



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

- “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

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

Source: <http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf>

Traditional disk-based approaches

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

Traditional disk-based approaches

CPU L1 cache reference	0.5 ns
CPU Branch mispredict	5 ns
CPU L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek (15000 rpm)	4,000,000 ns
Disk seek (7200 rpm)	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet US->NL->US	150,000,000 ns

Performance with solid state media

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

- Using SSDs will normally double MySQL performance
- But: SSDs are $10,000,000 / 70,000 = \sim 142x$ 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

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.44    0.00    8.06   85.35    0.00    6.15

Device:            rrq      wrq     rrqm/s   wrqm/s      rd
sda                367630  414554940  0.00     0.00     530612
sdb                41846985  58760824  0.00    27.00    30269547
sdc                333637027  397925918  1.00     1.50    132485303
sdd                334204962  398662881 22.50    19.50    133167134
sde                334064095  398585248  5.00     4.00    132776722
sdf                333495879  398311036  1.00     0.50    132487925
sdg                333776443  398597479  1.00     1.00    133031410
sdh                333698665  398531550  0.00     0.00    132968138
sdi                333704247  398436367 30.00    22.00    133157011
sdj                333514115  398383453  0.50     1.00    133288634
md_d0              0          0         0.00     0.00    110159228
```

```
wr_ticks avgqu-sz  await  svctm  %util
2209827108  15.38  0.63  0.04  100.00
169669443  15.08  0.60  0.04  100.00
185045913  1.17  0.17  0.07  51.90
185806885  1.37  0.21  0.08  54.10
185421417  4.14  0.70  0.14  79.85
183294210  1.82  0.26  0.09  59.55
183637552  1.23  0.18  0.07  49.90
185028090  0.49  0.12  0.07  27.75
186309938  2.07  0.51  0.18  73.75
185681887  0.94  0.16  0.08  43.90
0          0.00  0.00  0.00  0.00
```

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

Software bottleneck

```
top - 14:40:28 up 30 days, 5:14, 3 users, load average: 3.49, 2.34, 5.23
Tasks: 318 total, 1 running, 317 sleeping, 0 stopped, 0 zombie
Cpu(s): 15.7%us, 5.3%sy, 0.0%ni, 72.0%id, 4.2%wa, 0.2%hi, 2.5%si, 0.0%st
Mem: 65981528k total, 62186720k used, 3794808k free, 445192k buffers
Swap: 32764528k total, 12152k used, 32752376k free, 54503672k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
17937	mysql	15	0	7833m	5.1g	5288	S	350.7	8.1	3:52.62	mysqld
7410	root	10	-5	0	0	0	S	3.7	0.0	569:27.50	md_d0_raid5
5902	root	19	0	776m	96m	9820	S	0.3	0.1	139:32.95	java
9606	root	10	-5	0	0	0	S	0.3	0.0	7:35.91	kjournald
31338	root	34	19	0	0	0	S	0.3	0.0	61:22.63	kipmi0
1	root	15	0	10348	568	536	S	0.0	0.0	0:07.77	init
2	root	RT	-5	0	0	0	S	0.0	0.0	0:05.73	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.28	ksoftirqd/0
4	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
5	root	RT	-5	0	0	0	S	0.0	0.0	0:01.43	migration/1
6	root	34	19	0	0	0	S	0.0	0.0	0:01.22	ksoftirqd/1
7	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	watchdog/1
8	root	RT	-5	0	0	0	S	0.0	0.0	0:01.68	migration/2
9	root	34	19	0	0	0	S	0.0	0.0	0:03.46	ksoftirqd/2
10	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	watchdog/2
11	root	RT	-5	0	0	0	S	0.0	0.0	0:01.39	migration/3
12	root	34	19	0	0	0	S	0.0	0.0	0:15.72	ksoftirqd/3

