facebook

facebook

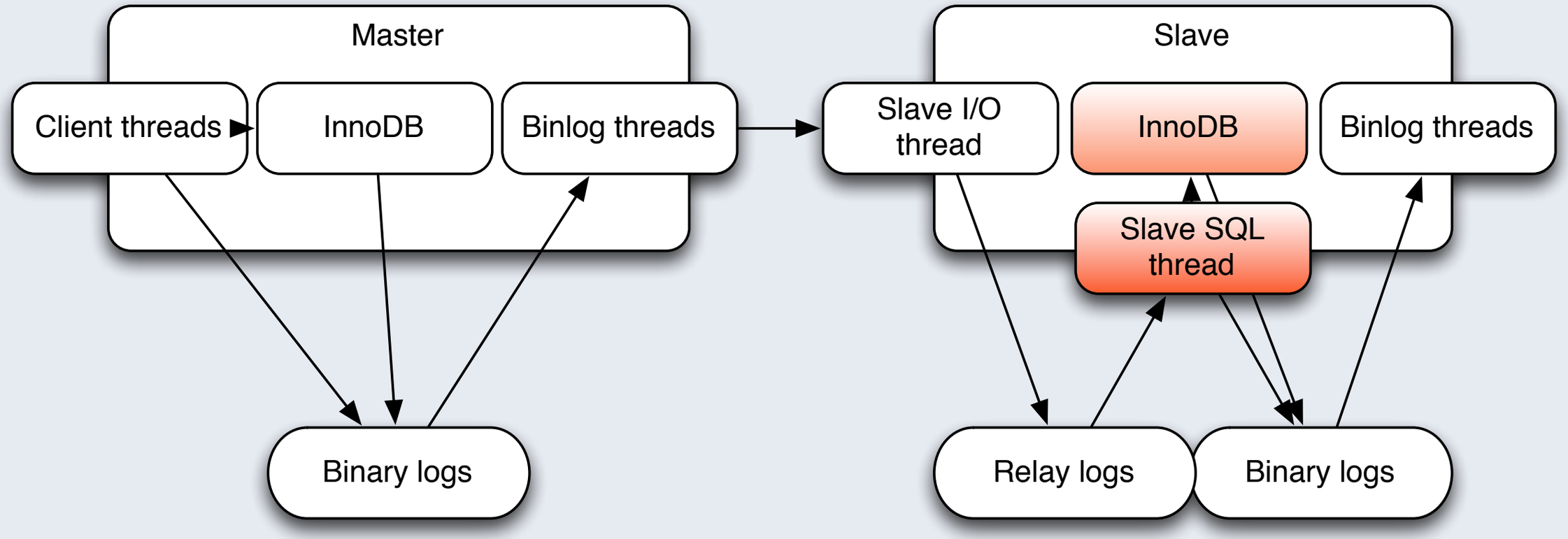
High performance replication MySQL @ Facebook experiences

