Let's Talk About Storage & Recovery Methods for Non-Volatile Memory OLTP Database Systems

Andy Pavlo + Joy Arulraj
Carnegie Mellon University

Winter 2013: First Blood

- Initial evaluation of existing DBMSs on Intel NVM SDV
- Results published ADMS@VLDB'14

A Prolegomenon on OLTP Database Systems for Non-Volatile Memory

Justin DeBrabant
Brown University
debrabant@cs.brown.edu

Joy Arulraj
Carnegie Mellon University
jarulraj@cs.cmu.edu

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

Subramanya R. Dulloor
Intel Labs
subramanya.r.dulloor@intel.com

ABSTRACT

The design of a database management system's (DBMS) architecture is predicated on the target storage hierarchy. Traditional disk-oriented systems use a two-level hierarchy, with fast volatile memory used for caching, and slower, durable device used for primary storage. As such, these systems use a buffer pool and complex concurrency control schemes to mask disk latencies. Compare this to main memory DBMSs that assume all data can reside in DRAM, and thus do not need these components.

But emerging non-volatile memory (NVM) technologies require us to rethink this dichotomy. Such memory devices are slightly slower than DRAM, but all writes are persistent, even after power loss. We explore two possible use cases of NVM for on-line transaction processing (OLTP) DBMSs. The first is where NVM completely replaces DRAM and the other is where NVM and DRAM coexist in the system. For each case, we compare the performance of a disk-oriented DBMS with a memory-oriented DBMS using two OLTP benchmarks. We also evaluate the performance of different recovery algorithms on these NVM devices. Our evaluation shows that in both storage hierarchies, memory-oriented systems are able to outperform their disk-oriented counterparts. However, as skew decreases the performance of the two architectures converge, showing that neither architecture is ideally suited for an NVM-based storage hierarchy.

1. INTRODUCTION

Disk-oriented DBMSs are based on the same hardware assumptions that were made in 1970s with the original relational DBMSs. The architectures of these systems use a two-level storage hierarchy: (1) a fast but volatile byte-addressable memory for caching (i.e., DRAM) and (2) a slow, non-volatile block-addressable device for permanent storage (i.e., HDD or SSD). These systems take a pessimistic assumption that a transaction could access data that is not in memory and thus will incur a long delay while a OS retrieves the data needed from disk. They employ a heavyweight concurrency control scheme that allows multiple transactions to safely run at the same time; when one transaction stalls because of the disk, another transaction can continue execution. This requires the use of buffer pools and complex transaction serialization schemes.

Recent advances in manufacturing technologies have greatly increased the capacity of DRAM available on a single computer. But disk-oriented systems were not designed for the case where most, if not all, of the data resides entirely in memory. The result is that many of their legacy components have been shown to impede their scalability for OLTP workloads [15]. In contrast, the architecture of main-memory DBMSs assume that all data fits in main memory and thus are able to remove the slower, disk-oriented components from the system. As such, they have been shown to outperform disk-oriented DBMSs [27]. TimesTen [2] was an early example of this approach, and newer systems include H-Store [16], Hekaton [10], and HyPer [17].

In some OLTP applications, however, the database can grow to be larger than the amount of memory available to the DBMS. Although one could partition a database across multiple machines so that the aggregate memory is sufficient, this can increase the number of multi-node transactions in some applications and thereby degrade their performance [23]. Recent work has explored adding back slower disk-based storage in main memory DBMSs as a place to store "cold" data, thereby freeing up in-memory storage [9, 26]. These techniques exploit the skewed access patterns of OLTP workloads to support databases that exceed the memory capacity of the DBMS while still providing the performance advantages of a memory-oriented system.

