The MicroGrid: a Scientific Tool for Modeling Computational Grids

H. J. Song X. Liu D. Jakobsen R. Bhagwan X. Zhang K. Taura[*] A. Chien

mgrid@csag.ucsd.edu

Department of Computer Science and Engineering

University of California, San Diego

In Proceedings of SC'2000, Dallas, Texas

Abstract

The complexity and dynamic nature of the Internet (and the emerging Computational Grid) demand that middleware and applications adapt to the changes in configuration and availability of resources. However, to the best of our knowledge there are no simulation tools which support systematic exploration of dynamic Grid software (or Grid resource) behavior.

We describe our vision and initial efforts to build tools to meet these needs. Our MicroGrid simulation tools enable Globus applications to be run in arbitrary virtual grid resource environments, enabling broad experimentation. We describe the design of these tools, and their validation on micro-benchmarks, the NAS parallel benchmarks, and an entire Grid application. These validation experiments show that the MicroGrid can match actual experiments within a few percent (2% to 4%).

1 Introduction

The explosive growth of the Internet and its use in computing, communication, and commerce have made it an integral and critical infrastructure of our society. The Internet's increasing capability have created excitement about a vision for the next-generation Internet which enables seamless integration of computing, storage, and communication into *the Computational Grid*. While demonstrations on large scale test-beds using software infrastructures such as Globus [6] and Legion [7] highlight the potential of computational grids to enable large-scale resource pooling and sharing (compute, communicate, storage, information) in heterogeneous environments, significant challenges in the design of middleware software,

[*]Also affiliated to University of Tokyo, Japan

application software, and even Grid hardware configuration remain. For instance, Internet/Grid environments exhibit extreme heterogeneity of configuration, performance, and reliability. Consequently, software must be flexible and adaptive to achieve either robust or even occasionally good performance. To put it directly, we have no systematic way to study the dynamics of such software to evaluate its effectiveness, robustness, or impact on Grid system stability. We believe that one critical challenge facing the computer systems community is to understand the decidedly non-linear dynamics of Internet/Grid scale systems. The current practice is to perform actual test-bed experiments (at moderate scale) or simplistic simulations that are not validated.

At UCSD, we are pursuing a research agenda which explores how to simulate and model large-scale Grid structures[1] – applications, services, and resource infrastructure[2]. Our objective is to develop a set of simulation tools called the **MicroGrid** that enable systematic design and evaluation of middleware, applications, and network services for the Computational Grid. These tools will provide an environment for scientific and repeatable experimentation. It is our hope that the MicroGrid will catalyze the development of experimental methodologies to robustly extrapolate grid simulation results and rational grid design and management. To achieve these goals, at a minimum, Grid simulation tools must support realistic grid software environments, modeling of a wide variety of resources, and scalable performance.

We have designed an initial set of MicroGrid tools which allow researchers to run Grid applications on virtual Grid resources, allowing the study of complex dynamic behavior. In this paper, we describe our initial design, implementation, and validation of the MicroGrid tools. First, these tools enable the use of Globus applications without change by *virtualizing* the execution environment, providing the illusion of a virtual Grid. Thus, experimentation with a wide variety of existing Grid applications is feasible. Second, these tools manage heterogeneous models in the virtual grid, using a global virtual time model to preserve simulation accuracy. Third, the MicroGrid provides basic resource simulation models for computing, memory, and networking. We describe the implementation of these elements, and validate them at the following three levels.

- **Micro-benchmarks** Individual resource model tests which show these models to be accurate.
- **Parallel benchmarks** Use of the NAS Parallel Benchmark suite which shows the combined accuracy of the MicroGrid for modeling system and application performance.
- **An Application** A complex application program (CACTUS PDE solver) which further validates

[1]The ultimate simulation goals are networks of 10 to 100 million entities.

[2]The development of these tools is part of the NSF funded Grid Application Development Software (GrADS) project, led by Ken Kennedy at Rice University. (http://hipersoft.cs.rice.edu/grads/)

the utility and fidelity of the MicroGrid models.

Experiments with MicroGrids on the NPB Class A data sets ultimately match within 2 to 4% while maintaining high execution efficiency. In addition, we perform an internal validation showing that the execution at each time step within a MicroGrid simulation closely follows that in an actual execution, varying only from 3 to 8%. This internal validation is achieved using the Autopilot [17] tools, and running them within the MicroGrid environment. Finally, experiments with an entire application (CACTUS), match closely as well, within 5 to 7%.

These validations form an important first step in building a set of tools which can support large scale experimentation with software for the Computational Grid. However, while we are pleased with our initial progress, significant challenges remain. To achieve the larger vision of Grid simulation, significant advances in scalability, precision of modeling and network traffic modeling must be achieved.

The remainder of this paper is organized as follows. Section 2 describes the elements of the MicroGrid and their implementation in our prototype. Experiments and validation of our basic system can be found in Section 3. Section 4 discusses related work, and Section 5 summarizes our results and points out new directions.

2 MicroGrid Architecture

2.1 Overview

The basic functionality of the MicroGrid is to allow Grid experimenters to simulate their applications on a virtual Grid environment. The simulation tool should be made flexible to allow accurate experimentation on heterogeneous physical resources, as shown in Figure 1. Our MicroGrid implementation supports Grid applications which use the Globus Grid middleware infrastructure.

The basic architecture of the MicroGrid is shown in Figure 2, and each element in the figure corresponds to one of the major challenges in constructing a high-fidelity virtual Grid. These challenges are as follows:

Virtualization The application must perceive only the virtual Grid resources (host names, networks), independent of the physical resources being utilized.

Global Coordination To provide a coherent global simulation of potentially different numbers of varying virtual resources on heterogeneous physical resources, global coordination of simulation progress is required.

