veDB-HTAP: a Highly Integrated, Efficient and Adaptive HTAP System

Jianjun Chen, Li Zhang, Yu Xie, Wei Ding, Lixun Cao, Ye Liu, Yonghua Ding, Fangshi Li, Ke Wu, Haibo Xiu*, Kui Wei, Le Cai, Rui Chang, Yuxiang Chen, Yuanjin Lin, Shangyu Luo, Jianfeng Qian, Xu Wang, Zikang Wang, Jian Zhang, Mingyi Zhang, Shicai Zeng, Jason Sun, Lei Zhang, Rui Shi, Pengwei Zhao
ByteDance Inc *Duke University

ABSTRACT

In this paper, we describe veDB-HTAP, a highly integrated, efficient, and adaptive HTAP system recently built in ByteDance. veDB-HTAP adopts a highly integrated system architecture by leveraging the Secondary Engine mechanism provided by MySQL and provides a seamless query processing experience across OLTP and OLAP engines. In addition, we introduce a cost-based and machine-learning-based smart query router that significantly outperforms the rule-based query router used in ByteHTAP, a precursor of veDB-HTAP. A key design principle of veDB-HTAP is the collaboration and adaptability of major system components, including query planning, query execution, and unified storage. Our adaptive query execution can be classified into two categories: 1) adaptive execution that dynamically collects and utilizes runtime statistics for better query performance; 2) utilizing runtime resource information to achieve a high quality of service even under heavy workloads. The experiments show that veDB-HTAP can achieve more than 3× speedup for TPC-H while consuming only one-third of the resources compared to ByteHTAP.

PVLDB Reference Format:
Jianjun Chen, Li Zhang, Yu Xie, Wei Ding, Lixun Cao, Ye Liu, Yonghua Ding, Fangshi Li, Ke Wu, Haibo Xiu, Kui Wei, Le Cai, Rui Chang, Yuxiang Chen, Yuanjin Lin, Shangyu Luo, Jianfeng Qian, Xu Wang, Zikang Wang, Jian Zhang, Mingyi Zhang, Shicai Zeng, Jason Sun, Lei Zhang, Rui Shi, Pengwei Zhao. veDB-HTAP: a Highly Integrated, Efficient and Adaptive HTAP System. PVLDB, 18(12): 4896 - 4909, 2025.
doi:10.14778/3750601.3750614

1 INTRODUCTION

As ByteDance's business grows, we have observed many applications that require both transactional processing and analytical processing over fresh data with strong data consistency requirements. To meet those business requirements, we built ByteHTAP [32] system, which was launched in the middle of 2021.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750614

Dr. Jianjun Chen is the corresponding author, jianjun.chen@bytedance.com.

To provide context for this paper, we begin with a brief overview of ByteHTAP system. ByteHTAP employs an architecture of *separate engines* over *unified storage*, integrating a ByteDance's existing OLTP (TP) system (i.e., veDB [18, 76]) and an open-source OLAP (AP) system (i.e., Apache Flink [6]). ByteHTAP provides a unified API for query processing, facilitating seamless interaction with both OLTP and OLAP systems. A rule-based query router within the proxy layer automatically routes queries to the appropriate engine—either OLTP or OLAP—based on predefined rules. The Metadata Service provides centralized metadata access for the OLAP query optimizer and storage nodes. ByteHTAP guarantees strong data consistency by providing consistent data snapshots for its queries. The Metadata Service relays a globally committed Log Sequence Number (LSN) to the OLAP engine. Each query is assigned a read LSN derived from the globally committed LSN, ensuring that the query operates on a consistent data snapshot.

We aim to provide HTAP capability on ByteDance's public cloud (i.e., Volcano Engine [9]) as an extension to our veDB product [18], besides supporting internal customers. Many external use cases are expected to involve upgrades from our existing veDB MySQL customers, who wish to leverage enhanced OLAP query-processing capabilities without modifying their applications, which requires ByteHTAP to provide consistent query semantics across its OLTP and OLAP engines. In addition, many ByteHTAP customers want a more efficient and cost-effective OLAP query engine. Below we listed three substantial limitations of ByteHTAP that motivated the design of *veDB-HTAP*:

Subtle semantic differences between OLTP and OLAP engines. First, even though ByteHTAP's OLTP engine uses MySQL read committed (RC) isolation level as default, its OLAP engine only supports snapshot isolation which often uses a slightly older snapshot than the most recently committed one in the OLTP engine due to log replication delay. Hence, the same query executed by both the OLTP and OLAP engines may produce different results. Therefore, ByteHTAP does not support session-level data consistency. Queries directed to the OLAP engine may not reflect earlier updates made by DML operations within the same session. Second, ByteHTAP adopts Apache Flink as its OLAP query engine, which uses its own SQL dialects and lacks certain MySQL dialects support. The system's unified storage also needs enhancement to support the MySQL dialects.

Simple rule-based query routing in the proxy has many limitations. The rule-based query router in ByteHTAP's proxy layer determines query routing based solely on predefined rules, without considering the performance characteristics or syntax support of OLTP and OLAP engines. This can lead to query failures if a query is sent to an engine that does not support its syntax. To address this, we propose a more sophisticated query router that leverages a cost-based query optimizer for in-depth plan analysis while maintaining MySQL compatibility. Additionally, we are exploring machine learning (ML)-based approaches to further enhance routing accuracy.

Limitations in Flink's query engine and resource scheduler to support large-scale OLAP workloads. Although ByteHTAP has made significant improvements over Flink and achieved up to 25% performance improvements on the TPC-H benchmark compared to Flink's open source version [32], it did not satisfy the requirements of our new use cases, which demanded a more efficient and elastic OLAP system. For example, ByteDance's application-platform-as-a-service team wants to build a scalable platform-as-a-service platform on top of ByteHTAP to support thousands of developers with tens of millions of tables. In addition, they want great OLAP query performance over complex queries, i.e., seconds of query latency, that may consist of many joins and other complex operators. Furthermore, they want our system to be elastic and cost-effective, and it can be used in a multi-tenant environment.

In summary, we aim to build a highly integrated HTAP system that alleviates users from concerns about underlying engine differences by providing consistent querying semantics across its OLTP and OLAP engines. In addition, we seek to improve query routing to ensure each query is directed to the appropriate engine for execution. Furthermore, we strive to develop an efficient and elastic OLAP query engine capable of handling complex OLAP workloads in a multi-tenant environment.

Based on the aforementioned considerations, we propose veDB-HTAP, a system designed to serve as a generic HTAP solution capable of accommodating both large-scale and light-weight HTAP processing needs, including: (1) catering to large-scale customers with multi-tenant demands, and providing support for up to tens of millions of tables while ensuring high performance and scalability, and (2) offering solutions for light-workloads OLTP clients, and prioritizing cost efficiency and excellent MySQL compatibility.

We released veDB-HTAP to internal customers in 2024 and plan to expand its availability to external customers on ByteDance's Volcano Engine soon. Since its release, several customers have been onboarded and we are actively migrating ByteHTAP customers to veDB-HTAP. Customers of veDB-HTAP have reported significant improvement in performance and resource utilization compared to ByteHTAP. veDB-HTAP's support of MySQL Read Committed isolation level also simplifies the migration of our customers' workloads. In summary, our key contributions are as follows:

- veDB-HTAP transforms system architecture of ByteHTAP into a highly integrated system. In veDB-HTAP, queries are always sent to the OLTP engine and we utilize the Secondary Engine mechanism [13] provided by MySQL to direct query to OLAP engine when necessary. veDB-HTAP also enables a seamless querying experience across OLTP and OLAP engines by providing an unified Read Committed isolation level. In addition, a cost-based smart query routing is implemented and used in production. We also designed and implemented a query routing prototype based on machine learning techniques, which showed some interesting and promising results in our preliminary study presented in this paper.
- We demonstrate how to build a high-performance and elastic OLAP query engine in veDB-HTAP to replace the Flink engine in ByteHTAP. A key aspect of our design strategy for achieving high performance and elasticity in our OLAP query system is to ensure that the major system components, including query planning, query execution, and the unified storage, operate in a collaborative and adaptive manner, which will be illustrated with concrete examples within the paper. Overall, our adaptive query execution can be classified into two categories: 1) adaptive execution that dynamically collects and utilizes runtime stats for better query performance; 2) utilizes runtime resource information for better quality of service. The experiments show that veDB-HTAP can achieve more than 3× speedup while consuming 1/3 fewer resources than ByteHTAP.

The rest of the paper is organized as follows: Section 2 gives an overview of related work. Section 3 describes the overall architecture of veDB-HTAP as well as the implementations of its key components. Section 4 shows how veDB-HTAP supports a highly integrated HTAP processing through architecture change and smart query routing. Section 5 focuses on how veDB-HTAP's OLAP query engine achieves high performance and elasticity. In Section 6, we provide some empirical measurements of veDB-HTAP. Finally, Section 7 concludes our work.

2 RELATED WORK

In this section, we first briefly review early HTAP databases and techniques in industry and research literature. Then, we discuss the latest HTAP systems and techniques in detail. Lastly, we present works in the area of resource management and workload isolation.

HTAP systems and techniques have been studied over the years [31, 41, 43, 49, 56, 59, 65, 65, 66, 70, 71]. For example, SAP Hana [39, 48, 55, 75] and WildFire [22–24, 50] are dedicated commercial HTAP systems. In addition, many existing systems are extended to support HTAP functionalities, including Oracle's Database In-Memory Option [46, 60], the BLU Acceleration [68] and IDAA [29] in IBM DB2, Google's F1 Lightning [84], and the HTAP-related components in Microsoft SQL Server 2016 [47] and in Greenplum [51]. Lastly, HTAP techniques are also discussed in some research works [52, 69, 74].

In recent years, HTAP systems have evolved to integrate with new architectures and technologies, leading to the emergence of innovative HTAP databases and prototypes. For instance, TiDB [40] is an open-source distributed HTAP system built on top of TiKV and TiFlash. TiKV serves as a row-based storage engine for transactional workloads, while TiFlash functions as a column-based storage engine for analytical queries. Starting with TiDB v7.0.0, TiFlash introduced support for disaggregated storage and compute [16]. TiDB features its own TiDB Server, which supports MySQL's protocol and common syntax, though not all MySQL features are currently supported [15]. In contrast, veDB-HTAP employs the MySQL Secondary Engine to provide strong MySQL compatibility.

Alibaba PolarDB [78, 85, 86] is a cloud-native relational database. PolarDB-IMCI [78] extends PolarDB with in-memory column indexes and a column-based execution engine to speed up the processing for complex OLAP queries. It executes OLAP queries in its MySQL Read-Only (RO) nodes to separate OLTP workloads from OLAP workloads. AlloyDB [4] features a columnar analytical processing engine that can run on both primary instances and read pool instances. However, similar to PolarDB-IMCI, it is an in-memory single node query engine without distributed query execution capability. In contrast, veDB-HTAP's architecture utilizes MySQL Secondary Engine extension [13] and can plug in any OLAP engine with its modular design. veDB-HTAP has a powerful proprietary MPP OLAP engine that can execute queries across different servers.

