PERCONA

www.percona.com

Percona Monitoring Plugins Documentation

Release 1.1.8

Feb 05, 2018

CONTENTS

The Percona Monitoring Plugins are high-quality components to add enterprise-grade MySQL capabilities to your existing in-house, on-premises monitoring solutions. The components are designed to integrate seamlessly with widely deployed solutions such as Nagios, Cacti and Zabbix, and are delivered in the form of templates, plugins, and scripts.

At Percona, our experience helping customers with emergencies informs our monitoring strategies. We have analyzed a large database of emergency issues, and used that to determine the best conditions to monitor. You can read about our suggested approaches to monitoring in our white papers.

Monitoring generally takes two forms:

- Fault detection.

  Fault detection notifies you when systems become unhealthy or unavailable. In general, fault detection monitoring tends to fail because of false alarms, which cause personnel to ignore the alerts or not notice when the monitoring system itself fails. As a result, it is very important to choose very carefully when you monitor for faults: monitor only on actionable conditions that are not prone to false positives, and definitely indicate a problem, but do not duplicate other information or tell you something you already know. The classic example of a poor-quality check is a cache hit ratio, or a threshold such as the number of sort merges per second.

- Metrics collection and graphing.

  By contrast to fault detection, it is a good idea to collect and store as much performance and status information about the systems as possible, for as long as possible, and to have a means of visualizing it as graphs or charts. These are good to glance at periodically, but they are really most useful when you are trying to diagnose a condition whose existence you have already identified. For example, if you see a period of degraded service on one chart, you might look at other charts to try to determine what changed during that period.

In summary, you should alert as much as you need, no more no less, and prefer fewer alerts on broader conditions. You should never ignore an alert. But you should collect as many metrics as possible, and ignore most of them until you need them.

We make our monitoring components freely available under the GNU GPL. If you would like help setting up the components, integrating them into your environment, choosing alerts, or any other task, Percona consulting and support staff can help.

You can download the Percona Monitoring Plugins from the Percona Software Downloads directory, including our Apt and Yum repositories. For specific installation instructions, read the detailed documentation on each type of components below.

CHAPTER

ONE

PLUGINS FOR NAGIOS

Nagios is the most widely-used open-source fault-detection system, with advanced features such as escalation, dependencies, and flexible notification rules.

Percona Monitoring Plugins for Nagios

Many of the freely available Nagios plugins for MySQL are poor quality, with no formal testing and without good documentation. A more serious problem, however, is that they are not created by experts in MySQL monitoring, so they tend to cause false alarms and noise, and don't encourage good practices to monitor what matters.

These plugins offer the following improvements:

- Created by MySQL experts.
- Good documentation.
- Support for the newest versions of MySQL and InnoDB.
- Integration with other Percona software, such as Percona Server and Percona Toolkit.
- Easy to install and configure.
- Real software engineering! There is a test suite, to keep the code high quality.

The plugins are designed to be executed locally or via NRPE. Most large installations should probably use NRPE for security and scalability.

In general, the plugins either examine the local UNIX system and execute commands, or they connect to MySQL via the `mysql` commandline executable and retrieve information. Some plugins combine these actions. Each plugin's documentation explains its commandline options and arguments, as well as the commands executed and the privileges required.

System Requirements

The plugins are all written in standard Unix shell script. They should run on any Unix or Unix-like operating system, such as GNU/Linux, Solaris, or FreeBSD.

The plugins are designed to be used with MySQL 5.0 and newer versions, but they may work on 4.1 or older versions as well.

Installation Instructions

You can download the tarball from the Percona Software Downloads directory or install the package from Percona Software Repositories:

```
yum install percona-nagios-plugins
```

or:

```
apt-get install percona-nagios-plugins
```

Configuration Best Practices

These plugins can be used locally or via NRPE. NRPE is the suggested configuration. Some plugins execute commands that require privileges, so you may need to specify a command prefix to execute them with `sudo`.

For security reasons, it is recommended to not pass MySQL access credentials in the arguments. You can create /etc/nagios/mysql.cnf and the plugins will use it like the default .my.cnf file. For example:

```
[root@centos6 ~]# cat /etc/nagios/mysql.cnf
[client]
user = root
password = s3cret
[root@centos6 ~]# chown root:nagios /etc/nagios/mysql.cnf
[root@centos6 ~]# chmod 640 /etc/nagios/mysql.cnf
```

Also with MySQL 5.6 client you can use the login-path instead of the password or .my.cnf, /etc/nagios/mysql.cnf files. For this, you need to specify `-L` option, e.g. `-L safelogin`. The actual .mylogin.cnf file should be created with `mysql_config_editor` tool and in case of Nagios, placed into nagios user home directory.

Here you can find an excerpt of potential Nagios config `click here`.

And here is an excerpt of related NRPE config:

```
command[rdba_unix_memory]=/usr/lib64/nagios/plugins/pmp-check-unix-memory -d -w 96 -c␣
↪98
command[rdba_mysql_pidfile]=/usr/lib64/nagios/plugins/pmp-check-mysql-pidfile
```

Support Options

If you have questions, comments, or need help with the plugins, there are several options to consider.

You can get self-service help via Percona's forums, or the Percona mailing list.

You can report bugs and submit patches to the Launchpad project.

If you need help with installation, troubleshooting, configuration, selecting services to monitor, deciding on appropriate thresholds, writing more plugins, extending or modifying existing plugins, or fixing bugs in plugins, you may wish to consider a MySQL Support Contract from Percona. These monitoring plugins are fully supported under all Percona contracts.

List of Plugins

pmp-check-aws-rds.py

pmp-check-aws-rds.py - Check Amazon RDS metrics.

SYNOPSIS

```
Usage: pmp-check-aws-rds.py [options]
```

```
Options:
  -h, --help            show this help message and exit
  -l, --list            list of all DB instances
  -n PROFILE, --profile-name=PROFILE
                        AWS profile from ~/.boto or /etc/boto.cfg. Default:
                        None, fallbacks to "[Credentials]".
  -r REGION, --region=REGION
                        AWS region. Default: us-east-1. If set to "all", we
                        try to detect the instance region across all of them,
                        note this will be slower than you specify the region.
  -i IDENT, --ident=IDENT
                        DB instance identifier
  -p, --print           print status and other details for a given DB instance
  -m METRIC, --metric=METRIC
                        metric to check: [status, load, storage, memory]
  -w WARN, --warn=WARN  warning threshold
  -c CRIT, --crit=CRIT  critical threshold
  -u UNIT, --unit=UNIT  unit of thresholds for "storage" and "memory" metrics:
                        [percent, GB]. Default: percent
  -t TIME, --time=TIME  time period in minutes to query. Default: 5
  -a AVG, --avg=AVG     time average in minutes to request. Default: 1
  -f, --forceunknown    force alerts on unknown status. This prevents issues
                        related to AWS Cloudwatch throttling limits Default:
                        False
  -d, --debug           enable debug output
```

REQUIREMENTS

This plugin is written on Python and utilizes the module `boto` (Python interface to Amazon Web Services) to get various RDS metrics from CloudWatch and compare them against the thresholds.

- Install the package: `yum install python-boto` or `apt-get install python-boto`
- Create a config /etc/boto.cfg or ~nagios/.boto with your AWS API credentials. See http://code.google.com/p/boto/wiki/BotoConfig

This plugin that is supposed to be run by Nagios, i.e. under `nagios` user, should have permissions to read the config /etc/boto.cfg or ~nagios/.boto.

Example:

```
[root@centos6 ~]# cat /etc/boto.cfg
[Credentials]
aws_access_key_id = THISISATESTKEY
aws_secret_access_key = thisisatestawssecretaccesskey
```

If you do not use this config with other tools such as our Cacti script, you can secure this file the following way:

```
[root@centos6 ~]# chown nagios /etc/boto.cfg
[root@centos6 ~]# chmod 600 /etc/boto.cfg
```

DESCRIPTION

The plugin provides 4 checks and some options to list and print RDS details:

- RDS Status
- RDS Load Average
- RDS Free Storage
- RDS Free Memory

To get the list of all RDS instances under AWS account:

```
# ./pmp-check-aws-rds.py -l
```

To get the detailed status of RDS instance identified as `blackbox`:

```
# ./pmp-check-aws-rds.py -i blackbox -p
```

Nagios check for the overall status. Useful if you want to set the rest of the checks dependent from this one:

```
# ./pmp-check-aws-rds.py -i blackbox -m status
OK mysql 5.1.63. Status: available
```

Nagios check for CPU utilization, specify thresholds as percentage of 1-min., 5-min., 15-min. average accordingly:

```
# ./pmp-check-aws-rds.py -i blackbox -m load -w 90,85,80 -c 98,95,90
OK Load average: 18.36%, 18.51%, 15.95% | load1=18.36;90.0;98.0;0;100 load5=18.51;85.
↪0;95.0;0;100 load15=15.95;80.0;90.0;0;100
```

Nagios check for the free memory, specify thresholds as percentage:

```
# ./pmp-check-aws-rds.py -i blackbox -m memory -w 5 -c 2
OK Free memory: 5.90 GB (9%) of 68 GB | free_memory=8.68;5.0;2.0;0;100
# ./pmp-check-aws-rds.py -i blackbox -m memory -u GB -w 4 -c 2
OK Free memory: 5.90 GB (9%) of 68 GB | free_memory=5.9;4.0;2.0;0;68
```

Nagios check for the free storage space, specify thresholds as percentage or GB:

```
# ./pmp-check-aws-rds.py -i blackbox -m storage -w 10 -c 5
OK Free storage: 162.55 GB (33%) of 500.0 GB | free_storage=32.51;10.0;5.0;0;100
# ./pmp-check-aws-rds.py -i blackbox -m storage -u GB -w 10 -c 5
OK Free storage: 162.55 GB (33%) of 500.0 GB | free_storage=162.55;10.0;5.0;0;500.0
```

By default, the region is set to `us-east-1`. You can re-define it globally in boto config or specify with -r option. The following command will list all instances across all regions under your AWS account:

```
# ./pmp-check-aws-rds.py -r all -l
```

The following command will show the status for the first instance identified as `blackbox` in all regions:

```
# ./pmp-check-aws-rds.py -r all -i blackbox -p
```

Remember, scanning regions are slower operation than specifying it explicitly.

CONFIGURATION

Here is the excerpt of potential Nagios config:

```
define host{
      use                             mysql-host
      host_name                       blackbox
      alias                           blackbox
      address                         blackbox.abcdefgh.us-east-1.rds.amazonaws.com
      }
```

```
define servicedependency{
      host_name                       blackbox
      service_description             RDS Status
      dependent_service_description   RDS Load Average, RDS Free Storage, RDS Free␣
↪Memory
      execution_failure_criteria      w,c,u,p
      notification_failure_criteria   w,c,u,p
      }
```

```
define service{
      use                             active-service
      host_name                       blackbox
      service_description             RDS Status
      check_command                   check_rds!status!0!0
      }
```

```
define service{
      use                             active-service
      host_name                       blackbox
      service_description             RDS Load Average
      check_command                   check_rds!load!90,85,80!98,95,90
      }
```

```
define service{
      use                             active-service
      host_name                       blackbox
      service_description             RDS Free Storage
      check_command                   check_rds!storage!10!5
      }
```

```
define service{
      use                             active-service
      host_name                       blackbox
      service_description             RDS Free Memory
      check_command                   check_rds!memory!5!2
      }
```

```
define command{
      command_name    check_rds
```

```
      command_line    $USER1$/pmp-check-aws-rds.py -i $HOSTALIAS$ -m $ARG1$ -w $ARG2$␣
↪-c $ARG3$
      }
```

VERSION

Percona Monitoring Plugins pmp-check-aws-rds.py 1.1.8

pmp-check-lvm-snapshots

pmp-check-lvm-snapshots - Alert when LVM snapshots are running out of copy-on-write space.

SYNOPSIS

```
Usage: pmp-check-lvm-snapshots [OPTIONS]
Options:
  -c CRIT     Critical threshold; default 95%.
  -w WARN     Warning threshold; default 90%.
  --help      Print help and exit.
  --version   Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin looks at the output of the 'lvs' command to find LVM snapshot volumes that are beginning to run out of copy-on-write space. If a snapshot fills up its copy-on-write space, it will fail. This is also a useful way to detect whether some process, such as a backup, failed to release a snapshot volume after finishing with it.

PRIVILEGES

This plugin does not access MySQL.

This plugin executes the following UNIX commands that may need special privileges:

- lvs

VERSION

Percona Monitoring Plugins pmp-check-lvm-snapshots 1.1.8

pmp-check-mongo.py

pmp-check-mongo.py

“”“

pmp-check-mysql-deadlocks

pmp-check-mysql-deadlocks - Alert when pt-deadlock-logger has recorded too many recent deadlocks.

SYNOPSIS

```
Usage: pmp-check-mysql-deadlocks [OPTIONS]
Options:
  -c CRIT          Critical threshold; default 60.
  --defaults-file FILE Only read mysql options from the given file.
                   Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST          MySQL hostname.
  -i INTERVAL      Interval over which to count, in minutes; default 1.
  -l USER          MySQL username.
  -L LOGIN-PATH    Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS          MySQL password.
  -P PORT          MySQL port.
  -S SOCKET        MySQL socket file.
  -T TABLE         The database.table that pt-deadlock-logger uses; default percona.
↪deadlocks.
  -w WARN          Warning threshold; default 12.
  --help           Print help and exit.
  --version        Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin looks at the table that pt-deadlock-logger (part of Percona Toolkit) maintains, and when there have been too many recent deadlocks, it alerts.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SELECT` from the `pt-deadlock-logger` table.

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-mysql-deadlocks 1.1.8

pmp-check-mysql-deleted-files

pmp-check-mysql-deleted-files - Alert when MySQL's files are deleted.

SYNOPSIS

```
Usage: pmp-check-mysql-deleted-files [OPTIONS]
Options:
  -c CRIT         Critical threshold; ignored.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST         MySQL hostname.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -w WARN         Warning threshold; changes the alert to WARN instead of CRIT.
  --help          Print help and exit.
  --version       Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin looks at the files that the mysqld process has open, and warns if any of them are deleted that shouldn't be. This typically happens when there is a poorly written logrotate script or when a human makes a mistake at the command line. This can cause several bad effects. If a table has been deleted, of course, it is a serious matter. Such a file can also potentially fill up the disk invisibly. If the file is the server's log, it might mean that logging is effectively broken and any problems the server experiences could be undiagnosable.

The plugin accepts the -w and -c options for compatibility with standard Nagios plugin conventions, but they are not based on a threshold. Instead, the plugin raises a critical alert by default, and if the -w option is given, it raises a warning instead, regardless of the option's value.

This plugin doesn't alert about deleted temporary files, which are not a problem. By default, this plugin assumes that the server's temporary directory is either the TMPDIR environment variable, or if that is not set, then /tmp/. If you specify MySQL authentication options, the value will log into the specified MySQL instance and look at the `tmpdir` variable to find the temporary directory.

This plugin looks at the first running instance of MySQL, as found in the system process table, so it will not work on systems that have multiple instances running. It probably works best on Linux, though it might work on other operating systems. It relies on either lsof or fstat or the ability to list the files in the process's /proc/pid/fd directory.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SELECT` the system variable `@@tmpdir`.

This plugin executes the following UNIX commands that may need special privileges:

- ps
- `lsof` or `ls /proc/$pid/fd` (Linux), `fstat` (BSD)

The plugin should be able to find mysqld PID using `ps` command.

On BSD, if `sysctl` option `security.bsd.see_other_uids` is set to 0, `ps` will not return mysqld PID if the plugin run from non-root user.

VERSION

Percona Monitoring Plugins pmp-check-mysql-deleted-files 1.1.8

pmp-check-mysql-file-privs

pmp-check-mysql-file-privs - Alert if MySQL data directory privileges are wrong.

SYNOPSIS

```
Usage: pmp-check-mysql-file-privs [OPTIONS]
Options:
  -c CRIT         Critical threshold; makes a privilege issue critical.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -g GROUP        The Unix group who should own the files; default mysql.
  -H HOST         MySQL hostname.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -u USER         The Unix user who should own the files; default mysql.
  -w WARN         Warning threshold; ignored.
  --help          Print help and exit.
  --version       Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin checks to make sure that the MySQL data directory, and its contents, is owned by the correct Unix user and group. If the ownership is incorrect, then the server might fail due to lack of permission to modify its data. For example, suppose a system administrator enters a database directory and creates a file that is owned by root. Now a database administrator issues a DROP TABLE command, which fails because it is unable to remove the file and thus the non-empty directory cannot be removed either.

The plugin accepts the -g and -u options to specify which Unix user and group should own the data directory and its contents. This is usually the user account under which MySQL runs, which is mysql by default on most systems. The plugin assumes that user and group by default, too.

The plugin accepts the -w and -c options for compatibility with standard Nagios plugin conventions, but they are not based on a threshold. Instead, the plugin raises a warning by default, and if the -c option is given, it raises an error instead, regardless of the option's value.

By default, this plugin will attempt to detect all running instances of MySQL, and verify the data directory ownership for each one. It does this purely by examining the Unix process table with the `ps` tool. However, in some cases the process's command line does not list the path to the data directory. If the tool fails to detect the MySQL server process, or if you wish to limit the check to a single instance in the event that there are multiple instances on a single server, then you can specify MySQL authentication options. This will cause the plugin to skip examining the Unix processlist, log into MySQL, and examine the datadir variable from SHOW VARIABLES to find the location of the data directory.

In case an user you are calling this plugin from has no permissions to examine the datadir the plugin raises an unknown with the explanation.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SELECT` the MySQL system variables `@@datadir` and `@@basedir`.

This plugin executes the following UNIX commands that may need special privileges:

- ps
- find `datadir`

The plugin should be able to either get variables from MySQL or find mysqld PID using `ps` command.

On BSD, if `sysctl` option `security.bsd.see_other_uids` is set to 0, `ps` will not return mysqld PID if the plugin run from non-root user.

Also an user you run the plugin from should be able to access MySQL datadir files, so you may want to add it into mysql unix group etc.

VERSION

Percona Monitoring Plugins pmp-check-mysql-file-privs 1.1.8

pmp-check-mysql-innodb

pmp-check-mysql-innodb - Alert on problems inside InnoDB.

SYNOPSIS

```
Usage: pmp-check-mysql-innodb [OPTIONS]
Options:
  -C CHECK        What to alert on; default idle_blocker_duration.
                  Other options: waiter_count, max_duration.
  -c CRIT         Critical threshold; default varies.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST         MySQL hostname.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -w WARN         Warning threshold; default varies.
  --help          Print help and exit.
  --version       Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin alerts on various aspects of InnoDB status in several ways, depending on the value of the -C option:

idle_blocker_duration

This is the default behavior. It alerts when a long-running transaction is blocking another, and the blocker is idle (Sleep). The threshold is based on how long the transaction has been idle. Long-running idle transactions that have acquired locks but not released them are a frequent cause of application downtime due to lock wait timeouts and rollbacks, especially because applications are often not designed to handle such errors correctly. The problem is usually due to another error that causes a transaction not to be committed, such as performing very long tasks in the application while holding the transaction open.

This check examines the INFORMATION_SCHEMA tables included with InnoDB version 1.0 and newer. The default critical level is 600, and warning is 60. If the tables do not exist, the exit status is OK, with a note that the tables do not exist.

waiter_count

Alerts if too many transactions are in LOCK WAIT status. Uses information from SHOW ENGINE INNODB STATUS if the INFORMATION_SCHEMA tables are not available. The default critical level is 25, and warning is 10.

max_duration

Alerts if any transaction is too old. Uses information from SHOW ENGINE INNODB STATUS if the INFORMATION_SCHEMA tables are not available. The default critical level is 600, and warning is 60.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SHOW ENGINE INNODB STATUS`.
- `SELECT` against the `INFORMATION_SCHEMA` InnoDB transaction and lock tables.

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-mysql-innodb 1.1.8

pmp-check-mysql-pidfile

pmp-check-mysql-pidfile - Alert when the mysqld PID file is missing.

SYNOPSIS

```
Usage: pmp-check-mysql-pidfile [OPTIONS]
Options:
  -c CRIT         Critical threshold; makes a missing PID file critical.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST         MySQL hostname.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -w WARN         Warning threshold; ignored.
  --help          Print help and exit.
  --version       Print version and exit.
```

```
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin checks to make sure that the MySQL PID file is not missing. The PID file contains the process ID of the MySQL server process, and is used by init scripts to start and stop the server. If it is deleted for some reason, then it is likely that the init script will not work correctly. The file can be deleted by poorly written scripts, an accident, or a mistaken attempt to restart MySQL while it is already running, especially if mysqld is executed directly instead of using the init script.

The plugin accepts the -w and -c options for compatibility with standard Nagios plugin conventions, but they are not based on a threshold. Instead, the plugin raises a warning by default, and if the -c option is given, it raises an error instead, regardless of the option.

By default, this plugin will attempt to detect all running instances of MySQL, and verify the PID file's existence for each one. It does this purely by examining the Unix process table with the `ps` tool. However, in some cases the process's command line does not list the path to the PID file. If the tool fails to detect the MySQL server process, or if you wish to limit the check to a single instance in the event that there are multiple instances on a single server, then you can specify MySQL authentication options. This will cause the plugin to skip examining the Unix processlist, log into MySQL, and examine the pid_file variable from SHOW VARIABLES to find the location of the PID file.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SELECT` the system variables `@@pid_file` and `@@basedir`.

This plugin executes the following UNIX commands that may need special privileges:

- ps

The plugin should be able to either get variables from MySQL or find mysqld PID using `ps` command.

On BSD, if `sysctl` option `security.bsd.see_other_uids` is set to 0, `ps` will not return mysqld PID if the plugin run from non-root user.

Also an user you run the plugin from should be able to access MySQL PID file file, so you may want to add it into mysql unix group etc.

VERSION

Percona Monitoring Plugins pmp-check-mysql-pidfile 1.1.8

pmp-check-mysql-processlist

pmp-check-mysql-processlist - Alert when MySQL processlist has dangerous patterns.

SYNOPSIS

```
Usage: pmp-check-mysql-processlist [OPTIONS]
Options:
  -C CHECK        What to alert on; default states_count.
                  Other options: max_user_conn.
  -c CRIT         Critical threshold; default varies.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST         MySQL hostname.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -w WARN         Warning threshold; default varies.
  --help          Print help and exit.
  --version       Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin examines MySQL processlist in several ways, depending on the value of the -C option:

states_count

Alerts when there are too many processes in various states. The list of checks is as follows:

Unauthenticated users appear when DNS resolution is slow, and can be a warning sign of DNS performance problems that could cause a sudden denial of service to the server.

Locked processes are the signature of MyISAM tables, but can also appear for other reasons.

Too many processes copying to various kinds of temporary tables at one time is a typical symptom of a storm of poorly optimized queries.

Too many processes in the "statistics" state is a signature of InnoDB concurrency problems causing query execution plan generation to take too long.

The thresholds should be given as count. The default critical level is 32, and warning is 16.

max_user_conn

Alerts when `@@max_user_connections` is configured on MySQL and any user reaches this limit. The output of this check will display the user with maximum connections consumed, its count and percentage of the actual limit.

The thresholds should be given as percentage. The default critical level is 95, and warning is 90.

Examples:

```
# /usr/lib64/nagios/plugins/pmp-check-mysql-processlist
OK 0 unauthenticated, 0 locked, 0 copy to table, 0 statistics | processes=0;16;32;0;
```

```
# /usr/lib64/nagios/plugins/pmp-check-mysql-processlist -C max_user_conn
OK User with max connections: myappuser (70) = 2% | max_user_conn=2;90;95;0;100
```

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SHOW PROCESSLIST;`
- `SELECT @@max_user_connections;`

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-mysql-processlist 1.1.8

pmp-check-mysql-replication-delay

pmp-check-mysql-replication-delay - Alert when MySQL replication becomes delayed.

SYNOPSIS

```
Usage: pmp-check-mysql-replication-delay [OPTIONS]
Options:
  -c CRIT          Critical threshold; default 600.
  --defaults-file FILE Only read mysql options from the given file.
                   Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST          MySQL hostname.
  -l USER          MySQL username.
  -L LOGIN-PATH    Use login-path to access MySQL (with MySQL client 5.6).
  -m CRIT          Minimal threshold to ensure for delayed slaves; default 0.
  -p PASS          MySQL password.
  -P PORT          MySQL port.
  -S SOCKET        MySQL socket file.
  -s SERVERID      MySQL server ID of master, if using pt-heartbeat table. If
                   the parameter is set to "MASTER" the plugin will lookup the
                   server_id of the master
  -T TABLE         Heartbeat table used by pt-heartbeat.
  -u               Use UTC time to count the delay in case pt-heartbeat is run
                   with --utc option.
  -w WARN          Warning threshold; default 300.
  --master-conn NAME  Master connection name for MariaDB multi-source replication.
  --channel NAME  Master channel name for multi-source replication (MySQL 5.7.6+).
  --unconfigured  Alert when replica is not configured at all; default no.
  --ensure-sbm    Disallow Seconds_Behind_Master to be NULL for delayed slaves when -
↪m is used
  --help           Print help and exit.
  --version        Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin examines whether MySQL replication is delayed too much. By default it uses SHOW SLAVE STATUS, but the output of the Seconds_behind_master column from this command is unreliable, so it is better to use pt-heartbeat from Percona Toolkit instead. Use the -T option to specify which table pt-heartbeat updates. Use the -s

option to specify the master's server_id to compare against; otherwise the plugin reports the maximum delay from any server. Use the -s options with the value "MASTER" to have plugin lookup the master's server_id

If you want to run this check against the delayed slaves, e.g. those running with pt-slave-delay tool, you may want to use -m option specifying the minimal delay that should be ongoing, otherwise the plugin will alert critical.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SHOW SLAVE STATUS [NONBLOCKING|NOLOCK]`

  or

- `SELECT` from the `pt-heartbeat` table.

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-mysql-replication-delay 1.1.8

pmp-check-mysql-replication-running

pmp-check-mysql-replication-running - Alert when MySQL replication stops.

SYNOPSIS

```
Usage: pmp-check-mysql-replication-running [OPTIONS]
Options:
  -c CRIT         Report CRITICAL when replication is stopped with or w/o errors.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -d              Useful for slaves delayed by pt-slave-delay. It will not alert
                  when IO thread is running, SQL one is not and no errors.
  -H HOST         MySQL hostname.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -w WARN         Report WARNING when SHOW SLAVE STATUS output is empty.
  --master-conn NAME  Master connection name for MariaDB multi-source replication.
  --channel NAME  Master channel name for multi-source replication (MySQL 5.7.6+).
  --help          Print help and exit.
  --version       Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin examines whether replication is running. It is separate from the check for delay because it is confusing or impossible to handle all of the combinations of replication errors and delays correctly, and provide an appropriate type of alert, in a single program.

By default, this plugin treats it as critical when the either thread stops with an error, and a warning when threads are stopped with no error. You can provide critical and warning thresholds with the -c and -w options, for compatibility with Nagios plugin conventions, but they don't work as thresholds. Instead, if you specify a critical threshold, this plugin will treat it as critical if either thread is stopped, with or without an error.

The warning threshold makes the plugin report a warning when SHOW SLAVE STATUS produces no output, which means it is not configured as a replica. By default, this plugin will report that replication is healthy when a server isn't configured as a replica.

If you want to run this check against the delayed slaves, e.g. those running with pt-slave-delay tool, you may want to specify -d option. It will not alert when Slave_IO_Running is Yes, Slave_SQL_Running is No and there are no errors.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SHOW SLAVE STATUS [NONBLOCKING|NOLOCK]`

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-mysql-replication-running 1.1.8

pmp-check-mysql-status

pmp-check-mysql-status - Check MySQL SHOW GLOBAL STATUS output.

SYNOPSIS

```
Usage: pmp-check-mysql-status [OPTIONS]
Options:
  -c CRIT         Critical threshold.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -C COMPARE      Comparison operator to apply to -c and -w.
                  Possible values: == != >= > < <=. Default >=.
  -H HOST         MySQL hostname.
  -I INCR         Make SHOW STATUS incremental over this delay.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -o OPERATOR     The operator to apply to -x and -y.
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -T TRANS        Transformation to apply before comparing to -c and -w.
                  Possible values: pct str.
  -w WARN         Warning threshold.
  -x VAR1         Required first status or configuration variable.
  -y VAR2         Optional second status or configuration variable.
  --help          Print help and exit.
  --version       Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin captures SHOW GLOBAL STATUS and SHOW GLOBAL VARIABLES from MySQL and evaluates expressions against them. The general syntax is as follows:

```
VAR1 [ OPERATOR VAR2 [ TRANSFORM ] ]
```

The result of evaluating this is compared against the -w and -c options as usual to determine whether to raise a warning or critical alert.

Note that all of the examples provided below are simply for illustrative purposes and are not supposed to be recommendations for what to monitor. You should get advice from a professional if you are not sure what you should be monitoring.

For our first example, we will raise a warning if Threads_running is 20 or over, and a critical alert if it is 40 or over:

```
-x Threads_running -w 20 -c 40
```

The threshold is implemented as greater-or-equals by default, not strictly greater-than, so a value of 20 is a warning and a value of 40 is critical. You can switch this to less-or-equals or other operators with the -C option, which accepts the arithmetic comparison operators ==, !=, >, >=, <, and <=.

You can use any variable that is present in SHOW VARIABLES or SHOW STATUS. If the variable is not found, there is an error. To warn if Threads_connected exceeds 80% of max_connections:

```
-x Threads_connected -o / -y max_connections -T pct -w 80
```

The -T `pct` option only works when you specify both -x and -y and implements percentage transformation. The plugin uses awk to do its computations and comparisons, so you can use floating-point math; you are not restricted to integers for comparisons. Floating-point numbers are printed with six digits of precision. The -o option accepts the arithmetic operators /, *, +, and -. A division by zero results in zero, not an error.

If you specify the -I option with an integer argument, the SHOW STATUS values become incremental instead of absolute. The argument is used as a delay in seconds, and instead of capturing a single sample of SHOW STATUS and using it for computations, the plugin captures two samples at the specified interval and subtracts the second from the first. This lets you evaluate expressions over a range of time. For example, to warn when there are 10 disk-based temporary tables per second, over a 5-second sampling period:

```
-x Created_tmp_disk_tables -o / -y Uptime -I 5 -w 10
```

That is somewhat contrived, because it could also be written as follows:

```
-x Created_tmp_disk_tables -I 5 -w 50
```

The -I option has the side effect of removing any non-numeric SHOW STATUS variables. Be careful not to set the -I option too large, or Nagios will simply time the plugin out, usually after about 10 seconds.

This plugin does not support arbitrarily complex expressions, such as computing the query cache hit ratio and alerting if it is less than some percentage. If you are trying to do that, you might be doing it wrong. A dubious example for the query cache might be to alert if the hit-to-insert ratio falls below 2:1, as follows:

```
-x Qcache_hits -o / -y Qcache_inserts -C '<' -w 2
```

Some people might suggest that the following is a more useful alert for the query cache:

```
-x query_cache_size -c 1
```

To check Percona XtraDB Cluster node status you may want to use the following alert similar to what its clustercheck does:

```
-x wsrep_local_state -C '!=' -w 4
```

To compare string variables use -T `str` transformation. This is required as numeric and string comparisons are handled differently. The following example warns when the slave_exec_mode is IDEMPOTENT:

```
-x slave_exec_mode -C '==' -T str -w IDEMPOTENT
```

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SHOW STATUS`.
- `SHOW VARIABLES`.

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-mysql-status 1.1.8

pmp-check-mysql-ts-count

pmp-check-mysql-ts-count - Generic alert based on pmp-check-mysql-deadlocks to count number of rows written in the last interval.

SYNOPSIS

```
Usage: pmp-check-mysql-ts-count [OPTIONS]
Options:
  -c CRIT         Critical threshold; default 60.
  --defaults-file FILE Only read mysql options from the given file.
                  Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST         MySQL hostname.
  -i INTERVAL     Interval over which to count, in minutes; default 1.
  -l USER         MySQL username.
  -L LOGIN-PATH   Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS         MySQL password.
  -P PORT         MySQL port.
  -S SOCKET       MySQL socket file.
  -t TIMESTAMP    The name of the timestamp column to be monitored; default ts.
  -T TABLE        The database.table to be monitored; default percona.deadlocks.
  -w WARN         Warning threshold; default 12.
  -x TARGET       Metric monitored; default deadlocks.
                  Other options: kills, fkerrors.
  --help          Print help and exit.
  --version       Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin looks at a table and counts the number of rows since the last interval, and alerts if this exceeds the threshold. This could be the table referenced by pt-deadlock-logger, pt-kill, pt-fk-error-logger, or a custom table supplied. Default behaviour is the same as pmp-check-mysql-deadlocks, can also specify target to be `kills` or `fkerrors` to monitor default tables created by pt-kill or pt-fk-error-logger respectively, or supply custom metric and table.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SELECT` from the supplied table.

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-mysql-ts-count 1.1.8

pmp-check-pt-table-checksum

pmp-check-pt-table-checksum - Alert when pt-table-checksum finds data differences on a replica.

SYNOPSIS

```
Usage: pmp-check-pt-table-checksum [OPTIONS]
Options:
  -c CRIT          Raise a critical error instead of a warning.
  --defaults-file FILE Only read mysql options from the given file.
                   Defaults to /etc/nagios/mysql.cnf if it exists.
  -H HOST          MySQL hostname.
  -l USER          MySQL username.
  -L LOGIN-PATH    Use login-path to access MySQL (with MySQL client 5.6).
  -p PASS          MySQL password.
  -P PORT          MySQL port.
  -S SOCKET        MySQL socket file.
  -i INTERVAL      Interval over which to ensure pt-table-checksum was run,
                   in days; default - not to check.
  -T TABLE         The checksum table; default percona.checksums
  -w WARN          Warning threshold; ignored.
  --help           Print help and exit.
  --version        Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin examines whether MySQL replication has drifted out of sync with the master's data, according to checks performed by the pt-table-checksum tool in Percona Toolkit. It uses the following query to determine whether the server's data matches its master's:

```
SELECT /\* MAGIC_checksum_diff_query \*/
CONCAT(
   COUNT(\*),
   ' chunks differ in ',
   COUNT(DISTINCT CONCAT(db, tbl)),
   ' tables, including ',
   MIN(CONCAT(db, '.', tbl)))
FROM CHECKSUM_TABLE
WHERE master_cnt <> this_cnt OR master_crc <> this_crc
   OR ISNULL(master_crc) <> ISNULL(this_crc)
HAVING COUNT(\*) > 0
```

The word CHECKSUM_TABLE is replaced by the value of the -T option. If the table specified by -T does not exist, unknown is raised.

Optionally, you can specify an interval in days over which to ensure pt-table-checksum was run. It is useful in cases when the cron job doing the checksumming suddenly stopped working. This option will have an effect when no diffs are found and the checksum table is not empty.

Alerts are raised at a WARNING level by default, but specifying the -c option with any value will change this to CRITICAL instead.

PRIVILEGES

This plugin executes the following commands against MySQL:

- `SELECT` against the specified table.

This plugin executes no UNIX commands that may need special privileges.

VERSION

Percona Monitoring Plugins pmp-check-pt-table-checksum 1.1.8

pmp-check-unix-memory

pmp-check-unix-memory - Alert on low memory.

SYNOPSIS

```
Usage: pmp-check-unix-memory [OPTIONS]
Options:
  -c CRIT     Critical threshold; default 95%.
  -w WARN     Warning threshold; default 90%.
  --help      Print help and exit.
  --version   Print version and exit.
Options must be given as --option value, not --option=value or -Ovalue.
Use perldoc to read embedded documentation with more details.
```

DESCRIPTION

This Nagios plugin examines `/proc/meminfo` (Linux) or the output of `sysctl` (BSD) to check whether the system is running out of memory and finds the largest process in memory from `ps` output.

The plugin is tested on GNU/Linux and FreeBSD.

PRIVILEGES

This plugin does not access MySQL.

This plugin executes the following UNIX commands that may need special privileges:

- cat /proc/meminfo (Linux), sysctl (BSD)
- ps

VERSION

Percona Monitoring Plugins pmp-check-unix-memory 1.1.8

CHAPTER

TWO

TEMPLATES FOR CACTI

Cacti is a popular PHP- and MySQL-based web front-end to RRDTool, providing intuitive point-and-click configuration and browsing of graphs and metrics.

Percona Monitoring Plugins for Cacti

Many Cacti templates you'll find online are often poor quality and have many problems. Cacti's design can also cause inefficiency if you don't know how to use it correctly, and most templates don't avoid those inefficiencies. The Percona's Cacti templates alleviate these problems, which are common in other templates:

- No duplicated data in RRD files.
- No unused data in RRD files.
- No wasted polling for the same data, which can cause timeouts and increased load

These templates offer the following improvements:

- Versioning and backwards compatibility.
- Good documentation.
- Much more data is collected and graphed than you'll typically find.
- Graphs have attractive colors and consistent formatting. Metrics are printed as well as graphed, so you can read the numbers as well as look at the picture.
- Support for the newest versions of MySQL and InnoDB.
- Integration with other Percona software, such as Percona Server and Percona Toolkit.
- Easy to install and configure.
- Easy and safe to upgrade, with support for a configuration file so you can upgrade the scripts without losing your configuration.
- Debugging features to help find and solve problems.
- Templates don't conflict with your existing Cacti installation; they don't use anything pre-defined in Cacti. That means you can import them without fear of overwriting your customized settings.
- Real software engineering! There is a test suite, to keep the code high quality.

In addition, the software supports a much easier way to generate graphs and templates than you will find elsewhere, so you can create your own custom graphs of anything you desire. If you need help with this, Percona can also assist with that.

Frequently Asked Questions on Cacti Templates

My graphs have NaN on them. What is wrong?

> Please read and follow the directions at http://www.cacti.net/downloads/docs/html/debugging.html.

My Cacti error log has many lines like "WARNING: Result from CMD not valid. Partial Result: U" What's wrong?

> Please read and follow the directions at http://www.cacti.net/downloads/docs/html/debugging.html.

Running `poller.php` or the PHP script shows the correct output, but I see nothing in the graphs in Cacti. Why?

> If you ran either of those commands before Cacti did, then the cache file that the PHP script uses may have the wrong ownership. Check that the Cacti user has access to this cache file. Also check the ownership on the PHP script itself; does the Cacti user have permission to execute it? Sometimes the problem is due to installing or running as root from the command line, not realizing that Cacti runs as a different user.

The output is truncated when I use Spine. Why?

> This is a Cacti bug. Use cmd.php for now.

I get a blank page when I try to import the templates. Why?

> Check your webserver log. PHP might have run out of memory. If you see something like "Allowed memory size of 8388608 bytes exhausted (tried to allocate 10 bytes)" then you should increase `memory_limit` in your `php.ini` file and restart the webserver. Check that you've changed the correct `php.ini` file. If the problem persists, you might be changing the wrong one or doing it wrong. Some users have reported that they need to add `ini_set('memory_limit', '64M');` to the top of include/global.php.

When I try to import the templates, my browser asks me if I want to download templates_import.php. Why?

> This might be the same problem as the blank page just mentioned. Check your web server's error logs.

If I have a host that is both an Apache and MySQL server, should I be creating two separate devices in cacti, each with the appropriate Host Template?

> You don't need to. You can simply make it a MySQL server, create the appropriate graphs, and then switch to an Apache server and create the appropriate graphs.

How do I graph a MySQL server that uses a non-standard port?

> See the documentation on customizing the templates.

Installing Percona Monitoring Plugins for Cacti

This page explains how to install and use the pre-built templates that ship with this project. If the templates are not exactly what you need, see the documentation on creating custom templates.

The following instructions assume you have the necessary privileges to make changes to your Cacti server. You probably need to become root on that server.

It is a good idea to make sure your Web browser doesn't save your password when you're using Cacti. There are some screens in the administrative interface that have hidden password fields you can't see, but the browser will fill in, and it'll cause the templates to be configured incorrectly.

Downloading

You can download the tarball from the Percona Software Downloads directory or install the package from Percona Software Repositories.

Installing from tarball

You will need to get a copy of the scripts on the server, and you need access to the templates locally, so you can import them through your web browser. Therefore, you should probably download the monitoring plugins to your own computer, and then upload them to your Cacti server as well.

Let's assume you have just downloaded version 1.0.0, in percona-monitoring-plugins-1.0.0.tar.gz. Unpack the archive with the following command at a command prompt:

```
root@cactiserver# tar zxf percona-monitoring-plugins-1.0.0.tar.gz
```

Repeat the same process on your local computer. You should now have a directory containing several files. (The directory name will change with each new release). Change into this directory:

```
root@cactiserver# cd percona-monitoring-plugins-1.0.0/cacti/
```

Before you install, read the specific instructions for the templates you plan to install. These are linked from this document's table of contents.

The general process is to copy the data-gathering scripts into place, and then to import the templates via the web interface.

Copy the PHP scripts into your Cacti installation's scripts directory, on the server that hosts Cacti. This is usually `/usr/share/cacti/site/scripts/`, but might be different on your system. Let's assume you want to install the MySQL templates:

```
root@cactiserver# cp scripts/ss_get_mysql_stats.php /usr/share/cacti/site/scripts
```

Now import the template files through your web browser. In the Cacti web interface's Console tab, click on the *Import Templates* link in the left sidebar. Browse to the directory containing the unpacked templates, select the XML file for the templates you're installing, and submit the form. In our example, the file will be named something like `cacti_host_template_percona_mysql_server_ht_0.8.6i-sver1.0.0.xml`.

Inspect the page that results. You should see something like the following:

```
Cacti has imported the following items:

CDEF
[success] Percona Negate CDEF [new]

GPRINT Preset
[success] Percona MySQL Server Version t1.0.0:s1.0.0 [new]
[success] Percona Normal [new]

Data Input Method
[success] Percona Get MySQL Stats/MyISAM Indexes IM [new]
... snip ...

Data Template
[success] Percona MyISAM Indexes DT [new]
... snip ...
```

```
Graph Template
[success] Percona Indexes GT [new]
... snip ...

Host Template
[success] Percona Server HT [new]
```

The above is an abbreviated list. Every line should show "success" at the beginning, and "new" (or "update" if you're upgrading) at the end.

Installing from package

To install scripts and templates you can run:

```
yum install percona-cacti-templates
```

or:

```
apt-get install percona-cacti-templates
```

Now you have to import templates using Cacti web interface as described in the tarball installation above (do not need to copy any scripts but requires the local access to the templates from a tarball) or simply import templates from the command line, e.g.:

```
php /usr/share/cacti/cli/import_template.php --filename=/usr/share/cacti/resource/
↪percona/templates/cacti_host_template_percona_gnu_linux_server_ht_0.8.6i-sver1.0.3.
↪xml \
--with-user-rras='1:2:3:4'
```

Configuring

The templates themselves don't need to be configured, but you might need to configure the scripts that they execute to gather their data. For example, you might need to specify a username and password to connect to MySQL and gather statistics. There are several ways to do this.

Embedding Configuration

The simplest way is to embed the configuration options in the script file itself. Open the script file (such as `scripts/ss_get_mysql_stats.php`) with your favorite text editor, and look for a section like the following:

```
# ============================================================================
# CONFIGURATION
# ============================================================================
# Define MySQL connection constants in config.php.  Arguments explicitly passed
# in from Cacti will override these.  However, if you leave them blank in Cacti
# and set them here, you can make life easier.  Instead of defining parameters
# here, you can define them in another file named the same as this file, with a
# .cnf extension.
# ============================================================================
$mysql_user = 'cactiuser';
$mysql_pass = 'cactiuser';
$mysql_port = 3306;
... [snip]...
```

Each PHP file has its own configuration options, and there should be comments that explain them. In the above example, the options are MySQL connection options. Change them as desired, and save and close the PHP file.

This method this has some disadvantages. If you upgrade the PHP script file, you'll lose your configuration. And this only works if all of your monitored resources need the same configuration parameters.

A Configuration File

If you don't want to store the configuration options directly into the PHP script file or you want to preserve your settings after the package update, you can create another file with the same name and the filename extension `.cnf`. Place this under `/etc/cacti/` and ensure it is valid PHP. This file will be included by the PHP script file, so you can define the same configuration options there that you might define in the PHP script file. For example, you might create `/etc/cacti/ss_get_mysql_stats.php.cnf` with the following contents:

```
<?php
$mysql_user = "root";
$mysql_pass = "s3cret";
```

Notice the opening PHP tag, but the absence of a closing PHP tag. This is to comply with PHP standards and avoid problems. Be careful not to add any extra lines or whitespace at the beginning or end of the configuration file, because that can cause whitespace to be included in the script's output.

This method still has the disadvantage that it works only if you use the same global configuration for every monitored resource. If you need to specify a username and password for each host or each graph, it won't work.

A MySQL user should be configured with *the proper privileges*.

Securing Your Setup

You can also place `.cnf` file in the same directory as the PHP script file (just to keep the backward compatibility) but this is a security risk as `scripts/` folder falls under the web directory. So `/etc/cacti/` is the recommended location for `.cnf` file. In any case, ensure that any files under `scripts/` are not accessible from Web. Check out *Hardening Cacti setup* guide.

