DELIVERABLE D2.1

PART IV: SOFTWARE REQUIREMENTS SPECIFICATION FOR GRID-ENABLED PERFORMANCE MEASUREMENT AND PERFORMANCE PREDICTION

WP2.4 Interactive and semiautomatic performance evaluation tools

Document Filename:	**CG-2.4-DOC-CYFONET001-SRS**
Work package:	**WP2.4 Interactive and semiautomatic performance evaluation tools**
Partner(s):	**CYFRONET, TUM, USC, CSIC**
Lead Partner:	**CYFRONET**
Config ID:	**CG-2.4-DOC-0001-1-0-PUBLIC-B**
Document classification:	**PUBLIC**

Abstract: This document specifies the software requirements for CrossGrid Task 2.4 'Interactive and semiautomatic performance evaluation tools'.

Delivery Slip

	Name	Partner	Date	Signature
From	WP2	CYFRONET	16/05/2002	
Verified by	Moderator:WP1 Celso Martínez Rivero, Reviewers: WP2: Rafael Menéndez de Llano WP3: Norbert Meyer WP3: Pawel Wolniewicz	CSIC Santander Universidad de Cantabria PSNC PSNC	24/05/2002	
Approved by	Steering Group	WP5		

Document Log

Version	Date	Summary of changes	Author
1-0-DRAFT-A	6 May 2002	Draft version	Włodzimierz Funika Francisco F. Rivera Roland Wismüller
1-0-DRAFT-B	13 May 2002	Almost final draft	Włodzimierz Funika Francisco F. Rivera Roland Wismüller
1-0-PUBLIC-A	30 May 2002	Final version	Włodzimierz Funika Francisco F. Rivera Roland Wismüller
1-0-PUBLIC-B	1 June 2002	Final layout	Holger Marten

CONTENTS

1. INTRODUCTION

1.1. PURPOSE

This document specifies the software requirements for the performance measurement and prediction tool, within Task 2.4 'Interactive and Semiautomatic Performance Evaluation Tools'. The intended audience is both the Task itself and dependent tasks.

1.2. SCOPE

The product of Task 2.4 is a performance measurement tool, called G-PM, which consists of four components (c.f. Fig. 3.1):

(a) a performance measurement component (PMC),
(b) a component for high level analysis (HLAC),
(c) a component for performance prediction (PPC) based on analytical performance models of application kernels,
(d) a user interface and visualization component UIVC.

The performance measurement component will provide the functionality for basic performance measurements of both Grid applications and the Grid environment. The results of these basic measurements can be directly visualized by the visualization component. In addition, they can serve as an input to the high level analysis component and the performance prediction component.

The high level analysis component will support an application and problem specific analysis of the performance data. On the one hand, it will allow to measure application specific performance metrics. On the other hand, it will provide a specification language which allows to combine and correlate different performance measurements in order to derive higher-level metrics. The objective of this approach is to provide more meaningful data to application developers.

The performance prediction component will provide models for the performance of specific computationally intensive kernels. These models can be used to predict the behaviour of the codes under different hardware conditions. These kernels will be mainly provided by partners in the WP1.

The user interface and visualization component will allow to request performance measurements and/or prediction and will display the resulting performance information in various graphical ways. PMC and UIVC will be extensions/adaptations of PATOP.

1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

APART	Automatic Performance Analysis: Resources and Tools ESPRIT IV WG
AVISPA	A VISualizatio tool for PAraiso
ASL	Apart Specification Language
CrossGrid	The EU CrossGrid Project IST-2001-32243
DataGrid	The EU DataGrid Project IST-2000-25182
G-OCM	Grid-enabled OMIS-Compliant Monitor
G-PM	Grid-enabled Performance Measurement tool
GVK	Grid Visualization Kernel

HLAC	High-level analysis component
HPF	High Performance Fortran
MPI	Message Passing Interface
OMIS	On-line Monitoring Interface Specification
PARAISO	Parallel Iterative Solvers Library
PATOP	Performance analysis tool
PMC	Performance measurement component
PMD	Performance measurement data
PPC	Performance Prediction Component
R-GMA	DataGrid relational Grid monitoring architecture
RMD	Raw monitoring data
SQL	Structured query language
UIVC	User interface and visualization component
w.r.t.	with respect to

1.4. REFERENCES

ASL	http://www.par.univie.ac.at/~tf/apart/wp2/report-wp2.ps.gz
AVISPA	http://www.ac.usc.es/~paraiso/
APART	http://www.fz-juelich.de/apart
CrossGrid	CrossGrid Project Technical Annex CROSSGRID ANNEX1_V3.1.DOC
DataGrid	DataGrid Project Technical Annex DATAGRIDANNEX1V3.1.DOC
G-OCM	CrossGrid CG-3.3-DOC-0001-SRS
Motif	http://www.opengroup.org/motif
OMIS	http://wwwbode.informatik.tu-muenchen.de/~omis
PATOP	wwwbode.in.tum.de/Par/tools/Projects/Patop.html
PARAISO	http://www.ac.usc.es/~paraiso/
R-GMA	EU DataGrid Project Deliverable 3.2
TCL	http://www.tcl.tk

1.5. OVERVIEW

This document provides the software requirements for the G-PM performance measurement tool.

Section 2 provides an overall description of the G-PM tool, which specifies the product perspective, functions, user characteristics, constraints, assumptions and dependencies as well as apportioning of requirements.

Section 3 provides a description of specific requirements for the G-PM tool, which comprise external interfaces, functions, performance requirements, logical database requirements, design constraints, standards compliance and software system attributes.

