GridBench: A Tool for Benchmarking Grids*

George Tsouloupas Marios D. Dikaiakos
Dept. of Computer Science
University of Cyprus
1678, Nicosia, Cyprus
{georget,mdd}@ucy.ac.cy

Abstract

The aim of the GridBench suite of Benchmarks is to bring together a core set of benchmarks for characterizing Grid nodes or collections of Grid resources. In order to do this in an organized and flexible way we provide a framework for running benchmarks on Grid environments as well as collecting, archiving, and publishing the results. This framework allows for convenient integration of new and existing benchmarks into the suite.

1. Introduction

The experimental performance evaluation of parallel computer systems relies typically on benchmarks, such as Linpack [10], NAS [3], Parkbench [14], Splash [23]. Benchmarks provide a commonly accepted basis for comparing different computer systems in terms of their performance [9]. They are also used to investigate performance properties of parallel systems under carefully tuned, benchmark-induced workloads that stress particular aspects of system performance. Benchmarks can also be helpful for predicting the performance and scalability of a category of complex applications on a given system; to this end, kernels representative of this category of applications are produced and executed to estimate overall application performance. Performance evaluation based on benchmarks often employs codes and workloads that represent different types of computation and are developed with various programming paradigms and tools. Such a diversity is crucial for the performance assessment of general-purpose parallel systems in the context of different use-case scenarios and workload types. Benchmarking has also been used for assessing the performance of software systems connected on Internet, such as Web infrastructures [15, 21, 19], mobile-agent middleware [8], etc.

The emergence of Grids as platforms for high-performance and high-throughput computing over the Internet raises many questions regarding the experimental performance evaluation of Grid infrastructures. A key issue is whether the parallel-benchmark paradigm can be ported and used "as is" in a Grid setting. This issue relates to a number of questions about the applicability of benchmarks on Grids, the meaning of benchmark results and their interpretation, the support required to administer Grid benchmarks and access their results, etc. Our conjecture is that parallel benchmarks and their context of use cannot be ported directly to the Grid.

Nevertheless, benchmarking can be very useful for Grid users. Benchmarking metrics published on the Grid can provide a basis for users to assess the "quality of service" expected by 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 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 goal of our work is to design and develop **GridBench**, a tool for researchers that wish to investigate

*This work was supported by the European Union as part of the CrossGrid project under contract IST-2001-32243.

various aspects of Grid performance using a variety of well-understood benchmarks that range from simple micro-benchmarks to kernels representative of more complex Grid applications. By providing access to a corpus of such kernels and facilitating the specification and dispatch of parameterized kernel runs on Grids, GridBench will enable the characterization of factors that affect application and infrastructure performance, the quantitative comparison of different middleware solutions, algorithms for scheduling, resource allocation, etc.

The rest of this paper is organized as follows: in Section 2, we discuss the scope and challenges of Grid benchmarking and propose a hierarchical framework for addressing these issues. In Section 3, we describe the design of GridBench, a tool that we are implementing for specifying and administering Grid benchmarks. Section 4 describes a core set of benchmarks included in the first version of GridBench and presents our experiences from our implementation and testing efforts so far. We conclude in Section 5.

2. Benchmarking the Grid

2.1 Challenges and Scope

In order to produce valid and usable results, Grid benchmarking must abide by the general rules of scientific experimentation. Therefore, Grid middleware and benchmarking tools should provide the necessary support for conducting controlled and reproducible experiments, for measuring performance indicators accurately, and for interpreting benchmarking results in a proper context.

For the purpose of our work, we consider Grid benchmarks as well-understood and relatively simple codes that can be deployed and run on top of Grid resources to:

- Generate metrics that describe the performance capacity of selected *Grid nodes in isolation* through measurements of computational power, file-transfer speed, inter-process communication bandwidth, scalability, etc. Consolidation and proper storage of these metrics can support advanced resource brokers and job schedulers for the Grid. Moreover, it can provide a basis for performance-aware resource allocation and Virtual-Organization formation.
- Generate metrics that characterize the performance capacity of resources belonging to a Virtual Organization and *spanning across multiple Grid nodes*, in terms of computational power, file-transfer speed, inter-process communication bandwidth, application-kernel performance, scalability etc. It should be noted that the proper interpretation of these metrics dictates the capability of associating them with adequate descriptions of the actual sets of resources allocated to the particular benchmark-executions that produce metrics data.
- Generate metrics that characterize the performance of *Grid middleware services* available at the programming level through API's.
- Provide a *tool* for researchers that wish to investigate various aspects of Grid performance, using well-understood kernels that are representative of more complex applications deployed on the Grid. Having access to a corpus of such kernels and being able to easily specify and dispatch parameterized runs of these kernels on Grids, facilitates the characterization of factors that affect application and infrastructure performance, the quantitative comparison of different middleware solutions, algorithms for scheduling, resource allocation, etc.
- Test emerging Grid middleware services to feed performance prediction tools with measured parameters and to validate performance-prediction approaches.