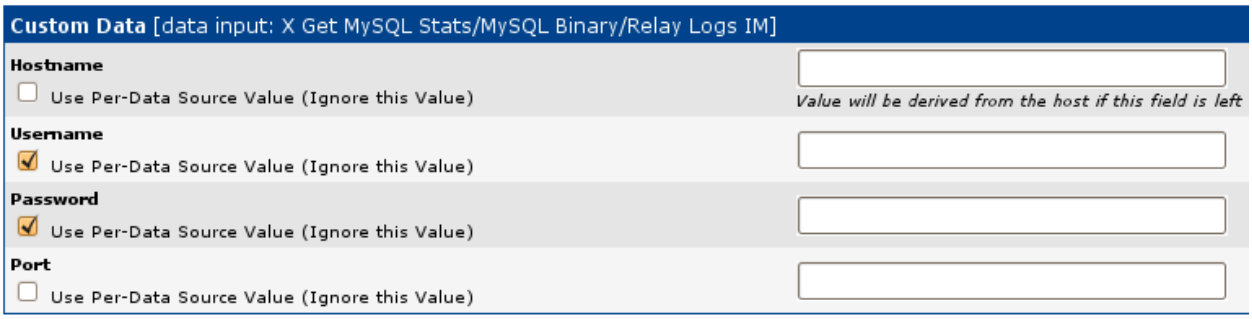
Passing Command-Line Arguments

The above configuration methods make configuration available to the scripts as PHP variables, but it is also possible to pass command-line arguments to the scripts. If you execute the script without any options, you'll see the available options. For example:

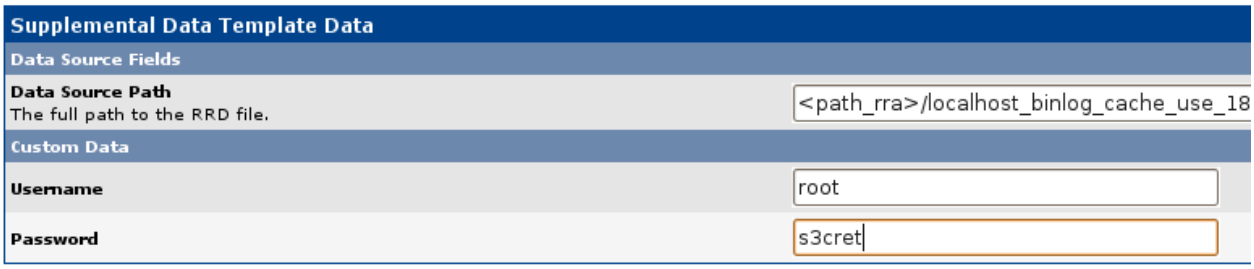
```
# php ss_get_mysql_stats.php
Required option --host is missing
Usage: php ss_get_mysql_stats.php --host <host> --items <item,...> [OPTION]

   --host      MySQL host
   --items     Comma-separated list of the items whose data you want
   --user      MySQL username
   --pass      MySQL password
   --port      MySQL port
   --server-id Server id to associate with a heartbeat if heartbeat usage is enabled
   --nocache   Do not cache results in a file
   --help      Show usage
```

You can make Cacti pass configuration options to the script with these command-line options when it executes the script. To do this, you will need to do one of two things. You can customize specific graphs that require configuration options, or you can generate your own templates so every graph requires you to fill in values for the options.

Here's how to make specific graphs accept command-line arguments. From the Console tab, click into Data Templates. Find the desired Data Template and click it so you can edit it. We will use 'Percona MySQL Binary/Relay Logs DT' as an example. Now, check the checkboxes so the desired command-line options use per-data-source values. This means that the global template's value doesn't override the individual graph's values; the individual graphs must specify their own values. For example, the following figure shows how to set the checkboxes so that username and password are per-data-source:



Next find the data source by clicking into Data Sources. Now that you've specified that this data source should use per-data-source values for the username and password, there are text boxes to fill in:



Cacti will now pass the given arguments to the PHP script when it executes. Here's a snippet from the Cacti log, showing this in action:

```
10/26/2009 03:00:09 PM - CMDPHP: Poller[0] Host[1] DS[18] CMD:
   /usr/bin/php -q /usr/share/cacti/site/scripts/ss_get_mysql_stats.php
   --host 127.0.0.1 --items kx,ky --user root --pass s3cret --port 3306
```

Creating Graphs

Creating graphs is the easiest step of the process.

- In Cacti's Console tab, browse to the "Devices" link in the sidebar and click on the device you'd like to graph.
- The third item from the top of the screen should say *Host Template*. Change this to the name of the template you imported, such as "Percona MySQL Server HT."
- Scroll to the bottom of the page and click the Save button.
- After the page loads, click on the "Create Graphs for this Host" link at the top of the page.
- Tick the checkbox at the top right of the list of graph templates. This should select every graph template that applies to this host but doesn't exist yet.
- Scroll to the bottom of the page and click the Create button.

If you're upgrading from an earlier version of the template, you might need to change the Host Template to None, submit the change, and then change it back to the desired template after the page reloads.

After you create the graphs, wait until the poller runs once, and then check to make sure your new graphs render as images.

Customizing Percona Monitoring Plugins for Cacti

The templates that are included in the release packages are generic, designed to be suitable for default installations. However, if they don't meet your needs, you can generate your own easily.

It is important to note that these templates are designed to avoid the problems caused by modifying templates within Cacti and then exporting them. Instead of doing this, you should use the provided command-line tools to modify the templates before you import them. If you want to, you can modify them over and over again and keep re-importing them. Cacti will update its database to match the changes that you import.

You can customize many aspects of the templates. The following sections will explain the possible customizations. All of these are possible simply by passing the correct options to the `pmp-cacti-template` command-line tool.

Generating Templates

The process of generating a template is very simple. You simply execute the `pmp-cacti-template` program and give it the associated script and template definition file. For example, to create MySQL templates identical to the ones in the release file:

```
$ pmp-cacti-template --script scripts/ss_get_mysql_stats.php \
  definitions/mysql.def > mysql-template.xml
```

Generate Templates for a Specific Cacti Version

Cacti templates are version-specific, because the hash identifiers that are used as GUIDs have a version number embedded in them. This can prevent templates exported from one version of Cacti from being imported to another, even if there is no real incompatibility. The `--cactiver` option to the `pmp-cacti-template` script will control this.

The version numbers it understands are embedded in the program, and if you specify an illegal value, it'll let you know. The versions are forwards-compatible, so templates generated for an earlier version of Cacti should work on a newer version too.

Accept Input in Each Data Source

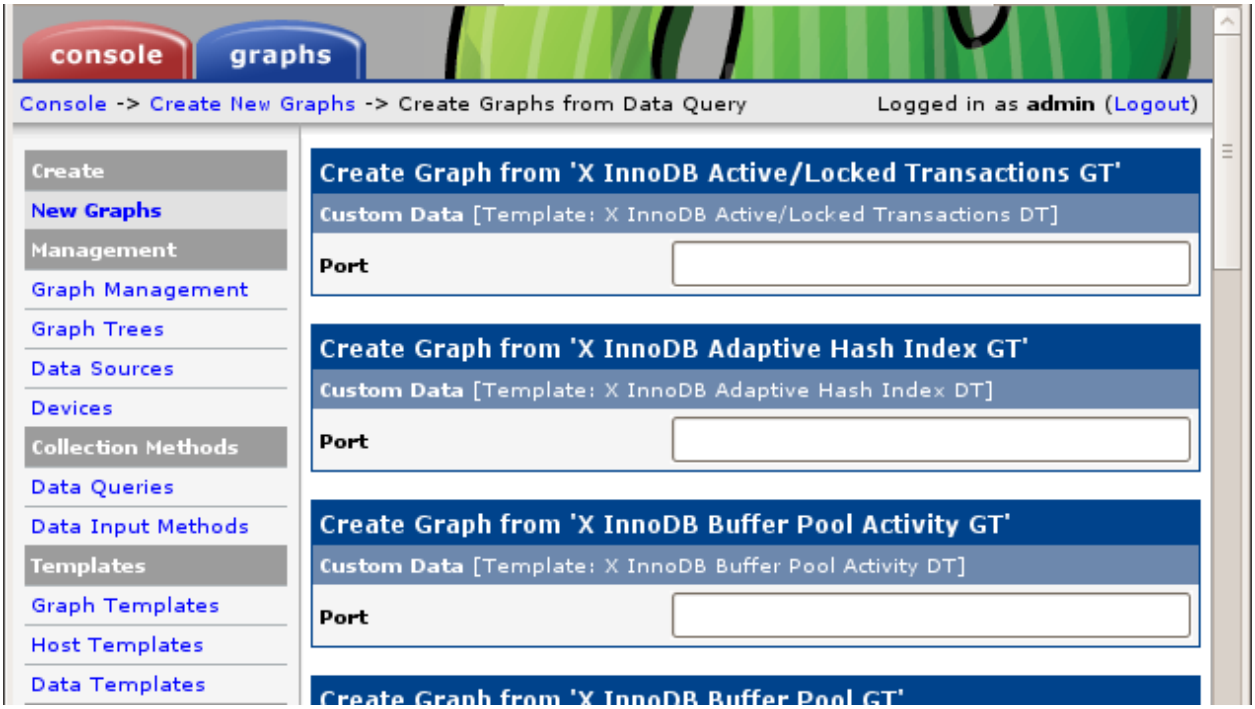
If you want to specify command-line options to data sources, you can easily make certain command-line options for the script required per-graph. For example, let's suppose that you want to ensure the `ss_get_mysql_stats.php` script is executed with the `--port` command-line option.

You can generate templates that require this. The option to use is `--mpds`, which is short for "make per data source". You give it a comma-separated list of options. Here's an example with `--port`:

```
pmp-cacti-template \
  --script scripts/ss_get_mysql_stats.php definitions/mysql.def \
  --mpds port > templates_requiring_port.xml
```

Be aware that the SSH-based templates use `--port` to specify the SSH port, and `--port2` to connect to the resource you're graphing, so you might need to customize `--port2`, not `--port`.

If you import the resulting XML file, and then edit a host to bind it to the "Percona MySQL Server HT" host template, when you create the graphs you'll be prompted to fill in a value for the port:



Specifying a value for a port

Specify a Different Graph Width or Height

The default size of the Cacti graphs is 120 pixels high by 500 pixels wide. If you would like to specify a different size, you can use the `--graph_height` and `--graph_width` options. For example:

```
$ pmp-cacti-template \
  --script scripts/ss_get_mysql_stats.php definitions/mysql.def \
  --graph_height 240 --graph_width 1000 > templates_240x1000.xml
```

Specify a Different Name Prefix

The default naming convention for every item created by the templates starts with "Percona", the name of the item, and an abbreviation at the end, such as "DT" for Data Templates. This makes all the items sort together, and makes them distinctive so you don't confuse them with others that might be named similarly. If you want to specify a different prefix, you can use the `--name_prefix` option. For example, you might specify "Big" for templates that you want to generate at a larger size, as in the previous example. Then you'll have templates named like "Big MySQL Select Types GT".

Specify a Different Polling Interval

The default polling interval for most Cacti installations is 5 minutes, or 300 seconds. The templates need to match the polling interval, because the RRD files are created for a specific polling interval. If you have configured Cacti to use

a different interval, you can generate matching templates with the `--poll_interval` option, which accepts the number of seconds.

Change the Default Maximum Permitted Value in RRD Files

The default maximum value that the RRD files will recognize as valid is used to detect garbage input and prevent spikes in graphs. By default, it is set to the size of a 64-bit unsigned integer. If you want rollover and out-of-bounds detection for 32-bit integer values, use the `--smallint` option.

Percona MySQL Monitoring Template for Cacti

This page gives installation instructions specific to the MySQL graph templates, shows examples of graphs in the MySQL template collection, and shows what they do. You might want to look at http://dev.mysql.com/doc/refman/5.1/en/server-status-variables.html to learn the meaning of the status variables contained in the graphs.

Installation Notes

The MySQL templates work by executing a PHP script that gathers information from MySQL servers and returns it to Cacti. The script makes an ordinary MySQL connection to gather its input.

It is highly recommended that you use the same MySQL username and password for all servers you want to graph, to ease the installation and configuration. If you don't, you will need to customize your templates to accommodate your installation. See below for detailed information on the privileges.

The script requires that you be able to connect to MySQL from your Cacti server. You can test this with the `mysql` command-line program. Debugging MySQL connection problems is beyond the scope of this documentation; refer to the MySQL manual if you have trouble.

To install,

- Create a MySQL user with the SUPER and PROCESS privileges on each server you want to monitor. Assuming you use "cacti" and "s3cret" as your username and password, execute the following command on each server: `GRANT SUPER, PROCESS ON *.* TO 'cacti'@'%' IDENTIFIED BY "s3cret";`
- If you want to monitor replication with `pt-heartbeat` from Percona Toolkit (recommended), you must grant SELECT on the heartbeat table also. Assuming the `pt-heartbeat` table is `percona.heartbeat`, execute `GRANT SELECT ON percona.heartbeat TO 'cacti'@'%';`
- Copy ss_get_mysql_stats.php into your Cacti installation's scripts/ directory.
- All other steps are the same as mentioned in in installation document.

If you want to specify a different MySQL port for various servers, see the instructions on how to accept input in each data source.

User Privileges

The suggested user privileges mentioned above are sufficient for the common case. In some cases you might not want or have such access. The following list explains the queries that the data-gathering script executes, the functionality, and how to disable if it's unwanted:

`SHOW /*!50002 GLOBAL */ STATUS` This query requires no special privileges and is necessary for core functionality.

`SHOW VARIABLES` This query requires no special privileges and is necessary for core functionality.

SHOW ENGINES This query requires no special privileges and is necessary for core functionality.

SHOW SLAVE STATUS This query requires either SUPER or REPLICATION CLIENT. It is necessary for tracking replication lag on replication slaves, which is enabled by default. To disable, edit the `$chk_options` array in the configuration. Alternatively, use `pt-heartbeat` and grant SELECT on the heartbeat table. If disabled, parts of the the MySQL Replication Status and MySQL Binary/Relay logs graphs will be empty. Also the script leverages lock-free `SHOW SLAVE STATUS [NONBLOCKING|NOLOCK]` if available.

SHOW MASTER LOGS This query is used to count up the size of binary logs. It requires the SUPER privilege. If disabled in the `$chk_options` array, then part of the MySQL Binary/Relay logs graph will be empty.

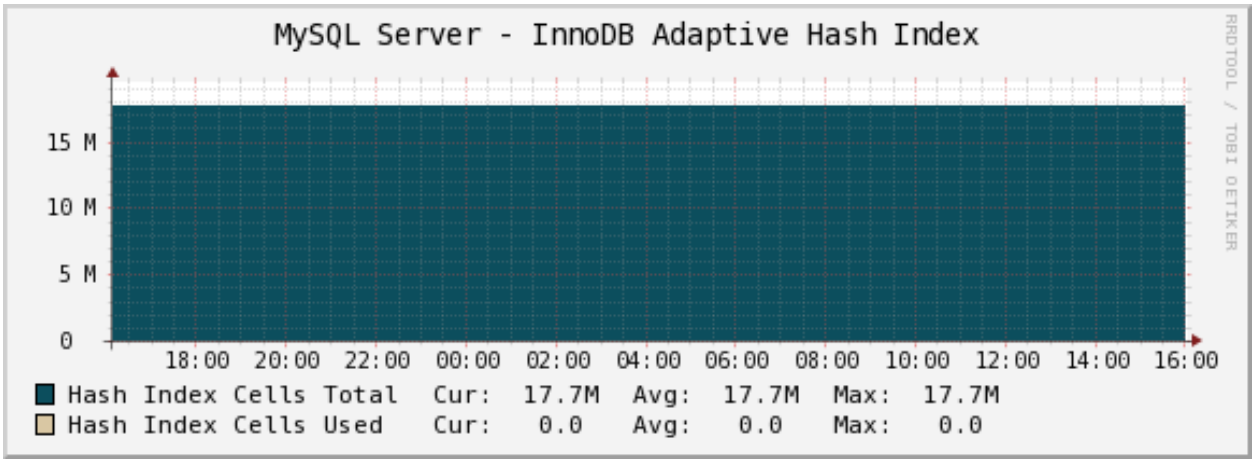
SHOW PROCESSLIST This query requires the PROCESS privilege to generate the MySQL Processlist graph. You can disable this query by editing the `$chk_options` array in the configuration. If you don't grant this privilege Cacti will graph only its own process states.

SHOW /*!50000 ENGINE*/ INNODB STATUS This query requires the SUPER privilege in MySQL 5.1.23 and older versions. It is required for all of the InnoDB graphs. You can disable this query by editing the `$chk_options` array in the configuration. In MySQL 5.1.24 and greater, the required privilege is PROCESS, not SUPER.

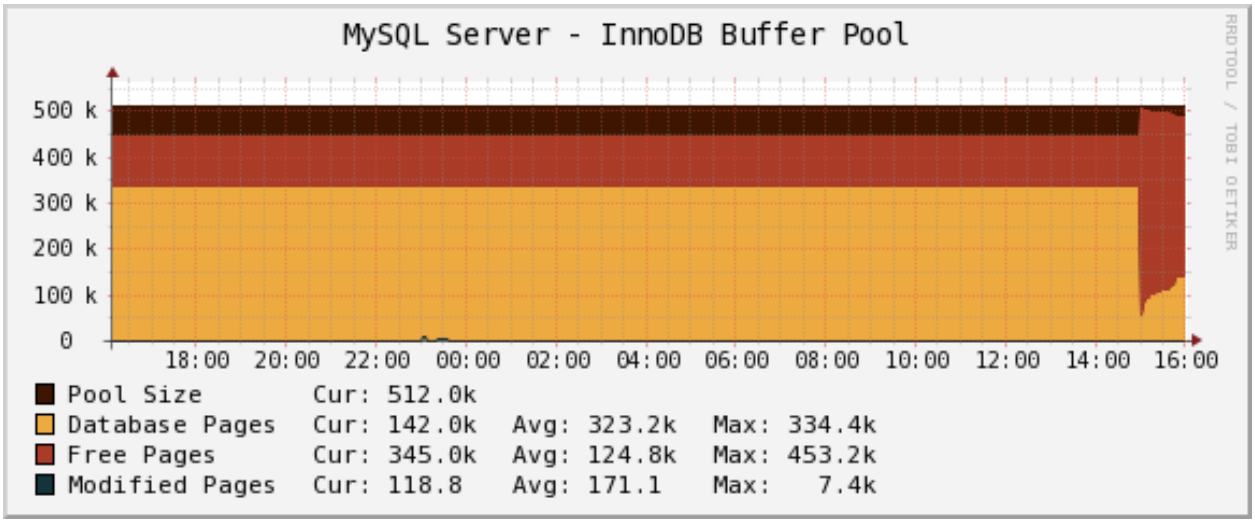
If you disable significant portions of the functionality, such as the InnoDB graphs, then you might want to edit the Host Template to remove unwanted graphs.

Sample Graphs

The following sample graphs demonstrate how the data is presented.

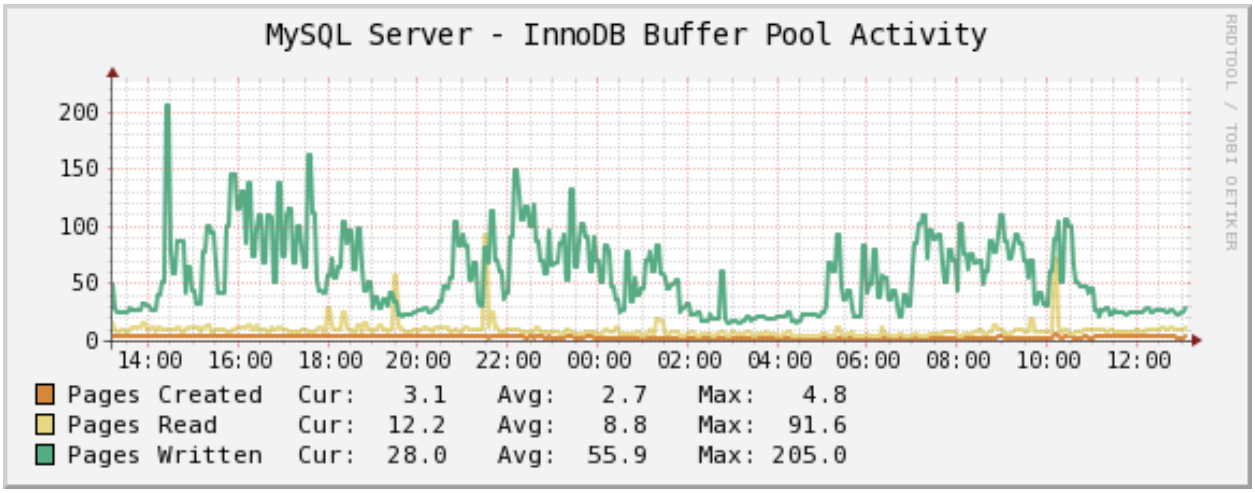


The InnoDB Adaptive Hash Index graph shows the hash index's cells total and cells used. There isn't really anything actionable about this graph: the adaptive hash index isn't designed to be user-tunable, although you can disable it. However, should something go wrong with performance, this graph might provide diagnostic information.

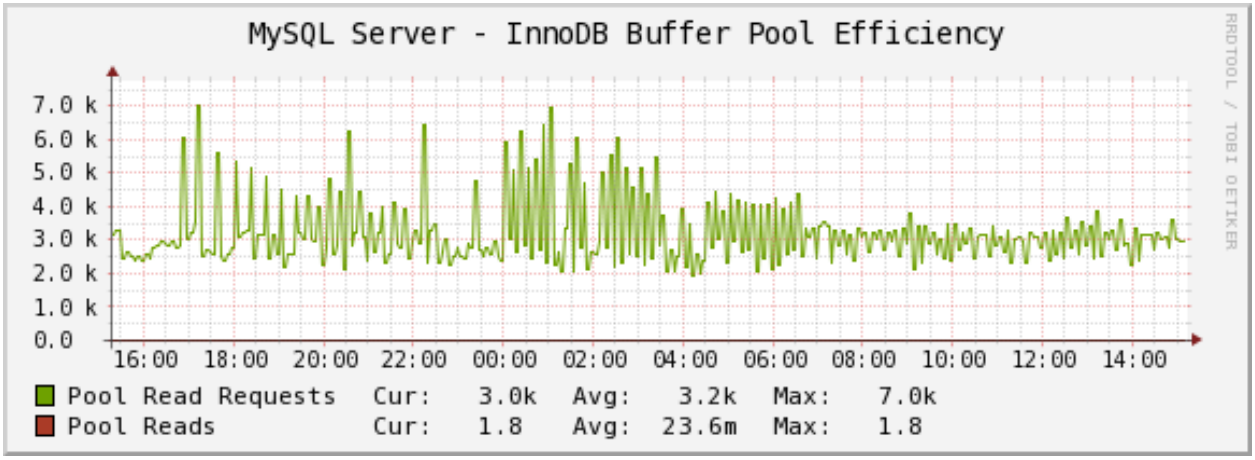


The InnoDB Buffer Pool graph shows the current status of the InnoDB buffer pool: the size, free pages, used (database) pages, and dirty (modified) pages. If too much of the buffer pool fills with dirty pages and InnoDB starts to flush aggressively to reduce that number, you could see cyclical behavior. This might be correlated with intense disk activity and/or periods of reduced throughput. Recent versions of the InnoDB plugin, Percona Server, and Percona XtraDB have various solutions for this problem, should you experience it.

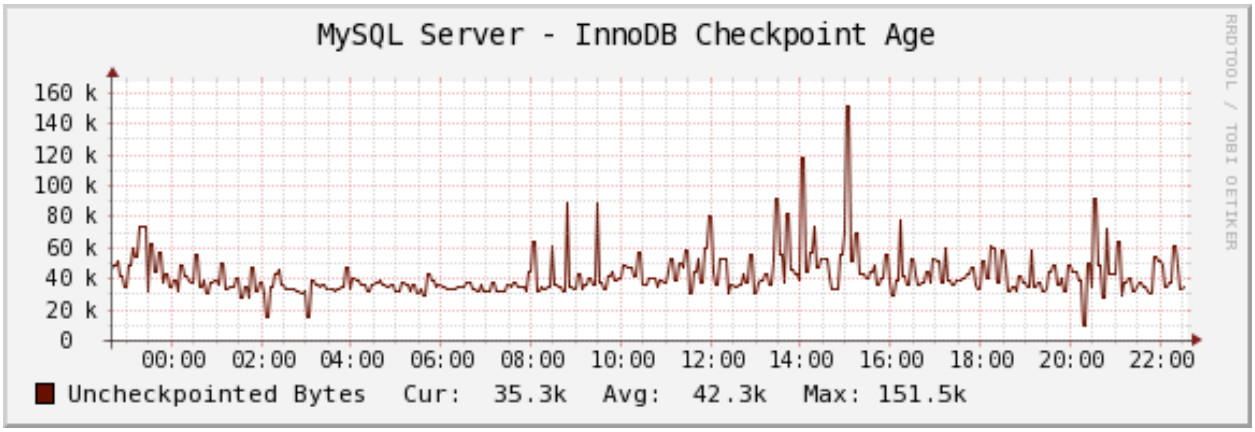
The example graph shows what happens when InnoDB restarts: the buffer pool empties and then fills again.



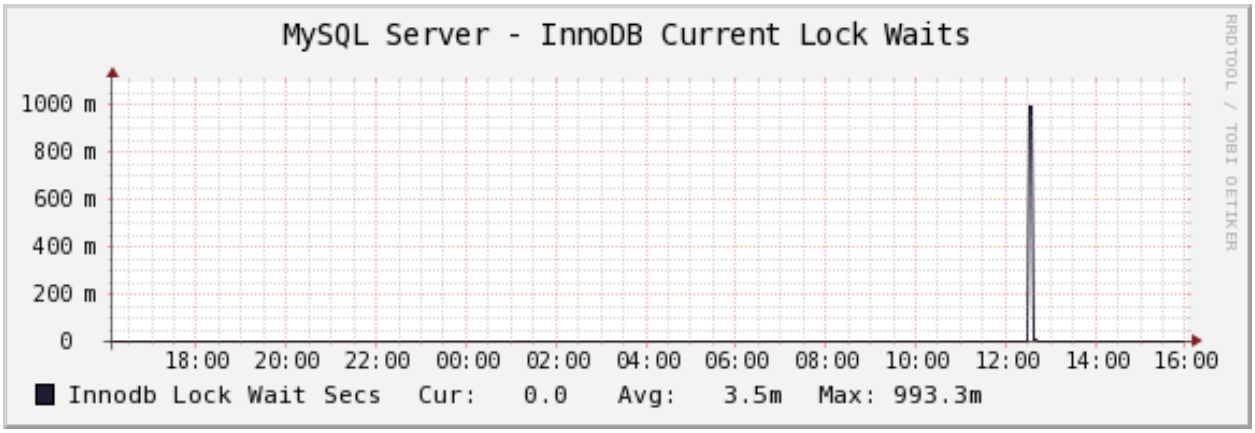
The InnoDB Buffer Pool Activity graph shows activity inside the buffer pool: pages created, read, and written. You can consider it roughly equivalent to the Handler graphs. If you see a sudden change in the graph, you should try to trace it to some change in your application.



The InnoDB Buffer Pool Efficiency graph shows the number of logical read requests InnoDB has done and the number of logical reads that InnoDB could not satisfy from the buffer pool, and had to read directly from the disk.

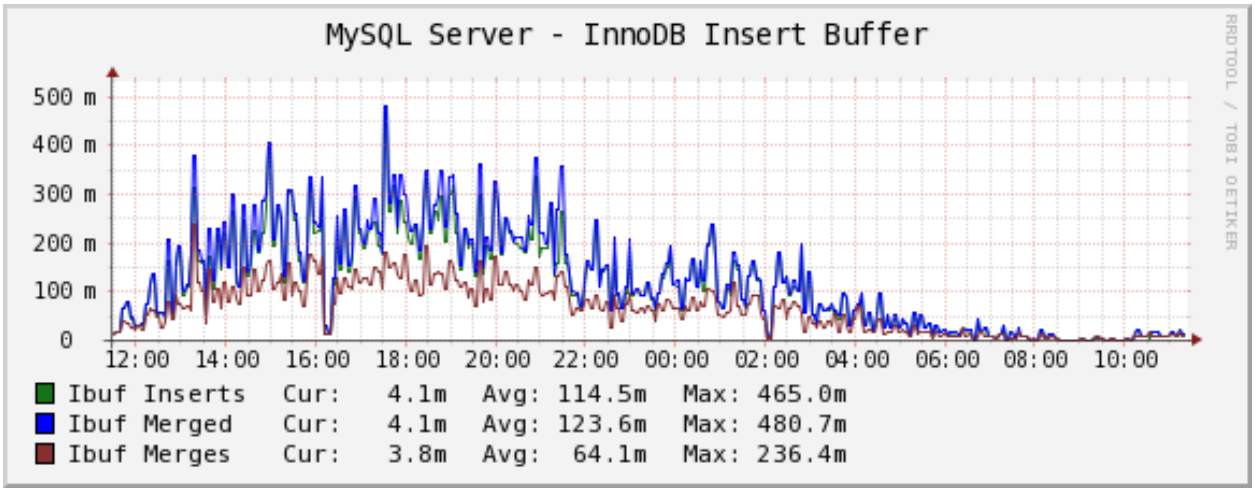


The InnoDB Checkpoint Age graph shows the InnoDB checkpoint age, which is the same thing as the number of uncheckpointed bytes, and thus the amount of log that will need to be scanned to perform recovery if there's a crash. If the uncheckpointed bytes begin to approach the combined size of the InnoDB log files, your system might need larger log files. In addition, a lot of un-checkpointed data might indicate that you'll have a long and painful recovery if there's a crash. If you are writing a tremendous amount of data to the log files, and thus need large log files for performance, you might consider the enhancements in Percona Server.

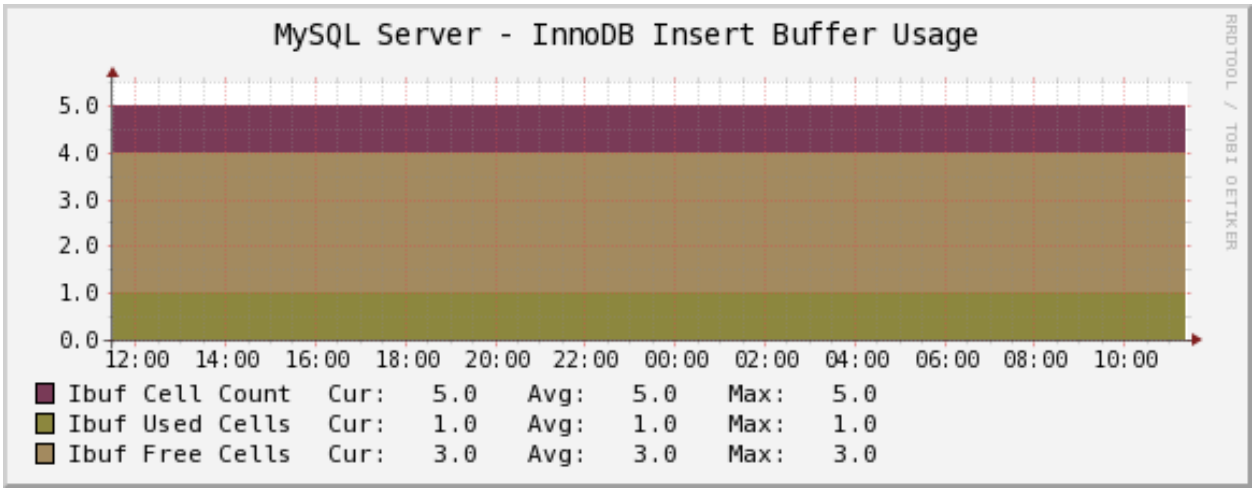


The InnoDB Current Lock Waits graph shows the total number of seconds that InnoDB transactions have been waiting for locks. This is related to the InnoDB Active/Locked Transactions graph, except that it's the sum of the lock

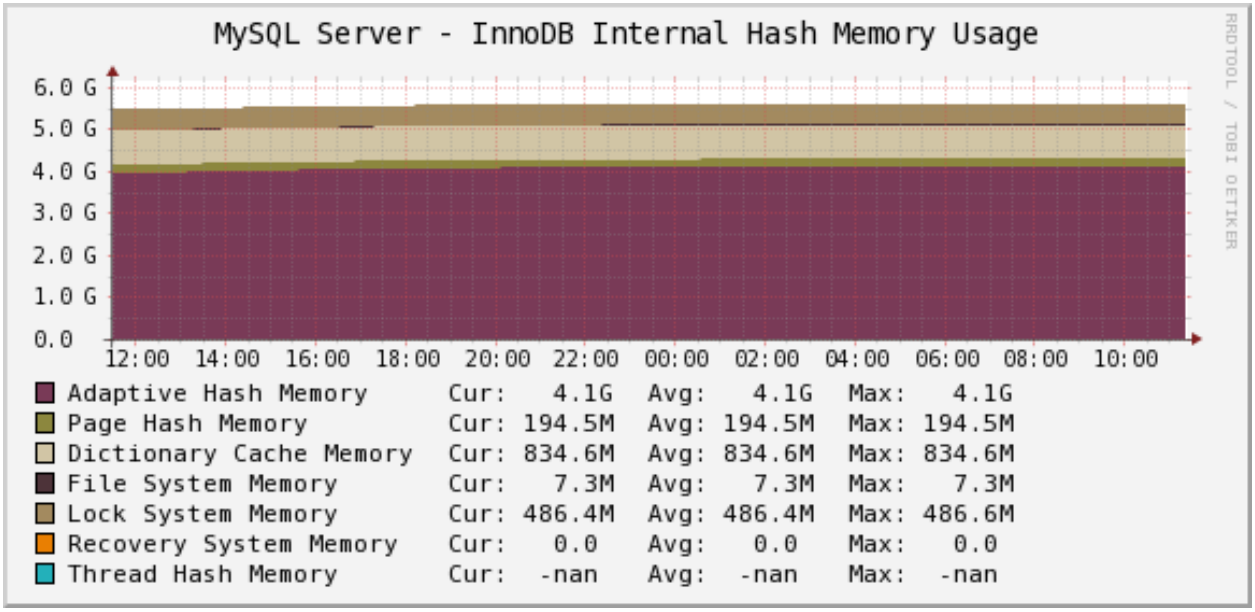
wait time. You might have only one transaction in LOCK WAIT status, but it might be waiting a very long time if `innodb_lock_wait_timeout` is set to a large value. So if you see a large value on this graph, you should investigate for LOCK WAIT transactions.



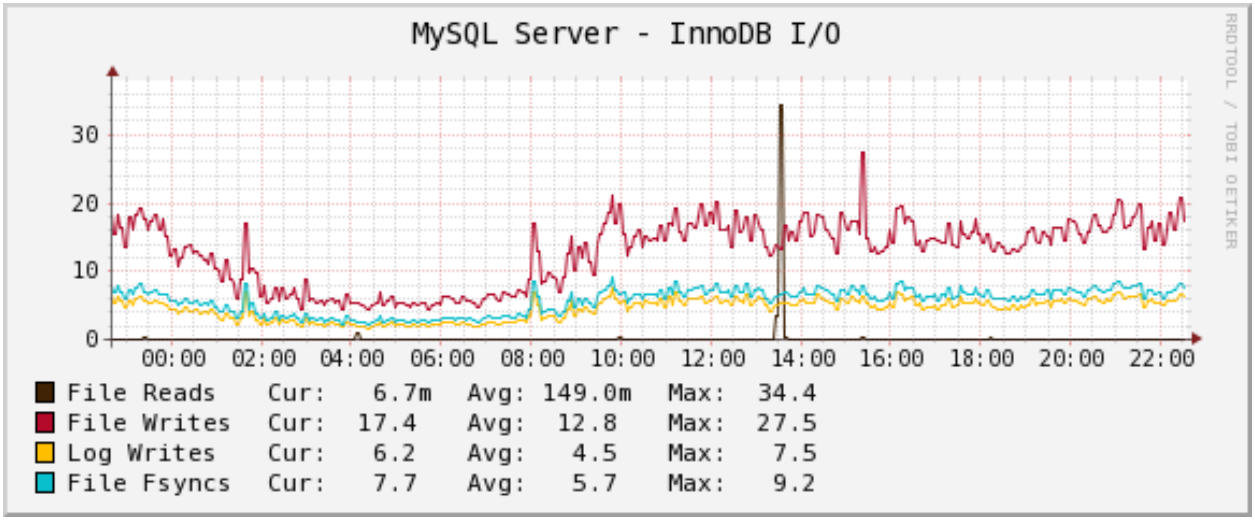
The InnoDB Insert Buffer graph shows information about InnoDB's insert buffer: inserts, merge operations, and merged records. This is not generally actionable, because the insert buffer is not user-configurable in standard MySQL. However, you can use it to diagnose certain kinds of performance problems, such as furious disk activity after you stop the server from processing queries, or during particular types of queries that force the insert buffer to be merged into the indexes. (The insert buffer is sort of a delayed way of updating non-unique secondary indexes.) If the insert buffer is causing problems, then Percona Server might help, because it has some configuration parameters for the buffer.



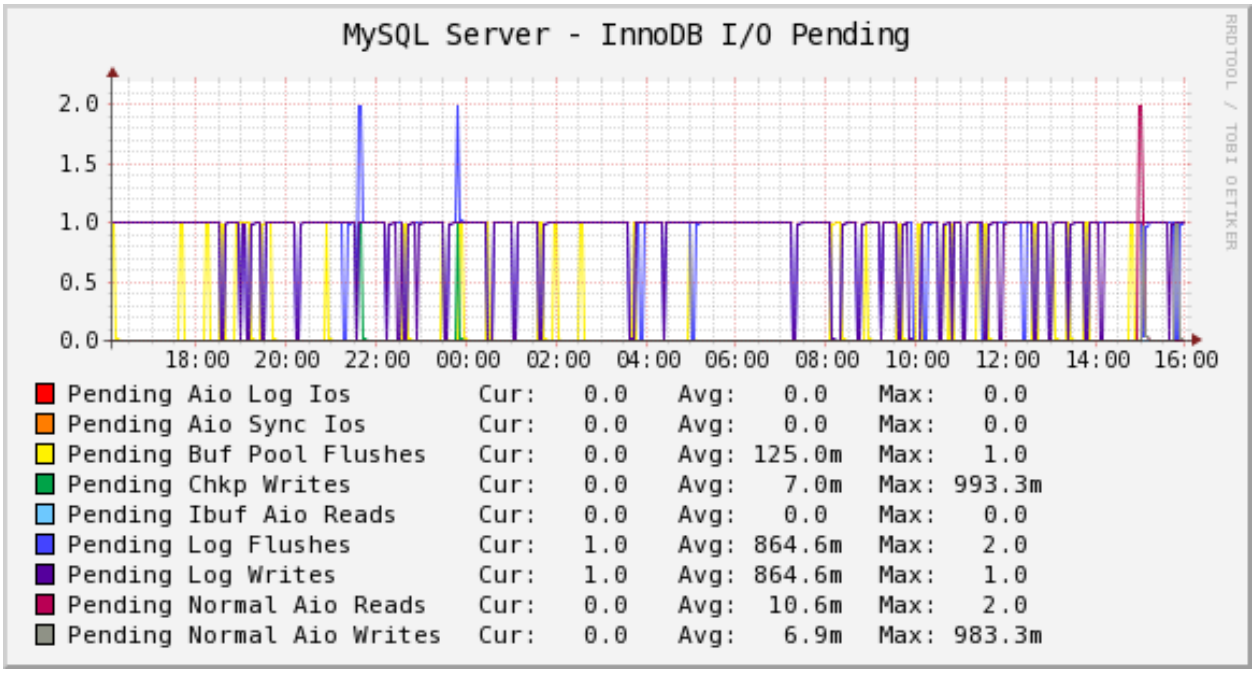
The InnoDB Insert Buffer Usage graph shows the total cells in the insert buffer, and the used and free cells. This is diagnostic only, as in the previous graph. You can use it to see the buffer usage, and thus correlate with server activity that might be hard to explain otherwise.



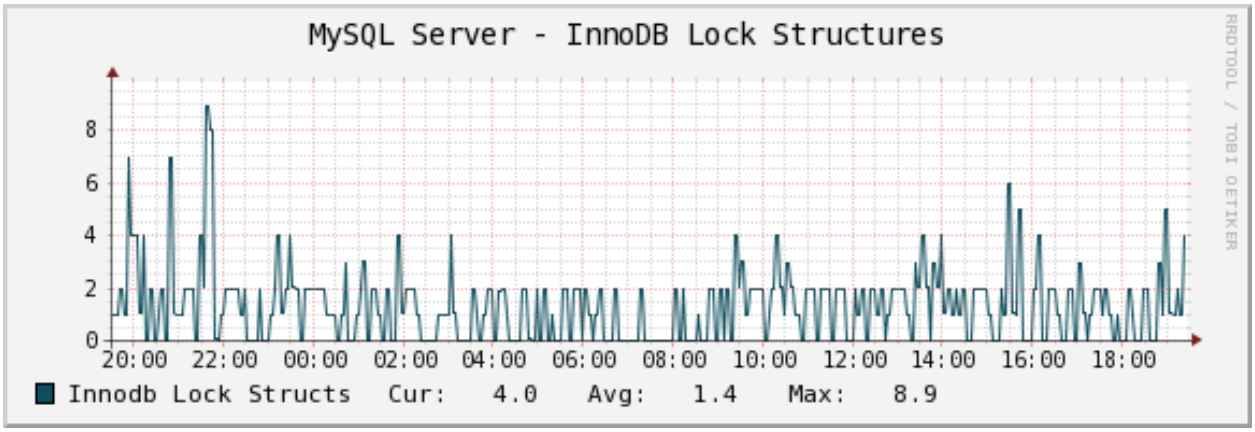
The InnoDB Internal Hash Memory Usage graph shows how much memory InnoDB uses for various internal hash structures: the adaptive hash index, page hash, dictionary cache, filesystem, locks, recovery system, and thread hash. This is available only in Percona Server, and these structures are generally not configurable. However, you might use it to diagnose some kinds of performance problems, such as much greater than expected memory usage. In standard InnoDB, the internal data dictionary tends to consume large amounts of memory when you have many tables, for example. Percona Server lets you control that with some features that are similar to MySQL's table cache.



The InnoDB I/O Activity graph shows InnoDB's I/O activity: file reads and writes, log writes, and fsync() calls. This might help diagnose the source of I/O activity on the system. Some of this can be influenced with InnoDB settings, especially `innodb_flush_log_at_trx_commit`.



The InnoDB I/O Pending graph shows InnoDB's pending synchronous and asynchronous I/O operations in various parts of the engine. Pending I/O is not ideal; ideally you'd like InnoDB's background thread(s) to keep up with writes, and you'd like the buffer pool large enough that reads are not an issue. If you see a lot of pending I/O, you might need more RAM, a bigger buffer pool (or use O_DIRECT to avoid double-buffering), or a faster disk subsystem.

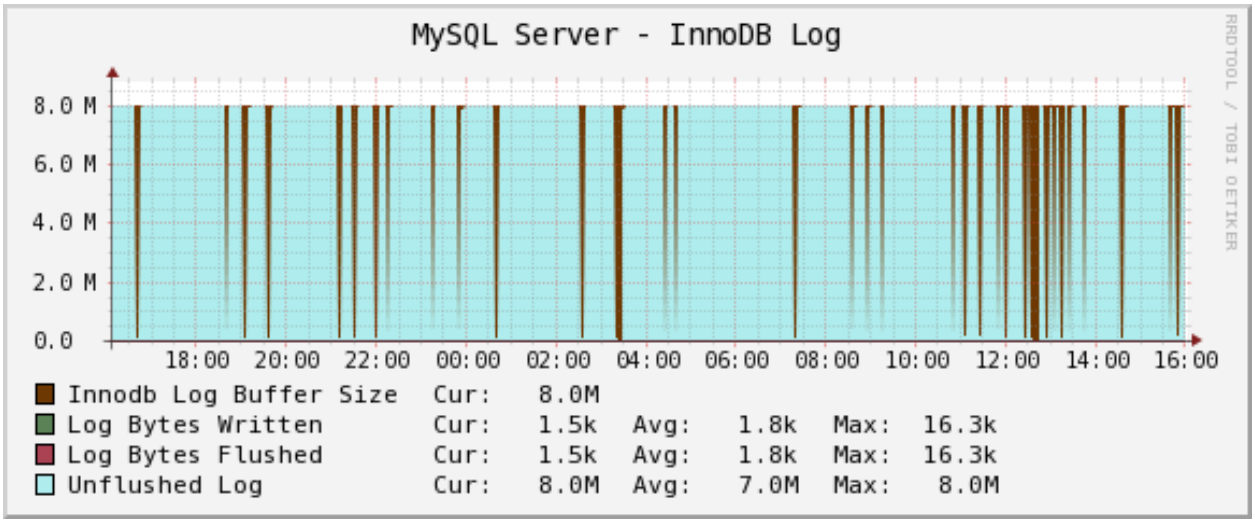


The InnoDB Lock Structures graph shows how many lock structures InnoDB has internally. This should correlate roughly to the number of row locks transactions are currently holding, and might be useful to help diagnose increased lock contention. There is no hard rule about what's a good or bad number of locks, but in case many transactions are waiting for locks, obviously fewer is better.

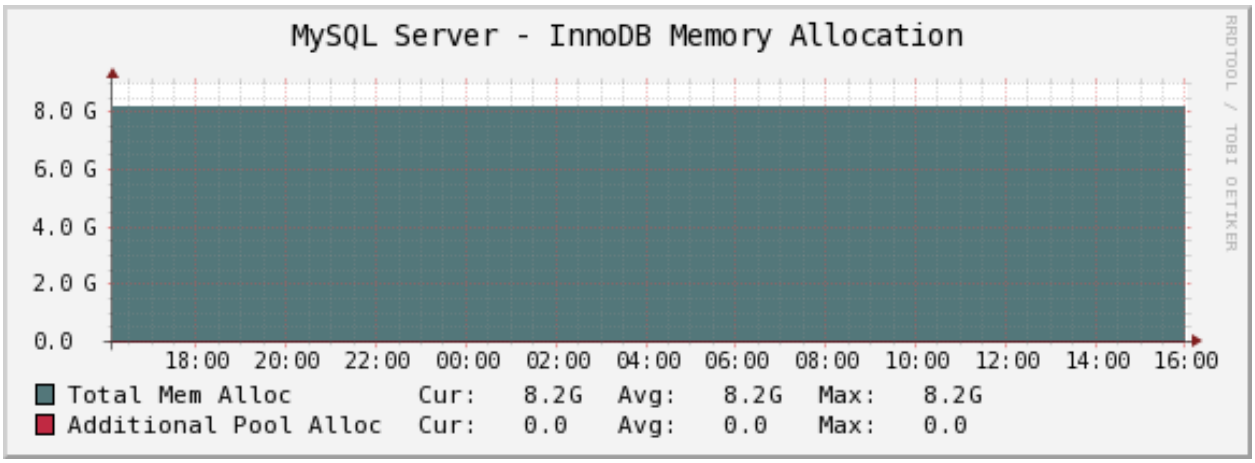
The data comes from lines in SHOW INNODB STATUS such as the following:

```
# 23 lock struct(s), heap size 3024, undo log entries 27
# LOCK WAIT 12 lock struct(s), heap size 3024, undo log entries 5
# LOCK WAIT 2 lock struct(s), heap size 368
```

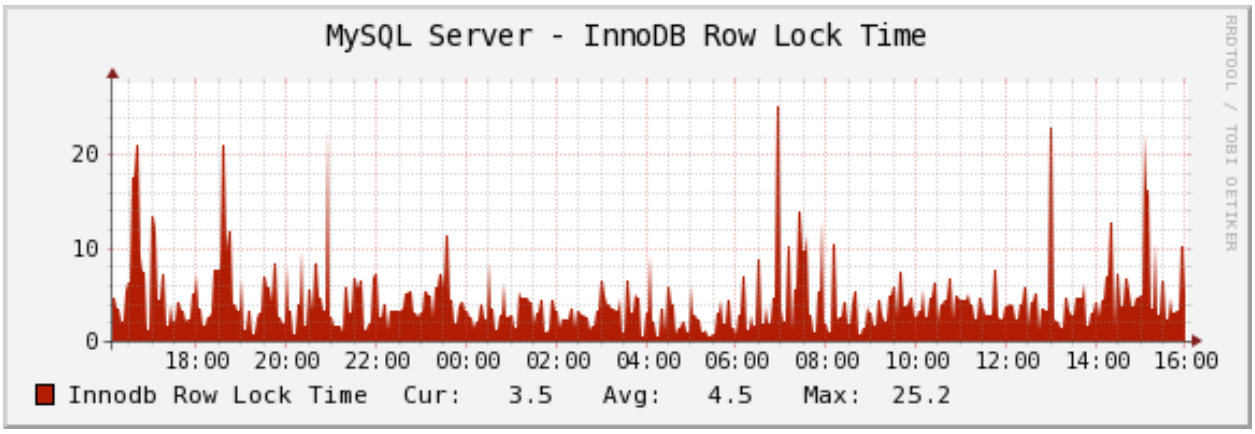
It is the sum of all of the `N lock struct(s)` values.



