



DELIVERABLE D2.3
PART III: PROTOTYPE DOCUMENTATION FOR
GRIDBENCH

Task 2.2 Benchmarks and Metrics

Document Filename:	CG2.3-D2.3-v2.0-UCY011-PrototypeDoc.doc
Work package:	WP2 Grid Application Programming Environment
Partner(s):	UCY
Lead Partner:	UCY
Config ID:	CG2.3-D2.3-v2.0-UCY011-PrototypeDoc
Document classification:	CONFIDENTIAL (to change go to File->Properties)

Abstract: This document covers the Task2.3 Benchmarks and Metrics (GridBench) prototype. This is the second, updated version of the document, which includes recent development. For the purpose of this prototype, a set of two component benchmarks in the GridBench suite were selected for implementation, as well as some utility components necessary for making efficient measurements.



Delivery Slip

	Name	Partner	Date	Signature
From				
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
1.0	16/01/2003	First Draft	Diakiakos Marios, Tsouloupas George
2.0	28/5/2003	--Updated Document to reflect the current status of development --Included several sections from D2.4 Updated definitions, state of the art added, contribution to grid technology added, implementation structure added, user manual updated.	Tsouloupas George, Dikaiakos Marios

CONTENTS

1. EXECUTIVE SUMMARY	4
2. INTRODUCTION	5
2.1. PURPOSE.....	5
2.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS.....	5
3. REFERENCES	7
3.1. SOFTWARE.....	7
3.2. CONTACT INFORMATION	7
4. STATE OF THE ART	8
5. CONTRIBUTION TO GRID TECHNOLOGY	9
6. IMPLEMENTATION STRUCTURE	11
6.1. INTRODUCTION	11
6.1.1. <i>Micro-benchmarks at the “Worker-node” level:</i>	12
6.1.2. <i>Micro-benchmarks at the “Site” level:</i>	12
6.1.3. <i>Micro-benchmarks at the “Grid” level:</i>	12
6.1.4. <i>Micro-kernels at the Site level:</i>	12
6.1.5. <i>Micro-kernels at the Site level:</i>	13
6.1.6. <i>Application kernels at the Site and Grid level:</i>	13
6.2. PROTOTYPE IMPLEMENTATION	14
6.3. SPECIFICATION AND RSL GENERATION	16
6.4. MDS INFORMATION PROVIDER	20
7. USER MANUAL	23
7.1. DEPENDENCIES	23
7.2. INSTALLATION	23
7.3. RUNNING	23
7.4. GB_EPWHESTONE	23
7.5. GB_SITE_HPL.....	26
8. INTERNAL TESTS	29
9. ISSUES	30
10. APPENDIX A	31

1. EXECUTIVE SUMMARY

Grid benchmarking is a new and promising area of research in Performance Engineering for Grid Computing. To the best of our knowledge, CrossGrid is the only research project funded by the European Union that focuses on this topic. The importance of Grid benchmarking can be attested by the recent establishment of a Global-Grid-Forum Research Group on Grid Benchmarking, which focuses on the specification of benchmarks for the Grid, based on existing benchmarking suites like NAS benchmarks and the appearance of a number of research papers by USA researchers on this topic in upcoming major conferences like Supercomputing 2003.

According to the Technical Annex of CrossGrid (Task Descriptions), Task 2.3 (Metrics and Benchmarks) seeks to “*propose a set of performance metrics to describe concisely the performance capacity of Grid configurations and application performance*” and to “*develop and implement benchmarks that are representative of typical Grid workloads*.” As specified in the Technical Annex, “*benchmarks will be used to estimate the values of performance-metrics for different Grid configurations, to identify important factors that affect end-to-end application performance, and to provide application developers with initial estimates of expected application performance*.” The ultimate goal of Task 2.3 is to “*deploy and validate a suite of benchmarks in the CrossGrid Testbed*.”

The expected outcome of Task 2.3 was given the code-name GridBench. GridBench is a software system that includes:

- A suite of benchmarks (synthetic or application-based) for characterization of Grids. This suite is designed to meet the requirements of Grid benchmarking as described in the Technical Annex. It will include kernels extracted from and representative of the CrossGrid suite of applications, which are considered to be characteristics of emerging Grid applications with interactivity requirements.
- An implementation of the benchmark suite, deployed on the CrossGrid testbed.
- A software management system (part of the CrossGrid middleware and portal) designed and implemented to:
 - Enable the easy specification and configuration of GridBench executions in XML;
 - Provide automatic generation of job-descriptions suitable for execution on the CrossGrid testbed;
- Enable the archival of GridBench metrics and the retrieval from the middleware of measured metrics.

For the purpose of the prototype two benchmarks are implemented as well as some “utility” components such as the RSL compiler that are necessary for the efficient definition and execution of benchmarks.

The remainder of this document includes a section on the “state of the art” of Grid benchmarking, a short description of the contribution GridBench makes to Grid technology. It then proceeds to give a description of the current functionality of GridBench including a description of each component. This is then followed by usage information.

2.INTRODUCTION

2.1.PURPOSE

The purpose of this document is to describe the prototype for Task 2.3 Benchmarks and Metrics. This document includes a general description of the benchmarks that have been chosen to serve as a prototype. The document also includes a description of the input (parameter) file. Directions are given for building an RPM package and invoking the benchmark.

2.2.DEFINITIONS, ABBREVIATIONS, ACRONYMS

MDS	Metadata Directory service
XML	eXtensible Markup Language
RSL	Resource Specification Language
JDL	Job Description Language
GGF	Global Grid Forum
NGB	NAS Grid Benchmarks
NPB	NAS Parallel Benchmarks
CE	Computing Element
SE	Storage Element
WN	Worker Node
BLAS	Basic Linear Algebra Subroutines
NFS	Network File System
MPI	Message Passing Interface
HPL	High Performance Linpack
DTD	Document Type Definition
Xindice	DBMS for storage, retrieval, querying, and management of XML documents
LDAP	Light-weight Directory Access Protocol
LDIF	LDAP Data Interchange Format
Globus	Grid middleware
GVK	Grid Visualization Kernel
OGSA	Open Grid Services Architecture
SANTA-G	Grid-enabled System Area Network Trace Analysis
API	Application Programming Interface
Jiro	Sun Jiro, an implementation of the Federated Management Architecture, defined by Sun Microsystems
R-GMA	DataGrid relational Grid monitoring architecture
OCM-G	OMIS Compliant Monitoring system for the Grid

