_{\phi} C\big(\mathbb{P}\Leftarrow\phi(D),\ Q\big).$$

Definition 2 (Dynamic Horizontal Partitioning). *For a database running over future n intervals $T=(t_1,\ t_2,\ \ldots,\ t_n)$, where the corresponding submitted queries are $\mathbb{Q}=(Q_1,\ Q_2,\ \ldots,\ Q_n)$, and the initial partition scheme is $\mathbb{P}_0$, our goal is to design an optimal controller $\mathcal{G}^*$. This controller analyzes current partitions and queries at each interval t_i to decide whether to deploy new partitions ($\mathbb{P}_i$) or maintain existing ones ($\mathbb{P}_i=\mathbb{P}_{i-1}$). The aim is to minimize the sum of I/O costs for achieving optimization objectives and data migration costs during the entire running process.*

$$\min\sum_{i=1}^{n}\left(C_i^{(1)}+C_i^{(2)}\right),$$
$$\text{s.t.}\begin{cases}\mathbb{P}_i &= \mathcal{G}\left(Q_{i-1},\mathbb{P}_{i-1}\right),\\ C_i^{(1)} &= C_{\mathrm{r}}\left(\mathbb{P}_{i-1},\mathbb{P}_i\right),\\ C_i^{(2)} &= \sum\limits_{q\in Q_i} C\left(\mathbb{P}_i,q\right),\end{cases}$$

where $C_{\mathrm{r}}(\mathbb{P}_i,\ \mathbb{P}_j)$ calculates the minimum data migration cost required to reorganize the partition files when the partition scheme changes from $\mathbb{P}_i$ to $\mathbb{P}_j$.

3.2 Feature Extraction

Database Schema. Depending on the given database schema, we can 1) classify tables into large/small ones based on the number of tuples, and static/dynamic ones based on data changes; 2) analyze the characteristics of numerical columns, including data type, constraints, indexes, and triggers, etc.; 3) learn the foreign key relationships and constraints between tables to help construct co-partitions[19, 28, 32, 33].

Table Data. When analyzing numerical column data, distribution types (e.g., uniform, skewed/hot

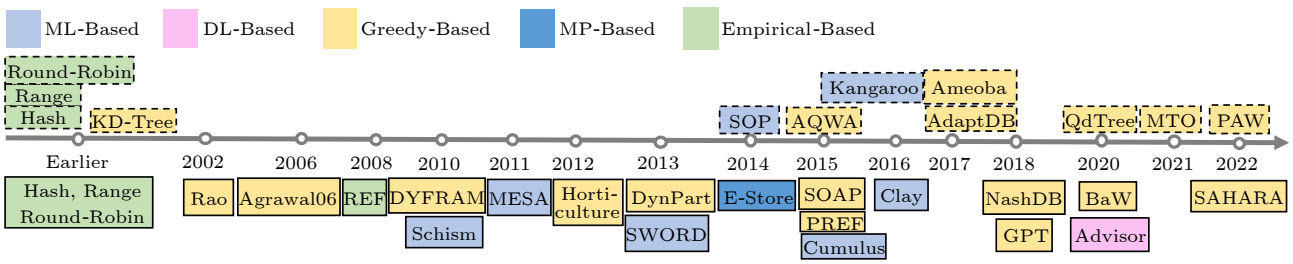

Fig.3. Timeline of HP research development, including general empirical-based approaches (round-robin, range, and hash), as well as on-axis studies (KD-tree[3], SOP[9], AQWA[10], Kangaroo[11], Ameoba[12], AdaptDB[13], QdTree[14], MTO[15], and PAW[16]) focused on centralized environments and off-axis studies (Rao[17], Agrawal06[18], REF[19], DYFRAM[20], Schism[21], MESA[22], Horticulture[23], DynPart[24], SWORD[25], E-Store[26], SOAP[27], PREF[28], Cumulus[29], Clay[30], NashDB[31], GPT[32], BaW[33], Advisor[34], and SAHARA[35]) on distributed environments.

spot[20, 23, 24, 26, 35], discrete) and domain statistical metrics (e.g., median[12, 13, 16], maximum, and minimum values[15, 16]) are considered. These can also be depicted using histogram technologies[15, 20].

Workload. In HP, important query logical features (e.g., filter conditions, join keys, operator cost estimates, and SQL keywords) and physical features (e.g., read-to-write ratios, occurrence frequencies, submission/completion times, and inserted/updated rows) can be extracted from query plans. Some studies count partition or tuple access frequencies[21, 25, 30] to identify hot and cold data[26, 35] by tracking query-tuple accesses. Furthermore, load can be classified as either heavy or light based on the average query arrival rate.

Database Runtime Metric. OS-level metrics related to HP are chosen to monitor the database state, including resource usage (e.g., memory, CPU, disk), performance (e.g., query latency, throughput), and machine hotspots.

3.3 Partitioning Process in Centralized Databases

In this subsection, we discuss studies designed for centralized systems or those that neglect factors such as multi-nodes, replicas, and network costs.

Empirical-Based. Range partitioning typically splits data based on a pre-defined range of values derived from partition keys. This method is suitable for data with prior statistics but requires careful selection of partition boundaries, which is difficult for large-scale datasets. Hash partitioning maps tuples to specific partitions using a hash function, ideal for unordered data. Round-robin partitioning is a special type of hash partitioning that assigns data to available N machine nodes in a circular fashion, i.e., assigning the i-th data row to the $(i \mod N)$-th node, to ensure equi-sized balanced partitions. These traditional methods are data-driven and do not require prior load knowledge.

ML-Based. SOP[9] (Skipping-Oriented Partitioning) adopts the Apriori algorithm[36] to extract m representative filter predicates from load, and converts each tuple into an m-bit one-hot feature vector with each bit indicating tuple-predicate satisfaction. These vectors are clustered into different blocks via the Ward algorithm[37], with each block generating a union vector (as known as partition map) by performing bitwise OR operations on its vectors. These maps act as a classifier, partitioning new data and guiding incoming queries to skip unnecessary blocks. Kangaroo[11] utilizes grid and tree structures for partitioning. In a 2D table space, the grids are represented by two bit strings, with positions marked as 1 acting as the partition boundaries. Kangaroo then applies a genetic algorithm (GA) for partition initialization and merging, deriving the optimal partition scheme. Its tree-based approach replaces the grid with a tree representation within the GA process.

Greedy-Based. To solve SOP limitations, such as the exponential growth in execution time with more predicates, Yang *et al.*[14] proposed a greedy-built query data routing tree (QdTree). QdTree is a binary tree created by selecting the predicate with the maximum split benefit as the split condition at each tree expansion step until no further splits are possible. Each leaf node maintains metadata for routing, with the path from root to leaf serving as the search process for assigning tuples to partitions. Ding *et al.*[15] extended QdTree to multi-table datasets with a multi-table optimizer (MTO), leveraging sideways information passing through joins. MTO periodically computes a reward value to decide the best repartition timing and then uses dynamic programming (DP) to find the optimal reorganization set of non-overlapping subtrees. Li *et al.*[16] proposed PAW (Partitioning Aware of Workload Variance), focusing on creating partitions adaptable to future load variances by scaling historical queries and employing multi-step splits to replace multiple one-step predicate splits in QdTree when splitting smaller nodes.

However, in a new environment where query logs are unavailable, query-driven physical design techniques will become ineffective, leading to the database's cold start issue. Moreover, collecting representative queries is sometimes difficult; for instance, a study[13] on IoT startups revealed that, even after analyzing the first 80% of historical queries, the remaining 20% still contained 57% new queries previously unseen. To tackle this issue, Aly *et al.*[10] developed an adaptive query-workload-aware partitioning (AQWA). AQWA utilizes the KD-tree[3] structure for creating initial partitions with equal spatial points distribution. It dynamically maintains update plans for all visited nodes, considering split gain and data migration costs. To support KNN queries, AQWA uses MinDist and MaxDist indicators[38] along with the virtual grid technology to compute query boundaries. Amoeba[12] initializes a heterogeneous binary tree, similar to KD-tree, and dynamically modifies it for in-

coming queries using three node update operations: swap, pushup, and rotate. AdaptDB[13] adapts Amoeba for join operations by splitting each Amoeba tree based on joined columns. It employs a greedy search strategy to co-partition joined blocks, yielding a superior hyper-join operation over shuffle-join. AdaptDB manages repartitioning via a fixed-length query window, refreshing the tree for new queries, and reallocating old nodes.

Table 7 summarizes the horizontal partitioning techniques discussed above for centralized environments. The "Cost" column indicates whether a cost model is used or not. The "Deployment" column indicates whether the partitions have been deployed in a real database environment. The "Method Content" column uses various symbols to represent different partitioning stages: partition initialization (∇), partition optimization ($\bigcirc$), and partition update ($\circlearrowleft$). The representations are applied to all subsequent tables.

3.4 Partitioning Process in Distributed Databases

Data-driven approaches are universally applicable to various database environments and can always achieve data balancing. However, the performance of query-driven approaches, tailored for E-CH/S environments, might be limited by new factors in E-DH/S environments. Thus, in this subsection, we introduce the studies specifically designed for distributed environments.

3.4.1 Disk Storage Environment

Optimizing data placement on hard and solid-state drives has greater potential in boosting system throughput, due to their slower read/write speeds than memory drives. Early partitioning studies[39–41] in E-DH/S environments relate to physical design tools offering layout suggestions for data and load balancing. However, they do not design a cost function for accurate evaluation of alternative solutions. Rao *et al.*[17] combined a rank-based method with cost estimations derived from query optimizer statistics to quickly recommend partition keys. Agrawal *et al.*[18] refined this by treating workload as a sequence with temporal features, eliminating redundant and inefficient designs. Other similar studies[42, 43] utilize optimizer and load information, adopting greedy and heuristic-based strategies for effective partitioning.

However, while these strategies mentioned above excel in large-scale data scans, they easily incur distributed (i.e., cross-node) calls during small transactions touching only a few tuples.

