Characterization of Computational Grid Resources Using Low-level Benchmarks

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

Abstract— **An important factor that needs to be taken into account by end-users and systems (schedulers, resource brokers, policy brokers) when mapping applications to the Grid is the performance capacity of hardware resources attached to the Grid and made available through its Virtual Organizations. In this article, we examine the problem of characterizing the performance capacity of Grid resources using benchmarking. We examine the conditions under which such characterization experiments can be implemented in a Grid setting and present the challenges that arise in the Grid context. We specify a small number of performance metrics and propose a suite of micro-benchmarks to estimate these metrics for sites that belong to large Virtual Organizations. We describe benchmarking experiments conducted with, and published through GridBench, a tool that we built to manage benchmarking experiments over the Grid and to publish and analyze performance metrics. Finally we show how results derived from GridBench can help end-users assess the performance capacity of resources belonging to a large Virtual Organization.**

I. Introduction

Information about the performance capacity of computational Grid resources is essential for the intelligent allocation of resources to Grid applications. This need arises from the diversity in performance capacity, which is common-place in heterogeneous Grids. Performance capacity estimates can help users and schedulers make more informed resource allocation decisions by combining this information with information about application performance (empirical or other).

Our goal is to complement the information available via Grid information and monitoring services by annotating resources with low-level performance metrics. Our conjecture is that the use of lightweight benchmarking for performance evaluation and functionality tests can be useful for schedulers, resource brokers, service providers and end-users.

In order to achieve our goal, a number of challenges need to be addressed:

- First, we need to determine a *small* set of simple, easy to understand and clearly defined performance metrics that effectively characterize the performance capacity of heterogeneous resources.
- Then, we have to select and implement a set of *minimally intrusive* benchmarks that will help us derive measurements of the chosen metrics upon resources deployed on real Grids. The deployment of the selected benchmarks on a Grid should not require excessive administrative effort or the reconfiguration of existing middleware, and should not raise issues related to software licensing. Benchmarking experiments should take minimal time and should not result to a major disruption of Grid operations.
- The management of benchmarking experiments should not incur excessive cost upon end-users or system administrators. The benchmarks should be executed on the different Grid resources of a Virtual Organization, both in a *periodic* and in an "*on-demand*" manner. Periodic execution can be performed if benchmarking is to be used as a mechanism for active, non-intrusive, end-to-end performance monitoring, which can provide resource brokers and Grid schedulers with an up-to-date feed of performance metadata. In this paper we focus on on-demand execution, which can be useful to end-users and administrators that wish to investigate the performance attributes of resources participating to a Virtual Organization.

In the remainder of this article, Section II describes our approach to characterization of computational Grid resources through benchmarking and provides a short description of related work, as well as the GridBench framework which we used to obtain our results. Section III describes the metrics and benchmarks which we propose for use in the characterization of resources. Section IV presents the experiments we conducted, while Section V describes some practical issues we faced. In the last section we present our conclusions and the general direction of our future work.

II. Resource characterization

In existing Grid infrastructures, the performance capacity of Grid resources can be obtained through Grid Information Services (such as the Monitoring and Discovery Service [12]). Scheduling decisions based on the performance capacity (or "speed") of the candidate resources have to rely, at best, on the size of main memory, number of CPU's, and their nominal speed (e.g. in MHz). This is due to the fact that this is what users or schedulers can expect from information services such as MDS. Despite that, Grid information service designers recognized the importance of performance capacity and made allowances in their information schemata for including performance information. An example is the GLUE Schema [7], developed for interoperability between US and European projects used by middleware systems supporting large infrastructures, such as the EGEE [1], and the OSG [4]. The Glue schema includes placeholders for the `GlueHostBenchmarkSF00` and `GlueHostBenchmarkSI00` attributes, in order to accommodate the SPECFloat2000 and SPECInt2000 benchmarks. Such information is usually obtained and specified by system administrators, and therefore it is potentially inaccurate because it is prone to human error, be it intentional or unintentional. The fact that this is static information also creates problems because experience shows that a resource's performance capacity does change over time, either by the addition or removal of CPU's or even by simple alterations in the configuration of the hardware or software.

A. Related work

The Grid Assessment Probes [8] test and measure performance of basic grid functions such as job submission, file transfers, and performance of Grid Information Services. DiPerF [11] is a distributed performance-testing framework, that aims to automate *service* performance evaluation. It coordinates a pool of machines that test a target service, collects and aggregates performance metrics, and generates performance statistics for service "fairness" and service throughput. The ALU Intensive Grid Benchmarks (AIGB) [10] aim to measure the performance of Grids via pre-defined work-flows using the established NAS Parallel Benchmarks as computational kernels.

Infrastructure Monitoring tools such as NWS [26] can provide useful real-time information of several system aspects, and while most monitoring tools *do* measure network latency and bandwidth between distributed Grid resources, they do not address computational performance of the monitored resources. NWS has "CPU sensors" which can provide measurements mainly for the purpose of forecasting CPU availability, but it does not provide *benchmarking* metrics to measure computational performance and capacity.

