Rethink Query Optimization in HTAP Databases

Anonymous Submission #182

Abstract

The advent of data-intensive applications has fueled the evolution of hybrid transactional and analytical processing (HTAP). To support mixed workloads, mainstream HTAP databases typically maintain two data copies that are specially tailored for data freshness and performance isolation. In particular, a copy in a row-oriented format is well-suited for OLTP workloads, and a second copy in a column-oriented format is optimized for OLAP workloads. Such a hybrid design opens up a new design space for query optimization: plans can be optimized over different data formats and can be executed over isolated resources, which we term ***hybrid plans***.

In this paper, we demonstrate that *hybrid plans* can largely benefit query execution (e.g., up to 11× speedups in our evaluation). However, these benefits cannot be fulfilled, or will be at the cost of sacrificing data freshness and performance isolation since traditional optimizers cannot precisely model and plan the execution of analytical queries on real-time updated HTAP databases.

Therefore, we propose **Metis**, an HTAP-aware optimizer. We demonstrate, both theoretically and experimentally, that using the proposed optimizations in Metis, a system can benefit from hybrid plans, and these optimizations are robust to the changes in workloads without damaging HTAP properties.

CCS Concepts

• **Information systems → Data management systems**; Database management system engines; Database query processing; Query optimization.

Keywords

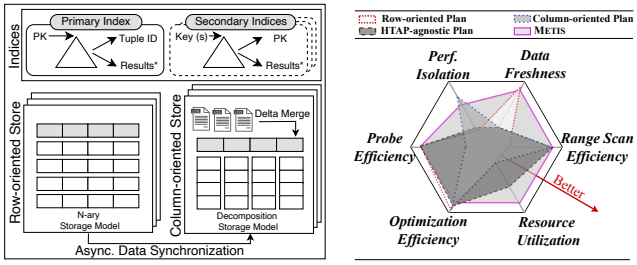
HTAP database, hybrid physical format, query optimization

ACM Reference Format:

Anonymous Submission #182. 2018. Rethink Query Optimization in HTAP Databases. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Today, data-intensive applications often utilize vast amounts of data for diverse real-time business tasks (e.g., data-driven decisions [8, 16, 31]), necessitating weaving analytical and transactional processing techniques together [50]. In response, many recent academic and industrial efforts have been devoted to developing hybrid transactional and analytical processing (HTAP) systems [6, 18, 35, 37, 39, 46–48, 55, 57, 62–64, 67, 69, 73], which are expected to provide ❶ prompt analysis of fresh data and ❷ isolate the performance of interleaved workloads.



(a) Hybrid Physical Layout. **(b) Performance Comparison.**

Figure 1: Figure 1a shows an example of the hybrid physical layout in modern HTAP systems (e.g., SQL Server [33], TiDB [37]): the row-oriented tables are well-suited for updates and probes; a second copy in a column-oriented layout is optimized for range scan. Leveraging a hybrid physical layout, Metis strikes a practical balance between performance, isolation, and freshness for HTAP (see 1b).

A practical HTAP database generally consists of an online transactional processing (OLTP) engine that supports high throughput transaction processing, and an online analytical processing (OLAP) engine supports complex analytics with low latency. To handle mixed workloads efficiently, mainstream HTAP databases (e.g., SQL Server [47], TiDB [37], SAP HANA [48], ByteHTAP [18], and AlloyDB [35]) typically employ the two engines with specialized data stores. An example is shown in Figure 1a: a row-oriented store (for short, row store) that stores data tuples as rows are optimized for operating on a single record at a time and accessing many attributes, favor for OLTP; a column-oriented store (for short, column store) that stores tuples attribute-at-a-time in columns is optimized for operating on a few attributes with massive rows, favor for OLAP. Given such a design, different workloads can be independently processed on their desirable stores, providing isolations in the storage layer.

Unfortunately, restricting each workload to its specialized store leaves much of the performance potential unrealized. This is because, for read-only queries, both the row and column store can significantly outperform one another based on the characteristics of system implementations and workloads [5, 33, 44] (see our experimental results in §2.2). Thus, even for a single analytical query, neither row store nor column store can be the optimal data source since row store may be ideal for a portion of sub-queries, and column store can be optimal for the rest.

To reach the full potential of the hybrid physical layouts, several HTAP systems [33, 37] have integrated the two stores as alternative data access methods in their query optimizers to generate ***hybrid plans*** for queries. Specifically, a hybrid plan allows a single query to retrieve data from both the row and column stores simultaneously and calculate results based on a consistent snapshot (data view).

Motivation. Nevertheless, existing approaches [33, 37] select access paths and do query optimizations simply based on selectiv-

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ity [44], neglecting the ***data dynamicity*** of HTAP databases. In §3, we show that blindly pursuing hybrid plans can easily make the generated plans sub-optimal and damage important HTAP properties (e.g., performance isolation).

Therefore, in this paper, we take the first step to systematically study the optimizations of hybrid plans given HTAP databases. Our key insight is that, to keep hybrid plans efficient, we should put *data dynamicity* into the design of the query optimizer by capturing the effect of the mutual relationship between reads (i.e., read-only queries) and writes (i.e., write transactions). Based on our insight, we identify three key challenges below.

The first challenge is *how to precisely model the cost of data access paths when new writes continuously update the replicated data copy for reads?* Modern HTAP databases support timely updates in the read-optimized column store (i.e., the data copy for reads) through a separate delta store (or a write-optimized storage layer). Delta store accumulates updates continuously and periodically merges them into the columnar storage (see Figure 1a, detailed in §2.1). This delta-main architecture makes the traditional cost model imprecise for evaluating the cost of data access paths: there is no fixed selectivity threshold for access path selection; rather, the division depends on the workload's dynamicity (i.e., the concurrency of writes).

To address this challenge, we propose a new cost model incorporating the data dynamicity into the optimizer: Demain (**Del**ta-**main** model). Demain captures the performance of select operators in both delta stores and column stores and thus can efficiently guide the access path selection.

The second challenge is *how to optimize data freshness and execution time together, especially when new writes are propagated asynchronously?* Generally, in an HTAP database, optimizing execution time can be at the cost of data freshness. This is because, due to the nature of data replication, the visibility of new writes in the column store (i.e., the data copy for reads) is always delayed. Hence, even if the column store may outperform the row store (i.e., the data copy for writes) on the sequential scan, the execution must be blocked until the new writes are fully synchronized to the column store, leading to a longer response latency.

Multiple existing works [37, 63, 67] strive to minimize the visibility delay by evolving their system architectures. However, depending on the deployment, the problem is still pronounced (e.g., 10*s* delay in DB2 IDAA [15], 8*mins* delay in production at Google [73], 606*ms* delay in experiments at ByteDance [18]).

For this challenge, we propose a new visibility-aware plan selection algorithm. It firstly estimates the visibility delay between the row store and column store based on the ongoing and predicted workload characteristics. When optimizing queries, it advances the query performance by pre-executing plans on the available data, thus masking the notorious visibility delays.

The final challenge is *how to ensure isolated performance between reads and writes when query plans are hybrid?* A strawman approach is using a pre-defined quota for the reads in row stores (i.e., the data copy for writes). For example, TiDB limits the default access table size on its row store for the OLAP workload to at most 500 *MB* [37]. However, manual intervention cannot effectively utilize resources while reducing query latency. A configuration that works well for one workload is unlikely to work well for another.

We develop our query re-optimization approach for hybrid plans in HTAP databases. Instead of scheduling resources [63] or limiting resource usage [37, 47, 55], our re-optimization approach can automatically adapt to the workload shift. In our approach, when a high resource contention is detected, it re-optimizes the plans by opportunistically combining efficient sub-plans of previously-optimized plans into a good new plan, which can alleviate the resource contention without a whole plan re-optimization.

System Integration. We combine all these new optimization techniques into our prototype: Metis[1], an HTAP-aware plan optimizer. Metis is developed based on the storage that supports both on-disk row store and column store. Updates are propagated from the row store to the column store continuously and asynchronously. We detail our storage model of the integrated system in §5.

Overall, Metis captures the *data dynamicity* to keep hybrid plans efficient. Metis pushes the boundary of traditional optimizers' design space by adding data freshness and performance isolation as new design goals. Figure 1b compares Metis, row-oriented, column-oriented, and HTAP-agnostic plans. Among them, Metis achieves a practical point in the design space and can speed up analytical queries in HTAP databases without sacrificing HTAP properties.

Contributions. To the best of our knowledge, this paper provides the first treatment of efficiently accommodating hybrid plans in HTAP databases. Our contributions are four-fold:

- We systematically analyze hybrid plans given HTAP databases.
- We develop Metis, along with a new cost model (Demain), a new visibility-aware plan selection algorithm, and a new plan re-optimization approach to ensure performance isolation.
- We extensively evaluate Metis under various workloads, including CH-benCHmark [22], TPC-DS [24], and YCSB [29]. The evaluation results demonstrate the effectiveness of Metis: it can generate efficient and HTAP-friendly hybrid plans; the plans are robust to the change of workloads and will not damage the properties of HTAP databases.
- Metis can be a practical template for the future adoption of HTAP-aware query optimizations in other HTAP databases.

The rest of the paper is organized as follows. §2 discusses the background of HTAP databases and the motivation for hybrid plans. §3 details the problems of HTAP-agnostic hybrid plans. §4 provides an overview of Metis. §5 discusses our new cost model. §6 presents our visibility-aware algorithm and re-optimization approach. §7 evaluates the performance of Metis. Finally, a discussion of related works is available in §8, and §9 concludes the paper.

2 BACKGROUND AND MOTIVATION

In this section, we first provide an overview of hybrid data formats in HTAP databases and show the motivation for hybrid plans. For discussions on the related works, we refer readers to §8.

2.1 Hybrid Data Format in HTAP

Following the philosophy of *"one size doesn't fit all"* [58], many of the state-of-the-art HTAP systems (e.g., SQL Server [47], TiDB [37], SAP HANA [48], Oracle Dual [46], Vegito [67], Janus [10], UniStore [39], L-store [64], IBM DB2 IDAA [15], PolarDB-IMCI [21], F1 Lightning [73], and AlloyDB [35]) utilize multiple physical de-

[1] Metis was the Titan goddess of good counsel, planning, and wisdom, which signifies our optimizations for generating an efficient execution plan.

signs to handle complicated HTAP workloads efficiently. In this paper, we focus on typical implementations of row stores and column stores, as they are good standards and are implemented by almost all the HTAP databases with hybrid data formats. We summarize their common features and optimizations below.

