Advanced InnoDB Optimization

Peter Zaitsev, Tobias Asplund
MySQL AB

MySQL Users Conference 2005
Santa Clara, CA April 18-22

© MySQL AB 2005
Introductions

• Peter Zaitsev, MySQL Inc.
  – Senior Performance Engineer
  – MySQL Performance Group Manager
  – MySQL Performance consulting and partner relationships
Table of Contents

• InnoDB storage engine at glance
• InnoDB vs other storage engines
• Table design issues
• Loading data
• Workload optimization
• Server Settings
• OS and Hardware
• Benchmark results
Question Policy

• Interrupt us if something is unclear
• Keep long generic questions to the end
• Approach us during the conference
• Write us: peter@mysql.com, tobias@mysql.com
Do you use InnoDB

- How many of you?
  - Do not use MySQL yet?
  - Use MySQL with MyISAM tables only?
  - Use InnoDB with other storage engines?
  - Are using only InnoDB or mostly InnoDB?
InnoDB at glance

• Transactional storage engine
  – Fully ACID
  – Row level locks with multi-versioning
    • no locks for standard selects
  – Multiple isolation modes
  – Foreign Key support
  – Automatic crash recovery
    • no need for CHECK TABLE on power failure
  – Data clustered by PRIMARY KEY

• Developed by Heikki Tuuri, Innobase Oy
  – Standard part of MySQL distribution
InnoDB and MyISAM

• Transaction commit overhead
  – Simple updates outside of transaction slower than MyISAM

• Journaling and versioning overhead
  – Large data modifications are slower than MyISAM

• Significantly larger footprint than MyISAM
  – Tables can be 2-3 times larger than MyISAM
  – Memory fit is normally better with MyISAM
  – Indexes are not packed
    • May be 10 times larger than MyISAM for VARCHAR columns

• Clustering by PRIMARY KEY
  – Can be badly fragmented on random inserts
  – Slower than MyISAM for disk full table scan
InnoDB and MyISAM

- **BLOBs** stored outside of the main row, in many pages
  - Slower **BLOB** retrieval and much slower updates
- Very slow index creation (**ALTER TABLE**, **LOAD DATA**)
  - Indexes are currently built row by row
- Some features absent
  - GIS, Full Text search, RTREE indexes
- Some features present
  - Do you need transactions? Foreign Keys? Safety guarantees?
- Fast recovery on crash failure
  - How long **REPAIR TABLE** will take for 1 billion of rows MyISAM table?
- On hardware failure InnoDB recovery can be tricky
InnoDB vs MyISAM

- Much better concurrency for read-write workloads
  - Multiple users can write to the table at the same time
- Data clustered by PRIMARY KEY
  - Primary key lookups up to 2 times faster for disk bound case
- Larger pages (16K) – faster warm up
- Background flushing of modified pages
  - OS can help for MyISAM but not as safe and good
- Only referenced columns are retrieved
  - Faster accessed for tables with BLOB/TEXT columns
- Both data and Index are cached in process memory
  - MyISAM only caches Indexes
- Hash indexes built in memory for faster access
InnoDB table design

• Use short PRIMARY KEY
  – Primary key is part of all other indexes on table
  – Consider artificial auto_increment PRIMARY KEY and UNIQUE for original PRIMARY KEY
  – INT keys are faster than VARCHAR/CHAR

• PRIMARY KEY is most efficient for lookups
  – Reference tables by PRIMARY KEY when possible

• Do not update PRIMARY KEY
  – This will require all other keys to be modified for row
  – This often requires row relocation to other page

• Cluster your accesses by PRIMARY KEY
  – Inserts in PRIMARY KEY order are much faster.
InnoDB table design

- UNIQUE keys are more expensive than non unique
  - Insert buffering does not work
- Prefix indexes especially useful
  - No key compression as in MyISAM
- Long row reduce overhead
  - Blob reading can be skipped if not referenced
  - Vertical partition rarely makes sense
- Manual partitioning still make sense
  - ie users01, users02... users99
  - Table locks is not the problem but ALTER TABLE is
## Loading data to InnoDB table

- Loading data or bulk inserts are much slower than MyISAM
- Loading should be done in **PRIMARY KEY** order
  - Sort externally if needed
  - Performance difference 100x+, faster resulting table
- **SET UNIQUE_CHECKS=0**
  - Do not check keys for uniques – uses insert buffer
    - Make sure it is really unique or mild corruption will happen
- **SET FOREIGN_KEY_CHECKS=0**
  - Improve load speed, do not depend on table order
  - No tool to verify everything is intact after loading
- Good if non-Primary keys fit in buffer pool
  - Insert buffer helps but in memory operations are still faster
Loading data to InnoDB

