A study of Grid Performance Measurements using G-PM and OCM-G

June 14th 2004

Derek Groen
(djgroen@science.uva.nl)
Abstract
Performance measurement has always been an important aspect within Distributed Computing, and the arrival of Grid Computing provided even more challenges (and applications) for performance monitoring tools and benchmarks. Instead of simply being used for application assessment, performance data can now also be used for the job migration in a more optimal fashion. With the Grid rapidly developing for several years, the first serious performance measurement projects are starting to mature. This report provides an overview of some projects, and also focuses on a particular case to find out how complex it is to monitor scientific application running on the Grid nowadays. We attempted to install and run a single Grid application with monitoring enabled. By doing so we discovered that Grid performance monitoring seems to be in an early phase and that there is a lot of work to be done.
1. Introduction
Once upon a time, performance measurement was relatively simple. With one computer, one processor, one application to execute at a time and no networking, program execution time could easily be measured, and was representative for the performance of the application. As the computing world grew and evolved, however, systems grew increasingly complex. Parallel computing opened the door to different and more complex ways of measuring, introducing the concept of running one program on multiple nodes and revealing the importance of scalability and parallelism.

Conventionally, the main advantage of measuring performance was to evaluate the performance of a specific program under specific static conditions. The results were then used to find potential bottlenecks or to discover how suitable the application was for a given system.

With the arrival of Grid computing middleware a new component known as a Resource Broker (RB, see [1]) became necessary. The RB is expected to assign various resources to processes in an optimal way under dynamic circumstances. Conventional Distributed Computing assumes more static conditions where users access machines at which they have an account. On the other hand, Grid computing does not allow single performance measurements to represent a Grid virtual machine (VM) as it changes in physical location over time. These dynamic conditions make recent measurements more likely to be accurate than older measurements of the same Grid VM [2].

These dynamics create a whole new viewpoint for performance measurement. Instead of just providing users with a measurement overview, a RB can now also be provided with performance information. With this information, a RB may be able to determine the optimal location for a given job more precisely, and may be able to prevent future jobs of a similar type from getting run on overloaded or slow computing nodes. But to predict performance and therefore adjust the allocation in a sensible way, the RB needs to receive recent and useful performance data from monitoring tools. Then, is it possible to provide both recent and useful performance data with the current monitoring tools?

This issue is central throughout this paper, and an attempt is made to answer it in a two-fold way. First, we give an overview of the existing projects in the Grid Performance Measurement area, along with a comparison of the different approaches. Second, there is a detailed report of experiences with a specific case study. This case study describes the installation and testing of a specific performance monitor on a Grid, and some test runs with a scientific application. After this, a conclusion is, and several suggestions for future work are given.
2. Literature Review
This chapter presents an overview of the different projects and products that are related to Grid performance measurement. Before the overviews of the different projects in the field, some general notes about performance measurement are made. Also, we discuss a few projects that are indirectly involved with Grid performance measurement, but still have relevance to the matter. The final part of this chapter picks out some points of special interest and describes the general state of the Grid performance measurement world.

2.1 Performance Measurement
There are basically three types of performance measuring. The first type is performance monitoring, which encompasses measuring performance of an existing application by means of ‘monitoring’ the traffic. The second type is ‘real benchmarking’, which grades a computer based on the performance of certain benchmarking code. Usually these kinds of benchmarking code are not full applications, but represent implemented computational algorithms instead. Finally, the third type is performance measuring is performance simulation, which basically consists of simulating an actual Grid in software, and measuring the performance by examining the simulation.

Although performance simulation is highly useful to test systems that are not in production yet, or to discover obvious bottlenecks, it is harder to use such a program to measure the raw performance in a given time step in real-life circumstances. Therefore in this work we will especially focus on the first two types of performance measurement (a brief overview of a few Grid performance simulation tools can be found in Appendix A).
### 2.2 Grid Performance Measurement Overview

Table 1 provides a birds-eye overview of 9 different projects related to Grid performance measurement. Several known Grid Performance Monitors and Grid Benchmarks are included, as well as two other applications that are closely related to measuring network performance in a Grid. Further descriptions of the individual projects are provided below.

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Project Type</th>
<th>Origin</th>
<th>Short Description</th>
<th>Latest Release*</th>
<th>Special</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-GMA [3]</td>
<td>Low-level Grid monitor</td>
<td>DataGrid</td>
<td>A monitor that combines the GMA [2] concept with RDBMS model.</td>
<td>3.4.18</td>
<td></td>
</tr>
<tr>
<td>GRM [4]</td>
<td>R-GMA based monitoring tool</td>
<td>DataGrid</td>
<td>Allows user-defined monitoring with R-GMA</td>
<td>2.5.4</td>
<td></td>
</tr>
<tr>
<td>OCM-G [5]</td>
<td>Low-level Grid monitor</td>
<td>CrossGrid</td>
<td>Measures individual metrics, which can be used by other programs</td>
<td>1.2.5</td>
<td>OCM-G will be used as basis for the case study</td>
</tr>
<tr>
<td>G-PM [6]</td>
<td>High-level Grid performance monitor</td>
<td>CrossGrid</td>
<td>Allows monitoring with user-defined metrics</td>
<td>0.4.0</td>
<td>Like OCM-G, G-PM will be used for the case study</td>
</tr>
<tr>
<td>AIGB [7]</td>
<td>Grid Benchmark</td>
<td>NASA</td>
<td>Grid benchmark based on fluid dynamics applications</td>
<td>3.0</td>
<td>Based on NPB.</td>
</tr>
<tr>
<td>GridBench [9]</td>
<td>Framework for benchmarking on the Grid</td>
<td>CrossGrid</td>
<td>Allows parallel benchmarks to be run on the Grid</td>
<td>0.2.0</td>
<td></td>
</tr>
<tr>
<td>Topomon [10]</td>
<td>Grid Network Monitoring Tool</td>
<td>Vrije Universiteit Amsterdam</td>
<td>incorporates topology info into traffic monitors</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NetLogger [11]</td>
<td>Network Monitoring Tool</td>
<td>Univ. of California</td>
<td>Monitoring tool aimed at distributed environments</td>
<td>2.2.6</td>
<td>Not really suitable for massively parallel applications</td>
</tr>
</tbody>
</table>

*At time of writing. Newer versions may have been released after this.