The InnoDB Log Activity graph shows InnoDB log activity: the log buffer size, bytes written, flushed, and unflushed. If transactions need to write to the log buffer and it's either not big enough or is currently being flushed, they'll stall.

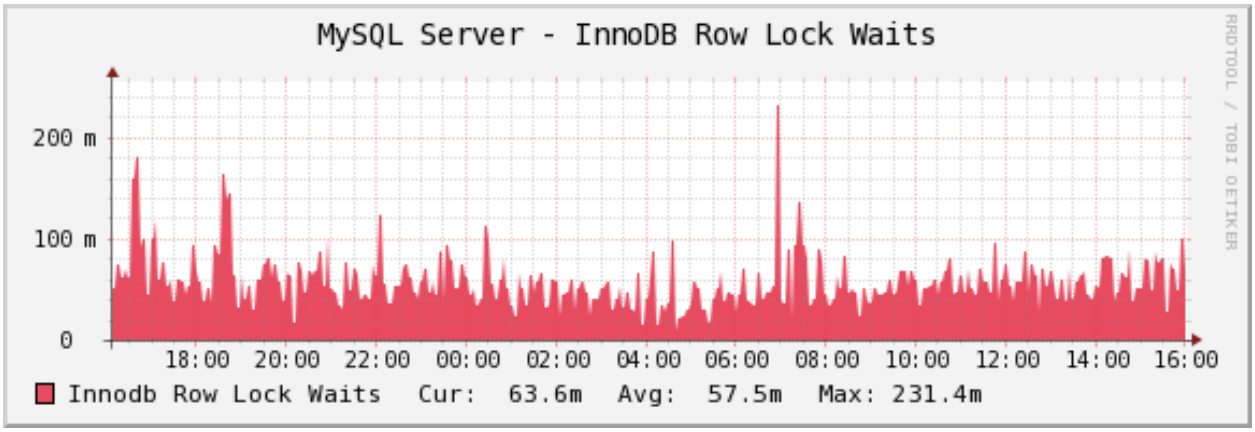


The InnoDB Memory Allocation graph shows InnoDB's total memory allocation, and how much of that is in the additional pool (as opposed to the buffer pool). If a lot of memory is in the additional memory pool, you might suspect problems with the internal data dictionary cache; see above for more on this. Unfortunately, in standard InnoDB it's a bit hard to know where the memory really goes.

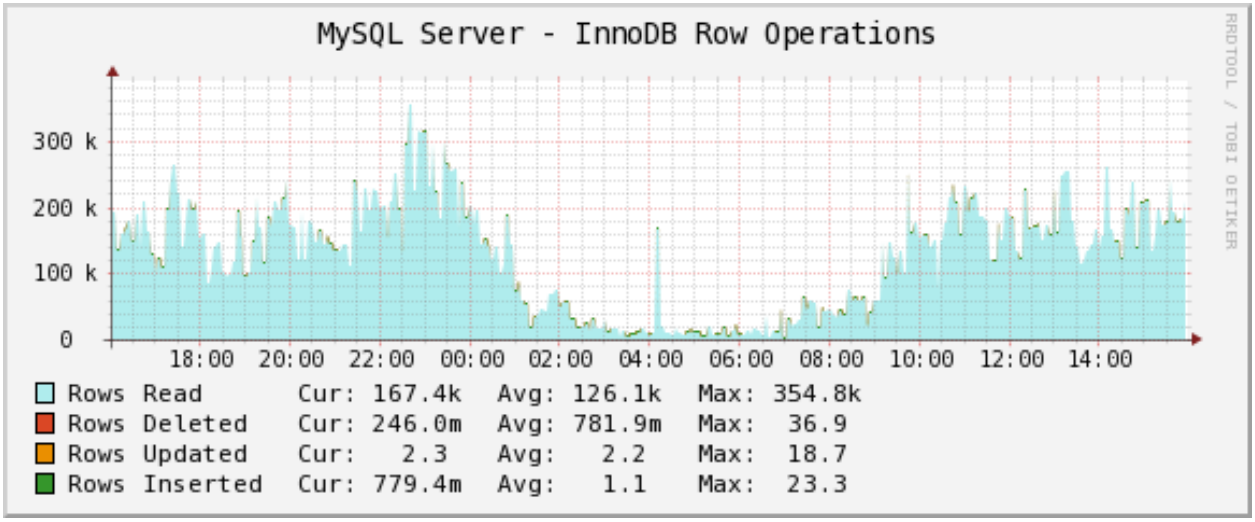


The InnoDB Row Lock Time graph shows the amount of time, in milliseconds, that InnoDB has waited to grant row

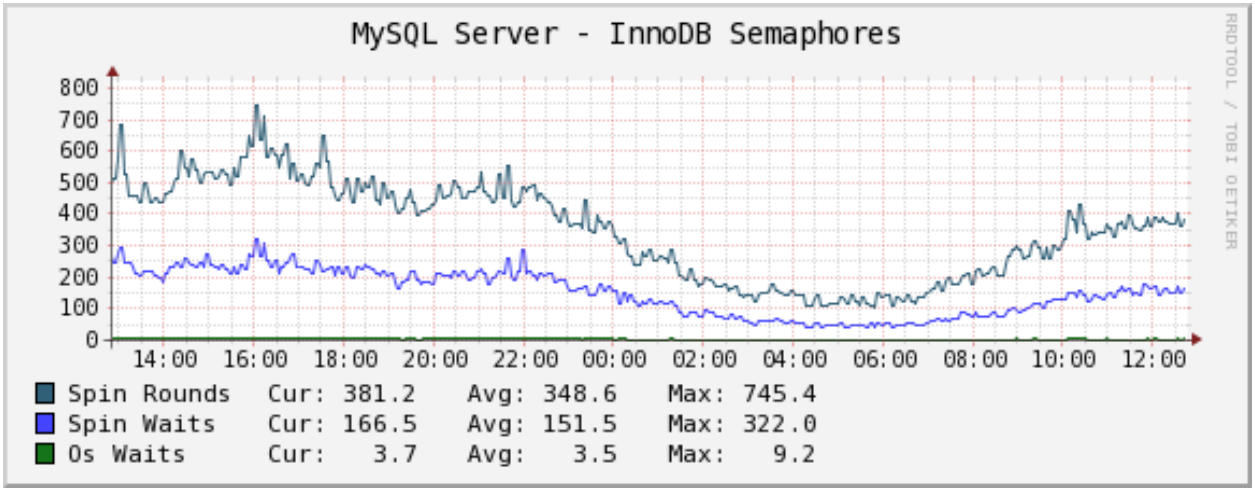
locks. This comes from the `Innodb_row_lock_time` status variable.



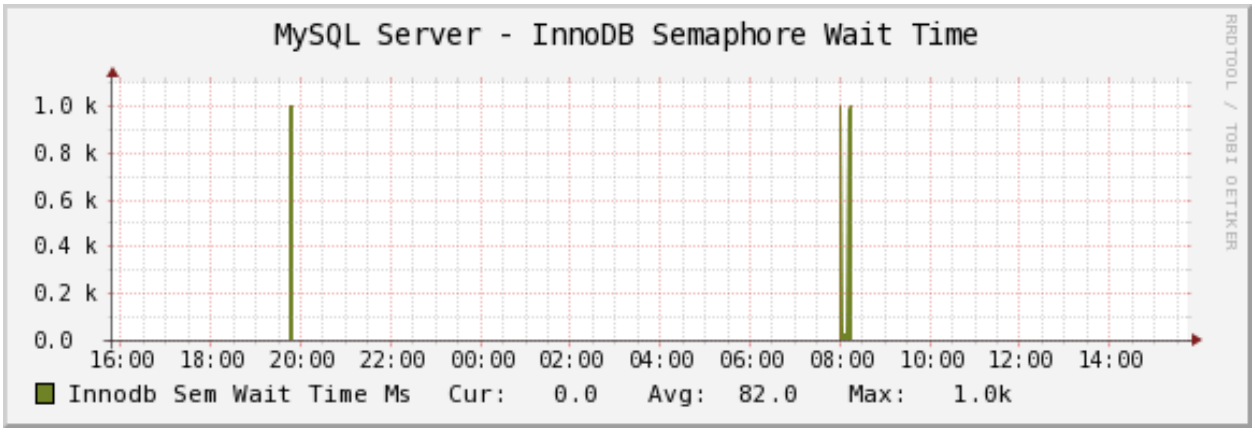
The InnoDB Row Lock Waits graph shows the number of times that InnoDB has waited to grant row locks. This comes from the `Innodb_row_lock_waits` status variable.



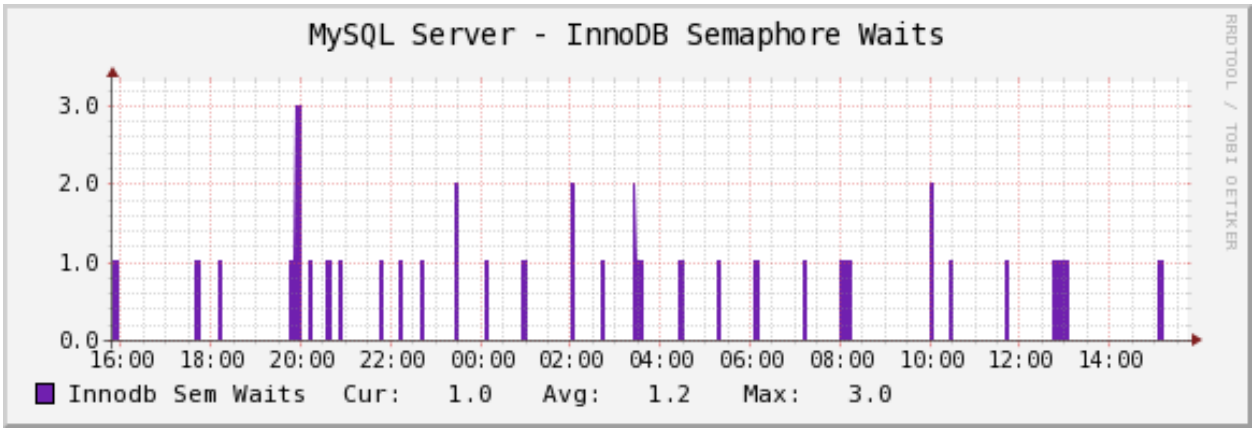
The InnoDB Row Operations graph shows row operations InnoDB has performed: reads, deletes, inserts, and updates. These should be roughly equivalent to Handler statistics, with the exception that they can show internal operations not reflected in the Handler statistics. These might include foreign key operations, for example.



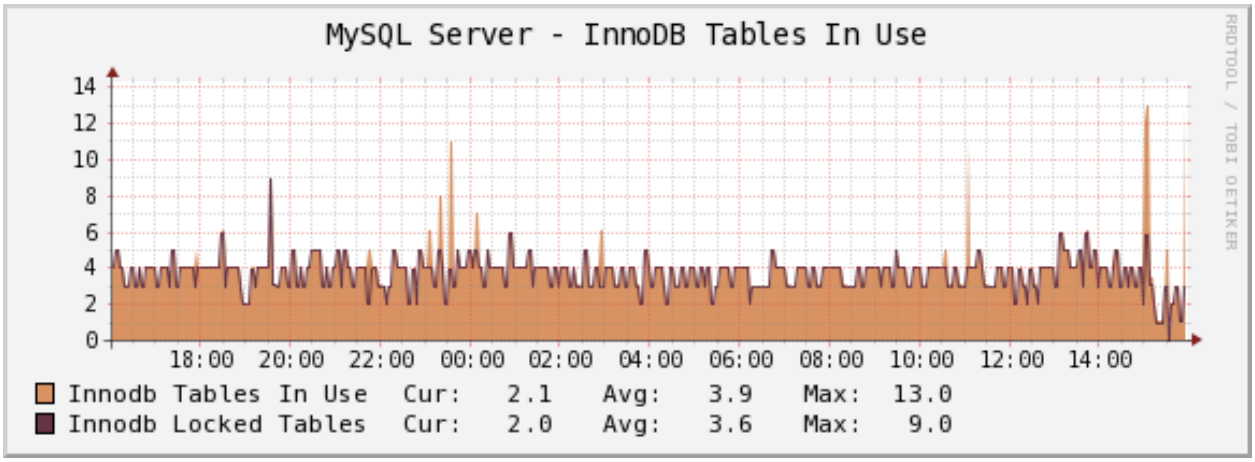
The InnoDB Semaphores graph shows information on InnoDB semaphore activity: the number of spin rounds, spin waits, and OS waits. You might see these graphs spike during times of high concurrency or contention. These graphs basically indicate different types of activity involved in obtaining row locks or mutexes, which are causes of poor scaling in some cases.



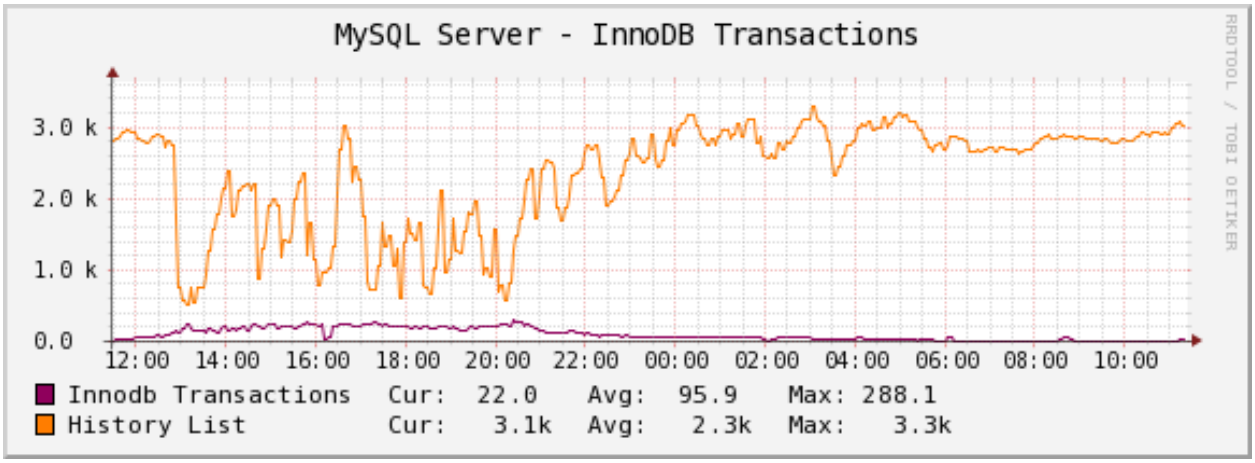
The InnoDB Semaphore Wait Time graph shows the amount of time, in milliseconds, that threads have waited for the semaphore.



The InnoDB Semaphore Waits graph shows the number of times that threads have waited for the semaphore.

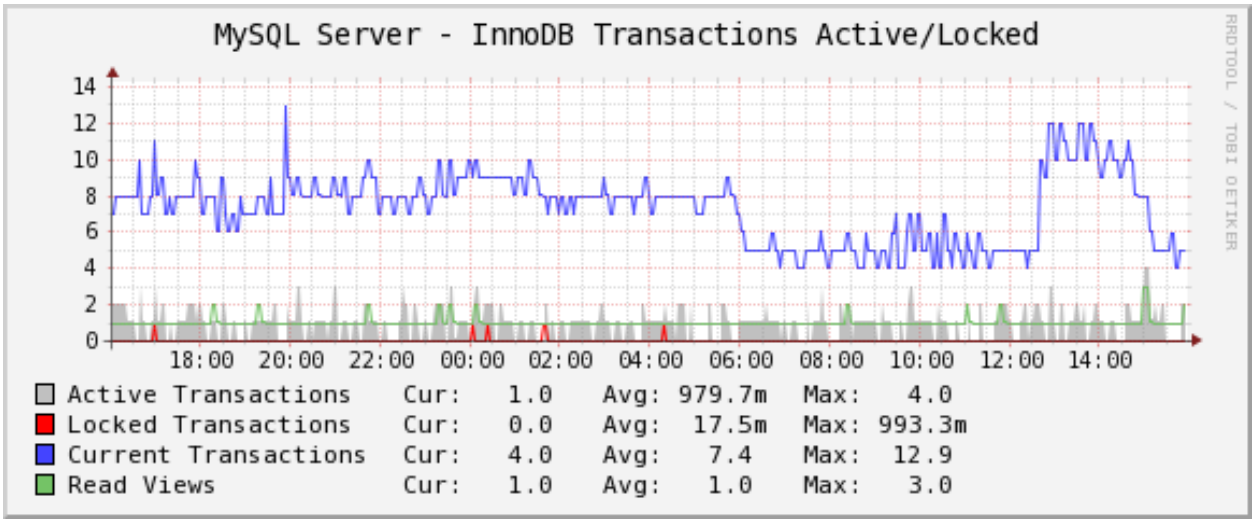


The InnoDB Tables In Use graph shows how many tables InnoDB has in use and how many are locked. If there are spikes in these graphs, you'll probably also see spikes in LOCK WAIT and other signs of contention amongst queries.



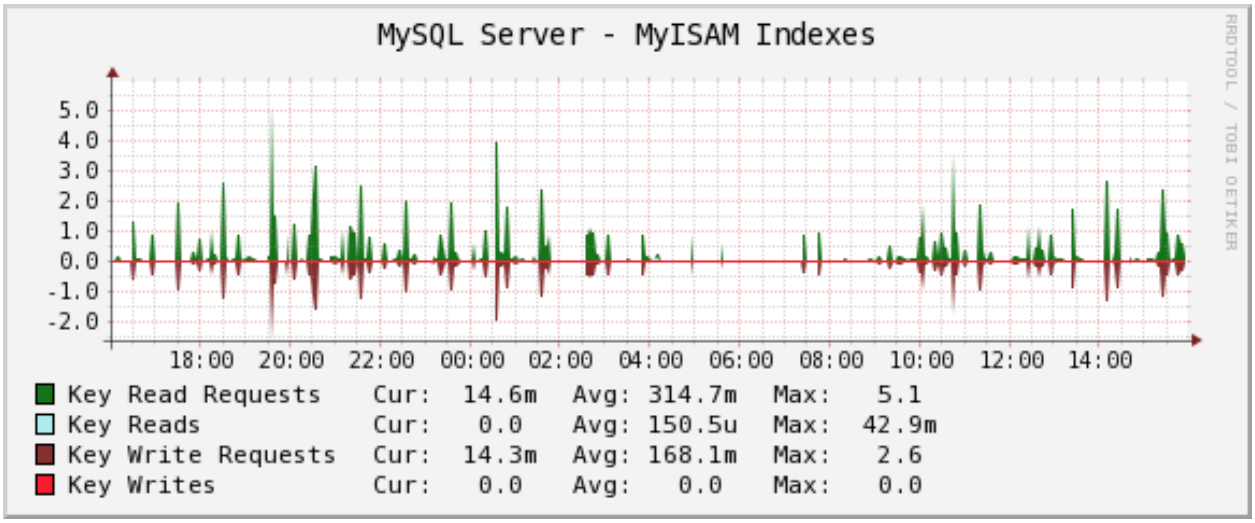
The InnoDB Transactions graph shows information about transactions within InnoDB.

- How changes the internal transaction counter (Trx id counter).
- The length of the history list shows how old the oldest unpurged transaction is. If this grows large, you might have transactions that are staying open a very long time. This means InnoDB can't purge old row versions. It will get bloated and slow as a result. Commit your transactions as quickly as you can.

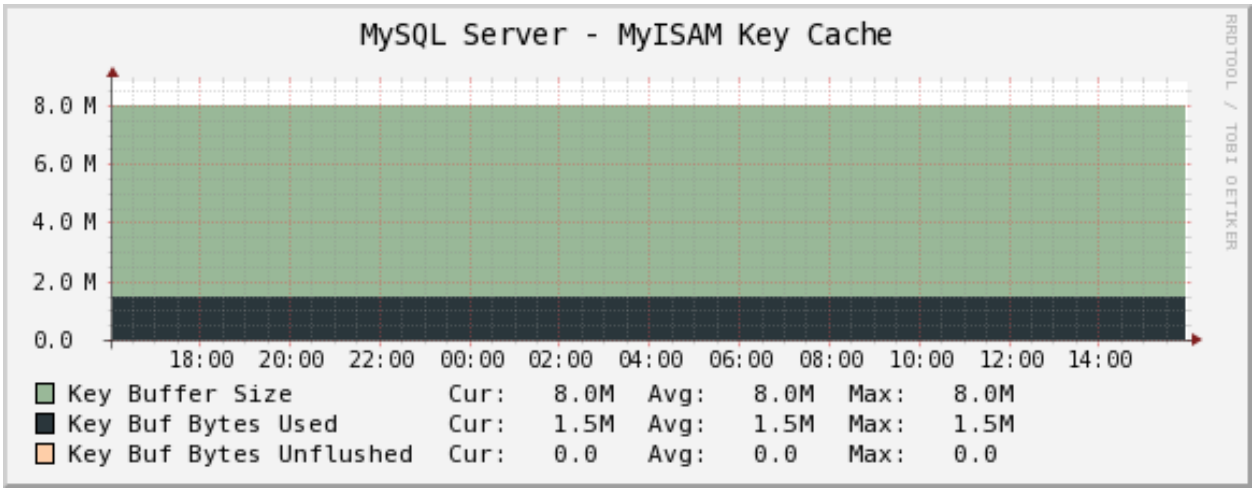


The InnoDB Active/Locked Transactions graph shows InnoDB transaction counts:

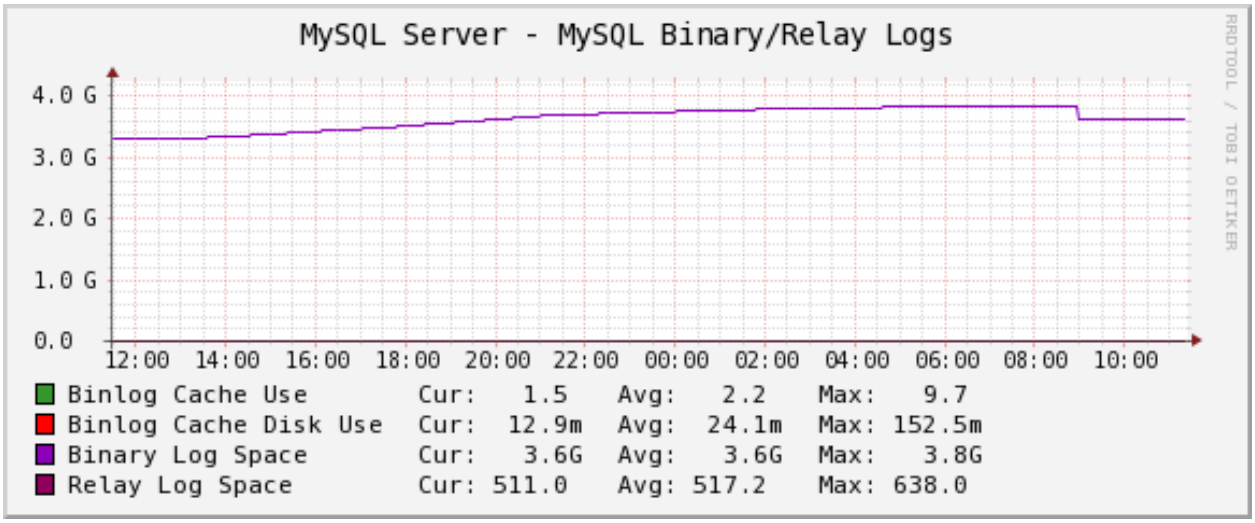
- An active transaction is a transaction that's currently open. It's possible for transactions to be in "not started" status, which really means that this connection to MySQL doesn't actually have a transaction open. A transaction is active between BEGIN and COMMIT. It's also active whilst a query is running, although it might commit immediately due to auto-commit, if applicable. This graph really just shows how much transactional activity is happening on the database.
- A locked transaction is in LOCK WAIT status. This usually means it's waiting for a row lock, but in some cases could be a table lock or an auto-increment lock. If you start to see lock waits, you need to check SHOW INNODB STATUS and search for the string "LOCK WAIT" to examine what's waiting. Lock waits can come from several sources, including too much contention on busy tables, queries accessing data through scans on different indexes, or bad query patterns such as SELECT .. FOR UPDATE.
- The current transactions are all transactions, no matter what status (ACTIVE, LOCK WAIT, not started, etc).
- The number of read views open shows how many transactions have a consistent snapshot of the database's contents, which is achieved by MVCC.



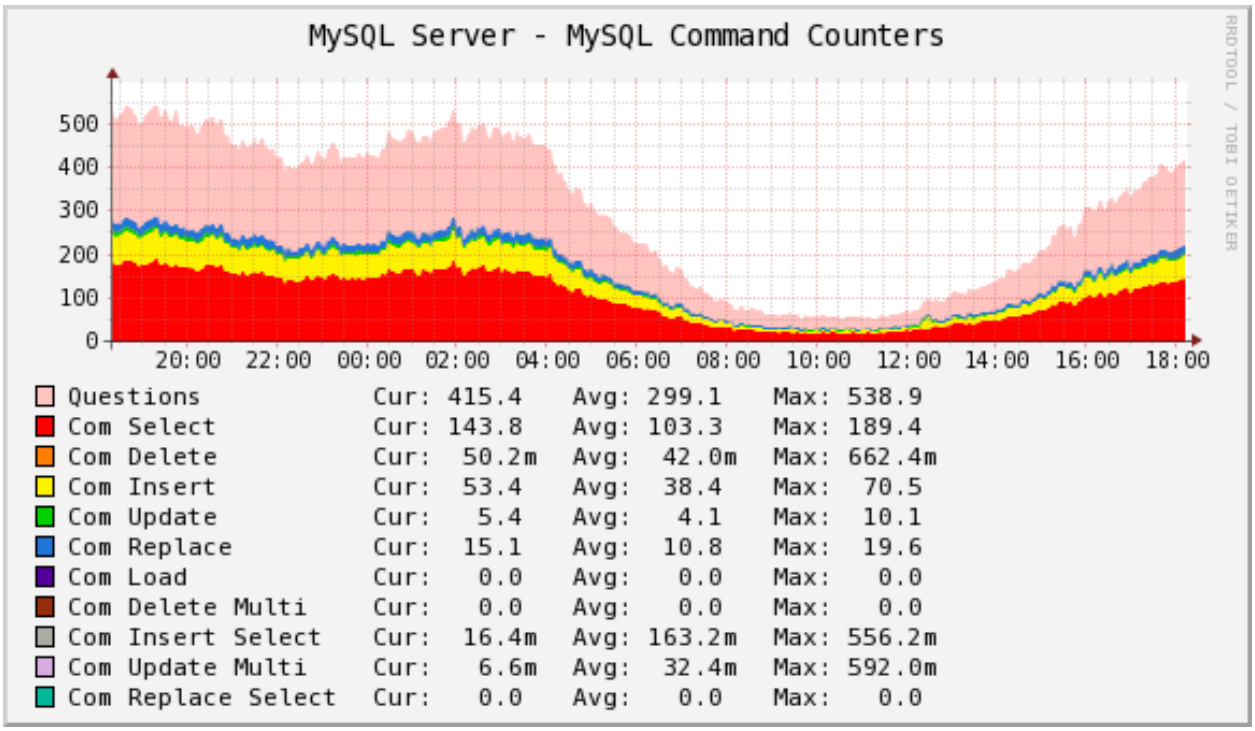
The MyISAM Indexes graph shows information about how many logical and physical reads and writes took place to MyISAM indexes. Probably the most important one is the physical reads. The ratio between logical and physical reads is not very useful to monitor. Instead, you should look at the absolute number of physical reads per second, and compare it to what your disks are capable of. (RRDTool normalizes everything to units of seconds, so this graph's absolute value is the number you need.)



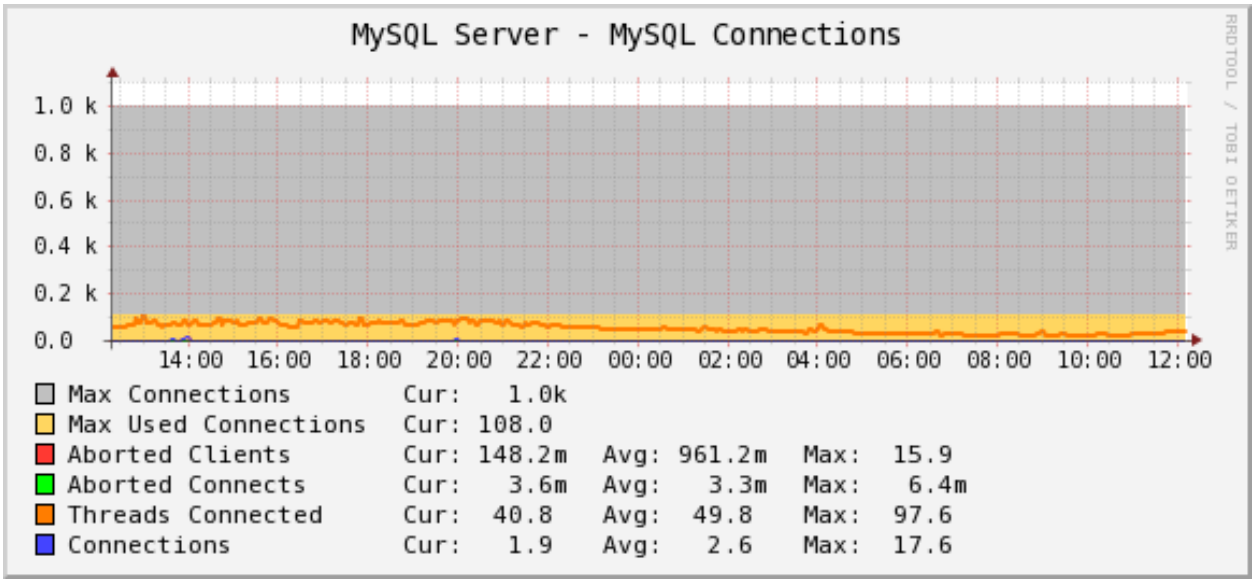
The MyISAM Key Cache graph shows the size of the key buffer, how much of it is used, and how much is unflushed. Memory that isn't used might not really be allocated; the key buffer isn't allocated to its full size.



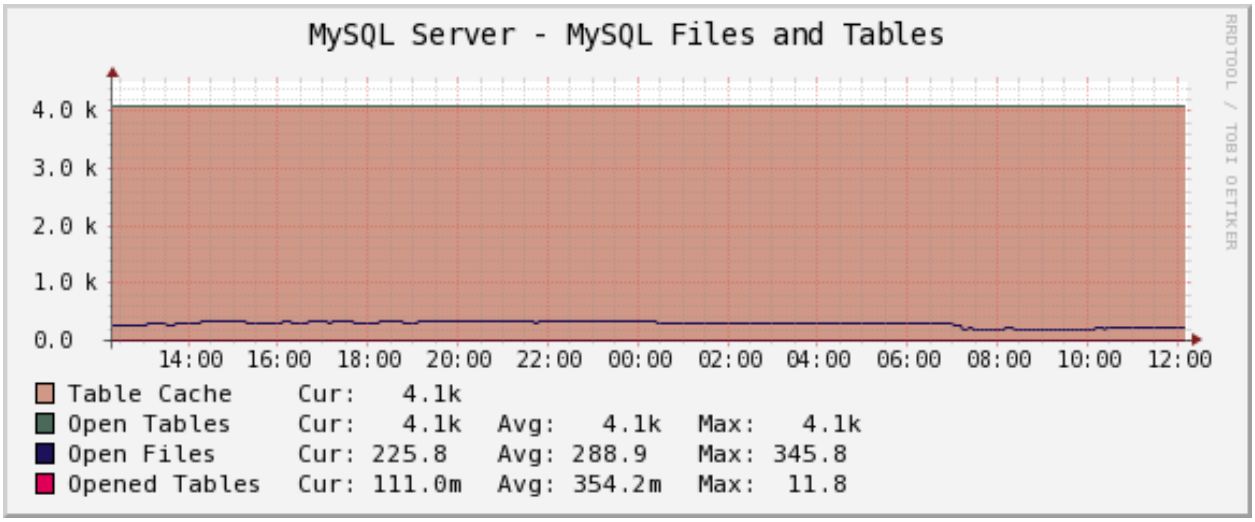
The MySQL Binary/Relay logs graph shows information about the space used by the server binary and relay logs. The variations in the sizes are when the logs are purged, probably due to `expire_logs_days` being set. If this suddenly grows large, look for problems in purging, which might be caused by a configuration change, or by someone manually deleting a file and causing the automatic purge to stop working.



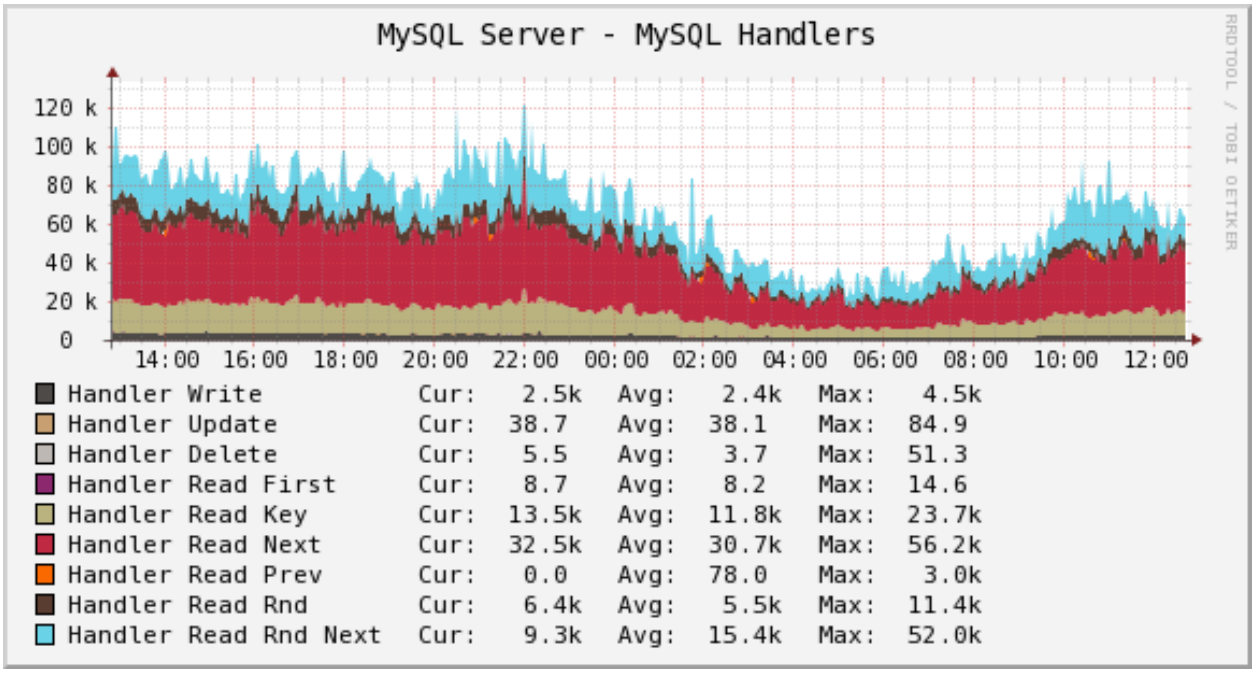
The MySQL Command Counters graph shows counters for various MySQL commands. These are derived from the `Com_` counters from `SHOW STATUS`. If there is a change in the graph, it indicates that something changed in the application.



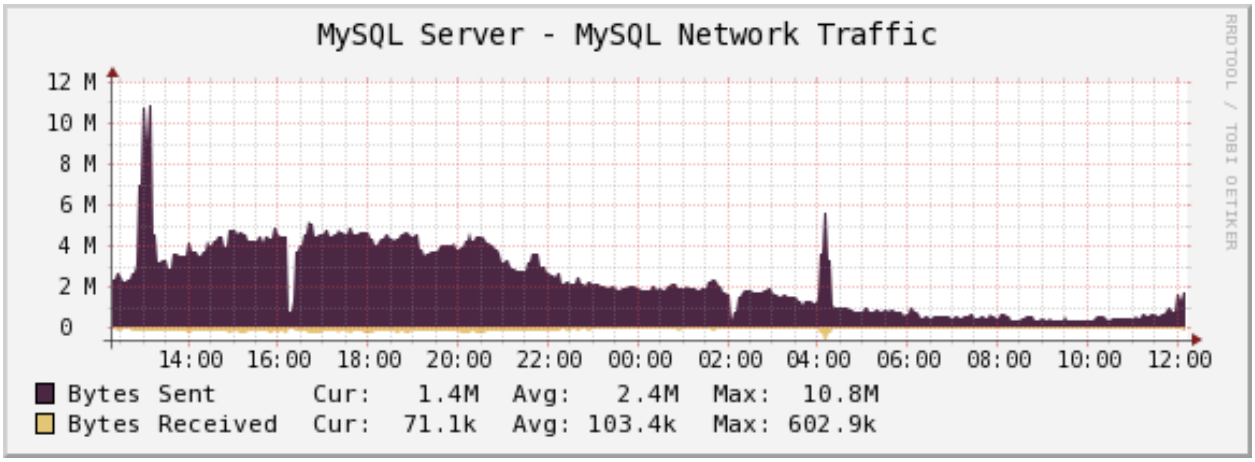
The MySQL Connections graph shows information about the connection parameters and counters inside MySQL: connections permitted, connections used, connections aborted, clients aborted, current connections, and connections created. Probably the most interesting are the aborted clients and connections, which might indicate a malfunctioning application that disconnects ungracefully, an idle connection timing out, network problems, bad authentication attempts, or similar.



The MySQL Files and Tables graph shows status of MySQL's table cache and file handles: the size of the cache, and how many open files and tables there are. This graph is not likely to contain much information in the normal course of events.

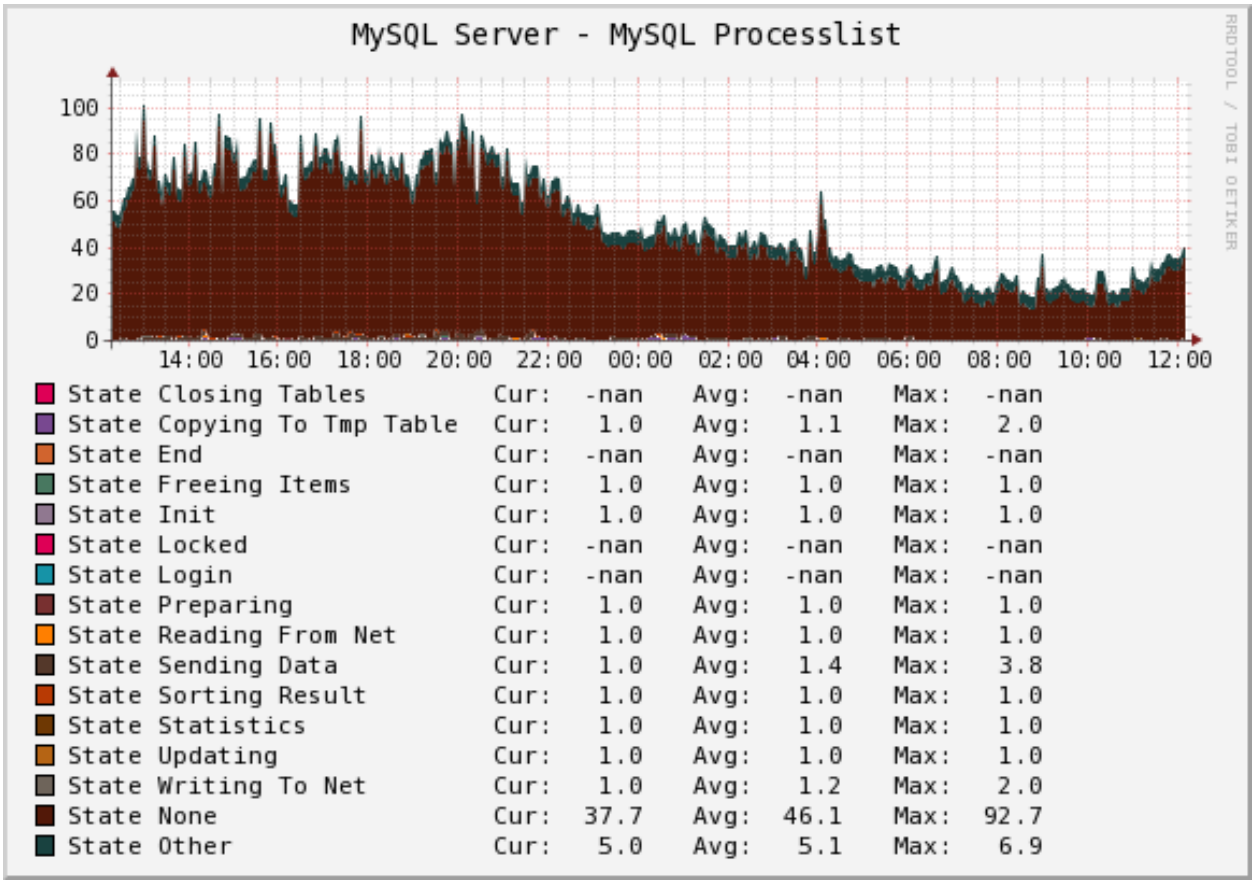


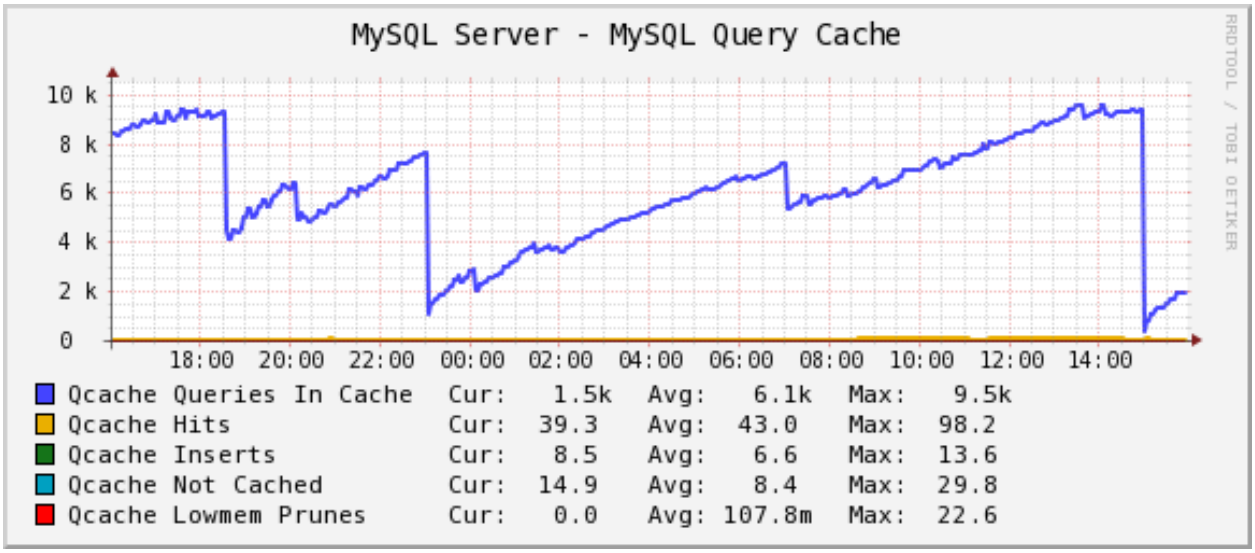
The MySQL Handlers graph shows the various Handler counters, which record how many operations MySQL has done through the storage engine API. Changes in indexing will probably show up clearly here: a query that used to do a table scan but now has a good index to use will cause different Handler calls to be used, for example. If you see sudden changes, it probably correlates with schema changes or a different mixture of queries. If you see a large spike of `Handler_read_rnd_next`, it probably means something was doing a lot of table scans.



