Activities and Findings

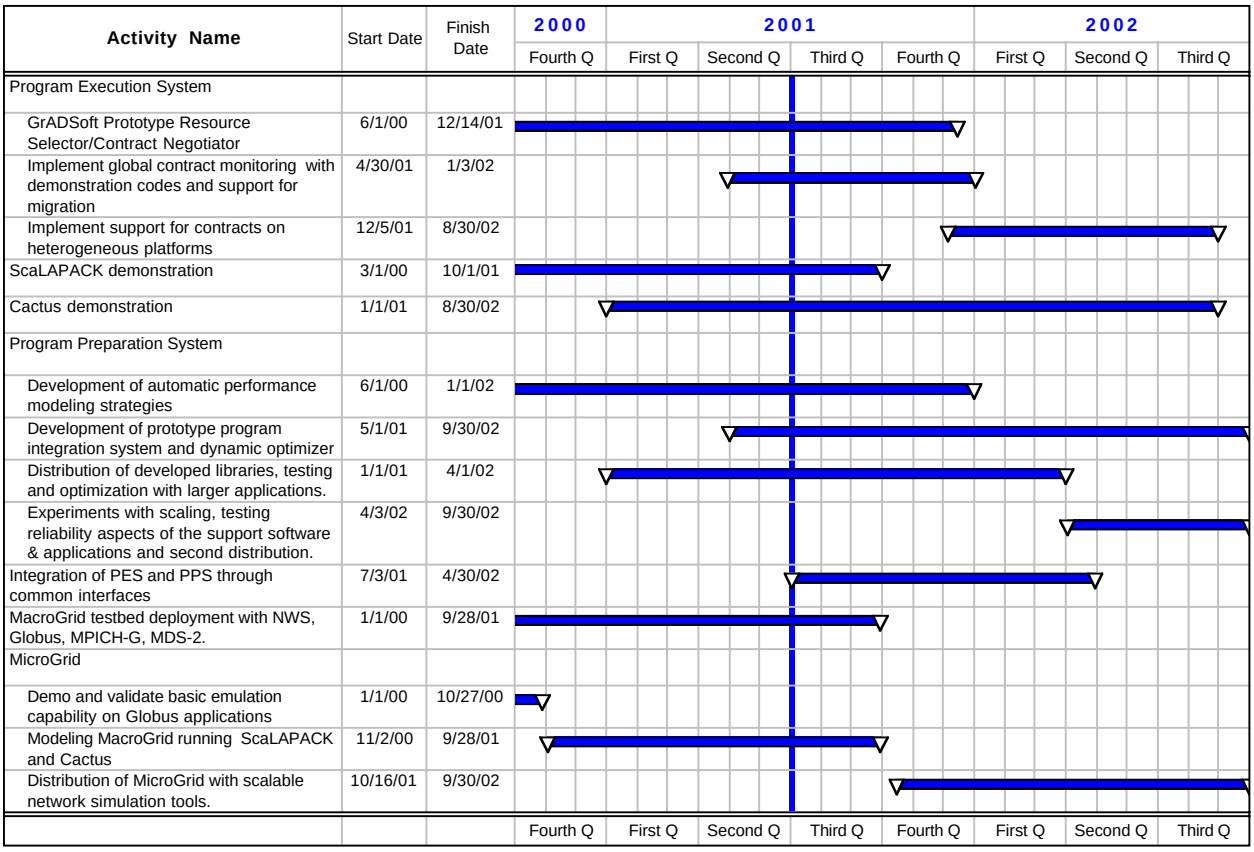
I. Project Activities

The goal of the Grid Application Development Software (GrADS) project is to simplify distributed heterogeneous computing in order to make Grid application development and performance tuning for real applications an everyday practice. Research in key areas is required to achieve this goal:

- Grid software architectures that facilitate information flow and resource negotiation among applications, libraries, compilers, linkers, and runtime systems;
- base software technologies, such as scheduling, resource discovery, and communication, to support development and execution of performance-efficient Grid applications;
- policies and the development of software mechanisms that support exchange of performance information, performance analysis, and performance contract brokering;
- languages, compilers, environments, and tools to support creation of applications for the Grid and solution of problems on the Grid;
- mathematical and data structure libraries for Grid applications, including numerical methods for control of accuracy and latency tolerance;
- system software and communication libraries needed to make distributed computer collections into usable computing configurations; and
- simulation and modeling tools to enable systematic, scientific study of the dynamic properties of Grid middleware, application software, and configurations.

During the current reporting period (7/1/00-6/30/01), GrADS research focused on the six inter-institutional efforts described in the following sections: *Program Execution System (PES), Program Preparation System (PPS), ScaLAPACK Experiment, Cactus Experiment, MacroGrid*, and *MicroGrid*. Project publications and additional information can be found at http://www.hipersoft.rice.edu/grads. Project design and coordination were enabled through regular PI and subproject teleconferences, PI meetings (7/24/00, Argonne; 11/8/00, Dallas; 1/22/01, ISI; 5/1/01, Rice), and workshops (7/24-25/00, Argonne and 1/22-23/01, ISI). The next GrADS workshop and PI meeting is scheduled for 7/9-10/01 at Rice. In addition, a GrADS NSF project review (4/30/01) was held at Rice.

In the process of project management we have developed a Gantt chart for major project activities that are reported in this section. This chart is included on the next page. Activities are only shown over the current and next fiscal years.



1. Program Execution System (PES)

The Program Execution System (PES) works in cooperation with the Program Preparation System (PPS) to (1) identify resources appropriate to application needs, (2) map the application onto the resources, (3) initiate application execution and performance monitoring, (4) detect when actual performance does not meet expected performance, and (5) reconfigure the application or system when appropriate to achieve the desired performance, if possible.

During the past year we have continued to refine our understanding of the requirements to realize the scenario described above. Guided by ongoing discussions with other members of GrADS and parameterized experimentation with applications such as ScaLAPACK and Cactus, we have implemented and refined prototypes for various components, begun the design and coding of the component interfaces and infrastructure, and conducted basic research to support future development efforts. Some of these activities are detailed in the following sections.

1.1 General Infrastructure Development

In collaboration with members of the PPS group, we are designing a software architecture called *GrADSoft* that provides a generic framework for information sharing and negotiation among the GrADS program development and execution components. In this architecture, an *Application manager* coordinates activities of other components by storing information and invoking services. Our first prototype of this architecture is approximately 10,000 lines of code and implements major portions of the architectural design. The prototype is proving to be a useful

tool not only for managing component interactions, but also as a development framework for core scheduling technologies.

The architecture design document is GrADS Working Document III, *GrADSoft – A Program-level approach to using the Grid*. Code documentation for the prototype implementation is found at http://gridlab.ucsd.edu/~grads/GrADSoft_htmldoc.

1.2 Resource Selection and Scheduling

We are exploring run-time adaptive scheduling technologies to support resource selection and mapping for GrADS-enabled applications. To match applications with appropriate resources dynamically, the scheduler must adapt to user-specific scheduling parameters, application resource requirements, and dynamic resource availability.

The availability of accurate information on users, applications, and environment is vital for resource selection and scheduling. Hence, we have developed interfaces to support the necessary communication mechanisms among GrADS components. In conjunction with the MacroGrid efforts, we are developing improved mechanisms for Grid resource information collection and communication. In the PPS compiler efforts, we are exploring automatic application analysis and characterization. In the PPS problem solving environment efforts, we are exploring new user interfaces.

