Grid Resource Ranking using Low-level Performance Measurements [⋆]

George Tsouloupas Marios D. Dikaiakos
{georget,mdd}@cs.ucy.ac.cy

Dept. of Computer Science,
University of Cyprus
1678, Nicosia, Cyprus

Abstract. This paper outlines a feasible approach to ranking Grid resources based on an easily obtainable application-specific performance model utilizing low-level performance metrics. First, Grid resources are characterized using low-level performance metrics; Then the performance of a given application is associated to the low-level performance measurements via a *Ranking Function*; Finally, the Ranking Function is used to rank all available resources on the Grid with respect to the specific application at hand. We show that this approach yields accurate results.

1 Introduction

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 [6], 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.

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 to 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 [4]. In such cases, the information published by resource providers and Grid monitoring systems does not provide sufficient detail and accuracy. 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. To provide Grid users with such a *test-driving* functionality, we designed and implemented

[⋆] This work was supported in part by the European Commission through projects EGEE (contract INFSO-RI-031688) and g-Eclipse (contract 034327).

GridBench, 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 [9, 10]. 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.) [11].

In this paper, we present *SiteRank*, a component that we 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: (i) Take as arguments performance metrics derived with the low-level benchmarks of GridBench [11]; the selection of these metrics can be done manually or semi-automatically by the end-user, through the user interface of GridBench. (ii) 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.

In the remainder of this paper, we describe the methodology followed by SiteRank to develop ranking functions. Furthermore, we demonstrate the use of SiteRank in the ranking of the computational resources of EGEE, which is the largest production-quality Grid in operation today [1]. To this end, we 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 modelling. Our results show that SiteRank functions can provide an accurate ranking of EGEE resources, in accordance to the different requirements that each application has. Furthermore, that the careful selection of the low-level metrics used in the linear model is very important for the construction of accurate ranking functions.

The remaining of this paper is organized as follows: Section 2 introduces SiteRank and its ranking methodology. Section 3 describes the use of SiteRank in the ranking of EGEE resources for the two applications of choice: povray and SimpleScalar. We conclude in Section 4.

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.

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. Unfortunately, values quoted by site administrators cannot be relied upon; This is evident in Figure 1 which compares the effectiveness of a **quoted** metric (Figure 1 left) in contrast to a measured metric(Figure 1 right). The charts speak for themselves; Clearly, the **quoted** metric does a very poor job in justifying application performance[1]. It is important to note that this would be *inadequate* even if the quoted values were correct, since application performance depends on much more than just two metrics (see Section 3).

[1] Similar results are obtained with GlueHostBenchmark**SI**00 just as with GlueHostBenchmark**SF**00.

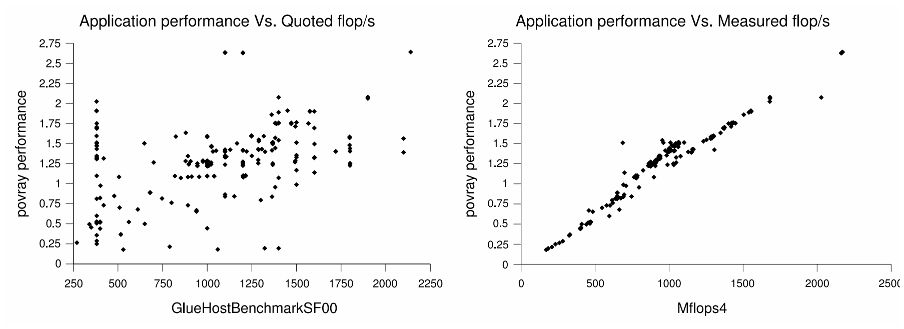


Fig. 1. The Relationship of application performance to **quoted** metrics and **measured** metrics.

Another approach for obtaining a more realistic ranking of resources would be to run the application itself on the resources, collect, analyse and make decisions based on the results. This, of course, would be a rather costly endeavour 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 closing 200, 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 230 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.

2.1 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 2).
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.

