MySQL Replication

Solutions & Enhancements

Shlomi Noach

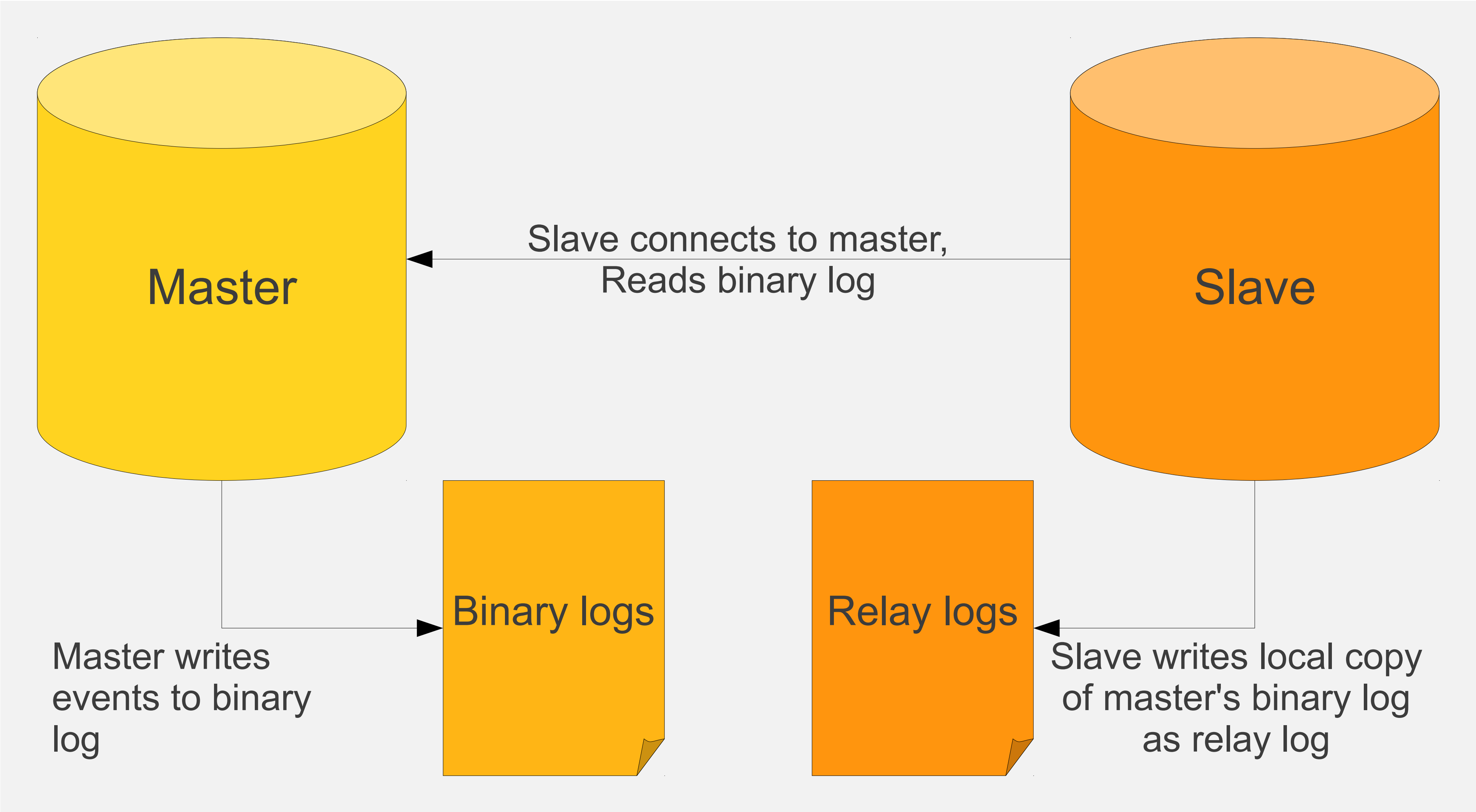
openark.org

June 2011

What is MySQL Replication?

- Replication is a mechanism built into MySQL.
  - It allows a MySQL server (Master) to log changes made to schema & data.
  - A second server (Slave) can pick up those logs and apply them.
  - When both servers are initially identical, it follows that a replicating slave reflects the schema & data on the master. Essentially, it becomes a duplicate of the master.

Replication workflow



Replication properties

- Replication is asynchronous. The master does not wait upon the slave.
  - With MySQL 5.5 semi-sync replication, this changes.
- Replication is only consistent only to some point.
  - While binary logs manage random, time & session variables info, it is possible to break consistency using nondeterministic functions.

