Lecture #02: In-Memory Database Management Systems

15-721 Advanced Database Systems (Spring 2018)
http://15721.courses.cs.cmu.edu/spring2018/
Carnegie Mellon University
Prof. Andy Pavlo

1 Background

- Much of the history of DBMSs is about dealing with the limitations of hardware.
- The first DBMSs in the 1970s were:
  - Uniprocessor (single core CPU)
  - RAM was severely limited
  - Database had to be stored on disk
  - Disk is slow. **Seriously slow**
- Now DRAM capacities are large enough that most structured databases will entirely fit in memory. Unstructured or semi-structured data sets are much larger (think log files stored on HDFS).

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. The database is organized 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.
- **Buffer Pool:**
  - 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 [3].
  - Log entries contain before and after image of record modified.
- In a disk-based system, only approximately 7% of instructions are done on actual work [4].

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: [8]
  - Locking/latching
  - Cache-line misses
  - Pointer chasing
  - Predicate evaluation
  - Data movement and copying
  - Networking (between application and DBMS)
- **Data Organization:**
  - In-memory DBMS will organize tuples in blocks/pages.
  - Direct memory pointers vs record ids.
  - Fixed-length vs variable-length data pools.
  - Use checksums to detect software errors from trashing the database.
- **Concurrency Control:**
  - Fine-grained locking allows for better concurrency but requires more locks.
  - Coarse-grained locking requires fewer locks but limits the amount of concurrency.
  - The DBMS can store locking information about each tuple together with its data.
  - The new bottleneck is contention caused from transactions trying to access data at the same time.
- **Indexes:**
  - Indexes are usually rebuilt in an in-memory DBMS after restart to avoid logging overhead.
  - Modern databases typically do not log index updates.
- **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.
  - May be possible to use more lightweight logging schemes (e.g., only store redo information).
  - Since there are no "dirty pages", we do not need to maintain LSNs throughout the systems.
  - System also still takes checkpoints, however, there are different methods for checkpointing.
- **Non-Volatile Memory:** Emerging hardware that is similar to speed of DRAM with the persistence guarantees of an SSD [1].
- Notable Early In-memory DBMSs:
  - **TimesTen**: Originally Smallbase [5] from HP Labs. Multi-process, shared memory DBMS. Bought by Oracle in 2005 [7].
  - **Dali**: Multi-process shared memory storage manager using memory mapped files [6].
  - **P*TIME**: Korean in-memory DBMS from the 2000s [2]. 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] J. Arulraj, A. Pavlo, and S. Dulloor. Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. In *Proceedings of the 2015 International Conference on Management of Data*, SIGMOD '15, pages 707–722, 2015.

[2] 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`.

[3] 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`.

[4] 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.

[5] 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.

[6] 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`.

[7] 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`.

[8] 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`.