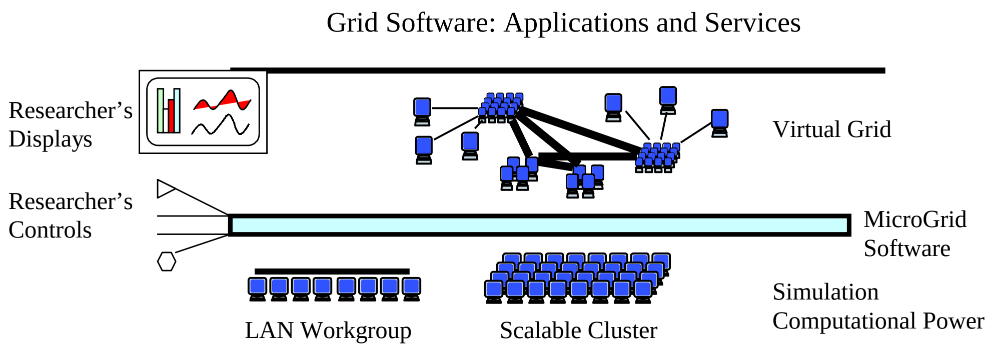


Figure 1: Illustration of a user experimenting with a MicroGrid system to explore a range of virtual resources and executing on a range of physical resources

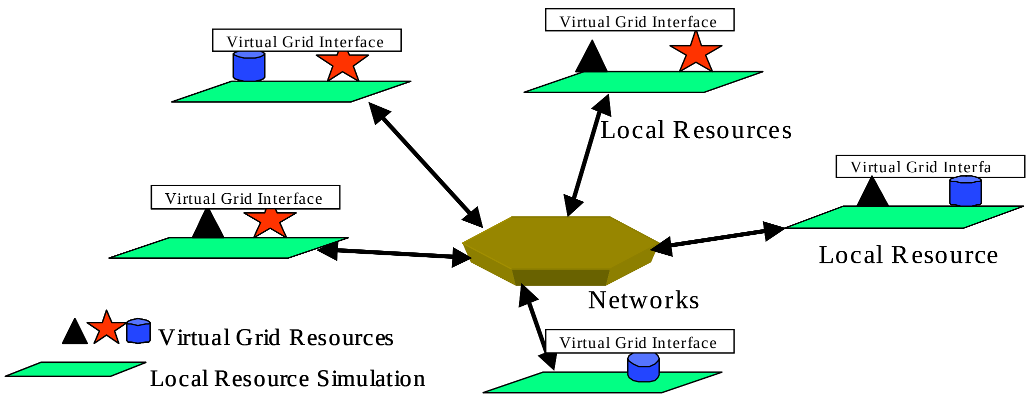


Figure 2: MicroGrid simulator diagram: the local resource simulations provide a virtual Grid resource environment and the network simulator captures the interaction between local virtualized resources.

Resource Simulation Each virtual resource (host, cpu, network, disk, etc.) must be modeled accurately as an element of the overall simulation.

Our approach towards overcoming these challenges is discussed in the following subsections.

2.2 Virtualization

To provide the illusion of a virtual Grid environment, the MicroGrid intercepts all direct uses of resources or information services made by the application. In particular, it is necessary to mediate over all operations which identify resources by name either to use or retrieve information about them.

2.2.1 Virtualizing Resources

In general, the MicroGrid needs to virtualize processing, memory, networks, disks, and any other resources being used in the system. However, since the operating system in modern computer systems effectively virtualizes each of these resources – providing unique namespaces and seamless sharing – the major challenge is to virtualize host identity. In the MicroGrid, each virtual host is mapped to a physical machine using a mapping table from virtual IP address to physical IP address. All relevant library calls are intercepted and mapped from virtual to physical space using this table. These library calls include:

- gethostname()
- bind, send, receive (e.g. socket libraries)
- process creation[3]

By intercepting these calls, a program can run transparently on a virtual host whose hostname and IP address are virtual. The program can only communicate with processes running on other virtual Grid hosts. Many other program actions which utilize resources (such as memory allocation) only name hosts implicitly, and thus do not need to be changed. A user of the MicroGrid will typically be logged in directly on a physical host and submit jobs to a virtual Grid. Thus, the job submission must cross from the physical resources domain into the virtual resources domain. For the Globus middleware, our current solution is to run all gatekeeper, jobmanager and client processes on virtual hosts. Thus jobs are submitted to virtual servers through the virtual Grid resource's gatekeeper. We can run any socket-based application on the virtual Grid as the MicroGrid completely virtualizes the socket interface.

2.2.2 Virtualizing Information Services

Information services are critical for resource discovery and intelligent use of resources in Computational Grids. Since the MicroGrid currently supports Globus, this problem amounts to virtualization of the Globus Grid Information Service (GIS).

Desirable attributes of a virtualized GIS include:

- Compatibility: virtualized information should be used as before by all programs

[3]We currently capture processes created through the Globus resource management mechanisms, but not those created via other mechanisms.

- Identification and Grouping: easy identification and organization of virtual Grid entries should be provided
- Use of identical information servers: there should be no incompatible change in the entries

Our approach achieves all of these attributes by extending the standard GIS LDAP records with fields containing virtualization-specific information. Specifically, we extend records for compute and network resources. Extension by addition ensures subtype compatibility of the extended records (a la Pascal, Modula-3, or C++). The added fields are designed to support easy identification and grouping of the virtual Grid entries (there may be information on many virtual Grids in a single GIS server). Finally, all of these records are placed in the existing GIS servers — no additional servers or daemons are needed. Figure 3 shows an example of the extensions to the basic host and network GIS records:

```
Virtual host MDS records

hn=vm.ucsd.edu, ou=Concurrent Systems Architecture Group, ...
Is_Virtual_Resource=Yes
Configuration_Name=Slow_CPU_Configuration
Mapped_Physical_Resource=csag-226-67.ucsd.edu
CpuSpeed=10
MemorySize=100MBytes

Virtual network MDS records
nn=1.11.11.0, nn=1.11.0.0, ou=Concurrent Systems Architecture Group,
Is_Virtual_Resource=Yes
Configuration_Name=Slow_CPU_Configuration
nwType=LAN
speed=100Mbps 50ms
```

Figure 3: Example Virtual GIS Host and GIS Network Records

2.3 Global Coordination

Achieving a balanced simulation across distributed resources requires global coordination. Based on the desired virtual resources and physical resources employed (CPU capacity and network bandwidth/latency), the virtual time module can determine the *maximum feasible simulation rate.*

The method for calculating the maximum feasible simulation rate is explained here, starting with a few definitions. The simulation rate (*SR*) is defined for each resource type r as the value

$$SR_r = \frac{\text{specification of physical resource } r}{\Sigma \text{ specification of the virtual resources of type } r \text{ mapped to this physical resource}}$$

The specifications are parameters such as CPU speed, network bandwidth, reciprocal of propagation delay, etc[4]. The significance of the *SR* can be explained in the following way. Suppose a process running on the physical resource takes x time to complete. Then, simplistically, it can be said that the same process would take $x * SR$ time to complete on the virtual resource.

The maximum value of *SR* over all the resources represents the *fastest* rate at which the simulation can be run in a functionally correct manner, and is therefore termed the maximum feasible simulation rate. No resource should be allowed to work "faster" than this rate though it can since this would lead to incorrect results. This global coordination mechanism for the rate of simulation over all available resources ensures accurate performance analysis of the processes run on the MicroGrid.

