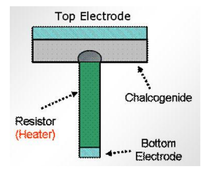
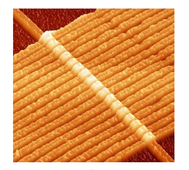
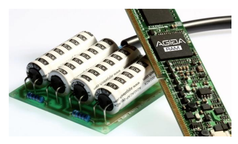
Scalable Logging through Emerging Non-Volatile Memory

Tianzheng Wang and Ryan Johnson Department of Computer Science, University of Toronto

What? Traditional RDBMS ***doesn't scale*** on multicore, multi-socket hardware
Why? Various centralized bottlenecks, especially ***logging***
How? Distributed logging + ***byte-addressable, non-volatile memory***

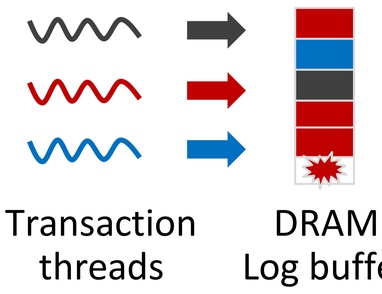






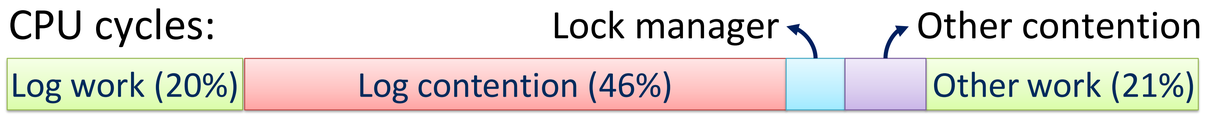
PCM – Memristor – NVDIMM

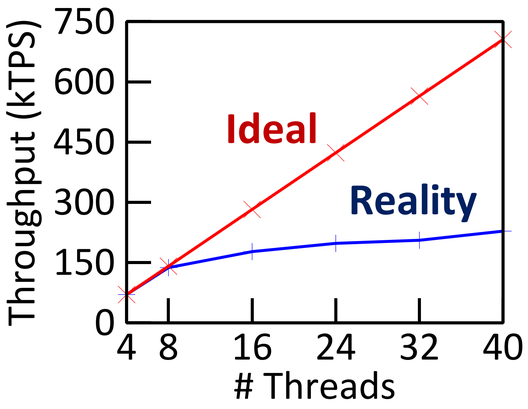
Traditional, *centralized* logging



❶ Grab log buffer mutex
❷ Write log records
❸ Release mutex
❹ Flush log buffer at commit

CPU cycles:





Contention for the ***single log buffer*** is a serious bottleneck

Group and async. commit?
- Better I/O performance
- But contention unchanged

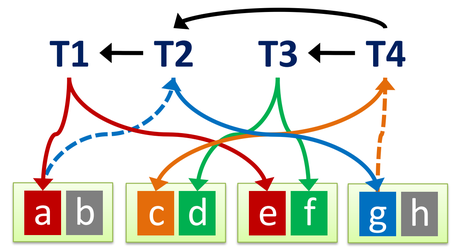
Distributed logging considered impractical

It reduces buffer contention, but…

❶ **Log space partitioning:** by page or xct?
– Impacts locality, recovery strategy

❷ **Dependency tracking:** before commit, ***T4*** must ***persist*** log records written by:
– itself
– direct xct deps: ***T4*** → ***T2***
– direct page deps: ***T4*** → ***T3***
– ***transitive deps:*** ***T4*** → {***T3***, ***T2***} → ***T1***

❸ **Storage is slow**
– T4 flushes all four logs upon commit (instead of one)



Heavy dep-tracking + I/O overheads = **A prohibitively slow d-log based system!**

Solution: buffer log records in byte-addressable, non-volatile memory (de-stage to disk/flash)

- **NVM: A brave new world of storage**
  - Persistent like disk/flash, byte-addressable like DRAM
  - DIMM form factor, attached to the memory bus
  - Performance similar to DRAM
- **Available today: DRAM backed by flash/super-capacitor**

- ***NVM as log buffers*** **– log records durable once written**
  - No dependency-tracking
  - No flush-before-commit

