Anti-Caching:
A New Approach to Database Management System Architecture
DeBraband et al., 2013

July 7 2021
Nanda Kishore Sreenivas
Overview

There are issues with disk-based DBMS and main-memory DBs.

Solution: ‘Anti-caching’ is a MM DB, where cold data is moved to the disk.

Result: Upto 9x faster than disk-based DBMS with cache.
Existing problems

Disk blocks and a buffer pool of blocks in memory.

When query needs a disk block, first check in memory. If not available, evict a block.

Problem: Substantial overhead, $\frac{1}{3}$ of CPU cycles to maintain buffer pool*.

Main memory DBs were introduced as a solution.

Limitation: Database size must fit in memory; page faults occur if not.

*S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. OLTP through the looking glass, and what we found there. In SIGMOD, pages 981–992, 2008.
Two-tier model

Issues:

1. Data may reside in cache and buffer pool.
2. Cache and DBMS synchronization.
Anti-caching

- Primarily main-memory system.
- When full, “cold” data is moved to disk.
- Data resides only in one place at a time.
Advantages over Virtual Memory

Anti-caching is very similar to virtual memory swapping in OS. So, why not use that to do the paging of data*?

- Fine-grained eviction
  Anti-caching offers tuple-level eviction instead of a whole page.
- Non-blocking fetches
  OS blocks the process when a page fault occurs => no transaction during fetch.
  In anti-caching, transaction aborted and restarted later.

Anti-caching System Model

When size of database reaches threshold, cold tuples are evicted.

DBMS constructs a fixed-size block with LRU tuples and writes to disk.

A *catalog* is maintained (in memory) to keep track of evicted tuples.

If a transaction needs an evicted tuple, it moves to ‘pre-pass’ mode where all such evicted tuples it may need are identified.

Transaction is then aborted, and once data is available it is restarted.
Storage Architecture

**Block Table (D)**
- Hash table that maintains evicted blocks.
- All blocks of same size.
- Serialized evicted tuples from the same table.

**Evicted Table (M)**
- Keeps track of evicted tuples.
- 4-byte identifier = offset in the block.
- DBMS updates indexes to refer evicted table

![Figure 3: A logical representation of the layout of the in-memory Evicted Table and the disk-resident Block Table. The arrows represent integer offsets of tuples in a block.](image-url)
Storage Architecture (contd.)

LRU Chain

- All tuples of each table sorted in LRU order.
- Doubly linked list.
- The pointers are embedded in the tuples’ headers.
- To reduce CPU overhead, only some transactions update the chain.
  Rate of sampling controlled by parameter $0 < \alpha \leq 1$. 
Block Eviction

Amount of data accessed in each table is monitored.

#evictions from each table is inversely proportional to its “hotness” since last eviction.

After determining how many tuples are to be evicted, 1+ block(s) are constructed.

Group of blocks written to disk in single sequential write.
receive transaction → execute transaction → evicted data accessed?

YES → requeue transaction → merge blocks

NO → commit

pre-pass execution → fetch blocks
Transaction Execution

Simple case: If all required tuples are in normal storage, just proceed.

Pre-pass Phase

- Entered when the first evicted tuple is encountered.
- For each evicted tuple required, get the block-id and offset from ET.
- Required in-memory tuples are updated, to avoid eviction.

Transaction is aborted, and restarted when necessary data is fetched from disk.

Rarely, multiple pre-passes are required.
Block Retrieval

- Non-blocking read to retrieve from disk (on another thread).
- The retrieved blocks are kept in a *staging buffer* invisible to queries.
- DBMS performs “stop and copy”, where the retrieved data is merged.

Block-merging:

- All tuples are merged, required ones to the end of LRU chain; others at top (?).
- High overhead, unnecessary tables will be evicted soon again.

Tuple-merging:

- Only required tuples are merged; others discarded.
- Stale copy in disk, but cannot be accessed as the lookup in ET is erased.
- Holes accumulate; if holes are too high, do block-merging.
Experiments

1. H-Store (anti-caching)
2. MySQL (disk-based)
3. MySQL + MemCached (2-tier system with a cache at the top)

1. Yahoo Cloud Serving Benchmark (YCSB)
   ~20 GB, 20 mil. records, 1 table with 10 columns.
2. TPC-C (data that mimics an online store)
   ~10 GB, multiple tables.
Figure 6: YCSB experiments. In aLRU, $\alpha = 0.01$. 
YCSB Results

Skewed workload => Older data accessed much less frequently. Lower skew is hard to handle, because caches/memory needs to be rewritten often.

aLRU in this case is 1 in 100 sampling, so always better than LRU.

For high data size with high skew, there are considerable benefits of anti-caching. Compared with MySQL+MemCached, it is 4x (R-H) and 9x (W-H).

MemCached offers benefit in read-only workloads. Writes cause sync problems and low skew means a lot of cache misses.
Block-merge vs Tuple-merge

Tuple-merge (hole threshold = 50%) is consistently better than block-merge. Why?

1. Large merge costs
2. Uneviction/re-eviction cycle causes overhead

![Figure 8: Merge Strategy Analysis](image)

- YCSB read-only, 2x memory, 1MB evict blocks.

(a) Eviction/Uneviction Costs. For each block size, the left bar represents the block-merge costs and the right bar represents the tuple-merge costs.
Block and Tuple Sizes

1. Throughput decreases with block size; added costs of fetching larger blocks.
2. Larger tuples result in higher throughput.
   See #tuples, and for each there is cost associated with ET entries and update index.
LRU Chain Overhead

For high skew, double LL is 20x faster. Why?

- Choosing for eviction, and adding is O(1).
- But, updating a tuple is O(n).
- DLL allows traversing from the bottom.
- Memory overhead of DLL is worth it.

![Figure 10: Eviction Chain Analysis.](image)
Future Work

Three major issues:

1. Larger-than-memory Queries
   Few proposals outlined, but no performance gain is expected.
2. Block reorganization
   Lazy compaction in tuple-merging isn’t ideal.
   Consider semantic relation between tuples, and put them in same block.
3. Query Optimization
   Evict a few columns in a tuple instead.
4. Identifying cold data
   Only LRU is considered, there may be better methods.
Related papers: https://hstore.cs.brown.edu/publications/

Some related work that may be interesting:
