GRIDBENCH: RESOURCE PERFORMANCE RANKING AND AUDITING IN COMPUTATIONAL GRIDS

George P. Tsouloupas

University of Cyprus, 2009

Over the recent years the area of Grid Computing has seen an astonishing growth. Grid infrastructures have become the platform of choice for large-scale eScience. The world's largest Grid infrastructure – EGEE – currently comprises 300 sites distributed around the world, petabytes of storage capacity and CPU's in excess of 80,000. The different computing resources in these heterogeneous infrastructures gather impressive and unprecedented computational potential, yet, in order to utilize them, users need mechanisms for selecting the right resources for the right job. Users and Virtual Organization administrators also need end-to-end mechanisms to evaluate the performance of resources and audit resources according to their advertised performance. This can be a complicated process, and when large infrastructures are involved, it becomes unmanageable and prohibitively tedious in the absence of specialized tools.

Performance ranking in a large, shared, heterogeneous and dynamic environment is a complex task because it needs to be done in an efficient and unobtrusive way. At the same time, it has to address many different types of application that come from several Virtual Organizations.

The thesis presents several contributions in the field of resource performance evaluation of computational Grids. A first contribution is the proposal of a methodology for putting *correct*, *meaningful* and *contextualized* performance information at the user's disposal, thus facilitating the *ranking* of computational resources based on customizable criteria. Contextualization is achieved by enriching the measurements with metadata about *when*, *where*, *how* and in many cases *under what circumstances* the measurement is obtained. The thesis proceeds to propose a user-driven approach for ranking resources by employing custom ranking functions.

A second contribution is the introduction of *GridBench*; an extensible tool that has been designed and implemented in the context of this thesis and along the lines of the aforementioned methodology. It allows for context-augmented performance evaluation using several types of benchmarks, ranging from synthetic micro-benchmarks to real-world parallel applications. GridBench features a user-friendly graphical interface that facilitates the invocation of tests and benchmark and the collection, archival and analysis of results. The thesis also introduces a simple, easily obtainable CPU cache metric with very good correlation to real application performance.

A main contribution is the introduction of a methodology for ranking resources based on an application specific combination of low-level, easily obtained metrics. An important component of GridBench, *SiteRank*, implements this methodology and enables the interactive user-driven creation of custom ranking functions.

The methodology and tools are applied through several experiments to the largest production Grid infrastructure in existence today. Among the arguments of the thesis is that the use of evidence-based "measured" data, in contrast to the "quoted" data advertised in information services by resource owners, is imperative. The existing de facto approach for selecting resources according to performance is shown to be insufficient and unreliable.

GRIDBENCH: RESOURCE PERFORMANCE RANKING AND AUDITING IN COMPUTATIONAL GRIDS

George P. Tsouloupas

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

June, 2009

APPROVAL PAGE

Doctor of Philosophy Dissertation

GRIDBENCH: RESOURCE PERFORMANCE RANKING AND AUDITING IN COMPUTATIONAL GRIDS

Presented by

George P. Tsouloupas

Research Supervisor _____
Dr. Marios D. Dikaiakos

Committee Member _____
Dr. Paraskevas Evripidou

Committee Member _____
Dr. Yiannakis Sazeides

Committee Member _____
Dr. Eleni Karatza

Committee Member _____
Dr. Beniamino Di Martino

University of Cyprus

June, 2009

ACKNOWLEDGEMENTS

Over the rather long road to this thesis I have received support and encouragement from many people. My advisor, Prof. Dikaiakos, patiently supported me in my academic life. My family, especially my wife Aphrodite, supported me and endured my balancing act of concurrently going through the PhD study, the bringing up of two children and the starting of a business.

I owe my sincere thanks to Prof. Dikaiakos for his guidance and patience. I thank him for the constructive criticism that was given when necessary and his constant contribution and input. Most importantly I thank him for his insistence on high quality research work and his refusal to accept anything less.

During my employment at the High Performance Computing systems Laboratory I had the pleasure of the company of many wonderful people with which I had many scientific and personal discussions on issues ranging from processors to politics. I'd like to thank, in no particular order, Maria, Wei, Kyriakos, Christiana, Eleni, Koula, Marilena, Asterios, Nicolas and Nicolas, for their input and all the good times that we had together.

During my time at the HPCL, I have had the blessing of working for several excellent research projects and I've had the chance to travel and make friends that I hope will be lifelong. CrossGrid the first project I was involved in proved to be an invaluable experience. I have met and worked with literally hundreds of people. I had the pleasure of collaborating very closely, and to published work with Alfredo Tirado Ramos, Marcus Hardt, Ariel

Garcia. Working as “foot-soldiers” around the time that the first Grid infrastructures were becoming a reality, laid the foundations for a firm understanding of the ins and outs of the Grid, down to the hairy details. I would like to thank Harald Kornmayer for our discussions.

Of all the people that have contributed for me to make it this far, the bulk of the thanks should go to my wife. She has had to put up with my long hours, and she managed to see light at the end of tunnel even when I could not.

CREDITS

1. G. Tsouloupas, M. Dikaiakos, “GridBench: A Tool for Benchmarking Grids.” In P*roceedings of the 4th International Workshop on Grid Computing (Grid2003)*, pages 60-67, Phoenix, Arizona, 17 November 2003, IEEE Computer Society

2. G. Tsouloupas, M. D. Dikaiakos. “GridBench: A Workbench for Grid Benchmarking.” 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, pages 211-225, Springer, June 2005

3. E. Kenny, B. Coghlan, G. Tsouloupas, M. Dikaiakos, J. Walsh, S. Childs, D.O’Callaghan and G. Quigley, “Heterogeneous Grid Computing: Issues and Early Benchmarks.” In *Computational Science - ICCS 2005*, 5th International Conference, Atlanta, GA, USA, May 22-25, 2005, Proceedings, Part III. Lecture Notes in Computer Science, vol. 3516, pages 870-874, Springer, May 2005.

4. A. Tiramo-Ramos, G. Tsouloupas, M. Dikaiakos , P. Sloot, “Grid Resource Selection by Application Benchmarking: a Computational Haemodynamics Case Study.” In *Computational Science - ICCS 2005*, 5th International Conference, Atlanta, GA, USA, May 22-25, 2005, Proceedings, Part I. Lecture Notes in Computer Science, vol. 3514, pages 534-543, Springer, May 2005.

5. G. Tsouloupas and M. D. Dikaiakos, “Ranking and Performance Exploration of Grid Infrastructures: An Interactive Approach.” *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, Barcelona, September 28th-29th 2006.

6. G. Tsouloupas and M. D. Dikaiakos. “Characterization of Computational Grid Resources Using Low-level Benchmarks.” *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing*, Amsterdam, Dec. 4-6, 2006.

7. G. Tsouloupas, M. D. Dikaiakos, “GridBench: A Tool for the Interactive Performance Exploration of Grid Infrastructures”*Journal of Parallel and Distributed Computing*, Elsevier, vol. 67 (2007), pp 1029-1045

8. G. Tsouloupas, M. D. Dikaiakos, “Grid Resource Ranking using Low-level Performance Measurements.”, *The 13th International Euro-Par Conference*, Rennes, France, August 28-31, 2007, Lecture Notes in Computer Science, Springer, 2007, pp.467-476.

Technical Reports and other Grid-Related Publications

9. J. Marco, R. Marco, D. Rodriguez, J. Salt, S. Gonzales, J. Sanchez, A. Fuentes, J. Gomes, M. David, J. Martins, L. Bernardo, M. Hardt, A. Garcia, P. Nyczyk, A. Ozjeblo, P. Wolniewicz, A. Padee, M. Bluj, C. Fernadez, J. Fontan, A. Gomez, I. Lopez, Y. Cotronis, V. Floros, G. Tsouloupas, W. Xing, M. Dikaiakos et al., "First Prototype of the CrossGrid Testbed." *Grid Computing. First European Across-Grids Conference*, Santiago de Compostela, Spain, February 2003, Revised Papers, Lecture Notes in Computer Science series, vol. 2970, pages 67-77, Springer, 2004

10. A. Garcia, M. Hardt and G. Tsouloupas. "Simplified Deployment of a LCG cluster via LCFG-UML". In *Computing in High Energy and Nuclear Physics (CHEP) 2004*, 2004.

11. M. Georgiadou, G. Tsouloupas, M. Dikaiakos, A. Tsapalis, C. Alexandrou, "Running QCD Computations on the Grid", *Technical Report TR-2004-02*, Department of Computer Science, University of Cyprus, July 2004

12. J. Gomes; M. David; J. Martins; L. Bernardo; A. Garcia; M. Hardt; H. Kornmayer; J. Marco; D. Rodriguez; I. Diaz; D. Cano; J. Salt; S. Gonzalez; J. Sanchez; F. Fassi; V; Lara; P. Nyczyk; P. Lason; A. Ozieblo; P. Wolniewicz; M. Bluj; K. Nawrocki; A. Padee; W. Wislicki; C. Fernandez; J. Fontan; Y. Cotronis; E. Floros; G. Tsouloupas; W. Xing; M. Dikaiakos ; J. Astalos; B. Coghlan; E. Heymann; M. Senar; C. Kanellopoulos; A. Tirado-Ramos and D.J. Groen: 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, pages 98-110, Springer, June 2005.

13. M. D. Dikaiakos, A. Artemiou, and G. Tsouloupas. "Towards a Universal Client for Grid Monitoring Systems." In *CDROM Proceedings of the 11th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, 20th IEEE International Parallel and Distributed Processing Symposium, (IPDPS 2006), April 25-29, 2006, Rhodes, Greece. IEEE Computer Society, 2006

14. G. Tsouloupas, M. Dikaiakos, "Characterization of Computational Grid Resources Using Low-level Measurements." Technical Report TR-2004-05, Department of Computer Science, University of Cyprus, October 2004

15. G. Tsouloupas, M. Dikaiakos, "Grid Resource Ranking using Low-level Performance Measurements" Technical Report TR-07-02, Department of Computer Science, University of Cyprus, February 2007

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Grids have emerged as wide-scale, distributed infrastructures that comprise heterogeneous computing and storage resources, and support resource sharing in dynamic, multi-institutional Virtual Organizations (VO's) [33, 34, 35]. 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 [30, 51]. As an example, at the time of this writing, EGEE assembles over 300 sites around the world with more than 80,000 CPU's and about 5PB 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 300,000 jobs per day [30].

There are several, loosely defined, categories of Grids such as *Computational Grids* which focus on high-performance and high-throughput computing, *Data-Grids* that put emphasis on data storage and replication, *Scavenger Grids* which target idle cycles on

non-dedicated user workstations. While Grids come in all shapes and sizes what is described herein assumes an architecture that resembles the one presented in Figure 1. It is the structure adopted by numerous projects like [20, 30, 44, 51].

1.1 A Typical Grid Infrastructure

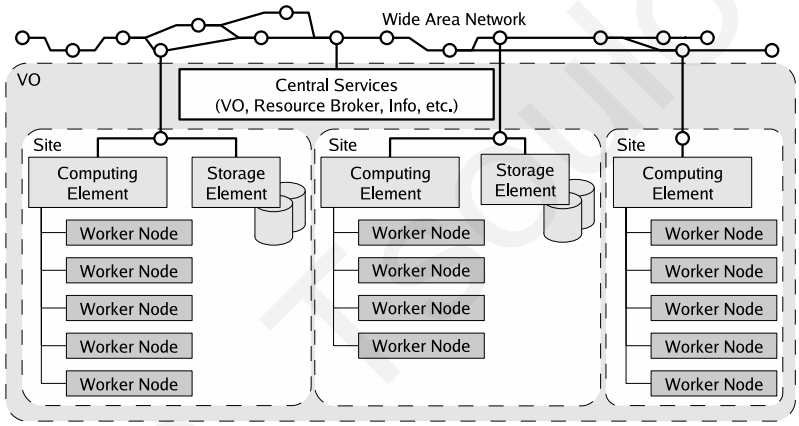


Figure 1: Basic Grid infrastructure architecture.

In this architecture, a Grid Virtual Organization (VO) is made up of a set of geographically distributed sites (resources). Each site contains a Computing Element (the Gatekeeper in Globus [3] terminology) which manages a set of "Worker Nodes" for performing computations. A site may contain a "Storage Element" which is an interface to mass storage. Typically the Computing Element and Worker Nodes have direct (Local Area Network) 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 , an information service, VO membership server etc. The Sites are connected via shared wide-area links.

1.2 Problem Statement

Over the last decade, Grids have grown in size at an admirable rate. An increasing number of resources are put at Grid users' disposal, more applications are ported to the Grid, an increasing number of large-scale scientific experiments target the Grid as *the* platform for their computational needs, while an increasing number of users collaborate to form Virtual Organizations (VO's). This explosion of the Grid in terms of size comes at the cost of users having to deal with this vast, dynamic and heterogeneous infrastructure. Users, be they real end-users or automated resource brokers, need to pick resources that are best suited for specific needs and applications.The selection is generally based on functionality and performance. This raises the problems of: *(i) How does one test, monitor and characterize a large, distributed and volatile system such as the Grid, in a way that imposes minimal disruption?* and *(ii)How can this characterization help users rank resources so that they can identify the right ones?*

While there have been considerable advances in the area of Grid information, monitoring and scheduling systems (See for instance [32, 74, 36, 11, 19, 63]), the fundamental problem of picking the right resources remains. If the user has little constraints on if/when his jobs finish and cost is of no relevance, then this is not much of an issue. If Grid resources come at a price, or the user needs resources to be carefully selected in terms of their performance capacity, the users is left with the difficult task of effectively identifying the right resources. The task is difficult enough when the infrastructure is static and stable, but the problem is worsened in the case of numerous, volatile and frequently changing resources.

Automated decision makers, i.e. Grid schedulers or Resource Brokers, somewhat simplify the task of identifying the right resources, but the source of the problem remains: current systems lack the information source that will deliver performance information about the participating resources. Work has already been done in this direction (see Chapter 2 for systems such as *NWS*, *GRASP* and *DiPerf*) but none of these systems addresses the problem of lack of detailed, accurate and descriptive information about the *performance* capabilities of the candidate resources. Be it by an end-user or a Resource Broker, when a selection needs to be made among candidate resources there are two important pieces of information that need to be accessible: *(i)* Which of these resources are actually operational, and *(ii)* how do they compare in terms of performance.

The classification of resources into operational ones and non-operational ones is not straight forward. The operational status of a resource could be classified at different levels. At a first level, the resource/host must be up and reachable; this can be determined using most Grid monitoring tools. At a second level, the resource must be able to execute jobs that require some functionality; this is accomplished by testing tools. Finally, the resource must be running at full capacity. This can be viewed as the "non-degraded" state or even that the resource is operating at a level that conforms to a service-level agreement; this requires more advanced performance monitoring tools.

In order to determine which resources are operational at any time, most Grids in existence today have some sort of testing system implemented that provides some information about the operational status of the resources (see Chapter 2). Acknowledging that it is practically impossible to *fully* test such a complex system, these testing systems do have

some *fundamental* short-comings in that they are almost never designed to be end-to-end (i.e. they usually run as daemons, not as jobs by end-users), but more importantly they usually overlook problems such as degraded performance because they simply state if a specific function works or not. In order to compare resources in terms of performance capability, today one has to to rely on what resource providers claim by publishing *static* performance information in Grid Information Services. This information is usually manually provided by resource administrators and may not reflect the actual capabilities of the resources. This is due to many reasons, ranging from inaccuracy to failure to reflect internal changes at the resource. What is needed, is a more reliable and accurate resource performance characterization and auditing approach.

Auditing refers to determining the operational status of Grid resources, in terms of functionality, performance, reliability and availability. Meaningful auditing requires that, in addition to functionality testing, one must also address performance, reliability and availability in order to expose resources that operate in a degraded state. The main problems in doing this are:

- Since meaningful auditing implies relevant measurements (i.e. that are of interest to users), one must identify the right metrics that would satisfy the majority of users. On the other hand, users should also be able to define “personalized” tests.

- The underlying infrastructure is large, highly distributed and volatile. Being large and highly distributed makes auditing difficult by making it costly to obtain measurements. Also, deploying and administering auditing experiments in a large distributed architecture is tedious and time-consuming and sometimes non-trivial. The

underlying infrastructure is volatile both in terms of resources entering and leaving the Grid, and in terms of internal changes in the resources (such as hardware upgrades, middleware upgrades or even minor changes in configuration that could have non-trivial effects on performance). This volatility further complicates matters since measurements soon become outdated and need to be repeated. One major factor contributing to this is that machines in large clusters are generally phased out in a period of three years, with about one third of the machines replaced each year. Furthermore clusters are sometimes internally re-structured with machines of different capabilities being dedicated to different queues. Deciding *when* it's necessary to take a new measurement is of key importance.

- There are considerable restrictions to job deployment, mainly in terms of the time resource providers or users are willing to spend auditing the system.

- Knowing exactly what is being measured, and being able to correctly interpret the results is not straight-forward because the Grid employs resource sharing at different levels; resource sharing is in fact one of the main ideas behind the Grid. The Grid runs thousands of jobs concurrently, resources (i.e. clusters) run several concurrent jobs and, in many cases, processes from different jobs may share the same communication channel, memory controller or even CPU. Given the absence of a comprehensive way of expressing policy (e.g. a resource provider declaring precisely how a job from a given user will run at his site), one way of addressing this is by measuring "end-to-end" performance. Furthermore, even if policies of this kind

were somehow uniformly expressed, the problem of validating that the policies are honored still remains.

- Since it is not easy (or cost-effective) to lock down a resource for measurement, auditing measurements, especially in the form of light-weight benchmarks, can be affected by the current state of the resource (i.e. other running jobs). There have to be mechanisms to detect invalid measurements (e.g. the collection and analysis of monitoring information) and filter out the “bad” measurements.

- Some tests or light-weight benchmarks will need to be tuned according to individual resource attributes. Doing this manually when faced with a large set or resources is not really feasible; it needs to somehow be automated.

- It would be beneficial to perform auditing at different levels so that a more complete picture is provided. Considering the architecture described in the introduction, one could perform auditing at the worker-node level, the resource/Computing-Element level, or even at the Grid or VO level. A form of hierarchical auditing could for example expose that a resource gets a low auditing “score” due to the performance of one specific mis-configured worker-node.

Resource monitoring and testing has been recognized as a vital activity in Grid systems. Thus, current large infrastructures such as the EGEE [30] employ a multitude of testing and monitoring tools. At the time of writing the EGEE employs *at least* the SFT [58], gstat [37] and gridice [5]. While running a set of monitoring tools provides

useful information to Grid users and operators, it raises strong and valid concerns regarding the cost of running all of these tools. What's more, delivering resource performance characterization and auditing on a large system like the Grid, imposes additional problems to the problems already faced by Grid Monitoring/Information systems which stem from the largely distributed nature of the Grid. The important difference is that the cost of obtaining the performance characterization itself is by no means negligible, since it may hold up the resource for a considerable amount of time if care is not taken[1] . As an example, the EGEE infrastructure has, at the time of writing, 300 resources centers (i.e. clusters). Running a set of tests and benchmarks (CPU, memory, disk, network and MPI [48] measurements) that may very well occupy 16 CPU's for about 1 hour, would result in the system spending about $300 \times 16 \times 1 = 4,800$ CPU-hours each time the measurements are taken. If the system is measured once a week that would result in about 250,000 CPU-hours per year; if the system is measured once a day that would lead to about 3-4 million CPU-hours per year. Currently infrastructures such as EGEE are tested mainly using very basic functionality tests [58]. These tests run eight times a day, on one worker-node and take about 30 minutes. The fact that the tests are run every 3 hours indicates the need to maintain up-to-date information. Furthermore, most of the resources provide several job queues, where each could arguably be considered a separate resource, raising the number or resources to another order of magnitude. Each queue provides potentially a different type of hardware or software, and usually widely different policies. Users are mapped to one or more of the queues depending on the VO they belong to,

[1] Reading a machine attribute for monitoring, e.g. CPU load, usually takes just a system call that returns under a second, while measuring just a few performance aspects even with reduced confidence using a benchmark takes at least a couple of minutes

therefore a user belonging to one VO has a totally different view of the infrastructure than a user belonging to another VO. The view is different because *(i)* different sites may or may not support a given VO, *(ii)* a site may expose a very different set of resources depending on the VO to which the user belongs, and *(iii)* VO-specific quotas.

The problem can be summarized in the following questions: *(i)* What information is required for a resource ranking that takes into account performance, reliability and other metrics; Can this substantially improve the resource allocation process? *(ii)* How will this information be obtained? *(iii)* How will this information be kept up-to-date in a cost-efficient way? *(iv)* How is can this information be used to provide effective ranking and auditing of Grid resources?

The statement of the problem is simple enough: Given a large, dynamic set of resources and a set of tests, obtain measurements by executing the tests, and then update the measurements as needed in order to provide automated ranking/auditing. To accomplish this one must address the various problems mentioned earlier; i.e. determine *when* to invoke a test based on a set of factors, respect a strict budget, tailor test parameters to individual resources and user requirements, filter results based on a measure of their validity and aggregate metrics based on a hierarchy that reflects the infrastructure. At the same time, coordinate automated invocation of tests in parallel with the interactive invocation of tests by end-users and take into account the ranking "preferences" of users when performing automated ranking. All of these need to be done in a way that is scalable, end-to-end, and minimally disruptive to the infrastructure.

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 [55] 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 [9, 45]. 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 CPU's of each resource, their nominal speed, the nominal size of main memory, the number of free CPU's, available bandwidth, etc. This information is retrieved from Grid information services like the Monitoring and Discovery System of Globus [22] or R-GMA [19, 57]. This approach works well in cases where the main consideration of end-users is to allocate sufficient numbers of idle CPU's in order to achieve a high job-submission throughput with opportunistic scheduling [54].

In several scenarios, however, the reliance on simple matchmaking is not enough: practical experience from the operations of production Grid facilities like CrossGrid [38] and EGEE [30] has shown that many users wish to select and rank the resources 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.

Chapter 2

Grid Resource Ranking and Performance Evaluation

This chapter consists of an overview of related work and the key challenges that Grid resource ranking and performance evaluation poses. Section 2.1 provides an overview of related work. It delves specifically into work that is related to *Grid* performance evaluation with a focus on *conputational* performance. Section 2.3 outlines four key challenges: *Scale and complexity*, *Volatility*, *Heterogeneity* and *Virtualization*. In terms of performance evaluation, they pose the main challenges inherent in Grids and that need to be addressed.

2.1 Related Work

The GridBench tool and the work presented in this thesis relate to two general areas: *(i)* infrastructure performance monitoring and testing platforms and *(ii)* Grid benchmarking and performance evaluation. These areas are outlined in sub-sections 2.1.2 and 2.1.3. Before we proceed, it would be beneficial to elaborate on the scope of this work with regard to performance evaluation and benchmarking.

2.1.1 Performance Evaluation, Benchmarking and the Scope of this Work

Performance evaluation and benchmarking have applications and requirements that are directly dependent on the perspective of interested party. A consumer in an effort to buy a desktop with a good price/performance ratio, a computer gamer with high 3D acceleration requirements, a researcher in computer architecture or a chip manufacturer evaluating their architecture, or an engineer out to buy a computational cluster, all have different perspectives, requirements and expectations from a benchmark. In the context of the Grid, where users or resource brokers must select resources in a heterogeneous dynamic environment the problems (outlined in Sections 1.2 and 2.3) and requirements (Section 4.2) differ.

One example were perspectives may differ is the issue of "reproducibility". Reproducibility is generally accepted as the attribute of a *good* benchmark; it is in fact one of the main principles of the scientific method. Researchers strive to achieve it even when the nature of the measurement makes it difficult to reliably reproduce [39]. If we were to enforce reproducibility in the Grid setting we would have to do away with key features and aspects of the Grid, such as dynamicity and the shared nature of resources. If we did this, measurements would have little value given the motivations behind measuring performance of Grid resources; these are the issues that we needed to address in the first place. From the point of view of the Grid user, the actual "experienced" measurements –even in a statistical sense– obtained in an end-to-end fashion, is preferred over "reproducible" measurements.

Somewhat related to reproducibility is *accuracy*. Exhaustive and accurate measurements that address individual aspects of a CPU's architecture are probably overkill, as this usually comes with a high overhead of running relatively large suites of benchmarks (e.g. 25 benchmarks just for CPU in SPEC2006). It becomes more of an overkill if what is required is a ranking of the performance of resources and not their "absolute" performance.

Long-standing and recognized work such as the EEMBC [29] benchmarks, the SPEC [62] benchmarks and SPLASH [59, 75] address many areas of benchmarking, with a wide scope. It is important to pay attention tothis influential work, and how it has evolved, in order to get a grasp on what is widely considered important, even though they are not *directly* applicable in the Grid context.

