Accelerating Cloud-Native Databases with Distributed PMem Stores

Jason Sun[1], Haoxiang Ma, Li Zhang, Huicong Liu, *Haiyang Shi, *Shangyu Luo, *Kai Wu[2], Kevin Bruhwiler[3], Cheng Zhu, Yuanyuan Nie, *Jianjun Chen, Lei Zhang, Yuming Liang
ByteDance US Infrastructure System Lab; ByteDance Inc.

***Abstract*—Relational databases have gone through a phase of architectural transition from a monolithic to a distributed architecture to take full advantage of cloud technology. These distributed databases can leverage remote storage to maintain larger amounts of data than monolithic databases at the cost of increased latency. At ByteDance, we have built a distributed database called veDB based on the popular compute-storage separation architecture, however we have observed the system is unable to provide both low latency and high throughput required by some business critical applications, such as batched order processing.**

In this paper we present our novel approaches to tackle this problem. We have modified our system's storage to utilize persistent memory (PMem) coupled with a remote direct memory access (RDMA) network to reduce read/write latency and increase the throughput. We also propose a query push-down framework to push partial computations to the PMem storage layer to accelerate analytical queries and reduce the impact of the transaction workload in the computation layer. Our experiments show that our methods improve the throughput by up to 1.5× and reduce latency by up to 20× for standard benchmarks and real-world applications.

***Index Terms*—distributed database, cloud-native database, PMem, RDMA, cloud computing, query push-down**

I. Introduction

ByteDance uses RDBMS (e.g., MySQL) internally for storing and querying data for various business applications, such as managing user account information, processing purchase orders, and tracking shipping and logistics. These applications can be divided roughly into two categories: applications that manage the display and content of the business—which are very read-heavy and write-light, and applications that keep track of user actions and business transactions—which are write-heavy. In recent years, the volume of data produced by those applications has grown so quickly that it is more and more difficult for MySQL databases to handle it properly. Therefore, we built our scalable cloud-native database, *Volcano Engine Database (veDB)* [1], and migrated 30% of our existing MySQL deployment to it.

The transition to veDB alleviated the concern of storage capacity as veDB uses distributed storage. However, it brought up other issues: a high transaction latency fluctuation due to network I/O and a high query latency when scanning large amounts of data. As veDB separates computing and storage, those latency issues are conspicuous. There are two reasons for this:

1) veDB has a high log write latency due to its remote data access. Its I/O path requires the participation of a remote binary large object (blob) store, which introduces latency from thread scheduling and contention.
2) veDB takes longer to read pages from the remote storage compared to MySQL's page reading from local disks. This results in a long processing time for large data read requests.

Such read/write latency issues are common for cloud-native and compute-storage disaggregated systems [2], and resolving them is challenging. Many works have studied how to mitigate this problem [3]–[7]. However, most of them focus on reducing read latency and do not pay much attention to write latency. Using SSD-based storage would alleviate latency issues of data read [8], but it is still unable to meet our applications' requirements for fast data write. To improve both read and write performance we explored two key technologies: persistent memory (PMem) [9] and remote direct memory access (RDMA). PMem brings lower latency for both reads and writes compared to SSD devices [10], and RDMA allows accessing remote PMem directly without involving remote processors. In addition, the kernel-bypass provided by RDMA can reduce the kernel overheads such as scheduling overheads and contention issues.

In this paper, we describe how to use a distributed PMem store coupled with RDMA, called *AStore*, to improve the performance of a cloud-native database and leverage query push-down to AStore for fast analytical query execution. We have the following key contributions:

1) We build a low latency PMem storage engine and employ one-sided RDMA to provide efficient access to it. Our design can ensure read-write consistency while data is concurrently accessed by one-sided RDMA verbs.
2) Our RDMA-accelerated AStore can reduce log write and transaction latency, increase transaction throughput, and maintain latency fluctuation within a narrow range.
3) Our RDMA-accelerated AStore is also used to create an

1. Jason Sun is the corresponding author, jason.sun@bytedance.com
2. Kai Wu is now with Microsoft Corporation, wukai@microsoft.com
3. Kevin Bruhwiler is currently a Ph.D student at University of California at Irvine. Work was done while he did an internship at ByteDance. kevin.bruhwiler@gmail.com

extended buffer pool (EBP). The extended buffer pool can cache hot pages and reduce page read time, which is especially useful when reading large amounts of data.
4) We propose a query push-down framework that can leverage the RDMA-accelerated AStore to conduct partial computations in *both* the extended buffer pool and storage to reduce the overall query latency.
5) We conduct a set of experiments to show our design's effectiveness on both standard benchmarks and our internal workloads. The experiments on our real workloads proved that our design could reduce our system's latency by up to 20× and is applicable to different products.

The remainder of this paper is organized as follows. Section II reviews related work. Section III gives an overview of veDB's architecture. Section IV describes our design of RDMA-accelerated AStore. Section V presents how AStore and RDMA reduce write and read latency and increase the throughput. Section VI provides our design of a query push-down framework. Section VII presents an experimental evaluation of our design. Finally, Section VIII summarizes the paper.

II. Related Work

A. Cloud Database Systems

Relational databases are an essential building block in modern information technology infrastructure, and cloud vendors have invested significant efforts to grow their relational database service (RDS) business. Some examples are Amazon Aurora [11], Microsoft Socrates [12], and Huawei Taurus [13]. These systems use an architecture based on the log-is-database principle and offload REDO processing to a fault-tolerant, self-healing, and multi-tenant scale-out storage service. They also eliminate the need for checkpoints in the compute layer. veDB shares many features with those databases, including an append-only log-is-database architecture, separated compute and storage layers, and replicated remote storage. veDB also proposes its own design for some components, such as replication protocol and page reconstruction from REDO logs.

Computation push-down is a common technique to mitigate network bottlenecks in database systems with disaggregated storage [11], [14] and systems that support federation [15]–[17]. Amazon Aurora [11] offers a feature called parallel query, which takes advantage of Aurora's unique architecture to push down and parallelize query processing across thousands of CPUs in the the storage layer. Microsoft Polybase [15] translates SQL operators on HDFS-resident data into MapReduce jobs and then executes them on the Hadoop cluster. The feature has been supported in SQL Server since 2016. PushdownDB [18] studied the effectiveness of pushing parts of DBMS analytics queries into the Simple Storage Service (S3) engine of Amazon Web Services (AWS). FlexPushdownDB (FPDB) [19] is an OLAP cloud DBMS prototype that supports fine-grained hybrid query execution to combine the benefits of caching and computation push-down in a storage-disaggregation architecture. veDB has a fine-grained hybrid query execution guided by a similar design principle as FPDB yet supports the capability of push-down query that is similar to Aurora's parallel query. Compared with FPDB and Aurora, veDB leverages the computing power of AStore servers and the data pages resided on PMem – instead of those in the persistent storage layer – to support in-memory processing and thus deliver high performance.

B. Hardware-Accelerated Database Systems

Non-Volatile Memory. Recent years have seen a proliferation of studies on the use of non-volatile memory techniques. [20]–[23], [23]–[34] propose hybrid storage architectures to balance performance and cost. [35]–[37] introduce new transaction schemes for managing transactional updates to data structures designed for PMem. Additionally, there has been research on persistent NVM indexes, such as B+-Trees [38]–[40], hash tables [41]–[44], range indexes [45], learned indexes [46], and high-performance join algorithms [47]. These studies are complementary to the design of veDB, and the system can also adopt PMem-optimized transaction schemes or indexes to enhance performance. In this paper, we focus on building efficient and effective disaggregated PMem stores to accelerate a cloud-native database system.

The database community has been evaluating the use of Optane PMem in database workloads as it becomes available. Wu *et al.* [48] have run TPC-H and TPC-C benchmarks on an SQL Server utilizing PMem and have demonstrated the benefits of PMem. Benson *et al.* [49], [50] have also explored the best practices for using PMem in OLAP workloads and examined if PMem can replace NVMe SSDs. Similarly, Shanbhag *et al.* [51] have evaluated OLAP workloads using Optane PMem in AppDirect mode. Koutsoukos *et al.* [20] have conducted a comprehensive analysis of existing relational database engines, such as MySQL and Postgres, under different PMem configurations. Their evaluation results have suggested that placing data in PMem increases performance in TPC-C due to the lower latency and higher bandwidth of PMem when compared with SSD/HDD. While these evaluations and characterizations have shed light on the design of veDB, this work aims to propose a practical, end-to-end industry database system solution built with disaggregated PMem stores.

Remote Direct Memory Access (RDMA). The advancements in high-performance networks utilizing RDMA have opened up new possibilities for modern distributed database systems [35], [52]–[54]. Microsoft has made many contributions to this area, including the development of a main memory distributed computing platform named FaRM [55]. This platform utilizes one-sided RDMA to improve its optimistic concurrency control protocol and allows upper layers to take advantage of its disaggregated memory model. This has made FaRM a key innovation enabler and a building block for high performance computing platform. For example, subsequent studies [56], [57] have shown how FaRM can be used to enable high-performance transactions. A1 [58] illustrates a distributed in-memory graph database built on top of FaRM. Other works in the field include FaSST [59], a

scalable distributed in-memory transaction processing system built on top of an all-to-all RPC system that uses RDMA's datagram transport for CPU efficiency. DrTM+H [60] utilizes RDMA to design efficient transaction execution and logging. Storm [61] proposes an RDMA-accelerated transactional data plane. Additionally, RDMA is widely used to accelerate distributed join processing [62]–[64] and data shuffling [65]–[69]. In comparison, veDB focuses on designing a distributed PMem storage engine that fully leverages one-sided RDMA primitives to deliver high-performance logging and page caching while preserving PMem's persistency to support fast recovery.

Recent years have witnessed a runaway rise in using RDMA and PMem. This trend has prompted several well-known database vendors to design their next-generation database systems or to re-design their existing systems. SAP HANA [22], [70] has attempted an early adoption of persistent memory, moving its columnar store to PMem and keeping only hot data in DRAM, allowing for a fast restart and recovery. Unlike SAP HANA, which couples PMem storage with database engines, our work adopts a disaggregated architecture that decouples PMem storage from database engines for better scalability.

