



Preventing MySQL Emergencies

A Percona White Paper

Baron Schwartz, Chief Performance Architect

Abstract

Preventing downtime-causing emergencies in MySQL can be difficult because they are caused by complex combinations of several things going wrong. Efforts to be proactive may be sincere, but without knowledge of the causes of emergencies, they often fail to prevent further problems. This white paper explains dozens of ways that real emergencies could have been prevented in production systems, and suggests specific actions to accomplish these. It grew from a study of hundreds of downtime-causing emergencies for Percona's customers, and is the companion to an article in the Q1 2011 issue of IOUG's *SELECT* magazine, which presents analysis of the causes and natures of the incidents. Each recommendation in this paper could have prevented at least one incidence of production downtime. The paper includes checklists that can be used to help perform and document the measures discussed.

The common measures to prevent emergencies by being "proactive" generally do not work. Searching the Internet for terms such as "proactive database administration" generally yields no concrete suggestions, other than checking log files and looking for bad queries. The key missing ingredient is knowledge of the real causes of database emergencies.

This paper is organized in sections of related suggestions intended to help avert emergency failures in MySQL-based applications. Each section explains several practices, and justifies them by explaining what types of failures they could have helped prevent.

This paper grew from a study of several hundred emergency incidents over the course of about a year. It is the companion to an article in the Q1 2011 issue of IOUG's *SELECT* magazine, which will also be republished as a white paper on Percona's website. This paper includes no analysis of the nature or causes of the emergencies, but focuses solely on the steps that could have prevented them. The *SELECT* article analyzes the nature and causes of downtime in these issues.

Following every suggestion in this paper would be extremely conservative. Each of the suggestions could have prevented an incident, but every incident could have been prevented by any one of a number of measures. There is never a single root cause of an emergency incident—there is always a chain of failures, and a single disruption to the chain can miti-

gate or prevent the emergency.

This paper should not be misconstrued as evidence that MySQL is inferior or flawed. Every complex software product has weaknesses, be they bugs or design limitations or dependencies on other systems. The intention is not to discredit the MySQL database server—to the contrary, if configured and administered properly, it is reliable, secure, and high performance. The hope is that readers see opportunities to improve their own systems based on the suggestions in this paper, and their systems become more reliable as a result.

Readers should beware of attempting to draw conclusions beyond the boundaries of the sample from which the emergency incidents were drawn. Because Percona's customers generally deploy MySQL on Unix systems, and particularly on Red Hat and Debian Linux, the emergencies and remedies are biased towards those operating systems. Most of the advice in this paper is relevant for MySQL versions 4.1 through 5.1; MySQL 5.5 is relatively recent and might present different problems and opportunities. Where appropriate, version-specific notes are added.

1 Application Stack Configuration

Failures in the application stack cause problems inside MySQL by subjecting it to unexpected load or by causing the application to be unable to connect to and query the database.

1.1 Examine cache expiration policy

When a cached item expires, it is possible for many processes to try to regenerate and populate the cache simultaneously, causing a storm of queries to the database server. Examine your cache expiration logic and ensure that this cannot happen. Prevention cannot rely on a low likelihood of a cache stampede; if it can happen, it will. Techniques for preventing a cache storm include auxiliary cache items that act as a lock or semaphore to ensure that only one process tries to regenerate the cache entry at a time; probabilistic (exponentially growing) pre-expiration and pre-regeneration; and background processes to regenerate the cache entries on a schedule, while setting the cache expiry time to more than that actually desired.

1.2 Inspect connection pool settings

Common failures in connection pooling are as follows: connections are not recycled periodically; connections can be returned to the pool dead, but the pool does not check that they are live and the application does not handle a dead connection gracefully; and connections are not reset to a default state properly, leaving context active such as open but idle transactions, temporary tables, or configuration variables. Beware, however, inefficient techniques to accomplish the above, such as sending a 'ping' to the database server constantly, or executing many queries upon retrieving a connection from the pool. These can add unacceptable load to the server, or introduce many round-trips to the application, causing relatively large latency especially for simple queries.

1.3 Test high availability and load-balancing

Many high-availability systems actually cause downtime because of naive techniques for failover. The key is to stress-test the systems under realistic conditions, such as a system being alive but unresponsive, or a failover when replication is delayed and there is active traffic on the system. Many failures we have observed came from problems such as the use of a non-journaling filesystem, or failure to stop queries from executing while promoting a

replica to replace its master. Load-balancing systems tend to fail in similar ways as high-availability systems—directing traffic to the wrong place at the wrong time. A common problem is using DNS for failover, load balancing, or read-write splitting. DNS cannot be controlled well enough to perform a clean, atomic traffic switch between servers.

2 MySQL Server Configuration

It can be counterproductive to spend too much time tuning MySQL, but basic configuration is very important. The following suggestions can help prevent serious problems.

2.1 Use InnoDB

MyISAM, the default storage engine in servers prior to MySQL 5.5, is not crash-safe or transactional, and has table-level locking. Unless there is a compelling need to use it (such as reliance on one of the features it provides), you should use InnoDB and configure it as the default. Use other storage engines only if truly necessary. Ensure that the server cannot start without InnoDB by setting `innodb=force` in the configuration file, if applicable for your version. Configure MySQL not to perform engine substitution to prevent silently using a different storage engine.

2.2 Configure buffer pool size and log file size

The two most important settings for InnoDB are the buffer pool size and the log file size. Nearly every other setting in the server can be left at its default without causing much harm—or at least, the defaults are acceptable in many cases—but these two settings cannot be left at their defaults, and are never acceptable for production servers.

2.3 Clean up the configuration file

The default configuration file that ships with many server distributions contains hundreds of lines of misleading comments and unnecessary settings. Most servers need a configuration file with only a dozen or two settings. A file full of comments and outdated advice is a temptation for a DBA with too much spare time, and makes it difficult to see what

has actually been changed from default values explicitly. Such large files are also frequently filled with duplicate and conflicting settings because it is impossible to fit onto one screen in a text editor and see what the real configuration is.

2.4 Use *mk-variable-advisor*

The *mk-variable-advisor* tool from Maatkit contains a set of rules that can help find many mistakes in configuration, without relying on simplistic rules of thumb. It does not give “tuning advice,” but rather identifies potential problems. It will be updated more frequently than this white paper, and should be considered as an aid to finding problems. However, as with any tool, its findings must be interpreted by a person familiar with the server and application.

3 Starting and Stopping MySQL

The currently available init scripts, whether from MySQL or those included in operating system distributions, have many shortcomings. A full solution will require much work and testing, but this section shows some simple steps you can take to reduce exposure to these problems.

3.1 Inspect startup errors carefully

If the init script claims that it failed, do not attempt to start MySQL again until you verify that there is no existing instance. The init script might have timed out during a process such as crash recovery, and the database needs to be left to complete this and will start normally. Re-executing the init script can cause conflicts as two instances try to gain exclusive control over the data files.

3.2 Disable automatic database initialization

The init scripts shipped with some distributions will check to see if there is a data directory upon startup, and if there is none or if it is empty, they will create a default database instance and then start the server. Disable this behavior. If the data directory fails to mount or does not exist for some reason, the server should not start.

