Using Grid Benchmarks for Dynamic Scheduling of Grid Applications

NAS Technical Report NAS-03-015
October 2003

Michael Frumkin, Robert Hood[+]
NASA Advanced Supercomputing Division
[+]Computer Sciences Corporation
M/S T27A-2, NASA Ames Research Center
Moffett Field, CA 94035-1000
{frumkin,rhood}@nas.nasa.gov

Abstract

Navigation or dynamic scheduling of applications on computational grids can be improved through the use of dynamic characterization of grid resources. We use the NAS Grid Benchmarks to map performance of grid resources into a "GridScape" representing the dynamic state of the grid. Then, we use the GridScape for automatic assignment of these tasks to grid resources. For a moderate number of machines in our experiments, the scalability of the scheduling system was achieved by limiting the mapping and the scheduling overhead to a few percent of the application resource requirements and by distributing the GridScape across the grid. The use of the GridScape in task submission and assignment process guarantees that the application tasks do not congest the grid. We show that our approach reduces the turn-around time of a data mining application and of a flow simulation application by 25-35%.

Keywords: computational grids, benchmarks, performance, navigation, dynamic scheduling.

1 Introduction

An automated scheduling system for matching the application requirements with dynamically changing heterogeneous grid resources would make the grid more efficient and accessible by improving application turn-around times and lowering user effort. A number of projects have been devoted to obtaining, monitoring, and forecasting the state of grid resources, and to the scheduling of individual grid resources and applications. These projects provide tools and services that can be used in a navigation system, however; the problem of congestion-free automatic navigation on the grid remains open. Moreover, crucial components of grid resources, such as the number of currently available processors per grid machine and the efficiency of their use on applications, are usually obtained with a spin test (NWS) or by consulting a batch scheduler (MDS2), neither one sufficient for making good scheduling decisions.

A navigation system considers an application as a set of communicating tasks. It makes scheduling decisions depending on the state of the tasks and the state of grid resources, and takes into account the spatial component of the grid such as latency of the communications. It is more general than scheduling, which is traditionally considered to be rigid planning and assignment of known tasks to a fixed set of resources. Our approach to the automation of navigation is based on automatic characterization of grid resources, extrapolating an application's performance profile to relevant grid resources, and assigning the tasks to the best fitting grid resources. In [8] we have shown that this approach reduces an application benchmark's turnaround time by 20%. Scalability of the system was achieved by using benchmarks that have smaller resource requirements than the application, by distributing the GridScape across the grid machines, and involving it in the task assignment process.

In this paper we extend this approach in two directions. First, we extend the grid mapping capabilities of the navigation system to a grid monitoring system that periodically updates grid performance information. Second, we introduce the **W**aiting-**A**ssigned-**R**unning-**E**xecuted task submission protocol. This results in a congestion-free (cf."Bushel of AppLeS Problem" [2]) navigation system. The architecture of the system is shown in Figure 1. We describe the interaction between servers and navigators in Section 3, and the acquisition and use of the GridScape in Section 4.

We demonstrate the efficiency of our navigation system on a data mining application (the Arithmetic Data Cube (ADC) [9]) and on a flow simulation application (Cart3D [3]), Section 5. The results in Section 6 show that our navigation system reduces ADC turnaround time by 25% and Cart3D by 35%. For our experiments we used a grid of 7 machines containing about 2,000 processors and having peak performance of 1.6 TFLOPS, Table 1.

As an abstraction of grid resources we use a *Grid-*

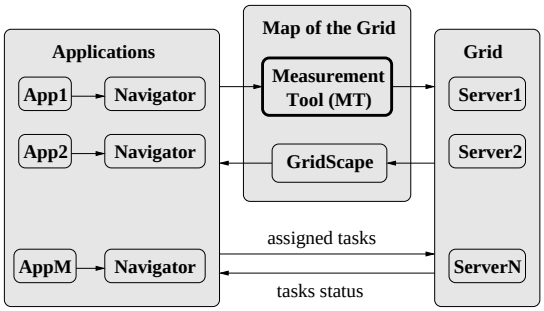


Figure 1. The Architecture of our Navigation System. Each navigator executes the iterations of the Navigational Cycle (see Figure 2). The navigators and servers follow the WARE protocol of Figure 3.

