The MicroGrid: Using Emulation to Predict Application Performance in Diverse Grid Network Environments

Huaxia Xia[*], Holly Dail[†], Henri Casanova[*†], Andrew Chien[*]

[*]Department of Computer Science and Engineering
University of California at San Diego

[†]San Diego Supercomputer Center
University of California at San Diego

[hxia,hdail,casanova,achien]@cs.ucsd.edu

Abstract

Improvements in networking and middleware technology are enabling large-scale grids that aggregate resources over wide-area networks to support applications at unprecedented levels of scale and performance. Unfortunately, existing middleware and tools can not in general inform a user as to the suitability of a given grid topology for a specific grid application. Instead, users generally develop ad hoc performance models for mapping their applications to resource and network topologies. Because application behavior alone is complex, and resource and network behavior further complicate the picture, users are typically reduced to nearly blind experimentation to discover how to deploy their application in the new grid environment. Only after the fact can a user discovers if the match was a good one. Further, even after finding a desirable configuration, there is no basis on which to determine if a much better configuration exists.

We present a richer methodology for evaluating grid software and diverse grid environments based on the MicroGrid grid emulator. With the MicroGrid, users, grid researchers, or grid operators can define and emulate arbitrary collections of resources and networks. This allows study of an existing grid testbed under controlled conditions or even to study the efficacy of higher performance environments than are available today. Further, the MicroGrid supports direct execution of grid applications unchanged. These application can be written with MPI, C, C++, Perl, and/or Python and use the Globus middleware. This enables detailed and accurate study of application behavior. This paper presents: (1) the first validation of the MicroGrid for studying whole-program performance of MPI Grid applications and (2) a demonstration of the MicroGrid as a tool for predicting the performance of applications on a range of grid resources and novel network topologies.

[0]This material is based upon work supported by the National Science Foundation under Grant #9975020 and #0103759.

1 Introduction

Rapid improvements in the performance of wide-area networks and the pervasiveness of commodity resources provide us grid computing infrastructures with tremendous potential [4]. This potential has been widely realized and many grid middleware projects such as Globus [18], Condor [31], and NetSolve [1] have been pursued to provide uniform, secure and efficient access to remote resources. Unfortunately, there are few tools that assist a user in predicting *if* their application will obtain suitable performance on a particular platform. Instead, with little information on predicted performance, users must invest significant time to obtain accounts on the new grid, adapt their application to new middleware, debug their grid executions, and finally run experiments to determine the best deployment of their application on the new grid. At last the user knows if the blind date was a good one: did the application run efficiently on the grid? If not, it is time to set up a new blind date and start over. Two major problems with this approach are that it is both labor and resource intensive, and does not provide any assurance of the quality of results. As resource environments and application performance structure continues to increase in complexity, it is likely the distance between achieved results and optimal will increase further.

The goal of the MicroGrid project is to develop and implement emulation tools that provide a vehicle for the convenient scientific study of grid topologies and application performance issues. The MicroGrid provides a virtual grid infrastructure that enables scientific experimentation with dynamic resource management techniques and adaptive applications by supporting controlled, repeatable 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. Further if application or middleware behavior is difficult to model accurately, the use of direct application execution enables accurate modeling. The Microgrid provides reduced setup effort for simulation and increases the observability of application behavior.

Previous papers have provided a broad overview of an early version of the MicroGrid [29], and studied approaches for load-balancing the workload of network emulation to enable scalable MicroGrid emulation [22]. This paper provides the following contributions.

- An extension of MicroGrid Toolkit features including added support for emulation of grid components written in C, C++, Python, and Perl.

- The first validation of the efficacy of the MicroGrid emulation approach for whole-program MPI applications. The paper presents experiments comparing the measured application behavior of five grid applications in real-world testbed environments against their behavior as measured in emulation. Note that our goal is not perfect prediction, but rather to provide users with some expectation of application performance in new topologies.

- The first demonstration of the MicroGrid as a tool for predicting the behavior of whole-program MPI applications on grid environments without the need for real-world access to those environments. These experiments explore topologies in existence today such as the TeraGrid [30] and high-performance network topologies that do not yet exist.

