Lecture #01: In-Memory Databases

15-721 Advanced Database Systems (Spring 2019)
https://15721.courses.cs.cmu.edu/spring2019/
Carnegie Mellon University
Prof. Andy Pavlo

1 Background

The history of DBMSs development is about dealing with the limitations of hardware. The first DBMSs in the 1970s were designed in environment with the following characteristics:

- Uniprocessor (single core CPU)
- RAM was severely limited
- Database had to be stored on disk
- Disk is slow. **Seriously slow**

But now DRAM capacities are large enough that most structured databases will entirely fit in memory. This merits us to rethink all aspects of the DBMS to account for this. This course is about ways to do this.

2 Disk-Oriented Database Management Systems

For a disk oriented DBMS, the system architecture is predicated on the assumption that data is stored in non-volatile memory. This means that the DBMS may have to read data from disk during query execution.

In a disk-based system, only approximately 7% of instructions are done on actual work [3]. The majority of the DBMS's instructions time are in managing three of its key components: (1) buffer pool, (2) concurrency control, (3) logging/recovery.

Buffer Pool

The DBMS organizes the database as a set of fixed-length blocks called *slotted pages*. The system uses an in-memory (volatile) *buffer pool* to cache the blocks cached from disk.

- When a query accesses a page, the DBMS checks to see if that page is already in memory.
- If not, the DBMS retrieves the memory from disk and copies it into a frame in its buffer pool.
- Once the page is in memory, the DBMS translates any on-disk addresses to their in-memory addresses.
- Every tuple access has to go through the buffer pool manager regardless of whether that data will always be in memory.

Concurrency Control

In a disk oriented DBMS, the system assumes that a transaction could stall at any time when it tries to access data that is not in memory.

The system's concurrency control protocol allows the DBMS to execute other transactions at the same time to improve performance while still preserving atomicity and isolation guarantees.

Logging and Recovery

Most DBMS use `STEAL` + `NO-FORCE` buffer pool policies so all modifications have to be flushed to the WAL before a transaction can commit [2]. Log entries contain before and after image of record modified.

3 In-Memory Database Management Systems

The system architecture assumes that the primary storage location of the database is in memory. This means that the DBMS does not need to perform extra steps during execution to handle the case where it has to retrieve data from disk. If disk I/O is no longer the slowest resource, much of the DBMS architecture will have to change to account for other bottlenecks: [7]

- Locking/latching
- Cache-line misses
- Pointer chasing
- Predicate evaluation
- Data movement and copying
- Networking (between application and DBMS)

Data Organization

An in-memory DBMS splits the data for tuples into fixed-length and variable-length pools. Indexes use pointers to the fixed-length data for each tuple. These tuples then have 64-bit pointers to any variable-length values stored in a separate memory location.

Concurrency Control

In-memory DBMSs still use either a pessimistic or optimistic concurrency control schemes to interleave transactions. They will use modern variants of these algorithms that are designed for in-memory data storage. The new bottleneck is contention caused from transactions trying to access data at the same time.

One key difference is that an in-memory DBMS can store locking information about each tuple together with its data. This is because the cost of a transaction acquiring a lock is the same as accessing data. Contrast this with disk-oriented DBMSs where locks are physically stored separate from their tuples because the tuples may get swapped out to disk.

Indexes

Like with concurrency control schemes, in-memory DBMSs will use data structures for their indexes that are optimized for fast, in-memory access.

In-memory DBMSs will not log index updates. Instead, the system will rebuild the indexes upon restart when it loads the database back into memory. This avoids the runtime overhead of logging updates to indexes during transaction execution.

Query Processing

The best strategy for executing a query plan in a DBMS changes when all the data is already in memory. Sequential scans are no longer significantly faster than random access.

The traditional tuple-at-a-time iterator model is too slow because of function calls.

Logging and Recovery

The DBMS still needs WAL on non-volatile storage since the system could halt at anytime. In many cases, however, it may be possible to use more lightweight logging schemes (e.g., only store redo information). For example, since there are no "dirty pages", the DBMS does not need to maintain LSNs throughout the

systems. In-memory DBMSs still takes checkpoints to reduce the amount of log that the system has to replay during recovery.

4 Notable Early In-memory DBMSs

- **TimesTen**: Originally Smallbase [4] from HP Labs. Multi-process, shared memory DBMS. Bought by Oracle in 2005 [6].
- **Dali**: Multi-process shared memory storage manager using memory mapped files [**?**].
  P*TIME: Korean in-memory DBMS from the 2000s [1]. Lots of interesting features (e.g., hybrid storage layouts, support for larger-than-memory databases). Sold to SAP in 2005 and is now part of HANA.

References

[1] S. K. Cha and C. Song. P*time: Highly scalable oltp dbms for managing update-intensive stream workload. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 1033–1044, 2004. URL http://dl.acm.org/citation.cfm?id=1316689.1316778.

[2] M. J. Franklin. Concurrency control and recovery. In *Computing Handbook, Third Edition: Information Systems and Information Technology*, pages 12: 1–21. 2014. URL http://db.lcs.mit.edu/6.893/F04/ccandr.pdf.

[3] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. OLTP through the looking glass, and what we found there. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 981–992, 2008. doi: http://doi.acm.org/10.1145/1376616.1376713.

[4] M. Heytens, S. Listgarten, M.-A. Neimat, and K. Wilkinson. Smallbase: A main-memory dbms for high-performance applications. Technical report, Hewlett-Packard Laboratories, 1995.

[5] H. Jagadish, D. Lieuwen, R. Rastogi, and A. Silberschatz. Dali: A high performance main memory storage manager. *VLDB*, pages 48–59. URL http://www.vldb.org/conf/1994/P048.PDF.

[6] T. Lahiri, M.-A. Neimat, and S. Folkman. Oracle timesten: An in-memory database for enterprise applications. *IEEE Data Eng. Bull.*, 36(2):6–13, 2013. URL http://sites.computer.org/debull/A13june/TimesTen1.pdf.

[7] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1150–1160, 2007. URL http://hstore.cs.brown.edu/papers/hstore-endofera.pdf.