Table 1: Overview of projects related to Grid Performance Measurement.
2.2.1 Grid Performance Monitors

R-GMA (Relational Grid Monitoring Architecture) [3]

**Origin**
R-GMA originates from the Council for the Central Laboratory of the Research Councils (CCLRC), which is a research group in England. Also, it is a part of the European DataGrid (the foundation of the CrossGrid).

**Properties**
R-GMA, as its name implies, is a mixture of two different concepts. The project is based on the Grid Monitoring Architecture design from the Global Grid Forum (GGF). It represents its information by means of a relational database for every Virtual Organization. There is a browser available which can display monitoring information through dynamically generated web pages, and PULSE is a tool that can provide a simple graphical display of the monitoring results.

**Status**
Successfully deployed (see [12]) and well documented.

GRM (Grid Resource Monitor)[4]

**Origin**
GRM originates from the Hungarian Academy of Sciences in Budapest, and is a part of the DataGrid project.

**Properties**
GRM is an R-GMA based monitoring tool, which delivers trace data to an on-line visualization tool called PROVE. In order to monitor a Grid application properly, the user needs to implement so-called “trace generation functions” in the application source. There are some basic functions including one to start tracing, end tracing and send events, but in addition to that the user can define their own monitoring events (although their format has to be declared before usage). In summary, GRM has two main functions, namely channeling data from R-GMA to the PROVE visualization tool, and allowing users to fuel R-GMA with their monitoring information through trace generation functions.

**Status**
RPM’s and an installation guide are available, but no references were found about deployment attempts.

OCM-G (Grid-enabled OMIS-Compliant Monitor) [5]

**Origin**
OCM-G is a low-level performance monitoring tool from German and Polish origin that measures individual metrics, which can then be used by performance analyzers, benchmarking tools or debuggers to obtain relevant results.

**Properties**
The monitoring consists of three different parts, namely Service Managers, Local Monitors and Application linked libraries. Service Managers start when OCM-G starts and Local Monitors start when the relevant application starts. Finally, the Application linked libraries are embedded in the Grid application itself. Although OCM-G is not exactly suitable for direct benchmarking, it does provide a basis for Grid performance measurement programs (such as G-PM).

**Status**
Version 1.2.4 has been tested and approved for deployment on the CrossGrid.
G-PM (Grid Performance Monitor) [6]
G-PM is a Grid Performance Monitor that is based on OCM-G. A detailed overview of this project is given in chapter 3.1, as G-PM is used for the case study.

### 2.2.2 Grid Benchmarks

AIGB (ALU Intensive Grid Benchmarks) or NGB (Nas Grid Benchmarks) or GridNPB (Grid-enabled Nas Parallel Benchmarks) [7]

**Origin**
The AIGB intends to accomplish on the Grid what the NPB (NAS Parallel Benchmarks) accomplished on the platform of parallel computing. The NPB has been designed, built and used to measure the capability to run computational fluid dynamics applications of certain systems. The NASA Advanced Supercomputing Division develops both the NPB and the AIGB. The AIGB was the first available Grid benchmark.

**Properties**
The NPB consists of five different parallel kernels and three simulated application benchmarks, all mimicking computational and data movement aspects of computational fluid dynamics (also called CFD’s). The AIGB has a slightly different outlook, as it combines the parallel kernels and simulated applications into four “compound tasks”. These compound tasks mimic different kinds of grid applications, and according to the creators, the four compound tasks altogether ‘mimic the computation and data movement characteristics of scientific grid applications’.

**Status**
Although NPB is stable and complete, the port to the Grid is relatively new. It has already been released in public.

**Test types:**
- **Embarrassingly Distributed**
  This compound task runs flow solvers on every available node. Since flow solvers do not need to communicate with each other in order to complete their task, the only communication needed is initial startup instructions and the gathering of final results. Therefore, all of the nodes can work independently, which represents an Embarrassingly Distributed task.
- **Helical Chain**
  This task executes several sequences of BT (Block-Triangle), SP (Scalar Pentadiagonal) and LU (Lower and Upper triangular systems) on the different nodes. Node 1 executes a BT, then Node 2 a SP and then Node 3 a LU, after which Node 4 will execute another BT and so on. BT, SP and LU are all computational fluid dynamics applications, and are executed in a sequential way on the different nodes.
- **Visualization Pipeline**
  The Visualization Pipeline uses sequences of a flow solver, post-processor and visualization module. Unlike the Helical Chain task, the sequences can be “pipelined” when it comes down to dependency. For example, one node can
already start a new flow solver while other nodes are still busy with post-processing and visualizing previous sequences.

- Mixed Bag
  Mixed Bag is somewhat similar to the Visualization Pipeline in design, but the parallelization scheme of Mixed Bag is asymmetric and much more unpredictable. Some tasks are heavier than others and some are more urgent than others, making this compound task especially suitable for testing resource allocation.

**Simplified data flow graphs of the 4 AIGB compound tasks**

*Fig.2.1: Simplified data flow graphs of the 4 AIGB compound tasks. Based on the graphs from the paper “ALU Intensive Grid Benchmarks”* [7]
GRASP (GRid ASsessment Probes) [8]

Origin
GRASP originates from the University of California in San Diego. Whereas AIGB is based on fluid dynamics applications, GRASP is a benchmark that is more oriented towards data-intensive applications in the domain of physics or bioinformatics.

