Innodb and XtraDB Architecture and Performance Optimization

Apr 11-14, 2011
O'Reilly MySQL Conference and Expo
Santa Clara, CA
by Peter Zaitsev, Percona Inc
Architecture and Performance

- Advanced Performance Optimization requires transparency
  - X-ray vision
- Impossible without understanding system architecture
- Focus on Conceptual Aspects
  - Exact Checksum algorithm InnoDB uses is not important
  - What matters
    - How fast is that algorithm?
    - How checksums are checked/updated
Aspects or Architecture

- General Architecture
- Storage and File Layout
- Threads
- Memory
- Disk IO
- Logging
- Indexes
- Multi Versioning
- Row Locking
- Latching
Aspects of Architecture 2

- Page flushing and Replacement
- Insert Buffering
- Adaptive Hash Index
- BLOB Storage
- Recovery
- Compression Features
- Foreign Keys
Innodb Versions

- MySQL 5.1 and below
  - Lots of limits. Poor Scalability.
- Innodb Plugin for MySQL 5.1 (1.0.x)
  - Scales Better, Fast index creation, Compression
- MySQL 5.5 (version 1.1.x)
  - Scalability further improved
- XtraDB
  - Based on Innodb Plugin 5.1 and version in 5.5
  - Has added features, performance improvements
  - Available in Percona Server and MariaDB
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 Tablespaces 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
  – `ibdata1`
  – 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 Space Allocation

- Small Segments (less than 32 pages)
  - Page at the time
- Large Segments
  - Extent at the time (to avoid fragmentation)
- Free pages recycled within same segment
- All pages in extent must be free before it is used in different segment of same tablesapce
  - `innodb_file_per_table=1` - free space can be used by same table only
- Innodb never shrinks its tablesapces
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.
More on Log Files

• Total log file size is limited to 4GB
  – In Percona Server this limit is removed
• Percona Server allows different log file block size
  – `innodb_log_block_size`
• If you're using compressed pages full pages can be logged to log file.
• Dramatic Recovery time improvement in Innodb Plugin
  – Can safely use longer log files than before
Storage Tuning Parameters

- `innodb_file_per_table`
  - Store each table in its own file/tablespace
- `innodb_autoextend_increment`
  - Extend `system` tablespace in this increment
- `innodb_log_file_size`
- `innodb_log_files_in_group`
  - Log file configuration
- `innodb_page_size`
  - Percona Server only
Using to File per Table

- Typically more convenient
- Reclaim space from dropped table
- **ALTER TABLE ENGINE=INNODB**
  - reduce file size after data was deleted
- Store different tables/databases on different drives
- Backup/Restore tables one by one
- Support for compression in Innodb Plugin/XtraDB
- Will use more space with many tables
- Longer unclean restart time with many tables
- Performance is typically similar
Drop Table with innodb_file_per_table

- Dropping the tablespace is expensive operation in InnoDB
  - And gets slower the more memory you have
  - Drop operation have to scan buffer pool and remove all pages
  - It is done while holding the lock, essentially blocking server
  - See http://bugs.mysql.com/bug.php?id=51325

- Option in XtraDB
  - innodb_lazy_drop_table=1
  - Do not remove blocks on drop, do it as they are replaced from LRU list.
Dealing with Run-away tablespace

- Main Tablespace does not shrink
  - Consider setting max size
  - `innodb_data_file_path=ibdata1:10M:autoextend:max:10G`

- Dump and Restore

- Export tables with XtraBackup
  - And import them into “clean” server
Resizing Log Files

- You can't simply change log file size in my.cnf
  - InnoDB: Error: log file ./ib_logfile0 is of different size 0 5242880 bytes
  - InnoDB: than specified in the .cnf file 0 52428800 bytes!

