Beyond Main Memory
Anti-Caching in Main Memory Database Systems

Justin DeBrabrant
Brown University
Talk Overview

1. anti-caching architecture
   - larger than memory datasets in main memory DBMS

2. anti-caching + persistent memory
   - ditching the disk (finally!)
A bit of history…
1974 – System R

- query optimization
- recovery
- transaction serialization
  - allows concurrent execution of transactions
  - lots of locks
Change is Good

Price per GB of DRAM

Great, that’s what the buffer pool is for...right?
More Memory $\rightarrow$ Higher Throughput?

- all data resides in memory (i.e. in buffer pool)
  - No disk stalls
- must still...
  - maintain buffer pool
  - lock/latch data
  - maintain ARIES-style recovery logs
- question: What is the overhead of all these things?
OLTP Through the Looking Glass, and What We Found There

SIGMOD '08

Buffer Pool: 31%
Locking: 31%
Recovery: 26%
Real Work: 12%
Ok, no buffer pool, how about a distributed cache?
Distributed Caches

- e.g. memcached
- just in-memory key-value pair
- no inherent persistence
- application programmer must maintain consistency
  - or not!
So, we need a hybrid of these two architectures.
H-Store: A High-Performance, Distributed Main Memory Transaction Processing System

Copyright 2008 VLDB Endowment, ACM 000-0-00000-000-0/00/00.
VLDB '08, publisher, ACM. or to redistribute to lists, requires a fee and/or special permission from the
and notice is given that copying is by permission of the Very Large Data
the VLDB copyright notice and the title of the publication and its date appear,
Permission to copy without fee all or part of this material is granted provided
shortjived transaction executionsi that are hindered by
systems have particular propertiesi such as repetitive and
the workloads for onjline transaction processing eOLTPf
of architectural components inherited from the original Sysj
systems [ui v]k Many of these traditional systems use a myriad
outperform traditional or "one size fits all" database sysj
1. INTRODUCTION
lenges inherent in this operating environmentk
insight on the development of a distributed main memory
formi and to implement some of the ideas presented in the
have since set out to design a more complete execution platj
were obtained using a barejbones prototype that was develj
databases by a significant factork These resultsi howeveri
redesign of RDBMSsk This previous work showed that such
of stored procedures are factors that portend a cleanjslate
main memoryi the lack of user stallsi and the dominant use
ticulari the availability of multiplejcoresi the abundance of
crasingly falling short of optimal performance [nm]k In parj
cation shifts have resulted in modern OLTP databases inj
Our previous work has shown that architectural and applij
ABSTRACT
H-Store:
Jonathan Natkins
Robert Kallman
Brown University

Alexander Rasin
Massachusetts Institute of Technology

Evan P . C. Jones
Samuel Madden

Michael Stonebraker

Daniel J. Abadi
Y ale University

John Hugg
Vertica Inc.

H-Store: A High-Performance, Distributed Main Memory Transaction Processing System

VLDB 2008. 1496
H-Store Architecture

- partitioned, shared-nothing
  - data is sharded across nodes
- single-threaded main memory execution
  - no need for locks and latches
H-Store Architecture (cont’d)

▸ stored procedures
  ▪ no ad hoc queries in OLTP
  ▪ command logging

▸ recovery
  ▪ snapshots + command log
H-Store Assumptions

1. OLTP workload
   - short-lived transactions that touch only a few records at a time

2. mostly single-site transactions
   - distributed transactions need multi-node coordination

3. data fits in memory
   - virtual memory is bad!
YCSB, update-heavy, data < memory
Assumptions: Revisited

1. OLTP workload
   ▶ “One Size Fits All”: An Idea Whose Time Has Come and Gone
     ▶ ICDE ‘05
   ▶ The End of An Architectural Era: (Its Time for A Complete Rewrite)
     ▶ VLDB ‘07
Assumptions: Revisited

2. mostly single-site transactions

- Skew-Aware Automatic Database Partitioning In Shared-Nothing, Parallel OLTP Systems
  - *SIGMOD* ‘12
- On Predictive Modeling For Optimizing Transaction Execution in Parallel OLTP Systems
  - *VLDB* ‘11
Assumptions: Revisited
Workload Skew Exists!

▸ hot data in memory
▸ cold data to disk

▸ goals
  ▸ maintain transactional consistency
  ▸ avoid blocking
Anti-Caching
Anti-Caching Phases

▸ evict
▸ pre-pass
▸ fetch
▸ merge
Evict

1. data > anti-cache threshold
2. dynamically construct anti-cache blocks of coldest tuples
3. asynchronously write to disk
Pre-Pass

1. a transaction enters pre-pass when evicted data is accessed
2. continues execution, creating list of evicted blocks
3. abort, queue blocks to be fetched
Fetch

1. data is fetched asynchronously from disk
   ▶ avoids blocking

2. copied into merge buffer
Merge

1. previously aborted transaction is restarted
2. moves data from merge buffer to normal table
3. transaction executes normally
Multiple Restarts

- in-memory data for restarted transaction is relatively cold
  - mark tuples in pre-pass phase as hot
- data dependencies with evicted tuples
  - mark recently merged tuples as hot
- larger-than-memory queries still an issue
  - not in OLTP
Other Design Points