Replication properties

- A master can have any number of slaves.
- A slave can only be connected to one master.
- The slave follows up on its master using binary log coordinates:
  - Binary log file
  - Position within binary log file
- The slave uses two threads:
  - One for reading master binary log and writing as relay log
  - One for applying relay log entries
- It follows that slave SQL execution is *single threaded*. We thus call replication to be single threaded.

Replication solutions

- Replication solves, or helps in solving a wide range of problems:
  - Scale out
  - Backups
  - High Availability
  - Version upgrades
  - Schema upgrades
  - Reporting
  - More...
- We discuss these, in no particular order

Backups

- Acquiring a MySQL server backup involves some interruption to normal workload
  - Some backup solutions require locks to be taken, if only for an instant moment
  - Increased I/O is usually noticed during backup time
- Solution: use replication, backup from slave
  - The master does not care if the slave is down, or lagging
  - Connections to the master are completely unaware of the slave being backed up

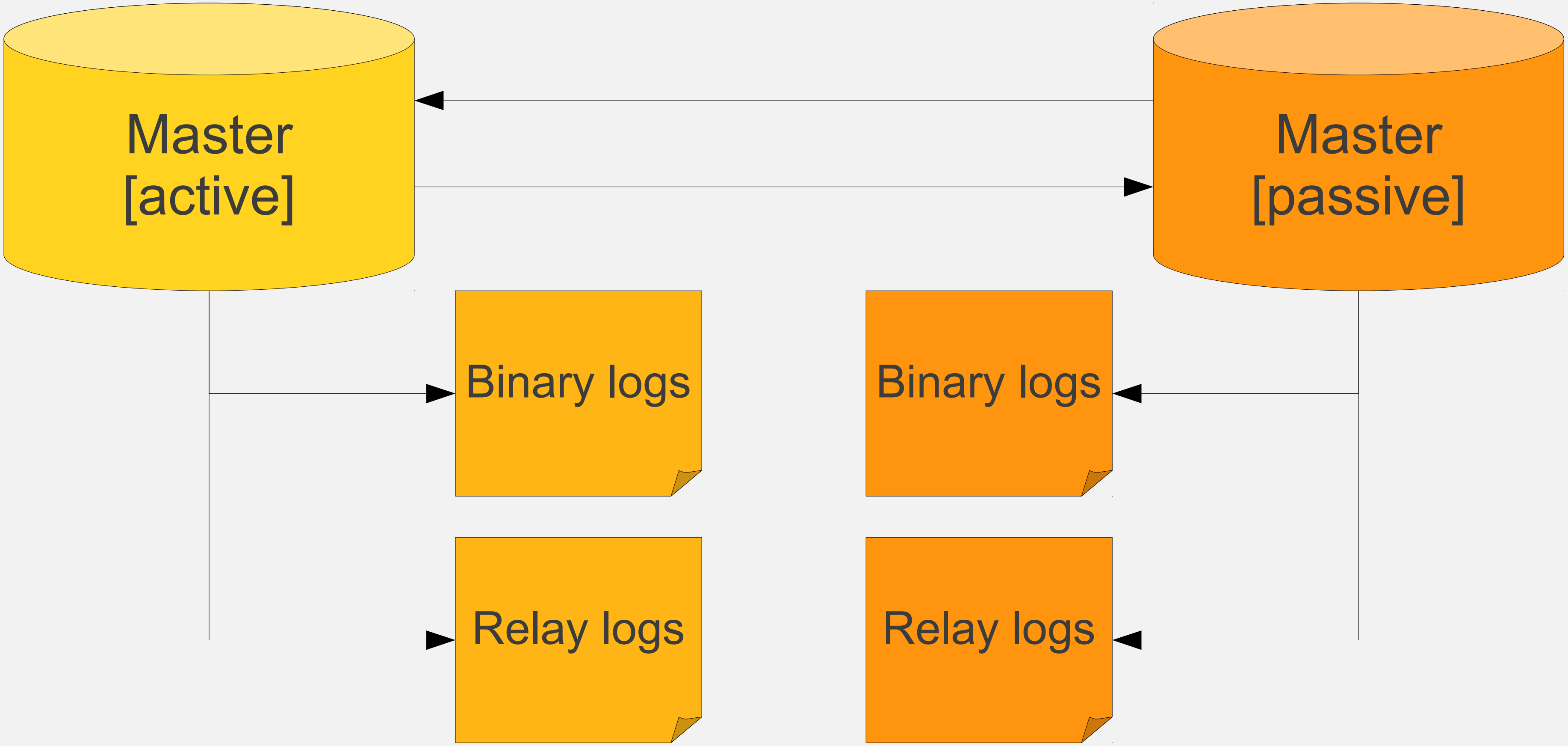
Backups

- Can we be absolutely certain that slave data is 100% compatible with master data?
- What can make for data changes (data drift)?
- We discuss shortly

Version upgrades

- Upgrading a MySQL version can always have hidden risks.
- There are many steps to make for a safer version upgrade
- Upgrading a slave's version is key part to the safe approach.
  - The master is unaware of the upgrade; the application is unaffected
  - By thoroughly testing behavior on slave, we lower the chances of getting affected by version changes or newly introduced bugs.
- See: http://www.mysqlperformanceblog.com/2010/01/05/upgrading-mysql/
- Once the slave is upgraded and tested, we switch over to the slave.
  - We call this *Slave promotion*.

