International Journal of Information Technology and Management

Vol. V, Issue No. I, August-2013, ISSN 2249-4510

A NEW APPROACH OF DBMS ARCHITECTURE FOR DATA ADMIITANCE IN MEMORY THROUGH ANTI-CACHING TECHNIQUE

A New Approach of DBMS Architecture for Data Admiitance in Memory through Anti-Caching Technique

Abul Hasan Khan[1] Dr. D. K. Pandey[2]

[1]Research Scholar, CMJ University, Shillong, Meghalaya

[2]Assistant Professor

Abstract – The traditional wisdom for building disk-based relational database management systems (DBMS) is to organize data in heavily-encoded blocks stored on disk, with a main memory block cache. In order to improve performance given high disk latency, these systems use a multi-threaded architecture with dynamic record-level locking that allows multiple transactions to access the database at the same time.

Previous research has shown that this results in substantial overhead for on-line transaction processing (OLTP) applications. The next generation DBMSs seek to overcome these limitations with architecture based on main memory resident data. To overcome the restriction that all data fit in main memory, we propose a new technique, called anti-caching, where cold data is moved to disk in a transaction ally-safe manner as the database grows in size. Because data initially resides in memory, an anti-caching architecture reverses the traditional storage hierarchy of disk-based systems. Main memory is now the primary storage device.

We implemented a prototype of our anti-caching proposal in a high-performance, main memory OLTP DBMS and performed a series of experiments across a range of database sizes, workload skews, and read/write mixes. We compared its performance with an open-source, disk-based DBMS optionally fronted by a distributed main memory cache. Our results show that for higher skewed workloads the anti-caching architecture has a performance advantage over either of the other architectures tested of up to 9x for a data size 8x larger than memory.

In this paper, we analyze state-of-the-art approaches to achieving this goal for in-memory databases, which is called as "Anti-Caching" to distinguish it from traditional caching mechanisms. We conduct extensive experiments to study the effect of each fine-grained component of the entire process of "Anti-Caching" on both performance and prediction accuracy. To avoid the interference from other unrelated components of specific systems, we implement these approaches on a uniform platform to ensure a fair comparison. We also study the usability of each approach, and how intrusive it is to the systems that intend to incorporate it. Based on our findings, we propose some guidelines on designing a good "Anti-Caching" approach, and sketch a general and efficient approach, which can be utilized in most in-memory database systems without much code modification.

--------------------------♦----------------------------

INTRODUCTION

Data is invaluable in product prediction, scientific exploration, business intelligence, and so on. However, the sheer quantity and velocity of Big Data have caused problems in data capturing, storage, maintenance, analysis, search, and visualization. The management of a huge amount of data is particularly challenging to the design of database architectures.

In addition, in many instances when dealing with Big Data, speed is not an option but a must. For example, Facebook makes an average of 130 internal requests sequentially for generating the HTML for a page, thus making long latency data access unacceptable. Supporting ultra-low latency service is therefore a requirement. Effective and smart decision making is enabled with the utilization of Big Data, however, on the condition that real-time analytics is possible. Otherwise, profitable decisions could become stale and useless. Therefore, efficient real-time data analytics is important and necessary for modern database systems.

Distributed and No SQL databases have been designed for large scale data processing and high scalability, while the Map-Reduce framework provides a parallel and robust solution to complex data computation. Synchronous Pregel-like message-passing or asynchronous Graph Lab processing models have been utilized to tackle large graph analysis, and stream processing systems have been designed to deal with the high velocity of data generation. Recently, in-memory database systems have gained traction as a means to significantly boost the performance.

Towards In-memory Databases - The performance of disk-based databases, slowed down by unpredictable and high access latency of disks, is no longer acceptable in meeting the rigorous low-latency, real-time demands of Big Data. The performance issue is further exacerbated by the overhead (e.g., system calls, buffer manager) hidden by the I/O flow. To meet the strict real-time requirements for analyzing massive amount of data and servicing requests within milliseconds, an in-memory database that keeps data in the main memory all the time is a promising alternative.