3.3 Disable automatic startup scripts

Debian and Debian-derived distributions such as Ubuntu might attempt to run intrusive check scripts upon startup. These generally run `CHECK TABLES` on every table in the database. This can run for days on large servers with many tables, and it is even possible for several instances to run redundantly. Disable this behavior by editing the file `/etc/mysql/debian-start` to exit before it runs any commands.

3.4 Inspect shutdown errors carefully

If the init script fails during shutdown, check carefully to see whether MySQL is still running. It might simply be taking a long time to shut down, or it might be failing to shut down due to a missing pid file or similar problem. You might need to wait or issue a `kill` command. Do not issue a `kill -9` command if the server is simply taking a long time to shut down. You will lose any time you gained when the server restarts and has to perform recovery.

3.5 Flush buffers before shutdown

InnoDB’s shutdown sequence can take a long time as the database flushes dirty buffers to disk. This can cause unexpected unavailability of the server as it stops accepting new queries and simply flushes buffers. To avoid this, set the parameter `innodb_max_dirty_pages_pct` to 0 before shutting down. This will cause InnoDB to try to flush all dirty pages. This will have some performance impact, but the database server is likely to still respond to queries except in extreme cases. Watch the number of dirty pages in the buffer pool, and when it stabilizes or reaches a number that satisfies you, shut down the server.

3.6 Check for temporary tables before shutdown

Replicas using statement-based replication cannot be safely shut down if any temporary tables remain open for the replication thread. Replication will fail upon restart if there are any. Before shutting down a replica, issue `STOP SLAVE`, inspect the output of `SHOW GLOBAL STATUS`, and verify that

`Slave_open_temp_tables` is 0. If not, start the replication process again, and repeat the stop-and-check cycle until there are no open temporary tables.

3.7 Save replication parameters before shutdown

Before shutting down the server, stop replication and issue `SHOW SLAVE STATUS`. Save the result to a file and refer to it after restart to ensure that replication starts in the correct position.

3.8 Check configuration before shutdown

Before shutting down the server, check that the configuration file matches the contents of `SHOW VARIABLES` so that you do not restart the server in a different configuration than expected.

4 MySQL Server Logging

Logs are vital for diagnosing problems after the fact. Unfortunately, they often turn out to be configured incorrectly, so they are either useless for troubleshooting, or they cause problems during normal operation.

4.1 Use syslog for error logging

MySQL's file-based error logging is problematic because both the server and the `angel` program that watches it try to manage the logs. If you add in `logrotate`, which you should have, then you have three things fighting over the logs. The result is that error logs often get double-rotated, deleted, and truncated so the useful information is gone. Logging with `syslog` solves these problems.

4.2 Store the error log on a separate disk volume

If the error log is stored on the same disk volume as MySQL, and there is any trouble with the disk volume, then you will not be able to use the error log to help diagnose problems such as failure to start or messages about the volume being full. To avoid this, store the error log on a separate disk volume.

4.3 Verify that the error log is working

A smoothly running server might have no entries in the error log for months. But a server whose error logging is broken can be indistinguishable from one that has no errors. To verify that the error log is actually working, cause a deliberate error and check for it in the log. One method is to touch a new `.frm` file in a database directory, making MySQL believe a table exists; then issue a command that accesses this nonexistent table; and then remove the file again. The error log should contain a complaint about the missing table.

4.4 Segregate the error log and keep it long-term

System logs such as `/var/log/messages` are usually rotated away after a week or so. MySQL's error logs should be kept long-term; at least six months is helpful. In addition, it is appropriate to segregate the log from the system log on a server dedicated to MySQL. Configure `syslog` to log MySQL's errors separately from the general operating system log, and configure the `logrotate` policy for it appropriately.

4.5 Configure logrotate correctly

The `logrotate` scripts that are installed by default in many MySQL installations are faulty and can result in logs that are unlinked, but still resident on the filesystem. These logs can consume large amounts of disk space, even filling the disk completely. The problem is that the `FLUSH LOGS` command is not sent correctly to the server. The `logrotate` program usually runs with an empty environment, so the command `mysqladmin flush-logs` is unable to log into the MySQL server to which it connects. The best solution is to explicitly specify a defaults file with the `--defaults-file=` option to the `mysqladmin` program. Make sure that you check the results after a day or so, because `logrotate` that is executed from `cron` is different from `logrotate` executing manually from the command line.

In addition, the `logrotate` scripts perform unnecessary checks that `mysqladmin` is executable and that the server can be pinged. These two checks are op-

opportunities for false failures; remove them and simply execute the flush command.

Finally, if you wish to configure the server more securely and avoid the need for `logrotate` to actually connect and log in to the server, you can simply send a `SIGHUP` signal to the server; this triggers `FLUSH LOGS` inside the server. One way to do this is with the command `killall -HUP mysqld`. If you do this, `mysqladmin` is not needed at all.

5 Replication Configuration

Replication is one of MySQL's most-used features. It is vulnerable to several types of configuration errors, many of them simple, that can cause it to fail.

5.1 Configure all servers in read-only mode

The most common cause of replication failures is data conflicts on the replica, usually caused by something other than the replication process making changes there. Read-only replicas are an easy way to prevent many such errors. However, all servers—not just replicas—should start in read-only mode to protect the entire dataset from errors.

Configure all servers (replicas *and* masters) with the `read_only` option. Do not grant the `SUPER` privilege to users that do not need it, because it circumvents the read-only setting.

In general, any server that is involved in replication in any way should refuse writes until a human explicitly instructs it to leave read-only mode by disabling the read-only setting. This might seem inconvenient, but it is far better to lose a few extra minutes of uptime after a crash than to lose days of uptime because of a large-scale data recovery effort caused by conflicting writes. Setting read-only in the configuration file on *all* servers ensures that systems are properly configured if a replica is promoted to a master or vice versa. This is also a safe configuration option for systems such as the MMM replication manager.

It is not necessary or safe to disable the read-only setting on a replica after starting the server. Disable it only on the master. Replication is designed to function on a read-only server. Normal users will

be denied the ability to modify data, protecting the replica; replication is not denied this privilege.

5.2 Prevent replication from starting automatically

Many replication failures are caused by problems that can be corrected before replication starts, such as the replica connecting to the wrong position on its master and re-executing statements. If replication is not allowed to start automatically, then a human must connect to the replica and start it, and there is a chance to solve these problems. Before starting replication, check the error log for errors, and verify that the output of `SHOW SLAVE STATUS` matches what was saved at shutdown.

5.3 Permit large packets

Replication often sends much larger packets through the protocol than normal querying, so it is common for replication to fail with the error that the maximum packet size has been exceeded. To avoid this, configure the `max_allowed_packet` setting on both the master and the replica. The default setting is needlessly small. The maximum possible setting is 1GB and there is little harm in setting it that large, unless your server is exposed to traffic from anyone on the Internet, which it should not be.