Virtualizing Time Many Grid programs monitor their progress by calling system routines for time (e.g. gettimeofday()), and if their correct behavior is to be maintained, the illusion of virtual time must be provided. Using the chosen simulation rate, we have implemented a virtualization library which returns appropriately adjusted times to the system routines to provide the illusion of a virtual machine at full speed.

2.4 Resource Simulation

Within the MicroGrid simulation, each of the Grid resources must also be simulated accurately, provide real-time performance feedback to the simulation, and be simulated at the rate at which virtual time is allowed to progress. While ultimately many resources may be critical, we initially focus on two resource types — computing and communication.

2.4.1 Computing Resource Simulation

For Grid applications, many of which are compute intensive, accurate modeling of CPU speed is critical. Therefore, our first simulation module focuses on simulating the execution speed of various processors. For each virtual host, the speed of its processor is stored as a GIS attribute on the host record (see Figure 3). Given the virtual host CPU speed, the physical processor speed can be used to calculate the simulation rate, which in this case yields an actual CPU fraction which should be allocated for this Grid simulation. This CPU fraction is then divided across each process on a virtual host. The resulting fractions are then enforced by the *local MicroGrid CPU scheduler*.

[4]The parameters need to be expressed such that a higher value signifies faster execution. For example, if the CPU speed of a physical resource is 100 MIPS and the CPU speed of the virtual resource is 200 MIPS, the *SR* is 0.5.

The local MicroGrid CPU scheduler is a scheduler daemon which uses signals [8] to allocate the local physical CPU capacity to local MicroGrid tasks. The current scheduler uses a round-robin algorithm (see Figure 4), and a quantum of 10 milliseconds as supported by the Linux timesharing scheduler.

A challenge for this scheduler is to achieve *exactly* the desired simulation rate, as against many multimedia systems, where the objective is to achieve a minimum CPU fraction for interactive response.

```
/**********************************************************************/
/* The goal is to achieve myUsedTime == cpu_Fraction*presentTime       */
/**********************************************************************/

currentRunTime = presentTime - startTime;
cputime = cpu_Fraction * runningTime;
if(cputime >= myUsedTime)
do
        startTime = gettimeofday();
        start_process(jobID);
        Sleep(oneQuantum);
        stop_process(jobID);
        stopTime = gettimeofday();
        myUsedTime += (stopTime - startTime);   //Update total process time
end do
```

Figure 4: The Local CPU Scheduler algorithm

2.4.2 Network Simulation

In Grid applications, network behavior is often a critical element of performance [22, 13]. As such, the basic requirements for network simulation include precise modeling of arbitrary network structure and online simulation (conveyance of the communication traffic to the right destination with the right delay). Ultimately, scalability to the extreme (tens of millions of network elements) is an important requirement. These requirements for network simulation provide one of the key challenges for the ultimate viability of the MicroGrid for large-scale modeling.

We have explored VINT/NSE, PDNS, and DaSSF and no existing network simulator fully satisfies the MicroGrid's requirements. For our initial MicroGrid prototype, we have chosen to modify the real-time version VINT/NSE (an online simulator), and have integrated it into the MicroGrid. Due to its better scalability, we are currently exploring the use of DaSSF for our next MicroGrid.

The VINT/NSE simulation system allows definition of an arbitrary network configuration. Our MicroGrid system reads desired network configuration files and inputs a network configuration for NSE according to the virtual network information in the GIS. The network simulator (NSE) is then connected to our virtual communication infrastructure, and thereby mediates all communication. Of course, NSE delivers the communications to each destination according to the network topology at the expected time.

While this system works well, as shown in Section 3, VINT/NSE still presents several significant challenges for use in a MicroGrid system:

- NSE is a best-effort simulator, delivering the messages in close to real-time. If it cannot achieve real-time, it simply gives a warning and continues at the best it can.
- NSE requires an unpredictable amount of computational effort to simulate a network and traffic pattern. As a result, determining a safe global simulation rate is difficult.
- NSE performs detailed simulation, with high overhead.
- NSE is a sequential simulator, and therefore does not scale up to large simulations well.

3 Experiments and Validation

We are executing a series of experiments designed primarily to validate the MicroGrid simulation tools. These experiments can be categorized as follows:

1. Micro-benchmarks of MicroGrid simulation models
2. Benchmark MPI Programs (NAS Parallel Benchmarks)
3. Application programs (CACTUS)

These experiments test successively large parts of the MicroGrid simulation, and with successively more complex application behavior. With a solid validation of the tools, the range of interesting experiments is virtually unbounded.

3.1 Experimental Environment and Methodology

All of the experiments were run on a cluster of 533MHz 21164 Compaq Alpha machines with 1 GB main memory each, and connected by a 100 Mbit Ethernet. The software used is Globus 1.1 run on top of the MicroGrid layer. For each type of benchmark and experiment, we report different metrics.

3.2 Micro-benchmarks

We have performed extensive compute node simulation and network simulation experiments. We use some micro-benchmarks to validate the accuracy of the models.

3.2.1 Memory Capacity Modeling

The following test is performed with the goal of verifying the scheduler's ability to enforce the memory limitation as specified when assigning a process to a virtual machine.

A MicroGrid process is initialized on the scheduler and a process, and that process which constantly allocates memory until it generates an out of memory error. The test is repeated with various memory limits for the process and the maximum amount of memory successfully allocated is logged for each repeat of the test. Figure 5 shows the result of the tests where memory limitations from 1KB to 1MB are specified.

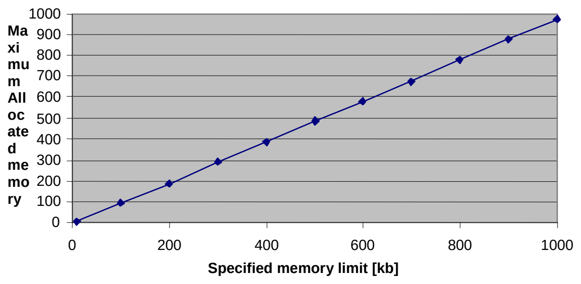


Figure 5: Memory microbenchmark

As shown in Figure 5, there is a clear linear correlation between the memory limit and the amount of memory accessible by the process. In each case, the process could allocate about 1KB less then the specified memory limitation. This is due to memory overhead for the process.

