The VLDB Journal
https://doi.org/10.1007/s00778-024-00858-9

REGULAR PAPER

A survey on hybrid transactional and analytical processing

Haoze Song[1] · Wenchao Zhou[2] · Heming Cui[1] · Xiang Peng[2] · Feifei Li[2]

Received: 18 April 2023 / Accepted: 20 May 2024
© The Author(s) 2024

Abstract
To provide applications with the ability to analyze fresh data and eliminate the time-consuming ETL workflow, hybrid transactional and analytical (HTAP) systems have been developed to serve online transaction processing and online analytical processing workloads in a single system. In recent years, HTAP systems have attracted considerable interest from both academia and industry. Several new architectures and technologies have been proposed. This paper provides a comprehensive overview of these HTAP systems. We review recently published papers and technical reports in this field and broadly classify existing HTAP systems into two categories based on their data formats: monolithic and hybrid HTAP. We further classify hybrid HTAP into four sub-categories based on their storage architecture: row-oriented, column-oriented, separated, and hybrid. Based on such a taxonomy, we outline each stream's design challenges and performance issues (e.g., the contradictory format demand for monolithic HTAP). We then discuss potential solutions and their trade-offs by reviewing noteworthy research findings. Finally, we summarize emerging HTAP applications, benchmarks, future trends, and open problems.

Keywords Information system · Hybrid transactional and analytical processing · Real-time analysis · Storage design

1 Introduction

Driven by the increasing connectivity between data generation (e.g., Online Transaction Processing, for short, OLTP) and data consumption (e.g., Online Analytical Processing, for short, OLAP), real-time analytics based on new data has attracted much interest from academia and industry [18, 78, 111, 168, 175]. The fundamental motivation behind this trend is that much information is most valuable when it first appears and is usually time-decayed [8, 44, 95]. Meanwhile, data is increasingly consumed by intelligent algorithms (e.g., AI-assisted [118, 171]) but not human readers to conduct timely decision automatically [8, 14, 56].

Nevertheless, due to historical reasons, the connection between OLTP and OLAP is traditionally processed by an Extract-Transform-Load (ETL) workflow [164, 166], which usually takes minutes to hours. This workflow can be good enough for traditional business intelligence, where data is processed in large batches, and hence, the execution time takes minutes to hours (e.g., daily report [144] and data mining [75]). In these scenarios, the data freshness loss (i.e., the time consumed in ETL) and execution time are of the same magnitude and can be tolerable. However, for real-time applications that take seconds for execution to conduct time-critical decisions (e.g., real-time pricing [60], fraud detection [36, 136], smart industry [161, 174]), ETL is too slow and expensive. This trend emphasizes the importance of achieving high data freshness for analysis. We compare the goals and scopes of time-critical decisions and traditional business intelligence in Fig. 1.

In response, Hybrid Transactional and Analytical Processing (HTAP) was born to handle OLTP and OLAP requests in a single system, thus eliminating ETL. Specifically, HTAP systems are featured in their strong ability to interleave transactions and analytical queries [63, 103]. That is, analytical queries can observe the latest write made by transactions, and

✉ Haoze Song
hzsong@cs.hku.hk

Wenchao Zhou
zwc231487@alibaba-inc.com

Heming Cui
heming@cs.hku.hk

Xiang Peng
pengxiang.px@alibaba-inc.com

Feifei Li
lifeifei@alibaba-inc.com

[1] Department of Computer Science, The University of Hong Kong, Hong Kong, China

[2] Database and Storage Lab Alibaba Cloud, Hangzhou, China

Published online: 04 June 2024

transactions can promptly use the results of analytical queries as payloads. We summarize the common target scopes of HTAP databases in Sect. 3.

Such technical coherence between OLTP and OLAP raises new design challenges over data storage, execution engines, and query optimizers. To tackle these challenges, tremendous efforts have been made in this area, resulting in a rich literature of related papers and solutions (see Table 1). The adopted architectures and design choices may vary significantly. To systematically oversee existing designs, we broadly classify existing HTAP systems based on the design of data formats, which is also adopted by a previous study [126]. Differently, our survey greatly extends the two-way classfication of HTAP by summarizing common issues and discussing the combination of potential solutions.

In particular, we first study monolithic HTAP systems, which serve OLTP and OLAP workloads with identical data format designs. Typically, OLTP and OLAP workloads show different data access patterns. OLTP can operate on a single data tuple at a time and access many attributes, while OLAP typically accesses a massive number of rows at a time with a subset of tuple attributes. Monolithic HTAP systems strive to make a "one-size-fits-all" format design and allow OLAP to make progress regardless of concurrent OLTP. Monolithic HTAP may incur significant performance issues when serving mixed workloads. We summarize three common issues: contradictory format demand, performance degradation of MVCC, and performance isolation. In response to these common issues, several new techniques for data grouping, MVCC, in-memory snapshots, and read/write splitting have been proposed. We discuss them in Sect. 4.

We then study HTAP systems using hybrid data formats, which independently optimize the underlying data formats for OLTP and OLAP. Although hybrid data formats mitigate the contradictory format demand, they bring new performance issues. We summarize six common issues for HTAP with a hybrid data format: data synchronization, data consistency, gap of write efficiency, hybrid data access, performance isolation, and sharding strategy. To understand the design rationales, we further classify HTAP with hybrid data formats into four typical architectures: row-oriented, column-oriented, separated, and hybrid architectures. Based on our taxonomy, we discuss the existing solutions for the six common issues one by one in Sect. 5

To enlighten new HTAP designers, we make a pros and cons summary of existing solutions (monolith design, column-oriented hybrid design, row-oriented hybrid design, separated hybrid design, and hybrid architecture with hybrid data formats) in various scopes (transaction performance, query performance, storage overhead, data freshness, performance isolation, scalability, performance stability, and system complexity). This can be a good starting point for newcomers (see Sect. 6). We also present readers with emerging new applications, benchmarks, and future directions.

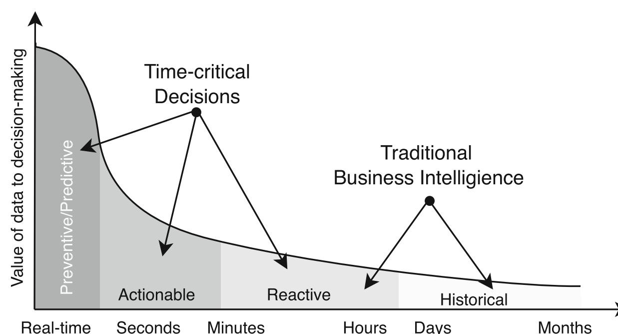

Fig. 1 A comparison between the targets of HATP databases and traditional business intelligence [35]. HTAP databases provide real-time analytics to maximize the value of data

1.1 Contributions

To the best of our knowledge, our survey takes the first step to deeply review the existing HTAP architectures, compare the resident technologies, and give a comprehensive overview of HTAP regarding system design, applications, and benchmarks. Our contribution includes:

– We systematically summarize the common design goals of HTAP. We then intensively study the common design issues and existing solutions to enlighten and guide researchers and practitioners in this area.
– We present a new fine-grained taxonomy, which comprehensively organizes HTAP architectures based on the storage layer. This provides us with a uniform methodology to compare different implementations and trade-offs.
– We present readers with cutting-edge real-time applications powered by HTAP systems. They can be practical templates for developing new ones and motivating new research in HTAP.
– Last but not least, we also present the emerging benchmarks, unique evaluation metrics, and standard evaluation methods of HTAP.

1.2 Related surveys and research collections

There are two tutorials related to our survey. Özcan et al. presented a tutorial on the taxonomy of HTAP systems in SIGMOD 2017 [126]. It covers multiple data processing systems, including NoSQL [157], SQL-on-Hadoop [2, 68, 162], and Spark SQL [17, 22, 109, 179]. However, due to the intensive research and development efforts, the architectures and technologies of new emerging HTAP systems (Table 1) are not covered in this tutorial (especially for the designs that appear after 2017). Compared to previous designs, these new

Table 1 This table lists the HTAP systems from academia and industry since 2014

	HTAP system	Developed by	Year
Academia	FSM [19]	Arulraj et al.	2017
	BatchDB [111]	Makreshanski et al.	2017
	Janus [18]	Arora et al.	2017
	H^2TAP [16]	Appuswamy et al.	2017
	Ptree [158]	Sun et al.	2019
	RDE [138]	Raza et al.	2020
	VEGITO [150]	Shen et al.	2021
	vWEAVER [89]	Kim et al.	2021
	Diva [90]	Kim et al.	2022
	Proteus [6]	Abebe et al.	2022
Industry	SQL Server [93]	Microsoft Inc	2015
	Oracle Dual [92]	Oracle Inc	2015
	SingleStore [153]	SingleStore Inc	2016
	L-store [142]	IBM Inc	2016
	SAP HANA (ATR) [97]	SAP Inc	2017
	CitusDB [54]	Microsoft Inc	2018
	F1 Lightning [175]	Google Inc	2020
	TiDB [78]	PingCAP Inc	2020
	IBM DB2 IDAA [34]	IBM Inc	2020
	PolarDB [168]	Alibaba Cloud Inc	2021
	OceanBase [69]	Ant Group Inc	2021
	Greenplum [110]	VMware Inc	2021
	Heatwave [123]	Oracle Inc	2021
	AlloyDB [65]	Google Inc	2022
	Unistore [80]	SnowFlake Inc	2022
	ByteHTAP [39]	ByteDance Inc	2022

The year attribute for each system is when its HTAP feature was first announced. We do not cover HTAP solutions from the big-data communities (e.g., Spark and Hadoop) but discuss them in related work

HTAP systems are more tightly coupled and leverage multiple advanced experiences from data warehouse systems. For instance, column stores are regarded as a common optimization approach in the HTAP systems with hybrid data format (Sect. 5). Moreover, as discussed in Sect. 1, our paper is more than a two-way classification. It greatly extends the taxonomy in [126] to oversee the common design issues of HTAP and lead the solutions.

Li et al. summarized the pros and cons of existing HTAP systems in the tutorial of SIGMOD 2022 [103]. Different from our work, their classification is limited to the selected systems (i.e., a new HTAP system may not fall into any categories) and does not discuss the HTAP systems with monolithic data formats (Sect. 4). In contrast, we comprehensively review different HTAP architectures, applications, benchmarks, metrics, and evaluation methods. Moreover, our survey deeply explores the dependencies and trade-offs between different design choices, providing a more fine-grained, hierarchical taxonomy to guide future research.

1.3 Survey organization

We first give a general view of ETL, a brief history of HTAP, and a discussion of other related systems in Sect. 2. Then, we summarize the common design goals and define two crucial metrics of HTAP in Sect. 3. After that, we review HTAP architectures with monolithic and hybrid data formats in Sects. 4 and 5, respectively. Based on our taxonomy, we discuss the lessons we learned in Sect. 6. In Sect. 7, we summarize the cutting-edge applications and standard benchmarks for HTAP. In Sect. 8, we analyze the future directions and challenges. Finally, we conclude this survey in Sect. 9.

2 Background

2.1 Extract-transform-load workflow

Traditional Extract-Transform-Load workflows, which we term *the first generation of ETL*, extract data from various

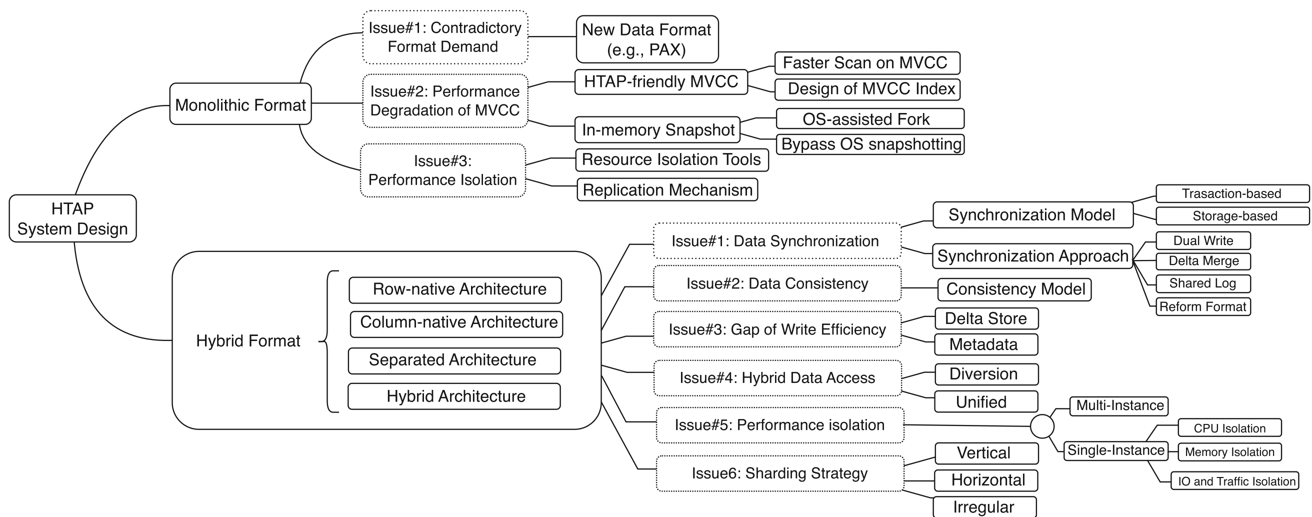

Fig. 2 An overview of our taxonomy. We classify existing HTAP systems into two main streams based on data format and discuss each stream's design issues and potential solutions

sources and transform and load them into a data warehouse. Without HTAP, ETL is a mandatory step in projects implementing decision-making information or knowledge management systems within organizations. However, traditional ETL was a long and costly process. It may be favored for routine queries (e.g., daily reports) and large-scale analysis (e.g., Spark SQL) while lacking adequate support for real-time analysis. In particular, the workflow periodically issues heavy ETL queries on the private storage of OLTP (e.g., MySQL), then extracts data updates from the query results, and finally merges updates into the private storage of OLAP (e.g., MonetDB).

Recent works of ETL focus on improving the timeliness of ETL to cater to the demand of real-time data injection, which we term *the second generation of ETL*. For instance, several works [164, 166] exploit the advantages of task parallelization to speed up the workflow. Other works [34] use batch processing to improve the throughput of transformation. As both HTAP and real-time ETL strive to provide high-performance OLTP and OLAP with fresh data, they share a similar design target.

However, different from HTAP, real-time ETL has several different design scopes and thus has a different application domain. First, the real-time ETL workflow is designed as a transparent service to the underlying database, and it should be compatible with different database engines and backends. In contrast, HTAP is a built-in service inside the database and is fully managed by the database engine. To capture the changed data, an HTAP database with hybrid data formats usually triggers a monitor inside the database instead of using ETL queries, which is studied as change data capture (CDC) in the literature [156]. Second, as a real-time ETL workflow processes data in a batch manner, they can clean data before injection using a customer program (e.g., a specific code written by the database manager). In contrast, such a user-defined data clean is not usually supported by HTAP.

Due to the aforementioned two different design scopes, a real-time ETL workflow usually leads to worse data freshness than HTAP. Meanwhile, due to the flexibility of ETL, it is more likely suitable for big-data analysis (e.g., MapReduce and Spark). Some previous studies [34, 126] indeed blur the boundary between real-time ETL and HTAP. In our survey, we differentiate them because they have evolved into two different research directions and have different optimization focuses in recent years. Our classification is based on whether the data synchronization is a built-in service of HTAP.

2.2 A brief history of HTAP

The word "HTAP" was first introduced to the public in 2014 by a Gartner report [130]. Essentially, before the word was created, multiple works [12, 26, 62, 85, 86, 102, 121, 132] have strived to handle both OLTP and OLAP workloads simultaneously, which are formerly mentioned as hybrid OLTP & OLAP, mixed workloads processing, or OLxP. Although these research prototypes may have been phased out in the recent HTAP production, some key techniques and designs have become the foundation for further development.

In recent years, driven by the increasing demands for all-in-one data management solutions and real-time analytics, more and more HTAP systems have been proposed and developed. We list these advanced HTAP systems in Table 1. These systems adopt different ways to implement HTAP. Each of them has different design decisions due to different design goals. To systematically study the performance trade-offs behind the design decisions, we broadly classify existing

HTAP systems into two main streams, and each stream contains multiple sub-streams (Fig. 2). We make a comprehensive discussion on their design decisions (e.g., storage model, synchronization model, and execution engine) and design rationales in Sects. 4 and 5.

Briefly, the first stream uses a monolithic data format with an optimized data structure to run transactions and queries simultaneously (e.g., PTree [158] and Diva [90]). Typical designs allow OLAP queries to observe transaction updates by sharing the same data copy. However, this may incur physical and logical resource conflict between OLTP and OLAP. Moreover, it neglects the opportunities for independently optimizing data engines and layouts for OLTP and OLAP. As such, the optimizations in OLAP (e.g., batch iteration [4] and late materialization [4, 5]) may not be adopted in these systems. The second stream leverages hybrid data format (e.g., TiDB [78] and BatchDB [111]). These systems run analytical queries and transactions on the dedicatedly optimized storage and thus allow for better performance optimization (i.e., independent storage engine, independent execution optimization, and independent scalability). Nevertheless, managing a hybrid data format is not free. Data synchronization between two data formats incurs additional overhead. For such hybrid architecture, a key challenge is how to maintain a consistent view of analytical queries efficiently, ensuring atomicity and isolation of the DBMS.

