Production MySQL Backups in the Enterprise: Part II

Thomas Weeks

Last month, I covered the historical problems of getting good backups with MySQL and described the common free and commercial tools for doing dump and raw backups. I outlined the pros and cons of each method, discussing the hot and warm online forms of dump and raw-based backup tools and then posed some basic requirement questions to help determine the right tool for the job. Now let's look at those questions again.

Which Type of Backup Is Right for You?

Q: Do you require online (either read-only or r/w) backups? If no, use "Service Stop & Copy" backups (using raw or dump). If yes, move on to next question.

Q: Can you afford any interruption (even brief) in write access? If yes, use warm lock/flush dump, hotcopy, or plain snapshot solution. If no, go to next question.

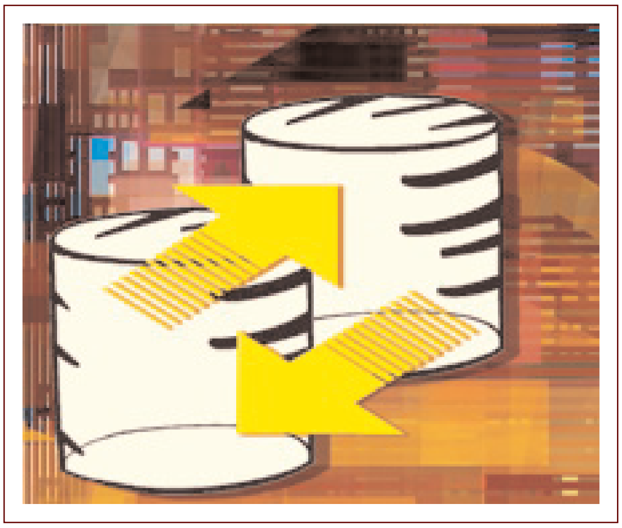
Q: Can you afford a dedicated replication-based backup system? If yes, build slave system (use mysqlsnapshot to set it up). If no, or if this is not a feasible solution for your environment, go to next question.

Q: Are you using InnoDB table types? If yes, good; use ibback or LVM-based snapshots for "hot backups". If no, convert to InnoDB table types and use ibbackup or snapshots.

Bear in mind that raw or snapshot-based backups using regular MyISAM tables will end up being warm/raw (i.e., they still need lock/flush). It's not until you use InnoDB table types that you get the true, hot, online backups, without some type of master/slave replication type of backup configuration. This method, in turn, requires additional hardware and network bandwidth to stay in sync.

Dump methods are commonly used for small or medium-sized DBs, raw file copying for medium-sized or many small DBs, and snapshots or the OTS solutions are a good fit for larger DB or many DBs using InnoDB table types.

Okay. We've beat this dead horse. Let's dive into some real-world examples and "do the needful" (IT translation = get the job done).



Using Mysqldump

The most time-tested way of doing online MySQL DB backups, the `mysqldump` command automatically locks and flushes your tables and saves the output to a SQL flat file from which you can easily restore. This form of backup is popular because it is good for all table types and because the output can be used when migrating between databases (e.g., Oracle to MySQL).

The following example shows how to dump three tables (users, addresses, and cars) from the single database named mydb1 out to the flat SQL file mydb1-tables.sql.

```
# mysqldump --opt mydb1 users addresses cars>mydb1-tables.sql
```

But what if you want to back up more than one database? The following example shows how:

```
# mysqldump --opt --databases mydb1 mydb2>mydb1-n-2.sql
```

And here is how to back up all local databases:

```
# mysqldump --opt --all-databases > all-your-dbs.sql
```

Here's another example. It shows a nice way of remotely using mysqldump via ssh and standard out to back up all databases on the remote server called dbserver.example.com to a local file called dbserver-backup.sql:

```
# ssh root@dbserver.example.com "mysqldump --opt \
  --all-databases" > dbserver-backup.sql
```

That last example is very powerful and is handy to run simply as a safeguard before doing some major change to your database server.

