HyBench: A New Benchmark for HTAP Databases

Chao Zhang
Tsinghua University
cycchao@mail.tsinghua.edu.cn

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Tao Lv
China Software Testing Center
lvtao@cstc.org.cn

ABSTRACT

In this paper, we propose, HyBench, a new benchmark for HTAP databases. First, we generate the testing data by simulating a representative HTAP application. We particularly develop a time-dependent generation phase and an anomaly generation phase for testing HTAP with large cardinality and various anomalies. Second, we propose a set of hybrid workloads. Specifically, we design 18 read/write transactions, 13 analytical queries, and a mix workload of 6 analytical transactions and 6 interactive queries. We also develop a graph-based parameter curation method to control the access patterns including skew access and data contention of the hybrid workload. Third, we propose a unified metric for quantifying the overall HTAP performance. Particularly, we introduce a query-driven method that evaluates the data freshness (lag time between analytics and transactions). Then we introduce a three-phase execution rule to compute a unified metric, combining the performance of OLTP (TPS), OLAP (QPS), and OLXP (XPS) and data freshness. To verify the effectiveness of HyBench and to debunk the myth of different HTAP architectures, extensive experiments have been conducted over five HTAP databases.

PVLDB Reference Format:
Chao Zhang, Guoliang Li, and Tao Lv. HyBench: A New Benchmark for HTAP Databases. PVLDB, 17(5): 939 - 951, 2024.
doi:10.14778/3641204.3641206

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Rucchao/HyBench-2023.

1 INTRODUCTION

Since Gartner coined the term Hybrid Transactional and Analytical Processing (HTAP), we have recently witnessed a surge of various HTAP databases, ranging from standalone HTAP databases [12, 13, 21, 26], distributed HTAP databases [10, 20, 34], to the cloud-native HTAP databases [14, 27]. Different HTAP databases emphasize different characteristics [15]. For instance, row-based architectures favor OLTP-oriented workloads; column-based architectures excel at OLAP-oriented workloads; HTAP architectures with both row store and column store aim to balance the performance on OLTP and OLAP workloads; Unfortunately, it is still unclear how they perform on complicated workloads that include the OLTP, OLAP, and OLXP (i.e., a mix of transactions and analytical queries) simultaneously.

Proceedings of the VLDB Endowment, Vol. 17, No. 5 ISSN 2150-8097.
doi:10.14778/3641204.3641206

Database benchmarking [4, 6, 8, 31, 32, 36–38] is a common practice for evaluating the database performance with three key elements: *data, workload, and metric*. Over the past fifty years, many remarkable benchmarks have been proposed and widely used to facilitate the development of database techniques. Four key criteria should be considered for a domain-specific benchmark: *relevant, portable, scalable, and simple*. However, existing benchmarks [2, 3, 11, 18] are far from an ideal HTAP benchmark. The major concern is that they were adapted from the established OLTP or OLAP benchmarks (e.g., CH-Benchmark [3] combines TPC-C and TPC-H; HATrick [18] extends SSB [22] with TPC-C), thus they lack HTAP-oriented data, workload, and metric. Consequently, it calls for a new benchmark that is designed for HTAP databases in the beginning.

Motivation 1: Data Generation. Existing benchmarks simply generate testing data based on the original data generators, leading to an impedance mismatch between the testing scenario and the realistic applications. First, those data schemas originated from the OLTP or OLAP applications that were not developed for any HTAP applications such as online finance, e-commerce, and fraud detection. Second, those generators produced the data uniformly, but did not simulate any data distributions for HTAP operations. For instance, they cannot generate large cardinality for the queries and cannot produce anomalies (e.g., illegal transactions and risky operations) to evaluate the HTAP performance.

Motivation 2: HTAP Workload. Existing benchmarks mainly blend the original OLTP and OLAP workloads, thus lacking realistic hybrid workloads. For instance, CH-Benchmark [3] has a mixed workload of TPC-C transactions and TPC-H queries, but the "OLTP first, OLAP later" applications, such as finance and banking, treat the transactional throughput as the primary metric, and they normally do not allow for running bandwidth-intensive analytical queries simultaneously. Moreover, existing works did not control the access patterns for the HTAP workload, which can not provide testing cases for simulating various degrees of data contention, version traversal cost, hot-spot access, and deadlock occurrence.

Motivation 3: HTAP Metric. Existing benchmarks lack effective metrics to quantify the HTAP performance. First, they fall short of efficient evaluation on throughput and freshness. Second, they lack a unified metric that can intuitively reflect the important aspects of HTAP performance, including transactional throughput, analytical throughput, data freshness, and workload isolation. For instance, HATtrick [18] introduced an excellent way to visualize the 2D curve of Transactions Per Second (TPS) and Queries Per Second (QPS), but it needs to evaluate the performance many times (at least 72 times) to obtain the throughput frontier. It also added a cross-join operation to every analytical query for calculating the average freshness score using the timestamps of transaction ids.

Comparison with Existing Benchmarks. Table 1 gives a comparison between existing benchmarks [2, 3, 11, 18] and HyBench. It

Table 1: A Comparison between HyBench and Existing End-to-End HTAP Benchmarks

Features	Description	CH-Benchmark[3]	HTAPBench[2]	OLxPBench[11]	HATtrick[18]	HyBench
HTAP Application	HTAP schema and data generation	×	×	×	×	√
Multi-Set Workload	OLTP, OLAP, and OLXP workloads	×	×	×	×	√
Analytical Transaction	Data analytic in the transaction	×	×	√	×	√
Real-Time Query	Data analytic over transaction data	×	√	√	×	√
Data Access Control	Control the access pattern for HTAP	×	×	×	×	√
Isolation Evaluation	Evaluate the isolation for HTAP	√	×	×	√	√
Freshness Evaluation	Evaluate the data freshness for HTAP	×	×	×	√	√
Unified Metric	Quantify the overall HTAP performance	×	×	×	×	√

summarizes eight important HTAP features and shows that existing works only support at most two features while HyBench covers all these features. For instance, existing work blends the original OLTP and OLAP workload while HyBench has the separated OLTP, OLAP, and OLXP workloads; HyBench has a unified metric that can quantify the overall HTAP performance while existing works evaluate the partial performance, e.g., TPS or QPS. More importantly, HyBench designs a realistic HTAP application in the field of finance technology (FinTech) and it simulates the real-world workloads that existing benchmarks do not take into account. For example, both Alipay [17] and WeBank [29] allow for automatic loan reviewing and risk controlling during transaction processing, i.e., analytical transactions, and they can perform the fraud detection over the transaction data directly, i.e., real-time queries. These two kinds of workloads are often handled in a unified HTAP system [28, 35].

Challenges and Solutions. In order to develop an HTAP-specific benchmark, there are three challenges (refers to C1, C2, C3).

(C1) HTAP Workload Design. HTAP applications have different kinds of workloads, e.g., OLTP, OLAP, and OLXP. However, it is challenging to design an HTAP workload to cover all the aspects due to the large design space. On the one hand, the workload should mimic typical operations and use cases. On the other hand, it should present unique benchmarking challenges for HTAP databases. To address C1, we propose the three-phase execution, simulating the OLTP, OLAP, and OLXP workloads separately. Regarding each phase, we propose the choke-point method that designs HTAP-oriented testing points for the transactions and queries, and each template has an associated use case. For instance, we design the analytical transactions that contain joins and aggregations for risk controlling; we design the interactive queries that analyze the latest transaction data for tracing abnormal transactions.

(C2) Access Pattern Control. In order to simulate skew access distributions and real-time anomaly detection, it is necessary to control the workload access patterns such as access distribution, transaction rollback rate, and data contention between queries and transactions. However, it is challenging to control the access patterns simultaneously as the HTAP workloads have complex dependencies (e.g., one query could correlate with multiple transactions). To address C2, we introduce a query-driven anomaly generation phase and a graph-based parameter curation method. It works in two steps. First, it generates various anomalies as the by-product of data generation based on specific risk-controlling operations. Second, it builds a data dependency graph that can find the related transactions and queries with data contention. Consequently, given

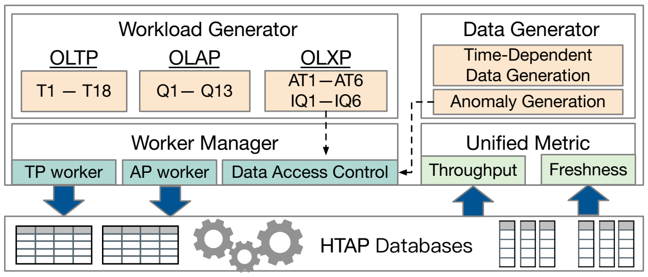

Figure 1: *HyBench* Overview

the specified risk rate and access distributions, it can control the access patterns for the transactions and queries simultaneously.

(C3) Freshness Evaluation. Quantifying the data freshness for an analytical query is challenging from two aspects. On the one hand, evaluating the freshness during the benchmarking process could incur additional overhead, thereby affecting the results of the throughput evaluation. On the other hand, calculating the freshness after the benchmarking process requires an extra analysis process and may not accurately reflect the real-time freshness. To address C3, we adopt a non-intrusive and piggyback method that analyzes the freshness while evaluating the throughput. Specifically, we introduce a contrastive method that compares the query result sets from the OLTP instance and OLAP instance. Since the result sets contain the real-time timestamps, the max difference of their timestamps reflects the data freshness.