Scape—a map of the grid resources represented by a directed graph, where each grid machine and router is represented as a node of the graph with an attached list of machine resources and a history of their dynamics. Communication links between machines are represented by the arcs of the graph labeled with the observed bandwidth and latency of the links. The initial information about the GridScape is acquired during installation of the benchmark/application servers (Figure 1). The dynamic component of the GridScape is measured by means of the NAS Grid Benchmarks (NGB) during the monitoring, and by requests of the navigators. The benchmarks represent typical activity of Computational Fluid Dynamics applications, and their performance results may be used as a good indicator of the efficiency of using the grid resources for these applications.

We consider grid applications consisting of a number of tasks, cf. [2]. Each task obtains its input data either from its predecessors, or from the launching task. A task sends data either to its successors, or to the reporting task and is ready for execution when data from all its predecessors have been delivered. As soon as the task has finished, it sends the results to its successors. We model such applications by directed acyclic data flow graphs. The graph nodes represent tasks, and the arcs represent data flow between tasks.

2 Related Work

Dynamic scheduling of applications is addressed in the Application Level Scheduler (AppLeS) project [2]. AppLeS's agents use static and dynamic information about both the application and the grid to select viable resource configurations and evaluate their potential performance, then, they interact with the resource management system to implement application tasks. The "Bushel of AppLeS" problem formulated in [2] is a problem of grid congestion caused by applications competing for the best grid resources. Our navigational system follows the WARE protocol of Section 3 and uses a dynamic GridScape, thus preventing applications from overusing grid resources.

A partial map of a grid can be obtained with the Network Weather Service (NWS) [15]. NWS measures end-to-end TCP/IP performance (bandwidth and latency), available CPU fraction, and available non-paged memory. The NWS operates a distributed set of performance sensors from which it gathers readings of the instantaneous conditions. It then uses numerical models to forecast what the conditions will be for a given timeframe. NWS uses the UNIX `uptime` and `vmstat` commands and spin test to obtain the performance of grid machines. NWS employs a push information model.

The Grid Information Protocol [6] (MDS2) provides information about grid resources and protocols to discover this information. The implementation of MDS2 is based on the Lightweight Directory Access Protocol (LDAP), which has significant latency in updating the directory and thus should be considered a repository for static information about grid resources.

The Metascheduler for the Grid [14] is built on the Grid Application Development Software (GrADS) architecture [4]. The Metascheduler has the ability to discover, reserve, and negotiate grid resources to insure that multiple scheduling requests will not cause grid congestion and that all applications make progress. The Metascheduler assumes that numerous types of grid services are working properly.

A resource Brokering Infrastructure for Computational Grids [1] provides a job submission and monitoring mechanism for computational grids. It uses the Jini technology for resource discovery and provides a plug-and-play framework for scheduling algorithms and grid applications. This infrastructure can be used with both Globus and Sun Grid Engine.

An agent-based infrastructure for load balancing of the grid machines that incorporates a performance-driven task scheduler, is reported in [7]. The agents cooperate with each other to balance workload in a global grid environment using service advertisement and discovery mechanisms.

The listed projects give a representative sampling of existing grid scheduling projects. None of the known grid scheduling approaches use benchmark-based characterization of grid resources to build a distributed map of those resources. We believe that the GridScape-based approach better fits the task of informing schedulers about available grid resources than MFLOPS/MB based methods.

3 The Navigational Cycle

In our navigation system, tasks are submitted to the grid resources by the navigators[1], Figure 1. The decision to accept or reject a submission is performed by a server. The navigators run on launch machines, while the servers are on grid machines. For a given application, a navigator performs the following functions:

- Obtains a list of tasks that are ready to be executed;

[1] In some papers on dynamic scheduling, cf. [1] they are called brokers.

- Finds the grid resources that are able to execute these tasks;
- Submits the tasks to the grid resources that will provide the fastest advance of the application;
- Repeats this sequence until all tasks of the application are executed.

In order for a navigator to accomplish these functions, it should be able to understand an application's requirements and know the current state of grid resources.

In a grid environment, the application description should include the software requirements (OS, compiler, libraries, run time systems) and the application performance model (expected execution time, parallel efficiency, memory size, size of I/O data). These requirements and the performance model may be created by a user or may be extracted from the application by a tool [7].

