Innodb Architecture and Internals

Peter Zaitsev

Percona Live, Washington DC
11 January 2012

About Presentation

- Brief Introduction in Innodb Architecture
  - This area would deserve many books

Innodb Versions

- MySQL 5.5 Current GA
  – Lots of improvements compared to previous versions
- MySQL 5.6 Current development release
  – Will mention some changes
- Percona Server 5.5
  – Based on MySQL 5.5 with improvements

General Architecture

- Traditional OLTP Engine
  - “Emulates Oracle Architecture”
- Implemented using MySQL Storage engine API
- Row Based Storage. Row Locking. MVCC
- Data Stored in Tablespaces
- Log of changes stored in circular log files
- Data pages as pages in “Buffer Pool”

Storage Files Layout

Physical Structure of Innodb Tabespaces and Logs

Innodb Tablespaces

- All data stored in Tablespaces
  - Changes to these databases stored in Circular Logs
  - Changes has to be reflected in tablespace before log record is overwritten
- Single tablespace or multiple tablespace
  - **innodb_file_per_table=1**
- System information always in main tablespace
  - Main tablespace can consist of many files
    - They are concatenated

Tablespace Format

- Collection of Segments
  - Segment is like a “file”
- Segment is number of extents
  - Typically 64 of 16K page sizes
  - Smaller extents for very small objects
- First Tablespace page contains header
  - Tablespace size
  - Tablespace id

Types of Segments

- Each table is Set of Indexes
  - Innodb has “index organized tables”
- Each index has
  - Leaf node segment
  - Non Leaf node segment
- Special Segments
  - Rollback Segment(s)
  - Insert buffer, etc

Innodb Log Files

- Set of log files (ib_logfile?)
  - 2 log files by default. Effectively concatenated
- Log Header
  - Stores information about last checkpoint
- Log is NOT organized in pages, but records
  - Records aligned 512 bytes, matching disk sector
- Log record format “physiological”
  - Stores Page# and operation to do on it
- Only REDO operations are stored in logs.

Separate Undo Tablespace

- MySQL 5.6 allows to store unto tablespace in separate set of files
  - **innodb_undo_directory**
  - **innodb_undo_tablespaces**
  - **innodb_undo_logs**
- Note once you enable these options you can't downgrade
- Offers another flexibility of using fast storage (such as SSD)

Innodb Threads Architecture

What threads are there and what they do

General Thread Architecture

- Using MySQL Threads for execution
  - Normally thread per connection
- Transaction executed mainly by such thread
  - Little benefit from Multi-Core for single query
- **innodb_thread_concurrency** can be used to limit number of executing threads
  - Reduce contention
- This limit is number of threads in kernel
  - Including threads doing Disk IO or storing data in TMP Table.

Helper Threads

- Main Thread
  - Schedules activities – flush, purge, checkpoint, insert buffer merge
- IO Threads
  - Read – multiple threads used for read ahead
  - Write – multiple threads used for background writes
  - Insert Buffer thread used for Insert buffer merge
  - Log Thread used for flushing the log
- Purge thread(s) (MySQL 5.5 and XtraDB)
- Deadlock detection thread & Others

Memory Handling

How Innodb Allocates and Manages Memory

Memory Allocation Basics

- Buffer Pool
  - Set by **innodb_buffer_pool_size**
  - Database cache; Insert Buffer; Locks
  - Takes More memory than specified
    - Extra space needed for Latches, LRU etc
- Additional Memory Pool
  - Dictionary and other allocations
  - **innodb_additional_mem_pool_size**
    - Not used in newer releases
- Log Buffer (**innodb_log_buffer_size**)

Disk IO

How Innodb Performs Disk IO

Reads

- Most reads done by executing threads
- Read-Ahead performed by background threads
  - Linear
  - Random
  - Do not count on read ahead a lot
- Insert Buffer merge process causes reads

