



# Learning from others' MySQL Performance Mistakes

**Matt Yonkovit**
**Sun/MySQL 2009**

# The Same Old Thing

- We tend to get the same issues over and over again
- Customers face similar challenges
- Some fixes are very easy and we need to promote common knowledge

# Lets be honest though

- Sometimes applications are like Forest Gump...
- They start off small and unassuming, 1 server and 2 guys workings hard , people laugh at them...
- Then overnight they become really successful and busy
- What worked before doesn't work you need more shrimp boats!!!!

# Top 10 Repeats

1. More is not always better
2. Text fields can be bad
3. Size ( data size ) does matter
4. Type Comparison

- Left Outer joins
- The Solo Index
- Always an IO Problem
- Filesystem and Lun Layout
- No Black Box
- Self-induced fragmentation

# More is not always better

- People go hog wild on per thread buffers like read_buffer, read_rnd_buffer, join_buffer, and sort_buffer.
- Try to reserve a minimum of 4GB for the OS and per thread buffers, more if you know you have a lot of threads ( 8GB ).
- Setting these too high can cause bad things to happen.  Too many times I see systems swapping.
- Interesting memory issues when buffers are set to high

# Malloc -vs- mmap

- Back in 2007, we ran into a performance issue when the read_buffer was set to more then 256K. Internally Malloc is used to allocate memory when the read_buffer is set to less then 256K, when this is set higher mmap is used to allocate memory. What's the difference?

From Monty's blog he put together a quick program to test how long it takes for malloc to allocate memory ( total for 100K ) :

Time for 128k: 0.035105

Time for 256k: 0.028631

Time for 1024k: 0.942634

Time for 5120k: 0.985856

# Monty's oprofile

OPROFILE

CPU with 5MB read_buffer

561612 25.0417 /lib64/tls/libc-2.3.4.so memset

429457 19.1491 /usr/lib/debug/lib/modules/2.6.9-34.ELsmp/vmlinux clear_page

214268 9.5540 /usr/lib/debug/lib/modules/2.6.9-34.ELsmp/vmlinux do_page_fault

144293 6.4339 /usr/lib/debug/lib/modules/2.6.9-34.ELsmp/vmlinux do_no_page

94410 4.2097 /usr/lib/debug/lib/modules/2.6.9-34.ELsmp/vmlinux buffered_rmqueue

64998 2.8982 /lib64/tls/libc-2.3.4.so memcpy

CPU with 128K read_buffer

2220 15.7760 /lib64/tls/libc-2.3.4.so memset

785 5.5785 /lib64/tls/libc-2.3.4.so memcpy

608 4.3206 /usr/lib/debug/lib/modules/2.6.9-34.ELsmp/vmlinux thread_return

398 2.8283 /usr/libexec/mysqld yyparse(void*)

337 2.3948 /usr/lib/debug/lib/modules/2.6.9-34.ELsmp/vmlinux system_call

“The results on the system as a whole were that with read buffer set to 5M, extra 3-5% usr and 6-12% sys cpu time taken on an 8 cpu box (as reported by top) or, to look at it another way, about 100% of a single CPU taken by mysqld.”

# Don't leave them as default either!

- Don't leave the my.cnf as default configs either!
  - Minimally set:
    - Innodb buffer pool
    - Key Buffer
    - Query cache
    - Trx commit
- The sample my.cnf are just samples!
- I have some better examples on bigdbahead.com

# Swap?

- Despite what really smart people say... don't turn off swap...  slow or dead?
- Whats your swapiness? Cat /proc/sys/vm/swappiness
- Carefully manage your memory to avoid swap...

# Text Fields

- Many clients do not realize that text fields and varchars have the same max size (64K)
- Text fields have to use temporary tables on disk, while varchars can use memory based temporary tables
- Distinct, aggregates, group by, order by can all cause excessive disk utilization due to temp tables.
- Ruby and the select *

# Data Size Matters

- Thin is in!
- Smaller datatypes mean less storage on disk
- Less storage per row means faster retrieval of rows
- Less storage per row means more rows fit into memory
- Size especially matters with innodb PK's!
  - PK's are clusters indexes
  - Every subsequent index contains the PK!