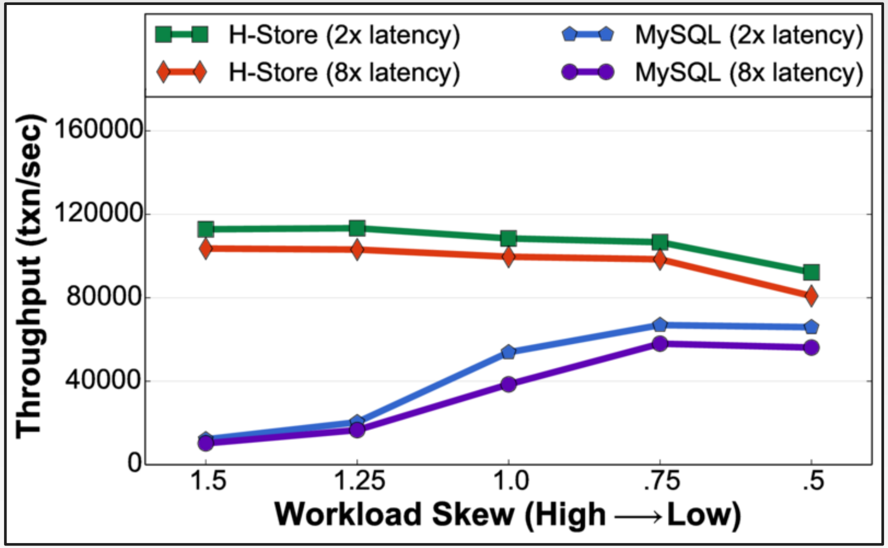
The advent of *non-volatile memory* (NVM)[1] offers an intriguing blend of the two storage mediums. NVM is a broad class of technologies, including phase-change memory [25] and memristors [28], that provide low latency reads and writes on the same order of magnitude as DRAM but with persistent writes like an SSD. Researchers also speculate that NVM will have much higher storage densities than what is possible with current DRAM devices. A comparison of performance characteristics of non-volatile memory technologies relative to DRAM and Flash is shown in Table 1. Such low-latency, high-capacity NVM storage has the potential to significantly change the design of DBMS architectures [6]. It is unclear, however, which DBMS architecture is ideal for NVM or whether a new architecture design is necessary.

Given this outlook, this paper presents our initial foray into the use of NVM in OLTP DBMSs. We test several DBMS architectures on an experimental, hardware-based NVM emulator and explore their trade-offs using two OLTP benchmarks. The read and

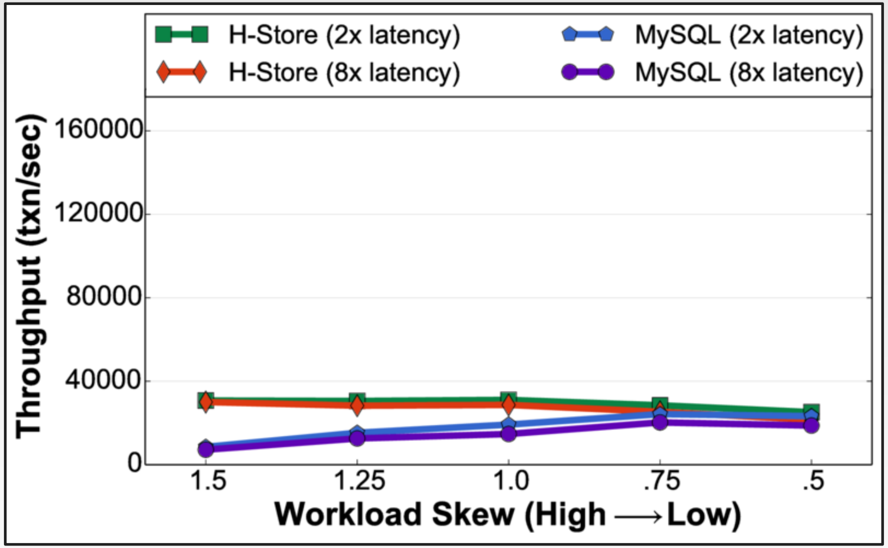
Proceedings of the VLDB Endowment, Vol. 7, No. 14

[1] NVM is also referred to as *storage-class memory* or *persistent memory*.

