Production MySQL Backups in the Enterprise: Part I

Thomas Weeks

This article is the first of a two-part series in which I'll describe how to get your MySQL backed up to a safe, static state [1]. In this part, I'll provide an overview of the available technology and requirements. Next month, I will step through the tricks, tips, and methodologies for getting your data to a safe place according to your own online access and DB system requirements.

Before getting knee deep in the various tools and methods for doing effective production MySQL DB backups, in this article, I'll first examine the general types of backups that are available, look at the pros and cons of each, and consider some basic questions regarding the reality of production backup needs (a.k.a. requirements).

Introduction

I work at a large Managed Hosting company that is famous for managing customers' dedicated Internet server's needs, and making sure that those customers are safe and secure from data corruption. Around half of our 18,000 Internet servers are Linux-based Web/email servers, but as with any Linux Internet type offering, a large percentage of those servers (more than 6,000) are running and using the MySQL database atop some distro-specific flavor of GNU/Linux. I tell customers who are new to running their own Linux Web/DB servers that, "Regular system backups are like car insurance. You don't need it until you need it... but when you do, it had better be there or you're in trouble".

Even with basic system backups in place, many people less experienced with running Web/database servers often overlook the special needs of backing up a live database. And, if they're not told otherwise, they often don't find out about these special needs until it's too late.

Demystifying Databases Backup Needs to Laypersons

Some times a client or a boss will balk at the concept of needing specific database backups on top of OS backups. To combat such kickback, a good database backup analogy for the boss or pointy-haired IT manager who has to approve funding is the "Wiggling Paper and the Xerox Machine". The logical exchange goes something like this:

"Sir, have you ever tried to copy a document on a xerography machine while you're wiggling it?" "Well, yes." "What do you get?" "A bad copy." "Right; it's the same with open files and databases. Do you want to get a bad copy of our corporate data?" "Well... No!" "Then we either need to stop the wiggling while the copy is happening (stop the database) or purchase/create a system to track the wiggling so that we can get a good backup."

That usually does a good job of politely demystifying the database backup issue for them and helps you get the resources needed to implement your DB backup system. However, before you dive right into doing production-grade backups of your MySQL database, you should become familiar with the backup choices that are available, look at the pros and cons of each, and examine which best fits your needs.

History: The Traditional Problems with MySQL Backups

Traditionally, the open source community has dealt with the fact that MySQL had no native API or other tools for backup either by shutting down the database to back it up or by simply locking and dumping the database to flat files and allowing the system-level backups to get the static flat files to tape. While these methods may work just fine, they do not allow for hot backups (i.e., online with r/w access), if they're done online at all. For a time, the lack of polished backup tools relegated use of MySQL to the likes of hobbyists, low-budget mom and pop shops, or DBAs who really knew what they were doing. There were no "backups for the masses" with MySQL.

Note that the contextual term "database backup" here refers simply to getting the database content to a safe, static state where your system-wide backups can safely get them to tape. I will not be covering backups as they pertain to "system backups", archiving to tape, or other forms of offline or long-term storage.

There are still not many backup suites out there that know how to properly back up all MySQL table types and guarantee backups of only fully committed transactions, not to mention how to get good copies of your table space in general. This has tended to keep MySQL out of the enterprise DB mix and has contributed to its lowly reputation in the eyes of many IT managers — regardless of the fact that a competent DBA and backup engineer could make it sing and dance and get it safely backed up just like the big databases.

The situation has been changing for this robust and speedy open source database, though, and now there are several free backup tools available, as well as several full-blown COTS MySQL backup applications. Furthermore, the near future of MySQL is slated to include a universal table backup API that will allow all MySQL

engines to "talk" to tape backup agents and be backed up hot (online in full r/w mode).

Types of Database Backups

There are essentially two common do-it-yourself types of MySQL backups along with some less traditional and commercial solutions. The first two common types of MySQL backups are dump-based and raw-based backup solutions. Each type has its own subset of methods, tools, pros, and cons.

Dump-Based Backups

Dump-based backups are commonly seen as the traditional catch-all method of getting your various database tables to a safe, static place. This method is common partly because of its simplicity in dumping the database content to flat files. Flat files are just text-based files with the data and SQL statements needed to reconstruct the tables and/or databases. But dumps are also popular because of their universal acceptance. Most databases out there allow for this type of flat file backup (MS-SQL, Oracle, PostgreSQL, etc.), and there are some real advantages to this method, especially if you're migrating data between systems or converting the data from one SQL database engine to another (e.g., Oracle to MySQL).

A dump backup for MySQL, usually using the command-line tool mysqldump for example, is done by locking and flushing the tables and then dumping individual rows, tables, or DBs out to a file system-level flat file, or even directly to a backup device. As a result of the locking, this form of backup usually results in a warm (or read-only) database state during the backup. Such dumps can take from tens of seconds to hours.