5.4 Use MySQL 5.5's improvements

MySQL 5.5 includes many improvements to replication. Two “no-brainers” are to enable the master heartbeat and relay log recovery features. These features make replication more robust.

5.5 Keep enough binary logs

The master's `expire_logs_days` parameter instructs it to discard binary logs after a number of days. If a log is discarded and a replica still needs it for replication, then replication will fail. Ensure that the parameter is large enough to support the longest anticipated delay since the last backup. For example, if you use a backup to set up a new replica, and the data is from a week ago, then you need to keep at least a week's worth of binary logs, and preferably more as a safety margin.

5.6 Do not hard-code connection parameters

Replication configuration options that instruct the replica how and where to connect to a master server should never be placed in the `my.cnf` file. The contents of the `master.info` file override these, and if that file is lost, then the server will connect to a master at the earliest possible point and begin replication, potentially destroying the dataset and requiring it to be thrown away and initialized from scratch.

5.7 Do not use replication filters

Replication filters such as `binlog_do_db` are often used when they are not really needed. They can cause replication to break because of their unintuitive behavior. Unless absolutely needed, replication should not be filtered by database or table at all. The safest configuration is to replicate the entire server's data.

6 MySQL Server Versions

The MySQL server version makes a great deal of difference to avoiding emergencies. Older server versions contain unfixed bugs and pitfalls. How the server is compiled is also important. MySQL is easy to compile badly, and it is best to avoid compiling it yourself unless you are certain you need to.

6.1 Do not run a debug binary

A debug binary of MySQL is specially compiled with the `-DWITH_DEBUG` compilation option. This enables a number of code changes inside the server to help catch problems in development environments, such as unsafe memory allocation practices. These code changes carry an enormous performance overhead; a debug version of the server cannot be expected to run a production workload. You can detect a debug server build by looking for the existence of a variable named `debug` in the output of `SHOW VARIABLES`. If the variable exists, even if it has no value, then the server is not usable for production workloads.

6.2 Do not run a stripped binary

The topic of 'debug' and 'stripped' binaries is confusing for many MySQL users. In the MySQL terminology, a debug binary is one built with `-DWITH_DEBUG`, as mentioned previously. This is bad; but another common meaning for 'debug binary' is one that contains debugging symbols for use by `gdb`, `oprofile`, and other tools. This is a good thing, because it lets users inspect the server without adding any overhead in normal operation, aside from making the server binary slightly larger. Servers that are 'stripped' do not contain debugging symbols, and tasks such as understanding stack traces and profiling are either much more difficult or impossible. You can detect a stripped binary by executing the `nm` command on it; if it reports "no symbols," then it is a good idea to replace the server with a version that has symbols.

6.3 Use the latest release within your series

There is nothing wrong with running a server a few major versions old. Running an early release of an old version, however, can be very risky. In the 5.0 series, for example, InnoDB's performance was terrible until the late 5.0.30's, and new features such as triggers and stored procedures were extremely buggy in early server versions. Many serious replication problems were not fixed until the late 5.0.60's. You should never run MySQL 5.0.22, or 5.0.45, or 5.0.51b, or any of the other default versions that shipped with popular operating system distributions. If you want to run MySQL 5.0, upgrade it to the most recent release. Of course, you might not want to upgrade every time a new minor version is released, and that is understandable, but you should upgrade on a reasonable schedule.

7 Troublesome Server Features

Some features in MySQL are simply riskier than others due to hazards such as fragility, complexity, or ease of misuse. This section explains features that you should not use blindly. These features are beneficial in many scenarios, but if they are used, a careful review is probably indicated.

7.1 Triggers

The use of triggers can be risky because of the complex interplay between triggers, statement-based binary logging, and features such as auto-increment primary keys. The trigger actions are not always reliably reproducible through statement-based logging, so replicas might diverge from the master, and roll-forward recovery from binary logs might suffer the same problem. A common scenario where triggers might cause problems is as an audit mechanism, where records of changes are saved to an auxiliary table.

Beware also of the additional locking overhead and complexity of triggers, regardless of whether binary logging is used.

7.2 Views

Views can cause performance problems for two reasons: developers query the views without realizing that they encapsulate complex queries, and the query optimizer can suffer performance issues while generating an execution plan. Sometimes queries against views can be rewritten to exactly equivalent forms against the base tables with much better results. The code path within the server is different when views are involved, and this is something to treat with care.

7.3 Temporary tables

Temporary tables are unsafe to use with statement-based binary logging. There is no way to avoid the possibility that the server will be shut down or crashed while a temporary table is open.¹ If the server is a replica, then replication will fail after the server is brought back up. If the server's data is snapshotted for backup purposes while a temporary table is open, then point-in-time recovery from the backup with binary logs will be impossible. There are several alternatives to using temporary tables that do not suffer from these problems.

¹It does not matter whether the temporary table is used within a transaction, or only within the scope of a single stored procedure, or only for a single statement. The problems are unavoidable.

7.4 The query cache

The query cache is not designed for multi-core servers, and is a frequent cause of severe and intermittent server lockups that can be difficult to diagnose. Some workloads can benefit from the query cache, but that can be complex to determine. A good cache hit ratio can trick you into believing that the query cache is helping, when it might be harmful. A safe policy is to disable the query cache by default, and enable it only if you have the expertise to prove its benefit.

7.5 Reverse DNS lookups

MySQL's privilege system assigns privileges to users based in part on their source hostname and IP address, which requires MySQL to perform a forward and reverse DNS lookup upon login to avoid spoof attacks. When DNS servers fail to respond quickly, the server can effectively suffer a denial-of-service attack from login attempts. To avoid this, remove any hostname-based user accounts, and set the `skip_name_resolve` parameter so the server does not perform any DNS lookups.

7.6 Client tab-completion

Many client programs query the server to retrieve a list of objects and their names to help with such tasks as tab-completion. These queries can be very expensive and add a great deal of load to the server. To help prevent this, avoid connecting to production servers with GUI administration programs until you are familiar with their behavior, and place the `no-hash` option in the `[mysql]` section of your server's configuration file to prevent the `mysql` command-line client from retrieving tab-completion information.

8 Special Features in Percona Server

Percona Server is an enhanced version of the MySQL database server, with many improvements that make it more reliable and performant. Most of them are enabled by default, but some are not. If you use

Percona Server, you should take advantage of the following valuable features.

8.1 Enable fast warmup

Enable automatic LRU dump and restore to reduce server downtime for restarts on servers with large amounts of memory and/or slow disks. A powerful server can take hours or days to warm up enough to serve production traffic otherwise.

8.2 Enable user statistics

Percona Server can provide activity statistics for users, tables, and indexes, but this is disabled by default. Enabling it permits much valuable information to be gathered, such as finding unused indexes or recording statistics about how quickly the workload on individual tables is increasing.

8.3 Enable query response time distribution

Percona Server provides a feature to capture query response times in aggregate, which is ideal for graphing with a trending tool. This information can help with activities such as capacity planning and troubleshooting. It is disabled by default, but is very valuable to enable.

8.4 Enable the dedicated purge thread

Percona Server lets you create a dedicated thread for purging old row versions from InnoDB, which helps stabilize server performance under high load and prevents stalls and lockups.

