GrADS Annual Report Year 3

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 (6/1/01-5/31/02), GrADS research focused on the six inter-institutional efforts described in the following sections: *Program Execution System (PES), Program Preparation System (PPS) & Libraries, ScaLAPACK, Cactus, MacroGrid & Infrastructure,* 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 (7/10/01, 5/15/02), and workshops (7/9-10/01, 3/15/02, and 5/14-15/02).

As part of the project management process, we developed a set of project milestones for major project activities reported in Section I, "Project Activities." We have also produced a report describing the activities of personnel supported by the GrADS project during the reporting period. The milestones and personnel report are included at the end of Section I.

1 Program Execution System (PES)

Our efforts over the last year have concentrated on two areas: (a) fundamental research on the components of the GrADS execution system and (b) construction of a testbed for experimenting with those components. The development activity has focused on producing a demonstration version of the execution system by October 2002

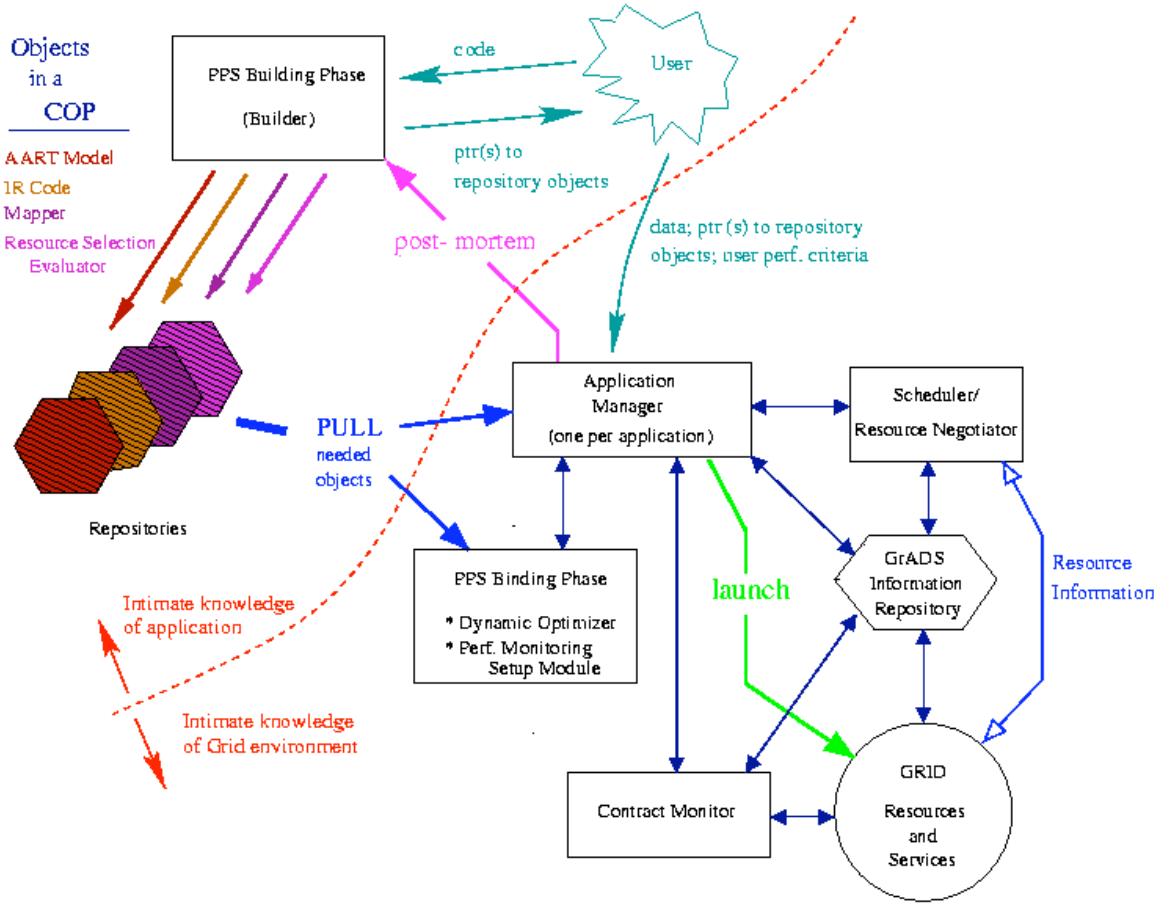
1.1 Scheduling

To achieve and maintain application performance on Computational Grids, it is vital that the GrADS system intelligently match application resource requirements and Grid resource conditions. We have developed a scheduling component that utilizes application-specific models and dynamic resource information from the GrADS information service to automatically identify an appropriate resource set and a mapping of data to those resources. The scheduler performs this evaluation process at each invocation of the application. An initial prototype scheduler was developed as part of Holly Dail's M.S. thesis work at UCSD [1]; extensive validation experiments on the MacroGrid demonstrated the effectiveness of this approach in real environments for two test applications.

The GrADS scheduler ensures that application requirements and resource availabilities are well matched at run-time. For long-running applications, for applications with changing resource requirements, and for testbeds with rapidly changing resource conditions, application performance may degrade during application execution. Otto Sievert's M.S. thesis work at UCSD is developing and prototyping a lightweight rescheduling strategy. Our approach is to *migrate* those application processes that do not meet performance expectations, while leaving most of the application in place. Compared to common rescheduling approaches, our strategy is more flexible than load balancing and lighter weight than checkpointing and restarting the entire application.

1.2 Application Manager

The figure below depicts the overall program execution and launch facility within GrADSoft



The GrADS Application Launch and Execution Process

A fundamental component of the prototype execution system is the GrADS Application Manager. Once a configurable object program, plus input data, is provided to the GrADS execution system, the system must initiate resource selection, launch the problem run, and monitor execution. In the GrADS execution framework, the *application manager* is responsible for these activities—either directly or through the invocation of other GrADS components or services. In this scenario, individual GrADS components need only know how to accomplish their task(s); the question of when and with what input or state is the responsibility of the application manager.

