

# Causes of Downtime in Production MySQL Servers

A Percona White Paper

Baron Schwartz

## Abstract

Everyone wants to prevent database downtime by being proactive, but are measures such as inspecting logs and analyzing SQL really proactive? Although these are worthwhile, they almost always identify problems that already exist. To be truly proactive, one must prevent problems, which requires studying and understanding the reasons for downtime.

This paper presents the results of a study of emergency downtime issues that Percona was asked to resolve. It sheds light on what types of problems can occur in production environments. The results might be surprising. Conventional wisdom holds that bad SQL is the leading cause of downtime, but the outcome of this research was quite different.

In this paper, downtime is defined as time that the database server is not providing correct responses with acceptable performance. Downtime could be because a server has just been rebooted and is running on cold caches with increased response times, or data has been destroyed so the server is returning wrong answers, or replication is delayed, or any of a number of other circumstances besides complete unavailability.

This paper explains how we categorized the incidents, presents take-aways about downtime and its causes, and concludes with some recommendations that seem likely to help prevent downtime. Our goal is that MySQL users and support professionals can advance their understanding of what it means to be proactive, increasing the quality of service to end customers.

## Research Method

We drew sample incidents from 200 emergency issues in Percona's support database. We categorized each incident by its nature, causes, resolution, and measures that in our professional judgment could have prevented the problem from happening. Our conclusions are based on reading each issue's e-mails, technician notes, chat transcripts, and when appropriate, terminal session transcripts. Some issues were false-positives or did not have detailed enough notes to categorize and codify them, so the final sample was 154 incidents.

The sample data is biased and might not represent the overall population of MySQL users. There might be a high number of website emergencies compared to the general population. Web application developers and startup companies often work in a less disciplined fashion than, for example, banks or pharmaceutical research companies. (Percona's customer list does include those industries, and the sample includes emergency downtime issues from those types of customers.) Therefore, this study might over-represent mistakes made by people building applications in a hectic startup environment without proper resources or planning. Furthermore, many emergencies are resolved without calling Percona; we are often called only for perplexing problems. For example, although the data shows that SAN failures are a significant cause of storage problems, that could be because SAN failures are difficult to diagnose.

We found that downtime incidents can be grouped into four major categories. Figure 1 shows the categories and how frequently they occurred in the sample.

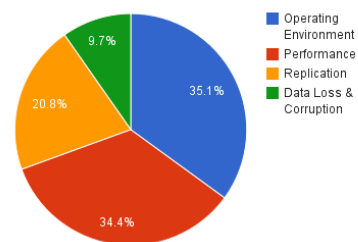


Figure 1: Categories of Downtime

Let's look at each of these categories in a bit more detail.

## The Operating Environment

More than a third of the emergencies were caused by problems in the database's operating environment, such as the storage system, network, or operating system. Table 1 shows the proportion of problems in each sub-category:

Item	Incidents	% of Total
Storage System	24	44.4%
Network	9	16.7%
Operating System	8	14.8%
External Software	5	9.3%
Init Scripts	4	7.4%
Power Loss	2	3.7%

Table 1: Failures in the Operating Environment

Storage systems were the leading component that failed in the operating environment; nearly half of the incidents were due to storage failures. Of the remaining incidents, the most common was network failure, including DNS failure; MySQL is vulnerable to DNS failures in its default configuration. This was followed by operating system configuration, such as kernel bugs. The category of external software includes malfunctioning high-availability systems, connection pools, and cache servers.

There were four instances of system init scripts breaking the database in a variety of ways. For example, one system crashed and had to run recovery upon MySQL startup, but the init script would not wait long enough for it to complete, so it repeatedly killed it and started it again in a loop. Other systems started multiple instances of the MySQL server, or did not shut the server down cleanly. One control panel uninstalled MySQL and reverted it back to a buggy and poorly performing earlier version automatically, but the downgrade was unclean, and the system entered a loop of crashing and restarting.

Storage failures are so common that they are worth drilling into. Table 2 is a breakdown of storage failure incidents.

Item	Incidents	% of Total
Disk Full	9	37.5%
SAN	5	20.8%
RAID Controller	4	16.7%
Other Storage	3	12.5%
Filesystem	2	8.3%
Permissions	1	4.2%

Table 2: Storage System Failures

SAN failures tended to be harder to diagnose than RAID controller failures; in one instance a SAN control panel showed nothing wrong with the system even though it had degraded performance due to a failed disk. Disk failures themselves are not included in this table because unless multiple disks fail, which did not occur in this sample of incidents, a functioning SAN or RAID controller does not fail completely. The "Other Storage" category includes NFS servers and Amazon EBS volumes.

