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 initial nine-month reporting period (10/1/99-6/30/00), GrADS research focused on the five inter-institutional efforts described in the following sections: *Program Execution System (PES)* led by Fran Berman, *Program Preparation System (PPS)* led by Ken Kennedy, *Macro Testbed* led by Carl Kesselman, *MicroGrid* led by Andrew Chien, and *Applications* led by Dennis Gannon. Project design and coordination was enabled through regular PI and subproject teleconferences and through three GrADS workshops (9/30/99-10/1/99, Houston; 11/17/99, Portland; and 2/7-8/00, San Diego). A fourth GrADS workshop is scheduled for 7/24-25/00 at Argonne.

1. Program Execution System (PES)

Research activities in the PES (Program Execution System) group have centered on four thrusts: Installation and maintenance of a consistent inter-site development environment, development of a detailed design of the PES component interaction, implementation of a prototype PES system for a Globus-enabled version of the LU routine from ScaLAPACK, and exploration of various contract specification rulebases and measurement probes for the ScaLAPACK LU driver application. We will discuss each of these activities in the paragraphs below.

Installation and maintenance of a consistent inter-site development environment has proved to be a substantive undertaking. In order to demonstrate and develop a GrADS prototype, all partner sites were tasked with installing consistent versions of ScaLAPACK, Globus, the Network Weather Service, MPICH-G, Autopilot and other

software. All packages needed to be tuned/adapted in order to provide this platform and consistent versions will need to be maintained at all GrADS sites.

The development of a detailed design for the PES component interaction has been a focused research activity of this group. A number of subgroup discussions and teleconferences have concentrated on how to do pairwise coupling of components of the system, with the goal of ensuring that ultimately all components work together. The system will be implemented as a set of low-level modules, which will fall into the domain of one or more general GrADS components (compiler, scheduler, PSE, etc.). We give a higher level description of the PES below with the caveat that some of these functions may end up being performed by alternative GrADS components:

- A user interacts with PSE (part of the Program Preparation System (PPS)).
- The PSE passes the user request to the compiler (part of the PPS).
- The compiler gathers library information, preprocesses the code for execution, and passes a performance model to assist with resource selection to a scheduler.
- Using relative performance measures, the scheduler discovers and identifies one (or more) potentially performance-efficient virtual machines, and provides this information to the compiler.
- The compiler uses the information about the possible resources (virtual machines) to instantiate an application performance model and passes this information back to the scheduler.
- An AppLeS-style adaptive scheduler determines an ordered list of one (or more) schedules, reserves resources for the first schedule(s), and passes the information back to the compiler so that an executable can be built.
- The compiler builds and instruments executables for the targeted machine. A contract specification may also be built which is passed onto the monitoring component of the run-time system.
- The compiler passes the scheduled executable and contract specification to the run-time system, which commits resources, deploys the program, and monitors program behavior during execution.

We are currently implementing the low-level modules required for the above scheme. Our goal is to have something running by the end of the summer. The extraordinary number of modules and their frequent and complex interactions require careful design, development, and integration. Our work with the ScaLAPACK LU routine has allowed us to discuss and test our theories and implementations on a widely used piece of software for which we have expertise within the GrADS community. There is an interesting set of tunable parameters within the LU routine that make it ideal for an initial test of the GrADS infrastructure, and the lessons learned will be directly applicable to other MPI applications in the Grid environment.

As part of the PES design and development efforts, we have been exploring various contract specification rulebases and measurement probes within the context of the ScaLAPACK LU driver program. To date, this effort has involved manual instrumentation of the LU code with Autopilot sensors and real-time monitoring of the

performance data by a contract validation component driven by a fuzzy logic rulebase. We can run the instrumented application with MPICH-G, introduce an additional load on some processors, and detect and report a drop in measured performance for the application being monitored. Identifying where to put measurement probes so that they capture critical values without excessively perturbing the application or over-reacting to short-term anomalies in the system has been one focus of the effort. As the PPS and PES mature, the code instrumentation and rulebase definition will be integrated into the GrADS infrastructure and performed automatically, guided by lessons learned in the current "manual" experiments.