This year, a significant focus of the program execution system work was use of the experience we gained building Grid-enabled versions of ScaLAPACK and Cactus to guide our design and implementation of a general-purpose framework to support resource selection, launch, and execution control of Grid programs. Our work has focused on designing and developing

interfaces and data structures to support a GrADS Application Manager - an application-centric agent that controls the launch and execution of Grid programs.
and implementation of a general-purpose framework to support resource selection, launch, and execution control of Grid programs. Our work has focused on designing and developing interfaces and data structures to support a GrADS Application Manager - an application-centric agent that controls the launch and execution of Grid programs. The figure in this section shows the relationship between the application manager and other components of the GrADS system.

As part of the work on the application manager, we have refined the interfaces between the application manager and the service/resource negotiator for selecting the resources allocated to an execution of a Grid application. In particular, we extended the definition of the Application Abstract Resource Topology (AART) model, a description of an application's resource requirements and needs, to include functional attributes. These attributes are a flexible and expressive mechanism for representing both complex application requirements and their relationship to the attributes of the systems where the application will be mapped. These attributes are interpreted code fragments written in *Scheme* that consist of procedure definitions and expressions that invoke them. These functional attributes will be used to represent performance models for resource selection among other things.

Current design and implementation work in the application manager is focused on refining the application programming interfaces that form the basis for interactions with the scheduler/resource negotiator, dynamic optimizer/binder, contract monitor, and the launch mechanism.

1.3 Contract Monitoring

As a part of the PES effort, we are studying the feasibility, design, and implementation of a system of performance contracts. These contracts allow the level of performance expected of system modules to be quantified and then measured during execution of an application.

The current version of performance contracts includes software that uses a fuzzy rule set to quantify expected performance as a function of available resources. Such contracts include criteria, specified via measurable resources like MFLOPS or MB/s, needed to guide runtime environments in configuring object programs for available resources and in deciding if the observed performance matches expectations. When expectations are not met, a contract violation occurs. If there is such a contract violation, execution can be interrupted and reconfigured by the scheduling mechanisms being developed in GrADS, to achieve better application performance.

By using performance models obtained from a baseline execution of the underlying application, it was possible to detect contract violations on two Grid applications: a linear algebra solver from the ScaLAPACK package and a master-worker synthetic example. For those two applications, the contract monitoring mechanism correctly detected and quantified violations caused by external loads on the computational or network resources.

We believe our contract mechanism is sufficiently general to accommodate other types of performance models (e.g., those derived by compiler analysis of the application code or created in real time, based on the initial behavior observed during application execution). This work was validated using the ScaLAPACK code. Also, we plan to adapt this second form in the monitoring of a Grid version of the Cactus code and will be applying contracts to the GrADS project demonstration currently being defined. Finally, we plan to apply these techniques to the FASTA sequence code, described below.

Another major focus of our current work is to study the suitability of performance contracts to dynamically select the best code alternative for a certain task. Based on observed runtime conditions detected by performance sensors from our Autopilot toolkit, one can potentially choose, among the various implementations available in a runtime library, the particular version that is most effective under a certain scenario. Finally, we have been investigating low overhead statistical techniques for estimating the performance of large systems and execution signatures for describing application behavior.

1.4 GrADS Program Execution System Demonstration

We are currently working to complete a coordinated GrADSoft system that will include prototypes of all the major components of the GrADS architecture. The prototype GrADSoft system will be demonstrated with ScaLAPACK, Cactus, and a third test application—a genomic sequencing application based on FASTA—in October 2002. The latter application has been chosen to demonstrate the ability of the GrADS infrastructure to handle location-specific resource allocation and data dependent load balancing.

The prototype system will include simple versions of most of the components depicted in the figure above, including the application manager, the scheduler/resource negotiator, the contract monitoring system, and the binder. For the purposes of this demonstration, the GrADS information repository will consist of simple interfaces to MacroGrid services such as NWS.

[1] Holly Dail, "A Modular Framework for Adaptive Scheduling in Grid Application Development Environments," Masters Thesis, University of California at San Diego, March, 2002, Available as UCSD Tech. Report CS2002-0698

2 Program Preparation System (PPS) & Libraries

The goal of the GrADS program preparation system is to assist in the construction of configurable object programs that include interfaces used to coordinate the launch and execution of Grid applications. Research activities in the program preparation system this year focused in five areas: developing software technology for semi-automatically constructing scalable performance models for parallel applications, design and implementation of components of the GrADS application manager, investigation of compiler technology for a dynamic optimizer, a component framework for Grid services, and developing library technologies for adaptive programs. We discuss progress in each of these efforts in turn below.

2.1 Performance Modeling of Parallel Applications

For the program execution system to be able to evaluate alternative sets of resources for executing a Grid application, a configurable object program 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 determines whether or not a node is suitable for inclusion in a requested virtual machine topology. In last year's work, we found that manual construction of accurate models was quite difficult and developing techniques to automate or semi-automate this process was identified as an important goal. The problem of composing models from different program components was viewed as a particularly challenging aspect of this work. Accordingly, a major thrust of the program preparation system work during the current project year was to devise techniques and prototype tool support for semi-automatic construction of scalable performance models for parallel applications. Characterizing and modeling the performance of parallel applications has been a long-standing goal of computer science research.

We began our work this year by attempting to construct a scalable performance model of an application's node performance by composing models for the cost of each individual loop nest and procedure in the program. This addresses the problem of model composition by modeling and composing only low-level models. As a first attempt to construct a scalable model for a particular architecture, we tried to build a model for an individual loop nest by fitting piece-wise polynomials to data collected from hardware performance counters for multiple executions of the application using different problem sizes. We found that without a detailed model of a node program's sensitivity to memory latency, it was impossible to derive a unique model because of linear relationships between misses at different levels in the memory hierarchy. To overcome this problem, we began investigating strategies for constructing detailed architecture-neutral *application signatures* that characterize an application's performance in terms of its instruction mix, its sensitivity to memory hierarchy characteristics, and its communication characteristics. These signatures are the cornerstone for our current approach, which we call *black-box performance modeling*.

