NAME

rrdtool − Round Robin Database Tool

SYNOPSIS

rrdtool − [workdir]| *function*

DESCRIPTION

OVERVIEW

It is pretty easy to gather status information from all sorts of things, ranging from the temperature in your office to the number of octets which have passed through the FDDI interface of your router. But it is not so trivial to store this data in an efficient and systematic manner. This is where **RRDtool** comes in handy. It lets you *log and analyze* the data you gather from all kinds of data-sources (**DS**). The data analysis part of RRDtool is based on the ability to quickly generate graphical representations of the data values collected over a definable time period.

In this man page you will find general information on the design and functionality of the Round Robin Database Tool (RRDtool). For a more detailed description of how to use the individual functions of **RRDtool** check the corresponding man page.

For an introduction to the usage of RRDtool make sure you consult the rrdtutorial.

FUNCTIONS

While the man pages talk of command line switches you have to set in order to make **RRDtool** work it is important to note that **RRDtool** can be remotely controlled through a set of pipes. This saves a considerable amount of startup time when you plan to make **RRDtool** do a lot of things quickly. Check the section on "REMOTE CONTROL" further down. There is also a number of language bindings for RRDtool which allow you to use it directly from Perl, python, Tcl, PHP, etc.

create Set up a new Round Robin Database (RRD). Check rrdcreate.

update Store new data values into an RRD. Check rrdupdate.

updatev Operationally equivalent to **update** except for output. Check rrdupdate.

graph Create a graph from data stored in one or several RRDs. Apart from generating graphs, data can also be extracted to stdout. Check rrdgraph.

graphv Create a graph from data stored in one or several RRDs. Same as graph, but metadata are printed before the graph. Check rrdgraph.

dump Dump the contents of an RRD in plain ASCII. In connection with restore you can use this to move an RRD from one computer architecture to another. Check rrddump.

restore Restore an RRD in XML format to a binary RRD. Check rrdrestore

fetch Get data for a certain time period from a RRD. The graph function uses fetch to retrieve its data from an RRD. Check rrdfetch.

tune Alter setup and structure of an RRD. Check rrdtune.

first Find the first update time of an RRD. Check rrdfirst.

last Find the last update time of an RRD. Check rrdlast.

lastupdate
Find the last update time of an RRD. It also returns the value stored for each datum in the most recent update. Check rrdlastupdate.

info Get information about an RRD. Check rrdinfo.

resize Change the size of individual RRAs. This is dangerous! Check rrdresize.

xport Export data retrieved from one or several RRDs. Check rrdxport.

flushcached
Flush the values for a specific RRD file from memory. Check rrdflushcached.

HOW DOES RRDTOOL WORK?

Data Acquisition

When monitoring the state of a system, it is convenient to have the data available at a constant time interval. Unfortunately, you may not always be able to fetch data at exactly the time you want to. Therefore **RRDtool** lets you update the log file at any time you want. It will automatically interpolate the value of the data-source (**DS**) at the latest official time-slot (interval) and write this interpolated value to the log. The original value you have supplied is stored as well and is also taken into account when interpolating the next log entry.

Consolidation

You may log data at a 1 minute interval, but you might also be interested to know the development of the data over the last year. You could do this by simply storing the data in 1 minute intervals for the whole year. While this would take considerable disk space it would also take a lot of time to analyze the data when you wanted to create a graph covering the whole year. **RRDtool** offers a solution to this problem through its data consolidation feature. When setting up an Round Robin Database (**RRD**), you can define at which interval this consolidation should occur, and what consolidation function (**CF**) (average, minimum, maximum, last) should be used to build the consolidated values (see rrdcreate). You can define any number of different consolidation setups within one **RRD**. They will all be maintained on the fly when new data is loaded into the **RRD**.

Round Robin Archives

Data values of the same consolidation setup are stored into Round Robin Archives (**RRA**). This is a very efficient manner to store data for a certain amount of time, while using a known and constant amount of storage space.

It works like this: If you want to store 1'000 values in 5 minute interval, **RRDtool** will allocate space for 1'000 data values and a header area. In the header it will store a pointer telling which slots (value) in the storage area was last written to. New values are written to the Round Robin Archive in, you guessed it, a round robin manner. This automatically limits the history to the last 1'000 values (in our example). Because you can define several **RRA**s within a single **RRD**, you can setup another one, for storing 750 data values at a 2 hour interval, for example, and thus keep a log for the last two months at a lower resolution.

The use of **RRA**s guarantees that the **RRD** does not grow over time and that old data is automatically eliminated. By using the consolidation feature, you can still keep data for a very long time, while gradually reducing the resolution of the data along the time axis.

Using different consolidation functions (**CF**) allows you to store exactly the type of information that actually interests you: the maximum one minute traffic on the LAN, the minimum temperature of your wine cellar, ... etc.

