

MySQL/InnoDB performance Server and Schema



Peter Zaitsev,
MySQL AB

MySQL Users Conference 2004
Orlando, FL April 14-16

Let me Introduce Myself

- Peter Zaitsev, MySQL Inc
 - Senior Support Engineer
 - Benchmark Project leader
 - Does on site and remote consulting
 - Participates in Server Development
- Before joining MySQL
 - CTO of www.mytrix.com - Web statistics projects
 - One of the largest MySQL/InnoDB installations in Russia at the time
- Based at Seattle, WA

About Presentation

- Optimizing MySQL/InnoDB Performance
 - Two presentations in series
- This part is about MySQL server options and Schema optimizations
- Using DBT2 by OSDL, TPC-C like benchmark
- Practical approach – how to locate and eliminate next bottleneck
- Real performance figures
 - Numerical value of each performance improvement
- Why InnoDB ?
 - MyISAM can't run transactional benchmark
 - There seems to be lack of InnoDB optimization info

Question Policy

- Interrupt me if something is unclear
- Keep long generic questions to the end
- Approach me during the conference
- Write me to peter@mysql.com

DBT2 Benchmark Info

- Developed by OSDL (Mark Wong and Co)
 - MySQL port done by Alexey Stroganov (not in the mainline yet)
- More information: <http://sourceforge.net/projects/osdl/dbt2>
- Quite close TPC-C implementation, but allows wider parameter range
- Results are incompatible with TPC-C and can't be compared to them
- TPC-C Benchmark Description
<http://www.tpc.org/tpcc/detail.asp>

Benchmark Configuration

- Two workloads “large” and “small”
- 200 Warehouse database – about 30Gb real size on the disk
- “small” workload touches 10 of them, being CPU bound
- Using 200 “terminals” and 20 connections in all cases
- Zero think delay to fully load database
- Hardware
 - 4*Xeon 2.0 Ghz MP (with HT), 512K cache, 4G of memory
 - 8SATA 7200 drives in RAID10, 1024K chunk on 3WARE8500-8
 - System on Separate set of SCSI drives
- Software: RH AS 3.0, MySQL 4.1.1-alpha

Running Benchmark

- Default Kernel: 2.4.21-9.ELhugemem
- Default Filesystem: EXT3
- Swap partition disabled
- Run series
 - 4 runs, 2 “large” load , followed by 2 “small” load
 - Sleeps between loads to allow database to settle down
 - 15min + 5 min warmup for each of loads
- Best out of 2 results is taken in most cases
- Additional experiments performed to measure accuracy of approach
- Results in TPM (transactions per minute), More is better

Default Schema and MySQL options

- Very poor results
 - Lets try to take a look what we can do with schema

	LARGE	SMALL
Default Options	55	69

Analysing Results

- Running “**mysqladmin -i10 -r extended**”
- Massive amount of range or index scans

Handler_commit	41	
Handler_delete	0	
Handler_read_first	4	
Handler_read_key	4599	
Handler_read_next	7063454	
Handler_read_prev	0	
Handler_read_rnd	329	

Enabling logging to catch the query

- Enable slow query log, adding options to **my.cnf**
 - **Log-slow-queries**
 - **Log-long-format**
 - **Long-query-time=2**
- Rerunning benchmark to get slow queries logged

Analyzing slow query log

- # Query_time: 21 Lock_time: 0 Rows_sent: 0
Rows_examined: 0 UPDATE warehouse SET w_ytd =
w_ytd + 2628.000000 WHERE w_id = 25;
- Can't run **EXPLAIN** with **UPDATE**, changing to **SELECT**

```
mysql> explain select * from warehouse where w_id=25;
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
| 1 | SIMPLE | warehouse | const | PRIMARY | PRIMARY | 4 | const | 1 |
```

This is "false" slow query. It is very simple query by nature but system was likely to overloaded or there was Row level lock wait which delayed it. We can't know without response time profiling.

Analysing Slow Query Log

- # Query_time: 10 Lock_time: 0 Rows_sent: 2
Rows_examined: 3002 SELECT c_id FROM customer WHERE c_w_id = 25 AND c_d_id = 4 AND c_last = 'ANTIEINGANTI' ORDER BY c_first ASC;
- mysql> **EXPLAIN SELECT c_id FROM customer WHERE c_w_id = 25 AND c_d_id = 4 AND c_last = 'ANTIEINGANTI' ORDER BY c_first ASC;**

```
| id | select_type | table      | type | possible_keys | key      | key_len | ref | rows | Extra |
| 1 | SIMPLE      | customer  | ref  | PRIMARY      | PRIMARY |         |    | 8 | const,const |6028 |
Using where; Using filesort |
```