ML-Based. Schism[21] addresses this issue by minimizing distributed transactions. Fig.4 illustrates its partitioning process. 1) Data preparation, inputting table data and transaction information (omitted). 2) Partitioning. A hypergraph is created, with nodes representing tuples or tuple replicas. Replication edges connect a tuple to its replicas, while transaction edges connect all tuples accessed by the same transaction. A Metis partitioner[44] then splits the hypergraph into multiple balanced partitions with minimal cross-partition transactions. In the illustrated example with five tuples, we get partitions 0 and 1 after graph splitting. 3) Explanation and validation. Decision trees are constructed based on tuple features within each partition to find predicate-based explanations for adapting new data. In Fig.4, the decision tree is construct-

Table 7. Major Horizontal Partitioning Strategies for Centralized Environments

Category	Work	Baseline	Objective	Automatic	Cost	Deployment	Method Content
Empirical	Range, hash, round-robin	N/A	O1, O2	✘	✘	✔	Partitioning by columns or data insertion order$^{\nabla}$
ML	SOP[9]	SimpleRange	O3	✘	✔	✔	Frequent itemset+Ward clustering$^{\nabla}$
ML	Kangaroo[11]	Random schemes	O4, O8	✘	✔	✔	GA-based grid/tree generation$^{\nabla}$; partition initialization using DP^{O}
Greedy	AQWA[10]	Uniform grids	O4, O8	M-TH+ D-RE	✔	✔	Spatial data-based recursive KD-tree$^{\nabla}$; greedy tree node split selection$^{\circlearrowleft}$
Greedy	Ameoba[12], AdaptDB[13]	FullScan, SOP[9]	O3, O5	M-QW+ D-RE/RM	✔	✔	Heterogeneous tree$^{\nabla}$; heuristic-group^{O}; predicate-based tree update$^{\circlearrowleft}$
Greedy	QdTree[14]	SOP[9]	O3	✘	✘	✔	Greedy-based binary predicate tree
Greedy	MTO[15]	QdTree[14]	O3, O8	M-TH+ D-MP	✘	✔	QdTree$^{\nabla}$; join-induced predicates^{O}; tree update using DP$^{\circlearrowleft}$
Greedy	PAW[16]	QdTree[14]	O3	✘	✘	✔	Query deviation prediction+ multi-group split$^{\nabla}$; data replication^{O}

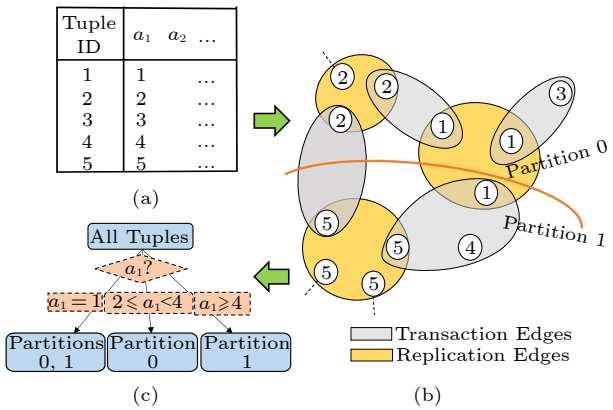

Fig.4. Graph partitioning process introduced in [21]. (a) Input: table data. (b) Hypergraph creation and partitioning. (c) Decision tree construction.

ed with the a_1 column serving as the decision point, using criteria such as $a_1 = 1$, $2 \leqslant a_1 < 4$, and $a_1 \geqslant 4$ for decision branches. The leaf nodes indicate that tuples meeting the specified criteria are allocated to their respective partitions. Nehme *et al.*[22] developed the MEMO-based search algorithm (MESA) for long-running analytical transactions touching large-scale tuples, while Schism adapts to small short-lived transactions. The MEMO structure is a search space for parallel query optimization. MESA generates MEMOs for each query and then faster simulates and explores tree-style partition candidate configurations using a branch and bound strategy.

To adapt Schism to load changes, SWORD[25] compresses the hypergraph into virtual nodes, periodically monitors load variations, and sets a threshold for distributed transaction ratios to determine repartition timings, employing virtual node swaps for incremental graph updates to minimize data movement. Cumulus[29] filters out infrequent transactions and predicts future transaction frequency with an exponential moving average. It dynamically re-partitions data in a user-driven live migration to avoid potential hotspots, balancing the increase in repartitioning overhead with the decrease in distributed transaction costs.

Greedy-Based. DYFRAM[20] addresses the cold start problem by initially creating simple range partitions for equi-width data distribution histograms, then periodically evaluates whether to replicate partitions based on partition size limitations and cross-partition overheads. DynPart[24], designed for continuously growing database (e.g., observation and log data). As data volume increases, DynPart models the affinity between data and partition based on given queries, proposing heuristic rules for efficiently distributing incoming data. Unlike SWORD's approach of isolating updated data during repartitioning, SOAP[27] integrates repartition operations into normal transactions for smooth partition management. SOAP employs a cost-based method to prioritize repartition transactions and utilizes a feedback model for scheduling their executions. NashDB[31] supports user-defined query prioritization and efficient resource use, combining economic models, dynamic programming, and the Munkres algorithm[45] to optimize node usage and minimize data migration costs.

Table 8 summarizes common horizontal partitioning techniques for distributed disk storage environments.

3.4.2 *Distributed Partition Key Recommendation in Disk Storage Environments*

Non-co-located joins cause excessive data transfer overhead among machine nodes, adversely affecting join performance. Co-partitioning tables using shared join keys can significantly reduce data shuffling. We term this problem as Distributed Partition Key Recommendation (DKR). For example, in Spark SQL, data can be organized into multiple buckets according to the hash or range values of selected partition keys. Costa *et al.*[46] verified that creating a consistent number of buckets for join keys across two large tables can significantly boost join performance over traditional sort-merge joins.

Empirical-Based. When facing joins with reference constraints, the query executor requires copying partition keys and strategies from parent to child tables, and subsequently repeats partition merging, splitting, or key updates across all parent-child tables. Eadon *et al.*[19] proposed reference partitioning (REF) that enables partition maintenance operations performed on parent tables to be extended to child tables, ensuring the migration of child tuples is handled as a single atomic operation when the partition key in the parent table is modified.

Greedy-Based. PREF[28] (Predicate-Based Reference Partitioning) improves REF by supporting co-partitioning of tables for any join predicate, not just foreign keys, through tuple duplication. A join graph is defined with each node denoting a table and each edge indicating joins over two tables. PREF assigns weights to each edge as the connected smaller table size, and extracts candidate key configurations from

Table 8. Major Horizontal Partitioning Strategies for Distributed Disk Storage Environments

Category	Work	Baseline	Objective	Automatic	Cost	Deployment	Method Content
ML	Schism[21]	Manual partitions	O1, O2, O6	✘	✘	✔	Metis$^{\triangledown}$, decision tree^{O}
ML	MESA[22]	Rao *et al.*[17], Schism[21]	O3, O8	✘	✘	✘	Memo-based search$^{\triangledown}$; pruning branch and bound tree^{O}
ML	SWORD[25]	Schism[21], simple hash partitions	O1, O2, O4, O7	M-TH+D-HC	✔	✔	Graph compression/partition$^{\triangledown}$; node swapping/replication$^{\circlearrowleft}$
ML	Cumulus[29]	Schism[21]	O2, O7	M-SD+D-RE	✔	✔	Multi-objective cost model; on-demand repartition$^{\circlearrowleft}$
Greedy	DYFRAM[20]	Optimal solution	O3, O7	M-TH+D-HC	✔	✘	Histogram+rule-based replication/partitioning$^{\triangledown}$
Greedy	DynPart[24]	Schism[21]	O3, O7	M-TH+D-HC	✔	✘	Single partition$^{\triangledown}$; affinity-based heuristic strategy^{O}
Greedy	SOAP[27]	SWORD[25]	O2, O4	M-CT	✔	✔	PID-controller$^{\circlearrowleft}$
Greedy	NashDB[31]	SWORD[25], optimal solution	O2, O4, O5	✘	✔	✔	Economic model$^{\triangledown}$; greedy Munkres algorithm$^{\circlearrowleft}$

each query, greedily merging them to minimize the graph weights. GPT[32] reduces data redundancy in PREF. It first selects vertices and edges to be added from the join graph by considering both the storage overhead and shuffle-free query benefits, and then adopts a multi-column partitioning to hash partition key values for each edge. BAW[33] (Best of All Worlds) is an assumption-free framework that uses exact integer linear programming and heuristic variants to transform the DKR problem into a graph matching problem, unlike prior studies[19, 28] that rely on many assumptions not generally applicable.

RL-Based. Hilprecht *et al.*[34] introduced a partition advisor using Q-learning[47] to automatically assess and recommend partition keys under varying loads. The advisor refines a network-centric cost model with actual runtimes and designs a training environment consisting of three parts: 1) State, which is a one-hot encoding of table attributes indicating whether an attribute at each position is a partition key. 2) Action, comprising a candidate set that includes actions to replicate or (de-)activate edges between partition keys. 3) Reward function, which utilizes the cost model to calculate the performance gains of each action as the reward, disregarding data migration overheads.

Table 9 summarizes the partition key recommendation techniques for distributed disk environments.

3.4.3 *Main Memory Storage Environment*

In modern OLTP systems with small, repetitive, and short-lived transactions, applications can keep their entire dataset in memory through widely shared server clusters, making it more feasible to develop new storage system prototypes than to add indexes to traditional disk-oriented DBMSs. H-Store[48] is such a main memory database that supports user-defined layout designs. The studies[23, 26, 30, 35] discussed below are all designed on H-Store, where network latency

Table 9. Major Distributed Partition Key Recommendation Strategies for Optimizing Join Operations

Category	Work	Baseline	Objective	Automatic	Cost	Deployment	Method Content
Empirical	REF[19]	N/A	O4, O12	✘	✘	✔	Reference partitioning$^{\triangledown}$
Greedy	PREF[28]	REF[19]	O4, O12	✘	✔	✔	Schema/query driven design$^{\triangledown}$
Greedy	GPT[32]	PREF[28]	O4, O12	✘	✔	✔	Join graph+hash-based multi-column partitioning$^{\triangledown}$
Greedy	BAW[33]	Greedy matching	O4, O7	✘	✘	✔	Integer linear programming$^{\triangledown}$; graph matching$^{\triangledown}$
DL	Hilprecht *et al.*[34]	PREF[28]	O4	M-RL	✔	✔	Network-centric cost model+ Q-learning algorithm$^{\circlearrowleft}$

and resource utilization have become critical factors.

Horticulture[23] estimates the coordination and skew costs between machine nodes to achieve load balancing and reduce distributed transactions. To handle complex database schemas and a larger number of partitions, it uses a large neighborhood search algorithm converging to near-optimal partitioning solutions within a reasonable time overhead. However, it does not provide any partition update strategy. E-Store[26] dynamically reallocates resource to accommodate demand spikes and new transactions. It periodically collects metrics at the tuple, partition, OS levels, identifies hot keys for hot tuple assignment, and evenly distributes cold data in large chunks for the remaining space. If CPU utilization exceeds a given threshold, E-Store scales cluster nodes and uses a two-tiered bin packing algorithm to optimize tuple-to-partition assignments. Clay[30] enhances E-Store by addressing the issue of accessing tuples in multiple blocks and non-colocated on the same cluster node. It adopts a two-tier partitioning with fine-grained mapping (Metis[44] for hypergraphs) for hot tuples and coarse-grained mapping (simple range/hash strategies) for cold tuples. When some partitions become overloaded, Clay employs a threshold-based sub-graph migration algorithm to update them.

Reducing hardware expenses alongside improving performance is also an important research topic. SAHARA[35] minimizes resource overhead while satisfying all performance objectives by leveraging query access skew to move cold data to cheaper storage layers, retaining only hot data in main memory.

Table 10 summarizes major horizontal partitioning techniques for distributed memory environments.

3.5 Cost Estimation for Horizontal Partition Scheme

Table 11 compares representative HP cost models. Notably, function-based cost models are prevalent, focusing on a wide range of elements including block skipping, join overhead, and hardware resources.