PolarDB-SCC [86] adopts several techniques to reduce the data freshness gap between its Read-Write (RW) and RO nodes. Similarly, veDB-HTAP also provides Read Committed isolation and utilizes techniques such as multi-level LSNs to reduce the gap of data freshness between its OLTP and OLAP engines. In contrast, under frequent updates, the columnar data in AlloyDB may become stale relative to the row-format data[4].

There are other commercial and prototype database systems[25, 44] that are enhanced to support HTAP workloads. Amazon Redshift [20] provides some HTAP capabilities by facilitating both the in-place querying of data in the OLTP services by using Redshift's Federated Query, and the seamless copying and synchronization of data to Redshift by using Glue Elastic Views [72]. SingleStoreDB [67], formerly known as MemSQL [33], utilizes the cloud-native architecture and takes advantage of the elastic infrastructure. It supports transactions that perform scans on column stores and perform the seek on row stores, and uses indexes to speed up point reads and writes.

In the evaluation of an HTAP system, HATtrick[57] proposed a method from the perspectives of performance and freshness. A concept called *throughput frontier* is introduced to capture the performance of an HTAP system. HyBench[90] proposed a new benchmark with a hybrid workload that supports throughput and freshness evaluation simultaneously.

Lastly, some works have discussed resource management and workload isolation in multi-tenant database systems [21, 30, 38, 61, 64, 77, 88, 91]. SQLVM [62, 63] provides an abstraction for performance isolation in a multi-tenant RDBMS based on the promise of reservation of resources. It proposes a fine-grained resource scheduling mechanism to meet each tenant's reservations. It also provides a novel metering logic to audit the promise of resources. Another work [36] focuses on the problem of CPU sharing among tenants co-located at a server. It discusses how to use SQLVM [62] as an abstraction of CPU reservations to provide resources to tenants without any restriction on the tenant's workloads. In comparison, veDB-HTAP adopts an adaptive approach to handle resource contention for both CPU and memory.

3 SYSTEM ARCHITECTURE AND IMPORTANT COMPONENTS

In this section, we first give a brief overview of MySQL Secondary Engine. Then, we talk about veDB-HTAP's architecture and important components.

3.1 MySQL Secondary Engine Overview

MySQL has provided built-in support for pluggable storage engines. Customized storage engines such as [79] have been developed and combined with existing storage engines for extended functionality. Coupled with additional query processing capability for workload off-loading or acceleration, this type of alternative storage engine is referred to as Secondary Engine by MySQL Heatwave [13]. MySQL can send different query plans to appropriate storage engines for execution based on the data processing characteristics of different storage engines and the cost, thereby leveraging multiple storage engines for optimal workload performance.

In integrating the Secondary Engine into MySQL, an additional phase is added at the end of the optimizer to decide whether to send the query plan to the primary or secondary engine for execution. Users can specify whether to use a specific storage engine through the Secondary Engine keyword defined in the CREATE TABLE or ALTER TABLE command for existing MySQL tables. As an example, the following command

```
create table orders (
  oid int not null primary key,
  cust_id int not null, o_date date
) secondary_engine = RAPID
```

would create a table with all columns also stored in the secondary engine registered in the system as RAPID, which is the query processing engine used in Heatwave. We can plugin any customized query processing engine such as our own OLAP engine in the same fashion. Once the table has been created with the Secondary Engine support, loading the table will automatically load both MySQL's default storage engine as well as the Secondary Storage. For existing tables that have been altered with Secondary Engine support, a background process will start to replicate the data from the existing MySQL storage to the newly added Secondary Storage.

veDB-HTAP utilizes the same Secondary Engine mechanism as MySQL Heatwave to provide a seamless integration between OLTP and OLAP engines, but they have key differences. MySQL Heatwave provides a highly integrated in-memory OLAP accelerator. Prior to serving user queries, data must be loaded into the memory of all Heatwave compute nodes. Its underlying storage is a simple object store that only handles data durability but does not support predicate pushdown. Propagations of data change also happen at data nodes. Essentially, Heatwave's OLAP MPP query engine adopts a share-nothing architecture where computations happen at in-memory data nodes. In contrast, veDB-HTAP adopts a disaggregated storage architecture. Its MPP query processor is stateless and does not require preloading all data into memory to process queries, making it more elastic than Heatwave. veDB-HTAP's smart unified storage handles data change replications and supports query predicate pushdown. Furthermore, veDB-HTAP has a model-based smart query router besides heuristic and cost-based query routing, while Heatwave only supports heuristic and cost-based routing.

3.2 veDB-HTAP System Overview

As shown in Figure 1, veDB-HTAP inherits ByteHTAP's cloud-native architecture, with the following major changes.

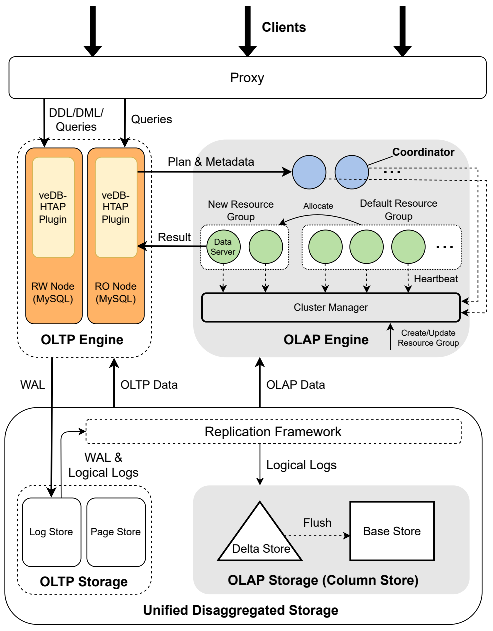


Figure 1: An illustration of the veDB-HTAP architecture.

3.2.1 veDB-HTAP Plugin. As shown in Figure 1, we extended the MySQL OLTP engine using the veDB-HTAP Plugin, our proprietary Secondary Engine extension, on both the Read-Write (RW) and Read-Only (RO) nodes. With this new architecture, veDB-HTAP's proxy always routes user queries to the OLTP engine, which determines the appropriate engine to use based on query complexity and other properties. In contrast, ByteHTAP's proxy decides sending queries to either OLTP or OLAP engines. In veDB-HTAP, DDLs and DMLs can only be sent to the RW node, while read-only queries can be directed to both the RW and RO nodes.

3.2.2 Adopt a proprietary high-performance and elastic MPP OLAP engine. ByteHTAP utilizes Flink as the OLAP engine, which has been widely used in our company for ETL processing in streaming applications. However, it has some limitations in being used in large scale OLAP processing, e.g., resource scheduling/managing for high QPS workload, and lack of vectorized query engine. We have been improving the Flink engine for a while and it can support many existing customers currently.

In veDB-HTAP, we want to significantly extend its OLAP capabilities to further improve query performance in terms of QPS and query latency and reduce the cost of customers through great multi-tenancy support. For example, one of our largest customers manages thousands of shared tenants with a diverse range of job SLAs and priorities. Enabling resource sharing and isolation among a large number of tenants within a single veDB-HTAP cluster is essential for cost reduction. We built a proprietary MPP query processing engine in veDB-HTAP, which consists of a set of *Coordinators* for distributed plan generation and optimization, and a set of *Data Servers* (DS) for distributed plan execution (see Figure 1).

We have done significant work for adaptive query processing to make the OLAP engine highly efficient and elastic, which will be described in detail in Section 5. In addition, we have spent a considerable amount of engineering efforts to make veDB-HTAP MySQL compatible, including data types (especially date, time, and decimals) and functions etc. In the case a query sent to the MPP engine for execution is not supported by the OLAP engine, veDB-HTAP will retry the execution using the OLTP engine automatically.

To support better resource sharing and resource isolation for multi-tenancy, veDB-HTAP introduces a centralized *Cluster Manager* to allow users to create resource groups among Data Servers. For example, customers can create a shared resource group to hold many low-priority tenants with lenient SLA and dedicated resource groups for tenants with stringent SLA. To achieve this, we first send *create resource groups* commands to the cluster manager with the corresponding min/max CPU cores and memories. The cluster manager will then calculate the initial number of Data Servers to be allocated to the dedicated resource groups. When the workload on a resource group changes, Data Servers can be added to or removed from this group by sending *update resource groups* commands to the cluster manager on the fly. Currently this process has not been fully automated yet, which can be a future work to explore.

4 HIGH INTEGRATION BETWEEN OLTP AND OLAP ENGINES

In this section, we describe how veDB-HTAP seamlessly integrates its OLAP engine with its OLTP engine through veDB-HTAP Plugin to provide extendable OLAP query processing capability, MySQL Read Committed transaction isolation level, and smart routing.

4.1 Extensible OLAP Engine Integration Using veDB-HTAP Plugin

As shown in Figure 1, veDB-HTAP Plugin reuses the parser and analyzer from MySQL to generate a MySQL-compatible query plan and performs simple optimizations. It then sends the partially optimized query plan, together with required metadata, to a *coordinator* for distributed plan generation and optimization. After optimization, the colocated query scheduler divides the plan into a set of *fragments* in a distributed fashion and sends them to a set of *Data Servers* for execution, which will read the data from the unified storage and periodically send heartbeats to the cluster manager. The final result blocks will be buffered at a dedicated Data Server and fetched by veDB-HTAP Plugin.

An important design decision is to have the OLAP query optimizer take a partially optimized logical plan from the OLTP planner instead of SQL text. For example, the expression handling, such as data type deducing, is done by the OLTP planner. Some complex OLAP-oriented optimization such as join reorder and common plan subtree sharing, are all handled in the OLAP optimizer, which is specifically designed for such tasks. In addition, we have defined an interface to translate an OLTP logical plan into an OLAP logical plan to allow future extensions to connect to different OLTP engines beyond MySQL. In contrast, Fabric DW [28] adopts a single unified query optimization framework to handle both local and distributed query plan generation. Since veDB-HTAP adopts a seperate engine approah for its TP and AP processing, veDB-HTAP's two-staged plan generation approach can fully take advantage of both engines' plan generation capabilities, provide great compatibility with its OLTP engine (i.e. MySQL), and offer great extensibility for plugging in new query engines.

Another important design decision is that we choose to physically decouple Coordinators (OLAP Query Optimizers) and Data Servers from the MySQL nodes to achieve better isolation between OLTP and OLAP workloads and better scalability and elasticity. As OLAP query optimization tasks can be resource-intensive, running them on separate processes minimizes the impact to OLTP workloads. As the metadata is stored inside MySQL system tables, our OLAP optimizer does not need to store any metadata, which makes it stateless and easily scaled out.