Row store and indices. Row stores that store all attributes for a single tuple contiguously are ideal for write-intensive workloads. In existing HTAP databases, write transactions are all handled by row stores, and the databases do not support independent write transactions on column stores. In this paper, we follow this standard and do not consider the potential of optimizing write transactions across hybrid physical layouts but focus on read-only queries.

We consider indices as a part of the row store, which is critical to performance. For example, *B+* tree indices provide ordering of data based on the key columns in the index and allow efficient lookups. Compared to the column store, retrieving data through the *B+* tree can benefit I/O bandwidth when the size of the results set is small. *B+* trees can also provide a sort order when querying specified columns.

Column store and its implementations. Recent years have witnessed the popularity of column stores to speed up analytical queries. The rationale is that the column-oriented data format can reduce I/O costs when operating on a few attributes with massive rows. Furthermore, a column store can absorb a series of optimizations on its scan operator, including vectorized execution [13, 41, 51] and working over compressed data [3, 4].

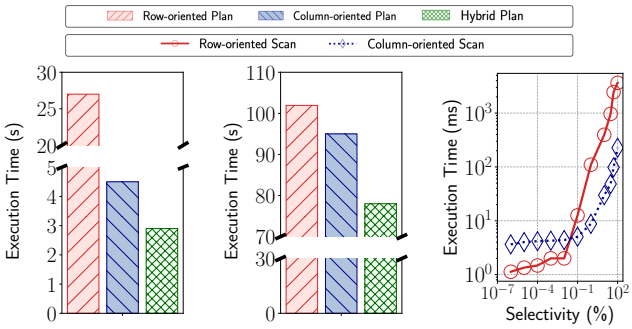
Unlike traditional data warehouses (e.g., [28, 71]), most HTAP databases opt to exclude secondary indexing over their column store since it incurs intolerable overhead and complexity of real-time updates [45]. Our paper considers sequential column scans as the major access paths in column stores. However, our methodology is general to the implementations.

Delta store and data synchronization. As discussed previously (§1), the delta store is a common and important by-production of HTAP databases [37, 50]. Principally, the delta store is designed to support real-time updates on column stores efficiently. As the column stores are heavily read-optimized, a write-optimized delta store (or a write-optimized storage layer) is necessary to keep column stores (even loosely) synchronized with row stores. Otherwise, column stores may never catch up with the state of row stores due to the gap in write efficiency.

We argue that delta store is universal in HTAP databases, e.g., delta stores in SQL Server and ByteHTAP, deltaTree in TiDB, L2-delta in SAP HANA, a write-optimized storage layer (i.e., organizing data in insertion order) in PolarDB-IMCI, and transaction maps in Oracle Dual. We consider such an HTAP-specific design as a significant distinction of our Demain model compared to other existing cost models when performing access path selection.

Given the delta-main architecture of column stores, updates generated from row stores are first written into the delta stores and periodically merged into the underlying read-optimized column storage (usually in the background).

When performing a scan on column store, queries first retrieve fresh data from delta stores and combine them with the results set from read-optimized columns to generate a fresh data view.



(a) CH-benCHmark Q7 (b) TPC-DS Q72 (c) Access Path Selection

Figure 2: Motivation of using the hybrid plan in an HTAP database: neither the row-oriented plan nor the column-oriented plan could be optimal for a number of given queries (e.g., 2a and 2b). One of the reasons is that row-oriented operations may outperform column-oriented operations when query predicates are selective (e.g., see 2c).

2.2 Motivation of Hybrid Plans

We now experimentally motivate hybrid plans and show practical scenarios where hybrid plans take effect. For theoretical analysis, we refer readers to the discussions of the Demain model (§5).

We show the potential of hybrid plans using HTAP-agnostic hybrid plans on a static database without real-time updates. Specifically, HTAP-agnostic hybrid plans are generated by simply adding column scans as an alternative access path into the cost model of row stores without considering the data dynamicity of HTAP databases. When putting the plans into the HTAP context, such plans can lead to sub-optimal performance, which we study in §3.

We did the experiments on two well-studied benchmarks: CH-benCHmark [22] and TPC-DS [24]. Both of them contain multiple queries with wide variations in complexity and range of scanned data. We calculated the speed-ups for each query by comparing the execution time of the hybrid plan to the faster one of the row- and column-oriented plans. Detailed experiment configurations for hardware, software, and workloads are shown in §7.

The evaluation results show: in CH-benCHmark, nine queries (i.e., 40.9%), out of twenty-two, benefit from hybrid plans and achieve 1.68× speedups in geometric mean; in TPC-DS, seventy-seven queries (i.e., 77.8%), out of ninety-nine, benefit from hybrid plans and achieve 3.06× speedups. We show two representative queries from each workload in Figure 2a and Figure 2b.

Based on our experiments, we conclude three factors that motivate the desirability of hybrid plans. The first factor is the diversity of data access patterns inside a single query. When a query joins multiple tables, the variance of data size and query selectivity[2] on each table motivates using different access paths for different tables. The performance comparison of the row- and column-oriented scan with different selectivity is shown in Figure 2c.

The second factor is the division of data schema. Star schemas [34, 59] and snowflake schemas [49, 70], as two successful templates, provide a clear division between dimension tables and fact tables, where dimension tables are most likely to join fact tables with

[2]Same as previous papers [45, 66], we say a predicate (filter) is selective (or its selectivity is low) when the result set has few qualifying tuples. A selectivity that ranges from 0% to 100% signifies the percentage of qualifying tuples in the database.

their primary keys. Therefore, when using such schemas in HTAP databases, retrieving data from row stores (with the primary indices) for dimension tables and retrieving data from column stores for fact tables can be a competitive candidate for the optimal plan.

The third factor is the queries' requirement for physical properties (e.g., sort order). When a query requires specific properties on a portion of tables, either demand by user requests (e.g., "order by" in SQL) or demand by an inside operator (e.g., sort-merge join requires ordered data), specific stores (e.g., a *B+* tree index that delivers sorted data) have the opportunities to outperform the others on these tables, leading to hybrid plans.

3 Problems of HTAP-agnostic Hybrid Plans

In this section, we study the performance issues caused by the mutual relationship between reads (read-only queries) and writes (write transactions) when using HTAP-agnostic hybrid plans. In particular, we study the impact of writes on reads in §3.1 and §3.2, and in turn, the impact of reads on writes in §3.3. Based on the study, we motivate the desiderata of HTAP-aware plans.

3.1 Impact of Data Synchronization

Data synchronization causes read amplification on column scans, changes the selectivity division between column stores and row stores, and thus influences the decisions on access path selection.

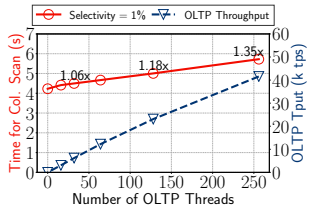
Practically, an HTAP database can trade write performance for read by adopting different delta-merge strategies. For instance, database administrators can set a rigid boundary for the size of delta stores and enforce delta merge operations immediately when the size of delta stores exceeds the limitations. Thus, the effect of read amplification on column scans can be bounded at a low level. However, such an approach must block writes (known as write stalls [54, 74]) when the write workloads are heavy, and the speed of delta merge cannot catch up with the new writes.

Instead of imposing such a hard boundary, we explore the impact of data synchronization from the optimizer's view, which should be general to the underlying strategies. We assume that the database engines merge delta periodically in the background. For a real-time workload, we consider OLTP write concurrency as the major factor contributing to the overhead of data synchronization.

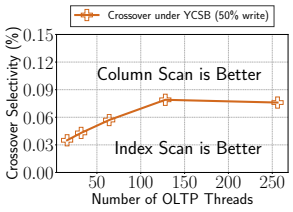
In our experiments, we executed the YCSB [29] workload for OLTP to fine-tune its concurrency and read/write ratio and used a plain query (i.e., the *Q2* in §7.1) for OLAP. Specifically, the OLAP query scans a table in the database with a predictor to control the selectivity. We run OLTP workloads for 10 minutes to warm up. Figure 3a shows the results: the execution time for column scans increased proportionally with the concurrency of OLTP workloads.

We then put the impact of data synchronization into the picture of access path selection. Figure 3b shows the results. Same as in previous papers [45], we define selectivity crossover as the selectivity that the row store and column have an identical execution time. Thus, the column store should be optimal for queries with higher selectivity than the crossover. The results show: as the concurrency grows, the crossover point rises to higher selectivity at the beginning and plateaus eventually when the throughput of data synchronization is close to saturation.

Takeaways. Retrieving data from delta stores causes additional overhead to column scan, which is critical to access path selection.



(a) Impact of OLTP Concurrency on Column Scan (YCSB 50% write).



(b) Impact of OLTP Concurrency on Crossover Selectivity.

Figure 3: Impact of Delta Store (Read Amplifications).

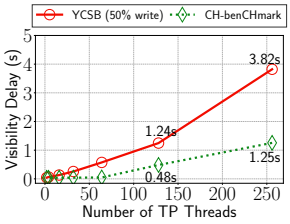


Figure 4: Freshness Loss

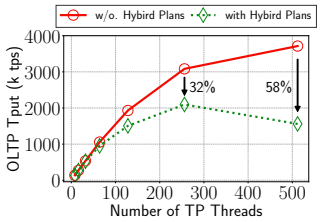


Figure 5: Throughput Drop

3.2 Impact on Data Freshness

Ensuring high data freshness is one of the most important design goals of real-time HTAP databases. Generally, in HTAP, row stores have better data freshness than column stores as all data are generated on row stores and then propagated into column stores asynchronously. As suggested by previous papers [37, 50, 73], in most deployment scenarios, such a freshness loss can not be ignored (§1).

We studied the visibility delay, defined as the time delay between an update committed on the row store and when queries on the column store can read that update. We report the 99.9th visibility delay (in ten seconds) of our integrated HTAP databases. The results are shown in Figure 4. Overall, the visibility delay increased (i.e., positively correlated) with the concurrency of OLTP workloads due to the overhead of processing more data.

