The CrossGrid Performance Analysis Tool for Interactive Grid Applications*

Marian Bubak[1,2], Włodzimierz Funika[1], and Roland Wismüller[3,4]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM, Inst. f. Informatik, Technische Universität München, D-80290 München
[4] Institute for Software Sciences, University of Vienna, A-1090 Vienna, Austria

Abstract. The EU CrossGrid project aims at widening the use of Grid technology for interactive applications. In order to meet the efficiency constraints of these applications, a Grid applications performance analysis tool is developed that not only allows the standard measurements, but also supports application-specific metrics and other high-level measurements. The article outlines the projected functionality of this tool, as well as design considerations and a use case.

1 Introduction

Grid computing has the potential of providing a cost- and resource-efficient way of solving data and compute intensive problems. In the EU DataGrid project, for instance, Grid technologies are used to enable a world wide, efficient access to the huge amounts of data that will be produced by future High Energy Physics (HEP) experiments. In March 2002, a new EU project – CrossGrid[4] – was launched to extend existing Grid technologies towards a new direction: *interactive applications*. Besides providing the necessary Grid services and testbeds, four demonstrators for interactive Grid applications are being developed in CrossGrid [3]: simulation of vascular blood flow, flooding crisis team support tools, distributed data analysis in HEP, and simulation of air pollution combined with weather forecast.

Efficiently using the Grid as an environment for such large-scale interactive applications requires programmers to exactly know the performance behavior of their applications. However, even with a good knowledge of an application's code, its detailed run-time behavior in a Grid environment is often hard to figure out, not the least because of the dynamic nature of this infrastructure. In order to support this task, the CrossGrid project develops a tool, named G-PM, which not only allows to measure just the standard performance metrics, but allows to determine higher-level performance properties and application specific metrics. E.g., for an interactive client/server application it is important to measure the response time between a user interaction and the proper reaction of the program. Even more important is the breakdown of this time, e.g. how much of it is

* Partly funded by the European Commission, project IST-2001-32243, CrossGrid

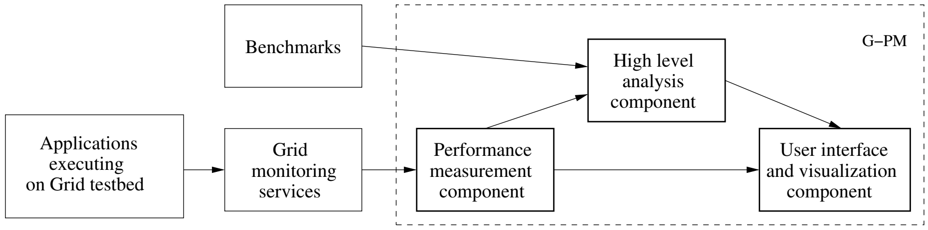

Fig. 1. Component structure of G-PM

due to the client's computation, due to network delays, or due to the server's computation.

For providing this kind of information, the G-PM tool uses three sources of data: performance measurement data related to the running application, measured performance data on the execution environment, and results of micro-benchmarks, providing reference values for the performance of the execution environment. This acquired data is heavily processed and transformed within G-PM in order to take into account the structural and architectural details of the Grid. In this way, information on performance properties and/or bottlenecks in applications can be provided without a need for the user to know the Grid's detailed architecture.

The paper is organized as follows: Section 2 outlines the projected functionality of the tool, Section 3 presents some details on its design, while Section 4 contains a simple use case. We conclude with a discussion of related work and the current status of our own work.

2 Projected Functionality of G-PM

The performance analysis tool G-PM is structured into three major components (c.f. Fig. 1) that enable a user to optimize the performance of a Grid application:

1. performance measurement component (PMC),
2. component for high level analysis (HLAC),
3. user interface and visualization component (UIVC).

The PMC provides 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.

The HLAC supports an application and problem specific analysis of the performance data. On the one hand, it allows to measure application specific performance metrics. On the other hand, it provides a specification language which allows to combine and correlate different performance measurements in order to derive higher-level metrics, like e.g. load imbalance. The objective of this approach is to provide more meaningful data to application developers.

Finally, the UIVC allows the user to specify performance measurements and visualizes the performance data produced by PMC and/or HLAC.

