Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics

Xin Liu, Huaxia Xia, Andrew A. Chien

Department of Computer Science and Engineering
and Center for Networked Systems
University of California, San Diego
{xinliu, hxia, achien}@cs.ucsd.edu

Abstract-- Large-scale grids that aggregate and share resources over wide-area networks present major challenges in understanding dynamic application and resource behavior for performance, stability, and reliability. Accurate study of the dynamic behavior of applications, middleware, resources, and networks depends on coordinated and accurate modeling of all four of these elements simultaneously.

We have designed and implemented a tool called the MicroGrid which enables accurate and comprehensive study of the dynamic interaction of applications, middleware, resource, and networks. The MicroGrid creates a virtual grid environment – accurately modeling networks, resources, the information services (resource and network metadata) transparently. Thus, the MicroGrid enables users, grid researchers, or grid operators to study arbitrary collections of resources and networks. The MicroGrid includes the MaSSF online network simulator which provides packet-level accurate, but scalable network modeling.

We present experimental results with applications which validate the implementation of the MicroGrid, showing that it not only runs real grid applications and middleware, but that it accurately models both their and underlying resource and network behavior. We also study a range of techniques for scaling a critical part of the online network simulator to the simulation of large networks. These techniques employ a sophisticated graph partitioner, and a range of edge and node weighting schemes exploiting a range of static network and dynamic application information. The best of these, profile-driven placement, scales well to online simulation of large networks of 6,000 nodes using 24 simulation engine nodes.

1 INTRODUCTION

Improvements in networking and middleware technology are enabling large-scale grids that aggregate and share resources over wide-area networks to support applications at unprecedented levels of scale and performance. Because the aggregation and sharing of resources in grids presumes dynamic application and resource configuration, grids present significant new challenges for performance modeling and design of adaptive middleware and adaptive applications. Further, as grid and internet applications increasingly couple end system and network behavior – and service quality depends on end-to-end performance – accurate study of the dynamic behavior of applications, middleware, resources, and networks depends on coordinated and accurate modeling of all four of these elements simultaneously.

A number of grid middleware projects have been developed to enable access to grid resources, such as Globus [1], Legion [2], Condor [3], NetSolve [4], and GrADS [5]. However, these middleware layers today are providing only basic mechanisms for execution in a grid environment and do not provide solutions which ensure resource stability, application stability, predictable behavior, guaranteed quality of service, etc., in open, shared, efficiently utilized grid environments. All these middleware systems themselves and applications that use them are developed and painstakingly evaluated in a few grid environments before being released for early use. Only after some time, and extensive ad hoc testing and use, are their dynamic behaviors considered stable. In fact, the breadth of understanding their dynamic properties in novel resource environments or in the presence of novel competitive resource demands is minimal. In short, understanding the dynamic behavior of grid environments (applications, middleware, resources, and networks) remains an open research challenge, and the subsequent engineering need to ensure resource stability, application performance stability, application quality of service, and also efficient resource utilization remains daunting. It is this problem which motivates the efforts described in this paper.

The goal of the MicroGrid project is to develop and implement simulation tools that enable scientific and systematic of the dynamic behavior of applications, middleware, resources, and networks. These tools will provide a vehicle for observable, repeatable study and systematic exploration of design spaces for a wealth of design problems for applications and middleware, exploration of rare or extreme situations, and rational choices in application deployment and grid resource and network design.

We have designed and implemented a tool called the MicroGrid which enables accurate and comprehensive study of the dynamic interaction of applications, middleware, resource, and networks. The MicroGrid creates a virtual grid environment – accurately modeling networks, resources, the information services (resource and network metadata). Thus, the MicroGrid enables users, grid researchers, or grid operators to study arbitrary collections of resources and networks. Further, the MicroGrid virtualizes transparently, allowing the direct study of complex applications or middleware whose internal dynamics are difficult to model accurately. That is, real application software and middleware

can be used unchanged and executed on arbitrary virtual grid structures. In short, *the MicroGrid provides a virtual grid infrastructure that enables scientific and systematic experimentation with dynamic resource management techniques and adaptive applications by supporting controllable, repeatable, observable experiments.*

The MicroGrid complements experimentation with actual grid testbeds because the MicroGrid can be used to explore a wide variety of grid resource configurations and scenarios (such as catastrophic failure), which may not be possible to exhibit in the actual resources. The MicroGrid provides reduced setup effort for simulation and increases the observability of application behavior. We describe the design of the MicroGrid includes the following key innovations:

- Because network performance and protocol behavior is critical in many applications, we have paid careful attention to detailed network modeling, so the MicroGrid employs packet-level online simulation.
- Studying large applications, compute, and storage resources with high fidelity requires efficient and scalable execution performance. The MicroGrid employs parallel, direct execution of applications.
- Studying large grid networks with high speed communication requires an efficient parallel online simulation. The MicroGrid includes the MaSSF system, a scalable online network simulator.
- Because dynamic behavior of applications and middleware is difficult to model accurately, the MicroGrid system employs direct execution, enabling use of the actual implementations as models.

However, a design with these goals and techniques alone is insufficient to enable the systematic study of dynamic behavior. This paper also describes the following key results for the MicroGrid:

- validation of the CPU resource model on one and several virtual resources per physical resource,
- validation of the online network simulation models exercised by real transport protocol stacks,
- validation of the MicroGrid system on a range of grid application programs ranging from kernels to full-blown applications on two grid resource configurations,
- design and evaluation of three approaches for the critical load-balancing problem for scalable network simulation, and
- a range of experiments which show that while static prediction can achieve good parallel scaling, only profile information can provide even better scaling.

The remainder of the paper is organized as follows. In Section 2, we describe the problem of modeling dynamic grid behavior in detail. The simulation-based approach used by the MicroGrid is covered in Section 3. The design and implementation used in the MicroGrid 2.4 is presented in Section 4. Validation of the constituent models, and the entire MicroGrid system on applications is described in Section 5. Section 6 explores challenges in scaling the MaSSF network simulator, particularly several load-balancing approaches. Section 7 discusses related work and Section 8 summarizes our results and discusses some future directions.

2 THE PROBLEM: MODELING DYNAMIC GRID BEHAVIOR

We face daunting research challenges in understanding the dynamic behavior of grid environments (applications, middleware, resources, and networks), and critical practical challenges in the engineering. To support the next generation of network services which will support a broad variety of critical commercial, scientific, and societal functions, we must be able to engineer resource stability, application performance stability, application quality of service, and also efficient resource utilization. The evolution of the distributed systems to grid environments is happening rapidly – driven by business pressures to reduce management cost, increase resource efficiency, and accelerate the process of designing and deploying information technology solutions.

Traditionally, distributed applications and networks have been studied largely separately – each community employing relative simple models for the other domain. For example, distributed systems researchers often used simple latency, bandwidth, and reliability models for networks, and networking researchers often used application models based on simple web-browsing or other simple models of workloads for applications. These methodologies have produced significant advances, but we are increasing faced with the reality that a broad range of distributed applications are now strongly network dependent, that is their performance depends directly on detailed dynamic network properties such as packet loss, protocol behavior, latency, bandwidth, etc. While significant advances have been made in aggregate modeling of network behavior[6, 7], at present only detailed packet-level or close analogs can accurately model protocol dynamics, particularly in extreme cases[8, 9]. At the same time, increasingly complex and dynamic applications can have dramatic impacts on networks; for example, peer-to-peer file sharing, viruses such as MyDoom, and multi-gigabit stream transfers for scientific applications. In particular, peer-to-peer file sharing and multi-gigabit scientific applications are exemplars of a future generation of applications which are highly network performance aware, and adapt their behavior and thereby network use rapidly and drastically in response to the experienced network performance. These concurrent changes motivate a strong need for integrated simulation and modeling of distributed systems and networks. Further, the increasing complexity of adaptive application and middleware behavior motivates the use of integrated simulation tools which allow these complex software systems to be used directly – accurate modeling is difficult.

Low-end pervasive or ubiquitous computing systems (i.e. Jini, Windows CE, Cell phone, etc.) systems also have similar needs. These applications often depend on open shared resource environments, must ensure application quality of service, and are subject to large fluctuations in load (which

may arise from crowds of devices!). While the structure of solutions for pervasive computing and grid systems may ultimately differ, the simulation and modeling needs for coupled network and resource modeling are remarkably similar.

In summary, the rapidly evolving needs of application, middleware, grid, and network designers as well as users and operators demand integrated simulation tools. Without tools that integrate resource, network, and software system modeling, accurate study of general system dynamics is not possible. Our goal is to develop and implement simulation tools that meet these needs, enable scientific and systematic of the dynamic behavior of applications, middleware, resources, and networks. These tools will provide a vehicle for observable, repeatable study and systematic exploration of design spaces for a wealth of design problems for applications and middleware, exploration of rare or extreme situations, and rational choices in application deployment and grid resource and network design.

3 An Online Simulation Approach: the MicroGrid

We describe the MicroGrid approach which provides an online simulation capability for real grid applications and middleware, enabling accurate experiments with large numbers of resources with arbitrary performance ratios. The MicroGrid enables study of large complex grids of today and those that will exist with future technologies.

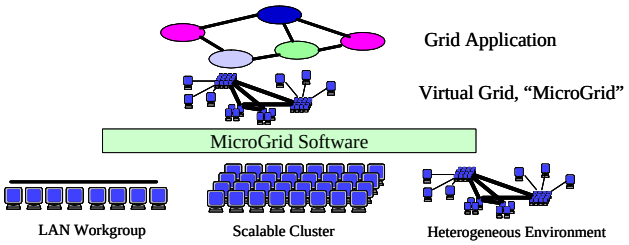

Figure 1. The MicroGrid Approach

The MicroGrid provides an online simulation of virtual grid environments transparently, allowing applications to be run unchanged. At launch, the MicroGrid reads a virtual grid configuration, and then builds corresponding simulation objects so as to provide the experience of running on virtual grid. These simulation objects implement models of network elements, compute resources, or grid information services. The MicroGrid can implement the virtual grid simulation using essentially any physical resources, including homogeneous clusters, heterogeneous grid resources, or even on a single computer.

High speed grid simulation is achieved by direct execution of applications and middleware against a CPU scheduler which controls the virtual speed and capacity of the resource. Direct execution allows experiments to proceed at near real-time. The MicroGrid uses a wrapper library which automatically intercepts library functions in user applications, thereby creating hooks for the virtual grid simulation system. Thus MicroGrid users can run any applications on the MicroGrid by simply re-linking the applications to the "wrapper" libraries; no changes to application or middleware source codes or understanding is needed.