Writes

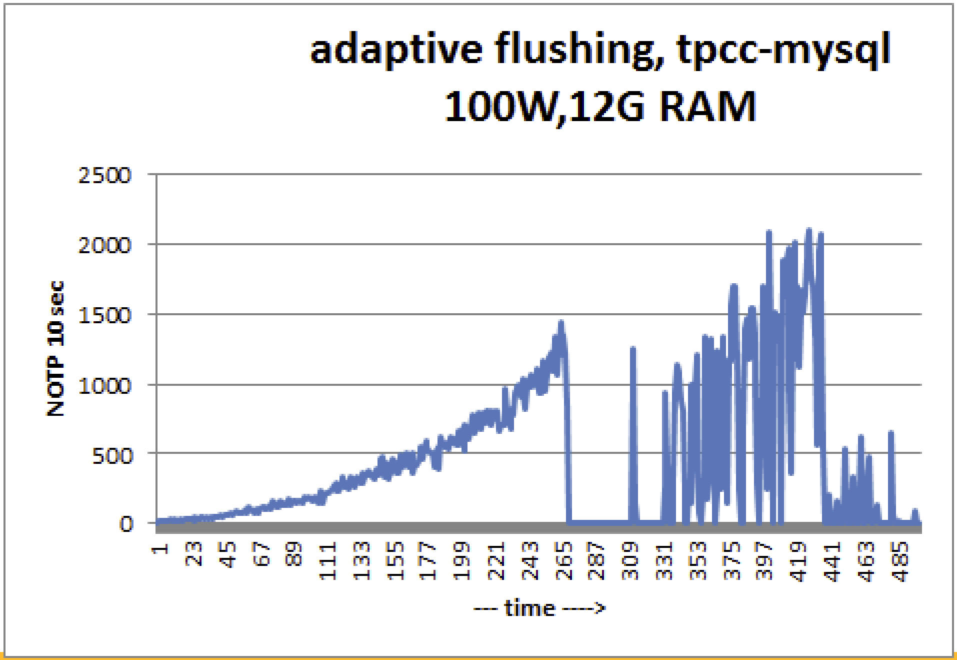
- Data Writes are Background in Most cases
  - As long as you can flush data fast enough you're good
- Synchronous flushes can happen if no free buffers available
- Log Writes can by sync or async depending on **innodb_flush_log_at_trx_commit**
  - 1 – fsync log on transaction commit
  - 0 – do not flush. Flushed in background ~ once/sec
  - 2 – Flush to OS cache but do not call fsync()
    - Data safe if MySQL Crashes but OS Survives

Flush List Writes

- Flushing to advance “earliest modify LSN”
  – To free log space so it can be reduced
- Most of writes typically happen this way
- Number of pages to flush per cycle depended on the load
  – “**innodb_adaptive_flushing**”
  – Percona Server has more flushing modes
    - See **innodb_adaptive_flushing_method**
- If Flushing can't keep up stalls can happen

Example of Misbehavior

- Data fits in memory and can be modified fast
  - Yet we can't flush data fast enough
- Working on solution in XtraDB



LRU Flushes

- Can happen in workloads with data sets larger than memory
- If Innodb is unable to find clean page in 10% of LRU list
- LRU Flushes happen in user threads
- Hard to see exact number in standard Innodb
  - XtraDB adds **Innodb_buffer_pool_pages_LRU_flushed**

LRU Flushes in MySQL 5.6

- MySQL 5.6 adds “page_cleaner” to avoid LRU flushes in User Threads
- **innodb_lru_scan_depth=N**
  - Controlls how deeply page cleaner will examine Tail of LRU for dirty pages
  - Happens once per second

Page Checksums

- Protection from corrupted data
  - Bad hardware, OS Bugs, Innodb Bugs
  - Are not completely replaced by Filesystem Checksums
- Checked when page is Read to Buffer Pool
- Updated when page is flushed to disk
- Can be significant overhead
  - Especially for very fast storage
- Can be disabled by **innodb_checksums=0**

Double Write Buffer

- Innodb log requires consistent pages for recovery
- Page write may complete partially
  - Updating part of 16K and leaving the rest
- Double Write Buffer is short term page level log
- The process is:
  - Write pages to double write buffer; Sync
  - Write Pages to their original locations; Sync
  - Pages contain tablespace_id+page_id
- On crash recovery pages in buffer are compared to their original location

Direct IO Operation

- Default IO mode for Innodb data is **Buffered**
- Good
  - Faster flushes when no write cache
  - Faster warmup on restart
  - Reduce problems with inode locking on EXT3
- Bad
  - Lost of effective cache memory due to double buffering
  - OS Cache could be used to cache other data
  - Increased tendency to swap due to IO pressure
- **innodb_flush_method=O_DIRECT**

