NAME

rrdcached – Data caching daemon for rrdtool

SYNOPSIS

rrdcached [**–a** *alloc_size*] [**–b** *base_dir* [**–B**]] [**–F**] [**–f** *timeout*] [**–G** *group*]] [**–g**] [**–j** *journal_dir*] [**–L**] [**–l** *address*] [**–m** *mode*] [**–O**] [**–o** *log_file*] [**–P** *permissions*] [**–p** *pid_file*] [**–R**] [**–s** *group*] [**–t** *write_threads*] [**–U** *user*]] [**–V** *log_level*] [**–w** *timeout*] [**–z** *delay*]

DESCRIPTION

rrdcached is a daemon that receives updates to existing RRD files, accumulates them and, if enough have been received or a defined time has passed, writes the updates to the RRD file. A *flush* command may be used to force writing of values to disk, so that graphing facilities and similar can work with up-to-date data.

The daemon was written with big setups in mind. Those setups usually run into IO related problems sooner or later for reasons that are beyond the scope of this document. Check the wiki at the RRDtool homepage for details. Also check ''SECURITY CONSIDERATIONS'' below before using this daemon! A detailed description of how the daemon operates can be found in the ''HOW IT WORKS'' section below.

OPTIONS

–l *address*

Tells the daemon to bind to *address* and accept incoming TCP connections on that socket. If *address* begins with `unix:`, everything following that prefix is interpreted as the path to a UNIX domain socket. Otherwise the address or node name are resolved using `getaddrinfo()`.

For network sockets, a port may be specified by using the form **`[`***`address`***`]:`***`port`*. If the address is an IPv4 address or a fully qualified domain name (i. e. the address contains at least one dot (`.`)), the square brackets can be omitted, resulting in the (simpler) *`address`*`:`*`port`* pattern. The default port is **42217**. If you specify a network socket, it is mandatory to read the ''SECURITY CONSIDERATIONS'' section.

The following formats are accepted. Please note that the address of the UNIX domain socket **must** start with a slash in the second case!

```
unix:</path/to/unix.sock>
/<path/to/unix.sock>
<hostname-or-ip>
[<hostname-or-ip>]:<port>
<hostname-or-ipv4>:<port>
```

Given a port without a host (e.g. `-l :42217`) the daemon will listen on that port on all network interfaces. Use `-L` to avoid the need to explicitly provide the port if the default port is desired.

If no **–l** option is not specified the default address, `unix:/tmp/rrdcached.sock`, will be used. Multiple **–l** options may be provided.

–L Tells the daemon to bind to the default TCP port on all available interfaces. It is equivalent to `-l ''` without the confusion of the empty string parameter.

–s *group_name*|*gid*

Set the group permissions of a UNIX domain socket. The option accepts either a numeric group id or group name. That group will then have both read and write permissions (the socket will have file permissions 0760) for the socket and, therefore, is able to send commands to the daemon. This may be useful in cases where you cannot easily run all RRD processes with the same user privileges (e.g. graph generating CGI scripts that typically run in the permission context of the web server).

This option affects the *following* UNIX socket addresses (the following **–l** options) or the default socket (if no **–l** options have been specified), i.e., you may specify different settings for different sockets.

The default is not to change ownership or permissions of the socket and, thus, use the system default.

−m *mode*

Set the file permissions of a UNIX domain socket. The option accepts an octal number representing the bit pattern for the mode (see **chmod** (1) for details).

Please note that not all systems honor this setting. On Linux, read/write permissions are required to connect to a UNIX socket. However, many BSD-derived systems ignore permissions for UNIX sockets. See **unix** (7) for details.

This option affects the *following* UNIX socket addresses (the following **−l** options) or the default socket (if no **−l** options have been specified), i.e., you may specify different settings for different sockets.

The default is not to change ownership or permissions of the socket and, thus, use the system default.

−P *command*[,*command*[,...]]

Specifies the commands accepted via both a network and a UNIX socket. This allows administrators of *RRDCacheD* to control the actions accepted from various sources.