Unknown Data

As mentioned earlier, the **RRD** stores data at a constant interval. Sometimes it may happen that no new data is available when a value has to be written to the **RRD**. Data acquisition may not be possible for one reason or other. With **RRDtool** you can handle these situations by storing an **UNKNOWN** value into the database. The value '**UNKNOWN**' is supported through all the functions of the tool. When consolidating a data set, the amount of **UNKNOWN** data values is accounted for and when a new consolidated value is ready to be written to its Round Robin Archive (**RRA**), a validity check is performed to make sure that the percentage of unknown values in the data point is above a configurable level. If not, an **UNKNOWN** value will be written to the **RRA**.

Graphing

RRDtool allows you to generate reports in numerical and graphical form based on the data stored in one or several **RRD**s. The graphing feature is fully configurable. Size, color and contents of the graph can be defined freely. Check rrdgraph for more information on this.

Aberrant Behavior Detection
by Jake Brutlag

RRDtool provides the building blocks for near real-time aberrant behavior detection. These components include:

- An algorithm for predicting the value of a time series one time step into the future.
- A measure of deviation between predicted and observed values.
- A mechanism to decide if and when an observed value or sequence of observed values is *too deviant* from the predicted value(s).

Here is a brief explanation of these components:

The Holt-Winters time series forecasting algorithm is an on-line (or incremental) algorithm that adaptively predicts future observations in a time series. Its forecast is the sum of three components: a baseline (or intercept), a linear trend over time (or slope), and a seasonal coefficient (a periodic effect, such as a daily cycle). There is one seasonal coefficient for each time point in the period (cycle). After a value is observed, each of these components is updated via exponential smoothing. This means that the algorithm ''learns'' from past values and uses them to predict the future. The rate of adaptation is governed by 3 parameters, alpha (intercept), beta (slope), and gamma (seasonal). The prediction can also be viewed as a smoothed value for the time series.

The measure of deviation is a seasonal weighted absolute deviation. The term *seasonal* means deviation is measured separately for each time point in the seasonal cycle. As with Holt-Winters forecasting, deviation is predicted using the measure computed from past values (but only at that point in the seasonal cycle). After the value is observed, the algorithm learns from the observed value via exponential smoothing. Confidence bands for the observed time series are generated by scaling the sequence of predicted deviation values (we usually think of the sequence as a continuous line rather than a set of discrete points).

Aberrant behavior (a potential failure) is reported whenever the number of times the observed value violates the confidence bands meets or exceeds a specified threshold within a specified temporal window (e.g. 5 violations during the past 45 minutes with a value observed every 5 minutes).

This functionality is embedded in a set of related **RRAs**. In particular, a FAILURES **RRA** logs potential failures. With these data you could, for example, use a front-end application to **RRDtool** to initiate real-time alerts.

For a detailed description on how to set this up, see rrdcreate.

REMOTE CONTROL

When you start **RRDtool** with the command line option '−' it waits for input via standard input (STDIN). With this feature you can improve performance by attaching **RRDtool** to another process (MRTG is one example) through a set of pipes. Over these pipes **RRDtool** accepts the same arguments as on the command line and some special commands like **cd, mkdir, pwd, ls** and **quit**. For detailed help on the server commands type:

```
rrdtool help cd
```

When a command is completed, RRDtool will print the string 'OK', followed by timing information of the form **u:***usertime* **s:***systemtime*. Both values are the running totals of seconds since RRDtool was started. If an error occurs, a line of the form 'ERROR: *Description of error*' will be printed instead. **RRDtool** will not abort, unless something really serious happens. If a **workdir** is specified and the UID is 0, RRDtool will do a chroot to that workdir. If the UID is not 0, RRDtool only changes the current directory to **workdir**.

RRD Server

If you want to create a RRD-Server, you must choose a TCP/IP Service number and add them to */etc/services* like this:

```
rrdsrv      13900/tcp                       # RRD server
```

Attention: the TCP port 13900 isn't officially registered for rrdsrv. You can use any unused port in your services file, but the server and the client system must use the same port, of course.

With this configuration you can add RRDtool as meta-server to */etc/inetd.conf*. For example:

```
rrdsrv stream tcp nowait root /opt/rrd/bin/rrdtool rrdtool – /var/rrd
```

Don't forget to create the database directory /var/rrd and reinitialize your inetd.

If all was setup correctly, you can access the server with Perl sockets, tools like netcat, or in a quick interactive test by using 'telnet localhost rrdsrv'.

NOTE: that there is no authentication with this feature! Do not setup such a port unless you are sure what you are doing.

RRDCACHED, THE CACHING DAEMON

For very big setups, updating thousands of RRD files often becomes a serious IO problem. If you run into such problems, you might want to take a look at rrdcached, a caching daemon for RRDtool which may help you lessen the stress on your disks.

SEE ALSO

rrdcreate, rrdupdate, rrdgraph, rrddump, rrdfetch, rrdtune, rrdlast, rrdxport, rrdflushcached, rrdcached

BUGS

Bugs? Features!

AUTHOR

Tobias Oetiker <tobi@oetiker.ch>