- LRU chain
  - embedded in tuple headers $\rightarrow O(1)$ updates
- EvictedTable
  - stores $<\text{tuple id, block id}>$ pairs for evicted tuples
- anti-cache
  - BerkeleyDB
Sounds like swapping...
Anti-Caching vs. Swapping

- fine-grained eviction
- blocks constructed dynamically
- non-blocking fetches
- remove disk from critical path
Sounds like caching…
Anti-Caching vs. Caching

- data exists in exactly one location
  - caching architectures have multiple copies, must maintain consistency
  - data is moved, not copied
- goal is data size, not throughput
Benchmarking

- YCSB
- Zipfian skew
- data > memory
- read/write mix
- MySQL, MySQL + memcached
YCSB, read-only, data 8X memory

The graph shows the throughput of different systems as a function of workload skew. The systems compared are:

- **anti-cache**
- **MySQL**
- **MySQL + memcached**

As the workload skew decreases, the throughput for all systems decreases as well, indicating a saturation point. The graph highlights the point of **Disk Saturation** for the given data sets.
Conclusions

- 8-17X improvement for skewed workloads at 8X memory
  - avoids blocking for disk
  - fine-grained eviction
- disk becomes the bottleneck
Anti-Caching: A New Approach to Database Management System Architecture

Justin DeBrabant
Brown University
debabrant@cs.brown.edu

Andrew Pavlo
Brown University
pavlo@cs.brown.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

Stan Zdonik
Brown University
szdonik@cs.brown.edu

ABSTRACT

The traditional wisdom for building disk-based relational database management systems (DBMSs) is to organize data into relations stored in disk blocks, with a memory buffer pool to mediate between disk and main memory. However, recent advances in CPU and memory technology have fostered the development of new DBMS architectures that put the entire database in main memory, including the buffer pool. These in-memory DBMSs (iDBMSs) offer significant performance benefits, but they are limited by the memory available on each machine. To address this limitation, we propose a new approach that we call anti-caching. This approach builds on the traditional iDBMS architecture based on main-memory metadata data. To overcome the restriction of using all data in main memory, we propose a new technique, called anti-caching, where cold data is moved to disk as transactional side effects occur in the database. This allows cold data to be fetched from disk as needed, while still maintaining an eviction order policy. The key idea behind anti-caching is to allow the database to maintain a header for each disk block that points to its current version in main memory. Whenever a user retrieves a tuple, the DBMS first checks to see whether the block already exists in the buffer pool. If not, it is fetched from disk as needed, and then the tuple is returned to the user. This allows the iDBMS to maintain a header that describes the location of the tuple in main memory, and the system can maintain an eviction order policy, allowing the iDBMS to evict older data in a disk-based DBMS. Our results show that this approach can significantly improve performance, and it is especially effective for heavily skewed read/write mixes, where cold data is moved to disk as transactional side effects occur in the database. The key idea behind anti-caching is to allow the iDBMS to maintain a header for each disk block that points to its current version in main memory. Whenever a user retrieves a tuple, the DBMS first checks to see whether the block already exists in the buffer pool. If not, it is fetched from disk as needed, and then the tuple is returned to the user.
Memory Latencies (cycles)

- SRAM: 1-30
- DRAM: 100-300
- flash: 25,000-2,000,000
- disk: 5,000,000+
NVM

› non-volatile
› random-access
› high write endurance
  › except flash
› byte-addressable
  › except flash
The Arms Race

- FeRAM
  - high write endurance
- MRAM
  - DRAM-like latency
- PCM (PRAM)
  - DRAM-like capacity
FeRAM, MRAM, or PCRAM, combines the advantages of SRAM, DRAM, and flash.

Good opportunity to rethink the memory hierarchy design.
NVM Emulation

- goal: provide product-independent analysis
- test wide range of latency profiles
- automatically add specified latency
- built by collaborators at Intel
H-Store + NVM Architectures

1. Anti-Cache to H-Store
   + fully utilizes memory hierarchy
   - added memory overhead

2. H-Store on NVM
   + no anti-cache overhead
   - wear-leveling necessary
Architecture 1

Application

Primary Storage

Anti-Cache

DRAM

NVM
Architecture 2

Application

Primary Storage

NVM
Architecture 1, YCSB, read-only, data 8X memory

![Graph showing throughput vs. workload skew with different latency conditions: 2X latency (red), 4X latency (blue), and 16X latency (green).]
Work in Progress

› implementation of Architecture 2
  › also in H-Store

› further benchmarking
  › YCSB, TPC-C, possibly others
  › Architecture 1 vs. Architecture 2

› paper in preparation
Conclusions

▸ hardware has changed

▸ anti-caching to disk has 18-17X speedup over disk-based architecture
  ▸ better utilization of available memory

▸ next-generation persistent memories extend the benefits of anti-caching beyond skewed workloads
Collaborators

Stan Zdonik (Brown)
Andy Pavlo (CMU)
Mike Stonebraker (MIT)
Stephen Tu (MIT)
Joy Arulraj (CMU)
Questions?

debrabant@cs.brown.edu