2. OVERALL DESCRIPTION

2.1. PRODUCT PERSPECTIVE

The G-PM performance measurement tool provides measurement information for the user working on or with applications from within WP1. These users will use this information to support the optimization of the code or use of applications by extracting high level performance properties of applications, to predict the performance of the program without running it, to produce basic measurements and visualize them. The tool collects information from the grid monitoring services and the results of benchmarks.

2.1.1. System interfaces

The G-PM tool will provide interfaces to:

1. the grid monitoring system
2. the benchmark results

The requests on Grid applications will comply with the syntax of the monitoring services which provide Grid application-related performance data according to OMIS 2.0. The requests on Grid infrastructure will comply with the syntax and semantics of the services which supply Grid-related performance data.

2.1.2. User interfaces

The G-PM tool will provide a graphical user interface based on X-Windows. It consists of four different classes of windows:

1. The main window provides menus to enter user commands and displays the following information:
   a. a list of all active performance measurements,
   b. a list of all open visualization windows.

   These lists are used to manage (i.e. modify or delete) measurements and visualization windows.
2. A measurement definition window. It allows to specify the performance measurements to be performed. It comprises the type of metrics, objects involved, time interval and location of the measurement.
3. A visualization definition window. It allows to specify, in which way a selected set of performance measurements should be visualized. They comprise visualization windows with which metrics are presented as accumulated over a time interval, as a graph against real time, or as a graph against the number of executions of specified probes.
4. An arbitrary number of visualization windows.

2.1.3. Hardware interfaces

The G-PM tool does not require any special hardware interfaces.

2.1.4. Software interfaces

The performance analysis tool requires the following software packages:

- Linux operating system, kernel 2.2 or higher
- X-Windows X11R6 (XFree86)
- OSF Motif 2.1 widget set
- TCL/TK GUI library v..8.3.4

2.1.5. Communications interfaces

N/A

2.1.6. Memory constraints

N/A

2.1.7. Operations

N/A

2.1.8. Site adaptation requirements

G-PM requires the following site-specific data, which may be either specified during the installation sequence or by means of environment variables:

1. Information needed to establish a communication channel to the Grid monitoring system.
2. Pathname of the directory containing auxiliary files (on-line help files, specification files for performance metrics, analytical models of application kernels)

In addition, a site may provide its own site-specific performance metric specifications and analytical models.

2.2. PRODUCT FUNCTIONS

The G-PM tool enables the user to optimize the performance of the application with support for:

- measurements of various aspects of the program execution on the Grid (computation time, communication time, data volume, response time), which supports finding execution bottlenecks by interpreting the performance data; this part of G-PM tool is realized by Performance Measurement Component (PMC)
- extracting high level performance properties (load imbalance, application specific metrics) of an application; this is the task of High Level Analysis Component (HLAC)
- prediction of how an application will behave under certain conditions and parameters with a tool based on analytical modelthis work is performed by Performance Prediction Component (PPC)
- specification of performance measurements and/or predictions as well as visualization of performance data is realized by the User Interface and Visualization Component (UIVC)

Performance Measurement Component

In order to obtain performance information, we need to access a source of raw data as well as the techniques enabling to process performance metrics based on the raw data.

Functionality from the user's point of view

The user can formulate a request on applications running on the Grid (e.g. CPU utilization, volume of data sent) as well as the Grid infrastructure (e.g. network bandwidth, CPU speed) with UIVC. For example, let us assume that via UIVC the user may want to request the whole volume of communication between two processes. To define the measurement, the user specifies the *delay in receive, the delay in send* metrics, the process identifiers, the interval of the measurement (*when* dimension), the whole program (*where* dimension). In a visualization window, the user will obtain the summary information on the request, which will be presented with one of possible display (e.g. two bargraphs, each per each direction of message passing or as a communication matrix where a pair (column, row) determines a *sending process-receiving process* pair with corresponding communication volume values.

There are two types of measurement requests: application-related and infrastructure-related requests. The *application-related requests* specify information to be returned on the whole or part of a Grid application. These requests can be issued either before the application is started, or while it is already running.
The *infrastructure-related requests* specify information to be returned on the whole or part of the Grid infrastructure.

The PMC will allow to acquire the following types of application related performance data:

1. Data transfer
   a. Amount of data sent / received
   b. Amount of data read from / written to files
2. Resources utilization
   a. CPU usage (computation time)
   b. Memory accesses (number, volume, time)
   c. Disk accesses (number, volume, time)
   d. Cache operations (number, volume, time)
3. Delays due to
   a. communication
   b. synchronization
   c. I/O operations
4. Application specific events and data

The following Grid infrastructure related performance data can be acquired:

1. Availability of resources (on, off)
2. Dynamic resource information
   a. Node
      1. CPU load
   b. Link
      1. Load average
3. Static resource information
   a. Node
      1. CPU architecture
      2. CPU performance
      3. Memory hierarchy features
      4. Memory performance
      5. Secondary storage features
   b. Link
      1. Bandwidth and latency

Measurements can be defined with different granularities in space and time. It should be possible to apportion the measurements w.r.t.

1. Sites
2. Hosts
3. Processes (only for application related requests)
4. Functions inside a Process's code (only for application related requests)

Three different granularites in time will be availabe:

1. A single performance value comprising the whole execution of the application
2. Summary values for a selectable period of time
3. Progression of a performance metrics over time, with selectable resolution. The finest resolution should be in the order of 1s.

The inner working

The PMC's basic function is to accept performance measurement requests from the user interface, HLAC, PPC and to issue corresponding requests to the monitoring services as described in Task 3.3 within WP3. In addition, the PMC has to obtain the resulting replies from the monitoring services and to process them in order to produce the requested performance measurement data. This data is then forwarded to the UIVC, HLAC or PPC.