MySQL vs. H-Store



90% Reads / 10% Writes

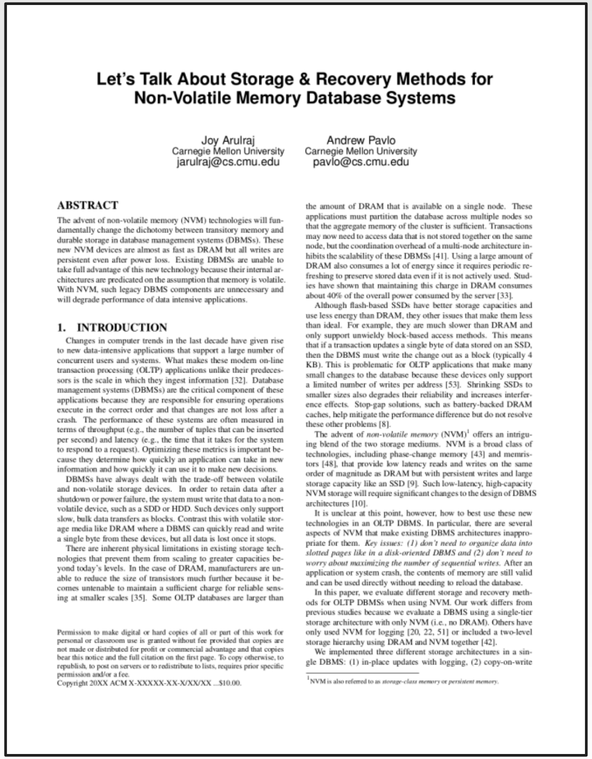


50% Reads / 50% Writes

ISTC BIG DATA

Summer 2014: First Blood, Part II

- Evaluate storage and recovery methods for NVM.
- Preparing SIGMOD'14 submission.



Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems

Joy Arulraj
Carnegie Mellon University
jarulraj@cs.cmu.edu

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

ABSTRACT

The advent of non-volatile memory (NVM) technologies will fundamentally change the dichotomy between transitory memory and durable storage in database management systems (DBMSs). These new NVM devices are almost as fast as DRAM but all writes are persistent even after power loss. Existing DBMSs are unable to take full advantage of this new technology because their internal architectures are predicated on the assumption that memory is volatile. With NVM, such legacy DBMS components are unnecessary and will degrade performance of data intensive applications.

1. INTRODUCTION

Changes in computer trends in the last decade have given rise to new data-intensive applications that support a large number of concurrent users and systems. What makes these modern on-line transaction processing (OLTP) applications unlike their predecessors is the scale in which they ingest information [32]. Database management systems (DBMSs) are the critical component of these applications because they are responsible for ensuring operations execute in the correct order and that changes are not loss after a crash. The performance of these systems are often measured in terms of throughput (e.g., the number of tuples that can be inserted per second) and latency (e.g., the time that it takes for the system to respond to a request). Optimizing these metrics is important because they determine how quickly an application can take in new information and how quickly it can use it to make new decisions.

DBMSs have always dealt with the trade-off between volatile and non-volatile storage devices. In order to retain data after a shutdown or power failure, the system must write that data to a non-volatile device, such as a SDD or HDD. Such devices only support slow, bulk data transfers as blocks. Contrast this with volatile storage media like DRAM where a DBMS can quickly read and write a single byte from these devices, but all data is lost once it stops.

There are inherent physical limitations in existing storage technologies that prevent them from scaling to greater capacities beyond today's levels. In the case of DRAM, manufacturers are unable to reduce the size of transistors much further because it becomes untenable to maintain a sufficient charge for reliable sensing at smaller scales [35]. Some OLTP databases are larger than the amount of DRAM that is available on a single node. These applications must partition the database across multiple nodes so that the aggregate memory of the cluster is sufficient. Transactions may now need to access data that is not stored together on the same node, but the coordination overhead of a multi-node architecture inhibits the scalability of these DBMSs [41]. Using a large amount of DRAM also consumes a lot of energy since it requires periodic refreshing to preserve stored data even if it is not actively used. Studies have shown that maintaining this charge in DRAM consumes about 40% of the overall power consumed by the server [33].