In addition to our work on information exchange, we are developing new scheduling methodologies targeted to the Grid. We have investigated three general scheduling methodologies and have implemented functional schedulers for each:

- For the ScaLAPACK effort, we modified the library to include accurate resource requirements and performance models. Resource data (e.g., processor availability or bandwidth predictions) is now collected in real-time and utilized by scheduling code incorporated into the library. This effort demonstrated that, given accurate application and resource availability data, we could achieve good application performance in a Grid environment.
- For the Cactus application effort, we deployed a resource selector as a stand-alone GrADS service that could be offered anywhere on the Grid. We incorporated automatic adaptation to application characteristics and user scheduling preferences and requirements. This scheduler was developed within the GrADSoft framework, ensuring its interoperability with other GrADS components.
- We have defined a characterization method for describing resource requirements for iterative, mesh-based applications, and are currently developing a scheduler that employs this characterization to adapt to a variety of application requirements. In preparation for automatic compiler application characterization, we are also investigating best-effort scheduling given variable levels of application information. This work will be the focus of Dail's M.S. thesis at UCSD.

1.3 Performance Contract Specifications and Monitoring

Performance models derived from a variety of sources, including application and library developer knowledge, compilers, and execution history, are used to map applications to resources. Once resources have been selected and the application has been mapped to those resources, a *performance contract* based on the model states the expected performance of the application on the allocated resources. During execution, application performance measurements are compared to the contract specifications by the contract monitoring system. When observed performance deviates beyond a certain level of tolerance from the expected performance, a contract violation is signaled and reconfiguration can be triggered.

During the past year, we have refined the notion of a performance contract and enhanced the software infrastructure supporting contract monitoring. Autopilot sensors inserted in the application capture performance measurements and make them available to contract monitor module(s) responsible for detecting performance problems. In turn, fuzzy logic rules as part of the contract monitor system control when a contract is violated. The tolerance of these rules can be adjusted to accommodate known uncertainty in the model, expected variation in the resource capabilities, and "small" performance fluctuations that would not warrant a reconfiguration of the execution. In addition to flagging a violation, the rules are designed to help detect the cause of the violation (e.g., determining which processor is overloaded or which network is slow).

The performance contract specification and monitoring system has been used with a developer-supplied model for the ScaLAPACK application. Monitoring output showed that the initial model was not a good predictor of actual performance in the Grid environment, and suggested how the mapping of the application to heterogeneous Grid resources impacted loop iteration performance. The developers have now substantially revised the model to address the initial shortcomings.

In addition to work with developer-supplied models, we have formulated an *application signature* performance model. This model uses historical information about application resource demands, together with estimates of resource responses to those demands to predict *execution metrics* that express rates of progress.

Autopilot sensors in the executing application periodically capture and advertise metrics (e.g., *messages/second* and *instructions/second),* using MPI profiling wrappers and the PAPI hardware counter interface to collect the data. Contract monitors on other Grid resources receive these measurements and compare them to the values predicted by the model. The contract monitor output indicates the level of violation as determined by the rule base controlling the comparison.

We tested the application signature model with the ScaLAPACK application and with a synthetic master-worker code. In both cases, through comparison of the actual and expected execution metrics, the contract monitor was able to detect when an additional load was placed on a resource in use by the GrADS-enabled application. This model shows promise when developer or compiler models are unavailable. In addition, experiments with this model allowed us to test our monitoring infrastructure and fuzzy logic decision procedure with a different contract scenario.

GrADS Working Document IV, *Specifying and Monitoring GrADS Contracts*, discusses GrADS contracts and the monitoring infrastructure. A paper on the application signature performance model has been submitted for publication. In addition, this work will be the focus of Vraalsen's M.S. thesis at UIUC.

1.4 Execution Reconfiguration

We are also exploring *execution reconfiguration techniques (i.e.,* the adaptation of an executing Grid program to its environment to meet the expectations of a performance contract). We are investigating the basic concepts and techniques used by the *renegotiator,* the entity whose job is to dynamically adapt program execution.

Throughout execution, the renegotiator will answer the following questions:

- Is the application achieving expected performance on the current resources?
- Can we utilize the current resources more effectively?
- Are better resources available?
- Should we continue on the current resources or switch to other resources?

The renegotiator has access to run-time information about the application and the computational resources. From this information it can choose to dynamically reconfigure execution by:

- Removing, adding, or replacing computational resources
- Redistributing work on the computational resources
- Altering the communication topology of the computational resources
- Modifying application behavior (e.g., decreasing frame rate when resources are heavily loaded)

This work will be the focus of Sievert's M.S. thesis at UCSD. GrADS Working Document II, *Contract Renegotiation,* provides additional details.

1.5 G-commerce

Performance contracts provide a mechanism for implementing a mapping between application requirements and resource capabilities and for monitoring and reconfiguring the application execution to achieve reliable performance. As part of the GrADS project, we are also investigating the policies under which performance contracts should be established. Collectively, these policies are generally termed a *performance economy.* We call performance economies for GrADS *G-commerce.*

The use of performance feedback to control application execution, in effect a distributed control system, introduces the possibility of oscillation under certain operating conditions. To ensure that resources are not wasted on feedback-induced oscillation, we are investigating G-commerce, computational market formulations for GrADS and the Grid.

G-commerce focuses on defining and modeling the basic rules that must be observed by GrADS-enabled applications, each of which is using performance contracts to control its dynamic resource mappings. Through the use of mathematical principles developed for the analysis of real-world markets, we are gaining an understanding of the contracting rules that should govern resource allocations to maintain stability and equilibrium. We plan to incorporate successful G-commerce economic formulations into the GrADS PES framework to ensure that GrADS resource utilization will remain efficient from a global perspective without the need for a global control mechanism.

A paper on this work, *G-commerce: Market Formulations Controlling Resource Allocation on the Computation Grid*, was presented at IPDPS in April 2001.

2. Program Preparation System (PPS)

The overarching goal of the GrADS Program Preparation System is to make it easy for end users to develop applications for the Grid. To achieve this goal, the PPS effort must address several significant challenges:

1. *Programming Environment:* We must develop convenient programming models and programming support systems that hide many of the complexities of Grid programming from the end user. To this end, we have focused on a long-term strategy involving the construction of domain-specific problem-solving systems from libraries coded by experts in the problem domain working with experts on Grid programming. The framework we are developing, called *telescoping languages,* would permit the integration of complete, efficient applications from scripts that invoke underlying domain-specific components.

2. *Libraries:* For use in our high-level programming environments, we must develop libraries that encapsulate common components from a variety of domains and that employ Grid-aware algorithms.

3. *Configurable Object Programs:* We must elaborate the notion of a *configurable object program*, which would serve as the principal input to the GrADS Program Execution System (PES). Such an object program must include not only code, but also flexible *mapping strategies* that can be used to adapt the program to different grid configurations. In addition, the object program must include *Grid performance models* that can be used to select the right Grid resources for execution and to support the performance monitoring process, helping to determine when the execution should be interrupted and reconfigured. We must develop compiler systems capable of integrating configurable object programs from Grid-aware components.

4. *Dynamic Optimizer:* We must develop strategies for preparing programs for execution once the Grid resources on which they will execute are known. This process must implement the mapping strategy, build correct and optimized object programs for the various computational components, insert probes and sensors that support the

performance monitoring process, and perform any last minute optimizations needed to facilitate the distributed execution.

Although our activities over the past year have touched on all of these areas, most of the effort has been concentrated on the interaction with the execution systems through configurable object programs. In addition, we have been active participants in the preliminary experiments, particularly ScaLAPACK. To that end, we have helped to establish the GrADS experimental facilities, particularly the MacroGrid.

2.1 Programming Environment

The focus of work on the programming environment has been to investigate high-level Grid programming interfaces. A principal goal is to make it possible to develop script-based application development mechanisms that produce useful applications with acceptable performance. This work has proceeded in two major thrusts:

1. *Telescoping languages:* We are developing a framework for generating high-level, domain-specific languages from domain-specific, grid-enabled libraries. A goal of this work is to provide a flexible programming interface that shields the user from much of the complexity of Grid programming while constructing applications that achieve acceptable performance levels.