Another emerging work from a multi-disciplinary group of researchers is the 'Thirteen Dwarfs" [8]. This is an extension of previous work, the "Seven Dwarfs" [17]. The goal of this work is to capture patterns of computation and communication common to a class of "important" applications. The current "dwarfs" *(1)* Dense Linear Algebra, *(2)* Sparse Linear Algebra, *(3)* Spectral Methods, *(4)* N-Body Methods, *(5)* Structured Grids, *(6)* Unstructured Grids, *(7)* MapReduce, *(8)* Combinational Logic, *(9)* Graph Traversal, *(10)* Dynamic Programming, *(11)* Backtrack and Branch-and-Bound, *(12)* Graphical Models and *(13)* Finite State Machines. While being implementation independent, they present a method for capturing the common requirements of classes of applications [13]. This is rather exciting work, as it aims to address the issue of finding a suite of codes that could characterize a large portion of applications.

2.1.2 Infrastructure Performance Monitoring

"Performance monitoring" is distinct from generic infrastructure monitoring tools. Monitoring tools that can be applied to the Grid are in abundance [40, 28, 58, 36, 5] but they mostly address network performance, monitoring and status; they are not covered here. Performance monitoring tools actively invoke some test/workload and observe the outcome. This is in contrast to generic monitoring tools which simply read machine attributes and report them. One approach to infrastructure performance monitoring is the one taken by the **Grid Assessment Probes** [15] (GRASP). These probes test and measure performance of basic Grid functions such as job submission, file transfers, and performance of Grid Information Services. The GRASP employ 3 types of probe: the *3-Node probe*, the *Circle probe* and the *Gather probe*. The 3-Node probe is meant to represent a pipelined application and involves: *(1)* the transfer of a 100MB file for a *Database Node* to a *Compute Node* and *(2)* the processing of this data at the *Compute Node* generating another file which is then transferred to a *Results Node*. The *Circle probe*, which is meant to represent an application performing a token passing-passing operation around Grid sites, involves sending a 100MB file around a ring of processes. The *Circle probe* performs data-integrity validation at each step, and finally compares the original file with the resultant file at the originating node, as soon as the loop is closed. The *Gather probe* is similar to the *3-Node probe*, the difference being the existence of multiple *source nodes* instead of one *database node*. The tool aims to test basic Grid functionality, so in that sense it is more of a "testing" tool than a performance measurement tool. Yet, as the three *probes* suggest, the GRASP can be used to evaluate performance for the perspective of

data-transfer intensive Grid applications; it is not clear if/how this can provide insight that can be directly applied to ranking resources according to performance.

An approach similar to the one taken by GRASP, without the emphasis on data-transfer, but with an emphasis on *service* performance, is the approach taken by **DiPerF** [27]. DiPerF 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 DiPerF framework consists of two main components: the *testers* and the *controller*. The controller receives the "client code" and the address of the target service and is responsible for starting and coordinating the *testers* in order to carry out the experiment. The DiPerF framework reports the following metrics: *(1) Service response time* – the time taken to "serve" a request minus the network latency and the duration of execution of the "client code"; *(2) Service throughput* – number of successful job completions in a short time interval; *(3) Offered load* – concurrent service requests per second; *(4) Service utilization* – ratio of satisfied requests for one client (tester) to the total number of requests; finally *(5) Service Fairness* – the ratio of jobs completed to service utilization. This work focuses on measuring the performance of services and it is highly geared for that; it applies a workload of service requests. It is unclear if there is some way to use this to deliver metrics about the computational resource itself, not the service that provides the interface to the resource.

Employing an architecture similar to DiPerF, the **Inca test harness and reporting framework** [61] is a system that aims to automate the testing of resources, automate

resource data collection, perform resource verification and monitor service agreements. Inca constitutes of 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 employed by the TeraGrid [64] project for monitoring and performance data collection.

Similarly, a more direct, strictly testing and very system-specific approach is the one taken by the **Site Functional Tests** (SFT) [58]. The SFT's are tests that are periodically executed in order to evaluate the functionality of different middleware at Grid sites participating in EGEE [30]. The tests are scheduled to run several times daily using a simple cron job. Both Inca and the SFT are highly geared towards their respective specific infrastructures, and employ middleware specific tests that test such things as job-submission, file replication and the accessibility of services.

On the borderline between *testing* and *benchmarking* tools, the **Application Control and Monitoring Environment** (ACME) [50] is a system intended to facilitate the tasks of benchmarking, testing and management of Internet-scale systems. The ACME consists of two parts: the ENgine for TRiggering Internet Events (ENTRIE), and the Internet Sensor In-Network aGgregator (ISING). A user can create an XML definition and invoke ENTRIE, which generates events (based on timers, completion events and sensor events) and passes them to ISING. ISING utilizes a tree-structured overlay network to aggregate sensor data and provide measurements. From the point of view of benchmarking, the

environment can be used to automatically invoke large numbers of jobs and take measurements, but no specific benchmarks are proposed. Also this approach (as well as the ones of GRASP, DiPerf and INCA) does not target end-to-end tests.

Infrastructure Monitoring tools such as the **Network Weather Service** [74] (NWS) 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 of CPU "availability", but this is different from benchmarking in that it provides only the "instantaneous" capacity of a machine, 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.

2.1.3 Grid Benchmarking

One way to evaluate the performance of a system as a whole is by taking measurements through the application of a workload. Workloads are either real (i.e. taken from an actual live production environment) or synthetic. There are arguments for both sides as to which is preferable. The main argument for synthetic workloads is increased reproducibility and easier to control experiments, yet a synthetic workload may not reflect reality. The synthetic workload is the approach taken by GrenchMark [42]. GrenchMark is a framework for synthetic grid workload generation and submission. It is extensible by allowing the inclusion of new types of grid application in the workload generation and it

allows workload parameterization. It contains a component that can process the generated workload units and create job descriptions targetted at specific Grid middleware. The focus is on the performance of the Grid system as a whole and not on the performance of the constituting resources.

The **ALU Intensive Grid Benchmarks** [23] (AIGB) aim to measure the performance of Grids via pre-defined work-flows using the established NAS Parallel Benchmarks [10] as computational kernels. The AIGB, which are endorsed by the Global Grid Forum Grid Benchmarking Research Group, define a set of data-flow graphs (DFG's). The four data-flow graphs - "Embarrassingly Distributed", "Helical Chain", "Visualization Pipeline" and "Mixed Bag" - represent four main types of Grid applications. The *Embarrassingly Distributed* (ED) DFG represents high-throughput applications, such as parameter sweeps, that are made up of large sets of independent tasks. The *Helical Chain* DFG represents long-running simulation applications in the form of work-flows. Each task in the HC DFG/work-flow depends on the successful completion of the previous task, which results in a linear DFG. The *Visualization Pipeline* DFG represents applications where the (intermediate) results are visualized as they are produced while the simulation is in progress. Finally, the *Mixed Bag* DFG is very similar to the Visualization Pipeline but focuses on asymmetry by introducing different amounts of computation and the transfer of different amounts of data between tasks. The AIGB are pencil-and-paper specifications describing several problem-sizes or "classes". This approach aims to capture the performance of Grids as several scales by varying both the amount of computation performed by each task as well as the size of the DFG. In contrast to the AIGB, the work

presented here focuses on testing and lightweight benchmarking for the functional status and performance characterization of Grid resources, while the AIGB are pencil-and-paper definitions of workflow-type applications.

The focus of the GridBench tool that is described in the following sections, is rather different from what is described above. 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; thus, they do not address the computational performance of a Grid site. Most importantly, these testing tools do not provide direct interactivity with the user, and are generally not targeted at the user. The measurements are mostly set up by a central administrator and the end-users can simply review reports that are published on a web-page.

Related to the use of benchmarking for ranking resources on the Grid for the purpose of resource selection, benchmarking as a data-source for resource-brokering is explored in [2]. This work suggests the application of *weights* to different resource attributes and the use of *application benchmarks* to obtain a ranking that can eventually be used for resource brokering; this has also been suggested in previous work of the author [66]. In contrast, this work enables users to interactively obtain their own custom measurements, and to specify their own custom ranking functions that make use of an extensible set of low-level metrics.

Table 1: Grid Performance Evaluation: Overview of Related Work

	GRASP	DiPerf	Inca	AIGB	GrenchMark	Dwarfs	GridBench
Interactive/user-driven							●
User End-to-end	○		●	○	●	○	●
Test-Centric		●	●		●		●
Service-Centric	●	●	●		●		●
Repository/Hist. data			●		●		●
Analysis							●
Grid Workloads		●		●	●		●
Infrastructure Independent	●			●		●	●
Computational Performance	●			●		●	●
Detailed Context							●

Table 2.1.3 provides an overview of related efforts in Grid Performance valuation. While not aiming to be comprehensive in terms of the functionality and focus of the differen related works, it does provide a comparison of GridBench with the rest of the tools or approaches. A full dot "●" denotes that the tool supports the functionality. GridBench pre-dates all of the related tools (AIGB is not a tool), yet as seen in the table, GridBench still differs in three main areas: *User End-to-end*, *Analysis* and *Detailed Context*. With regards to *User End-to-end*, some efforts are marked with "○" to denote that while there is no explicit mention for the specific functionality in the tool's literature, it could probably be easily modified to support it.

2.2 The Grid Context

This work assumes 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 (VO's). In Figure 2 a model of this infrastructure is presented, inspired by the architecture of large-scale Grid testbeds such as LCG [44],

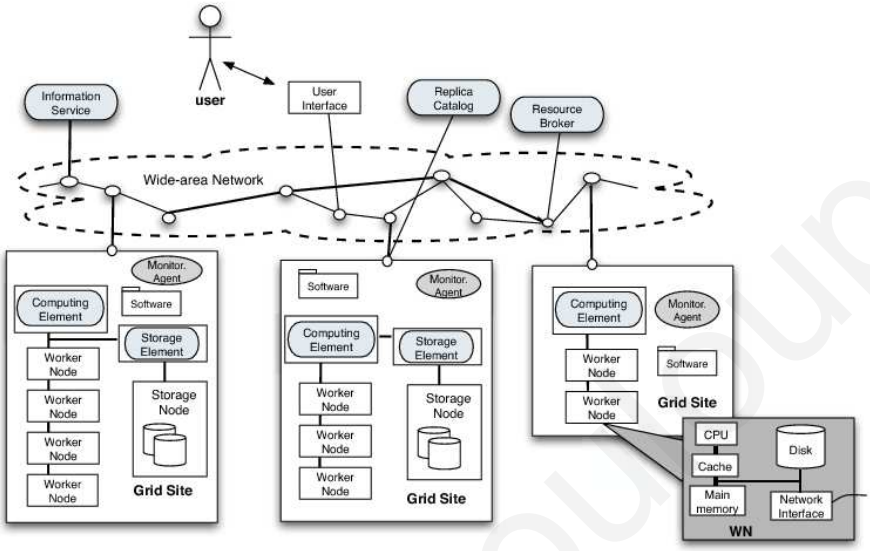


Figure 2: A more detailed (but by no means complete) Grid architecture.

EGEE [30] and CrossGrid [38]. A Grid site comprises a cluster of *Worker Nodes* (WN), which are typically off-the-shelf PC's 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.

The operation of the Grid infrastructure is supported by a number of central services, such as the Grid Information Service and the Resource Broker. The *Grid Information Service* (GIS) publishes information essential to the operation of the infrastructure: static

descriptions of resources, services, software, and applicable policies, and dynamic representations of resource status, performance, and availability. The *Resource Broker* (RB) undertakes the matchmaking between resource requests and available resources.

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 CE's apply for the various VO's.

2.3 Key Challenges

To support the selection and ranking of resources using configurable on-demand tests in the context described above, one needs to address a number of challenges that arise from the inherent characteristics of Grids (an extensive discussion on these challenges can be found in [24]):

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 numerous 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 open wide-area networks, which are used to interconnect Grid sites and which are shared with millions of Internet users. Access policies that some Grid sites apply, allowing the dynamic co-allocation of multiple Grid jobs on the same Worker Node, further complicate the situation. 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 [43, 56].

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 CE's 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 VO's may have a totally different view of the infrastructure's performance.

Chapter 3

Context-augmented performance measurements: GBDL

A number by itself has little meaning; this holds true, of course, for performance measurements as well. For example, the measurement $1,242\ MFlop/s$, while it may be clear to some that it is some sort of performance metric for some machine, it is essential that this measurement is put into context. First, it is important to know *where* and *how* this measurement was obtained. As a first, obvious step, one can start by specifying $\{$*Resource X* , *Benchmark A* , $1,242 MFlop/s\}$. By introducing this *context* the measurement becomes useful. To make this information even more useful, especially in a Grid setting, one must provide a lot more information. The purpose of this chapter is to detail my approach for defining Grid resource performance measurements and putting them into context.

3.1 Contextualizing resources

Taking a closer look, while keeping the architecture of the Grid in mind, the specification of *Resource X* and *Benchmark A* is rather vague: Grid *Resource X* probably

encapsulates a cluster, and quite possibly, this cluster is heterogeneous (i.e. it contains heterogeneous nodes). This implies that the measurement could in fact reflect the performance of any one of the heterogeneous nodes. Moreover the Grid is of a dynamic nature; resources are added, removed, upgraded and reconfigured. The *virtualization* of resources together with the dynamicity of the Grid call for a more detailed description of the context in which the measurement was taken.

While context of a measurement can be arbitrarily complex. The context that accompanies a performance measurement on the Grid should encode information about the following: *(i)* the definition of a specific test or benchmark invocation with specific parameters (the *what*); *(ii)* the target resources (the *where*); *(iii)* the time of execution (the *when*); *(iv)* the status of the target machines during execution collected through monitoring (the *state*); and, obviously, *(v)* the resulting metrics (the *result*).

Specifying exactly how a benchmark is to be executed on the target resource is essential for reliable evaluation of the result. A meaningful analysis or performance comparison is only valid if one knows *what* was executed, such as **which benchmark** with **which parameters** and **which work-load**.

The *virtualization* of resources bears a strong impact on an effort to evaluate the performance of resources. Depending on the type of Grid infrastructure at hand, there can be several layers of virtualization. Taking the most common type of computational Grid as an example, a computational task (i.e. a job) is simply “submitted to the Grid”. The Grid, or rather the Grid services such as the Resource Broker, hide the underlying Grid

resources, providing the first level of virtualization. The job is propagated to a Grid computational resource, which itself encapsulates a set of computational nodes – usually a cluster – providing a second level of virtualization. To account for this, information on *where* a measurement was performed must be maintained in the measurement's context. The level of detail that should be maintained depends on required level of detail of the performance evaluation and analysis, i.e. an in-depth analysis of performance generally requires a detailed context.

Large Grids are quite dynamic in several respects. Maintaining *when* a measurement is taken is important for two reasons:

- Measurements taken at one point in time may not reflect the performance of a resource as it is continuously reconfigured or updated.
- The behavior of a resource of a time may convey a lot of useful information, such as reliability or predictability of performance.

Capturing the *state* of the machine under measurement is also important since it can help explain the results. The fact that Grid paradigm leaves all of the resource's policy and configuration at the discretion of the resource provider on the one hand, and the *virtualization* of resources on the other, result in cases where odd performance-wise behavior of resources is hard to explain. It is often the case that resources (intentionally or unintentionally) allow several independent jobs to run on the same hardware, thus seriously affecting the performance of individual jobs. Maintaining the state of a machine during the time that a measurement is taken can provide a lot of insight. Obtaining machine state to a high level of detail is quite difficult since one has to account for a lot of the machine

internals. Nevertheless, even simple monitoring of CPU and main-memory usage can provide a lot of information.

To address these issues and requirements, I have introduced the GridBench Definition Language, which is the subject of the following section.

3.2 The GridBench Definition Language

The GridBench Definition Language (GBDL) has been introduced in order to describe the measurement's context. GBDL is an XML-based language that encodes basic information required to describe and execute tests and benchmarks. Moreover, it maintains the context of the measurements since it supports the annotation of test or benchmark definitions with performance-related metadata representing the conditions of a particular experiment and the metrics derived from that experiment.

Instead of using an existing job description language, such as RSL[3] or JDL[52], GBDL was introduced in order to: *(i)* allow for a standardized definition of tests and benchmarks that is independent of the underlying middleware platforms used to execute them; *(ii)* enable the specification of the monitoring information that should be collected during a benchmark execution from an available Grid monitoring system; and *(iii)* serve as a container for the context-augmented results.

3.2.1 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

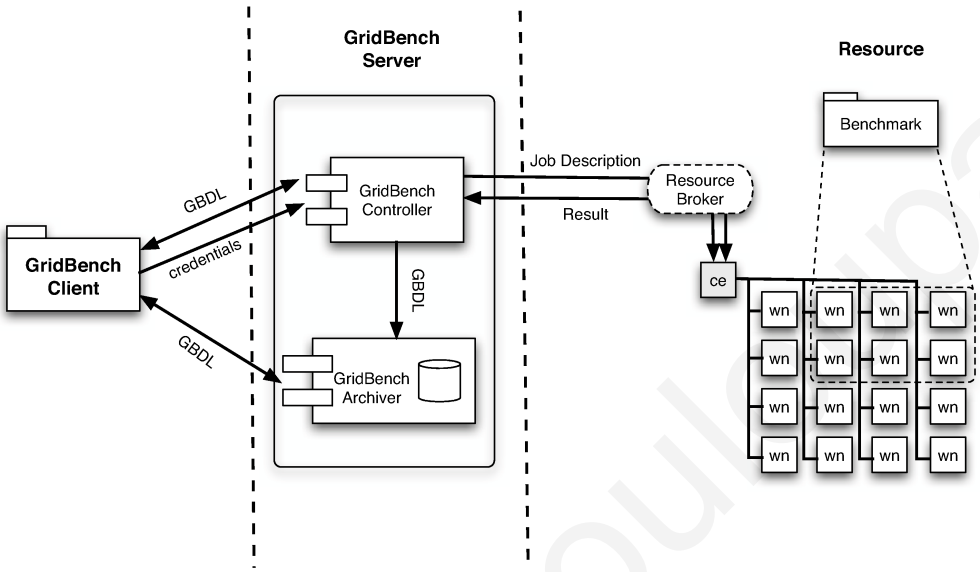


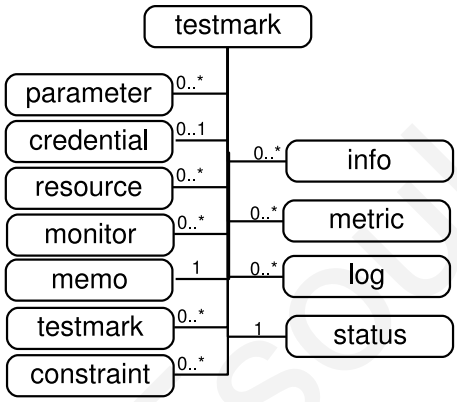
Figure 3: GridBench Component communication.

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 Figure 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.

3.2.2 Syntax

A high-level structure of GBDL documents is presented in Figure 4-top. According to the GBDL syntax, the top-level XML element in a GBDL document is the <**testmark**>.




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

Figure 4: **Top:** A schematic overview of GBDL; shown in rounded boxes are the main parts of a GBDL document. **Bottom:** A real example of GBDL definition, showing a *flops* micro-benchmark definition that requires 2 CPU's on 2 separate worker-nodes at *ce101.grid.ucy.ac.cy*.

<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 Figure 4-top. An example of a GBDL definition is given in Figure 4-bottom. The elements on the left side of the tree-structure of Figure 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 following sub-sections outline the different elements of the language and describe their syntax, semantics and usage.

3.2.2.1 The *testmark* element

The <**`testmark`**> element is the top-level element of a GBDL document. It is the element that encompasses the context and the outcome of a test or a benchmark. Furthermore, a <testmark>, can contain nested <testmark> elements, mainly useful for the definition of work-flow benchmarks.

The <testmark> element defines the following attributes:

- **id**: A unique identifier for the <testmark>.
- **name**: A name for the test or benchmark .
- **jobtype**: The type of job that will be used to execute the benchmark. I can take the values of *MPI* or *plain.*

- **wfid**: An identifier for the <testmark> to be used in defining workflows. It is referenced by the *wfref* attribute of the <constraint> element.

3.2.2.2 The *parameter* element

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:

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. For example, in the following definition the <parameter> element states which Resource Broker (RB) to use for submitting the benchmark.

```
<parameter name="RB"
           type="conf">rb101.grid.ucy.ac.cy</parameter>
```

user parameters are test-specific and initialize the input parameters of the test executable. For example, specifying the number of loops to run in a specific benchmark.

```
<parameter name="loops"
           type="user">100</parameter>
```

3.2.2.3 The *metric* element

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>
```

3.2.2.4 The *credential* element

The <**credential**> element carries the credentials of the user submitting a test to a Grid, such an x509 proxy certificate, in hexadecimal form.

3.2.2.5 The *resource* element

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 CPU's to be requested from that site, and how the CPU's should be distributed on that site's Worker-Nodes. 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 a job submission service.

3.2.2.6 The *monitor* element

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 plug-in of the GridBench Controller.

3.2.2.7 The *constraint* element

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*. An example of a workflow[1] definition in GBDL is given in Figure 5. This workflow corresponds

[1]This work-flow is from the "Visualization Pipeline" benchmark of the ALU-Intensive Grid Benchmarks [23].

to the workflow illustrated in Figure 6. In the given example there are three kernels (BT, MG and FT) the order of execution is specified by assigning a unique *wfid* for each nested testmark and referring to it by *wfref* attribute in the *constraint* elements.

A *constraint* of type *corequisite* means that the testmark containing this constraint should be started after the testmark specified by *wfref* starts. A *constraint* of type *pre-requisite* means that the testmark containing this constraint should be started after the testmark specified by *wfref* terminates.

Figure 7 shows the GBDL description for a High-Performance Linpack benchmark execution on `ce101.grid.ucy.ac.cy` using 16 CPU's.

The GBDL syntax also provides 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.

3.2.2.8 The *info* element

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"/>
```

3.2.2.9 The *memo* element

The <**memo**> element holds a short description of the test defined by the enclosing *testmark*.

```
<testmark name="VP" wfid="1135076543003036000">
   <testmark name="BT" wfid="BT1" jobtype="MPI">
   ...
   </testmark>
   <testmark name="MG" wfid="MG1" jobtype="MPI">
       ...
      <constraint wfref="BT1" type="corequisite">
      </testmark>
   <testmark name="FT" wfid="FT1" jobtype="MPI">
       ...
      <constraint wfref="MG1" type="corequisite">
   </testmark>
   <testmark name="BT" wfid="BT2" jobtype="MPI">
       ...
      <constraint wfref="BT1" type="prerequisite">
   </testmark>
   <testmark name="MG" wfid="MG2" jobtype="MPI">
       ...
      <constraint wfref="BT2" type="corequisite">
   </testmark>
   <testmark name="FT" wfid="FT2" jobtype="MPI">
       ...
      <constraint wfref="MG2" type="corequisite">
      <constraint wfref="FT1" type="prerequisite">
   </testmark>
   <testmark name="BT" wfid="BT3" jobtype="MPI">
       ...
      <constraint wfref="BT2" type="prerequisite">
   </testmark>
   <testmark name="MG" wfid="MG3" jobtype="MPI">
       ...
      <constraint wfref="BT3" type="corequisite">
   </testmark>
   <testmark name="FT" wfid="FT3" jobtype="MPI">
      ...
     <constraint wfref="MG3" type="corequisite">
     <constraint wfref="FT2" type="prerequisite">
   </testmark>
</testmark>
```

Figure 5: Workflow definition of the GGF AIGB benchmark “VP”shown in Figure 6.

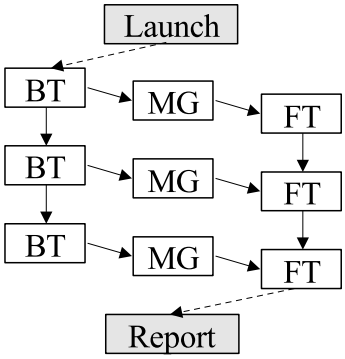


Figure 6: The AIGB VP benchmark.

```
<testmark name="HPL"
     xmlns="http://gridbench.ucy.org/definition"
     id="11350023837373000"
     tstart=""
     duration=""
     node=""
     jobtype="MPI">
  <parameter name="problem_size"
             type="user">180</parameter>
  <parameter name="blocks"
             type="user">40</parameter>
  <parameter name="p"
             type="user">4</parameter>
  <parameter name="q"
             type="user">4</parameter>
  <parameter name="threshold"
             type="user">16.0</parameter>
   . . .
  <parameter name="L1"
             type="user">0</parameter>
  <parameter name="mem_alignment"
             type="user">8</parameter>
  <resource
      name="ce101.grid.ucy.ac.cy:2119/\
            jobmanager-lcgpbs-dteam"
      cpucount="16"/>
</testmark>
```

Figure 7: Definition of an instance of the High Performance Linpack benchmark. (Several parameters omitted to conserve space.)

