cmlgplot: GNU Plot Data Generator Version 1

CML00069-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

1 Introduction

This manual describes how to use the Code Magus tools `cmlgplot` and `cmlgplot_print`.

`cmlgplot` is a tool for preparing input to and shell scripts for `gnuplot`, for drawing graphs, by processing metric data from various sources. It performs either of the following functions:

- Convert legacy `ptester` statistic files to CML metric files; see section 6 on page 16.
- Under the control of an input rules file, generates plot data for `gnuplot` from Code Magus Metrics (CML metrics) input file produced using the Code Magus Metric library [2].

`cmlgplot` is described fully in the following sections.

`cmlgplot_print` is specifically used for supplying the actual input data to `gnuplot` at run time; It is described fully in section 5 on page 16.

2 Command Line Options

`cmlgplot` is invoked from the command line and by specifying the parameter '`--help`' will produce help information about the command line parameters. These parameters are explained in detail below:

```
user@machine:~>cmlgplot --help

Code Magus Limited cmlgplot V1.0: build 2011-04-18-13.36.00
[./cmlgplot] $Id: cmlgplot.c,v 1.9 2011/04/15 14:01:03 janvlok Exp $
Copyright (c) 2010 by Code Magus Limited. All rights reserved.
 [Contact: stephen@codemagus.com].
Usage: cmlgplot [OPTION...] <access>(<object>[,<options>]) ...
  -r, --plot-rules=<file name>                          Plot rules/specification
  -g, --gmt-offset={0|<seconds>}                        GMT offset of the origin of
  -s, --start-timestamp=<CCYY/MM/DD:hh:mm:ss>           Start timestamp for plotting
  -e, --end-timestamp=<CCYY/MM/DD:hh:mm:ss>             End timestamp for plotting
  -p, --output-plot-metric=<filename>                   Output file name for the plo
  -c, --convert2metric={TEXT|RDW}                       Convert old style ptester st
                                                        metric files
  -o, --output-metric=<access>(<object>[,<options>])    Output file open spec for co
  -a, --append-title=<string>                           Append this to the graph tit
  -v, --verbose                                         Verbose output

Help options:
  -?, --help                                            Show this help message
  --usage                                               Display brief usage message
```

Where:

- '-r|--plot-rules' Specifies the plot rules file for cmlgplot, see section 3 on page 4.
- '-g|--gmt-offset' Specifies the offset in seconds to GMT for the originator of the input metric files.
- '-s|--start-timestamp' Only process metrics that are older than this timestamp. The format of the timestamp is CCYY/MM/DD:hh:mm:ss.
- '-e|--end-timestamp' Only process metrics that are younger than this timestamp. The format of the timestamp is CCYY/MM/DD:hh:mm:ss.
- '-p|--output-plot-metric' Specifies the base filename for the output plot metrics, see section 4 on page 16.
- '-c|--convert2metric' Convert a legacy ptester statistic file to a CML metric file, see section 6 on page 16.
- '-o|--output-metric' Specifies the output CML metric file for the conversion. This must be given as a formatted Recio [1] open string specification naming the access method, the file (object) and any parameters to the access method.
- '-a|--append-title' Specifies a string to be appended to the title of the graphs.
- '-v|--verbose' When specified, detailed diagnostic information is produced as the program runs.

The input files to cmlgplot are specified on the command line following the options. More than one input may be specified. An input file specification is a formatted Recio [1] open string specification naming the access method, the file (object) and any parameters to the access method.

Metrics are generated by a number of provider systems and although the syntax is consistent (they are all generated using the Code Magus Metric Library [2]) the usage of the number of transacting devices (or instances) differs. Some providers (such as Orkhestra [3]) are aware of and provide the number of transacting devices on every metric. Others (such as the AIX metric probe) provide machine specific metrics (like CPU utilisation or data throughput of a network interface) and therefore have no concept of the number of devices in a system under test that may be executing during the same time frame.

Generally (but not always) all input metric files will cover the same time frame as each other. This is especially true where the purpose of the graphs are to represent the state of both the system under test (from Orkhestra [3]) and the underlying machine (from the machine specific probe) within the same time frame.

If graphs are plotted against the number of transacting devices (rather than for example time) then those input metric files that contain that information must be presented first (named first on the command line). Further metrics that are processed from subsequent metric files that do not have the number of transacting devices need this value to be set correctly so that the graph point can be plotted. The value is obtained from a metric that is chronologically the prior closest metric that does contain a valid number for the transacting devices. cmlgplot keeps track of the number of transacting devices as it processes each metric for the time period that the metrics were captured for.

3 Plot Rules Grammar

Plot rules control the execution of cmlgplot and specifies how one or more graphs are to be plotted.

3.1 Grammar Elements

The elements of the grammar comprise reserved words, identifiers, string literals and numbers. The grammar is free format and white spaces have no grammatical meaning except where they might appear within string literals.

3.1.1 Comments

Comments are introduced by using a double minus ("--") and continue up to the end of the current input line.

Examples:

```
-- File: t24_plot_stats.rules
--
-- Plotting rules for T24 stress run.
--
-- Copyright (c) 2010 Code Magus Limited. All rights reserved.
```

3.1.2 Reserved Words

Reserved words have a special meaning in terms of directing the parsing of commands. The reserved words are:

```
count         description  gplot_header    graph
group         instances    interval_count  metric
output        plot         rate            response
scale_factor  time         title           type
x1label       x1plot_type  y1label         y1plot_type
y1range       y1           y2label         y2plot_type
y2range       y2           yaxis
```

3.1.3 Identifiers

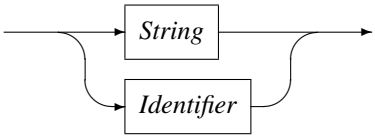
Identifiers are case sensitive and must start with a letter which can be followed by any number of letters, digits, decimal point '.' or the under-score character.

Examples:

```
cpu.global.idle  sample
```

3.1.4 File names

Filename



A *Filename* is usually written as a *String* but may also be an *Identifier*. Most importantly a *Filename* must conform to any constraints of the underlying file system.

3.1.5 Numbers

A number consists of a nonempty sequence of decimal digits that

- possibly contains a radix character (decimal point '.').
- is optionally followed by a decimal exponent; consisting of an 'E' or 'e' followed by an optional plus or minus sign followed by a nonempty sequence of decimal digits that indicates multiplication by a power of 10.

Examples:

```
1234
0.001
1.2
123.45E-12
```

3.1.6 Strings

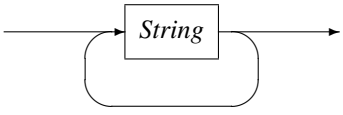
Strings are:

- any sequence of characters (except double quotes and the newline character) enclosed by double quotes.
- any sequence of characters (except single quotes and the newline character) enclosed by single quotes.

Examples:

```
"Clock ticks spent in system mode"
'$Revision: 1.13 $'
"User's log on time"
```

Strings can be concatenated:



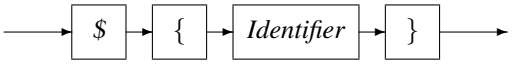
This facilitates the splitting of a long string over multiple lines to aid readability.