The ability to control resource and network speeds in an online simulation (as opposed to emulation) enables the MicroGrid to support arbitrary performance ratios between elements in the simulation. This capability can be used to simulate future networks or processors which are much faster connected to slow 100Mbit networks, future 100Gbit networks, or every different speed in between. For example, by slowing the simulated speed of computing resources, the effect of future high speed transparent optical networks can be studied.

To use the MicroGrid, a user specifies the following:

First, the set of virtual resources, including network connectivity and protocols must be described.

- Network topology (Nodes, including routers and hosts and Network links, link capacity and link latency)
- Network protocol (Transport protocols -- TCP or UDP and Routing protocols – OSPF, BGP)
- Node properties related to communication protocols (e.g. TCP buffer, send window, receive window, segment size, etc.)
- Compute (relative CPU speed)
- Compute Node Connections into the network

Second, the MicroGrid simulation must be deployed against the physical resources. The MicroGrid simulation takes as input a set of physical resources used for the compute and online network simulation.

Based on the specification of the virtual and the physical resources, the MicroGrid intelligently maps virtual machines to physical hosts. The automatic mapping balances the compute and memory load across physical machine and reduces the network traffic between them. Both of these optimizations improve the scalability of simulations. If desired, the user can manually control these mappings.

4 Design & Implementation

4.1 Overview

The basic functionality of the MicroGrid allows grid experimenters to directly execute their applications in a virtual grid environment. The MicroGrid can exploit either homogeneous or heterogeneous physical resources (see Figure 1). We describe the MicroGrid 2.4 implementation, released in February 2004 and available from http://www-csag.ucsd.edu/. The MicroGrid 2.4, succeeding earlier MicroGrid implementations which go back as far as October 2000, supports Grid applications that use the Globus Toolkit 2 middleware infrastructure. The key challenges in constructing such a high-fidelity virtual grid are as follows.

- **Virtualization**: The application perceives only the virtual grid resources (host names, networks), independent of the

physical resources being utilized. This is achieved by virtualizing the grid information services and virtualizing/simulating the appropriate operating system resources.

- **Global Coordination**: The MicroGrid provides a coherent global simulation of dynamic virtual resources, all running on heterogeneous physical resources. One major function is to coordinate the simulation speed of different virtual resources. Based on the desired virtual resources and physical resources employed (CPU capacity and network bandwidth/latency), the virtual time module determines the *maximum feasible simulation rate*, under which all resource simulation can be run in a functionally correct manner.

- **Resource Simulation**: Each virtual resource (host, CPU, network, disk, etc.) is modeled accurately as an element of the overall 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.

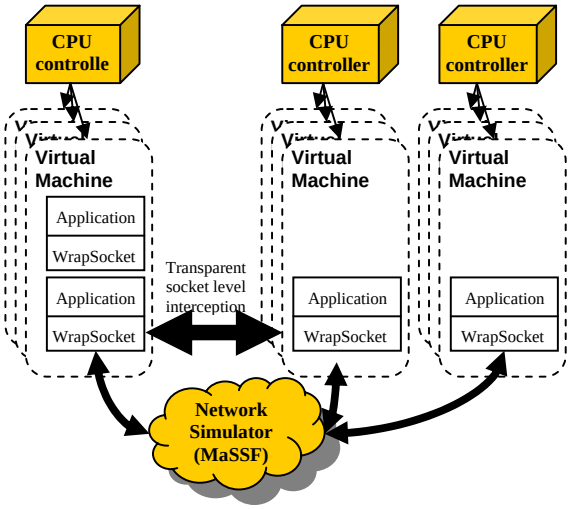

Figure 2. Architecture of the MicroGrid tools

The MicroGrid approaches which address these challenges are discussed in the following subsections.

4.2 Virtualization

To provide 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.

4.2.1 Virtualizing Resources

In general, the MicroGrid needs to virtualize processing, memory, networks, disks, and any other resources being used in the system. However, since operating systems effectively virtualize 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

By intercepting these calls, a program can run transparently on a virtual host with the appearance of the virtual hostname and IP address. The interception ensures that the program can communicate with processes running on other virtual Grid hosts. Many program actions which utilize resources (such as memory allocation) only name hosts implicitly, and thus do not need to be changed. We can run any socket-based application on the virtual Grid as the MicroGrid completely virtualizes the socket interface.

An interactive user of the MicroGrid typically logs in directly on a non-virtualized host and submits jobs to a virtual Grid. Thus, the job submission must cross from the real resources domain into the virtual resources domain. For the Globus middleware, our current solution is to run all gatekeepers, jobmanagers, and client processes on virtual hosts. Thus jobs are submitted to virtual servers through the virtual Grid resource's gatekeeper, which runs in the real domain and is modified to connect into the virtual host domain.

4.2.2 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
- 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. The following 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

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

4.3 Online Network Simulation (MaSSF)

MaSSF (pronounced "massive") is a scalable packet-level network simulator that supports direct execution of unmodified application. MaSSF consists of four parts.

- **Simulation Engine**: MaSSF uses a distributed simulation engine based on DaSSF[10]. It utilizes MPI-connected cluster systems to achieve scalable performance. We also employ a soft real-time scheduler to allocate CPU proportionately. This scheduler can also run in a scaled-down mode when the simulated system is too large to be run in real time on available hardware. With the global coordination of the MicroGrid, this feature provides extreme flexibility to simulate a wide range of networks accurately.

- **Network Modeling**: MaSSF provides necessary protocol modules for detailed network modeling, such as IP, TCP/UDP, OSPF, and BGP4. We have built simplified implementations of these protocols which maintain their behavior characteristics. We also use a network configuration interfaces similar to a popular Java network simulator implementation, SSFNet[11], for user convenience.

- **Online Simulation Capability**: To support simulation of traffic from live applications, we employ an Agent which accepts and dispatches live traffic from application wrapper to the online network simulation. Traffic is also sent back to application through the Agent module.

- **Live Traffic Interception**: Application processes use a wrapper library called WrapSocket to intercept live network streams at the socket level. The WrapSocket then talks with the Agent module to redirect traffic into the network simulator and vice versa. WrapSocket can be either statically or dynamically linked to application processes and requires no application modification.

These components and their relationship are illustrated in Figure 3. In the following sections we will present a more detailed description and rationale for our design choices.

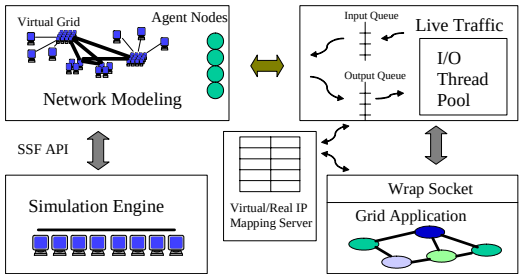

Figure 3. The MaSSF Scalable Network Simulation System

4.3.1 Scaled-Real time Online Network Simulation

One major goal of MaSSF is to support direct execution of real applications. So we need to intercept live traffic from applications and present it to the network simulator. There are many approaches to achieve this. We can either make it happen at the socket level by intercepting the send(), recv() network related system calls or we can make it at the IP packet level by manipulating the IP packet directly. The difference of these two approaches is whether we use TCP stack of the node operating system (see Figure 4). The advantage of the second approach is that it does not require us to model the TCP stack, a much simpler implementation. However, using original TCP stack means that we have to do the simulation on real time, since the OS TCP stack observes the real packet RTT(round trip time) and adjusts its send rate according to whatever RTT it gets. This is a big constraint, since in many situations the physical resources are not fast enough to achieve real time simulation. So in MaSSF, we take the first approach, intercepting the live traffic at socket level for scaled-real time simulation.

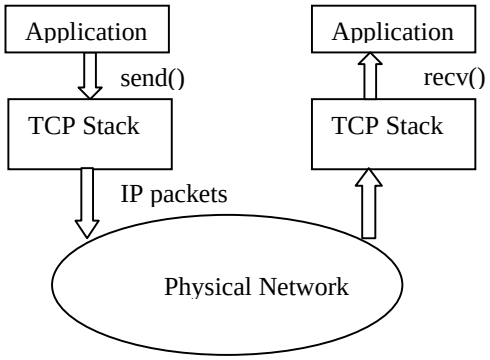

Figure 4. Traffic Flow in Real World

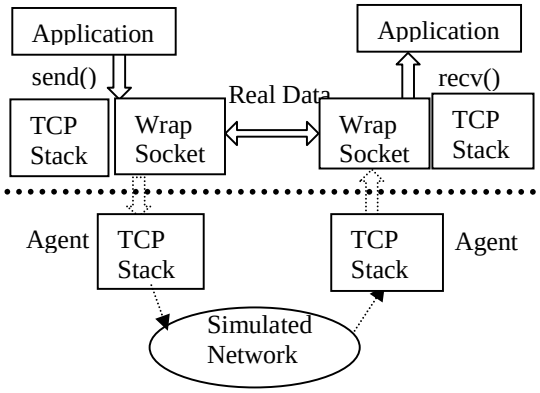

Figure 5. Traffic Flow in MaSSF

As shown in Figure 5, MaSSF intercepts all network related system calls using a library WrapSocket. This library can be either statically or dynamically linked to application programs. Every virtual host has a corresponding Agent inside the simulator, and the WrapSocket deliver to the Agent a logical reference for each intercepted network operation. A detailed TCP stack is implemented inside MaSSF and packet movement and timing are simulated accurately. Only a packet reference is routed in the simulated network, the real data stays in the WrapSocket and is delivered directly to the destination processes' WrapSocket library. There is no extra data copy, and minimal real network traffic is incurred. When all required data arrive the destination Agent, it returns the recv() to WrapSocket successfully. At this point, we expect that all real data is already waiting in the WrapSocket, since it is transferred directly through the fast local network. Then the application recv() call is returned with the real data.

In our approach, all network behaviors (including TCP sliding window management, link congestion, and packet drop, etc.) are modeled precisely inside the simulator, and the only source of distortion is the delay for transferring a logic reference from WrapSocket to Agents. Since this is just a small amount of data (~60 bytes) moving across a fast local link, its impact on the simulation of a wide-area network delay is negligible.