In the following subsections, we provide more details on these components.

2.1 Performance Measurement Component

The PMC supports two types of measurement requests: *Application related requests* ask for information on the whole or a part of a Grid application. They can be issued either before the application is started, or while it is already running. *Infrastructure related requests* ask for information on the whole or a part of the Grid infrastructure.

The PMC can acquire the following types of application related performance data:

1. *data transfer*: amount of data received from / sent to other processes / hosts, amount of data read from / written to files,
2. *resources utilization*: CPU usage (computation time), memory accesses (number, frequency, time), disk accesses, cache operations,
3. *delays*: due to communication, synchronization, and I/O operations.

The Grid infrastructure related performance data includes:

1. *availability of resources*,
2. *dynamic resource information*: CPU load and link load,
3. *static resource information*: CPU architecture, CPU performance, memory hierarchy features, memory performance, and secondary storage features of nodes, bandwidth and latency of links.

Measurements can be defined with different granularities in space and time. In space, it should be possible to apportion the measurements with respect to sites and hosts, and – for application related requests – also with respect to processes and functions inside a process. In time, three different granularities are provided:

1. single performance value comprising a complete execution,
2. summary values for a selectable period of time,
3. progression of a performance metrics over time, with selectable resolution.

The PMC can be used directly via the UIVC. For example, let us assume that the user requests the measurement of the total volume of communication between two processes. To define this measurement, the user selects the "*received data volume*" and the "*sent data volume*" metrics, the process identifiers (resolution in space), and the interval of the measurement (resolution in time). In a visualization window, G-PM then displays the requested performance information in one of several possible forms, e.g. as two bar-graphs, one for each direction of message transfer, or e.g. as a communication matrix where a pair (column, row) represents a *sending process – receiving process* pair with corresponding communication volume values.

2.2 High Level Analysis Component

The HLAC provides two major functions to the user:

1. 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.
2. 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.

To achieve the latter, the application programmer needs to mark relevant places in the code, like e.g. the beginning and the end of a computation phase. This is done by inserting a special function call (*probe*) into the application's source code. Any number of scalar parameters may be passed to a probe, which allows to provide application-specific data relevant for performance analysis. Since performance is so crucial for interactive Grid applications, we expect that programmers are willing to insert this small and limited amount of instrumentation into their applications in order to get more adequate performance information. Once the probes are available in an application, G-PM allows to measure:

– any application-related or infrastructure-related metrics mentioned in subsection 2.1 in an execution phase defined by any two probe executions,
– application-specific metrics defined by any parameter passed to any probe,
– any metrics that can be (recursively) derived from already existing ones.

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

The specification language ASL, developed in the IST Working Group APART [5], is used to define application-specific metrics, i.e. to specify, how these metrics are computed from the data delivered by the application's probes and the generic performance measurement data provided by the PMC. Thus, using the same probes, different metrics can be measured, e.g. minimum, maximum and mean time for an iteration, amount of communication per iteration, etc. As mentioned, probes may also pass application-specific data to the HLAC, e.g. a residuum value that allows to compute a solver's convergence rate.

For each application, the ASL specifications of all application-specific performance metrics can be stored in a configuration file. The user of G-PM can load the proper file at run-time to customize the tool for his 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. Note 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. The overhead of an inactive probe, which is a call to a function containing a

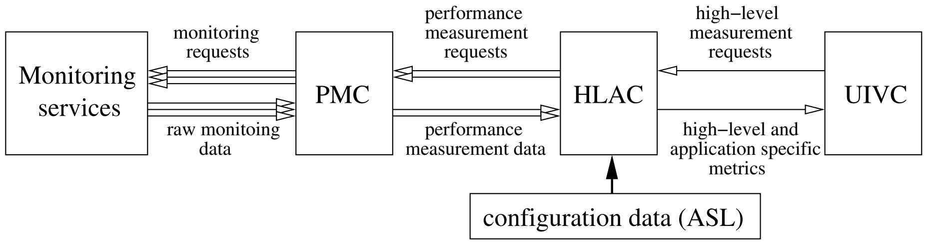

Fig. 2. Request and data transformation in G-PM

non-executed conditional block, is negligible in all practical cases. For a probe receiving a `double` value as an argument, we measured $79ns$ on a 300 MHz Sun UltraSPARC-II.

