

MySQL Performance Tuning

A practical guide

A large, faint, light gray outline of the MySQL fish logo is positioned on the left side of the slide.

Oli Sennhauser
Senior Consultant
osennhauser@mysql.com

Introduction

- Who we are?
- What we want?

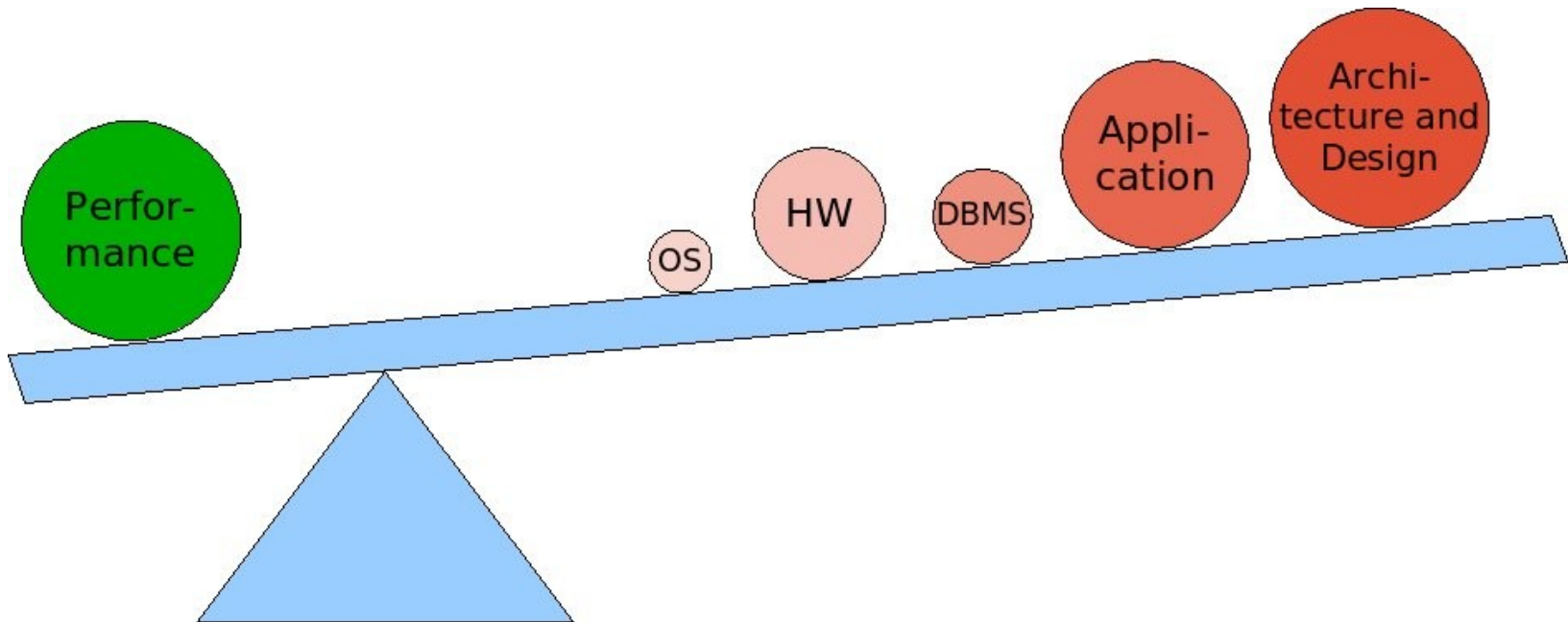
Table of Contents

- Find the problem
- MySQL architecture
- Database settings
- Detect and eliminate slow queries
- Table tuning
- Application tuning
- Alternatives
- Prevention
- Dirty tricks and other stuff
- Now it's up to you...

DBA: We have a problem!

- What does performance mean to you?
- How does it look like?
 - DB is (suddenly!?) slow.
 - No historical data (or not the one we need).
 - “Screw something on the DB!”
 - We are short before going live and much too slow!!!
- We have a problem. And what now?

Efficiency of tuning measurements



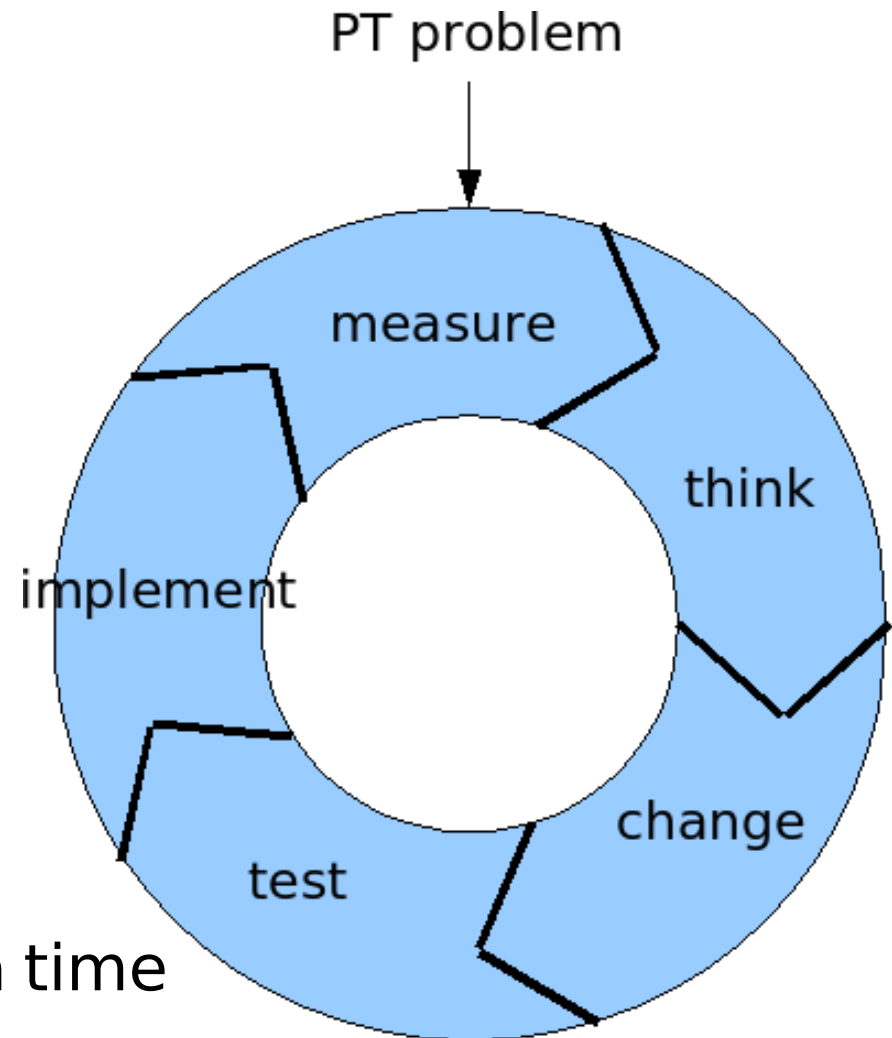
- Application/Architecture/Design
 - No way! For what ever reason :-)
- So on the DBA side: Memory, DB settings, I/O, Indexes, etc.