Our observation is that, from the optimizer's perspective, queries can always enjoy the best data freshness by either executing queries on the row store or blocking the query execution on the column store until all new data becomes visible. However, it introduces a new opportunity for hybrid plans: an optimizer can opt to pre-execute a portion of sub-plans on the row store while optimizing the execution time for the rest of the plans on desirable data sources without blocking the entire plan. We detail our visibility-aware plan selection algorithm in §6.1.

3.3 Impact on Performance Isolation

Another essential property of HTAP databases is performance isolation, which is critical in providing independent service-level-agreement for both OLTP and OLAP. As pointed out by several real-world studies [16, 19, 60], in an HTAP application, the OLTP service always serves mission-critical tasks. Its performance should be maintained in the face of OLAP workloads.

Without hybrid plans, operations for retrieving data in each kind of workload are handled by separate stores. In this case, database administrators can simply assign hardware resources (e.g., CPU cores) to each store with resource management tools (e.g., Cgroup [43]) or deploy stores across machines to provide isolation by nature, along with independent scalability for OLTP and OLAP.

Desiderata	Roadmaps of Metis	Design Knobs	Proposed Solution
Low Execution Time	Access Path Selection	Cost Model	Demain Model
High Data Freshness	Masking the Cost of Visibility Delay	Plan Selection Algo.	Visibility-aware Optimizations
Strong Performance Isolation	Restricting the Abuse of Hybrid Plans	Re-optimization Algo.	Proactive Re-optimization with Plan Stitch

Table 1: Desideratas, roadmaps, design knobs, and solutions of Metis for efficient query optimization in HTAP Databases.

However, when using hybrid plans, such isolation is broken. Issues are two-fold. The first is the performance drop in OLTP. Figure 5 shows the OLTP throughput loss of the HTAP databases on the CH-benChmark workload (see §7 for detailed configurations). When the workload for OLTP is light (i.e., less than 256 OLTP threads), hybrid plans have little effect on OLTP throughput. Things change when loads of row stores are closed to be saturated: hybrid plans impose an OLTP throughput drop (up to 58%) due to the competition for both physical resources and logical resources (e.g., latches in internal data structures). The second issue is the inefficiency of hybrid plans. Under high contention, hybrid plans that operate row-oriented scans must wait for the schedule, causing their performance to fall short of expectations.

These two new issues are specific to HTAP databases and may not be pronounced in traditional data warehouses that only perform read-only queries since they do not target to provide performance isolation between workloads.

3.4 Desiderata of HTAP-aware Optimizations.

Referring to our discussions above, we summarize three desiderata of an HTAP-aware optimizer, which should be general to different databases' architectures and optimizers' designs.

- *Low Execution Time.* Firstly, a well-customized HTAP optimizer should leverage layout and storage-specific optimizations (e.g., hybrid physical layouts) to fully optimize query plans and the plans should be robust to the change of workloads.
- *High Data Freshness.* Second, an HTAP-aware query optimizer should not only demand the queries to reply fast but keep the data insights as fresh as possible. HTAP introduces data freshness as an additional dimension to analytical processing [63]. Hence, both data freshness and execution time should be considered when applying new query optimizations.
- *Strong Performance Isolation.* Third, a practical HTAP optimizer should keep performance isolation between OLTP and OLAP workloads even if the storage is shared with the upper engines.

4 Metis Overview

This section provides an overview of Metis, a prototype of our HTAP-aware query optimizations. As shown in Table 1, Metis responds to each desideratum of HTAP-aware optimizations by redesigning several critical design knobs of the optimizers.

4.1 Workflow and Key Components

Figure 6 illustrates the workflow of Metis. After receiving query optimization requests, Metis first performs cost estimations for enumerated plans based on the cost model and cardinality estimation, in particular, using the Demain model for access path selection (§5). Then, Metis eliminates those far-from-optimal plans and uses its visibility-aware planning algorithm to pick up a set of seed plans (§6.1). To execute, Metis starts with the plan that has the

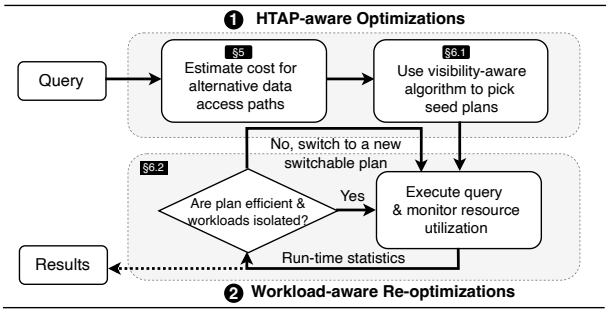


Figure 6: An overview of Metis's workflow for HTAP-aware optimizations and workload-aware re-optimizations.

lowest estimation cost and continuously monitors the query performance (e.g., by validating its statistics estimation). To ensure performance isolation, Metis monitors the resource utilization of the database. When the plan fails to fit in its performance boundary (e.g., 20% in our implementations, by default), either caused by violating performance isolation or errors in estimates for intermediate subexpressions, Metis stitches a new plan from the set of seed plans based on a more accurate run-time measurement (§6.2). Finally, the results are returned to the clients.

Different from sticking to such a *plan-first execute-next* approach, Metis interleaves plan optimization and execution. Metis prepares a set of seed plans in the first pass of query optimization, enabling the optimizer to defer the plan choice to run-time. In this way, Metis can combine the knowledge of accurate statistics and run-time resource utilization, conforming to the nature of continuously updated HTAP databases. Meanwhile, predefined seed plans may avoid losing partial work in the query execution pipeline.

Design Rationales. Metis improves the accuracy of traditional cost models by considering the mainstream architectures of HTAP databases and picks visibility-aware plans by considering the data synchronization mechanism in which replicated data becomes visible in sequence. Metis proactively ensures performance isolation between workloads by re-optimizing generated "optimal" plans.

4.2 Limitations and Discussions

Metis has two limitations. First, same as other cost-model-based optimizers, Metis relies on cardinality estimation to predicate the size of the results set, which may not be accurate for complex queries [14, 32]. In our implementations, Metis inherits cardinality estimation, transformation rules, and logical optimizations (e.g., join re-ordering algorithm) from [9, 20, 37]. However, thanks to the adoption of proactive re-optimizations, Metis can mitigate the negative effects of errors by switching to a new plan in run-time.

Second, Metis models column scans as sequential scans and assume all data are decompressed before computing. Notable data warehouses advance their OLAP performance by working directly over compressed data. However, it could be challenging when con-

sidering hybrid plans: in this case, we need to determine an assemblage point or design a new intermediate representation for data from the row store and column store, as well as model the cost. We leave these exciting explorations as our future work.

5 Demain Model

Demain provides a template of our methodology to evaluate the performance penalty of delta stores. To begin with, we first draft the storage model of our integrated system below.

Storage Model. Following the trends of new HTAP databases, our system adopts the storage-computation separation architecture and leverages the in-storage computing power to perform table scans efficiently. The row store of our system is implemented over log-structured merge trees (for short, LSM-tree). Specifically, each row in the row store is stored as a key-value pair, using its Table ID and Row ID as a key and storing all of the attributes (columns) contiguously as a value. When performing a table scan, the storage engine retrieves all key-value pairs that have the same table ID as the given table. To speed up the look-up performance on the row store, we can build both primary and secondary indices. All indices are also implemented as key-value pairs. For instance, an entry of a primary index stores its primary key as a key, and the value is the corresponding row ID of the indexed row.

Data in the column store is stored by columns and in the form of arrays (i.e., vectors or chunks of columns) during the execution. Particularly, to handle HTAP, it supports timely updates by a columnar delta tree that organizes an append-only delta store with a *B+* tree index to locate updates efficiently. When performing a column scan, the storage engine retrieves data from both the delta store and the stable column chunks and then merges the results for output.

In the rest of this section, we provide model preliminaries in §5.1, model the access paths in §5.2, and discuss the selection in §5.3.

5.1 Model Preliminaries

As shown in Table 2, Demain models access paths in HTAP from four perspectives: queries, datasets, hardware, and storage. In particular, we differentiate the bandwidth for row store and column store since they can be deployed across machines or proclaimed with an isolated quota in a single machine. In the later reference, we also differentiate the I/O bandwidth for sequential and random access with a superscript.

Without loss of generality, our cost model targets range scans, which typically filter out data according to the given predictors. We show a sample in *SQL* below:

```
SELECT col1 FROM table WHERE col1 between ${a} and ${b}
```

When answering range queries in an LSM tree database, besides the requested data, tombstones (i.e., the metadata that records invalid instances of the deleted key) and invalid entries have to be read and discarded [65]. We omit the cost of reading the tombstones since their size contributes little to a full table scan. We model read amplifications by the size ratio of the LSM tree (i.e., T in Table 2), which is a factor that captures how much the capacity of the LSM tree level i ($i \geq 1$) is greater than that of the level $i-1$.

5.2 Modeling Access Path In HTAP

Network I/O Cost. The storage-computation separation architecture incurs new overhead in retrieving data. We model the network cost by considering the transmission delay and in-network delay. For simplicity, we exclude the stack delay on the end host. Thus, the cost of forwarding data results from storage nodes to computation nodes is:

$$Cost_{net} = \frac{sel \cdot N \cdot w_{res}}{B_{net}} + L_{net} \quad (1)$$

Row Scan. Scanning data in the underlying row stores are accessed sequentially for a given table. The cost of retrieving data on the storage node includes three parts. First, it requires moving data from disks (e.g., SSD) to memory. Second, it relies on moving data from memory to the CPU cache to perform scans. Third, it consumes CPU circles to filter data according to the predictor. It should be noted that all these steps can be done in a pipeline manner. Hence, the cost should be dominated by the struggler. Thus, we have the cost for a row scan in seconds:

$$Cost_{row} = T \cdot N \cdot max\left\{\frac{ts}{DB_{row}^{seq}}, \frac{ts}{MB_{row}^{seq}}, f_p \cdot p\right\} \approx \frac{T \cdot N \cdot ts}{DB_{row}^{seq}} \quad (2)$$

Index Scan. Given an index on the accessed column, the cost of retrieving data from the index are two-fold. First, the index needs to be sequentially traversed to find a set of row IDs corresponding to the requested value range. Second, the storage engine should filter rows according to the proposed set without paying for the overhead of reading the entire key-value pairs. Here, we omit the minor cost of memory and CPUs. Without loss of generality, we have:

$$Cost_{index} = Cost_{IndexTraversal} + Cost_{DataTraversal} \quad (3)$$

$$\approx \frac{T \cdot N \cdot (w_k + w_{id})}{DB_{row}^{seq}} + \frac{T \cdot N \cdot (w_{id} + sel \cdot ts)}{DB_{row}^{rand}} \quad (4)$$

A special case is that if all needed data can be obtained from the indexed keys (a.k.a covering index), the cost of data traversal can be eliminated (e.g., the SQL in §5.1).

