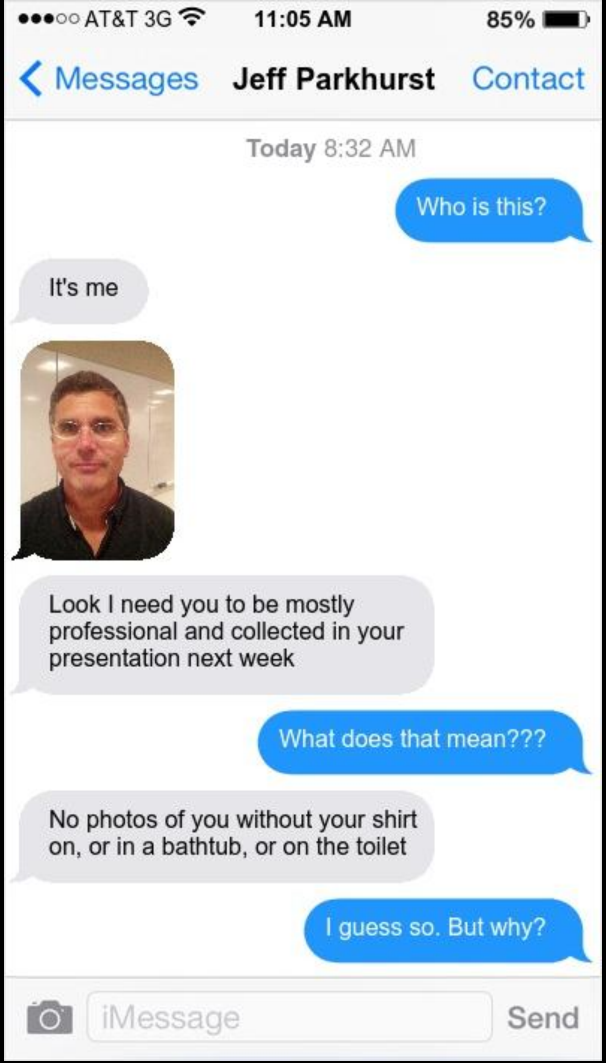


Jeff Parkhurst

Today 8:32 AM

Who is this?

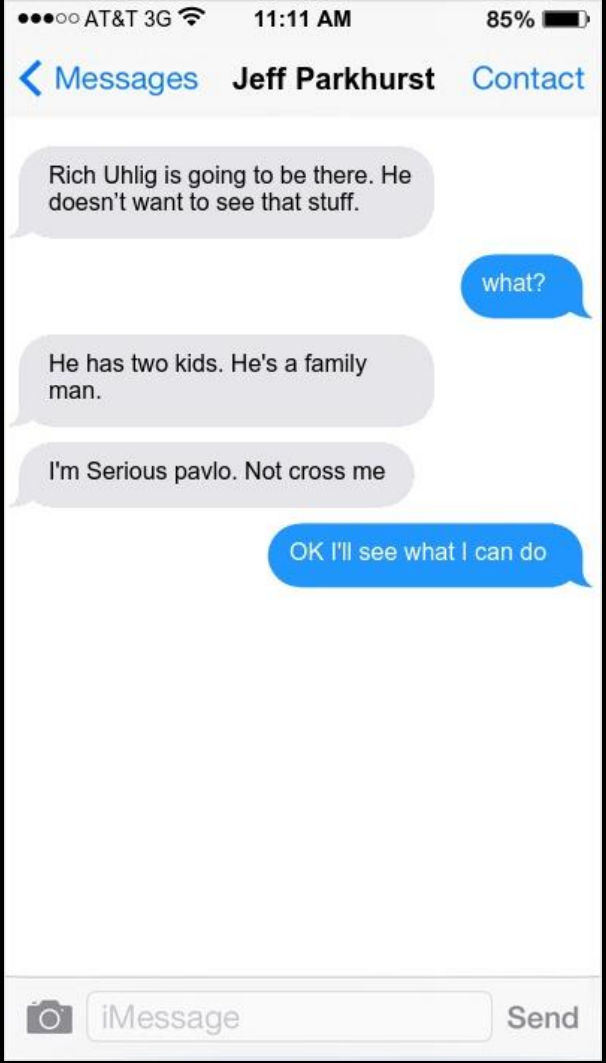
It's me

Look I need you to be mostly professional and collected in your presentation next week

What does that mean???

No photos of you without your shirt on, or in a bathtub, or on the toilet

I guess so. But why?



Jeff Parkhurst

Rich Uhlig is going to be there. He doesn't want to see that stuff.

what?

He has two kids. He's a family man.

I'm Serious pavlo. Not cross me

OK I'll see what I can do

ISTC BIG DATA

Andy Pavlo's

Mostly Professional and Collected Presentation about

Getting Down & Dirty in OLTP Databases with Intel's NVM SDV

September 25, 2015

intel MIT BROWN Carnegie Mellon University W UT Portland State University ISTC BIG DATA

The Story So Far

2014: Comparing Existing DBMSs

2015: Evaluating Storage Architectures

2014: Existing DBMSs

Comparison of disk vs. main-memory DBMSs running on Intel NVM SDV.

Found that logging is (still) the main bottleneck in both systems.

Paper: 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at ADMS'14, a workshop co-located with the 40th Intl. Conf. on Very Large Data Bases, September 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 14
Copyright 2014 VLDB Endowment 2150-8097/14/10.

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

2015: Storage Architectures

Evaluated storage and recovery methods for OLTP DBMSs.

Developed NVM-optimized methods that achieve 5.5x better throughput with 2x fewer writes.

Paper: SIGMOD'15

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

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

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

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

ABSTRACT

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

To better understand these issues, we implemented three engines in a modular DBMS testbed that are based on different storage management architectures: (1) in-place updates, (2) copy-on-write updates, and (3) log-structured updates. We then present NVM-aware variants of these architectures that leverage the persistence and byte-addressability properties of NVM in their storage and recovery methods. Our experimental evaluation on an NVM hardware emulator shows that these engines achieve up to 5.5× higher throughput than their traditional counterparts while reducing the amount of wear due to write operations by up to 2×. We also demonstrate that our NVM-aware recovery protocols allow these engines to recover almost instantaneously after the DBMS restarts.