3.5.1 *Centralized Environment*

Most studies[9, 12–16] evaluate partition quality by calculating the number of scanned tuples using a skipping-based cost function. In the SOP[9] model, the given query set Q is initially encoded into n distinct feature vectors $\boldsymbol{F} = (\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_n)$. The number of queries satisfying $\boldsymbol{F}_i$ is represented as z_i. A function $f(P, \boldsymbol{F}_i)$ returns the number of accessed tuples when

Table 10. Major Horizontal Partitioning Strategies for Distributed Main Memory Environments

Category	Work	Baseline	Objective	Automatic	Cost	Deployment	Method Content
Greedy	Horticulture[23]	Schism[21], manual partitions	O2, O6	✘	✔	✔	Skew-aware model+ large-neighborhood search$^{\nabla}$
Greedy	E-Store[26]	Optimal solution	O5	M-TH+D-MP	✘	✔	Two-tiered partitioning$^{\nabla}$; greedy/first-fit$^{\circlearrowleft}$
Greedy	SAHARA[35]	Unpartitioned state, DB-expert	O9, O11	✘	✔	✔	Hot/cold data division$^{\nabla}$; MaxMinDiff range partitions^{O}
ML	Clay[30]	E-Store[26], Metis[44]	O2, O5	M-TH+D-RE	✘	✔	Tuple grouping+graph split$^{\nabla}$; heuristic data migration plan$^{\circlearrowleft}$

Table 11. Comparative Analysis of Major Horizontal Partitioning Cost Models in Diverse Environments

Category	Cost Model	Objective	Environment	Characteristic
Optimizer	Rao *et al.*[17], MESA[22]	O3, O8	E-DH/S	Adjusting query plan node costs for different partitions based on table/index statistics
Function	SOP[9], AdaptDB[13], MTO[15]	O3, O5	E-C(D)H/S	Skipping-based block scan cost and join cost
Function	Horticulture[23]	O2, O6	E-DM	Quantifying the effects of load skew on the cluster
Function	DYFRAM[20], SOAP[27], SWord[25], E-Store[26], Clay[30]	O5, O4, O6, O10	E-DH/S, E-DM	Costs for dynamic environments (replication/repartition operations, cold/hot data)
Function	PREF[28], GPT[32], BAW[33]	O3, O12	E-DH/S	Fine-grained cost designs for the PKR problem
Function	NashDB[31]	O1, O3	E-DH/S	A monetary value function for tuples; converting the HP problem into an economic problem
Function	SAHARA[35]	O11	E-DM	A novel objective for reducing hardware cost
Learning	Hilprecht *et al.*[34]	O4, O8	E-DH/S	A network-centric cost model

running $\boldsymbol{F}_i$ over the given partition P. The total query cost $C(\mathbb{P},\ Q)$ equals the sum of scanning tuples in the data layout $\mathbb{P}$ when executing Q, i.e.,

$$C(\mathbb{P},\ Q)=\sum_{P\in\mathbb{P}}\sum_{i=1}^{n} f(P,\ \boldsymbol{F}_i)\times z_i.$$

However, this model has a significant issue, i.e., the number of scanned block files only affects the cost of scan operators and is not linearly correlate with the final query latency, despite a positive relationship. Thus, it is crucial to consider more factors that affect the overall execution of query plans, such as filter, join, and write operators.

3.5.2 Distributed Environment

In distributed environments, query execution costs are often associated with the number of local and distributed transactions, as well as the uniformity of data and load distribution. Horticulture[23] considers both factors. It defines $C(\mathbb{P},\ Q)$ as the weighted sum of coordination cost C_{coo} and skew factor F_{skew}, i.e.,

$$C(\mathbb{P},\ Q)=\frac{w_{\text{coo}}\times C_{\text{coo}}(\mathbb{P},\ Q)+w_{\text{skew}}\times F_{\text{skew}}(\mathbb{P},\ Q)}{w_{\text{coo}}+w_{\text{skew}}},$$

where w_{coo} and w_{skew} are user-specified weights for the coordination and skew costs of machine nodes, respectively; C_{coo} measures how effectively $\mathbb{P}$ reduces distributed transactions in Q. C_{coo} is computed as the ratio of accessed partitions ($\bar{N}_{\text{par}}$) to the maximum possible partitions ($\hat{N}_{\text{par}}$), which is then scaled by the ratio of cross-partition transactions (N_{dtxn}) to all accessing transactions (N_{txn}). We have:

$$C_{\text{coo}}(\mathbb{P},\ Q)=\frac{\bar{N}_{\text{par}}}{N_{\text{txn}}\times\hat{N}_{\text{par}}}\times\left(1+\frac{N_{\text{dtxn}}}{N_{\text{txn}}}\right).$$

To get the skew factor F_{skew}, Horticulture first computes the skew factor for each t-th interval (SK_t) by dividing the average partition skew value by the ideal skew value, i.e.,

$$SK_t(\mathbb{P},\ Q)=\left(\sum_{i=1}^{|\mathbb{P}|}\log\left(\frac{N_{\text{par}}^i/N_{\text{par}}}{\bar{\rho}_{\text{txn}}}\right)/\hat{N}_{\text{par}}\right)/\log\frac{1}{\bar{\rho}_{\text{txn}}},$$

where N_{par}^i represents the number of transactions accessing the i-th partition, and $\bar{\rho}_{\text{txn}}$ represents the ideal transaction distribution, estimated as $1/\hat{N}_{\text{par}}$.

Next, Horticulture accumulates them to obtain the final skew factor, i.e.,

$$F_{\text{skew}}(\mathbb{P},\ Q)=\frac{\sum_{t=0}^{n} SK_t(\mathbb{P},\ Q)\times N_{\text{txn}}^t}{N_{\text{txn}}},$$

where n is the number of time intervals, and N_{txn}^t is the number of accessing transactions during the t-th interval.

Another representative cost model, as seen in SAHARA[35], utilizes limited device resources to improve performance in two ways: 1) optimizing performance under a given maximum resource budget; 2) optimizing resource budgets for given optimization objectives (denoted as SLA). SAHARA employs the latter, using the β-second rule to classify the given partition as cold or hot and estimating their memory footprint to achieve the set objective.

If a partition is accessed more frequently than every β seconds, it is classified as hot; otherwise, it is deemed cold. Data from cold partitions is loaded from disk as necessary, while data from hot partitions is kept entirely in memory. Given a partition size in bytes ($|P_i|$) and the memory overhead cost per unit of buffer pool (C_{RAM}), the memory footprint of a hot partition is $\mathcal{M}_{\text{hot}}(|P_i|)=C_{\text{RAM}}\times|P_i|$. Considering the estimated access frequency of P_i denoted as $\widehat{N}_i^{\text{col}}$, an allowed maximum query execution time (i.e., SLA), I/O operations per second (PS), the disk cost per page (C_{disk}), and the page size (S_{page}), the memory footprint of a cold partition can be expressed as:

$$\mathcal{M}_{\text{cold}}\left(|P_i|,\ \widehat{N}_i^{\text{col}},\ SLA\right)=\frac{\widehat{N}_i^{\text{col}}}{SLA}\times\left\lceil\frac{|P_i|}{S_{\text{page}}}\right\rceil\times\frac{C_{\text{disk}}}{PS}.$$

Therefore, the memory footprint cost of a partition P_i that fulfills SLA is

$$\mathcal{M}\left(|P_i|,\ \widehat{N}_i^{\text{col}},\ SLA,\ \beta\right)=\begin{cases}\mathcal{M}_{\text{hot}}(|P_i|), & \text{if } SLA/\widehat{N}_i^{\text{col}}\leqslant\beta,\\ \mathcal{M}_{\text{cold}}\left(|P_i|,\ \widehat{N}_i^{\text{col}},\ SLA\right), & \text{otherwise.}\end{cases}$$

Finally, we discuss the partition update costs, which typically consider cost savings of new partitions and data migration expenses. They directly determine whether the repartition scheme is executed. The cost savings arise from lower transaction execution and resource costs, whereas data migration expenses cover the overheads tied to partition and replica modifications.

3.6 Summary

We summarize key characteristics of HP as follows: 1) It is crucial to survey the storage and deploy-

ment environments before designing partitions. 2) When query features are scarce, query-driven methods often utilize data features as a supplement to build finer-grained or size-constrained partitions. 3) Each partitioning strategy has unique strengths and weaknesses. Mathematical programming requires feasibility verification due to partitioning's NP-hard nature. Learning-based algorithms exhibit high performance but adapt poorly to environmental changes. Conversely, greedy algorithms offer more flexibility for existing partitioning constraints, but may lack stable performance, which could be improved with additional optimization phases.

4 Vertical Partitioning

Subsection 4.1 and Subsection 4.2 provide the definition and feature extraction of the vertical partitioning (VP) problem, respectively. Mainstream VP construction strategies for centralized and distributed environments are presented in Subsections 4.3 and 4.4, respectively, and their cost models are introduced in Subsection 4.5. Fig.5 depicts the development trajectory of VP methods.

4.1 Formalization

Definition 3 (Static Vertical Partitioning). *Static VP is a two-phase partitioning technique for processing the collected queries Q. A table data D is initially divided vertically into disjoint column groups CGs, which are subsequently split horizontally into k distinct partitions $\mathbb{P} = (P_1, P_2, \ldots, P_k)$ through two candidate strategies:* 1) *all CGs are split into $\mathbb{P}$ as a whole containing aligned tuples;* 2) *each CG is independently split into partitions and then merged into $\mathbb{P}$. The objective of VP is to generate the optimal combination of CGs and $\mathbb{P}$ that minimizes the final processing cost C of Q. VP first identifies optimal column groups (CGs^*) by introducing an additional cost function C_{cg} to evaluate only the division of each CG, and then finds optimal classifier (ϕ^*) to generate $\mathbb{P}$.*

$$\begin{aligned} CGs^* &= \arg\min_{CGs} \sum_{CG \in CGs} C_{\text{cg}}(CG,\ Q), \\ P_i &= \phi(e,\ CGs^*),\ \forall e \in D, \\ \phi^* &= \arg\min_{\phi} C\big(\mathbb{P} \Leftarrow \phi(D,\ CGs^*),\ Q\big). \end{aligned} \tag{1}$$

Definition 4 (Dynamic Vertical Partitioning). *This concept exhibits parallel characteristics to dynamic HP and will not be defined repeatedly here.*

4.2 Feature Extraction

Database Schema. 1) VP is essentially an extension of table splitting, and high-frequency column groups can be directly extracted from independent business scenarios in advance. 2) Distinguishing between indexed and non-indexed columns[52, 64, 72] considerably affects column grouping. 3) Small/large tables are categorized based on the number of attributes to verify algorithms' execution efficiency.

Table Data. When constructing column groups, examining attribute types, such as primary/constrained keys, can help reduce join costs. When creating range-based horizontal splits, the attribute distribution characteristic[73] serves as a crucial reference factor in determining partition keys.

Workload. Query features[50, 53, 60, 65] such as accessing attributes (projection, filter, and join columns), affected rows, selectivity, SQL keywords[57, 69, 73], and submission time are commonly extracted in VP. Here, accessing attributes are used to calculate co-occurrence frequency between attributes; selectivity[72, 73] reflects the proportion of scanned tuples in the total table tuples, with higher selectivity typically indicates a greater query weight.

Database Runtime Metric. Similar to HP, the VP layout primarily focuses on key metrics like system throughput, processor stalls, and resource utilization.