3.2.2.10 The *log* element

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>
```

3.2.2.11 The *status* element

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.

3.3 Translating GBDL for execution

GBDL encodes the basic information that is required for executing the test or benchmark job and obtaining the measurement. Different middleware require different Job Definition Languages such as RSL and JDL. Consider the following GBDL definition:

GBDL

```
<benchmark name="epwhetstone" date="20031209105938" type="mpi">
  <location>
    <resource name="ce01.lip.pt"
              cpucount="8" jobmanager="jobmanager-pbs"/>
  </location>
  <parameter name="executable" type="attribute"
             dataType="0">epwhetstone</parameter>
  <parameter name="execpath" type="attribute"
             dataType="0">/opt/cg/gridbench/bin</parameter>
  <parameter name="stage_executable" type="attribute"
             dataType="0">manual</parameter>
  <parameter name="nloops" type="value"
             dataType="0">10000</parameter>
</benchmark>
```

The information included in the execution definition of the benchmark can be put into middleware specific definitions. The following definition is in the syntax of the Job Definition Language employed by EGEE:

JDL

```
#Automatically Generated by GridBench
StdOutput       = "std.out";
StdError        = "std.err";
Arguments       = "10000";
InputSandbox    = {"/opt/cg/gridbench/bin/epwhetstone"};
Executable      = "epwhetstone";
JobType         = "mpich";
NodeNumber      = 8;
OutputSandbox   = {"std.out","std.err"};
Requirements    = other.CEId == "ce01.lip.pt/jobmanager-pbs";
```

Another example of translation to a middleware specific language is the following, which specifies how to execute the benchmark on a Globus 2 infrastructure, using the Resource Specification Language.

RSL

```
&(resourceManagerContact="ce01.lip.pt")
 (executable=$(GLOBUSRUN_GASS_URL)
                              /opt/cg/gridbench/bin/epwhetstone)
 (jobtype=mpi)
 (count=8)
 (arguments=10000)
```

Further examples can be found in Appendix A.

3.4 Machine State Monitoring During Measurement

Monitoring the resources under test, in other words collecting some important aspects machine state, can provide considerable insight. First, the collected data can be used to investigate why certain results are not as expected. For example, a specific measurement may be noticeably lower that previous measurements of the same resource. By observing the amount of used memory and swap usage one may determine the cause of the problem, i.e. other processes taking up large amount of memory. Second, the data can be used to filter results based on some criteria. For example one could filter based on whether the resource is being shared between jobs.

Monitoring of the resources sub-systems (CPU, RAM, Disk bandwidth etc.) can be accomplished in different ways. One way that has proved useful is the invocation of a

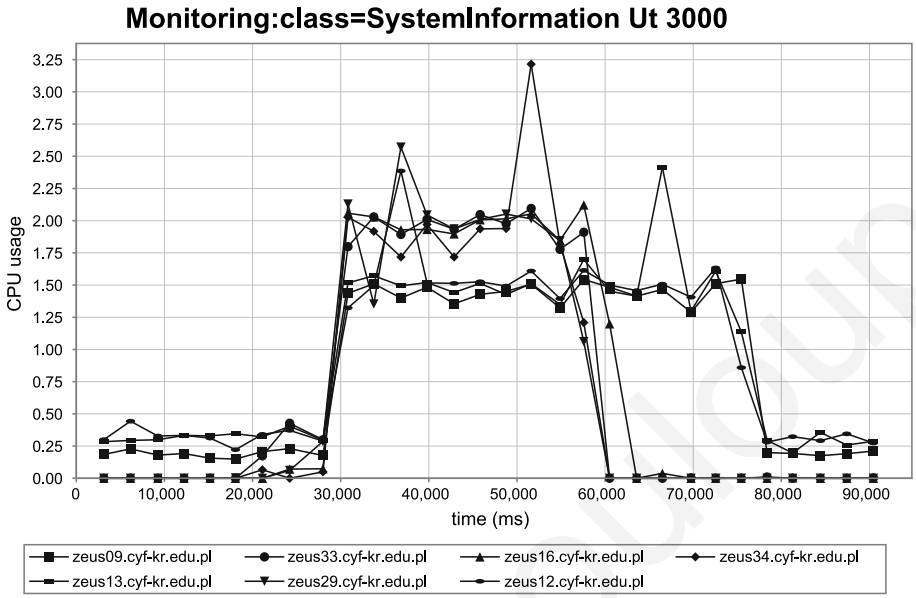


Figure 8: Monitoring machine state during a benchmark

custom script that runs in parallel with the job and collects data directly from the Operating System. Monitoring services can also be used provided the underlying infrastructure allows it.

To illustrate the problem, one can take an example where the execution of an embarassingly parallel (MPI) benchmark was giving suspicious results. Figure 8 shows the CPU usage (user-time) collected during the application execution (i.e. between the 30,000ms mark and the 80,000ms mark).

As indicated by the very high CPU usage on some of the nodes prior to the beginning of the measurement (indicated as “Approximate benchmark duration”), some nodes were putting all of their CPU time into this application, some others were busy doing other things. It is known for a fact that the application itself, being embarassingly parallel, treats all nodes equally and is not responsible for this kind of behavior. The point that can be drawn out of this is that the measurement is not necessarily indicative of the performance

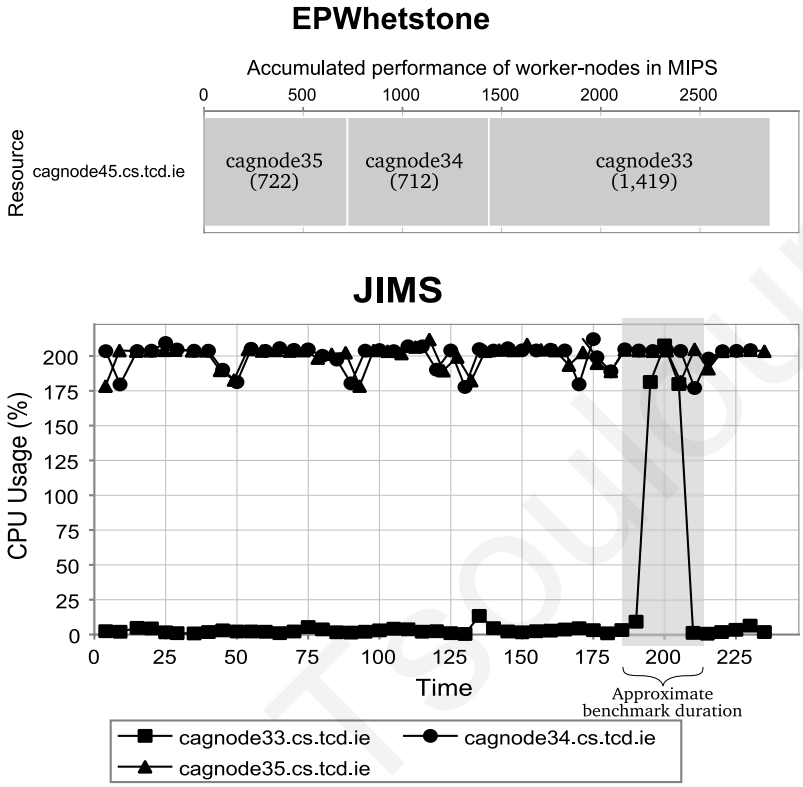


Figure 9: Monitoring a CPU benchmark execution, using the JIMS monitoring service.

of the resource. This is not to say that the measurement is useless, only that it cannot be taken for face-value. Such analysis is the subject of following chapters.

Referring to an actual example, with real performance measurements may provide an even better understanding. Figure 9(top) shows an example of monitored benchmark execution where the results are invalid due to non-exclusive use of the resources by the benchmark. Figure 9(bottom) shows the CPU usage (automatically collected by the JIMS[12] monitoring system) while Figure 9(top) shows the measurements. In Figure 9(bottom) there is significant CPU usage on two of the three worker nodes before and after the "gray" area, which indicates the duration of the benchmark execution.[2] As a

[2]Values of 200% for CPU usage are due to the worker nodes having dual CPU's. The benchmark was executed with two processes on each worker node

result, the measured performance of **cagnode34** and **cagnode35** severely suffers, as shown in Figure9(top), due to CPU usage by other applications.

Chapter 4

Grid-Resource Performance Evaluation in Production Environments

The task of performance evaluation in large, production Grid environments goes well beyond invoking a set of benchmarks on a few resources. In fact the overhead and complexity of obtaining useful results in large, live Grid environments has been a key challenge and motivation behind this work.

4.1 GridBench

The GridBench platform was designed and implemented with these issues in mind. The general goals and requirements of this platform can be summarized in the following:

1. Provide the ability to execute benchmarks and tests on an underlying Grid infrastructure;

2. Manage potentially hundreds of thousands of benchmark invocations;

3. Collect and organize the results and allow them to be easily shared among users;
4. Provide mechanisms to easily customize or add new benchmarks;
5. Provide extension points for the easy integration of different Grid middleware, without the need for new benchmark definitions or implementations;
6. Enable users to analyze results and create performance comparison charts;
7. Allow users to utilize the results and apply that knowledge in order to improve the performance of their applications of the performance of the Grid altogether;
8. Appeal to end-users and Grid administrators alike via a user-friendly interface.

The next section goes into more detail about why these general goals and requirements are important, and how the platform goes about addressing them.

4.2 Goals and Requirements

4.2.1 Executing Benchmarks

While seemingly trivial, benchmark execution in a Grid environment poses several subtle challenges. Before a user can test or measure the performance of resources belonging to her VO, she must first obtain the set of target resources, i.e. perform resource discovery. Resource discovery is handled differently in different Grids, therefore a tool that tackles benchmark execution should also address resource discovery. This usually implies an integration with Grid information services. Sometimes this implies several

different services, or even communicating to resources directly in order to obtain a complete enough picture of the Grid.

Next, it is important for such a tool to be able to handle deployment. Deployment of software in Grids is *mainly* handled in two ways: *(i)* pre-installing the software on all worker-nodes belonging to all Grid sites, and *(ii) staging*, i.e. putting in place, the required files just before invoking the processes. Deployment by pre-installation, is a good fit for applications that are active for long periods of time, and requires considerable overhead in keeping track of what is installed where, overhead in establishing agreements with resources owners in order to allocate special storage locations. Staging the files just before execution is a better fit for testing and benchmarking since it maximizes flexibility and minimizes overhead, especially if the frequency of running tests or benchmarks is not very high. A tool to benchmark Grids must be able to handle staging of executables and configuration files using different middlewares, regardless of the level of file-staging support of the middleware.

Since the tool is meant to be used by Grid users, it is preferable that users would *not* need to install the grid middleware on their personal computers in order to be able to run benchmarks and analyze results. [1]

Finally, and most importantly, the execution of benchmarks should be carried out in a end-to-end fashion. Tests performed in ways other than how a regular user job is actually executed may not sufficiently test the system. Benchmarking metrics obtained not by using regular job pathways may – and probably *will not* – reflect the performance

[1]The important implication of this is that depending on the architecture and the availability of interfaces or API's of the underlying infrastructure, the system under consideration will need to provide proxy services.

experienced by end users. It is essential that a tool the performs benchmarks does so in an end-to-end fashion, representing the performance that is to be experienced by the user.

4.2.2 Managing invocations

A Grid testing and benchmarking tool should be able to manage numerous concurrent benchmark invocations. To this end, the status of benchmarking jobs needs to be monitored, through integration with several job-submission and information services. This is important for two reasons: *(i)* In order to determine possible problems with the infrastructure; and *(ii)* enable the execution of simple workflows such as *invoke - wait for completion - fetch output*, and more complex workflows such as workflow benchmarks.

For large benchmarking experiments, such as benchmarks involving numerous jobs, or benchmarks involving workflows, it is important that the system has at least some components that run as a *daemon* in order to manage benchmark executions, thus freeing up the users client machine.

The tool should handle the fetching and verification of benchmark/test output. While the verification of the outputs of tests and benchmarks is usually an application specific task, some tests should be performed on the results to ensure that the results valid. As a simple example, following a failure on a Grid node the job may return partial results. As it is many times the case, the job may return a status of success even if there was a failure on the node. Moreover the tool should allow benchmark-specific verification of the output.

Closely related with the verification and validity of the results is the monitoring resources during benchmark execution. In traditional benchmarking of high performance computers, or benchmarking in general for that matter, it is generally accepted that the machine under measurement should be otherwise idle. That is the machine should be doing nothing else other than running the benchmark itself. In a live, running production grid where resources and not under the users control this is difficult to ensure. On the other hand, if one is to be consistent with the end-to-end representative measurements goal, this exclusive execution of benchmarks on otherwise idle machines is not desirable. Reserving a node for benchmarking, while it may give more reproducible results, may not capture the performance that is most likely to be experienced by users running in a real grid setting with quotas, restrictions different queue allocations etc. Capturing the state of the target node during benchmark execution, as explained in the previous chapter, could help explain performance results. In cases where the peak performance of resources is required the collected monitoring data can be used to filter results, e.g. when the CPU load right before and after the execution is not negligible.

4.2.3 Organizing and sharing metadata and results

Once the performance or test data is collected, it must be organized into a usable, easily searchable form. This form must be scalable as to accommodate results from benchmark invocations that could run in to the hundreds of thousands.

It is also desirable to be able to share results between users so as to minimize the number of benchmark executions. One way to accomplish this is by the implementation

of a central repository for results, which can be consulted before determining that a new benchmark execution is needed. It is important that these data are well described by metadata.

4.2.4 Benchmark customizations and extensions

A tool that is to be used for Grid testing and benchmarking needs to be flexible and extensible in order to cope with an infrastructure like the Grid. The main reason why this is important lies in the heterogeneity of the infrastructure. The tool needs to flexible and extensible enough to address different platforms, operating systems or architectures. A tool would need to address this in a reasonable manner by enabling the user to *easily* define and execute benchmarks for heterogeneous resources, while at the same time allowing the tuning of tests or benchmarks for different resources. Tuning of benchmarks can be manual, i.e. specifically defined by the user for a given set of resources, or automatic, where the tool infers benchmark or test parameters based on a given resource's attributes.

Another reason why flexibility and extensibility are so important is the rapidly evolving infrastructure, especially in terms of the middleware. From one perspective this is in fact adding to the heterogeneity of the system since not all resources upgrade or migrate to different middleware at the same time. Flexibility and extensibility go beyond the easy definition or tuning of benchmarks, it goes right to the core of the tool which should be designed with middleware-independence in mind.

4.2.5 Middleware Independence

There are already several core middleware that drive Grid infrastructures. A tool that is strictly tied to any one of them would not be applicable to many infrastructures that employ a different middleware. But, more subtly, a tool that is not middleware independent to some extent would sooner or later be obsolete due to the rapid evolution of Grid middleware.

4.2.6 Data analysis

Beyond the invocation of tests and the collection of results, a Grid benchmarking tool should allow for basic data analysis. This is in addition to the ability to export result in a usable form for further processing, for example in an advanced statistics software. The goal is, of course, not to re-implement the functionality of statistics software package, but to be able to simplify the cycle of "experiment – analyze – make decision".

The ability to query a large archive of results is essential, since the size of the data-set depends directly on the size of the Grid, the frequency of invocation and the amount of retained history. The ability to query result is a requisite for the key functionality of constructing intuitive charts in order to compare resource performance.

4.2.7 Putting collected information into use

The tool needs mechanisms for putting the collected information into use. This refers mainly to the ability to perform resource selection based on specific *performance* aspects but also on the detection of several problems pertaining to *availability* and *dependability*.

4.2.8 Functional and user-friendly interface

Generally a tool which targets end-users of the grid should be user-friendly and functional. The main areas of concern in this specific type of tool is the easy invocation of jobs on multiple sites in parallel; the ability to have a clear view of what is going on, i.e. job monitoring; and finally, the user-friendly data analysis via the easy construction of performance charts.

"GridBench" was designed and implemented based on these requirements. The following section details the GridBench design.

4.3 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.

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 *(iii)* the *SiteRank*, which implements the models used to rank Grid resources.

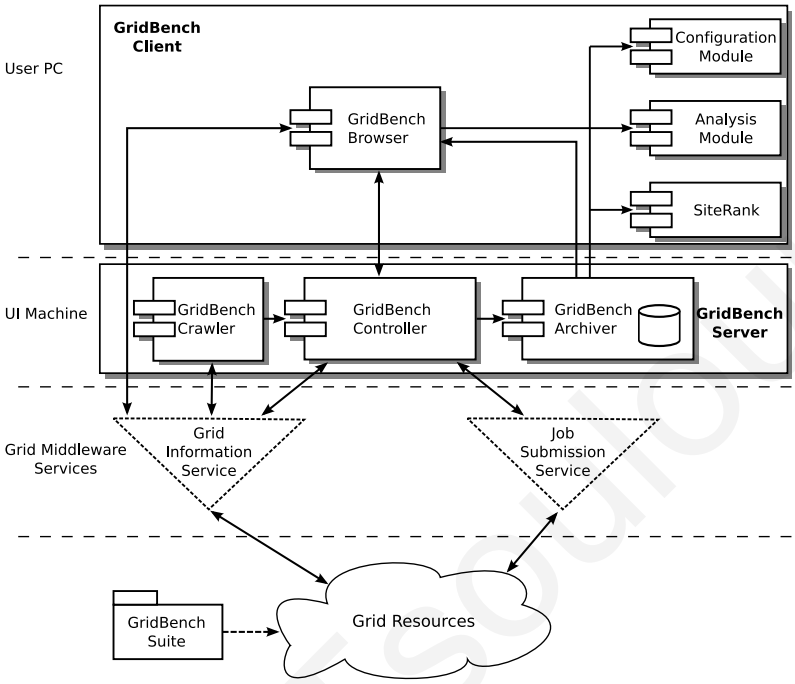


Figure 10: GridBench components.

GridBench is implemented in Java and employs 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 modular and makes extensive use of plug-ins to provide integration with various middleware.

4.3.1 Server-side components

4.3.1.1 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 Figure 11 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 in Figure 11):

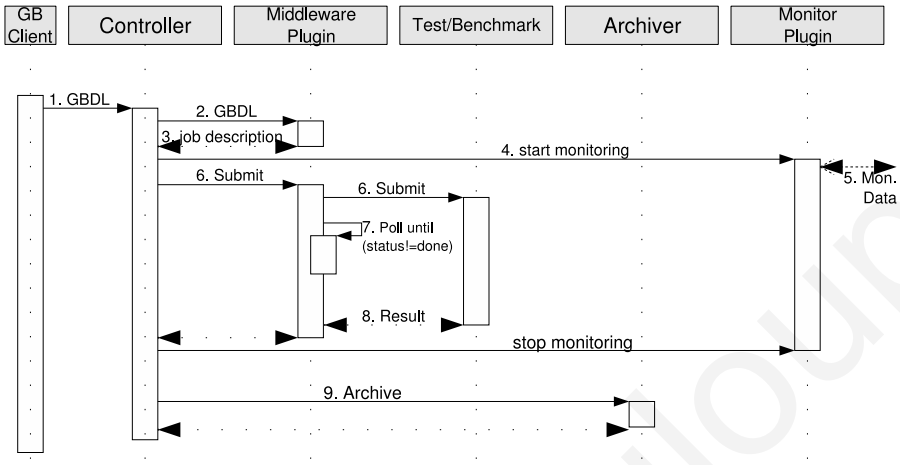


Figure 11: A UML sequence diagram describing the basic Controller functionality

(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, 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 and the result retrieved;

(8) The results of the benchmark in the form of *metric* elements and its associated monitoring data are incorporated into the 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.

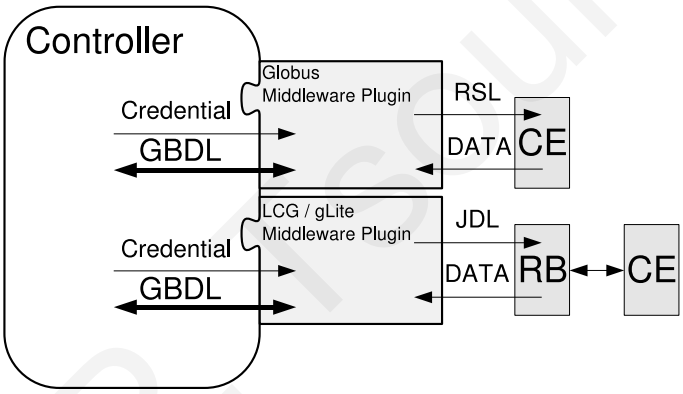


Figure 12: Middleware plugin functionality.

4.3.1.2 Middleware Plugins

Middleware Plugins (Figure 12) 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.

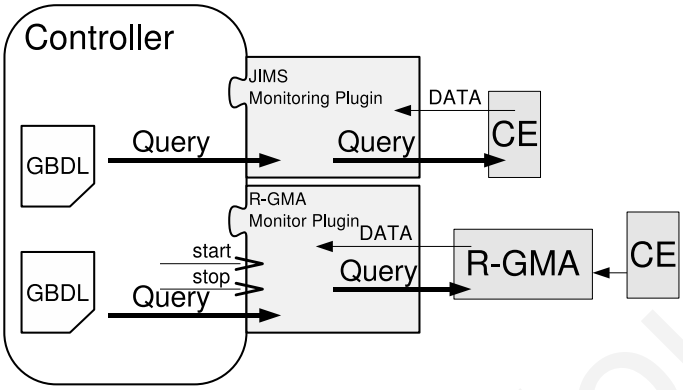


Figure 13: Monitor plugin functionality.

Based on the job submission mechanism selected when the *Controller* service is invoked, the GridBench Controller must interface with the underlying middleware mainly for submitting the job and getting the job status. Integration with diverse underlying Grid middleware is accomplished by the use of the Middleware Plugin Interface.

Interface Definition

String GetJobDescription(String gbdl)

Returns a middleware-specific job description (e.g. RSL for the Globus middelware) that reflects the provided GBDL benchmark description.

String execute(String gbdl,boolean async)

Execute the provided GBDL returning a middleware-specific job ID. If the boolean *async* is true then the method returns immediately after the job is *submitted*. If the value of *async* is false then the method returns upon completion of the benchmark.

String getJobStatus()

Returns a middleware-independent job status which can be one of:

waiting : The benchmark is waiting to be submitted to the underlying middleware;

submitted : The benchmark has been submitted using the specified mechanism;

running : The benchmark has started running;

error : The benchark is in a error state;

done : The benchmark has finished;

String getError()

Returns a middleware-dependent description of the error as reported by the underlying middleware or the empty string in case of no error.

The return string has the following format:

```
<job-status> <status from middleware>
```

As shown above, the middleware-specific status is also provided following the generic job status given by the Controller. The purpose of this is to provide a more descriptive job status and help debugging.

String getJobResult()

Returns the textual output of the benchmark (usually comprised of XML fragments).

4.3.1.3 Monitor Plugins

Monitor Plugins (Figure 13) 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.

4.3.1.4 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 mapping table fields to XML elements.

4.3.1.5 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.[2] This provides an initial body of results, which the user can complement with her own executions. This allows for more meaningful analysis of the resources' availability and dependability, 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.

4.3.2 Client-side components: 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. Figure 14 shows the main graphical user interface for the definition of benchmarks. Observe the list of available test/benchmarks (the list on the left) and the available resources (the list on the right). The resource list shows resources retrieved from

[2] Care should be taken to make sure that the results obtained through automated execution are representative of the user's VO.

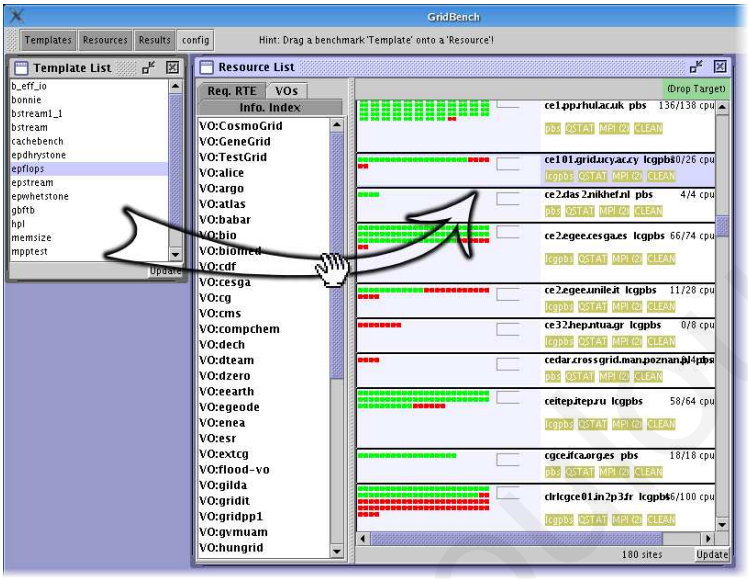


Figure 14: The list on the left is a list of tests/benchmarks that are integrated into GridBench. The list on the right shows the currently available resources and their status in terms of busy/free CPU's.

one or more Grid Information Systems, with details about each resource's composition, such as free/busy CPU's and Worker nodes, dual/single CPU machines etc. Additionally a set of tests can be performed on each resource. In Figure 14 one 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*.

4.3.2.1 CE Test Plugins

CE Test Plugins (Figure 15) encapsulate invocations of tests in order to *(i)* integrate them directly into the GridBench browser -shown in Figure 16(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.

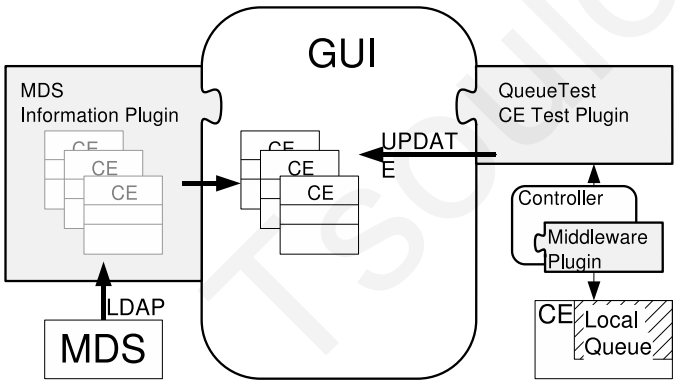


Figure 15: Information plugin and CE-test plugin functionality

4.3.2.2 Information Plugins

Information Plugins (Figure 15) are used by the GUI to retrieve information about Grid resources. The Information Plugin retrieves data from Grid Information Systems (GIS) 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 [4, 6], as well as additional information about individual Worker-Nodes (such as state and number of CPU's 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 CPU's on Worker-Nodes and local queue type).

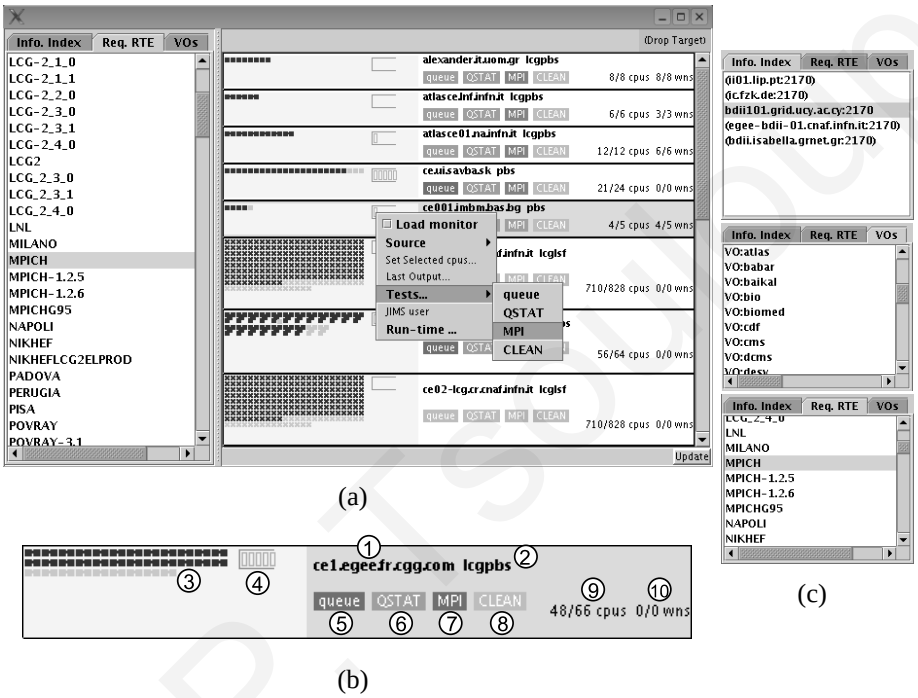


Figure 16: **(a)** Resource browser, showing the state of resources from the EGEE test-bed that "advertise" support for MPI (MPICH run-time environment; **(b)** The resource renderer; **(c)** *Top:* Information index sources selection. *Middle:* Querying for specific Virtual Organizations. *Bottom:* Specifying run-time environment support.

Information displayed in the **resource browser** (Figure 16(a)) is obtained by querying one or more information systems (Figure 16(c)-top). The information for each resource is displayed in graphical form and contains a rendering of the status of CPU's and the configuration of CPU's into worker-nodes. Figure 16(b) shows the resource renderer where *(1)* is the resource name; *(2)* is the local queue type, *(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 CPU's; and *(10)* shows Free/Total WN's (updated after the invocation of the "queue test" *CETest* 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 –whenever possible– 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 to limit her view based on VO (Figure 16(c)-middle), or support of a specific run-time environment (Figure 16(c)-bottom). Defining and executing a test or benchmark is as easy as dragging a benchmark onto one of the resources (shown in Figure 14).

4.3.2.3 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 CPU's 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 (Figure 17). 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.

4.3.2.4 The Analysis Module:

The Analysis Module has the main functionality of presenting results, and allowing the user to query the underlying relational database that holds the contectualized results.

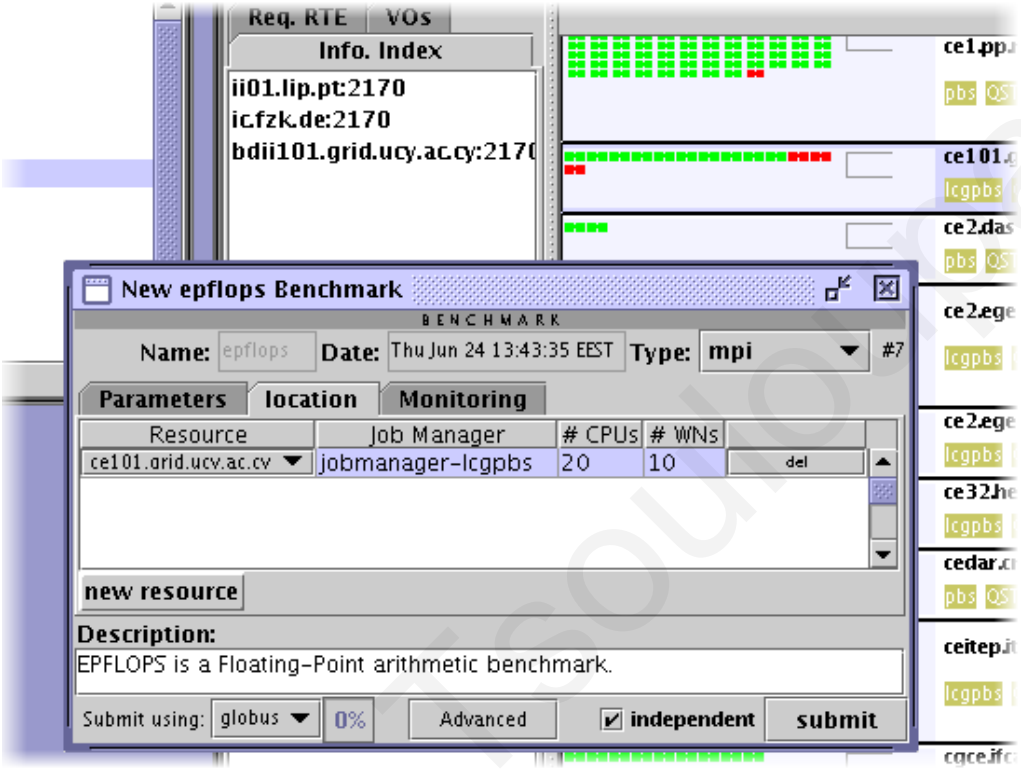


Figure 17: Benchmark configuration panel.

A use query is in the form of a standard SQL Query. For example:

```
SELECT benchmark.id from benchmark, resource, parameter
 WHERE resource.bid=benchmark.id
   AND parameter.bid=benchmark.id
   AND benchmark.name='epwhetstone'
   AND parameter.name='nloops'
   AND parameter.value>10000000};
```

This query will provide all the available results for all resources where “epwhetstone” has been executed with an “nlopps” paremeter greater than 10000000.

The user can easily construct graphs as the ones in the results section by browsing results (metrics) previously archived in the database (Figure 18,19). The interface enables the user to perform custom queries on the archived results and to select the results that

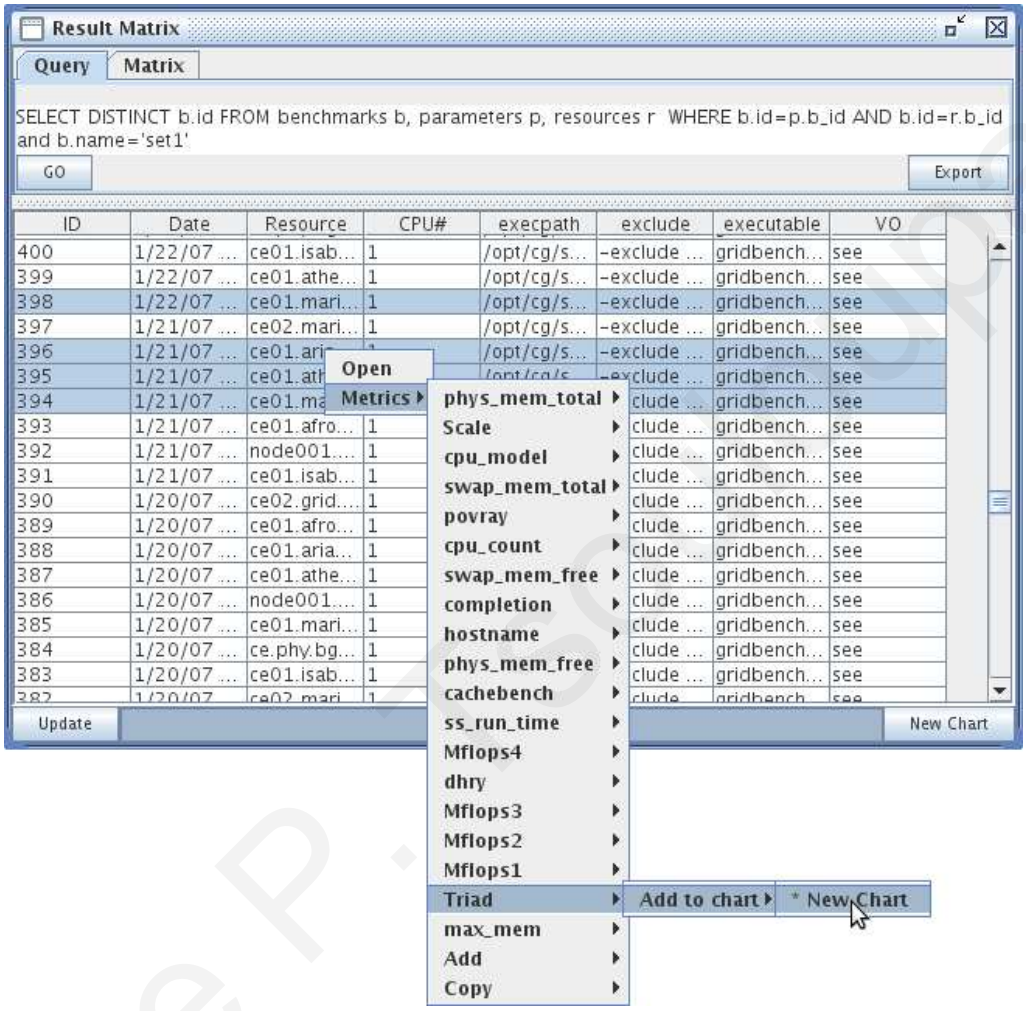


Figure 18: The interface for querying the results and selecting which results to render.

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.

4.3.3 Benchmarks

The GridBench suite offers three categories of benchmarks: *micro-benchmarks*, *micro-kernels* and *application kernels*. The purpose of having these three categories of benchmarks is essentially the introduction of different points of view of Grid performance.

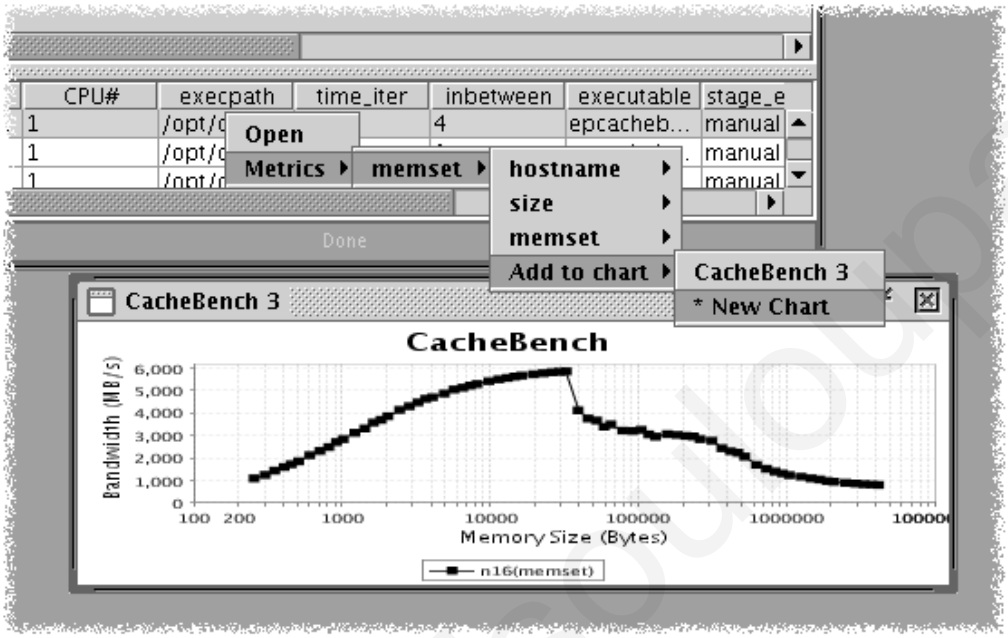


Figure 19: Generation of charts from historical data. The result shown is from a memory cache benchmark.

Micro-benchmarks aim to measure a single performance factor of a system . They are good for identifying the basic performance properties of each of the four levels mentioned previously, i.e. grid constellations, sites, resources within a site (e.g. individual worker nodes) and Grid services. What constitutes “basic performance properties” is somewhat subjective and depends on the level of the architecture one is trying to measure. In general, these basic performance measurements are low-level measurements; for example, a file transfer between two sites when measuring a Grid constellation, or the achievable floating point operations per second (in a tight loop) when measuring a CPU. Each basic performance property is measured by “stress testing” of a simple operation invoked in isolation with the least possible dependence on other performance aspects.

Micro-kernel benchmarks are benchmarks that usually employ a computational kernel which is usually a synthetic kernel. In this context, a *synthetic* kernel is defined as

one that has been specifically designed to measure performance, versus one that is derived from a real application. Micro-kernel benchmarks are therefore generic High Performance Computing /High Throughput Computing kernels which include general and often-used kernels in Grid/Cluster environments. Their aim is to measure several performance aspects of a system collectively. **Application** or **application-kernel benchmarks** are derived from real applications, e.g. CrossGrid applications, and will resemble the originating application. They aim to measure the characteristics of "representative" applications by capturing high-level metrics such as completion time, throughput and speedup.

4.3.3.1 Micro-benchmarks

Micro-benchmarks generally produce low-level metrics. There are numerous distinct low-level aspects of a resource that can be measured but 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, the performance of its interconnects and I/O performance. Of course there is a wealth of other factors affecting machine performance ranging from Operating System robustness to fitness for running a specific application. With respect to micro-benchmarks, it was a deliberate choice to initially focus on this set of low-level metrics, keeping in mind the easy inclusion of more metrics as deemed necessary. In terms of specific low-level metrics the following were chosen: *(i) Floating Point* or *Integer Operations Per Second* for CPU performance, *(ii) Bytes per second* for writing and reading to and from main memory, *(iii) Bandwidth* for evaluating the machine's interconnects and *(iv) I/O bandwidth* for evaluating disk performance (shared or

Table 2: Metrics and Benchmarks

Factor	Metric	Delivered By
CPU	Operations per second (mixture of floating point and integer arithmetic)	EPWhetstone
CPU FP	Floating Point Operations per second	EPFlops
CPU INT	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
Interconnect	latency and bisection bandwidth	MPPTest
I/O	disk latency and bandwidth	b_eff_io

local storage). These metrics are easily understood and well-established for evaluating their respective performance factor.

In order to deliver the required metrics, six benchmarks are employed:

(i) EPWhetstone, (ii) EPFlops, (iii) EPDhrystone, (iv) EPStream, (v) CacheBench, (vi) MPPTest, (vii) b_eff_io.

EPWhestone

EPWhestone is a simple adaptation of the traditional Whetstone CPU benchmark [21] so that it runs in parallel on a set of CPU's at the same time. It is implemented in C, and uses MPI for collecting the final measurements from each process(communication time is excluded from measurements). Each process runs independently of the others with no communication taking place until the very end, when each process reports the final result. Each process performs a mixture of operations, such as integer arithmetic, floating point arithmetic, function calls, trigonometric and other functions. The benchmark gets

the current time using *gettimeofday()*, runs for a few seconds,calculates the wall-clock time difference and reports the rate at which these operations were performed on average. The "EP" prefix denotes the "embarrassingly parallel"nature of its execution. The sum of the results of all the processes (operations per second) is then reported as the CPU performance of the whole resource. Just as with the other benchmarks, it is imperative that for the duration of the execution, the only process imposing substantial load on the CPU is the EPWhetstone process, especially since the results are calculated using wall-clock time. The typical execution time is less than10 seconds.

EPFlops

EPFlops is a floating-point CPU benchmark adapted from the "flops" benchmark [1]. 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. 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. Table 3 gives a summary of the distribution of floating-point operations in the four metrics delivered by the "EPFlops" benchmark. For example, the *mflops-2* metric does 152 operations per loop. Out of the 152 operations, 58 (38.2%) are additions, 14 (9.2%) are subtractions, 66 (43.4%) are multiplications, and 14 (9.2%) are divisions.

Table 3: Metrics returned by the “EPFlops” benchmark and the operation counts in each reported metric.

Metric name	FADD	FSUB	FMUL	FDIV	Total
mflops-1	21 (40.4%)	12 (23.1%)	14 (26.9%)	5 (9.6%)	52
mflops-2	58 (38.2%)	14 (9.2%)	66 (43.4%)	14 (9.2%)	152
mflops-3	62 (42.9%)	5 (3.4%)	74 (50.7%)	5 (3.4%)	146
mflops-4	39 (42.9%)	2 (2.2%)	50 (54.9%)	0 (0.0%)	91

EPDhrystone

EPDhrystone is an integer operations benchmark, adapted from the C version of the “dhrystone” benchmark [73]. 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. It focuses on “systems programming” applications which perform mainly integer operations. 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.

EPMemsize

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_free_memory()" or 'get_total_memory()" system call is many times not the real amount of physical memory that can be allocated by an application; the system kernel, services, other processes, as well as filesystem caches also hold up memory. 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

EPStream is a simple adaptation of the C implementation of the well-known STREAM memory benchmark [46] so that it runs in parallel on a set of CPU's at the same time.Again, each process runs independently of the others with no communication taking place until the very end, when each process reports the final result 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*. Table 4 outlines each operation.

Table 4: The STREAM benchmark Operations

Name	Operation
copy	a[i]=b[i]
scale	a[i]=q*b[i]
sum	a[i]=b[i]+c[i]
triad	a[i]=b[i]+q*c[i]

It is imperative that for the duration of the execution, the only process imposing substantial load on the CPU or Main memory is the EPStream process (results are calculated using wall-clock time). The typical execution time is less than 10 seconds.

CacheBench

CacheBench [49] is a benchmark aiming at evaluating the performance of the local memory hierarchy of a machine. Similarly to the previously described benchmarks,an instance of CacheBench is invoked on each CPU of the resource (i.e. cluster) under study. 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. The operations at each size run for a configurable amount of time(default is 2 seconds) and the average bandwidth (MB/s) is reported. Table4.3.3.1 outlines each operation. While this benchmark produces a similar metric to the STREAM benchmark, it runs for a longer time since it takes measurements at different memory sizes (executions are typically in the order of 5 minutes). The time it takes to finish depends strictly on the input parameters. It also focuses on the performance of memory *caches* providing insight to the different levels of cache available to the CPU's.

Table 5: The CacheBench Operations

Name	Operation
read	register=m[i]
write	m[i]=register++
read/write	m[i]=m[i]++
memset()	(system call)
memcopy()	(system call)

Since this benchmark runs considerably longer than the EPStream benchmark, it was initially infered that it would make sense to invoke it when need arises (i.e. the user is explicitly interested in cache performance, and the sustained memory bandwidth produced by EPStream is not adequate). As experimentation progressed, it was discovered that the cache had substantially more effect than the *EPStream* benchmark could capture, and a new metric (c512k, analyzed in the next chapter) needed to be introduced. As with the other benchmarks, it is imperative that for the duration of the execution, the only process imposing substantial load on the CPU or Main memory is the CacheBench process (results are calculated using wall-clock time).

MPPTest

MPPTest [39] 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 by the MPPTest creators to be reproducible since the reported measurements

are the minimum of several runs. For the purpose of resource characterization it is desirable to have a focused set of measurements and to this end, only two measurements are performed:*(i)* bisection bandwidth and *(ii)* the "scatter" (broadcast) collective operation. The typical execution time is in the order of minutes (depending on the measurement detail)and results are calculated using wall-clock time.

b_eff_io

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" [53]. *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 [53]). Given that shared disk I/O is usually performed over the network, the results obtained by this benchmark may be correlated with the results obtained by the MPPTest benchmark. The benchmark is implemented in C.

4.3.3.2 Micro-kernel benchmarks

High Performance Linpack

HPL [26] is one of the most widely known benchmarks in HPC; HPL now ranks the TOP500 [25] 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.

4.3.3.3 Application benchmarks

Bstream

Bstream is a blood-flow simulation code aimed as a pre-operative support decision system for vascular surgeons. The blood-flow simulation is part of an interactive Grid application that involves processing of 3D data obtained from MRI scanners, operation planning (such as a bypass), simulation, and finally blood-flow visualization [60]. Shown here is the computationally intensive part of the application, which is based on a lattice Boltzmann solver and uses MPI. This code was instrumented to measure elapsed time for each iteration, and integrated into GridBench.

Figure 20: POVRay rendering of the `benchmark.pov` benchmark scene.

POVRay

A photorealistic rendering (ray-traycing application) “Persistence of Vision”. Version 3.6 of the application is used and applied to a scene file, `banchmark.pov` (Figure 20), composed by the POVRay developers and is explicitly intended for benchmarking.

4.4 Performance Evaluation and Auditing Using GridBench

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. This section, describes a number of different use-cases that were demonstrated with GridBench.

4.4.1 Application Performance

In the first use-case, 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, kernels extracted from three real scientific applications deployed on CrossGrid [38] were used:

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 [31].

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. The parallel (very tightly-coupled) version of the code [47] was used.

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 [60].

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>
```

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 (Figures 21, 22 and 23).

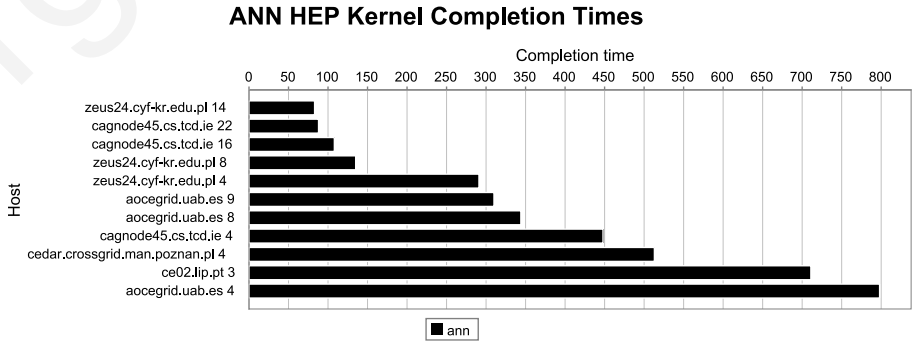


Figure 21: Results for the parallel Artificial Neural Network training application kernel. The number of CPU's is indicated next to the resource name and the completion times are sorted (best-performing first).

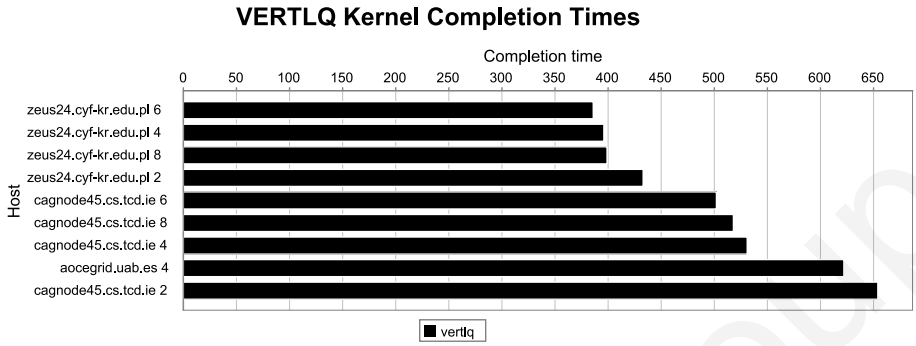


Figure 22: Results for the VERTLQ kernel from the air pollution simulation application. The number of CPU's is indicated next to the resource name and the completion times are sorted (best-performing first).

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. Figure 21 shows results from the ANN kernel while Figure 22 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 CPU's 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 CPU's or 8 CPU's from site A?*", or "*Should I use 4 CPU's from site A or 6 CPU's from site B?*".