8.5 Enable better handling of corruption

Percona Server lets you configure the server to continue to run if a table is corrupted, instead of crashing the entire server.

8.6 Install UDFs for fast checksums

Percona Server redistributes some UDFs (user-defined functions) from the Maatkit toolkit to enable much faster and more reliable data checksums.

Maatkit will use these automatically if they are installed. This makes operations such as verifying replication integrity faster and helps avoid blocking or other performance impacts. The UDFs are installed automatically into the filesystem and simply need to be installed into the server with SQL commands.

8.7 Configure verbose query logs

The so-called “slow query log” is greatly enhanced in Percona Server to provide much more information about query execution. Configure this additional information to perform richer analysis of the server’s workload.

9 Development Instances

Many incidences of database downtime are caused by interactions between development environments and production environments. The practices in this section are suggested to help alleviate this problem.

9.1 Segregate development instances

Development databases should be running on a different server instance, on a different server, on different physical hardware from production. This recommendation should be balanced against your budget. If separate hardware is not feasible, then perhaps separate virtual machines are, for example. The more segregation between development and production, the better. Ideally there should be no shared resources such as sharing drives in a SAN.

9.2 Restrict developer accounts

Many disasters happen because users type commands into the wrong window accidentally. Each developer should have a separate login account to the development servers, and these accounts should not exist on the production servers. This makes it less likely that a developer will connect accidentally to a production instance and execute commands such as DDL. Developer accounts should also be limited at the operating system level; developers should not have the ability to restart or upgrade the production server.

9.3 Use different passwords in production

Applications are just as likely to misbehave as people. The development application instance should not be allowed to connect to the production servers. Use at least a different password, and preferably a different username as well, so that it is easy to notice a development process connecting to a production server.

9.4 Use login banners

Using a login message, such as `/etc/motd`, to indicate a server's function and status can help prevent many casual mistakes caused by executing commands on the wrong server. In addition, systems have a tendency to change function without changing their hostnames. No one who breaks something on a server that identifies itself as a test machine should be held responsible.

10 Basic Security Restrictions

Security practices are an area where some insist that no effort is too great, and others are satisfied with avoiding gross negligence. The following practices are relatively easy to implement and can be highly beneficial.

10.1 Restrict external access

To prevent malicious attacks, the database server should not be accessible from external IP addresses. Do not rely on MySQL's privilege system to restrict access; instead, use a firewall or similar technologies. An external IP address should not be able to open a TCP connection to the server's port 3306.

10.2 Remove unnecessary privileges

User accounts should have only the privileges they need to perform their work, within reason. (Following the principle of least privilege fully can be a great deal of work for minimal extra benefit.) In particular, the `SUPER` privilege should never be granted unless it is needed. A useful level of diligence is to differentiate between users that should have read

privileges and those that should have write privileges also. For example, when performing read-write splitting as part of a scaling strategy, it is a good idea to assign only `SELECT` privileges to the connections that are supposed to be read-only. This will prevent many simple bugs and mistakes. Similarly, replicas should connect to their masters with a restricted account. This is particularly important because the replication password is often relatively easy to compromise.

10.3 Restrict permissions to the data directory

Access to the data directory and files, usually located in `/var/lib/mysql/`, can permit a nosy or malicious user to read information that should be private, such as the replication username and password, which is stored in plain-text in the `master.info` file. The data directory and data files should not be world-readable.

10.4 Disable old-style passwords

Old-style passwords—those in use before MySQL version 4.1—are insecure because they are transmitted in plain text across the TCP connection when logging in. There has been a new and secure login handshake for many years, but many operating systems still ship MySQL with a configuration file that enables old passwords. The server should be configured not only to create secure passwords for new user accounts, but to reject any attempt to log in with the old insecure handshake. All user accounts should be checked to ensure that they have new-style passwords.

10.5 Disable anonymous access and default users

By default, new MySQL installations are created with several passwordless accounts and an anonymous user. These should be removed, and there should be no users in the system tables without a username or a password.

11 Storage Volumes

Many serious problems are caused by the volumes on which MySQL's data is stored. Proper setup can help avoid many of the simplest problems.

11.1 Configure the RAID controller

Some RAID controllers have defaults that can cause problems. MegaRAID controllers, including Dell's PERC controllers, have an automatic battery test cycle that disables the write cache for the duration of the test. This causes a sudden drop in write performance. You should disable such features, and schedule a time to perform them manually.

11.2 Name the volume meaningfully

Sometimes the simplest mistakes happen when people are tired or rushed. If the MySQL data volume's name includes the word "mysql" to differentiate it from others, it can help avoid problems such as accidentally unmounting or reformatting the volume. Similarly, it is a good idea to name LVM physical volumes, volume groups, and logical volumes with the word "mysql." Finally, the data directory's fully resolved path should include the word "mysql"—that is, if you change your working directory to the data directory and execute `/bin/pwd`, the output should include "mysql."

11.3 Store data outside the filesystem root

Servers whose data directory is located in the root directory of a mounted filesystem can encounter problems. First, the presence of a directory named `lost+found` can cause many tools to malfunction, believing it to be a database because it appears in the output of `SHOW DATABASES`. Second, if the filesystem should fail to mount for some reason, the data directory will still exist, and unless you have solved the init script bugs mentioned earlier, MySQL could start and initialize to a default database. To avoid these problems, create a directory (perhaps named something suggestive such as "mysql") inside the root of the filesystem, and then configure that to be the data directory. Now the database sees a clean and empty directory, and should the filesystem fail to mount, MySQL will abort during startup because of the missing data directory.

11.4 Use a journaling file system

File systems that do not have a journal are very susceptible to corruption and data loss, especially

through unexpected channels such as SAN replication. On Linux, do not use the ext2 file system; use at least ext3, and for best performance, use XFS.

11.5 Reserve space in LVM volume groups

A common default installation in systems such as Red Hat is a volume group that has no space left for snapshots or other maintenance tasks. It is very useful to have some free space in the volume group. Snapshot volumes (for backups and other purposes) require free space, but it is also helpful to have free space to temporarily create a new physical volume for some purposes, such as moving data between physical volumes to enable disk upgrades.

11.6 Reserve space on the filesystem

A full filesystem is a common cause of problems. Worse yet, sometimes a full filesystem cannot be fixed without some free space. For example, freeing disk space might require altering a table, which cannot be done on a completely full filesystem. If there is no way to free some space, you might need to copy files to another system and then back again, or simply delete data. To avoid this, use the `dd` tool to create a file whose only purpose is to reserve space for emergencies. If the emergency occurs, simply delete the file to gain some working space. In many cases as little as a few megabytes could be helpful, but modern disks are so large and cheap that a gigabyte or more is a good idea.

12 Operating System Configuration

A few fundamentals are important to configure for a dedicated database server to avoid serious problems.

12.1 Disable the out-of-memory killer

Linux's out-of-memory killer can choose to kill the MySQL server when the system is stressed by a periodic task such as a cron job, instead of killing the cron job. A database server should give priority to the database, and kill everything else if it interferes. To prevent the MySQL server being chosen as the victim, adjust `/proc/<pid>/oom_adj` to -17.