IBM's DB2 pureScale [71] adopts cluster-scale distributed caching facilities to achieve high resource utilization and elasticity, as well as centralized global locking and page cache management for high levels of availability and scalability. The centralized global locking mechanism leverages RDMA Atomics to minimize the involvement of remote processors, while the get, put, and invalidation operations towards the global page cache are carried out by one-sided RDMA primitives to deliver outstanding performance.

Microsoft SQL Server [72] makes use of RDMA to leverage available cluster-wise remote memory to improve query performance. It has been shown that using remote memory, as opposed to local disks, can improve the latency of various TPC-H and TPC-DS queries by 2-100 times.

PolarDB [14] is a cloud database that uses a computation and storage separation architecture. To address the challenge of transferring large amounts of data for analytical queries, PolarDB deploys computational storage drives in the Alibaba Cloud to accelerate analytical workloads. By pushing down data-intensive tasks from front-end database nodes to back-end computational storage, it leverages the in-storage computing power to perform table scans much more efficiently. Furthermore, Alibaba proposed a serverless version called PolarDB Serverless [73], which adopts a disaggregated architecture for its page cache and leverages RDMA to mitigate the performance penalties caused by one extra network hop along the critical data paths between compute servers and the disaggregated memory pool.

Oracle Exadata [31], [74] is a popular industrial production system that adopts PMem in a storage and computing separation architecture. The storage servers in Exadata are equipped with multiple tiers of memory devices (e.g., PMem, flash, and SSD) and are interconnected to the database servers via RDMA-capable networks. The system's internal replacement policy intelligently caches hot blocks in the PMem cache layer to improve data block read speeds. It also buffers database transaction log writes in PMem layer to reduce commit time. Data blocks and REDO log records are transmitted by one-sided RDMA primitives to avoid IO cliff.

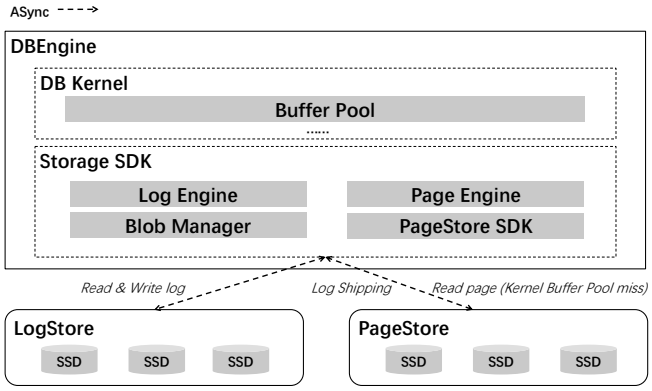

Fig. 1. The architecture of veDB.

Unlike pureScale and PolarDB, veDB's Extend Buffer Pool (EBP), which offers page cache management as well, is backed with disaggregated PMem stores that are interconnected via RDMA. This provides faster recovery and higher availability by leveraging PMem's persistency capability. In contrast to SQL Server and Exadata, veDB's disaggregated EBP is separated from the PageStore and can be scaled horizontally. Additionally, veDB's DBEngine is able to push down queries onto EBP components to make use of the CPU resources saved by utilizing one-sided RDMA verbs.

III. Background

In this section, we provide a short architectural overview of veDB, ByteDance's internal RDBMS. As shown in Figure 1, veDB is based on an architecture that separates computing and storage, following the log-is-database principle [75]. The DBEngine is the computing component of veDB. It processes transactions and performs computations for queries. Each transaction generates a REDO log that records the modification to the database. The REDO log is organized as a consecutive stream, and the offset in the log stream is called the Log Sequence Number (LSN). Therefore, LSN can indicate the order of REDO logs. The storage layer of veDB consists of LogStore and PageStore. LogStore receives the REDO log from DBEngine, and persists it to disk. PageStore stores versions of the data pages and continuously applies REDOs to construct the latest version of the data pages. Both stores are built on top of distributed storage and have a similar architecture to provide high availability and persistence. LogStore uses an append-only distributed blob storage, and PageStore provides random read capability at page granularity.

DBEngine is veDB's computing layer. It is responsible for query processing, data modification, and transaction management. It reads from and writes REDO logs to the storage layer and manages the buffer pool. Other operations, such as checkpointing, are offloaded to the storage layer. DBEngine communicates with the storage layer through embedded storage SDK, which provides simple APIs that encapsulate the

details of managing a remotely distributed storage system. We will discuss its details in the next section. After the REDO log is written to the LogStore, the thread of the LogStore will flush it to disk and notify the transaction processing thread in DBEngine through a callback function.

LogStore is a crucial part of veDB's storage layer and is responsible for persisting REDO logs. veDB uses the ARIES [76] method to handle transaction persistency and recovery. The persistence of REDO records uses write-ahead logging (WAL) and it guarantees the durability of transactions when a crash happens. Each REDO log record is assigned a unique LSN by LogStore based on their persistence order.

LogStore is built over an append-only blob storage system, ensuring data durability, high availability, and fault tolerance. In LogStore, each log record is persisted and replicated to three or six replicas (according to the configuration) before acknowledging DBEngine.

The storage SDK must be able to manage a large number of REDO logs and process incoming write requests simultaneously with low latency. It appends REDO logs using a data structure called *BlobGroup*. BlobGroup is a logical container that is composed of multiple append-only blobs. By default, each BlobGroup contains four blobs and provides 40GB of storage space. I/O operations on blobs within the same BlobGroup occur concurrently. Each I/O request is executed physically in a fixed size (8KB by default).

Logically, all log append requests against the same BlobGroup are merged into a single, longer-length request. This request is then divided into multiple smaller I/O requests at fixed sizes. Then, these fixed-size requests are assigned to different blobs within the BlobGroup in a round-robin fashion for execution. This model allows log append requests of a large I/O size to be divided into smaller requests, executed in parallel, and completed with lower latency. Additionally, multiple small I/O requests may also be merged into one request for execution, which reduces the number of physical I/Os that occur.

PageStore is responsible for page persistence and constantly replays transactions from the REDO logs to keep pages up to date. PageStore has its own storage unit called *segment*. It manages the entire life-cycle of segments. including data organization, distribution/replication, I/O control, state transition, and fault detection and recovery. Multiple REDO log records are assigned to one segment, and the records that belong to the same segment are assembled together and shipped to PageStore through RPC. If a REDO log record is shipped to PageStore and acknowledged, it is considered durable and will be applied to the corresponding pages asynchronously, checkpointing is not required.

Segments in PageStore are replicated to three or six copies across multiple availability zones. Considering the long-tail latency issue, fault tolerance, and availability goal, we choose to implement a quorum replication, and use a gossip protocol for filling in missing records. Before a log record is shipped to a PageStore segment, we attach a back-link that represents the LSN of the preceding log record for constructing the page in the same segment. With the back-link mechanism a PageStore instance can detect missing logs and gossip with other instances to retrieve them.

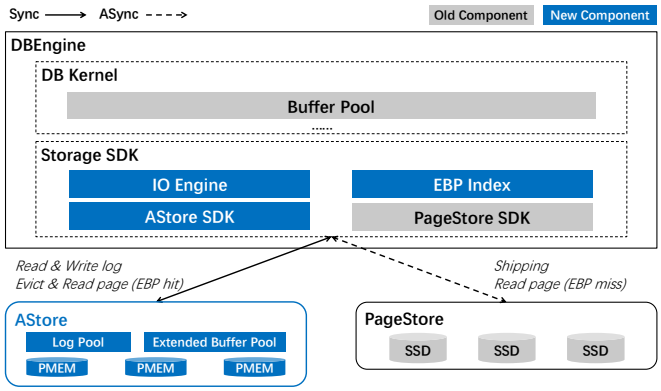

Fig. 2. The PMem accelerated architecture of veDB. AStore sits in a remote cluster with PMem hardware.

IV. Distributed PMem Storage Engine: AStore

After operating a large fleet of veDB instances for a few years, we observed that both LogStore's write latency and PageStore's read latency were high for some applications. To solve the latency issue identified in veDB, we introduce a PMem-based storage layer. Instead of embedding the PMem cache in the same host as the DBEngine (which would increase CPU contention and latency), we follow a disaggregation architecture [77] that pools the PMem resources from different hosts and connects them through a high-speed RDMA network. This results in a storage layer called *AStore* which replaces the LogStore. The advantage of having a disaggregated PMem pool is that it can be scaled horizontally and is independent of the computing layer.

However, there are some associated performance penalties. For example, the access speed of the remote cache is slower than that of the local PMem, and the transmission of pages across the network incurs additional traffic. To mitigate these issues, we use low-latency, one-sided RDMA verbs to get access to AStore. As a result, the I/O operation is truly run-to-completion, bypassing the CPU of the storage server, which provides AStore an extremely high read and write performance—with a read latency of 10µs and a write latency of 20µs. In contrast, traditional storage systems usually have a latency of a hundred microseconds.

Meanwhile, financial costs should also be taken into consideration. Although persistent memory devices are expensive, AStore can provide higher throughput and guarantee more stable performance at a similar cost compared with the previous SSD-based storage system in our deployment (see Section VII-B). Furthermore, in typical scenarios, the lifespan of REDO log in veDB's LogStore is short because it can be garbage collected once it is applied to the pages in PageStore, which happens quickly. Therefore, the capacity reserved for REDO logs in AStore for each database instance is small and limited to GB level by default. Overall, the financial cost of AStore is acceptable in our actual world deployment.

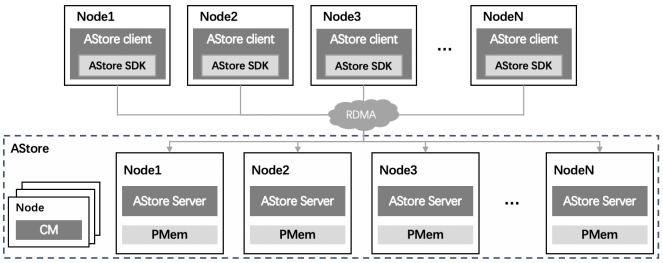

Fig. 3. The key modules in the AStore architecture. The system runs on multiple nodes with PMem hardware. The PMem is accessed directly by the veDB nodes.