1. INTRODUCTION

Changes in computer trends have given rise to new on-line transaction processing (OLTP) applications that support a large number of concurrent users and systems. What makes these modern applications unlike their predecessors is the scale in which they ingest information [41]. Database management systems (DBMSs) are the critical component of these applications because they are responsible for ensuring transactions' operations execute in the correct order and that their changes are not lost after a crash. Optimizing the DBMS's performance is important because it determines how quickly an application can take in new information and how quickly it can use it to make new decisions. This performance is affected by how fast the system can read and write data from storage.

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

In addition, there are inherent physical limitations that prevent DRAM from scaling to capacities beyond today's levels [46]. Using a large amount of DRAM also consumes a lot of energy since it requires periodic refreshing to preserve data even if it is not actively used. Studies have shown that DRAM consumes about 40% of the overall power consumed by a server [42].

Although flash-based SSDs have better storage capacities and use less energy than DRAM, they have 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 [66]. 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 [11].

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 [55], memristors [60], and STT-MRAM [26] that provide low latency reads and writes on the same order of magnitude as DRAM, but with persistent writes and large storage capacity like a SSD [13]. Table 1 compares the characteristics of NVM with other storage technologies.

It is unclear at this point, however, how to best leverage these new technologies in a DBMS. There are several aspects of NVM that make existing DBMS architectures inappropriate for them [14, 23]. For example, disk-oriented DBMSs (e.g., Oracle RDBMS, IBM DB2, MySQL) are predicated on using block-oriented devices for durable storage that are slow at random access. As such, they maintain an in-memory cache for blocks of tuples and try to maximize the amount of sequential reads and writes to storage. In the case of memory-oriented DBMSs (e.g., VoltDB, MemSQL), they contain certain components to overcome the volatility of DRAM. Such components may be unnecessary in a system with byte-addressable NVM with fast random access.

In this paper, we evaluate different storage and recovery methods for OLTP DBMSs from the ground-up, starting with an NVM-only storage hierarchy. We implemented three storage engine architectures in a single DBMS: (1) in-place updates with logging, (2) copy-on-write updates without logging, and (3) log-structured updates.

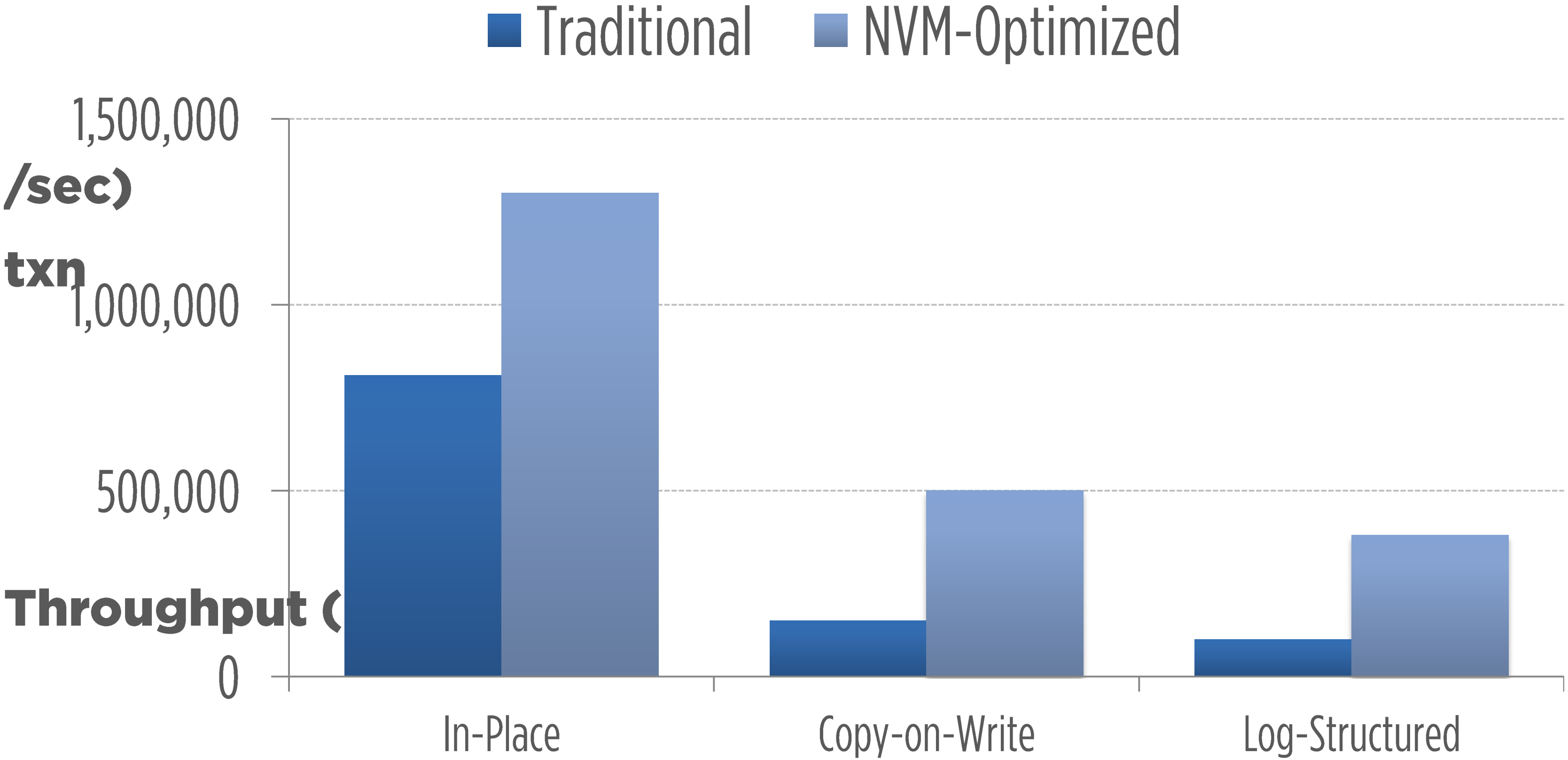
[1] NVM is also referred to as storage-class memory or persistent memory.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2749441.