12.2 Choose a good queue scheduler

The default block device queue scheduler on most enterprise Linux distributions is CFQ, which is not suitable for a server and can cause serious performance problems. Opinions vary on which is best, but either the noop or deadline scheduler is much better than CFQ.

13 Monitoring and Alerting

You should install a monitoring and alerting system to check the server's health and notify you when something is wrong. This is one of the most important components of a robust and resilient system, so you should place a high priority on making it work well.

All of the checks in this section should be considered carefully for your environment. Many of them are inappropriate for certain environments, or must be configured appropriately for the workload to avoid noisiness. However, each of them was or could have been the sole or earliest warning of at least one emergency incident we studied.

13.1 Monitor the monitoring system

Monitoring systems frequently fail to do their job because they or the systems on which they run crash, or cannot communicate with the outside world, or some similar problem. Many emergencies were detected early by a monitoring system that has been mistakenly firewalled off and cannot send email, for example. To avoid this problem, use a separate system to verify end-to-end operation of the monitoring application. This can be as simple as a cron job to send Nagios a passive check, and then an external service such as a hosted monitoring system can alert you if the resulting email does not appear in the designated mailbox.

13.2 Try to eliminate noise

A common mistake is to monitor a large variety of conditions and metrics in the database server. This will result in many false-positive problems. It is very important to reduce the noise. Ideally, when something goes wrong, you should only get one alert

about it; you should not get alerts from several different health checks about the same problem. In particular, you should not monitor things that do not reliably indicate a problem, such as cache hit ratios. If you find yourself tuning thresholds to silence an alert, you should probably ask whether you are doing something wrong.

13.3 Measure effects, not intentions

Many problems happen because a buggy component of a system claims that all is well. For example, MySQL's replication status command `SHOW SLAVE STATUS` can fail to detect that the replica has lost its connection to the master, and therefore believe that it is working and not delayed. A better way to measure whether replication is functioning and up-to-date is to make a change on the master and check for that change on the replica. Whenever possible, you should check for the desired result or correct state in your systems, and avoid checking whether something claims that the result or state exists.

14 MySQL Health Checks

The following suggested health checks can help identify problems inside the MySQL database. Each check is written in the negative, as a problem that should be noticed and solved.

14.1 Inability to query application data

This is the single most important health check for the database server. If you can query the application's data within response-time tolerances, then it is also guaranteed that you can connect to the server, you can log in, you have privileges to the tables, the tables are not corrupt or missing, and the system is not overloaded. Therefore, you do not need separate checks for those things.

This check should be designed carefully. The goal should be to verify that the system as a whole is not malfunctioning badly. The two most important attributes of the check are the results of the query and its response time. The results should indicate that there has been normal system activity recently. For example, if the system is designed to receive sensor data, then you should query for the most recent data

stored in it, and verify that this is within freshness tolerances. At the same time, the query should be written to exercise the application's most critical tables (even if it does not return data from them), to verify that they are operational. The query should not be so trivially simple that it runs quickly when the rest of the system is performing poorly, but it should not be expensive and cause undue load on the system.

The check should mimic production access to the data in as many ways as possible. For example, it should ideally be run from one of the application servers, to verify that the application server is able to connect to the database server. If it runs from the monitoring server, it only verifies that the monitoring server can connect to the application server. It should also log in as the application user, not as a special user for monitoring checks.

14.2 Replication is stopped or delayed

Alert when replication is stopped (either the SQL thread or the relay log thread), or when replication delay exceeds tolerances. Do not rely on `SHOW SLAVE STATUS` for monitoring replication delay, because it is not reliable. Instead, use a tool such as Maatkit's *mk-heartbeat*, which cannot report an artificially low delay. It is a good idea to configure separate alerts for stopped and delayed replication, because replication delays frequently become a noisy alert, which could cause staff to ignore alerts about replication being completely stopped.

14.3 Replicas are not set read-only

Replicas should have the read-only setting enabled, unless they are in a master-master relationship. It is appropriate to alert on a replica that is not also a master.

14.4 Replicas should not write binary log entries

Replicas that do not have the `log_slave_updates` setting enabled should not write any data to their binary logs, if enabled. If they do, it is an indication that something other than replication is changing data on the replica. If `SHOW MASTER LOGS` shows that the most recent binary log's size is too large (the

size of a freshly created binary log varies, but is 106 bytes on MySQL 5.1.41 for example), then there is likely to be a user transaction in the log. The alert can be cleared by flushing the binary logs, which will rotate the log and begin a new one.

14.5 Replication is nearing its capacity

With the additional information exposed in enhanced versions of the MySQL server such as Percona Server or MariaDB, it is possible to inspect the replication process and determine its utilization. If utilization approaches 100%, then it is likely that replication will soon begin to lag the master, at least for some portions of the day. It might be desirable to alert about this.

14.6 There are long-running transactions

Long-running transactions consume system resources, prevent maintenance work from being done, and block other transactions. Do not confuse long-running transactions with slow queries. A long-running transaction can be a series of very fast queries without a `COMMIT`, or a transaction that simply sits idle without committing. Older versions of MySQL expose transactions in the output of `SHOW ENGINE INNODB STATUS`; newer versions expose them through tables in the `INFORMATION_SCHEMA`.

14.7 There are many lock waits

Transactions that are in `LOCK WAIT` status usually indicate a problem in the application, such as faulty logic or spending too long "thinking" between activities. This check might be redundant to a check for long-running transactions, depending on your typical workload characteristics.

14.8 Many queries in the InnoDB queue

The presence of queries in the InnoDB queue indicates potentially severe contention inside the server. This check should be considered in light of your application's typical workload characteristics, because it might be prone to giving false positives.

14.9 Long-lived mutex waits

The presence of a long-lived mutex wait is generally an indication of a severe problem with the database. This could potentially indicate hardware, software, or workload issues. It is possible for it to duplicate other health checks, but sometimes an issue is only indicated by a mutex wait in a background thread in InnoDB, such as background I/O threads.

14.10 Many recent deadlocks

You should configure a tool such as Maatkit's *mk-deadlock-logger* and alert when deadlocks are much more frequent than usual. This check's relevance depends greatly on your application's workload.

14.11 Many recent foreign key errors

This check is similar to deadlocks, and there is also a tool in Maatkit to facilitate it.

14.12 There are unauthenticated users

The presence of threads whose user appears as "unauthenticated user" in the output of `SHOW FULL PROCESSLIST` usually means that login is slow for some reason. The most common reason is that DNS is failing. Sometimes applications that connect and disconnect extremely rapidly will exhibit a certain number of threads in this state on a regular basis, but for most applications, this is either a problem or the beginning of a problem.

14.13 There are locked processes

The presence of many threads in `Locked` status in the output of `SHOW FULL PROCESSLIST` usually points to MyISAM lock contention. On a system that uses many MyISAM tables, this might be a normal occurrence, but it is still not a good thing. If you are not yet migrating to InnoDB, it might be better to start that project than to alert on locked processes. If you already use InnoDB, it probably means that an application is using an explicit `LOCK TABLES` command. If you use MyISAM and do not have locked processes, you should probably monitor for it so that you are alerted when it starts to become a problem.

