Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 1029–1045
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

GridBench: A tool for the interactive performance exploration of Grid infrastructures

George Tsouloupas*, Marios D. Dikaiakos

Department of Computer Science, University of Cyprus, 75 Kallipoleos Street, P.O. Box 20537, CY-1678 Nicosia, Cyprus

Received 10 October 2006; received in revised form 22 February 2007; accepted 5 April 2007
Available online 3 May 2007

Abstract

As Grids rapidly expand in size and complexity, the task of benchmarking and testing, interactive or unattended, quickly becomes unmanageable. In this article we describe the difficulties of testing/benchmarking resources in large Grid infrastructures and we present the software architecture implementation of GridBench, an extensible tool for testing, benchmarking and ranking of Grid resources. We give an overview of GridBench services and tools, which support the easy definition, invocation and management of tests and 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 and user-driven resource ranking. In order to illustrate the usage of the tool, we describe scenarios for using the GridBench framework to perform test/benchmark experiments and analyze the results.

Keywords: Grids; Performance; Benchmarking; Testing; Ranking

1. Introduction

Grids have emerged as wide-scale, distributed infrastructures that comprise heterogeneous computing and storage resources, and support resource-sharing in growing dynamic infrastructure. Grids are quickly gaining popularity, especially in the scientific sector, where projects like *EGEE* (Enabling Grids for E-sciencE) and the *Open Science Grid* provide the infrastructure that accommodates large experiments with thousands of scientists, tens of thousands of computers, and petabytes of storage [9,16]. As an example, at the time of this writing, EGEE assembles over 200 sites around the world with more than 30,000 CPUs and about 5 PB of storage, supporting over 80 Virtual Organizations and an increasing number of large-scale applications from a variety of disciplines resulting to an average of 10,000 concurrent jobs [9].

An important issue faced by users of large-scale Grids is the selection of the specific set of resources upon which to dispatch a Grid job. In state-of-the-art Grid infrastructures, resource selection is based on the *matchmaking* approach introduced by the Condor project [19] adapted to multi-domain environments and Globus; it has been extended to cover aspects such as data access and workflow computations, interactive Grid computing, and multi-platform interoperability [2,14]. Matchmaking produces a ranked list of resources that are compatible to submitted resource requests.

In current Grid systems, resource selection and ranking decisions are typically based on a combination of static and dynamic monitoring information regarding the number of CPUs of each resource, their nominal speed, the nominal size of main memory, the number of free CPUs, available bandwidth, etc. This information is retrieved from Grid Information Services like the Monitoring and Discovery System of Globus [6] or R-GMA [5]. This approach works well in cases where the main consideration of end-users is to allocate sufficient number of idle CPUs in order to achieve a high job-submission throughput with opportunistic scheduling [18].

In several scenarios, however, reliance to simple matchmaking is not enough: practical experience from the operations of production Grid facilities like CrossGrid [11] and EGEE [9] has shown that many users wish to select and rank the resources

* Corresponding author. Fax: +357 22892701.
E-mail addresses: georget@ucy.ac.cy (G. Tsouloupas), mdd@ucy.ac.cy (M.D. Dikaiakos).

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.04.009

upon which to dispatch their jobs, adapting the selection criteria to their preferences in a dynamic and interactive manner; also, that VO operators want to audit the delivered performance, the availability, and the configuration status of their providers' computing resources in an end-to-end fashion. In such cases, the information published by resource providers and Grid monitoring systems is not of sufficient detail, scope, and accuracy. Grid users need, instead, the capability to define and configure on-demand various kinds of tests, tailored to the structure and the characteristics of individual resources and to their application requirements. Grid users need also the capability to easily administer such tests and to analyze test results in an interactive fashion.

However, inherent characteristics of Grids, like the virtualization of resources, the layered structure of the Grid architecture, and resource heterogeneity, render the development of a reliable, interactive performance exploration environment a challenging task. To address this challenge, we designed and implemented GridBench, a modular software system that enables the performance exploration of large-scale Grids in an interactive manner. GridBench supports the definition, deployment and execution of parameterized tests and benchmarks on the Grid, while at the same time allowing for the validation, archival, retrieval, and analysis of test results.

In this paper, we describe the architecture and functionality of the GridBench framework. Also, we present the mechanisms that GridBench has for supporting the interactive, user-driven ranking of Grid resources with user-specified metrics, custom ranking functions and ranking models. Finally, we demonstrate the use of GridBench for the effective selection and ranking of resources belonging to EGEE, the European production Grid [9]. The remaining of this paper is organized as follows: in Section 2, we provide an overview of the main challenges that need to be addressed in order to support interactive resource selection and ranking of Grid resources. Also, we present the main decisions that we adopted in the design and implementation of GridBench to address those challenges. In Section 3, we describe GBDL, the XML schema that we designed to define and automate tests conducted with GridBench; we use GBDL to support the storage and integration of performance measurements, configuration parameters, and monitoring information that are required for the proper evaluation of test results. Section 4 describes in more detail three main components of the GridBench system: the GridBench Controller, the GridBench Browser, and the GridBench Crawler. In Section 5, we describe the GridBench Ranking Module. A number of use-case scenarios involving real infrastructures and applications are presented in Section 6. In Section 7 we provide a summary of related work and we finish off in Section 8 with conclusions and future work.

2. GridBench design considerations

2.1. The Grid context

In this work, we assume a Grid infrastructure consisting of a set of geographically distributed, heterogeneous Grid sites connected over a shared network (e.g. the Internet) and supporting several Virtual Organizations. In Fig. 1, we present a model of this infrastructure, inspired by the architecture of large-scale Grid testbeds such as EGEE [9]. A Grid site comprises a cluster of *worker nodes* (WN), which are typically off-the-shelf PCs or server-class machines, interconnected via a high-speed local area network. Access to a Grid site is provided through a *computing element* (CE), a node that hosts the site's job submission, queuing, VO management and accounting capabilities. Typically, a site also comprises a *storage element* (SE), which is an interface to mass storage. Each site also hosts a *monitoring agent* (MA), which collects information from local operating systems, configuration files and cluster-management systems, and publishes it through a Grid-wide Information Service.

A user can initiate Grid-job submission through a *user interface* (UI) machine, which hosts the necessary middleware components and services, and serves as user gateway to the Grid. To this end, the user needs to have proper security credentials and be a member of a supported Virtual Organization. VO membership specifies the resources that will be assigned to user jobs, according to the access rights and usage policies that CEs apply for the various VOs.

2.2. Key challenges

To support the selection and ranking of resources using configurable on-demand tests in the context described above, we need to address a number of challenges that arise from the inherent characteristics of Grids (an extensive discussion on these challenges can be found in [7]).

Scale and complexity: The multi-layered structure of Grids suggests that the observed performance of Grid resources is affected by several factors: (i) the capacity of local Grid sites and inter-connecting networks; (ii) the performance and overhead of libraries and services providing Grid applications with support for communication, synchronization, bulk data transfer, database querying, and other higher-level Grid programming abstractions; (iii) the performance and overhead of Grid services supporting job submission and management, such as workload management systems, resource brokers, and Grid Information Services, and (iv) the reliability and robustness of Grid middleware and services.

Therefore, the selection and ranking of Grid resources needs to address the numerous observable aspects of the Grid architecture that affect the functionality and performance of resources. Administering such tests, collecting and interpreting measurements on large-scale Grids can be a tedious, time-consuming, and costly process, especially if one needs to evaluate a substantial fraction of available resources distributed across multiple administrative domains.

Volatility: With too many organizations, sites, and resources participating in a Grid infrastructure, the infrastructure is typically in a continuous change as different sites add or withdraw resources, conduct hardware or software upgrades, re-configure their hardware or middleware, suspend their operation due to failures, etc. Additional sources of volatility are the

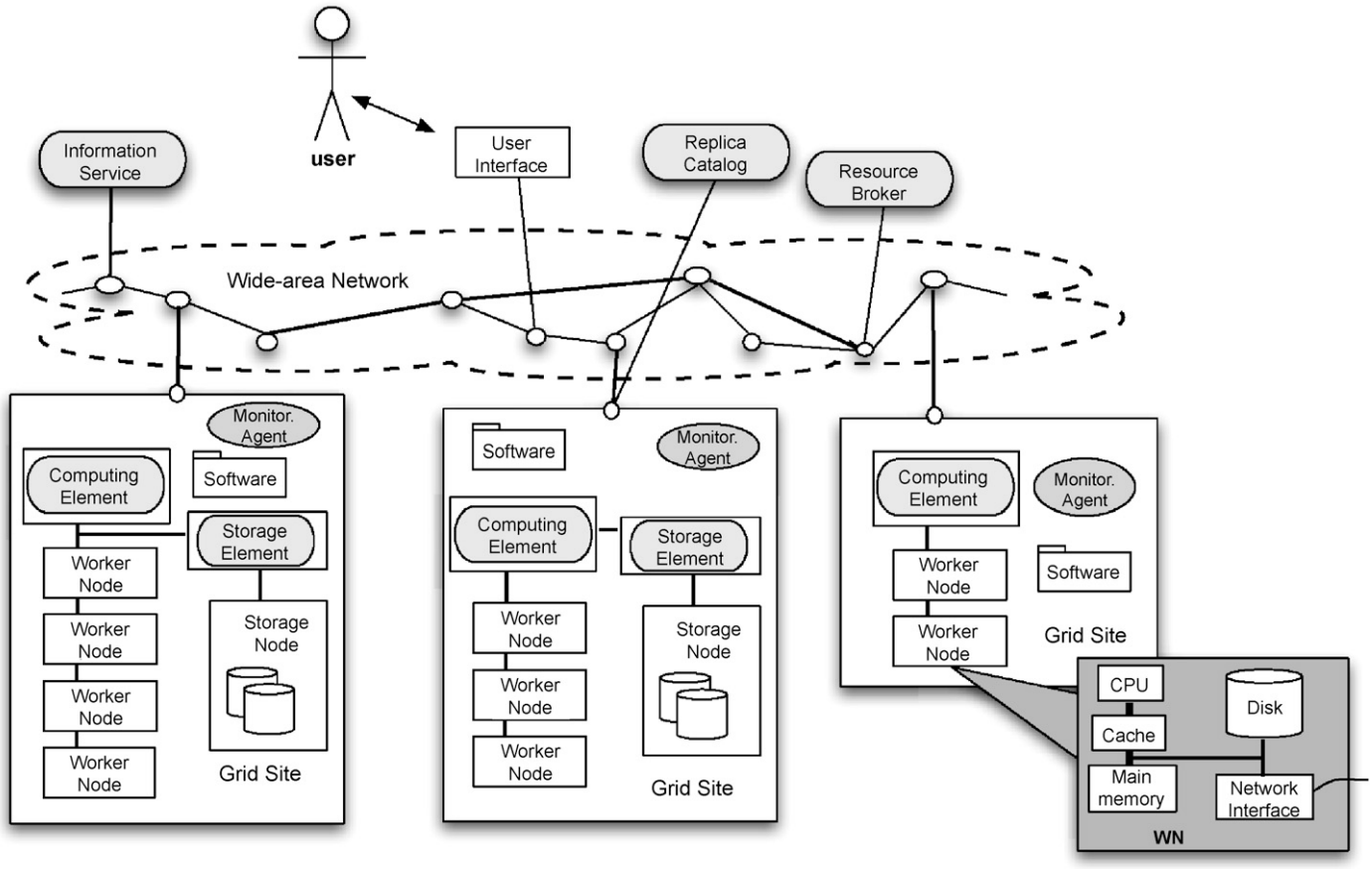


Fig. 1. A typical Grid architecture.