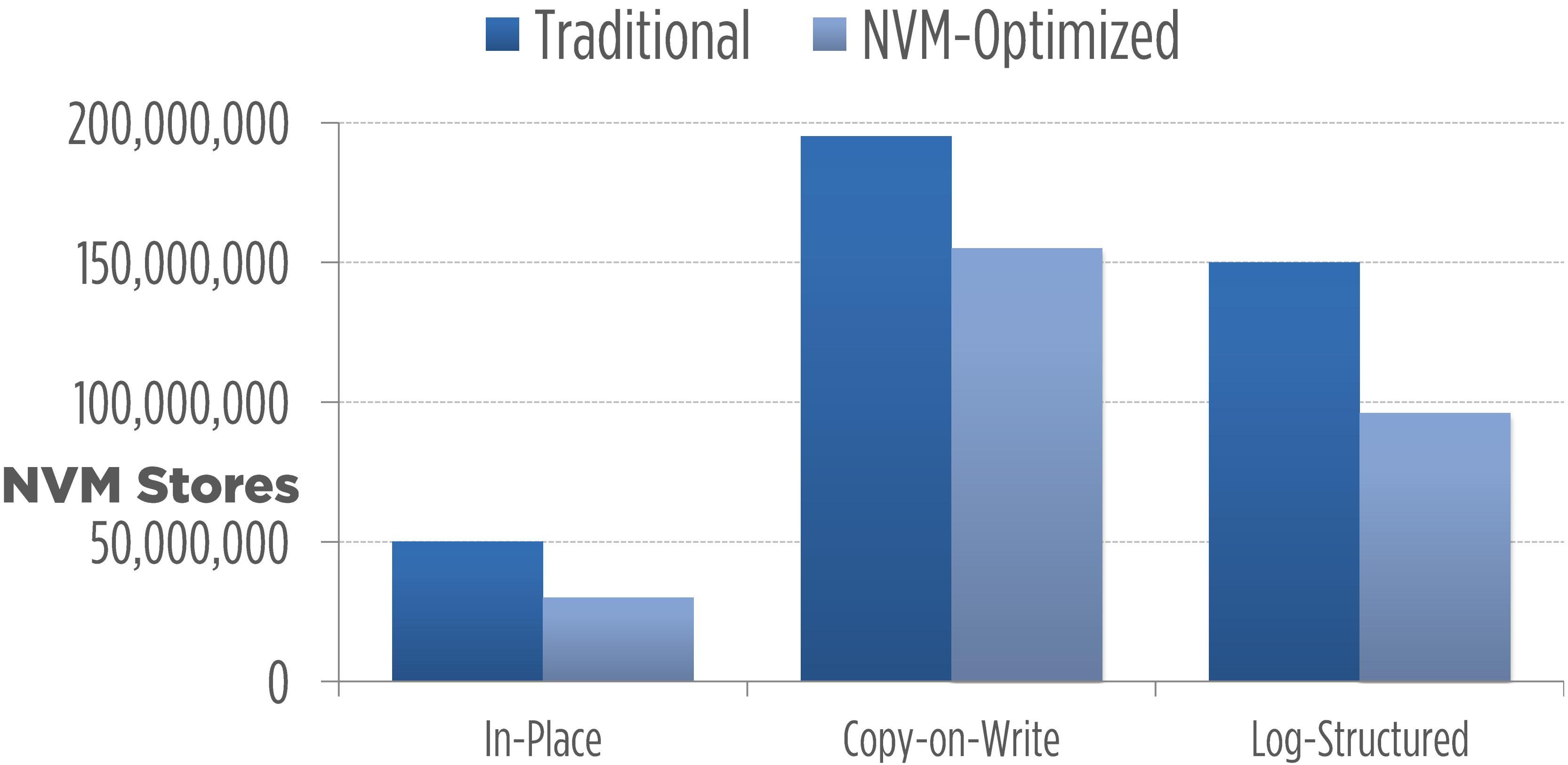
2015: Storage Architectures

	Table Storage	Logging	Example
In-Place	Yes	Yes	VOLTDB
Copy-on-Write	Yes	No	LMDB
Log-based	No	Yes	RocksDB

