



PERCONA
Performance Consulting Experts

Outrun The Lions

Baron Schwartz

Percona Inc (via OfferPal Media)

FaceApps Developer Conference

Las Vegas, NV

July 12, 2008

Who Am I?

- Lead author of High Performance MySQL 2nd edition
- Author of Maatkit and innotop
- Consulting team lead at Percona
 - We are full-stack performance experts
- Background in e-commerce, retail, search engine advertising and marketing
- Degree in Computer Science from University of Virginia



Somewhere in the Serengeti...



I can see the future

I see your load increasing.



Most of you will follow similar patterns.

A sample of common problems

- Badly optimized queries
- Poor performance: locking, swapping
- Inability to predict load or plan for capacity
- Lack of usable backups, inability to take backups
- Uncertainty about data integrity
- Inability to handle failures
- Inability to recover from failures (long downtimes)
- Database server is needlessly critical to app function
- Crises of all kinds

Disclaimer

- There are reasons to break every rule
- This presentation is generic advice
 - Based on the Facebook apps I see often
- I am saying what I think is best, without saying why
- I am not presenting alternatives
- I am not explaining the nuances
- You should research the nuances yourself
 - Or get advice from someone knowledgeable
 - Or just read the “why” in our book

Some good strategies

- Automate everything
- Use the best technology and tools you can
- Measure everything
- Plan for success, so it doesn't kill you
- Optimize out every cost you can
 - Performance matters because cost and agility matter
- Balance your efforts
 - Be prudent, unless you have a lot of funding
 - If you have a lot of funding, talk to me afterwards...

Use InnoDB

- It's fast, it's stable, it's recoverable
- It makes hot backups with LVM possible
 - You can also use InnoDB Hot Backup
- It has good concurrency, MVCC, row-level locking
- Don't use MyISAM for general-purpose tables
 - Big MyISAM tables are hard to recover after a crash
 - MyISAM doesn't even try to get your data to disk
 - And it has no write concurrency
 - For large volumes of read-only data it can be OK

Some basic InnoDB configuration

- The following advice depends on your hardware!
- Read in our book about how to set your InnoDB buffer pool size and log file size
- Use `innodb_file_per_table`
- Use `innodb_flush_method=O_DIRECT`
- Always watch the error log
 - Never ignore anything InnoDB tells you there!

Set MySQL to “sane” mode

- Make MySQL more strict
 - So you don't get silent errors
 - Take a look at the `sql_mode` setting
 - Especially the `ONLY_FULL_GROUP_BY` setting
- Use `no_engine_substitution`
- Explicitly name your error logs, slow logs, etc etc

Prepare for replication

- Set up binary logging
- Explicitly specify the path to your binary logs
- Unlike the InnoDB logs, it's actually a good idea to put the binary logs on a different volume
 - If you lose your disks you still have point-in-time recovery capability from last backup
- On slaves, set `read_only`, `skip_slave_start`, and explicitly specify the path to the relay log files
 - Also set binary logging so you can catch unwanted modifications on slaves, and don't grant SUPER
- Specify `expire_logs_days`, and don't delete binlogs manually

Be safe with replication

- Don't use non-replication-safe things
 - Look in the manual for the list
- Don't use temporary tables
 - Unless you don't mind re-cloning your slaves after errors
 - Use ordinary tables with unique names instead
- Don't mix transactional and non-transactional storage engines (set default storage engine)
- Test replication integrity with Maatkit's `mk-table-checksum`
 - You may be using unsafe query constructs without knowing it

Know how replication works

- Know the characteristics of replication
- If you use replication filters, learn them well
 - Generally, keep masters and slaves identical
 - There are uses for filters, but most of you don't need them
- Writes are repeated on all slaves
 - You cannot scale writes with replication
- Slave lag is a fact of life
 - Slaves are at a disadvantage compared to masters
 - Your app must be able to handle slave lag
- Smart load balancing can avoid cache duplication
 - get more out of your hardware

Hardware

- Generic advice – take with a grain of salt
 - Balance performance optimization with hardware \$\$\$
- 4 CPU cores, faster is better
 - 8 cores can be worse
- Up to 32GB RAM is common, try to hold working set
- Fast I/O subsystem, if you can afford it
 - Best of breed right now: 15k 2.5” SAS drives
 - 2 drives (RAID 1) for OS install, RAID 10 array for MySQL
 - Use LVM, leave space for snapshots
 - Don't separate InnoDB logs and data
 - RAID controller, battery-backed write cache

Operating System

- OS: GNU/Linux
 - Make sure everything is 64-bit! 32-bit is common mistake
- I like CentOS, Debian, others are fine too
- Do not let your DB server swap
 - Monitor si and so columns in “vmstat 5”
 - Tune /proc/sys/vm/swappiness
 - Set innodb_flush_method=O_DIRECT
- What about EC2?
 - It depends: do you have a low write workload and does the working set fit in memory?