Although flash-based SSDs have better storage capacities and use less energy than DRAM, they other issues that make them less than ideal. For example, they are much slower than DRAM and only support unwieldy block-based access methods. This means that if a transaction updates a single byte of data stored on an SSD, then the DBMS must write the change out as a block (typically 4 KB). This is problematic for OLTP applications that make many small changes to the database because these devices only support a limited number of writes per address [53]. Shrinking SSDs to smaller sizes also degrades their reliability and increases interference effects. Stop-gap solutions, such as battery-backed DRAM caches, help mitigate the performance difference but do not resolve these other problems [8].

The advent of *non-volatile memory* (NVM)[1] offers an intriguing blend of the two storage mediums. NVM is a broad class of technologies, including phase-change memory [43] and memristors [48], that provide low latency reads and writes on the same order of magnitude as DRAM but with persistent writes and large storage capacity like an SSD [9]. Such low-latency, high-capacity NVM storage will require significant changes to the design of DBMS architectures [10].

It is unclear at this point, however, how to best use these new technologies in an OLTP DBMS. In particular, there are several aspects of NVM that make existing DBMS architectures inappropriate for them. *Key issues: (1) don't need to organize data into slotted pages like in a disk-oriented DBMS and (2) don't need to worry about maximizing the number of sequential writes.* After an application or system crash, the contents of memory are still valid and can be used directly without needing to reload the database.

In this paper, we evaluate different storage and recovery methods for OLTP DBMSs when using NVM. Our work differs from previous studies because we evaluate a DBMS using a single-tier storage architecture with only NVM (i.e., no DRAM). Others have only used NVM for logging [20, 22, 51] or included a two-level storage hierarchy using DRAM and NVM together [42].

We implemented three different storage architectures in a single DBMS: (1) in-place updates with logging, (2) copy-on-write

[1] NVM is also referred to as *storage-class memory* or *persistent memory*.

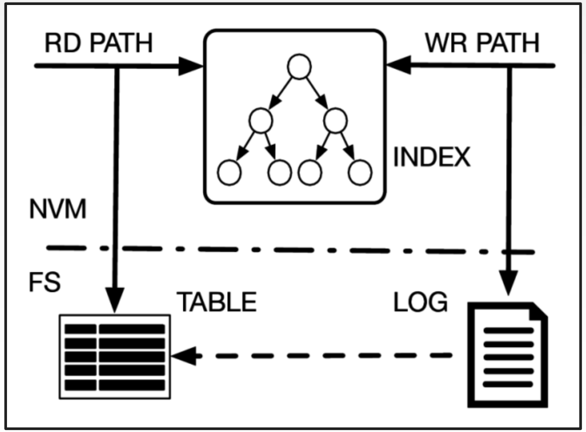
DBMS Testbed

- Custom lightweight DBMS.
  - *Uses NUMA & PMFS interfaces.*
  - *No volatile DRAM.*
- Partition-based locking CC.
- Pluggable architecture:
  - *Supports different storage engines.*

ISTC BIG DATA

Engine #1 – In-place Updates

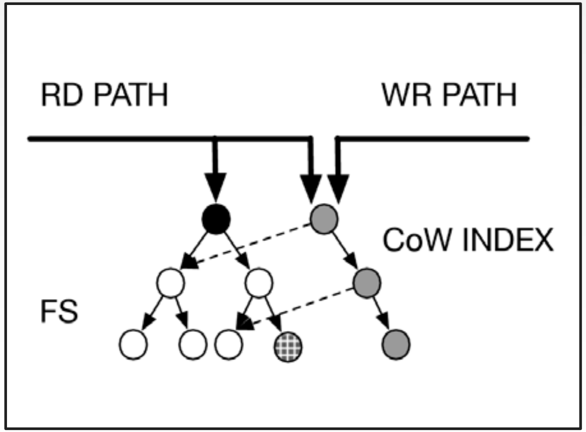
- Apply change to tuples directly.
  - *VoltDB with ARIES.*
  - *Table storage + write-ahead log.*
  - *STX B+Tree*



