MemSQL Architecture - The Fast (MVCC, InMem, LockFree, CodeGen) and Familiar (SQL)

Tuesday, August 14, 2012 at 9:13AM

Todd Hoff in MySQL, Product, databases, lock-free

This is an interview with MemSQL *cofounder's* Eric Frenkiel *and* Nikita Shamgunov, *in which they try to answer critics by going into more depth about their technology.*

MemSQL ruffled a few feathers with their claim of being the fastest database in the world. According to their benchmarks MemSQL can execute 200K TPS on an EC2 Quadruple Extra Large and on a 64 core machine they can push 1.2 million transactions a second.

Benchmarks are always a dark mirror, so make of them what you will, but the target market for MemSQL is clear: projects looking for something both fast and familiar. Fast as in a novel design using a combination of technologies like MVCC, code generation, lock-free data structures, skip lists, and in-memory execution. Familiar as in SQL and nothing but SQL. The only interface to MemSQL is SQL.

It's right to point out MemSQL gets a boost by being a first release. Only a limited subset of SQL is supported, neither replication or sharding are implemented yet, and writes queue in memory before flushing to disk. The next release will include a baseline distributed system, native replication, n-way joins, and subqueries. Maintaining performance as more features are added is a truer test.

And MemSQL is RAM based, so of course it's fast, right? Even among in-memory databases MemSQL hopes to convince you they've made some compelling design choices. The reasoning for their design goes something like:

- Modern hardware requires a modern database. The idea is to strip everything down and rethink it all out again.
- Facebook scaled for two reasons: Memcached and Code Generation. Memcached provides in-memory key-value access and HipHop translates PHP to C++. Applying those ideas to a database you get an in-memory database that uses SQL instead of KV.
- Since performance requires operating out of RAM, the first assumption is the data set fits in RAM.
- Reading fast from RAM has been solved by Memcached, which is basically a hash table sitting behind a network interface. What is not solved is the fast write problem.
- The fast write problem is solved by eliminating contention. The way to eliminate contention is by using lock-free data structures. Lock-free data structures scale well which is why MemSQL has faster writes than Memcached.
- Hash tables are an obvious choice for key-value data structures, but what would you use for range queries? Skip lists are the only efficient data structures for range queries and are more scalable than b-trees. Lock-free skip lists are also difficult to build.
- When competing with in-memory technologies you need to execute fewer instructions for each SQL query. This goal is met via code generation. In a traditional database system there is a fixed overhead per query to set up all the contexts, prepare the tree for interpretations, and then run the query through the plan. All that is

sidestepped by generating code, which becomes more of win as the number of cores and the number of queries increases.

- MVCC is a good match with an in-memory databases because it offers both efficiency and transactional correctness. It also supports fast deletes. Rows can be marked deleted and then cleaned up behind the scenes.

On the first hearing of this strange brew of technologies you would not be odd in experiencing a little buzzword fatigue. But it all ends up working together. The mix of lock-free data structures, code generation, skip lists, and MVCC makes sense when you consider the driving forces of data living in memory and the requirement for blindingly fast execution of SQL queries.

In a single machine environment MemSQL makes an excellent case for their architecture. In a distributed environment they are limited in the same way every distributed databases is limited. MVCC doesn’t offer any magic as it doesn’t translate easily across shards. The choices MemSQL has made reflect their primary use case of fast transactions plus fast real-time SQL based analytics. MemSQL uses a sharded shared nothing approach where queries are run independently on each shard and merged together on aggregation nodes. Transactions across shards won’t be supported until two phase commit is implemented, but then they will perform like any other database. What they really want to do well is run fast real-time aggregations across a cluster so that’s what their design reflects.

A lot of other questions come to mind with such a novel design. Will MemSQL perform common operations like “return the top 5 X” as programmers have come to expect? Will MemSQL still perform when hit with a wide variety of different SQL queries? They say yes given code generation and their data structure choices. Is SQL expressive enough to solve real world problems across many domains? When you start adding stored procs or user defined functions will the carefully orchestrated dance of data structures still work?

To answer these questions and more, let’s take a deeper look into the technology behind MemSQL.

