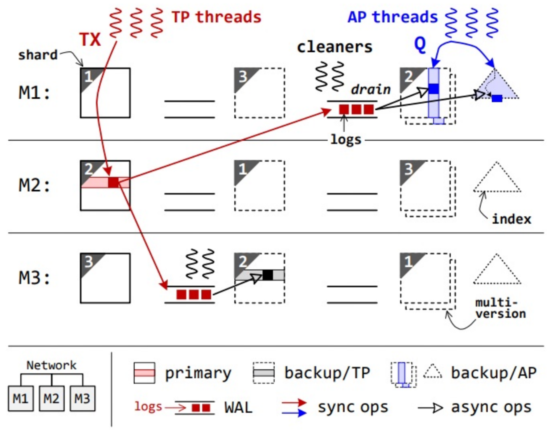
HTAP两个关键目标: Freshness 写入与查询分析延时应接近实时

Performance 与专用系统相比 事务与查询分析应同时执行, 并且性能没有太大下降

VEGITO 内存HTAP 将数据划分为分布在多台无共享机器上的多个分片进行扩展, 同时允许事务及分析查询跨任意数量机器

VEGITO { 执行层 / 内存存储

3机3片



个人觉得最大的贡献点在于提出了一种基于Epoch机制的TP和AP同步的方法，主动的将事务处理分成不同批次，而不是一般的当事务来的时候元数据调度系统被动的被申请全局时间戳，简化了时间戳分配和比较的环节，可能更适合于高冲突的场景

基于epoch

周期广播一个epoch来更新每台机器的epoch, 必须等到所有机器的ACK

AP 需大量历史数据分析

TP 当前事务处理

在线事务用TP库, 数据分析事务用AP库

当分析时需同步TP库, 处理后再数据返回给在线库/Data serving层

VEGITO 同时就会同步AP库

Retrofitting High Availability Mechanism to Tame Hybrid Transaction/Analytical Processing

Sijie Shen, Rong Chen, Haibo Chen, and Binyu Zang, *Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University; Shanghai Artificial Intelligence Laboratory; Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China*

https://www.usenix.org/conference/osdi21/presentation/shen

This paper is included in the Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation.

July 14–16, 2021

978-1-939133-22-9

Open access to the Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation is sponsored by USENIX.

Retrofitting High Availability Mechanism to Tame Hybrid Transaction/Analytical Processing

Sijie Shen, Rong Chen, Haibo Chen, Binyu Zang
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Shanghai Artificial Intelligence Laboratory
Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China

Abstract

Many application domains can benefit from hybrid transaction/analytical processing (HTAP) by executing queries on real-time datasets produced by concurrent transactions. However, with the increasingly speedy transactions and queries thanks to large memory and fast interconnect, commodity HTAP systems have to make a tradeoff between data freshness and performance degradation. Fortunately, we observe that the backups for high availability in modern distributed OLTP systems can be retrofitted to bridge the analytical queries and transactions in HTAP workloads. In this paper, we present VEGITO, a distributed in-memory HTAP system that embraces freshness and performance with the following three techniques: (1) a lightweight gossip-style scheme to apply logs on backups consistently; (2) a block-based design for multi-version columnar backups; (3) a two-phase concurrent updating mechanism for the tree-based index of backups. They collectively make the backup fresh, columnar, and fault-tolerant, even facing millions of concurrent transactions per second. Evaluations show that VEGITO can perform 1.9 million TPC-C NEWORDER transactions and 24 TPC-H-equivalent queries per second simultaneously, which retain the excellent performance of specialized OLTP and OLAP counterparts (e.g., DrTM+H and MonetDB). These results outperform state-of-the-art HTAP systems by several orders of magnitude on transactional performance, while just incurring little performance slowdown (5% over pure OLTP workloads) and still enjoying data freshness for analytical queries (less than 20 ms of maximum delay) in the failure-free case. Further, VEGITO can recover from cascading machine failures by using the columnar backup in less than 60 ms.

1 Introduction

For more than four decades, online transaction processing (OLTP) and online analytical processing (OLAP) are two separate pillars in the database community, with their own design targets and specific fields. Nowadays, many application domains are highly demanding the combination of OLTP and OLAP, such as fraud detection [24, 77, 78], business intelligence [54, 80, 85], healthcare [27, 93], personalized recommendation [117], and IoT [16]. The fundamental reason behind this trend is that much information is most valuable when it first appears, but the value diminishes over time [10, 115]. For example, on 2018 Alibaba's Double 11 Online Shopping Festival (similar to Black Friday Day in the US), the peak throughput reaches 6,000,000 transactions per second, and Alibaba's real-time monitoring system behind it expects to provide a time delay of 20 ms [12]. Meanwhile, users may search for the hottest items and receive personalized advertisements [117]. Vendors also need to detect and prevent online transaction fraud [24] and rely on immediate information to adjust price and stock timely [63].

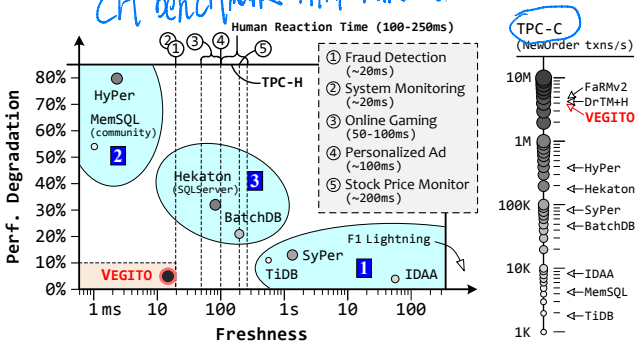


Fig. 1. *The performance-freshness tradeoff for existing HTAP systems with three different architectures and the OLTP performance of existing HTAP systems for TPC-C [95] or CH-benCHmark [29]. VEGITO is located in the desired area of HTAP systems and can offer comparable performance with modern distributed in-memory OLTP systems [46, 87, 107]. **Sources**: published results of HTAP systems [33, 34, 43, 54, 60, 62, 65, 72, 112] and real-time requirements of various application domains [12, 45, 63, 76, 77].*

In response, many recent academic and industrial efforts have been devoted to developing hybrid transaction/analytical processing (HTAP) systems [4, 14, 31, 43, 48, 54, 55, 60, 62, 80, 84, 88, 112], which are expected to support *real-time operational analytics* by breaking the walls between OLTP and OLAP systems. More specifically, analytical queries should be executed on real-time datasets quickly updated by transactions simultaneously. This implies two overarching goals for HTAP systems [43, 60, 101, 112]. **Freshness**: the maximum time delay between the tuple's value written by transactions and read by analytical queries should be near real-time (e.g., tens of milliseconds) in the failure-free case. **Performance**: the transactions and analytical queries should be executed concurrently with little performance degradation (e.g., less than 10%), compared to specialized systems.

Several architectures were proposed (see Fig. 2) but can hardly satisfy both goals of HTAP systems simultaneously, as depicted in Fig. 1. Alternative 1 (DUAL-SYSTEM) connects two OLTP and OLAP-specific systems and performs both workloads at (almost) full speed. However, the cross-system data transfer will cause a large delay (seconds to

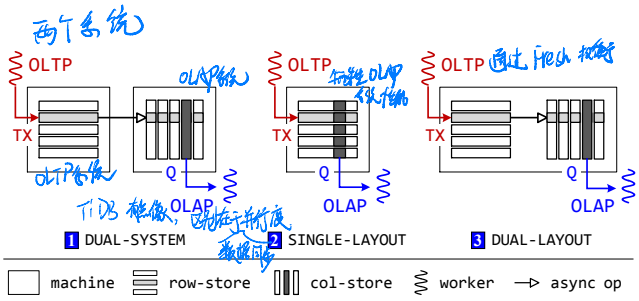


Fig. 2. *A comparison of three existing HTAP architectures.*

minutes) [72, 112].[1] It also doubles the memory cost and incurs additional CPU and network overhead. Alternative 2 (SINGLE-LAYOUT) directly builds an HTAP system derived from one specialized system (e.g., OLTP), which uses a single layout (e.g., row store) for both transactions and analytical queries to ensure data freshness. Thus, it will certainly prefer one type of workload while sacrificing the performance significantly in another (e.g., more than 50% performance degradation [43, 60]). Alternative 3 (DUAL-LAYOUT) carefully combines two different modules into a single system with intra-system data transfer, which ameliorates this problem with a performance-freshness tradeoff. However, the fundamental issues remain (i.e., noticeable performance degradation and time delay).

This paper presents VEGITO, as highlighted in Fig. 1, a distributed hybrid transaction/analytical processing system that retrofits the high availability mechanism (e.g., primary-backup replication) to support HTAP workloads. Specifically, VEGITO executes transactions and analytical queries on primary and backup replicas separately; the transaction updates are always replicated *synchronously* to *multi-version columnar* backups and *tree-based* indexes. Unlike many prior HTAP systems [43, 54, 101], VEGITO keeps both primary and backup replicas in main memory and adopts a symmetric model—each machine both runs HTAP workloads and stores data. However, modern distributed in-memory OLTP systems [34, 74, 83, 89, 107, 113] can provide extremely high throughput (millions of transactions per second) never encountered and targeted by existing HTAP systems (see Fig. 1); it poses new challenges to key components in VEGITO, causing severe performance degradation of both OLTP and OLAP workloads (see §3.2).

To remedy this, we first introduce a classic concept (epoch) into a new context (HTAP) and further propose three new techniques. First, VEGITO introduces a lightweight *gossip-style* scheme to allocate consistent epoch numbers for dependent (distributed) transactions. It allows all logs on a single backup from different transactions to be drained in *parallel*. Meanwhile, to refrain from strict synchronization among different backups, VEGITO demands the analytical query to use the latest stable epoch for reading multiple backups *consistently*. Second, VEGITO chooses a *block-based* design to build multi-version columnar backups for analytical queries, instead of conventional wisdom (chain-based design). Since the transaction updates are applied in rows while the tuples are stored in columns, VEGITO further proposes two optimizations—*row-split* and *column-merge*—to exploit both spatial and temporal locality for writing a columnar backup in rows. Third, VEGITO introduces a *two-phase* concurrent updating mechanism for the tree-based index (e.g., B$^+$-tree) of backups, which is essential to achieve high performance for analytical queries. VEGITO splits the insert operations within an epoch into two phases (i.e., location and update) and parallelizes two phases with two different approaches (i.e., task and data parallelism), respectively.

In addition, while VEGITO uses columnar backups to support OLAP workloads, it still preserves the same availability guarantees for *free*; namely, the backup is still *fault-tolerant*. VEGITO retrofits the replication protocol carefully and restricts changes to the *data layout* of the backup. Thus, it retains the capability of failure recovery. Besides, the original recovery protocol could be used as usual in most cases.

We implemented VEGITO by extending DrTM+H [107], a state-of-the-art distributed in-memory OLTP system. The extensions include retrofitting fault-tolerant backups to run analytical queries and integrating an efficient distributed in-memory OLAP engine, similar to MonetDB [6]. To demonstrate the efficacy of VEGITO, we have conducted a set of evaluations using several micro-benchmarks and a typical HTAP benchmark, CH-benCHmark [29], which combines TPC-C [95] and TPC-H [96] to form a complex mixed workload. For OLTP-only workloads, VEGITO (with 3-way replication) can commit 3.7 million NEWORDER transactions per second when running the TPC-C transaction mix on 16 machines. For OLAP-only workloads, VEGITO can run TPC-H-equivalent queries in 216 ms on average (geometric mean) using a single thread. These results are comparable to the excellent performance of specialized counterparts (e.g., DrTM+H [107] and MonetDB [6]). For HTAP workloads (i.e., CH-benCHmark), VEGITO can achieve a peak throughput of 1.9 million NEWORDER transactions per second and 24 TPC-H-equivalent queries per second simultaneously on a cluster of 16 machines with up to a 1.2 TB dataset. It outperforms state-of-the-art HTAP systems with three different architectures (i.e., MemSQL [4], TiDB [8], and SQL Server [54]) by several orders of magnitude on transactional performance (from 2,911X to 53,138X).[2] Meanwhile, different from prior systems, which have severe performance degradation and poor data freshness (e.g., more than 70% OLTP performance degradation in MemSQL and an 1,500-millisecond delay in TiDB), VEGITO limits the adverse effects to less than 5% and 20 ms simultaneously in the failure-free case. Further, VEGITO can recover from cascading machine failures by using the columnar backup in less than

[1] The data transfer relies on general ETL (Extract, Transform, and Load) tools [2] or specialized techniques (e.g., log shipping [65, 103]).

[2] Note that MemSQL is an in-memory HTAP system, while TiDB and SQL Server are on-disk HTAP systems.

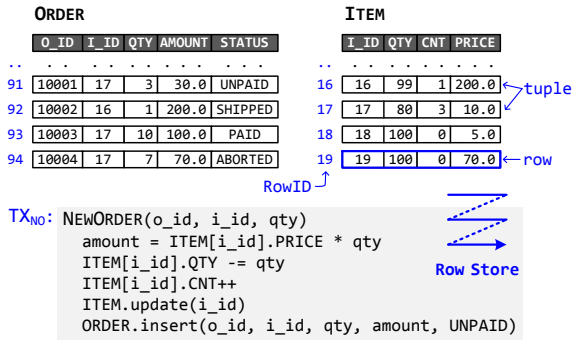


Fig. 3. *A simplified dataset on a row store and a sample transaction (TX_{no}) in TPC-C.*

60 ms. On the other hand, VEGITO can provide comparable failure-free freshness with Amazon Aurora [101].[3] Yet, it also supports fault tolerance and a much higher rate of transactions (e.g., 1.9 million vs. 1,232 NEWORDER transactions per second) by scaling out and processing transactions in the main memory.