In order to request e.g. for *the whole volume of communication between two processes,* the tool needs to sum up all the messages transferred between the processes. This can be realized by capturing each event "message sent" between the processes and adding the volume of the message to the sum. Capturing a "message sent" event is realized as a notification on each invocation of a "send message" function of a communication library. The last action is realized by a raw data supplier underlying the performance tool, which are called collectively the monitoring services. This can be a monitoring system which supplies performance data *on-line* or a database system from which performance data is retrieved in *quasi on-line* or *off-line* mode. Thus, accordingly to the goal and scope of the measurement, it is up to the underlying layer to supply the relevant raw monitoring data.

High Level Analysis Component

The high-level analysis component (HLAC) adds another layer of data analysis to G-PM.

Functionality from the users's point of view

The HLAC will provide two major functions to the user:

- It enables to combine and/or correlate performance measurement data from different sources. E.g. it allows to measure load imbalance by comparing an application's CPU usage on each node used. In a similar way, the portion of the maximum network bandwidth obtained by an application can be computed by comparing performance measurement data with benchmark data.
- It allows to measure application specific performance properties, e.g. the time used by one iteration of a solver, the response time of a specific request, convergence rates, etc.

The latter is achieved by inserting probes (i.e. calls to a special, empty function) at strategic places in the application code (e.g. at the beginning of an iteration).

The ASL is used to specify, how application specific performance properties are computed from the data delivered by these probes and the generic performance measurement data. Thus, using the same

probes, different properties can be measured, e.g. min/max/mean time for an iteration, amount of communication for each iteration, etc. The probes may also pass application specific data to the HLAC, e.g. a residuum value, which can be used to compute a solver's convergence rate.

For each application, the ASL specifications of all application specific performance properties can be stored in a configuration file. The user of G-PM can load the proper configuration file at run-time to customize the tool for his application.

The inner working

While the performance measurement component (PMC) transforms raw monitoring data (RMD) to performance measurement data (PMD, see Fig. 3.1), the high-level analysis component transforms PMD to higher-level and/or application specific properties. The main differences between the HLAC and the PMC are:

- While the transformations in the PMC are fixed, the HLAC will be fully configurable w.r.t.
  - the performance measurement data it gets as input
  - the high-level properties it should provide to the user
  - the way how these properties are computed from different types of performance measurement data
- Since HLAC is configurable, it can use / provide application specific performance data. This can be achieved by loading application specific configurations on demand of the user.

Similar to PMC, HLAC must provide transformations in two directions:

- When the user requests some high-level property to be measured, HLAC requests the PMC to measure all the required performance measurement data.
- When the performance measurement data arrives, the high-level properties are computed according to a rule specified in the HLAC's configuration data.

The configuration data is based on the APART Specification Language ASL [ASL]. ASL allows to specify both a data model for the incoming performance measurement data, as well as performance properties. The performance property specification defines how high-level performance properties should be computed from performance measurement data or other high-level performance properties

Performance Prediction Component

The functions of the performance prediction component are the following:

Several analytical models will be developed for the kernels provided by WP1 for different performance values. These models will be functions that depend on code and hardware characteristics. These models can be used to predict the behaviour of the codes under different hardware conditions in such a way that the user can analyse the influence of these code and hardware characteristics. This function is static in the sense that it can be used independently of the execution of the codes. For example, the analytical models can be developed to predict the cost of communications, the computation times, the load balance, the overall runtime, etc. The code characteristics that influence these models are application dependent, some of them are the number of FLOPs, the number and type of communications, the size (amount of information) of these communications, the locality in the access to the data, etc. Finally, the hardware characteristics to be included in the models are, among others, network latency between each pair of nodes, its bandwidth, computation power of each node, memory hierarchy features.

As an example taken from PARAISO, a library of iterative solvers developed by USC for the kernel Conjugate Gradient (that is a sparse solver), the number of floating point operations per iteration is given by the following analytical expression:

$19n + 7a - 3$

where n is the number of columns of the matrix, and a is the number of its entries.

Analogous models can be extracted for other performance features like the number of communications or their costs.

The use of these models will be included in the UIVC, and it could be very useful for application developers to take decisions about the code, in order to make a priori optimisations of the performance.
In addition the same models can be applied at runtime using real hardware parameters provided by the monitoring. With this information, an estimation of the performance under real conditions is obtained. As the models are fast computed, the user can use the tool to analyse new situations close to the hardware conditions at sight.

Eventually, the results of these models at runtime can be used to modify at runtime certain dynamic parameters of the parallelisation of the kernels if the applications allow it.

The performance prediction component will be based on methods and software that have been developed for the PARAISO library and the performance tool AVISPA.

User Interface and Visualization Component

This component enables to use a graphical user interface to specify measurements and display measurement and prediction values. It provides a measurement specification dialog window, a display specification dialog window, and performance display windows.
The UIVC will be based on the user interface of the PATOP performance analysis tool, refer to [PATOP] for more details.

2.3. USER CHARACTERISTICS

The users of the product are developers of Grid applications, i.e. experienced computer scientists. These developers have a good knowledge of the code of the application, which they analyze using G-PM. Nevertheless, the detailed run-time behavior of this code on a Grid infrastructure is often not known to them, not the least because of the dynamic nature of this infrastructure. Since the application's performance is very important for the developers, and since they have the possibility to modify the application's source code, they are usually willing to make a small and limited amount of modifications to their application's code in order to get detailed performance information.