Stats

- Largest Installation: 500 node cluster deployed on EC2 (4000 cores)
- Biggest single machine deployment: 320-core SGI supercomputer; 4 TB RAM
- 20 billion records in a single table on a single machine
- 1.25m inserts/sec on a single 64-core machine
- Run 25 16-core servers 24/7 for testing
- 15 employees, heavy on engineering

Information Sources

- Interview over Skype.
- Email Q&A.
- Everything listed under section Related Articles.

Use Cases

- Data is proliferating and there are always green field markets that need fixing. Giving a relational interface is a good way to get real-time solved in an understandable way.
- Targeted at high throughput workloads with small transactions in a heavily concurrent environment.
- Users with lots of CPU and RAM who need performance. MemSQL is a complex high performance piece of software. You’ll use MemSQL if you are making money.

- Answer what is happening right now questions. Most successful use case is simultaneous insert, which is only possible with a row based system, and simultaneous select, which supports real-time analytics, like min/max/distinct/ave etc.
- Not in the BI market. Works well with products like Vertica. MemSQL works well with real time analytics. Data inserts transactionally yet supports a real-time analytical layer.
- Relying on startups that value time over money. Build or buy? Time is the most valuable thing. Don’t need to create vertical infrastructure. Memcached is not needed which simplifies layers in the stack.
- Write-heavy, read-heavy workloads
    - Machine data
    - Traffic spikes
    - Streaming data

Origin Story

- Started in Jan 2011. Worked in stealth mode for 14 months.
- Both Eric and Nikita worked at Facebook and decided there was a better way to give SQL at scale and speed.
- They quit and applied to Y Combinator with zero lines of code. Y Combinator normally expects to see a demo but bought the argument that they could build a very fast database based on their experience. Nikita worked on the Microsoft SQL Server core engine and has other geek credentials. Eric worked on Platform at Facebook and Nikita worked on Infrastructure.
- Any Facebook partner that touched the social graph immediately saw problems with scale. Games would add 2-3 million users in a matter of weeks. Media companies would have to start tracking Like buttons and comments on social deployments.
- Aha moment was realizing not just Facebook had these problems.

Downstream traffic from social networks and new data sources like Capital markets that may have to consume a million messages a second.

Why Faster?

- Lock free + code generation + MVCC is faster on multiple cores than an approach using partitioning by core and serializing data structure access per core.
- Code generation minimizes code execution paths within queries and removes interpretation overhead. SQL is being hardwired into the server.
- C++ is used instead of Java.
- Skip lists are used instead of b-trees because b-trees don’t scale.
- CPU efficiency means higher throughput. Since MemSQL uses fewer instructions per query they can achieve higher throughput. More queries can be pushed through the system because they’ve minimized parsing, caching, and plan cache matching.

The Y Combinator Cabal

- The Y Combinator network is very powerful. Through Y Combinator they were able to get a first customer way before release, which helped them prioritize features and avoid the be everything to everybody trap.
- Their first customer grew to a million users in 6 weeks, put all their data in RAM, and supported just the SQL that they needed.

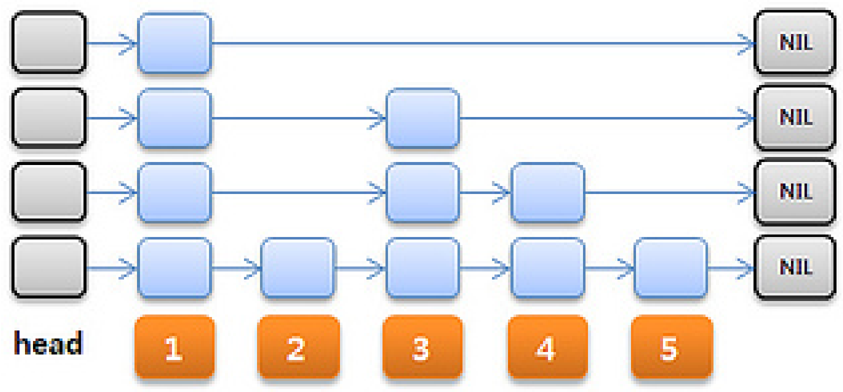
Lock-Free Data Structures

- Lock-free data structures scale exceptionally well as more resources (CPU, RAM) are added to a system. Lock-free data structures

minimize wasted CPU during points of high contention.
- Every component of the MemSQL engine is built on lock-free data structures: linked lists, queues, stacks, skip lists, and hash tables.
- Lock-free queues and stacks are used throughout the system for managing state in transactions and memory managers.
- The lock-free hash table is used to map query shapes to compiled plans in the plan cache.
- Lock-free skip lists and hash tables are available as index data structures.

