From Cluster Monitoring to Grid Monitoring Based on GRM*

Zoltán Balaton, Péter Kacsuk, Norbert Podhorszki, and Ferenc Vajda

MTA SZTAKI
H-1518 Budapest, P.O.Box 63. Hungary
{balaton,kacsuk,pnorbert,vajda}@sztaki.hu

Abstract. GRM was originally designed and implemented as part of the *P-GRADE* graphical parallel program development environment running on supercomputers and clusters. In the framework of the biggest European Grid project, the DataGrid we investigated the possibility of transforming GRM to a grid application monitoring infrastructure. This paper presents the architectural redesign of GRM to become a standalone grid monitoring tool.

Keywords: grid monitoring, message passing, grid monitoring architecture.

1 Introduction

With the emergence of computational grids it became very important to support grid-based applications with dynamic performance monitoring infrastructure. GRM 1 and PROVE were and still are available as parts of the *P-GRADE* graphical parallel program development environment 2. *GRM* is a "semi-on-line monitor" that collects information about an application running in a distributed heterogeneous system and delivers the collected information to the *PROVE* visualisation tool. The information can be either event trace data or statistical information of the application behaviour. "Semi-on-line" monitoring means, that any time during execution all available trace data can be requested by the user, and the monitor is able to gather them in a reasonable amount of time. PROVE has been developed for performance visualisation of Tape/PVM 3 trace files. It supports the presentation of detailed event traces as well as statistical information of applications. It can work both off-line and semi-on-line, and it can be used for observation of long-running distributed applications.

P-GRADE is a graphical programming environment integrating several tools to support the whole life cycle of building parallel applications. It provides an easy-to-use, integrated set of programming tools for development of general message passing applications to be run in heterogeneous computing environments. Its main benefits are the visual interface to define all parallel activities in the application, the syntax independent graphical definition of message passing instructions, full support of compilation and execution in a heterogeneous environment and the integrated use of the debugger and the performance visualisation tool. For detailed overview of the

* This work was supported by a grant of the Hungarian Scientific Research Fund (OTKA) no. T032226

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 874-881, 2001.

tools in P-GRADE, see 4 while further information, tutorial and papers about P-GRADE can be found at 2.

In this paper we present four possible architectural designs and related issues of the GRM monitor for semi-on-line general application monitoring in a grid environment. In the next Section, the original design goals and the structure of GRM are shortly presented. In Section 3 the problems with GRM in a grid environment is discussed and four architectures designed for grid monitoring are presented.

2 Original Design Goals and Structure of GRM

The monitoring in GRM is *event-driven*, both *trace collection* and *counting* are supported. The measurement method is *software tracing* and the instrumentation method is *direct source code instrumentation*. For a classification of monitoring techniques, see 5. Direct source code instrumentation is the easiest way of instrumentation. Since P-GRADE controls the whole cycle of application building, and source code instrumentation is supported by graphics, the chosen one was a natural option. The precompiler inserts instrumentation function calls into the source code and the application process generates the trace events.

The main goals in the original design of GRM have been strongly related to the P-GRADE environment. The monitor and the visualisation tool are parts of an *integrated* development environment and they support monitoring and visualisation of P-GRADE applications at source level. The monitor is portable among different UNIX operating systems (Irix, Solaris, Linux, Tru64 UNIX, etc.) which is achieved by using only standard UNIX programming solutions in the implementation. GRM is a semi-on-line monitor, that is, the user can let GRM to collect the actual trace data or statistical information about the application any time during the execution. Semi-on-line monitoring is very useful for the evaluation of long-running programs and for supporting debugging with execution visualisation. Both trace collection and statistics are supported by the same monitor and by the same instrumentation of the application. Trace collection is needed to pass data to PROVE for execution visualisation. Statistics mode has less intrusion to the execution by generating fixed amount of data and it supports initial evaluation of long-running applications.

For trace storage *shared-memory segments* have been used on each host, since semi-on-line monitoring requires direct access to all trace data any time during the execution. The monitor can read the shared buffer independently from the application process, when the user asks to collect trace data. Moreover, if a process aborts its trace data can be saved and analysed to the point of failure.