2.3 Other systems with a similar design goal as HTAP

Besides real-time ETL and HTAP, some other works also share a similar design goal, which tries to connect data generation and consumption in a real-time manner (i.e., a ground truth in data processing). Compared to HTAP, the specific focus of these systems is a bit different, and thus, they are tailored for different applications and deployment scenarios. We discuss two popular and representative categories below.

Streaming Systems. At birth, streaming systems do not support transactional semantics and provide a weak guarantee for atomicity. That is, considering two SQL operations in a single transaction, traditional streaming systems treat them as two individual events. When processing these two events on the OLAP side, a streaming system only guarantees they are applied in order while an OLAP query may still observe the intermediate results of the transaction (i.e., a single SQL operation in our former example).

Recent works consider supporting transactional analysis in the flight of streaming processing. They focus on the optimizations of consistent event processing and adopt the concurrency control model from the database community for correctness. However, even using a transactional streaming system for OLAP, the database manager still needs to configure the input events (which are essentially the updates from OLTP) using change data capture.

To summarize, traditional streaming systems are good for real-time alerting and monitoring but lack efficient support for transactions. Emerging transactional streaming systems do not have built-in database services and rely on additional stateful operations to process transactional requests.

Data Lakes. The concept of data lakes is processing different data sources in a single system and thus provides one-stop data management (same as HTAP). However, different from HTAP, data lakes target the process of all structured, semi-structured, and unstructured data and store data in its native format. On the contrary, HTAP only processes structured data generated by transactions and analyzes the generated data directly for better data freshness. In addition, HTAP generally provides stronger consistency and isolation guarantees than data lakes.

3 Design goals and evaluation metrics

By definition, HTAP systems target high-performance OLTP and OLAP. However, besides high transaction throughput and low query latency, unique challenges can always arise when mixing these two types of workloads, leading to specific design goals. We summarize these goals as follows.

- *[Transparent Query Execution]* As a unified system, database users should not be required to understand the working logic of the systems, nor should they identify query types or transactions manually [168]. HTAP systems should provide a unified interface for transactions and queries and route them automatically.
- *[High Data Freshness]* A significant motivation of HTAP is to eliminate ETL, thus providing intelligent insights into fresh data at generation speed. For that reason, freshness is commonly mentioned as an essential design goal of HTAP [39, 58, 78, 83, 103, 111, 150, 154, 168].
- *[Strong Performance Isolation]* Performance isolation refers to the ability to maintain the performance of a specified workload (e.g., OLTP) while another workload changes. Generally, in an HTAP application, transactions often play a mission-critical role in production. The performance degradation of OLTP can always lead to bad application quality. Thus, performance isolation is always treated as a significant design goal of practical HTAP systems [78, 103, 111, 150, 154, 168].
- *[Strong Consistency Across OLTP and OLAP]* Strong consistency is always good to have. Multiple HTAP systems (e.g., [39, 78, 168]) target maintaining strong consistency across OLTP and OLAP. By doing so, application developers do not need to spend extra effort handling data consistency issues in the application layer, largely reducing the development complexity.

– *[Excellent Elasticity and Scalability]* In HTAP scenarios, the consumption of physical resources (e.g., CPU and disk IO) may fluctuate significantly depending on the mixture of different types of workloads. Thus, supporting excellent elasticity and scalability can also be a significant design goal of HTAP systems [131, 138, 168].

Given such design goals, several evaluation metrics have been proposed. We highlight two HTAP-specific ones (i.e., data freshness and performance isolation). In contrast, the metrics for other design goals can fit into traditional ones (e.g., consistency models from OLTP are still applicable for HTAP).

3.1 Metrics of data freshness

To gauge data freshness, the metric should quantify how recently each analytical query sees the view (a.k.a. snapshot) of OLTP data in an HTAP system [117]. We summarize existing freshness metrics into two categories. The first category uses the occupied space ratio for the metric, while the second uses time intervals.

3.1.1 Space-based freshness metrics

Raza et al. propose *freshness-rate* in [138]. Assuming OLTP and OLAP engines have two different private storage. *Freshness-rate* represents the rate of data tuples that are the same between two private storage. We show an example in Fig. 3. We assume the data in the red box is visible to OLTP, and the part of the data in the blue box is visible to OLAP. As OLTP continuously generates new data, the visibility of OLTP and OLAP may differ.

Accordingly, a higher *freshness-rate* means better data freshness. When the two engines share the same data storage (e.g., using the monolithic data format with a single data copy), the *freshness-rate* metric will always be 1. Instead, when their storage is independent (e.g., using a hybrid data format and maintaining two data copies), this metric should generally be less than 1.

$$freshness\text{-}rate = \frac{count(tuples_{OLTP} \cap tuples_{OLAP})}{count(tuples_{OLTP} \cup tuples_{OLAP})} \quad (1)$$

Thus, HTAP systems can achieve a high *freshness-rate* either by the two engines sharing the same data storage or by speeding up the transfer of the corresponding delta (i.e., the grey data log in Fig. 3).

3.1.2 Time-based freshness metrics

Another freshness metric: *freshness-score* is based on the time dimension. It is more intuitive than *freshness-rate*.

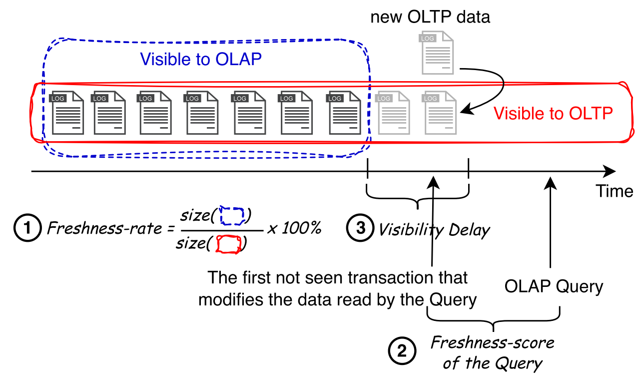

Fig. 3 This diagram shows the comparison of three freshness metrics

Briefly, the *freshness-score* for a single analytical query q is defined as a quantitative measure below, where t_q^s means the start time of q and t_q^{fns} represents the commit time of the first transaction which updates the data read by the query but not seen by q.

$$freshness\text{-}score_q = max(0, t_q^s - t_q^{fns}) \quad (2)$$

We show an example in Fig. 3. As *freshness-score* is defined for each query, we need to run the query multiple times to measure the performance of the whole system. Then, the freshness score of an HTAP system is defined as the aggregation of the freshness scores of all analytical queries that are continuously executed in the database:

$$freshness\text{-}score = AVG(freshness\text{-}score_q) \quad (3)$$

Accordingly, a smaller *freshness-score* means better data freshness. In particular, *freshness-score* = 0 implies that the HTAP system can always provide the most recent version of the operational data to all analytical queries. *freshness-score* = α seconds means that, on average, the snapshot used by the analytical queries is outdated by α seconds.

Note that identifying the first not-seen transaction can be challenging in a practical HTAP system, as we may not have a centralized sequencer that records the entire serializable execution history. It may need more effort to evaluate, mainly when the databases are partitioned and spanned over multiple machines.

Besides *freshness-score*, *visibility delay* is another time-based freshness metric. By definition, *visibility delay* is the time interval during which updates to the database can be visible to OLAP queries. Different from *freshness-score*, it may not directly represent the delay observed by users but focuses on the performance evaluation inside the system. This makes *visibility delay* much easier to measure as it does not rely on finding the first transaction not seen by the analytical queries. As a result, *visibility delay* is one of the most popular metrics evaluated by the previous papers [18, 34, 39, 78, 97, 111,

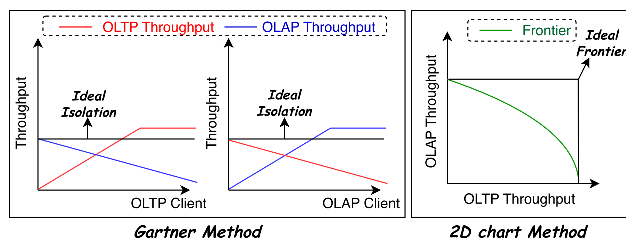

Fig. 4 This diagram compares two isolation evaluation methods

150, 175]. We show an example of *visibility delay* in Fig. 3, which capture the time window of data visibility to OLTP and OLAP.

3.2 Evaluation method of performance isolation

To measure performance isolation, a simple approach suggested by Gartner is to instruct one kind of workload client (e.g., OLTP clients) to sustain a configured throughput (e.g., about half of peak throughput) and allow another kind of client (e.g., OLAP clients) to saturate the throughput [47, 150]. When the number of later clients increases, the less performance degradation of the previous workload, the better performance isolation is achieved. We show an example in Fig. 4. Ideal isolation indicates no performance drop and thus plots a horizontal line in the Figure. This approach is followed by [18, 78, 111, 150, 168].

Milkai et al. extend this approach by visualizing the throughput frontier in a 2D chart [117]. In the chart, the x-axis represents OLTP throughput, and the y-axis represents OLAP throughput. Each pointer in the chart means the HTAP system can achieve a fixed OLTP throughput and a fixed OLAP throughput. Then, a frontier can be generated when given a fixed OLTP throughput, and the OLAP throughput is the maximum. An HTAP system achieves ideal performance isolation when each of the OLTP and OLAP workloads performs as if executed independently, which plots a rectangle frontier in the throughput frontier chart. See the example in Fig. 4. Compared to the previous approach, the chart can provide a more comprehensive view of the performance isolation property.

4 HTAP with monolithic data format

4.1 Common issues

We now discuss the challenges and solutions for HTAP systems with a monolithic data format. Sharing the same physical data format across OLTP and OLAP workloads is the most straightforward approach to executing analytical queries on the fresh data generated by transactions. As a mitigation solution for HTAP, most of the systems belonging to this stream evolved from legacy databases.

A few decades ago, OLTP and OLAP workloads were not clearly divided and were mixed in traditional databases (e.g., MySQL) without giving HTAP context. More recently, several works have been proposed to make the monolithic design suitable for HTAP's performance. Our survey focuses on these HTAP-specific optimizations.

In general, unified architecture has strengths in reducing data redundancy and eliminating the burden of managing consistency [90]. However, it has some performance issues:

- [***Issue1: Contradictory Format Design***] OLTP workloads prefer row-oriented data format for efficient tuple inserts and updates; however, it is inefficient for read-only analytical queries since they only have to read a tiny subset of attributes (with massive rows) from disk (or from memory) [4], wasting on I/O.
- [***Issue2: Performance Degradation of MVCC***] Multi-version concurrency control (MVCC) is widely adopted in OLTP DBMSs [172] to allow readers to make progress regardless of concurrent writers. However, the increased lengths of version chains further degrade the scan performance of analytical queries. Interchangeably, the long lifetime of analytical queries blocks the garbage collection (GC) and causes storage overhead [88]. We further detail this issue in Sect. 4.3
- [***Issue3: Weak Performance Isolation and Scalability***] Simply marrying OLAP to OLTP breaks performance isolation. This is because analytical queries are usually compute-intensive, which take hundreds to thousands of times longer execution time and consume much more resources than short-lived transactions [61, 83]. In such a case, analytical queries may block the execution of transactions and cause resource starvation [32, 146]. In addition, scalability can be a major problem for monolithic HTAP. The read and write side may have different scale-out needs. OLTP workloads may be CPU-intensive, and OALP workloads typically require larger memory space for storing the intermediate query results.

In the rest of this section, we dive into these research problems posted by HTAP and show the key technologies to solve (or mitigate) these performance issues in the monolithic HTAP architecture. We show the potential solutions for ***Issue1*** in Sect. 4.2, the mitigation for ***Issue2*** in Sects. 4.3–4.4, and the mitigation for ***Issue3*** in Sect. 4.5. Most of these solutions are orthogonal to each other and can be combined into a single HTAP database.

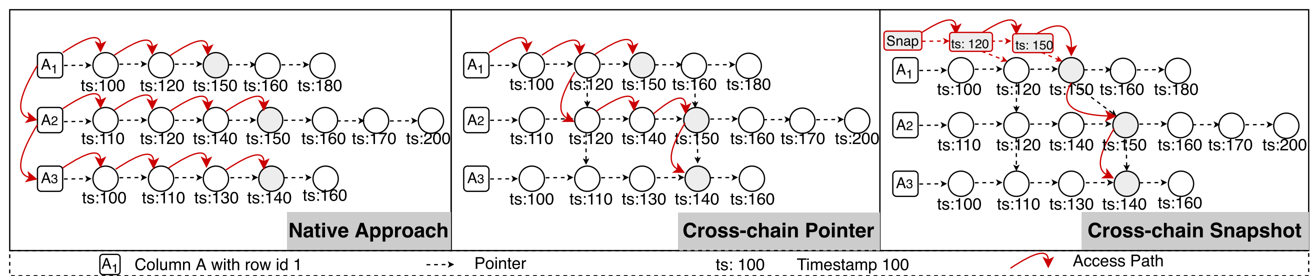

Fig. 5 Faster Scan of MVCC. This diagram provides an overview of the existing approaches that reform the MVCC chain for better scan performance. It shows the access path of a column scan operator on column A with timestamp 150

4.2 New data format for HTAP

Several works strive to overcome the contradictory performance properties by redesigning the existing formats. Their new format designs target providing high performance for OLTP, which should be the same as a row-oriented data format, and providing high performance for OLAP, which should be the same as a column-oriented data format. We introduce some of the representative works below.

Partition Attributes Across (PAX) was proposed in [10] by Ailamaki et al. Generally, PAX is designed for on-disk deployment and specifies how to group rows and columns together. Instead of storing data row-by-row within a disk block as row-stores, PAX stores rows column-by-column in a single disk block. This differs from a pure column store, which stores each column in separate disk blocks. The key difference is that if you had a table with 10 attributes, then in a pure column store, data from each original tuple is spread across 10 different disk blocks, whereas in PAX, all data for each tuple can be found in a single disk block. Since a disk block is a minimum granularity with which data can be read off of disk, in PAX, even if a query only accesses only 1 out of the 10 columns, it is impossible to read only this single column off of disk, since each disk block contains data for all 10 attributes of the table. Generally, PAX was able to achieve the CPU efficiency of column-stores while maintaining the disk I/O properties of row-stores.

Several commercial works adopted PAX and introduced how to use PAX in their product, e.g., Spanner [20] and Vectorwise [27]. Besides, several successive research works also follow the PAX data format and propose new optimizations. For instance, Jin et al. reorders columns within a single disk block and puts frequently co-accessed columns into nearby positions [81]. Such a way is effective for wide tables with many columns, as a random disk seek takes a large proportion of the I/O cost when reading a few from the many columns.

4.3 Making MVCC HTAP-friendly

4.3.1 Faster scan on MVCC

In MVCC, a version storage schema specifies how the system stores versions and what information each version contains [172]. We show an example of multi-version storage layouts in Fig. 5. We term the basic MVCC implementation as the native approach. In particular, a native approach records each item's version in a linked list and tags each version with a version ID (e.g., commit timestamp). In our example, A1, A2, and A3 are the three data items. Each of them has multiple versions generated by OLTP updates. The versions are organized in chronological order (e.g., A1 has five versions, and the latest version is tagged as 180).

To perform a scan, the executor traverses each version chain from front to back until the valid data versions are found. For instance, assuming an OLAP query performs a scan using the timestamp 150, thus the required data version is shown in grey. To find these versions, the scan starts from A1's initial version (i.e., 100) and moves forward to the next version (i.e., 120) until the version ID is equal to or bigger than 150 (or the version chain ends). Then, the scan on A1 ends on the version with the biggest version ID, which is smaller or equal to 150. After finding the version for A1, the scan continues on A2 and A3. Significant performance issues can arise when the version chain is long and the number of required data items is huge.

Several works are proposed to reform the multi-version storage layer to support faster scans. Among the proposals, the key idea is maintaining pointers between stable data version (i.e., the data version has been committed by the transactions) to speed up version traversal, which enhances scan performance at the cost of updates' complexity. Due to the different design goals, strategies vary in when and how to construct the cross-chain link and trade-off between update efficiency, scan efficiency, and storage overhead.

We summarize existing proposals into two categories. The first category adds cross-chain pointers to the adjacent items. Hence, quires can filter unnecessary data versions when

scanning over the version chain. As shown in Fig. 5, when traversing the version chain of A1, it records the the cross-chain pointer for A2 when accessing the version 120 of A1. Thus, when reading A2, the scan begins at the record position (i.e., version 120 of A2) instead of the initial position of the version chain.

For correctness, a safe cross-chain pointer always starts from the version with a higher (or equal) version ID to a version with a lower version (or equal) ID. Thus, the required version will not be neglected in such an optimization. With the cross-chain pointer, the executor can find the target versions with less traversal effort (see Fig. 5). A practical implementation is vWEAVER. vWEAVER features a frugal version of skip lists [133, 178] and leverages a new probabilistic search algorithm to decide whether to generate a cross-chain pointer by coin-flipping algorithm.

Another category leverages pointers to link the data items that will be (potentially) accessed in the same snapshot together when adding new data items into the multi-version chains. So that scans can be done vertically through the pointers and skip unrelated data versions efficiently. We show an example in Fig. 5. When finding the required version for the given timestamp (e.g., 150), the scan operation directly transverses the chain of snapshots and uses the weaved version to A1, A2, and A3. Compared to the first category, the second category is more aggressive and may lead to higher scan performance, while it also incurs much more complexity for version management.

To our knowledge, P-Tree [158] is an instance belonging to the second category. Although the P-tree is essentially an in-memory tree index, it holds all the data items in the index and thus works as an in-memory MVCC store. This property makes it different from the MVCC indexes we will introduce later. In particular, P-tree is a nested tree structure and uses linked snapshots for fast versioning. With the nested structure, P-trees traverse the linked snapshot to find the versions and read the version using nested pointers.

