PERCONA

Avoiding common traps when designing a MySQL application

Stéphane Combaudon
January 16th, 2013

Agenda

- Architecture for scaling
- Configuration
- Schema/Indexes
- Queries
- Hardware
- Backup/Recovery
- Instrumentation

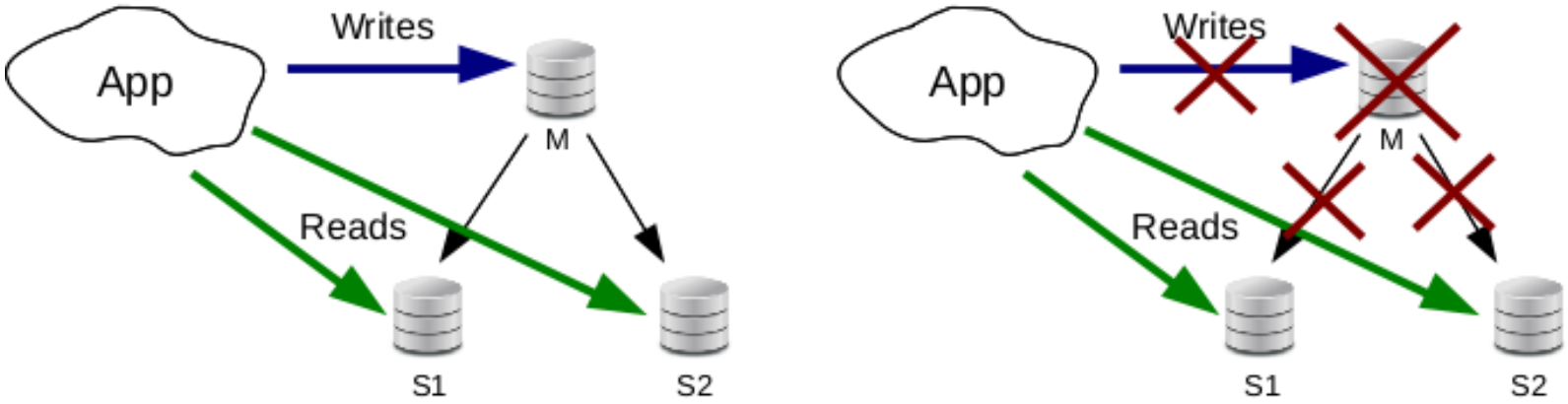
Architecture for scaling

- Common mistake: sharding is a must
  - Only if a single server can't handle the write load
  - Adds lots of complications
  - Functional partitioning is safer and easier
  - Read-mostly apps scale well with replication
  - Not every app is easy to shard
- Keep it simple
  - Not every application is Facebook size!
  - Operations are cheaper if the architecture is simple

Typical replication topologies

- Master-master
  - Great to improve HA, not to improve write capacity
  - Never write on both masters!
- Master-slaves
  - The “standard” setup, very good to scale a read-mostly application
  - Promoting a slave if the master crashes is not as easy as it seems