In summary, the contributions of this paper are:

- A new distributed in-memory HTAP architecture that retrofits fault-tolerant backups to support hybrid transaction/analytical processing (§3) without compromising high availability (§5).
- Three key techniques with epoch scheme to collectively make a fresh, multi-version columnar backup with tree-based indexes for analytical queries (§4).
- A set of evaluations that confirm the efficacy of VEGITO for HTAP workloads even facing millions of transactions per second (§6).

2 BACKGROUND

Online Transaction Processing (OLTP). The workloads for OLTP systems (e.g., database) usually contain repetitive, short-lived transactions to retrieve and modify tuples (e.g., create/read/update/delete) with ACID guarantees, which are the basis of many applications such as stock exchange, e-commerce, and online order processing. Fig. 3 illustrates a simplified dataset and NEWORDER transaction in TPC-C [95], a popular OLTP benchmark. The sample transaction (TX_{no}) adds a new order (o_id) for selling qty items (i_id), which will append a tuple to ORDER table and update ITEM table. OLTP systems use the row store to exploit data locality and access patterns in transactions, where all attributes of a single tuple are stored continuously.

Moreover, modern in-memory OLTP systems [26, 30, 34, 46, 57, 113] are becoming mainstream, which scale out by sharding a large volume of data across multiple shared-nothing machines and supporting distributed in-memory transaction processing with high throughput and low latency. They usually rely on replication schemes (e.g., primary-backup replication [52] or Paxos state machine replication [30]) to provide high availability even with failures.

[3] Amazon Aurora [101] reports that each read replica typically lags behind the writer by a short interval (20 ms or less). We interprete it as the failure-free freshness. For freshness with failures, we have neither the number for Aurora nor for VEGITO.

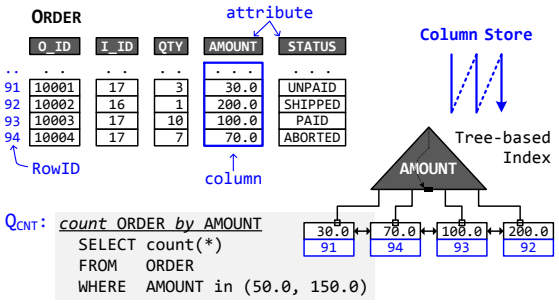


Fig. 4. *A simplified dataset on a column store with a tree-based index and a sample analytical query (Q_{cnt}) in TPC-H.*

Online Analytical Processing (OLAP). By contrast, the workloads for OLAP systems (e.g., data warehouse [32, 40]) usually contain analytical queries to consistently read several attributes of massive tuples (e.g., select/join/filter/aggregate), which also are the basis of many other applications such as business intelligence, financial reporting, and data mining. Fig. 4 illustrates a simplified dataset with a tree-based index, and a sample analytical query in TPC-H [96], a popular OLAP benchmark. The sample query (Q_{cnt}) counts the number of orders (count(*)) with a given range of amounts (from 50.0 to 150.0), which needs to scan ORDER table and the index for AMOUNT attribute. Thus, the column store (aka columnar store) with tree-based indexes is widely used to exploit data locality and access patterns in analytical queries.

Hybrid Transaction/Analytical Processing (HTAP). OLTP and OLAP systems have their own design targets and specific fields, yet many application domains are highly demanding the combination of them; *analytical queries should be executed on real-time datasets quickly updated by transactions simultaneously*. For example, massive new orders are submitted by users (TX_{no} in Fig. 3). Meanwhile, the seller may want to see the number of orders with a given range of amounts in real-time (Q_{cnt} in Fig. 4). The HTAP system should meet the following two goals—freshness and performance—also appearing in recent literature [49, 60, 70, 75, 82, 100, 103].

Freshness. The maximum time delay between the tuple's value written by transactions and read by analytical queries should be near real-time (e.g., tens of milliseconds).

Performance. The transaction and analytical workloads should be executed concurrently with little performance degradation (e.g., <10%) compared to specialized systems.

Nowadays, several HTAP architectures are proposed but could hardly meet two goals simultaneously in the failure-free case—for example, a maximum delay of 20 ms and 10% performance degradation, as depicted in Fig. 1.

3 Approach and Challenges

Opportunity: fault-tolerant backup. It is important and imperative for distributed transaction processing (OLTP) systems to provide high availability (HA), which is guaranteed by replicating tuples on remote machines before committing a transaction. A common approach is to use vertical Paxos [52] with primary-backup replication [4, 7, 26, 34, 46, 59, 69, 97, 108][4], where each shard is commonly configured to use 3-way replication (one primary and two backups). The transaction will *synchronously* ship updates (i.e., logs) to all backups before committing on the primary.

We observe that the *consistent* and *fresh* backup in high availability (HA) provides the foundation for hybrid transaction/analytical processing (HTAP)—running OLTP and OLAP workloads on primary and backup replicas separately. Specifically, for **freshness**, high availability guarantees strong consistency between primary and backups by using synchronous log shipping. Thus, analytical queries can always see the latest updates of transactions. For **performance**, running different workloads on different replicas can avoid interference naturally and deploy optimizations individually (e.g., row store and column store). Moreover, reusing fault-tolerant backups and synchronous log shipping for **free**—there is no compromise on availability (§5)—can avoid extra memory for read replicas and CPU for data synchronization to support real-time OLAP.

3.1 Our approach

VEGITO is a distributed in-memory hybrid transaction/analytical processing system, which targets concurrent OLTP and OLAP workloads over one large volume of data. It scales out by partitioning data into many shards spreading across multiple shared-nothing machines while allowing both transactions and analytical queries to span any number of machines. VEGITO can provide serializability for both transactions and analytical queries. We build VEGITO out of two independent components: execution layer and memory store. An overview of VEGITO's architecture is shown in Fig. 5, which also illustrates the execution of transaction and analytical workloads.

The execution layer employs a worker-thread model by running n worker threads atop n cores playing different roles; each worker thread executes a transaction (e.g., TX_{no}) or a query (e.g., Q_{cnt}) at a time, according to its role (TP or AP worker thread). Following prior modern transaction processing systems [26, 34, 46], VEGITO also leverages 3-way primary-backup replication for high availability. To reduce the overhead of replication, a non-volatile write-ahead log (WAL) is used to buffer the updates (logs) for each backup. Transactions (synchronously) append updates to the WAL of backups involved before committing on the primary, and

[4]Our work is also applicable to other replication-based HA mechanisms, like chain replication [99] and state machine replication [30, 43].

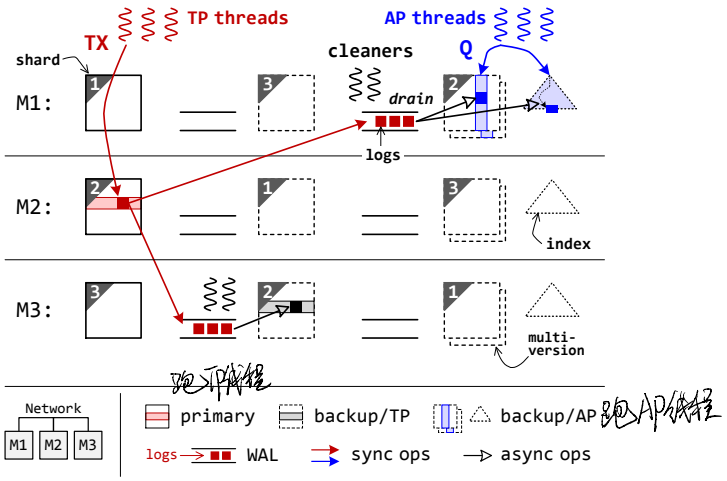


Fig. 5. *An overview of* VEGITO *with three machines and three shards. The transaction (TX) updates an attribute of a tuple in the 2nd shard (M2), and the query (Q) scans the attributes of all tuples.*

then auxiliary (cleaner) threads drain the logs in a lazy and batched manner (asynchronously). VEGITO runs TP and AP worker threads over the primary and one of the backup replicas (aka backup/AP), respectively.[5]

The memory store adopts a general key/value store over a distributed hash table to support a partitioned global address space. Each machine (e.g., M1) stores several primary and backup replicas of different shards. The primary and backup/TP use row stores, while the backup/AP uses a multi-version column store. Specifically, each key-value pair stores an attribute of a tuple. All of the attributes of a single tuple are stored continuously in row stores, while the same attributes of all tuples are stored continuously in column stores. The memory store provides a general key-value store interface (e.g., get and put) and a specific row/column store interface (e.g., row and column) to the above execution layer. Further, tree-based indexes are also maintained with backup/AP for range scans in analytical queries.

Each client contains a client library that parses and ships transactions (TX) and analytical queries (Q) to TP and AP worker threads, respectively. As shown in Fig. 5, the transaction will be executed on the primary using a concurrency control protocol (e.g., two-phase locking [18] or optimistic concurrency control [51]). Before committing the transaction on the primary, all updates are first appended to the write-ahead log (WAL) queue at each machine with a backup. The logs will be applied to backup replicas asynchronously by cleaner threads. On the other hand, the analytical query will be executed on the columnar backup replicas (backup/AP).

Further, the architecture of VEGITO can integrate existing OLTP and OLAP systems instead of writing code from scratch, including transaction/analytical engine, key-value store, data replication and recovery support [26].

[5]This paper uses backup/TP to denote the vanilla backup that provides high availability of transaction processing, and uses backup/AP to denote the columnar backup that also supports analytical processing.

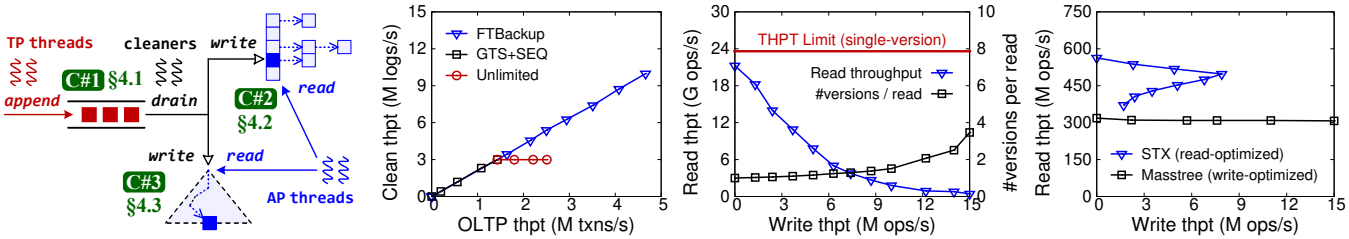


Fig. 6. *(a) A diagram of challenges in* VEGITO. *A performance analysis of three key components in our HTAP architecture, including (b) WAL queue, (c) multi-version column store, and (d) tree-based index. The open-loop clients will send as many transactions and analytical queries as possible until the throughput of some component saturated. Note that each transaction will produce two logs per shard (3-way replication).* ***Workload****: all transactions in TPC-C [95] as OLTP workloads and two simplified analytical queries (similar to Q02 and Q01 in TPC-H [96]) as OLAP workloads for (c) and (d) respectively.* ***Testbed****: A cluster of 16 machines; each machine hosts 8 TP, 10 AP, 4 log-cleaner, and 2 client threads (see §6.1).*

3.2 Challenges

We note that recent HTAP systems also propose to run analytical queries on a separate, read-optimized snapshot of transactional data [43, 60, 92, 112]. However, none of them could meet two goals simultaneously—freshness (a maximum delay of 20 ms) and performance (10% performance degradation)—even under much lower OLTP throughput (e.g., several thousand transactions per second [30, 43, 101]). Differently, our approach reuses *synchronous* log shipping to keep backups consistent and fresh; however, it indeed raises new challenges for minimizing performance degradation on transaction and analytical processing, especially when facing millions of transactions per second.

C#1: consistent and parallel log cleaning. To avoid blocking transaction committing, the updates are appended to WAL queues synchronously and then applied to the backup asynchronously by cleaner threads in parallel (see the left part of Fig. 6(a)). This design is enough and efficient to maintain a fault-tolerate backup [26, 34, 46]. In Fig. 6(b), OLTP throughput (*FTBackup*) can reach about 4.7 million transactions per second, and WAL queues are never full. However, OLAP workloads demand consistent backups. It means that the cleaner threads should drain logs following the dependency in transactions. A common solution is to record a global timestamp in each log and drain logs in sequence [55, 65, 103, 112]. This causes a significant loss (70%) in throughput (*GTS+SEQ*), dropping to 1.4 M txns/s of OLTP throughput. Given WAL queues with unlimited memory (*Unlimited*), we further decouple the performance bottlenecks of transaction processing and log cleaning. Performance degradation can happen for two reasons. First, the OLTP throughput is limited to 2.5 M txns/s due to assigning global timestamps for every transaction in a cluster of 16 machines, causing high contention [106]. Second, draining logs sequentially limits the clean throughput to 3.0 M logs/s and further limits the OLTP throughput to 1.4 M txns/s, since all transactions would be blocked when WAL queues are full. Therefore, VEGITO needs a new approach to draining logs at each machine in a *consistent* and *parallel* way.