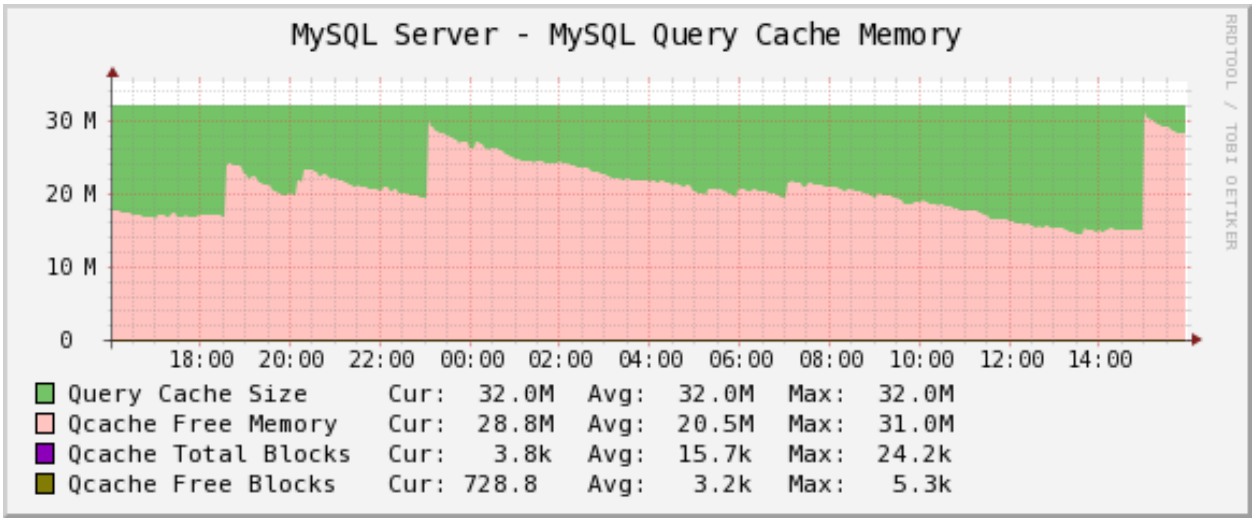
The MySQL Network Traffic graph shows network traffic to and from the MySQL Server, in bytes.

The MySQL Processlist shows the number (count) of queries from SHOW PROCESSLIST in given statuses. Some of the statuses are lumped together into the "other" category. This is a "scoreboard" type of graph. In most cases, you should see mostly Other, or a few of the statuses like "Sending data". Queries in Locked status are the hallmark of a lot of MyISAM table locking. Any mixture of statuses is possible, and you should investigate sudden and systemic changes.

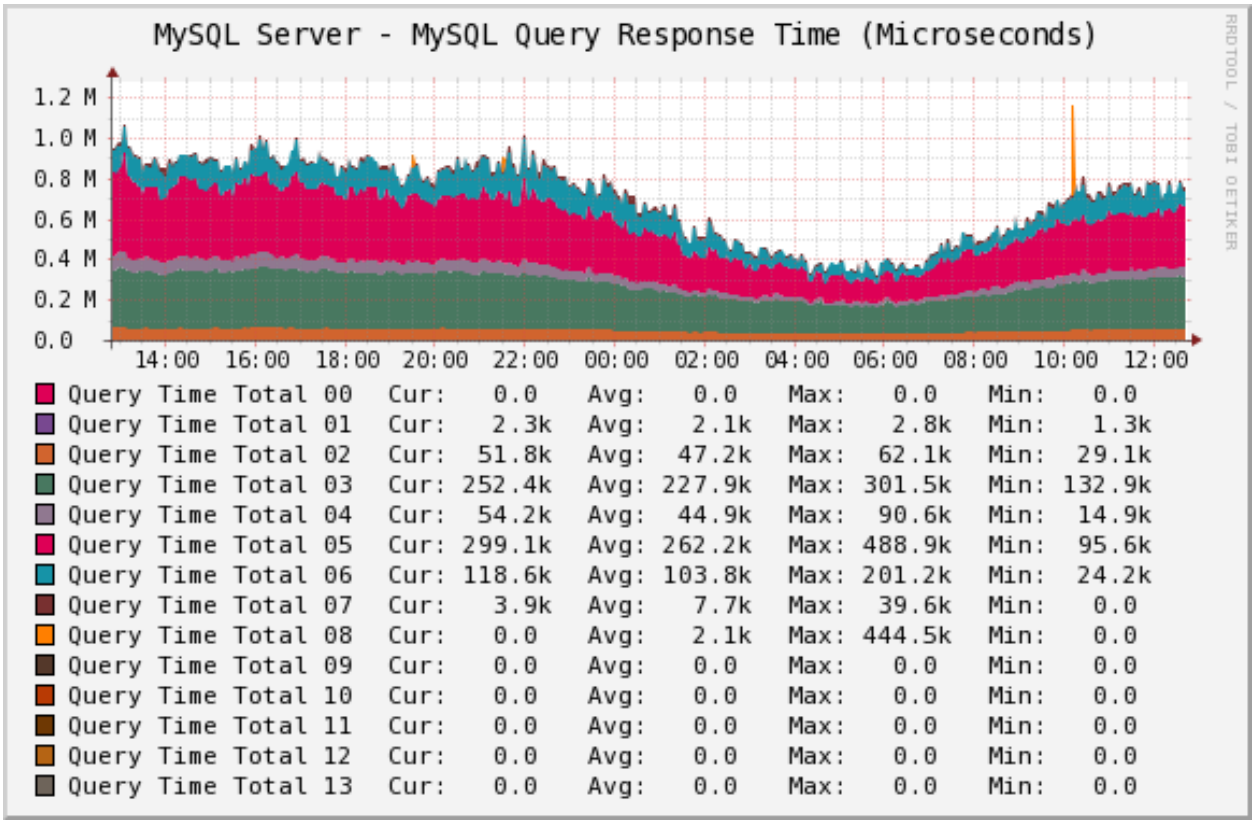

MySQL Server - MySQL Processlist
RRDTOOL / TOBI OETIKER
100
80
60
40
20
0
14:00 16:00 18:00 20:00 22:00 00:00 02:00 04:00 06:00 08:00 10:00 12:00
State Closing Tables Cur: -nan Avg: -nan Max: -nan
State Copying To Tmp Table Cur: 1.0 Avg: 1.1 Max: 2.0
State End Cur: -nan Avg: -nan Max: -nan
State Freeing Items Cur: 1.0 Avg: 1.0 Max: 1.0
State Init Cur: 1.0 Avg: 1.0 Max: 1.0
State Locked Cur: -nan Avg: -nan Max: -nan
State Login Cur: -nan Avg: -nan Max: -nan
State Preparing Cur: 1.0 Avg: 1.0 Max: 1.0
State Reading From Net Cur: 1.0 Avg: 1.0 Max: 1.0
State Sending Data Cur: 1.0 Avg: 1.4 Max: 3.8
State Sorting Result Cur: 1.0 Avg: 1.0 Max: 1.0
State Statistics Cur: 1.0 Avg: 1.0 Max: 1.0
State Updating Cur: 1.0 Avg: 1.0 Max: 1.0
State Writing To Net Cur: 1.0 Avg: 1.2 Max: 2.0
State None Cur: 37.7 Avg: 46.1 Max: 92.7
State Other Cur: 5.0 Avg: 5.1 Max: 6.9



The MySQL Query Cache graph shows information about the query cache inside MySQL: the number of queries in the cache, inserted, queries not cached, queries pruned due to low memory, and cache hits.



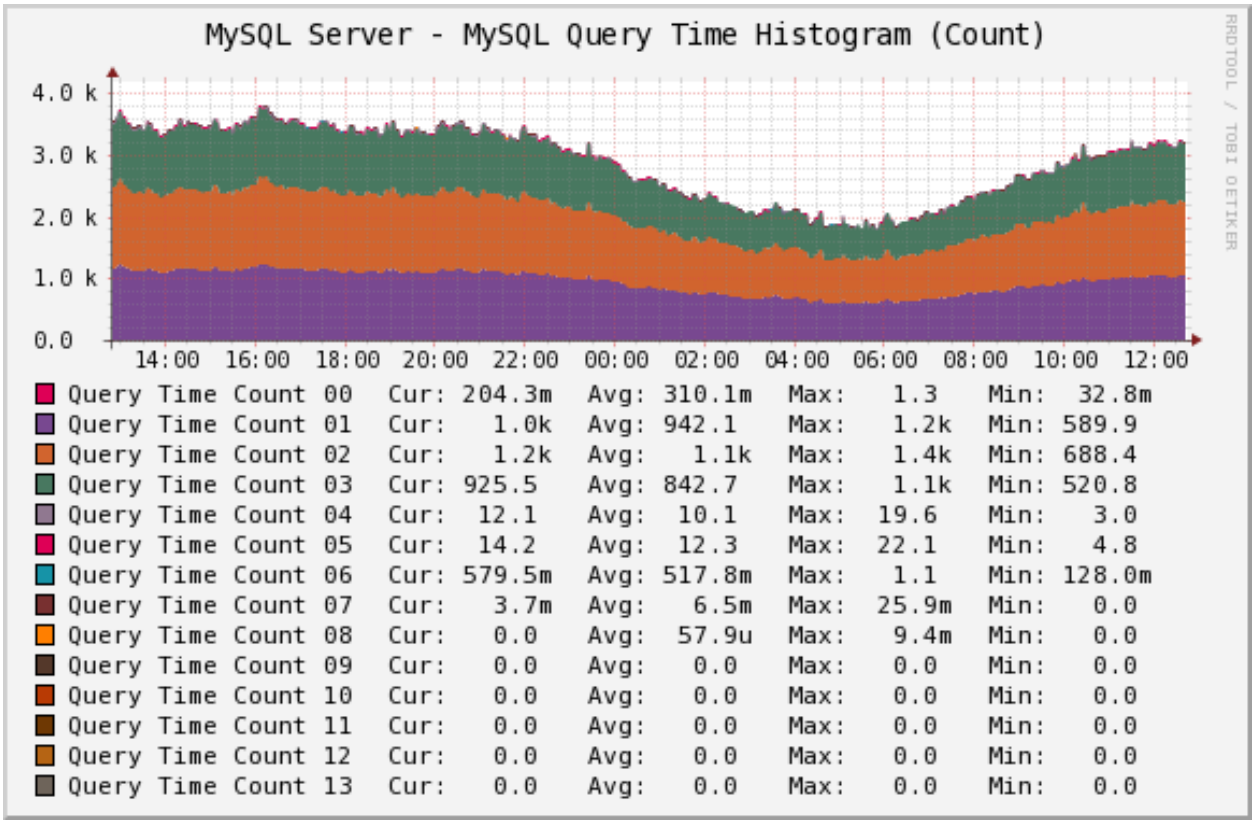
The MySQL Query Cache Memory graph shows information on the query cache's memory usage: total size, free memory, total blocks and free blocks. Blocks are not of a uniform size, despite the name.



The MySQL Query Response Time (Microseconds) graph displays a histogram of the query response time distribution available in Percona Server 5.1/5.5. Because the time units are user-configurable, exact unit labels are not displayed; rather, the graph simply shows the values. There are 14 time units by default in Percona Server, so there are 13 entries on the graph (the 14th is non-numeric, so we omit it).

The graph actually displays the amount of response time spent by the server on queries of various lengths. See the Percona documentation for more. The units are in microseconds on the graph, because RRDtool cannot store floating-point values.

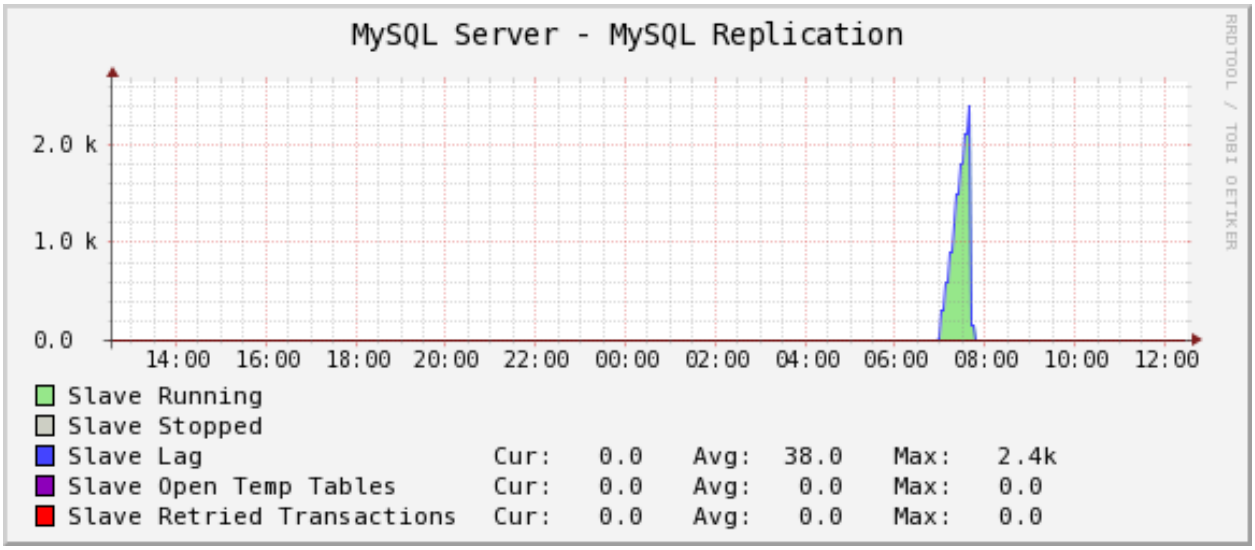
This also may work with MariaDB.



The MySQL Query Time Histogram (Count) graph displays a histogram of the query response time distribution available in Percona Server 5.1/5.5. Because the time units are user-configurable, exact unit labels are not displayed; rather, the graph simply shows the values. There are 14 time units by default in Percona Server, so there are 13 entries on the graph (the 14th is non-numeric, so we omit it).

The graph displays the number of queries that fell into each time division. See the Percona documentation for more.

This also may work with MariaDB.

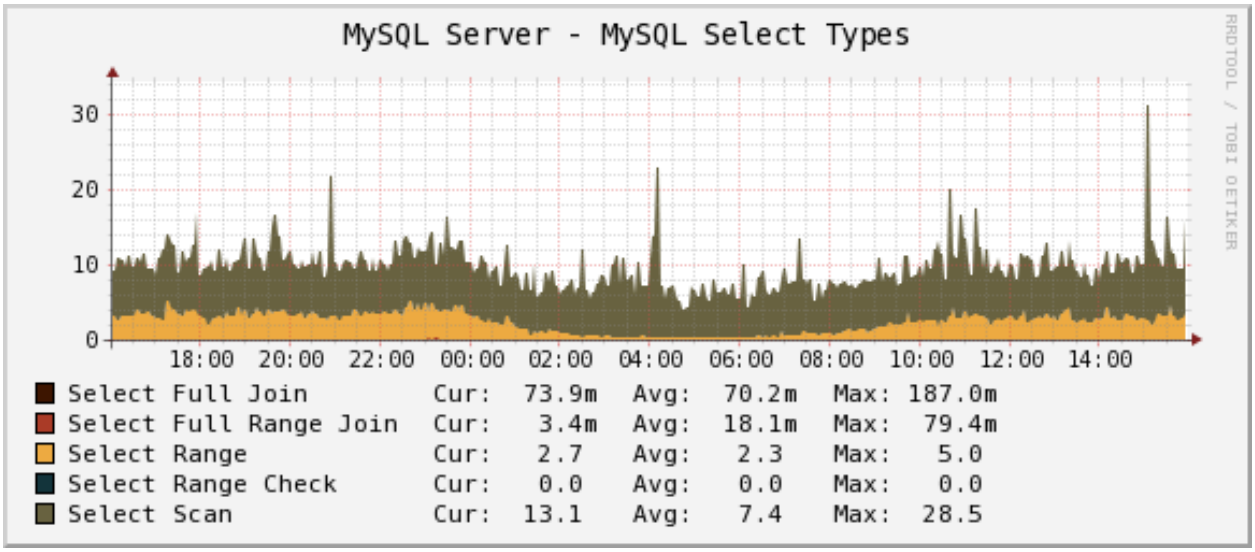


The MySQL Replication Status graph displays the status of the replication thread. There are two ways to measure the replication delay:

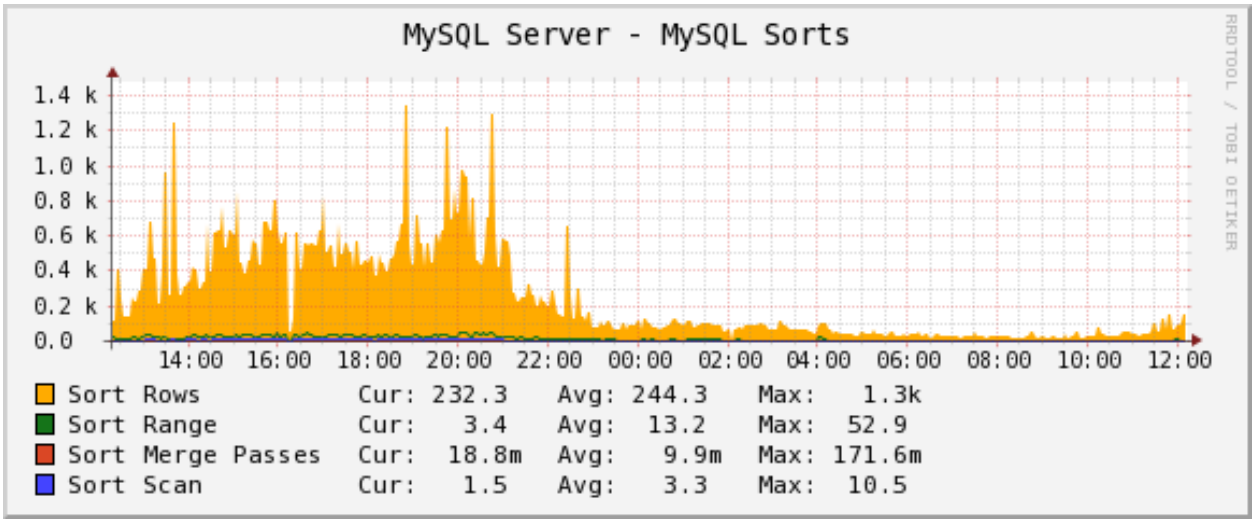
- By looking at SHOW SLAVE STATUS's Seconds_behind_master column, which is shown as Secs Behind Master
- By looking at a heartbeat table such as those supported by the `pt-heartbeat` tool in Percona Toolkit. You must configure the `ss_get_mysql_stats.php` file to do this.

When replication is running, there is an AREA of the same size as the replication delay, colored green. When it's stopped, there's an AREA of the same size as the replication delay, colored red. What this means is that you'll see a graph of replication delay, colored in with the appropriate color (green or red) to indicate whether replication was stopped at that moment. If replication isn't delayed, you won't see any green or red. If you're using Seconds_behind_master instead of `pt-heartbeat` to measure delay, it's impossible to measure delay when the slave is stopped, so you won't see any red. This is one of the reasons Seconds_behind_master from SHOW SLAVE STATUS is not as useful as `pt-heartbeat`.

The graph also shows open temporary tables and retried transactions.

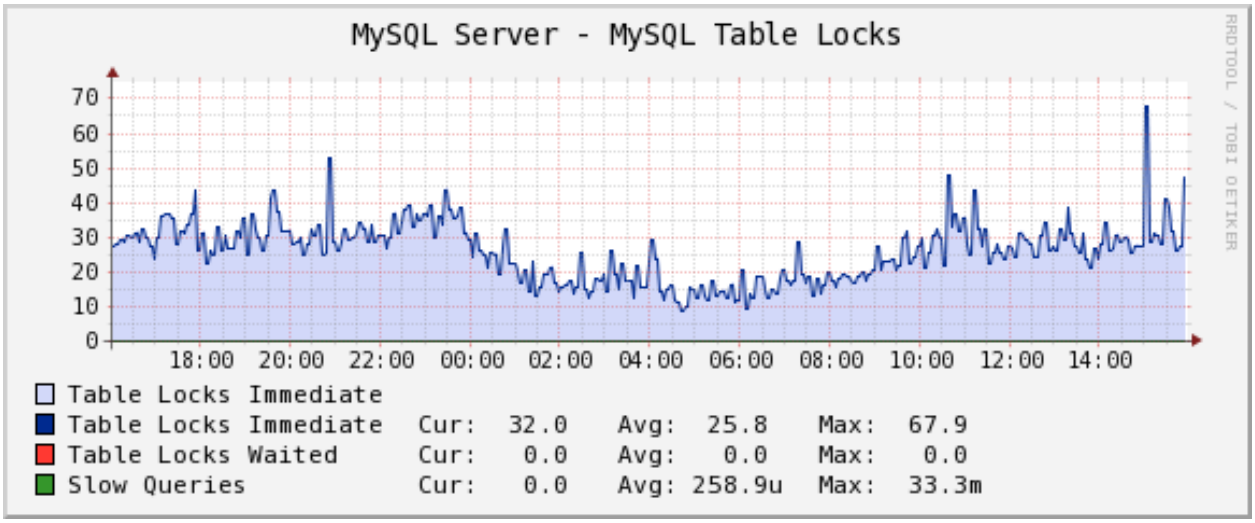


The MySQL Select Types graph shows information on how many of each type of select the MySQL server has performed: full join, full range join, range, range check, and scan. Like the Handler graphs, these show different types of execution plans, so any changes should be investigated. You should strive to have zero Select_full_join queries!

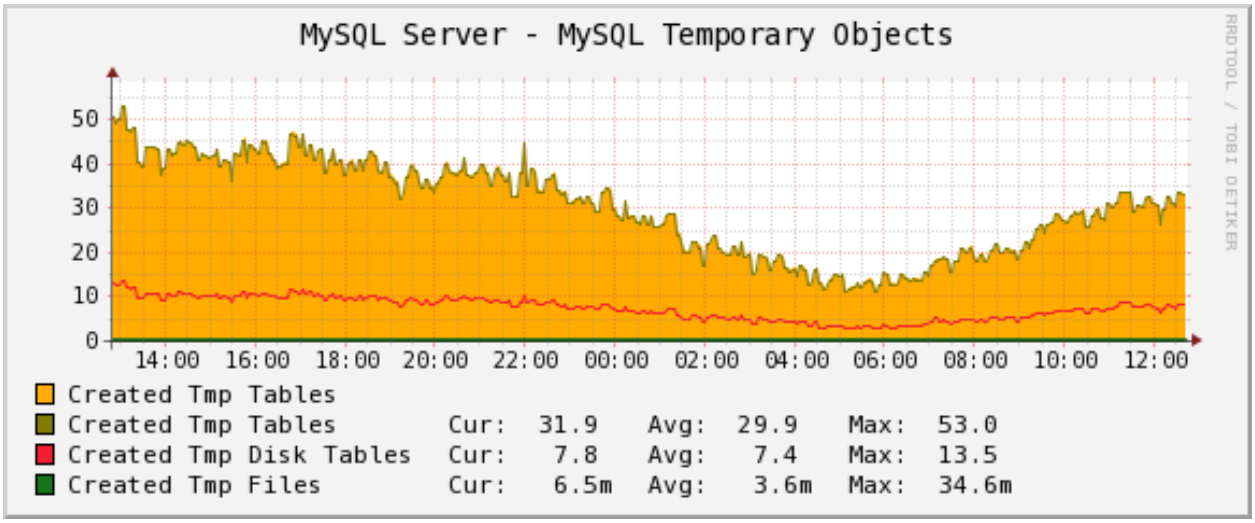


The MySQL Sorts graph shows information about MySQL sort operations: rows sorted, merge passes, and number of sorts triggered by range and scan queries. It is easy to over-analyze this data. It is not useful as a way to determine

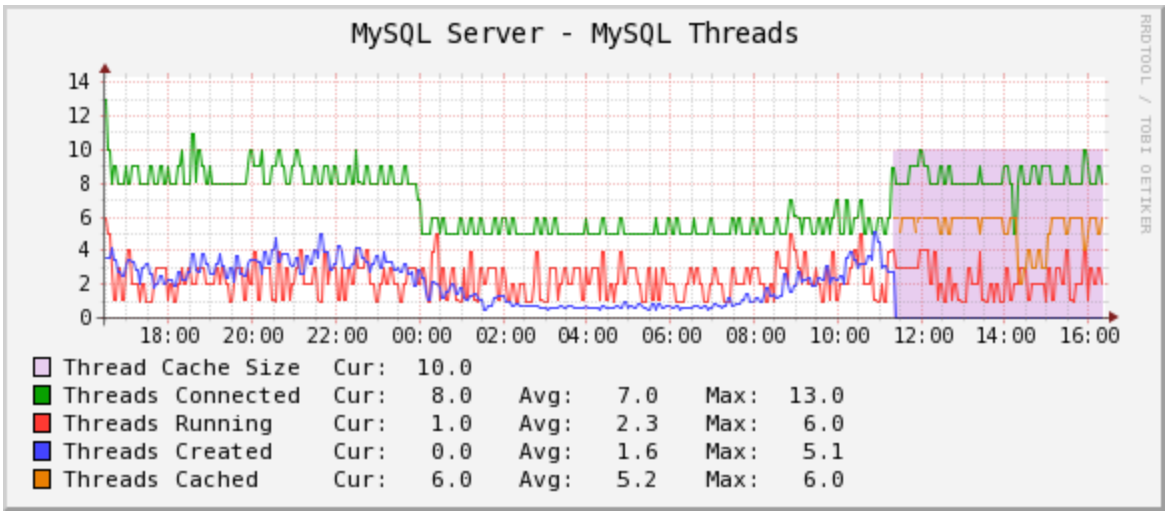
whether the server configuration needs to be changed.



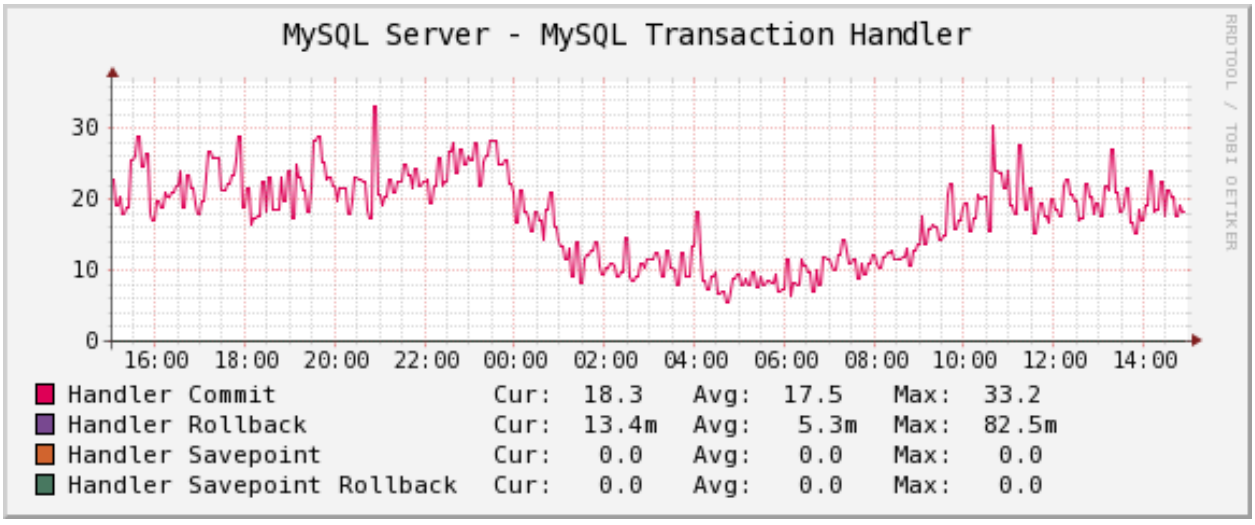
The MySQL Table Locks graph shows information about table-level lock operations inside MySQL: locks waited, locks granted without waiting, and slow queries. Locks that have to wait are generally caused by MyISAM tables. Even InnoDB tables will cause locks to be acquired, but they will generally be released right away and no waiting will occur.



The MySQL Temporary Objects graph shows information about temporary objects created by the MySQL server: temporary tables, temporary files, and temporary tables created on disk instead of in memory. Like sort data, this is easy to over-analyze. The most serious one is the temp tables created on disk. Dealing with these is complex, but is covered well in the book *High Performance MySQL*.



The MySQL Threads graph shows the size of thread cache the server is configured with and the number of threads of each type. On this example we can observe that once thread_cache_size was set to 10, MySQL stopped creating new threads and started using the cached ones.



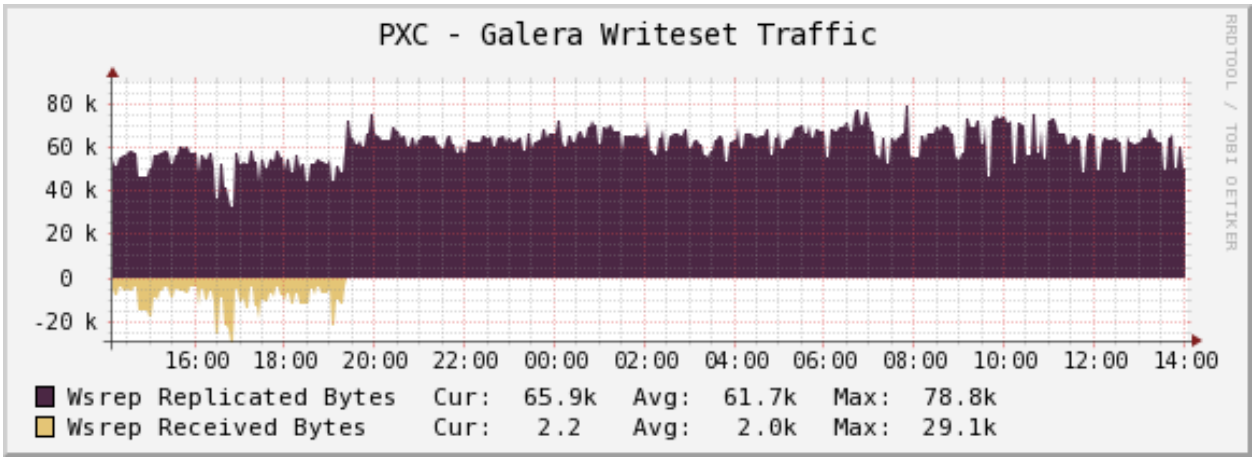
The MySQL Transaction Handler graph shows the transactional operations that took place at the MySQL server level.

Percona Galera/MySQL Monitoring Template for Cacti

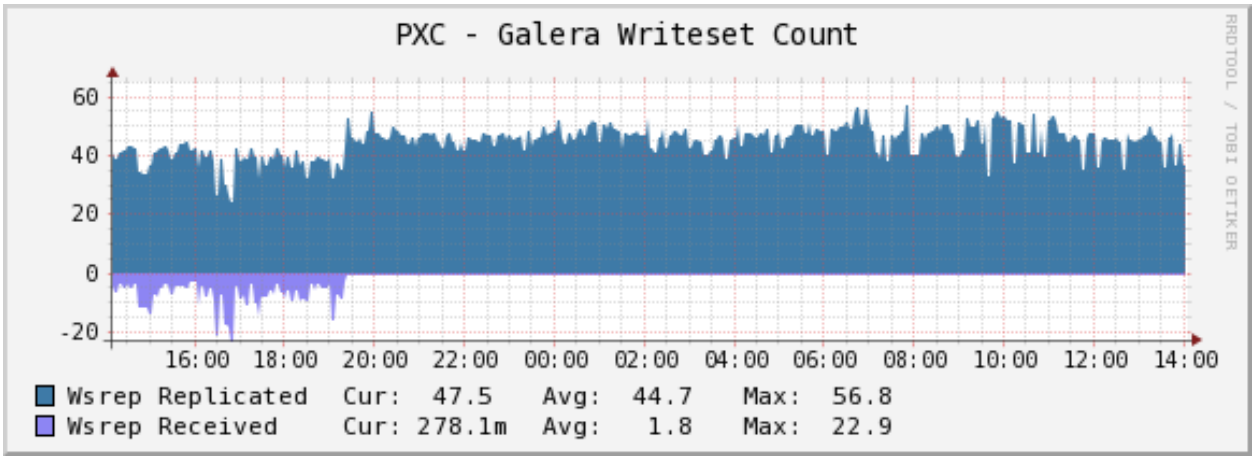
Galera/MySQL is synchronous multi-master cluster for InnoDB databases. Please refer to *MySQL Monitoring Template* for installation intructions.

Sample Graphs

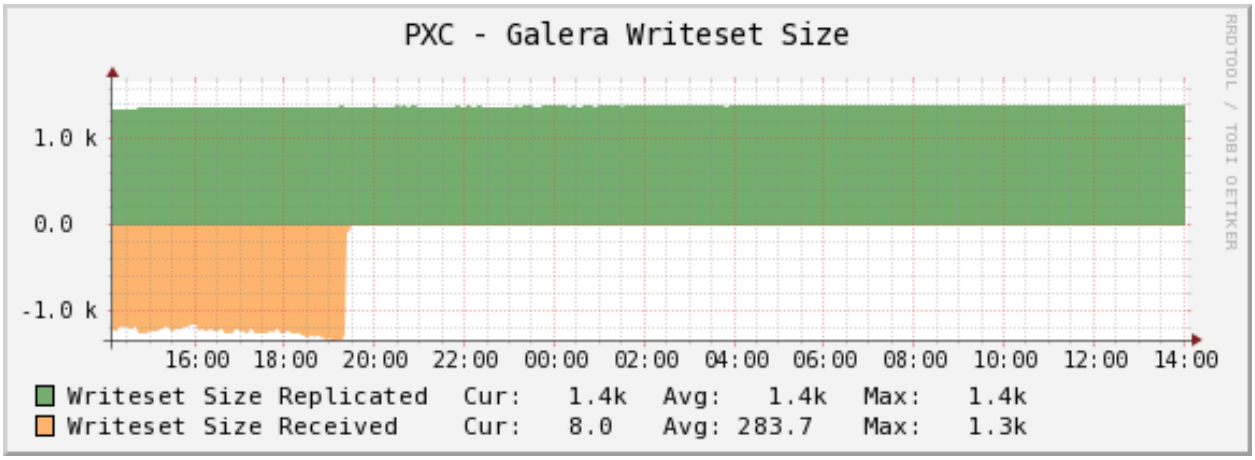
The following sample graphs demonstrate how the data is presented.



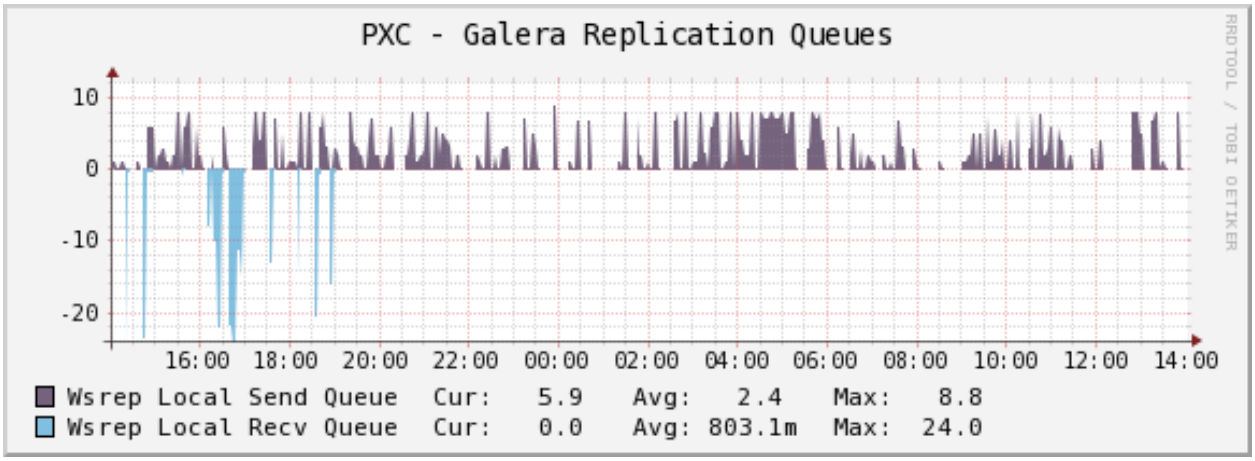
Galera Writeset Traffic. This graph shows the bytes of data replicated to the cluster (from this node) and received from the cluster (any other node).



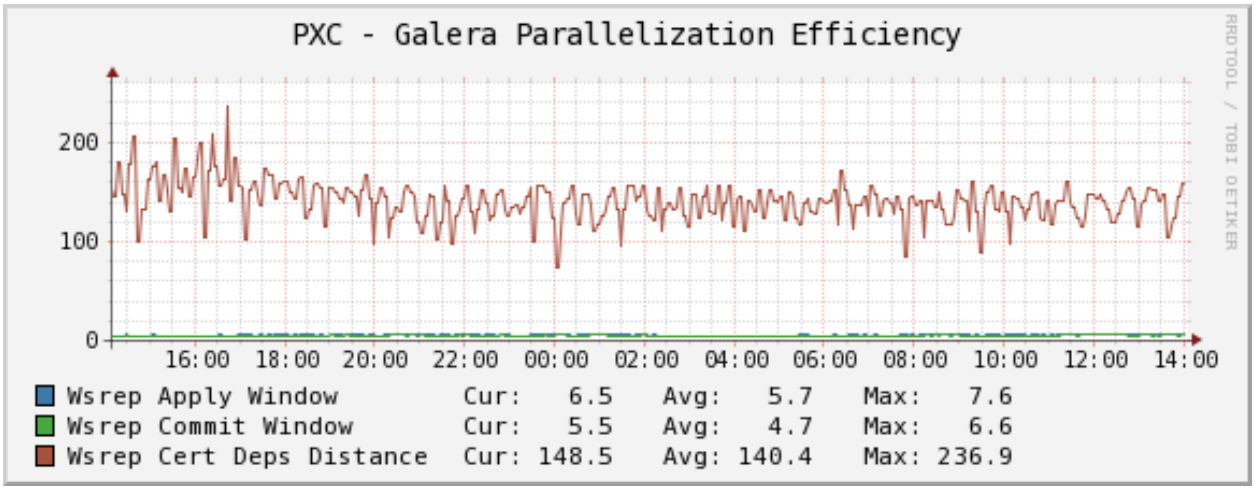
Galera Writeset Count. This graph shows the count of transactions replicated to the cluster (from this node) and received from cluster (any other node).



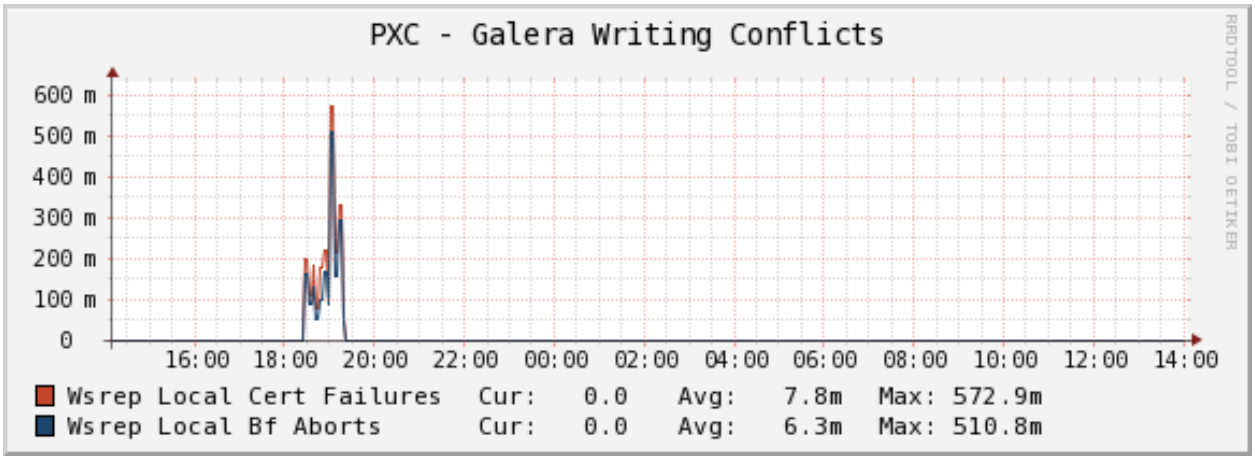
Galera Writeset Size. This graph shows the average transaction size sent/received.



Galera Replication Queues. The length of send and recv queues.

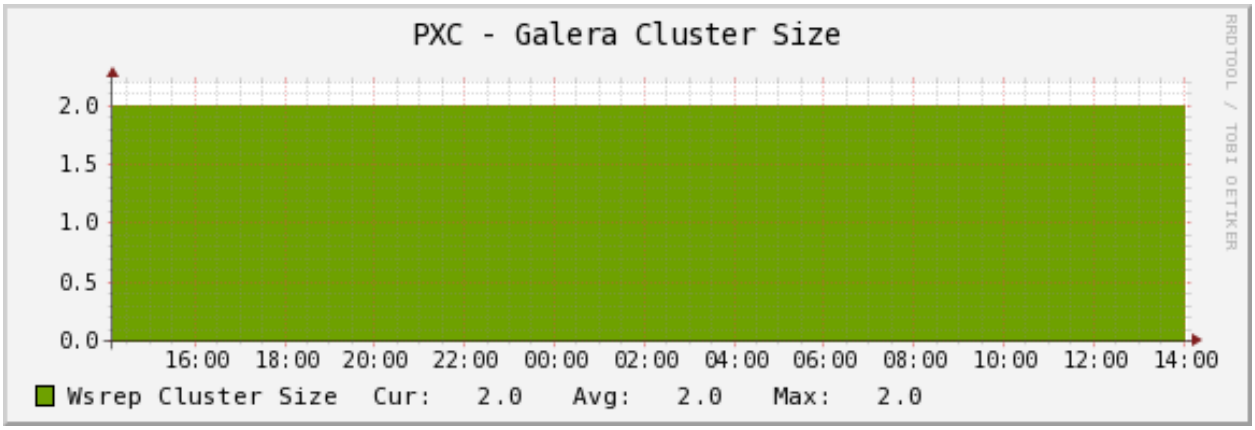


Galera Parallelization Efficiency. This graph shows the average distances between highest and lowest seqno that are concurrently applied, commited and can be possibly applied in parallel (potential degree of parallelization).

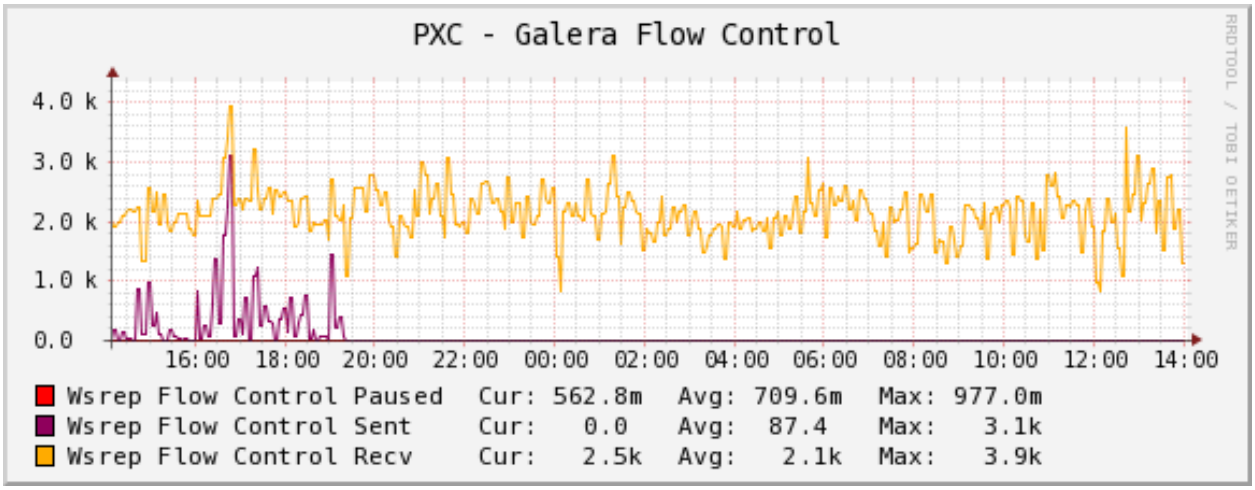


Galera Writing Conflicts. This graphs shows the number of local transactions being committed on this node that failed certification (some other node had a commit that conflicted with ours) – client received deadlock error on commit and also the number of local transactions in flight on this node that were aborted because they locked something an applier thread needed – deadlock error anywhere in an open transaction. The spike on the graph was created on the time of

writing to the same table potentially the same rows from 2 nodes.



Galera Cluster Size. This graph shows the number of members currently connected to the cluster.



Galera Flow Control. This graph shows the number of FC_PAUSE events sent/received. They are sent by a node when its replication queue gets too full. If a node is sending out FC messages it indicates the problem. On the graph you can observe FC sent/received when we were writing to both nodes and once we stopped writing to the current node, FC sent becomes 0 but still receiving FC messages from the neighbouring node.

IMPORTANT NOTE: the Flow Control graph can be faulty until this bug is fixed. The reason is the respective status variables are nullified on every `SHOW STATUS` query which means if something else runs it the Cacti script will see zeros right after that.

Installing SSH-Based Templates

This document explains how to prepare systems for graphing with the SSH-based scripts, which use only standard SSH and Unix commands to gather data from servers. The example server we will graph is 192.168.1.107.

The high-level process is as follows:

- Set up an SSH keypair for SSH authentication.
- Create a Unix user on each server you want to graph.
- Install the public key into that user's authorized_keys file.

- Install and configure the PHP file.
- Test the results.

List of Templates

Percona Apache Monitoring Template for Cacti

These templates use `ss_get_by_ssh.php` to connect to a server via SSH and extract statistics from the Apache server running there, by executing the `wget` program with the url `/server-status`.

Installation

Once the SSH connection is working, configure Apache to report its status. The typical setup is to enable status for 127.0.0.1 at the URL `/server-status`. If you decide to use a different URL, you'll have to configure that in the script configuration (covered in the general install guide) or pass a command-line option (also covered in the general install guide).

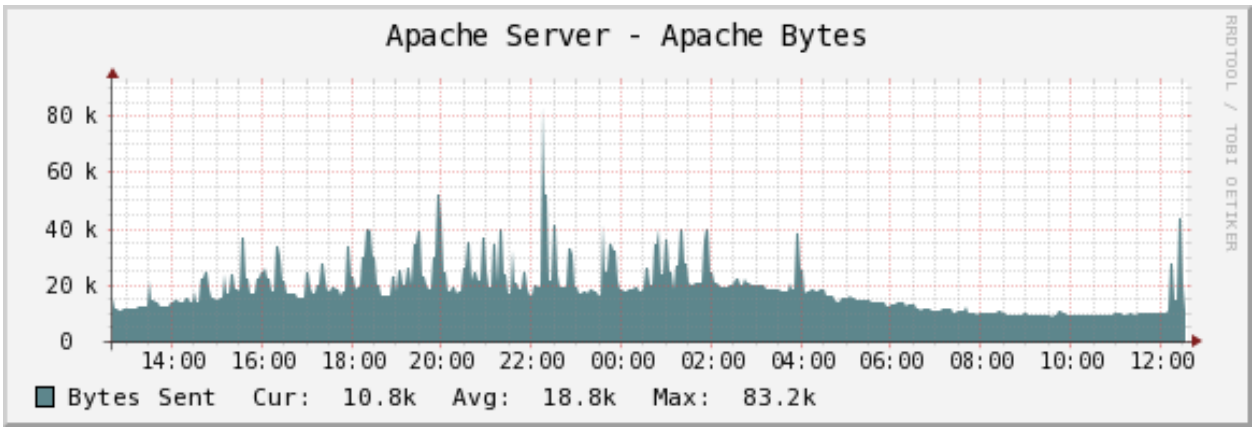
Be sure to configure Apache with the `ExtendedStatus On` directive so you get full status.