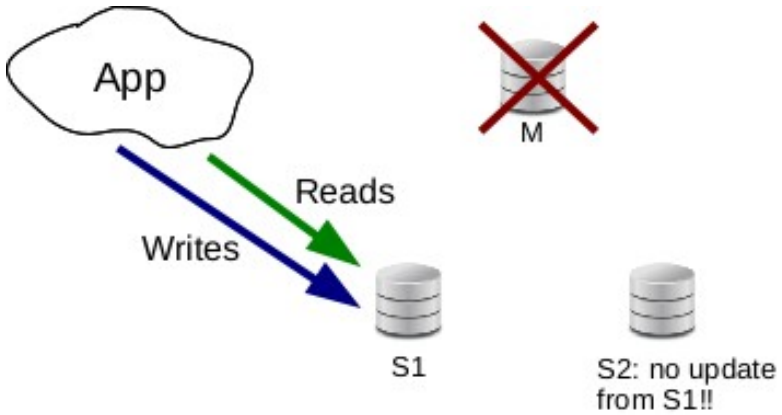
Master-slave: master crash



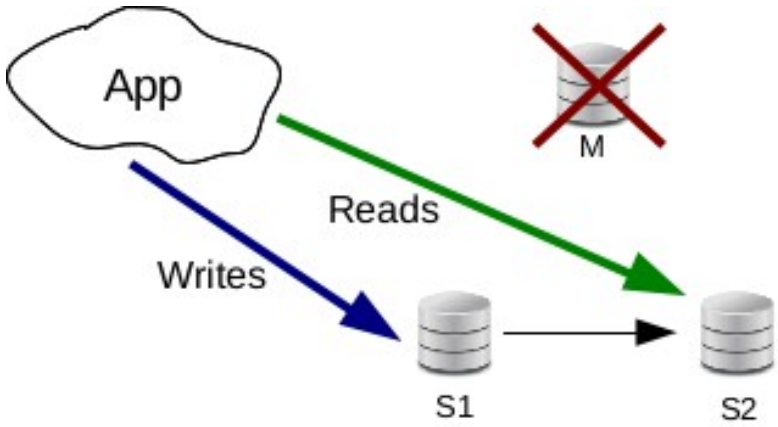
- If master crashes
  - No writes available
  - You have to promote one of the slaves

Promoting a slave

- Promoting S1 is not enough



- Need to set up replication between S1 and S2



Tools for slave promotion

- Tools can help you
  - Well documented and well tested procedure for manual promotion
  - MHA
  - PRM
  - ...

Configuration

- Most common traps
  - Keep the default my.cnf
  - Spend weeks to fine-tune every setting
  - Google tuning
  - New 2x powerful HW != 2x larger settings
  - Bigger is not always better
  - Changing 10 settings at a time
- These can lead to performance problems, instability and frustration

Essential settings

- InnoDB
  - Buffer pool: `innodb_buffer_pool_size`
  - Redo logs: `innodb_log_file_size`
  - `innodb_file_per_table` (but you can live without)
- No query cache (almost always correct)
  - `query_cache_type = 0`
  - `query_cache_size = 0`
- Done! Was it easier than you thought?

Easy to configure and helpful

- InnoDB durability: `innodb_flush_log_at_trx_commit`
- Sync binlog contents to disk at each commit: `sync_binlog = 1`

- Easy to configure by reading the doc
  - Size of the table_cache: `table_definition_cache`, `table_open_cache`
  - Size of thread cache: `thread_cache`
  - Upper limit of concurrent cnx: `max_connections`
  - Disabling DNS lookups: `skip_name_resolve`

What you should not configure

- Specialized buffers
  - `sort_buffer_size, join_buffer_size, ...`
- Esoteric settings
  - `innodb_concurrency_tickets, back_log, ...`
- Obsolete or unused settings
  - `thread_concurrency, master-host, ...`
- Change them only if you do know what you're doing

Conf. for master and slaves

- If same HW, conf. should be approx. the same
- You can take shortcuts on slaves for perf.
  - No binary logging
  - Relaxed InnoDB durability
  - `read_only` parameter to avoid most accidental writes
- If you promote a slave, don't forget to change its configuration!

Schema

- Real performance killer designs when misused
  - Having everything in a BLOB column
  - Entity-Attribute-Value
- Is normalization a performance killer?
  - It increases the number of joins
  - And joins increase the number of random ops
  - Ok, but it's only true for hot spots in high load apps

(De)Normalization

- Normalize and index correctly first
  - Indexes can: filter, sort AND cover (AND, not OR!!)
  - SSDs mitigate the cost of random ops
- Denormalize if some queries become slow
  - Some combinations of filtering/sorting can't be solved with indexes
  - Some queries will have to read lots of data
    - Often the case with COUNT(*) or GROUP BY queries

Other useful tips

- Don't be too generous when sizing
  - The whole size is used for implicit temp tables
  - Whenever possible
    - Use TINYINT/SMALLINT/MEDIUMINT instead of INT
    - Use small VARCHAR instead of VARCHAR(255)
- InnoDB tables and primary keys
  - The PK holds the data (clustering index)
  - It is implicitly included in all secondary keys
  - So set a PK explicitly, as short as possible