Find the problem / the bottleneck

- No history data!?! :-(
- Best if:
 - you can simulate it
 - it happens predictable and/or periodically
- Your friends are:
 - vmstat / dstat
 - iostat
 - top
 - any graphical history of values

Tuning means ...

- The tuning life cycle:

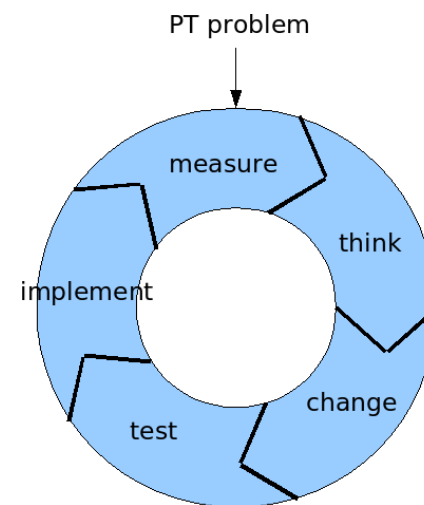


- Only one change at a time

Measure

- Find the bottleneck / limiting resource:
 - I/O
 - Memory
 - CPU
 - Network bandwidth

- But how?



Measure I/O

- vmstat

```
# vmstat 1
procs ---swap-- ----io---- ----cpu----
 r  b   si   so   bi    bo  us  sy  id  wa
 0  0    3    3   94   143 21  21 56  2
 0  0    0    0    0     4  9  37 54  0
```

- iostat (--> sysstat package)

```
# iostat -x 1
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5.88    0.00   34.31    2.94    0.00   56.86

Device:            r/s    w/s  kB/s  kB/s  await  svctm  %util
hda                 0.00   0.00   0.00   0.00   0.00   0.00   0.00
hdc                 0.00   2.94   0.00  23.53  14.67  12.00   3.53
```

Measure memory

- ps

```
# ps -eo user,pid,%cpu,%mem,vsz,rsz,comm --sort -vsz | \
  egrep 'mysql|COMMAND'
```

USER	PID	%CPU	%MEM	VSZ	RSZ	COMMAND
mysql	1361	0.0	1.5	108368	16444	mysqld
mysql	1210	0.0	0.1	4536	1956	bash
mysql	1289	0.0	0.1	4060	1444	safe_mysqld
mysql	1204	0.0	0.1	4048	1404	su

- free / top:

```
#free
```

	total	used	free	shared	buffers	cached
Mem:	1036016	983864	52152	0	35484	547432
-/+ buffers/cache:	400948	635068				
swap:	4202112	96148	4105964			

Measure CPU

- top

```
Cpu0  :  7.1%us, 12.8%sy,  0.0%ni, 71.4%id,  1.5%wa,  0.0%hi,  7.2%si,  0.0%st
Cpu1  : 16.5%us,  3.4%sy,  0.0%ni, 79.4%id,  0.0%wa,  0.0%hi,  0.7%si,  0.0%st
Cpu2  : 99.8%us,  0.1%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.1%si,  0.0%st
Cpu3  :  8.5%us,  2.3%sy,  0.0%ni, 58.5%id, 28.2%wa,  2.3%hi,  0.2%si,  0.0%st
```

- vmstat

```
# vmstat 1
procs -----memory----- --swap--  ----io----  -system--  ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo    in   cs  us  sy  id  wa
 1  0   96148 56096 35936 548792    0    0     0   656  379  343  5 38 57  0
 0  0   96148 56096 35936 548792    0    0     0     0  260  357  5 34 61  0
 0  0   96148 56096 35936 548792    0    0     0     0  306  399  9 29 62  0
 3  0   96148 49192 35940 549808    0    0  1020     0  289  431 91  4  3  2
 1  0   96148 47424 35944 551572    0    0   896     0  310  378 98  2  0  0
 1  0   96148 45656 35944 553344    0    0   896     0  260  359 98  1  0  1
 2  0   96148 43948 35944 555112    0    0   896     0  280  355 97  3  0  0
 1  0   96148 42056 35952 556884    0    0   904     0  260  374 99  0  0  1
 1  0   96148 40288 35984 558672    0    0   896  3772  312  398 97  3  0  0
 1  0   96148 38520 35984 560424    0    0   896     0  259  365 97  1  0  2
```

- dstat

```
# dstat
----total-cpu-usage----  -dsk/total-  -net/total-  ---paging--  ---system--
usr  sys  idl  wai  hiq  sig | read  writ | recv  send | in   out | int  csw
 21   6   56   2   0   14 | 25k   39k |    0    0 | 764B  880B | 129  762
  9   2   55   0   0   34 |    0    0 | 262B 1680B |    0    0 | 297  374
  6   2   59   0   0   33 |    0    0 |1075B 1467B |    0    0 | 284  372
  8   3   54   5   1   29 |    0  208k |1046B  884B |    0    0 | 309  377
 14   2   54   0   1   29 |    0  236k |3479B 3669B |    0    0 | 333  362
 18   5   47   1   0   29 |    0  164k |2800B 3632B |    0    0 | 351 2257
 30  69   0   0   0   1 |    0    0 |1807B 1181B |    0    0 | 651 243k
 24  74   2   0   0   0 |    0    0 |2380B 2183B |    0    0 | 685 240k
```

Measure network bandwidth

- dstat

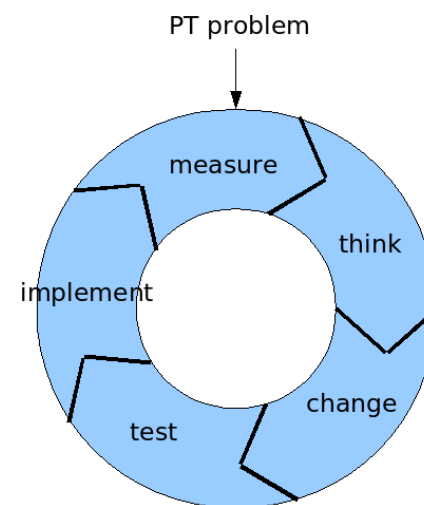
```
# dstat
----total-cpu-usage---- -dsk/total- -net/total-
usr sys idl wai hiq siq| read writ| recv send
 21  5  56  2  0 15| 25k  39k|  0    0
 13  3  84  0  0  0|  0    0 | 994B 437B
  8  4  88  0  0  0|  0    0 | 632B 484B
```

