



CrossGrid User Manual Guide

G-PM

Task 2.4.1 - G-PM

Document Filename:	CG-UserManual
Workpackage:	Task 2.4.1 - G-PM
Partner(s):	CYF
Lead Partner:	CYF
Config ID:	cg-usermanual-v0.1
Document classification:	PUBLIC

Abstract:



Delivery Slip

	Name	Partner	Date	Signature
From	Tomasz Arod, Marcin Kurdziel, Wodzimierz Funika	CYFRONET	Nov 2004	
Verified By				
Approved By				

Document Log

Version	Date	Summary of changes	Author
0.1	Nov 3rd, 2004	First draft version	Piotr Nowakowski

Contents

CopyrightNotice	4
1 Introduction	5
1.1 Product Usage	5
1.2 Running the Product	6
1.3 Basic Operation	7
1.4 Advanced Features	10
2 User Interface Reference Guide	14
2.1 Main window of the G-PM	14
2.2 Main Menu	15
2.3 The "Measurement Definition" window	16
2.4 The "Visualization Definition" window	17
2.5 The "Edit custom metric" window	18
2.6 The "Bargraph" window	19
2.7 The "MultiCurve" window	20
2.8 The "Histogram" window	20
2.9 The "PieChart" window	21
2.10 The "Communication Matrix" window	21
3 PMSL Reference Guide	23
3.1 Introduction	23
3.2 PMSL Overview	23
3.3 PMSL Usage Examples	24
3.4 PMSL Syntax	28
3.5 Semantic Aspects of PMSL	30
4 Troubleshooting Q and A	36
5 GNU General Public License	37

Copyright Notice

Copyright (c) 2005 by **ACC CYFRONET AGH, Krakow, Poland; AGH University of Science and Technology, Krakow, Poland; Technische Universität München, Germany; Universität Siegen, Germany**. All rights reserved.

Use of this product is subject to the terms and licenses stated in the GPL license agreement. (***or other copyright agreement - please specify***). Please refer to Chapter 5 for details.

This research is partly funded by the European Commission IST-2001-32243 Project CrossGrid.

1 Introduction

The G-PM is the CrossGrid [2] performance monitoring tool for parallel grid applications. Its purpose is to monitor run-time behavior of applications and detect possible performance bottleneck. The tool was designed to provide the user with performance data in an on-line fashion. Consequently it is not necessary to wait for the end of application's execution in order to analyze its performance. Rather, the analysis is being performed constantly and the user can react to improper application behavior as soon as it occurs. This is particularly important in case of application that have long execution times, which are common in the grid environment. The tool works in *XWindows* environment and provides convenient graphical interface for defining measurement and visualizing of performance data.

One of the most important features of the G-PM is its flexibility. The tool can be customized to support a huge range of monitoring scenarios. First, the G-PM provides a number of predefined performance metric for MPI applications. New metrics can be added by the user with an aid of the dedicated Performance Measurement Specification Language (PMSL). The PMSL also provides a *probe* mechanism that can be used to monitor control flow during application execution and to retrieve content of internal application variables. This features are especially useful for the application developer. Using them the developer can design monitoring scenarios usefully in detection of origins of possible performance bottlenecks. Finally, the measurement of both predefined and PMSL-based metrics can be narrowed to any subset of computing sites, nodes, processes or application functions. The measurement of the PMSL metrics can be further narrowed to intervals in application execution that are specified by enclosing probes.

The G-PM monitors the application by appropriate programming of the another CrossGrid service, i.e. OCM-G [1]. The main task of OCM-G is to perform low-level monitoring of application in an distributed way. Also, the initial processing of performance measurement results (e.g. partial aggregation of measured values) is preformed by OCM-G. This approach significantly reduces the computational overhead associated with the monitoring, thus minimizing the influence of the G-PM on the grid application behavior. Furthermore, it reduces the overhead on the user workstation, which is important in case of monitoring large number of processes.

The main novelty of our approach is the combination of on-line monitoring with the support for user-defined, application specific performance metrics. Other on-line monitoring tools either focus on hardware infrastructure (e.g. Network Weather Service [3]) or does not provide user-defined metrics whose functionality is comparable to the one of G-PM (e.g. GRM [4], Paradyn [5], Autopilot [6], TAU [7]).

1.1 Product Usage

The G-PM operates on the end user workstation. The user provides the tool with the identifier of the grid application that is to be monitored, as a command-line parameter. During startup the G-PM establishes the communication with OCM-G and retrieve the list of computing sites, nodes and application processes. At this stage also the list of application functions is retrieved from OCM-G. After initialization the tool can perform application monitoring.

Normally, after starting G-PM the user should define at least one performance measurement. In particular, the user should specify a metric that is to be measured and, if necessary, narrow the measurement to a subset of computing units and/or application functions. The performance measurements can be also specified using the PMSL language. The G-PM does not limit the number of performance measurements that can be performed simultaneously. This is limited only by the speed of the CPU and the size of available memory.

After defining the performance measurement, the user should selects the visualization window for presentation of results (e.g. bargraph diagram, matrix diagram, etc.) and specify its parameters (e.g. scale

partition, update interval, etc.). It is possible use multiple visualization windows simultaneously. Furthermore, each visualization window can present results from multiple performance measurements (with the exception of matrix diagram - see bellow), and each performance measurement can be presented in multiple visualization windows simultaneously.

The G-PM allows to save the current session, i.e. defined performance measurements and parameters of opened visualization windows, to a file. This way the user can restore the session on the next G-PM startup, saving the time needed to setup the tool from the beginning. However, it should be noted that results of performance monitoring are not saved, and thus cannot be restored.

The detailed description of various stages of performance measurement process along with tool functionality and graphical interface is provided in the following sections.

1.2 Running the Product

1.2.1 Operating Requirements

Local hardware requirements

The G-PM requires a typical workstation hardware. No specific limitations are given, as long as the workstation is capable of running graphical applications.

Local software requirements

G-PM is intended to run on RedHat Linux, version 7.3. The G-PM requires shared C++ library, gcc3, version 3.2.2. The G-PM tool requires also the GTK+ library in the version 1.2.10 and the GLib library in versoin 1.2.10. Moreover, XFree86-libs with version 4.2.1 is needed.

Grid infrastructure requirements

In order to use G-PM, the monitoring system OCM-G must be installed on all user interface (UI), computing element (CE), and worker node (WN) machines used by the application to be analysed. The current version of G-PM (0.8.0) requires version 1.9.4 of the OCM-G to be installed. Moreover, Vdt_globus_essentials in version VDTALT1.1.8-14.edg4 should be present in the system for Globus-IO communication.

1.2.2 Step-by-step Setup

Setting up of G-PM requires a working installation of OCM-G, as the G-PM RPM is dependent on the OCM-G RPM package. Please refer to the OCM-G installation and user manuals for more details.

The G-PM tool is distributed as a binary RPM. It can be obtained from: <http://savannah.fzk.de/distribution/crossgrid/autobuilt/i386-rh7.3-gcc2.95.2/wp2/RPMS/cg-gpm-0.8.0-1.i386.rpm> The installation of RPM is done by a typical rpm invocation:

```
rpm -ivh cg-gpm-0.8.0-1.i386.rpm
```

Once the OCM-G is properly configured and G-PM is installed, creation of grid security credentials is required. The user should have a valid Grid proxy certificate on the machine where he or she wants to start G-PM. If necessary, a proxy should be created with grid-proxy-init. Apart from security certificates, no further setup is needed. The user may proceed to staring the performance monitoring and analysis, as detailed in the next section.

1.3 Basic Operation

In order to monitor and analyse the performance of an application, it has to be started in a way conforming to the needs of OCM-G monitoring system. For detailed description of this process, please refer to OCM-G user manual. However, below, we give a short example of the starting procedure.

1.3.1 Starting the OCM-G and the monitored application

Before the application or G-PM can be started, the OCM-G Main Service Manager must be running. It can execute on any machine with inbound connectivity in the port range used by Globus-IO. Usually, the UI machine will be used.

1. Login to the UI machine.
2. Ensure that you have a valid Grid proxy certificate. If necessary, run **grid-proxy-init** to create one.
3. Invoke `cg-ocmg-monitor`. This will print a line looking like

```
Main SM connection string: 8d6383e4:800d
```

The string **8d6383e4:800d** is called the Main SM connection string. It is needed later to tell the application and G-PM where to connect. Note that this string will be different each time you invoke the Main Service Manager.

Once the OCM-G Main Service Manager is running, the application can be started in any suitable way, provided only that you add **--ocmg-appname testApp --ocmg-mainism 8d6383e4:800d** to the command line arguments of each process, where *testApp* is an arbitrary name given to the application, and **8d6383e4:800d** must of course be replaced by the proper connection string printed by the Main Service Manager. In the following, we provide two examples for the start-up of an MPI application.

1. Starting an MPICH-P4 application on a local cluster:
 - Login to the cluster front-end and go to the directory where the application's executable file is located.
 - Ensure that your application has been linked with the instrumented MPICH-P4 library (c.f. the OCM-G User's Manual).
 - Run the application using the **mpirun** command, e.g.:

```
mpirun -machinefile machines -np 8 testApp.exe \  
--ocmg-appname testApp --ocmg-mainism 8d6383e4:800d
```

2. Starting an MPICH-G2 application with EDG job submission commands:
 - Login to the cluster front-end and go to the directory where the application's executable file and the job description (JDL) file is located.
 - Be sure that you have a valid Grid proxy certificate.
 - Edit the JDL file and add the **--ocmg-appname** and **--ocmg-mainism** options with correct parameters, e.g. **--ocmg-appname testApp --ocmg-mainism 8d6383e4:800d**, to the job's command line arguments.
 - Submit the job using e.g.

```
edg-job-submit testApp.jdl
```

Note that if you do **not** use MPI, you must ensure that a valid Grid proxy certificate exists on each node used by the application (c.f. the OCM-G User's Manual).