A principal focus of our work this year has been on automating the collection of data for constructing application signatures. We have been engaged in the design and construction of language-independent tools that (1) analyze executable binaries for parallel applications to capture static information about an application's code and (2) instrument them to capture information about the application's dynamic behavior. We have constructed a prototype binary analysis and instrumentation tool based on the *Executable Editing Library*.

At analysis time, our tool collects data about the instruction mix of each basic block in a program. Ongoing static analysis work is focused on refining an architecture-independent scheduling module that provides estimates about the execution time for a code sequence on any given architecture. We have built a prototype estimator module that computes performance estimates for a MIPS processor based on Sparc application binaries. The estimator module takes instructions from an application binary compiled for the Sparc architecture and constructs a schedule by mapping these instructions to an architecture-neutral instruction format and using information about a particular target architecture's resources,

constraints, operation latencies, and primary cache latency to construct a schedule for the instruction stream. A critical aspect of this schedule is that it will reflect the sensitivity of the schedule to memory hierarchy latency, namely, how much memory hierarchy latency (for each individual memory access) can be hidden by the instruction schedule before it will lengthen the execution time.

After collecting static information about an application, our tool adds instrumentation to the application binary to capture information about the dynamic characteristics of the application during sample executions. Currently, our tool augments the binary to collect information about an application's communication and computation. To collect communication information, our tool enables one to specify a signature for any communication routines about which information should be captured. The tool adds instrumentation that causes an execution of the program to collect a trace, similar to that recorded by many trace-based visualization tools, including information about communication partners and communication volume for each communication event. Work is underway to augment the dynamic data collection to capture an architecture-neutral description of memory hierarchy behavior by measuring the average "reuse distance" – the number of unique memory locations accessed between two accesses to a single location – for each load and store instruction in an application.

The fundamental challenge at this point in this work is to assemble the detailed information we are collecting into parameterized models.

2.2 Automatic Construction of Mappers

As part of program preparation system research, we also have been investigating strategies for automatically constructing a mapper, which determines how to assign application processes to available processor nodes. For this effort, we are focusing on construction of mappers from program task graphs, which can be constructed from MPI programs or from programs written in high-level languages. The strategies for mapper generation are based on clustering algorithms of various kinds, including fusion methods.

We are currently building a prototype mapper which leverages infrastructure from the NSF POEMS project to automatically generate a "task graph" from HPF application source. The mapper will use the task graph as the basis for mapping application processes to Grid nodes. If this is successful, we should be able to demonstrate HPF programs running on the Grid via the GrADSoft infrastructure sometime this year.

2.3 Binder/Dynamic Optimizer

The work has focused on understanding the structure of binary-form executables produced from optimized code, and on optimizations that make sense in the context of a dynamic (run-time) optimizer for GrADS. The major research thrusts in this work have been on x86-to-x86 optimization and translation, low-level performance modeling, and on reconstructing memory access patterns from the binary form.

We have worked extensively on reconstructing program information from binary executables including design and implementation of a new algorithm for building control flow graphs

from application binaries for complex microarchitectures. In our work, we developed a new strategy for computing flow graph dominator information. This is the fastest known technique (measured times, not asymptotic complexity) for computing dominators and has led to fast ways of building SSA-form from low-level code such as x86 executables. Both of these efforts directly support the GrADS dynamic optimizer and are necessary precursors to any restructuring of application binaries.

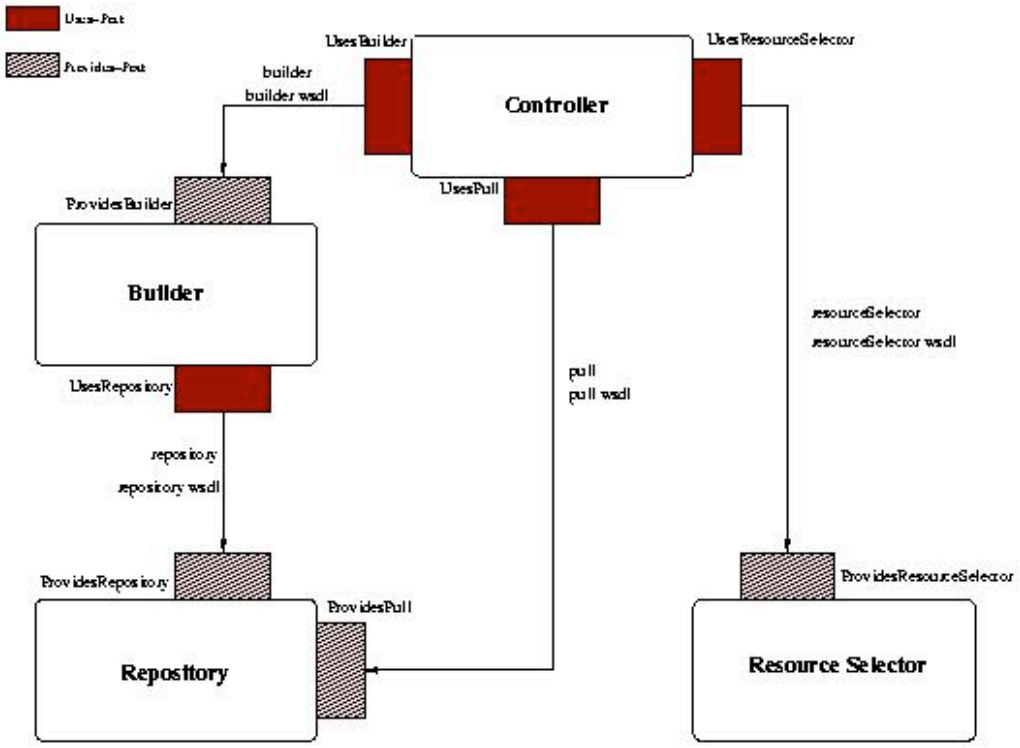
On top of this analysis infrastructure, we have been developing tools for x86 to x86 binary translation. This work has two aims. First, it will serve as the basis for the tool that inserts probes and monitoring code into application binaries to track their execution progress and performance. Second, it will also serve as the basis for optimization of binaries including low-level value numbering and re-scheduling. The current activity includes an attempt to vectorize Linux x86 application binaries to take advantage of the Pentium 4 SIMD unit.