4.2 Support Read-Committed Isolation Level

In ByteHTAP, the OLAP engine supports consistent reads over a snapshot, which might miss the latest committed data in the OLTP engine due to log shipping and processing delays. Thus, ByteHTAP fails to maintain read consistency between the OLAP and OLTP engines. For instance, a query directed to the OLAP engine might not reflect the outcomes of recent DML operations. veDB-HTAP addresses this shortcoming by implementing a Read-Committed (RC) transaction isolation level that spans both OLTP and OLAP engines. This upgrade guarantees that the effects of a write operation will be visible to subsequent read operations, irrespective of whether these operations occur within the same or different sessions.

Such support requires that a query in the OLAP engine must read and only read data committed prior to the query's arrival at the OLTP engine. This necessitates: (1) acquiring aligned read snapshot across OLTP and OLAP engines; (2) guaranteeing the query execution after the OLAP engine receives the corresponding latest data snapshot.

Acquire aligned read snapshot on read-only (RO) nodes. The data is replicated to the OLAP engine by applying logical log records. The veDB-HTAP Plugin assigns a logical log LSN to a query as its read snapshot. However, RO nodes construct OLTP read view based on physical log from RW nodes. When a query is sent to a RO node to execute, RO needs to construct a read snapshot based on a logical log LSN aligned with OLTP read view. To address this issue, veDB-HTAP introduces a new type of physical log record. On a RW node, when a transaction commits, this type of record captures metadata such as the database name, table name, and the commit logical log LSN. RW nodes periodically send physical log records to RO nodes for OLTP data synchronization. The veDB-HTAP Plugin is enhanced to process physical log records to retrieve the logical log LSN, that reflects the latest commit and OLTP snapshot. With this improvement, RO nodes can send queries to the OLAP engine using the most recently committed LSN as the snapshot, ensuring compliance with RC isolation level.

Reduce delay of query execution caused by waiting for the corresponding latest data snapshot. When a query requires the Read Committed (RC) isolation level, the veDB-HTAP Plugin may automatically delay its execution to ensure the required data is ready on the OLAP side. To minimize this delay, two strategies are employed: (1) a carefully designed pipelined waiting mechanism and (2) use of a fine-grained commit-LSN examination. The pipelined waiting mechanism allows queries to progress through the plan optimizer and be dispatched to DS nodes for execution, even if the storage nodes have not yet applied the necessary log records. A secondary check occurs when DS nodes send queries to the storage nodes, ensuring that the required data is ready. This approach overlaps the delay caused by log application with the network latency and query plan optimization time, effectively reducing or even eliminating the waiting period. In the fine-grained commit-LSN examination, the veDB-HTAP Plugin utilizes both instance-level and table-level commit LSNs to determine whether a query can proceed without delay. This method builds upon similar principles to enhance efficiency and responsiveness as [86].

For applications with a stringent query latency requirement, veDB-HTAP provides a compatible mode, called *AP Snapshot mode*, which allows a query to be executed immediately on the OLAP side with its currently available logical LSN.

4.3 Smart Routing

In order to execute a query correctly and efficiently, ByteHTAP used a smart query router to automatically decide which engine to use, mainly based on pre-defined rules over query syntaxes, e.g., queries with multiple joins and aggregates will be sent to the OLAP engine. Apparently, such an approach, while being extremely simple, cannot capture real query costs and cover all potential queries.

In veDB-HTAP, since all queries enter the system via the OLTP engine, we can leverage its query optimizer to get the costs of all incoming queries to make more accurate and cost-based routing decisions. Since OLAP queries typically incur higher costs than OLTP queries, we establish a cost threshold to differentiate between them. If a query plan's cost is lower than the pre-defined threshold, it will be executed by the OLTP engine; otherwise, it will be forwarded to the OLAP engine through veDB-HTAP Plugin. The production results showed that cost-based routing outperformed rule-based routing in the majority of cases. Moreover, the smart router also ensures that queries sent to the OLAP engine accommodate the syntax and LSN requirements.

4.3.1 *Limitations in cost-based routing.*

However, a query's cost may not always align with its actual execution time, which can lead to incorrect routing decisions, i.e., sending a query to an engine that spends more time than the other one. This challenge is further exacerbated by veDB-HTAP's architecture, which relies on two completely different query processing engines. We can use three examples to illustrate the problem, using the schema and data from the TPC-H 100G benchmark.

Example 4.1. A Top-N query with ORDER BY and LIMIT clauses. The cost of the query in the OLTP engine is 1.0×10^3*, with an execution time of* 20*ms. The query's execution time is* 650*ms in the OLAP engine.*

```
SELECT l_orderkey, l_linenumber FROM lineitem
WHERE l_shipdate <= DATE '1993-6-01'
ORDER BY l_orderkey, l_linenumber LIMIT 11500, 1700;
```

Example 4.2. A query joins seven tables, with an index on the column c_phone. Its cost from the OLTP engine is 2.2×10^5*, where its execution time is* 70*ms. In comparison, the OLAP engine completes the same query with an execution time of* 3.54*s.*

```
SELECT SUM(l1.l_extendedprice) AS total_extended_price
FROM customer c, nation n, orders o, lineitem l1, orders o1,
     partsupp p, orders o2
WHERE c.c_phone IN ('15-741-346-9870', '15-741-346-9871',
     '15-741-346-9872', '15-741-346-9873')
AND n_name <> 'egypt' AND l1.l_quantity > 5
AND o.o_custkey = c.c_custkey AND n.n_nationkey = c.c_nationkey
AND l1.l_orderkey = o.o_orderkey AND o1.o_custkey = o.o_custkey
AND o1.o_custkey = o2.o_custkey AND l1.l_partkey = p.ps_partkey;
```

Example 4.3. A query joins three tables and has the SUBSTRING function in the WHERE clause. Its cost from the OLTP engine is 5.2×10^3*, and its execution time in OLTP engine is* 5.80s*. The OLAP engine has an execution time of* 310ms *for the same query.*

```
SELECT COUNT(*) FROM customer, nation, orders
WHERE SUBSTRING(c_phone, 1, 2) IN ('20', '40', '22', '30', '39',
     '42', '21')
AND c_mktsegment = 'machinery'
AND n_name = 'egypt' AND o_orderstatus = 'p'
AND o_custkey = c_custkey AND n_nationkey = c_nationkey;
```

The examples above show that the cost obtained from the OLTP engine does not always align with the runtime performance. In Example 4.1, the query's cost is 1.0×10^3, and the OLTP engine takes 20ms to execute. However, in Example 4.2, despite a 220× higher cost than Example 4.1, the query finishes in only 70ms, which is just 3.5× slower than Example 4.1. On the contrary, the query's cost in Example 4.3 is about 1/42 of the cost in Example 4.2. However, its execution time in OLTP engine is conversely 74× higher than the time in Example 4.2.

These discrepancies highlight two key challenges for cost-based query routing. First, the estimated cost may not be always reliable enough to reflect the true performance of execution. A high cost may not always lead to slow execution, and a low cost may not guarantee a fast execution. Therefore, using a single static cost threshold for routing may produce incorrect choices of engines.

Second, it is difficult to decide the cost threshold for routing as the cost obtained from the OLTP engine cannot estimate the performance of the OLAP engine. In the examples above, we cannot find a single threshold to route all three queries correctly. We have also considered to include the plan cost from OLAP engine. However, the OLTP cost and OLAP cost cannot be directly compared as they are generated by different systems.

4.3.2 Learning-based routing. Inspired by recent advancements in learning-based query optimization [34, 37, 53, 82, 87, 92, 93] and its successful application in real-world industrial systems [80], we propose a learning-based smart router. Based on an enhanced tree-based convolutional neural network (Tree-CNN) [54, 58], our model learns from historical query executions to predict which engine would execute a query more efficiently.

Workflow of smart router. The Figure 2 shows the workflow of the smart router, which is integrated with the workflow of veDB-HTAP and cost-based routing. Firstly, a query is sent to the OLTP engine and an OLTP plan is generated. If the query's cost is below a low threshold, it will be executed in the OLTP engine directly. We set this threshold conservatively to avoid the overhead of the learning-based routing for relatively simple queries. The remaining more complex queries will be sent to the OLAP engine. Then, the OLAP engine generates the OLAP plan and sends both OLTP and OLAP plans to the smart router. The smart router takes those plans as inputs, and predicts which engine will execute the query more efficiently. Lastly, the query is routed to the corresponding engine according to the prediction.

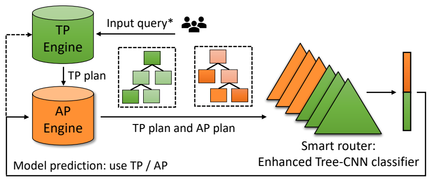

TP Engine
Input query*
TP plan
AP Engine
TP plan and AP plan
Smart router: Enhanced Tree-CNN classifier
Model prediction: use TP / AP

Figure 2: Workflow of the Learning-based Smart Router.

Model structure. Tree-CNN has been proven effective for performance prediction for queries [53, 80, 92]. Our work differs from previous studies in two major aspects: First, our model is more practical and more efficient to train because it only needs to select a better engine from two candidates according to the optimized plans, instead of predicting the actual query performance for all candidate plans. Second, our model processes and compares plans generated by two distinct engines with different characteristics. This introduces additional challenges compared to processing plans from a single engine.

Our model begins by formatting the input plans into tree structures and then encoding them using a series of convolutional layers. This encoding process transforms each plan tree into a vector representation that encapsulates key patterns and relationships within each plan. We also apply an attention mechanism to each convolutional layer to integrate runtime factors, which allows the model to prioritize the most critical aspects of the encoded features based on the current runtime conditions. Then, the results from the convolutional layers are processed through dynamic pooling and fully connected layers, which produces a final score to indicate which engine will have a better performance for the input query.To protect user privacy and mitigate the risk of overfitting during model training, user-specific information in the query plans, such as table names and column names, is anonymized.

Experimental setup and results. To train and evaluate our model, we created customized workloads based on the queries collected from ByteDance's online applications. These queries range from simple single-table queries to complex queries involving up to seven-way joins. The workloads also include queries that the cost-based routing is difficult to handle and may produce incorrect routing decisions, such as *Top-N* queries with an `ORDER BY`, `LIMIT`, or `OFFSET` clause. In our experiments, in order to test the performance of our methods across different training sizes and evaluate the generalizability of our methods, a workload of a total of 7,033 queries were utilized, with 10% of queries randomly selected as the testing dataset and varied percentages of the remaining queries were used as the training dataset. The experiments were conducted on a cluster, with the details of its configuration provided in Section 6. For all experiments, the training time was under five minutes.

