

high concurrency mysql

Domas Mituzas

Facebook, Wikipedia, ex-MySQL

Components

- Hardware
- Operating (file) system
- Database engine
- Workload
- Monitoring
- Acrobatics

Opposition from within

```
/* If there were less than 200 i/os during the 10 second period, we  
assume that there is free disk i/o capacity available, and it makes  
sense to flush 100 pages. */
```

-- InnoDB I/O policy in MySQL 3.23 - 5.1

An application should try to keep the number of concurrently open transactions less than 1000. Having lots of open transactions can cause bottlenecks in locking, for example.

-- Heikki Tuuri, Bug#26950

An application that has this kind of hot rows isn't going to get very far anyway

-- Michael Izioumtchenko, Bug#49047

I/O Hardware

- Flash doesn't care much about concurrency - linear execution of requests
 - Lots of concurrency problems disappear
- Single NCQ-capable disk performance may increase with multiple outstanding parallel requests
- Multiple disk RAID setups love concurrent workloads
- +175% performance improvement for concurrent accesses
- Example - X4240 - 16 disk box
 - TPS of random 16k reads:
 - 25ms await at 128 threads
 - 7ms await at 16 threads
 - writes were always at ~2000 tps
- Write behind caching can be odd
- Always upgrade firmware.

# threads	20G	200G
1	270	215
16	3000	2250
32	4500	3300
64	6000	4250
128	7500	5200
256	8700	6200

File system requirements

- XFS:
 - Data alignment on stripe boundaries
 - Usually the reason behind strip-size mis-conception
 - Avoids extra +5-20% reads/writes on spindles
 - Parallel Direct-IO on single inode (XFS-only)
 - Fast sync file appends (XFS twice faster than ext3)
- Virtual Memory, buffers, swap:
 - `vm.swappiness=0`
 - `vm.dirty*`
- `posix_fadvise()` for all large non-DIO reads
- `sync()` or `sync_file_range()` for most of large non-DIO writes
- Kill CFQ - deadline or noop I/O scheduling are a must

MySQL basics

- Thread cache
 - can be equal to max connections (Threads_created=0)
 - doesn't hold buffers, except thread_stack
 - Still too slow for high connect counts
- Table cache & LOCK_open
 - Currently on top of broken functionality - serialized and slow table opening, 10x faster with a patch
 - Must set between # of tables and #tables*connections
 - Doesn't scale well for some engines (e.g. MyISAM)
 - Fixes are all quite low hanging
- MyISAM (fast scans, not reads) - use multiple key caches
 - key_buffer - major source of contention
- 5.1 with InnoDB Plugin - earliest good enough version
 - Or 5.0 with Facebook Patch

InnoDB scheduling

- `innodb_thread_concurrency` is a mystery
 - mostly, because it is flawed concept
 - 'unlimited' works, until it does not. then it falls down.
- General idea: " $\# \text{ CPUs} * 2 + \# \text{ disks}$ ", manual says:
 - "The correct value for this variable is dependent on environment and workload. You will need to try a range of different values to determine what value works for your applications. A recommended value is 2 times the number of CPUs plus the number of disks."
- Should be two separate settings:
 - In-core concurrency (~4-8)
 - I/O concurrency ($\# \text{ disks} * 2-8$)
- Gotta balance with what we have :(
- Settings accommodating I/O capacity may hit mutex contention, so that's the major problem to watch for
- `innodb_concurrency_tickets` should be at 99th percentile of rows accessed per query
- Slots held on waits for I/O (though not lock waits)
- FLIFO (fb) for people with >1000 threads running

InnoDB I/O

- 5.1 with InnoDB plugin is much much better
- async is just serially 'in background' executed queue
- Reads ahead by default, in single thread
 - Major contention spot on range-read workloads
 - Especially problematic with batch tasks running
 - Unnecessary I/Os in many cases
 - No benefit in high concurrency
- Single page flush thread
 - Major problem on NFS and other high latency systems
 - Does include checksumming
 - Write-behind caching mandatory
- Flushing policies back from 1995
 - Tune I/O capacity
 - Adaptive flushing
- Concurrency slots held during I/O
 - Single block read can block multiple threads / slots

InnoDB kernel

- Becomes slower with large transaction counts
 - Establishing read-view needs to traverse full transaction list ('read committed' slows down most)
- Becomes slower with large lock counts
 - Bug#49047 - deadlock detection has exponential costs
 - Disabling deadlock detection entirely helps
 - 1.0.7 Plugin has a patch
 - Releasing lots of locks can be expensive
- Blocks LOCK_open on tables open while:
 - Does random reads for statistics from table
 - Reads auto-increment information
 - Waits for kernel_mutex - escalates into locking disaster
- Kernel mutex needed for LOCK_thread_count (connect/disconnect) operations

Understanding the impact

- Top-N monitoring - finding servers with worst case behavior
- 'dogpiled' - overload watchdog, with extensive data collection
 - fires on high 'threads_running'
 - processlists, innodb status
 - iostat, system information
 - PMP!
- Poor Man's Profiler (.org)
 - Allows seeing bottlenecks
 - `grep -v pthread`
 - Allows seeing threads locked up by bottlenecked threads
 - `grep pthread`
 - Perfect low hanging fruit detection