For the definition of benchmarks and the interpretation of their results at a conceptual level close to the users' perception of the Grid, we need models capturing the basic characteristics of benchmarks and of the underlying architecture reserved for benchmark executions. Such models can be used to abstract low-level implementation details and measurements, while helping users to identify and isolate key issues that determine benchmark performance. The problem of establishing realistic and general yet simple models for parallel computation has been studied extensively with varying levels of success [20]. A "good" model for grid computation appears quite distant, since it would have to account for the heterogeneity of resources, varying communication speeds, the dynamic nature of the Grid, and the lack of a widely accepted programming paradigm. In the design of GridBench we adopt a model for the basic Grid infrastructure architecture that is representative of the CrossGrid testbed [16] and is quite similar with most Grid testbeds. In this architecture, which is depicted in Figure 1, a Grid Virtual Organization (VO) is made of a set of geographically distributed sites. Each site contains a Computing Element that manages a set of "Worker Nodes" for performing computations, and an optional "Storage Element," which is an interface to mass storage. Typically Computing Element has direct (Local Area Network)

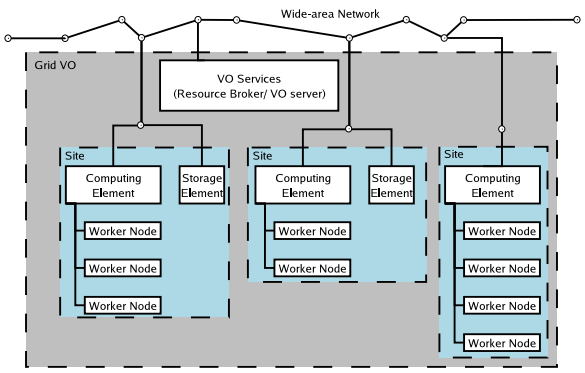

Figure 1. Basic Grid infrastructure architecture.

access to mass storage on the Storage Element that is close to it (e.g. via Network File System). The Grid VO also contains some VO services such as a resource broker, VO membership server etc. The Sites are connected by shared wide-area links over the Internet.

2.2. A Hierarchical Framework

The multi-layered structure of the Grid calls for separate performance investigation of various aspects of the Grid infrastructure. Using benchmarks, we can isolate the performance properties of:

1. Individual *Resources*, such as cluster nodes or Storage Elements;
2. *Sites*, i.e., collections of resources interconnected through a local- or system-area network that belong to one administrative domain (e.g. a cluster of PCs or a symmetric multiprocessor system).
3. *Grid Constellations* comprised of multiple sites which constitute 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 gives the programmer the Grid as a shared resource.

To benchmark these aspects of the Grid infrastructure we propose the use of three categories of benchmarks: *micro-benchmarks*, *micro-kernels* and *application kernels*. Each category provides information about a different point of view of Grid performance. **Micro-benchmarks** are small and simple programs written specially to isolate basic performance characteristics regarding processing, local memory, input/output, communication, synchronization, contention, etc. Each micro-benchmark measures a single and basic performance aspect of the four layers mentioned earlier by "stress-testing" a simple operation invoked in isolation. Basic performance properties are extracted with low-level measurements; for instance of the file-transfer time between two different sites of a VO, or of the floating point operations per second achieved by a particular CPU. **Micro-kernel benchmarks** are synthetic codes designed to stress-test several performance aspects of a Grid component with realistic workloads. Typically, micro-kernels are designed to be simple so that their performance traits can be represented by analytical models with a good approximation. Micro-kernel workloads should be representative of widely used, computationally challenging applications; therefore, micro-kernel benchmarks are derived from core libraries of High Performance Computing/High Throughput Computing applications deployed on Grids or clusters. **Application benchmarks** are derived from real applications deployed on Grids. Typically, they consist of multiple micro-kernels combined in patterns and workflows in typical Grid applications, and resemble the originating applications [7]. Their aim is to measure the performance characteristics of "representative" applications by capturing high-level metrics such as completion time, throughput and speedup.