Skip Lists

- A skip list is a popular data structure that performs extremely well in-memory. It shares many fundamental properties with a randomized tree. For example, it offers O(logN) time to seek to a specific value.
- The basic idea is that the bottom layer is a sorted linked list. Each higher layer is an “express” lane for the lists below. An element in layer i appears in layer i+1 with some fixed probability p (usually something like ½ or ¼).
- Among the data structures that offer the efficient seek and insert properties of a tree, skip lists are notable for being implemented lock-free and perform extremely well in highly concurrent environments.
- Skip lists have two main tradeoffs:
  - Compared to b-trees, skip lists are slightly slower for long sequential scans.
  - Lock free skip lists are unidirectional. So in MemSQL, you have to specify for each skip list index whether it should be ascending or descending. If you need both directions, you need two indexes.



MVCC

MemSQL implements multi version concurrency control to implement transactional semantics

- Every time a transaction modifies a row, MemSQL creates a new version that sits on top of the existing one. This version is only visible to the transaction that made the modification. Read queries the access the same row “see” the old version of this row.
- Versions in MemSQL are implemented as a lock-free linked list.
- MemSQL only takes a lock in the case of a write-write conflict on the same row. MemSQL takes a lock because it is easier to program against. The alternative would be to fail the second transaction, which would require the programmer to resolve the failure.
- MemSQL implements a lock-wait timeout for deadlocks. The default value is 60 seconds. If the timeout occurs, the transaction is aborted.
- MemSQL queues modified rows for the garbage collector. This lets the garbage collector clean up old versions very efficiently by avoiding a full-table scan.
- Wherever possible, MemSQL can optimize single-row update queries down to simple atomic operations.

MemSQL Durability

- How it Works
  - MemSQL pushes transactions to disk as fast as the disk will allow.
  - Transactions first commit to an in memory buffer, and then asynchronously start writing to disk. If the size of the transaction buffer is zero, then the transaction is not acknowledged until it is committed to disk (full synchronous durability).
  - A log flusher thread flushes the transactions to disk every few milliseconds. MemSQL leverages group commit to dramatically improve disk throughput.
  - Once MemSQL log files reach a certain size (2 GB by default), they are compressed into snapshots. Snapshots are more compact and faster for recovery. This number is configurable as snapshot-trigger-size.
  - When MemSQL is restarted, it recovers its state by reading the snapshot file (a mullti-threaded process) and then replaying the remaining log file to restore its state. Snapshot recovery is significantly faster than log recovery.
  - Durability can be fully disabled for workloads that do not require it. Unless the transaction buffer fills up, this has no impact on query throughput or performance. It does not improve read performance.
  - MemSQL uses checksums in its snapshot and log files to validate data consistency.

- Why it's Fast
  - Group commit makes MemSQL faster in highly concurrent use cases (many writers pushing a lot of data into the log). This scenario is common to customers seeking a high throughput database.
  - The on-disk backup of MemSQL is extremely compact, which

reduces I/O pressure.
- Without a buffer pool, writes to disk are limited to sequentially writing logs and snapshots, so MemSQL is able to efficiently take advantage of sequential I/O.
- MemSQL writes both its snapshot and log files to disk sequentially. Both hard disk and solid state drives offer significantly better performance for sequential I/O than they do for random I/O.
- MemSQL completely avoids page-swapping and can guarantee consistent high-throughput SLAs on read/write queries. No read query in MemSQL will ever wait for disk. This property is extremely important for real-time analytics scanning over terabytes of data.