The arguments given to the **−P** option is a comma separated list of commands. For example, to allow one the `FLUSH` and `PENDING` commands one could specify:

```
rrdcached -P FLUSH,PENDING $MORE_ARGUMENTS
```

The **−P** option affects the *following* socket addresses (the following **−l** options) or the default socket (if no **−l** options have been specified). In the following example, only the IPv4 network socket (address `10.0.0.1`) will be restricted to the `FLUSH` and `PENDING` commands:

```
rrdcached -l unix:/some/path -P FLUSH,PENDING -l 10.0.0.1
```

A complete list of available commands can be found in the section ''Valid Commands'' below. There are two minor special exceptions:

- The `HELP` and `QUIT` commands are always allowed.
- If the `BATCH` command is accepted, the **.** command will automatically be accepted, too.

Please also read ''SECURITY CONSIDERATIONS'' below.

−V *log_level*

rrdcached under load can severely flood the logs. This command line option specifies the maximum log_level to be used, meaning that a message with verbosity *higher* than log_level is muted (LOG_EMERG being the lowest and LOG_DEBUG highest).

Accepted values for ''log_level'' (lowest to highest verbosity): LOG_EMERG, LOG_ALERT, LOG_CRIT, LOG_ERR, LOG_WARNING, LOG_NOTICE, LOG_INFO, LOG_DEBUG

Default log level when this flag is *NOT* present: **LOG_ERR**

See also: *syslog.h*

−o *log_file*

Log to the given file instead of syslog.

−w *timeout*

Data is written to disk every *timeout* seconds. An optional suffix may be used (e.g. `5m` instead of `300` seconds). If this option is not specified the default interval of 300 seconds will be used.

−z *delay*

If specified, rrdcached will delay writing of each RRD for a random number of seconds in the range [0,*delay*). This will avoid too many writes being queued simultaneously. This value should be no greater than the value specified in **−w**. An optional suffix may be used (e.g. `3m` instead of `180` seconds). By default, there is no delay.

−f *timeout*
Every *timeout* seconds the entire cache is searched for old values which are written to disk. This only concerns files to which updates have stopped, so setting this to a high value, such as 3600 seconds, is acceptable in most cases. An optional suffix may be used (e.g. `1h` instead of `3600` seconds). This timeout defaults to 3600 seconds.

−p *file*
Sets the name and location of the PID-file. If not specified, the default, *`$localstatedir`*`/run/rrdcached.pid` will be used.

−t *write_threads*
Specifies the number of threads used for writing RRD files. The default is 4. Increasing this number will allow rrdcached to have more simultaneous I/O requests into the kernel. This may allow the kernel to re-order disk writes, resulting in better disk throughput.

−j *dir*
Write updates to a journal in *dir*. In the event of a program or system crash, this will allow the daemon to write any updates that were pending at the time of the crash.

On startup, the daemon will check for journal files in this directory. If found, all updates therein will be read into memory before the daemon starts accepting new connections.

The journal will be rotated with the same frequency as the flush timer given by **−f**.

When journaling is enabled, the daemon will use a fast shutdown procedure. Rather than flushing all files to disk, it will make sure the journal is properly written and exit immediately. Although the RRD data files are not fully up-to-date, no information is lost; all pending updates will be replayed from the journal next time the daemon starts up.

To disable fast shutdown, use the **−F** option.

−F ALWAYS flush all updates to the RRD data files when the daemon is shut down, regardless of journal setting.

−g Run in the foreground. The daemon will not **fork**().

−b *dir*
The daemon will change into a specific directory at startup. All files passed to the daemon, that are specified by a **relative** path, will be interpreted to be relative to this directory. If not given the default, `/tmp`, will be used.

```
+-----------------------+-----------------------+
! Command line           ! File updated          !
+-----------------------+-----------------------+
! foo.rrd                ! /tmp/foo.rrd          !
! foo/bar.rrd            ! /tmp/foo/bar.rrd      !
! /var/lib/rrd/foo.rrd   ! /var/lib/rrd/foo.rrd  !
+-----------------------+-----------------------+
Paths given on the command  line and paths actually
updated by the daemon,  assuming the base directory
"/tmp".
```

WARNING: The paths up to and including the base directory **MUST NOT BE** symbolic links. In other words, if the base directory is specified as:

```
-b /base/dir/somewhere
```

... then **NONE** of the following should be symbolic links:

```
/base
/base/dir
/base/dir/somewhere
```

−B Only permit writes into the base directory specified in **−b** (and any sub-directories). This does **NOT** detect symbolic links. Paths containing `../` will also be blocked.