14.14 Threads are copying to temp tables or file-sorting

If the output of `SHOW FULL PROCESSLIST` shows many threads performing a filesort or copying to a temporary table on disk, then you could be reaching the limits of the system's capacity to do those operations. These tend to be disk-intensive and/or CPU-intensive to the point that there is a sharp "tipping point," and if the load increases much more, the system might not be able to recover.

14.15 The server has restarted recently

Many performance or functionality emergencies have been preceded by a system restart, often without the knowledge of those trying to diagnose the problem. A one-time alert if the server's `Uptime` status variable is below a threshold is a valuable way to make the context of a problem clear. In addition, if you follow the other recommendations in this paper, then you might need to log in to the server and perform some manual checks and startup sequences after an unplanned restart.

14.16 Zero-sized entries in SHOW MASTER LOGS

A zero-sized entry in `SHOW MASTER LOGS` usually means that the binary log index file, which contains a list of the binary logs, does not match what is on disk. This generally happens because binary logs are purged or removed externally to MySQL, instead of using the `PURGE MASTER LOGS` command. In some server versions, a missing binary log will cause automatic log expiration and removal to fail silently, which could fill up the disk.

15 Environmental Health Checks

Problems in the operating environment are a frequent cause of database failures. In addition to the specific checks mentioned below, you should monitor the entire environment, such as network performance.

15.1 A file is deleted but still open

If the *lsof* tool reports that the MySQL server has an open filehandle that is deleted, which is not a temporary file, then there might be a problem. A typical scenario is a deleted slow query log, whether from a broken logrotate script or a person trying to free up disk space. Sometimes a critical data file is deleted, such as the InnoDB system tablespace, and must be re-linked before the server shuts down and closes the filehandle.

15.2 The server's pid file is missing

Tools such as init scripts will fail if the server's pid file, which is usually stored in a location such as `/var/run/mysqld/`, is missing. The file can be deleted because of various mistakes, such as an attempt to start the server while it is already running, which causes the file to be deleted as the second server instance shuts down.

15.3 MySQL lacks privileges to the data directory

Mistakes can cause MySQL to have only read permissions to all or part of the data directory. Sometimes MySQL runs this way for a while, but then the system crashes at unexpected times, such as when a query needs to create a temporary file and cannot. To prevent this, you should alert if any file or directory within the data directory is not owned and writable by the server.

15.4 There is a new entry in the error log

A properly configured server rarely or never writes to the log except during events such as startup, shutdown, or replication initialization. If you are not using a log monitoring system that makes it difficult to miss such an unusual event, then you should configure a check to alert you when there is a new entry in the log, because it should be investigated as soon as possible.

15.5 The system is swapping

Active paging is a serious concern for MySQL, which is not designed to deal with critical parts of

memory actually being on disk. If a thread acquires a lock under the assumption that it will be short-lived, and then has to wait for virtual memory to be paged in, performance will slow to a crawl. You should monitor swap activity.

15.6 There are multiple server instances

Unless you intend to run multiple *mysqld* instances on a single physical server, you should monitor for the presence of more than one instance. Buggy init scripts or human error can cause several instances to start and compete for resources.

15.7 Custom daemons and tasks are not working

Any systems that you install (including the monitoring system) should be monitored. For example, if you decide to install a Maatkit tool to run as a daemon and monitor replication delay, you should alert if the heartbeat on the master is not updated.

16 Storage Health Checks

The storage system and filesystem are such common sources of database problems that they should be monitored closely to detect problems as early as possible.

16.1 The volume is nearly full

The most basic health check is for enough free space on the disk. Alert when either a percentage threshold is exceeded, an absolute minimum of free space is not available, or both. Sometimes a hybrid rule can be a good check for all of the volumes on a system, both large and small.

16.2 An LVM snapshot has failed

Backup systems that create snapshot LVM volumes should release them when the backup finishes. If the system has an error and does not clean up the snapshot volumes, their copy-on-write space can fill, causing the snapshot to fail. The *lvs* command shows snapshot volumes and the percent allocation of copy-on-write space. In addition to alerting when a volume fails, it might be a good idea to alert if

a volume is close to running out of copy-on-write space, because this can provide early warning that backups will soon need larger snapshot volumes. Finally, you can alert if a snapshot exists at an unexpected time. For example, you might alert if the volume exists at noon when the backup should finish at 8:00am.

16.3 A RAID controller is degraded

A surprising number of disks fail and are never noticed until their mirror fails, causing complete data loss. Controllers with battery-backed write caches can also have dead or low batteries. If this happens, the write cache is usually disabled, causing a sudden loss of performance.

RAID controllers have configuration utilities such as *arcconf* or *MegaCli64* that will report disk and volume (physical and virtual device) health, as well as battery health. All of those should be monitored. Some controllers can also output their most recent log entries, which can indicate problems as well.

17 Server Upgrades

When a new version of MySQL is released, review the changelogs between the current version and the new version, and consider upgrading. Upgrades should be tested with a tool such as Maatkit's *mk-upgrade*, which can find problems such as performance regressions, warnings, or result-set differences.

When you upgrade the server, disable any automatic restarts that are built into packages, if you are using a package management system to perform the upgrade. It is better to upgrade the server separately from the restart operation, so that if there is a failure you are not left with a broken server, and so that you can perform a controlled shutdown and restart for minimum downtime.

18 Helper Daemons and Tools

The MySQL server is better when used in combination with external tools to make it easier to administer and more reliable to operate. This section lists tools and processes that can be a beneficial part

of the production database environment. Many of these are partially implemented as parts of toolkits such as Maatkit or open-ark-kit.

The actions taken by such helper tools should be logged, and the monitoring system should alert if there are many within a short time period. The tool should also update a heartbeat in the database so that the monitoring system can ensure it is running.

18.1 Kill rogue queries

MySQL does not have sufficient built-in resource constraints to prevent users from impacting other users. Many environments have relatively few types of queries that all run quickly, and bad queries are easy to spot with an automated tool. Candidates for killing include long-running idle transactions, long-running updates that will cause replicas to fall behind, and long-running queries with a bad EXPLAIN plan.

18.2 Verify replication integrity

MySQL's replication includes no built-in safeguards to verify that replicas have the same data as their masters. The *mk-table-checksum* tool from Maatkit is useful for this.

18.3 Restart replication after certain errors

Some replication errors are safe to resolve by simply restarting replication, and these are not difficult to detect automatically. These include lock wait timeouts, deadlocks, and corrupt relay logs.

18.4 Monitor replication delay

The best way to measure replication delay is with a heartbeat record that is updated on the master and inspected on replicas.

18.5 Log deadlocks and foreign key violations

A record of deadlocks and foreign key violations is helpful for historical purposes as well as for monitoring and alerting.