The Inca test harness and reporting framework [20] is a system that aims to automate the testing of resources, automate resource data collection, perform resource verification and monitor service agreements. The Site Functional Tests (STF) [5] are a set of tests that are periodically executed in order to evaluate the functionality of different middleware at Grid sites participating in EGEE [1]. The test results could potentially provide an indication of the performance of some basic and some not-so-basic middleware tasks such as file replication.

In contrast to the work presented in this paper, DiPerF and GRASP focus on *service* and file-transfer performance and do not address computational resource performance. Inca and the SFT are testing frameworks that were not specifically designed for computational performance evaluation and so do not address the computational performance of a Grid site. These tools and the benchmarks we propose could work in synergy. As suggested in [20], the benchmarks proposed by GridBench [22] and GRASP [8] could be views as "tests" and invoked using the Inca framework. In contrast to the AIGB, our effort focuses on lightweight benchmarking for the performance characterization of Grid resources, while the Computationally Intensive Grid Benchmarks are pencil-and-paper definitions of workflow-type applications. Finally, the work presented in this paper aims to complement, and by no means replace, the information provided by monitoring tools (such as NWS) by providing up-to-date low-level performance metrics.

B. Characterization through benchmarking

Micro-benchmarks provide a commonly accepted basis for comparing different computer systems in terms of their performance. They are also used to investigate performance properties of computer systems under carefully tuned, benchmark-induced workloads that stress particular aspects of system performance. For example, CPU benchmarks, such as *Whetstone* [9], focus strictly on CPU arithmetic operations in a tight loop with minimal memory requirements, thus disregarding the effect of other factors on overall performance. Micro-benchmarks have many uses to different kinds of users: for example, administrators could use benchmark measurements to detect and *pin-point* faults or problems in general, and end-users could use measurements to select appropriate resources for running their application.

A *small* set of suitable metrics is necessary for quantitative performance characterization. These metrics need to reflect the basic factors affecting performance of heterogeneous computational Grid resources. The size of this set, i.e. the number of metrics, should be of reasonable size. Too small a set would run the danger of missing essential performance factors while a large set of metrics would complicate decision-making by providing too much information, but more importantly it would impose a higher cost from the additional benchmark executions.

Once the set of metrics is established, a carefully selected set of benchmarks must be employed. It is also essential that the selected benchmarks run for the minimum amount of time and produce reliable results. While long-running codes (i.e, real or synthetic kernel-benchmarks) may potentially provide more accuracy or higher-level metrics, they are not desirable since they would tend to be more application-specific and they would incur a large cost for running benchmarks, especially if they are run at regular intervals.

The heterogeneity of Grid resources (introduced by both different hardware architectures and operating systems) imposes the additional requirement of *portable* benchmarks. A key factor in the selection of benchmarks should be their portability and code availability. Heterogeneity, its impact on measured performance and the portability issues that arise have been the subject of study in [14].

The collected results must be made easily available for use by the decision-making process. The benchmarking process produces a high volume of measurement data, considering that some measurements are not simple scalars and that it would be of interest to maintain a historical record of performance measurements, e.g. for the statistical assessment of resource availability and dependability.

C. GridBench

GridBench [22], [24] is a set of tools that aim to facilitate the characterization of Grid nodes or collections of Grid resources. In order to perform benchmarking measurements in an organized and flexible way, the GridBench framework is provided as a means for running benchmarks on Grid environments as well as collecting, archiving, and publishing the results. We used the GridBench framework to perform the experiments described in Section IV. GridBench provides an interactive graphical user interface for configuring and invoking benchmarks. We used this interface to conduct and manage our experiments, and used its facilities to create the charts that were used in the analysis. We made use of Gridbench's functionality for executing benchmarking jobs that are generated from high-level benchmark definitions and for archiving results to a database back-end. This functionality of GridBench is exposed via web-services. For interoperability with the underlying test-bed we used the *"GlobusPlugin"* middleware plug-in provided by GridBench.

III. Metrics and Micro-benchmarks

A. Metrics

A critical step in our methodology is the selection of a concise set of metrics for the low-level characterization of the Grid's computational resources. It is a reasonable assumption to make that the resource's performance depends mainly on the performance of its CPU's, the performance of its memory and caches, and the performance of its interconnects. Of course there is a wealth of other factors affecting machine performance ranging from I/O performance to Operating System robustness to fitness for running a specific application. We chose to limit the set of metrics to a concise size, but kept the design open for easy inclusion of more metrics as deemed necessary. In terms of specific metrics we have chosen (i) *Operations Per Second* for CPU performance (integer/floating-point), (ii) *Available Memory* and *Bytes per second* for writing and reading to and from main memory/cache (iii) *Latency* and *Bandwidth* for evaluating the machine's interconnects, and (iv) I/O *bandwidth*. These metrics are easily understood and well-established for evaluating their respective performance factor; they are given in Table I.

TABLE I
Metrics and Benchmarks.