In-memory databases have been studied as early as the 80s. Recent advances in hardware technology re-kindled the interest in implementing in-memory databases as a means to provide faster data accesses and real-time analytics. Most commercial database vendors (e.g., SAP, Microsoft, Oracle) have begun to introduce in-memory databases to support large-scale applications completely in memory. Nevertheless, in-memory data management is still at its infancy with many challenges, and a completely in-memory design is not only still prohibitively expensive, but also unnecessary. Instead, it is important to have a mechanism for in-memory databases that utilize both memory and disks effectively. It is similar to the traditional caching process, which is however the other way around: instead of fetching data that is needed from disk into main memory, cold data is evicted to disk, and fetched again only when needed. In this case, main memory is treated as the main storage, while disk acts as a backup. We call it as “Anti-Caching”, to emphasize the opposite direction of data movement. The goal of “Anti-Caching” is to enable in-memory databases to have the “capacity as data, speed as memory and price as disk”, being a hybrid and alternative between strictly memory based and disk-based databases.

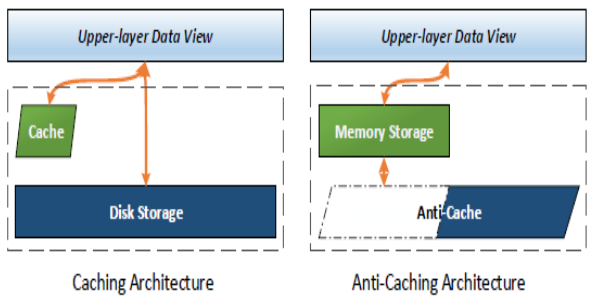
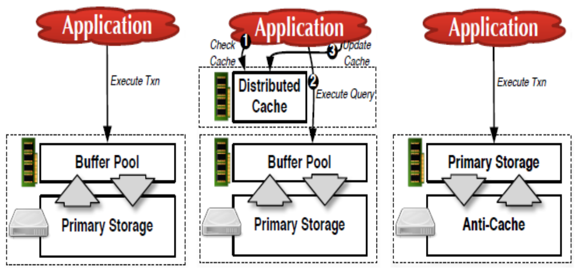


Fig. 1: Caching vs “Anti-Caching”

THE ANTI-CACHING ARCHITECTURE

The ideal system for these larger-than-memory on-line transaction processing (OLTP) datasets would have similar performance to a main memory system while scaling to handle larger-than-memory datasets as gracefully as a traditional disk-based system. To address this need, we have constructed a new architecture for main memory database systems called anti-caching. The goal of anticaching is to allow the DBMS to handle larger-than-memory datasets while still maintaining a throughput advantage over a traditional disk-based architecture. To do this, we take advantage of the skew inherent in most on-line transaction processing (OLTP) workloads. In a DBMS with anti-caching, when memory is exhausted, the DBMS gathers the “coldest” tuples and writes them to the on-disk anti-cache with minimal translation from their main memory format, thereby freeing up space for more recently accessed tuples. As such, the “hotter” data resides in main memory, while the colder data resides on disk in the anti-cache portion of the system.

Unlike a traditional DBMS architecture, tuples do not reside in both places; each tuple is either in memory or in a disk block, but never in both places at the same time. In this new architecture, main memory, rather than disk, becomes the primary storage location. Rather than starting with data on disk and reading hot data into the cache, data starts in memory and cold data is evicted to the anti-cache on disk. This is the inverse of what happens in a traditional disk-based architecture where hot data is cached in the buffer pool, and for this reason we call this approach anti-caching.



(a) Disk-oriented DBMS (b) Disk-oriented DBMS with a Distributed Cache (c) Main Memory DBMS with Anti-Caching

Figure 2: An overview of architecture for larger-than-memory OLTP datasets. In (a) and (b), the disk is the primary storage for the database and data is brought into main memory as it is needed. With the anti-caching model shown in (c), memory is the primary storage and cold data is evicted to disk.