In summary, both categories speed up data scanning by sacrificing update performance and incurring extra storage overhead. To our knowledge, existing proposals are mainly designed for flat schemas. How to effectively support complex data types (e.g., arrays and nested fields) is still an open problem. Some other limitations of the two approaches may include: 1). when the values in a column are tiny (in comparison to the size of the pointers), the pointers can become too expensive and ultimately negate the version-skipping benefits. 2). pointers complicate garbage collection and can cause more fragmentation in storage. It should also be noted that other implementation details (e.g., garbage collection policy and concurrency model) can also be critical to the end-to-end performance and further affect the HTAP design, but we do not discuss them due to space considerations.

4.3.2 MVCC Index

Another challenge raised by HTAP is how to support concurrent index updating. Even though several efforts [23, 112, 148, 160, 169] have been devoted to building concurrent tree-based data structures before HTAP is motivated, they may not be directly applicable to HTAP scenarios. The reason is that the data access pattern of the analytical queries differs from the typical read-only transactions, e.g., including more wide-range scans.

Taking the B+ tree [49] as an example, the lookup operation has to protect access to the inner and leaf nodes since other operations (e.g., inserts and deletes) may change them concurrently. Thus, the two types of operations will be blocked by each other, leading to sub-optimal performance (e.g., long tail latency of OLAP). It desires a new mechanism to efficiently split and insert inner and leaf nodes with the existence of scan-oriented data access.

VEGITO [150] poses the design challenges of the MVCC index in HTAP and proposes an epoch-base updating mechanism that parallelizes the updating in the same epoch (with both task parallelism and data parallelism) to reduce the conflicts. By its design, VEGITO provides slightly stale snapshots to the analytical queries.

Diva [90] suggests a new provisional version indexing for HTAP based on the observation that data versions are continuous and visible only for a sliding time window. The idea is to co-locate a record and its first old version in the main index and store the rest of the versions in separate version space (i.e., provisional version indexing). This separation lets Diva conduct rapid version searching and prompt cleaning of stale data versions simultaneously. Note that prompt garbage collecting benefits scan operation (with the reduced length of the version chain) and alleviates storage costs.

4.4 In-memory snapshot algorithm

In addition to conducting analytical queries on the MVCC storage, another straightforward approach is taking an in-memory snapshot to serve OLAP workloads. Using separated consistent snapshots can efficiently eliminate the concurrency issues (e.g., race condition), as well as the performance degradation on MVCC (i.e., ***Issue2***).

Several efficient snapshot algorithms have been proposed to construct a snapshot with low overhead, including Copy-on-Write (CoW) [104, 106] and Zigzag [37], and multiple database vendors and research prototypes adopt memory snapshot approach for hybrid transactional and analytical processing, e.g., Hyper-O[1] [85], SwingDB [116], AnKer [149], Kvell+ [100], and update-aware NDP [167].

[1] Hyper adopts the snapshot solutions for their initial version [85] and transfers to MVCC for their later development [31, 124]. To distinguish them, we term the initial version as Hyper-O.

Nevertheless, executing analytical queries with an in-memory snapshot may have the following limitations: First, the application scopes are limited. In-memory snapshots are primarily used for in-memory databases that are deployed in a single machine. It can be difficult to scale out when the database is partitioned into multiple shards. Second, in-memory snapshots incur additional overhead memory footprints, which can become severe when serving large-scale update-intensive applications. We summarize substantial research works by classifying them into two categories: OS-assisted Fork and bypass OS snapshotting.

4.4.1 OS fork

`Fork` is a copy-on-write mechanism implemented by operating systems. Hyper-O [85] leverages the Unix `fork()` to generate in-memory snapshots for analytical queries in a single machine. In particular, `fork()` is used to spawn child processes that share their entire virtual memory with the parent process. Practically, Hyper forks a new snapshot and thus starts a new OLAP query session process periodically (or on demand). The fork algorithm is always executed between two transactions to guarantee consistency and isolation. An incoming analytical query will be assigned to a specific OLAP query session, and the ongoing transactions will not be blocked. However, a process fork in Unix can be expensive when the snapshots are taken frequently. As a result, this snapshot mechanism is disused in the later version of Hyper [31, 124].

Scale-out Hyper (Scyper) [121] is a variant of Hyper-O, aiming at scaling out Hyper-O horizontally (without partitions). Scyper uses REDO logs to copy data from the primary (i.e., the single machine of Hyper-O) to other secondary replicas and let OLAP queries execute on secondary replicas. For load balance and consistency, Scyper uses a centralized coordinator for OLAP before the queries are scheduled.

4.4.2 Bypass OS snapshotting

Sharma et al. study the overhead of in-memory snapshot algorithms and propose a new lightweight snapshot mechanism via `vm_snapshot` in AnKer [149]. Their key motivation and observation is that, unlike other snapshotting purposes (e.g., for generating checkpoints), HTAP workloads require taking snapshots at a much higher frequency to realize better data freshness. In response, Anker introduces a new custom system call (`vm_snapshot`) and integrates the concept of rewiring directly into the Linux kernel.

The co-design between underlying components and the databases overcomes the restrictions of the OS. Individual snapshots in AnKer reserve a few short version chains instead of calculating totally transparent snapshots.

4.5 Leveraging replication mechanism

Recall the ***Issue3*** of monolithic HTAP. To achieve isolation when processing OLTP and OLAP workloads in a single machine, several methods are proposed to isolate CPU, memory, I/O bandwidth, and network traffic (e.g., binding CPU Cores, restricting the usage of memory and network, etc.) We refer readers to Sect. 5.5.5 for more details.

Besides processing OLTP and OLAP workloads together, leveraging the replication mechanism is another approach to handling HTAP workloads. For instance, our previous example (i.e., Scyper) in Sect. 4.4.2 falls into this approach. Scyper can be configured to serve OLTP on the primary and serve OLAP on the secondary to provide performance isolation between the two types of workloads.

It should be noted that replication is also commonly used in the legacy OLTP databases (e.g., MySQL [122] and PostgreSQL [84]) for fault-tolerance [41, 72, 73] and better performance (e.g., read and write division [50, 108]). OLAP queries can be naively executed on the replica for performance isolation and scalability, which is also known as read/write splitting in the industrial community. In addition, several optimizations can be applied to customize the approach, providing better performance in the context of HTAP.

For instance, PostgreSQL-SR [117] replicates data by replaying Write-Ahead Logging (WAL) records in a streaming manner (i.e., without waiting for the WAL to be filled), which makes the replica stay more up-to-date. To serve a hybrid workload, the primary is generally used for OLTP, and the replica serves read-only queries. Separating OLTP and OLAP workloads into different processing nodes (machines) provides physical resource isolation.

We assume the replication approach discussed in this section adopts the same physical data format (e.g., row-oriented) between primary and secondaries (a.k.a. peer replicas) and will discuss the approaches using different data formats in Sect. 5.

5 HTAP with hybrid data format

5.1 Common issues

We now discuss the case for HTAP with the hybrid data format. Column-oriented data storage has become a paradigmatic choice for OLAP DBMSs (e.g., MonetDB [28], SnowFlake [55], ClickHouse [45], RedShift [74], and Vectorwise [180]) in the recent decades. The key feature of a column-oriented database is that it serializes all of the values of a column together [145]. Compared to row-oriented storage, column-oriented storage has the potential to reduce the amount of data read by orders of magnitude (only reading

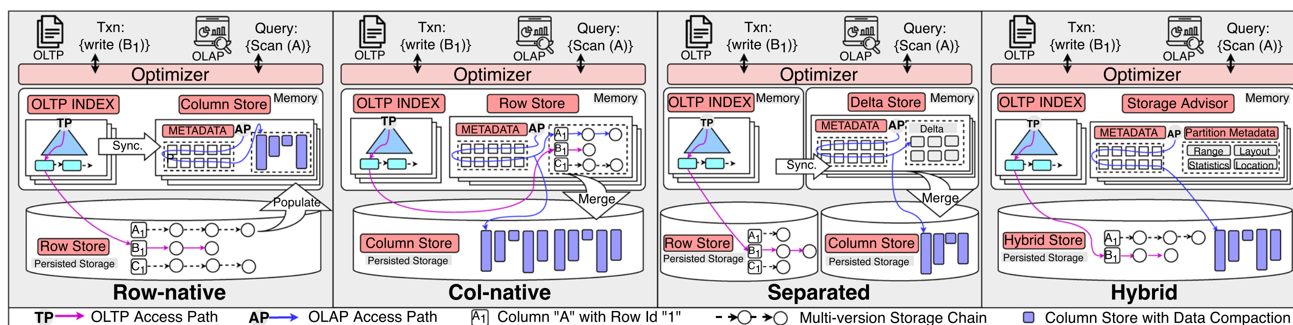

Fig. 6 An overview of different HTAP architectures with hybrid data format. The taxonomy is based on the different data formats adopted in the persisted storage layer, which are the root causes that influence the design decisions of in-memory storage, processing engine, data synchronization, data access path, and query optimization

in relevant columns) and benefit from column-specific optimization (e.g., column-oriented compression and execution [3, 5], new join algorithm [4]).

Given this, following the philosophy that one size does not fit all, many developers turn to host hybrid data formats (i.e., row store and column store) in a single database, especially attracting much more interest from the industry. The advantages of using a hybrid data format are significant: it provides the opportunity to optimize OLTP and OLAP independently and adhere them together in a lightweight manner. However, it indeed raises new challenges. Below, we summarize the common issues in implementing HTAP databases with hybrid data formats.

– ***[Issue#1:Data synchronization]*** How to keep the two storage (a.k.a data copy) abreast with each other without introducing much more additional overhead [78, 111] is one of the foremost challenges given a hybrid format design. A satisfactory synchronization method should meet the following requirements: First, it causes a minimal perturbation in the normal case of OLTP and OLAP. Second, it keeps the hybrid storage up-to-date, i.e., updates from OLTP are shipped and applied to the column store continuously. Third, it should be scalable to the number of transactions and analytical queries.
– ***[Issue#2: Data consistency]*** Along with the data synchronization, it is non-trivial to guarantee the consistency of HTAP databases when maintaining multiple data copies in hybrid data format. A safe guarantee of strong consistency is always keeping an intact snapshot of the whole row store in the column store. However, this approach may incur high coordination costs as the column store should always ask the entire row store for snapshots. In addition, the problem can become more complex when the row store or column store is distributed or partitioned into multiple shards.
– ***[Issue#3: The Gap of Write efficiency]*** Regardless of the specific implementation adopted by the row store and column store, there is a common issue in the gap of write efficiency between the two stores. Specifically, a row store is optimized for data updates and tuple inserts, while it becomes much more costly for a column store to absorb all the newly generated data since updates are separated into multiple columns across the format, and the column store is read-optimized, which groups different rows of the same column together.
– ***[Issue#4: Hybrid Data Access]*** Serving a request in a hybrid data format may incur hybrid data access. That is, to serve a single query, the execution engine may need to travel from both row and column stores and combine the results for answering queries. It's challenging because the transformation of different data formats is not free. Moreover, the execution engines of OLTP and OLAP have contradictory optimization demands. OLTP engine commonly uses volcano-style per-tuple iterators [66] (e.g., MySQL [122]), which is favored for processing small data. In contrast, per-tuple iterators (or even using a small batch size) largely limit the parallelism in OLAP and cause avoidable function calls frequently. It suggests a dilemma: an execution engine with a fixed block iteration size is either sub-optimal for OLTP workloads or sub-optimal for OLAP workloads.
– ***[Issue#5: Performance Isolation]*** Compared to the monolithic architecture, hybrid data formats mitigate the contention between OLTP and OLAP by attaching the two types of workloads to their desired data formats. However, it still deserves a careful design to isolate the performance between the two types of workloads since data is continuously synchronized from row store to column store. The design of data synchronization and hybrid data access may break the isolation between row and column stores. That is, row stores may need to generate

additional logs for synchronization, and column stores should always absorb all updates timely.

– ***[Issue#6: Sharding Strategy]*** Partitioning the database into multiple shards is a significant method for scaling out. Generally, OLTP and OLAP have different scale-out needs and have different spike patterns. The data that is read-hot is not necessarily write-hot. When grouping row and column data into shards, an efficient strategy should co-locate the frequently accessed data based on different workload characteristics.

5.2 A taxonomy of existing achitectures

To handle these issues, multiple technologies are proposed, and massive design decisions are made. Most of these design choices are highly dependent on each other and rely on the assumption of the underlying layer. We classify the HTAP systems with hybrid data formats into four categories to systematically study the trade-offs of these design decisions and technologies. The classification is based on the different data formats adopted in the persisted storage layer.

Note that we ***do not*** differ the systems based on their deployment model (e.g., shared-nothing, shared disk, or shared everything) and deployment scopes (e.g., within a data center using RDMA or cross data center using wide area network). Specific optimizations on these aspects (e.g., using RDMA for fast data synchronization) can be important for a given HTAP database. However, we do not plan to cover them in our survey as our survey targets providing high-level design guidelines, and these optimizations rely on specific hardware assumptions and are worth a more detailed investigation.

Figure 6 gives an overview of the four architectures. We illustrate the major characteristics of each category below. *Row-native architectures* persist row-oriented data and construct in-memory column store as secondary storage. To keep the column store up-to-date, a typical design is synchronizing new updates from the in-memory execution engine directly. When building a new column store (either on the primary node or on a stand-by replica), it populates data from the persisted storage layer. This architecture represents the road to equipping row-oriented DBMSs with a plug-in column data format. Typical implementations are SQL Server [58], Oracle Dual [92], and AlloyDB [65].

As mentioned previously, we do not restrict our classification by the deployment model. That is, row-native architectures may also be applicable to be deployed at multiple machines (nodes) using a shared-nothing architecture. An example is PolarDB-IMCI [168], which separates OLTP and OLAP workloads into different nodes. Unlike the separated architecture we discussed later, it relies on replicating a row data copy to construct its column stores on a new node.

Unlike row-native architectures, *column-native architectures* typically persist column-oriented data and constructs in-memory delta store (i.e., update-optimized storage) for efficient updating. Then, the newly generated updates will be merged periodically into the persisted column store.This architecture is commonly adopted by HTAP systems originating from OLAP DBMSs. Notable implementations include SAP HANA [152], MemSQL [38] (renamed as SingleStore [153] in 2020), and NoisePage [105].

Separated architecture usually maintains row and column stores individually. Updates are shipped from the row store to the column store in a streaming manner (with consistency guarantees). To alleviate the gap in write efficiency between row store and column store, an in-memory delta store is typically resident on the column side. Similar to the column native architectures, the updates in the delta store will be merged into the persisted column store periodically. This architecture represents the road to building an HTAP database on the shoulder of existing row and column stores. Typical implementations include TiDB [78], Heatweave [123], and F1 Lightning [175], IBM BLU [137], and ByteHTAP [39].

The key concept of *hybrid architectures* is the adaptive storage layer [6, 12, 19, 57]. Adaptive storage automatically evolves the data format of the stored data to achieve the best performance for both OLTP and OLAP workloads. For instance, HTAP systems belonging to this architecture always keep the data that are frequently updated by OLTP in row format and group the data that are always read together into column format. The shift of data format depends on the historical statistics of data access patterns, i.e., frequently updated tuples are stored in rows, while frequently scanned tuples are stored in columns. A major difference between separated and hybrid architecture is that hybrid architecture stores the full copy of data neither in rows nor columns. To the best of our knowledge, hybrid architectures are still limited to research efforts due to engineering complexity and performance reliability. This architecture represents the road to building an HTAP database from scratch. Typical research prototypes includes H_2O [12], FSM [19], and Proteus [6].

In the rest of this section, we dive into the technical aspects of the four architectures regarding HTAP-specific issues. Sections 5.3 and 5.4 introduces the essential background of row and column store, which is the foundation of hybrid HTAP. We compare the design of row and column stores and show how *Issue3* and *Issue4* come up.

Section 5.5 focuses on the solution to HTAP-specific issues given the four architectures. In particular, Sects. 5.5.1 and 5.5.2 presents the solution to *Issue1*. Section 5.5.3 discusses *Issue2*. Section 5.5.4 show the design for handling *Issue3*. Section 5.5.5 details *Issue4*. Sections 5.5.6 and 5.5.7 target *Issue5*. Finally, Sect. 5.5.8 discusses *Issue6*.

Table 2 Implementations of HTAP with hybrid data format (1)

Category	System	OLTP Storage Index	Version Storage	OLAP Storage Type	Engine Compression	Vectorization	Parallel
Row-native	SQL Server [58]	B Tree	Append-only	Column Store	✓	✓	✓
	Oracle Dual [92]	B Tree	Delta	Column Index	✓	✓	✓
	AlloyDB [65]	B Tree	Append-only	Column Store	✓	✓	✓
	PolarDB IMCI [168]	B Tree	Delta	Column Index	✓	✓	✓
Col-native	SAP HANA [152]	B Tree	Append-only	Column Store	✓	✓	✓
	MemSQL [131]	Skiplist	Append-only	Column Store	✓	✓	✓
	NoisePage [105]	B Tree	Delta	Column Store	✓	✓	✓
Separated	TiDB [78]	LSM Tree	Append-only	Column Store	✓	✓	✓
	Heatwave [123]	B Tree	Delta	Column Store	✓	✓	✓
	F1 [175]	B Tree	Append-only	Column Store	✓	✓	✓
Hybrid	FSM [19]	B Tree	Append-only	Column Store	✓	✓	✓
	Proteus [6]	Hash	Append-only	Column Store	✓	×	✓

A summary of different design decisions made by both commercial and research HTAP systems on transactional processing and analytical processing

5.3 Design of row store

Row stores in HTAP inherit the design from existing OLTP systems. MVCC is one of the common choices. All HTAP systems in Table 2 use multi-version data structures in their row store. Additionally, indices are used over version storage to speed up updates and lookups. In the rest, we discuss the index and storage designs of the representable HTAP databases in Table 2. It should be noted that, in hybrid HTAP, the implementation of row stores majorly influences the performance of OLTP. Thus, the design of the row store may not be specific to HTAP but implies significant design considerations in OLTP. Nevertheless, when the hybrid plans are enabled, the physical design of the row store can become a part of an HTAP-specific problem (to be illustrated in Sect. 5.5.7); thus, we briefly list the design choice of representable HTAP systems here for interested readers.