As part of our work on the dynamic optimizer, we spent time studying the detailed performance characteristics of scientific and media programs. We expected this work to produce methods that would be applicable in the GrADS dynamic optimizer. We developed new analysis and transformations techniques for memory disambiguation. As the research turned out, these techniques are probably better suited for use in a big-iron compiler, like the GrADS compiler, rather than in the dynamic optimizer because of the expense of the analysis.

2.4 Component Framework for Grid Services

At Indiana University, research and development this year has focused on implementing the GrADS framework as a distributed system of services deployed on the MacroGrid. Within the GrADS execution model, applications are either decomposed into or synthesized from a set of components, which execute on remote resources and communicate with each other across the Grid network. This communicating network of components can be coordinated by a central agent or by a distributed algorithm. In either case, when the Grid environment or the application undergoes a change that requires a redistribution of the component execution, the control mechanism must contact the resource broker and builder services.

The approach we have been following is to use a version of the Common Component Architecture that has been adapted to a Grid Web Services architecture. This work began with the non-distributed version of the GrADS infrastructure developed at UCSD. Based on this code base, we have "componentized" the GrADS prototype Builder, Repository, Controller, and Resource Selector. These individual pieces have been wrapped as CCA components, which expose and use web service interfaces and are connected as shown below.



The result of this experiment has been described in two draft reports. The next phase of this work involves solving three problems.

1. How can we capture the state of running components so that they can be easily moved from one execution resource to another?
2. What is the appropriate means for executing components to notify the controller of significant changes in state?
3. How can we automate the construction of the individual application components from high-level specification?

Work on the first two problems is underway at Indiana University. Work on the third problem is the most challenging for GrADS, and it is being done in collaboration with Rice, University of Houston, and UTK.

2.5 Libraries

Libraries research at the University of Houston has focused on UHFFT. In the GrADS context, this work has focused on research in the architecture of high-performance scientific library software consistent with the GrADS software architecture of separating program

development into a program preparation and program execution phases. The UHFFT adapts automatically to the hardware it is running on by using a dynamic construction (execution phase) of composable blocks of code generated and optimized for the underlying architecture during the installation of the library (program preparation).

The UHFFT performance has been compared with other FFT libraries as stipulated in the Statement of Work and is very competitive with the best-known public domain libraries and even some vendor libraries. Compared to libraries with a similar approach, such as the FFTW, the UHFFT offers some additional flexibility in optimization with respect to the execution environment at installation time (program preparation). The UHFFT is entirely based on standard C, assuring ease of deployment and portability.

Performance models are still under development, but at this time do predict the onset of cache thrashing with good accuracy under a broad range of data layout/access patterns. Work is still ongoing to accurately estimate the magnitude of cache thrashing across a broad range of data layouts, access patterns, and architectures. The dynamic construction of executable code is currently based on a database of performance data generated at installation time and updated as executions are performed.

A collection of tools and an infrastructure for adaptive code generation and optimization have been developed enabling efficient automatic generation and optimization of library modules using a special-purpose compiler. The code generator, written in C, makes the library flexible and extensible and the entire collection of codelets can be (re)generated in a matter of seconds. The software generation and optimization infrastructure is quite general and it can be easily extended to other well-structured, composable functions.

The work on a new set of library interfaces assuring portability has began recently and is carried out in collaboration with Intel. A prototype implementation is planned for Q3 2002.

3 ScaLAPACK

3.1 ScaLAPACK LU Solver Enhancements

The ScaLAPACK LU solver was previously implemented as a demonstration of the GrADS framework, showing how a numerical library routine could be seamlessly deployed and monitored on a computational Grid with minimal alteration to the routine itself. Good performance and scalability was achieved on very large problems.

Better Performance Model: In the current work, the performance model for the ScaLAPACK LU solver (PDSCAEX) has been improved to take better account of the communication costs. This work was initiated because there were some unexpected variations in the per-iteration predictions that made it difficult to detect possible contract violations. The new performance model has led to improvement in the ability to detect contract violations; however this still needs some work.

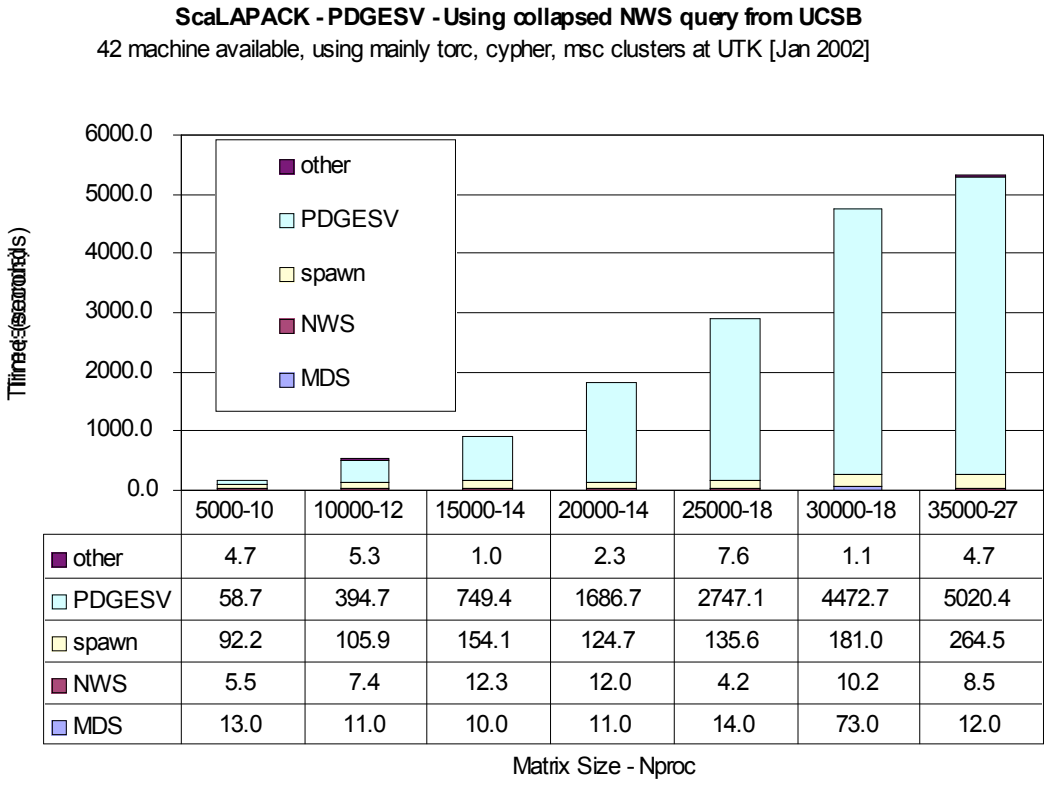
Improvements in the Information Lookup Overhead: In the initial version of the GrADS ScaLAPACK demonstration, the overhead required for the grid information services such as MDS and NWS was substantial when compared to the time required for execution. Algorithmic improvements were used to reduce the number of NWS queries and to significantly improve the cost for looking up grid information.