Query	sel	Selectivity of query q (%)
	w_{res}	Results witdh (bytes per output tuple)
Dataset	N	Data size (tuples per column)
	ts	Tuple size (bytes per tuple)
Hardware Resource	B_{net}	Network bandwidth (bytes/s)
	L_{net}	Latency of in-Network delay (s)
	DB_{row}	Disk bandwidth of read in row store (bytes/s)
	DB_{col}	Disk bandwidth of read in column store (bytes/s)
	MB_{row}	Memory bandwidth of read in row store (bytes/s)
	MB_{col}	Memory bandwidth of read in column store (bytes/s)
	p	The inverse of CPU frequency
	f_p	Factor accounting for instruction pipeline
	f_{vec}	Factor accounting for vectorized OLAP engine
Storage	T	Size ratio of the LSM tree, reference value = 10
	w_a	Attribute width (bytes)
	w_{id}	RowID width (bytes)
	b_d	*B+* tree fanout for delta store
	w_k	Key width of the index (bytes)
	N_d	Delta size (Unconsolidated tuples per column)

Table 2: Preliminaries and notations for Demain. We color all those HTAP-related preliminaries in grey.

Column Scan on Static Data. To perform a column scan on static (consolidated) column storage, we can retrieve data from the specified single column directly instead of reading the entire tuple. Thus,

$$Cost^*_{col} = N \cdot max\left\{\frac{w_a}{DB^{seq}_{col}}, \frac{w_a}{MB^{seq}_{col}}, \frac{f_p \cdot p}{f_{vec}}\right\} \approx \frac{N \cdot w_a}{DB^{seq}_{col}} \quad (5)$$

Delta Scan. We now explore the cost model for range scans on the delta store. As an optimization, our system has implemented *B+* trees over its append-only delta stores (see storage model in §5). We model the cost of retrieving data from the *B+* tree in two parts. The first part is for traversing the internal structure of the *B+* tree to find the starting point in the first leaf node corresponding to the requested value range. The second part is for traversing leaf nodes to read the indexed row IDs and find the tuples in the delta store according to the row IDs.

Particularly, in the delta store, a tuples ID in the leaf node of the *B+* tree can be either matched to a *delete* or an *insert* operation. For those *delete* operations, a row ID does not essentially cost a read in the delta store. We can simply merge those *delete* operations into the results of the column scan by ignoring the corresponding rows. In practice, we use lightweight statistics in the delta stores to count the number (ratio) of *insert* operations; the others are *delete* operations. We assume each tuple in the delta store matches a full tuple update, i.e., the size of the indexed tuple equals its tuple size in row format. We also assume a uniform distribution for data access and updates. Hence, the worst case for the cost of a delta scan becomes:

$$Cost_{delta} = Cost_{TreeTraversal} + Cost_{DataTraversal} \quad (6)$$

$$Cost_{TreeTraversal} = (1 + \lceil log_b(N_d) \rceil) \cdot \frac{b}{2} \cdot (f_p \cdot p + \frac{1}{DB^{rand}_{col}}) \quad (7)$$

$$Cost_{DataTraversal} = N_d \cdot \frac{(w_{id} + sel \cdot ts)}{DB^{rand}_{col}} \quad (8)$$

Column Scan in HTAP. Putting Equation 6 and Equation 5 together, we have the overall cost of a column scan in HTAP:

$$Cost_{col} = Cost^*_{col} + Cost_{delta} \quad (9)$$

The total cost of retrieving data should also combine with the cost of network I/O for transmission data from storage nodes to computation nodes (see Equation 1), which can be critical for estimating the latency or further optimizing the physical plans (e.g., reducing network I/O cost by performing additional in-storage computation). However, for a simple access path selection task that is discussed above, the cost is not critical since the storage engine filters data according to the predictor for each access path (i.e., row scan, index scan, and column scan) locally and thus the size of the transmitted results set should be identical.

5.3 Access Path Selection

Using the equations in §5.2, we first detail the comparison between row scans, column scans on static data, and index scans. From Equation 2 and Equation 5, it's evident that, for a disk-based database, the major advancement of column stores is to help reduce I/O cost (i.e., slim down the cost from $T \cdot ts$ to w_a for each tuple, where ts is always $\geq w_a$ and T is always ≥ 1). This benefit is pronounced especially when the table has massive columns, and the number of requested columns is few. It becomes a bit tricky when an index exists on the conditional columns. The read performance of the row store can be enhanced since the index can help skip unnecessary tuple access but only read the entire tuples corresponding to the requested value range. Given this, traditional optimizers decide the threshold of switching access path depending on the query selectivity (i.e., *sel* in Equation 3). When the query has a lower selectivity, the overall cost of the index scan should be lower. It should also be noted that index scans may have poorer performance than row scans when the *sel* is particularly large since they pay additional overhead on sequential index traversal and do data traversal randomly.

We then discuss the access path selection in Metis. Metis refines the cost of column scan in Equation 9. Hence, the row store has more potential to outperform the column store due to the performance penalty of data synchronization on the delta scan (Equation 6). According to our model, Metis prefers a row scan when the *sel* is particularly large, and the performance penalty of N_d outpaces the benefit of slimming I/O cost $T \cdot ts$ to w_a. When *sel* is particularly small, Metis prefers an index scan. Otherwise, Metis chooses a column scan.

6 Runtime Optimizations

In this section, we propose two optimizations to make plans generated by Metis visibility-aware and provide performance isolation property to the database in the face of resource contention.

6.1 Visibility-aware Plan Optimization

Given the revised cost model, when performing plan enumeration, Metis represents each physical plan in a directed acyclic graph (DAG), which constructs a partial order for plan execution. Metis leverages DAGs to predicate plan performance. Each vertex in the graph corresponds to a physical operator. Each edge represents a dependency between two operators due to data dependency. Specifically, there exist two types of edges: if an operator (O_i) must wait for the whole data output of another operator (O_j) before execution, we term such a dependency as a hard dependency ($O_i \rightarrow O_j$); otherwise, if O_i can be executed in a pipeline manner, we term it as a soft dependency ($O_i \rightsquigarrow O_j$).

For example, if a hybrid plan ***P1*** retrieves data from table A in the row store using an index scan (O_{A_index}), retrieves data from table B in the column store using a column scan (O_{B_column}), and joins table A and table B using a pipelined hash join [36] by assuming table A as a build table (O_{A_build}) and table B as a probe table (O_{B_probe}), then there exist three edges between four operators: $O_{A\_index} \rightsquigarrow O_{A\_build}$, $O_{B\_column} \rightsquigarrow O_{B\_probe}$, and $O_{A\_build} \rightarrow O_{B\_probe}$.

Recall the critical insight into the visibility-aware plan selection: scheduling pre-execution on the available data ahead instead of blocking queries until all data becomes visible can help mask the visibility delay of HTAP databases. Therefore, in ***P1***, Metis can retrieve data from the row store and perform the first phase (i.e., building a hash table) of the hash join ahead of retrieving data from the column store, reducing the overall response latency. Compared to an alternative physical plan ***P2*** that retrieves all data from the column store and then performs hash join, ***P1*** may outperform ***P2*** in query latency even if the execution time of the index scan is a bit longer than the column scan since ***P2*** has to be blocked until

Algo 1: Freshness-efficient Plan Selection (§6.1).

```
1  Para: G_p ← The DAG representation of a hybrid plan p
2  Para: M ← The cost model that estimations cost in seconds
3  Para: α ← The visibility delay in the HTAP database
4  Function planSelection(DAGs, α) do
5      bestPlan ← ∅;  Cost_bset ← 0;
6      for each G_p in DAGs do
7          Cost_p ← calculateCost(G_p, α);
8          if bestPlan == ∅ ∨ Cost_p < Cost_best :
9              bestPlan = p;  Cost_best ← Cost_p;
10     return bestPlan;
11 Function calculateCost(G_p, α) do
12     preCost ← 0;  exeCost ← estimateCost (G_p, M) ;
13     invalidSet ← findAllPendingTasks(G_p);
14     subs ← removePendingTasks(G_p, invalidSet);
       /* removePendingTasks returns the strongly connected
          components after removing pending tasks in G_p. */
15     foreach sub ∈ subs do
16         Cost_sub ← estimateCost (sub, M);
17         if preCost < Cost_sub : preCost ← Cost_sub ;
18     return Cost ← exeCost + α - min {preCost, α};
```

the new data of table A is integrated into the column store.

Another comprehensive example is shown in Figure 7, where the parts of plans in the green box are ready to be executed.

Algorithm 1 shows the pseudocode for our visibility-aware plan selection. Based on the DAG abstraction, we first calculate pre-execution tasks for each plan by removing unavailable pending tasks in the graph (Lines 13-14). By doing so, we get a set of pre-execution tasks (i.e., *subs*). Each task in the set is a strongly connected component that contains multiple physical operators. Since there is no data dependency between the tasks in *subs*, they can be executed in parallel. Therefore, the overall performance improvement of scheduling pre-execution should be the execution time of the longest task. We combine the knowledge of visibility delay into the plan cost at Line 18, which captures the end-to-end query latency observed by users. Finally, we use the revised cost to guide the selection function (Lines 4-10) and generate a visibility-aware physical plan that has the cheapest cost on query latency.