YCSB :: 10/90 RW :: 2x Latency



YCSB :: 10/90 RW :: 2x Latency



ISTC BIG DATA

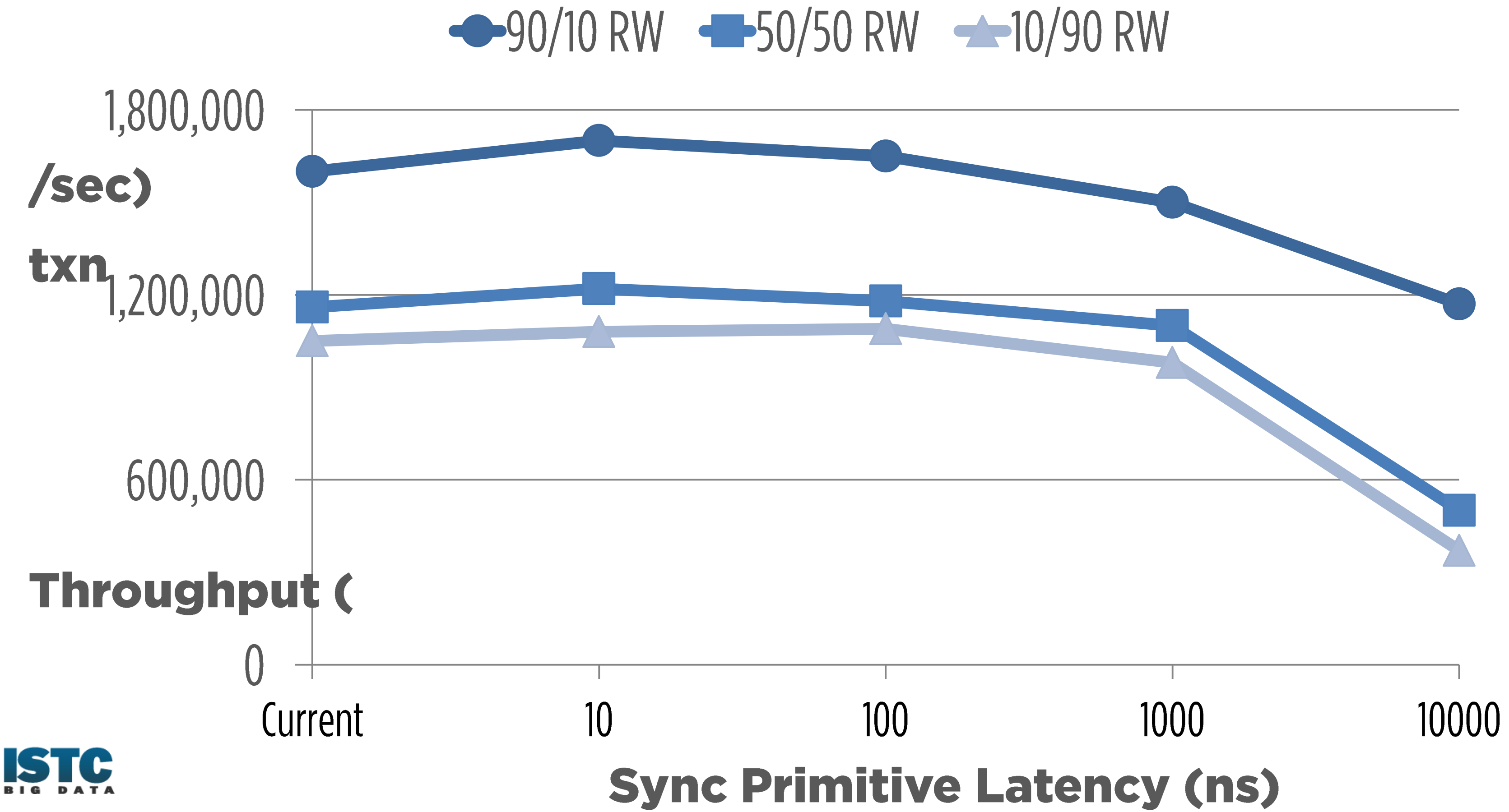
PCOMMIT Evaluation

Weakly-ordered sync primitive that retains data in the flushed cached lines.

Emulated with **RDTSC** and **PAUSE** instructions on NVM SDV.

Summer 2015: ~10,000 **PCOMMIT** invocations per second per CPU core.