Example:

```
description("Plot from test data file ${T24_BASE_FNAME}\n"
            "mailto:stephen@codemagus.com")
```

3.1.7 Environment Variables

EnvironmentVariable

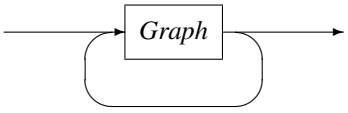


An *EnvironmentVariable* is a reference to a system environment variable, the value of which replaces the reference when encountered in a string.

3.2 Syntax

This section describes the syntax and semantics of the grammar used in preparing the plot data for a set of graphs.

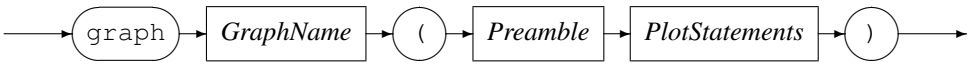
3.2.1 Defining a Set of Graphs



More than one graph can be defined in the input rules file.

3.2.2 *Graph*

Graph

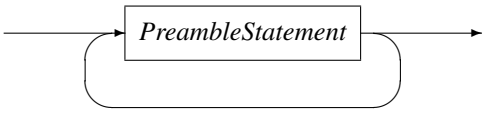


GraphName

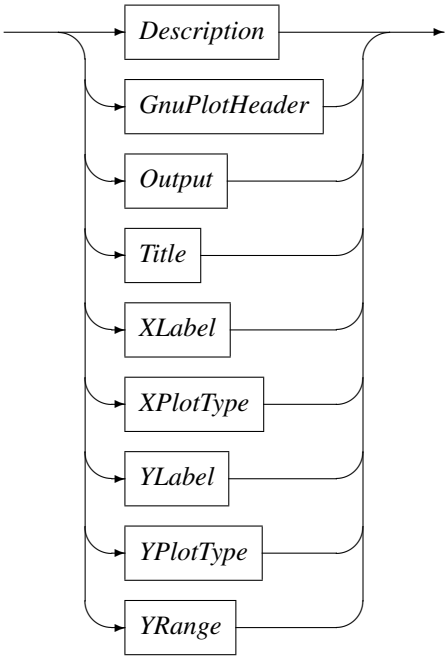
GraphName identifies the graph to be plotted. Defining a graph starts with the *Preamble* statements as described below, followed by the *PlotStatements* that describe the metrics to be plotted as described in section 3.2.13 on page 12.

3.2.3 *Preamble*

Preamble



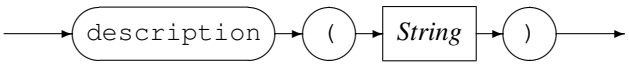
PreambleStatement



The preamble statements define the parameters required for the plot. They can be specified in any order.

3.2.4 *Description*

Description



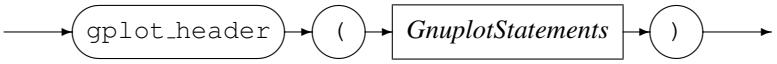
The description is mandatory and is used to describe the graph.

Example:

```
description("Plot from test data file ${T24OPS_BASE_FNAME}\n"
            "mailto:stephen@codemagus.com")
```

3.2.5 *GnuPlotHeader*

GnuPlotHeader



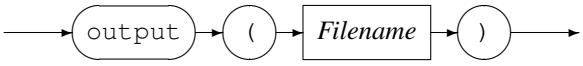
The plot header is optional. It allows for additional `gnuplot` statements to be specified and will be included at the head of the plot file. For the plot header any sequence of characters (except for the left '(' and right bracket ')') are allowed.

Example:

```
gplot_header
(
set key left Left reverse
set terminal postscript color
set y2tics
```

3.2.6 *Output*

Output



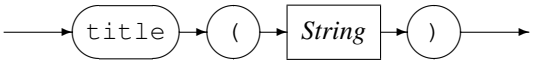
The *Output* option is mandatory. It is the name of the graphical file that is the final output of `gnuplot`. See section 3.1.4 on page 6 for the syntax of `Filename`.

Example:

```
output("${T24OPS_BASE_FNAME}_net_instances.eps")
```

3.2.7 *Title*

Title



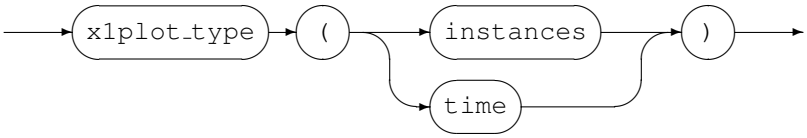
Title is mandatory. It is used for the title of the final graph generated by `gnuplot`.

Example:

```
title("T24ops Network Utilisation")
```

3.2.8 *XPlotType*

XPlotType



XPlotType specifies what the plot values are plotted against:

- `instances`
  The plot is against the number of instances specified in the CML metric.

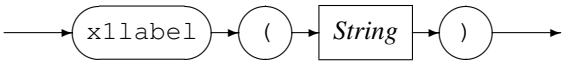
- `time`
  The plot is a time line.

Example:

```
x1plot_type(instances)
x1plot_type(time)
```

3.2.9 *XLabel*

XLabel



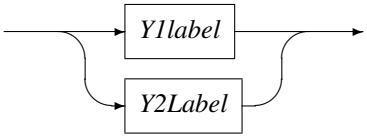
XLabel is mandatory and specifies the label to be used for the graph's X axis.

Example:

```
x1label("Number of Transacting devices")
```

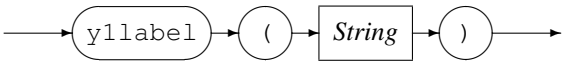
3.2.10 *YLabel*

YLabel



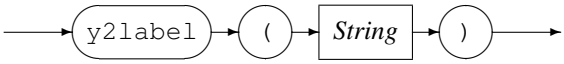
YLabel is used to specify the left and/or the right Y-axis as follows.

Y1Label



Y1Label specifies the label to be used for the graph's Y1 axis (left). It is required if the plot type is set for the Y1 axis; see section 3.2.11 on page 11.

Y2Label



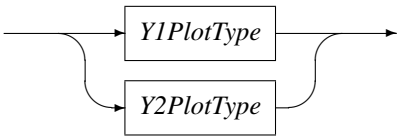
Y2Label specifies the label to be used for the graph's Y2 axis (right). It is required if the plot type is set for the Y2 axis; see section 3.2.11 on page 11.

Example:

```
y1label("Packet Rate")
y2label("Byte Rate")
```

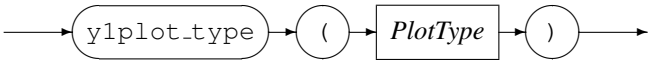
3.2.11 *YPlotType*

YPlotType



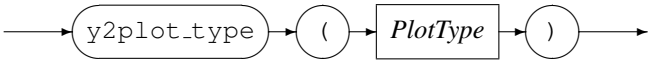
YPlotType is used to specify the right and/or the left Y-axis plot type as follows.

Y1PlotType



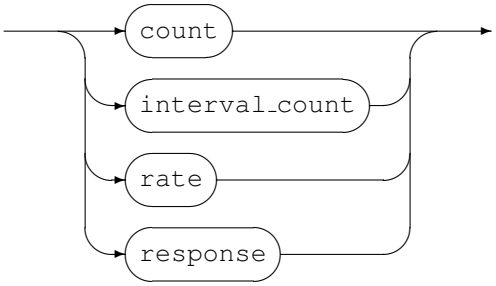
This sets the plot type for Y1 axis (left).