Choice of Indexes. Most existing HTAP systems follow the classical template and use B-Tree indices (or its variant) [49, 67]. Several systems use hash indices and skip lists, which are delicately optimized for lookup efficiency at the cost of update overhead. In turn, the LSM Tree adopted by TiDB has better write performance with a slower read.

Design of Version Storage. Following the taxonomy in the paper [172], version storage of HTAP systems in Table 2 lies in two typical implementations: append-only storage and delta storage. Append-only storage always allocates an empty slot for the new version and applies the modifications to the data in the newly allocated version slot. On the contrary, delta storage only creates a delta version that contains the modified values instead of the entire tuples.

This makes the two designs differ in read and write efficiency. Each version in the append-only storage can be accessed independently, making it more suitable for performing read requests. In contrast, delta storage is ideal for updating since it reduces the overhead for memory allocation.

Overall, the design of the row store targets different performance aspects. However, they all target efficient updates (using MVCC to improve the concurrency) and are suitable for a full-row insert.

5.4 Design of column store

5.4.1 Storage layer of column store

The critical decision with the column store is how the columns are physically structured. We classify existing designs of column stores in HTAP into two categories.

Native Column Store. Columns are actually collections of rows with the same attribute. A common approach to storing column data is partitioning each collection into multiple segmentations with a fixed number of rows. Then, each column segmentation can be encoded and stored independently (i.e., on different pages) [94, 110]. Data can be ordered within each segmentation by the column attribute or the primary key [155]. By doing so, every insert in column stores results in a collection of physical inserts on different segmentations. Thus causing a gap in write efficiency (i.e., *Issue3*) when compared to write-optimized indexes and version storage of the row stores in Sect. 5.3.

In-memory Column Index. Instead of storing columns directly, an additional approach is building a column index based on the row store. In-memory column indexes serve as an in-memory data buffer to speed up column-oriented data access. The initial idea of column index dates back to 2008 when Abadi et al. proposed index-only plans in [4]. By creating a

Table 3 Implementations of HTAP with hybrid data format (2)

Category	System	HTAP Data Sync Model	Approach	Delta (Metadata)	Freshness	Perf. Isolation	Optimizer Hybrid
Row-native	SQL Server [58]	Txn-based	Dual Write	TP-storage(Tail Index)	high	Bind Core	✓
	Oracle Dual [92]	Txn-based	Dual Write	TP-storage (Txn Map)	high	–	–
	AlloyDB [65]	Txn-based	Dual Write	TP-storage (Txn Map)	high	–	✓
	PolarDB-IMCI [168]	Txn-based	Dual Write	Append Index (BitMap)	high	Instance	–
Col-native	SAP HANA [152]	Storage-based	Delta Merge	TP-storage (Delta Index)	high	–	–
	MemSQL [131]	Storage-based	Delta Merge	TP-storage (Skiplist)	high	–	–
	NoisePage [105]	Storage-based	Delta Merge	TP-storage (B Tree)	high	–	–
	TiDB [78]	Storage-based	Consensus	LSM Tree (Delta Index)	medium	Instance; Quota	✓
Separated	Heatwave [123]	Storage-based	Shared log	None (None)	medium	Instance	–
	F1 [175]	Storage-based	Shared log	PAX (B Tree)	medium	Instance	–
Hybrid	FSM [19]	Storage-based	Reform Format	None (Col Map)	high	–	✓
	Proteus [6]	Storage-based	Reform Format	None (Hash)	high	Thread Pool	✓

A summary of different design decisions made by both commercial and research HTAP systems on HTAP-specific issues. Note that, for the column "Perf. Isolation" and "Hybrid Optimizer", if no method is supported or is mentioned in publicly available materials, it is marked as "–". Moreover, the systems have "instance-level" isolation, indicating their OLTP and OLAP can be deployed into multiple machines (i.e., distributed), while other systems typically deploy OLTP and OLAP together

collection of indices, it is possible to answer the query that covers all of the columns used in an analytical query without ever going to the underlying row-oriented stores. Thus, the in-memory column indexes essentially serve as a column-oriented data storage for OLAP queries.

SQL Server [94] introduced column index in 2011. Instead of directly building indices on the row-oriented tables, it stores columns into column segmentations. The column index is built based on the segmentations, serving as a dictionary. Oracle Dual [92] builds column indices on its heap tables. Heap table [79] is a row-oriented table representation format that stores data in no particular order (i.e., without a clustered index). Hence, operations on the in-memory column index will never re-structure physical data organizations in the heap table. In turn, updates and deletes are handled as an in-place operator within the heap table and will not affect the column index. This loosely coupled structure makes adding column indices to the existing database safe and convenient as a plug-in extension. PolarDB-IMCI [168] performs column index as a secondary index based on InnoDB's row-based buffer pool. Same as the index-only plans [4], it can create indices that cover all or a subset of columns.

5.4.2 Execution engine

A specialized execution engine plays a vital role in performing analytical queries. As shown in Table 2, almost all selected HTAP systems embrace this OLAP technology ecosystem. Nevertheless, it leads to *Issue4*, as discussed previously. In this subsection, we provide the background on such engine optimizations.

Column-oriented Compression and Execution. Column store has a different compression pattern from row stores. Column store allows compressed values from more than one row at a time. Several well-studied compression algorithms are proposed in the literature, e.g., run-length encoding (RLE) [141]. Based on compression, late materialization [4, 5] lets execution engines directly operate compressed data to defer data decoding until all predicates have been applied. This optimization enables database systems to construct fewer tuples at runtime, reducing I/O and computation costs. However, how to perform late materialization on different data formats with different compression strategies is still an open problem.

Vectorization and Intra-query Parallelization. Vectorized query processing [4, 29] and intra-query parallelism [11, 107] are also critical to the performance of analytical queries. However, they are strongly coherent with the design of the column-oriented data format and may not be common in the engine of the row store. It still calls for solutions to enable efficient data access when both row and column engines are desired (see our discussion in Sect. 5.5.7).

5.5 Road to HTAP

Given the design of the row store and column store above, we now present the design choices for bridging the two data formats into a single system as well as the potential solutions to the aforementioned issues. In addition, we also add a summary of the pros and cons of the candidate solution in each sub-section to guide future development. Table 3 summarizes existing designs of representative systems.

5.5.1 Data synchronization model

Existing data synchronization models fall into two categories: transaction-based and storage-based. Figure 7 shows how the writes of a transaction T are synchronized and can finally be observed in both storages. Depending on the consistency guarantees the HTAP system provides, synchronization in both models can be done either synchronously or asynchronously. When the data synchronization is done asynchronously (i.e., the step with a dotted line is done in the background), it removes the data transfer and communication out of the critical path of transactional processing, thus for better performance isolation.

Using the transaction-based model, the coordinator of transaction T performs a double write to both row and column stores. On receiving the full copy of all writes in the write set of T and getting ready to be committed (e.g., all needed latches are preserved), the storage node sends an acknowledgment message to the transaction coordinator. Then, the coordinator finalizes the transaction with a commit message. To enforce consistency, it can notify the client of the success of T only when the coordinator receives both acknowledgments of the commit message from the two stores.

Alternatively, the storage-based model synchronizes data on the ordered log. The units of log records can be transactions or read/write operations. To commit a transaction T, the coordinator performs a write operation to row store. Then, a log containing a set of writes will be shipped from row storage to column storage. Once writing in both two stores is finished, coordinators can finalize the transaction. Until the row store receives the acknowledgment of the commit message from the column store and T has been committed to it, it can send an acknowledgment of the commit message to the coordinator. Finally, the coordinator can notify the client about the success of T.

Pros and Cons. Transaction-based model is usually adopted in the row-native architectures (see Table 3). This model benefits development and management as correctness is provided in the transaction unit. That is, the storage layer in this model is not required to understand the semantics of transactions. Moreover, column stores in this architecture are fully in memory; thus, transaction-based synchronization will not incur much overhead on transactional processing. On the con-

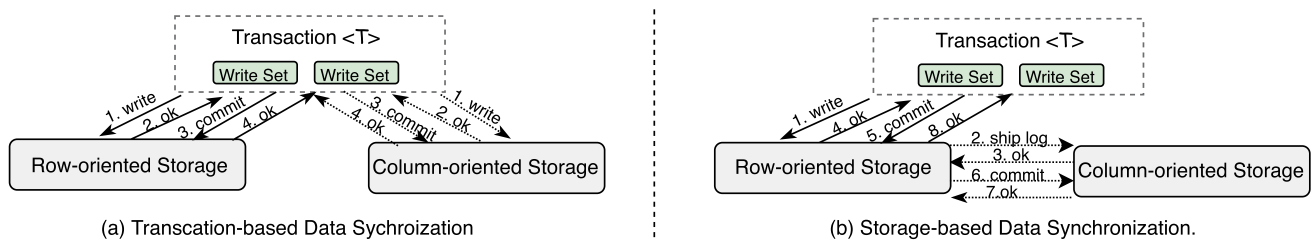

Fig. 7 Data synchronization model. This diagram shows how data is synchronized from OLTP storage into OLAP storage when a read-write transaction is scheduled. The steps with dotted lines can be processed asynchronously for better performance if the OLAP storage does not provide a strongly consistent guarantee

trary, the column-native, separated, and hybrid architectures usually adopt the storage-based model to decouple the data synchronization from transactional processing for better flexibility and scalability.

5.5.2 Data synchronization approach

Dual Write. Coinciding with the synchronization model, HTAP with row-native architectures adopt dual write to perform an atomic write operation in both row store (with in-memory buffer pool) and in-memory column store, which has been introduced in Sect. 5.5.1.

Delta Merge. For HTAP with column-native architectures, the databases continuously merge the new data generated from the in-memory row store to the persisted column store. To enforce efficient processing based on the storage-based synchronization model, a general optimization parses internal storage representation in multiple stages [105, 152]. For instance, SAP HANA [152] features two provisional in-memory delta storage (i.e., L1-Delta and L2-Delta) instead of a single row store. L1-Delta is a row store optimized for fast insert, delete, field update, and record projection. For performance reasons, L1-Delta does not perform any data compression. L2-Delta is organized in the column store format and employs dictionary encoding for better memory usage. Different from the persisted column store, the dictionary is unsorted. With the two provisional delta storage, a data update is first caught by L1-Delta. When the L1-delta absorbs updates more than a specific threshold (e.g., 10,000 rows), updates in the L1-delta are then merged into the L2-delta. Similarly, L2-Delta only merges the updates into the main store (i.e., persisted column store) when the number of updates exceeds the threshold. This workflow mitigates the overhead of transforming row-oriented tuples into column-oriented ones since the processing can be divided into multiple stages, and tuples can be batched.

Shared Log or Consensus. For separated architectures, the data synchronization approach deserves a more careful design to handle *Issue1*. BatchDB [111], F1 lightning [175], VEGITO [150], Janus [18], and Heatwave [123] ship updates via shared log. All committed transactional updates are saved in the logical logs (i.e., logs record read/write operations without depending on physical format). Logs are continuously shipped from the row store to the column store. A key challenge is guaranteeing the updates belonging to the same transaction can be consistently observed in the column store. Specifically, the partial view of an atomic transaction should never be observed by queries, and transactions' updates should be observed in the commit order.

To handle this challenge, BatchDB [111] groups transactions into batches, and the updates are only visible when a batch is successfully applied. To form a transaction batch, BatchDB assigns a batch ID to each transaction. The invariant is that a transaction can only belong to one batch ID, thus providing atomicity. Similarly, VEGITO [150] tags each transaction with an epoch id. The transactions within the same epoch should be applied and observed together. F1 Lightning [175] reuses the safe timestamp mechanisms of Spanner [50]. The safe timestamp indicates the watermark of visibility to OLAP. Each transaction in F1 is assigned a timestamp. The transaction can be visible if and only if the transaction's timestamp is smaller than the safe timestamp.

Another straightforward option is reusing consensus mechanism [77, 120] (e.g., state machine replication [147]). Consensus is essentially an abstraction of the shared log approach with clear, safe guarantees and a mature toolkit. Therefore, we classify them into the same category.

Wiser [21] seeks consensus protocols to achieve the serial order of writes. A query can only see updates made before it in a serializable order generated by the consensus (i.e., Raft [125]). TiDB [78] introduces Raft learners to improve the performance of data synchronization. A learner in Raft does not participate in leader elections, and log replication from the leader to a learner is asynchronous. Thus, adding more followers will not significantly impact the consensus group's performance because the leader does not need to wait for responses from followers. Also, to satisfy the hybrid format, data is transformed from row-oriented to column-oriented when data is synchronized to Raft learners.

Compared to shared logs, the benefits of leveraging a well-studied consensus protocol to realize data synchronization lie in three folds: proofed correctness, well-studied implementations, and strong compatibility.

Reform Format. For hybrid architecture, the database performs data synchronization by reforming data format directly. For instance, Proteus [6] changes the storage format by reading a consistent data snapshot into memory, and bulk loads the data into the respective storage format. The changes are generally triggered by a storage advisor continuously monitoring data access patterns.

Pros and Cons. The design choices of data synchronization largely depend on the underlying storage layer. The four data synchronization approaches mentioned in this sub-section are affiliated with their corresponding architecture. Multiple optimizations can be applied within each architecture to make synchronization more efficient.

In addition to minimizing performance overhead, a significant design goal is to achieve higher data freshness. The design of data synchronization approaches can also contribute to data freshness. For instance, batch-based log shipping may have worse data freshness than streaming shipping.

5.5.3 Consistency model

Given the sophisticated data synchronization models and approaches, the consistency model is worth revisiting. Different from the monolithic HTAP (Sect. 4), where the consistency model can be simply inherited (since each analytical query can be treated as a read-only transaction, e.g., in *MySQL*), the consistency model in hybrid HTAP may degrade. This is because data synchronization in hybrid HTAP breaks the consistency boundary between two data formats. To make it worse, data synchronization is practically done asynchronously to ensure a better performance isolation between OLTP and OLAP workloads.

Three of the foremost consistency models (a.k.a. isolation level) used in the database community are read committed, snapshot isolation, and serializability. To the best of our knowledge, most of the existing hybrid HTAP systems [6, 18, 92, 111, 150, 152, 175] provide snapshot isolation, even when transactional processing is promised with a more substantial consistency guarantee.

Under snapshot isolation, analytical queries can observe a complete data view (i.e., snapshot). As these HTAP databases usually provide a stronger model on OLTP, the degradation of the consistency model should be carefully noted. Application programmers or database administrators should handle the interleaving usage of transactions and analytical queries. A straightforward question is whether HTAP databases can ensure a stronger consistency model. The answer is absolutely yes, but maybe at the cost of performance and needs a careful design. Ensuring stronger consistency for HTAP in a lightweight manner is still an open problem.

For instance, TiDB [78] is a practical implementation that provides analytical queries to read the latest write made by transactions at the cost of longer latency. Specifically, it relies on the raft learner to calibrate and absorb all committed transactions from the raft leader before an analytical query can actually be scheduled. Obviously, this approach produces additional overhead for the raft leader and needs to block analytical queries for a while. PolarDB-IMCI uses an asynchronous replication mechanism and can ensure different consistency models (e.g., strict serializability) through the proxy layer. The proxy node keeps track of the log sequence numbers of both row stores and column stores and may only route queries to the column store when the log sequence number of the column store is not less than the log sequence number of the row store.

5.5.4 Delta buffer and meta-data

Recall *Issue3*, due to the gap in writing efficiency between the row and column store, when the write rate in the row store is high, the column store may not be able to absorb all updates timely. To make things worse, this lagging will continuously accumulate. Thus, to bridge the efficiency gap, an HTAP DBMS commonly maintains a delta buffer and meta-data for recent transactions and updates. An overview of the delta buffers and meta-data adopted by the existing hybrid HTAP systems is shown in Table 3. In the next, we describe these designs in more detail.

Without introducing additional physical storage space, row-native architecture can directly treat row stores as a delta buffer for column stores. This is because row and column stores are always co-located in this architecture. As such, to perform a data update, dual-write may not finish the column store's writing process immediately. Instead, it only needs to update a lightweight meta-data. The meta-data distinguishes whether an update has already been merged into the column store or is only resident in the row store. Then, the actual format changes can be done in the background. To serve an analytical query, the executor fetches most data from the column store and a small portion of unmerged updates from the row store. The trade-off is also significant: the reading process is more expensive.

