Leveraging Persistent Memory in Cloud-Native Database Systems

JOY ARULRAJ, *Georgia Institute of Technology*
DAVID COHEN, *Intel*

1 CLOUD-NATIVE DATABASE SYSTEMS

The availability of cost-effective, highly-available, performant cloud computing platforms (*e.g.*, Amazon Web Services, Microsoft Azure) over the last decade has given rise to a new class of cloud-native database management systems (DBMSs) [12]. These systems differ from their traditional counterparts in the following ways:

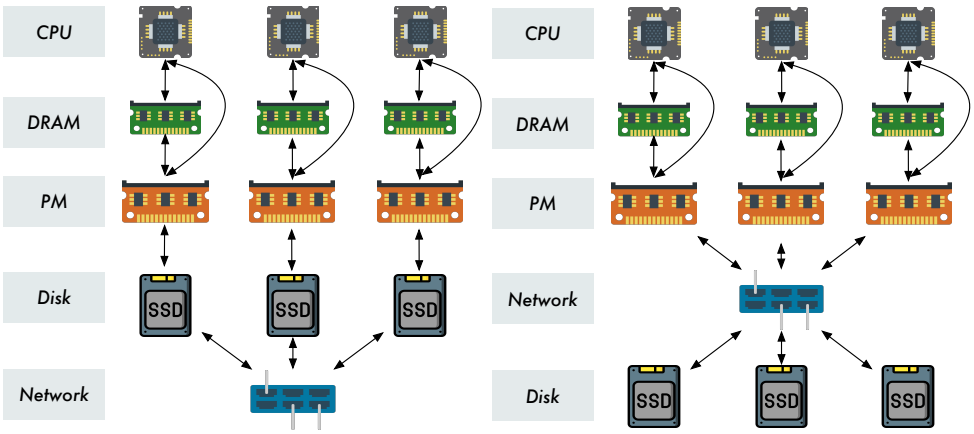
- **Hardware Disaggregation:** Cloud-native DBMSs enable independent scaling of compute, storage, and networking resources. They only consume as much hardware resources as is truly needed for the workload that they are serving.
- **Data Differentiation:** Data has immense value when it is created, but that value diminishes over time. Cloud-native DBMSs leverage this property by storing older data on less expensive storage technologies.
- **Shared-Storage Architecture:** Traditional DBMSs are typically based on a shared-nothing architecture [1, 10], as illustrated in Figure 1a. In this model, each compute node has its own set of storage devices. Cloud-native DBMSs instead tend to adopt a shared-disk model to decouple compute and storage resources as shown in Figure 1b. This allows them to scale these resources independently.
- **Deeper Storage Hierarchy:** The cloud opens up many more shared-storage options, including: (1) local persistent memory (PM) [4], (2) remote PM-based SSDs accessible via NVMe-over-Fabrics (NVMe-oF) [3], and (3) long-term cold storage. Each of these tiers exhibit different price-performance characteristics.

2 STORAGE ARCHITECTURE

Cloud-native DBMSs adopt a shared-disk model so that they can independently scale the compute and storage resources. The canonical storage hierarchy of these systems consists of the following tiers:

- **Local Persistent Memory:** The DBMS caches frequently-accessed data (*e.g.*, indices) on the PM devices that are directly attached to the compute server where the database instance is running. The CPU directly accesses these PM devices through `load` and `store`

Authors' addresses: Joy Arulraj, *Georgia Institute of Technology*; David Cohen, *Intel*.



(a) Shared-nothing architecture (b) Shared-disk architecture

Fig. 1. **DBMS Architectures** - Comparison of shared-nothing and shared-disk architectures.

assembly instructions [4, 11]. The DBMS uses the direct access (DAX) mechanism to bypass the traditional I/O stack (page cache and block layer). It manages the data on a file system extended for DAX-enabled PM (*e.g.*, XFS on a `fsdax` device [6]). DAX enables direct, byte-addressable access to the contents of the file system.

- **Remote SSDs:** Only the DBMS's working set resides on the compute server. The rest of the state resides on a collection of dis-aggregated storage devices attached to the compute server accesses via an NVMe-over-Fabric (NVMe-oF) interface. Cloud-native DBMSs often leverage distributed storage systems backed by these remote storage devices (*e.g.*, distributed shared log [12]). These systems are designed around the assumption that network bandwidth is plentiful and cheap.
- **Long-Term Cold Storage:** Database backups are infrequently used in cloud-native DBMSs due to their reliance on replication. So, these backups are migrated to a cheaper remote, long-term cold storage tier. They are only leveraged during disaster recovery. They help meet governance and compliance needs for long-term retention of data.

