# GridBench: A Workbench for Grid Benchmarking

George Tsouloupas and Marios D. Dikaiakos

Department of Computer Science,
University of Cyprus, 1678 Nicosia, Cyprus
{georget, mdd}@ucy.ac.cy

**Abstract.** In this article we present the GridBench, an extensible tool for benchmarking and testing Grid resources. We give an overview of the GridBench services and tools that provide easy invocation of benchmarks and management of results. We also show how the tool can be used in the analysis of 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 and the GridBench "virtual workbench" to perform benchmarking experiments and analyze the results.

## 1 Introduction

High Performance Computing and it's users have greatly benefited from benchmarking over the years; benchmarking can be just as beneficial for computational Grid computing. Benchmarking metrics published on the Grid can provide a basis for users to assess the "quality of service" expected of a Grid resource or a Virtual Organization providing computational services at a given cost. Grid benchmarks can be used by middleware developers to compare different middleware solutions such as job submission services, resource allocation policies, scheduling algorithms, etc. Grid-oriented benchmarks can serve as an evaluation of the fitness of a collection of distributed resources for running a specific application. As common programming models or paradigms start to emerge for programming in Grid environments, Grid benchmarks can serve as a feasibility study of running a general class of applications (or applications following a similar programming paradigm). A key aspect of Grids and Grid resources is their dynamic nature and Grid benchmarks can help study the effect of this dynamic nature of the Grid on application performance. Additionally, they can provide some insight to the properties of Grid Architectures.

The heterogeneity of Grid platforms and the dynamic nature of Grid resources makes the archival and interpretation of measured metrics a complex task and raises questions about the overall applicability of benchmarking. Existing platforms are largely under continuous re-design and development, with very limited cross-platform interoperability, making the specification, submission, and management of jobs is a tedious process. Measuring and/or monitoring

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 211–225, 2005.

performance metrics at the application level of the Grid is currently a target of ongoing research work. Performance measurements are affected by a variety of factors, including the characteristics of resources allocated for a particular run, the time-dependent latency and bandwidth of shared Internet links used for communication between remote sites, the performance capacity of middleware libraries used at the application level, etc.

In the remainder of this article we describe the GridBench tool for benchmarking and testing Grids. In the next section we describe our current implementation of the Gridbench architecture, services as well as the GridBench User Interface and how we used it to perform experiments on a mid-sized Grid infrastructure. Finally we provide some use-case scenarios and results.

## 2 GridBench

Grids and Grid Resources in general are characterized by static information provided by Grid Information systems. Grid end users and central Grid Services (such as resource brokers) need a better source of information on which to base decisions. These decisions mainly refer to resource allocation or scheduling decisions. The use of results from micro and macro-benchmarks can improve the decision making process, but at this point there is no easy, automated way to obtain, manage and deliver these measurements. GridBench is aimed at fulfilling this purpose.

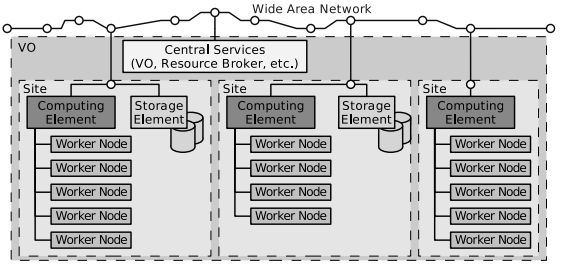



**Fig. 1.** A generic view of the target infrastructure

GridBench assumes an underlying hardware infrastructure that loosely adheres to the one depicted in figure 1. This basic infrastructure, a Grid Virtual Organization (VO) consists of a set of geographically distributed sites connected over a shared network (i.e. the Internet). Each site contains a Computing Element which manages a set of "Worker Nodes" for performing computations. Typically a CE is associated with a "Storage Element", which is an interface to mass storage, and to which it has direct (Local Area Network) access (e.g. via the Network File System). The Grid VO also contains some VO services such as Grid Information Services, a resource broker, VO membership server etc.

GridBench, as a tool for benchmarking grids, has two main objectives:

1. 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.
2. 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.