Properties
GRASP contains various Perl scripts and C programs which are used to conduct three different probes. One of the programs is able to generate a 100 MB random data file, which is used as data input. GRASP has been specially tailored for use with Globus.

Status
GRASP has been publicly released and supposedly tested. No explicit test reports were found.

Test types:
- 3-Node probe
  The 3-node probe transfers a 100MB file from node 1 to node 2, executes a ‘compute’ program on this file on node 2 and finally transfers its result to node 3. The amount of computation on node 2 and the amount of output can be defined beforehand. Also, This probe can be launched multiple times creating a structure, which has some similarities with AIGB’s Visualization Pipeline.
- Circle probe
  This probe transfers a 100MB file around a circle of nodes, and performs a checksum on each node. At the end, the difference between the original file and the current file is measured, with a difference 0 being desirable. It is meant to mimic token passing of large data files (somewhat similar to the Helical Chain task in AIGB)
- Gather probe
  The gather probe is largely similar to the 3-node probe, except that the data for the computing node does not come from a single node, but multiple source nodes instead. It is meant to mimic an application that gathers data from multiple sites in the Grid.
Simplified data graph of the 3 different GRASP probes (with additional comments)

3-Node Probe

Circle Probe

Gather Probe

Black arrows represent both control and data flow. Red arrows represent control flow only.

Fig 2.2: Simplified data graph of the 3 different GRASP probes (with additional comments). Based on the descriptions in "Benchmark Probes for Grid Assessment" [8]
GridBench [9]

Origin
GridBench has been developed by the University of Cyprus and is not so much a benchmark on its own, but rather a framework upon which existing benchmarks can be run in a Grid environment.

Properties
The GridBench framework defines 3 different levels upon which benchmarks can be run. These are the Worker Node level, the Site level and the Grid VO (Virtual Organization) level. The worker node level focuses on individual nodes, the site level on individual clusters and the Grid VO level on individual Virtual Organizations. Many popular parallel benchmarks can be run on one of these levels in GridBench. So, in a way, GridBench ports these benchmarks to the Grid

Status
GridBench has been successfully developed and deployed.

Test types:
- Some of the benchmarks which can be run on GridBench are:
  - EPWhetstone
  - High Performance LinPack
  - MPPTest
  - Gb_ftb (file transfer benchmark that measures transfer rates)
  - Some kernels from the NPB (NAS Parallel Benchmarks)

2.2.3 Other Relevant Programs

TopoMon [10]
TopoMon is an improvement to the Network Weather Service that incorporates topology information into network traffic analyzers. In Grids it can be quite likely that conceptually different connections are forced to share the same physical links. TopoMon reveals these potential bottlenecks, and thereby clears up some of the network speed unreliability issues in Grids.

NetLogger is a network monitoring tool aimed at distributed systems. It can be used to trace possible performance bottlenecks or to give a clear performance overview with help of the NLV (NetLogger Visualization) visualization tool. Although this tool is useful for performance monitoring, it is specifically not recommended for Massively Parallel Programs, as the visualization tool only supports 20 event types at a time. Also, it delivers excessive overhead at more fine-grained Grid applications.
2.3 Review Discussion

In the search for information about related work it was rather easy to find theoretical specifications of the different applications. It proved to be much harder, however, to obtain any actual evidence of the progress state. TopoMon and NetLogger make use of well-tested and proven technology, but are insufficient to provide proper Grid Monitoring on their own. Topomon focuses exclusively on network topology, but it does not, for example, monitor the CPU load of certain nodes or the amount of available storage. Although NetLogger has a broader scope than TopoMon, it still lays heavy emphasis on network connections, and does not take statistics of individual nodes into account (such as the earlier mentioned CPU (Central Processing Unit) load or data space).

The current state of Grid Benchmark tools is pretty good overall, with AIGB successfully used in several different projects and GRASP publicly released as well. GridBench still resides in a somewhat earlier stage as it still beta-tested and not very widely used yet. The underlying benchmarks of GridBench are in a stable phase however, so the tool rests on a firm foundation.

The state of Grid Performance Monitors, on the other hand, is less sophisticated. Both of the major Grid Performance Monitoring Projects (R-GMA/GRM and OCM-G/G-PM) take place within Crossgrid, and are still under heavy testing and development. Initial attempts at deploying R-GMA succeeded but did not go without obstacles on their own. OCM-G version 1.2.4 has been tested and approved by the CrossGrid ITEAM, although some issues arose during testing [13].

An interesting contrast appears when comparing the progress of Grid Performance Monitors to Grid Performance Benchmarks. Considering that objective measurements for Grid Performance have not yet been found, it would be intuitive to assume that continual monitoring tools are to be preferred over one-shot benchmarks. Yet after this overview, it can be concluded that the Grid Performance Benchmarks are generally more widely used and in a later stage than the Grid Performance Monitors.

The fact that they are more widely used may have to do with the ease of deployment of benchmarks. Often, they are simple self-measuring applications and therefore they can be adjusted to different platforms without having to intrude into the source of other components. Since Performance Monitors require instrumentation in their respective applications, they take much more effort to be embedded correctly into a new application and platform.

The differences in stages of development are related to the deployment differences, but another important factor to note is the difference of development foundation. At least two out of the three reviewed Grid Benchmarks (GRASP possibly being an exception) are directly overhauled from existing Parallel Benchmark suites. Logically, it gives these projects a head start in terms of development, something that G-PM and GRM only have in a small degree, as they need to be fitted specifically to the structure of the Grid.