- Parallel load is better done in different tables
  - Increased fragmentation may happen with same table.
- Parallel load makes sense in two cases
  - You have enough memory to cache both tables
  - You're disk bound in any case and have many disks
- Adjust InnoDB server settings for initial import
  - If your database size is much larger than memory
  - `innodb_buffer_pool_size` up to 90% of memory
  - `innodb_log_file_size` to `innodb_buffer_pool_size`
  - Note you can’t just change the option, check manual
- Load in chunks (ie by 10000 rows)
  - Rolling back failed large import transaction may take days
InnoDB BLOB/TEXT support

- First 512 bytes of BLOB are always stored with row
  - So you can have prefix indexes on BLOB/TEXT
  - 8000 bytes row size applies to this prefix
    - If you have 20 blob fields you may fail on insert
    - Watch for upgrades to utf8 as data size increases

- Remaining stored in its own file segment
  - Allocated page by page up to 32 pages, by 64 pages after
    - Serious size overhead may happen – 33 pages blob will take 96
    - BLOB can be significantly fragmented by 16K chunks
      - Depends on workload a lot. Frequently chunks are larger.

- On update new BLOB is allocated in new place
  - Opposite to how row data is treated

- Single BLOB/TEXT with data in XML may be faster
Optimal transaction size

• Size of transaction is often partially in your control
  – Web application – one transaction per page load or one per “module”? 
• Too small transaction
  – Expensive transaction commit overhead
• Too large transaction
  – Undo space growing (and may need to be flushed to disk)
  – Excessive locking, larger chance of deadlocks
  – Binary log cache overflow
• SELECTs only - fastest outside of transaction
  – but watch for data consistency
• Medium transactions offer best performance
Number of connections

- Easy to control in some applications
  - Java connections pooling
- Harder in LAMP applications but possible
  - i.e. Using FastCGI for PHP processing
- If not connections, limit number of active heavy queries.
  - `SELECT GET_LOCK("mycomplexquery01");`
- Depending on load optimal number is different
  - CPU/disk bound?
  - Query complexity? Application processing delay?
  - Read/write ratio?
- After certain point performance drops significantly
  - If this happened server may never recover by itself
Number of Connections

- `innodb_thread_concurrency` – protection from too many active threads
  - Try setting to `(num_cpus+num_disks)*(2..4)`
  - Try setting to 1000 to disable and see if OS does the job
- Too many threads busts CPU cache efficiency
- Too many threads may randomize disk IO
- Too many threads cause more locks and latches conflicts
- Too many threads result in context switching waste
- Need many threads to load CPUs and disks fully
- Transaction group commit starts to work
  - Committing several transactions with single log write
Caching

- If queries are well designed, memory is most common bottleneck
- InnoDB is much better with its own cache than OS cache
  - All your memory should be accessible by MySQL
  - Look at 64Bit Platform if you have more than 4G RAM
- InnoDB has 16K pages for index and data
  - Fast warm up (data fetched in large chunks)
  - Limited number of “random” rows can be cached at the time
- Moving frequently accessed data to separate table may be good solution
  - Horizontal partitioning, ie “news” and “topnews” tables
Transaction isolation

- Default (**REPEATABLE-READ**) is fast enough
  - Repeatable read results, even no phantoms
  - **SELECT FOR UPDATE/LOCK IN SHARE MODE** will bypass versioning!
- **READ-COMMITTED** allows to save a bit on undo space
- **READ-UNCOMMITTED** – access to the transaction result as it goes
  - You can see how you long transaction is progressing
- **SERIALIZABLE** – set locks on reads
  - More deadlock prone
  - Worse concurrency
  - Waste memory for locks
Locking Reads

- Locking reads bypass versioning
  - You can't lock the row which does not exist any more
- Locking read requires access to the row
  - Covering index will not benefit
    - index on all columns referenced in select for the table
    - `EXPLAIN` will not show it!
  - `SELECT ... LOCK IN SHARE MODE` can be much slower than non-locking `SELECT`
- Locking reads may help prevent deadlocks
  - Using `FOR UPDATE` when reading rows, to be updated in the same transaction
- Locking reads may lead deadlocks, lock waits
  - If used unwisely
Fighting Deadlocks, Lock waits