We compared the routing accuracy of the learning-based routing with the cost-based routing. For cost-based routing, an exhaustive search was first performed to identify the optimal cost threshold

Training Size	**5%**	**10%**	**20%**	**50%**	**100%**
Learning-based	86.2%	87.9%	91.6%	92.4%	94.3%
Cost-based	80.2%	80.1%	80.0%	80.0%	79.1%

Table 1: Routing Accuracies with Different Training Sizes

that maximized the routing accuracy for the training set. Table 1 shows the experimental results. The learning-based router consistently outperforms the cost-based router across all training sizes, and the performance improvement is up to 15%. Additionally, the routing accuracy of the learning-based method consistently improves as the training size increases, which demonstrates the effectiveness of our training. Note that the learning-based routing can have more than 90% routing accuracy with only 20% of training data, which shows that our method is very effective when the query patterns in the training and testing workloads are similar. This is very useful since the query patterns in most of our online applications are fairly stable.

To the best of our knowledge, our work presents the first deep-learning-based query router for a production HTAP system. However, several challenges remain to be addressed before the learning-based router can be fully adopted in production applications with complex workloads. For example, a key challenge is determining when and how to retrain the model to accommodate evolving workloads. We are actively working on resolving these issues.

5 HIGHLY EFFICIENT AND ADAPTIVE QUERY PROCESSING

As the ByteHTAP users grow, we have noticed a few limitations in the query performance and resources utilization of the Flink-based OLAP engine. For example, for TPC-H 100G benchmark, we typically need 128 CPU cores to achieve 140s total running time. In comparison, a well-known HTAP system on the market (referred to as HTAP-X) only needs 32 CPU cores to achieve 80s total running time for the same dataset. This has motivated us to develop our in-house massively-parallel-processing query engine.

In contrast to Flink's batch/stream-based data processing engine, which usually aims for streaming applications, our new engine is designed with optimal analytical workload performance in mind by employing a push-based and vectorized execution engine. To maximize the CPU utilization, the execution engine adopts a coroutine-based asynchronous scheduler that can efficiently perform user-space context switch with minimal overhead. As shown later, for the above TPC-H 100G benchmark, we can achieve approximately 35s total run time with only 32 CPU cores.

One key design of the new query processing engine is to have the query optimizer and executor work together so the performance-aware features can be adaptively performed based on the compile time and runtime information, and, therefore, achieve optimal query performance in practice. Dynamically performing query re-optimization has been studied to improve the efficiency of query execution [1, 2, 83]. [83] and [2] introduced approaches in their systems to change query plans on-the-fly based on runtime statistics. BigQuery [1] uses a dynamic partitioning mechanism that intelligently chooses partitioning and degree-of-parallelism based on runtime information. Different from those works, our work mainly focuses on dynamically changing the executions of query operators without changing the query plans.

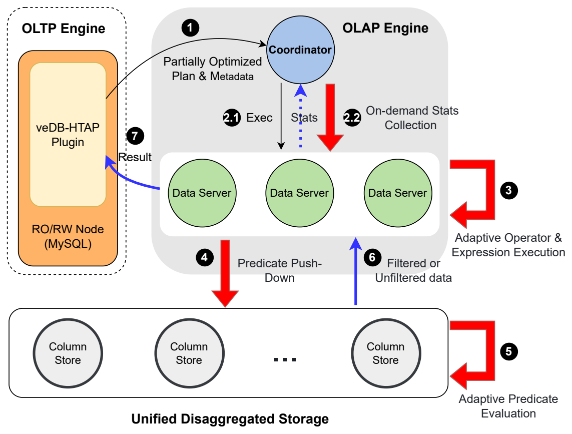

OLTP Engine
OLAP Engine
1
Partially Optimized Plan & Metadata
Coordinator
veDB-HTAP Plugin
RO/RW Node (MySQL)
7
Result
2.1 Exec
Stats
2.2 On-demand Stats Collection
Data Server
Data Server
Data Server
3
Adaptive Operator & Expression Execution
4 Predicate Push-Down
6 Filtered or Unfiltered data
Column Store
Column Store
...
Column Store
5
Adaptive Predicate Evaluation
Unified Disaggregated Storage

Figure 3: Adaptive Query Execution Workflow

Figure 3 shows the high level design of our adaptive query execution workflow. During the planning phase, the Coordinator can collect the missing column stats on-demand to generate optimal distributed plans (Section 5.1.1). Some of these stats, such as table row count, can be attached to plan and sent to the Data Server for better runtime decision. The Data Server can also collect necessary runtime stats along with its resource utilization metrics to help determine its execution behavior, such as hash table selection, disk spill decision, etc (Sections 5.1.2 to 5.2.1). In addition, the underlying storage can execute the push-down predicates in an adaptive fashion based on the resource utilizations on storage nodes to ensure the system stability and quality of service (Section 5.2.2). In summary, our adaptive query execution can be classified into two categories: (1) adaptive execution that dynamically collects and utilizes runtime stats for better query performance; (2) utilizes runtime resource information for better quality of service.

5.1 Runtime Stats-Based Adaptiveness

In this section, we will first explain how stats are collected on-demand in veDB-HTAP and then showcase how these stats are used at runtime to improve the execution efficiency.

5.1.1 On-Demand Stats Collection. In cost-based query optimizers, the estimation of operator stats is of paramount importance in generating good plans. While most current commercial DBMSs offer some form of automatic statistics collection functionality, they typically support statistics over entire column data. In our approach, we dynamically collect statistics over selection predicates that appeared in user queries, which allows us to get more accurate cardinality estimation over complex compound predicates.

We introduced our lightweight and highly effective on-demand stats collecting feature, which aims to relieve users from cumbersome stats-related operations. In addition, we leverage the cache eviction policy to retain the most frequently accessed statistics and expel stale statistics when tables get updated.

There are a couple of key challenges in this approach. First, the initial time needed to obtain on-demand stats during query planning contributes to the overall query execution time. To solve this problem, we offer an asynchronous option to the on-demand stats approach where a reasonable wait-time can be configured. If the on-demand stats job does not finish within the wait-time, the optimizer proceeds without stats. Nevertheless, the stats collecting job continues to retrieve the stats and store them in the cache for the benefit of subsequent queries.

Second, on-demand stats collection may increase system overhead and interfere with other critical user workloads running in parallel. To address this challenge, the first solution is to make on-demand stats lightweight. Based on our customers' workloads and benchmark studies, we identified three most essential stats for our optimizer: 1. Base table row count: it can be collected easily when a table is populated. 2. Selectivity on base table filters: we employ a sampling-based approach, which issues a scan request with filter predicates on a small portion of the table to obtain a sampled selectivity. We believe this is a more accurate and lightweight approach than a histogram-based method, especially in complex predicate cases such as compound predicates or LIKE predicate. To address the issue of inaccurate estimation caused by sampling, we also implemented a runtime feedback mechanism. This mechanism overwrites inaccurate selectivity estimates in our plan cache with the actual selectivity collected after query completion, thus benefiting subsequent queries. 3. Column NDV: we extended our on-demand sampling approach to issue an asynchronously approximate count distinct request on target columns. In addition, we also utilize dedicated resource work group for these stats collecting jobs to prevent resource contention from normal user queries.

5.1.2 *Adaptive Hash Table Selection.*

Hash tables are the fundamental data structures used for GROUP BY and JOIN operators. It provides constant lookup time O(1) for inserting a key-value pair and retrieving the value from a given key. If the hash table can store $ht_{capacity}$ elements in its memory block, and ht_{ndv} elements are currently inserted, the *load factor* of the hash table l can be defined as follows:

$$l = ht_{ndv}/ht_{capacity} \tag{1}$$

We call $ht_{capacity}$ the capacity and ht_{ndv} the size of the hash table.

The hash table lookup performance usually stays stable as long as its load factor stays within its maximum value l_{max}, and degrades exponentially once it exceeds this threshold [45] due to collisions [1]. Most existing work on hash tables focuses on either improving the lookup performance [11, 26, 27] or enhancing the memory efficiency [5, 12] by designing hash tables with high l_{max}. However, it is hard to find an optimal solution that satisfies both criteria, which motivates us to propose an adaptive hash table selection strategy that dynamically selects the most memory-efficient hash table based on the estimated hash table capacity at runtime with great lookup performance.

To estimate the hash table size, we utilize the HyperLogLog (HLL) algorithm [42] to calculate the number of distinct values (ndv) on the build side at runtime (i.e., ht_{ndv} in (1)). Because the HLL algorithm is able to estimate cardinalities of $> 10^9$ with a typical accuracy (standard error) of 2%, using 1.5 KB of memory, this approach works well in practice with minimal computation overhead. Once the estimated number of distinct values in hash table ht_{ndv} is calculated, based on (1), the hash table capacity $ht_{capacity}$ can be expressed as:

$$ht_{capacity} = next_{pow2}(ht_{ndv}/l_{max}) \tag{2}$$

where $next_{pow2}$ is a function to get the next power of two and l_{max} is the hash table's suggested maximum load factor.

In veDB-HTAP, we currently maintain two hash table implementations, with a higher l_{max} and a lower l_{max} separately. In order to achieve high l_{max}, additional metadata needs to be stored in the hash table, which causes extra indirections during lookup. Our experimental results have shown that, only when under the same capacity, the one with higher l_{max} exhibits worse lookup performance (around 10%) than the other. In all other cases, it consumes less memory and shows better lookup performance. Based on above findings we provide a simple but effective strategy to adaptively select the hash table implementation. In specific, before hash build, we calculate the memory capacity used for each type of hash table according to (2). If they are equal, we select the one with better lookup performance, i.e., the one with lower l_{max}, otherwise, we always select the one with smaller capacity, i.e., the one with higher l_{max}. We will provide more experimental results in Section 6.

Due to HLL's computational overhead, adaptive hash table selection may not be advantageous in all cases. It is mainly beneficial when the build-side's NDV is large, since the calculating HLL overhead is outweighed by the benefits of choosing the more efficient hash table and reduction of the hash table resize cost. When the build-side's NDV is small, both the associated HLL calculation overhead and the potential benefits are small.

5.1.3 *Adaptive Runtime Filter.*

Runtime filtering has been a widely adopted optimization feature for several decades [7, 8, 14, 83, 89]. The basic idea is to generate a bloom filter [19] over the build side table and push it down to the probe side table to pre-filter non-matching rows before the join operation to save the join cost.