3. GridBench Design

3.1 Software Architecture

As mentioned earlier, **GridBench** is a tool designed to administer the proposed hierarchical framework for Grid benchmarks and to cope with challenges described in the previous section. The software architecture of GridBench is presented in Figure 2. The main components of this architecture are:

- RSL/JDL Compiler (converts XML descriptions of benchmarks to a job description language);
- Orchestrator (manages benchmark execution and collects results);
- Benchmark Component (the benchmark executable, e.g. Linpack);
- Monitoring Component (collects monitoring information);
- Archive Database (archives benchmark results);
- Information provider (publishes results to information services);
- Benchmark Definition UI (GUI for defining and executing benchmarks), and
- Benchmark Browser (GUI for browsing and analyzing benchmark results).

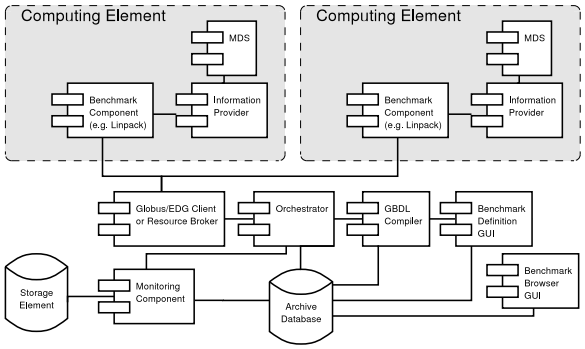

Figure 2. Software Architecture of GridBench.

The RSL/JDL Compiler parses a specification of a benchmark written in the platform-independent XML-based *GridBench Definition Language* (GBDL) and generates the job description language that is necessary for launching the benchmark on a specific middleware infrastructure. At the outset, we set no limitation on which description language is to be produced by the compiler, since we would like GridBench to be usable on top of Grid middleware platforms which support different job description languages. In current implementation, we support the *RSL* language [6] of Globus, and the *JDL* language of the DataGrid project [2], which is based on Condor-G classads [18]. The **Benchmark Components** are the actual benchmarking codes, such as the High-Performance Linpack kernel. The **GridBench MDS Information Provider** publishes results to the local MDS. Data is in the form of an XML document that is placed in a publicly accessible directory and picked up by the GridBench Information provider at regular intervals. The **Data-Transfer Component** deals with transferring data once a component has finished and its output data is required by another component at a different location. The **Orchestrator component** is responsible for coordinating the start-up and execution of the various components, especially in cases where the middleware does not support ordering of sub-jobs. On completion of the benchmark, the *Orchestrator* uses the *Archiver* for storing the resultant XML into a database. The **Monitoring Component** is a client to monitoring services. Monitoring of the resources under measurement is invaluable for understanding the results. The monitoring component takes a description of what to monitor through the GBDL and collects monitoring information. The GBDL specifies the type of monitor (such as R-GMA or OCM-G) and the target resource. The collected data are then stored in a Storage Element with a reference stored back into the GBDL.(See section 3.2).

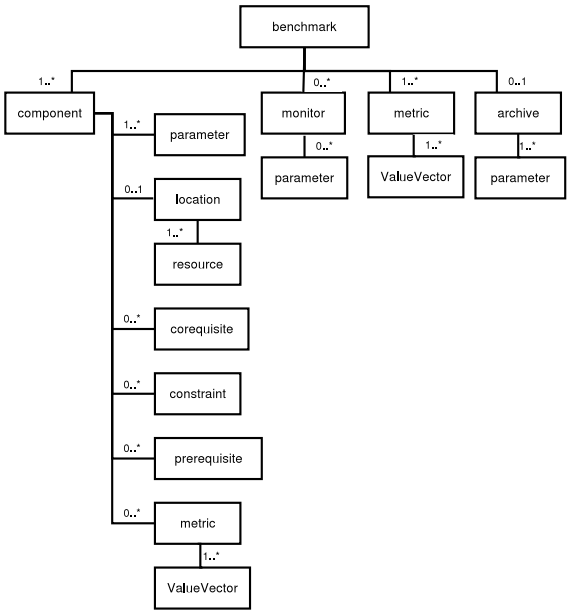

Figure 3. The GBDL specification schema.

