Breaking Down MySQL/Percona Query Latency
With DTrace

Brendan Gregg

Lead Performance Engineer, Joyent

Percona Live, May 2011

Joyent smart computing

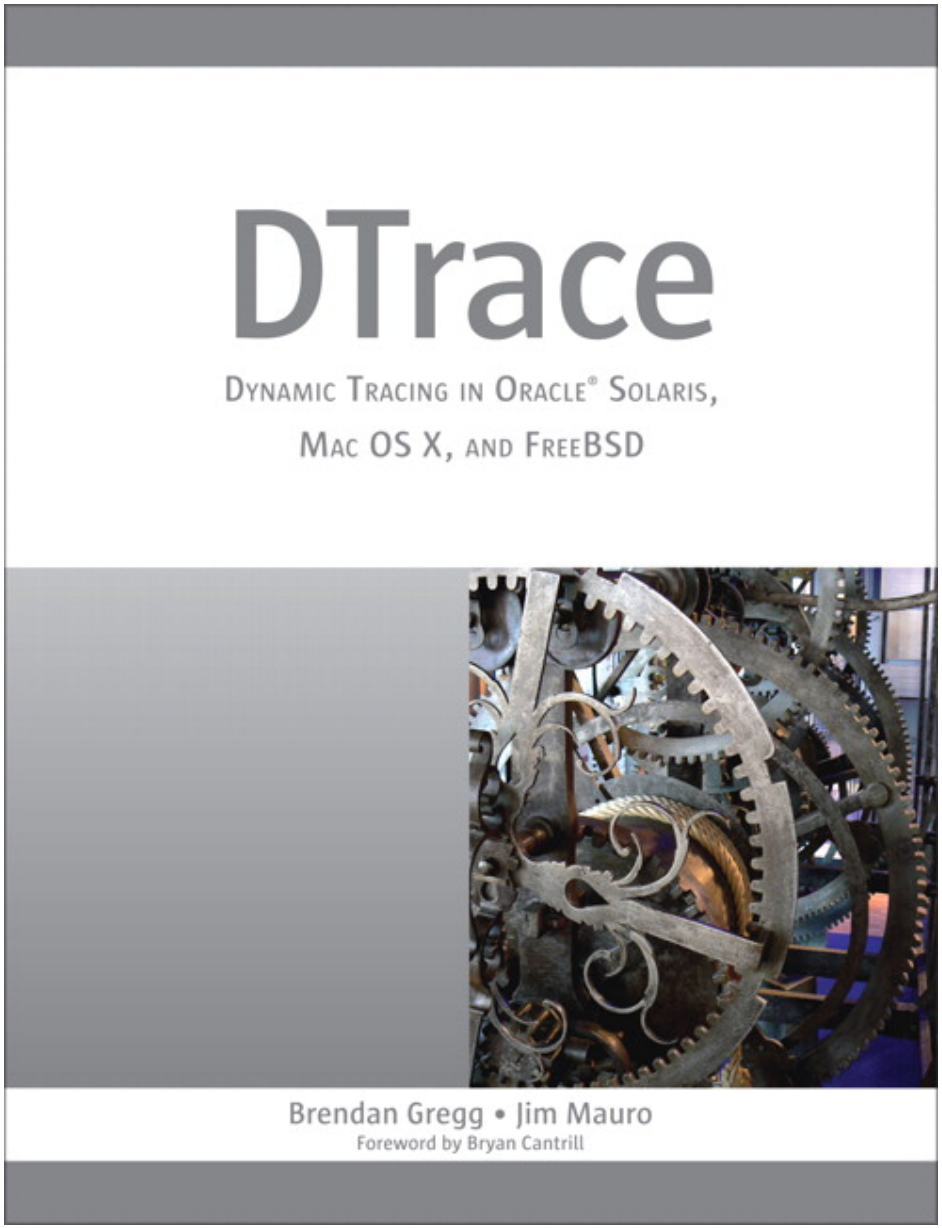
Agenda

- DTrace and Dynamic Tracing
- Latency and Query Latency
- Query Latency Components
  - on-CPU, off-CPU, File System, innodb thread concurrency
- Questions

Joyent smart computing

G’Day, I’m Brendan

- I do performance analysis
- I also write performance tools out of necessity


DTrace
Dynamic Tracing in Oracle® Solaris,
Mac OS X, and FreeBSD
Brendan Gregg • Jim Mauro
Foreword by Bryan Cantrill

Joyent smart computing

Joyent

- We do cloud computing
- Many customers and instances of MySQL
- Now offering SmartMachines with Percona!
- Performance and scalability is critical
- We use DTrace

Joyent smart computing

DTrace

- A performance analysis and troubleshooting tool
- User- and kernel-land tracing
- Production safe; in use since 2005
- C & awk inspired language
- Currently for:
  - Solaris-based OSes (Solaris 10+, SmartOS, Illumos, etc.)
  - Mac OS X 10.5+
  - FreeBSD 7.1+
  - Linux? http://crtags.blogspot.com

Joyent smart computing

DTrace Users

- Command line end-users:
  - System Administrators
  - Database Administrators
  - Developers

Joyent smart computing

DTrace Users

- Command line end-users:
  - System Administrators → resource usage
  - Database Administrators → database internals
  - Developers → latency

Joyent smart computing

DTrace Users

- Command line end-users:
  - System Administrators → resource usage
  - Database Administrators → database internals
  - Developers → latency

  Either by writing scripts, or running other people's
  - Hire/assign a DTrace guy for the team

Joyent smart computing

DTrace Users

- Command line end-users:
  - System Administrators → resource usage
  - Database Administrators → database internals
  - Developers → latency

  Either by writing scripts, or running other people's
  - Hire/assign a DTrace guy for the team
- Indirect end-users:
  - Apple's Instruments
  - Joyent's Cloud Analytics
  - and more…

Joyent smart computing

DTrace is Different

- Not just-another performance tool
- 1st implementation of *dynamic tracing*
- Has static tracing as well