To address the two main objectives mentioned, Gridbench has two constituents: the *GridBench Benchmark Suite* and the *GridBench Benchmarking Framework*. The GridBench Benchmark suite is a collection of new and existing micro-benchmarks, micro-kernel benchmarks and application benchmarks; its purpose is to generate the metrics that will characterize resources and virtual organizations. The GridBench suite takes a layered approach as shown in figure 2. The multi-layered structure of the Grid (shown in figure 1) calls for performance measurements at the different layers of the Grid. GridBench seeks to investigate performance properties of the following "layers" of the Grid architecture:

1. The Resource, for example a cluster node or a Storage Element;
2. The Site, which is a collection of resources interconnected through a local- or system-area network, and belonging to one administrative domain(e.g. a cluster of PCs or a symmetric multiprocessor system);
3. The Grid Constellation, which includes multiple sites constituting the computing platform of a Virtual Organization.
4. The Middleware, that is the software layer providing access to shared resources of a Grid constellation and which gives the programmer the Grid as a shared resource.

The suite includes benchmarks for CPU (Floating Point and Integer operations), memory bandwidth, cache performance, detecting available physical memory size, interconnect performance (MPI), synthetic benchmarks and application kernels. A detailed description of the GridBench suite is beyond the scope of this article, more details on the Gridbench suite can be found in [10, 11].

### 2.1 The GridBench Back-End

The GridBench Benchmarking Framework provides facilities for defining and running benchmarks as well as archiving, retrieving and analyzing the results of the GridBench benchmark suite.

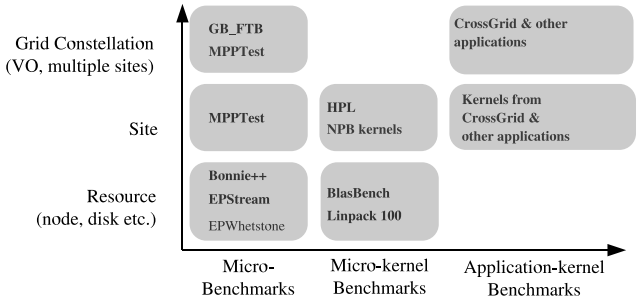



**Fig. 2.** A layered approach to benchmarking, with *micro-benchmarks*, *micro-kernel* benchmarks and *application benchmarks* on the x-axis, and *resource*, *site* and *grid constellation* on the y-axis

GridBench was designed to be as independent of specific middleware as possible. The design is open enough to allow easy replacement of the underlying middleware by the use of *Middleware plugins*. Currently implemented are plugins for Globus and the LCG2-compatible [8] EU CrossGrid middleware. The user can use the Globus MDS [5] for information retrieval, and either the EU CrossGrid [6, 7] Resource Broker or the Globus GRAM for job execution.

## 2.2 Overview

Figure 3 outlines the software architecture of GridBench and (at a very high level) indicates which components interact with each other. This is indicated by a line connecting the two interacting components. The main components of this architecture are:

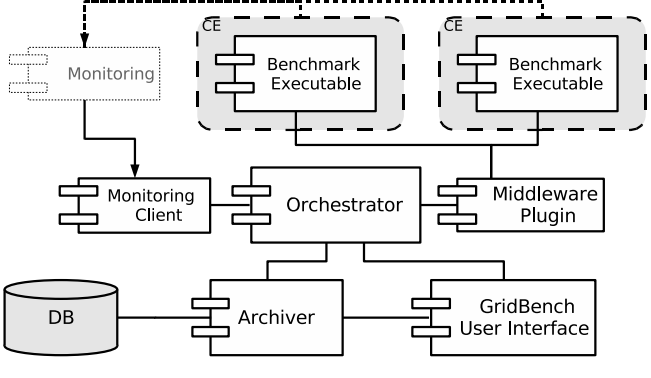



**Fig. 3.** The GridBench architecture overview. An outline of system's major components and their interaction

- the **GridBench Suite**
  - is made up of the benchmark executables (e.g. Linpack).
- the **GridBench UI**
  - Interacts with the *Archiver* to retrieve benchmark *models*[1] and results.
  - Defines and submits benchmarks to the *Orchestrator*.
  - Analyzes results, create charts.
- the **Orchestrator** web-service
  - Accepts benchmark definitions generated by the *GridBench GUI* (or any other source) and manages their execution by the use of the appropriate *Middleware Plugin*.
  - Monitors the job status for each benchmark job and on completion retrieves and archives the resulting metrics.
- the **Archiver** web-service;
  - Maintains a repository of benchmark results and model definitions.
  - Provides an interface to a relational database back-end.
- the **Middleware Plugin**
  - Middleware-specific job execution, output retrieval.
  - Translation of XML descriptions of benchmarks to a job description language;
  - There are currently two implementations of the Middleware Plugin interface: one for Globus and one for the EU-CrossGrid middleware.
- the **Monitoring Client** (collects monitoring information);
  - Collects infrastructure monitoring data (as specified in each GBDL) by using different *Monitoring Clients*.
  - Infrastructure monitoring data can be used to interpret benchmark results based on the state of the infrastructure during benchmark execution.