3.2. Meta-data

To specify Grid benchmarking experiments in a platform-independent way, we introduce an XML-grammar named the **GridBench Definition Language** (**GBDL**). A GBDL definition specifies the codes to be executed while conducting a benchmarking experiment on the Grid, the target resources, the invocation of monitoring software and references to monitoring data, the definition of measurements to be taken and references to measurement archives. The exact structure of a GBDL document is given in Figure 3. As shown in this diagram, a benchmark definition includes benchmark component elements, metric, monitor and archive elements.

Each **benchmark component** element is described by:

1. A set of *parameters*, which can be of type *value* or *attribute*. *Attribute* parameters are used directly in the generation of RSL/JDL, while *value* parameters are command-line arguments passed to the benchmark executable; For example, to pass the command-line argument "nbodies=10000" to a benchmark executable, we use the following syntax:
   ```
   <parameter name="nbodies" type="value" value="10000"/>
   ```
   On the other hand, to insert the `(count=16)` attribute in the job description language produced by the RSL/JDL Compiler, we use:
   ```
   <parameter name="cpucount" type="attribute" value="16"/>
   ```
2. The *location*, which is made up of at least one *resource*;

3. A set of *co-requisites*, specifying the set of other components that must be running while this component is running;
4. A set of *prerequisites*, that is the set of components that must have finished before this component can start.
5. A set of *constraints* for the runtime of the component, e.g., that the target resource must support MPI.
6. A set of *metrics* that make up the values of the results of the benchmark component.

Metric elements encode the high-level results of a benchmark as a whole, in contrast to the metric elements that are found inside a component element, which are results of just one component. For example, a benchmark composed of several components could have a "completion time" benchmark-level metric, denoting the total wall-clock time taken to complete all components. While the nature of metrics varies depending on the type of a benchmark, each metric can be expressed in one of the following ways:

- *Single-value measurements*, such as average instructions per second, e.g.:

```
<metric-name="epwhetstone_IPS" type="value">
  <vector-unit="MIPS">53</vector>
</metric>
```

- *Multi-value measurements*, such as throughput during a file transfer, e.g.:

```
<metric-name="xfer-rate" type="list">
  <vector-unit="bps">
   2039430,2083930,1909830,2184750,...</vector>
  <vector-unit="second" toffset="1055327287">
   0,10,20,30,...</vector>
</metric>
```

Monitor elements specify what monitoring (type, time period etc.) is to be performed by the monitoring component; instructions for the collection of monitoring data during a benchmark execution. The following is an example of a monitoring data collection specification:

```
<component name="data-transfer" ID="xfer01">
...</component>
<monitor type="RGMA" source="ccwp71.in2p3.fr:3306"
    query="select * from NetworkTCPThroughput
  where NMIdSource='adc0003.cern.ch'
    and NMIdDestination='ccwp7.in2p3.fr'
  <parameter name="begin">comp-begin="xfer01"</parameter>
  <parameter name="end">comp-end="xfer01"</parameter>
</monitor>
```

In this example, TCP throughput monitoring data are gathered for the duration of the "data-transfer" component execution. Note that the *begin* and *end* times are determined during runtime and correspond to the begin and end times of the component with ID='xfer01'.

Finally, **archive elements** specify the URL of the native-XML database (namely the Apache-Xindice database) used to store the final XML documents with the incorporated results

The outcome of benchmarking experiments is encoded as a set of metrics, which we incorporate into the GBDL XML document along with the benchmark definition and the monitoring references. This is done because benchmark results on the Grid make little sense without a representation of the conditions under which they were obtained.

3.3. Archival and Publication of Results

Benchmark results along with the benchmark definition are archived for reference purposes. This must be done in a manner that facilitates comparisons between runs of benchmarks and the statistical analysis of results collected over time. The possibility of selecting the results by providing a list of parameters with which a benchmark was run, is especially useful for analysis. The archival requirements were met by storing the data in a native-XML database (Xindice [1]). Storing results in such a database means that the user can retrieve historical results by specifying the benchmark type, while he can narrow down the search by specifying parameters with which the benchmark was executed.