4.3.2 Detailed Network Simulation

MaSSF's goal is detailed modeling and simulation of Internet protocols and networks. It uses object-oriented simulation components to construct a network, setup network protocols running on hosts and routers, and create/accept traffic to be simulated. MaSSF models are self-configuring - that is, each MaSSF class instance can autonomously configure itself from a configuration file in DML format [12].

An input DML file specifies the network topologies, including network entities (host/router) and link between entities. The link latency and bandwidth are also specified in the DML file. For each entity, the user can also decide which network protocols are running on it. For example, a host can be configured with *IP*, *TCP*, and *Socket* protocols, plus traffic generator module *tcpClient* or live traffic Agent module. A router can be configured with *IP*, *TCP*, *OSPF* modules as internal AS router and it can also be configured with *IP*, *TCP*, *BGP* modules to be used as a BGP router. MaSSF provides these basic components and users can construct a network entity using any reasonable module combination. Users can also write their own protocol modules for new application or network devices.

All simulation modules are implemented above the SSFAPI[13], using the underlying conservative discrete event simulation engine. Basically each network packet is represented by a simulation event, and models the IP packet movement in the network hop by hop, including link transfer delay, queuing delay in a router queue, and packet drop. The simulation engine has a real-time scheduler that delivers the event at the exact time specified by the event timestamp. In this way, we can capture the link congestion and network dynamics in the real world.

4.3.3 Distributed Simulation Engine

To achieve scalable performance, MaSSF uses a distributed simulation engine running on a cluster. Given a network topology and available cluster nodes, MaSSF partitions the virtual network to multiple blocks, assigns each block to a cluster node, and simulates in parallel, as shown in Figure 6. Every cluster node runs a discrete event simulation engine and events are exchanged among cluster nodes. To maintain the simulation accuracy, these cluster nodes also need to synchronize periodically.

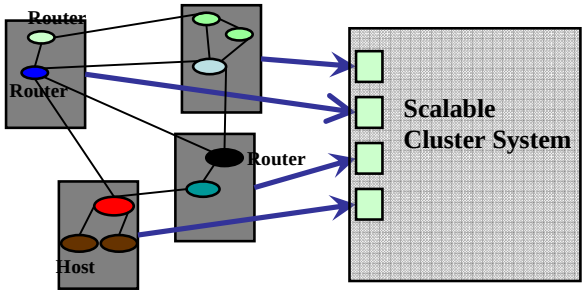

Figure 6. Mapping routers to physical resources

For large simulations, the network mapping cannot be done manually or casually. Instead, the mapping is a critical and demanding challenge. First we need to achieve load balance across all cluster nodes. This is difficult because the workload on each physical node varies greatly, depending both on the virtual mapping and network traffic in that subset of virtual network [Figure 7]. And we should also consider two more optimization goals. One is to maximize link latency across partitions to reduce the frequency of synchronization among simulation engines and maximize concurrency, a critical element of scalability for large scale simulation. This feature is an attribute of our MaSSF system and all other network simulators based on conservative discrete event simulation engines. The second optimization

goal is to minimize the communication of simulation events between simulation engine nodes. It is expensive to transfer a simulation event across physical nodes both in terms of computation overhead and communication latency. Also, the physical network of the simulation engine nodes is often a performance bottleneck for the whole simulation. Hence, it is important to minimize this communication.

To achieve the optimal load balance even if the traffic were known is an NP-Complete problem, and in practice, a network mapping problem can be naturally modeled as a graph partitioning problem and solved with the classical graph partitioning algorithms. With detailed traffic information, we can estimate the number of simulation events on each single link and use it to calculate the edge weight. We discuss this approach in greater detail in Section 6.

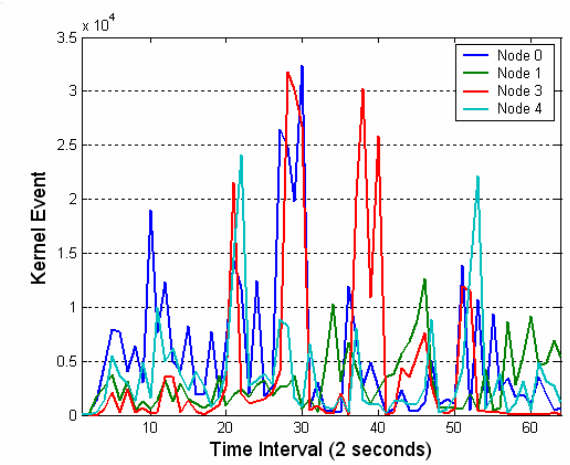

Figure 7. Load Variation over the Lifetime of Simulation

4.4 The MicroGrid CPU controller

The CPU controller virtualizes the CPU resources, providing appropriate performance for the processes running on virtual compute resources. The MicroGrid uses one CPU controller on each physical host to monitor the resource utilization of the processes on each virtual machine, and starts/stops them using SIGSTOP and SIGCONT signals. The controller consists of three parts:

- **Live Process Interception**: Whenever a virtualized process or a thread is created or is destroyed, the CPU controller detects the event via intercepted main() or exit() function calls and updates its internal process table.
- **CPU Usage Monitoring**: Every 20ms, the controller reads the /proc file system to check the CPU usage of all the processes in its process table.
- **Process Scheduling**: The controller calculates the CPU usage of each virtual host in a time window. If the amount of effective cycles exceeds the speed of the virtual hosts, the controller sends a SIGSTOP signal to all processes of the virtual host; otherwise, the controller wakes up the processes and let them proceed.

The CPU controller also supports the ability to scale down the execution speed of all virtual compute resources, enabling it to simulate arbitrary relative CPU speeds.

For each step of process scheduling, we use a sliding window algorithm to track CPU usage information and make scheduling decision. Because Linux schedules processes in the unit of 10ms, called "jiffy"s, the controller uses a window size which is an integral number of jiffies. At the same time, we hope to keep the sliding window as small as possible – otherwise, the communication latencies may be masked by our scheduling granularity. So we determine the minimal sliding window so that the simulation error can be reasonably small. We use 5% as acceptable error and assume the scaled virtual machine speed is *p* (fraction of physical CPU), then the sliding window size (*w* jiffies) and the available jiffies *n* for virtual machine should satisfy:

$$w = \text{round}(n/p) \quad \text{and} \quad |1 - p*n/w| < 0.05$$

This architecture allows simulation of large numbers of machines (100's to thousands) on a small number of machines. Further, grids with extremes of heterogeneous performance from slow to fast machines can be modeled accurately.

5 Validation Experiments

5.1 CPU Modeling Using CPU Controller

To test the accuracy of the CPU Controller, we use a simple program "cpuhog" which only does computation without any input/output operations. We first run it directly on a physical machine, get the real running time T. Then we run it on a virtual machine, which is given different fraction λ of CPU by the CPU Controller, to get a controlled time T_λ. If CPU controller is accurate, the value $\lambda * T_\lambda / T$ should equal to 1.

Our experiments were performed on a dual 450MHz PII machine. "cpuhog" takes 10 seconds to complete without CPU controller. Recall that the CPU controller design uses a 5% acceptable error margin (see Section 4.4).

:

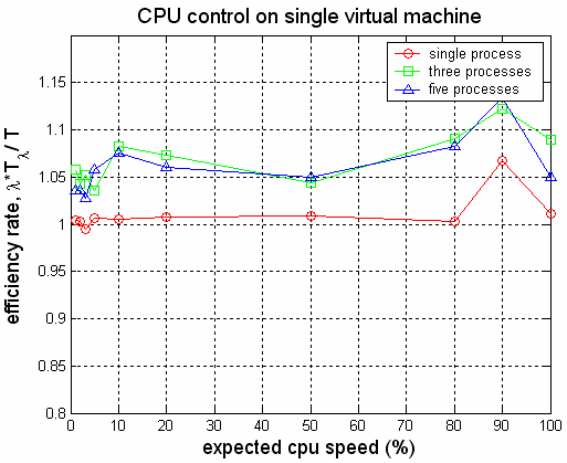

Figure 8. CPU Controller Performance for single virtual resource

Figure 8 are the results for single virtual resource. The results show that when there is only one process, the error is almost always in 2%, except as we near full utilization of the

underlying physical resource. At 90% CPU, we observe a 6.7% error. When there are multiple processes, the running time becomes about 6-8% longer.

We then run multiple virtual resources on each physical machine to understand the performance of CPU controller with some competitive workload. We do two groups of experiments with three virtual resources and five virtual resources on each physical machine respectively. Each time, we create the virtual resources, and launch one "cpuhog" on each virtual resource. Then use the average completion time as the virtual running time to calculate the efficiency rate $\lambda*T_\lambda/T$. The results are shown in Figure 9. The "aggregated CPU speed" is the sum of speeds of all the virtual machines. Most of the tests have an error of less than 4%, with the one exception a 9% error when total CPU is 78%.

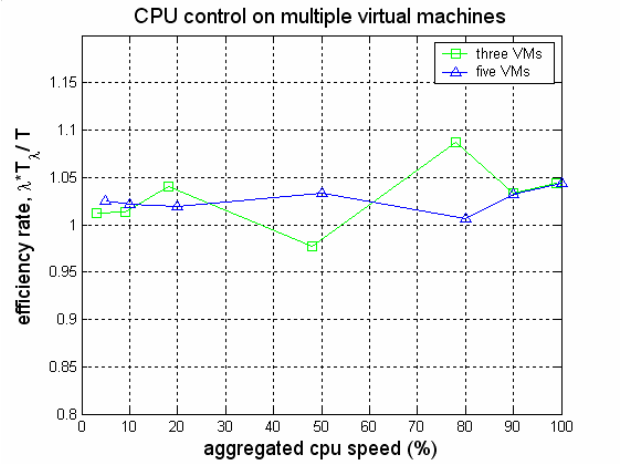

Figure 9. CPU Controller Performance for multiple virtual resources

The inaccuracy for 90% CPU in Figure 8 and for 78% CPU in Figure 9 mainly comes from the 5% acceptable error in the sliding window algorithm: Since we allow 5% error and we always choose the window size as small as possible, when virtual machine has speed 90%, we would schedule the application for six of seven jiffies rather than nine of ten, which causes theory speed of 85.7% CPU with 4.8% error from 90% CPU; in the multiple-virtual resource experiments, each virtual machines each has 26% CPU and is scheduled for one jiffy every four jiffies, which leads to 25% actually speed with about 4% error from 26% CPU.

