InnoDB Change Buffer: Unsafe at Any Speed

Marko Mäkelä
Lead Developer InnoDB
MariaDB Corporation

What was the InnoDB Change Buffer good for?

- If a B-tree *leaf page* of a *secondary index* is not available in the buffer pool, write modifications to a separate change buffer B-tree in the system tablespace.
  - `SELECT`, `CHECK TABLE`, or any unbuffered operation will force a merge of changes.
  - Converts some random access to sequential. (Remember HDD seek times?)
  - Initially for `INSERT`, later (5.5) for `DELETE` and purge, but never for `ROLLBACK`.
- Problems: Unpredictably growing system tablespace, hangs, corruption, …
  - Write amplification: not only `INSERT`, but also `DELETE` and purge must duplicate the entire record and some metadata (to update as little as 1 bit in the final page)
  - Unconditional overhead of maintaining change buffer bitmap (how full is a page?) in case someone might enable insert buffering later
  - Mystery corruptions (MDEV-9663) that are extremely hard to reproduce

[Public]

Why is it Hard to Cover the Change Buffer in Tests?

- If a B-tree *leaf page* of a *secondary index* is not available in the buffer pool, write modifications to a separate change buffer B-tree in the system tablespace.
  - `SELECT`, `CHECK TABLE`, or any unbuffered operation will force a merge of changes.
- Lots of “stars” need to be aligned, in several threads:
  - Page writes or eviction may be blocked by page latches held by some threads
  - `innodb_change_buffering_debug=1` (evicting pages to exercise the change buffer) won’t work if the current thread is holding latches on dirty pages.
    - Even more so with MDEV-30400, which fixes some hangs introduced in MySQL 5.7.
  - Purge of committed transaction history may be blocked by active read views.
- Effective tests will require smart “cool down” periods and (un)lucky timing.

[Public]

Magic Bullets: Random Query Generator (RQG) and RR

- Due to the complexity, impossible to guess how some corruption evolved
- Enter rr: lightweight recording & deterministic debugging
  - Saves a *deterministic* execution trace of *randomly interleaved* processes or threads.
  - The exact sequence of events from the start is available in a GDB based interface.
  - Breakpoints, watchpoints, forward and backward execution (`reverse-continue`)
  - Optimized code can be debugged too (at register and instruction level if needed).
- Perfect for “once in a blue moon” cases for which RQG simplifier is impractical

[Public]

The Tale of a Corruption Bug introduced in MySQL 5.7

MariaDB

Corruption after DROP INDEX, ADD INDEX, INSERT (1/3)

mysqld: /data/Server/bb-10.6-MDEV-30009A/storage/innobase/ibuf/ibuf0ibuf.cc:3615: dberr_t ibuf_insert_to_index_page_low(const dtuple_t*, rec_offs**, mem_heap_t*, mtr_t*, page_cur_t*): Assertion `!__builtin_expect(((page_cur->block)->page.zip.data) != 0, 0)' failed.

- This means that a page overflow occurred during a change buffer merge.
  - The assertion is related to a last-resort fixup for `ROW_FORMAT=COMPRESSED`.
- Why? `DROP INDEX` did not discard old entries, and neither did `ADD INDEX`
  - Lazy deletion: Usually `buf_page_create()` collects the garbage.
  - But, the MySQL 5.7 “bulk index creation” failed to pay back this maintenance debt.
  - “Complexity is the friend of security bugs” (source: a mandatory Oracle course)

[Public]

Corruption after DROP INDEX, ADD INDEX, INSERT (2/3)

- Immediate root cause: The “buffered changes exist” bit was **cleared without actually deleting** the change buffer records for the page.
- How to prove this in `rr replay`? Condensed version:
  - `break ibuf_bitmap_page_get_bits_low`
  - `reverse-finish` …, set a write `watch` point on the bitmap byte for this page
    - Thank $DEITY for the 80386 debug registers and their GDB support!
  - Set breakpoints on `ibuf_insert()` and `ibuf_delete_recs()` for this page
  - `reverse-continue`, `backtrace`, `print index.id`, `print index.name`
- We were unable to create a simplified RQG grammar or test case for this.
  - The large RQG grammar made use of `innodb_change_buffering_debug=1`.

[Public]

Corruption after DROP INDEX, ADD INDEX, INSERT (3/3)

Possible consequences of applying bogus changes to index pages:

- Wrong results, broken MVCC or locking in anything that uses the index
- Crash on change buffer merge (as part of any operation, even CHECK TABLE)
  - In our `rr replay` trace: Page overflow on applying an INSERT operation
- January 2023 support case: Running out of space on when splitting a page
  - The index page on a NULLable column contained records for a NOT NULL column, apparently due to merging garbage change buffer records.
  - Length bytes were misinterpreted as “null flags bitmap” and bogus lengths were read
- Various incarnations of the long-time mystery bug MDEV-9663
  - Some causes of “index out of sync with table” involve the change buffer, some don’t.