➡ ***NVM allows a cheap d-log!***

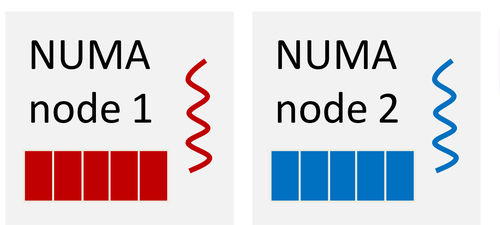
- **Major distributed logging and NVM challenges**
  - Partial order of log records, NUMA effect, volatile CPU cache

Distributed logging and NVM challenges

❶ **Log records only partially ordered**
– Recovery needs total order within any log/page/transaction
Solution: logical clock style ***global sequence number (GSN)***
– Update page, log and xct GSNs at each access

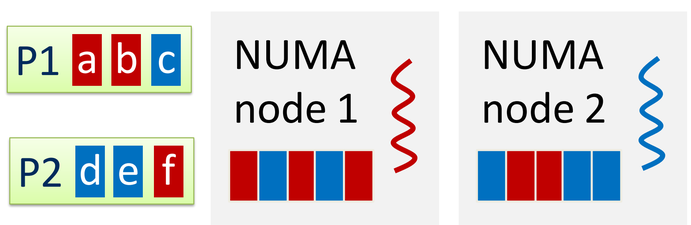
❷ **NUMA effect – threads prefer to access local NVM node**

Transaction level:



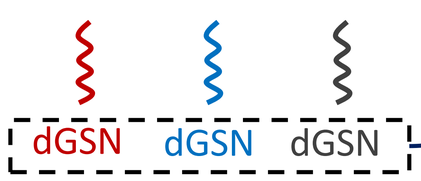
☺ **NUMA-friendly**

Page level:



☹ **Cross NUMA boundary**

❸ **Durability – CPU cache still volatile!** Records must leave CPU before commit, preferably without heavy dependency tracking
Solution: passive group commit
– *Workers* flush own caches, record dGSN, enqueue xct
– *Commit daemon* monitors min dGSN, dequeues durable xct



on commit:
1. Flush local caches
2. Update **local dGSN**
3. Enqueue transaction

Passive group commit daemon

Get ***min*** dGSN: 8

Dequeue xct with dGSN <= 8

Transaction	dGSN
~~Xct 1~~	~~5~~
Xct 2	10

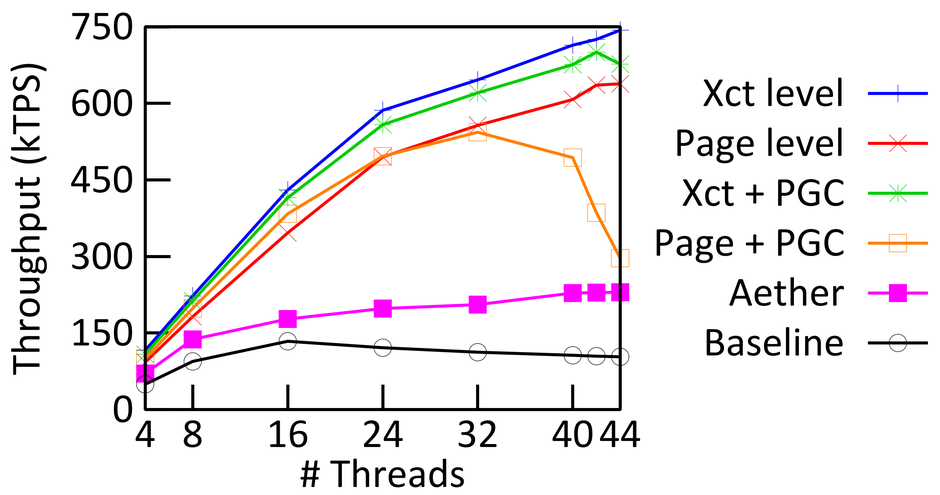
Commit queue

Performance

HW: 4-socket 6-core Intel E7-4807, 64GB RAM, data on tmpfs
Implemented in Shore-MT, comparing systems:
❶ **Baseline:** traditional centralized logging
❷ **Aether :** state-of-the-art centralized logging
❸ **Distributed logging** : page/xct level + passive group commit

TATP
Update Location
(stress test)

~3x better



TPC-C
Transaction Mix
(write-heavy)

~2x better