Troubleshooting MySQL

Baron Schwartz

Agenda

- What is troubleshooting?
- Troubleshooting method
- Troubleshooting performance problems
- Troubleshooting intermittent problems
- Troubleshooting tools
- Case studies

Slides will be available online.

Part 1: Introduction

What's Happening?

- Percona customer case 22667
  - NOT a perfectly handled case!
- 120404 10:22:11 [ERROR] Slave SQL: Error 'Lock wait timeout exceeded; try restarting transaction' on query. Default database: 'foo'. Query: 'update Trips set id="650778",changed="2012-04-04 10:21:20",is_paid="1" where id = 650778', Error_code: 1205

InnoDB Status Information?

mysql tables in use 1, locked 1

LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)

MySQL thread id 67461359, query id 3090919832 Updating

Update... = 650778

------- TRX HAS BEEN WAITING 36 SEC FOR THIS LOCK

index `PRIMARY` lock_mode X locks rec but not gap waiting

TABLE LOCK table lock mode IX

Any Other Transactions?

---TRANSACTION 2EF7E984E, ACTIVE 36 sec, process no 24348, OS thread id 139928730597120

437 lock struct(s), heap size 47544, 508 row lock(s)

MySQL thread id 83363361, query id 3090919874 10.88.182.107 webapp_read

Trx read view will not see trx with id >= 2EF7E984F, sees < 2E93F9A5F

TABLE LOCK table `foo`.`Trips` trx id 2EF7E984E lock mode IS

Next Steps?

- Not enough information!
- Enable the general query log

Ultimate Diagnosis...

- http://bugs.mysql.com/bug.php?id=46947
- SELECT * FROM Table1 where fieldValue= (SELECT fieldValue FROM Table2 where otherValue='something');

What is Troubleshooting?

- Discovery?
- Diagnosis?
- Resolution?
- Prevention?

What is Troubleshooting?

- Discovery
- Diagnosis
- Resolution
- Prevention

High-Level Overview

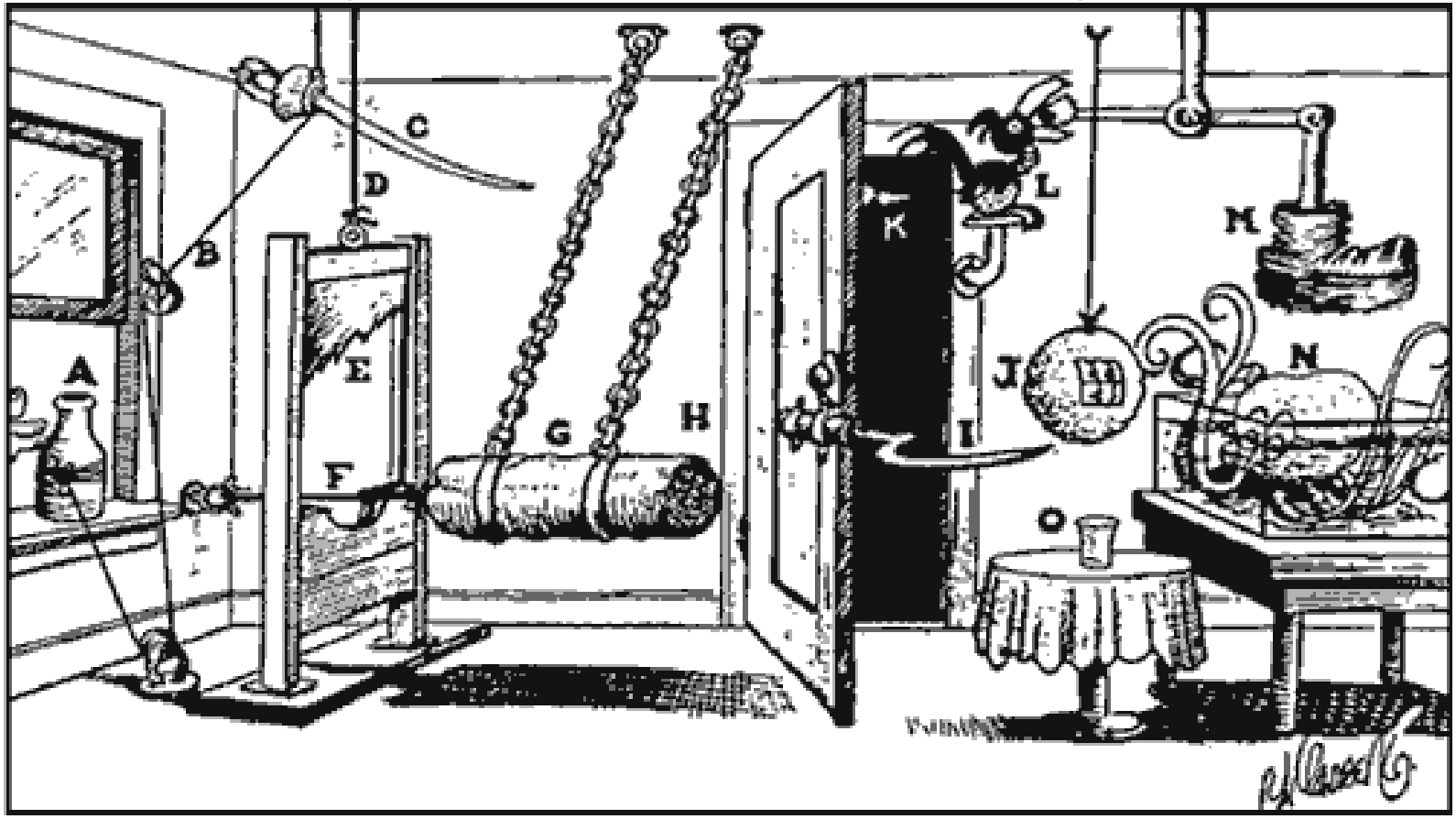
- Frame the question
- Understand the system
- Identify the missing information
- Get the information
- Interpret the information
- Find the cause
- Find the solution (?)

Interpreting Data

- Understand and characterize the data
- Identify causes and effects clearly
- Don't blame the victim

Finding the Solution

The answer will be obvious, if you have the right data and you understand the system well



Part II: Troubleshooting Method

What We're Troubleshooting

- Crashes
- Bugs
- High load on the server
- Slow queries
- Server stalls

One Ring To Rule Them All

One Ring To Rule Them All

- Computers are state machines
- Understand the system internals
- Begin at the beginning
- Work from beginning to end

No Guessing, No Shortcuts

In reality we will balance the purist approach with the pragmatic approach.

You can get a lot done with crude, inexact tools and methods.

BUT!

Learn the rules first, and then you will know when to break them.

Part III: Performance Troubleshooting

What is Performance?