In this paper, we propose an end-to-end benchmark for HTAP databases, named HyBench. It features a new data generator, a multi-set workload, and a unified metric. First, we design a schema by simulating a realistic online finance HTAP scenario, then we develop a data generator based on a time-dependent generation phase and an anomaly generation phase. Regarding the workload, we design three sets of workloads for OLTP, OLAP, and OLXP, evaluating the performance of transaction processing, analytical processing, and hybrid processing, respectively. To quantify the overall HTAP performance, we propose a unified metric, H-Score, that combines the performance of OLTP (TPS), OLAP (QPS), and OLXP (XPS) and data freshness.

To summarize, we make the following contributions:

(1) We propose a new benchmark for HTAP databases based on a realistic HTAP application.
(2) We develop a data generator based on a time-dependent generation phase and an anomaly generation phase.

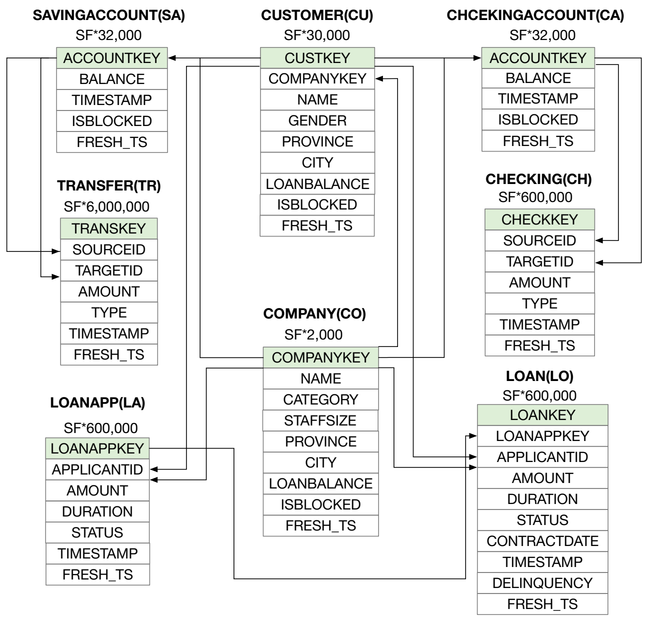

Figure 2: The Data Schema of *HyBench*

(3) We propose a hybrid workload of OLTP, OLAP, and OLXP, based on the choke-point design. To control the access patterns of the hybrid workload, we introduce a graph-based parameter curation method.
(4) We design a unified metric to make a holistic evaluation of HTAP databases, including freshness, transactional, analytical, and hybrid throughput. We also propose a contrastive method to evaluate the freshness effectively.
(5) We have utilized HyBench to evaluate and analyze five widely-used databases with various HTAP architectures. The experimental results show that it provides a more principled approach to quantify the overall HTAP performance.

2 HTAP-NATIVE BENCHMARK

Figure 1 depicts an overview of HyBench, including the workload generator, data generator, worker manager, and the unified metric.

2.1 Data Generation

Figure 2 presents the schema of HyBench that is based on an online finance scenario, which contains many transactional operations (e.g., transferring and checking) and real-time data analytics (e.g., risk controlling for blocked accounts).

2.1.1 Data Schema. HyBench is based on an online finance application inspired by the real-world HTAP applications of Alipay [16, 35] and WeBank [28, 30]. Both Alipay and WeBank have HTAP applications in the field of finance technology (FinTech). Alipay from Ant Group is a leading platform for payments and digital services in China. While handling highly concurrent payment and loan transactions, it also needs to provide high system security by analyzing the data during transaction processing. A typical application is the *3-1-0* model [17] where customers can make a loan application in *3* minutes, and the application can be processed in *1* second with risk controlling, and the whole reviewing process has *0* human intervention. Particularly, *3* is associated with the transaction processing (TP) and *1* pertains to the analytical processing (AP) in risk controlling. WeBank from Tencent Group is a leading internet bank supporting many online banking services, targeting at small and medium sized companies and customers. Similar to Alipay, it has a risk-controlling HTAP application: it supports the automated approval for loan applications within *1* minute [29] by identifying core risks such as credit fraud during transaction processing.

HyBench's schema consists of eight tables that exist in the real schema of Alipay and WeBank. Specifically,the main entities are CUSTOMER and COMPANY, and all the financial activities are involved by them with the associated saving or checking accounts. Particularly, TRANSFER table records the payment transactions by SAVINGACCOUNT; CHECKING table saves the check transactions by CHECKINGACCOUNT. LOANAPP table contains the loan applications issued by customers or companies, and LOAN manages all the loan transactions. Note that the primary key (PK) of each table is marked in green, and each PK points at the referenced foreign key (FK) of the other tables, e.g., CU.CUSTKEY → LA.APPLICANTID. The TIMESTAMP field stores the synthetic timestamps and FRESH_TS stores the real-time timestamps for freshness evaluation. The DELINQUENCY of a delayed loan will be set to 1. If any fraudulent transaction (e.g., overdue loan) is detected, the related users and accounts will be blocked by setting the ISBLOCKED to 1. Consequently, a risk-controlling operation will rollback the transactions for the related blocked accounts.

2.1.2 Time-Dependent Data Generation. HyBench generates the testing data based on the scaling model defined in Figure 2. To determine the table size, we design the scaling models based on those in SSB [22] and HATtrick [18]. For instance, three tables of *Customer, Company, Transfer* have the same scaling of *Customer, Supplier, Lineorder* in SSB and HATtrick. The sum of the table sizes of *Customer* and *Company* amounts to the table size of *Savingaccount*. Since customers normally make much more transfers than loans and checks, we reduce the table size of *Loanapp*, *Loan*, and *Checking* in an order of magnitude. Particularly, a testing dataset is generated based on a given scale factor (SF), and the data size grows linearly as the SF increases. Similar to the existing works [2, 3, 11, 18], the data is mainly generated based on a uniform distribution for the simplicity of benchmarking. Nevertheless, we incorporate the time-dependent data generation to generate data efficiently and make the data more realistic. For example, transfers are only made by customers if they have saving accounts. By dividing the time ranges, transfers can be generated in streams without accessing the full information of saving accounts.

2.1.3 Skew Data Generation. To generate skewed data distributions with HyBench, we have developed a skew data generator based on power law distribution and exponential distribution. Specifically, we use the realistic GDP ranks for the provinces and cities, then generate the data with the specified skew distribution. We also use the correlation to generate the transactions. For example, customers and companies in the same city are more likely to have transfers or make checks than those in different cities.

Table 2: A Summary of Choke Points of HyBench Workload

Category	Description	Choke Points
OLTP	18 Operational Transactions	ACID Test, Row Storage, Batch Write, Hot Spot, Chain Logic, Concurrency Control for High Parallelism
OLAP	13 Analytical Queries	Window Function, Dependent Group-by, Flattening Subqueries, Large IN, Join Ordering, CTE Pushdown
OLXP	6 Analytical Transactions (AT)	Joins and Aggregations within the Transaction, Join Cycle, Left Join Optimization, Concurrency Control
	6 Interactive Queries (IQ)	Few Column Selection, Bi-directional Search, Left Join Optimization, Result Reuse, Join Ordering
	Hybrid Execution (AT & IQ)	High Data and Resource Contention, Long Chain Traversing, Skewed Data Access, Data Freshness

Specifically, since the year of 2014 is the time when the HTAP term was formally defined, we set the start date of data generation to "2014-01-01". With a ten-year range, we assume the current date is "2024-01-01". The generator produces the data in three phases. For phase one (2014-2019), it generates the base data of CUSTOMER, COMPANY, SAVINGACCOUNT, and CHECKINGACCOUNT. We collect the realistic dictionary data of popular names, citys, provinces and company categories in China, then generate the data uniformly. For phase two (2019-2023), it generates the transfers and checkings uniformly. For phase three (2023-2024), it produces the loan applications and accepted loans.

2.1.4 Anomaly Generation. When generating the data, HyBench deliberately generates various anomalies for risk controlling. Specifically, given a probability p, it generates the blocked accounts with p, which is 1% by default. Given a probability for the delinquency loans, it generates the loan applications with curated time range. We use a query-driven method to produce the anomalies based on the analytical transactions. For instance, given an analytical transaction that examines if a target account has related blocked accounts, we will check such a condition while generating the transfers, and the satisfied accounts will be recorded and be written into a file. Similarly, we can generate anomalies for other cases. However, as it is costly to curate a large set of parameters in the runtime, we employ the reservoir sampling to keep a small set of samples. Given a curation parameter γ, it selects a random sample without replacement with the equal probability.

