HTAP Databases: A Survey

Chao Zhang, Guoliang Li*, *Fellow*, *IEEE*, Jintao Zhang, Xinning Zhang, Jianhua Feng

Abstract—Since Gartner coined the term, Hybrid Transactional and Analytical Processing (HTAP), numerous HTAP databases have been proposed to combine transactions with analytics in order to enable real-time data analytics for various data-intensive applications. HTAP databases typically process the mixed workloads of transactions and analytical queries in a unified system by leveraging both a row store and a column store. As there are different storage architectures and processing techniques to satisfy various requirements of diverse applications, it is critical to summarize the pros and cons of these key techniques. This paper offers a comprehensive survey of HTAP databases. We mainly classify state-of-the-art HTAP databases according to four storage architectures: (a) Primary Row Store and In-Memory Column Store; (b) Distributed Row Store and Column Store Replica; (c) Primary Row Store and Distributed In-Memory Column Store; and (d) Primary Column Store and Delta Row Store. We then review the key techniques in HTAP databases, including hybrid workload processing, data organization, data synchronization, query optimization, and resource scheduling. We also discuss existing HTAP benchmarks. Finally, we provide the research challenges and opportunities for HTAP techniques.

Index Terms—HTAP databases, Data organization, Data synchronization, Query optimization, Resource scheduling, Benchmarks

✦

1 INTRODUCTION

Hybrid Transactional, and Analytical Processing (HTAP) was defined by Gartner [98], [35]. Since then, HTAP techniques have been deployed in various data-intensive applications, e.g., banking and finance [87], E-commerce [137], and fraud detection [106]. The Gartner report envisioned that, by 2024, HTAP techniques will be widely adopted in numerous business applications that entail real-time data analytics. Compared with the traditional data processing pipeline that processes transactions and analytical queries separately, HTAP architecture enables a unified system that not only can handle online transactional processing (OLTP) efficiently, but also can perform online analytical processing (OLAP) concurrently [35]. Such an architecture aims to eliminate the need for an explicit Extract-Transform-Load (ETL) process, thereby enabling real-time data analytics on transaction data. For instance, HTAP databases allow entrepreneurs in retail applications to analyze the latest transaction data in real-time, and then they can roll out promotional sales based on the gained insights. In finance applications, HTAP databases enable vendors to process customer transactions efficiently while detecting fraudulent transactions simultaneously [99].

Over the last decade, we have witnessed the emergence and evolvement of various HTAP databases [17], [24], [28], [38], [45], [57], [60], [74], [78], [83], [86], [102], [103], [108], [118], [137]. Since it is well recognized that a row store is ideal for OLTP workloads, and a column store is better suited for OLAP workloads [32], [57], HTAP databases mainly adopt a dual-store architecture that leverages both a row store and a column store. However, different categories of HTAP databases adopt disparate storage strategies and processing techniques to cater to various applications. For instance, it depends on whether OLTP or OLAP has a higher priority for the applications (e.g., OLTP is the first citizen for banking scenarios while OLAP dominates the analytical reporting). It also depends on the requirements of availability, scalability, performance, and data freshness [19]

specified in the service level agreements (SLAs) [38] (e.g., large-scale E-commerce applications require high scalability while banking and finance applications demand high data freshness). Nevertheless, HTAP databases must strike a trade-off between performance isolation and data freshness when handling the mixed workloads of OLTP and OLAP. The main reason is that the two types of workloads exhibit a completely different computing pattern and can intervene with each other, i.e., OLTP workloads [121] are update-heavy and short-lived while OLAP workloads [119] are read-heavy and bandwidth-intensive. Consequently, HTAP databases cannot guarantee the metrics of query throughput and data freshness simultaneously, albeit with a unified architecture. As different applications deliver disparate requirements, it is critical to study and understand the pros and cons of HTAP databases under various architectures.

There are five main challenges that HTAP databases need to address, including (C1) hybrid workload processing, (C2) data organization, (C3) data synchronization, (C4) query optimization, and (C5) resource scheduling.

Challenge 1. Hybrid Processing Challenge. The first challenge is to process mixed workloads of OLTP and OLAP efficiently while maintaining a high data freshness. As there is a trade-off between performance isolation and data freshness, it is challenging for HTAP databases to balance performance and freshness. Hence, these systems must make a compromise on either performance isolation or data freshness, depending on the specific HTAP applications.

Challenge 2. Data Organization Challenge. The second challenge is to organize the data adaptively for HTAP workloads with high throughput and low storage costs. There are two rules of thumb: (i) the row store is suitable for OLTP workloads and (ii) the column store is ideal for OLAP queries. However, storing a copy of the entire data for both formats leads to high storage overhead (i.e., space cost and maintenance cost). Therefore, HTAP systems need to make wise decisions for data organization (e.g., row-wise or column-wise data layout) to reduce the storage overhead while delivering high system performance.
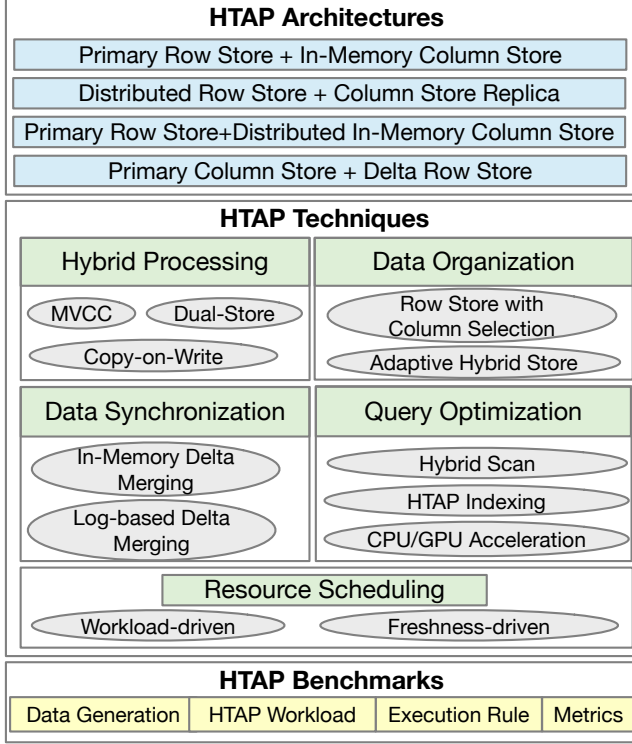
HTAP Architectures
Primary Row Store + In-Memory Column Store
Distributed Row Store + Column Store Replica
Primary Row Store+Distributed In-Memory Column Store
Primary Column Store + Delta Row Store

HTAP Techniques

Hybrid Processing	Data Organization
MVCC Dual-Store Copy-on-Write	Row Store with Column Selection Adaptive Hybrid Store

Data Synchronization	Query Optimization
In-Memory Delta Merging Log-based Delta Merging	Hybrid Scan HTAP Indexing CPU/GPU Acceleration

Resource Scheduling
Workload-driven   Freshness-driven

HTAP Benchmarks			
Data Generation	HTAP Workload	Execution Rule	Metrics

Fig. 1. An Overview of HTAP Architectures, Techniques, Benchmarks

Challenge 3. Data Synchronization Challenge. The third challenge is to decide when to synchronize the delta data from the transactional store to the analytical store for high data freshness while keeping high throughput and scalability. On the one hand, immediately merging the delta data to the analytical store can keep high data freshness. However, it can greatly affect the performance due to the merging overhead. On the other hand, merging the delta data on demand can improve the throughput but lead to low data freshness. Hence, HTAP databases need to synchronize the data adaptively.

Challenge 4. Query Optimization Challenge. The fourth challenge is to optimize the queries with both a row store and a column store. In the HTAP databases, a query can be executed against either the row store or the column store; thus the query optimizer must judiciously decide whether the row-based execution or the columnar execution is more beneficial. However, it is challenging to generate an optimal query plan in a large search space. Therefore, the HTAP optimizer must balance the trade-off between the planning time and query execution latency.

Challenge 5. Resource Scheduling Challenge. The fifth challenge is to schedule the resources (e.g., CPU threads and memory) for OLTP and OLAP instances effectively for high throughput and data freshness. On the one hand, assigning more resources to OLAP workloads favors high query throughput but may block the OLTP threads due to the limited bandwidth. On the other hand, scheduling more resources to OLTP workloads can accelerate transaction processing but may lead to low data freshness. Hence, the HTAP resource scheduler must balance the trade-off between performance isolation and data freshness as well.

In this work, we provide a comprehensive survey of HTAP databases in three aspects, including HTAP architectures, techniques, and benchmarks. Figure 1 presents an overview of HTAP-related techniques. It gives a taxonomy of HTAP architectures and HTAP techniques, respectively. It also presents four components of HTAP benchmarks. We pay particular attention to how existing approaches address the above-mentioned challenges, respectively.

1.1 HTAP architectures

We mainly study HTAP databases that utilize row stores and column stores together to handle the mixed workloads of OLTP and OLAP efficiently in a single database system. Based on the storage strategies and processing paradigm, we divide their architectures into four categories as follows:
(1) Primary Row Store+In-Memory Column Store. This category of HTAP databases leverages a primary row store as the basis for OLTP workloads and processes OLAP workloads with an in-memory column store. Updates are appended to the delta store, which will be periodically merged with the column store. We review four representatives: Oracle [57], SQL Server [60], and DB2 BLU [108].
(2) Distributed Row Store+Column Store Replica. This category relies on a distributed architecture to support HTAP. The master node handles the read-write transactions and asynchronously replicates the logs to the slave nodes. The primary storage relies on a distributed row store, and some slave nodes will be chosen as column-store servers for query acceleration. We introduce two representatives: TiDB [45] and F1 Lightning [136].
(3) Primary Row Store+Distributed In-Memory Column Store. This type of database utilizes a primary with a distributed in-memory column store (IMCS) to enable HTAP. We present a representative: MySQL Heatwave [83].
(4) Primary Column Store+Delta Row Store. This category of databases utilizes an in-memory column store as the basis for OLAP, and handles OLTP with a delta row store, which will be eventually merged into the column store. We introduce SAP HANA [118] and Hyper [86].
Other HTAP architectures. We also review other types of HTAP architectures that complement the major architectures, including (i) row-only HTAP architectures; (ii) column-only HTAP architectures; (iii) Spark-based HTAP architectures; and (iv) cloud-native HTAP architectures.

1.2 HTAP techniques

This section takes a deep dive into the key techniques of HTAP databases, including hybrid workload processing, data organization, data synchronization, query optimization, and resource scheduling.
(1) Hybrid Workload Processing. There are three kinds of hybrid workload processing techniques, including (i) Multi-Version Concurrency Control (MVCC) techniques [55], [54] (ii) Copy-on-Write (CoW) techniques [52], [9]; and (iii) Dual-store based processing [57], [60], [83], [118], [45]. First, MVCC-based techniques process the hybrid workloads on the same copy of multi-versioned data. Second, the CoW technique relies on snapshotting to support HTAP, where the main process handles the transactions and the forked

processes perform the queries. Third, dual-store-based processing utilizes a transactional store for OLTP workloads, and employs an analytical store for OLAP workloads.

(2) Data Organization. We introduce two types of data organization techniques, including (i) primary row store with the selected column store and (ii) adaptive hybrid data storage. Both methods adaptively generate the data storage based on the given workloads. The former organizes the data by selectively replicating the data from the row store to the column store, including frequency-based heatmap [92] and integer programming [16]. The latter organizes each table in a hybrid row and columnar format by vertically and horizontally partitioning the tables, including the cost-based approaches [5], [12], a clustering approach [11], and a deep-learning-based approach [2].

(3) Data Synchronization (DS) There are two types of DS techniques for various HTAP databases including (i) in-memory delta merge [57], [60], [83], [108], [118] and (ii) log-based delta merge [21], [45], [136]; The first category periodically merges the newly-inserted in-memory delta data to the memory-based column store. The second category [45], [136] merges the deltas to the column store based on multi-level delta merging, including log shipping and replaying.

(4) Query Optimization. We introduce three types of query optimization techniques, including (i) hybrid row/column scan [45], [60]; (ii) HTAP indexing [110], [127] and (iii) CPU/GPU acceleration for HTAP [9], [62]. The first type [83], [92] optimizes query plans by selecting the access path of a row store or a column store. The second type relies on path-copying and multi-version indexing techniques to speed up HTAP. The third type leverages heterogeneous CPU/GPU architecture to accelerate HTAP workloads.

(5) Resource Scheduling. There are two types of scheduling techniques: the workload-driven approaches [118], [120] and the freshness-driven approach [109]. The former adjusts the parallelism of OLTP and OLAP threads based on the performance of executed workloads. The latter [109] switches the execution modes on resource allocation and data exchange for OLTP and OLAP workloads by considering the data freshness.

1.3 HTAP benchmarks

We present state-of-the-art end-to-end benchmarks and micro-benchmarks for evaluating HTAP databases. We introduce four end-to-end HTAP benchmarks, including CH-benchmark [26], HTAPBench [25], OLxPBench [51], HATtrick [80], and HyBench [141]. We focus on the key components, including data generation, HTAP workload, execution rule, and metrics. In addition to the end-to-end benchmarks, we will introduce three synthetic micro-benchmarks for data organization [11], [12], [49].

1.4 Contributions

Differences with existing surveys. In this paper, we focus on fundamental techniques of HTAP databases [64]. We also summarize the pros and cons of various architectures and techniques. Özcan et al. [93] discussed various HTAP systems rather than the key techniques of HTAP databases. Hieber et al. [44], [43] reviewed HTAP systems from several dimensions, including architecture, query handling, and

concurrency control. However, it lacked an in-depth analysis of HTAP databases and neglected many fundamental HTAP techniques, such as hybrid workload processing, data organization, data synchronization, and query optimization. Compared with a previous survey [140], this work has a significant amount of new content: (1) it gives a systematic overview and introduces five HTAP challenges in Section 1; (2) it reviews the evolution of HTAP databases in the history and introduces a trade-off between data freshness and performance isolation in Section 2; (3) it gives a more detailed analysis on the architectures in Section 3, including the cloud-native HTAP architectures; (4) it introduces a new taxonomy of the key techniques and presents each type of techniques in more details in Section 4, including the hybrid workload processing; (5) it presents eight state-of-the-art HTAP benchmarks in Section 5, including OLxPBench [51], HATtrick [80], HyBench [141], and mOLxPBench [49]; (6) it presents six research directions with many new open problems in Section 6, including HTAP for multi-model data analytic [117], serving atop HTAP [79], [50], and cloud-native HTAP techniques [39], [103], [122].