The remainder of the paper is organized as follows. Section 2 gives some background on the MicroGrid and Section 3 describes the five MPI applications we deployed in the real-world and the MicroGrid. Next, Section 4 presents experiments validating the MicroGrid and applying it to novel grid topologies. Section 5 provides an overview of related work and Section 6 concludes the paper and highlights future directions.

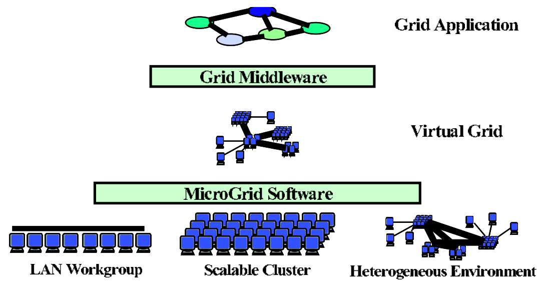

Figure 1. Architecture of the MicroGrid Emulation Toolkit.

2 MicroGrid Architecture

In this section we provide background on the MicroGrid but refer the reader to [29, 22] for complete details. The basic functionality of the MicroGrid allows grid experimenters to execute their applications in a virtual grid environment. The virtual environment runs on actual physical resources, providing the application with virtual grid information services and virtual resources. Note that the MicroGrid can exploit either homogeneous or heterogeneous physical resources (see Figure 1). The MicroGrid has to virtualize two different grid resources, network resources and compute resources, which is achieved by the MicroGrid network emulator (MaSSF) and CPU controller, respectively. We describe both of these components below. Note that the MicroGrid ensures coherent virtual executions by coordinating the simulation speed of the different virtual resources.

2.1 MaSSF: Scalable Network Simulation

MaSSF (pronounced “massive”) is a scalable packet-level network emulator that supports direct execution of unmodified applications. MaSSF consists of four parts.

- **Simulation Engine**: MaSSF uses a distributed simulation engine based on DaSSF [9, 21]. It utilizes MPI-connected cluster systems to achieve scalable performance. We also implement a real-time scheduler in order to enable best-effort emulation. This scheduler can also run in a scaled-down mode when the emulated system is too large to be emulated in real time on available hardware. With the global coordination of the MicroGrid, this feature provides extreme flexibility to emulate a wide range of networks accurately.

- **Network Modeling**: MaSSF provides necessary protocol modules for detailed network modelling, such as IP, TCP/UDP, OSPF, and BGP4. We have strived to simplify these protocols and maintain their behavior characteristics at the same time. We also use a network configuration interface similar to a popular Java implementation, SSFNET [9], for user convenience.

- **Emulation Capability**: To support simulation of traffic from live applications, we implement an Agent module to accept and dispatch live traffic from applications to the network modeling module. Traffic will also be sent back to application through the Agent Module.

- **Live traffic interception**: Application processes use a module 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 emulator and vice versa. WrapSocket can be either statically or dynamically linked to application processes and requires no application modification.

2.2 The MicroGrid CPU controller

The CPU controller virtualizes the CPU resources and processes of the physical machines by sending SIGSTOP and SIGCONT signals to physical processes. The controller consists of three parts:

- **Live Process Interception**: Whenever a 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 sliding 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. Same as MaSSF, the CPU controller also supports scaled-down mode to emulate virtual machines which are faster than avaliable physical resources.

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

3 Applications

In this section we describe five classic grid applications used for research and development in the GrADS project [5, 15]. We will use these applications for MicroGrid validation experiments in the following section.

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 [25], Jacobi [11], Game of Life [11], Fish [28], and FASTA [13], which was integrated by Asim YarKhan.

ScaLAPACK [6] 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 provides an important test for the MicroGrid emulation. In our experiments we used a matrix size of 6000×6000.

FASTA [24]. 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** [3] 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 9600×9600.

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 [14] 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 9600×9600.

3.1 Summary

This group of applications provides an interesting test suite for the MicroGrid emulation toolkit as it provides broad coverage of a number of application characteristics of interest.

