

Design and Implementation of GridBench^{*}

George Tsouloupas Marios D. Dikaiakos
{georget,mdd}@ucy.ac.cy

Department of Computer Science
University of Cyprus
1678 Nicosia, Cyprus

Abstract. In this article we present the software architecture and implementation of GridBench, an extensible tool for benchmarking and testing grid resources. We give an overview of GridBench services and tools, which support the easy definition, invocation and management of benchmarking experiments. We also show how the tool can be used in the analysis of benchmarking results and how the measurements can be used to complement the information provided by Grid Information Services and used as a basis for resource selection. In order to illustrate the usage of the tool, we describe scenarios for using the GridBench framework to perform benchmarking experiments and analyze the results.

1 Introduction

Grids are emerging as wide-scale, distributed infrastructures that comprise heterogeneous computing and storage resources and seek to support resource sharing in dynamic, multi-institutional Virtual Organizations (VOs). VO members make use of various tools and services provided by grid middleware in order to describe their computations, to receive authorization and allocate resources, and to dispatch jobs. The effective identification of resources upon which to dispatch a job, however, can be a difficult task in the presence of too many, heterogeneous, frequently changing resources. To this end, it is essential to have access to accurate and descriptive information about the performance of these resources.

Benchmarking represents an effective approach for producing a reliable performance characterization of grid resources [7]. Benchmarking is essential for the performance-based resource selection either by end-users or automated decision-makers, such as resource brokers and schedulers. Besides measuring performance, benchmarks can also be used as “end-to-end” tests for grid resources by end-users or administrators, ensuring that resources are available and functional. However, with the emergence of large grid infrastructures, such as CrossGrid [5] or EGEE [1], the administration of benchmarking experiments and the management of their results becomes a tedious and complex task. Moreover, some of the key aspects of grids, i.e., lack of central control, resource virtualization,

^{*} This work was supported in part by the European Union through the CrossGrid project (contract IST-2001-32243).

co-allocation of resources, and cross-platform interoperability, raise several challenges related to the proper interpretation of benchmarking metrics.

In this paper, we describe GridBench, an extensible framework for benchmarking and testing grid resources. We give an overview of GridBench services and tools, which support the easy definition, invocation, and management of benchmarking experiments. We show how the tool can be used in the analysis of benchmarking results and how its measurements can be used to complement the information provided by Grid information services and used as a basis for resource selection. The remainder of this article is organized as follows: Section 2 provides an overview of GridBench. Sections 3 and 4 describe in more detail the software architecture of GridBench. Section 5 presents briefly two illustrative use-case scenarios implemented with GridBench. We conclude in Section 6.

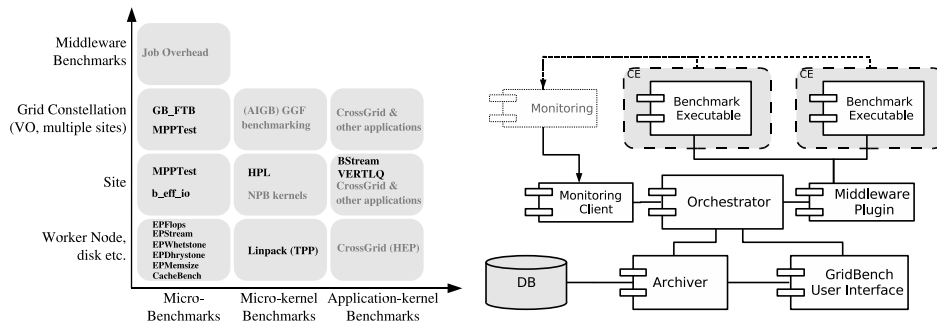
2 GridBench Overview

GridBench assumes an underlying hardware infrastructure consisting of a set of geographically distributed, heterogeneous sites connected over a shared network (e.g. the Internet) and participating to one or more Virtual Organizations (VOs). Each site contains a Computing Element (CE) which manages a set of Worker Nodes for performing computations. Typically a CE is associated with a Storage Element (SE), which is an interface to mass storage, and to which the CE has Local-Area-Network access (e.g. via the Network File System). The grid VO also contains some central VO services, such as a Grid Information Service, a resource broker, a VO membership server etc.

GridBench was designed with two main objectives in mind: i) To generate metrics that characterize the performance capacity of resources belonging to a Virtual Organization and *span across multiple grid nodes*, in terms of computational power, file-transfer speed, inter-process communication bandwidth, application-kernel performance, scalability etc. ii) To provide a *tool* for end-users that wish to investigate various aspects of grid performance, using well-understood kernels that are representative of more complex applications deployed on grids. 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, scheduling algorithms, resource allocation, etc.