To summarize, we make the following contributions:

1) We survey HTAP databases. We introduce a taxonomy of state-of-the-art HTAP databases according to their storage architectures. We also discuss their pros and cons on performance, scalability, and data freshness.
2) We summarize HTAP techniques. We take a deep dive into the key HTAP techniques concerning hybrid workload processing, data organization, data synchronization, query optimization, and resource scheduling.
3) We review HTAP benchmarks. We introduce the state-of-the-art benchmarks on HTAP databases. We present their schema, workloads, execution rules, and metrics.
4) We provide new research challenges and discuss future directions, including data organization for distributed HTAP databases, HTAP query optimization, and cloud-native HTAP techniques.

2 BACKGROUND OF HTAP DATABASES

In this section, we introduce the background of HTAP databases. We first review the evolution of HTAP databases by introducing four phases of HTAP development and applications, and then we introduce a trade-off between data freshness and performance isolation.

2.1 The Evolution of HTAP Databases

Figure 2 depicts a timeline of the HTAP databases between 2010 and 2022. We place the systems in the year when they first released the HTAP functionality from the literature or from the publicly released material. By investigating these HTAP databases, we mainly classify the development of HTAP databases into four phases as follows:

Year 2010-2014: Standalone Column-based HTAP databases. In the first phase, HTAP databases mainly adopted standalone column-based databases, representatives are SAP HANA [34], Hyrise [40], Hyper [52]. Since HTAP was not formally defined at that time, they named such a technique as "hybrid OLTP&OLAP" [52] or "OLxP" [84]. Back then,
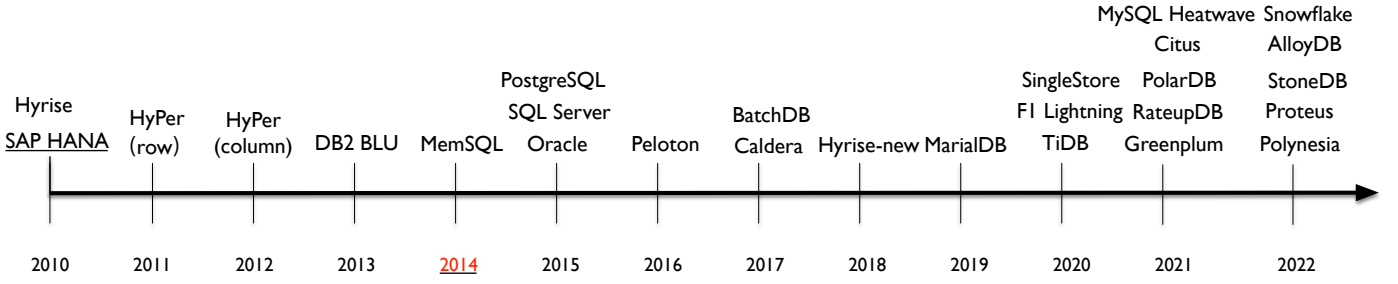
Fig. 2. A Timeline of HTAP databases that first released the HTAP functionality in the literature

the applications they targeted were mainly analytical applications with read-heavy transactional workloads, such as the ERP applications [34].

Year 2014-2019: Standalone Row-based HTAP databases. The year 2014 is the time when the HTAP term was formally defined by a Gartner report [98]. It initially defined HTAP as an application architecture that utilized in-memory computing technologies to enable hybrid processing on the same in-memory data store. In 2018, Gartner extended the HTAP concept to "In-Process HTAP" [35], which supported weaving hybrid workload processing techniques together as needed to accomplish the business task. Such a new definition indicated that HTAP is no longer limited to in-memory computing techniques, which significantly expanded the HTAP applications. During this period, major relational databases extended the primary row store with a column store. To name a few, DB2 BLU [108], MemSQL [89], SQL Server [60], Oracle [57], PostgreSQL [84] and MarialDB [78]. The applications they targeted were medium-scale transaction processing applications with real-time data analytics, such as banking and finance services [87] and applications of fraud detection [107], [106].

Year 2019-2022: Distributed HTAP databases. In the third phase, there emerged many distributed HTAP databases, such as SingleStore [103], Citus [28], F1 Lightning [136], and TiDB [45]. On the one hand, these databases embraced the NewSQL movement [95] by developing distributed SQL-based transaction processing systems with high scalability and strong consistency. On the other hand, they caught up with the HTAP wave by expanding their capacity of scalability and consistency to HTAP, e.g., adding distributed columnar storage and unified data replication. Generally, they are suitable for large-scale data-intensive applications such as E-commerce [137], Internet of Things (IoT) [46], and real-time social media [69].

Year 2022-present: Cloud-Native HTAP databases. As cloud databases are proliferating, there have been emerging many cloud-native HTAP databases [63]. Two representatives are AlloyDB [39] and Snowflake [122]. With the disaggregation of computing and storage, they enable HTAP with high elasticity, high availability, and multi-tenancy. First, since compute and storage resources can be scheduled on demand individually, they provide high elasticity for HTAP. Second, as the data is replicated across many availability zones and is backed by the scalable cloud service, cluster and node failures can be recovered quickly. Third, as the resources are virtualized and shared by multiple tenants, they are more cost-efficient. Since Gartner predicted [100] that the revenue
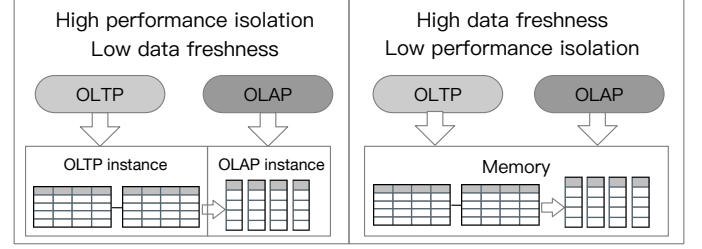

Fig. 3. A trade-off between data freshness and performance isolation

from the cloud databases will account for 50% of total DBMS market revenue in the near future, we believe cloud-native HTAP databases will find a wide spectrum of applications.

2.2 A Trade-off between Freshness and Isolation

Data freshness. As OLTP workloads are updating the data, HTAP databases need to guarantee that the fresh data is accessed by the analytical queries. Hence, *the data freshness refers to the lag time between the analytics and transactions.* One way of quantifying data freshness is to take the maximum value of the timestamp differences between the result sets of the OLAP client and OLTP client [141]. Additionally, the data replication latency can also reflect the data freshness. Particularly, with the separated transactional store and analytical store, the fresh data is periodically replicated in the analytical store. Since many approaches [45], [116], [136] must merge the newly-updated data to the analytical store before performing the data analytics, it measures how fast the recently committed transactions are synchronizing to the analytical store such that the analyzed data is fresh.

Performance isolation. In HTAP databases, *performance isolation [25], [26], [45], [105], [116] refers to the system's capacity of reducing the performance degradation when processing the dynamic hybrid workload.* In other words, it reflects how well the systems can reduce the interference between OLTP and OLAP workloads when executing them concurrently. Regardless of the isolation architectures (in the same node or different nodes), such metrics directly reflect HTAP databases' performance of handling hybrid workloads w.r.t. the same number of OLTP/OLAP clients and the same scale factor of a dataset.

A Trade-off. HTAP databases must balance a trade-off between data freshness and performance isolation. As shown in Figure 3, handling the hybrid workloads with separated instances can provide high-performance isolation but may lead to low data freshness, hence the outdated results of data analytics. For instance, TiDB [45] only degraded up

to 10% of the performance while it took up to 1000 ms to apply the change logs for data replication. Handling the hybrid workloads in a unified memory space of a single sever favors high data freshness but it can greatly degrade the performance. For instance, Hyper [86] took only microseconds to create a fresh snapshot, but it degraded up to 40% of the performance. To quantify the effectiveness of performance isolation, the degradation percentage [45], [105] of transactional/analytical throughput is mostly used. One way of quantifying the performance isolation is to compare the performance between a sequential execution and a hybrid execution with the hybrid workload[141]. The lower the metric is, the better the performance isolation is.

3 HTAP ARCHITECTURES

In this work, we focus on the HTAP architectures with a row store and a column store in a single DBMS. We do not consider the loosely-coupled HTAP architectures [93] as those architectures employ multiple databases to form an HTAP solution that entails a costly ETL process. We classify the tightly-coupled HTAP architectures into four categories: (a) Primary Row Store and In-Memory Column Store; (b) Distributed Row Store and Column Store Replica; (c) Primary Row Store and Distributed In-Memory Column Store; and (d) Primary Column Store and Delta Row Store. Table 1 presents their representative databases, OLTP paradigm, OLAP paradigm, and delta storage.

3.1 Primary Row Store+In-Memory Column Store

This category of databases [57], [60], [108] leverages a primary row store and an in-memory column store in a single node. It relies on a row store for handling the OLTP workload, and the data will be persisted to the disk in a row-wise page store. Column store is optimized for OLAP workload with compression techniques [1], [15] and standalone columnar scans [57], [60], [108]. An in-memory delta store is utilized to record the recent DML operations, which will be merged into the column store periodically. For transaction processing, read/write transactions will be handled by the row engine with the ACID guarantee based on MVCC; data updates will also be recorded in the delta store. For analytical processing, long-running queries are processed using the columnar scan, and the delta data that has not been merged should be traversed to access the fresh data. This category of system has high analytical throughput because the OLAP workloads are processed using an in-memory column store. Data freshness is also high because the column store engine can access the latest data in the main memory. However, limited by memory capacity, the scalability of this category of system is low. In addition, because transactional and analytical workloads are processed in the same machine, the isolation of the system is low.

3.1.1 Representatives of HTAP Databases

With architecture (a), Oracle in-memory dual-format database [57], [92] combines the row-based buffer cache with an in-memory column store (IMCS) to handle OLTP and OLAP workloads simultaneously. IMCS consists of in-memory compression units (IMCUs), and each IMCU is immutable and can only be populated from the buffer; the data changes are cached in the snapshot metadata unit (SMU), and each IMCU is associated with one SMU. To merge the updates to the IMCU, it must create a new IMCU that incorporates the data in the corresponding SMU. Another representative is SQL Server [60], where an in-memory row-based engine, called Hekaton [30], is integrated for handling OLTP workloads. Its underlying storage is based on a disk-based row store for logging the operations and persisting the data. It also builds the column store index (CSI) for handling complex queries. Different from Oracle, the in-memory columns in CSI that are infrequently accessed will be compressed and persisted to the disk, and CSI is updatable. Besides, the data changes are appended to the tail of the in-memory table, and they are indexed by a tail index, e.g., a B-tree. By scanning the tail index, data changes will be periodically merged into the column store. DB2 also adopts such an architecture; it has been deeply integrated with an IMCS accelerator called DB2 BLU [108] to incorporate advanced column-based techniques such as compression-based operations and single-instruction multiple-data (SIMD) instructions. BLU also supports updating the column store.

3.1.2 Challenges & Opportunities

HTAP databases with architecture (a) need to improve the scalability and performance isolation while maintaining high performance and high data freshness. The main challenge is how to scale and isolate the OLAP workloads using the column store. One possible solution is to scale the OLAP workloads with a distributed in-memory architecture [125]. In addition, placing which columns into the column store with a memory budget is also challenging as it is an NP-hard problem [16].

3.2 Distributed Row Store+Column Store Replica

The second architecture is a distributed cluster including a master node and multiple secondary nodes. The master node handles the read/write transactions; the secondary nodes are read-only. Particularly, when handling the transaction requests, the master node asynchronously replicates the logs to the secondary nodes for data synchronization. The primary node adopts a row store, certain secondary nodes will be chosen to adopt a column store for query acceleration. Transactions are handled in a distributed way for high scalability; complex queries are performed in the secondary nodes with a column store. For the pros, it has high performance isolation and high scalability as the mixed workloads are processed on separated nodes. For the cons, the data freshness is low since newly-updated data may not have been merged into the column store.

3.2.1 Representatives of HTAP Databases

Two representatives with architecture (a) are TiDB [45] and F1 Lightning [136]. TiDB is a Raft-based distributed HTAP database [91], which consists of multiple row-based nodes, called TiKV nodes, among which a leader node handles the read-write transactions. The leader node asynchronously sends the data to other follower nodes; the follower nodes only serve the read transactions and consistently communicate with the leader. Only if the leader node has failures, a follower node can become a leader by voting. For the data
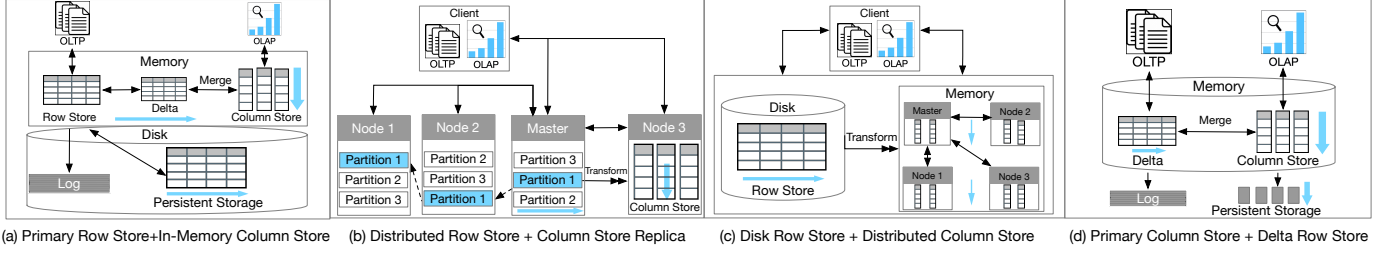
Fig. 4. A Taxonomy of State-Of-The-Art HTAP Databases based on the Storage Architecture and Processing Paradigm

TABLE 1
A Classification of State-Of-The-Art HTAP Databases based on the Storage Architecture

Category	HTAP databases	OLTP Paradigm	OLAP Paradigm	Delta Store
Primary Row Store + In-Memory Column Store	Oracle [57], SQL Server [60]	Standalone Row-wise MVCC	Standalone Columnar Scan with Delta Traverse	In-Memory Table
Distributed Row Store + Column Store Replica	TiDB [45] F1 Lightning [136]	Distributed Row-wise 2PC + Paxos	Distributed Columnar Scan with Log Replay	B-tree, Change Log
Primary Row Store + Distributed In-Memory Column Store	MySQL Heatwave[83]	Standalone Row-wise MVCC	Distributed Columnar Scan with Log Replay	Change Log
Primary Column Store + Delta Row Store	SAP HANA [118], Hyper [86]	Cache-based Row-wise MVCC	In-Memory Columnar Scan with Delta Traverse	In-Memory Dictionary

storage, the underlying data is partitioned into multiple row-based regions. One or more servers will be selected as learner nodes that store columnar replicas for analytical processing. Learner nodes are read-only and do not participate in voting. It adopts a global 2-Phase-Commit (2PC) protocol [96] to handle distributed transactions based on a global time stamp. For analytical processing, it develops a centralized cost-based optimizer that supports cross-engine query processing where queries can be pushed down to either a row engine or a column engine. TiDB relies on the Raft protocol for data replication [91]. The master node asynchronously replicates logs to the follower and learner nodes for log replaying, and it builds a delta merge tree to track the changes and merges them to the column store periodically. F1 Lightning [136] is another representative, which integrates a data replication service into an OLTP engine that is built on top of Spanner [27], a distributed OLTP database with strong consistency and high scalability that organizes the row-based partitions in multiple regions. The lightning server contains a component, called changepump, which uses the change data capture mechanism to detect new changes, then transforms them from row-wise format to columnar format and merges them into the storage. The memory-resident deltas are row-wise B-tree, and Lightning periodically checkpoints memory-resident deltas to disk. When the deltas are too large, Lightning merges and transforms them into the column store. The log-structured merge (LSM) reader merges both memory-resident deltas and disk-based deltas by merging and collapsing. Particularly, merging deduplicates changes in the deltas and copies distinct versions to the new delta; collapsing combines multiple versions of the same key into a single version. F1 lightning adopts a Paxos-based 2PC protocol to handle distributed transactions; a distributed query engine [58], F1 Query [113], is employed for query processing. For the specified times-

tamp that is in the query window (normally 10 hours), the pushdown evaluator reads the corresponding data from the columnar file in the lightning server, and then obtains other data from Spanner.