C#2: multi-version column store building. The backup for analytical processing (backup/AP) should store tuples in a columnar format to achieve high performance. Meanwhile, cleaner threads and AP threads will write and read the same backup simultaneously, especially with different flavors of locality (row-wise writes vs. column-wise reads). Multi-version concurrency control (MVCC) [19, 110] is commonly used to resolve conflicts between read and write operations by maintaining multiple snapshots. The chain-based design (see the top right corner of Fig. 6(a)) is widely used by multi-version (row) stores [33, 39, 50, 58, 60, 110], and prior HTAP systems [20, 68, 84] also follow this design. However, column-wise reads with a given version (snapshot) have to access pointer-linked tuples (chains) and frequently check their versions, causing massive cache misses and severe performance degradation for analytical queries [58]. As shown in Fig. 6(c), the read throughput drops more than 90% (from 21.2G to 1.8G ops/s) with growing write throughput (10M ops/s), even just accessing 0.5 more versions per read on average. Note that the query (similar to Q02 in TPC-H [96]) simply reads one column updated by TPC-C transactions (like QTY attribute of ITEM table in Fig. 4), and the median latency is about 180 ms over a single-version store, close to the average latency of queries in CH-benCHmark [29]. Therefore, VEGITO demands a new approach to building a multi-version column store that can preserve the locality of both row-wise writes (log cleaners) and column-wise reads (analytical queries).

C#3: concurrent tree-based index updating. The tree-based index (e.g., B^+-tree) is imperative to support range scans for analytical queries. In HTAP systems, the index has to serve both writes (cleaner threads) and reads (AP threads) simultaneously (see the bottom right corner of Fig. 6(a)). Although several research efforts have been devoted to building concurrent tree-based data structures [17, 61, 86, 94, 102, 104, 114], to our knowledge, none of them satisfies our requirements and exploits HTAP workload characteristics. Fig. 6(d) shows the throughput of range scans with growing write throughput for read-optimized and write-optimized tree-based index structures (i.e., STX [21] and

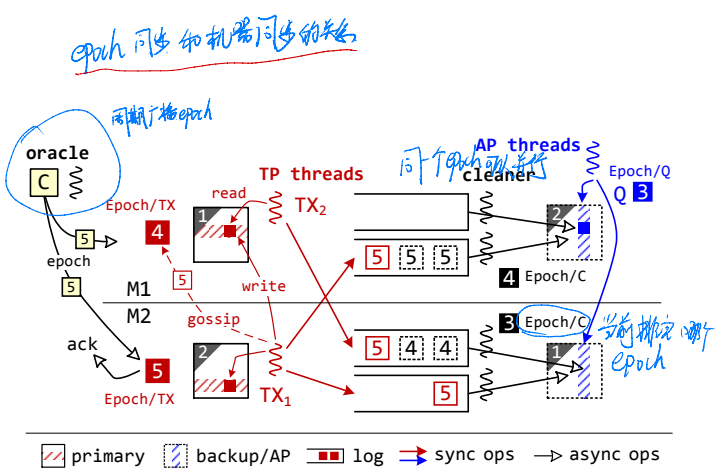


Fig. 7. *An example of lightweight epoch assignment and parallel log cleaning for two machines and two shards.*

Masstree [61]). STX can outperform Masstree by 1.8× for read-only workloads (564 M vs. 319 M ops/s). However, its write throughput is limited to 7.9 M ops/s due to high contention on concurrent writes and collapses when the clients send more write requests. In addition, the read throughput drops up to 34% because of the interference from heavy updates. On the contrary, Masstree is highly optimized to handle fast concurrent writes at the expense of read performance (e.g., unordered keys in leaf nodes). Thus, VEGITO should optimize tree-based index for concurrent writes and reads.

4 DESIGN AND IMPLEMENTATION

To overcome the challenges, we introduce a classic concept (*epoch*) into a new context (HTAP). A centralized (epoch) oracle partitions time into non-overlapping epochs.[6] Epoch is the granularity at which VEGITO guarantees the consistency and visibility of backup/AP replicas to analytical queries. It opens opportunities to exploit parallelism and preserve locality for providing consistent, fresh, and columnar backups. In this section, we detail main techniques in our epoch-based solution employed by VEGITO.

4.1 Consistent and Parallel Log Cleaning

To provide *consistent* backups for OLAP workloads, log cleaner threads on multiple machines should apply logs following the *dependency* in transactions; all logs of one transaction should be applied *atomically*, and all logs from different transactions should be applied *in order*.

A traditional approach is to assign global or vectorized timestamps to logs of transactions and then apply logs sequentially at both machine and thread levels according to their timestamps [55, 65, 101]. When facing millions of transactions per second, this approach would incur excessive cost to transaction processing, and sequential log cleaning would be extremely slow (see *GTS+SEQ* in Fig. 6b).

The epoch-based approach simplifies the assignment and comparison of timestamps with a local scalar value (epoch number), and also allows logs within an epoch (assigned the same epoch number) to be drained in parallel. However, dependent transactions at the epoch boundary must be assigned epochs matching the serial order; namely, committed transactions in earlier epochs never transitively depend on transactions in later epochs. Further, logs in different epochs should still be drained in order.

Consistent epoch assigning. VEGITO introduces a lightweight gossip-style scheme to assign consistent epoch numbers for dependent transactions. An epoch oracle will periodically broadcast a new epoch to update the transaction epoch number (Epoch/TX) on each machine atomically; it always waits for ACKs from all machines such that the epoch gap among machines must not be bigger than 1. Each transaction will assign Epoch/TX on machines involved to its logs during committing (see Fig. 8). For stand-alone dependent transactions on the same machine, the order of epoch numbers can always agree with the serial order due to using the concurrency control scheme (e.g., 2PL or OCC), similar to Silo [98]. For distributed transactions, the epochs from different machines are likely the same, which means all transactions on these machines are in the same epoch. In rare cases, the distributed transaction involves machines within different epochs, as shown in Fig. 7. The distributed transaction TX_1 executes on two machines and observes the epoch on one machine (M1) is behind the other (M2). Suppose a local transaction TX_2 on M1 depends on TX_1, assigning a smaller epoch number (Epoch/TX=4) to TX_2's log would violate the serial order. To avoid this, TX_1 is responsible for synchronizing the epoch (Epoch/TX=5) on machines involved using point-to-point messaging (*gossip*), so that TX_2 will commit its log with the correct epoch number (5).

Fig. 8 shows the commit protocol [34, 107] with a new distributed epoch synchronization step.[7] It first gains the latest epoch number (Line 1-3) from machines involved (mset), and then updates the epoch if needed (Line 4-6). Specifically, we can blindly overwrite the epoch instead of using atomic operations as there are at most two epoch numbers across the cluster. Finally, it will send logs to backups with the consistent epoch number (Line 8). Note that epochs are synchronized after write locking and before read validation. Placing it after write locking ensures that all transactions in the later epochs would see at least the conflict tuple locked; placing it before read validation ensures that committed transaction in earlier epoch never reads the tuple updated by transactions in later epoch. Thus, assigning epochs obeys both dependencies and anti-dependencies.

Parallel log cleaning. Logs from different machines (and threads) are buffered in different queues [26, 34], and mul-

[6] Note that the oracle only needs to periodically broadcast a new epoch to all machines in the cluster, instead of transactions or queries involved. Hence, it will definitely not be the bottleneck in the cluster with thousands of machines even using very small epochs (e.g., a few milliseconds) [15, 30].

[7] The single-machine epoch scheme usually replies on total-store-order (TSO) architectures (like x86-64) to synchronize the epoch among worker threads, like Silo(R) [98, 116].

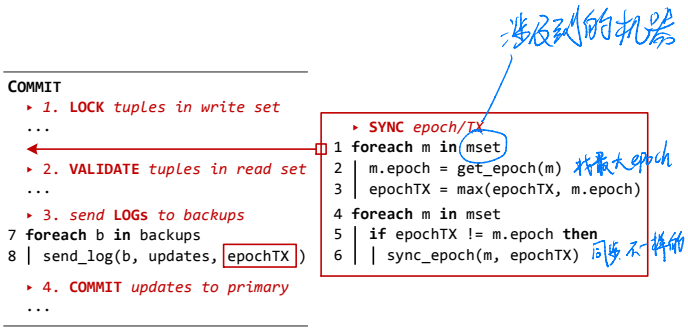


Fig. 8. *Commit protocol with a lightweight epoch scheme run at the end of every transaction.*

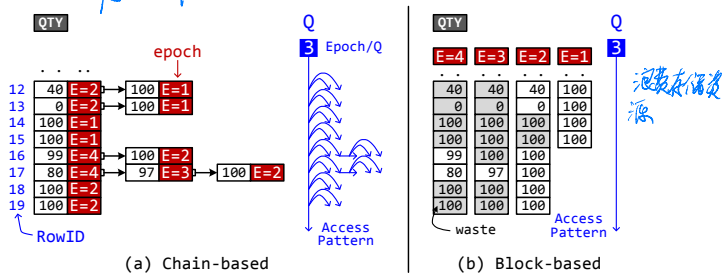


Fig. 9. *Different designs of a multi-version column store for QTY attribute in* ITEM *table. The grey box indicates that the entry is wasted since the tuple is not changed in this epoch.*

tiple cleaner threads can drain logs of the same epoch in parallel. VEGITO uses a hybrid design to exploit both intra-machine and inter-machine parallelism. First, each machine maintains a cleaner epoch number (Epoch/C), meaning that logs at this epoch have been drained. As shown in Fig. 7, after the cleaner thread on M2 applies the last two logs at epoch 4 from M1, the cleaner epoch will increase to Epoch/C=4, and then two cleaner threads could start to drain logs at epoch 5 on M2 in parallel. The runtime schedules queues dynamically across available cleaner threads to achieve load balance [81].

Second, to reduce waiting time among cleaner threads, VEGITO refrains from the synchronization among the cleaner threads on different machines, so that the backup replicas on different machines may not keep up the pace of change. For example, in Fig. 7, the cleaner threads on M1 and M2 are draining logs in different epochs (5 and 4 respectively). To remedy this, VEGITO supports multi-version backup replicas (see §4.2) and makes analytical queries read consistent backups at a (stable) query epoch (Epoch/Q), which is the minimum value of cleaner epochs on machines involved, like Epoch/Q=3 in Fig. 7.

4.2 Locality-preserving Multi-version Column Store

To avoid contention between cleaner threads (write) and AP threads (read), backup/AP replicas require to adopt a multi-version *column* store (MVCS) at the epoch level. Specifically, the cleaner threads will generate a new version of column store for each epoch by applying logs in parallel. Meanwhile, AP threads will run analytical queries over the latest stable version of the column store to retrieve *fresh* results.

The chain-based design is widely used by multi-version (row) stores [33, 39, 50, 58, 60, 110]. Prior HTAP systems [20, 68, 84] also follow this design. As shown in Fig. 9(a), the column store maintains an array for each attribute to store the latest value of tuples with its version (e.g., epoch). Each entry also maintains a backward chain for values in the earlier versions, which is designed to support atomic updates efficiently. Before updating new value in-place, the cleaner thread will copy the original entry and update the chain atomically. Although this design preserves the locality for recent values, analytical queries commonly access the latest *consistent* data. For example, in Fig. 9(a), the cleaner threads are currently draining logs at epoch 4, and the queries can only access the values up to epoch 3. Consequently, the AP thread has to traverse the chains of updated entries and frequently check the versions (see the access pattern in Fig. 9(a)). Besides, the garbage collection for chains would also be complicated and time-consuming.

VEGITO proposes a *block-based* design to exploit optimal performance for analytical queries. As shown in Fig. 9(b), the design is straightforward, which maintains an array for each epoch. When starting a new epoch, the cleaner thread copies the array of last epoch and applies logs to it. This creates a complete snapshot on demand in each epoch. Given an epoch, the AP thread can scan the array with perfect locality but without interference from the cleaner threads. Moreover, the cleaner thread can also garbage collect expired arrays efficiently and reuse the memory easily. However, this design has an apparent and critical drawback (see Fig. 20 in §6.7): data copying may waste lots of CPU and memory resources, especially for append-only attributes (e.g., the attributes in ORDER table). In Fig. 9(b), most of the entries are wasted (grey box) to repeatedly store tuples, which are not changed in the current epoch.

To remedy this, we optimize the naive block-based design by exploiting both *spatial* and *temporal* locality observed in transaction workloads, the data source of MVCS.

Row-split. We observe that *the transactions may focus on updating tuples in a small scope for a while*, like discounted products, batch orders, and social events. Thus, VEGITO first splits values into multiple pages, and each page enables a copy-on-write mechanism independently to implement fine-grained on-demand data copying. There are no new copies for pages without updates at the current epoch. As shown in Fig. 10(a), values of attribute QTY are grouped into two pages. The first page has only two copies for epoch 1 and epoch 2 (i.e., E=1 and E=2), since there is no update in later epochs. To balance the read (AP threads) and write (cleaner threads) performance of the multi-version column store, VEGITO uses 4KB page size, which is enough to exploit the cache locality [44, 91]. Although an insert can be treated as a normal update and triggers the copy-on-write mechanism for a new epoch as well, it is costly for insert-mostly tables

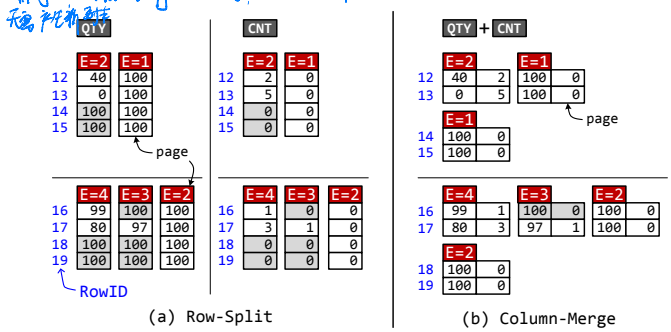


Fig. 10. *Optimizations for block-based multi-version column store with (a) row-split and (b) column-merge for QTY and CNT attribute in* ITEM *table.*

(e.g., ORDER). VEGITO avoids page copying from inserts by appending new values and maintaining offsets for each epoch.