Keep in mind that mysqldump only gets the raw data from your databases. It does not get other external content such as bin-logs or index files. The indexes are usually okay, because they will be rebuilt when restoring from mysqldump. However, the bin-logs are critical if you need to do point-in-time recoveries. You will need to get these to a nice static location where your system-level (tape) backups can get them off the server for future recovery needs. If SSH is your remote session transport, then scp or sftp should work fine for getting those other, non-dumped files.

Before you just start using mysqldump, consider the tips below and look over the man page or MySQL AB's documentation on the command at:

```
http://mysql.com/mysqldump
```

Note that any MySQL command or tool that is documented by MySQL AB (the commercial company of MySQL) can be found by appending the command in question to the site's URL, like: http://mysql.com/COMMAND.

- For backup automation, create a dedicated MySQL backup/restore user (or use root) at the OS level and put the MySQL-based username/password in the user's home directory's ~/.my.cnf file, like this:

```
[client]
user = root
password = mysqlrootpassword
host = localhost
```

- Grant your backup/restore user (in mysql.user) permissions to at least SELECT, RELOAD, FILE (for backups), and INSERT, CREATE, DROP, INDEX (for restores).
- Enable binary logging (in /etc/my.cnf add bin-log under [mysqld]) to retain at least two full backups worth of binary logs and perform daily flushes (or flush daily via backup).
- Be sure to manage your binary logs and delete the ones you no longer need. Use the `mysqlbinlog` command to peer into your logs to see what you need to keep and what you can trash.
- The `--opt option` is on by default in >=v4.1 and includes the command-line options `--add-drop-table`, `--add-locks`, `--all`, `--extended-insert`, `--quick`, and `--lock-tables`.
- To get inter-DB referential integrity when backing up all databases, use `--all-databases` along with `--opt`.
- For incremental dump-based backups and to guarantee all DBs and their binary logs are in sync, use both `--opt` and `--flush-logs` (with log-bin enabled). Only this will allow "point-in-time recoveries".

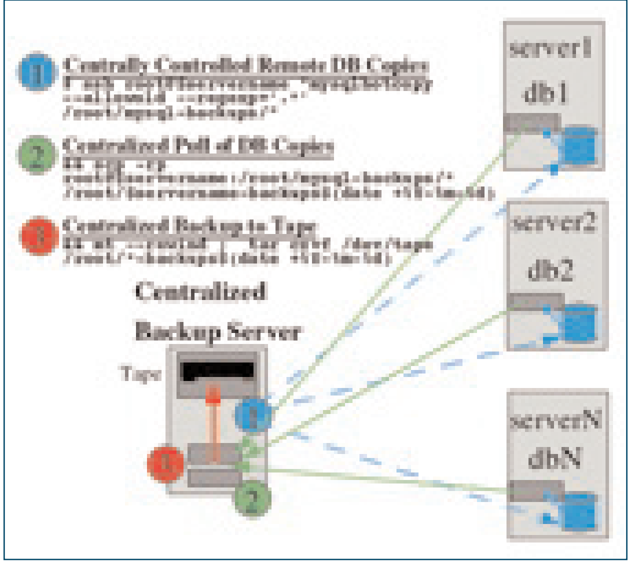

Centrally Controlled Remote DB Copies
Centralized Pull of DB Copies
Centralized Backup to Tape
Centralized
Backup Server
Tape
server1
db1
server2
db2
serverN
dbN

Figure 1 *Scripting ideas for automating multi-server DB backup pulls*

- For InnoDB table types (only), use the `--single-transaction` (MySQL v4.0.2 or higher). This will allow for "Hot Backups" with full r/w ("lock-less") access during the dump. As of 4.1.8, use this with the `--master-data` option for point-in-time recoveries.
- For large DBs (< v4.1), use `--quick` and `--extended-insert`, or in =>4.1, just use `--opt` (default). This will create dump files with combined inserts, greatly decreasing restore times on large dumps.
- To get faster restores on very large (single) DBs, use dump with the `-T` option to save out to a separate paired table.sql (structure) and table.txt (TSV data) files. Then for fast restores, use:

```
# mysql < table.sql && mysql -e LOAD DATA INFILE table.txt
```

- For MySQL on Windows, use the `--result-file=db.sql` option instead of standard output file redirect ">".