- How do you measure performance?
- How is it different from these?
  - Throughput
  - Latency
  - Scalability
  - Load
  - CPU utilization

In MY opinion:

Performance is Response Time

- The units are time / tasks
- Throughput is inversely related
  - But not a reciprocal, and not the same
- All those other things are not performance

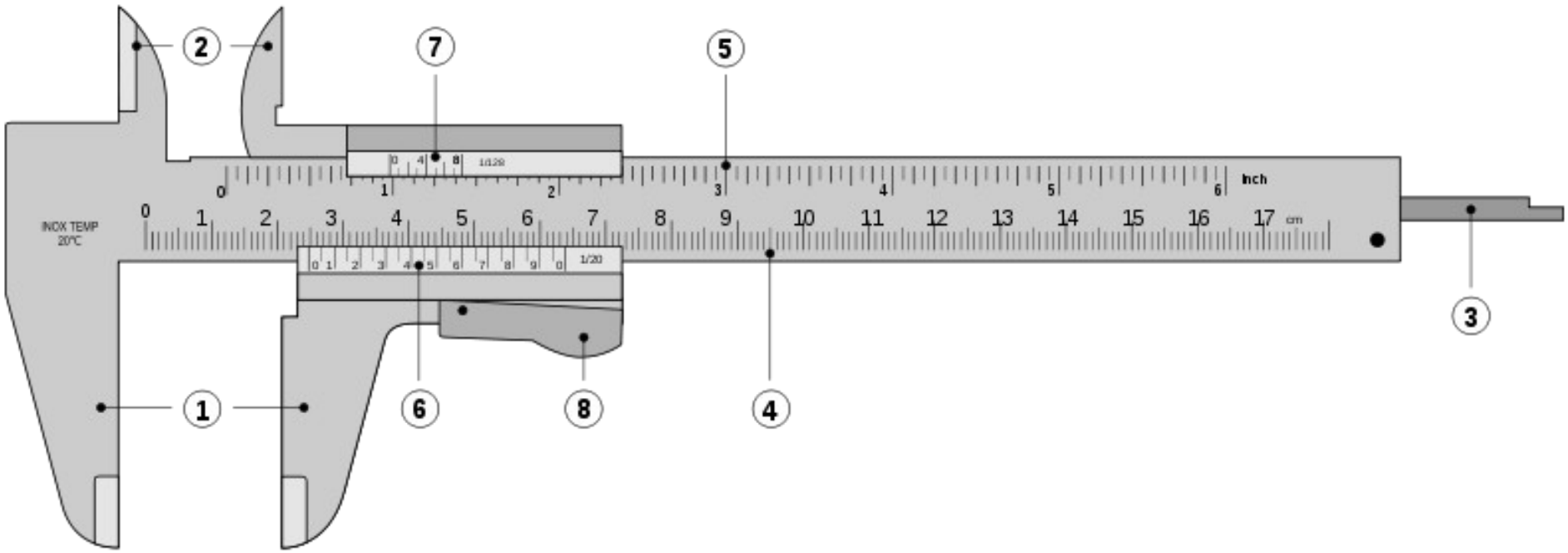
What is Optimization?

The system is optimal when the cost of improvement is higher than the benefit.

The Key to Success

- You don't need to do 100 things.
- You don't even need to do 7 things.
- You only need to do one thing!

Measure



The #1 Mistake

Spending too little time understanding and measuring the system

Why Measure?

Because you cannot optimize what you cannot measure.

What to Measure

- Everything you can
- Focus on the goal

Proper Scoping

- Measure only the thing you care about
- Measure only when the problem occurs

Common Scoping Mistakes

- Aggregate measurements
  - Global counters
- Non-system measurements
  - System-wide IO usage
- Wrong time frame
  - Counters since system boot

Focus on the Goal

- Measure what you want to improve
- Focusing on something else (e.g. CPU usage) is a common mistake

What if Measurements are Wrong?

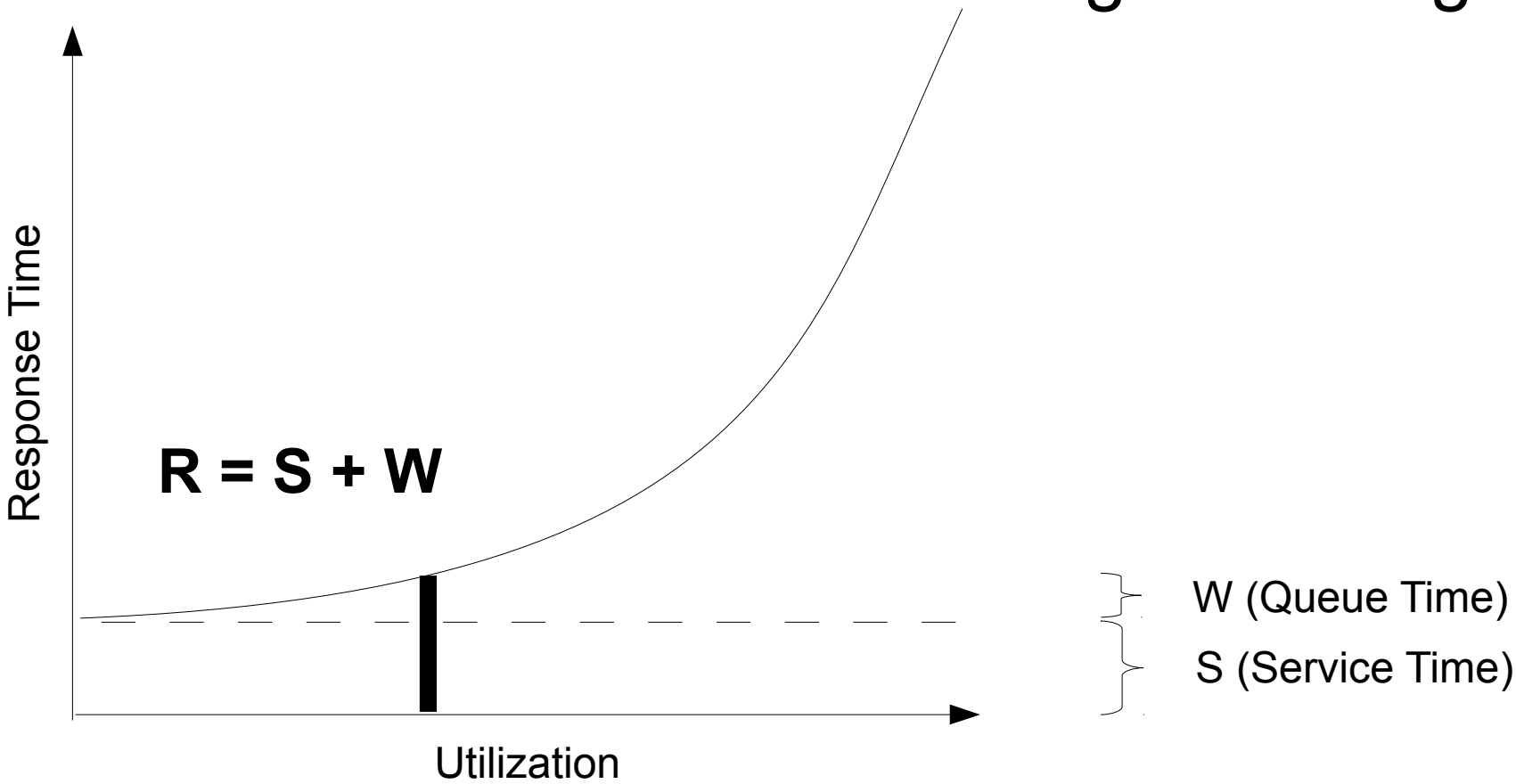
Measurements are wrong by definition.

Q: Why are Tasks Slow?

- A: Because they take too much time.
- Q: Why do they take too much time?
- A: One of three reasons.
  - They do things they shouldn't.
  - They do things too many times.
  - The things they do are too slow.
    - Q: Why are the things they do too slow?
    - A: Because they take too much time.
    - Q: Why do...