1.3.2 Starting the G-PM and performing basic measurements

The G-PM can be started with the following command:

```
cg-gpm testApp --num-procs 8 --terminate --ocmg-mainism 8d6383e4:800d
```

where:

testApp is the application identifier specified during the submission of the monitored application.

--num-procs 8 instructs G-PM to wait until 8 processes have started. This allows G-PM to be started even before the application job is actually running.

--terminate tells G-PM to shut down the monitoring system when it exits (the application, however, continues; it just no longer can be monitored).

--ocmg-mainism 8d6383e4:800d specifies the connection string to the Main Service Manager.

After G-PM successfully connected to all application processes, it prints a message like:

```
Attached to application.  
Init complete. Got 8 processes.  
Synchronizing clocks, please be patient ...
```

After a while (ca. 30 seconds, depending on the number of sites and worker nodes used), the Main Window appears. The window is depicted in Fig. 1.1.

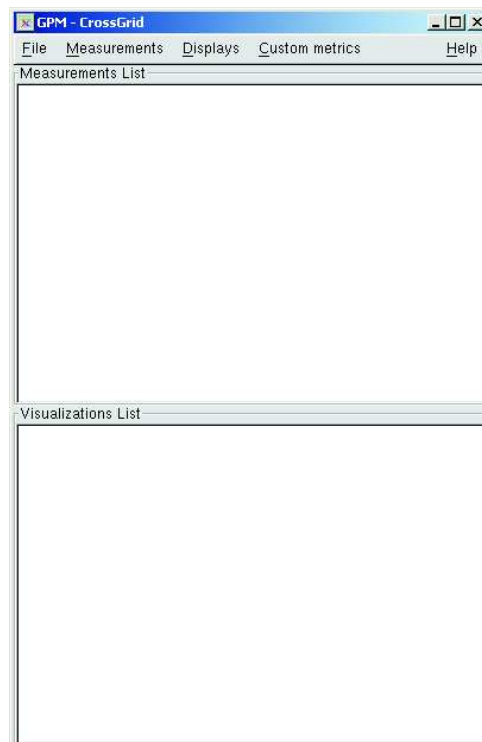


Figure 1.1: The G-PM main window

To create a performance measurement, the user should choose *Measurements->New* from the main menu. The Measurement Definition Window should appear. It is depicted in Fig. 1.2.

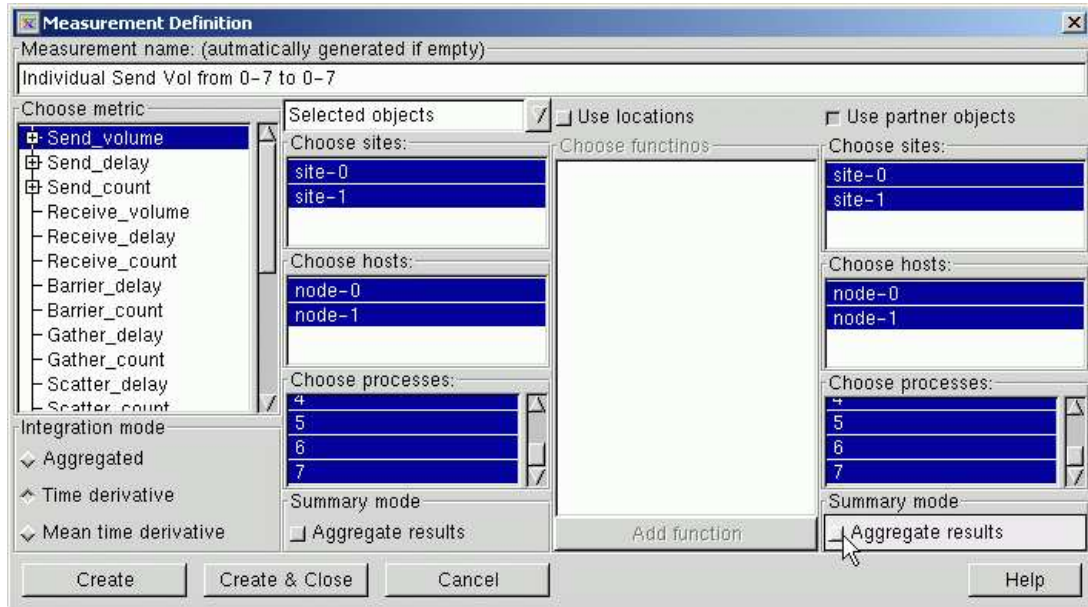


Figure 1.2: Specification of the performance measurement

After the performance measurement window is displayed, the user should specify which performance value should be measured, where it should be measured and how consecutive results should be integrated. This is done in the following steps:

1. The metric is selected from the leftmost panel. For example, the MPI.Send_volume metric can be selected.
2. The objects on which the measurement should be done are specified on the next panel. In this example, the Selected Object option from the pull-down list at the top of the panel is selected. This enables the monitoring of particular processes, as opposed to the Whole Application option, which is used to monitor the selected metric for the whole application.
3. The sites, hosts and processes are chosen. The process identifier 0 to 7 here is the MPI rank of the process.
4. The integration mode should be specified in the bottom-left panel of the window. In this example, a mean value of the send volume for each update interval is measured; thus the Time Derivative option is checked.
5. After the parameters are specified, the OK button creates the performance measurement.

After the performance measurement is specified, the new measurement appears in the G-PM main window, as depicted in Fig. 1.3.

Next, the user should create a new visualization window. This is done in the following steps.

1. Select the defined measurements in the upper panel of the G-PM main window and select Displays->New display from the main menu.
2. The Visualization Definition Window should appear. It is depicted in Fig. 1.4.
3. Specify a desired visualization type in the upper-left panel of the window. In 1.4, a Matrix visualization is selected.

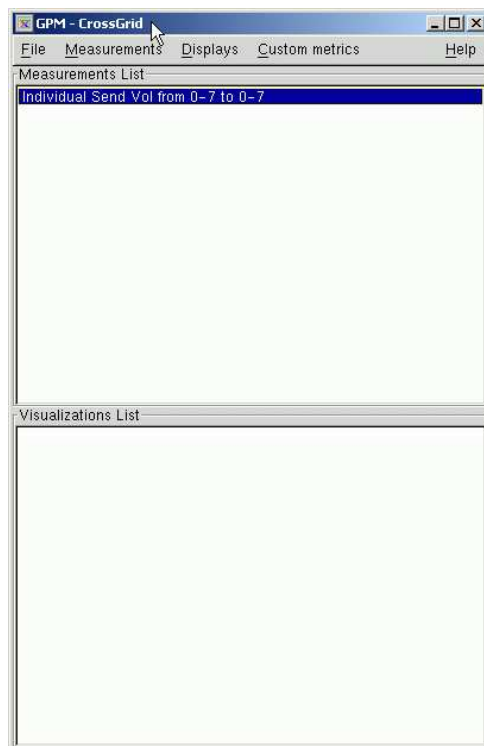


Figure 1.3: G-PM Main Window with the created measurement

4. Additional parameters of the visualization can be specified if necessary:
 - Time Mode of the display - currently only the Real Time is implemented.
 - Partition of the axis scale: it can be either Linear or Logarithmic. In the latter case the values are displayed against a log₁₀-type scale.
 - Behaviour of the axis when displayed values exceed current scale boundaries. Two modes are possible: Variable - the axis is rescaled to the new values and Fixed - the axis is not rescaled and the displayed curve/bar spans the whole display range.
 - Lower/Upper boundary of the scale. These are the initial scale boundaries. However, if Fixed is checked, they are preserved during the whole measurement process.
 - Update interval - i.e. the interval between consecutive updates to the display window.
5. After the parameters of the visualization are specified, The user should click the OK button. A new visualization window should appear, as presented in Fig. 1.5.

The default behaviour when an application is started with OCM-G monitoring is that the application waits at the beginning, until it is started by the G-PM tool. This allows for monitor the performance behaviour from the very beginning of the execution. To start the application, select *File->Start program* from the main menu.