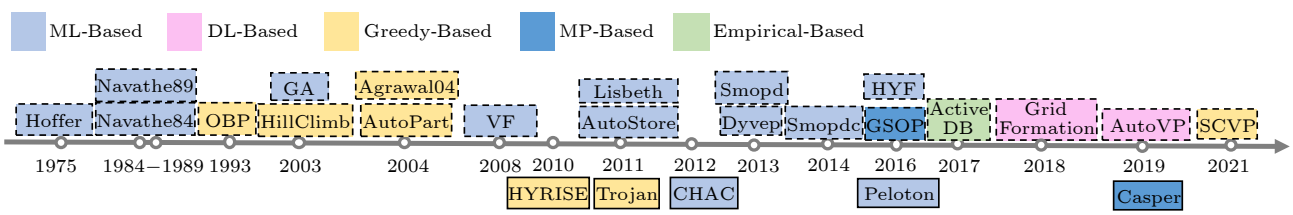

Fig.5. Timeline of VP research development, including on-axis studies (Hoffer[49], Navathe84[50], Navathe89[51], OBP[52], GA[53], HillClimb[54], AutoPart[55], Agrawal04[56], VF[57], Lisbeth[58], AutoStore[59], Smopd[60], Dyvep[61], Smopdc[62], GSOP[63], HYF[64], ActiveDB[65], GridFormation[66], AutoVP[67], and SCVP[68]) based on centralized environments and off-axis studies (HYRISE[69], Trojan[70], CHAC[71], Peloton[72], and Casper[73]) based on distributed environments.

4.3 Partitioning Process in Centralized Databases

Recent research[66, 67] has highlighted the benefits of applying reinforcement learning algorithms for dynamic partition updates. This marks a major evolution from earlier methods[49–51, 53, 55–58, 64], which depended on the partitioning feature of attribute affinity. These methods aimed to enhance performance but often at the cost of increased execution time. However, as the field progressed, there was a shift towards more efficient, lightweight, end-to-end partitioning methods[59–63, 65, 68], reflecting a continuous effort to balance system design's efficiency and effectiveness.

ML-Based. Hoffer *et al.*[49] calculated an attribute affinity matrix (AAM) from column accesses and applied the BEA[74] algorithm to cluster the AAM into column groups (CGs). Navathe *et al.*[50] refined this approach by introducing a two-phase partitioning with a cost model to select appropriate cost types for the given environment. Initially, they used Hoffer's method to obtain CGs, followed by a binary partitioning to recursively split each candidate CG into two finer-grained ones. Navathe *et al.*[51] reduced the time complexity of previous work[50]. They created an undirected weighted graph with nodes representing table attributes and edge weights denoting attribute affinity, and identified minimal-weight closed "loops" as the final CGs. Ng *et al.*[53] designed a cost function focusing solely on the transaction's page access and the penalty cost for cross-partition transactions. A GA model was trained to find optimal CGs and tuple clusters within each CG.

Greedy-Based. OBP[52] (Optimal Binary Partitioning) treats transaction attributes as basic units for building a binary search tree. Each leaf node is expanded by assigning attributes referenced from the selected unit to its left branch and remaining attributes to its right. AutoPart[55] identifies discrete condition values in queries as horizontal splits to create atomic partitions. Each atomic partition contains all attributes referenced by its access queries, with no query accessing merely a subset of its attributes. To reduce join overhead, AutoPart merges atomic partitions and adds redundant attributes to form composite partitions. Agrawal *et al.*[56] considered partition manageability, introducing an interestingness score for CG effectiveness and employing greedy approaches to generate CG candidates. They designed algorithms MergeColumns and MergeRanges to identify the optimal CG solution meeting partition aligned constraints. PAX[75] (Partition Attributes Across) layout decomposes relations at the page level to avoid the join expenses of prior VP studies[49–53] that break down a table into multiple subtables. HillClimb[54] extends PAX by defining a finer page layout. Starting with PAX's single-column partitions, it merges the two partitions offering the largest query cost reduction in iterative rounds until no further reduction is possible. Fig.6 illustrates this process. The CGs $|a_1|a_2|a_3|a_4|a_5|a_6|$ are the initial page layout. The first round merges a_1 and a_2 for their greatest merging benefit. Then we update merging benefits of valid candidate mergers, and a_5 and a_6 are next merged. The process continues until reaching the optimal state $|a_1a_2a_3|a_4a_5a_6|$ with no feasible mergers left.

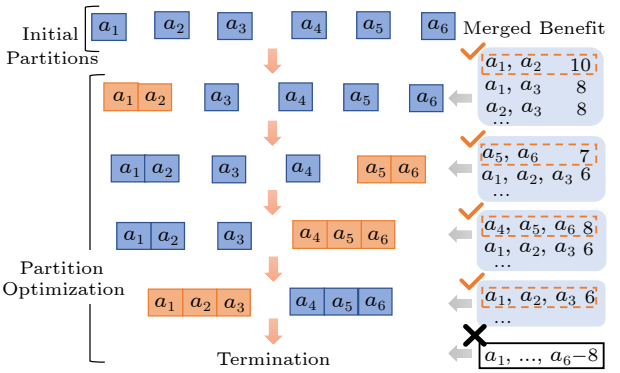

Fig.6. Exemplary case of the HillClimb algorithm[54] for column group creation.

MP-Based. Sun *et al.*[63] generalized the SOP[9] model (GSOP) by introducing column grouping (vertical splits) before local feature selection (horizontal splits). To obtain CGs, GSOP constructs an ILP equation to balance block skipping and tuple reconstruction.

ML-Based. Several studies[57, 58, 64, 68] leveraged frequent itemset mining algorithms to identify key column groups from loads. Gorla *et al.*[57] introduced a vertical fragmentation (VF) method that selects the top-k non-overlapping Apriori[36]-generated patterns for forming complete CGs, along with a cost function for assessing the partition scanning and concatenation overheads in transaction operations. Lisbeth *et al.*[58] proposed a VP technique that sets the minimum support threshold automatically, while VF requires manual specification. HYF[64] assigns new weights to frequent patterns by multiplying their support value with the cosine similarity of all patterns. It generates multiple candidate complete schemes like VF and designs a cost function considering sequence/

index scans for optimal selection. SCVP[68] improves HYF's cost model by incorporating tuple reconstruction costs. Leveraging the cost independence property between CGs, SCVP first designs an estimation function for rapid CG division gain calculation, making it suitable for large tables and heavy loads. It then applies spectral clustering on AAM to form initial CGs and adopts a greedy search strategy to split and merge CGs based on frequent patterns.

Constructing a self-adaptive VP layout is crucial. AutoStore[59] introduces O2P (One-dimensional Online Partitioning) to monitor query changes through a query window and updates AAM online. O2P uses the BEA algorithm to recluster only the CGs referenced by new queries, and designs a transforming benefit model to decide whether the repartition decision should be executed. SMOPD[60] improves on AutoStore by determining appropriate checkpoint intervals for repartitioning based on historical data analysis, employing AutoClust[76] for partition updates. SMOPD-C[62] further adapts SMOPD to distributed settings by updating monitoring procedures. To solve cold start issues of VP, DYVEP[61] designs a statistic collector to monitor the changes in query patterns and database schema, creating new partitions or triggering repartitioning when query latency increases or table attributes are deleted.

Empirical-Based. ActiveDB[65] uses 21 active rules to monitor both internal and external system and user activities. The first 15 rules gather query-related statistical indicator changes. Two rules estimate the current performance change to determine the necessity for partition updates. The final four rules use statistical features to create new partitions and access their performance improvement threshold.

DL-Based. GridFormation[66] is the first learning-based agent using Q-learning[47] for online VP layout design. The state is defined as a collection of sets, each indicating a partition containing a list of tuple IDs. GridFormation's partitioning process follows a Markov decision process (MDP), with rewards calculated based on touched partitions and tuple access ratio of each query. AutoVP[67] redesigns the GridFormation agent to accelerate training, offering three optional DQN variants[77] and using HillClimb[54] and HDD[78] to evaluate temporary partitions. It simplifies state representation to a 2D array, with each row corresponding to a query and each column to a table attribute. Rewards are based on the cost difference between the current state and HillClimb's ideal state, enabling faster experience learning and MDP process.

Table 12 summarizes vertical partitioning techniques for centralized environments.

4.4 Partitioning Process in Distributed Databases

In big data systems, VP layouts are commonly built on page-level stores like [75]. Trojan[70] defines an interestingness score to reflect how effectively the CG accelerates most queries, then solves a 0-1 knapsack problem to select the optimal CG combinations. Trojan achieves layout-aware replication by designing unique CGs for each replica, better adapting to given queries. CHAC[71] (Column-oriented Hadoop Attribute) extracts frequent closed item sets from a frequency-weighted AAM to generate overlapping and non-overlapping candidate clustering solutions, and designs a cost model to select the optimal solution.

VP-based hybrid storage is customized for HTAP databases. HYRISE[69] measures cache misses resulting from data movement from RAM to cache, mitigating cache pollution in update operations using non-temporal writes. It creates CG layouts that adapt to cache lines to accelerate read operations. Peloton[72] clusters queries by their co-accessed attributes via the k-means algorithm, selecting representative queries for each cluster by optimizer estimates and submission time. It then prioritizes these queries, using a greedy policy to extract CGs and maintains recent query statistics in a time series graph to periodically replace old CGs. Casper[73], a column layout that works with VP algorithms like HYRISE and Peloton, optimizes HTAP load processing in in-memory DBMS. It estimates block read/write I/O costs for various transaction operations, aligns block sizes with cache lines, and track each operation access via block domain histograms. This helps establish ILP equations to allocate data while satisfying constraints related to read/update latencies.

Table 13 summarizes vertical partitioning techniques for distributed environments.