To meet these objectives, Gridbench has two constituents: the *GridBench Benchmark Suite* and the *GridBench Framework*. The GridBench suite is a *layered* collection of new and pre-existing micro-benchmarks, micro-kernels, and application benchmarks (see Figure 1(a)). These benchmarks were carefully chosen and tuned in order to generate a comprehensive set of metrics that characterize the performance of grid resources concisely. More details on the Gridbench suite can be found in [8].

Figure 1(b) outlines the software components that comprise the GridBench Framework and provide the necessary functionalities for defining and running benchmarks as well as for archiving, retrieving and analyzing benchmarking-experiment results. The design of the GridBench Framework facilitates the port-



(a) The GridBench Suite layered approach to benchmarking, with micro-benchmarks, micro-kernel benchmarks and application benchmarks on the x-axis, and resource, site and grid constellation on the y-axis.

(b) Key *GridBench Framework* components: Orchestrator, Archiver, GridBench User Interface.

Fig. 1. GridBench constituents: The *GridBench Suite* and the *GridBench Framework*.

ing of GridBench to different grid middleware platforms. To this end, it concentrates all platform-specific functionalities in *middleware* and *monitoring plugins*, which can be easily replaced without affecting the design of other components. In the current GridBench version, we have implemented plugins for Globus 2 [2] and CrossGrid [5]/LCG2 [3]. Furthermore, GridBench uses Globus MDS [2] in order to retrieve information about the available resources, and either the Globus2 GRAM or the EU CrossGrid Resource Broker [4] for job submission. The GridBench Framework is comprised of web services and user interfaces implemented in Java and Tomcat. MySQL is used as the back-end database. The following sub-sections describe in more detail the key components of the GridBench Framework.

3 The GridBench Definition Language (GBDL)

The GridBench Definition Language (GBDL) is an XML-based language that encodes basic information required to describe and execute a benchmark. Moreover, it supports the annotation of a benchmark definition with performance-related metadata representing the conditions of a particular benchmarking experiment and the metrics derived from that experiment. We designed GBDL instead of using an existing job description language, such as RSL or JDL, in order to: i) allow for a standardized definition of benchmarks that is independent of the underlying middleware platforms used to execute the benchmarks; ii) enable the specification of the monitoring information that should be collected during a benchmark execution from an available grid monitoring system; and iii) serve as a container for associating benchmark definitions to the resulting metrics and to the collected monitoring data. By associating these three pieces of information,

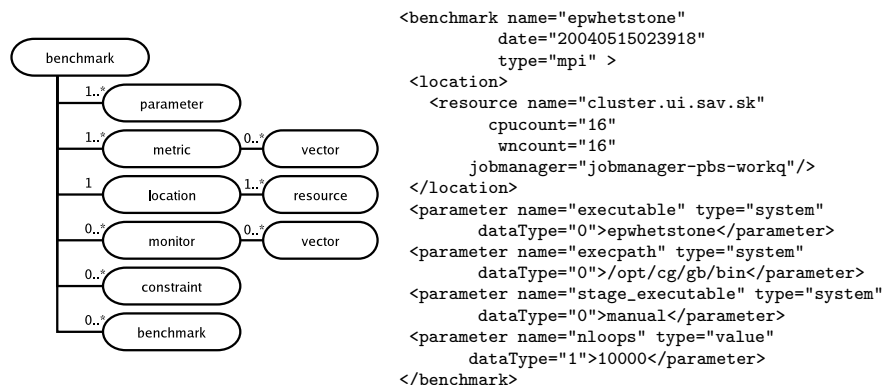


Fig. 2. Left: A schematic overview of GBDL; shown in rounded boxes are the main parts of a GBDL document. Right: A real example of GBDL definition, showing an *epwhetstone* benchmark definition that uses MPI on 16 CPU’s at *cluster.ui.sav.sk*, using one CPU on each Worker Node.

we can take into consideration the conditions of the infrastructure upon which a benchmarking experiment is executed and, thus, derive accurate conclusions about the performance capacity of examined resource [8].