A typical implementation is used by SQL Server [93]. SQL Server uses Tail Index as its meta-data, which covers rows not yet included in the column store. Inserting a new row or row version into the table consists of inserting the row into the in-memory Hekaton table (i.e., its row store) and adding it to all indices, including the Tail Index (i.e., the meta-data). Then, a background task will eventually copy the data to the CSI (i.e., its column store). Deleting a row from the table consists of first locating the target row in the Hekaton

table and deleting it from there. If the column store contains a valid value, a row with its row-id is inserted into the Deleted Rows Table, which logically deletes the row from the column store. An update operation can be implemented as an insert follows deletion. To update a row, it is first updated in the Hekaton table, and the new version is also included in the Tail Index but not immediately added to the column store. If an old version is included in the column store, its row ID is masked and deleted in the Deleted Rows Table to logically delete the old version from the column store.

Oracle Dual [92] and AlloyDB [65] do not index the updates directly. Instead, they create a transaction map as its meta-data to record the committed transactions that do not appear in the column store. The transaction map will record its transaction ID and write sets whenever a transaction is committed in the row store. After the updates of a committed transaction have been successfully merged into the column store, the entry with its transaction ID is removed. Compared to the Tail Index in SQL Server, they simplify the data modification process (insert, update, and delete) at the cost of reading performance.

PolarDB-IMCI [168] adopts a different approach. Instead of leveraging the row store as its temporary delta buffer, it performs the data writes directly to the column store when a transaction is ready to commit. To alleviate the problem with writing efficiency, the column store is append-only, and all updates to PolarDB-IMCI are out-place. PolarDB-IMCI moves the task of column encoding, data compression, and sorting to a background process. Thus, data modifications on the column store do not incur additional overhead compared to the row store. Inserting a new row in PolarDB-IMCI's table is straightforward. It includes a standard insert in the row store and an append-only insert in the column store. Like SQL Server, deleting a row in the column store is conducted by adding a delete mask to the bitmap (i.e., the meta-data). For an update operation, without performing an in-place update in the column store, the update is executed by a delete in bitmap and an append-only insert.

Fundamentally speaking, the row store is essentially a delta buffer in column-native architectures. Since data modifications (i.e., inserts, updates, and deletes) are periodically merged into the persisted column store, the column store always lags behind the row store. The tricky thing is that, like a generally designed delta buffer, a row store does not store the complete data copy and only contains a small portion of incremental data. Thus, the overall scan performance will not be largely influenced by the sub-efficient scan on the row store. Meta-data in this architecture enhances performance on the delta buffer.

For instance, SAP HANA [152] adopts a secondary index structure (named delta index) on its L2-delta (i.e., its delta buffer) to support access patterns of point query optimally. NoisePage [105] uses a traditional B-tree on its delta buffer. MemSQL [153] uses skiplist as its memory-optimized delta indices. Skiplists can use pointers to rows directly since the delta buffer is fully in-memory. This makes it different from the rows-native architecture indices: it avoids the indirection caused by page tables.

For separated architectures, persisted row store and column store are stored independently. They can be independently stored in a single persisted storage or separated into different machines. As such, to handle the problem efficiency gap, a typical solution is allocating additional storage space and co-locating it with the column store to serve as a delta buffer. Meta-data in this architecture serves the same target as column-native architecture.

TiDB [78] proposes DeltaTree (an optimized variant of log-structured merge-tree (LSM-tree) [127]) as its delta buffer. Specifically, DeltaTree consists of a delta space and a stable space. In delta space, data modifications are recorded in append-only logs with a serialized order, achieving near-optimal write efficiency. However, these modifications are usually stored in many small files and induce large IO overhead when read. To enhance read performance, DeltaTree continuously compacts these small records into a larger one, then flushes the larger logs into stable space. Note that the separation of delta space and stable space is designed to realize the same goal as L1-Delta and L2-Delta in SAP HANA (Sect. 5.5.2), i.e., dividing the heavy merge process into multiple stages, and thus amortize the merge overhead to both read and write operations.

TiDB also includes an important mete-data: delta index. Delta index is developed to accelerate read operations, and the implementation is based on B-tree. Briefly, reads on log-structured storage require a sequential logs scan to find the latest data version. Delta index speeds up the sequential scan by recording a valid but may not the latest data version and its location in delta space for each index entry. Then, read operations can leverage the delta index to skip old versions and start the scan process from the pointed location instead of a scan from beginning to end.

F1 Lightning [175] adopts another data structure for its delta buffer: PAX [9]. In PAX, rows are first divided into row groups and then stored column-wise within a row group (see Sect. 4.2). PAX can exhibit superior cache and memory bandwidth utilization than a purely row-oriented data format, favoring the performance of retrieving new data. The index of PAX contains a sparse B-tree index on the primary key where the leaf entry tracks the key range of each row group. Though the data modifications on PAX are not as efficient as LSM-tree, they practically mitigate the write efficiency problem. Therefore, it balances the performance between range scan and point lookup.

Hybrid architectures do not rely on delta buffer and meta-data directly. However, the format change indeed incurs additional overhead. An adaptive threshold is embedded in

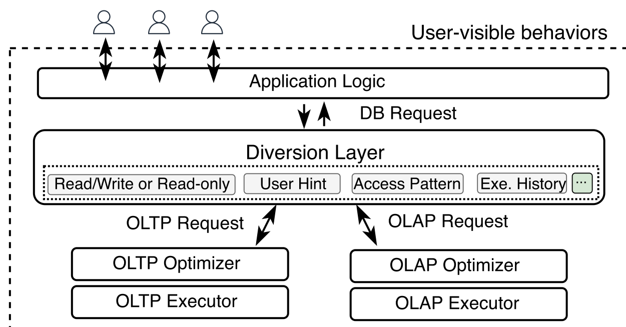

Fig. 8 An example of how the HTAP systems generate execution plans for both OLTP and OLAP requests (Diversion Approach)

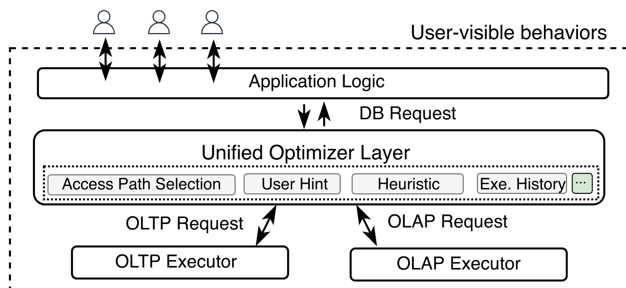

Fig. 9 An examples of how the HTAP systems generate execution plans for both OLTP and OLAP requests (Unified Optimizer)

the storage advisor [6, 19] to avoid changing format frequently.

Pros and Cons. Multiple data structures can be used for delta buffer, e.g., row-oriented format, LSM-tree, and PAX. Overall, these data structures perform better writing efficiency than column stores, thus filling the write efficiency gap between row stores and column stores. On the other hand, they trade-offs among write performance, range scan performance, and point query lookup performance. Meta-data is delicately designed on a delta buffer to optimize further read/write performance. They either mark the unmerged data modifications or index the incremental data. Nevertheless, maintaining meta-data increases the complexity of systems.

5.5.5 Performance isolation

In HTAP, data synchronization (Sects. 5.5.2, 5.5.1), delta buffer (Sect. 5.5.4), and meta-data (Sect. 5.5.4) make it challenging to realize fine-grained performance isolation between OLTP and OLAP. A summary of the existing solutions adopted by HTAP systems is shown in Table 3.

Instance level isolation is a native property that separates different workloads into different instances. An instance can either stand for a stand-alone physical machine or a virtualized resource group (e.g., virtual machine [64], container [70]). Separated architectures (e.g., TiDB, Heatwave, F1 Lightning) can embrace this isolation naturally. PolarDB-IMCI [168] achieves this isolation by dividing the read-write and read-only nodes. By doing so, the OLAP workload will only be scheduled for the read-only nodes.

The HTAP systems that share a single instance for both OLTP and OLAP workloads deserve a more careful design on performance isolation, which shares the same issue with the monolithic HTAP (see Sect. 4.5). Several methods are proposed to isolate CPU, memory, I/O bandwidth, and network traffic usage. We illustrate each of them below.

CPU Isolation. Binding CPU core [58, 110], thread pool [6], and variants are common approaches to separate CPU resource. They can be implemented either in the programming language layer (e.g., goroutine pool in Go [163]) or in the OS kernel layer (e.g., the control group in Linux kernel [71]).

Memory Isolation. Memory isolation is achieved by adaptive scheduling or configuring a fixed quota for memory usage. For instance, Greenplum [110] utilizes a memory scheduler in the database kernel to control memory usage among different workloads. TiDB [78] limits the default access table size on TiKV for analytical queries to 500 MB.

I/O Bandwidth and Network Traffic. Ideally, the isolation of I/O bandwidth and network traffic can be achieved in the same way as memory isolation. However, it becomes nontrivial if I/O bandwidth and network traffic are consumed by background processes asynchronously [110], e.g., asynchronous data merge (Sect. 5.5.4) and asynchronous data synchronization (Sect. 5.5.2). Worse still, such bandwidth-consuming processes may have complex dependencies with each other, making it an open problem and continuously calling for solutions.

5.5.6 Query optimization and execution

Recall that one of the most important design principles of HTAP is providing users (applications) with OLTP and OLAP in a single database system using a transparent interface (see Sect. 3). Thus, it demands a unified interface for both transactions and analytical queries. Expressly, when a user (application) submits a database request to the HTAP system, the database should take the duty to optimize the request, execute the requests, and generate the results automatically (i.e., *Issue5*).

To the best of our knowledge, there exist two approaches (Figs. 8 and 9). In the first approach (Fig. 8, diversion approach), database requests are diverted into OLTP and OLAP requests using a routing-based approach. After receiving user requests, they rely on embedded user-level hints or a middleware layer to differentiate OLTP and OLAP queries. Then, they execute queries on the desirable engines and stores. Typical implementations are PolarDB-IMCI [168] and ByteHTAP [39].

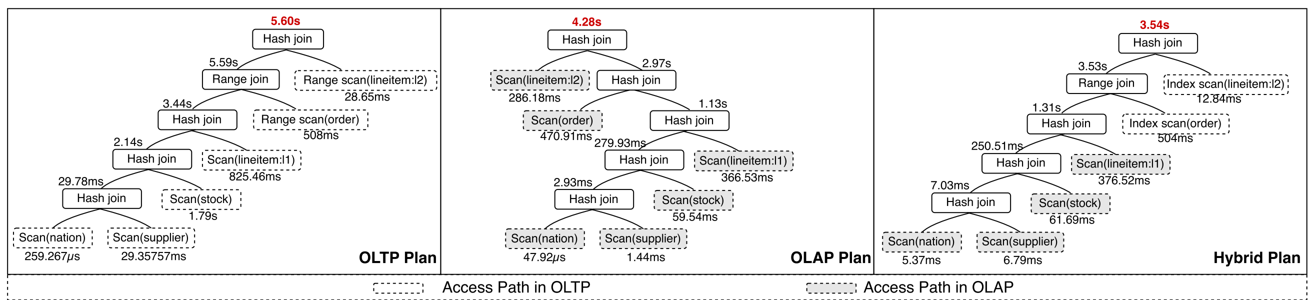

Fig. 10 Hybrid plan. An example of the hybrid query plan generated by an anonymous HTAP system for TPC-H Q21

Table 4 A Comparison of Physical Operation in OLTP and OLAP

	An OLTP Engine (MySQL v8.0.18 [122])	An OLAP Engine(ClickHouse v22.6.3.35 [45])
Data Source	Index-only Scan	Sequential Column Store Scan
	Index Scan	Dictionary
	Sequential Row Store Scan	Materialized View
	Materialized View	–
Join Algorithm	Hash Join (Centralized)	Hash Join (Distributed [Broadcast, Shuffle, Co-locate])
	Loop Join (SNL, BNL, INL)	Merge Join (Sort-Merge)
Sort Algorithm	Index Order, File Sort	Column Order, File Sort

A case study on the physical operators with stand-alone OLTP (i.e., MySQL) and OLAP (i.e., ClickHouse) implementations regarding data source, join algorithms, and sort algorithms

Another approach handles the diversion problem with a unified optimizer (Fig. 9). The optimizer can generate both OLTP and OLAP query plans. Thus, the diversion (i.e., choosing an appropriate executor and store format) can be made inside the optimizer, thus driving the request to its desired execution engine based on request properties (i.e., using OLTP executor for point reads and writes and using OLAP executor for large scans).

We introduce several implementations belonging to this approach. For instance, F1 Lightning [175] generates logical plans using its F1 optimizer (i.e., an optimizer designed for OLTP) and considers lightning-only indexes and views during physical planning. SQL Server [58] analyzes and recommends column stores by its Database Engine Tuning Advisor (DTA) when suitable for a given workload. TiDB [78] extends query optimizer to explore physical plans accessing both row and column stores. Oracle Dual [92] supplements column indices to its optimizer as an alternate execution method for high-speed table scans.

Pros and Cons. The advantages of the first approach are that it can reuse the well-developed optimizers and executors and independently absorb the advanced technologies from the OLTP and OLAP community. However, it is non-trivial to distinguish the requests before they have been planned, optimized, and executed, especially for read-only transactions (e.g., point queries) and analytical queries (e.g., queries with frequent scans). The second approach can leverage the knowledge of the optimizer for division and thus can be more precise when compared to the first approach. Moreover, it provides the opportunity for hybrid plans, which we will discuss in the next subsection.

5.5.7 Hybrid plan

A hybrid plan leverages row store, column store, OLTP, and OLAP executors for a single query. It has the potential to generate a more efficient execution plan for HTAP. The fundamental motivation behind the hybrid plan is that row store with specific indices (e.g., clustered B+ tree index) is more suitable for short-range scans than column store [58] since the index allows efficient point and short-range lookups. Recent studies [58, 87] point out that access path selection in modern databases should be based on a selectivity threshold, along with the underlying hardware properties, system design parameters, and data format. Thus, the row store may be optimal for a portion of sub-queries in a complex analytical query, and the column store may be optimal for the others. Naively binding analytical queries to column store

and OLAP executors leaves much potential for the performance of HTAP systems unrealized.

In addition to the data access path, many optimized query operator implementations rely on underlying storage characters. For example, the index-merge join (or sort-merge join) algorithm requires the data source to be ordered by the join key; otherwise, it will incur a computation-heavy sorting phase. Thus, an available sorted data source (either in row store or column store) may potentially speed up the query execution. Table 4 provides a case study on the physical implementations of query operators in two popular OLTP and OLAP databases (i.e., MySQL and ClickHouse).

To the best of our knowledge, several HTAP systems have implemented hybrid plans to optimize the performance of OLAP. For instance, TiDB [78] complements column store as an alternative access approach in the cost model and lets the query optimizer choose the access approach that has the lowest estimated cost during the optimization. Specifically, the query optimizer can choose both row and column stores to access different tables in the same query, while it does not consider retrieving the same table from different data sources. SQL Server [59] develops a Database Engine Tuning Advisor (DTA) to recommend a suitable combination of B+ tree (i.e., the row store) and column store indexes for a given workload. After a column store index is built, the SQL server uses the column store indexes in its optimizer simply as other secondary indexes.

In addition, HTAP databases using hybrid architecture have to deal with hybrid plans by nature, as they do not hold a full data copy for either row or column. For example, Proteus [6] generates physical execution plans based on its workload models and the current storage layout and reuses past decisions to accelerate these processes.

We show a practical example of the hybrid plan generated by TiDB in Fig. 10. The query plan is for TPC-H [52] Query 21 with a scale factor = 100. We ran the query using the same parallelism (i.e., 16 threads) in the database engines and thus used similar CPU and memory resources among the three candidate plans (i.e., OLTP plan, OLAP plan, and hybrid plan). Overall, the hybrid plan has the potential to achieve a better execution time (3.54 s) than both the OLTP plan (5.60 s) and the OLAP plan (4.28s).

Unsurprisingly, scans of TABLE nation, supplier, stock, lineitem:l1, and order on column store are significantly faster than row store. However, as for TABLE lineitem:l2, the scan performance on the row store is better, i.e., 28.65 ms compared to 286.18 ms on the column store. This is because the query's selectivity for TABLE lineitem:l2 is high (i.e., a theta join), and a sorted index exists for the accessed column. Thus, it incurs smaller CPU, memory, and I/O consumption compared to a sequential scan on a column store.

Moreover, due to the sorted order, if the TABLE order is accessed on the row store, it can be joined by a range join algorithm instead of a hash join algorithm, which can be potentially more efficient. Thus, in the hybrid plan, even though the time for a scan on the row store (504.00 ms) is slightly longer than the column store (470.91 ms), the optimizer should still select the row store for its data source.

Nevertheless, there is no free lunch. With hybrid plans, the optimizer's search space is much larger, especically when both row and column store are available. Actually, even though the available options in a traditional OLTP or OLAP database are much more limited, they still suffer from a long optimization time for complex analytical queries [99, 114, 115]. Heuristics must be adopted to prune the search space to keep optimization times within reasonable bounds [58].

To our knowledge, existing solutions have not yet well explored these aspects and leave them as their future work (as admitted in [59]). Our vision is that the emerging machine-learning-assisted optimization approach (a.k.a learned optimizer) may help with optimizers for handling the complexity and pruning challenges since many of them have achieved excellent success on typical OLAP-only workloads (to be further illustrated in Sect. 8). Meanwhile, the models used for hybrid architecture (e.g., the learned decision in Proteus) may also be helpful in finding heuristics.

5.5.8 Sharding strategy

Partitioning databases into multiple data shards can improve query processing performance, increase database manageability [1], and achieve high scalability. Silimar to format design, which groups data into storage blocks (e.g., disk pages), sharding strategies also abstract the way of data grouping. The key difference is that the format design specifies the smallest unit to organize data while sharing strategies consider scaling the organized data into multiple machines or data sites. Briefly, there are three primary ways of partitioning a relational database: horizontal, vertical, and irregular. We show a comparison of the three strategies in Fig. 11.