3.2 More Performance Models

As part of integrating more numerical libraries into the GrADS framework, we have been building more performance models for major SCALAPACK routines, namely QR factorization and eigenvalue problems.

3.2.1 QR factorization:

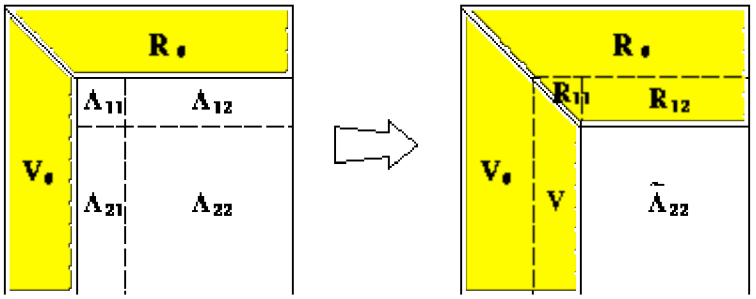
The performance model for QR estimates the times taken for broadcasting the Householder transformations and the update of trailing matrices. The following diagram illustrates the process of QR factorization.



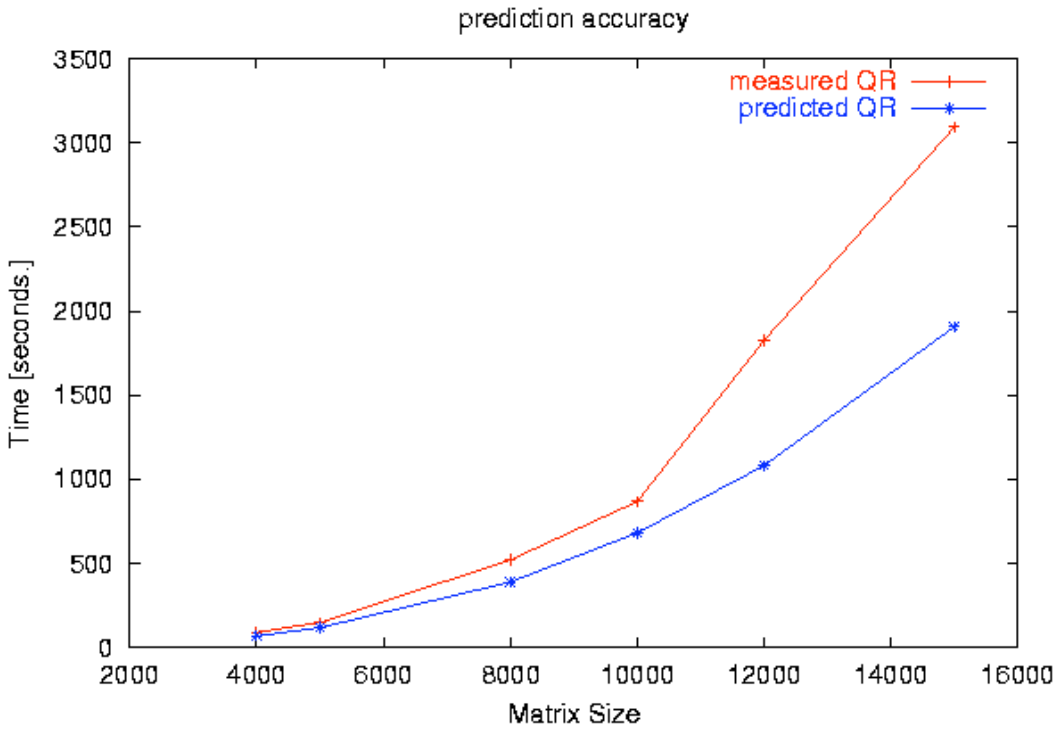
	5000-10	10000-12	15000-14	20000-14	25000-18	30000-18	35000-27
other	4.7	5.3	1.0	2.3	7.6	1.1	4.7
PDGESV	58.7	394.7	749.4	1686.7	2747.1	4472.7	5020.4
spawn	92.2	105.9	154.1	124.7	135.6	181.0	264.5
NWS	5.5	7.4	12.3	12.0	4.2	10.2	8.5
MDS	13.0	11.0	10.0	11.0	14.0	73.0	12.0

Figure 3.1: Breakdown of execution time for LU solver PDGESV; the GrADS framework overhead has been greatly reduced

Figure 3.2. QR factorization.