VO	Virtual Organization
NAS	NASA Advanced Supercomputing

3. REFERENCES

- [1] CrossGrid Deliverable D2.1, Task 2.3 Requirements Document. March 2003 (last version). CG2.3-D2.1-v1.2-UCY008-SRS
- [2] CrossGrid Deliverable D2.2, Task 2.3 Design Document. June 2002. CG2.3-D2.2-v1.0-UCY001-GridBenchDesign.doc
- [3] CrossGrid Deliverable D2.3, Task 2.3 Prototype Document. Jan 2003. CG2.3-D2.3-v1.1-UCY007-PrototypeDoc.doc
- [4] Jack Dongarra, P.L. and A. Petitet, The LINPACK Benchmark: Past, Present, and Future. December, 2001.
- [5] Dongarra, J.J., H.W. Meuer, and E. Strohmaier, TOP500 Supercomputer Sites, 11th Edition. 1998(UT-CS-98-391).
- [6] Michael Frumkin, Rob F. Van der Wijngaart "NAS Grid Benchmarks: A Tool for Grid Space Exploration," NASA Technical Report.
- [7] CrossGrid Deliverable D2.4, Task 2.2 Internal Progress Report. April 2003. CG2.2-D2.4-v1.2-UST006-ReportDoc.doc
- [8] Bailey et al. NAS Parallel Benchmarks, 1994
- [9] Curnow, H., Wichmann, B. [1976]. "A Synthetic Benchmark", *The Computer J.*
- [10] CrossGrid Document Task 2.3 "Application Questionnaire", March 2003. CG2.3-Q1-v1.0-UCY008-AppQuestionnaire
- [11] CrossGrid Deliverable D2.2, Task 2.3 Design Document. May 2003. CG2.3-D2.2-v1.2-UCY001-GridBenchDesign.doc
- [12] Philip Mucci et al. The BlasBench Report, February 1999
- [13] Dieter Kranzlmuller et al. Grid-Enabled Visualization with GVK

3.1. SOFTWARE

http://gridportal.fzk.de/distribution/crossgrid/crossgrid/wp2/wp2_3-bench/gridbench-0.1.i386.rpm

Contains executables for the HPL and EPWhetstone benchmarks. The executables are statically linked using the "globus-device" for MPI communication.

3.2. CONTACT INFORMATION

Task Leader: Dr. Marios Dikaiakos (mdd@ucy.ac.cy)
George Tsouloupas (georget@ucy.ac.cy)

4.STATE OF THE ART

Grid Benchmarking is an emerging area of research with the first developments appearing over the last year. It is generally recognized that there is a lack of ways for measuring performance, which has led to the formation of a Global Grid Forum Research Group on Grid Benchmarking, whose developments were followed closely by the partners of Task 2.3.

The main product of the GGF Research Group effort has been a specification for “Computationally Intensive Grid Benchmarks”(used to be NAS Grid Benchmarks)[6]. The NGB utilize the NAS Parallel Benchmarks [8] as building blocks. This is done in an attempt to *“provide a pencil-and-paper specification of a benchmark suite for computational Grids”*[6]. The central idea of the specification is the introduction of Data Flows between instances of the NPB kernels (i.e. the output of one kernel is the input of another). The selected Data Flow Graphs correspond to 'typical' Grid applications, and much like NPB, NGB suggests several standard problem sizes. Implementation is left to the implementer who chooses what language and Grid infrastructure to use (i.e. the NAS Grid Benchmarks are only a specification).

5. CONTRIBUTION TO GRID TECHNOLOGY

Task 2.3 and GridBench make several contributions to Grid technology, of which the main ones are given below:

1. We conducted a re-examination and re-definition of the role of benchmarking in the context of the Grid, an open and dynamic wide-area distributed system. According to the GridBench approach, Grid benchmarks are programs that are deployed and run on top of Grid resources to [1]:
 - Generate metrics that describe the performance capacity of selected, isolated Grid nodes through measurements of computational power, file-transfer speed, inter-process communication bandwidth, scalability, etc. Consolidation and proper storage of these metrics can support advanced resource brokers and job schedulers for the Grid. Moreover, it can provide a basis for performance-aware resource allocation and Virtual-Organization formation.
 - Generate metrics that characterize the performance capacity of resources belonging to a Virtual Organization and spanning across multiple Grid nodes, in terms of computational power, file-transfer speed, inter-process communication bandwidth, application-kernel performance, scalability etc. Notably, proper interpretation of these metrics dictates the capability of associating them with adequate descriptions of the actual sets of resources allocated to the particular benchmark-executions that produce metrics data.
 - Generate metrics that characterize the performance of Grid middleware services available at the programming level through API's.
 - Provide a tool for researchers that wish to investigate various aspects of Grid performance, using well-understood kernels that are representative of more complex applications deployed on the Grid. Having access to a corpus of such kernels and being able to easily specify and dispatch parameterized runs of these kernels on Grids, facilitates the characterization of factors that affect application and infrastructure performance, the quantitative comparison of different middleware solutions, algorithms for scheduling, resource allocation, etc.
 - Collect and provide a well-understood and relatively simple set of codes that can be used to test emerging Grid middleware services, to feed performance prediction tools with measured parameters, and to validate performance-prediction approaches. To this end, we are examining a wide range of kernels, including ones extracted from CrossGrid applications of WP1.
2. We introduced a novel approach to Grid benchmarking that consists of a hierarchy of benchmarks seeking to capture different aspects of Grid performance. We experimented with porting established benchmarking codes to the Grid, beyond those proposed by the NASA group, in order to select a small set of benchmarking codes to "populate" the GridBench hierarchy.
3. We are building the first suite of benchmarks for Grids; currently, and to the best of our knowledge, there are no other Grid Benchmarking suites. (The NAS Grid Benchmarks are simply a specification.).
4. We investigated approaches for storing, archiving and providing access to GridBench results (i.e., GridBench metrics measurements) via MDS. To this end, we introduced an MDS data-

model that takes into consideration a high-level model of the Grid architecture, and therefore will enable the interpretation of GridBench metrics at a high-level, close to the conceptual model of Grid-application programmers.

5. GridBench provides a framework for the specification of Grid benchmark executions in XML, and the automatic compilation of job-description scripts (RSL and JDL) out of XML specifications. The most important implication of this is that benchmarks can easily be modeled after specific complex applications. For example, when an application changes with respect to its structure (e.g. switches from one kernel to another) or a new application is introduced altogether, it will be relatively easy to model a new benchmark after it, by specifying (1) how it will execute, (2) the metrics that can be collected and (3) the monitoring data collection, all these through the GridBench benchmark definition language.

GridBench will also provide a graphical tool for the simple analysis of performance measurements with the ability to combine metrics and monitoring of the infrastructure.