- Predominantly floating point (ScaLAPACK, Jacobi, and Fish) vs. predominantly integer (Game of Life and FASTA)
- Fortran (ScaLAPACK) vs. C (the others)
- Varying, complicated sharing communication phases (ScaLAPACK) vs. regular, straightforward exchange of vector(s) all-to-all (Jacobi, Fish) vs. neighbor-to-neighbor communications only (Game of Life) vs. master-worker send-receive (FASTA)
- Full range of communication to computation balance from highly synchronized, large number of messages (ScaLAPACK) to small number of master-worker send-receive pairs (FASTA).

4 Experiments

In this section we describe experiments we performed using the MicroGrid and the five grid applications described in the previous section. In the first set of experiments, we test how accurately the MicroGrid emulates application execution behavior. For these experiments, we compare real-world executions of grid applications against MicroGrid emulations of the same environments. In the second set of experiments, we emulate a number of new, high-performance grid architectures and test the performance of our applications on these emulated architectures. These tests demonstrate how the MicroGrid can be used to evaluate applications and architectures that are not available in the real-world (or where controlled experiments are not possible).

4.1 MicroGrid validation

In our first set of experiments, we compare grid application behavior for real-world application executions against the behavior observed in the MicroGrid emulation. Specifically, we selected a multi-site grid and single-site cluster from the GrADS project testbed and ran the five grid applications on these two testbeds. Next, we built a MicroGrid resource model of the grid and cluster including the speed, architecture, and number of compute nodes and the TCP/IP configuration, latency and bandwidth of the links. Then, we executed the same applications on the MicroGrid emulation.

4.1.1 Real-world testbed configuration

A more detailed view of our testbed choices may be useful in considering which aspects of the MicroGrid emulation have been exercised by the validation. The GrADS project has developed a testbed called the MacroGrid, used for, among other things, development and testing of new grid technologies. The MacroGrid currently includes over 100 machines (see Figure 2), of which the following 10 machines were used in our studies.

- **UCSD cluster**: three 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 Figure 2. For local area networks the NWS measured bandwidth via 64 KB messages; for wide area networks the NWS used 1 MB messages.

For our experiments, we label our test cases **GrADS Grid**, including 3 machines from each of the 3 sites listed above, and a **GrADS Cluster**, including just the 4 machines from the UTK cluster. We selected a relatively small subset of the GrADS testbed for experimentation for two reasons: we needed consistent access to the testbed machines (quite difficult for larger testbed subsets), and we needed application executions that could be emulated in a timely manner on the MicroGrid (emulation speed is partly dependent on the size of the testbed).