3.2.2 Processor Speed Modeling

This test focuses on how precisely the MicroGrid processor scheduler maintains the processing model in the presence of CPU competition on the physical machine. This test is done in several steps: For each step a virtual machine of a given speed is created on the scheduler, a process performing a fixed computation is scheduled on that virtual machine, and after the process completes, the virtual machine is deleted. This test is repeated with various speeds for the virtual machines and with competition in form of CPU intensive and IO intensive processes running in parallel with the MicroGrid scheduler.

The process scheduled on the virtual machine is a reference process performing a fixed CPU-intensive computation (i.e. the process is never blocked) work and its execution speed running as the only process on the physical CPU.

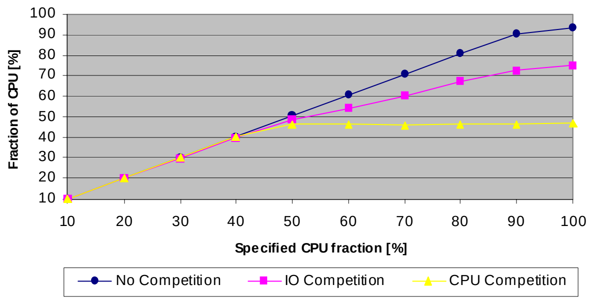


Figure 6: Processor microbenchmark

The three versions of the processor microbenchmark are:

- No Competition: This is used as reference. During this test the scheduler is running as the only process on the CPU.
- CPU Competition: In parallel with the scheduler daemon a computationally intense process is executing. The computationally-intensive process does floating-point divisions continuously.
- IO Competition: In parallel with the scheduler daemon, an IO intensive process is executing. The IO intensive process used during this test continuously flushes a 1MB buffer to disk.

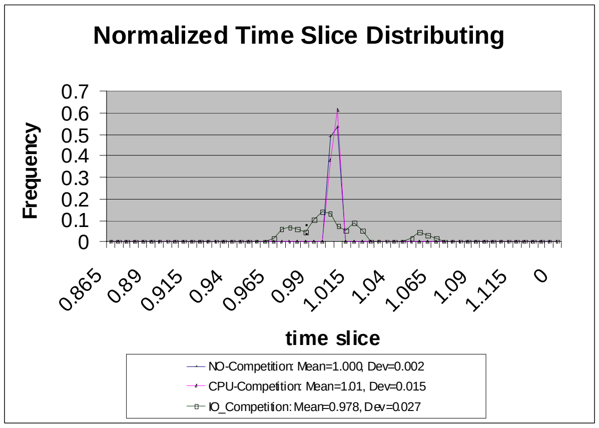


Figure 7: Distribution of Quanta sizes, normalized to a mean of unity

From Figure 6 is it seen that the MicroGrid scheduler is able to schedule the reference process according to the specified speed for the virtual machine across a wide range of speeds up to 95% of the total CPU capacity. Both the operating system and the scheduler's CPU usage form this upper boundary. For the IO competition and CPU competition tests, the results show that above a specified CPU speed of 40%, the virtual machine does not deliver the specified CPU fraction. This inadequacy is caused by the characteristics of the time-sharing scheduling policy in Linux.

The following test is performed with the goal of characterizing the stability of the scheduled quanta size. We modified the MicroGrid scheduler daemon to collect performance data, logging the time slice size assigned to each process. The test consists of three sessions, producing about 9000 samples, corresponding to about 90 seconds of test. The process that actually runs on the MicroGrid during this test is an inactive process that constantly sleeps.

Figure 7 shows that with no competition, when the MicroGrid scheduler is running as the only process on the CPU, the CPU time allocated to a process is almost the same as the virtual time specified by the user. This test also shows that computational intensive processes running in parallel with the MicroGrid scheduler did not have any influence on the performance of the scheduler. Extreme IO competition did affect resulting quanta size, but not severely.

3.2.3 NSE Network Modeling

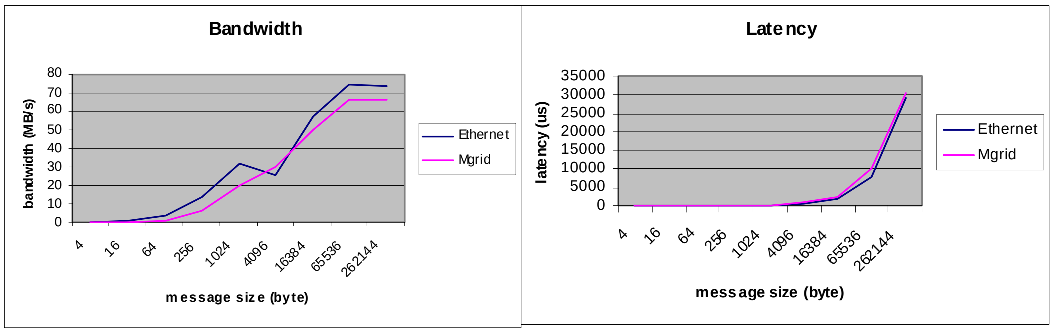


Figure 8: NSE network modeling

In this experiment we use MPI bandwidth and latency benchmarks to test the accuracy the simulated network performance. These benchmarks run on two virtual nodes connected by virtual networks (a 100Mb Ethernet). The results are compared with outputs from the real system. We can see that simulated network has the similar characteristics with the real system.

3.3 Benchmarks

We used the NPB (NAS Parallel Benchmarks) 2.3 applications [19] to perform a more extensive validation of the MicroGrid tools, encompassing not only the compute node and network models, but also their integration into a unified simulation.

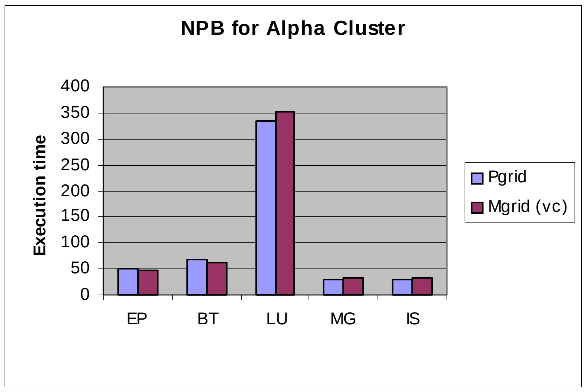
3.4 Total Run Time

3.4.1 Machine Configurations

Our first goal was to validate the MicroGrid simulator for the NAS Parallel Benchmarks across a range of machine configurations. To do so, we ran all of the benchmarks across a range of processor and network configurations matching those for which performance data is available on the NPB web site (http://www.nas.nasa.gov/Software/NPB/). We studied the virtual Grid configurations shown in Table 9.