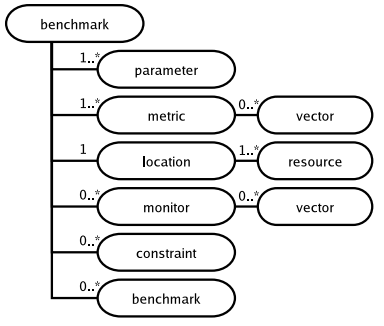
### 2.3 The GridBench Definition Language

The GridBench Definition Language was introduced to the system for several reasons:

- To allow easy definition of benchmarks, including work-flow benchmarks;
- To introduce a middleware-independent definition of benchmarks;
- To serve as a container for associating a definition to the resulting metrics as well as the collected monitoring data.

Figure 3 provides a high-level schematic view of the GridBench Definition Language. The benchmark definition includes all necessary information needed to run a benchmark. It includes a set of *parameters*, which specify details for the benchmark execution (such as the path to the executable and benchmark-specific parameters). It also contains a *location* which specifies the resources on which it should run. A *benchmark* can be hierarchical in nature, meaning that

[1] A *model* definition is a template benchmark definition with default parameters. A model is used for the creation of a new benchmark definition.




```
<benchmark name="epwhetstone"
          date="20040515023918"
          type="mpi" >
 <location>
   <resource name="cluster.ui.sav.sk"
         cpucount="16"
          wncount="16"
       jobmanager="jobmanager-pbs-workq"/>
 </location>
 <parameter name="executable" type="value"
        dataType="0">epwhetstone</parameter>
 <parameter name="execpath" type="value"
        dataType="0">/opt/cg/gb/bin</parameter>
 <parameter name="stage_executable" type="value"
        dataType="0">manual</parameter>
 <parameter name="nloops" type="value"
       dataType="1">10000</parameter>
</benchmark>
```

**Fig. 4.** Left: A schematic overview of GBDL; shown in boxes are the main parts of a GBDL document. Right: An example GBDL definition

it can be made up of other *benchmarks*. This, in conjuction with the use of the execution *constraint* elements, can be used to specify simple workflows. A benchmark *metric* may be in the form of a single value or in the form of a *vector* of values (such as bandwidth at different packet-sizes).

### 2.4 Archiver Web-Service

The *Archiver* allows the storage and retrieval of results generated by executions of the GridBench Suite Benchmarks through the Gridbench Framework.

The *Archiver* was introduced in order to serve the following purposes:

- To 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.
- To provide a central repository for the results allowing access to measurements for users or Grid services.
- To hold a set of *model* definitions serving as customizable benchmark definitions.

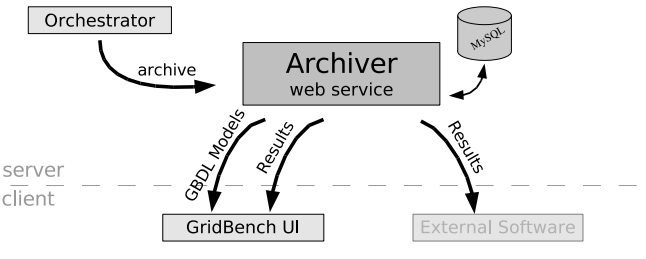



**Fig. 5.** Diagram describing the Archiver functionality

The *Archiver* is an *interface* implemented as a web service. The *Archiver* interface may have several implementations depending on the back-end in use. There are already implementations for using the *Apache Xindice* native XML database as a back-end and the (newer) *MySQLArchiver* implementation using the MySQL relational database as a back-end.

### 2.5 Orchestrator Web-Service

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, as well as all the collected monitoring data. The final GBDL is then archived using the Archiver service.