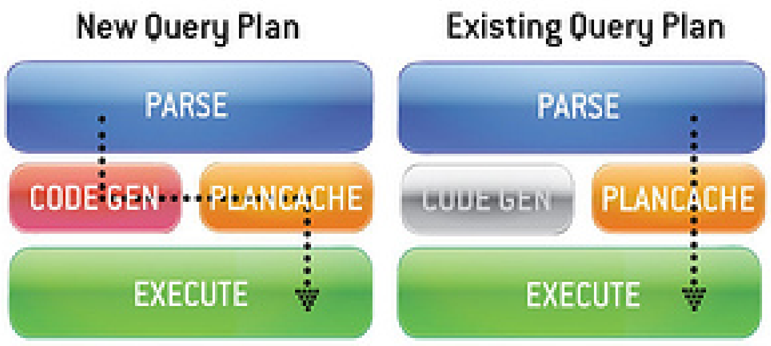
Code Generation

- How it Works
  - MemSQL compiles SQL queries to native code with SQL to C ++ code generation. C++ is compiled with GCC and loaded into the database as a shared object.
  - Compilation happens at run time. MemSQL is a just-in-time compiler for SQL. Compiled query plans are reused across server restarts so they only have to be compiled once in the lifetime of an application.
  - MemSQL uses a two-phase parser. The first parser is a one-pass lightweight layer called the auto-parameterizer, which strips numbers and strings out of plans.
  - For example "SELECT * FROM t WHERE id > 5 and name='John';" is converted to "SELECT * FROM t WHERE id > @ and name = ^;". These plans are stored in a hash table that maps parameterized queries to compiled query plans.
    - If the hash table contains the plan, then the parameters

are passed into the compiled code which executes the query.

- Otherwise, the query is processed by a traditional tree-based SQL parser and compiled into C++ code. The next time a query with the same shape is run, it will match the compiled plan in the hash table.

- DDL queries (CREATE/ALTER) and DML queries (SELECT/INSERT/UPDATE/DELETE) all go through code generation.
- Compiled C++ code is stored in the /plancache directory. Feel free to dive in and take a look.
- Does not support "if statements" or any procedural type logic. It's SQL and only SQL (for now). Though they contend using the SQL base for creating generated code yields optimum performance because it removes as much interpretation as possible. Machine code generated from SQL is hardwired into the code path.
- Dynamic SQL is supported. The code generation is handled in the background by calling gcc, which is not unusual for for system that combine DSLs with dynamic linking.



- Why it's Fast
  - The resulting speedup is comparable to upgrading from an interpreted language (PHP) to a compiled language (C++). This is the premise behind HipHop's PHP -> C++ compilation used at Facebook.

- ❍ The hot code path for query plans that have been compiled by the system is optimized very carefully. The hash table used to manage the plan cache is lock-free. Read queries smaller than 4kb use either pre-allocated or stack-allocated memory. Write queries allocate table memory via header-inlined slab allocators.
- ❍ malloc() is never called. Memory is allocated on process creation and managed internally from the on. This allows for complete control of garbage collection.
- ❍ Index data structures are defined in a per-table header file, which is included in every query on that table. This enables all storage engine operations to be inlined directly into the code and avoids the overhead of expensive per-row virtual function calls.
- ❍ Memory based systems can be slow depending on their design. Taking a global write lock or using memory mapped files is slow. A granular lock is taken only on a write-write conflict.
- ❍ Query compilation latency. A fresh installation of MemSQL comes with an empty plancache. Every new query shape not in plancache will be compiled with GCC. GCC compilation is still a little bit slow compared to query compilation in MySQL. Compilation takes 0.5 to 10 seconds per query per thread (depending on hardware).

Replication

- MemSQL replication is row-based, and supports master/multi-slave configuration.
- Supports K-Safety. As many servers as required can be used for High Availability.
- MemSQL supports online provisioning. Provisioning works by

shipping and recovering from a snapshot, and then continuing to replay from the log. This process fits naturally into MemSQL’s durability scheme and is what enables online provisioning.

- A slave will never encounter conflicts because order of execution is serialized in the transaction log.
- MemSQL does not support master-master replication. A master-master design has a negative trade off:loss of data consistency.
- MemSQL also supports synchronous replication. The tradeoff is higher write-query latency. Does not slow down reads.
- Load is balanced across slaves.
- When using asynchronous replication slaves are a few milliseconds behind.