In a summary, our experiments show that the CPU controller can model CPU speed accurately. The multiple-virtual resource experiments also demonstrate its capability to model multiple virtual CPUs on one physical machine accurately.

5.2 Network Modeling Using MaSSF

To test the performance of MaSSF, we use a client/server program which sends and receives packets using TCP/IP between two nodes. In each iteration, the sender sends a packet to the receiver then wait for a one-byte reply from the receiver. When packet size is small, the time for each iteration is the roundtrip time (RTT); when packet size is large enough, the bandwidth approximates the maximum bandwidth between the two nodes.

The TCP performance is affected by network latency (L), TCP window size (W), network capacity (C), and packet loss [14]. If there is no packet loss, the maximum bandwidth should be close to:

$$\text{Bandwidth} = \min(C, W/(2*L))$$

Our experiments first test the network performance between two nodes on a cluster. The nodes are dual Xeon 2.4GHz machines connected by GigE, configured with 128KB TCP window. Experiments show that real network has latency 0.222ms and bandwidth 782.87Mbps. On the MicroGrid, we simulate the two nodes with 128K TCP window and 0.2ms wire latency. The simulated results are shown in Figures 10 and 11.

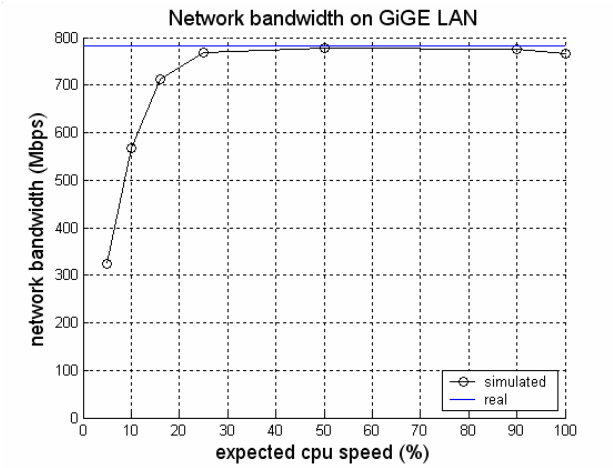

Figure 10. Network bandwidth on GigE LAN. The MicroGrid is configured with 0.2ms latency. The physical hosts are dual Xeon 2.4GHz machine.

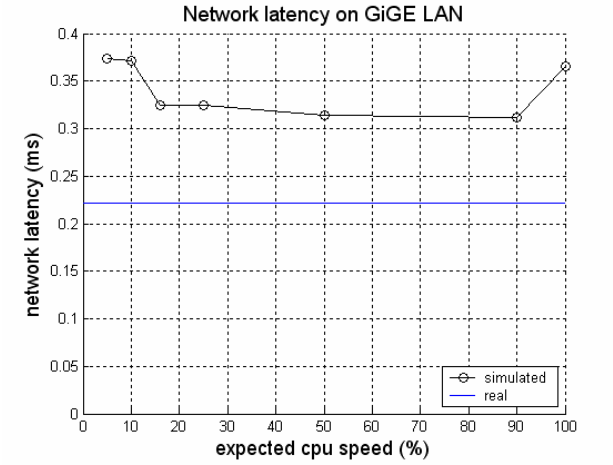

Figure 11. Network Latency on GigE LAN. The MicroGrid is configured with 0.2ms latency. The physical hosts are dual Xeon 2.4GHz machine.

The figures show that the virtual bandwidth (simulated) is

close to the target bandwidth when virtual CPU speed is faster than 25% of 2.4GHz Xeon. When virtual CPU speed is not fast enough to deal support the memory and I/O operations, the bandwidth falls off.

The network latency is about 0.15ms longer than the configured wire latency. This is presumed to be due to Agent overhead, overhead through TCP/IP stacks, and the overhead of the MaSSF simulator.

The next set of tests use a network topology with a 1ms latency between the two nodes, and varies the TCP window size from 32KB to 128KB. The results are shown in Figure 12 and 13.

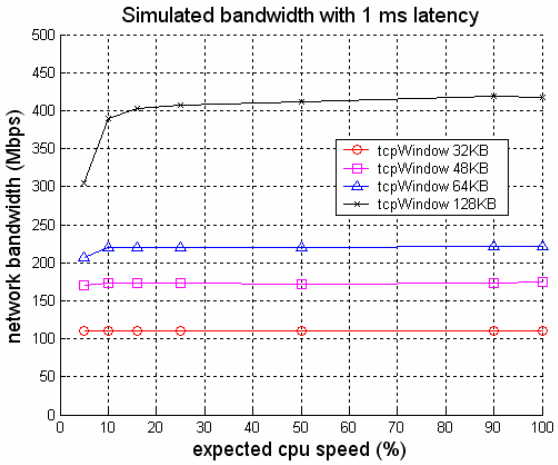

Figure 12. Network Bandwidth on MAN, the latency between nodes is 1 ms.

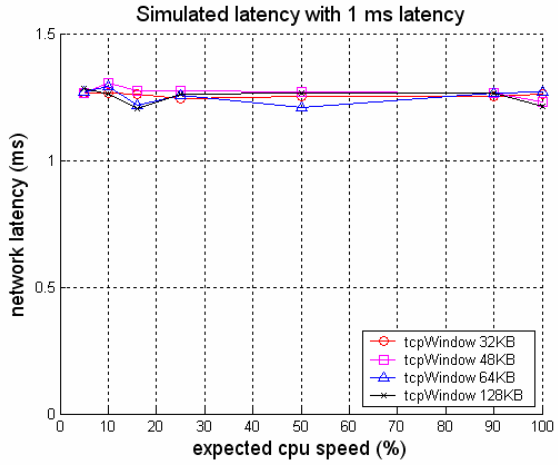

Figure 13. Network Latency on MAN, the latency between nodes is configured as 1 ms.

In this case, the network capacity is not the bottleneck any longer, so the TCP bandwidth is mainly decided by latency and TCP window size. We calculate the bandwidth upper bound in theory, as shown in Table 1.

From Figure 12 and Table 1 we see that our simulator achieves 82-90% of the theoretical maximum bandwidth. Considering the overheads on TCP stacks and application's memory operations, these are excellent results.

As for latency, the simulated value, as shown in Figure 13, is about 0.25ms higher than the configured wire latency. Still, this is due to overheads on TCP stacks, application memory operations, and MaSSF overhead.

	32KB	48KB	64KB	128KB
1 ms	128Mbps	192Mbps	256Mbps	512Mbps
5ms	25.6Mbps	38.4Mbps	51.2Mbps	102.4Mbps
10ms	12.8Mbps	19.2Mbps	25.6Mbps	51.2Mbps

Table 1. Theoretical Maximum Bandwidth on a Network Channel

The following figures show the bandwidth on network channel with latency 5 ms and 10ms respectively. The results are consistent to the theoretical bounds in Table 1.

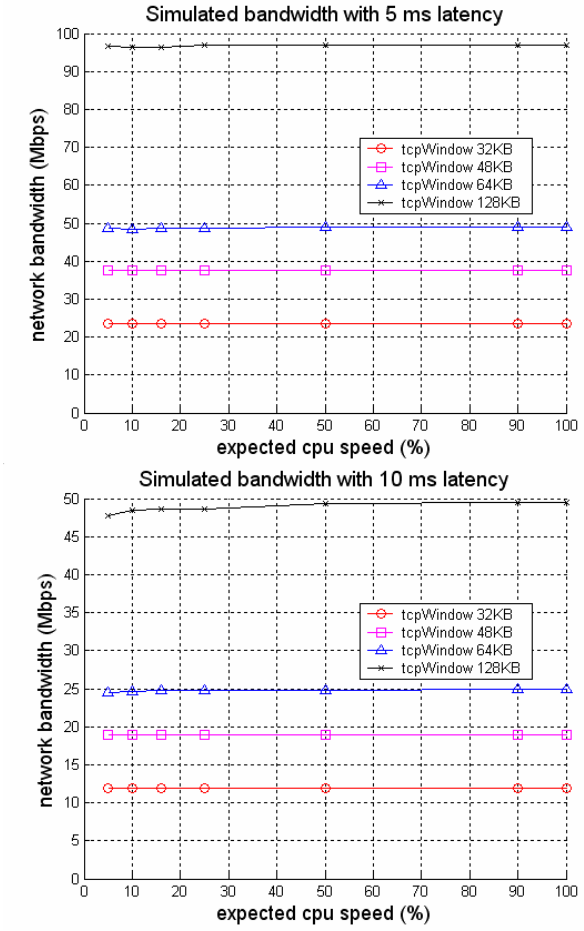

Figure 14. Simulated Network Bandwidth on network channel with latency 5ms and 10ms.

Based on these experiment results, we conclude that the MaSSF network simulator can model TCP communications accurately. With no network congestion, the modeled maximum bandwidth approximates real results in local, metro, and wide area networks. The network latency is also modeled accurately, except that MaSSF has some extra overhead which takes about 0.15-0.25ms per message.

We did not evaluate the simulator with network congestion,

although our simulator supports the capability to model competitive traffic (background traffic). Performance with congestion is not easy to evaluate since it depends on the competitive traffic model. In the following experiments, we model the network without and competitive traffic, likely overestimating performance.

5.3 Application Running on Emulated Environment

In this section, we run five classic applications on both real environment and virtual environment simulated using the MicroGrid. Before the results, we first introduce the five applications briefly. These applications are used in the GrADS project [5, 15].

All five applications are SPMD MPI applications and have been previously tested on the GrADS testbed in various real-world experiments. These applications were integrated into the GrADS framework and tested in various experiments as part of the following efforts: ScaLAPACK [16], Jacobi [17], Game of Life [17], Fish [18], and FASTA [19].

ScaLAPACK [20] is a popular software package for parallel linear algebra, including the solution of linear systems based on LU and QR factorizations. We use the ScaLAPACK right-looking LU factorization code based on 1-D block cyclic data distribution. The application is implemented in Fortran with a C wrapper. The data-dependent and iteration-dependent computation and communication requirements of ScaLAPACK provide an important test for the MicroGrid simulation. In our experiments we used a matrix size of 6000x6000.