2. *Simple scripting interfaces to remote Grid computations:* This work, carried out primarily at Indiana, is attempting to make it possible for the user to construct local scripts that invoke remote software components to solve problems.

These two efforts are elaborated in the following paragraphs.

Telescoping Languages. We have begun the development of a simple framework for a generation of high-level programs based on Matlab. The basic idea is, for a specific domain, to augment Matlab with a collection of primitive operations that are implemented by Grid-aware components developed by experts in the application domain. Initially these components would be by-products of our application partnerships with ScaLAPACK and Cactus, but eventually we envision working with the libraries effort to construct whole collections of components that could be used in a variety of contexts.

Telescoping languages is a general framework for the development of high-level software systems. In the context of Grid programming, there are a number of technical challenges to applying the ideas. First, the application integration step must produce a configurable object program with many of the decisions left unmade (see Section 2.3). Second, much of the compiler generated by this framework will run in the dynamic optimizer, so routines specialized for specific architectures will not be selectable until the execution resources are known. These challenges will be addressed more specifically as we better understand the process of building configurable object programs.

We have begun the development of the Matlab infrastructure and have developed the concepts of telescoping languages in a series of three papers, cited below. The reported work includes some entirely original optimization strategies for telescoping languages.

Related Publications:

"Telescoping Languages: A Compiler Strategy for Implementation of High-Level Domain-Specific Programming Systems," K. Kennedy, *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, (May 2000).

"Reduction in Strength of Procedures: An Optimizing Strategy for Telescoping Languages," A. Chauhan and K. Kennedy, *Proceedings of the 2001 International Conference on Supercomputing*, Sorrento, Italy (June 2001).

"Telescoping Languages: A Strategy for Automatic Generation of Scientific Problem-Solving Systems from Annotated Libraries," K. Kennedy, B. Broom, K. Cooper, J. Dongarra, R. Fowler, D. Gannon, L. Johnsson, J. Mellor-Crummey, and L. Torczon, accepted for publication in *Journal of Parallel and Distributed Computing*.

Simple scripting interfaces. The approach taken at Indiana subdivides the task into three parts:

1. The "main" part of the computation is a script that runs on the user's workstation. The script has the responsibility of allocating any required remote resources. (In its current form, this is a static allocation. In the year ahead, this allocation will be based on a request/negotiation with the GrADS services.) When the resources are allocated, the script uses Globus to launch remote software components on the allocated resources.

2. The remote components of the application consist of the computationally intensive parts of the code that are best executed on the remote resource. In some cases these may be data-intensive parts of the computation that are done remotely because of the location of the data source. Each remote component has a function interface that can be invoked remotely.

3. The "main" script begins execution of the program. When the remote computations are required, the "main" part of the computation makes a remote procedure call to invoke the remote service component. When the main computation completes, the remote components are terminated. We have used this model with two test applications. In one case, the application was a computational chemistry experiment where the two remote computations were a finite element simulation and a Monte Carlo simulation. The main program linked these two computations to build a multiscale simulation of copper deposition on silicon in semiconductor manufacturing. The other application involved testing sparse matrix solvers. In this case, a finite-element based PDE was solved by distributed computation by having a "matrix assembly" component on one machine and a sparse matrix solver on another. The main program provided the iteration and convergence control.

These simple experiments demonstrated that decomposing a computation along the lines of functionality of the program submodules can be effectively used in the Grid environment. NetSolve exploits the same concept very effectively. The script execution and remote component method invocation is an effective “assembly language” for higher level “telescoping” language implementations.

Related Publications:

“Requirements for and Evaluation of RMI Protocols for Scientific Computing,” Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley, and Dennis Gannon, *IEEE SC2000*.

“A Component Based Services Architecture for Building Distributed Applications,” R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri, *Proceedings High Performance Distributed Computing Conference 2000*.

2.2 Libraries

Most of the libraries effort has been focused on the ScaLAPACK demonstration project described in Section 3. In the long run, we expect the application collaborations to spin out library components based on the common elements from a community-based application effort such as ScaLAPACK or Cactus.

Grid-aware library components will need to be able to deal with access latencies that vary widely with resources and load. To address this issue, we have begun an investigation of latency tolerant approaches for Grid computing. Work by others in the project measuring the Grid-enabled ScaLAPACK benchmark showed that its synchronous algorithms periodically lead to long delays when a processor requires data from nodes on the far side of slow communication links. A key requirement for ScaLAPACK and other codes is support for collective communication. We have begun to investigate how to best perform collective communication in a wide-area environment. We have begun exploring how to orchestrate collective communication operations to accommodate latency, bandwidth, and connectivity limitations that are unavoidable characteristics of wide area networks. Currently, we are developing an algorithm to select an optimal spanning tree to minimize the latency of a broadcast in a wide-area network.

2.3 Configurable Object Program

An important goal of the PPS project is to define the nature of a configurable object program and develop compiler technology that can assist the user in building them. To address this issue, we will need to construct mapping strategies and performance models that can interact with the PES and the contract monitoring mechanisms. Much of the work over the past year has focused on two topics: performance model construction and interfaces to the PES.

Model Construction. A goal of the program preparation system is to produce one or more models of an application that will serve as the basis for determining resource requirements, selecting resources, and mapping the application to available resources. For the PES to be able to

evaluate alternative sets of resources, the PPS must provide it with a model that describes the desired virtual machine topology, approximate memory requirements per node, computation cost, communication volume, and a measure of the application's communication latency tolerance. Each of these characteristics poses a constraint that can be used to determine whether a node is suitable for inclusion in a requested virtual machine topology.

To be generally usable, the development of the model should be automated or semi-automated (e.g., annotation languages could be used to capture expert knowledge). In libraries we have examined, it appears that the library routines can be reasonably described fully via annotations, avoiding model construction each time. However, mixed specifications (part model, part annotation knowledge) raise the issue of model composability, which will be important for full applications.

We have initiated investigation of how to create scalable, architecture-independent application performance models of computation. To map a computation to a collection of heterogeneous nodes, one must determine how large a piece of the computation to map to each of the available compute nodes. Having a precise understanding of how the application will perform on each architecture is a key to effective computation partitioning. To this end, we are working to develop a tool that combines data from sample executions, information about the architecture on which this data was measured, and information about the program's structure to develop an architecture-independent model of program performance. Our work in this area leverages infrastructure we developed under DOE funding for extracting information about a program's loop nesting structure from application binaries, and correlating data measured from hardware performance counters with the program's loop nesting structure. We have collected a variety of run-time performance metrics including cycles, cache misses at various levels, and floating point computation for a sample application (a parallel semi-coarsening multigrid solver) using a variety of data sizes. We have begun the process of trying to synthesize these metrics into an accurate model of application performance. This effort has drawn heavily on the insights developed through the ScaLAPACK experiment.

Interfaces to the PES. In an attempt to get an integrated GrADS prototype system up and running, the PPS project has been working with the PES group to develop a detailed interface specification, and an overall design for the integrated run-time system, in which PPS models can be populated initially with handcrafted components that embody expert knowledge.

The ScaLAPACK and Cactus experiments have helped us to address two fundamental questions facing the development of the PPS:

- The PPS must adopt a compilation model that allows the PES to tailor the application's execution to the resources at hand, and to reconfigure the application as available resource performance varies dynamically.
- The PPS and the PES must be able to exchange information via a flexible and well-defined interface, which we have begun to define.