Master-master replication



Each server replicates the other using standard replication.
The two servers are unaware of the cyclic architecture

High Availability

- A master-slave setup makes for a *High Availability* solution
- A master-master setup even more so.
- Since the slave follows up on the master, it makes for a hot standby replacement should the master go down.
- This is good in theory. Reality has its say, though...

Replication integrity issues

- Replication is asynchronous
  - The slave could be lagging far behind the master
  - Before promoting it to master (and before directing connections), we must verify it has completed executing all entries in relay log file.
  - MySQL **5.5** introduces *semi-synchronous replication*
    - A commit does not return to the user until the master verifies the entries have been written to a slave's relay log file
    - This makes for less possible lag time
    - It may also make for master slowdown

Replication integrity issues

- Replication is asynchronous in other respects as well:
  - By default, binary logs are not flushed to disk per entry.
    - A crashed server may not have flushed to file all entries.
    - Use **sync_binlog=1** for safest (and slower) setup
  - Relay logs and replication info files are not flushed to disk, either.
    - A crashed slave may lose its replication position, or worse, re-apply statements
    - MySQL **5.6** introduces crash safe threads: replication position is written to transaction logs
    - This feature already exists in **Google Patches** and **Percona Server**

Availability Zones

- The asynchronous nature of replication makes it particularly useful for setting up availability zones.
- A NY master can be replicated by a SF slave.
- In case of NY zone disaster (e.g. blackout), the SF slave can be promoted to master and pick from there.
  - While VPN easily solves the issue of securely transferring data coast-to-coast, replication also natively supports SSL.

Replication delay

- What happens when a user accidentally issues a **DROP DATABASE production_db**?
- Yes. It actually happens.
- Replication picks up on that command and executes it. The slaves become useless as well.
- We can explicitly make replication lag behind the master by, say, **1** hour.
  - This gives up time to detect the problem and fail over to the slave.
  - Use Maatkit's *mk-slave-delay*
  - MySQL **5.6** introduces time delayed replication

Schema upgrades

- On occasion, refactoring is required.
- An **ALTER TABLE** statement completely locks down the table.
  - On very large tables this could mean hours or days. of lock down.
- MySQL allows replication between tables of different schemata, as long as statements are equally valid on both. In particular, new columns or indexes are typically safe.
  - This depends on replication type (SBR vs. RBR) and constraints.

Schema upgrades

- This allows us to alter a table on the slave, without breaking down replication.
  - Replication will have to wait for the duration of refactoring.
  - It will take additional time to catch up with lost hours.
  - But during that time, master is unaffected
  - Once replication catches up, we can promote the slave.
  - This is particularly useful in a master-master setup.

Schema upgrades

- It does require bringing the slave down.
- There are tools which mimic the **ALTER** statement online, without interruption to normal work (although with overhead):
  - openark-kit's *oak-online-alter-table*
  - **Facebook**'s *osc* (Online Schema Change), derived from oak-online-alter-table
  - Maatkit's *mk-online-schema-change*, based on both.

Scale out

- Replication is by far the most common scale out solution for MySQL.
- **Google**, **Facebook** & **Wikipedia** are most well known for their large install base
  - Google and Facebook manage tens of thousands of servers, according to estimation
  - Both use a combination of sharding and replication.
- Replication is used both as high availability solution, but mostly for purposes of load balancing.

Scale out

- Assuming slaves are up-to-date, a read query can be executed against any slave.
- Replication makes for *read scale out*.
- Writes, however, must continue to execute on master.

Replication data drift

- There are several reasons why a slave would contain data not 100% compatible with master:
  - Statements which are nondeterministic in nature
  - I/O failures
  - Queries accidentally issued against the slave
  - Master/slave failures, with non synced logs
- Facebook recently estimated a **0.00056%** data drift between master and slaves.

Replication data drift

- Data drift can be detected, but with large data this is a lengthy process.
  - Use Maatkit's *mk-table-sync* to detect data changes.
- MySQL **5.6** introduces replication checksums
  - This already exists in **Google Patches** v3.

Thank you!

- I blog at http://openark.org
- Find open source projects on http://code.openark.org/forge/
- Questions?