open wide-area networks, which are used to interconnect Grid sites and which are shared with millions of Internet users, and the access policies that some Grid sites apply, allowing the dynamic co-allocation of multiple Grid jobs on the same Worker Node (time-sharing instead of space-sharing). Grid volatility can have non-trivial effects on the consistency of the performance delivered to users by Grid resources; it complicates testing and evaluation of Grids since measurements can be irrelevant or soon become out-dated.

Heterogeneity: Typically, different sites of a Grid infrastructure host resources that differ in architecture, configuration, performance capacity, etc. Often, even the clusters of individual Grid sites are non-homogeneous. Resource testing has to be adapted to the attributes of individual Grid resources to provide meaningful measurements. Also, the derived measurements should expose the levels and the impact of heterogeneity to the service that end-users get from the Grid [12,20].

Virtualization: One of the key goals of Grid Computing is the virtualization of distributed resources. Virtualization, combined with the open nature of Grids and the lack of central administrative control therein, complicates the interpretation of test measurements and the reliability of derived conclusions. For instance, many CEs support several job queues, with each queue providing access to a potentially different type of hardware or software and possibly serving a different VO. Consequently, users belonging to different VOs may have a totally different view of the infrastructure's performance.

2.3. Main design concepts

In view of the challenges described above, Grid resource selection tools should be designed to address the following key concerns:

- The *minimization of the testing effort*; this can be achieved by adopting user-friendly interaction paradigms and implementing automation mechanisms for the administration of tests and the analysis of measurements. Also, by providing standardized and widely accepted tests that can be easily configured according to the specific characteristics of Grid resources and user preferences.
- The *annotation of measurements with experimentation metadata*, which represent various conditions under which the corresponding experiments were carried. Annotation is important in order to: (i) help in exposing the effects that Grid virtualization has on derived measurements; (ii) enable the identification and filtering of irrelevant or invalid measurements; (iii) identify the presence of heterogeneity and expose its effects.

- The *mapping of raw measurements* of Grid resources to higher-level metrics that characterize Grid resources at a level of abstraction that is closer to the end-users' view of the Grid.

To address these concerns and to provide an interactive resource exploration framework, we designed GridBench along the following axes:

End-to-end testing: GridBench follows an *end-to-end* approach in the deployment and execution of tests. Tests are performed using the exact same mechanisms as regular Grid jobs and are subject to the very same limitations and treatment. The outcome of end-to-end testing represents the performance and functionality of Grid resources as it is experienced by Grid users, taking into account the effects of virtualization.

Functional testing and benchmarking: GridBench supports *functional testing* and *benchmarking* of Grid resources. The former, comprises the execution of small probes or queries to obtain functionality and availability information about Grid resources. Information extracted from functional testing is used to update, complement or validate information published in Grid Information Services, to test the operational status of local resource managers, and to configure performance benchmarks with the right set of parameters. Benchmarking employs well-understood instrumented codes that investigate the performance behavior of Grid entities belonging to different layers of the Grid architecture of Fig. 1.

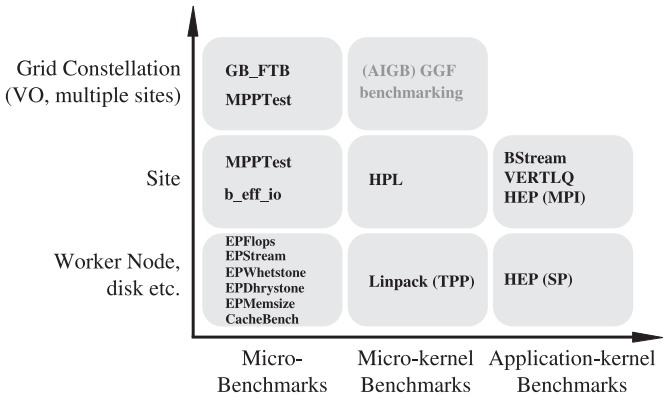


Fig. 2. The GridBench Suite of benchmarks.

Performance benchmarks: GridBench provides a preselected *layered* suite of widely acceptable and portable benchmarks that provide a thorough and concise characterization of Grid-resource performance and whose open deployment and use are not restricted by intellectual property rights (see Fig. 2). This suite comprises: (i) *Micro-benchmarks* for stress-testing and measuring the performance of some Grid entity in isolation. (ii) *Micro-kernels*, for measuring the performance of some composite Grid entity under some synthetic workload, designed to stress-test simultaneously several aspects of the Grid entity's performance. (iii) *Application kernels* and *Grid applications* used to investigate the performance of some Grid entity (i.e., a WN or site). The tool is flexible and extensible enough to accommodate most types of benchmarks, targetting a single resource, or benchmarks of a distributed nature (such as the GB_FTB GridFTP data-transfer benchmark). Nevertheless, benchmarking of Grid services (such as stress-testing resource brokers by applying different workloads), or measuring the capacity of wide-area network connectivity falls outside the scope of this paper. Tools that can be used in such situations are briefly outlined in Section 7.

A detailed list of the benchmarks integrated in the GridBench suite and used for the performance exploration of Grid testbeds like CrossGrid and EGEE is given in [23].

Client–server architecture: GridBench is designed as a client–server system (see Fig. 3) primarily in order to enable the sharing of measurements between different users. This architecture also enables the operation of the system from personal computers without the need to have special middleware

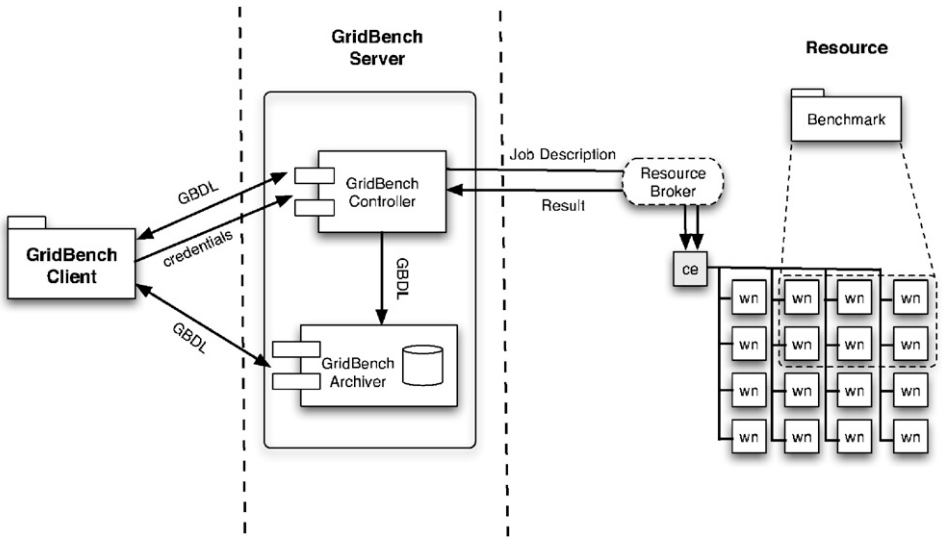


Fig. 3. GridBench component communication.

installed or to bypass firewalls. The GridBench *server* is implemented as a set of Web-services installed on a UI-machine that is configured to access remote Grid Sites using Grid protocols. The GridBench *client* can be executed as a stand-alone application, a Java applet, or a plugin to virtual desktops like the CrossGrid Migrating Desktop [13].

GridBench metadata: Grid resource selection and ranking requires the retrieval, integration and processing of quantitative information from several sources. These sources differ in their access mechanisms, timing, protocols, and schemas used to produce and publish their data. To derive a performance measurement, GridBench actively submits for execution a Grid job carrying an instrumented benchmark that is selected and configured to measure a specific aspect of some resource. This benchmark is deployed through the Grid on the resource under investigation.

To make GridBench operations easily configurable and to enable the integration of test results with experimentation metadata (configuration values, resource load information, etc.), we define a common way of specifying the various operations launched by GridBench and the parameters thereof, by introducing the XML-schema-based *GridBench definition language* (GBDL) described in the following section.

3. The GridBench definition language

3.1. Scope

GBDL documents drive the operations of the GridBench Controller and its interactions with other components and services. Furthermore, GBDL is used to represent raw measurements derived from GridBench tests, and to annotate those measurements with metadata necessary for the processing, aggregation, and interpretation of metrics. Communication between GridBench components is performed through the exchange of GBDL documents (see Fig. 3). GBDL documents are stored in the GridBench Archiver.

A complete GBDL document includes the definition of a test or benchmark invocation with specific parameters, the target resources under testing, a time-stamp of the experiment undertaken, the status of the target machines during execution as captured by monitoring systems, and the resulting metrics.