Factor	Metric	Delivered By
CPU	Operations per second (mixture of floating point and integer arithmetic)	EPWhetstone
CPU	Floating-Point operations per second	EPFlops
CPU	Integer operations per second	EPDhrystone
memory	sustainable memory bandwidth in MB/s (copy,add,multiply,triad)	EPStream
memory	Available physical memory in MB	EPMemsize
cache	memory bandwidth using different memory sizes in MB/s	CacheBench
Inter connect	latency, bandwidth and bisection bandwidth	MPPTest
I/O	Effective I/O bandwidth	b_eff_io

B. Micro-benchmarks

In the choice of benchmarks, we have had to deal with a trade-off which involves (i) minimizing the overlap of what the benchmarks actually measure, and (ii) providing as complete a characterization as possible. In order to deliver the required metrics, eight benchmarks are employed:
(i) *EPWhetstone*, (ii) *EPFlops*, (iii) *EPDhrystone*, (iv) *EPStream*, (v) *CacheBench*, (vi) *EPMemsize*, (vii) *MPPTest* and (viii) *b_eff_io*. Primarily, these benchmarks were selected for their acceptance in the community and because they are open-source.

During the execution of each of the benchmarks listed above, it is imperative that the only process imposing substantial load on the CPU is the benchmark process, especially since the results are calculated using wall-clock time. Another thing to note is that the "EP" prefix of some benchmark names (namely EPWhetstone, EPFlops, EPDhrystone and EPStream) denotes the "embarrassingly parallel" nature of its execution, which means that each process runs on a CPU independently without any communication during the measurement. The accumulated result from all the processes is then reported as the performance of the whole resource. In many cases it is useful to have results from benchmarks executed as both *i)* one process per CPU and *ii)* one process per SMP node, in order to expose characteristics such as shared main memory and shared network interfaces (see the description of *EPStream* for an example).

EPWhetstone is a simple adaptation of the traditional Whetstone CPU benchmark [9] so that it runs simultaneously on a set of CPU's using MPI. It is implemented in C, and uses MPI for collecting the final measurements from each process (communication time is excluded from measurements). Each process performs a mixture of operations, such as integer arithmetic, floating point arithmetic, function calls, trigonometric and other functions. The benchmark the average rate at which these operations were performed, using wall-clock time. The typical execution time is less than 10 seconds.

EPFlops is a floating-point CPU benchmark adapted from the "flops" benchmark [6]. It is modified so that it runs simultaneously on a set of CPU's using MPI. It measures the performance of a CPU's floating-point operations in different "mixes" of floating-point operations. The benchmark employs a set of 8 modules, where each module is made up of a different mix of operations. Different combinations of the 8 modules yield a set of four metrics ("ratings") with different ratios of each of the four floating-point operations (these are described in detail in [6]). The benchmark tries to maximize register usage in order to be as independent as possible from the performance of the memory sub-system. It is implemented in C and typical execution times are under 5 minutes.

EPDhrystone is an integer operations benchmark, adapted from the C version of the "dhrystone" benchmark [25]. It is modified so that it runs simultaneously on a set of CPU's using MPI. Dhrystone is based on a workload from an extensive set of applications, but does not target numerical computations.While EPDhrystone may not correlate with many codes from the natural sciences, but they do for some others (such as Discrete Simulation and microprocessor simulation).As before, the benchmark has been adapted to run concurrently on a set of CPU's using MPI. The benchmark returns the accumulated result from all the processes in "dhrystones" per second. Typical execution times are under 10 seconds.

EPMemsize is a platform independent benchmark that aims to measure memory capacity. It is written in C and it runs simultaneously on a set of CPU's using MPI. It first determines the maximum amount of memory that can be allocated. It then proceeds to determine the maximum amount of memory that can be allocated *in physical memory*. The size of physical memory available is important to memory-intensive applications that profit from allocating as much memory as possible while avoiding the use of slow swap memory. Detecting the physical memory in the machine in a platform-independent way may not depend on any system-specific system call to get the memory size. More importantly, the value that is returned by a "get_physical_memory()" system call is usually not the real amount of physical memory that can be *allocated* by an application; the system kernel, services as well as other processes also take up memory, file-system caches etc. The benchmark operates by accessing memory until a substantial delay occurs (determined by a configurable delay threshold). The process is performed repeatedly and the maximum amount of memory allocated without incurring swapping is returned.

EPStream is a simple adaptation of the C implementation of the well-known STREAM memory benchmark [15] so that it runs simultaneously on a set of CPU's using MPI. The STREAM benchmark measures the sustainable local memory bandwidth (MB/s). It is a simple synthetic benchmark program and in addition to providing memory bandwidth it also gives an idea of the corresponding computation rate for simple vector kernels. The STREAM benchmark measures bandwidth while performing four operations: *copy*, *scale*, *sum* and *triad* – a commonly quoted metric for memory performance. In the case of SMP machines, such as clusters of dual-CPU or quad-CPU machines, this benchmark can provide useful information when run in either of two modes: *i)* One process per SMP node (e.g. 1 process on a dual node) and *ii)* One process per CPU (e.g. 4 processes on a quad node). This information can be crucial since the memory bandwidth available may be shared between more than one CPU's[1]. The typical execution time of EPStream is around 10 seconds.

CacheBench is a benchmark aiming at evaluating the performance of the local memory hierarchy of a machine [17]. The benchmark is implemented in C and performs a set of operations – *read*, *write*, *read/modify/write*, *memset()* and *memcopy()* – varying the underlying array size thus exposing the performance of the (potentially multi-level) cache. For example, a knee can be observed at the different cache sizes when the results are plotted on a graph (see Figure 5(a)). An instance of CacheBench is invoked on each CPU of the resource under study and results are reported independently for each CPU. The operations at each size run for a configurable amount of time (default is 2 seconds) and the average bandwidth (MB/s) is reported. Typical execution times are under 5 minutes.