3.2.2 Challenges & Opportunities

HTAP databases with architecture (b) need to increase the data freshness due to the high overhead of log shipping, transformation, and delta merging in the distributed architecture. The main challenge is how to efficiently merge the delta files to the column store. Two possible solutions are to (1) develop a memory-based delta logging and shipping [116] and (2) design new indexing techniques for delta merging [114], [73]. Moreover, how to effectively organize the data (e.g., data layout, data placement, and column compression) for a distributed HTAP database is also a challenging task [2], [3].

3.3 Primary Row Store+Distributed IMCS

This kind of HTAP databases [83], [125] utilizes a row store with a distributed in-memory column-store (IMCS) to support HTAP. Two main differences between architecture (a) and (c) are about the OLAP paradigm and delta store. For the OLAP paradigm, the former one adopts the standalone in-memory columnar scan with delta traverse while the latter one relies on distributed in-memory columnar scan with log replay. Regarding the delta store, architecture (a) employs the in-memory table while architecture (c) uses the change log, so they have the different data synchronization methods (See Section 4.3).

With architecture (c), the row engine processes the OLTP workloads, and the IMCS handles the query processing. The columnar data is extracted from the row store, and the hot data resides in IMCS and the cold data will be evicted to disk. For the pros, it has high performance isolation as the

hybrid workloads are processed in different nodes. Moreover, it has a high OLAP throughput and scalability because of the distributed IMCS. For the cons, it has medium or low data freshness, depending on the deployment mode of the IMCS cluster (e.g., on-premise or on-cloud). In addition, it has low horizontal scalability on OLTP due to a standalone row store for transaction processing.

3.3.1 Representatives of HTAP Databases

MySQL Heatwave [83] is a representative that employs architecture (c). Specifically, it tightly couples the MySQL database with a distributed IMCS cluster, called Heatwave, to enable real-time analytics. Transactions are fully executed in the MySQL database. Columns that are frequently accessed will be loaded into the Heatwave. When a complex query comes in, it can be pushed down to Heatwave for query acceleration. The columnar data is extracted from the MySQL database, and the hot data resides in the Heatwave cluster. For data synchronization, the latest transaction data will be automatically transformed and merged to the column store in three cases: (i) every 200 milliseconds; (ii) when the buffer size reaches 64 MB, or (iii) when the queries need to access the latest updated data. The heatwave cluster also developed the auto-pilot service to automate the processes of data partition, query execution and scheduling.

3.3.2 Challenges & Opportunities

HTAP databases with architecture (c) need to increase the data freshness due to the distributed in-memory column store. The main challenge is how to balance the data freshness and OLAP throughput. Possible solutions are to design (1) cost functions or (2) ML models for column data management, including column selection and compression [47].

3.4 Primary Column Store+Delta Row Store

This category of databases utilizes the primary column store as the basis for OLAP, and handles OLTP with a delta row store. The primary column store stores the whole data in the main memory. Data updates are appended to the row-based delta store. The system periodically merges the delta data into the column store. For the pros, it has high data freshness as the hybrid workloads are processed in the main memory. Moreover, it has a high OLAP throughput because of the primary column store. For the cons, it has low OLTP scalability due to the delta-based row store. In addition, it has low-performance isolation due to the standalone architecture for hybrid workload processing.

3.4.1 Representatives of HTAP Databases

SAP HANA [34], [118] and Hyper [52], [86] are two representatives of architecture (d). SAP HANA divides the in-memory data store into three layers: L1-delta, L2-delta, and Main. The L1-delta keeps data updates in a row-wise format. When a threshold is reached (e.g., 100k tuples), the data in L1-delta is transformed and merged to L2-cache based on a local dictionary. The L2-delta transforms the data into columnar data, then merges the data into the main column store based on a global dictionary. Finally, the columnar data is persisted in the disk storage. For high data freshness, the OLAP client will traverse the delta

records when scanning the column store. In addition, for non-transformed records, it can use in-place updates for the delta store. To update transformed records, it replaces an update operation with a delete and an insert operations on the delta store. Hyper [52], [86] was initially based on a row store. Now it has supported the architecture of a primary column store with a delta row store. Specifically, it uses the buffer to handle concurrent transactions based on an MVCC protocol. The version vector stores all the versions for each unique tuple. Instead of periodically merging the data in the version vector to the column store, Hyper adopts transaction-level garbage collection, where all of the versions that were generated by the transactions can be safely removed after the transactions have been applied to the column store. Finally, the columnar data is updated in-place by applying all the committed transactions.

3.4.2 Challenges & Opportunities

For HTAP databases with architecture (d), two main problems need to be addressed. (1) they need to increase the OLTP scalability due to the delta-based transaction processing. (2) they need to increase performance isolation due to the unified memory pool for HTAP. The main challenge is how to traverse the delta storage efficiently while keeping high throughput for HTAP. Although they have offered certain scaling-out solutions [38], [82], these approaches need to be further justified due to the centralized transaction scheduler.

3.5 Applications of HTAP Architectures

HTAP databases with the above-mentioned architectures have their merits and demerits. Hence, "one HTAP database cannot fit all", especially for different applications. By comparing their pros and cons, we summarize the gained insights and give the recommendations as follows.

(1) Architecture (a) is suitable for the applications that require high throughput and data freshness, but the demand for scalability is not high. Representatives are the banking and finance services [87] that need to process and analyze the customers' transactions efficiently. Therefore, these applications have a high requirement on the system throughput. Since the number of target customers is almost fixed, these applications do not require high scalability.

(2) Architecture (b) is ideal for applications that require high scalability and can have tolerable data freshness. A representative is an E-commerce application with real-time data analytics. Such applications need to process a large number of concurrent transactions from multiple customers, especially for holidays, e.g., Double-Eleven in Alibaba [137]. Therefore, the scalability requirement must be fulfilled. Nevertheless, such applications do not force zero freshness as it is acceptable that there is any inconsistency in a short period of time, e.g., customer retention rate.

(3) Architecture (c) is suitable for applications that require high analytical throughput and scalability, but the demand for data freshness is not high. A representative is the IoT applications with real-time data analytics [46]. On the one hand, these applications have a high requirement for analytical throughput and scalability. Thus, a distributed in-memory column store is a good fit for such cases. On the

other hand, the data updates in these scenarios are rare, thus a standalone row store is sufficient.

(4) Architecture (d) is a good fit for applications that require high analytical throughput and high data freshness. A representative is real-time fraud detection [106], [107], which requires high data freshness as they can not tolerate any fraud, which could cause significant consequences. Nevertheless, the scalability requirement is not high.

3.6 Other Architectures

Other than the four main types of HTAP architectures, there are other HTAP architectures, including (1) row-only HTAP architectures; (2) column-only HTAP architectures; (3) Spark-based HTAP architectures; and (4) cloud-native HTAP architectures. We summarize them as follows:

(1) Row-only HTAP architectures [52], [77] rely on the purely row store to enable HTAP. For instance, the first version of Hyper [52] employed a copy-on-write mechanism to fork an OLAP process to operate on a separate snapshot. Its main process handles the transactions simultaneously, and both the OLTP and OLAP processes are based on the row-based data store. BatchDB [77] leverages the primary-secondary replication, where the primary row-wise replica is used for OLTP, and the secondary row-wise replica is in charge of OLAP. It uses a batch-based propagation mechanism to synchronize the updates from the primary replica to the secondary replica. This type of architecture relies heavily on the row-based query processing in the main memory to process the hybrid workload. Thus, both the freshness and OLTP throughput are high. However, the major drawback is that the analytical throughput is much lower compared to the column-based query processing.

(2) Column-only HTAP architectures [9], [40], [62], [112] rely solely on a column store to support HTAP. For example, Hyrise [40] initially employed an adaptive columnar layout to support HTAP, and the basic idea is to use narrower column groups to handle OLAP workloads and leverage wider column groups to handle OLTP workloads. The latest version of Hyrise [31] develops the chunk-based column store, a PAX [4]-like data layout that horizontally divides a table into partitions (a.k.a., row groups), and each partition is organized in columns. NoisePage [88] (previously named Peloton [11]), a self-driving columnar database [20], [75], [76], [94], also adopts such a data layout based on Apache Arrow [8] format. Caldera [9] relies on the copy-on-write mechanism with CPU/GPU architecture where OLTP workloads are handled by multi-threads of CPU and OLAP workloads are executed with GPU in parallel. Another case is RateupDB [62], which also adopts the CPU/GPU architecture. Different from Caldera [9], it relies on the primary-secondary replication to enable HTAP. TiQue is a recent work that adds the transaction logic in SQL and enables HTAP on top of a column store. The basic idea is to add the delta table and transaction metadata for an analytical database, thereby accelerating the transaction processing. Overall, column-only HTAP architectures have a high analytical throughput, but the data updates and data synchronization can significantly affect the system's performance.

(3) Spark-based HTAP architectures. Such systems [13], [81] support HTAP by combining an OLTP engine and an OLAP engine. The OLTP engine is dominated by columnar databases such as HBase [42], and the OLAP engine mainly uses a Spark [139] engine. Both engines share the data in a distributed file system (such as HDFS [18]). For example, Splice Machine [124] and Phoenix [101] handle data updates based on HBase, and leverage Spark for big data analytics. SnappyData [81] integrates a transactional engine, Apache Geode [37], into the Spark for processing streaming, transactional, and interactive analysis simultaneously. Wildfire [13] tightly couples the Spark with a distributed transaction engine. In the upper layer, it provides a unified interface to take as input the hybrid workloads. The middle layer schedules the OLTP workloads with Scala API to the transactional engines, and routes the Spark SQL queries to Spark executors. In the storage layer, they organize the data in the LSM-tree format with an indexing support [73]. The Spark-based HTAP systems have high scalability and are suitable for big data analytics with modest data updates. However, the data freshness is low due to the shared file storage.

(4) Cloud-Native HTAP architectures [22], [23], [39], [103], [122] rely on the disaggregation of compute and storage to enable HTAP. For instance, AlloyDB [39] is a PostgreSQL-compatible cloud-native HTAP service. In its compute layer, the compute nodes rely on machine learning to convert the in-memory data from the row format to columnar format for query acceleration. In its storage layer, AlloyDB has developed an elastic log storage service that organizes the data in shards and asynchronously materializes the WAL records to fresh pages, and the dirty pages in the compute layer are never flushed into the storage [132]. Another example is SingleStore (the successor of MemSQL [89]), which proposes a unified table storage structure based on a cloud-native architecture. It relies on a distributed in-memory row store for handing updates. The data is persisted to the disk-based columnar storage, and is organized as LSM trees in the storage layer; the secondary hash indexes are built for accelerating point queries. Snowflake [29], which is a cloud-native OLAP database, has utilized its metadata storage to implement an HTAP solution, called Unistore [122], supporting real-time data analytics. PolarDB-IMCI builds an in-memory column index in a disaggregation architecture of PolarDB [23]. Data synchronization is performed by replaying physical redo logs from the shared storage. Cloud-native architectures empower HTAP with high elasticity, availability, and multi-tenancy. However, the data freshness is low due to the philosophy of "log is the database", especially when the log has not been replayed in the storage layer.

