MySQL/Innodb performance optimization

Peter Zaitsev,
MySQL AB

O'Reilly Open Source Convention 2004
Portland,OR July 26-30

About Presentation

- Optimizing MySQL/Innodb Performance
- Quick walk through – parameters/schema/OS
- Using DBT2 by OSDL, TPC-C like benchmark
- Practical approach – finding and eliminating 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

DBT2 Benchmark Info

- Developed by OSDL (Mark Wong and Co)
  - MySQL port done by Alexey Stroganov
  - Finally available in DBT2 mainline
- More information: http://sourceforge.net/projects/osdldbt
- 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 touchs 10 of them, being CPU bound
- Using 200 “terminals” and 20 connections in all cases
- Zero “think time” 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 SCSI drives
- Software: RH AS 3.0, MySQL 4.1.3-beta

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 acuracy 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 issue

- # 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 mamory
  - 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 **CREATE INDEX**

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
- No 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 !
  - More than hardware can physically do
- 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
  - Massive dirty page flushes, which can be postponed
- 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 clearly
  - 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
  - small logs do not guaranty short recovery time

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 prefix (these could break)
    - This table has many inserts, so saving keys is important

```
CREATE TABLE `new_order` (
        `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_i`),
       KEY `no_d_id` (`no_d_id`,`no_w_id`,`no_o_id`)
)
```

Shorterning 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

Other usable hints

- Use short integer primary keys
  - All secondary keys reference rows by primary keys
- Avoid random primary keys (ie GUID)
  - Inserts in random places in BTREE fragment table badly
- Use prefix keys where possible
  - Innodb does not have key compression as MyISAM
  - **ALTER TABLE TBL ADD KEY (COL(16))**

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

RH AS Kernel Variants

- "Hugemem" has 4G:4G memory split but at cost
  - Allowing you to allocate almost 4G of memory
- New RedHat kernel improved, but still can be worse than "smp"

	LARGE	Ratio	SMALL	Ratio
2.4.21-9.ELhugemem	1250		4640	
2.4.21-9.ELsmp	1485	1.19	8132	1.75
2.4.21-15.ELhugemem	1664	1.33	4999	1.08

What is about 2.6 ?

- Performance in 2.6 is getting better
  - CPU bound load still not up to speed
- Deadline scheduler is best one.
  - "**elevator=deadline**" kernel boot option

2.4.21-9.ELsmp	1485		8132	
2.6.3 (as)	551	0.37	7587	0.93
2.6.3 (deadline)	1872	1.26	7630	0.94
2.6.7 (as)	1778	1.2	7657	0.94
2.6.7 (deadline)	2238	1.51	7715	0.95
2.6.7 (noop)	2219	1.49	7701	0.95

Should I keep my swap enabled ?

- 2.6.7 Does not suffer from enabled swap anymore
  - There was no swapping at all during test
- 2.4 suffers from enabled swap really badly
- 2.6.7 without swap is unsafe (OOM can trigger)

	LARGE	Ratio	SMALL	Ratio
2.4.21-9.ELsmp noswap	1485		8132	
2.4.21-9.ELsmp swap	384	0.26	3386	0.42
2.6.7 noswap	2238		7715	
2.6.7 swap	2252	1.01	7755	1.01

Performance of Direct IO

- Direct IO is enabled by **innodb_flush_method=O_DIRECT**
  - Not all filesystems and kernels support it yet
- Direct IO seems to reduce 4:4 memory split penalty

Kernel 2.4.21-9.ELhugem	LARGE	Ratio	SMALL	Ratio
Buffered IO	1250		4640	
Direct IO	1931	1.54	5543	1.19

Kernel 2.6.7 (deadline)	LARGE	Ratio	SMALL	Ratio
Buffered IO	2238		7715	
Direct IO	2342	1.05	8737	1.13

RAID10 Block Sizes

- RAID10 3WARE8500-8, 8SATA 7200RPM Drives,
- Kernel 2.4.21-9.ELhugemem, EXT3
- No clear winner
- Large chunk is best for “large” workload
- 256K seems to be optimal for set of the two
- Chunk size has huge impact on performance

	LARGE	Ratio	SMALL	Ratio
16K	761	0.61	4719	1.02
64K	1059	0.85	4869	1.05
256K	1237	0.99	4924	1.06
1024K	1250		4640	

RAID Levels

- Using 64K chunk
  - It is only supported by this hardware for RAID5
- RAID0 best results but, not secure
  - Good candidate if you have many slave servers
- RAID5 is very slow, especially for “LARGE” workload
- RAID10 is normally the best choice

	LARGE	Ratio	SMALL	Ratio
RAID10	1059		4869	
RAID0 (insecure)	1206	1.14	4925	1.01
RAID5	264	0.25	3422	0.7

Conclusion

- Over **100** times performance improved by tuning
  - 69 -> 8737 for small workload
  - One can do even better spending more time
- Server tuning, schema, OS and hardware all important
  - schema is best place to start
- 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
- http://dev.mysql.com/tech-resources/articles/ - Tech Zone
- http://www.livejournal.com/users/peter_zaitsev/ - My BLOG
- Write me to peter@mysql.com if you have questions
- Ask me during the the conference