All in all there are various reasons for the difference in usage and stages of development, but are these differences in focus justified when it comes to accurate measurement of Grid Performance? The various Grid benchmarks may provide accurate one-shot samples, but the sheer ‘bulkiness’ renders them rather useless for continual gauging of Grid performance (unless you happen to have an extremely fast network). Not only do these benchmarks take a lot of time (by the time the results are in, they are already outdated by new workload
conditions), but they also overload the Grid, possibly wasting a large amount of valuable resources.

It makes much more sense to focus on ways of performance measurement that provide results on a periodical basis (the Resource Broker has more use for more recent data), and stress the Grid as little as possible. Because performance monitors monitor existing programs instead of launching a new benchmark the impact on the Grid capacity is limited.

Grid benchmarking certainly has its uses, but it might make little sense to run one suite a single time and chalk down the result. It would make much more sense to base a Grid benchmark on an existing Grid application coupled with a performance monitor, and monitor this application for a much longer time in order to reach a benchmarking score. Because of the time span of the measurement, the result would be much more representative and the use of an existing Grid application ensures the realism of the benchmark (and also reduces the impact on capacity if the application is executed for a productive purpose other than benchmarking).

In short, good Grid performance monitors are the key to accurate Grid performance measurement and may be much more vital than Grid Benchmarks.

In order for a performance monitor to succeed, it needs to satisfy several requirements: First of all it needs to be stable, running reliably across the Grid without glitches or crashes. Second, it needs to be easy to install. In order to accomplish that it should have correct documentation and a convenient way of deployment. Third, it should provide data that is relevant to the performance of a particular application. Either the performance monitor should detect this automatically, or allow the user to implement this. Fourth, it should be easy to use. If every Grid application is to be instrumented with monitoring commands, then it is important that application developers can instrument their application in a convenient way. Also, they should be able to instrument their application despite limited knowledge of the Performance Monitor. Also, an application should easily be able to be run with proper monitoring. Fifth, the performance overhead needs to be as small as possible. And finally, it should be able to output this data in a way that is useful for consumers (be it a human analyst or a RB).

A simple overview is not enough to provide insight into these matters; therefore a more detailed analysis will be given of one specific Grid Performance Monitor.
3. G-PM and OCM-G as a Case Study
We have provided a general impression of some Performance Monitoring related projects publicly available. Instead of briefly running and testing all of the projects mentioned in the overview we focus on a single project in depth.

We have G-PM 0.4.0 and OCM-G 1.2.5 for our study, which are both CrossGrid projects largely developed in Germany and Poland. OCM-G has had many prior versions and tests, and although it does not provide complete reliability, version 1.2.4 has been extensively tested (see [13]) by the CrossGrid Integration Team and approved suitable for deployment. G-PM has only barely surpassed the state of being a prototype, and was tested by the CrossGrid Integration Team at the same time this case study took place.

G-PM will receive the main focus, as the program will be assessed on its capabilities to provide both recent and useful performance data. In addition to that, the requirements of a good Grid Performance Monitor will be reviewed and it will be determined how well G-PM fits these requirements.

First, we give an explanation about G-PM. After that the requirements will be explicitly stated, followed by an experimental set-up, which aims at testing these requirements. Finally, the results of the tests are given.

3.1 The Grid Performance Monitor (G-PM)
G-PM is a part of the CrossGrid project, and the initial design and development of the tool started in 2002, during the early days of the CrossGrid project. The main developers of G-PM are Roland Wismueller from the TU München and Włodzimierz Funika, Tomasz Arodź and Marcin Kurdziel from the University of Krakow.

Although until recently G-PM did not explicitly require OCM-G, it is fully based and dependant of the OMIS-Compliant Grid Monitor. According to the developers ([14]), G-PM has four main goals:

- On-line Analysis and Visualization (direct correlation between performance and interactions)
- Support for User Defined Metrics (with minimal effort from the user)
- Meaningful Performance Measurement in the Grid (relate application performance to that of the Grid infrastructure)
- Efficient Distributed Data Evaluation (scalability, no centralization of event traces)

In order to accomplish these goals, the designers of G-PM have chosen to provide a set of „standard“ metrics (such as memory size or communication time), accompanied with the ability to define custom metrics. These custom metrics can be based on the existing standard metrics or on application-specific events that can be defined by instrumenting the source code of the application. This instrumenting is done by placing specific ‘probes’ in the source code, which pass on instructions to specifically start, change (increment) or end a particular measurement.

The measurements themselves are defined with the Performance Metric Specification Language (PMSL) in a separate probes file. In addition to that measurements can be picked out and adjusted in the G-PM Measurement Interface.
Fig. 3.1: Overview of OCM-G and G-PM [14].

Fig. 3.1 provides a clear overview of how OCM-G and G-PM cooperate in order to generate an accurate performance measurement. OCM-G gathers performance data from various processes (labelled P1 to Pn in the overview) and third party applications, such as Grid Information Services. The obtained data can then be retrieved by G-PM through the On-line Monitoring Interface Specification (OMIS). This data is initially processed in the Performance Measurement Component (PMC) and the High-Level Analysis Component (HLAC). The behaviour of these two components can be modified by the used through the Measurement Interface. Finally the modified performance data is shown in the User Interface and Visualization Component (UIVC).

As a reference, it may be useful to rephrase the requirements mentioned in the previous chapter in a more clear form:

- G-PM needs to be easy to install. In order to accomplish that it should have correct documentation and a convenient way of deployment.
- G-PM needs to be stable, running reliably across the Grid without glitches or crashes.
- G-PM should be able to provide data that is relevant to the performance of a particular application. Either G-PM should detect this automatically, or allow the user to implement this.
- G-PM should be easy to use. If every Grid application is to be instrumented with monitoring commands, then it is important that application developers can instrument their application in a convenient way. Also, they should be able to instrument their application despite limited knowledge of the Performance
Monitor. An application should easily be able to be launched with proper monitoring.
- G-PM should be able to output this data in a way that is useful for consumers (be it a human analyst or a RB).
- The performance overhead of G-PM needs to be as small as possible.