Finally, test one of your hosts like this. You may need to change some of the example values below, such as the cacti username and the hostname you're connecting to:

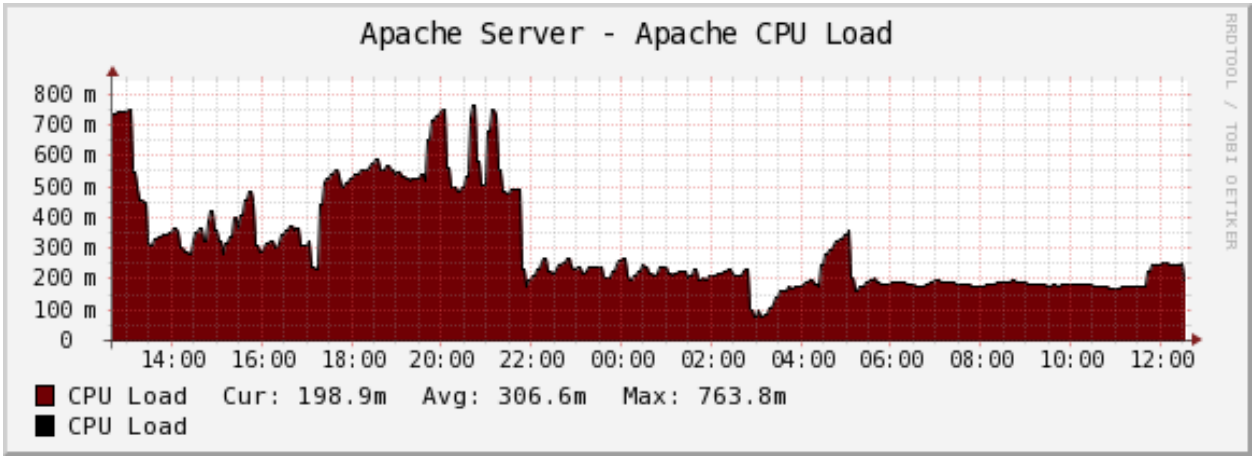
```
sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php --type apache --host 127.
↪0.0.1 --items gg,gh
```

Sample Graphs

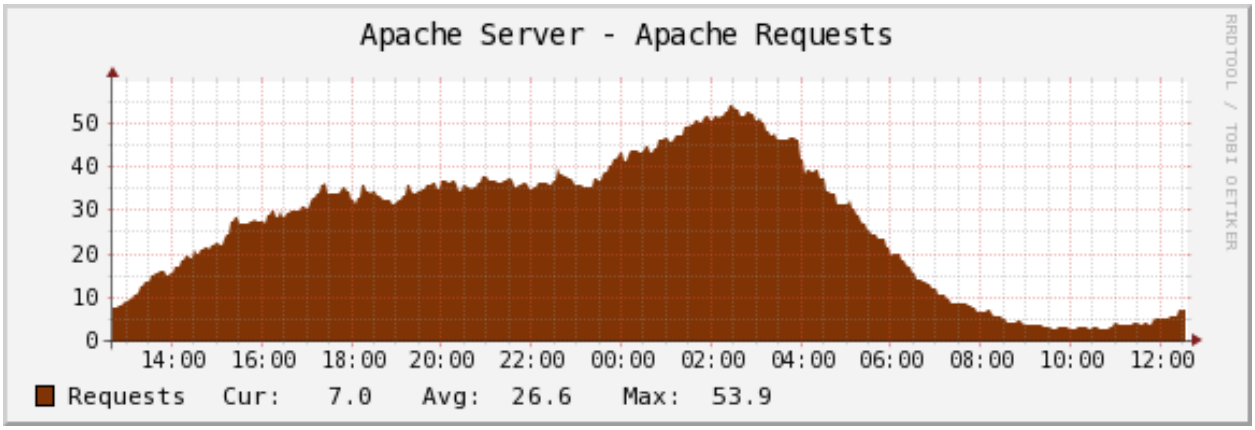
The following sample graphs demonstrate how the data is presented.



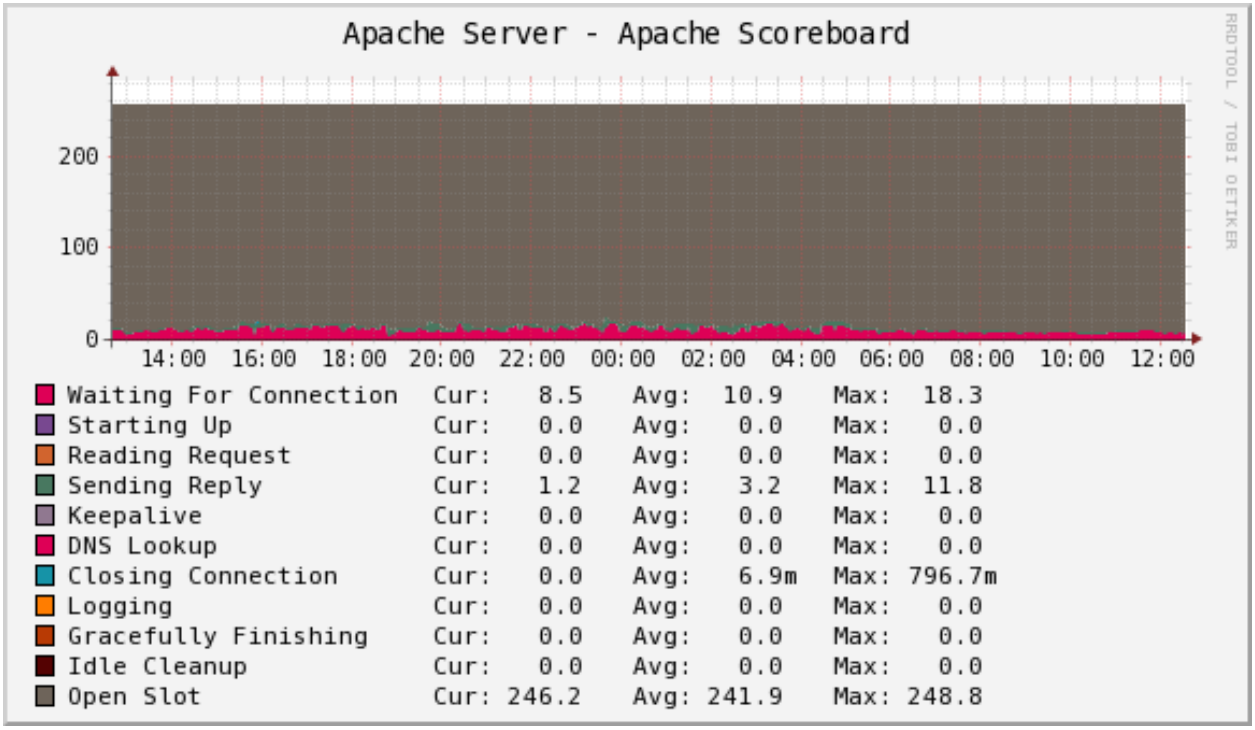
The number of bytes sent by Apache.



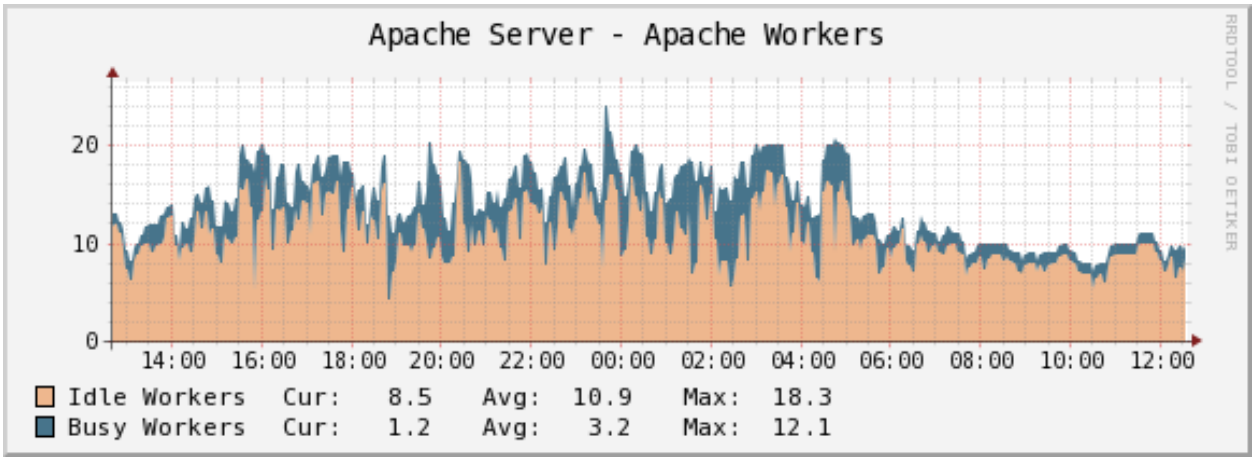
Apache's CPU usage.



The number of requests that Apache has handled.



The number of Apache processes in each of a variety of statuses.



The number of worker processes busy or idle at any given time.

Percona JMX Monitoring Template for Cacti

These templates use `ss_get_by_ssh.php` to connect to a server via SSH and extract statistics from the JMX server.

Installation

Once the SSH connection is working, you need to make the Java runtime information available through JMX by adding the JMX system properties when you start the program you want to monitor. For a normal Java program, you can learn more about this from

Monitoring and Management Using JMX. If you are using Tomcat, you can read Monitoring and Managing Tomcat. A simple example of how to start Notepad.jar with JMX instrumentation follows:

```
java -jar -Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=9012 \
-Dcom.sun.management.jmxremote.ssl=false \
-Dcom.sun.magement.jmxremote.authenticate=false \
/path/to/Notepad.jar
```

Now you need to install `ant` and the XML file with the JMX definitions. Copy `misc/jmx-monitor.xml` to the monitoring user's `$HOME` directory on the server you want to monitor. Then download `catalina-ant-jmx.jar`, which is part of Tomcat, to the `$HOME/.ant/lib` directory.

Before you test the Cacti script's functionality, test that the instrumentation is available to JMX. Run the following command on the host you want to monitor, from the Cacti user's home directory. Replace any values as needed:

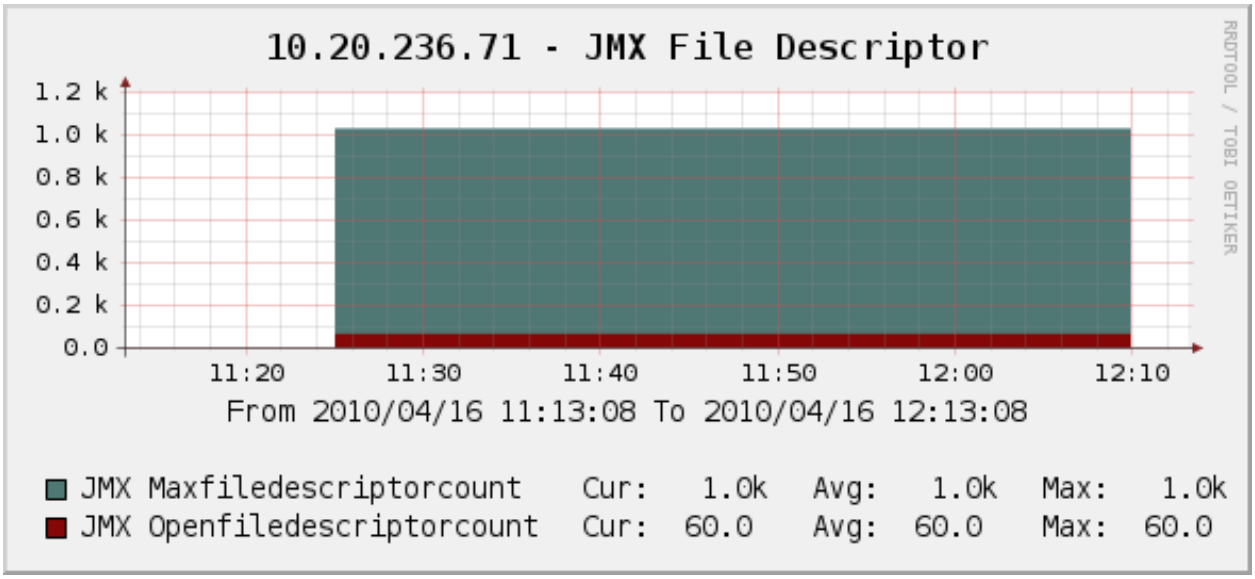
```
ant -Djmx.server.port=9012 -e -q -f jmx-monitor.xml
```

Now on the Cacti host, test a command similar to the following, replacing any values necessary with ones appropriate for your environment:

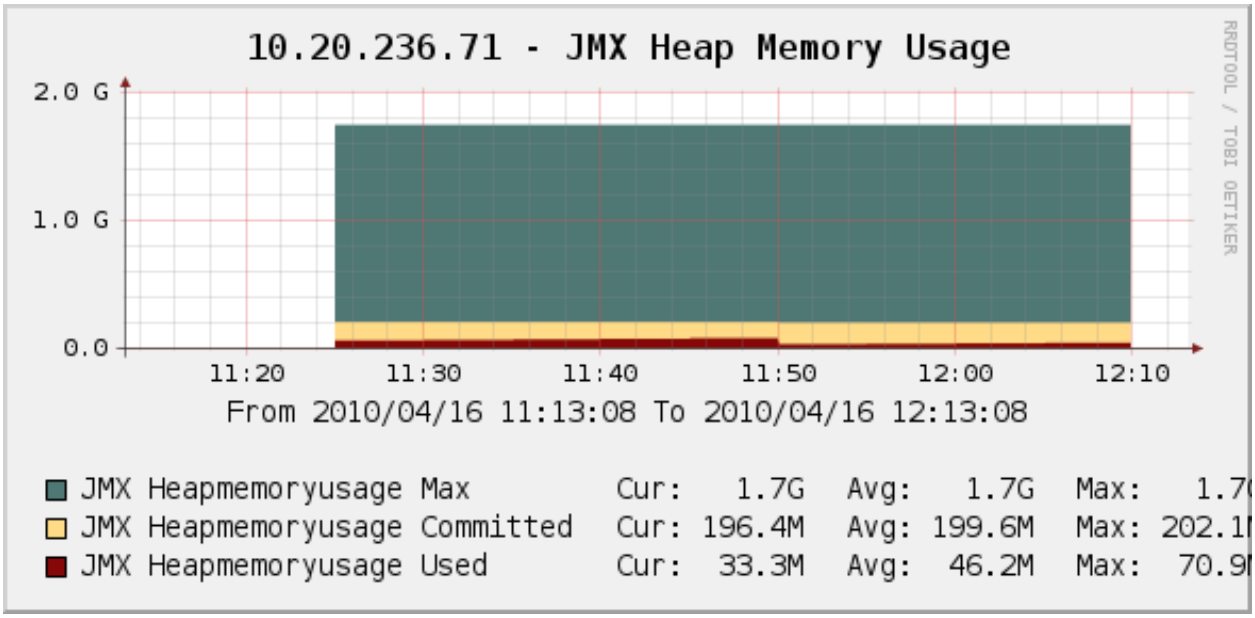
```
sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php --type jmx --host 127.0.
↪0.1 --items lt,lu
```

Sample Graphs

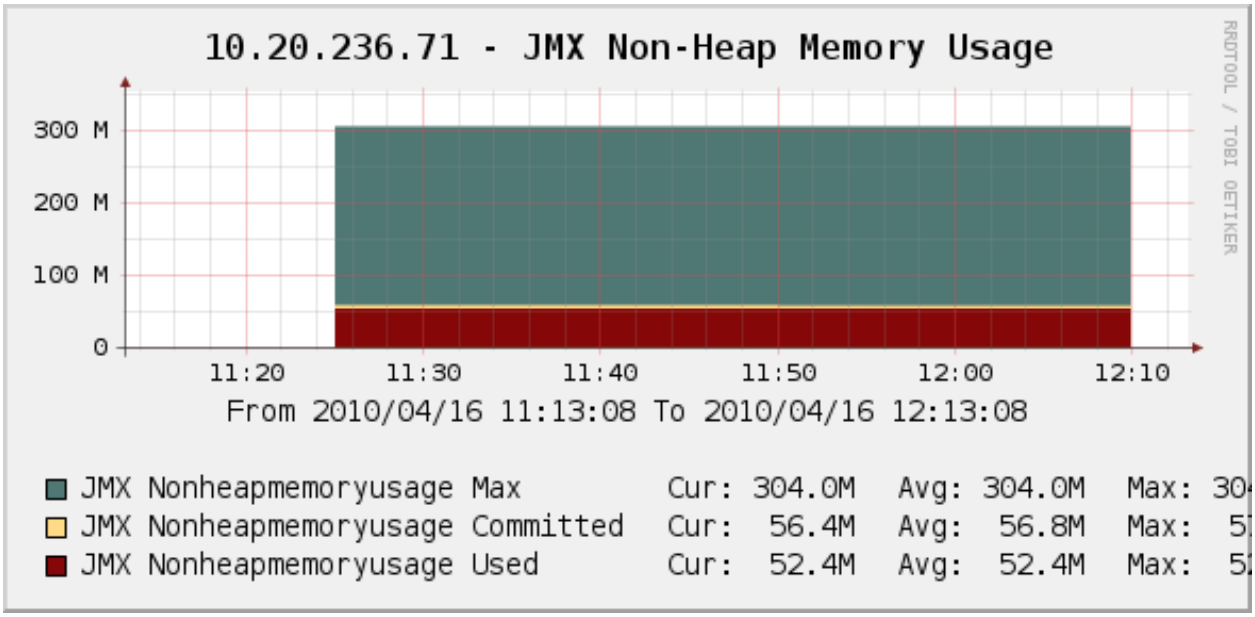
The following sample graphs demonstrate how the data is presented.



The file descriptors used by the JMX process.



The heap memory usage used by the JMX process.



The non-heap memory usage used by the JMX process.

Percona Linux Monitoring Template for Cacti

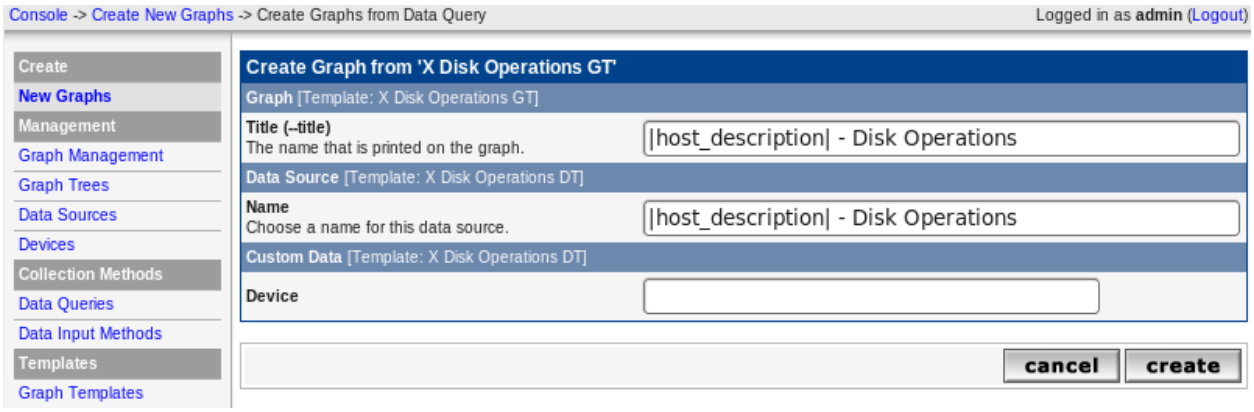
These templates use `ss_get_by_ssh.php` to connect to a server via SSH and extract standard metrics such as memory usage, number of users, and CPU usage. This is a good substitute for the standard kinds of system metrics one might graph via SNMP, when SNMP is not available or not desired.

Installation

Once the SSH connection is working, no special installation is necessary for most of the graphs.

The disk I/O graphs are special. Each graph requires that you specify the device you want to graph. For `/dev/sda`, for example, you should specify `sda`. Do not create the graphs through the normal host template method. Rather, add the graphs to the host manually, one at a time, by clicking "Create Graph" and selecting the desired graph template. Edit not only the device name in the command line, but the name of the graph and data input. Append the name of the device. This will make the items visually distinctive.

See the following screenshot for an example:



You should append `sda` in every textbox shown in that screenshot, if you want to monitor `/dev/sda`.

However, for "Disk Space" graph you have to specify the volume.

Device is a block device name as it appears in `/proc/diskstats` and not always seeing from `df` output especially when logical volumes are in use, e.g. `sda3`, `xvda1`, `cciss/c0d0p1`. It also can be verified from `lsblk` output. If `lsblk` is not available use `pvdisplay` from lvm2 package.

Volume is a filesystem absolute path as it appears under the first column of `df` output, e.g. `/dev/sda3`, `/dev/cciss/c0d0p1`, `/dev/mapper/vglocal01-mysql00`.

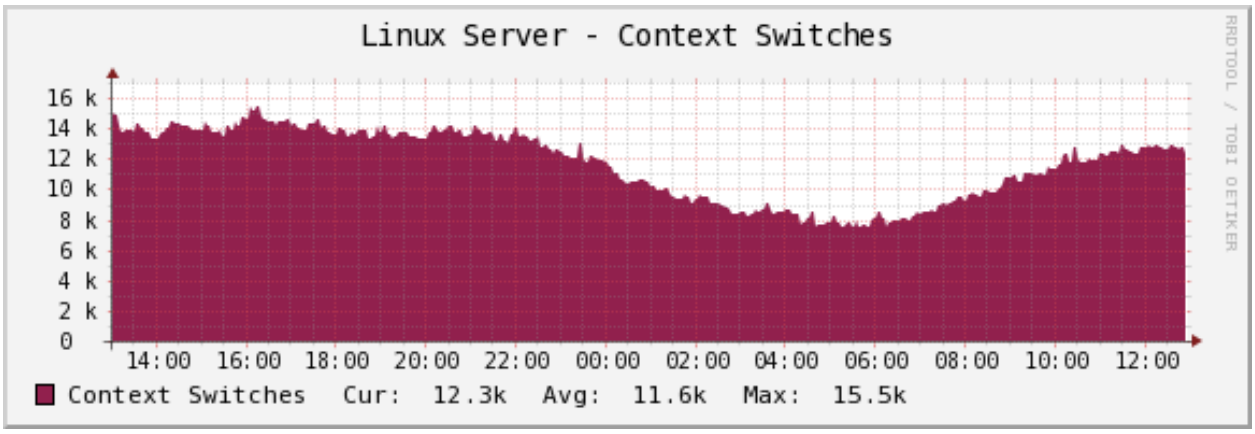
Example with logical volumes:

```
[root@localhost ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vglocal02-root00
                      271G  5.5G  252G   3% /
tmpfs                 127G     0  127G   0% /dev/shm
/dev/sda1             243M   32M  199M  14% /boot
/dev/mapper/vglocal02-tmp00
                      2.0G   68M  1.9G   4% /tmp
/dev/mapper/vglocal01-mysql00
                      700G  198G  503G  29% /mnt/mysql-fs
[root@localhost ~]# lsblk
NAME                                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                                   8:0    0 278.9G  0 disk
|-sda1                                8:1    0   250M  0 part /boot
`-sda2                                8:2    0 278.6G  0 part
  |-vglocal02-swap00 (dm-0)         253:0    0     2G  0 lvm  [SWAP]
  |-vglocal02-root00 (dm-1)         253:1    0 274.6G  0 lvm  /
  `-vglocal02-tmp00 (dm-3)          253:3    0     2G  0 lvm  /tmp
sdb                                   8:16   0   744G  0 disk
`-sdb1                                8:17   0   744G  0 part
  `-vglocal01-mysql00 (dm-2)        253:2    0   700G  0 lvm  /mnt/mysql-fs
```

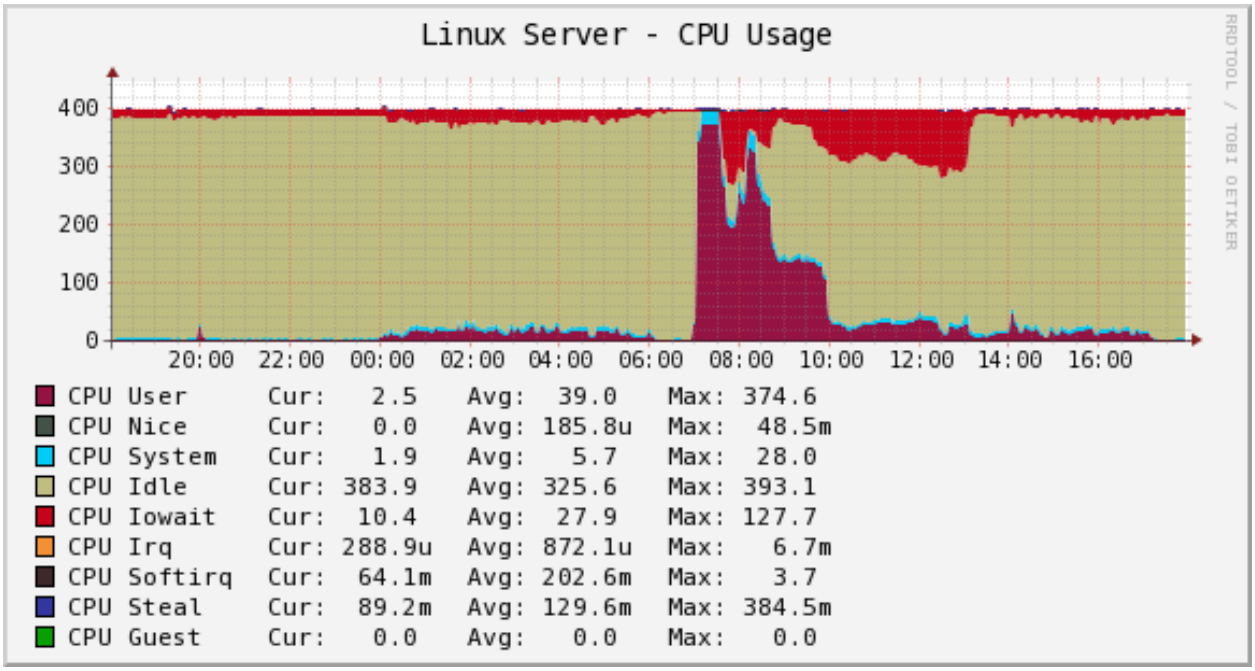
Device is `sdb1` and volume is `/dev/mapper/vglocal01-mysql00` in this example, if you want to monitor /mnt/mysql-fs.

Sample Graphs

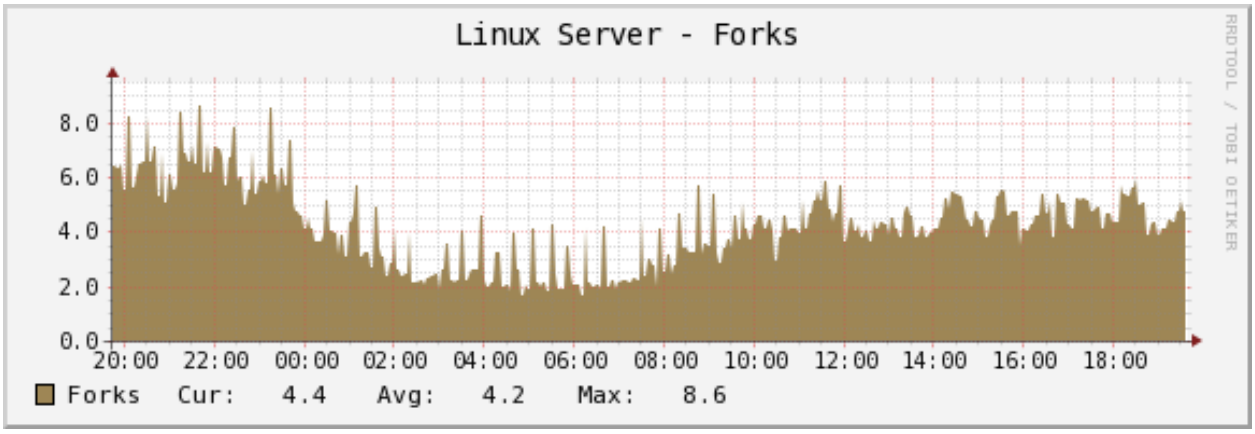
The following sample graphs demonstrate how the data is presented.



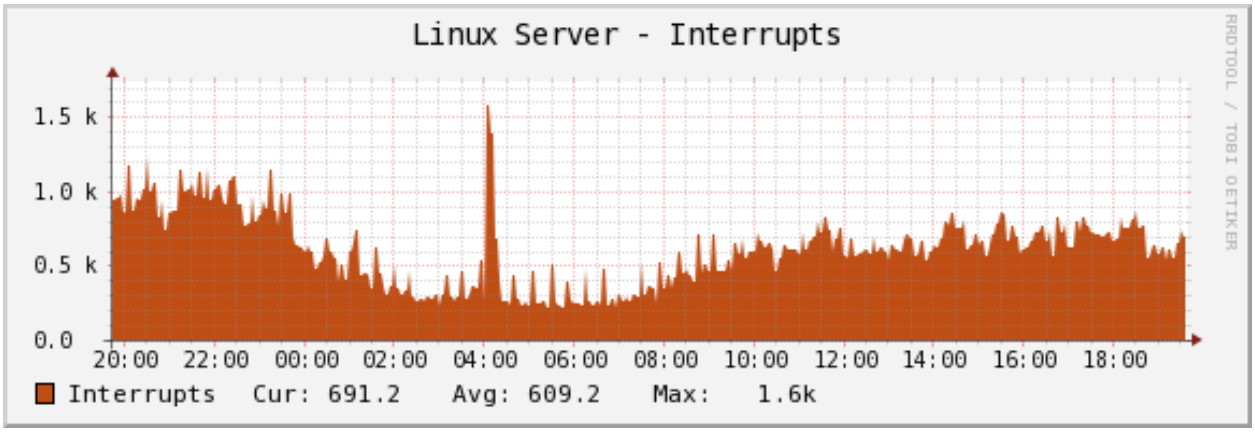
The number of context switches performed by the server.



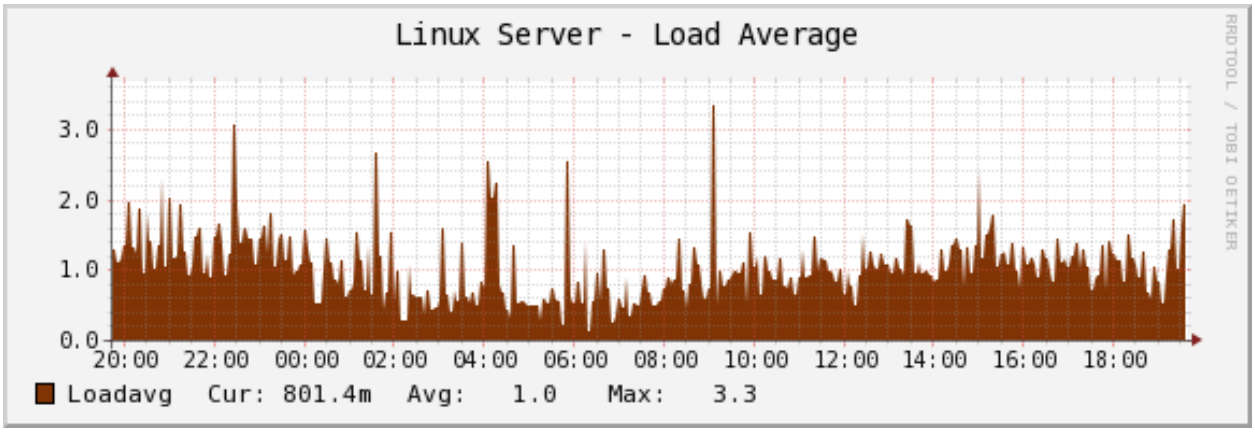
The CPU usage. The example shows a server having 1 processor with 2 cores, 4 threads, i.e. 4 virtual CPUs. The values will increase by 100 with each added virtual CPU.



The number of new processes created by the system.

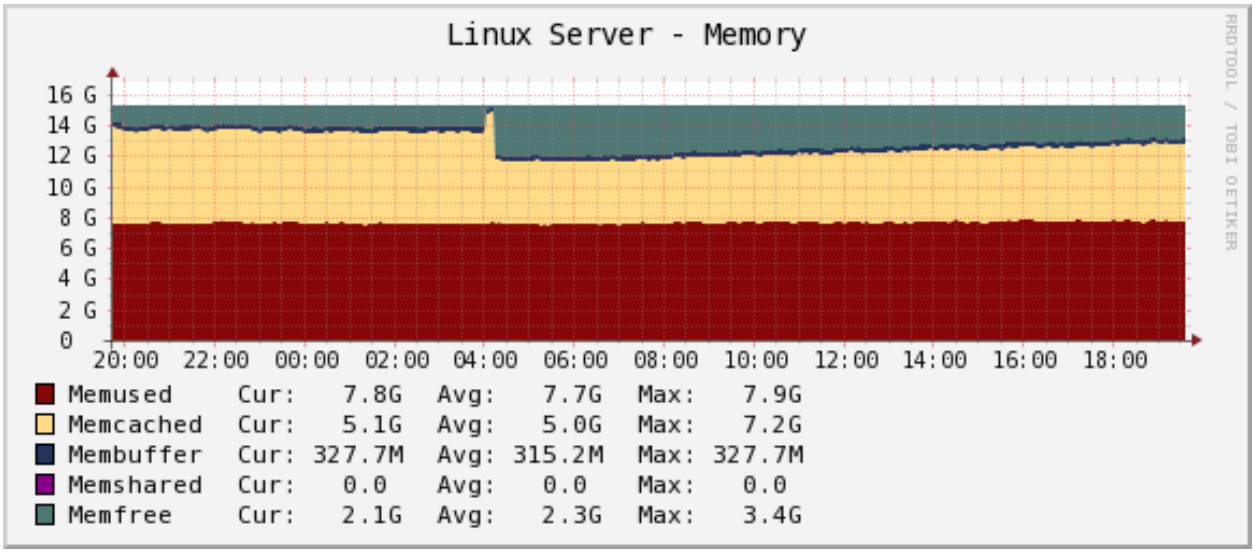


The interrupts the system handles.

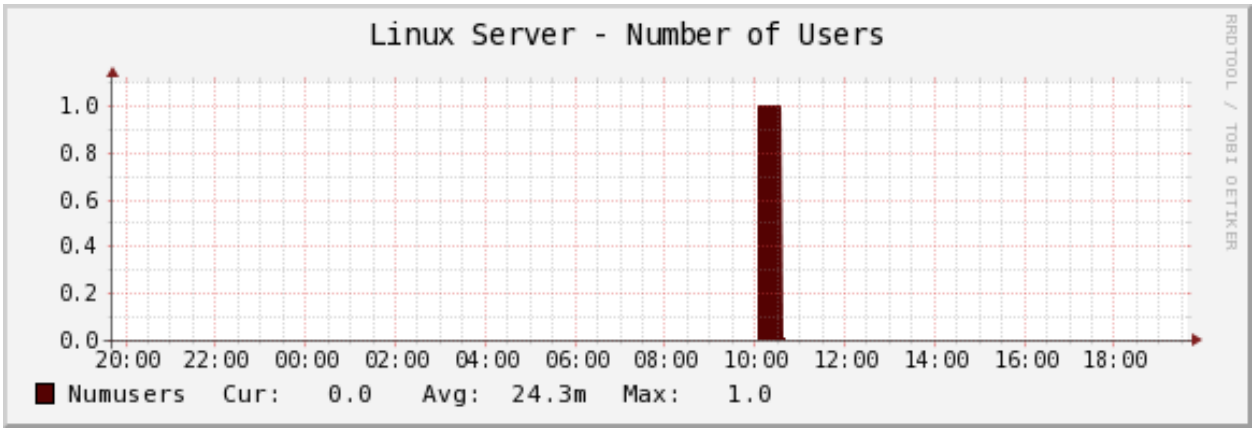


The system load average.

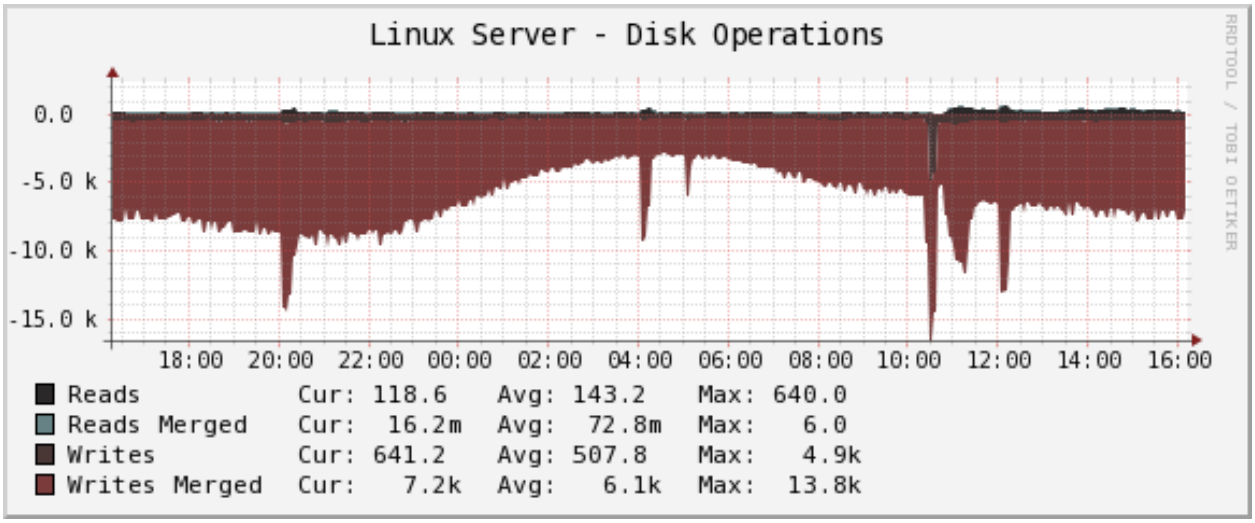
If you're used to Cacti's standard load average graph, you might think this one has less information. That is not true; the standard graph that comes with Cacti simply shows the same information averaged over three time intervals, which is redundant. RRDTool is natively capable of doing that.



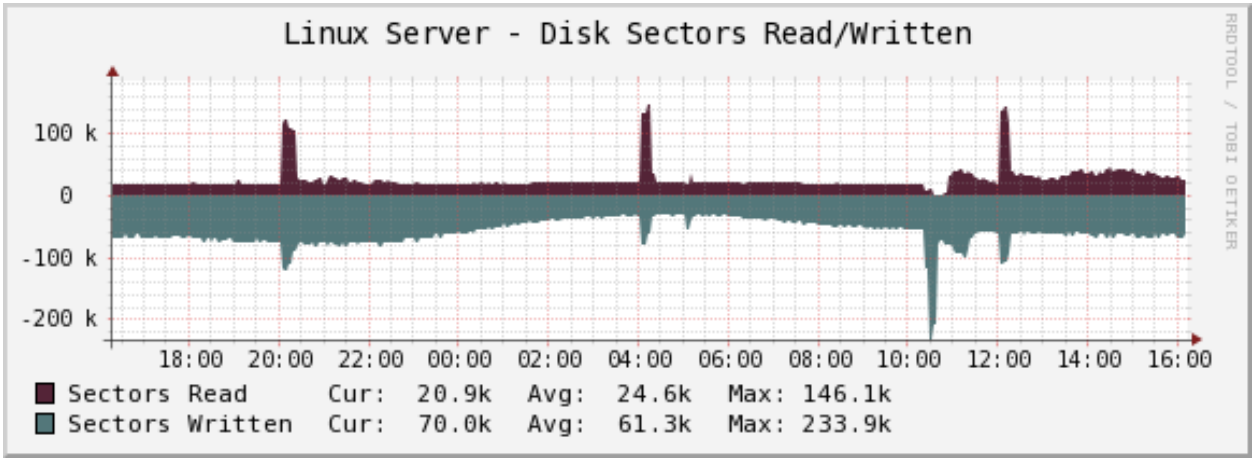
The system's memory usage.



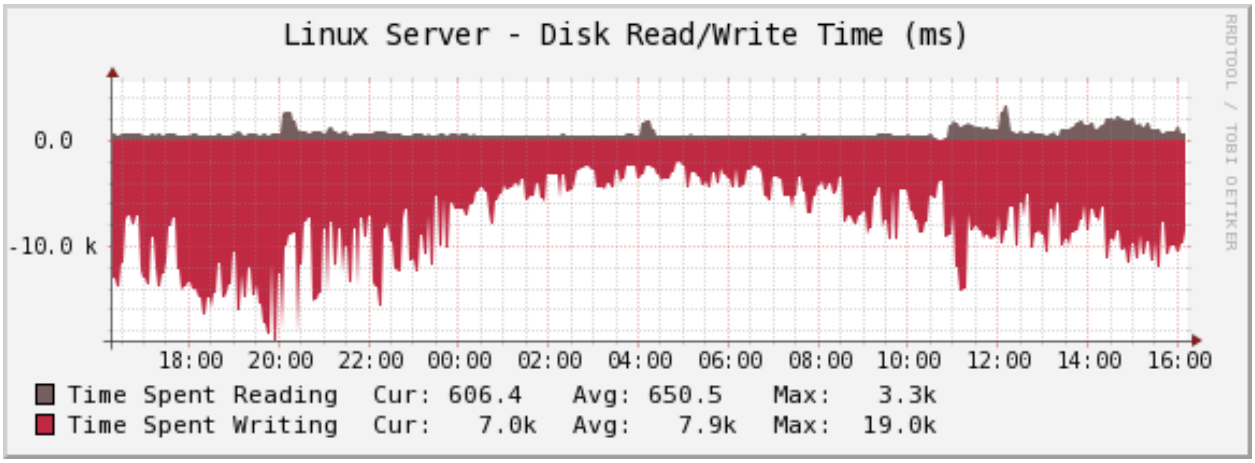
The number of users that were logged into the system, as reported by the "w" command.



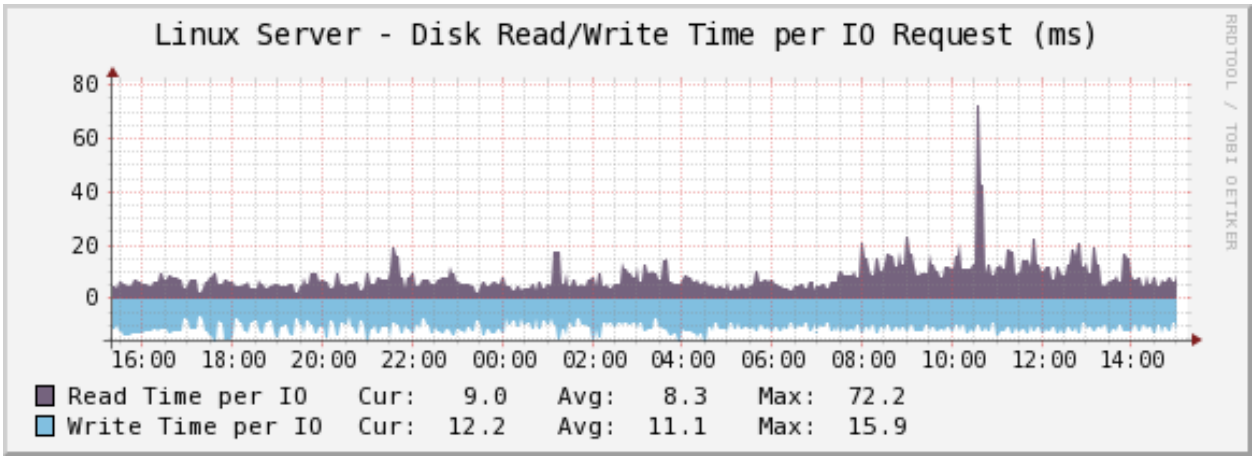
The number of read and write operations completed, and how many reads and writes were merged.



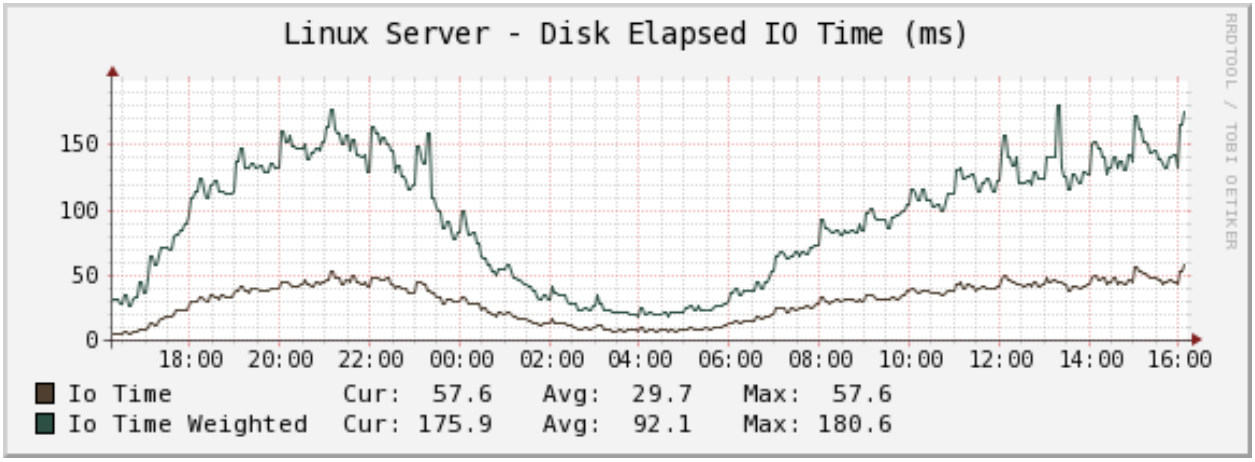
The number of disk sectors read and written.