- ifconfig

```
# watch -n 1 -d "/sbin/ifconfig | egrep 'Link|bytes'"
eth0      Link encap:Ethernet  HWaddr 00:30:1B:2D:67:B4
          RX bytes:1751779749 (1670.6 Mb)
          TX bytes:191340381 (182.4 Mb)
```

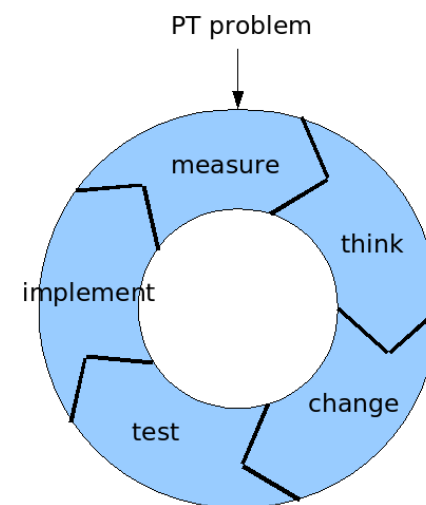
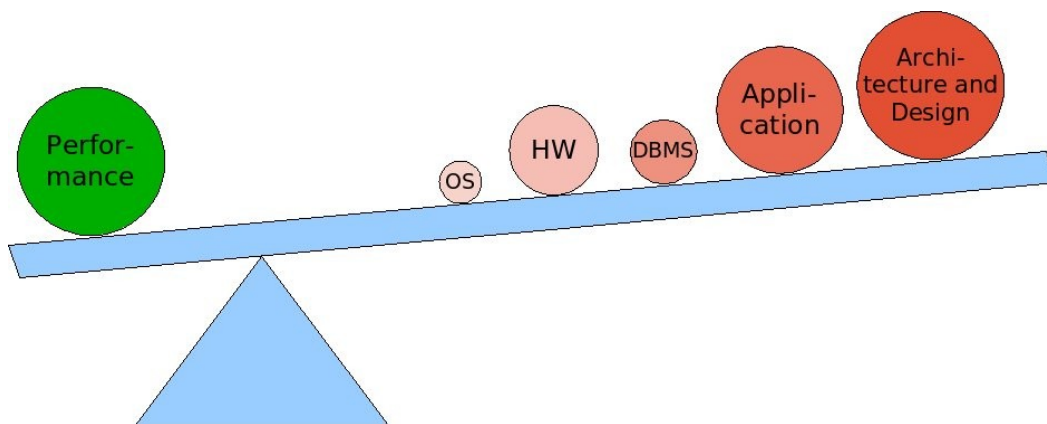
Think

- I/O
 - Who does it?
 - Is it read or write?
 - Is it random I/O or sequential I/O?
- Memory
 - Easy to find!
 - DB sizing
 - Is it somebody else?
- CPU
 - Easy to find!
 - Who is “burning” CPU?
- Network bandwidth
 - Who does it?
 - Sniff traffic?



Change

- What could be changed?
- Hardware -> I/O system (RAID5), RAM, CPU, NW
- O/S -> do not touch (kernel upgrade)
- DB -> my.cnf
- Application -> Queries!!!