−R Permit recursive subdirectory creation in the base directory specified in **−b** (and any sub-directories). Can only be used when **−B** is also set.

−a *alloc_size*
Allocate value pointers in chunks of *alloc_size*. This may improve CPU utilization on machines with slow `realloc()` implementations, in exchange for slightly higher memory utilization. The default is 1. Do not set this more than the **−w** value divided by your average RRD step size.

−O Prevent the CREATE command from overwriting existing files, even when it is instructed to do so. This is for added security.

−G *−group*
When running as daemon and invoked from a privileged account, reset group privileges to those of *group*. The group may be specified as a name or as a group ID. The daemon will exit with a diagnostic if it cannot successfully transition to the specified group.

−U *−user*
When running as daemon and invoked from a privileged account, reset user privileges to those of *user*. The user may be specified as a name or as a user ID. The daemon will exit with a diagnostic if it cannot successfully transition to the specified user.

AFFECTED RRDTOOL COMMANDS

The following commands may be made aware of the **rrdcached** using the command line argument **−−daemon** or the environment variable **RRDCACHED_ADDRESS**:

- dump
- fetch
- flush
- graph
- graphv
- info
- first
- last
- lastupdate
- update
- xport
- create
- list

The **update** command can send values to the daemon instead of writing them to the disk itself. All other commands can send a **FLUSH** command (see below) to the daemon before accessing the files, so they work with up-to-date data even if the cache timeout is large.

ERROR REPORTING

The daemon reports errors in one of two ways: During startup, error messages are printed to `STDERR`. One of the steps when starting up is to fork to the background and closing `STDERR` − after this writing directly to the user is no longer possible. Once this has happened, the daemon will send log messages to the system logging daemon using **syslog** (3). The facility used is `LOG_DAEMON`.

HOW IT WORKS

When receiving an update, **rrdcached** does not write to disk but looks for an entry for that file in its internal tree. If not found, an entry is created including the current time (called ‘‘First’’ in the diagram below). This time is **not** the time specified on the command line but the time the operating system considers

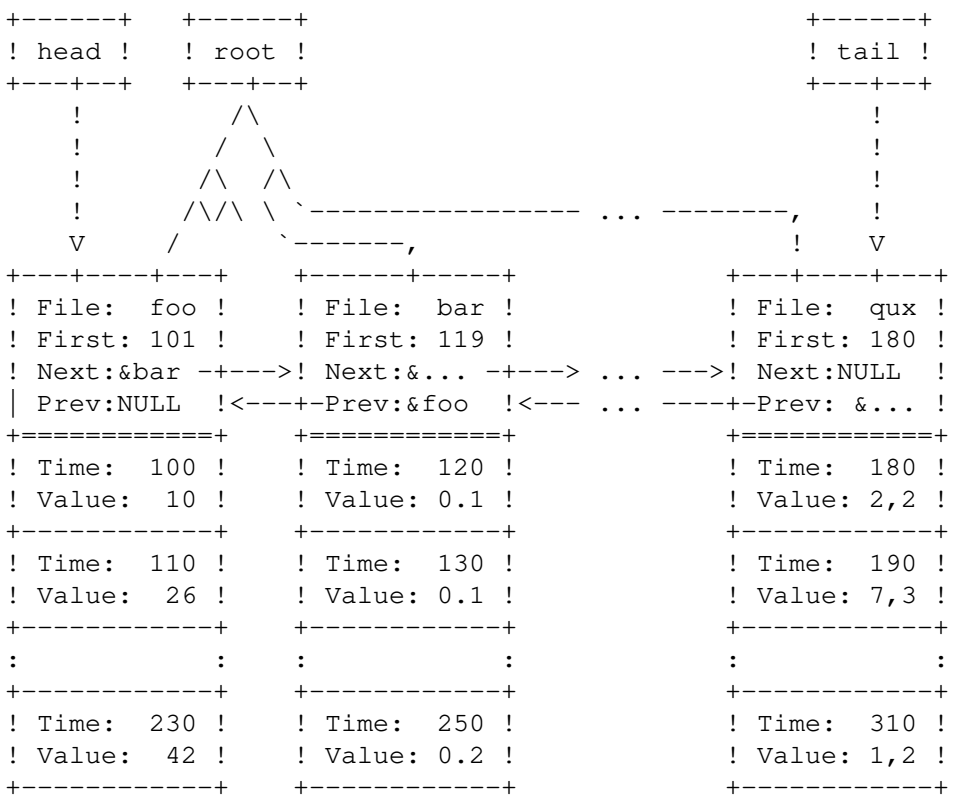
to be ‘‘now’’. The value and time of the value (called ‘‘Time’’ in the diagram below) are appended to the tree node.