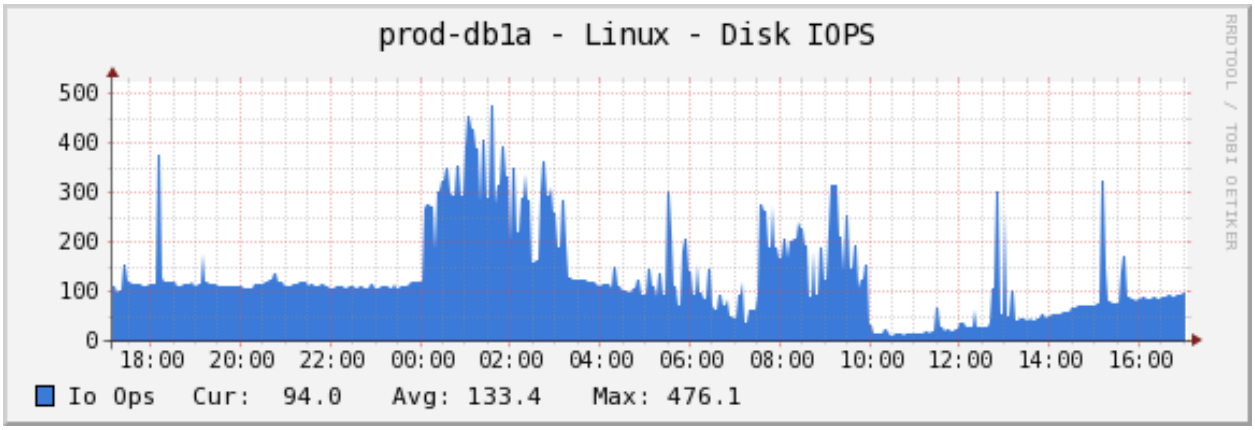
The amount of time spent reading and writing.



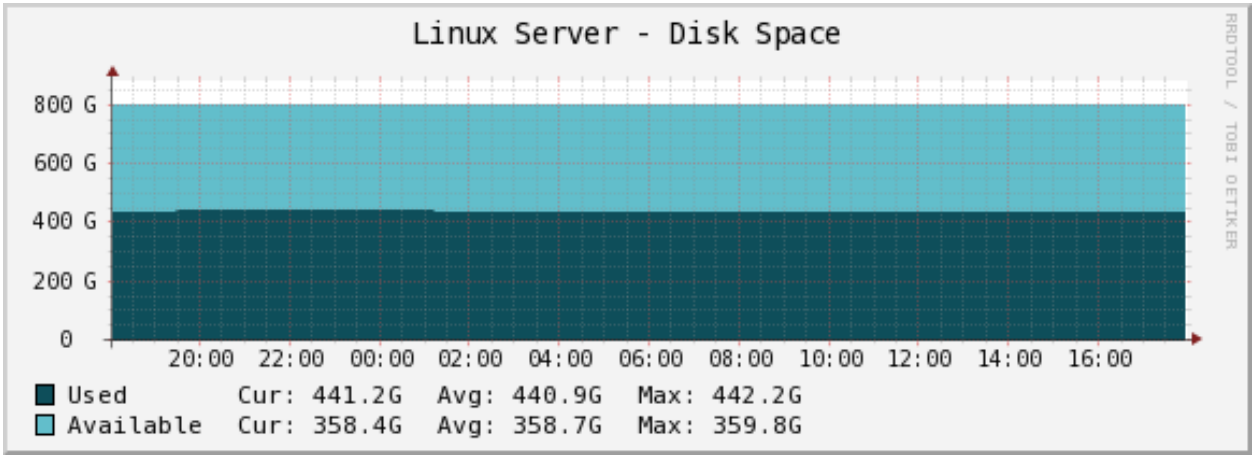
The amount of time spent reading and writing per 1 IO request.



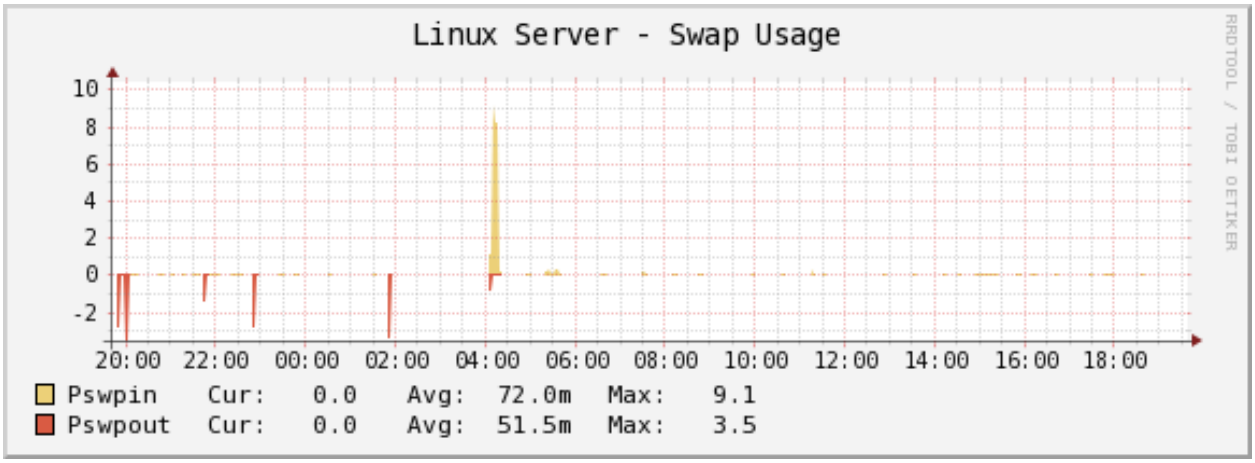
The disk utilization. This graph shows how much time was spent in disk I/O overall (busy time), and how much weighted time was spent doing disk I/O. The latter is a useful indication of I/O backlog. The weighted time is the number of requests multiplied by the busy time, so if there are 5 requests that take 1 second, it is 5 seconds. (If they all happen at the same time, the busy time is only 1 second.)



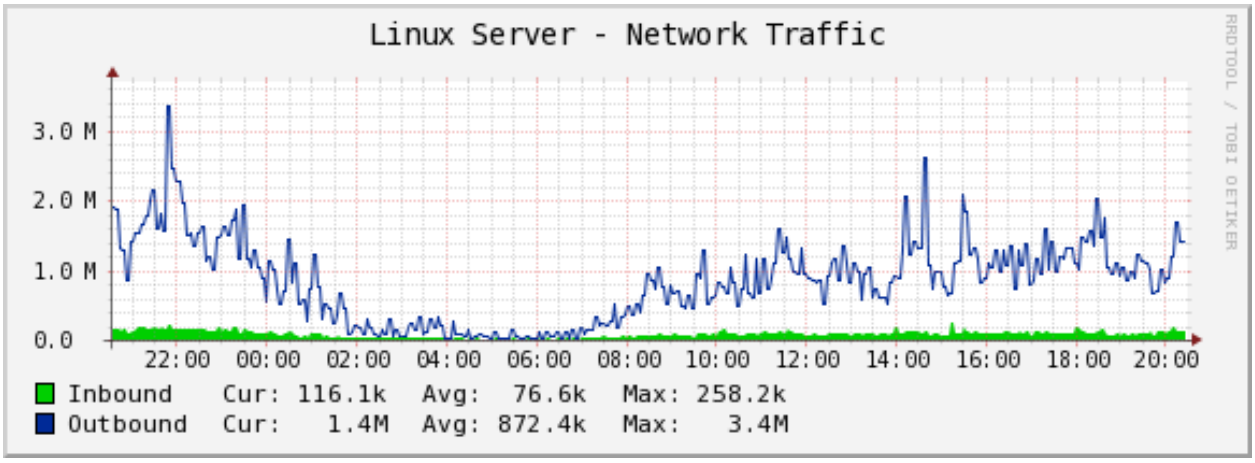
The number of disk IO operations per second. Actually, this is a sum of reads + writes.



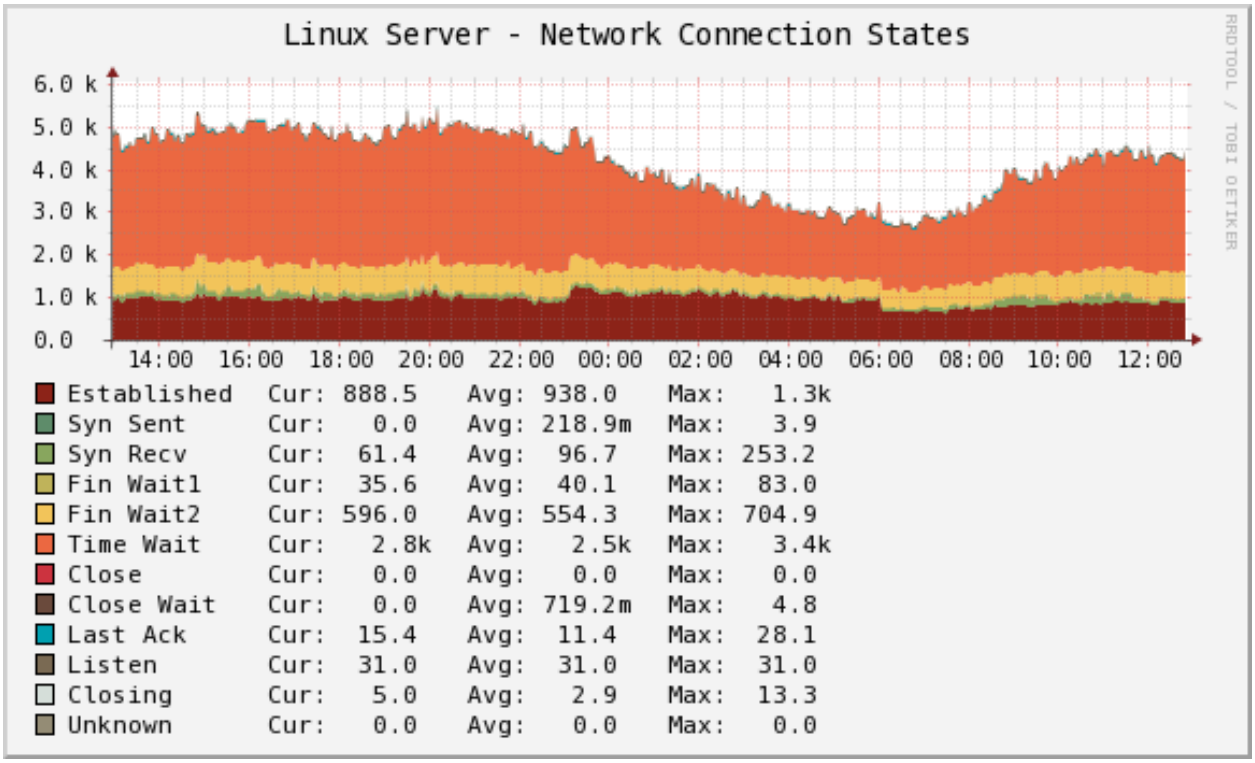
The disk space for the volume.



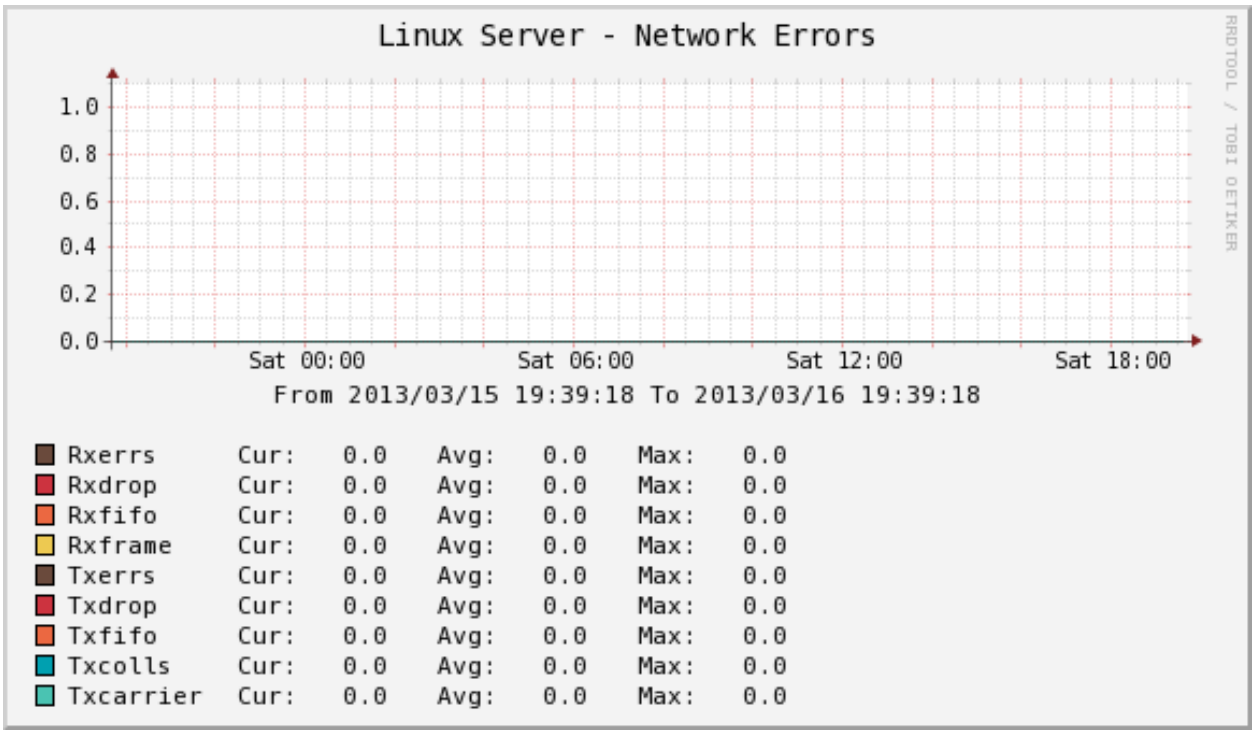
The swap usage of the system.



The network traffic for the adapter in bits/sec.



The network connection states for the adapter.



The network errors for the adapter.

Percona Memcached Monitoring Template for Cacti

These templates use `ss_get_by_ssh.php` to connect to a server via SSH and extract statistics from the memcached server running there, by executing the `nc` (netcat) program with the command "STAT". This means you don't need any memcached APIs installed. Standard Unix command-line tools are all you need.

Installation

Once the SSH connection is working, you need to test the memcached function. You may need to change some of the example values below, such as the cacti username and the hostname you're connecting to:

```
sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php --type memcached --host␣
↪127.0.0.1 --items ij,ik
```

You need `nc` on the server. Some versions of `nc` accept different command-line options. You can change the options used by configuring the PHP script. If you don't want to do this for some reason, then you can install a version of `nc` that conforms to the expectations coded in the script's default configuration instead.

On Debian/Ubuntu, `netcat-openbsd` does not work, so you need the `netcat-traditional` package, and you need to switch to `/bin/nc.traditional` with the following command:

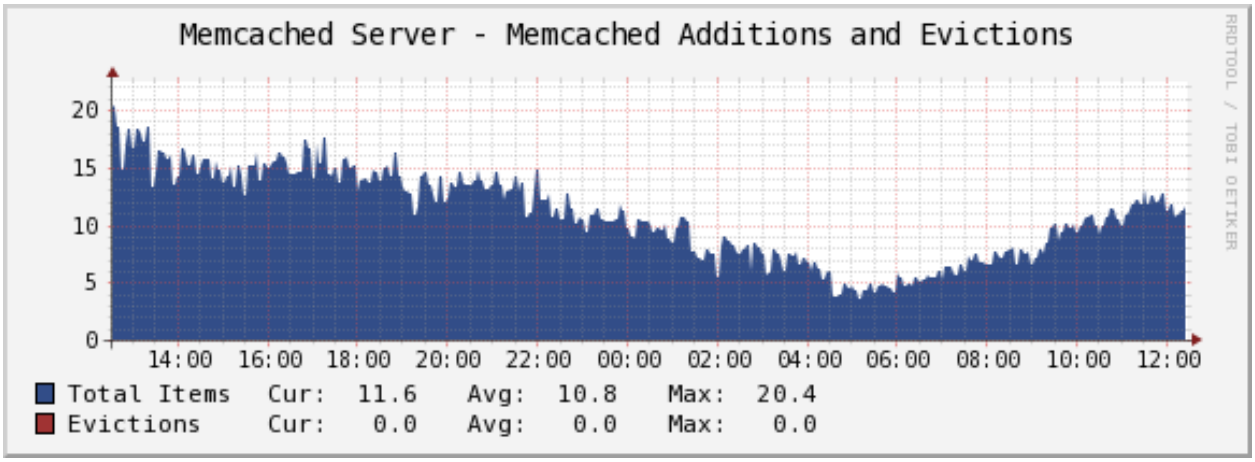
```
update-alternatives --config nc
```

Also for Debian re-define PHP variable in ss_get_by_ssh.php.cnf this way:

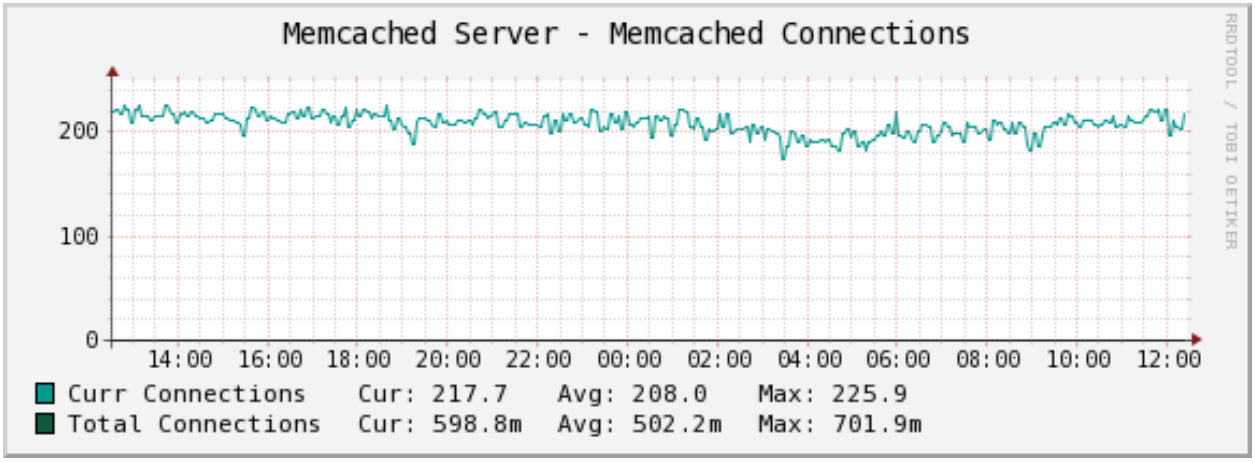
```
<?php
$nc_cmd = 'nc -q1';
```

Sample Graphs

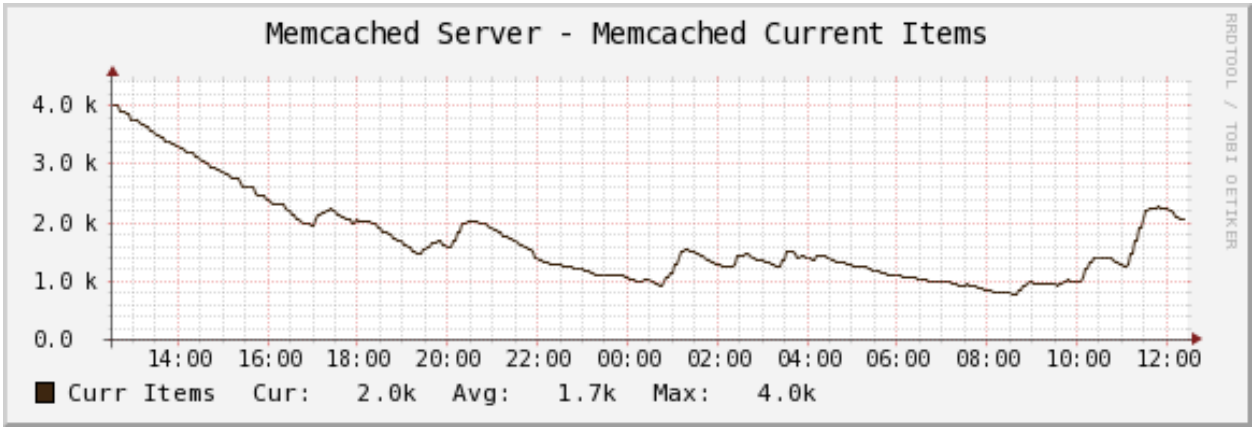
The following sample graphs demonstrate how the data is presented.



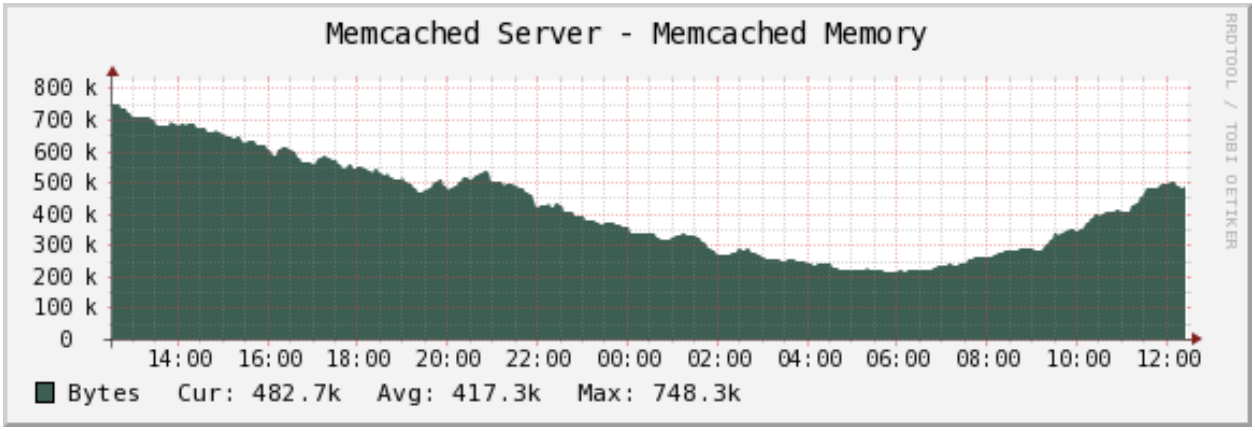
Shows how many items were added and evicted.



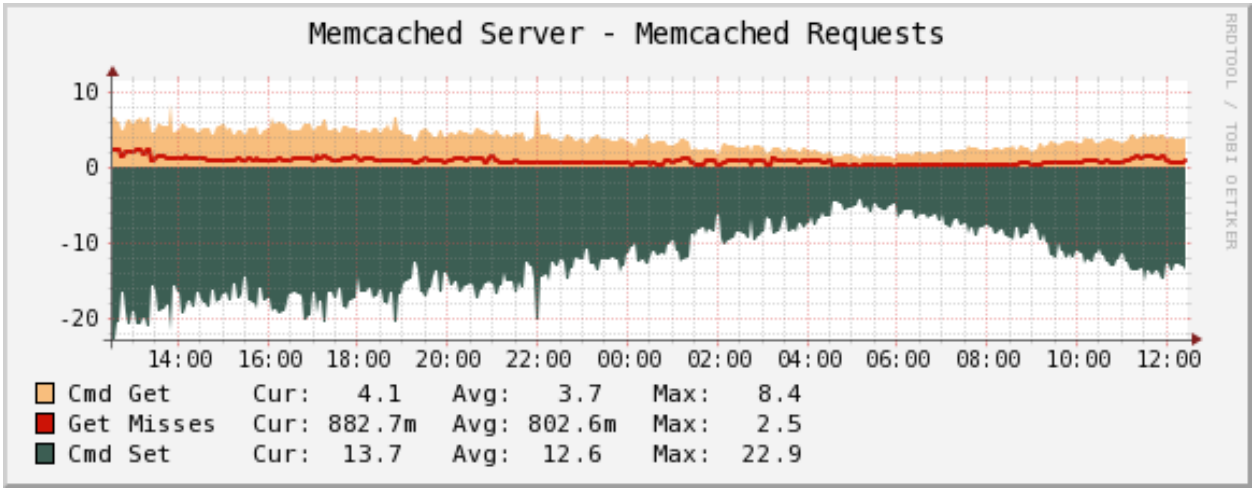
Shows how many connections have been made.



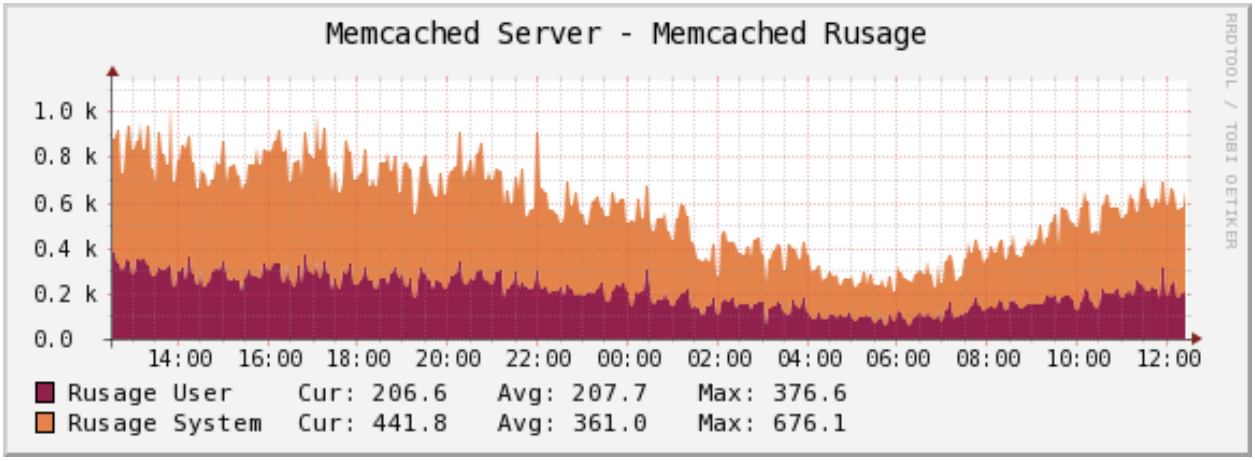
Shows how many items are stored in the server.



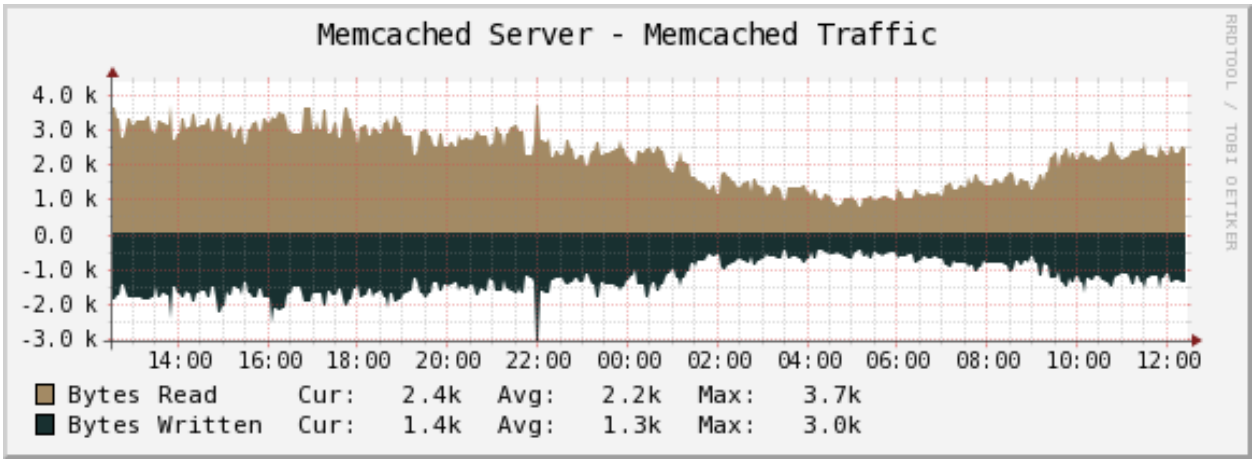
Shows how much memory the server is using.



Shows how many gets and sets have happened, as well as how many of the gets were misses (there was no item in the cache).



Shows the resource usage statistics reported by memcached, in system and user CPU time.



Shows the network traffic in and out of the memcached server.

Percona MongoDB Monitoring Template for Cacti

These templates use `ss_get_by_ssh.php` to connect to a server via SSH and extract statistics from the MongoDB server running there, by executing the `serverStatus` admin command from the MongoDB shell. This means that the `mongo` CLI needs to be in `$PATH` and you must be running version 1.2 or newer of MongoDB.

Installation

Once the SSH connection is working, confirm that you can login to MongoDB from with the "mongo" cli tool. From this tool, confirm that the `serverStatus` command is present by running:

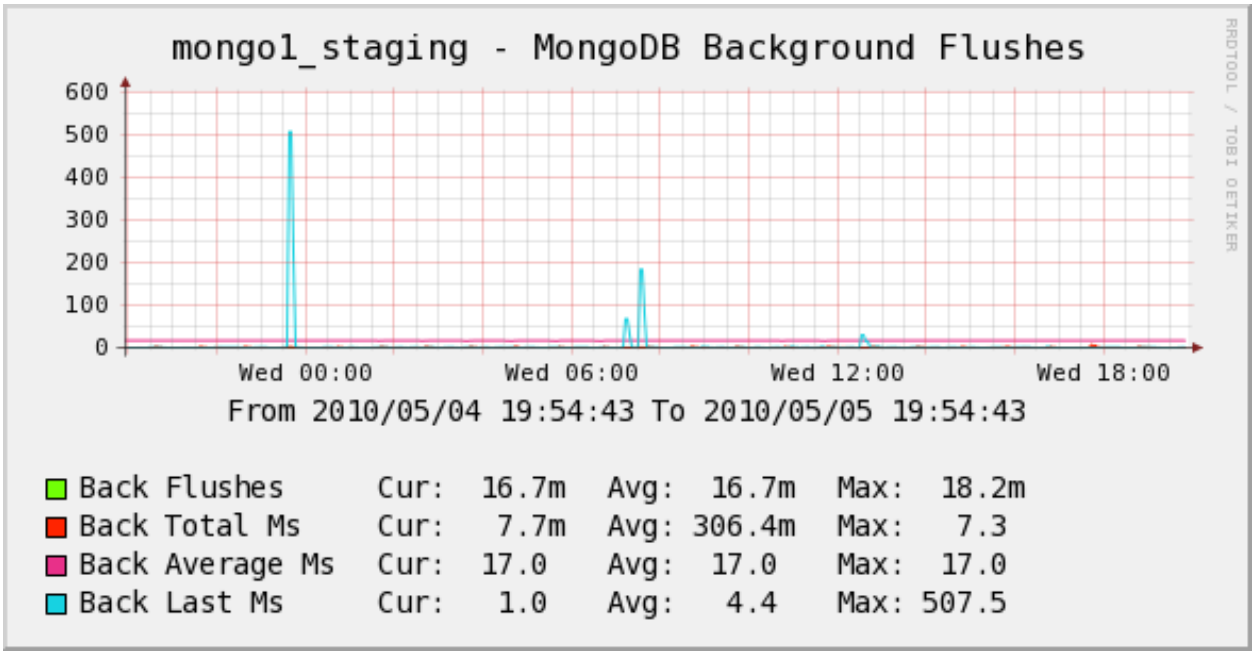
```
db._adminCommand({serverStatus : 1});
```

This should produce quite a bit of output. With all of this confirmed, test one of your hosts with the command below. You may need to change some of the example values below, such as the cacti username and the hostname you're connecting to:

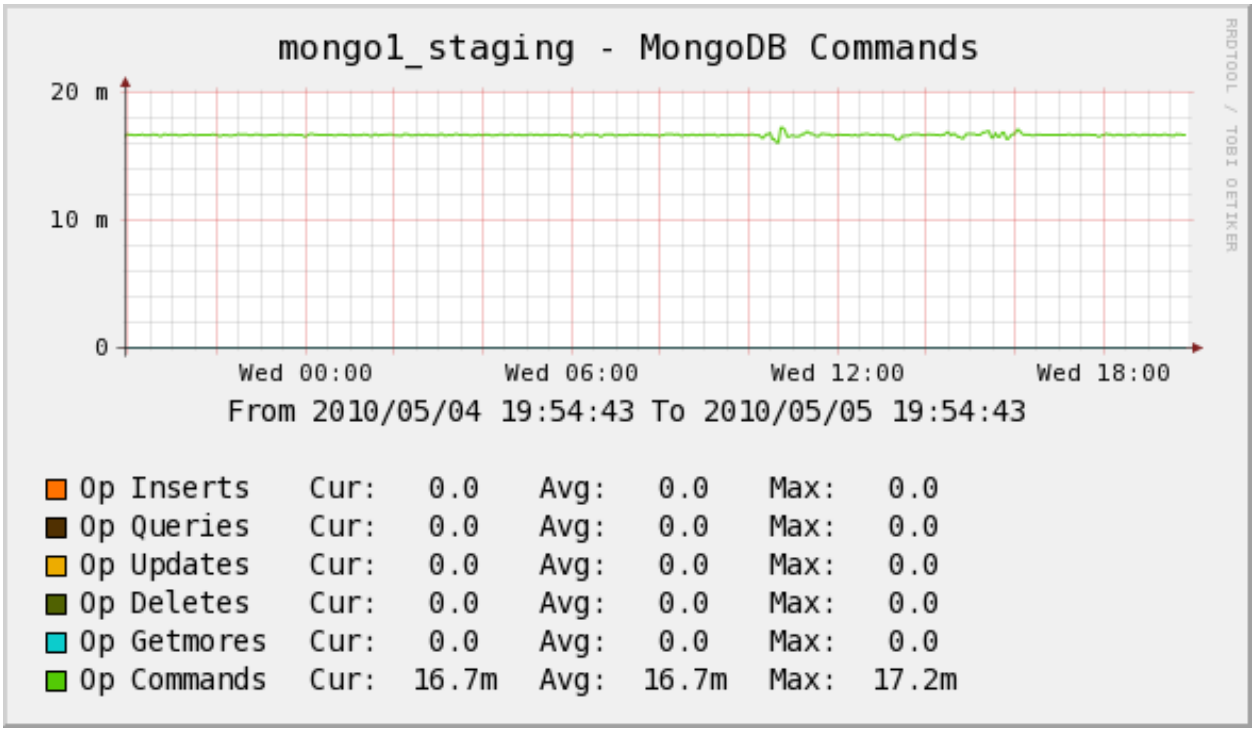
```
sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php --type mongodb --host␣
↪127.0.0.1 --items mk,ml
```

Sample Graphs

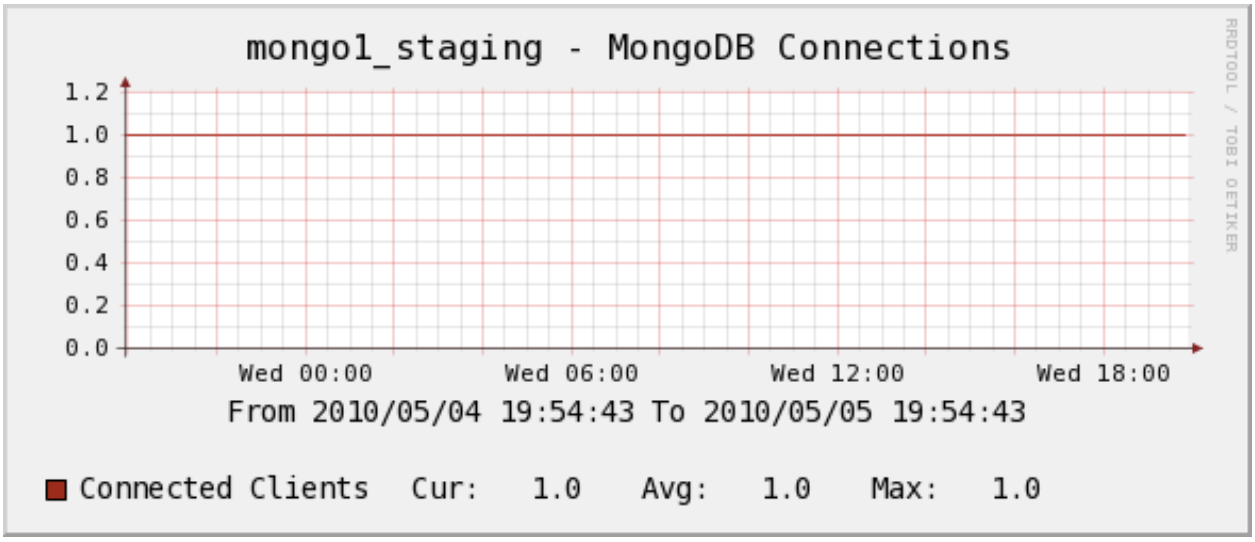
The following sample graphs demonstrate how the data is presented.



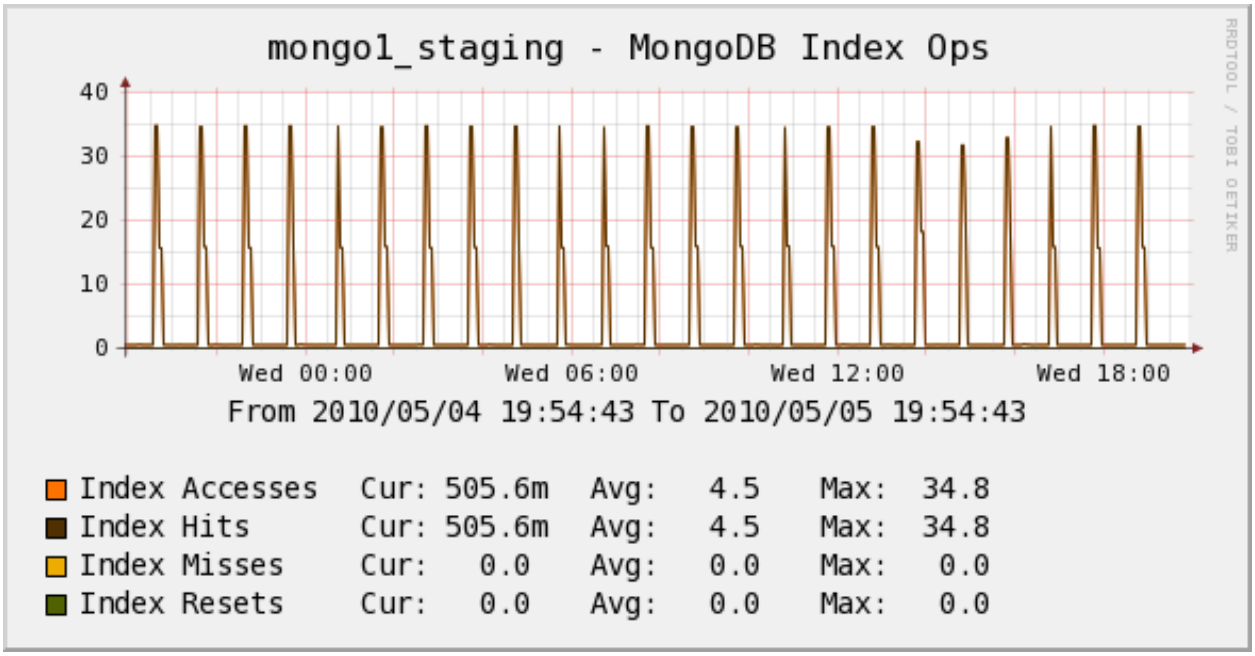
Background flushes



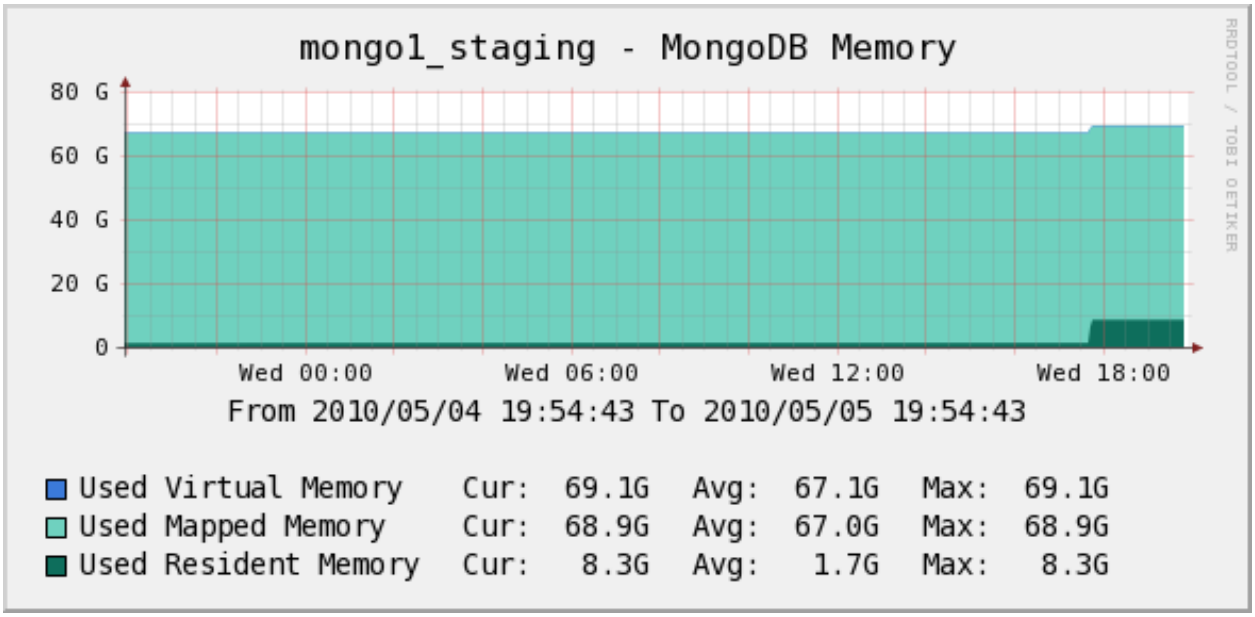
Commands



Connections



Index Operations



Memory

Percona Nginx Monitoring Template for Cacti

These templates use `ss_get_by_ssh.php` to connect to a server via SSH and extract statistics from the Nginx server running there, by executing the `wget` program with the url `/server-status`.

Installation

Once the SSH connection is working, configure Nginx to report its status. You can add the following to any server context and restart Nginx:

```
location /server-status {
   stub_status on;
   allow 127.0.0.1
   # deny all;
}
```

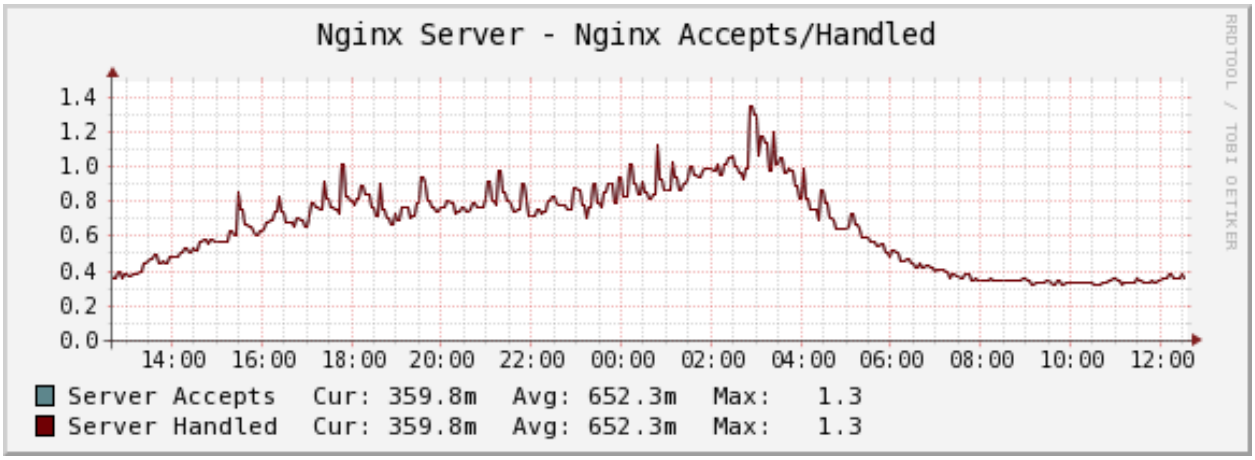
If you decide to use a different URL, you'll have to configure that in the script configuration (covered in the general install guide) or pass a command-line option (also covered in the general install guide).