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 fine level (see next sub-section). In the case of the "bstream" kernel, the application performs iterations which are controlled by a main loop. In total, about ten lines of code were

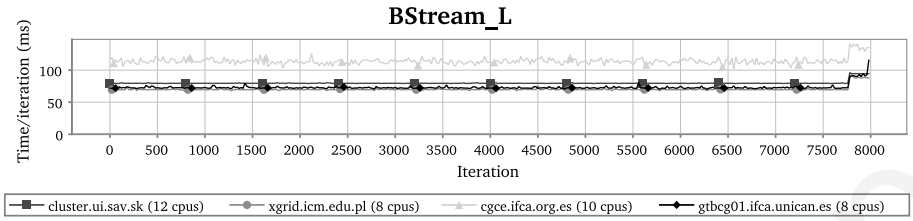


Figure 23: Results for the "bstream" kernel, showing iteration times on a set of four resources.

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>
```

4.4.2 User-driven Resource Ranking

The second use-case considers two users, each with a different application and different requirements in terms of performance. They require a performance-ranked set of resources[3] 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 as in Figure 24.

Obtaining the right coefficients to determine the weight of each metric in an application-specific ranking is described in detail in [71]. For the purpose of this scenario it is assumed

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

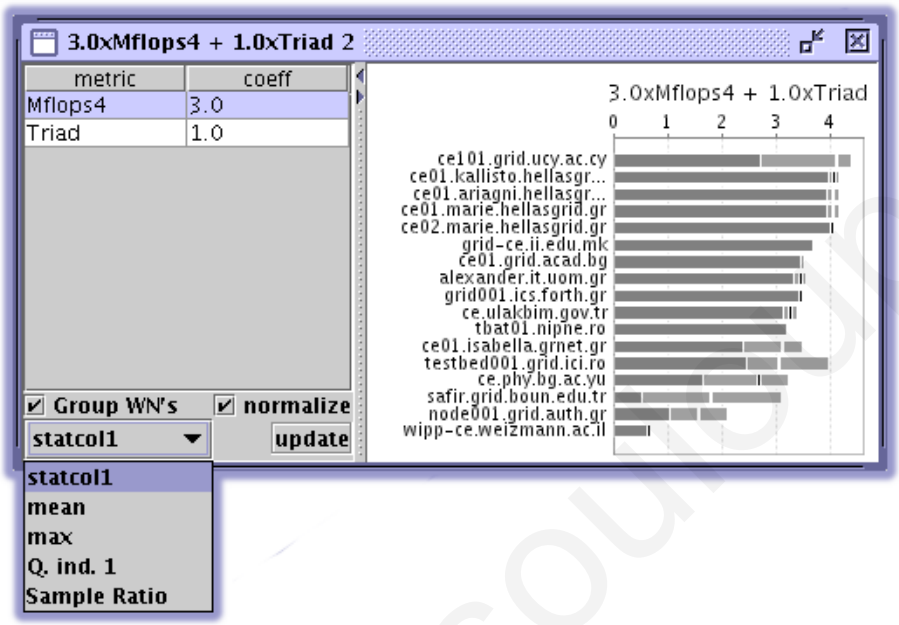


Figure 24: Snapshot of the ranking module.

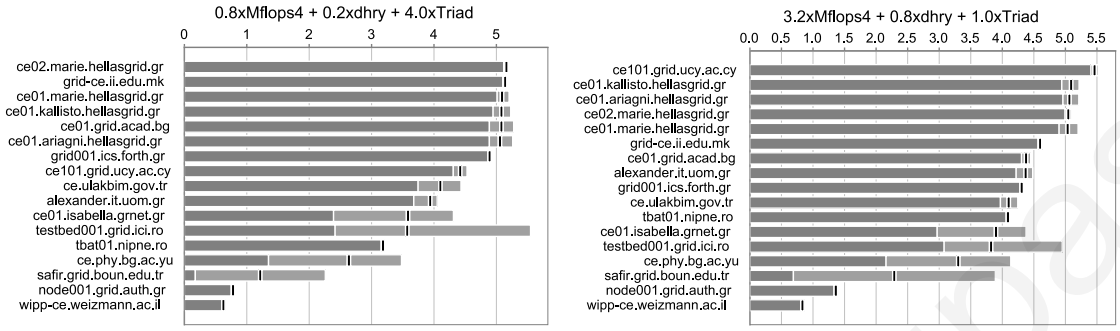
that the user has some insight as to how the different low-level metrics relate to the performance of the application at hand.

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

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

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

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



(a) (b)

Figure 25: Ranking of SE Europe resources by putting more emphasis on CPU or main-memory performance.

Figure 25(a) is the result of ranking function R_A, while Figure 25(b) is the result of the modified ranking function R_B. The resulting composite performance metric varies considerably; a resource that ranks 8^{th} in R_A, ranks 1^{st} in R_B.

Admitedly, the coefficients used here are rather arbitrary. The goal was to illustrate their use and how thay can affect the ranking. An appropriate way of deducing the coefficients is presented in the next chapter.

Chapter 5

Performance Ranking of Grid Resources

Matching between resource requests and resource offerings is one of the key considerations in Grid computing infrastructures. Currently, the implementation of matching is based on the *matchmaking* approach introduced by the Condor project [55], adapted to multi-domain environments and Globus, and extended to cover aspects such as data access and work-flow computations, interactive Grid computing, and multi-platform interoperability. Matchmaking produces a ranked list of resources that are compatible to the submitted resource requests. Ranking decisions are based on published information regarding the number of CPU's of each resource, their nominal speed, the nominal size of main memory, the number of free CPU's, available bandwidth, etc. This information is retrieved from Grid information services such as the Monitoring and Discovery Service of Globus, or the BDII of EGEE.

This approach works well in cases where the main consideration of end-users is to allocate sufficient numbers of idle CPU's in order to achieve a high job-submission throughput with opportunistic scheduling. In several scenarios, however, reliance on matchmaking is not sufficient; for instance, when end-users wish to "shop around" for Grid computing resources before deciding where to deploy a high-performance computing application, or when Virtual Organization (VO) operators want to audit the real availability and configuration status of their providers' computing resources. In such cases, the information published by resource providers and Grid monitoring systems does not provide sufficient detail and accuracy. The need for control over which resources are chosen to run a VO's applications is very real. For example, EGEE developed a system [18] for *whitelisting* and *blacklisting*, in other words including and excluding resources. The tool involved the insertion of a "proxy" Information Index where certain "bad" or very low performance resources are blacklisted, or some known "good" resources are *whitelisted*. The tool was in the right direction, but lacked the important functionality of actually helping the user determine which were the good or bad resources (it was, in fact, geared towards Grid administrators). Grid users need instead the capability to interactively administer benchmarks and tests, retrieve and analyze performance metrics, and select resources of choice according to their application requirements. GridBench was designed to provide Grid users with such a *test-driving* functionality. As described in earlier chapters it is essentially a framework for evaluating the performance of Grid resources interactively. GridBench facilitates the definition of parameterized execution of various probes on the Grid, while at the same time allowing for archival, retrieval, and analysis of results [68, 69, 72].

GridBench comes with a suite of open-source micro-benchmarks and application kernels, which were chosen to test key aspects of computer performance, either in isolation or collectively (CPU, memory hierarchy, network, etc.) [70].

This chapter provides a more in-depth view of application performance, and presents *SiteRank*, a component developed on top of GridBench to support the user-driven ranking of computational Grid resources. SiteRank enables Grid users to easily construct and adapt ranking functions that:

1. Take as arguments performance metrics derived with the low-level benchmarks of GridBench [70]; the selection of these metrics can be done manually or semi-automatically by the end-user, through the user interface of GridBench.

2. Combine the selected metrics into a linear model that takes into account the particular requirements of the application that the user wishes to execute on the Grid (e.g., memory vs. floating-point performance bound). Using a ranking function, Grid users can derive rankings of Grid resources that are tailored to their specific application requirements.

Furthermore, I demonstrate the use of SiteRank in the ranking of the computational resources of EGEE, which is the largest production-quality Grid in operation today. To this end, I examine two alternative applications running on EGEE: povray, a ray-tracing application, and SimpleScalar, a simulator used for hardware-software co-verification and micro-architectural modeling. The results show that SiteRank functions can provide an accurate ranking of EGEE resources, in accordance to the different requirements that each application has. Furthermore, it is evident that the careful selection of the low-level

metrics used in the linear model is important for the construction of accurate ranking functions.

Section 5.1 outlines one approach in understanding application performance and ranking resources, while Section 5.2 introduces SiteRank and its ranking methodology. Section 5.3 describes the use of SiteRank in the ranking of EGEE resources for the two selected applications: povray and SimpleScalar.

5.1 Application Performance: A more in-depth look

Resource selection in Grid environments is a crucial problem. Regardless of who performs the resource selection, be it users or automated systems (i.e. schedulers or resource brokers), the decision makers are faced with the difficult task of matching/mapping jobs to resources. Previous work on the specification of resources and services in complex heterogeneous computing systems and metacomputing environments in general [14] and, particularly, in grid environments [16], has led to a better understanding of the issues. Nevertheless, in many cases there is a need for addressing application-specific characterization of the resources available. Employing an application introduced in the previous sections, the pre-operative support/ blood-flow simulation solver, it can be shown that GridBench can be used to *easily* obtain measurements that can be used to select the right resources on which to schedule instances of the application. In the following sub-section I present the results of performing a number of experiments to investigate the levels of performance offered by hardware resources distributed across a subset of a computational Grid. The application, and support on correctly instrumenting it were kindly provided by

a team of fellow researchers at the University of Amsterdam. These experiments show how one can rank resources based on a benchmark derived from the blood-flow simulation kernel. The work presented here is in collaboration with researchers at the University of Amsterdam [66].

5.1.1 The Application

These benchmarking experiments employ a parallel solver from the Virtual Radiology Explorer (VRE) Grid-based Problem Solving Environment (PSE), a type of integrated collaborative environment [41] that includes simulation, interaction, and visualization components for pre-treatment planning in vascular interventional and surgical procedures. This PSE was developed by the University of Amsterdam and deployed within the European Crossgrid project [65]. The VRE contains an efficient parallel computational hemodynamics solver [7] that computes pressure, velocities, and shear stresses during a full systolic cycle. The simulator is based on the Lattice-Boltzmann method (LBM), a mesoscopic approach for simulating fluid flow based on the kinetic Boltzmann equation [67]. The data used as input for the VRE can be obtained from several imaging techniques used to detect vascular disorders. For instance, 3D data acquired by Computed Tomography or Magnetic Resonance Imaging, or particularly Magnetic Resonance Angiography for imaging blood vessels that contain flowing blood. To convert the medical scans into meshes the solver can work with, the raw medical data is first segmented so that only the arterial structures of interest remain in the data set (Figure 26[1]).

[1]Image courtesy of the University of Amsterdam.

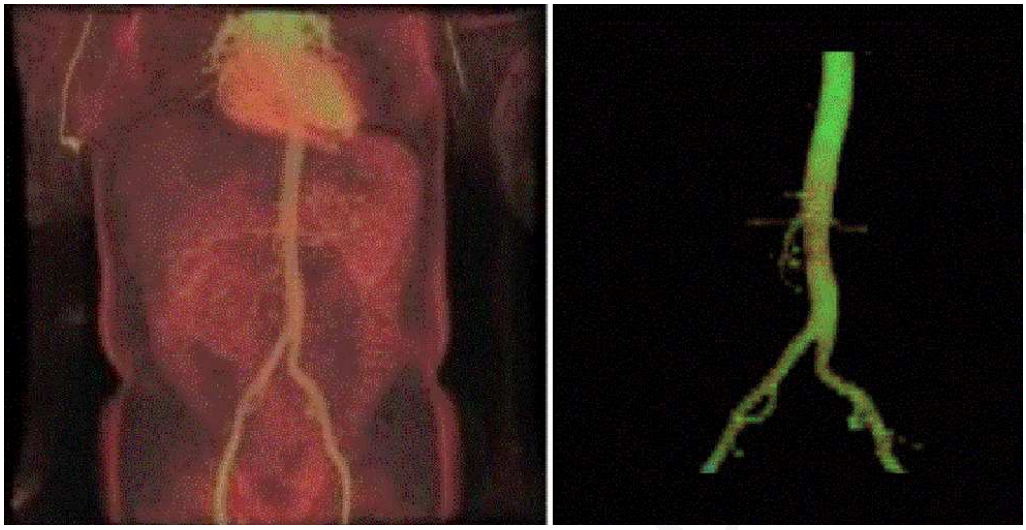


Figure 26: Segmented medical data from the abdominal aorta, accessible via Grid Storage Elements functioning as medical repositories.

Measurements are important for diagnoses. Clinical decision-making relies on evaluation of the vessels in terms of the degree of narrowing for stenosis and dilatation (increase over normal arterial diameter) for aneurysm. The selection of a bypass (its shape, length, and diameter) depends on sizes and geometry of an artery.

5.1.2 Using GridBench

As described in the first section of this chapter, the main goal of GridBench is to generate metrics that characterize the performance capacity of resources belonging to a Virtual Organization (VO). In addition one can see how GridBench can be a used as 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. In order to perform benchmarking measurements in an organized and flexible way, GridBench provides a means for running benchmarks on Grid environments as well as

collecting, archiving, and publishing the results. The framework allows for convenient integration of new applications benchmarks into the suite, as well as the customization of existing benchmarks through parameters.

5.1.3 Results

The application benchmark, the BStream kernel, is part of an interactive Grid application that involves processing of 3D data, which makes it computationally expensive. Shown here is the computationally intensive part of the application, which uses the Message Passing Interface (MPI) for parallelization. This code was instrumented to measure elapsed time for each iteration as well as the time spent on MPI communication, and integrated into GridBench[2] . Dataset files from simple tube-like artery structures to aorta segments containing bifurcartions. They were used to represent typical workload.

5.1.3.1 Resource Comparison

Figure 27(a) shows the measured iteration times of the BStream kernel on 13 sites available in the testbed. In each case, the same workload was applied by using identical input data and parameters. Figure 27(a) shows the results obtained by using 2 CPU's in each measurement. For the 2 CPU measurements, using 2 CPU's on the same Worker Node was preferred over using two CPU's on two different Worker-Nodes. This is important, since it was found that this would seriously impact performance of this kernel (Figure 29). Resources *cluster.ui.sav.sk* and *loki01.ific.uv.es* employ single-CPU nodes

[2]The charts presented in this section were automatically generated using the GridBench Graphic User Interface.

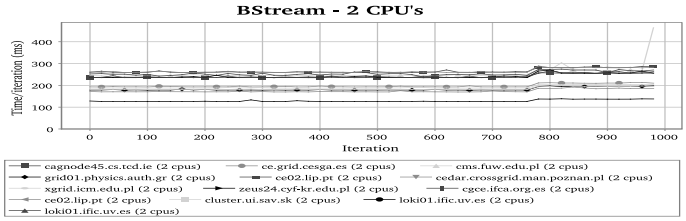
while the majority of site employ dual-CPU Worker Nodes. In Figure 27(a) (as well as the rest of the charts in Figure 27) one observes that iteration times remain fairly constant throughout the duration of the computation. For the experiments, the application kernel was set to run for 800 iterations, so it can be seen that right before the end of each run (at around 760 to 780 iterations) a jump in performance of about 30% larger time per iteration values is experienced in all nodes. This is mainly due to the design of the current version of the kernel, where the first processor that started running gathers data from all other processors before producing the final output[3] . Nevertheless, iteration times remain relatively invariant regardless of the number of iterations. For this reason it is reasonable to assume that short run-time experiments (using a small number of iterations) are representative of real-life experiments, in which larger iteration counts are used.

Figures 27(b), 27(c) and 27(d) show the performance of the kernel at a set of sites using 4, 8 and 12 CPU's respectively. Generally one observes a "downward" trend indicating that the code is somewhat scalable, i.e. using a larger number of CPU's at a given site will yield a faster run-time, but more on scalability will be given in the next sub-section.

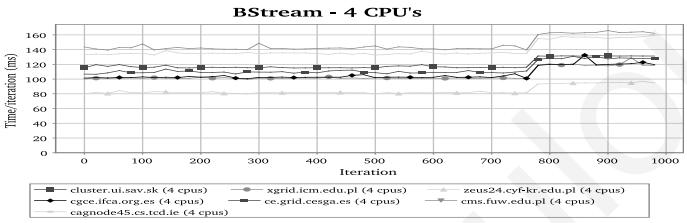
5.1.3.2 Scalability

Figure 28 shows the scalability of the kernel as it is measured at four sites: *cgce.ifca.org.es* (up to 12 CPU's), *cluster.ui.sav.sk* (up to 12 CPU's), *xgrid.icm.edu.pl* (up to 8 CPU's) and *zeus24.cyf-kr.edu.pl* (up to 8 CPU's). It is quite interesting to observe that the different

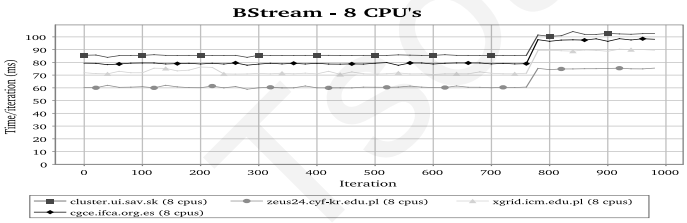
[3]The new version of the BStream kernel, which is work in progress at the time of writing, has a different design that addresses this issue.



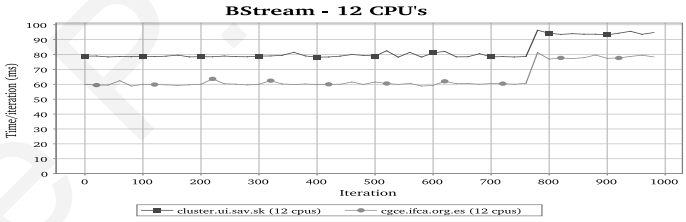
(a)



(b)



(c)



(d)

Figure 27: The performance of the kernel at a set of sites using 2, 4, 8 and 12 CPU's

sites display a different scalability. For example, in Figure 28(a) the runtime is reduced to less than 30% when going from 2 CPU's to 8 CPU's, while in Figure 28(b) the improvement is only just under 50%. Similarly, while in 28(a) there is approximately a 25% improvement in runtime when going from 8 CPU's to 12 CPU's, in 28(b) there is only marginal improvement. Scalability *at each resource* needs to be taken into consideration for efficient resource selection.

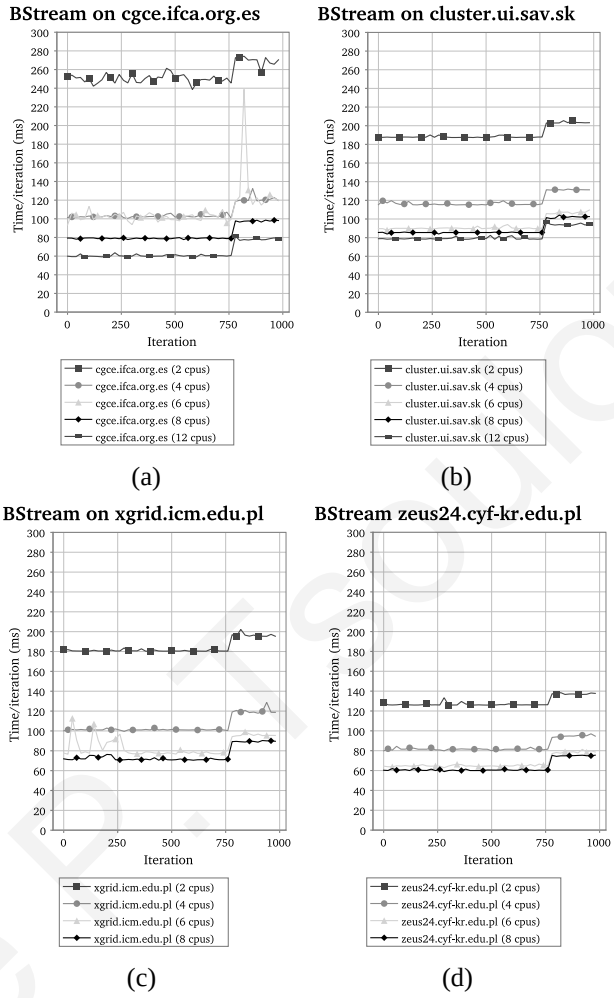


Figure 28: Scalability as it is measured at four sites. Lower iteration times are better.

5.1.3.3 Communication measurements

The BStream kernel uses MPI for inter-process communication, which were compiled using the MPICH4 device. The code is highly coupled and it is expected that the performance of the interconnect, i.e., the LAN connecting the cluster nodes will have a considerable impact on the performance of the kernel. To investigate this, the BStream code was instrumented to measure the time spent in communication[4] . To isolate the

[4]The impact of the instrumentation was measured and was found to be insignificant.

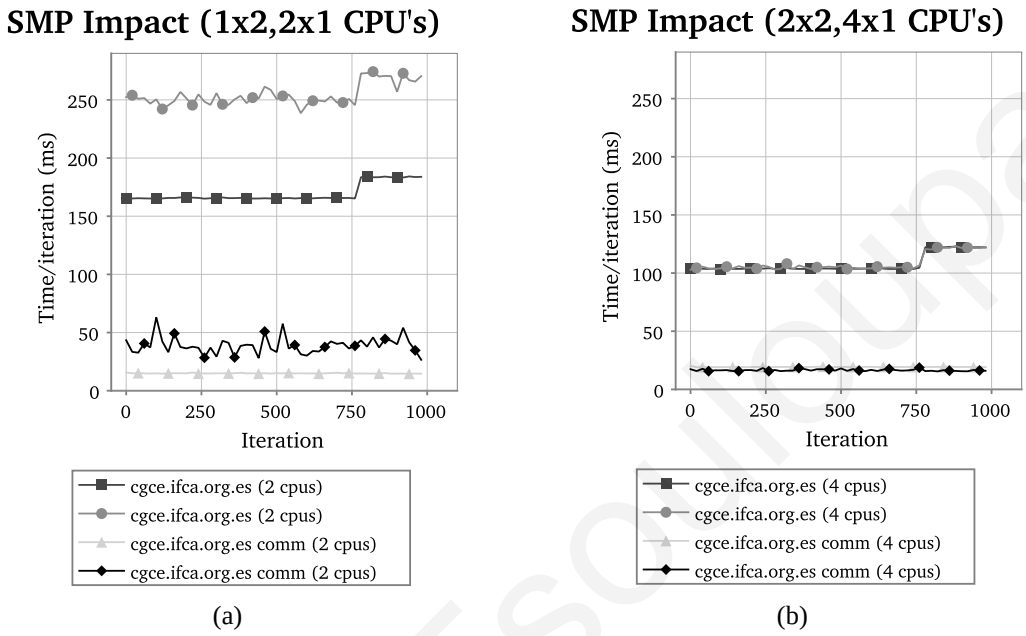


Figure 29: Impact of MPI communication on runtime. 29(a) Iteration and communication times using 2 CPU's on the same (dual) Worker Node (1x2), and 1 CPU on each of 2 Worker Nodes. 29(b) Iteration and communication times using 2 CPU's on each of 2 (dual) Worker Nodes (2x2), and 1 CPU on each of 4 Worker Nodes (4x1).

effect of the network, the code was executed using just two CPU's on a dual-CPU Worker Node (1x2)[5] , using two CPU's on two different (identical) Worker Nodes (2x1). This is shown in Figure 29(a). Figure 29(b) shows a similar experiment using 4 CPU's. In 29(a) one observes that there is considerable difference in communication performance which also impacts the time per iteration. On the other hand, in 29(b) it can be observed that there is no significant difference when running in either mode, since the network is used is both cases (both in 2x2 and in 4x1).

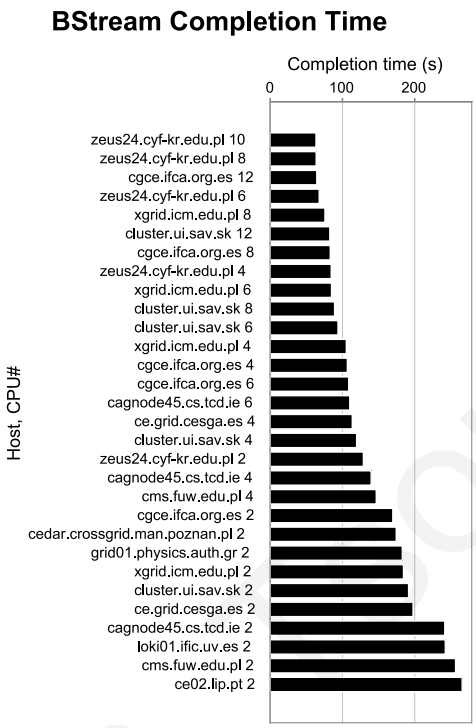


Figure 30: Completion times of the BStream kernel using different numbers of CPU's on several resources.

5.1.3.4 Decision-making