3.2 Experimental Set-up
It is important to note that this section describes the initial setup of the case study. During the case study the configuration changed. These changes are reported in their respective places.

All of the applications were run on RedHat 7.3 machines with Globus 2.2.4 and compiled with gcc-2.95.2. An attempt to install and run OCMG and G-PM was made on the Das2 in the Netherlands (fs0.das2.nikhef.nl).

OCM-G version 1.2.5 was chosen for the case study, as this was considered the latest stable release by the developers. G-PM 0.4.0-1 was picked for the same reasons.

In addition to that the simplified BStream blood flow solver 1.0, [15], was selected as application to be run monitored. This application has been instrumented by Roland Wismueller with several example probes for OCM-G. Mpirun was used to execute this application.

The course of action was as follows:
1. Install OCM-G and G-PM
2. Install the BStream Solver
3. Execute the BStream Solver with mpirun without further monitoring
4. Execute the BStream Solver with mpirun and monitoring enabled.
5. Assess G-PM.

3.3 Results

3.3.1 Step 1: Installation of OCM-G and G-PM
As a start, we attempted to install OCM-G 1.2.5 on fs0.das2.nikhef.nl. To do this, a manual compilation was performed on the Das2. Right away, several crucial Globus libraries turned out to have incorrect permissions. Since these errors showed up one after another, and the availability of the Das2 administrator was very limited, it was soon decided that it would be best to first get started on the official Crossgrid autobuild site before migrating to fs0.das2.nikhef.nl.

The Crossgrid autobuild site is located at ui100.fzk.de. Like fs0.das2.nikhef.nl the site is equipped with Globus 2.2.4 and gcc 2.95-2.

Because the current autobuild versions of OCM-G and G-PM on the user interface machine at Forschungszentrum Karlsruhe (ui100.fzk.de) were outdated, it was recommended to manually install the new versions. This was done by configuring and making the source code.
- The first attempts at compiling OCM-G at ui100.fzk.de failed due to the fact that somewhere during copying the files, carriage returns (cr) were introduced into the source files. A later examination showed up that these cr were introduced by unzipping the source files with Winzip. A different way of obtaining and unpacking the files easily solved this issue.
- At the same time, it was discovered that $ocmgroot/bin/makeregistry and $ocmgroot/bin/gen_ext_register did not have execute permission for owner. This too was easily corrected with a permission change.
- Several libraries in /opt/globus/etc/globus_packages/globus_openldap were not world-readable, but this was easily corrected by Marcus Hardt, the administrator at fzk.de

After these issues were resolved, G-PM and OCM-G compiled successfully, as well as the BStream application.

**Getting BStream up-to-date**

Now it was time to run the BStream solver without further monitoring. Initially the following output was generated:

```bash
bash-2.05a$ mpirun -np 4 BStream_l tube
ssh: connect to address 137.138.229.41 port 22: Connection timed out
p0_27121: p4_error: Child process exited while making connection to remote process on lxshare0405.cern.ch: 0
bash-2.05a$
```

In order to run properly, a machines file had to be created, containing several machines to run the solver on. After this was done, the BStream solver did start and run, but refused to end after an hour of running.

For sake of comparison, the original non-instrumented BStream solver was installed on ui100 and completed its task within 20 minutes. Further examination revealed that the difference was caused by the high amount of iterations specified in the configuration file.

Whereas the original Simple bloodflow solver ran for 900 iterations, the instrumented version kept running for a total of 8000 iterations. Although this problem was easily solved by editing `tube.conf`, another interesting difference arose during the examination of the two applications. Besides the instrumentation added, there were several other differences between in the code of the two programs. After some brief research it turned out that the ‘original’ version was actually a newer version than the one instrumented originally.

It was decided to re-instrument the original version as this version was much more reliable. This re-instrumentation went smoothly, and after doing so the BStream solver indeed turned out to be quite reliable.
**OCM-G 1.2.5 issues**

Armed with a freshly re-instrumented, recompiled and tested BStream solver, we executed the BStream solver with monitoring enabled. Initially, the following error came up:

```
bash-2.05a$ gpm Blood --terminate --num-procs 4 --ocmg-mainsm 8d34a02a:4e21
Can't open a configuration file!
Using default values!
SiteSM host in not set! Trying using localhost
```

This was caused because of calling ‘gpm’ (which called an outdated version at /opt/cg/bin/gpm because 0.4.0 failed to compile at the autobuild) instead of ‘./gpm’. After correcting that, the following showed up in ocm-g, however:

```
bash-2.05a$ ./cg-ocmg-monitor
Can't open a configuration file!
Using default values!
SiteSM host in not set! Trying using localhost
```

These messages are normal for much older versions of OCM-G, but were supposed to be removed in version 1.2.5. Since it was quite clear that version 1.2.5 was executed, it was time to dig further into the configuration, compilation and execution output to see if there is something. The amount of output supplied by OCM-G and G-PM during configuration and compilation is quite huge, but it was time to read all of it.

After a careful reread of the G-PM output of ‘./configure’ the following was noticed:

```
ERROR: Cannot open /opt/globus/libexec/globus-build-env-gcc32dbg.sh!
(Hint: This error is most commonly caused when no packages of the specified flavor type are installed..)
checking for OCMG... no
*** Warning: OCMG not found!
*** Using built-in version of OCMG for G-PM building.
*** This may lead to errors.
```

This error did not so much derive from an issue with OCM-G or G-PM, but rather an issue with the Globus autobuild machine at FZK. The reason why OCMG was not detected was not so much because it was placed in a wrong directory, but rather because the correct globus-IO flavor was not installed on the autobuild. The lack of this flavor did not only give errors during this case study, but also with the automatic building of G-PM and OCM-G on the machine.

