Moving the Abyss:

Database Management on Future 1000-core Processors

NFS XPS Workshop
June 1-2, 2015

Andy Pavlo
CMU

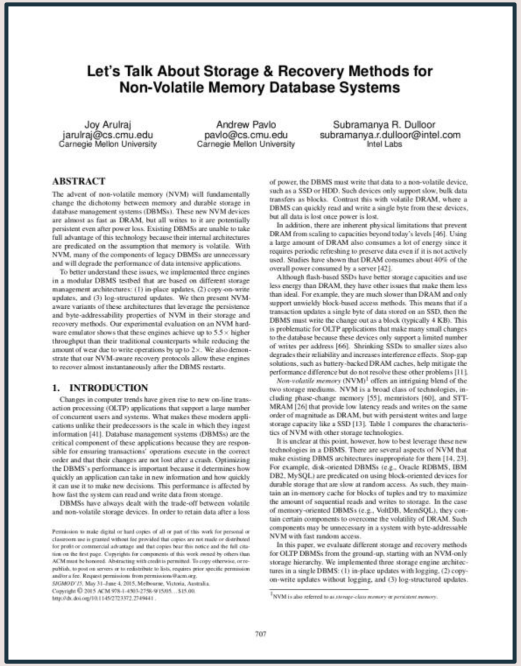
Srini Devadas
MIT

Future Database Architectures

Develop new DBMS components for future many-core CPU architectures.

- Concurrency Control
- Storage Methods
- Logging / Recovery
- Indexing

Non-Volatile Memory



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.

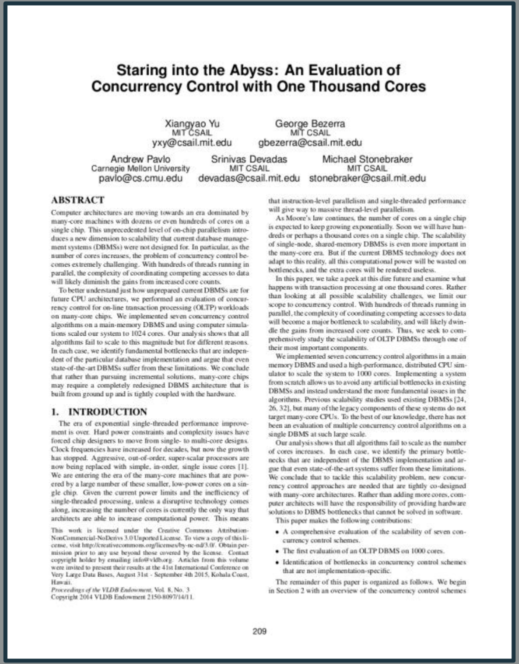
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

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 .

---

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

Many-Core Processors



Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores

Xiangyao Yu
MIT CSAIL
yxy@csail.mit.edu

George Bezerra
MIT CSAIL
gbezerra@csail.mit.edu

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

Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

ABSTRACT

Computer architectures are moving towards an era dominated by many-core machines with dozens or even hundreds of cores on a single chip. This unprecedented level of on-chip parallelism introduces a new dimension to scalability that current database management systems (DBMSs) were not designed for. In particular, as the number of cores increases, the problem of concurrency control becomes extremely challenging. With hundreds of threads running in parallel, the complexity of coordinating competing accesses to data will likely diminish the gains from increased core counts.

To better understand just how unprepared current DBMSs are for future CPU architectures, we performed an evaluation of concurrency control for on-line transaction processing (OLTP) workloads on many-core chips. We implemented seven concurrency control algorithms on a main-memory DBMS and using computer simulations scaled our system to 1024 cores. Our analysis shows that all algorithms fail to scale to this magnitude but for different reasons. In each case, we identify fundamental bottlenecks that are independent of the particular database implementation and argue that even state-of-the-art DBMSs suffer from these limitations. We conclude that rather than pursuing incremental solutions, many-core chips may require a completely redesigned DBMS architecture that is built from ground up and is tightly coupled with the hardware.

