In-memory Data Management Systems — Challenges and Opportunities

Zhang Hao
zhangh@comp.nus.edu.sg
National University of Singapore

23 Feb. 2016@RUC
Outline

• Introduction
  – why in-memory?
• Hardware Innovation
  – NUMA
  – HTM
  – RDMA
• System Calls
• Anti-caching
Outline

• Introduction
  – why in-memory?

• Hardware Innovation
  – NUMA
  – HTM
  – RDMA

• System Calls

• Anti-caching
“Memory is the new disk, disk is the new tape”

- **SPEED** is **EVERYTHING** in business
- Low-latency and Real-time
  - Disk I/O KILLS everything

Pain Points of Big Data
(Source: Aberdeen Group Survey)
“Memory is the new disk, disk is the new tape”

- DRAM becomes **BIGGER** and **CHEAPER**
- CPU is much **STRONGER**

![Graph showing the decrease in cost per MB from 1980 to 2014.](image)

- Multi-Core Architecture (8 x 8core CPU per blade)
- Massive parallel scaling with many blades
- One blade ~$50,000 = 1 Enterprise Class Server
- 64bit address space – 2TB in current servers
- 100GB/s data throughput
- Dramatic decline in price/performance
Speed

CPU
- Core
- L1 Cache
- L2 Cache
- L3 Cache

L1 Cache: 1ns
L2 Cache: 4ns
L3 Cache: 7ns

Main Memory: 100ns

NVM: 200ns-1
- 50K

Disk: 4M ns

Intel Core i5
Outline

• Introduction
  – why in-memory?

• Hardware Innovation
  – NUMA
  – HTM
  – RDMA

• System Calls

• Anti-caching
HW Innovation – NUMA

- NUMA node owns its local memory
- Different access speed to local/remote memory
HW Innovation – NUMA

Example: Topology of our epic server (likwid-topology) (epic.d1.comp.nus.edu.sg)
HW Innovation – NUMA

• Related works – NUMA-aware
  – shared-nothing architecture within one NUMA server, e.g., Bubba [1], Gamma [2]
  – Hardware islands with UNIX sockets [3]
  – Scheduling: marsel query execution [5]
  – Specific algorithms: sort-merge join [7]
Outline

• Introduction
  – why in-memory?

• Hardware Innovation
  – NUMA
  – HTM
  – RDMA

• System Calls

• Anti-caching
HW Innovation – HTM

- Optimistic concurrency control in HW level

Intel® TSX Interface: HLE

Library

Application

acquire_lock (mutex)
; do critical section
; function calls,
; memory operations,...
release_lock (mutex)

mov mutex, 0

If lock not free, execution will abort either early (if pause used) or when lock gets free

Commit HLE execution

mov mutex, 0
Limitations

- The transaction size is limited to the size of L1 data cache.
- Cache associativity makes it more prone to false conflicts.
- HTM transactions may be aborted due to interrupt events.
• Related works
  – A database transaction is divided into a set of relatively small HTM transactions with timestamp ordering (TSO) concurrency control and minimizing the false abort probability via data/index segmentation [8].
  – protects single data read, and validate/write phases using HTM transactions [9]
Outline

• Introduction
  – why in-memory?

• Hardware Innovation
  – NUMA
  – HTM
  – RDMA

• System Calls

• Anti-caching
HW Innovation – RDMA

• Remote Direct Memory Access

OS or CPU is not involved!
HW Innovation – RDMA

• Comparison with Ethernet TCP/IP
HW Innovation – RDMA

- Related works
  - Pilaf [10]: multiple one-sided RDMA READ with self-verifying data structures for GET operations
  - HERD [11]: reducing latency (RDMA WRITE from client, and SEND from server); RDMA-specific features (e.g., inlining, selective signaling)
In-Memory Big Data Management and Processing: A Survey [12]
Outline

• Introduction
  – why in-memory?
• Hardware Innovation
  – NUMA
  – HTM
  – RDMA
  – NVRAM
• System Calls
• Anti-caching
Facts

Simply moving the storage layer from disk to memory will not enable the DB to take full advantage of the memory performance.

Many reasons:
1) Pointer chasing
2) Cache unfriendly data structures
3) System calls
4) …
The Problem with System Calls

A system call is required every time an application requires “service” from the OS.