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

Memory capacity

```
top - 14:40:28 up 30 days, 5:14, 3 users, load average: 3.49, 2.34, 5.23
Tasks: 318 total, 1 running, 317 sleeping, 0 stopped, 0 zombie
Cpu(s): 15.7%us, 5.3%sy, 0.0%ni, 72.0%id, 4.2%wa, 0.2%hi, 2.5%si, 0.0%st
Mem: 65981528k total, 62186720k used, 3794808k free, 445192k buffers
Swap: 32764528k total, 12152k used, 32752376k free, 54503672k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
17937	mysql	15	0	7833m	5.1g	5288	S	350.7	8.1	3:52.62	mysqld
7410	root	10	-5	0	0	0	S	3.7	0.0	569:27.50	md_d0_raid5
5902	root	19	0	776m	96m	9820	S	0.3	0.1	139:32.95	java
9606	root	10	-5	0	0	0	S	0.3	0.0	7:35.91	kjournald
31338	root	34	19	0	0	0	S	0.3	0.0	61:22.63	kipmi0
1	root	15	0	10348	568	536	S	0.0	0.0	0:07.77	init
2	root	RT	-5	0	0	0	S	0.0	0.0	0:05.73	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.28	ksoftirqd/0
4	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
5	root	RT	-5	0	0	0	S	0.0	0.0	0:01.43	migration/1
6	root	34	19	0	0	0	S	0.0	0.0	0:01.22	ksoftirqd/1
7	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	watchdog/1
8	root	RT	-5	0	0	0	S	0.0	0.0	0:01.68	migration/2
9	root	34	19	0	0	0	S	0.0	0.0	0:03.46	ksoftirqd/2
10	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	watchdog/2
11	root	RT	-5	0	0	0	S	0.0	0.0	0:01.39	migration/3
12	root	34	19	0	0	0	S	0.0	0.0	0:15.72	ksoftirqd/3

- 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

- 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

- 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

Hardware

2 Quad Core 5560 Xeons (with 2 HT per core, 16 CPUs via Linux)

8 SSDs

64GB Memory

DBT2 benchmark – osldbt.sf.net

1000 warehouses – 100G data

32 connections

Zero think time

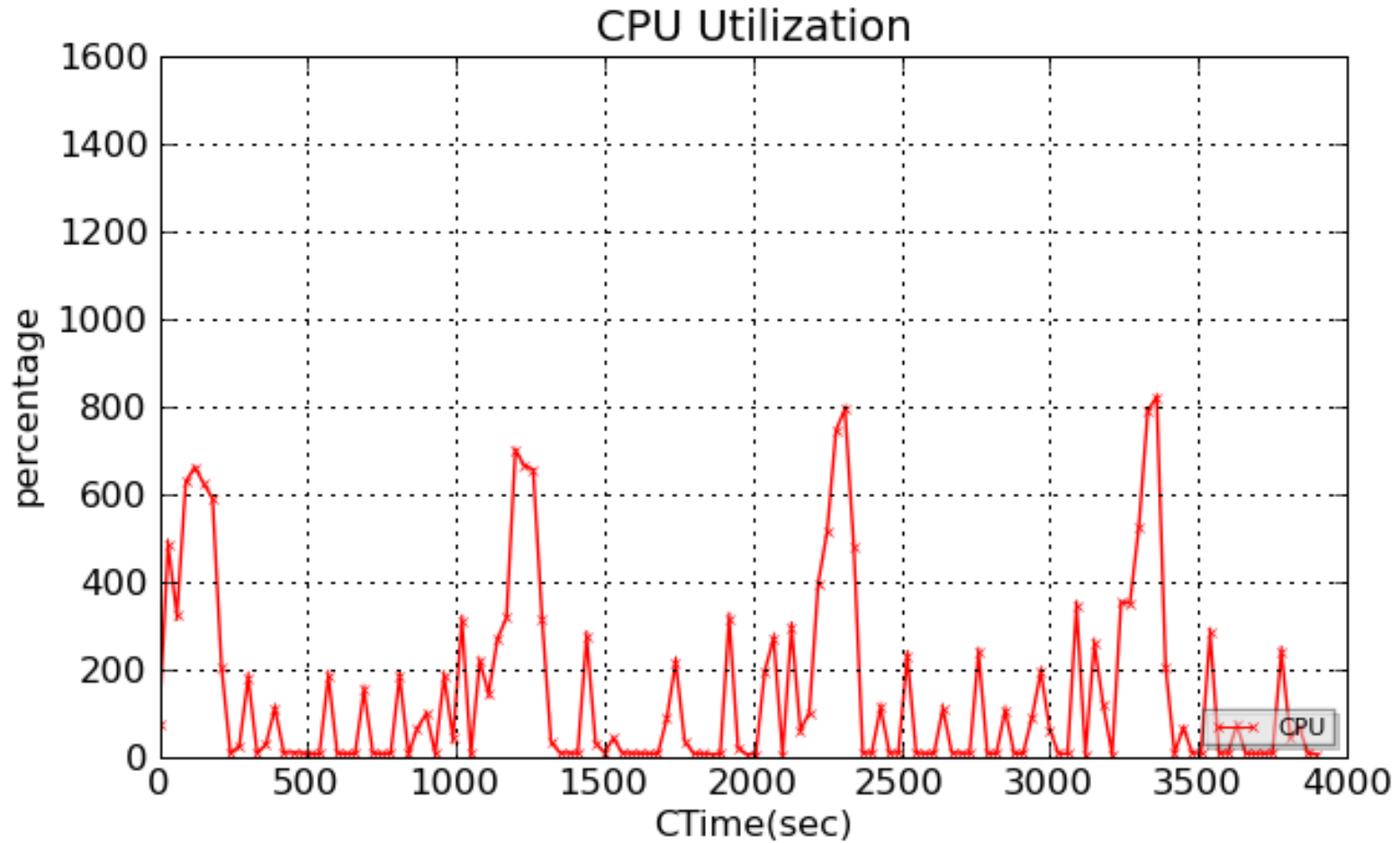
MySQL Configuration