GRM consists of the following three main components (see its structure in Fig. 1):

Client Library
The application is instrumented with functions of the client. Both trace events and statistics can be generated by the same instrumentation. The trace event types support the monitoring and visualisation of P-GRADE programs. An instrumented application process does not communicate outside of the host it is running on. It places trace event records or increments counters in a shared memory buffer provided by the Local Monitor.

Local Monitor

A Local Monitor (LM) is running on each host where application processes are executed. It is responsible for handling trace events from processes on the same host. It creates a shared memory buffer where processes place event records directly. Thus even if the process terminates abnormally, all trace events are available for the user up to the point of failure. In statistics collection mode, the shared memory buffer is used to store the counters and LM is responsible for generating the final statistics data in an appropriate form.

Main Monitor

The Main Monitor (MM) is co-ordinating the work of the Local Monitors. It collects trace data from them when the user asks or a trace buffer on a local host becomes full. The trace is written into a text file in Tape/PVM format (see 3), which is a record based format for trace events in ASCII representation. The Main Monitor also performs clock synchronisation among the hosts.

PROVE collects trace data for execution visualisation. PROVE communicates with MM and asks for trace collection periodically. It can work remotely from the Main Monitor process. With the ability of reading new volumes of data and removing any portion of data from its memory, PROVE can observe applications for arbitrary long time.

The integration of GRM into a development environment made it possible to put several functionalities of a stand-alone monitoring tool into other components of P-GRADE.

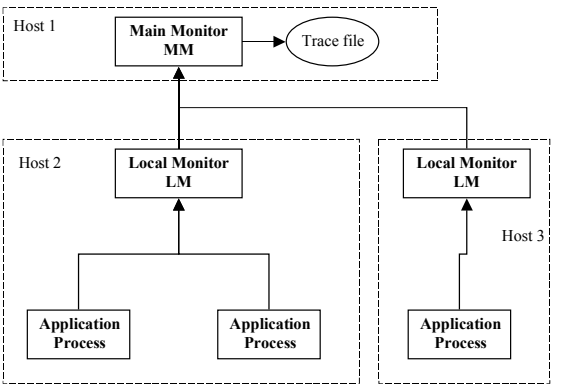

Fig. 1. Structure of GRM

Instrumentation is done in the *GRED* 4 graphical editor of P-GRADE. Trace events of different processes are not sorted into time order since the pre-processing phase in PROVE does not need a globally sorted trace file. The monitor is started and stopped by GRED and GRM does no bookkeeping of processes. Local Monitors are started on the hosts defined by the environment.

3 Required Modifications of GRM in a Grid Environment

Monitoring of applications in a grid environment brings new requirements for a monitoring tool. The monitor should be scalable to a large number of resources and events. The start-up of the monitor and collection of data from remote sites are more complex and difficult than in a single cluster. Measurements must have accurate cross-site timestamps. Moreover, original design goals of GRM must be reviewed. In

particular, two goals must be changed. First, general application monitoring should be supported, not only P-GRADE-based. This also requires user defined event data types. Second, GRM and PROVE should be standalone monitoring and visualisation tools, that is, not part of an integrated development environment. Other design goals (see Section 2) are unchanged. Portability remains very important, since a grid consists of heterogeneous resources. Semi-on-line monitoring must be supported, because a grid is a changing environment, where off-line monitoring does not help. Both statistics and event trace collection is needed to monitor large and long running applications that are typical in a grid. Getting trace data to the point of failure is very important, since errors are more frequent in a grid.

The start-up of the monitoring system becomes difficult because Local Monitors cannot be started on a host explicitly, since it is the competence of the local job-manager on a grid resource to decide where jobs will run. This local policy cannot be influenced. Because of this, the Main Monitor cannot start the Local Monitors as in GRM. Instead, it should be prepared to accept connections from them.