3.2. Syntax

A high-level structure of GBDL documents is presented in Fig. 4(a). According to the GBDL syntax, the top-level XML element in a GBDL document is the ⟨`testmark`⟩. ⟨`testmark`⟩ elements assemble all information that is related to a GridBench experiment and contain the set of other XML elements shown schematically in the tree-structure of Fig. 4(a). An example of a GBDL definition is given in Fig. 4(b). The elements on the left side of the tree-structure of Fig. 4(a) (`parameter`, `credential`, `resource`, `monitor`, `memo`, `testmark`, and `constraint`) specify the configuration of a test, i.e., the operations that need to be undertaken in order to launch a test and derive meaningful measurements. The elements on the right (`info`, `metrics`, `log`, and `status` entries) correspond to information produced or retrieved during the execution of a test on a Grid; this information is embedded in the document during and upon completion of the execution.

The ⟨**credential**⟩ element carries the credentials of the user submitting a test to a Grid, such an x509 proxy certificate, in hexadecimal form. The ⟨**parameter**⟩ element enables the definition of any parameters that GridBench needs to pass to the underlying middleware or to the testing codes. GBDL currently supports two types of `parameter` elements: (i) `conf` parameters are middleware-specific and act as directives to the underlying middleware in order to configure the submission of a test as a Grid job. (ii) `user` parameters are test-specific and initialize the input parameters of the test executable.

The ⟨**resource**⟩ element specifies the resources that are targeted by the enclosing ⟨`testmark`⟩. For example, this element can define the name of a Grid site, the number of CPUs to be requested from that site, and how the CPUs should be distributed on that site's WN. To this end, the element has three associated attributes: `cpucount`, `wncount`, and `name`. The GridBench Controller extracts information from the contents and attributes of the ⟨`resource`⟩ element in order to configure accordingly the job submitted for execution through the Job Submission Service.

The ⟨**monitor**⟩ element provides directives on what to monitor during benchmark execution. It can contain a monitoring system-dependent query and a specification of the monitoring system that will execute this query. The contents of a ⟨`monitor`⟩ element are interpreted and executed by a corresponding plugin of the GridBench Controller.

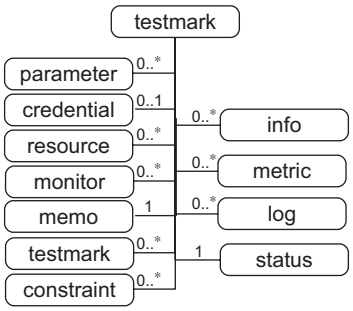
Measurements derived from GridBench experiments are represented as the ⟨**metric**⟩ element of GBDL. This element accepts a `node` attribute, associating the represented measurement with the name of the resource under measurement, and a `name` attribute, which specifies the type of the measurement. Nested inside ⟨`metric`⟩ is the ⟨`value`⟩ element, which encodes the actual values of a measurement. The following is an example from the "flops" micro-benchmark. It shows the "MFLOPS(1)" metric (623.5 MFlop/s) measured on the *wn113.grid.ucy.ac.cy* Worker Node.:

```
<metric node="wn113.grid.ucy.ac.cy" name="MFLOPS(1)">
  <value unit="MFLOP/s">623.5426</value>
</metric>
```

Some results are in the form of vectors rather than single scalar metric values. For example, the cache benchmark produces a series of values that state the memory bandwidth using arrays of progressively larger size:

```
<metric node="wn113.grid.ucy.ac.cy"
  name="cache-write">
  <vector name="size">
 256 384 512 768 ... 134217728</vector>
  <vector name="bandwidth">
 1999.2 2202.2 2328.6 ... 488.5</vector>
</metric>
```

a



b

```
<testmarkname="flops"
  xmlns="http://gridbench.ucy.org/definition"
  id="1135002331003036000"
  tstart=""duration=""node=""
  validate="no">
    <parameternname="RB"
      type="conf">rb101.grid.ucy.ac.cy</parameter>
    <parameternname="loops"
      type="user">100</parameter>
    <resourcenname="ce101.grid.ucy.ac.cy"
      cpucount="2"
      wncount="2"/>
</testmark>
```

Fig. 4. (a) A schematic overview of GBDL. Shown in rounded boxes are the main parts of a GBDL document. (b) A real example of GBDL definition, showing a *flops* micro-benchmark definition configured to run using two CPUs on two separate worker-nodes at *ce101.grid.ucy.ac.cy*.

It is worth noting that, according to the GBDL syntax, a ⟨testmark⟩ element may contain other nested ⟨testmark⟩ elements. The combination of nesting with ⟨constraint⟩ elements allows for the definition of tests of arbitrary complexity, such as workflow-like benchmarks. The ⟨**constraint**⟩ element accepts a type attribute, which is used to distinguish between corequisite and prerequisite constraints, and a wfref attribute, which is used to point to an associated ⟨testmark⟩ component. A corequisite constraint means that the ⟨testmark⟩ containing this constraint should be started *after* the test specified by *wfref* has started its execution. A prerequisite constraint means that the testmark containing this constraint should be started after the termination of the execution of the testmark specified by *wfref*.

The GBDL syntax provides also a number of elements that can be used to encode and represent additional semi-structured information that is useful for the visualization and analysis of GridBench measurements. The ⟨**memo**⟩ element holds a short description of the test defined by the enclosing *testmark*. The ⟨info⟩ element assembles information about the characteristics of resources under testing in terms of name-value pairs. For example, a testmark running on dual *AMD Opteron* worker node would contain:

```
<info name="cpu_model" value="AMD Opteron(tm)
Processor 246"/>
<info name="cpu_count" value="2"/>
```

The ⟨**log**⟩ elements are used for keeping the history of a specific testmark execution in the form of entries that log-specific activities of GridBench components during GridBench experimentation. These entries are inserted by the GridBench components (identified in the origin attribute) as they process the GBDL document. For example:

```
<log time="113500233"
origin="Controller">Request received.</log>
<log time="113500235"
origin="LCG-plugin">Job submitted to RB.</log>
```

Finally, the ⟨**status**⟩ element reflects the current status of execution of a test. It can take the values of *pending*, *failed*, *done*, *warn* and *valid*. When the GBDL definition is first created, the value of this tag is set to *pending*. Upon successful completion the value is set to *done*, and in the case of a failed test the value is set to *failed*. When the test finishes successfully and the *validate* attribute is set to *yes*, the output of the test will be validated for correctness. Validation of the result of a test, is performed by invoking a script based on a naming convention. Depending on the outcome of the script the content of the <status> element will be updated to *warn* or *valid* accordingly.

4. GridBench system design

GridBench employs a client–server architecture using Web-services. The GridBench server comprises: (i) the GridBench *Controller*, which manages the testing process by interacting with Job Submission Services, Resource Brokers, and Grid Information Services, (ii) the *Crawler*, which automates the performance exploration of a whole infrastructure, and (iii) the *Archiver*, which undertakes the storage, management, and provision of test templates and of derived measurements (Fig. 5).

The GridBench client comprises: (i) the GridBench *Browser*, which provides a graphical framework through which an end-user can visualize the status of Grid resources, and interact with GridBench; (ii) a *Configuration Module*, which supports the interactive configuration of GridBench tests, (iii) an *Analysis Module*, which supports the processing, visualization and interactive manipulation of collected metrics; and (iv) the *SiteRank*, which implements the models used to rank Grid resources.

GridBench is implemented in Java and Tomcat. The tool has been designed with extensibility in mind, both in terms of easy integration of new tests and benchmarks and in terms of integration with new middleware. The system design is

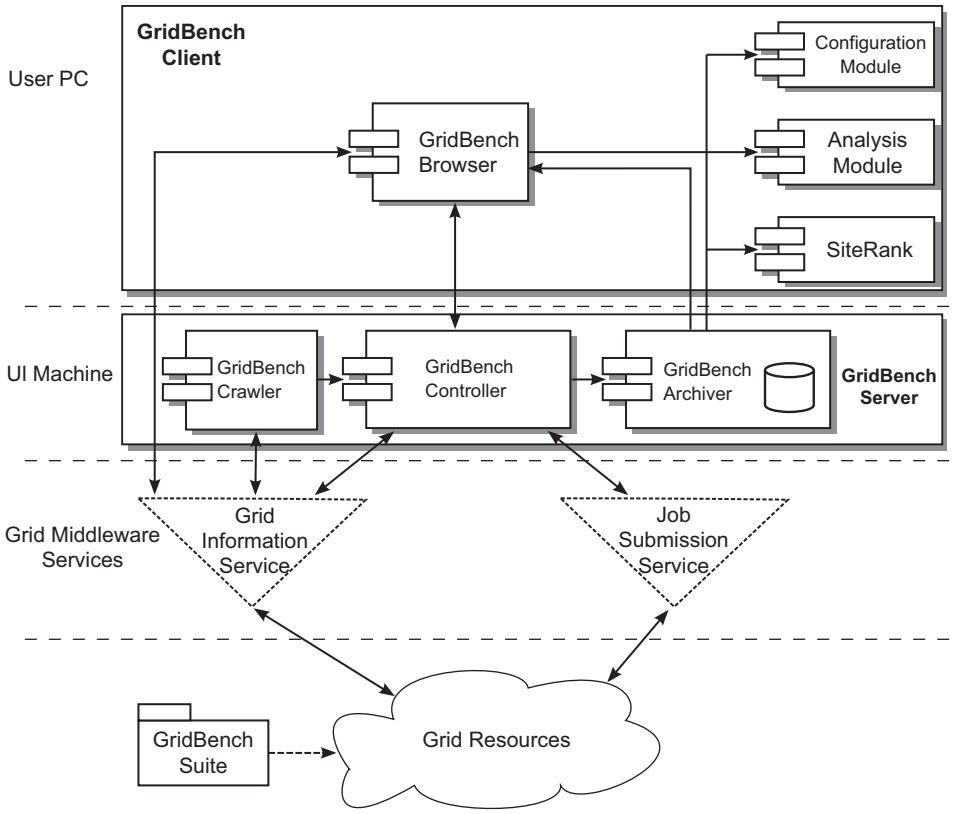


Fig. 5. GridBench components.