MPPTest is a benchmark that tests MPI communication speeds by various ways and provides a variety of options for a detailed performance analysis [13]. MPPtest is platform and MPI-implementation independent and can therefore be used with any MPI implementation. For the purpose of resource characterization it is desirable to have a focused set of measurements and to this end, only three types of measurement are performed: (i) Latency, (ii) point-to-point bandwidth and (iii) bisection bandwidth. "Bisection bandwidth" refers to measurement of bandwidth with all processes participating in contrast to the *point-to-point* measurement where only two processes communicate at any time. The typical execution time is on the order of minutes (depending on the measurement detail) and results are calculated using wall-clock time. Typical execution times are under 3 minutes.

The b_eff_io benchmark is included in order to evaluate the shared I/O performance of (shared) storage at a resource (site). This benchmark is used "to achieve a characteristic average number for the I/O bandwidth achievable with parallel MPI-I/O applications" [18]. *B_eff_io* produces a metric given in Megabytes per second, which represents the average obtained by performing several storage access patterns. Access patterns include: (i)Multiple processes read/write data scattered in a file; (ii) Multiple processes read/write adjacent data; (iii) Multiple processes read/write data in separate files; and (iv) each of the multiple processes accesses data in a different segment of a segmented file (a detailed description of the access patterns can be found in [18]). Typical execution times are under 10 minutes.

IV. Experimentation

The experiments described in this section were conducted on a medium-size testbed [2], [3] with 17 sites and 140 CPU's. The experiments were conducted wholly within the GridBench framework and the charts were generated using the GridBench GUI.

We first looked at CPU performance; we selected a set from the available resources at the time and invoked the CPU benchmarks. We did this with a simple drag-and-drop from the benchmark list onto the the selected resources in the resource list. Figure IV shows the result of invoking the three CPU micro-benchmarks *EPDhrystone*, *EPWhetstone* and *EPFlops* on the set of currently available resources. Quite apparently, the shape of the graphs is very similar, since all the sites participating in the testbed under study have pretty much the same type of processors (Intel PIII/P4). It is also apparent that results from *EPWhetstone* and *EPFlops* are very close. Again this is expected since Whetstone relied more on floating-point operations than on integer operations.

Selecting one of the CPU performance metrics for analysis, Figure 2(a) shows results for the *EPWhetstone* benchmark, providing a "view" of the available resources at a point in time, from a CPU performance perspective. When EPWhetstone is executed on a Grid resource it returns a set of values, each

[1]An example of this is that Dual Intel "Xeon" nodes typically have a single memory controller per SMP machine, while AMD "Opteron" SMP machines typically have a separate controller for each CPU and can thus achieve a higher memory bandwidth.

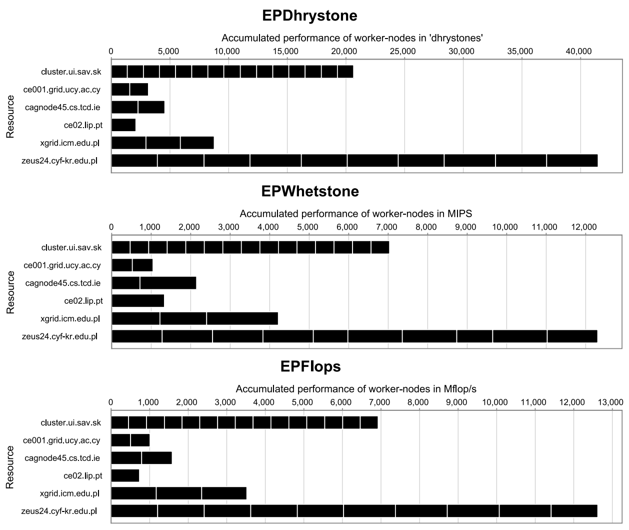
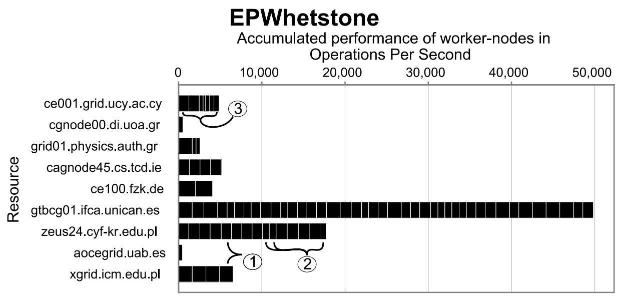
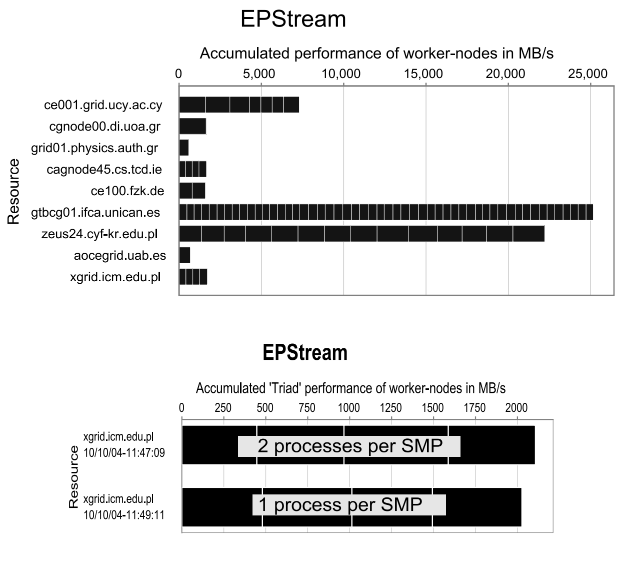