To this end, we have collaborated with UC San Diego on the design and development of an Application Manager interface between the PPS and the resource allocator. This interface enables the PPS and developers to provide information about program requirements to the

resource allocator. We have designed and implemented a representation for specifying an Abstract Application Resource Topology (AART), an application-centric virtual machine that includes descriptions of properties of compute nodes (e.g., available compute power, memory, and local disk storage), their logical interconnection topology, and properties of the communication links connecting them (e.g., average observed latency or available bandwidth). Properties can be specified as minimum requirements or desirable characteristics. Relative importance of desirable characteristics can also be specified. The Application Manager interface also enables the resource allocator to invoke an application-specific performance model.

We have also designed an XML-based external representation for an application-centric virtual machine and implemented a parser that reads this representation and instantiates its corresponding dynamic representation, which can be passed to the resource allocator.

We have developed a virtual machine representation that allows the PES to communicate the relevant performance characteristics to the PPS. At present, the ScaLAPACK and Cactus runtime schedulers use this representation to perform resource selection and mapping. However, we designed this representation with the PPS in mind, and the next step is to consider PPS-driven enhancements. Of equal importance is the PPS-to-PES representation. While we do not yet have a working prototype, we have developed an initial design, which will be developed into such a prototype.

2.4 Dynamic Optimizer

The role of the dynamic optimizer is to tailor the configurable object program to achieve the best possible performance on Grid resources once they are assigned and to insert probes and sensors required by the performance monitoring process. Achieving this goal requires a careful interaction with the compilation system that produces the configurable object program, along with a strong interaction with the PES.

Given that the resource performance that will eventually be available to the program is not known at compile time, the compilation system is faced with two choices. It can either compile several versions of the program, each optimized for a range of performance, and then give the PES a discriminating function that allows it to choose the best version (based on what performance is available) at run time. This is the telescoping languages approach. Alternatively, the compilation system can defer the final optimization steps and do them “just-in-time” based on available performance.

Based on our experience with the ScaLAPACK and Cactus codes, both approaches seem viable. For ScaLAPACK, an overdecomposed, cyclic strip decomposition should be used if the message start-up times do not exceed the neighbor-wise transfer times (based on available bandwidth). For a given bandwidth, there is a problem size, then, under which the code should be executed only on a single machine. However, since network performance is highly variable on the Grid, it is often difficult to determine the “single” bandwidth that will be available from a given link. The best we can currently do is to provide a range of bandwidths within which the deliver level is likely to fall. Using the NWS forecasting system and its conditional performance variance readings, it is possible to estimate the range of bandwidths that a specified link is likely to

deliver. It is possible, then, to determine the worst case and likely case from the NWS, and for the PES to choose the correct mapping for each.

Alternatively, it may be desirable to use a high-level intermediate representation for the code and then compile it to machine-executable binary form only when the performance that will be available is known. Recent research in the area of dynamic Java optimization indicates that this approach is feasible, but for GrADS it would need to be developed for C and Fortran to be truly effective. In this case as well, the NWS can provide a forecast of available resource performance at the time the program is compiled for the time frame in which it will execute. It is then up to the PES to perform the expected mapping and launch the program.

Preliminary work at Rice on the dynamic optimizer has focused on mechanisms for compiling and optimizing a configurable object program for a variety of platforms. This work has two components:

1. Binary-to-binary translation (or asm-to-asm translation). We have developed new techniques to reconstruct the CFG from assembly or machine code. We will use this in a preliminary tool to insert instrumentation in GrADS executables.

2. We have done some preliminary work on fluff removal and hot-path optimization (dynamically redoing code layout to improve performance). Our initial attempt will produce a system that does these transformations off-line, using estimated execution times rather than profile data. We intend to insert this technology into the preliminary binary-to-binary translator (mentioned in point 1).

We have examined several systems to assess whether or not they would serve well as the base for an eventual GrADS compilation system. These systems include GCC, LCC, the SGI Pro64 compiler, and the SUIF/machSUIF systems. None of these systems is completely satisfactory and we are investigating building upon in-house tools and creating a new ILOC-based code generator.

In the next year, we intend to field a binary-to-binary translator that can automatically discover the “right” places for instrumentation and insert the instrumentation directly into the code. This would eliminate the manual insertion of probes done today and be a first step toward building the dynamic optimizer.

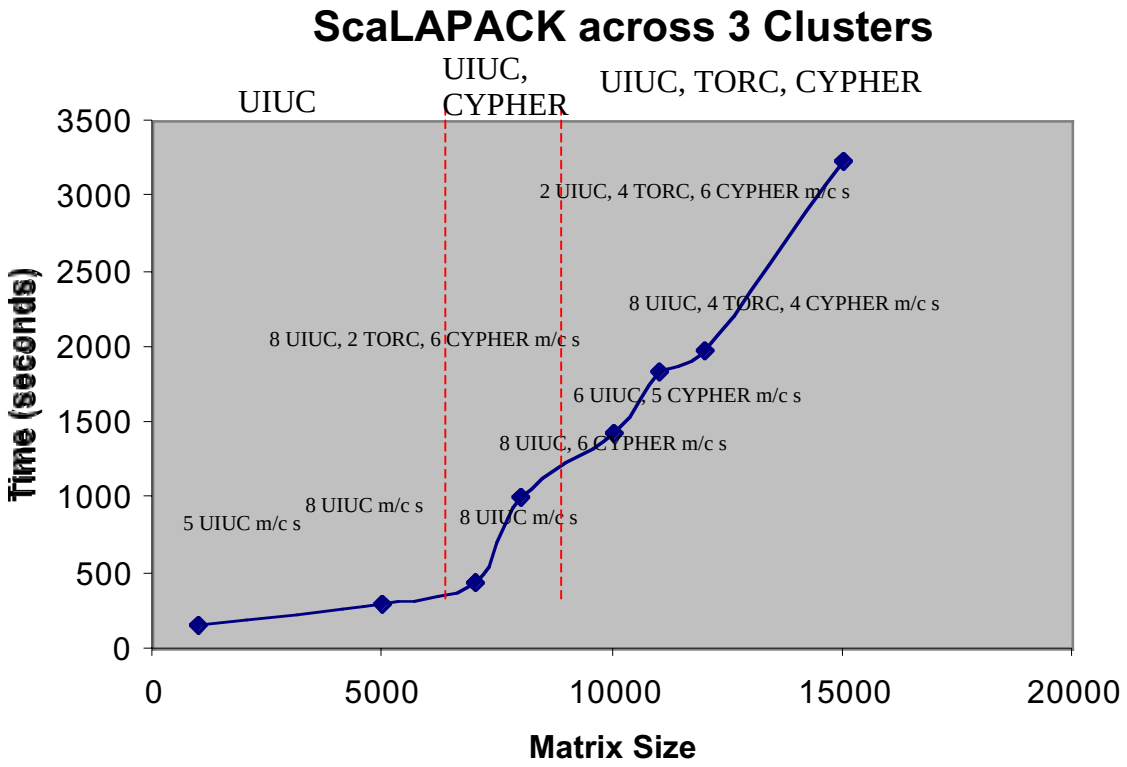
3. ScaLAPACK Experiment

The ScaLAPACK experiment demonstrated the feasibility of solving large-scale numerical problems over the Grid and analyzed the added cost of performing the calculation over machines spanning geographically distributed sites. We solved a simple linear system of equations over the Grid using Gaussian elimination with partial pivoting via the ScaLAPACK routine, PDGESV. We illustrated how users, without much knowledge of numerical libraries, could seamlessly use numerical library routines over the Grid. We also determined the steps that are necessary for a library writer to integrate his library into the Grid System. While ease of use is an important aspect of the Grid, performance improvement and the ability to perform tasks that are too large

to be performed on a single tightly-coupled cluster are important motivations behind the use of the Grid. Our experiments showed that effective resources can be selected to solve the ScaLAPACK problem and, for our experiments, scalability (as the problem size and the number of processors increases) of the software is maintained.