Test your hardware

- Use sysbench to measure your I/O performance
- Yank the power plug and make sure the BBU works
- Don't let the hosting company talk you into low-grade machines. Many of them don't really understand what a DB server requires.
 - One hosting provider's quote: “7200 RPM SATA drives are as fast as anything you can buy.”
- Calculate the concurrency from iostat
 - $\text{concurrency} = (\text{r/s} + \text{w/s}) * (\text{svctm} / 1000)$
 - make sure you're getting the concurrency you paid for

Schema/Query Optimization

- These two query patterns are the single biggest performance killers I commonly see – MySQL does not optimize them well.
 - Don't use IN() subqueries
 - Use joins, with DISTINCT if needed
 - Don't use NOT IN() subqueries
 - Use LEFT OUTER JOIN
 - Search Google for “exclusion join”
- Ranking and pagination are typical problems
- Learn EXPLAIN and use it
 - Thorough appendix in our book
 - Slides online at percona.com

Schema/Query Optimization

- Problem: long-running queries or too much load
- Think small.
 - Touch as little data as you can
 - Archive, [pre]aggregate, purge. Get rid of unwanted data
 - Use the smallest data type you can
- Big tables are slower.
 - Slower to query
 - Slower to maintain
 - Slower to recover after a crash
 - Slower to run ALTER TABLE

Archive aggressively

- Use Maatkit's mk-archiver or roll your own
- Chop up older data and keep it in a separate table
- Split tables by date ranges
- Store old data in another server, or purge it entirely
- Goal: reduce the size of your working set
- Goal: keep your 16K InnoDB pages full of “hot” data
- Measure table growth
 - Know which tables are growing fastest
 - Archive them before they get too big to archive efficiently

Other big-data strategies

- Sharding/partitioning (more on this)
- Use InnoDB's clustered primary key
 - Especially good for keeping related data close together
 - For example, an ad-clicks-by-day table
 - Make the primary key (day, ad_id)

DNS: A crisis waiting to happen

- Build for zero reliance on DNS
- Treat DNS as completely uncontrollable
- Don't use round-robin DNS to load balance
- Set `skip_name_resolve` in your `my.cnf`

Anticipate lots of data

- Servers with large amounts of data take a long time to shut down
 - Use `innodb_max_dirty_pages_pct` trick
- They are slow to start, too
 - You must “warm them up” before they're even usable
- Don't rely on the `INFORMATION_SCHEMA` on servers with lots of data.
 - In other words, don't rely on it on small servers either... a small server is just a big server waiting to happen

Backups

- Using mysqldump does not work on lots of data
- Use LVM snapshots on a slave and copy the files elsewhere
 - Then start a MySQL instance on them and let InnoDB perform recovery
 - Then use CHECK TABLES to prove the backup is good
 - Remember to test for data integrity on the slave
- Practice recovery before a crisis hits
- Enable binary logs and keep enough for PITR
- Good practices are in our book

Don't use MySQL

- Remove MySQL from your critical path
 - Click counting usually does NOT need to be real-time
 - Design your app to run without the DB server up (as much as possible)
- Don't reinvent full-text search
 - MySQL's full-text search is better than LIKE
 - Sphinx and Lucene are even better

Scaling really big

- MySQL Cluster is not what you think it is, and it won't save you
 - The same is true for MMM clusters
- It is presently pretty much impossible to build a cluster that looks and acts just like a single machine
 - Even if it were possible, it wouldn't be cost-effective
- Your app has to have smarts built into it
- Sharding is the only way to scale writes

Master-master replication for HA

- This is a HA solution, not a scaling solution
- Make only one node writable
 - NEVER write to both masters! Chaos will result.
- The “passive” master can be used for maintenance
 - Offline OPTIMIZE
 - Offline ALTER TABLE
 - Offline backups
- The MMM toolkit can help, but be careful
 - It is tricky to set up correctly
 - <http://code.google.com/p/mysql-master-master/>

Protect the database

- Memcached
 - Enough said
 - Actually, there are others... and they are useful
- Consider caching proxies
- Consider reverse proxies
- Take it easy on Apache
 - You can use lighttpd or nginx
- See the “application optimization” chapter in the book

Measure and Profile

- The importance of good metrics can't be overstated
 - Measure everything you can
 - You need data for forensics, capacity planning, tuning, etc
- Cacti, Ganglia, MRTG, Munin: pick one
 - <http://code.google.com/p/mysql-cacti-templates/>
 - Also capture disk, network, memory, CPU...
- Nagios for alerting when things go bad
 - There are others but beware of complexity
 - Don't forget to monitor your RAID controller
- Build profiling into your code
- Patch your servers with our microslow log patches
 - Learn how to analyze the slow log

Be a scientist and engineer

- Automate everything you can
 - Make everything repeatable and scripted
- Use proper software engineering principles
- Use version control and sane release policies
 - Version your config files and your database schema
- Private developer DB instances (mysql-sandbox)
- Unit test suites can really save you during upgrades
- Document
 - Wiki, wiki, wiki – just pick one. I like Dokuwiki
- Use an issue-tracking system
 - I like Trac, and it has a very nice wiki too

Especially for Offerpal partners

- Your job may become about turning numbers into knowledge
 - Finding patterns in your data
 - Aggregating large amounts of data
- Prefer facts over predictions
 - If possible, ask! Use surveys, polls, capture data from user searches, etc
- Totally different performance characteristics
 - Most facebook apps usually deal with a few rows at a time
 - Harder to optimize big complex queries
 - Mixed workloads are even harder yet