The mysqldump examples in this section can be a part of a company-wide database backup system. For example, you could easily set up a centralized network DB backup server with tape changer and dedicated DB staging space for getting everything centralized to disk and then to tape. Then, using SSH-based remote mysqldumps pulled back to your backup staging space, you can systematically get any DB server on the network conveniently and securely backed up to one central location. I'll show a variation of this type of strategy with mysqlhotcopy in the next section.

Using mysqlhotcopy

If you're only using ISAM or MyISAM table types, mysqlhotcopy is usually recommended over dump. Mysqlhotcopy is faster, simpler, has smaller output, and yields "drag and drop" restores on some MySQL versions. Mysqlhotcopy gets all of your database files (i.e., all *.frm (format), *.MYI (index) and *.MYD (tables) files) in your database directory /var/lib/mysql/databasename/.

Mysqlhotcopy Examples

Here is how one would get a local backup of just the users table from the mydb1 DB, stored in the secure /root/mysql-backups/ directory:

```
# mysqlhotcopy --allowold mydb1./users/  /root/mysql-backups/
```

Restoration of such backup files is quick and easy. You simply shut off the database, copy the files back into your DB's directory (e.g., /var/lib/mysql/db-name/ for many Linux systems), and start the database backup.

Here is how to back up a single database called "mydb1" to /root/mysql-backups/ using regular expressions to ID it:

```
# mysqlhotcopy --allowold --regexp=mydb1 /root/mysql-backups/
```

The `--allowold` option will ensure that that the backup will not abort if there is already an old copy in the backup directory.

Here's the same operation, except this time we're getting all databases with the ".*" regular expression and keeping the last old backup of the database also:

```
# mysqlhotcopy --keepold --allowold \
  --regexp=".*"  /root/mysql-backups/
```

Or, you can use this if you want nightly full backups of your database, without having to go dig for tapes somewhere:

```
# mkdir -p /root/mysql-backups_$(date +%Y-%m-%d) && mysqlhotcopy \
  --allowold  --regexp=".*"  /root/mysql-backups_$(date +%Y-%m-%d)
```

This creates a full backup nightly with its own directory name, like: /root/mysql-backups_2006-02-19/.

Warning: This type of nightly full DB backup can take up a lot of space over time. Be sure to keep your system from filling up or becoming too cluttered. I typically delete everything more than one week old.

Note that you can use a cronjob (or put in your main backup script) command to delete backups more than 7 days old like this:

```
# find /root/mysql-backups_* -maxdepth 0 -mtime +7 \
  -exec /bin/rm -rf {} \;
```

Now we can combine what we've learned about mysqlhotcopy with SSH and scp to do remote backups and backup downloads back to a centralized backup server (with tape or archiving media):

```
# ssh root@dbserver.example.com "mysqlhotcopy --allowold \
  --regexp='.*' /root/mysql-backups/" && scp -rp  \
  root@dbserver.example.com:/root/mysql-backups/* \
  /root/mysql-backups/
```

Very powerful indeed! This is the type of stuff that really makes you love the Unix/Linux command line.

Now that you have this type of remote power, a multi-DB server backup strategy can be easily implemented. See Figure 1 to get your creative backup script juices flowing. On a small to medium database, step #1 (where your DB would be read-only locked) executes in just seconds. All of your DB servers could be done at once within a known "read-only window". After this step, all of your databases will be unlocked (back to r/w mode), and you can proceed. In step #2, you would use scp to pull down each of the backups into a common directory locally. In step #3, you archive them all to tape.

This is a particularly popular solution for sys admins who have been handed a network full of MySQL application servers that are already in production and which they are now being asked to back up.

If you're using MyISAM table types and you don't mind a couple of seconds of read locks, mysqlhotcopy is definitely the way to go. Now let's look at some useful tips when using mysqlhotcopy.

Mysqlhotcopy Tips

Here are some additional tips to keep in mind before relying on mysqlhotcopy as a part of your overall MySQL backup strategy:

- Use `--allowold` so that hotcopy won't abort if it encounters existing backup data.