The diagram in figure 6 describes the Orchestrator functionality in a series of steps. The steps are given below (the numbers correspond to the circled items in the diagram):

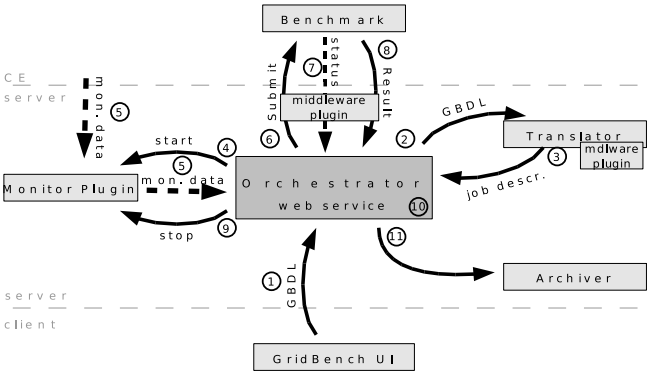



**Fig. 6.** Diagram describing the Orchestrator functionality

1. The *Orchestrator* receives a benchmark description in the GridBench Description Language (XML). This will originate from the GridBench GUI or from an automated system performing automated / periodic executions;
2. The GBDL is passed to the *GBDL translator* (which is part of the *Middleware Plugin*) which generates a middleware-specific job description in the syntax and format required by the underlying middleware;
3. The middleware-specific job description is then returned to the *Orchestrator*;
4. The *Orchestrator* determines all monitoring that need 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 banchmark is put in the 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 benchmark job's 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, as well as the monitoring data, are incorporated into the original GBDL. If the *resources* specified in 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.

### 2.6 The GridBench User Interface: The "Virtual Workbench"

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 archives results. Figure 8 shows the main graphical use interface for the definition of benchmarks.

In Figure 8 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 8 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 8). 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 7.

## 3 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 in using the tool to get performance metrics for Grid resources. First we describe the scenario where a user would like to get a "picture" of the currest status of a set of resources in terms of low-level performance metrics. In the second case the user has a specific application in mind and would like to select a resource onto which to execute the application. Many other use-case scenarios are possible; in fact some do not even involve an end-user. For example, metrics obtained through GridBench mechanisms can be used by a scheduler that performs resource ranking on an application basis in a way that is completely transparent to the user.

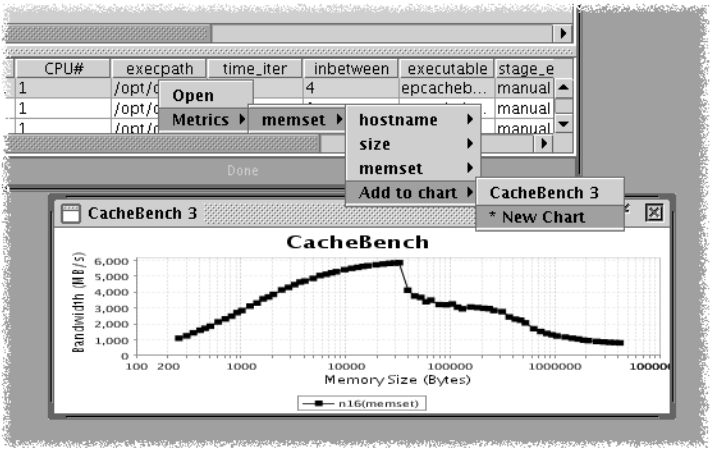



**Fig. 7.** The GridBench graphical user interface showing the generation of charts from historical data. The result shown is from a cache benchmark

### 3.1 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:

1. Determine which metrics will tell you what you want to know about the resources. In this case, the metrics for these factors can be delivered by a set of benchmarks as summarized below:

| Factor | Metric | micro-benchmark |
|---|---|---|
| CPU | OP/s | EPWhetstone |
| Memory | bandwidth | EPStream |

2. Using the GridBench GUI simply drag each of the benchmarks onto each resource and submit the benchmark (Figure 8). When the benchmark execution finishes, the result will be automatically archived.
3. Using the GridBench GUI put together comparative charts for the resources for each benchmark (Figure 9).

From the results on Figure 9 (the charts were generated using the GridBench GUI) we observe that the three sides that were chosen for comparison vary in