6.IMPLEMENTATION STRUCTURE

6.1.INTRODUCTION

The set of components that make up the GridBench suite have the purpose of measuring the performance of the CrossGrid testbed and its constituents. To do so effectively, the suite will have to capture the performance at different levels. The "levels" of the Grid are i) the Worker-node level, ii) the Site level and iii) the Grid level. Apart from the point of view of the Grid layers, GridBench will also provide different approaches to measuring performance: Micro-benchmarks, Micro-Kernels and Application kernels (More details in [1],[2]).

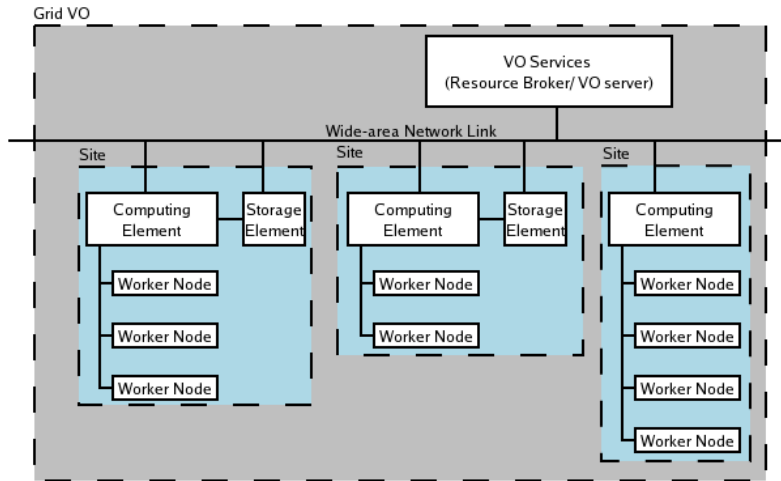


Figure 1: Grid Layers (Grid VO, Site and Worker-Node levels)

The following table shows a break-down of the GridBench benchmark into categories based on the "level" (Worker-Node, site and Grid) and benchmark type (micro-benchmarks, micro-kernels and application kernels).

	Micro-benchmarks	Micro-kernels	Application Kernels
WN	gb_epwhetstone gb_blasbench		
SITE	gb_site_nfs gb_site_mpptest	gb_site_hpl gb_site_nas_*	Gb_site_cg_* [Task1.1] Kernels [Task1.2] Kernels [Task1.3] Kernels [Task1.4] Kernels
GRID	gb_grid_mpptest gb_grid_job	gb_grid_lin gb_grid_ngb	gb_grid_cg_*

The benchmarks that appear in highlight were selected for implementation for the prototype. Descriptions for these benchmarks are included in section 4.

The following is a description of the remaining benchmarks in the GridBench suite which are under development:

6.1.1. Micro-benchmarks at the “Worker-node” level:

gb_blasbench

gb_blasbench is based on the BlasBench benchmark which is designed for testing the performance of the BLAS (Basic Linear Algebra Subroutines) library.

6.1.2. Micro-benchmarks at the “Site” level:

gb_site_nfs

gb_site_nfs is a benchmark to measure disk access performance over NFS. This is useful because the storage available to a Computing Element is over NFS (served by the SE at the same site).

gb_site_mptest

This benchmark is for measuring the performance of MPI calls (e.g. time to perform an MPI_Gather()) on a single cluster.

6.1.3. Micro-benchmarks at the “Grid” level:

gb_grid_mptest

This benchmark is for measuring the performance of MPI calls (e.g. time to perform an MPI_Gather()) using a set of processes that are distributed on the Grid.

gb_grid_job

This benchmark is targeted at measuring the end-to-end performance of the system in starting a job. It includes the start-up of regular and MPI jobs.

6.1.4. Micro-kernels at the Site level:

While the micro-benchmarks attempt to capture the performance characteristics of a site, they do so at such a low-level that it is almost impossible to extrapolate or predict performance. Micro-kernel benchmarks are included because they employ well-known codes and can provide results that can be compared with other high-performance computers.

gb_site_hpl

Linear equation solver based on the High Performance LINPACK[4] benchmark.

gb_site_nas_is

Integer Sort based on the NAS Parallel Benchmarks.

gb_site_nas_ep

An Embarrassingly Parallel benchmark based on the NAS Parallel Benchmarks[9, 10]

gb_site_nas_ft

A fast Fourier Transform benchmark based on the NAS Parallel Benchmarks[9, 10].

gb_site_nas_bt

A Block Tridiagonal solver benchmark based on the NAS Parallel Benchmarks[9, 10].

gb_site_nas_sp

A Pentadiagonal Solver benchmark based on the NAS Parallel Benchmarks[9, 10].

gb_site_nas_lu

A LU solver benchmark based on the NAS Parallel Benchmarks [1].

6.1.5. Micro-kernels at the Site level:**gb_grid_lin**

This is an adaptation of the High Performance Linpack benchmark that can run distributed on the grid.

gb_grid_ngb

gb_grid_nas is an implementation of the NAS Grid Benchmarks [6] that are targeted at measuring computational Grids.

6.1.6. Application kernels at the Site and Grid level:**gb_grid_cg_* and gb_site_cg_***

By an initial analysis of the SRS and the design of the Tasks1.1-4 an initial list of computational kernels has been derived, which will form the basis of the gb_grid_cg_* benchmarks.

1. [T1.1] Lattice-Boltzmann kernel
2. [T1.2] Meteorological kernel
3. [T1.2] Hydrological kernel
4. [T1.2] Hydraulic kernel
5. [T1.3] HEP kernel
6. [T1.4] Sea wave models (WAM4,SWAN)
7. [T1.4] Weather prediction kernel

GridBench is not simply a collection of benchmark codes, but also a framework within which users are able to easily conduct benchmarking experiments with convenient access to results. It also provides mechanisms for making benchmarking results available to others services (such as advanced schedulers).

6.2. PROTOTYPE IMPLEMENTATION

For implementation in the scope of the prototype we selected two benchmarks out of the GridBench suite. We implemented these benchmarks and around them we built two “utility” components that are part of the GridBench architecture. The implementation of the prototype can be broken up into the following four parts:

1. Specification of the benchmark definition and implementation of the RSL Generator;
2. Implementation of the `gb_site_hpl` benchmark;
3. Implementation of the `gb_epwhetstone` benchmark;
4. Implementation of the MDS Information Provider.

We implemented the RSL to automatically generate the RSL in order to eliminate the tedious work of manually creating the RSL.