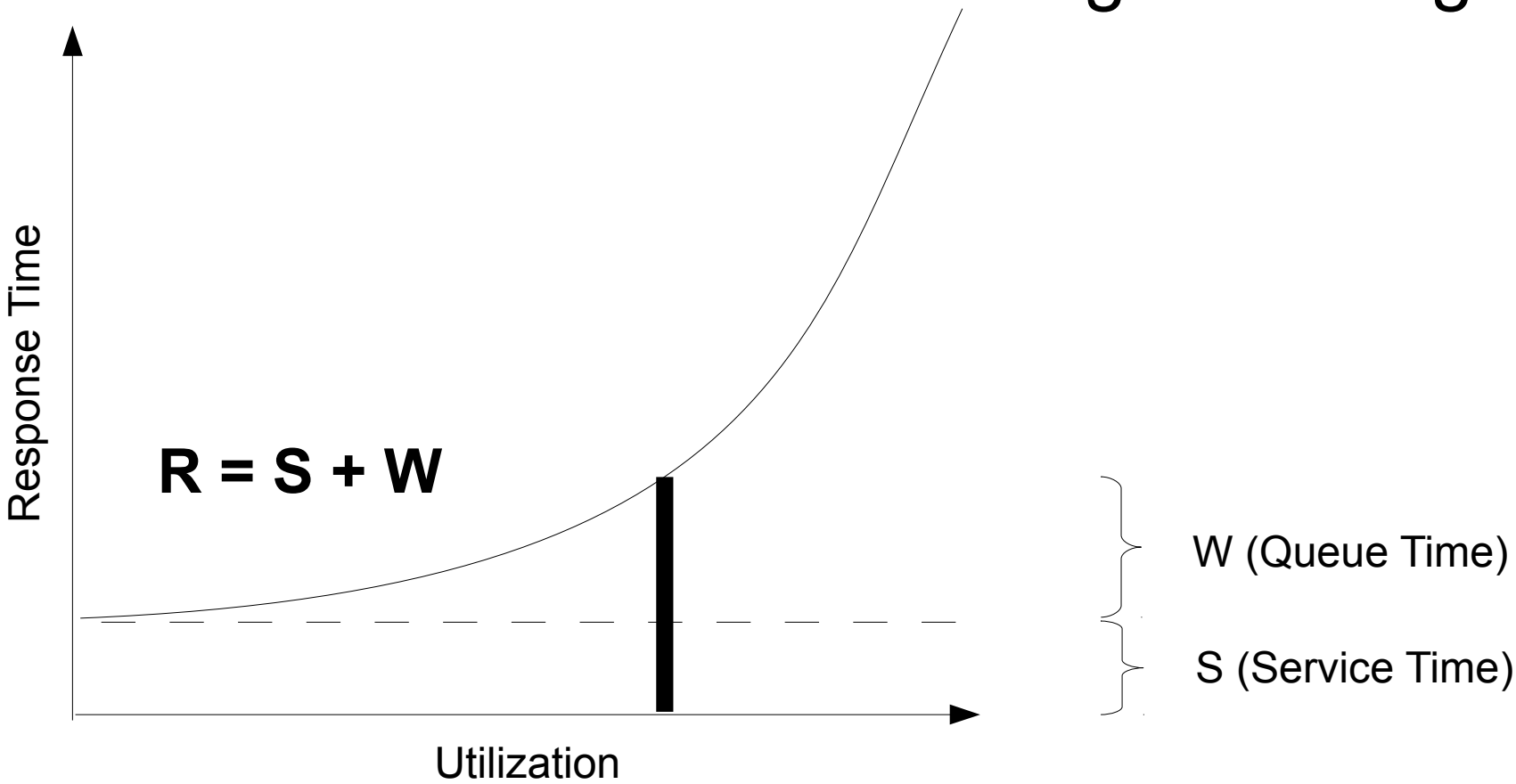
A Quick Intro to Queueing Theory

Tasks are either executing or waiting.



A Quick Intro to Queueing Theory

Tasks are either executing or waiting.



Want Some Math?

The formal definition is given by the Erlang C formula, for the most common type of traffic (random arrivals, exponentially distributed service times). It gives the probability a request must wait.

$$E_c(m,u)=\frac{\frac{u^m}{m!}}{\frac{u^m}{m!}+(1-\rho)\sum_{k=0}^{m-1}\frac{u^k}{k!}}$$

Read more at mitan.co.uk/erlang/elgcmath.htm

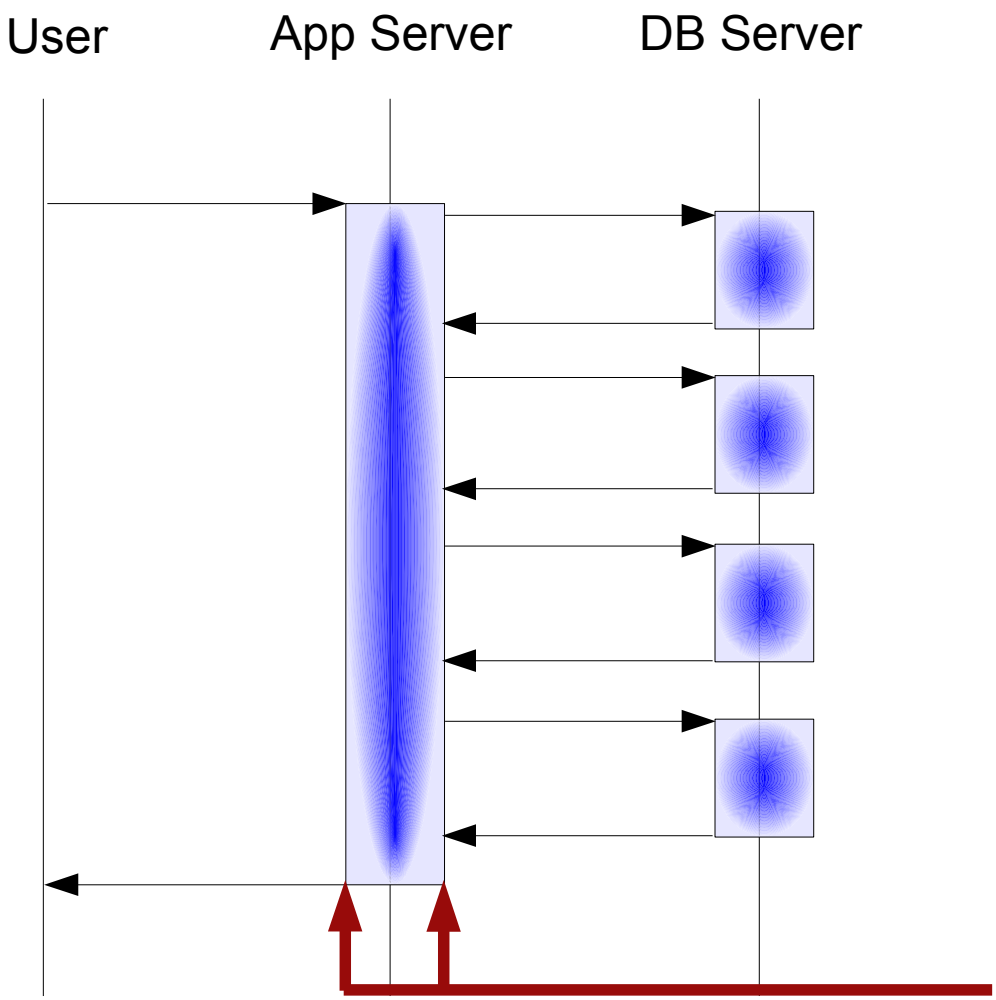
Why Does Queueing Matter?