1.4 Advanced Features

The above-described means for inspecting performance are very useful for typical situations. However, often a need to measure and visualize data more closely related to the application's structure and execution

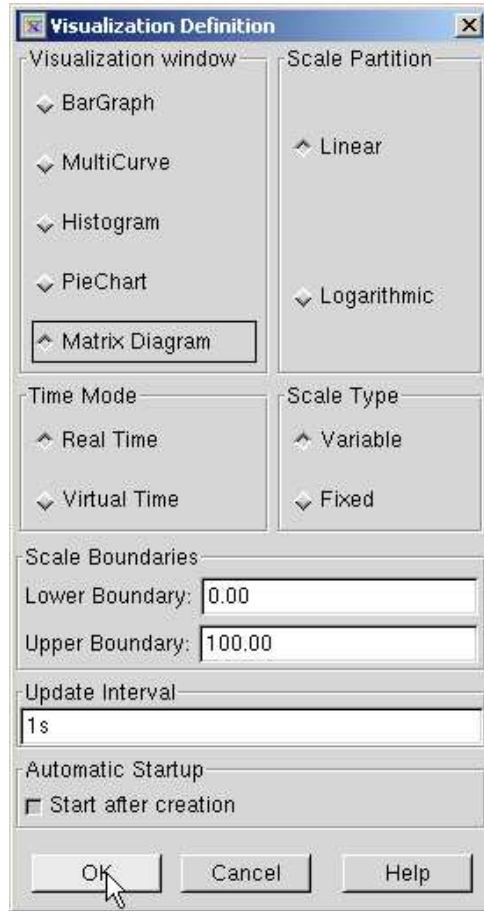


Figure 1.4: Choosing the appropriate visualization

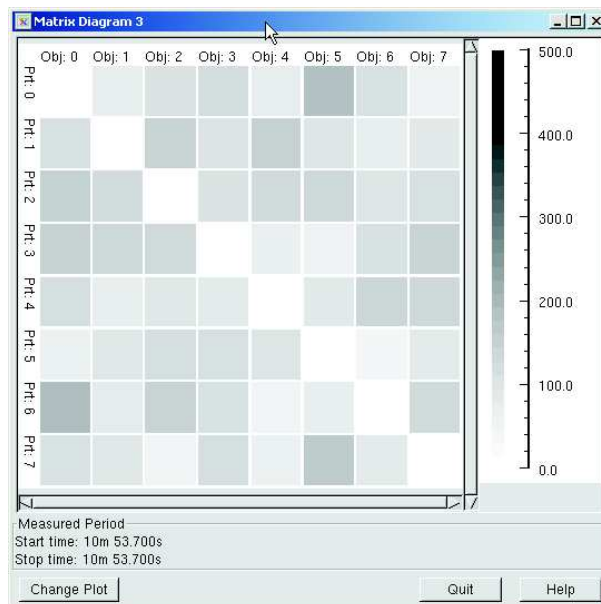


Figure 1.5: The created visualization window

behaviour is required. G-PM supports such measurements via user-defined metrics, i.e. measurable quantities, which can be defined by the user *at runtime*. The definition of a user-defined metrics can also take into account the occurrences of certain events in the application program. These events usually mark interesting points in the application's execution, e.g. transitions between execution phases or iterations of a program's main loop. Since these points are application-specific, the programmer has to mark them in the application's source code by inserting *probes*, which are just calls to (empty) functions. A probe must receive an integer parameter, the *virtual time* that indicates which events (e.g. on different processes) belong together. In addition, other application-specific data can be passed to a probe.

Progress of Application Execution

Assume that in a simulation application, a probe has been inserted at the end of the time loop. The probe receives the simulated time as an additional parameter. A very simple use of this probe is a user-defined metric that determines the application's progress, i.e. the current simulation time. Such a metric can be defined by the user via G-PM's Performance Metrics Specification Language PMSL in the "Edit Custom Metrics" window depicted in Fig. 1.6. This specification states that the value of the metrics is the parameter `simtime` provided by the execution of probe `loop_stop`.

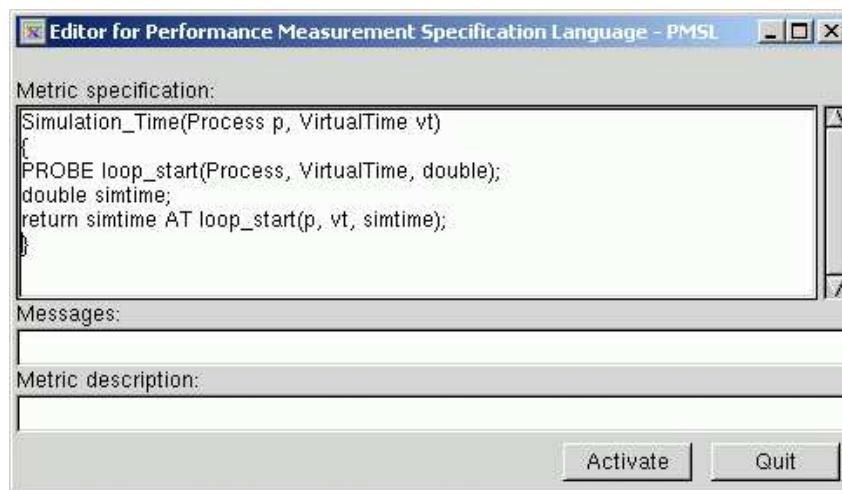


Figure 1.6: Specifying the new metric in "Edit custom metric" window

Once the new metric is parsed, it is added to the tree of all metrics, and can be chosen in the "Measurement Definition" window similar to any built-in metric, as depicted in Fig. 1.7.

The value then can be visualized by G-PM e.g. as a bar graph shown in Fig. 1.8, which acts as an on-line progress bar.

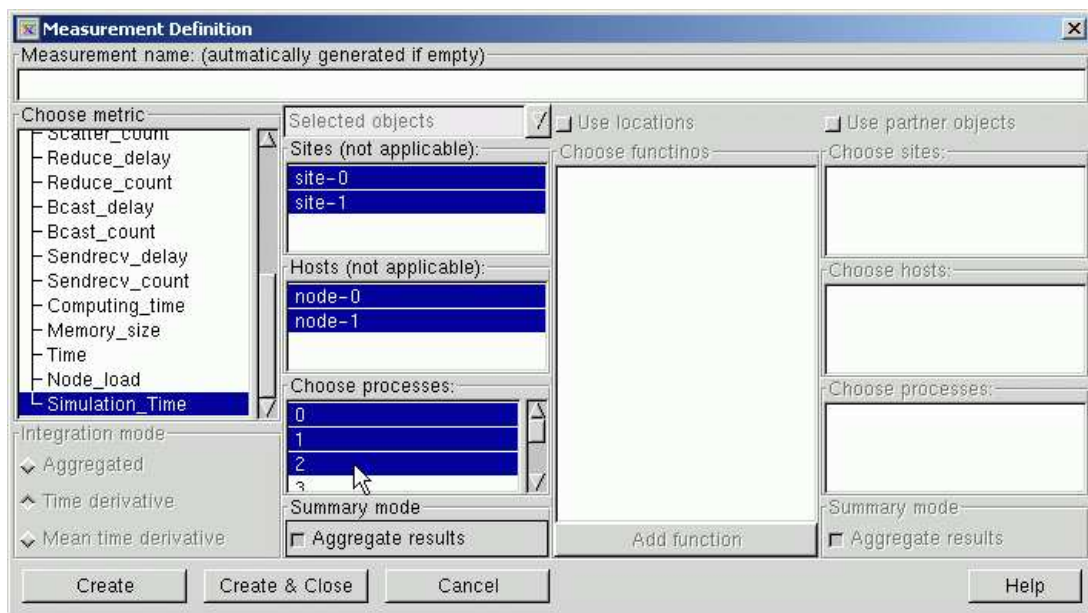


Figure 1.7: New metric in the "Measurement Definition" window

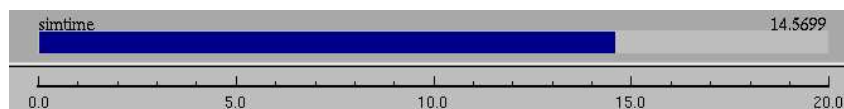


Figure 1.8: Bar graph showing application's progress (simulation time)

2 User Interface Reference Guide

This section presents various parts of the G-PM graphical interface. The functionality of all windows, menu items, buttons, check-boxes and input fields is described. Screenshots for all of the tool windows are provided.

2.1 Main window of the G-PM

The main window of the G-PM tool is presented in Fig. 2.1. The top part of the window contains menu bar whose structure and functionality is described in sect. 2.2. Under the menu bar there are two list widgets. The first one contains the description of all measurements that are currently available in the G-PM, whereas the second lists all currently displayed visualization windows. The commands from the main menu that are connected with measurements or visualization windows usually refer to the items selected in these lists.



Figure 2.1: The G-PM main window

2.2 Main Menu

The main menu of the G-PM tool, the main control centre for the application, consists of five submenus:

- File
- Measurements
- Displays
- Custom metrics
- Help

The File submenu contains the following commands associated with the handling of files and the monitored application:

- Start program - wakes the processes of the application whose performance is measured
- Start program and selected visualizations- wakes the processes of the application whose performance is measured and starts all selected visualizations
- Load settings - loads settings, this is equivalent to successively loading measurements and displays
- Load measurements - loads the contents of the list of measurements from the selected settings file
- Load displays - loads the contents of the list of displays from the selected settings file
- Save settings - saves the contents of the lists of measurements and displays in a settings file with a name chosen by the user
- Exit - exits the G-PM tool, leaving the application running

The Measurement submenu is associated with the measurements list and contains the commands that enable manipulating the list. The submenu contains the following commands:

- New - Opens a "Measurement Definition" window with default measurement settings for a user to create a new measurement which is then appended to the measurements list
- Remove - Removes selected measurements from the measurements list
- Move to Top - Moves selected measurements to the top of the measurements list
- Sort - Sorts selected measurements within the measurements list.

The Display submenu is associated with the list of displays and contains the commands that enable manipulating the list. The submenu contains the following commands:

- New Display - Opens a "Display Definition" window with the measurement from the measurements list as a basis for creating a display. The created display is appended to the visualizations list
- Remove Displays - Removes the selected displays from the visualizations list
- Start Displays - Starts the displays selected in the visualizations list
- Stop Displays - Stops the displays selected in the visualizations list
- Update Interval - Changes the update intervals of the displays selected in the visualizations list according to the ones selected from the submenu:

- Half - divides the current values by 2
- Double - multiplies the current values by 2
- Set - opens a window to enter a new update interval value
- Raise Displays - Raises the displays selected in the visualizations list

The "Custom metrics" submenu contains commands for the management of user-defined metrics. The submenu contains the following commands:

- New custom metric - Opens an empty "Edit custom metric" window to create and edit the new user-defined metric specification
- Load custom metric - Loads one or more specifications of user-defined metrics from a selected file
- Save custom metric - Open a window to choose a user-defined metric to save and enter the file name for the metric being saved

The "Help" submenu contains the commands that show information on the G-PM application as well as provide some guidance on using the application.