When appending a value to a tree node, it is checked whether it’s time to write the values to disk. Values are written to disk if `now() - First >= timeout`, where `timeout` is the timeout specified using the **−w** option, see ‘‘OPTIONS’’. If the values are ‘‘old enough’’ they will be enqueued in the ‘‘update queue’’, i. e. they will be appended to the linked list shown below. Because the tree nodes and the elements of the linked list are the same data structures in memory, any update to a file that has already been enqueued will be written with the next write to the RRD file, too.

A separate ‘‘update thread’’ constantly dequeues the first element in the update queue and writes all its values to the appropriate file. So as long as the update queue is not empty files are written at the highest possible rate.

Since the timeout of files is checked only when new values are added to the file, ‘‘dead’’ files, i. e. files that are not updated anymore, would never be written to disk. Therefore, every now and then, controlled by the **−f** option, the entire tree is walked and all ‘‘old’’ values are enqueued. Since this only affects ‘‘dead’’ files and walking the tree is relatively expensive, you should set the ‘‘flush interval’’ to a reasonably high value. The default is 3600 seconds (one hour).

The downside of caching values is that they won’t show up in graphs generated from the RRD files. To get around this, the daemon provides the ‘‘flush command’’ to flush specific files. This means that the file is inserted at the **head** of the update queue or moved there if it is already enqueued. The flush command will return only after the file’s pending updates have been written to disk.



The above diagram demonstrates:

- Files/values are stored in a (balanced) tree.
- Tree nodes and entries in the update queue are the same data structure.
- The local time (‘‘First’’) and the time specified in updates (‘‘Time’’) may differ.

- Timed out values are inserted at the ‘‘tail’’.
- Explicitly flushed values are inserted at the ‘‘head’’.
- ASCII art rocks.

SECURITY CONSIDERATIONS

Authentication

If your rrdtool installation was built without libwrap there is no form of authentication for clients connecting to the rrdcache daemon!

If your rrdtool installation was built with libwrap then you can use hosts_access to restrict client access to the rrdcache daemon (rrdcached). For more information on how to use hosts_access to restrict access to the rrdcache daemon you should read the **hosts_access** (5) man pages.

It is still highly recommended to install a packet filter or similar mechanism to prevent unauthorized connections. Unless you have a dedicated VLAN or VPN for this, using network sockets is probably a bad idea!

Authorization

There is minimal per-socket authorization.

Authorization is currently done on a per-socket basis. That means each socket has a list of commands it will accept and it will accept. It will accept only those commands explicitly listed but it will (currently) accept these commands from anyone reaching the socket.

If the networking sockets are to be used, it is necessary to restrict the accepted commands to those needed by external clients. If, for example, external clients want to draw graphs of the cached data, they should only be allowed to use the `FLUSH` command.

Authorization does not work when rrdcached is socket-activated by systemd.

Encryption

There is no encryption.

Again, this may be added in the future, but for the time being it is your job to keep your private data private. Install a VPN or an encrypted tunnel if you statistics are confidential!

Sanity checking

There is no sanity checking.

The daemon will blindly write to any file it gets told, so you really should create a separate user just for this daemon. Also it does not do any sanity checks, so if it gets told to write values for a time far in the future, your files will be messed up good!

Conclusion

- Security is the job of the administrator.
- We recommend to allow write access via UNIX domain sockets only.
- You have been warned.

PROTOCOL

The daemon communicates with clients using a line based ASCII protocol which is easy to read and easy to type. This makes it easy for scripts to implement the protocol and possible for users to use telnet to connect to the daemon and test stuff ‘‘by hand’’.

The protocol is line based, this means that each record consists of one or more lines. A line is terminated by the line feed character `0x0A`, commonly written as `\n`. In the examples below, this character will be written as `<LF>` (‘‘line feed’’).

