Anti-Caching: A New Approach to Database Management System Architecture

Justin DeBrabant
Andrew Pavlo
Stephen Tu
Michael Stonebraker
Stan Zdonik

Presented by:
Suman Nakshatri
DBMS Architectures

(a) Disk-oriented DBMS

(b) Disk-oriented DBMS with a Distributed Cache
Main Memory DBMS: H-Store

- Single-Partition Transaction: Transactions are local to a single node.

- Multi-Partition Transaction: Transactions consist of multiple phases; one or more of the phases touches multiple partitions.
Main Memory DBMS

- Performance - Amazing!!

- What happens if db size > main memory?
  
  - OS starts to page virtual memory -> Page faults -> Transaction execution stalled until data is fetched.

- Remedy:
  
  1. Provision new hardware and migrate database to a larger cluster -> Not Scalable!
  
  2. Fall back to a traditional disk-based system -> Back to Square One!
Solution: Anti-Caching

Straight Forward Concept

- Coldest tuples removed from Main Memory.
- Opposite to Traditional DBMS.
- Similar to Virtual Memory Swapping.
Solution:
Anti-Caching

Straight Forward Concept

• Coldest tuples removed from Main Memory.

• Opposite to Traditional DBMS.

• Similar to Virtual Memory Swapping.

......... eh?
What’s different?

• With anti-caching, it is the responsibility of the DBMS to read and write data as needed.

• Advantages:
  1. Fine-grained control of the data evicted to disk.
  2. Non-blocking reads of evicted data from disk.
What’s different?

• Fine-Grained Eviction:
  • In virtual memory, OS makes eviction decisions at the page-level.
  • Anti-caching, eviction decisions are performed at the tuple-level.

• Non-Blocking Fetches:
  • In a virtual memory system, the OS blocks a process when it incurs a page fault.
  • In Anti-caching, a transaction that accesses evicted data is simply aborted and then restarted at a later point once the data that it needs is retrieved from disk.

  • Pre-pass execution.
System Model

Evicted Table

<table>
<thead>
<tr>
<th>BlockId</th>
<th>TupleId</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>&lt;offset&gt;</td>
</tr>
<tr>
<td>999</td>
<td>&lt;offset&gt;</td>
</tr>
<tr>
<td>997</td>
<td>&lt;offset&gt;</td>
</tr>
<tr>
<td>997</td>
<td>&lt;offset&gt;</td>
</tr>
<tr>
<td>997</td>
<td>&lt;offset&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Block Table

- `<blockId>`
- `<creation-timestamp>`
- `<tuple-length>`
- `<tuple-data>`
- `<string-data>`

LRU Chain

```plaintext
newest
4   6   01101010101010101010
--  3   110101010001111111
2   4   1010001010101000
3   1   1010000010101010
6   --  0101101010101010
1   5   0000010101101010
...```

oldest
If data needed is from evicted data:
1. Abort transaction.
2. Non-blocking read to retrieve evicted data.
3. Reschedule the transaction.
Merging Data Back to Memory

• Block-Merging
  • Entire block is merged.
  • Requested tuples added at the back of LRU.
  • Tuples not needed added to the front.
  • Overhead?
    • Continuous un-eviction/re-eviction cycle.

• Tuple-Merging
  • Only the requested tuples are merged back.
  • The fetched block is discarded back to disk, however tuples are not removed from the block.
  • Cons?
    • Duplicate data in memory and disk.
    • Generation of “holes” - Valid data in each block is reduced. —> Lazy block compaction.
Experiments

• What was compared?
  • MySQL - which uses disk storage with main memory buffer pool.
  • MySQL + Memcached - distributed cache.
  • H-Store with Anti-Caching - Main memory DB.

• What data was used for benchmarking?
  • YCSB
  • TPC-C
YCSB
TPC-C

Graph showing the performance of different systems with varying data sizes.
Results
(In terms of throughput)

• Anti-caching Vs MySQL (for datasets 8 * memory)
  • 9 * for read-only
  • 18 * for read-heavy
  • 10 * on write-heavy workloads.

• Anti-caching Vs Memcached (for datasets 8 * memory)
  • 2 * for read-only
  • 4 * for read-heavy
  • 9 * on write-heavy

• Anti-caching provides a 7 * improvement in throughput over the other architectures.
Other Experiments

• Block-merge Vs Tuple-merge

• Block-size and Tuple-size Analysis
Future Work

• Restricts the scope of queries to fit in main memory.
  • Table locks.
  • Timestamp ordering.
  • Dirty Reads.

• Block Reorganization/ Lazy compaction.
  • Holes accumulation puts garbage data in blocks.
  • Lazy merging will still lead to cold data being merged.
  • Semantically related data put in the same block.

• Query optimization
  • Evict specific unused columns instead of the who tuple.
Critique/ Discussion

- Is Aborting the query the right way to go?

- Can DBMS find all the evicted data in pre-pass?

- Can it lead to series of same transaction aborting?

- Waste of time executing query twice. Is the user kept waiting?

- Looks good for OLTP transaction, may be not that great for Analytical queries?

- How does anti-caching perform against original H-store with data > memory size?
Competition

DataBlitz
SAP’s-HANA
TimesTen
Hyper
MemSQL
RAMCloud
Hyper
TxCache
Calvin
PRISMA/DB
Project-Siberia
EXTremeDB
VoltDB
Figure 1: VoltDB throughput when paging to a SSD
Hekaton/Project Siberia

- Fully integrated with Microsoft SQL server.
- Framework Siberia manages hot and cold data.
- No LRU chain or evicted table, hence better resource utilization.
- A sample of records are logged for the classification algorithm to classify hot and cold records. Migration done on record granularity.
- Reads bring cold data to the transaction in-private cache. After processing, cache is committed back to cold storage.
- Inserts are done on hot storage.
- Updates make a new entry in hot storage and remove the cold data in disk.
- An update memo to hold the current status of cold records.
- Designed for OLTP.
- MVCC applied for concurrency control in transactions. Unlike Anti-caching, they are not executed serially.
HyPer/ScyPer

• Hybrid OLTP and OLAP high performance db.

• Transactions executed serially, parallelism with partitioning.

• OLAP executed on virtual memory snapshot by forking a child process.

• Relies on virtual memory paging for data overflow.

• SyPer has a primary-secondary architecture, primary for OLTP and secondary for OLAP queries.
Enabling Efficient OS Paging for Main-Memory OLTP Databases

- Rely on the OS’s virtual memory paging mechanism.
- Tuple level re-organization done offline.
- Cold data is moved to a memory location that is more likely to be paged to disk by the OS.
- How is it different from Anti-caching?
  - No transaction aborts.
Conclusion

• A new architecture to handle data > memory size.

• Memory is primary storage with cold data evicted to disk.

• Cold data on the tuple level is found using LRU chain.

• Two methods of merging cold data back to memory.

• Performs much better than both disk based and caching based dbs.

• Leaves some open questions for future work.