- Stop MySQL (make sure it is clean shutdow)
- Rename (or delete) ib_logfile*
- Start MySQL with new log file settings
  - It will create new set of log files
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 (MySQL 5.5 and XtraDB)**
- **Deadlock detection thread.**
- **Monitoring Thread**
Memory Handling

How Innodb Allocates and Manages Memory
Innodb Memory Allocation

• Take a look at **SHOW INNODB STATUS**
  – XtraDB has more details

  Total memory allocated 1100480512; in additional pool allocated 0
  Internal hash tables (constant factor + variable factor)
    Adaptive hash index 17803896  (17701384 + 102512)
    Page hash 1107208
  Dictionary cache 8089464  (4427312 + 3662152)
  File system 83520  (82672 + 848)
  Lock system 2657544  (2657176 + 368)
  Recovery system 0  (0 + 0)
  Threads 407416  (406936 + 480)
  Dictionary memory allocated 3662152
  Buffer pool size 65535
  Buffer pool size, bytes 1073725440
  Free buffers 64515
  Database pages 1014
  Old database pages 393
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`
Configuring Innodb Memory

• `innodb_buffer_pool_size` is the most important
  – Use all your memory nor committed to anything else
  – Keep overhead into account (~5%)
  – Never let Buffer Pool Swapping to happen
  – Up to 80-90% of memory on Innodb only Systems

• `innodb_log_buffer_size`
  – Values 8-64MB typically make sense
  – Larger values may improve contention
  – May need to be larger if using large BLOBs
  – See number of data written to the logs
  – Log buffer covering 10sec is good enough
Ever wondered what is in BP?

- Check out INNODB_BUFFER_POOL_* tables
  - Available in XtraDB

```sql
mysql> select count(*), sum(dirty=1) from INNODB_BUFFER_POOL_PAGES_INDEX where index_id in(31,32);
+----------------------|
| count(*) | sum(dirty=1) |
|-----------+-----------|
|   40      |     22    |
+----------------------+
1 row in set (0.00 sec)
```
Data Dictionary

• Holds information about InnoDB Tables
  – Statistics; Auto Increment Value, System information
  – Can be 4-10KB+ per table
• Can consume a lot of memory with huge number of tables
  – Think hundreds of thousands
• \texttt{innodb\_dict\_size\_limit}
  – Limit the size in Percona Server
  – Make it act as a real cache
Disk IO

How Innodb Performs Disk IO
Reads

- Most reads done by executing threads
- Read-Ahead performed by background threads
  - Linear
  - Random (removed in later versions)
  - 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`
Merging Neighbor Pages

- To make IO more Sequential Innodb will look for neighbor pages and flush them again
  - It is ALL “old” pages in the page proximity (+- 32 pages)
    - It does not have to be sequential range of pages
- Such behavior may be very poor choice
  - Especially for SSD which do not have random IO Penalty
- XtraDB has option
  - \texttt{innodb\_flush\_neighbor\_pages}
  - Working on option to flush sequential pages only
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`
  - Not Recommended for Production
  - `innodb_fast_checksum` in XtraDB
    - Not compatible format. Several times faster.
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
Disabling Double Write

- Overhead less than 2x because write is sequential
- Relatively larger overhead on SSD;
  - Also impacts Flash Life time
- Can be disabled if FS guaranties atomic writes
  - ZFS
- innodb_doublewrite=0
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 blocks less than 4K
  - Read has to happen before write
- 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 need to 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.
SYS_TABLES Example

- Table can be viewed in XtraDB:

```sql
mysql> select * from INNODB_SYS_TABLES limit 10;
+----------------+-------------+----------------+----+----------+----------+--------+
| TABLE_ID       | SCHEMA      | NAME           | FLAG| N_COLS   | SPACE    |
|----------------+-------------+----------------+----+----------+----------+--------|
| 11             | SYS_FOREIGN |                | 0  | 7        | 0        |
| 12             | SYS_FOREIGN_COLS |            | 0  | 7        | 0        |
| 17             | percona     | transactions   | 1  | 18       | 0        |
| 23             | sbtest      | sbtest#P#p0    | 1  | 7        | 0        |
| 24             | sbtest      | sbtest#P#p1    | 1  | 7        | 0        |
| 25             | sbtest      | sbtest#P#p2    | 1  | 7        | 0        |
| 26             | sbtest      | sbtest#P#p3    | 1  | 7        | 0        |
| 18             | stats       | tables         | 1  | 12       | 0        |
| 62             | test        | a              | 41 | 5        | 12       |
| 55             | test        | btest          | 1  | 5        | 0        |
+----------------+-------------+----------------+----+----------+----------+--------+
10 rows in set (0.00 sec)
```
SYS_INDEXES example