Figure 2 (left) provides a high-level, schematic view of the GridBench Definition Language. The benchmark definition includes two families (types) of *parameters*: *system* parameters, which specify details for the benchmark execution, such as the path to the executable, and *value-type* parameters, which are benchmark-specific. The *location* element contains a set of *resource* elements, where each resource element specifies the name of a grid site, the number of CPU’s to be requested from that site, and how the CPU’s should be distributed on that site’s Worker-Nodes (*cpucount* and *wncount*, respectively). The resource *name* attribute is optional; its absence signals that the underlying middleware should allocate resources to run the benchmark. *Monitor* elements contain “instructions” of what to monitor during benchmark execution. A monitor element contains a monitoring system-dependent query, and a specification of which infrastructure monitoring system to use.

GBDL definitions can specify simple or composite benchmarks, i.e., benchmarks consisting of multiple components. This, in conjunction with the use of the execution *constraint* elements, can be used to specify simple workflows. *Constraint* elements can be of type *prerequisite* or *corequisite* and denote dependencies between components. For example, a benchmark involving three component benchmarks, *comp 1*, *comp 2* and *comp 3*, where *comp 2* needs to start after *comp 1* has finished, and *comp 3* must run concurrently with *comp 2* but after *comp 2* starts, would be specified like this:

```

<benchmark name="flow">
  <benchmark id="comp 1"/>
  <benchmark id="comp 2">
    <constraint type="prerequisite" id="comp 1">
  <benchmark id="comp 3"/>
    <constraint type="corequisite" id="comp 2">
  </benchmark>}

```

GBDL grammar defines also an element to describe performance *metrics*, which are computed at run-time by instrumentation probes inserted in the benchmark code. Metric elements are represented either as single attribute-value pairs or as vectors of attribute-value pairs (for instance, bandwidth values at different packet-sizes).

4 GridBench Framework Core Components

4.1 Orchestrator Web-service

The *Orchestrator* component has the task of managing benchmark executions. Most of the Orchestrator's functionality is implemented in the form of *middleware plugins*; plugins for Globus and the EDG/LCG middleware are provided. These plugins mainly deal with i) job description creation (e.g. RSL for Globus), ii) job submission and job status monitoring, iii) file staging and iv) result retrieval. When a new benchmark description (in the form of GBDL) is delivered to the *Orchestrator* web service for execution, the GBDL is translated to the job description language required by the underlying middleware. All specified monitoring data collection is initiated and the job is submitted. When the job finishes, it's output (the metrics) are incorporated into the benchmark definition, as well as all the collected monitoring data. The *location* element of GBDL is usually a part of the definition of the benchmark and included in the GBDL document that the Orchestrator receives. If it is not provided and the underlying middleware used supports resource brokering [5, 4] then the *location* is determined and instantiated after job completion by querying the underlying middleware. This is useful in cases where the user is not interested in specifying the exact set of resources to run the benchmark and allows the middleware to automatically allocate resources. The Orchestrator web service provides the following interface:

jobid **execute**(*GBDL,method,cred*): execute the *GBDL* benchmark definition using the middleware plugin identified by *method* using the credentials *cred* for authentication, returning a job identifier.

status **jobStatus**(*jobid*): return the job status of the job identified by *jobid*. The diagram in Figure 3 describes the Orchestrator functionality in a series of steps. The steps are given below (the numbers correspond to the circled items in the diagram): **(1)** The Orchestrator receives a benchmark description in the form of GBDL. This will originate from the GridBench GUI or from an automated system performing automated/periodic executions; **(2,3)** The GBDL is translated to a middleware-specific job description via a *Middleware Plugin*, which is in the syntax and format required by the underlying middleware; **(4)** The Orchestrator determines all monitoring that needs to be performed, which is specified by the *monitor* element(s) of the GBDL. Using the *type* and *query* attributes of the *monitor*, the correct monitoring plugin is invoked. **(5)** Monitoring data collection is started. In the event where the benchmark is put in the