ISTC BIG DATA

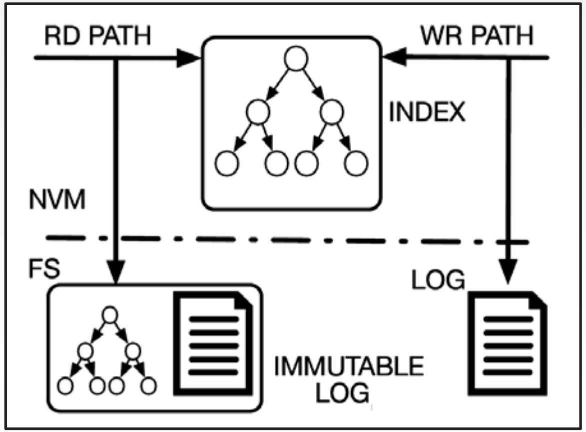
Engine #2 – Copy-on-Write Updates

- Make new copy before updating:
  - *Shadow paging using LMDB Persistent B+Tree.*
  - *No logging.*
  - *Background garbage collection.*



Engine #3 – Log-based Updates

- Changes only written to log.
  - *Based on LevelDB's LSM.*
  - *No table storage.*
  - *Background level compaction.*



Storage Engines

	Table Storage	Logging	Example
In-Place	**Yes**	**Yes**	VoltDB
Copy-on-Write	**Yes**	**No**	LMDB
Log-based	**No**	**Yes**	LevelDB

ISTC BIG DATA

NVM Optimized Engines

- Refactored engines to be "pointer-oriented".
- Extended Intel's *libpmem* allocation library.
  - *Added arena-based allocation.*
  - *Significantly improved throughput.*

