Databases on New Hardware

@Andy_Pavlo // 15-721 // Spring 2018
ADMINISTRIVIA

Snowflake Guest: May 2\textsuperscript{th} @ 3:00pm

Final Exam Handout: May 2\textsuperscript{nd}

Code Review #2: May 2\textsuperscript{nd} @ 11:59pm
\rightarrow We will use the same group pairings as before.

Final Presentations: May 14\textsuperscript{th} @ 8:30am
\rightarrow GHC 4303 (ignore schedule!)
\rightarrow 12 minutes per group
\rightarrow Food and prizes for everyone!
ADMINISTRIVIA

Course Evaluation
→ Please tell me what you really think of me.
→ I actually take your feedback in consideration.
→ Take revenge on next year's students.

https://cmu.smartevals.com/
<table>
<thead>
<tr>
<th>Instructor</th>
<th>Questions</th>
<th>Text Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAVLO</td>
<td>Question: Comments</td>
<td>Andy smells terrible. He has this weird odor like a he rolled around in trash can and then let feral cats piss all over him. I don't think he washes those shitty database shirts that he wears for the videos. In addition to filling out this course evaluation, I also filled a complaint with the Allegheny County Health Department. I don't think a grown man should smell as bad as Andy does. I guess the database part of the course was okay but man that odor was tough to deal with each class.</td>
</tr>
<tr>
<td>PAVLO</td>
<td>Question: Comments</td>
<td></td>
</tr>
<tr>
<td>PAVLO</td>
<td>Question: Comments</td>
<td></td>
</tr>
</tbody>
</table>
People have been thinking about using hardware to accelerate DBMSs for decades.

**1980s:** Database Machines

**2000s:** FPGAs + Appliances

**2010s:** FPGAs + GPUs
TODAY’S AGENDA

Non-Volatile Memory
GPU Acceleration
Hardware Transactional Memory
NON-VOLATILE MEMORY

Emerging storage technology that provide low latency read/writes like DRAM, but with persistent writes and large capacities like SSDs. → aka Storage-class Memory, Persistent Memory

First devices will be block-addressable (NVMe)
Later devices will be byte-addressable.
FUNDAMENTAL ELEMENTS OF CIRCUITS

Capacitor (ca. 1745)

Resistor (ca. 1827)

Inductor (ca. 1831)
FUNDAMENTAL ELEMENTS OF CIRCUITS

In 1971, Leon Chua at Berkeley predicted the existence of a fourth fundamental element.

A two-terminal device whose resistance depends on the voltage applied to it, but when that voltage is turned off it permanently remembers its last resistive state.
FUNDAMENTAL ELEMENTS OF CIRCUITS

- Capacitor (ca. 1745)
- Resistor (ca. 1827)
- Inductor (ca. 1831)
- Memristor (ca. 1971)
MERISTORS

A team at HP Labs led by Stanley Williams stumbled upon a nano-device that had weird properties that they could not understand.

It wasn’t until they found Chua’s 1971 paper that they realized what they had invented.
TECHNOLOGIES

Phase-Change Memory (PRAM)
Resistive RAM (ReRAM)
Magnetoresistive RAM (MRAM)
PHASE-CHANGE MEMORY

Storage cell is comprised of two metal electrodes separated by a resistive heater and the phase change material (chalcogenide).

The value of the cell is changed based on how the material is heated.
→ A short pulse changes the cell to a ‘0’.
→ A long, gradual pulse changes the cell to a ‘1’.
RESISTIVE RAM

Two metal layers with two TiO$_2$ layers in between. Running a current one direction moves electrons from the top TiO$_2$ layer to the bottom, thereby changing the resistance.

May be programmable storage fabric… → Bertrand Russell’s Material Implication Logic
MAGNETORESISTIVE RAM

Stores data using magnetic storage elements instead of electric charge or current flows.

Spin-Transfer Torque (STT-MRAM) is the leading technology for this type of NVM. → Supposedly able to scale to very small sizes (10nm) and have SRAM latencies.
WHY THIS IS FOR REAL THIS TIME

Industry has agreed to standard technologies and form factors.

Linux and Microsoft have added support for NVM in their kernels (DAX).

Intel has added new instructions for flushing cache lines to NVM (CLFLUSH, CLWB).
NVM DIMM FORM FACTORS

NVDIMM-F (2015)
→ Flash only. Has to be paired with DRAM DIMM.

NVDIMM-N (2015)
→ Flash and DRAM together on the same DIMM.
→ Appears as volatile memory to the OS.