Good query from glance view, But we can do better extending key to (c_w_id,c_d_id,c_last,c_first) – this will avoid to avoid filesort as well

Slow Log: Found real performance killer

- # Query_time: 350 Lock_time: 0 Rows_sent: 977
Rows_examined: 1818725 SELECT no_o_id FROM new_order WHERE no_w_id = 3 AND no_d_id = 1
- mysql> EXPLAIN SELECT no_o_id FROM new_order WHERE no_w_id = 3 AND no_d_id = 1
- | 1 | SIMPLE | new_order | index | NULL | PRIMARY | 12 | NULL | 1543785 |
Using where; Using index
- Query full index scan while it can be “ref” type
- Will need to add key with (no_w_id,no_d_id) prefix

Slow Query Hunting & optimizing tips

- Hard to find all bad queries from first run
 - Flooded with most common slow queries
 - Optimize them and run once again
- Use **SHOW PROCESSLIST** to catch most typical queries and check them.
- Running **ALTER TABLE** for large **INNODB** table
 - Set **innodb_buffer_pool** to 80% of physical memory
 - Increase **innodb_log_file_size=512M**
 - Allow **ALTER TABLE** to complete, rollback at startup is even longer
- Benchmark with smaller database size to find optimal indexes
- Use **ALTER TABLE** to add all indexes at once instead of

Results for Indexed run

- We get almost 8.5 times better performance by indexing
- Strange enough SMALL load is slower than LARGE
 - Likely due to increased concurrency

	LARGE	Ratio	SMALL	Ratio
Default	55		69	
Indexed	463	8.42	409	5.93

Review how status changed

- Now “**mysqladmin -i10 -r extended**” looks much healthier
- Now excessive index scans

Handler_read_first	0
Handler_read_key	120614
Handler_read_next	573905
Handler_read_prev	0
Handler_read_rnd	7783
Handler_read_rnd_next	0
Handler_rollback	10
Handler_update	0
Handler_write	10003

SHOW INNODB STATUS Review

- **SHOW INNODB STATUS** – command to view Internal **INNODB** runtime stats
- **FILE IO**
 - 520155 OS file reads, 146300 OS file writes, 7609 OS fsyncs
1106.36 reads/s, 17133 avg bytes/read, 301.87 writes/s,
15.05 fsyncs/s
- Very high amount of reads !
- Default **innodb_buffer_pool** is 8M
 - Increasing it to 1800M
 - Can't do more due to 32bit limits and GLIBC limits

Results with larger Buffer Pool

- Once again good performance improvement
- SMALL workload is now faster as expected

	LARGE	Ratio	SMALL	Ratio
Default	55		69	
Indexed	463	8.42	409	5.93
IBP=1800M	931	16.93	2107	30.54

Taking a closer look at SHOW INNODB STATUS

- Running “**SHOW INNODB STATUS**” during the test
- Looking at “BUFFER POOL AND MEMORY” section
 - Pages read 576279, created 13448, written 1082267 **85.84 reads/s**, 4.03 creates/s, **542.08 writes/s** Buffer pool hit rate **999 / 1000**
- We have 6 times more writes than reads
 - We do not expect it for our benchmark
 - Reason: default **innodb_log_file_size=5MB**
- Buffer pool hit rate almost perfect
 - confirming “hit ratios” are not always helpful

Increasing log files sizes

- We'll set **innodb_log_file_size=512M**
- Tricky Operation !
 - Shut down MySQL Server cleanly
 - Move your old InnoDB logs to the safe place
 - Change **innodb_log_file_size**
 - Restart and wait for new logs to be created
- Larger logs require longer recovery time
 - Be careful !

Results with innodb_log_file_size=512M

- Good improvement for both “large” and “small” loads

	LARGE	Ratio	SMALL	Ratio
Default	55		69	
Indexed	463	8.42	409	5.93
IBP=1800M	931	16.93	2107	30.54
ILFS=512M	1223	22.24	3099	44.91

Detailed Analyses

- **MySqladmin -i10 -r extended**
 - Opened_tables 502
 - More than 50 table opens per second, need to increase table_cache
- **SHOW INNODB STATUS**
 - 6 queries inside InnoDB, 4 queries in queue
 - Increase **innodb_thread_concurrency=32**
(num_disks+num_cpus)*2
 - 89275 log i/o's done, 73.81 log i/o's/second
 - Rather high log IO
 - Set **innodb_log_buffer_size=8M**

Results after these optimizations

- Extra 10% for SMALL load
 - Typical for optimization
 - Few changes give major impact
 - Afterwards you only move by few percent