Name	# Procs	Type Procs	Network	Compiler
Alpha Cluster	4	DEC21164, 533 MHz	100Mb Ethernet	GNU Fortran
HPVM	4	PentiumII, 300 MHz	1.2Gb Myrinet	Digital Fortran V5.0

Figure 9: Virtual Grid Configurations Studied



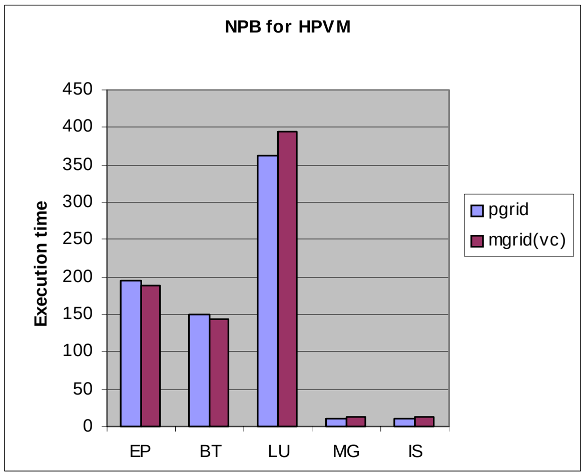


Figure 10: NPB benchmark performance (Class A), comparing physical machine runs with equivalent MicroGrid simulations

A bar graph comparing the actual run times with those simulated on the MicroGrid is shown in Figure 10. As shown in the figure, the MicroGrid matches IS, LU, and MG within 2%. For EP and BT, the match is slightly worse, but still quite good, within 4%. Because the simulation tracks the performance differences across configurations fairly well, we consider the MicroGrid validated for these applications.

In searching to explain the slight mismatches in performance, microscopic analysis of these programs behavior revealed that the MicroGrid's current 10 millisecond scheduling quantum was introducing modeling error, when processes synchronized are shorter intervals.

To test this hypothesis, we explored a range of quanta sizes for each of the benchmark applications, and used the Class S (small) data sets, which would exacerbate the inaccuracies introduced by the quanta size. The results from these experiments are shown in Figure 11, and demonstrate that varying the time quanta can significantly improve the modeling precision.

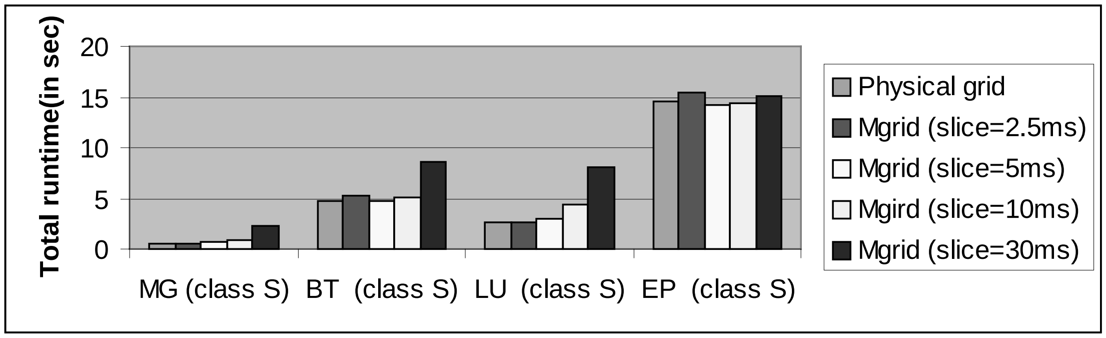


Figure 11: The effect of varying the scheduling quanta length on MicroGrid modeling accuracy (NPB Class S Benchmarks)

For the MG, BT, LU, and EP benchmarks on the Class S benchmarks, the best quanta sizes for matching are 2.5ms, 5ms, 2.5ms, and 10ms respectively. Without exception, the benchmarks that synchronize frequently match better with shorter time quanta. The matches with the best time quanta in each case are quite close, with differences of 12%, 0.6%, 0.4%, and 1.3% respectively. Clearly, for codes that communication and synchronize frequently, scheduling with smaller time quanta is an important feature of future MicroGrids.

3.4.2 Additional Experiments

The MicroGrid can also be used to extrapolate likely performance on systems not directly available, or those of the future. For example, a MicroGrid simulation can be used to explore the future implications of technology scaling at different rates – network, memory, and processor speed – for application or system software. For example, Figure 12 explores the impact of faster processors, holding network performance constant (1 Mbps and 50 millisecond latency). For these benchmarks, significant speedups can be achieved solely based on increases in processor speed.

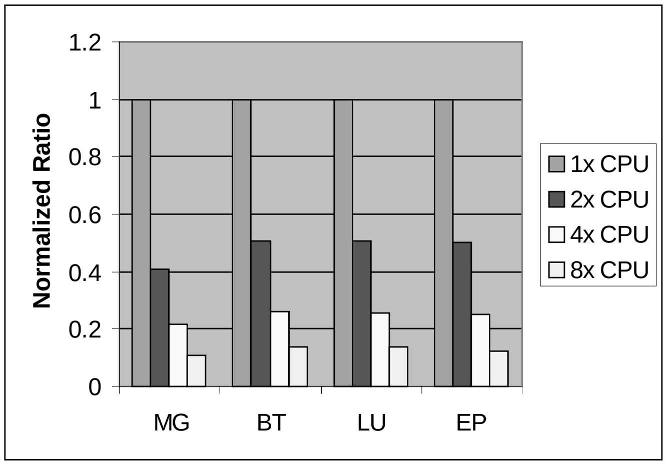


Figure 12: Total run times varying only the virtual cpu.

The MicroGrid can also be used to study systems with complex network topologies and structures (as will be common in many Grid systems). For example, a MicroGrid simulation can model the effect of spreading an application over a wide-area network testbed such as the United States National Science Foundation's vBNS testbed [21] (see Figure 13). We study the possibility of executing the NAS parallel benchmarks over a fictional vBNS distributed cluster testbed, varying the speed of the WAN network links. Our experiment uses 4-process NPB jobs with two processes at UCSD (in CSE department LAN) and two processes at UIUC (on the CS department LAN). Thus, the path from one cluster to another traverses LAN, OC3, and OC12 links as well as several routers (finer detail such as switches and hubs are not modelled). In the experiment, we vary a bottlneck link from 10Mb/s to OC to OC12, producing the NPB benchmark performance shown in Figure 14.