Each partition is a separate data store in horizontal partitioning, and all partitions share the same schema. This partitioning is general to row store, as a row is the smallest unit in a horizontal partition. For vertical partitioning, each partition holds a subset of the columns in the data store. Thus, column stores can benefit from this partition schema when two commonly accessed columns are located in the same partition. In addition, irregular partitioning is typically adopted by the HTAP systems with a hybrid architecture (e.g., Proteus [6]). Irregular partitioning does not shard the database at the granularity of rows or columns but extends the existing abstraction of hybrid architecture by spanning hybrid formats into multiple partitions. Similar to the adaptive policy of working on a single partition, it adaptively shifts data across partitions based on workload perspective.

Fig. 11 Comparison of sharding strategies. This figure is a reprint of Fig. 1 in [82]

Excluding using an irregular partition strategy for HTAP with a hybrid architecture, which essentially blurs the boundary of row and column stores, it's challenging to draft a consistent and efficient partitioning strategy for the two types of data formats (i.e., rows and columns). Existing strategies can be classified into two categories: symmetric and asymmetric.

Symmetric partitioning adopts the same partition strategy for both row store and column store. Thus, each data partition in the column store can be precisely mapped to a data partition in the row store, simplifying data management by eliminating cross-partition data synchronization. Typical implementations include VEGITO [150] and TiDB [78].

On the other hand, asymmetric partitioning has the potential to generate a more efficient partitioning strategy by customizing the strategy for each data store and workload independently. For instance, SAP HANA [96] maintains a row store in a single physical machine without partitioning, while its column store is independently partitioned and distributed across multiple smaller physical machines. This enables it to avoid the cross-partition two-phase commit [119] (2PC) for OLTP workloads while serving partition-friendly OLAP workloads in a more scalable way. Janus [18] also uses horizontal partitioning for the row store and vertical partitioning for the column store by default.

6 Discussion and lessons learned

In the previous sections, we have examined the design of different HTAP systems. In this section, we aim to shed light on the reasons why these designs persisted and provide coarse-grained design guidelines for newcomers.

Sincerely, most of the existing HTAP systems have evolved from legacy OLTP or OLAP architecture to serve their needs of HTAP, and their evolution routes diverge from the root. For instance, SQL Server, Oracle, and PolarDB-IMCI start from an existing OLTP system and strive to provide a plug-in OLAP acceleration using columnar data formats. In turn, SingleStore and SAP HANA start from a current OLAP system and try to provide essential OLTP performance over the column store. Separated design bridges two existing OLTP and OLAP systems using a similar approach as real-time ETL (see Sect. 2.1). The difference is that for better data freshness, the separated design implements the data synchronization inside the database and uses system-specific design at the cost of sacrificing versatility. The hybrid architecture seems to be the best native approach for handling HTAP. However, there are several open problems when deploying them into the industry. First, the performance of such a design heavily relies on the algorithms that reform the data formats, which inevitably incurs instability in complex real-world applications. Second, debugging and management of such a design is always tricky. Third, it introduces a fair complexity in managing secondary indexes since the indexed data may have different formats and redundant copies.

To guide an HTAP system designer in selecting data formats and architectures from scratch, we summarize our main findings concerning the pros and cons of different architectures in Table 5. This is a coarse-grained guideline, and practical databases are much more complex than our abstraction. The data format and architecture choice can be important, but not all. Specific optimizations can also contribute to the overall performance.

7 Applications and benchmarks

Thus far, we have reviewed the different HTAP architectures and their unique challenges and solutions. We now briefly discuss HTAP applications and benchmarks.

7.1 Real-time applications

Real-time applications drive the development of HTAP. In addition to bridging existing OLTP and OLAP applications, several cutting-edge real-time applications are proposed in the literature, including analytical applications, dashboards, and real-time prediction using machine learning.

For instance, fraud detection applications [36, 135, 136] analyze the continuously generated transactions to prevent

Table 5 A comparison of different formats and architecture choices in various aspects

	Txn Perf	Query Perf	Storage Overhead	Data Freshness	Perf. Isolation	Scalability	Perf. Stability	Complexity
Monolithic	Good	Limited	Small	High	Poor	Poor/Medium	Poor	Low
Row-native	Good	Medium	Medium	High	Medium	Medium	Medium	Medium
Column-native	Limited	Good	Medium	High	Medium	Medium	Medium	Medium
Separated	Good	Good	High	Medium	Strong	Strong	Strong	Medium
Hybrid	Medium	Medium	Small	High	Poor	Poor/Medium	Poor	High

money or property from being obtained through false pretenses. System monitoring applications [46, 170] derive real-time system metrics swiftly based on data logs. Social trading [129, 176] applications offer a sense of trading community to traders by monitoring the influx of retail traders. Innovative industry applications [134, 174] automate the management of manufacturing and supply chains by conducting timely decisions based on fresh information.

A thorough review of these applications is beyond the scope of this paper; however, we list their desirable requirements for data freshness and consistency in Table 6 and several key inspirations below. First, automatic decision-making applications desire data freshness as high as possible. In contrast, human-in-the-loop applications desire data freshness close to the boundary of human reaction time (i.e., 150–300 ms). Second, mission-critical applications (e.g., fraud detection) have stringent consistency requirements for generating analytical results with high quality, while routine analysis services may tolerate some inconsistent data. This suggests that the consistency model of a full-fledged HTAP system should be configurable, further influencing the design of HTAP (e.g., the data synchronization approaches).

7.2 Benchmarks

As more and more HTAP systems have been developed, a benchmark with representative data schema and workloads is essential for evaluating existing system designs and directing future development. For this purpose, several HTAP benchmarks are proposed in the literature (see Table 7).

Most existing HTAP benchmarks mix the OLTP and OLAP workloads from existing ones. We observe that these benchmarks usually target a much simpler querying scenario than a benchmark traditional data warehouse (e.g., TPC-DS).

Specifically, many existing HTAP benchmarks use TPC-C [53] for OLTP workloads and TPC-H [52] for OLAP workloads. Among them, CH-benCHmark is one of the most representative ones, which is widely adopted by the existing HTAP systems, e.g., BatchDB [111], TiDB [78], VEGITO [150], etc. In particular, CH-benCHmark [48] unifies the schema of TPC-C and TPC-H by keeping the TPC-C schema unmodified and adding some necessary tables to fulfill equivalent queries from TPC-H. The benchmark can be scaled by partitioning a database into multiple warehouses. Each warehouse is a data shard and can be deployed across multiple physical machines or data sites. Generally, CH-benCHmark keeps transactions and queries unmodified from the TPC-C and TPC-H and thus has a similar complexity as the well-studied benchmarks.

HTAPBench [47] follows a similar schema design, confining the schema from TPC-C and TPC-E. Additionally, HTAPBench proposes a new evaluation metric: *QpHpW*. Instead of evaluating transactions and queries using inde-

Table 6 HTAP applications

Real-time HTAP Applications	Freshness (ms)	Consistency
Fraud Detection [36, 135, 136]	∼20	strong
System Monitoring [46, 170]	∼20	strong
Innovative Industry 4.0 [134, 174]	∼20	strong
HealthCare [42, 159]	∼100	strong
Online Gaming [165]	∼100	medium
Stock Price Monitor [43, 91]	∼200	medium
Dynamic Pricing [40, 56]	∼200	medium
E-commerce Intelligence [93, 143]	∼200	medium

A summary of the cutting-edge real-time applications with stringent data freshness and consistency requirements. Human reaction time takes 150–300 ms on average

pendent metrics (e.g., throughput for OLTP and latency for OLAP), *QpHpW* calculates the throughput of transactions and analytical queries in the same equation. As such, *QpHpW* can be a representative metric for HTAP. Same as CH-benCHmark, HTAPBench keeps transactions and queries unmodified from the TPC-C and TPC-H.

HTAPTrick [117] bridges TPC-C and TPC-H with a star schema (i.e., TPC-H SSB [128]). Star schema provides a clear division between dimension tables and fact tables, where dimension tables are most likely to join fact tables with their primary keys. As star schemas can provide performance enhancements for read-only reporting applications, they have become popular in the OLAP community. HTAPTrick follows this design to explore real-world use cases. To run TPC-C on star schemas, HTAPTrick rewrites the transactions while keeping the logic unmodified. As such, HTAPTrick has the same complexity as TPC-C and TPC-H, while the queries can result in improved performance for aggregation operations (e.g., Join) thanks to the star schema.

TPC-HC [158] is another benchmark over TPC-C and TPC-H. Different from the aforementioned benchmarks, TPC-HC uses the schema directly from TPC-H and integrates TPC-C transactions by rewriting them and tailoring them to the schema of TPC-H. By doing so, the transactions of TPC-HC are simpler than TPC-C, while the OLAP queries are the same as TPC-H.

CBTR [25] is a composite benchmark for evaluating order-to-cash applications. CBTR is based on a global enterprise's data set and thus targets real-world workloads. Nevertheless, to our knowledge, CBTR is not open-sourced.

Besides mixing OLTP and OLAP workloads for building HTAP workloads, several studies [63, 103, 126, 130] feature that HTAP transactions should be able to contain analytical operations inside transactions. These benchmarks target to model a widely observed behavior pattern: making a quick decision while consulting real-time analysis. To our knowledge, Kang et al. has proposed a benchmark called Olxpbench, where a real-time query is embedded between transactions. Overall, the benchmarks contain three domain-specific schemas and workloads.

Due to the integration of query operators, the transactions in Olxpbench are much more complex and have a larger execution, which may lead to a big contention window (i.e., the period to conflict with other transactions). The complexity of OLAP remains unmodified as its initial benchmark. For interested readers, the OLAP queries in Subenchmark are the same as queries in TPC-H. Fibenchmark involves many scan-intensive queries while fewer joins. The OLAP queries of Tabenchmark have moderate scans and join complexity.

8 Future directions

Although many novel technologies and architectures have been proposed to make HTAP practical, some further opportunities and challenges exist. In this section, we outline and discuss some prospective and interesting research directions.

8.1 Software and hardware co-design

Co-designing software and hardware in HTAP is currently on the rise due to the increased adoption of accelerators (e.g., FPGAs and GPUs). There are several existing works worth noting. Polynesia [30] proposed an energy-efficient in-memory HTAP implementation, which leverages hardware and software co-designed components (called islands). Each island is specialized for specific types of queries (either for OLTP or OLAP). In particular, islands cooperate with processing-in-memory (PIM) hardware and specifically optimized algorithms to speed up the computation.

In addition, GPU databases accelerate queries by dividing query processing into multiple fine-grained data-parallel GPU tasks. Several works [16, 98, 139] have been contributed to exploring such a heterogeneous HTAP database by executing OLAP queries on GPUs. However, there are still some open problems. For instance, the scalability of heterogeneous HTAP is limited, and the interleaving between CPU/GPU

Table 7 HTAP benchmarks

Benchmarks	Originated From	Scenarios	Tables	Columns	Indices	Txns	Queries
CH-benCHmark [48], HTAPBench [47]	TPC-C [53], TPC-H [52]	E-commerce	12	106	3	5	22
TPC-HC [158]	TPC-C [53], TPC-H [52]	E-commerce	8	61	0	3	22
HATrick [117]	TPC-C [53], TPC-H (SSB) [128]	E-commerce	6	63	1	3	13
CBTR [25]	-	Order-to-cash	4	28	0	11	5
Olxpbench-Subenchmark [83]	TPC-C [53], TPC-H [52]	E-commerce	9	92	3	10	9
Olxpbench-Fibenchmark [83]	Small Bank [13]	Banking	3	6	4	12	4
Olxpbench-Tabenchmark [83]	TATP [51]	Telecom	4	51	5	13	5

A summary of existing HTAP benchmarks and their characters

processing is complex. Note that the interleaving between CPU/GPU can be unavoidable when an analytical operator is embedded in a transaction (see Sect. 7.2).

Other hardware, such as persistent memory, secure hardware, and remote direct memory access NICs, may also be worth attention. How to develop HTAP systems on this new hardware remains unexplored.

8.2 Serverless and cloud-native HTAP

The advent of new serverless and cloud-native architectures has introduced new opportunities for HTAP. The benefits of adapting HTAP systems to cloud-native architectures include elastic resource scheduling, independent fault-tolerance between storage and computing, stringent service-level agreement (SLA) to ensure reliability, and easy manageability with reduced operational cost [101].

There are several cloud-native HTAP systems that have been proposed. For instance, MemSQL [131] (renamed as SingleStore in 2022) absorbs two key innovations from the cloud-native designs to shift its HTAP database. First, it separates the storage and computing layers to provide excellent resource elasticity in both of the two layers. Second, it unifies its table storage for row and column storage to mitigatcost of storing redundant data copies. PolarDB-IMCI [168] is another example that follows the separation of storage and computing layers. By adopting such a design, PolarDB-IMCI is ablcan up in seconds and scale out in minutes to handle workload spikes of OLAP. Moreover, it introduces a new checkpoint mechanism for fast recovery in case of the single point of failure using cloud facilities.

Although these existing explorations are exciting, the design choices are based on their unique architecture, leaving other aspects unexplored (e.g., the cloud-native architecture for monolithic HTAP). Furthermore, memory disaggregation has become more and more popular nowadays, especially because the next generation of in-datacenter networks can be ultra-fast (e.g., powered by CXL). Generally, the disaggregated memory architecture detaches CPUs and memory by abstracting them into resource pools to reduce the total cost of resource ownership in a cloud-native environment. How to leverage this concept to guide the design of HTAP is still an open problem.

8.3 Beyond column store

As far as we consider, column store is treated as the default data format complements to row store. Actually, although real-world applications usually prefer row stores for transactions due to the given relational model, they may desire another specialized data format for analytical queries. For instance, complementary to column store, databases with novel data models are increasingly popular for analytical

workloads, e.g., graph databases [15, 33], spatial databases [76, 173], and time series databases [24, 140]. To absorb the advancements of these new databases, a well-customized HTAP system demands a re-design of data synchronization approaches, delta buffer, meta-data, execution engine, and query optimizer.

To our knowledge, several works have explored customized HTAP designs for graph databases. Jibril et al. proposed a new data synchronization approach when building HTAP over graph database. In particular, they use an update-friendly table in an optimized sparse matrix format to propagate transactional updates. Another work introduces GART [151], performing dynamic graph analytical processing tasks on the datasets generated by relational OLTP. In particular, GART customizes an efficient and mutable compressed sparse row representation for graph scans and proposes a coarse-grained (MVCC) scheme to reduce the temporal and spatial overhead of the version. These issues can be specific to graph HTAP, which may not be studied in existing HTAP systems that use column stores.

8.4 Optimizer for HTAP workloads

Designing optimizers in an HTAP database is challenging. As evident in Sect. 5.5.7, the extension of both data source and operators is accompanied by an exponential increase in plan search space. Thus, efficient heuristics to prune the search space are highly desirable. Existing methods [58, 78] only consider leveraging cost functions to select the access path (i.e., data source) while neglecting store-specific query operators (see Table 4). Additionally, as asynchronous data synchronization causes a visibility delay between the row and column stores, row stores in HTAP always have better data freshness than column stores. As such, interesting trade-offs should be made to achieve the best timeliness of queries: accelerating OLAP queries using column stores or retrieving the required data from row stores for better freshness.

In addition, learned optimizers [114] have attracted a lot of research interest and shown practical gains by learning a mapping between an incoming query and the execution strategy [103]. For instance, Bao [113] steers query optimizers using reinforcement learning. Specifically, Bao automatically learns from the changes in query workloads, data, and schema and provides per-query optimization hints.

In HTAP, how to efficiently utilize those learning algorithms is unclear. Unlike predicting query latency on a static data copy, ongoing transactions may influence the latency due to physical and logical resource contention (e.g., latches on a data object). To forecast the best query plan, a learned optimizer may need to learn the properties of OLTP workloads before estimating the cost.

As pointed out by previous papers [177], learning-based optimizers cannot work well for dynamic workloads. Some recent works also consider combining learning with the existing cost models. For instance, Yu et al. propose a hint-based candidate generation method that leverages the learning-based method to generate highly beneficial hints and uses a cost-based method to supplement the hints to generate complete plans as candidates. Yang et al. takes a similar approach while identifying important parameters within the cost model and using a fast-learning model to adjust them for each specific hardware and software configuration.

In our vision, combining the learning approach with the traditional cost model (as well as statistics) can be a moderate solution for HTAP optimizers.

8.5 Learned data format

Recall the design of HTAP with a hybrid architecture. A system with a hybrid architecture can efficiently learn physical format design, thus blurring the boundary of row and column stores. Existing approaches (e.g., [6, 7]) learn the cost of data accesses and predict their latency under different storage formats. For instance, Tiresias [7] makes predictions by collecting observed latencies and access histories to build predictive models in an online manner, enabling autonomous storage and index adaptation.

To our knowledge, these learning algorithms are commonly used for hybrid architecture and have not been well explored in other architectures. For instance, for row-oriented architecture, systems may also need an advisor to suggest which column (or segmentation of the column) can be most valuable for constructing an index when the memory space can be limited. We regard these interesting explorations and development as future works.

9 Conclusion

Hybrid transactional/analytical processing (HTAP) is an increasingly important subject of research and development. It introduces massive technical challenges and opens many opportunities to the database community.