And because of the failure in automatic building, the current versions of GPM and OCM-G were not available, which resulted in executing an outdated version when using commands like ‘gpm’ instead of ‘./gpm’ in the correct directory.

But even with this issue out of the way, there still was the strange ‘old’ configuration file message that showed up during OCM-G execution. Upon submission of this message to Roland Wismueller, he discovered that it was not so much a glitch in version 1.2.5, but that the 1.2.5 package was completely broken, and contained older file versions than the OCM-G 1.2.4 package.

Almost immediately, the WP 3.3 team worked on OCM-G 1.2.6 while the administrators at the FZK focused on fixing the autobuild machine.
Within a few weeks version 1.2.6 of OCM-G was officially released and the issues at the autobuild were solved. Both G-PM and OCM-G 1.2.6 were reinstalled just in case, and this time only two minor issues showed up.

The first issue had to do with the fact that both programs were installed in a user-specific directory. Because of this, it was not possible to run the BStream solver on multiple nodes with monitoring enabled. The second issue was that the instrumented MPI library was not created, but a single addition to the $PATH and the execution of a script called cg-ocmg-mpich-instr easily solved this problem.

### 3.3.2 Running G-PM

Once this issue was resolved, it was possible to run G-PM combined mpirun and obtain an impression of the graphical user interface. One downside of running the blood flow solver using mpirun is that it is limited to using pre-selected nodes. It runs more like a parallel application than a Grid application in that respect.

Despite the fact that there were only a few days left, we decided that it was worth the effort to try and get the blood flow solver running on the Grid with edg-job-submit. Edg-job-submit launches a job by reading its corresponding Job Definition Language (jdl) file and submitting this to the RB, which dispatches the job to a suitable location.

We created a new jdl file for the solver by modifying an existing example, which executed a separate script called wrapper.sh. The contents of both files are available in Appendix D. With these scripts available, we performed the following steps:

1. Logging in to ui100.fzk.de using "ssh -X" three times on three different terminals.
2. Running grid-proxy-init
3. In terminal 1, we ran the following:
   ```
   rm -fr /tmp/ocm* /tmp/.ocm*
   cd /home/site/alfredo/dgtestarea/opt/cg/bin
   ./cg-ocmg-monitor
   ```
   Note the connection string! (often 8d34a02a:4e20 in our case)
4. In terminal 2, we ran the following:
   ```
   cd /home/site/alfredo/dgtestarea/modified_solver/B_Stream1.0_Light
   edg-job-submit modified_solver.jdl
   ```
5. Waited until the application linked with OCM-G (noticeable by the fact that OCM-G generated the following output):
   ```
   PEER [ rendes ] TYPE STATE 0xMID SOCK/OFF
   0 MID RSHPID CONNECTED
   ```
6. In terminal 3, we ran the following:
   ```
   cd /home/site/alfredo/dgtestarea/opt/cg/bin
   ./gpm Blood --terminate --num-procs 2 --ocmg-mainsm 8d34a02a:4e20
   ```

The corresponding job status and final output during execution can be viewed in Appendix C. Note the presence of the files velocity.0 and velocity.1 in the output after execution. This indicates that the simple blood flow solver has managed to complete its task. The example in
Appendix C coincidentally was assigned to the FZK cluster, but successful submissions have also been performed to other clusters in both Germany and Poland.

In addition to executing the blood flow solver on the Grid, we were able to start G-PM and start the execution of the solver by manually specifying this through the user interface.

![Image of measurement selection screen in G-PM](image)

**Fig. 3.2. The measurement selection screen in G-PM.**

Fig. 3.2 shows how a measurement can be selected within G-PM. In this example, we run two different processes on the FZK cluster. The menu on the left side shows what kind of standard metrics can be selected for measurement, and the Hosts menu in the middle is used to select the desired cluster. In this cluster, one or more processes can be selected for measurement. Finally, in the bottom left the Integration mode can be selected.

In a separate screen, custom metrics can be created from scratch. The loading of predefined settings is not enabled in this prototype of G-PM, but as the options are present in the user interface, it is likely that this will be implemented in the future. Due to some technical issues related to the tool development and lack of time, we were not able to get meaningful measurements on the screen. The fact that the monitored application is running with edg-job-submit and G-PM links with the application does indicate that it should be possible to acquire sensible measurements once the tool is stable.
4 Conclusions

Based on the information provided by the case study, the installation of the instrumented blood stream solver was much more difficult than initially expected. We stumbled upon numerous issues, and we were only partially able to discover how well G-PM meets the requirements mentioned in chapter 3.

One could claim that G-PM is not suitable for production runs, but it is important to keep in mind that most of the issues encountered in the case study were not directly caused by G-PM itself, but rather by an inconsistent OCM-G release and middleware conflicts on the FZK cluster. Once this inconsistent release was superseded by a correct and improved version, G-PM turned out to be much easier to install.

Because of the limited time-span of this project, we were only able to run G-PM without being able to define and perform proper measurements. The user interface is quite detailed, and the visualization tool made a generally stable impression, despite its early state. Whether the measurements provided by the G-PM metrics and the PMSL are sufficiently complete is hard to say, but with a much more stable OCM-G release and a reliable, instrumented simple stream solver the door has been opened to finding this out in the future.