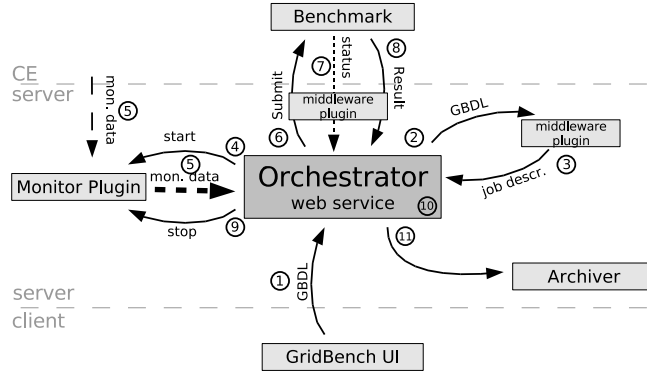


Fig. 3. Diagram describing the Orchestrator functionality

target-resource’s local queue, synchronization of monitoring data collection and the actual benchmark execution is performed by job-status monitoring; (6) The benchmark job is then submitted using the *Middleware plugin*; (7) The benchmarking job status is monitored either by an “in-process wait” or by polling; (8) The benchmark job finishes and the result (i.e. the standard output containing the **metrics**) is returned to the Orchestrator by the *Middleware Plugin*; (9) The *Monitoring Plugin* is then signaled to stop collecting monitoring data and the collected data is returned to the Orchestrator; (10) The results of the benchmark in the form of *metric* elements and its associated monitoring data are incorporated into the original GBDL. If the *resources* specified in the the *location* element were not specified explicitly (i.e. resources were allocated by the system) then location element is also updated; (11) Finally, the resulting GBDL is passed to the Archiver, concluding the Orchestrator’s role as it relates to this specific benchmark.

4.2 Archiver Web-service

The *Archiver* allows the storage and retrieval of results generated by executions of the GridBench Benchmark Suite through the Gridbench Framework. The *Archiver* was introduced in order to: (i) manage a potentially large number of results depending on the size of the Grid under study, the number of benchmarks and the frequency of their execution; (ii) provide a central repository for the results allowing access to measurements for users or Grid services; and (iii) hold a set of *benchmark models*, which serve as customizable benchmark definitions.

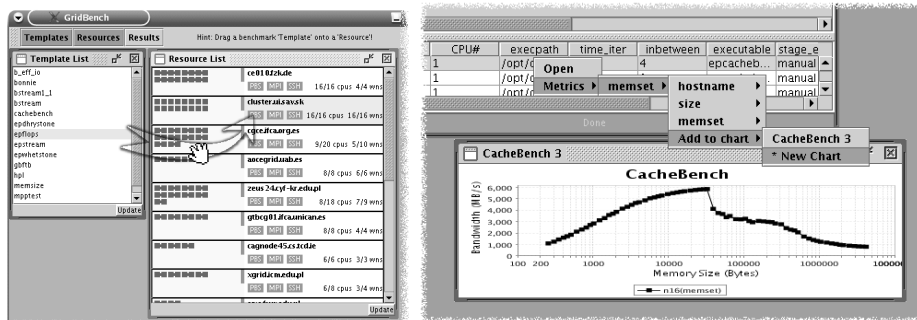
The Archiver is implemented as a web service and provides the following interface:

docid **archive**(*GBDL*): put document *GBDL* in the database and return the numeric document identifier *docid*.

GBDL **getBenchmark**(*docid*): retrieve document *docid* from the database and return it in *GBDL* form.

docidList **getDocList**(*benchName, resName*): retrieve the list of documents *docid* containing results for benchmark *benchName* on resource *resName*.

docidList **getModelList**(): retrieve the list of numeric document id’s.



(a) Benchmarks and Resources.

(b) Analysis tools.

Fig. 4. Screen-shots of the GridBench graphical user interface. *Left:* The list on the left is a list of benchmarks that are integrated into GridBench. The list on the right shows the currently available resources and their status in terms of busy/free CPU's. *Right:* Generation of charts from historical data. The result shown is from a cache benchmark.

The MySQL database schema (which is the back-end for the Archiver), contains tables for each element type in the GBDL document. Namely, it includes the tables: *benchmarks*, *parameters*, *metrics*, *metric_data*, *monitors*, *monitor_data* and *resources*. GBDL document structure integrity during transformation from XML to the relational form is guaranteed by the use of foreign keys in the database tables.

4.3 The GridBench User Interface

GridBench provides a user-friendly graphical interface for defining and executing benchmarks, as well as browsing results. Additionally, it provides tools for result analysis through the easy construction of custom graphs from archived results. Figure 4(a) shows the main graphical user interface for the definition of benchmarks, where we can observe the list of available benchmarks (the list on the left) and the available resources (the list on the right). The resource list shows resources retrieved from one or more Grid Information Systems (MDS), with details about each resource's composition, such as free/busy CPU's and Worker nodes, dual/single CPU machines etc. Additionally a set of tests can be performed on each resource. In Figure 4(a) we can see tests such as the "PBS" test and the "MPI" tests. These tests will test each resource for correct configuration of PBS and MPI respectively. Tests involving multiple sites (e.g. using MPICH-G2) can also be performed. Such tests are usefull for detecting configuration problems as well as connectivity/firewall issues. More tests (e.g. targetting other local queuing systems) can be easily added by implementing simple Java interfaces.