In this paper, we systematically review the existing HTAP architectures. We summarize the common HTAP-specific issues and challenges in implementing an efficient HTAP system. We also compare different design choices based on the assumption of their underlying architecture and discuss how the proposed methods can handle the aforementioned challenges. Moreover, we discuss the cutting-edge applications, benchmarks, and future directions to push forward future research in this area.

Acknowledgements We thank all anonymous reviewers for their valuable comments. The work is supported in part by National Key R&D Program of China (2022ZD0160200), HK RIF (R7030-22), HK ITF (GHP/169/20SZ), the Huawei Flagship Research Grants in 2021 and

2023, HK RGC GRF (Ref: HKU 17208223), the HKU-SCF FinTech Academy R&D Funding Schemes in 2021 and 2022, and the Shanghai Artificial Intelligence Laboratory.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Abadi, D.: Data Partitioning, pp. 599–600. Springer, Boston (2009)
2. Abadi, D., Babu, S., Özcan, F., Pandis, I.: Sql-on-hadoop systems: tutorial. Proc. VLDB Endow. **8**(12), 2050–2051 (2015)
3. Abadi, D., Madden, S., Ferreira, M.: Integrating compression and execution in column-oriented database systems. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 671–682 (2006)
4. Abadi, D.J., Madden, S.R., Hachem, N.: Column-stores vs. row-stores: how different are they really? In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 967–980 (2008)
5. Abadi, D.J., Myers, D.S., DeWitt, D.J., Madden, S.R.: Materialization strategies in a column-oriented dbms. In: 2007 IEEE 23rd International Conference on Data Engineering, pp. 466–475. IEEE (2006)
6. Abebe, M., Lazu, H., Daudjee, K.: Proteus: Autonomous adaptive storage for mixed workloads. Technical report, University of Waterloo. https://cs.uwaterloo.ca (2022)
7. Abebe, M., Lazu, H., Daudjee, K.: Tiresias: enabling predictive autonomous storage and indexing. Proc. VLDB Endow. **15**(11), 3126–3136 (2022)
8. Agrawal, N., Vulimiri, A.: Low-latency analytics on colossal data streams with summarystore. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 647–664 (2017)
9. Ailamaki, A., DeWitt, D.J., Hill, M.D.: Data page layouts for relational databases on deep memory hierarchies. VLDB J. **11**(3), 198–215 (2002)
10. Ailamaki, A., DeWitt, D.J., Hill, M.D., Skounakis, M.: Weaving relations for cache performance. VLDB **1**, 169–180 (2001)
11. Akal, F., Böhm, K., Schek, H.-J.: Olap query evaluation in a database cluster: a performance study on intra-query parallelism. In: East European Conference on Advances in Databases and Information Systems, pp. 218–231. Springer (2002)
12. Alagiannis, I., Idreos, S., Ailamaki, A.: H2o: a hands-free adaptive store. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 1103–1114 (2014)
13. Alomari, M., Cahill, M., Fekete, A., Rohm, U.: The cost of serializability on platforms that use snapshot isolation. In: 2008 IEEE 24th International Conference on Data Engineering, pp. 576–585. IEEE (2008)
14. Andoga, R., Schreiner, M., Moravec, T., Főző, L., Schrötter, M.: Automatic decision making process in a small unmanned airplane. In: 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), pp. 000301–000306. IEEE (2018)
15. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1), 1–39 (2008)
16. Appuswamy, R., Karpathiotakis, M., Porobic, D., Ailamaki, A.: The case for heterogeneous htap. In: 8th Biennial Conference on Innovative Data Systems Research, number CONF (2017)
17. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A. et al.: Spark sql: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1383–1394 (2015)
18. Arora, V., Nawab, F., Agrawal, D., El Abbadi, A.: Janus: a hybrid scalable multi-representation cloud datastore. IEEE Trans. Knowl. Data Eng. **30**(4), 689–702 (2017)
19. Arulraj, J., Pavlo, A., Menon, P.: Bridging the archipelago between row-stores and column-stores for hybrid workloads. In: Proceedings of the 2016 International Conference on Management of Data, pp. 583–598 (2016)
20. Bacon, D.F., Bales, N., Bruno, N., Cooper, B.F., Dickinson, A., Fikes, A., Fraser, C., Gubarev, A., Joshi, M., Kogan, E. et al.: Spanner: becoming a sql system. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 331–343 (2017)
21. Barber, R., Garcia-Arellano, C., Grosman, R., Lohman, G., Mohan, C., Muller, R., Pirahesh, H., Raman, V., Sidle, R., Storm, A., et al.: Wiser: a highly available htap dbms for iot applications. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 268–277. IEEE (2019)
22. Barber, R., Huras, M., Lohman, G., Mohan, C., Mueller, R., Özcan, F., Pirahesh, H., Raman, V., Sidle, R., Sidorkin, O., et al.: Wildfire: concurrent blazing data ingest and analytics. In: Proceedings of the 2016 International Conference on Management of Data, pp. 2077–2080 (2016)
23. Bender, M.A., Farach-Colton, M., Jannen, W., Johnson, R., Kuszmaul, B.C., Porter, D.E., Yuan, J., Zhan, Y.: An introduction to b-trees and write-optimization. login; magazine, **40**(5), (2015)
24. Bitincka, L., Ganapathi, A., Sorkin, S., Zhang, S.: Optimizing data analysis with a semi-structured time series database. In: Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML 10) (2010)
25. Bog, A., Kruger, J., Schaffner, J.: A composite benchmark for online transaction processing and operational reporting. In: 2008 IEEE Symposium on Advanced Management of Information for Globalized Enterprises (AMIGE), pp. 1–5. IEEE (2008)
26. Bog, A., Sachs, K., Zeier, A., Plattner, H.: Normalization in a mixed oltp and olap workload scenario. In: Technology Conference on Performance Evaluation and Benchmarking, pp. 67–82. Springer (2011)
27. Boncz, P.A., Zukowski, M.: Vectorwise: beyond column stores. IEEE Data Eng. Bull. **35**(1), 21–27 (2012)
28. Boncz, P.A., Manegold, S., Kersten, M.L., et al.: Database architecture optimized for the new bottleneck: memory access. VLDB **99**, 54–65 (1999)
29. Boncz, P.A., Zukowski, M., Nes, N.: Monetdb/x100: hyper-pipelining query execution. Cidr **5**, 225–237 (2005)
30. Boroumand, A., Ghose, S., Oliveira, G.F., Mutlu, O.: Enabling high-performance and energy-efficient hybrid transactional/analytical databases with hardware/software cooperation. arXiv preprint arXiv:2204.11275 (2022)
31. Böttcher, J., Leis, V., Neumann, T., Kemper, A.: Scalable garbage collection for in-memory mvcc systems. Proc. VLDB Endow. **13**(2), 128–141 (2019)
32. Buchmann, A.P., McCarthy, D.R., Hsu, M., Dayal, U.: Time-critical database scheduling: a framework for integrating real-time scheduling and concurrency control. In: Proceedings. Fifth Inter-

national Conference on Data Engineering, pp. 470–471. IEEE Computer Society (1989)
33. Buragohain, C., Risvik, K.M., Brett, P., Castro, M., Cho, W., Cowhig, J., Gloy, N., Kalyanaraman, K., Khanna, R., Pao, J. et al.: A1: A distributed in-memory graph database. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 329–344 (2020)
34. Butterstein, D., Martin, D., Stolze, K., Beier, F., Zhong, J., Wang, L.: Replication at the speed of change: a fast, scalable replication solution for near real-time htap processing. Proc. VLDB Endow. **13**(12), 3245–3257 (2020)
35. Camilleri, C., Vella, J.G., Nezval, V.: Htap with reactive streaming ETL. J. Cases Inf. Technol. **23**(4), 1–19 (2021)
36. Cao, S., Yang, X., Chen, C., Zhou, J., Li, X., Qi, Y.: Titant: online real-time transaction fraud detection in ant financial. arXiv preprint arXiv:1906.07407, (2019)
37. Cao, T., Vaz Salles, M., Sowell, B., Yue, Y., Demers, A., Gehrke, J., White, W.: Fast checkpoint recovery algorithms for frequently consistent applications. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 265–276 (2011)
38. Chen, J., Jindel, S., Walzer, R., Sen, R., Jimsheleishvilli, N., Andrews, M.: The memsql query optimizer: a modern optimizer for real-time analytics in a distributed database. Proc. VLDB Endow. **9**(13), 1401–1412 (2016)
39. Chen, J., Ding, Y., Liu, Y., Li, F., Zhang, L., Zhang, M., Wei, K., Cao, L., Zou, D., Liu, Y., et al.: Bytehtap: bytedance's htap system with high data freshness and strong data consistency. Proc. VLDB Endow. **15**(12), 3411–3424 (2022)
40. Chen, M.K., Sheldon, M.: Dynamic pricing in a labor market: surge pricing and flexible work on the uber platform. *Ec*, 16, 455 (2016)
41. Chen, X., Song, H., Jiang, J., Ruan, C., Li, C., Wang, S., Zhang, G., Cheng, R., Cui, H.: Achieving low tail-latency and high scalability for serializable transactions in edge computing. In: Proceedings of the Sixteenth European Conference on Computer Systems, pp. 210–227 (2021)
42. Chisholm, S.: Adopting medical technologies and diagnostics recommended by nice: the health technologies adoption programme (2014)
43. Choudhury, S., Ghosh, S., Bhattacharya, A., Fernandes, K.J., Tiwari, M.K.: A real time clustering and svm based price-volatility prediction for optimal trading strategy. Neurocomputing **131**, 419–426 (2014)
44. Cipar, J.: Trading latency for freshness in storage systems (2012)
45. Inc. ClickHouse. ClickHouse—open source distributed column-oriented DBMS. https://github.com/ClickHouse/ClickHouse/tree/22.6
46. ALIBABA CLOUD. Double 11 Real-Time Monitoring System with Time Series Database. https://www.alibabacloud.com/blog/594855
47. Coelho, F., Paulo, J., Vilaça, R., Pereira, J., Oliveira, R.: Htapbench: hybrid transactional and analytical processing benchmark. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, pp. 293–304 (2017)
48. Cole, R., Funke, F., Giakoumakis, L., Guy, W., Kemper, A., Krompass, S., Kuno, H., Nambiar, R., Neumann, T., Poess, M., et al.: The mixed workload ch-benchmark. In: Proceedings of the Fourth International Workshop on Testing Database Systems, pp. 1–6 (2011)
49. Comer, D.: Ubiquitous b-tree. ACM Comput. Surv. **11**(2), 121–137 (1979)
50. Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., et al.: Spanner: Google's globally distributed database. ACM Trans. Comput. Syst. **31**(3), 1–22 (2013)
51. TATP Benchmark Council. TATP Benchmark Description (Version 1.0). http://tatpbenchmark.sourceforge.net
52. THE TRANSACTION PROCESSING COUNCIL. TPC-H. http://www.tpc.org/tpch/
53. THE TRANSACTION PROCESSING COUNCIL. TPC-C. http://www.tpc.org/tpcc/, 2014
54. Cubukcu, U., Erdogan, O., Pathak, S., Sannakkayala, S., Slot, M.: Citus: distributed postgresql for data-intensive applications. In: Proceedings of the 2021 International Conference on Management of Data, pp. 2490–2502 (2021)
55. Dageville, B., Cruanes, T., Zukowski, M., Antonov, V., Avanes, A., Bock, J., Claybaugh, J., Engovatov, D., Hentschel, M., Huang, J, et al.: The snowflake elastic data warehouse. In: Proceedings of the 2016 International Conference on Management of Data, pp. 215–226 (2016)
56. Science Direct. Real-Time Pricing. https://www.sciencedirect.com/topics/engineering/real-time-pricing
57. Dittrich, J., Jindal, A.: Towards a one size fits all database architecture. In: CIDR, pp. 195–198 (2011)
58. Dziedzic, A., Wang, J., Das, S., Ding, B., Narasayya, V.R., Syamala, M.: Columnstore and b+ tree-are hybrid physical designs important? In: Proceedings of the 2018 International Conference on Management of Data, pp. 177–190 (2018)
59. Dziedzic, A., Wang, J., Das, S., Ding, B., Narasayya, V.R., Syamala, M.: Columnstore and b+ tree-are hybrid physical designs important? In: Proceedings of the 2018 International Conference on Management of Data, pp. 177–190 (2018)
60. Fu, Y., Soman, C.: Real-time data infrastructure at uber. In: Proceedings of the 2021 International Conference on Management of Data, pp. 2503–2516 (2021)
61. Funke, F., Kemper, A., Neumann, T.: Benchmarking hybrid oltp&olap database systems. Datenbanksysteme für Business, Technologie und Web (BTW) (2011)
62. Funke, F., Kemper, A., Neumann, T.: Compacting transactional data in hybrid oltp & olap databases. arXiv preprint arXiv:1208.0224 (2012)
63. Gartner. Setting the Record Straight—HTAP & OPDBMS. https://blogs.gartner.com/donald-feinberg/2018/01/11/setting-record-straight-htap/
64. Goldberg, R.P.: Survey of virtual machine research. Computer **7**(6), 34–45 (1974)
65. Google. Alloydb for postgresql under the hood: columnar engine. https://cloud.google.com/blog/products/databases/alloydb-for-postgresql-columnar-engine, (2022)
66. Graefe, G.: Volcano: an extensible and parallel query evaluation system. IEEE Trans. Knowl. Data Eng. **6**(1), 120–135 (1994)
67. Graefe, G., et al.: Modern b-tree techniques. Found. Trends Databases **3**(4), 203–402 (2011)
68. Gray, S., Özcan, F., Pereyra, H., van der Linden, B., Zubiri, A.: Ibm big sql 3.0: Sql-on-hadoop without compromise (2014)
69. Ant Group. OceanBase. https://www.oceanbase.com/en
70. Docker Group. What is a Container? https://www.docker.com/resources/what-container/
71. Linux Group. Control Groups. https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html
72. Guerraoui, R., Schiper, A.: Fault-tolerance by replication in distributed systems. In: International Conference on Reliable Software Technologies, pp. 38–57. Springer (1996)
73. Guerraoui, R., Schiper, A.: Software-based replication for fault tolerance. Computer **30**(4), 68–74 (1997)
74. Gupta, A., Agarwal, D., Tan, D., Kulesza, J., Pathak, R., Stefani, S., Srinivasan, V.: Amazon redshift and the case for simpler data warehouses. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1917–1923 (2015)
75. Gupta, M.K., Chandra, P.: A comprehensive survey of data mining. Int. J. Inf. Technol. **12**(4), 1243–1257 (2020)