	LARGE	Ratio	SMALL	Ratio
Default	55		69	
Indexed	463	8.42	409	5.93
IBP=1800M	931	16.93	2107	30.54
ILFS=512M	1223	22.24	3099	44.91
Optimized	1259	22.89	3442	49.88

Taking a closer look at Schema

- We previously added key to this table to optimize query
 - It matches **PRIMARY KEY** but order of columns is different.
 - We can change primary key to it and have only one key
 - This assumes we do not have any queries using current PK
 - `CREATE TABLE `new_order` (
 This table has many inserts, so saving keys is important
 `no_o_id` int(11) NOT NULL default '0',
 `no_d_id` int(11) NOT NULL default '0',
 `no_w_id` int(11) NOT NULL default '0',
 PRIMARY KEY(`no_o_id`,`no_d_id`,`no_w_id`),
 KEY `no_d_id` (`no_d_id`,`no_w_id`,`no_o_id`)
)`

Shortening primary key

- **(s_w_id,s_i_id)** is already unique, so **s_quantity** is not needed in PK
- This field is frequently updated
 - Updates of Primary key columns are very expensive

```
CREATE TABLE `stock` (  
  `s_i_id` int(11) NOT NULL default '0',  
  `s_w_id` int(11) NOT NULL default '0',  
  `s_quantity` double NOT NULL default '0',  
  `s_dist_01` varchar(24) default NULL,  
  ...  
  `s_ytd` decimal(16,8) default NULL,  
  `s_order_cnt` double default NULL,  
  `s_remote_cnt` double default NULL,  
  `s_data` varchar(50) default NULL,  
  PRIMARY KEY (`s_w_id`,`s_i_id`,`s_quantity`)  
)
```

Changing physical row layout

- Rebuilding table can help performance
 - Reducing table fragmentation
 - Faster table scan
 - Making data better clustered by Primary Key
 - Reducing data size
 - More tight page packing
- Done by “**ALTER TABLE tbl TYPE=INNODB**”
 - Or just “**OPTIMIZE TABLE**” in newer MySQL versions.

Updated Results

- Good improvement for “small” load
- About the same performance for “large” load
 - “large” load may suffer from more page splits

	LARGE	Ratio	SMALL	Ratio
Default	55		69	
Indexed	463	8.42	409	5.93
IBP=1800M	931	16.93	2107	30.54
ILFS=512M	1223	22.24	3099	44.91
Optimized	1259	22.89	3442	49.88
Layout	1250	22.73	4650	67.39

Eliminating long primary key

- This table has very long primary key
 - Long primary keys may be problem in InnoDB due to internal structure
 - This is normally problem with 2 or more keys
 - We'll still show how to deal with it for illustration purposes

- Replace primary key with UNIQUE key, adding auto_increment PK

- There are downsides of such change.

```
CREATE TABLE `order_line` (
  `ol_o_id` int(11) NOT NULL default '0',
  `ol_d_id` int(11) NOT NULL default '0',
  `ol_w_id` int(11) NOT NULL default '0',
  ...
  `ol_amount` double default NULL,
  `ol_dist_info` varchar(24) default NULL,
  PRIMARY KEY (`ol_w_id`,`ol_d_id`,`ol_o_id`,`ol_number`)
)
```

Converting table

```
mysql> CREATE TABLE `order_line2` (  
-> pseudo_id int unsigned not null auto_increment primary key,  
-> `ol_o_id` int(11) NOT NULL default '0',  
-> `ol_d_id` int(11) NOT NULL default '0',  
-> `ol_w_id` int(11) NOT NULL default '0',  
-> `ol_number` int(11) NOT NULL default '0',  
-> `ol_i_id` int(11) default NULL,  
-> `ol_supply_w_id` int(11) default NULL,  
-> `ol_delivery_d` timestamp NOT NULL,  
-> `ol_quantity` double default NULL,  
-> `ol_amount` double default NULL,  
-> `ol_dist_info` varchar(24) default NULL,  
-> UNIQUE KEY (`ol_w_id`,`ol_d_id`,`ol_o_id`,`ol_number`)  
->) TYPE=InnoDB;
```

Query OK, 0 rows affected (0.53 sec)

```
mysql> insert into order_line2 select 0,  
ol_o_id,ol_d_id,ol_w_id,ol_number,ol_i_id,ol_supply_w_id,ol_delivery_d,ol_quantity,ol_a  
mount,ol_dist_info from order_line;
```

```
mysql> rename table order_line to order_line_old;
```

Query OK, 0 rows affected (0.24 sec)

```
mysql> rename table order_line2 to order_line;
```

Query OK, 0 rows affected (0.24 sec)

Updated Results

- We got slower performance from this change (expected)
- “small” load suffered more as it takes CPU to maintain keys