# Data type size

Data size effects disk utilization as well as memory.

| DataType | bytes | size | Prefered | bytes | size |
|---|---|---|---|---|---|
| datetime | 8 bytes | | timestam | 4 bytes | seconds since epoch (up to 2038) |
| bigint | 8 bytes | Unsigned Up to: 18,446,744,073,709,551,615 | int | 4 bytes | Unsigned Up to: 4,294,967,295 |
| text | variable | up to 65,535 | varchar | variable | up to 65,535 |
| float | 4 bytes | | Decimal | variable | |
| double | 8 bytes | | Decimal | variable | |

People index the full varchar fields, end up with key sizes of 300bytes or more

# General data type reminders

- Avoid floats when not needed
- Remember to unsign variables when need be
- People still think int(1) means 1 byte or one digit
- Don't forget Enums
- Bigints as 64 bit masks?  Avoid it, really dumb.
- Datetime is 8 bytes, date/time separate columns is 6...
- Ip varchars of doom... use INET_ATON()
- Hash values sometimes better for indexes
- Not null can save space...  why are you allowing nullables?

# Avoid Needless data conversions

- DON'T convert column data with functions, avoid something like :
  - Select * from users where lower(name) = 'matt'
  - This will negate the use of an index on name
  - Consider inserting data with with the function, or use a case insensitive collation
- If you are building a higher performance system avoid implicate conversions of strings to ints... more on next slide.
- If your going to store numbers don't use a char or varchar

# Time to convert datatypes...

- **Implicit conversion of data types wastes resources... some small some large...**

Single quote of integer is small overhead, but in a really busy system this can add up... below my benchmark shows < 1 ms difference in runtimes. You would save 11 seconds per 100K queries... but in a busy system you may have millions of queries per minute.

```
mysql> select qryid, note, avg(averagequeryruntime),  count(*), querytext from Logger_single group by qryid, note;
+-------+------------------+-------------------------+----------+-----------------------------------------------------+
| qryid | note             | avg(averagequeryruntime) | count(*) | querytext                                           |
+-------+------------------+-------------------------+----------+-----------------------------------------------------+
|     1 | 1 file no restart |     0.00042705535888672 |      210 | select * from fragtest.test_frag1 where id = 30718;   |
|     2 | 1 file no restart |     0.00052125908079601 |      210 | select * from fragtest.test_frag1 where id = '30718'; |
|     3 | 1 file no restart |     0.00041392644246419 |      210 | select * from fragtest.test_frag1 where id = 90718;   |
|     4 | 1 file no restart |     0.00052596046811058 |      210 | select * from fragtest.test_frag1 where id = '90718';  |
+-------+------------------+-------------------------+----------+-----------------------------------------------------+
4 rows in set (0.02 sec)
```

A system issuing 100K selects per minute would save 4.5 hours of CPU time each day ( or 19% of 1 cpu ) by not using single qoutes! That scales up, 200K selects is 9 hours of CPU, @1 million selects a minute you would save 45 hours, which is almost 2 full CPU's.

# Type Comparison

Every knows that if you miss enclosing a character reference in a select statement it will complain. For instance:

mysql> explain select * from MyTest where name =matt;

ERROR 1054 (42S22): Unknown column ‘matt’ in ‘where clause’

But what most miss is this is not always the case:

mysql> explain select * from MyTest where name =1234;

```
+—-+———-+——-+——+———————+——+———+——+——+———————-+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+—-+———-+——-+——+———————+——+———+——+——+———————-+
| 1 | SIMPLE | MyTest | ALL | myidx | NULL | NULL | NULL | 1 | Using where |
+—-+———-+——-+——+———————+——+———+——+——+———————-+
```

The impact of this can be huge, I was at a client who used a varchar as a PK on a table ( bad by itself ). All the values in the varchar field where

numeric, but all the statements coming from the application used unescaped integers to within the SQL statement ( i.e. id = 123 instead of id = ‘123′ ).

This has the unfortunate side effect of bypassing the index, which in this extreme case meant no PK access … which was very, very bad.