In order to select which benchmarks would be part of the prototype we set the main criterion of a good balance between simplicity and functionality. The benchmarks that were selected, `gb_whetstone` and `gb_site_hpl`, satisfy this criterion. `gb_whetstone` is a CPU micro-benchmark and `gb_site_hpl` is an MPI micro-kernel. More on the two benchmarks can be found in sections 4.4 and 4.5.

The last component included in the prototype implementation is the MDS Information Provider, which performs the task of receiving the XML-encoded benchmark results and publishing them into the MDS directory.

While this set of components is manageable, which was desirable for the purpose of a first prototype, it provides a good cover of the GridBench planned functionality.

Figure 2 depicts the functionality of the prototype. Following the diagram is a description of the prototype functionality.

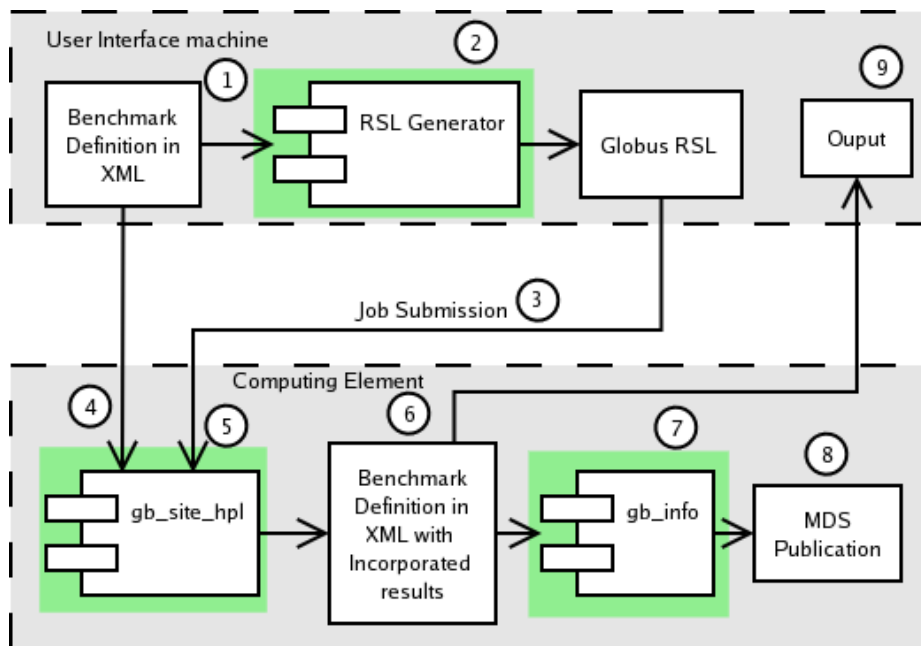


Figure 2: The prototype functionality.

1. The benchmark is specified using an XML-based language (outlined in 4.3);
2. The XML benchmark description is taken as input by the RSL generator which generates RSL;
3. The job (described by the RSL) is submitted to the specified computing element(s);
4. The benchmark executable takes parameter information from the benchmark definition which is bundled with the benchmark job.
5. The benchmark runs to completion;
6. The results of the benchmark are incorporated back into the benchmark definition;
7. The resulting XML document is passed to the GridBench Information Provider (gb_info), which parses it to extract a summary of the results;
8. The information from the XML document is published in the MDS ;
9. The resulting XML document is returned to the User Interface Machine as output.

6.3.SPECIFICATION AND RSL GENERATION

An essential part of GridBench is the specification of benchmark runs. The specification of how a benchmark runs, which includes information of where and when it ran along with the parameters with which it ran, is important for the interpretation of results. To this end the run-time specification is stored together with the results in the same document. The result of this (i.e. keeping the specification and results in the same document) is that there is a strong association between the two. In other words, the specification and the results really start to make sense when used together.

The following diagram and the more detailed DTD after it, outline what is included in the specification of a benchmark run. The diagram and DTD also give information about storing results (i.e. the metrics) and monitoring (not part of the prototype).

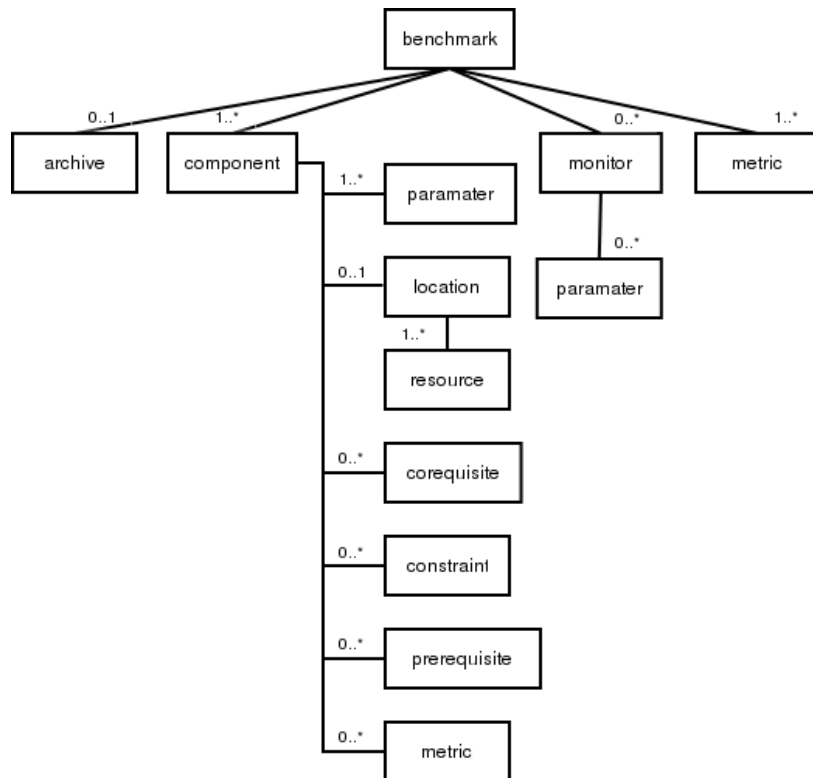


Figure 3: The benchmark definition schema

As shown in Figure 3 a benchmark is made up of:

1. A set of one or more benchmark **component** elements, where each **component** is described by:
 - a. A set of **parameters**, which can be of type *value* or *attribute*. *Attribute* parameters are used directly in the generation of RSL, while *value* parameters are command-line parameters to the benchmark executable itself;
 - b. The **location**, which is made up of at least one **resource**;

- c. A set of **co-requisites**, which defines a set of other components that *must* be running while this component, is running;
 - d. A set of **prerequisites**, which is a set of components that need to have finished before this component can start.
 - e. A set of **constraints** for the runtime of the component. (e.g. minimum amount of RAM \geq 256)
 - f. A set of **metrics** that make up the values of the results of the benchmark component.
2. An **archive** element, which specifies the URL of the native-XML database (namely the *Apache-Xindice* database) that is to store the final XML documents with the incorporated results;
 3. A set of **monitor** elements which specify what monitoring (type, time period etc.) is to be performed by the monitoring component;
 4. A set of **metric** elements that that make up the values of the results of the benchmark as a whole (in contrast with the **metric** elements that are found inside a **component** element which are results of just one component)

When it comes to the generation of RSL from the benchmark description, each **component** element in the XML is transformed into a set of sub-jobs. The target resource is obtained by the **resource** element and the executable is determined by the parameter with name="executable". Since the each of the components has a different scheme for command-line parameters each benchmark executable must have an associated *ParameterHandler_<benchmark-name>* class, which implements the ParameterHandler interface, e.g. *ParameterHandler_gb_hpl*.