Finally, test one of your hosts like this. You may need to change some of the example values below, such as the cacti username and the hostname you're connecting to:

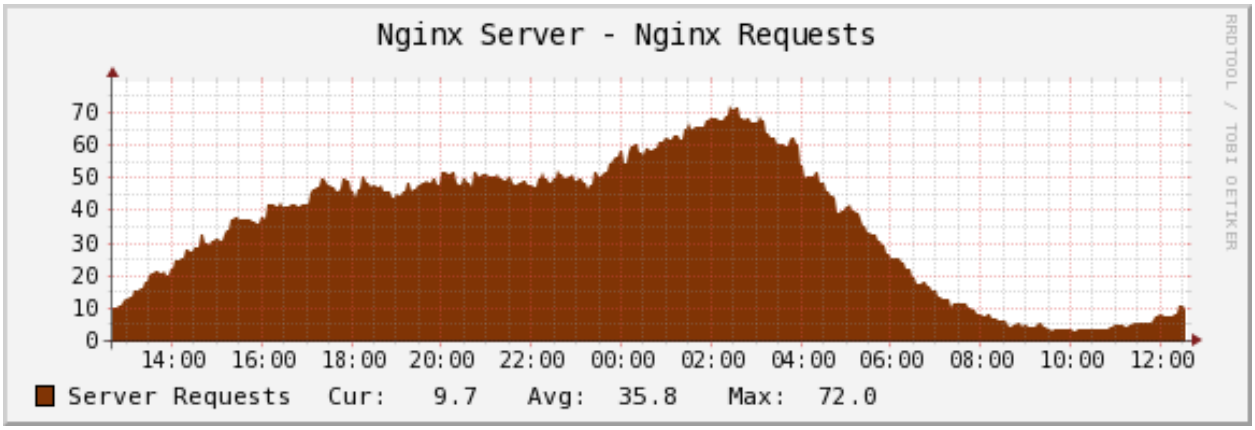
```
sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php --type nginx --host 127.
↪0.0.1 --items hw,hx
```

Sample Graphs

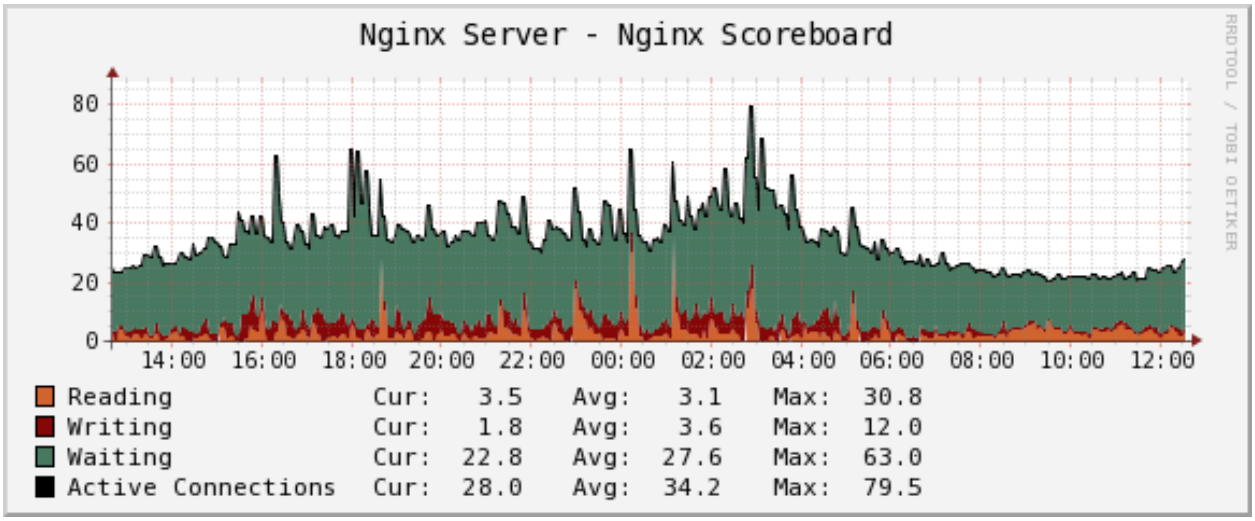
The following sample graphs demonstrate how the data is presented.



The number of connections the server accepted and handled.



The number of requests the Nginx server received.



The number of connections to the Nginx server in various states.

Percona OpenVZ Monitoring Template for Cacti

These templates use `ss_get_by_ssh.php` to connect to a server via SSH and extract statistics from the OpenVZ container, by concatenating the contents of `/proc/user_beancounters` to standard output.

Installation

Once the SSH connection is working, import the OpenVZ template and apply it to your host, then add the graphs.

Depending on your version of OpenVZ or Virtuozzo, the file `/proc/user_beancounters` might not be accessible to ordinary users. This means you either need to SSH as root, or use a program such as beanc to read the file. If you choose the latter approach, you should change the `$openvz_cmd` configuration variable.

You can test one of your hosts like this. You may need to change some of the example values below, such as the cacti username and the hostname you're connecting to:

```
sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php --type openvz --host 127.
↪0.0.1 --items jn,jo
```

Sample Graphs

No sample graphs are available yet.

Percona Redis Monitoring Template for Cacti

These templates use `ss_get_by_ssh.php` to connect to a server and extract statistics from the Redis server with the INFO command. The templates *do not use SSH*, but connect directly with TCP sockets to gather the information.

Installation

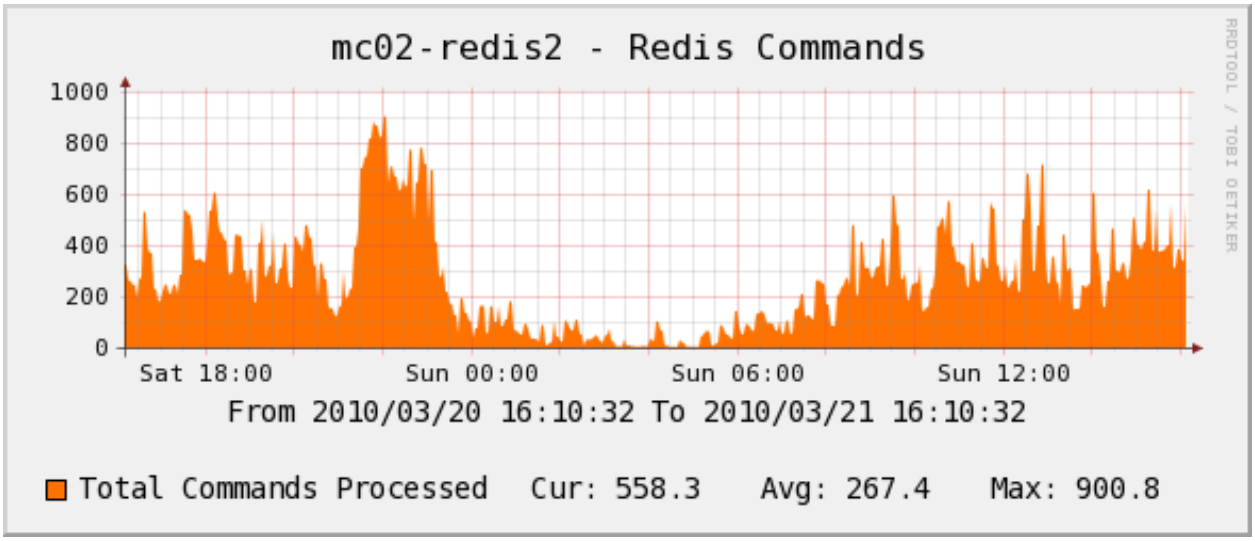
Import the Redis template and apply it to your host, then add the graphs.

You can test one of your hosts like this. You may need to change some of the example values below, such as the cacti username and the hostname you're connecting to:

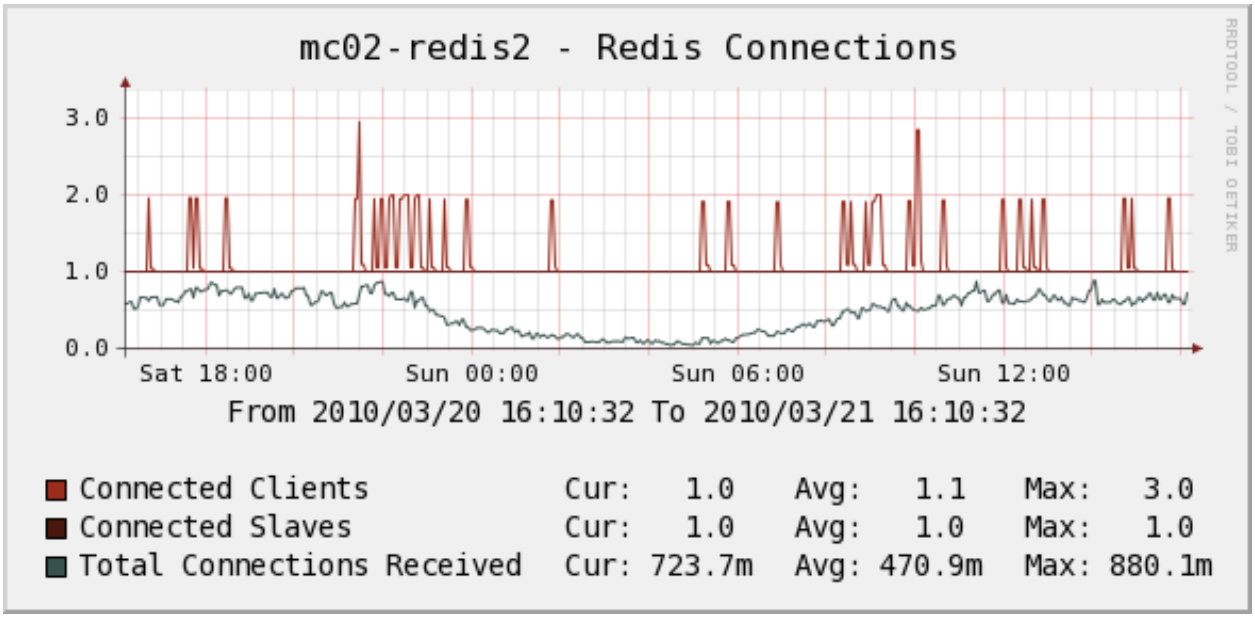
```
sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php --type redis --host 127.
↪0.0.1 --items ln,lo
```

Sample Graphs

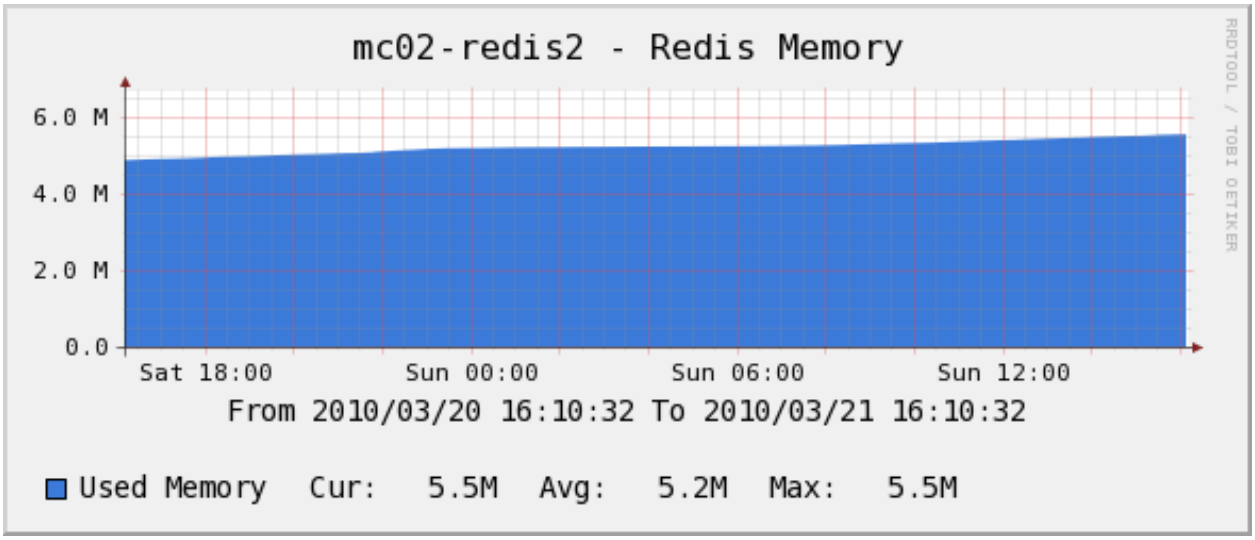
The following sample graphs demonstrate how the data is presented.



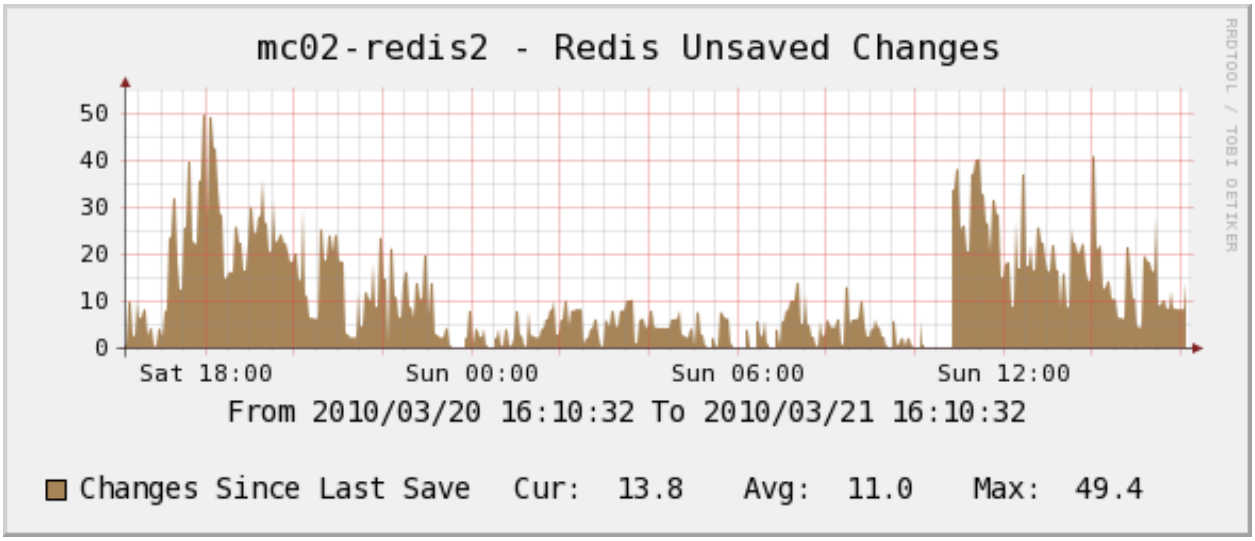
Shows commands to the Redis server.



Shows connections to the Redis server. The top two items are *current* connections at the time the poller sampled the data; the bottom line, Total Connections, is the number of new connections created per second during the polling interval. That's the important number to watch, and despite the name it should not be equal to the sum of the first two lines!



Shows memory usage.



Shows unsaved changes.

Creating an SSH Key Pair

After importing the desired template, which is covered in the template-specific documentation, the next thing to do is set up SSH keys for the poller process to use. To do this, you need to know what user the Cacti poller runs as. You can look in the cron job that runs the poller:

```
debian:~# grep -r cacti /etc/cron*
/etc/cron.d/cacti:*/5 * * * * www-data php /usr/share/cacti/site/poller.php >/dev/
↪null 2>/var/log/cacti/poller-error.log
```

Another way is to simply look at who owns the log files:

```
debian:~# ls -l /var/log/cacti/
total 68
-rw-r----- 1 www-data www-data 53816 2009-10-27 17:55 cacti.log
-rw-r----- 1 www-data www-data  7120 2009-10-25 06:20 cacti.log.1.gz
-rw-r--r-- 1 www-data www-data     0 2009-10-27 17:55 poller-error.log
-rw-r----- 1 www-data www-data     0 2009-10-19 18:57 rrd.log
```

In both cases, you can see that it runs as www-data. You'll need to keep this in mind as you set things up further.

Now you will create an SSH key pair without a passphrase. When `ssh-keygen` asks you where to save the key, you will specify a convenient location. This example is using a Debian server, and Debian keeps the Cacti configuration in `/etc/cacti`, which seems like a better place than `/var/www` (the www-data user's default home directory):

```
debian:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /etc/cacti/id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /etc/cacti/id_rsa.
Your public key has been saved in /etc/cacti/id_rsa.pub.
```

The key has been created with permissions that will not let the www-data user access it, and you need to fix that:

```
debian:~# chown www-data /etc/cacti/id_rsa*
debian:~# ls -l /etc/cacti/
total 16
-rw-r--r-- 1 root     root      539 2008-08-08 21:43 apache.conf
-rw-r----- 1 root     www-data  575 2009-10-20 16:23 debian.php
-rw------- 1 www-data root     1675 2009-10-27 18:07 id_rsa
-rw-r--r-- 1 www-data root      393 2009-10-27 18:07 id_rsa.pub
```

That should work fine.

Creating the User

Now create the user on the server you want to graph. For this example, we'll call this user "cacti". Remember, the server you to graph is 192.168.1.107.

This example shows how to create the user manually and give it a suitable password, but you can create the user however you please:

```
debian:~# ssh 192.168.1.107 adduser cacti
Adding user `cacti' ...
...
```

Installing the Public Key

Once the user is created, you're ready to copy the SSH key into its home directory:

```
debian:~# ssh-copy-id -i /etc/cacti/id_rsa.pub cacti@192.168.1.107
cacti@192.168.1.107's password:
Now try logging into the machine, with "ssh 'cacti@192.168.1.107'", and check in:

  .ssh/authorized_keys
```

```
to make sure we haven't added extra keys that you weren't expecting.

debian:~# ssh -i /etc/cacti/id_rsa cacti@192.168.1.107 echo "it works"
it works
```

Notice that you copied the public key (id_rsa.pub) and then logged in with the private key (id_rsa).

Installing and Configuring the PHP Script

You should now be ready to use the PHP script to connect to this server over SSH. All you need to do is copy ss_get_by_ssh.php to the Cacti script directory and set the proper configuration variables. This example shows how to do it with an external configuration file, but you can do it any way you please:

```
debian:~# cp scripts/ss_get_by_ssh.php /usr/share/cacti/site/scripts/
debian:~# cat > /etc/cacti/ss_get_by_ssh.php.cnf
<?php
$ssh_user   = 'cacti';
$ssh_iden   = '-i /etc/cacti/id_rsa';

CTRL-D
```

If you need a more complex configuration setup, such as connecting to a different SSH port on different servers, follow the instructions to customize the data templates and accept input in each data source.

Securing Your Setup

You can also place `.cnf` file in the same directory as the PHP script file (just to keep the backward compatibility) but this is a security risk as `scripts/` folder falls under the web directory. So `/etc/cacti/` is the recommended location for `.cnf` file. In any case, ensure that any files under `scripts/` are not accessible from Web. Check out *Hardening Cacti setup* guide.

Testing the Setup

Finally, you'll test the script to see if it can connect and retrieve values. It is important to do this as the same user the crontab runs under, with an empty environment, just as the crontab does. Otherwise the results will not necessarily whether Cacti's polling will succeed or fail! The sample call to the script that follows is a good example. Make sure you specify the correct username; the example uses www-data. If the resource you're graphing runs on a non-standard port, use the `--port2` option:

```
debian:~# su - www-data -c 'env -i php /usr/share/cacti/site/scripts/ss_get_by_ssh.
↪php --type memory --host 192.168.1.107 --items gu,gv'
gu:30842880 gv:2244608
debian:~#
```

In the example above, the script did not print a newline after its output, so the prompt is likely to be mangled afterwards, but the output is "gu:30842880 gv:2244608", followed by the command prompt, "debian:~# ".

Everything looks fine, so the graphing should be working! Continue with the template-specific documentation.

Percona Amazon RDS Monitoring Template for Cacti

This page gives installation instructions specific to the RDS graph template, shows examples of graphs in the RDS template collection, and shows what they do.

Installation Notes

This template utilizes the Python script and `boto` module (Python interface to Amazon Web Services) to get various RDS metrics from CloudWatch.

To make the script working, please follow the instructions:

- Install the package: `yum install python-boto` or `apt-get install python-boto`
- Create a config /etc/boto.cfg or ~cacti/.boto with your AWS API credentials. See http://code.google.com/p/boto/wiki/BotoConfig

The script `~cacti/scripts/ss_get_rds_stats.py` that is run under `cacti` user should have permissions to read the config /etc/boto.cfg or ~cacti/.boto.

For example:

```
[root@centos6 ~]# cat /etc/boto.cfg
[Credentials]
aws_access_key_id = THISISATESTKEY
aws_secret_access_key = thisisatestawssecretaccesskey
```

If you do not use this config with other tools such as our Nagios plugin, you can secure this file the following way:

```
[root@centos6 ~]# chown cacti /etc/boto.cfg
[root@centos6 ~]# chmod 600 /etc/boto.cfg
```

IMPORTANT: If you decide to create `~cacti/.boto` instead, which is not secure as it falls under the web directory, ensure this file is not accessible from Web. Check out *Hardening Cacti setup* guide.

Test the script assuming DB instance identifier is `blackbox`:

```
[root@centos6 ~]# sudo -u cacti ~cacti/scripts/ss_get_rds_stats.py --ident=blackbox --
↪metric=CPUUtilization
gh:6.53
```

To check RDS details you can run:

```
[root@centos6 ~]# sudo -u cacti ~cacti/scripts/ss_get_rds_stats.py --ident=blackbox --
↪print
...
```

Now, you can add a device to the Cacti using Amazon RDS Server template and graph it.

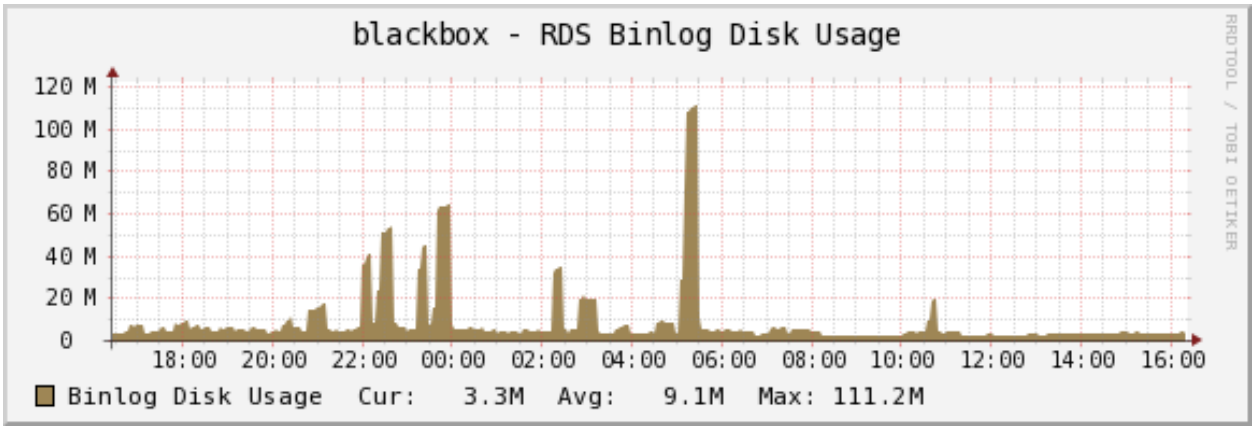
NOTE: you need to specify DB instance identifier as `Hostname` on adding device page.

By default, the region is set to `us-east-1`. You can re-define it globally in boto config or specify per instance on data source level in Cacti. You can also set region to `all` which will have the script to find region for a given instance automatically. However, this will work much slower than specifying region explicitly.

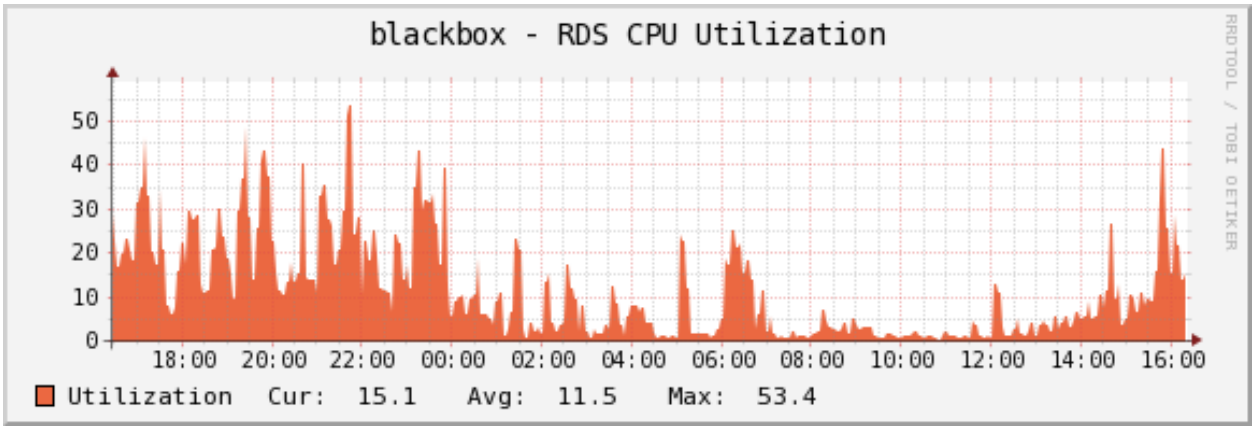
Also you can specify boto profile name on data source level in Cacti in case you have multiple in use.

Sample Graphs

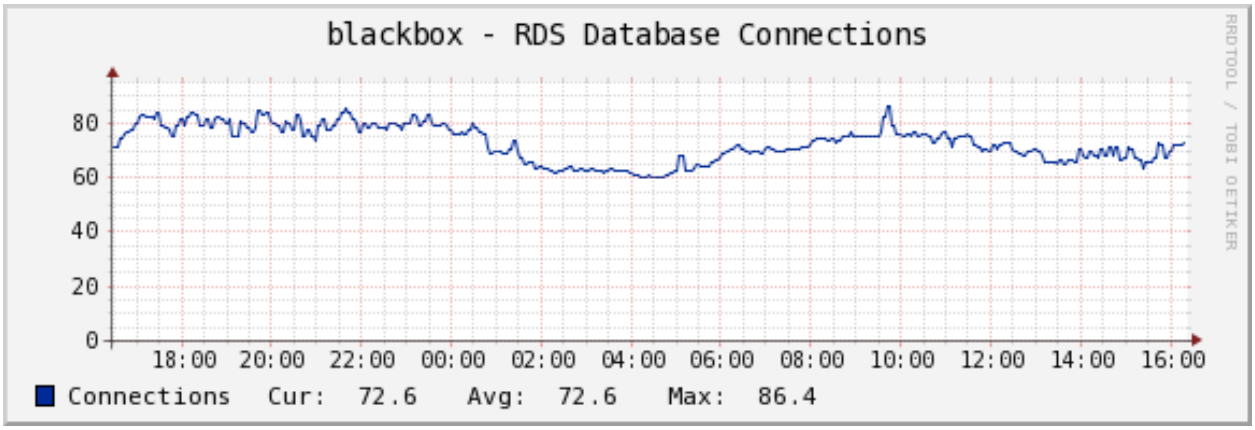
The following sample graphs demonstrate how the data is presented.



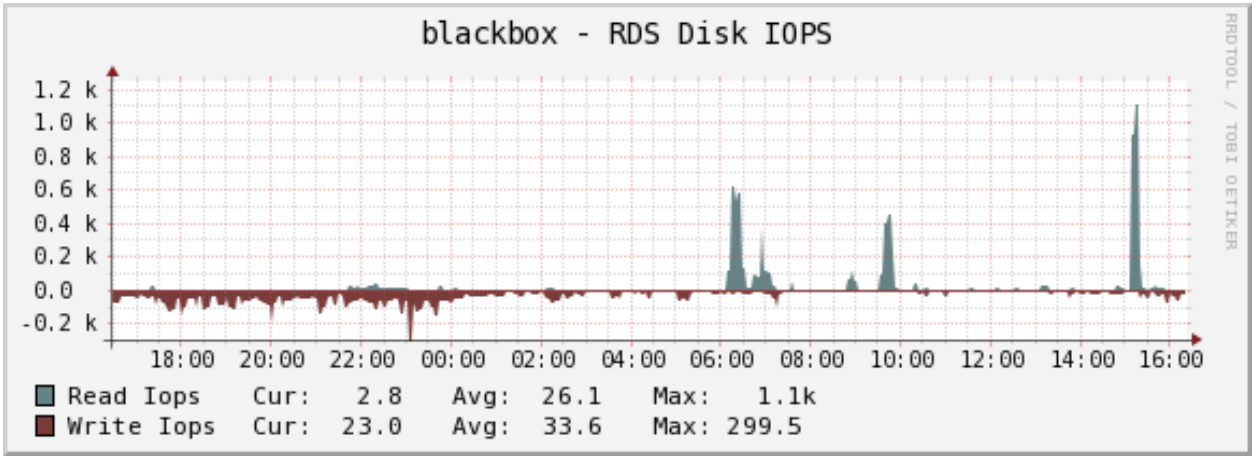
The amount of disk space occupied by binary logs on the master.



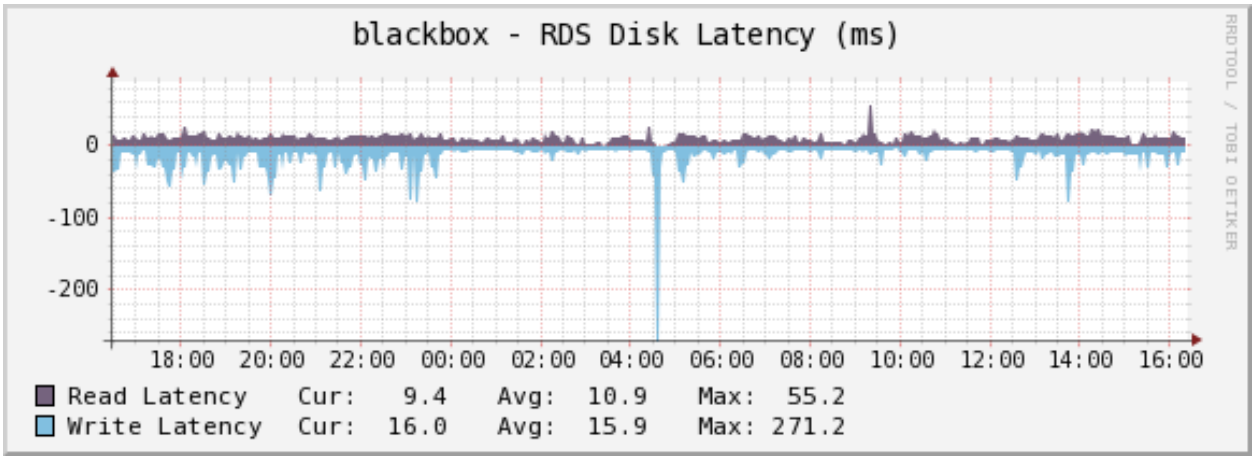
The percentage of CPU utilization.



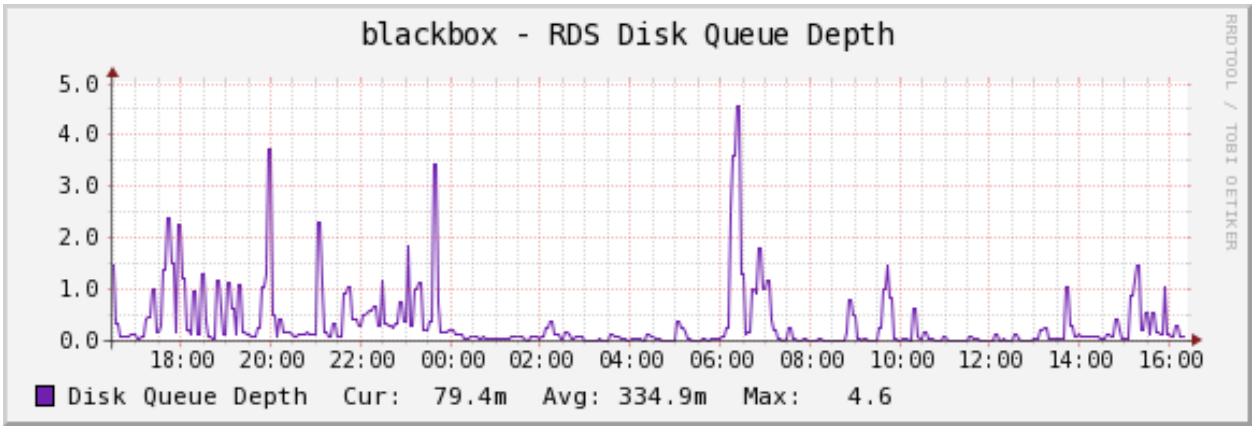
The number of database connections in use.



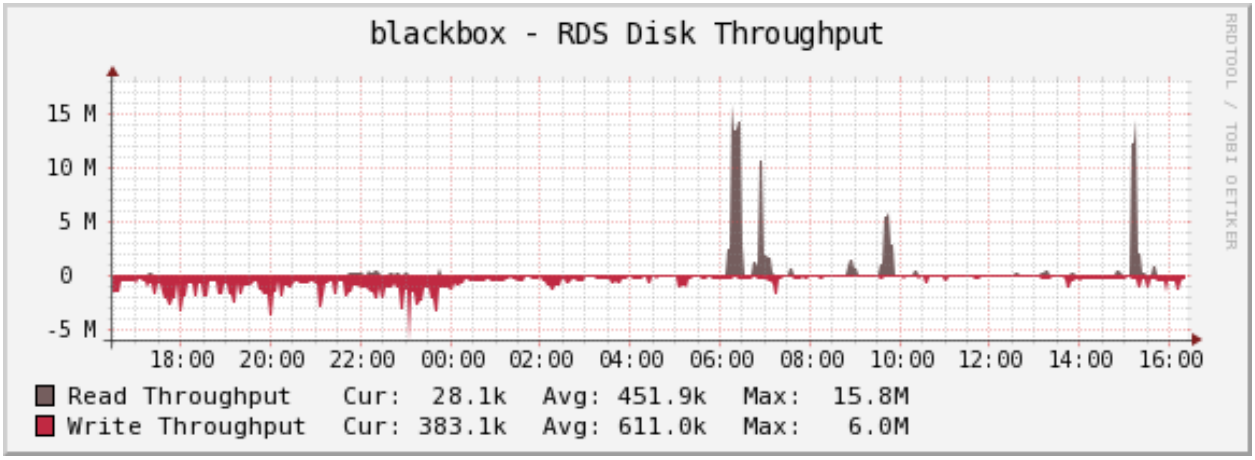
The average number of disk I/O operations per second.



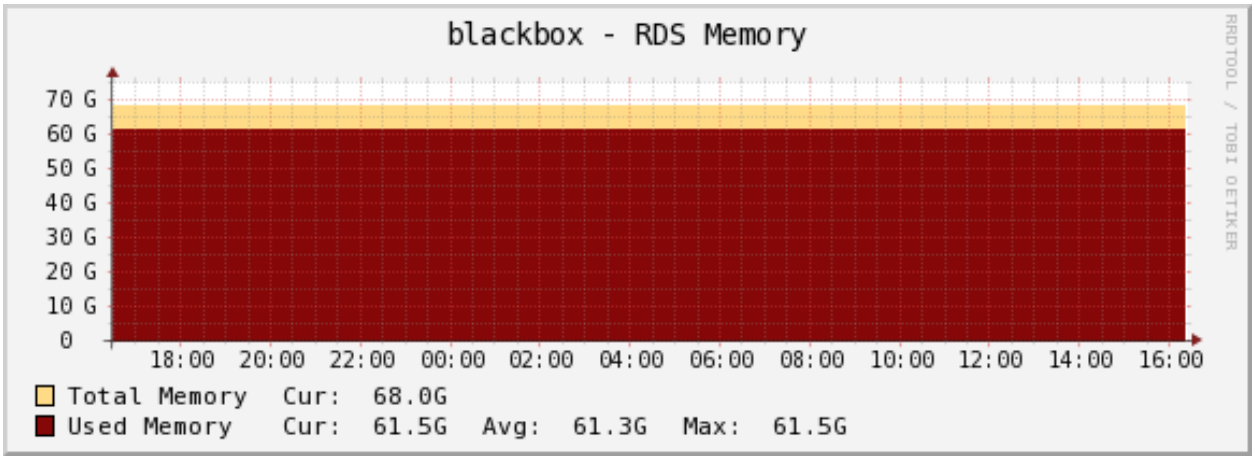
The average amount of time taken per disk I/O operation.



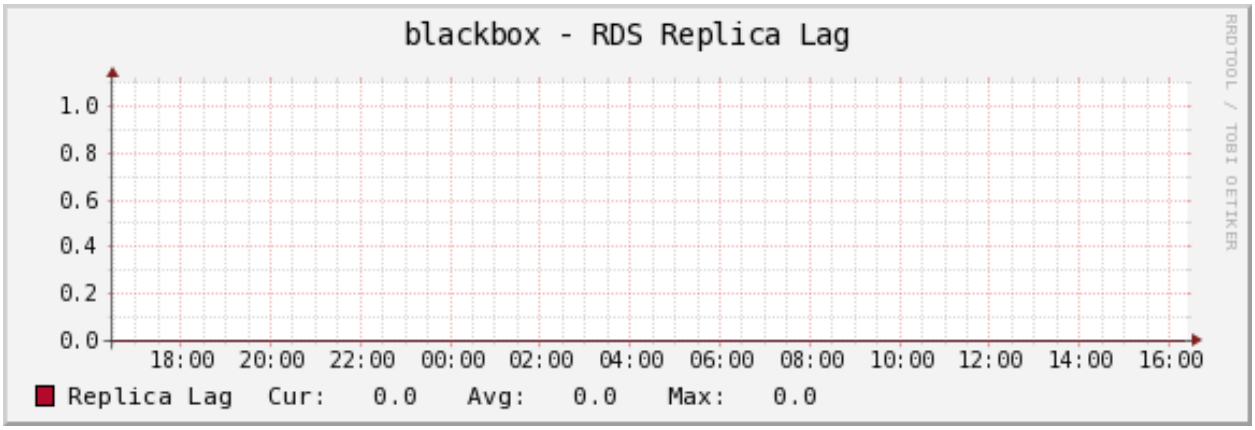
The number of outstanding IOs (read/write requests) waiting to access the disk.



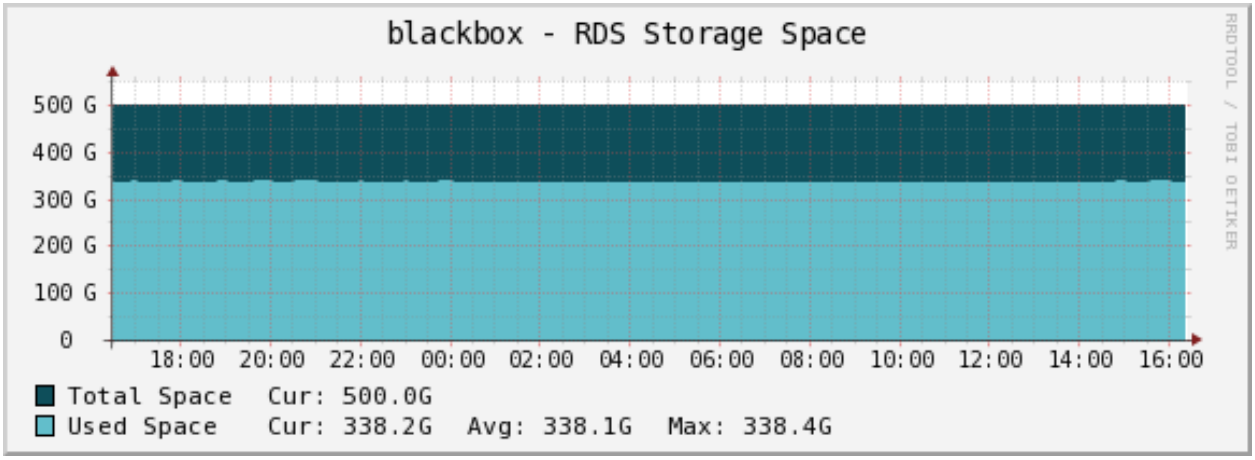
The average number of bytes read from/written to disk per second.



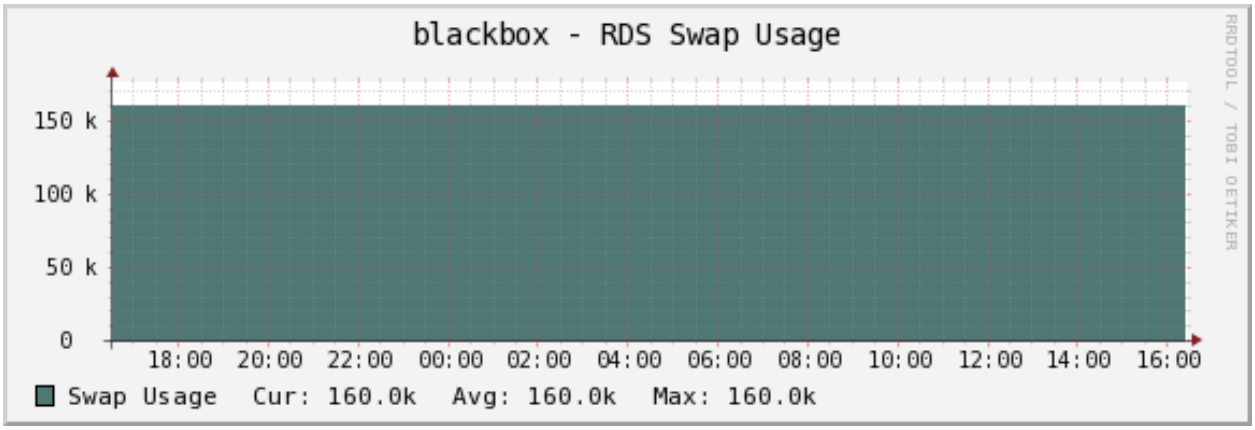
The amount of used random access memory. The total available memory is the value according to the instance class. See http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html



The amount of time a Read Replica DB Instance lags behind the source DB Instance.



The amount of used storage space.



The amount of swap space used on the DB Instance.

Cacti Templates Developer Documentation

This documentation is for users who wish to extend the Cacti templates to create new graphs, collect new types of data, and so on.

Adding Graphs

This document shows you how to add graphs to an existing template. Much of the technical background is covered in the documentation on creating graphs, so you can treat this page as a quick-reference.

Here's what to do:

1. Make sure there is a blueprint, so the work is identified and tracked.
2. Do whatever's necessary to return the data items from the script (usually as integers, not floats, unless you're using `GAUGE`).
3. Add the data items to the magic array.
4. Write tests as appropriate.
5. Ensure all existing tests still pass.

6. Add a new graph definition in the `definitions/` Perl file.
7. Add the new data items to the end of the `definitions/` Perl file, in the input.
8. Make the hashes unique.
9. Test the graphs. Don't forget to upgrade the PHP script!
10. Update the documentation. Add new samples and explanatory text.

Cacti Hash Identifiers

Cacti generates a type of GUID identifier for each object. This has a few characters of metadata: the text `hash_`, the object type, the Cacti version that generated the GUID, and a random string. The inclusion of the Cacti version number means that graphs aren't backwards compatible if you regenerate them on a newer version of Cacti and try to install them on an older version of Cacti. The older version will examine the version string in the hash and generate an error.

To avoid this problem, this templating system generates hashes that look like this:

```
hash_10_VER_ac260a1434298e088f15f70cd1a5f726
```

The template generation process replaces the `_VER_` constant with an appropriate value for the target version of Cacti. As an example, if you're generating for Cacti 0.8.6g, the value will look like the following:

```
hash_100010ac260a1434298e088f15f70cd1a5f726
```

If you need to add support for a newer version, look for `%hash_version_codes` in `pmp-cacti-template`.

The hashes in the template definition `.def` files should be globally unique. It's difficult to generate them manually, so there is a `pmp-cacti-make-hashes` helper tool to make this easier. You can read more about this in the documentation on creating graphs.

Creating Graphs

This document explains how to create Cacti templates simply and reliably. For this example, we'll assume you will add a new set of graphs into the `ss_get_by_ssh.php` file. This means you will write the following:

- A command to SSH to a server and collect some data
- Code to parse that into a list of named values to graph
- Graphs to display the result

This example will show a fairly complex set of graphs. If you understand this example, you should be ready to create quite complex graphs yourself. We'll collect data from `/proc/diskstats` on Linux and graph it.

Collecting and Parsing the Data

Getting the data is quite simple. The template will SSH to the server you're graphing and `cat /proc/diskstats`. Then you'll parse the result into name-value pairs.

Start by writing tests against sample files, before you write any code. This is the most important part of creating reliable graphing code. Collect several samples of /proc/diskstats from different sources so you get a variety of cases. For example, it's best to collect samples from different kernel versions or distributions. Save these into files such as `t/samples/diskstats-001.txt` as test samples.

Next, write the tests. Open `t/get_by_ssh.php` and add `is_deeply()` calls to the bottom to test each sample. You'll test a function called `diskstats_parse()`. (But don't write the function yet!) The name of this function is important, as you'll see a bit later. The naming convention is how the PHP script will find and execute the function. It needs to be called SOMETHING_parse, where SOMETHING is the command-line argument given to the script's `--type` option. In this case, "diskstats" seems like a good name. If you use that, then the command-line will eventually become `php ss_get_by_ssh.php --type diskstats ...`, which is reasonable.

Each call to the function will return information about one disk device, so you'll need to add a `--device` option to the PHP script eventually. For right now, all you need to know is that the prototype of `diskstats_parse()` is an array of command-line options, and a string of text with the contents of `/proc/diskstats`. The call will look like this:

```
diskstats_parse( array('device' => 'hda1'), "<contents of /proc/diskstats....>" );
```

Refer to documentation as needed and manually inspect the sample of `/proc/diskstats`, then write out the values you expect to be returned in the test. This should be a name-value list with *meaningful names*. This is quite important. The names you choose here will be used in many different places, including as text on the graphs you'll eventually create. Separate the words with underscores, and use all lowercase letters. Choose a unique 4-letter uppercase prefix for the name. Here is an example:

```
is_deeply(
   diskstats_parse( array('device' => 'hda1'), file_get_contents('samples/diskstats-
↪001.txt') ),
   array(
      'DISK_reads'              => '12043',
      'DISK_reads_merged'       => '387',
      'DISK_sectors_read'       => '300113',
      'DISK_time_spent_reading' => '6472',
      'DISK_writes'             => '12737',
      'DISK_writes_merged'      => '21340',
      'DISK_sectors_written'    => '272616',
      'DISK_time_spent_writing' => '22360',
      'DISK_io_ops'             => '24780',
      'DISK_io_time'            => '12368',
      'DISK_io_time_weighted'   => '28832'
   ),
   'samples/diskstats-001.txt'
);
```

Make sure that your parsing code outputs integers, not floating-point numbers, for counters that will increase as time passes. This is because floating-point numbers cannot be stored into `COUNTER` and `DERIVE` values in RRD files.

Once you have written the test, execute the test file and ensure that the test fails! If it doesn't fail, you have no reliable indicator that a passing test really means success:

```
t/cacti $ php get_by_ssh.php
```

You should see a fatal error that the function `diskstats_parse()` doesn't exist. Now it's time to write it. Open `scripts/ss_get_by_ssh.php` and move all the way to the bottom of the file. Scroll upwards three functions. Those three functions should be the three magically named functions for some other parsing process. Use them as a reference for writing your own function. Copy the block comment header (don't worry much about its contents right now), and start your own function. It should look like this:

```
function diskstats_parse ( $options, $output ) {
   $result = array();
   # write some code here
   return $result;
}
```

The returned value should be the array that you're testing for. Write until your test passes, and then write similar tests for the other two sample files. Now you're done with the parsing code!

Writing the Command Line

If you've gotten this far, you've collected sample data, and written well-tested code to parse it. This code is contained in a function called `diskstats_parse()`, which follows a specific naming convention. You need two more functions that follow the same naming convention, and the command-line option that will cause these functions to be executed.

The first function specifies a cache filename. Caching is an efficiency and consistency feature that helps work around a flaw in Cacti's design. The filename needs to be unique for every different host and device you want to graph, so you need to include the `--host` and the `--device` command-line options in the filename.