The development and coordination of an adaptive system this large anchored by a large number of research software packages has provided a challenging educational and training experience to project personnel. Human, software, and hardware coordination has been a tremendous collaborative effort and learning experience for all involved.

2. Program Preparation System (PPS)

Research activities in the Program Preparation System (PPS) group have focused on three thrusts: installation and maintenance of a consistent inter-site development environment, experimentation with and analysis of the LU routine from ScaLAPACK (which will serve as our first "application" target), and analysis of alternative design strategies for automatically constructing performance models. These activities are described briefly in the following paragraphs.

As discussed in § 1, installation and maintenance of a consistent inter-site development environment has proved to be a substantive undertaking.

The 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.

Experimentation with the LU routine from ScaLAPACK has focused on understanding the static structure of this code as well as its dynamic behavior. The goal of this effort was to determine how this application can be modeled to provide information in a suitable form that will enable the program execution system to acquire resources and map the application's data and computation to the processors.

Previous efforts at mapping applications to computational grids have focused on algorithms with loosely-coupled synchronization that are suitable for embedding in linear virtual machine topologies. The ScaLAPACK LU routine, in contrast, has complex serialization patterns and requires some form of two-dimensional block-cyclic distribution of data and computation to achieve even load balance. Our initial work with LU focused on attempting to develop a detailed performance model represented as a directed acyclic graph (DAG), in which nodes represented computation, and edges represented communication and temporal serialization. Our intent was that the PES would consider the performance requirements for each node and edge in this graph, and map the nodes and edges to processors and network links in a way that would result in balanced parallel execution.

We have begun investigation of approaches for developing accurate machine-independent models of computation cost. Our current plan is to focus on measuring characteristics of an application's computation on a data tile for different tile sizes and characterizing the computation with an automatically derived equational model. By constructing a detailed model that focuses on application balance characteristics (namely, the ratios of memory operations, FLOPS, and integer operations) at the level of individual loops, we believe that performance on different architecture configurations can be predicted by comparing application balance with the balance characteristics of the target machine.

In an attempt to get an integrated GrADS prototype system up and running, the PPS project effort has shifted to coordinating 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 hand-crafted components that embody expert knowledge.

3. Macro Testbed

The goal of the macro-testbed effort is to develop the basic experimental infrastructure enabling GrADS research, and with which the GrADS research results can be evaluated. The macro-testbed and MicroGrid (described in § 4) provide complementary capabilities. Where the MicroGrid seeks to provide a carefully controlled simulation-based environment, the goal of the macro-testbed is to provide a real-world, large-scale, execution environment for Grid services and applications.

The macro-testbed consists of three elements: physical infrastructure (e.g. grid fabric), basic grid services, and administrative and management infrastructure. No new physical computational or networking elements were deployed as part of the macro-testbed.

Consequently, the initial phase of macro-testbed development focused on the identification of extant compute resources at each participating GrADS institution, and the deployment of an initial software suite of Grid services. Heterogeneity in the resource set was desirable so as to represent a realistic environment. The current testbed consists of 11 computers, and includes HP, Silicon Graphics, Sun, and Linux based machines, with processor counts ranging up to 34. The vBNS and Internet II provide network connectivity between these computers. The network provides a 622 Mbit/sec backbone bandwidth. Local campus connections vary from 10 to 155 Mbits/sec.

Initial basic grid services defined for the testbed are those provided by the Globus Grid Toolkit V1.1.2 and the Network Weather Service Beta-6 release. Globus provides security, resource control, and information services. The Network Weather Service serves as an instrumentation infrastructure, providing information on current and predicted network bandwidth and CPU availability. In addition, MPICH-G, a Grid enabled version of the MPI message passing library, was selected as the baseline programming environment, both to ease early development and to support legacy distributed codes. In identifying these services, our focus was on defining the minimum functionality required to conduct an initial set of adaptive execution experiments using ScaLAPACK as a test application.