Consider the following example, where corresponding parts in the XML definition and the generated RSL have been similarly colored:

XML benchmark definition:

```
<benchmark date="" benchmark_name="nbody">
  <component id="A" name="nbody" type="mpi">
    <location type="single">
      <resource cpuccount="2" name="apelatis.grid.ucy.ac.cy" />
    </location>
    <parameter name="executable" type="attribute">/bin/nbody.exec</parameter>
    <parameter name="nparticles" type="value">1000</parameter>
  </component>
</benchmark>
```

Generated RSL:

```
+ (&(resourceManagerContact="apelatis.grid.ucy.ac.cy")
  (label="subjob 0")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0))
  (count=1)
  (arguments="-n 1000")
  (executable="/bin/nbody.exec" ))
(&(resourceManagerContact="apelatis.grid.ucy.ac.cy")
  (label="subjob 1")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1))
```

```
(count=1)
(arguments="-n 1000")
(executable="/bin/nbody.exec" ) )
```

1. Blue indicates how the CPU-count of “2” triggers the generation of sub-jobs “0” and “1”.
2. Red indicates how the resource specified in the XML (apelatis.grid.ucy.ac.cy) becomes the “resourceManagerContact”.
3. Green shows the translation of *value* parameters to command-line parameters. Since each benchmark executable has different command-line parameters (formats, ordering, etc.) the “arguments” RSL attribute is generated by a module named *ParameterHandler_nbody* (loaded dynamically). When called by the RSL compiler, this module looks for a parameter of type=*attribute* and name=*nparticles*. It then returns the formatted command-line arguments as “-n 1000”

At the moment, RSL generation is rudimentary and just sufficient for generating RSL for running the prototype benchmarks. More examples of RSL generation from a benchmark description are given in sections 7.4 and 7.5.

6.4. MDS INFORMATION PROVIDER

The results of the benchmarks, in the form of metrics, are published into MDS for easy access by users and services. Since this information is available in MDS it can be used in exactly the same way that other MDS information is accessed (e.g. grid-info-search) and with which many Grid users are already familiar.

In this first version of the publication schema, the published metrics are split up into categories with potentially more categories being added as more benchmarks are developed:

1. CE Benchmarks
2. Network Performance

(Since only Computing Element benchmarks were included in the prototype we will focus on those.)

The following is a diagram of the part of the MDS schema that is of interest when it comes to the prototype:

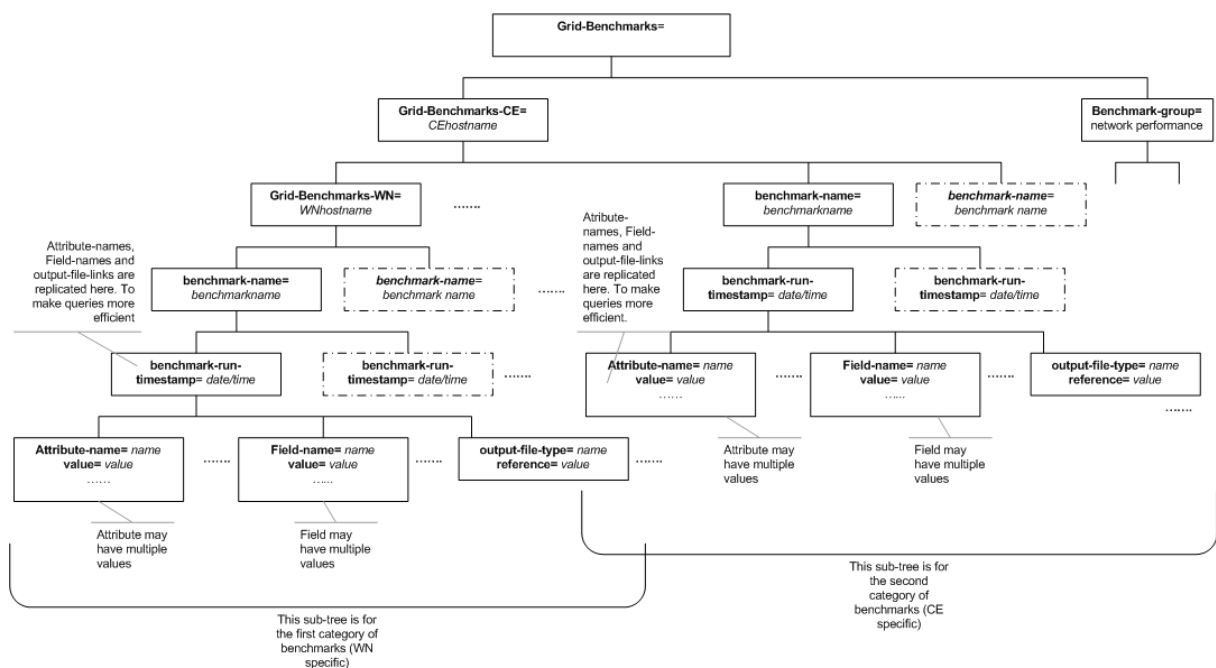


Figure 4: Section of interest of MDS publication schema

The hierarchical schema for publication to MDS, as it is related to the prototype (i.e. only addressing CE and WN benchmarks) is shown in Figure 4. Under the root node of **Grid-Benchmarks** is the category of **Grid-Benchmarks-CE**. Then underneath, as in the architecture of the testbed (Figure 1), is the category of **Grid-Benchmarks-WN**. Each of the **Grid-Benchmarks-CE** and **Grid-Benchmarks-WN** nodes contains a series of benchmarks identified by the benchmark name (benchmark-name=<benchmark name>). Then each run of a specific benchmark is identified by a timestamp (benchmark-run-timestamp=<date/time>).