Fig. 1. CPU performance



(a) EPWhetstone performance



(b) STREAM performance and SMP impact

Fig. 2. EPWhetstone and STREAM performance.

value measuring the CPU performance of a single CPU. When the resource under study is a cluster, a *stacked bar-chart* is generated, where each segment represents the contribution of a single cluster node. If a number of CPU's come from the same SMP worker node, their performance is aggregated and displayed as a single segment.

The first deduction to be made is that these resources *are operational*; all the processes were initiated and completed successfully and the results returned. A successful benchmark execution just before staging and running the actual application, can verify that the resource is operational from a hardware and middleware point of view and could indicate degraded performance of the resource.

It is also evident in Figure 2(a) that the different resources vary greatly in terms of processing power. They vary both in terms of the number of CPU's and in terms of individual CPU performance. For example, resource **zeus24.cyf-kr.edu.pl** has a larger number of CPU's than **xgrid.icm.edu.pl**, but the latter resource has faster CPU's. (See (1) in Figure 2(a)). If we focus on resource **zeus24.cyf-kr.edu.pl** we can observe that 3 CPU's (indicated as (2) in Figure 2(a)) appear to be performing slightly worse than the rest. This could be attributed to other processes running on the specific cluster nodes. This could also be attributed to other problems, ranging from hardware faults to software misconfigurations. It was in fact determined, by observing the monitoring information collected during the benchmark execution, that on the three machines in question there was non-negligible CPU load right before and immediately after the execution of the benchmark. For some applications this could be of little importance, but for some tightly-coupled codes a single slow node could seriously impact performance. Internal resource uniformity can also be evaluated. Unevenly sized segments could result from a set of cluster nodes that are of dissimilar performance (e.g. (3) on Figure 2(a)) where the resource is known to contain a mixture of single- and dual-CPU nodes.

Having looked at the CPU performance we then proceeded to memory performance. Again we parform a simple drag-and-drop of the EPStream benchmark onto the resources. Figure 2(b) shows results for the *EPStream* benchmark, characterizing the Grid resources from a memory bandwidth perspective. By comparing Figures 2(a) and 2(b), we observe that the relative performances of the resources are in fact different when comparing based on memory performance rather than on CPU performance. With memory intensive codes in mind, it would make sense for the user to make a decision based on memory bandwidth results rather than on CPU results. It is also notable that the large difference in CPU performance between resource **zeus24.cyf-kr.edu.pl** and resource **gtbcg01.ifca.unican.es** in Figure 2(a) is much smaller when comparing memory performance in Figure 2(b). A user intending to run a memory intensive code could choose to do so on resource **zeus24.cyf-kr.edu.pl** since it has good aggregate memory bandwidth using a smaller number of CPU's (potentially enjoying a speedup because of lower communication overhead).

Figure 2(b)-*bottom* illustrates how memory bandwidth is shared between processes running on the same dual-CPU machine. The resource **xgrid.icm.edu.pl** provides 4 dual Intel PIII nodes. In this case, memory bandwidth does not scale with the number of CPU's, in fact the aggregate memory bandwidth remains almost the same. This is another factor that could be taken into account when running memory-intensive codes.

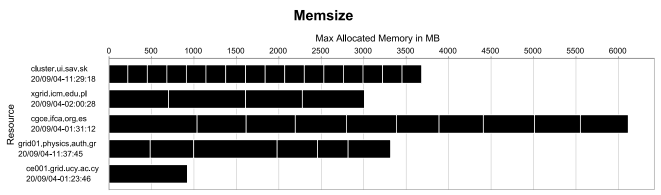


Fig. 3. EPMemsize benchmark showing the approximate maximum amount of memory that could be allocated in physical memory.

In addition to memory bandwidth, the size of main memory is also important. At this point, in order to illustrate the functionality, instead of invoking new benchmarks we pulled archived measurements from the database. Figure 3 shows the maximum amount of *physical* memory that could be allocated on the worker nodes in a set of resources.