2.4. CONSTRAINTS

The implementation of the G-PM tool is subject to the following constraints:

1. For security reasons, G-PM must only be allowed to gather such performance data as its user is authorized to access. For application related data, this means that only the Grid user who started the application must be allowed to use G-PM in order to analyze it.
2. Some performance measurements done by G-PM require to combine performance data coming from different hosts in the Grid. As a matter of principle, for such combined data the resolution in time is limited by the accuracy of clock synchronization between the hosts.

2.5. ASSUMPTIONS AND DEPENDENCIES

For the performance prediction tool, information about the main kernels of applications of WP1 is needed.

2.6. APPORTIONING OF REQUIREMENTS

The first prototype of G-PM should fulfill the following requirements:

1. It should support applications executing at a single site, i.e. a local cluster of homogeneous workstations.
   Support for applications running at different sites should be added in later prototypes.
2. It should support the measurement of the following application related performance data as specified in Section 2.2: 1.a, 2.a, 3.a, 4
   Support for 1.b, 2.b, 2.c, 2.d, 3.b, 3.c, as well as for Grid infrastructure related performance data should be added in later prototypes.
3. It should include a few specialized examples of high-level performance properties.
   The ability for configuration/adaptation via the ASL should be added in later prototypes.
4. Performance prediction should be available for at least one example kernel. As the first kernel, PARAISO library of iterative solvers and preconditioners for multiprocessor systems called will be used.
   Analytical models for other relevant kernels of the applications developed in WP1 should be included in later prototypes.
5. The user interface and visualization component should support bar-graph displays for summary values and graph displays showing the progression of values over time.
   Other displays should be added in later prototypes.

3. SPECIFIC REQUIREMENTS

3.1. EXTERNAL INTERFACES

The goal of performance measurements is, on one hand, getting performance data based on raw monitoring information, and, on the other hand, getting performance estimations under different hardware circumstances, on demand of the user who has a graphical user interface or on demand from an automatic component. The measurements may refer to the Grid application or the Grid infrastructure.

Figure 3.1 shows the components of the G-PM, as well as the information flows between these components and external software components developed in other Tasks of the CrossGrid project. Compared to the original scheme of information flow within G-PM, there is an additional information flow from benchmarks (Task 2.3) to Performance Prediction Component.

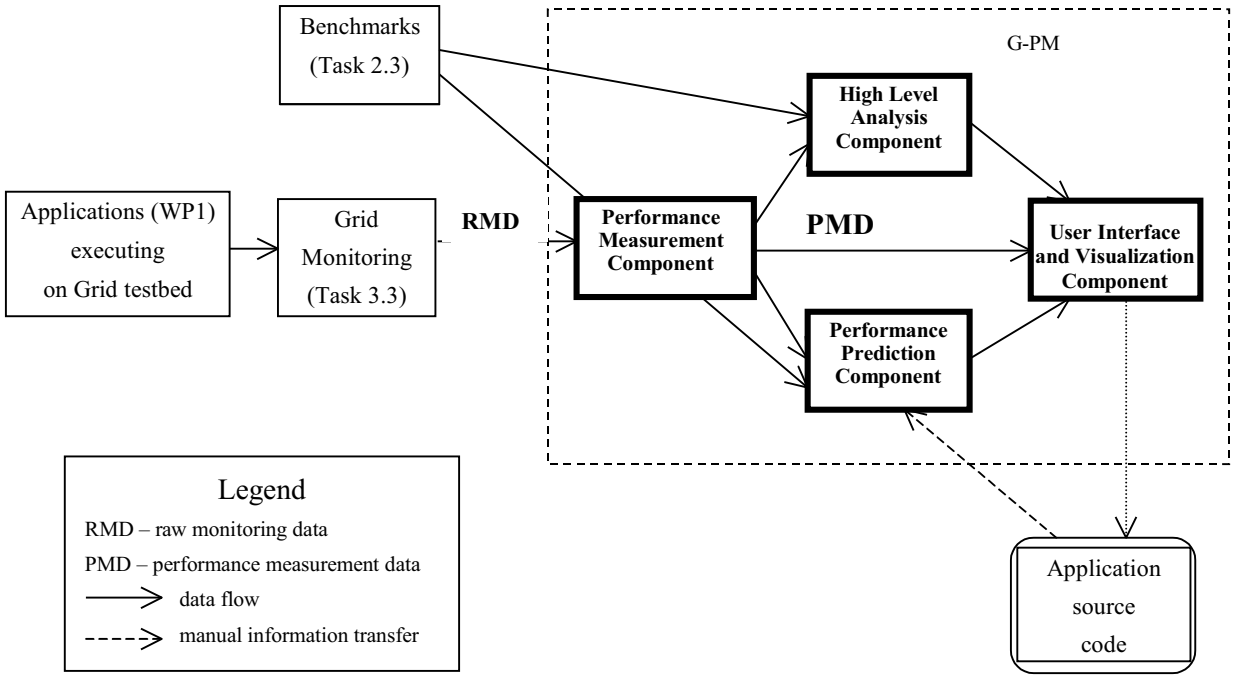

Figure 3.1 Information flows between the components of G-PM and external software components.

The purpose of PMC is to transform measurement requests from the user interface, HLAC, or PPC into corresponding monitoring requests (or a series of such requests) to the monitoring services of CrossGrid. Vice versa, it must transform the responses from the monitorig services into the required measurement values for the requesting modules.

The monitoring requests on Grid applications will comply with the syntax and semantics of the monitoring interface(s) as defined by Task 3.3.