```
innodb_buffer_pool_size          = 48G
```

```
innodb_flush_log_at_trx_commit = 1
```

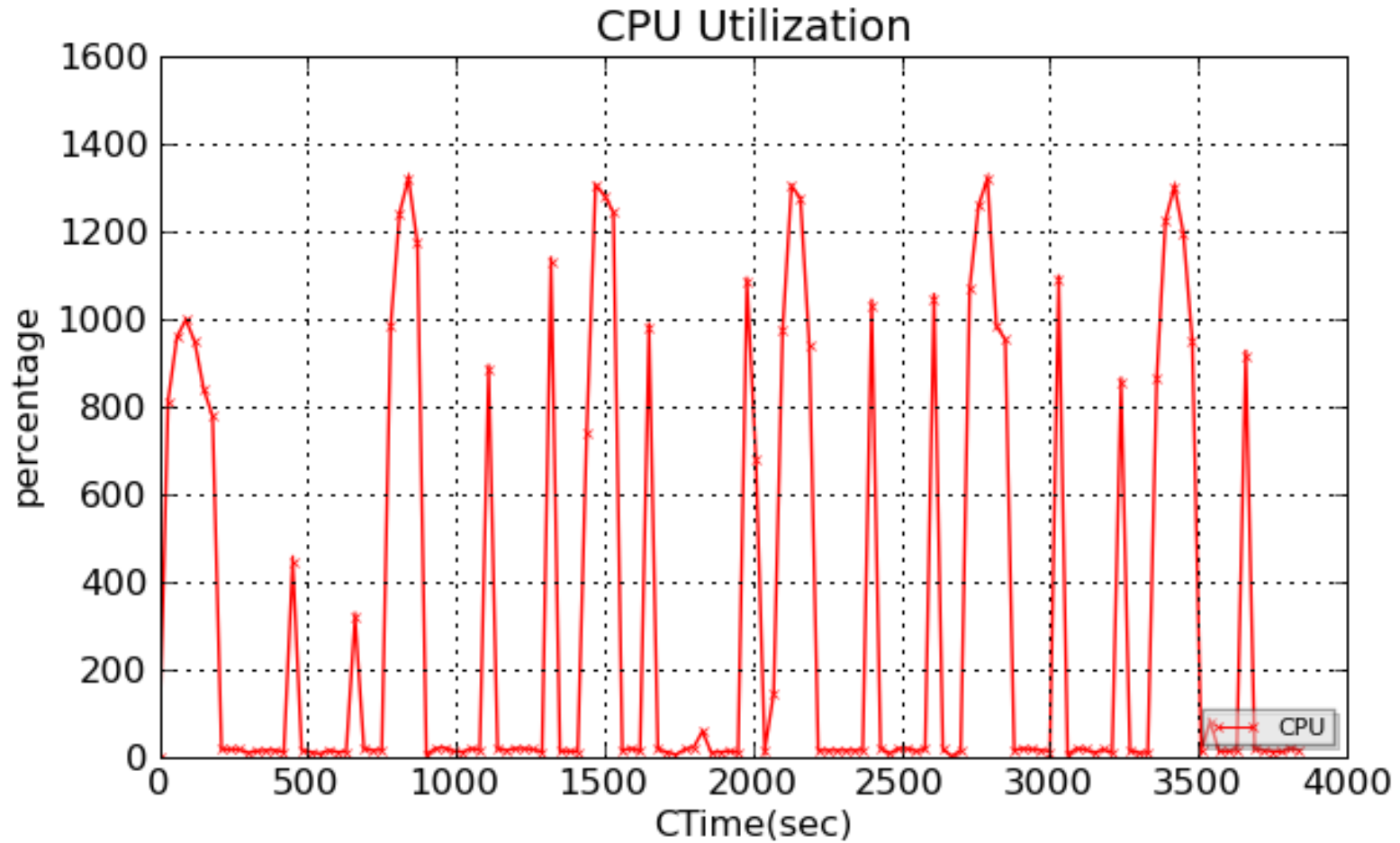
- Popular open-source database benchmark inspired by TPCC™
 - Focuses on OLTP (online-transaction processing)