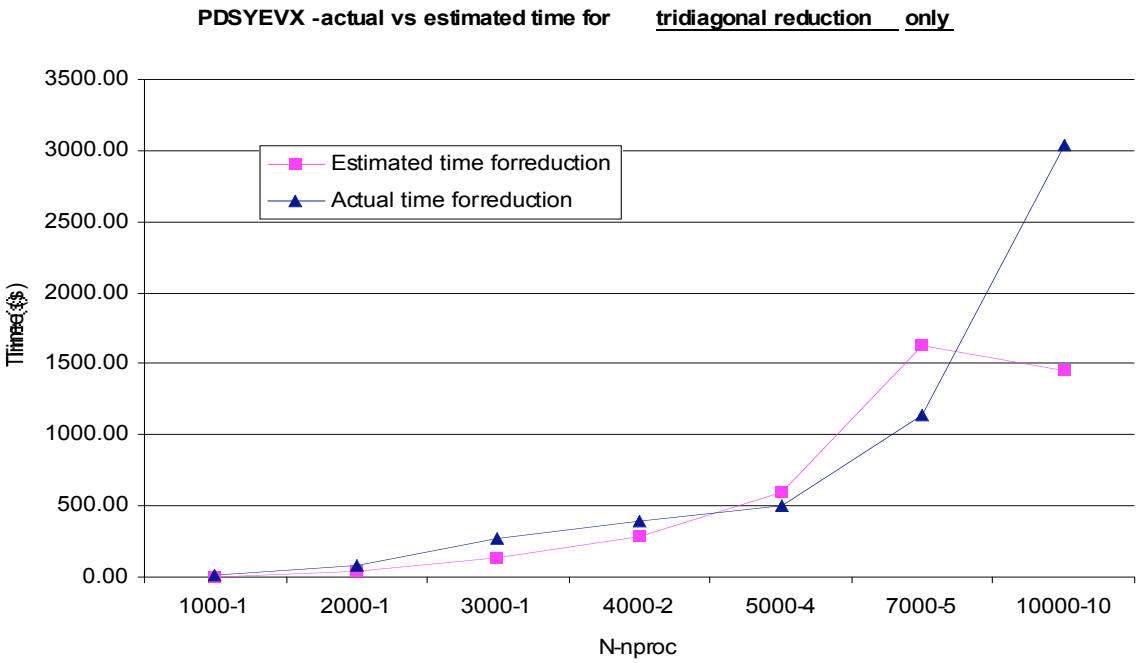


The following graph illustrates the accuracy of the QR performance model. The results were obtained on a Linux Pentium Cluster in UT.



3.2.2 Symmetric Eigensolver Routine

We have been working to confirm that the results of the GrADS ScaLAPACK LU solver experiment could be repeated with different numerical routines, and to estimate the effort required in generalizing and perhaps automating, the process of adapting a numerical library



routine to the GrADS framework. To that end, the ScaLAPACK eigensolver routine PDSYEVX was adapted to the GrADS framework. The routine consists of several distinct phases. The first phase, tridiagonal reduction, was analyzed and a Performance Model/RSE was hand crafted for this phase; later phases can have an unpredictable number of iterations, and so were not included in the Performance Model.

The same GrADS framework that was used in the earlier LU solver experiment was used for the eigenvalue solver. The new performance model was used to generate a schedule for the application on a subset of the machines in the experimental testbed. The eigenvalue application was run and various measurements were made. We find that the overhead due to the GrADS framework is relatively small compared to the execution times of the applications. Also, the estimated execution time generated by the Performance Model is very close to the measured execution time, implying that the Performance Model can be used for scheduling and contract monitoring.

3.3 Checkpointing and Rescheduling of ScaLAPACK applications

UT is also investigating the various aspects of rescheduling already running ScaLAPACK applications. These aspects include designing the architecture for checkpointing ScaLAPACK applications, developing heuristics to determine the performance benefits of rescheduling the applications, evaluating the cost of checkpointing the ScaLAPACK applications, determining the new set of resources onto which the applications can be migrated etc. Mechanisms have been implemented in the ScaLAPACK code that will enable the ScaLAPACK application to be stopped and restarted on possibly different number of processors.

A component, called Runtime Support System (RSS) is started even before the application is started. This RSS is specific to the ScaLAPACK application and registers to a Database Manager (DM) with the application identifier. When the application is started, it gets the location of the runtime system from the DM and contacts the RSS and performs some initialization. When the scheduler component wants to stop the application, it sends a STOP signal to the RSS. The application, between the iterations, contacts the RSS to check if it has received the STOP signal. When the STOP signal is received, each process of the application opens up storage of specific size on the local disk and checkpoints the data to the disk. The handles to the stored data are stored in the RSS. When the application is restarted on a different set of processors, the application contacts the RSS through the DM, obtains the handles to the data, reads in the checkpointed data, continues from its previous state, and proceeds to completion. When the number of processors in the restarted application is different from the number of processors in the original configuration, the RSS redistributes data from the previous storage to the new storage.

3.3.1 Experiments

A service called *Expander* is built in the GrADS system. Another component called Database manager maintains the states of the different GrADS applications. The Expander queries the database manager at regular intervals for completed applications. When an application completes, the expander determines if performance benefits can be obtained for an already executing application by expanding the application to utilize the resources freed by the completed application. If the expander detects such an executing application, it stops the application and continues the application on the new set of resources.

For the easy demonstration of the experiments, 2 clusters from the GrADS testbed were used. The clusters are called *mscs* and *torcs,* each consisting of 8 dual-node processors. The clusters are connected to each other by 100Mb Ethernet links. Table 1 gives the specifications of the machines.

Table1. Specification of the Experimental Testbed

Machine Name	**Processor Type**	**Speed (MHz)**	**3.4 Memory (Mbytes)**	**Network**
Torc	Pentium III	550	512	100 Mb switched Ethernet
MSC	Pentium III	933	512	100 Mb switched Ethernet

For the experiments in this section, ScaLAPACK QR factorization code was used. An application, app_1, was introduced into the system such that it consumed most of the memory of 8 msc machines. During the execution of app_1, an app_2, which intended to use 11 machines, 3 torcs and 8 mscs, was introduced into the system. Since the 8 msc machines were occupied by app_1, app_2 was able to utilize only the 3 torc machines. When app_1 completed, the 8 msc machines were freed and app_2 was able to utilize the extra resources to reduce its remaining execution time. The Expander evaluated the performance benefits of allowing app_2 to utilize the extra 8 processors.