Joyent smart computing

Dynamic Tracing of MySQL

```
# dtrace -ln 'pid$target:::entry' -p 29008
   ID   PROVIDER        MODULE                          FUNCTION NAME
77532   pid29008        mysqld                             _start entry
77533   pid29008        mysqld                            _mcount entry
77534   pid29008        mysqld  _ZN18st_parsing_options5resetEv entry
[...]
82039   pid29008        mysqld                mysql_send_query entry
82040   pid29008        mysqld                mysql_real_query entry
82041   pid29008        mysqld             mysql_real_connect entry
82042   pid29008        mysqld             mysql_store_result entry
82043   pid29008        mysqld                mysql_fetch_row entry
[...]
83939   pid29008        mysqld                  os_file_write entry
83941   pid29008        mysqld                   os_file_read entry
[...]
97074   pid29008     libc.so.1                          strlen entry
97075   pid29008     libc.so.1                         strncmp entry
[...22163 lines truncated...]
```

- Plus entry arguments, return values, timestamps (ns)

Joyent smart computing

Static Tracing of MySQL

```
# dtrace -ln 'mysql*:::'
   ID   PROVIDER    MODULE                          FUNCTION NAME
12173   mysql268    mysqld           _Z10do_commandP3THD data-receive-finish
12174   mysql268    mysqld           _Z10do_commandP3THD data-receive-start
12175   mysql268    mysqld          log_reserve_and_open flush-log-finish
12176   mysql268    mysqld          log_reserve_and_open flush-log-start
12177   mysql268    mysqld _ZN11ha_innobase11index_firstEPc innodb-index-
first-finish
[...20 lines truncated...]
12222   mysql268    mysqld                 wait_for_lock myisam-wrlck-start
12223   mysql268    mysqld _ZN11Query_cache21send_result_to_clientEP3THDPcj
query-cache-hit
12224   mysql268    mysqld _ZN11Query_cache21send_result_to_clientEP3THDPcj
query-cache-miss
12225   mysql268    mysqld _Z21mysql_execute_commandP3THD query-execute-finish
12226   mysql268    mysqld _Z21mysql_execute_commandP3THD query-execute-start
12227   mysql268    mysqld       _Z11choose_planP4JOINy query-plan-finish
12228   mysql268    mysqld       _Z11choose_planP4JOINy query-plan-start
```

- Easy to use and a stable & documented interface

Joyent smart computing

Static vs Dynamic

- Static is better, use it when available
- Limited, like any traditional (pre-DTrace) metric
- Dynamic tracing is the catchall:
  - Will work on old and new MySQL versions
  - Can see everything
- I currently spend more time using dynamic over static tracing of MySQL
  - but this could change with more static probes

Joyent smart computing

Dynamic Tracing

- DTrace is an implementation of Dynamic Tracing
  - One that has proven the concept
  - Has been used in large scale production for 5+ years
- Dynamic Tracing provides a new perspective:
  - What metrics would I like?
  - Given the ability to observe everything, …

Joyent smart computing

Latency

- Time between events
  - typically measured as request to completion
- A primary measure of application pain
  - When measured as a synchronous component of the workload
- Easy to get with DTrace
  - Either event by event, or summaries

Joyent smart computing

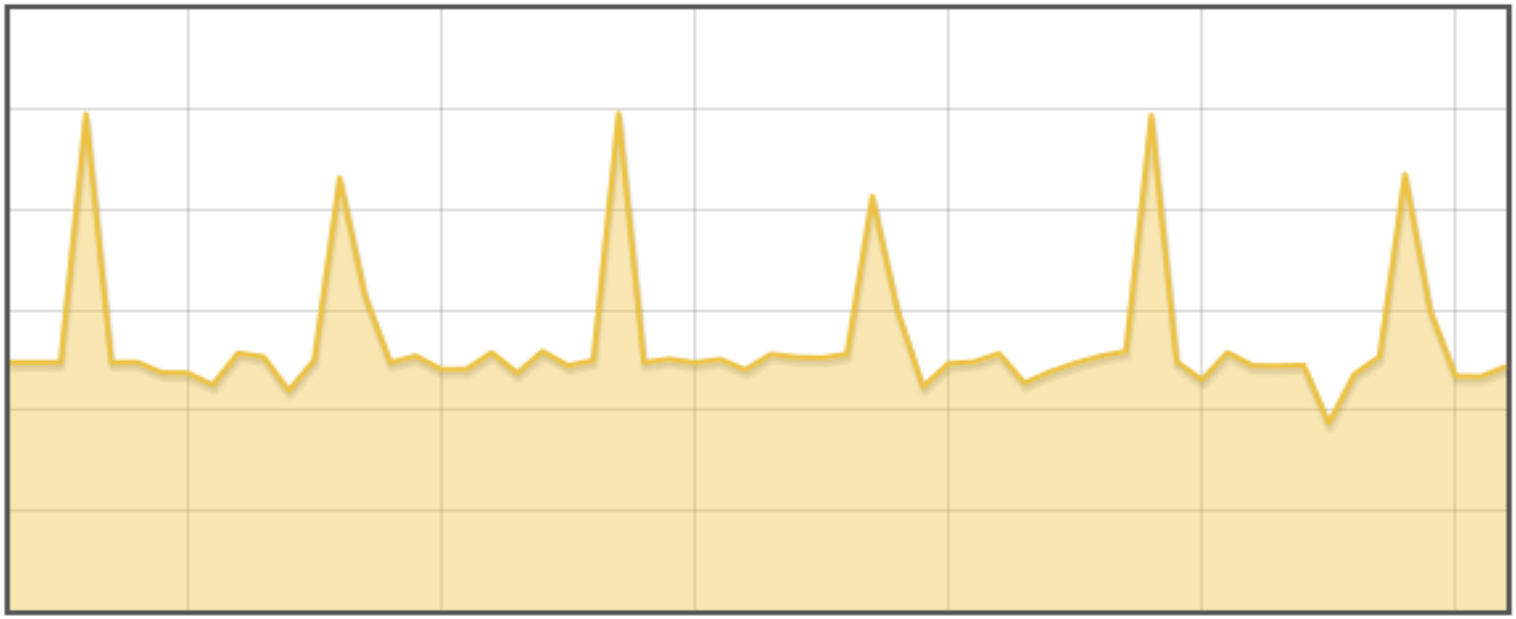
Latency: Event-by-event

```
# ./iosnoop –Dots
STIME        TIME         DELTA  DTIME    UID   PID D    BLOCK   SIZE     COMM PATHNAME
949417936651 949417948636 11984  11999    104 29008 R 99310470  16384   mysqld <none>
949418267667 949418275701 8033   8052     104 29008 R  1947809  16384   mysqld <none>
949418843669 949418843808 139    156        0     3 W    29024   2048  fsflush /var/log/…
949418873385 949418873488 103    121        0     3 W  6695855   2048  fsflush <none>
949418873564 949418873617 52     57         0     3 W  1829584    512  fsflush <none>
949418921970 949418932931 10960  10976    104 29008 R 95362430  16384   mysqld <none>
949419684613 949419692319 7706   7723     104 29952 R 81475146  16384   mysqld <none>
949419693574 949419699461 5886   5906     104 29952 R 60593276  16384   mysqld <none>
949422857833 949422857981 148    168        0     3 W    26720   4096  fsflush /var/adm/…
949423846191 949423846355 163    181        0     3 W  1990648   4096  fsflush /var/log/…
949426420134 949426420265 130    151        0     0 R      400   8192    sched <none>
949426420346 949426420423 77     85         0     0 W       65    512    sched <none>
949426420367 949426420459 92     35         0     0 W      129    512    sched <none>
949426420386 949426420490 103    30         0     0 W      146    512    sched <none>
949426420404 949426420566 161    76         0     0 W      193    512    sched <none>
949426420530 949426420604 73     37         0     0 W      206    512    sched <none>
949426420547 949426420679 131    75         0     0 W      210    512    sched <none>
[...thousands of lines...]
```