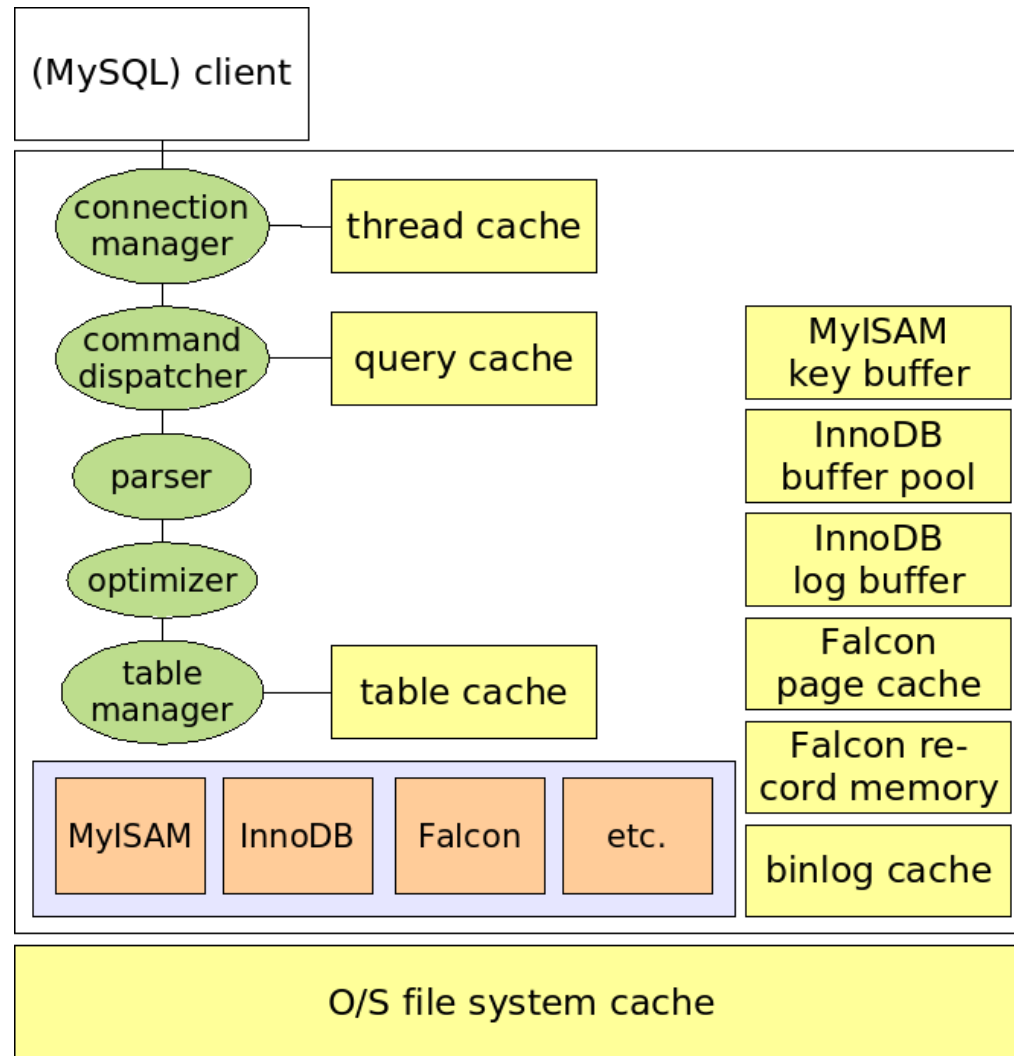
Change Hardware

- More RAM helps more!!!
- Faster CPU if it is the bottleneck (not more!)
- More expensive I/O system:
 - RAID5 is bad for databases!!!
 - RAID10 is good.
 - Many spindles
 - Battery buffered I/O system cache???
- 1 Gbit Network?
- Forget about virtualization (VMware etc.)!!!

Change O/S

- Use mainstream O/S -> for MySQL this means SLES/RHEL!
 - Use 64-bit architecture (> 4 GB RAM)
 - Use recent kernel (>= 2.6.12)
 - Use mainstream file system -> ext3 and xfs
 - Take what you are most familiar with!
- > But on O/S you cannot change much. They are already optimal! :-)

Change MySQL: Architecture



Change MySQL: Performance Features

- The magic of caching: “Do as little work as possible: Be lazy!!!”
- Performance features:
 - Thread cache
 - Query cache
 - Prepared statements
 - Stored Procedures (see “the SP trap!”)
 - delayed INSERT (MyISAM only)

Change MySQL: database settings

- “The big 3!”
 - key_buffer_size
 - innodb_buffer_pool_size
 - innodb_log_file_size
- Some others: query_cache_size, thread_cache_size
- My approach:
 - use defaults (or templates)
 - add: “the big 3” + 2 (see above)
 - do NOT change except it was proved and measured that is useful!

Change MySQL

- Where to change?
 - my.cnf (Caution: many possible places!!!)
- Where to measure?
 - `SHOW /*!50000 GLOBAL */ STATUS;`
- Where to cheat?
 - <http://dev.mysql.com/doc/refman/5.0/en/index.html>
 - 5.2.3. System Variables
 - 5.2.5. Status Variables

The big 3

- MyISAM

```

key_buffer_size           = 25-33% of RAM

Key_blocks_unused        --> actual value
Key_blocks_used          --> high water mark
Key_read_requests / Key_reads --> >= 99% ideally
    
```

- InnoDB

```

innodb_buffer_pool_size      = 80% of RAM

Innodb_buffer_pool_pages_free
Innodb_buffer_pool_read_requests /
Innodb_buffer_pool_reads    --> >= 99% ideally
    
```

The big 3

- InnoDB

```
innodb_log_file_size      = 32 - 128 Mbyte
```

```
Innodb_os_log_pending_fsyncs --> ???
```

```
--> hiccups!
```

Query cache & thread cache

- Query cache

```
query_cache_size          = 32 - 128 Mbyte (caution: 512!)
```

```
Qcache_total_blocks
```

```
Qcache_free_blocks
```

```
Qcache_free_memory      --> Fragmentation
```

```
Qcache_hits
```

```
Qcache_inserts          --> Hit ratio, ideally >> 2 : 1
```

```
Qcache_lowmem_prunes    --> too small or too fragmented
```

- Thread cache

```
thread_cache_size       = 8 - 128
```

```
Threads_cached
```

```
Threads_created        --> should not grow much over time
```

Some more...

- That's it! :-)
- Avoid any kind of I/O: logging!

```
sync_binlog      --> 0 !!!  
#log            --> Not on production!!!  
#log_bin        --> Where we do NOT need it!!!  
log_slow_queries --> is OK, we do not have such :-)
```

- Try to avoid sync writing:

```
innodb_flush_log_at_trx_commit = 2
```

→ Simulates MyISAM behaviour for InnoDB. But caution!

Some more...

- Table cache

```
table_cache      = 64 - 2048
```

```
Open_tables      --> <= table_cache
```

```
Opened_tables    --> should change moderately
```

- Other InnoDB settings:

```
innodb_additional_mem_pool_size
```

→ Do NOT change this! > 20 Mbyte is non sense!

```
innodb_flush_method
```

→ Sometimes O_DIRECT or O_DSYNC can help. But test before!

Change Application!

- Transaction log and binlog cache:

```
Binlog_cache_disk_use  --> increase binlog_cache_size
Innodb_log_waits      --> increase innodb_log_buffer_size
```

→ Too big transactions???

- Temporary results:

```
max_heap_table_size    = 16 - 256 Mbyte
tmp_table_size         = 32 - 512 Mbyte

Created_tmp_disk_tables --> changes often
```

→ Too big temporary results?

Change Application!

- Sort buffer:

```
sort_buffer_size    = 2 - 16 Mbyte  
Sort_merge_passes  --> sort buffer too small
```

→ Too big sorts???

- Application or Network problems:

```
Aborted_clients  
Aborted_connects
```

- Network traffic:

```
Bytes_received  
Bytes_sent
```

Change Application!

- Locking:

```
Table_locks_immediate
Table_locks_waited
```

- Too high concurrency or too slow queries! -> Optimize queries or try InnoDB.

```
InnoDB_row_lock_current_waits
InnoDB_row_lock_time
InnoDB_row_lock_time_avg
InnoDB_row_lock_time_max
InnoDB_row_lock_waits
```

- InnoDB locking! Optimize queries or think about changing the application.

Change Application!

- Missing Indexes:

```
Select_full_join  
Select_range_check --> should both be zero!!!
```

→ Missing Index!

- Full-Table-Scan:

```
Select_scan  
Handler_read_rnd_next  
Sort_scan
```

→ Find the queries! :-)

Find the slow queries!

- Quick:

```
SHOW [FULL] PROCESSLIST;
```

- Proper: Enable the slow query log!

```
# my.cnf

log_slow_queries          = slow_query.log
long_query_time           = 1
log_queries_not_using_indexes = 1
```

→ And now??? Thousands of queries!!!

Find the slow queries!