Discussions. For data consistency, Metis guarantees snapshot isolation and always uses the latest timestamp to execute queries, guaranteeing the best data freshness. In Figure 7, we assume all data in the column store is not visible at the plan-generated time and becomes visible atomically when all new data is synchronized into the column store. This simplified abstraction matches the behavior that the storage engine retrieves data with a global timestamp ts_q, and blocks all reads until the timestamp of the column store $ts_{col} \geq ts_q$. In practice, several optimizations are proposed to optimize such centralized timestamp management by using a subdivided timestamp for each table or partition. For example, our database system manages individual timestamps for different data chunks, where updates to a data chunk are sequenced individually. Using fine-grained timestamps, a portion of data can be available in the

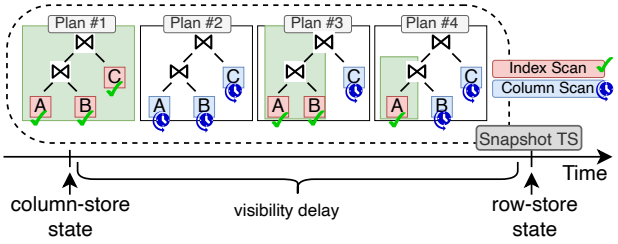


Figure 7: An example of visibility-aware plan selection. The parts of the plans in the green box are ready to be executed.

column store at the plan-generated time, and the visibility of the column store increments gradually. We follow such optimizations from the integrated databases in our implementations.

Additionally, incremental computation techniques can be combined into the pre-execution on the column store to improve the performance further. We leave these developments as our future work since it is orthogonal to our proposal. Given incremental computation, our visibility-aware algorithm is still applicable.

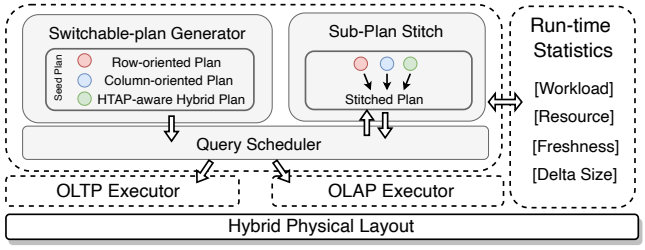
6.2 Proactive Query Re-optimizations

To keep plans efficient and workloads isolated, Metis re-optimizes plans proactively. Figure 8a shows an overview of our approach. Intuitively, when performing query optimizations, Metis generates a set of seed plans (i.e., a row-oriented plan, a column-oriented plan, and an HTAP-aware plan). When executing queries, Metis starts with the cheapest plan and continuously re-optimizes plans by stitching sub-plans from other seed plans.

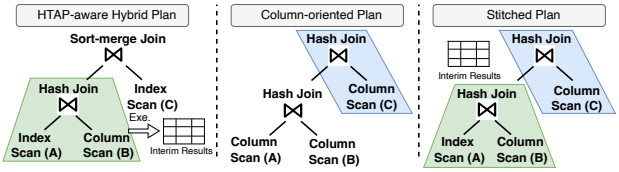
Seed Plans. To generate seed plans, Metis uses the revised cost model (§5) and visibility-aware plan selection algorithm (§6.1). Metis first generates a hybrid plan with the cheapest cost. Based on the same logical structure of the hybrid plan, Metis generates an optimized row-oriented plan and an optimized column-oriented plan by considering different physical operators. Therefore, all seed plans have the same logical structure but may adopt different physical operators (e.g., row scan versus column scan, nested loop join versus hash join, and stream aggregation versus hash aggregation). Generally, seed plans consist of three physical plans when there exists a hybrid plan that outperforms the row-oriented and column-oriented plans; otherwise, seed plans consist of two plans.

We do not individually optimize the logical structure for the row-oriented and column-oriented plans since the cardinality estimations that can influence the optimality of the logical plan (e.g., to decide an optimal join order) are exactly the same. For re-optimizations on the logical structures at runtime (e.g., runtime join reorder), several notable works [7, 11, 52] have been proposed, which are orthogonal to our paper and can be adopted by Metis.

Runtime Statistics. To decide whether a query plan should be re-optimized, Metis quantitatively evaluates the effects of its decisions based on runtime statistics. For efficient execution, Metis re-optimizes physical operators when the estimated cardinality is far from the real statistics (e.g., beyond 20%). Multiple existing works also adopt a similar strategy [12, 30, 32, 75]. For the performance isolation between workloads, Metis stitches sub-plans to alleviate resource contention. We use a threshold of the observed physical resource utilization to control the use of the physical resource (e.g., 80% CPU utilization) and use a threshold of contention footprint



(a) An overview of proactive plan re-optimization in Metis.



(b) An example of combining sub-plans to keep the query execution efficient and ensure the performance isolation between workloads.

Figure 8: Proactive plan re-optimization and sub-plan stitch.

(i.e., the number of conflict transactions) to control the logical contention on each table.

Plan Stitch. Inspired by [30], when re-optimizing a plan, Metis stitches sub-plans from the seed plans without losing partial work in the query execution pipeline. As the seed plans in Metis have the same logical structure, stitching a new plan is essentially re-optimizing the selection of physical operators for the remaining un-executed plans. For future extensions, our framework is general to different re-optimization techniques. Besides runtime re-optimizations on logical plans mentioned previously, one may also consider combining efficient previously-executed physical plans into the seed plans.

Example. We now show an example of query re-optimizations. The example query in Figure 8b joins Tables A, B, and C. Metis starts the execution with the cheapest hybrid plan and generates the interim results by joining Table A and B. Due to inaccurate cardinality or resource contention, re-optimization is trigged when retrieving data from Table C. In this case, Metis stitches a new plan with the column-oriented plan in the set of seed plans. The stitched plan uses a column scan for Table C instead of an index scan for better performance. Consequently, Metis uses a hash join instead of a sort-merge join because the cost will be higher when the column store can not provide a sort order of the input data. Given this stitched plan, Metis then reuses the interim results and continues the execution.

7 EXPERIMENTS

In this section, we extensively study the performance of Metis and compare the performance of HTAP-aware plans with row-oriented, column-oriented, and HTAP-agnostic plans in various aspects.

Our evaluation focused on the following questions:

- §7.2 Can Metis benefit the performance of analytical queries?
- §7.3 How does Metis's visibility-aware plan selection algorithm help with reducing response latency?
- §7.4 Can Metis retain performance isolation between workloads when executing hybrid plans?
- §7.5 How does Metis adapt to the shifting workloads?

7.1 Evaluation Setups

Hardware Configurations. We ran all experiments on a cluster with seven machines. Each machine has a 2.60GHz Intel(R) Xeon(R) E5-2690 v3 CPU (i.e., 24 cores with a single NUMA node), 64GB memory with 544Gbps bandwidth, 960GB Dell DD4G0 SSD with 6Gbps bandwidth, and a 40Gbps NetXtreme NIC.

System Deployments. We compiled Metis on Ubuntu 18.04. We ran three row stores and three column stores. Each of them is deployed on individual machines. Data is loaded and continuously written into the row store and asynchronously replicated into the column store. We set up a client program along with a computation node on a single machine to send client requests and perform off-storage computation. We emulated 3*ms* in-network delays among machines using Linux tc [38], which is in line with the network latency inside a data center [1, 2].

Workloads. To emulate diverse application scenarios and analyze the performance of Metis, we used three well-studied workloads.

CH-benCHmark. We used CH-benCHmark, a notable workload in HTAP scenarios, to evaluate the performance of Metis under hybrid workloads. It integrates an OLAP workload (i.e., TPC-H [25]) into an OLTP workload (i.e., TPC-C [26]) with a unified data schema. It contains five types of transactions and twenty-two types of analytical queries. Similar to TPC-C, CH-benCHmark organizes data in warehouses. We used 100 data warehouses in our experiments.

TPC-DS. We adopted TPC-DS [24] with $sf = 100$ for analyzing hybrid plans in §2.2. Overall, TPC-DS contains much more complex analytical queries than CH-benCHmark. For instance, TPC-DS has 24 tables and 7.9 join operators per query, while CH-benCHmarks has 12 tables and 2.9 join operators per query on average. As TPC-DS is a pure OLAP workload without any transactions or data modifications, we did not use it in evaluating HTAP-aware plans.

YCSB. YCSB is a performance benchmark suite. To provide a more in-depth analysis of Metis under hybrid workloads, we developed a micro-benchmark using the APIs of the YCSB [23]. The micro-benchmark includes two tables (A and B). Each table has 100 million rows with a 64-bit primary key attribute and ten data attributes. Primary indices are built on each table. In addition to the transactions that perform ten reads or writes on these attributes, we create an analytical query (**Q1**) that joins the two tables and controls the amount of data accessed by each table with range predictors. We also make a simple select query (**Q2**) that queries Table A with a range predictor to provide a micro analysis on access path selection.

Baselines. The primary competitor of Metis is using HTAP-agnostic plans. Specifically, the HTAP-agnostic approach reused the state-of-the-art cost model of the row store and simply added column scans as an alternative data access path without considering any HTAP contexts (§3). The approach is used by multiple existing HTAP databases [37, 46]. In our experiments, we generated the HTAP-agnostic plans using the same codebase of Metis while ignoring the new HTAP-aware optimizations (§5 and §6).

We also study the performance of two additional baselines: row-oriented and column-oriented plans, where the space of query optimization is restricted to row stores or column stores, respectively.

7.2 Overall Performance

7.2.1 Benefits of hybrid plans. We first evaluate the performance of Metis under the default settings of the CH-benCHmark workload.