Domas Mituzas
Small data engineer
April 23, 2013

Replication performance problem

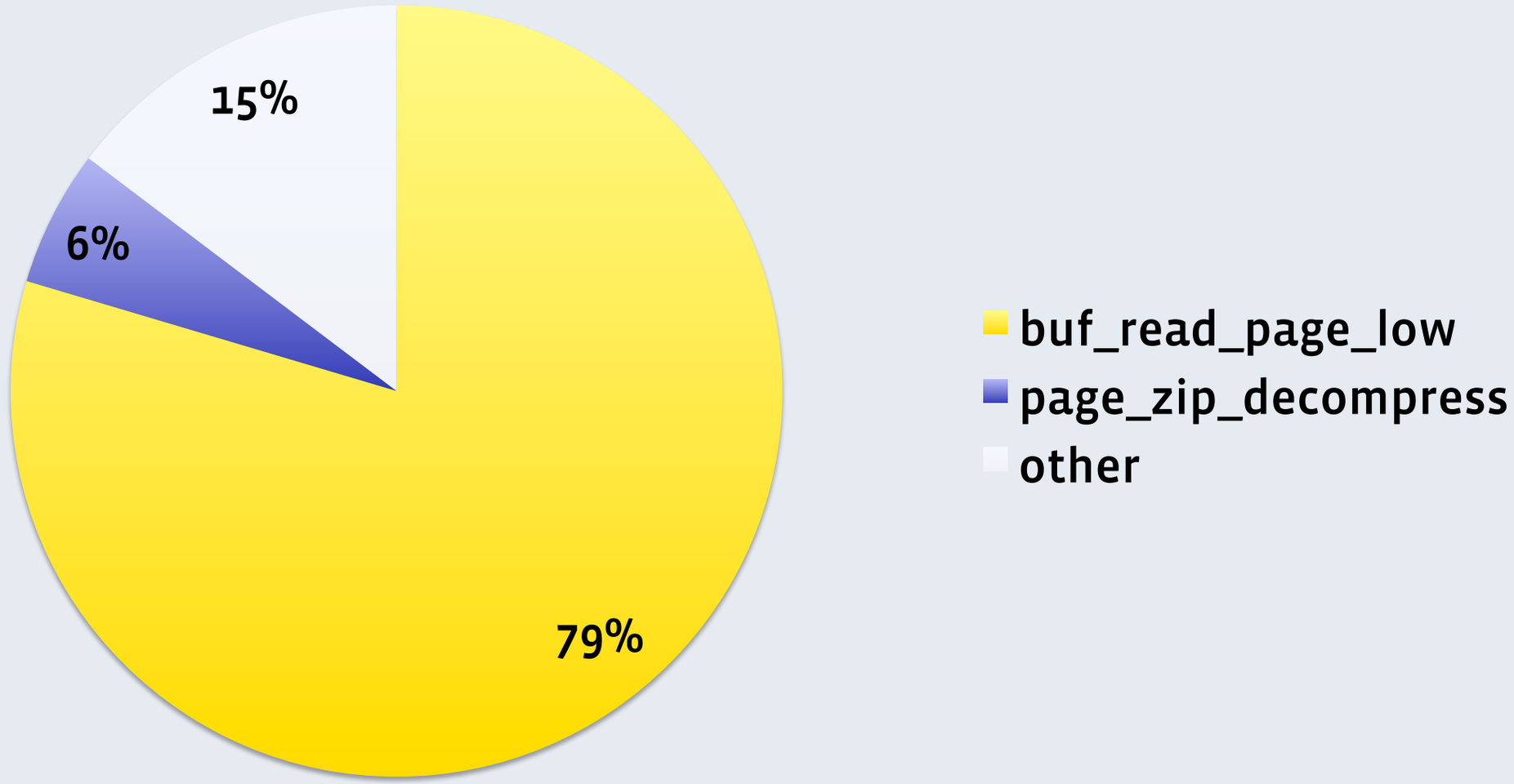
Replication has many components

All are problematic, but one is the center of pain



What does SQL thread do?

Read lots, then maybe a bit of something else



Why read so much time?

- Too high slave read workload
- Too expensive working set for slave reads

- Too much cold data accessed by replication
- Too many writes done
- Too much data written/modified

Solutions?

- More hardware:
  - Sharding to reduce datasets
  - Flash storage to make reads cheap
- Better software
  - Avoid reads in SQL thread
  - Smarter queries, tables, indexing, etc
  - Have more SQL threads!