- Profile the slow query log:

```
# mysqldumpslow -s t slow-query.log > slow_query.profile
```

- That's how the profile looks like:

```
Count: 4498 Time=212.72s (956824s) Lock=0.04s (198s) Rows=0.0 (0)
create table TMP.SS_temp2_36 select l.initlot,s.lot,s.wafer,s.x,s.y,

Count: 810 Time=121.74s (98610s) Lock=0.30s (245s) Rows=0.0 (0)
insert into TOD.row_descr select l.initlot,w.lot,w.wafer,'S' dataset,'S'

Count: 477 Time=149.99s (71547s) Lock=0.01s (4s) Rows=2.7 (1284)
SELECT l.lot,count(h.MFG_STEP_NAME) cnt FROM DB1.lot_7000 l left join

Count: 92 Time=573.43s (52756s) Lock=0.00s (0s) Rows=325.6 (29958)
SELECT ps.X, ps.Y, SUM(N*ps.PARVALUE)/COUNT(ps.PARVALUE) PARMEAN FROM
```

→ Start working now! EXPLAIN ...

MySQL EXPLAIN

- Generate an execution plan:

```
EXPLAIN
SELECT i.number, l.answer
  FROM poll_item i
  JOIN poll_item_l l ON (l.poll_id = i.poll_id
                        AND l.number = i.number)
WHERE i.poll_id = '4'
      AND l.language_id = '2'
ORDER BY i.number ASC;
```

id	select_type	table	type	pos_keys	key	k_len	ref	rows	Extra
1	SIMPLE	i	ref	PRIMARY	PRIMARY	2	const	5	Using where; Using index
1	SIMPLE	l	eq_ref	PRIMARY	PRIMARY	5	const,...	1	Using where

- Rewrite DML into SELECT.
- Be cautious with Subqueries! They are executed!

MySQL visual explain

- <http://mysqltoolkit.sourceforge.net/>

```

./mysql-visual-explain test.exp

JOIN
+- Filter with WHERE
| +- Bookmark lookup
|   +- Table
|     | table          1
|     | possible_keys PRIMARY
|     +- Unique index lookup
|       key            1->PRIMARY
|       possible_keys PRIMARY
|       key_len        5
|       ref            const, topodb.i.number, const
|       rows           1
+- Filter with WHERE
  +- Index lookup
    key            i->PRIMARY
    possible_keys PRIMARY
    key_len        2
    ref            const
    rows           5

```

Table tuning

- Indexing
 - See above.
 - What should be indexed and how?
- Data type tuning
 - `mysqldump -all-databases --no-data`
- Table design

Table tuning – Indexing

- What should be indexed?
 - All attributes where you JOIN
 - All attributes where you filter (WHERE)
 - All attributes where you ORDER or GROUP BY
 - All attributes where you want to do an Index Scan instead of a Table scan.
 - NOT on attributes with an evenly distributed low cardinality.
- How should be indexed?
 - Indexes can only be used from left to right.
 - Keep them short.
 - Compound indexes: INDEX(a, b).
 - Prefixed indexes: INDEX(a, b(10)).
 - Do not function-cover indexed attributes

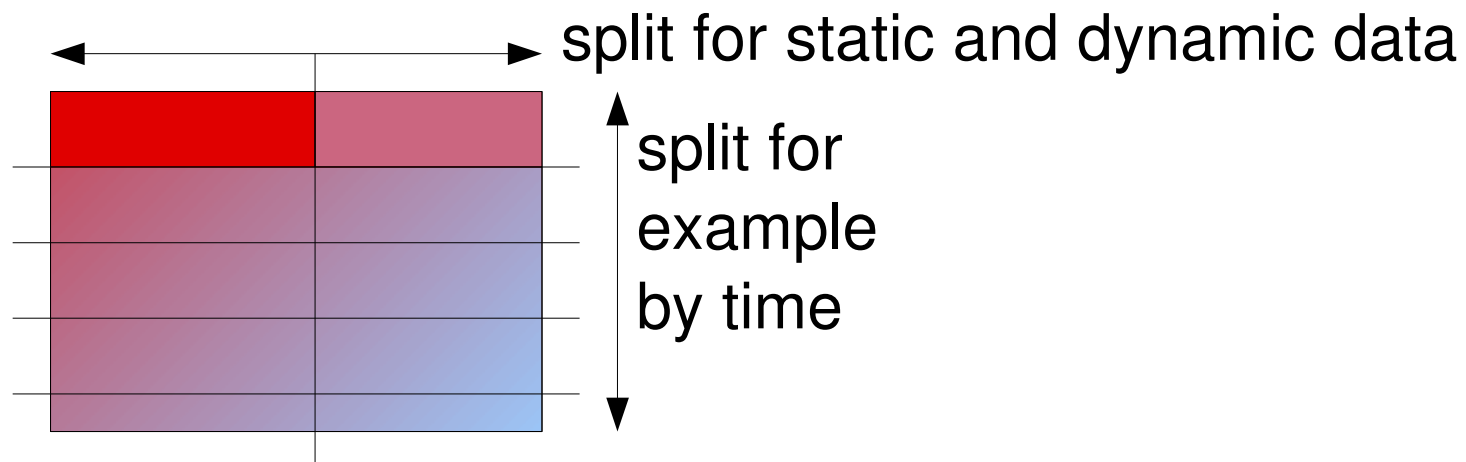
Table tuning – data type tuning

- Idea behind: Increase the data density!
- How to get: mysqldump --no-data

```
CREATE TABLE betatesters (
  user_id bigint(20) NOT NULL,
  nick varchar(255) NOT NULL,
  registerdate varchar(30) NOT NULL,
  daysregistered int(11) NOT NULL,
  value double default NULL,
  timestamp_data bigint(15) default NULL,
  ip varchar(16) default NULL
  PRIMARY KEY (`nick`),
  UNIQUE KEY user_id (`user_id`)
  KEY (`user_id`, `nick`)
) DEFAULT CHARSET=utf8;
```

Table tuning – table design

- Normalization versus de-normalization
 - Joins are expensive --> CPU
 - Denormalized is big --> high redundancy --> RAM --> Disk --> Slow
 - Find the trade-off!
 - Bring everything in 3rd NF --> then start denormalizing if necessary.
- vertical and horizontal partitioning:



Locality of Reference

- In theory: We should not care how data are stored internally.
- In practice: It is sometimes good to know!
- Why?
- 2 examples from the last 9 months:
 - wind mills
 - vehicle tracking for parcel delivery

Example 1

- Several 100 wind mills
- 50 measured values per wind mill
- Every 5-15 minutes
- Up to 10 years
- Dozens of GB of data
- Record size up to 2k!

- Search pattern: Give me value x from wind mill #13 in this time range!