YCSB // In-Place Engine



New Stuff

NVM vs. SSD

Multi-level Anti-Caching

DRAM+NVM storage manager

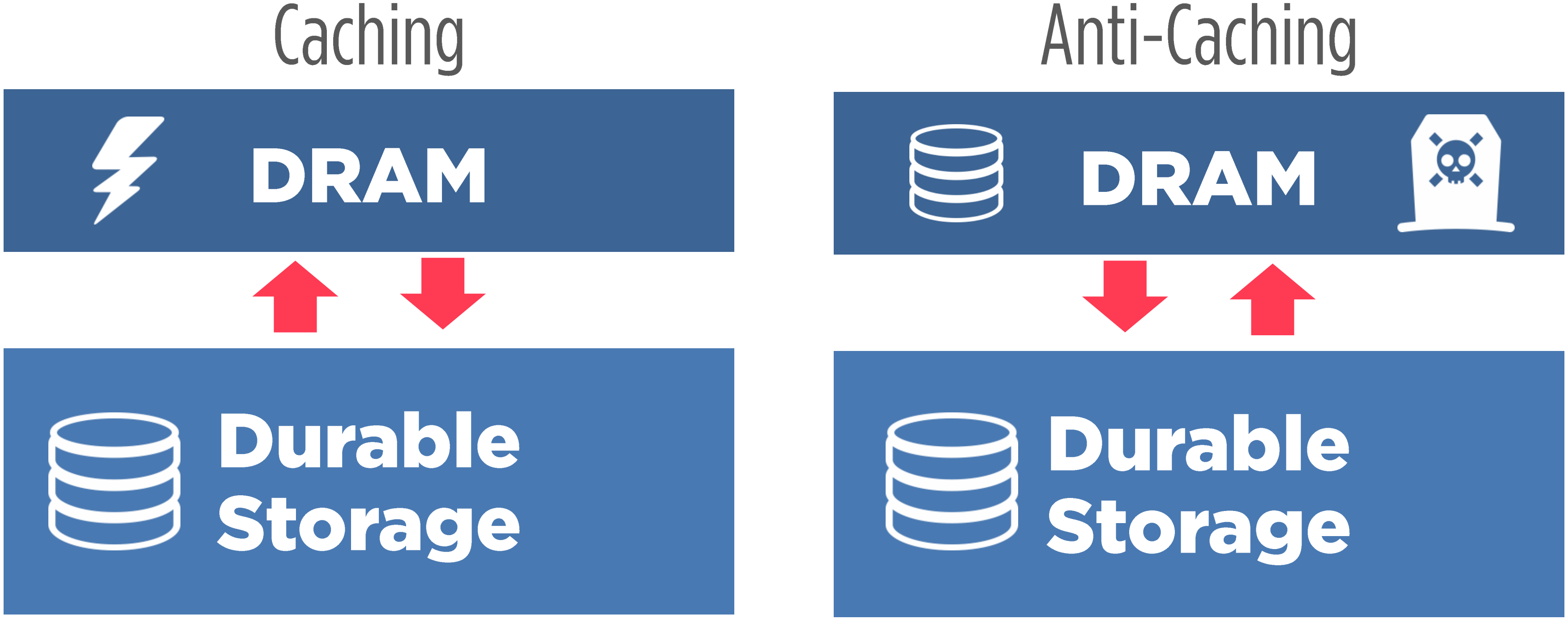
NVM vs. SSD

Two-level Storage Hierarchy

Disk-oriented vs. Memory-Oriented

- *Caching (MySQL)*
- *Anti-caching (H-Store)*