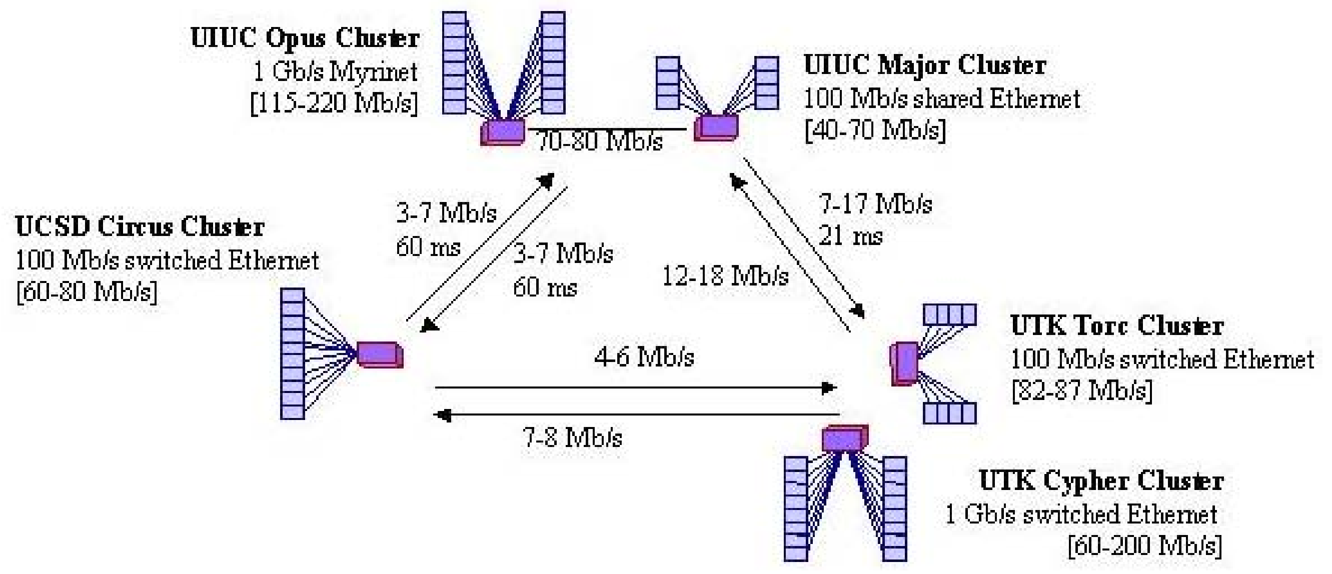

Figure 2. GrADS testbed with network links annotated with bandwidth and latency values collected with the NWS.

4.1.2 MicroGrid emulation configuration

Several steps are necessary to build an emulation environment in the MicroGrid. The first steps involve configuring the emulation environment for a particular testbed. We only had to perform this step once each for the GrADS Grid and GrADS Cluster; the emulation environments could then be used for each of the five applications with only a small change to processor speed assumptions, as explained in the next paragraph.

First, the set of machines to be included in the emulation has to be defined in a configuration file and each machine should be annotated with its processor speed and memory capacity. We wanted to emulate a variety of AMD and IA32 processors, while the MicroGrid compute platform contained only one type of IA32 processors. To obtain a mapping between processors, we selected basic integer and floating point serial benchmarks from the NAS Parallel Benchmark Suite [10] and then ran them on each architecture in the GrADS Grid and Cluster and on the MicroGrid compute platform processors. We then classified each of our grid applications as predominantly integer or floating point and used the respective benchmark results to provide an appropriate machine speed mapping in the MicroGrid emulation.

Next, the network connecting the nodes must be defined and virtualized on top of an existing network. The MicroGrid virtual network topology is defined by latency and TCP sender window size. Based on NWS data collected from the GrADS testbed, we assume all the machines have 64KB TCP window size and we assume the following latencies between hosts: 31ms between UCSD and UIUC hosts; 30ms between UCSD and UTK hosts; and 11 ms between UIUC and UTK hosts.

At the time of our experiments the MicroGrid did not support injection of load traces; thus, for these initial experiments we sought relatively unloaded processors in the real-world and then modeled the environment as unloaded in the MicroGrid. Of course we were not able to obtain such control for the networks, leading to one source of error for our validation experiments.

To prepare MPI applications for execution on the MicroGrid, the application does not need to be modified; the application is simply linked with MicroGrid libraries. A small configuration file must also be written to assist the MicroGrid in finding all necessary files during the emulation launch process. Note that the application and application configuration files can refer directly to real-world machine names or IP addresses; the MicroGrid virtualization process translates these names into appropriate physical addresses on the compute platform.

4.1.3 Results

The results of our validation experiments are shown in Figure 3 for the GrADS cluster and in Figure 4 for the GrADS Grid. We did six repetitions of each application emulation; when we report emulated results we report an average of the six runs and error bounds based on standard deviation. Repeatable runs were much harder to obtain in the real-world

	GrADS Cluster	GrADS Grid
ScaLAPACK	36.8%	34.2%
Jacobi	28.1%	23.6%
Fish	26.7%	-15.1%
Game of Life	16.8%	-0.4%
FASTA	15.6%	59.0%
Average	24.8%	26.5%

Table 1. Summary of MicroGrid prediction errors.

tests and we only obtained between one and three repetitions of each experiment. We do not have sufficient real-world runs to calculate a meaningful average and standard deviation; instead, for the real-world results we selected the minimum measured time (minimum because we sought unloaded conditions).

Table 1 summarizes the percentage difference between the real-world execution time and that predicted by the MicroGrid emulation. Overall, the MicroGrid provides a reasonable estimate of predicted performance, especially given the fluctuating loads and other uncertainties of execution in the real-world that can not be captured in emulation. It is encouraging that the MicroGrid properly ranks the performance of all of the applications in the cluster environment and of Jacobi, Fish, and Game of Life in the grid environment. In other words, those results could inform a user as to which applications will run fastest in these environments.