Defining and executing a benchmark is as easy as dragging a benchmark onto one of the resources (shown in Figure 4(a)). The user has the opportunity to tune the benchmark parameters prior to execution via a benchmark configuration

panel. The user can easily construct graphs as the ones in the results section by using the “result matrix” shown in Figure 4(b).

5 Use-case scenarios

We present 2 simple use-case scenarios for GridBench in order to illustrate the functionality visible to the end-user and the overall simplicity of using the tool to get performance metrics for Grid resources. Many other use-case scenarios are possible, such as a scheduler that performs resource ranking on an application-kernel performance basis in a way that is completely transparent to the user.

5.1 Use-case Scenario 1: Comparing resources

As a first use-case, we consider a user who wants to compare a set of resources in terms of 2 “basic” performance factors : CPU FLOP/s and memory bandwidth. The user would like to use “fresh” data so she opts to invoke new benchmark executions instead of fetching historical data. The user can perform the following steps: i) Determine which metrics will tell the user what she wants to know about the resources (e.g. CPU performance and memory bandwidth). ii) Using the GridBench GUI, the user simply drags each of the benchmarks onto each resource and submits the benchmark (Figure 4(a)). When the benchmark execution finishes, the result will be automatically archived. iii) Using the GridBench GUI the user puts together comparative charts for the resources for each benchmark (Figure 5).

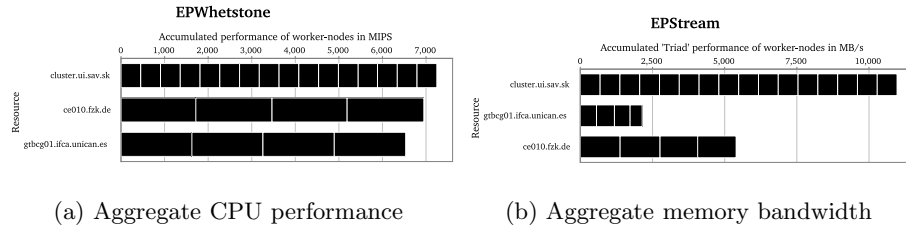


Fig. 5. Results for use-case scenario 1.

From the results on Figure 5 (the charts were generated using the GridBench GUI) we observe that the three sides that were chosen for comparison vary in their measurements. At this point it is important to note that two of the resources (`ce010.fzk.de` and `gtbcg01.ifca.unican.es`) use dual-CPU worker-nodes. In terms of aggregate CPU performance they vary only slightly. In terms of memory bandwidth the performance varies greatly as shown in figure 5(b) (probably due to the memory technology employed at each resource). Considering a memory-intensive application where the main requirement is memory bandwidth then a user (or resource broker) can select the four worker nodes from `ce010.fzk.de` rather than the four from `gtbcg01.ifca.unican.es`.

5.2 Use-case Scenario 2: Application Performance

As a second use-case we consider a user who wants to compare resources based on performance of a given application or kernel ¹. The user needs to find the best resources to run a set of simulations. One of the primary design goals of the GridBench framework is the easy inclusion of new benchmarks/kernels. In this use-case scenario the user wishes to include a frequently used kernel; the required steps are: i) Create a new GBDL description (model) and add it to the Archiver database; iii) Instrument the code of the kernel to generate metrics. The following is the new GBDL description rewired to run the instrumented kernel:

```
<benchmark name="bstream1_1" date="" type="mpi" model="true" >
  <parameter name="executable" type="system">bstream1.1</parameter>
  <parameter name="iterations" type="value">40</parameter>
  <parameter name="Reynolds" type="value">20</parameter>
  <parameter name="data_id" type="value">tube38x40x40</parameter>
  <parameter name="stage_file" type="system">tube38x40x40.bs</parameter>
</benchmark>
```

This description specifies the *executable* and files to be staged, and sets the *iterations*, *Reynolds* and *data_id* application-specific parameters.