Dump Methods

Local warm backups (online but read only) can be done simply with the mysqldump command running against a local table, a database, or all a server's databases. Dump methods can also be employed remotely from a whole group of database servers all back to one centralized backup server that can be staged for tape archiving. I will examine this type of remote dump-and-pull configuration later.

You usually do not see basic dump methods that yield hot backups (full online r/w access) until you start talking about creating specialized replication backup servers. Replication backup slave servers are dedicated servers that can be disconnected from the master and lock/flush dumped for the backup, thus leaving the master server in a "hot" state. This is a safe and effective method that many larger Web hosters use for their production MySQL systems — especially because the dump itself does not affect the production master server's system resources (e.g., CPU and RAM) during the backup.

Dump Pros:

- Simple — This method is simple to do and simple to integrate into existing pre-backup scripts.
- Open — Backups can be manually edited or grep'ed for data.
- Flexible — Good for migrating to other SQL engines or with replication.
- Universal — Good for all table types (and for converting table types).
- Remote — Can be easily configured to dump over the network.

Dump Cons:

- Fat — Can require more space/backup than DB files themselves.
- Slower — It's slower than RAW backup/restore and takes more CPU/RAM.
- Limited — It's limited to smaller DBs (due to dump and restore speed).

Raw Backups

Raw backup methods are not as common, probably because they are traditionally considered risky by the old timers. However, if done properly, raw backups can be a very powerful and efficient way to get your data to a safe place.

Raw backups basically work by flushing, freezing, and grabbing the binary DB files at the file system level in some way. Although this "raw backup" is usually faster than the dump process, there are different methods of getting the data depending on what kind of availability and speed you require.

Raw Backup Methods

A typical simplistic example of cold, or offline, raw backups is gracefully shutting down the database and copying all the DB files to a different location where your system tape backups can then archive them. This might seem old-fashioned or undesirable; however, many banks and other financial institutions still use this method where 100% rock-solid backups are required, but 100% database uptime availability is not. You may have even encountered this at your bank if you've ever tried logging into their Web site or phone system in the wee hours of the morning to get a real-time balance.

Warm, raw backups (online read only) are done by locking and cloning the raw DB files in some way. This can safely be done by locking and flushing the tables and then simply copying the DB files. Another example of warm, raw backups would be using a quick lock/flush and copy application, such as mysqlhotcopy.

As previously mentioned, indirect, hot, raw backups can be accomplished by designing your system with a second replication DB slave server that is dedicated to performing backups. With such a setup for doing backups, you simply break the connection from the master/slave (by either flush/locking or shutting down the slave), backing up the slave's DB files (by binary copying in the case of raw), then unlocking or restarting the slave and allowing it to re-sync with the master. Thus, you get a backup without ever locking the master DB server or dragging the production master's resources through the mud.

An example of getting hot (r/w), raw backups on a single server can be seen by using InnoDB table types together with LVM/snapshot-based backups, or by using one of several "other forms" of backups (covered in the next section), including commercial backup clients.

Here are some of the pros and cons of doing raw backups:

Raw Pros:

- Faster — Faster than dumps usually, since it's working at the file system level.
- Scalable — Somewhat scalable. Better for larger databases than plain dumps.
- Powerful — It can be implemented hot with online LVM snapshots.
- Friendly — Friendly to most table types and normalized DB designs.

Raw Cons:

- Limited — Limited to in-place restores or same DB table/version migrations.
- Monolithic — Because backups are binary and not easily editable/customizable.

Other Forms of MySQL Backups

There are a number of other forms of backups available; some use exotic DIY methodologies based on dump and raw methods outlined above, while some use commercial backup clients that "talk to" the database itself. One of the most powerful forms of DIY backups is that of using LVM and file system-level snapshots, which I'll look at next.

Snapshot Backups

The theory behind snapshots is that the file system and/or logical volume system abstracts you and the database application from direct disk access. When you ask it to, the LVM/file system duplicates entire files or volumes at an instant in time, handling all reads and writes separately from what's actually going to disk, and creating a point-in-time image for you on the target volume. If you are using InnoDB table types in MySQL, which are always consistent (or valid) on disk, then you can effectively get a solid, reliable, "hot image" of your database and all of its related files.

This hot-snapshot method, of course, is not hot for other table types that are not always guaranteed consistent on disk, such as MyISAM. And, to ensure consistency, you would need to perform a lock/flush during the snapshot. Although snapshots are nearly instantaneous, the fact that you still have to issue a lock/flush (with MyISAM) technically makes this type of backup "warm". It may effectively be considered hot on a non-busy server, but what it comes down to is that, if you require hot snapshots, you need to be running InnoDB table types.