3 CASE STUDY: INTEL DATA MANAGEMENT PLATFORM

We now illustrate the design principles of cloud-native database systems through a case study. We present the architecture of the Data Management Platform (DMP) developed by Intel [5]. We later discuss how we extend a transactional DBMS to leverage PM on DMP.

Overview: DMP is a distributed, data management system that is geared towards diverse workloads (*e.g.*, transactional databases, machine learning pipelines). It manages a collection of dis-aggregated, containerized NVMe SSDs that are accessible via an NVMe-over-Fabric (NVMe-oF) interface [3]. The logical volumes residing in these containers are optimized for

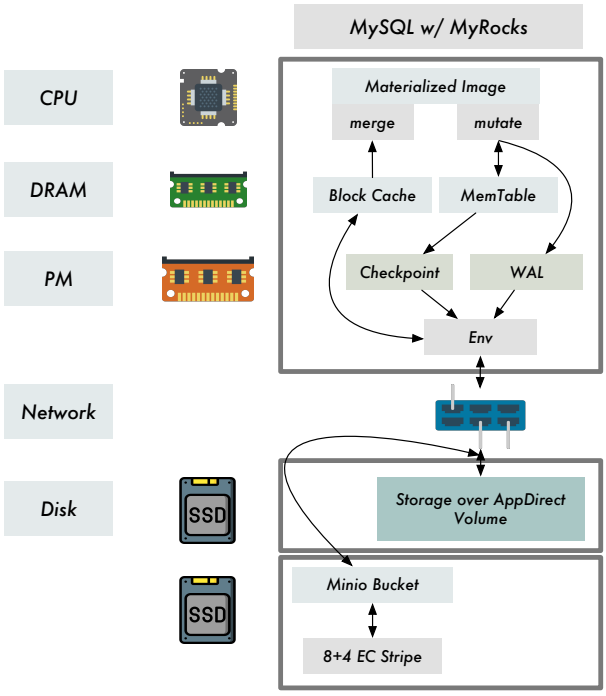


Fig. 2. **MySQL on Intel's Data Management Platform:** MySQL instance with a MyRocks storage engine hosted on Intel's Data Management Platform [2, 8].

sequential I/O and serve as a replacement for high-capacity SATA/SAS HDDs. DMP relies on top of two distributed storage systems that are backed by these remote volumes: (1) distributed object store, and (2) distributed shared log.

MySQL on DMP: We next discuss how DMP supports transactional database workloads. Consider a MySQL instance with a MyRocks storage engine hosted on DMP [2]. The MyRocks storage engine leverages an log-structured merge (LSM) tree for managing data, as shown in Figure 2. We tailor this engine to make use of the RocksDB-Cloud library [8]. A strongly consistent image of the database is maintained across the distributed shared log and the distributed object store.

Distributed Object Store: DMP continuously backs up data and meta-data to a distributed object store (*e.g.*, MinIO [7]) to provide high availability. We tailor the MyRocks engine so that it persists the immutable, lower-level tiers of the LSM tree in the object store. Each *sorted string table* (SSTable) is persisted as a separate object in a MinIO bucket. Since these objects are immutable, they can be effectively cached in the locally-attached PM device. Each object is encoded as an erasure-coded stripe consisting of 8 data and 4 parity blocks. To ensure crash-consistency, we extend RocksDB's transactional log (*i.e.*, local manifest) to include a *cloud manifest* that is maintained in the bucket. This log tracks all the LSM tree mutations

for the duration of the operations of this DBMS instance. This ensures that the cloud object store has a record of all the modifications applied to the database, thereby enabling support for point-in-time recovery.

Distributed Shared Log: In a traditional DBMS, the write-ahead log (WAL) is the source of truth while recovering from a failure. In a cloud-native DBMS, we generalize this idea to a distributed shared log (*i.e.*, an event stream) [12]. We could configure the WAL to use such a cloud event stream. However, the latency of immediately persisting entries to the distributed log is too expensive. So, as a stop-gap solution, the WAL is locally stored on PM. A replicated logging module that is tailored for PM would increase the write throughput achievable by the cloud-native DBMS.