The results (see 14) show that the performance of the NAS parallel benchmarks distributed over a wide-area coupled cluster is only mildly sensitive to network bandwidth. With the exception of EP, the latency effects dominate, producing poor performance for nearly all network bandwidths. This confirms the commonly held perspective that Grid applications need be latency tolerant to achieve good performance.

The MicroGrid can be run at a variety of actual speeds, yet yield identical results in virtual Grid time. This capability is useful in sharing simulation resources, or in interfacing the MicroGrid simulations with external resources for which we may not be able to control rates precisely. For example, slowing the processor and network simulations can be used to make a slow disk seem much faster, or an external sensor data rate much higher (to model a future higher speed sensor). The results in Figure 15 show how the fidelity of simulation varies as the speed of simulation varied over an order of magnitude.

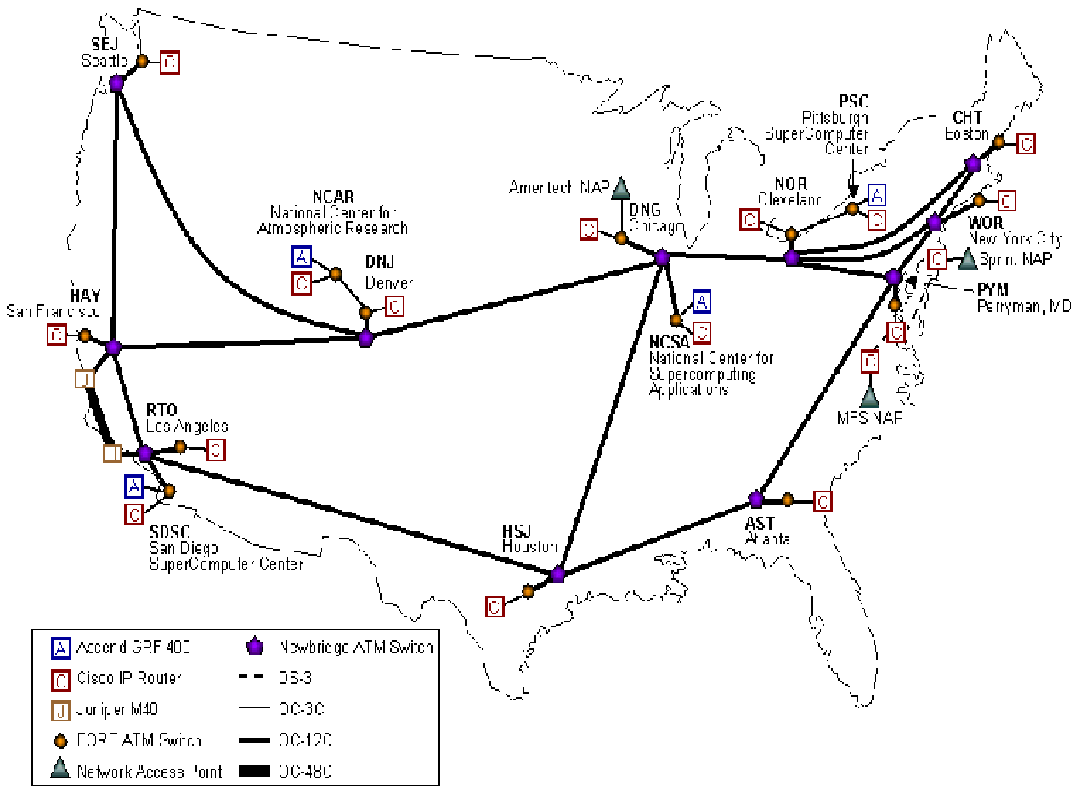


Figure 13: Our fictional distributed cluster testbed uses the vBNS and models the network using the MicroGrid simulator.

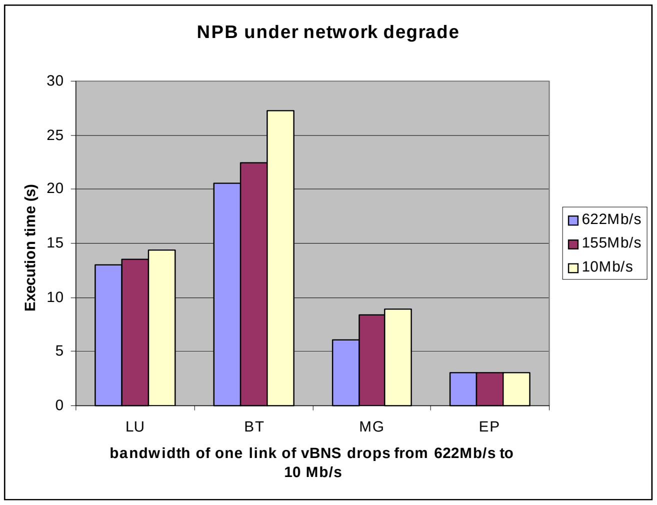


Figure 14: Performance of NPB over the vBNS distributed cluster testbed, varying the network speed of the major WAN links.

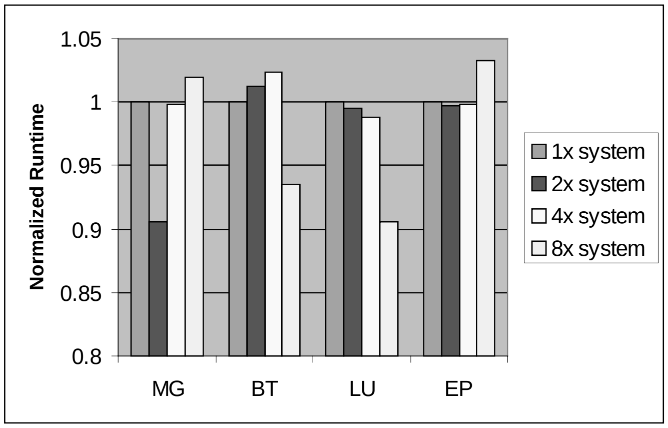


Figure 15: Total run times varying emulation rates.

3.5 Applications

The largest validation tests must include full-blown application programs. We have studied and validated the MicroGrid simulation on the CACTUS code – http://www.cactuscode.org/ – a flexible parallel PDE solver.