It is however a non-trivial task for a query optimizer to compute the accurate cost saving achieved by pushing down the runtime filter. The classic join cardinality estimation [81] between table R and S on attribute A is $|R \bowtie S| = |R| \cdot |S|/max(NDV_R(A), NDV_S(A))$, where $|R|$ and $|S|$ represent the number of rows in R and S respectively, and $NDV_R(A)$ and $NDV_S(A)$ denote the number of distinct values of A in R and S. Assume R is the probe-side base table, and S is the build side table. In order for runtime filter to be effective, $NDV_R(A)$ needs to be larger than $NDV_S(A)$. In the case of hash join using runtime filter over column A, we have $|S| = NDV_S(A)$, and , the estimated filtered rows can hence be expressed as $|R| \cdot (1 - NDV_S(A)/NDV_R(A))$. Multiplying it by δ (the time required to scan one row) yields the basic cost savings, as expressed in (3) below:

$$cost_{saving} = |R| \cdot (1 - NDV_S(A)/NDV_R(A)) \cdot \delta, \tag{3}$$

The overhead of the runtime filter depends on the number of build-side rows, which is $NDV_S(A)$. Multiplying $NDV_S(A)$ by β (the build time per distinct value) gives the time needed to construct

[1]We limit our discussion to hash tables using open addressing method [45] for collision resolution where all hash table entries are stored in the bucket array itself, as most modern hash table is implemented in this way to achieve low memory overhead and good lookup performance.

the bloom filter. Both δ and β are session variables with default values obtained through testing on common clusters.

$$cost_{overhead} = NDV_S(A) \cdot \beta, \quad (4)$$

Since R is usually a base table, query optimizer usually can estimate $|R|$ and $NDV_R(A)$ fairly accurately by using statistics on R. However, it is challenging to obtain accurate $NDV_S(A)$ because S may be an intermediate result from a complex sub-query plan. In order to solve this issue, we collect runtime statistics over $NDV_S(A)$ at the time of building hash table on S. In this way, we can get accurate $NDV_S(A)$, which can also be used to reduce the likelihood of hash resizing when generating the runtime bloom filter [19].

Different from the work in [83], which only uses the runtime stats such as row counts on both sides to make a rough decision, we estimate the bloom filter ndv in a more accurate way.

5.2 Resource-Aware Adaptiveness

Next, we explain how the executor dynamically changes the execution behavior based on runtime resource information such as CPU usage, memory availability, etc, to improve both query performance and quality of service.

5.2.1 Adaptive Disk Spill. Our multi-tenant Data Server clusters can overcommit memory for cost efficiency, as concurrently running queries may not require their full memory allocation simultaneously. If a Data Server encounters memory shortages, disk spilling acts as a safeguard mechanism to ensure the successful execution of queries.

Most disk spill implementations [7, 8, 10, 73] employ preset memory thresholds at the query level or the operator level for *spillable* operators, such as aggregation and join, so that disk spill can take place whenever the consumed memory exceeds the thresholds. For example, the optimizer can estimate an operator's memory usage and compare it with the threshold to make the spill decision during planning. Similarly, the executor can also compare the current operator's memory consumption with this threshold to make the spill decision at runtime.

However, in real-world scenarios, selecting the right query or operator level threshold is a challenging task for several reasons. First, workloads are dynamic, and their memory consumption is unpredictable. Second, accurate cardinality estimation is difficult for the optimizer, especially with complex queries. Third, the threshold must balance preventing severe memory shortages on a Data Server with fully utilizing available resources.

This motivates us to propose an adaptive disk spill approach based on the runtime memory utilization of Data Servers. Two memory utilization thresholds, referred to as the *soft limit* and *hard limit*, are introduced. Each Data Server checks its memory usage against these thresholds at the beginning of each plan fragment execution, as well as periodically in the remaining execution process. Before the memory utilization reaches the *soft limit*, no disk spill is required, and queries can allocate memory as needed. When the memory consumption exceeds the *hard limit*, all running and incoming queries will be terminated to prevent the system from out-of-memory (OOM).

When the memory usage falls between these two limits, indicating that the system memory is under pressure but not critically low, disk spill will be triggered. However, instead of spilling all running queries, in our adaptive approach, disk spill will only be selectively applied to a subset of queries and their execution fragments. These candidate fragments are chosen based on their runtime memory usage, with the goal of bringing the overall memory consumption down below the *soft limit*. Specifically, we sort all fragments based on their execution start time in descending order, and spill the running operators in the top k fragments whose sum of consumed memory is right above a threshold. If only spilling these queries are not fast enough to achieve this goal, additional queries may be considered to spill in a more aggressive way. This adaptive approach minimizes the performance impact of disk spilling to the overall system throughput by targeting only a subset of queries, and provides a fast response to ensure system stability at critical resource pressures.

5.2.2 Adaptive Predicates Push-Down. veDB-HTAP's unified storage system is engineered to not only facilitate columnar data generation and scanning but also support a broad spectrum of push-down computations, such as filters and aggregates, which can significantly accelerate query processing by minimizing data transfer across the network, albeit at the expense of higher resource consumption. Besides push-down processing and data scans across thousands of data partitions on a storage node, the unified storage engine also needs to execute concurrent background tasks such as log replication and apply, Delta Store flush, compaction, and garbage collection on those data partitions. A crucial aspect of the unified storage engine is its ability to segregate the performance effects of these distinct tasks.

To enhance cost efficiency, the engine optimally serves a multi-tenant workload at elevated resource utilization levels. To protect it from ever getting overloaded, the multi-tenant unified storage engine coordinates with other system components, such as Data Servers and the replication framework, allowing for the selective delay, avoidance, rejection, or parameter value changing of operations based on resource utilization while guaranteeing the correctness of query executions.

One of those approaches is adaptive predicates push-down. The main idea is that during scan processing, the storage engine checks its CPU utilization periodically. If it exceeds a certain threshold, the rest of the scanned data will be returned to the Data Server without any predicate evaluation. Each data block returned from the storage possesses a flag, which indicates whether the corresponding predicates have been evaluated. If not, the predicates will be evaluated at the Data Server to ensure correctness.

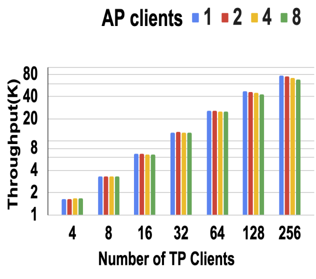
6 PERFORMANCE STUDY

In this section, a set of experiments are conducted to evaluate our system. Except for the elasticity experiment, all the experiments are conducted on a cluster of six machines with configurations listed in Table 2. Note the cluster contains four Data Servers and one Coordinator that is co-located on one of Data Servers. In addition, the cluster uses one read-write (RW) node and one read-only (RO) node to handle HTAP SQL queries.

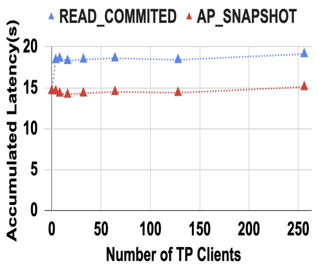
To evaluate the HTAP capabilities of our highly integrated system, we conduct experiments using a hybrid OLTP and OLAP workload based on the CH-benCHmark [35]. This benchmark combines

RW/RO Node	16v CPU, 1 NUMA node 64GB DRAM
Data Server/Coordinator	8v CPU, 1 NUMA node 32GB DRAM
Storage	Unified Storage
OS	Debian 4.14.81
Network	25Gbps Ethernet

Table 2: Cluster Configurations



(a) Perf isolation of OLTP



(b) Perf isolation of OLAP

Figure 4: Performance isolation on HTAP workloads

the unmodified TPC-C [3] workload with modified TPC-H queries. The experiments are performed on CH-benCHmark datasets with 100 warehouses. Additionally, to assess the efficiency and adaptability of our new MPP OLAP engine in handling complex query processing, we use the widely adopted TPC-H [17] benchmark with data volumes of 100GB and 1TB. We also compare the execution times of individual queries against the published results of a well-known HTAP system on the market, referred to as HTAP-X.

6.1 Hybrid OLTP and OLAP Workloads

The performance isolation between OLTP and OLAP is critical for HTAP systems. We test the performance isolation on the RW node with mixed workloads from CH-BenCHmark. It enables running both OLTP and OLAP queries on a set of shared tables in one database, and the data is firstly loaded from the OLTP side and then replicated to the OLAP side. As Figure 4 (a) shows, when the number of OLTP clients is less than 128, OLTP's throughput is almost not affected as the number of OLAP clients increases. However, when we use 256 OLTP clients, as increasing OLAP clients to issue analytical queries, there is a small OLTP throughput degradation(<10%) since CPU resources on the RW node become a bottleneck.

Next, we conducted similar experiments by submitting analytical queries to the RO node while keeping the TP workload on the RW node. We verified that there was no OLTP throughput degradation in such configurations. In production, we recommend customers to run the veDB-HTAP Plugin on MySQL RO nodes if they want to avoid potential performance degradation over OLTP workloads.

Figure 4 (b) shows the performance impact on OLAP queries when we increase the number of OLTP clients with different transaction isolation levels. As we increase the number of TP clients from 4 to 256, the accumulated AP query latency only increases slightly, which demonstrates great performance isolation in our system. In addition, we can observe that the total query latency in Read Committed (RC) isolation is about 4 seconds higher than the AP

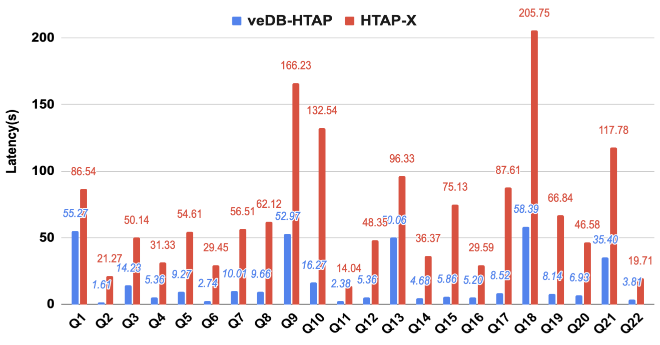
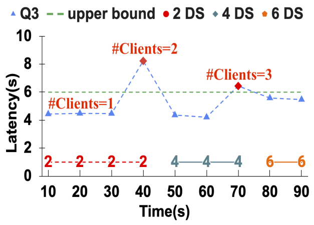
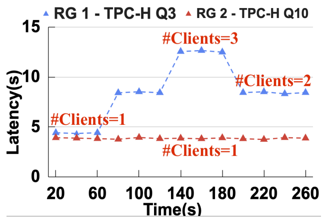


Figure 5: TPC-H 1T performance



(a) Elasticity on TPC-H Q3



(b) Resouce Group Isolation

Figure 6: Elasticity and Isolation

snapshot mode (for each query, we notice there is about 100-200ms latency delay), where AP queries can be evaluated over current data snapshots like in ByteHTAP. This performance degradation is expected since AP queries need to wait for the data snapshot with the latest committed LSN in the RC mode.

6.2 OLAP Performance

Achieving advanced OLAP performance is one of the foremost motivations of veDB-HTAP. In this section, we evaluate veDB-HTAP's OLAP performance, and compare it with HTAP-X's latest published results, using equivalent computing resources.

Figure 5 shows the result for TPC-H 1TB, where veDB-HTAP's total latency is 372 seconds, which is about 4× speedup compared to HTAP-X's 1534 seconds of latency. Except Q1, all other queries' running times on veDB-HTAP are at least 2× speedup than HTAP-X because of better query plans and more efficient query execution. We also conduct a comparison with ByteHTAP on TPC-H 1T. veDB-HTAP can achieve about 3× speedup (372 seconds vs 1034 seconds) with less than one-third computing resources. In addition, we observe that disk spill happens for memory-intensive query Q18. Without the disk spill functionality, TPCH 1T won't succeed under our experiment setup due to constrained memory configuration.