Figure 30 conveys a lot of useful information since it provides a ranking of run-times on all of the resources available at the time. This ranking could be used directly in resource selection *especially* in cases where the relative CPU, Memory and network speeds at each resource (site) are not known. For example, it appears that it is better to run the code at *zeus24.cyf-kr.edu.pl* using 4 CPU's than at *cluster.ui.sav.sk* using 8 CPU's.

Through this set of results it can be deduced that ranking resources based on the performance of a stripped-down and instrumented version of an application can give us

[5]The MPI library used was not optimized for SMP, and communication still went through the TCP/IP stack.

realistic resource rankings that reflect the performance of the application itself. Yet, while this is a viable solution it is still an expensive one since the performance measurement is application specific and the performance charecteristics many times prove quite different between applications.

5.2 SiteRank

Computational resources on the Grid *exhibit considerable variance in terms of different performance characteristics*. This leads to non-uniform application performance that significantly varies between applications. A quick survey of the distribution of performance of grid resources yields the results in Fig. 31. It shows three histograms, each showing the performance distribution of the same set of resources under different workloads , i.e. when different performance criteria are taken into account. For each workload, the performance of 159 computational resources was measured and categorized into 22 bins; the frequencies were plotted in the histograms.

The three histograms are clearly dissimilar, even in the case of the microbenchmarks *c512k* and *mflops-4*. Results from *mflops-4* resemble a Normal distribution whereas results from the *povray* rendering application and the *c512k* cache benchmark are skewed in opposite directions.

The useful information that comes out of this is that there is a need for resource performance to be characterized by several criteria. Grid resources *do* exhibit distinct performance characteristics, and the performance of a resource is *not* always directly

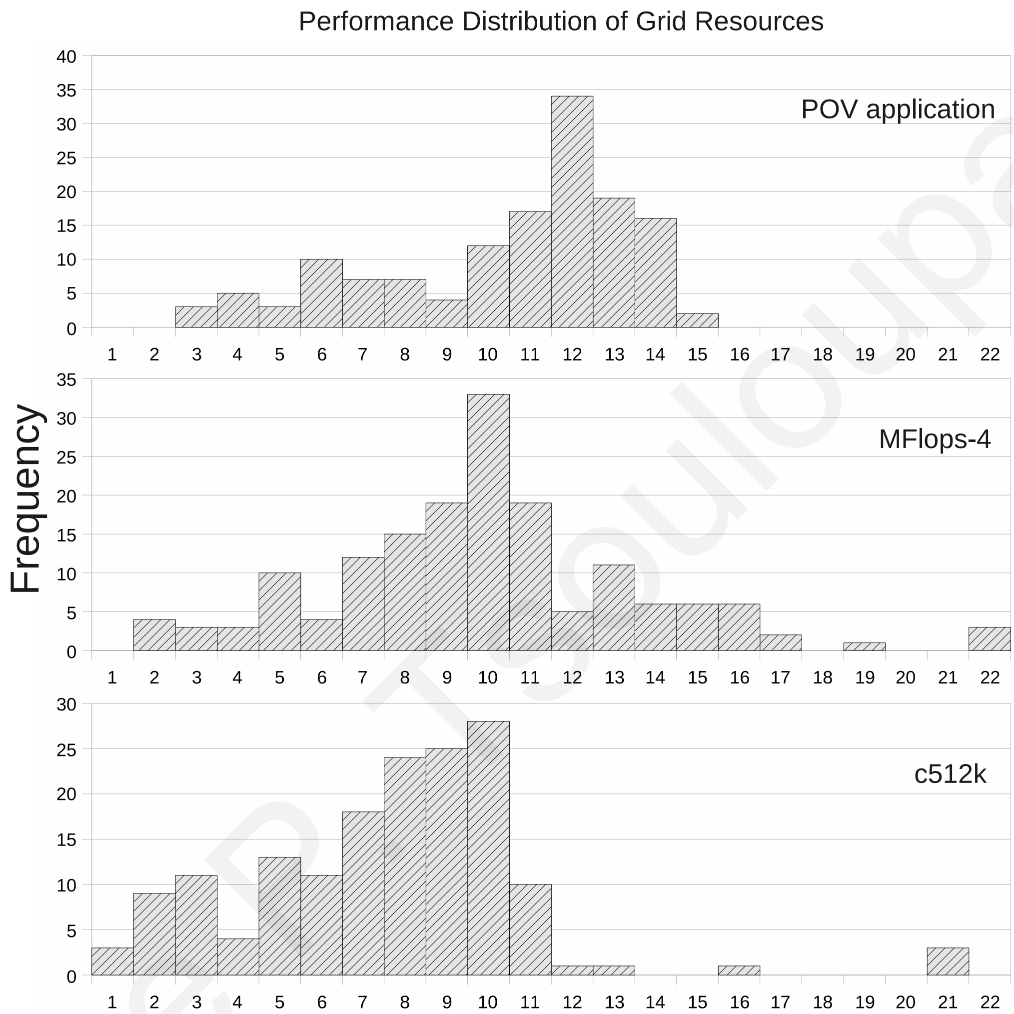


Figure 31: Performance distribution of resources by different performance criteria.

proportional to another; the workload by which performance is measured will affect the resource's relative performance.

5.2.1 Auditing and the deficiencies in current approaches

One approach for ranking resources in terms of performance is the one taken by the current (EGEE) infrastructure, which is to publish GlueHostBenchmarkSF00 (SPEC-Float 2000 floating point performance metric) and GlueHostBenchmarkSI00 (SPEC-Int

2000 integer performance metric) values for each site. These values are supplied by the site administrators by manually entering these values in a configuration file. While administrators are advised to include this data, they are not forced to do it. It is indicative that at the time of data collection, over 21% of the resources were not publishing any Spec-Int value and over 45% did not publish any Spec-FP value (usually quoted as "0") at all.

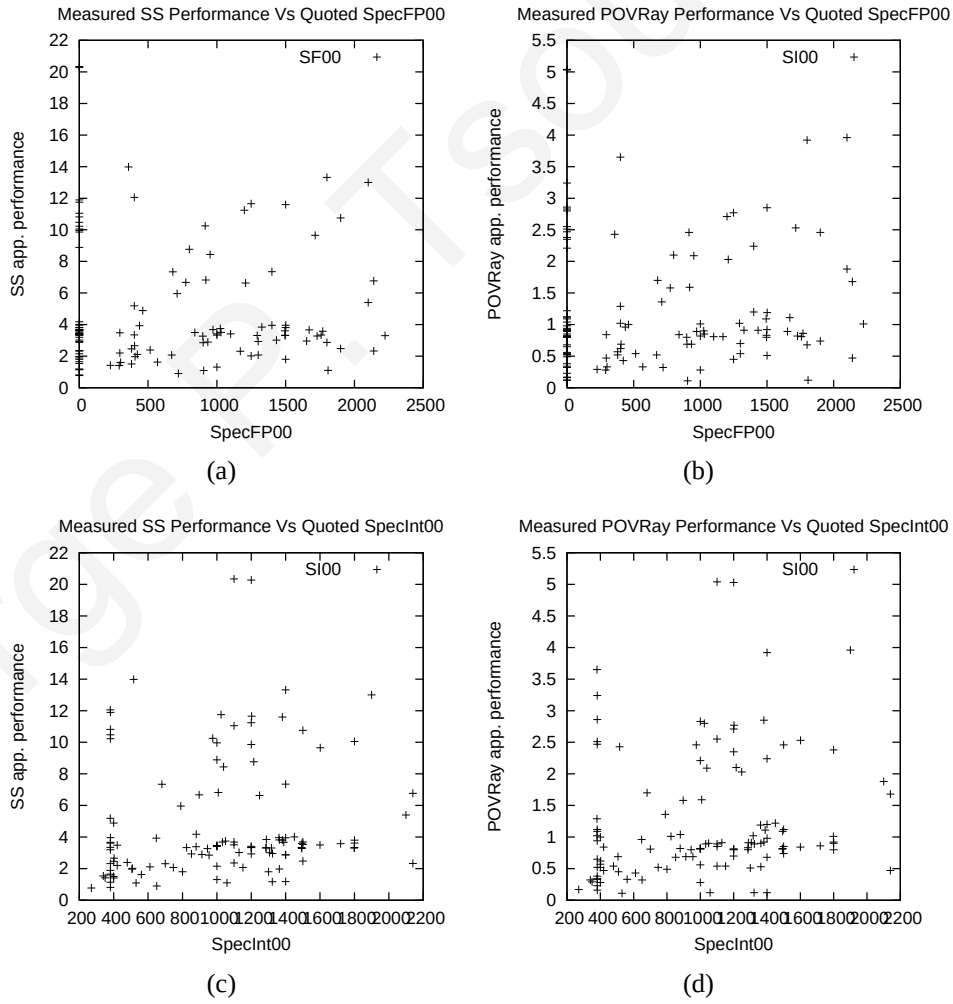


Figure 32: How the *quoted* performance metrics in Informations Systems relate to actual application performance.

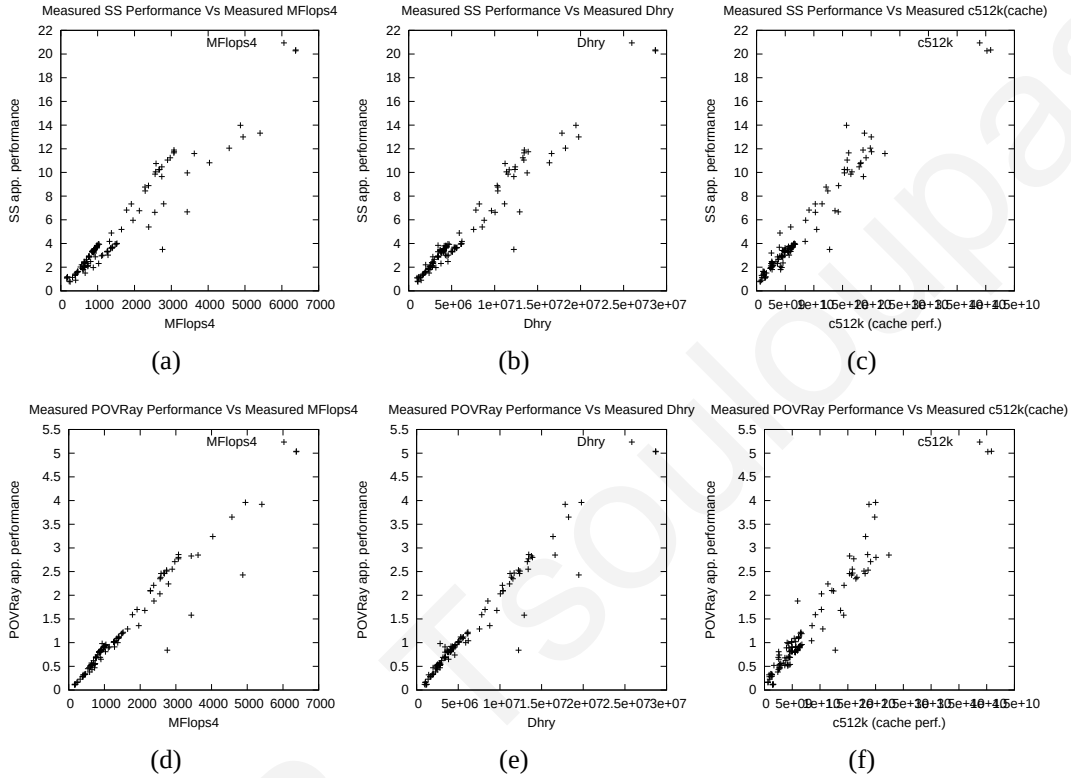


Figure 33: How the *measured* performance metrics in Informations Systems relate to actual application performance.

Since the SPEC benchmarks are commercial, it is unlikely that Resource providers or administrators will actually run the benchmarks on the specific machines. When the values *are* provided they are usually the number published by the vendor of the hardware, a measurement probably obtained under quite different circumstances and in a very tuned setting.

Even when actually supplied, values quoted by site administrators cannot be relied upon; Figure 32 shows how the quoted performance metrics in Informations Systems relate to actual application performance.[6]

[6]It is important to note that the benchmarks themselves are not criticized, not even their application in this context. The SPECInt, SPECFloat and their variants are highly respected and appreciated and would

Figure 33 show how the *measured* performance metrics relate to application performance. Regardless of the type of microbenchmark used, it is obvious that the metrics correlate to the actual application performance (a detailed study on metric follows in later sections).

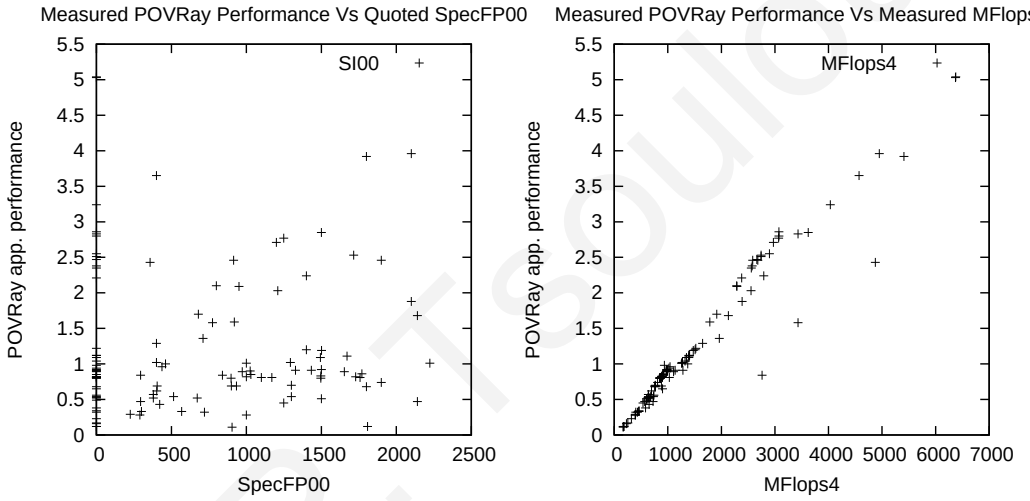


Figure 34: How the quoted performance metrics in Informations Systems relate to actuall application performance.

Contrasting the two graphs, illustrates the effectiveness of a **quoted** metric (Figure 34(a)) in contrast to a measured metric (Figure 34(b)). The charts speak for themselves; Clearly, the **quoted** metric does a very poor job in justifying application performance.

Another approach for obtaining a more realistic ranking of resources would be to run the application itself on the resources, collect, analyze and make decisions based on the

probably do a much better job characterizing resources had they met the free/open-source criterion that needs to be imposed.

results. This has been the subject of Section 5.1. This, of course, would be a rather costly endeavor for the following reasons:

- The number of applications that run on the grid is growing rapidly. Taking one large infrastructure (EGEE) as example, the number of VO's alone is over 100, and each VO potentially has several applications in it's toolkit.
- VO's are usually mapped to different resources, so the performance experienced by one VO can be quite different than that of another.
- Application performance is in some cases dependent on input parameters and data-sets, thus further increasing the number of experiments that need to be done.
- The number of resources is growing, for example the EGEE infrastructure currently spans around 300 sites having queues that are are well into the thousands.
- The infrastructure is volatile, new nodes enter and leave the grid, VO resource allocations change often, Grid resources are upgraded, re-configured and many times mis-configured. This calls for repeated measurements in order to have up-to-date information.

In order to overcome these problems, the number of measurements that need to be taken has to be radically reduced. The methodology that follows tries to address exactly these issues.

5.2.2 The Ranking Methodology

The GridBench tool provides a *SiteRank module* that allows the user to interactively and semi-automatically build a *ranking model*. A *ranking model* consists of *filtering*, *aggregation* and *ranking functions* (Figure 35).

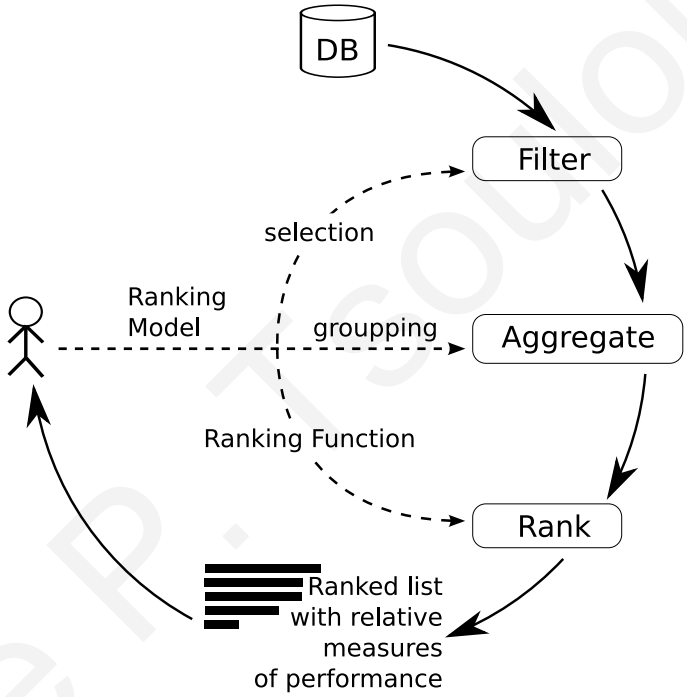


Figure 35: The ranking process.

Filtering refers to a user selection regarding which results will be included or excluded in the ranking process. *Attribute filtering* 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.

Aggregation allows the user to specify grouping of the measurements. The user can specify whether each measurement will count equally towards the evaluation of a site,

irrespective of which worker-node it was executed on. In this case, the reported metric may possibly be less representative of the resource as a whole because some individual worker-nodes may be over-represented. On the other hand, this will tend to be more representative of what the user actually experiences 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 a base value. The base values are configurable; in these experiments values from a typical 3.0GHz Xeon worker-node were used. For example, the value of 1050.0 was used to normalize the Mflops4 metric. The aggregation step is also important for the conversion of vector-type metrics, such as the ones produced by CacheBench into scalars (see later description on the *c512k* metric) so that they can be used in ranking functions.

Ranking Function Construction: The end goal of this methodology is a ranked list of computational resources that reflects the performance that users will experience running a specific application. It involves establishing a relationship between application performance and a set of low-level measurements. The process is illustrated in Figure 36, and it is outlined by the following steps:

1. **Sampling**: Obtain low-level performance metrics $\vec{m}$ for a small sample of resources – typically 10-15% of the full-set of resources. For the same sample of resources also obtain application performance measurements, i.e. application completion times. The application performance of this sample is denoted α where each $\alpha = 1/(completion\ time)$.

2. **Ranking Function Generation**: Determine a *Ranking Function* R based on the low-level metric data $\vec{m}$ and application performance α, so that $\alpha = R(\vec{m})$. This involves the selection of the low-level metrics that closely correlate to this application's performance, followed by a linear fit of the data, i.e. multivariate regression.

3. **Estimation**: For the set of the remaining resources, obtain only low-level performance metrics $\vec{M}$, and apply the ranking function in order to obtain an estimate of the application performance A_{est} such that $A_{est} = R(\vec{M})$. Sorting A_{est} produces the *Rank Estimation*.

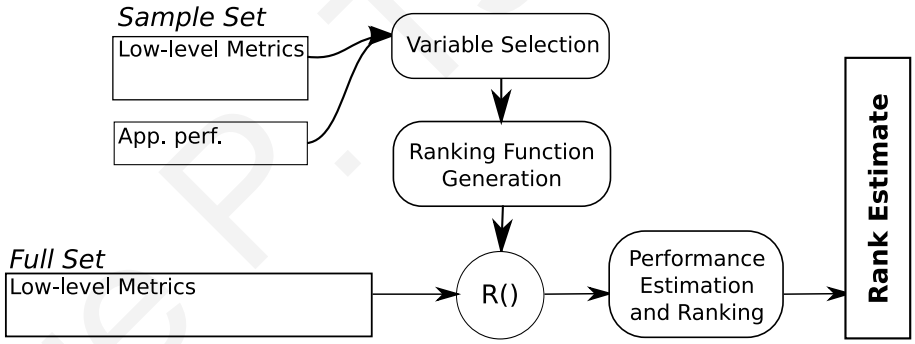


Figure 36: *Rank Estimate* generation process outline.

Section 5.3 provides a complete experiment that illustrates this process in greater detail.

5.2.3 Metrics

Selecting the *right* metrics to characterize the resources is of utmost importance in order to adequately characterize the major computational characteristics that affect application performance. In fact, a *good set of metrics* one that can adequately explain the

Table 6: Metrics and Micro-Benchmarks.

Factor	Metric	Delivered By
CPU	Floating-Point operations per second	Flops
CPU	Integer operations per second	Dhrystone
Main memory	sustainable memory bandwidth in MB/s (copy,add,multiply,triad)	Stream
Main memory	Available physical memory in MB	Memsize
Cache	memory bandwidth using different memory sizes in MB/s	CacheBench
Disk (local)	Disk bandwidth for read/write/rewrite	bonnie++
Interconnect (MPI)	latency, bandwidth and bisection bandwidth	MPPTest

performance of several distinct applications. In the process of picking the right metrics and the right benchmarks to deliver these metrics, the investigation was limited to freely available tools that could be widely deployed and run. Keeping the number of metrics low was a primary aim and, whenever possible, well-known metrics were favored. A more detailed discussion on the benchmarks can be found in [70].

Table 5.2.3 shows a list of low-level metrics and the associated benchmarks.

The Flops benchmark yields 4 metrics, *Mflops1*, *Mflops2*, *Mflops3* and *Mflops4*, each consisting of different mixes of floating-point additions, subtractions multiplications and divisions. Dhrystone yields the *dhry* integer performance metric. The STREAM memory benchmark yields the *copy*, *add*, *multiply* and *triad* metrics which measure memory bandwidth using different operations.

5.2.3.1 The c512k cache metric

Because of the multi-level hierarchical structure of memory architectures –that involve Level1, Level2, Level3 caches and sometimes beyond– and due to the fact that there are numerous heterogeneous types of CPU's in a typical Grid environment, there is a need

to uniformly characterize cache without the assumption of availability of special cache instruments. A widely accepted approach for cache performance measurement is the one taken by cachebench [49].

In this empirical approach to determining some of the parameters of the memory subsystem, the benchmark measures memory bandwidth B by allocating and accessing progressively larger array sizes s, the CacheBench benchmark produces a series of values B_s where $s = 2^8, 2^9, 2^{10} \ldots 2^n$.

This is easiest to interpret in the form of a graph of *size* versus *bandwidth*. An illustration of such a graph is shown in Figure 37. It shows the memory bandwidth (y-axis) as it is measured when accesing progressively larger allocated memory sizes. We observe the first "knee" on the curve when the allocation of memory that is accessed exceeds the first level of cache. The are typically more knees on the curve as the memory allocation reaches subsequent levels of cache.

Figure 37: Typical result of *cachebench*

Figure 38 shows examples of cache measurement from different machines. They are obviously different in both the actual size of the different levels of cache *and* the bandwidth.

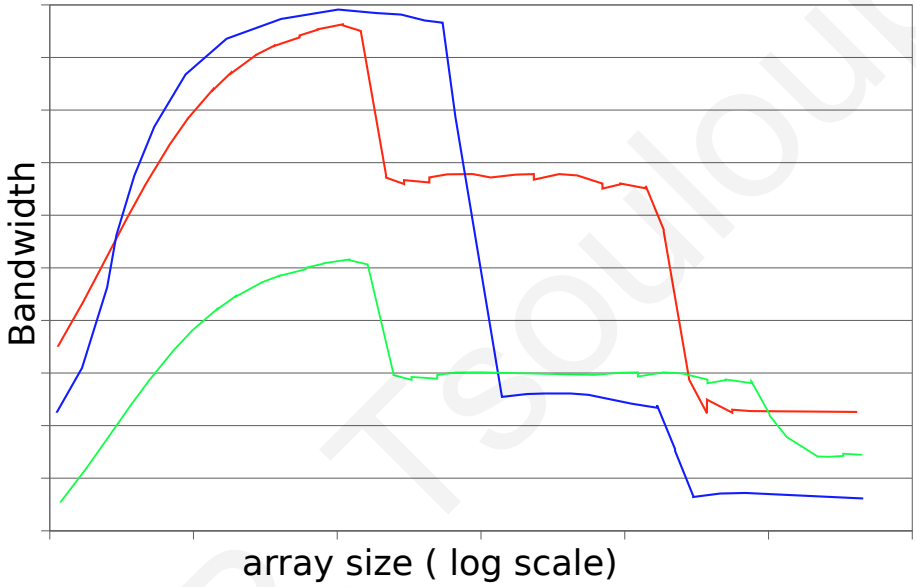


Figure 38: Heterogeneous resources yield different results that are difficult to compare

The *c512k*, *c1M*, *c2M*, *c4M* and *c8M* are introduced in order to be able to uniformly compare cache performance. They are simple metrics that aim to characterize cache from the point of view of applications with differnt memory foot-prints. We make the assumption that application performance is proportinal to cache size *and* proportional to cache bandwidth – perhaps not directly proportional in either case but proporional nonetheless. The calculation of the metrics involves calculating the area under the curve in specific intervals as shown in Figure 39.

Each of the metrics is calculated by summing up the product of the bandwidths and respective sizes, thus we derive a metric that takes into account both the cache size *and*

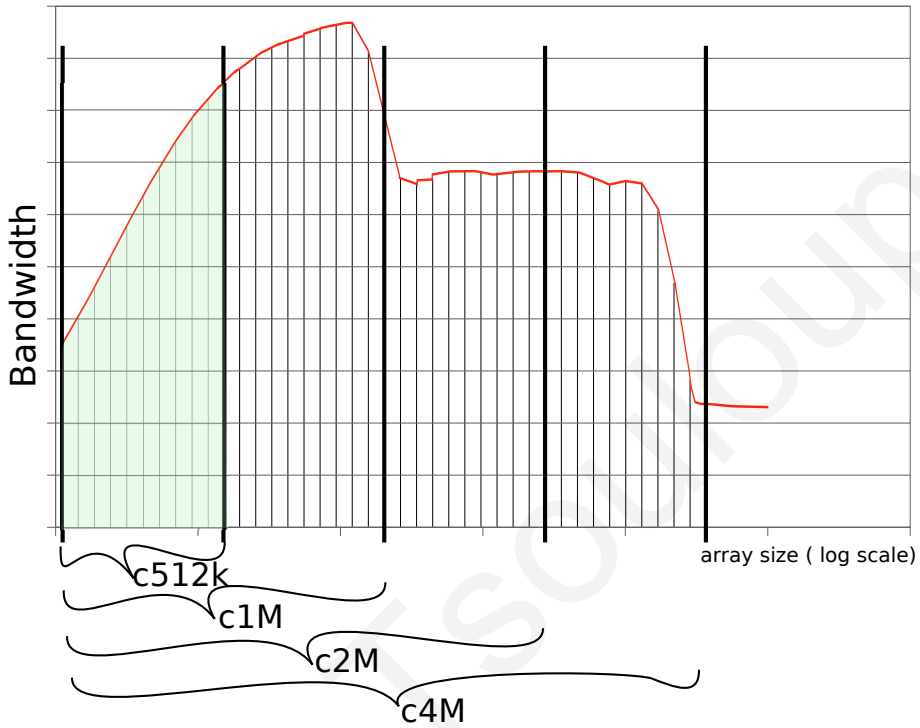


Figure 39: The area under the size/bandwith curve

the cache speed (i.e. bandwidth) : $\sum_{s=8}^{n} s \times B_s$. For example, summing up to 512kb, i.e. $\sum_{s=8}^{19} s \times B_s$ yields the *c512k* metric. This is done for sizes up to 512kb, 1Mb, 2Mb, 4Mb, 8Mb yielding the metrics *c512k*, *c1M*, *c2M*, *c4M* and *c8M* respectively. This is roughly equivalent to taking the *average* of the measured bandwidths in the given range.

This approach alleviates the problem of looking up the cache size for the multitude of CPU's on the Grid, or detecting the cache sizes of a potentially multilevel cache, while at the same time taking the cache bandwidth into account.

5.3 Experimentation

In this section I demonstrate the proposed methodology by automatically determining a *Ranking Function*, obtaining a *Ranking Estimate* and validating that the Ranking Estimate is accurate by directly measuring the performance of the application. This is done for two applications, on a set of about 230 sites that belong to the EGEE infrastructure. Two serial applications were used:

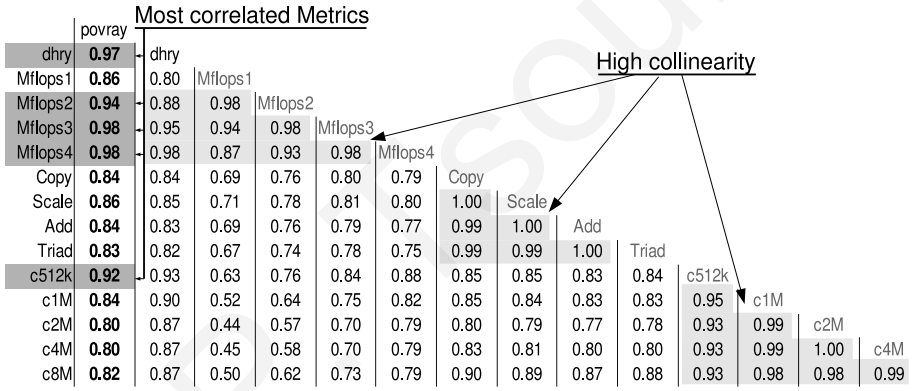
- **povray**: The Povray v3.6 ray-tracing application using the *benchmark.pov* scene at a 40x40 resolution.
- **sisc**: The SimpleScalar, computer architecture simulation using a sample data-set.[7]

For this experiment, I aimed at having between 2 and 3 measurements from each computational resource. One noteworthy fact is that I could only obtain results for about 160 out of the 230 sites. This was partly due to errors and site unavailability, but also due to exhausted quotas at some resources. I used the GridBench framework to obtain the measurements. The process of integrating the two applications into GridBench including the compilation took less than one hour and only needs to be performed once. The process of actually running all the experiments took less than 10 minutes, although I did have to

[7] Limited execution privileges for the Virtual Organization through which I performed the experiments, dictated that I use parameters resulting in short application completion times. This applied both to **povray** and to **sisc**.

wait for a day or so until the results from all the queued jobs were in. I then exported these results into an open-source statistics software package[8] ("R").

The data-set obtained by running the benchmarks on all the available computational resources will be referred to from now on as the *full-set*. Out of the full-set, I obtained a random sample, henceforth referred to as the *sample-set*, with results from 24 resources (15% of the full-set). A *correlation matrix* indicates which metrics are most correlated



	povray	dhry	Mflops1	Mflops2	Mflops3	Mflops4	Copy	Scale	Add	Triad	c512k	c1M	c2M	c4M
dhry	**0.97**													
Mflops1	**0.86**	0.80												
Mflops2	**0.94**	0.88	0.98											
Mflops3	**0.98**	0.95	0.94	0.98										
Mflops4	**0.98**	0.98	0.87	0.93	0.98									
Copy	**0.84**	0.84	0.69	0.76	0.80	0.79								
Scale	**0.86**	0.85	0.71	0.78	0.81	0.80	1.00							
Add	**0.84**	0.83	0.69	0.76	0.79	0.77	0.99	1.00						
Triad	**0.83**	0.82	0.67	0.74	0.78	0.75	0.99	0.99	1.00					
c512k	**0.92**	0.93	0.63	0.76	0.84	0.88	0.85	0.85	0.83	0.84				
c1M	**0.84**	0.90	0.52	0.64	0.75	0.82	0.85	0.84	0.83	0.83	0.95			
c2M	**0.80**	0.87	0.44	0.57	0.70	0.79	0.80	0.79	0.77	0.78	0.93	0.99		
c4M	**0.80**	0.87	0.45	0.58	0.70	0.79	0.83	0.81	0.80	0.80	0.93	0.99	1.00	