- Understanding queueing makes troubleshooting easier.
- Many tools are best suited for measuring either resource consumption or waiting, but not both.
- Example:
  - oprofile shows you where time is consumed
  - GDB backtrace analysis shows waiting
- You need to pick the appropriate tool in many cases.

Measuring Where Time Goes

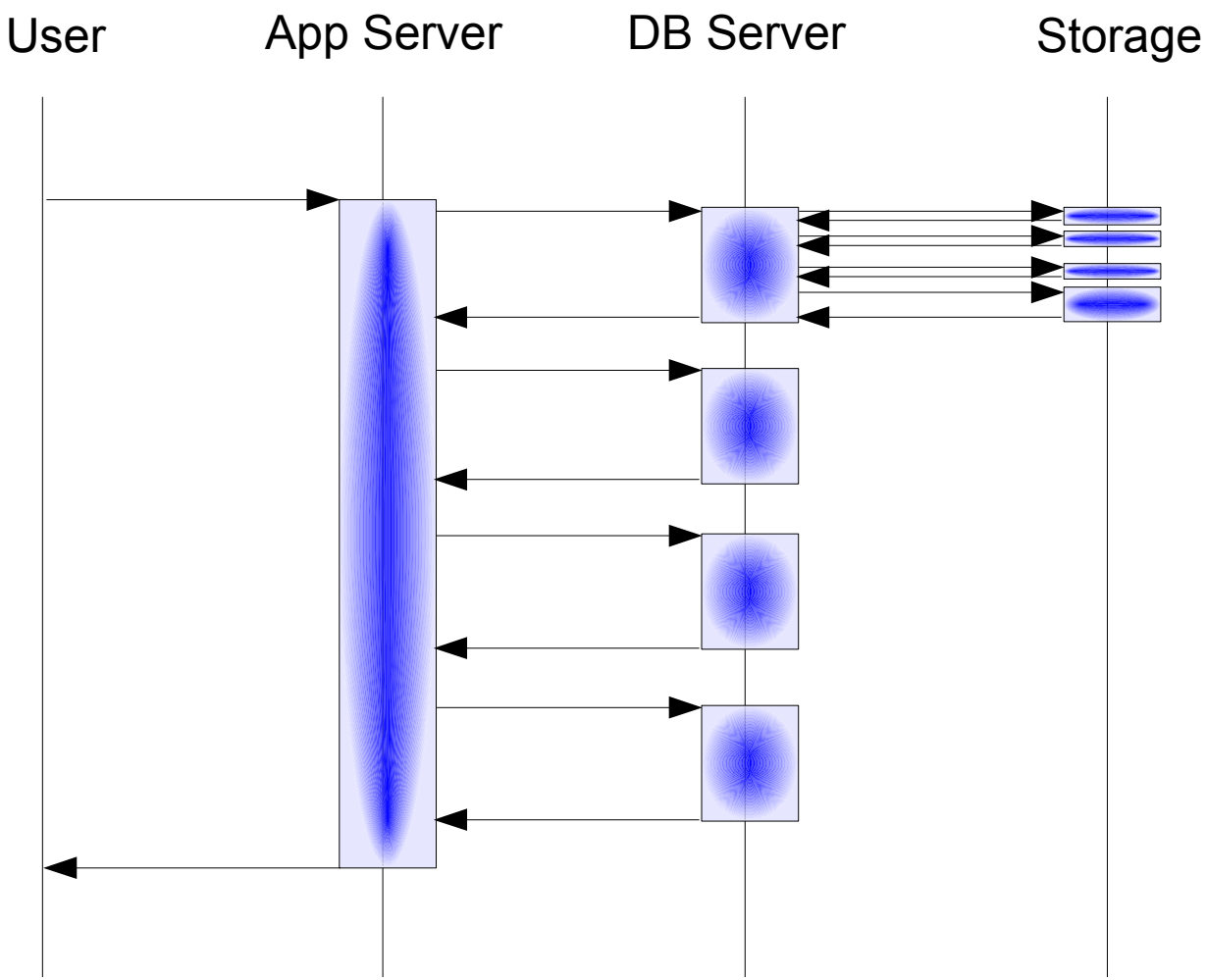
Back to the basics: understand the execution path, and measure the time elapsed on it.

The Sequence Diagram



Inclusive versus exclusive time: measuring one layer up or down

The Sequence Diagram



This is complete, but not very helpful. How can we make it more helpful?

Which Layer to Measure

- You can measure the DB from the app server
- Or you can measure the DB from the DB
- The risk of measuring the DB externally is you can't measure everything that happens on it
- The benefit is that you can see its usage in context

Good Enough

- Perfection demands internal instrumentation
- In reality, external (tcpdump?) is often enough

The Profile

A profile is an aggregated sequence diagram.

```
Rank Response time    Calls R/Call Item
==== ================ ===== ====== =======
   1 11256.3618 68.1% 78069 0.1442 SELECT InvitesNew
   2  2029.4730 12.3% 14415 0.1408 SELECT StatusUpdate
   3  1345.3445  8.1%  3520 0.3822 SHOW STATUS
   4  1341.6432  8.1%  3509 0.3823 SHOW STATUS
```

How to Read the Profile

- What can you see?
  - How often similar tasks execute
  - The most expensive type of task is at the top
- Profiles are helpful, but not complete
  - Opposite of the sequence diagram

What is Missing?

- Worthwhile-ness and optimizability
  - Ahmdahl's Law, ???
- Outliers
  - Some tasks are suboptimal but infrequent
- Distribution
  - Averages hide the details
- Missing data
  - The “unknown unknowns”

Three Ways to Profile MySQL

- Profile the entire workload
- Profile an individual query
- Profile the server internals

When is each of these useful?

Profiling the Workload

- Use pt-query-digest and the slow query log
  - Or capture queries from tcpdump
- MySQL Enterprise Monitor Query Analyzer
- New Relic
- xhprof
- instrumentation-for-php
- PERFORMANCE_SCHEMA (someday)

Using pt-query-digest

```
# Profile
# Rank Query ID           Response time    Calls R/Call V/M   Item
# ==== ================== ================ ===== ====== ===== =======
#    1 0xBFCF8E3F293F6466 11256.3618 68.1% 78069 0.1442  0.21 SELECT InvitesNew?
#    2 0x620B8CAB2B1C76EC  2029.4730 12.3% 14415 0.1408  0.21 SELECT StatusUpdate?
#    3 0xB90978440CC11CC7  1345.3445  8.1%  3520 0.3822  0.00 SHOW STATUS
#    4 0xCB73D6B5B031B4CF  1341.6432  8.1%  3509 0.3823  0.00 SHOW STATUS
# MISC 0xMISC               560.7556  3.4% 23930 0.0234   0.0 <17 ITEMS>
```