For a detailed description of :
1. performance measurements and visualization refer to [PATOP]

2. syntax and semantics of monitoring requests and specification of data format returned on Grid applications refer to [OMIS]
3. syntax and semantics of monitoring requests and specification of data format returned on Grid infrastructure refer to [R-GMA]
4. performance prediction tool refer to [PARAISO]

DEPENDENCIES:

G-PM will use information supplied by the monitoring services defined in Task 3.3, which may be represented either by unique CrossGrid services (referred to as OMIS) or R-GMA services as defined in DataGrid and referred to as R-GMA. The implementation of the HLAC will make ues of the APART specification language and will be partially based on design ideas developed in the APART project. It is not yet clear how much of the DataGrid software and/or of any software developed in the APART2 project can be used in the implementation of G-PM.

The dependency tables below indicate where information may be most easily acquired from: either the DataGrid R-GMA, or unique Task 3.3 sources via specific interfaces.

Task	Flow	Information	Source
T3.3 [monitoring services]	←T2.4	software requirements specification	
T3.3 [monitoring services]	→ T2.4	*Application-related measurements:*	
		Data transfers	(R-GMA via OMIS??)
		Synchronization delay	(R-GMA via OMIS??)
		I/O delay	(R-GMA via OMIS??)
		CPU utilization	OMIS
		Network utilization	OMIS
		Storage utilization	OMIS
		Communication library function call (MPI-1, MPI-2, MPI-I/O, Dynamite)	OMIS
		Specific library function call (GVK)	OMIS
		Application specific events (e.g. w.r.t. C, C++, Java, Fortran 77/90)	OMIS
		Grid-related measurements:	
		CPU load average	R-GMA (via OMIS??)
		Initial location of processes (to locate monitored processes)	R-GMA (via OMIS??)

		Network delays and latencies Storage load average Static application characteristics Dynamic application characteristics Architecture OS Static Memory Config Dynamic Memory Config	R-GMA (via OMIS??) R-GMA(via OMIS??) R-GMA (via OMIS??) R-GMA (via OMIS??) R-GMA (via OMIS??) R-GMA (via OMIS??) R-GMA (via OMIS??) R-GMA (via OMIS??)
T2.3 [benchmarks]	←T2.4	Performance measurement data	Depends on performance measurement request
T2.3 [benchmarks]	→T2.4	Performance measurement request	

The Grid monitoring system developed in Task 3.3 of the CrossGrid project consists of several subsystems: The **Grid-enabled OMIS-compliant Monitoring system**(OCM-G) is intended to supply application-related performance data. **Jiro** will supply data on the network-related issues. An SQL-based service will enable access to infrastructure related performance data. The service will be provided based on the database approach when real-time constraints are not critical At the moment it is not decided whether the performance data will be accessed via a single monitoring interface compliant with OMIS requirements or the data stream will be split into several data streams according to the semantics of performance data, i.e. those related to application, Grid infrastructure, network issues, and non-invasive monitoring.

Nevertheless, the performance information is roughly divided into those related to application (we associated it with accessing it via OMIS) and those related to Grid infrastructure (we associated it with DataGrid R-GMA facilities owing to the fact that practically all the Grid performance-related data is not time-critical and can be logged/retrieved from the R-GMA).

At the moment we assume that most probably the Grid-related data will also be accessed via the OMIS interface.

The interface between PMC and the monitoring services from within Task 3.3 is built on the basis of processing of monitoring requests/replies to/from the monitoring services, which may belong to one of the following types:

1. information request– supplying information in a reply to a request,
2. manipulation request– executing an action as a reply to a request,
3. event request- capturing events specified in a request to the monitoring services and triggering a list of defined actions.

Let us consider two requests to the monitoring services which illustrate possible kinds of monitoring requests:

1. **A (information) request** *Return CPU load on nodes n_1, n_2, n_3* should return a list of three elements containing information on CPU load on nodes n_1, n_2, and n_3.
2. To support performance analysis of an application, capturing application events and performing associated events is needed, which means the use of **(event) requests,** e.g. , if we want to detect a function call and perform measurements, then requests can be issued on the following events:
   1. *function A has started*
   2. *function A has ended*

Whenever these events are captured, a list of callback actions will be triggered. In the above case actions can be used to execute performance measurements, e.g. to obtain the volume of data transferred (one request is needed) or data send time (two requests are needed). In the first case, a possible request can have the following form: *Whenever the execution of function A has started in process p_1 return the time stamp of the event and the third parameter of the call.* This request is applied to process identified as *p_1*, whereas the result of each execution will be a time stamp and the actual value of the third parameter of the call, e.g. volume of the data to be transferred. In the second case, one request should associate the start of a timer with the start of the function invocation, and the second request should associate the stop of the timer with the end of the function call.

The high-level analysis component poses additional requirements to the monitoring infrastructure:

1. It must support the monitoring of probes in the application's code. This is very similar to the 'function A has started' event specified above.

   The monitoring system must allow to associate the following actions with this event:
   a. start (enable) or stop (disable) specified performance measurements
   b. increment a named counter by a specified amount (which may be defined by one of the parameters passed to the probe)
   c. increment a named integrating counter by a specified amount (which may be defined by one of the parameters passed to the probe)
   d. store a record (which may include some of the parameters passed to the probe) in an event trace
   e. send event information (which may include some of the parameters passed to the probe) directly to the G-PM tool
2. It must allow to access the results of micro-benchmarks:
   a. given a host and a host-related metrics, return the benchmark result for the specified host
   b. given two hosts and a network-related metrics, return the benchmark result for the network connection between the specified hosts

ARCHITECTURE:

As it is not yet decided whether the G-PM tool will access the monitoring services via a single interface or multiple interfaces according to their semantics, we formulate the issues of performance measurement in terms of OMIS only. Provided the decision on the monitoring interface(s) is taken the document will be enhanced accordingly.