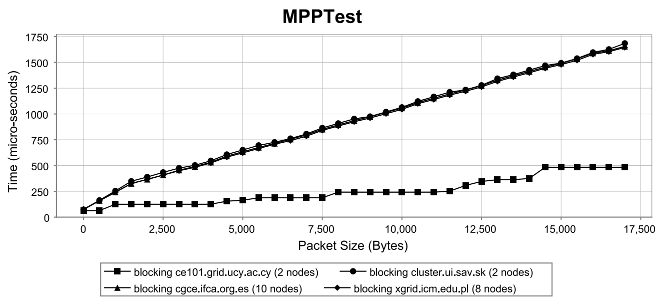
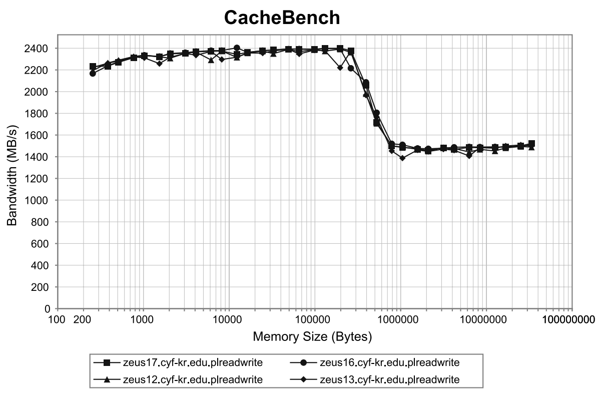


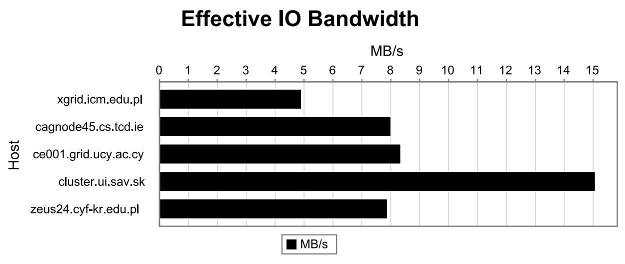
Fig. 4. MPI point-to-point messaging 3 100Mbit/s and 1 1000Mbit/s Ethernet resources.

If the application of interest is a parallel code then it makes sense to look at the performance of the interconnect; we invoke the MPPtest. Figure 4 shows the point-to-point communication performance on four resources. Three of the measurements coincide since the three sites employ the same network infrastructure, i.e. switched 100Mbit/s Ethernet network. The fourth site has a 1Gbit/s network and performs significantly better (at least in terms of bandwidth). Nearly identical results are obtained when measuring bisection bandwidth since none of the resources had enough nodes to saturate the local interconnect. Latency and bandwidth can be read on the graph by considering the zero-size and max-size (16,384) packet-size measurements respectively. The bandwidth is calculated at approximately 8.7MB/s for the 100Mbit/s sites and at approximately 33MB/s for the 1Gbit/s site (33MB/s is possibly a result of a PCI bus bottleneck).

Deciding that we need more information on memory performance, we look closer at cache performance. Figure 5(a) shows the execution of the CacheBench benchmark on 4 CPU's on a resource. The specific metric is the "read/modify/write" metric giving the memory bandwidth in MB/s. The effect of the cache is apparent at the drop-off around 512 KB A user could use this information to tune application parameters for optimal use of the cache for a specific resource.



(a) CacheBench benchmark showing the effect of the memory cache on memory bandwidth



(b) Effective Disk I/O

Fig. 5. Interconnect and memory cache performance.

In terms of disk I/O performance, Figure 5(b) shows the results of the effective I/O bandwidth measurements on several resources. For each execution, 2 CPU's were used on 2 separate worker nodes. It can be seen that the resource *cluster.ui.sav.sk* performs considerably better than the others. We discovered that while the worker nodes were in fact connected over a 100Mbit network, the shared storage had a 1Gbit interface connected to the switch's 1Gbit uplink.

All charts shown in this article were generated using the GridBench GUI using data archived (from previous runs) in its XML database and from newly obtained data. This data is available for retrieval not only by end users, but also by automated decision-makers such as schedulers. A scheduler could use micro-benchmark results to *"rank"* the resources based on performance (CPU, memory or MPI). This could be taken a step further where the ranking depends on a user-specified function of several micro-benchmarking metrics. So a user, while "shopping around" for the right resources to purchase, could assign more weight to memory bandwidth than to floating-point CPU performance and zero-weight to interconnect latency, and rank based on these. A scheduler could periodically evaluate a resource's "health" by invoking one of the micro-benchmarks. Since execution times for most benchmarks are typically less than 10 seconds, this would impose little additional delay and would potentially save a scheduler from time-consuming failed submissions.

Application Performance

To illustrate how this information can be used, we selected three resources from our testbed that support MPI and have at least 4 CPU's and ran some experiments on application performance. We used two applications from two distinct areas of science: Air Pollution Simulation, and Blood-flow Simulation. For air pollution simulation, we used the VERTLQ kernel which 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 [16]. This code benefits mostly from a fast CPU. For Blood-flow Simulation, the "bstream" kernel is extracted from a medical application, developed at the Univ. of Amsterdam, 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 [19]. This code benefits mostly from a high memory bandwidth.