- Deadlocks hurt performance
  - You need to restart transaction and redo it over
- Row lock waits increase response time.
- Handle deadlocks and timeouts
  - It is hard to guaranty they will never happen for most apps
- Control number of connections you have
- Lock data in the same order in all transactions
- Do not lock data when possible.
- Lock data you will need to lock anyway in the begin of transactions
  - Use `SELECT FOR UPDATE` fetching rows you’ll update
Fighting Deadlocks

- Rows which are not affected may be locked
  - `UPDATE tbl SET status=1 WHERE name like "%s00me%"`
    - All rows in the table will be locked.
  - InnoDB locks the row when it reads it, if it did not match `WHERE` clause on MySQL level it is not unlocked
  - Make sure you use index whenever possible
  - Statement chopping may help if you do not have index
    - `UPDATE tbl SET status=1 WHERE name like "%s00me%" and id between 10000 and 19999;`

- Next-key locks – will not allow to insert just before/after the row in index order
  - Needed to avoid phantom rows
  - `innodb_locks_unsafe_for_binlog` to disable next-key locks
InnoDB Server Settings

• **innodb_buffer_pool_size** – InnoDB data and index cache size
  – Set 70-80% of memory for dedicated MySQL/InnoDB boxes
  – Much more efficient than OS cache especially for write loads

• **innodb_buffer_pool_awe_mem_mb**
  – Windows only AWE “extended” buffer pool
  – Allows to address over 4G of ram on 32bit boxes
  – Much better to use true 64bit platforms these days

• **innodb_additional_mem_pool_size** – Caching dictionary and other internal structures
  – Normally 8-16MB is enough,
  – Requirements growth with number of tables and connections
  – If size is not enough allocation from conventional memory happens
  – Setting it more than needed is just waste of memory
InnoDB server settings

- `innodb_autoextend_increment` — growth chunk for “autoextend” tablespace
  - Default value is 8 (MB) is normally OK
  - Larger value help in intensive growths, reduce file system fragmentation

- `innodb_fast_shutdown` — Should Innodb shutdown fast or with full cleanup
  - Default value (ON) should be good for most cases
  - Fast shutdown does not increase startup time
  - Insert buffer merge can be just done during normal operation on restart

- `innodb_file_io_threads` — number of threads to use for IO helpers
  - Only works on Windows. Unix/Linux has fixed 4 threads
  - More threads may help with more IO devices
  - 4 threads means one thread of each type (read, write, insert_buffer,log)
InnoDB_File_Per_table

- `innodb_file_per_table` – create each table in its own file/tables
  - Option only affects new tables.
    - You can switch it back to create some tables using system tablespace
  - Data and all indexes are stored in the same single file
  - System tablespace is used for undo records
  - You can’t move .ibd file between directories or servers
  - Helpful to manually balance IO between IO devices
  - Helpful for binary backup on per table basics
  - Helpful against per inode IO kernel locks in some OS
  - May have larger update overhead
    - `fsync()` needed for each file, flushing doublewrite buffer
**Innodb_data_file_path**

- Specifies one or several files for system tablespace
  - `ibdata1:2G;ibdata2:10M:autoextend:max:500M`
  - Files are logically concatenated, filled from the first one
    - No IO balancing between files of any sort
- Single file allows to save on `fsync()` calls
- Multiple files help in case of poor kernel locking
  - Limited effect as no control where tables are stored
- You can use Raw partitions for Data files with Innodb
  - `/dev/hdd1:5Graw;/dev/hdd2:2Graw`
    - "newraw" first time to set up tablespace
- Logs have to remain in files
- Very limited performance improvement, especially when DIRECT IO is used.
The `innodb_flush_log_at_trx_commit` parameter can be set to control log flush behavior on transaction commit. Here are the options:

- **Value 0**: No log flushing on transaction commit. Use when performance is paramount. Transaction loss possible when MySQL server crashes.

- **Value 2**: Log is not flushed to the disk, but to OS cache. Usually just a bit slower compared to value 0. Transaction loss only when OS/Hardware crashes. Similar to MyISAM guarantees at some extent.

- **Value 1** (default): Fully ACID, log flushed to disk on commit. Use for applications when no transaction loss can be allowed. Use RAID Battery write back cache for performance. Make sure OS does proper flush to disk or may be worthless.