- Available in XtraDB too

```sql
mysql> select * from INNODB_SYS_INDEXES where table_id=23;
+-------------------------------------+----------+-------+-----+---------+--------+-------+
| INDEX_ID | NAME   | TABLE_ID | TYPE | N_FIELDS | PAGE_NO | SPACE |
+-------------------------------------+----------+-------+-----+---------+--------+-------+
| 31       | PRIMARY | 23      | 3    | 1       | 98306  | 0     |
| 32       | k       | 23      | 0    | 1       | 98307  | 0     |
+-------------------------------------+----------+-------+-----+---------+--------+-------+
2 rows in set (0.00 sec)
```
More on Clustered Index

• PRIMARY KEY lookups are the most efficient
  – Secondary key lookup is essentially 2 key lookups
  – Optimized with Adaptive Hash Index

• PRIMARY KEY ranges are very efficient
  – Build Schema keeping it in mind
  – (user_id,message_id) may be better than (message_id)

• Changing PRIMARY KEY is expensive
  – Effectively removing row and adding new one.

• Sequential Inserts give compact, least fragmented storage
  – ALTER TABLE tbl=INNODB can be optimization
More on Indexes

- There is no Prefix Index compressions
  - Index can be 10x larger than for MyISAM table
  - Innodb has page compression. Not the same thing.
- Indexes contain transaction information = fat
  - Allow to see row visibility = index covering queries
- Secondary Keys built by insertion
  - Often outside of sorted order = inefficient
- Innodb Plugin and XtraDB building by sort
  - Faster
  - Indexes have good page fill factor
  - Indexes are not fragmented
Fragmentation

• Inter-row fragmentation
  – The row itself is fragmented
  – Happens in MyISAM but NOT in InnoDB
• Intra-row fragmentation
  – Sequential scan of rows is not sequential
  – Happens in InnoDB, outside of page boundary
• Empty Space Fragmentation
  – A lot of empty space can be left between rows
• ALTER TABLE tbl ENGINE=INNODB
  – The only medicine available.
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 committed data at it was on start of transaction
- **READ-COMMITTED**
  - Read committed data as it was at start of statement
- **READ-UNCOMMITTED**
  - 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 Implementation

• 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 tablesapce
• The number of versions stored is not limited
  – Can cause system tablesapce size to explode.
• Access to old versions require going through linked list
  – Long transactions with many concurrent updates can impact performance.
Multi-Versioning Internals

- Each row in the database has
  - `DB_TRX_ID (6b)` – Transaction inserted/updated row
  - `DB_ROLL_PTR (7b)` - Pointer to previous version
  - Significant extra space for short rows!
- Deletion handled as Special Update
- `DB_TRX_ID + list of currently running transactions` is used to check which version is visible
- Insert and Update Undo Segments
  - Inserts history can be discarded when transaction commits.
  - Update history is used for MVCC implementation
Undo Segment Limits

- Undo Segment was often limited factor
- InnoDB Plugin and Before
  - Max 1024 undo segments
  - May cap at 512 active transactions
    - Transaction may require 2 undo segments
- Increased in XtraDB to 4072
- MySQL 5.5 Increases it to some 128K
Multi Versioning Performance

• Short rows are faster to update
  – Whole rows (excluding BLOBs) are versioned
  – Separate table to store counters often make sense

• Beware of long transactions
  – Especially containing many updates

• “Rows Read” can be misleading
  – Single row may correspond to scanning thousand of versions/index entries
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-COMMITTED
  - Need to read everything at statement start
- Purge Thread 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
Handling Blobs

- Blobs are handled specially by Innodb
  - And differently by different versions
- Small blobs
  - Whole row fits in ~8000 bytes stored on the page
- Large Blobs
  - Can be stored full on external pages (Barracuda)
  - Can be stored partially on external page
    - First 768 bytes are stored on the page (Antelope)
- Innodb will NOT read blobs unless they are touched by the query
  - No need to move BLOBs to separate table.
Innodb BLOG != MySQL BLOG