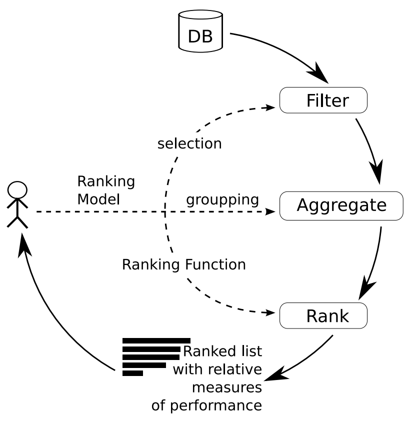


Fig. 2. The ranking process.

Aggregation allows the user to specify grouping of the measurements. The user can specify whether each measurement will count equally, irrespective of which worker-node it was executed on. In this case, the reported metric may possibly be less representative of the resource as a whole because some 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 and in our experiments we used values from a typical 3.0GHz Xeon worker-node. For example, we used the value of 1050.0 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 3, and it is outlined by the following steps:

1. **Sampling**: Obtain low-level performance metrics $\boldsymbol{m}$ 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 $\boldsymbol{m}$ and application performance α, so that $\alpha = R(\boldsymbol{m})$. 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 $\boldsymbol{M}$, and apply the ranking function in order to obtain an estimate of the application performance A_{est} such that $A_{est} = R(\boldsymbol{M})$. Sorting A_{est} produces the *Rank Estimation.*

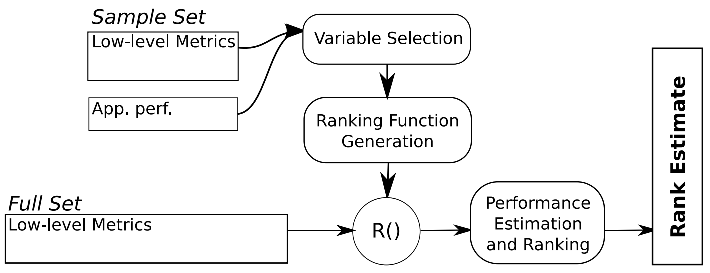


Fig. 3. *Rank Estimate* generation process outline.

Section 3 provides a complete experiment that illustrates this process in greater detail.

2.2 Metrics

Selecting the *right* metrics to characterise the resources is of utmost importance in order to adequately characterize the major computational characteristics that affect application performance. In fact, we consider a *good set of metrics* one that can adequately explain the performance of several distinct applications. In the process of picking the right metrics and the right benchmarks to deliver these metrics, we limited ourselves to freely available tools that we could widely deploy and run. We also aimed at keeping the number of metrics low and we favored well-known metrics. A more detailed discussion on the benchmarks can be found in [11].

Table 1 shows a list of low-level metrics and the associated benchmarks.

Table 1. Metrics and 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

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. For cache performance we needed a metric in the form of a scalar number, that would characterise both the size and the bandwidth of the cache. CacheBench provides a vector of bandwidths at different memory size allocations. Out of this we computed "cumulative bandwidth" for sizes up to 512kb, 1Mb, 2Mb, 4Mb, 8Mb yielding the metrics *c512k*, *c1M*, *c2M*, *c4M* and *c8M* respectively.

3 Experimentation

In this section we 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. We user two serial applications:

– **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.[2]

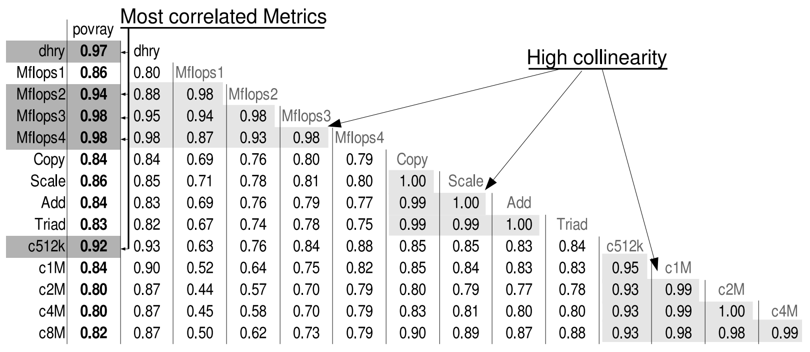
For this experiment, we aimed at having between 2 and 3 measurements from each computational resource. One noteworthy fact is that we could only obtain results for about 160 out of the 230 sites. This was partly due to errors and site unavailability, but also due to exhausted quotas at some resources. We used the GridBench framework to obtain our 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 we did have to wait for a few hours until the results from all the queued jobs were in. We then exported these results into an open-source statistics software package[3] ("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, we 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 to application performance; this is shown in Figure 4. The problem of collinearity must be taken into consideration when narrowing down the selection of metrics. As shown in Figure 4 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 we 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:

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

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

[3] 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 the paper we created using GridBench


Most correlated Metrics
High collinearity

	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

Fig. 4. Correlation Matrix for the *povray* application.

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 our 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, we apply this model on the *full-set* in order estimate the performance of the application:

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

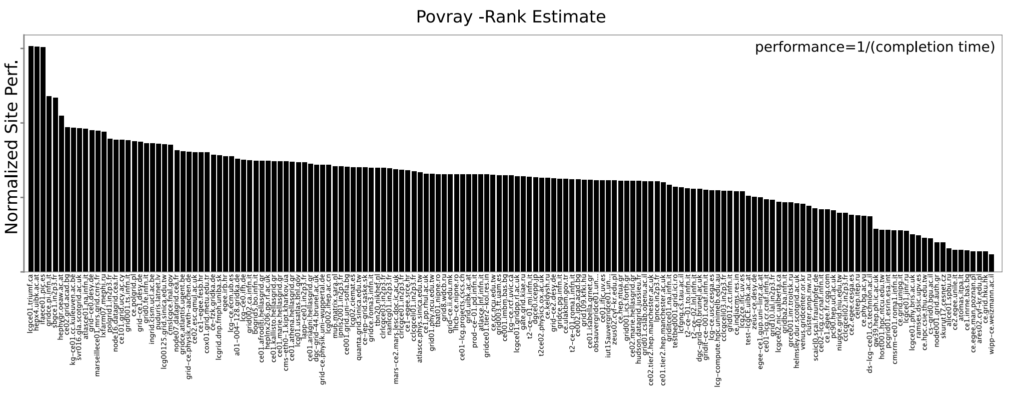


Fig. 5. Rank Estimate for the **povray** application

Ordering the list of resources by A_{povray} gives the *Rank Estimate*. The Rank Estimate is shown in Figure 5.

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 validate the model and not part of the methodology. The measured performance is shown in Figure 6.

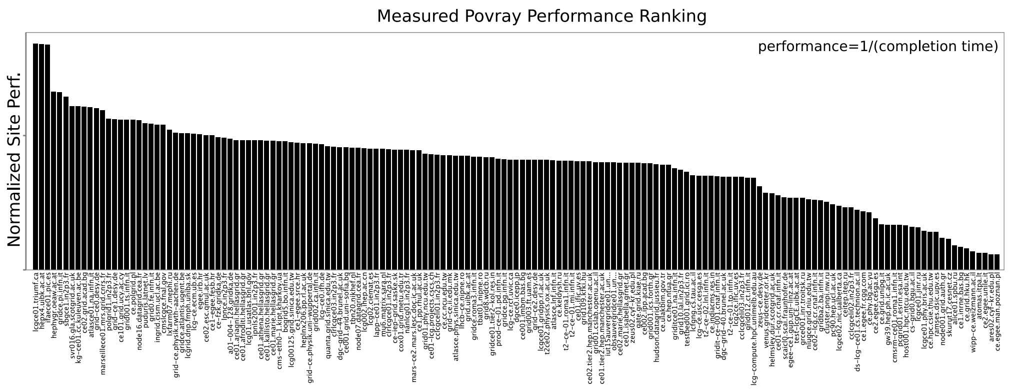


Fig. 6. Measured **povray** performance on 159 resources of the EGEE infrastructure.

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 our 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,the performance of the resources is nearly identical, so the reordering is not very significant. For this reason the authors are inclined to take $\rho = 0.977$ as the more representative measure.

For the second application **sisc** we used the same methodology and 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:

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

The Ranking estimate is given in Figure 7. 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.