2.1.5 Algorithm Description. As shown in Algorithm 1, given the scale factor *SF*, four dates, and two probabilities, the data generation procedure is divided into three phases, and the table aliases are given in Figure 2, e.g., *CU* refers to *Customer*. The anomaly generation has been plugged into the data generation, and the generated anomalies are used to simulate various risk controlling cases during transaction processing. Specifically, line 1 gets all the scaling models concerning *SF*; line 2 utilizes the *DataGen* function to generate the base data uniformly and the timestamps are generated between d_1 and d_2; line 3 employs the *AnomalyGen* function to generate the blocked accounts with a probability of p_1. That is, if a customer or a company is selected to be an abnormal one, we set the ISBLOCKED field to 1 and we also set the ISBLOCKED fields of its associated saving and checking accounts to 1; line 4 produces the transfer and checking transactions between d_2 and d_3; line 5 identifies the query-driven anomalies based on the analytical transactions (AT) and the generated data. For example, for a generated transfer transaction in TR, we treat a target account in $\hat{SA}_2$ as an abnormal account when its source account in $\hat{SA}_1$ is a blocked account according to

Algorithm 1: Time-Dependent Data Generation

```
Input: Scale Factors: $SF$, Dates: $d_1$, $d_2$, $d_3$, $d_4$, Probabilities: $p_1$, $p_2$
Output: A HyBench Dataset $D$ and Anomaly Parameters $\hat{A}$.
1  $\mathbb{N} = \mathcal{S}(SF)$ ; // Get all the scaling models
   // Phase one: generate the base data
2  $CU, CO, SA, CA$ = DataGen($seed, d_1, d_2, \mathbb{N}_{CU,CO,SA,CA}$);
3  $\hat{CU}, \hat{CO}, \hat{SA}_1, \hat{CA}_1$ = AnomalyGen($p_1, CU, CO, SA, CA$);
   // Phase two: generate the payment transactions
4  $TR, CH$ = DataGen($seed, d_2, d_3, \mathbb{N}_{TR,CH}$) ;
5  $\hat{TR}, \hat{CH}, \hat{SA}_2, \hat{CA}_2$ = AnomalyGen($TR, CH, IQ, \hat{SA}_1, \hat{CA}_1$);
   // Phase three: generate the loan transactions
6  $LA, LO$ = DataGen($seed, d_3, d_4, \mathbb{N}_{LA,LO}$) ;
7  $Duration$ = [30, 60, 90, 180, 365] ;
8  $CuratedDate$ = DateCurate($p_2$, Duration, $d_4$) ;
9  $\hat{LA}, \hat{LO}$ = AnomalyGen($seed, CuratedDate, d_4, Duration$) ;
10 $\hat{A}$ = $Reservoir\_sampling(\hat{SA}_1, \hat{SA}_2, \hat{CA}_1, \hat{CA}_2, \hat{TR}, \hat{CH}, \hat{LA}, \hat{LO})$;
11 return $D = \{CU, CO, SA, CA, TR, CH, LA, LO\}, \hat{A}$ ;
```

AT1; line 6 generates loan transactions between d_3 and d_4; line 7 defines the loan duration in days; line 8 curates the date based on a given probability p_2, the duration, and end date d_4; line 9 generates the delayed loans between *CuratedDate* and d_4. As a result, we can control the number of delayed loans with probability p_2. For instance, given the delinquency probability of 0.1, it generates the loan applications between the curated date "2023-11-01" and end date "2023-12-31", then samples a duration item uniformly from the candidates {30,60,90,180,365}, so the probability to generate a delay loan is 1/2 * 1/5= 1/10 where the delayed loans were signed before "2023-12-01" and had the duration of 30; line 10 selects the anomaly parameters with reservoir sampling such that each parameter is sampled with the same probability, and these parameters are used for simulating the given risk rate; line 11 finally returns the generated dataset and the sampled parameters.

2.2 Hybrid Workload

We have designed three types of workload: OLTP, OLAP, and OLXP. The rationale for having the separating OLTP phase and OLAP phase is that the HTAP databases are often employed to handle these two types of workloads in different time ranges in real scenarios, e.g., process the OLTP workload in the morning and perform the OLAP workload in the evening. The OLXP workload is mainly used for a risk-controlling scenario, where customers can transfer money and apply for loans, and the service provider can analyze the data and control the risks simultaneously.

Table 2 presents a summary of the HyBench's workload, and we design the workloads based on choke-point design [1, 5, 9], which presents many benchmarking challenges for HTAP databases. The first part contains 18 read/write transactions, and the choke points include ACID test, batch write, hot spot, chain logic, etc. The second part includes 13 analytical queries, and the choke points include window function, dependent group-by, flattening subqueries, join ordering, etc. The third part mixes 6 analytical transactions (AT) and 6 interactive queries (IQ) to form a hybrid workload with new HTAP choke points. Hybrid execution refers to the concurrent execution of ATs and IQs, and it also brings new choke points including high data and resource contention, long version chain traversing, skewed data access, and data freshness.

2.2.1 OLTP Workload. We have designed 18 transactions to cover the typical operations in the finance applications.

EXAMPLE 1 (T10: SALARY PAYMENT). *Given the salary @s, this transaction pays the salary to all the employees of a company, which comprises a long transaction as the staff size can be up to 100.*

```
BEGIN;  # Begin the transaction
custList= (SELECT custID FROM CUSTOMER
           WHERE COMPANYID = @ID)
IF(balance< size*@s)
ROLLBACK;  # Rollback the transaction
For custID in custList:
@ID.balance = @ID.balance - @s
custID.balance = custID.balance + @s
COMMIT; # Commit the transaction
```

The OLTP workload contains 8 read-only transactions (T1-T8), and 10 write transactions (T9-T18). T1-T8 frequently view the account information and the related balance; T9 adds a transfer between two accounts, and T10 is the salary payment transaction; T12-T16 involve the loan management. T17-T18 focus on transferring between the saving accounts and the checking accounts.

2.2.2 OLAP Workload. We have designed 13 analytical queries in three levels: customer (Q1-Q4), company (Q5-Q8), and provider (Q9-Q13). Similar to existing benchmarks, each query is associated with a common business case. For instance, Q2 summarizes the loans of a customer, Q7 reports the big accounts for a company.

EXAMPLE 2 (Q9: SUMMARIZE THE REVENUE OF THE NEW LOAN APPLICANTS). *Given two dates, this query aggregates the related transfers for the companies that have loan applications but have no loans yet. Since the query has the clause of EXISTS and NOT EXISTS, it examines if the query optimizer can make the proper optimization, e.g., transform them to a semi-join and an anti-join.*

```
SELECT c.name, sum(tr.amount) as revenue
FROM Company co, Transfer tr
WHERE @Date1< TIMESTAMP <@Date2
AND co.companyId = tr.targetId
AND EXISTS Subquery with EXISTS
(SELECT * FROM loanapp la
WHERE la.applicantid=c.companyid)
AND NOT EXISTS Subquery with NOT EXISTS
(SELECT * FROM loan lo
WHERE lo.applicantid=c.companyid)
GROUP BY co.name
ORDER BY revenue DESC;
```

2.2.3 OLXP Workload. We propose an OLXP workload that blends 6 analytical transactions (AT) and 6 interactive queries (IQ). The rationale of the OLXP workload is that current level of risk controlling is high, thus the transactions and queries need to be ran concurrently to avoid losing money. On the one hand, analytical transactions need to detect the users or transactions with the potential risk. On the other hand, interactive queries need to analyze the newly-emerged abnormal users and transactions immediately. The new set of OLXP workload not only mimics many use cases in a real HTAP application, but also introduces new benchmarking challenges for HTAP databases. For instance, it checks if the long version chain can be traversed and vacuumed efficiently; it examines if the systems can achieve high throughput and high data freshness when the write concurrency and the data contention degree are high; it exercises if the deadlocks can be detected immediately and be addressed properly.

Analytical Transaction (AT). We have carefully designed six ATs (AT1-AT6). Particularly, each AT is associated with one or more risk-controlling cases, if any risking case is detected, it performs the corresponding operations, e.g., rollback the transactions or block the accounts. Take AT1 as an example as follows:

EXAMPLE 3 (AT1: RISK CONTROLLING FOR A TRANSFER). *This transaction incorporates the operations of risk controlling into a transfer operation from a sourceId to a targetId. Firstly, it verifies two conditions: (1) the accounts are not blocked, and (2) the balance is sufficient. Secondly, it checks if the target account has related transfers to any blocked users. If these conditions are satisfied, the related balance will be updated and the transaction will be committed. Otherwise, the transaction will be rolled back.*

```
BEGIN;  # Begin the transaction
IF(Isblocked==1 or balance< @a)
ROLLBACK;  # Rollback the transaction
SELECT COUNT(*) AS CNT
FROM Transfer tr, SavingAccount sa
WHERE sa.isblocked=1
AND tr.targetId=sa.accountId
AND tr.sourceId=@targetId;
IF (CNT>0) ROLLBACK; # Risk controlling
sourceId.balance = sourceId.balance - @a
targetId.balance = targetId.balance + @a
COMMIT; # Commit the transaction
```

Specifically, AT1 identifies the risk that the transferring target has the related blocked accounts; AT2 detects the risk that the checking source has the related blocked accounts; AT3 analyzes if the loan applicant has a balance expenditure; AT4 analyzes a gang analysis case in loan application. AT5 checks if the accepted loan has delayed to repay. AT6 blocks the account that has a delayed loan contract over a duration. To simulate the real distribution, we set the default ratio to { 35%, 25%, 15%, 15%, 7%, 3%} by referring to the real statistic of a hybrid workload from Alipay that was collected by the OceanBase [35] team.