Log IO

- Log are opened in buffered mode
  - Even with **innodb_flush_method=O_DIRECT**
  - XtraDB can use O_DIRECT for logs
    - **innodb_flush_method=ALL_O_DIRECT**
- Flushed by fsync() - default or O_SYNC
- Logs are often written in 512 byte blocks
  - **innodb_log_block_size=4096 in XtraDB**
- Logs which fit in cache may improve performance
  - Small transactions and **innodb_flush_log_at_trx_commit=1 or 2**

Indexes

How Indexes are Implemented in Innodb

Everything is the Index

- Innodb tables are “Index Organized”
  - PRIMARY KEY contains data instead of data pointer
- Hidden PRIMARY KEY is used if not defined (6b)
- Data is “Clustered” by PRIMARY KEY
  - Data with close PK value is stored close to each other
  - Clustering is within page ONLY
- Leaf and Non-Leaf nodes use separate Segments
  - Makes IO more sequential for ordered scans
- Innodb system tables **SYS_TABLES** and **SYS_INDEXES** hold information about index “root”

Index Structure

- Secondary Indexes refer to rows by Primary Key
  - No update when row is moved to different page
- Long Primary Keys are expensive
  - Increase size of all Indexes
- Random Primary Key Inserts are expensive
  - Cause page splits; Fragmentation
  - Make page space utilization low
- AutoIncrement keys are often better than artificial keys, UUIDs, SHA1 etc.

Multi Versioning

Implementation of Multi Versioning and Locking

Multi Versioning at Glance

- Multiple versions of row exist at the same time
- Read Transaction can read old version of row, while it is modified
  - No need for locking
- Locking reads can be performed with SELECT FOR UPDATE and LOCK IN SHARE MODE Modifiers

Transaction isolation Modes

- SERIALIZABLE
  - Locking reads. Bypass multi versioning
- REPEATABLE-READ (default)
  - Read commited data at it was on start of transaction
- READ-COMMITED
  - Read commited data as it was at start of statement
- READ-UNCOMMITED
  - Read non committed data as it is changing live

Updates and Locking Reads

- Updates bypass Multi Versioning
  - You can only modify row which currently exists
- Locking Read bypass multi-versioning
  - Result from SELECT vs SELECT .. LOCK IN SHARE MODE will be different
- Locking Reads are slower
  - Because they have to set locks
  - Can be 2x+ slower !
  - SELECT FOR UPDATE has larger overhead

Multi Version Implementaition

- The most recent row version is stored in the page
  – Even before it is committed
- Previous row versions stored in undo space
  – Located in System tablespace
- The number of versions stored is not limited
  – Can cause system tablespace size to explode.
- Access to old versions require going through linked list
  – Long transactions with many concurrent updates can impact performance.

Multi Versioning Indexes

- Indexes contain pointers to all versions
  - Index key 5 will point to all rows which were 5 in the past
- Indexes contain TRX_ID
  - Easy to check entry is visible
  - Can use "Covering Indexes"
- Many old versions is performance problem
  - Slow down accesses
  - Will leave many "holes" in pages when purged

Cleaning up the Garbage

- Old Row and index entries need to be removed
  - When they are not needed for any active transaction
- REPEATABLE READ
  - Need to be able to read everything at transaction start
- READ-COMMITED
  - Need to read everything at statement start
- Purge Thread(s) may be unable to keep up with intensive updates
  - Innodb “History Length” will grow high
- **innodb_max_purge_lag** slows updates down
  - Not very reliable

Innodb Locking

How Innodb Locking Works

Innodb Locking Basics

- Pessimistic Locking Strategy
- Graph Based Deadlock Detection
  - Takes shortcut for very large lock graphs
- Row Level lock wait timeout
  - **innodb_lock_wait_timeout**
- Traditional “S” and “X” locks
- Intention locks on tables “IS” “IX”
  - Restricting table operations
- Locks on Rows AND Index Records
- No Lock escalation

Gap Locks

- Innodb does not only locks rows but also gap between them
- Needed for consistent reads in Locking mode
  – Also used by update statements
- Innodb has no Phantoms even in Consistent Reads
- Gap locks often cause complex deadlock situations
- “infinum”, “supremum” records define bounds of data stored on the page
  – May not correspond to actual rows stored
- Only record lock is needed for PK Update