Example 2

- Several 100 vehicles
- 24 h/d
- Every 2 min position
- Status/position per vehicle, later per parcel!!!
- Dozens of GB of data
- Record size 400 bytes
- Search pattern: Give me all positions of vehicle #13 from the last 24 hours.

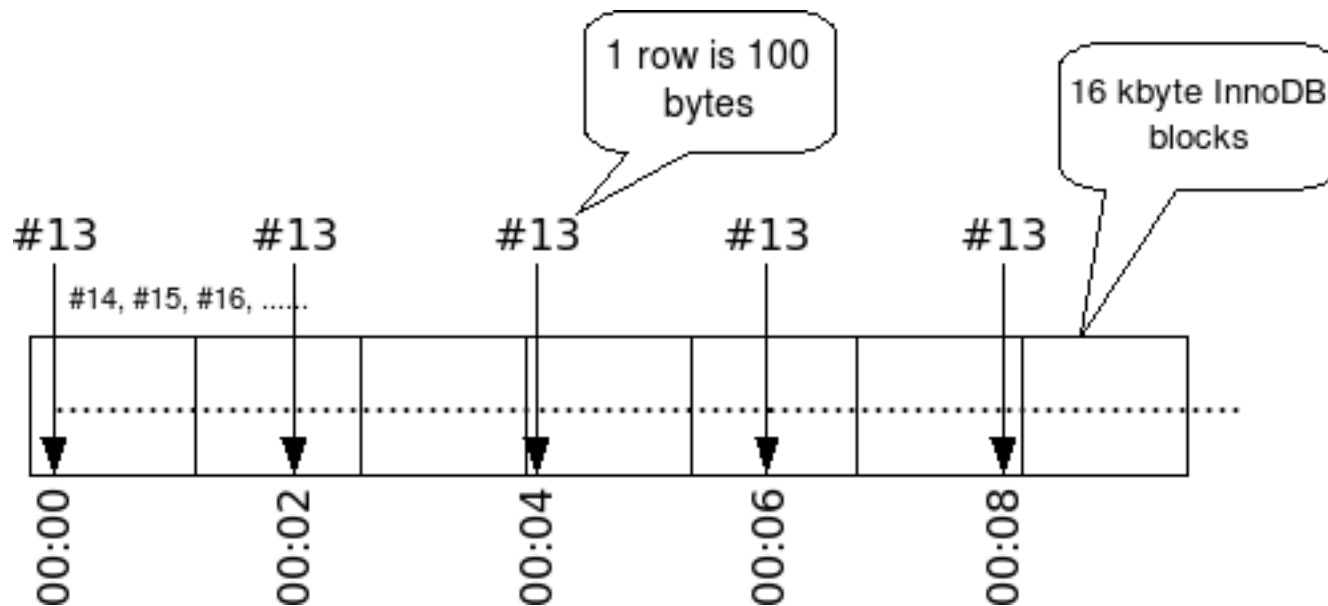


Locality of Reference

- These 2 examples have one behaviour in common:
- Delivery of data is completely different than search pattern.
 - Usually data are delivered sorted by time and also (more or less) retrieved by time.
 - In this cases time has a secondary influence!
- But what happens???

Locality of Reference

- Block size is 16k/4k
- PK is AUTO_INCREMENT



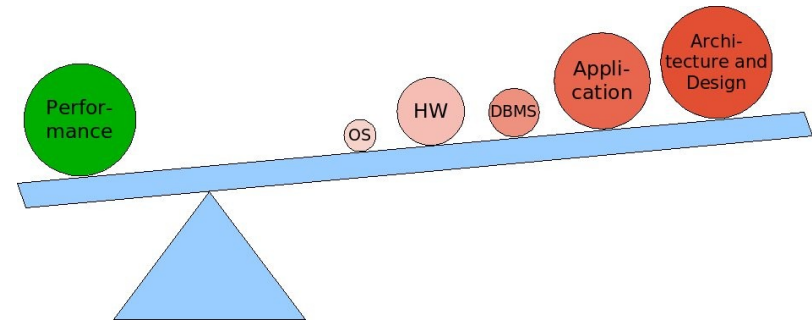
- Synthetical PK are sometimes dangerous!

Locality of Reference

- What to do???
- PK on (vehicle_id, ts) for example or
- PK on (windmill_id, data, ts)
- Can be up to 100 times more efficient (not necessarily faster)
- What about MyISAM?
- What about Falcon? (Mail from Ann can be provided).

Change Application

- Where are we now?