After the connection has been established, the client is expected to send a ‘‘command’’. A command consists of the command keyword, possibly some arguments, and a terminating newline character. For a list of commands, see ‘‘Valid Commands’’ below.

Example:

```
FLUSH /tmp/foo.rrd<LF>
```

The daemon answers with a line consisting of a status code and a short status message, separated by one or more space characters. A negative status code signals an error, a positive status code or zero signal success. If the status code is greater than zero, it indicates the number of lines that follow the status line.

Examples:

```
0 Success<LF>

2 Two lines follow<LF>
This is the first line<LF>
And this is the second line<LF>
```

Valid Commands

The following commands are understood by the daemon:

FLUSH *filename*

Causes the daemon to put *filename* to the **head** of the update queue (possibly moving it there if the node is already enqueued). The answer will be sent **after** the node has been dequeued.

FLUSHALL

Causes the daemon to start flushing ALL pending values to disk. This returns immediately, even though the writes may take a long time.

PENDING *filename*

Shows any ''pending'' updates for a file, in order. The updates shown have not yet been written to the underlying RRD file.

FETCH *filename CF* [*start* [*end*] [*ds ...*]]

Calls `rrd_fetch` with the specified arguments and returns the result in text form. If necessary, the file is flushed to disk first. The client side function `rrdc_fetch` (declared in `rrd_client.h`) parses the output and behaves just like `rrd_fetch_r` for easy integration of remote queries. ds defines the columns to dump – if none are given then all are returned

FETCHBIN *filename CF* [*start* [*end*] [*ds ...*]]

Calls `rrd_fetch` with the specified arguments and returns the result in text/binary form to avoid unnecessary un/marshalling overhead. If necessary, the file is flushed to disk first. The client side function `rrdc_fetch` (declared in `rrd_client.h`) parses the output and behaves just like `rrd_fetch_r` for easy integration of remote queries. ds defines the columns to dump – if none are given then all are returned

FORGET *filename*

Removes *filename* from the cache. Any pending updates **WILL BE LOST**.

QUEUE

Shows the files that are on the output queue. Returns zero or more lines in the following format, where <num_vals> is the number of values to be written for the <file>:

```
<num_vals> <file>
```

HELP [*command*]

Returns a short usage message. If no command is given, or *command* is **HELP**, a list of commands supported by the daemon is returned. Otherwise a short description, possibly containing a pointer to a manual page, is returned. Obviously, this is meant for interactive usage and the format in which the commands and usage summaries are returned is not well defined.

STATS

Returns a list of metrics which can be used to measure the daemons performance and check its status. For a description of the values returned, see ''Performance Values'' below.

The format in which the values are returned is similar to many other line based protocols: Each value is printed on a separate line, each consisting of the name of the value, a colon, one or more spaces and

the actual value.

Example:

```
9 Statistics follow
QueueLength: 0
UpdatesReceived: 30
FlushesReceived: 2
UpdatesWritten: 13
DataSetsWritten: 390
TreeNodesNumber: 13
TreeDepth: 4
JournalBytes: 190
JournalRotate: 0
```

PING

PING-PONG, this is very useful when using connection pool between user client and RRDCACHED.

Example:

```
0 PONG
```

UPDATE *filename values* [*values* ...]

Adds more data to a filename. This is **the** operation the daemon was designed for, so describing the mechanism again is unnecessary. Read ''HOW IT WORKS'' above for a detailed explanation.

Note that rrdcached only accepts absolute timestamps in the update values. Updates strings like ''N:1:2:3'' are automatically converted to absolute time by the RRD client library before sending to rrdcached.

WROTE *filename*

This command is written to the journal after a file is successfully written out to disk. It is used during journal replay to determine which updates have already been applied. It is *only* valid in the journal; it is not accepted from the other command channels.

FIRST *filename* [*rranum*]

Return the timestamp for the first CDP in the specified RRA. Default is to use RRA zero if none is specified.

LAST *filename*

Return the timestamp for the last update to the specified RRD. Note that the cache is *not* flushed before checking, as the client is expected to request this separately if it is required.

INFO *filename*

Return the configuration information for the specified RRD. Note that the cache is *not* flushed before checking, as the client is expected to request this separately if it is required.