In addition to application-specific ASL files, a generic ASL file bundled with G-PM contains the definitions of higher-level metrics independent of a specific application. Examples of such metrics are load imbalance and relative CPU / network performance. The last two metrics present an application's performance as a percentage of the execution platform's "optimal" performance, which is measured via micro-benchmarks.

2.3 Graphical User Interface Component

The G-PM tool provides a graphical user interface based on the PATOP performance analysis tool [2]. It consists of four different classes of windows:

1. The main window offers a menu to enter user commands and displays a list of all active performance measurements, as well as a list of all open visualization windows. These lists are used to manage (i.e. modify or delete) both measurements and visualization windows.
2. A measurement definition window to specify the performance measurements to be performed (type of metrics, objects involved, time interval, location of the measurement).
3. A visualization definition window. It allows to specify the way in which a selected set of performance measurements should be visualized. These different ways include bar graphs as well as graphs plotted against real time or against the number of executions of specified probes.
4. An arbitrary number of visualization windows.

3 Design Considerations

Besides providing the user interface for defining and visualizing performance measurements, the main task of G-PM is to transform the user's (high-level) measurement requests into low-level requests that can be handled by Grid monitoring services. Vice versa, G-PM has to compute performance metrics from the raw measurement data returned by the monitoring services (c.f. Fig 2).

In the following, we will focus on these transformations within the PMC and the HLAC, as well as the requirements imposed on the Grid monitoring services.

3.1 Grid Monitoring Services

G-PM is based on the premise that there exist Grid monitoring services that provide on-line, low-level data on the Grid infrastructure and running applications. Typical services will e.g. return the current CPU load of a node or monitor the occurrence of events in an application process. The PMC uses these services to implement performance measurements. For example, consider the measurement of the total volume of communication between two processes mentioned in subsection 2.1. The PMC asks the monitoring services to capture all "*message sent*" events (i.e. invocations of the communication library's send function) between these processes, and to add the message's size to a previously defined counter.

The actual interface between the PMC and the monitoring services is based on the OMIS 2.0 specification [10] and is adapted and further developed within the CrossGrid project. For a more detailed discussion, see [1]. The interface accepts monitoring requests, which may belong to one of the following types:

1. *information requests* just return some information on the infrastructure or application in the reply,
2. *manipulation requests* result in a manipulating action,
3. *event requests* capture a specified event and trigger a list of actions (either for returning information or for manipulation).

The G-PM tool mainly uses information requests and event requests, e.g.:

- The information request *return CPU load on nodes n_1, n_2, n_3* immediately returns a list of three numbers containing the load values.
- To perform measurements related to a function A within an application, event requests with events: "*function A has started*" and "*function A has ended*" are used. Whenever these events occur, a list of actions is executed by the monitoring services.

These actions then implement the actual performance measurements, e.g. to obtain the execution time or the volume of data transferred. Among others, the following actions are available for this purpose:

- *increment a named counter* or *integrating counter* by a specified amount (which may be defined by a parameter describing the current event),
- *store a record* (which may include parameters of the current event) in an event trace,
- *send event information* immediately to the PMC,
- *enable* or *disable* other event requests (which results in starting or stopping other performance measurements).

To support the HLAC, the monitoring services offer two additional features: First, they allow to monitor probes in the application's code. This is similar to the "*function A has started*" event mentioned above. Second, they allow to access the results of micro-benchmarks: given a host and a host-related metrics, return the result for the specified host; given two hosts and a network-related metrics, return the result for the network connection between these hosts.

3.2 Performance Measurement Component

The PMC supplies performance measurement data on applications and the Grid infrastructure on request from the UIVC or the HLAC. Such a request may include the *metrics to be measured* (time, data volume), an *object(s) specification* (site, host, process), a *time interval* (complete execution, time/program phases), a *location* (program, module, function), and *constraints* (metrics limit values, communication partners).