A. Key Modules

AStore is divided into three modules: the AStore Client, the AStore Server, and the central node cluster manager (CM). The design of AStore is generic, and is applicable on other database engines.

The AStore Client is an access module and provides an external interface with read and write operations over an append-only space. When the application needs to write, it first calls the open interface to obtain the segment addresses from the CM. Then it goes to the corresponding storage node to obtain a writable continuous segment space. We use multiple copies of segments to ensure data reliability, with a configurable replication factor for different segments. By default, the segment that stores the log has three copies and the segment storing the page has only one copy. The routing information is cached in the memory of the AStore Client, and subsequent reads and writes do not need to go through the CM.

The AStore Server's primary task is to efficiently manage PMem resources, including the data layout, metadata cache, and background tasks. In addition, the AStore Server will register the full physical address of PMem devices to the RDMA network interface card (NIC) and obtain the virtual address through *mmap* for space management. The AStore Server divides the memory into the superblock, segment meta, I/O meta, and segment storage areas. We use a bitmap to manage segment applications and releases; every time a segment is created, a bitmap in memory is used to track free space. The corresponding bitmap is reset to zero when the application releases the space.

The Cluster Manager (CM) is responsible for managing the resources of the entire cluster. Its main functions include storage node management, registration, fault detection, background task scheduling, capacity expansion, and load balancing. In addition, the CM monitors the survival status of the AStore Server nodes by maintaining heartbeat messages with each node. The AStore Server node reports capacity, I/O load, and segment health status in the heartbeat message. When a new segment is created, the AStore Client obtains the route of the AStore Server node from the CM and the CM returns the appropriate nodes according to the capacity and load of the AStore Server nodes.

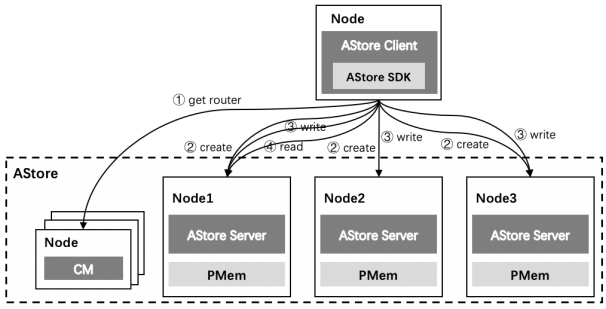

Fig. 4. The I/O model for AStore. Client nodes first retrieve routing information from the CM and then can interact with the PMem on AStore nodes directly.

B. Supported Operations

In this section, we introduce the critical processes of AStore in detail, including how the operations, such as create, write, read, and delete, work.

Create. To create a new segment, the AStore Client sends an RPC message to the CM to request the allocation of a segment. Then, the CM selects a storage node, and after receiving confirmation, the AStore Client sends an RPC message to apply for new storage space in persistent memory. Because it is using RPC messages, the entire process of *Create* takes a few milliseconds. This delay is much more significant than the write delay, which is about 20µs.

Write. To write to a segment, the AStore Client appends the data to the offset of the corresponding segment through a one-sided RDMA WRITE and records the offset and length. This way a segment's effective data length is known after a failure occurs. If writing to three (configurable) storage nodes is successful, AStore updates the written length of the segment in the AStore Client's segment-meta. If any copy fails, it returns a failure to the application and freezes the segment with the current effective length. The writing process only appends to the end of a segment, which is more friendly to PMem media and allows the system to achieve extremely low latency.

AStore highly relies on the persistence of PMem devices to deliver fast recovery. Therefore, when using RDMA and PMem in combination, it is necessary to ensure that the data is actually written to PMem's persistence domain. Furthermore, for PMem devices in our environment that only support Asynchronous DRAM Refresh (ADR), we have to guarantee that the data is flushed to PMem's memory controller. In addition to the persistency requirement, we would also like to bypass the AStore Server to extract the RDMA network's performance potentials fully. After exploring multiple approaches [78], we finally choose to disable Intel's data direct I/O (DDIO) feature [79] on the AStore Server, and leverage one-sided RDMA READ to flush the data from the cache to the memory controller of PMem devices and therefore persist the data in PMem. This approach has been demonstrated to deliver a near-optimal performance in practice. The entire write operation combines two one-sided RDMA WRITEs and one-sided RDMA READ, and these work requests are chained

together to reduce memory-mapped I/O (MMIO) operations. The use of one-sided RDMA verbs in the critical I/O path and the zero-copy capability of RDMA results in ultra-low write latency.

Read. The application specifies reading a particular piece of data using two segment parameters, i.e., offset and length. The AStore Client checks the validity of the parameters and selects an online copy to read through one-sided RDMA READ.

Delete. The AStore Client sends an RPC message to both the CM, requesting to delete a segment from its metadata, and to the AStore Server, which releases the segment and resets the bitmap of the allocation manager.

If an AStore Server fails, the segment allocated on the AStore Server cannot be written. Instead, the application opens a new segment to write data to, and as long as there are three (configurable) healthy storage nodes in the cluster, the writing service can be provided.

C. Data Consistency with One-Sided RDMA

In order to ensure the reliability of the data, the CM detects any storage failure node and selects a new storage node to rebuild a lost copy of the segment. If the failed node returns to the cluster, the segments on it are considered stale and will be cleaned up by the CM, releasing the PMem space. This brings significant challenges to the use of one-sided RDMA READ and WRITE. Because of the read and write process with one-sided RDMA verbs bypassing the server's software stack, the AStore Client directly operates on the PMem of the storage node. If the replica set carrying the segment changes due to failure and reconstruction, the AStore Client may not be aware of it in time and may read and write the wrong segment, causing an inconsistency.

In order to resolve this issue, the AStore Client and the AStore Server perform a series of background tasks. First, the AStore Client regularly checks with the CM to see if the segment's route has changed. The AStore Server does not handle the CM's request to clean the stale segment immediately but instead periodically cleans it. This cleaning cycle is much longer than the AStore Client query cycle. Therefore, the AStore Client can recognize the stale segments in time and will not read/write inconsistent data. In order to speed up the release of the stale segment, the time interval between the AStore Client receiving updated routes from the CM is short compared to the interval of segment cleaning. In addition, the routing update is performed at segment granularity so that the number of segments in AStore does not make the CM a bottleneck.

There is another scenario that may cause data inconsistencies. Suppose AStore Client *A* fails after creating segment *X*, and the new AStore Client *B* reclaims the space of segment *X*. After a while, AStore Client *A* returns to the cluster and issues a now inconsistent RDMA WRITE to segment *X*. We avoid this problem by directly maintaining a lease in the AStore Client and the CM. When AStore Client *A* returns to the cluster and communicates with the CM, it will find that the lease has expired or the owner has been changed to AStore Client *B*.

V. Boosting veDB Performance with AStore

In veDB, LogStore was initially built over an SSD-based blob storage system and asynchronously submitted I/O to BlobGroups in parallel to reach high throughput. But there are several bottlenecks in the system:

1) For a single write request, SSD and TCP based write path is still high in latency.
2) CPU resources are required to schedule every I/O request. This I/O schedule burden increases latency and causes periodic spikes in latency.

Compared to SSD-based LogStore, AStore offers a low latency storage by adopting one-sided RDMA verbs to directly access remote persistent memory. It can improve veDB's performance in REDO log write. In addition, AStore also offers a larger and cheaper capacity for caching compared to DRAM. Therefore, we replaced LogStore with AStore for veDB and added Extended Buffer Pool backed by AStore, as shown in Figure 2. In this section we will discuss the details of using AStore for fast log persistency and extended buffer pool.

A. Log Space Management

As described in the Section III, in the original design of LogStore, the storage SDK manages data using logical containers named BlobGroup. In the updated implementation that incorporates AStore, a new logical container called SegmentRing has been introduced to manage a collection of append-only segments. This new design simplifies both write and read processing, reduces the number of RDMA calls and improves overall performance of veDB.

There are two key differences between BlobGroup and SegmentRing. First, the large log write I/O is not split into smaller ones in SegmentRing. Writing 256KB block using one-side RDMA takes about 0.1ms according to our test. That is fast enough in our use cases.

Second, SegmentRing utilizes a ring buffer structure to manage a collection of segments arranged in a circular fashion. The size of each segment within the container is fixed and can be configured by clients (default size is 1GB). Additionally, the size of the SegmentRing is also configurable, with a typical setting of 50 segments. Upon initialization of DBEngine, all segments with an index starting from 0 within the ring are pre-created by the storage SDK.

Inside each segment, the content is divided into two parts: the header and the REDO log data. The header contains a status and an LSN field. The status field indicates whether a segment is empty, in-use, full, or in-error, and the LSN field indicates the LSN of the first REDO log record stored in the segment. When DBEngine crashes, a binary search can be performed on all headers in the SegmentRing and it can efficiently identify the largest LSN. This LSN will be used as the new starting LSN when normal operations resume.

B. High-Performance Log Write Path

In the original veDB architecture, writing each REDO log buffer to remote storage asynchronously resulted in additional

costs such as data copying and thread-context switching. By replacing LogStore with AStore, the I/O submission process was simplified to a run-to-completion model where requests are either executed to finish or queued. Directly writing the REDO log buffer to the remote PMem managed by AStore through RDMA eliminates the need for redundant data copying and thread-context switching. The use of one-sided RDMA primitives also bypasses the software stack on the AStore server. Registering the global log buffer (PMem-backed) of DBEngine to the RDMA NIC during initialization further reduces data copying.

Because the log space is organized as a SegmentRing in AStore, when the storage SDK receives a REDO log write request, it checks the usage status of the current segment. If the current segment has enough space for the incoming write, it appends the REDO log to the segment. Otherwise, it freezes the current segment, advances to the next segment in the ring, and sets its header to the start LSN of the current REDO log. Once the write request is completed, the global persistent LSN is advanced, and the database transaction can be committed.