Y1PlotType



This set the plot type for Y2 axis (right).

PlotType



Specifies the calculation required for plotting the metrics:

- `count`
  Plot value is count. This is the first value in the CML metric of type `METRIC_IS_VALUES`.
- `interval_count`
  As above, but adjusted to reflect the count in the last interval. The interval is the time lapsed between metrics of the same name.
- `rate`
  Plot value is rate. This is calculated by dividing the interval count by the interval.
- `response`
  Plot value is response. This is calculated by dividing the sum (of response time) for the interval by the interval count. The count is the first value in the CML metric and the sum (of response time) is the second value.

  Note the response time is in seconds.

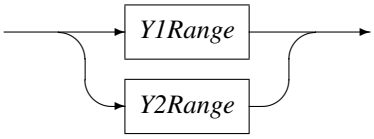
Refer to metric: Metric Library Reference [2] for the documentation of CML metrics.

Example:

```
y1plot_type(response)
y2plot_type(rate)
```

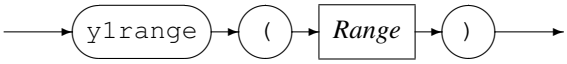
3.2.12 *YRange*

YRange



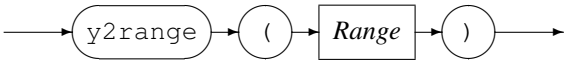
YRange is used to specify the right and/or the left Y-axis range as follows.

Y1Range



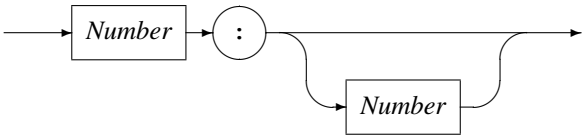
Y1Range is optional and specifies the Y1 axis (left) range.

Y2Range



Y1Range is optional and specifies the Y2 axis (right) range.

Range



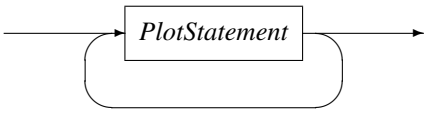
Example:

```
y1range(0:)
y1range(0:2)
```

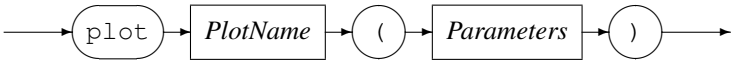
3.2.13 *PlotStatements*

.

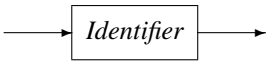
PlotStatements



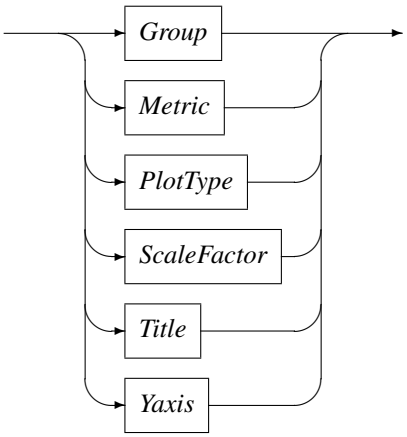
PlotStatement



PlotName



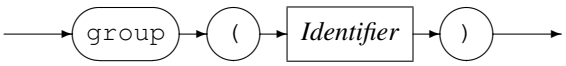
Parameters



PlotName identifies the plot on the graph and will be used as the title for the plot on the graph. This can be over written by using the parameter `title`; see section 3.2.18 on page 15. The parameters may be specified in any order.

3.2.14 *Group*

Group



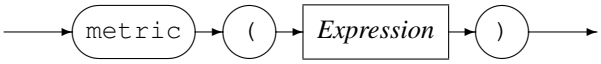
Group identifies the group to which the CML metrics belong to for this plot.

Example:

```
group(t24)
```

3.2.15 *Metric*

Metric



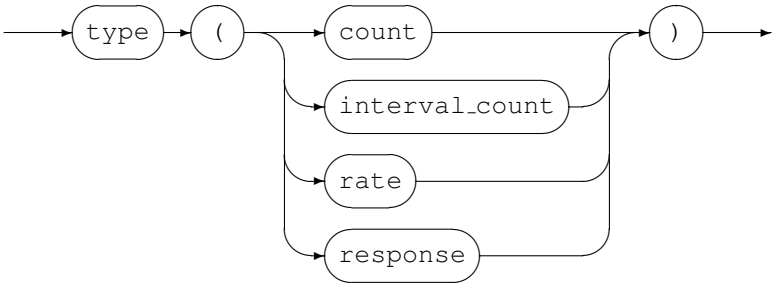
Expression is a regular arithmetic expression where the variables are CML metrics that makes up this plot.

Examples:

```
metric
   (
   choice_what_transaction.choice.deposit
   +choice_what_transaction.choice.withdrawal
   )
metric(wait_deposit_response.DEPOSIT_OK*1000)
```

3.2.16 *PlotType*

PlotType



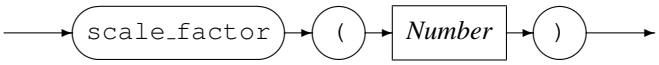
Refer to section 3.2.11 on page 11 for a description of the plot types.

Example:

```
type(rate)
```

3.2.17 *ScaleFactor*

ScaleFactor



ScaleFactor specifies the scaling factor to apply to the final plot value.

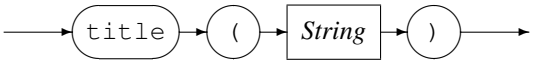
Example:

Response time is in seconds, to plot it in milliseconds, a scaling factor of 1000 needs to be applied.

```
   scale_factor (1E3)
or
   scale_factor (1000)
```

3.2.18 *Title*

Title



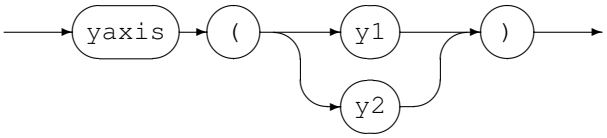
Title specifies the title for the plot. If omitted the plot name is used.

Example:

```
title("DEPOSIT_request (rate)")
```

3.2.19 *Yaxis*

Yaxis



Yaxis specifies which Y axis to use for the plot. It is only required if plotting to both the Y axes and they are both of the same plot type.

Example:

```
type(rate) yaxis(y1)
type(rate) yaxis(y2)
```

4 cmlgplot Output Files

cmlgplot creates the following output files:

- The '-p|--output-plot-metric' command line parameter is used as the base filename for the following two output files:
  - A suffix of '.instances' is appended to the base file name for the file holding the plot values against number of instances.
  - A suffix of '.time' appended to the base file name for the file holding the plot values against time.
- For each graph (section 3.2.2 on page 7) two files for the actual plotting by gnuplot are created. The name of these files are the graph name with the following extensions:
  - '.plot' Plot statements for gnuplot.
  - '.sh' Shell script to run gnuplot and to create a PDF file from the output plot.

5 cmlgplot_print

cmlgplot_print is called by gnuplot to supply the plot values of a metric.