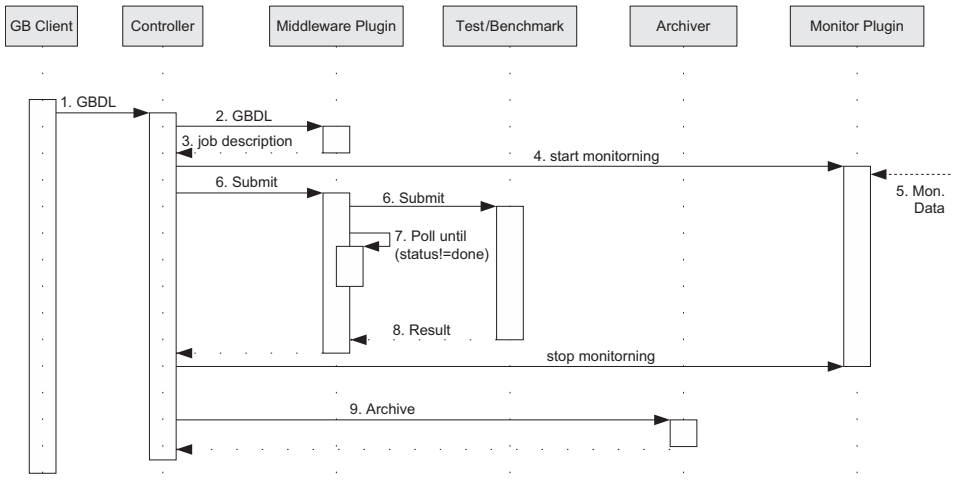


Fig. 6. A UML sequence diagram describing the basic Controller functionality.

modular and makes extensive use of plug-ins to provide integration with various middleware.

4.1. Server-side components

4.1.1. The GridBench controller Web-service

The *Controller* component has the task of managing test and benchmark executions. Most of the Controller's functionality is implemented in the form of *middleware plugins*.

The diagram in Fig. 6 describes the Controller functionality in a series of steps. The steps are given below (the numbers correspond to the numbered arrows in the UML diagram): (1) The Controller receives a benchmark description in the form of GBDL. This will originate from the Client or the Crawler; (2,3) The *Middleware Plugin* translates the GBDL to a middleware-specific job description, which is in the syntax and format required by the underlying middleware; (4) The Controller determines all monitoring that needs to be performed,

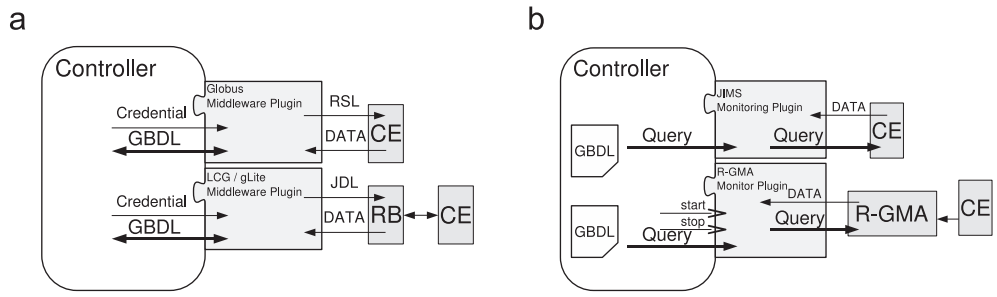


Fig. 7. *Left:* Middleware plugin functionality. *Right:* Monitor plugin functionality.

which is specified by the *monitor* elements 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 test or benchmark 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. Depending on the type of monitoring, steps 4 and 5 may come after step 6; (6) The benchmark job is then submitted using the *Middleware plugin*; (7) The job status is monitored by polling until the job finishes; (8) The results of the benchmark in the form of *metric* elements and its associated monitoring data are incorporated into the GBDL. If the *resource* name was not specified explicitly in the test specification, the *resource* element is also updated, indicating the exact location where the job ran; (9) Finally, the resulting GBDL is passed to the Archiver, concluding the Controller's role as it relates to this specific execution.

Middleware plugins (Fig. 7(a)) allow the GridBench framework to work with different underlying middleware. It is assumed that the underlying middleware supports basic operations such as copying (staging) files, submitting a job for execution and retrieving the result. The plugins mainly deal with (i) job description compilation (e.g. RSL for Globus), (ii) job submission and job-status monitoring, (iii) file staging and (iv) result retrieval. Plugins for Globus and the LCG/gLite middleware are provided in the current implementation.

Monitor plugins (Fig. 7(b)) are connectors to existing monitoring systems. They are employed by the *Controller* to collect monitoring information during test/benchmark execution. The user specifies what monitoring data is to be collected, as well as the start-time and finish-time of data collection. The start and end-times can be absolute times, or relative to the start and end-times of the actual test/benchmark execution. Initially, the GBDL contains a monitoring system-specific query that the monitoring plugin can interpret and connect to the monitoring system. In the absence of monitoring systems, the tool allows the collection of certain system attributes locally on the target resource (such as CPU-load and swap usage) during execution of the benchmark job.

The GridBench 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 provides a simple interface that includes: (i) storing a GBDL document, (ii) retrieving a GBDL document by it's ID, and (iii) retrieving a set of results based on an SQL query. The XML is transformed to relational data, where elements in the XML become records in the database tables by a simple mapping.

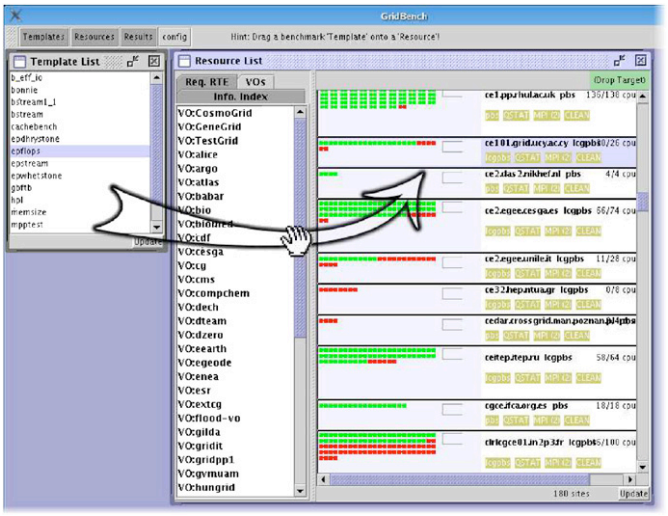


Fig. 8. The list on the left is a list of tests/benchmarks that are integrated into GridBench. The list on the right shows the currently available resources and their status in terms of busy/free CPUs.

The GridBench crawler: In many cases it is desirable to have the system perform tests and simple benchmarks periodically and make the results available to users. This provides an initial body of results, which the user can complement with her own executions. This allows for more meaningful analysis of the availability and dependability of resources, since the measurements taken by users are generally too sparse to be very useful in this context. The crawler monitors the Grid Information Services for the list of resources and periodically invokes tests on the ones that are available. The crawler runs as a daemon and needs to be provided with: (i) credentials with which to invoke tests, (ii) a set of GBDL definitions that are to be invoked periodically, and (iii) the period with which they are invoked. Benchmarks and tests submitted by the crawler are handled by the GridBench Controller as regular benchmark submissions.

4.2. Client-side components

4.2.1. The GridBench browser

In order to facilitate interactive, on-demand testing and benchmarking, GridBench provides a user-friendly graphical interface that simplifies the definition and execution of benchmarks and tests, as well as the browsing of results. Additionally, it provides tools for result analysis through the easy construction of custom graphs from archived results. Fig. 8 shows the main graphical UI for the definition of benchmarks, where we can observe the list of available tests/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, with details about each resource's composition, such as free/busy CPUs and WN, dual/single CPU machines, etc. Additionally a set of tests can be performed on each resource. In Fig. 8 we can see tests such as the "Queue", "QSTAT" and "MPI" tests. Tests involving multiple sites (e.g. using MPICH-G2) can also be performed. Such tests are useful for detecting configuration problems as well as connectivity/firewall issues. More tests (e.g. targeting other local queuing systems) can be easily added by implementing *CE test-plugins*.

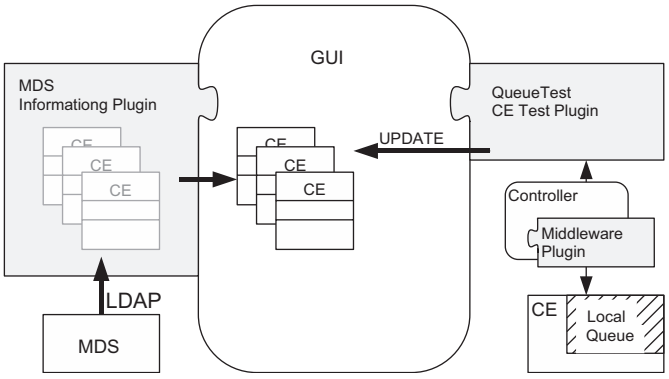


Fig. 9. Information plugin and CE test-plugin functionality.

CE test-plugins (Fig. 9) encapsulate invocations of tests in order to (i) integrate them directly into the GridBench Browser (shown in Fig. 10(b)), and (ii) provide a level of abstraction of local job manager systems employed by different Grid sites. For example, querying for the queue status at a site with a specific local job manager (such as PBS or LSF) requires a specifically crafted GBDL. The *test-plugin* determines the type of local job manager and based on that, submits the right GBDL to the site. The *test-plugin* also updates the status of a test in the GridBench Browser by updating the ⟨*status*⟩ element in the GBDL, to show whether the test is successful, failed or pending. CE test-plugins can be added to the browser by implementing a simple Java interface.

Information plugins (Fig. 9) are used by the GUI to retrieve information about Grid resources. The information plugin retrieves data from Grid Information Systems and populates a data-structure (CE objects) that holds information about resources. The CE objects contain a subset of the information specified in the GLUE schema for MDS [1], as well as additional information about individual WN (such as state and number of CPUs in each Worker Node). The CE objects are then used for rendering resource information on the GUI. The CE objects are also used to automate the creation of GBDL definitions, since they contain details on site configuration (e.g. configuration of CPUs on WN and local queue type).