NVDIMM-P (2018)
→ True persistent memory. No DRAM or flash.
NVM CONFIGURATIONS

DRAM as Hardware-Managed Cache

NVM Next to DRAM

NVM as Persistent Memory

Source: Ismail Oukid
Block-addressable NVM is not that interesting.

Byte-addressable NVM will be a game changer but will require some work to use correctly.
→ In-memory DBMSs will be better positioned to use byte-addressable NVM.
→ Disk-oriented DBMSs will initially treat NVM as just a faster SSD.
STORAGE & RECOVERY METHODS

Understand how a DBMS will behave on a system that only has byte-addressable NVM.

Develop NVM-optimized implementations of standard DBMS architectures.

Based on the N-Store prototype DBMS.
SYNCHRONIZATION

Existing programming models assume that any write to memory is non-volatile.
→ CPU decides when to move data from caches to DRAM.

The DBMS needs a way to ensure that data is flushed from caches to NVM.
Existing programming models assume that any write to memory is non-volatile. → CPU decides when to move data from caches to DRAM.

The DBMS needs a way to ensure that data is flushed from caches to NVM.
SYNCHRONIZATION

Existing programming models assume that any write to memory is non-volatile. → CPU decides when to move data from caches to DRAM.

The DBMS needs a way to ensure that data is flushed from caches to NVM.
NAMING

If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.
If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.
If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.
NVM-AWARE MEMORY ALLOCATOR

Feature #1: Synchronization
→ The allocator writes back CPU cache lines to NVM using the CLFLUSH instruction.
→ It then issues a SFENCE instruction to wait for the data to become durable on NVM.

Feature #2: Naming
→ The allocator ensures that virtual memory addresses assigned to a memory-mapped region never change even after the OS or DBMS restarts.
DBMS ENGINE ARCHITECTURES

Choice #1: In-place Updates
→ Table heap with a write-ahead log + snapshots.
→ Example: VoltDB

Choice #2: Copy-on-Write
→ Create a shadow copy of the table when updated.
→ No write-ahead log.
→ Example: LMDB

Choice #3: Log-structured
→ All writes are appended to log. No table heap.
→ Example: RocksDB
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

Tuple #00
Tuple #01
Tuple #02

Durable Storage

Write-Ahead Log
Snapshots
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

Tuple #00
Tuple #01
Tuple #02

Durable Storage

Write-Ahead Log
Tuple Delta
Snapshots
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

Durable Storage

1. Write-Ahead Log
   Tuple Delta

2. Snapshots

Tuple #00

Tuple #01 (!)

Tuple #02

Snapshots
In-Place Updates Engine

In-Memory Index

In-Memory Table Heap

Tuples

Tuple #00

Tuple #01 (!)

Tuple #02

Durable Storage

Write-Ahead Log

Tuple Delta

Snapshots

Tuple #01 (!)
IN-PLACE UPDATES ENGINE

In-Memory

In-Memory

Durable

Duplicate Data

Recovery Latency

Tuple #01

Tuple #02

Snapshots

Write-Ahead Log

In-Memory Table Heap

Tuple #00

Tuple #02

CMU 15-721 (Spring 2018)
NVM-OPTIMIZED ARCHITECTURES

Leverage the allocator’s non-volatile pointers to only record what changed rather than how it changed.

The DBMS only has to maintain a transient UNDO log for a txn until it commits.
→ Dirty cache lines from an uncommitted txn can be flushed by hardware to the memory controller.
→ No REDO log because we flush all the changes to NVM at the time of commit.
NVM IN-PLACE UPDATES ENGINE

NVM Index

NVM Table Heap
- Tuple #00
- Tuple #01
- Tuple #02

NVM Storage
- Write-Ahead Log
NVM IN-PLACE UPDATES ENGINE

NVM Index

NVM Table Heap

1. Tuple #01
2. Tuple #02
3. Tuple #00

NVM Storage

Write-Ahead Log

Tuple Pointers
NVM IN-PLACE UPDATES ENGINE

NVM Index

NVM Table Heap

1
Write-Ahead Log
Tuple Pointers

2

NVM Storage

Tuple #00

Tuple #01 ()

Tuple #02
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Leaf 2

Page #00

Page #01
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Leaf 2

Updated Leaf 1

Page #00

Page #01

Page #00
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Page #00

Dirty Directory

Leaf 2

Page #01

Updated Leaf 1

Page #00
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Dirty Directory

Leaf 1

Leaf 2

Updated Leaf 1

Page #00

Page #01

Page #00
NVM COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1
- Tuple #00

Leaf 2
- Tuple #01
NVM COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1
- Tuple #00