The component that is responsible for the publication to MDS is the “Information Provider” (`gb_info`). The XML-document, containing the results and other information, is put in a temporary directory from where the Information Provider can pick it up. The Information Provider then takes the resulting XML and generates a LDIF file that is in the format required by the Globus MDS service. This LDIF file is then published in the MDS LDAP directory via the regular mechanisms provided by Globus.

The output of a run of `gb_info` on the results of the `gb_epwhetstone` benchmark is given below. This output contains the result of running the EPWhetstone benchmark on a Computing Element with two worker nodes. The achieved MIPS of each Worker Node are published under its name.

```
dn:hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-name=local,o=grid
objectClass:BenchmarksCE
Mds-Host-hn=apelatis.grid.cs.ucy.ac.cy
```

```
dn:=hn=wn001.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-name=local,o=grid
objectClass:BenchmarksWN
hn=wn001.cs.ucy.ac.cy
```

```
dn:Benchmarks-Benchmark-
Name=EPWetstone,hn=wn001.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-
name=local,o=grid
objectClass:BenchmarksBenchmark
Benchmarks-Benchmark-Name=EPWetstone
```

```
dn:Benchmarks-Start-Time=2002-12-17T16:30:47+02:00,Benchmarks-Benchmark-
Name=EPWetstone,hn=wn001.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-
name=local,o=grid
objectClass:BenchmarksBenchmarkRun
Benchmarks-Start-Time=2002-12-17T16:30:47+02:00
Benchmarks-End-Time=2002-12-17T16:30:47+02:00
```

```
dn:Benchmarks-Attribute-Name=Number of processes ,Benchmarks-Start-Time=2002-12-
17T16:30:47+02:00,Benchmarks-Benchmark-
Name=EPWetstone,hn=wn001.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-
name=local,o=grid
objectClass:BenchmarksAttribute
Benchmarks-Attribute-Name=Number of processes
Benchmarks-Type=int
Benchmarks-Value=235
```

```
dn:Benchmarks-Attribute-Name=Number of repetitions,Benchmarks-Start-Time=2002-12-
17T16:30:47+02:00,Benchmarks-Benchmark-Name=EPWetstone,hn=wn001.cs.ucy.ac.cy,
hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-name=local,o=grid
objectClass:BenchmarksAttribute
Benchmarks-Attribute-Name=Number of repetitions
Benchmarks-Type=int
Benchmarks-Value=30
```

```
dn:Benchmarks-Field-Name=MIPS,Benchmarks-Start-Time=2002-12-17T16:30:47+02:00,
Benchmarks-Benchmark-Name=EPWetstone,hn=wn001.cs.ucy.ac.cy,
hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-name=local,o=grid
```

```
objectClass: BenchmarksOutputField
Benchmarks-Field-Name=MIPS
Benchmarks-Type=int
Benchmarks-Value=21548545
```

```
dn:=hn=wn002.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-name=local,o=grid
objectClass: BenchmarksWN
hn=wn002.cs.ucy.ac.cy
```

```
dn: Benchmarks-Benchmark-
Name=EPWetstone,hn=wn002.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-
name=local,o=grid
objectClass: BenchmarksBenchmark
Benchmarks-Benchmark-Name=EPWetstone
```

```
dn: Benchmarks-Start-Time=2002-12-17T16:30:47+02:00,Benchmarks-Benchmark-
Name=EPWetstone,hn=wn002.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-
name=local,o=grid
objectClass: BenchmarksBenchmarkRun
Benchmarks-Start-Time=2002-12-17T16:30:47+02:00
Benchmarks-End-Time=2002-12-17T16:30:47+02:00
```

```
dn: Benchmarks-Attribute-Name=Number of processes, Benchmarks-Start-Time=2002-12-
17T16:30:47+02:00, Benchmarks-Benchmark-
Name=EPWetstone,hn=wn002.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-
name=local,o=grid
objectClass: BenchmarksAttribute
Benchmarks-Attribute-Name=Number of processes
Benchmarks-Type=int
Benchmarks-Value=235
```

```
dn: Benchmarks-Attribute-Name=Number of repetitions,Benchmarks-Start-Time=2002-12-
17T16:30:47+02:00,Benchmarks-Benchmark-Name=EPWetstone,
hn=wn002.cs.ucy.ac.cy,hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-name=local,o=grid
objectClass: BenchmarksAttribute
Benchmarks-Attribute-Name=Number of repetitions
Benchmarks-Type=int
Benchmarks-Value=30
```

```
dn: Benchmarks-Field-Name=MIPS,Benchmarks-Start-Time=2002-12-17T16:30:47+02:00,
Benchmarks-Benchmark-Name=EPWetstone,hn=wn002.cs.ucy.ac.cy,
hn=apelatis.grid.cs.ucy.ac.cy,Mds-Vo-name=local,o=grid
objectClass: BenchmarksOutputField
Benchmarks-Field-Name=MIPS
Benchmarks-Type=int
Benchmarks-Value=32358545
```

7.USER MANUAL

7.1.DEPENDENCIES

Having minimal dependencies for this prototype is important. There are two dependencies:

1. A dependency on BLAS (Basic Linear Algebra Subprograms) to which `gb_site_hpl` must be linked. More precisely, the C interface (`libcblas`) is currently required.
2. A working MPI environment.

7.2.INSTALLATION

`Gb_site_hpl` and `gb_epwhetstone` are installable by RPM:

```
rpm -Uvh gridbench-0.1.i386.rpm
```

If the binary rpm is not available it can be built from source:

```
cd <path>/wp2.3-bench  
make rpm
```

The RPM is to be included in the next CG release and automatically installed in the User Interface machines. The code for the prototype is currently in the CrossGrid CVS: cvs.fzk.de.

7.3.RUNNING

In order to run the Gridbench benchmarks, the user must

1. Create an input XML document based on the benchmark definition DTD (given in Appendix A);
2. Compile the XML description into a RSL job specification using the *RSLCompile*;
3. Submit the RSL as a regular Globus job.

Sections 7.4 and 7.5 describe the process for each of the prototype benchmarks.

7.4.GB_EPWHETSTONE

The `gb_epwhetstone` benchmark (which is based on the `whetstone` benchmark, [9]) has been selected to serve as part of the prototype. This decision was based on the fact that this benchmark is simple and it poses as a good example of a micro-benchmark. The decision was also based on the fact that a lot of

the source code of the application is available and thoroughly tested. The code was extended so that it could be run as an MPI application, which measures the raw CPU power of the machines it is running on. Another reason for selection is that this benchmark can be used with no dependency, with the exception of MPI, on grid middleware.