Column-merge. We further observe that *a certain type of transactions usually updates a fixed set of attributes at a time*. For example, NEWORDER transaction in Fig. 3 always updates two attributes (QTY and CNT) of ITEM table together. Thus, VEGITO will merge related attributes for the same tuple into a single page. As shown in 10(b), QTY and CNT attributes of ITEM table are merged. Using column-merge improves the performance of draining logs for cleaner threads and also reduces data copying operations with the same page size, due to finer-grained partitioning for each attribute. VEGITO can automatically discover correlations between attributes from transaction logs and reorganize them into a single page at the start of next epoch.[8] The epoch-based reorganization will not interfere with running queries at all since new pages will not be read by current analytical queries. Further, analytical queries could benefit from the optimization two epochs later.

Finally, after enabling the two optimizations, only 2% of updates incur page copying when using 4KB page size and 15ms epoch interval in a typical HTAP workload (i.e., CH-benCHmark [29]). The median latency to copy a 4KB page is about 6 microseconds. Moreover, our two optimizations are orthogonal to the preceding techniques for the column store [32, 40], such as compression and range filter, so both are applicable in a complementary manner. We leave it to future work.

4.3 Two-phase Concurrent Index Updating

The order-preserving indexes commonly use tree-based data structures to support range scan operations. The update operation may involve more complicated steps (e.g., traversal and split), which will cause new challenges to support fast concurrent updates (by cleaner threads) and lookups (by AP threads). Without loss of generality, the rest of this paper

[8] We collect the statistics on the column family to decide how to merge columns with conflicting requirements.

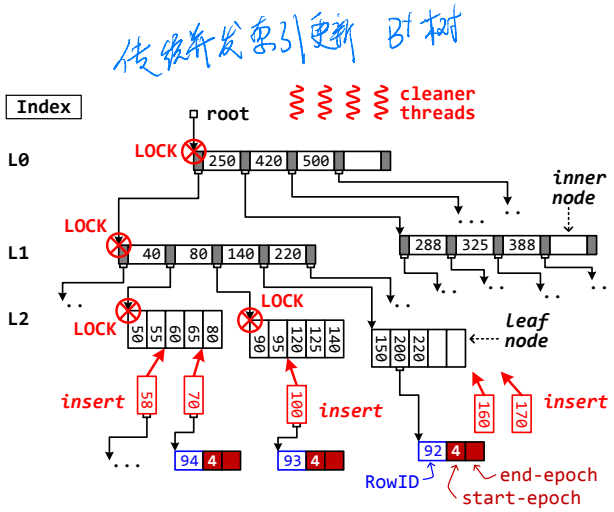


Fig. 11. *An example of traditional concurrent index updating.*

will use B$^+$-tree to explain the issues and introduce our design, since it is widely adopted by OLTP and OLAP systems [37, 43, 47, 101, 105, 109].

As shown in Fig. 11, B$^+$-tree contains two types of nodes: inner nodes and leaf nodes. The inner node stores the values and links to the next level. The last level (L2) contains (sorted) leaf nodes, which are used to store the sorted value of the indexed attribute (AMOUNT) with a link to the row ID and its start/end epochs. The end epoch is used to delete a value, which will simply write an end epoch. The cleaner thread will garbage collect expired values in a lazy and batched manner. The update operation is treated as a delete operation for the original value and an insert operation for the new value. So that we mainly consider the insert operations by cleaner threads and the lookup operations by AP threads.

The INSERT operation consists of three steps.

- **Locate leaf node.** Search a leaf node to store the value by traversing from the root
- **Split/Insert leaf node.** Split the leaf node if it is full, and then insert the value into the sorted leaf node.
- **Split/Insert inner node.** (*Recursively*) Split the upper-level inner node if it is full, and then insert a value and a link into the sorted inner node.

We observe that the throughput of insert operations drops with the increase of threads, even using an optimized B$^+$-tree [104] (see Fig. 21(a) in §6.8). The main reason is that the second step of the insert operation may block other concurrent insert operations. As shown in Fig. 11, for inserting the value 70, the cleaner thread has to recursively lock the leaf node and the upper-level inner nodes due to node split. It will block the concurrent insert operations on the whole subtree (e.g., 58 and 100). Even worse, the node split may cause some blocked operations to lock or rollback the first step since the leaf node has changed (e.g., 58 and 70). Even no split, the second step may still hold the lock of the leaf node for a long time, since it has to move values for keeping

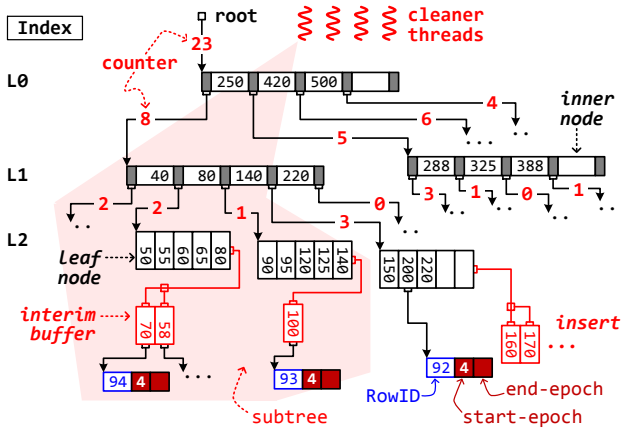


Fig. 12. *An example of two-phase concurrent index updating.*

them in order. However, these efforts may be in vain, as the later insert may move these values again (e.g., 160 and 170). On the other hand, the existence of insert operations will also significantly impact the performance of lookup operations by AP threads (see Fig. 21(b) in §6.8). The lookup operation has to protect access to the inner/leaf node and the value, since the insert operations may change them concurrently.

Fortunately, the epoch-based design provides an opportunity to fully parallelize the index updating. More specifically, the insert operations in the current epoch only need to become visible by the lookup operations until the next epoch. VEGITO introduces a *two-phase index updating* mechanism that splits the insert operations within an epoch into two phases (i.e., *location* and *update*) and parallelizes them using two different approaches (*task parallelism* and *data parallelism*), as shown in Fig. 12.

In the location phase, each thread searches a leaf node to append the value into its interim buffer, and recursively updates the counter at each level. The interim buffer is unordered, and atomic instructions (e.g., CAS) are used to append the value and increase the counter. VEGITO simply uses a vector to implement the interim buffer and resize it according to workloads. Note that the interim buffer is absolutely transparent to lookup operations; thus there is no read/write conflict.

In the update phase, we first use a top-down greedy strategy to partition the tree into non-overlapping subtrees according to the counters at each level, so that each subtree has a similar amount of tasks. Then each thread will insert the values within a subtree in a batch, which also avoids redundant node splits and data movements. Finally, we use a single thread to split the top-level (L0) node if necessary. Consequently, there are no conflicts between cleaner threads in both location and update phases. However, the lookup operations (by AP threads) still may conflict with the update phase of the insert operations. To minimize the impact on lookup operations, cleaner threads can leverage RCU [64] mechanism or HTM [41] to implement the update phase.

5 No Compromise: Availability

VEGITO assumes that the OLTP system has already provided high availability using replication (e.g., 3-way primary-backup replication [26, 34, 46]) and other fault-tolerant techniques (e.g., failure detection and non-volatile WAL). VEGITO reuses this mechanism to support HTAP workloads and still preserves the same availability guarantees for free—namely, *there is no need for extra replicas*. Because VEGITO just reorganizes the *data layout* of one backup replica (backup/AP), from row-wise store to column-wise store; the backup/AP can still provide the capability of failure recovery. Besides, the original recovery protocol [26, 34] is used as usual in most common cases.

Backup failure. When the backup/TP fails, VEGITO will rebuild a row-wise backup from the primary by following the original protocol. When the backup/AP fails, VEGITO will rebuild a column-wise backup to the next epoch, because both the primary and the backup/TP do not store epochs associated with tuples for memory savings and good locality. Meanwhile, VEGITO needs to re-execute analytical queries involved with the new epoch.

Primary failure. When the primary fails, VEGITO always *prefers* to promote a surviving backup/TP to be the new primary and rebuild a new backup/TP on another machine later in the background, which still follows the original protocol. When both the primary and the backup/TP fail (a rare case), VEGITO rebuilds a new primary based on the surviving backup/AP on the same machine (∼42 ms for 12 warehouses of TPC-C, see §6.5), instead of promoting it, and then migrates the backup/AP to another machine later in the background. This *rebuild-and-migrate* design avoids lengthy data reorganization between row store and column store, compared to the conventional *promote-and-rebuild* approach. Note that our block-based design also simplifies and accelerates this procedure. Therefore, VEGITO can still offer comparable performance against promoting a backup/TP (∼7% overhead). In addition, it also avoids interrupting analytical queries.

It should be noted that the recovery scheme prefers OLTP performance. VEGITO chooses to abort the analytical query that accesses failed machines and retry it after recovery, since the long-running analytical query is unusual in HTAP workloads [29] (e.g., real-time analytics), especially for in-memory systems. If the long-running analytical query is a serious problem, for example the query latency exceeds the mean time to failure (MTTF) of HTAP systems, both the primary and the backup/TP should store epochs associated with tuples. VEGITO thus could suspend and resume the analytical query after recovery.

6 Evaluation

We implemented VEGITO by extending DrTM+H [107], a state-of-the-art distributed in-memory OLTP system. The

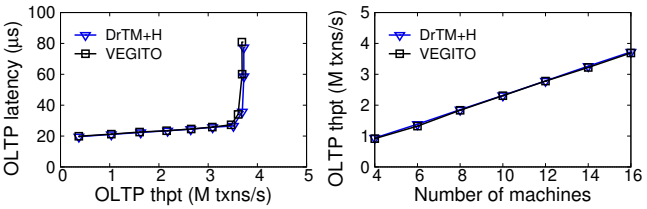


Fig. 13. *Comparison of (a) performance and (b) scalability between* VEGITO *and DrTM+H using CH-benCHmark with OLTP-only workloads.*

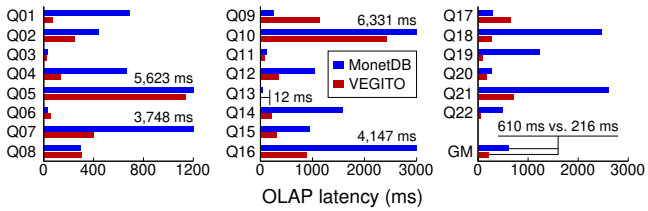


Fig. 14. *Comparison of single-threaded latency (ms) between* VEGITO *and MonetDB using CH-benCHmark with OLAP-only workloads, equivalent to TPC-H with SF=10.*

extensions include retrofitting the high availability mechanism (3-way primary-backup replication) for hybrid transaction/analytical processing and integrating a distributed in-memory OLAP engine, similar to MonetDB [6, 44], a column-store database that maps analytical queries into a series of array operations [23].

6.1 Experimental Setup

Hardware configuration. All experiments were conducted on a rack-scale cluster of 16 machines. Each machine has two 12-core Intel Xeon processors, 128GB of RAM, two ConnectX-4 100Gbps IB NICs and an Intel 10GbE NIC. Unless otherwise noted, we reserve 4 cores for log cleaner threads and 2 cores to generate transactions and analytical queries in parallel for local worker threads, which avoid the impact of networking between clients and servers, as done in prior work [26, 97, 98, 105, 111]. For HTAP workloads, we pin 8 TP threads and 10 AP threads on the remaining cores.

Benchmarks. We use CH-benCHmark [29], a typical HTAP benchmark derived from unmodified TPC-C [95] (OLTP benchmark) with some necessary tables to fulfill equivalent queries from TPC-H [96] (OLAP benchmark). It contains 5 types of transactions and 22 analytical queries. We run the full mix and report OLTP throughput as the number of NEWORDER transactions committed per second and OLAP throughput as the number of analytical queries executed per second. CH-benCHmark scales by partitioning a database into multiple warehouses spreading across multiple machines. We deploy 12 warehouses on each machine with 3-way replication, namely 12 primary, 12 backup/TP and 12 backup/AP replicas. Each machine hosts about 6GB initial data, which rapidly grows up to 75GB (a total of 1.2TB) through continually running transactions (e.g., NEWORDER) for about 40 seconds. To eliminate the effect of growing data, the analytical query will access a fixed size of latest data by using LIMIT statement.