5.1 Synopsis

```
Usage: cmlgplot_print [OPTION...] <metric file name>
  -m, --plot-metric={<name>}     Print the plot values for this metric
  -v, --verbose                  Verbose output

Help options:
  -?, --help                     Show this help message
  --usage                        Display brief usage message
```

Where:

- '-m|--plot-metric' Print the plot values for this metric.
- '-v|--verbose' When specified, cmlgplot_print operates in a verbose manner.

6 Convert legacy ptester Statistic Files

This section describes how to convert legacy ptester statistic files to CML metric files. The two command line options pertinant to this are:

- '-c|--convert2metric' identifies the type of the statistic file:
  - 'TEXT' Textual format.
  - 'RDW' Binary format.

- '-o|--output-metric' specifies the output CML metric file for the conversion as a formatted Recio [1] open string specification naming the access method, the file (object) and any parameters to the access method.

  The input statistic file to be converted is specified on the command line as a formatted Recio [1] open string specification naming the access method, the file (object) and any parameters to the access method.

Example:

Convert the legacy ptester statistic file 't24_stats_D20100802.rdw' to the CML metric file 't24_stats_D20100802.metric'.

```
cmlgplot --convert2metric=RDW \
   --output-metric="text(t24_stats_D20100802.metric,mode=w)" \
   "binary(t24_stats_D20100802.rdw,recfm=v,mode=rb)"
```

A Example1 Plot

This example shows graphs displaying various metrics plotted against the number of transacting devices (or instances).

The metrics for this example were collected from:

- An `Orkhestra` [3] stress test. This is the primary input and were recorded in a CML metric file: '`t24_stats_D20100802.metric`'.
- A CML probe running on the host machine under stress. The metrics were recorded in a CML metric file: '`t24ops_D20100802.metric`'.

  These metrics describe the underlying machine state over the same time frame as those in the primary input file. Metrics at this level have no concept of the transacting devices in `Orkhestra` [3] and therefore this value is set to zero when the metric is produced. As the metric is processed by `cmlgplot` the value used for the number of transacting devices is taken from a metric in the primary input file that is chronologically the prior closest.

The rules for creating the graphs are shown in appendix A.2 on page 20.

The following graphs were created (see appendix A.3 on page 25):

- '`load_instances`' Plot the rate and response of the transactions against the number of transacting devices.

  The output graph is '`t24_stats_D20100802_load_instances.eps`' and in PDF format '`t24_stats_D20100802_load_instances.pdf`'
- '`cpu_instances`' Plot the host CPU utilisation against the number of transacting devices.

  The output graph is '`t24ops_D20100802_cpu_instances.eps`' and in PDF format '`t24ops_D20100802_cpu_instances.pdf`'
- '`net_instances`' Plot the host network utilisation against the number of transacting devices.

  The output graph is '`t24ops_D20100802_net_instances.eps`' and in PDF format '`t24ops_D20100802_net_instances.pdf`'

The shell script (see appendix A.1 on page 19) for creating the example graphs has two phases to it:

- Run `cmlgplot` to create the required input files for `gnuplot` and the shell scripts for creating the graphs.
- Run the three shell scripts, created by the previous phase, for `gnuplot` to create the graphs.

In phase one the following files were created:

- Plot values against number of instances:
  `t24_stats_D20100802.plot.metric.instances`
- Plot values against time:
  `t24_stats_D20100802.plot.metric.time`
- `gnuplot` plot statements for the graphs:
  `cpu_instances.plot`
  `load_instances.plot`
  `net_instances.plot`

- Shell scripts for running gnuplot to create the graphs:
  cpu_instances.sh
  load_instances.sh
  net_instances.sh

A.1 example1.sh

Shell script to run cmlgplot:

```
#!/bin/bash
#
# File: example1.sh
#
# Copyright (c) 2010 Code Magus Limited. All rights reserved.
#
# $Author: janvlok $
# $Date: 2011/06/15 13:16:06 $
# $Id: example1.sh,v 1.2 2011/06/15 13:16:06 janvlok Exp $
# $Source: /home/cvs/cvsroot/cmlgplot/example1.sh,v $
# $Revision: 1.2 $
# $State: Exp $
#
# $Log: example1.sh,v $
# Revision 1.2  2011/06/15 13:16:06  janvlok
# Remove legacy ptester example
#
# Revision 1.1  2010/10/04 06:54:58  janvlok
# Take on
#

# Set the date of our input files:
DATE=20100802

# Plotting start and end time steam's.
FROM="--start-timestamp=2010/08/02:14:26:00"
TO="--end-timestamp=2010/08/02:14:54:00"

# The GMT offset for the originator of the input metrics:
GMT="--gmt-offset=7200"

# Plotting rules:
RULE=${HOME}/software/cmlgplot/example1.rules

# Path to the input metrics and plot output:
STATSPATH=${HOME}/software/cmlgplot/testing

# orkhestra stress stats/metrics, excluding any extensions:
export T24_BASE_FNAME=t24_stats_D${DATE}

# Metrics from host for stress run, excluding any extensions:
export T24OPS_BASE_FNAME=t24ops_D${DATE}
```

```
# cd to our stats:
cd ${STATSPATH}

# Create the plot files:
echo Creating plot files
cmlgplot --plot-rules=${RULE} ${GMT} ${FROM} ${TO} \
   --output-plot-metric=${T24_BASE_FNAME}.plot.metric \
   "text(${T24_BASE_FNAME}.metric,mode=r)" \
   "text(${T24OPS_BASE_FNAME}.metric,mode=r)"

rc=$?
if [ ${rc} -ne 0 ]
then
   exit
fi

# Using the shell scripts, outputted by cmlgplot, do the plots
echo Plot graphs:
./load_instances.sh
./cpu_instances.sh
./net_instances.sh
```

A.2 example1.rules

Rules for the `cmlgplot` example1:

```
-- File: example1.rules
--
-- Plotting rules for a example1 stress run.
--
-- Copyright (c) 2010 Code Magus Limited. All rights reserved.
--
-- $Author: janvlok $
-- $Date: 2011/06/14 08:31:27 $
-- $Id: example1.rules,v 1.5 2011/06/14 08:31:27 janvlok Exp $
-- $Name:  $
-- $Revision: 1.5 $
-- $State: Exp $
--
-- $Log: example1.rules,v $
-- Revision 1.5  2011/06/14 08:31:27  janvlok
-- Spelling and scale factor
--
-- Revision 1.4  2011/04/26 12:33:35  janvlok
-- Metric library changes
--
-- Revision 1.3  2010/10/05 07:09:58  janvlok
-- Made gnuplot header obsolete
--
```