In summary, veDB-HTAP delivers a significant improvement in OLAP performance over ByteHTAP, and demonstrates substantially superior performance compared to HTAP-X.

6.3 Elasticity and Isolation

To evaluate the elasticity of our system, we use Q3 in the TPC-H 100GB benchmark with an upper-bound latency requirement of 6 seconds, under different resource group configurations. As shown in Figure 6 (a), when increasing #Clients (the number of query streams) from 1 to 2, the query latency also increases and exceeds the upper bound of the latency requirement. Meanwhile, all DS nodes' CPUs are fully utilized and the system is overloaded. After adding two more DS nodes to the resource group, the query latency drops down in a few seconds. We repeat the above process: add another two DS nodes into the resource group after increasing #Clients from 2 to 3. Similar reductions in latency are observed, demonstrating that our system scales out efficiently and rapidly.

To demonstrate the resource isolation capability, we run TPC-H Q3 and TPC-H Q10 on two different resource groups concurrently, where each resource group contains two DS nodes. As shown in Figure 6 (b), increasing the #Clients of TPC-H Q3 on resource group 1 has no impact for TPC-H Q10 on resource group 2.

6.4 Adaptive Query Processing

In this section, we conduct experiments to evaluate the effectiveness of adaptive query processing in veDB-HTAP including runtime stats collection and resource aware adaptiveness discussed in Section 5.

On-Demand Stats Collection To evaluate the effectiveness of the on-demand stats, we conduct experiments on the TPC-H1T benchmark, as shown in Figure 7. Initially, we execute all 22 queries without collecting any stats, and some queries such as Q5, Q8, and Q13 cannot even be completed due to suboptimal plans. The remaining queries are finished in 1415 seconds. Subsequently, we sequentially run all 22 queries with the on-demand stats feature. All 22 queries are completed in 417 seconds, and we observe that the plans are gradually improved starting from Q4. In the third run, since all the collected stats are saved in the cache, we achieve the optimal performance of 379 seconds, which is the same as if we manually collect all necessary stats in advance. This experiment demonstrates how our system can gradually achieve excellent performance without any human intervention.

Adaptive Hash Table Selection To evaluate the effectiveness of adaptive hash table selection, we conduct experiments on TPC-H 1T benchmark with different hash table implementations. As Figure 8 shows, neither the high nor the low load hash table is optimal for both queries. For TPC-H Q3, the high load factor hash table selection can achieve about 13% performance improvement compared with the low load factor hash table because its allocated capacity is smaller in this query. However, choosing the high load factor hash table causes 7% performance degradation for TPC-H Q14 than the low load factor hash table, since the high load factor hash table implementation has a worse lookup performance while their allocated capacity is same.

By adopting the adaptive hash table selection method, we can achieve the optimal performance and memory usage for both queries. Note that in our design, there is an overhead in calculating the estimated NDV. However, this experiment shows the gain from our approach outweighs the overhead of calculating the NDV.

Adaptive Runtime Filter Since our system uses a disaggregated storage architecture, optimizations, such as runtime filters and predicate pushdown, can significantly reduce data transfer costs from storage to Data Servers through the network. Our experiment shows that we achieve around 41.36% performance improvement

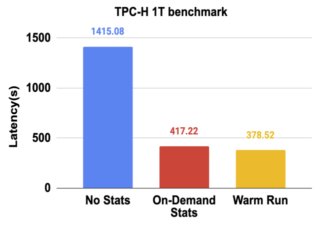


Figure 7: On-demand stats collection

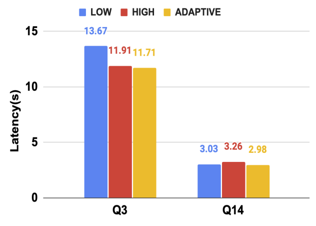


Figure 8: Adaptive hash table selection

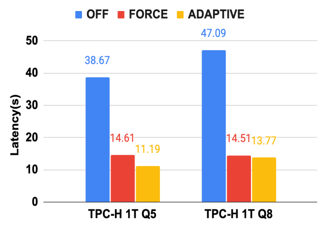


Figure 9: Adaptive runtime filter

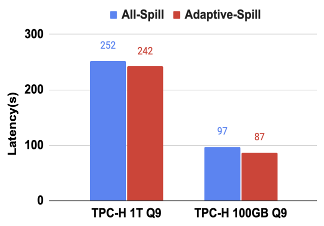
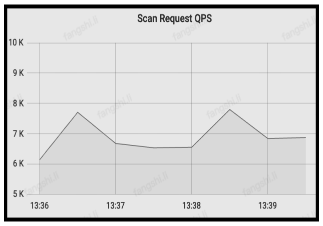
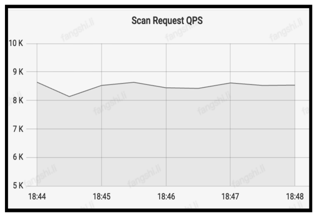


Figure 10: Adaptive disk spill



(a) Disable



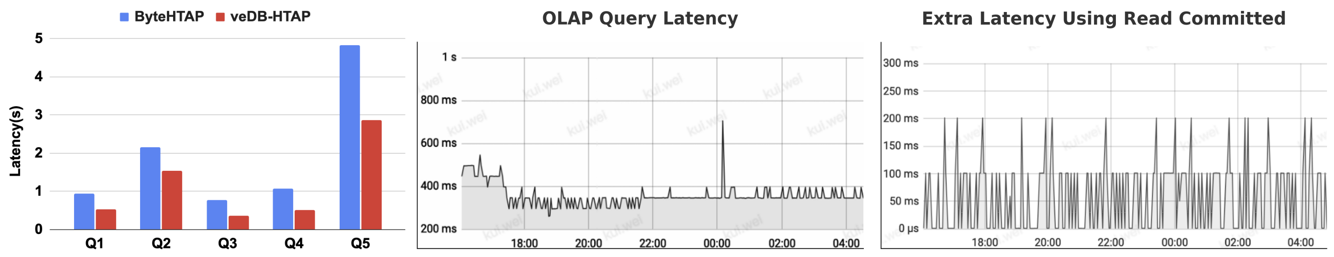
(b) Enable

Figure 11: Adaptive Predicates Push-Down

on the TPC-H 100GB benchmark with the help of the runtime filter, and even more improvement on the TPC-H 1T workload.

Since building and applying runtime filters during query execution can incur additional CPU overhead on both Data Servers and storage nodes, it's essential to choose the right joins with high selectivity as the runtime filter candidates in an adaptive manner as we described in Section 5. We conduct experiments to evaluate the adaptive runtime filter on TPCH 1T dataset since some queries contain a mix of high and low-selectivity joins. For Q5, Figure 9 shows we can achieve about 62.2% performance improvement from 38.67 seconds to 14.61 seconds by forcing runtime filters on all join conditions compared with the case without using runtime filters at all. With the adaptive runtime filter, we can further improve its latency to 11.19 seconds by choosing only highly selective joins, i.e., not using runtime filters over the join between the tables SUPPLIER and LINEITEM to avoid unnecessary runtime filter overhead. Similar to Q5, Q8 also gets performance improvement with adaptive runtime filter due to the similar reason.

Adaptive Disk Spill To demonstrate the effectiveness of adaptive disk spill, we concurrently run 1 TPC-H 1T Q9 and 8 TPC-H 100G Q9 on the cluster that has 128GB total memory. We compare the performance across three setups, namely, (1) no-spill, which



(a) Queries Latency Comparision **(b) OLAP query latency** **(c) Extra latency using Read Committed**

Figure 12: Production workloads

simply disables the disk spill feature; (2) all-spill, which spills all the queries when the soft limit is reached; and (3) adaptive-spill, which selectively spills queries as needed in an adaptive manner as explained in Section 5.2.1. As expected, the no-spill setup can't finish because of limited memory capacity. Figure 10 illustrates the results for the all-spill and adaptive-spill setups, where adaptive-spill provides 4% and 12% performance improvements over all-spill for TPC-H 1T Q9 and TPC-H 100G Q9, respectively. This is because with adaptive-spill, only a small portion of the 8 TPC-H 100G Q9 queries are spilled, which explains the 12% overall performance gain over all-spill. TPC-H 1T Q9 will always be spilled in the given environment due to insufficient system memory. Since the cluster resource is fully utilized in this experiment, more resource can be allocated to TPC-H 1T Q9 with the performance improvement of TPC-H 100G Q9 with adaptive-spill, which is the reason for the 4% performance gain of TPC-H 1T Q9.

Adaptive Predicates Push-Down In Figure 11, we present the outcomes of a workload consisting of two distinct query sets, one with predicates that can be pushed down to the storage, and the other without any predicates. We want to demonstrate how adaptive predicates push-down can effectively improve the QPS of a mixed query workload, particularly in scenarios where CPU is heavily loaded in a multi-tenant storage. Since the unified storage nodes cannot scale out as easily as the Data Servers as they contain states, it is crucial to prevent storage nodes from being overloaded under heavy workloads.

As shown in Figure 11(a), without adaptive predicates push-down, query execution under a heavy workload can result in significant variance in the storage scan QPS. In contrast, Figure 11 (b) shows that the average QPS is about 20% higher and exhibits less variation when adaptive predicates push-down is enabled. With this feature, the storage system selectively evaluates predicates to avoid overloading its CPUs. As a result, spare CPU capacity can be reallocated to other queries, thereby increasing the overall QPS.

6.5 Performance of Production Workloads

Since ByteHTAP became generally available in the middle of 2021, it has been deployed widely to serve many ByteDance internal customers, including Douyin, Lark, Finance, etc. Since veDB-HTAP was released in 2024, we are working on migrating existing customers from ByteHTAP to veDB-HTAP for performance and resource utilization improvement.

In this section, we studied veDB-HTAP's performance on several real-world workloads in our production environment. Figure 12 (a) shows the performance improvement on some representative queries from one early adopter after migrating from ByteHTAP to veDB-HTAP. This customer achieves about 2× performance improvement, by using only about 20% CPU resource for veDB-HTAP compared with ByteHTAP to serve their production workload. In Figure 12 (b), another new customer of veDB-HTAP requires read committed transaction isolation with a database over 10 Terabytes in size. Figure 12 (b) shows over a 12-hour period, the OLAP query latency consistently remained below 1 second while the system handled thousands of QPS from concurrent OLTP queries. In addition, Figure 12 (c) illustrates the additional latency incurred when waiting for the latest snapshot under the Read Committed isolation level, which remained below 200ms.

As veDB-HTAP continues to be adopted by more customers, we will gain deeper insights into the system and share our findings in the future.

