DATA MANAGEMENT ON NON-VOLATILE MEMORY

JOY ARULRAJ
Carnegie Mellon University
EVOLUTION OF MEMORY TECHNOLOGY

1960 DRAM
1970
1980 FLASH MEMORY
1990
2000
2010 NON-VOLATILE MEMORY
2020

intel HP
## NON-VOLATILE MEMORY [NVM]

<table>
<thead>
<tr>
<th>PERFORMANCE</th>
<th>VOLATILE</th>
<th>NON-VOLATILE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DRAM</td>
<td>NVM</td>
</tr>
<tr>
<td>FAST</td>
<td>SSD</td>
<td></td>
</tr>
<tr>
<td>SLOW</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DURABILITY**
<table>
<thead>
<tr>
<th>CHARACTERISTIC</th>
<th>DRAM</th>
<th>NVM</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Latency</td>
<td>1x</td>
<td>10x</td>
<td>1000x</td>
</tr>
<tr>
<td>Byte-Addressability</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Durability</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>High Capacity</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cost/GB</td>
<td>100x</td>
<td>10x</td>
<td>1x</td>
</tr>
</tbody>
</table>
50 YEARS OF DATABASE SYSTEMS RESEARCH

1. SEPARATE MEMORY & STORAGE
   - DRAM
   - SSD

2. COMPUTE/STORAGE BALANCE
   - CPU
   - SSD

3. RANDOM VS. SEQUENTIAL
   - SSD

2 1 3
RESEARCH OVERVIEW

• How to manage data on NVM?
  – *Challenging because of NVM’s unique characteristics*
  – *Important given the sudden shift in compute/storage balance*
#1: INDUSTRY STANDARDS

• Standardization of NVM technologies
  – Design standards
  – Interface specifications
#2: OPERATING SYSTEM SUPPORT

- Major operating systems natively support NVM
  - Linux 4.8
  - Windows 10
#3: ARCHITECTURAL SUPPORT

- New assembly instructions in ISA updates
  - Efficiently flush data from volatile CPU cache to NVM
  - Kaby Lake processor
HOW DO TODAY’S DATABASE SYSTEMS PERFORM ON NON-VOLATILE MEMORY?
NVM HARDWARE EMULATOR [INTEL]

• Emulates a wide range of NVM technologies
  – Special CPU microcode
  – Supports recently added assembly instructions
TODAY’S DATABASE SYSTEMS ON NVM

• Database System: MySQL

• Storage device performance
  – NVM’s performance compared to that of disk
  – I/O benchmark

• Database system performance
  – On NVM compared to that on disk
  – TPC-C benchmark
STORAGE HIERARCHIES

1. DISK-BASED HIERARCHY
   - DRAM
   - SSD

2. NVM-BASED HIERARCHY
   - NVM

BOTH MEMORY & STORAGE
**STORAGE DEVICE PERFORMANCE**

I/O Throughput (Operations/sec/thread)

- **SOLID-STATE DISK:** 2541
- **NON-VOLATILE MEMORY:** 278971

Higher is Better

>100x
DATABASE SYSTEM PERFORMANCE

**Transactional Throughput** (Transactions/sec)

- Higher is Better

- **Disk-centric design hurts performance on NVM**

<table>
<thead>
<tr>
<th>Solid-State Disk</th>
<th>Non-Volatile Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1750</td>
<td>5510</td>
</tr>
</tbody>
</table>

**3x**

**9,000**

**6,000**

**3,000**

**0**
PELOTON NVM DATABASE SYSTEM

QUERY PROCESSING (SIGMOD’16)

STORAGE MANAGEMENT (SIGMOD’15)

INDEXING (VLDB’18)

LOGGING AND RECOVERY (VLDB’17)

LAYERS OF A DATABASE SYSTEM
WRITE-BEHIND LOGGING

BZTREE INDEX

FUTURE DIRECTIONS
LOGGING & RECOVERY: MOTIVATION

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION

DATABASE SYSTEM

SHOPPING APPLICATION

DURABILITY
FAULT-TOLERANCE

ATOMICITY
ALL-OR-NOTHING
WRITE-AHEAD LOGGING: DURABILITY

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION
WRITE-AHEAD LOGGING: ATOMICITY