The `gb_epwhetstone` benchmark is an “embarrassingly parallel” MPI application (hence the “ep” in “epwhetstone”), with a very simple user interface. It takes as input an XML file (the benchmark definition), which conforms to the DTD given in the previous section. The output is a simple metric, *Million Instructions Per Second*, and it is reported separately for each worker node in which it run and as a total. Usage of `gb_epwhetstone` is quite simple:

1. The user specifies the parameters of the benchmark (number of loops and number of processors) via the benchmark definition XML file. Also specify the target resource (Computing Element) in the benchmark definition.
2. The benchmark definition is passed to the RSL generator which gives an RSL file:

```
$ java GenRSL < gb_epwhetstone.xml > gb_epwhetstone.rsl
```
3. The job is submitted via the regular Globus job submission mechanisms:

```
$ glubusrun -o -f gb_epwhetstone.rsl
```

The following is the benchmark definition XML document *with the incorporated results*:

Please note the inline comments that describe the parts of the benchmark definition.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE benchmark SYSTEM "benchmark.dtd">
<!-- The following is a specification for the epwhetstone benchmark -->
<!--The benchmark ran at 17:45 on 1/4/2003, the benchmark name is
epwhetstone-->
<benchmark date="20030401174523" benchmark_name="epwhetstone">
<!-- The result is to be stored in a xindice database at archive.grid.ucy.ac.cy on
port 4080 -->
<archive type="xindice">archive.grid.ucy.ac.cy:4080</archive>
<!-- There is a single mpi component in the benchmark -->
<component id="A" name="epwhetstone" type="mpi">
  <location type="single">
    <!-- the benchmark will run on a single Computing Element-->
    <resource cpucount="5" name="apelatis.grid.ucy.ac.cy" />
  </location>
<!--parameters of type "attribute" are used during the generation of the RSL-->
<parameter name="executable" type="attribute">epwhet</parameter>
<!--parameters of type "value" or "list" are used as parameters to the benchmark-->
<parameter name="nloops" type="value">10000</parameter>
<!--the component produces one metric "MIPS" which is a list of the number-->
<!-- easured at each distinct node.-->
<metric name="MIPS" type="list">845,844,844,840,844</metric>
<!-- Additional information the "MIPS" metric.-->
<!-- This gives the node hostname for each value given in the above metric-->
```

```
<metricspec metric="MIPS">wn003.grid.ucy.ac.cy, wn004.grid.ucy.ac.cy,
wn02.grid.ucy.ac.cy, wn001.grid.ucy.ac.cy, wn001.grid.ucy.ac.cy </metricspec>
</component>
<!--The metric produced by the benchmark is "total_MIPS-->
<!--It is a sum of the MIPS measured at each node-->
<metric name="total_MIPS" type="value">4203</metric>
</benchmark>
```

The following is the RSL that was generated from the benchmark definition included in the previous XML document. (Note that the document given above includes the benchmark results along with the benchmark definition; i.e. the XML document is shown in its “final form”, after the results have been incorporated into it.)

```
+
(&(resourceManagerContact="apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs")
 (count=1)
 (label="subjob 0")
 (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
 (GLOBUS_GRAM_JOB_CONTACT
 apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs))
 (arguments= "10000")
 (executable="./gb_epwhet")
&(resourceManagerContact="apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs")
 (count=1)
 (label="subjob 1")
 (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
 (GLOBUS_GRAM_JOB_CONTACT
 apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs))
 (arguments= "10000")
 (executable="./gb_epwhet ")
&(resourceManagerContact="apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs")
 (count=1)
 (label="subjob 2")
 (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
 (GLOBUS_GRAM_JOB_CONTACT
 apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs))
 (arguments= "10000")
 (executable="./gb_epwhet ")
&(resourceManagerContact="apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs")
 (count=1)
 (label="subjob 3")
 (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
 (GLOBUS_GRAM_JOB_CONTACT
 apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs))
 (arguments= "10000")
 (executable="./gb_epwhet ")
&(resourceManagerContact="apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs")
 (count=1)
 (label="subjob 4")
 (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
 (GLOBUS_GRAM_JOB_CONTACT
```

```

        apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs))
(arguments= "10000")
(executable="./gb_epwhet ")
)

```

7.5.GB_SITE_HPL

The `gb_site_hpl` benchmark (which is based on the HPL benchmark, [4]) has been selected to serve as part of the prototype. This decision was based on the fact that this benchmark is neither overly complex nor overly simplistic. The decision was also based on the fact that most of the source code of the application (with the exception of the interfaces) is available and quite thoroughly tested (HPL is quite a popular benchmark). Another reason for selection is that this benchmark can be used with no dependency, with the exception of MPI, on grid middleware.

The `gb_site_hpl` benchmark is a monolithic, parallel (MPI) application, with a fairly simple user interface. It takes as input an XML file (the benchmark definition), which conforms to the DTD given in the previous section. The output is again an XML file that contains two simple metrics: *FLOPS* and *completion time*. Using `gb_site_hpl` is quite simple:

1. The user specifies the parameters of the benchmark (problem size, algorithm details, number of processors etc.) via the benchmark definition XML file. Also specify the target resource (Computing Element) in the benchmark definition.
2. The benchmark definition is passed to the RSL generator which gives an RSL file:

```
$ java GenRSL < gb_hpl.xml > gb_hpl.rsl
```
3. The job is submitted via the regular Globus job submission mechanisms:

```
$ glubusrun -o -f gb_hpl.rsl
```

The following is the benchmark definition XML document *with the incorporated results*:

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE benchmark SYSTEM "benchmark.dtd">
<benchmark date="20030401145412" benchmark_name="gb_hpl">
  <component id="A" name="hpl" type="mpi">
    <location type="single">
      <resource cpucount="2" name="apelatis.grid.ucy.ac.cy" />
    </location>
    <parameter name="outfile" type="value">HPL.out</parameter>
    <parameter name="problemsize" type="value">1000</parameter>
    <parameter name="blocksize" type="value">64</parameter>
    <parameter name="processgrid" type="value">1,2</parameter>
    <parameter name="threshold" type="value">16.0</parameter>
    <parameter name="PFACT" type="value">0</parameter>
    <parameter name="NBMIN" type="value">8</parameter>
  </component>
</benchmark>