4 HTAP TECHNIQUES

As shown in Table 2, we summarize five types of HTAP techniques, including (1) hybrid workload processing; (2) data organization; (3) data synchronization; (4) query optimization; and (5) resource scheduling. It also presents their main methods, key techniques, pros and cons. 6 illustrates an overview of the HTAP techniques. From a top-down perspective, (1) OLTP and OLAP workloads are handled by hybrid workload processing. (2) HTAP resource scheduling dynamically assigns the resources to the hybrid workload. (3) Query optimization relies on row and column stores

TABLE 2
An Overview of HTAP Techniques

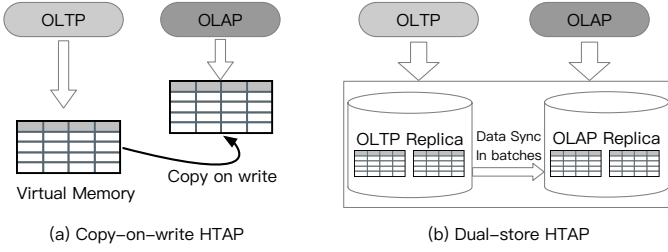| Task Type	Main Method	Key Technique	Pros	Cons
Hybrid Workload Processing	MVCC-based HTAP	Version Chains for OLTP and OLAP [55]	High Freshness	Long Version Chains
	Copy-on-Write based HTAP	Snapshotting OLTP for OLAP [52]	High Freshness	Large Memory Size
	Dual-Store based HTAP	Separated Stores for OLTP and OLAP [57]	High Isolation	High Sync Cost
Data Organization	Primary Row Store with Selected Column Store	Frequency-based Heatmap [92]	Arbitrary Queries	Low Utility
		Cost-Based Linear Programming [16]	High Utility	Single-Table Queries
	Adaptive Hybrid Data Storage	Cost Functions [2], [5], [11], [12]	High Efficiency	Low Utility
		Machine Learning Models [2], [3]	High Utility	Training Overhead
Data Synchronization	In-Memory Delta Merging	Threshold-based Merging [57]	Fast Insertion	Slow Lookup
		Delete table based Merging [60]	Fast Lookup	Slow Insertion
		Dictionary-based Merging [118]	High Efficiency	Low Scalability
	Log-based Delta Merging	Multi-Level Delta Merging [45], [136]	High Scalability	High Merge Cost
		Change Data Capture [21], [83]	High Isolation	High Latency
Query Optimization	Hybrid Row/Column Scan	Cost-based Execution [45], [32], [123]	High Utility	Large Search Space
		Rule-based Execution [57], [60]	High Efficiency	Low Utility
	HTAP Indexing	Parallel Binary Tree [127]	High Throughput	Large Memory Size
		Multi-Version Partitioned B-tree [110]	High Scalability	Low Throughput
	CPU/GPU Acceleration	CPU for OLTP, GPU for OLAP [9], [62]	High AP Throughput	Low Freshness
Resource Scheduling	Freshness-Driven Method	Execution Mode Switching [109]	High Freshness	Low Throughput
	Workload-Driven Method	Rule-Based Resource Assignment [120]	High Throughput	Low Freshness

Fig. 5. Hybrid Processing based on Copy-on-Write and Dual-Store

to execute the query. (4) Data synchronization merges the updates from the row store to the column store. (5) Data organization adaptively organizes the hybrid storage.

4.1 Hybrid Workload Processing

Hybrid workload processing aims to handle the mixed workload under a unified architecture. It mainly consists of three types of techniques: (1) MVCC-based HTAP; (2) copy-on-write-based HTAP; and (3) dual-store-based HTAP.

4.1.1 MVCC-based HTAP

Multi-Version Concurrency Control (MVCC) is the most widely adopted transaction management technique in major relational DBMSs [135]. However, it is challenging to support HTAP solely based on MVCC. This is because HTAP workloads will lead to frequent version traversing and cleaning of stale data versions. Therefore, the main challenge is how to reduce the resource contention and improve the query performance by efficiently traversing and reclaiming the version chains [6]. To make the MVCC-based databases more HTAP friendly, Weaver [53] utilizes the frag skip lists to speed up the version chain lookup by using two pointers for each node, which can accelerate the in-chain version traversing and cross-chain version traversing, respectively; Diva [54] separates the version searching and version cleaning by maintaining a provisional index and performing an interval-based version clearning separately, which results in a better HTAP performance. Note that

MVCC-based systems [135] can implement various isolation levels including *read committed*, *repeatable read*, *snapshot isolation*, and *serializable snapshot isolation*. The concurrency control techniques should be different when it comes to different isolation levels. As a result, HTAP techniques may vary slightly because the delta table should adapt to different isolation levels. As similar techniques of concurrency control in OLTP can be applied to HTAP systems, existing HTAP techniques did not discuss these issues.

Pros and Cons. MVCC-based HTAP processing has a high freshness as the analytical queries can access the latest visible data by traversing the version chains. However, when it comes to the long version chains, it will incur a significant overhead for version traversing and cleaning.

4.1.2 Copy-on-Write based HTAP

This category of techniques [9], [52], [127] relies on in-memory techniques with the Copy-on-Write (CoW) mechanism to support HTAP. The basic idea is to create snapshots for handing OLAP workloads when the main storage encounters a write operation. As shown in Figure 5(a), a snapshot is generated by forking the main OLTP process with a consistent virtual memory.

For transaction processing, the main OLTP process uses a single thread to execute the transactions in a sequential way without locking and latching [52], resulting in lock-free transaction processing. Since the data is maintained in the main memory, the OLTP process can execute transactions at the rate of tens of thousands per second. Moreover, CoW can also handle transactions using multiple cores [127] or multiple threads [52] where each thread processes the sequential transactions over a data partition.

For analytical processing, the forked OLAP process handles the queries with the up-to-date snapshots. Particularly, the OLAP process can utilize the OS-enabled interface to create the updated pages on demand, which only takes several microseconds. With multiple threads, the queries can be performed in parallel against a single snapshot or

multiple snapshots [52]. Besides, the OLAP process can utilize GPUs to accelerate the query processing [9].

Pros and Cons. This type of technique enjoys high freshness as the OLAP process can always analyze the fresh data via snapshotting. However, it suffers from large memory size because each process will create a new snapshot, especially for the write-heavy workload. What is more, it has low isolation due to the shared resources in the same instance.

4.1.3 Dual-Store based HTAP

This type of techniques [10], [34], [45], [62], [57], [77], [83], [84] relies on a dual-store architecture to handle HTAP workloads. Most of HTAP databases introduced in Section 3 employ such an approach by developing a dual-store with both row and columnar format [45], [57], [83], [84], or purely row format [77] and purely column format [34], [62], [86]. As shown in Figure 5(b), a main store is used for handling OLTP workloads, and a secondary store is employed for processing OLAP workloads. The recently updated data is synchronized from the main store to the secondary store in batches periodically and asynchronously.

For transaction processing, the main store relies on MVCC protocols [135] to process the transactions. Specifically, each insert is first written to the log and the row store, then is appended to the delta store. An update creates a new version of a row with a new lifetime of a begin timestamp and an end timestamp, and the older version is marked as a delete row in a delete bitmap.

For analytical processing, queries are performed using column-oriented techniques [1] such as compression-aware processing [59], single-instruction multiple-data (SIMD) instructions [57], and vector processing [86]. Queries can also be accelerated with new hardware such as heterogeneous CPU/GPU processors, Processing-in-memory (PIM) chips [17], [72], and FPGAs [111]. To analyze the fresh data, the query engine traverses the delta data [57], [60], or uses a merge-on-read mechanism [45], [136] that merges the delta data to the main store before analyzing the whole data.

Pros and Cons. Dual-store HTAP processing has high isolation as the workload can be processed with two separated stores using either logical isolation [57], [60], [77], [62], [39] or physical isolation [45], [83]. However, it has a high synchronization cost as the delta data shall be converted and merged to the secondary store.

4.2 Data Organization

Data organization in HTAP databases requires choosing an optimized data layout, e.g., row-wise or column-wise data layout for the mixed workload. As maintaining two copies of data for OLTP and OLAP workloads is costly, existing methods seek to strike a trade-off between the throughput and storage cost. They fall into two categories of approaches: primary row store with selected column store and adaptive hybrid data storage.

4.2.1 Primary Row Store with Selected Column Store

The first category of methods persists the transaction data in the row store and chooses part of the attributes to be included in the column store. The major objective is to select the beneficial columns (i.e., having high utility of query
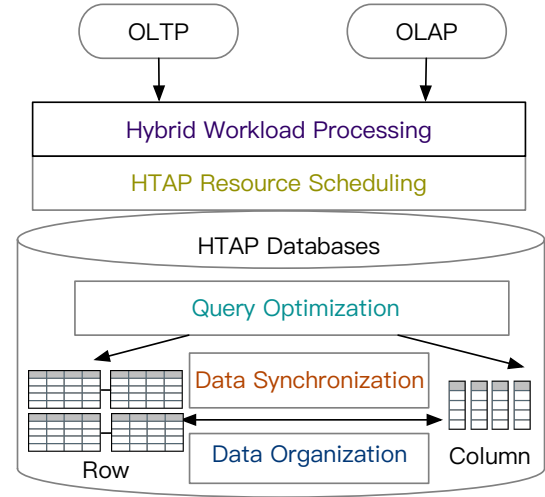

Fig. 6. The Key Techniques of HTAP Databases

acceleration and low update cost) into the memory from the primary row store. With such an approach, OLTP workloads are handled by the row store, and OLAP workload can be accelerated by in-memory columnar scan. It has two main methods: (1) frequency-based heatmap and (2) cost-based linear programming.

(1) Frequency-Based Heatmap. The heatmap approach [92] selects the columns based on the access frequency from the workload. Particularly, the frequently-accessed columns will be kept in the memory and rarely-accessed columns will be evicted to disk. Basically, it groups the in-memory columns into three clusters based on their access frequencies: hot, candidate, and cold columns. The candidate columns are the columns of interests. If some of them were frequently-accessed, then they would be marked as hot columns, which will be populated from the persistent row store. The columns are marked as cold if they were not touched during a time window, e.g., several days. If the cold columns have been populated, they are compressed and evicted to the persistent columns. As extracting the columns from the row store is expensive, the evicted columns can be loaded back if accessed later.

Pros and Cons. The pros of heatmap approach is that it supports arbitrary queries and is easy to implement. The downside is that it may have a low utility as it only considers the access frequency and neglects the effect of various column combinations concerning different queries.

(2) Cost-Based Linear Programming. Another approach is the cost-based linear programming [16], which formalizes the in-memory column selection as a knapsack-style problem, and then uses an integer linear programming (ILP) method to solve the problem. Given a set of queries, its objective function is a cost function that sums over the scan cost of each query over the involved columns. The goal is to minimize the cost function with a set of columns subject to a given constraint, e.g., the total column size is not greater than a memory budget.

Pros and Cons. The ILP method can have high utility because the selected columns can reduce the cost of the queries. However, it is unclear how the cost functions can estimate the cost of complex queries that involve multiple tables and complex operations.

4.2.2 Adaptive Hybrid Data Storage

This line of works [2], [5], [11], [12] organizes the data adaptively based on the given workloads and designed cost functions. Such approaches adopt a fine-grained hybrid storage scheme. For example, H2O [5] supports three storage layouts: columnar layout, row-wise layout, and column groups (i.e., width-varying vertical partitions of the tables). Given a query workload, it evaluates the cost of different storage schemes (including processing cost and conversion cost), and then selects an optimal storage solution. Casper [12] selects an optimal layout of columns based on the mixed workload of read-only queries and updates. It considers various column features, including the number of partitions, the size and range of each partition, the sorting method (i.e., sorted or unsorted), the updating method (e.g., in-place or out-of-place), and the cache size. By evaluating the cost of each scheme based on the defined cost function, it obtains a solution that minimizes the cost with the SLA-aware constraints. Peloton [11] uses a flexible schema to logically divide a relational table into different tiles, where each tile is a sliced data block with vertical and horizontal partitions. With a clustering method, it physically materializes the tiles on disk based on the workload. Beyond the cost-based data organization in a standalone system, Proteus [2] leverages machine learning methods to select the optimal mixed row/column storage based on a distributed architecture. It considers a larger design space for the storage, including data format (row-wise or column-wise), data placement (partial or full copy on a node), data compression, and data tiering (memory or disk). Then, it characterizes the workload concerning the storage layout. Finally, it selects a generated storage schema through a learned cost model. Its follow-up work, Tiresias [3], further extends Proteus to support automatic indexing.

Pros and Cons. The adaptive hybrid data layout has a lower storage cost and a higher throughput. However, such methods have two main drawbacks. First, the hybrid storage increases the system's complexity in query processing as many execution rules need to be re-implemented. Second, the transaction processing over hybrid storage leads to frequent random accesses, resulting in multiple disk I/Os.

4.3 Data Synchronization

As data resides in multiple replicas, efficiently synchronizing the latest transaction data, i.e., the deltas, to the read replicas is required. In addition, since some read replicas adopt a columnar format, it is important to have a tailored method to merge the deltas to the column store efficiently. To address such a problem, two kinds of synchronization methods are proposed. Namely, in-memory delta merging and log-based delta merging.

4.3.1 In-Memory Delta Merging

This type of technique synchronizes the data between the row store and column store using the in-memory delta merging. Particularly, there are three key techniques: (1) threshold-based merging; (2) delete-table merging; and (3) dictionary-based merging.

(1) Threshold-Based Merging. This technique periodically merges the delta data to the column store, which has two steps. First, data updates (inserts/updates/deletes) are recorded in a delta table that is normally implemented using a heap table. Second, the delta data will be migrated to the column store when its size reaches a threshold. Particularly, it can use a trickle-based mechanism that constantly migrates the data in the background.

Pros and Cons. This method supports fast insertion as the updates can be inserted into the heap table quickly. However, when the threshold has not been hit, it could slow down the query processing. This is because the analytical query will not only scan the columnar data (i.e., columnar scan) but also traverse the in-memory delta data that has not been merged in order to access the fresh data. Therefore, it has a larger delta traversing overhead since the heap table is unordered, and the analytical query requires a full-table scan to access the heap table.

(2) Delete-Table Merging. This method depicts the delete table-based merging [60], which periodically merges the delta data to the column store based on a delete table; the delta store is an index-organized table (i.e., B-tree) that maintains the latest transaction data. The delete table is a bitmap that holds the row IDs (RIDs), where each one indicates a row's location in the column store. The delta merging consists of two phases. In the first phase, it assigns a RID to each row from the delta store and inserts it into the delete table. Then, it transforms the row-wise delta data to the column store with the RIDs that are hidden by the delete table. In the second phase, it removes the RIDs from the delete table, and truncates the data in the delta store. Note that the first phase and the second phase are committed as a transaction, respectively. This is mainly for ensuring the data consistency.

Pros and Cons. Delete-table merging supports fast lookup as the delta data is indexed. However, inserting the data has additional overhead due to the data insertion to the index and deleting the table.

(3) Dictionary-Based Merging. This method shows the dictionary-based merging [118], which organizes the delta data in a columnar format and merges the delta to the primary column store based on dictionaries. Particularly, it organizes the delta column by column, and maintains a dictionary with a data vector for each column. The delta merging also consists of two phases. Firstly, new data is merged to a delta column store with a local dictionary and a data vector. Secondly, the local dictionaries are merged into a primary column store with a global dictionary and a global data vector.

Pros and Cons. This method has high efficiency as both the delta store and primary store are organized in a columnar format, and data is indexed using dictionaries. However, as each delta column is indexed by a local dictionary, the volume of delta data grows drastically when the number of data updates increases.