To describe the grid resources, we use the GridScape which lists the capabilities of grid machines and the interconnections between them. When deciding to submit a task, a navigator uses a GridScape to match the task requirements with abilities of grid resources. It compares the task requirements with the current abilities of the grid resources as listed in the GridScape. Then, it estimates the time it will take to execute the task on each grid resource using a relative ranking of the resources by means of the benchmarks. Finally, it submits the task to the resource that minimizes the application turnaround time. In summary, the navigator performs the routine of the *navigational cycle*, Figure 2.

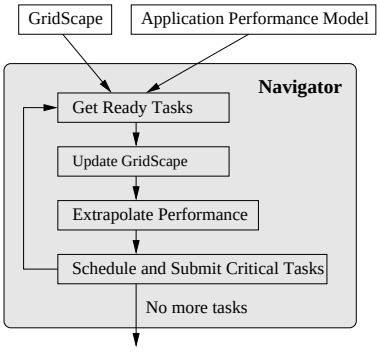


Figure 2. The Navigational Cycle.

Each grid machine runs one server performing the following functions:

- Checks that each task was submitted with use of the current GridScape;
- Accepts the qualified tasks for an execution;
- Returns the unqualified tasks to the navigator;
- Updates the GridScape after accepting a task and executing a task.

The navigators and the servers follow the WARE protocol represented in Figure 3 by passing the tasks through the **W**aiting-**A**ssigned-**R**unning-**E**xecuted sequence of states to insure that a task does not get assigned to a grid resource without knowledge of its current state. This prevents both congestion of grid resources and underuse of the resources.

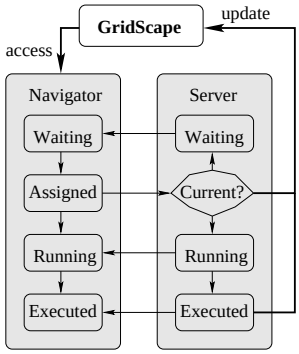


Figure 3. The transition diagram of the WARE protocol. The thin arrows show task state transitions. The thick arrows show update/access of the GridScape.

4 Acquiring the GridScape

The GridScape serves as an abstract description of grid resources and represents the state of the grid. The navigators use it to make submission decisions, while the servers use it to qualify submitted jobs. As a result, the quality of scheduling and the overall efficiency of the navigation system depend on how well the GridScape is synchronized with task submissions, and changes in the state of grid resources.

We use three ways to update the GridScape to achieve a good synchronization. First, each server updates the GridScape when it changes the state of a task to (or from) Running, by subtracting from (or adding to) the GridScape the resources used by the task, Figure 3. Second, a grid monitor periodically updates the GridScape. Finally, each navigator can request an update to the GridScape.

The grid monitor and navigators use the Measurement Tool (MT) of Figure 1, which can be an existing grid performance measurement tool such as `ldapsearch`, `grid-info-search`, MDS2, NWS, or LSF [12]. As an alternative, MT can be built from the architecture-specific commands `traceroute`, `uptime`, `df`, `ps`, `cpustat`, and `vmstat`. In either case we get a characterization of the grid resources in the terms of MFLOPS and MB. Instead, we use the NAS Grid Benchmarks [10] for acquiring the GridScape.

MT can be in "monitor" mode, "probe" mode, or both. In the "monitor" mode, MT runs the NGB across all grid machines with intervals of time that are automatically adjusted

accordingly to the grid volatility, here defined as the gradient of the benchmark's turnaround time. The benchmarking results, including the execution time of the tasks and the communication times between machines, are recorded in the GridScape. A visualization of the GridScape is shown in Figure 4.

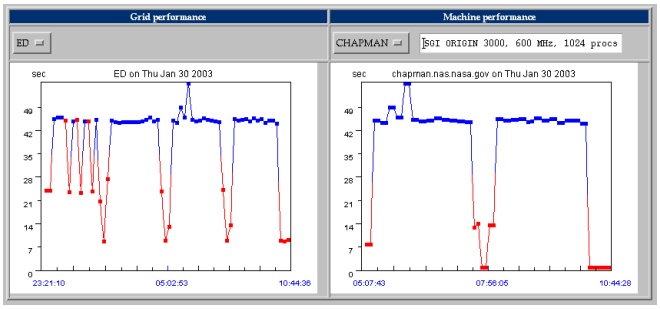


Figure 4. A visualization of the GridScape obtained with the NGB. The dips in ED run on the left indicate that some tasks have failed (in actual graphs the appropriate dots have red color). The graph on the right shows performance of the tasks assigned to `chapman`. The dips indicate that the machine was unavailable.