# Left Outer Joins

One of the big pet peeves of mine is looking through thousands of SQL statements and seeing everyone of the statements is doing a left outer join. My Absolute favorite is something like this:

**select a.var, b.var from a left join b on a.id=b.id where b.var = ‘xxx’ ….**

| aid | bid | avar | bvar |
|---|---|---|---|
| 1 | 1 | xxx | xxx |
| 2 | 2 | xxx | xxx |
| 3 | NULL | xxx | NULL |
| 4 | NULL | xxx | NULL |

folks if you set the value of a column in the b table your left join is meaningless, think about it… if the ids do not match the b side will all return null… a null b.var will be filtered out. What you probably want is:

**select a.var, b.var from a left join b on a.id=b.id and b.var = ‘xxx’ where ….**

# Left Joins 2

Also … another little left join nugget. I don;t know why but several folks have done this type of an operation:

**select a.var1, a.var2 from a left join b on a.id=b.id where a.var1 = ‘xxx’**

The question is what is the benefit of joining to b?

| aid | bid | avar | bvar |
|---|---|---|---|
| 1 | 1 | xxx | xxx |
| 2 | 2 | xxx | xxx |
| 3 | NULL | xxx | NULL |
| 4 | NULL | xxx | NULL |

My favorite argument of left join happy shops is, well we put it in there just in case we missed something… enough said.

Be Aware hibernate loves left joins!!

# Mixing Storage engines

- Splitting half the tables between MyIsam and innodb is always a bad idea
  - Get the worst of both worlds
  - Config must split memory

# Indexing Tips

- AVOID THE Solo index on every column
  - Seen 3 or 4 clients do this...
  - Every column has a single index
  - Too many indexes slow down inserts/updates
  - Bloat memory
  - Bloat disk
  - Index merges can happen in some cases, but they are slower then multiple key indexes
- Remove redundant indexes
  - KEY ( id ) and KEY ( id, type ) are redundant!
- Run explain!

# PK Innodb & Disk

- Use Small Pks, remember data size!
- Remember the order of the PK = Disk sorted order
- Huge performance improvement 20x at customer
- Don't always use auto-increment... or put it at the end.

# IO issues

One of my goals in almost any engagement is to reduce the amount of data being read from or written to disk... why?

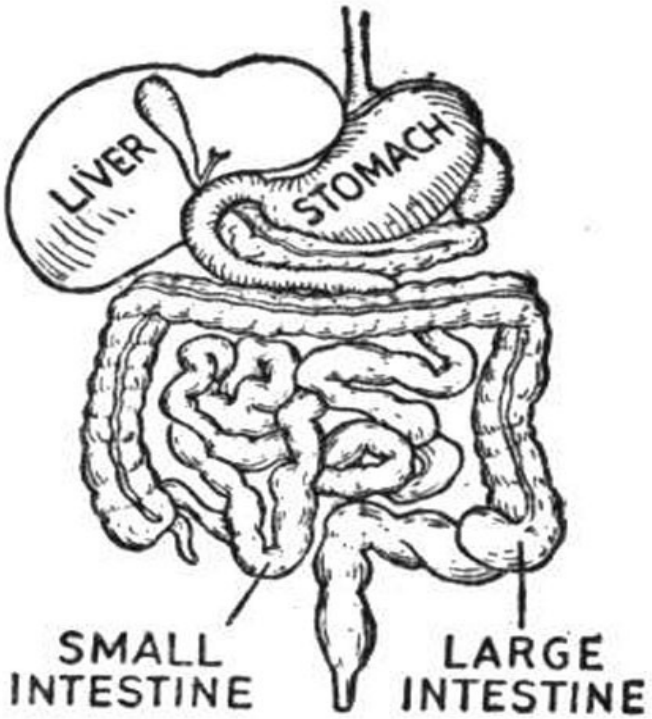
Because it's almost always an IO issue ( most of the time anyway )

# Is it always an IO Problem?

- Yes...
- Because wiki is 100% accurate:
  - In computing, input/output, or I/O, refers to the communication between an information processing system (such as a computer), and the outside world – possibly a human, or another information processing system. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it. The term can also be used as part of an action; to "perform I/O" is to perform an input or output operation.
- In order to make Forrest run faster we need to fix his IO!

# Let’s Clarify some more

- IO is not just disk!  Disk is just the rear end...
- IO is transfer data around the various inards of a computer, much like: ( Yes I am comparing IO to the digestive system )



# Consider

- These are all IO issues:
  - The lines at your local amusement park
  - Disks can not retrieve blocks fast enough, and have to wait for huge queues to be cleared
  - Too much bloat in memory is causing in memory operations to take 100 nanoseconds instead of 40 nanoseconds.  ( not a cpu bottleneck even though CPU may be maxed out )
  - The network throughput is maxed out, why?  Because someone is retrieving millions of rows... network is not bad ...
  - There is more....