C. Extended Buffer Pool Using AStore

veDB uses a computing and storage separation architecture. This architecture has many advantages, but suffers from increased network communication as a result. In veDB's original design, if the data page being accessed is not cached in the computing node, it needs to be read from the remote PageStore, causing a delay of approximately 1ms. During this wait time, the transaction thread cannot perform any other actions, resulting in increased transaction latency.

AStore resolves this issue by reducing read latency. Reading a 16KB data page only takes AStore 20µs, which is in the same order of magnitude as a local SSD. Additionally, PMem provides a much larger capacity than RAM. So in addition to cache data pages in DBEngine's buffer pool, we also cache pages evicted from buffer pool in AStore. We call this cache in AStore the extended buffer pool (EBP).

We implement a global extended buffer pool management structure that encapsulates segment creation, writing, reading, and release. Each data page is read and written with a unique identifier. Since the loss of EBP pages only reduces EBP hit ratio and will not impact the query's correctness, we use the replication factor as one for EBP segments (AStore supports configurable replication factor for segments). The client side of the storage layer is modified to maintain the list of pages available in EBP.

The EBP is managed by the storage SDK through a data structure called the *EBP Index*, which is a collection of key-value pairs **{(space_no, page_no), lsn + segment_id + offset}**. The key (space_no, page_no) is called page ID. The EBP Index is updated whenever the EBP is modified during buffer pool load and evict operations. When the DBEngine calls the storage SDK interface to evict a page, it will decide whether to write to the EBP according to the space priority and EBP capacity. After successfully writing the page to the EBP, the storage SDK updates the EBP Index in memory. For

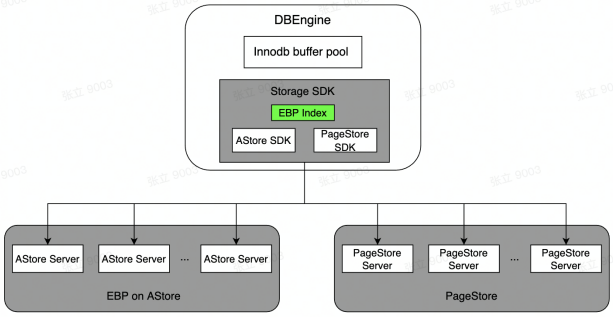

Fig. 5. An illustration of how the extended buffer pool (EBP) fits into the system architecture of the DBEngine.

the pages existing in the EBP index, if DBEngine modifies them in its local buffer pool, a mapping of page ID and its latest LSN will be maintained and periodically sent to the AStore server in batches. AStore servers keep this mapping in memory for pruning out stale data pages when rebuilding EBP index during recovery of DBEngine failures.

We provide two policies for managing the EBP capacity: Priority or flat. The flat strategy does not partition the EBP space, and all pages evicted from the main buffer pool are placed into the space evenly. Under the priority management strategy, we divide the total EBP space into areas with priority. Pages of priority can be placed in any space with the same or lower priority, and pages with the highest priority can occupy up to 100% of the total space.

D. Page Management in the EBP

Page management includes two processes: compaction and eviction. In the compaction process, AStore writes data in an append-only manner. When a page is written to AStore multiple times, the page space of the older version on the segment will be marked as *garbage*. Dropping or truncating a table will also produce garbage . In order to improve the space utilization of the EBP, AStore will track the amount of garbage on each segment and periodically process segments with a high amount of garbage, moving the valid data to the new segment and releasing the old segment. This process is called compaction, it is transparent to the DBEngine, and its frequency is configurable. If compaction is not enabled, the segments with high amounts of garbage will be released directly, releasing part of the valid pages in the process.

We use a simple least recently used (LRU) cache replacement policy to manage EBP space for page eviction. In order to reduce the contention of multiple threads updating the LRU cache simultaneously, we use multiple LRU lists to manage these pages. When the DBEngine reads or writes a page from the EBP, it computes the hash of page ID to determine which LRU list the page belongs to. If the space usage of the EBP has reached a threshold we select a certain number of pages to evict based on the order of recency. The evicted page space is reclaimed for reuse through compaction.

E. Recovering from Failures

Recovering the REDO log space. When the DBEngine process crashes, it uses the standard MySQL recovery proce-

dure to recover. This involves scanning the logs starting from a checkpoint (determined by the LSN) and applying all REDO logs. The starting log record can be easily located using a binary search of the SegmentRing, as the start LSN is stored in the header of each segment. After all REDO logs have been applied, any uncommitted transactions are rolled back and the recovered DBEngine continues to service the application.

Since log segments are replicated, the failure of one AStore Server is transparent to DBEngine. When a writing failure occurs in a segment, the storage SDK will close the failed segment, create a new segment, and automatically retry to write to the new segment.

Recovering of Extended Buffer Pool. There are two cases to consider. The first case is DBEngine process failure. Since we use disaggregated PMem store for EBP it usually survives DBEngine failures. To use the data in EBP, DBEngine collects information on pages cached in each AStore server and rebuilds the EBP index. The system will keep the most up-to-date EBP pages and discard stale ones. Each AStore server periodically receives page IDs and their latest LSN from DBEngine and stores them in memory. In response to a recovery request from the DBEngine each AStore server scans pages stored in local PMem, compares their LSNs with the one in memory, discards those with older LSNs, then returns on the valid page IDs along with their position to the DBEngine.

The other case is AStore server failure. In the current implementation, when a AStore server crashes, pages stored in it are considered lost and are removed from EBP index. Since the EBP pages are evenly scattered on a cluster of AStore servers, failure of one (or a small number) of AStore servers will only reduce the cache hit ratio, and will not affect query correctness. Since AStore uses PMem as storage, it can leverage the persistent property of PMem and recover EBP data pages locally once the AStore server is restarted. We left this optimization for future work.

VI. Push-Down for Query Acceleration

veDB uses a single-threaded processing model for handling queries. This means that some operations, such as semantics analysis, optimization, data reading, and execution, are completed without multi-thread synchronization, meaning no extra waiting time. However, when the amount of data is large, only one thread processes the request, while other idle threads and resources cannot help accelerate the process, causing their response time to increase. This disadvantage is even more prominent because of additional network traffic generated by accessing remote storage.

We introduce a push-down query (PQ) feature to solve these issues. The idea is to identify eligible fragments of the query plan and execute them in storage servers (e.g., PageStore). Since the storage is distributed, we decompose each push-down request into multiple concurrent tasks based on the page distribution in PageStore and send them to the corresponding server for execution. The computing layer collects the executed results for additional processing (e.g., secondary aggregation) and returns them to the user. With the addition of an extended buffer pool powered by AStore, the query fragments can also be executed in the AStore server, leveraging the pages cached in EBP.

The query push-down feature provides the following advantages:

1) Parallel reading of pages from multiple storage nodes improves I/O access performance
2) Push-down operations to the storage layer reduce network transmission overhead. For example, projection, filtering, and aggregation operations can be pushed down directly to the storage layer.
3) CPUs in the storage layer can be used to assist queries. This is especially effective on PMem servers because access data in the PMem page cache is through one-sided RDMA primitives, with their CPUs largely uninvolved.

A. Query Push-Down Framework

veDB's original query optimizer is modified to recognize and mark query plan fragments that can be pushed down to the storage layer for parallel processing. During the execution of these plan fragments, we identify the data pages needed by the query and generate RPC calls containing the plan fragment and page IDs plus the corresponding page LSN and send them through storage SDK. Presently, only scans with simple filters and/or aggregation over a single table reference are supported. The push-down plan fragment has no joins or subqueries. Each plan fragment is serialized and passed down to the storage layer through these RPC calls, and then the returned results are passed back to the DBEngine and merged to be consumed by the following plan operator through registered callback functions .

Currently, the decision of pushing down the execution of plan fragments or not is simply decided by a threshold of the number of rows to be scanned by the plan fragment and a session variable enabling the PQ feature. Developing a cost-based optimization is planned for a future release of veDB.

A separate process containing the veDB executor code for scan, filter, and aggregation operator is deployed in each PageServer and AStore server. The processes will accept the plan fragments from DBEngine, compute the results and return them to DBEngine.

B. Push Query Execution Down to AStore

When a PQ request is sent through storage SDK, the original request gets split up into parallel tasks by looking up the requested pages in the EBP index. We created a group of tasks for those pages in the EBP index, one for each AStore server that cached eligible pages. Similarly, for those pages not found in the EBP index, a group of tasks will be created for each PageServer that persists in the relevant data pages. Each task contains the push-downed plan fragment, a list of data pages, and their corresponding LSNs. These tasks are dispatched to corresponding servers in parallel.

When executing the PQ task, the AStore Server performs similarly to the PageServer: it reads the corresponding pages from the EBP and then applies the operators to these pages.

Component	Deployment	Configuration
PageStore, LogStore	4 Bare metal boxes for each store: 1 Root Server 3 Data Servers	**CPU**: Intel Xeon Gold 5218 @2.30GHz 64 cores **Memory**: 376 GB **NVMe SSD**: 4 × 3.4TB, 2 × 2.8TB **Network**: Mellanox ConnectX-5 25Gbps; OFED-5.0
AStore	4 Bare metal boxes: 1 Root Server 3 Data Servers	**CPU**: Intel Xeon Platinum 8260 @2.40GHz 96 cores **Memory**: 128 GB **PMem**: 1TB Intel® Optane™ Persistent Memory **Network**: Mellanox ConnectX-5 25Gbps × 2; OFED-5.0
veDB, Client	Virtual Machine: CPU 20 core Memory 160GB Storage 250GB	**CPU**: Intel Xeon Platinum 8260 @2.40GHz 24 CPU **Network**: Mellanox ConnectX-5 25Gbps; OFED-5.0

TABLE I
THE SPECIFICATIONS OF EACH PART OF THE SYSTEM USED BY OUR EXPERIMENTAL ANALYSIS.

The difference is that AStore Server reads the local PMem while the PageServer reads the local disk.

The read and write interfaces provided by the AStore Server are through one-sided RDMA verbs. These operations do not require the participation of the AStore Server CPUs, so the EBP on the AStore Server may only need a few cores, leaving plenty of CPU resources idle. In addition, the large number of pages stored in the EBP is warm data likely to be requested by incoming queries. By executing push-down query fragments on the PMem servers, we can use idle CPU resources and warm data pages in the EBP.