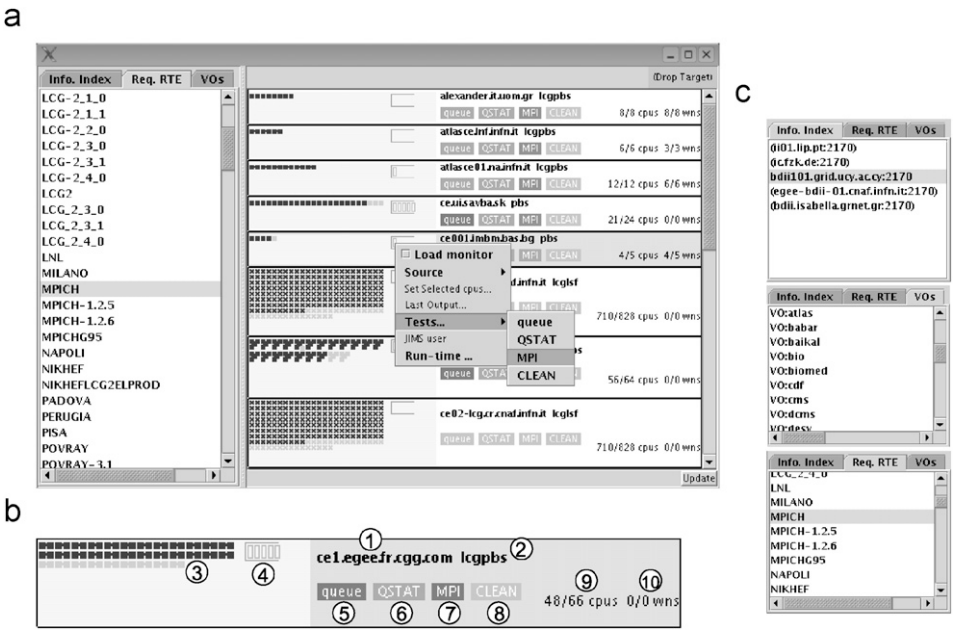


Fig. 10. (a) Resource browser, showing the state of resources from the EGEE test-bed that "advertise" support for MPI (MPICH run-time environment); (b) the resource renderer; (c) *top*: information index sources selection. *Middle*: querying for specific Virtual Organizations. *Bottom:* specifying run-time environment support.

Information displayed in the *resource browser* (Fig. 10(a)) is obtained by querying one or more information systems (Fig. 10(c)-top). The information for each resource is displayed in graphical form and contains a rendering of the status of CPUs and the configuration of CPUs into WN. Fig. 10(b) shows the resource renderer where (1) is the resource name; (2) is the local queue type, and (3) shows CPU organization and status; (4) shows the default queue status; (5) shows a test in progress (blue); (6) shows a successful test (green); (7) shows a failed test (red); (8) shows a test not run; (9) shows free/total CPUs; and (10) shows free/total WNs (updated after the invocation of the "queue test" *CE test*-plugin). In the rendering of the status of the local resource queue, the user's jobs are shown in a different color. The different test-plugins and their status are displayed in real time, enhancing interactivity. The resource browser allows for multiple selections of resources to act as drop-targets for invoking a benchmark or test on a set of resources. The browser allows for the user limit her view based on VO (Fig. 10(c)-middle), or support of a specific run-time environment (Fig. 10(c)-bottom). Defining and executing a test or benchmark is as easy as dragging a benchmark onto one of the resources (shown in Fig. 8).

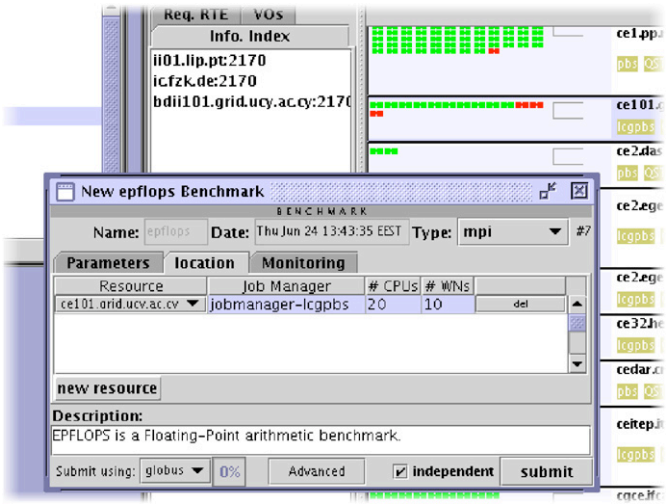


Fig. 11. Benchmark configuration panel.

The configuration module: The heterogeneity of resources sometimes requires that the test is tailored to the specific resource under test. The tuning may be quite trivial, such as setting the number of CPUs to be used, or it can be more complex such as setting up several memory or network parameters to configure a benchmark. The user has the opportunity to tune test parameters prior to execution via a configuration panel (Fig. 11). The benchmark configuration panel is a GUI front-end that customizes a GBDL template document. Namely, the module allows the tuning of parameters to the benchmark, the definition of the target resource(s) and the definition of what is to be monitored during execution.

The analysis module: The user can easily construct graphs as the ones in the results section by browsing results (metrics) previously archived in the database (Fig. 12). The interface enables the user to perform custom queries on the archived results and to select the results that are relevant and of interest. The user can then use the metrics included in the results to interactively build charts. The analysis/graphing module can handle several metric types and present each metric on an appropriate chart.

5. SiteRank

The GridBench tool provides the *SiteRank* module that allows the user to interactively build and employ a *ranking model*, for ranking resources. A *ranking model* consists of *filtering*, which is basically the selection of a subset of the results, *aggregation*, which is the combination of single metrics into higher-level metrics and finally, a *ranking function* (Fig. 13).

Filtering refers to a user selection regarding which metrics should be taken into account in the ranking process. The user

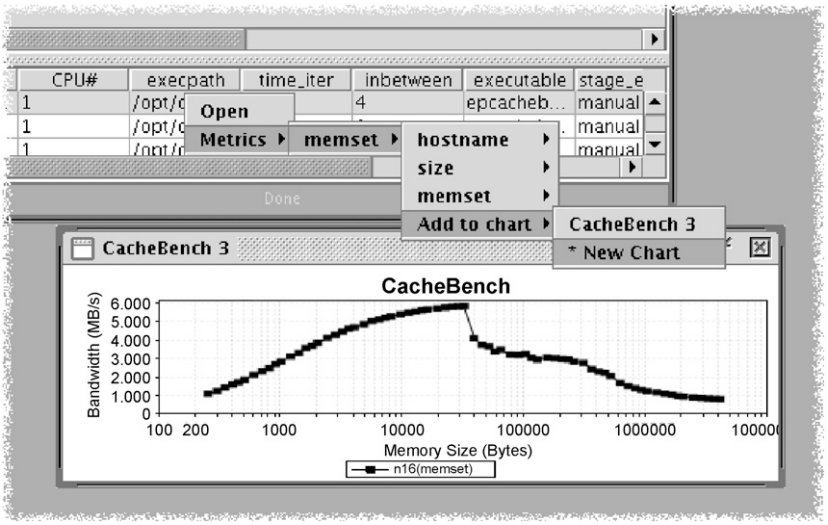


Fig. 12. Generation of charts from historical data. The result shown is from a memory cache benchmark.

selection can be based on any of the following types of filtering: (i) *attribute filtering*, (ii) *monitoring-based filtering* and (iii) *filtering based on metric quality*. Attribute filtering is done by specifying an SQL query from within the GridBench Browser, which allows the user to limit the selected set of measurements to the ones that match certain criteria in the benchmark description. For example, the user can limit the selection to a specific VO or to a specific type of CPU. The user can also limit results based on the date and time they were obtained, thus limiting the selection to recent results. Monitoring collected during each benchmark execution allows the user to inspect the state of the worker node during the execution, and determine whether to manually exclude measurements that are deemed not fit. Finally, the user can choose to filter results based on the quality of the metrics by requiring a minimum number of execution repetitions or a minimum sample ratio.

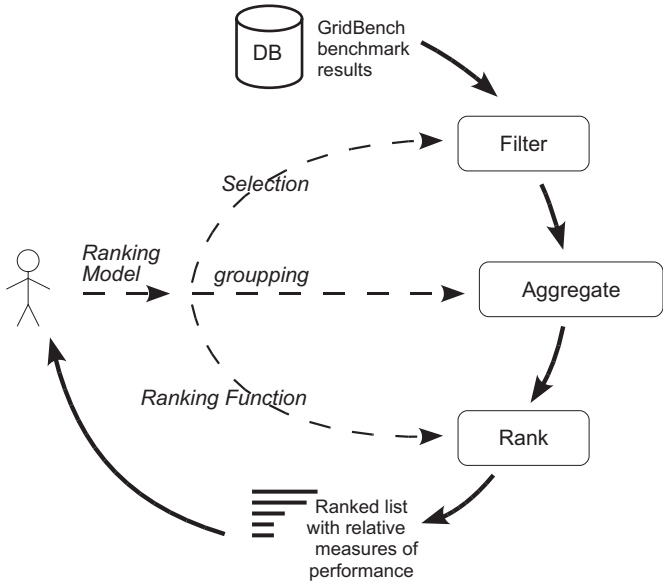


Fig. 13. The ranking process.

Aggregation allows the user to specify a grouping of the selected measurements. The user can specify whether each measurement will count equally, irrespective of which worker node it was executed on. In this case, the reported metric may be less representative of the resource as a whole because some WN may be over-represented. On the other hand, this will tend to be more representative of what users actually experience once the resource's policy is applied. The *Aggregation* step produces a set of statistics for each metric: *mean*, *standard-deviation*, *min*, *max*, *average-deviation* and *count*. During the aggregation step, the raw metrics are normalized according to configurable base values; for the experiments presented in this paper, we used as reference a 3.0 GHz xeon worker node. The aggregation step is also important for the conversion of vector-type metrics into scalars so that they can be used in ranking functions. An example of a vector-type metric would be the raw MPI round-trip-time measurements at different packet sizes, which can be summarized into latency or bandwidth.