Publication of the benchmark results in the local Information Services (MDS) [12] of a resource, and optionally to the high-level Virtual Organization is useful for several reasons. First, it makes benchmarking metrics data available via a well-known service. Second, the metrics data available in the MDS (or any Information Service for that matter) can be used by schedulers, administrators or even application users who want to know how "good" a resource is at running their code. Similar work is done by the Network Weather Service [22] with the main difference that NWS publishes monitoring data, not benchmarking results. A diagrammatic version of the MDS/LDAP schema used for publishing the result in the MDS is given in Figure 4.

The GridBench architecture does not rely on MDS for any of its basic functionality. The publication of results to a different Information Service is as simple as modifying the *GridBench MDS Information Provider* to interface to the other Information Service. In fact the archival of the results using XML simplifies the process of creating Information Providers.

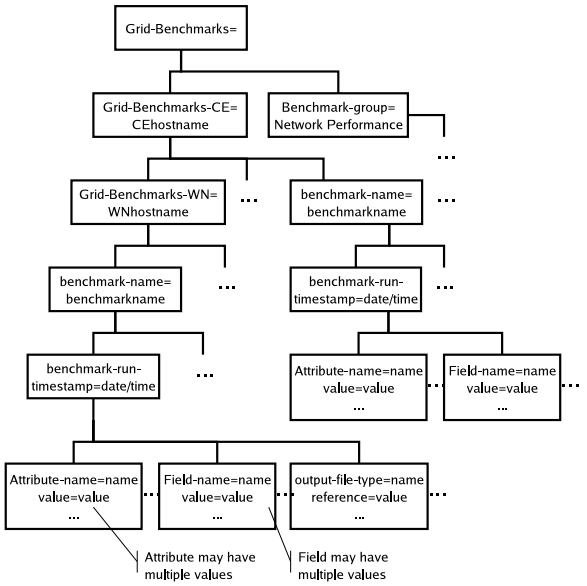

Figure 4. MDS schema for GridBench results.

4. Implementation and Testing

In order to validate the design of GridBench, several benchmarks were selected for implementation as a proof-of-concept.

4.1. Micro-benchmarks and Micro-Kernels

In the case of the Grid, and more specifically the CrossGrid testbed, micro-benchmarks can be applied to the different levels of the Grid architecture. The three levels of the architecture that micro-benchmarks will target are the *CPU*, *Site* and *Grid VO* level.

Micro-benchmarks at the Worker Node level

EPWhetstone is based on the whetstone [5] benchmark, which is a sequential, synthetic floating point benchmark. In this modified version, EPWhetstone, aims to measure the performance of individual CPUs in a Computing Element (cluster) by running on all CPUs in a parallel with each process running independently.The benchmark returns Million Instructions Per Second (MIPS).

BlasBench [17] evaluates the performance of a CPU by using the BLAS library for performing basic Linear Algebra operations. Returns FLOP/s and MB/s.

Micro-benchmarks at the Site (CE/SE) level

Bonnie++ [4] is a benchmark used to evaluate I/O performance. In the CrossGrid architecture, a Computing Element is usually paired with a Storage Element which makes storage space available via NFS. The CE mounts that directory and can access the data stored in the SE. The benchmark returns MB/s during reading and MB/s during writing.

MPPtest [13] is a benchmark that tests MPI communication speeds by various ways and provides a variety of options for a detailed performance analysis. MPPtest is platform and MPI implementation independent and can therefore be used with any MPI implementation. MPPtest aims to make reproducible measurements of MPI performance and results are claimed to be reproducible since the reported measurements are the minimum of several runs (Returns a set of measurements that characterize MPI Performance).

Micro-benchmarks at the Grid VO level

MPPtest: The same principle as for the CE mpptest can be applied to the Grid, where each participating resource (CE) can either host a single process running on the CE or a set of processes, each running on a different Worker Node.

gb_ftb: This File Transfer Benchmark is a straightforward file transfer benchmark transfers of large files. The user needs to specify *Hostname 1*, *hostname 2* and *protocol. Hostname 1* and *hostname 2* are the source and target systems respectively. The *protocol* is the protocol used in the file transfer (such as ftp, gridftp, gsiftp, sftp, ...). The file to be transfered will be pregenerated, generated on the fly or staged at *hostname 1* (the source) depending on the user options. A user-supplied *measurement interval* will determine the frequency of recording the transfer progress.