4.5 Cost Estimation for Vertical Partition Scheme

This subsection reviews common function-based cost models (see Table 14) employed for VP evaluation, including two-phase partitioning and partition updates. They consider query execution on VP layouts, partition updates, and the impact of indexes,

Table 12. Major Vertical Partitioning Strategies for Centralized Environments

Category	Work	Baseline	Objective	Automatic	Cost	Deployment	Method Content
ML	Navathe[50]	Hoffer *et al.*[49]	O3	✘	✔	✘	BEA$^{\triangledown}$; binary partitioning$^{\mathrm{O}}$
ML	DYVEP[61]	AutoPart[55]	O4	M-TH	✘	✔	Query table$^{\circlearrowleft}$+Navathe84$^{\triangledown\mathrm{O}}$
ML	SMOPD(-C)[60, 62]	Static VP using AutoClust[76]	O2, O4, O8	M-QW	✘	✘	Statistics monitoring$^{\circlearrowleft}$+ filtered AutoClust$^{\triangledown}$
ML	VF[57]	Unpartitioned state	O3	✘	✔	✘	Apriori$^{\triangledown}$; greedy search$^{\mathrm{O}}$
ML	HYF[64]	Lisbeth *et al.*[58], VF[57]	O3	✘	✔	✘	Aprior with cosine similarity$^{\triangledown}$; greedy search$^{\mathrm{O}}$
ML	SCVP[68]	HillClimb[54], HYF[64]	O3, O8	✘	✔	✘	Spectrum clustering$^{\triangledown}$, greedy splitting/merging$^{\mathrm{O}}$
Greedy	OBP[52]	Navathe *et al.*[51]	O3	✘	✔	✘	Binary search tree using transaction-based splits$^{\triangledown}$
Greedy	HillClimb[54]	Navathe *et al.*[50, 51]	O10	✘	✔	✔	HillClimb algorithm$^{\triangledown}$
Greedy	AutoPart[55]	Navathe *et al.*[50]	O3	✘	✘	✘	Composite partitions$^{\triangledown}$; pair-wise merging$^{\mathrm{O}}$
Greedy	Agrawal[56]	Navathe *et al.*[51]	O12	✘	✔	✘	Interesting column groups$^{\triangledown}$ greedy selection/merging$^{\mathrm{O}}$
Greedy	AutoStore[59]	HillClimb[54]	O3, O8	M-QW	✔	✔	O2P+query window$^{\circlearrowleft}$
MP	GSOP[63]	SOP[9], HillClimb[54]	O3, O_7	✘	✔	✔	ILP$^{\triangledown}$; Apriori+Ward$^{\mathrm{O}}$
DL	GridFormation[66]	Manual partitions	O3	M-RL	✘	✘	Q-learning algorithm$^{\triangledown\circlearrowleft}$
DL	AutoVP[67]	AutoPart[55], O2P[59]	O3	✘	✘	✘	DQN and its variants$^{\triangledown}$

Table 13. Major Vertical Partitioning Strategies for Distributed Environments

Category	Work	Baseline	Objective	Automatic	Cost	Deployment	Method Content
Greedy	HYRISE[69]	Simple partitions	O10	✘	✔	✔	Kmetis$^{\triangledown}$; greedy merging$^{\mathrm{O}}$
Greedy	CHAC[71]	Hoffer *et al.*[49]	O3	✘	✔	✘	AAM+frequent closed items$^{\triangledown}$
ML	Peloton[72]	Simple row/column partitioning	O3	M-TH+D-RE	✘	✔	k-means$^{\triangledown}$; greedy selection$^{\mathrm{O}}$
MP	Trojan[70]	Hadoop-row, Hadoop-PAX, HYRISE[69]	O3, O8	✘	✔	✘	Interestingness grouping$^{\triangledown}$; 0-1 knapsack programming$^{\mathrm{O}}$
MP	Casper[73]	DSM+leading columns, DSM+equi-width partitions	O3	✘	✔	✔	Equ-size partitions+ histogram-based frequency model+ILP$^{\triangledown}$

Table 14. Comparative Analysis of Function-Based Vertical Partitioning Cost Models in Diverse Environments

Cost Model	Objective	Environment	Characteristic
VF[57], GSOP[63]	O3	E-CH/S	Incorporating the cost of tuple construct across partitions
ACO[78]	O3	E-CH/S	Cost designs for bandwidth-based disk access operations
AutoPart[55], HYF[64]	O3	E-CH/S	Approximating the costs of index scans and block joins
AutoStore[59]	O3, O5	E-CH/S	Considering the repartitioning potential benefit
CHAC[71], Trojan[70]	O3, O5	E-DH/S	Finer cost estimations for map-reduce phases
DataMorhing[54], HYRISE[69]	O10	E-DM	Estimating cache misses for diverse data access operations
Casper[73]	O3	E-DM	Modeling costs for five distinct data access operations

joins, and map-reduce operations. For non-PAX VP techniques, the additional tuple reconstruction cost for cross-partition queries is another crucial factor.

4.5.1 *Centralized Environment*

To determine when to repartition, some studies[59] use a fixed query window, while [60–62] employ dynamic windows based on query performance thresholds. The choice of monitoring approach does not impact the modeling of repartitioning benefits. However, AutoStore[59] differs from other approaches by considering potential benefits of new partitions rather than solely evaluating them based on historical loads. It introduces a transformation benefit B_{tf}, resulting from

updating current partitions $\mathbb{P}$ to a new scheme $\mathbb{P}'$ when executing n queries Q collected in the window. B_{tf} is calculated as $C_{\text{cg}}(Q,\mathbb{P}) - C_{\text{cg}}(Q,\mathbb{P}')$, with C_{cg} denoting the query processing cost over a given vertical layout. AutoStore assumes the presence of multiple future windows with workloads similar to the current window, and estimates their frequency using an exponential decaying model with a shape parameter γ, i.e., $freq = 1/(1-\gamma^{-n})$. The potential benefit of updating current partitions is then calculated as $B_{\text{r}} = freq \times B_{\text{tf}} - C_{\text{cg}}$, and new partitions are deployed only if $B_{\text{r}} > 0$.

Various approaches[57, 59, 63, 64, 78] calculate C_{cg} by breaking down the total query cost into scan and tuple reconstruction costs of multiple accessed CGs. The scan cost counts the number of both random and sequential I/O blocks, with random I/O accounting for unclustered and clustered index scan costs plus index lookup costs. The tuple reconstruction cost considers only the join cost (e.g., hash and sort-merge joins) if tuples between different CGs are not aligned; otherwise, a minimal tuple addressing cost is considered.

4.5.2 *Distributed Environment*

VP layout is prevalent in distributed Hadoop environments. For example, both Trojan[70] and CHAC[71] consider the impact of column groups during the map phase as the main cost factor. However, unlike Trojan, CHAC estimates costs roughly, focusing solely on data access volume and omitting disk read/write characteristics and network cost considerations. We will introduce the Trojan cost model next.

To avoid tuple reconstruction, Trojan is based on PAX and considers data reading and network costs. The known parameters include the block size S_{b}, number of machines n, map tasks m, and the split size S_{split}. S_{split} determines the number of data slices, with each being handled by a single mapper. When processing a query q, the number of blocks read is denoted as N_{b}, and then the number of map phases is calculated as $N_{\text{map}} = N_{\text{b}} \times S_{\text{b}}/(S_{\text{split}} \times m \times n)$.

The read cost for each map phase includes both random I/O, $C_{\text{rand}}(q) = F_{\text{rand}} \times (S_{\text{split}} \times |C'_{\text{cg}}|/(S_{\text{buffer}} \times |C_{\text{cg}}|))$, and sequential I/O, $C_{\text{seq}}(q) = S_{\text{split}} \times |C'_{\text{cg}}|/(\text{BW}_{\text{disk}} \times |C_{\text{cg}}|)$. F_{rand} denotes the average random seek time (0.005 s); S_{split} is set to 256 MB; C_{cg} and C'_{cg} represent the complete and accessed column sets, respectively; S_{buffer} is the buffer size (512 KB), and BW_{disk} is the average disk bandwidth (100 MB/s). When local data is not available, the network cost C_{tr} arises from transferring data from one machine to another, i.e., $C_{\text{tr}} = (1-p_{\text{tr}}) \times (S_{\text{split}}/BW_{\text{net}})$. BW_{net} denotes the network bandwidth (1 GB/s) and p_{tr} is the occurrence probability (0.97) of remote accesses. Assuming a map initialization time of 0.1 s (C_{init}), the total latency of query q over the Trojan layout is computed as $(C_{\text{tr}}(q) + C_{\text{rand}}(q) + C_{\text{seq}}(q) + C_{\text{init}}) \times N_{\text{map}}$.

4.6 Summary

Differing from HP, VP involves a two-phase process of column grouping and horizontal division of tuples, with each phase being NP-hard. In the first phase, mathematical programming algorithms efficiently identify CGs in small tables, while greedy and ML-based algorithms are preferred for large tables. In the second phase, partitions within each CG are typically generated using hash or range values of keys. Additionally, cost models play a crucial role in the VP process, calculating scan costs for CGs and cross-CG reconstruction costs for selecting candidate partitions. Despite its advantages, deploying and evaluating VP in real-world databases is challenging due to the limited native support for VP creation.

5 Irregular Partitioning

Irregular partitioning (IP) is a cutting-edge technique for handling analytical and mixed loads. However, deploying it poses challenges such as maintaining storage structure, updating partitions, and coordinating query executors. Furthermore, there is a scarcity of relevant studies according to [8, 81]. In this section, we define the IP problem in Subsection 5.1. The partitioning features required by IP discussed in Section 3 and Section 4, will not be reintroduced. Subsequently, we describe several classic IP techniques in Subsection 5.2 and provide a summary in Subsection 5.3. Fig.7 depicts a simple development trajectory of IP methods.

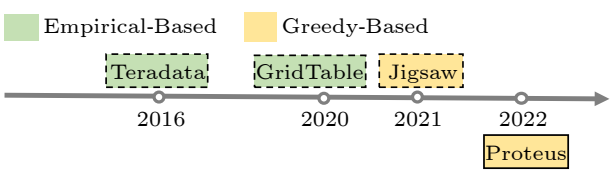

Fig.7. Timeline of IP research development, including on-axis studies (Teradata[79], GridTable[80], and Jigsaw[81]) based on centralized environments and off-axis studies (Proteus[82]) based on distributed environments.

5.1 Formalization

Definition 5 (Static Irregular Partitioning). *The IP problem uses a predefined classifier $\phi(\cdot)$ to generate a set of partitions with arbitrary shapes for a given table, denoted as $\mathbb{P} = (P_1, P_2, \ldots, P_k)$. The table data consists of m tuples $D = (e_1, e_2, \ldots, e_m)$, where each tuple e has n attributes. The y-th attribute value of the x-th tuple e_x in T is represented as $e_{x,y}$. The objective of IP is to assign specific irregular partition for each $e_{x,y}$ in order to minimize the I/O cost of processing load Q.*

$$\phi^* = \arg\min_{\phi} C\big(\mathbb{P} \Leftarrow \phi(D),\ Q\big),$$
$$P_i = \phi(e_{x,y}),\ \forall e_x \in D \ \text{ and } \ y = 1, \ldots, n.$$

Definition 6 (Dynamic Irregular Partitioning). *This concept adheres to the identical update mechanism employed by both the HP and VP algorithms, and will not be detailed further in this context.*

5.2 Partitioning Process

Empirical-Based. Teradata[79] introduces a hybrid row-column layout via multi-level definitions, with the first level for column partitions and subsequent levels for further row partitions. Teradata uses a partition number combining row IDs and partition levels to make file systems identify arbitrary partition shapes, reducing partition scanning and optimizing DML operations. GridTable[80] extends VP layouts like HYRISE and Peloton with a flexible grid layout. Each grid is self-contained, organizing tuples column-wise or row-wise independently. It supports tuple-centric read/write operations and efficient range query executions. However, neither study provides any partition creation guidance.

Greedy-Based. Jigsaw[81] provides ultimate data skipping for static queries with tetris-shaped partitions managed by logical segments. These segments are classified as active or frozen based on if they can be split for I/O reduction. Jigsaw first partitions tables into frozen segments using a split function, then merges or splits these segments for adapting block size. Jigsaw layout requires a hash table for tuple reconstruction and manages partitions effectively by materializing logical segments into rectangular physical ones storing tuples with the same columns. Proteus[82] adaptively designs and updates storage layouts for different table areas, aligning partition shapes with queried data areas and selecting appropriate row or column formats for specific transactions to support efficient data reads and updates. When storage limits are reached or performance issues arise, Proteus optimizes storage formats and reorganizes hot partitions and replicas. To evaluate the potential benefits of data migration plans, Proteus uses recurrent neural networks (RNNs) and linear predictors to predict future data access patterns.