The clock synchronisation in GRM is done through the sockets connecting Local Monitors with the Main Monitor, see details in 1. MM performs the synchronisation with each LM. This technique works well on clusters of workstations connected by a LAN. In a grid environment the resources (e.g. clusters) are usually connected by WAN links that have higher latency than the LAN used inside the resource. GRM determines the clock offsets of each LM (running on a host at the remote grid resource) relative to the host of the MM but the accuracy of this measurement is limited by the latency of the WAN link. Because of this, the error of the clock-offset measurement can be comparable to or bigger than the time intervals between events generated at the remote resource (e.g. the start and end of a communication on the LAN). Since there are several tools (e.g. NTP) that can be used to synchronise clocks, this problem can be solved independently from monitoring. For this reason, GRM does not support clock synchronisation in grid environments, instead it assumes that the clocks are already synchronised.

There are a number of performance measurement and resource control systems used in grid (e.g. Network Weather Service, Autopilot, NetLogger, etc.). For their comparison see 6.

In the following subsections four different architectures of GRM are examined that can be used in a grid environment for application monitoring.

3.1 Simple Architecture

In the simplest monitoring architecture (see Fig. 2) Local Monitors are omitted and application processes send trace events immediately to the Main Monitor which is waiting for trace data in a remote site. The application process can store trace records temporarily in its local memory. However, with this architecture we loose the following advantages of GRM.

Local buffering of trace data in GRM is done independently from the monitored application, so the Main Monitor can collect it any time. In the simple architecture data should be sent immediately to the remote collector process or can be buffered locally by the application process but in this case the visualisation tool must wait until the block of trace data have arrived. In GRM the visualisation tool can request for trace data any time and the monitor collects data from the Local Monitor processes.

With local buffering in a shared memory segment, application processes and sensors can give trace events to the monitoring tool quickly so they can have very low intrusiveness. Thus, the application can run almost as efficiently as without instrumentation. In the simple architecture, the application process is blocked while it sends trace events to the collecting process over wide-area network.

The simple architecture is not as scalable as GRM using Local Monitors. Moreover, with trace buffering in the local memory of the process we loose trace events when a process aborts. Thus, the visualisation tool cannot show all events until the point of failure.

The start-up of the monitor is simple. Application processes should be started with the Main Monitor address as parameter and should connect to the Main Monitor. Since Local Monitors are excluded, both the Main Monitor code and the instrumentation library should be modified for the new functionality. However, one problem remains. The use of firewalls in the different computing sites can prohibit the direct connection of the application processes to the Main Monitor. Either firewalls should enable this connections (which cannot be expected from the administrators) or some proxy-like solutions should be introduced for indirect connections.

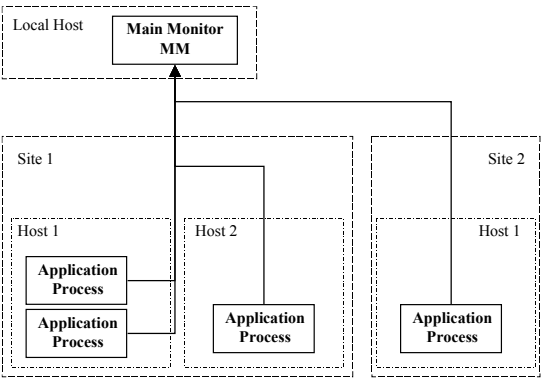

Fig. 2. Simple monitoring architecture

3.2 Local Monitors as Site Monitors

If we want to use Local Monitors to keep the benefits of GRM, Local Monitors can be used in two different ways. The first one is to use one Local Monitor for each computing site. This option is discussed in this subsection. The second one is to use Local Monitors in each machine, using them the same way as in the original GRM. This solution is discussed in the next subsection.