The first is that anti-caching is non-blocking. In a virtual memory system, the OS blocks a process when it incurs a page fault from reading a memory address that is on disk. For certain DBMSs, this means that no transactions are executed while the virtual memory page is being fetched from disk. In an anti-caching DBMS, 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. In the meantime, the DBMS continues to execute other transactions without blocking. Lastly, since every page fault triggers a disk read, queries that access multiple evicted pages will page fault several times in a sequential fashion. We instead use a pre-pass execution phase that attempts to identify all evicted blocks needed by a transaction, which will allow all blocks to be read together. Another key advantage of anti-caching over virtual memory is the granularity at which data can be evicted. In anti-caching, eviction decisions are performed at the tuple-level.

The goal of anti-caching is to get the performance benefits of a main memory system while being able to scale to datasets larger than main memory. In general, there are only three architectures that are able to handle larger-than-memory on-line transaction processing (OLTP) datasets: (1) a traditional disk-based architecture, (2) a traditional disk-based architecture fronted by a main memory distributed cache, and (3) an anti-caching architecture.

STATE-OF-THE-ART APPROACHES

In general, state-of-the-art “Anti-Caching” approaches can be classified into three categories.

1) *User-space:* “Anti-Caching” performed in the userspace is the most commonly used approach. This enables ad hoc optimizations based on application semantics and fine-grained operations. On the other hand, the need to cross the OS boundary for I/O introduces additional overhead and constraints.

2) *Kernel-space:* Virtual memory management (VMM), which is available in most operating systems, can be regarded as a simple and general solution for “Anti-Caching” since the current generation of 64-bit OS supports up to 128 TB (247 bytes) of virtual address space, which is sufficient for most database applications. However, due to the lack of upper-layer application semantics, kernel space solutions often suffer from inaccurate eviction decisions.

Furthermore, the constraint of operating in units of pages when swapping data incurs extra overhead.

3) *Hybrid of user- and kernel-space:* A hybrid approach that combines the advantages from both semantics-aware user-space and hardware-assisted kernel-space approaches is promising. This can be done in either a user-centric or kernel centric way. Such an approach would exploit the application’s semantics as well as the efficiency provided by the OS in handling disk I/O, access tracking and book-keeping. It can also act as a general approach for most systems, rather than an ad hoc solution for a specific system.

ANTI-CACHING SYSTEM MODEL

We call our architecture anti-caching since it is the opposite architecture to the traditional DBMS buffer pool approach. The disk is used as a place to spill cold tuples when the size of the database exceeds the size of main memory. As stated earlier, unlike normal caching, a tuple is never copied. It lives in either main memory or the disk based anti-cache.

At runtime, the DBMS monitors the amount of main memory used by the database. When the size of the database relative to the amount of available memory on the node exceeds some administrator defined threshold, the DBMS “evicts” cold data to the anti-cache in order to make space for new data. To do this, the DBMS constructs a fixed-size block that contains the least recently used (LRU) tuples from the database and writes that block to the anti-cache. It then updates a memory-resident catalog that keeps track of every tuple that was evicted. When a transaction accesses one of these evicted tuples, the DBMS switches that transaction into a “pre-pass” mode to learn about all of the tuples that the transaction needs. After this pre-pass is complete, the DBMS then aborts that transaction (rolling back any changes that it may have made) and holds it while the system retrieves the tuples in the background. Once the data has been merged back into the in-memory tables, the transaction is released and restarted.

We now describe the underlying storage architecture of our antic ache implementation. We then discuss the process of evicting cold data from memory and storing it in the non-volatile anti-cache. Then, we describe how the DBMS retrieves data from the antic ache. All of the DBMS’s operations on the anti-cache are transactional and any changes are both persistent and durable.