18.6 Install a graphing and trending system

A graphing and trending system is as important as a monitoring and alerting system. It is helpful to have a system that can record historical metrics about your entire system, not merely the database server. RRDTOol-based systems are popular for this, and good templates exist, especially for Cacti.

18.7 Record response times

Server-wide metrics of response time, throughput, and derivative measures such as the variance of response time are invaluable for capacity planning and troubleshooting. Even though the database server has dozens of status counters, not a single one measures query response time, or even a related metric. External tools such as *tcprstat* are essential for collecting and recording these statistics.²

18.8 Record data sizes

Collecting table sizes on a daily basis helps identify growth patterns and aids in capacity planning.

18.9 Record Queries

Maintaining a database of the types of queries that execute, along with attributes such as their response times, frequency, and execution plans, is a good idea for analytical activities such as investigating changes and scaling patterns.

19 Periodic Tasks

Preventing emergencies is not a one-time effort. For best results, it should probably be a systematic effort that is acknowledged and supported through the organization, even if only part of the organization is involved in carrying it out. It seems useful to group recurring tasks into weekly, monthly, and on-demand time schedules. Your preferences for scheduling tasks might differ from those suggested here; the following sections are only a starting point.

²Enhanced versions of MySQL such as Percona Server expose added statistics about response time.

20 Weekly Tasks

20.1 Restart servers as needed

Check whether any servers are scheduled for restart.

20.2 Check replication

Replication needs regular attention to capture problems and solve the root causes. A weekly routine could be to perform compare data and schema between masters and replicas, check for replication thread utilization approaching capacity, and check to ensure that nothing is changing data on replicas.

20.3 Review changes

Changes tend to happen one at a time in the midst of other activities, and it is easy to lose sight of them and forget how much has changed in aggregate. It is a good idea to review changes on a weekly basis by comparing the current state to the previous week's. Topics to investigate might include privilege changes, schema changes, and changes to configuration files and variables at runtime—the configuration file should match the running server.

In addition, it might be a good idea to rank and compare activity data to the previous week. Types of data to rank might include queries (including reads and writes separately), user activity, table usage, and table size.

20.4 Check for performance early-warning

Performing server profiling with *oprofile* and wait analysis with stack traces at peak load can help to detect the onset of scaling problems before they become severe. In addition, statistical analysis to find and quantify variations in response time and throughput can be valuable ways to detect extremely short stalls in processing, which are often present long before they become severe enough for humans to notice directly. I/O system latency can be analyzed in the same fashion. Finally, a quick review of key utilization, throughput, and response time graphs for the network, CPU, memory, and disk can help reveal problems before they become severe.

20.5 Analyze workload and performance

Most performance problems inside the database are caused by poorly written queries or improperly designed tables and indexes. When these become severe enough to cause downtime, it is often because several unchecked changes have accumulated, which singly would not pose a problem, but together can impact the system greatly. A weekly review of changes to queries and schema is probably a good idea to detect such problems, which might develop even when each individual change is inspected as it is deployed.

These checks might include looking for unused indexes, unused columns, queries with unstable execution plans, queries that do execute optimally, a review of all “new” queries, and automated “lint-checking” of new SQL.

20.6 Perform scalability modeling

Mathematical models of system scalability, such as Neil J. Gunther’s Universal Scalability Law, can help to predict how close the system is to its capacity. The advantage of Dr. Gunther’s model is that the input data is simple and easy to collect, making it potentially feasible to perform on a weekly basis.

20.7 Review activity from utilities

It is useful to review the data collected by helper tools, such as the number and frequency of deadlocks and foreign key errors, automatically killed queries, and automatic replication restarts.

20.8 Validate backups

Ensure that backups are recoverable by testing a recovery. Record the time and system resources needed for the recovery.

20.9 Review logs and monitoring

Review the MySQL error log, and if there are no log entries, test that it works. Review system logs and the output of *dmesg*.

20.10 Analyze query errors

Applications are often bad at handling or logging errors, and it is possible for SQL statements to re-

turn fatal errors such as invalid syntax and never be noticed. A weekly SQL error analysis is important to expose these problems. If you are using Percona Server, you can find error numbers in the enhanced slow-query log; if not, you can capture the errors from the TCP traffic.

20.11 Check for SQL injection

The queries that are sent to the server are not always what the application developers intended. Use an automated tool to find queries that might be SQL injection attempts.

21 Monthly Tasks

Monthly tasks might include the following:

- Review backup policies and procedures
- Review system documentation
- Update all servers’ `/etc/motd`
- Review users and privileges
- Review disk usage and data growth
- Review auto-increment primary keys to avoid reaching their limits
- Review MyISAM tables to ensure they are not reaching their maximum size
- Review the archiving and purging plan
- Clean up the home directory of users such as root
- Clean any old `my.cnf` versions left in the system
- Remove any non-data files in the data directory
- Remove old MySQL-related files from directories such as `/var/log/` and `/var/run/`

Acknowledgments

Thanks to Tom Basil, Espen Braekken, and Morgan Tocker for reviewing this paper.

About Percona

Percona is the oldest and largest independent provider of commercial support, consulting, training, and engineering services for MySQL databases and the LAMP stack. If you would like help with your database servers, we invite you to contact us through our website at <http://www.percona.com/>, or to call us. In the USA, you can reach us during business hours in Pacific (California) Time, toll-free at 1-888-316-9775. Outside the USA, please dial +1-208-473-2904. You can reach us during business hours in the UK at +44-208-133-0309.

About Percona Software

Percona is committed to producing open-source software for Percona Server, MySQL, and MariaDB users. We offer a range of our own software, and also participate actively in many non-Percona software projects. All of our software is open-source and free of charge.

Percona Server is an enhanced version of the world's most popular open-source database, MySQL. MySQL is used by many of the world's largest websites, including Facebook, Flickr, and YouTube. MySQL is also deployed widely in industries such as financial services, government, education, pharmaceuticals, and telecommunications. Its simplicity, reliability, and ease of use make it cost-effective to manage, and because it is open-source, it can be used without license fees. Percona Server is derived from the MySQL database, to which it adds features such as enhanced monitoring and configurability.

Percona XtraDB is an enhanced version of the InnoDB storage engine. Storage engines are a unique feature of the MySQL database architecture. They are the software that stores and retrieves the data, and executes queries at the lowest level. MySQL supports a large variety of storage engines with differing characteristics. The user can choose which storage engine is best suited for each table, based on features such as ACID compliance, full-text indexing, and clustering. InnoDB is the most popular general-purpose OLTP storage engine. It is transactional and ACID compliant, with foreign keys, row-level locking, and an advanced MVCC (multi-version concurrency control) architecture. It is stable and mature, and has been in production use for many years. Percona XtraDB builds on this foundation with improved performance and scalability, and adds a number of useful features.

Percona Server with XtraDB is the combination of Percona Server and the XtraDB storage engine. It includes a companion hot-backup tool, **Percona XtraBackup**, which can also back up standard MySQL and InnoDB data. XtraBackup can make non-blocking backups while the server is running, without interrupting the database's normal operation.