PM-centric Optimizations: DMP enables the MyRocks storage engine to exploit the byte-addressability of PM devices by eschewing the page-centric optimizations inherent in other cloud-native DBMSs with minimal code modifications [9]. The MyRocks engine maintains its local state across the combination of DRAM and PM (XFS on a `fsdax` device [6]). Since the PM device exhibits memory-like performance, the page cache pages would be unnecessary copies of the data stored on that device. DAX eliminates the these extra copies by directly performing reads and writes to the PM device (configured in App Direct Mode). The MyRocks engine maps the PM device directly into userspace. It accesses 256 B chunks of an SSTable mapped into PM as opposed to loading the entire 4 KB page. The engine stores the tail of the WAL and caches the top-level tiers of the LSM tree on local PM. It stores the mutable memory tables and a few SSTables on DRAM.

We tune the RocksDB parameters to minimize the impact of flushing MemTables to new SSTable files and SSTable compaction operations. We disable the RocksDB block cache. Thus, the engine directly fetches the blocks from the appropriate SSTable file cached in the locally-attached PM volume (*i.e.*, Storage-over-AppDirect). We plan to leverage persistent skiplists in the future to guide the read operations through the LSM tree.

The MySQL cloud-native DBMS supports two key capabilities: (1) intra- and inter-cluster replication, and (2) point-in-time recovery (PiTR). It currently provides these capabilities by leveraging two logs: (1) the unmodified MySQL binary log (binlog), and (2) the MyRocks WAL. The unmodified binlog does not adhere to the principles of a cloud-native DBMSs. The I/O overhead associated with the binlog and the MySQL group commit effectively throttles achievable write throughput. We plan to address this issue in the future by developing a PM-aware, replicated tail-of-the-log module.

4 OPEN PROBLEMS

Several open problems arise with the advent of cloud-native DBMSs:

- How can we best leverage the compute-local, byte-addressable PM devices?
- How should data be migrated across nodes?
- How should the core DBMS protocols be tailored for the shared-disk model?

- How does NVMe-oF performance affect system throughput and latency metrics?
- How does the shared-disk model compare against the canonical shared-nothing model on performance metrics?

5 SUMMARY

Cloud-native DBMSs are moving away from the monolithic architecture of their traditional counterparts by decoupling storage and compute resources. The performance of these systems is, thus, constrained by I/Os written over the network. Locally-attached persistent memory and remote PM-based SSD devices accessible via an NVMe-over-Fabric interface help alleviate this bottleneck, as we illustrated through our case study of MyRocks storage engine on Intel's Data Management Platform. The advent of cloud-native DBMSs has given rise to several open problems that should be of interest to both researchers and practitioners in storage systems.

REFERENCES

[1] David DeWitt and Jim Gray. 1992. *Parallel database systems: The future of high performance database processing.* Technical Report. University of Wisconsin-Madison.

[2] Facebook. 2016. MyRocks: A space- and write-optimized MySQL database. https://engineering.fb.com/core-data/myrocks-a-space-and-write-optimized-mysql-database/.

[3] NVM Express Inc. 2016. NVM Express over Fabrics. https://nvmexpress.org/wp-content/uploads/NVMe_over_Fabrics_1_0_Gold_20160605-1.pdf.

[4] Intel. 2019. Intel Optane Memory. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html.

[5] Intel. 2019. Manage and Monetize Exponential Data Growth with Intel's Data Management Platform. https://itpeernetwork.intel.com/manage-monetize-exponential-data-growth-with-intels-data-management-platform/.

[6] Intel. 2019. Provision Intel Optane DC Persistent Memory. https://software.intel.com/en-us/articles/quick-start-guide-configure-intel-optane-dc-persistent-memory-on-linux.

[7] MinIO. 2019. High Performance Object Storage. https://min.io/.

[8] Rockset. 2019. RocksDB-Cloud: A Key-Value Store for Cloud Applications. https://github.com/rockset/rocksdb-cloud.

[9] Reza Sherkat, Colin Florendo, Mihnea Andrei, Rolando Blanco, Adrian Dragusanu, Amit Pathak, Pushkar Khadilkar, Neeraj Kulkarni, Christian Lemke, Sebastian Seifert, et al. 2019. Native store extension for SAP HANA. In *VLDB 2019.*

[10] Michael Stonebraker. 1986. The Case for Shared Nothing. *Data Engineering Bulletin* 1 (1986).

[11] Alexander van Renen, Viktor Leis, Alfons Kemper, Thomas Neumann, Takushi Hashida, Kazuichi Oe, Yoshiyasu Doi, Lilian Harada, and Mitsuru Sato. 2018. Managing non-volatile memory in database systems. In *SIGMOD 2018.*

[12] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *SIGMOD.*