The results are disappointing for ScaLAPACK in both environments and for FASTA in the grid environment. ScaLAPACK involves many small messages and complicated communicated patterns; we plan to run further tests to determine why these patterns are difficult to emulate accurately. FASTA is the only application involving significant I/O (for reading databases) and also involves load-imbalance among processors. These factors do not explain however why the emulation would be reasonably accurate for the cluster case and not the grid case. We plan to investigate these issues for the final version of this paper.

Another interesting point is that because we did not emulate competing load in the MicroGrid we expected the MicroGrid to consistently under-predict execution times. In fact, the reverse is true. This leads us to believe that the MicroGrid emulation may introduce some overheads into the application execution that are not properly accounted for currently. We plan to run some experiments to better understand and account for the overheads associated with the emulation process.

In general, although we plan to improve the MicroGrid validation, we feel that these initial results are promising. As users do not currently have any advance information of the application performance they can expect on new topologies, information at this level of accuracy is already useful.

4.2 High-performance grid architectures

In this set of experiments, we demonstrate the application of the MicroGrid for testing the performance of applications on grid architectures that are not available for real-world experimentation. This application of the MicroGrid could prove useful for a variety of reasons: the grid could exist but be unavailable to the user currently, it could be impossible to perform repeatable, controlled experiments, or the topology could even be an imagined grid architecture including higher performance components than are currently available.

In these experiments, we emulate faster and richer resources than the GrADS Grid and Cluster used in the previous section. We investigate a wide range of types and performance for networks, CPUs and storage. The immediate goal of these experiments is to discover whether these grid architectures are well suited to the five applications described in Section 3. Note that since we do not have access to real-world versions of these topologies we cannot directly validate these performance predictions. We refer to the experiments in the previous section as initial evidence that the MicroGrid provides reasonable performance predictions for these five applications.

4.2.1 Testbeds

The **TeraGrid** is a large-scale grid project that combines large supercomputer style resources from across the US [30]. The initial TeraGrid design, planned for operation in 2003, includes five large-scale Linux clusters at ANL, Caltech, NCSA, PSC, and SDSC. At the time of our experiments, the TeraGrid was not yet available as a production grid. We therefore emulate the TeraGrid to provide advance results.

In our experiments, we use two machines from each of the five clusters, which are connected by long wide area links (5-30ms) and multiple switch/routers and 10Gbps links. For the local resources, we make the following performance assumptions: each node has 3GHz CPU, 1GB memory and 1Gbps network interface, the TCP window size is 1MB. These numbers do not match the current TeraGrid specification exactly as we configured our emulation based on projected hardware specifications before the actual platform was put in place.

The Optiputer project seeks to exploit high bandwidth DWDM optical networks to support data-intensive scientific research and collaboration. The communication links are dedicated lambdas with guaranteed high bandwidth and low latency. They may run TCP or new protocols under

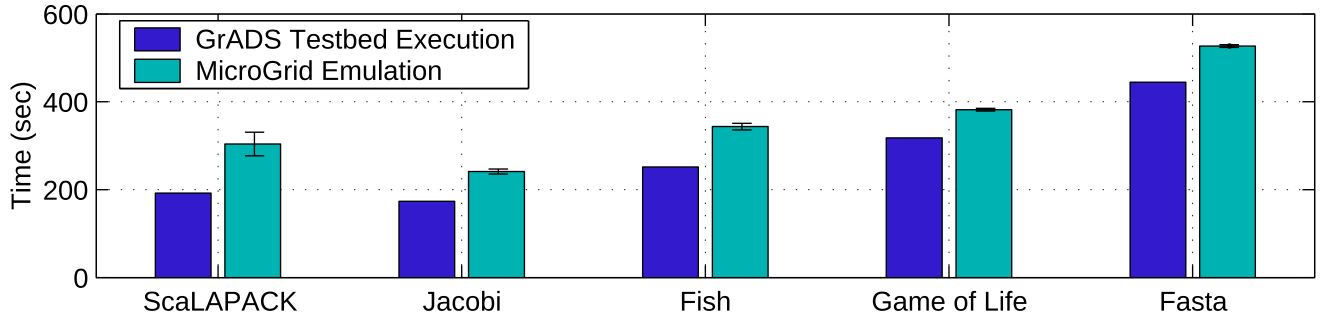

Figure 3. Comparison of application execution times on the GrADS Cluster and as measured in a MicroGrid emulation of the GrADS Cluster.