2.3 The "Measurement Definition" window

The "Measurement Definition" window presented in Fig. 2.2 enables the user to define a new measurement. The window is displayed when the user selects "New measurement" command from the "Measurement" submenu of the Main Menu.

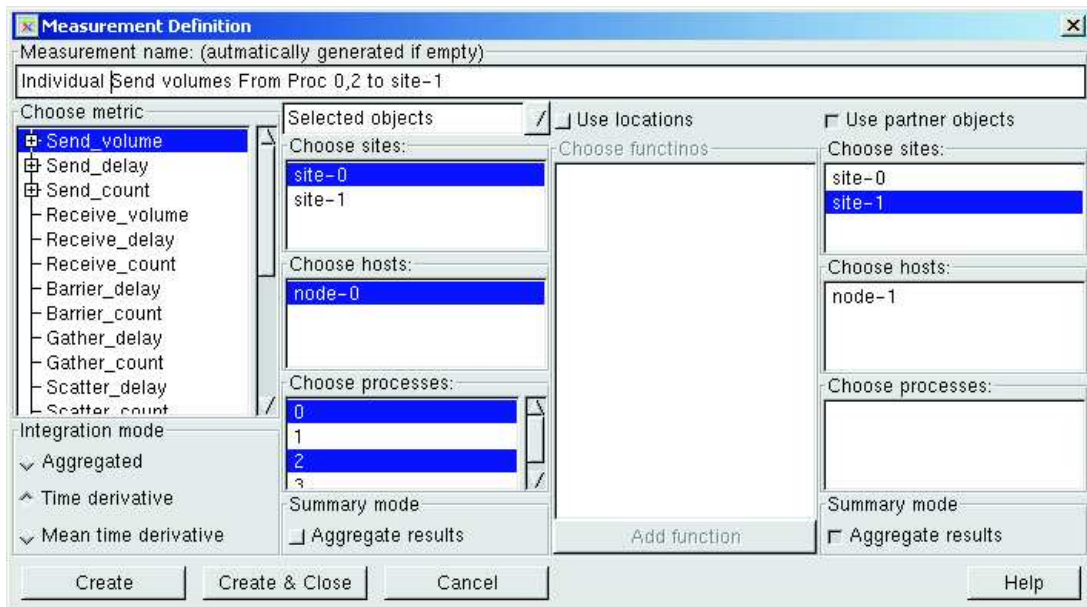


Figure 2.2: The "Measurement definition" window

The window is composed of four panels that specify different measurement aspects: "Metric", "Objects", "Locations" and "Partner Objects". After having selected all the desired parameters within those panels, the user can click "Create", thus creating the new measurement and keeping the window open. The user may also click "Create&Close", creating the measurement and closing the window afterwards, can click "Cancel" to dispose of the "Measurement Definition" window.

2.3.1 The "Metric" panel

The "Objects" panel enables the user to select a measurement metric from the metrics hierarchy. The panel is implemented using a typical "tree browser" widget in which metrics-submetrics relations are represented by levels in the tree. Below the "tree browser" widget are radio buttons, which enables the user to choose the integration mode that specifies whether measurement result is the currently measured value, mean of the currently measured value or the measured values integrated over time.

2.3.2 The "Objects" panel

In the second panel, user may choose, within a pull-down list box, whether the metric should be measured for the whole application (the "Whole application" option), or just for a selected subset of sites, hosts, processes (the "Selected objects" option). When the user selects a site, a list of all hosts in that site, on which the application is running, is displayed in the "hosts" list. The same mechanism applies to selecting a host and the list of application processes within selected host is displayed in "processes" list.

In case when the user chooses multiple objects, he or she is enabled to choose, with the "Aggregate results" check-box, the summary mode whose value specifies whether the measurement result should consist of one, aggregated value or of one value per each object.

2.3.3 The "Locations" panel

The "Locations" panel allows the user to set the location parameter of the metrics. The user, after having enabled the panel with the "Use locations" check-box, has an opportunity to narrow the measurement to the events that happen only within a set of functions. The user can add the functions to be used by clicking the Add function button.

2.3.4 The "Partner Objects" panel

The fourth, "Partner objects" panel is enabled with the "Use partner objects" check box. If the metric chosen in the first panel is not a communication-type metric, the check box is fixed in the not-checked state. If the check-box is checked, the user can choose a set of partner objects for the objects in the "Objects" panel. If some partner objects are chosen, within the communication-type metrics the values are measured only if the communication is between one of the objects and one of the partner objects.

In case the user chooses multiple partner objects, he or she is allowed to select the summary mode, specifying whether to aggregate results into one value or return multiple values, one per each partner object, from the measurement.

2.4 The "Visualization Definition" window

The "Visualization Definition" window presented in Fig. 2.3 enables the user to define an output window for a measurement. The window is displayed when the user selects the "New Display" command from the "Displays" submenu of the main menu.

The "Visualization window" panel in the upper left corner of the window enables the user to select the type of the new display. This can be one of "Bargraph" window, "MultiCurve" window, "Histogram" window, "PieChart" window or "Matrix Diagram" window. The "Time Mode" panel is used to specify whether the measurement should be done in "real time" or in the "virtual time" associated with probe mechanism. It should be noticed that some combinations of monitoring objects, partner objects and summary modes (see "Measurement Definition" window) are not compatible with some display types. E.g. the "Matrix

Diagram” display requires both object and partner object to be specified and measurement results not to be aggregated. In such cases appropriate error/warning window is displayed.

The ”Scale partition” panel on enables the user to specify the partition of the visualization window scale. This can be either linear or logarithmic. The behavior of the display when the measured values are out of the scale can be specified in the ”Scale Type” panel. With the variable scale each out-of-scale value causes the scale to be expanded accordingly, whereas the fixed scale always preserves the initial boundaries (that are specified in the ”Scale Boundary” panel). The scale boundaries must be a proper floating-point numbers, with the upper boundary being greater than the lower. In addition, the logarithmic scale partition requires the boundaries to be greater than 1.0.

The ”Update interval” input box specifies how often the display will read the measured values. The specified value must be in one of the forms: X, Xs, Xms, XsY, or XsYms. Here, X and Y are the integer values, ”s” denotes seconds and ”ms” denotes milliseconds.

The ”Start after creation” check-box species wether the visualization window should start the monitoring process straight after creation. If the check-box is unselected, the window will not measure performance data until it is enabled from the main menu (see sect. 2.2).

After all parameters are specified the user can create a new visualization window by pressing ”OK” button or leave without making any changes by pressing ”Cancel” button.

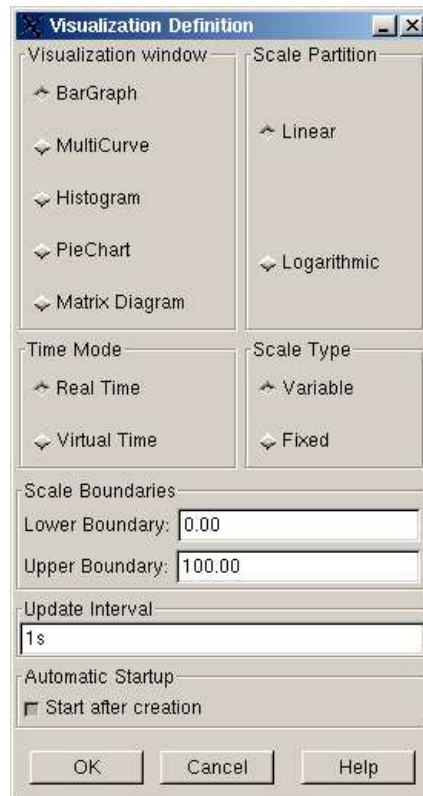


Figure 2.3: The ”Visualization definition” window

2.5 The ”Edit custom metric” window

The ”Edit custom metric” presented in Fig. 2.4 window enables the user to create a new user-defined metric. It is opened from the ”New custom metric” command from the ”Custom metrics” submenu.

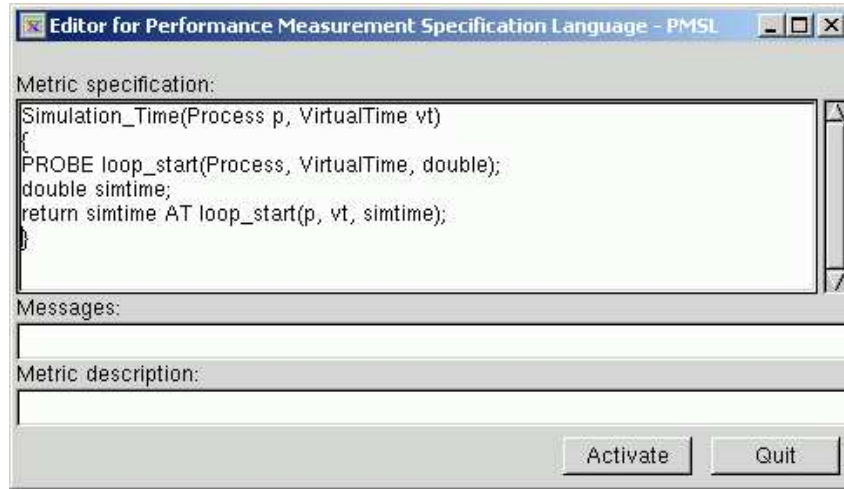


Figure 2.4: The "Edit custom metric" window

The user can enter the specification of the metric in the "Metric specification" field. After completing the work, the user can activate the metric by pressing the "Activate" button. The metric is then parsed, and if the specification does not contains errors, it becomes active. If this is not the case, the error message is displayed in the "Messages" field and the text in the "Metric specification" is placed upon the line with the error.