Efficient replication

Replication data prefetching

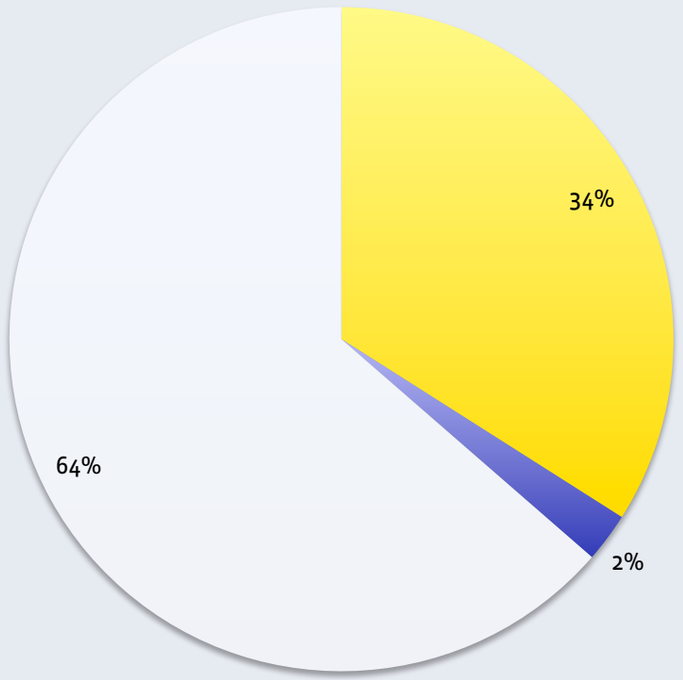
- Rewrite INSERT/UPDATE to SELECT?
  - Mk-slave-prefetch
  - Readahead.py with custom rules

- DML works with multiple indexes – complication!
- DML touches neighboring pages – showstopper!

- Does not provide more than 50% reads coverage – limited to up to double replication performance

Fake changes & faker

- Don’t rewrite INSERT/UPDATE to SELECT!
  - innodb_fake_changes=1
  - Supports InnoDB and Blackhole only
- High coverage, but not 100%
- In Percona Server



- 4x faster replication with our workload
- Lp:mysqlatfacebook/tools/faker

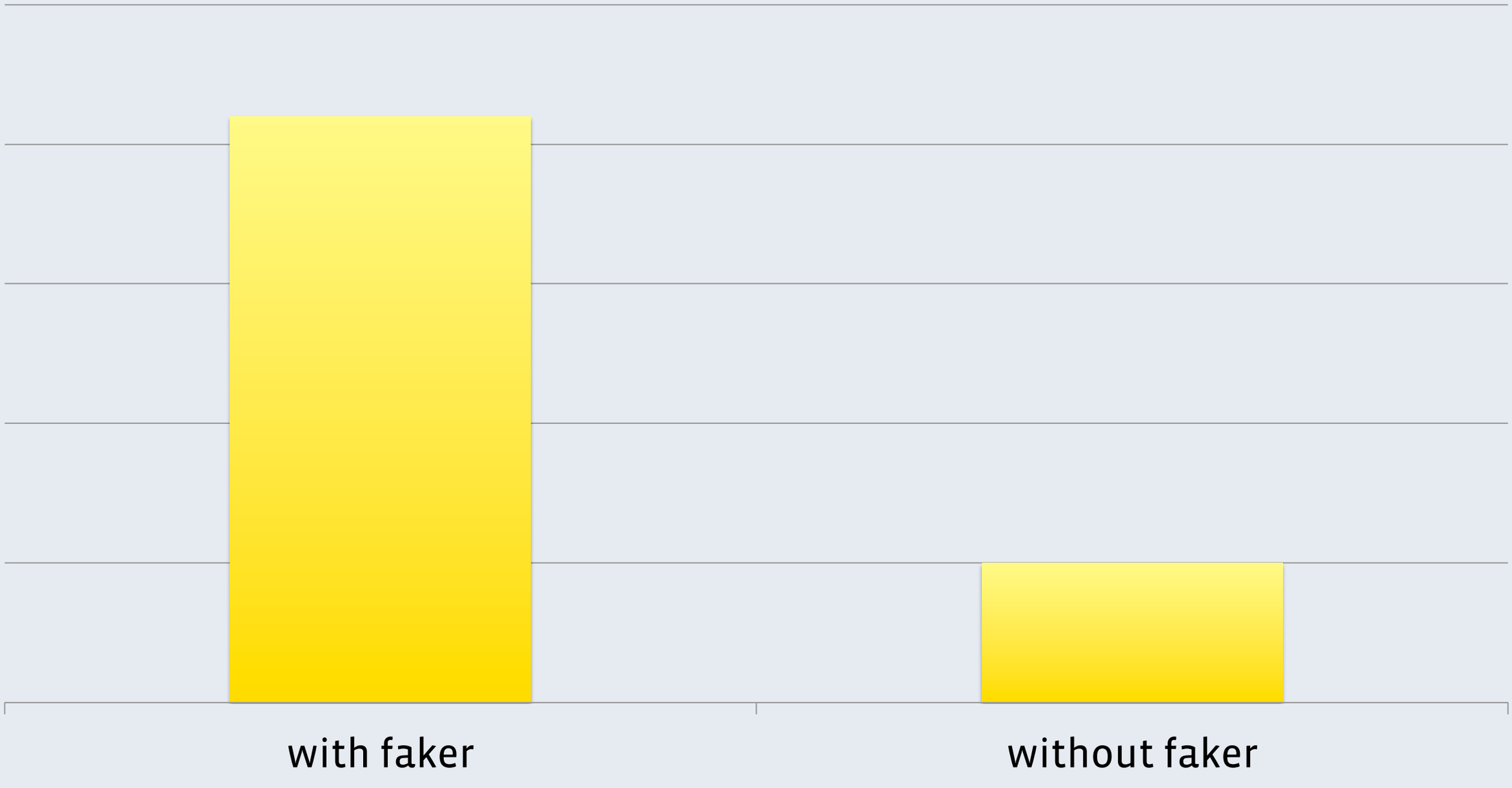
With fake changes on...