Last, but not least, a user can invoke the SiteRank module of GridBench to specify, customize, and/or invoke a *ranking function* that computes a ranking of Grid resources according to user preferences. For instance, the user can specify the values (weights) of a set of coefficients, which is combined with a set of aggregated metrics into a customized linear model for resource ranking. This process can be accomplished interactively through a module of the GridBench Browser (see Fig. 14): the user chooses experiments from the Analysis Module, selects the metrics of interest, adds them to a chart, and specifies the desired metric coefficients. The ranking function is evaluated

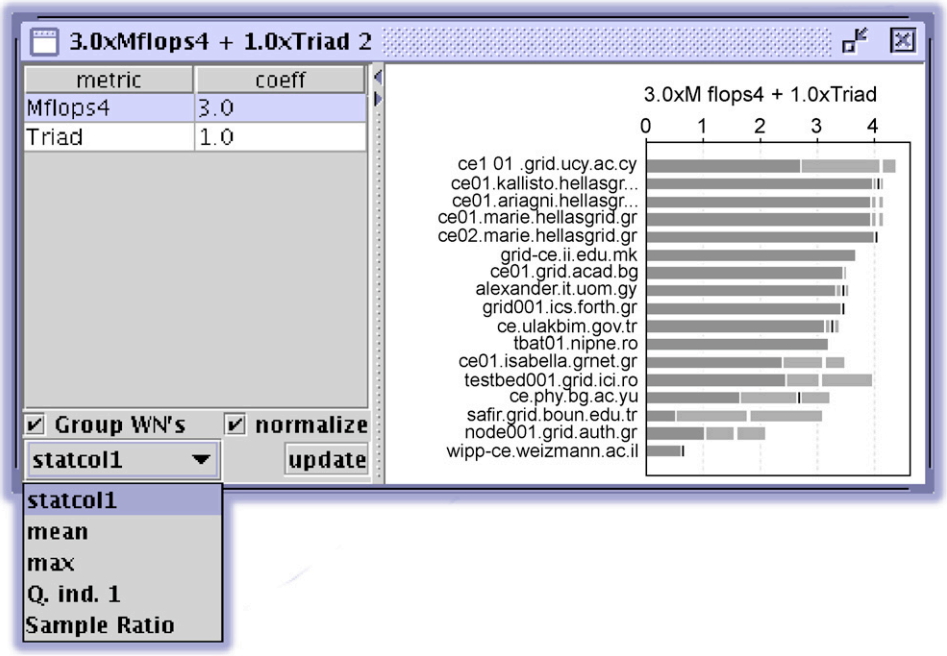


Fig. 14. The ranking module dialog.

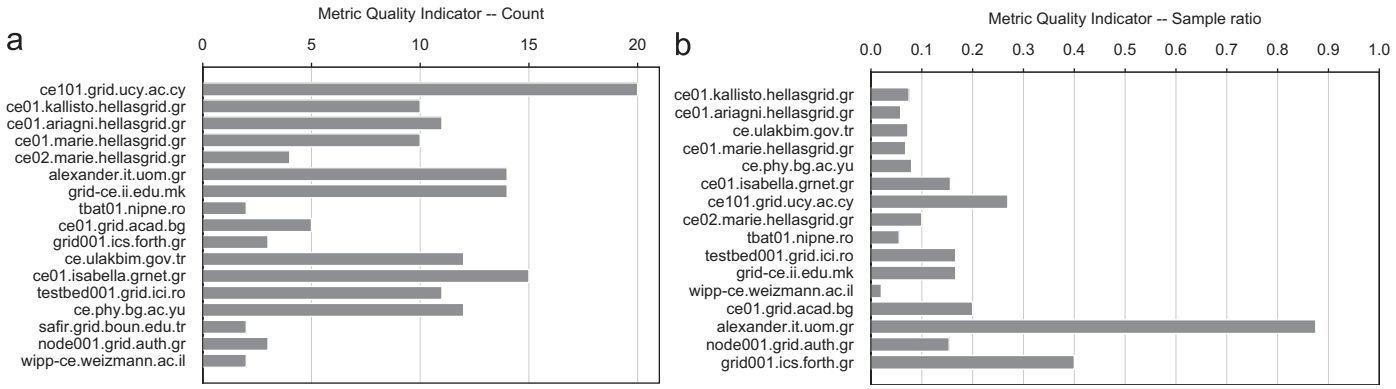


Fig. 15. Mflops4 metric quality indicators.

for each resource that appears in the initial result selection. The resources are then presented in a ranked order along with their respective score. More complex models can be specified programmatically and easily integrated into SiteRank. Besides the aggregated metrics, ranking models can take into account the various Quality Metric Indicators computed by GridBench (see below).

Metric quality: Directly related to the issue of internal heterogeneity of resources is the issue of the quality of the obtained measurements. Due to the nature of the underlying infrastructure, which prohibits targeted measurements of specific WN, the WN that are actually measured are in fact a "sample" of what is available at a given resource. This sample, which in many occasions is *not* random, follows the policy that the resource applies to the given user/VO. In order to have a better idea of how accurate the reported metric is and how representative the sample is, the tool provides two indicators of *metric quality*. The first metric quality indicator is the *count*, which is the number of measurements that are used in calculating the reported value of each metric. Fig. 15(a) shows the *count* metric quality indicator for the *Mflops4* metric. The second quality indicator is the *Sample ratio* which reflects just how representative of the whole resource the currently available measurements are. Fig. 15(b) shows the *sample ratio* metric quality indicator for the *Mflops4* metric. This value is a real number that can range from 0 to 1; it is usually easier for resources with fewer WN to have a high *sample ratio* value since they require fewer measurements. Both metric quality indicators are subject to the availability of resources and to the extent to which the resources are busy. In some cases the quota for a specific VO is used up, which also leads to resource unavailability.

6. Experiments

The GridBench software has been released for testing and experimentation, and has been used by application developers and infrastructure operators on top of production-level, large-scale Grid infrastructures, such as CrossGrid, GridIreland and EGEE. In this section, we describe a number of different use-cases that we demonstrated with GridBench.

6.1. Application performance

In the first scenario, GridBench users are interested in deploying high-performance computing applications on a Grid. To this end, they seek to explore the relative performance and scalability of different Grid resources, according to the performance behavior of the computationally intensive kernels of their applications. To explore this scenario in the context of the CrossGrid testbed, we used kernels extracted from three real scientific applications deployed on CrossGrid [11]:

1. *High-energy physics ANN training*: This kernel is taken from a parallel Artificial Neural Network training application. The architecture of the code is based on a client–server model and the code is loosely coupled [17].
2. *Air pollution simulation*: The VERTLQ kernel comes from the STEM-II Eulerian numerical model that is used for the simulation of air pollutant factors. We have used the parallel (very tightly coupled) version of the code [15].
3. *Blood-flow simulation*: The "bstream" kernel is extracted from a medical application for pre-operative planning of vascular reconstruction. It is a tightly coupled code that involves blood-flow simulation using a Lattice Boltzmann method in 3-D artery models [21].

One of the primary design goals of GridBench is the easy inclusion of new tests, benchmarks or kernels. The required steps are: (i) create a new GBDL description template and add it to the Archiver database; (ii) create a "parameter handler" (usually a simple shell script); (iii) *optionally* instrument the code of the kernel to generate additional metrics. The following is the new GBDL description template required to integrate "bstream" into GridBench:

```
<testmark name="bstream1_1">
  <parameter name="iterations"
   type="user">40</parameter>
  <parameter name="Reynolds"
   type="user">20</parameter>
  <parameter name="data_id"
   type="user">tube38x40x40</parameter>
</testmark>
```

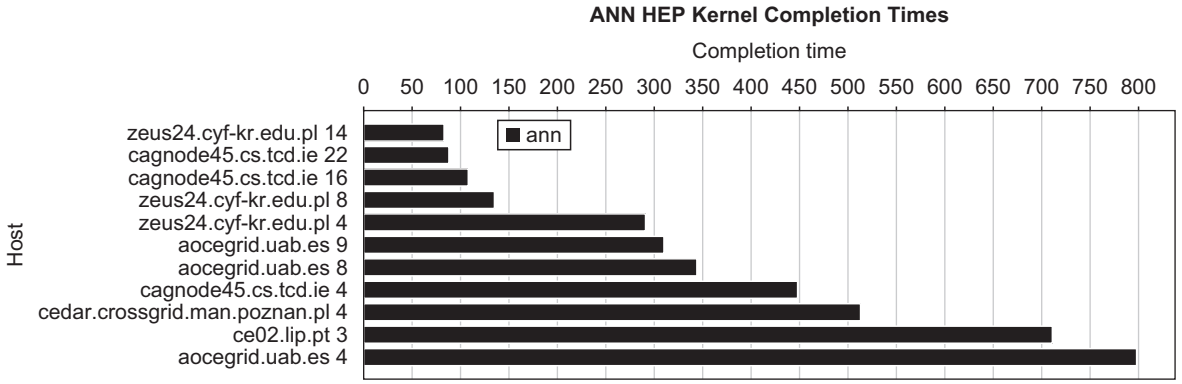


Fig. 16. Results for the parallel Artificial Neural Network training application kernel. The number of CPUs is indicated next to the resource name and the completion times are sorted (best-performing first).

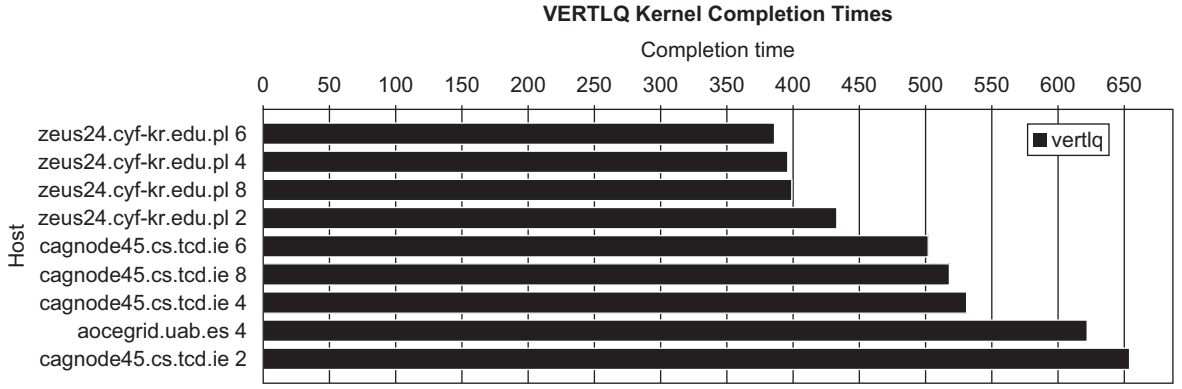


Fig. 17. Results for the VERTLQ kernel from the air pollution simulation application. The number of CPUs is indicated next to the resource name and the completion times are sorted (best-performing first).