Comparing targets. We choose DrTM+H [107] and MonetDB [6] (v11.33.3) as the representative in-memory OLTP and OLAP systems, respectively, to show that both OLTP and OLAP performance of VEGITO are comparable to specialized counterparts. VEGITO follows 3-way replications of DrTM+H except for replacing one of backup/TP replicas with one backup/AP replica. To eliminate the performance discrepancy, VEGITO also uses the query plans generated by MonetDB for all of the analytical queries. VEGITO optimizes distributed joins as MemSQL by adding reference tables (copies) on each machine and aggregating intermediate results to avoid the whole table transferring [4]. In addition, the default intervals of epoch and garbage collection (GC) are set as 15 ms and 1 second, respectively.

For HTAP workloads, we mainly focus on the performance degradation and the freshness in VEGITO against three state-of-the-art HTAP systems with three different architectures—namely TiDB v4.0 [8] with TiFlash [9] (DUAL-SYSTEM), the community edition of MemSQL v7.0 [4] (SINGLE-LAYOUT), and SQL Server 2019 [7] (DUAL-LAYOUT). Note that MemSQL is an in-memory system, while TiDB and SQL Server are on-disk systems. For SQL Server, we host all data in main memory by using tmpfs, an in-memory file system. TiDB demands all data on the disk with the ext4 file system. In addition, we deploy TiDB on the cluster with different settings[9] and always report the best results of them. Differently, MemSQL and SQL Server can only run on a single machine, and we deploy them on one of our testbed machine without replication (just 12 warehouses). Finally, to avoid the impact of compiling and interpreting analytical queries, we directly evaluate the performance of executing analytical queries on servers.[10]

6.2 Overall Performance

OLTP-only workloads. We first compare OLTP performance of VEGITO and DrTM+H using CH-benCHmark with OLTP-only workloads, like TPC-C [95]. As shown in Fig. 13, the peak throughput of VEGITO reaches 3.7 million NEWORDER transactions per second when running the full mix on 16 machines (each has 14 TP threads), just 1% lower than DrTM+H. This is thanks to our epoch-based scheme and gossip-style parallel log cleaning, which avoid blocking transactions. The best published TPC-C performance we know of is from FaRMv2 [87], which can commit 5.4 million NEWORDER transactions per second on 90 machines. In

[9] As recommended in TiDB's official website [9], we deployed TiKV and TiFlash in both the same and different nodes.

[10] The systems evaluated in our paper use different approaches to run analytical queries—namely VEGITO (hand-written C++), MonetDB (interpreted SQL), MemSQL (compiled SQL), TiDB (complied SQL), and SQL Server (compiled SQL).

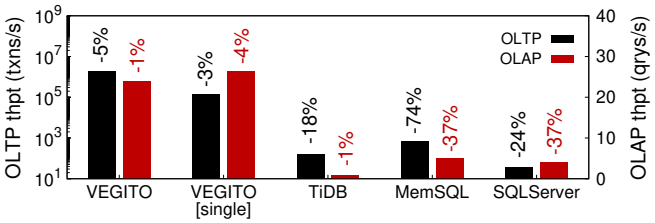


Fig. 15. *Comparison of OLTP and OLAP performance of different HTAP systems using CH-benCHmark with hybrid workloads. The labels upon histogram are performance degradation. Note that the left y-axis (OLTP throughput) is in log scale.*

general, VEGITO can offer OLTP performance comparable to state-of-the-art specialized systems (e.g., DrTM+H and FaRMv2) and scales well on a cluster with tens of machines.

OLAP-only workloads. We further compare OLAP performance of VEGITO and MonetDB using CH-benCHmark with OLAP-only workloads, like TPC-H [96]. Fig. 14 compares the single-threaded latency of analytical queries on VEGITO and MonetDB; both of them use the same query plans generated by MonetDB. To compare with the published TPC-H results [35, 38, 67], we scale the database in CH-benCHmark following a similar approach in TPC-H by a scale factor of 10 (SF=10). As shown in Fig. 14, the average (geometric mean) latency of VEGITO (GM) outperforms MonetDB by 2.8× (216 ms vs. 610 ms). The main performance improvement in VEGITO is due to combining some operators manually and using efficient string operations by hand-written C++. VEGITO also outperforms published TPC-H results for various query processing engines [35, 38]. Specifically, the average (geometric mean) latency of all TPC-H queries (SF=10) using a single thread is 568 ms for HyPer [68], 541 ms for Umbra [67], 1,125 ms for Hyrise [36] and 619 ms for MonetDB [6].[11] Overall, VEGITO's OLAP performance matches state-of-the-art specialized systems.

HTAP workloads. Fig. 15 shows both OLTP and OLAP throughput of VEGITO and other available HTAP systems using CH-benCHmark with hybrid workloads. To study performance degradation, we evaluate each system twice. We first run OLTP and OLAP workloads separately and tune the number of clients to use half of CPU resources. Then, we run HTAP workloads with the same number of clients to saturate CPU resources with a balance between OLTP and OLAP engines. The results of performance degradation in Fig. 15 (labels) are the difference between the two runs.

VEGITO can perform 1.9 million TPC-C NEWORDER transactions and 24 TPC-H-equivalent queries per second simultaneously. The OLTP throughput of VEGITO is several orders of magnitude higher than that of its competitors (11,808× for TiDB, 2,911× for MemSQL, and 53,138× for SQL Server). This means that the bridge between two ends of the world in VEGITO—parallel log cleaning, column store

[11] Note that we calculate the geometric mean of the query times based on the reported results of every query [35, 38].

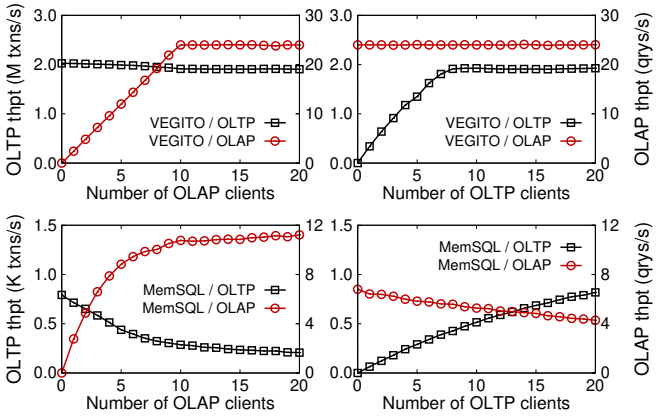


Fig. 16. *Performance degradation on* VEGITO *and MemSQL with the increase of OLAP and OLTP workloads, respectively.*

building, and tree-based index updating—is strong enough to face the challenge of extremely high throughput, which is never appeared in prior published results of HTAP systems, to the best of our knowledge. Compared to single-machine HTAP systems, like MemSQL and SQL Server, VEGITO still has orders of magnitude higher OLTP throughput per machine (120 K txns/s) with support for scaling out and fault tolerance.

Moreover, VEGITO also provides little throughput degradation when running hybrid workloads, just 5% for OLTP and 1% for OLAP respectively. In contrast, existing HTAP systems, TiDB, MemSQL, and SQL Server, suffer from significant performance degradation, reaching 18%, 74%, 24% for OLTP and 1%, 37%, 37% for OLAP respectively. It matches well with the characteristics of different HTAP architectures (see Fig. 1).

We further deploy and evaluate VEGITO on a single machine by hosting all three replicas of each shard (one primary and two backups) on the same machine. VEGITO still synchronously send transaction logs between primary and backups by the NIC. As shown in Fig. 15, on a single machine, VEGITO can perform 132 thousand TPC-C NEWORDER transactions and 26.5 TPC-H-equivalent queries per second simultaneously. Note that running analytical query on a single machine is more efficient due to eliminating network overhead.

6.3 Performance Degradation

To study the impact of performing hybrid workloads simultaneously, we follow Gartner's recommendation to instruct one kind of clients (e.g., OLTP) to sustain a configured throughput (i.e., about half of peak throughput) and allowing another kind of clients (e.g., OLAP) to saturate the throughput [28].

VEGITO can provide strong performance isolation by dedicating a fixed number of worker threads for OLTP and OLAP workloads. We carefully put the memory of two classes of threads into different cache lines (e.g., write epoch and read epoch, write offset and read offset) to mitigate the impact

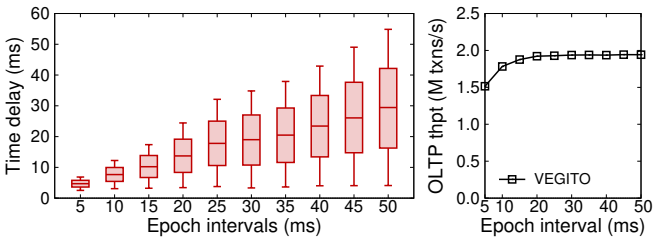


Fig. 17. *(a) The time delay and (b) OLTP throughput of* VEGITO *with the increase of epoch intervals in the failure-free case.*

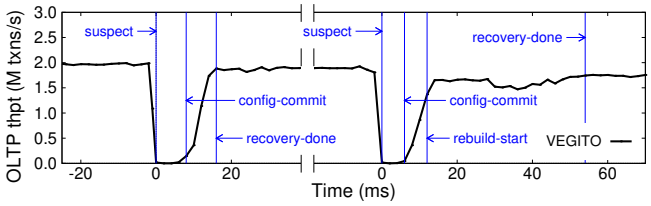


Fig. 18. *The timeline of failure recovery.* ***suspect****: the failed machine is detected;* ***config-commit****: new configuration is committed at all surviving machines;* ***recovery-done****: the recovery of primary is done;* ***rebuild-start****: backup/AP starts to rebuild primary.*

on the cache. As shown in Fig. 16, with the increase of AP clients, OLTP performance degradation of VEGITO is less than 5%. After OLAP performance is saturated, OLTP performance also remains stable. When the roles are reversed, OLAP performance degradation becomes trivial (1%) since OLAP worker threads always use a stable epoch to perform analytical queries on a specified column store and index. In contrast, MemSQL suffers from severe performance degradation of both OLTP and OLAP workloads, even if there are adequate resources. In Fig. 16, the performance degradation of MemSQL reaches 74% and 37% for OLTP and OLAP respectively, with the increase of another type of workloads. This is largely due to the high contention between OLTP and OLAP engines over shared data.

6.4 Freshness

The freshness is defined as the maximum time delay between an update was committed by the transaction (OLTP workload) and this update can be read by the analytical query (OLAP workload). Fig. 17(a) shows the freshness of VEGITO with the increase of epoch intervals in the failure-free case. The median time delay is about 70% of the epoch interval, and the maximum delay is up to 1.3× of epoch interval. It implies that we could roughly limit the freshness in VEGITO by setting an appropriate epoch interval.

Moreover, by setting the epoch interval, there would be a tradeoff between the freshness (OLAP) and the performance degradation (OLTP) in VEGITO. In Fig. 17(b), when using a relative short epoch interval (less than 10 ms), performance degradation would become non-trivial (10%) since epoch-based design limits the parallel log cleaning within an epoch, and the cost to build a column store for each epoch is hard to be amortized. Considering the latency of analytical queries (see Fig. 14), the epoch interval with tens of milliseconds would be moderate and reasonable. The default epoch interval is set as 15 ms, providing a freshness less than 17.4 ms.

As a reference, on our testbed, the maximum delay in TiDB, MemSQL, and SQL Server are about 1,534 ms, 1.2 ms, and 46 ms, respectively. The results are compatible with the characteristics of different HTAP architectures (see Fig. 1). Further, VEGITO can provide a comparable failure-free freshness with Amazon Aurora [101], which reports the read replica typically lags behind the writer by 20 ms or less.

6.5 Recovery

VEGITO follows a 3-way primary-backup replication of DrTM+H except for replacing one of backup/TP replicas with one backup/AP replica. During the evaluation, we kill one machine by turning off its networking, and the primary on the failed machine will be recovered by promoting its backup/TP on one of the surviving machines. We disable the primary to re-replicate a new backup/TP for emulating a rare case. Then, we kill the recovered primary again, and its backup/AP will be used to rebuild a new primary locally and migrate itself to another machine in the background.

Fig. 18 shows the timeline with OLTP throughput aggregated at 2 ms intervals, which is a zoomed-in view around the failure. VEGITO uses about 10 ms for failure detection and reconfiguration. Promoting backup/TP to primary takes about 8 ms, and rebuilding primary based on backup/AP takes 42 ms for 12 warehouses with the initial size (about 2GB). Note that the recovery load is handled by a single machine (limited by DrTM+H), causing a relatively long rebuilding time that mainly depends on the data size. Thus, it could be easily balanced across the cluster by fine-grained sharding [34, 69]. The throughput is not fully recovered since the failed machines are not back. Besides, rebuilding primary will slightly impact on throughput (10%) due to sharing CPU cores.

6.6 Parallel Log Cleaning

To study the performance impact of different log cleaning approaches, we implement three approaches on VEGITO.

- **Parallel/Inconsistent**: a fully parallel scheme used by OLTP-specific systems [26, 34], which can provide high availability but not ensuring the consistency of backups.
- **GTS+SEQ**: a global timestamp-based scheme used by prior HTAP systems [55, 65, 103, 112], which provides consistent backups by draining logs in a sequential way.
- **Parallel/Consistent**: a lightweight gossip-style scheme used by VEGITO, which also ensures the consistency of backups but drains logs in parallel.