Here are some general pros and cons for raw snapshots.

Snapshot Pros: (type of raw)

- Hot — If LVM2 (which supports r/w snaps) is used with InnoDB tables.
- Instant — At the file system level anyway.
- Scalable — Handled at the LVM/file system level.
- Many DBs — Good for many DBs, especially when using InnoDB tables.

Snapshot Cons:

- Complex — Needs file systems to be set up already before backups.
- Warm — If other table types (MyISAM) are used, it needs lock/flush.

Snapshot-based backups really are quite robust and desirable if you can meet the various file system and database requirements:

- Linux LVM2 or other logical volume system (e.g., Veritas VxVM)
- A file system that you can control/lock (e.g., XFS or Veritas VxFS)
- The reserved drive space to create and maintain LVM snapshots
- Database needs to be InnoDB table types (for hot snapshots)
- Binary logs turned on (for point-in-time recoveries)

Other Snapshots?

There is one other raw method that does what it calls snapshots, but it's not an LVM/filesystem-locking type of snapshot. It is more of a MySQL client-based script that does its own form of lock/flush and copy, getting all the DBs and bin-logs and tar-gzipping them up by DB name. It's called mysqlsnapshot [2]. Since it was only designed for MyISAM table types, I won't spend a lot of time on it here. However, it deserves honorable mention because it was designed for (and is very good at) copying all the files needed to set up a slave server from your live production DB. This can help you quickly get a slave set up if you want to use the replication backup slave method previously mentioned.

So, as I'm sure you can perceive, LVM/snapshot-based backups are powerful, but unlike basic dump or raw type backups, they require a bit of forethought and setup. Next month, I'll look at some real-world examples of how to actually perform snapshots. Now, let's take a closer look at replication-based backups.

Replication Slave Backups

Replication-based backups, as previously mentioned, can be set up if you don't mind throwing another server into the mix. Some people have been known to run multiple MySQL server daemons on the same hardware to implement a master/slave arrangement on the same server, but this type of configuration is not recommended in a production environment. You typically do not want your master production database and your slave backup instance fighting over the same system resources. I suppose you could hard-limit the slave daemon or run it in a virtual server on the master server, but you do not typically want to resort to this type of frugality in a production environment. If you can spare even just some hand-me-down hardware for setting up a dedicated backup slave, that would be better. Figure 1 shows a time-line/process diagram of a master and a slave replication backup setup.

The progression for this type of backup is shown as four steps:

1. Normal operation of r/w master and r/only slave.
2. The slave is either stopped or Lock/Read/Flushed.
3. The slave is then backed up to disk or tape (if stopped, then raw; if Lock/Flushed, then raw or dump).
4. Backup finished, and the slave is restarted or UNLOCKed.

Replication Slave Pros:

- Hot — Indirect backups of master (production) server.
- Disaster Recovery — The backup slave can be put into production use if the master ever has a catastrophic failure.

Replication Slave Cons:

- Expenses — Requires another whole server to implement.
- Network — Additional network traffic (recommend multi-homing).
- Not Scalable — Not as scalable as other solutions outlined (e.g., LVM/snapshots) on very busy servers (slave never catches up in time for next backup).

Backup Slave Tip: Be sure that you always save your MySQL replication files like master.info file, your relay logs, relay index, etc. You'll need them if you have to reconfigure after a restore.

A backup slave replication setup is easy to configure if you're already running a multi-database server or replication server environment. Just stand up another slave, write a LOCK/stop UNLOCK/start backup script, and you're done. That said, if you don't have the space or money to dedicate to this type of setup, you may want to look at another solution.

If you want to more information on setting up your first replication server for doing backups, please refer to the book *High Performance MySQL Optimization — Backups, Replication, Load Balancing & More* by Jeremy Zawodny and Derek J. Balling (O'Reilly 2004).

Commercial OTS MySQL Backup Tools

The pointy-haired IT manager types often insist on purchasing OTS software solutions. They somehow think that do-it-yourself solutions are doomed to mediocrity and eventual failure. If this sounds like your business environment, the following tools might be for you.