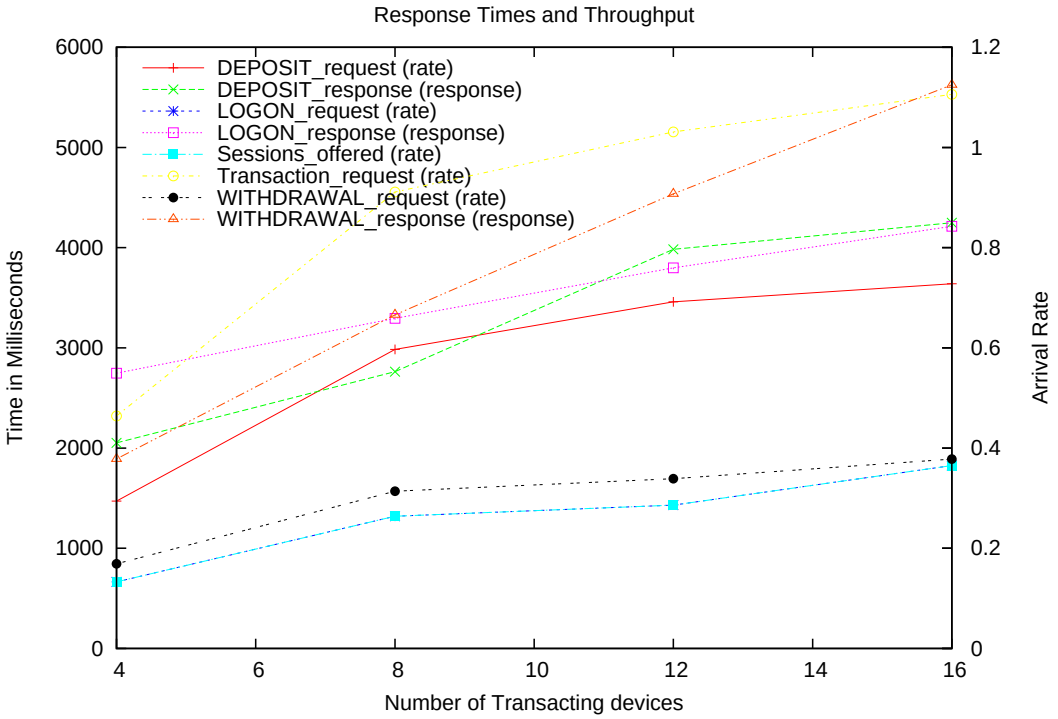
```
-- Revision 1.2  2010/10/04 09:21:02  janvlok
-- Change the load description
--
-- Revision 1.1  2010/10/04 06:54:58  janvlok
-- Take on
--
graph load_instances
(
 x1plot_type(instances)
 y1plot_type(response)
 y2plot_type(rate)
 y1range(0:)
 y2range(0:)
 output("${T24_BASE_FNAME}_load_instances.eps")
 title("Response Times and Throughput")
 description("Plot from test data file ${T24_BASE_FNAME}\n"
             "mailto:stephen@codemagus.com")
 x1label("Number of Transacting devices")
 y1label("Time in Milliseconds")
 y2label("Arrival Rate")

plot DEPOSIT_request
   (
   title("DEPOSIT_request (rate)")
   group(t24)
   type(rate)
   metric(choice_what_transaction.choice.deposit)
   )
plot DEPOSIT_response
   (
   title("DEPOSIT_response (response)")
   group(t24)
   type(response)
   scale_factor(1E3)
   metric(wait_deposit_response.DEPOSIT_OK)
   )
plot LOGON_request
   (
   title("LOGON_request (rate)")
   group(t24)
   type(rate)
   metric(wait_connection.connect)
   )
plot LOGON_response
   (
   title("LOGON_response (response)")
   group(t24)
   type(response)
   scale_factor(1E3)
   metric(wait_logon_user.LOGON_OK)
   )
plot Sessions_offered
```

```
    (
    title("Sessions_offered (rate)")
    group(t24)
    type(rate)
    metric(device_idle.timer_expire.device_ready)
    )
plot Transaction_request
    (
    title("Transaction_request (rate)")
    group(t24)
    type(rate)
    metric
        (
        choice_what_transaction.choice.deposit
        +choice_what_transaction.choice.withdrawal
        )
    )
plot WITHDRAWAL_request
    (
    title("WITHDRAWAL_request (rate)")
    group(t24)
    type(rate)
    metric(choice_what_transaction.choice.withdrawal)
    )
plot WITHDRAWAL_response
    (
    title("WITHDRAWAL_response (response)")
    group(t24)
    type(response)
    scale_factor(1E3)
    metric(wait_withdrawal_response.WITHDRAWAL_OK)
    )
)

graph cpu_instances
(
 x1plot_type(instances)
 y1plot_type(rate)
 y1range(0:2)
 output("${T24OPS_BASE_FNAME}_cpu_instances.eps")
 title("T24ops CPU Utilisation")
 description("Plot from test data file ${T24OPS_BASE_FNAME}\n"
             "mailto:stephen@codemagus.com")
 x1label("Number of Transacting devices")
 y1label("CPU Load")

plot Cpu_idle
    (
    group(t24)
    type(rate)
    metric(cpu_global_idle)
    )
```

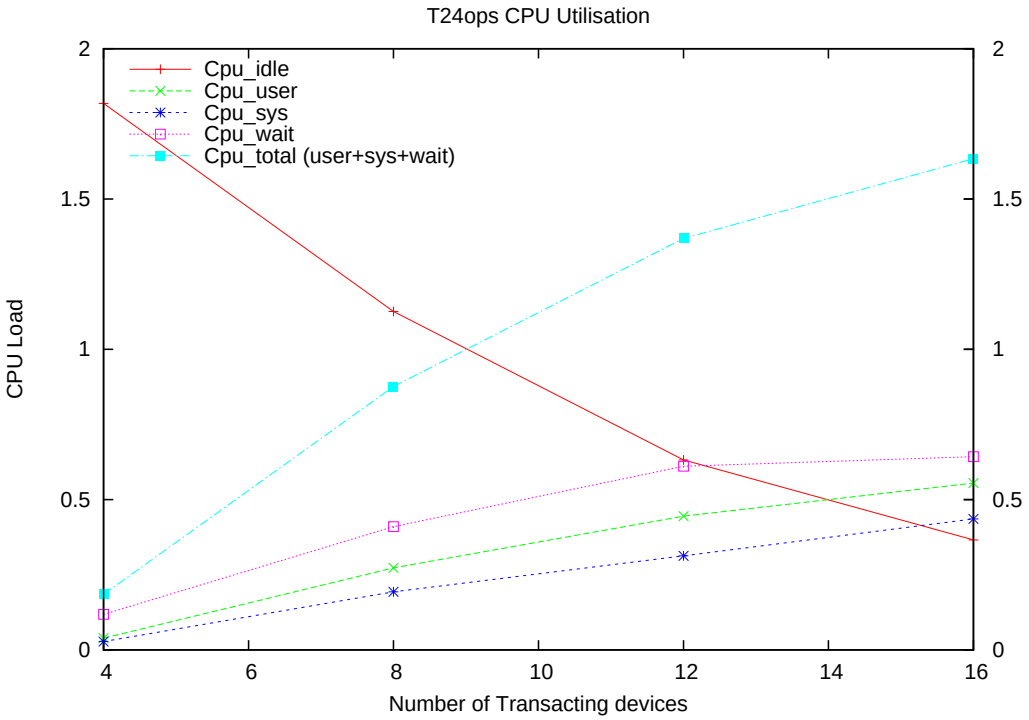
```
plot Cpu_user
   (
   group(t24)
   type(rate)
   metric(cpu_global_user)
   )

plot Cpu_sys
   (
   group(t24)
   type(rate)
   metric(cpu_global_sys)
   )

plot Cpu_wait
   (
   group(t24)
   type(rate)
   metric(cpu_global_wait)
   )

plot Cpu_total
   (
   group(t24)
   type(rate)
   metric(cpu_global_user+cpu_global_sys+cpu_global_wait)
   title("Cpu_total (user+sys+wait)")
   )
)

graph net_instances
(
 x1plot_type(instances)
 y1plot_type(rate)
 y2plot_type(rate)
 y1range(0:)
 y2range(0:)
 output("${T24OPS_BASE_FNAME}_net_instances.eps")
 title("T24ops Network Utilisation")
 description("Plot from test data file ${T24OPS_BASE_FNAME}\n"
             "mailto:stephen@codemagus.com")
 x1label("Number of Transacting devices")
 y1label("Packet Rate")
 y2label("Byte Rate")

plot netinterface_en4_ipackets
   (
   title("netinterface_en4_ipackets")
   group(t24)
   type(rate) yaxis(y1)
   metric(netinterface_en4_ipackets)
```

```
    )
plot netinterface_en4_opackets
    (
    title("netinterface_en4_opackets")
    group(t24)
    type(rate) yaxis(y1)
    metric(netinterface_en4_opackets)
    )

plot netinterface_en4_ibytes
    (
    title("netinterface_en4_ibytes")
    group(t24)
    type(rate) yaxis(y2)
    metric(netinterface_en4_ibytes)
    )
plot netinterface_en4_obytes
    (
    title("netinterface_en4_obytes")
    group(t24)
    type(rate) yaxis(y2)
    metric(netinterface_en4_obytes)
    )

)
```