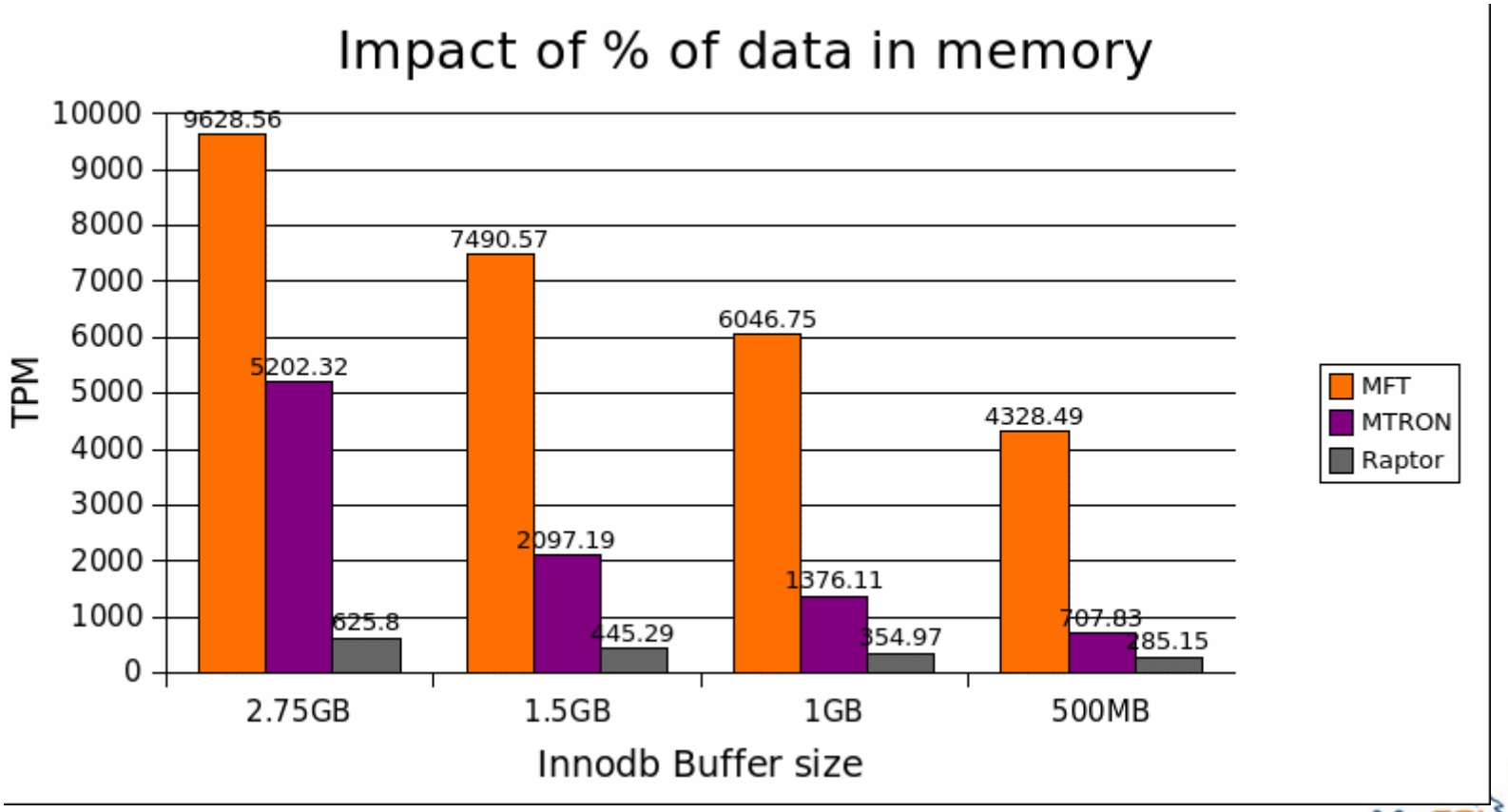
# Reduce!!!!!