Interactive Query (IQ). We have also designed interactive queries in the hybrid workload. Takes IQ1 as an example:

EXAMPLE 4 (IQ1: FIND RELATED TRANSFERS FOR A BLOCKED USER). *Given a blocked user, this query returns the related transfers. It contains a bi-directional search on the related transfers, and then merges the transfers and returns the top-10 recent transfers. Since a blocked* *user could be found in real time, thus such a query examines if the HTAP databases can successfully find the related accounts in time.*

```
SELECT tr.timestamp, cu.custId
FROM Transfer tr, Customer cu
WHERE tr.sourceId=@blockedId
AND tr.targetId=cu.custId
UNION # Union the transfers
SELECT tr.timestamp, cu.custId
FROM Transfer tr, Customer cu
WHERE tr.targetId=@blockedId
AND tr.sourceId=cu.custId
ORDER BY tr.timestamp LIMIT 10
```

The OLXP workload contains six IQs. Specifically, IQ1 aims to pinpoint the related transfers for a blocked user; IQ2 finds the related checks for a blocked company; IQ3 returns all the employee's information for a blocked company; IQ4 finds the invested companies and individuals from a blocked company; IQ5 searches for the related customers and companies for a delayed loan; IQ6 returns all the related loans and applications for a delayed loan.

2.3 Worker Manager

2.3.1 Worker Control. Given a configuration (n, m), the worker manager will launch n TP workers for the OLTP workload and m AP workers for the OLAP workload, and (n, m) workers for the OLXP workload (i.e., launch n workers for the ATs and m workers for the IQs. Then they are executed concurrently).

2.3.2 Data Access Control. Worker manager can also control the data access patterns for the hybrid workload of ATs and IQs. The basic idea is to control the transaction rollback rate and data contention with the given risk rate and access distribution. Recall that we have curated the parameters by generating the anomalies during the data generation phase, we can utilize such information to control the rollback rate. We also introduce three types of access distributions with different degree of the data contetion: *uniform, power law,* and *latest.* The first mode accesses the data uniformly; the second mode generates the parameters with a skewed distribution following a power law distribution; the third one access the latest n newly-inserted data.

Since the ATs and IQs have different data dependencies, we introduce a graph-based parameter curation method that is based on a dependency graph. As a result, the curation method can control the data access patterns with the risk rate and access distribution simultaneously. It works in three steps as follows:
Step 1. for each IQ, it builds a dependency graph that identifies the related transactions with data contention. Particularly, the dependency graph depicts their source tables by start edges, read tables by read edges, and write tables by write edges, e.g., AT1 has a start edge from SA, read edges from TR and SA, and write edges to TR and SA. The dependency contains two parts: (1) static dependency and (2) dynamic dependency. The former one is identified by tracing their common start edges, and the latter one is found by intersecting their start edges and end edges.
Step 2. given a risk rate $0 < \alpha < 1$, the IQs and the ATs with static dependency samples the anomalies parameters with the probability

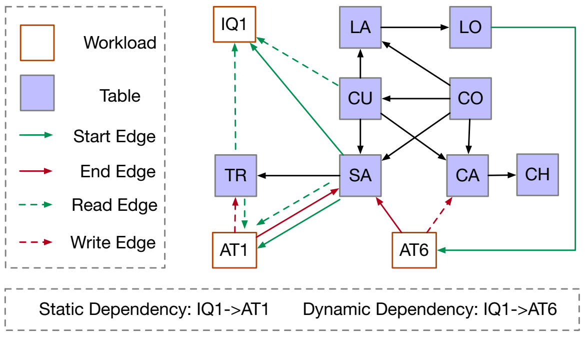

Figure 3: Data Access Control with the Dependency Graph

α. Otherwise, it samples normal parameters with the specified access distribution. Hence, the higher the α is, the higher the transaction rollback rate is. Note that the size of the anomalies parameters does not affect the rollback rate but has impact on data contention.
Step 3. If the given distribution is uniform, it samples the normal parameters randomly. If the specified distribution is power law, it samples the normal parameters with the specified law's exponent (the smaller the exponent is, the more skewed the access distribution is). We implement a binary search to obtain the index, thus both integer and categorical parameters can be sampled with the particular power law distribution. If the target distribution is *latest*, it creates a fixed-size queue to maintain the dynamic parameters, and the dependent IQs and ATs operate the queue simultaneously. When the number of queue reaches the limit, it remove the head item to keep the latest parameters.

EXAMPLE 5 (DATA ACCESS CONTROL). *Figure 3 depicts a dependency graph for IQ1. Tables and workload are represented as square nodes with different colors. We can see that IQ's start edge is from SA, and its read edges are from TR and CU. By tracing the start edges, IQ1 has a static dependency with AT1. By intersecting the start edge with the end edges, IQ1 has a dynamic dependency with AT1 and AT6. When running the hybrid workload with a risk rate α, IQ1 and AT1 sample the related anomalies parameters with the probability α. Otherwise, they sample the normal parameters following the specified distributions (e.g.,* `uniform` *or* `power law` *distributions). Note that AT6 will sample the related parameters with other IQs. If the specified distribution is* `latest`*, IQ1, AT1, and AT6 will have the probability of $1 - \alpha$ to access the latest data written by AT1 and AT6.*

3 HTAP METRICS

3.1 Freshness Evaluation

Data freshness reflects how *fresh* the data is accessed by the analytical queries. We define F-Score to quantify the data freshness.

3.1.1 F-Score Definition. Concerning a query Q, the result sets in OLTP and OLAP are denoted as R^Q_{TP} and R^Q_{AP}, respectively; each tuple in the result set is defined as $r = (i, d, ts)$, where i is the tuple ID, d is the data, and ts_r is its timestamp. We define the F-Score $\overline{f_s}$ as follows:

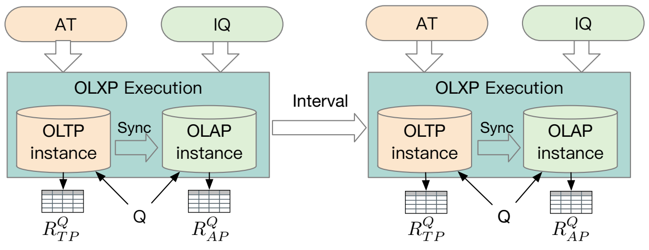

Figure 4: Freshness Evaluation with the Contrastive Method

$$\overline{f_s} = max \begin{cases} ts_{r_a} - ts_{r_b} | r_a \in R_{TP}^{Q}, r_b \in R_{AP}^{Q}, i_{r_a} = i_{r_b} & update \\ ts_{Q} - ts_{r_a} | r_a \in R_{TP}^{Q}, r_b \notin R_{AP}^{Q}, i_{r_a} = i_{r_b} & insert \\ ts_{Q} - ts_{r_a} | r_a \notin R_{TP}^{Q}, r_b \in R_{AP}^{Q}, i_{r_a} = i_{r_b} & delete \end{cases} \quad (1)$$

where *max* takes the maximum value of the timestamp differences between the result sets of R_{TP}^{Q} and R_{AP}^{Q} concerning the cases of update, insert, and delete; ts_Q is the starting time of query Q;

3.1.2 Evaluation Method. To quantify the F-Score effectively, we propose a contrastive method that compares the query result sets from the OLTP instance and OLAP instance while performing the OLXP workload. Since we have added a *fresh_ts* to each table, the query result set should contain the real-time timestamps that are injected by the transactions, and we compute the difference of their max timestamps as the freshness. As shown in Figure 4, AT and IQ are operating over the data consistently, and we launch two IQ threads for fresh evaluation simultaneously. To reduce the evaluation overhead, the fresh is evaluated periodically, and we use a parameter (20 by default) to indicate the number of freshness evaluation. For instance, if the measuring time is 1 min, the freshness evaluation interval is 1*60/20=3s. We launch the additional thread to update the instance periodically, which has the same target id as that of the corresponding fresh query.

We design a query-driven evaluation method to support the F-Score calculation with updates, inserts, and deletes. Specifically, given the result sets of R_{TP}^{Q} and R_{AP}^{Q}, it takes their full outer join on the column of tuple ID. Regarding a tuple that appears both in R_{TP}^{Q} and R_{AP}^{Q} with the same ID, it computes the differences of the timestamps to quantify the data freshness, i.e., $ts_{r_a} - ts_{r_b}$. For an inserted tuple that has not yet emerged in R_{AP}^{Q}, it computes the difference between the query's starting time and the tuple's inserting time, i.e., $ts_Q - ts_{r_a}$. According to the *consistent read* protocol [12, 19], every transaction is associated with a monotonically increasing timestamp, and a query only sees changes made by transactions with smaller timestamps. Hence, the timestamp difference quantifies the data freshness at the time of Q's starting. To support the delete cases, we record the deleted ID and timestamp in a deleted map. Then for a deleted tuple that exists in R_{AP}^{Q} but disappears in R_{TP}^{Q}, we compute the difference between the query's starting time and the tuple's deleting time, i.e., $ts_Q - ts_{r_a}$. Finally, we take the maximum value among the timestamp differences.

3.2 The Evaluation of Workload Isolation

We use the percentage of performance regression/degradation to quantify it by comparing the performance between a sequential execution and a hybrid execution with the OLXP workload. Let W_{TP} and W_{AP} denote the workload isolation for AT and IQ, respectively. The definitions are as follows:

$$W_{TP} = (ATS^{(n,0)} - ATS^{(n,m)})/ATS^{(n,0)} \quad (2)$$

$$W_{AP} = (IQS^{(0,m)} - IQS^{(n,m)})/IQS^{(0,m)} \quad (3)$$

where ATS is the processed analytical transactions per second, and IQS is the processed interactive queries per second. (n, m) is the configured number of TP and AP workers. We define the minimum value of workload isolation is zero.