4.3.2 Log-based Delta Merging

This type of technique records the delta data in the change log, and then synchronizes the data between the row store and column store with log shipping and replaying. There are two kinds of log-based merging techniques: (1) multi-level delta merging; and (2) change data capture mechanism;

(1) Multi-Level Delta Merging. This method [45], [136] merges the deltas in the memory level and the deltas in the disk level. It contains four levels from a top-down view: B+tree, memory level, delta space, and persistent storage. Firstly, data manipulation language (DML) operations, such as insert, delete, and update operations, are inserted into a B+ tree after committing the write-ahead log. Secondly, the write operations of a batch are appended to a small delta in the memory, thus the merging process can be performed in batches. Thirdly, the small deltas are compacted and merged into larger delta files in the disk. These small deltas are merged together in the order in which they were written, so these delta files are unordered. Nevertheless, the multi-version, duplicate, and rolled-back records will be removed in the process of merging. Lastly, the unordered delta files will be periodically merged into the persistent storage in a columnar format. Particularly, the persistent storage organizes the data with the ordered chunks, where each one covers a part of the range of the data. Since the delta files are out of order, the merge operation will produce a large overhead. Therefore, it will locate the data by searching over the B+ tree, and then merge the data with the order.

Pros and Cons. Multi-level delta merging has high scalability as the delta data is organized in multiple stages with partitioned files. However, it has a high merging cost due to the large I/O overhead.

(2) Change Data Capture (CDC). CDC is another merging mechanism [21], [61], [83], which monitors the data updates in the change log, and replicates the change log to the analytical store asynchronously. Generally, it treats the log as the first citizen, and then migrates the valid log records to the column store.

Pros and Cons. The CDC mechanism has a high-performance isolation as the OLTP and OLAP workloads are physically isolated in different nodes or systems. However, it incurs high latency due to the log shipping and replaying.

4.4 Query Optimization

We summarize three types of query optimization techniques: (1) hybrid row/column scan [45], [60]; (2) multi-version indexing [127], [110] and (3) CPU/GPU acceleration [62], [9]. Next, we dive into each type of technique.

4.4.1 Hybrid Row/Column Scan

In the HTAP databases, a complex query can be routed to perform against either the row store or the column store. We call such an execution mode as hybrid row/column scan [45], [60], [92]. Moreover, a query can also be executed in a fine-grained way such that part of the operators are processed in the column store, and the rest of the operators are processed in the row store, and finally the results are combined. For instance, the short-range or point queries can be performed using B+ tree indexes in a row store; column scans and complex aggregations can be processed using SIMD scans in a column store; the query coordinator merges the results from both execution engines to the final results. Suppose a SQL query finds the license and color of the vehicles registered in Beijing as follows:

```
SELECT V.license, V.color
FROM Register R, Vehicle V
```

WHERE R.VID=V.ID and R.place ="BJ"

Such an SQL query contains a two-way join between the Register and Vehicle tables with an equality predicate on the place field. The logical plan is separated into a hybrid plan, which relies on a B+ tree to search for the qualified records in the Register table, then joins their VIDs with the IDs of the Vehicle table in the column store, finally it returns the results by projecting the columns of license and color in the Vehicle table. Hence, the query execution can benefit from both the index scan in the row store and the columnar scan.

Various interfaces for hybrid scan [39], [57], [60], [45] have been developed. For instance, Oracle [57] can create an in-memory columnar table by altering the table with "INMEMORY" keyword, then any given SQL query can be executed with a rule-based hybrid scan. SQL Server [60] supports the hybrid scan by building a "COLUMNSTORE INDEX" over the target attributes or tables, and then it accelerates the queries with cost-based columnar scans. AlloyDB [39] performs a cost-based hybrid scan in the operator granularity after enabling its columnar engine. TiDB [45] allows for creating n columnar replicas by altering a table T with "SET TIFLASH REPLICA n", then the queries can be executed using distributed columnar scans. It also supports the hint-based hybrid scan, which can force the access paths using hints. Recall the above SQL query. If a hint "read_from_storage(TIKV[Register], TIFLASH[Vehicle])" is added to the SQL query, then the Register table will be probed from the TIKV row store and the Vehicle table will be scanned from the TIFLASH column store.

The key to hybrid row/column scans is to determine whether a query or an operator should be executed against the row or column store. However, it is not always straightforward to generate an optimal plan for a more complex query to which the plan space is large. Existing methods are mainly grouped into two types: (1) rule-based execution; and (2) cost-based execution.

(1) Rule-based Execution. This type of method utilizes heuristic rules to execute the queries. They rely on two rules of thumb, namely, (i) the columnar scan is more efficient than the index scan and row scan; and (ii) the index scan is more efficient than the row scan. For instance, Oracle [92] performs the hybrid scan by following a "column first, row later" principle: if some columns do not exist in the column store, it scans them in the row store, and finally merges the results from the columnar scan and row scan.

Pros and Cons. The rule-based methods have a high efficiency because of the short planning time by the rules of thumb. However, they may miss the optimal plan as they do not explore the global plan space. For example, an index row scan may be more efficient than a columnar scan, depending on the cost of the specific plans. Therefore, it is preferable to consider the cost of candidate plans as well.

(2) Cost-based Execution. This type of methods [45], [60], [102] selects the access path by comparing the cost of the candidate execution paths. For instance, TiDB [45] builds a cost model among the columnar scan, row scan, and index scan, and it selects the access path with the minimum cost. Unfortunately, the cost model considers only scan operations. PolarDB-IMCI [134] utilizes a threshold-based cost model to select either the row scan or columnar scan.

It relies on the row-based cost model, and if a query cost is beyond a pre-defined threshold, the query is routed to the column store. Metis [123] is a recent work that can generate HTAP-aware hybrid plans by considering the cost of data updates and data synchronization. Its core contribution is a new cost model that considers the delta scan, columnar scan, index scan, and row scan, which can guide the access path selection.

Pros and Cons. This method can produce query plans of high quality. However, existing cost functions are based on independence and uniform distribution assumptions, so the estimation may be inaccurate when these assumptions do not hold. Besides, existing works lack a global and comprehensive cost model for HTAP workloads.

4.4.2 Multi-Version Indexing

Multi-version indexing aims to accelerate HTAP through new indexing methods. Two representatives are Parallel Binary Tree (P-Tree) [127] and Multi-Version Partitioned B-Tree (MV-PBT) [110].

The main idea of P-Tree is to replicate the data paths involved in the latest transaction on the balanced binary tree, and it leverages multi-core processors to operate on the data concurrently. Furthermore, P-Tree implements the snapshot isolation level, so queries can also access snapshot data visible on the index at the same time. The read operation obtains the pointer of the root node, and then follows the index to find the visible versions of data on the path; the update operation will create a new version of the root node, then copies all relevant paths and updates the target node; both read and update operations can be completed in O(log n) time. P-Tree also supports nested mode across multiple tables, so queries can access cross-table data through indexes without joins.

MV-PBT utilizes a multi-version partitioned B-Tree to index the updated data versions. The motivation is that there could be long version chains for an MVCC-based HTAP database, so efficient indexing for the version chains is required. The main challenge is how to efficiently index multi-version data of the same data and support fast queries of the latest visible version data. MV-PBT is based on the partitioned B-Tree that is divided based on the specified key, and each partition has the same search key. All updates of the transaction will be written to the memory buffer; when the buffer is full, the data will be persisted to the corresponding partitioned B-Tree.

Pros and Cons. P-Tree has a high efficiency as the data can be read and updated concurrently. However, the downside is that it consumes large memory size and CPU resources as a single update leads to a copied path. Since different versions of the same data have been indexed into the partition B-Tree, it supports a fast search of visible versions of the data. Compared to the P-Tree, MV-PBT has a higher scalability as the data is indexed using a disk-based B-tree. However, it has a low throughput due to the row-based query processing.

4.4.3 CPU/GPU Acceleration

The heterogeneous integrated processor of CPU/GPU is also an important technology for HTAP query optimization [9], [62], [138]. This type of technique utilizes the task-parallel nature of CPUs and the data-parallel nature of GPUs for handling OLTP and OLAP, respectively. Existing CPU/GPU methods for HTAP are based on a dual store to isolate the workload execution of OLTP and OLAP workloads. That is, using CPU and transactional store for OLTP and using GPU and analytical store for OLAP. Therefore, the kind of technique is also called heterogeneous HTAP (H^2TAP) [9]. Particularly, the HTAP workload is classified into OLAP workload and OLTP workload; OLAP is executed on the analytical store through the GPU, and OLTP is processed on the transactional store with CPU cores; the data updated by the transactional storage can be synchronized to the analytical storage in batches. Ideally, the analytical store will organize the data in a columnar format, and the transaction store will use a row-based format. However, existing methods only adopt purely column-based format [9], [62] due to the consideration of the engineering complexity. Therefore, this approach favors high OLAP throughput but has a low OLTP throughput. It is worthwhile to mention that the architecture and processing paradigm of H^2TAP has many variants and can be changed for different types of workloads accordingly. For instance, some CPU cores can also be deployed in the GPU chips to accelerate the short queries in parallel [9]. Besides, recent work shows that transactions can be accelerated using GPU [14], and read-only queries can be executed in both CPU and GPU with a proper data placement strategy [138].

Pros and Cons. The H^2TAP method favors high analytical throughput because queries can be accelerated by GPU or hybrid scan of CPU/GPU. However, it suffers from the low freshness issue due to the low network bandwidth of PCIe between CPU and GPU.

4.5 Resource Scheduling

HTAP databases need to support the efficient execution of OLTP and OLAP workloads simultaneously, but the performance degradation could be remarkable due to the data synchronization and resource contention. Besides, data freshness is another concern if OLAP cannot read the latest updated data. Existing methods include freshness-driven scheduling and workload-driven scheduling.

4.5.1 Freshness-Driven Scheduling

This method [109] switches the execution modes based on a freshness threshold. Each execution mode adopts a particular strategy for resource allocation and data exchange. For instance, the scheduler controls the execution of OLTP and OLAP in isolation for high throughput, and then periodically synchronizes the data. Once the data freshness becomes low, it switches to an execution mode where OLTP and OLAP share the same copy of data such that the queries can access the fresh data directly.

Such a method works by varying three execution modes: (S1) Co-located execution with the OLTP instance and OLTP instance, where the OLTP instance handles transactions and can access the fresh data with the copy-on-write mechanism; (S2) Isolated execution with the OLTP instance and OLAP instance, where OLTP instance handles transactions and OLAP instance processes the queries, and the delta data in OLTP instance is periodically synchronized to the OLAP

instance; (3) hybrid execution with two OLTP instances and the OLAP instance, where OLAP queries need to analyze the base data in the OLAP instance and the delta data in OLTP instance simultaneously. For resource scheduling, the system executes S2 mode by default, which favors high-performance isolation. When the data freshness is less than the given threshold specified in the service-level agreement (SLA), it can jump to S1 or S3. There is a trade-off among S1, S2, and S3: S1 can analyze the fresh data immediately, but the analytical capacity is limited; S3 can analyze the fresh data with more CPU and storage resources, but it needs to access two instances for query processing. S2 handles the hybrid workloads with better isolation, but the data freshness is lower due to the latency of data exchange.

Pros and Cons. The freshness-driven scheduling strikes a good balance between performance and freshness. However, the system performance may fluctuate due to the frequent mode switching. Therefore, it might be helpful to design some mechanisms for lazy switching.

4.5.2 Workload-Driven Scheduling

This kind of methods [104], [105], [120] dynamically schedules resources such as CPU, shared cache, and memory bandwidth, by monitoring the execution of the mixed workload. Since the access patterns of OLTP and OLAP workloads are different, the resource scheduling needs to adapt to their performance characteristics.

For CPU resources, workload-driven scheduling adjusts the parallelism threads of OLTP and OLAP tasks. The initial number of threads of OLTP is set to the number of CPU cores, and the number of threads of OLAP is set to an average parallelism based on the history statistic. The scheduling method adaptively adjusts the number of threads based on the performance of executed workloads. For example, when CPU resource is saturated by OLAP threads, the task scheduler can decrease the parallelism of OLAP while enlarging the OLTP threads. When the monitoring process (e.g., a watchdog [104]) detects a blocked transaction thread, it will join it to a blocking queue and will try to restart the blocked task. For shared cache and memory bandwidth resources, it can also be dynamically adjusted by observing the workload execution [120]. For example, when the OLAP throughput decreases drastically in a hybrid execution, it indicates that the OLTP execution affects the OLAP execution. Hence, more shared cache, e.g., LLC cache, can be assigned to the OLAP instance. In addition, when the OLAP throughput drops due to the synchronization process, it is necessary to allocate more resources to the synchronization process.

Pros and Cons. This method has high throughput as it regards the throughput metric as the first-class citizen. However, it has a low freshness as it does not consider the freshness metric at all.

5 HTAP BENCHMARKS

Table 3 summarizes eight state-of-the-art HTAP benchmarks, including five end-to-end benchmarks and three micro-benchmarks.

5.1 CH-Benchmark

CH-benchmark [26], a.k.a., TPC-CH [36], is a widely-used end-to-end HTAP benchmark that combines two classical TPC benchmarks, i.e., TPC-C [130] for benchmarking transactional processing systems, and TPC-H [131] for benchmarking analytical reporting systems.

(1) **Data Schema**. CH-benchmark unifies the schema of TPC-C and TPC-H in an application domain of retail business, simulating the behaviors of wholesale suppliers that process the customers' orders and analyze the fresh sales data simultaneously. Specifically, it combines TPC-C's nine tables and TPC-H's eight tables to a schema of 12 tables by merging the overlapping tables from TPC-H (i.e., customer, orders, lineitem, part) and by removing its partsupp table. It has adjusted the scaling model for data generation based on the number of warehouses.

(2) **Workloads** Since the schema of TPC-C has not changed, the CH-benchmark preserves all the five transactions of TPC-C for OLTP. For OLAP workloads, it has made several modifications on TPC-H's. First, it has adjusted the tables' names and join keys of the original 22 queries. Second, it has reduced the arithmetic operations in the queries. Third, it has removed the refresh function of TPC-H as TPC-C has included the operations of data updates.

(3) **Execution Rule**. The execution rule involves two steps: (i) it first executes n streams of OLTP workloads and m streams of OLAP workloads in isolation, then (ii) it performs the mixed workloads with the same number of streams in parallel. The main purpose is to evaluate how HTAP systems can handle the interference of two types of workloads by comparing the performance between the isolation mode and the hybrid mode.