Fig.8 displays four distinct layouts, with the right arrow (→) indicating row-wise storage and the down arrow (↓) indicating column-wise storage. 1) Teradata vertically splits the table into three column partitions (P_{1-2}, P_{3-4}, P_{5-6}), each further hierarchically partitioned by rows; 2) GridTable breaks the table into six grids of variable rows and columns, each supporting either row-major or column-major storage; 3) Jigsaw supports arbitrary partition shapes, similar to Tetris; 4) Proteus generates column partitions (P_1, P_2) and row partitions (P_3, P_4) without hierarchical order. It can be observed that Teradata and Jigsaw layouts prefer row-oriented storage, while GridTable and Proteus layouts exhibit flexibility in their storage formats.

Table 15 summarizes the characteristics of the four irregular partitioning techniques.

5.3 Summary

The IP field aims to enhance mixed and analytical loads by maximizing the optimization potential of query-driven strategies. It fully utilizes current query distributions to create complex partitioning rules to satisfy diverse query access patterns. Despite its advantages, IP still faces several challenges, such as the need to develop a unified transaction execution interface and handling the management complexities asso-

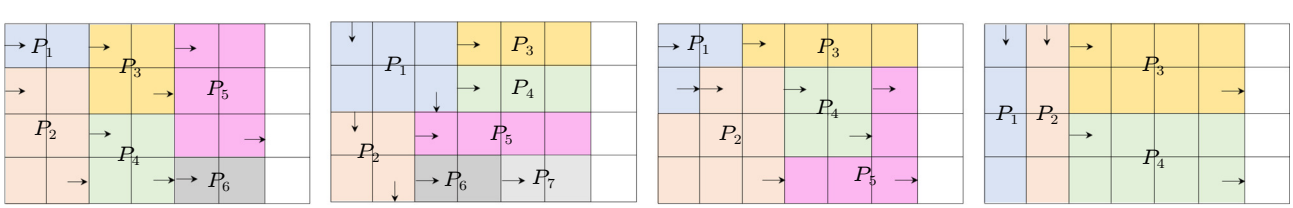

Fig.8. Comparison of four IP designs. (a) Teradata layout[79]. (b) GridTable layout[80]. (c) Jigsaw layout[81]. (d) Proteus layout[82].

Table 15. Summary of Irregular Partitioning Strategies in Centralized and Distributed Environments

Category	Work	Baseline	Objective	Automatic	Cost Composition	Content
Empirical	Teradata[79]	Simple range partitions	O3, O10	✘	Optimizer-based I/O costs; CPU metrics	Rowid-based storage+ multi-level range partitioning$^{\triangledown}$
Empirical	GridTable[80]	N/A	O4	✘	Access and transition costs between grids	Three level-specific data manipulation operations
Greedy	Jigsaw[81]	Schism, Schism+Peloton	O3	✘	Read I/Os of layouts; memory for hash tables	Segment partitioning$^{\triangledown}$; greedy merging$^{\mathrm{O}}$
Greedy	Proteus[82]	TiDB	O4	M-TH+D-RE	Costs for layout-aware/ -agnostic storages	Layout creation rules$^{\triangledown}$; hybird predictors for queries$^{\circlearrowleft}$

ciated with irregular partition replication, maintenance, and joins.

6 Data Partitioning in Industry

Table 16 compares popular database products and their partitioning support. Most DBMSs, e.g., Redshift③, Firebolt④, Databricks⑤, GaussDB⑥, TiDB⑦, OceanBase⑧, and SingleStore⑨, offer user-defined HP strategies such as range, hash, key, list, and round-robin, where partition keys necessitate manual selection and updates. These systems prioritize balanced resource utilization among nodes and cluster scalability/parallelism through partitioning, rather than focusing solely on maximizing system performance. Besides, their simplicity enables DBAs to effortlessly create and manage partitions. Organizing data based on data distribution can also makes it easier to conduct data analysis, particularly for time-series data.

In contrast, certain products (e.g., Vertica⑩, Greenplum⑪, and VoltDB⑫) incorporate load analysis into their partitioning design. VoltDB is an in-memory DMBS for fast data processing tasks like online gaming and IoT sensors. By analyzing historical load and data distribution, it scales transaction processing capacity, creating optimal range partitions. This ensures load balancing and allows high-frequency transactions to be executed locally.

Some products, e.g., ClickHouse⑬, StarRocks⑭, Apache Hudi⑮, Oracle Autonomous Database⑯, and Snowflake⑰, provide automated partition key selec-

Table 16. Partitioning Support Comparison of Popular Database Products for OLAP, OLTP, and HTAP Scenarios

Scenario	Type	Partitioning Strategy	Strategy Type	Automatic	Representative Product
OLAP	HP	Key, hash, range, list, round-robin	Data-driven	✘	Redshift③, Firebolt④, Databricks⑤, GaussDB⑥
	HP	Round-robin, list, hash, range	Data-driven	M-TH	ClickHouse⑬, StarRocks⑭, Apache Hudi⑮
	HP	Automatic interval/list	Data-/query-driven	M-SD	Oracle Autonomous Database⑯
	HP	Auto clustering	Data-driven	M-TH	Snowflake⑰
	HP&VP	Range, table projections+hash	Data-/query-driven	✘	Vertica⑩, Greenplum⑪
OLTP	HP	Key, hash, range, list	Data-driven	✘	PostgreSQL, MySQL, Oracle, SQLServer
	HP	Key, hash, range, list	Data-/query-driven	✘	VoltDB⑫
	VP	Sharding+table views	Data-/query-driven	✘	PostgreSQL, MySQL, Oracle, SQLServer
HTAP	HP	Key, hash, range, list	Data-driven	✘	TiDB⑦, OceanBase⑧
	HP	Hash	Data-driven	✘	SingleStoreDB⑨

③https://docs.aws.amazon.com/redshift/latest/dg/c-spectrum-external-tables.html, Mar. 2024.
④https://docs.firebolt.io/working-with-partitions.html, Mar. 2024.
⑤https://docs.databricks.com/sql/language-manual/sql-ref-partition.html, Mar. 2024.
⑥https://support.huaweicloud.com/intl/en-us/twp-dws/dws_11_0013.html, Mar. 2024.
⑦https://docs.pingcap.com/zh/tidb/v7.0/partitioned-table, Mar. 2024.
⑧https://en.oceanbase.com/docs/common-oceanbase-database-10000000001105730, Mar. 2024.
⑨https://docs.singlestore.com/cloud/create-a-database/choosing-a-table-storage-type, Mar. 2024.
⑩https://docs.vertica.com/12.0.x/en/admin/projections, Mar. 2024.
⑪https://docs.vmware.com/en/VMware-Greenplum/7/greenplum-database/admin_guide-ddl-ddl-partition.html, Mar. 2024.
⑫https://docs.voltactivedata.com/UsingVoltDB/DesignPartition.php, Mar. 2024.
⑬https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/custom-partitioning-key, Mar. 2024.
⑭https://docs.starrocks.io/docs/table_design/dynamic_partitioning, Mar. 2024.
⑮https://hudi.apache.org/docs/file_layouts, Mar. 2024.
⑯https://docs.oracle.com/en/cloud/paas/autonomous-database, Mar. 2024.
⑰https://docs.snowflake.com/en/user-guide/tables-auto-reclustering, Mar. 2024.

tion and updates. The Oracle Autonomous Database service is a resource-intensive and time-consuming operation, invoked on-demand rather than running periodically. Analyzing workload information, it automatically identifies candidate partitioning tables and recommends partitions for optimal I/O reduction using strategies: automatic interval, automatic list, and hash. Snowflake creates micro-partitions via ZoneMap indexes and column distribution histograms. Data is organized in a natural order (unclustered state), then clustered by selected keys to prevent cluster key value duplication across partitions (clustered state). As new data arrives, the number of duplicate key values across different partitions increases; each partition's depth is quantified by the count of its overlapping partitions. To preserve overall data order, Snowflake prioritizes selecting micro-partitions with higher depths and sorts and merges them independently.

Vertica and Greenplum are among the few products natively supporting VP creation for efficient partition pruning. This is achieved by creating local column group projections on disk for partitioned tables and evenly distributing projected data to partitions via hashing. By storing related data together, Vertica can more efficiently utilize system resources. Fine-grained projection replicas make it easier to achieve high availability and data recovery. Conversely, other products simulate VP by combining sharding and views, an approach complicating table schema, subtable data consistency, and query planning, which may lead to performance issues like overloaded shards and increased partition maintenance costs.

7 Open Problems

In this section, we explore remaining challenges and potential solutions in the current data partitioning community.

Partitioning for Non-Numeric Columns. Query access patterns pertaining to non-numeric columns are often ignored, which greatly limits the optimization space of partitioning. A feasible solution to this dilemma involves transforming non-numeric column data into numeric data via data encoding. Date columns can be transformed into numeric values through timestamp functions, while enumeration columns are dictionary-encoded based on their semantic or alphabetical order. For more complex column values, a trie-based index tree[83] can be built, with a depth-first traversal to derive encoding keys.

Block Allocation Within VP. Current research adopts simple data-driven methods to allocate tuples into blocks after obtaining column groups (CGs). Although [53, 56] have considered load information, they still encounter convergence or performance issues. This inefficiency prevents the VP algorithm from achieving its optimal potential, even when the CG division is aligned with column access patterns. A promising solution is to incorporate proven effective query-driven HP algorithms like QdTree[14] into VP.

Reliability of Partition Updating. Monitoring services frequently rely on recently collected query logs to design new partitions; however, this method neglects the similarity between future and historical loads, making it challenging to estimate updated partitions' potential performance. While [34, 59, 82] have tried to model special scenarios to calculate future benefits of new partitions, these assumptions often prove unrealistic. This issue presents significant optimization potential in two aspects: firstly, improving the prediction accuracy of future load for generating better new partitions; and secondly, reducing the number of problem assumptions.

Deep Learning Models for Cost Estimation. To the best of our knowledge, no public, network-centric cost model exists for partitioning. However, the learning and generalization capabilities of deep neural networks render them particularly suitable for such tasks. The main challenge lies in collecting sufficient training samples due to the high partition deployment cost and the vast partition solution space. A viable solution entails compressing or trimming the solution space by identifying factors influencing the query plan. This could be achieved by using a pruned branch bounding tree for candidate partitions and removing the deployment and metric measurements of cold data. Subsequently, query plans and execution metrics for various partitions are collected to train an RNN-stacked tree network.

8 Conclusions

In this paper, we modularized the partitioning technique, emphasizing the significance of cluster and storage environments in formulating an efficient partitioning path. Our approach enhances the tracking of partitioning progress and clarifies the considerations necessary at each partitioning stage, ensuring optimal designs. Before partitioning, it is crucial to align cost models and partition types with specific environmental characteristics. Furthermore, the intricate re-

lationship between data migration plans during partition updates and cluster configuration underscores the importance of a holistic approach. We also classified partition generation strategies based on algorithm types, distinguishing key features such as model convergence and partition quality to aid in strategy selection. For future research, we would like to explore feasible solutions for addressing existing key challenges including non-numeric column-based partitioning and the reliability of partition updating. We hope our framework and findings could contribute to the advancement of partitioning systems and provide practical insights for DBAs in various environments.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Melnik S, Gubarev A, Long J J *et al*. Dremel: A decade of interactive SQL analysis at web scale. *Proceedings of the VLDB Endowment*, 2020, 13(12): 3461–3472. DOI: 10.14778/3415478.3415568.