- A lot of data to sift through; effective as a last resort

Joyent smart computing

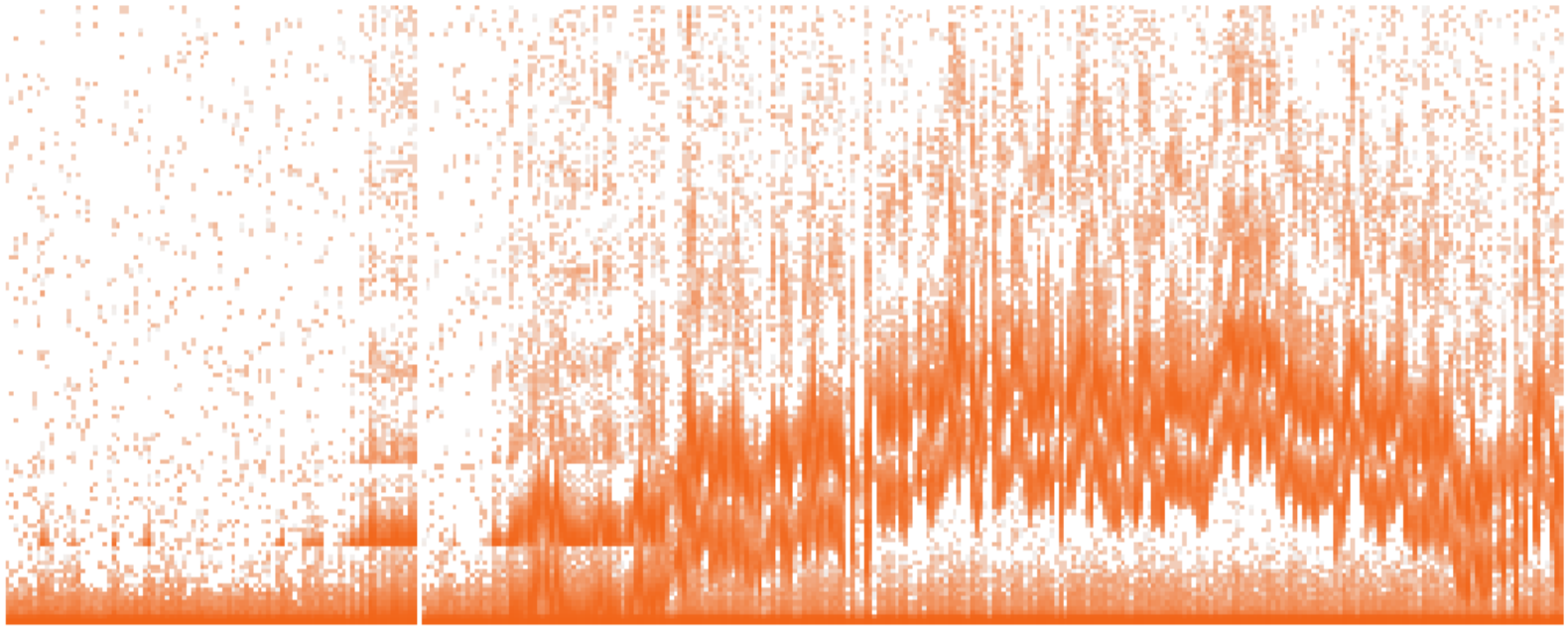
Latency: Average



- Some patterns more visible, outliers hidden
- x-axis = time, y-axis = average latency

Joyent smart computing

Latency: Distribution



- Great! And this example is MySQL query latency
- x-axis = time, y-axis = latency, z-axis (color) = count

Internals

- Not only is the distribution great, it's also efficient
- DTrace grouped data into buckets that span the distribution (quantized)
  - done in the kernel, at the point of data collection
- Summaries are read by user-land at a gentle rate
  - by default every 1 hertz (switchrate)
  - dtrace(1M) via ioctl()
- The performance impact is minimal