	Row-oriented	Col.-oriented	HTAP-agnostic	HTAP-aware
Q1	69.06s	7.37s	7.37s ($\mathcal{C}$)	7.37s ($\mathcal{C}$)
Q2	26.98s	2.42s	2.42s ($\mathcal{C}$)	2.42s ($\mathcal{C}$)
Q3	48.71s	4.52s	48.71s ($\mathcal{R}$)	4.52s ($\mathcal{C}$)
Q4	5.68s	7.02s	3.51s ($\mathcal{H}$)	3.07s ($\mathcal{H}$)
Q5	10.27s	11.32s	7.78s ($\mathcal{H}$)	7.10s ($\mathcal{H}$)
Q6	9.33s	1.00s	1.00s ($\mathcal{C}$)	1.00s ($\mathcal{C}$)
Q7	29.55s	4.59s	2.89s ($\mathcal{H}$)	2.53s ($\mathcal{H}$)
Q8	46.18s	8.42s	8.42s ($\mathcal{C}$)	8.42s ($\mathcal{C}$)
Q9	158.92s	28.97s	183.81s ($\mathcal{H}$)	28.97s ($\mathcal{C}$)
Q10	3.70s	4.68s	4.99s ($\mathcal{H}$)	2.83s ($\mathcal{H}$)
Q11	14.77s	2.78s	2.78s ($\mathcal{C}$)	2.78s ($\mathcal{C}$)
Q12	26.27s	2.56s	42.19s ($\mathcal{H}$)	2.56s ($\mathcal{C}$)
Q13	60.74s	5.73s	5.73s ($\mathcal{C}$)	5.73s ($\mathcal{C}$)
Q14	12.73s	1.40s	1.40s ($\mathcal{C}$)	1.40s ($\mathcal{C}$)
Q15	150.10s	2.63s	2.63s ($\mathcal{C}$)	2.63s ($\mathcal{C}$)
Q16	27.05s	1.25s	1.25s ($\mathcal{C}$)	1.25s ($\mathcal{C}$)
Q17	92.02s	12.02s	12.02s ($\mathcal{C}$)	12.02s ($\mathcal{C}$)
Q18	8.67s	14.11s	8.67s ($\mathcal{R}$)	8.67s ($\mathcal{R}$)
Q19	35.55s	2.82s	28.38s ($\mathcal{H}$)	2.82s ($\mathcal{C}$)
Q20	41.69s	6.38s	6.38s ($\mathcal{C}$)	6.38s ($\mathcal{C}$)
Q21	OOM	18.93s	9.42s ($\mathcal{H}$)	8.97s ($\mathcal{H}$)
Q22	3.57s	1.21s	9.69s ($\mathcal{H}$)	0.87s ($\mathcal{H}$)
G-Mean	24.87s	4.53s	6.91s	3.86s

Table 3: A comparison of the row-oriented, column-oriented, HTAP-agnostic, and HTAP-aware query optimization approaches under CH-benCHmark. We mark the generated row plans with $\mathcal{R}$, column plans with $\mathcal{C}$, and hybrid plans with $\mathcal{H}$ in the last two columns. For the HTAP-agnostic approach, we highlight the negative optimization cases in grey. For the HTAP-aware approach, we highlight the hybrid plans with "optimal" latecny in the hatched cells.

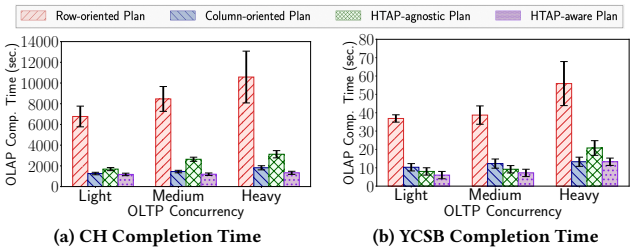
For OLTP, we ran 256 threads to saturate the throughput, achieving 3082.64 *tps*. For OLAP, we executed analytical queries in sequence. Each query is optimized and then executed one by one using multi-cores. By default, we used 24 threads for intra-query parallelism.

Table 3 shows the results of analytical queries' response latency. The column-oriented plans outperformed the row-oriented plans in most of the queries. Exceptions are $Q4$, $Q5$, $Q10$, and $Q18$. For HTAP-agnostic and HTAP-aware plan optimization approaches, we marked the generated row plans with $\mathcal{R}$, column plans with $\mathcal{C}$, and hybrid plans with $\mathcal{H}$.

As shown in Table 3, Metis (using HTAP-aware plans) always performed better than the competitors for all analytical queries. Compared to the HTAP-agnostic approach, Metis's closest competitor, Metis achieved 1.79× speedups in geometric mean.

As illustrated in §3, HTAP-agnostic plans can lead to poor performance due to error-prone cost estimation. Hence, the response latency of the HTAP-agnostic plans was even 1.52× than the column-oriented plans, let alone realizing the potential of hybrid plans. We highlight the sub-optimal cases in grey. Similar negative effects were validated in [33]. We also observed that the performance of the hybrid plans generated by the HTAP-agnostic approach might be even poorer than both row-oriented and column-oriented plans on specific queries (e.g., $Q9$, $Q10$, $Q12$, and $Q22$).

Given this, one may think about building an HTAP optimizer without hybrid plans but select the row-oriented and column-oriented plans for each query, achieving the best of the two. However, it should only be feasible with a unified cost model that can precisely predicate the cost of both two plans. When creating a unified cost model, the problems inside the HTAP-agnostic approach still exist, i.e., such an optimizer can always choose sub-optimal plans due to the error-prone cost estimation.



(a) CH Completion Time **(b) YCSB Completion Time**

Figure 9: Analytical Queries Completion Time (ten rounds).

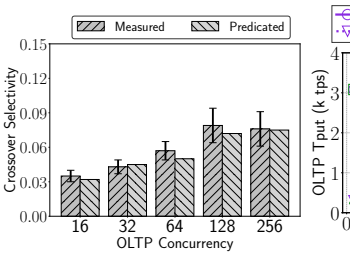


Figure 10: Accuracy of Demain.

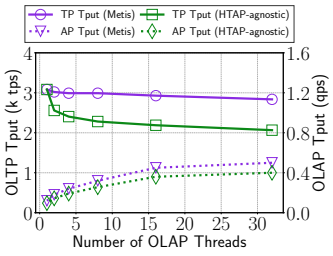


Figure 11: Impact on OLTP.

Metis corrects sub-optimal plans in two-fold. First, the Demain model provides a revision for access path selection. For example, in $Q9$, the HTAP-agnostic plan retrieves data from the row store for the table `order` and the rest of the data from the column store. On the contrary, according to our new cost model, Metis additionally retrieves data from the row store for the table `nation` and `supplier`. Second, Metis's runtime re-optimization alleviates the performance degradation caused by resource contention, especially when accessing data from the row store. For instance, the `new order` transactions that updated `d_next_o_id` frequently in CH-Benchmark can cause a big footprint in the corresponding data chunk. In such a case, Metis prefers using the column store for retrieving data from the corresponding table `district`.

Furthermore, Metis enhanced the query performance with its visibility-aware optimization algorithm (§6.1). Benefited queries are $Q4$, $Q5$, $Q10$, $Q21$, and $Q22$. We provide an ablation study in §7.3.

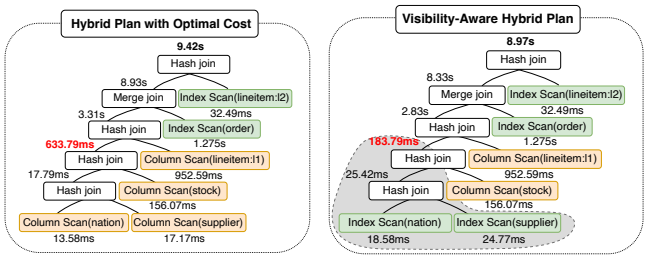
Takeaways. Neither the row store nor the column store can be superior for retrieving data. Metis can achieve the best potential of hybrid plans under HTAP workloads.

7.2.2 Performance under different OLTP concurrency. As the OLTP concurrency in HTAP workloads may change over time, we next study how Metis performed under various OLTP concurrencies. The adaptivity of Metis was studied in §7.5.

We created three distinct scenarios for both CH-benCHmark and YCSB workloads, where the OLTP concurrency is light (~20% peak throughput), medium (~50% peak throughput), and high (~80% peak throughput). Specifically, we used 32 OLTP threads for light concurrency, achieving 732.54 *tps* for CH-benCHmark and 8180.42 *tps* for YCSB. We used 128 OLTP threads for medium concurrency, achiev-

	Q4	Q5	Q7	Q10	Q21	Q22
Visibility-Agnostic	3.51s	7.78s	2.89s	3.57s	9.42s	1.07s
Visibility-Aware	3.07s	7.10s	2.53s	2.83s	8.97s	0.87s

Table 4: Impact of visibility-aware optimizations.



(a) Hybrid plan with optimal cost. **(b) Visibility-aware hybrid plan**

Figure 12: An example of the impact of pre-execution.

ing 1930.98 *tps* for CH-benCHmark and 23108.06 *tps* for YCSB. We used 256 OLTP threads for high concurrency, achieving 3082.64 *tps* for CH-benCHmark and 41556.60 *tps* for YCSB.

We reported the completion time of analytical queries in ten rounds. For CH-benCHmark, OLAP clients issued 220 TPC-H-like queries iteratively. For YCSB, OLAP clients issued 10 $Q1$ with 0.01% selectivity in Table A and 1% selectivity in Table B.

Figure 9 shows the experimental results with error bars.

CH-benCHmark. The results demonstrated that OLTP concurrency could affect the performance of all types of plans. This is because, as the OLTP concurrency increased, more transactions could consume more physical and logical resources in the row store and cause more data synchronization operations in the column store. Compared to HTAP-agnostic plans, the performance degradation of Metis (using HTAP-aware plans) was relatively small.

YCSB. For YCSB, we observed that, under light and medium OLTP concurrency, both the HTAP-agnostic and HTAP-aware plans could be optimal. We checked the physical plan by the `ANALYZE` statements in *SQL*. The plan retrieved data from Table A in the row store and from Table B in the column store. Then, the plan joined Table A with Table B using Hash Join. Metis performed slightly better thanks to the pre-execution scheduling.

Under high OLTP concurrency, Metis shifted to the column plan to alleviate resource contention on the row store. At the same time, the HTAP-agnostic approach still used the hybrid plan, leading to sub-optimal performance (i.e., 1.56× latency in our evaluation).

7.2.3 Accuracy of Demain. We analyzed the accuracy of the Demian model by comparing the predicted crossover selectivity against the measured crossover selectivity for access path selection. Recall the definition of crossover in §3.1. The crossover is the selectivity when the row store and column have an identical execution time. We used YCSB $Q2$ and fine-tuned the selectivity of the predictor to find the measured crossover. The predicated crossover was calculated by solving the equation when the index scan has the exact cost as the column scan in §5. As shown in Figure 10, Metis accurately predicted the crossover point under different OLTP concurrency. The error rate was up to 12.28% and within the standard deviation of the measured crossover selectivity.