- Use `--keepold` to keep the previous backup after current one succeeds. Backup DB directories receive the _old suffix.
- To skip backing up the full index files (.MYI), use the `--noindices` option.
- If running binary logs, use the `--flushlog` option to flush them after the tables are locked. This is important for point-in-time recoveries.
- Don't forget to get a copy of your binary logs. They are stored in the same directory as your DB directory name (e.g., /var/lig/mysql/localhost-bin.*).
- For automating password-less and secure network backup/downloads, set up SSH key-based authentication. (For detailed directions on setting up key-based SSH authentication, you can just Google +ssh-keygen +authorized_keys.)

Now that we've covered mysqlhotcopy, let's take a look at snapshot-based backups and other OTS tools.

Snapshot-Based DB Backups and Commercial Tools

If you have the luxury of designing and building the actual system that your corporate MySQL databases reside on, then you can take the time to build the database partitions atop an LVM system with a separate file system for the database that you control, such as Linux+LVM2 and XFS. Additionally, if you can convince your DBAs to use InnoDB table types, then this can yield a 100% hot backup system that will gain you praise far and wide.

LVM/XFS Snapshot Example (Linux/LVM2/XFS)

As previously discussed, if you are backing up InnoDB table types, LVM/file system snapshots are fine without flush locks. The files on the disk are consistent and can be rebuilt from the undo/redo logs.

Here is an example of the order of operations that you might use to do LVM DB snapshots on a system that is properly configured with the required partitions, LVM, and file systems:

- Freeze the XFS file system on the LVM-based volume that you want to take a snapshot of (this is okay because the InnoDB databases are running out of RAM when the DB is active).
- Create a 500MB snapshot of the "prod" LV called "vgmysql-snap".
- Mount the new snapshot to the mount point /var/lib/mysql-snap.
- Unfreeze the previously frozen production file system.
- Tar the snapshot out to tape, unmount, and destroy the snapshot.

Here is what it would look like typed out from the command line, or what you would need to include in your snapshot backup script:

```
# xfs_freeze -f /var/lib/mysql \
&& lvcreate -L 500M -s -n snap /dev/vgmysql/prod \
&& mount -o nouuid,ro /dev/mapper/vgmysql-snap /var/lib/mysql-snap \
;xfs_freeze -u /var/lib/mysql \
&& tar czvf /dev/tape /var/lib/mysql-snap
&& umount /var/lib/mysql-snap \
&& lvremove -f /dev/vgmysql/snap
```

The actual freeze, snapshot, mount, and unfreeze all happen in just a couple of seconds. Meanwhile the InnoDB database runs in full hot (online r/w) mode without a second thought.

Gotchas with MyISAM Snapshots (Warm Only)

As previously stated, if you are backing up InnoDB table types, hot snapshots are fine, because the files on disk can always be considered consistent. However, when performing LVM/filesystem snapshots on My/ISAM tables, you must first issue a FLUSH TABLES WITH READ LOCKS before starting the snapshot and then an UNLOCK TABLES after the snapshot is complete. This is what such a backup script would look like functionally:

```
FLUSH TABLES WITH READ LOCK
-do snapshot here-
UNLOCK TABLES
```

After the snapshot is complete, then you can mount the snapshot and back it up to your backup media.

Hot/Raw Backups with the OTS InnoDB Hot Backup Package

After covering all the pros, cons, and little details of all the other DIY methods, the usage and simplicity of using InnoDB hot backup is really very appealing. To back up everything (InnoDB tables only), simply issue:

```
# ibbackup /etc/my.cnf /etc/ibbackup.cnf
```

where the my.cnf is your MySQL server's config file, and the second config file is ibbackup's config file detailing where and how the backups are to be done.

To get both InnoDB and MyISAM table types and related files (warm), use the included Perl wrapper script, innobackup, in a similar fashion:

```
# innobackup --user=dba --password=xyz --compress \
  /etc/my.cnf /backups
```

Restores are also simple:

```
# ibbackup --restore  /etc/ibbackup.cnf
```

If you want hot/online, quick, and easy MySQL backups and you don't want to do it yourself, then InnoDB table types and the InnoDB Hot Backup (ibbackup) really are the way to go.