1. INTRODUCTION

The era of exponential single-threaded performance improvement is over. Hard power constraints and complexity issues have forced chip designers to move from single- to multi-core designs. Clock frequencies have increased for decades, but now the growth has stopped. Aggressive, out-of-order, super-scalar processors are now being replaced with simple, in-order, single issue cores [1]. We are entering the era of the many-core machines that are powered by a large number of these smaller, low-power cores on a single chip. Given the current power limits and the inefficiency of single-threaded processing, unless a disruptive technology comes along, increasing the number of cores is currently the only way that architects are able to increase computational power. This means that instruction-level parallelism and single-threaded performance will give way to massive thread-level parallelism.

As Moore's law continues, the number of cores on a single chip is expected to keep growing exponentially. Soon we will have hundreds or perhaps a thousand cores on a single chip. The scalability of single-node, shared-memory DBMSs is even more important in the many-core era. But if the current DBMS technology does not adapt to this reality, all this computational power will be wasted on bottlenecks, and the extra cores will be rendered useless.

In this paper, we take a peek at this dire future and examine what happens with transaction processing at one thousand cores. Rather than looking at all possible scalability challenges, we limit our scope to concurrency control. With hundreds of threads running in parallel, the complexity of coordinating competing accesses to data will become a major bottleneck to scalability, and will likely dwindle the gains from increased core counts. Thus, we seek to comprehensively study the scalability of OLTP DBMSs through one of their most important components.

We implemented seven concurrency control algorithms in a main memory DBMS and used a high-performance, distributed CPU simulator to scale the system to 1000 cores. Implementing a system from scratch allows us to avoid any artificial bottlenecks in existing DBMSs and instead understand the more fundamental issues in the algorithms. Previous scalability studies used existing DBMSs [24, 26, 32], but many of the legacy components of these systems do not target many-core CPUs. To the best of our knowledge, there has not been an evaluation of multiple concurrency control algorithms on a single DBMS at such large scale.

Our analysis shows that all algorithms fail to scale as the number of cores increases. In each case, we identify the primary bottlenecks that are independent of the DBMS implementation and argue that even state-of-the-art systems suffer from these limitations. We conclude that to tackle this scalability problem, new concurrency control approaches are needed that are tightly co-designed with many-core architectures. Rather than adding more cores, computer architects will have the responsibility of providing hardware solutions to DBMS bottlenecks that cannot be solved in software.

This paper makes the following contributions:

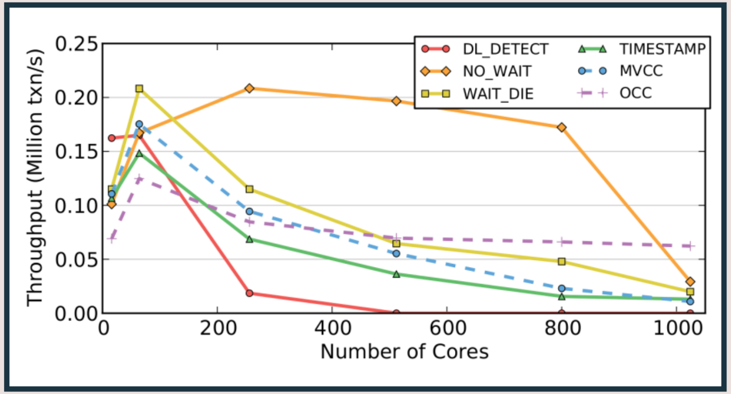
- A comprehensive evaluation of the scalability of seven concurrency control schemes.
- The first evaluation of an OLTP DBMS on 1000 cores.
- Identification of bottlenecks in concurrency control schemes that are not implementation-specific.