4 Conclusions

The work presented here, i.e. Ranking based on derived models of low-level metrics, describes an alternative way of choosing and ranking resources. We 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 analysing 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 verify the "advertised"

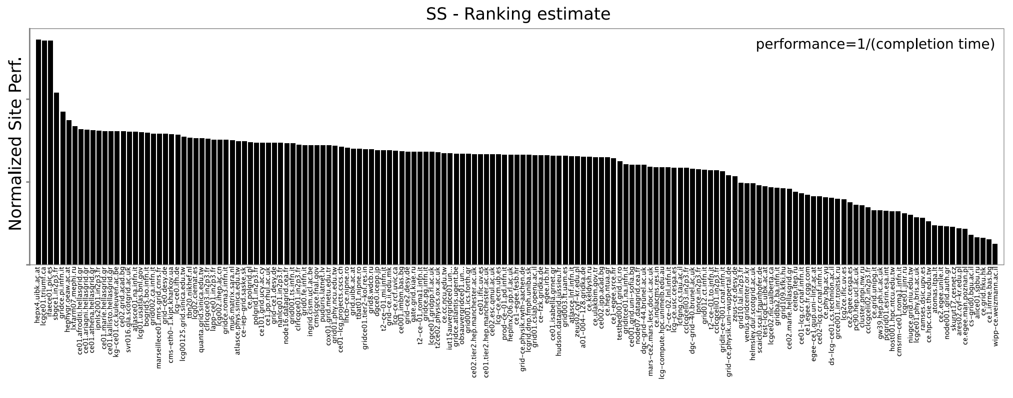


Fig. 7. Rank Estimate for the **sisc** application on the EGEE infrastructure.

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*.

We 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 we have investigated.

Other tools in the general are area of Grid testing and benchmarking include the Grid Assessment Probes [3], DiPerF [5] and the Inca test harness and reporting framework [7]. These are testing/benchmarking frameworks that provide functionality ranging from testing of Grid services to the monitoring of service agreements. In contrast, we focus on user-driven performance exploration and ranking. Benchmarking as a datasource 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; we have also suggested this in our previous work [8].

Choosing the right metrics to collect is of vital importance, as an incomplete set of metrics will yield poor characterization. For example, our initial experiments did not include metrics that characterize the memory cache. While we had been collecting measurements about the cache, the data was in a form that was rather difficult to integrate into a regular function. Also, we had falsely assumed that the cache effects would be largely accounted for in other metrics. The initial results were not at all encouraging; but including the cache metrics, i.e. *c512k*, completely changed the situation. Indicative was the improvement of the ρ rank correlation statistic from approximately $\rho = 0.8$ to $\rho = 0.96$ for the *SimpleScalar* application. This also confirms the importance of a well-sized, fast cache to computational applications.

During our experimentation we have observed 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. It is our conjecture that this heterogeneity of cluster nodes will considerably affect observed application performance. We are currently investigating the inclusion of higher-level metrics (possibly derived from the existing metrics) in order to characterize cluster uniformity/heterogeneity.

Further plans include the investigation of more applications, especially applications that are not CPU/memory bound, in order to evaluate the extent to which the metrics that we collect provide sufficient characterization. Building on the work presented in this article, we plan to investigate the use of ranking functions in scheduling and resource allocation on the Grid in greater detail. We are also working in the direction of automated parameter selection for benchmark tuning. In addition, we plan to investigate the use of micro-benchmarks for automated evaluation of Grid "resource health" and automated detection of degraded performance.

References

1. Enabling Grids for E-SciencE project. http://www.eu-egee.org/.
2. Enis Afgan, Vijay Velusamy, and Purushotham V. Bangalore. Grid resource broker using application benchmarking. In *EGC*, pages 691–701, 2005.
3. 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.
4. 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).
5. 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.
6. Rajesh Raman, Miron Livny, and Marvin H. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138, 1999.
7. 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.
8. 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.
9. 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.
10. 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.
11. George Tsouloupas and Marios D. Dikaiakos. Characterization of computational grid resources using low-level benchmarks. *e-science*, 0:70, 2006.