The total time taken by the transfer will be the *completion time* (excludes time for staging or pregeneration of files). The *transfer rate* is a vector of data points at every measurement interval starting at the beginning of the transfer . The output will include the *average* and *standard deviation* of the transfer rate.

Micro-kernels at the Site (CE) level

High Performance Linpack [10] is one of the most widely known benchmarks in HPC; HPL now ranks the TOP500 [11] super-computers. The HPL benchmark solves a dense system of linear equations by Gaussian elimination. It is MPI-based and in order to perform its basic computations it utilizes either of two libraries, the Basic Linear Algebra Subprograms or the Vector Signal Image Processing Library.

Selected kernels from the NAS Parallel benchmarks [3]**:** These are a set of kernels and simple applications from Computational Fluid Dynamics that have been thoroughly analyzed and formulated into a set of benchmarks.

Micro-kernels at the Grid VO level

Computationally Intensive Grid Benchmarks: By some minor modifications the kernels from the NAS

Benchmark	Metric	Retention
EPWhetstone	MIPS	in GBDL
BlasBench	flop/s, MB/s throughput	in GBDL
Bonnie++	I/O bandwidth in MB/s	in GBDL
MPPtest	time (s) for MPI operations	reference
gb_ftb	completion time, transfer rate	reference
Job start-up	time (s)	in GBDL
HPL	flop/s	in GBDL
NGB	flop/s	in GBDL
CIGB	completion time	in GBDL

Table 1. GridBench metrics

Parallel Benchmarks can be combined into large-scale applications that run distributed, as proposed in [7]. Varying the combination of kernels and the architecture it is possible to specify benchmarks that resemble different classes of applications, such as visualizations and cascades of simulations.

Table 1 contains a list of the metrics that will be produced and stored by the benchmarks previously described. In the table, *retention* refers to whether the metric values are kept inside the GBDL document or if they are stored separately in a repository, in which case only a reference is stored in the GBDL.

4.2 Compilation to RSL/JDL

When it comes to the generation of RSL/JDL from the benchmark description, each component element in the XML is transformed into a set of sub-job specifications. The target resource is obtained by the *resource* element and the executable is determined by the parameter with `name='executable'`. Since each of the components has a different scheme for command-line parameters, each benchmark executable must have an associated ParameterHandler_*benchmark-name* class, which puts the parameters for the benchmark in the correct command-line format. In the following paragraph, we provide a single-component example with the corresponding GBDL XML definition and the generated RSL:

GBDL benchmark definition:

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

Generated RSL:

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

4.3. Experiments and Testing

In our prototype implementation we obtained some measurements of the CrossGrid testbed [16] as proof of concept. Figure 5 shows graphs that are derived from some of the initial results of running the *gb_site_hpl*, the *gb_epwhetstone* and the *gb_ftp* file transfer benchmarks.

The *gb_site_hpl* was executed using two CPU's, on several of the Crossgrid testbed sites (the benchmark ran independently at each site). The *gb_epwhetstone* benchmark ran on two CPUs at each of the sites shown. The graph of the *gb_ftb* file transfer benchmark shows the progress of transferring a small file from one site to the others using GSIFTP (transfers were not concurrent). A steeper curve implies a faster transfer and the completion time for each file transfer is indicated by its maximum x value.

5. Conclusions and Future Work

In this paper we examined issues related to the employment of benchmarks for the Grid. We presented the design of GridBench, a tool that we are implementing to benchmark Grid systems. We described how benchmarks can be specified using the GridBench Definition Language and compiled into the RSL and JDL job specification languages supported by the Globus and EDG middleware. We also described the mechanisms by which results and monitoring data are stored in XML databases and published in Grid information systems. As proof-of-concept we implemented a set of benchmarks and executed them on the CrossGrid testbed, starting from GBDL definitions.

The GridBench facilities for defining and executing benchmarks, as well as archiving and publishing benchmark results have numerous uses for researchers, administrators and users of the Grid, who can benefit from easy access to and statistical analysis of multiple archived executions of benchmarks. Grid users can benefit by having access to measurements from several resources, from which they can select for running their application.