Joyent smart computing

MySQL Query Latency

- slow query log: useful, but original 1+ sec was coarse
- Fixed with microseconds patch (thanks Percona!)
  - fixed in 5.1.21 and 6.0.4, bug #25412
- Good starting point with DTrace:
  - can build upon query latency analysis to include and compare components
  - distribution plots can reveal details not visible from the slow query log
  - can be done from either mysql or pid providers

Joyent smart computing

mysql Provider

- A User Statically Defined Tracing (USDT) provider
- Added to 6.0.8 and 5.1.30 with --enable-dtrace
- One-liners
  - Trace query string:
    dtrace -n 'mysql*:::query-start { trace(copyinstr(arg0)); }'
  - Count query strings:
    dtrace -n 'mysql*:::query-start { @[copyinstr(arg0)] = count(); }'
  - Row event count:
    dtrace -n 'mysql*:::*-row-* { @[probename] = count(); }'
  - Lock event count:
    dtrace -n 'mysql*:::*lock-* { @[probename] = count(); }'

Joyent smart computing

mysql Provider Query Latency

- Can be measured as a (long) one-liner:

```
# dtrace –n ‘
mysql*:::query-start { self->start = timestamp; }
mysql*:::query-done /self->start/ {
 @[“ns”] = quantize(timestamp – self->start);
 self->start = 0;
}’
```

Joyent smart computing

mysql Provider Query Latency Distribution

```
ns
           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |                                         16
            4096 |@                                        93
            8192 |                                         19
           16384 |@@@                                      232
           32768 |@@                                       172
           65536 |@@@@@@                                   532
          131072 |@@@@@@@@@@@@@@@@@@                       1513
          262144 |@@@@@                                    428
          524288 |@@@                                      258
         1048576 |@                                        127
         2097152 |@                                        47
         4194304 |                                         20
         8388608 |                                         33
        16777216 |                                         9
        33554432 |                                         0
```

Joyent smart computing

mysql Provider Arguments

- Easily include other attributes (thanks to USDT):
  - query string: copyinstr(arg0)
  - database name: copyinstr(arg2)
  - user name: copyinstr(arg3)
  - host or IP: copyinstr(arg4)

Joyent smart computing

myqld_qsnoop.d

- Printing all these details:

```
# mysqld_qsnoop.d
TIME(ms) DATABASE          USER@HOST              ms RET QUERY
2208     wikidb            wikiuser@localhost      2   0 show tables
5974     wikidb            wikiuser@localhost     63   0 select * from user
8727     wikidb            wikiuser@localhost     22   0 select * from image
9590     wikidb            wikiuser@localhost      0   0 select * from image
29262    wikidb            wikiuser@localhost      0   1 select * from bogus
^C
```

- This does event-by-event tracing
  - example is from the DTrace book

Joyent smart computing

mysql Provider Probes

- Other probes exist for:
  - connection
  - command
  - query, query parse, query cache, query execution
  - row events
  - index reads
  - locks
  - filesort
  - statements
  - network
  - keycache
- http://dev.mysql.com/doc/refman/5.6/en/dba-dtrace-mysqld-ref.html

Joyent smart computing

pid Provider

- Dynamic Tracing
- Unstable Interface – scripts may break between releases of MySQL
- Works anywhere DTrace and the pid provider is available

Joyent smart computing

pid Provider Query Latency

- Can also be measured as a (long) one-liner:

```
# dtrace –n ‘
pid$target::*dispatch_command*:entry {
 self->start = timestamp; }
pid$target::*dispatch_command*:return /self->start/ {
 @[“ns”] = quantize(timestamp – self->start);
 self->start = 0;
}’ –p PID
```

- Matches “_Z16dispatch_command19enum_server_commandP3THDPcj” via c++filt is:
  dispatch_command(enum_server_command, THD*, char*, unsigned int)