Storage Architecture - The anti-cache storage manager within each partition contains three components: (1) a disk-resident hash table that stores evicted blocks of tuples called the Block Table, (2) an in-memory Evicted Table that maps evicted tuples to block ids, and (3) an in-memory LRU Chain of tuples for each table. As with all tables and indexes in H-Store, these data structures do not require any latches since only one transaction is allowed to access them at a time.

Block Eviction - Ideally, our architecture would be able to maintain a single global ordering of tuples in the system, thus globally tracking hot and cold data. However, the costs of maintaining a single chain across partitions would be prohibitively expensive due to the added costs of inter-partition communication. Instead, our system maintains a separate LRU Chain per table that is local to a partition. Thus, in order to evict data the DBMS must determine (1) what tables to evict data from and (2) the amount of data that should be evicted from a given table. For our initial implementation, the DBMS answers these questions by the relative skew of accesses to tables.

The amount of data accessed at each table is monitored, and the amount of data evicted from each table is inversely proportional to the amount of data accessed in the table since the last eviction. Thus, the hotter a table is, the less data will be evicted. For the benchmarks tested, this approach is sufficient, but we expect to consider more sophisticated schemes in the future.

Transaction Execution - Main memory DBMSs, like H-Store, owe their performance advantage to processing algorithms that assume that data is in main memory. But any system will slow down if a disk read must be processed in the middle of a transaction. This means that we need to avoid stalling transaction execution at a partition whenever a transaction accesses an evicted tuple. We now describe how this is accomplished with anti-caching.

Block Retrieval - After aborting a transaction that attempts to access evicted tuples the DBMS schedules the retrieval of the blocks that the transaction needs from the Block Table in two steps. The system first issues a non-blocking read to retrieve the blocks from disk. This operation is performed by a separate thread while regular transactions continue to execute at that partition. The DBMS stages these retrieved blocks in a separate buffer that is not accessible to queries. Any transaction that attempts to access an evicted tuple in one of these blocks is aborted as if the data was still on disk.

Distributed Transactions - Our anti-caching model also supports distributed transactions. H-Store will switch a distributed transaction into the “pre-pass” mode just as a single-partition transaction when it attempts to access evicted tuples at any one of its partitions. The transaction is aborted and not required until it receives a notification that all of the blocks that it needs have been retrieved from the nodes in the cluster. The system ensures that any in-memory tuples that the transaction also accessed at any partition are not evicted during the time that it takes for each node to retrieve the blocks from disk.

ANTI-CACHE MEMORY OPTIMIZATIONS

One of the assumptions made in the design of the anti-caching architecture is that workload skew is very dynamic, and will likely change throughout the course of workload execution.

Because of this, we decided that all workload tracking (i.e., the mechanisms used to determine how recently or frequently a tuple has been accessed) should also be done dynamically, on-line, during the normal execution of transactions. If it was the case that workload skew is present, but is relatively static, it would be possible to determine the hot and cold areas of data offline, and give this information to the system at runtime in order to guide eviction policies. This approach of learning workload skew offline in the context of main memory databases has been explored in previous work. However, this approach is unlikely to adapt to a quickly changing workload skew. In this paper, we proposed an approximation of an LRU eviction policy where transactions are randomly sampled from the workload, where, if selected, the transaction will cause an update to the LRU chain depending on which tuples are accessed. The control over the sampling rate allows control over how aggressively transactions are sampled, with more transactions being sampled more likely to adapt to changing workload conditions but incurring additional overhead in updating the LRU chain. The runtime performance of updating the LRU chain is negligible.

However, the time taken to update the LRU chain is only part of the cost. The other is the memory overhead of storing the LRU chain information. In the current implementation, each tuple in the LRU chain stores a 4-byte ID of both the previous and next tuple in the chain. Given that the point of anti-caching is to allow the system to better-utilize the available memory resources, this memory overhead is less than ideal. This memory overhead also means that anti-caching is more effective on larger tuple sizes, since more memory can be reclaimed each time a tuple is evicted relative to the in-memory per-tuple overhead of anti-caching.