3.1 Preliminary Experiments

As part of the GrADS project, we are building a grid-enabled framework for numerical libraries. The experiment used ScaLAPACK, Globus, NWS, and Autopilot to demonstrate that large-scale numerical problems can be solved over the wide-area Grid, and to analyze the added cost of performing the calculation over a wide-area network of machines. Preliminary experiments have been conducted. The results of some of these experiments are shown in the following graph.



The experiment shown in the graph used clusters at the University of Illinois at Urbana-Champaign (UIUC) and on SInRG at the University of Tennessee, Knoxville (UTK) campus. In the experiment, GrADS chose only UIUC machines for matrix sizes up to 8000. The UIUC machines are connected by high-speed Myrinet while the UTK clusters uses Ethernet. Hence, for smaller problem sizes, it is advantageous to use the UIUC cluster. For matrix size 8000, when 8 machines are used, the amount of memory needed by a single machine is on the order of 512 MB. Since none of the UIUC machines have this much memory, GrADS utilized both the UIUC and CYPHER machines, which are NSF supported, for matrix sizes greater than 8000. Within the GrADS infrastructure, preference was given to CYPHER machines over TORCs because of the superior network speed of CYPHERs. Hence we find a steep increase in execution time from matrix size 7000 to 8000. We also find that the number of machines chosen for matrix size 10000 is less than the number of machines chosen for matrix size 8000. This is because certain

UIUC machines have very small memory size and these machines were found to be unfit by the GrADS system for matrix size 10000. Also, since the experiment was conducted on non-dedicated machines, GrADS rejected some machines from the experiments when the system and network loads corresponding to those machines significantly increased. For matrix sizes greater than 10000, machines from all the 3 clusters were chosen. We find the transition from 10000 to 11000 is not as steep as the transition from 7000 to 8000. This is because the transition from 7000 to 8000 involved Internet connections between UIUC and UTK machines and the transition from 10000 to 11000 involved UTK campus interconnections between the CYPHER and TORC clusters.

Thus we find that GrADS makes intelligent decisions based the application and system parameters.

3.2 Research & Education Activities

The primary goals of our effort in numerical libraries are to develop a new generation of algorithms and software libraries needed for the effective and reliable use of dynamic, distributed and parallel environments, and to validate the resulting libraries and algorithms on important scientific applications. To consistently obtain high performance in the Grid environment will require advances in both algorithms and supporting software.

Some of the challenges in this arena have already been encountered. For example, to make effective use of current high-end machines, the software must manage both communication and the memory hierarchy. This problem has been attacked with a combination of compile-time and run-time techniques. On the Grid, the increased scale of computation, depth of memory hierarchies, range of latencies, and variability in the run-time environment will make these problems much harder.

To address these issues, we must rethink the way that we build libraries. The issues to consider include software architecture, programming languages and environments, compile versus run-time functionality, data structure support, and fundamental algorithm design. The challenges that we face include:

- The library software must support performance optimization and algorithm choice at run time.
- The architecture of software libraries must facilitate efficient interfaces to a number of languages, as well as effective support for the relevant data structures in those languages.
- New algorithmic techniques will be a prerequisite for latency tolerant applications.
- New scalable algorithms that expose massive numbers of concurrent threads will be needed to keep parallel resources busy and to hide memory latencies.
- Library mechanisms that will interact with the Grid to dynamically deploy resources in solving the posed users' problems will be required.

These considerations lead naturally to a number of important areas where research in algorithm design and library architecture is needed.

To enable the use of the Grid as a seamless computing environment, we are developing parameterizable algorithms and software annotated with performance contracts. These annotations will help a dynamic optimizer tune performance for a wide range of architectures. This tuning will, in many cases, be accomplished by having the dynamic optimizer and run-time system provide input parameters to the library routines that will enable them to make a resource-efficient algorithm selection. We are also developing new algorithms that use adaptive strategies by interacting with other GrADS components. For example, libraries will incorporate performance contracts for dynamic negotiation of resources, as well as runtime support for adaptive strategies to allow the compiler, scheduler, and runtime system to influence the course of execution.

Our goal is to adapt the existing distributed memory software libraries to fit into the Grid setting without too many changes to the basic numerical software. We want to free the user from having to allocate the processors, make decisions on which processors to use to solve the problem, determine how the data is to be decomposed to optimally solve the problem, allocate the resources, start the message passing system, monitor the progress, migrate or restart the application if a problem is encountered, collect the results from the processors, and return the processors to the pool of resources.

3.3 Publications and conferences

The numerical libraries research effort has produced the following publications during the reporting period:

- "Numerical Libraries And The Grid," Antoine Petitet, Susan Blackford, Jack Dongarra, Brett Ellis, Graham Fagg, Kenneth Roche, and Sathish Vadhiyar, (submitted to IEEE Conference SC2001, Denver CO).
- "Numerical Libraries And The Grid: The GrADS Experiments With ScaLAPACK," Antoine Petitet, Susan Blackford, Jack Dongarra, Brett Ellis, Graham Fagg, Kenneth Roche, and Sathish Vadhiyar, University of Tennessee Tech Report UT-CS-01-460, April 28, 2001, (to appear in *International Journal of High Performance Computing Applications)*.

4. Cactus Experiment

We have adopted the Cactus framework as a test case for the adaptive techniques being developed by the GrADS project. Our goals are to use Cactus to (a) explore and evaluate, via hand coding, the adaptive techniques that we may later wish to apply automatically, and (b) apply and evaluate automated techniques being developed within other GrADS subprojects, for such purposes as contract monitoring, resource selection, generation of performance models, and so forth. In the process, we also explore two elements that we believe will be significant for future Grid computing, namely *Grid-enabled computational frameworks* that incorporate the adaptive techniques required for operation in dynamic Grid environments and *Grid runtimes* that provide key services required by such frameworks, such as security, resource discovery, and resource co-allocation. Such computational frameworks and runtimes allow users to code applications at a high level of abstraction (e.g., as operations on multi-dimensional arrays), delegating to the framework and runtime difficult issues relating to distribution across Grid

resources, choice of algorithm, and so forth. Such frameworks and runtimes have of course been applied extensively within parallel computing; however, the Grid environment introduces new challenges that require new approaches.

In our work to date, we have developed, in collaboration with the Cactus project team, a prototype Grid-enabled framework and runtime. The framework is based on Cactus, a powerful modular toolkit for the construction of parallel solvers for partial differential equations. This framework has been extended with new modules for dynamic data distribution, latency-tolerant communication algorithms, and detection of application slowdown. The runtime exploits services provided by the Globus Toolkit for security, resource discovery, and resource access, and also provides new Resource Locator and Migrator services. We report here on the overall architecture of this Grid-enabled Cactus system and our experiences applying it to a specific challenge problem, namely automated migration of computationally demanding astrophysics computations to "better" resources when currently available resources become overloaded.

4.1 Cactus and Dynamic Grid Computing

We discuss scenarios that lead Cactus developers to be interested in Grid computing, and use these scenarios to arrive at requirements for a Grid-enabled framework and runtime.
Cactus is widely used to perform large-scale simulations in computational astrophysics, for example to study the physics of extreme events such as black hole or neutron star collisions including computing the gravitational wave emission from such events. The latter calculations are currently of great importance due to the major investments being made in gravitational wave detectors.

The complexity of the Einstein equations that must be solved in numerical relativity means that Cactus simulations can easily consume thousands of floating point operations (flops) per grid point per time step. When applied to the large three-dimensional grids needed for accurate simulations, the aggregate computing demands can reach trillions of flops per time step. Thus, Cactus simulations may require weeks of run time on large multiprocessors: this is not a class of simulations that runs effectively on individual workstations.