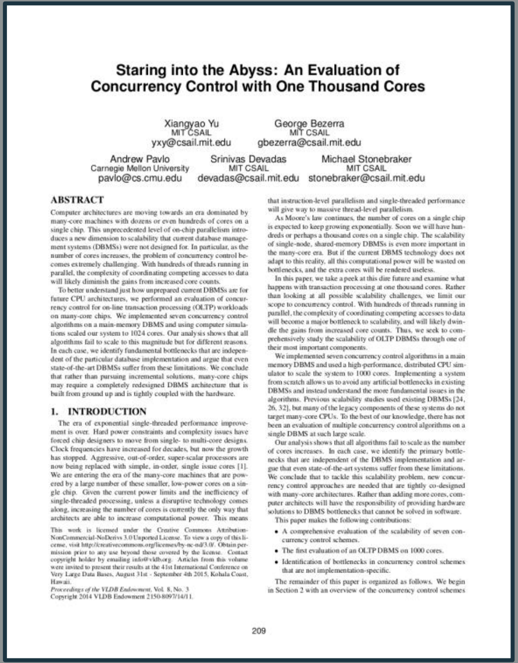
The remainder of this paper is organized as follows. We begin in Section 2 with an overview of the concurrency control schemes

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 3
Copyright 2014 VLDB Endowment 2150-8097/14/11.

Many-Core Processors

- Evaluate concurrency control schemes for transaction processing on 1000 cores.
- Custom test environment:
    - *DBx1000*
    - *MIT Graphite Simulator*
- No scheme scales due to lock thrashing, memory copying, and timestamp allocation bottlenecks.





Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores

Xiangyao Yu
MIT CSAIL
yxy@csail.mit.edu

George Bezerra
MIT CSAIL
gbezerra@csail.mit.edu

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

Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

ABSTRACT

Computer architectures are moving towards an era dominated by many-core machines with dozens or even hundreds of cores on a single chip. This unprecedented level of on-chip parallelism introduces a new dimension to scalability that current database management systems (DBMSs) were not designed for. In particular, as the number of cores increases, the problem of concurrency control becomes extremely challenging. With hundreds of threads running in parallel, the complexity of coordinating competing accesses to data will likely diminish the gains from increased core counts.

To better understand just how unprepared current DBMSs are for future CPU architectures, we performed an evaluation of concurrency control for on-line transaction processing (OLTP) workloads on many-core chips. We implemented seven concurrency control algorithms on a main-memory DBMS and using computer simulations scaled our system to 1024 cores. Our analysis shows that all algorithms fail to scale to this magnitude but for different reasons. In each case, we identify fundamental bottlenecks that are independent of the particular database implementation and argue that even state-of-the-art DBMSs suffer from these limitations. We conclude that rather than pursuing incremental solutions, many-core chips may require a completely redesigned DBMS architecture that is built from ground up and is tightly coupled with the hardware.

1. INTRODUCTION

The era of exponential single-threaded performance improvement is over. Hard power constraints and complexity issues have forced chip designers to move from single- to multi-core designs. Clock frequencies have increased for decades, but now the growth has stopped. Aggressive, out-of-order, super-scalar processors are now being replaced with simple, in-order, single issue cores [1]. We are entering the era of the many-core machines that are powered by a large number of these smaller, low-power cores on a single chip. Given the current power limits and the inefficiency of single-threaded processing, unless a disruptive technology comes along, increasing the number of cores is currently the only way that architects are able to increase computational power. This means that instruction-level parallelism and single-threaded performance will give way to massive thread-level parallelism.

As Moore's law continues, the number of cores on a single chip is expected to keep growing exponentially. Soon we will have hundreds or perhaps a thousand cores on a single chip. The scalability of single-node, shared-memory DBMSs is even more important in the many-core era. But if the current DBMS technology does not adapt to this reality, all this computational power will be wasted on bottlenecks, and the extra cores will be rendered useless.

In this paper, we take a peek at this dire future and examine what happens with transaction processing at one thousand cores. Rather than looking at all possible scalability challenges, we limit our scope to concurrency control. With hundreds of threads running in parallel, the complexity of coordinating competing accesses to data will become a major bottleneck to scalability, and will likely dwindle the gains from increased core counts. Thus, we seek to comprehensively study the scalability of OLTP DBMSs through one of their most important components.