ScaLAPACK problems of sizes 20000 and 21000, depending on the available memory on mscs when the experiments were run, were used for app_1. A ScaLAPACK problem of size 11000 was used for app_2.

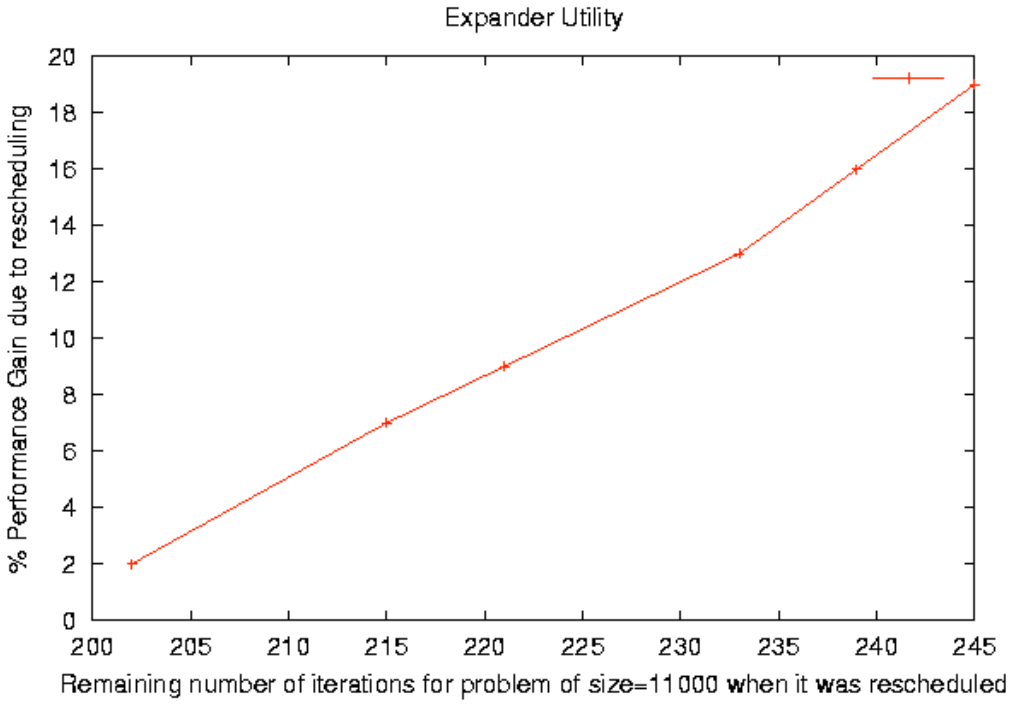
We define

1. Total execution time of app_2 on 3 torcs without rescheduling, $exec_{without\_re}$.
2. Total execution time of app_2 with rescheduling, $exec_{with\_re}$.
3. Percentage rescheduling gain for app_2, *percentage_gain*.

$$percentage\_gain = (exec_{without\_re} - exec_{with\_re}) / exec_{without\_re}$$

app_2 was introduced at various points of time after the starting of app_1. Hence additional resources will be available for app_2 at various points of time into its execution. The total number of iterations needed by the ScaLAPACK problem of size 11000 was 275. Figure 3.3 illustrates the utility of rescheduling as a function of the remaining number of iterations left for app_2 when app_2 was rescheduled. We observe that the percentage rescheduling gain for app_2 increases when the remaining execution time left for app_2 at the time of rescheduling increases. The rescheduling gain depends on a number of parameters, like the time taken for redistribution of data and the number of additional resources available, etc. These parameters depend on the specific application for which rescheduling is done. Work is in progress to build interfaces for the application library writer by which the system can determine the rescheduling parameters for the applications.

Figure 3.3. Rescheduling of QR application

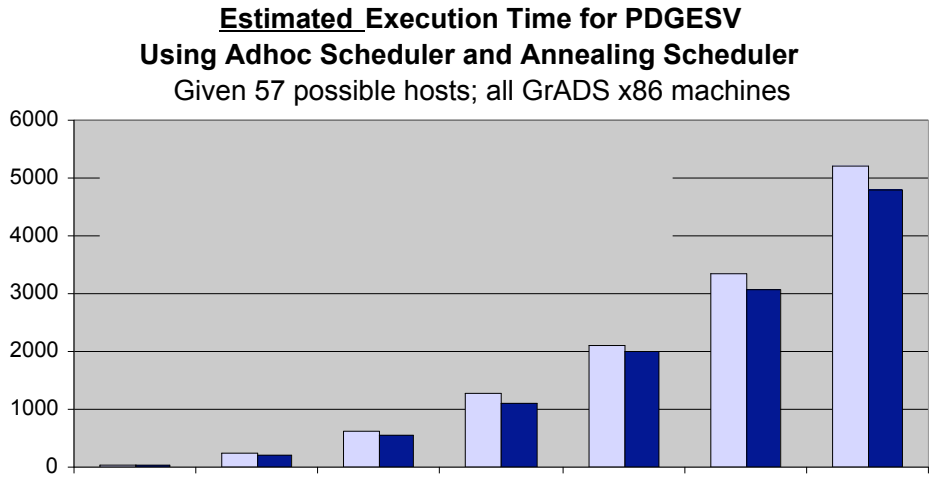


3.5 Simulated Annealing Scheduler

Scheduling applications on a Grid is a challenging problem that has to be addressed if high performance is to be achieved in such a geographically distributed, heterogeneous environment. In the current GrADS effort, we work with application level schedulers that use detailed performance models of the applications. The current schedulers provide good machine schedules, however, additional experimentation needed to be done with alternative, global scheduling mechanisms.