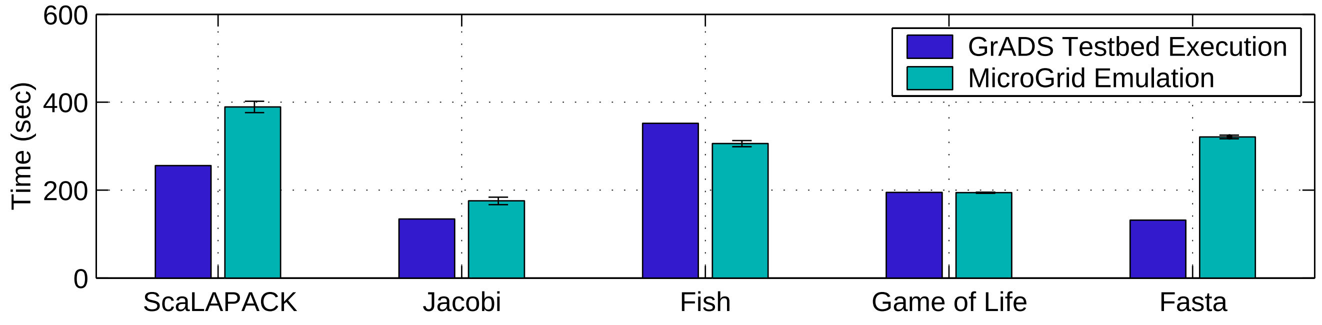

Figure 4. Comparison of application execution times on the GrADS Grid and as measured in a MicroGrid emulation of the GrADS Grid.

development. We approximate the future Optiputer environment using the following network configuration.

Three sites are connected through 80Gbps switched optical backbone network; each site consists of four machines with 3GHz CPU and 2GB memory, each of them connects to the optical core router in 4Gbps. Two configurations vary the latency between the sites. **Optiputer1** has 5ms and **Optiputer2** has 60ms. To simulate the dedicated optical connections, we assume that all the nodes use infinite TCP send windows.

The **Optiputer3** models a combined Optiputer and Internet-2 grid environment. Two sites are connected through 80Gbps optical backbone network with high bandwidth and moderate (10ms) latency; each site consists of four machines with 3GHz CPUs and 2GB memory. Each machine connects to the optical core router in 4Gbps. A third site is from another backbone network with 2.4Gbps bandwidth and 40ms latency to the first two sites. This site has four machines, also with 3GHz CPU and 2GB memory, but with a 1Gbps switch and 128KB TCP window.

4.2.2 Results

Figure 5 shows the execution times measured when we ran our applications on the novel architectures described above. While these quantitative results are in themselves interesting to grid users, we can make the following two observations:

- Different applications run well on different architectures and the MicroGrid makes it possible for users to predict which architecture will be best for their applications. There is no single architecture that is best for all applications.
- The different between the Optiputer1 and Optiputer2 results make it possible to directly evaluate the impact of network latencies on the application executions (in this case an increase in latency by a factor 12). For instance, this shows that while ScaLAPACK is highly impacted by network latencies, Game of Life is not.
- For the ScaLAPACK application, the communication time dominates the overall running time. This is because the application has a large number of small com-