Each navigator can request the MT to probe a specified subset of grid resources. The work performed by the probe must be limited to a few percent of the work of the task to-be-assigned. The MT grants these requests depending on the frequency of the requests and the grid volatility. This mechanism allows navigators to maintain a GridScape synchronization with the state of the grid resources. In addition, it limits the acquisition overhead to a few percent of the work of the navigated applications and makes the measurement process scalable.

We estimate the latency and the bandwidth of the interconnection links between machines by fitting measured communication times with parameters of the *LogP* model, cf. [11]. On machines under the control of a batch scheduler we use information from the scheduler to get the amount of memory available, otherwise we use `malloc()` library call. Currently, we do not have a portable method to check the amount of available disk space on grid machines.

The grid itself is represented as a directed graph in the GridScape. Each network and router is represented as a node. The arcs connecting the networks and routers are obtained with the `traceroute` tool. On each network we have two MT's running on two different machines to provide fault tolerance. Each MT obtains and maintains a copy of the GridScape for the current network.

5 Using GridScape for Navigation

To demonstrate the ability of our navigation system to reduce application turn-around time, we use a data-intensive application Arithmetic Data Cube (ADC) and a flow simulation package *Cart3D* [3].

5.1 The Arithmetic Data Cube

ADC represents a typical computation of data mining and *On-Line Analytical Processing* systems. The main subject of these applications is a dataset characterized by a number d of dimension attributes and a measure attribute. The dataset consists of tuples $(i_1, \ldots, i_d, c)$. Each dimension attribute i_j assumes values in some range, say in an interval $[1, m_j]$, and c is a cost function (a measure) associated with the tuple. A standard data mining tool is the *Data Cube Operator* (DCO)[5, 13], which computes views of the dataset.

We used an implementation of ADC called AdcView (provided by CrossZ Solutions). The input of *AdcView* is an Arithmetic Dataset [9] having integer dimension and measure attributes. The *AdcView* output for a specified k-element subset of 2^d views is a sorted list of k-tuples. For reduction of I/O, *AdcView* computes a view from a smallest parent. It keeps all information on already calculated views to provide an efficient search of the smallest parent.

For the experiments we chose a 9-attribute dataset of 10^6 tuples and 63 views of this dataset. This results in a 44MB dataset and a 955MB output file containing all views of the dataset; it takes about 291 seconds on the fastest machine of the grid to perform the whole computation.

5.2 The Cart3D

The *Cart3D* package [3] is a production NASA package used for high-fidelity inviscid analysis in conceptual aerodynamic design. It performs CFD analysis on complex geometries. A data flow graph of the *OneraM6* test case is shown in Figure 5. It encapsulates nine executables written in FORTRAN and C. The package includes utilities for geometry import, surface modeling and intersection, mesh generation, and flow simulation and analysis.

The geometry of an aircraft in *Cart3D* is represented as a collection of components, called a *configuration*. The component lists, generated by *net2p3d*, may be fed into *triangulate* which triangulates the configuration. It converts components specified in structured geometry sources into intersection-ready triangulations. The output of *triangulate* is a *Cart3D* component triangulation which is ready for intersection.

The flow simulation starts with mesh generation by *cubes*. *Cubes* produces topologically unstructured, adaptively refined, Cartesian meshes around the configuration. *Reorder* reorders the meshes (Mesh.c3d) produced by *cubes* using a space-filling-curve ordering. *FlowCart* takes these re-ordered meshes and partitions them on-the-fly onto any number of processors. *MgPrep* is the mesh coarsening module which creates coarse meshes from an initial input mesh. These meshes are used in *flowCart* for multigrid convergence acceleration. The flow simulation in *flowCart* is a