Disk vs. Memory Oriented DBMSs



Caching

DRAM

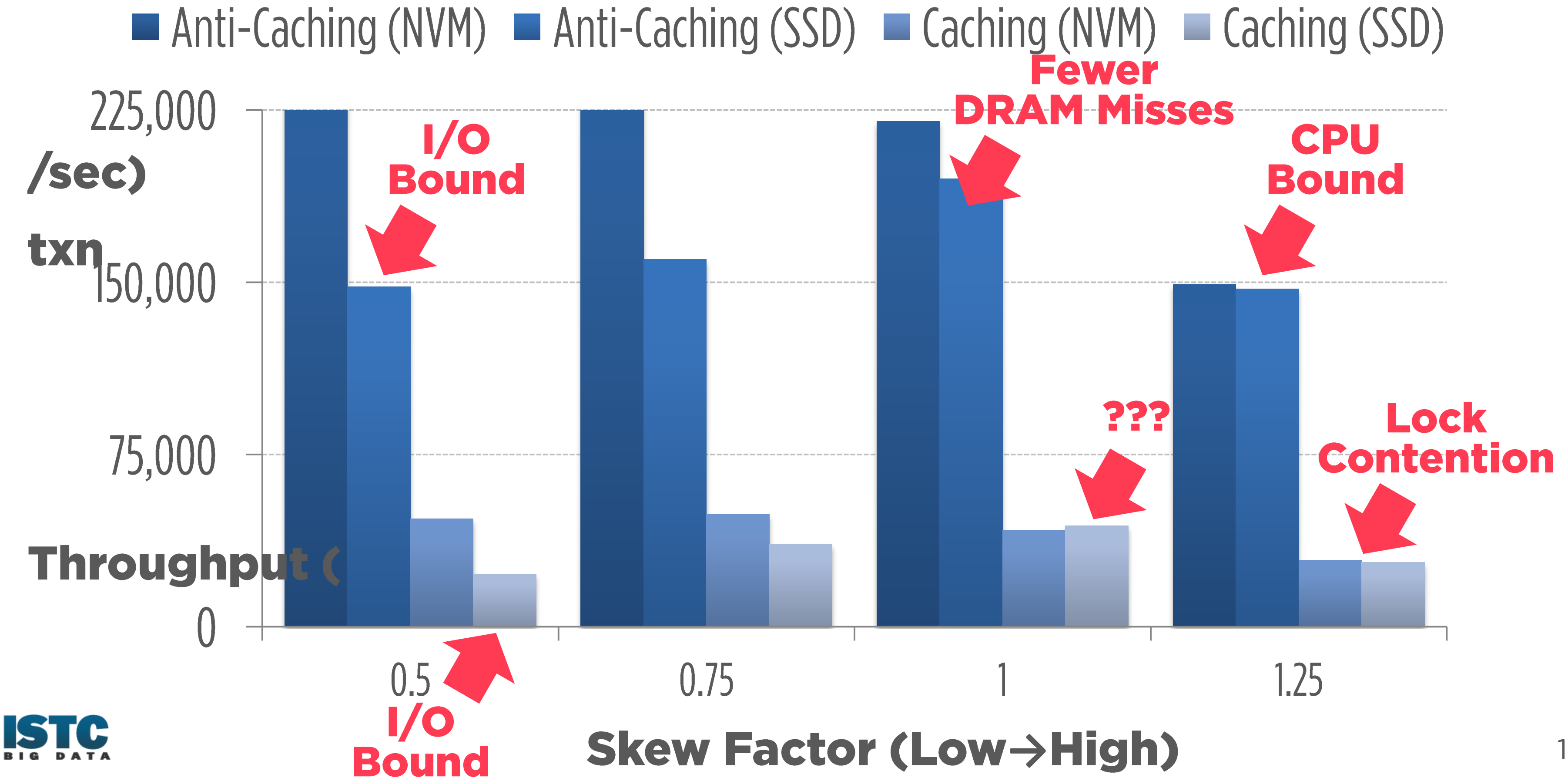
Durable Storage

Anti-Caching

DRAM

Durable Storage

YCSB :: 90/10 RW :: 4x Latency



I/O Bound

Fewer DRAM Misses

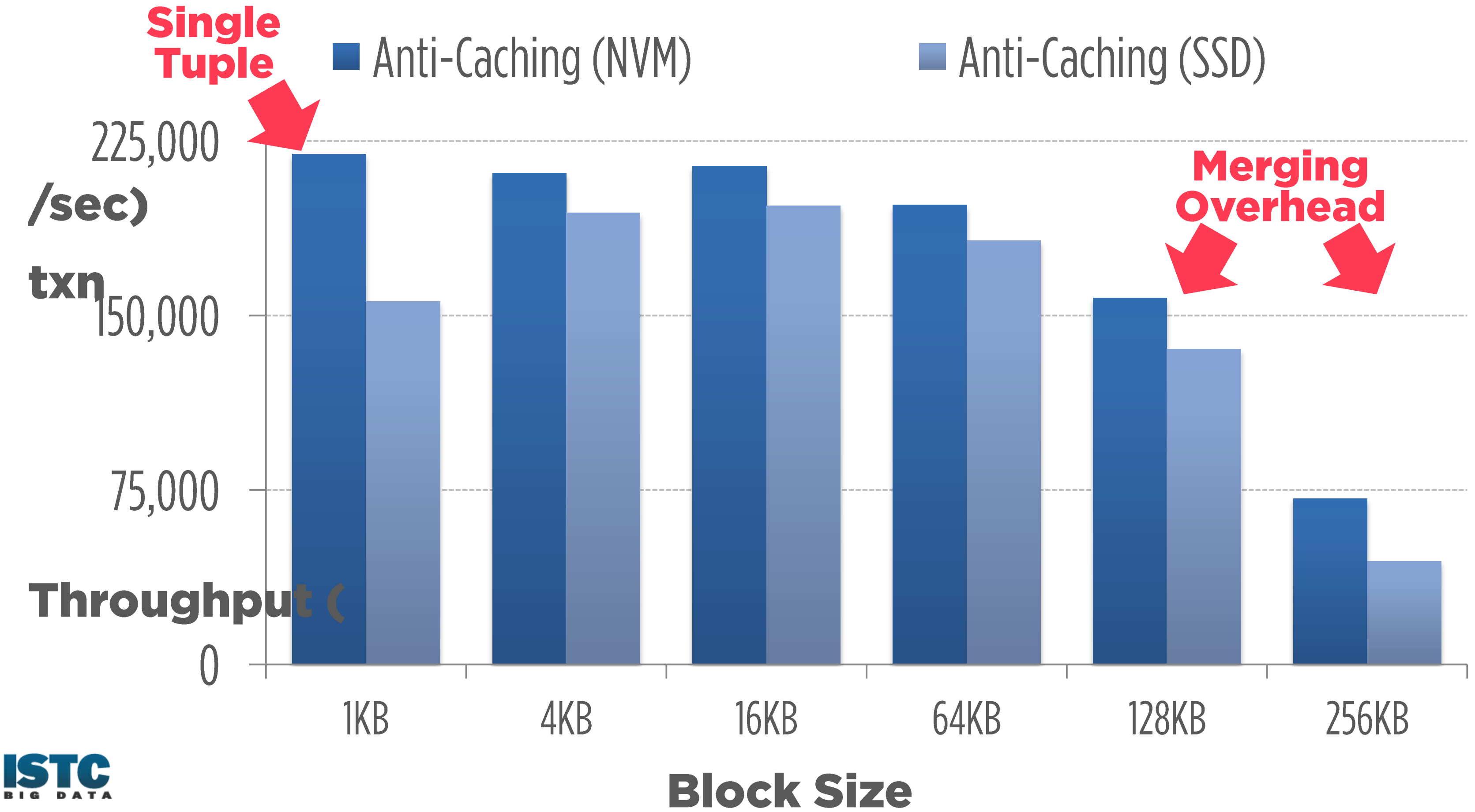
CPU Bound

???