Recall that we implemented two strategies for managing EBP space, priority and flat. The flat strategy is not optimal for push-down queries because pages have an equal chance of being evicted from the EBP, reducing the chances that the requested data remains in the EBP for consecutive queries. Consequently, the priority strategy is better for supporting push-down queries. We can allocate more high-priority space for tables used often by the push-down queries to ensure that most of its pages are not evicted from the EBP.

VII. EXPERIMENTAL EVALUATION

In this section, we quantify the impact of our PMem read/write acceleration, extended buffer pool, and query push-down. First, we compare the transaction performance of veDB with a traditional SSD-based LogStore and our PMem-based LogStore using the TPC-C [80] benchmark. Then, we demonstrate the acceleration of our PMem based extended buffer pool with both TPC-C and TPC-CH [81] benchmarks. Lastly, we test the effect of pushing execution partial queries to our EBP hosts to accelerate the execution of queries. All experiments are performed in a cluster with the configuration shown in Table I.

A. AStore for Logging Acceleration

In the first set of tests, we show the effect of using AStore as REDO log storage and test it on both a standard benchmark and our internal customer workloads. We first conducted a simple log write test and saw significant improvement in both bandwidth and latency. Then, we measured the latency and transaction rate changes using the TPC-C benchmark and our internal workloads.

	Avg. Write Latency (ms)	Avg. I/OPS	Avg. Bandwidth (MB/s)
W/O PMem	0.638	1,527	5.97
W/ PMem	0.086	11,465	44.79

TABLE II
MICRO BENCHMARK FOR LOG WRITING WITH AND WITHOUT PMEM

Log Writing Micro-Benchmark. Since the database kernel usually spends a significant amount of time writing the REDO log, we conducted a micro benchmark to measure how the log write latency was affected using AStore. We develop a micro benchmark tool that continuously writes 4KB pages to either AStore or the regular LogStore in a single thread and measures the latency, I/OPS, and bandwidth. The results are shown in Table II. AStore provides an almost 7× improvement in write latency, I/OPS, and bandwidth.

TPC-C Throughput and Latency. Here we compare the throughput and latency of TPC-C benchmark on an original veDB deployment with the one deployed with AStore. As shown in Figure 6, veDB with AStore performs better in all cases, and the throughput significantly increases with more workload pressure. Without AStore, the throughput peaks at 68,000 transactions per second (TPS); with AStore, the throughput peaks at nearly 90,000 TPS, bringing more than a 30% performance improvement.

Additionally, we collected both P95 and P99 latency for both configurations and saw that veDB with AStore achieves a lower latency consistently. The results of P95 are shown in Figure 7. The results of P99 are similar and omitted due to space limitations. Note that AStore's latency improvements begin to decline as the number of clients increases beyond 64. This pattern of dropping throughput under high concurrency is consistent with what we have observed with MySQL 8.0 in a similar test where TPC-C throughput peaked with 32 concurrent clients.

Order Processing Workload. We have identified some internal use cases that did not perform well on veDB, and we would like to see if AStore can help improve the performance. One of them is an application that performs batch operations on order requests. In this test scenario, the orders of a vendor are batched into one transaction block. The vendor's account balance record is updated in the transaction, and the updated balance record is returned and inserted into the order flow table.

The requirements of this scenario are:

1) The INSERT data is wide, about 2KB.
2) The UPDATE is a hot row update. There are often many concurrent updates for the same merchant.
3) The customer requires the performance of this scenario to reach 10,000+ TPS.

We tested the performance for a single insert operation and the whole order processing transaction. Figure 8 shows that using AStore brings significant improvements for both tests. For the single insert transaction, veDB with AStore can reach over 10,000 TPS with only 8 clients. In contrast, veDB without AStore just has 3,339 TPS with 8 clients. We observe more than 3× improvement. For the order processing transaction, veDB with AStore reached over 10,000 TPS with just 64

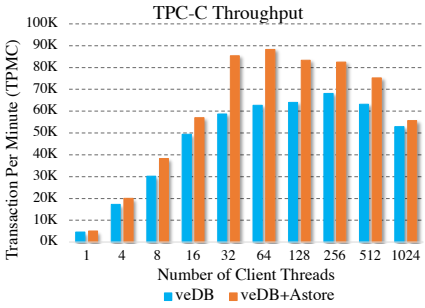

Fig. 6. veDB with AStore improves throughput in TPC-C test with peak performance of nearly 90,000 TPS with 64 clients, up 30%.

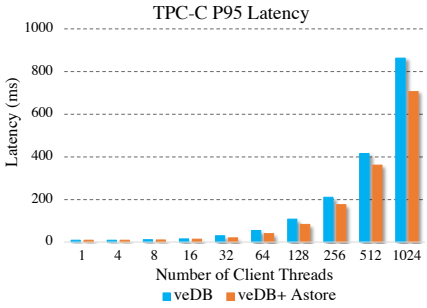

Fig. 7. veDB with AStore reduces transaction latency in TPC-C test, showing here p95 reduced by 50% at 32 clients at most

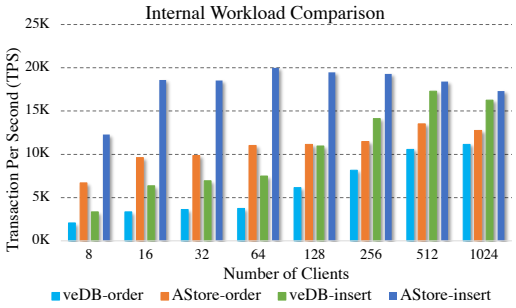

Fig. 8. An evaluation of veDB with AStore using internal customer workloads. With AStore, veDB reached the required TPS with 64 clients.

Fig. 9. Average and maximum transaction latency while running an internal advertisement library with and without AStore. Using AStore is approximately 20× faster.

clients, but the native veDB needed more than 512 clients to meet the target.

Advertisement Workload. Internally, we have a core data processing library for advertising, with a strict latency requirement for its queries (∼10ms for P99 latency). As veDB separates computing and storage, our initial tests showed that the P99 latency for the tested queries might reach up to ∼150ms on veDB. Adding AStore to accelerate logging reduces the P99 latency on veDB to ∼5ms, which is much better than the stock veDB and MySQL.

We conducted the test in an environment where the real application workloads were duplicated and directed simultaneously to a stock veDB and a veDB with AStore enabled. We show a comparison of query latency between those two cases in Figure 9. On average, AStore accelerates queries by nearly 20×, where most queries are complete in ∼5ms. The latency improvement over the worst case is equally significant for average, which drops from ∼500ms to ∼20ms. Note that the improvement in this test is significantly greater than the writing latency improvement seen in the single-threaded micro-benchmark in Section VII-A. This is because AStore uses one-sided RDMA and has fewer CPU contentions, which brings more benefits in the multi-threads scenario as the simultaneous transactions do not have to wait for each other.

The standard TPC-C and the internal benchmarks show that using AStore for logging acceleration improves throughputs and dramatically reduces transaction latencies. We also observed that the throughput of veDB with AStore peaked earlier (at 64 clients) than the one without AStore (peaked at 128 clients). This behavior is consistent with other studies of PMem [20], [21], which also observe a decrease in read and write performance as concurrent accesses increase, leading the workload to be CPU-bound.

B. Extended Buffer Pool

Impact of AP workload. Firstly, we test if the extended buffer pool can reduce the impact of adding analytical processing (AP) workloads when the database is dealing with transaction processing (TP) workloads. We use the TPC-CH benchmark for this test. The database is loaded with 1000 warehouses of data and has 32 TP clients. We use different numbers of AP clients, either 0, 1, or 8, and run the test for 30 minutes after 5 minutes of warm up. As the Figure 10 shows, adding one AP query stream reduces the TP throughput by 5%, and eight AP streams reduce the throughput by nearly 30%. The contention of pages in the buffer pool causes this. However, if we turn on the extended buffer pool, we see consistent improvement in transaction throughput with EBP.

AP query latency. In this experiment, we test the acceleration effect of the extended buffer pool on analytical queries. We use the TPC-CH benchmark dataset with 1000 warehouses and select a subset of OLAP queries whose execution time is below 1000 seconds. The experiment is done with two buffer pool configurations, one with a 16GB buffer pool and one with 32GB. In both cases, the EBP is configured with a fixed size of 256GB and is enabled and disabled alternately. Each query is run once to get the buffer pool warmed up, then the query is run three times more, and the average elapsed time of those three runs is recorded. Finally, we divide the EBP-disabled elapsed time by the EBP-enabled elapsed time to compute the speedup factor provided by the EBP.

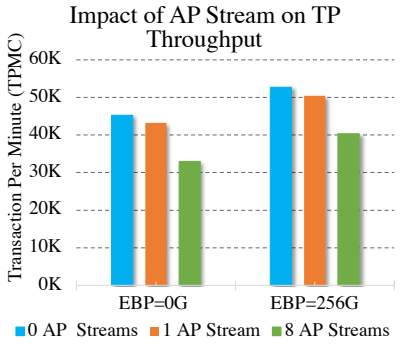

Fig. 10. TPC-CH evaluation for AP impact, with and without the extended buffer pool. The EBP provides a consistent improvement to TP throughput with different numbers of AP streams.

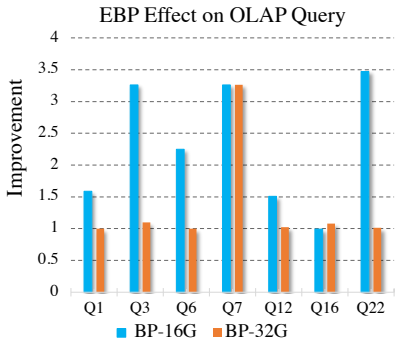

Fig. 11. The performance improvement on a selection of TPC-CH benchmark queries. The EBP provides performance gains when the size of the query's working set is larger than the size of the native buffer pool.