Fig. 19 shows the throughput of OLTP and log cleaning with the increase of machines. *Parallel/Inconsistent* is used by OLTP-specific system to build fault-tolerant backups (see

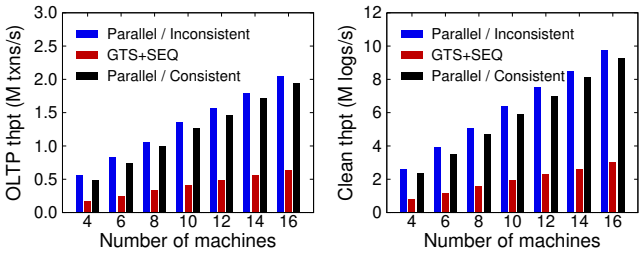


Fig. 19. *The comparison of (a) OLTP and (b) clean throughput for different log cleaning approaches using CH-benCHmark.*

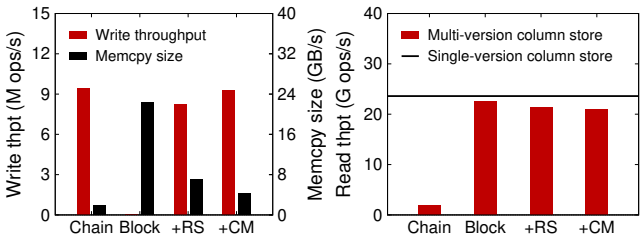


Fig. 20. *The comparison of (a) clean throughput, memory copy size, and (b) read throughput for different types of multi-version column stores.*

FTbackup in Fig. 6(b)), which are not consistent for analytical queries. *GTS+SEQ* just achieves up to 31.5% and 30.9% throughput for OLTP and log cleaning, respectively. There are two main reasons. First, assigning a global timestamp for each transaction will increase the execution time. Second, draining logs in a sequential way limits the throughput of cleaner threads and further blocks the execution of transactions. By contrast, for OLTP and log cleaning, our approach in VEGITO (*Parallel/Consistent*) only incurs about 4.5% and 4.7% slowdown compared to *Parallel/Inconsistent* and outperforms *GTS+SEQ* by up to 3.0× and 3.1×. It can drain about 9.3 million 1 KB logs per second in parallel. According to the TPC-C specification, there are 1% of accesses to a remote warehouse in NEWORDER transactions by default [95], resulting in about 9% of distributed transactions. Consequently, our gossip-style scheme only increases 7% of remote accesses due to the epoch synchronization step in the commit protocol (see Fig. 8). In the worst case, namely 100% of distributed NEWORDER transactions, our approach can still limit the performance degradation of OLTP throughput to 15% or less. The overhead of additional remote accesses increases to 21%.

6.7 Multi-version Column Store

For multi-version column store (MVCS) in VEGITO, the conventional (chain-based) approach could achieve the best performance to build the store (by cleaner threads) but the worst performance to scan the store (by AP threads). To study the effect of our locality-preserving design and optimizations, we implement four types of MVCS on VEGITO and report the steady-state throughput for them.

- **Chain**: a chain-based design [20, 68, 84].
- **Block**: a block-based design without optimizations.
- **+RS**: a block-based design with row-split optimization.
- **+CM**: a block-based design with row-split and column-merge optimizations.

As shown in Fig. 20, as expected, *Chain* can achieve the best write throughput (9.4 M ops/s), which outperforms the naive block-based design (*Block*) by 157× due to fewer memory copy operations. On the contrary, *Block* can provide about 95% of read throughput over a single-version column store, which outperforms *Chain* by about 12.4×. However, both write and read throughputs are important for HTAP systems. The row-split optimization (*+RS*) can achieve about 87.3% of write throughput of *Chain* and 95.4% of read throughput of *Block*. The column-merge optimization (*+CM*) further provides a tradeoff between two operations. It improves write throughput by 13% due to exploiting the locality of attributes updated by transactions, while reduces 2% of read throughput since one column of tuples will spread more pages.

Further, GC for the block-based design is very efficient and incurs a negligible impact on OLAP performance. It only uses one core with less than 10% of CPU utilization lasting about 70 ms (retrieve about 4.8GB), compared to 35% and 350 ms used by GC for the chain-based design.

6.8 Concurrent Index Updating

To study the performance of different tree-based indexes with concurrent read and write operations, we compare three typical data structures.

- **STX+HTM**: a generally-used C++ B$^+$-tree library [21], using hardware transactional memory (HTM) to support multiple writers and readers, as done in DBX [105].
- **Masstree** [61]: a trie-like concatenation of B$^+$-trees with cache-friendly design, using a combination of fine-grained lock and version.
- **B$^+$-tree w/ 2PU**: a standard B$^+$-tree with two-phase concurrent index updating, which is adopted by VEGITO.

We first evaluate the performance of insert operations (write-only) with the increase of worker threads using write-intensive transactions (NEWORDER) in CH-benCHmark. As shown in Fig. 21(a), *STX+HTM* does not scale with the increase of writers due to heavy contentions on node splits. *Masstree* is heavily optimized for concurrent operations by using fine-grained locks and optimistic mechanism, but it still cannot avoid contention thoroughly. For *B$^+$-tree w/ 2PU*, the insert operation is very efficient (single writer) due to using a lazy and batched manner to avoid redundant operations (node splits and data movement). Moreover, *B$^+$-tree w/ 2PU* also scales well with concurrent writers, thanks to avoiding

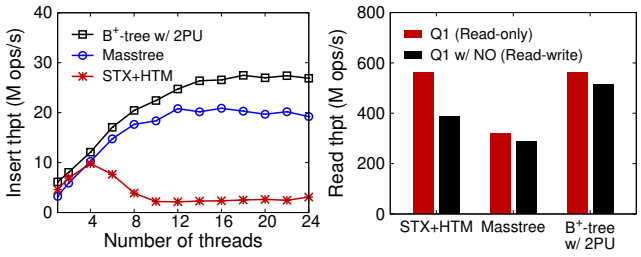


Fig. 21. *The comparison of (a) insert and (b) read throughput for different tree-based indexes.*

redundant node splits and data movements. Therefore, *B^+-tree w/ 2PU* outperforms *STX+HTM* and *Masstree* by up to 8.7× and 1.4×, respectively.

We further evaluate read performance for different indexes using Q01 from CH-benCHmark with and without NEWORDER transactions (NO), as shown in Fig. 21(b). For the read-only workload, *STX+HTM* achieves the best performance (564 M ops/s) by using more balanced tree, while *Masstree* just provides 56.6% of throughput (319 M ops/s) due to out-of-order keys in leaf nodes. For the read-write workload, the read performance in *STX+HTM* significantly decreases by 31% due to massive read-write contentions, and *Masstree* can still achieve 290 M ops/s. The read throughput of *B^+-tree w/ 2PU* achieves 563 M and 515 M ops/s for read-only and read-write workloads, respectively, which is competent for HTAP workloads.

7 RELATED WORK

HTAP systems. The increasing importance of real-time operational analytics has stimulated considerable interest in both academia and industry. There are three classes of systems.

DUAL-SYSTEM. Connecting two specialized systems is a common design alternative [56, 62, 71, 80, 82, 112]. Recently, several systems [42, 55, 65, 103] also propose to use a single node (primary) for OLTP workloads and multiple nodes (backups) for OLAP workloads, where transaction logs are shipped to backups *asynchronously*. Google F1 Lightning [112] is a loosely coupled HTAP solution (HTAP-as-a-service) that aims at providing a transparent experience to OLTP systems. TiDB [43] is a Raft-based HTAP database that *asynchronously* replicates logs from a row store (TiKV) to a column store (TiFlash). MySQL allows running analytical queries on (row-based) backups and provides semi-synchronous replication [66]. Further, many cloud databases also allow OLTP and OLAP workloads to run on different instances, which are also replicated by log shipping in the background, like Amazon Aurora [1] and MS Azure [5]. Differently, VEGITO runs analytical queries over multi-version columnar backups for *efficiency* and ships transaction updates before committing on the primary for *freshness*.

SINGLE-LAYOUT. There are several efforts aiming at building HTAP systems from one specialized system (i.e., OLTP or OLAP) [3, 13, 48, 84, 88]. HyPer [48] is an in-memory HTAP system, which leverages hardware-assisted virtual memory snapshots, session-based OLAP, and hot/cold page management [49] to maintain consistent snapshots for OLAP. AnKer [88] leverages virtual memory snapshots and adds new system calls to accelerate page copying. L-Store [84] introduces an update-friendly lineage-based data store to support both OLTP and OLAP workloads. Many SQL-on-Hadoop systems [3, 25, 31] have extended existing OLAP engines with transactional support. Using a single layout may prohibit certain optimizations (e.g., frequency compression [79]) and cause poor performance for part of workloads [14]. To avoid data contention between transactions (read-write) and analytical queries (read-only), MVCC scheme becomes essential. Prior work [53, 68] has also reported 20–45% throughput degradation due to using MVCC schemes even under low contention.

DUAL-LAYOUT. Recent systems support HTAP workloads by combining two different data layouts in a single system [4, 11, 14, 22, 54, 60, 90]. MemSQL [4] adopts an in-memory row store for OLTP workloads at scale and an on-disk column store for OLAP workloads. SAP HANA [90] stores records in either row or column format for both transactional and analytical workloads. It further uses life cycle management to ship and merge records asynchronously. SQL Server [37, 54] has added updatable columnstore indexes and batch mode processing to speed up analytical queries. Peloton [73] proposes a hybrid data layout (i.e., FSM [14]) for HTAP workloads, which stores tuples with different formats and supports online reorganization. BatchDB [60] alternates between the execution of transactions and a batch of queries (e.g., 200 ms). OLTP updates are first queued and then propagated to OLAP replicas in-between two batches of queries.

8 CONCLUSION

This paper presents VEGITO, a distributed in-memory HTAP system that retrofits high availability mechanism to meet two overarching goals simultaneously—performance (e.g., 10% performance degradation) and freshness (e.g., a maximum delay of 20 ms). Evaluations using CH-benCHmark show the efficacy of VEGITO for HTAP workloads even facing millions of concurrent transactions per second.

9 ACKNOWLEDGMENT

We sincerely thank our shepherd Dushyanth Narayanan and the anonymous reviewers for their insightful comments and feedback. This work was supported in part by the National Key Research & Development Program of China (No. 2020YFB2104100), the National Natural Science Foundation of China (No. 61772335, 61925206), and the High-Tech Support Program from Shanghai Committee of Science and Technology (No. 19511121100). Corresponding author: Rong Chen (rongchen@sjtu.edu.cn).

REFERENCES

[1] Amazon Aurora FAQs: High Availability and Replication. https://aws.amazon.com/rds/aurora/faqs/.

[2] AWS Glue. https://aws.amazon.com/glue/.

[3] Hive Transactions. https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.3.0/bk_dataintegration/content/hive-013-feature-transactions.html.

[4] MemSQL. http://memsql.com/.

[5] Microsoft Azure. https://docs.microsoft.com/en-us/azure/.

[6] MonetDB. http://www.monetdb.org/.

[7] SQL Server 2019. https://www.microsoft.com/en-us/sql-server/sql-server-2019.

[8] TiDB. https://pingcap.com/.

[9] TiFlash Overview. https://pingcap.com/docs/stable/reference/tiflash/overview/.

[10] AGRAWAL, N., AND VULIMIRI, A. Low-Latency Analytics on Colossal Data Streams with SummaryStore. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), SOSP '17, p. 647–664.

[11] ALAGIANNIS, I., IDREOS, S., AND AILAMAKI, A. H2o: A hands-free adaptive store. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (2014), SIGMOD '14, p. 1103–1114.

[12] ALIBABA CLOUD. Double 11 Real-Time Monitoring System with Time Series Database. https://www.alibabacloud.com/blog/594855, 2019.

[13] APPUSWAMY, R., KARPATHIOTAKIS, M., PAROBIC, D., AND AILAMAKI, A. The case for heterogeneous HTAP. In *8th Biennial Conference on Innovative Data Systems Research* (2017), CIDR '17.

[14] ARULRAJ, J., PAVLO, A., AND MENON, P. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *Proceedings of the 2016 International Conference on Management of Data* (2016), pp. 583–598.

[15] BALAKRISHNAN, M., MALKHI, D., WOBBER, T., WU, M., PRABHAKARAN, V., WEI, M., DAVIS, J. D., RAO, S., ZOU, T., AND ZUCK, A. Tango: Distributed Data Structures over A Shared Log. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), pp. 325–340.

[16] BARBER, R., GARCIA-ARELLANO, C., GROSMAN, R., LOHMAN, G., MOHAN, C., MULLER, R., PIRAHESH, H., RAMAN, V., SIDLE, R., STORM, A., ET AL. WiSer: A Highly Available HTAP DBMS for IoT Applications. In *2019 IEEE International Conference on Big Data (Big Data)* (2019).

[17] BENDER, M. A., FARACH-COLTON, M., JANNEN, W., JOHNSON, R., KUSZMAUL, B. C., PORTER, D. E., YUAN, J., AND ZHAN, Y. An Introduction to Bε-trees and Write-Optimization. *login; magazine 40*, 5 (2015), 22–28.

[18] BERNSTEIN, P. A., AND GOODMAN, N. Concurrency Control in Distributed Database Systems. *ACM Comput. Surv. 13*, 2 (June 1981), 185–221.

[19] BERNSTEIN, P. A., AND GOODMAN, N. Multiversion Concurrency Control Theory and Algorithms. *ACM Trans. Database Syst. 8*, 4 (Dec. 1983), 465–483.

[20] BESTA, M., AND HOEFLER, T. Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing* (2015), HPDC'15, pp. 161–172.