In this architecture (see Fig. 3) a Local Monitor collects all trace data in a computing site (cluster or supercomputer) and sends data to the remote Main Monitor. Data is sent over the wide-area network only when the Main Monitor requests for it. With this solution, the remote connection of the application processes can be avoided. A process should send trace events to the Local Monitor, which is on the same site as the process. This architecture is more scalable and less intrusive than the simple architecture. However, the start-up of the monitor becomes much more complex. The Local Monitor should be started remotely in a site where an application process is to be started. The Main Monitor and the user have no exact information at launch time about where the application will be executed. Moreover, the Main Monitor cannot

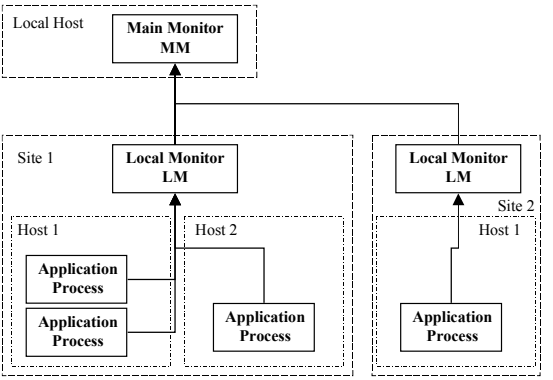

Fig. 3. Local Monitors at each site

start a process in a remote site freely. The grid management software itself should be prepared to launch a Local Monitor on a site where an application process is started.

The problem of the firewalls can be solved if the Local Monitor is started on the server machine of the site, which has connections to the external world. Administrators should enable only one connection between the Main Monitor and the server machine instead of enabling connections from any machines in their domain.

The code of the Local Monitor and the instrumentation library should be modified because shared memory cannot be used between two machines. Data exchange between the Local Monitor and the application processes should be performed now through sockets instead of shared memory.

3.3 Local Monitors with Their Original Functionality

The Local Monitor, Main Monitor and the instrumentation library of GRM can be used without modification if Local Monitors are started on all single machines where processes of the application are started. The structure and the logical work of this solution (see Fig 4) is the same as the original GRM (Fig 1). The only difference is that a Local Monitor and the Main Monitor should communicate through a WAN and not in a LAN. Thus, we can use shared memory buffering of trace data, less intrusive monitoring of the processes and more efficient transfer of data.

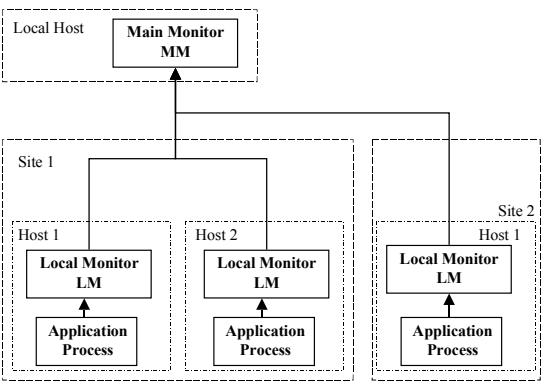

Fig. 4. Local Monitors on each machine

The problem of this architecture is the start-up of the Local Monitor. The executable of the Local Monitor should be transferred to the grid resource where the application is run. The easiest way to do this is to link the LM executable to the process as a library. This way we have a single executable which contains the code both of the application and the LM, and can be started the same way as the application. This solution is independent of any specific grid implementation but requires that the application can be relinked.

However, the application has to be instrumented for monitoring with GRM, so this can be assumed.

The start-up of this architecture in a grid environment goes in the following way. The user launches the MM which gives back its port number. The hostname and port number pair identifies this MM. The user starts the application that also contains the LM linked in, giving it the MM identifier as a parameter. An application process calls an instrumentation function at start that tries to connect to the Local Monitor through a FIFO that contains the MM identifier in its name. If the process detects that there is no LM listening on this FIFO yet, it forks, and becomes the Local Monitor. Its child continues as the application process. The LM creates the shared buffer and the FIFO through which processes can now connect to it. When an LM is created, the application process connects to it and the LM connects to the MM. After successfully connecting to the Main Monitor, the LM notifies the process. From this point, the process can start generating trace events. The problem of firewalls is the same as at the simple architecture. Here Local Monitors in the machines should be able to connect to the remote Main Monitor.

This architecture is examined in detail in 7.