FASTA [21] The search for similarity between protein or nucleic acid sequences is an important and common operation in bio-informatics. Sequence databases have grown immensely and continue to grow at a very fast rate; due to the magnitude of the problems, sequence comparison approaches must be optimized. FASTA is a sequence alignment technique that uses heuristics to provide faster search times than more exact approaches, which are based on dynamic programming techniques. Given the size of the databases, it is often undesirable to transport and replicate all databases at all compute sites in a distributed grid. We use an implementation of FASTA that uses remote, distributed databases that are partially replicated on some of the grid nodes. FASTA is structured as a master-worker and is implemented in C. For MicroGrid validation purposes, an important aspect of FASTA is that each processor is assigned a different database (or portion of a database) so the MicroGrid must properly handle input files and provide proper ordering of data assignments onto processors. In our experiments the sizes of the databases are 8.5MB, 1.7MB and 0.8MB respectively. The query sequence is 44KB.

The **Jacobi** method [22] is a simple linear system solver. A portion of the unknown vector x is assigned to each processor. During each iteration, every processor computes new results for its portion of x and then broadcasts its updated portion of x to every other processor. Jacobi is a memory-intensive application with a communication phase involving lots of small messages. In our experiments we used a matrix size of 9600x9600.

The **Fish** application models the behavior and interactions of fish and is indicative of many particle physics applications. The application calculates Van der Waals forces between particles in a two-dimensional field. Each computing process is responsible for a number of particles that move about the field. The amount of computation depends on the location and proximity of particles, so Fish exhibits a dynamic amount of work per processor. In our experiments we used 6,000 particles.

Conway's **Game of Life** [23] is a well-known binary cellular automaton. A two-dimensional mesh of pixels is used to represent an environment of cells. In each iteration every cell is updated with a 9-point stencil and then processors send data from their edges (ghost cells) to their neighbors in the mesh. Game of Life has significant memory requirements compared to its computation and communication needs. In our experiments we used a matrix size of 9600x9600.

We use a subset of the multi-site testbed for the GrADS project as our testbed. The 11 machines used are as following:
UCSD cluster: four 2100+ XP Athlon AMD (1.73 GHz) with 512 MB RAM each. These systems run Debian Linux 3.0 and are connected by Fast Ethernet.
UIUC cluster: three 450 MHz PII machines with 256MB memory connected via TCP/IP over 1Gbps Myrinet LAN. These systems run RedHat Linux 7.2.
UTK cluster: four PIII 550 MHz machines with 512MB memory, running RedHat Linux 7.2, and connected with Fast Ethernet.

The three sites are connected by the Internet2 network with 2.4Gbps backbone links. During our experiments, we observed NWS latency and bandwidth values over a period of 12 hours and obtained ranges as shown in table 2.

	UCSD machine	UIUC machine	UTK machine
UCSD machine	60-80Mbps, 0.2 ms	3-7Mbps 31 ms	4-6Mbps 30 ms
UIUC machine	3-7Mbps 31 ms	115-220Mbps 0.2 ms	7-17Mbps 11 ms
UTK machine	7-8Mbps 30 ms	12-18Mbps 11 ms	82-87Mbps 0.2ms

Table 2. Network performance of the testbed, reported by NWS. The variance of the bandwidth is due to resource sharing.

In our simulation, we configure all the machines to have 64KB TCP window. The wide area latency is as shown in Table 2; the LAN latency is 0.2 ms. We have to remind the audience that, the simulated LAN latency might higher than real latency due to simulation overhead as shown in subsection 5.2; while the simulated WAN bandwidth will higher than real

bandwidth due to lack of contention.

We do two groups of experiments: "Cluster" group uses four UTK machines to do clustering computation, "Grid" group uses three machines from each of the three sites. Both groups use a separate UCSD machine to run Globus gatekeeper. The results are shown in Figure 15.

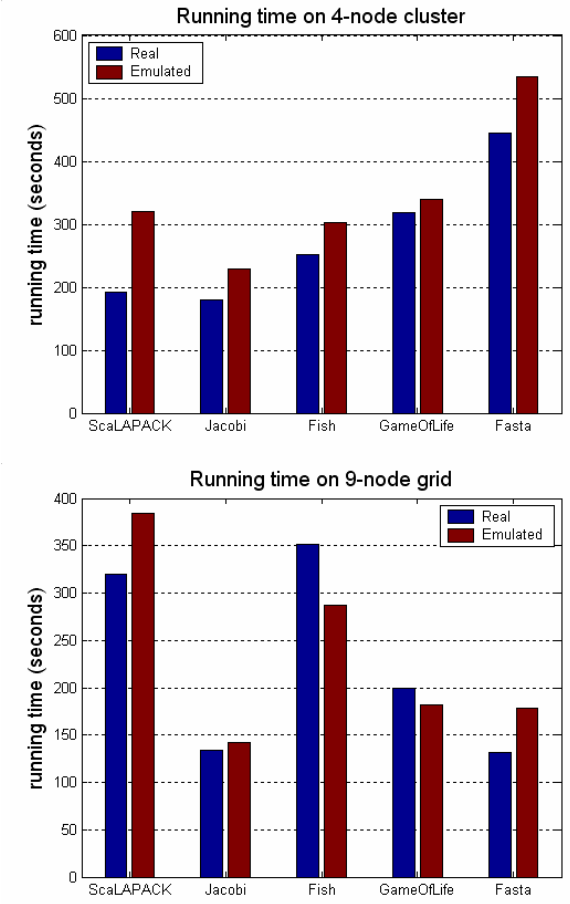

Figure 15. Running Time of Applications, on four-node cluster and nine-node Grid respectively.

For the cluster, all applications run slower on the MicroGrid than on real testbed; most of them have error in 6%-27%, except 66% for ScaLAPACK. The extra overhead comes from two major sources: 1) MaSSF has some overhead which increases network latency. 2) WrapSocket wraps many system functions for simulation, which will cause some overhead.

For the grid environment, the simulated time has about 5% - 35% errors. We can see several interesting differences from the cluster results. ScaLAPACK still runs slower on the MicroGrid than on real testbed, but much closer than on cluster, because ScaLAPACK uses a lot of small communications and the simulation overhead will have more impact on simulated LAN latency than on simulated WAN latency (as shown in subsection 5.2). Fish and GameOfLife run faster on the MicroGrid than on real grid. A possible reason is that they both use many large communications, and the simulated network bandwidth is higher than real system due to lack of contention.

6 IMPROVING SCALABILITY

The MicroGrid must be scalable to support the study of large networks, resources, middleware, and applications. While most resources can be naturally simulated in parallel with enough physical resources, all the coordination, synchronization and dynamic interaction amongst resources must go through network communication. This means the network must be simulated as a single system with global coordination, and thus the scalability of network simulation is a critical challenge for the entire MicroGrid system. In particular, the challenge is scalable detailed packet-level simulation combined with online simulation. We require packet-level simulation to ensure fidelity in simulation of network, protocol, and application behavior. Higher level simulation approaches, such as flow level simulation and approximation through network aggregation provide insufficient fidelity for our problems if interest in dynamic distributed systems.

As mentioned in Section 4.3, our network simulator MaSSF uses distributed discrete event simulation engine to achieve scalable performance. But only this is not enough to provide a scalable simulation. Like all other distributed or parallel applications, MaSSF must have good load balance for good speedup, and such load balance is challenging for network simulation. In this section we will present our approaches and results of load balance techniques for scalable network simulation.

6.1 Modeling Network Mapping as a Graph Partitioning Problem

Typical graph partitioning algorithms generally solve single objective partition problems such as:

Given an input graph G = (V, E) with weighted vertices and edges, we want to partition it into k parts such that,
*- each part has roughly the same total vertex weight(**constraint**)*
*- the edge-cut (*the number of edges*) that straddles partitions is minimized(**objective**)*

By setting the vertex and edge weights appropriately, mapping a simulated network to a set of physical simulation resources can be modeled as a graph partitioning problem and solved using a generic graph partitioning algorithm.

As a well studied problem, we expect that any high quality graph partitioning package (in this case METIS[24]) should produce results comparable to other graph packages. So our challenge is how to apply the graph partitioning algorithm in METIS to solve the mapping problem by defining the suitable input graph G, constraint conditions, and optimization objectives for the graph partitioning algorithm.

- **Input Graph**: The input graph G is defined by two categories of parameters: network structure and traffic

information. The network structure includes detailed network topology, link latency, and link bandwidth. Network traffic information is used to define edge weights in the graph, and it may also affect vertex weights.

- **Constraints**: The constraint is the vertex weight to be balanced among multiple vertices. In the network mapping problem, the vertex weight can be defined as weighted sum of computation and memory requirement on each simulation engine node.
- **Objectives**: The objective is the edge-cut to be minimized. In the network mapping problem, the optimization can use two objectives, maximal link latency and minimal communication across partitions, mentioned in Section 4.3.3.

In summary, the mapping process can be modeled as shown in Figure 16. First, it takes the network structure and traffic information as input, creates a graph G, and builds objectives and constraints for the graph partitioning algorithms. Then the mapping process applies partitioning algorithms to get a partitioned network. The partitioned network defines the mapping of simulated network nodes to physical resources (subject to additional arbitrary choices of placement amongst symmetric physical resources). In different cases, we explore how the abstractions of the network mapping problems are varied and use different constraints and objectives in the graph partitioning algorithm. The remaining problem is how to collect and use the traffic information, which will be discussed in the following section.

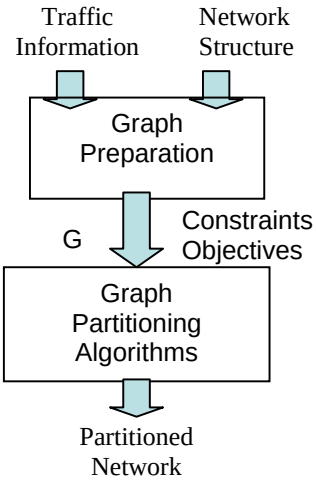

Figure 16. Process of Network Mapping

6.2 Traffic Based Network Mapping

We explore three different approaches for network mapping. These approaches vary how network topology, background traffic, and application traffic are represented and used in the partition. The more accurately an approach predicts the actual simulation work (i.e. network traffic), the better partitioning, and thereby better load balance are expected. However, there are tradeoffs between the specificity of the information used and the generality of the partition produced.

6.2.1 Network Topology-Based Mapping

Our first approach only considers the virtual network topology, link bandwidth, and latency. In this approach, TOP, each virtual node is weighted with the total bandwidth in and out of it. The optimization objective is to maximize the link latency between simulation engine nodes. This maximizes decoupling, supporting efficient parallel simulation.

