The Dangers of Logical Replication and a Practical Solution

Jose M. Faleiro
Microsoft Research

with
Daniel Abadi, Jeff Helt, Wyatt Lloyd, Abhinav Sharma
Log-shipping replication
Replication lag
Primary-backup parallelism gap

Cannot reuse primary’s concurrent execution mechanisms
Multi-threading, synchronization

Weak isolation
Backup must be serial-equivalent

Transaction scheduling
Why now?

**Before: Parallelism gap hidden behind I/O bottleneck**
Multi-threading used to mask over I/O latency on primary

**Backups: Read-ahead into the log to prefetch pages**
I/O bottleneck is less relevant today

Servers equipped with increasingly large DRAM
Larger portion of working set can fit in memory

Lower bufferpool miss penalty
Servers equipped with SSDs
100x lower latency than HDDs

Increasing hardware parallelism with multi-cores
Replication lag in the wild

Github
“Maintaining low replication lag is challenging”
Investment in monitoring and throttling (freno)

Booking.com
Reported cases where backups lagged by 90 minutes

VividCortex
“… suffering some serious delays. The worst was behind by at least 16 hours”
GitLab incident

19:00 UTC – GitLab experiences increase in database load

23:00 UTC – Replication process experiences a serious failure
Primary drops log records needed for replication

23:30 UTC – Primary data is partially wiped out
Attempt to restore backup from checkpoint
Admins try wipe out backup, but wipe out primary instead

GitLab.com melts down after wrong directory deleted, backups fail
Upstart said it had outgrown the cloud – now five out of five restore tools have failed
By Simon Sharwood 1 Feb 2017 at 02:02

GitLab offline after catastrophic database error loses mountains of data

Lots of Ops "holier than tho" over the GitLab outage. Sure, they made some mistakes, but let they without Ops sin cast the 1st stone.

5:39 AM - 1 Feb 2017
Addressing replication lag in MySQL at Facebook

Published work on deterministic execution for multi-cores
… but I wanted a compelling real-world use case

Log shipping-based replication seemed like a perfect fit
Parallelize the execution of totally ordered log records

Talked to lots of real-world practitioners
Mark Callaghan pointed me to relevant folks at Facebook
Replication mechanism used fairly widely in production
Requirements

**Backups must use at least as much parallelism as primaries**
Robust to innovations in concurrency control
Robust to weak isolation

**Backups must provide snapshot isolation for read requests**
Reads must observe complete log prefix
Primer on MySQL replication

Single threaded log application

Log records are logical
Transaction statements
Insert, update, delete
Primary keys always specified

Transaction management
MyRocks
RocksDB
Storage engine
Parallelization strategy: Single threaded shards

**Soft partition DB**
Route log records to appropriate partitions

Log record could span many shards
Issues with single threaded shards

Each write must be wrapped in its own transaction
Extra overhead to begin and commit transactions (compared to primary)

Restrict each log record to a single write on primary
Adds overhead on primary
Not general enough, breaks compatibility

Unpack log record on dispatcher thread
Additional overhead on a centralized component
Memory management overhead
Dispatcher thread performance

![Graph showing throughput (inserts/sec) vs. number of threads]

Throughput (inserts/sec) vs. Number of threads

- Dispatcher thread
- Backup offline
- Primary
Parallelization strategy: Short-duration locks

**Assign each transaction to worker threads**
1:1 correspondence between primary and backup transactions
But avoid long-duration locks

**Dispatcher determines schedules based on conflicts**
Locks released after statement finishes executing
Short-duration locks example

Dispatcher correctly determines dependencies prior to execution

No deadlocks
No logical aborts
Issues with short-duration locks

**Chain of uncommitted writes on each record**

In addition to being uncommitted, each needs to be visible

Need help from storage engine

Compromise: Short-duration at replication subsystem, but long-duration within MyRocks
Requirements

Backups must use at least as much parallelism as primaries
Robust to innovations in concurrency control
Robust to weak isolation

Backups must provide snapshot isolation for read requests
Reads must observe complete log prefix
Snapshot isolation for reads

Snapshot is a complete prefix of transaction log

No guarantee that committed transactions are complete prefix

Need a mechanism for correctly capturing snapshots
Snapshot strategy: Low-watermarks

No guarantee that committed transactions are complete prefix
But a subset of committed transactions forms a complete prefix

Maintain a low-watermark corresponding to this subset
Serve reads off this low-watermark

Low-watermark
Issues with low-watermarks

**Replication subsystem can’t pick timestamps**
RocksDB does, and they are assigned when transactions commit

**Backup timestamps don’t match those on the primary**
Transactions might commit in different order
Snapshot strategy: Asynchronous checkpointing

Pick a future point in the log at which to checkpoint
Allow transactions before that point to commit
Allow transactions after that point to execute, but not commit
We implemented asynchronous checkpointing (with some difficulty)

Transactions not in the checkpoint execute but don’t commit

Implicitly assumes that cost of execution $>>$ cost of commit

This was not always true of RocksDB… but thankfully true when we got to work

Issue with uncommitted chains of writes
Performance
Point inserts

![Graph showing throughput (inserts/sec) over time elapsed (seconds) for Slave, Master, Backup, and Primary.]
TPC-C NewOrder

Commits/sec vs Time elapsed (seconds)

- Backup
- Primary
Lessons / Random stuff

Replication must be a first-class citizen of any design
… if not, you’re entering a world of pain

Building modular high-performance systems is very tricky
But worth it, code is easier to work with and evolve
Doesn’t work with InnoDB due to implicit gap locking, which can deadlock

Higher levels of stack now complain that replication is too fast
Conclusions

Replication lag can be a serious emergent issue
Impact on operations, availability, user experience

Parallel replication subsystem for MySQL
Parallel log playback + snapshot isolation reads
Deployed at Facebook

Be prepared for lag despite best efforts
Lag detection mechanisms
Log archiving
Q&A

jmf@microsoft.com
Github: facebook/mysql-5.6