2.6 The "Bargraph" window

The "Bargraph" window presented in Fig. 2.5 is one of the visualization windows. It shows a single or several independent values in a form of horizontal bars of variable length. The length of a bar specifies the value of the measurement while the name of the measurement and the current value are displayed above the corresponding bar.

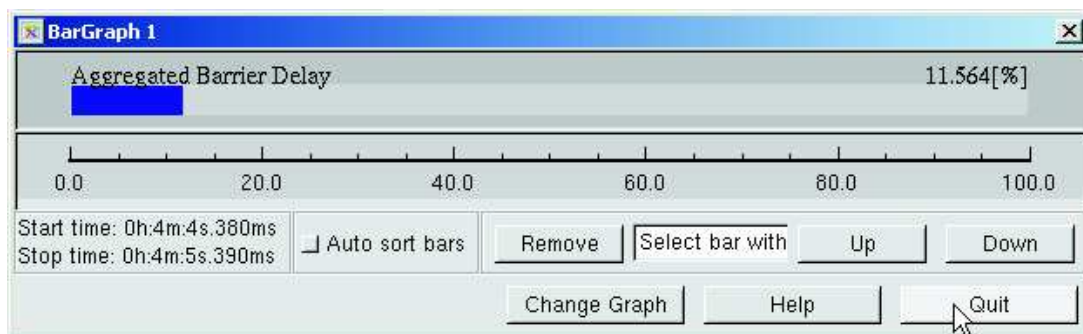


Figure 2.5: The "Bargraph" window

Under the panel with the bar graphs there are three panels. The first panel specifies the start and stop time of the display. Within the second panel, the user can specify, with the "Auto sort graphs" check box, whether the horizontal bars should be automatically sorted vertically, i.e. the bar with a lower measurement value should always be over any bars with higher measurement values.

Within the third panel, the user can specify an own vertical sequence of bars. This can be done by clicking with a mouse on a bar in the graphs panel and then positioning it with the up and down arrows

within the third panel. The bar selected can also be removed from the display with the "Remove" button. With help of the "Change graph" button, the user can change the visualization parameters.

2.7 The "MultiCurve" window

The "MultiCurve" window is shown in Fig. 2.6. It displays multiple values simultaneously, against the time axis, as scrollable curves. Additionally, it enables the user to compute some aggregated values from the measurement results.

The measured values are drawn as curves within the plot area, each having unique color. At the bottom of the area there is a horizontal scrollbar that enables the user to scroll the display. The boundaries of currently displayed time interval are provided below the bottom axis of the plot. If the scrollbar is aligned to the right the display is automatically updated with the arrival of the measured values.

The descriptions of measured values are provided in the list located under the plot area. One of the measured values is always selected within the list. Its description is displayed in the bar at the top of the window and its curve is displayed in red.

It is possible to select an area within the curves display. Such area is visualized as a pair of vertical lines which enclose a time interval. The beginning of the interval can be specified by pressing a left mouse button on the display area. Similarly, the end can be specified by pressing the right mouse button. The boundaries of selected time interval are displayed in the text fields of the "Selected time interval" panel at the bottom of the window. This is used to compute the aggregated value of the curve selected in the measured value list - i.e. this aggregation is performed for this interval only. The aggregated value is displayed in a bar plot to the right of curves plot area. The exact value is also displayed in the text field above. The type of the computed aggregate is specified with the drop-down list button located under the list of performance measurements. Following aggregate values that can be computed: mean value, integral, variance, standard deviation, minimal value and maximal value.

The "Plot resolution" panel shows the length of the time interval displayed within the curves plot. The "Measured Period" panel provides the time interval for which measured values are available.

Various buttons in the window controls the curves plot. The purpose of each of them is explained below:

- Zoom in - decrease the length of the time interval displayed within the curves plot
- Zoom out - increase the length of the time interval displayed within the curves plot
- Focus - starts or stops the automatic scrolling of the display area
- "Change Plot" - changes the visualization parameters.

2.8 The "Histogram" window

The "Histogram" window presented in Fig. 2.7 another visualization windows. It shows a distribution of values for objects or partner objects for a single measurement in a form of vertical bars of variable length. The length of a bar specifies the number of entities for which the values of the measurements fall into the range specified for that bar. The ranges of values associated with the bars are shown below the diagram.

Under the panel with the histogram a panel that specifies the start and stop time of the display is present. In addition, similar to other visualization windows, the buttons that allow for changing the parameters of the display, closing the display or for obtaining some help are present.

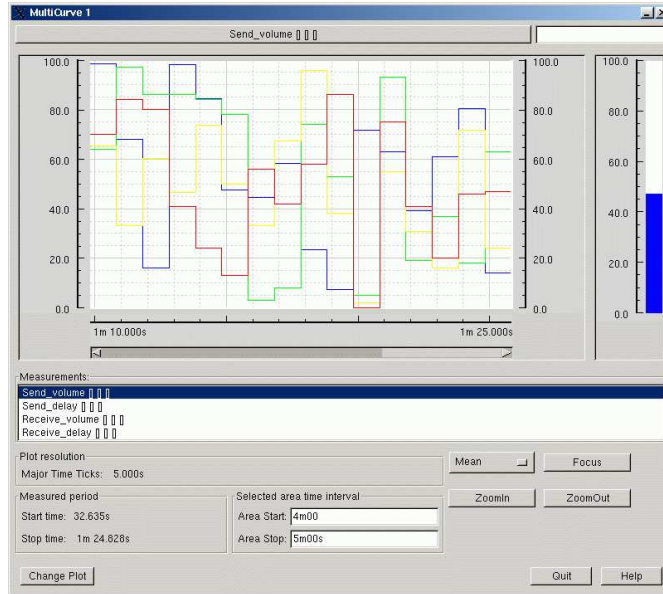


Figure 2.6: The "MultiCurve" window

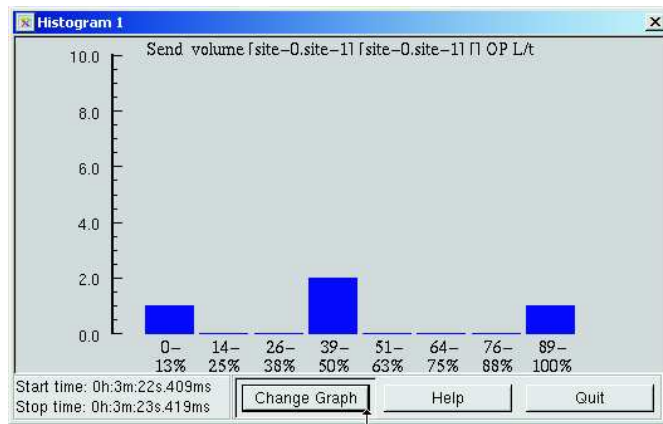


Figure 2.7: The "Histogram" window

2.9 The "PieChart" window

The "PieChart" window is presented in Figure 2.8. The window displays measurement values in a piechart diagram. Therefore, it is suited for visual comparison of relations between measurement results (in terms of their magnitude). The list of displayed performance measurements is displayed below the piechart diagram.

2.10 The "Communication Matrix" window

The "Communication Matrix" window is presented in Fig. 2.9. It can be used to visualize the communication volume or communication associated overhead in the monitored application. The horizontal axis of the diagram refers to the originators of a communication messages, whereas the vertical axis to their destinations. On the intersections, a gray rectangles are displayed. Each rectangle is displayed with a

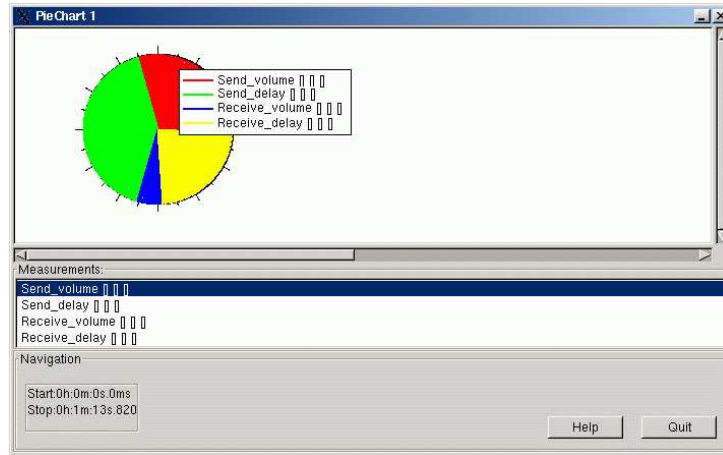


Figure 2.8: The "PieChart" window

colour that reflects the measured volume or time delay relative to the remaining values. More specifically, the lower the measured value the brighter the colour used to display associated rectangle.

The "Communication Matrix" can be used to visualize a single performance measurement only. Furthermore, the measurement must specify both object and partner objects. No aggregation can be used.