- MySQL Has limit of 65535 bytes per row excluding BLOB and TEXT column
  - This limit applies to VARCHAR() columns
- Innodb limit is only 8000 (half a page)
  - So long VARCHAR fields may be stored as a BLOB inside Innodb

```sql
mysql> create table ai(c varchar(40000), d varchar(40000));
ERROR 1118 (42000): Row size too large. The maximum row size for the used table type, not counting BLOBs, is 65535. You have to change some columns to TEXT or BLOBs
```
Blob Allocation

- Each BLOB Stored in separate segment
  - Normal allocation rules apply. By page when by extent
  - One large BLOB is faster than several medium ones
  - Many BLOBs can cause extreme waste
    - 500 byte blobs will require full 16K page if it does not fit with row
- External BLOBs are NOT updated in place
  - Innodb always creates the new version
- Large VARCHAR/TEXT are handled same as BLOB
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
Types of Locks in InnoDB

- **Next-Key-Lock**
  - Lock Key and gap before the key
- **Gap-Lock**
  - Lock just the gap before the key
- **Record-Only-Lock**
  - Lock record only
- **Insert intention gap locks**
  - Held when waiting to insert into the gap
Advanced Gap Locks Stuff

• Gaps can change on row deletion
  – Actually when Purge thread removes record
• Leaving conflicting Gap locks held
• Gap Locks are “purely inhibitive”
  – Only block insertion.
  – Holding lock does not allow insertion. Must also wait for conflicting locks to be released
• “supremum” record can have lock, “infinum” can't
• This is all pretty complicated and you rarely need it in practice
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
Latchng

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
Latch Performance

- Was improving over the years
- Still is problem for certain workloads
  - Great improvements in MySQL 5.5 & 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
Current Hotspots

- **kernel_mutex**
  - A lot of operations use global kernel mutex
- **log_mutex**
  - Writing data to the log buffer
- **Index->lock**
  - Lock held for duration of low level index modification
  - Can be serious hot spot for heavy write workloads
- **Adaptive has latch**
  - Global latch. Problem with heavy read/write mix
  - `innodbadaptive_hash_index=0`
    - Slow things down but reduce contention
Recent Achievements

- MySQL 5.5 support for multiple buffer pools
  - `innodb_buffer_pool_instances=8`
  - Reduces a lot of buffer pool related contention
  - Make sure each buffer pool is at least 1GB in size
- Partitioned Adaptive Index in recent XtraDB
  - `innodb_adaptive_hash_index_partitions=8`
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
Page Flushing

- Scheduled by Main Thread in Background
  - Keep portion of the pages clean
  - Make sure we have log space
- `innodb_io_capacity`
  - Amount of writes per second server can do
  - Affects number of background flushes and insert buffer merges (5% for each)
- Server will do merges and flushes faster when it is idle
Maintaining clean pages

- **innodb_max_dirty_pages_pct**
  - Default 90, later 75
- Innodb will start flushing pages faster if it is reached
  - This is not the hard limit
- Value 0 is helpful for Fast Shutdown
  - Set to 0 and wait until number of dirty pages is low
- Innodb looks for next/prev dirty pages and flushes it as well to keep IO more bulky
  - Can be harmful for SSD storage
  - Controlled by `innodb_flush_neighbor_pages` in XtraDB
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 workload
Recovery

How Innodb Recovers from Crash
Recovery Stages

- Physical Recovery
  - Recover partially written pages from double write buffer
- Redo Recovery
  - Redo all the changes stored in transactional logs
- Undo Recovery
  - Roll back not committed transactions
Redo Recovery

- Foreground
  - Server is not started until it is complete
- Larger Logs = Longer recovery time
  - Though row sizes, database size, workload also matter
- Scan Log files
  - Buffer modifications on per page basics
  - Apply modifications to data file
- LSN stored in the page tells if change needs to be applied
Tuning Redo recovery