Using pt-query-digest

```
# Query 1: 24.28 QPS, 3.50x concurrency, ID 0xBFCF8E3F293F6466 at byte 5590079
# This item is included in the report because it matches --limit.
# Scores: V/M = 0.21
# Query_time sparkline: | _^_.^_ |
# Time range: 2008-09-13 21:51:55 to 22:45:30
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         63   78069
# Exec time     68  11256s    37us      1s   144ms   501ms   175ms    68ms
# Lock time     85    134s       0   650ms     2ms   176us    20ms    57us
# Rows sent      8  70.18k       0       1    0.92    0.99    0.27    0.99
# Rows examine   8  70.84k       0       3    0.93    0.99    0.28    0.99
# Query size    84  10.43M     135     141  140.13  136.99    0.10  136.99
# String:
# Databases    production
# Hosts
# Users        fbappuser
```

Using pt-query-digest

```
# Query_time distribution
#   1us
#  10us  #
# 100us  ##################################################
#   1ms  ###
#  10ms  ################
# 100ms  ##########################################################
#    1s  #
#  10s+
# Tables
#    SHOW TABLE STATUS FROM `production ` LIKE 'InvitesNew82'\G
#    SHOW CREATE TABLE `production `.`InvitesNew82`\G
# EXPLAIN /*!50100 PARTITIONS*/
SELECT InviteId, InviterIdentifier FROM InvitesNew82 WHERE (InviteSetId = 87041469)
AND (InviteeIdentifier = 1138714082) LIMIT 1\G
```

Profiling Individual Queries

- SHOW PROFILES
- Slow query log in Percona Server
- PERFORMANCE_SCHEMA (someday)
- SHOW STATUS? Never.
  - Why not?

The Ultimate Tool

The “slow query log” is overall the richest, best source of diagnostic data in MySQL at the moment. This will change in future releases.

Using SHOW PROFILES

```
mysql> SELECT * FROM sakila.nicer_but_slower_film_list;
[query results omitted]
```

Using SHOW PROFILES

```
mysql> SELECT STATE, SUM(DURATION) AS Total_R,
          ROUND(
             100 * SUM(DURATION) /
                (SELECT SUM(DURATION)
                 FROM INFORMATION_SCHEMA.PROFILING
                 WHERE QUERY_ID = @query_id
             ), 2) AS Pct_R,
          COUNT(*) AS Calls,
          SUM(DURATION) / COUNT(*) AS "R/Call"
       FROM INFORMATION_SCHEMA.PROFILING
       WHERE QUERY_ID = @query_id
       GROUP BY STATE
       ORDER BY Total_R DESC;
```

STATE	Total_R	Pct_R	Calls	R/Call
Copying to tmp table	0.090623	54.05	1	0.0906230000
Sending data	0.056774	33.86	3	0.0189246667
Sorting result	0.011555	6.89	1	0.0115550000
removing tmp table	0.005890	3.51	3	0.0019633333
logging slow query	0.000792	0.47	2	0.0003960000
checking permissions	0.000576	0.34	5	0.0001152000
Creating tmp table	0.000463	0.28	1	0.0004630000
Opening tables	0.000459	0.27	1	0.0004590000
statistics	0.000187	0.11	2	0.0000935000
starting	0.000082	0.05	1	0.0000820000
preparing	0.000067	0.04	2	0.0000335000
freeing items	0.000059	0.04	2	0.0000295000
optimizing	0.000059	0.04	2	0.0000295000
init	0.000022	0.01	1	0.0000220000
Table lock	0.000020	0.01	1	0.0000200000
closing tables	0.000013	0.01	1	0.0000130000
System lock	0.000010	0.01	1	0.0000100000
executing	0.000010	0.01	2	0.0000050000
end	0.000008	0.00	1	0.0000080000
cleaning up	0.000007	0.00	1	0.0000070000
query end	0.000003	0.00	1	0.0000030000

Now What?

- Optimize the query, of course.
  - Optimizing the query is another topic.
  - But the profile makes it obvious that the temp table is a problem.
    - However, we don't know whether *copying data into* the table, or *getting the data to copy into the table*, is the problem.
- What if..
  - The query doesn't seem optimizable?
  - The query is only slow when you're not looking?

- Take A Break -

Part IV: Intermittent Problems

What's Different?

- Hard to observe == hard to solve.
- It's tempting to revert to trial and error.

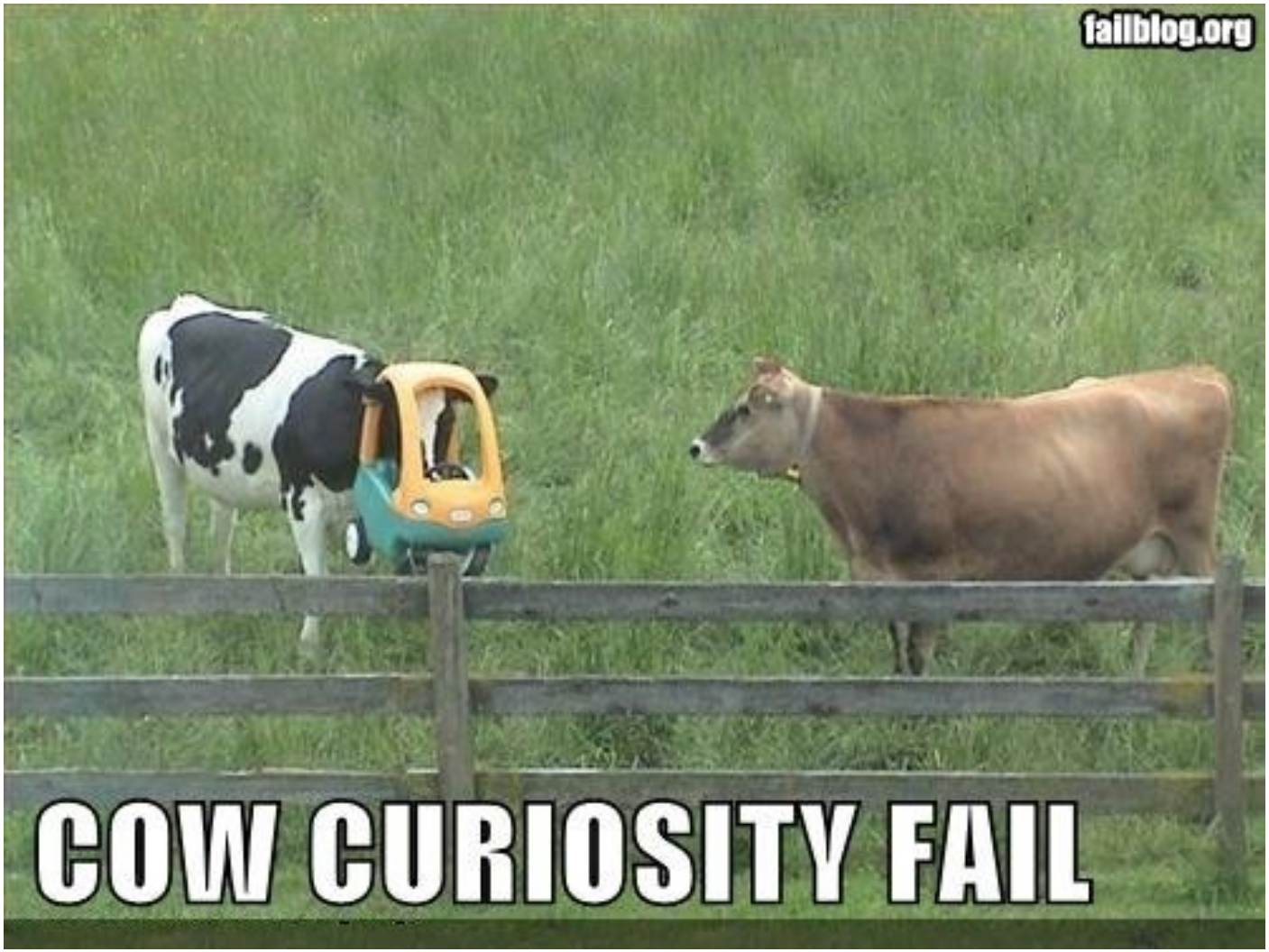
War Story #1

“Histamine”

