Integrating of the OCM-G Monitoring System into the GRID superscalar

Włodzimierz Funika¹, Marcin Smętek¹, Marian Bubak¹,²
Inst. Comp. Sci., AGH, al. Mickiewicza 30, Kraków, Poland
ACC CYFRONET, Nawojki 11, Kraków, Poland
{funika,smetek,bubak}@uci.agh.edu.pl
phone: (+48 12) 617 44 66, fax: (+48 12) 633 80 54

1 Introduction

Performance monitoring facilities are an important part of any distributed system and are essential in Grid environments because performance and monitoring information is required not only by the user to get information about

Abstract

In this paper we describe a Grid-enabled system for monitoring GRID superscalar-compliant applications to enable on-line performance analysis. The monitoring is based on the OCM-G monitoring system developed with the EU IST CrossGrid project. We show how the design concepts of the OCM-G enable easy adaptation to the monitoring of GRID superscalar applications. The solution presented in this paper is transparent to the user application. We discuss the issues related to performance analysis of GRID superscalar applications as well as the architecture and implementation issues of integration.

Keywords: grid computing, GRID superscalar, monitoring, OCM-G, performance tools
the infrastructure and the running applications, but also by most Grid services to enable correct resource allocation and job submission, data access optimization services, and schedulers. However, due to its complexity and dynamics, various entities including infrastructure elements, applications, middleware, and others, need to be monitored and analyzed in order to understand and explain performance behavior on the Grid.

One of widely used approaches for distributing applications is the GRID superscalar (GS) [2] which supports the development of applications, in a way transparent and convenient for the user. Its aim is to reduce the development complexity of Grid applications to the minimum, in such a way that writing an application for a computational Grid can be as easy as a sequential program. The idea assumes that a lot of applications is based on some repeating actions, e.g. in form of loops. The granularity of these actions is of the level of simulations or programs, and the data objects will be files.

GS provides an underlying run-time environment capable of detecting the inherent parallelism of the sequential application and performs concurrent task submission. In addition to a data-dependence analysis based on these input/output task parameters which are files, techniques such as file renaming and file locality are applied to increase the application performance. The run-time is underlied by the Globus Toolkit 2.x APIs [3].

The reasons presented above motivated designing a monitoring system that supports the development of applications to be run in the Grid environment using the GS system, to help in their effective and fault-tolerant execution. Unfortunately, existing monitoring systems which provide mainly off-line access to monitoring data do not allow to analyse and react on-line to the performance problems arising within the application’s execution.

In this paper, we focus on a concept and some implementation ideas of adapting the OCM-G system [7], developed in the CrossGrid project [4], to support GS applications. Its role is to help the user or an automatic facility to determine when the application is performing badly.

The features of OCM-G allow to fit it well into the requirements of running an application on the Grid. In particular, we discuss what metrics are important to assess the performance of the application, these related to standard metrics like an operation time as well application specific metrics, expressed in a special language, PMSL, allowing the user to define performance indicators most meaningfully giving the context-dependent features of the application. Then we come to the general architecture of the functioning of OCM-G in the GS, and its implementation details.
2 Related work

The ability to monitor and control the elements of a GRID superscalar enabled application is useful for its efficient execution and the environment itself. A number of tools are designed specifically for the Grid, other ones come from distributed environments.

Paraver [5] is a tool that provides performance information on GS applications with a lot of informative displays. Its main drawback lies in an off-line-oriented mode of operation, so it does not allow to undertake actions whenever interesting events occur. There are a number of systems for monitor Grid infrastructure like Ganglia[9] or JIMS [8], however there are not designed for application monitoring.

To provide meaningful information on the performance of GS applications, a monitoring system needs to supply monitoring data on-line, preferably to operate in the event-action paradigm. The monitoring system should function as a distributed system to avoid scalability problems. Moreover, it should enable to provide performance data in such a way so to make it as application-specific as possible.

To assess the performance features of an application running in GS environment, we need to analyse such metrics as Communication volume and associated overhead, Overhead due to task synchronization, Time of data dependency solving, File forwarding time, Task submission time, Task execution time, Resource availability time. Part of this data can be based on getting relevant events captured by the monitoring system. Otherwise performance evaluation would need accessing data from the GS run-time.

3 Grid-enabled monitoring vs. GS applications

Due to the above requirements, we have decided to use the OCM-G [7] described below, since it is Grid-enabled and compliant with monitoring standards [1]. The modular architecture of the OCM-G separates the actual monitoring system from the tools that are intended to analyze selected monitoring data. This feature fosters mutual independence of system components and enables users to use their own tools to monitor application performance without any additional effort. The OCM-G is an autonomous, distributed, decentralized system which exposes monitoring services via a standardized
interface called OMIS [1].

The OCM-G is composed of two types of components: per-host Local Monitors and per-site Service Managers (SM). Additionally, there is a Main Service Manager (MSM) which distributes data to and collects it from site-SMs. The architecture of the OCM-G is shown in Fig. 1. The Main Service Manager is the component which exposes the functionality of the system to performance analysis tools. Owing to the standardized protocol used between the monitoring system and possible tools, OCM-G can be very easily adapted to the architecture described in Section 3.

The most important features of the OCM-G include low monitoring overhead, flexibility, security, and transparent service-oriented operation. These features are provided inherently due either to the ideas of OMIS underlying the OCM-G or to the implementation properties of the system.

The OCM-G has been successfully used with an independent performance analysis tool for Grid applications, the G-PM [6] tool which provides a large number of standard and user-defined metrics for interactive Grid applications. The latter metrics are supported with the Performance Measurement Specification Language (PMSL).

The way in which OCM-G is used to monitor MPI applications is de-
scribed in [7]. In the present paper we propose adaptation of these solutions to different constraints defined by GS–based application features. Below, we focus on the architecture and functionality of a monitoring system for GS applications.

The architecture of a GS-oriented application monitoring system is presented in Fig. 2. Between GS application and GS run-time, we insert an additional event triggering wrapper. The wrapper transparently passes calls from the application to GS run-time, and sends all required monitored data (the name of a GS method, and call’s parameters) to the OCM-G.

![Figure 2: Monitoring of GS–based application - architecture](image)

Our system also supports monitoring other user events, e.g. it can send all kinds of messages (integers or floating numbers, character strings) to the monitoring tool. This is done using probes, functions that trigger events. The tool which is connected to OCM-G can enable/disable probes in order to enable/disable event triggering.

By using the OCM-G, it is possible to gather all information needed by a tool which supplies information directly to the user or an automatic tool. In particular, the system can monitor invocations of GS primitives: \texttt{GS\_On()}, \texttt{GS\_Off()}, \texttt{Execute()}, \texttt{GS\_Open()}/\texttt{GS\_Close()}, \texttt{GS\_Barrier()}, etc.
The system allows for gathering the data needed for performance measurements such as the amount of data sent, time of data transmission, and process identifiers. The OCM-G architecture allows to control the amount and frequency of monitored data sent from the monitored application to monitoring system.

To allow the system under discussion to monitor GS–based applications, some specific start-up steps have to be performed:

- OCM-G should be initialized in the application code,
- **OCM-G Main Service Manager** (MSM) should be up and running,
- an application should be started (this step should be performed with additional parameters pointing to the **Main Service Manager**. During this step, the **Service Manager** (SM) and the **Local Monitor** (LM) are created automatically, if needed, by the application process, or the process can connect to already running SMs and LMs,
- now, any component that can communicate using the OMIS [1] protocol can connect to the MSM, subscribe to the required probes and receive monitored events, e.g. to make decisions about moving the application to another Grid node if necessary.

As mentioned before, it is possible to use the OCM-G to monitor user events by inserting custom probes into the code. These probes can be monitored just like any other GS events, as described above.

## 4 Implementation issues

In order to intercept calls to GS run-time primitives we instrumented the GS run-time library. OCM-G distribution provides a tool that performs instrumentation of application libraries to enable triggering the events related to *enter/exit* to library functions. The developer must provide a library and a so-called instrumentation description file that specifies which functions inside the library should be instrumented and which of its parameters should be sent to the monitor when the function is called. But this tool cannot be directly used in the case of GS run-time libraries, because it operates on static libraries, and the GS binary distribution provides a compiled runtime in form of dynamic linked libraries. A simple solution to overcome this
limitation is just to create a static library wrapper for dynamic linked GS libraries and then instrument the wrapper with the provided tool and link to the application. We also take into consideration adapting the existing instrumentation tool for dynamic libraries.

Distributed and parallel applications usually consist of concurrently executed processes. In order to identify which processes belong to the same application, OCM-G introduces an application name as ID. In order to be registered in OCM-G monitoring system, each process must run with the application name and with the address of MSM. Usually, it can be done by passing all this information to an application command line. The GS also uses the application name to bind distributed parts of the application, a front-end, called master, on the client machine, and workers on the computational GRID. Both OCM-G and GS uses similar models to deal with distributed application, what simplifies an integration of these systems. In order to allow the monitoring of an application execution, a code line that is responsible for a process registration into OCM-G must be inserted to the master as well to the worker programs. Next, both programs must be compiled using the OCM-G wrapper script that links with additional OCM-G related objects and libraries.

An application prepared in this way can be controlled and observed using OCM-G. The inserted instrumentation code sends all required monitoring data (invoking GS methods) to the monitor. All these operations are completely transparent to the user application and no changes to the original GS runtime source code are needed.

5 Summary

Distributed GS-based applications usually require access to large-scale computing resources. This fact poses a need in a system that handles the execution of such applications and ensures their effective performance and robustness.

In this paper, we focus on an approach to the monitoring of GS applications. It is based on the OCM-G monitoring system [7]. We present the design and implementation issues of the research. The achieved output will be used in the G-PM performance evaluation tool to decide when the performance of the application should be improved. In our further papers we will address the issues of performance analysis of GS applications.
Acknowledgments The authors wish to thank Prof. Rosa M. Badia for valuable discussions and suggestions.

References


[3] \url{http://www.globus.org}

[4] \url{http://www.eu-crossgrid.org}

[5] \url{http://www.cepba.upc.es/paraver}