3.3 Unified Metric

3.3.1 Execution rule. We design a three-phase execution rule to execute the HyBench's workload in a row, then we obtain a unified metric. Given the concurrency number (n, m) and the dataset with scale factor SF, the first phase executes the OLAP workload with m streams, and each stream contains all the 13 queries with a random order. The main metric is the number of processed queries per second (QPS). We ensure each stream is completely processed, and calculate QPS based on its processed queries and actual running time. The second phase performs the OLTP workload with n streams. The main metric is the number of processed transactions per second (TPS). The third phase executes the OLXP workload by running the ATs and IQs concurrently, and they are running with n and m workers, respectively. Two main metrics are the XPS (i.e., the sum of processed AT and IQ operations per second) and F-Score for data freshness.

3.3.2 H-Score Definition. Consequently, we propose a unified metric, named H-Score, to quantify the overall HTAP performance concerning the concurrency of TP workers and AP workers (n, m):

$$\texttt{H-Score} = SF * \frac{\sqrt[3]{TPS * QPS * XPS}}{\overline{f_s} + 1} \quad (4)$$

where XPS = ATS+IQS, and SF is the used scale factor; We adopt the geometric mean of the throughput quantifies the overall performance; $\overline{f_s}$ is measured in seconds and serves as a scaling down factor; it is added with 1 to avoid that the denominator is zero.

H-Score is beneficial as solely relying on one aspect cannot reflect the true HTAP performance. The five components (scale factor, transactional throughput, analytical throughput, hybrid processing throughput, and freshness) in H-Score are widely recognized as the most important factors for quantifying the HTAP performance. However, previous benchmarks [2, 3, 18] only emphasize the hybrid throughput and freshness but neglecting the pure transactional throughput and analytical throughput. In contrast, H-Score combines them into a unified metric for a horizontal comparison. Moreover, it inherently reflects the trade-off between workload isolation and freshness, i.e., the higher the workload isolation, the higher the XPS is, but the freshness could also be larger.

.

4 EXPERIMENTS

4.1 Experimental Setting

Experimental Environment. All experiments are performed on the same type of servers, each of which has a 2.2 GHz Intel Xeon(R) CPU E5-2630 processor with 40 physical cores, 128GB RAM, and a 1T SSD disk. We deploy a four-node cluster, which contains one primary node and three secondary nodes.

Benchmark Configuration. We deploy the benchmark tool on the primary node for ease of data loading. We produce testing datasets with three scale factors, SF1, SF10, and SF100, with raw data of sizes 518MB, 5.3GB, and 56GB respectively. To obtain steady performance results, each phase includes a warm-up run and the measurement run. Particularly, 3 min warm-up and 3 min measurement phase for SF1, 3 min+6 min for SF10, and 6 min+9 min for SF 100. In order to evaluate different workload patterns, e.g., read-heavy, read-write, and write-heavy, we choose most commonly-used combinations with relative ratios of TP and AP workers. Namely, (n:m) ∈ {(50:50), (20:80), (80:20)}. We also evaluate the peak performance with the highest numbers of concurrency, denoted as T-max and A-max for TP and AP workers, respectively. For SF1 and SF10, we set both T-max and A-max to 100. For SF100, we change it to 10 to avoid the out-of-memory issue. We repeat each execution three times and report the average results. We set the buffer size of each database to 100GB for a fair comparison. That is, the testing data fits in memory for all the SUTs. The risk rate is set to 10% by default. For the row store of each database, we built the same indices on all the primary keys and foreign keys.

4.2 Evaluated HTAP Databases

We study five state-of-the-art HTAP architectures. For each architecture, we evaluate one representative. In addition to PostgreSQL, we refer the commercial SUTs to the architecture names (i.e., A1, A2, A3, A4). We anonymize the systems' names because they are commercial databases and have the "*Dewitt Clause*" [33] that forbids the publication of database benchmarks when the database vendor has not sanctioned. Therefore, we discuss the benchmarking results from the perspective of HTAP architectures rather than discussing the specific products. All the results could be reproduced and the configuration files could be found under the *conf* folder, and the README and wiki pages give all the detailed instructions.

Primary Row Store with Streaming Replication. PostgreSQL is an open-sourced relational database supporting HTAP with MVCC. We evaluate PostgreSQL 14 with streaming replication, called `PostgreSQL-SR`, which contains a primary node and three read-only secondary nodes. The primary node processes the read-/write transactions and periodically ships the (Write-Ahead-Log) WAL logs to the secondary seconds for data synchronization. The secondary nodes are used to handle the read queries to achieve better workload isolation. To achieve high freshness, we set the parameters *synchronous_commit* and *hot_standby_feedback* to *on* mode. Nevertheless, it may have lag time due to the log replay latency in the secondary nodes. We route the transactions to the primary node's endpoint, and route the queries to one of the secondary node's endpoints in a round-robin fashion.

(A1) Primary Row Store with In-Memory Column Store. `A1` is a kind of HTAP databases, which handle the transactions using the primary row store, and they speed up the queries with the in-memory column store (IMCS). The columns are automatically selected into the main memory based on the historical queries, and the queries can be accelerated by the column store based on the cost model. An in-memory delta store is utilized for recording the recent DML operations, which will be merged into the column store periodically. Representatives of `A1` are Oracle Dual-Format [12], SQL Server [13], and DB2 BLU [24].

(A2) Distributed Row Store with Column Replica. `A2` is a sort of distributed HTAP databases that rely on hybrid row and column store to enable HTAP. The primary row store is partitioned into multiple segments stored in different nodes. Transactions are handled in the primary node and the logs are replicated to the column store asynchronously. We deploy `A2` with three row-based nodes and one columnar replica. `A2` only exposes a unified endpoint to the client, and it automatically routes the query or operator to either the row store or the column store based on its cost model. To evaluate its freshness, we force the evaluating queries to read the column replica with the corresponding hint. Representatives of `A2` are TiDB [10] and F1 Lightning [34].

(A3) Primary Row Store with Distributed In-Memory Column Store. `A3` is a class of of cloud-based HTAP databases, which utilize a row store with a distributed IMCS to support HTAP. Compared with `A1`, it has better isolation as the transactions and queries are executed in the isolated nodes. The columns are automatically selected to the main memory based on the historical queries, and the analytical queries could be processed with a hybrid scan of row store and column store. Data updates will be synchronized from the primary node to the secondary nodes with streaming replications. Representatives of `A3` are AlloyDB [7] and MySQL Heatwave [20].

(A4) Primary Column Store with Delta Row Store. `A4` is a kind of HTAP databases, which utilize the primary column store as the basis for query processing, and they handle transactions with a delta row store. The primary column store stores the whole data in the main memory. Data updates are appended to the row-based delta store. The system periodically merges the delta data into the column store. We deploy its latest public release on the primary node. However, we only evaluate the performance on SF1 and SF10 due to its memory constraint. Representatives of `A4` are SAP HANA [26] and Hyper [21].

4.3 Overall Throughput

Figure 5 illustrates the overall throughput of all SUTs with varied scale factors and concurrency numbers. The first three groups of bars are the QPS, TPS, and XPS, respectively. The last group of bars shows the total throughput. The results are presented in a log scale for better illustration.

It is clearly visible that the total throughput of `A4` is the highest. We attribute its performance gain to its in-memory columnar query processing, which achieves the average QPS of 386 and 46 on SF1 and SF10, respectively. Moreover, it also achieves the highest TPS of 218 using its delta-based transaction processing on SF10. However, it is not good at processing the OLXP workload, leading to the an XPS of 5.1 on SF10. `A2` has the best performance on SF100, and it can achieve the highest average throughput (QPS: 1.39, TPS: 30.64, XPS: 14.27) simultaneously. We attribute its performance gain to