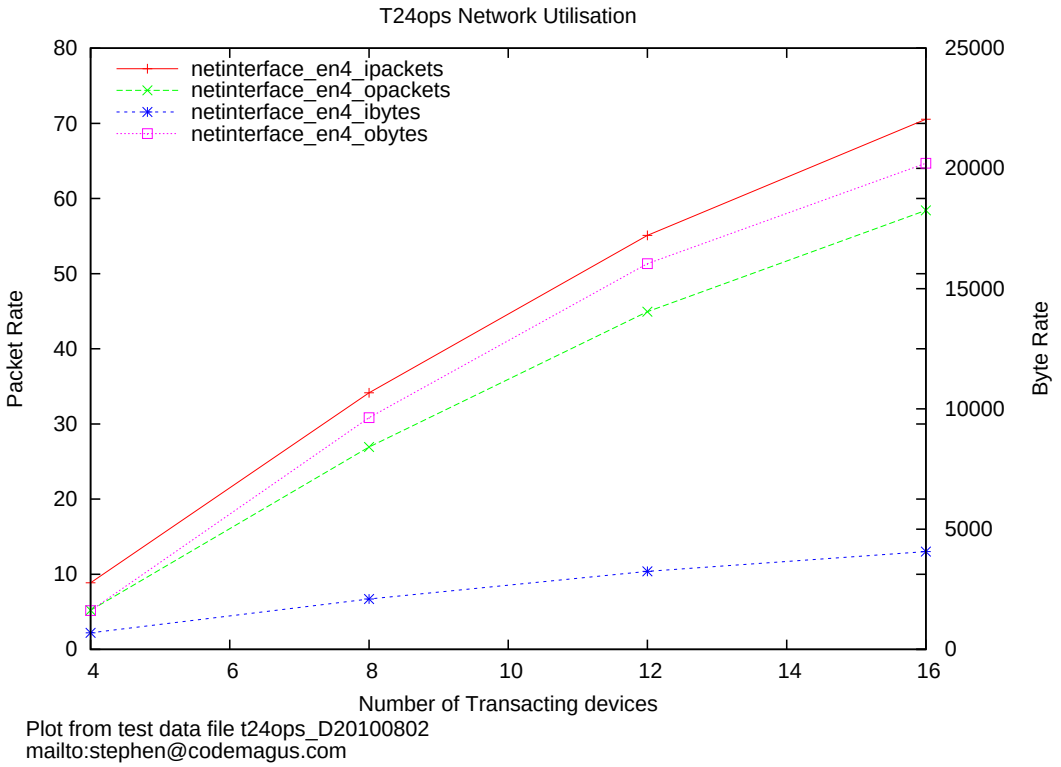
A.3 Graphs



Plot from test data file t24_stats_D20100802
mailto:stephen@codemagus.com



Plot from test data file t24ops_D20100802
mailto:stephen@codemagus.com



B Example2 Plot

This example shows graphs displaying various metrics plotted against the time. These graphs could also have been produced within the first example, but have been done here to highlight graphs and example data that do not require the number of transacting devices (or instances). The order in which the input CML metric files are specified is therefore not important.

The metrics for this sample were collected from:

- An `Orkhestra` [3] stress test. The metrics were recorded in a CML metric file: '`t24_stats_D20100802.metric`'.
- A CML probe running on the host machine under stress. The metrics were recorded in a CML metric file: '`t24_stats_D20100802.metric`'

The rules for creating the graphs are shown in appendix B.2 on page 28.

The following graphs were created (see appendix B.3 on page 33):

- '`load_time`' Plot the rate and response of the transactions against time.
  The output graph is '`t24_stats_D20100802_load_time.eps`' and in PDF format '`t24_stats_D20100802_load_time.pdf`'
- '`cpu_time`' Plot the host CPU utilisation against time.
  The output graph is '`t24ops_D20100802_cpu_time.eps`' and in PDF format '`t24ops_D20100802_cpu_time.pdf`'

- 'net_time' Plot the host network utilisation against time.
  The output graph is 't24ops_D20100802_net_time.eps' and in PDF format 't24ops_D20100802_net_time.pdf'

The shell script (see appendix B.1 on page 27) for creating the example graphs has two phases to it:

- Run cmlgplot to create the required input files for gnuplot and the shell scripts for creating the graphs.
- Run the three shell scripts, created by the previous phase, for gnuplot to create the graphs.

In phase one the following files were created:

- Plot values against number of instances:
  t24_stats_D20100802.plot.metric.time
- Plot values against time:
  t24_stats_D20100802.plot.metric.time
- gnuplot plot statements for the graphs:
  cpu_time.plot
  load_time.plot
  net_time.plot
- Shell scripts for running gnuplot to create the graphs:
  cpu_time.sh
  load_time.sh
  net_time.sh