An initial version of the testbed based on the above configuration has been deployed. The resulting testbed has been used to execute simple distributed MPI and ScaLAPACK experiments in the wide area across multiple institutions. Our initial testbed deployment provided little administrative and management structure. Based on the experiences of our initial experiments, we have a better understanding of testbed usage patterns and have used this information to define a more flexible administrative and management structure. An additional goal of the refined testbed definition is to exploit the improved information services architecture that is included as part of the latest Globus release (V1.1.3). An immediate benefit of this updated testbed plan is that we have been able to develop a tight integration between the Network Weather Service and the Globus testbed information services. Coordinating the Globus-Network Weather Service integration has also exposed the need for a Grid software release management infrastructure, which we will develop and institute.

4. MicroGrid

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

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

MicroGrid research activities have been focused on exploring how to simulate and model large-scale Grid structures—applications, services, and resource infrastructure. The ultimate simulation goals are networks of 10 to 100 million entities. The objective is to develop a set of simulation tools, called the *MicroGrid*, that enables systematic design and evaluation of middleware, applications, and network services for the Computational Grid. These tools will provide an environment for scientific and repeatable experimentation. It is our hope that the MicroGrid will catalyze the development of experimental methodologies to robustly extrapolate grid simulation results and rational grid design and management. To achieve these goals, at a minimum, Grid simulation tools must support realistic grid software environments, modeling of a wide variety of resources, and scalable performance.

We have designed, implemented, and validated an initial set of MicroGrid tools that allows researchers to run arbitrary Grid applications on virtual Grid resources, enabling the study of complex dynamic behavior. These tools enable the use of Globus applications without change by *virtualizing* the execution environment, providing the illusion of a virtual Grid. Thus, experimentation with a wide variety of existing Grid applications is feasible. Additionally, these tools manage heterogeneous models in the virtual grid, using a global virtual time model to preserve simulation accuracy. The MicroGrid also provides basic resource simulation models for computing, memory, and networking. These elements have been implemented and validated at three levels:

1. *Micro-benchmarks*: Individual resource model tests that show these models to be accurate.
2. *Parallel benchmarks*: Use of the NAS Parallel Benchmark (NPB) suite, which shows the combined accuracy of the MicroGrid for modeling system and application performance.
3. *An Application*: A complex application program, which further validates the utility and fidelity of the MicroGrid models.

For example, experiments with MicroGrids on the NPB suite ultimately matched within 1 to 12%, while maintaining high execution efficiency. These validations form an important first step in building a set of tools that can support large-scale experimentation with software for the Computational Grid. However, while we are pleased with our initial progress, significant challenges remain. To achieve the larger vision of Grid simulation, significant advances in scalability, precision of modeling, and network traffic modeling must be achieved.

5. Applications

The ScaLAPACK experiment described above is designed as an initial test of GrADS tools in an application context. However, we do not consider it to be a complete application. After considering several possible application targets, we have chosen to focus on the Cactus system (www.cactuscode.org). Cactus is an open source problem-solving environment designed for scientists and engineers. Its primary use is for large experiments in numerical relativity, but it can be adapted to other applications. Its modular structure easily enables parallel computation across different architectures and collaborative code development between different groups. Cactus originated in the academic research community, where it was developed and used over many years by a large international collaboration of physicists and computational scientists.

There are three reasons that GrADS is interested in Cactus. First, it is an application that has already made extensive use of Globus, and it has been ported to all the major scalable server platforms. Second, its modular architecture is very attractive for our goals for distributed computation. Third, the authors of the Cactus code are very interested in working with the GrADS project.

Cactus consists of a framework for composing application modules called "thorns." Each thorn provides a well-defined interface to its functionality, and thorns can be composed in a variety of ways to generate an executable MPI-based application. One of the interests of the Cactus group is to be able to solve very large problems distributed over a wide-area set of computing resources. However, another possible way in which Cactus can be decomposed is to allow different thorns to execute on different resources. The GrADS group will explore both of these execution models.

In the future, the GrADS group will use what we learn from Cactus and apply it to other application domains. One possible collaboration is with the NCSA Alliance Chemical Engineering team, which has a set of applications that they would like to execute in a Grid environment. Several other application groups have expressed similar interests.