

Life of a dirty page InnoDB disk IO

Mark Callaghan

URLs

- slides at <http://tinyurl.com/dzshl2>
- v3 Google patch at code.google.com
- many similar features in Percona binaries
- soon to be many similar features in MySQL 5.4

Putting things in context

How to make a great OLTP engine

- Go back in time about 15 years
- Start with the Gray & Reuter text
- Develop on a typical PC with one disk and one CPU

Then servers with many cores and 10k IOPs arrive.

I will focus on the design and implementation of official InnoDB.

Questions

- Should InnoDB switch to 64kb pages?
 - The 5 Minute Rule suggests it should
 - Nobody has published performance results for this
- Can InnoDB effectively use 10,000 IOPs?
 - Yes for read-intensive workloads
 - Not yet for write-intensive workloads

High-level architecture

- Some IO done directly in user threads (mostly reads)
- Background threads perform simulated AIO for
 - Log writes
 - Dirty page writes
 - Prefetch reads
- Extent is 64 16kb pages
- Adjacent requests are merged
- Rate limits based on capacity of single-disk server

Medium-level architecture

Background thread tasks (`srv_master_thread`)

- Once per second
- Once per 10 seconds
- Only when idle

Tasks to do once per second

- Force transaction log to disk
- Make space available in tx log buffer (maybe)
- Fuzzy checkpoint (maybe)
- Read up to 5 pages to merge insert buffer records using sync IO. Skipped unless server is somewhat idle.
- Write 100 dirty pages from the buffer pool using async IO. Select pages from the flush list. Skipped when there are too many dirty pages. Restart the task loop immediately if this is done.

Tasks to do once per 10 seconds

- Write 100 or 10 dirty pages from the buffer pool using async IO while selecting pages from the flush list. Skipped unless the server is somewhat idle.
- Read up to 5 pages to merge insert buffer records using sync IO
- Force transaction log to disk
- Remove deleted rows
- Fuzzy checkpoint

Tasks to do when the server is idle

- Remove deleted rows
- Read up to 20 pages to merge insert buffer records using sync IO
- Write 100 dirty pages from the buffer pool using sync IO. Select pages from the flush list.
- Fuzzy checkpoint
- Repeat if server still idle

Processing IO requests (os0file.c)

Request arrays

- 256 slots per array
- Arrays for insert buffer, tx log, page writes, prefetch
- One thread does all of the IO per array using sync IO

Choosing the next request

- Requests older than 2 seconds get priority
- Otherwise choose the request with the lowest file offset when considering the low 32 bits of the offset
- This requires 2 passes of the request array

Processing IO requests (2)

Request merging

- Merge requests for adjacent blocks. Alas, this does one pass on the array per adjacent block merged.
- Stop trying to merge when there is no pending request for the next block or 64 requests have been merged.
- Allocate a temporary buffer for the multi-block IO when requests have been merged.
- Use one read/write system call. This is a large IO operation when merging is used. Wait for entire request to complete.

Processing IO requests (3)

Request completion

- Remove the request for the first block in the merged request (or only block) from the IO array.
- Completion for other requests is done on the next iteration of the request handling loop.

Page write protocol

Protect against partial page writes by writing each dirty page first to a serial log before updating it in place. This can be disabled for copy-on-write filesystems.

Important functions are

- `buf_flush_post_to_doublewrite_buf`
- `buf_flush_batch`
- `buf_flush_buffered_writes`

Page write protocol (2)

- First write page to in-memory doublewrite log
- When doublewrite log is full
 - Write doublewrite log to a file (many pages, 1 big IO).
This file is the *doublewrite buffer* .
 - Force writes for doublewrite buffer to disk.
 - Submit async IO requests to update pages just logged in the doublewrite buffer (1 write per page).
 - Wait for writes to complete.
 - Force writes for dirty pages to disk.

Prefetch for random access

- `buf_read_ahead_random`
- There are 64 blocks per extent
- Before requesting a block read:
 - Count number of blocks in extent that were recently read based on position in buffer pool LRU list.
 - If more than 13 were recently read, prefetch others

Prefetch for sequential access

- `buf_read_ahead_linear`
- May be used when
 - Accessing first or last page in extent
 - Many (24) pages in extent have been accessed
 - Access pattern was sequential
- Checks are made prior to issuing a read request.
- When used, issue read requests for extent that contains that page that follows or precedes this extent.

Insert buffer (ibuf0ibuf.c)

- Persistent and index-organized on (space id, page#)
- Allows changes to secondary index leaf blocks to be deferred when the block is not in the buffer pool.
- Used for insert and the insert portion of an update
- Used until the insert buffer is half of the buffer pool
- Fully resident in the buffer pool

Merging insert buffer records

- `ibuf_contract_ext`
- Choose random page from the insert buffer
- Open a cursor on a random record on that page
- Read insert buffer entries from that cursor to find at most 8 pages that should be fetched.
- Issue async IO requests
- When async read IO completes, callback is done to apply deferred changes
- (Usually) wait for IO to complete

Buffer pool (buf0buf.h)

- Protected by one mutex
- Contains
 - Flush list - ordered by oldest modification LSN
 - LRU list
 - ordered by approximate access time
 - I have not found code to guard against wiping out LRU list and buffer pool on large table scan
 - Hash table for pages in buffer pool

More on writing dirty pages

- Most pages to be written selected from end of flush list
 - min(modification LSN from any page) must be greater than the min LSN from the current log files. Pages flushed to enforce this.
 - If a page is written from the LRU list it is moved to the free list on IO completion.
- When a page is selected, other dirty pages in the same extent are also selected.

Problems (PA == patch available)

- Rate limits for background IO are based on a single-disk server (PA)
- Only one thread per IO request array (PA)
- Insert buffer merge rate limit is much too small (PA)
- Dirty page writing rate limit is much too small (PA)
- Request arrays are too small for high IOPs servers (PA)
- Request array iteration uses too much CPU (PA)
- BUF_READ_AHEAD_AREA uses too much CPU
- SHOW STATUS and SHOW INNODB STATUS need more statistics (PA)

Problems (PA == patch available)

- InnoDB not good at enforcing max dirty pages (PA)
- No option for enforcing max insert buffer size (PA)
- Limited visibility into utility of prefetching
- Prefetch requests for sequential access are submitted late
- When IO requests are merged, must wait for large IO to complete before any blocks can be used

IO performance problems

- Insert buffer useless when full, but not emptied quickly
- Updates and inserts are much slower without it
- Run the insert benchmark (iibench.py)
 - 2 disk server, SW RAID 0, XFS
 - 1 GB InnoDB buffer pool, O_DIRECT
 - Start with 50M rows in the table, restart test
 - Eventually unpatched InnoDB does 25 inserts/sec versus 200 inserts/sec for the v3 Google patch
 - Result has been repeated on larger servers

IO Performance problems