The basic task of the PMC is to implement these requests by means of the available monitoring services. For example, when a measurement "*execution time of function A in processes p_1, p_2, p_3*" is requested, the PMC ensures that:

- the measurement is transformed into two event requests ("*function A has started / ended*") for each process, with appropriate actions, which use an integrating counter to compute the difference of the end time and the start time of function A in each of the processes,
- these requests are submitted to the monitoring services,
- the data returned by the monitoring services is processed and returned in the required form to the requesting component (UIVC or HLAC).

3.3 High Level Analysis Component

While the PMC transforms raw monitoring data to performance measurement data, the HLAC further transforms the latter to higher-level and/or application specific properties (see Fig. 2). The main differences between these two components are:

- While the transformations in the PMC are fixed, the HLAC is fully configurable with respect to the performance measurement data it gets as input, the high-level properties it should provide to the user, and the way how these properties are computed from the different types of performance measurement data.
- Since the 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 the PMC, the HLAC must provide transformations in two directions: When the user requests some high-level property to be measured, the HLAC requests the PMC to measure all the required performance measurement data. Vice versa, when this data arrives, the high-level properties are computed according to the rules specified in the HLAC's configuration data.

The configuration data is based on the APART Specification Language ASL, which allows to specify both a data model for the incoming performance measurement data, as well as computation rules for the performance properties. Thus, the HLAC basically implements an interpreter for the ASL.

4 A Use Case

The medical application developed in the CrossGrid project will make use of a special library, called Grid Visualization Kernel (GVK) [8] for the communication between its simulation and visualization components. Assume that the programmer wants to analyze the performance of the visualization, in particular the impact of GVK. In the following, *server* denotes a host executing parts of the simulation, while *client* is the host visualizing the results.

1. In order to acquire GVK specific performance data, the programmer (manually) inserts probes at three places in the source code of GVK:
   (a) immediately after (on the server) GVK is requested to visualize a frame,
   (b) immediately after (on the server) the data is sent to the client,
   (c) immediately before the data is passed to the graphics engine on the client. The size of the data is passed to the probe as an argument.
   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:
   - Generated image frames per second = 1 / (Time between successive invocations of probe a).
   - Compression factor = (data size passed to probe c) / (communication volume from server to client between executions of probe a and b). The communication volume between hosts is a standard metrics of the PMC.
   - GVK processing time for a frame = time interval between the execution of probe a and the execution of probe b.

   Note that the same probes are used to define different metrics. Since these probes and metrics are of more general interest, they may already have been inserted / defined by the developer of GVK. In this case, the GVK library will already come with an accompanying ASL specification file for G-PM.
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. He 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 can now select a measurement for "*generated image frames per second*" in the measurement definition window. In addition, he can select a suitable visualization style (e.g. as a function over time).
7. If the programmer now wants to measure the communication volume within the simulation component for each generated image frame, he can define the new metrics on-line, without a need to change or even restart the application, since the probe required for this metrics is already available.

5 Related Work

There already exists a couple of performance tools for the Grid. In contrast to G-PM, most tools that support the monitoring of applications are based on an off-line analysis of event traces. A prominent example is NetLogger [7]. Besides the problem of scalability, trace-based off-line tools are not useful for the analysis of interactive applications, since they cannot show the performance metrics concurrently with (i.e.: in correlation to) the user's interaction with the application. Currently, on-line tools are available only for infrastructure monitoring, since their main use is for resource management, which requires on-line data. Examples are the Heartbeat Monitor [6] and the Network Weather Service [14] that forecasts network and node performance.

Some performance tools implement ideas closely related to the ones presented in this paper. User-defined instrumentation, in particular timers that allow to measure the CPU time of code sections, is used in the TAU performance analysis environment [12]. Autopilot [13], a distributed performance measurement and resource control system, has a concept called *sensors* that very closely corresponds to our probes. In both of these systems, however, the metrics derived from this instrumentation are fixed by the tool's programmer and cannot be configured by the tool's user. A metrics definition language (called MDL) is used in the Paradyn tool [11]. However, the MDL [9] is hidden inside the tool and is not intended for the definition of application-specific metrics. Paradyn is one of the few performance tools that strictly follow the on-line approach, where even the instrumentation is inserted and/or activated during run-time. Another such tool is PATOP [2], which forms a basis of G-PM.