LOG

| TRANSACTION #1 – BEGIN |
| TRANSACTION #1 – ADD ORDER |
| TRANSACTION #2 – BEGIN |
| TRANSACTION #1 – UPDATE STOCK |
| TRANSACTION #3 – BEGIN |
| TRANSACTION #3 – ADD ORDER |

SYSTEM CRASH

BEGIN TRANSACTION
BEGIN TRANSACTION
BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION
COMMIT TRANSACTION
COMMIT TRANSACTION
WRITE-AHEAD LOGGING: RECOVERY PROTOCOL

**ACTIVE TRANSACTION TABLE**

<table>
<thead>
<tr>
<th>TXN ID</th>
<th>STATUS</th>
<th>LATEST CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXN #2</td>
<td>RUNNING</td>
<td>LOG RECORD #7</td>
</tr>
<tr>
<td>TXN #3</td>
<td>RUNNING</td>
<td>---</td>
</tr>
</tbody>
</table>

**DIRTY PAGE TABLE**

<table>
<thead>
<tr>
<th>PAGE ID</th>
<th>CHANGE THAT DIRTIED PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE #30</td>
<td>LOG RECORD #5</td>
</tr>
<tr>
<td>PAGE #40</td>
<td>LOG RECORD #7</td>
</tr>
</tbody>
</table>

**Diagram**

- **BUFFER POOL**
- **DATABASE**
- **LOG**
- **DRAM**
- **SSD**

**Process Steps**

1. **WRITE-AHEAD LOGGING**
2. **REDO CHANGES**
3. **UNDO CHANGES**
PROBLEM #1: DATA DUPLICATION

1. BUFFER POOL
2. DATABASE
3. NVM

PERFORMANCE
STORAGE COST
PROBLEM #2: SLOW RECOVERY

BUFFER POOL

DATABASE

LOG

AVAILABILITY

LINEAR-TIME RECOVERY

NEEDS TO REDO LOG
HOW TO IMPROVE PERFORMANCE AND AVAILABILITY ON NON-VOLATILE MEMORY?
WRITE-BEHIND LOGGING: OVERVIEW

• NVM-centric design
  – *Improves availability by enabling instant recovery*
  – *Provides same guarantees as write-ahead logging*

• Key techniques
  – *Directly propagate changes to the database*
  – *Only record meta-data in log*
DATA VERSIONING USING TIMESTAMPS

DATA VERSION

TRANSACTION TIMESTAMP

DATABASE

STOCK

STOCK'

STOCK''

DATA VERSIONING USING TIMESTAMPS

STOCK

STOCK'

STOCK''

DATABASE

ORACLE

IBM

DB2

Microsoft SQL Server

Hekaton

MySQL

PostgreSQL
WRITE-BEHIND LOGGING: DURABILITY

L1 CACHE
L2 CACHE
NVM

BUFFER POOL

DATABASE

LOG

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION
WRITE-BEHIND LOGGING: ATOMICITY

DATABASE SYSTEM

TIMESTAMP INTERVAL FOR EACH TRANSACTION BATCH

<10, 20>

DATABASE

LOG

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION

BEGIN TRANSACTION
Add Order
Update Stock
COMMIT TRANSACTION

STOCK'
ORDER
ORDER
STOCK''
STOCK

NVM
SOLUTION #1: NO DATA DUPLICATION

NVM

DATA

META-DATA

DATABASE

LOG

ORDER

<10, 20>

STOCK

PERFORMANCE

STORAGE COST

30
SOLUTION #2: INSTANT RECOVERY

WRITE-AHEAD LOGGING  Linear-Time Recovery

1 ANALYSIS  2 REDO  3 UNDO

WRITE-BEHIND LOGGING  Constant-Time Recovery

1 ANALYSIS

AVAILABILITY
WRITE-BEHIND LOGGING

• Enables instant recovery from failures
• Eliminates data duplication
• Generalizes to single-versioned database systems
• Supports a multi-tier storage hierarchy
• Handles long lived transactions
• Copes with failures during recovery
EVALUATION

• Logging Protocols: Write-Behind vs. Write-Ahead Logging
  – Recovery Time
  – Database System Performance

• Workload: TPC-C benchmark on Peloton

• Storage devices
  – Solid-state disk
  – Non-volatile memory
RECOVERY TIME

- **WRITE-AHEAD LOGGING**
- **WRITE-BEHIND LOGGING**

**Recovery Time (sec)**

- **SOLID-STATE DISK**
  - Recovery Time: 260 sec
  - Lower is better

- **NON-VOLATILE MEMORY**
  - Recovery Time: 48 sec
  - Lower is better

Comparison:

- **250x** improvement for SOLID-STATE DISK
- **160x** improvement for NON-VOLATILE MEMORY
DATABASE SYSTEM PERFORMANCE

Write-behind logging only works well on NVM

Transaction Throughput (Transactions/sec)

- Higher is Better

<table>
<thead>
<tr>
<th>SOLID-STATE DISK</th>
<th>NON-VOLATILE MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>8K</td>
<td>1.7K</td>
</tr>
<tr>
<td>2x 88K</td>
<td>41K</td>
</tr>
</tbody>
</table>

TRANSACTIONAL

Throughput

Higher is Better

Write-Ahead Logging

Write-Behind Logging
Advances the state of the art by shifting the complexity class of the recovery protocol on NVM
PELOTON NVM DATABASE SYSTEM

QUERY PROCESSING
(SIGMOD’16)

STORAGE MANAGEMENT
(SIGMOD’15)

LAYERS OF A DATABASE SYSTEM

INDEXING
(VLDB’18)

LOGGING AND RECOVERY
(VLDB’17)
INDEXING DATA: MOTIVATION

BEGIN TRANSACTION
Update Stock By Stock ID
COMMIT TRANSACTION

DATABASE SYSTEM

SHOPPING APPLICATION

STOCK ID > 5 AND < 15

STOCK INDEX
SYNCHRONIZATION WITH LOCKS

BEGIN TRANSACTION
Update Stock by Stock ID
COMMIT TRANSACTION

BEGIN TRANSACTION
Update Stock by Stock ID
COMMIT TRANSACTION

BEGIN TRANSACTION
Update Stock by Stock ID
COMMIT TRANSACTION
BWTREE: LOCK-FREE B+TREE [MICROSOFT]

SINGLE-WORD COMPARE-AND-SWAP INSTRUCTION

CPU
BWTREE: DURABILITY & ATOMICITY

BEGIN TRANSACTION
Update Stock by Stock ID
COMMIT TRANSACTION

INDEX-SPECIFIC LOGGING & RECOVERY

BUFFER POOL

INDEX

LOG

DRAM

SSD
PROBLEM #1: HIGH CODE COMPLEXITY

SINGLE-WORD COMPARE-AND-SWAP INSTRUCTION

LOCK-FREEDOM INTERMEDIATE STATES
PROBLEM #2: INDEX-SPECIFIC PROTOCOL
HOW TO SIMPLIFY PROGRAMMING ON NON-VOLATILE MEMORY?

BZTREE: A HIGH-PERFORMANCE LATCH-FREE INDEX FOR NON-VOLATILE MEMORY
VLDB 2018
BZTREE: OVERVIEW

• NVM-centric design
  – *Uses a new software primitive to simplify programming*
  – *Provides same guarantees as disk-centric BwTree*

• BzTree supersedes BwTree
  – *But, we skipped BxTree & ByTree*
  – *Because we think it’s the “last” index you will ever need!*

• Key techniques
  – *Offload programming complexity to the software primitive*
  – *Adopt a simpler NVM-centric architecture*
NVM-CENTRIC SOFTWARE PRIMITIVE

HARDWARE PRIMITIVE

VOLATILE SINGLE-WORD COMPARE-AND-SWAP

SOFTWARE PRIMITIVE

PERSISTENT MULTI-WORD COMPARE-AND-SWAP
BZTREE: NVM-CENTRIC ARCHITECTURE

BUFFER POOL

INDEX

SOFTWARE PRIMITIVE

BEGIN TRANSACTION
Update Stock by Stock ID
COMMIT TRANSACTION

BEGIN TRANSACTION
Update Stock by Stock ID
COMMIT TRANSACTION

BEGIN TRANSACTION
Update Stock by Stock ID
COMMIT TRANSACTION

L1 CACHE
L2 CACHE

NVM
### BZTREE: DURABILITY AND ATOMICITY

#### Operation Table

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>EXPECTED OLD VALUE</th>
<th>NEW VALUE</th>
<th>FLUSHED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>OLD CHILD POINTER</td>
<td>NEW CHILD POINTER</td>
<td>1</td>
</tr>
<tr>
<td>0x200</td>
<td>OLD NODE STATUS</td>
<td>NEW NODE STATUS</td>
<td>1</td>
</tr>
<tr>
<td>0x300</td>
<td>OLD PARENT POINTER</td>
<td>NEW PARENT POINTER</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Persistent Multi-Word Compare-And-Swap**