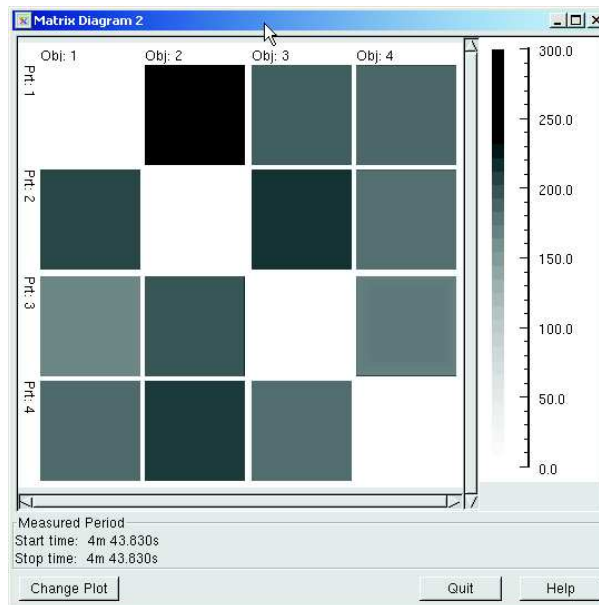


Figure 2.9: The "Communication Matrix" window

3 PMSL Reference Guide

3.1 Introduction

A Grid performance tool should support application specific metrics and also should be able to relate application performance to infrastructure performance. In G-PM, both goals are met by introducing user-defined metrics, which can be specified at run-time. User-defined metrics can combine the results of any existing metrics, but can also take into account additional, application-specific information:

- Occurrences of relevant events in the application's execution, like e.g., the start and end of a user-interaction, or the start of a new iteration of an iterative numerical solver.
- Associations between related events, e.g., between beginning and end of the same activity, or between corresponding events in different application processes.
- Scalar data computed by the application itself, e.g., the residuum in a numerical solver or any other performance-related data.

This application-specific information must be provided by a source code instrumentation performed manually by the application programmer. Technically, the programmer must insert function calls at the proper places in the application code, which receive a *virtual time* and optionally any other scalar data as parameters. The virtual time is an arbitrary, but monotonically increasing integer value, which is used to specify associations between corresponding events, i.e., probe executions.

The following sections will explain in more detail how user-defined metrics are specified and how they can be used.

3.2 PMSL Overview

User-defined metrics are specified in textual form, using the Performance Metrics Specification Language PMSL, which has been specifically designed for G-PM. G-PM uses a textual language rather than a graphical or menu-driven specification because of its greater flexibility. Since the language is exposed to the user, it is reasonably simple and does not force the user to think about the *implementation* of the metrics.

PMSL is a *declarative, functional* language, which only provides single assignment variables and includes neither control flow constructs nor alterable state. Besides the normal arithmetic operators and some statistical functions, PMSL includes a special operator AT, which takes the value of an expression at the time of an event occurrence and thus allows to define metrics based on application-specific events. In addition, the language provides a couple of set operations to support data aggregation.

Syntactically, a metrics specification resembles a function definition in C or C++. The metrics can have a couple of parameters, which represent

1. the object(s) to be measured (e.g., a list of processes),
2. restrictions to the measurement, like partner objects or code regions,
3. a time specification. This can be a point in time or a time interval.

This reflects the fact that a (time-dependent) metrics must be measured for a set of objects, at a specific time or during a given time interval. For *sampled* metrics, where the measurement does not have a duration, the point in time of the measurement can be given either as real time, or as virtual time. The latter is defined by the parameter passed to the probes in the application-specific instrumentation.

The function's return value then specifies, what the result of a measurement of this metrics for a given set of objects and a given measurement time should be.

3.3 PMSL Usage Examples

The following subsections show most of the key features of PMSL, using simple but relevant examples. The exact specification of the PMSL syntax is provided in Section 3.4.

3.3.1 Computing new metrics from already existing ones

This is the most basic use of PMSL. For example, the built-in metrics of G-PM allow to measure the number of message send operations (metrics `Send_count`), as well as the total amount of data sent (`Send_volume`). Given these metrics, the mean size of sent messages can easily be measured by defining the metrics shown in Fig. 3.1.

```
Mean_message_send_size(Process[] p, TimeInterval ti)
{
    return Send_volume(p, ti) / Send_count(p, ti);
}
```

Figure 3.1: Metrics for the mean size of sent messages

The value of this metrics, when measured for a given set of processes (parameter `Process[] p`) and for a given interval of time (parameter `TimeInterval ti`) is just the volume divided by the send count.

3.3.2 Aggregation

PMSL allows to specify more advanced data aggregations than the simple summary modes supported by G-PM's measurement definition window (see Sect. 2.3). As an example, consider a parallel MPI application running across multiple sites. An important metrics for such an application is the amount of inter-site communication. Fig. 3.2 shows the specification of this metrics.

```
Intersite_comm_volume(TimeInterval ti)
{
    Process p, q;
    return SUM(Send_volume(p, q, ti)
              WHERE p.site != q.site);
}
```

Figure 3.2: Metrics for the amount of communication between sites

There are several issues to point out in this example: First, it shows the use of an aggregation function `SUM`, which takes as arguments an expression depending on one or more free variables and a condition on these variables. A free variable is a local variable which has not yet been assigned a value. The aggregation function then iterates over all possible variable values (i.e., all application processes in our case), checks whether the condition is fulfilled, and if so adds the result of the expression to the final

result. Besides the sum, there is also product, minimum, maximum, mean, standard deviation, and a few others.

In Fig. 3.2, the expression `Send_volume(p, q, ti)` denotes the message volume sent by process `p`, restricted to partner process `q`, i.e., the amount of data sent from `p` to `q`. The `SUM` operation then sums up the volumes for all pairs `(p,q)` where `p` is located on a different site than `q`. Besides the `site` attribute, PMSL offers a couple of others, e.g., `node` and `rank`, the latter being the process's MPI rank.

The example also shows that PMSL is a strictly typed language, i.e., the variables `p` and `q` need to be declared with the correct type before they can be used in the `SUM` expression.

Finally, note that the specified metrics does not depend on an object; it implicitly is measured for the whole application.

3.3.3 Data computed by the application

As mentioned in the introduction of this section, metrics should be able to take into account scalar data computed by the application itself. For this to work, the user must insert a proper probe into the application's code. e.g., assume that the residuum value of an iterative solver should be displayed. The user will have to add a probe at the end of the solver's main loop, e.g.,

```
iter_end(i, residuum);
```

where `iter_end` is the probe's name, chosen arbitrarily by the user, `i` is the virtual time, in this case the loop index, and `residuum` is the value computed by the application.

With this instrumentation, the metrics shown in Fig. 3.3 provides the value of `residuum` for each iteration of the solver's main loop.

```
Residuum(VirtualTime vt)
{
    Probe iter_end(Process, VirtualTime, double);
    Process p;
    Process p0 = UNIQUE(p WHERE p.rank == 0);
    double resid;
    return resid AT iter_end(p, vt, resid);
}
```

Figure 3.3: Metrics for a residuum value computed by a solver

The `AT` operator returns the value of the probe's last parameter whenever the probe is executed. Note that in PMSL, the probe has the executing process as an additional parameter. The `UNIQUE` function is used to initialize `p` with the process that has MPI rank 0, i.e., the master process. Note that a measurement of this metrics is based on *virtual* time, i.e., a measurement will return a value each time the probe is executed. Each value also includes a time stamp for the virtual time and the real time. Thus, this measurement is best visualized with a curve diagram, plotting the sequence of values against real time or virtual time (i.e., the number of loop executions).

If there is no need to get the value for each single loop execution, the values can also be aggregated over time using the operators presented in the previous subsection, as shown in Fig. 3.4. The value of this metrics is the maximum of all residuum values, whose (real) time stamp lies within the specified time interval.

There are two notable concepts to point out here: First, PMSL supports arrays with arbitrary index types. An assignment like `val[vt]=Residuum(vt)` is implicitly executed for every possible value of the

```

MaximumResiduum(TimeInterval ti)
{
    VirtualTime vt;
    Value[] val;
    val[vt] = Residuum(vt);
    return MAX(val[vt] WHERE val[vt].time IN ti);
}

```

Figure 3.4: Metrics for the maximum residuum value in a time interval

index, i.e., in the example for every value of the virtual time¹. Second, user defined metrics can not only be based on built-in metrics, but also on user defined ones.

3.3.4 Metrics based on application specific events

Finally, PMSL also allows to compute metrics from application specific events and existing metrics. This allows, e.g., to measure response times or performance indicators for single user interactions of interactive applications.

Consider the iterative solver from the last paragraph, which already has been instrumented with a probe at the end of its main loop. The programmer can re-use this instrumentation to define many other metrics at run-time, while he is analysing the application.

E.g., the amount of data sent to other processes within one iteration can be measured by defining the metrics shown in Fig. 3.5². The expression `Send_count(p, [START, NOW]) AT iter_end(p, vt)` represents the total amount of data sent between the start of the measurement and the time when the probe has been executed. Subtracting the value at the previous loop iteration (at virtual time `vt-1`) from the value at the current one results in the contribution of the current iteration.

```

Send_count_per_iteration(Process p, VirtualTime vt)
{
    PROBE iter_end(Process, VirtualTime);
    return Send_count(p, [START, NOW]) AT iter_end(p, vt)
        - Send_count(p, [START, NOW]) AT iter_end(p, vt-1);
}

```

Figure 3.5: Metrics for the amount of data send in a single loop iteration

Application specific events also allow to specify very high level metrics, which can provide a fast assessment of the application's overall performance. As an example, consider the metrics shown in Fig. 3.6, which computes the number of loop iterations in the given time interval by summing up the constant 1 for each iteration in the interval. Combined with the proper integration mode (see Sect. 2.3), it allows to measure the execution speed of this loop in iterations per second.

¹These assignments are actually implemented similar to macro expansion.