The Distributed Story

- Note, this feature is not released yet, it’s set to be released in late September.
- Query Routing
    - Two-tier architecture with aggregators and leaves. Leaf nodes run MemSQL and store data. Aggregators have a query planner that receives queries, breaks them into smaller queries, and routes them to one or more leaf nodes. They aggregate the results intelligently back to the user.
    - Leaf nodes have a shared-nothing design. Data sharding across leaf nodes is managed by the aggregators.
    - MemSQL uses standard SQL partitioning syntax to implement range and hash (key) partitioning. Range partitioning involves defining every range of data to split against, while hash partitioning takes only a shard key to hash against. MemSQL uses consistent hashing to minimize data movement in the event of a failure.
    - In the event of a network partition, an aggregator

communicates with the metadata node to either (a) resync changed metadata or (b) assume responsibility for remapping data ranges and update it. During this time, queries are blocked up to a tolerance timeout.

- MemSQL supports single-shard OLTP queries (routing) and complex cross-shard OLAP queries that iterate over the entire dataset.
- Because the aggregators push most of the work to the leaf nodes, MemSQL scales almost linearly with each added leaf node.
- The aggregator also uses code generation to compile the logic. For simple queries the overhead is less than .5 milliseconds.
- Leaf nodes are dumb, they do not know about other leaf nodes. MVCC only works on each leaf node, not across leaf nodes.
- There are no cross shard transactions. Updates happen independently on each shard. Two phase commit is a future feature.
- Optimize for throughput and scalability.
    - By using a shared nothing architecture and by not using a two phase commit means a very high number of nodes can be supported with this approach.
    - Use case for this is integration with Vertica, which requires a long load step, but can store data larger than memory. MemSQL lets you see what is happening with a system right now.
    - Vertica is a column store which can do fast aggregations but can’t do fast updates, which is what MemSQL is optimized for.
    - Example use case is a financial system where sharding by stocks makes sense and stats are calculated by stock.
    - JDBC/ODBC can be used to sync to 3rd party systems.

- Clustering
  - Analytical queries can be run that touch every single node in order to produce aggregations.
  - Clustering is managed by aggregator nodes communicating with an external metadata service. The metadata service can be backed by MemSQL, MySQL, or ZooKeeper.
  - Each aggregator syncs and updates against the metadata service. The unified-metadata design was picked because it minimizes chatter in the system. If aggregators cannot contact the metadata node, they cannot perform DDL operations.
  - Short answer about CAP theorem: MemSQL can be configured for AP (availability, partition tolerance) or CP (consistency, partition tolerance). With asynchronous replication and load-balanced reads, MemSQL can even be configured for eventual consistency.
  - Long answer about the CAP theorem: As Eric Brewer (CAP inventor) points out, CAP theorem and the “pick 2 out of 3” mentality are too simplistic for analyzing a complex system (http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed). Instead, the discussion should be about how the system “maximizes combinations of consistency and availability that make sense for the specific application.”
  - Users can configure replication to match the requirements of their application:
    - Replication enables high availability in the system. In the event of a network partition, MemSQL will immediately elect a new replication master to keep the system available.
    - Asynchronous replication minimizes query latency for write queries. In the event of a failure, failover to replication slave and suffer partial data loss for the

replication delta. MemSQL replication is powerful enough to detect and in some cases resolve divergences when the partitioned leaf node is restarted and restored to the system.

- Synchronous replication maintains strong consistency. It enables the system to be configured for k-safety (the system remains consistent and available in the presence of k failures). This option results in higher write query latencies.

Deployment and Management

- Easy new query deployment.
    - Deploying new queries doesn’t require wrapping code in Java, compiling the Java, stopping the server and then deploying.
    - MySQL clients send SQL statements to the server. Nothing extra is needed.
    - Implication is simplicity is from SQL, adding any third party code won’t work.
- How get up and running with MemSQL:
    - Download of developer edition is available directly on the website. http://www.memsql.com/#download
    - Streamlined deployment on EC2. Pre-configured AMI that accepts your license key and launches MemSQL for you on Ubuntu 12.04. 20% of production MemSQL deployments are run this way.
    - Can also run on a variety of Linux 64-bit systems by downloading a tar.gz file from download.memsql.com.
- Client Libraries
    - The MySQL protocol is used instead of making their client protocol. This makes it easy to integrate into existing environments.

