Future of RDBMS is RAM Clouds & SSD

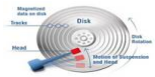
Posted 4 days back at igvita.com

Rumors of the demise of relational database systems are greatly exaggerated. The NoSQL movement is increasingly capturing the mindshare of the developers, all the while the academia have been talking about the move away from "RDBMS as one size fits all" for several years. However, while the new storage engines are exciting to see, it is also important to recognize that relational databases still have a bright future ahead - *RDBMS systems are headed into main memory, which changes the playing field all together*.

Performance is only one aspect that influences the choice of a database. Tree and graph structures are not easy to model within a relational structure, which in turn leads to complicated schemas and system overhead. For that reason alone, document-stores (Tokyo, CouchDB, MongoDB), graph stores (Neo4J), and other alternative data structure databases (Redis) are finding fertile ground for adoption. However, the end of "RDBMS as one size fits all" does not mean the end of relational systems all together. It is too early to bury RDBMS in favor of No (or Less) SQL. We just need to reset how we think about the RDBMS.

Disks are the New Tape

The evolution of disks has been extremely uneven over the last 25 years: disk capacity has increased 1000x, data transfer speeds increased 50x, while seek and rotational delays have only gone up by a factor of 2. Hence, if we only needed to transfer several hundred kilobytes of data in the mid 80's to achieve good disk utilization, then today we need to read at least 10MB of data to amortize the costs of seeking the data - refresh your memory on seek, rotational, and transfer times of our rusty hard drives.



When the best we can hope for is 100-200 IOPS out of a modern hard drive, the trend towards significantly larger block sizes begins to make a lot more sense. Whereas your local filesystem is likely to use 4 or 8kb blocks, systems such as Google's GFS and Hadoop's HDFS are opting out for 64MB+ blocks in order to amortize the cost of seeking for the data - by using much larger blocks, the cost of seeks and access time is once again brought down to single digit percent figures over the transfer time.

Hence, as we generate and store more and more data, the role of the disks must inevitably become more archival. Batch processing systems such as Map-Reduce are well suited for this world and are quickly replacing the old business intelligence (BI) systems for exactly these reasons. In the meantime, the limitations imposed by the random access to disk mean that we need to reconsider the role of disk in our database systems.

OLTP is Headed Into Main Memory & Flash

An average random seek will take 5-10ms when hitting the physical disk and hundreds of microseconds for accessing data from cache. Compare that to a fixed cost of 5-10 microseconds for accessing data in RAM and the benefits of a 100-1000x speed difference can be transformative. **Instead of treating memory as a cache, why not treat it as a primary data store?** John Ousterhout and his co-authors outline a compelling argument for "RAMCloud". After all, if Facebook keeps over 80% of their data in memcached, and Google stores entire indexes of the web in memory many times over, then your average database-backed application should easily fit and be able to take advantage of the pure memory model also.

The moment all of the data is available in memory, it is an entirely new game: access time and seek times become irrelevant (no disk seeks), the value of optimizing for locality and access patterns is diminished by orders of magnitude, and in fact, entirely new and much richer query models can enable a new class of data-intensive applications. In a world where the developer's time is orders of magnitude more expensive than the hardware (a recent phenomenon), this also means faster iterations and less data-optimization overhead.

The downside to the RAMCloud is the equivalent order of magnitude increase in costs - RAM prices are dropping, but dollar for dollar, RAMCloud systems are still significantly more expensive. Flash storage is an obvious compromise for both speed and price. Theoretical access time for solid-state devices is on the order of 50 microseconds for reads, and 200 microseconds for writes. However, in reality, wrapping solid-state storage in SATA-like hardware devices brings us back to ~200 microseconds for reads, or ~5000 IOPS. Though, of course, innovation continues and devices such as FusionIO's PCI-E flash storage controller bring us back to 80 microsecond reads at a cost of ~$11 per Gigabyte.

However, even the significantly higher hardware price point is often quickly offset once you factor in the saved developer time and adjacent benefits such as guaranteed performance independent of access patterns or data locality. Database servers with 32GB and 64GB of RAM are no longer unusual, and when combined with SSDs, such as the system deployed at SmugMug, often offer a much easier upgrade path than switching your underlying database system to a NoSQL alternative.

Database Architecture for the RAMCloud

Migrating your data into RAM or Flash yields significant improvements via pure speedup in hardware, however, "it is time for a complete rewrite" argument still holds: majority of existing database systems are built with implicit assumptions for disk-backed storage. These architectures optimize for disk-based indexing structures, and have to rely on multithreading and locking-based concurrency to hide latency of the underlying storage.

When access time is measured in microseconds, optimistic and lock-free concurrency is fair game, which leads to much better multi-core performance and allows us to drop thousands of lines of code for multi-threaded data structures (concurrent B-Trees, etc). RethinkDB is a drop-in MySQL engine designed for SSD drives leveraging exactly these trends, and Drizzle is a larger fork of the entire MySQL codebase aimed at optimizing the relational model for "cloud and net applications": massively distributed, lightweight kernel and extensible.

Migrating Into Main Memory

Best of all, you can start leveraging the benefits of storing your data in main memory even with the existing MySQL databases - most of them are small enough to make the memory buffers nothing but a leaky abstraction. Enable periodic flush to disk for InnoDB (innodb_flush_log_at_trx_commit=2), and create covering indexes for your data (a covering index is an index which itself contains all the required data to answer the query). Issue a couple of warm-up requests to load the data into memory and you are off to the races.

Of course, the above strategy is at best an intermediate solution, so investigating SSD's as a primary storage layer, and if you are adventurous, give RethinkDB a try. Also keep an eye on Drizzle as the first production release is aimed for summer of 2010. Alternative data storage engines such as Redis,

MongoDB and others are also worth looking into, but let us not forget: laws of physics still apply to NoSQL. There is no magic there. Memory is fast, disks are slow. Nothing is stopping relational systems from taking advantage of main memory or SSD storage.