²Note that the probe's result parameter can be omitted in the declaration.

```
Loop_executions(Process p, TimeInterval ti)
{
  PROBE iter_end(Process, VirtualTime);
  VirtualTime vt;
  Value[] val;
  val[vt] = 1 AT iter_end(p, vt);
  return SUM(val[vt] WHERE val[vt].time IN ti);
}
```

Figure 3.6: Metrics for the number of loop iterations in a time interval

3.4 PMSL Syntax

The following table provides an exact specification of the syntax of PMSL, together with a short description of the semantics of special language constructs.

<pre>metricsDefinition: ID '(' parameterDecList ')' body</pre>	<p>A user defined metrics specification consists of the metrics name, its parameter list and its body.</p>
<pre>parameterDecList: parameterDec parameterDecList ',' parameterDec</pre>	<p>The parameter list is a comma separated list of parameter declarations. See Section 3.5.1 for more details.</p>
<pre>parameterDec: type ID</pre>	<p>Each parameter must have a type and an identifier.</p>
<pre>type: ID dimensions</pre>	<p>A type is specified by its name. See Section 3.5.2 for a list of available types.</p>
<pre>dimensions: '' dimensions '[' ID ']'</pre>	<p>One or more optional trailing bracket pairs ('[]') indicate that a variable is a one- or more-dimensional array instead of a scalar.</p>
<pre>body: '' statementList ''</pre>	<p>The body of a specification is just a list of statements.</p>
<pre>statementList: declaration statementList assignment statementList 'RETURN' expr ';' ;</pre>	<p>A statement list can be a declaration, an assignment, or a return statement. The list must end with the RETURN statement, which defines the value of the metrics. The RETURN statement accepts an expression of type Value, double, or int.</p>
<pre>declaration: type names ';' ; type ID '=' expr ';' ; 'PROBE' ID '(' probeParameterTypeList ')' ';' ;</pre>	<p>All variables must be declared before they can be used. A declaration of a scalar variable can include an initialization. Probes also must be declared, including their parameters.</p>
<pre>names: ID names ',' ID</pre>	<p>During declaration, it is also possible to declare more than one variable for the same type by specifying a comma-separated list of names.</p>
<pre>probeParameterTypeList: probeParameterType probeParameterTypeList ',' probeParameterType</pre>	<p>Parameter list for probe declarations: parameters are separated by comma.</p>
<pre>probeParameterType: type type ID</pre>	<p>For probe parameters, only the type must be specified. The specification of a parameter name is optional.</p>
<pre>assignment: lhs '=' expr ';' ;</pre>	<p>PMSL strictly enforces single assignment variables. Thus, each variable can be assigned only once.</p>
<pre>lhs: ID defIndices</pre>	<p>The identifier, which is assigned to, can be indexed, if it has been declared as an array.</p>
<pre>defIndices: '' defIndices '[' ID ']' ;</pre>	<p>The number of indices must match the variables declaration. Indices must be free variables (i.e. variables that have not been assigned to), which also occur on the right hand side of the assignment. The assignment is performed for all possible values of these free variables, i.e., all array elements are assigned to.</p>

<pre> expr: - expr + expr expr '+' expr expr '-' expr expr '*' expr expr '/' expr expr '%' expr term term 'AT' ID '(' parameterList ')'</pre>	<p>The modulus operator is also defined for floating point data types.</p> <p>The AT operator takes the value of the term, whenever the specified probe is executed.</p>
<pre> term: '(' expr ')'</pre> <p> MATHFUNC defIndices '(' expr optWhere ')'</p> <p> MATHFUNC '(' expr ',' parameterList ')'</p> <p> ID '(' parameterList ')'</p> <p> DOUBLE</p> <p> INTEGER</p> <p> ID indices</p> <p> term '.' ID</p> <p> term '.' ID '(' parameterList ')'</p> <p> '[' 'START' ',' expr ']' </p>	<p>Aggregation functions. The value of <code>expr</code> is computed and aggregated for all combinations of free variables, where the <code>optWhere</code> expression is true.</p> <p>Aggregation function with explicitly specified expressions, which should be aggregated.</p> <p>Use of an already defined metrics.</p> <p>Floating point constant.</p> <p>Integer constant.</p> <p>Array element.</p> <p>Returns the value of the named attribute of the object specified by the term.</p> <p>For future use: returns the result of the named method of the object specified by the term.</p> <p>Constructor for values of type <code>TimeInterval</code>. The <code>expr</code> must either evaluate to the constant <code>NOW</code> (allowed only inside the scope of an <code>AT</code> operator) or to the value of the <code>Time</code> parameter passed to the metrics. The term <code>START</code> indicates that the beginning of the time interval always is the time when the measurement of the defined metrics has been started.</p>
<pre> indices: '' indices '[' expr ']'</pre>	<p>Array variables must be indexed with the number of indices given in their declaration.</p>
<pre> parameterList: expr parameterList ',' expr</pre>	<p>Parameter list: comma-separated list of expressions.</p>
<pre> optWhere: '' 'WHERE' boolExpr</pre>	<p>The <code>WHERE</code> clause restricts the values of the free variables taken into account in aggregation functions. If the <code>WHERE</code> clause is missing, all possible values of the free variables are taken into account (e.g., all processes in case of a <code>Process</code> variable).</p>

<pre>boolExpr: boolExpr 'OR' boolExpr boolExpr 'AND' boolExpr 'NOT' boolExpr '(' boolExpr ')' expr 'IN' expr expr RELOP expr</pre>	<p>The IN operator tests whether the object identified by the first expression is contained in the array identified by the second expression. The array is regarded as a set here.</p>
<pre>MATHFUNC: 'SUM' 'PROD' 'MIN' 'MAX' 'MEAN' 'STDEV' 'COUNT' 'UNIQUE'</pre>	<p>Aggregation functions: sum, product, minimum, maximum, mean, and standard deviation of the specified values. COUNT simply returns the number of values, UNIQUE just returns an arbitrary element from the specified set of values.</p>
<pre>RELOP: '==' '!=' '>' '<' '>=' '<='</pre>	<p>These operators compare two scalar values and return a Boolean value.</p>

3.5 Semantic Aspects of PMSL

3.5.1 Parameters of a Metrics Definition

A PMSL metrics can have up to four parameters, which represent

1. the object(s) to be measured (e.g., a list of processes),
2. the partner object(s) for the measurement (i.e., a restriction to the measurement),
3. the code region(s) for the measurement (i.e., a restriction to the measurement),
4. a time specification. This can be a point in real time, an interval in real time, or a point in virtual time.

The only mandatory parameter of a metrics is the time specification, all others are optional, although most metrics will also include an argument for the object(s) to be measured.

The object(s) and partner object(s) parameter can have one of the following types:

- `Process`, or `Process []`
- `Node`, or `Node []`
- `Site`, or `Site []`

The code region(s) parameter must have either type `Region` or `Region []`.

If a metrics just specifies a scalar parameter (i.e., `Process`, `Node`, or `Site` for the object and/or partner object parameter, and `Region` for the code region parameter), G-PM automatically creates a second, overloaded definition of the metrics, which accepts arrays (i.e., sets) of these objects and returns a measurement value, which is the sum of the measurement results for all objects, partner objects, and/or regions which are specified when a measurement of this metrics is requested.

The time parameter can have one of the following types:

- **Time** – this is appropriate for a *sampled* metrics. Measurement results of such a metrics which can be read by G-PM at any time and just provide a performance value for this specific time. An example is the built-in metrics `Memory_size`, wich measures the current memory use of a process at some given point in time.
- **TimeInterval** – this is appropriate for metrics which, when measured, result in a performance value for some interval in time. I.e., the result is some mean or aggregated value over an interval of time. An example is the built-in metrics `Send_volume`, wich measures the amount of data sent by a process during the measurement interval.
- **VirtualTime** – a metrics with a `VirtualTime` is based on at least one probe. The metrics, when measured, result in a performance value for each single execution of this probe. Since probe executions are distinguished by their virtual time, the metrics must receive this virtual time as its argument. As an example, consider the PMSL metrics defined in Fig 3.3 and Fig 3.5.

A metrics only needs to specify the parameters it requires (with the exception of the mandatory time specification), unused parameters can be omitted. It is even possible to specify the parameters in an arbitrary order, however, it is recommended to follow the order specified above.

Thus, possible metrics definitions could look like (the body is omitted here for brevity):

```
Example1(Process p, Time t) { ... }
Example2(Node[] n, Process[] partners, Region r, VirtualTime vt) { ... }
Example3(TimeInterval ti) { ... }
Example4(Region[] r, Time t) { ... }
```

3.5.2 Supported Data Types

The data types supported by PMSL are summarized in the following table.

Type	Meaning
int	(Signed) integer values
double	Floating point values
bool	Boolean values
Value	Measurement values, i.e., result values of a Metrics. A <code>Value</code> object consists of a <code>double</code> value and several time stamps, which can be accessed via attributes (see Section 3.5.5. All operators and functions which can be used with <code>double</code> operands, can also be used with <code>Value</code> operands.
Site	Site objects
Node	Node objects
Process	Process objects
File	File objects (reserved; not yet implemented)
Region	Code region objects
Time	Data type for a point in real time
TimeInterval	Data type for an interval in real time
VirtualTime	Data type for a point in virtual time

In addition to these scalar types, it is also possible to define arrays of arbitrary dimension. E.g.:

```
Value[][] valArray;
```

defines a two-dimensional array of `Value` objects. The array can be assigned to with a statement like

```
valArray[x][y] = Send_volume(x, y, time);
```

where x and y must be free variables (i.e., no value has been assigned to these variables yet). Note that it is not possible to assign to specific array elements, just to the whole array. From a pragmatic point of view, arrays in PMSL can be viewed as macros with arguments, i.e., in the example, when later the term `valArray[u][v]` is used, it is just an abbreviation for `Send_volume(u, v, time)`.