[21] BINGMANN, T. STX B+ Tree C++ Template Classes. https://panthema.net/2007/stx-btree/, 2013.

[22] BOISSIER, M. Reducing the Footprint of Main Memory HTAP Systems: Removing, Compressing, Tiering, and Ignoring Data. In *Proceedings of the VLDB 2018 PhD Workshop* (2018).

[23] BONCZ, P. A., ZUKOWSKI, M., AND NES, N. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proceedings of the 2nd Conference on Innovative Data Systems Research* (2005), vol. 5 of *CIDR '05*, pp. 225–237.

[24] CAO, S., YANG, X., CHEN, C., ZHOU, J., LI, X., AND QI, Y. TitAnt: Online Real-Time Transaction Fraud Detection in Ant Financial. *Proceedings of the VLDB Endowment 12*, 12 (Aug. 2019), 2082—-2093.

[25] CAO, Y., CHEN, C., GUO, F., JIANG, D., LIN, Y., OOI, B., VO, H., WU, S., AND XU, Q. ES2: A cloud data storage system for supporting both OLTP and OLAP. pp. 291–302.

[26] CHEN, Y., WEI, X., SHI, J., CHEN, R., AND CHEN, H. Fast and General Distributed Transactions using RDMA and HTM. In *Proceedings of the European Conference on Computer Systems* (2016), EuroSys'16, p. 26.

[27] CHISHOLM, S. Adopting medical technologies and diagnostics recommended by NICE: The Health Technologies Adoption Programme, 2014.

[28] COELHO, F., PAULO, J. A., VILAÇA, R., PEREIRA, J., AND OLIVEIRA, R. HTAPBench: Hybrid Transactional and Analytical Processing Benchmark. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering* (2017), ICPE '17, pp. 293—-304.

[29] COLE, R., FUNKE, F., GIAKOUMAKIS, L., GUY, W., KEMPER, A., KROMPASS, S., KUNO, H., NAMBIAR, R., NEUMANN, T., POESS, M., ET AL. The mixed workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems* (2011), p. 8.

[30] CORBETT, J. C., DEAN, J., EPSTEIN, M., FIKES, A., FROST, C., FURMAN, J. J., GHEMAWAT, S., GUBAREV, A., HEISER, C., HOCHSCHILD, P., HSIEH, W., KANTHAK, S., KOGAN, E., LI, H., LLOYD, A., MELNIK, S., MWAURA, D., NAGLE, D., QUINLAN, S., RAO, R., ROLIG, L., SAITO, Y., SZYMANIAK, M., TAYLOR, C., WANG, R., AND WOODFORD, D. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst. 31*, 3 (Aug. 2013).

[31] COSTEA, A., IONESCU, A., RĂDUCANU, B., SWITAKOWSKI, M., BÂRCA, C., SOMPOLSKI, J., UNDEFINEDUSZCZAK, A., SZAFRAUNDEFINEDSKI, M., DE NIJS, G., AND BONCZ, P. VectorH: Taking SQL-on-Hadoop to the Next Level. In *Proceedings of the 2016 International Conference on Management of Data* (2016), SIGMOD '16, pp. 1105—-1117.

[32] DAGEVILLE, B., CRUANES, T., ZUKOWSKI, M., ANTONOV, V., AVANES, A., BOCK, J., CLAYBAUGH, J., ENGOVATOV, D., HENTSCHEL, M., HUANG, J., LEE, A. W., MOTIVALA, A., MUNIR, A. Q., PELLEY, S., POVINEC, P., RAHN, G., TRIANTAFYLLIS, S., AND UNTERBRUNNER, P. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (2016), SIGMOD '16, p. 215–226.

[33] DIACONU, C., FREEDMAN, C., ISMERT, E., LARSON, P.-A., MITTAL, P., STONECIPHER, R., VERMA, N., AND ZWILLING, M. Hekaton: SQL Server's Memory-optimized OLTP Engine. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013), SIGMOD'13, pp. 1243–1254.

[34] DRAGOJEVIĆ, A., NARAYANAN, D., NIGHTINGALE, E. B., RENZELMANN, M., SHAMIS, A., BADAM, A., AND CASTRO, M. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP'15, pp. 54–70.

[35] DRESELER, M., BOISSIER, M., RABL, T., AND UFLACKER, M. Quantifying TPC-H choke points and their optimizations. *Proceedings of the VLDB Endowment 13*, 10 (2020), 1206–1220.

[36] DRESELER, M., KOSSMANN, J., BOISSIER, M., KLAUCK, S., UFLACKER, M., AND PLATTNER, H. Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. In *Proceedings of the 22nd International Conference on Extending Database Technology* (2019), pp. 313–324.

[37] DZIEDZIC, A., WANG, J., DAS, S., DING, B., NARASAYYA, V. R., AND SYAMALA, M. Column-store and B+ tree - Are Hybrid Physical Designs Important? In *Proceedings of the 2018 International Conference on Management of Data* (2018), pp. 177–190.

[38] ESSERTEL, G., TAHBOUB, R., DECKER, J., BROWN, K., OLUKOTUN, K., AND ROMPF, T. Flare: Optimizing Apache Spark with Native Compilation for Scale-up Architectures and Medium-Size Data. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (2018), pp. 799–815.

[39] GU, J., YU, Q., WANG, X., WANG, Z., ZANG, B., GUAN, H., AND CHEN, H. Pisces: A Scalable and Efficient Persistent Transactional Memory. In *Proceedings of 2019 USENIX Annual Technical Conference* (2019), pp. 913–928.

[40] GUPTA, A., AGARWAL, D., TAN, D., KULESZA, J., PATHAK, R., STEFANI, S., AND SRINIVASAN, V. Amazon redshift and the case for simpler data warehouses. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (2015), pp. 1917–1923.

[41] HERLIHY, M., AND MOSS, J. E. B. Transactional Memory: Architectural Support for Lock-free Data Structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture* (1993), ISCA'93, pp. 289–300.

[42] HONG, C., ZHOU, D., YANG, M., KUO, C., ZHANG, L., AND ZHOU, L. KuaFu: Closing the parallelism gap in database replication. In *Proceedings of 2013 IEEE 29th International Conference on Data Engineering* (2013), pp. 1186–1195.

[43] HUANG, D., LIU, Q., CUI, Q., FANG, Z., MA, X., XU, F., SHEN, L., TANG, L., ZHOU, Y., HUANG, M., WEI, W., LIU, C., ZHANG, J., LI, J., WU, X., SONG, L., SUN, R., YU, S., ZHAO, L., CAMERON, N., PEI, L., AND TANG, X. TiDB: A Raft-Based HTAP Database. *Proc. VLDB Endow. 13*, 12 (Aug. 2020), 3072–3084.

[44] IDREOS, S., GROFFEN, F., NES, N., MANEGOLD, S., MULLENDER, K. S., AND KERSTEN, M. MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Eng. Bull. 35* (2012), 40–45.

[45] JOHN PIEKOS. Measuring real-time. https://www.infoworld.com/article/3220430/measuring-real-time.html, 2017.

[46] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (2016), OSDI'16, pp. 185–201.

[47] KAUFMANN, M., MANJILI, A. A., VAGENAS, P., FISCHER, P. M., KOSSMANN, D., FÄRBER, F., AND MAY, N. Timeline Index: A Unified Data Structure for Processing Queries on Temporal Data in SAP HANA. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013), pp. 1173–1184.

[48] KEMPER, A., AND NEUMANN, T. HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *Proceedings of 2011 IEEE 27th International Conference on Data Engineering* (2011), pp. 195–206.

[49] KEMPER, A., NEUMANN, T., FUNKE, F., LEIS, V., AND MÜHE, H. HyPer: Adapting Columnar Main-Memory Data Management for Transactional AND Query Processing. *IEEE Data Eng. Bull. 35*, 1 (2012), 46–51.

[50] KIM, K., WANG, T., JOHNSON, R., AND PANDIS, I. Ermia: Fast Memory-optimized Database System for Heterogeneous Workloads. In *Proceedings of the 2016 International Conference on Management of Data* (2016), pp. 1675–1687.

[51] KUNG, H. T., AND ROBINSON, J. T. On Optimistic Methods for Concurrency Control. *ACM Trans. Database Syst. 6*, 2 (June 1981), 213–226.

[52] LAMPORT, L., MALKHI, D., AND ZHOU, L. Vertical Paxos and Primary-backup Replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing* (2009), PODC'09, pp. 312–313.

[53] LARSON, P.-Å., BLANAS, S., DIACONU, C., FREEDMAN, C., PATEL, J. M., AND ZWILLING, M. High-Performance Concurrency Control Mechanisms for Main-Memory Databases. *Proc. VLDB Endow. 5*, 4 (2011), 298–309.

[54] LARSON, P.-R., BIRKA, A., HANSON, E. N., HUANG, W., NOWAKIEWICZ, M., AND PAPADIMOS, V. Real-Time Analytical Processing with SQL Server. *Proc. VLDB Endow. 8*, 12 (Aug. 2015), 1740–1751.

[55] LEE, J., MOON, S., KIM, K. H., KIM, D. H., CHA, S. K., AND HAN, W.-S. Parallel Replication Across Formats in SAP HANA for Scaling out Mixed OLTP/OLAP Workloads. *Proc. VLDB Endow. 10*, 12 (Aug. 2017), 1598–1609.

[56] LI, F., ÖZSU, M. T., CHEN, G., AND OOI, B. C. R-store: A Scalable Distributed System for Supporting Real-time Analytics. In *Proceedings of the 2014 IEEE 30th International Conference on Data Engineering* (2014), pp. 40–51.

[57] LI, J., MICHAEL, E., AND PORTS, D. R. K. Eris: Coordination-Free Consistent Transactions Using In-Network Concurrency Control. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), SOSP '17, pp. 104–120.

[58] LIM, H., KAMINSKY, M., AND ANDERSEN, D. G. Cicada: Dependably Fast Multi-core In-memory Transactions. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), pp. 21–35.

[59] LOCKERMAN, J., FALEIRO, J. M., KIM, J., SANKARAN, S., ABADI, D. J., ASPNES, J., SEN, S., AND BALAKRISHNAN, M. The FuzzyLog: A Partially Ordered Shared Log. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (2018), OSDI'18, pp. 357—-372.

[60] MAKRESHANSKI, D., GICEVA, J., BARTHELS, C., AND ALONSO, G. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), pp. 37–50.

[61] MAO, Y., KOHLER, E., AND MORRIS, R. T. Cache Craftiness for Fast Multicore Key-value Storage. In *Proceedings of the 7th ACM European Conference on Computer Systems* (2012), EuroSys'12, pp. 183–196.

[62] MARTIN, D., KOETH, O., KERN, J., AND IVANOVA, I. Near Real-Time Analytics with IBM DB2 Analytics Accelerator. In *Proceedings of the 16th International Conference on Extending Database Technology* (2013), EDBT '13, pp. 579–588.

[63] MARY SHACKLETT. See real-time big data analytics in milliseconds with IMDG technology. https://www.techrepublic.com/article/see-real-time-big-data-analytics-in-milliseconds-with-imdg-technology/, 2014.

[64] MCKENNEY, P. E., AND SLINGWINE, J. D. Read-Copy Update: Using Execution History to Solve Concurrency Problems. In *Parallel and Distributed Computing and Systems* (1998), pp. 509–518.

[65] MÜHLBAUER, T., RÖDIGER, W., REISER, A., KEMPER, A., AND NEUMANN, T. ScyPer: Elastic OLAP Throughput on Transactional Data. In *Proceedings of the Second Workshop on Data Analytics in the Cloud* (2013), DanaC '13, p. 11–15.

[66] MYSQL. MySQL 8.0 Reference Manual: Chapter 17 Replication. https://dev.mysql.com/doc/refman/8.0/en/replication-semisync.html.

[67] NEUMANN, T., AND FREITAG, M. J. Umbra: A Disk-Based System with In-Memory Performance. In *Proceedings of the 10th Conference on Innovative Data Systems Research* (2020), CIDR '20.

[68] NEUMANN, T., MÜHLBAUER, T., AND KEMPER, A. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015), SIGMOD '15, pp. 677–689.

[69] ONGARO, D., RUMBLE, S. M., STUTSMAN, R., OUSTERHOUT, J., AND ROSENBLUM, M. Fast Crash Recovery in RAMCloud. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), pp. 29–41.

[70] ÖZCAN, F., TIAN, Y., AND TÖZÜN, P. Hybrid Transactional/Analytical Processing: A Survey. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), pp. 1771–1775.

[71] PAREEK, A., KHALADKAR, B., SEN, R., ONAT, B., NADIMPALLI, V., AGARWAL, M., AND KEENE, N. Striim: A Streaming Analytics Platform for Real-time Business Decisions. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics* (2017), BIRTE '17, pp. 4:1–4:8.

[72] PAREEK, A., KHALADKAR, B., SEN, R., ONAT, B., NADIMPALLI, V., AND LAKSHMINARAYANAN, M. Real-time ETL in Striim. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics* (2018), BIRTE '18, pp. 3:1–3:10.

[73] PAVLO, A., ANGULO, G., ARULRAJ, J., LIN, H., LIN, J., MA, L., MENON, P., MOWRY, T. C., PERRON, M., QUAH, I., ET AL. Self-Driving Database Management Systems. In *Proceedings of the 8th Conference on Innovative Data Systems Research* (2017), CIDR '17.