War Story #2

“Bed Slime Omen”

If You Guess, You Fail

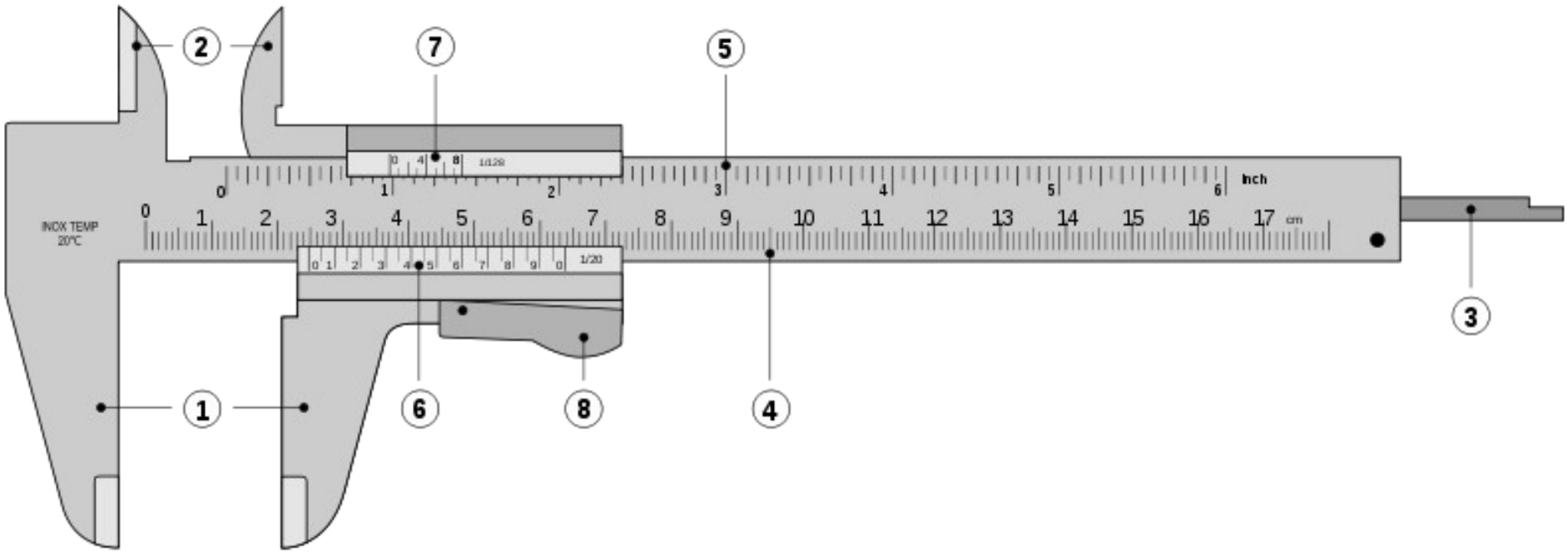


Sad Stories

- App executes curl
- DNS times out
- Memcached expirations cause stampede
- Query cache locks up
- InnoDB has mutex contention

The First Step

Measure



Understand the Scope

- Is it a problem with a single query?
- Or is it server-wide?
- Why does it matter?
  - Distinguishing victims from perpetrators
- How can you tell?

Determining Scope

- Look for evidence of server-wide problem
- Three easy techniques:
  - SHOW STATUS
  - SHOW PROCESSLIST
  - Query log analysis

Using SHOW STATUS

```
$ mysqladmin ext -i1 | awk '
   /Queries/{q=$4-qp;qp=$4}
   /Threads_connected/{tc=$4}
   /Threads_running/{printf "%5d %5d %5d\n", q, tc, $4}'
2147483647   136     7
  798   136     7
  767   134     9
  828   134     7
  683   134     7
  784   135     7
  614   134     7
  108   134    24
  187   134    31
  179   134    28
 1179   134     7
 1151   134     7
 1240   135     7
 1000   135     7
```

Drop in Queries Per Second
Spike of Threads_running
Threads_connected doesn't change

Using SHOW PROCESSLIST

```
$ mysql –e 'SHOW PROCESSLIST\G' | grep State: \
   | sort | uniq -c | sort -rn
    744   State:
     67   State: Sending data
     36   State: freeing items
      8   State: NULL
      6   State: end
      4   State: Updating
      4   State: cleaning up
      2   State: update
      1   State: Sorting result
      1   State: logging slow query
```

Using The Slow Query Log

```
$ awk '/^# Time:/{print $3, $4, c;c=0}/^# User/{c++}' slow-query.log
080913 21:52:17 51
080913 21:52:18 29
080913 21:52:19 34
080913 21:52:20 33
080913 21:52:21 38
080913 21:52:22 15
080913 21:52:23 47
080913 21:52:24 96
080913 21:52:25 6
080913 21:52:26 66
080913 21:52:27 37
080913 21:52:28 59
```

Spike, followed by a drop, in queries per second

When to Guess

- Honor your intuition, but verify it
- A guess can bypass a full investigation
- Guessing isn't the same thing as trial and error
- Don't trust the guess unless you can prove it

Part V: Diagnosis Tools

Tools Are Essential

- You need to measure the problem, whether you can observe it or not.
  - Even if you see the problem happen, you can't observe 45 things at once.
  - If you can't see it happen, you can still capture diagnostic data
- Percona Toolkit
  - pt-stalk
  - pt-sift

The Diagnostic Trigger

- Determine a reliable condition to trigger the tool
- Not too low!
  - You'll get false positives
- Not too high!
  - You'll miss the problem and it will hurt longer
  - You'll diagnose the wrong problem

The Threshold

- Threads_running is very good
- Threads_connected sometimes too
- Queries per second is hard to use
  - You have to compare this vs previous sample
- PROCESSLIST works sometimes
  - Too many queries with some status (grep -c)
- Text in SHOW INNODB STATUS (awk/grep)
- Other creative triggers...

What Value Should You Use?

```
$ mysqladmin ext -i1 | awk '
   /Queries/{q=$4-qp;qp=$4}
   /Threads_connected/{tc=$4}
   /Threads_running/{printf "%5d %5d %5d\n", q, tc, $4}'
2147483647   136     7
  798   136     7
  767   134     9
  828   134     7
  683   134     7
  784   135     7
  614   134     7
  108   134    24
  187   134    31
  179   134    28
 1179   134     7
 1151   134     7
 1240   135     7
 1000   135     7
```

Configuring pt-stalk

```
Threshold=100
# Collect GDB stacktraces?
collect-gdb=0

# Collect oprofile data?
collect-oprofile=0

# Collect strace data?
collect-strace=0

# Collect tcpdump data?
collect-tcpdump=0
```

Capturing Data

- pt-stalk stores data in /var/lib/pt-stalk
- There will be A LOT of data

Using pt-sift

Let me 'splain.