The information is returned, one item per line, with the format:

```
I<keyname> I<type> I<value>
```

CREATE *filename* [−s *stepsize*] [−b *begintime*] [−r *sourcefile* ...] [−t *templatefile*] [−O] *DSdefinitions* ... *RRAdefinitions* ...

This will create the RRD file according to the supplied parameters, provided the parameters are valid, and (if the −O option is given or if the rrdcached was started with the −O flag) the specified *filename* does not already exist.

BATCH

This command initiates the bulk load of multiple commands. This is designed for installations with extremely high update rates, since it permits more than one command to be issued per **read()** and **write()**.

All commands are executed just as they would be if given individually, except for output to the user.

Messages indicating success are suppressed, and error messages are delayed until the client is finished.

Command processing is finished when the client sends a dot (“.”) on its own line. After the client has finished, the server responds with an error count and the list of error messages (if any). Each error messages indicates the number of the command to which it corresponds, and the error message itself. The first user command after **BATCH** is command number one.

```
client:  BATCH
server:  0 Go ahead.  End with dot '.' on its own line.
client:  UPDATE x.rrd 1223661439:1:2:3           <--- command #1
client:  UPDATE y.rrd 1223661440:3:4:5           <--- command #2
client:  and so on...
client:  .
server:  2 Errors
server:  1 message for command 1
server:  12 message for command 12
```

LIST [RECURSIVE] I/<path>

This command allows to list directories and rrd databases as seen by the daemon. The root “directory” is the base_dir (see ’−b dir’). When invoked with ’LIST RECURSIVE /<path>’ it will behave similarly to ’ls −R’ but limited to rrd files (listing all the rrd bases in the subtree of <path>, skipping empty directories).

SUSPEND *filename*

Suspend writing to an RRD file. While a file is suspended, all metrics for it are cached in memory until **RESUME** is called for that file or **RESUMEALL** is called.

RESUME *filename*

Resume writing to an RRD file previously suspended by **SUSPEND** or **SUSPENDALL**.

SUSPENDALL

Suspend writing to all RRD files. While a file is suspended, all metrics for it are cached in memory until **RESUME** is called for that file or **RESUMEALL** is called.

RESUMEALL

Resume writing to all RRD files previously suspended by **SUSPEND** or **SUSPENDALL**.

QUIT

Disconnect from rrdcached.

Performance Values

The following counters are returned by the **STATS** command:

QueueLength *(unsigned 64bit integer)*

Number of nodes currently enqueued in the update queue.

UpdatesReceived *(unsigned 64bit integer)*

Number of UPDATE commands received.

FlushesReceived *(unsigned 64bit integer)*

Number of FLUSH commands received.

UpdatesWritten *(unsigned 64bit integer)*

Total number of updates, i. e. calls to `rrd_update_r`, since the daemon was started.

DataSetsWritten *(unsigned 64bit integer)*

Total number of “data sets” written to disk since the daemon was started. A data set is one or more values passed to the **UPDATE** command. For example: `1223661439:123:456` is one data set with two values. The term “data set” is used to prevent confusion whether individual values or groups of values are counted.

TreeNodesNumber *(unsigned 64bit integer)*
Number of nodes in the cache.

TreeDepth *(unsigned 64bit integer)*
Depth of the tree used for fast key lookup.

JournalBytes *(unsigned 64bit integer)*
Total number of bytes written to the journal since startup.

JournalRotate *(unsigned 64bit integer)*
Number of times the journal has been rotated since startup.

SIGNALS

SIGINT and SIGTERM
The daemon exits normally on receipt of either of these signals. Pending updates are handled in accordance with the **−j** and **−F** options.

SIGUSR1
The daemon exits AFTER flushing all updates out to disk. This may take a while.

SIGUSR2
The daemon exits immediately, without flushing updates out to disk. Pending updates will be replayed from the journal when the daemon starts up again. **WARNING: if journaling (−j) is NOT enabled, any pending updates WILL BE LOST**.

BUGS

No known bugs at the moment.

SEE ALSO

rrdtool, rrdgraph

AUTHOR

Florian Forster <octo at verplant.org>

Both **rrdcached** and this manual page have been written by Florian.

CONTRIBUTORS

kevin brintnall <kbrint@rufus.net> Steve Shipway <steve@steveshipway.org> Martin Sperl <rrdtool@martin.sperl.org>