Thus, the information flows between the monitoring services and the performance measurement component is limited to a single flow as shown in Fig. 3.1.

3.2. FUNCTIONS

Performance measurements/predictions

The PMC is intended to supply performance measurement data on applications and the Grid infrastructure on request from the UIVC, HLAC, and PPC. The request is transformed into a number of monitoring requests to the monitoring services. The monitoring data replied by these services is then transformed to performance values that are forwarded to the UIVC or HLAC, respectively.

A target of measurements will be viewed by the PMC as one of two sets of objects. The first one will include application-related elements, such as processes, threads, messages, and queues of messages. The second one will include Grid infrastructure-related elements, such as compute elements, network elements, and storage elements.

The PMC will offer a set of measurements on application and Grid performance. A typical measurement comprises:

- value to be measured (time, data volume)
- object(s) specification (site, host, process, thread)
- time interval (complete execution, time/program phases)
- location (program, module, function, block)
- constraints (metrics limit values, communication counterparts).

For each measurement request, a list of objects is specified to which the request should be applied to. For example, return “Execution time of the program PR_1 on hosts [h_1, h_2, h_3]”. The performance measurement component ensures that:

- the measurement is defined as a difference of the value of the end time and the start time of the program on each host
- the measurement is transformed into two monitoring requests to the monitoring services, which define the relevant events of the start and the end of the program execution on each host, compliant with OMIS 2.0
- the relevant requests are submitted to the monitoring services
- the event data is processed and returned as a required measurement value to the requesting component

The PMC will provide the following possibilities to produce application-related performance metrics:

- computing time
- delay in I/O operations
- delay in synchronization operations
- delay in receive operations
- delay in send operations
- amount of data send in messages

- amount of data received in messages
- CPU utilization
- network utilization
- storage utilization
- user defined events and data

The PMC will provide the following possibilities to produce the Grid infrastructure-related performance data (dynamic and static):

- CPU load
- Network delays
- Memory bandwidth
- Availability of resources
- Static application characteristics
- Dynamic application characteristics
- Static Memory Config
- Dynamic Memory Config

High level performance analysis with HLAC

Via the HLAC, the G-PM tool also supports the measurement of metrics specific to the monitored application. To use this feature, the application programmer needs to mark relevant places in the code, like e.g. the beginning and the end of a computation phase. This marking is done by inserting a function call (*probe*) into the application's source code. An arbitrary number of scalar parameters may be passed to this probe, which allows to provide application-specific data relevant for performance analysis (such as e.g. problem size).

All probes available in an application have to be defined in an interface definition file. A utility coming with the G-PM tool will create code from these definitions, which will then be linked to the application.

Once these probes are available in an application, the G-PM tool allows to measure:

- any aforementioned application-related or infrastructure-related metrics in any execution phase defined by two (not necessarily different) probes
- application-specific metrics defined by any parameter passed to any probe
- any metrics that can be (recursively) derived from existing metrics

The measured values can be:

- accumulated over the whole run-time of the application
- plotted as a graph against real time
- plotted as a graph against the number of executions of specified probes (i.e. against the application's execution phases)

The rules to compute these application-specific metrics are defined in a specification file which is based on the APART specification language ASL [ASL]. The proper specification file can be loaded

into the G-PM tool at run-time, depending on the monitored application. Once the specification file is loaded, all defined metrics appear in the UIVC's measurement definition window and can be selected in the same way as the standard metrics.

It should be noted that the probes in the application code themselves do not define any measurement, they just mark interesting points in the execution. An actual measurement is performed only when requested by the user via the UIVC.

In addition to application-specific ASL files, the G-PM tool will come with a generic ASL file defining higher-level metrics independent of a specific application. It will define the following derived metrics:

- load imbalance
- relative CPU and network performance, i.e. application performance as a percentage of the execution platform's 'optimal' performance, which is measured via micro-benchmarks
- more metrics will be defined later

Performance visualization with UIVC

To specify measurement requests and to display measurement and prediction values, the user will be enabled to use a graphic user interface. The user defines measurements in a *measurement specification dialog window*. This window is activated when a corresponding item in a measurement menu of the main GUI window is selected. The measurements may be performed with various *resolutions in space (measurement location)*, i.e. refer to the whole system, sets of nodes, sets of processes or threads or particular nodes, processes, threads, all functions, subsets of functions or any other fragments of a program, and *resolutions in time (measurement interval)*, i.e. refer to the complete execution or particular phases. In communication related measurements, the user can specify the counterparts of the location-related part of the measurement.
Once specified the measurement the user can select the appropriate output window to display the results of the measurement or prediction, in a *display specification dialog window*. The measurements to be displayed in the new output window have to be selected in the measurement list of the main dialog window. The definitions of measurements and displays can be stored in files and can be loaded from a file.
Three kinds of output windows will be supported:

- performance data is displayed *over a time axis*, with scrolling over the complete measurement interval,
- *single values* for each metric computed for a specified measurement interval are displayed,
- particular *event occurences* are displayed over the complete measurement interval.

The measurements can be repeatedly *stopped* and *started* for the existing output windows.

3.3. PERFORMANCE REQUIREMENTS

1. To provide efficiency the PMC can choose between the different modes of performance data gathering and delivering to the relevant tools, which are provided by the monitoring services:
   - buffering performance data in application address space; refer to [OCM]
   - using efficient data structures to store pre-processed data; refer to [OCM]
   - using database structures to store and retrieve logged data when needed; refer to [R-GMA]

For each measurement request, the PMC shall use the method that results in the smallest perturbation of the monitored application.