3.5.3 Argument and Result Types of Operators

The following table lists the operands of PMSL and the applicable operand types. Note that PMSL is a strictly typed language, so no type conversion is available.

Operator	First Operand	Second Operand	Result
+, -, *, /, %	int	int	int
	double	double	double
	int	double	double
	double	int	double
	Value	Value	Value
	int, double	Value	Value
	Value	int, double	Value
==, !=	any scalar type T	T	bool
<, <=, >, >=	double, int, Value	double, int, Value	bool
AND, OR, NOT	bool	bool	bool
IN	any scalar type T	T[]	bool
AT	double, int, Value	Probe spec.	Value

The aggregation functions `ADD`, `MULT`, `MIN`, `MAX`, `MEAN`, and `STDEV` accept operands of types `double`, `int`, or `Value`. Their return value is `double`, `int`, or `Value`, respectively.

The aggregation function `COUNT` accept operands of any scalar data type and returns `int`.

The aggregation function and `UNIQUE` accept operands of any scalar data type and returns a value of this type.

3.5.4 Predefined Variables

PMSL provides a couple of predefined symbols (variables), which can be used without declaring them. The following table lists these symbols, their data type, and a short description.

Variable	Type	Meaning
<code>allProcesses</code>	<code>Process []</code>	Array containing all processes of the monitored application.
<code>allNodes</code>	<code>Node []</code>	Array containing all compute nodes used by the monitored application.
<code>allSites</code>	<code>Sites []</code>	Array containing all sites used by the monitored application.
<code>allFiles</code>	<code>File []</code>	Reserved for future use.
<code>allRegions</code>	<code>Region []</code>	Reserved for future use.
<code>NOW</code>	<code>Time</code>	This special variable can be used only in the context of the first operand of the <code>AT</code> operator. It denotes the (real) time when the probe specified as the second operand to <code>AT</code> is executed, i.e., the time when the first operand is evaluated.

3.5.5 Available Attributes

The following table lists all attributes available in PMSL, together with the allowed operand type, the result type, and a short description.

Attribute	Operand Type	Result Type	Meaning
time	Value	Time	This attribute is applicable only to a <code>Value</code> constructed by a metrics based on virtual time (i.e., a metrics receiving a <code>VirtualTime</code> argument). It returns the real time, when the value was constructed, i.e., when the probe that triggered the construction of the value was executed.
upper	TimeInterval	Time	This attribute returns the upper bound of a time interval.
duration	Value	double	This attribute is applicable only to a <code>Value</code> constructed by a non-sampled metrics based on real time (i.e., a metrics receiving a <code>TimeInterval</code> argument). It returns the length of the measurement interval comprised by the <code>Value</code> .
site	Node, Process	Site	This attribute returns the site, where the given worker node or process is located.
node	Process	Node	This attribute returns the worker node, where the given process is located.
rank	Process	int	This attribute returns the MPI rank of the given process (in case of MPI applications; otherwise, it returns the process number passed to <code>ocmg_register</code> when the process registered to the OCM-G).

Note that the use of the `time` attribute is restricted to aggregation over time, i.e., it can be used only within expressions like

```
SUM(val[vt] WHERE val[vt].time IN ti);
```

as shown in Fig. 3.4 and Fig. 3.6.

The `duration` attribute is, e.g., useful to convert aggregated data volumes into data rates:

```
Send_rate(Process[] p, TimeInterval ti)
{
    Value volume = Send_volume(p, ti);
    RETURN volume / volume.duration;
}
```

3.5.6 Available Built-In Metrics

The following table lists all the built-in metrics of G-PM, which can be used within PMSL specifications, together with the applicable parameter types.

Metrics	Objects	Partner Objects	Regions	Time
Send_volume	Process [] or Process or Node [] or Node or Site [] or Site	Process [] or Process or Node [] or Node or Site [] or Site or none	Region [] or Region or none	TimeInterval
MPI_Send_volume				
MPI_Bsend_volume				
MPI_Rsend_volume				
MPI_Ssend_volume				
Send_delay				
MPI_Send_delay				
MPI_Bsend_delay				
MPI_Rsend_delay				
MPI_Ssend_delay				
Send_count				
MPI_Send_count				
MPI_Bsend_count				
MPI_Rsend_count				
MPI_Ssend_count				
Receive_volume	Process [] or Process or Node [] or Node or Site [] or Site	<i>not applicable</i>	Region [] or Region or none	TimeInterval
Receive_delay				
Receive_count				
Barrier_delay				
Barrier_count				
Gather_delay				
Barrier_count				
Scatter_delay				
Scatter_count				
Reduce_delay				
Reduce_count				
Bcast_delay				
Bcast_count				
Sendrecv_delay				
Sendrecv_count				
Computing_time	Process	<i>not applicable</i>	<i>not applicable</i>	Time
Memory_size	Process	<i>not applicable</i>	<i>not applicable</i>	Time
Time	<i>not applicable</i>	<i>not applicable</i>	<i>not applicable</i>	Time
Node_load	Node	<i>not applicable</i>	<i>not applicable</i>	Time

If a parameter is optional (indicated by the line *or none* in the table), it can be omitted when the metrics is “called” in a PMSL specification. Thus, the following examples of metrics uses are legal:

```

Example(Process[] pset, Node n, Region[] rset, TimeInterval ti) {
    // Total number of send operations of processes in pset during
    // time interval ti
    Value v1 = Send_count(pset, ti);

    // Total number of send operations from processes in pset to
    // worker node n during time interval ti
    Value v2 = Send_count(pset, n, ti);

    // Total number of send operations from processes in pset to
    
```

```
// worker node n during time interval ti, which occur in a code
// region contained in rset
Value v3 = Send_count(pset, n, rset, ti);

...
}
```

4 Troubleshooting Q and A

Following, are the most common problems that may be encountered in running and operating the G-PM.

Q: The G-PM does not start properly.

A: There are several possible causes for this behaviour:

- OCM-G may not have been installed properly or the Main Service Manager may not be running.
Solution: double check the OCM-G installation. Start the Main Service Manager as outlined in Sect. 1.3.1.
- The application may not be prepared properly for the use with OCM-G. Use the **cg-ocmg-check-app** utility to check the application's executable (c.f. the OCM-G User's Manual).
Solution: Re-create the executable as described in the OCM-G User's Manual.
- The application identifier provided at the G-PM tool startup may not be correct.
Solution: restart the tool with the application identifier as specified during job submission.
- One of the computers involved may not have a valid Grid proxy certificate.
Solution: create or copy a certificate using **grid-proxy-init** command.

Q: The visualization window was created, however no values are displayed.

A: Typically, one of two situations may cause this type of behaviour:

- The application may not have been started.
Solution: Start the application, by selecting *File->Start program* command from main menu.
- The measured values may be too small for the specified scale boundaries.
Solution: Create a new visualization window with smaller values for the Upper Boundary of the axis scale.
- The application may not be linked against the instrumented MPI libraries needed for communication-related measurements. Use the **cg-ocmg-check-app** utility to check the application's executable (c.f. the OCM-G User's Manual).
Solution: Re-create the executable as described in the OCM-G User's Manual.

Q: After the visualization window has been defined, a large overhead on the user workstation is observed.

A: The Update Interval specified in Visualization Definition Window may be too small.

Solution: Create a new visualization window with larger values for the Update Interval in the Visualization Definition Window.

Q: The visualization window displays two curves. However, one of the curves displays values far greater than the second one - thus rendering its observation impossible.

A: Modify the visualization window, such that it uses logarithmic axis scale, or use different visualization windows.

5 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 - c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions

of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C)
19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliography

[QAP] WP5, CYRFRONET; **Quality Assurance Plan**; Evolving document

- [1] Baliś, B., Bubak, M., Funika, W., Szepieniec, T., and Wismüller, R.: An Infrastructure for Grid Application Monitoring. In: Kranzlmüller, D. et al. (Eds.), *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting*, Sept. - Oct. 2002, Linz, Austria, Lecture Notes in Computer Science 2474, pp. 41-49, Springer-Verlag, 2002.
- [2] *CrossGrid - Development of Grid Environment for interactive Applications*, EU Project, IST-2001-32243, Technical Annex. <http://www.eu-crossgrid.org>
- [3] Wolski, R., Spring, N., and Hayes, J.: *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Future Generation Computer Systems, vol. 15, pp. 757-768, 1999.
- [4] Balaton, Z., Kacsuk, P., Podhorszki, N., and Vajda, F.: From Cluster Monitoring to Grid Monitoring Based on GRM. In: Sakellariou, R. et al. (eds.), *Euro-Par 2001 Parallel Processing, 7th International Euro-Par Conference*, August 2001, Manchester, UK, Lecture Notes in Computer Science 2150, pp. 874-881, Springer-Verlag, 2001.
- [5] Miller, B.P., et al.: *The Paradyn Parallel Performance Measurement Tools*. IEEE Computer, 28(11): 37-46, Nov. 1995.
- [6] Vetter, J.S., and Reed, D.A.: *Real-time Monitoring, Adaptive Control and Interactive Steering of Computational Grids*. The International Journal of High Performance Computing Applications, vol. 14, pp. 357-366, 2000.
- [7] Malony, A., and Shende, S.: Performance Technology for Complex Parallel and Distributed Systems. In: Kotsis, G., and Kacsuk, P. (eds.), *Proc. Third Austrian-Hungarian Workshop on Distributed and Parallel Systems, DAPSYS 2000*, pp. 37-46, Kluwer, 2000.