(4) **Performance Metrics**. CH-benchmark has a reference-based metric to measure performance by combining the metrics of tpmC and QphH. Particularly, tpmC is the number of transactions processed by the system per minute, and QphH is the number of queries handled by the system per hour. Two reference-based metrics are as follows:

$$M(OLTP) = \frac{tpmC}{QphH}@tpmC \tag{1}$$

$$M(OLAP) = \frac{tpmC}{QphH}@QphH \tag{2}$$

where both metrics consist of two parts: the former part measures a ratio between tpmC and QphH, and the latter part is the referenced primary metric, i.e., tpmC for M(OLTP) and QphH for M(OLAP). Suppose M(OLTP) is used, and the execution rule follows the isolation-and-hybrid execution mode. For the isolated/sequential execution, M(OLTP) is 2.5@5000 tpmC. Meanwhile, for the hybrid execution, M(OLTP) becomes 3@5500 tpmC, indicating the OLTP throughput is increased with hybrid execution.

5.2 HTAPBench

HTAPBench [25] also combines TPC-C and TPC-H. It adopts the same data schema and the same hybrid workloads in CH-benchmark. Its main contribution is two-fold. First, it curates the parameters of OLAP queries with the concept of a dynamic window. Second, it contains an execution rule for

TABLE 3
An Overview of HTAP Benchmarks

Benchmark	Type	Domain/Task	Benchmark/Workload	Execution Rule	Metrics
CH-Benchmark [26]	End-to-End Benchmark	Retail Business	TPC-C+ TPC-H	Isolated Execution+ Hybrid Execution	Referenced Throughput
HTAPBench [25]	End-to-End Benchmark	Retail Business	TPC-C+ TPC-H	Fixed OLTP workers+ varied OLAP workers	Referenced Throughput
OLxPBench [51]	End-to-End Benchmarks	Retail Business	TPC-C+9 queries+5 txns	Hybrid Execution	Throughput
		Banking	Small bank +4 queries+6 txns	Hybrid Execution	Throughput
		Telecom	TATP+5 queries + 6 txns	Hybrid Execution	Throughput
HATtrick [80]	End-to-End Benchmark	Retail Business	SSB + 3 txns	Hybrid Execution	2D Throughput Freshness
HyBench [141]	End-to-End Benchmark	Online Finance	18 txns+13 queries +12 mixed operations	Hybrid Execution	F-Score H-Score
ADAPT [11]	Micro-benchmark	Data Organization	1 insert + 4 select queries	Hybrid Execution	Throughput
HAP [12]	Micro-benchmark	Data Organization	6 CRUD queries	Hybrid Execution	Throughput
mOLxPBench [49]	Micro-benchmark	Data Synchronization	6 CRUD queries	Hybrid Execution	Tail Latency

targeting a fixed OLTP throughput by adaptively launching the OLAP workers.

(1) **Workloads**. HTAPBench employs the mixed workload of TPC-C and TPC-H. However, since TPC-H adopts the fixed parameters for the queries, the performance results across runs are often incomparable in the hybrid execution mode. This is mainly because TPC-C's workload may update different parts of data during execution and change the data distribution. To address such a problem, HTAPBench leverages the dynamic query generator, which utilizes a density consultant to generate the parameters for the analytical queries to ensure the same query selectivity during execution. Particularly, it curates the parameters based on the DATE field to slide the time windows of queries for accessing the newly inserted data.

(2) **Execution Rules**. Instead of scaling the query streams of OLTP and OLAP simultaneously, HTAPBench regards the OLTP throughput as the first citizen. That is, it targets a fixed OLTP throughput and evaluates how well the SUT systems process the OLAP workloads while keeping the OLTP throughput and scaling the OLAP workers. Specifically, according to the TPC-C's specification that defines the maximum of 1.286 tpmC per client, it computes the number of clients and warehouses for a target tpmC, then populates the databases and launches the OLTP workload execution. Then, it gradually starts the OLAP workers based on a client balancer, which monitors the throughput gap between the actual tpmC and the target tpmC, and periodically decides whether or not to start an additional OLAP worker. Namely, if the current gap is larger than a margin and the resource is not saturated, then a new OLAP worker will be started. Otherwise, the balancer keeps monitoring.

(3) **Performance Metrics**. Based on the execution rule, HTAPBench proposed a OLTP-oriented metric for measuring the HTAP performance as follows:

$$QpHpW = \frac{QphH}{\#OLAPworkers}@tpmC \qquad (3)$$

where QpHpW denotes the number of processed Queries per Hour per Worker. The former part is a ratio between TPC-H's metric and the total number of OLAP workers. The latter part is the target tpmC.

5.3 OLxPBench

OLxPBench [51] is a composite HTAP benchmark suite. It consists of three domain-specific HTAP benchmarks inherited from three established OLTP benchmarks, respectively. Namely, *Subenchmark* is from the TPC-C benchmark [130] at a retail business scenario, *Fibenchmark* is from the Small-Bank [7] benchmark at a bank scenario, and *Tabenchmark* is from the TATP [128] benchmark at a telecom scenario. OLxPBench makes two modifications. First, it contains new analytical queries with the transactions to compose an HTAP workload. Second, it includes analytical transactions that perform real-time queries inside the transactions.

(1) **Data Schema**. The OLxPBench benchmarks have the schema of three original OLTP benchmarks. Particularly, *Subenchmark* preserves nine tables from TPC-C; *Fibenchmark* keeps three tables from SmallBank, i.e., ACCOUNT, SAVING, CHECKING; *Tabenchmark* includes all the five tables from TATP, which simulates a Home Location Register (HLR) database used by a mobile carrier.

(2) **Workloads and Execution Rules**. Unlike CH-benchmark where the analytical queries only operate on the overlapping tables of TPC-C, OLxPBench follows semantic consistency, meaning that both queries and transactions shall cover all the tables. For instance, the history, warehouse, and district tables will never be touched in the queries of CH-benchmark, while OLxPBench contains several queries that analyze the records from these tables. Moreover, OLxPBench includes analytical transactions that are originally proposed by Gartner, which envisioned HTAP transactions that include analytical operations. Specifically, OLxPBench implements such a transaction by adding a real-time query to each transaction in all the three benchmarks. With the new workload, OLxPBench offers three execution modes: (1) a sequential execution mode of transactions and queries. (2) a concurrent execution mode of mixed workloads. and (3) an execution mode of analytical transactions.

5.4 HATtrick

HATtrick [80] is an end-to-end HTAP benchmark, which combines an analytical benchmark, SSB [90], with a transactional component inspired by TPC-C. It has two contributions to the performance metrics. First, it proposes the

metric of the throughput frontier to capture the OLTP and OLAP throughput with a 2D visualization graph. Second, it uses the freshness score to quantify the transaction's recency for a query by the analytical client (A-client).

(1) **Data Schema**. HATtrick modifies the SSB schema from two aspects. First, it adds a HISTORY table and updates the relations of CUSTOMER, SUPPLIER, and PART by adding new attributes that will be used by the transactions. Second, it incorporates a set of entries $FRESHNESS_j$, where each entry stores an integer transaction number TXN-NUM for the i-th transactional client (T-client). TXNNUM is used to calculate the freshness score for the queries.

(2) **Workloads and Execution Rules**. For analytical queries, HATtrick contains all 13 queries of SSB with a modification to return the freshness entries. Each A-client performs a batch of 13 queries recursively with random orders. For transactions, HATtrick defined three transactions based on TPC-C, including two read-write transactions, NEW ORDER and PAYMENT, and a read-only transaction, COUNT ORDERS. Each T-client defined the ratio of transactions with 48%, 48%, and 4%, respectively. Each transaction committed by the T-client j will update the $FRESHNESS_j$ with the transaction ID accordingly. By configuring the number of T-clients and A-clients, HATtrick executes the workloads concurrently.

(3) **Performance Metrics**. HATtrick introduces two metrics: throughput frontier and freshness score. Throughput frontier is a 2D graph where the x-axis represents the throughput of T-clients, and the y-axis depicts the throughput of A-clients. By fixing either the T-clients or A-clients while varying the number of the other types of clients, the throughput frontier can be effectively computed. The plotted throughput frontiers can reflect the relationships between performance characteristics and workload interference. For instance, if the frontier is below the diagonal line of the bounding box, then the hybrid throughput is relatively low, and workload contention is high. If the frontier is close to the bounding box, then the hybrid throughput is high, and the performance isolation is good. With a global clock, HATtrick defined the freshness score f_{A_q} of an analytical query A_q as follows:

$$f_{A_q} = max(0, t^s_{A_q} - t^{fns}_{A_q}) \tag{4}$$

where $t^{fns}_{A_q}$ is the commit time of the first transaction t^{fns} that is unseen by A_q. $t^s_{A_q}$ is the start time of A_q. Intuitively, the smaller the score is, the fresher the data is. The larger the score is, the analyzed data is more stale.

5.5 HyBench

As existing benchmarks [25], [26], [51], [80] heavily rely on traditional OLTP benchmarks (e.g., TPC-C) or OLAP benchmarks (e.g., TPC-H), they fall short of providing representative HTAP data, workload, and metrics.

HyBench [141] is a newly-emerged benchmark for HTAP databases, which features a new data generator, a multi-set workload, and a unified metric. First, it contains a schema by simulating a realistic online finance HTAP scenario, and it provides a data generator based on a time-dependent generation phase and an anomaly generation phase. Regarding the workload, it has three sets of workloads for OLTP,

OLAP, and OLXP, evaluating the performance of transaction processing, analytical processing, and hybrid processing, respectively. To quantify the overall HTAP performance, it proposes a unified metric, H-Score, that combines the performance of OLTP (TPS), OLAP (QPS), and OLXP (XPS) and data freshness.

(1) Data Schema. Its schema is based on an online finance application inspired by the real-world HTAP applications in the field of finance technology (FinTech) [71], [129], [137]. The schema consists of eight tables, including CUSTOMER, COMPANY, SAVINGACCOUNT, CHECKINGACCOUNT, TRANSFER, CHECKING, LOANAPP, and LOAN, simulating the widely-used finance activities such as saving, payment, and loan application. The data generation produces the testing data based on a given scale factor (SF), and the data size grows linearly as the SF increases. Instead of using uniform data generation [80], [33], HyBench leverages a time-dependent data generation to generate data in three time ranges, enabling efficient and realistic data generation. Additionally, it proposes an anomaly generation phase to produce blocked accounts and illegal transactions, which simulate realistic anomalies.

(2) Workloads and Execution Rules. HyBench contains 18 operational transactions, 13 analytical queries, and 12 mixed operations that include six analytical transactions (AT) and six interactive queries (IQ), providing a rich set of workloads for benchmarking HTAP databases. For instance, AT1 makes a transfer while performing the risk controlling by analyzing if the target has any risks; IQ1 finds related transfers for a blocked user on-the-fly. HyBench follows a three-phase execution rule to execute the hybrid workload in a row, including an AP phase, a TP phase, and an XP phase, and then it outputs a unified metric.

(3) Performance Metrics. HyBench proposes two new metrics tailored to HTAP databases, namely, F-Score and H-Score. F-Score is used to measure the data freshness by measuring the timestamp difference between the result set from the OLTP instance and the OLAP instance. Particularly, it is a general method that supports the cases of insert, update, and delete. H-Score is a unified metric that relies on geometric mean to measure the overall HTAP performance. Given the concurrency of TP workers and AP workers (n, m), it is defined as follows:

$$\text{H-Score} = SF * \frac{\sqrt[3]{TPS * QPS * XPS}}{\overline{f_s} + 1} \tag{5}$$

where XPS = ATS+IQS, and ATS and IQS are analytical transactions per second and interactive queries per second, respectively; SF is the used scale factor; $\overline{f_s}$ is the F-Score that is measured in seconds, and it is added with 1 to avoid that the denominator is zero.

5.6 Micro-Benchmarks

Other than the end-to-end benchmarks, there are three synthetic micro-benchmarks [11], [12], [49] that were developed to evaluate the adaptive data layout in HTAP databases. Particularly, ADAPT [11] contains two synthetic tables with 50 and 500 integer attributes, respectively. Each table owns a primary key a_0 and has ten million tuples. The workload operates on either the narrow table or the wide table, i.e.,

the table with 500 attributes. It contains an insert query, three selection queries, and a self-join query. The selection queries project a subset of the attributes $(a_1, \ldots a_p)$ with a filter on the primary key a_0. The self-join query projects a subset of the attributes with a theta-join on two random attributes, i.e., $X.a_i < Y.a_j$. HAP [12] was inspired by ADAPT. For the data schema, it contains a narrow table of 16 columns and a wide table of 160 columns. Regarding the workload, it contains six queries, including a point query with a projection, two aggregation queries with range filters, an insert query, a delete query, and an update query with an equality filter on the primary key. mOLxPBench [49] is a new HTAP microbenchmark, which contains a single ITEM table with 59 attributes and includes five queries and one update. A salient feature is that it can control the rate at which fresh data is generated and the scan range. Therefore, the greater the rate and range are, the more data needs to be synchronized, and the tail latency increases sharply.

6 Open Problems and Challenges

Data Organization for Distributed HTAP Databases. As various HTAP databases are going towards distributed architectures, the data organization poses many new challenges. The first challenge is to decide which columns to keep in memory, which on disk, and on which nodes. The second challenge is to decide whether the data is organized in a row-wise or column-wise or unified format [70], [112]. The third challenge is to decide which compression methods [15] should be used, and in which granularity (table or column or segment [31]). Proteus [2], [3] is a recent representative that employed an offline learning method considering many factors to organize the data. However, the offline learning method needs to be further justified due to the high complexity of model learning and data organization. It might be helpful to address these challenges separately to reduce the complexity. In addition, combining the offline learning with lightweight online learning methods [65], [66], [67], [145], [146], [126] can also mitigate the training overhead.

Query Optimization in HTAP Databases. There are several open problems for HTAP query optimization. The first one is about hybrid scans for analytical queries. As the existing interface has limited functionality for hybrid scans (e.g., data cannot be exchanged between the row and column data), it calls for more flexible and effective methods to generate hybrid plans. The second open problem is the FPGA-enabled HTAP. Recent works [111], [133] have shown some promising results on such a task, but more HTAP workloads need to be explored as many operators have yet to be implemented. Finally, how to incorporate the learned indexes [56], [68] for HTAP is also an open issue.

Holistic Scheduling for HTAP. Existing freshness-driven scheduling [109] relies on a rule-based approach to control the execution modes with different freshness settings. The workload-driven approaches [118], [120] only adjust the number of OLTP and OLAP threads but do not consider the freshness. Consequently, there still lacks a holistic scheduling method that can orchestrate the workloads, resources, and freshness together. For example, if the current delta store is too cumbersome, some OLAP queries may be scheduled to OLTP instances for high freshness. Therefore, it is preferable to develop a holistic scheduling method that not only captures the workload pattern for better performance, but also satisfies the requirements of data freshness and cost.