```

```

<parameter name="NDIV" type="value">2</parameter>
<parameter name="RFACT" type="value">2</parameter>
<parameter name="BCAST" type="value">1</parameter>
<parameter name="DEPTH" type="value">1</parameter>
<parameter name="SWAP" type="value">2</parameter>
<parameter name="swapthreshold" type="value">80</parameter>
<parameter name="L1" type="value">0</parameter>
<parameter name="U" type="value">0</parameter>
<parameter name="equilibration" type="value">0</parameter>
<parameter name="mem_align" type="value">8</parameter>
<parameter name="executable" type="attribute">gb_hpl</parameter>
<metric name="FLOPS" type="value">55930000</metric>
<metric name="completion_time" type="value">11.95</metric>
</component>
<component id="B" name="hpl" type="mpi">
  <location type="single">
    <resource cpucount="2" name="apelatis.grid.ucy.ac.cy" />
  </location>
  <parameter name="outfile" type="value">HPL.out</parameter>
  <parameter name="problemsize" type="value">2000</parameter>
  <parameter name="blocksize" type="value">64</parameter>
  ...
  <parameter name="executable" type="attribute">gb_hpl</parameter>
  <prerequisite id="A" />
  <metric name="FLOPS" type="value">60760000</metric>
  <metric name="completion_time" type="value">10.98</metric>
</component>
<metric name="best_FLOPS" type="value">55930000</metric>
<metric name="best_completion_time" type="value">11.95</metric>
</benchmark>

```

The following is the RSL that was generated from the above benchmark definition.

```

+
(&(resourceManagerContact="apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs")
 (count=1)
 (label="subjob 0")
 (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
  (GLOBUS_GRAM_JOB_CONTACT
   apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs))
 (arguments= "./xhpl.dat")
 (executable="./gb_site_hpl")
&(resourceManagerContact="apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs")
 (count=1)
 (label="subjob 1")
 (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
  (GLOBUS_GRAM_JOB_CONTACT

```

```
        apelatis.grid.ucy.ac.cy:2119/jobmanager-pbs))  
(arguments= "./xhpl.dat")  
(executable="./gb_site_hpl")  
)
```

8.INTERNAL TESTS

Testing of the first prototype during and after month 12 has shown that:

- The functionality of the implemented benchmarks, as it is outlined in Annex 1, is in place.
- The measurements produced by the benchmark prototypes generally agree with the expected values. The metrics obtained for the test-bed resources are consistent with results obtained, by us or other parties, on resources outside the test-bed.
- Specifying jobs and executing them on the test-bed is rather tedious and not user friendly. Given the current submission mechanisms and state of MPI support on the test-bed, need arises for the automated generation of RSL or JDL. GridBench must have full control over the generated RSL or JDL.

9. ISSUES

The development for the first prototype and the integration process went generally smooth. While there were some obstacles to be overcome and there were no “show-stoppers”

This section outlines some of the issues that were met during development and integration.

1. Problems related to MPI affect all CrossGrid Tasks that use MPI. We describe here the problems that were met before and during the integration, that are related to MPI.
 - a. *Availability of installed MPI libraries.*

Since MPI is not part of the EDG/CrossGrid distribution, not all sites have MPI installed (in fact, very few do). The work around this was to use statically-linked executables. A development environment with the correct version of the MPICH (Globus-enabled) software is needed in order to produce statically-linked executables. One of the downsides of this solution is that the resulting executables are very large (e.g. `gb_site_hpl` executable was ~1.5MB vs. the ~8MB of the statically linked version.)

Furthermore statically linked executables cannot use *optimized* libraries that may be installed on a system.
 - b. *Support of MPI Jobs by JDL*

Since the current version of the CrossGrid/DataGrid middleware does not provide support for MPI applications, we have had to resort to generating plain Globus RSL. While this allows us to run our benchmarks it circumvents most of the CrossGrid middleware like the Resource Broker, with the possibility of creating scheduling conflicts, incorrect accounting etc.
2. *JDL Vs RSL for job-description.*

In addition to the JDL MPI-related problem stated above, we have met the problem were RSL is inadequate for describing complex runs of several components the execution of which must be ordered. (This functionality is in development for JDL, as it was determined through inter-task meetings that took place during the integration meeting in February 2003). The problem that arises is that some required functionality is provided by one language and not by the other, so there is not one language that covers all of the requirements of running benchmarks. Development (especially of JDL) will cover the requirements (MPI + ordered execution of multiple components) but solutions must be found for the interim period.
3. While the availability of *optimized* numerical libraries (such as the BLAS routines used by `gb_site_hpl`) does not hinder development in any way, in production environments the actual metrics obtained by the benchmarks may be skewed. (For example, the BLAS library used in the CrossGrid testbed is compiled for i386 whereas most resources in the testbed are i686.)

10.APPENDIX A

This is the DTD to which all GridBench benchmark run specifications must conform.

```

<?xml version='1.0' encoding='UTF-8'?>

<!ELEMENT benchmark (archive?,component+,metric*,metricspec*,monitor*)>
<!--ATTLLIST benchmark
  date CDATA #REQUIRED
  benchmark_name CDATA #REQUIRED -->

<!ELEMENT resource (node)* >
<!--ATTLLIST resource
  cpucount CDATA #REQUIRED
  name CDATA #REQUIRED -->

<!ELEMENT component
(location,(parameter|constraint|prerequisite|corequisite|metric|metricspec|monitor)*)>
<!--ATTLLIST component
  id ID #REQUIRED
  name CDATA #REQUIRED
  type (mpi|simple|util) #REQUIRED -->

<!ELEMENT corequisite EMPTY>
<!--ATTLLIST corequisite
  id IDREF #REQUIRED -->

<!ELEMENT constraint (#PCDATA)>
<!--ATTLLIST constraint
  name CDATA #REQUIRED -->

<!ELEMENT prerequisite EMPTY>
<!--ATTLLIST prerequisite
  id IDREF #REQUIRED -->

<!ELEMENT location (resource)*>
<!--ATTLLIST location
  type (undef|single|multi) #REQUIRED
  assigned (manually|broker) #IMPLIED -->

<!ELEMENT node (#PCDATA)>

<!ELEMENT parameter (#PCDATA)>
<!--ATTLLIST parameter
  name CDATA #REQUIRED
  type (value|list|attribute|attributelist) #REQUIRED -->

<!ELEMENT metric (#PCDATA)>
<!--ATTLLIST metric
  name CDATA #REQUIRED
  type (value|list|reference) #REQUIRED
  id ID #IMPLIED-->

<!ELEMENT metricspec (#PCDATA)>
<!--ATTLLIST metricspec
  metric_id IDREF #REQUIRED
  dt CDATA #IMPLIED-->

<!ELEMENT monitor (parameter)*>
<!--ATTLLIST monitor
  type CDATA #REQUIRED -->
<!ELEMENT archive (#PCDATA)>
<!--ATTLLIST archive
  type CDATA #REQUIRED
  comp IDREF #IMPLIED -->

```