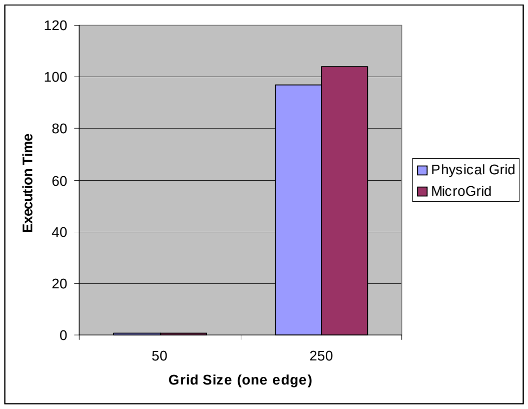
Cactus [1] is an open source problem solving environment designed for scientists and engineers. It originated in the academic research community, where it was developed and used over many years by a large international collaboration of physicists and computational scientists.

To validate the MicroGrid, we again use a model of DEC Alpha machines in a cluster. We present run times for CACTUS running WaveToy on this testbed and compare to runs on the MicroGrid with appropriate processor and network speed speed settings. These results (see Figure 16), show excellent match, within 5 to 7%. Thus, a micro-benchmark, full blown benchmark suite, and entire application program validation has been done for the MicroGrid tools, demonstrating its accurate modeling capabilities for full program runs.

3.6 Internal Performance

A more demanding validation of the MicroGrid matches the internal application behavior and timing of each run with that on an actual machine. To make this comparison, we used identical performance tools (the Autopilot system [17]) and monitored the NAS Parallel Benchmarks. Autopilot enables the tracking of the values of certain types of program variables over the execution of the program. The resulting data, normalized over execution time of the programs are shown in Figure 17.

The graph shows the the changing value of a periodic function of counter variables in EP, BT and MG



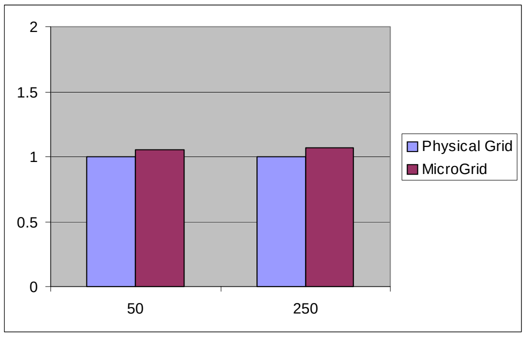


Figure 16: CACTUS runs on a physical cluster and on the MicroGrid tools modeling that same cluster performance.

with time for class A data sizes for the Alpha cluster (using 100% CPU time) and the MicroGrid (using 4% CPU time). The simulation rate is therefore 0.04. The x-axis shows the time, with one sample of the variables being made every 1 second for the Alpha cluster, and every 25 seconds for the MicroGrid to take into account the simulation rate. We see that the traces for both the physical system and the simulation follow the same structure, and match fairly well in time. We calculated the "skew" int the graphs for each benchmark as the root mean square percentage difference recorded at each sample time. This value was found to be 3.08% for EP, 2.02% for BT and 8.33% for MG. The closeness of the internal matching provides stronger evidence that the MicroGrid is precisely matched the modeled Grid system's behavior.

4 Related Work

While there are many related efforts, we know of no other Grid, distributed system, or parallel system modeling efforts that seek this level of modeling of Computational Grids – validated, general purpose direct simulation.

Early papers on the Globus MDS described a feature called "resource images" which are similar to our virtual Grid entries. However, to our knowledge, these images were never used. The most closely related project is the Ninf/Bricks project [20] which has focused on evaluating scheduling strategies based on synthetic models of applications and innovative network traffic models. However, to date Bricks has not used Grid applications as a direct driver.

There are a wide variety of simulation tools for parallel systems; however, they focus primarily on architectural simulations [18, 16, 4, 14] and program debugging [12, 5]. These tools typically do not

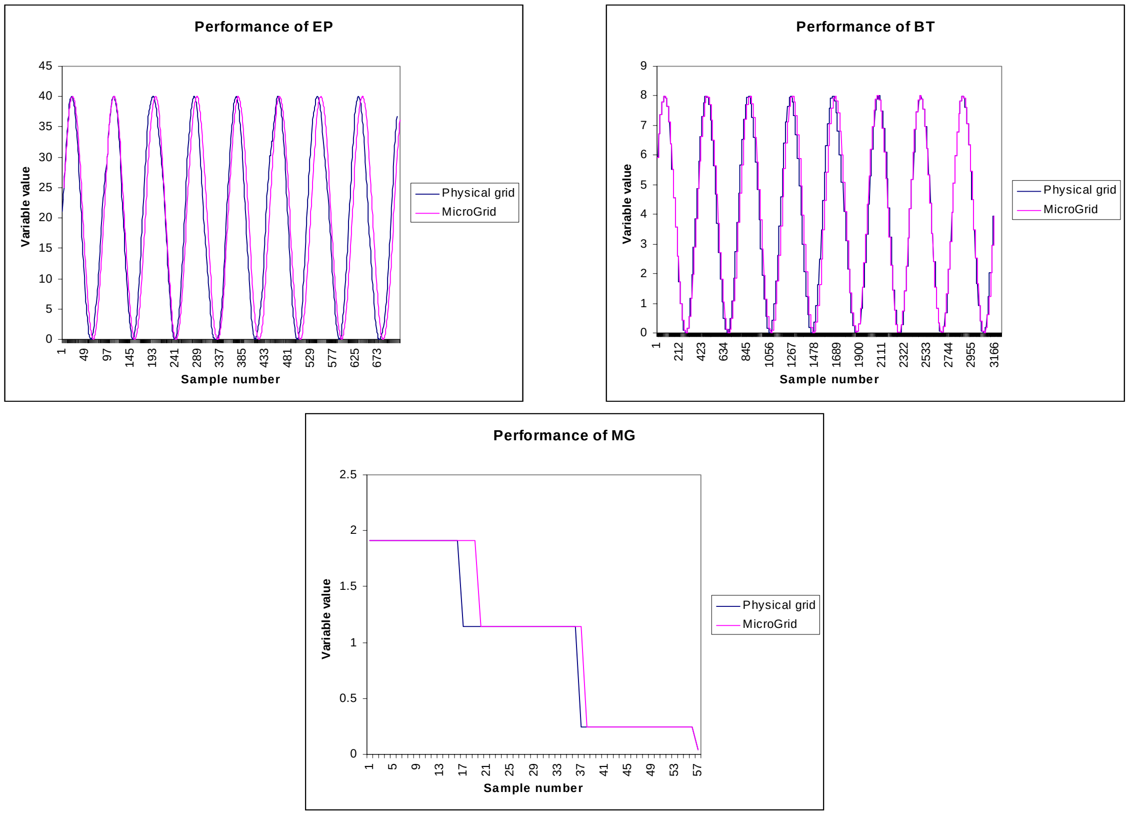


Figure 17: Comparing Autopilot data from a physical system and a MicroGrid simulation

model system heterogeneity, irregular network structure, a complex Grid software environment, and the traffic modeling required by the network environments.