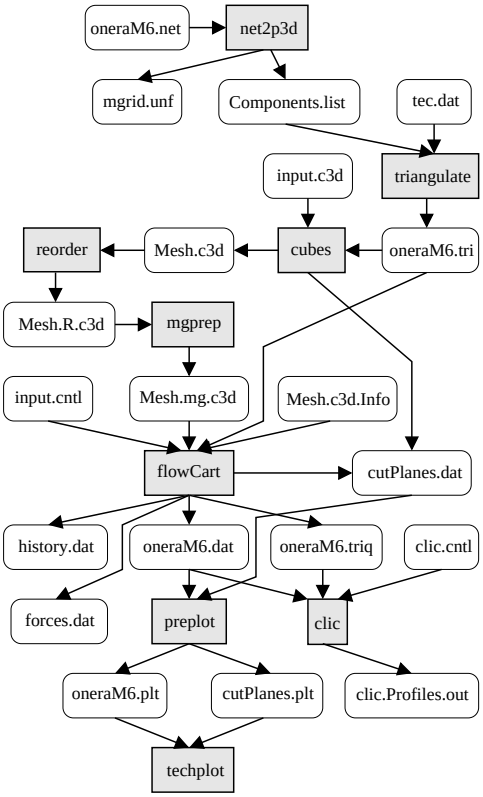


Figure 5. The OneraM6 test case of the Cart3D package. Grey boxes show executables, other boxes show I/O files.

scalable, multilevel solver for the Euler equations governing the inviscid flow of a compressible fluid. The *clic* and *tecplot* programs are used to analyze the simulated flow.

We used the "OneraM6" wing test case. In this case, *cubes* created a mesh with about $3 \cdot 10^5$ cells. *FlowCart* performed 31 iterations on the original mesh, 8 iterations on the first level of refinement, and 11 iterations on the second level of refinment. The initial conditions represented two trajectories: one with increasing mach number, the other with an increasing angle of attack. Each trajectory contained six points; the computation at each point is independent of the others. The overall performance of the job was determined by profiling the execution times of each executable and taking into account that *flowCart* can use up to 16 processors with an efficiency about of 70% on this example.

6 Experiments

The navigation system is implemented in Java. It uses the Java Registry to install task services on grid machines and the Java Remote Method Invocation (RMI) to run the benchmark tasks and to communicate data between them. In addition, it uses the Java Native Interface (JNI) to invoke the *AdcView* and *Cart3D* tasks written in C or FORTRAN. We tested the navigation system on the grid – it's nodes are shown in Table 1. The code of the navigation system was compiled on the U60 and installed on other machines without modifications. The shared library routine used by the Java Native Interface had to be recompiled on O2K and SF880. During our experiments all grid machines had normal production loads.

To launch jobs, we implemented the `jgrun` command, which has an interface similar to `mpirun`. GridScape performance was acquired by using the Java version of ED.S of the NAS grid Benchmarks. We used automatic submission of the servers to the queue, requesting 16 processors on the machines controlled by the PBS batch scheduler.

We tested navigation of ADC by submitting and scheduling the job using the `BalancedLoad` scheduler. Two sets of performance results for 20 test runs each are shown in Figure 6. In the first case, we used the *RoundRobin* scheduler that assigns loads to the machines based on their MFLOPs performance. The second set uses a load balancing scheduler (*BalancedLoad*) which uses the dynamic performance information stored in the GridScape to assign the tasks so they finish simultaneously. The graphs show that the navigation based on a dynamic GridScape reduced ADC turnaround time by 25%.

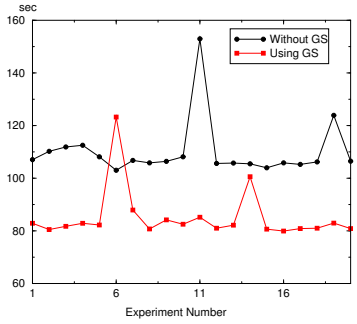


Figure 6. Navigation of AdcView with 9 attributes, 10^6 tuples, and 63 views. We used a load balancing scheduler that tries to finishing all tasks on all grid machines at the same time.

We tested navigation of *Cart3D* (Figure 5) by submitting two trajectories containing six points each. Two sets of performance results for the two trajectories are shown in Figure 7. In the first case we used the *RoundRobin* scheduler. In the second case we used *GreedySched*, which takes into account the current load of the grid machines and assigns a ready task to the machine so that it executes it as quickly as possible. The graphs show that use of a GridScape for navigation reduces *Cart3D* turnaround time by 35%.

7 Conclusions

We have described an architecture and implementation of a system that automates navigation of applications on

Table 1. The the grid machines used in our experiments.

