Bottleneck Hunters: How Schooner increased MySQL throughput by more than 800%

Jeremy Cole <jeremy.cole@schoonerinfotech.com>
• On the genesis of Schooner:
  • Hardware is massively under-utilized
  • I/O has long been a bottleneck
  • Commoditized flash prices are coming down fast
  • MySQL can’t make good use of Flash or CPU cores

• All of this means:
  • Too much hardware gets bought for sharding applications which make poor use of the hardware
  • Too many human and environmental resources are wasted
  • Too much power and ongoing cost is wasted
  • It may not even perform well enough anyway!
• Some basics of performance modeling and tuning

• Tools and techniques

• Impact of optimization and balanced systems on performance of MySQL
Crash course in scaling

• “What is saturated?”

• There are 5 major performance-impacting factors:
  • I/O throughput, latency
  • Memory capacity and throughput
  • CPU speed and number of cores
  • Network (1Gbps vs. 10Gbps, event handling)
  • Software (parallelism, locking, mutexes)
Performance reference points

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>CPU Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>CPU L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek (7200 rpm)</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet US-&gt;NL-&gt;US</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>

### Traditional disk-based approaches

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>CPU Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>CPU L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek (7200 rpm)</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet US-&gt;NL-&gt;US</td>
<td>150,000,000</td>
</tr>
<tr>
<td>Operation</td>
<td>Time (ns)</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>CPU L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>CPU Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>CPU L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek (15000 rpm)</td>
<td>4,000,000</td>
</tr>
<tr>
<td>Disk seek (7200 rpm)</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet US–&gt;NL–&gt;US</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>
### Performance with solid state media

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>CPU Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>CPU L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read from solid state media (SSD)</td>
<td>70,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek (15000 rpm)</td>
<td>4,000,000</td>
</tr>
<tr>
<td>Disk seek (7200 rpm)</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet US-&gt;NL-&gt;US</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>
Performance expectations from SSDs

- Using SSDs will normally double MySQL performance
- But: SSDs are 10,000,000 / 70,000 = ~142x faster!
- If I/O is the bottleneck, why aren’t we seeing a 142x increase when going to SSDs?
Finding bottlenecks
Do you have an I/O bottleneck?

- Be careful: iostat utilization is wrong
  - `# iostat -x 2`

- Utilization (in red) expects single CPU (broken SMP)
- IO Wait (in yellow) is a good indication of IO saturation
Software bottleneck

- 15.7% CPU Utilization (us)
- 4.2% Waiting on IO (wa)
- 72.0% Idle (id)
Memory capacity

- Tons of memory free
- Use memory as a substitute for software algorithmic problems
• Build or find benchmarks/conditions to ensure repeatable workload which demonstrates bottlenecks
• Capture system and software metrics during each run
• Produce graphs of all metrics – look for correlation
• Build micro-benchmarks to test theories – eliminate layers of complexity which can confuse
• Remove code paths to test theories (simplify the problem and/or remove variables)
• Don’t change more than one thing at a time
• Results must be reproducible – don’t chase flukes
Tools for finding bottlenecks

- Use oprofile to find hot/busy functions
- Use new instrumentation to find overused locks
- Use PMP to find very busy common code paths
- Solaris has dtrace framework
- Linux has SystemTap
Real-world load – capture

• Testing with Real Workload is better than any synthetic benchmark

• Take a database snapshot
  • XtraBackup or filesystem snapshot (SAN, LVM, or cold copy)

• Get queries via tcpdump
  • # tcpdump -s 65535 -v -x -n -q -tttt -i ethX port 3306 > tcp.log

• Put data into “slow.log” format
  • # mk-query-digest --type tcpdump --no-report --print tcp.log > slow.log
• Playback on development system
  • `# mk-log-player --split Thread_id --session-files 16 --base-dir ./sessions slow.log`
  • `# mk-log-player --play ./sessions --base-dir ./results h=host1`

• Scale up number of clients until saturation
• Saturation is noticed when all cores are busy and adding more clients doesn’t increase throughput
Benchmarking MySQL on Solid State Storage
Benchmarking configuration

Hardware

2 Quad Core 5560 Xeons (with 2 HT per core, 16 CPUs via Linux)
8 SSDs
64GB Memory

DBT2 benchmark – osdldbt.sf.net

1000 warehouses – 100G data
32 connections
Zero think time

MySQL Configuration

innodb_buffer_pool_size = 48G
innodb_flush_log_at_trx_commit = 1
About the DBT2 benchmark

- Popular open-source database benchmark inspired by TPCC™
  - Focuses on OLTP (online-transaction processing)
  - Scales with data-size, includes ramp-up and steady state
  - High write rate, requires good locality in buffer pool
- Transactions with Select, Update, Insert, Delete
- Throughput metric: TPM (New Order)
  - Transaction Ratios: New Order 45% (with 1% rollback), Payment 43%, Stock Level 4%, Order Status 4%, Delivery 4%
- Results include TPM, response time (avg and 90th %ile), CPU, iostat, etc.
Stock MySQL 5.1.44 on SSDs

CPU Utilization

13.4k TPM
Stock MySQL 5.5.4m3 on SSDs

CPU Utilization

23.6k TPM
Stock MySQL 5.5.4m3 on Fusion-io

CPU Utilization

49.3k TPM
Schooner (5.1)

CPU Utilization

118.1k TPM
Benchmarking results summary

- Stock MySQL 5.1.44
- Stock MySQL 5.5.4m3
- Stock MySQL 5.5.4m3 Fusion-io
- Schooner (5.1)
The Schooner Approach
Schooner R&D efforts

- Workload characterization
- Holistic view on architecture
- Modeling performance, reliability, power, and cost
- Researching new hardware, technologies, and trends
- Prototyping and verification
- Product development
Schooner benefits

- Three calendar years of effort
- Changes to algorithms, locks, and core data structures require careful planning, research, and extensive testing
- Stringent qualification process for release
  - Functional regression
  - Performance regression
  - Longevity (no memory leaks, etc.)
  - Durability (transactional model is correct)
- Working with many different workloads from customers, benchmarks, and micro-benchmarks
Removed bottlenecks: balanced system

- **Hardware**
  - Flash
  - Memory
  - Number of cores

- **Software**
  - Increase parallelism
  - Granular concurrency control
  - CPU path lengths
  - Resource management algorithms
A balanced system costs less

Relative 3-Year TCO
Thank you!

Schooner Information Technology
www.schoonerinfotech.com
www.schoonerinfotech.com/benchmarks

Jeremy Cole
jeremy.cole@schoonerinfotech.com