Lock Storage

- Innodb locks storage is pretty compact
  – This is why there is no lock escalation !
- Lock space needed depends on lock location
  – Locking sparse rows is more expensive
- Each Page having locks gets bitmap allocated for it
  – Bitmap holds lock information for all records on the page
- Locks typically take 3-8 bits per locked row

Auto Increment Locks

- Major Changes in MySQL 5.1 !
- MySQL 5.0 and before
  - Table level AUTO_INC lock for duration of INSERT
  - Even if INSERT provided key value !
  - Serious bottleneck for concurrent Inserts
- MySQL 5.1 and later
  - **innodb_autoinc_lock_mode –** set lock behavior
  - "1" - Does not hold lock for simple Inserts
  - "2" - Does not hold lock in any case.
    - Only works with Row level replication

Latching

Innodb Internal Locks

Innodb Latching

- Innodb implements its own Mutexes and RW-Locks
  - For transparency not only Performance
- Latching stats shown in SHOW INNODB STATUS

```
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 13569, signal count 11421
--Thread 1152170336 has waited at ./../include/buf0buf.ic line 630 for 0.00 seconds the semaphore:
Mutex at 0x2a957858b8 created file buf0buf.c line 517, lock var 0
waiters flag 0
wait is ending
--Thread 1147709792 has waited at ./../include/buf0buf.ic line 630 for 0.00 seconds the semaphore:
Mutex at 0x2a957858b8 created file buf0buf.c line 517, lock var 0
waiters flag 0
wait is ending
Mutex spin waits 5672442, rounds 3899888, OS waits 4719
RW-shared spins 5920, OS waits 2918; RW-excl spins 3463, OS waits 3163
```

Latching Performance

- Was improving over the years
- Still is problem for certain workloads
  - Great improvements in MySQL 5.5,5.6 & XtaDB
  - Still hotspots remain
- **innodb_thread_concurrency**
  - Limiting concurrency can reduce contention
  - Introduces contention on its own
- **innodb_sync_spin_loops**
  - Trade Spinning for context switching
  - Typically limited production impact

Page Replacement

Page Replacement Flushing and Checkpointing

Basic Page Replacement

- Innodb uses LRU for page replacement
  - With Midpoint Insertion
- Innodb Plugin and XtraDB configure
  - **innodb_old_blocks_pct, innodb_old_blocks_time**
  - Offers Scan resistance from large full table scans
- Scan LRU Tail to find clean block for replacement
- May schedule synchronous flush if no clean pages for replacement

Checkpointing

- Fuzzy Checkpointing
  - Flush few pages to advance min unflushed LSN
  - Flush List is maintained in this order
- MySQL 5.1 often has “hiccups”
  - No more space left in log files. Need to wait for flush to complete
- Percona Patches for 5.0 and XtraDB
  - Adaptive checkpointing: **innodb_adaptive_checkpoint**
- Innodb Plugin **innodb_adaptive_flushing**
  - Best behavior depends on worload

Percona Live DC Sponsors

Media Sponsor

Friends of Percona Sponsor

MySQL Conference & Expo 2012
Presented by Percona Live

The Hyatt Regency Santa Clara & Santa Clara Convention Center

April 10th-12th, 2012

Featured Speakers

Mark Callaghan (Facebook), Jeremy Zawodny (Craigslist), Marten Mickos (Eucalyptus Systems)

Sarah Novotny (Blue Gecko), Peter Zaitsev (Percona), Baron Schwartz (Percona)

Learn More at www.percona.com/live/

Want More MySQL Training?

Percona Training in Washington DC Next Week

January 16th-19th, 2012

MySQL Workshops

MySQL Developers Training - Monday

MySQL DBA Training - Tuesday

MySQL InnoDB / XtraDB - Wednesday

MySQL Operations – Thursday

Use Discount code DCPL30 to save 30%

Visit http://bit.ly/dc-training

Upcoming Percona MySQL Training

Washington, DC – January 16, 2012

Vancouver, Canada - February 6, 2012

Frankfurt, Germany - February 13, 2012

Irvine, CA – February 14, 2012

Denver, CO - February 20, 2012

San Francisco, CA - March 12, 2012

Austin, TX - March 19, 2012

Visit http://percona.com/training

Percona Live MySQL Conference, April 10-12 , Santa Clara,CA

PERCONA LIVE

www.percona.com/live