About the Checklists

The checklists that follow the prose portion of this white paper, beginning on the next page, are intended as an aid to tracking and reporting on the tasks discussed in this paper. They are licensed more permissively than the rest of the white paper, so they can be used and improved by others as long as credit is given to Percona.

Percona, XtraDB, and XtraBackup are trademarks of Percona Inc. InnoDB and MySQL are trademarks of Oracle Corp.

Server and Application Review Checklist

Date _____
Technician _____
Company _____
Server _____

System and Application Architecture Review

Results

- Cache stampedes are prevented
- Connection pools are recycled periodically
- Connection pools check for dead connections
- Connection pools restore 'clean' connection state
- Connection pools do not do a 'ping' constantly for short queries
- Failover procedures are tested realistically under load

Safeguards and Security Review

Results

- Login banners such as /etc/motd are used and updated
- External access is restricted at the TCP/IP layer
- Development passwords are different from production
- No unnecessary privileges are granted
- The SUPER privilege is not granted unless needed
- The data directory has restrictive permissions
- Old-style passwords are disabled
- No accounts have old-style passwords
- No accounts lack passwords
- No anonymous privileges are granted

Operating System and Storage Configuration Review

Results

- Out-of-memory killer is disabled
- The queue scheduler is appropriate
- RAID auto-learn cycles are disabled
- Data volumes, mount points, etc are named with the word "mysql"
- The data directory's full path contains the word "mysql"
- The data directory is not in the filesystem root
- The data directory is on a journalled filesystem
- There is free space for snapshots in LVM volumes
- Space is reserved on the filesystem with a filler file

MySQL Installation and Usage Review

Results

- MySQL server is not a debug build
- MySQL server is not a stripped build
- MySQL server is recent within its release series
- MySQL init script does not install a new database
- MySQL init script does not check tables
- MySQL error logs are rotated correctly
- MySQL error logs are kept long-term
- MySQL error logs are stored on separate disk volumes from data
- MySQL error logs are verified as working

- MySQL error logs are separated from system error logs
- MySQL slow query logs are rotated correctly
- Triggers are safe for replication
- Triggers are used only as necessary
- Views are used only as necessary
- Temporary tables are not used with statement-based binary logging
- InnoDB is used unless counter-indicated

MySQL Server Configuration Review

Results

- Configuration file is clean and minimal
- Client tab-completion is disabled in configuration file
- Recommendations from *mk-variable-advisor* have been examined
- InnoDB is the default storage engine
- The server cannot start without InnoDB
- InnoDB buffer pool is sized correctly
- InnoDB log files are sized correctly
- Large packets are permitted
- MySQL starts in `read_only` mode
- MySQL does not start replication automatically
- Binary logs are kept as long as needed for recovery
- Binary logs are rotated automatically with `expire_logs_days`
- Replication configuration parameters are not hard-coded
- Replication is not filtered unless truly needed
- The query cache is disabled unless proven beneficial
- The server does not perform DNS lookups on login
- Master heartbeat is enabled (MySQL 5.5)
- Relay log recovery is enabled (MySQL 5.5)
- Fast warmup is enabled (Percona Server)
- User, table, and index statistics are enabled (Percona Server)
- Query response time distribution is enabled (Percona Server)
- Dedicated purge thread is enabled (Percona Server)
- Corrupt InnoDB tables cause an error, not a server crash (Percona Server)
- Maatkit's UDF functions are installed (Percona Server)
- Query logging is configured to enable added statistics (Percona Server)

Monitoring System Health Checks Review

Results

- The monitoring system is monitored
- The monitoring system is not sending noise alerts
- Monitoring rules measure effects, not intentions
- The monitoring system checks real application queries
- The monitoring system checks for real recent system activity
- The monitoring system runs queries from the application server
- Replication is running
- Replication is not delayed
- Replicas are read-only
- Replicas do not write binary log entries

- There are not too many long-running transactions
- There are not too many lock waits
- There are not too many queries in InnoDB's queue
- There are no long-lived mutex waits
- There are not too many recent deadlocks
- There are not too many recent foreign key errors
- There are not too many unauthenticated users
- There are not too many locked processes
- There are not too many threads sorting or copying to temp tables
- The server has not restarted recently
- There are no zero-sized entries in SHOW MASTER LOGS
- There are no broken file handles (deleted files held open)
- The pid file exists
- The server has complete access to the data directory
- There are no new entries in the error log
- The system is not swapping actively
- There is only one server instance
- Custom daemons and cron tasks are working
- Storage volumes are not filling up
- No LVM snapshots are failed or filling up
- The RAID controller is healthy
- The RAID controller cache battery is healthy

Weekly and Monthly Tasks Checklist

Date _____
 Technician _____
 Company _____
 Server _____

General Items and Log Review

Results

- Servers restarted as scheduled _____
- Replication integrity checked with *mk-table-checksum* _____
- Schema changes reviewed _____
- Privilege changes reviewed _____
- Configuration changes reviewed _____
- MySQL error log contents checked _____
- MySQL error log tested to ensure it is functional _____
- System logs checked _____
- Backup logs checked _____
- dmesg checked _____
- Review of *mk-archiver* activity _____
- Review of *mk-deadlock-logger* activity _____
- Review of *mk-fk-logger* activity _____
- Review of *mk-kill* activity _____
- Review of *mk-slave-restart* activity _____

Performance Reviews / Early Warning Detection

Results

- MySQL stack trace (poor man's profiler) review _____
- MySQL profiling with *oprofile* _____
- MySQL response time variation analysis _____
- MySQL stall detection _____
- I/O latency, utilization, variation, and stall analysis _____
- Network utilization and throughput reviewed _____
- CPU utilization reviewed _____
- Memory utilization reviewed _____
- Disk utilization reviewed _____

Workload and Schema Analysis

Results

- New queries reviewed _____
 - SQL injection analysis _____
 - Query plan _____
 - Index usage _____
 - Correctness of query _____
 - Determinism _____
 - Server warning messages _____
 - Performance _____
- Read-to-write ratio reviewed _____
- User activity reviewed _____
- Table usage reviewed _____
- Table size reviewed _____
- Unused indexes reviewed _____

- Queries causing errors captured and reviewed _____
- Queries with unstable execution plans reviewed _____
- Simple scalability modeling performed _____

Backup and Recovery Testing

Results

- Restore operation tested _____
- Point-in-time recovery tested _____
 - Total time required to restore _____
 - CPU utilization _____
 - Network utilization _____
 - Disk/storage utilization _____
 - Memory utilization _____
 - Disk space required _____
 - Network transfer required _____

Monthly Tasks

- Backup policies reviewed _____
- Backup access and security reviewed _____
- Backup procedures reviewed _____
- System documentation updated _____
- Login banners updated _____
- MySQL users and privileges reviewed _____
- Disk usage and capacity forecasted _____
- Auto-increment primary keys reviewed _____
- MyISAM tables reviewed vs. maximum capacity _____
- Archiving and purging plan reviewed _____
- Clean up root's home directory _____
- Clean up data directory _____
- Clean up /etc/mysql _____
- Clean up /var/run/mysql/ _____
- Clean up /var/log/mysql/ _____
- Clean up configuration files _____