These tremendous computational requirements have led computational astrophysicists to be aggressive early adopters of parallel computing and major users of supercomputer centers. However, the resources provided by such centers are never sufficient for delivering adequate throughput for either the routine computations performed in the course of daily research and development, or the large-scale computations required for accurate production runs needed for physical results and insights.

In this context, Grid computing becomes attractive as a means of delivering a new computational resource for both routine computations (the aggregate workstation and small-cluster computing resources of a collaboration such as the multi-institutional Cactus team) and large-scale computations (coupling supercomputers at multiple sites). However, these resources are difficult to harness due to their heterogeneous and dynamic nature. The long duration of computational astrophysics simulations exacerbates this problem. Another complicating factor is that computational requirements can change significantly over time.

Table 1 illustrates a scenario that captures the various elements that we believe are required in a complete computing solution for Grid environments. The capabilities required to support this

scenario include information service mechanisms able to locate appropriate resources, determine their characteristics, and monitor their state over time; security and resource allocation mechanisms that allow resources to be brought into computations; configurable applications that can be set up to run effectively on various combinations of resources; mechanisms for monitoring application behavior, for example to detect slowdown; and migration mechanisms that allow the set of resources allocated to a computation to be changed over time. Our architecture incorporates all of these capabilities.

Table 1: Cactus execution scenario in a dynamic Grid environment

Time	Activity
09:00	A user poses a computational astrophysics black hole problem that will require around one hundred CPU hours on a fast workstation. The system locates 50 processors that are currently available at one of the sites participating in the collaboration and starts the simulation.
09:30	A 200-processor cluster becomes available. The system notifies the application, which chooses to migrate to the new system.
09:50	The 200-processor cluster becomes unavailable. The system locates 100 processors distributed across three sites connected via high-speed networks, configures Cactus to run on this heterogeneous system, and restarts the simulation.
10:00	The simulation enters a phase in which it wishes to spawn, every 10 iterations, an analysis task that looks for black hole event horizons in the simulation data. The system locates another 20 processors and brings these into the mix. These processors then working independently of the main simulation.
10:15	Some of the processors allocated at 09:50 become overloaded. The computation is migrated off these processors onto other machines.
10:25	The computation completes.

4.2 A Grid-Enabled Framework and Runtime

We are developing a software system that addresses the requirements established in the preceding section. As illustrated in Figure 2, this system is intended to support both application-level adaptation and adaptive resource selection; we view these two techniques as equally important means of achieving performance in a dynamic environment.

The “software system” that we are developing comprises, in fact, two distinct elements: a Grid-enabled framework (the Cactus framework described at http://www.cactuscode.org, enhanced with a set of thorns (modules) to address adaptation to heterogeneous resource sets) and an adaptive runtime (a set of services constructed on top of Globus and other services). The additional Cactus thorns include support for dynamic domain decompositions and adaptive communication schedules, and the detection of performance degradation and subsequent migration (the so-called “Tequila” thorn). The adaptive runtime includes a resource selector, a

computational degradation detection module (a.k.a. contract monitor), a general logic overseer, and a migrator, described in the remainder of this subsection.

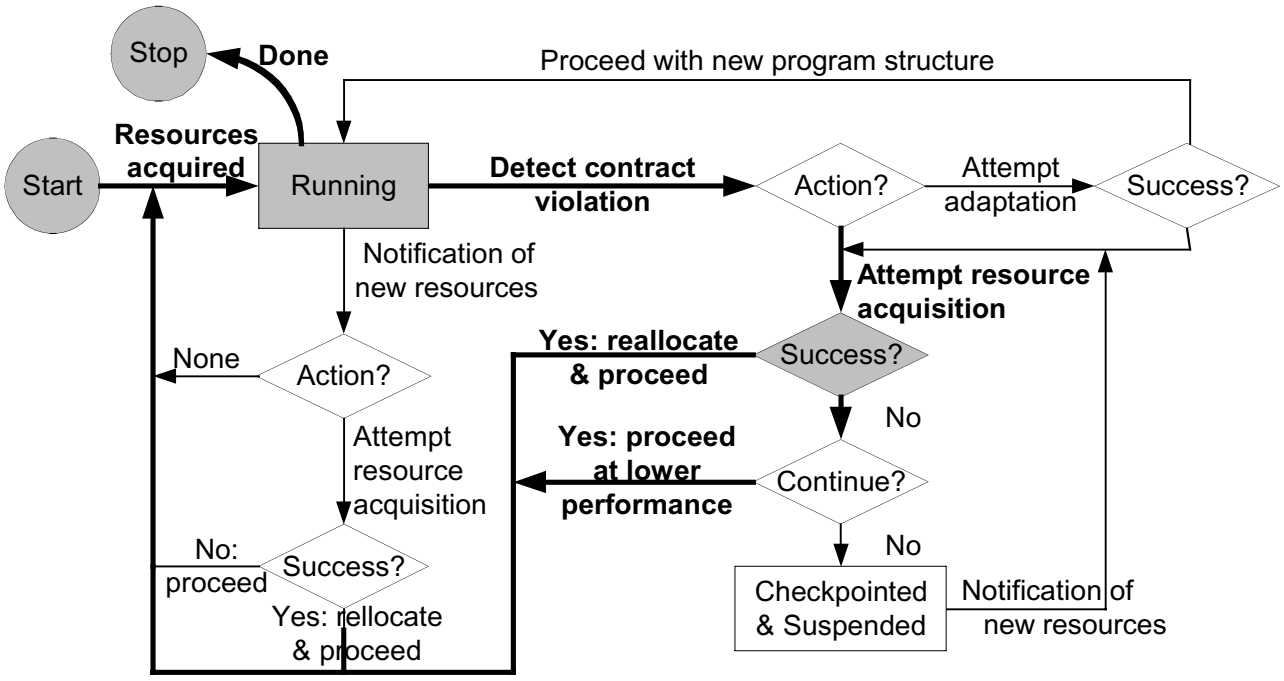


Figure 1: Flowchart showing the principal decision points that we envision for our Grid-enabled Cactus system. In boldface and shaded are the components discussed in this report.

We focus here on the elements in boldface and shaded in ***Figure 1***, that is, those concerned with resource reallocation as a result of a contract violation detected within the application. These components comprise, specifically, the Tequila thorn and the Resource Selector and Migrator services. In brief (see also Figure 2), the Tequila thorn first *determines that a contract violation has occurred* and that resource reallocation may hence be necessary. It then *communicates with the Resource Selector*, which *seeks to identify suitable alternative resources* from among those *discovered and characterized by the selector*. If such resources are located, the Tequila thorn communicates with the Migrator to *effect the movement of the computation from one resource to another*.