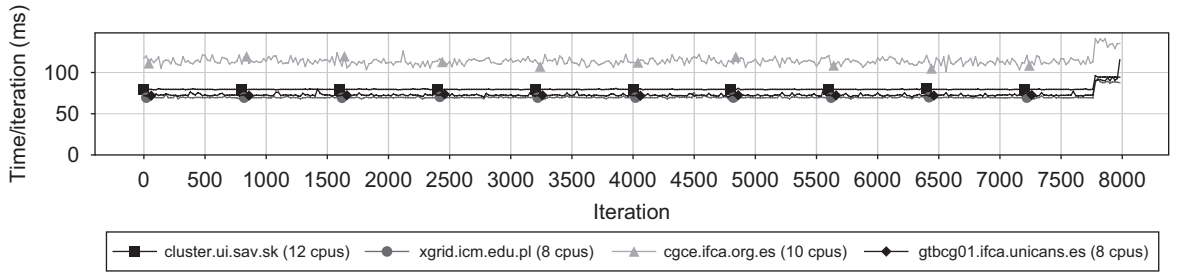


Fig. 18. Results for the "bstream" kernel, showing iteration times on a set of four resources.

This description specifies the *iterations*, *Reynolds* and *data_id* application-specific parameters. Once this specification is inserted into GridBench, the user can invoke it through the GridBench Browser by dragging the newly created template onto a set of resources. The general steps taken: (1) retrieve previously archived results for this kernel; (2) benchmark the resources for which there are no archived results; (3) compare the results (Figs. 16–18).

If the user wishes to study the scalability of different Grid clusters for the particular application of interest, he just has to invoke the corresponding kernel–benchmark on a subset of the available resources, using different CPU-count parameters. The resulting metrics are archived in the database (along with possibly pre-existing metrics) and made available for analysis. Fig. 16 shows results from the ANN kernel while Fig. 17 shows results from VERTLQ. The results are sorted by completion time—i.e., best performance—thus effectively providing a ranking of the resources. The number of CPUs used is indicated next to the resource name. The user can then make decisions based on these results and answer questions like "*Should I use* 4 *CPUs or* 8 *CPUs from site A*?", or "*Should I use* 4 *CPUs from site A or* 6 *CPUs from site B*?".

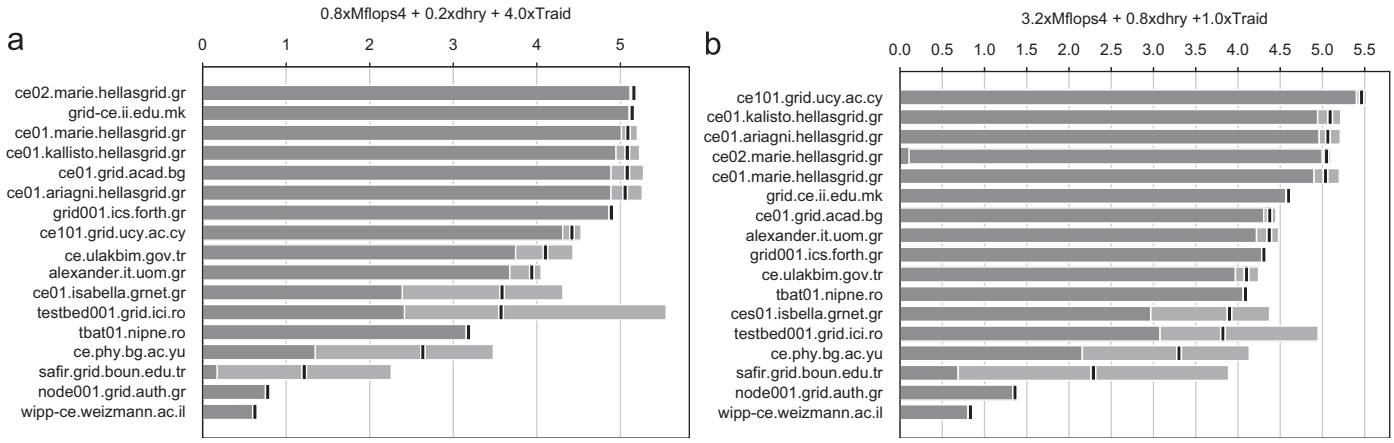


Fig. 19. Ranking of SE Europe resources by putting more emphasis on CPU or main-memory performance.

If additional metrics are desired, the instrumentation of codes is application-specific and usually involves trivial modification of the source code to obtain timings at a high level. In the case of the "bstream" kernel, the application performs iterations which are controlled by a main loop. In total, about 10 lines of code were added in order to time each iteration and output the following metrics onto the standard output (in addition to the default "completion time" metric):

```
<metric name="iteration_times"
 node="cluster.ui.sav.sk">
  <value unit="s">0.079617 0.079529 0.079511
   0.079498 ... 0.094326</value>
</metric>
```

6.2. User-driven resource ranking

In the second scenario we consider two users, each with a different application and different requirements in terms of performance. They require a performance-ranked set of resources[1] tailored to their needs. User *A* has an application that heavily relies on *memory* performance, while user *B* has an application that relies heavily on *CPU* performance. In terms of low-level CPU and memory metrics, the tool provides *Mflops4* and *dhry* for floating-point and integer CPU performance, respectively, and *Triad* for main memory performance. Utilizing measurements stored in the archive, each user can interactively construct a *ranking function* from within the GridBench GUI (see Fig. 14). Obtaining the right coefficients to determine the weight of each metric in an application-specific ranking is described in detail in [25]. For the purpose of this scenario we assume that the user has some insight as to how the different low-level metrics relate to the performance of the application at hand.

User *A* with a preference on memory performance assigns higher coefficients for memory and lower coefficients for CPU, producing a ranking function R_A:

$$R_A = 0.8 \cdot Mflops4_{\text{mean}} + 0.2 \cdot dhry_{\text{mean}} + 4.0 \cdot Triad_{\text{mean}}.$$

User *B*, on the other hand, chooses to put more weight on CPU rather than on memory, producing a different ranking function R_B:

$$R_B = 3.2 \cdot Mflops4_{\text{mean}} + 0.8 \cdot dhry_{\text{mean}} + 1.0 \cdot Triad_{\text{mean}}.$$

In R_A the memory metric is given four times the weight of the CPU metrics $(0.8 + 0.2)$. The opposite is used in R_B. Fig. 19(a) is the result of ranking function R_A, while Fig. 19(b) is the result of the modified ranking function R_B. The resulting composite performance metric varies considerably; a resource that ranks eighth in R_A, ranks first in R_B.

7. Related work

A number of tools proposed in the recent literature focus on performance monitoring and benchmarking of Grid and network-centric infrastructures. In contrast to generic infrastructure monitoring tools, which typically read, collect, and report machine attributes, performance monitoring refers to the active invocation of tests and/or workloads for the controlled measurement of Grid systems. Several performance monitoring tools have been reported in recent literature. For instance, the *Grid assessment probes* system (GRASP) [4], tests and measures the performance of basic Grid functions such as job submission, file transfers, and performance of Grid Information Services. GRASP aims to test basic Grid functionality, so in that sense it is rather a "testing" than a performance measurement tool. *DiPerF* is a distributed performance-testing framework that aims to automate *service* performance evaluation [8], by providing mechanisms for coordinating a pool of machines that test a target service. DiPerf collects and aggregates performance metrics, and generates performance statistics for service "fairness" and service throughput. Employing an architecture similar to DiPerF, the *Inca test harness and reporting framework* [22] is a system that automates the testing of resources

[1] The results shown are taken from the EGEE South-East Europe Region.

and resource data collection, and supports resource verification and service-agreement monitoring. Inca comprises a central *controller* and a set of *reporters*. Additionally, the Inca framework includes a *distributed controller* which resides on the Grid resource (e.g. periodic measurements are handled locally and need not involve the centralized controller). Inca is geared towards its underlying Grid infrastructure and employs middleware-specific tests that assess job-submission, file replication, accessibility of services, etc.

Infrastructure monitoring tools, like the *network weather service* [27] (NWS), can provide useful real-time information of several system aspects, including network latency and bandwidth between distributed Grid resources. Such systems, however, do not address computational performance of the monitored resources. For instance, NWS has "CPU sensors" which provide measurements of CPU "availability." However, this is different from benchmarking, as it provides a snapshot of the "instantaneous" performance capacity of a machine, and not what can be expected of the machine in terms of performance. NWS is best known for its forecasting capabilities which are based on past measurements. Last, but not least, the *ALU intensive Grid benchmarks* [26] (AIGB) are paper-and-pencil specifications of reference benchmarks proposed for measuring the overall performance of Grid infrastructures. AIGB benchmarks are essentially synthetic applications, synthesized as workflows of traditional benchmarks adopted from the NAS Parallel Benchmark suite [3].

In contrast to the systems presented above, GridBench emphasizes mainly on providing an *interactive*, *user-oriented*, and *easily customized* environment for *end-to-end* tests and performance measurements of Grid resources. Interactivity is provided via a powerful graphical UI for specifying tests and for analyzing results. Customization is supported by an underlying XML schema, which enables also the virtualization of tests, and the easy integration of test results with information retrieved from other (monitoring) sources. In contrast to typical monitoring systems, where measurements are mostly set up by a central administrator and users can simply review tabular or graphical representations of measurements, GridBench allows end-users to easily customize tests, data visualization and analysis according to their own requirements. Furthermore, GridBench adopts an end-to-end approach in testing and benchmarking Grid infrastructures; consequently, the deployment and management of GridBench tests does not require any special software installations or updates of the middleware. This can be a significant advantage when testing large-scale, production-rate Grid facilities. Moreover, this approach allows end-users to explore and understand the effects that resource virtualization has on the application performance experienced by end-users. In addition to performance metrics, GridBench provides quality indicators that can be used to assess the quality of measurements and the effects of virtualization upon these measurements. Last, but not least, the collection and organization of test results, metrics, and quality indicators, facilitates the specification, customization, and deployment of different ranking functions that can rank Grid resources according to individual end-user requirements. It is worth noting that the current GridBench deployment comprises a suite of open-source benchmarks of different granularity, which measure different aspects of resource performance [24]; nevertheless, the extension of this suite is quite easy and does not require any changes in the underlying middleware or GridBench installation.