2. The dynamic data on applications shall be accessible within a time that is not significantly higher than the communication latency between the host where the monitored piece of the application is located and the host where the G-PM tool is executed.
3. The non-time critical data on Grid infrastructure shall be accessible in less than 30 s.
4. If no measurements are defined, the overhead of a probe must be negligible, i.e. it must not be significantly higher than the time needed to call a function which immediately returns.
5. If a measurement is defined for a specific probe, the probe's overhead should be less than 100 μs.

3.4. LOGICAL DATABASE REQUIREMENTS

N/A

3.5. DESIGN CONSTRAINTS

3.6. STANDARDS COMPLIANCE

Where applicable, the software will comply with SQL. In addition, it would comply with OMIS 2.0 and R-GMA, which however are not formal standards.

3.7. SOFTWARE SYSTEM ATTRIBUTES

4. APPENDIXES

Use cases from performance measurement component:

1. Simple performance measurements on application:

- The user may want to get info on the program's spending time in communication, I/O and computations within an interval of time (logical phase or absolute time value).

The user should be able either to start the application under the control of G-PM or to attach to a running application. The user gets the information on hosts and processes from the Grid infrastructure (e.g. the resource manager or MPICH-G2, via the monitoring services). Based on this information the user should be able to provide a job ID as an argument when attaching G-PM to the application. Next the following scenarios of getting info on the application may happen – the user may want to know:

  - *data transfer volume* within a loop/iteration/block. The measurement necessitate to instrument communication functions and a fragment of the code. Communication functions should be instrumented automatically whereas the insertion of probes before and after the relevant places in the code is the application programmer's task. The idea of instrumentation is that this enables the user to get not only the above metrics but also all possible metrics (time, number of messages, ...). The probes supply: the name of the host, program, code fragment, optional parameters (number of iteration(s), time phase (time interval, logical phase id, e.g. step number).
  - the above information may be restricted to particular sites, hosts, processes, threads, message types.

- The user may want to get info on the values of CPU (network, storage) utilization within a fragment of code (block/loop/iteration), a logical phase of a program, the whole program.
- The user may need to know which communication links feature the transfer time over 1 s and under 2 s for messages over 1 MB.
-

2. Simple performance measurements on Grid infrastructure:

- Measure a network bandwidth between two sites,
- Measure a network metrics between two hosts within a site,
- Measure memory access bandwidth on a host,
- Measure a minimal CPU load on a set of hosts.

Use case for the high level analysis component:

We assume the following situation: we have the medical application developed in Task 1.1 of the CrossGrid project, using the Grid Visualization Kernel (GVK) for the communication between the application's simulation component and its visualization component. We will call the host(s) where the simulation is running the server(s), while the host that shows the visualized results will be called the client. The programmer wants to analyse the performance of the visualization, especially the impacts of GVK.

1. In order to acquire GVK specific performance data, the programmer (manually) inserts three probes into the source code of GVK:
   a. at a place immediately after (on the server side) GVK is requested to visualize a frame,

b. at a place immediately after (on the server side) the data is sent to the client,

c. at a place immediately before the data is passed to the graphics engine on the client side. As an additional argument, this probe receives the size of the data.

After inserting the probes, GVK needs to be recompiled.

2. The programmer writes an ASL specification for new metrics related to the GVK, for instance:

   a. *Generated image frames per second* = 1 / (Time between successive invocations of probe a).

   b. *Compression factor* = (data size passed to probe c) / (communication volume from server to client between executions of probe a and b)

   c. *GVK processing time for a frame* = time interval between the execution of a and the execution of probe b

   Notes:

   - metric specifications are not shown in ASL syntax in this use case
   - the same probes are used to define different metrics
   - 'communication volume between server and client' is a standard measurement provided by the PMC
   - Since the probes outlined in 1. and the metrics outlined in 2. are of more general interest, they will probably have already been inserted / specified by the developer of GVK. In this case, the GVK libraries will already come with an accompanying ASL specification file for performance analysis.

3. The programmer links his application with the instrumented GVK library from step 1, as well as with an instrumented communication library. The latter one is part of the G-PM distribution.
4. The programmer starts his application under control of G-PM.
5. The programmer loads the GVK-specific ASL file into G-PM. This is the file created in step 2. After the file has been loaded, the measurement definition dialog of the UIVC shows the GVK-specific metric in addition to the standard metrics.
6. The programmer selects a measurement for 'gererated image frames per second' in the measurement definition window. He can restrict the measurements to specific processes by choosing them from a list. The list shows only the appropriate processes, i.e. those linked with the instrumented GVK library.
7. The programmer selects a visualization window, where the measurement results should be visualized (e.g. as a function over time).
8. Assume the programmer now wants to measure the communication volume within the simulation component *for each generated image frame.* In this case, he would perform the following steps:

   a. Open a window in G-PM that allows to define a new metric using ASL.

   b. Define the metrics:

      Communication volume per frame = (Communication volume between simulation processes between successive executions of probe a)

   c. Optionally: store the definition in a file for later re-use

   d. After this, the new metrics will appear in the measurement definition window and can be used like any of the standard measurements

Use cases from performance prediction component

Assumption: The applications developers provided the parallel kernels to be considered by the tool.

1. The parallel kernels are analysed in detail. Some events are counted: FLOPs, number of communications, their size, etc. All these events depend on the applications parameters: size of arrays, number of iterations of loops, etc. Some of them could be static, others could be established at runtime, when, for example, they depend on the input data for the application. In addition, these parameters can depend on the parallel implementation of the kernels: type of distribution of data among processes, size of the blocks, ... So, in other words, these parameters are application dependent.