The difference in loading times between running the blood stream solver monitored and unmonitored were marginal, although by default the application will not start monitored until startup is specifically requested in G-PM (or the ‘-ocmg-regcont’ flag is enabled). Once the application is started from the G-PM interface, the monitor runs for as long as the application runs and continually gathers performance data. This data cannot be exported as of yet, and it is uncertain whether this will be implemented in the future. It might be interesting to take a look at the RB if it is able to obtain performance data from G-PM.

Overall, G-PM is project that has a lot of potential, but lacked a solid basis to operate on at the start of this project. A lot of the issues we found have been solved, and along with a much more stable OCM-G release that is a good step towards a stronger fundament for G-PM.

Finally, only one question remains: is it possible to provide both recent and useful performance data with the current monitoring tools? G-PM is able to provide recent data, although means to export this data are yet to be implemented. It is hard to say whether the data G-PM supplies is useful. We have only had the chance to discover G-PM to a limited degree and moreover, it has yet to be discovered what metrics are useful for Grid Performance Measurement.
Acknowledgements
Thanks go out to:

- Alfons Hoekstra, for guiding me in the early stages of the project and giving highly useful feedback.
- Alfredo Tirado Ramos, for guiding me throughout the project and helping out in countless ways.
- Roland Wismueller, for supplying the initial instrumented BStream application, an example jdl file and wrapper script and providing heaps of support during this project.
- Marcus Hardt, for providing solid support on the FZK site related issues.
- All the teachers and students at the UvA Bachelor group 2003/2004 for providing advice and feedback during meetings.

References


2. Tierney et al. 2002
   “A Grid Monitoring Architecture“
   From the Global Grid Forum Document Process, GFD.7

3. Norbert Podhorszki and Peter Kacsuk
   “Presentation and Analysis of Grid Performance Data”
   R-GMA Website: http://www.r-gma.org

4. Norbert Podhorszki and Peter Kacsuk
   “Presentation and Analysis of Grid Performance Data”
   WP3 Website: http://hepunx.rl.ac.uk/edg/wp3/

5. Bartosz Baliś, Marcin Radecki and Tomasz Szepieniec
   “Prototype documentation WP 3.3.1”

6. Harald Kornmayer and many others
   “Deliverable 2.5 Demonstration and Report on WP2 2nd prototype” (section 4.3)
   Internet: <http://gridportal.fzk.de/distribution/crossgrid/crossgrid/wp2/CG2.0-D2.5-v0.1-FZK019-2ndPrototypeReport_20040128.doc>

7. R.F. van der Wijngaart and M.A. Frunkin
   “ALU Intensive Grid Benchmarks”
   Internet: <http://www.ggf.org/Meetings/GGF11/Documents/GB_ALU.pdf>
8. Greg Chun, Holly Dail, Henri Casanova and Allan Snavely
   “Benchmark Probes for Grid Assessment”
   Grasp Website: <http://grail.sdsc.edu/projects/grasp/>

9. George Tsouloupas and Marios D. Dikaiakos
   “GridBench: A Tool For Benchmarking Grids”
   Internet: <http://grid.ucy.ac.cy/Papers/gridb.pdf>

10. Mathijs den Burger, Thilo Kielmann, and Henri E. Bal
    “TopoMon: A Monitoring Tool for Grid Network Topology”

11. Brian Tierney, Dan Gunter
    “NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging”

12. Andrew Cooke and many others
    “R-GMA: First Results After Deployment”

13. Mario David and Jorge Gomes
    “OCM-G Test Report” (1-1-DRAFT-B)

14. Roland Wismueller
    “Performance Analysis on the Grid”

15. The bloodflow solver used is a stripped down version of the production bloodflow
    solver developed by the Section Computational Science, University of Amsterdam.
    For details we refer to A.M. Artoli, Mesoscopic Computational Haemodynamics,
Appendix A: Grid Performance Simulation

This appendix provides a brief overview of a few performance simulation projects.

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Project Type</th>
<th>Origin</th>
<th>Short Description</th>
<th>Latest Release*</th>
<th>Special</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bricks</td>
<td>Grid Simulation Application</td>
<td>Various universities &amp; companies in Japan</td>
<td>Grid simulation aimed at comparing scheduling schemes</td>
<td>None available</td>
<td></td>
</tr>
<tr>
<td>MicroGrid</td>
<td>Grid Simulation Application</td>
<td>Uni. of California + Uni. of Illinois</td>
<td>General purpose Grid simulation</td>
<td>2.4.4</td>
<td></td>
</tr>
</tbody>
</table>

Performance Simulation descriptions

Bricks (simulation)
Bricks is able to simulate different types of global grid systems and is aimed at analyzing and comparing various scheduling schemes on a given platform. Several parameters can be set within this program, including network topology, communication models and server architectures. The Bricks project is a collaboration of various universities and research groups in Japan and was first defined in August 1999. Although Bricks can function as an excellent tool for scheduling decisions, it is not a realistic tool to measure real-life performance with.

Microgrid (simulation)
Considered to be “Emulation Tools for Computational Grid Research”, Microgrid is also a simulation tool, but with an even more general purpose than Bricks. The benchmarking aspects of this project are barely worked out; hence this project is not of much interest at the moment.
Appendix B: Differences between Roland’s Instrumented version and Lilit’s ‘original’ Bloodflow solver

BStream_l.cpp

the original version has:

```c
#define SLICE 0 /* Slice decomposition */
#define BOX 1 /* Box decomposition */
#define ORB 2 /* The 2D ORB decomposition strategy */
```

after

```c
#define BUFFER_LENGTH
```

but the modified version does not.