Joyent smart computing

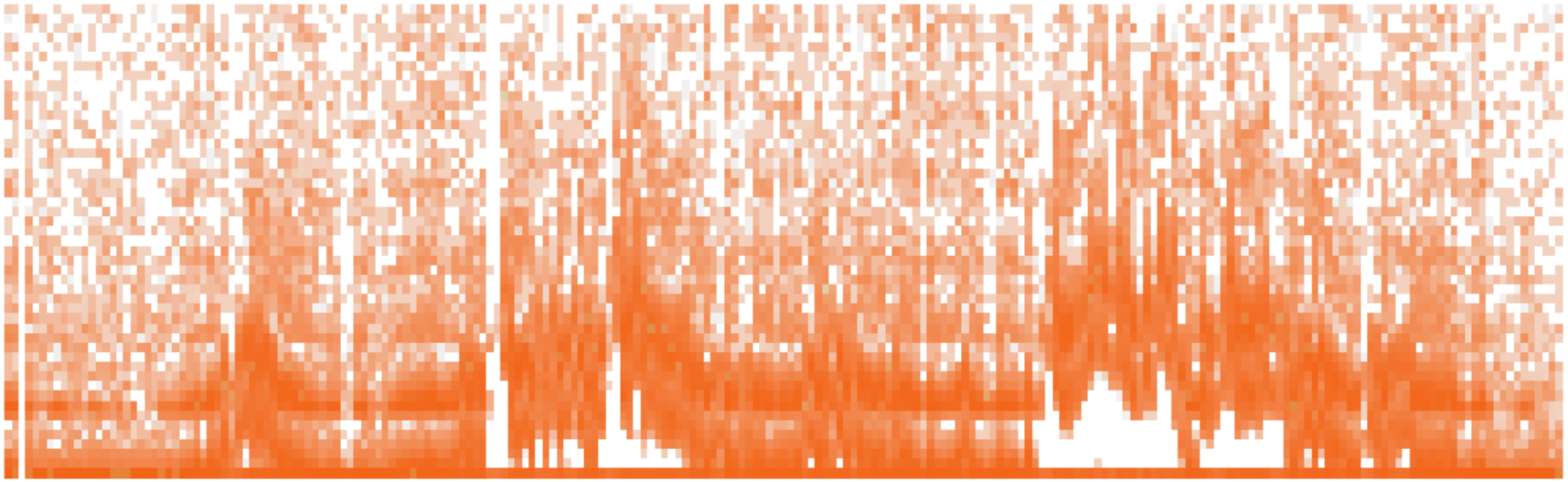
pid Provider

- DEMO
- mysqld_pid_latency.d

Joyent smart computing

Query Latency Distributions

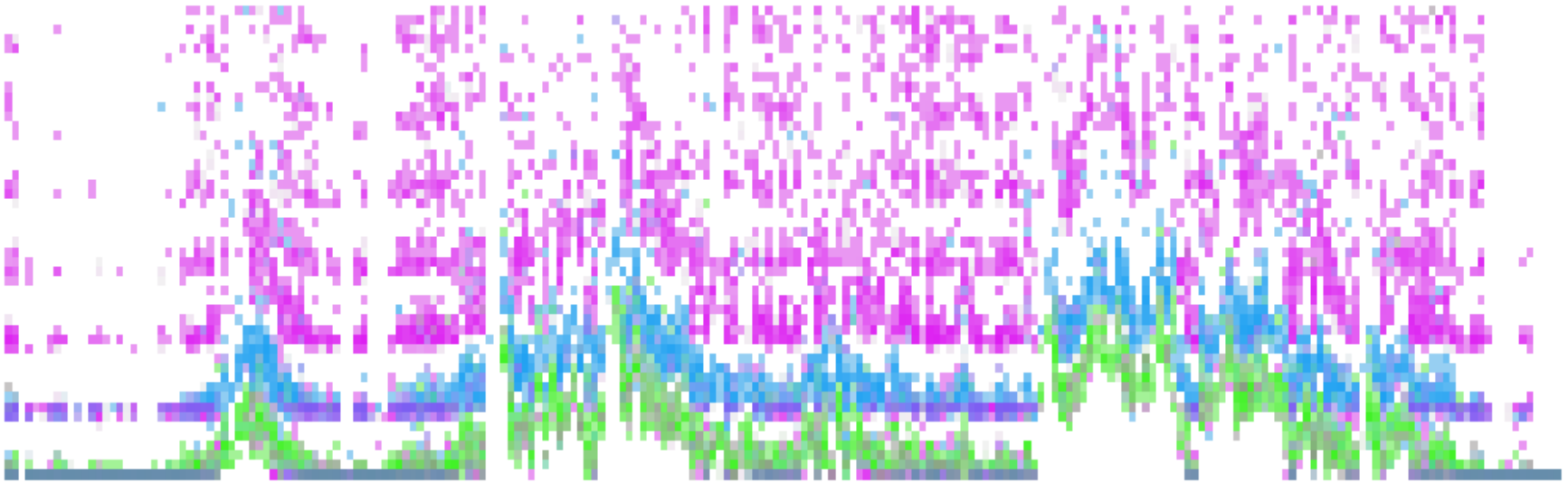
- With the right visualization, query latency distributions can reveal new information:



Joyent smart computing

Query Latency Distributions

- With the right visualization, query latency distributions can reveal new information:



- Highlighting 3 different tables from the queries

Joyent smart computing

Query Latency Components

- For a high-level breakdown, query latency can be broken into two components

Joyent smart computing

Query Latency Components

- For a high-level breakdown, query latency can be broken into two components:
- On-CPU

- Off-CPU

Joyent smart computing

Query Latency Components

- For a high-level breakdown, query latency can be broken into two components:
- On-CPU
  - Time spent running in MySQL processing a query
- Off-CPU
  - Time spent waiting for another resource

Joyent smart computing

Query Latency Components

- For a high-level breakdown, query latency can be broken into two components:
- On-CPU
  - Time spent running in MySQL processing a query
  - Examples: scanning a large table
- Off-CPU
  - Time spent waiting for another resource
  - Examples: blocked on file system I/O, network I/O, locks

Joyent smart computing

Query Latency Components

- For a high-level breakdown, query latency can be broken into two components:
- On-CPU
  - Time spent running in MySQL processing a query
  - Examples: scanning a large table
  - Teams: Developers, DBAs
- Off-CPU
  - Time spent waiting for another resource
  - Examples: blocked on file system I/O, network I/O, locks
  - Teams: SysAdmins, DBAs

Joyent smart computing

mysqld_pid_slow.d

- Answers on-CPU vs off-CPU breakdown

```
# ./mysqld_pid_slow.d -p 726 100
Tracing... Min query time: 100000000 ns.

 TIME(ms) CPU(ms)  QUERY
 111      4        SELECT xxx.* FROM xxx xxx WHERE xxx.xxx=xxx AND …
 283      0        UPDATE xxx SET xxx=xxx, xxx = xxx,xxx = xxx WHERE …
 155      0        UPDATE xxx SET xxx=2 WHERE xxx=xxx AND xxx=xxx AND …
 172      0        UPDATE xxx SET xxx=xxx WHERE xxx=xxx AND xxx=xxx\0
 173      6        SELECT count(*) xxx FROM xxx WHERE xxx =xxx AND …
 176      0        UPDATE xxx SET xxx=xxx WHERE xxx=xxx AND xxx=xxx\0
 178      177      SELECT xxx, xxx, xxx, xxx, xxx, xxx, xxx, xxx, xxx, xxx,
xxx, xxx, xxx FROM xxx WHERE xxx LIKE xxx AND xxx=xxx ORDER BY xxx DESC
LIMIT 10 OFFSET 0\0
 [...]
```