- Same SQL gets executed
- Same optimization paths are hit
- Same index dives are done, data pulled
- Page modification itself is skipped
- No assertions violated!

Prefetching issues

- Additional benefits
  - Decompress needed pages
  - Precache relay logs too (for lagged replicas)
- It has problems
  - Additional CPU overhead/cost
  - Incomplete coverage (optimistic paths become pessimistic in real execution)
  - Does not help with other replication bottlenecks

Faster replication with faker



Multi-Threaded-Slave!!!

- Still learning, still problematic
  - No 5.1->5.6 path, faker mandatory for migration
  - Does what is expected, heavy sharding necessary
  - Does not help with single-shard avalanches
    - Assertion factory (in debug builds)
- Shares lots of same bottlenecks as faker
- Faster than faker for many workloads

Other bottlenecks

Query execution performance

Query execution on CPU

- Multiple instances or multi-threaded slave helps

Global internals

- MTS does not increase logging performance
- Checkpointing causes global stalls
  - Multiple instances › MTS
  - Nearly linear scaling with multiple instances in some cases

Binary log transfer

Reading files

- Locks binlog mutex [unless patched]
- Does binlog/relaylog reads in 8k blocks [unless patched]

Network transfer

- Large window sizes needed [sysctl: net.*.*mem*]
- Compression greatly improves log transfer QoS

Write activity

InnoDB checkpointing

- Background checkpointing is a must (just “adaptive” doesn’t help)
  - See Bug#55004, probably fixed in 5.6

Logging is expensive

- InnoDB log flushing should not be 1 (use crash-safe slave instead)
- Ditto for sync_binlog
- Transaction logs need to stay in memory (Bug#69002)

Live analysis

Logical workload

Replication profiling

- Sample ‘system user’ thread in processlist, breakdown queries by various features
- Slow query log analysis

Server statistics

- User stats for system user
- Write metrics in table statistics
- Buffer pool contents statistics

Server operation

PMP / Quickstack

- Sampling just replication LWP is cheap enough
- Gives much deeper picture of what is going on
- Identifies main replication cost better than anything else

Perf

- CPU cost breakdown on replication thread
- Can identify expensive query/workload features

facebook