## Performance Problems

Performance problems of various sorts caused 53 downtime incidents, nearly as many as problems in the operating environment. This is probably no surprise to readers, and neither is the leading cause of performance problems: badly optimized SQL, followed by badly designed schema. Table 3 shows the relative proportions:

Item	Incidents	% of Total
SQL	20	37.8%
Schema and Indexing	8	15.1%
InnoDB	8	15.1%
Configuration	7	13.2%
Idle Transactions	4	7.6%
Other	4	7.6%
Query Cache	2	3.8%

Table 3: Causes of Bad Performance

In most cases, the InnoDB problems were caused by old versions of MySQL, and could be solved by upgrading the server. Some of them, however, required custom patches to the server.

Most configuration problems were caused by gross user error such as complete lack of configuration, and fine-tuning the server was not a solution for any of the incidents we looked at. For example, one powerful server with dozens of gigabytes of memory was used for a large InnoDB database—but the InnoDB buffer pool was unconfigured and defaulted to 8MB. Another frequent error was failing to set the InnoDB log file size, so the server was running with the default 10MB of redo logs instead of hundreds of megabytes.

Idle transactions—transactions that had modified data and then never committed—caused problems

by blocking other transactions, which timed out and failed. The query cache problems were all solved by simply disabling the query cache; it is not a scalable component of the MySQL server. The problems in the “other” category included server bugs, running a broken server build, badly designed backups, and long-running ALTER TABLE statements.

## Replication Problems

Replication is a key strategy for users to scale their applications, provide redundancy, take backups, run reporting queries, and many other uses. Unfortunately, MySQL’s built-in replication is somewhat fragile. It is not hard to configure and use, but it is also not hard to break, and there were 32 incidents of replication downtime. Table 4 shows the types of replication problems:

Item	Incidents	% of Total
Data Differences	14	43.8%
Configuration	6	18.8%
Other	5	15.6%
Delays	4	12.5%
Bugs	4	12.5%

Table 4: Replication Problems

The leading cause of replication problems is replication stopping due to the master and replica datasets diverging. The usual culprit is a careless user or misbehaving application that connects to the replica and modifies data there, causing an update from the master to fail. The second leading cause of failures is a configuration problem, such as the infamous `max_packet_allowed` configuration variable not being set large enough.

Replication delays do not cause replication to stop, but the data on the replica can become so stale that it is unusable, and read-write splitting systems such as the MMM replication manager will remove the replica from the pool of read-only servers, effectively causing downtime for that replica. Most replication delays are actually performance problems at root, but they are often caused by bugs in replication, such as a bug causing the replica not to use an index for an update.

Another portion of bugs can cause replication to fail entirely. Most of these occur in very old server ver-

sions, and are easy to solve by upgrading to a sensible server version. A surprising number of users install MySQL from old installation CDs, such as the default version of MySQL 5.0.22 that shipped with a popular enterprise Linux distribution for a number of years. This version is very buggy and was obsoleted almost immediately—but what is on the CD is what gets installed in many cases.

The catch-all “Other” category includes badly executed upgrades, user errors, and attempts to run replication in brittle configurations such as a “ring” topology.

## Data Loss and Corruption

The sample contained 15 cases of data loss or corruption. These are usually caused by a mistake or hardware failure. Rarely is arbitrary corruption found in the database; it is generally introduced from the outside, and InnoDB detects it and refuses to operate. In most cases, lack of recent restorable backups is the only reason the problem becomes an emergency. Table 5 shows the causes of data loss and corruption in the cases we studied:

Item	Incidents	% of Total
DROP TABLE	9	60.0%
Data Corruption	3	20.0%
Server Bug	2	13.3%
Malicious Attack	1	6.7%

Table 5: Causes of Data Loss and Corruption

This data confirmed our belief that it is more common for users to destroy their own data than for a system component to fail. The practice of using a replica as a backup does not protect against this type of problem.