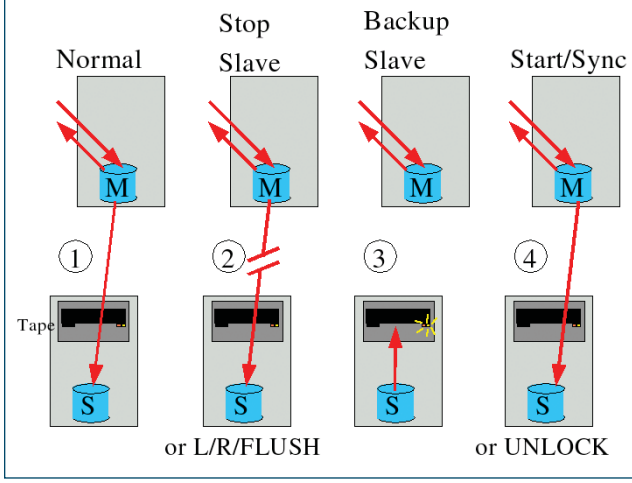
InnoDB Hot Backup, or ibback as it's known from the command line, is one of the most well-respected commercial MySQL backup tools available (`http://www.innodb.com/`, now owned by Oracle). It is actually designed by the creator of the innodb engine Heikki Tuuri and does a nice job of getting all InnoDB and MyISAM table types backed up safely.

This is usually the first tool that serious MySQL DBAs reach for if a commercial MySQL backup product is required. It is fairly priced (as compared to other big-boy, database backup client price tags) and is really a rock-solid performer.

True Image, by Acronis is the up-and-coming MySQL backup contender, but it's a full backup suite (DB and operating system) for both Linux and Windows. It is, at its heart, a commercial OTS version of the LVM/snapshot raw-based methods that we looked at earlier. The cool thing about True Image is that it offers a full-volume, image-level backup suite that does image clone-based differential and incremental backups as well as bare metal backup and recovery. Think of it as Norton Ghost on steroids but for both Windows and Linux. True Image even has disk-to-disk migration features and a special boot media environment that allows you to boot into your backup images.

After talking to one of the developers of True Image Server at the MySQL user conference last year, the only problem that I see

Figure 1 *Time-line view of the master/slave backup process*



with this tool on Linux is that it seems to rely on Linux kernel module-based control of the OS file system kernel calls to quiesce (and control) file system I/O. I really can't speak from experience about this tool, but I tend to be cautious about any tool that requires kernel-level wedge-ware. That being said, I would love to hear about readers' experiences with True Image Server, as it is giving MySQL DBAs more options, some of which are pretty nifty.

The Future of MySQL Backups: The Backup API

The much anticipated universal table backup API is slated for release in the upcoming MySQL v5.1. The significance of this backup API will make much of the previous discussion of backup tricks and methods moot. It also makes Oracle and Microsoft very nervous. In case you have not been following the news, Oracle is buying up pieces of MySQL and has even tried to buy MySQL AB itself, albeit unsuccessfully [3]. Stay tuned, as this is a hot topic, and the upcoming backup API is the new hotness that many a DBA has been waiting for.

Which Type of Backup Is Right for You?

So far I've presented the various types of backups and the pros and cons of each. But which one is right for you? By now you probably know, but go ahead and consider these basic questions anyway:

- Do you require online backups? If no, use "Service Stop & Copy" backups (using raw or dump). If yes, move on to next question.
- 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.
- 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.
- 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.

Conclusion

To wrap things up for this month, remember that raw backups or snapshots with regular MyISAM table types end up being warm, raw backups (i.e., they still need lock/flush). Additionally, it's not until you use InnoDB table types that you get the true hot, online backups, without some type of master/slave replication configuration (which requires additional hardware and network bandwidth to stay in sync).

Don't forget about the scalability issues. Dump methods and raw file copying are commonly used for small- to medium-sized DBs. Snapshots or OTS solutions are a good fit for larger DBs or many DBs using InnoDB tables.

Also, take a moment and jot down a few notes regarding the needs and requirements that you've identified for your MySQL databse environment. You will be able to springboard off those notes next month to begin to create your own custom MySQL backup solution.

Next Month: Do the Needful

Okay, now you've armed yourself with some preliminary information about the various forms of backups and have noted some important requirements about your own DB environment. You're now primed and ready for the details on getting the job done or, in other words, "doing the needful" [4].

Next month, I will show you how to do almost all of the MySQL backup methods discussed this month. I will show detailed examples and concise MySQL backup tips to allow you to piece together a custom backup solution for your production MySQL environment. I will also provide some ready-to-use, comprehensive MySQL backup system designs to tie everything together, so stay tuned.

Resources

1. Based on presentation "Ensuring Effective MySQL Backups on Enterprise Production Systems", at 2005 MySQL User Conference — http://conferences.oreillynet.com/cs/ \ mysqluc2005/view/e_sess/7055
2. Mysqlsnapshot — http://jeremy.zawodny.com/mysql/ \ mysqlsnapshot/
3. "Oracle tried to buy open-source MySQL", cnet — http://news.com.com/Oracle+tried+to+buy+open-source+ \ MySQL/2100-7344_3-6040197.html?tag=nefd.lede
4. See http://www.cafepress.com/theneedful

Tom 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.*