[2] Bayer R, McCreight E. Organization and maintenance of large ordered indices. In *Proc. the 1970 ACM SIGFIDET* (*Now SIGMOD*) *Workshop on Data Description, Access and Control*, Nov. 1970, pp.107–141. DOI: 10.1145/1734663.1734671.

[3] Bentley J L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 1975, 18(9): 509–517. DOI: 10.1145/361002.361007.

[4] Guttman A. R-trees: A dynamic index structure for spatial searching. In *Proc. the 1984 ACM SIGMOD International Conference on Management of Data*, Jun. 1984, pp.47–57. DOI: 10.1145/602259.602266.

[5] Yuan H T, Li G L, Feng L, Sun J, Han Y. Automatic view generation with deep learning and reinforcement learning. In *Proc. the 36th IEEE International Conference on Data Engineering*, Apr. 2020, pp.1501–1512. DOI: 10.1109/ICDE48307.2020.00133.

[6] Han Y, Li G L, Yuan H T, Sun J. An autonomous materialized view management system with deep reinforcement learning. In *Proc. the 37th IEEE International Conference on Data Engineering*, Apr. 2021, pp.2159–2164. DOI: 10.1109/ICDE51399.2021.00217.

[7] Zhang H, Chen G, Ooi B C, Tan K L, Zhang M H. In-memory big data management and processing: A survey. *IEEE Trans. Knowledge and Data Engineering*, 2015, 27(7): 1920–1948. DOI: 10.1109/TKDE.2015.2427795.

[8] Mahmud M S, Huang J Z, Salloum S *et al*. A survey of data partitioning and sampling methods to support big data analysis. *Big Data Mining and Analytics*, 2020, 3(2): 85–101. DOI: 10.26599/BDMA.2019.9020015.

[9] Sun L W, Franklin M J, Krishnan S, Xin R S. Fine-grained partitioning for aggressive data skipping. In *Proc. the 2014 ACM SIGMOD International Conference on Management of Data*, Jun. 2014, pp.1115–1126. DOI: 10.1145/2588555.2610515.

[10] Aly A M, Mahmood A R, Hassan M S, Aref W G, Ouzzani M, Elmeleegy H, Qadah T. AQWA: Adaptive query workload aware partitioning of big spatial data. *Proceedings of the VLDB Endowment*, 2015, 8(13): 2062–2073. DOI: 10.14778/2831360.2831361.

[11] Aly A M, Elmeleegy H, Qi Y, Aref W. Kangaroo: Workload-aware processing of range data and range queries in Hadoop. In *Proc. the 9th ACM International Conference on Web Search and Data Mining*, Feb. 2016, pp.397–406. DOI: 10.1145/2835776.2835841.

[12] Shanbhag A, Jindal A, Madden S, Quiane J, Elmore A J. A robust partitioning scheme for ad-hoc query workloads. In *Proc. the 2017 Symposium on Cloud Computing*, Sept. 2017, pp.229–241. DOI: 10.1145/3127479.3131613.

[13] Lu Y, Shanbhag A, Jindal A, Madden S. AdaptDB: Adaptive partitioning for distributed joins. *Proceedings of the VLDB Endowment*, 2017, 10(5): 589–600. DOI: 10.14778/3055540.3055551.

[14] Yang Z H, Chandramouli B, Wang C *et al*. Qd-tree: Learning data layouts for big data analytics. In *Proc. the 2020 ACM SIGMOD International Conference on Management of Data*, Jun. 2020, pp.193–208. DOI: 10.1145/3318464.3389770.

[15] Ding J L, Minhas U F, Chandramouli B *et al*. Instance-optimized data layouts for cloud analytics workloads. In *Proc. the 2021 International Conference on Management of Data*, Jun. 2021, pp.418–431. DOI: 10.1145/3448016.3457270.

[16] Li Z, Yiu M L, Chan T N. PAW: Data partitioning meets workload variance. In *Proc. the 38th IEEE International Conference on Data Engineering*, May 2022, pp.123–135. DOI: 10.1109/icde53745.2022.00014.

[17] Rao J, Zhang C, Megiddo N, Lohman G. Automating physical database design in a parallel database. In *Proc. the 2002 ACM SIGMOD International Conference on Management of Data*, Jun. 2002, pp.558–569. DOI: 10.1145/564691.564757.

[18] Agrawal S, Chu E, Narasayya V. Automatic physical design tuning: Workload as a sequence. In *Proc. the 2006 ACM SIGMOD International Conference on Management of Data*, Jun. 2006, pp.683–694. DOI: 10.1145/1142473.1142549.

[19] Eadon G, Chong E I, Shankar S, Raghavan A, Srinivasan J, Das S. Supporting table partitioning by reference in oracle. In *Proc. the 2008 ACM SIGMOD International Conference on Management of Data*, Jun. 2008, pp.1111–1122. DOI: 10.1145/1376616.1376727.

[20] Hauglid J O, Ryeng N H, Nørvåg K. DYFRAM: Dynamic fragmentation and replica management in distributed database systems. *Distributed and Parallel Databases*, 2010, 28(2): 157–185. DOI: 10.1007/s10619-010-7068-1.

[21] Curino C, Jones E, Zhang Y, Madden S. Schism: A workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment*, 2010, 3(1/2): 48–57. DOI: 10.14778/1920841.1920853.

[22] Nehme R, Bruno N. Automated partitioning design in parallel database systems. In *Proc. the 2011 ACM SIG-*

MOD International Conference on Management of Data, Jun. 2011, pp.1137–1148. DOI: 10.1145/1989323.1989444.

[23] Pavlo A, Curino C, Zdonik S. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In *Proc. the 2012 ACM SIGMOD International Conference on Management of Data*, May 2012, pp.61–72. DOI: 10.1145/2213836.2213844.

[24] Liroz-Gistau M, Akbarinia R, Pacitti E *et al*. Dynamic workload-based partitioning algorithms for continuously growing databases. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems XII*, Hameurlain A, Küng J, Wagner R (eds.), Springer, 2013, pp.105–128. DOI: 10.1007/978-3-642-45315-1_5.

[25] Quamar A, Kumar K A, Deshpande A. 2013. SWORD: Scalable workload-aware data placement for transactional workloads. In *Proc. the 16th International Conference on Extending Database Technology*, Mar. 2013, pp.430–441. DOI: 10.1145/2452376.2452427.

[26] Taft R, Mansour E, Serafini M, Duggan J, Elmore A J, Aboulnaga A, Pavlo A, Stonebraker M. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proceedings of the VLDB Endowment*, 2014, 8(3): 245–256. DOI: 10.14778/2735508.2735514.

[27] Chen K J, Zhou Y L, Cao Y. Online data partitioning in distributed database systems. In *Proc. the 18th International Conference on Extending Database Technology*, Mar. 2015, pp.1–12. DOI: 10.5441/002/edbt.2015.02.

[28] Zamanian E, Binnig C, Salama A. Locality-aware partitioning in parallel database systems. In *Proc. the 2015 ACM SIGMOD International Conference on Management of Data*, May 2015, pp.17–30. DOI: 10.1145/2723372.2723718.

[29] Fetai I, Murezzan D, Schuldt H. Workload-driven adaptive data partitioning and distribution—The Cumulus approach. In *Proc. the 2015 IEEE International Conference on Big Data*, Oct. 29–Nov. 1, 2015, pp.1688–1697. DOI: 10.1109/BigData.2015.7363940.

[30] Serafini M, Taft R, Elmore A J *et al*. Clay: Fine-grained adaptive partitioning for general database schemas. *Proceedings of the VLDB Endowment*, 2016, 10(4): 445–456. DOI: 10.14778/3025111.3025125.

[31] Marcus R, Papaemmanouil O, Semenova S, Garber S. NashDB: An end-to-end economic method for elastic database fragmentation, replication, and provisioning. In *Proc. the 2018 International Conference on Management of Data*, May 2018, pp.1253–1267. DOI: 10.1145/3183713.3196935.

[32] Nam Y M, Kim M S, Han D. A graph-based database partitioning method for parallel OLAP query processing. In *Proc. the 34th IEEE International Conference on Data Engineering*, Apr. 2018, pp.1025–1036. DOI: 10.1109/ICDE.2018.00096.

[33] Parchas P, Naamad Y, Van Bouwel P, Faloutsos C, Petropoulos M. Fast and effective distribution-key recommendation for amazon redshift. *Proceedings of the VLDB Endowment*, 2020, 13(12): 2411–2423. DOI: 10.14778/3407790.3407834.

[34] Hilprecht B, Binnig C, Röhm U. Learning a partitioning advisor for cloud databases. In *Proc. the 2020 ACM SIGMOD International Conference on Management of Data*, Jun. 2020, pp.143–157. DOI: 10.1145/3318464.3389704.

[35] Brendle M, Weber N, Valiyev M, May N, Schulze R, Böhm A, Moerkotte G, Grossniklaus M. SAHARA: Memory footprint reduction of cloud databases with automated table partitioning. In *Proc. the 25th International Conference on Extending Database Technology*, Mar. 29–Apr. 1, 2022. DOI: 10.5441/002/edbt.2022.02.

[36] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In *Proc. the 20th International Conference on Very Large Data Bases*, Sept. 1994, pp.487–499.

[37] Ward J H Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 1963, 58(301): 236–244. DOI: 10.1080/01621459.1963.10500845.

[38] Roussopoulos N, Kelley S, Vincent F. Nearest neighbor queries. In *Proc. the 1995 ACM SIGMOD International Conference on Management of Data*, May 1995, pp.71–79. DOI: 10.1145/223784.223794.

[39] Sacca D, Wiederhold G. Database partitioning in a cluster of processors. *ACM Trans. Database Systems*, 1985, 10(1): 29–56. DOI: 10.1145/3148.3161.

[40] Copeland G, Alexander W, Boughter E, Keller T. Data placement in Bubba. In *Proc. the 1988 ACM SIGMOD International Conference on Management of Data*, Jun. 1988, pp.99–108. DOI: 10.1145/50202.50213.

[41] Stöhr T, Märtens H, Rahm E. Multi-dimensional database allocation for parallel data warehouses. In *Proc. the 26th International Conference on Very Large Data Bases*, Sept. 2000, pp.273–284.

[42] Bruno N, Chaudhuri S. An online approach to physical design tuning. In *Proc. the 23rd IEEE International Conference on Data Engineering*, Apr. 2007, pp.826–835. DOI: 10.1109/ICDE.2007.367928.

[43] Garcia-Alvarado C, Raghavan V, Narayanan S, Waas F M. Automatic data placement in MPP databases. In *Proc. the IEEE 28th International Conference on Data Engineering Workshops*, Apr. 2012, pp.322–327. DOI: 10.1109/ICDEW.2012.45.