- TIME(ms): Query time, CPU(ms): Time spent on-CPU

Joyent smart computing

mysqld_pid_slow.d

- Answers on-CPU vs off-CPU breakdown

```
# ./mysqld_pid_slow.d -p 726 100
Tracing... Min query time: 100000000 ns.

 TIME(ms) CPU(ms)  QUERY
 111      4        SELECT xxx.* FROM xxx xxx WHERE xxx.xxx=xxx AND …
 283      0        UPDATE xxx SET xxx=xxx, xxx = xxx,xxx = xxx WHERE …
 155      0        UPDATE xxx SET xxx=2 WHERE xxx=xxx AND xxx=xxx AND …
 172      0        UPDATE xxx SET xxx=xxx WHERE xxx=xxx AND xxx=xxx\0
 173      6        SELECT count(*) xxx FROM xxx WHERE xxx =xxx AND …
 176      0        UPDATE xxx SET xxx=xxx WHERE xxx=xxx AND xxx=xxx\0
 178      177      SELECT xxx, xxx, xxx, xxx, xxx, xxx, xxx, xxx, xxx, xxx,
xxx, xxx, xxx FROM xxx WHERE xxx LIKE xxx AND xxx=xxx ORDER BY xxx DESC
LIMIT 10 OFFSET 0\0
 [...]
```

Off-CPU

On-CPU

- TIME(ms): Query time, CPU(ms): Time spent on-CPU

Joyent smart computing

mysqld_pid_slow.d script

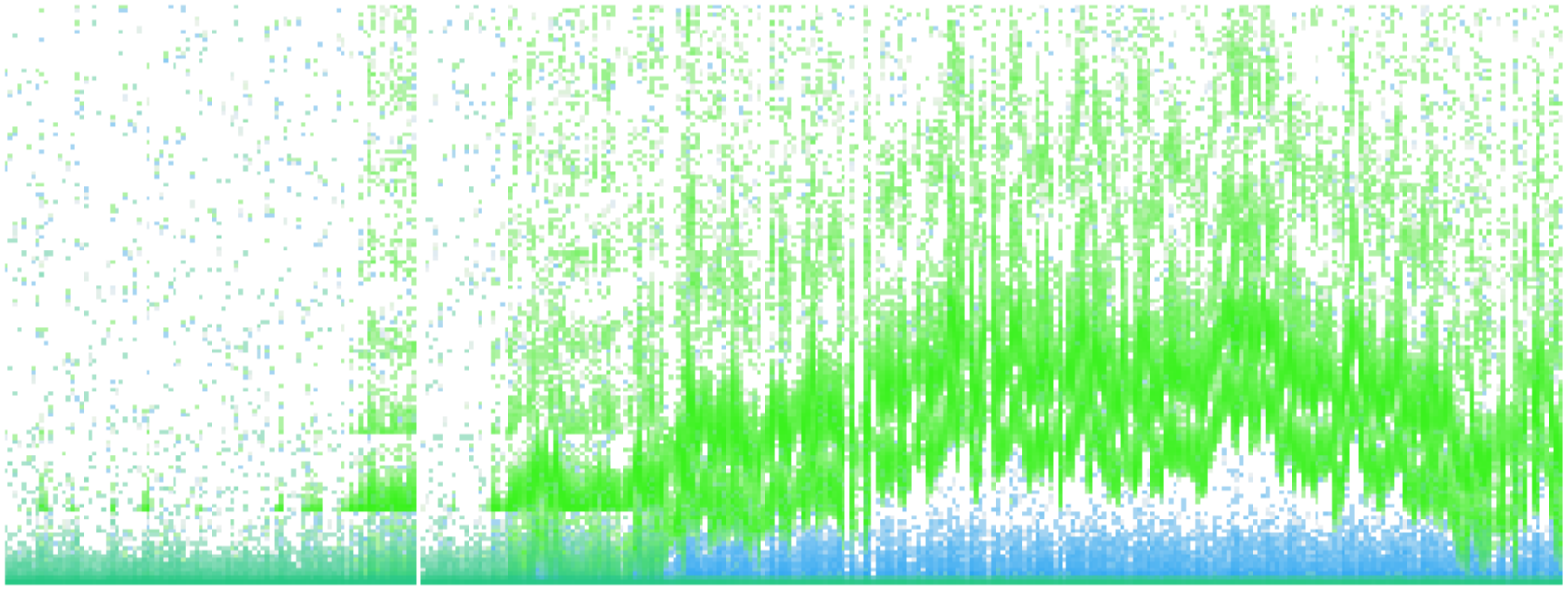
- timestamp deltas for query latency
- vtimestamp deltas for on-CPU time
  - vtimestamp is only incremented when that thread is on-CPU

```
[...]
pid$target::*dispatch_command*:entry
{
        self->query = copyinstr(arg2);
        self->start = timestamp;
        self->vstart = vtimestamp;
}

pid$target::*dispatch_command*:return
/self->start && (timestamp - self->start) > min_ns/
{
        this->time = (timestamp - self->start) / 1000000;
        this->vtime = (vtimestamp - self->vstart) / 1000000;
        printf(" %-8d %-8d %S\n", this->time, this->vtime, self->query);
}
[...]
```