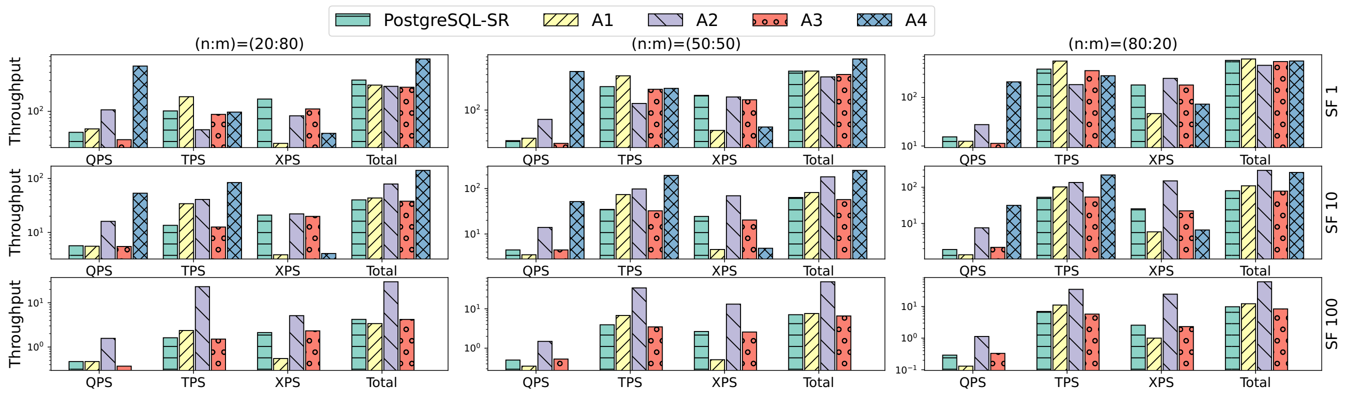

Figure 5: Overall Throughput with Varied Scale Factors and Concurrency Numbers

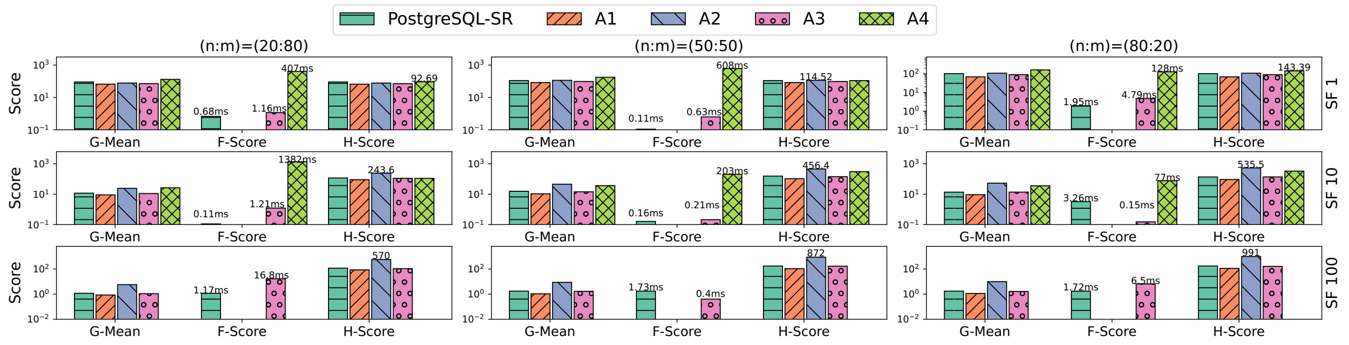

Figure 6: Overall Scoring with Varied Scale Factors and Concurrency Numbers

its distributed storage and MPP-based columnar query processing, which has good scalability for processing large datasets. Although it has the worst overall throughput on SF1 due to high contention on a small dataset, the throughput of A2 is on par with that of A4 on SF10 when (n:m)= (80:20). `PostgreSQL-SR` has the most balanced performance across all the workloads and datasets. Particularly, it is good at processing OLXP workload because of its isolated OLTP and OLAP engines, resulting in the highest XPS of 176 on SF1. With the parallel workers and MVCC mechanism, it can also achieve good performance on QPS and TPS. The most striking one is A3, which has worse performance than pure row-based solution, `PostgreSQL-SR`. By analyzing the query plans, we find the reason is that the hybrid scan with the in-memory column store has worse performance than the pure row-based scan for half of the analytical queries. For example, as the row store with index scan will have less planning time and execution time, it outperforms the hybrid scan on Q9 and Q10, resulting in a higher overall QPS. Such an issue also results in a lower IQS than the row store. A1 achieves the highest TPS in all cases and has the highest average TPS of 369 on SF1. However, it has poor performance on OLAP and OLXP workloads. First, its processing engine excels at transaction processing and the in-memory columnar engine bring a little performance gain on QPS even all the columns are loaded into the column store. Second, it favors high freshness when processing OLXP workload, thus it spends a lot of resources synchronizing the data, leading to the lowest XPS on SF1, SF10, and SF100.

4.4 Overall Scoring

Figure 6 depicts the overall scoring in a log scale including the Geometric mean of the throughput (G-Mean for short), F-Score, and H-Score. We label the F-Score in milliseconds and highlight the winner with H-Score for each plot.

We have five main observations. (1) it is surprising that the one copy solution, A4, has a relatively larger F-Score than others, which has an average of 381ms and 554ms on SF1 and SF10, respectively. By analyzing the query results, we find the reason is that the concurrent queries read the inconsistent data from its versioning delta store. Nevertheless, it still has the highest average H-Score on SF1 (i.e., 115.13) because its higher G-Mean pays off. (2) A2 can always achieve zero freshness as it has a particular validation protocol with the global timestamp oracle (TSO) to ensure read consistency among the replicas. Only when the data version is consistent with the latest version, does it process the queries and returns the results to the client. This also explains why it has the lowest XPS on SF1 as it has an additional validation process for data synchronization. Nevertheless, A2 strikes a good balance between workload isolation and data freshness as it has the highest average H-Score of 412 and 810 on SF10 and SF100, respectively. (3) following A4 and A2,

Table 3: Evaluation on Workload Isolation.

System	W_{TP} 2,8	W_{AP} 2,8	W_{TP} 5,5	W_{AP} 5,5	W_{TP} 8,2	W_{AP} 8,2	W_{TP} max	W_{AP} max
PG-SR	21%	3%	4%	0%	2%	1%	3%	0%
A1	10%	26%	0%	2%	0%	0%	55%	50%
A2	83%	0%	73%	6%	39%	26%	95%	11%
A3	8%	2%	5%	7%	0%	1%	6%	0%
A4	53%	76%	75%	27%	31%	65%	92%	27%

PostgreSQL-SR ranks the second one and third one on SF1 and SF100. As the number of TP workers increases, it incurs a larger F-Score since the workload touches more data to be synchronized to the secondary nodes. (5) A3 has a generally larger F-Score than PostgreSQL-SR due to the additional data synchronization cost for the in-memory column store. Meanwhile, it has the smaller throughput due to the inappropriate hybrid scan, leading to a smaller overall H-Score. (5) A1 has the best performance on SF10 since its TPS is high enough to achieve the highest H-Score. It has also the zero F-Score because of the in-memory delta store.

4.5 Evaluation of Workload Isolation

We run HyBench with *run_xp* mode on SF10, and we compare the results with two sequential executions (e.g., (2,0) and (8,0)) and a hybrid execution (e.g., (2,8)). We also evaluate the *max* workload isolation with (T-max, A-max) =(100, 100).

Table 3 shows the workload isolation of all SUTs. Overall, A3 and PostgreSQL-SR have good workload isolation because they handle transactions and queries in separate nodes. When the concurrency numbers reach the *max*, PostgreSQL-SR and A3 have no performance degradation in IQS and have only 3% and 6% drop in ATS, respectively. A side observation is that executing the hybrid workload occasionally outperforms executing the workloads separately because more related data could be cached. A2 utilizes a unified engine to enable HTAP and it can route queries to the follower nodes for reducing the peak pressure of the primary node. However, since it forces zero freshness, it will wait for the data to be updated until the replica is fresh enough, leading to significant performance degradation. Particularly, it has 95% and 11% drop on ATS and IQS for peak performance. The standalone solutions, A1 and A4, have large workload isolation as they support HTAP in a unified memory space, which causes higher resource contention. Particularly, A1 has a roughly half drop in ATS and IQS and A4 has 92% drop in ATS and 27% decrease in IQS. A1 is better than A4 as A1 has the primary row store and an in-memory column store while A4 relies on one copy of data.

4.6 Evaluation of Freshness and Throughput

We run HyBench with *run_fresh* mode to evaluate the freshness and throughput of all the SUTs. We use SF10 and report the ATS and IQS by varying the ratio of A-max (resp. T-max) with the fixed T-max (resp. A-max). In the upper part of Figure 7, we set the number of TP workers to T-max and vary the number of AP workers. Hence, x-axis stands for the number of AP workers. For instance, we take (T-max, A-max) = (100, 100), so (T-max, 20) refers to 100 TP workers

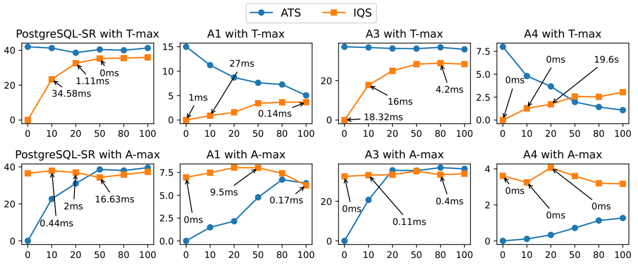

Figure 7: Evaluation of Freshness and Throughput

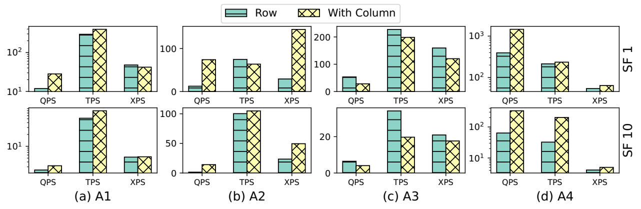

Figure 8: Performance Comparison with Columnar Engine

and 20 AP workers. In the lower part with A-max, x-axis stands for the number of TP workers and y-axis depicts the throughput.

Figure 7 shows the experimental results. For PostgreSQL-SR, it has the highest ATS and IQS for both cases of T-max and A-max. The results also indicate that it achieves good workload isolation because of its isolated architecture. As the concurrency number increases, the ATS line and IQS line are relatively steady. The F-Score increases gradually ranging from 34.58ms to 0ms. A3 also achieves good throughput and workload isolation but its F-Score is higher than PostgreSQL-SR on average. For the T-max case, it incurs a F-Score ranging from 18.32ms to 4.2ms. Besides, it has a lower IQS due to the negative impact of the hybrid scan. The performance of A1 and A4 is significantly affected when the concurrency numbers increase. For the T-max case, the ATS of A1 ranges from 15 to 5 and IQS is below 5. In addition, it can no longer ensure zero freshness and the F-Score increases from 1ms to 27ms. For the A-max case, A1 has the smaller ATS and IQS. Besides, its F-Score increases from 0ms to 9.5ms. A4 has a significant performance degradation for the T-max case with ATS decreases from 7.99 to 1.08. Surprisingly, we observe that it incurs an F-Score of 19.6s in the case of (100, 20). For the A-max case, A4 can ensure zero freshness, but this is achieved by delivering a small ATS and IQS.