3.4 Local Monitors and Site Monitors

A fourth possible architecture (see Fig. 5) is the combination of the previous two ones. Local Monitors are used as in the original GRM and Site Monitors are introduced as intermediate monitoring processes between the Main Monitor and the Local Monitors. Site Monitors are used to forward data from Local Monitors to the Main Monitor avoiding the problems of the firewall. With this solution, Local Monitors are exactly the same as the Local Monitors of the original GRM, communicating with a higher-level monitoring process in the local area network.

This architecture is the most scalable for a grid. Site Monitors can store trace data of the site. Application processes, Local Monitors and executing machines are freed from the overhead of trace storage and long wide-area network communications while the load of the Main Monitor can be controlled more easily. The firewall problem is also reduced to the level of the second architecture. Only the remote connection between the Site Monitor and the Main Monitor should be enabled. However, the start-up problem is raised to the level of the second architecture. The start-up of the Site Monitor should be supported by the grid management software.

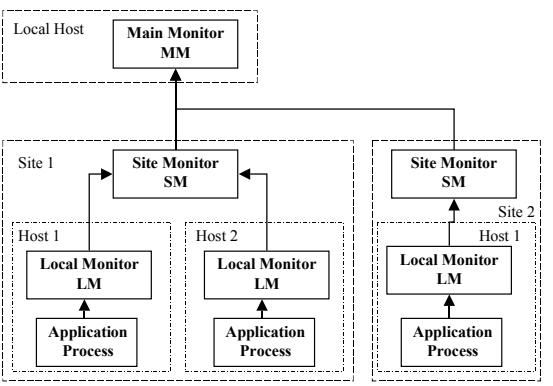

Fig. 5. Combination of Local and Site Monitors

4 Conclusions

A grid environment brings new requirements for monitoring. The GRM monitoring tool of the P-GRADE graphical parallel programming environment is a good candidate to be a standalone grid-application monitoring tool. We examined its features and monitoring mechanisms and compared them to the requirements of a grid. With architectural redesign and code modifications GRM can collect traces from large distributed applications in a grid that can be analysed. In the architectural design of GRM, scalability and problematic start-up issues in grid were considered. Since the third structure keeps the original structure of GRM and the functionality of the elements we have chosen this architecture to implement the first version of the grid application monitor.

References

1. N. Podhorszki, P. Kacsuk: "Design and Implementation of a Distributed Monitor for Semi-on-line Monitoring of VisualMP Applications", Proceedings. DAPSYS'2000 Distributed and Parallel Systems, From Instruction Parallelism to Cluster Computing, Balatonfüred, Hungary, pp. 23-32, 2000.
2. P-GRADE Graphical Parallel Program Development Environment: *http://www.lpds.sztaki.hu/projects/p-grade*
3. É. Maillet: "Tape/PVM: An Efficient Performance Monitor for PVM Applications. User's guide", LMC-IMAG, Grenoble, France, 1995. Available at *http://www-apache.imag.fr/software/tape/manual-tape.ps.gz*
4. P. Kacsuk, G. Dózsa, T. Fadgyas, and R. Lovas: "The GRED graphical editor for the GRADE parallel programming environment", FGCS journal, Special Issue on High-Performance Computing and Networking, Vol. 15 (1999), No. 3, April 1999, pp. 443-452.
5. J. Chassin de Kergommeaux, É. Maillet and J-M. Vincent: "Monitoring Parallel Programs for Performance Tuning in Cluster Environments", In "Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments" book, P.Kacsuk and J.C.Cunha eds, Chapter 6., Nova Science, 2000.
6. Z. Balaton, P. Kacsuk, N. Podhorszki and F. Vajda: "Comparison of Representative Grid Monitoring Tools", Reports of the Laboratory of Parallel and Distributed Systems (SZTAKI), LPDS-2/2000, 2000. Available at: ftp://ftp.lpds.sztaki.hu/pub/lpds/publications/reports/lpds-2-2000.pdf
7. Z. Balaton, P. Kacsuk, N. Podhorszki: "Application Monitoring in the Grid with GRM and PROVE ", Proceedings of the ICCS'2001 (Intl. Conf. on Computational Science), San Francisco, CA, May 28-31, 2001.