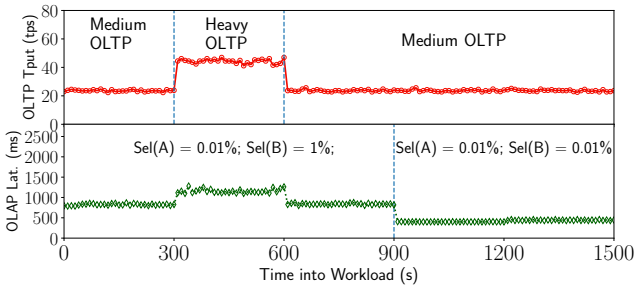


Figure 13: Impact of Shifted Workload.

7.3 Impact of Visibility-Aware Optimizations

Ablation Study. Table 4 shows an ablation study by independently removing our visibility-aware optimization algorithm in Metis. For clarity, we only present the queries that can potentially benefit from pre-execution in CH-Benchmark.

Case Study of Ablation. Figure 12 shows the physical plan of CH-benCHmark $Q21$ with the accumulated execution time for each operator. In particular, Figure 12a shows the hybrid plan with "*optimal*" cost, which was generated by disabling our visibility-aware algorithm. Figure 12b reveals the visibility-aware hybrid plans.

To retrieve data from table `nation` and `supplier`, column scan can outperform row scan slightly (i.e., 13.58*ms* versus 18.58*ms* and 17.17*ms* versus 24.77*ms*). However, due to the visibility delay (i.e., ~800*ms* measured in our experiment), data in the column store was unavailable until all new data was synchronized from the row store. Therefore, the plan with the "*optimal*" cost had to be blocked and not scheduled until new data became visible in the column store. On the contrary, our visibility-aware plans could be scheduled ahead of the full data synchronization as new data was available in the row store. By doing so, a portion of the plan (i.e., the grey part in Figure 12b) could be executed ahead of time to mask the visibility delay. Then, the amortized cost (183.79*ms*) of the sub-plan was cheaper than the sub-plan of the "*optimal*" plan (633.79*ms*).

As a result, though the visibility-aware plan could pay more cost to retrieve data from table `nation` and `supplier`, its overall response latency was shorter than the plan with the "*optimal*" cost (i.e., 8.97*s* compared to 9.42*s*).

7.4 Performance Isolation of Workloads.

As discussed in §3.3, hybrid plans introduce new challenges in performance isolation, and the HTAP-agnostic approach may largely degrade OLTP performance. We study that impact in this experiment using Metis under the CH-benCHmark workload. We first set up 256 OLTP threads to saturate the OLTP throughput and then increased the OLAP workloads by adding OLAP threads (clients).

As shown in Figure 11, for Metis, the OLAP throughput increased with the number of OLAP threads and eventually plateaued; the OLTP throughput was basically preserved throughout the experiment. On the contrary, the HTAP-agnostic approach incurred a much more server OLTP throughout degradation compared to Metis, in line with the evaluation results in §3.3. This is because Metis will proactively re-optimize in-efficient plans in the face of resource contention. As evidence, as shown in Table 3, when using 256 OLTP threads, the HTAP-agnostic approach totally generated

nine hybrid plans for analytical queries in CH-benCHmark, which should be optimal when given static data. In contrast, the number of hybrid plans in Metis was six.

Meanwhile, we observed that the throughput of OLTP indeed incurred a slight degradation (up to 8%). We conclude this degradation for two reasons: (1). OLTP and OLAP requests shared the same front end of the database (e.g., sessions and *SQL* parser) and (2). a part of internal service inside the database (e.g., *timestamp* allocation).

7.5 Impact of Changed Workloads

So far, we have studied the performance variance of Metis under different OLTP concurrency in §7.2.2, the accuracy of access path selection in §7.2.3, and the efficiency of visibility-aware plan selection in §7.3. In this experiment, we study the adaptive capabilities of Metis by examining its OLTP throughput and OLAP latency over time.

Figure 13 shows the OLTP throughput and OLAP (i.e., YCSB $Q1$) latency in YCSB. Recall in §7.2.2, $Q1$ prefers a hybrid plan that retrieves data from Table A in the row store and Table B in the column store when the contention between workloads is low. As we shifted the number of OLTP threads from 128 to 256 at 300*s*, the OLTP increased correspondingly, which consumes $\sim$ 92% of CPU circles on the row store. In such a case, Metis avoided retrieving data from the row store favor for both query efficiency and performance isolation between workloads. Metis re-optimized the hybrid plan by its proactive re-optimization technique (§6.2) and thus retrieved data from Tables A and B in the column store, in line with the results in §7.2.

At 900*s*, we changed the selectivity on Table B. Metis generated a row-oriented plan that retrieved all data in the row store and joined the two tables using Sort-merge Join instead of Hash Join.

8 RELATED WORK

Query optimization is a well-studied research area. Massive influential works have been proposed since the 1970s. In this paper, we rethink optimization techniques in the era of HTAP by standing on the shoulders of giants. We summarize related works below.

8.1 Query Optimization in HTAP Databases

Independent optimizers. Several HTAP databases [15, 18, 57] process query optimizations for OLTP and OLAP independently, using a routing-based approach, and are not designed for hybrid data access. After receiving *SQL* requests, they rely on embedded user-level hints or a smart middleware layer to differentiate point and scan-intensive queries. Then, they execute queries on the desirable engines and stores. This approach is easy to implement; however, at the cost of performance since optimizations are conducted in isolation, and the prior knowledge of queries can always be limited and inaccurate.

Integrated optimizer. Another approach adopted by multiple HATP databases [6, 21, 27, 35, 37, 39, 46, 47, 55, 61, 68, 73] is supplementing column stores as an additional data access path. For instance, F1 Lightning [73] generates logical plans using its F1 optimizer (i.e., an optimizer designed for OLTP) and considers lightning-only indexes and views during physical planning. SQL Server [33] analyzes and recommends column stores by its Database Engine Tuning Advisor (DTA) when suitable for a given workload. TiDB [37] extends query optimizer to explore physical plans accessing both row and column stores. Oracle Dual [46] supplements column indices to its optimizer as an alternate execution method for high-speed table scans. Compared to Metis, all of them either do not support *hybrid plans* or generate *HTAP-agnostic plans*.

8.2 Access Path Selection in Modern Databases

Access path selection is one of the most fundamental optimizations in databases for retrieving tuples from tables. Besides the important works [17, 56, 66] proposed at the beginning of the database system, recent studies focus on the analysis of access path selection over large-scale column databases, in-memory row-oriented databases, and hybrid physical designs in HTAP databases.

Kester et al. [45] analyze the problem of access path selection for in-memory analytical databases by comparing probes on *B+* trees to shared scans on column stores. Dziedzic et al. [33] present the analysis of access path selection for a commercial-strength database, considering secondary *B+* trees on top of column-store indexes. Abadi et al. [5] provide an experimental study to quantify the significant differences between column store and row-oriented *B+* trees. Unlike existing works, our paper discusses the access path selection over the specially-tailored HTAP databases (i.e., using delta-main architectures) and models row stores as LSM trees.

8.3 Proactive Query Re-optimizations

Several previous works [12, 30, 32, 40, 42, 53, 72, 75] inspire Metis's resource-aware query re-optimization.

Proactive Re-optimizations. Particularly, Babu et al. [12] take the first step to re-optimize plans proactively. They estimate statistics computed as bounding boxes and generate a switchable seed plan for runtime re-optimization. Ding et al. [30] harness valuable information of efficient sub-plans collected from other previously-executed plans and stitch these sub-plans at runtime.

Resource-aware Query Plan. Viswanathan et al. [72] integrate resource planning within a query planner using a cost-based model in Hive and Spark. Li et al. [53] propose a resource-aware deep-learning model that can predict the execution time of plans and thus combine the knowledge of available resources into query planning.

While sharing the same goal to re-optimize query plans for efficiency, Metis is additionally designed to keep performance isolation between workloads. To do so, Metis prepares three seed plans for different data access paths and continuously monitors resource utilization (§6.2).

9 CONCLUSION

In this paper, we demonstrate that *hybrid plans* are desirable for HTAP databases since all row scans, index scans, and column scans can benefit analytical queries. We systematically analyze the challenges and desiderata of hybrid plans. As HTAP databases evolve specially-tailored architectures and target specific design goals, existing works either generate sub-optimal plans or sacrifice HTAP properties, prompting a revisit of query optimization techniques.

Based on our analysis, we propose Metis, an HTAP-aware hybrid optimizer, which uses a revised cost model for access path selection, selects plans with visibility-aware optimization and re-optimizes queries proactively to keep plans efficient and isolate the performance between workloads. Our experiments on extensive workloads show the efficiency and adaptivity of Metis.

References

[1] AWS Latency Monitoring. https://www.cloudping.co/grid.

[2] Regions, Availability Zones, and Local Zones. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[3] Daniel Abadi, Samuel Madden, and Miguel Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682, 2006.

[4] Daniel J Abadi, Peter A Boncz, and Stavros Harizopoulos. Column-oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665, 2009.

[5] Daniel J Abadi, Samuel R Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980, 2008.

[6] Michael Abebe, Horatiu Lazu, and Khuzaima Daudjee. Proteus: Autonomous adaptive storage for mixed workloads. Technical report, Technical Report. University of Waterloo. https://cs. uwaterloo. ca …, 2022.

[7] Sameer Agarwal, Srikanth Kandula, Nicolas Bruno, Ming-Chuan Wu, Ion Stoica, and Jingren Zhou. Reoptimizing data parallel computing. In *NSDI*, volume 12, pages 281–294, 2012.

[8] Nitin Agrawal and Ashish Vulimiri. Low-latency analytics on colossal data streams with summarystore. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 647–664, 2017.

[9] Rafi Ahmed, Allison Lee, Andrew Witkowski, Dinesh Das, Hong Su, Mohamed Zait, and Thierry Cruanes. Cost-based query transformation in oracle. In *VLDB*, volume 6, pages 1026–1036, 2006.

[10] Vaibhav Arora, Faisal Nawab, Divyakant Agrawal, and Amr El Abbadi. Janus: A hybrid scalable multi-representation cloud datastore. *IEEE Transactions on Knowledge and Data Engineering*, 30(4):689–702, 2017.

[11] Ron Avnur and Joseph M Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 261–272, 2000.

[12] Shivnath Babu, Pedro Bizarro, and David DeWitt. Proactive re-optimization. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 107–118, 2005.

[13] Peter A Boncz, Marcin Zukowski, and Niels Nes. Monetdb/x100: Hyper-pipelining query execution. In *Cidr*, volume 5, pages 225–237. Citeseer, 2005.