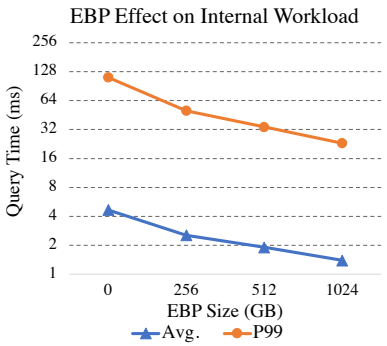

Fig. 12. The effect of the EBP size on an internal benchmark. For all EBP sizes, the latencies are reduced.

As Figure 11 shows, query 7 with EBP enabled has more than 3× improvement in elapsed time in both 16GB and 32GB settings, but query 16 shows little improvement. This is because the working set of query 7 is larger than 32GB, so it can use the EBP after it uses up the native buffer pool. Accessing data from the EBP through RDMA is significantly faster than fetching that data from a remote storage server. Query 16, on the other hand, is a simple join with two tables. Its working set is minimal and can be fit into the 16GB buffer pool. Therefore, its performance is barely impacted by the presence of the EBP. All other queries show different degrees of EBP acceleration, up to 3.5× faster.

Evaluation on internal workload. One of our core operation databases contains a large amount of data. The primary table data is nearly 17TB, the index data is 2.3TB, and the buffer pool is 120GB. The typical query patterns are lookup queries on primary keys or secondary indexes. However, due to the large data size, the hit rate of the buffer pool is about 95%, resulting in a long average response time and a significant P99 latency. Therefore, we decide to use the EBP to reduce the overall response time in this database. The experimental results are shown in Figure 12.

The results show that even with a modest extended buffer pool of 256GB, the average response time is reduced by 45%, and P99 query time is reduced by more than 50%. The results also show a diminishing return of larger EBP size, with each doubling of the EBP size reducing latency by approximately half as much as the last, as there is only so much data eligible to be cached in the EBP.

Combining log caching and EBP. We evaluate the combined effect of AStore for log caching and EBP using the Sysbench [82] transactional benchmarking tool to fully measure the impact of adding AStore layer to veDB compared to the original veDB. Since adding AStore to the deployment would introduce additional cost, we want to see that giving roughly the exact total cost of the hardware, which deployment has a better performance. Considering the price per GB of PMem used in the test system is approximately one-third of the cost of DRAM [83], we reduce the buffer pool size of veDB + AStore by XGB, and set the EBP size as $3X$GB. The specific configurations are shown in Table III. The experiments were run in a real-world cloud environment.

veDB		veDB+AStore		
Cores	**Buffer Pool**	**Cores**	**Buffer Pool**	**EBP**
32	100GB	32	40GB	180GB
16	40GB	16	20GB	60GB
8	20GB	8	10GB	30GB

TABLE III
THE DEPLOYMENTS THAT ARE USED TO COMPARE VEDB AND VEDB + ASTORE (WITH EBP) IN FIGURES 13.

Figure 13 shows the percentage of improvement in queries-per-second (QPS) of veDB with AStore and EBP enabled comparing to the stock veDB. The results show that the performance gains provided by AStore with EBP are significant with less than 64 clients. However, as the concurrent clients increase, the improvements decrease. When the number of clients reaches 256, the improvement diminishes. This is because veDB would have to do more and more EBP maintenance work when the throughput increases under high concurrency. In our current design, each AStore client (DBEngine in this case) manages the space allocated for EBP and maintains the EBP index when flushing buffer pool pages to AStore. This will consume additional CPU resources in the DBEngine. Moreover, the client uses a lock mechanism to control concurrent access to the EBP index, which can cause performance degradation under high concurrency. In future work, we plan to explore alternatives for the EBP index management to reduce resource contention under high concurrency.

C. Push-Down Queries

In this experiment, we assess the effect of offloading some query operators to the storage layer using the TPC-CH benchmark. First, we run the 22 AP queries in TPC-CH with the original veDB configuration (no EBP, no query push-down) as a baseline. Second, we enable both query push-down and the EBP, then rerun the 22 queries. With this setting, eligible query operators will be pushed down to the EBP host for execution if the required page can be found in EBP (by checking EBP index). If the required page is not found in the

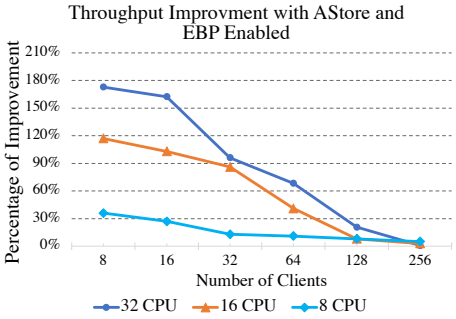

Fig. 13. The throughput improvement of veDB with AStore and EBP enabled. Lower concurrency environments show more substantial improvement.

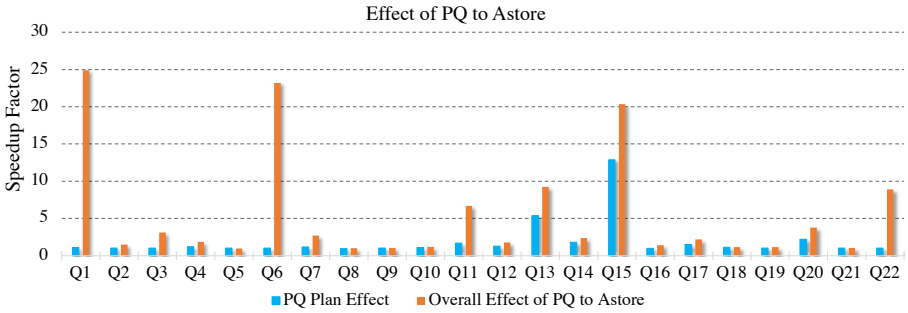

Fig. 14. The effect of push-down queries on the EBP. Some queries show dramatic improvement as their push-down computations are executed in parallel.

EBP, the operators will be pushed to PageStore for execution. To minimize the cold buffer effect, all queries are run three times in each experiment, and the average time of the second and third runs is used as the query elapsed time. We divided the query elapsed time in the baseline experiment by that of the experiments with query push-down, and EBP enabled to get the speedup factor.

The push-down effect on EBP is shown in Figure 14 orange bars. When the EBP and query push-down are enabled, queries 1, 6, 11, 13, 15, 20, and 22 have significant improvements which range from 4× to 24×. Queries 1, 6, and 22 have their aggregation operations pushed down, significantly reducing the data transferred and the latency drastically. Likewise, queries 11, 13, and 15 have a selective filter pushed down, significantly reducing elapsed time. Overall, the geometric mean of elapsed time speedup of all queries is about 2.8×.

We notice that, for a given query, some query plans are more friendly to operator push-down than other query plans. For example, veDB selects a nest-loop join plan for the join of customer and order tables in query 13 by default. However, a hash join plan would be selected with query push-down enabled because it allows more plan fragments to be push-down. To gauge the effect of query plan changes, we also run another test in which we disable the EBP and push-down queries and use a query hint to force the query plan to be the same as the best push-down plan. The effect of plan change alone is shown in Figure 14 blue bars. Using the result of this experiment as a new baseline, we still see 2× speedup of the geometric mean of elapsed time.

VIII. Conclusion

In this paper, we demonstrated how to use PMem and RDMA to accelerate cloud-native databases. We add PMem to a distributed storage engine and employ RDMA to access it. This enhanced distributed storage engine can replace SSD- or HDD-based storage to accelerate the persistence of the REDO log, increase the transaction throughput and reduce latency. It can also host an extended buffer pool to cache hot pages for queries that access large amounts of data, which is a typical pattern of analytical queries.

Furthermore, since the data stored in the extended buffer pool is accessed directly through RDMA without involving the host CPU, we can push operators to the hosts of the extended buffer pool and leverage the idle CPU to filter and aggregate the data.

We demonstrate how these innovations significantly improve the performance of both transactional and analytical queries. The experiments show that our design can improve throughput by up to 1.5× and reduce latency by as much as 20× for both standard query processing benchmarks and real-world applications.

There are a few directions for future development that can be further explored. The first is to integrate a cost-based strategy for query push-down so that the system can automatically decide the push-down strategy to maximize the benefit. Another is to expand the usage of EBP. For example, it could be used by stand-by instances that serve read-only queries. It could also be used to speed up the warm-up process for the buffer pool during crash recovery. The third is improving concurrency management for EBP, especially for concurrent EBP index access. The last is to leverage PMem persistency to recover EBP locally when an AStore server fails.

Acknowledgments

We thank all people who made contributions to the design and development of the veDB system. Fan Yang and Yong Zhou conducted most of the tests during their internship at ByteDance. We greatly appreciate their efforts. Finally, we thank Varun Gupta and Ron Hu for their careful proofreading of this manuscript and suggestions for correction.

References

[1] ByteDance, "veDB for MySQL," https://www.volcengine.com/product/vedb-mysql, (Accessed on 2022-10-05).

[2] J. Tan, T. Ghanem, M. Perron, X. Yu, M. Stonebraker, D. DeWitt, M. Serafini, A. Aboulnaga, and T. Kraska, "Choosing a cloud dbms: architectures and tradeoffs," Proceedings of the VLDB Endowment, vol. 12, no. 12, pp. 2170–2182, 2019.

[3] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang et al., "The snowflake elastic data warehouse," in Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 215–226.

[4] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak et al., "Delta lake: high-performance acid table storage over cloud object stores," Proceedings of the VLDB Endowment, vol. 13, no. 12, pp. 3411–3424, 2020.

[5] Y. Yang, M. Youill, M. Woicik, Y. Liu, X. Yu, M. Serafini, A. Aboulnaga, and M. Stonebraker, "Flexpushdowndb: Hybrid pushdown and caching in a cloud dbms," Proceedings of the VLDB Endowment, vol. 14, no. 11, pp. 2101–2113, 2021.

[6] D. Durner, B. Chandramouli, and Y. Li, "Crystal: a unified cache storage system for analytical databases," Proceedings of the VLDB Endowment, vol. 14, no. 11, pp. 2432–2444, 2021.

[7] H. Li, Alluxio: A virtual distributed file system. University of California, Berkeley, 2018.