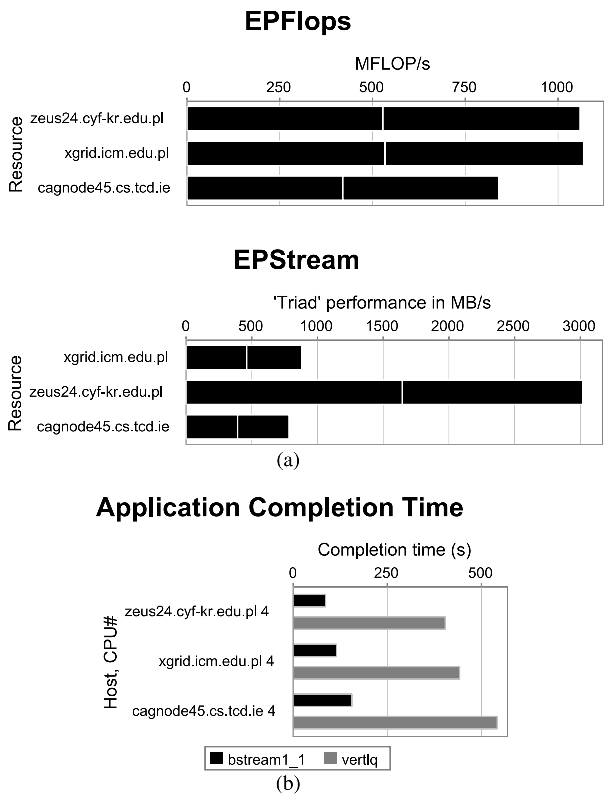


Fig. 6. Application completion times.

We first invoked a CPU benchmark (EPFlops) and a memory benchmark (EPStream) onto the resources, the results are shown in Figure 6(a). In terms of CPU performance **zeus** and **xgrid** appear to be almost identical, while **cagnode** is slightly behind. In terms of memory performance **zeus** clearly dominates the other two. If we were to pick a resource for the memory-intensive *bstream*, we would clearly pick **zeus**. If we were to pick a resource for the CPU-intensive *vertlq* we would probably pick **zeus** again, considering the huge memory performance difference, although **xgrid** is slightly faster.

To run the applications, we again used the GridBench tool by integrating the two applications just as with regular benchmarks. The integration of a new application into the tool (depending on the complexity of the application) is quite straight forward and it basically involves specifying a new GBDL template [23]. Once an application is integrated, the tool can be used to submit the application as a regular benchmark and automatically obtain measurements such as completion-time and staging-time. This process as well as a more extended analysis can be found in [21].

The results in Figure 6(b) verify the selections we made earlier based on the microbenchmarks. Resource **zeus** outperforms in both cases, which was somewhat expected. What is more interesting that for *bstream* there is a 45% drop in performance by going from **zeus** to **cagnode**, while for *vertlq* there is only a 25% drop. Additionally, if the user was making a selection based purely on CPU performance then they would've picked **xgrid**. This would've lead to a 9% drop in performance of *vertlq* and a 25% drop in performance of *bstream*. Without the extra information from the performance exploration, the user could have easily suffered a considerable drop in performance.

V. Practical Issues

During our experimentation and our effort to characterize a set of resources, we have come across several issues regarding the functionality of the infrastructure. The fact that these problems surfaced, and could therefore be addressed, is another incentive for running benchmarks. For example, early on in the experimentation there were issues regarding the execution of MPI codes. The errors were sometimes reproducible and sometimes not. By running micro-benchmarks it was determined that some cluster nodes at several Grid sites were inaccessible (due to outdated OpenSSH keys), yet they were reported to be available. The administrators were contacted and the problem resolved. Another configuration issue we met often was the incorrect spawning of processes. Specifically, more processes than available CPU's were spawned on a worker node, which was easily detected by looking at the results of CPU micro-benchmarks. A problem that was often met and is categorized as a "general system issue" is the presence of run-away processes on several resources. This was detected by the observation of degraded performance by some micro-benchmarks.

In many cases the underlying reason for failed benchmark executions or degraded performance of a benchmark was not determined, it is important though that many problems were detected and action could be taken to correct then. Some of these issues could have been detected by proper monitoring, but many of them would not surface without using an end-to-end test (involving most of the hierarchy of employed middleware) such as a benchmark.

Other issues include some "hidden costs" of running benchmarks involving multiple worker-nodes. In our experimentation, the benchmarks were submitted as regular jobs, and while extra care was taken to have benchmarks that run for very short times, many queues at the different resources were seriously affected. The reason for this was that the benchmark job (or, for that matter, any other parallel job) will hold back other jobs until enough resources are available for it to execute. This is a well known problem and it emphasizes the need for carefully scheduled benchmark executions, possibly using special permissions and, of course, carefully tuned benchmark parameters that respect to resource attributes.

VI. Conclusions and Future Work

We have presented a concise set of benchmarks for the characterization of computational Grid resources, in terms of the performance of CPU, main memory, interconnects and I/O. We have also presented an adopted set of benchmarks to deliver those metrics. This small set of lightweight benchmarks can be run on the Grid resources with little overhead and with minimal effort by the user. The resulting measurements are archived and made available via a web-service.

We have also presented a set of results obtained from our Grid test-bed and described how results such as these can be useful. These results emphasize the variation of the performance capacity of Grid resources and the need to quantitatively assess the performance of each resource.