The Future of MySQL and Hot Backup API

Going beyond MySQL v4.x, MySQL AB is developing a full online MySQL Backup API (if Oracle doesn't buy them first). At the time of last month's writing, the Backup API was still being developed and so MySQL v5.1 beta was released without it. However, I have spoken with the backup team and the API is under active development and is slated (but not promised) for MySQL v5.2. Watch for news and reports in the MySQL Backup Forum here:

```
http://forums.mysql.com/list.php?28
```

Watch for news of inclusion of a libmysqlbackup.so library by the backup API folks Brian Aker and Greg "groggy" Lehey! Keep up the great work guys!

General Enterprise Backup and Restoration Tips

If you are more of a DBA and less of a sys admin, then doing the actual system backups (i.e., getting things to tape) may be the hard part for you. If that's the case, here are some tested backup tips that I've recommended over the years:

- Test: After even minor system upgrades or unrelated changes, *always* test your backups via restores to make sure nothing is broken.
- Monitor: Configure your systems to send backup success/failure notifications. You need to know if your backups start failing *before* you need to rely on them!
- Rotate: Rotate your tape pools offsite (e.g., to a remote fire safe), but always retain a local copy of your last full system backup.
- Clean: Regularly clean your tape drives (once a month on production systems) and backup hardware, and always keep extra cleaning cartridges on hand. Also have a laminated card of your tape drives error flash codes affixed to the backup server or tape drive itself.
- Buy Before: Purchase media *before* you need it. When budgeting, build your servers with TCO figures that include recurring annual tape/media replacement figures.
- Boot Media: Keep emergency boot media and any special backup hardware drivers physically with each backup server.
- Other Info: Also keep and maintain any system details, database versions, accounts/passwords, and related information needed for rebuilds physically with each of your servers. This is usually done by keeping a related ring binder for each server under your care. If properly done, this is an invaluable administrative practice.

Conclusion

I hope these two articles have helped you to formulate a solid production MySQL backup strategy. Even though there previously has not been a lot of information out there for formally doing MySQL backups, there is now. There are also other free backup tools available, and some serious commercial OTS backup packages, too. Understanding the tools and your own requirements can really help you get your production MySQL DB systems backed up safely to tape. Although this might still require a bit of manual work, it *can* be accomplished and in some cases can be done even in a full hot/online state.

The open source movement is continually evolving and refining MySQL, in most cases faster than limited closed source vendors can both fix and grow their own flagship products. Great people like those at MySQL AB (with assistance from the worldwide open source developer community) continually develop and empower MySQL with features like the new backup API. Commercial companies are also rallying behind MySQL and creating OTS backup tools, such as ibbackup and Acronis True Image. All the buzz around MySQL indicates that it has really matured quite well and has now gathered mainstream adoption momentum. In my opinion, MySQL should be the poster child for how open source is changing the world of commercial software. Viva open source!

References

1. This article is based on my presentation given at the 2005 MySQL user's conference — `http://www.mysqluc.com/cs/ \ mysqluc2005/view/e_sess/7055`
2. Mysqlsnapshot script download — `http://jeremy.zawodny.com/mysql/mysqlsnapshot/`
3. Some points are taken from my Linux Sys Admin/Troubleshooting book, *The Linux Troubleshooting Bible*, Wiley 2004 — `http://www.amazon.com/exec/ \ obidos/tg/detail/-/076456997X/`

Thomas holds a BS-EET/Telecom degree from Texas A&M. Since 1999, he has worked with Rackspace Managed Hosting in the roles of Sys Admin, Corporate Technical Trainer, Lead Systems Engineer, and liaison between customer support/security/product/engineering departments. Thomas is the author of the new book, Linux Troubleshooting Bible *(Wiley), and has been president of the San Antonio technology user group "X-otic Computer Systems of San Antonio" (xcssa.org) since 1996. "Tweeks" can be reached via his site:* `http://theweeks.org/` *or at:* `tweeksjunk2@theweeks.org`.