8. Conclusions and future work

We have developed a system that aims to alleviate many of the complexities of testing and benchmarking large number of Grid resources. We have provided an overview of the GridBench tool, which can serve as a "virtual workbench" for performing test/benchmark experiments easily and interactively. The tool—extensible via plugin mechanisms—allows for submitting tests and benchmarks, archiving specifications and results, and serves as an aid for the analysis of the resulting metrics. The GridBench tool puts a set of tests, micro-benchmarks and kernel benchmarks at the users disposal and allows the easy tuning of the existing tests/benchmarks and the easy addition of new ones. Users can drag-and-drop tests/benchmarks onto a graphical and dynamic representation of Grid resources, get feedback on the progress of execution, keep track of the quality of the reported metrics, and combine fresh with historical results into charts, all from the same UI. Though the tool's ranking module the user can interactively follow a three-step process—*filter*, *aggregate*, *rank*—that allows for flexible, user-driven ranking of resources.

Through the presented use-case scenarios, we have illustrated the functionality and ease of use of the tool: the first use-case illustrates how a user or application developer can obtain results from new application-based benchmarks using the GridBench framework. In the second use-case we provide a short glimpse on the functionality of the tool in terms of interactive, user-driven ranking of resources using ranking functions.

These tasks would simply not be feasible if they had to be done manually, especially if they had to be done by the end-user. Users can now use this approach in the search for the right resources on which to run their application, by testing and ranking resources by what *they* consider important in terms of functionality or performance. Furthermore, users can verify the "advertised" performance of a resource by running light-weight benchmarks. Eventually, resource performance information will be coupled with resource pricing information. Users will then be able to "shop around" and pick the right resources, using simple ranking functions/models that account for pricing information, in order to influence the matchmaking process is a way that benefits them.

In this paper's introduction we have emphasized at least four challenges that need to be addressed, i.e., *Scale and complexity*, *Volatility*, *Heterogeneity* and *Virtualization*. GridBench, by simplifying the process of executing tests and benchmarks on a large number of resources, as well as the coupling of definitions, results and metadata, can be used effectively on infrastructures of a large scale and complexity. The same functionality allows for easy repeated collection of measurements and associated metadata thus alleviating the problem of volatility. The end-to-end approach to measuring the system,

addresses the issues raised by the virtualization and the different views of VOs with respect to the infrastructure. The heterogeniety of different resources is tackled by (i) using simple, architecture-independent benchmarks, and (ii) allowing for the tailoring of benchmarks to the specific resource's characteristics.

In on-going and future work we are working on approaches to further automate the SiteRank module so that *ranking models* are automatically generated based on results from a given application. This would eliminate the expertise required from the user's side and that ultimately, this would establish relationships between benchmark results and the performance of user applications, with minimal user involvement.

We are working on addressing the important issues 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. The open Grid services architecture (OGSA) [10] effort has recently introduced a relevant service—the "Candidate Set Generator". This emphasizes the importance of resource selection and sets the general guidelines for such a service; we plan to develop a service along these lines. 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] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J.M. Schopf, M. Viljoen, A. Wilson, GLUE Schema Specification—Version 1.2, December 2005.

[2] G. Avellino, S. Beco, B. Cantalupo, A. Maraschini, F. Pacini, M. Sottilaro, A. Terracina, D. Colling, F. Giacomini, E. Ronchieri, A. Gianelle, M. Mazzucato, R. Peluso, M. Sgaravatto, A. Guarise, R.M. Piro, A. Werbrouck, D. Kouril, A. Krenek, L. Matyska, M. Mulac, J. Pospísil, M. Ruda, Z. Salvet, J. Sitera, J. Skrabal, M. Vocu, M. Mezzadri, F. Prelz, S. Monforte, M. Pappalardo, The DataGrid workload management system: challenges and results, J. Grid Comput. 2 (4) (2004) 353–367.

[3] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, D. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, S.K. Weeratunga, The NAS parallel benchmarks, Internat. J. Supercomputer Appl. 5 (3) (Fall 1991) 63–73.

[4] G. Chun, H. Dail, H. Casanova, A. Snavely, Benchmark probes for grid assessment, in: 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM/Abstracts Proceedings, Santa Fe, New Mexico, USA, 26–30 April 2004, IEEE Computer Society, 2004.

[5] A. Cooke, A.J.G. Gray, L. Ma, et al., R-GMA: an information integration system for grid monitoring, in: On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science, vol. 2888, Springer, Berlin, 2003, pp. 462–481.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), IEEE Computer Society, 2001, pp. 181–194.

[7] M.D. Dikaiakos, Grid Benchmarking: vision, challenges and current status, Concurrency Comput.: Pract. Experience 19 (1) (2007) 89–105.

[8] C. Dumitrescu, I. Raicu, M. Ripeanu, I. Foster, Diperf: an automated distributed performance testing framework, in: Proceedings of the Fifth International Workshop on Grid Computing (GRID2004), IEEE, November 2004.

[9] Enabling Grids for E-SciencE project, ⟨http://www.eu-egee.org/⟩.

[10] I. Foster, et al., The open grid services architecture, version 1.0, ⟨http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf⟩ (accessed January 2006), 2005.

[11] J. Gomes, M. David, et al., Experience with the international testbed in the crossgrid project, in: Advances in grid computing—EGC 2005, European Grid Conference, Amsterdam, The Netherlands. February 14–16, 2005, Revised Selected Papers, Lecture Notes in Computer Science, vol. 3470, Springer, Berlin, 2005, pp. 98–110.

[12] E. Kenny, B. Coghlan, G. Tsouloupas, M.D. Dikaiakos, J. Walsh, S. Childs, D. O'Callaghan, G. Quigley, Heterogeneous grid computing: issues and early benchmarks, in: Computational Science—ICCS 2005, Fifth International Conference, Atlanta, GA, USA, May 22–25, 2005, Proceedings, Part III, vol. 3516, Springer, Berlin, 2005, pp. 870–874.

[13] M. Kupczyk, R. Lichwala, N. Meyer, B. Palak, M. Póciennik, M. Stroinski, P. Wolniewicz, The migrating desktop as a gui framework for the "applications on demand" concept, in: International Conference on Computational Science, 2004, pp. 91–98.

[14] C. Liu, L. Yang, I. Foster, D. Angelo, Design and evaluation of a resource selection framework for grid applications, in: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02), IEEE Computer Society, 2002, pp. 63–72.

[15] J.C. Mouriño, D.E. Singh, M.J. Martín, J.M. Eiroa, F.F. Rivera, R. Doallo, J.D. Bruguera, Parallelization of the stem-ii air quality model, in: HPCN Europe, 2001, pp. 543–546.

[16] Open Science Grid, ⟨http://www.opensciencegrid.org⟩ (accessed September 2005).

[17] O. Ponce, et al., Training of neural networks: interactive possibilities in a distributed framework, in: D. Kranzlmüller et al. (Ed.), Ninth European PVM/MPI (LNCS), vol. 2474, Springer, Berlin, 2002, pp. 33–40.

[18] R. Raman, M. Livny, M.H. Solomon, Matchmaking: distributed resource management for high throughput computing, in: Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC '98), IEEE Computer Society, 1998, pp. 140–147.

[19] R. Raman, M. Livny, M.H. Solomon, Matchmaking: an extensible framework for distributed resource management, Cluster Comput. 2 (2) (1999) 129–138.

[20] H. Rasheed, Quantification of grid resource heterogeneity and impact on application performance, Master's Thesis, Department of Electronic, Computer and Software Systems, Royal Institute of Technology (KTH), Sweden, July 2006.

[21] P.M.A. Sloot, A. Tirado-Ramos, A.G. Hoekstra, M. Bubak, An interactive grid environment for non-invasive vascular reconstruction, in: Second 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.

[22] S. Smallen, C. Olschanowsky, K. Ericson, P. Beckman, J.M. Schopf, The inca test harness and reporting framework, in: SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, Washington, DC, USA, IEEE Computer Society, 2004, p. 55.

[23] G. Tsouloupas, M.D. Dikaiakos, Characterization of computational grid resources using low-level benchmarks, Technical Report TR-2004-5, Department of Computer Science, University of Cyprus, December 2004, ⟨http://grid.ucy.ac.cy/reports/TR-04-5.pdf⟩.

[24] G. Tsouloupas, M.D. Dikaiakos, Characterization of computational grid resources using low-level benchmarks, e-science 0 (2006) 70.

[25] G. Tsouloupas, M.D. Dikaiakos, Grid resource ranking using low-level performance measurements, Technical Report TR-07-02, Department of Computer Science, University of Cyprus, February 2007, ⟨http://grid.ucy.ac.cy/reports/TR-07-02.pdf⟩.

[26] R.F. Van der Wijngaart, M. Frumkin, Alu intensive grid benchmarks, ⟨http://forge.gridforum.org/projects/gb-rgs⟩, 2004.

[27] R. Wolski, N. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service in metacomputing, J. Future Generation Comput. Systems 15 (5–6) (1999) 757–768.

George Tsouloupas is a Ph.D. candidate at the Computer Science Department of the University of Cyprus. His research interests include distributed and parallel computing, with a focus on Grid computing. Tsouloupas completed his Master of Science at the Fairleigh Dickinson University in 2001 after completing a Double Major B.Sc. in Mathematics and Computer Science (Summa Cum Laude, Academic Deans Award, 1999). He has worked for several years at the High Performance Computing systems Laboratory at the University of Cyprus, contributing to several Grid-related research projects.

Marios D. Dikaiakos is an Associate Professor of Computer Science at the University of Cyprus and Head of the High-Performance Computing Systems Laboratory. Dikaiakos received his Ph.D. from Princeton University (1994), an M.A. degree from Princeton (1991), and a Dipl.-Ing. degree from the National Technical University of Athens (summa cum laude, 1988). He has been a Research Associate in the University of Washington, Seattle (1995, 1996) and a Visiting Research Professor in Rutgers University, USA (Spring 2005). He has been a principal investigator for several projects funded by the European Union and the Research Promotion Foundation of Cyprus, and regularly serves on the program committees of International scientific conferences. His research interests focus on network-centric computing systems, with an emphasis on Grids and the World Wide Web.