 - Scales with data-size, includes ramp-up and steady state
 - High write rate, requires good locality in buffer pool
- Transactions with Select, Update, Insert, Delete
- Throughput metric: TPM (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



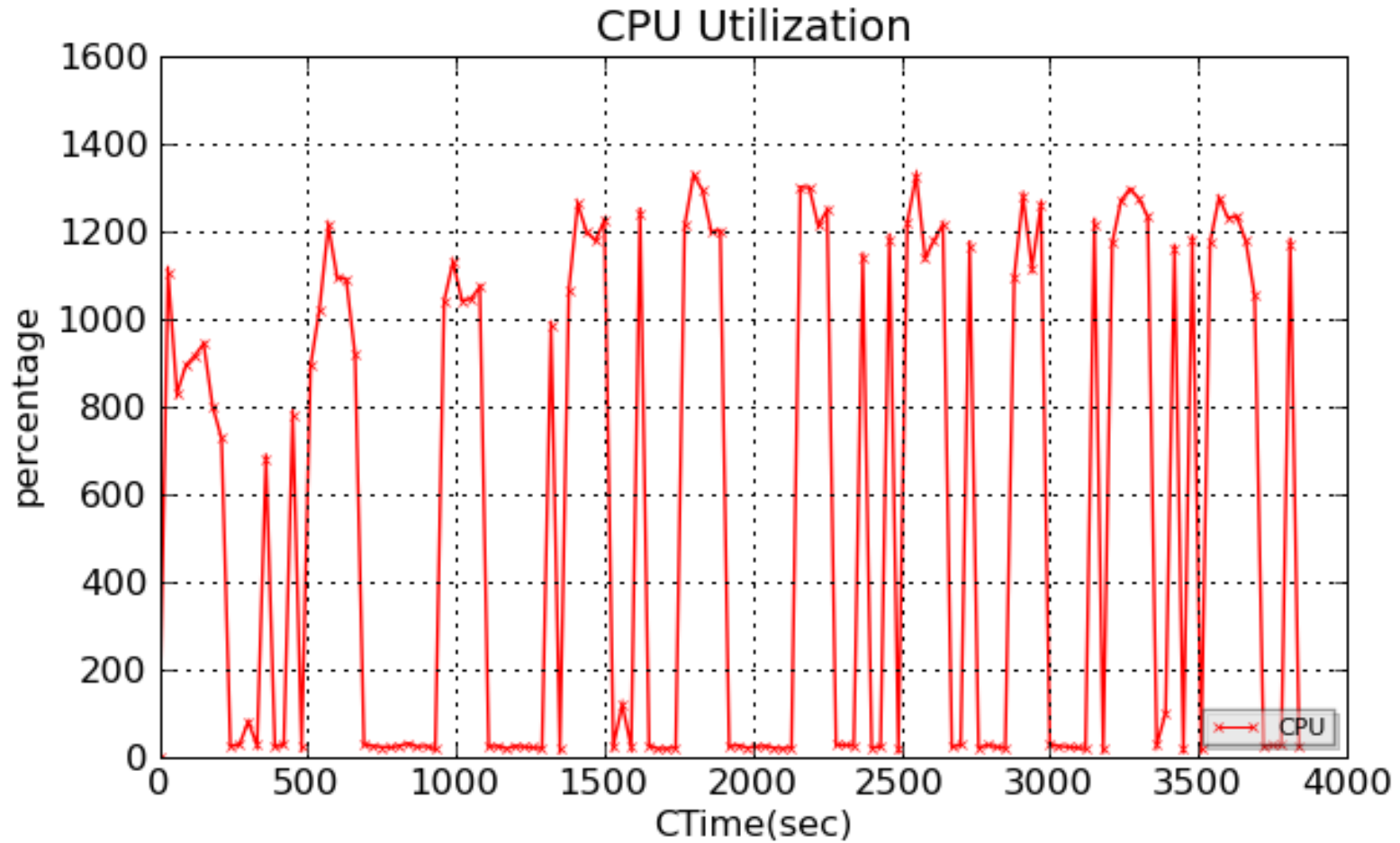