A common problem

- Let's design the user table of a social network

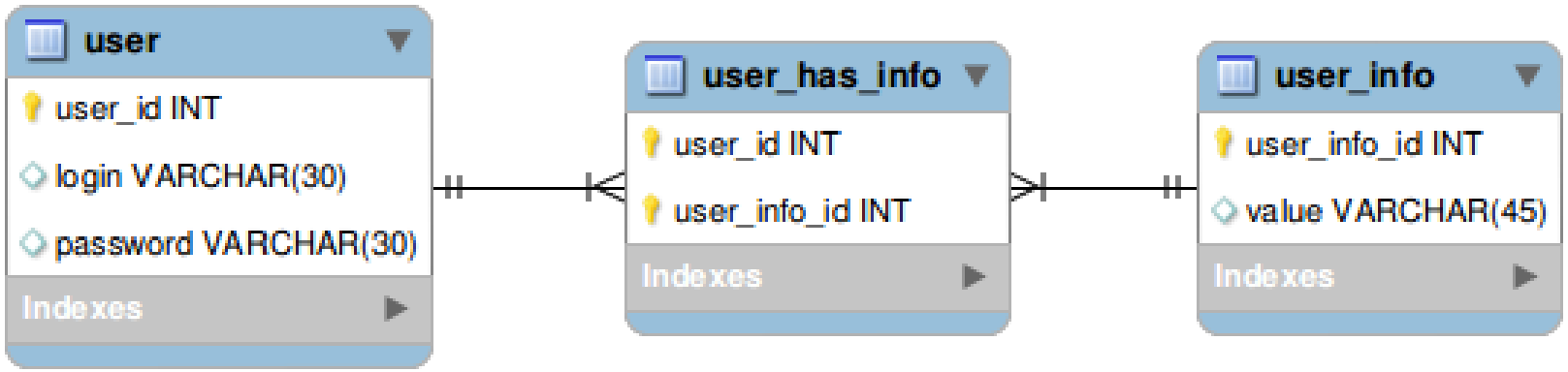
```
CREATE TABLE user(
  user_id INT AUTO_INCREMENT,
  login VARCHAR(30),
  password VARCHAR(30),
  PRIMARY KEY(user_id)
)ENGINE = InnoDB
```

  - Over time, you will add contact information(phone, address, …), preferences, etc
    - The table will quickly grow very big, queries will be slow
    - Fragmentation and slow ALTER TABLEs will be the norm

- What can you do?

A solution?

- Let's add 2 new tables:



- Now adding a property is easy and generic:
  - Insert a row in user_info to register the property
  - Insert a row in user_has_info for each user_id having the property

Pros and cons

- Benefits
  - You no longer need to alter the user table
  - Development is easier
- Drawbacks
  - The user_has_info table will not scale well
    - Every user will have 10s of rows in the table
  - Some queries are difficult to write efficiently
    - List of the users not having property xxx

Final thoughts

- In this specific situation:
  - Some properties can go to the user table
  - Some can go to the user_info table
  - Some will need dedicated tables
- As your application grows, you will have to deal with such situations
  - Be creative!
  - But try not to over-engineer

Queries

- If my queries are slow, what can I do?
  - “Well, buy RAM and SSDs”
- Not always the solution
  - Cost
  - Physical limit of your HW
  - Contentions in MySQL
  - What if your queries are just waiting to sth external to the database?

Improving queries

- Means improving response time
  - Low and stable response time is your goal
  - Stability is often overlooked
- How to make a query run faster?
  - Remove unnecessary work
  - Run necessary work as efficiently as possible

Remove unnecessary work

- Select only columns you really need
  - Exception: `SELECT *` can be useful for caching purposes
- Select only rows you really need
  - Use a `LIMIT N` clause if you want the top N results
- Use caching to offload the DB
  - “The fastest query is the query you don't run”

Optimize necessary work

- All access types are not equal
  - `type` column in `EXPLAIN` output: `index` (index scan) and `ALL` (full scan) are the worst ones
  - An index scan can be order of magnitudes slower than a full scan
- Rewriting queries
  - Subqueries may perform very badly (much better in 5.6 and in new versions of MariaDB)
  - Use INNER JOIN instead of LEFT JOIN when possible

Indexing

- Correct indexing is key to good performance
  - Not as easy as it seems
  - Too few idx kill perf, too many idx kill perf too...
  - Tools can help you find improvements
    - pt-duplicate-key-checker will find duplicate idx
    - pt-query-digest and pt-index-usage will help you find slow queries
    - user_statistics feature in MariaDB and Percona Server is useful to identify useless indexes