**Software Primitive**

---

**Diagram: Node Connections**
SOLUTION #1: LOW CODE COMPLEXITY

SOFTWARE PRIMITIVE

EXPONENTIALLY FEWER INTERMEDIATE STATES

SPLITTING A NODE
SOLUTION #2: NO INDEX-SPECIFIC PROTOCOL

SOFTWARE PRIMITIVE

DURABILITY & ATOMICITY

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>FLUSHED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>OLD CHILD POINTER</td>
<td>NEW CHILD POINTER</td>
<td>1</td>
</tr>
<tr>
<td>0x200</td>
<td>OLD NODE STATUS</td>
<td>NEW NODE STATUS</td>
<td>1</td>
</tr>
<tr>
<td>0x300</td>
<td>OLD PARENT POINTER</td>
<td>NEW PARENT POINTER</td>
<td>0</td>
</tr>
</tbody>
</table>
EVALUATION

• Index data structures: BzTree vs. BwTree index
  – Code complexity
  – Index Performance

• Benchmark: Yahoo Cloud Serving benchmark
  – Read-mostly workload
  – Balanced workload

• Storage devices
  – Non-volatile memory (BzTree only works on NVM)
### CODE COMPLEXITY [NODE SPLIT PROTOCOL]

<table>
<thead>
<tr>
<th>CODE COMPLEXITY METRIC</th>
<th>BWTREE</th>
<th>BZTREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYCLOMATIC COMPLEXITY</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>LINES OF CODE</td>
<td>750</td>
<td>200</td>
</tr>
</tbody>
</table>

- Lower is Better
- 2x Fewer Intermediate States
- 4x No Index-Specific Protocol
INDEX PERFORMANCE

In addition to simplifying programming, BzTree also delivers better performance.
BZTREE: SUMMARY

Advances the state of the art by illustrating a simpler way to design data structures for NVM
OTHER RESEARCH PROJECTS
CURRENT RESEARCH AGENDA

AREA #1:
NON-VOLATILE MEMORY
DATABASE SYSTEMS

AREA #2:
SELF-DRIVING
DATABASE SYSTEMS
AREA #1: NVM DATABASE SYSTEM

QUERY PROCESSING (SIGMOD’16)

STORAGE MANAGEMENT (SIGMOD’15)

INDEXING (VLDB’18)

LOGGING AND RECOVERY (VLDB’17)

LAYERS OF A DATABASE SYSTEM
AREA #2: SELF-DRIVING DATABASE SYSTEM

- Storage Layout Tuning (SIGMOD’16)
- Index Tuning (Under Review)
- Self-Driving Design (CIDR’17)
CROSS-MEDIA STORAGE MANAGEMENT

• Storage management tailored for a multi-tier hierarchy
  – *Industry collaboration: Intel Labs, Samsung Research*
• Most hardware-centric optimizations are system-specific

This approach does not scale!
DECLARATIVE HARDWARE MANAGEMENT

DATABASE SYSTEM

DECLARATIVE REQUESTS

HARDWARE-CENTRIC MECHANISMS

DECLARATIVE HARDWARE MANAGER

PELOTON

TensorFlow

MACHINE LEARNING SYSTEM

NVM

TPU
CONCLUSION

• Non-volatile memory invalidates age-old design assumptions
• Presented the design of a new NVM-centric database system
• Broader impact on other types of data processing systems

WRITE-BEHIND LOGGING

BZTREE INDEX

64
DECLARATIVE STORAGE MANAGEMENT

DATABASE SYSTEM

PELOTON

DECLARATIVE STORAGE ALGEBRA

DECLARATIVE HARDWARE MANAGER

PELOTON

MACHINE LEARNING SYSTEM

TensorFlow

• DATA LAYOUT
• DATA MIGRATION
• DATA ORDERING
• DATA DISTRIBUTION
• DATA PACKING

STORAGE-CENTRIC MECHANISMS

DRAM

SSD

NVM
WRITE-BEHIND LOGGING: RELATED WORK

• NVM-centric logging, but only support linear-time recovery
  – NVM Group Commit [VLDB’13], Passive Group Commit [VLDB’14]