## Root Cause Analysis

Root cause analysis is something of a mythical beast. It sounds simple and easy to do, but there are always multiple “root” causes, and if you follow the trail far enough, you will end up at the CEO of the organization most of the time. Still, we were able to make some generalizations, and assigned blame for the incidents to five broad categories: failures

in change control, configuration, capacity planning, backups, and monitoring. Most of the issues could have been prevented by several different measures. For example, lack of disk space could have been detected before it became an emergency with a monitoring and alerting system, or it could have been predicted by capacity planning, or prevented by fixing a buggy logrotate script. Configuration errors could have been detected early by performance analysis required for capacity planning, or prevented by doing the configuration properly in the first place, or detected by a monitoring system. And although most data loss could have been prevented by backups, having backups doesn't prevent the problem that leads to the data loss in the first place. This plurality of root causes makes it hard to say which are the most prevalent.

In our opinion, though, lack of change control is easily the most common cause of emergencies. By change control we mean that when someone changes the database server or application, they should test and verify that nothing bad happens. Similarly, many changes to the server and application should be made, but are not, and that causes problems. Here are two representative examples:

**Upgrading carelessly.** Server upgrades should be tested before they are performed in production environments. Many of the issues were caused by changes in behavior with new server versions. These included queries no longer using indexes, differences in query results, bugs, and instability.

**Not upgrading.** New versions of the server are released for a reason. The longer an old version of the server runs, the more likely it is that someone will trigger one of its many bugs. For example, many replication problems in 5.0 server versions are caused by bugs that have been fixed for years—but nobody upgraded the server to avoid them.

Notice the contradiction: without the resources and time to upgrade properly, users can cause problems both by upgrading and by not upgrading. Poor change control might be a large cause of problems in part because it is difficult to do well.

We group several other types of problems under the heading of change control. For example, SQL queries that perform badly do not usually get past good pre-production testing. Without such testing, which is a form of change control, bad SQL in production is inevitable.

## Preventing Emergencies

In most cases, the 154 emergencies we analyzed could have been prevented best by a systematic, organization-wide effort to find and remove latent problems. Many of the activities involved in this effort could appear to be unproductive, and might be unrewarding for people to do.

Along with the findings presented earlier in this paper, we created a list of detailed preventive measures for every issue we investigated. This is published separately as a white paper on Percona's web site.

## Conclusion

Our study of a subset of emergency issues for Percona's customers confirmed what we believed, while revealing patterns we had not noticed before. For example, we have long believed that many organizations over-invest in database tuning, but do not dedicate enough continuing resources for backups, monitoring, and gathering historical performance data for analysis and capacity planning. This research showed that database tuning is seldom the resolution or prevention for emergencies, but these "second quadrant" activities surely are.

At the same time, we had never questioned the old saw that bad SQL is the leading cause of downtime in databases. The numbers tell a different story: bad SQL is responsible for only 20 of the issues we saw, and even if we group bad SQL and improper schema design together, it is still only 28 issues out of 154. Nearly twice as many issues arose in the operating environment.

Another eye-opening moment came when we realized how much trouble is caused by poor-quality external tools, such as init scripts, tuning tools, and high-availability systems. It is ironic but true that high-availability tools can cause downtime. Solving problems in third-party software will require a significant engineering effort in most cases.

It can be difficult to convince organizations to focus on tasks such as testing deployments and upgrades more carefully. For those who are interested in preventing downtime in their MySQL instances, we hope that this paper, along with the concrete steps in the companion white paper on Percona's website, will prove useful.

## About Percona, Inc.

Percona is the oldest and largest independent MySQL services company, with an international team of over 40 MySQL experts providing 24x7 support, consulting, training, and engineering services for MySQL to over 1000 customers since 2006. Customers include Cisco Systems, Alcatel-Lucent, Groupon, the BBC, and StumbleUpon.

Percona's founders are world-famous for their expertise in MySQL performance and scaling. They are the authors of *High Performance MySQL, 2nd Edition*. Percona also develops software for MySQL users, including Percona Server and Percona XtraBackup. Percona Server is an enhanced version of the MySQL database server. With Percona Server, users achieve faster and more consistent query execution, better insight into server behavior, and the ability to consolidate servers with commodity hardware and flash storage. Percona XtraBackup is the world's only open-source hot backup utility for MySQL's default transactional storage engine InnoDB, and is praised by social networking giant Facebook. Both Percona Server and Percona XtraBackup are free and open-source.

If you are interested in Percona's products or services, 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.

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

This white paper originally appeared as an article in IOUG's *SELECT* magazine, Q1 2011.