Pile analysis

- Worst cases immediately reveal lowest hanging fruits
 - Lots of rows scanned
 - Bad selectivity
 - Thundering herds
 - Abused aggregations
 - Few rows scanned
 - I/O-heavy workloads (bad locality)
 - OS / FS / HW problems
 - MySQL-internal lockups
 - Commit contention
- "The worst of the worst" targeting based optimizations still directly affect whole cluster performance
- Workload interaction is as important as individual stats



Best performance debugger!

```
gdb -ex "set pagination 0" -ex "thread apply all bt" --batch -p $(pidof mysqld) |  
awk 'BEGIN { s = ""; }  
    /Thread/ { print s; s = ""; }  
    /^#\#/ { if (s != "" ) { s = s ", " $4 } else { s = $4 } }  
    END { print s }' - |  
sort |  
uniq -c |  
sort -r -n -k 1,1
```

Understanding the workload

- **/* Query Comments Are Important */**
- tcpdump / mysqlsniffer
 - high-resolution time in transaction composition - sometimes gaps between statements as important
- Slow log aggregation
 - Much more metrics in slow log are mandatory
- Slocket (log_datagram) - datagram-based query log
 - Custom real-time monitoring & aggregation
 - Mapping of user/application activity to low level metrics
 - Evaluating upper layer logical component costs

Managing the workload

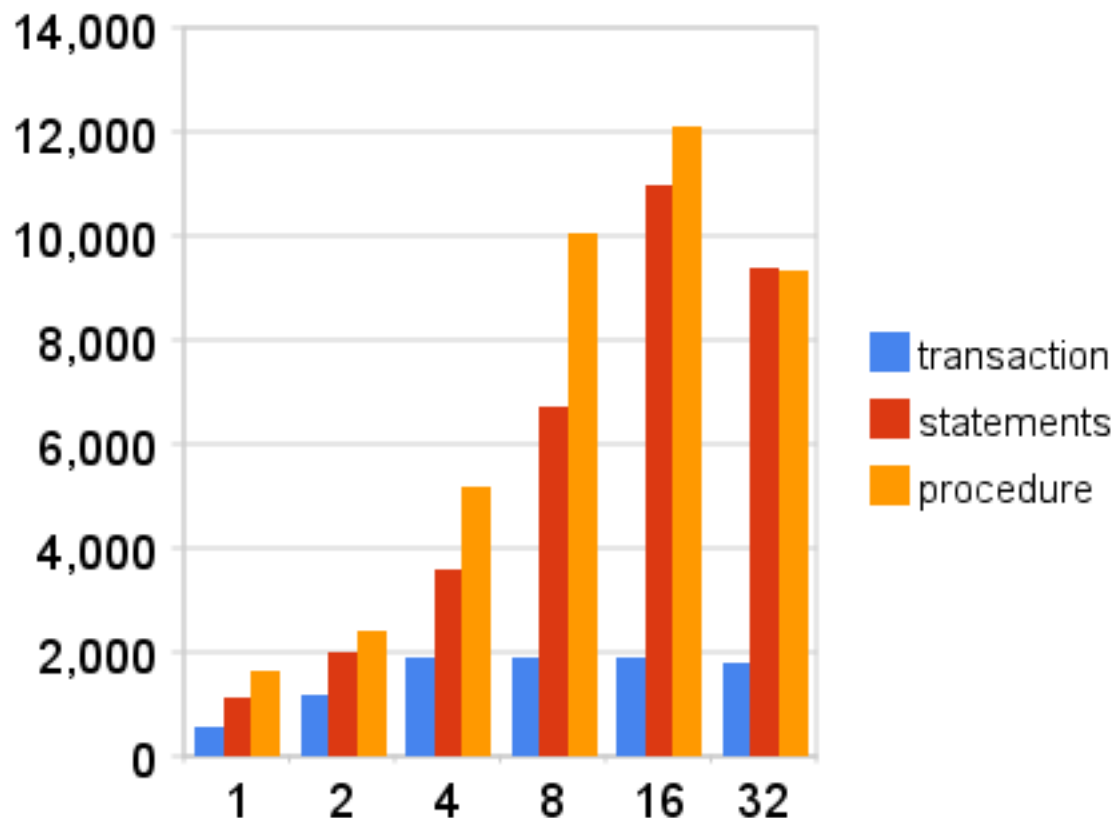
- Overload agent (pylander), when `threads_running > 1000`
 - Duplicate elimination - oldest stays, for cache priming
 - Desperate measures - kill long running queries
 - Durability management - reduce durability settings for short bursts to overcome commit contention
 - Threads management - allow more connected threads, if not all of them are running
- Extreme database measures (fail early)
 - `wait_timeout=3`
 - `innodb_lock_wait_timeout=1`
- Provide separate accounts for different load sources, set per-account `max_connections` (need `max_running`)

Use caches

- When and how?
 - Cache everything what can be cached in memcached
 - Cache empty datasets too
 - Use application cluster-wide cache mutexes for database access - thundering herds are deadly
 - Pre-cache data, instead of invalidation, if possible
- Don't fully rely on cache, neither internal nor external
 - Backups, restarts, time-dependent loads will wipe it
 - Existing cache shouldn't be an excuse for sloppy SQL

Use stored procedures

- We don't, yet. But we should, in one form or another.
- Avoids application locking gaps
- Example: aggregation table maintenance performance



Use intelligent proxies

- Application aware proxying:
 - Keeping transaction logic closer to MySQL
 - Write-behind caching and merging of requests
 - Less connections/transactions open on DBMS
 - Especially important for multiple datacenter setups

Questions?

<http://fb.me/mysqlatfacebook>

<http://dom.as>