Joyent smart computing

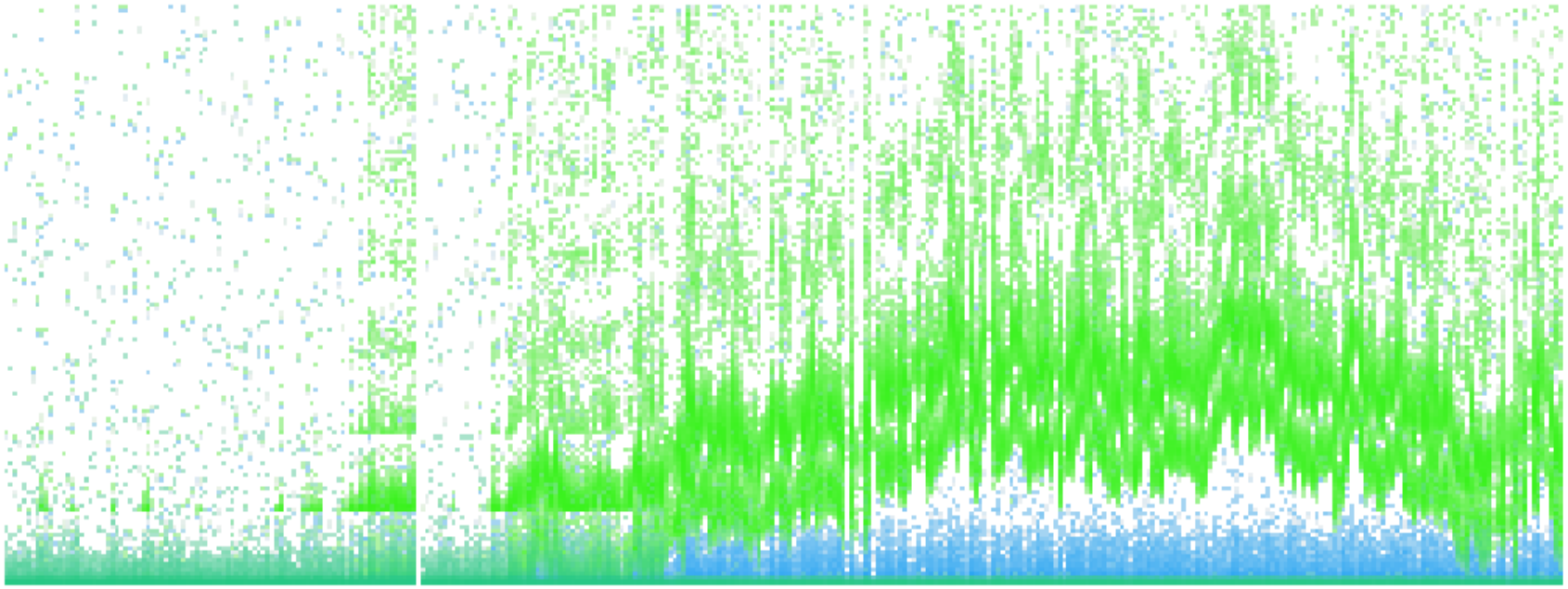
On-CPU Distribution



- Query Latency = green, On-CPU = blue

Joyent smart computing

On-CPU Distribution



- Query Latency = green, On-CPU = blue
- Divergence is innodb thread concurrency backoff
  - I think this has already been fixed in Percona!

Joyent smart computing

Analyzing Off-CPU Time

- Will be either blocking device I/O, or lock contention, or thread sleeps (voluntary off-CPU!)
- Analyze either generically:
  - Tracing thread context switch off-cpu to on-cpu latency
  - Can be done using the DTrace sched or syscall providers
- Or specifically:
  - Tracing related code for a type, such as File System latency via the MySQL storage engine

Joyent smart computing

File System Latency

- Using the traditional OS tools, this is usually inferred via iostat(1M):

```
# iostat –xnz 1 10
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    1.1   33.8   78.8 1208.1  0.0  1.0    0.0   27.8   0   4 c0t1d0
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  208.0    0.0 26619.9    0.0  0.0  1.9    0.0    9.2   0  99 c0t1d0
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  208.0    0.0 26624.4    0.0  0.0  1.7    0.0    8.2   0  95 c0t1d0
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  106.0  368.9 13566.1 26881.3  0.0  2.7    0.0    5.7   0  93 c0t1d0
[...]
```

Joyent smart computing

File System Latency

- Using the traditional OS tools, this is usually inferred via iostat(1M):

```
# iostat –xnz 1 10
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    1.1   33.8   78.8 1208.1  0.0  1.0    0.0   27.8   0   4 c0t1d0
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  208.0    0.0 26619.9    0.0  0.0  1.9    0.0    9.2   0  99 c0t1d0
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  208.0    0.0 26624.4    0.0  0.0  1.7    0.0    8.2   0  95 c0t1d0
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  106.0  368.9 13566.1 26881.3  0.0  2.7    0.0    5.7   0  93 c0t1d0
[...]
```

- Looks bad: up to 99% “busy” and 9.2 ms avg I/O time
- This may not resemble the latency felt by MySQL

Joyent smart computing

File System I/O != Disk I/O

- Disk I/O can include *unrelated* I/O from:
  - other applications
  - file system prefetch
  - file system dirty data flushing

Joyent smart computing

File System I/O != Disk I/O

- Disk I/O can include *unrelated* I/O from:
  - other applications
  - file system prefetch
  - file system dirty data flushing
- Disk I/O can be *inflated* from the File System:
  - Rounded up to the file system record size
  - Extra metadata for on-disk format
  - read-modify-write of RAID-5

Joyent smart computing

File System I/O != Disk I/O cont.

- Disk I/O can be *deflated* from the File System:
  - Read caching
  - Write buffering

Joyent smart computing

File System I/O != Disk I/O cont.

- Disk I/O can be *deflated* from the File System:
  - Read caching
  - Write buffering
- Disk I/O analysis can be *blind* to File System issues:
  - Lock contention in the file system
  - File system software bugs
  - File system queue latency
  - Disk cache flush latency

Joyent smart computing

Tracing File System Latency

- Using the DTrace pid provider:

```
# ./mysqld_pid_fslatency.d -n 'tick-10s { exit(0); }' -p 7357
Tracing PID 7357... Hit Ctrl-C to end.
MySQL filesystem I/O: 55824; latency (ns):

  read
           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |@@@@@@@@@@                               9053
            4096 |@@@@@@@@@@@@@@@@@@                       15490
            8192 |@@@@@@@@@@@@                             9525
           16384 |@@                                       1982
           32768 |                                         121
           65536 |                                         28
          131072 |                                         6
          262144 |                                         0
```

Joyent smart computing

Tracing File System Latency cont.

```
write
           value  ------------- Distribution ------------- count
            2048 |                                         0
            4096 |                                         1
            8192 |@@@@@@                                   3003
           16384 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@        13532
           32768 |@@@@@                                    2590
           65536 |@                                        370
          131072 |                                         58
          262144 |                                         27
          524288 |                                         12
         1048576 |                                         1
         2097152 |                                         0
         4194304 |                                         10
         8388608 |                                         14
        16777216 |                                         1
        33554432 |                                         0
```

