“ANTI-CACHING”-BASED ELASTIC MEMORY MANAGEMENT FOR BIG DATA

Hao Zhang ‡, Gang Chen †‡, Beng Chin Ooi ‡,
Weng-Fai Wong ‡, Shensen Wu †‡, Yubin Xia *

‡ yzBigData Co., Ltd.
‡ National University of Singapore
† Zhejiang University
* Shanghai Jiao Tong University
MOTIVATION

- In-memory databases for Big Data
- 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
OUTLINE

- Caching vs. “Anti-caching”
- State-of-the-art Approaches
- Understanding the components of anti-caching
- User-space Virtual Memory Management (UVMM)
- Conclusions
CACHING VS. “ANTI-CACHING”

- **Common**
  - Deal with the same level of storages

- **Difference**
  - Assumption about the memory size
  - Target for different types of systems
  - Different primary data locations
THE COMPONENTS OF IN-MEMORY DB

- **Access tracking**
  - Granularity: Tuple vs page
- **Eviction Strategy**
  - LRU, MRU, CLOCK, WSCLOCK
- **Book-keeping**
  - Tables (hashed or otherwise), indexes, etc.
- **Swapping strategy**
  - What, how much, and when
USER VS KERNEL

► At user/application level
  ▶ More semantics information
  ▶ Different granularities (tuple, column, row, tables, page)
  ▶ Platform-independence (possible)

► At kernel level
  ▶ Directly use hardware
  ▶ Only know pages

► Crossing the user-kernel divide is expensive

“ANTI-CACHING”-BASED ELASTIC MEMORY MANAGEMENT FOR BIG DATA
<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 logging</td>
<td>Offline classification</td>
<td>Bloom and range filter</td>
<td>Tuple-level migration</td>
</tr>
<tr>
<td>Spark</td>
<td>N/A</td>
<td>LRU based on insertion time</td>
<td>Hash table</td>
<td>Block-level swapping</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 tracking</td>
<td>LRU, NRU, WSCLOCK, PPRA, etc</td>
<td>Page table</td>
<td>Page-level swapping</td>
</tr>
<tr>
<td>Efficient OS Paging</td>
<td>Tuple-level access logging</td>
<td>Offline classification and OS Paging</td>
<td>OS-dependent</td>
<td>OS-dependent</td>
</tr>
<tr>
<td>Access Observer in Hyper</td>
<td>h/w-assisted page-level tracking</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
A DETAILED STUDY OF THE COMPONENTS

- **Platform**
  - Implement different approaches inside one system – Memcached
    - To avoid interference introduced by other components
    - More fair to various approaches
    - Simple to monitor and perf

- **Benchmark**
  - YCSB (synthetic)
    - Varying skewness
    - Varying ratio of available memory to data
ACCESS TRACKING OVERHEAD

(d) Real Time Overhead

Real-time Overhead Ratio [%]

VMA Protection ---
LRU - - -
Logging - - -
ALRU - - -

Workload Skew

0.99 0.75 0.50 0.25
ACCESS TRACKING - INSIGHTS

- Virtual memory access (VMA) is very expensive.

- If the average tuple size is less than 4-KB for doubly-linked LRU list, or 1-KB for ALRU, their memory overheads are much higher than that of page-table based tracking.
EVICION STRATEGY

“ANTI-CACHING”—BASED ELASTIC MEMORY MANAGEMENT FOR BIG DATA
EVICION STRATEGY - INSIGHTS

- Kernel-based eviction approaches suffer from poor accuracy
  - Lack of semantics information
- Access-logging based offline classification do well
- LRU/ALRU do reasonably well
BOOK-KEEPING - INSIGHTS

- Book-keeping using indexed eviction table has higher space overhead
- Bloom and other filters are quite space efficient
- Page tables and VMA use tables that are there anyway. So overhead is lowest.
TOWARDS AN EFFICIENT GENERAL APPROACH
- USER-SPACE VIRTUAL MEMORY MANAGEMENT (UVMM)

Three-layer Hierarchy

- Applications
  - key-value store, analytics
  - logging(addr, size)
  - malloc

- Allocator
  - memory allocation
  - allocate(addr, size)

- UVMM
  - memory/disk management
  - page table

- Memory

- Disk
TOWARDS AN EFFICIENT GENERAL APPROACH
- USER-SPACE VIRTUAL MEMORY MANAGEMENT (UVMM)

- Design Principles
  - No indirection
    - Real pointer
  - Non-intrusiveness
    - Backward compatibility and transparent upgrading
    - API-based (e.g., malloc)
  - Flexibility
    - List of options for different levels of intrusiveness
    - Optional user-provided access logging
TOWARDS AN EFFICIENT GENERAL APPROACH
- USER-SPACE VIRTUAL MEMORY MANAGEMENT (UVMM)

Design Principles

- Reduced CPU overhead for normal operations
  - Page table
  - User-supplied access logging
- Reduced Memory overhead
  - Page level tracking
  - Access distribution provided by logging (within one page)
TOWARDS AN EFFICIENT GENERAL APPROACH
- USER-SPACE VIRTUAL MEMORY MANAGEMENT (UVMM)

- Implementation
  - Access Tracking
    - *A combination of access tracking methods*
    - *Page table, malloc-inject, access logging, etc.*
  - Eviction Strategy
    - *Optimized LRU/WSCLOCK with consideration of user-provided access logging*
    - *Standard eviction strategies: aging-based LRU, WSCLOCK, FIFO, RANDOM*
TOWARDS AN EFFICIENT GENERAL APPROACH
- USER-SPACE VIRTUAL MEMORY MANAGEMENT (UVMM)

- Implementation
  - Book-keeping
    - VMA protection
  - Data swapping
    - Compression = lz4
    - Kernel Asynchronous I/O
PUTTING THEM TOGETHER

Throughput

(a) Throughput \( \frac{M}{D} = \frac{3}{4} \)

(b) Throughput \( \frac{M}{D} = \frac{1}{2} \)

(c) Throughput \( \frac{M}{D} = \frac{1}{4} \)

LRU – H-store
OS Paging – standard
UVMM – our proposal

ALRU – Redis
Efficient OS Paging – Stoica & Ailamaki
Logging - Hekaton Siberia
CONCLUSIONS

- User- and kernel-space approaches exhibit different strengths
  - User-space: more application semantics, finer operation granularity, more accurate eviction strategy
  - Kernel-space: hardware (CPU, I/O) assistance, good resource utilization
- Combination of user- and kernel-space approaches needed for the best anti-caching performance
  - CPU, I/O performance, memory utilization
  - General and efficient
  - User-space virtual memory management (UVMM)
THANK YOU
Q&A