7 CONCLUSIONS

veDB-HTAP is designed to be a highly integrated, efficient and adaptive HTAP system. veDB-HTAP adopts a highly integrated system architecture and provides a seamless query processing experience across OLTP and OLAP engines with a newly built cost-based and ML-based smart query router. One key design principle of veDB-HTAP is to make major system components, including query planning, query execution, and unified storage, collaborative and adaptive. The experiments show that veDB-HTAP can achieve more than 3× speedup for TPC-H while consuming only one-third of the resources compared to ByteHTAP.

ACKNOWLEDGMENTS

We would like to extend our thanks to all people who made contributions to the design and development of this veDB-HTAP system: Xiaofeng Bao, Peijun Hao, Ron Hu, Shuaiyu Jin, Yupeng Jin, Gang Liao, Bingyu Liu, Zixiong Liu, Yintong Ma, Yuanyuan Nie, Siyuan Ren, Xin Sui, Mengqi Wang, Peipei Wang, Lingyan Yin, Li Zhang, Lutong Zhang, Ruyi Zhang, Xiahao Zhang, Jie Zhou, Yihao Hu.

REFERENCES

[1] 2016. *In-memory query execution in Google BigQuery.* Retrieved Jun 9, 2025 from https://cloud.google.com/blog/products/bigquery/in-memory-queryexecution-in-google-bigquery

[2] 2017. *Adaptive Plans in Oracle Database 12c Release 1 (12.1).* Retrieved Jun 9, 2025 from https://oracle-base.com/articles/12c/adaptive-plans-12cr1
[3] 2022. *TPC-C Specification.* Retrieved February 23, 2022 from http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf
[4] 2023. *About the AlloyDB columnar engine.* Retrieved Noverber 28, 2023 from https://cloud.google.com/alloydb/docs/columnar-engine/about
[5] 2024. *Abseil Flat HashMap.* Retrieved October 25, 2024 from https://abseil.io/docs/cpp/guides/container
[6] 2024. *Apache Flink.* Retrieved March 12, 2024 from https://flink.apache.org
[7] 2024. *Apache Impala.* Retrieved February 26, 2024 from https://impala.apache.org/
[8] 2024. *Apache Spark.* Retrieved February 23, 2024 from https://spark.apache.org
[9] 2024. *ByteDance Volcano Engine.* Retrieved March 6, 2024 from https://www.volcengine.com
[10] 2024. *Clickhouse.* Retrieved February 26, 2024 from https://clickhouse.com/
[11] 2024. *ClickHouse Hash Table.* Retrieved October 25, 2024 from https://clickhouse.com/blog/hash-tables-in-clickhouse-and-zero-cost-abstractions
[12] 2024. *F14 Hash Table.* Retrieved October 25, 2024 from https://github.com/facebook/folly/blob/main/folly/container/F14.md
[13] 2024. *MySQL Heatwave.* Retrieved February 23, 2024 from https://dev.mysql.com/doc/heatwave/en/mys-hw-analytics.html
[14] 2024. *Presto.* Retrieved February 23, 2024 from https://prestodb.io
[15] 2024. *TiDB MySQL Compatibility.* Retrieved March 6, 2024 from https://docs.pingcap.com/tidb/stable/mysql-compatibility
[16] 2024. *TiFlash Disaggregated Storage and Compute Architecture and S3 Support.* Retrieved March 6, 2024 from https://docs.pingcap.com/tidb/stable/tiflash-disaggregated-and-s3
[17] 2024. *TPC-H.* Retrieved February 11, 2024 from http://www.tpc.org/tpch/
[18] 2024. *veDB for MySQL.* Retrieved March 6, 2024 from https://www.volcengine.com/product/vedb-mysql
[19] 2025. *Bloom Filter.* Retrieved Jun 2, 2025 from https://en.wikipedia.org/wiki/Bloom_filter
[20] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J Green, Monish Gupta, Sebastian Hillig, et al. 2022. Amazon Redshift re-invented. In *Proceedings of the 2022 International Conference on Management of Data.* 2205–2217.
[21] Pankaj Arora, Surajit Chaudhuri, Sudipto Das, Junfeng Dong, Cyril George, Ajay Kalhan, Arnd Christian König, Willis Lang, Changsong Li, Feng Li, et al. 2023. Flexible Resource Allocation for Relational Database-as-a-Service. *Proceedings of the VLDB Endowment* 16, 13 (2023), 4202–4215.
[22] Ronald Barber, Christian Garcia-Arellano, Ronen Grosman, Guy Lohman, C Mohan, Rene Muller, Hamid Pirahesh, Vijayshankar Raman, Richard Sidle, Adam Storm, et al. 2019. Wiser: A highly available HTAP DBMS for iot applications. In *2019 IEEE International Conference on Big Data (Big Data).* IEEE, 268–277.
[23] Ronald Barber, Christian Garcia-Arellano, Ronen Grosman, Rene Mueller, Vijayshankar Raman, Richard Sidle, Matt Spilchen, Adam J Storm, Yuanyuan Tian, Pinar Tözün, et al. 2017. Evolving Databases for New-Gen Big Data Applications. In *CIDR.*
[24] Ronald Barber, Matt Huras, Guy Lohman, C Mohan, Rene Mueller, Fatma Özcan, Hamid Pirahesh, Vijayshankar Raman, Richard Sidle, Oleg Sidorkin, et al. 2016. Wildfire: Concurrent blazing data ingest and analytics. In *Proceedings of the 2016 International Conference on Management of Data.* 2077–2080.
[25] Haoqiong Bian, Tiannan Sha, and Anastasia Ailamaki. 2023. Using Cloud Functions as Accelerator for Elastic Data Analytics. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.
[26] Altan Birler, Tobias Schmidt, Philipp Fent, and Thomas Neumann. 2024. Simple, Efficient, and Robust Hash Tables for Join Processing. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) *(DaMoN '24).* Association for Computing Machinery, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3662010.3663442
[27] Maximilian Böther, Lawrence Benson, Ana Klimovic, and Tilmann Rabl. 2023. Analyzing Vectorized Hash Tables across CPU Architectures. *Proc. VLDB Endow.* 16, 11 (July 2023), 2755–2768. https://doi.org/10.14778/3611479.3611485
[28] Nicolas Bruno, César Galindo-Legaria, Milind Joshi, Esteban Vargas, Kabita Mahapatra, Sharon Ravindran, Guoheng Chen, Ernesto Juárez, and Beysim Sezgin. 2024. Unified Query Optimization in the Fabric Data Warehouse. In *Proceedings of the 2024 ACM SIGMOD International Conference on Management of Data.* 18–30.
[29] Dennis Butterstein, Daniel Martin, Knut Stolze, Felix Beier, Jia Zhong, and Lingyun Wang. 2020. Replication at the speed of change: a fast, scalable replication solution for near real-time HTAP processing. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3245–3257.
[30] Wei Cao, Feifei Li, Gui Huang, Jianghang Lou, Jianwei Zhao, Dengcheng He, Mengshi Sun, Yingqiang Zhang, Sheng Wang, Xueqiang Wu, et al. 2022. Polardb-x: An elastic distributed relational database for cloud-native applications. In *2022 IEEE 38th International Conference on Data Engineering (ICDE).* IEEE, 2859–2872.
[31] Jianjun Chen, Yu Chen, Zhibiao Chen, Ahmad Ghazal, Guoliang Li, Sihao Li, Weijie Ou, Yang Sun, Mingyi Zhang, and Minqi Zhou. 2019. Data management at huawei: Recent accomplishments and future challenges. In *2019 IEEE 35th International Conference on Data Engineering (ICDE).* IEEE, 13–24.
[32] Jianjun Chen, Yonghua Ding, Ye Liu, Fangshi Li, Li Zhang, Mingyi Zhang, Kui Wei, Lixun Cao, Dan Zou, Yang Liu, Lei Zhang, Rui Shi, Wei Ding, Kai Wu, Shangyu Luo, Jason Sun, and Yuming Liang. 2022. ByteHTAP: Bytedance's HTAP System with High Data Freshness and Strong Data Consistency. *Proc. VLDB Endow.* 15, 12 (sep 2022), 3411–3424. https://doi.org/10.14778/3554821.3554832
[33] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. 2016. The MemSQL Query Optimizer: A modern optimizer for real-time analytics in a distributed database. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1401–1412.
[34] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. 2023. LOGER: A learned optimizer towards generating efficient and robust query execution plans. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1777–1789.
[35] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, et al. 2011. The mixed workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems.* 1–6.
[36] Sudipto Das, Vivek R. Narasayya, Feng Li, and Manoj Syamala. 2013. CPU sharing techniques for performance isolation in multi-tenant relational database-as-a-service. *Proceedings of the VLDB Endowment* 7, 1 (2013), 37–48. https://doi.org/10.14778/2732219.2732223
[37] Bolin Ding, Rong Zhu, Jingren Zhou, et al. 2024. Learned Query Optimizers. *Foundations and Trends® in Databases* 13, 4 (2024), 250–310.
[38] Aaron J Elmore, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. 2011. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data.* 301–312.
[39] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. 2011. SAP HANA database: data management for modern business applications. *Proceedings of the VLDB Endowment* 40, 4 (2011), 45–51.
[40] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, et al. 2020. TiDB: a Raft-based HTAP database. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3072–3084.
[41] Murtadha AI Hubail, Ali Alsuliman, Michael Blow, Michael Carey, Dmitry Lychagin, Ian Maxon, and Till Westmann. 2019. Couchbase analytics: NoETL for scalable NoSQL data analysis. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2275–2286.
[42] Matti Karppa and Rasmus Pagh. 2022. HyperLogLogLog: Cardinality Estimation With One Log More. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22).* Association for Computing Machinery, New York, NY, USA, 753–761. https://doi.org/10.1145/3534678.3539246
[43] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *2011 IEEE 27th International Conference on Data Engineering.* IEEE, 195–206.
[44] Jongbin Kim, Jaeseon Yu, Jaechan Ahn, Sooyong Kang, and Hyungsoo Jung. 2022. Diva: Making MVCC Systems HTAP-Friendly. In *Proceedings of the 2022 International Conference on Management of Data.* 49–64.
[45] Donald E. Knuth. 1998. *The art of computer programming, volume 3: (2nd ed.) sorting and searching.* Addison Wesley Longman Publishing Co., Inc., USA.
[46] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, et al. 2015. Oracle database in-memory: A dual format in-memory database. In *2015 IEEE 31st International Conference on Data Engineering.* IEEE, 1253–1258.
[47] Per-Åke Larson, Adrian Birka, Eric N Hanson, Weiyun Huang, Michal Nowakiewicz, and Vassilis Papadimos. 2015. Real-time analytical processing with SQL server. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1740–1751.
[48] Juchang Lee, SeungHyun Moon, Kyu Hwan Kim, Deok Hoe Kim, Sang Kyun Cha, and Wook-Shin Han. 2017. Parallel replication across formats in SAP HANA for scaling out mixed OLTP/OLAP workloads. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1598–1609.
[49] Kitaek Lee, Insoon Jo, Jaechan Ahn, Hyuk Lee, Hwang Lee, Woong Sul, and Hyungsoo Jung. 2023. Deploying Computational Storage for HTAP DBMSs Takes More Than Just Computation Offloading. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1480–1493.
[50] Chen Luo, Pinar Tözün, Yuanyuan Tian, Ronald Barber, Vijayshankar Raman, and Richard Sidle. 2019. Umzi: Unified Multi-Zone Indexing for Large-Scale HTAP. In *EDBT.* 1–12.
[51] Zhenghua Lyu, Huan Hubert Zhang, Gang Xiong, Gang Guo, Haozhou Wang, Jinbao Chen, Asim Praveen, Yu Yang, Xiaoming Gao, Alexandra Wang, et al. 2021. Greenplum: A Hybrid Database for Transactional and Analytical Workloads. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data.* 2530–2542.
[52] Darko Makreshanski, Jana Giceva, Claude Barthels, and Gustavo Alonso. 2017. BatchDB: Efficient isolated execution of hybrid OLTP+ OLAP workloads for interactive applications. In *Proceedings of the 2017 ACM International Conference on Management of Data.* 37–50.