1. File management
2. Device management
3. Communication
4. Process control
5. Information maintenance
The Problem with System Calls

Syscalls have two main problems

1. Introduce latency

2. Unsuitable abstraction for accessing memory
   - E.g. “read” syscall can read from a file (disk/SSD/NFS) or a network socket or arbitrary file-mapped device
System Calls in Databases

Four sources of system calls:
1. Data accesses
   - open/close/read/write/stat …
2. Communication among workers
   - socket/listen/accept/connect/sendmgs/recvmsg …
3. Synchronization among workers
   - pthread_mutex_lock/unlock, sem_wait/sem_post …
4. Fault tolerance and recovery
   - a mixture of the above

There are methods to replace most system calls during the basic operation of an in-memory database.
Towards No Syscalls

Traditional DB using syscalls

MemepiC with minimal syscalls
Outline

• Introduction
  – why in-memory?
• Hardware Innovation
  – NUMA
  – HTM
  – RDMA
  – NVRAM
• System Calls
• Anti-caching
Facts

• Memory never enough
  – Memory is still relatively scarce compared to HDD
  – Energy consumption
    • Memory is a significant contributor to the total system power
  – N-minute rule
    • cheaper to put the data in memory if it is accessed every N-minute
    • Cold data – stay on disk
    • Hot data – resident in memory
Caching vs. “Anti-Caching”

• **Common**
  – Deal with the same level of storages

• **Difference**
  – Assumption about the memory size
  – Different primary data locations
  – Target for different types of systems
Components of anti-/caching

• Access tracking
  – Granularity: Tuple vs page

• Eviction Strategy
  – LRU, MRU, CLOCK, WSCLOCK

• Book-keeping
  – indexes, filters, page table, etc.

• Swapping strategy
  – How much, and when
## State-of-the-art Approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Access Tracking</th>
<th>Eviction Strategy</th>
<th>Book-keeping</th>
<th>Data Swapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-Store anti-caching</td>
<td>Tuple-level tracking</td>
<td>LRU</td>
<td>Evicted table and index</td>
<td>Block-level swapping</td>
</tr>
<tr>
<td>Hekaton Siberia</td>
<td>Tuple-level access</td>
<td>Offline classification</td>
<td>Bloom and range filter</td>
<td>Tuple-level migration</td>
</tr>
<tr>
<td>logging</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spark</td>
<td>N/A</td>
<td>LRU based on insertion</td>
<td>Hash table</td>
<td>Block-level swapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache Systems</td>
<td>Tuple-level tracking</td>
<td>LRU, approximate LRU, etc</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Buffer Management</td>
<td>Page-level tracking</td>
<td>LRU, MRU, CLOCK, etc</td>
<td>Hash table</td>
<td>Page-level swapping</td>
</tr>
<tr>
<td>OS Paging</td>
<td>h/w-assisted page-level</td>
<td>LRU, NRU, WSCLOCK, PPRA,</td>
<td>Page table</td>
<td>Page-level swapping</td>
</tr>
<tr>
<td></td>
<td>tracking</td>
<td>etc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficient OS Paging</td>
<td>Tuple-level access</td>
<td>Offline classification</td>
<td>OS-dependent</td>
<td>OS-dependent</td>
</tr>
<tr>
<td></td>
<td>logging</td>
<td>and OS Paging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Observer in</td>
<td>h/w-assisted page-level</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Hyper</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory protection</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access tracking - insights

• If the average tuple size is less than 4-KB for doubly-linked LRU list, their memory overheads are much higher than that of page-table-based method.
Eviction strategy - insights

• OS-based eviction approaches suffer from poor accuracy
  – Coarser-granularity
  – Lack of semantics information

• Access-logging based offline classification do well
Book-keeping - insights

• Index and eviction table: higher space overhead
• Bloom and other filters: quite space efficient
• Page table: hardware support
Swapping - insights

• Block/page-level swapping is efficient in terms of disk I/O throughput
User-space vs kernel-space

• At user/application level
  – More semantics information
  – Flexible granularities (tuple, column, row, tables, page)
  – Platform-independence (possible)

• At kernel level
  – Directly use hardware
  – General
  – Only know pages
Towards An Efficient General Approach

- User-space Virtual Memory Management (UVMM) [13]

• Three-layer Hierarchy
### Numbers Everyone Should Know

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>
Useful Linux Tools

• Measure lower-level numbers
Resources


- Slides: http://www.comp.nus.edu.sg/~a0095627
References

Thanks
Q/A