We implemented seven concurrency control algorithms in a main memory DBMS and used a high-performance, distributed CPU simulator to scale the system to 1000 cores. Implementing a system from scratch allows us to avoid any artificial bottlenecks in existing DBMSs and instead understand the more fundamental issues in the algorithms. Previous scalability studies used existing DBMSs [24, 26, 32], but many of the legacy components of these systems do not target many-core CPUs. To the best of our knowledge, there has not been an evaluation of multiple concurrency control algorithms on a single DBMS at such large scale.

Our analysis shows that all algorithms fail to scale as the number of cores increases. In each case, we identify the primary bottlenecks that are independent of the DBMS implementation and argue that even state-of-the-art systems suffer from these limitations. We conclude that to tackle this scalability problem, new concurrency control approaches are needed that are tightly co-designed with many-core architectures. Rather than adding more cores, computer architects will have the responsibility of providing hardware solutions to DBMS bottlenecks that cannot be solved in software.

This paper makes the following contributions:

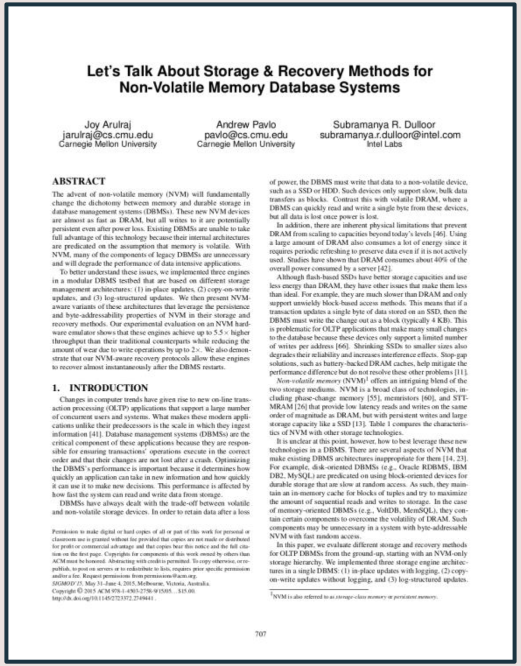
- A comprehensive evaluation of the scalability of seven concurrency control schemes.
- The first evaluation of an OLTP DBMS on 1000 cores.
- Identification of bottlenecks in concurrency control schemes that are not implementation-specific.

The remainder of this paper is organized as follows. We begin in Section 2 with an overview of the concurrency control schemes

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 3
Copyright 2014 VLDB Endowment 2150-8097/14/11.

Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores, *Proceedings of VLDB, vol. 8, iss. 3, pp. 209-220, November 2014*

Non-Volatile Memory



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.

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. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

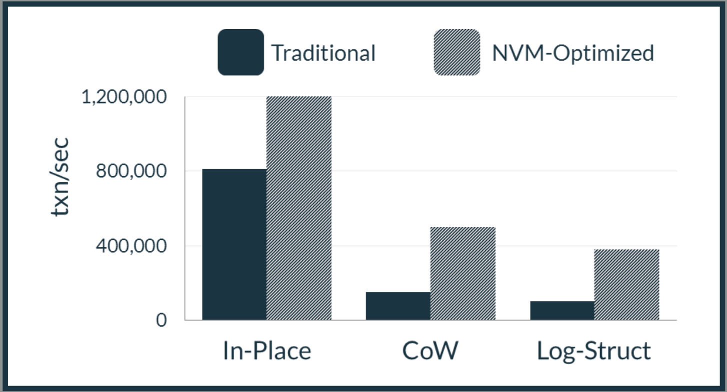
SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2749441 .

---

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

Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems, *Proceedings of SIGMOD, pp. 707-702, June 2015*

- Evaluate multiple methods with NVM-only storage hierarchy using a single test-bed platform.
- Three different architectures:
    - *In-place, Copy-on-Write, Log-structured Updates.*
- NVM-optimized components that use persistent pointers to reduce write-amplification.



Next Steps

Software/Hardware Co-Designs

Push key DBMS components into hardware extensions.

http://cmudb.io/1000cores

Special thanks to Daniel Muessig, Esq
412-454-5582