- IO requires a killer diet on most systems... ( and I could use one too )
  - Reducing data size helps!
  - Reducing what your asking for helps even more
  - Stored procs may help... keep the data local
  - Pre-summarize
- Along with diets you need exercise:
  - Faster pieces can help, cpu, memory, disk
  - But like a me, even if I work out... I can not just eat crap food.

# Always an IO Problem?

- Spindles not Capacity!  Clients often estimate how much space they need, but rarely how many IOPS!

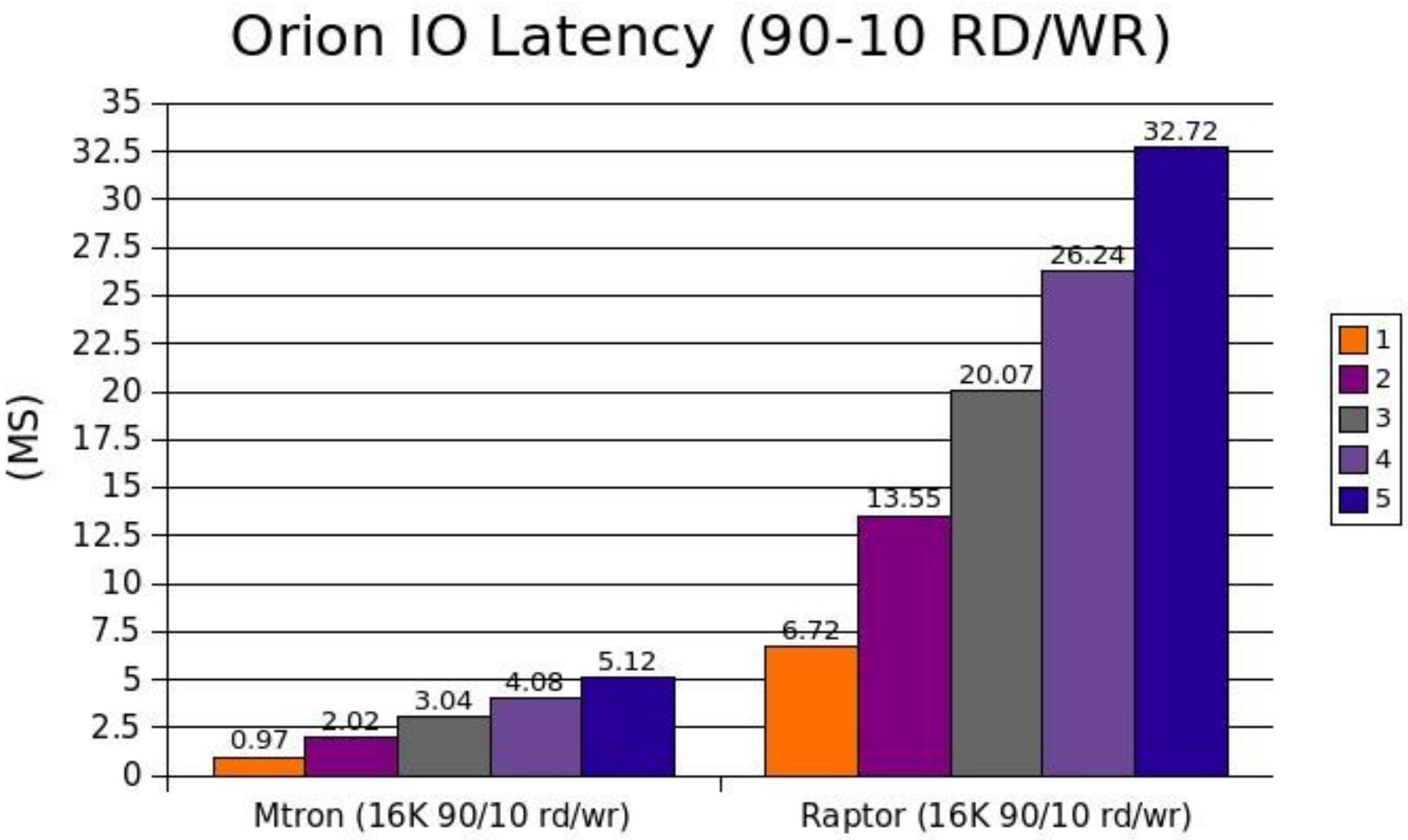
  Look how disk performance impacts DB performance:



# All about concurrency

- Many benchmarks, other people test throughput with 1 thread, who has a database that does only one thing?
- Todays databases demand high concurrency, disk is key to this...
- Disk arm movement ends up killing performance...
- The 7:30am client call.... 50 load average, 600ms disk time, from 500 users to 1000.