-----

original version has

```c
/* Decompose Simulation Box */
if (ge.decomp_mode == SLICE)
{
    ge.decompose_slice (); /* decompose data in slices "decompose.h" */
}
```

where modified version has

```c
ge.decompose_slice (); /* decompose data in slices "decompose.h" */
```

-----

original version has

```c
omega = 2. * PI / 600;
    nu = radius * radius * omega / alpha / alpha;

    ge.tau = (6. * nu + 1.) / 2.;

    /*check your geometry */
    nr_fluids = ge.calc_nbr_fluids (); /* statistics.h */
```
nu = 0.1 * radius / Reynolds * sqrt (3.);
ge.tau = (6. * nu + 1.) / 2.;

-----

original version has

#ifdef PARALLEL                      /*the "send_*" are from "communication.h" */
    if (ge.decomp_mode == SLICE)
        ge.per_bound_slice (); /*"lbee.h" */
#endif

where modified version has

#ifdef PARALLEL                      /*the "send_*" are from "communication.h" */
    ge.per_bound_slice (); /*"lbee.h" */
#endif

-----
Appendix C: A successful run of the instrumented B_Stream Solver with edg-job-submit

bash-2.05a$ edg-job-submit modified_solver.jdl

Connecting to host rb.fzk.de, port 7772
Logging to host rb.fzk.de, port 9002

***************************************************************************
******************
JOB SUBMIT OUTCOME
The job has been successfully submitted to the Network Server.
Use edg-job-status command to check job current status. Your job identifier (edg_jobId) is:


***************************************************************************
******************

bash-2.05a$ edg-job-status https://rb.fzk.de:9000/G4MvsL3PhP4P2UawCTVUvQ

***************************************************************************
******************

BOOKKEEPING INFORMATION:
Printing status info for the Job:
https://rb.fzk.de:9000/G4MvsL3PhP4P2UawCTVUvQ
Current Status: Waiting
Status Reason: unavailable
Destination: cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite
reached on: Mon Jun 14 19:46:17 2004

bash-2.05a$ edg-job-status https://rb.fzk.de:9000/G4MvsL3PhP4P2UawCTVUvQ

***************************************************************************
******************

BOOKKEEPING INFORMATION:
Printing status info for the Job:
https://rb.fzk.de:9000/G4MvsL3PhP4P2UawCTVUvQ
Current Status: Running
Status Reason: Job successfully submitted to Globus
Destination: ce100.fzk.de:2119/jobmanager-pbs-short
reached on: Mon Jun 14 19:47:01 2004

bash-2.05a$ edg-job-status https://rb.fzk.de:9000/G4MvsL3PhP4P2UawCTVUvQ

***************************************************************************
******************

BOOKKEEPING INFORMATION:
Printing status info for the Job:
https://rb.fzk.de:9000/G4MvsL3PhP4P2UawCTVUvQ
Current Status: Done (Success)
Exit code: 1
Status Reason: Job terminated successfully
Destination: ce100.fzk.de:2119/jobmanager-pbs-short
reached on: Mon Jun 14 19:56:02 2004

bash-2.05a$ edg-job-get-output
bash-2.05a$ edg-job-get-output
https://rb.fzk.de:9000/G4MvsL3PhP4P2UawCTVUvQ

Retrieving files from host rb.fzk.de

*************************************************************
JOB GET OUTPUT OUTCOME
*************************************************************

bash-2.05a$ cd /tmp/jobOutput/G4MvsL3PhP4P2UawCTVUvQ
bash-2.05a$ ls -la

OCM-G Version
Creating /tmp/.ocmglock_6020
*** Starting new LM...
LM is starting...
App. Proc: reg.message:Blood wn200.fzk.de ./BStream_l_ocmg 23036 0
134910964 1
Registration successful

OCM-G Version
Creating /tmp/.ocmglock_6020
Lock file already exists!
App. Proc: reg.message:Blood wn200.fzk.de ./BStream_l_ocmg 23044 1
134910964 1
Registration successful
authorization callback: peer identity:
/O=dutchgrid/O=users/O=uva/OU=wins/CN=Alfredo Tirado Ramos
0 PEER [ rendes ] TYPE STATE 0xMID SOCK/OFF
PID SHMID PROCESS
[ic] fifo fd: 8
*** Process 0 (pid 23036) of application 'Blood' registered at 'wn200.fzk.de'! *** Process 1 (pid 23044) of application 'Blood' registered at 'wn200.fzk.de'!
Process 1 registered with OCM-G
Output files are ready
*** unlink(/tmp/.ocmglock_6020)
Process 0 registered with OCM-G
Time per iteration = 0.903007
Output files are ready
~
"bstream.out" 24L, 908C
1,0-1 All
Appendix D: Contents of modified_solver.jdl and wrapper.sh

**modified_solver.jdl:**
Executable = "wrapper.sh";
Arguments = "tube --occg-appname Blood --occg-mainsm 8d34a02a:4e20";
StdOutput = "bstream.out";
StdError = "bstream.err";
InputSandbox = {"wrapper.sh", "BStream_l_ocmg", "tube.conf", "tube.sample"};
OutputSandbox = {"bstream.out", "bstream.err", "velocity.0", "velocity.1"};
JobType = "MpiCh";
NodeNumber = 2;

**wrapper.sh:**
#!/bin/sh

if [ -z $X509_USER_PROXY ]; then
  for i in `seq 1 10`; do
    if [ -f $HOME/.globus/userproxy.pem ]; then
      #                       echo "Setting certificate - slave" >> $HOME/log
      export X509_USER_PROXY=$HOME/.globus/userproxy.pem
      break;
    else
      sleep 1;
    fi
  done
  if [ -z $X509_USER_PROXY ]; then
    #               echo "Can't set user proxy!" >> log
    exit
  fi
else
  #   echo "Copying cert - master" >> $HOME/log
  cp -f $X509_USER_PROXY $HOME/.globus/userproxy.pem
fi

cd `dirname $0`
chmod +x BStream_l_ocmg
./BStream_l_ocmg "$@"