Leaf 2
- Tuple #01

Updated Leaf 1
- Only Copy Pointers
- Tuple #00 (!)
NVM COPY-ON-WRITE ENGINE

Current Directory

- Leaf 1
  - Tuple #00

Dirty Directory

- Leaf 2
  - Tuple #01

Master Record

Updated Leaf 1

- Only Copy Pointers
  - Tuple #00 (!)
LOG-STRUCTURED ENGINE
LOG-STRUCTURED ENGINE

MemTable

Write-Ahead Log

Tuple Delta

SSTable

Bloom Filter
LOG-STRUCTURED ENGINE

MemTable

Write-Ahead Log

Tuple Delta 1

SSTable

Bloom Filter

Tuple Delta 2

Tuple Data 3
LOG-STRUCTURED ENGINE

⚠️ Duplicate Data

⚠️ Compactions

Tuple Data
NVM LOG-STRUCTURED ENGINE

MemTable

SSTable

Write-Ahead Log

Bloom Filter

Tuple Delta

Tuple Data
NVM LOG-STRUCTURED ENGINE

MemTable

Write-Ahead Log

Tuple Delta

SSTable

Bloom Filter

Tuple Delta

Tuple Data

1

2

3
NVM LOG-STRUCTURED ENGINE

MemTable

Write-Ahead Log

Tuple Delta

1
NVM SUMMARY

Storage Optimizations
→ Leverage byte-addressability to avoid unnecessary data duplication.

Recovery Optimizations
→ NVM-optimized recovery protocols avoid the overhead of processing a log.
→ Non-volatile data structures ensure consistency.
GPU ACCELERATION

GPUs excel at performing (relatively simple) repetitive operations on large amounts of data over multiple streams of data.

Target operations that do not require blocking for input or branches:
→ Good: Sequential scans with predicates
→ Bad: B+Tree index probes

GPU memory is (usually) not cache coherent with CPU memory.
GPU ACCELERATION
GPU ACCELERATION
GPU ACCELERATION

- PCIe Bus (~16 GB/s)
- DDR4 (~40 GB/s)
- NVLink (~25 GB/s)
GPU ACCELERATION

Choice #1: Entire Database
→ Store the database in the GPU(s) VRAM.
→ All queries perform massively parallel seq scans.

Choice #2: Important Columns
→ Return the offsets of records that match the portion of the query that accesses GPU-resident columns.
→ Have to materialize full results in CPU.

Choice #3: Streaming
→ Transfer data from CPU to GPU on the fly.
HARDWARE TRANSACTIONAL MEMORY

Create critical sections in software that are managed by hardware.
→ Leverages same cache coherency protocol to detect transaction conflicts.
→ Intel x86: Transactional Synchronization Extensions

Read/write set of transactions must fit in L1 cache.
→ This means that it is not useful for general purpose txns.
→ It can be used to create latch-free indexes.
**HTM PROGRAMMING MODEL**

**Hardware Lock Elision (HLE)**
- Optimistically execute critical section by *eliding* the write to a lock so that it appears to be free to other threads.
- If there is a conflict, re-execute the code but actually take locks the second time.

**Restricted Transactional Memory (RTM)**
- Like HLE but with an optional fallback codepath that the CPU jumps to if the txn aborts.
HTM LATCH ELISION

Insert Key 25
HTM LATCH ELISION

Insert Key 25
HTM LATCH ELISION

Insert Key 25
HTM LATCH ELISION

Insert Key 25
HTM LATCH ELISION

Insert Key 25
HTM LATCH ELISION

Insert Key 25
HTM LATCH ELISION

Insert Key 25

TSX-START { 
LATCH A
Read A
LATCH C
UNLATCH A
Read C
LATCH F
UNLATCH C
}

TSX-COMMIT
Insert 25
UNLATCH F
HTM LATCH ELISION

Insert Key 25

TSX-START

LATCH A
Read A
LATCH C
UNLATCH A
Read C
LATCH F
UNLATCH C

TSX-COMMIT

Insert 25
UNLATCH F
HTM LATCH ELISION

Insert Key 25

TSX-START { LATCH A
Read A
LATCH C
UNLATCH A
Read C
LATCH F
UNLATCH C
}

TSX-COMMIT
Insert 25
UNLATCH F
HTM LATCH ELISION

Insert Key 25

TSX-START

{ LATCH A
Read A
LATCH C
UNLATCH A
Read C
LATCH F
UNLATCH C
}

TSX-COMMIT

Insert 25
UNLATCH F
PARTING THOUGHTS

Designing for NVM is important
→ Non-volatile data structures provide higher throughput and faster recovery

Byte-addressable NVM is going to be a game changer when it comes out.
NEXT CLASS

Final Exam Handout

snowflake