Lock Contention

I/O Bound

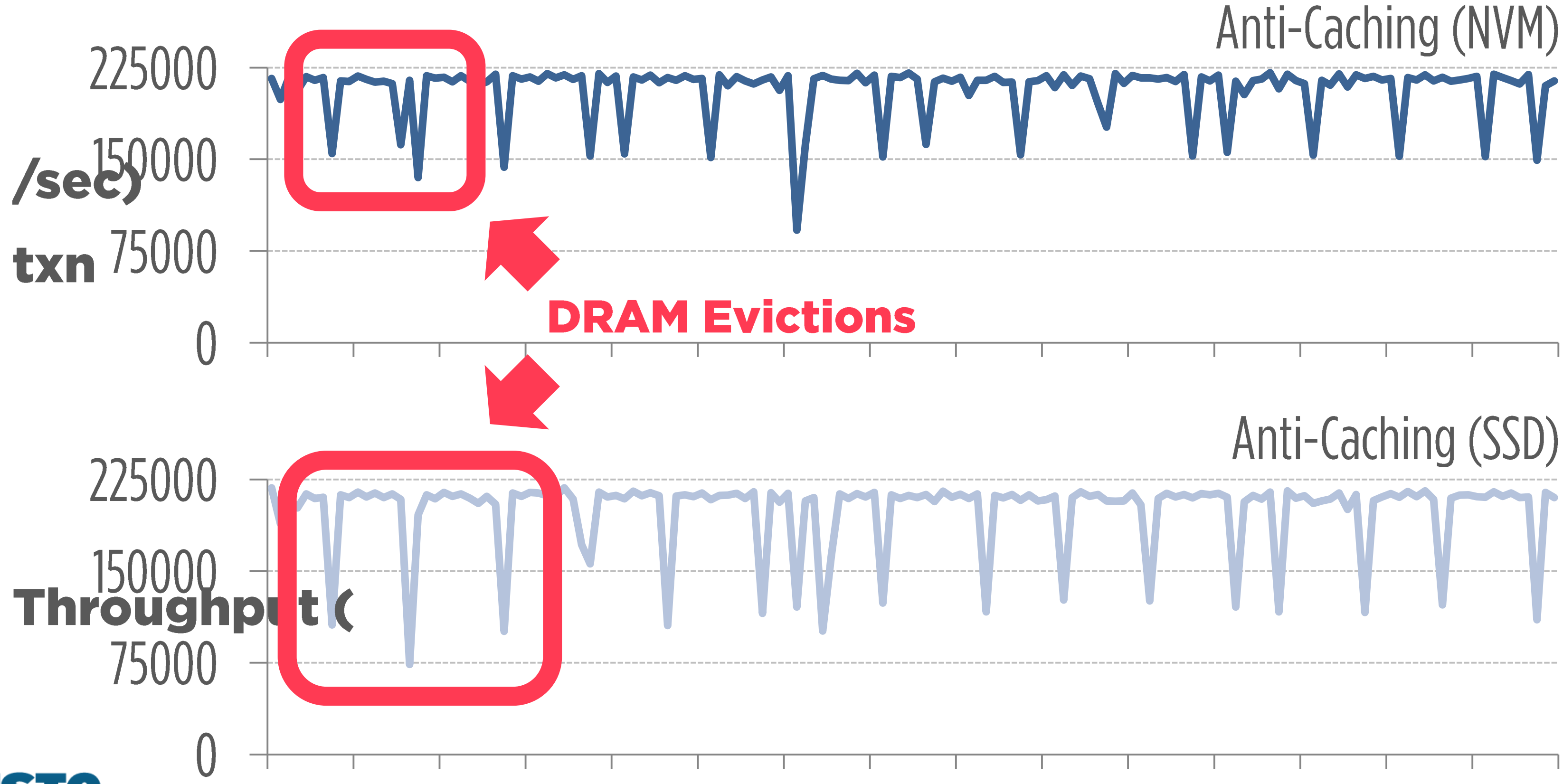
YCSB :: Byte-Addressable Access



Single Tuple

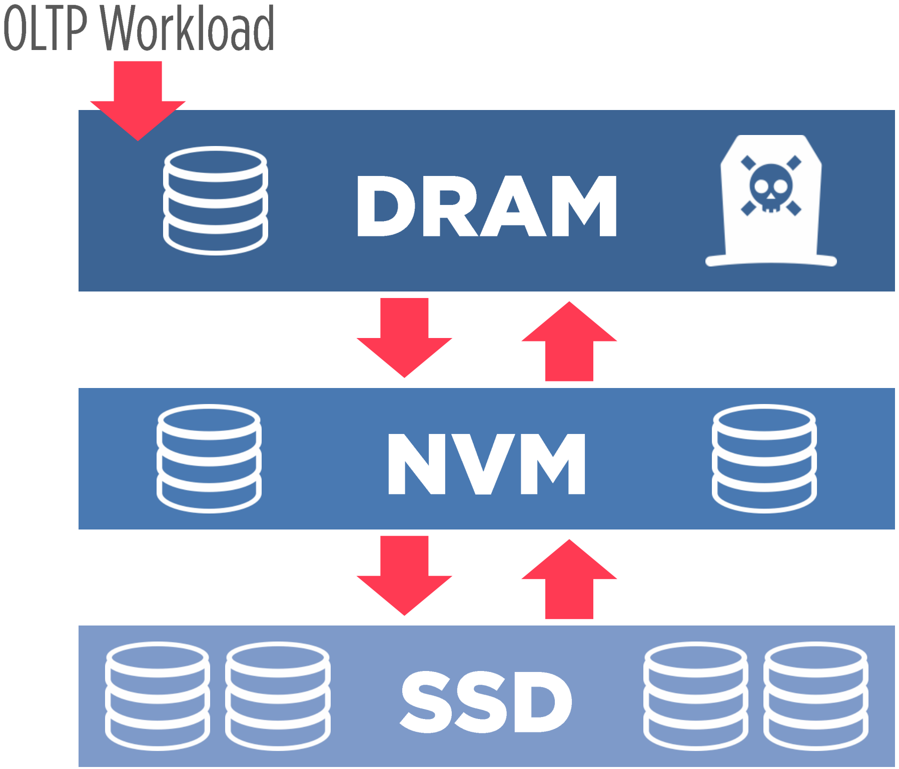
Merging Overhead

Voter :: 4x Latency



Elapsed Time

Multi-Level Anti-Caching



Current Investigation:

Eviction Policies

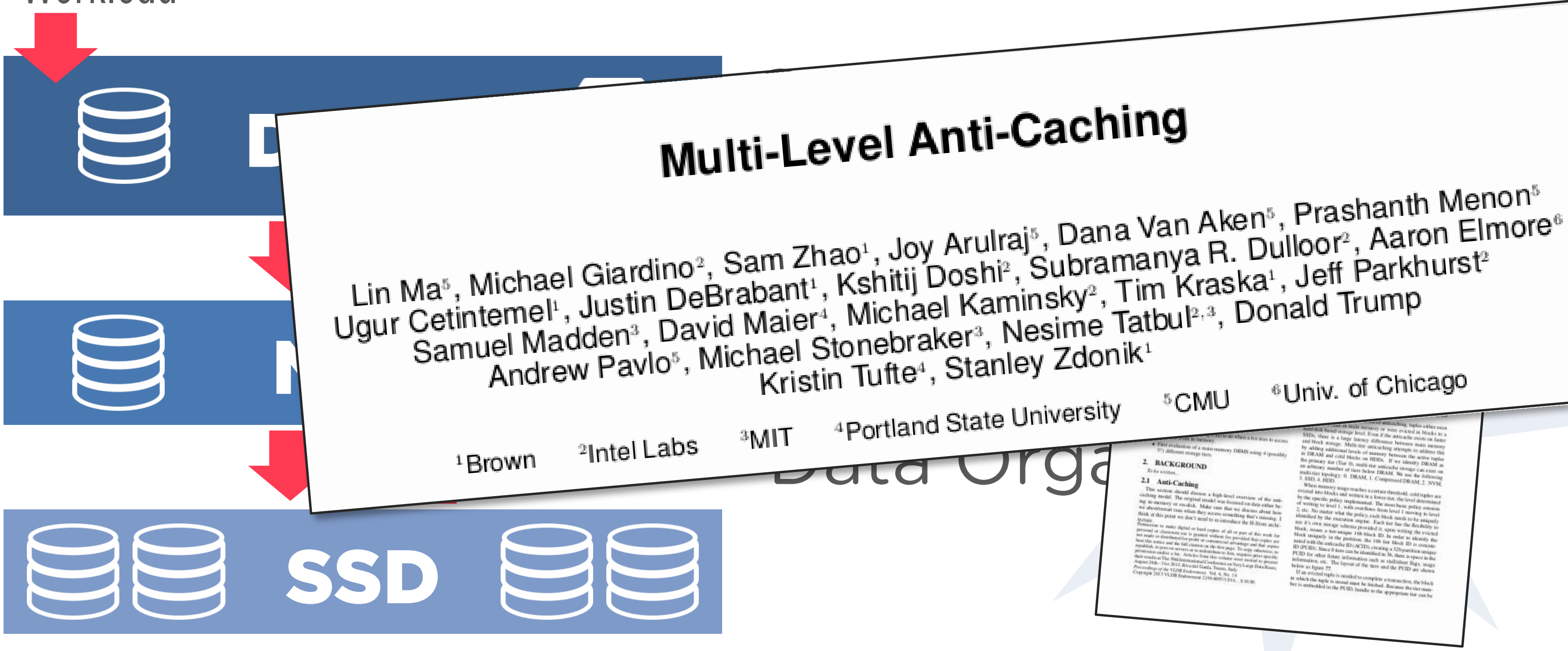
Retrieval Policies

Access Interfaces

Data Organization

Multi-Level Anti-Caching

OLTP Workload



Multi-Level Anti-Caching

Lin Ma[5], Michael Giardino[2], Sam Zhao[1], Joy Arulraj[5], Dana Van Aken[5], Prashanth Menon[5]
Ugur Cetintemel[1], Justin DeBrabant[1], Kshitij Doshi[2], Subramanya R. Dulloor[2], Aaron Elmore[6]
Samuel Madden[3], David Maier[4], Michael Kaminsky[2], Tim Kraska[1], Jeff Parkhurst[2]
Andrew Pavlo[5], Michael Stonebraker[3], Nesime Tatbul[2,3], Donald Trump
Kristin Tufte[4], Stanley Zdonik[1]

[1]Brown [2]Intel Labs [3]MIT [4]Portland State University [5]CMU [6]Univ. of Chicago

d to do when a txn tries to access not in memory.

- First evaluation of a main-memory DBMS using 4 (possibly 5?) different storage tiers.

2. BACKGROUND

To be written...

2.1 Anti-Caching

This section should discuss a high-level overview of the anti-caching model. The original model was focused on data either being in-memory or on-disk. Make sure that we discuss about how we abort/restart txns when they access something that's missing. I think at this point we don't need to re-introduce the H-Store architecture.