4.7 Benefit Evaluation of Columnar Engine

We compare the performance of two processing paradigms: (1) row-only solution and (2) column-based solution. For A2, we build the columnar replica for the entire database. For A3 and A1, we evaluate two column selection strategies: (i) manually selecting all the columns and (ii) using the automatic feature to select and evict the columns with several warm runs. We report the results for the selection strategy that achieves higher throughput.

Figure 8 presents the comparison results with (n:m)=(50:50). Regarding A1, A2, and A4, the results clearly indicate that the column-based solutions significantly outperform the row-based solutions in the overall throughput, which has 1.3x, 2.4x, 2.7x speedup on SF1, and 1.5x, 1.3x, 5.4x on SF10, respectively. It is visible that the performance gain mainly comes from the QPS and XPS. Concerning transaction processing, A2 has a decrease TPS of 15% on SF1 with the columnar replica due to additional synchronization overhead. On the contrary, A1 has an increase of 25% and 35% TPS on SF1 and SF10, respectively. A4 achieves 6.3x better TPS than the row-based solution. Unfortunately, A3 has the worse throughput with the column store, which decreases the overall throughput of 21% and 33% on SF1 and SF10, respectively.

4.8 Peak Performance Evaluation

In this part, we evaluate the peak performance of all SUTs with (T-max, A-max) = (100, 100) for SF1 and SF10, and (T-max, A-max) = (10, 10) for SF100. Table 4 presents the results and Figure 9 depicts a radar chart. Each dimension is graded from 1 to 5: we grade the max value a_m and min value a_n with a score of 5 and 1, respectively. Then we measure the score for the value v as follows: $s = i + 1$ if $v \geq a_n + i * (a_m - a_n)/4$. In addition, we define the grading of F-Score as [(5: 0ms), (4: 0-5ms), (3: 5ms-100ms), (2: 100ms-1s), (1: 1s-10s), (0: >10s)].

For SF1, PostgreSQL-SR is the winner and it achieves the highest H-Score of 320. The performance gain comes from the highest TPS and XPS because of its sophisticated concurrency control and isolated HTAP architecture. It achieves an F-Score of 24ms. A4 has the highest QPS of 676.29. However, it has the low XPS of 45.49 and incurs the largest F-Score of 78ms, leading to an H-Score of 200. A1 has the zero F-Score of 0ms, but its XPS is largely affected by the hybrid workload processing. Still, its TPS is the highest when it comes to highly concurrent workload. A3 incurs an F-Score of 34ms, and its performance bottleneck is the lowest QPS due to the hybrid scan. A2 has the lowest H-Score as it has the lowest TPS for handling the workload of high contention. For SF10, the ranks of H-Score are A2 > A4 > A1 > PostgreSQL-SR > A3. As the data size increases, A2 achieves the highest TPS and tops the list with the highest H-Score. A4 achieves the highest QPS, but it has the lowest XPS of 4.72. The performance bottleneck for PostgreSQL-SR and A3 is the low TPS and A1 suffers from the low XPS. For SF100, the ranks are A2 >PostgreSQL-SR >A3 >A1. The H-Score of A2 is 4.2x higher than A1, and it achieves zero freshness and the highest QPS, TPS, and XPS with the tailored validation protocol, columnar query processing, and distributed transaction processing.

4.9 HyBench Evaluation

Varying the risk rate. We evaluate the performance by varying the risk rate from 0% to 100% with the *run_fresh* mode. We adjust the transaction ratio to evaluate the risk-controlling operations. Figure 10 depicts the XPS lines with the marked F-Score. Regarding PostgreSQL-SR, as the risk rate increases, the throughput increases sharply. The reason is that the higher the risk rate is, the higher the transaction rollback rate is. Hence, more abnormal transactions will be processed with a rollback operation. Moreover, the F-Score decreases gradually as the number of write transactions is decreased.

Table 4: Evaluation on Peak Performance

SF	System	QPS	TPS	XPS	F-Score	H-Score
SF1	PG-SR	82.85	748.65	569.45	24ms	**320**
	A1	138.86	777.45	12.19	0	109
	A2	120.49	189.91	133.4	0	201
	A3	48.79	765.62	561.28	34ms	266
	A4	676.29	327.63	45.49	78ms	200
SF10	PG-SR	8.15	96.85	74.74	14ms	384
	A1	15.38	285.87	12.96	0	385
	A2	12.54	237.2	59.04	0	**559**
	A3	6.07	95.21	65.75	43ms	322
	A4	68.23	252.93	4.72	0	433
SF100	PG-SR	0.59	7.17	4.84	68ms	256
	A1	0.6	13.48	1.25	0	216
	A2	1.56	36.49	13.49	0	**916**
	A3	0.56	6.04	4.49	2ms	247

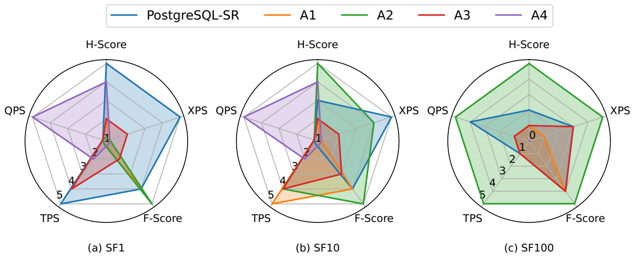

Figure 9: Radar Chart for Peak Performance

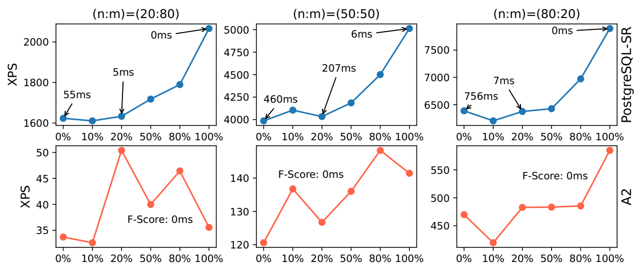

Figure 10: Varying Risk Rate and Concurrency Numbers

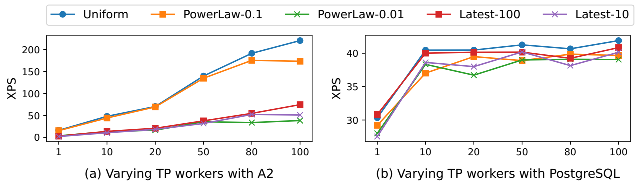

Figure 11: Varying Access Distribution and TP Worker

As for A2, it has a fluctuating XPS line as the risk rate increases. Besides, the more the AP workers are, the more fluctuating the line is. The reason is that it strikes to guarantee zero freshness, thereby blocking the queries that access the stale data.

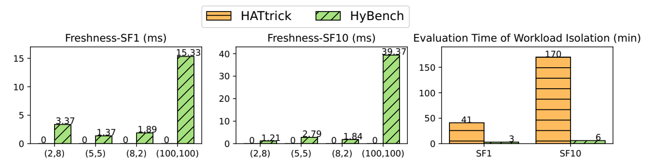

Figure 12: Comparison Between HyBench and HATtrick

Varying the access distribution. Figure 11 shows the XPS with varied access distributions, including a uniform distribution, two power law distributions with an exponent of 0.1 and 0.01, and two latest distributions that access the latest 10 and 100 updated items. We fix *m* to 10 and vary *n* from 1 to 100. Intuitively, the more skewed the access distribution is, the higher the data contention is, thereby having a larger impact on the throughput. Particularly, we have the contention ranks: PowerLaw-0.01 > Latest-10 > Latest-100 > PowerLaw-0.1 > Uniform. The throughput of A2 reflects such a phenomenon. Overall, the throughput of Uniform is 1.1x, 3.4x, 4.1x, and 4.9x than PowerLaw-0.1, Latest-100, Latest-10, and PowerLaw-0.01, respectively. For PostgreSQL-SR, the XPS lines are rather close, indicating PostgreSQL-SR is less sensitive to the skewed access distributions. Particularly, the throughput of Uniform is 5%, 6%, 2%, and 5% higher than PowerLaw-0.1, PowerLaw-0.01, Latest-100, and Latest-10, respectively.

Comparison with HATtrick. We make the experimental comparison between HyBench and HATtrick. Regarding freshness evaluation, we conduct the experiments on PostgreSQL-SR by varying the TP and AP workers. As shown in Figure 12, HATtrick got the zero freshness for all the cases. In principle, we expect to see some lag time as the streaming replication in ON mode replays the logs asynchronously, and some WAL records may not be applied to the OLAP instance in time. In contrast, HyBench found that PostgreSQL-SR can have up to a latency of 39.37ms when the concurrency is increased to 100. Regarding the evaluation of workload isolation, HATtrick computes the throughput frontier by fixing one dimension (e.g., the number of TP workers) and varying another dimension (e.g., the number of AP workers). Then users have to plot the curve and interpret what the curve indicates on their own. Instead, HyBench has the qualitative metric to measure the performance degradation with varied TP and AP workers, which is more intuitive. More importantly, our metric is much more efficient to compute while HATtrick has to perform many runs in each evaluation. As shown in Figure 12, with the same duration for each run (i.e., 1-min for SF1 and 2-min for SF10), HATtrick spent 41 min and 170 min to obtain the results while HyBench only took 3 min and 6 min to perform the evaluation.