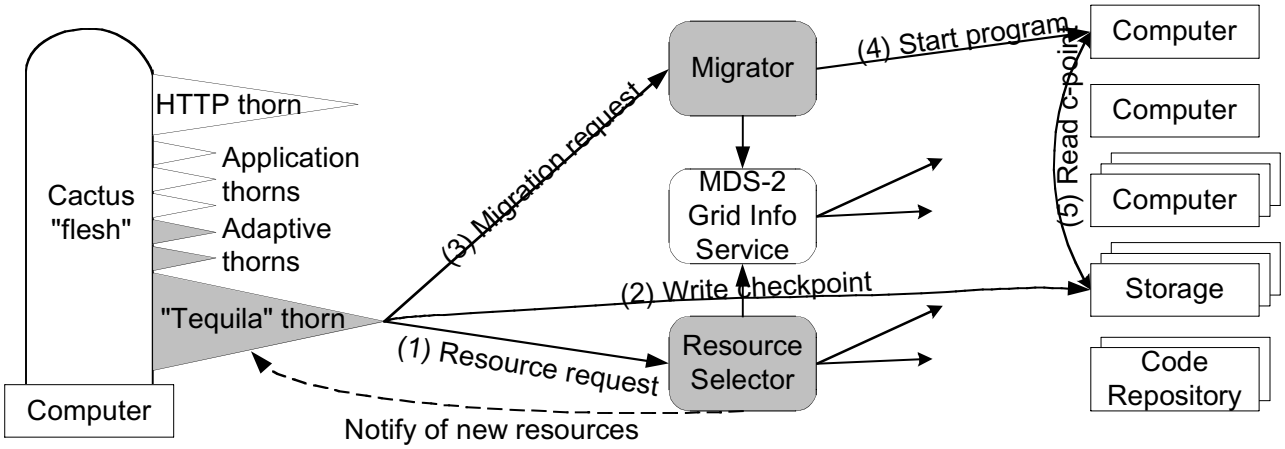


Figure 2: Overall architecture of the Grid-enabled Cactus framework, showing in particular the new elements developed in this and related work (shaded), including adaptive thorns, the "Tequila" thorn for managing migration, and the Resource Selector and Migrator services. Resources are discovered, monitored, and accessed using protocols and libraries provided via the Globus Toolkit. The steps involved in a migration operation are shown, numbered according to the order in which they are performed.

Our initial implementation uses a simple contract monitor that measures performance dynamically, in terms of iterations per second, and detects whether the initial performance attained degrades more than a certain percentage over the execution life of the application. The Tequila thorn monitors the performance of a Cactus application and, if performance degrades below a specified level, negotiates with the Resource Selector and Migrator services to move the computation to new resources. Note that the “reallocation” process that is undertaken here only involves migration to a new set of resources at this time.

The Resource Selector is responsible for resource discovery and selection based on application-supplied criteria such as the amount of memory required, the connectivity bandwidth required between processors, the current load of the processors, the OS, the software that must be installed, and/or the presence of a user account for that particular user. We define an application-Resource Selector communication protocol based on ClassAds.

The Migration Service is responsible for stopping the application on one set of resources and restarting it at the next sequential point in the calculations on a new set of resources.

When degradation is detected by the Contract Monitor and new resources are selected by the Resource Selector, the Cactus “Worm” Migration Unit will move the calculations to the new set of resources that are better suited (at the current time) to perform them. The sequence of the steps taken when a migration is initiated begins with creation of a checkpoint being made of the runtime state information of the application. This checkpoint data is moved to the new resources along with other data needed for restarting the application. Finally, the application is restarted on the new resources.

4.3 Experiences

We have demonstrated successful contract violation detection, resource selection, and migration. Our work now is focused on (a) studying the effectiveness and optimization of these procedures, and (b) investigating how techniques developed by other GrADS partners can be applied to automate some of these currently manually coded steps.

4.4 Education, Papers, and Presentations

Several graduate students at U. Chicago have been involved in the research, notably Matei Ripeanu, Anda Iamnitchi, and Chuang Liu. These students have participated in GrADS meetings and written papers on the research. Two of these papers will be presented at Europar'2001 and SC'2001.

Ian Foster presented results of this work in a keynote talk at the PPoPP'2001 conference on June 18, 2001.

5. MacroGrid

The primary goal of the GrADS MacroGrid activity is to provide an experimental environment to support the development of GrADS specific software and services and to provide a large-scale execution environment for GrADS application experiments. In designing the MacroGrid testbed, we strove to provide a real-world, large-scale, heterogeneous environment. The major part of the MacroGrid activities centered on the actual deployment and support of the experimental infrastructure.

Strictly speaking, design and operation of a testbed is not a research activity. However, the existence of this facility is critical to our ability to evaluate GrADS research results, and the complexity of setting up and maintaining such a testbed should not be underestimated. In spite of its operational nature, MacroGrid activities have resulted in the identification of new research problems in how to organize and present general purpose Grid resources in a manner that facilitates their use by application communities. This has led to the identification of a structure we refer to as a *virtual organization*, and the development of tools and techniques for defining and maintaining these domain specific views of general purpose Grid resources. These tools are described below.

By providing a structure that spans the range of alternatives over which adaptation may take place, we believe that the virtual organization concept and associated tools will play an important role in a GrADS system.

5.1 Infrastructure and Run-time environment

The past year has seen a doubling in the size of the MacroGrid from about 10 computing platforms to over twenty. The environment is highly heterogeneous and includes Sun, IBM, HP, SGI, and Linux platforms. In terms of processors, the total size of the testbed is somewhat modest, with a total of about 100 processors available. The vBNS and Internet 2 provide wide area network connectivity with a maximum communication bandwidth of 622 Mbits/sec.

Inclusion in the MacroGrid requires a resource to provide a standard set of software and services. This set includes basic grid services as well as specific libraries and tools that have been identified as required for the GrADS application experiments. During the reporting period, we have upgraded basic Grid services packages and expanded the supported basic tool set.

The initial testbed deployment was based on Globus Toolkit V1.1.2. We have since upgraded to Globus V1.1.3, which is the current Globus release. Important consequences of this upgrade are (1) the use of the new distributed information services model provided by MDS 2.0, and (2) support for MPICH-G2, which offers significantly better performance than MPICH-G. In addition to the core Globus software, Network Weather Service (NWS) beta-9 is used to provide higher level GrADS services with measured and predicted host load and available network bandwidth.

With the inclusion of the Cactus application experiments, the number of software packages required at each MacroGrid site has been expanded. These now include:

- MPICH-G for parallel application development,
- Autopilot**,**
- Version 5 of the Hierarchical Data Format (HDF) library for the Cactus application experiment, and
- BLACS, ATLAS, and ScalAPACK to support linear algebra experiments.

In the past year, we have developed tools that enable a resource participating on the GrADS testbed to report the version and location of installed software packages. Without the information, it would be extremely difficult run application experiments given the scale and heterogeneity of the MacroGrid testbed. An added benefit of tracking the installed software base is that it enables the testbed administrator to identify platforms that are not MacroGrid compliant.

5.2 Clearinghouse, Management and Administrative Infrastructure

Even in a relatively small, distributed project such as GrADS, communication between sites can be difficult. This is further complicated by the fact that MacroGrid involves a range of different types of personnel, including system administrators, research staff, and students. Thus a critical aspect of the MacroGrid is the coordination of resources across GrADS institutions and the dissemination of testbed-related information to administrators, resource owners, and individuals performing GrADS application experiments on the MacroGrid.

During the past year, we developed and operated a web-based clearinghouse geared towards both the scientific users of this MacroGrid as well as the local administrators of MacroGrid resources. Located at http://www.isi.edu/grads, the clearinghouse provides a range of information including how to get MacroGrid accounts, information about how to use the testbed to perform experiments, mailing lists for support, contact points at each member institution, and various links to documents and people.

As a result of the MacroGrid activities, we have been able to develop a close partnership between system administrators and users across the testbed, which has enabled us to quickly

resolve problems and make significant upgrades to the testbed as required to demonstrate the two applications so far. Collaborations created through this activity have demonstrated that issues encountered by one Grid development activity are likely to be encountered by others, and that pooling expertise on developing solutions for these issues yielded better results for all.

5.3 Technologies to support Specialized Grids

The previous discussion focused primarily on administrative and operational developments over the past year. In addition to these, we have initiated a research activity that is investigating how general purpose Grid resources can be organized to provide domain specific views. Rather than having to consider arbitrary Grid resources, these views present only the resources of "interest" to the end user. Within the context of GrADS, this means that the range of resources that must be considered for adaptation becomes well defined. The question we have started to address over the past year is how to specify, construct, and maintain such domain specific views, and how to present these views to the application or end user.