13.4k TPM

Stock MySQL 5.5.4m3 on SSDs



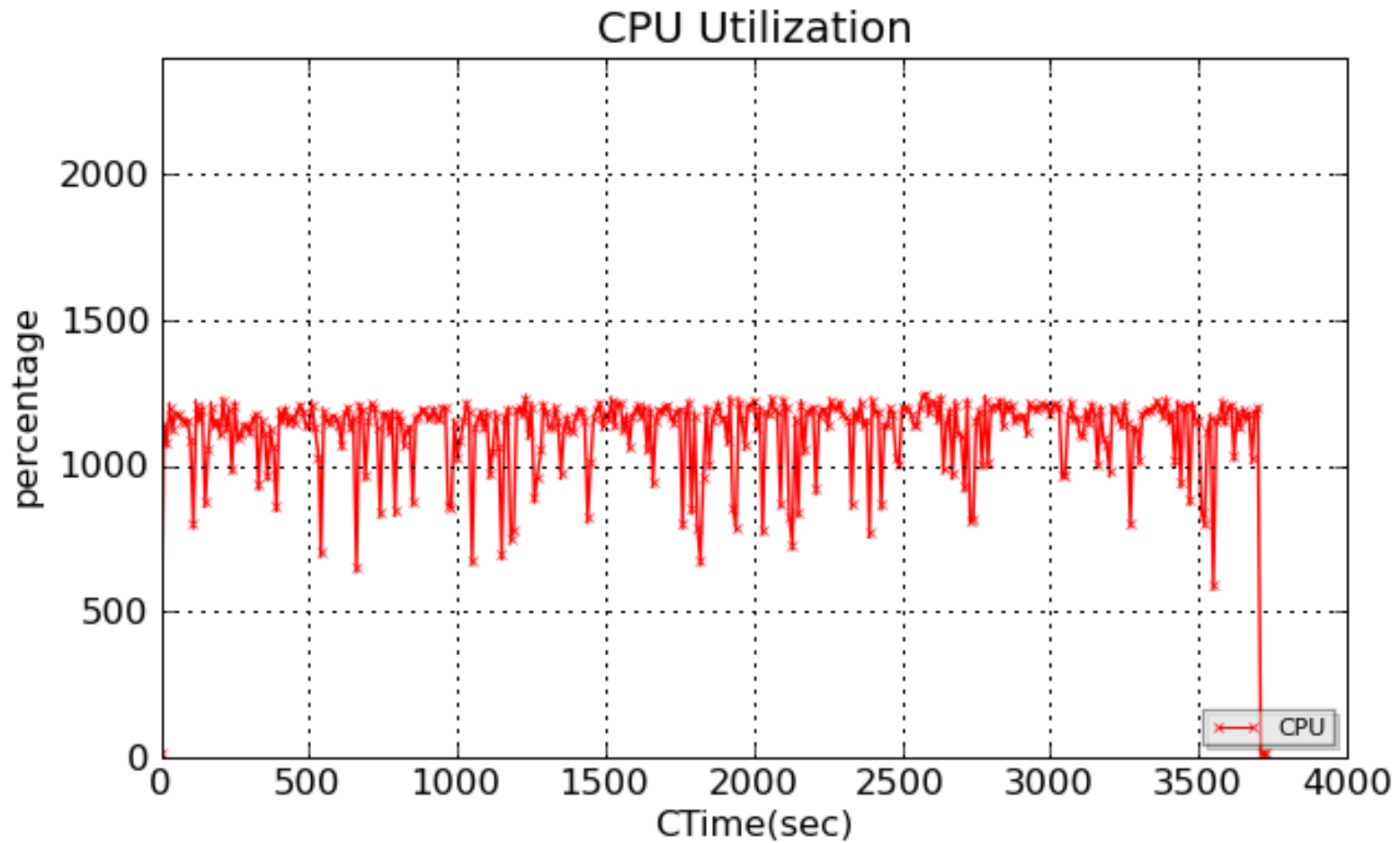
23.6k TPM

Stock MySQL 5.5.4m3 on Fusion-io



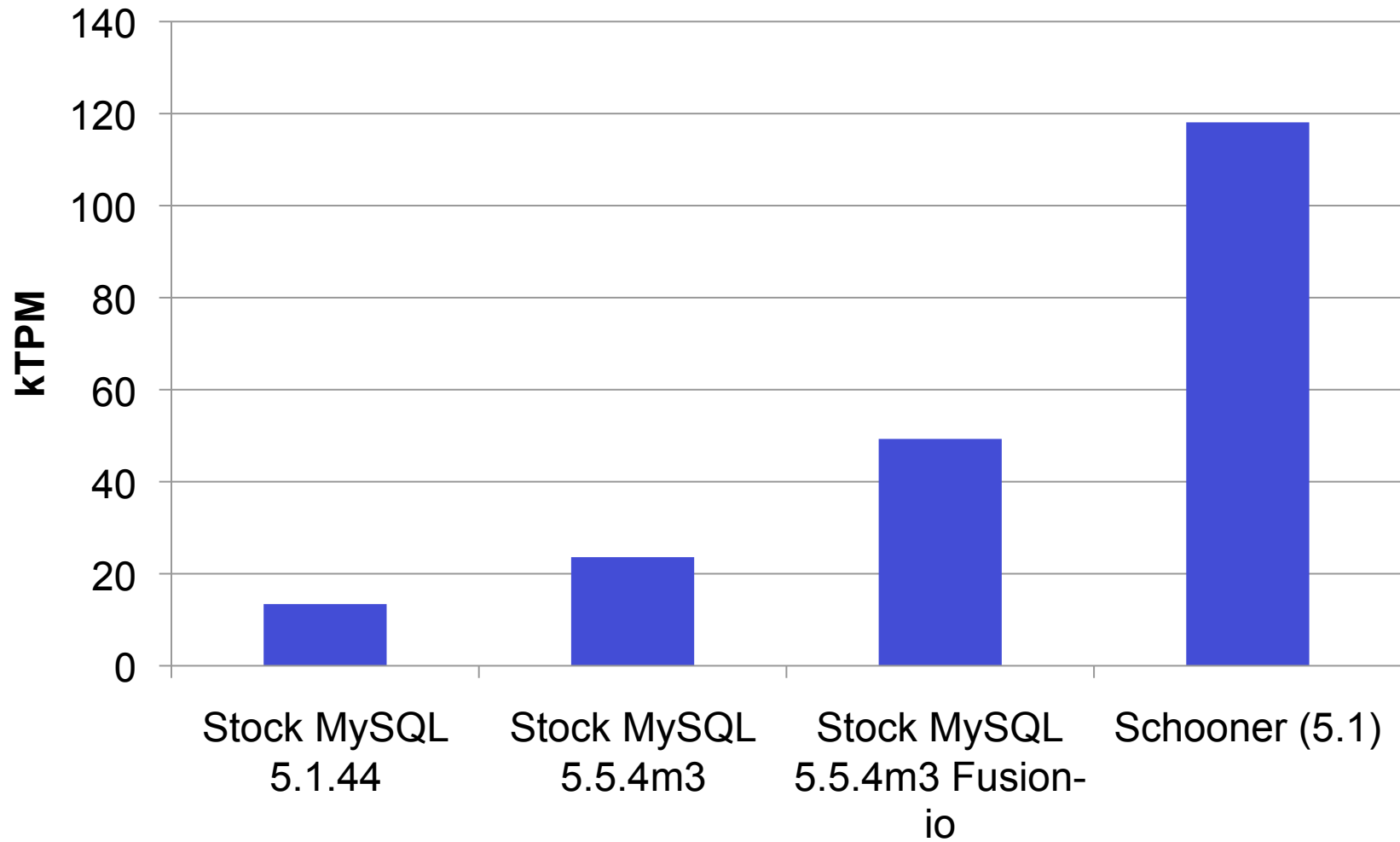
49.3k TPM

Schooner (5.1)



118.1k TPM

Benchmarking results summary