Machine Name	NP	Clock Rate (MHz)	Peak Perf. (GFLOPS)	Memory (GB)	Maker	Architecture	Batch System
U60	2	450	1.8	1	SUN	ULTRA60	-
O2K	32	250	16	8	SGI	Origin2000	-
O2K1	128	250	64	32	SGI	Origin2000	PBS
O3K1	512	400	400	262	SGI	Origin3000	PBS
O3K2	1024	600	1200	256	SGI	Origin3800	PBS
O3K3	256	400	200	98	SGI	Origin3000	PBS
SF880	8	900	14.4	16	SUN	UltraSparc 3	-

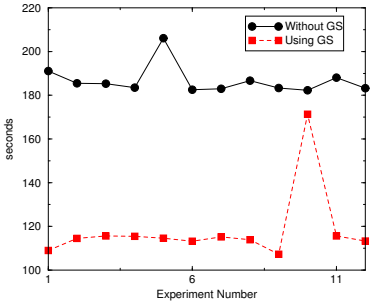


Figure 7. Navigation of Cart3D for two trajectories. The *GreedySched* algorithm assigns a ready task to the machine which would execute it as quickly as possible.

computational grids. Our navigation system automatically acquires a map of the grid and assigns tasks to grid machines. The system chooses resources that provide the fastest advance of application tasks and, as a result, decreases application turnaround time by 25-35% in our experiments. Scalability of the system is achieved by limiting the navigation overhead to a few percent of the application resource requirements. The navigation system follows the "WARE" protocol, which allows our test case to avoid congestion of the grid resources.

Acknowledgments. We want to thank Michael Aftosmis (NASA Ames) for providing the Cart3D package and the OneraM6 test case and Leonid Shabanov (CrossZ Solutions) for providing the source code for AdcView. We appreciate discussions of grid scheduling issues with Piyush Mehrotra (NASA Ames). This project was funded by the NASA CICT program.

References

[1] A. Al-Theneyan, P. Mehrotra, M. Zubair. "A Resource Brokering Infrastructure for Computational Grids." LNCS 2552, pp. 463-473, 2002.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, S. Shao. "Application Level Scheduling on Distributed Heterogeneous Networks." In *Proceedings of Supercomputing 1996*, 1996.

[3] Cart3D. http://people.nas.nasa.gov/∼aftosmis/cart3d/cart3Dhome.html

[4] Grid Application Development Software (GrADS) Project. http://www.hipersoft.rice.edu/grads.

[5] J. Gray, A. Bosworth, A. Layman, and H. Prahesh. *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total.* Microsoft Technical Report, MSR-TR-95-22, 1995.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. "Grid Information Services for Distributed Resource Sharing." *Proceedings of HPDC-10*, 7-9 August, 2001, San-Francisco, CA, pp. 181-194.

[7] J. Cao, D.P. Spooner, S.A. Jarvis, S. Saini, G.R. Nudd. "Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling." *Proceedings of IPDPS'2003*, 22-26 April, 2003, France. p. 49.

[8] M. Frumkin, R. Hood. "Navigation in Grid Space with the NAS Grid Benchmarks" *Proceedings of the 14th IASTED International Conference "Parallel and Distributed Computing and Systems"*, Cambridge, USA, Nov 4-6, 2002, pp. 24-31.

[9] M. Frumkin, L. Shabanov. "Arithmetic Data Cube as a Data Intensive Benchmark" *NAS Technical Report NAS-03-005, http://www.nas.nasa.gov/Research/*

[10] M. Frumkin, Rob F. Van der Wijngaart. "NAS Grid Benchmarks: A Tool for Grid Space Exploration." *Cluster Computing* 5, pp. 247-255, 2002.

[11] T.T. Lee. "Evaluating Communication Performance Measurement Methods for Distributed Systems." *Proceedings of the 14th IASTED International Conference "Parallel and Distributed Computing and Systems"*, Cambridge, USA, Nov 4-6, 2002, pp .45-51.

[12] Platform Computing. "Building Production Grids with Platform Computing." http://www.platform.com /industry/whitepapers.

[13] S. Sarawagi, R. Agrawal, and N. Megiddo. *Discovery-driven Exploration of OLAP Data Cubes.* In Proc. International Conf. of Extending Database Technology (EDBT'98), March 1998, 168-182.

[14] S.S. Vadyhiyar, J.J. Dongarra. "A Metascheduler for the Grid." *Proceedings of HPDC-11*, 23-26 July, 2002, Edinburgh, Scotland, pp. 343-351.

[15] R. Wolski, N.T. Spring, J. Hayes. "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing." *J. Future Generation Computing Systems*, 1999, http://nws.npaci.edu/NWS/.