[74] PELKONEN, T., FRANKLIN, S., TELLER, J., CAVALLARO, P., HUANG, Q., MEZA, J., AND VEERARAGHAVAN, K. Gorilla: A Fast, Scalable, in-Memory Time Series Database. *Proc. VLDB Endow. 8*, 12 (Aug. 2015), 1816–1827.

[75] PEZZINI, M., FEINBERG, D., RAYNER, N., AND EDJLALI, R. Hybrid Transaction/Analytical Processing Will Foster Opportunities for Dramatic Business Innovation. *Gartner* (2014).

[76] PUBNUB. How Fast is Realtime? Human Perception and Technology. https://www.pubnub.com/blog/how-fast-is-realtime-human-perception-and-technology/, 2015.

[77] QIU, X., CEN, W., QIAN, Z., PENG, Y., ZHANG, Y., LIN, X., AND ZHOU, J. Real-Time Constrained Cycle Detection in Large Dynamic Graphs. *Proc. VLDB Endow. 11*, 12 (2018), 1876–1888.

[78] QUAH, J. T., AND SRIGANESH, M. Real-time Credit Card Fraud Detection Using Computational Intelligence. *Expert systems with applications 35*, 4 (2008), 1721–1732.

[79] RAMAN, V., ATTALURI, G., BARBER, R., CHAINANI, N., KALMUK, D., KULANDAISAMY, V., LEENSTRA, J., LIGHTSTONE, S., LIU, S., LOHMAN, G. M., ET AL. DB2 with BLU Acceleration: So Much More than Just a Column Store. *Proc. VLDB Endow. 6*, 11 (2013), 1080–1091.

[80] RAMNARAYAN, J., MOZAFARI, B., WALE, S., MENON, S., KUMAR, N., BHANAWAT, H., CHAKRABORTY, S., MAHAJAN, Y., MISHRA, R., AND BACHHAV, K. SnappyData: A Hybrid Transactional Analytical Store Built On Spark. In *Proceedings of the 2016 International Conference on Management of Data* (2016), SIGMOD '16, pp. 2153—-2156.

[81] RANGER, C., RAGHURAMAN, R., PENMETSA, A., BRADSKI, G., AND KOZYRAKIS, C. Evaluating Mapreduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture* (2007), pp. 13–24.

[82] RAZA, A., CHRYSOGELOS, P., ANADIOTIS, A. C., AND AILAMAKI, A. Adaptive HTAP through Elastic Resource Scheduling. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020), SIGMOD '20, pp. 2043—2054.

[83] RÖDIGER, W., MÜHLBAUER, T., KEMPER, A., AND NEUMANN, T. High-speed Query Processing over High-speed Networks. *Proc. VLDB Endow. 9*, 4 (2015), 228–239.

[84] SADOGHI, M., BHATTACHERJEE, S., BHATTACHARJEE, B., AND CANIM, M. L-Store: A Real-time OLTP and OLAP System. In *Proceedings of the 21th International Conference on Extending Database Technology* (2018), EBDT '18.

[85] SAHAY, B., AND RANJAN, J. Real Time Business Intelligence in Supply Chain Analytics. *Information Management & Computer Security 16*, 1 (2008), 28–48.

[86] SEWALL, J., CHHUGANI, J., KIM, C., SATISH, N., AND DUBEY, P. PALM: Parallel Architecture-friendly Latch-free Modifications to B+ trees on Many-core Processors. *Proc. VLDB Endowment 4*, 11 (2011), 795–806.

[87] SHAMIS, A., RENZELMANN, M., NOVAKOVIC, S., CHATZOPOULOS, G., DRAGOJEVIĆ, A., NARAYANAN, D., AND CASTRO, M. Fast General Distributed Transactions with Opacity. In *Proceedings of the 2019 International Conference on Management of Data* (2019), SIGMOD '19, p. 433–448.

[88] SHARMA, A., SCHUHKNECHT, F. M., AND DITTRICH, J. Accelerating Analytical Processing in MVCC using Fine-Granular High-Frequency Virtual Snapshotting. In *Proceedings of the 2018 International Conference on Management of Data* (2018), pp. 245–258.

[89] SHI, J., YAO, Y., CHEN, R., CHEN, H., AND LI, F. Fast and Concurrent RDF Queries with RDMA-based Distributed Graph Exploration. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (2016), OSDI '16, pp. 317–332.

[90] SIKKA, V., FÄRBER, F., LEHNER, W., CHA, S. K., PEH, T., AND BORNHÖVD, C. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), SIGMOD '12, pp. 731–742.

[91] SOMPOLSKI, J., ZUKOWSKI, M., AND BONCZ, P. Vectorization vs. Compilation in Query Execution. In *Proceedings of the Seventh International Workshop on Data Management on New Hardware* (2011), pp. 33–40.

[92] STEVE ABRAHAM. Creating a proof of concept using Amazon Aurora. https://aws.amazon.com/blogs/database/creating-a-proof-of-concept-using-amazon-aurora/, 2019.

[93] TA, V.-D., LIU, C.-M., AND NKABINDE, G. W. Big Data Stream Computing in Healthcare Real-time Analytics. In *Proceedings of the 2016 IEEE International Conference on Cloud Computing and Big Data Analysis* (2016), pp. 37–42.

[94] TAI, A., WEI, M., FREEDMAN, M. J., ABRAHAM, I., AND MALKHI, D. Replex: A Scalable, Highly Available Multi-index Data Store. In *Proceedings of the 2016 USENIX Annual Technical Conference* (2016), pp. 337–350.

[95] THE TRANSACTION PROCESSING COUNCIL. TPC-C Benchmark V5.11. http://www.tpc.org/tpcc/.

[96] THE TRANSACTION PROCESSING COUNCIL. TPC-H Benchmark V2.17.3. http://www.tpc.org/tpch/.

[97] THOMSON, A., DIAMOND, T., WENG, S.-C., REN, K., SHAO, P., AND ABADI, D. J. Calvin: Fast Distributed Transactions for Partitioned Database Systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), SIGMOD'12, pp. 1–12.

[98] TU, S., ZHENG, W., KOHLER, E., LISKOV, B., AND MADDEN, S. Speedy Transactions in Multicore In-memory Databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), SOSP'13.

[99] VAN RENESSE, R., AND SCHNEIDER, F. B. Chain Replication for Supporting High Throughput and Availability. In *Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation* (2004), vol. 4 of *OSDI '04*, pp. 91–104.

[100] VASSILIADIS, P., AND SIMITSIS, A. Near real time ETL. In *New Trends in Data Warehousing and Data Analysis*. 2009, pp. 1–31.

[101] VERBITSKI, A., GUPTA, A., SAHA, D., BRAHMADESAM, M., GUPTA, K., MITTAL, R., KRISHNAMURTHY, S., MAURICE, S., KHARATISHVILI, T., AND BAO, X. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), SIGMOD '17, p. 1041–1052.

[102] WAN, Y. S. G. E. B., LIM, S., AND PAVLO, A. On Supporting Efficient Snapshot Isolation for Hybrid Workloads with Multi-Versioned Indexes. *Proc. VLDB Endow. 13*, 2 (2019).

[103] WANG, T., JOHNSON, R., AND PANDIS, I. Query Fresh: Log Shipping on Steroids. *Proc. VLDB Endow. 11*, 4 (2017), 406–419.

[104] WANG, X., ZHANG, W., WANG, Z., WEI, Z., CHEN, H., AND ZHAO, W. Eunomia: Scaling Concurrent Search Trees under Contention Using HTM. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2017), PPoPP '17, p. 385–399.

[105] WANG, Z., QIAN, H., LI, J., AND CHEN, H. Using Restricted Transactional Memory to Build a Scalable In-memory Database. In *Proceedings of the Ninth European Conference on Computer Systems* (2014), EuroSys'14, pp. 26:1–26:15.

[106] WEI, X., CHEN, R., CHEN, H., WANG, Z., GONG, Z., AND ZANG, B. Unifying Timestamp with Transaction Ordering for MVCC with Decentralized Scalar Timestamp. In *Proceedings of 18th USENIX Symposium on Networked Systems Design and Implementation* (Apr. 2021).

[107] WEI, X., DONG, Z., CHEN, R., AND CHEN, H. Deconstructing RDMA-enabled Distributed Transactions: Hybrid is Better! In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation* (2018), OSDI '18, pp. 233–251.

[108] WEI, X., SHEN, S., CHEN, R., AND CHEN, H. Replication-driven Live Reconfiguration for Fast Distributed Transaction Processing. In *Proceedings of the 2017 USENIX Annual Technical Conference* (2017), USENIX ATC '17, pp. 335–347.

[109] WEI, X., SHI, J., CHEN, Y., CHEN, R., AND CHEN, H. Fast In-memory Transaction Processing Using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), SOSP'15, pp. 87–104.

[110] WU, Y., ARULRAJ, J., LIN, J., XIAN, R., AND PAVLO, A. An Empirical Evaluation of In-memory Multi-version Concurrency Control. *Proc. VLDB Endow. 10*, 7 (Mar. 2017), 781–792.

[111] XIE, X., WEI, X., CHEN, R., AND CHEN, H. Pragh: Locality-preserving Graph Traversal with Split Live Migration. In *Proceedings of the 2019 USENIX Annual Technical Conference* (2019), USENIX ATC '19, pp. 723–738.

[112] YANG, J., RAE, I., XU, J., SHUTE, J., YUAN, Z., LAU, K., ZENG, Q., ZHAO, X., MA, J., CHEN, Z., GAO, Y., DONG, Q., ZHOU, J., WOOD, J., GRAEFE, G., NAUGHTON, J., AND CIESLEWICZ, J. F1 Lightning: HTAP as a Service. *Proc. VLDB Endow. 13*, 12 (Aug. 2020), 3313–3325.

[113] ZAMANIAN, E., BINNIG, C., HARRIS, T., AND KRASKA, T. The End of a Myth: Distributed Transactions Can Scale. *Proc. VLDB Endow. 10*, 6 (Feb. 2017), 685–696.

[114] ZHANG, H., ANDERSEN, D. G., PAVLO, A., KAMINSKY, M., MA, L., AND SHEN, R. Reducing the Storage Overhead of Main-memory OLTP Databases with Hybrid Indexes. In *Proceedings of the 2016 International Conference on Management of Data* (2016), pp. 1567–1581.

[115] ZHANG, Y., CHEN, R., AND CHEN, H. Sub-Millisecond Stateful Stream Querying over Fast-Evolving Linked Data. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), SOSP '17, p. 614–630.

[116] ZHENG, W., TU, S., KOHLER, E., AND LISKOV, B. Fast Databases with Fast Durability and Recovery Through Multicore Parallelism. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (2014), OSDI'14, pp. 465–477.

[117] ZHOU, J., LI, X., ZHAO, P., CHEN, C., LI, L., YANG, X., CUI, Q., YU, J., CHEN, X., DING, Y., AND QI, Y. A. KunPeng: Parameter Server Based Distributed Learning Systems and Its Applications in Alibaba and Ant Financial. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), KDD '17, pp. 1693–1702.

A ARTIFACT APPENDIX

Abstract

This artifact provides the prototype of VEGITO, including the document, source code and scripts to execute the main experiments and reproduce the experimental results. VEGITO is a fast distributed in-memory HTAP system, which retrofits high availability mechanism to tame hybrid transaction/analytical processing. An open-source version of VEGITO is available at https://github.com/SJTU-IPADS/vegito.

Scope

This artifact (including the document, source code and scripts) is used for artifact evaluation, which can reproduce the main experimental results in VEGITO. To use VEGITO in your research, we recommend using the `master` branch of the public repository, which would be maintained by members of the Institute of Parallel and Distributed Systems.

Contents

- **README and document:** A detailed description of the artifacts, including the steps of environment building, installation, usage of scripts and configuration files, and how to conduct experiments. Please read the `README.md` at first.
- **Source code:** We provide the prototype of VEGITO with the HTAP benchmark called CH-benCHmark and some micro-benchmarks.
- **Configuration files:** We record different configurations in some XML format files. The detailed format is described in the `README.md`.
- **Scripts:** We run the VEGITO by using the Python scripts and Shell scripts. These scripts use SSH for cluster deployment and management.

Hosting

- **Program:** `vegito`.
- **Compilation:** `g++` and `cmake`.
- **Hardware:** Intel CPU with RTM and Mellanox NIC with RDMA.
- **Execution:** Python scripts, Shell scripts, SSH.
- **Metrics:** Throughput, latency, and time lag (freshness).
- **Public link:** https://github.com/SJTU-IPADS/vegito.
- **Code licenses:** Apache License 2.0.

Requirements

Hardware Dependencies. At least three machines are used to reproduce the experimental results for distributed configurations. Each machine must have:

- **CPU:** Intel processors with 2 sockets and Restricted Transactional Memory (RTM) (e.g., Xeon E5-2650 v4).
- **NIC:** At least one (two is better) Mellanox RDMA network card (e.g., Mellanox ConnectX-4 100Gbps InfiniBand NIC).

Software Dependencies.

- **Operating system:** `Ubuntu` $\geq$ `16.04`.
- **Compile toolchain:** `g++` $\geq$ `5.4.4` and `cmake` $\geq$ `3.5.1`.
- **Libraries:** `Mellanox OFED`, `boost 1.61.0`, `ssmalloc`.

AE Methodology

Submission, reviewing and badging methodology is specified at https://www.usenix.org/conference/osdi21/call-for-artifacts.