With any value tables are not corrupted in case of crash.
innodb_flush_method

• Specifies a way InnoDB uses to ensure data made it to the disk
  - **fsync** (default). Both data and log files are flushed by fsync
    • Good for data files as multiple writes followed by single fsync() allows OS to optimize disk IO
  - **O_SYNC** fsync() is used for data files but log file is just opened with O_SYNC flag
    • Try it. Some OS are faster with fsync(), others with O_SYNC for single writes.
  - **O_DIRECT** (Linux) – use direct IO to bypass OS cache
    • Eliminates double buffering and extra copying
    • Sync data writes happen request by request
    • May be faster or slower depending on workload and kernel
    • Normally good choice for RAID with writeback cache
innodb_force_recovery

- Activate Special InnoDB data recovery more
  - Only change when recovering your data, switch back when done
- Start with lower values, continue increasing if InnoDB still asserts
  - Higher values increase chance of incomplete data recovered
- Avoid side load on the server when using this option
  - InnoDB will block updates for safety anyway
- Read manual carefully before using
innodb_lock_wait_timeout

- Timeout waiting for InnoDB row locks
  - InnoDB will abort, roll back transaction waiting too long
- Low values may increase number of timeout errors
- High values may stall your users for very long time
  - It is better to return "try again" to Web user when return nothing for minutes
- Value is global, can’t be set without restarting server
  - setting on per session basics will make a lot of sense
    - Yes Heikki, this is a hint :)
- Avoid long transaction when possible
- For long transaction (ie batch jobs) think what transactions it could be locking
- Use non-blocking reads when possible
innodb_locks_unsafe_for_binlog

- Relaxes InnoDB locking
  - Disables "next-key" locking
  - In MySQL 5.0 makes update/delete to only lock rows selected for update
- This option is named so because using it may break replication
  - Will be fixed when row level replication is implemented
- Phantom rows will appear in REPEATABLE-READ mode
  - READ-COMMITTED remains without changes
- Limited number of users use this option
- Use when nothing else helps
InnoDB Logging options

- **innodb_log_buffer_size** – buffer for log IO
  - Buffer is flushed on transaction commit or once per second, do not set it too high. 8MB usually enough. 64MB tops.

- **innodb_log_files_in_group** – number of log files
  - Default of 2 is normally good

- **innodb_log_file_size** – size of each of these files
  - Large size faster. Small files more buffer pool flushes
    - page has to be flushed from buffer pool before information about modification is overwritten in log file
  - Larger size increase “redo” recovery time
    - Roll back of uncommitted transaction is faster with large logs
  - Set value which you can afford based on your recovery time up to size of buffer pool.
InnoDB Server Options

- `innodb_max_dirty_pages_pct` - maximum percent of pages in buffer pool Innodb can have dirty
  - Some clean pages are needed to avoid page read stalls
  - Large values typically offer better performance
  - Innodb start do aggressive once it is reached.
  - Lowering value close to 0 makes sense before shutdown
    - makes clean shutdown fast even with large buffer pool

- `innodb_open_files` - number of .ibd files innodb can keep open
  - Only used if you use `innodb_file_per_table`
  - Keep large enough to avoid frequent file reopens
  - No simple way to identify number of reopens of .ibd files, or number of them currently open.
innodb_thread_concurrency

- Sets maximum number of threads can be executing Inside InnoDB kernel
  - Mainly needed when OS makes threads to thrash
  - May significantly improve performance
  - Too low value can significantly decrease performance
    - 10ms sleep before starting wait.
  - Worth to try 1000 to disable queueing
    - Shows best performance results on some workload
  - Reasonable values 2..4*(NumCPU+NumDisks)
  - Group commit works better with higher values
  - Resource consumption (ie additional_mem_pool usage) grows as this value is increased.
Hardware for InnoDB

• Identify if you have CPU bound or disk bound workload
  – Measure CPU consumption during benchmark
• Attempt to fit database (or working set) in memory first
  – No IO system can beat memory in access performance
  – Investment in IO subsystem a lot when this can't be done
• Get Hardware RAID write back cache if you have many small transactions
  – Increase transaction rate without sacrificing durability
• RAID10 is best choice. RAID5 is very slow on writes
• Use large (256K-1M RAID stripe size)
• Get 64bit CPU so all memory can be addressed
  – Opterons, Xeons with EM64T is good choice
OS for InnoDB

• OS should have reliable synchronous disk IO
  – So much data was lost due to broken fsync()
• Good thread library is critical
• DIRECT IO support (bypass OS cache)
• Good Virtual memory subsystem
  – So no swapping even with large buffer Pool
• Current top popular choices: Linux, Solaris, Windows
• Get 64bit OS if you have 64bit CPU
• Use “deadline” scheduler in 2.6 Linux Kernel
• Use journaling file system
  – Avoid long file system check on dirty restart.
DIKU Research

• Partnership between Copenhagen University and MySQL AB, focused on MySQL performance research and improvement
  
• Implemented Native Asynchronous IO support for Linux
  – To be included in MySQL standard release soon