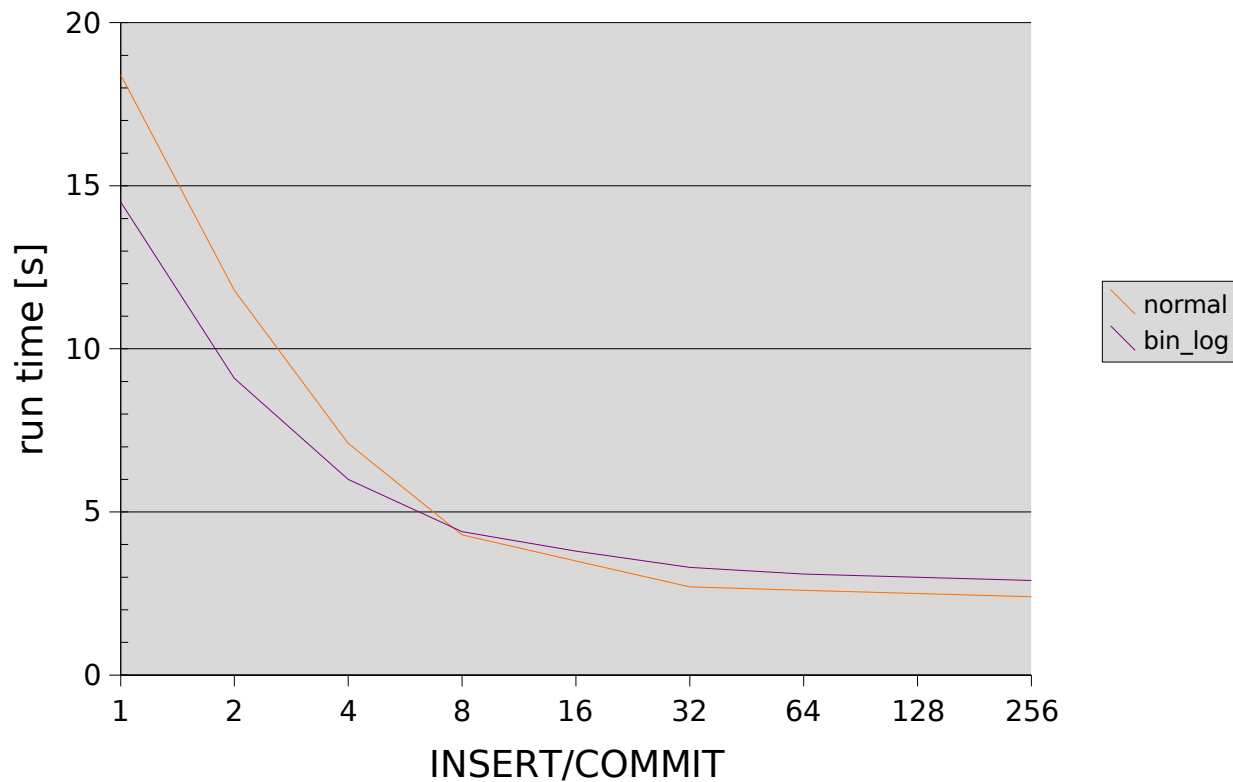


- What else can we do?
 - Avoid – reduce – optimize
- Do not!
 - Put more intelligence into your application!
- Reduce!
 - Do only once. Cache!
- Do it better!
 - Tune the statement, tune the code, tune the logic!

Change Application

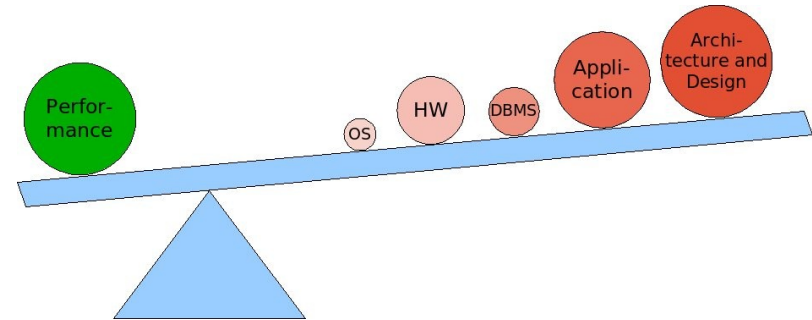
- Clean up first, before you invest into new hardware or even a redesign.
 - New hardware brings maybe a factor of 2x
 - Clean up can bring factors up to 10x
 - Sometimes new hardware is cheaper :-)
- Read issues are a caching problem.
 - Try to cache!
- Write issues are a batching problem.
 - Try to batch!

commit_demo.pl



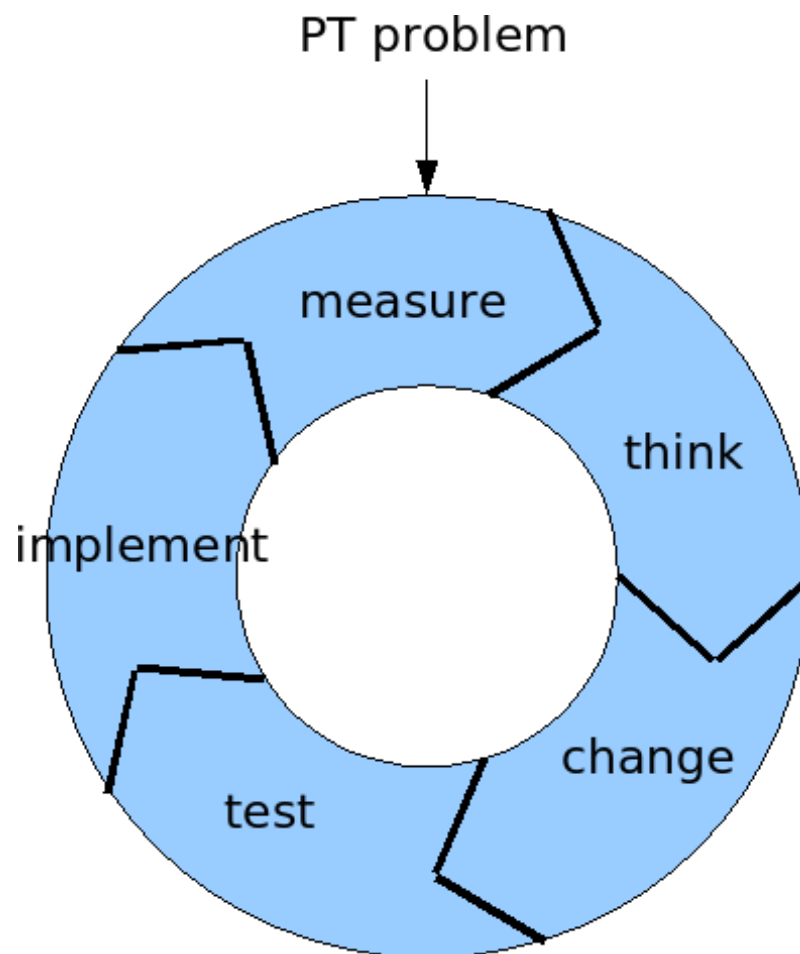
Alternatives when exhausted

- See this afternoon! :-)
- Change architecture.
 - Scale-Out?
- Tricks like Materialized Views?
- Application partitioning



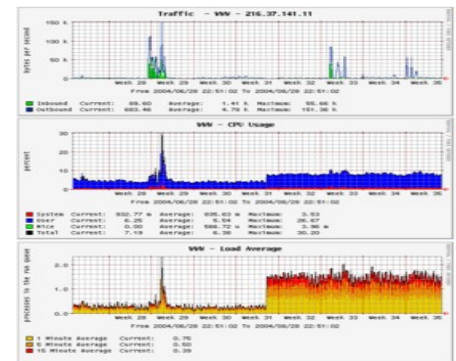
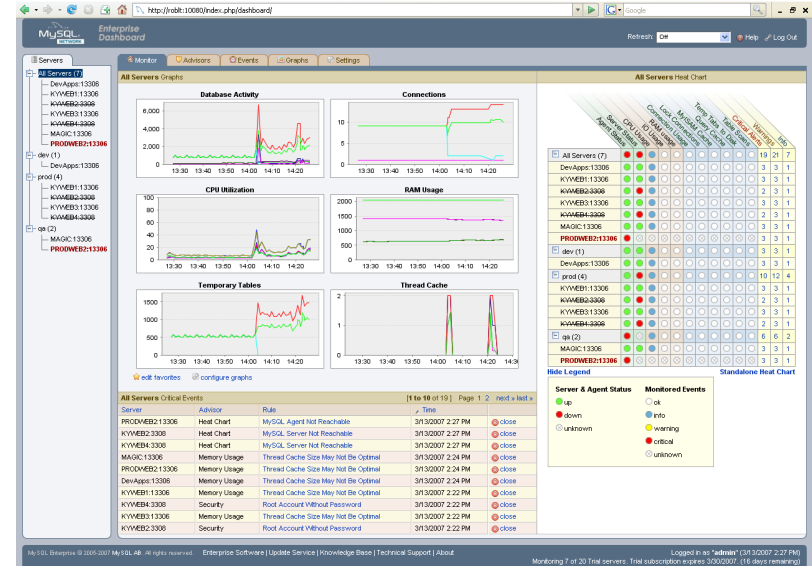
Prevention

- What can we do to prevent Performance problems?
 - Do load testing.
 - Do benchmarking.
 - Collect historical data and make predictions.
- An then: Measure and monitor...