B.1 example2.sh

Shell script to run cmlgplot:

```
#!/bin/bash
#
# File: example2.sh
#
# Copyright (c) 2010 Code Magus Limited. All rights reserved.
#
# $Author: janvlok $
# $Date: 2011/06/15 13:16:06 $
# $Id: example2.sh,v 1.2 2011/06/15 13:16:06 janvlok Exp $
# $Source: /home/cvs/cvsroot/cmlgplot/example2.sh,v $
# $Revision: 1.2 $
# $State: Exp $
#
# $Log: example2.sh,v $
# Revision 1.2  2011/06/15 13:16:06  janvlok
# Remove legacy ptester example
#
# Revision 1.1  2010/10/04 06:54:58  janvlok
# Take on
#
```

```
# Set the date of our input files:
DATE=20100802

# Plotting start and end time steam's.
FROM="--start-timestamp=2010/08/02:14:26:00"
TO="--end-timestamp=2010/08/02:14:54:00"

# The GMT offset for the originator of the input metrics:
GMT="--gmt-offset=7200"

# Plotting rules:
RULE=${HOME}/software/cmlgplot/example2.rules

# Path to the input metrics and plot output:
STATSPATH=${HOME}/software/cmlgplot/testing

# orkhestra stress stats/metrics, excluding any extensions:
export T24_BASE_FNAME=t24_stats_D${DATE}

# Metrics from host for stress run, excluding any extensions:
export T24OPS_BASE_FNAME=t24ops_D${DATE}

# cd to our stats:
cd ${STATSPATH}

# Create the plot files:
echo Creating plot files
cmlgplot --plot-rules=${RULE} ${GMT} ${FROM} ${TO} \
   --output-plot-metric=${T24_BASE_FNAME}.plot.metric \
   "text(${T24_BASE_FNAME}.metric,mode=r)" \
   "text(${T24OPS_BASE_FNAME}.metric,mode=r)"

rc=$?
if [ ${rc} -ne 0 ]
then
   exit
fi

# Using the shell scripts, outputted by cmlgplot, do the plots
echo Plot graphs:
./load_time.sh
./cpu_time.sh
./net_time.sh
```

B.2 example2.rules

Rules for the `cmlgplot` example:

```
-- File: example2.rules
```

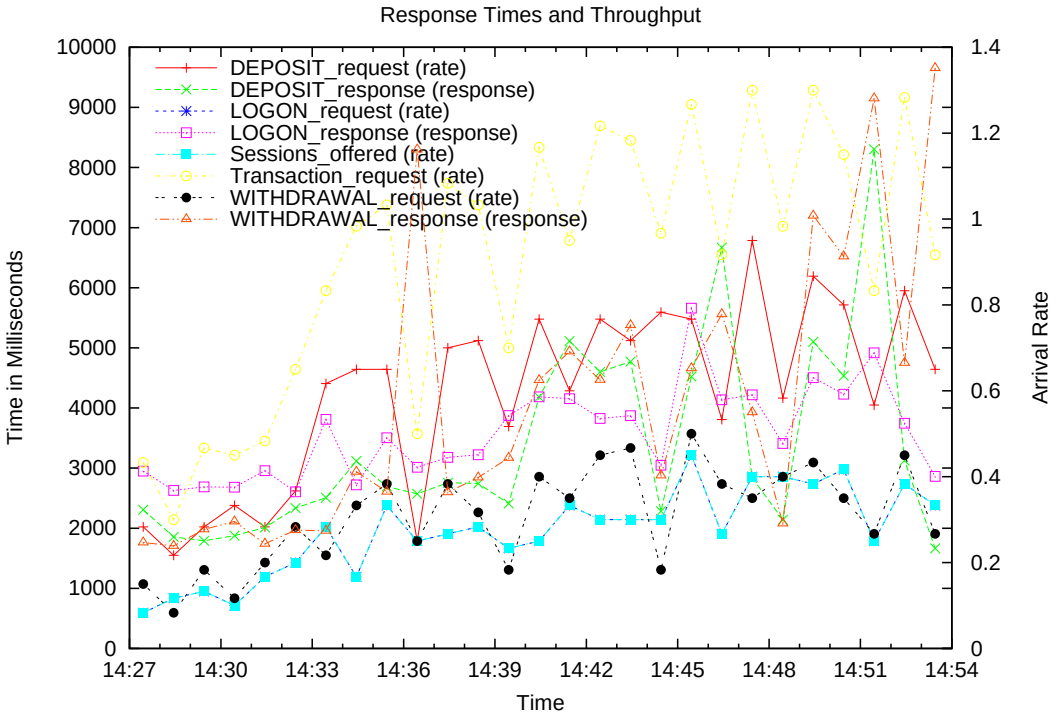
```
--
-- Plotting rules for a example2 stress run.
--
-- Copyright (c) 2010 Code Magus Limited. All rights reserved.
--
-- $Author: janvlok $
-- $Date: 2011/06/14 08:31:27 $
-- $Id: example2.rules,v 1.5 2011/06/14 08:31:27 janvlok Exp $
-- $Name:  $
-- $Revision: 1.5 $
-- $State: Exp $
--
-- $Log: example2.rules,v $
-- Revision 1.5  2011/06/14 08:31:27  janvlok
-- Spelling and scale factor
--
-- Revision 1.4  2011/04/26 12:33:35  janvlok
-- Metric library changes
--
-- Revision 1.3  2010/10/05 07:09:58  janvlok
-- Made gnuplot header obsolete
--
-- Revision 1.2  2010/10/04 09:21:02  janvlok
-- Change the load description
--
-- Revision 1.1  2010/10/04 06:54:58  janvlok
-- Take on
--
graph load_time
(
 x1plot_type(time)
 y1plot_type(response)
 y2plot_type(rate)
 y1range(0:)
 y2range(0:)
 output("${T24_BASE_FNAME}_load_time.eps")
 title("Response Times and Throughput")
 description("Plot from test data file ${T24_BASE_FNAME}\n"
             "mailto:stephen@codemagus.com")
 x1label("Time")
 y1label("Time in Milliseconds")
 y2label("Arrival Rate")

plot DEPOSIT_request
   (
   title("DEPOSIT_request (rate)")
   group(t24)
   type(rate)
   metric(choice_what_transaction.choice.deposit)
   )
plot DEPOSIT_response
   (
```

```
   title("DEPOSIT_response (response)")
   group(t24)
   type(response)
   scale_factor(1E3)
   metric(wait_deposit_response.DEPOSIT_OK)
   )
plot LOGON_request
   (
   title("LOGON_request (rate)")
   group(t24)
   type(rate)
   metric(wait_connection.connect)
   )
plot LOGON_response
   (
   title("LOGON_response (response)")
   group(t24)
   type(response)
   scale_factor(1E3)
   metric(wait_logon_user.LOGON_OK)
   )
plot Sessions_offered
   (
   title("Sessions_offered (rate)")
   group(t24)
   type(rate)
   metric(device_idle.timer_expire.device_ready)
   )
plot Transaction_request
   (
   title("Transaction_request (rate)")
   group(t24)
   type(rate)
   metric
      (
      choice_what_transaction.choice.deposit
      +choice_what_transaction.choice.withdrawal
      )
   )
plot WITHDRAWAL_request
   (
   title("WITHDRAWAL_request (rate)")
   group(t24)
   type(rate)
   metric(choice_what_transaction.choice.withdrawal)
   )
plot WITHDRAWAL_response
   (
   title("WITHDRAWAL_response (response)")
   group(t24)
   type(response)
   scale_factor(1E3)
```