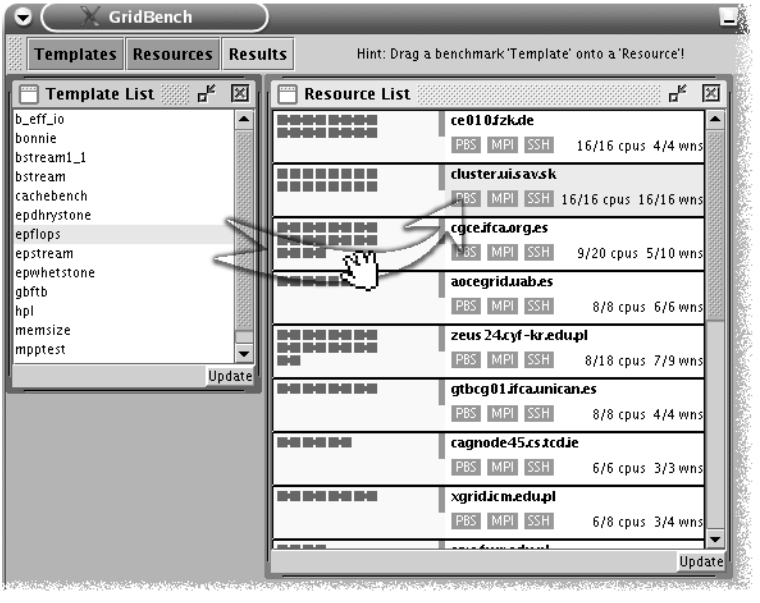
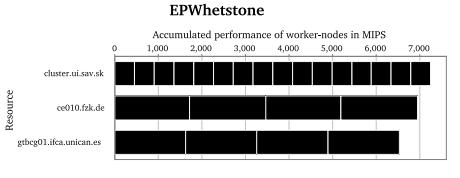
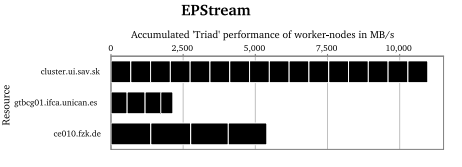



**Fig. 8.** Screen-shot of the GridBench graphical user interface. The list on the left is a list of benchmarks that are integrated into GridBench. The list on the right shows the currently available resources and their status in terms of busy/free CPU's. Invoking a benchmark on a resource is as simple as dragging a benchmark from the template list to a resource in the resource list




(a) Aggregate CPU performance




(b) Aggregate memory bandwidth

**Fig. 9.** Results for use-case scenario 1

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 sightly. In terms of memory bandwidth the performance varies greatly as shown in figure 9(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`.

### 3.2 Use-Case Scenario 2: Application Performance

As a second use-case we consider a user that wants to compare resources based on performance of a given application or kernel [2]. The user, in this case a surgeon, needs to find the best resources to run a set of simulations. The user has a given application that is used *frequently*, it is therefore justifiable to perform some trivial instrumentation/timings on the application's computational kernel (e.g. to measure iteration times or simply measure completion time on a given dataset) and make it part of the benchmarks available in GridBench.

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 following steps need to be taken:

1. Create a new GBDL description (model) and add it to the Archiver database;
2. Write a simple implementation of the ParameterHandler Java interface;
3. Instrument the code of the kernel to generate metrics.

*A New GBDL Description*

The first step in adding a new kernel is to create a new GBDL description such as the one that follows:

```
<benchmark name="bstream1_1" date="" type="mpi"
           model="true" description="B_stream 1.1 ..." >
  <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 states that:

- this is a benchmark description that is to be used as a model (*model="true"*);
- the parameters *iterations*, *Reynolds* and *data_id* are application-specific parameters required by the kernel executable;
- "bstream1.1" is the name of the *executable* and file "tube38x40x40.bs" needs to be staged;

[2] The kernel in question is from a medical application, developed at the University of Amsterdam, for pre-operative planning of vascular reconstruction. It involves blood-flow simulation using a Lattice Boltzmann method in arteries using 3-Dimensional data obtained from MRI scans of the patient [9].

Since the formatting of command-line arguments to the application executable is application-dependent the user needs to provide a ParameterHandler.

*Writing a ParameterHandler*

A benchmark-specific ParameterHandler is required for special formatting of command-line arguments (or creation of parameter files etc). In this use-case scenario the applications takes three parameters, which need to be provided in a given order on the command-line. A typical invocation would be:

```
bstream1_1 20 tube38x40x40 40
```

During translation of the GBDL to the middleware-specific job description, the class *ParameterHandler_**bstream1_1*** will be dynamically loaded:

```
public class ParameterHandler_bstream1_1 implements ParameterHandler{
  public java.util.Vector getCommandLineArguments(Benchmark benchmark){
    ParameterCollection parameters=benchmark.getParameters();
    Vector parameterVector=new Vector();

    Parameter data_id=parameters.getParameter("data_id");
    Parameter reynolds=parameters.getParameter("Reynolds");
    Parameter iterations=parameters.getParameter("iterations");

    parameterVector.add(raynolds.getValue());
    parameterVector.add(data_id.getValue());
    parameterVector.add(iterations.getValue());

    return parameterVector;
  }
  ...
}
```

The method *getCommandLineArguments()* is called and returns an ordered list of parameters correctly formatted and ready to be passed to the application executable.

*Instrumenting Application Codes*

Instrumentation of codes is highly application-specific and usually involves trivial modification of the source code to obtain timings at a high level. In our 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>
```

