**Replication**

**Topics are:**

* Overview of how it works
* Basic server setup
* Designing more advanced replication configurations
* Managing and optimizing your replicated servers

**Intro**

* Foundation for building large, high-performance applications on top of MySQL, using the so-called “scale-out” architecture
* Replication lets you configure one or more servers as replicas (or slaves) of another server
* Keeps their data synchronized with the master copy
* Not just useful for high-performance applications => also the cornerstone of many strategies for high availability, scalability, disaster recovery, backups, analysis, data warehousing, and many other tasks.
* Equally concerned with correctness and reliability when it comes to replication as to performance

**Overview**

* Basic Problem that replication solves is keeping one server’s data synchronized with another’s
* Many replicas can connect to a single master and stay in sync with it and a replica can act as a master
* Masters and replicas can be arranged in many different ways
* MySQL supports two kinds of replication: **statement-based replication (since MySQL 3.23)**and **row-based replication (since MySQL 5.1)**
* Both kinds work by recording changes in the master’s binary log and replaying the log on the replica
* Both are asynchronous => the replica’s copy of the data isn’t guaranteed to be up to date at any given instant
* No guarantees as to how large the latency on the replica might be
* Large queries can make the replica fall seconds, minutes, or even hours behind the master
* MySQL’s replication is mostly backward compatible => newer server can usually be a replica of an older server without trouble
* However, older versions of the server are often unable to serve as replicas of newer versions, because they might not understand new features or the SQL syntax the newer server uses, and there might be differences in the file formats replication uses
* For example, you can’t replicate from a MySQL 5.1 master to a MySQL 4.0 replica
* It’s a good idea to test your replication setup before upgrading from one major or minor version to another
* Upgrades within a minor version, such as from 5.1.51 to 5.1.58, are usually compatible
* Replication generally doesn’t add much overhead on the master
* It requires binary logging to be enabled on the master, which can have significant overhead => need that for proper backups and point-in-time recovery anyway
* Aside from binary logging, each attached replica also adds a little load (mostly network I/O) on the master during normal operation
* If you are replicating a very high-throughput workload (say, 5,000 or more transactions per second) to many replicas, the overhead of waking up all the replica threads to send them the events can add up
* Replication is relatively good for scaling reads, which you can direct to a replica, but it’s not a good way to scale writes unless you design it right
* Attaching many replicas to a master simply causes the writes to be done many times, once on each replica. The entire system is limited to the number of writes the weakest part can perform
* Replication is also wasteful with more than a few replicas, because it essentially duplicates a lot of data needlessly
* For example, a single master with 10 replicas has 11 copies of the same data and duplicates most of the same data in 11 different caches

**Problems Solved by Replication**

* Data distribution
  + MySQL’s replication is generally not bandwidth-intensive, though row-based replication in MySQL 5.1 can consume more bandwidth than statement-based replication
  + It’s useful for maintaining remote data copies, even over intermittent connections, but low replication lag requires a stable, low-latency link
* Load balancing
  + MySQL replication allows you to distribute read queries across multiple servers, making it ideal for read-intensive applications with basic load balancing achievable through simple code changes
  + Load distribution can be managed using methods from hardcoded hostnames and round-robin DNS to advanced solutions like network load balancers and the Linux Virtual Server (LVS) project
* Backups
  + Replication is a valuable technique for helping with backups
  + However, a replica is neither a backup nor a substitute for backups
* High availability and failover
  + Replication can help avoid making MySQL a single point of failure in your application
  + A good failover system involving replication can help reduce downtime significantly
* Testing MySQL upgrades
  + It’s common practice to set up a replica with an upgraded MySQL version and use it to ensure that your queries work as expected, before upgrading every instance

**How Replication Works**

1. The master records changes to its data in its binary log (These records are called *binary log events*)
2. The replica copies the master’s binary log events to its relay log
3. The replica replays the events in the relay log, applying the changes to its own data