- `innodb_log_file_size` - large logs longer recovery
- `innodb_max_dirty_pages_pct` – Fewer dirty pages faster recovery
- `innodb_buffer_pool_size` – Larger buffer faster IO recovery
  - Bug from 2007 which makes recovery slower with large buffer pool
  - Fixed in InnoDB Plugin, MySQL 5.5
  - Solution exists in Percona Patches for 5.0
Undo Recovery

- Is Background since MySQL 5.0
  - Performed after MySQL is started
- Speed depends on transaction length
  - Very large UPDATE, INSERT... SELECT is problem.
- Is NOT problem with ALTER TABLE
  - Commits every 10000 rows to avoid this problem
  - Unless it is Partitioned table
- Faster with larger `innodb_log_file_size`
- Be careful killing MySQL with run away update queries.
Advanced Features

Insert Buffering, Adaptive Hash Index, Foreign Keys, Compression
Insert Buffer

- Designed to speed up Inserts into large Indexes
  - Reported up to 15 times IO reduction for some cases
- Works for Non-Unique Secondary Indexes only
- If leaf index page is not in buffer pool
  - Store a note the page should be updated in memory
- If page containing buffered entries is read from disk they are merged transparently
- Innodb performs gradual insert buffer merges in background
Change buffer in MySQL 5.5

- Buffer not only Insert but also Update and Purge operations
  - Delete is covered as it is special update on the low level
- Can improve bulk update/delete 10x or more
- Read for more details
Insert Buffer Problems

- Can take up to half of buffer pool size
  - Persists in tablespace to keep things safe
  - `innodb_ibuf_max_size` in XtraDB to restrict it
  - Full Insert Buffer is useless and wastes memory
- Delayed Insert Buffer merge can cause slowdown
  - Too many merges need to happen on page reads
- Background merge speed may not be enough
  - Tune by `innodb_io_capacity, innodb_ibuf_accel_rate`
- After Restart Merge speed can slow down
  - Finding index entries to merge needs random IO
More tuning of Insert Buffer

- Innodb Plugin, XtraDB you can disable insert buffering
  - `innodb_change_buffering=0`
  - Can be good for SSDs
Adaptive Hash Index

- Built on top of existing BTREE Indexes to speed up lookups
  - Both PRIMARY and Secondary indexes
- Can be built for full index and prefixes
- Partial Index
  - Only built for index values which are accessed often

Hash table size 8850487, used cells 2381348, node heap has 4091 buffer(s)
2208.17 hash searches/s, 175.05 non-hash searches/s
Tuning Adaptive Hash Index

- Self tuning
  - No tuning options are available.
- Can be disabled for performance reasons
  - `innodb_adaptive_hash_index`
  - Improves concurrency but reduces performance
- Can be Partitioned in newer XtraDB versions
  - `innodb_adaptive_hash_index_partitions=8`
Foreign Keys

- Implemented on Innodb level
- Require indexes on both tables
  - Can be very expensive sometimes
- Checks happen when row is modified
  - No delayed checks till transaction commit
- Foreign Keys introduce additional locking overhead
  - Many tricky deadlock situations are foreign key related
Compression

- New in Innodb Plugin and XtraDB
  - Requires “Barracuda” and `innodb_file_per_table=1`
- Per Page compression (mostly)
- Uses zlib for compression (no settings available)
- Uses fancy tricks
  - Per page update log to avoid re-compression
  - Both Compressed and Uncompressed page can be stored in Buffer Pool
- `ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;`
  - Estimate how well the data will compress
Problems with Compression

- Filesystem level compression may be more efficient
  - ZFS
- Page size is too small for good compression
- Have to “Guess” Compression
- Compression setting is Per table
  - Though some indexes compress better than others
- `KEY_BLOCK_SIZE=16;`
  - Only compress externally stored BLOBs
  - Can reduce size without overhead
Fast Warmup

- Warmup can take very long time on restart
  - Especially with large amounts of memory
- XtraDB contains
  - `innodb_lru_dump=300`
  - Dump list of pages in LRU list
  - Will re-populate buffer pool with list on restart
  - Can improve warmup time 10x
Thanks for Coming

• Questions ? Followup ?
  – pz@percona.com

• Yes, we do MySQL and Web Scaling Consulting
  – http://www.percona.com

• Check out our book
  – Complete rewrite of 1st edition