[53] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*. 1275–1288.

[54] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: a learned query optimizer. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1705–1718.

[55] Norman May, Alexander Böhm, and Wolfgang Lehner. 2017. SAP HANA–The Evolution of an In-Memory DBMS from Pure OLAP Processing Towards Mixed Workloads. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)* (2017).

[56] John Meehan, Nesime Tatbul, Stan Zdonik, Cansu Aslantas, Ugur Cetintemel, Jiang Du, Tim Kraska, Samuel Madden, David Maier, Andrew Pavlo, et al. 2015. S-Store: Streaming Meets Transaction Processing. *Proceedings of the VLDB Endowment* 8, 13 (2015).

[57] Elena Milkai, Yannis Chronis, Kevin P. Gaffney, Zhihan Guo, Jignesh M. Patel, and Xiangyao Yu. 2022. How Good is My HTAP System?. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*. 1810–1824.

[58] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

[59] Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogesh Mahajan, Soubhik Chakraborty, Hemant Bhanawat, and Kishor Bachhav. 2017. SnappyData: A Unified Cluster for Streaming, Transactions and Interactice Analytics.. In *CIDR*.

[60] Niloy Mukherjee, Shasank Chavan, Maria Colgan, Dinesh Das, Mike Gleeson, Sanket Hase, Allison Holloway, Hui Jin, Jesse Kamp, Kartik Kulkarni, et al. 2015. Distributed architecture of oracle database in-memory. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1630–1641.

[61] Vivek Narasayya and Surajit Chaudhuri. 2022. Multi-tenant cloud data services: state-of-the-art, challenges and opportunities. In *Proceedings of the 2022 International Conference on Management of Data*. 2465–2473.

[62] Vivek Narasayya, Sudipto Das, Manoj Syamala, Badrish Chandramouli, and Surajit Chaudhuri. 2013. Sqlvm: Performance isolation in multi-tenant relational database-as-a-service. In *CIDR 2013*.

[63] Vivek Narasayya, Sudipto Das, Manoj Syamala, Surajit Chaudhuri, Feng Li, and Hyunjung Park. 2013. A demonstration of SQLVM: performance isolation in multi-tenant relational database-as-a-service. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 1077–1080.

[64] Vivek Narasayya, Ishai Menache, Mohit Singh, Feng Li, Manoj Syamala, and Surajit Chaudhuri. 2015. Sharing buffer pool memory in multi-tenant relational database-as-a-service. *Proceedings of the VLDB Endowment* 8, 7 (2015), 726–737.

[65] Krishna Kantikiran Pasupuleti, Boris Klots, Vijayakrishnan Nagarajan, Ananthakiran Kandukuri, and Nipun Agarwal. 2022. High Availability Framework and Query Fault Tolerance for Hybrid Distributed Database Systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3451–3460.

[66] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. 2017. Self-Driving Database Management Systems. In *CIDR*, Vol. 4. 1.

[67] Adam Prout, Szu-Po Wang, Joseph Victor, Zhou Sun, Yongzhu Li, Jack Chen, Evan Bergeron, Eric Hanson, Robert Walzer, Rodrigo Gomes, and Nikita Shamgunov. 2022. Cloud-Native Transactions and Analytics in SingleStore. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*. 2340–2352.

[68] Vijayshankar Raman, Gopi Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M Lohman, et al. 2013. DB2 with BLU acceleration: So much more than just a column store. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1080–1091.

[69] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. 2020. Adaptive HTAP through elastic resource scheduling. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2043–2054.

[70] Hemant Saxena, Lukasz Golab, Stratos Idreos, and Ihab F Ilyas. 2021. Real-Time LSM-Trees for HTAP Workloads. *arXiv preprint arXiv:2101.06801* (2021).

[71] Hemant Saxena, Lukasz Golab, Stratos Idreos, and Ihab F Ilyas. 2023. Real-time LSM-trees for HTAP workloads. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1208–1220.

[72] Mohit Saxena, Benjamin Sowell, Daiyan Alamgir, Nitin Bahadur, Bijay Bisht, Santosh Chandrachood, Chitti Keswani, G Krishnamoorthy, Austin Lee, Bohou Li, et al. 2023. The Story of AWS Glue. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3557–3569.

[73] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. 2019. Presto: SQL on Everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1802–1813. https://doi.org/10.1109/ICDE.2019.00196

[74] Sijie Shen, Rong Chen, Haibo Chen, and Binyu Zang. 2021. Retrofitting High Availability Mechanism to Tame Hybrid Transaction/Analytical Processing. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 219–238.

[75] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. 2012. 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*. 731–742.

[76] Jason Sun, Haoxiang Ma, Li Zhang, Huicong Liu, Haiyang Shi, Shangyu Luo, Kai Wu, Kevin Bruhwiler, Cheng Zhu, Yuanyuan Nie, et al. 2023. Accelerating Cloud-Native Databases with Distributed PMem Stores. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 3043–3057.

[77] Lalitha Viswanathan, Alekh Jindal, and Konstantinos Karanasos. 2018. Query and resource optimization: Bridging the gap. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1384–1387.

[78] Jianying Wang, Tongliang Li, Haoze Song, Xinjun Yang, Wenchao Zhou, Feifei Li, Baoyue Yan, Qianqian Wu, Yukun Liang, ChengJun Ying, Yujie Wang, Baokai Chen, Chang Cai, Yubin Ruan, Xiaoyi Weng, Shibin Chen, Liang Yin, Chengzhong Yang, Xin Cai, Hongyan Xing, Nanlong Yu, Xiaofei Chen, Dapeng Huang, and Jianling Sun. 2023. PolarDB-IMCI: A Cloud-Native HTAP Database System at Alibaba. *Proc. ACM Manag. Data* 1, 2, Article 199 (jun 2023), 25 pages. https://doi.org/10.1145/3589785

[79] Louis Woods, Istvan Zsolt, and Gustavo Alonso. 2014. Ibex: An Intelligent Storage Engine with Support for Advanced SQL Off-loading. *Proceedings of the VLDB Endowment* 7, 11 (2014), 963–974. https://doi.org/10.14778/2732967.2732972

[80] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query Execution Time Prediction in Amazon Redshift. In *Companion of the 2024 International Conference on Management of Data*. 280–294.

[81] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2022. FactorJoin: A New Cardinality Estimation Framework for Join Queries. arXiv:2212.05526 [cs.DB] https://arxiv.org/abs/2212.05526

[82] Xianghong Xu, Zhibing Zhao, Tieying Zhang, Rong Kang, Luming Sun, and Jianjun Chen. 2023. COOOL: A Learning-To-Rank Approach for SQL Hint Recommendations. *5th International Workshop on Applied AI for Database Systems and Applications* (2023).

[83] Maryann Xue, Yingyi Bu, Abhishek Somani, Wenchen Fan, Ziqi Liu, Steven Chen, Herman van Hovell, Bart Samwel, Mostafa Mokhtar, RK Korlapati, et al. 2024. Adaptive and Robust Query Execution for Lakehouses at Scale. *Proceedings of the VLDB Endowment* 17, 12 (2024), 3947–3959.

[84] Jiacheng Yang, Ian Rae, Jun Xu, Jeff Shute, Zhan Yuan, Kelvin Lau, Qiang Zeng, Xi Zhao, Jun Ma, Ziyang Chen, et al. 2020. F1 Lightning: HTAP as a Service. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3313–3325.

[85] Xinjun Yang, Yingqiang Zhang, Hao Chen, Feifei Li, Bo Wang, Jing Fang, Chuan Sun, and Yuhui Wang. 2024. PolarDB-MP: A Multi-Primary Cloud-Native Database via Disaggregated Shared Memory. In *Proceedings of the 2024 ACM SIGMOD International Conference on Management of Data*. 295–308.

[86] Xinjun Yang, Yingqiang Zhang, Hao Chen, Chuan Sun, Feifei Li, and Wenchao Zhou. 2023. PolarDB-SCC: A Cloud-Native Database Ensuring Low Latency for Strongly Consistent Reads. *Proc. VLDB Endow.* 16, 12 (aug 2023), 3754–3767. https://doi.org/10.14778/3611540.3611562

[87] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD/PODS '22)*. Association for Computing Machinery, New York, NY, USA, 931–944. https://doi.org/10.1145/3514221.3517885

[88] Zhifeng Yang, Quanqing Xu, Shanyan Gao, Chuanhui Yang, Guoping Wang, Yuzhong Zhao, Fanyu Kong, Hao Liu, Wanhong Wang, and Jinliang Xiao. 2023. OceanBase Paetica: A Hybrid Shared-Nothing/Shared-Everything Database for Supporting Single Machine and Distributed Cluster. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3728–3740. https://doi.org/10.14778/3611540.3611560

[89] Chaoqun Zhan, Maomeng Su, Chuangxian Wei, Xiaoqiang Peng, Liang Lin, Sheng Wang, Zhe Chen, Feifei Li, Yue Pan, Fang Zheng, and Chengliang Chai. 2019. AnalyticDB: real-time OLAP database system at Alibaba cloud. *Proc. VLDB Endow.* 12, 12 (aug 2019), 2059–2070. https://doi.org/10.14778/3352063.3352124

[90] Chao Zhang, Guoliang Li, and Tao Lv. 2024. HyBench: A new benchmark for HTAP databases. *Proceedings of the VLDB Endowment* 17, 5 (2024), 939–951.

[91] Ning Zhang, Junichi Tatemura, Jignesh Patel, and Hakan Hacigumus. 2014. Re-evaluating designs for multi-tenant OLTP workloads on SSD-basedI/O subsystems. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 1383–1394.

[92] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A learning-to-rank query optimizer. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1466–1479.

[93] Rong Zhu, Lianggui Weng, Bolin Ding, and Jingren Zhou. 2024. Learned Query Optimizer: What is New and What is Next. In *Companion of the 2024 International Conference on Management of Data*. 561–569.