There a wide variety of network simulation tools, including those for parallel systems [11, 15, 10] and those for Internet style networks [2, 3, 9]. The parallel system tools typically do not model the detailed structure in complex heterogeneous network environments. Those for Internet systems typically do not support live traffic simulation. Also, since Internet workloads are typically large numbers of WWW users on low-bandwidth links, these simulators do not focus on scaling with high intensity workloads and large dynamic range (six orders of magnitude) in single user network demand.

5 Summary and Future Work

We are pursuing a research agenda which explores how to simulate and model large-scale Grid structures. To this end, we have developed a set of simulation tools called the **MicroGrid** that enable experimenters to run arbitrary Grid applications on arbitrary virtual Grid resources, allowing the study of complex dynamic behavior. We describe our initial design, implementation, and validation of the MicroGrid tools on micro-benchmarks, a well-known parallel benchmark suites, the NAS parallel benchmarks, and a full-blown application program, the CACTUS system for parallel partial differential equation solvers. These studies show that the MicroGrid approach is feasible and the basic implementation is validated. In addition, we perform an internal validation showing that the execution at each time step within a MicroGrid simulation closely follows that in an actual execution. This internal validation is achieved using the Autopilot tools, and running them within the MicroGrid environment.

While we have made tangible progress, significant challenges remain. For some applications, sing smaller scheduling quanta, enabled by a real-time scheduling subsystem can improve the fidelity of our simulations. We are pursuing the development of a new scheduler based on real-time priorities. In the near term, we plan to support scaling to dozens of machines, dynamic mapping of virtual resources, and dynamic virtual time. In the longer term, questions of extreme scalability – how to get to 100 million simulated nodes, exploring a range of simulation speed and fidelity, understanding how to extrapolate from a small set of Grid simulations to a much broader space of network environment and application behavior. We will also pursue the use of the MicroGrid tools with a much larger range of applications, exploiting the growing range of Globus applications that are becoming available. We also invite the participation of other research groups to extend and enhance the MicroGrid simulation infrastructure.

Acknowledgements

The research described is supported in part by DARPA thru the US Air Force Research Laboratory Contract F30602-99-1-0534. It is also supported by NSF EIA-99-75020 and supported in part by funds from the NSF Partnerships for Advanced Computational Infrastructure – the Alliance (NCSA) and NPACI. Support from Microsoft, Hewlett-Packard, Myricom Corporation, Intel Corporation, and Packet Engines is also gratefully acknowledged.

References

[1] Gabrielle Allen, Tom Goodale, Gerd Lanfermann, Thomas Radke, and Edward Seidel. The cactus code, a problem solving environment for the grid. 2000.

[2] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, , and Daniel Zappala. Improving sim-

ulation for network research. Technical Report 99-702, University of Southern California, 1999. http://netweb.usc.edu/vint/.

[3] James Cowie, Hongbo Liu, Jason Liu, David Nicol, and Andy Ogielski. Towards realistic million-node internet simulations. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999. http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/.

[4] Gary Tyson Edward Tam, Jude Rivers and Edward S. Davidson. mlcache: A flexible multi-lateral cache simulator. Technical Report CSE-TR-363-98, University of Michigan Department of Electrical Engineering and Computer Science, 98.

[5] Mingdong Feng and Charles E. Leiserson. Efficient detection of determinacy races in cilk programs. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–11, 1997.

[6] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications*, 1997. http://www.globus.org.

[7] A. S. Grimshaw, W. A. Wulf, and the Legion team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 1997. http://legion.virginia.edu.

[8] Chih han Lin, Hao hua Chu, and Klara Nahrstedt. A soft real-time scheduling server on the windows nt. In *Proceedings of the Second USENIX Windows NT Symposium*, 1998.

[9] David M. Nicol James Cowie and Andy T. Ogielski. Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50, 1999. http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/.

[10] Lawrence Snyder Kevin Bolding, Melanie Fulgham. The case for chaotic adaptive routing. Technical Report UW-CSE-94-02-04, University of Washington, 1994.

[11] Jae Hoon. Kim and Andrew. Chien. Network performance under bimodal traffic loads. *Journal of Parallel and Distributed Computing*, 28(1), 1995.

[12] Lam/mpi parallel computing. Home Page, at http://mpi.nd.edu/lam/.

[13] Nancy Miller and Peter Steenkiste. Collecting network status information for network-aware applications. In *Infocom'00*, 2000. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/cmcl/www/remulac/index.html.

[14] Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve. RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. In *Proceedings of the Third Workshop on Computer Architecture Education*, February 1997. http://www.ece.rice.edu/˜rsim/.

[15] Sundeep Prakash. *Performance Prediction of Parallel Programs.* PhD thesis, University of California Los Angels, 1996. http://pcl.cs.ucla.edu/projects/sesame/.

[16] Steven K. Reinhardt, Mark D. Hill, James R. Larus, Alvin R. Lebeck, James C. Lewis, and David A. Wood. The wisconsin wind tunnel: Virtual prototyping of parallel computers. In *Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 48–60, 1993. http://www.cs.wisc.edu/˜wwt/.

[17] Randy L. Ribler, Jeffrey S. Vetter, Huseyin Simitci, , and Daniel A. Reed. Autopilot: Adaptive control of distributed applications. In *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*, 1998. http://www-pablo.cs.uiuc.edu/Project/Autopilot/AutopilotOverview.htm.

[18] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, , and Anoop Gupta. Complete computer simulation: The simos approach. *IEEE Parallel and Distributed Technology*, 1995. http://simos.stanford.edu/.

[19] William Saphir, Rob Van der Wijngaart, Alex Woo, and Maurice Yarrow. New implementation and results for the nas parallel benchmarks 2. NASA Ames Research Center. http://www.nas.nasa.gov/Software/NPB/.

[20] Atsuko Takefusa, Satoshi Matsuoka, Hidemoto Nakada, Kento Aida, and Umpei Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pages 97–104, 1999. http://ninf.etl.go.jp/.

[21] the United States National Science Foundations vBNS. Very high performance backbone network service. MCI WorldCom, Inc. http://www.vnbs.net/.

[22] Rich Wolski. Dynamically forecasting network performance using the network weather service. *Journal of Cluster Computing*, 1998. http://nws.npaci.edu/NWS/.