[44] Karypis G, Kumar V. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical Report, TR 97-061, Univeristy of Minnesota, 1997. https://hdl.handle.net/11299/215346, Mar. 2024.

[45] Kuhn H W. The Hungarian method for the assignment problem. In *50 Years of Integer Programming 1958-2008*, Jünger M, Liebling T M, Naddef D, Nemhauser G L, Pulleyblank W R, Reinelt G, Rinaldi G, Wolsey L A (eds.), Springer, 2010, pp.29–47. DOI: 10.1007/978-3-540-68279-0_2.

[46] Costa E, Costa C, Santos M Y. Evaluating partitioning and bucketing strategies for hive-based big data warehousing systems. *Journal of Big Data*, 2019, 6(1): 34. DOI: 10.1186/s40537-019-0196-1.

[47] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with deep reinforcement learning. arXiv: 1312.5602, 2013. https://arxiv.org/abs/1312.5602, Mar. 2024.

[48] Kallman R, Kimura H, Natkins J, Pavlo A, Rasin A, Zdonik S, Jones E P C, Madden S, Stonebraker M, Zhang Y, Hugg J, Abadi D J. H-store: A high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment*, 2008, 1(2): 1496–1499. DOI: 10.14778/1454159.1454211.

[49] Hoffer J A, Severance D G. The use of cluster analysis in physical data base design. In *Proc. the 1st International Conference on Very Large Data Bases*, Sept. 1975, pp.69–86. DOI: 10.1145/1282480.1282486.

[50] Navathe S, Ceri S, Wiederhold G, Dou J L. Vertical partitioning algorithms for database design. *ACM Trans. Database Systems*, 1984, 9(4): 680–710. DOI: 10.1145/1994.2209.

[51] Navathe S B, Ra M. Vertical partitioning for database design: A graphical algorithm. *ACM SIGMOD Record*, 1989, 18(2): 440–450. DOI: 10.1145/66926.66966.

[52] Chu W W, Ieong I T. A transaction-based approach to vertical partitioning for relational database systems. *IEEE Trans. Software Engineering*, 1993, 19(8): 804–812. DOI: 10.1109/32.238583.

[53] Ng V, Law D M, Gorla N, Chan C K. Applying genetic algorithms in database partitioning. In *Proc. the 2003 ACM Symposium on Applied Computing*, Mar. 2003, pp.544–549. DOI: 10.1145/952532.952639.

[54] Hankins R A, Patel J M. Data morphing: An adaptive, cache-conscious storage technique. In *Proceedings 2003 VLDB Conference*, Freytag J C, Lockemann P, Abiteboul S, Carey M, Selinger P, Heuer A (eds.), Elsevier, 2003, pp.417–428. DOI: 10.1016/B978-012722442-8/50044-6.

[55] Papadomanolakis S, Ailamaki A. AutoPart: Automating schema design for large scientific databases using data partitioning. In *Proc. the 16th International Conference on Scientific and Statistical Database Management*, Jun. 2004, pp.383–392. DOI: 10.1109/SSDM.2004.1311234.

[56] Agrawal S, Narasayya V, Yang B. Integrating vertical and horizontal partitioning into automated physical database design. In *Proc. the 2004 ACM SIGMOD International Conference on Management of Data*, Jun. 2004, pp.359–370. DOI: 10.1145/1007568.1007609.

[57] Gorla N, Betty P W Y. Vertical fragmentation in databases using data-mining technique. *International Journal of Data Warehousing and Mining (IJDWM)*, 2008, 4(3): 35–53. DOI: 10.4018/jdwm.2008070103.

[58] Rodríguez L, Li X O. A support-based vertical partitioning method for database design. In *Proc. the 8th International Conference on Electrical Engineering, Computing Science and Automatic Control*, Oct. 2011. DOI: 10.1109/ICEEE.2011.6106682.

[59] Jindal A, Dittrich J. Relax and let the database do the partitioning online. In *Proc. the 5th International Workshop on Enabling Real-Time Business Intelligence*, Sept. 2011, pp.65–80. DOI: 10.1007/978-3-642-33500-6_5.

[60] Li L Z, Gruenwald L. Self-managing online partitioner for databases (SMOPD): A vertical database partitioning system with a fully automatic online approach. In *Proc. the 17th International Database Engineering & Applications Symposium*, Oct. 2013, pp.168–173. DOI: 10.1145/2513591.2513649.

[61] Rodríguez L, Li X O, Cuevas-Rasgado A D, García-Lamont F. DYVEP: An active database system with vertical partitioning functionality. In *Proc. the 10th IEEE International Conference on Networking, Sensing and Control*, Apr. 2013, pp.457–462. DOI: 10.1109/ICNSC.2013.6548782.

[62] Li L Z, Gruenwald L. SMOPD-C: An autonomous vertical partitioning technique for distributed databases on cluster computers. In *Proc. the 15th IEEE International Conference on Information Reuse and Integration*, Aug. 2014, pp.171–178. DOI: 10.1109/IRI.2014.7051887.

[63] Sun L W, Franklin M J, Wang J N, Wu E. Skipping-oriented partitioning for columnar layouts. *Proceedings of the VLDB Endowment*, 2016, 10(4): 421–432. DOI: 10.14778/3025111.3025123.

[64] Huang Y F, Lai C J. Integrating frequent pattern clustering and branch-and-bound approaches for data partitioning. *Information Sciences*, 2016, 328: 288–301. DOI: 10.1016/j.ins.2015.08.047.

[65] Rodríguez-Mazahua L, Alor-Hernández G, Li X O, Cervantes J, López-Chau A. Active rule base development for dynamic vertical partitioning of multimedia databases. *Journal of Intelligent Information Systems*, 2017, 48(2): 421–451. DOI: 10.1007/s10844-016-0420-9.

[66] Durand G C, Pinnecke M, Piriyev R *et al.* GridFormation: Towards self-driven online data partitioning using reinforcement learning. In *Proc. the 1st International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, Jun. 2018, Article No. 1. DOI: 10.1145/3211954.3211956.

[67] Durand G C, Piriyev R, Pinnecke M *et al.* Automated vertical partitioning with deep reinforcement learning. In *New Trends in Databases and Information Systems*, Welzer T *et al.* (eds.), Springer, 2019, pp.126–134. DOI: 10.1007/978-3-030-30278-8_16.

[68] Liu P J, Li H Y, Wang T Y *et al.* Multi-stage method for online vertical data partitioning based on spectral clustering. *Journal of Software*, 2023, 34(6): 2804–2832. DOI: 10.13328/j.cnki.jos.006496.

[69] Grund M, Krüger J, Plattner H, Zeier A, Cudre-Mauroux P, Madden S. HYRISE: A main memory hybrid storage engine. *Proceedings of the VLDB Endowment*, 2010, 4(2): 105–116. DOI: 10.14778/1921071.1921077.

[70] Jindal A, Quiané-Ruiz J A, Dittrich J. Trojan data layouts: Right shoes for a running elephant. In *Proc. the 2nd ACM Symposium on Cloud Computing*, Oct. 2011, Article No. 21. DOI: 10.1145/2038916.2038937.

[71] Gu X Y, Yang X F, Wang W P, Jin Y, Meng D. CHAC: An effective attribute clustering algorithm for large-scale data processing. In *Proc. the 7th International Conference on Networking, Architecture, and Storage*, Jun. 2012, pp.94–98. DOI: 10.1109/NAS.2012.16.

[72] Arulraj J, Pavlo A, Menon P. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *Proc. the 2016 International Conference on Management of Data*, Jun. 2016, pp.583–598. DOI: 10.1145/2882903.2915231.

[73] Athanassoulis M, Bøgh K S, Idreos S. Optimal column layout for hybrid workloads. *Proceedings of the VLDB*

Endowment, 2019, 12(13): 2393–2407. DOI: 10.14778/3358701.3358707.

[74] McCormick W T, Schweitzer P J, White T W. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, 1972, 20(5): 993–1009. DOI: 10.1287/opre.20.5.993.

[75] Ailamaki A, DeWitt D J, Hill M D, Skounakis M. Weaving relations for cache performance. In *Proc. the 27th International Conference on Very Large Data Bases*, Sept. 2001, pp.169–180.

[76] Li L Z, Gruenwald L. Autonomous database partitioning using data mining on single computers and cluster computers. In *Proc. the 16th International Database Engineering & Applications Sysmposium*, Aug. 2012, pp.32–41. DOI: 10.1145/2351476.2351481.

[77] van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double Q-learning. In *Proc. the 30th AAAI Conference on Artificial Intelligence*, Feb. 2016, pp.2094–2100. DOI: 10.1609/aaai.v30i1.10295.

[78] Jindal A, Palatinus E, Pavlov V, Dittrich J. A comparison of knives for bread slicing. *Proceedings of the VLDB Endowment*, 2013, 6(6): 361–372. DOI: 10.14778/2536336.2536338.

[79] Al-Kateb M, Sinclair P, Au G, Ballinger C. Hybrid row-column partitioning in Teradata®. *Proceedings of the VLDB Endowment*, 2016, 9(13): 1353–1364. DOI: 10.14778/3007263.3007273.

[80] Pinnecke M, Durand G C, Broneske D, Zoun R, Saake G. GridTables: A *One-Size-Fits-Most* H^{2}TAP data store. *Datenbank-Spektrum*, 2020, 20(1): 43–56. DOI: 10.1007/s13222-019-00330-x.

[81] Kang D H, Jiang R C, Blanas S. Jigsaw: A data storage and query processing engine for irregular table partitioning. In *Proc. the 2021 International Conference on Management of Data*, Jun. 2021, pp.898–911. DOI: 10.1145/3448016.3457547.

[82] Abebe M, Lazu H, Daudjee K. Proteus: Autonomous adaptive storage for mixed workloads. In *Proc. the 2022 International Conference on Management of Data*, Jun. 2022, pp.700–714. DOI: 10.1145/3514221.3517834.

[83] Wang J Y, Chai C L, Liu J B, Li G L. FACE: A normalizing flow based cardinality estimator. *Proceedings of the VLDB Endowment*, 2021, 15(1): 72–84. DOI: 10.14778/3485450.3485458.

Peng-Ju Liu received his B.S. degree in information management and information system from Dalian Maritime University, Dalian, in 2020. He is currently pursuing his Ph.D. degree at the School of Information, Renmin University of China, Beijing. His research interests include adaptable data partitioning, load forecasting, and learning-based query optimization.

Cui-Ping Li is currently a professor at Renmin University of China, Beijing. She received her Ph.D. degree from Chinese Academy of Sciences, Beijing, in 2003. Before that, she received her B.S. and M.S. degrees from Xi'an Jiaotong University, Xi'an, in 1994 and 1997, respectively. She received the Second Prize of the National Award for Science and Technology Progress in 2018. Her main research interests include social network analysis, social recommendation, and big data analysis.

Hong Chen is currently a professor at Renmin University of China, Beijing. She received her Ph.D. degree from Chinese Academy of Sciences, Beijing, in 2000. Before that, she received her B.S. and M.S. degrees from Renmin University of China, Beijing, in 1986 and 1989, respectively. She received the Second Prize of the National Award for Science and Technology Progress in 2018. Her research interests include database technology and high-performance computing.