[14] Renata Borovica-Gajic, Stratos Idreos, Anastasia Ailamaki, Marcin Zukowski, and Campbell Fraser. Smooth scan: Statistics-oblivious access paths. In *2015 IEEE 31st International Conference on Data Engineering*, pages 315–326. IEEE, 2015.

[15] Dennis Butterstein, Daniel Martin, Knut Stolze, Felix Beier, Jia Zhong, and Lingyun Wang. Replication at the speed of change: A fast, scalable replication solution for near real-time htap processing. *Proc. VLDB Endow.*, 13(12):3245–3257, aug 2020.

[16] Shaosheng Cao, XinXing Yang, Cen Chen, Jun Zhou, Xiaolong Li, and Yuan Qi. Titant: Online real-time transaction fraud detection in ant financial. *Proc. VLDB Endow.*, 12(12):2082–2093, aug 2019.

[17] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43, 1998.

[18] Jianjun Chen, Yonghua Ding, Ye Liu, Fangshi Li, Li Zhang, Mingyi Zhang, Kui Wei, Lixun Cao, Dan Zou, Yang Liu, et al. Bytehtap: bytedance's htap system with high data freshness and strong data consistency. *Proceedings of the VLDB Endowment*, 15(12):3411–3424, 2022.

[19] M Keith Chen and Michael Sheldon. Dynamic pricing in a labor market: Surge pricing and flexible work on the uber platform. *Ec*, 16:455, 2016.

[20] Inc. ClickHouse. ClickHouse — open source distributed column-oriented DBMS. https://github.com/ClickHouse/ClickHouse/tree/22.6.

[21] Alibaba Cloud. PolarDB: Cloud-Native Relation Database. https://www.alibabacloud.com/product/polardb.

[22] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, et al. The mixed workload ch-benchmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, pages 1–6, 2011.

[23] Brian Cooper. Yahoo! cloud serving benchmark. https://github.com/brianfrankcooper/YCSB.

[24] The Transaction Processing Council. TPC-DS. http://www.tpc.org/tpcds/.

[25] The Transaction Processing Council. TPC-H. http://www.tpc.org/tpch/.

[26] The Transaction Processing Council. TPC-C. http://www.tpc.org/tpcc/, 2014.

[27] Umur Cubukcu, Ozgun Erdogan, Sumedh Pathak, Sudhakar Sannakkayala, and Marco Slot. Citus: Distributed postgresql for data-intensive applications. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2490–2502, 2021.

[28] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, pages 215–226, 2016.

[29] Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Röhm. Ycsb+ t: Benchmarking web-scale transactional databases. In *2014 IEEE 30th International Conference on Data Engineering Workshops*, pages 223–230. IEEE, 2014.

[30] Bailu Ding, Sudipto Das, Wentao Wu, Surajit Chaudhuri, and Vivek Narasayya. Plan stitch: harnessing the best of many plans. *Proceedings of the VLDB Endowment*, 11(10):1123–1136, 2018.

[31] Science Direct. Real-Time Pricing. https://www.sciencedirect.com/topics/engineering/real-time-pricing.

[32] Anshuman Dutt and Jayant R Haritsa. Plan bouquets: query processing without selectivity estimation. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1039–1050, 2014.

[33] Adam Dziedzic, Jingjing Wang, Sudipto Das, Bolin Ding, Vivek R Narasayya, and Manoj Syamala. Columnstore and b+ tree-are hybrid physical designs important? In *Proceedings of the 2018 International Conference on Management of Data*, pages 177–190, 2018.

[34] Matteo Golfarelli and Stefano Rizzi. From star schemas to big data: 20 years of data warehouse research. *A comprehensive guide through the Italian database research over the last 25 years*, pages 93–107, 2017.

[35] Google. Alloydb for postgresql under the hood: Columnar engine. https://cloud.google.com/blog/products/databases/alloydb-for-postgresql-columnar-engine, 2022.

[36] Hui-I Hsiao, Ming-Syan Chen, and Philip S Yu. On parallel execution of multiple pipelined hash joins. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 185–196, 1994.

[37] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, et al. Tidb: a raft-based htap database. *Proceedings of the VLDB Endowment*, 13(12):3072–3084, 2020.

[38] Bert Hubert. tc(8), linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html.

[39] SnowFlake Inc. Unistore: A modern approach to working with transactional and analytical data together in a single platform. https://www.snowflake.com/workloads/unistore/.

[40] Alekh Jindal, Lalitha Viswanathan, and Konstantinos Karanasos. Query and resource optimizations: A case for breaking the wall in big data systems. *arXiv preprint arXiv:1906.06590*, 2019.

[41] Ryan Johnson, Vijayshankar Raman, Richard Sidle, and Garret Swart. Row-wise parallel predicate evaluation. *Proceedings of the VLDB Endowment*, 1(1):622–634, 2008.

[42] Navin Kabra and David J DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 106–117, 1998.

[43] The kernel development community. Control Groups. https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html.

[44] Michael S Kester, Manos Athanassoulis, and Stratos Idreos. Access path selection in main-memory optimized data systems: Should i scan or should i probe? In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 715–730, 2017.

[45] Michael S Kester, Manos Athanassoulis, and Stratos Idreos. Access path selection in main-memory optimized data systems: Should i scan or should i probe? In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 715–730, 2017.

[46] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, et al. Oracle database in-memory: A dual format in-memory database. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1253–1258. IEEE, 2015.

[47] Per-Åke Larson, Adrian Birka, Eric N Hanson, Weiyun Huang, Michal Nowakiewicz, and Vassilis Papadimos. Real-time analytical processing with sql server. *Proceedings of the VLDB Endowment*, 8(12):1740–1751, 2015.

[48] Juchang Lee, SeungHyun Moon, Kyu Hwan Kim, Deok Hoe Kim, Sang Kyun Cha, and Wook-Shin Han. Parallel replication across formats in sap hana for scaling out mixed oltp/olap workloads. *Proceedings of the VLDB Endowment*, 10(12):1598–1609, 2017.

[49] Mark Levene and George Loizou. Why is the snowflake schema a good data warehouse design? *Information Systems*, 28(3):225–240, 2003.

[50] Guoliang Li and Chao Zhang. Htap databases: What is new and what is next. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2483–2488, 2022.

[51] Meng Li, Zheyu Miao, Di Wu, Feifei Li, Sheng Wang, Wei Cao, Zhi Qiao, Bin Yu Ruan, Kun Yu Liang, Jun Xin Yang, et al. Rovec: Runtime optimization of vectorized expression evaluation for column store. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[52] Quanzhong Li, Minglong Shao, Volker Markl, Kevin Beyer, Latha Colby, and Guy Lohman. Adaptively reordering joins during query execution. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 26–35. IEEE, 2006.

[53] Yan Li, Liwei Wang, Sheng Wang, Yuan Sun, and Zhiyong Peng. A resource-aware deep cost model for big data query processing. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 885–897. IEEE, 2022.

[54] Chen Luo and Michael J Carey. On performance stability in lsm-based storage systems. *Proceedings of the VLDB Endowment*, 13(4), 2019.

[55] Zhenghua Lyu, Huan Hubert Zhang, Gang Xiong, Gang Guo, Haozhou Wang, Jinbao Chen, Asim Praveen, Yu Yang, Xiaoming Gao, Alexandra Wang, et al. Greenplum: A hybrid database for transactional and analytical workloads. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2530–2542, 2021.

[56] Stefan Manegold, Peter Boncz, and Martin L Kersten. Generic database cost models for hierarchical memory systems. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 191–202. Elsevier, 2002.

[57] MySQL. Mysql heatwave. https://dev.mysql.com/doc/heatwave/en/heatwave-introduction.html, 2022.

[58] Fatma Özcan, Yuanyuan Tian, and Pinar Tözün. Hybrid transactional/analytical processing: A survey. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1771–1775, 2017.

[59] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. The star schema benchmark and augmented fact table indexing. In *Performance Evaluation and Benchmarking: First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers 1*, pages 237–252. Springer, 2009.

[60] Massimo Pezzini, Donald Feinberg, Nigel Rayner, and Roxane Edjlali. Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation. *Gartner (2014, January 28) Available at https://www. gartner. com/doc/2657815/hybrid-transactionanalyticalprocessing-foster-opportunities*, pages 4–20, 2014.

[61] Adam Prout, Szu-Po Wang, Joseph Victor, Zhou Sun, Yongzhu Li, Jack Chen, Evan Bergeron, Eric Hanson, Robert Walzer, Rodrigo Gomes, et al. Cloud-native transactions and analytics in singlestore. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, 2022.

[62] Vijayshankar Raman, Gopi Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M Lohman, et al. Db2 with blu acceleration: So much more than just a column store. *Proceedings of the VLDB Endowment*, 6(11):1080–1091, 2013.

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

[64] Mohammad Sadoghi, Souvik Bhattacherjee, Bishwaranjan Bhattacharjee, and Mustafa Canim. L-store: A real-time oltp and olap system. *arXiv preprint arXiv:1601.04084*, 2016.

[65] Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, and Manos Athanassoulis. Lethe: A tunable delete-aware lsm engine. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 893–908, 2020.

[66] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34, 1979.

[67] Sijie Shen, Rong Chen, Haibo Chen, and Binyu Zang. Retrofitting high availability mechanism to tame hybrid transaction/analytical processing. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pages 219–238, 2021.

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

[69] Inc. SingleStore. SingleStore: Real-Time Distributed SQL. https://www.singlestore.com/.

[70] T Spenser and T Loukas. From star to snowflake to erd: Comparing data warehouse design approaches. *Enterprise Systems Journal*, 14:62–69, 1999.

[71] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1041–1052, 2017.

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

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

[74] Ting Yao, Yiwen Zhang, Jiguang Wan, Qiu Cui, gLiu Tang, Hong Jiang, Changsheng Xie, and Xubin He. Matrixkv: reducing write stalls and write amplification in lsm-tree based kv stores with a matrix container in nvm. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, pages 17–31, 2020.

[75] Shaoyi Yin, Abdelkader Hameurlain, and Franck Morvan. Robust query optimization methods with respect to estimation errors: A survey. *ACM Sigmod Record*, 44(3):25–36, 2015.