Hardware

- In short
  - Use commodity: it doesn't mean junk!
  - 24 cores, 128GB RAM, 640 GB SSD is still commodity
- CPU
  - No parallelization of query execution, so fast CPUs are needed for good response times
  - MySQL scalability has greatly improved

RAM

- Memory
  - MySQL uses RAM to cache index/data and for buffers
  - If possible you want your working set to be cached in memory
- Working set?
- Fraction of data that is accessed frequently
  - Not easy to estimate it (1% to 100% of your data)

Disks

- The world is moving to flash storage
  - Much more IOPS and lower latency than HDDs
  - Especially good at random IOPS
  - SLC (performance) vs MLC (cost, capacity)
- You may need to update your my.cnf to take advantage of Flash
  - Percona Server and MariaDB offer the most flexibility
  - MySQL 5.6 is catching up

Flash technologies

- SSD
  - SATA interface: drop-in replacement for HDDs
  - You need RAID like for HDDs
- PCIe devices
  - Needs special drivers
  - Better performance than SSD
  - No need for RAID

SSD/HDD/RAM usage patterns

- Mixing HDDs and Flash
  - Flash for hot data / HDDs for archives
  - Flash for data files / HDDs for InnoDB redo logs
- More RAM is often the best, but not if
  - The amount of RAM is limited
  - You have a high-throughput write workload
    - You can't delay the writes forever!

HW for master and slaves

- Things to keep in mind
  - Slaves must be able to keep up with the master's write load
  - If you promote a slave, it should be as powerful as the master
- Common choices
  - Same HW for master and slaves
  - New HW for master, master's old HW for slave
  - Flash for slaves, HDDs for master

Backup/Recovery

- Typical mistake: focus on backup
  - A backup you can't restore is useless
  - So focus on restoring instead of backing up!

- Different needs
  - For backups: low-impact required, quick if possible
  - For restores: quick required, low-impact if possible

Defining the right strategy

- First you should know your RPO and RTO
  - RPO: Recovery Point Objective, ie how much data can you lose?
  - RTO: Recovery Time Objective, ie how much downtime can you afford?
- Relaxed RPO and RTO means you will have more options to choose a tool

Different kinds of backup

- Logical backups
  - Text files, easily readable, editable (grep, sed, awk)
  - Flexible – see list of options for mysqldump
  - Restoring is VERY slow
- Raw backups
  - Binary files
  - Restoring the whole backup is fast
  - Often not obvious to restore a single table/db

mysqldump

- A must for small databases (max ~ 10GB)
  - Very flexible
  - Backups are fast
  - Restores are not too slow
- But unusable for larger databases (50GB+)
  - Restore time is a showstopper
- You may want to look at mydumper

XtraBackup

- Online raw backups with low-impact
- Full / incremental backup
- Moving single tables from server to server
- Parallel backups
- Streaming backups
- In active development
- And more...

Instrumentation/monitoring

- Monitoring/alerting will warn you when sth is wrong
  - Nagios, Zabbix...
  - Spend some time designing meaningful checks
- Graphing/trending
  - Cacti, Munin...
  - Will help you identify why things have broken
  - Graph everything you can from CPU usage to size of the InnoDB buffer pool

Instrumentation inside MySQL

- EXPLAIN
- SHOW PROFILE
- SHOW GLOBAL STATUS
- SHOW ENGINE INNODB STATUS
- SHOW ENGINE INNODB MUTEX
- INFORMATION_SCHEMA
- PERFORMANCE_SCHEMA

Instrumentation outside MySQL

- vmstat, iostat, mpstat...
- top, free...
- innotop
- Percona Toolkit
- Learn at least how to use some of the tools
  - Sometimes you need realtime diagnostics
  - Graphical tools always have a lag

Percona Live

Percona Live MySQL Conference & Expo

April 22-25, 2013

Santa Clara Convention Center & Hyatt Regency Santa Clara

4 days of breakout sessions, tutorials, and keynotes

Mingle with the MySQL community at the Welcome Reception and the Community Networking Reception

Visit www.percona.com/live for Details

- Thanks for attending!
- Q & A
- My email: stephane.combaudon@percona.com