4.10 Summary of Main Findings

(1) Overall HTAP Performance. Measuring the overall HTAP performance is crucial as the HTAP databases are defined to handle the OLTP-oriented, OLAP-oriented, and hybrid workload together. Previous studies mainly focus on the hybrid throughput but pay little attention to the unified metric that takes into account freshness, transactional, analytical, and hybrid throughput. For example, HATtrick reported that system-X with the A1 architecture is better than PG-SR and A2. But in our evaluation, A1 can not achieve the highest H-Score when considering the overall performance.

(2) Workload Isolation and Caching. Previous studies conclude that running concurrent OLXP workload can only degrade the performance. We found that the query performance can also benefit from the transaction processing because of caching. Thus, if the resource contention is not intensive, it is preferable to accept a certain amount of transactional requests for better performance.

(3) Fine-Grained Freshness Evaluation. Previous studies mainly adopted the coarse-grained way to quantify the data freshness by using either the database statistics [25] or transaction committed information [18]. Instead, our method leverages the query results to quantify the data freshness, which is a fined-grained method. In our evaluation, we found that both A1 and A4 cannot ensure zero freshness when processing the OLXP workload. Therefore, it is critical to have a particular validation protocol to guarantee the zero freshness regardless of the architecture.

(4) Data Access Controlling. It is important to vary the data access patterns to simulate realistic cases in HTAP applications. The experiments in Section 4.9 have revealed that the HTAP databases perform differently under varied risk rates and access distributions. To the best of our knowledge, HyBench is the first benchmark that can control the data access patterns for the hybrid workload.

(5) Hybrid Scan. Existing systems [7, 23] report that the column store always improves the performance of the hybrid scan. However, the techniques of hybrid scans have a large room to be improved. First, the cost model should be improved. Second, the column selection should consider the memory budget.

(6) HTAP Architecture. We found that different HTAP architectures have their pros and cons. First, PostgreSQL-SR has good workload isolation and high freshness, but it cannot achieve the highest overall throughput on large datasets without a column store. Second, A1 favors high TPS and zero freshness, but has low XPS with the in-memory columnar store. Third, A2 has high throughput and scalability, but is more sensitive to the skew access distribution and hybrid workload. Fourth, introducing the column store into A3 may degrade the overall HTAP performance due to the worse query plan and synchronization overhead. Fifth, A4 has the highest QPS, but suffers high workload interference.

5 CONCLUSION

We propose a new benchmark HyBench, for HTAP databases with a hybrid workload of OLTP, OLAP, and OLXP, supporting throughput evaluation and freshness evaluation simultaneously. Experimental results over five representative HTAP databases verify the effectiveness of HyBench and offer a number of gained insights.

ACKNOWLEDGMENTS

This paper was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (61925205, 62232009, 62102215), Science and Technology Research and Development Plan of China Railway (K2022S005), Huawei, CCF-Huawei Populus Grove Challenge Fund (CCF-HuaweiDBC202309), TAL education, and Beijing National Research Center for Information Science and Technology (BNRist). Thank database teams of GaussDB, OceanBase, Dameng, Alibaba Cloud, and Tencent Cloud for benchmark discussion and evaluation. Guoliang Li is the corresponding author.

REFERENCES

[1] Peter A. Boncz, Thomas Neumann, and Orri Erling. 2013. TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In *TPCTC (Lecture Notes in Computer Science)*, Vol. 8391. Springer, 61–76.

[2] Fábio Coelho, João Paulo, Ricardo Vilaça, José Pereira, and Rui Oliveira. 2017. HTAPBench: Hybrid Transactional and Analytical Processing Benchmark. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. 293–304.

[3] Richard Cole, Florian Funke, Leo Giakoumakis, et al. 2011. The Mixed Workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*. 1–6.

[4] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.

[5] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. 2020. Quantifying TPC-H Choke Points and Their Optimizations. *Proc. VLDB Endow.* 13, 8 (2020), 1206–1220.

[6] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. 2015. The LDBC social network benchmark: Interactive workload. In *SIGMOD*. 619–630.

[7] Google AlloyDB. 2023. AlloyDB Omni overview. https://cloud.google.com/alloydb/docs/omni

[8] Jim Gray. 1993. Database and Transaction Processing Performance Handbook.

[9] Qingsong Guo, Jiaheng Lu, Chao Zhang, Calvin Sun, and Steven Yuan. 2020. Multi-model data query languages and processing paradigms. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 3505–3506.

[10] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, et al. 2020. TiDB: A Raft-based HTAP Database. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3072–3084.

[11] Guoxin Kang, Lei Wang, Wanling Gao, Fei Tang, and Jianfeng Zhan. 2022. OLxP-Bench: Real-time, Semantically Consistent, and Domain-specific are Essential in Benchmarking, Designing, and Implementing HTAP Systems. In *ICDE*. IEEE, 1822–1834.

[12] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, et al. 2015. Oracle Database In-Memory: A Dual Format In-Memory Database. In *ICDE*. IEEE, 1253–1258.

[13] Per-Åke Larson, Adrian Birka, Eric N Hanson, Weiyun Huang, Michal Nowakiewicz, and Vassilis Papadimos. 2015. Real-Time Analytical Processing with SQL Server. *VLDB* 8, 12 (2015), 1740–1751.

[14] Guoliang Li, Haowen Dong, and Chao Zhang. 2022. Cloud Databases: New Techniques, Challenges, and Opportunities. *VLDB* 15, 12 (2022), 3758–3761.

[15] Guoliang Li and Chao Zhang. 2022. HTAP Databases: What is New and What is Next. In *SIGMOD*. 2483–2488.

[16] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous Graph Neural Networks for Malicious Account Detection. In *CIKM*. ACM, 2077–2085.

[17] Lerong Lu. 2018. How a little ant challenges giant banks? The rise of Ant Financial (Alipay)'s fintech empire and relevant regulatory concerns. *International Company and Commercial Law Review (2018), Sweet & Maxwell, ISSN* (2018), 0958–5214.

[18] Elena Milkai, Yannis Chronis, Kevin P. Gaffney, Zhihan Guo, Jignesh M. Patel, and Xiangyao Yu. 2022. How Good is My HTAP System?. In *SIGMOD*. ACM, 1810–1824.

[19] MySQL 8.0. 2023. Consistent Nonlocking Reads. https://dev.mysql.com/doc/refman/8.0/en/innodb-consistent-read.html

[20] MySQL Heatwave. 2021. Real-time Analytics for MySQL Database Service.

[21] Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. 2015. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *SIGMOD*. 677–689.

[22] Patrick E. O'Neil, Elizabeth J. O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *TPCTC (Lecture Notes in Computer Science)*, Vol. 5895. Springer, 237–252.

[23] Oracle 21c. 2023. Automating Management of In-Memory Objects. https://docs.oracle.com/en/database/oracle/oracle-database/21/inmem/configuring-memory-management.html

[24] Vijayshankar Raman, Gopi Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M Lohman, et al. 2013. DB2 with BLU Acceleration: So Much More Than Just A Column Store. *VLDB* 6, 11 (2013), 1080–1091.

[25] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. 2020. Adaptive HTAP Through Elastic Resource Scheduling. In *SIGMOD*. 2043–2054.

[26] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. 2012. Efficient Transaction Processing in SAP HANA Database: The End of A Column Store Myth. In *SIGMOD*. 731–742.

[27] Snowflake Unistore. 2022. Getting Started with Transactional and Analytical data in Snowflake.

[28] Tecent. 2021. WeBank. https://segmentfault.com/a/1190000040792825/en

[29] Tecent. 2023. WeBank. https://www.webank.com/en/product/000001

[30] Tecent. 2023. WeBank. https://www.webank.com/en/characteristic/tech/bigdata

[31] Transaction Processing Performance Council. 2021. TPC-C.

[32] Transaction Processing Performance Council. 2021. TPC-H.

[33] Wikipedia. 2023. David DeWitt. https://en.wikipedia.org/wiki/David_DeWitt

[34] Jiacheng Yang, Ian Rae, Jun Xu, et al. 2020. F1 Lightning: HTAP as a Service. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3313–3325.

[35] Zhenkun Yang, Chuanhui Yang, Fusheng Han, Mingqiang Zhuang, Bing Yang, Zhifeng Yang, Xiaojun Cheng, Yuzhong Zhao, Wenhui Shi, Huafeng Xi, Huang Yu, Bin Liu, Yi Pan, Boxue Yin, Junquan Chen, and Quanqing Xu. 2022. OceanBase: A 707 Million tpmC Distributed Relational Database System. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3385–3397.

[36] Chao Zhang and Jiaheng Lu. 2020. Selectivity estimation for relation-tree joins. In *32nd International Conference on Scientific and Statistical Database Management (SSDBM)*. 1–12.

[37] Chao Zhang and Jiaheng Lu. 2021. Holistic evaluation in multi-model databases benchmarking. *Distributed Parallel Databases* 39, 1 (2021), 1–33.

[38] Chao Zhang, Jiaheng Lu, Pengfei Xu, and Yuxing Chen. 2018. UniBench: A Benchmark for Multi-model Database Management Systems. In *TPCTC*, Vol. 11135. Springer, 7–23.