The main contribution of G-PM is its unique combination of Grid awareness, on-line measurement, and automatic instrumentation for standard metrics on the one hand with a support for manual instrumentation and user-definable metrics on the other.

6 Status and Conclusions

We have recently finished the requirements specification and the design phase of the G-PM tool. The first prototype of G-PM is currently being implemented and will be available in February 2003. This prototype will include basic performance measurements, and some examples of higher-level metrics, but will not yet be fully configurable via the ASL. This feature will be added in the next prototype available in 2004. The final version of G-PM will be ready by the end of 2004. Like all other system software developed in CrossGrid, G-PM will then be freely available via a public software license.

We expect that the highly pre-processed, application-specific performance information provided by G-PM will significantly simplify program optimization and tuning, making the work of programmers and scientists more efficient, and thus eliminating possible barriers to work on the Grid.

References

1. B. Baliś, M. Bubak, W. Funika, T. Szepieniec, and R. Wismüller. An Infrastructure for Grid Application Monitoring. In *Proc. 9th European PVM/MPI Users' Group Meeting*, Linz, Austria, Sept. 2002. LNCS, Springer-Verlag.
2. M. Bubak, W. Funika, B. Baliś, and R. Wismüller. On-Line OCM-Based Tool Support for Parallel Applications. In Y. C. Kwong, editor, *Annual Review of Scalable Computing*, volume 3, chapter 2, pages 32–62. World Scientific Publishing Co. and Singapore University Press, 2001.
3. M. Bubak, M. Malawski, and K. Zając. Towards the CrossGrid Architecture. In *Proc. 9th European PVM/MPI Users' Group Meeting*, Linz, Austria, Sept. 2002. LNCS, Springer-Verlag.
4. CrossGrid Project. `http://www.eu-crossgrid.org`.
5. T. Fahringer, M. Gerndt, G. Riley, and J. L. Träff. Knowledge Specification for Automatic Performance Analysis – APART Technical Report. Technical report, ESPRIT IV Working Group on Automatic Performance Analysis, Nov. 1999. `http://www.par.univie.ac.at/~tf/apart/wp2/report-wp2.ps.gz`.
6. The Globus Heartbeat Monitor Specification. `http://www-fp.globus.org/hbm/heartbeat_spec.html`.
7. D. Gunter et al. NetLogger: A Toolkit for Distributed System Performance Analysis. In *Proc. IEEE Mascots 2000 Conference*, Aug. 2000. Lawrence Berkeley Natl. Lab., Report LBNL-46269.
8. P. Heinzlreiter, D. Kranzlmüller, G. Kurka, and J. Volkert. Interactive Visualization in Large-Scale, Distributed Computing Infrastructures with GVK. In *SCI 2002, 6th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida, July 2002.
9. J. R. Hollingsworth et al. MDL: A Language and Compiler for Dynamic Program Instrumentation. In *Proc. International Conference on Parallel Architectures and Compilation Techniques*, San Francisco, CA, USA, Nov. 1997. `ftp://grilled.cs.wisc.edu/technical_papers/mdl.ps.gz`.
10. T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. *OMIS — On-line Monitoring Interface Specification (Version 2.0)*, volume 9 of *LRR-TUM Research Report Series*. Shaker-Verlag, Aachen, Germany, 1997. ISBN 3-8265-3035-7. `http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz`.
11. B. P. Miller et al. The Paradyn Parallel Performance Measurement Tools. *IEEE Computer*, 28(11):37–46, Nov. 1995. `http://www.cs.wisc.edu/paradyn/papers/overview.ps.gz`.
12. S. Shende, A. D. Malony, J. Cuny, K. Lindlan, P. Beckman, and S. Karmesin. Portable Profiling and Tracing for Parallel Scientific Applications using C++. In *Proc. ACM SIGMETRICS Symp. on Parallel and Distributed Tools (SPDT '98)*, pages 34–145, Welches, Oregon, Aug. 1998.
13. J. Vetter and D. Reed. Real-time Monitoring, Adaptive Control and Interactive Steering of Computational Grids. *The International Journal of High Performance Computing Applications*, 14:357–366, 2000.
14. R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15:757–768, 1999.