For an anti-caching architecture, determining hot and cold tuples is an essential part of evicting data to disk, since evicting hot data to disk would have a significant negative impact on system performance. While the LRU chain is one way to do this, it requires the additional memory overhead of storing the LRU chain pointers in memory. In this section we present a timestamp-based method for tracking when tuples are accessed. Like the aLRU method, this is an approximate approach, with the goal of evicting some of the cold tuples, not necessarily the coldest tuples, which requires maintaining and exact ordering of when tuples are accessed. The basic idea of using timestamps is that each tuple is assigned a n-byte timestamp that can be controlled depending on the granularity of the eviction decisions desired. At runtime, each time a tuple is accessed; its corresponding timestamp is updated to the current system timestamp, which will depend on the number of bytes used in the timestamp. During eviction, a sample of tuples is selected, and those with the oldest timestamp are evicted. The number of tuples selected in a sample can be controlled to improve the likelihood that older tuples are chosen; the larger then sample size relative to the number of tuples to evict, the higher the likelihood that the tuples selected for eviction will be among the coldest.

CONCLUSION

In this paper, we presented a new architecture for managing datasets that are larger than the available memory while executing OLTP workloads. With anti-caching, memory is the primary storage and cold data is evicted to disk. Cold data is fetched from disk as needed and merged with in-memory data while maintaining transactional consistency. We also presented an analysis of our antic aching model on two popular OLTP benchmarks, namely YCSB and TPC-C, across a wide range of data sizes and workload parameters.

On the workloads and data sizes tested our results are convincing. For skewed workloads with data 8x the size of memory, anti-caching has an 8x-17x performance advantage over a disk-based DBMS and a 2x-9x performance advantage over the same disk-based system fronted with a distributed main memory cache. We conclude that for OLTP workloads, in particular those with skewed data access, the results of this study demonstrate that anti-caching can outperform traditional architectures popular today.

The “Anti-Caching” approach enables in-memory database systems to handle big data. In this paper, we conducted an indepth study on the state-of-the-art “Anti-Caching” approaches that are available in user- and kernel-spaces by considering both CPU and I/O performance, and their consequential runtime system throughput. We found that user- and kernelspace approaches exhibit strengths in different areas. More application semantics information is available to user-space approaches which also have finer operation granularity. This enables a more accurate eviction strategy.

REFERENCES

- Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. ICDE, pages 195–206, 2011.
- Fitzpatrick and A. Vorobey, “Memcached: a distributed memory object caching system,” http://memcached.org/, 2003.
- Fitzpatrick. Distributed Caching with Memcached. Linux J., 2004(124):5–, Aug. 2004.
- R. K. Ports, A. T. Clements, I. Zhang, S. Madden, and B. Liskov. Transactional consistency and automatic management in an application data cache. OSDI’10, pages 1–15, 2010.
- Sidlauskas and C. S. Jensen, “Spatial joins in main memory: Implementation matters!” in *PVLDB ’15*, 2014, pp. 97–100.
- Graefe, H. Volos, H. Kimura, H. Kuno, J. Tucek, M. Lillibridge, and A. Veitch, “In-memory performance for big data,” in *PVLDB ’15*, 2014.
- J. Baulier, P. Bohannon, S. Gogate, S. Joshi, C. Gupta, A. Khivesera, H. F. Korth, P. McIlroy, J. Miller, P. P. S. Narayan, M. Nemeth, R. Rastogi, A. Silberschatz, and S. Sudarshan. Datablitz: A high performance main-memory storage manager. VLDB, pages 701–, 1998.
- J. DeBrabant, A. Pavlo, S. Tu, M. Stonebraker, and S. Zdonik. Anti-caching: A new approach to database management system architecture. Proc. VLDB Endow., 6(14): 1942–1953, Sept. 2013. ISSN 2150-8097.
- 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. ISBN 978-1-60558-102-6.