*Obtaining Measurements*

Once the kernel has been integrated into GridBench the user can invoke it just like any other benchmark. The same steps listed in the previous use-case apply to this case as well. One difference is that now the kernel benchmark takes considerably longer to run (tens of minutes) than the micro-benchmarks (a few seconds) in the previous use-case. In this case the user opts to use previously archived executions of the kernel benchmark because it is considerably expensive to get fresh measurements. The steps are now:

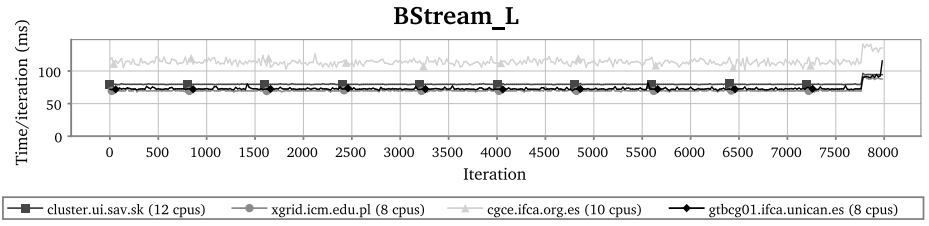



**Fig. 10.** Results for use-case 2, showing iteration times of a given kernel on four resources

1. Retrieve archived results for this kernel;
2. Benchmark the resources for which there are no archived results;
3. Compare the results.

Invoking the kernel benchmark on a set of resources allows us to construct the chart shown in Figure 10. Based on these results a user (or resource broker) can make relatively safe decisions for resource selection, given that the criterion for a "good" selection is the performance of the given kernel.

## 4 Related Work

The ALU-Intensive Grid Benchmarks [3] are a specification to run the the NAS Parallel Benchmarks [1] in pre-defined workflows and from that infer the performance of Grid systems. The AIGB aim to benchmark the ability of dynamic collections of Grid resources to executes several types of workflow, while the GridBench suite proposed an more hierarchical approach both in terms of infrastructure and of benchmark types. Nevertheless, the GridBench tool could serve as a means to execute these benchmarks just like any other benchmark.

Diperf [4] is a distributed performance-testing framework aimed at automating performance evaluation of services. It does not address computational resources or network performance directly.

Also, work has been done to "assess" the Grid using "probes" [2] but this work focuses mainly on file transfers, remote execution, and Information Service responses. Computational resource performance is not addressed.

## 5 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 an aid for analysis of metrics.

We have presented two elementary use-case scenarios and illustrated the ease of use of the tool: The first use-case illustrated how end-users and administrators can perform benchmarking experiments either for resource selection of 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.

We plan to extend 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).

## Acknowledgments

This work was supported by the European Union through the CrossGrid project (IST-2001-32243). The authors wish to acknowledge Alfredo Tirado-Ramos and Lilit Abrahamyan (University of Amsterdam) for the blood flow application code, and the support of the CrossGrid testbed team for running the distributed simulation.

## References

1. David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow. The nas parallel benchmarks 2.0. *The International Journal of Supercomputer Applications*, 1995.
2. Greg Chun, Holly Dail, Henri Casanova, and Allan Snavely. Benchmark probes for grid assessment. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*. IEEE Computer Society, 2004.
3. R.F Van der Wijngaart and Michael Frumkin. Alu intensive grid benchmarks. https://forge.gridforum.org/projects/gb-rgs, 2004.
4. Catalin Dumitrescu, Ioan Raicu, Matei Ripeanu, and Ian Foster. Diperf: an automated distributed performance testing framework. In *Proceedings of the 5th International Workshop on Grid Computing (GRID2004)*. IEEE, November 2004.
5. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symp. on High-Performance Distributed Computing*, pages 365–375. IEEE Computer Society, 1997.
6. The EU CrossGrid Project. http://www.eu-crossgrid.org.
7. The EU DataGrid Project. http://www.eu-datagrid.org.
8. The LCG Project. http://lcg.web.cern.ch/LCG/.

9. P.M.A. Sloot, 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.
10. George Tsouloupas and Marios D. Dikaiakos. Gridbench: A tool for benchmarking grids. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, pages 60–67, Phoenix, AZ, November 2003. IEEE.
11. 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, 2004.