Group commit issues for the storage engine API

Kristian Nielsen
knielsen@askmonty.org

MariaDB Developer
Monty Program AB

O'Reilly MySQL Conference & Expo 2010

Outline

Commit and durability

The “D” in ACID means durability

- When commit returns successfully, changes are guaranteed to persist even in case of crash
- Typically requires an expensive `fsync()` or similar.

Thread1
Update in-memory buffers
`write()` to transaction log
fsync() transaction log

Commit in multiple threads

Thread1	Thread2	Thread3
Update buffers		
		Update buffers
	Update buffers	
`write()` log		
fsync() log . . .		
		`write()` log
	`write()` log	wait . . .
	wait . . .	
fsync() done		(wake up)
		fsync() log
	(wake up)	fsync() done
	fsync() log	
	fsync() done	

Optimisation: Group commit

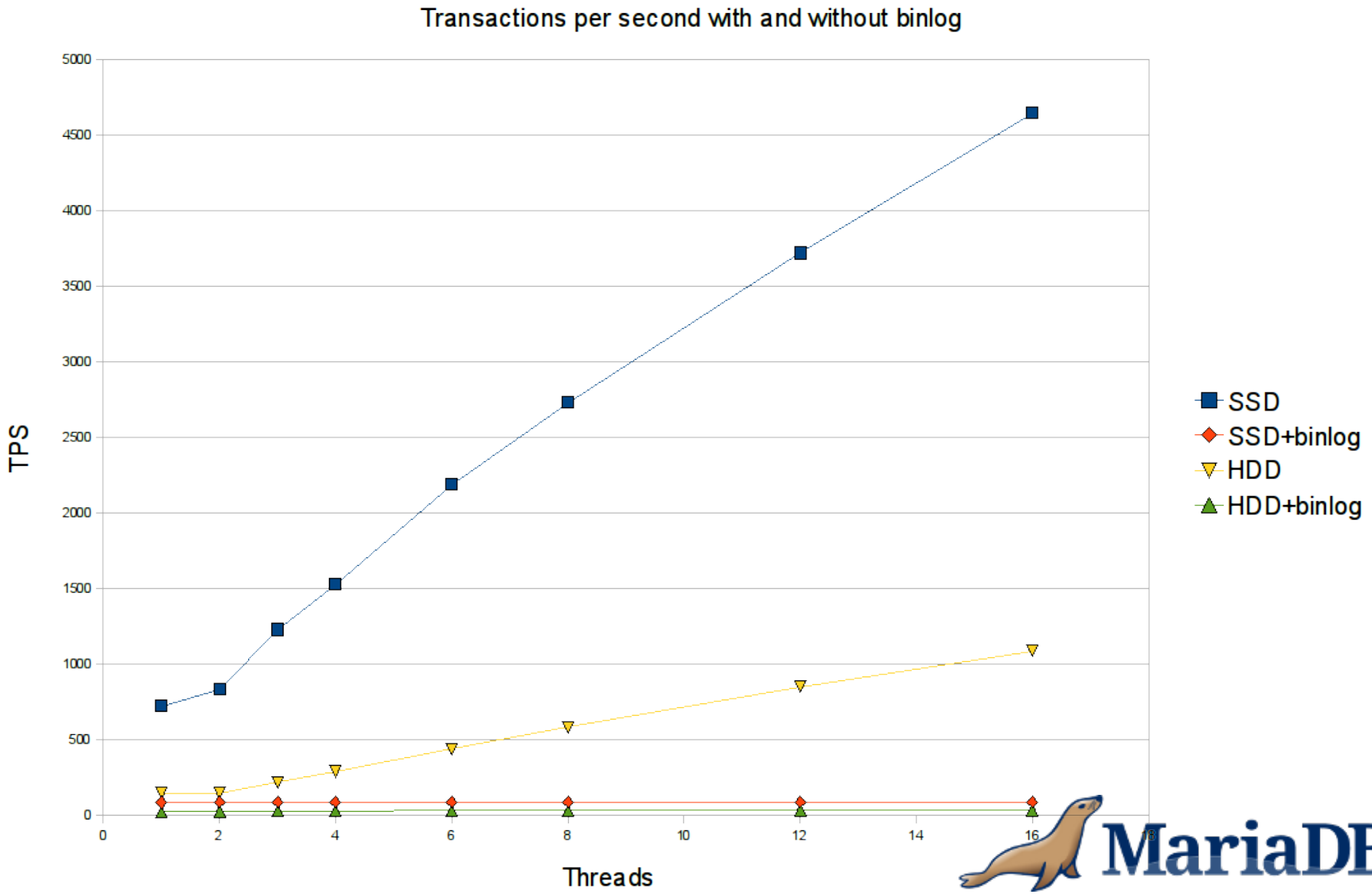
Thread1	Thread2	Thread3
Update buffers		
		Update buffers
	Update buffers	
`write()` log		
fsync() log . . .		
		`write()` log
	`write()` log	wait . . .
	wait . . .	
fsync() done	(wake up)	
	fsync() log for both threads . . .	
	fsync() done	
		(wake up)

Group commit: broken since 2005

. . . but it does not work!