[8] S. Shedge, N. Sharma, A. Agarwal, M. Abouzour, and G. Aluç, "An extended ssd-based cache for efficient object store access in sap iq," in 2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, 2022, pp. 1861–1873.

[9] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in Proceedings of the 36th annual international symposium on Computer architecture, 2009, pp. 2–13.

[10] M. Böther, O. Kißig, L. Benson, and T. Rabl, "Drop it in like it's hot: An analysis of persistent memory as a drop-in replacement for nvme ssds," in Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021), 2021, pp. 1–8.

[11] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in Proceedings of the 2017 ACM International Conference on Management of Data, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1041–1052. [Online]. Available: https://doi.org/10.1145/3035918.3056101

[12] P. Antonopoulos, A. Budovski, C. Diaconu, A. Hernandez Saenz, J. Hu, H. Kodavalla, D. Kossmann, S. Lingam, U. F. Minhas, N. Prakash, V. Purohit, H. Qu, C. S. Ravella, K. Reisteter, S. Shrotri, D. Tang, and V. Wakade, "Socrates: The new sql server in the cloud," in Proceedings of the 2019 International Conference on Management of Data, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1743–1756. [Online]. Available: https://doi.org/10.1145/3299869.3314047

[13] A. Depoutovitch, C. Chen, J. Chen, P. Larson, S. Lin, J. Ng, W. Cui, Q. Liu, W. Huang, Y. Xiao, and Y. He, "Taurus database: How to be fast, available, and frugal in the cloud," in Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1463–1478. [Online]. Available: https://doi.org/10.1145/3318464.3386129

[14] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan, Z. Liu, F. Zhu, and T. Zhang, POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database. USA: USENIX Association, 2020, p. 29–42.

[15] D. J. DeWitt, A. Halverson, R. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling, "Split query processing in polybase," in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1255–1266. [Online]. Available: https://doi.org/10.1145/2463676.2463709

[16] A. Modi, K. Rajan, S. Thimmaiah, P. Jain, S. Mann, A. Agarwal, A. Shetty, S. K. I, A. Gosalia, and P. Sarthi, "New query optimization techniques in the spark engine of azure synapse," Proc. VLDB Endow., vol. 15, no. 4, p. 936–948, apr 2022. [Online]. Available: https://doi.org/10.14778/3503585.3503601

[17] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner, "Presto: Sql on everything," 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1802–1813, 2019.

[18] X. Yu, M. Youill, M. Woicik, A. Ghanem, M. Serafini, A. Aboulnaga, and M. Stonebraker, "Pushdowndb: Accelerating a dbms using s3 computation," in 2020 IEEE 36th International Conference on Data Engineering (ICDE), 2020, pp. 1802–1805.

[19] Y. Yang, M. Youill, M. Woicik, Y. Liu, X. Yu, M. Serafini, A. Aboulnaga, and M. Stonebraker, "Flexpushdowndb: Hybrid pushdown and caching in a cloud dbms," Proc. VLDB Endow., vol. 14, no. 11, p. 2101–2113, jul 2021. [Online]. Available: https://doi.org/10.14778/3476249.3476265

[20] D. Koutsoukos, R. Bhartia, A. Klimovic, and G. Alonso, "How to use persistent memory in your database," 2021. [Online]. Available: https://arxiv.org/abs/2112.00425

[21] B. Daase, L. J. Bollmeier, L. Benson, and T. Rabl, "Maximizing persistent memory bandwidth utilization for olap workloads," in Proceedings of the 2021 International Conference on Management of Data, ser. SIGMOD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 339–351. [Online]. Available: https://doi.org/10.1145/3448016.3457292

[22] M. Andrei, C. Lemke, G. Radestock, R. Schulze, C. Thiel, R. Blanco, A. Meghlan, M. Sharique, S. Seifert, S. Vishnoi, D. Booss, T. Peh, I. Schreter, W. Thesing, M. Wagle, and T. Willhalm, "Sap hana adoption of non-volatile memory," Proc. VLDB Endow., vol. 10, no. 12, p. 1754–1765, aug 2017. [Online]. Available: https://doi.org/10.14778/3137765.3137780

[23] A. van Renen, V. Leis, A. Kemper, T. Neumann, T. Hashida, K. Oe, Y. Doi, L. Harada, and M. Sato, "Managing non-volatile memory in database systems," in Proceedings of the 2018 International Conference on Management of Data, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1541–1555. [Online]. Available: https://doi.org/10.1145/3183713.3196897

[24] G. Liu, L. Chen, and S. Chen, "Zen: A high-throughput log-free oltp engine for non-volatile main memory," Proc. VLDB Endow., vol. 14, no. 5, p. 835–848, jan 2021. [Online]. Available: https://doi.org/10.14778/3446095.3446105

[25] J. Arulraj and A. Pavlo, "How to build a non-volatile memory database management system," in Proceedings of the 2017 ACM International Conference on Management of Data, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1753–1758. [Online]. Available: https://doi.org/10.1145/3035918.3054780

[26] Arulraj, Joy and Pavlo, Andrew, "Non-volatile memory database management systems," vol. 11, no. 1. Morgan & Claypool Publishers, 2019, pp. 1–191.

[27] J. Arulraj, A. Pavlo, and S. R. Dulloor, "Let's talk about storage and recovery methods for non-volatile memory database systems," in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 707–722. [Online]. Available: https://doi.org/10.1145/2723372.2749441

[28] J. Arulraj, A. Pavlo, and K. T. Malladi, "Multi-tier buffer management and storage system design for non-volatile memory," 2019. [Online]. Available: https://arxiv.org/abs/1901.10938

[29] J. Arulraj, M. Perron, and A. Pavlo, "Write-behind logging," Proc. VLDB Endow., vol. 10, no. 4, p. 337–348, nov 2016. [Online]. Available: https://doi.org/10.14778/3025111.3025116

[30] A. Chatzistergiou, M. Cintra, and S. D. Viglas, "Rewind: recovery write-ahead system for in-memory non-volatile data-structures," Proc. VLDB Endow., vol. 8, no. 5, p. 497–508, jan 2015. [Online]. Available: https://doi.org/10.14778/2735479.2735483

[31] C. Craft, "Persistent memory in exadata x8m," 2020. [Online]. Available: https://blogs.oracle.com/exadata/post/persistent-memory-in-exadata-x8m

[32] T. Ziegler, C. Binnig, and V. Leis, "Scalestore: A fast and cost-efficient storage engine using dram, nvme, and rdma," in Proceedings of the 2022 International Conference on Management of Data, 2022, pp. 685–699.

[33] X. Zhou, J. Arulraj, A. Pavlo, and D. Cohen, "Spitfire: A three-tier buffer manager for volatile and non-volatile memory," in Proceedings of the 2021 International Conference on Management of Data, ser. SIGMOD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2195–2207. [Online]. Available: https://doi.org/10.1145/3448016.3452819

[34] M. Saxena, M. A. Shah, S. Harizopoulos, M. M. Swift, and A. Merchant, "Hathi: durable transactions for memory using flash," in Proceedings of the Eighth International Workshop on Data Management on New Hardware, 2012, pp. 33–38.

[35] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian, "The End of Slow Networks: It's Time for a Redesign," Proc. VLDB Endow., vol. 9, no. 7, p. 528–539, mar 2016. [Online]. Available: https://doi.org/10.14778/2904483.2904485

[36] T. Wang and R. Johnson, "Scalable logging through emerging non-volatile memory," Proc. VLDB Endow., vol. 7, no. 10, p. 865–876, jun 2014. [Online]. Available: https://doi.org/10.14778/2732951.2732960

[37] J. Arulraj, M. Perron, and A. Pavlo, "Write-behind logging," Proc. VLDB Endow., vol. 10, no. 4, p. 337–348, nov 2016. [Online]. Available: https://doi.org/10.14778/3025111.3025116

[38] S. Chen and Q. Jin, "Persistent b(+)-trees in non-volatile main memory," Proc. VLDB Endow., vol. 8, no. 7, p. 786–797, feb 2015. [Online]. Available: https://doi.org/10.14778/2752939.2752947

[39] D. Hwang, W.-H. Kim, Y. Won, and B. Nam, "Endurable transient inconsistency in byte-addressable persistent b+-tree," in Proceedings of the 16th USENIX Conference on File and Storage Technologies, ser. FAST'18. USA: USENIX Association, 2018, p. 187–200.

[40] S. Chen and Q. Jin, "Persistent b+-trees in non-volatile main memory," Proc. VLDB Endow., vol. 8, no. 7, p. 786–797, feb 2015. [Online]. Available: https://doi.org/10.14778/2752939.2752947

[41] M. Nam, H. Cha, Y.-R. Choi, S. H. Noh, and B. Nam, "Write-optimized dynamic hashing for persistent memory," in Proceedings of the 17th USENIX Conference on File and Storage Technologies, ser. FAST'19. USA: USENIX Association, 2019, p. 31–44.

[42] P. Zuo, Y. Hua, and J. Wu, "Write-optimized and high-performance hashing index scheme for persistent memory," in Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'18. USA: USENIX Association, 2018, p. 461–476.