- Latency suggests most I/O is returning out of DRAM

Joyent smart computing

Tracing File System Latency cont.

```
write
           value  ------------- Distribution ------------- count
            2048 |                                         0
            4096 |                                         1
            8192 |@@@@@@                                   3003
           16384 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@        13532
           32768 |@@@@@                                    2590
           65536 |@                                        370
          131072 |                                         58
          262144 |                                         27
          524288 |                                         12
         1048576 |                                         1
         2097152 |                                         0
         4194304 |                                         10
         8388608 |                                         14
        16777216 |                                         1
        33554432 |                                         0
```

- Latency suggests most I/O is returning out of DRAM
- Traced at the *exact same time* as previous iostat(1M)

Joyent smart computing

mysqld_pid_fslatency.d script

```
[...]
pid$target::os_file_read:entry, pid$target::os_file_write:entry,
pid$target::my_read:entry, pid$target::my_write:entry
{
        self->start = timestamp;
}

pid$target::os_file_read:return  { this->dir = "read"; }
pid$target::os_file_write:return { this->dir = "write"; }
pid$target::my_read:return       { this->dir = "read"; }
pid$target::my_write:return      { this->dir = "write"; }

pid$target::os_file_read:return, pid$target::os_file_write:return,
pid$target::my_read:return, pid$target::my_write:return
/self->start/
{
        @time[this->dir] = quantize(timestamp - self->start);
        @num = count();
        self->start = 0;
}
[...]
```

Joyent smart computing

Storage I/O Stack Example

1. MySQL Query
2. Storage Engine I/O ← mysqld_pid_fslatency.d
3. System Calls
4. Thread Scheduler
5. VFS
6. ZFS/UFS (and FS internals)
7. Block Device (sd)
8. SAS driver
9. PCI driver

- Any of these can be a source of latency

Joyent smart computing

As a Component of Query Latency

- File system latency can occur outside of queries, in other mysqld server threads
- To positively identify this latency as causing slow queries, it can be expressed as a ratio of query time
- mysqld_pid_fslatency_slowlog.d does this:

```
# ./mysqld_pid_fslatency_slowlog.d 29952
2011 May 16 23:34:00 filesystem I/O during query > 100 ms: query 538 ms, fs 509 ms, 83 I/O
2011 May 16 23:34:11 filesystem I/O during query > 100 ms: query 342 ms, fs 303 ms, 75 I/O
2011 May 16 23:34:38 filesystem I/O during query > 100 ms: query 479 ms, fs 471 ms, 44 I/O
2011 May 16 23:34:58 filesystem I/O during query > 100 ms: query 153 ms, fs 152 ms, 1 I/O
2011 May 16 23:35:09 filesystem I/O during query > 100 ms: query 383 ms, fs 372 ms, 72 I/O
2011 May 16 23:36:09 filesystem I/O during query > 100 ms: query 406 ms, fs 344 ms, 109 I/O
2011 May 16 23:36:44 filesystem I/O during query > 100 ms: query 343 ms, fs 319 ms, 75 I/O
[...]
```

Joyent smart computing

mysqld_pid_fslatency_slowlog.d script

- Measures query latency and sums FS latency:

```
[...]
pid$1::os_file_read:entry,
pid$1::os_file_write:entry,
pid$1::my_read:entry,
pid$1::my_write:entry
/self->q_start/
{
        self->fs_start = timestamp;
}

pid$1::os_file_read:return,
pid$1::os_file_write:return,
pid$1::my_read:return,
pid$1::my_write:return
/self->fs_start/
{
        self->total_ns += timestamp - self->fs_start;
        self->io_count++;
        self->fs_start = 0;
}
[...]
```

self->q_start is only set during queries, for measuring query latency. It's checked here to ensure we only trace file system I/O during queries.

Joyent smart computing

Other Components

- Can be traced and examined in a similar fashion
- The previous scripts can be modified to include others from the 22,000+ available probes
- There are ways to narrow down likely candidates, such as examining off-CPU stacks

Joyent smart computing

Generic Off-CPU Latency

- DEMO

Joyent smart computing

Generic On-CPU Latency

- DEMO

Joyent smart computing

Other Uses

- Apart from performance and latency analysis, Dynamic Tracing can be used for:

Joyent smart computing

Other Uses

- Apart from performance and latency analysis, Dynamic Tracing can be used for:
- Patch Checking
  - Upgrading software or patches usually costs downtime
  - Use DTrace to verify that the patched code-path is taken, to confirm that the patch is worth the cost

Joyent smart computing

Other Uses

- Apart from performance and latency analysis, Dynamic Tracing can be used for:
- Patch Checking
  - Upgrading software or patches usually costs downtime
  - Use DTrace to verify that the patched code-path is taken, to confirm that the patch is worth the cost
- Troubleshooting
  - If something isn't working and the traditional tools & logs aren't enough, DTrace can follow the code to the origin of the failure

Tools Used

- pid provider scripts mysqld_pid_*.d were from my blog, which includes the full listings:
  - http://dtrace.org/blogs/brendan/tag/mysql/
- mysql provider scripts are from the DTrace book, and available here:
  - http://dtracebook.com/index.php/Databases
- Distribution Visualizations ("heat maps") are being developed for Joyent Cloud Analytics:
  - http://www.joyent.com/software/smartdatacenter/cloud-analytics/
  - http://joyeur.com/2011/01/24/executive-speaker-series-bryan-cantrill-and-brendan-gregg-on-cloud-analytics/

Joyent smart computing

More Info

- http://dtrace.org — Many DTrace blogs
- http://dtrace.org/blogs/brendan — My (work) blog
- http://dtrace.org/blogs/brendan/tag/mysql — mysql related posts
- http://www.brendangregg.com/dtrace.html — My DTrace page
- http://www.dtracebook.com — All book scripts + sample chapter
- http://www.joyent.com — Company site

@brendangregg on twitter