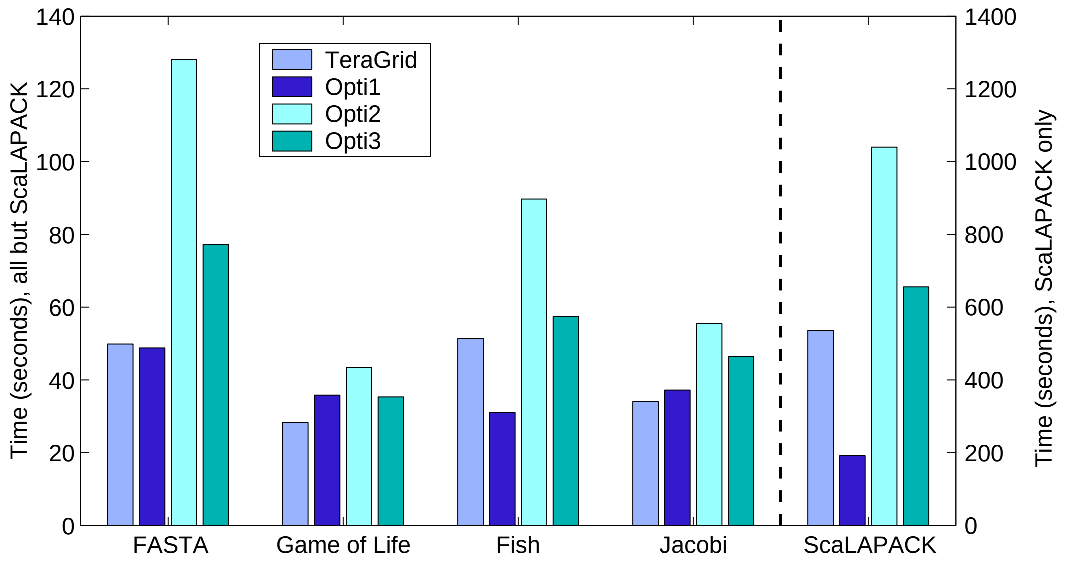

Figure 5. Forecasts of application behavior in new, high-performance grid architectures based on MicroGrid emulations. The right-hand y-axis scale corresponds to the ScaLAPACK application only while the left-hand y-axis scale corresponds to the other four applications.

munications, which makes network latency the main performance factor. Furthermore, we use a relativeley small problem size (6000×6000), so that the fast CPUs in the high performance testbed should not provide much benefit for the application. This is confirmed in the emulation results for ScaLAPACK showin both in Figure 4 (for the GrADS testbed) and in Figure 5 for the four novel architectures. Indeed, the emulation results show that the application execution time is roughly proportional to the network latencies, with the optiputer2 configuration being the worst with 60 ms latencies.

5 Related Work

Three methods have been used to perform grid experiments: live networks, simulation, and emulation.

Live networks use some specific set of real-world resources for experiments, which of course provides a realistic grid environment for experimentation. However, this method has two main limitations: (i) it is usually difficult to run experiments for a wide range of platform scenarios or for platforms that do not exist yet; and (ii) it is often impossible to predict and control the environment with shared networks and computers, which precludes reproducible experiments.

Another method of choice is simulation. A number of software tools provide general-purpose discrete-event simulation capabilities by which one can implement a model [27, 32, 23, 12, 16, 17]. Some tools are specifically targeted to the domain of grid application scheduling and provide convenient high-level abstractions [8, 20]. While these simulation tools make it possible to evaluate scheduling algorithms in an abstract sense, they often cannot simulate what would truly happen on a real system (e.g., software and I/O overheads, realistic network congestion).

Several lower-level simulation models provide more accurate network environment, like NS [7], GloMoSim [2], and DummyNet [26]. However, these tools only capture part of what is relevant to grid applications, i.e. the network resources.

An alternative to simulation is emulation, which is virtualizing simulated resources on real resources. Emulation tools enable coupling with live application programs and provide a more realistic environment than simulation. Besides the MicroGrid, the Albatross project [19], Emulab [34] and Modelnet [33] all attempt to run live application using a network simulator/emulator. However, their network emulator cannot support sophisticated network protocols, such as different routing protocols; they all use real-time emulation and cannot control the emulation speed freely. In contrast, the MicroGrid network emulator implements the full stack of OSPF and BGP4 routing protocols and has better accuracy; the MicroGrid CPU controller enables emulation of virtual computers with arbitrary speeds.

6 Conclusion

The increasing acceptance of grid computing in both scientific and commercial communities presents significant challenges for understanding the performance of applications and resources. No more are the associations simple and static, and dynamic resource sharing and application adaptation add new variables to the situation.

We have described a new tool, the MicroGrid, and both its validation and a demonstration of the new methodology it enables for studying performance questions involving grid applications, middleware, and network and edge resources. These capabilities represent a wide range of new opportunities for understanding grids.