[43] B. Lu, X. Hao, T. Wang, and E. Lo, "Dash: Scalable hashing on persistent memory," Proc. VLDB Endow., vol. 13, no. 8, p. 1147–1161, apr 2020. [Online]. Available: https://doi.org/10.14778/3389133.3389134

[44] Lu, Baotong and Hao, Xiangpeng and Wang, Tianzheng and Lo, Eric, "Scaling Dynamic Hash Tables on Real Persistent Memory," SIGMOD Rec., vol. 50, no. 1, p. 87–94, jun 2021. [Online]. Available: https://doi.org/10.1145/3471485.3471506

[45] L. Lersch, X. Hao, I. Oukid, T. Wang, and T. Willhalm, "Evaluating persistent memory range indexes," Proc. VLDB Endow., vol. 13, no. 4, p. 574–587, dec 2019. [Online]. Available: https://doi.org/10.14778/3372716.3372728

[46] Lu, Baotong and Ding, Jialin and Lo, Eric and Minhas, Umar Farooq and Wang, Tianzheng, "APEX: A High-Performance Learned Index on Persistent Memory," Proc. VLDB Endow., vol. 15, no. 3, p. 597–610, nov 2021. [Online]. Available: https://doi.org/10.14778/3494124.3494141

[47] G. Psaropoulos, I. Oukid, T. Legler, N. May, and A. Ailamaki, "Bridging the latency gap between nvm and dram for latency-bound operations," in Proceedings of the 15th International Workshop on Data Management on New Hardware, ser. DaMoN'19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3329785.3329917

[48] Y. Wu, K. Park, R. Sen, B. Kroth, and J. Do, "Lessons learned from the early performance evaluation of intel optane dc persistent memory in dbms," in Proceedings of the 16th International Workshop on Data Management on New Hardware, ser. DaMoN '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3399666.3399898

[49] M. Böther, O. Kißig, L. Benson, and T. Rabl, "Drop it in like it's hot: An analysis of persistent memory as a drop-in replacement for nvme ssds," in Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021), ser. DAMON'21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3465998.3466010

[50] B. Daase, L. J. Bollmeier, L. Benson, and T. Rabl, "Maximizing persistent memory bandwidth utilization for olap workloads," in Proceedings of the 2021 International Conference on Management of Data, ser. SIGMOD/PODS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 339–351. [Online]. Available: https://doi.org/10.1145/3448016.3457292

[51] A. Shanbhag, N. Tatbul, D. Cohen, and S. Madden, "Large-scale in-memory analytics on intel(®) optane(™) dc persistent memory," in Proceedings of the 16th International Workshop on Data Management on New Hardware, ser. DaMoN '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3399666.3399933

[52] D. Cohen, T. Talpey, A. Kanevsky, U. Cummings, M. Krause, R. Recio, D. Crupnicoff, L. Dickman, and P. Grun, "Remote direct memory access over the converged enhanced ethernet fabric: Evaluating the options," in 2009 17th ieee symposium on high performance interconnects. IEEE, 2009, pp. 123–130.

[53] Z. He, D. Wang, B. Fu, K. Tan, B. Hua, Z.-L. Zhang, and K. Zheng, "Masq: Rdma for virtual private cloud," in Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, 2020, pp. 1–14.

[54] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur et al., "Memcached design on high performance rdma capable interconnects," in 2011 International Conference on Parallel Processing. IEEE, 2011, pp. 743–752.

[55] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). Seattle, WA: USENIX Association, Apr. 2014, pp. 401–414. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi{\'c}

[56] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro, "No compromises: Distributed transactions with consistency, availability, and performance," in Proceedings of the 25th Symposium on Operating Systems Principles, ser. SOSP '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 54–70. [Online]. Available: https://doi.org/10.1145/2815400.2815425

[57] A. Shamis, M. Renzelmann, S. Novakovic, G. Chatzopoulos, A. Dragojević, D. Narayanan, and M. Castro, "Fast general distributed transactions with opacity," in Proceedings of the 2019 International Conference on Management of Data, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 433–448. [Online]. Available: https://doi.org/10.1145/3299869.3300069

[58] C. Buragohain, K. M. Risvik, P. Brett, M. Castro, W. Cho, J. Cowhig, N. Gloy, K. Kalyanaraman, R. Khanna, J. Pao, M. Renzelmann, A. Shamis, T. Tan, and S. Zheng, "A1: A distributed in-memory graph database," in Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 329–344. [Online]. Available: https://doi.org/10.1145/3318464.3386135

[59] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, scalable and simple distributed transactions with Two-Sided (RDMA) datagram RPCs," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). Savannah, GA: USENIX Association, Nov. 2016, pp. 185–201. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/kalia

[60] X. Wei, Z. Dong, R. Chen, and H. Chen, "Deconstructing RDMA-enabled distributed transactions: Hybrid is better!" in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). Carlsbad, CA: USENIX Association, Oct. 2018, pp. 233–251. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/wei

[61] S. Novakovic, Y. Shan, A. Kolli, M. Cui, Y. Zhang, H. Eran, B. Pismenny, L. Liss, M. Wei, D. Tsafrir, and M. Aguilera, "Storm: A Fast Transactional Dataplane for Remote Data Structures," in Proceedings of the 12th ACM International Conference on Systems and Storage, ser. SYSTOR '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 97–108. [Online]. Available: https://doi.org/10.1145/3319647.3325827

[62] P. W. Frey, R. Goncalves, M. Kersten, and J. Teubner, "Spinning relations: High-speed networks for distributed join processing," in Proceedings of the Fifth International Workshop on Data Management on New Hardware, ser. DaMoN '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 27–33. [Online]. Available: https://doi.org/10.1145/1565694.1565701

[63] C. Barthels, S. Loesing, G. Alonso, and D. Kossmann, "Rack-scale in-memory join processing using rdma," in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1463–1475. [Online]. Available: https://doi.org/10.1145/2723372.2750547

[64] W. Rödiger, S. Idicula, A. Kemper, and T. Neumann, "Flow-join: Adaptive skew handling for distributed joins over high-speed networks," in 2016 IEEE 32nd International Conference on Data Engineering (ICDE), 2016, pp. 1194–1205.

[65] W. Rödiger, T. Mühlbauer, A. Kemper, and T. Neumann, "High-speed query processing over high-speed networks," Proc. VLDB Endow., vol. 9, no. 4, p. 228–239, dec 2015. [Online]. Available: https://doi.org/10.14778/2856318.2856319

[66] F. Liu, L. Yin, and S. Blanas, "Design and evaluation of an rdma-aware data shuffling operator for parallel database systems," in Proceedings of the Twelfth European Conference on Computer Systems, ser. EuroSys '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 48–63. [Online]. Available: https://doi.org/10.1145/3064176.3064202

[67] P. Fent, A. v. Renen, A. Kipf, V. Leis, T. Neumann, and A. Kemper, "Low-latency communication for fast dbms using rdma and shared memory," in 2020 IEEE 36th International Conference on Data Engineering (ICDE), 2020, pp. 1477–1488.

[68] L. Thostrup, J. Skrzypczak, M. Jasny, T. Ziegler, and C. Binnig, DFI: The Data Flow Interface for High-Speed Networks. New York, NY, USA: Association for Computing Machinery, 2021, p. 1825–1837. [Online]. Available: https://doi.org/10.1145/3448016.3452816

[69] G. Alonso, C. Binnig, I. Pandis, K. Salem, J. Skrzypczak, R. Stutsman, L. Thostrup, T. Wang, Z. Wang, and T. Ziegler, "Dpi: the data processing interface for modern networks," CIDR 2019 Online Proceedings, p. 11, 2019.

[70] N. May, A. Böhm, and W. Lehner, "Sap hana – the evolution of an in-memory dbms from pure olap processing towards mixed workloads," in Datenbanksysteme für Business, Technologie und Web (BTW 2017), 2017, pp. 545–546.

[71] IBM, "IBM DB2 pureScale," https://www.ibm.com/docs/en/db2/10.5?topic=editions-introduction-db2-purescale-environment, (Accessed on 2022-10-05).

[72] F. Li, S. Das, M. Syamala, and V. R. Narasayya, "Accelerating Relational Databases by Leveraging Remote Memory and RDMA," in Proceedings of the 2016 International Conference on Management of Data, ser. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 355–370. [Online]. Available: https://doi.org/10.1145/2882903.2882949

[73] W. Cao, Y. Zhang, X. Yang, F. Li, S. Wang, Q. Hu, X. Cheng, Z. Chen, Z. Liu, J. Fang, B. Wang, Y. Wang, H. Sun, Z. Yang, Z. Cheng, S. Chen, J. Wu, W. Hu, J. Zhao, Y. Gao, S. Cai, Y. Zhang, and J. Tong, PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. New York, NY, USA: Association for Computing Machinery, 2021, p. 2477–2489. [Online]. Available: https://doi.org/10.1145/3448016.3457560

[74] Oracle, "Under the Hood of an Exadata Transaction," http://www.adms-conf.org/2021-camera-ready/jia_presentation.pdf, 2021 (Accessed on 2022-05-25).

[75] D. K. Gifford, "Weighted voting for replicated data," in Proceedings of the Seventh ACM Symposium on Operating Systems Principles, ser. SOSP '79. New York, NY, USA: Association for Computing Machinery, 1979, p. 150–162. [Online]. Available: https://doi.org/10.1145/800215.806583

[76] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "Aries: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," ACM Transactions on Database Systems (TODS), vol. 17, no. 1, pp. 94–162, 1992.

[77] W. Cao, Y. Zhang, X. Yang, F. Li, S. Wang, Q. Hu, X. Cheng, Z. Chen, Z. Liu, J. Fang, B. Wang, Y. Wang, H. Sun, Z. Yang, Z. Cheng, S. Chen, J. Wu, W. Hu, J. Zhao, Y. Gao, S. Cai, Y. Zhang, and J. Tong, "Polardb serverless: A cloud native database for disaggregated data centers," in Proceedings of the 2021 International Conference on Management of Data, ser. SIGMOD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2477–2489. [Online]. Available: https://doi.org/10.1145/3448016.3457560

[78] C. Douglas, "Rdma with pmem," 2015.

[79] Intel, "Intel® Data Direct I/O Technology," https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html, (Accessed on 2022-05-25).

[80] S. T. Leutenegger and D. Dias, "A modeling study of the tpc-c benchmark," ACM Sigmod Record, vol. 22, no. 2, pp. 22–31, 1993.

[81] Cole, Richard and Funke, Florian and Giakoumakis, Leo and Guy, Wey and Kemper, Alfons and Krompass, Stefan and Kuno, Harumi and Nambiar, Raghunath and Neumann, Thomas and Poess, Meikel and others, "The mixed workload ch-benchmark," in Proceedings of the Fourth International Workshop on Testing Database Systems, 2011, pp. 1–6.

[82] A. Kopytov, "Sysbench: a system performance benchmark," http://sysbench. sourceforge. net/, 2004.

[83] "Intel's Optane DIMM Price Model," https://thememoryguy.com/intels-optane-dimm-price-model/, (Accessed on 2022-10-10).