6.2.2 Application Placement-Based Mapping

To achieve a better network mapping, we need precise traffic information. The second approach is based on the observation that simulated network traffic typically consists of a background and a foreground load. Foreground traffic is created by the target application that a user wants to study, and background traffic is used to provide realistic network conditions. We estimate both traffic loads separately, then combine them to estimate the aggregated traffic data for better network mapping. We call this approach PLACE.

For background traffic, all traffic generators can provide some prediction of their generated traffic load, for example, specifying the average traffic bandwidth between two endpoints. The foreground load is typically the live traffic from a small set of application programs. Unlike background traffic prediction, it is difficult for users to predict the traffic of the real application. As an approximation, we determine the traffic injection points of the application, where its processes attach to the simulated network, assuming that the application fully utilizes the network link at each injection point and every node talks to all other nodes with evenly distributed bandwidth. While this approximation may seem coarse at first glance, it is acceptable when considering that most target applications in simulation are complex and network intensive. With the source/destination pairs of all traffic flows, we can compute the aggregated traffic on each link by summing the contribution from each flow.

6.2.3 Profile-Based Mapping

The third approach uses profiling techniques to obtain traffic information automatically from simulation experiments (PROFILE). The profiles are then used to estimate future network use, and to improve the network mapping. Typically this involves an initial simulation experiment using an initial partition and traffic monitoring. The simulation yields detailed traffic information and the network can be repartitioned based on this information.

The critical challenge for this approach is the efficient collection and representation of traffic information during profiling, and the use of this information to repartition the network. In MaSSF, we implement the Cisco NetFlow-like [25] function on each simulated router. This functionality is used to record every traffic flow on each router to a local file. The dump files record the average bandwidth and duration of every flow on every router. Parsing the dump files allows computation of the aggregated traffic on every router and link in the network. By tuning the granularity of the NetFlow, we can get detailed network traffic information with small overhead.

6.3 Experimental Evaluation

6.3.1 Methodology and Experimental Setup

To evaluate these mapping approaches, we implement them in the MaSSF network simulator of the MicroGrid Project [26]. We apply these approaches on a range of different simulated network topologies and background traffic conditions.

Metrics

Three evaluation metrics are used in the experiments: load imbalance, application simulation time, and network simulation time. We define the load of a simulation engine node as the simulation kernel event rate (essentially one per packet). Using these counters, we calculate the overall ***load imbalance*** across all the physical nodes. Assuming the simulation kernel event rates are k_1, k_2, …, k_n, for n nodes used by the simulation engine, the load imbalance is calculated as the normalized standard deviation of {k}.

The second metric is the ***application simulation time.*** If load balance is improved, this improvement should reduce the execution time of the application simulation. Since communication is typically the performance bottleneck for only part of the execution time, the application simulation time is not always directly correlated to network simulation load balance. Nevertheless, as faster simulation is the ultimate goal of load balance, it is an important criterion.

The third metric is ***network simulation time***, which directly measures how much time is required to simulate the traffic created by the application. MaSSF records all network traffic trace of a simulation execution, and then replays it without real computation in the application. When replaying, it tries to send out traffic as fast as possible, but still follows the real application casualty and message logic order. This is a direct measurement of the mapping approaches.

Network Topology	Router	Host	Simulation Engine Node
Campus	20	40	3
TeraGrid	27	150	5
Brite	160	132	8

Table 3. Network Topology Setup

Hardware Configuration

The experiments use two RedHat Linux clusters. The first cluster includes 24 dual 550MHz Pentium-II processors, linked with 100Mbps Ethernet switch, with 2Gbps backbone bandwidth. This cluster is used for the network simulation engine. The second cluster consists of 8 dual 1.6GPentium-III processors, linked with 1 Gbps Ethernet switch (with 24 Gbps backbone bandwidth). It is mainly used for the real application execution. Two clusters are connected by a single, full duplex gigabit Ethernet link.

Network Topologies

Three network topologies are used in our experiments. The first two represent real networks, such as the TeraGrid (http://www.teragrid.org/) and a section of a university campus network (Campus). To explore more complex network structures, our third network topology Brite is created by a generic topology generator (adapted from the BRITE[27] toolkits), which creates Internet-like topologies and also provides background traffic support.

Traffic Workloads

The experiments use aggregated traffic flows to create background traffic. Here HTTP clients and severs are selected randomly from endpoints in the virtual network. In this study, a HTTP traffic generator is used, which has been well-studied by other researchers [28]. While this background traffic model is not perfect, it exercises some range of network dynamics, allows user control of load intensity by changing those parameters, and is widely used [29-31].

Foreground traffic is created live from real Grid applications, including ScaLAPACK[16] and GridNPB3.0 [32]. GridNPB3.0 is a widely used set of grid benchmarks in a workflow style composition in data flow graphs encapsulating an instance of a slightly modified NPB task in each graph node, which communicates with other nodes by sending/receiving initialization data. GridNPB includes a range of computation types and problem sizes, and in our experiments we use the combination of Helical Chain (HC), Visualization Pipeline (VP), Mixed Bag (MB) applications, all run at class S size. These programs run for about 15 minutes on our platform.

6.3.2 Experiment Results

Load Imbalance

Application workloads are executed on three network topologies (Campus, TeraGrid, and Brite) with moderate background traffic, and the measured load imbalance for two applications (ScaLAPACK and GridNPB) is shown in Figures 17 and 18. The figures report the normalized load imbalance across the physical simulation engine nodes for each combination of mapping approach and network topology. Each mapping approach produces significantly different results. The application placement-based mapping (PLACE) improves significantly on topology-based mapping (TOP) for both ScaLAPACK and GridNPB applications. The profile-based mapping (PROFILE) further improve the load imbalance up to 66% and 48% for ScaLAPACK and GridNPB respectively. For both workloads, the profile-based mapping approach delivers the best performance among three approaches. It is clear that the use of detailed traffic information from a previous simulation execution provides a critical advantage in partitioning the network effectively.

The advantage of profile-based mapping over placement-based mapping for GridNPB is more significant than that for ScaLAPACK. This is due to the fact that for ScaLAPACK, the application-placement based traffic prediction is very close to the actual traffic pattern, so there is little improvement to be had for PROFILE. For GridNPB, in contrast, the traffic is more irregular and the application-placement based prediction is less accurate. As a result, significant load imbalance remains for PLACE, leaving

more room for improvement for PROFILE.

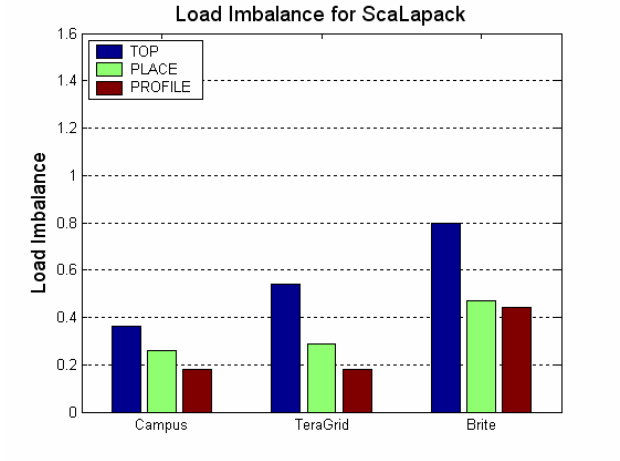

Figure 17. Load Imbalance for ScaLAPACK

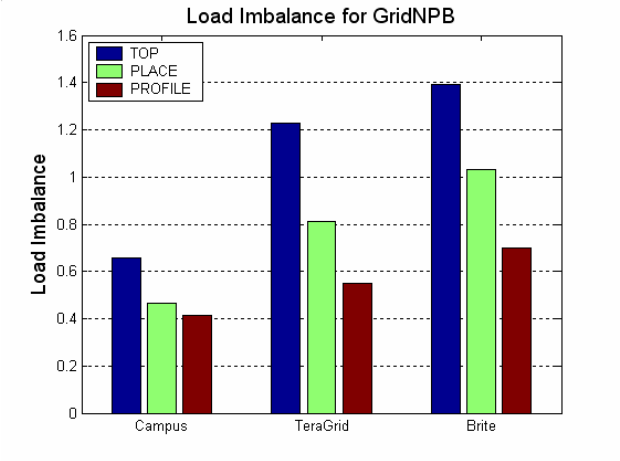

Figure 18. Load Imbalance for GridNPB

We can also see that the scale of the simulation affects the achieved load balance. The Campus network uses 3 simulation engine nodes, the TeraGrid uses 5 nodes, and the Brite network uses 8 nodes. The normalized load imbalance increases when the number of simulation engine nodes is increased, as one would expect if work were held constant (it is not across these experiments). When the simulation scales up, load balance is more critical to achieving high performance.

Application Simulation Time

The simulation time of both applications is shown in Figures 19 and 20. For ScaLAPACK, the use of application placement-based mapping (PLACE) reduces overall simulation time significantly (about 40%), and the use of the profile-based mapping (PROFILE) further reduces the simulation up to 50%. For the GridNPB workload, we can see the benefits of both PLACE and PROFILE mappings, but the improvement is much smaller (about 17%). As we have mentioned before, the simulation time is not a direct measurement of load imbalance, and because the execution time of GridNPB is computation rather than communication-intensive, improvement of the simulator gives little overall runtime benefit.