Acknowledgements

The authors would like to thank the rest of the MicroGrid and GrADS teams for their help and comments on this work. The test applications used in the paper and the experimental testbed were developed in the GrADS collaboration.

References

[1] S. Agrawal, J. Dongarra, K. Seymour, and S. Vadhiyar. *Grid Computing: Making The Global Infrastructure a Reality*, chapter NetSolve: Past, Present, and Future - A Look at a Grid Enabled Server. John Wiley, 2003. Hey, A. and Berman, F. and Fox, G., editors.

[2] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. GloMoSim: A Scalable Network Simulation Environment. Technical Report 990027, UCLA Computer Science Department, 1999.

[3] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[4] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software support for high-level Grid application development. *International Journal of Supercomputer Applications*, 15(4):327–344, 2001.

[5] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software support for high-level Grid application development. *International Journal of High Performance Computing Applications*, 15(4):327–344, 2001.

[6] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[7] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, 2000.

[8] 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)*, 14(13-15), 2002.

[9] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulations. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, June 1999. Las Vegas, Nevada.

[10] D. H. Bailey and E. Barszcz and J. T. Barton and D. S. Browning and R. L. Carter and D. Dagum and R. A. Fatoohi and P. O. Frederickson and T. A. Lasinski and R. S. Schreiber and H. D. Simon and V. Venkatakrishnan and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, 1991.

[11] H. Dail, F. Berman, and H. Casanova. A decoupled scheduling aproach for grid application development environments. *Journal of Parallel and Distributed Computing*, 63(5):505–524, May 2003.

[12] F. G. et al. Simkit: A high performance logical process simulation class library in c++. In *Proceedings of the 1995 Winter Simulation Conference*, 1995.

[13] FASTA package of sequence comparison programs at `ftp://ftp.virginia.edu/pub/fasta`.

[14] G. W. Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press, Cambridge, MA, 1998.

[15] The GrADS project. `http://hipersoft.cs.rice.edu/grads`.

[16] F. Howell and R. McNab. 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.

[17] F. Howell and M. 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*, Jan 1998.

[18] C. Kesselman and I. Foster. The Globus Toolkit. In C. Kesselman and I. Foster, editors, *The Grid: Blueprint for a New Computing Infrastructure, edited by Carl Kesselman and Ian Foster*, pages 259–278. Morgan Kaufmann Publishers, Inc., 1999.

[19] T. Kielmann, H. Bal, J. Maassen, R. van Nieuwpoort, L. Eyraud, R. Hofman, and K. Verstoep. Programming environments for high-performance grid computing: the albatross project. *Future Generation Computer Systems*, 18(8), 2002.

[20] L. Legrand, A. 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)*, 2003.

[21] J. Liu and D. M. Nicol. DaSSF 3.0 User's Manual, January 2001. available from http://www.cs.dartmouth.edu/research/DaSSF/docs.html.

[22] X. Liu and A. Chien. Traffic-based load balance for scalable network emulation. In *Proceedings of Supercomputing Conference*, Phoenix, Arizona, November 2003.

[23] 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.

[24] W. Pearson and D. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85:2444–2448, 1988.

[25] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerical libraries and the Grid: The GrADS experiment with ScaLAPACK. *International Journal of High Performance Computing Applications*, 15(4):359–374, 2001.

[26] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[27] H. Schwetman. Csim: A c-based, process oriented simulation language. In *Proceedings of the 1986 Winter Simulation Conference*, 1986.

[28] O. Sievert and H. Casanova. A simple MPI process swapping architecture for iterative applications. *International Journal of High Performance Computing Applications (IJHPCA)*, 2004. To appear.

[29] H. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: a scientific tool for modeling computational grids. In *Proceedings of Supercomputing Conference*, November 2000.

[30] TeraGrid. `http://www.teragrid.org/`.

[31] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley, 2003. Hey, A. and Berman, F. and Fox, G., editors.

[32] S. Toh. Simc: A c function library for discrete simulation. In *Proceedings of the 11th Workshop in Parallel and Distributed Simulation*, 1993.

[33] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. *ACM SIGOPS Operating Systems Review*, 36, 2002.

[34] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the 5th Symposium on Operating Systems Design and Implementation, Boston, MA*, 2002.