76. Güting, R.H.: An introduction to spatial database systems. VLDB J. **3**(4), 357–399 (1994)
77. Hsiao, H.-I., DeWitt, D.J.: A performance study of three high availability data replication strategies. Distrib. Parallel Databases **1**(1), 53–79 (1993)
78. Huang, D., Liu, Q., Cui, Q., Fang, Z., Ma, X., Fei, X., Shen, L., Tang, L., Zhou, Y., Huang, M., et al.: Tidb: a raft-based htap database. Proc. VLDB Endow. **13**(12), 3072–3084 (2020)
79. Oracle Inc. Heap-organized table. https://www.orafaq.com/wiki/Heap-organized_table
80. SnowFlake Inc. Unistore: A modern approach to working with transactional and analytical data together in a single platform. https://www.snowflake.com/workloads/unistore/
81. Jin, G., Bian, H., Chen, Y., Du, X.: Columnar storage optimization and caching for data lakes. In: EDBT, pp. 2–419 (2022)
82. Kang, D., Jiang, R., Blanas, S.: Jigsaw: a data storage and query processing engine for irregular table partitioning. In: Proceedings of the 2021 International Conference on Management of Data, pp. 898–911 (2021)
83. Kang, G., Wang, L., Gao, W., Tang, F., Zhan, J.: Olxpbench: real-time, semantically consistent, and domain-specific are essential in benchmarking, designing, and implementing htap systems. arXiv preprint arXiv:2203.16095 (2022)
84. Kemme, B., Alonso, G.: Don't be lazy, be consistent: postgres-r, a new way to implement database replication. In: VLDB, pp. 134–143. Citeseer (2000)
85. Kemper, A., Neumann, T.: Hyper: a hybrid oltp&olap main memory database system based on virtual memory snapshots. In: 2011 IEEE 27th International Conference on Data Engineering, pp. 195–206. IEEE (2011)
86. Kemper, A., Neumann, T., Funke, F., Leis, V., Mühe, H.: Hyper: Adapting columnar main-memory data management for transactional and query processing. IEEE Data Eng. Bull. **35**(1), 46–51 (2012)
87. Kester, M.S., Athanassoulis, M., Idreos, S.: Access path selection in main-memory optimized data systems: Should i scan or should i probe? In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 715–730 (2017)
88. Kim, J., Ahn, J., Lee, K., Ryu, M., Jung, H.: Hybrid transactional/analytical processing amplifies io in lsm-trees. IEEE Access (2022)
89. Kim, J., Kim, K., Cho, H., Yu, J., Kang, S., Jung, H.: Rethink the scan in mvcc databases. In: SIGMOD '21, pp. 938–950, New York, NY, USA, 2021. Association for Computing Machinery
90. Kim, J., Yu, J., Ahn, J., Kang, S., Jung, H.: Diva: Making mvcc systems htap-friendly. In: Proceedings of the 2022 International Conference on Management of Data, pp. 49–64 (2022)
91. Konana, P., Ram, S.: Semantics-based transaction processing for real-time databases: the case of automated stock trading. INFORMS J. Comput. **11**(3), 299–315 (1999)
92. Lahiri, T., Chavan, S., Colgan, M., Das, D., Ganesh, A., Gleeson, M., Hase, S., Holloway, A., Kamp, J., Lee, T.-H., et al. Oracle database in-memory: a dual format in-memory database. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 1253–1258. IEEE (2015)
93. Larson, P., Birka, A., Hanson, E.N., Huang, W., Nowakiewicz, M., Papadimos, V.: Real-time analytical processing with sql server. Proc. VLDB Endow. **8**(12), 1740–1751 (2015)
94. Larson, P., Clinciu, C., Hanson, E.N., Oks, A., Price, S.L., Rangarajan, S., Surna, A., Zhou, Q.: Sql server column store indexes. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 1177–1184 (2011)
95. LaValle, S., Lesser, E., Shockley, R., Hopkins, M.S., Kruschwitz, N.: Big data, analytics and the path from insights to value. MIT Sloan Manag. Rev. **52**(2), 21–32 (2011)
96. Lee, J., Kim, K.H., Lee, H., Andrei, M., Ko, S., Keller, F., Han, W.-S.: Asymmetric-partition replication for highly scalable distributed transaction processing in practice. Proc. VLDB Endow. **13**(12), 3112–3124 (2020)
97. Lee, J., Moon, S.H., Kim, K.H., Kim, D.H., Cha, S.K., Han, W.-S.: Parallel replication across formats in sap hana for scaling out mixed oltp/olap workloads. Proc. VLDB Endow. **10**(12), 1598–1609 (2017)
98. Lee, R., Zhou, M., Li, C., Shenggang, H., Teng, J., Li, D., Zhang, X.: The art of balance: a rateupdb^{TM} experience of building a cpu/gpu hybrid database product. Proc. VLDB Endow. **14**(12), 2999–3013 (2021)
99. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? Proc. VLDB Endow. **9**(3), 204–215 (2015)
100. Lepers, B., Balmau, O., Gupta, K., Zwaenepoel, W.: Kvell+: snapshot isolation without snapshots. OSDI'20, USA, 2020. USENIX Association
101. Li, F.: Cloud-native database systems at alibaba: opportunities and challenges. Proc. VLDB Endow. **12**(12), 2263–2272 (2019)
102. Li, F., Özsu, M.T., Chen, G., Ooi, B.C.: R-store: a scalable distributed system for supporting real-time analytics. In: 2014 IEEE 30th International Conference on Data Engineering, pp. 40–51. IEEE (2014)
103. Li, G., Zhang, C.: Htap databases: what is new and what is next. In: Proceedings of the 2022 International Conference on Management of Data, pp. 2483–2488 (2022)
104. Li, L., Wang, G., Wu, G., Yuan, Y.: Consistent snapshot algorithms for in-memory database systems: experiments and analysis. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 1284–1287. IEEE (2018)
105. Li, T., Butrovich, M., Ngom, A., Lim, W.S., McKinney, W., Pavlo, A.: Mainlining databases: Supporting fast transactional workloads on universal columnar data file formats. arXiv preprint arXiv:2004.14471 (2020)
106. Liedes, A.-P., Wolski, A.: Siren: a memory-conserving, snapshot-consistent checkpoint algorithm for in-memory databases. In: 22nd International Conference on Data Engineering (ICDE'06), pp. 99–99. IEEE (2006)
107. Lima, A.A.B., Furtado, C., Valduriez, P., Mattoso, M.: Parallel olap query processing in database clusters with data replication. Distrib. Parall. Databases **25**(1), 97–123 (2009)
108. Lu, Y., Yu, X., Cao, L., Madden, S.: Epoch-based commit and replication in distributed oltp databases (2021)
109. Luo, C., Tözün, P., Tian, Y., Barber, R., Raman, V., Sidle, R.: Umzi: unified multi-zone indexing for large-scale htap. In: EDBT, pp. 1–12 (2019)
110. Lyu, Z., Zhang, H.H., Xiong, G., Guo, G., Wang, H., Chen, J., Asim Praveen, J., Yang, Y., Gao, X., Wang, A., et al.: Greenplum: a hybrid database for transactional and analytical workloads. In: Proceedings of the 2021 International Conference on Management of Data, pp. 2530–2542 (2021)
111. Makreshanski, D., Giceva, J., Barthels, C., Alonso, G.: Batchdb: efficient isolated execution of hybrid oltp+ olap workloads for interactive applications. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 37–50 (2017)
112. Mao, Y., Kohler, E., Morris, R.T.: Cache craftiness for fast multicore key-value storage. In: Proceedings of the 7th ACM European Conference on Computer Systems, pp. 183–196 (2012)
113. Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., Kraska, T.: Bao: making learned query optimization practical. In: Proceedings of the 2021 International Conference on Management of Data, pp. 1275–1288 (2021)
114. Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., Kraska, T.: Bao: making learned query optimization practical. ACM SIGMOD Rec. **51**(1), 6–13 (2022)

115. Marcus, R., Negi, P., Mao, H., Zhang, C., Alizadeh, M., Kraska, T., Papaemmanouil, O., Tatbul, N.: Neo: a learned query optimizer. arXiv preprint arXiv:1904.03711 (2019)
116. Meng, Q., Zhou, X., Chen, S., Wang, S.: Swingdb: an embedded in-memory dbms enabling instant snapshot sharing. In: Data Management on New Hardware, pp. 134–149. Springer (2016)
117. Milkai, E., Chronis, Y., Gaffney, K.P., Guo, Z., Patel, J.M., Yu, X.: How good is my htap system? In: Proceedings of the 2022 International Conference on Management of Data, pp. 1810–1824 (2022)
118. Mishra, S., Tripathi, A.R.: Ai business model: an integrative business approach. J. Innov. Entrep. **10**(1), 1–21 (2021)
119. Mohan, C., Lindsay, B., Obermarck, R.: Transaction management in the r* distributed database management system. ACM Trans. Database Syst. **11**(4), 378–396 (1986)
120. Moiz, S.A., Sailaja, P., Venkataswamy, G., Pal, S.N.: Database replication: a survey of open source and commercial tools. Int. J. Comput. Appl. **13**(6), 1–8 (2011)
121. Mühlbauer, T., Rödiger, W., Reiser, A., Kemper, A., Neumann, T.: Scyper: elastic olap throughput on transactional data. In: Proceedings of the Second Workshop on Data Analytics in the Cloud, pp. 11–15 (2013)
122. MySQL. MySQL 8.0.18 (2019-10-14, General Availability). https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-18.html
123. MySQL. Mysql heatwave. https://dev.mysql.com/doc/heatwave/en/heatwave-introduction.html, (2022)
124. Neumann, T., Mühlbauer, T., Kemper, A.: Fast serializable multiversion concurrency control for main-memory database systems. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 677–689 (2015)
125. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference (Usenix ATC 14), pp. 305–319 (2014)
126. Özcan, F., Tian, Y., Tözün, P.: Hybrid transactional/analytical processing: a survey. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1771–1775 (2017)
127. O'Neil, P., Cheng, E., Gawlick, D., O'Neil, E.: The log-structured merge-tree (lsm-tree). Acta Informatica **33**(4), 351–385 (1996)
128. O'Neil, P., Chen, X., Betty, O.: Star Schema Benchmark. https://www.cs.umb.edu/~poneil/StarSchemaB.PDF
129. Pelster, M.: I'll have what s/he's having: a case study of a social trading network. ICIS 2017 Proceedings, (2017)
130. Pezzini, M., Feinberg, D., Rayner, N., Edjlali, R.: Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation. Gartner (2014, January 28) Available at https://www.gartner.com/doc/2657815/hybrid-transactionanalyticalprocessing-foster-opportunities, pp. 4–20 (2014)
131. Prout, A., Wang, S.-P., Victor, J., Sun, Z., Li, Y., Chen, J., Bergeron, E., Hanson, E., Walzer, R., Gomes, R., et al: Cloud-native transactions and analytics in singlestore. In: Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data (2022)
132. Psaroudakis, I., Wolf, F., May, N., Neumann, T., Böhm, A., Ailamaki, A., Sattler, K.-U.: Scaling up mixed workloads: a battle of data freshness, flexibility, and scheduling. In: Technology Conference on Performance Evaluation and Benchmarking, pp. 97–112. Springer (2014)
133. Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. 33(6):668-676 (1990)
134. Qiu, T., Chi, J., Zhou, X., Ning, Z., Atiquzzaman, M., Dapeng Oliver, W.: Edge computing in industrial internet of things: architecture, advances and challenges. IEEE Commun. Surv. Tutor. **22**(4), 2462–2488 (2020)
135. Qiu, X., Cen, W., Qian, Z., Peng, Y., Zhang, Y., Lin, X., Zhou, J.: Real-time constrained cycle detection in large dynamic graphs. Proc. VLDB Endow. **11**(12), 1876–1888 (2018)
136. Quah, J.T.S., Sriganesh, M.: Real-time credit card fraud detection using computational intelligence. Expert Syst. Appl. **35**(4), 1721–1732 (2008)
137. Raman, V., Attaluri, G., Barber, R., Chainani, N., Kalmuk, D., KulandaiSamy, V., Leenstra, J., Lightstone, S., Liu, S., Lohman, G.M., et al.: Db2 with blu acceleration: so much more than just a column store. Proc. VLDB Endow. **6**(11), 1080–1091 (2013)
138. Raza, A., Chrysogelos, P., Anadiotis, A.C., Ailamaki, A.: Adaptive htap through elastic resource scheduling. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 2043–2054 (2020)
139. Raza, S.M.A., Chrysogelos, P., Sioulas, P., Indjic, V., Anadiotis, A.C., Ailamaki, A.: Gpu-accelerated data management under the test of time. In: Online Proceedings of the 10th Conference on Innovative Data Systems Research (CIDR), number CONF, (2020)
140. Rhea, S., Wang, E., Wong, E., Atkins, E., Storer, N.: Littletable: a time-series database and its uses. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 125–138 (2017)
141. Robinson, H.A., Cherry, C.: Results of a prototype television bandwidth compression scheme. Proc. IEEE **55**(3), 356–364 (1967)
142. Sadoghi, M., Bhattacherjee, S., Bhattacharjee, B., Canim, M.: L-store: a real-time oltp and olap system. arXiv preprint arXiv:1601.04084 (2016)
143. Sahay, B.S., Ranjan, J.: Real time business intelligence in supply chain analytics. Inf. Manag. Comput. Secur., x(Mustafa)
144. Schiefer, J., Bruckner, R.: Container-managed etl applications for integrating data in near real-time (2003)
145. Computer science. Column-oriented DBMS. https://en.wikipedia.org/wiki/Column-oriented_DBMS
146. Computer science. Starvation (computer science). https://en.wikipedia.org/wiki/Starvation_(computer_science)
147. Computer science. State machine replication. https://en.wikipedia.org/wiki/State_machine_replication
148. Sewall, J., Chhugani, J., Kim, C., Satish, N., Dubey, P.: Palm: parallel architecture-friendly latch-free modifications to b+ trees on many-core processors. Proc. VLDB Endow. **4**(11), 795–806 (2011)
149. Sharma, A., Schuhknecht, F.M., Dittrich, J.: Accelerating analytical processing in mvcc using fine-granular high-frequency virtual snapshotting. In: Proceedings of the 2018 International Conference on Management of Data, pp. 245–258 (2018)
150. Shen, S., Chen, R., Chen, H., Zang, B.: Retrofitting high availability mechanism to tame hybrid transaction/analytical processing. In: 15th $\{USENIX\}$ Symposium on Operating Systems Design and Implementation ($\{OSDI\}$ 21), pp. 219–238 (2021)
151. Shen, S., Yao, Z., Shi, L., Wang, L., Lai, L., Tao, Q., Su, L., Chen, R., Yu, W., Chen, H., et al.: Bridging the gap between relational $\{OLTP\}$ and graph-based $\{OLAP\}$. In: 2023 USENIX Annual Technical Conference (USENIX ATC 23), pp. 181–196 (2023)
152. Sikka, V., Färber, F., Lehner, W., Cha, S.K., Peh, T., Bornhövd, C.: Efficient transaction processing in sap hana database: the end of a column store myth. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 731–742 (2012)
153. Inc. SingleStore. SingleStore: Real-Time Distributed SQL. https://www.singlestore.com/
154. Song, H., Zhou, W., Li, F., Peng, X., Cui, H.: Rethink query optimization in htap databases. Proc. ACM Manag. Data **1**(4), 1–27 (2023)

155. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., et al.: C-store: a column-oriented dbms. In: Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker, pp. 491–518 (2018)
156. Sukarsa, I.M., Wisswani, N.W., Darma, I.K.G.: Change data capture on oltp staging area for nearly real time data warehouse base on database trigger. Int. J. Comput. Appl. **52**(11), (2012)
157. Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., Shah, S.: Serving large-scale batch computed data with project voldemort. In FAST **12**, 18–18 (2012)
158. Sun, Y., Blelloch, G.E., Lim, W.S., Pavlo, A.: On supporting efficient snapshot isolation for hybrid workloads with multi-versioned indexes. Proc. VLDB Endow. **13**(2), (2019)
159. Ta, V.-D., Liu, C.-M., Nkabinde, G.W.: Big data stream computing in healthcare real-time analytics. In: 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 37–42. IEEE (2016)
160. Tai, A., Wei, M., Freedman, M.J., Abraham, I., Malkhi, D.: Replex: a scalable, highly available $\{Multi - Index\}$ data store. In: 2016 USENIX Annual Technical Conference (USENIX ATC 16), pp. 337–350 (2016)
161. Tang, L., Meng, Y.: Data analytics and optimization for smart industry. Front. Eng. Manag. **8**(2), 157–171 (2021)
162. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., Murthy, R.: Hive-a petabyte scale data warehouse using hadoop. In: 2010 IEEE 26th international conference on data engineering (ICDE 2010), pp. 996–1005. IEEE (2010)
163. Tunny. Tunny is a Golang library for spawning and managing a goroutine pool. https://github.com/Jeffail/tunny
164. Vaisman, A., Zimányi, E.: Data warehouse systems. Data-Cent. Syst, Appl (2014)
165. Vamvoudakis, K.G, Lewis, F.L, Mellouk, A.: Online gaming: real time solution of nonlinear two-player zero-sum games using synchronous policy iteration. In: Advances in Reinforcement Learning. Intech, (2011)
166. Vassiliadis, P.: A survey of extract-transform-load technology. Int. J. Data Warehous. Min. **5**(3), 1–27 (2009)
167. Vinçon, T., Knödler, C., Solis-Vasquez, L., Bernhardt, A., Tamimi, S., Weber, L., Stock, F., Koch, A., Petrov, I.: Near-data processing in database systems on native computational storage under htap workloads. Proc. VLDB Endow. **15**(10), 1991–2004 (2022)
168. Wang, J., Li, T., Song, H., Yang, X., Zhou, W., Li, F., Yan, B., Qianqian, W., Liang, Y., Ying, C.J., et al.: Polardb-imci: a cloud-native htap database system at alibaba. Proc. ACM Manag. Data **1**(2), 1–25 (2023)
169. Wang, X., Zhang, W., Wang, Z., Wei, Z., Chen, H., Zhao, W.: Eunomia: scaling concurrent search trees under contention using htm. ACM SIGPLAN Notices **52**(8), 385–399 (2017)
170. Wang, Z., Ma, T., Kong, L., Wen, Z., Li, J., Song, Z., Lu, Y., Chen, G., Cao, W.: Zero overhead monitoring for cloud-native infrastructure using {RDMA}. In: 2022 USENIX Annual Technical Conference (USENIX ATC 22), pp. 639–654 (2022)
171. Winston, P.H., Prendergast, K.A.: The AI business: commercial uses of artificial intelligence. Massachusetts Institute of Technology (1984)
172. Yingjun, W., Arulraj, J., Lin, J., Xian, R., Pavlo, A.: An empirical evaluation of in-memory multi-version concurrency control. Proc. VLDB Endow. **10**(7), 781–792 (2017)
173. Xie, D., Li, F., Yao, B., Li, G., Zhou, L., Guo, M.: Simba: efficient in-memory spatial analytics. In: Proceedings of the 2016 International Conference on Management of Data, pp. 1071–1085, (2016)
174. Hansong, X., Wei, Y., Griffith, D., Golmie, N.: A survey on industrial internet of things: a cyber-physical systems perspective. IEEE Access **6**, 78238–78259 (2018)
175. Yang, J., Rae, I., Jun, X., Shute, J., Yuan, Z., Lau, K., Zeng, Q., Zhao, X., Ma, J., Chen, Z., et al.: F1 lightning: Htap as a service. Proc. VLDB Endow. **13**(12), 3313–3325 (2020)
176. Yang, M., Zheng, Z., Mookerjee, V.: How much is financial advice worth? The transparency-revenue tension in social trading. Manag, Sci (2021)
177. Xiang, Y., Chai, C., Li, G., Liu, J.: Cost-based or learning-based? A hybrid query optimizer for query plan selection. Proc. VLDB Endow. **15**(13), 3924–3936 (2022)
178. Zhang, J., Wu, S., Tan, Z., Chen, G., Cheng, Z., Cao, W., Gao, Y., Feng, X.: S3: a scalable in-memory skip-list index for key-value store. Proc. VLDB Endow. **12**(12), 2183–2194 (2019)
179. Zhang, Z.: Spark-on-hbase: Dataframe based hbase connector
180. Zukowski, M., Van de Wiel, M., Boncz, P.: Vectorwise: a vectorized analytical dbms. In: 2012 IEEE 28th International Conference on Data Engineering, pp. 1349–1350. IEEE (2012)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.