- ❍ Very smart use of the extensive MySQL ecosystem by leveraging high performance MySQL clients instead of building their own.
- ❍ Just change your port and point to MemSQL instead of MySQL. Allows the focus to be on server development instead of client development.
- ❍ MemSQL works with any MySQL client library: ODBC, JDBC, PHP/Python/Perl/Ruby/etc, mysql c library.
- ❍ Popular manageability tools (Sequel Pro, PHPMyAdmin, MySQL Workbench), app frameworks (Ruby on Rails, Django, Symfony), and visualization tools (panopticon) work with MemSQL.

- Server Management
  - ❍ MemSQL controls out of memory with two knobs: maximum_memory and maximum_table_memory. maximum_memory limits the total memory use by the server and maximum_table_memory limits the amount used for table storage. If memory usage exceeds maximum_table_memory then write queries are blocked but read queries stay up. On the developer build maximum_table_memory is hardcoded to 10 GB.
  - ❍ MemSQL exposes custom statistics with the SHOW STATUS, SHOW STATUS EXTENDED, and SHOW PLANCACHE commands. You can get numbers on total query compilations, query execution performance, and durability performance.

SQL Support

- Very limited SQL support. Just joins between two tables. No outer or full outer joins.
- Does not support: views, prepared queries, stored procedures, user

defined functions, triggers, foreign keys, charsets other than utf8

- The only supported isolation level: READ COMMITTED
- MemSQL only supports single query transactions. Every query begins and commits in its own transaction.
- SQL is used:
    - To reduce training costs, people (traders, business types) know SQL.
    - Because they wanted to build something that was easy to use.

Pricing

- Pricing isn't being disclosed just yet. The thought is use an hourly model so more developers can deploy on the cloud today.

Lessons Learned

- Y Combinator isn't just about funding, it's a support network that helps you make much needed connections, like getting early customer wins.
- Use C++ for systems-level infrastructure. It allows you to build more efficient and robust software
- You should use established interfaces to drive adoption. Make it extremely easy for people to try your software
- Hire people who are "ahead of the curve" in their careers and promote from within
- Invest in a good code review system; we use Phabricator (Facebook's code review system, now at phabricator.org)
- Make it easy to add tests to the system and invest in hardware and software to make testing easy. Software will only become reliable from extensive testing. If you're testing 24/7, invest in your own hardware.

Related Articles

- Paper: High-Performance Concurrency Control Mechanisms for Main-Memory Databases
- Paper: A Provably Correct Scalable Concurrent Skip List
- Domas Mituzas: MySQL is bazillion times faster than MemSQL
- Reading Between the Benchmarks: How MemSQL Designs for Speed and Durability
- Hacker News: Ex-Facebookers launch MemSQL (YC W11) to make your database fly
- Hacker News: MySQL is bazillion times faster than MemSQL
- Slashdot: MemSQL Makers Say They've Created the Fastest Database On the Planet
- Quora: SQL: What benefits does MemSQL offer over running a MySQL database on ramdisk?
- Robert Scoble: The @memsql guys answer Twitter questions (much faster than MySQL). at Rackspace SF
- MemSQL Blog
- MemSQL Faq
- MemSQL Twitter
- Kurt Monash: Introduction to MemSQL
- Hacker News Comment: MemSQL vs VoltDB
- Bloomberg: Enterprise Technology: Revenge of the Nerdiest Nerds
- VoltDB Decapitates Six SQL Urban Myths And Delivers Internet Scale OLTP In The Process
- Paper: Distributed Transactions for Google App Engine: Optimistic Distributed Transactions built upon Local Multi-Version Concurrency Control
- Paper: Concurrent Programming Without Locks
- Herb Sutter: Choose Concurrency-Friendly Data Structures