Low-level performance metrics can be an aid for resource selection in the users' effort to "map" application kernels to appropriate resources.These metrics can be used by schedulers in the resource allocation process, as they can provide a basis for ranking resources using low-level performance metrics. Additionally, the execution of a micro-benchmark on a resource is in itself a validation of the operational state of the resource, playing an important role in tackling problems related to resource allocation. An additional use for the micro-benchmark characterization would be the combination of these performance measurements with other external information such as resource pricing. The low-level nature of the measurements makes no presumptions on the performance characteristics of any application, so these measurements could form the basis for a cost-model for charging for the use of computational resources. Furthermore, users could verify the "advertised" performance of a resource by running these lightweight benchmarks. Another example would be the administrative use of benchmarks to detect problems or faults in the Grid's computational resources.

In the future we plan to further investigate the relation between application performance and micro-benchmark performance in the context of Grid environments. Building on the work presented in this article, we plan to further investigate the use of characterization in scheduling and resource allocation on the Grid, and to investigate the use of micro-benchmarks for automated evaluation of Grid "resource health" and automated detection of degraded performance. We are also working in the direction of automated benchmark parameter selection.

References

[1] EGEE: Enabling Grids for eScience in Europe. http://www.eu-egee.org, (accessed April 2004).

[2] European CrossGrid Project. http://www.crossgrid.org (accessed April 2005).

[3] Interactive European Grid Project Project. http://www.interactive-grid.eu (accessed June 2006).

[4] Open Science Grid. http://www.opensciencegrid.org, (accessed Sep 2005).

[5] Site Functional Tests (SFT). http://lcg-testzone-reports.web.cern.ch/lcg-testzone-reports/sftestcases.html, (accessed Apr. 2005).

[6] Al Aburto. flops.c version 2.0. ftp://ftp.nosc.mil/pub/aburto (accessed Oct. 2004), 1992.

[7] S. Andreozzi et al. GLUE Schema Specification, version 1.2. http://infnforge.cnaf.infn.it/projects/glueinfomodel/ (accessed Apr. 2005).

[8] Greg Chun, Holly Dail, Henri Casanova, and Allan Snavely. Benchmark probes for grid assessment. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*. IEEE Computer Society, 2004.

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

[10] R.F Van der Wijngaart and Michael Frumkin. Alu intensive grid benchmarks. https://forge.gridforum.org/projects/gb-rgs, 2004.

[11] Catalin Dumitrescu, Ioan Raicu, Matei Ripeanu, and Ian Foster. Diperf: an automated distributed performance testing framework. In *Proceedings of the 5th International Workshop on Grid Computing (GRID2004)*. IEEE, November 2004.

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

[14] Eamonn Kenny, Brian Coghlan, George Tsouloupas, Marios Dikaiakos, John Walsh, Stephen Childs, David O'Callaghan, and Geoff Quigley. Heterogeneous grid computing: Issues and early benchmarks. In *International Conference on Computational Science (3)*, pages 870–874, 2005.

[15] John D. McCalpin. *Sustainable Memory Bandwidth in Current High Performance Computers*. Advanced Systems Division Silicon Graphics, Inc., October 1995.

[16] José Carlos Mouriño, David E. Singh, María J. Martín, J. M. Eiroa, Francisco F. Rivera, Ramon Doallo, and Javier D. Bruguera. Parallelization of the stem-ii air quality model. In *HPCN Europe*, pages 543–546, 2001.

[17] Phillip J. Mucci and Kevin London. The cachebench report, 1998.

[18] Rolf Rabenseifner, Alice E. Koniges, Jean-Pierre Prost, and Richard Hedges. The parallel effective i/o bandwidth benchmark: b_eff_io. Message Passing Interface Developer's and User's Conference (MPIDC), March 2000.

[19] P.M.A. Sloot, A. Tirado-Ramos, A.G. Hoekstra, and M. Bubak. An interactive grid environment for non-invasive vascular reconstruction. In *2nd International Workshop on Biomedical Computations on the Grid (BioGrid'04), in conjunction with Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*, Chicago, Illinois, USA, April 2004. IEEE.

[20] Shava Smallen, Catherine Olschanowsky, Kate Ericson, Pete Beckman, and Jennifer M. Schopf. The inca test harness and reporting framework. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 55, Washington, DC, USA, 2004. IEEE Computer Society.

[21] A. Tiramo-Ramos, G. Tsouloupas, M. D. Dikaiakos, and P. Sloot. Grid Resource Selection by Application Benchmarking: a Computational Haemodynamics Case Study. In *Proceedings of the International Conference on Computational Science 2005*. Springer, May 2005. To appear.

[22] G. Tsouloupas and M. D. Dikaiakos. GridBench: A Tool for Benchmarking Grids. In *Proceedings of the 4th International Workshop on Grid Computing (Grid2003)*, pages 60–67. IEEE Computer Society, November 2003.

[23] George Tsouloupas and Marios D. Dikaiakos. Characterization of Computational Grid Resources Using Low-level Benchmarks. Technical Report TR-2004-5, Dept. of Computer Science, University of Cyprus, December 2004.

[24] George Tsouloupas and Marios D. Dikaiakos. Gridbench: A workbench for grid benchmarking. In Peter M. A. Sloot, Alfons G. Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak, editors, *EGC*, volume 3470 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2005.

[25] Reinhold P. Weicker. Dhrystone: a synthetic systems programming benchmark. *Commun. ACM*, 27(10):1013–1030, 1984.

[26] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service in Metacomputing. *Journal of Future Generation Computer Systems*, 15(5-6):757–768, 1999.