• Implemented Priority IO support for InnoDB and Kernel
  – Increase performance by optimal loading of IO system

• MySQL Cache efficiency research
• IO pattern research Instrumentation
• Christoffer Hall-Frederiksen, visiting conference
  – Talk to him if you’re interested in details
Benchmarks

• Using SysBench benchmark tool
  – http://www.sourceforge.net/projects/sysbench
  – Developed for testing MySQL scalability on OLTP workloads
  – Easy to set up, a lot of options to adjust workload

• Different data access distributions
  – Uniform – all rows have same probability to be accessed
  – Special - highly skewed distribution, typical in real world

• No full detail disclosure due to time constrains

• Results in Transactions/sec - larger numbers better.

• Will be checking:
  – CPU scalability, Number of connections, Data size, Page sizes, File Systems, Kernel Versions
MySQL/InnoDB CPU Scalability

- Sun V40z / 4x 2390MHZ Opteron / Solaris 10/ 8GB RAM
- Read/Write, Special distribution, 1mil rows, 16 threads
- Similar scalability on Linux

<table>
<thead>
<tr>
<th>CPUs</th>
<th>Trans/sec</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>677</td>
<td>1.77</td>
</tr>
</tbody>
</table>

CPU Scalability
Connections Scalability

- Sun V40z / 4x 2390MHZ Opteron / Solaris 10/ 8GB RAM
- 1m rows, Special, Read Only, 4CPU
- Read Only workload has even larger regression

<table>
<thead>
<tr>
<th>Connections</th>
<th>Trans/sec</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1479</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1418</td>
<td>0.96</td>
</tr>
<tr>
<td>256</td>
<td>947</td>
<td>0.64</td>
</tr>
</tbody>
</table>

![Connection Scalability Chart](chart.png)
Data size Scalability

- 4*Xeon 2.0Ghz HT, 4GB RAM, 8 drives RAID10 (SATA)
- Benchmarks with 1, 10, 100m rows on EXT3, 32 threads
- Special and Uniform distributions, Read Write
- Data access pattern role is often underestimated

<table>
<thead>
<tr>
<th>Rows</th>
<th>Special</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m</td>
<td>508</td>
<td>516</td>
</tr>
<tr>
<td>10m</td>
<td>509</td>
<td>71</td>
</tr>
</tbody>
</table>
InnoDB Page sizes

- 4*Xeon 2.0Ghz HT, 4GB RAM, 8 drives RAID10 (SATA)
- 8k, 16k, 32k, 64k page sizes on EXT3, 32 threads
- 1m and 100m row workloads, RW, Uniform
- MySQL Server recompilation needed to change page size
  - Only 16K officially supported

<table>
<thead>
<tr>
<th>Page size</th>
<th>1m</th>
<th>100m</th>
</tr>
</thead>
<tbody>
<tr>
<td>8k</td>
<td>516</td>
<td>19</td>
</tr>
<tr>
<td>16k</td>
<td>516</td>
<td>16.1</td>
</tr>
<tr>
<td>32k</td>
<td>516</td>
<td>12.8</td>
</tr>
</tbody>
</table>
### File Systems

- **4*Xeon 2.0Ghz HT, 4GB RAM, 8 drives RAID10 (SATA)**
- **RH AS 4.0 kernel, elevator=deadline**
- **Read/Write, Special distribution**

<table>
<thead>
<tr>
<th>File Systems</th>
<th>1m</th>
<th>100m</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT3</td>
<td>508</td>
<td>86</td>
</tr>
<tr>
<td>ReiserFS</td>
<td>506</td>
<td>69</td>
</tr>
</tbody>
</table>
Linux Kernel versions

- 4*Xeon 2.0Ghz HT, 4GB RAM, 8 drives RAID10 (SATA)
- Read/Write, Special distribution, EXT3
- 2.4.30 vs 2.6.9, elevator=deadline “vanilla” versions.

File Systems | 1m | 100m
---|---|---
kernel 2.4.30 | 542 | 80

![Kernel versions chart](chart.png)
Resources

- MySQL Online Manual – great source for Information
- SysBench - Benchmark and Stress Test tool
  - [http://sourceforge.net/projects/sysbench](http://sourceforge.net/projects/sysbench)
- MySQL Benchmarks mailing list
  - [benchmarks@lists.mysql.com](mailto:benchmarks@lists.mysql.com)
- Write us your questions if you forgot to ask
  - [peter@mysql.com](mailto:peter@mysql.com)  [tobias@mysql.com](mailto:tobias@mysql.com)
  - Feel free to grab on the conference to discuss your problems