	LARGE	Ratio	SMALL	Ratio
Default	55		69	
Indexed	463	8.42	409	5.93
IBP=1800M	931	16.93	2107	30.54
ILFS=512M	1223	22.24	3099	44.91
Optimized	1259	22.89	3442	49.88
Layout	1250	22.73	4650	67.39
RemovedPK	1247	22.67	4341	62.91

Compare MySQL 4.1 to MySQL 4.0

- MySQL 4.0 is about the same for “large” load and faster for “small” load
- Features come at some CPU usage cost
- MySQL 4.1 is alpha is not fully optimized yet
- This applies to this load only. Some loads are significantly faster on MySQL 4.1

	LARGE	Ratio	SMALL	Ratio
MySQL 4.1	1250		4640	
MySQL 4.0	1224	0.98	4900	1.06

Testing new MySQL 4.1 features

- New InnoDB file per table format
 - Set **innodb_file_per_table=1**
 - This creates each table in each own tablespace, in its own file
- Using UTF8 Charset for data storage
 - We still stored only latin1 characters in data
 - Client-Server protocol used latin1

Performance using New Features

- File Per table gives expected performance improvement
 - Better data clustering
 - Less concurrency
 - Linux 2.4 does internal locking on inode level
- UTF8 Improvement is unexpected
 - Could be due to better layout archived by export/import
 - At least we see UTF8 does not slow down things a lot

	LARGE	Ratio	SMALL	Ratio
Default	1250		4640	
File Per Table	1336	1.07	4816	1.04
UTF8	1344	1.08	4983	1.07

Different Amount of connections

- Some applications allow to tune amount/maximum amount of connections they use.
- Number of connections more than number CPUs and number of disks is efficient
- With very many connections performance starts to decline
- **innodb_thread_concurrency** really helps with a lot of connections

Connections	LARGE	Ratio	SMALL	Ratio
4 (Phys. CPUs)	820	0.66	3254	0.7
8 (Log CPUs)	1002	0.8	4246	0.92
20 (Default)	1250		4640	
200	1070	0.86	-	
200*	971	0.78	-	

* - increased innodb_thread_concurrency=300 to allow all threads in InnoDB Kernel

InnoDB IO Modes

- Set by **innodb_flush_method** option
- O_DIRECT gives great result by avoiding double buffering and other overheads
 - Still rather new kernel feature, not all file systems support it
- O_DSYNC increases DoubleWrite overhead a lot. Avoid it

IO Mode	LARGE	Ratio	SMALL	Ratio
Default	1250		4640	
O_DIRECT	1931	1.54	5542	1.19
O_DSYNC	995	0.8	4498	0.97

What if you do not need full Durability

- In many cases (including ours) fsync() is fake, not flushing drive cache
 - We still do not get 100% durability
- InnoDB allows to avoid log flushing at commit
 - `innodb_flush_log_at_trx_commit=0` – do not flush log
 - `innodb_flush_log_at_trx_commit=2` – flush to OS cache only
- InnoDB still does full log flush to disk about once per second

Log flush	LARGE	Ratio	SMALL	Ratio
Default (1)	1250		4640	
Disabled(0)	1265	1.01	6071	1.31

Query Cache and Binary Log

- You might need to have binary log enabled for replication or Recovery
 - Enabled by setting **log-bin** option
- Query Cache is not expected to cache a lot in this workload but some tables are constant so some benefit is possible
 - This is not Query Cache friendly workload
 - Enabled by **query_cache_size=128M**

	LARGE	Ratio	SMALL	Ratio
Default	1250		4640	
Binary Log	1206	0.96	4393	0.95
Query Cache	1272	1.02	4728	1.02

Conclusion

- By tuning MySQL Server options and Schema we were able to get **87** times performance improvements for “small” workload (69->6071 TPM)
 - One can do even better spending more time
- Both Schema optimization and servers settings are important
- MySQL provides powerful and easy to use tools for performance tuning, more on the way.
- Optimization is never ending job
 - After few easy items you have mostly minor or expensive optimizations
 - Identify which performance do you need.

Resources

- <http://www.mysql.com/doc> - MySQL online Manual
- <http://lists.mysql.com> - MySQL mailing list
 - Especially Main mailing lists, Benchmarks list
- Get help and advice from MySQL employees
 - <http://www.mysql.com/support> - Support
 - <http://www.mysql.com/consulting> - Consulting
- Come to my next session to hear about Optimizing hardware and OS
 - Friday, April 16, 3:20 PM
- Signup for InnoDB Performance Tuning class, Friday April 16, 1:00 PM
- Write me to peter@mysql.com if you have questions