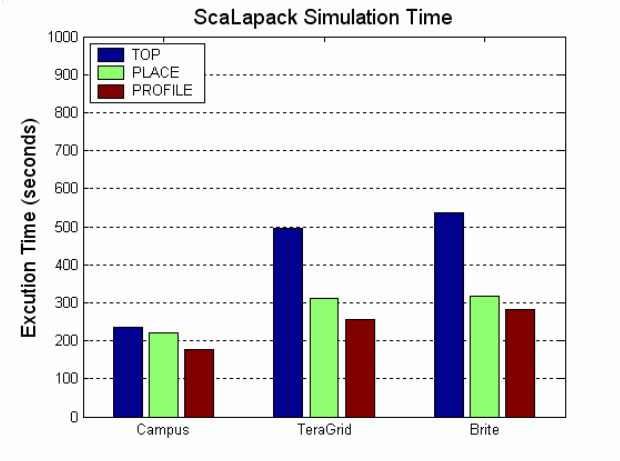

Figure 19. Simulation Time for ScaLAPACK

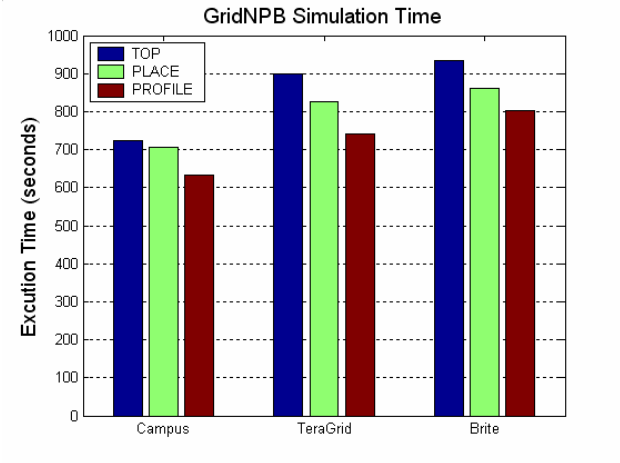

Figure 20. Simulation Time for GridNPB

Network Simulation Time in Isolation

All experiments above use the simulated application as targets, and the computation and communication are mixed together. To further understand the direct effect on network simulation, we use the MaSSF replay function to study the network simulation performance in isolation, as mentioned in Section 4.1.1. Figures 21 and 22 show that the simulation time for network traffic is improved significantly for ScaLAPACK replays, in consistent with the result of overall simulation time in Figure 19. For GridNPB, the network simulation time is also reduced by 30%, even when the execution time for the whole application shows less difference in Figure 20.

6.3.3 Scalability

To evaluate the effectiveness of our mapping approaches for larger network simulation, we use BRITE to build two network topologies with 3,000 routers and 3,000 hosts. The first network is a flat network in a single AS, using shortest path routing. The second network consists of 30 AS's and each AS has about 100 routers. BGP4 protocol is used for inter-AS routing and OSPF protocol is used for intra AS

routing. The simulator itself uses 24 simulation engine nodes and ScaLapack uses 5 additional nodes. For background traffic, there are 2,500 clients keeping continuously sending file requests to 300 servers. The average time gap between two successive requests is 5 seconds and average file size is 50KB. We only test the TOP and PROFILE approach here.

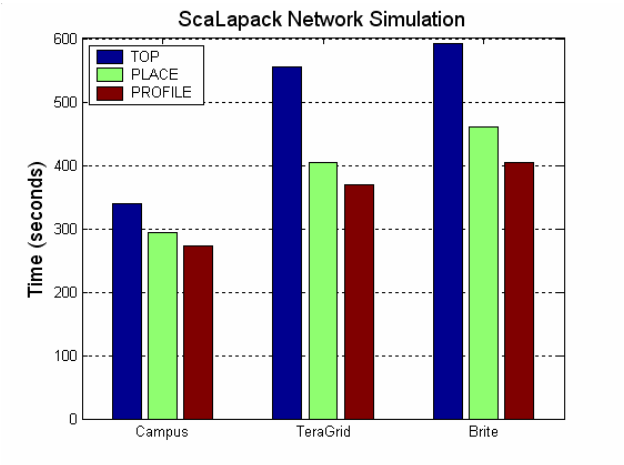

Figure 21. ScaLAPACK Isolated Network Simulation

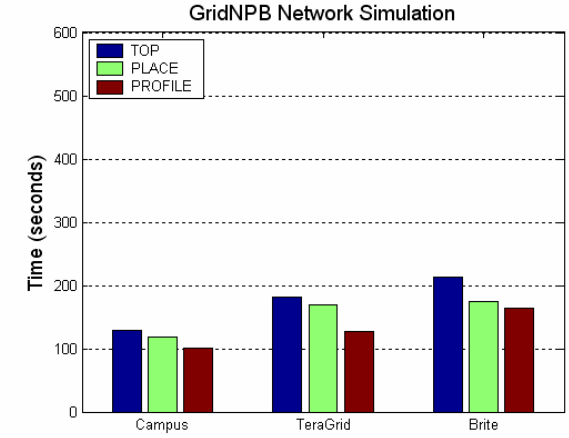

Figure 22. GridNPB Isolated Network Simulation

The results in Figure 23 show that the PROFILE approach continues to work well for these larger-scale networks, especially for the Multi-AS network. The Multi-AS network has much higher load imbalance when compared to the single AS network due to the different routing protocols used in these networks. For Multi-AS network, connectivity does not mean the reachability, due to differences between OSPF and hierarchical routing with BGP. However, because the PROFILE partition is based on the real traffic following the routing decision and flows, it can track the different routing structure and still balance load more effectively.

6.4 Summary

Experimental results show that network mapping using static network topology and predicted traffic information can improve load balance in large-scale network simulation. The topology-based approach (TOP) is fast and simple, and the placement-based approach (PLACE) can improve the performance for application with evenly distributed traffic load. For more irregular application and real large simulation, the profile-based approach (PROFILE) is most effective. Depending on the specific network structure and traffic load, it can improve load balance by up to 66% and speed up the simulation up to 50%. Further, PROFILE has also been shown to work well for large network simulations.

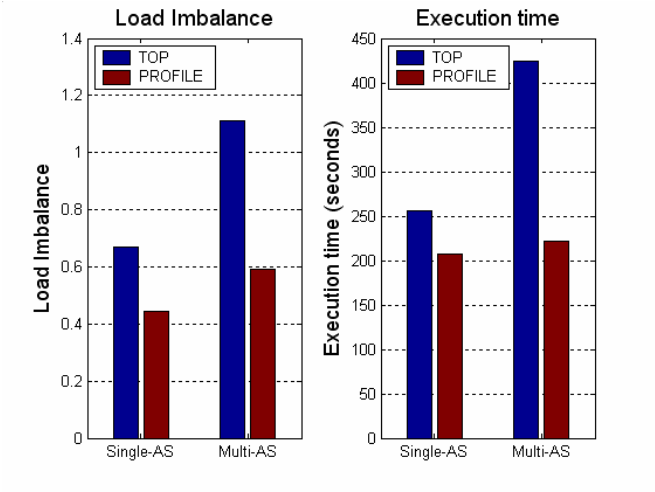

Figure 23. Scalability

7 Related Work

Three methods have been used to perform distributed system and Grid experiments: real testbeds, simulation, and emulation.

Real testbeds use a specific set of real resources for experiments, such as PlanetLab [33], TeraGrid, and GrADS testbed [5]. Real testbeds of course have the advantage of providing high speed execution and of course realistic execution. However, actual testbeds have a number of limitations, including: (i) limited experimental configurations (cannot run experiments for a wide range of platform scenarios or for platforms or networks that do not exist), (ii) non-observability – phenomena that occur which are not observable in routers, systems, networks, etc., and (iii) reproducibility – phenomena occur which cannot be repeated to be understood. We believe that tools such as MicroGrid are an essential complement to use of real testbeds.

Many research efforts explore network and computation simulation systems and techniques in order to model a wide range of distributed systems and networks. However, in early systems, distributed applications and networks have been studied largely separately – each community employing relative simple models for the other domain. These separate tools cannot be easily composed. For example, many network simulators have been built which provide accurate network environment (e.g. NS [34], GloMoSim [35]). However, these tools only capture part of what is relevant to

future distributed systems which couple resources and networks and have adaptive applications – they do not enable the network simulations to be coupled directly to applications.

A wide range of software tools provide general-purpose discrete-event simulation or even more focused Grid simulation libraries (GridSim and SimGrid) [36-40][41][42]. The challenge with all of these tools is that they do not allow easy use of existing applications and grid middleware, and thus the results achieved are only as good as the models which are developed for these complex pieces of software. In addition, these tools typically have simple models of networks and protocols – known to be inaccurate. No direct experimentation with applications, middleware, networks, and grid resources is supported.

Extensive research has been devoted to virtual machine monitors (VMM), including VMWare [43], Denali [44], and Xen [45]. The majority of these efforts focus on functional virtualization, and only secondarily on performance modeling. VMWare can aggregate a large number of distributed, heterogeneous resources as a single pool of processing, storage and networking power, on which user can run multiple off-the-shelf operating systems. Denali virtualizes hardware resources on single physical machine to enable running of multiple instances of a specific OS IIwaco. Xen is a virtual machine monitor for x86 that supports execution of multiple guest operating systems with both high performance and resource isolation. Commodity operating systems, such as Linux, BSD and Windows XP, can be ported to Xen. The major differences from the MicroGrid are: (i) their emphasis on functional virtualization, (ii) the need to maintain an entire OS kernel installation for each image, and (iii) the lack of support for detailed network simulation and performance modeling. Finally, the overhead and complexity of the MicroGrid wrapper and CPU scheduler is dramatically lower than any of these systems.

Several recent research efforts are most similar to the MicroGrid, including Albatross [46], Emulab [47] and Modelnet [48]. While these systems also support execution of real application over a modeled network, there are important differences between these efforts and the MicroGrid.

First, none of these systems model CPU speed, thus they cannot simulate grid environments with a wide range of heterogeneous computation resources. This also limits the ability to model relative compute and network speeds.

Second, the network modeling in these systems either use approximation models [49] or have limited scalability [34]. These approximations reduce the cost (compared to MicroGrid's global synchronized simulation) to achieve faster execution. For example, Emulab uses a set of real routers, switches and configurable software routers to emulate wide area network. This approach has the advantage of speed of emulation, but provides little in the way of detailed control of speed and modeling to the experiment designer. The ModelNet project at Duke University (and now at UCSD) is a software emulator. Their approach to scalability simplifies both network topology (a network of pipes) and routing (assuming a simple routing protocol based on shortest path) and then maps the resulting network of queues onto a set of emulation *cores*. This summarized network is an approximation to actual detailed network behavior. Further, there is no synchronization between these cores, so the number of cores can be used without affecting accuracy is unknown. In contrast, MaSSF uses full-scale detailed packet simulation based on a distributed discrete-event simulation engine. While there have been many efforts which use PDES for network simulation [50], we know of no other modeling efforts that achieve detailed online network simulation of the documented scale.

8 Summary and Future Work

The increasing acceptance of grid computing in both scientific and commercial communities presents significant challenges for understanding the performance of applications and resources. The associations between applications and resources are no longer static, and dynamic resource sharing and application adaptation further complicate the situation. To meet the emerging modeling needs and enable growth in understanding the dynamic properties of grids, we have designed and implemented a tool called the MicroGrid. The MicroGrid enables accurate and comprehensive study of the dynamic interaction of applications, middleware, resource, and networks. The MicroGrid creates a virtual grid environment – accurately modeling networks, resources, the information services (resource and network metadata) transparently. Thus, the MicroGrid enables users, grid researchers, or grid operators to study arbitrary collections of resources and networks. The MicroGrid includes the MaSSF online network simulator which provides packet-level accurate, but scalable network modeling.