• NVM-centric logging with non-commodity hardware features
  – MARS [SOSP’13], BPFS [SOSP’09]

Write-behind logging enables constant-time recovery using only commodity hardware features
BZTREE: RELATED WORK

• NVM-centric indexing, but with index-specific recovery logic
  – FP-Tree [SIGMOD’16], NV-Tree [FAST’15]

• DRAM-centric indexing
  – ART [ICDE’13], MassTree [Eurosys’12]

BzTree illustrates a simpler way to design persistent data structures by obviating the need for index-specific recovery
RESEARCH IMPACT

• Research groups are shaping their systems for NVM
  – Oracle, SAP HANA

• Byte-addressable NVM is still not commercially available
  – Intel shipped block-addressable NVM in 2017
  – Intel plans to ship byte-addressable NVM in 2019
  – Peloton will be the only open-source database system ready for NVM
# SHADOW PAGING

<table>
<thead>
<tr>
<th>RECOVERY PROTOCOL CHARACTERISTIC</th>
<th>SHADOW PAGING</th>
<th>WRITE-BEHIND LOGGING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROTOCOL TYPE</td>
<td>NO REDO/ NO UNDO</td>
<td>NO REDO/ UNDO</td>
</tr>
<tr>
<td>COMMIT OVERHEAD</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>SYSTEM INTEGRATION</td>
<td>COMPLEX</td>
<td>SIMPLE</td>
</tr>
<tr>
<td>CONCURRENCY SUPPORT</td>
<td>NEED LOGGING</td>
<td>YES</td>
</tr>
</tbody>
</table>
# VISTA RECOVERABLE MEMORY

<table>
<thead>
<tr>
<th>SYSTEM CHARACTERISTIC</th>
<th>RECOVERABLE MEMORY</th>
<th>WRITE-BEHIND LOGGING</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDO MECHANISM</td>
<td>PHYSICAL UNDO</td>
<td>LOGICAL UNDO</td>
</tr>
<tr>
<td>CONCURRENCY SUPPORT</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>DBMS INTEGRATION</td>
<td>COMPLEX</td>
<td>SIMPLE</td>
</tr>
</tbody>
</table>
THE LOG IS THE DATABASE

<table>
<thead>
<tr>
<th>SYSTEM CHARACTERISTIC</th>
<th>AMAZON AURORA</th>
<th>WRITE-BEHIND LOGGING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROTOCOL TYPE</td>
<td>REDO/ UNDO</td>
<td>NO REDO/ UNDO</td>
</tr>
<tr>
<td>MATERIALIZATION</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>READ OVERHEAD</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
</tbody>
</table>
BATTERY-BACKED DRAM

• Available only in specialized environments
  – General-purpose database systems are not designed to leverage it

• Limitations of battery-backed DRAM
  – Physical form factor, Availability, Reliability, Cost
MEDIA FAILURE

• Write-behind logging focuses on software failures
  – Transaction failures, System failures
  – Software failures outnumber media failures 10-to-1

• Media failure
  – Replicating data to another machine’s non-volatile memory
SECURITY/PROTECTION

• Virtual memory protection mechanism
  – All accesses should go through the TLB
  – Using write-permission bits in the page table
## Transactional Memory

<table>
<thead>
<tr>
<th>System Characteristic</th>
<th>Software Transactional Memory</th>
<th>Hardware Transactional Memory</th>
<th>Persistent Multi-Word CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durability</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Performance</td>
<td>Heavyweight</td>
<td>Lightweight</td>
<td>Lightweight</td>
</tr>
<tr>
<td>False Aborts</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
## READ-COPY-UPDATE

<table>
<thead>
<tr>
<th>SYNCHRONIZATION PRIMITIVE CHARACTERISTIC</th>
<th>READ-COPY-UPDATE</th>
<th>PERSISTENT MULTI-WORD CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTI-LOCATION</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>DURABILITY</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>WRITERS USE LOCKS</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>
CANDIDATE NVM TECHNOLOGIES

- Faster than DRAM
  - SPIN-TRANSFER TORQUE MRAM
- FLASH-BACKED DRAM
- MEMRISTOR
- PHASE-CHANGE MEMORY
- Slower than DRAM
  - 2D & 3D FLASH