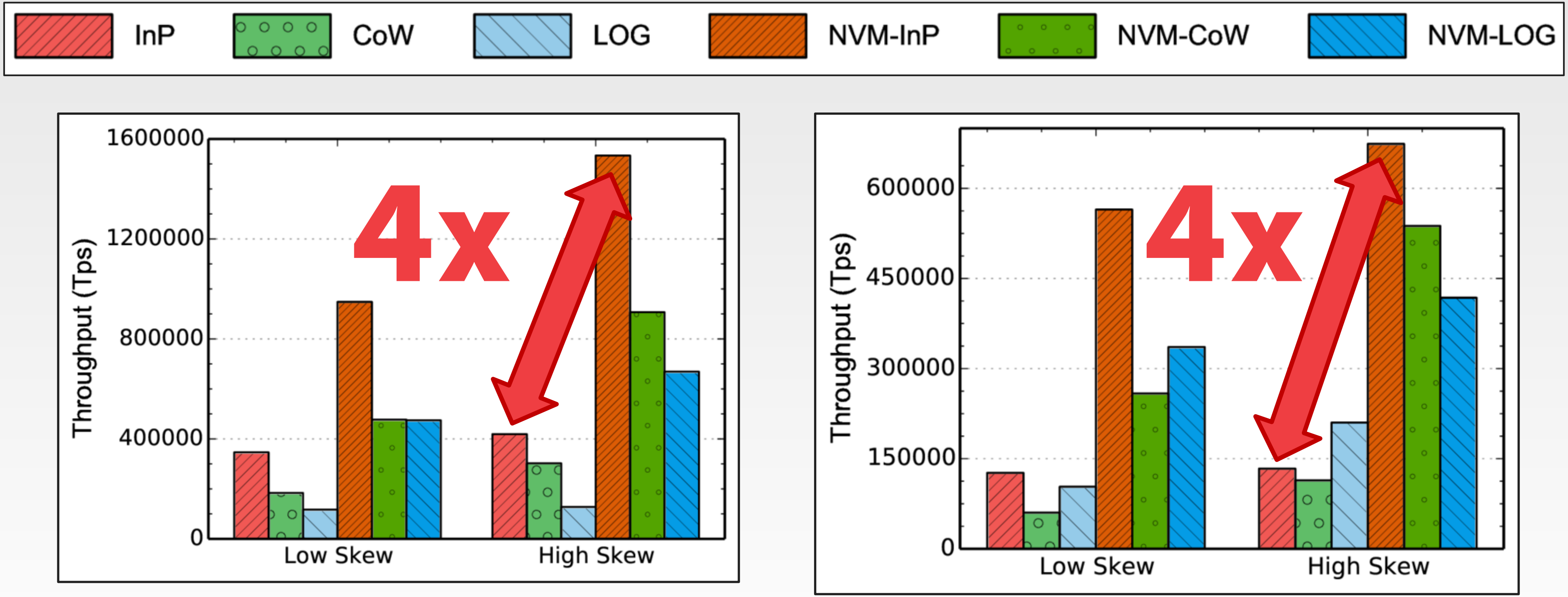
Experimental Evaluation

- Yahoo! Cloud Serving Benchmark:
  - *2 million records (~2GB)*
  - *Two workload mixtures*
  - *Two skew settings*
  - *1 million transactions*

Experimental Evaluation

- NVM Latency Configuration:
  - *2x DRAM (~200ns)*
  - *8x results not shown.*
- 8 partitions on 8 cores.