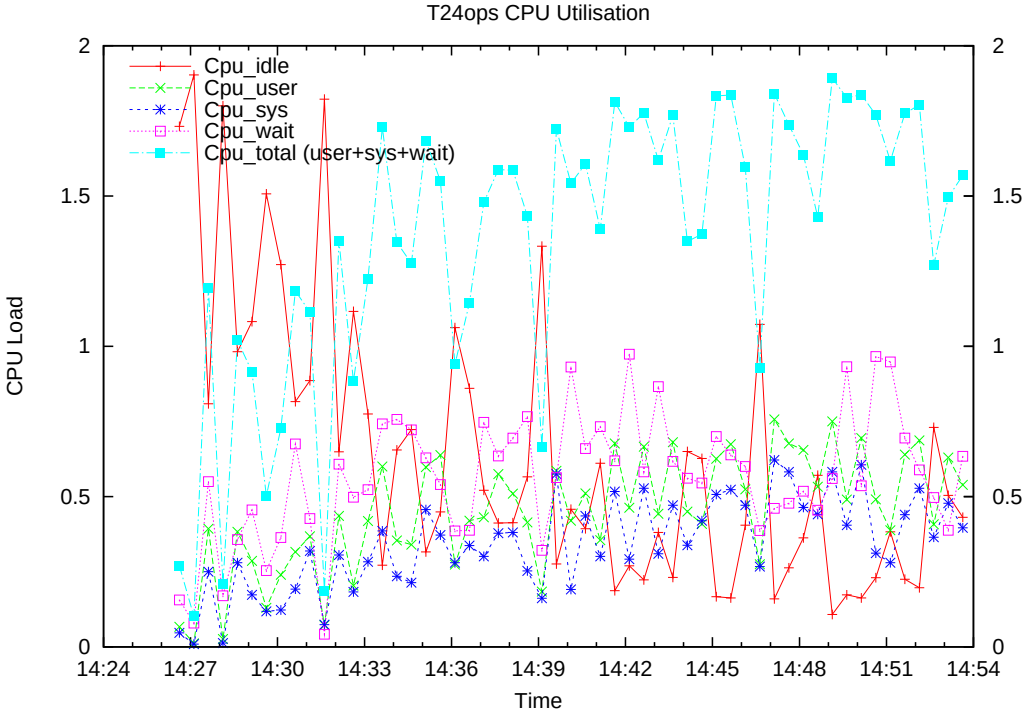
```
   metric(wait_withdrawal_response.WITHDRAWAL_OK)
   )
)

graph cpu_time
(
 x1plot_type(time)
 y1plot_type(rate)
 y1range(0:2)
 output("${T24OPS_BASE_FNAME}_cpu_time.eps")
 title("T24ops CPU Utilisation")
 description("Plot from test data file ${T24OPS_BASE_FNAME}\n"
             "mailto:stephen@codemagus.com")
 x1label("Time")
 y1label("CPU Load")

plot Cpu_idle
   (
   group(t24)
   type(rate)
   metric(cpu_global_idle)
   )

plot Cpu_user
   (
   group(t24)
   type(rate)
   metric(cpu_global_user)
   )

plot Cpu_sys
   (
   group(t24)
   type(rate)
   metric(cpu_global_sys)
   )

plot Cpu_wait
   (
   group(t24)
   type(rate)
   metric(cpu_global_wait)
   )

plot Cpu_total
   (
   group(t24)
   type(rate)
   metric(cpu_global_user+cpu_global_sys+cpu_global_wait)
   title("Cpu_total (user+sys+wait)")
   )
)
```

```
graph net_time
(
 x1plot_type(time)
 y1plot_type(rate)
 y2plot_type(rate)
 y1range(0:)
 y2range(0:)
 output("${T24OPS_BASE_FNAME}_net_time.eps")
 title("T24ops Network Utilisation")
 description("Plot from test data file ${T24OPS_BASE_FNAME}\n"
             "mailto:stephen@codemagus.com")
 x1label("Time")
 y1label("Packet Rate")
 y2label("Byte Rate")

plot netinterface_en4_ipackets
    (
    title("netinterface_en4_ipackets")
    group(t24)
    type(rate) yaxis(y1)
    metric(netinterface_en4_ipackets)
    )
plot netinterface_en4_opackets
    (
    title("netinterface_en4_opackets")
    group(t24)
    type(rate) yaxis(y1)
    metric(netinterface_en4_opackets)
    )

plot netinterface_en4_ibytes
    (
    title("netinterface_en4_ibytes")
    group(t24)
    type(rate) yaxis(y2)
    metric(netinterface_en4_ibytes)
    )
plot netinterface_en4_obytes
    (
    title("netinterface_en4_obytes")
    group(t24)
    type(rate) yaxis(y2)
    metric(netinterface_en4_obytes)
    )

)
```

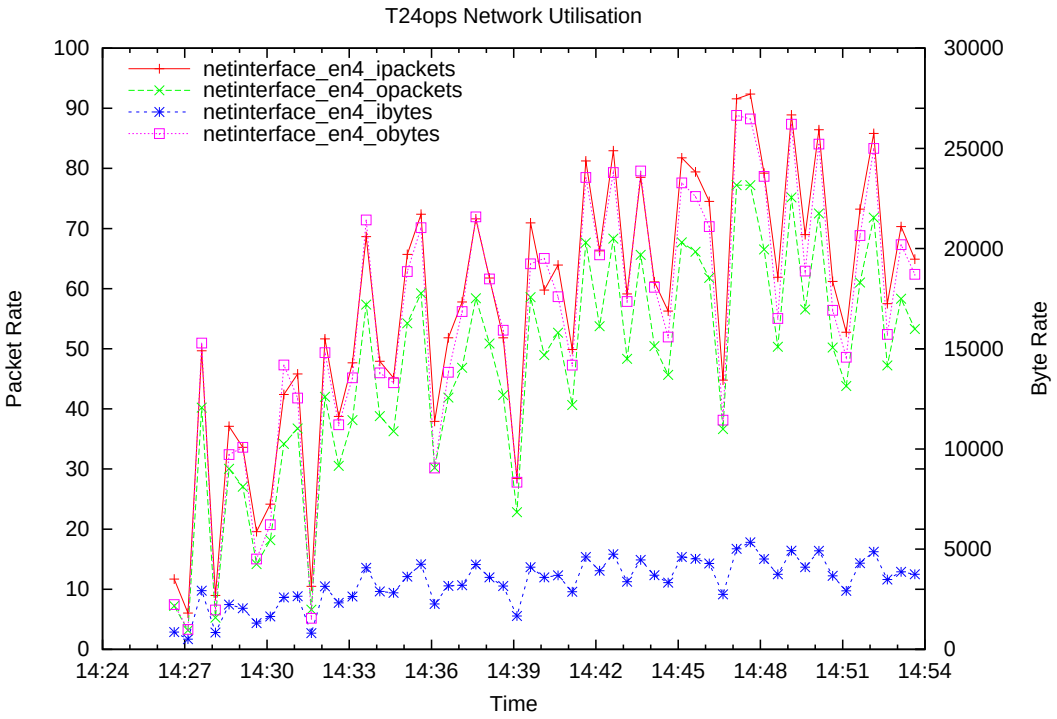
B.3 Graphs



Plot from test data file t24_stats_D20100802
mailto:stephen@codemagus.com



Plot from test data file t24ops_D20100802
mailto:stephen@codemagus.com



Plot from test data file t24ops_D20100802
mailto:stephen@codemagus.com

References

[1] recio: Record Stream I/O Library Version 1. CML Document CML00001-01, Code Magus Limited, July 2008. PDF.

[2] metric: Metric Library Reference. CML Document CML00006-01, Code Magus Limited, September 2010. PDF.

[3] orkhestra: Configuration and User Reference Version 1. CML Document CML00041-01, Code Magus Limited, June 2011. PDF.