# Concurrency Matters!



# Filesystem & LUN layout

- Swap and MySQL on the same LUN is BAD!
  - Asking for performance issues
- /var/log and /tmp on the same Lun as mysql is bad
  - /tmp filling up can freeze the database
  - Log flling up can freeze the database
- Best with its own separate disks.

# SAN != Black Box

- San is not a black box!
- Emc technology prime example:  Moves hot data around...  lots of cache, should never be an issue

# Self-induced fragmentation

- What happens when you have a 20GB table, but have multiple 2GB datafiles?

- What the separate data files do is they introduce self-induced fragmentation. When you go to scan the table you now have to potentially scan through 20 files instead of 1 file. This also potentially will disrupt the “clustered index” advantage that innodb provides.

- I put together some tests to show this:

# Test Setup

I created 4 identical tables with the following definitions:

```
CREATE TABLE `test_frag1` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(300) DEFAULT NULL,
`address` varchar(1000) DEFAULT NULL,
`state` char(2) DEFAULT NULL,
`val1` int(11) DEFAULT NULL,
`val2` int(11) DEFAULT NULL,
`val3` int(11) DEFAULT NULL,
`val4` int(11) DEFAULT NULL,
`val5` int(11) DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `name_idx` (`name`),
KEY `val_idx` (`val1`,`val2`)
) ENGINE=InnoDB AUTO_INCREMENT=1039773 DEFAULT CHARSET=latin1
```

# Test Setup (continue)

I loaded the data in the four tables 1 record at a time, alternating tables. I then went back and updated 10% the data to ensure a somewhat realistic dataset and fragmentation.

I then ran a sampling of the following queries:

1 select * from fragtest.test_frag1 where id = ‘30718′;
2 select * from fragtest.test_frag1 where val1 < 1000 limit 1,1;
3 select * from fragtest.test_frag1 where id = 90718;
4 select count(*), sum(val1), sum(val2), avg(val3) from fragtest.test_frag1 where val4 < 10000;
5 select sum(val4+val5) from fragtest.test_frag2;
6 select * from fragtest.test_frag3 where val1=5718;
7 select * from fragtest.test_frag4 where val1> 5700 and val2 < 10000 and val3 =14242;
8 select count(*), sum(val1), sum(val2), avg(val3) from fragtest.test_frag1
where state in (94,84,35,36,11,12,99,50) group by state ;
9 select * from fragtest.test_frag3 a, fragtest.test_frag1 b where a.val1 = b.id and a.id = 1111;
10 select * from fragtest.test_frag2 a, fragtest.test_frag4 b where a.val1 = b.id and b.id = 9690;
11 select sum(a.val2) from fragtest.test_frag2 a, fragtest.test_frag4 b where a.val1 = b.id
and b.id = 999;

# Test Setup (continue)

- I dropped the top and bottom 5% to ensure no skewed tests:

| Query ID | 1 Innodb File | 40 Innodb files | DIFF x |
|---|---|---|---|
| 1 | 0.0103 | 0.0159 | 53.53% |
| 2 | 0.0567 | 0.0478 | -15.72% |
| 3 | 0.0205 | 0.0226 | 10.16% |
| 4 | 7.0861 | 7.6782 | 8.36% |
| 5 | 7.8960 | 8.4054 | 6.45% |
| 6 | 3.7241 | 3.7422 | 0.49% |
| 7 | 8.3528 | 9.2764 | 11.06% |
| 8 | 7.0573 | 7.7802 | 10.24% |
| 9 | 0.0468 | 0.0505 | 8.05% |
| 10 | 1.3608 | 1.3966 | 2.64% |
| 11 | 0.0211 | 0.0262 | 24.14% |

# Customer Example

- This seriously effects clients performance.
- They have over 100 2GB datafiles....  just craziness!

# Summary

- Avoid the same mistakes and issues your IT brethren have done!

Want more?

Read more at bigdbahead.com!

Stop by and see me give this lecture in person at the MySQL Camp at the user conference

Or better yet stop by and see my sessions on Solid State disk & Waffle Grid!