The specification of more complex, multiple component benchmarks is necessary for truly understanding Grid architectures. The Computationally Intensive Grid Benchmarks[7] and benchmarks based on the

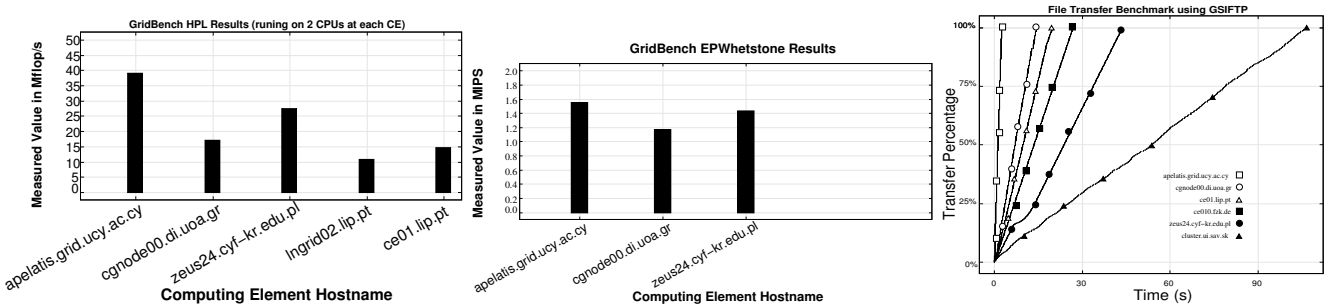

Figure 5. Measurements obtained by the gb_site_hpl, gb_epwhetstone and gb_ftb benchmark at several sites of the CrossGrid testbed.

CrossGrid applications will be implemented during the next phase of development. Future work will also focus on enhancement of the existing software with the addition of a graphical user interface, and integration with middleware beyond CrossGrid and Globus.

References

[1] Apache xindice,http://xml.apache.org/xindice.

[2] Datagrid project. http://www.eu-datagrid.org, 2000-2004. Information Societies and Technologies Program, Framework Program Five, European Union.

[3] D. Bailey, T. H. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The nas parallel benchmarks 2.0. *The International Journal of Supercomputer Applications*, 1995.

[4] R. Coker. Bonnie++. Technical report. http://www.coker.com.au/bonnie++/readme.html.

[5] H. J. Curnow and B. A. Wichmann. A synthetic benchmark. *The Computer Journal*, 19(1):43–49, 1976.

[6] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459:62–??, 1998.

[7] R. V. der Wijngaart. Computationally intensive grid benchmarks. *Global Grid Forum (2003)*, 2003.

[8] M. Dikaiakos, M. Kyriakou, and G. Samaras. Performance Evaluation of Mobile-Agent Middleware: A Hierarchical Approach. In G. P. Picco, editor, *Proceedings of the 5th International Conference on Mobile Agents (MA 2001)*, volume 2240 of *Lecture Notes in Computer Science*, pages 244–259. Springer, 2002.

[9] J. Dongarra and W. Gentzsch, editors. *Computer Benchmarks*. North Holland, 1993.

[10] J. Dongarra, P. Luszczek, and A. Petitet. The linpack benchmark: Past, present, and future. December 2001.

[11] J. J. Dongarra, H. W. Meuer, and E. Strohmaier. TOP500 supercomputer sites, 11th edition. Technical Report UT-CS-98-391, 1998.

[12] 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.

[13] W. Gropp and E. L. Lusk. Reproducible measurements of MPI performance characteristics. In *PVM/MPI*, pages 11–18, 1999.

[14] R. Hockney and M. Berry. Public international benchmarks for parallel computers report, 1994.

[15] S. Manley, M. Seltzer, and M. Courage. A Self-Scaling and Self-Configuring Benchmark for Web Servers. In *Proceedings of the 1998 Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 270–272. ACM, 1998.

[16] J. Marco et al. First Prototype of the CrossGrid Testbed. In *First European Across Grids Conference*, February 2003.

[17] P. Mucci. The blasbench report, 1998.

[18] R. Raman, M. Livny, and M. H. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC*, pages 140–, 1998.

[19] G. Samaras, M. Dikaiakos, C. Spyrou, and A. Liverdos. Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 50–64. IEEE-Computer Society, October 1999.

[20] L. Snyder. Experimental validation of models of parallel computation. *Computer Science Today*, pages 78–100, 1995.

[21] Transaction Processing Performance Council (TPC). *TPC Benchmark W (Web Commerce) - Draft Specification*, December 1999.

[22] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[23] S. C. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–37. ACM, June 1995.