We present experimental results with applications which validate the implementation of the MicroGrid, showing that it not only runs real grid applications and middleware, but that it accurately models both their and underlying resource and network behavior. We also study a range of techniques for scaling a critical part the online network simulator to the simulation of large networks. These techniques employ a sophisticated graph partitioner, and a range of edge and node weighting schemes exploiting a range of static network and dynamic application information. By carefully mapping the virtual network to physical resources using multi-objective graph partitioning algorithms, we achieve good load balance and better scalability in network simulation. Our studies show that the static network topology and application placement information can be exploited to achieve good balance for some application. In our experiments, it reduces the load balance by up to 66%. The profile-based mapping uses detailed traffic information and further reduces the application simulation time up to 50%. The best of these, profile-driven placement, scales well to online simulation of large networks of 6,000 nodes using 24 simulation engine

nodes.

In future work, we will use MicroGrid to study larger network and application, specially using a 256-node Itanium Linux cluster to simulation a network with 100,000 network entities, which can be taken as a non-trivial part of real Internet with hundreds of Autonomous System (AS). Under this scale of network, we expect to experience much larger load balance challenge and we have to make our traffic based load balance solution for better scalability. We will also use MicroGrid to study larger scale Grid applications, include resources scheduling and overlay network behaviors.

ACKNOWLEDGMENT

Supported in part by the National Science Foundation under awards NSF EIA-99-75020 Grads and NSF Cooperative Agreement ANI-0225642 (OptIPuter), NSF CCR-0331645 (VGrADS), NSF ACI-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from Hewlett-Packard, BigBangwidth, Microsoft, and Intel is also gratefully acknowledged.

The authors also acknowledge the contributions of Alex Olugbile to the system infrastructure which made this work possible.

REFERENCES

1. C. Kesselman, and I. Foster, *The Globus Toolkit*, in *The Grid: Blueprint for a New Computing Infrastructure*, and I. Foster C. Kesselman, Editor. 1999, Morgan Kaufmann Publishers, Inc. p. 259--278.
2. A. S. Grimshaw, W. A.Wulf, and the Legion Team, *The Legion Vision of a Worldwide Virtual Computer*. Communications of the ACM, 1997. **40**(1).
3. Douglas Thain, Todd Tannenbaum, and Miron Livny, *Condor and the Grid*, in *Grid Computing: Making The Global Infrastructure a Reality*, Anthony J.G. Hey Fran Berman, Geoffrey Fox, Editor. 2003., John Wiley.
4. S. Agrawal, J. Dongarra, K. Seymour, et al., *NetSolve: Past, Present, and Future - A Look at a Grid Enabled Server*, in *Grid Computing: Making The Global Infrastructure a Reality*, A. and Berman Hey, F. and Fox, G., editors, Editor. 2003, John Wiley.
5. Francine Berman, Andrew Chien, Keith Cooper, et al., *The GrADS Project: Software Support for High-Level Grid Application Development.* International Journal of High Performance Computing Applications, 2001. **15**(4): p. 327-344.
6. Vern Paxson and Sally Floyd, *Wide-Area Traffic: The Failure of Poisson Modeling.* IEEE/ACM Transactions on Networking, 1995. **3**(3): p. 226-244.
7. Vishal Misra, Weibo Gong, and Don Towsley. *A Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED*. in *ACM SIGCOMM'00*. 2000. Stockholm, Sweden.
8. James Cowie, Hongbo Liu, Jason Liu, et al. *Towards Realistic Million-Node Internet Simulations*. in *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*. June 28 - July 1, 1999. Las Vegas, Nevada.
9. Rich Wolski, Neil Spring, and Jim Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing.* Journal of Future Generation Computing Systems, October 1999. **15**(5-6): p. 757-768.
10. J. Liu, and Nicol, D., *DaSSF 3.1 User's Manual*. 2001.
11. *SSFNet webpage, http://www.ssfnet.org/*.
12. *How to write DML network models, http://www.ssfnet.org/InternetDocs/ssfnetTutorial-1.html*.
13. James H. Cowie, *SCALABLE SIMULATION FRAMEWORK API REFERENCE MANUAL*. 1999.
14. T. V. Lakshman, and U. Madhow, *The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss.* IFIP Transactions C-26, High Performance Networking, 1994: p. 135--150.
15. *The GrADS project, http://hipersoft.cs.rice.edu/grads*.
16. A.Petitet, S.Blackford, J.Dongarra, et al., *Numerical Libraries and the Grid: The GrADS Experiment with ScaLAPACK.* International Journal of High Performance Computing Applications, 2001. **15**(4): p. 359-374.
17. Holly Dail, Fran Berman, and Henri Casanova, *A Decoupled Scheduling Approach for Grid Application Development Environments.* Journal of Parallel and Distributed Computing, 2003.
18. Otto Sievert, and Henri Casanova, *A Simple MPI Process Swapping Architecture for Iterative Applications.* International Journal of High Performance Computing Applications (IJHPCA), 2004.
19. *FASTA package of sequence comparison programs at ftp://ftp.virginia.edu/pub/fasta*.
20. L. S. Blackford, J. Choi, A. Cleary, et al., *ScaLAPACK Users' Guide*. 1997: Society for Industrial and Applied Mathematics, Philadelphia, PA.
21. W. R. Pearson, and D. J. Lipman. *Improved tools for biological sequence comparison*. in *Proc. Natl. Acad. Sci.* 1988.
22. Richard Barrett, Michael W. Berry, Tony F. Chan, et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. 1994, Philadelphia, PA: SIAM.
23. Gary W. Flake, *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. 1998, Cambridge, MA: MIT Press.
24. Kirk Schloegel, George Karypis, and Vipin Kumar. *A New Algorithm for Multi-Objective Graph Partitioning*. in *Euro-Par'99 Parallel Processing*. 1999. Springer Verlag, Heidelberg.
25. Cisco Systems, *NetFlow*. 2001.
26. H. Song, X. Liu, D. Jakobsen, et al. *The MicroGrid: a Scientific Tool for Modeling Computational Grids*. in *IEEE Supercomputing (SC 2000)*. Nov. 4-10, 2000. Dallas, USA.
27. Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. *BRITE: An Approach to Universal Topology Generation.* in *In Proceedings of the International*

Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS '01. 2001. Cincinnati, Ohio.
28. Paul Barford and Mark Crovella. *Generating Representative Web Workloads for Network and Server Performance Evaluation*. in *Measurement and Modeling of Computer Systems 1998*. 1998.
29. David P. Olshefski, Jason Nieh, and Dakshi Agrawal. *Inferring Client Response Time at the Web Server*. in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2002)*. 2002. Marina del Rey, CA.
30. Rong Pan, Balaji Prabhakar Konstantinos Psounis, and Damon Wischik. *SHRINK: A Method for Scalable Performance Prediction and Efficient Network Simulation*. in *IEEE INFOCOM*. 2003.
31. Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. *Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites.* in *Proceeding of 11th World Wide Web conference*. 2002. Honolulu, Hawaii.
32. Rob F Van Der Wijngaart and Michael Frumkin, *NAS Grid Benchmarks Version 1.0*. 2002, NASA Ames Research Center.
33. *PlanetLab Website, http://www.planet-lab.org/*.
34. Lee Breslau, Deborah Estrin, Kevin Fall, et al., *Advances in Network Simulation.* IEEE Computer, May, 2000. **33**(5): p. 59-67.
35. Lokesh Bajaj, Mineo Takai, Rajat Ahuja, et al., *GloMoSim: A Scalable Network Simulation Environment*. May 1999, UCLA Computer Science Department Technical Report 990027.
36. H. Schwetman. *CSIM: A C-based, process oriented simulation language*. in *Proceedings of the 1986 Winter Simulation Conference*. 1986.
37. S. Toh. *SimC: A C Function Library for Discrete Simulation.* in *Proceedings of the 11th Workshop in Parallel and Distributed Simulation*. 1993.
38. A. Miller, R. Nair, and Z Zhang. *JSIM: A Java-Based Simulation and Animation Environment*. in *Proceedings of the 30th Annual Simulation Symposium (ANSS'97)*. 1997.
39. F. Gomes, S. Franks, B. Unger, et al. *SimKit: A High Performance Logical Process Simulation Class Library in C++*. in *Proceedings of the 1995 Winter Simulation Conference*. 1995.
40. F. Howell, and McNab R. *SimJava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling*. in *Proceedings of the First International Conference on Web-based Modelling and Simulation*. 1998.
41. R. Buyya, and M. Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing.* The Journal of Concurrency and Computation: Practice and Experience (CCPE), 2002. **14**(13-15).
42. A. Legrand, L. Marchal, and H. Casanova. *Scheduling Distributed Applications: The SimGrid Simulation Framework.* in *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan*. 2003.
43. *VMWare website, http://www.vmware.com/*.
44. Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. *Scale and Performance in the Denali Isolation Kernel*. in *Fifth Symposium on Operating System Design and Implementation (OSDI 2002)*. 2002. Boston, MA.
45. Paul Barham, Boris Dragovic, Keir Fraser, et al. *Xen and the Art of Virtualization*. in *Nineteenth ACM Symposium on Operating Systems Principles*. 2003. Bolton Landing, NY.
46. T. Kielmann, H. Bal, J. Maassen, et al., *Programming Environments for High-Performance Grid Computing: the Albatross Project.* Future Generation Computer Systems, 2002. **18**(8).
47. Brian White, Jay Lepreau, Leigh Stoller, et al. *An Integrated Experimental Environment for Distributed Systems and Networks*. in *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*. December 2002.
48. Amin Vahdat, Ken Yocum, Kevin Walsh, et al. *Scalability and Accuracy in a Large-Scale Network Emulator*. in *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*. December 2002.
49. L. Rizzo. *Dummynet and Forward Error Correction*. in *Proc. of the 1998 USENIX Anuual Technical Conf.* 1998. New Orleans, LA: USENIX Association.
50. Rob Simmonds, Russell Bradford, and Brian Unger. *Applying parallel discrete event simulation to network emulation*. in *14th Workshop on Parallel and Distributed Simulation (PADS 2000)*. May 28-31, 2000. Bologna, Italy.