HTAP for Multi-Model Data Analytic. As other data models such as graph models are calling for the support of HTAP [48], it is also promising to enable HTAP for multi-model data analytics. Gart [117] is a pioneering work that supports the HTAP over row store, and then synchronizes the delta logs to the graph store. Nevertheless, there still remain many opportunities, such as supporting HTAP-aware multi-model queries [143], [144], [41] and supporting other data models like semi-structured document [142].

Serving atop HTAP. Building data services on top of the HTAP databases is an interesting direction to enable freshness-driven data serving including real-time machine learning (ML) based data analytics. Such a concept is also called HSTAP [50], [79], and how to efficiently and effectively train the ML over the incremental transaction data remains unexplored.

Cloud-Native HTAP Techniques. Cloud-native HTAP techniques [39], [103], [122] are just unfolding and bring many new challenges for HTAP. First, since the compute and storage are disaggregated, it is challenging to deliver a high data freshness if the log in the storage layer has not been replayed. Thus, how to guarantee the data freshness with a low replication latency for the compute layer is an open problem. Second, it is challenging to schedule the OLAP and OLTP workloads to meet various requirements of multiple tenants (e.g., throughput or freshness or cost). Hence, it calls for SLA-aware HTAP scheduling methods. Third, as serverless computing [85], [97], [115] is becoming prevalent, it is still an open problem to utilize serverless computing to handle the HTAP workloads.

7 Conclusion

In this paper, we review the recent advancement of HTAP databases. We classify the state-of-the-art HTAP databases according to four storage architectures. We compare their pros and cons, summarize the challenges and opportunities, and discuss the suitable applications. Since "one HTAP database cannot fit all", we recommend choosing different HTAP architectures to meet the requirements of specific applications. Furthermore, we present their key techniques regarding hybrid workload processing, data organization, data synchronization, query optimization, and resource scheduling, we then summarize the pros and cons of various techniques. We also compare and summarize the state-of-the-art HTAP benchmarks, concerning domain applications, data schema, workload, execution rules, and metrics. Finally, we discuss the research challenges and open problems for HTAP techniques.

References

[1] D. Abadi, P. A. Boncz, S. Harizopoulos, S. Idreos, and S. Madden. The design and implementation of modern column-oriented database systems. *Found. Trends Databases*, 5(3):197–280, 2013.

[2] M. Abebe, H. Lazu, and K. Daudjee. Proteus: Autonomous adaptive storage for mixed workloads. In *SIGMOD*, pages 700–714. ACM, 2022.

[3] M. Abebe, H. Lazu, and K. Daudjee. Tiresias: Enabling predictive autonomous storage and indexing. *Proceedings of the VLDB Endowment*, 15(11):3126–3136, 2022.

[4] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In *VLDB*, pages 169–180. Morgan Kaufmann, 2001.

[5] I. Alagiannis, S. Idreos, and A. Ailamaki. H2o: a hands-free adaptive store. In *SIGMOD*, pages 1103–1114, 2014.

[6] A. Alhomssi and V. Leis. Scalable and robust snapshot isolation for high-performance storage engines. *Proceedings of the VLDB Endowment*, 16(6):1426–1438, 2023.

[7] M. Alomari, M. J. Cahill, A. D. Fekete, and U. Röhm. The cost of serializability on platforms that use snapshot isolation. In *ICDE*, pages 576–585. IEEE Computer Society, 2008.

[8] Apache Arrow. https://arrow.apache.org/, 2022.

[9] R. Appuswamy, M. Karpathiotakis, D. Porobic, and A. Ailamaki. The Case For Heterogeneous HTAP. In *CIDR*, 2017.

[10] V. Arora, F. Nawab, D. Agrawal, and A. El Abbadi. Janus: A hybrid scalable multi-representation cloud datastore. *TKDE*, 30(4):689–702, 2017.

[11] J. Arulraj, A. Pavlo, and P. Menon. Bridging the Archipelago between Row-stores and Column-stores for Hybrid Workloads. In *SIGMOD*, pages 583–598, 2016.

[12] M. Athanassoulis, K. S. Bøgh, and S. Idreos. Optimal Column Layout for Hybrid Workloads. *Proceedings of the VLDB Endowment*, 12(13):2393–2407, 2019.

[13] R. Barber, C. Garcia-Arellano, R. Grosman, R. Mueller, V. Raman, R. Sidle, M. Spilchen, A. J. Storm, Y. Tian, P. Tözün, et al. Evolving databases for new-gen big data applications. In *CIDR*, 2017.

[14] N. Boeschen and C. Binnig. GaccO - A GPU-accelerated OLTP DBMS. In *SIGMOD*, pages 1003–1016. ACM, 2022.

[15] M. Boissier. Robust and budget-constrained encoding configurations for in-memory database systems. *Proceedings of the VLDB Endowment*, 15(4):780–793, 2021.

[16] M. Boissier, R. Schlosser, and M. Uflacker. Hybrid data layouts for tiered HTAP databases with pareto-optimal data placements. In *ICDE*, pages 209–220. IEEE, 2018.

[17] A. Boroumand, S. Ghose, G. F. Oliveira, and O. Mutlu. Enabling high-performance and energy-efficient hybrid transactional/analytical databases with hardware/software cooperation. In *ICDE*. IEEE, 2022.

[18] D. Borthakur et al. Hdfs architecture guide. *Hadoop apache project*, 53(1-13):2, 2008.

[19] M. Bouzeghoub. A Framework for Analysis of Data Freshness. In *International workshop on Information quality in information systems*, pages 59–67, 2004.

[20] M. Butrovich, W. S. Lim, L. Ma, J. Rollinson, W. Zhang, Y. Xia, and A. Pavlo. Tastes great! less filling! high performance and accurate training data collection for self-driving database management systems. In *SIGMOD*, pages 617–630. ACM, 2022.

[21] D. Butterstein, D. Martin, K. Stolze, F. Beier, J. Zhong, and L. Wang. Replication at the speed of change - a fast, scalable replication solution for near real-time HTAP processing. *Proceedings of the VLDB Endowment*, 13(12):3245–3257, 2020.

[22] W. Cao, F. Li, G. Huang, et al. Polardb-x: An elastic distributed relational database for cloud-native applications. In *ICDE*, pages 2859–2872. IEEE, 2022.

[23] W. Cao, Y. Zhang, X. Yang, et al. Polardb serverless: A cloud native database for disaggregated data centers. In *SIGMOD*, pages 2477–2489. ACM, 2021.

[24] J. Chen, Y. Ding, Y. Liu, et al. Bytehtap: Bytedance's HTAP system with high data freshness and strong data consistency. *Proceedings of the VLDB Endowment*, 15(12):3411–3424, 2022.

[25] F. Coelho, J. Paulo, R. Vilaça, J. Pereira, and R. Oliveira. HTAP-Bench: Hybrid Transactional and Analytical Processing Benchmark. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 293–304, 2017.

[26] R. Cole, F. Funke, L. Giakoumakis, et al. The Mixed Workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, pages 1–6, 2011.

[27] J. C. Corbett, J. Dean, M. Epstein, et al. Spanner: Google's globally-distributed database. In *OSDI*, pages 251–264. USENIX Association, 2012.

[28] U. Cubukcu, O. Erdogan, S. Pathak, S. Sannakkayala, and M. Slot. Citus: Distributed postgresql for data-intensive applications. In *SIGMOD*, pages 2490–2502, 2021.

[29] B. Dageville, T. Cruanes, M. Zukowski, et al. The snowflake elastic data warehouse. In *SIGMOD*, pages 215–226. ACM, 2016.

[30] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL Server's Memory-Optimized OLTP Engine. In *SIGMOD*, pages 1243–1254, 2013.

[31] M. Dreseler, J. Kossmann, M. Boissier, S. Klauck, M. Uflacker, and H. Plattner. Hyrise re-engineered: An extensible database system for research in relational in-memory data management. In *EDBT*, pages 313–324, 2019.

[32] A. Dziedzic, J. Wang, S. Das, B. Ding, V. R. Narasayya, and M. Syamala. Columnstore and B+ tree-Are Hybrid Physical Designs Important? In *SIGMOD*, pages 177–190, 2018.

[33] J. Fan, T. Liu, G. Li, J. Chen, Y. Shen, and X. Du. Relational data synthesis using generative adversarial networks: A design space exploration. *arXiv preprint arXiv:2008.12763*, 2020.

[34] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA Database–An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[35] D. Feinberg. Setting the Record Straight – HTAP OPDBMS, 2018.

[36] F. Funke, A. Kemper, and T. Neumann. Benchmarking hybrid oltp&olap database systems. In *BTW*, volume P-180 of *LNI*, pages 390–409. GI, 2011.

[37] Geode. Performance is key. Consistency is a must, 2022.

[38] A. K. Goel, J. Pound, N. Auch, P. Bumbulis, S. MacLean, F. Färber, F. Gropengiesser, C. Mathis, T. Bodner, and W. Lehner. Towards Scalable Real-Time Analytics: An Architecture for Scale-Out of OLxP Workloads. *Proceedings of the VLDB Endowment*, 8(12):1716–1727, 2015.

[39] Google AlloyDB. AlloyDB for PostgreSQL, 2024.

[40] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. Hyrise: a main memory hybrid storage engine. *Proceedings of the VLDB Endowment*, 4(2):105–116, 2010.

[41] Q. Guo, C. Zhang, S. Zhang, and J. Lu. Multi-model query languages: taming the variety of big data. *Distributed and Parallel Databases*, 42(1):31–71, 2024.

[42] HBase. Apache HBase Reference Guide, 2016.

[43] D. Hieber and G. Grambow. Hybrid transactional and analytical processing databases-state of research and production usage.

[44] D. Hieber and G. Grambow. Hybrid transactional and analytical processing databases: A systematic literature review. In *DATA ANALYTICS*, pages 90–98, 2020.

[45] D. Huang, Q. Liu, Q. Cui, Z. Fang, X. Ma, F. Xu, L. Shen, L. Tang, Y. Zhou, M. Huang, et al. TiDB: A Raft-based HTAP Database. *Proceedings of the VLDB Endowment*, 13(12):3072–3084, 2020.

[46] N. Jeba and S. Rathi. Effective data management and real-time analytics in internet of things. *Int. J. Cloud Comput.*, 10(1/2):112–128, 2021.

[47] H. Jiang, C. Liu, J. Paparrizos, A. A. Chien, J. Ma, and A. J. Elmore. Good to the last bit: Data-driven encoding with codecdb. In *SIGMOD*, pages 843–856. ACM, 2021.

[48] M. A. Jibril, A. Baumstark, and K.-U. Sattler. Adaptive update handling for graph htap. *Distributed and Parallel Databases*, pages 1–27, 2023.

[49] G. Kang, S. Chen, and H. Li. Benchmarking htap databases for performance isolation and real-time analytics. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, 3(2):100122, 2023.

[50] G. Kang, L. Wang, S. Chen, and J. Zhan. Nhtapdb: Native htap databases. *arXiv preprint arXiv:2302.09927*, 2023.

[51] G. Kang, L. Wang, W. Gao, F. Tang, and J. Zhan. Olxpbench: Real-time, semantically consistent, and domain-specific are essential in benchmarking, designing, and implementing HTAP systems. In *ICDE*, pages 1822–1834. IEEE, 2022.

[52] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206. IEEE, 2011.

[53] J. Kim, K. Kim, H. Cho, et al. Rethink the scan in MVCC databases. In *SIGMOD*, pages 938–950. ACM, 2021.

[54] J. Kim, J. Yu, J. Ahn, S. Kang, and H. Jung. Diva: Making MVCC systems htap-friendly. In *SIGMOD*, pages 49–64. ACM, 2022.

[55] K. Kim, T. Wang, R. Johnson, and I. Pandis. ERMIA: fast memory-optimized database system for heterogeneous workloads. In *SIGMOD*, pages 1675–1687. ACM, 2016.

[56] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In G. Das, C. M. Jermaine, and P. A. Bernstein, editors, *SIGMOD*, pages 489–504. ACM, 2018.

[57] T. Lahiri, S. Chavan, M. Colgan, D. Das, A. Ganesh, M. Gleeson, S. Hase, A. Holloway, J. Kamp, T.-H. Lee, et al. Oracle Database In-Memory: A Dual Format In-Memory Database. In *ICDE*, pages 1253–1258. IEEE, 2015.

[58] L. Lamport. Paxos made simple, fast, and byzantine. In A. Bui and H. Fouchal, editors, *OPODIS*, volume 3, pages 7–9. Suger, Saint-Denis, rue Catulienne, France, 2002.

[59] H. Lang, T. Mühlbauer, F. Funke, P. A. Boncz, T. Neumann, and A. Kemper. Data blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation. In *SIGMOD*, pages 311–326. ACM, 2016.

[60] P.-Å. Larson, A. Birka, E. N. Hanson, W. Huang, M. Nowakiewicz, and V. Papadimos. Real-Time Analytical Processing with SQL Server. *VLDB*, 8(12):1740–1751, 2015.

[61] J. Lee, S. Moon, et al. Parallel Replication across Formats in SAP HANA for Scaling Out Mixed OLTP/OLAP workloads. *VLDB*, 10(12):1598–1609, 2017.

[62] R. Lee, M. Zhou, C. Li, S. Hu, J. Teng, D. Li, and X. Zhang. The Art of Balance: A RateupDB Experience of Building a CPU/GPU Hybrid Database Product. *Proceedings of the VLDB Endowment*, 14(12):2999–3013, 2021.

[63] G. Li, H. Dong, and C. Zhang. Cloud databases: New techniques, challenges, and opportunities. *Proceedings of the VLDB Endowment*, 15(12):3758–3761, 2022.

[64] G. Li and C. Zhang. HTAP databases: What is new and what is next. In *SIGMOD*, pages 2483–2488. ACM, 2022.

[65] G. Li and X. Zhou. Machine learning for data management: A system view. In *ICDE*, pages 3198–3201. IEEE, 2022.

[66] G. Li, X. Zhou, and L. Cao. AI Meets Database: AI4DB and DB4AI. In *SIGMOD*, pages 2859–2866, 2021.

[67] G. Li, X. Zhou, and L. Cao. Machine learning for databases. In *AIMLSystems 2021: The First International Conference on AI-ML-Systems, Bangalore India, October 21 - 23, 2021*, pages 28:1–28:2. ACM, 2021.

[68] G. Li, X. Zhou, and L. Cao. Machine learning for databases. *VLDB*, 14(12):3190–3193, 2021.