Our approach has been to leverage the new information model supported by MDS 2, the information service that was deployed as part of Globus 1.3. In MDS 2, there are two different types of information servers: information providers that supply information about one specific resource, and aggregate directories that index information provided by one or more information providers. Information providers are bound to aggregate directories by a soft-state registration protocol. An information provider can register with any number of aggregate directories.

In a standard Globus deployment, each resource typically registers with a single index, which is used to discover all of the resources that are available within an organization. To support MacroGrid, we have designed a *virtual organization* server that identifies all resources that are part of the MacroGrid (thus each resource belongs to its own organization as well as the MacroGrid testbed). In addition to identifying which resources are available, the virtual organization server provides access to status and software information. All information, including the elements of the testbed, is dynamically maintained, providing experiments with an accurate picture of the MacroGrid environment.

We set up and operated two virtual organization servers (for the sake of redundancy), one at USC/ISI and the other at UIUC, to provide status information to the GrADS community. This status information page continuously enumerates the resources available on the testbed, based on a service registration protocol. Applications and programs can access the same information using the LDAP protocol. For each hardware resource, information about its DNS name, operating systems, number of processors, CPU load, etc. are listed. Details about the installed software such as package name, version, patch level, and location of installation are also provided. Network bandwidth information has been partially integrated, and the model can easily be extended to incorporate other types of required information.

6. MicroGrid

During the reporting period, we have made significant progress in validating the MicroGrid emulation tools against MacroGrid experiments, adding new capabilities, and building

infrastructure which will ultimately enable other groups to more easily use the MicroGrid tools. These efforts are summarized below.

6.1 Validation/Demonstration of the Current MicroGrid System

In the past year, we have made significant progress in validating the MicroGrid tools as a high fidelity modeling tool for Grid applications and resource environments. This validation involved first enabling the MicroGrid simulation tools to run existing Globus applications unchanged. This was a significant effort to effectively virtualize the underlying Grid resources, but we successfully completed this, and then performed experiments with both the NAS Parallel Benchmarks and the Cactus applications.

Using the NAS applications programmed to an MPI interface, running on the MPICH-G implementation, we ran experiments on the MicroGrid tools. These NAS parallel benchmarks were run unchanged – identical versions that ran on other parallel systems supporting MPI. Results were compared to published runs for clusters, and matched within a few percent error on total application runtime. Specific cluster models include basic Beowulf, HPVM, and some wide area cluster configurations spanning networks between NCSA and SDSC.

The Cactus application framework supports a wide variety of computational applications on a wide variety of parallel systems. These Cactus applications are used by a large international collaboration of physicists and computational scientists. We successfully compiled and ran several Cactus applications on our MicroGrid simulation tools, and the experimental results show excellent match with the real performance. These applications were run on a local cluster configuration – the baseline – and then simulated on single or multiple workstations running the MicroGrid modeling tools. Again, the experiments demonstrated high fidelity modeling. For all of the Cactus applications, matching within a few percent in total runtime is achieved. We anticipate using larger Cactus applications for dynamic application adaptation experiments in the future, as the larger GrADS project is producing exciting application capabilities within the Cactus framework.

Finally, we also performed finer-grained validation, aligning internal progress points in an application execution both within an actual execution of the NAS parallel benchmarks and a MicroGrid emulated execution. The measurement of progress in both applications was achieved by inserting Autopilot sensors and collecting data from the actual execution and the emulated execution. It was seen that the monitored values for the MicroGrid coincided well with those for a real Grid. This validates the accuracy of the MicroGrid at a fine-grained level. In the future, we will be using both the ScaLAPACK and dynamic Cactus application programs being developed in other parts of the GrADS project to drive validation, and also to drive the design of adaptation techniques in the GrADS execution environment.

6.2 Adding New Capabilities and Infrastructure to the MicroGrid Tools

To enhance the capabilities of the MicroGrid tools, broadening their applicability and functionality, we added a number of new capabilities. These increase the efficiency of the

simulation and provide a foundation for more complete modeling of application and Grid resource behavior.

Dynamic Emulation Rates: Typically, the MicroGrid is used to emulate Grid applications at some slowdown below real-time execution. In the basic case, the MicroGrid execution speed parameters are statically determined. This typically underutilizes the emulation resources, for example, wasting network simulation capacity when there is little communication, etc. To increase emulation efficiency, we have added interfaces to the MicroGrid emulation modules that allow them to adaptively change the percentage of CPU time allocated to each process. Ultimately, this adaptation will be controlled automatically, responding in accord with the maximum rate at which the simulation can safely proceed. These changes have been validated using the EP benchmark of the NAS parallel suite.

Scalable Network Emulation: To meet the needs of scalable Grid modeling, we are exploring the design of scalable network emulators. We believe that network emulators built on distributed discrete event simulation engines can be scalable and support high fidelity emulation of an Internet-size network. The scalability can be achieved by using advanced synchronization algorithms and by exploiting the network hierarchy. Internet-size emulation can then be powered by scalable hardware systems. However, no one has demonstrated any of these key points. There are three key exploratory efforts to: (1) study and analyze the alternative synchronization algorithms and ways to exploit network topology, (2) build a prototype and test these alternatives, and (3) validate the emulator and use it for performance analysis of some real world applications.

So far, we have studied almost all currently available synchronization algorithms such as null-message, global barrier, and optimistic algorithms. We believe that a null-message-like algorithm is good for large-scale distributed simulation, but its success will depend on how much we can limit the overhead of null-messages. The basic conclusion is that we will try to implement this algorithm and adapt it to our emulator's requirements. The global barrier algorithm is also promising, considering the emergence of low-latency cluster interconnection. We will experiment with it. We will also try to mix the two algorithms to achieve better performance. The software of network emulators mainly consists of three parts: (1) a distributed simulation engine, (2) the network protocol model, such as TCP/IP, OSPF, BGP protocols, and (3) the emulation functionality and dynamic simulation rate parts. Currently, the simulation engine is almost complete; we are working on the network protocol modeling.

Background Traffic Modeling: On any actual shared network, in addition to the network traffic generated by the application that we are simulating, there may be a lot of traffic generated by other applications. We call this *background traffic*. The background traffic may share the same links and routers with the traffic that we are simulating, so it may affect our simulation results significantly. In order to do accurate simulation, we need to generate a background traffic model and simulate the effect of the background traffic. We are exploring a wide range of existing background traffic measurements and modeling techniques. These efforts take into account the special characteristics of Grid applications, and thus are quite different from more traditional models. We hope in the coming year to experiment with a number of these models within the MicroGrid structure.

6.3 Infrastructure Development

The MicroGrid requires a collection of machines to power both application execution and modeling. We have configured and deployed 10 Compaq/Alpha machines running Red Hat Linux. However, due to the lesser popularity of that configuration, it has become clear that we need to develop an x86 version. So, we have worked on a port of the system to x86 Red Hat Linux, and configured a cluster of over 18 machines. None of this hardware was purchased with funding from this grant.

We have improved the MicroGrid launching scripts to scale to large numbers of nodes, launching programs quickly and reclaiming resources as they shut down. In the new version, MicroGrid service can be started on multi-physical machines. Users also have more freedom to specify the resource used for simulation. As a result, expert users can control the simulation speed and resource powerfully, and the new users can do so conveniently.

In preparation for distribution of the software to the technical community, we have been building tests for the MicroGrid wrapper and the NSE simulation engine. Using these testers we hope to provide a fully functional, high quality, emulation environment. Additional efforts have gone into streamlined installation and configuration.