c8M	**0.82**	0.87	0.50	0.62	0.73	0.79	0.90	0.89	0.87	0.88	0.93	0.98	0.98	0.99

Figure 40: Correlation Matrix for the *povray* application.

to application performance; this is shown in Figure 40. The problem of collinearity must be taken into consideration when narrowing down the selection of metrics. As shown in Figure 40 some metric groups are highly collinear, in such cases we eliminate the collinear metrics by selecting one metric out of the group, i.e. the one with the highest correlation to the application. In this example I kept *Mflops4* and discarded *Mflops2*, *Mflops3* and *dhry*. Selecting the *Mflops4* and *c512k* metrics for building the Ranking Function, leads to the next step, i.e. calculating the a and b coefficients in order to best satisfy:

[8]Use of the R software was limited to establishing the relationship between the low-level metrics and application performance, and the validation of the results. All charts included in this chapter were created using GridBench

$$\alpha_{povray} = a \times Mflops4 + b \times c512k$$

Outlier removal is achieved by performing a linear regression, and data-points that fall more than two standard deviations away from the rest are filtered out. In this specific example, 2 out of the 18 points were dropped. Linear regression is performed once again using the filtered sample-set, which yields the coefficients $a = 0.94$ (for *Mflops4*) and $b = 0.46$ (for *c512k*). Finally, I apply this model on the *full-set* in order estimate the performance of the application:

$$\vec{A}_{povray} = 0.94\vec{M}_{Mflops4} + 0.46\vec{M}_{c512k}$$

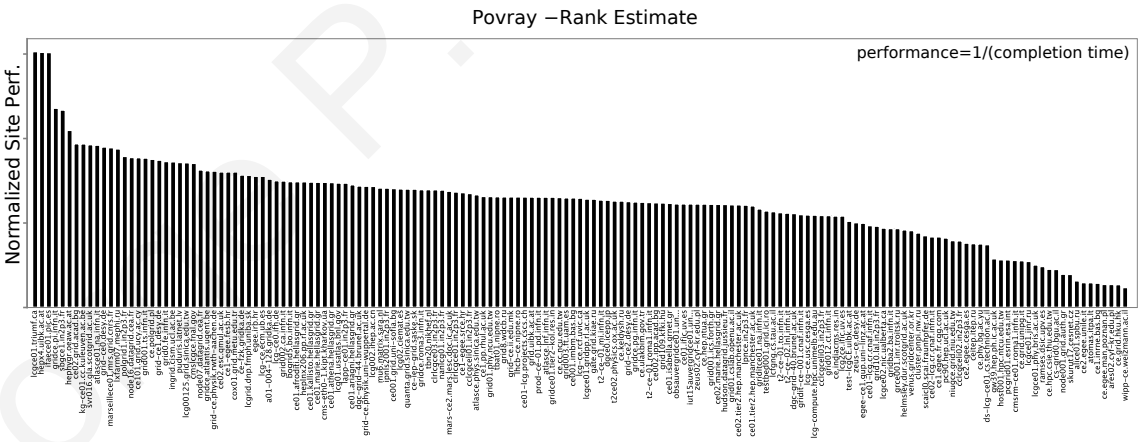


Figure 41: Rank Estimate for the **povray** application

Ordering the list of resources by A_{povray} gives the *Rank Estimate*. The Rank Estimate is shown in Figure 41.

In order to test that the Ranking Estimate is accurate the performance of the application was directly measured for the whole infrastructure. This is only necessary in order to

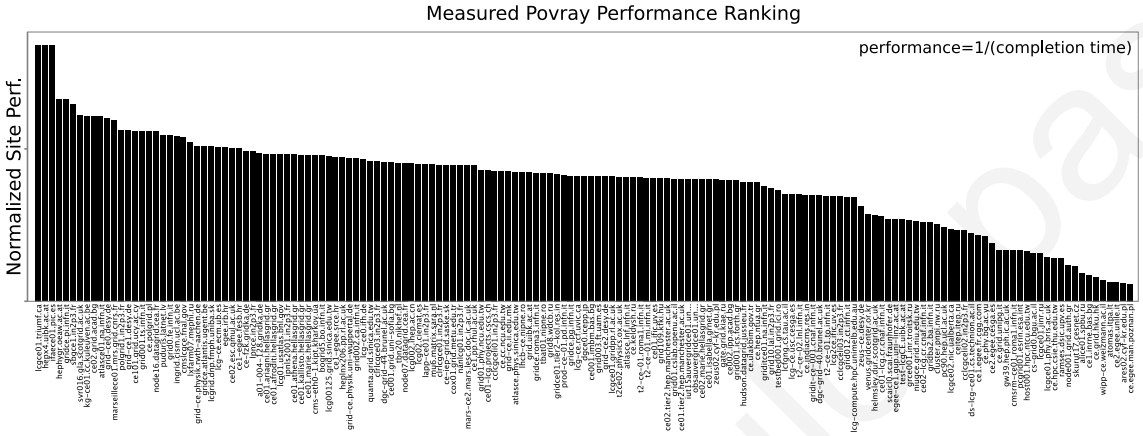


Figure 42: Measured **povray** performance on 159 resources of the EGEE infrastructure.

validate the model and not part of the methodology. The measured performance is shown in Figure 42.

The agreement between the Rank Estimate and the measured ranking can be statistically tested by calculating the *rank correlation*. There are several ways of doing this, such as Kendall's τ, which ranges from -1 to 1 and is also known as the "bubble-sort distance". Kendall's τ yielded $\tau = 0.90$. Spearman's ρ, which again ranges from -1 to 1, yielded $\rho = 0.977$. Finally, Pearson's correlation coefficient yielded 0.98. All three of the statistics show that the two rankings are quite similar. The τ statistic appears considerably lower that the other two, due to the fact that the data-set contains a lot of resources that are of almost identical performance. Extremely small fluctuations in measurement are enough to change the ordering. Yet, since the performance of the resources is nearly identical, so the reordering is not very significant. For this reason the author is inclined to take $\rho = 0.977$ as the more representative measure.

For the second application, **sisc**, the same methodology was used as well as *the same sample-set* that was used in the previous case. The metrics dictated by the correlation matrix are *dhry* and *c512k*. Performing the regression, outlier removal and then estimating the metric coefficients yields:

$$\vec{A}_{sisc} = 0.27\vec{M}_{dhry} + 0.18\vec{M}_{c512k}$$

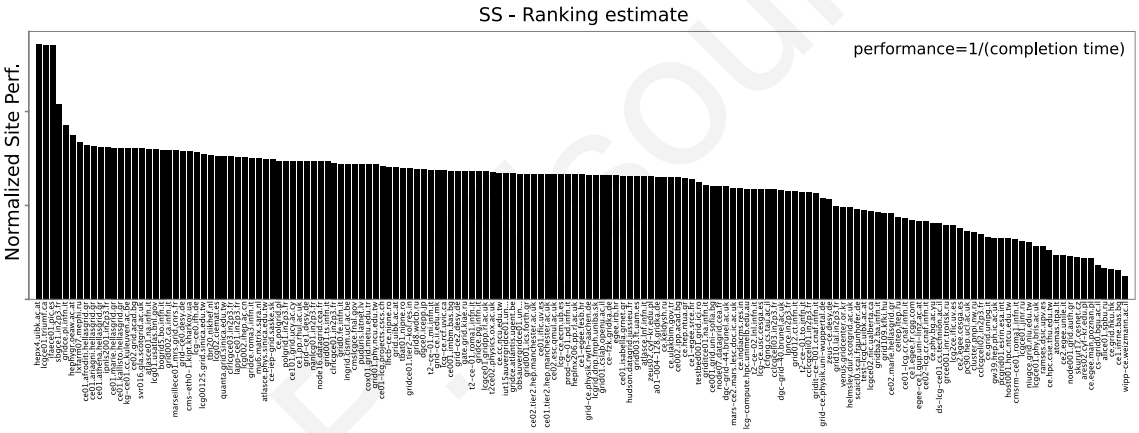


Figure 43: Rank Estimate for the **sisc** application on the EGEE infrastructure.

The Ranking estimate is given in Figure 43. The correlation of estimated and actual is again quire high with a value of $\rho = 0.959$. Thus, for both applications the ranking of resources based on low-level measurements provides results that are very close to the ranking produced by running the application itself.

Ranking based on derived models of low-level metrics, describes an alternative way of choosing and ranking resources. I propose a semi-automated user-driven approach to ranking Grid resources that employs user-specified metrics and *ranking functions*.

The process of running benchmarks collecting and analyzing results and generating ranked lists, would simply not be feasible if it had to be done manually, especially if it

had to be done by the end user. Furthermore, users could *audit* verify the "advertised" performance of a resource by running these light-weight benchmarks, or even detect problems at certain sites. Eventually, resource performance information will be coupled with resource pricing information. Users will then be able to "shop around" and pick the right resources (e.g. black-listing or white-listing) in order to influence the matchmaking process is a way that benefits them. The SiteRank module of the GridBench tool allows the user to interactively construct and modify ranking functions based on the collected measurements. The *Ranking Estimate* has proven to be quite accurate with a very high correlation to measured application performance for at least two applications, *povray* and *SimpleScalar*.

In this chapter I have illustrated that current approaches to expressing the performance of resources, such as publishing the *quoted*, not measured, GlueHostBenchmarkSF00 and GlueHostBenchmarkSI00 metrics into the information system are not satisfactory, since they do not correlate well with at least the two applications that I have investigated.

Chapter 6

Summary and Conclusion

The work presented here has the distinct goal of providing a methodology that will improve the way resource selection is performed as it pertains to computational resources in Grid environments.

The main contributions of this work can be summarized in the following:

- An approach towards *putting performance measurements into context* in order to effectively utilize measurements;
- Designed and implemented *a tool for testing and performance evaluation* of Grid resources that is useful on real, large, state-of-the-art systems in production today;
- Illustrated how the current approach using quoted metrics can be flawed and contrasted against the argument of using *measured user-obtained metrics*;
- Shown how an easily obtained metric, *c512k* (*c1M* , *c2M*), correlates well with the investigated applications.

- Proposed a powerful yet very simple *methodology for Grid computational resource ranking*;

6.1 Putting performance measurements in context.

For a metric or specific measurement to have any meaning, it needs to be put into context, especially in a large dynamic system such as the Grid. A measurement, with the associated context should at least contain *(i)* the definition of a specific test or benchmark invocation with specific parameters (the *what*); *(ii)* the target resources (the *where*); *(iii)* the time of execution (the *when*); *(iv)* the status of the target machines during execution collected through monitoring (the *state*); and, obviously, *(v)* the resulting metrics (the *result*).

GBDL (the GridBench Definition Language) is an XML-based language, introduced in order to describe the measurement's context. It encodes basic information required to describe and execute tests and benchmarks. It enables the annotation of test or benchmark definitions with performance-related metadata representing the conditions of a particular experiment and the metrics derived from that experiment. The main goals of GBDL are to:

- Allow for a standardized definition of tests and benchmarks that is independent of the underlying middleware platforms used to execute them;
- Enable the specification of the monitoring information that should be collected during a benchmark execution from an available Grid monitoring system;
- Serve as a container for the context-augmented results.

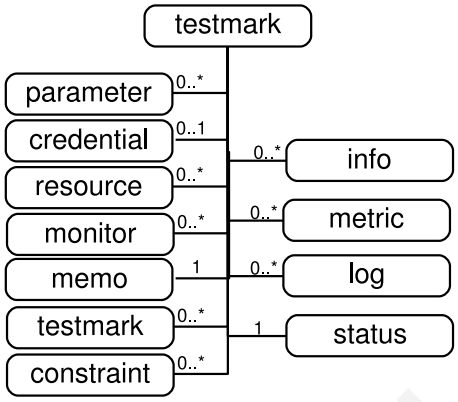


Figure 44: Structure of a GBDL document.

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

Different middleware require different Job Definition Languages such as RSL, JDL or JSDL. For this reason, GBDL encodes the basic required information for executing the test or benchmark job and obtaining the measurement. The GBDL can then be translated to the native description of the job as the specific middleware requires.

6.2 A tool for performance evaluation and testing

A system has been developed that aims to alleviate many of the complexities of testing and benchmarking large numbers of Grid resources. The tools serves 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 user interface (Figure 45). 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.

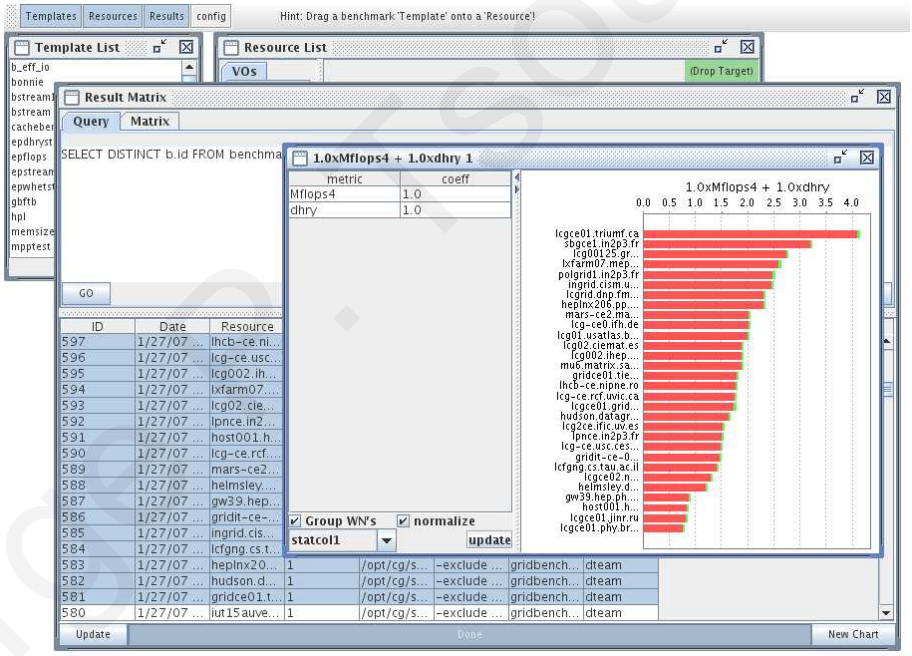


Figure 45: The GridBench user interface.

Several use-case scenarios were presented which illustrate the functionality and ease of use of the tool. One use-case illustrated how end-users and administrators can perform test/benchmark experiments, both for resource selection and for determining the operational status of resources. Another use-case illustrated how a user or application developer

can obtain results from new application-based benchmarks using the GridBench framework. Another use-case presented a run-though of 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.

6.3 Auditing resource performance: The argument for using measured, end-to-end user-obtained metrics.

Resource performance is interesting to users and operators. This is indicated by the fact that current Information Systems include such information in the form of *SPEC Int 2000* and *SPEC Float 2000* values. Yet, the mechanisms by which these performance metrics are obtained and published leaves a lot of room for error. For the purpose of illustrating this, Figure 46 shows how the quoted metrics and measured metrics correlate to application performance.

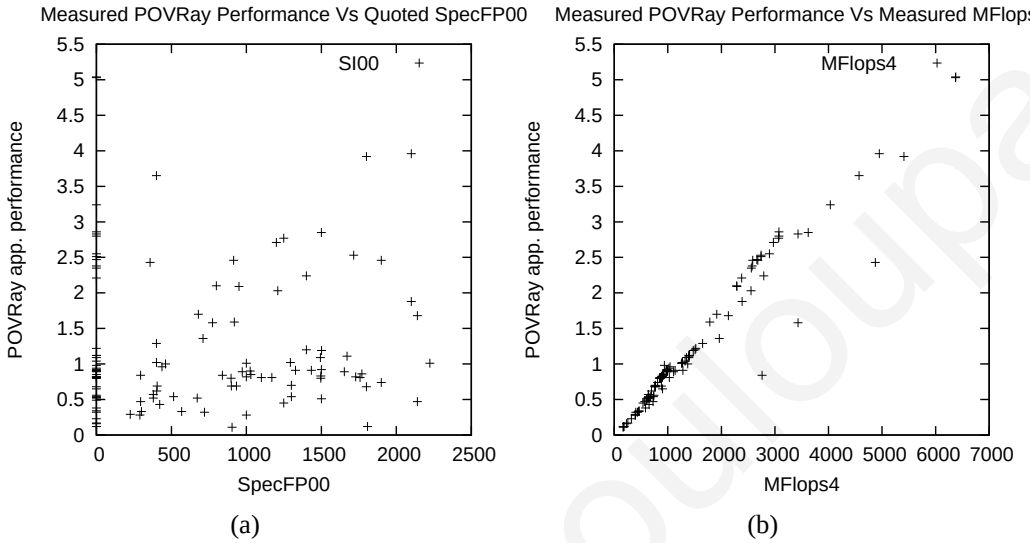


Figure 46: How the quoted performance metrics in Informations Systems relate to actuall application performance.

The spread of data-points in Figure 46(a) indicates the low correlation of the "advertised" metric (SPEC-Float) that resources *claim* to have, to actual application performance. Figure 46(b) shows a chart of actual application performance versus measured performance (MFlops4). The correlation in this case, manifested as a low spread, is much higher. This illustrates that the measured metrics are much more effective in justifying application performance.

6.4 The c512k metric and it's correlation to actual application performance.

Choosing the right metrics to collect is of vital importance, as an incomplete set of metrics will yield poor characterization. For example, initial experiments did not include metrics that characterize the memory cache. While collecting measurements about the cache, the data was in a form that was rather difficult to integrate into a regular function.

Also, it was initially falsely assumed that the cache effects would be largely accounted for in other metrics. The initial results were not very encouraging; but including the cache metrics, i.e. *c512k*, completely changed the situation.

By summing up the product of the bandwidths and respective sizes (obtained from CacheBench) we derive a metric that takes into account both the cache size *and* the cache speed : $\sum_{s=8}^{n} s \times B_s$. This is done for sizes up to 512kb, 1Mb, 2Mb, 4Mb, 8Mb yielding the metrics *c512k*, *c1M*, *c2M*, *c4M* and *c8M* respectively. Summing up to 512kb, i.e. $\sum_{s=8}^{19} s \times B_s$ yields the *c512k* metric.

The inclusion of *c512k* as a candidate variable improved correlation considerably. In one application, *SimpleScalar*, the ρ rank correlation statistic was improved from from approximately $\rho = 0.8$ to $\rho = 0.96$. This also reaffirms the wide-spread impression that a well-sized, fast cache is essential to many computational applications.

6.5 A methodology for computational resource ranking.

The *SiteRank module*, an integral module of the GridBench tool, allows the user to interactively build a *ranking model*; the process is outlined in Figure 47.

In this methodology **filtering** generates a selection of results the form the basis of the ranking model. The next setp, **aggregation** allows for grouping of the measurements. It can be seen as a pre-processing step that will generate normalized representative metrics for each resource, such as the *mean*, *standard-deviation*, *min*, *max*, *average-deviation* and *count*.

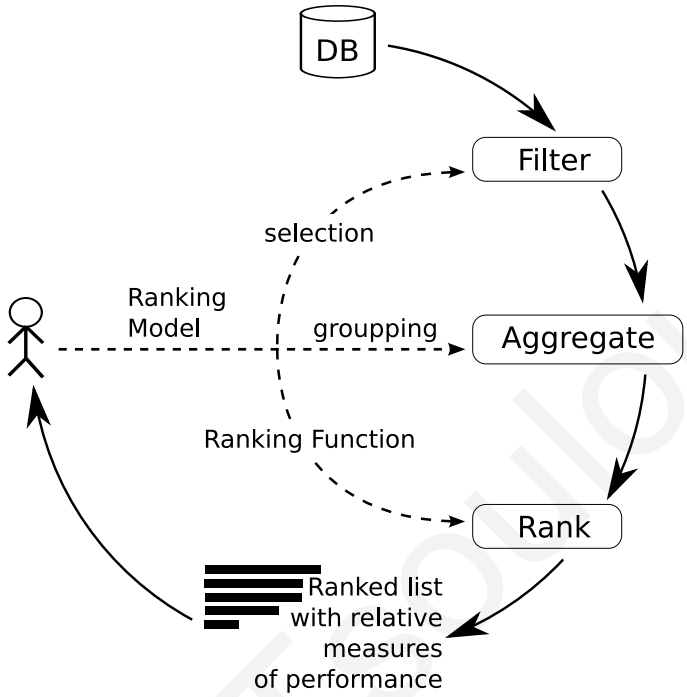


Figure 47: The ranking process workflow

The last and most important step is the **Ranking Function Construction**. It is itself comprised of three steps: **Sampling**, **Ranking Function Generation** and **Estimation**. In a nutshell, once low-level metrics and application performance performance metrics are collected for a sample of the resources, the measurements are used to derive a function that is to become the ranking function. Variable selection is done by selecting variables (i.e. low-level metrics) with the highest correlation with the performance of the application at hand.

While the function can be arbitrarily complex and not necessarily linear, it has been found that in the investigated scenarios a small number of metrics, and a linear fit of the data suffices to produce a good ranking estimate. The goodness of the ranking statistically evaluated by calculating Spearman's ρ. In the investigated scenarios Spearman's ρ gave values of over 0.95, indicating that the rank *estimate* accurately reflected the *actual* rank.

6.6 Taking This Work Further

6.6.1 Wider Application Scope

In this thesis a small number of applications is investigated and the results yieled by this methodology are encouraging. The investigation of more applications, especially applications that are not CPU or memory bound, is one direction to take. It is useful to evaluate the extent to which the already described metrics provide sufficient characterization.

6.6.2 Applying the proposed ranking to scheduling

Building on the work presented in this thesis, the application of user-driven ranking functions in scheduling and resource allocation can be investigated in greater detail. This could involve the inclusion of detailed performance metrics in existing Information Systems, and the extension of ranking criteria in existing Grid job description languages in order to make use of this information.

6.6.3 Furthering on Auditing

The concept of auditing can be further investigated in several directions. One direction could be the use of micro-benchmarks for automated evaluation of Grid "resource health" and automated detection of degraded performance. For example, it has been observed during experimentation that many Grid resources (i.e. clusters) exhibit varying degrees of "internal uniformity". This usually arises from upgrades of just a fraction of the machines that make up the cluster, or simply by mixing machines of different capabilities into one

cluster. This heterogeneity of cluster nodes will potentially considerably affect observed application performance.

6.6.4 Parallel versus High-Throughput Applications

Grid infrastructures have been extensively considered as a fitting platform on which to run parallel applications that are loosely or tightly coupled. An example of such an application is given in Chapter 5 Section 5.1 and explained in detail in [66]. Technical and other issues have somewhat impeded the widespread deployment of parallel applications on Grids such as the EGEE. (One example of such problems is the complexity of efficiently scheduling and queuing of MPI applications.) It is to be expected that deployment of MPI applications will increase as the infrastructure is becoming more and more stable, a trend also indicated by the establishment of the EGEE MPI working-group. Since parallel applications/jobs obviously occupy more individual machines in clusters than regular jobs, it becomes all the more important to effectively audit and evaluate the performance of resources, as a single “bad” cluster-node will seriously affect overall application performance. The application of the methodology proposed in this thesis can be investigated in this context.

6.6.5 Extending the Tool

Currently, one part of SiteRank that involves the calculation of the correlation matrix that effectively drives the variable selection process, is performed outside the GridBench

interface (i.e. in a statistical package). This, together with an automated variable selection process, with specific attention to co-linearity of the metrics can greatly improve the usability of the tool and streamline the process of user-driven ranking.

Appendix A

GBDL Definition and Examples

GBDL Translation Examples

A sample GBDL document describing an execution of the *cachebench* benchmark.