Throughput

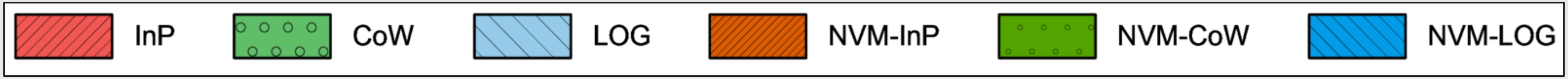


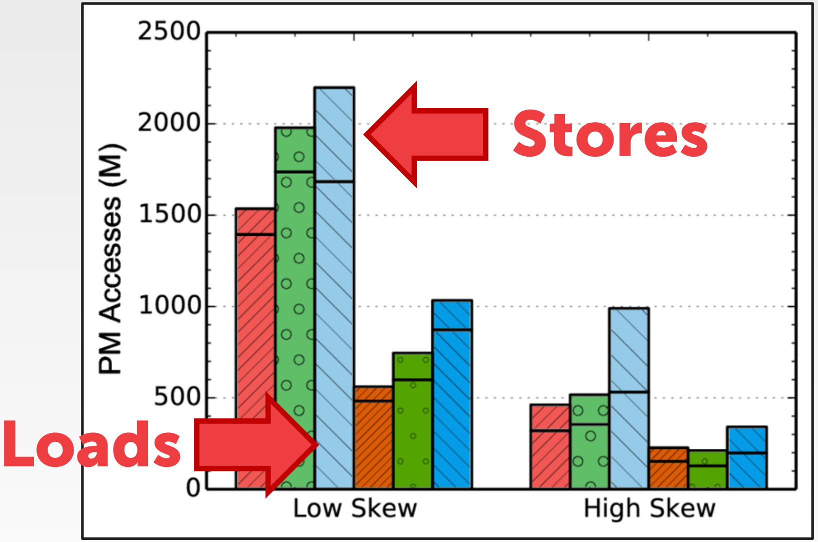
90% Reads / 10% Writes

50% Reads / 50% Writes

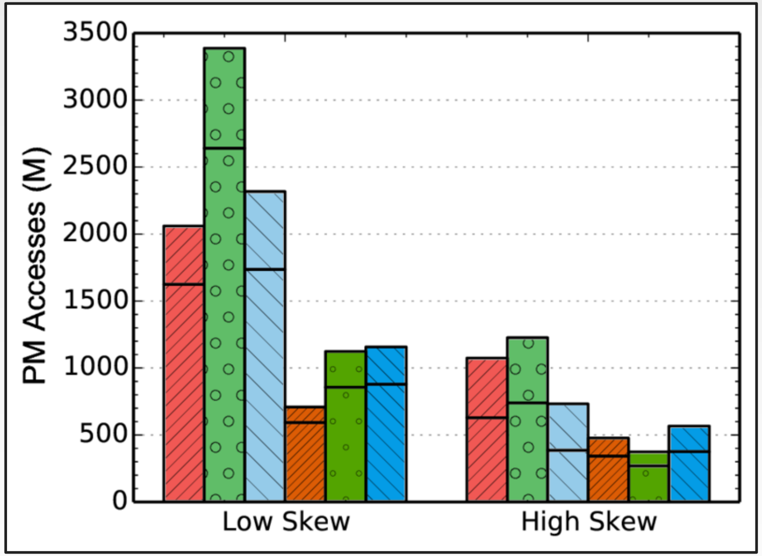
ISTC BIG DATA

NVM Reads/Writes





90% Reads / 10% Writes

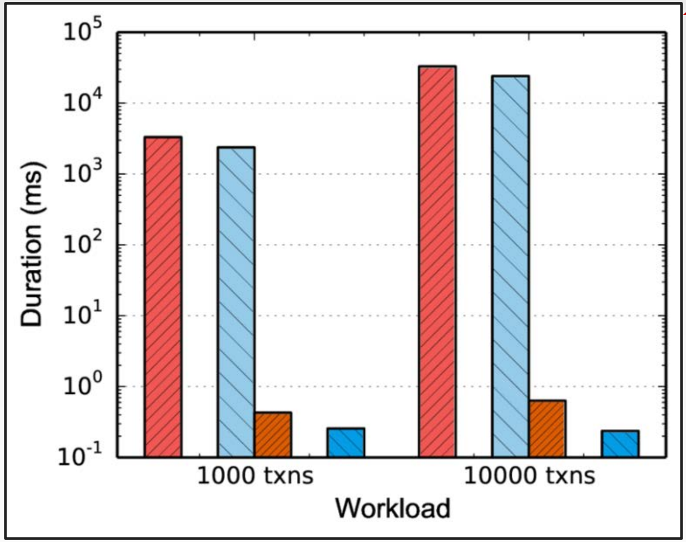


50% Reads / 50% Writes

ISTC BIG DATA

Recovery Time

Zero Recovery



Discussion

- NVM engines outperforms "traditional" engines:
  - *Higher throughput*
  - *Reduced wear on device.*
- In-place performs best overall.

ISTC BIG DATA

N-STORE

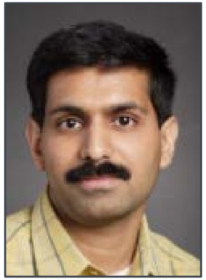
nstore.cs.cmu.edu

Fall 2014: N-Store

- First DBMS for NVM-only operating environment.
- OLTP/OLAP hybrid
  - *Column-store that supports fast in-place updates.*
- Indexing + Many-Core

Justin **DeBrabant**

Joy **Arulraj**

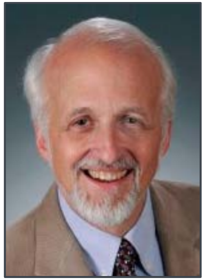


Rajesh **Sankaran**

Subramanya **Dulloor**

Andy **Pavlo**

Mike **Stonebraker**



Col. Stan **Zdonik**

Jeff **Parkhurst**

intel · BROWN · Carnegie Mellon University · Portland State University

END

@ANDY_PAVLO