The second function creates the command line that will gather the data from `/proc/diskstats` over SSH. The functions can be quite simple:

```
function diskstats_cachefile ( $options ) {
   $sanitized_host
       = str_replace(array(":", "/"), array("", "_"), $options['host']);
   $sanitized_dev
       = str_replace(array(":", "/"), array("", "_"), $options['device']);
   return "${sanitized_host}_diskstats_${sanitized_dev}";
}

function diskstats_cmdline ( $options ) {
   return "cat /proc/diskstats";
}
```

Now you need to add documentation for the new `--type` command-line option to the PHP script. The argument to this option can be free-form text, so all you need to do is add the text to the `--help` output. Here's a diff to show what to change:

```
@@ -197,7 +198,7 @@
    --server    The server (DNS name or IP address) from which to fetch the
                desired data after SSHing.  Default is 'localhost' for HTTP stats
                and --host for memcached stats.
-   --type      One of apache, nginx, proc_stat, w, memory, memcached
+   --type      One of apache, nginx, proc_stat, w, memory, memcached, diskstats
                (more are TODO)
    --url       The url, such as /server-status, where server status lives
    --use-ssh   Whether to connect via SSH to gather info (default yes).
```

There is one final detail, which is necessary because this is a rather advanced graphing task: you need to add a `--device` command-line option so the PHP code can figure out which disk device the user is interested in graphing. This should be added in two places: a) the command-line `--help` output you just saw, and b) in the `validate_options()` function. Here's another diff:

```
@@ -160,7 +160,7 @@
 function validate_options($options) {
    debug($options);
    $opts = array('host', 'port', 'items', 'nocache', 'type', 'url', 'http-user',
-                 'file', 'http-password', 'server', 'port2', 'use-ssh');
+                 'file', 'http-password', 'server', 'port2', 'use-ssh', 'device');
    # Required command-line options
    foreach ( array('host', 'items', 'type') as $option ) {
       if ( !isset($options[$option]) || !$options[$option] ) {
```

Now you can specify `--device sda1` or similar, and the code can access that through `$options['device']`, as you've seen in the examples above.

Adding a Custom Getter Function

The `ss_get_by_ssh.php` script assumes you're going to write an `XXX_cmdline()` function that will return the commandline to be executed via SSH. However, it is possible to bypass this functionality and provide your own code to execute directly, instead of fetching data over SSH. To do this, create a function called `XXX_get()` that returns the data directly. You can see an example of this in the Redis graphs, where sockets are used to get Redis status directly instead of via SSH.

Specifying a Short-Name Mapping

You already created long, descriptive names for the data values you're going to graph. Unfortunately, due to another Cacti limitation, these names can't be used safely everywhere. In most Cacti templates, the script returns a key:value string to Cacti, like this:

```
Name_of_data_value:1234 Name_of_another_data_value:5678
```

That will not work reliably in these templates, because they fetch all of their data at once for efficiency and consistency, as stated earlier. When a script returns dozens of values in a single call, Cacti loses the data, because it overflows Cacti's fixed-length buffer. As a result, you need need a mapping between the long names you've used previously, and some type of shorter names for Cacti's benefit. The template system has a naming convention that handles this for you.

The mapping is defined in an array in the PHP script, which is a single paragraph of text (no empty lines) preceded by the magic word MAGIC_VARS_DEFINITIONS. You need to append your data variables to this array and give each name a unique abbreviation. For example:

```
# MAGIC_VARS_DEFINITIONS: Define the variables to output
$keys = array(
   'DISK_reads'                         =>  'iw',
   'DISK_reads_merged'                  =>  'ix',
   'DISK_sectors_read'                  =>  'iy',
   'DISK_time_spent_reading'            =>  'iz',
   'DISK_writes'                        =>  'jg',
   'DISK_writes_merged'                 =>  'jh',
   'DISK_sectors_written'               =>  'ji',
   'DISK_time_spent_writing'            =>  'jj',
   'DISK_io_ops'                        =>  'jk',
   'DISK_io_time'                       =>  'jl',
   'DISK_io_time_weighted'              =>  'jm',
);
```

The convention is two-letter abbreviations, beginning at gg, gh, and so on. Do not use the letters from `a` through `f` or digits, because there is a bug in some versions of Cacti that treats an all-hexadecimal name as a value instead of a prefix that identifies the value. Append your data items to the list, and continue the convention.

Now you can see why the uppercase DISK identifier chosen earlier (during the test phase) is necessary. This makes the names unique. Otherwise you might create two items in this array named 'writes', which would cause a bug.

The short names are eventually used in the `--items` command-line argument. This argument can take any combination of short names. Now that you know what your short names will be, go back to the comment header right above the `diskstats_cachefile()` function, and write a sample command-line users can use to test the functionality you're creating, such as the following:

```
# ============================================================================
# Get and parse stats from /proc/diskstats
# You can test it like this, as root:
# sudo -u cacti php /usr/share/cacti/scripts/ss_get_by_ssh.php \
#    --type diskstats --host 127.0.0.1 --device sda1 --host 127.0.0.1 --items iw,ix
# ============================================================================
function diskstats_cachefile ( $options ) {
```

Notice that the `--items` argument is simply a comma-separated list of short names you defined in the mapping array. This is how Cacti will eventually execute the script to gather the data.

Write Another Test

You are now finished editing the PHP, except for one last thing: write another test case. Make it test the integration of all the code you've written, and ensure that it all works right together. Look in the test file for tests against the `ss_get_by_ssh()` function, and emulate that. For example:

```
is(
   ss_get_by_ssh( array(
      'file'    => 'samples/diskstats-001.txt',
      'type'    => 'diskstats',
      'host'    => 'localhost',
      'items'   => 'iw,ix,iy,iz,jg,jh,ji,jj,jk,jl,jm',
      'device'  => 'hda1'
   )),
   'iw:12043 ix:387 iy:300113 iz:6472 jg:12737 jh:21340 ji:272616 jj:22360'
   . ' jk:24780 jl:12368 jm:28832',
   'main(samples/diskstats-001.txt)'
);
```

Now you can go on to defining the graphs.

How the Graph System Works

Cacti's templating system is quite difficult to work with. It uses cryptic values, has a lot of redundant data, and uses randomly generated hashes as unique identifiers. The typical Cacti template is defined within Cacti and then exported, which causes problems for others who import that template. Finally, creating nice consistent templates through the web interface is tedious. You could easily spend several days doing it, one click at a time.

This Cacti template definition system alleviates those problems. It uses a highly compressed version of the Cacti template system with special conventions. This removes redundancy, and eliminates a lot of work and errors.

This system has a simple relationship between the parts it represents. If you're familiar with Cacti, the following might help you understand:

1. An input is defined only once, instead of repeated for every graph. This means that all the graphs for a related set of data draw their data from a common command. The input is defined by a command-line that executes it, command-line arguments it accepts, and values it outputs.
2. Each graph is associated with one graph template.
3. Each graph template has a corresponding data template, which has exactly the inputs and outputs that the graph needs, no more, no less. Data templates are not shared across several graph templates or vice versa; there is a strict one-to-one relationship.
4. Each RRD file definition maps exactly to one graph template and therefore to one data template, again in a one-to-one relationship.

5. The graph templates, data templates, and RRD definitions are named the same way, but with a distinguishing suffix automatically added by the template generation tools. This makes it easier to identify them.
6. The random hash identifiers are defined exactly once in the system, and are hard-coded into the definition file. They never change, which removes the randomness. The hashes are written in an abstract form in the definition file.

The summary of the above is "don't repeat yourself." Cacti repeats itself a lot; this template system simplifies by creating a one-to-one-to-one relationship from the data collection all the way through to the graph definition.

Now that you know this, you are ready to learn about the definition file.

Structure of the Definition File

The definition file is a Perl variable containing nested data structures. The relationship amongst the various types of data looks like this:

- There is one top-level template.
- The template contains some properties such as name and version.
- The template contains two major sections: graphs and inputs.
  - The graph section is an array of graph template definitions. Because of the one-to-one-to-one relationship amongst them, each graph template definition implicitly defines a corresponding data template and an RRD file definition.
  - The input section is an array of input definitions. Each one defines the data that flows between the PHP script you wrote above, and the graph templates.

This should become clearer as you read through the rest of this document.

Before we go on, though, you need to understand about hashes. If you examine the definition file, you'll see some things that look like this:

```
task   => 'hash_09_VER_e2a72b5aa0b06ad05dcd368ae0a131cf',
... snipped ....
hashes => [
   'hash_10_VER_3eae0c8f769939bb30c407d4edcee0c0',
   'hash_10_VER_25aaadab40c1c8e12c45ce61693099b7',
   'hash_10_VER_43f90f7f26a7c6b3ca41c7219afaa50c',
   'hash_10_VER_df9555d08c88c6c0336fe37ffe2ad74a'
```

Those hex digits are hashes. You will later create unique hashes, but for now, follow these steps to prevent problems:

1. Always create your template definitions by copying and pasting whatever you're working on. If you're creating a new input, copy and paste an old one.
2. Always copy and paste downwards in the file. Never take something from the file and copy/paste it higher up in the file.

Copying and pasting will create duplicate hashes, but that is okay for now. There is a tool to detect these and randomly generate new ones that aren't duplicates. This works well, as long as you don't copy/paste higher in the file. If you do that, the pre-existing hashes will get overwritten with newer ones, which is bad. Later you'll see how to check for this, just in case.

Defining an Input

The first step is to define your input. You created a whole new group of data, which you can access with `--type diskstats`. Create a new input for that by duplicating the input called "Get Proc Stats":

```
'Get Proc Stats' => {
   type_id      => 1,
   hash         => 'hash_03_VER_b8d0468c0737dcd0863f2a181484f878',
   input_string => '<path_php_binary> -q <path_cacti>/scripts/ss_get_by_ssh.php '
                 . '--host <hostname> --type proc_stat --items <items>',
   inputs => [
      {  allow_nulls => '',
         hash        => 'hash_07_VER_509a24f84c924e9252be9a82c6674a6f',
         name        => 'hostname'
      },
   ],
   outputs => {
      STAT_interrupts       => 'hash_07_VER_cf50d22f8b5814fbb9e42d1b46612679',
      STAT_context_switches => 'hash_07_VER_49aa057a3935a96fb25fb511b16a75fa',
      STAT_forks            => 'hash_07_VER_d5e03c6e39717cc6a58e85e5f25608c6',
      STAT_CPU_user         => 'hash_07_VER_edfd4ac62e1e43ec35b3f5dc10ae2510',
      STAT_CPU_nice         => 'hash_07_VER_474ae20e35b85ca08645c018bd4c29c4',
      STAT_CPU_system       => 'hash_07_VER_89c1f51e8cbf6df135e4446e9c656e9b',
      STAT_CPU_idle         => 'hash_07_VER_f8ad00b68144973373281261a5100656',
      STAT_CPU_iowait       => 'hash_07_VER_e2d5a3ef480bb8ed8546fe48c3496717',
      STAT_CPU_irq          => 'hash_07_VER_a8ff7438a031f05bd223e5a016d443b2',
      STAT_CPU_softirq      => 'hash_07_VER_b7055f7e8e745ab6c0c7bbd85f7aff03',
      STAT_CPU_steal        => 'hash_07_VER_5686b4b2d255e674f46932ae60da92af',
      STAT_CPU_guest        => 'hash_07_VER_367fbfbb15a0bbd73fae5366d02e0c9b',
   },
},
```

What does the above mean?

The name of the input will be called "Get Proc Stats". It is of type 1, which is a PHP script. It has a hash, which is its unique identifier. It has an input_string, which is really its command-line. You can see some special things in angle-brackets, which is Cacti's replacement variable notation.

Next it has inputs. (This is confusing, because the input to one thing is the output of another). There is only one input, the hostname. This is a placeholder for Cacti to insert the hostname into the script's command-line arguments when it executes the PHP. If you're wondering what gets put into the `<items>` argument placeholder in the input_string, that's taken care of automatically by the template generation system.

Finally, the input has outputs. These are the values that the PHP script will return when you call it. However, for sanity, they are mentioned here in their long form. As mentioned previously, the short-to-long mapping is defined only once, in the PHP file you edited. Everywhere else you will use the long form of the names, and the template generation system will take care of translating that to the short form where needed.

You need to copy and paste the text, and just update it to make a new input definition. You'll end up with something like this:

```
'Get Disk Stats' => {
   type_id      => 1,
   hash         => 'hash_03_VER_da6fa9ee8283a483d4dea777fd69c629',
   input_string => '<path_php_binary> -q <path_cacti>/scripts/ss_get_by_ssh.php '
                 . '--host <hostname> --type diskstats --items <items> '
                 . '--device <device>',
   prompt_title => 1,
   inputs => [
      {  allow_nulls => '',
         hash        => 'hash_07_VER_280cd9c759c52b2477b972334210f920',
         name        => 'hostname'
      },
      {  allow_nulls => '',
```

```
         hash        => 'hash_07_VER_e89872554729dcd0695528adec190dd2',
         name        => 'device',
         override    => 1,
      },
   ],
   outputs => {
      DISK_reads              => 'hash_07_VER_00e4dd20a4e29c673a4471b2ee173ac9',
      DISK_reads_merged       => 'hash_07_VER_8af205c19a7439e83cee53059096b8e3',
      DISK_sectors_read       => 'hash_07_VER_9c5a554f4d62343e5aaaf9f0d784ada0',
      DISK_time_spent_reading => 'hash_07_VER_e8fd959febe8cdd5b20b8282ba340f19',
      DISK_writes             => 'hash_07_VER_1384e83ff216c0377a5f213f9a88c6fa',
      DISK_writes_merged      => 'hash_07_VER_c9cb7f45fa6ad943c377efb3ba2e661d',
      DISK_sectors_written    => 'hash_07_VER_43f100a2f54d5b18c3cdc5e8b8a02293',
      DISK_time_spent_writing => 'hash_07_VER_e1886d79cfa3c526c899de03db6e07ee',
      DISK_io_time            => 'hash_07_VER_cad0f7e9d765ba4e9341de72c0366575',
      DISK_io_time_weighted   => 'hash_07_VER_d7ebd195f6d9048b8e1e84114e8a0b6d',
   },
},
```

That should look familiar to you from the work you've done already. The name is "Get Disk Stats". The outputs are what you chose in your first test case.

Although this document shows newly generated hashes here, don't worry about it. Copy/paste the hashes from the other input definition. As long as you paste *below* where you copy, generating new hashes is easy.

The only things really special here are because disk statistics have to know which device they're graphing:

- You need to add a command-line option for `--device`.
- You need to tell Cacti that this command-line option can't be left null: `allow_nulls => '',`.
- You need to tell Cacti to ask the user for the device every time the data template is applied to a graph: `override => 1,`. This is equivalent to checking the checkbox "Use Per-Graph Value (Ignore this Value)" on the data template in the Cacti interface.
- You need to tell Cacti to prompt the user to customize the graph title when creating graphs: `prompt_title => 1,`.

The result is that Cacti will permit data entry for `--device`, it will require it, and it will ask for it to be provided for every graph.

Defining the Graph

Now that you have the definition of the input that you're going to graph, you need to specify the graph itself – how that data should be presented visually.

Again, begin by copy/pasting another definition, but copy above, paste below. here's the result, with a few things snipped for brevity:

```
{  name       => 'Disk Sectors Read/Written',
   base_value => '1024',
   hash       => 'hash_00_VER_9fad7377daacfd611dae46b14cc4f67e',
   override   => { 'title' => 1 },
   dt         => {
      hash       => 'hash_01_VER_67811065b100a543ddeadf7464ae017c',
      input      => 'Get Disk Stats',
      ... snipped! ...
   },
   items => [
```

```
      ... snipped! ...
   ],
},
```

Ignoring the snipped sections for right now, here's what that means:

- *name* is pretty self-explanatory. This name will be used in all the redundant places that Cacti wants it: in the graph template, in the graphs themselves, and so on, including the graph title.
- *base_value* is usually 1000 or 1024. Use 1000 except for things where you'd expect a unit of 1024, such as when the things graphed are measured in bytes. Here we're using 1024 because we're talking about sectors read and written, and sectors are a power-of-two of bytes.
- *hash* is just a hash. Just copy/paste and let the uniquifying process take care of that.
- *override* does not need to be used for most graphs. Specifying an element here is equivalent to checking "Use Per-Graph Value (Ignore this Value)" next to that item on the graph template page inside of Cacti. It means that this item won't be taken straight from the template for each graph; when you create the graph you'll be prompted to supply a value for the item. We need to use it for this graph because we want to modify the graph's title to include the device or partition we're graphing in this graph. When you create a graph, you'll be prompted for the device to graph, and you'll be able to customize the graph title so you can see that device easily.
- *dt* defines things that are specific to the data template (remember, one graph template == one data template). You need a hash (again, copy/paste for now), and you need to specify which input the data comes from. Then, following this, you'll specify a varying number of sections, one for each item you want to graph from that input.
- *items* includes a varying number of sections, too – also one per thing you want to graph.

Now let's look at the bits that were snipped out of the code listing above. First, the sections that say what data to get out of the input:

```
DISK_sectors_read             => {
   data_source_type_id => '3',
   hash => 'hash_08_VER_80929ee708f7755d09443d3d930a29cc',
},
DISK_sectors_written          => {
   data_source_type_id => '3',
   hash => 'hash_08_VER_f5d85616af1e03a679042978c938a7ee',
},
```

That's two items. Each one basically says "graph this, and here's the type and hash for it." The thing to graph needs to be one of the data items that comes from the input. The hash you should leave copy/pasted for now. The *data_source_type_id* can have a few different values. These map directly to RRDTool data types:

- The value 1 means a GAUGE.
- The value 2 means a COUNTER (increasing, with overflow checks). It's best to use a DERIVE instead.
- The value 3 means DERIVE (increasing, with no overflow checks). It is usually best to use DERIVE with a minimum value of 0 instead of a COUNTER.

You should usually use a DERIVE or GAUGE. Anything that's a steadily increasing counter is a DERIVE, as in the example above. Remember that DERIVE (and COUNTER) cannot accept floating-point numbers, so make sure that the data is converted to integers somehow.

Here's the next section that is omitted from the code sample above. This one contains the items that will appear on the graph itself:

```
{  item   => 'DISK_sectors_read',
   color  => '542437',
   task   => 'hash_09_VER_38f255216fd118d6d88a46d42357323c',
```

```
   type   => 'AREA',
   hashes => [
      'hash_10_VER_7fe10cf273b9917b2bd9d4185c95c17d',
      'hash_10_VER_bf9926c2b2141684183bf54c53024c67',
      'hash_10_VER_93929e0d701da516c2c00b2a986f4afb',
      'hash_10_VER_61e3158871ff83b947fa61dd55bf0e62'
   ],
},
{  item   => 'DISK_sectors_written',
   color  => '53777A',
   task   => 'hash_09_VER_b5085578cca9a7fa280edef3196bbf53',
   type   => 'AREA',
   cdef   => 'Negate',
   hashes => [
      'hash_10_VER_f1b8a498e6aa39016e875946005468ca',
      'hash_10_VER_53f05855224d069625ee58c490ed1fb3',
      'hash_10_VER_4ac5653988f3493af2e4fa9550546a86',
      'hash_10_VER_43ca42b3dcd41d7cf16e2ef109931a0c'
   ],
},
```

You can see there's a one-to-one mapping between the items we're getting from the data source and the items we're putting onto the graph. In some special cases this isn't true, but it generally is; more on that in a minute. Each item has the following properties:

- *item* is the name of the data item to graph, as above.
- *color* is a hex color code. Try looking at http://www.colourlovers.com/palettes/top for some good ideas. Picking good colors is much harder than it seems.
- *task* is a hash; just copy/paste for now.
- *type* is the RRD display type, such as LINE1 or AREA or STACK.
- *cdef* is the optional name of a CDEF. 'Negate' is the most frequent one you'll see. This flips something across the Y axis. You can see that part of the graph grows up, and part of it is negated so it grows down.
- *hashes* is an array of hashes. Each hash will result in a bit of the caption being added to the graph. Depending on how many hashes are in the array, the graph will get varying bits of text below the picture. If you want a standard graph that has the label, current, average, maximum, and minimum value, put five hashes here. If you have only four, you'll get the label, current, average, and maximum; and so on.

There is a special case: ometimes you want to draw an item with an AREA in a light color, and then add a LINE1 with a darker color to give it a nice defined border. To do this, add the item with the AREA as in the examples above. After that, add the item again as a LINE1, but don't give it any hashes, so it doesn't get text captions on the graph.

If you put it all together, you'll get the full graph definition:

```
{  name       => 'Disk Sectors Read/Written',
   base_value => '1024',
   hash       => 'hash_00_VER_9fad7377daacfd611dae46b14cc4f67e',
   override   => { 'title' => 1 },
   dt         => {
      hash       => 'hash_01_VER_67811065b100a543ddeadf7464ae017c',
      input      => 'Get Disk Stats',
      DISK_sectors_read             => {
         data_source_type_id => '3',
         hash => 'hash_08_VER_80929ee708f7755d09443d3d930a29cc',
      },
      DISK_sectors_written          => {
```

```
         data_source_type_id => '3',
         hash => 'hash_08_VER_f5d85616af1e03a679042978c938a7ee',
      },
   },
   items => [
      # Colors from
      # http://www.colourlovers.com/palette/694737/Thought_Provoking
      {  item   => 'DISK_sectors_read',
         color  => '542437',
         task   => 'hash_09_VER_38f255216fd118d6d88a46d42357323c',
         type   => 'AREA',
         hashes => [
            'hash_10_VER_7fe10cf273b9917b2bd9d4185c95c17d',
            'hash_10_VER_bf9926c2b2141684183bf54c53024c67',
            'hash_10_VER_93929e0d701da516c2c00b2a986f4afb',
            'hash_10_VER_61e3158871ff83b947fa61dd55bf0e62'
         ],
      },
      {  item   => 'DISK_sectors_written',
         color  => '53777A',
         task   => 'hash_09_VER_b5085578cca9a7fa280edef3196bbf53',
         type   => 'AREA',
         cdef   => 'Negate',
         hashes => [
            'hash_10_VER_f1b8a498e6aa39016e875946005468ca',
            'hash_10_VER_53f05855224d069625ee58c490ed1fb3',
            'hash_10_VER_4ac5653988f3493af2e4fa9550546a86',
            'hash_10_VER_43ca42b3dcd41d7cf16e2ef109931a0c'
         ],
      },
   ],
},
```

Fix Your Hashes

After you're done with the above steps, you have everything you need to create templates. One thing remains: you need to resolve the duplication you created by copy/pasting hash values all over the place. There's a tool to do this. Run it like this:

```
$ pmp-cacti-make-hashes definitions/gnu_linux.def > temp.def
```

Now examine the generated file `temp.def` and make sure it is okay. You can use `vimdiff` to compare it to the original definitions file. Ensure that you pasted the new hashes *below* where you copied them from. *Hashes of the definition elements that pre-dated your work should never be changed!* If they are, they can cause problems with existing Cacti installations. The `diff` or `vimdiff` should reveal that new lines were added, but no old lines were changed. After you verify that, you can replace the original file with the `temp.def` file.

If you are creating a new definitions file based on an existing one, you can use the `--refresh` option to replace all hashes.

Generate Templates

Now you're ready to generate templates from your definition file. Here's how:

```
$ pmp-cacti-template --script ss_get_by_ssh.php unix.def > template.xml
```

At this point, the generated template file should be ready to import and use.

Optional Template Elements

Skip this unless you're an advanced user.

You can define these as children of the top level in the template definition:

- gprints – Custom sprintf formats. You don't need to modify these.
- rras – these are just some custom RRA definitions so you can keep more than the usual amount of data.
- cdefs – these are custom CDEF sections, which generally don't need to be modified.

If you don't define these, built-in defaults are used. They're kept in the `pmp-cacti-template` script.

Cacti Template Helper Tools

Aside from the tools documented elsewhere, there are the following tools:

pmp-cacti-graph-defs

This tool helps you make boilerplate text to plug into a definitions file. If you create the definitions file and add the data input as suggested in the documentation on creating graphs, then you can take the input items and copy them into a new file. Then group them into paragraphs in the order you want the graphs to be created. Leave one blank line between each paragraph. For example:

```
OPVZ_kmemsize_held        => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_kmemsize_failcnt     => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',

OPVZ_lockedpages_held     => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_lockedpages_failcnt  => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_privvmpages_held     => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_privvmpages_failcnt  => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_shmpages_held        => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_shmpages_failcnt     => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_physpages_held       => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_physpages_failcnt    => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_vmguarpages_held     => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_vmguarpages_failcnt  => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_oomguarpages_held    => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
OPVZ_oomguarpages_failcnt => 'hash_07_VER_2442551920abf4f05121043fe4cd51d7',
```

That's a sample used to create the OpenVZ graphs. It will create boilerplate for two graphs.

Then run this script with that file as input. The output will be generic text that's ready to paste into the `graphs` section of your definitions file. You'll need to edit all of the things like the data input name, the name of each file, the colors, data types, and so on. But this can be a fast way to get started.

When you're done, make sure you generate new hashes for everything, or you'll re-use hashes from another template. The best way to do this is to run `pmp-cacti-make-hashes` with the `--refresh` option on that to generate all new hashes.

Hardening Cacti setup

By default, the Cacti setup is closed from accessing from Web. Here is an excerpt from `/etc/httpd/conf.d/cacti.conf`:

```
<Directory /usr/share/cacti/>
     <IfModule mod_authz_core.c>
             # httpd 2.4
             Require host localhost
     </IfModule>
     <IfModule !mod_authz_core.c>
             # httpd 2.2
             Order deny,allow
             Deny from all
             Allow from localhost
     </IfModule>
</Directory>
```

In order, to access the Cacti web interface, most likely, you will be changing this configuration. Commenting out Deny/Require statements will open the Cacti to the local network or Internet. This will create **a potential vulnerability to disclose MySQL password** contained in scripts under the directory `/usr/share/cacti/scripts/`, in particular `/usr/share/cacti/scripts/ss_get_mysql_stats.php` and `/usr/share/cacti/scripts/ss_get_mysql_stats.php.cnf`, when trying to access them from Web.

Unfortunately, the folder `/usr/share/cacti/scripts/` is not closed by default as it is done with `/usr/share/cacti/log/` and `/usr/share/cacti/rra/` directories.

We strongly recommend to close any access from the web for these additional directories or files:

- /usr/share/cacti/scripts/
- /usr/share/cacti/site/scripts/ (for Debian systems)
- /usr/share/cacti/cli/
- /usr/share/cacti/.boto

Here is an example of httpd configuration that can harden your setup (goes to `/etc/httpd/conf.d/cacti.conf`):

```
<Directory ~ "/usr/share/cacti/(log/|rra/|scripts/|site/scripts/|cli/|\.ssh/|\.boto|.
↪*\.cnf)">
     <IfModule mod_rewrite.c>
             Redirect 404 /
     </IfModule>
     <IfModule !mod_rewrite.c>
             <IfModule mod_authz_core.c>
                     Require all denied
             </IfModule>
             <IfModule !mod_authz_core.c>
                     Order deny,allow
                     Deny from all
             </IfModule>
     </IfModule>
</Directory>
```

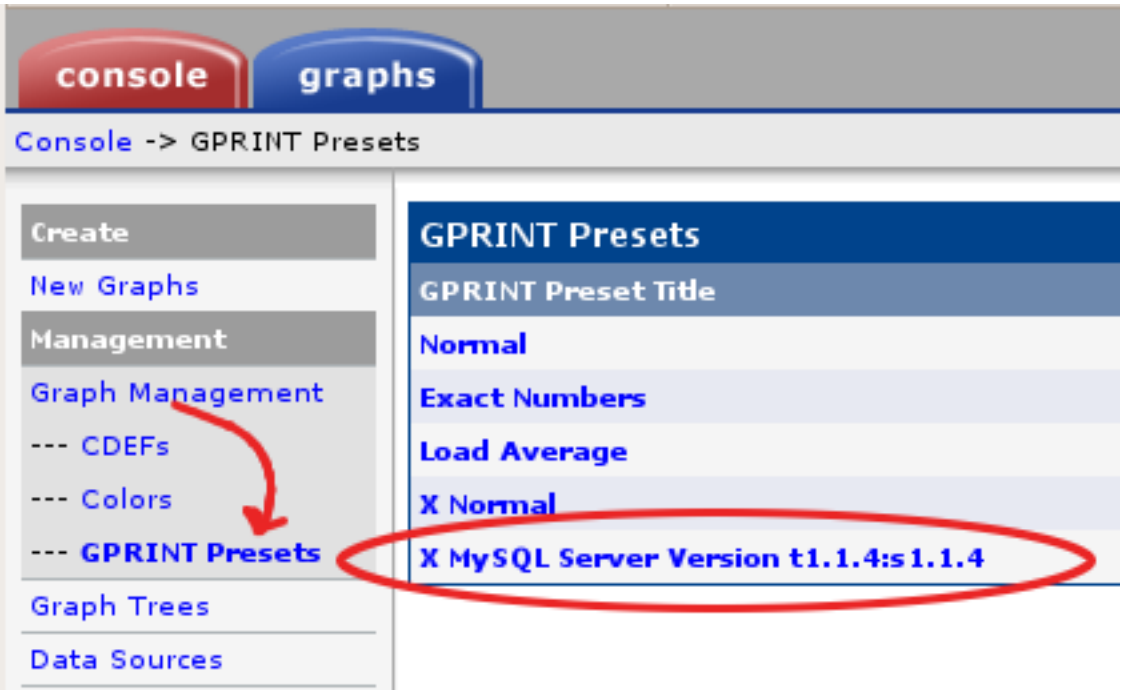
Even if you fully password-protected your Cacti installation using HTTP authentication, it is still recommended to double-secure the directories and files listed above.

Outlining the basic rules:

- keep your PHP config files `ss_get_mysql_stats.php.cnf` and `ss_get_by_ssh.php.cnf` outside the web directory `/usr/share/cacti/scripts/`. The recommended location is `/etc/cacti/`.
- do not put any SSH keys under cacti user home directory which is still the web directory.
- avoid placing `.boto` file under `~cacti/`, use `/etc/boto.cfg` instead (that's for RDS plugins).

Upgrading Percona Monitoring Plugins for Cacti

Upgrading is normally a simple process. Before you begin, find the version of the templates and scripts that is currently installed. You can find this as a GPRINT item, as in the following screenshot:



This shows that the MySQL templates installed were generated from version 1.1.4 of the templates, against version 1.1.4 of the PHP script file.

Check the installed scripts for their version:

```
# grep ^.version /path/to/ss_get_mysql_stats.php
$version = "1.1.4";
```

To upgrade Percona Cacti scripts simply overwrite ss_get_*.php files from a new tarball or update the package:

```
yum update percona-cacti-templates
```

or:

```
apt-get install percona-cacti-templates
```

Afterwards, re-import templates into Cacti using the web interface or from the command line, e.g.:

```
php /usr/share/cacti/cli/import_template.php --filename=/usr/share/cacti/resource/
↪percona/templates/cacti_host_template_percona_gnu_linux_server_ht_0.8.6i-sver1.0.3.
↪xml \
--with-user-rras='1:2:3:4'
```

Then rebuild the poller cache under Cacti -> System Utilities.

If any special upgrade steps are necessary, the changelog will explain them.

CHAPTER

THREE

TEMPLATES FOR ZABBIX

Zabbix is an enterprise-class open source distributed monitoring solution for networks and applications.

Percona Monitoring Plugins for Zabbix

These templates are mainly adopted from the existing *Cacti ones*. Currently, only MySQL template is available.

There are a few major differences between Cacti and Zabbix templates:

- Zabbix does not support negative Y axis, that's why you can see the negative values on some graphs to work out that - consider them the same as positive ones.
- Zabbix does not support stacked graph items with mixed draw styles, that's why some graphs may not look so nice like in Cacti as the stacks are replaced with lines.

Other Zabbix specific points:

- The items are populated by polling Zabbix agent.
- There are predefined triggers available to use.
- There is a screen as a placeholder for all graphs.
- 300 sec. polling interval - like with Cacti, the existing PHP script is used to retrive and cache MySQL metrics except some trigger-specific items. Due to the caching of results, PHP script runs only once per period.

System Requirements

- Zabbix version 2.0.x. The actual testing has been done on the version 2.0.9.
- Zabbix agent, php, php-mysql packages on monitored node.

Installation Instructions

Configure Zabbix Agent

1. Install the package from Percona Software Repositories:

   ```
   yum install percona-zabbix-templates
   ```

   or:

```
apt-get install percona-zabbix-templates
```

It will place files under `/var/lib/zabbix/percona/`. Alternatively, you can grab the tarball and copy folders `zabbix/scripts/` and `zabbix/templates/` into `/var/lib/zabbix/percona/`. See below for the URL.

2. Copy Zabbix Agent config:

```
mkdir -p /etc/zabbix_agentd.conf.d/
cp /var/lib/zabbix/percona/templates/userparameter_percona_mysql.conf /etc/zabbix_
↪agentd.conf.d/userparameter_percona_mysql.conf
```

3. Ensure /etc/zabbix_agentd.conf contains the line: `Include=/etc/zabbix_agentd.conf.d/`

4. Restart Agent:

```
service zabbix-agent restart
```

Configure MySQL connectivity on Agent

On this step we need to configure and verify MySQL connectivity with localhost on the Agent node.

1. Create .cnf file `/var/lib/zabbix/percona/scripts/ss_get_mysql_stats.php.cnf` as described at *configuration file*

   Example:

```
<?php
$mysql_user = 'root';
$mysql_pass = 's3cret';
```

2. Test the script:

```
[root@centos6 main]# /var/lib/zabbix/percona/scripts/get_mysql_stats_wrapper.sh gg
405647
```

   Should return any number. If the password is wrong in .cnf file, you will get something like:

```
[root@centos6 ~]# /var/lib/zabbix/percona/scripts/get_mysql_stats_wrapper.sh gg
ERROR: run the command manually to investigate the problem: /usr/bin/php -q /var/
↪lib/zabbix/percona/scripts/ss_get_mysql_stats.php --host localhost --items gg
[root@centos6 ~]# /usr/bin/php -q /var/lib/zabbix/percona/scripts/ss_get_mysql_
↪stats.php --host localhost --items gg
ERROR: Can't connect to local MySQL server through socket '/var/lib/mysql/mysql.
↪sock' (2)[root@centos6 ~]#
```

3. Configure ~zabbix/.my.cnf

   Example:

```
[client]
user = root
password = s3cret
```

4. Test the script:

```
[root@centos6 ~]# sudo -u zabbix -H /var/lib/zabbix/percona/scripts/get_mysql_
↪stats_wrapper.sh running-slave
0
```

Should return 0 or 1 but not the "Access denied" error.

Configure Zabbix Server

1. Grab the latest tarball from the Percona Software Downloads directory to your desktop.
2. Unpack it to get `zabbix/templates/` folder.
3. Import the XML template using Zabbix UI (Configuration -> Templates -> Import) by additionally choosing "Screens".
4. Create/edit hosts by assigning them "Percona Templates" group and linking the template "Percona MySQL Server Template" (Templates tab).

You are done.

Support Options

If you have questions, comments, or need help with the plugins, there are several options to consider.

You can get self-service help via Percona's forums, or the Percona mailing list.

You can report bugs and submit patches to the Launchpad project.

CHAPTER

FOUR

CHANGELOG

2018-02-05: version 1.1.8

- Add MySQL 5.7 support
- Changed canary check to use timestamp.now() and return a timedelta.seconds
- Remove an additional condition for the Dictionary memory allocated
- Fixed a *false-positive* problem when the calculated delay was less than 0 and the -m was not set.
- Fixed the problem where slaves would alert due to deadlocks on the master.
- If using pt-heartbeat, get_slave_status was only called when the -s option is set to *MASTER*
- Disabled UNK alerts by default (it is possible to enable them explicitly).
- A fix was added for MySQL Multi-Source replication.
- The graph *Percona InnoDB Memory Allocation* showed zeroes for the metrics *Total memory* (data source item nl) and *Dictionary memory* (data source item nm) when used for MySQL 5.7.18, because the syntax of SHOW ENGINE INNODB STATUS has changed in MySQL 5.7 (see https://dev.mysql.com/doc/refman/5.7/en/innodb-standard-monitor.html).
- The graph *Percona InnoDB I/O Pending* showed *NaN* for the metrics *Pending Log Writes* (data source item hn) and *Pending Chkp Writes* (data source item hk) when used for MySQL 5.7.18, because the syntax of SHOW ENGINE INNODB STATUS has changed in MySQL 5.7 (see https://dev.mysql.com/doc/refman/5.7/en/innodb-standard-monitor.html).
- Added server @@hostname as a possible match to avoid DNS lookups whilst allowing hostname-match.

2016-12-09: version 1.1.7

- New Nagios script pmp-check-mongo.py for MongoDB.
- Added MySQL socket and flag options to Cacti php script.
- Added disk volume check on "Mounted on" in addition to "Filesystem" to Cacti php script to allow monitoring of tmpfs mounts.
- Allow delayed slave to have SQL thread stopped on pmp-check-mysql-replication-delay check.
- Fix for –unconfigured flag of pmp-check-mysql-replication-delay.
- Fix for max_duration check of pmp-check-mysql-innodb when system and MySQL timezones mismatch.
- Fix rare nrpe broken pipe error on pmp-check-unix-memory check.

- Updated package spec files.

2016-01-12: version 1.1.6

- Added new RDS instance classes to RDS scripts.
- Added boto profile support to RDS scripts.
- Added AWS region support and ability to specify all regions to RDS scripts.
- Added ability to set AWS region and boto profile on data source level in Cacti.
- Added period, average time and debug options to pmp-check-aws-rds.py.
- Added ability to override Nginx server status URL path on data source level in Cacti.
- Made Memcached and Redis host configurable for Cacti script.
- Added the ability to lookup the master's server_id when using pt-heartbeat with pmp-check-mysql-replication-delay.
- Changed how memory stats are collected by Cacti script and pmp-check-unix-memory. Now /proc/meminfo is parsed instead of running *free* command. This also fixes pmp-check-unix-memory for EL7.
- Set default MySQL connect timeout to 5s for Cacti script. Can be overridden in the config.
- Fixed innodb transactions count on the Cacti graph for MySQL 5.6 and higher.
- Fixed –login-path option in Nagios scripts when using it along with other credential options.

Thanks to contributors: David Andruczyk, Denis Baklikov, Mischa ter Smitten, Mitch Hagstrand.

2015-06-22: version 1.1.5

- Added more DB instance classes to pmp-check-aws-rds.py (issue 1398911)
- Added configurable query period and average time to pmp-check-aws-rds.py (issue 1436943)
- Added region support to pmp-check-aws-rds.py (issue 1442980)
- Added an option to alert when server is not configured as replica to pmp-check-mysql-replication-delay (issue 1357017)
- Added an option to specify master connection to monitor MariaDB multi-source replication
- Improved usage of lock-free SHOW SLAVE STATUS query (issue 1380690)
- Fixed reporting of slave lag in ss_get_mysql_stats.php (issue 1389769)

2014-07-21: version 1.1.4

- Added login-path support to Nagios plugins with MySQL client 5.6 (bug 1338549)
- Added a new threshold option for delayed slaves to pmp-check-mysql-replication-delay (bug 1318280)
- Added delayed slave support to pmp-check-mysql-replication-running (bug 1332082)
- Updated Nagios plugins and Cacti script to leverage lock-free SHOW SLAVE STATUS in Percona Server (bug 1297442)

- Fixed pmp-check-mysql-replication-delay integer-float issue with MariaDB and MySQL 5.6 (bugs 1245934, 1295795)
- ss_get_rds_stats.py was not installed with 755 permissions from the package (bug 1316943)
- Cacti MySQL template item "handler_savepoint_rollback" was GAUGE type instead of DERIVE (bug 1334173)
- Fixed Zabbix running-slave check issue on some Debian systems (bug 1310723)
- Fixed paths in percona-cacti-templates package (bug 1349564)

2014-03-21: version 1.1.3

- Introduced more secure location of PHP script configs to harden a Cacti setup (bug #1295006)
- Addressed CVE-2014-2569

2014-03-14: version 1.1.2

- Added Nagios plugin and Cacti template for Amazon RDS
- Added Nagios config template to the documentation
- Added an option to pmp-check-pt-table-checksum to check MAX(ts) of latest checksum
- Added generic Nagios plugin for PT tables
- Extended pmp-check-mysql-processlist with max user connections check
- Zabbix MySQL.running-slave item failed with MySQL 5.6 (bug 1272358)
- ss_get_mysql_stats and MariaDB does not use have_response_time_distribution (bug 1285888)
- Cacti Monitoring plugins and SNMP via TCP (bug 1268552)

2013-12-30: version 1.1.1

- Cacti mysql graphs stop working with data input field "server-id" after 1.1 upgrade (bug 1264814)
- Non-integer poller errors for MySQL Query Response Time (bug 1264353)

2013-12-16: version 1.1

- Added MySQL template for Zabbix 2.0.x (first release)
- Added FreeBSD support to Nagios plugins, partially rewritten pmp-check-unix-memory (bugs 1249575, 1244081)
- Added new options to ss_get_mysql_stats.php to better support pt-heartbeat (bugs 1253125, 1253130)
- ss_get_mysql_stats.php script was opening multiple connections to the server (bug 1255371)
- sql query for idle_blocker_duraction check in pmp-check-mysql-innodb did not conform sql mode of ONLY_FULL_GROUP_BY (bug #1240417)

2013-10-02: version 1.0.5

- Added mysql-ca option to ss_get_mysql_stats.php (bug 1213857)
- Added user info to the idle_blocker_duration check of pmp-check-mysql-innodb (bug 1215317)
- Extended pmp-check-mysql-processlist with more locking states (bug 1213859)
- ss_get_mysql_stats.php did not work with custom mysql port (bug 1213862)
- ss_get_mysql_stats.php silently failed when a query returns too many rows (bug 1225070)
- Wrong description of percona-cacti-templates deb package (bug 1217782)

2013-07-22: version 1.0.4

- Added Galera/MySQL Monitoring Template for Cacti
- Added "Disk Read/Write Time per IO Request (ms)" graph
- Added "MySQL InnoDB Buffer Pool Efficiency" graph
- Switched ss_get_mysql_stats.php to PHP MySQLi extension and made it working with SSL (bug 1193097)
- Added user info to the max_duration check of pmp-check-mysql-innodb plugin (bug 1185513)
- ss_get_mysql_stats.php default values for '$status' array were null instead of 0 (bug 1070268)
- Introduction of innodb_read_views_memory overrode the InnoDB total memory allocated output in Cacti (bug 1188519)
- ss_get_by_ssh.php parsed MongoDB counters incorrectly when replica is set (bug 1087073)
- Cacti graph "Redis Unsaved Changes" was empty for Redis 2.6 (bug 1110372)
- Comparison of status variables that are strings didn't work with pmp-check-mysql-status (bug 1191305)
- pmp-check-mysql-processlist always showed 0 for "copy to table" counter (bug 1197084)
- percona-nagios-plugins package failed to install on Debian Squeeze when debsums is installed (bug 1194757)

2013-04-17: version 1.0.3

- MySQL 5.6 compatibility for InnoDB graphs (bug 1124292)
- Added performance data to Nagios plugins (bugs 1090145, 1102687)
- Added UTC option to pmp-check-mysql-replication-delay to be compatible with pt-hearbeat 2.1.8+ (bug 1103364)
- Added 1-second granularity to pmp-check-mysql-deadlocks (bug 1154774)
- Added package install/update instructions and other documentation updates (bugs 1139652, 1124200, 1015981)
- Updated documentation with the new Cacti sample images
- Updated "Network Traffic" to be blue and green and to show bits/sec (bug 1132900)
- Extended "MySQL Threads" graph with all kind of threads (bug 1157911)
- Some Cacti single-item graphs were broken due to cacti hexadecimal transformation (bug 1155513)

- Memcached graphs were broken when the wrong arguments for nc command are passed (bug 1155712)
- ss_get_by_ssh.php didn't gather mongodb stats without SSH (bug 1050537)
- ss_get_by_ssh.php didn't timeout commands that hang (bug 1160611)
- pmp-check-file-privs didn't throw the proper error on directory permissions issue (bug 1024001)
- pmp-check-mysql-replication-running reported OK when a slave is in "Connecting" state (bug 1089506)

Update note: Cacti templates have to be re-imported together with the updating of ss_get_*.php scripts. Then make sure you rebuilt the poller cache under Cacti -> System Utilities. Also the following Cacti graphs need to be recreated: MySQL "MySQL Threads", Linux "Network Traffic".

2013-02-15: version 1.0.2

- Created Debian and RPM packages
- Added "Disk IOPS" graph to Cacti Linux Templates
- Added an option to pmp-check-unix-memory to disable overriding the status based on the largest process in memory (bug 1052368)
- Added '!=' comparison operator to pmp-check-mysql-status (bug 1048627)
- pmp-check-mysql-replication-delay didn't alert if second behing master is NULL (bug 1040528)
- pmp-check-pt-table-checksum reported OK when checksums table does not exist (bug 1114425)
- pmp-check-pt-table-checksum threw "Bad substitution" error on Debian (bug 1071802)
- Minor updates to the documentation (bugs 1021855, 1014814, 1125233)

2012-06-12: version 1.0.1

- Cacti debug logs had a 12-hour timestamp instead of 24-hour (bug 973320).
- Nagios checks didn't remove temporary files on some platforms (bug 977514).
- Nagios plugins didn't use enough XXX in mktemp patterns (bug 2022766).
- Nagios check for deleted files didn't return 0 on success (bug 1009751).
- Nagios check for long-running txns didn't show thread ID (bug 1011625).

2012-04-02: version 1.0.0

- Load average was 15-minute instead of 1-minute (bug 968604).
- MySQL connection errors weren't logged (bug 958782).
- The pmp-check-mysql-deadlocks referred to pt-heartbeat (bug 934255).
- HTTP status was not fetched correctly when $use_ssh was disabled (bug 937017).
- MongoDB didn't support –port2 (bug 937018).
- Percona Server response time graph didn't tolerate <> 14 rows (bug 954118).
- The release tarball had an extra directory (bug 934227).

- pmp-check-mysql-status didn't remove one of its temp files (bug 959425).

2012-02-16: version 0.9.0

- Initial release. Not backwards compatible with Better Cacti Templates.