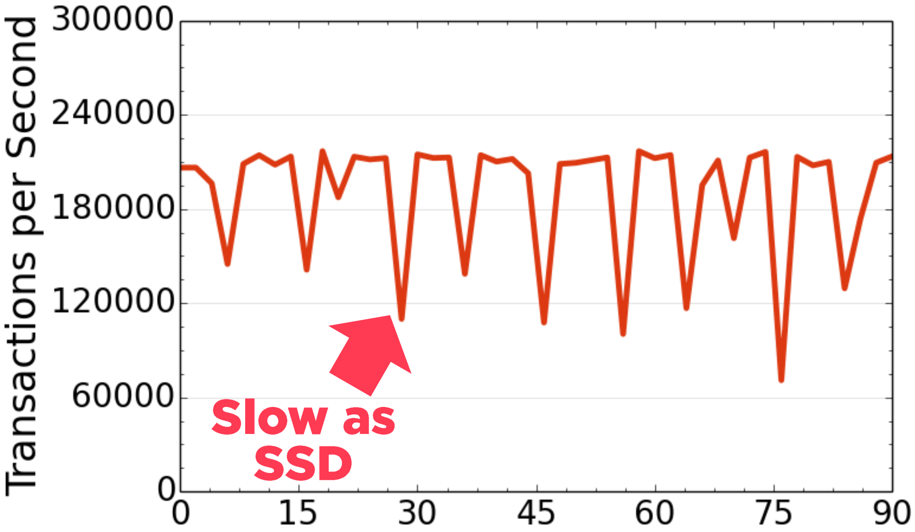
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 31st 2013, Riva del Garda, Trento, Italy.
Proceedings of the VLDB Endowment, Vol. 6, No. 14
Copyright 2013 VLDB Endowment 2150-8097/13/14... $ 10.00.

of anticaching, tuples either exist in main memory or were evicted in blocks to a hard disk based storage level. Even if the anticache exists on faster SSDs, there is a large latency difference between main memory and block storage. Multi-tier anticaching attempts to address this by adding additional levels of memory between the active tuples in DRAM and cold blocks on HDDs. If we identify DRAM as the primary tier (Tier 0), multi-tier anticache storage can exist on an arbitrary number of tiers below DRAM. We use the following multi-tier topology: 0. DRAM, 1. Compressed DRAM, 2. NVM, 3. SSD, 4. HDD.

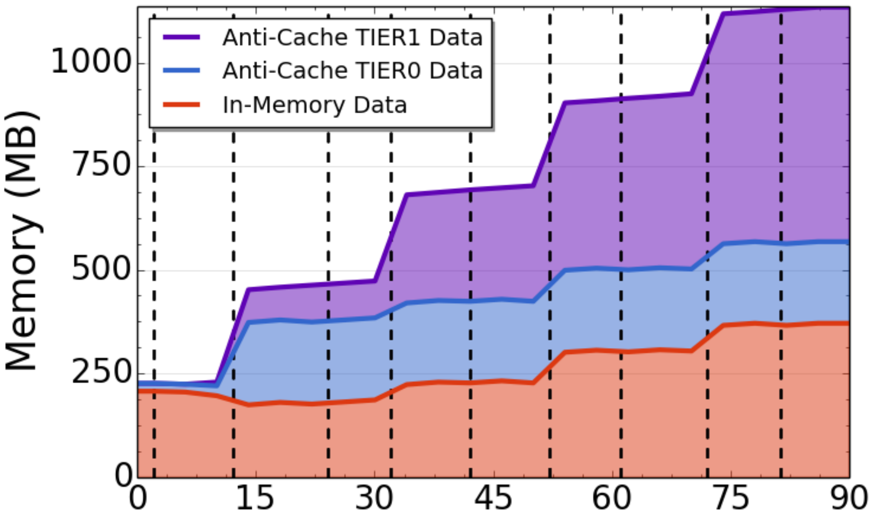
When memory usage reaches a certain threshold, cold tuples are evicted into blocks and written to a lower-tier, the level determined by the specific policy implemented. The most basic policy consists of writing to level 1, with overflows from level 1 moving to level 2, etc. No matter what the policy, each block needs to be uniquely identified by the execution engine. Each tier has the flexibility to use it's own storage schema provided it, upon writing the evicted block, issues a tier-unique 16b block ID. In order to identify the block uniquely in the partition, the 16b tier block ID is concatenated with the anticache ID (ACID), creating a 32b partition unique ID (PUID). Since 8 tiers can be identified in 3b, there is space in the PUID for other future information such as stall/abort flags, usage information, etc. The layout of the tiers and the PUID are shown below as figure ??.

If an evicted tuple is needed to complete a transaction, the block in which the tuple is stored must be fetched. Because the tier number is embedded in the PUID, handle to the appropriate tier can be

Voter :: Multi-Level :: 2x Latency



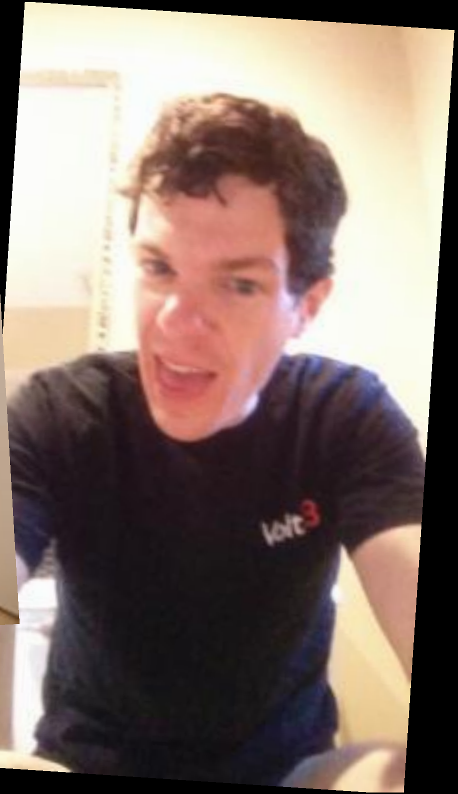
Slow as SSD



DRAM+NVM DBMS

Building a new storage manager for our new DBMS that will seamlessly incorporate NVM as an extension to its address space.

Upper-levels of the system are oblivious to “true” location of data.


volt3

END

@ANDY_PAVLO