[Public]

`rr replay` session (1/3): Setting Watchpoint on Bitmap

```
rr replay /data/results/1669137694/TBR-1672/1/rr/latest-trace
continue                        run from the start to SIGABRT
reverse-continue                to un-catch SIGABRT
tbreak ibuf0ibuf.cc:562         inside ibuf_bitmap_page_get_bits_low()
reverse-continue                backtrack to the above breakpoint
display/i $pc                   show the next instruction ($rip for Intel fans)
stepi                           execute the next instruction
watch -l *(char*)$rbx           set a write watchpoint on the bitmap byte
disable display 1
reverse-continue                backtrack to our watchpoint
```

MariaDB

[Public]

rr replay session (2/3): Evaluating the Watchpoint

```
frame 2                          a buffered INSERT had set the “buffered” flag
reverse-continue                 to the ADD INDEX that had cleared the flag
frame 2
set $id=block.page.id_.m_id
frame 3
print m_index.id                 394
print m_index.name               m_name="idx1"
frame 8
print m_user_thd.query_string
disable 2                        we are no longer interested in this watchpoint
```

[Public]

rr replay session (3/3): Conclusive evidence

`break ibuf0ibuf.cc:2287`	inside `ibuf_delete_recs()`
`tbreak btr0cur.cc:1598`	call of `ibuf_insert()`
`cond 4 page_id.m_id==$id`	
`reverse-continue`	hits the call of `ibuf_insert()`
`set $i=cursor.page_cur.index`	
`print $i.name`	`m_name="MarvÃ£o_idx3"` (not `"idx1"`)
`print $i.id`	321 (not 394)
`continue`	to SIGABRT

Because `ibuf_delete_recs()` was never called (for any page), the garbage from before `ALTER TABLE...ADD INDEX` was wrongly applied to the new index.

[Public]

Déjà vu? Bon voyage! (Matti Nykänen)

- The shutdown hang MDEV-30009 is similar to MDEV-20934. What happened?
- MDEV-19514 in MariaDB 10.5 aimed to made crashes more predictable by avoiding “unsolicited” change buffer merges (only do it when absolutely needed)
- We had never reproduced the shutdown hang ourselves; 2 customers did, in production. The older fix was for a MariaDB Server 10.1 hang, but in 10.5, it was (incorrectly) adjusted for MDEV-19514.
- We were finally able to reproduce the hang in MariaDB 10.6, thanks to
  - Simplified buffer pool and locks in 10.5 and 10.6
  - Improved tooling (`rr record` integrated with RQG)
  - Testing at scale (hundreds of concurrent server instances on two huge machines)

[Public]

Other Corruption Caused by the Change Buffer

MariaDB

Crashing on Corrupted Page is Unhelpful (MDEV-13542)

- Even `CHECK TABLE` could trigger a crash within a change buffer merge
  - Until MDEV-13542 (a.k.a. MySQL Bug #10132) was fixed in MariaDB Server 10.6
- MySQL Bug #61104 (2011) remained a mystery for years. Possible causes:
  - MDEV-22497: a false negative answer to “could the page become empty?”
  - MDEV-24709, MDEV-24448/MDEV-24449/MDEV-30422: race conditions while applying log in recovery or backup
  - (MDEV-30009 starting with MySQL 5.7/MariaDB 10.2): applying a stale purge
- It pays off to diagnose `rr replay` or core dumps of obscure assertion failures.
  - Assertions are like lottery tickets: if you do not write them, you cannot win.
  - Recovery improvements help find tricky cases: MDEV-12353, MDEV-12699, MDEV-14425, MDEV-15528, MDEV-24626, MDEV-25506, MDEV-30479.

[Public]

Mitigation and Lessons Learned

MariaDB

Some Corruption Mitigations in MariaDB Server

- MDEV-13542 (MariaDB 10.6) prevents many crashes due to corruption
  - Reports of any remaining crashes on corruption are ver; our fault injection can only cover fairly basic things, such as page checksum failures.
- MDEV-19514 (MariaDB 10.5) avoids “random” change buffer merges
  - `innodb_force_recovery=4` is no longer needed (and cannot corrupt further).
  - This turned out to *improve* performance, contrary to some fears.
- MDEV-20864 (MariaDB 10.2) Introduce debug option `innodb_change_buffer_dump` (diagnostic help)
- MDEV-21069 Crash on `DROP TABLE` if the data file is corrupted
- MDEV-29905 Change buffer operations fail to check for log file overflow

[Public]

Some More Corruption Mitigations in MariaDB Server

- MDEV-27734 (10.5): Set `innodb_change_buffering=none` by default
  - No significant performance regression was observed
- MDEV-27735 (10.9): Deprecate the parameter `innodb_change_buffering`
- MDEV-29694 (11.0): Remove the InnoDB change buffer
  - On upgrade from earlier versions, change buffer merge will be completed and the change buffer removed to prevent downgrade.
  - The change buffer bitmaps will be ignored and reset during upgrade.
  - Change buffer bitmaps need not be maintained (they were initialized to safe values).

[Public]

What can we Learn from This?

- InnoDB until MySQL 5.1 was based on *Transaction processing* by Gray&Reuter.
  - Except some InnoDB “innovations”: insert buffer, adaptive hash index.
- Layer boundaries are a powerful abstraction that should not be violated lightly.
  - Extensive tricks are needed to avoid deadlocks or inconsistency.
  - Those tricks and rules can be forgotten or overlooked too easily by future developers.
- Strictly following layers and simple design rules makes life easier.
  - Easier to write unit tests and reach full code coverage in integration testing.
  - Easier to determine what is right and wrong when debugging or reviewing code.
  - If you can’t explain something in simple terms, maybe something is wrong.
- Redundant or partly duplicated data structures are prone to cause inconsistency.

[Public]


MariaDB
THANK YOU