Measure

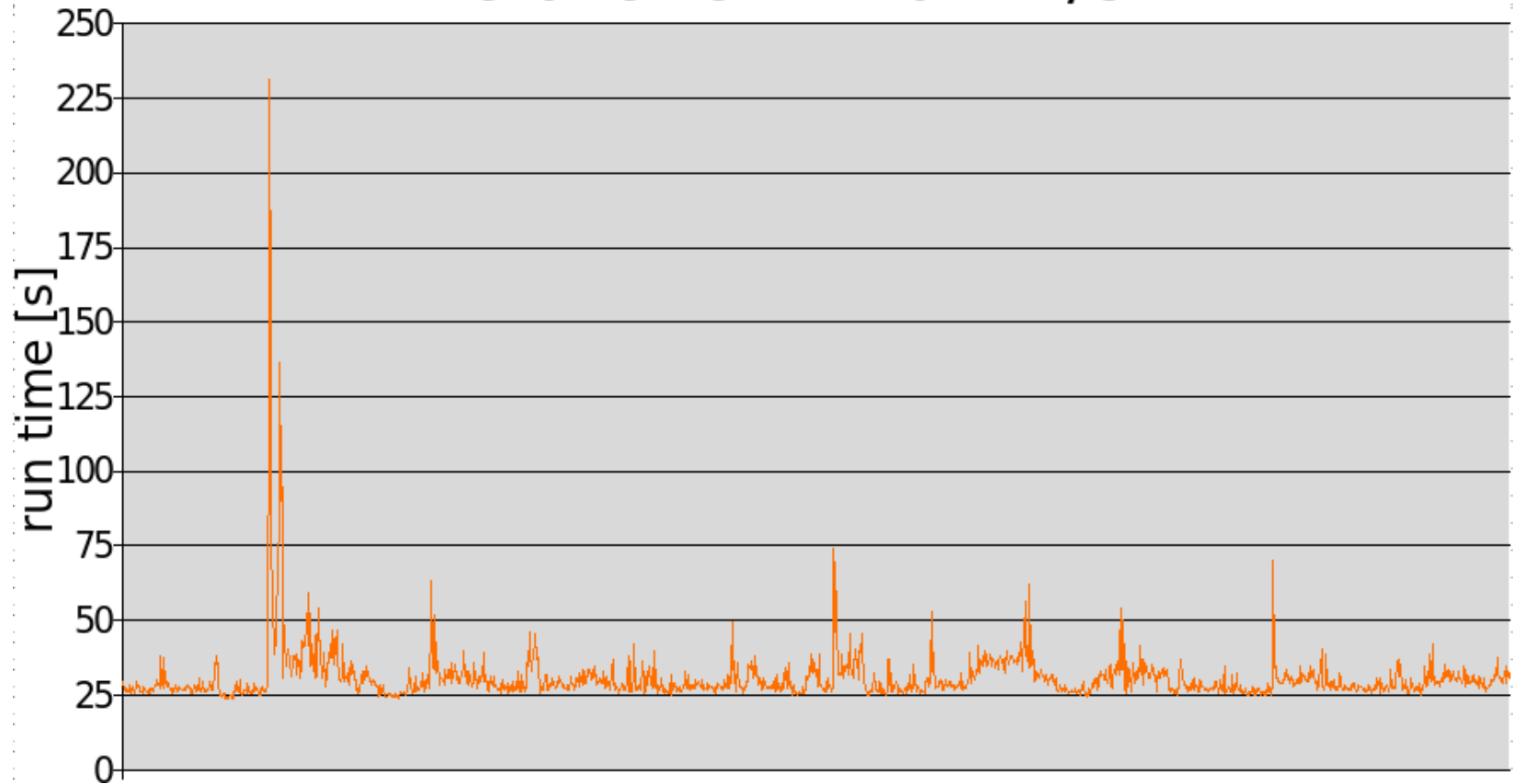
- top, vmstat, iostat, dstat, mstat, free, ...
- mytop, innotop, mtop
- Nagios, MySQL AR, MySQL Administrator, Cacti, MRTG, RRD, Munin, Moodds, Big Sister, MySQLStat, Zabbix, Hobbit, Monit, ...



http://www.shinguz.ch/MySQL/mysql_monitoring.html

Virtualization VM /SAN

trx time over 8 h in a VM/SAN



RAM disks (I)

- ORDER BY, GROUP BY, DISTINCT --> temp tables

- bigger than:

```
tmp_table_size           = 32M
max_heap_table_size     = 16M
```

- BLOB/TEXT

- Will be written into:

```
tmpdir                   = /tmp/
```

- Can be seen in:

```
Created_tmp_disk_tables  0
Created_tmp_tables       20
```

RAM disk (II)

- Both counters are increased!
 - Solutions?
 - Change your statement/requirements
 - Optimize your Query
 - Reduce size of result set
 - Avoid BLOB/TEXT
 - And if you cannot?
- > Use a RAM disk!

RAM disk (III)

- RAM disk is a disk in RAM :-) --> So you need much RAM (8 Gbyte on 32-bit systems?)!
- Can use your SWAP (we do not want that)!
- More info:
[/usr/src/linux/Documentation/filesystems](#)

```
# cat /proc/filesystems
# mount tmpfs -t tmpfs /mnt -o size=100m
# mount
```

- Bug in 5.0.4x!!! :-(

Now it's up to you...

- Output of: `SHOW /*!50000 GLOBAL */ STATUS;`
- Output of: `SHOW GLOBAL VARIABLES;`
- Slow query log.
- Slow queries
- Execution plans (`EXPLAIN SELECT ...`)
- Output of "vmstat 1" during peak time.