[69] Q. Li, A. Nourbakhsh, S. Shah, and X. Liu. Real-time novel event detection from social media. In *ICDE*, pages 1129–1139. IEEE Computer Society, 2017.

[70] T. Li, M. Butrovich, A. Ngom, W. S. Lim, W. McKinney, and A. Pavlo. Mainlining databases: Supporting fast transactional workloads on universal columnar data file formats. *Proceedings of the VLDB Endowment*, 14(4):534–546, 2020.

[71] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song. Heterogeneous graph neural networks for malicious account detection. In *CIKM*, pages 2077–2085. ACM, 2018.

[72] Z. Lu, Q. Cao, H. Jiang, Y. Chen, J. Yao, and A. Pan. Fluidkv: Seamlessly bridging the gap between indexing performance and memory-footprint on ultra-fast storage.

[73] C. Luo, P. Tözün, Y. Tian, R. Barber, V. Raman, and R. Sidle. Umzi: Unified multi-zone indexing for large-scale HTAP. In *EDBT*, pages 1–12. OpenProceedings.org, 2019.

[74] Z. Lyu, H. H. Zhang, G. Xiong, G. Guo, H. Wang, J. Chen, A. Praveen, Y. Yang, X. Gao, A. Wang, et al. Greenplum: A hybrid database for transactional and analytical workloads. In *SIGMOD*, pages 2530–2542, 2021.

[75] L. Ma, D. V. Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In *SIGMOD*, pages 631–645. ACM, 2018.

[76] L. Ma, W. Zhang, J. Jiao, W. Wang, M. Butrovich, W. S. Lim, P. Menon, and A. Pavlo. MB2: decomposed behavior modeling for self-driving database management systems. In *SIGMOD*, pages 1248–1261. ACM, 2021.

[77] D. Makreshanski, J. Giceva, C. Barthels, and G. Alonso. Batchdb: Efficient isolated execution of hybrid oltp+ olap workloads for interactive applications. In *SIGMOD*, pages 37–50, 2017.

[78] MariaDB. Deploy an HTAP Server with MariaDB ColumnStore 5.5 and Community Server 10.6, 2021.

[79] MatrixOne. HSTAP architecture. https://docs.matrixorigin.cn, 2024.

[80] E. Milkai, Y. Chronis, K. P. Gaffney, Z. Guo, J. M. Patel, and X. Yu. How good is my HTAP system? In *SIGMOD*, pages 1810–1824. ACM, 2022.

[81] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav. Snappydata: A unified cluster for streaming, transactions and interactice analytics. In *CIDR*, volume 17, pages 8–11, 2017.

[82] T. Mühlbauer, W. Rödiger, et al. Scyper: A hybrid oltp&olap distributed main memory database system for scalable real-time analytics. In *BTW*, pages 499–502, 2013.

[83] MySQL Heatwave. Real-time Analytics for MySQL Database Service, 2024.

[84] M. Nakamura, T. Tabaru, Y. Ujibashi, T. Hashida, M. Kawaba, and L. Harada. Extending postgresql to handle olxp workloads. In *INTECH 2015*, pages 40–44, 2015.

[85] V. R. Narasayya and S. Chaudhuri. Cloud Data Services: Workloads, Architectures and Multi-Tenancy. *Foundations and Trends in Databases*, 10(1):1–107, 2021.

[86] T. Neumann, T. Mühlbauer, and A. Kemper. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *SIGMOD*, pages 677–689, 2015.

[87] Nikita Ivanov. How HTAP Enables Real-Time Banking Services At Scale, 2021.

[88] NoisePage. https://noise.page/, 2022.

[89] D. P. A. Nugroho and H. A. Ismail. In-memory database and memsql.

[90] P. E. O'Neil, E. J. O'Neil, X. Chen, and S. Revilak. The star schema benchmark and augmented fact table indexing. In *TPCTC*, volume 5895 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2009.

[91] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC*, pages 305–319. USENIX Association, 2014.

[92] Oracle 21c. Automating Management of In-Memory Objects.

[93] F. Özcan, Y. Tian, and P. Tözün. Hybrid Transactional/Analytical Processing: A Survey. In *SIGMOD*, pages 1771–1775, 2017.

[94] A. Pavlo, G. Angulo, J. Arulraj, et al. Self-driving database management systems. In *CIDR*. www.cidrdb.org, 2017.

[95] A. Pavlo and M. Aslett. What's really new with newsql? *SIGMOD Rec.*, 45(2):45–55, 2016.

[96] D. Peng and F. Dabek. Large-scale incremental processing using distributed transactions and notifications. In *OSDI*, pages 251–264. USENIX, 2010.

[97] M. Perron, R. C. Fernandez, D. J. DeWitt, and S. Madden. Starling: A scalable query engine on cloud functions. In *SIGMOD*, pages 131–141. ACM, 2020.

[98] M. Pezzini, D. Feinberg, N. Rayner, and R. Edjlali. Hybrid Transaction/Analytical Processing Will Foster Opportunities For Dramatic Business Innovation. *Gartner*, pages 4–20, 2014.

[99] M. Pezzini, D. Feinberg, N. Rayner, and R. Edjlali. Real-time Insights and Decision Making using Hybrid Streaming, In-Memory Computing Analytics and Transaction Processing. 2016.

[100] M. Pezzini, D. Feinberg, N. Rayner, and R. Edjlali. Magic Quadrant for Cloud Database Management Systems. *Gartner (2021, December 13)*, pages 1–37, 2021.

[101] Phoenix. OLTP and operational analytics for Apache Hadoop.

[102] PolarDB. PolarDB HTAP Real-Time Data Analysis Technology Decryption, 2021.

[103] A. Prout, S. Wang, J. Victor, et al. Cloud-Native Transactions and Analytics in SingleStore. In *SIGMOD*, pages 2340–2352, 2022.

[104] I. Psaroudakis, T. Scheuer, N. May, and A. Ailamaki. Task scheduling for highly concurrent analytical and transactional main-memory workloads. In *ADMS*, pages 36–45, 2013.

[105] I. Psaroudakis, F. Wolf, N. May, T. Neumann, A. Böhm, A. Ailamaki, and K.-U. Sattler. Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. In *TPCTC*, pages 97–112. Springer, 2014.

[106] X. Qiu, W. Cen, Z. Qian, Y. Peng, Y. Zhang, X. Lin, and J. Zhou. Real-time constrained cycle detection in large dynamic graphs. *Proceedings of the VLDB Endowment*, 11(12):1876–1888, 2018.

[107] J. T. S. Quah and M. Sriganesh. Real-time credit card fraud detection using computational intelligence. *Expert Syst. Appl.*, 35(4):1721–1732, 2008.

[108] V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, et al. DB2 with BLU Acceleration: So Much More Than Just A Column Store. *VLDB*, 6(11):1080–1091, 2013.

[109] A. Raza, P. Chrysogelos, A. C. Anadiotis, and A. Ailamaki. Adaptive HTAP Through Elastic Resource Scheduling. In *SIGMOD*, pages 2043–2054, 2020.

[110] C. Riegger, T. Vinçon, R. Gottstein, and I. Petrov. MV-PBT: Multi-Version Indexing for Large Datasets and HTAP Workloads. In *EDBT*, pages 217–228, 2020.

[111] S. Roozkhosh, D. Hoornaert, J. H. Mun, et al. Relational memory: Native in-memory accesses on rows and columns. In *EDBT*, pages 66–79. OpenProceedings.org, 2023.

[112] M. Sadoghi, S. Bhattacherjee, B. Bhattacharjee, and M. Canim. L-store: A real-time oltp and olap system. *arXiv preprint arXiv:1601.04084*, 2016.

[113] B. Samwel, J. Cieslewicz, B. Handy, J. Govig, et al. F1 query: Declarative querying at scale. *Proceedings of the VLDB Endowment*, 11(12):1835–1848, 2018.

[114] H. Saxena, L. Golab, S. Idreos, and I. F. Ilyas. Real-Time LSM-Trees for HTAP Workloads. *arXiv preprint arXiv:2101.06801*, 2021.

[115] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, et al. What serverless computing is and should become: the next phase of cloud computing. *Commun. ACM*, 64(5):76–84, 2021.

[116] S. Shen, R. Chen, H. Chen, and B. Zang. Retrofitting High Availability Mechanism to Tame Hybrid Transaction/Analytical Processing. In *OSDI*, pages 219–238, 2021.

[117] S. Shen, Z. Yao, L. Shi, L. Wang, L. Lai, Q. Tao, L. Su, R. Chen, W. Yu, H. Chen, et al. Bridging the gap between relational {OLTP} and graph-based {OLAP}. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 181–196, 2023.

[118] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd. Efficient Transaction Processing in SAP HANA Database: The End of A Column Store Myth. In *SIGMOD*, pages 731–742, 2012.

[119] U. Sirin and A. Ailamaki. Micro-architectural analysis of OLAP: limitations and opportunities. *Proceedings of the VLDB Endowment*, 13(6):840–853, 2020.

[120] U. Sirin, S. Dwarkadas, and A. Ailamaki. Performance Characterization of HTAP Workloads. In *ICDE*, pages 1829–1834, 2021.

[121] U. Sirin, P. Tözün, D. Porobic, and A. Ailamaki. Micro-architectural analysis of in-memory oltp. In *SIGMOD*, pages 387–402, 2016.

[122] Snowflake Unistore. Getting Started with Transactional and Analytical data in Snowflake, 2024.

[123] H. Song, W. Zhou, F. Li, X. Peng, and H. Cui. Rethink query optimization in htap databases. *Proceedings of the ACM on Management of Data*, 1(4):1–27, 2023.

[124] Splice Machine. Defining HTAP, 2017.

[125] StoneDB. A Real-time HTAP Database, 2022.

[126] J. Sun, J. Zhang, Z. Sun, G. Li, and N. Tang. Learned cardinality estimation: A design space exploration and a comparative evaluation. *Proceedings of the VLDB Endowment*, 15(1):85–97, 2021.

[127] Y. Sun, G. E. Blelloch, W. S. Lim, and A. Pavlo. On Supporting Efficient Snapshot Isolation for Hybrid Workloads with Multi-Versioned Indexes. *VLDB*, 13(2), 2019.

[128] TATP. TATP Benchmark Description (Version 1.0), 2009.

[129] Tecent. Webank. www.webank.com, 2023.

[130] Transaction Processing Performance Council. TPC-C, 2021.

[131] Transaction Processing Performance Council. TPC-H, 2021.

[132] A. Verbitski, A. Gupta, D. Saha, et al. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *SIGMOD*, pages 1041–1052. ACM, 2017.

[133] T. Vinçon, C. Knödler, L. Solis-Vasquez, et al. Near-data processing in database systems on native computational storage under HTAP workloads. *Proceedings of the VLDB Endowment*, 15(10):1991–2004, 2022.

[134] J. Wang, T. Li, H. Song, X. Yang, W. Zhou, F. Li, B. Yan, Q. Wu, Y. Liang, C. Ying, et al. Polardb-imci: A cloud-native htap database system at alibaba. *Proceedings of the ACM on Management of Data*, 1(2):1–25, 2023.

[135] Y. Wu, J. Arulraj, J. Lin, R. Xian, and A. Pavlo. An empirical evaluation of in-memory multi-version concurrency control. *VLDB*, 10(7):781–792, 2017.

[136] J. Yang, I. Rae, J. Xu, et al. F1 lightning: Htap as a service. *Proceedings of the VLDB Endowment*, 13(12):3313–3325, 2020.

[137] Z. Yang, C. Yang, F. Han, et al. Oceanbase: A 707 million tpmc distributed relational database system. *Proceedings of the VLDB Endowment*, 15(12):3385–3397, 2022.

[138] B. W. Yogatama, W. Gong, and X. Yu. Orchestrating data placement and query execution in heterogeneous CPU-GPU DBMS. *Proceedings of the VLDB Endowment*, 15(11):2491–2503, 2022.

[139] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[140] C. Zhang, G. Li, F. Hua, and J. Zhang. Survey of Key Techniques of HTAP Databases. *Journal of Software*, 34(2):761–785, 2022.

[141] C. Zhang, G. Li, and T. Lv. HyBench: A New Benchmark for HTAP Databases. *Proceedings of the VLDB Endowment*, 17, 2024.

[142] C. Zhang and J. Lu. Selectivity Estimation for Relation-Tree Joins. In *SSDBM*, pages 1–12, 2020.

[143] C. Zhang and J. Lu. Holistic evaluation in multi-model databases benchmarking. *Distributed Parallel Databases*, 39(1):1–33, 2021.

[144] C. Zhang, J. Lu, P. Xu, and Y. Chen. UniBench: A Benchmark for Multi-model Database Management Systems. In *TPCTC*, volume 11135, pages 7–23. Springer, 2018.

[145] J. Zhang, C. Zhang, G. Li, and C. Chai. AutoCE: An Accurate and Efficient Model Advisor for Learned Cardinality Estimation. In *ICDE*, pages 2621–2633. IEEE, 2023.

[146] J. Zhang, C. Zhang, G. Li, and C. Chai. PACE: Poisoning Attacks on Learned Cardinality Estimation. *Proceedings of the ACM on Management of Data*, 2(1):1–27, 2024.

Chao Zhang is a postdoctoral researcher at Tsinghua University. He was awarded the Ph.D. degree in Computer Science at the University of Helsinki, Finland. He has given a tutorial on HTAP databases in SIGMOD 2022 and gave a tutorial on cloud databases in VLDB 2022. He serves as a PC member of SIGMOD 2024-2025, VLDB 2023 Tutorial, and ICDE 2023. His research interests focus on heterogeneous database management systems.

Guoliang Li is an IEEE fellow, and a full professor at the Department of Computer Science, Tsinghua University. His research interests include database systems, large-scale data cleaning, and integration. He received the VLDB 2017 Early Research Contribution Award, TCDE 2014 Early Career Award, SIGMOD 2023 Best Papers, VLDB 2020 Best Papers, and ICDE 2018 Best Paper. He served as a general chair of SIGMOD 2021, a demo chair of VLDB 2021, and an industry chair of ICDE 2022.

Jintao Zhang is a master student at Tsinghua University. He received his bachelor's degree in Computer Science at Xidian University. His research interests focus on the intersection between database and machine learning.

Xinning Zhang is a master student at Tsinghua University. He received his bachelor's degree in Computer Science at Zhejiang University. His research interests focus on HTAP databases.

Jianhua Feng is a full professor at the Department of Computer Science, Tsinghua University. He received his bachelor's degree in Computer Science at Tsinghua University. His research interests focus on cutting-edge database management systems.