   After this analysis, the events are resumed into analytical functions of all of these parameters. The following table shows how these analytical functions may look like:

2.

KERNEL	Number of FLOPS
CG	19n+7a-3
BiCG	23n+9a-4
GMRES	(6r+13)n+(2r+7)a-2
QMR	35n+9a+16
SOR	10n+5a-1

   Where parameters *a*, *n*, and *r* are related to the problem size.

3. Then, the prediction of the performance can be performed based on architectural properties: CPU power of each node responsible of each process, latency and bandwidth for the communication between any pair of nodes, etc. The result must be an analytical function on these hardware parameters that gives predictions about: computation times, communications times, waiting times, etc. And finally, the overall execution time for the kernels.

4. Using these functions, that are easily computed because they are simply analytical functions, a visualization tool is intended to help the user (the applications developer) to see how the kernels are affected by changes on both, the applications and the architecture parameters. This information could be valuable for knowing the behaviour of the kernels on different circumstances.

5. The user can analyse the influence, on the communication times of the kernel, of the latency in order to answer questions like: how fast should be the kernel if the latency of the network is twice faster, in addition the user can explore the influence on the overall execution time on the CPU power of some node, and see this in a 2-D figure. The figures that follow show the number and size of the communications *sent* and *received* for a kernel of PARAISO using 5 processors labelled as PE0 to PE4 (Fig.A1), and the amount of time spent in computations, in *sending* and in *receiving* messages for a particular processor (Fig.A2) .

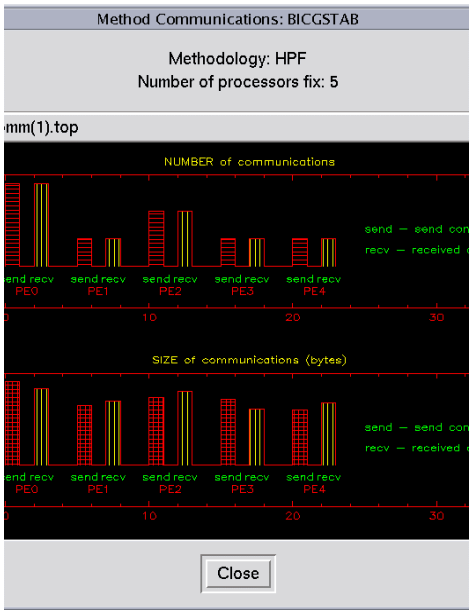

Fig. A1 Number of communications sent and received

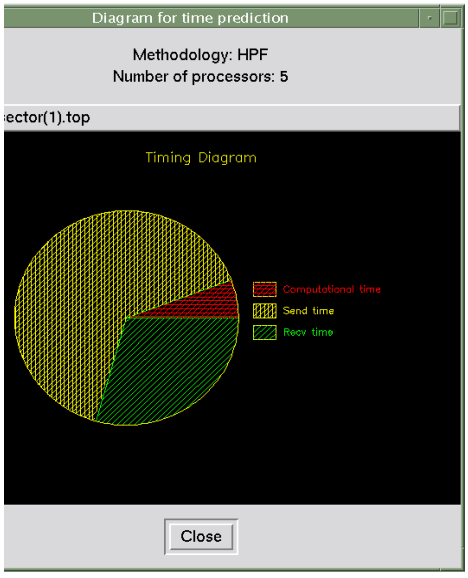

Fig. A2 Time spent in computation and communication

Up to now, the tool is static, and the main user is the applications developer.

6. When the application is finally being sent to be executed on the Grid, the monitoring tools provide information about hardware capabilities, and this information can be used by the performance prediction tool to give estimations about how the kernels will work on these specific and real environment. This information can be shown to the user with the visualization tool. In addition, in some applications, eventually it could be used to guide the parallel program in deciding, for example, the size of the blocks. Note that the predictions are obtained very fast because it only requires to compute the functions that define the analytical model.
7. During the execution of the application, or after that, with the monitoring information as basis, the user could obtain information about the influence of the hardware parameters on the execution of the kernels, and analyse the influence of changing some of these parameters. Note that this provides the user with a much rich analysis of the application performance because, he or she not only can see what really happens during the execution, but he or she can see what would happen when some specific hardware characteristics change. This information can be used in following executions.

Note that the performance prediction tool process is independent of the applications execution.

Use Case: Monitoring of Network Impact on Grid Performance

It is assumed that the application comes from Task 1.3, a distributed Neural Network. It runs on 1 *master* + *n slaves*, where part of these n slaves can be outside the cluster. The master sends a collective MPI and has to wait for answer of all nodes.

So the last node is either the slowest one or the one where the transfer takes most time.

If a typical job of 1000 NN epochs on a 10-15-15-1 NN architecture takes about 600 minutes of training on a 1M data sample when running on 1 PIII 1 GHz, it needs 10 minutes on 60 machines of a local cluster. Thus, each epoch training takes about 0.6 second/epoch on each machine. Now, if some of these machines were distributed across the Grid, the latency + bandwidth effect could be significant compared to these 0.6 seconds. In fact, it turned out to be very significant during first tests with one node joined via ADSL.

The first monitoring exercise is to run these distributed jobs on 10 machines in one cluster and another one in another cluster, and to check the effect. Compare and estimate having in mind the latency time obtained from other tools (like ping or trace?). Consider then 10 machines in each cluster. Derive optimal strategies for the use of the resources in interactive distributed way. Integrate CPU power and network power to estimate costs.

5. INDEX