No, there is too much. Let me sum up.

Did I mention lots of data?

Terminal - baron@ginger:~/collected

```
2011_07_21_11_08_20-opentables1      2011_07_21_11_40_41-diskstats
2011_07_21_11_08_20-opentables2      2011_07_21_11_40_41-hostname
2011_07_21_11_08_20-output           2011_07_21_11_40_41-innodbstatus1
2011_07_21_11_08_20-pmap             2011_07_21_11_40_41-innodbstatus2
2011_07_21_11_08_20-processlist1     2011_07_21_11_40_41-interrupts
2011_07_21_11_08_20-processlist2     2011_07_21_11_40_41-iostat
2011_07_21_11_08_20-procstat         2011_07_21_11_40_41-iostat-overall
2011_07_21_11_08_20-procvmstat       2011_07_21_11_40_41-log_error
2011_07_21_11_08_20-ps               2011_07_21_11_40_41-lsof
2011_07_21_11_08_20-slabinfo         2011_07_21_11_40_41-meminfo
2011_07_21_11_08_20-stacktrace       2011_07_21_11_40_41-mpstat
2011_07_21_11_08_20-sysctl           2011_07_21_11_40_41-mpstat-overall
2011_07_21_11_08_20-top              2011_07_21_11_40_41-mutex-status1
2011_07_21_11_08_20-trigger          2011_07_21_11_40_41-mutex-status2
2011_07_21_11_08_20-variables        2011_07_21_11_40_41-mysqladmin
2011_07_21_11_08_20-vmstat           2011_07_21_11_40_41-netstat_s
2011_07_21_11_08_20-vmstat-overall   2011_07_21_11_40_41-opentables1
2011_07_21_11_10_31-df               2011_07_21_11_40_41-opentables2
2011_07_21_11_10_31-diskstats        2011_07_21_11_40_41-output
2011_07_21_11_10_31-hostname         2011_07_21_11_40_41-pmap
2011_07_21_11_10_31-innodbstatus1    2011_07_21_11_40_41-processlist1
2011_07_21_11_10_31-innodbstatus2    2011_07_21_11_40_41-processlist2
2011_07_21_11_10_31-interrupts       2011_07_21_11_40_41-procstat
2011_07_21_11_10_31-iostat           2011_07_21_11_40_41-procvmstat
```

Using pt-sift

```
Terminal - baron@ginger:~/collected
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$
[baron@ginger collected]$ pt-sift .

  2011_07_21_10_14_58  2011_07_21_10_16_49  2011_07_21_10_19_16
  2011_07_21_10_22_02  2011_07_21_10_31_55  2011_07_21_10_40_36
  2011_07_21_10_44_47  2011_07_21_10_54_48  2011_07_21_10_57_36
  2011_07_21_11_00_11  2011_07_21_11_01_54  2011_07_21_11_04_05
  2011_07_21_11_06_13  2011_07_21_11_08_20  2011_07_21_11_10_31
  2011_07_21_11_12_40  2011_07_21_11_14_51  2011_07_21_11_16_59
  2011_07_21_11_19_09  2011_07_21_11_21_16  2011_07_21_11_23_27
  2011_07_21_11_25_35  2011_07_21_11_27_44  2011_07_21_11_29_35
  2011_07_21_11_32_03  2011_07_21_11_34_12  2011_07_21_11_36_23
  2011_07_21_11_38_30  2011_07_21_11_40_41

Select a timestamp from the list [2011_07_21_11_40_41]
```

Using pt-sift

Terminal - baron@ginger:~/collected

```
--vmstat--
 r b  swpd   free   buff    cache si so bi    bo    in    cs us sy id wa st
42 0 15840 760604 142084 16869724  0  0 36   381     0     0  7  1 92  0  0
 1 0 15840 742016 142284 16911564  0  0 73 13053 31329 56995 11  3 86  0  0
wa 0% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
--innodb--
    txns: 10xACTIVE (1s) 310xnot (0s)
    0 queries inside InnoDB, 0 queries in queue
    Main thread: flushing buffer pool pages, pending reads 1, writes 6, flush 0
    Log: lsn = 1777474532609, chkp = 1776282083670, chkp age = 1192448939
    Threads are waiting at:
    Threads are waiting on:
--processlist--
    State
    494
     19  Sending data
      5  Has sent all binlog to slave; waiting for binlog to be updated
      3  Reading from net
      2  freeing items
    Command
    496  Sleep
     25  Query
      6  Connect
      5  Binlog Dump
```

Case Study: Foresaw Radish

- Response time increased 50x overnight
- “We didn't change anything”
- Throttled the workload to the server to 1/2
  - This made no difference

Using oprofile

```
samples  %        image name   symbol  name
893793   31.1273  /no-vmlinux  (no symbols)
325733   11.3440  mysqld       Query_cache::free_memory_block()
117732    4.1001  libc         (no symbols)
102349    3.5644  mysqld       my_hash_sort_bin
 76977    2.6808  mysqld       MYSQLparse()
 71599    2.4935  libpthread   pthread_mutex_trylock
 52203    1.8180  mysqld       read_view_open_now
 46516    1.6200  mysqld     Query_cache::invalidate_query_block_list()
 42153    1.4680  mysqld       Query_cache::write_result_data()
 37359    1.3011  mysqld       MYSQLlex()
 35917    1.2508  libpthread   __pthread_mutex_unlock_usercnt
 34248    1.1927  mysqld       __intel_new_memcpy
```

The Solution

- In a nutshell: nothing was wrong
  - Indexing, schema, queries, etc...
- The server really did just slow down overnight
- Perhaps data growth crossed a threshold?
- oprofile shows the query cache is the problem
- Disabling the query cache fixed the problem

Case Study: Chinese Dragon

- A more detailed example.
- Step 1: what's the problem?
  - “Once every day or so, we get errors about max_connections exceeded”
- Step 2: what's been done?
  - Nothing! (Thank heaven)

Assessing the Situation

- 16 CPU cores
- 12GB of RAM
- Solid-state drive
- Total of 900MB of data, all in InnoDB
- Linux
- MySQL 5.1.37
- InnoDB plugin version 1.0.4

Quick Server Check

- Under normal circumstances, nothing's wrong
- No bad query plans
- Queries all < 10ms

Diagnostic Trigger

- Threads_connected is normally 15 or less.
- It increases to hundreds during the problems
- This is the obvious trigger

Workload Summary

- From 1k to 10k QPS, mostly garbage queries
- Between 300 to 2000 SELECTs per second
- About 5 UPDATEs per second
- Very little of anything else

Workload Summary

- Two kinds of SELECTs in the processlist, with minor variations

```
$ grep State: processlist.txt | sort | uniq -c | sort -rn
    161   State: Copying to tmp table
    156   State: Sorting result
    136   State: statistics
     50   State: Sending data
     24   State: NULL
     13   State:
      7   State: freeing items
      7   State: cleaning up
      1   State: storing result in query cache
      1   State: end
```

Symptom or Cause?