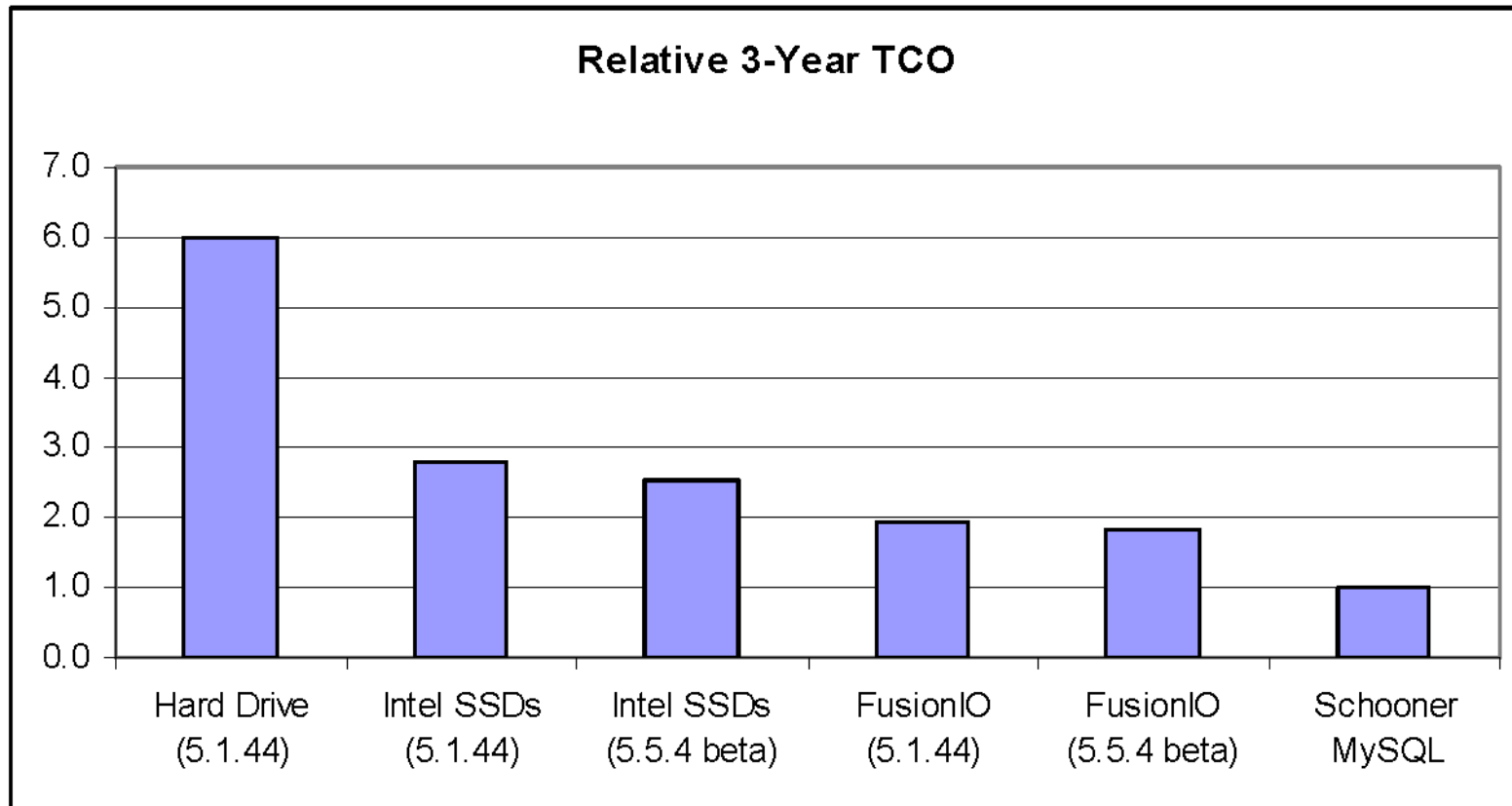
The Schooner Approach

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

- 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

- Hardware
 - Flash
 - Memory
 - Number of cores
- Software
 - Increase parallelism
 - Granular concurrency control
 - CPU path lengths
 - Resource management algorithms

A balanced system costs less



Thank you!

Schooner Information Technology

www.schoonerinfotech.com

www.schoonerinfotech.com/benchmarks

Jeremy Cole

jeremy.cole@schoonerinfotech.com