```
<benchmark name="cachebench" date="20040221195616"
           type="single" model="false" description="" >
  <location>
    <resource name="ce.grid.cesga.es" cpucount="2"
              wncount="1" jobmanager="null"/>
  </location>
  <parameter name="executable" type="value"
             dataType="0">cachebench</parameter>
  <parameter name="execpath" type="value"
             dataType="0">/opt/cg/gridbench/bin</parameter>
  <parameter name="stage_executable" type="value"
             dataType="0">manual</parameter>
  <parameter name="pattern" type="value" dataType="0"
      >read write readwrite memset memcopy</parameter>
  <parameter name="inbetween" type="value"
             dataType="0">1</parameter>
  <parameter name="mem_log2" type="value"
             dataType="0">28</parameter>
  <parameter name="time_iter" type="value"
             dataType="0">2</parameter>
  <parameter name="repeat" type="value"
             dataType="0">1</parameter>
  <parameter name="cpucount" type="value"
```

```
                dataType="0">2</parameter>
    <metric   name="memset" type="value">
      <vector name="hostname">grid03.grid.cesga.es</vector>
      <vector name="size">256 384 ... 201326592 268435456</vector>
      <vector name="memset">1006.9 1458.8 ... 1.9 1.8</vector>
    </metric>
</benchmark>
```

EPStream

GBDL

```
<benchmark name="epstream" date="20031209111612" >
  <location>
    <resource name="cgnode00.di.uoa.gr"
              cpucount="8" jobmanager="jobmanager-pbs"/>
  </location>
  <parameter name="executable" type="attribute"
             dataType="0">epstream</parameter>
  <parameter name="execpath" type="attribute"
             dataType="0">/opt/cg/gridbench/bin</parameter>
  <parameter name="stage_executable" type="attribute"
             dataType="0">manual</parameter>
</benchmark>
```

JDL

```
#Automatically Generated by GridBench
StdOutput       = "std.out";
StdError        = "std.err";
Arguments       = "";
InputSandbox    = {"/opt/cg/gridbench/bin/epstream"};
Executable      = "epstream";
JobType         = "mpich";
NodeNumber      = 8;
OutputSandbox   = {"std.out","std.err"};
Requirements = other.CEId=="cgnode00.di.uoa.gr/jobmanager-pbs";
```

RSL

```
&(resourceManagerContact="cgnode00.di.uoa.gr")
 (executable=
        $(GLOBUSRUN_GASS_URL)/opt/cg/gridbench/bin/epstream)
 (count=8)
 (jobtype=mpi)
```

GB_FTB

A file-transfer benchmark

GBDL

```
<benchmark name="gbftb" date="20031209113543" type="simple">
  <location>
    <resource name="ce01.lip.pt" cpucount="1"
              jobmanager="jobmanager-pbs"/>
  </location>
  <parameter name="executable"
             type="attribute" dataType="0">gbftb</parameter>
  <parameter name="execpath"  type="attribute"
             dataType="0">/opt/cg/gridbench/bin</parameter>
  <parameter name="stage_executable"
             type="attribute" dataType="0">manual</parameter>
  <parameter name="size"
             type="value" dataType="0">4096000</parameter>
  <parameter name="measurements"
             type="value" dataType="0">10</parameter>
  <parameter name="buffer_size"
             type="value" dataType="0">4096</parameter>
  <parameter name="target" type="value"
   dataType="0">gsiftp://apelatis.grid.ucy.ac.cy/tmp/gbftb</
      parameter>
  <parameter name="target1" type="value"
   dataType="0">gsiftp://ce010.fzk.de/tmp/gbftb</parameter>
  <parameter name="target2" type="value"
   dataType="0">gsiftp://xg001.inp.demokritos.gr/tmp/gbftb</
      parameter>
  <parameter name="target3" type="value"
   dataType="0">gsiftp://zeus24.cyf-kr.edu.pl/tmp/gbftb</
      parameter>
  <parameter name="target4" type="value"
   dataType="0">gsiftp://bee001.ific.uv.es/tmp/gbftb</parameter>
  <parameter name="target5" type="value"
   dataType="0">gsiftp://cluster.ui.sav.sk/tmp/gbftb</parameter>
  <parameter name="target6" type="value"
   dataType="0">gsiftp://ce01.lip.pt/tmp/gbftb</parameter>
   ...
</benchmark>
```

JDL

```
 #Automatically Generated by GridBench
StdOutput       = "std.out";
StdError        = "std.err";
Arguments       = " -s 4096000 -n 10 -b 4096
                  gsiftp://apelatis.grid.ucy.ac.cy/tmp/gbftb
                  gsiftp://ce010.fzk.de/tmp/gbftb
                  gsiftp://xg001.inp.demokritos.gr/tmp/gbftb
                  gsiftp://zeus24.cyf-kr.edu.pl/tmp/gbftb
                  gsiftp://bee001.ific.uv.es/tmp/gbftb
                  gsiftp://cluster.ui.sav.sk/tmp/gbftb
                  gsiftp://ce01.lip.pt/tmp/gbftb
                  gsiftp://cgnode00.di.uoa.gr/tmp/gbftb
                  gsiftp://aocegrid.uab.es/tmp/gbftb
                  gsiftp://xgrid.icm.edu.pl/tmp/gbftb";
InputSandbox    = {"/opt/cg/gridbench/bin/gbftb"};
Executable      = "gbftb";
OutputSandbox   = {"std.out","std.err"};
Requirements = other.CEId == "ce01.lip.pt/jobmanager-pbs";
```

RSL

```
&(resourceManagerContact="ce01.lip.pt")
(executable=$(GLOBUSRUN_GASS_URL)/opt/cg/gridbench/bin/gbftb)
(arguments= "-s" "4096000" "-n" "10" "-b" "4096"
 "gsiftp://apelatis.grid.ucy.ac.cy/tmp/gbftb"
 "gsiftp://ce010.fzk.de/tmp/gbftb"
 "gsiftp://xg001.inp.demokritos.gr/tmp/gbftb"
 "gsiftp://zeus24.cyf-kr.edu.pl/tmp/gbftb"
 "gsiftp://bee001.ific.uv.es/tmp/gbftb"
 "gsiftp://cluster.ui.sav.sk/tmp/gbftb"
 "gsiftp://ce01.lip.pt/tmp/gbftb"
 "gsiftp://cgnode00.di.uoa.gr/tmp/gbftb"
 "gsiftp://aocegrid.uab.es/tmp/gbftb"
 "gsiftp://xgrid.icm.edu.pl/tmp/gbftb")
```

High-Performance Linpack

This benchmark executable requires that a file ”HPL.dat” is in the current working directory. The class *ParameterHandler_hpl* generates this file and makes it available for staging to the target resource.

GBDL

```
<benchmark name="site_hpl" date="20031209114921" >
 <location>
  <resource name="ce01.lip.pt" cpucount="2" jobmanager="
     jobmanager-pbs"/>
 </location>
 <parameter name="executable" type="attribute">hpl</parameter>
 <parameter name="execpath" type="attribute">...</parameter>
 <parameter name="stage_executable" type="attribute">manual</
    parameter>
 <parameter name="stage_file" type="attribute">HPL.dat</
    parameter>
 <parameter name="problem_size" type="value">180</parameter>
 <parameter name="blocks" type="value">40</parameter>
 <parameter name="p" type="value">1</parameter>
 <parameter name="q" type="value">2</parameter>
 <parameter name="threshold" type="value">16.0</parameter>
 <parameter name="pfact" type="value">1</parameter>
 <parameter name="nbmin" type="value">2</parameter>
 <parameter name="ndiv" type="value">2</parameter>
 <parameter name="rfact" type="value">1</parameter>
 <parameter name="bcast" type="value">0</parameter>
 <parameter name="depth" type="value">0</parameter>
 <parameter name="swap" type="value">2</parameter>
 <parameter
        name="swapping_threshold" type="value">64</parameter>
 <parameter name="L1" type="value">0</parameter>
 <parameter name="U" type="value">0</parameter>
 <parameter name="equilibration" type="value">1</parameter>
 <parameter name="mem_alignment" type="value">8</parameter>
 <parameter name="cpucount" type="attirbute">2</parameter>
</benchmark>
```

JDL

```
#Automatically Generated by GridBench
StdOutput          = "std.out";
StdError           = "std.err";
InputSandbox       = {"/opt/cg/gridbench/bin/hpl","/home/georget/
   HPL.dat"};
Executable         = "hpl";
JobType            = "mpich";
NodeNumber         = 2;
OutputSandbox      = {"std.out","std.err"};
Requirements = other.CEId == "ce01.lip.pt/jobmanager-pbs";
```

GBDL DTD

```
<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT benchmark (location,(metric|monitor|parameter|
   benchmark)*)>
<!ATTLIST benchmark
     id ID #REQUIRED
     date CDATA #REQUIRED
     name CDATA #REQUIRED
     model CDATA #REQUIRED
     description CDATA #REQUIRED
     type (mpi|simple|composite) #REQUIRED >

<!ELEMENT location (resource)*>
<!ATTLIST location
     assigned (explicit|system) #IMPLIED >

<!ELEMENT parameter (#PCDATA)>
<!ATTLIST parameter
     name CDATA #REQUIRED
     dataType CDATA #IMPLIED
     type (value|list|attribute|attributelist) #REQUIRED >

<!ELEMENT metric (#PCDATA |vector)*>
<!ATTLIST metric
     name CDATA #REQUIRED
     unit CDATA #REQUIRED
     type (value|reference) #REQUIRED>

<!ELEMENT resource EMPTY >
<!ATTLIST resource
     cpucount CDATA #REQUIRED
     wncount CDATA #REQUIRED
     jobmanager CDATA #IMPLIED
     name CDATA #IMPLIED >

<!ELEMENT vector (#PCDATA)>
<!ATTLIST vector
     name CDATA #REQUIRED
     unit CDATA #IMPLIED
     url  CDATA #IMPLIED>

<!ELEMENT constraint (#PCDATA)>
<!ATTLIST constraint
     id IDREF #IMPLIED
     type (prerequisite|corequisite) #REQUIRED >
```

```
<!ELEMENT monitor (vector*)>
<!ATTLIST monitor
    type (jims|rgma) #REQUIRED
    query CDATA #REQUIRED>
```

Bibliography

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

[2] Enis Afgan, Vijay Velusamy, and Purushotham V. Bangalore. Grid resource broker using application 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 691–701. Springer, 2005.

[3] Globus Alliance. The Globus Toolkit. http://www.globus.org.

[4] S. Andreozzi. GLUE Schema implementation for the LDAP data model. Technical Report Technical Report. INFN/TC-04/16, Instituto Nazionale Di Fisica Nucleare, September 2004.

[5] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. Tortone, and C. Vistoli. GridICE: A Monitoring Service for the Grid. In *Proceedings of the Third Cracow Grid Workshop*, pages 220–226, October 2003.

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

[7] A. M. Artoli, A. G. Hoekstra, and P. M. A. Sloot. Mesoscopic simulations of systolic flow in the human abdominal aorta. *Journal of Biomechanics*, 39:873–884, 2005.

[8] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[9] G. Avellino, Stefano Beco, B. Cantalupo, Alessandro Maraschini, F. Pacini, Massimo Sottilaro, A. Terracina, D. Colling, F. Giacomini, E. Ronchieri, A. Gianelle, M. Mazzucato, R. Peluso, Massimo Sgaravatto, Andrea Guarise, Rosario M. Piro, Albert Werbrouck, Daniel Kouril, Ales Krenek, Ludek Matyska, M. Mulac, Jirí Pospísil, Miroslav Ruda, Zdenek Salvet, J. Sitera, J. Skrabal, M. Vocu, M. Mezzadri,

F. Prelz, Salvatore Monforte, and M. Pappalardo. The DataGrid Workload Management System: Challenges and Results. *Journal of Grid Computing*, 2(4):353–367, 2004.

[10] 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, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[11] Kazimierz Balos, Leszek Bizol, Michal Rozenau, and Krzysztof Zielilski. Interoperability architecture for grid networks monitoring systems. In *Proceedings of Cracow Grid Workshop*, pages 245–253, October 2003.

[12] Kazimierz Balos and Krzysztof Zielinski. JIMS - the Uniform Approach to Grid Infrastructure and Application Monitoring. In *4th Cracow Grid Workshop 2004*, December 2004.

[13] EECS Berkeley. Dwarf mine. http://view.eecs.berkeley.edu/wiki/Dwarf_Mine.

[14] Matthias Brune, Jörn Gehring, Axel Keller, Burkhard Monien, Friedhelm Ramme, and Alexander Reinefeld. Specifying resources and services in metacomputing environments. *Parallel Comput.*, 24(12-13):1751–1776, 1998.

[15] 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.

[16] Greg Chun, Holly Dail, Henri Casanova, and Allan Snavely. Benchmark probes for grid assessment. *Parallel and Distributed Processing Symposium, International*, 18:276a, 2004.

[17] Phillip Colella. Defining software requirements for scientific computing. presentation,(URL unavailable).

[18] J. Coles. Grid Deployment and Operations: EGEE, LCG and GridPP. In *Proceedings of the UK e-Science All Hands Meeting 2005*, 2005. http://www.allhands.org.uk/proceedings/2005 (accessed Oct. 2005).

[19] 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*, volume 2888 of *Lecture Notes in Computer Science*, pages 462–481. Springer, 2003.

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

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

[22] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, pages 181–194. IEEE Computer Society, 2001.

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

[24] Marios D. Dikaiakos. Grid benchmarking: Vision, challenges, and current status. *Concurrency and Computation: Practice and Experience*, 2006. In press (published online in June 13, 2006).

[25] J. J. Dongarra, H. W. Meuer, and E. Strohmaier. TOP500 supercomputer sites. *Supercomputer*, 11(2-3):133–163, June 1995.

[26] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15:1–18, 2003.

[27] 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.

[28] e2emonit. Egee network performance monitoring. http://www.egee-npm.org/e2emonit.

[29] EEMBC. The Embedded Microprocessor Benchmark Consortium. http://www.eembc.org.

[30] EGEE. Enabling Grids for E-SciencE project. http://www.eu-egee.org (last accessed December 2008).

[31] O. Ponce et al. Training of neural networks: Interactive possibilities in a distributed framework. In D. Kranzlmüller et al., editor, *9^{th} European PVM/MPI(LNCS)*, volume 2474, pages 33–40. Springer-Verlag, 2002.

[32] 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.

[33] I. Foster and C. Kesselman. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 4: Concepts and Architecture, pages 37–64. Elsevier, 2004.

[34] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[35] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3):200–222, 2001.

[36] Ganglia. http://ganglia.sourceforge.net (accessed Apr. 2005).

[37] Ganglia. The Ganglia Monitoring Systems. http://ganglia.sourceforge.net, (accessed Sep 2005).

[38] J. Gomes and 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*, number 3470 in Lecture Notes in Computer Science, pages 98–110. Springer, June 2005.

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

[40] Andreas Hanemann, Jeff W. Boote, Eric L. Boyd, Jérôme Durand, Loukik Kudarimoti, Roman Lapacz, D. Martin Swany, Szymon Trocha, and Jason Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *ICSOC*, volume 3826 of *Lecture Notes in Computer Science*, pages 241–254. Springer, 2005.

[41] E. N. Houstis, John R. Rice, and Randall Bramley, editors. *Enabling Technologies for Computational Science: Frameworks, Middleware and Environments*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[42] Alexandru Iosup and Dick Epema. Grenchmark: A framework for analyzing, testing, and comparing grids. *ccgrid*, 00:313–320, 2006.

[43] E. Kenny, B. Coghlan, G. Tsouloupas, M. Dikaiakos, J. Walsh, S. Childs, D. O'Callaghan, and G. Quigley. Heterogeneous grid computing: Issues and early benchmarks. In *Proc. ICCS 2005*, volume Part III of *LNCS3516*, pages 870–874, Atlanta, USA, May 2005.

[44] LCG. Large Hadron Collider Computing Grid. http://lcg.web.cern.ch (accessed Oct. 2004).

[45] Chuang Liu, Lingyun Yang, Ian Foster, and Dave Angelo. Design and Evaluation of a Resource Selection Framework for Grid Applications. In *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC '02)*, pages 63–72. IEEE Computer Society, 2002.

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

[47] 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.

[48] MPI. MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/docs/mpi-11.ps (accessed June 2006).

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

[50] David L. Oppenheimer, Vitaliy Vatkovskiy, Hakim Weatherspoon, Jason Lee, David A. Patterson, and John Kubiatowicz. Monitoring, analyzing, and controlling internet-scale systems with acme. *CoRR*, cs.DC/0408035, 2004.

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

[52] Pacini. Job Description Language: Attributes Specification. http://edms.cern.ch/document/590869/, May 2006.

[53] Rolf Rabenseifner, Alice E. Koniges, Jean-Pierre Prost, and Richard Hedges. The parallel effective I/O bandwidth benchmark: b_eff_io. In Christophe Cerin and Hai Jin, editors, *Parallel I/O for Cluster Computing*, chapter 4, pages 107–132. Kogan Page Ltd., February 2004.

[54] Rajesh Raman, Miron Livny, and Marvin 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)*, pages 140–147. IEEE Computer Society, 1998.

[55] Rajesh Raman, Miron Livny, and Marvin H. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138, 1999.

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

[57] RGMA. R-GMA: Relational Grid Monitoring Architecture. http://www.r-gma.org/ (accessed Dec. 2004).

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

[59] Jaswinder P Singh, Wolf Weber, and Anoop Gupta. Splash: Stanford parallel applications for shared-memory. Technical report, Stanford University, Stanford, CA, USA, 1992.

[60] 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.

[61] 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.

[62] SPEC. Standard Performance Evaluation Corporation. http://spec.org.

[63] Alan Su, Francine Berman, Richard Wolski, and Michelle Mills Strout Y. Using apples to schedule simple sara on the computational grid. *International Journal of High Performance Computing Applications*, 13, 1999.

[64] TerraGrid. The TeraGrid Project. http://www.teragrid.org, (accessed May 2004).

[65] Alfredo Tirado-Ramos, Peter M. A. Sloot, Alfons G. Hoekstra, and Marian Bubak. An integrative approach to high-performance biomedical problem solving environments on the grid. *Parallel Comput.*, 30(9-10):1037–1055, 2004.

[66] A. Tiramo-Ramos, G. Tsouloupas, M. D. Dikaiakos, and P. Sloot. Grid Resource Selection by Application Benchmarking: a Computational Haemodynamics Case Study. In *Computational Science - ICCS 2005, 5th International Conference, Atlanta, GA, USA, May 22-25, 2005, Proceedings, Part I.*, volume 3514, pages 534–543. Springer, May 2005.

[67] Francesca Tosi, Stefano Ubertini, S. Succi, and I. V. Karlin. Optimization strategies for the entropic lattice boltzmann method. *J. Sci. Comput.*, 30(3):369–387, 2007.

[68] 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.

[69] G. Tsouloupas and M. D. Dikaiakos. GridBench: A Workbench for Grid Benchmarking. In *In Advances in Grid Computing - EGC 2005. European Grid Conference. Amsterdam, The Netherlands. February 14-16, 2005, Revised Selected Papers*, number 3470 in Lecture Notes in Computer Science, pages 211–225. Springer, June 2005.

[70] George Tsouloupas and Marios D. Dikaiakos. Characterization of computational grid resources using low-level benchmarks. In *Second IEEE International Conference on e-Science and Grid Computing.*, Amsterdam, Netherlands, December 2006.

[71] George Tsouloupas and Marios D. Dikaiakos. Grid Resource Ranking using Low-level Performance Measurements. Technical Report TR-07-02, Dept. of Computer Science, University of Cyprus, February 2007. http://grid.ucy.ac.cy/reports/TR-07-02.pdf.

[72] George Tsouloupas and Marios D. Dikaiakos. Gridbench: A tool for the interactive performance exploration of grid infrastructures. *J. Parallel Distrib. Comput.*, 67(9):1029–1045, 2007.

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

[74] 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.

[75] Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder P. Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the Twenty Second Annual International Symposium on Computer Architecture*, pages 24–37, New York, 1995. ACM Press.