The Instrumentation of codes is highly application-specific and usually involves trivial modification of the source code to obtain timings at a high level. In this specific use-case, the application performs iterations which are controlled by a main loop. In total, about ten lines of code were added in order to time each iteration and output the following metrics onto the standard output:

```
<metric name="iteration_times" type="vector" unit="s" step="20" period="200">
  <vector name="time">0.079617 0.079529 0.079511 0.079498 ... 0.094326</vector>
</metric>
<metric name="completion_time" type="value" unit="s">639.633215</metric>
```

Once the kernel has been integrated into GridBench the user can invoke it as in the previous use-case. The steps are now: i) Retrieve archived results for this kernel; ii) Benchmark the resources for which there are no archived results; iii) Compare the results (Figure 6).

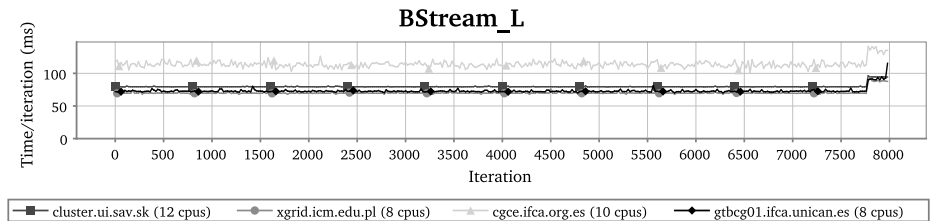


Fig. 6. Results for use-case 2, showing iteration times of a given kernel on four resources.

¹ The kernel in question is extracted from a medical application, developed at the Univ. of Amsterdam, for pre-operative planning of vascular reconstruction. It involves blood-flow simulation using a Lattice Boltzmann method in 3-D artery models [6].

6 Conclusions and Future Work

We have provided an overview the GridBench services and user interface which can serve as a “virtual workbench” for performing benchmarking experiments, archiving benchmark specifications and results, and as an aid for analysis of metrics. We have also presented two use-case scenarios and illustrated the functionality and ease of use of the tool: The first use-case illustrated how end-users and administrators can perform benchmarking experiments either for resource selection or for determining the operational status of resources. The second use-case illustrated how a user or application developer can obtain results from new application-based benchmarks using the GridBench framework.

In on-going and future work we are working on the implementation of more benchmarks focusing on the aspects of availability and performability and the derivation of higher-level metrics to express “quality features” of Grid infrastructures: homogeneity, trustworthiness of GIS, health of the infrastructure, reliability and robustness. We also plan to enrich the Gridbench suite with more benchmarks based on existing Grid applications. Finally, we are working on extending the GBDL specification to include constrained and automatic parameter selection and to include additional middleware plugins to provide interoperability with more infrastructures (such as UNICORE).

References

1. EGEE: Enabling Grids for eScience in Europe. <http://www.eu-egee.org>, (accessed Oct. 2004).
2. Globus project. <http://www.globus.org>, (accessed Sept. 2004).
3. Large Hadron Collider Computing Grid (LCG). <http://lcg.web.cern.ch>, (accessed Oct. 2004).
4. Elisa Heymann, Miquel A. Senar, Enol Fernandez, Alvaro Fernandez, and Jose Salt. Managing MPI Applications in Grid Environments. In M.D. Dikaiakos, editor, *Grid Computing. Second European AcrossGrids Conference, AxGrids 2004, Nicosia, Cyprus, January 2004, Revised Papers*, volume 3165 of *Lecture Notes in Computer Science*, pages 42–50. Springer, 2004.
5. J. Marco and R. Marco et al. First Prototype of the CrossGrid Testbed. In F. Rivera, M. Bubak, A. Gomez-Tato, and R. Doallo, editors, *Grid Computing. First European AcrossGrids Conference. Santiago de Compostella, Spain. February 2003. Revised papers*, volume 2970 of *Lecture Notes in Computer Science*, pages 67–77. Springer, 2004.
6. P.M.A. Sloat, A. Tirado-Ramos, A.G. Hoekstra, and M. Bubak. An interactive grid environment for non-invasive vascular reconstruction. In *2nd International Workshop on Biomedical Computations on the Grid (BioGrid'04), in conjunction with Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*, Chicago, Illinois, USA, April 2004. IEEE. CD-ROM IEEE Catalog # 04EX836C.
7. G. Tsouloupas and M. Dikaiakos. GridBench: A Tool for Benchmarking Grids. In *Proceedings of the 4th International Workshop on Grid Computing (Grid2003)*, pages 60–67. IEEE Computer Society, November 2003.
8. George Tsouloupas and Marios D. Dikaiakos. Characterization of Computational Grid Resources Using Low-level Benchmarks. Technical Report TR-2004-5, Dept. of Computer Science, University of Cyprus, December 2004.