- Bug#13669
- Peter Zaitsev, September 30, 2005
- "Group commit is broken in 5.0"
- `http://bugs.mysql.com/bug.php?id=13669`

Group commit benchmark



MariaDB

Group commit benchmark

Details:

- Simple `REPLACE` query
- `innodb_flush_log_at_trx_commit=1`
- `sync_binlog=1`
- Binlog enabled and disabled
- Digital Western 10k HDD and Intel X25-M SSD

Observation:

- With binlog disabled, scales well with more threads due to group commit
- With binlog enabled, no scaling due to broken group commit

The problem

- Binlog and engine commit using 2-phase commit / XA
- XA uses 2-phase `prepare()` and `commit()`
- InnoDB holds a global mutex across `prepare()` and `commit()`
- Result is complete serialisation, with no opportunity for transactions to queue up for group commit

The problem: serialised commit

Thread1

InnoDB `prepare()`
- Update buffers
- `write()` log
- `fsync()`
- `lock(prepare_commit_mutex)`

Binlog
- `lock(LOCK_log)`
- Binlog `write()`
- Binlog `fsync()`
- `unlock(LOCK_log)`

InnoDB `commit()`
- Update buffers
- `unlock(prepare_commit_mutex)`
- `write()` log
- `fsync()`

The problem: serialised commit

Thread1	Thread2	Thread3
`prepare()`		
`lock()`		
	`prepare()`	
Binlog	Wait . . .	`prepare()`
`commit()`		Wait . . .
`unlock()`	. . . wake up	
	`lock()`	
	Binlog	
	`commit()`	
	`unlock()`	. . . wake up
		`lock()`
		Binlog
		`commit()`
		`unlock()`

Why all the locking and `fsync()`?

- XA needed to keep engines and binlog in sync after crash
  - Otherwise could get difference between engine and binlog (and hence slaves).
- Same commit order needed in InnoDB and binlog
  - Otherwise InnoDB hot backup / XtraBackup may create state that does not exist in binlog, causing inconsistency on slaves.
  - Could also use for `START TRANSACTION WITH CONSISTENT SNAPSHOT`
  - Could also use for global transaction ID.

Idea for a solution

- InnoDB `commit()` part has a fast part and a slow part
  - The fast part updates in-memory buffers and fixes the commit order
  - The slow part does `write()` and `fsync()` of the transaction log
- Only the fast part needs to be synchronised with binlog commit order
- Only the slow part needs to participate in group commit
- So split the fast part out into an (optional) separate handlerton call

`start_commit()`

- Optional
- Called before `commit()`
- Guaranteed to be called in same order as binlog commit
- Idea is to do as little as needed to ensure commit order

`commit()`

- Same as existing handlerton commit call
- Do “the rest” of commit (log write, `fsync()`, etc.)
- Not guaranteed in same order as binlog commit
- Backwards compatible

Transaction commit pseudocode

`prepare(this)`

Queue up for group commit
If (binlog fsync is running)
    Wait until signalled
Else (binlog fsync is not running)
    For (all queued transactions T)
        Write T to binlog
    `fsync()` binlog
    For (all queued transactions T)
        `start_commit(T)`
        Signal T to wakeup

`commit(this)`

(Also wake up transactions queued during fsync)

InnoDB `start_commit()` and `commit()`

InnoDB commit code is already structured in this way

```
start_commit()
   Read binlog position
   trx->flush_log_later = TRUE;
   innobase_commit_low(trx);
   trx->flush_log_later = FALSE;

commit()
   trx_commit_complete_for_mysql(trx)
```

Can also handle group commit for multi-engine transactions

Other ideas?

Other possibilities for fixing group commit

- Do not require same commit order
  - Find some other way to make Innodb hot backup work correctly
- Introduce a global transaction ID
  - When writing to binlog, assign consecutive number (ID) to transactions
  - Pass transaction ID to commit()
  - Engine can order concurrent commits according to ID

Other ideas (continued)?

Other possibilities for fixing group commit (cont)

- Ensure binlog/engine consistency without XA
  - Durability only for binlog (`innodb_flush_log_at_trx_commit=0`)
  - Do `fsync()` for binlog before engine `commit()`
  - In crash recovery, playback missing transactions from binlog
  - Requires storing global transaction ID (or similar) in binlog and engines.
- Other suggestions?

Conclusion

- Group commit has been broken for **5 years**, an embarrassment!
- I want to fix it!

Slides:

`http://knielsen-hq.org/maria/uc2010.pdf`

Contact:

`knielsen@askmonty.org`