Query Behavior

- Mostly index or range scans – no full scans
- Between 20-100 sorts per second, 1k-12k rows sorted per second
- No table locking or query cache problems

InnoDB Status

- Main thread state: flushing buffer pool pages
- Dirty pages: a few dozen
- Almost no change in Innodb_buffer_pool_pages_flushed
- Last checkpoint is very close to the LSN
- Buffer pool has lots of free space
- Most threads waiting in the queue: “12 queries inside InnoDB, 495 queries in queue.”

iostat (simplified)

r/s	w/s	rsec/s	wsec/s	avgqu-sz	await	svctm	%util
1.00	500.00	8.00	86216.00	5.05	11.95	0.59	29.40
0.00	451.00	0.00	206248.00	123.25	238.00	1.90	85.90
0.00	565.00	0.00	269792.00	143.80	245.43	1.77	100.00
0.00	649.00	0.00	309248.00	143.01	231.30	1.54	100.10
0.00	589.00	0.00	281784.00	142.58	232.15	1.70	100.00
0.00	384.00	0.00	162008.00	71.80	238.39	1.73	66.60
0.00	14.00	0.00	400.00	0.01	0.93	0.36	0.50
0.00	13.00	0.00	248.00	0.01	0.92	0.23	0.30
0.00	13.00	0.00	408.00	0.01	0.92	0.23	0.30

oprofile

```
samples   %        image name       symbol name
 473653   63.5323  no-vmlinux       /no-vmlinux
  95164   12.7646  mysqld           /usr/libexec/mysqld
  53107    7.1234  libc-2.10.1.so   memcpy
  13698    1.8373  ha_innodb.so     build_template()
  13059    1.7516  ha_innodb.so     btr_search_guess_on_hash
  11724    1.5726  ha_innodb.so     row_sel_store_mysql_rec
   8872    1.1900  ha_innodb.so rec_init_offsets_comp_ordinary
   7577    1.0163  ha_innodb.so     row_search_for_mysql
   6030    0.8088  ha_innodb.so     rec_get_offsets_func
   5268    0.7066  ha_innodb.so     cmp_dtuple_rec_with_match
```

Poor Man's Profiler

```
$ pt-pmp -l 5 stacktraces.txt
    507 pthread_cond_wait,one_thread_per_connection_end...
    398 ...,srv_conc_enter_innodb,innodb_srv_conc_enter_innodb...
     83 ...,sync_array_wait_event,mutex_spin_wait,mutex_enter_func
     10 ....
```

What Next?

- Ask two questions:
  - "Which theories seem reasonable?"
  - "What is unusual/wrong/unexplainable?"

What's Wrong?

- How can a 900MB database, with only 5 UPDATEs per second, be writing 150MB of data to disk per second?

r/s	w/s	rsec/s	wsec/s	avgqu-sz	await	svctm	%util
1.00	500.00	8.00	86216.00	5.05	11.95	0.59	29.40
0.00	451.00	0.00	206248.00	123.25	238.00	1.90	85.90
0.00	565.00	0.00	269792.00	143.80	245.43	1.77	100.00
0.00	649.00	0.00	309248.00	143.01	231.30	1.54	100.10
0.00	589.00	0.00	281784.00	142.58	232.15	1.70	100.00
0.00	384.00	0.00	162008.00	71.80	238.39	1.73	66.60
0.00	14.00	0.00	400.00	0.01	0.93	0.36	0.50
0.00	13.00	0.00	248.00	0.01	0.92	0.23	0.30
0.00	13.00	0.00	408.00	0.01	0.92	0.23	0.30

Theory: Flush Stall

- InnoDB could be experiencing “furious flushing”
- Could this cause lots of I/O and block threads out of the kernel?
- No. InnoDB has only a few dirty pages, they don't decrease over time, and there isn't that much data.
- The flushing is stalled because I/O is starved, not the reverse. InnoDB is a victim, not the perpetrator.

Who's To Blame?

- Is the disk too slow... or are we just over burdening it?

```
 r/s    w/s rsec/s    wsec/s avgqu-sz  await svctm  %util
1.00 500.00   8.00  86216.00     5.05  11.95  0.59  29.40
0.00 451.00   0.00 206248.00   123.25 238.00  1.90  85.90
0.00 565.00   0.00 269792.00   143.80 245.43  1.77 100.00
0.00 649.00   0.00 309248.00   143.01 231.30  1.54 100.10
0.00 589.00   0.00 281784.00   142.58 232.15  1.70 100.00
0.00 384.00   0.00 162008.00    71.80 238.39  1.73  66.60
0.00  14.00   0.00    400.00     0.01   0.93  0.36   0.50
0.00  13.00   0.00    248.00     0.01   0.92  0.23   0.30
0.00  13.00   0.00    408.00     0.01   0.92  0.23   0.30
```

- Use your knowledge of disks to figure out whether it is performing up to specs.

Are We Abusing the Disk?

- Is MySQL to blame, or is something else happening?
- Is iostat trustworthy?
- What does MySQL write to disk?
  - Data
  - Logs
  - Temp Tables
  - Sort Files
  - Other things are uncommon

Theory: Sorts or Temp Tables

- Is that the source of excessive I/O?
- Hard / impossible to measure directly on this version of Linux
- But we can measure sort/temp file sizes easily
- Use lsof and df

Samples of “df -h”

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda3	58G	20G	36G	36%	/
/dev/sda3	58G	20G	36G	36%	/
/dev/sda3	58G	19G	36G	35%	/
/dev/sda3	58G	19G	36G	35%	/
/dev/sda3	58G	19G	36G	35%	/
/dev/sda3	58G	19G	36G	35%	/
/dev/sda3	58G	18G	37G	33%	/
/dev/sda3	58G	18G	37G	33%	/
/dev/sda3	58G	18G	37G	33%	/

Samples of “lsof”

```
$ awk '
  /mysqld.*tmp/ {
    total += $7;
  }
  /^Sun Mar 28/ && total {
    printf "%s %7.2f MB\n", $4, total/1024/1024;
    total = 0;
  }' lsof.txt
18:34:38 1655.21 MB
18:34:43    1.88 MB
18:34:48    1.88 MB
18:34:53    1.88 MB
```

Looks Like Temp Tables

- The queries in the processlist have filesorts and temporary tables
- It looks like there is a storm of them
- The cheap Intel SSD can't sustain that much writing (our benchmarks show that)
- All of the temp table writes queue, and the system becomes I/O bound

The Solution

- Two possible solutions:
  - Modify queries so they don't create temp tables
  - Figure out why there is a flood of such queries
- The ultimate answer:
  - Memcached expirations were handled very badly
  - There was a rush of queries to regenerate missing cache items

To Sum Up

Summary

- Understand performance
- Understand the goal
- Understand how the system works
- Measure, using any means necessary
  - Use good tools
- Distinguish between causes and effects
- The problem is usually obvious
- The solution is also usually obvious

Practice the disciplined, boring approach unless you are sure it is okay to break the rules.

I almost always tell myself “wait, stop, back to basics” when I am troubleshooting, even if I have seen something before.

And when I report on the outcome, or make a recommendation, I say whether I am certain.

Resources

- *High Performance MySQL 3rd Edition*
- *Optimizing Oracle Performance* by Cary Millsap
- *MySQL Troubleshooting* by Svetlana Smirnova
- *Goal-Driven Performance Optimization*
  - White paper available from Percona's website
- Percona Toolkit
  - pt-stalk
  - pt-sift