* The first part of the process is binary logging on the master
* Just before each transaction that updates data completes on the master, the master records the changes in its binary log
* MySQL writes transactions serially in the binary log, even if the statements in the transactions were interleaved during execution
* After writing the events to the binary log, the master tells the storage engine(s) to commit the transactions
* The next step is for the replica to copy the master’s binary log to its own hard drive, into the so-called *relay log*
* It starts a worker thread, called the *I/O slave thread*, that opens an ordinary client connection to the master, then starts a special *binlog dump* process
* The binlog dump process reads events from the master’s binary log
* If it catches up to the master, it goes to sleep and waits for the master to signal it when there are new events
* The I/O thread writes the events to the replica’s relay log
* The *SQL slave thread* handles the last part of the process
* It reads and replays events from the relay log, thus updating the replica’s data to match the master’s
* As long as this thread keeps up with the I/O thread, the relay log usually stays in the operating system’s cache, so relay logs have very low overhead
* The events the SQL thread executes can optionally go into the replica’s own binary log, which is useful for scenarios we mention later
* Figure 10-1 showed only the two replication threads that run on the replica, but there’s also a thread on the master: like any connection to a MySQL server, the connection that the replica opens to the master starts a thread on the master.
* This replication architecture decouples the processes of fetching and replaying events on the replica, which allows them to be asynchronous => the I/O thread can work independently of the SQL thread
* This means updates that might have run in parallel (in different threads) on the master cannot be parallelized on the replica, because they’re executed in a single thread => performance bottleneck for many workloads
* There are some solutions to this, but most users are still subject to the single-threaded constraint

**Setting Up Replication**

* Most basic scenario is a freshly installed master and replica
  + Set up replication accounts on each server
  + Configure the master and replica
  + Instruct the replica to connect to and replicate from the master
* Assuming your servers are called server1 (IP address 192.168.0.1) and server2 (IP address 192.168.0.2)

**Creating Replication Accounts**

* MySQL has a few special privileges that let the replication processes run
* The slave I/O thread, which runs on the replica, makes a TCP/IP connection to the master
* Must create a user account on the master and give it the proper privileges, so the I/O thread can connect as that user and read the master’s binary log
  + **GRANT REPLICATION SLAVE, REPLICATION CLIENT ON \*.\***
* We create this user account on both the master and the replica
  + The account you use to monitor and manage replication will need the REPLICATION CLIENT privilege, and it’s easier to use the same account for both purposes (rather than creating a separate user account for this purpose)
  + If you set up the account on the master and then clone the replica from it, the replica will be set up correctly to act as a master, in case you want the replica and master to switch roles

**Configuring the Master and Replica**

* The next step is to enable a few settings on the master, which we assume is named server1
  + Need to enable binary logging and specify a server ID
* If binary logging wasn’t already specified in the master’s configuration file, you’ll need to restart MySQL
* You must explicitly specify a unique server ID => chose to use 10 instead of 1, because 1 is the default value a server will typically choose when no value is specified
* To verify that the binary log file is created on the master, run SHOW MASTER STATUS
* The replica requires a configuration in its *my.cnf* file like the master, and you’ll also need to restart MySQL on the replica
* Only the server\_id parameter is required on a replica, but we enabled log\_bin too, and we gave the binary log file an explicit name
* Also added two other optional configuration parameters: relay\_log (to specify the location and name of the relay log) and log\_slave\_updates (to make the replica log the replicated events to its own binary log)
* Latter option causes extra work for the replicas, but we have good reasons for adding these optional settings on every replica

**Starting the Replica**

* Next step is to tell the replica how to connect to the master and begin replaying its binary logs
* You should not use the *my.cnf* file for this; instead, use the CHANGE MASTER TO statement
* This statement replaces the corresponding *my.cnf* settings completely
* **CHANGE MASTER TO MASTER\_HOST='server1',**

-> **MASTER\_USER='repl',**-> **MASTER\_PASSWORD='p4ssword',**-> **MASTER\_LOG\_FILE='mysql-bin.000001',**

-> **MASTER\_LOG\_POS=0;**

* The Slave\_IO\_State, Slave\_IO\_Running, and Slave\_SQL\_Running columns show that the replication processes are not running
* Astute readers will also notice that the log position is 4 instead of 0 => 0 isn’t really a log position; it just means “at the start of the log file
* To start replication, run the following command: **START SLAVE;**
* Seconds\_Behind\_Master is no longer NULL => The I/O thread is waiting for an event from the master, which means it has fetched all of the master’s binary logs
* If you make a change on the master, you should see the various file and position settings increment on the replica => should also see the changes in the databases on the replica
* On the replica, you should see two threads. One is the I/O thread, and the other is the SQL thread: **SHOW PROCESSLIST\G**
* These processes will always run under the “system user” user account, but the other column values might vary

**Initializing a Replica from Another Server**

* The previous setup instructions assumed that you started the master and replica with the default initial data after a fresh installation
* Usually you’ll have a master that has been up and running for some time, and you’ll want to synchronize a freshly installed replica with the master, even though it doesn’t have the master’s data
* TODO

**Recommended Replication Configuration**

* TODO

**Statement-Based Replication**

* TODO

**Row-Based Replication**

* TODO

**Comparison between both**

* TODO

**Other Systems (master-master, multiple replicas, …)**

* TODO