The Ad-Hoc greedy scheduler used in the original ScaLAPACK experiment starts once from each cluster, and seeds the schedule with the fastest machine in that cluster. The next machine selected is the one with the highest average bandwidth to the currently selected machines. The execution time for the application is estimated using a detailed Performance Model that takes into account the current status and connectivity of the machines. Machines are repeatedly added to the schedule until the performance starts to decrease. The best schedule from all the clusters is used to run the application. This scheduling mechanism generates good schedules; however it does not do a global search of the space of possible solutions. That is, if the optimal schedule is not ordered by bandwidth, this mechanism cannot find it.

Simulated Annealing is a Monte Carlo based minimization procedure. The scheduler tries various machine schedules, always accepting better schedules, and probabilistically accepting worse schedules depending on a "temperature" schedule. This search technique can escape local minima while still searching for a global minimum. The Simulated Annealing scheduler comes up with schedules that have better estimated execution times than current Ad-Hoc



greedy scheduler. However, the estimated execution times do not always match measured times, which implies that there is some application or environment knowledge built into the earlier Ad-Hoc greedy schedulers. Additionally, the Performance Model needs improvement to get a more accurate estimated execution time. If we are given an accurate Performance Model, the Simulated Annealing scheduler will produce better machine schedules than the Ad-Hoc scheduler.

3.6 ScaLAPACK on GrADSoft

GrADSoft defines a generic architecture for plugging Grid applications in the GrADS setting. Efforts are on to integrate the ScaLAPACK applications into GrADSoft so that the users of GrADS framework will be able to use a single interface to solve ScaLAPACK and other Grid related problems. This effort is also leading to the fine-tuning of the GrADSoft architecture to accept different performance models of different applications, different application launching mechanisms, etc.

3.7 LAPACK for Clusters

LAPACK for Clusters (LFC) is a University of Tennessee research "spin-off" of the GrADS effort. LFC is numerical software that employs information describing the state of the computational system at runtime to select an appropriate subset of resources that are best suited for solving the user's specific LAPACK numerical problem. The user is assumed to be working in a serial environment. The user is relieved of the complexities that arise when preparing problems for parallel execution (such as data mappings and movement) and is the beneficiary of system specific expert tuning of the kernels that drive the application routines. The target systems for LFC are Beowulf clusters. The design of LFC has depended strongly on the model developed by GrADS researchers coupled with the SANS (self-adapting numerical software) approach to software development. The hope is to couple a software interface as simple as LAPACK and deliver the computational power of ScaLAPACK.

3.8 Other GrADS Related Research Activities

NetSolve: Research is proceeding on integrating NetSolve with the GrADS framework.

Binding and Data Staging: Some work has been performed on staging large datasets to distributed processes using IBP (Internet Backplane Protocol) to move the data.

4 Cactus

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 automatic detection of and adaptation to 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 4.3, 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.

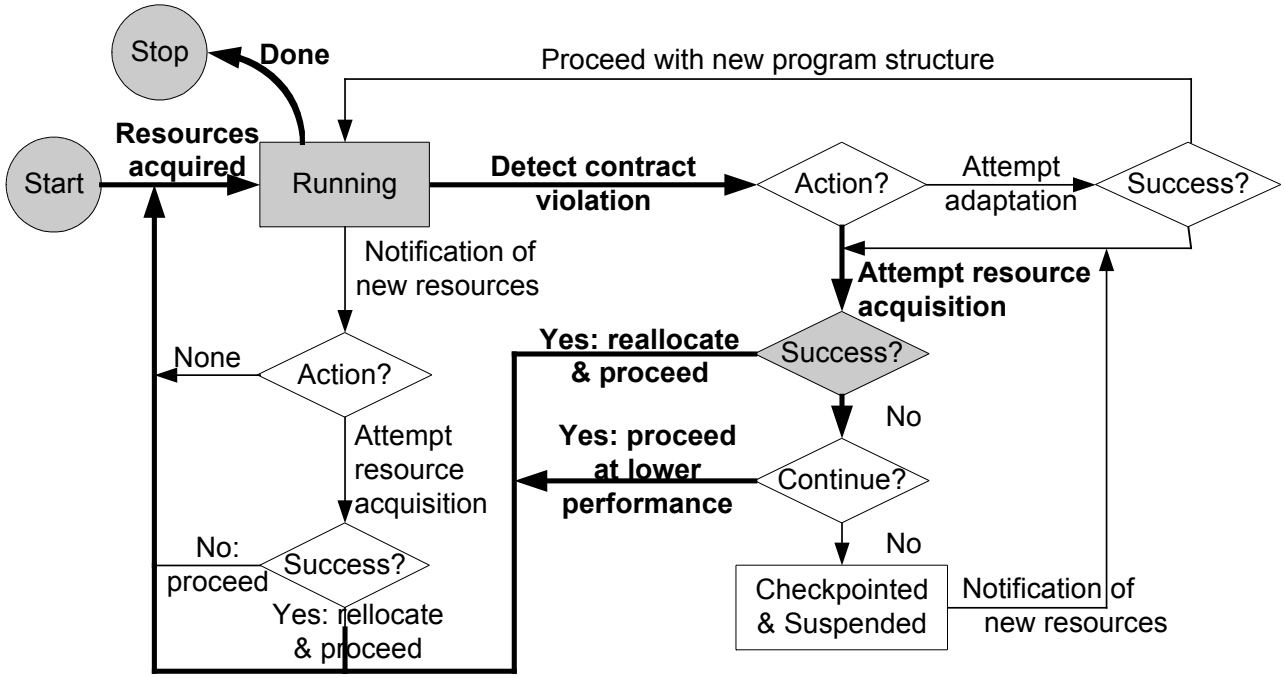


Figure 4.2: Flowchart showing the principal decision points that we envision for our Grid-enabled Cactus system. In boldface and shaded are the components discussed in this report.

We focus here on the elements in boldface and shaded in Figure 4.2, 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 4.3), 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*.

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 in the Set Extended ClassAds language.

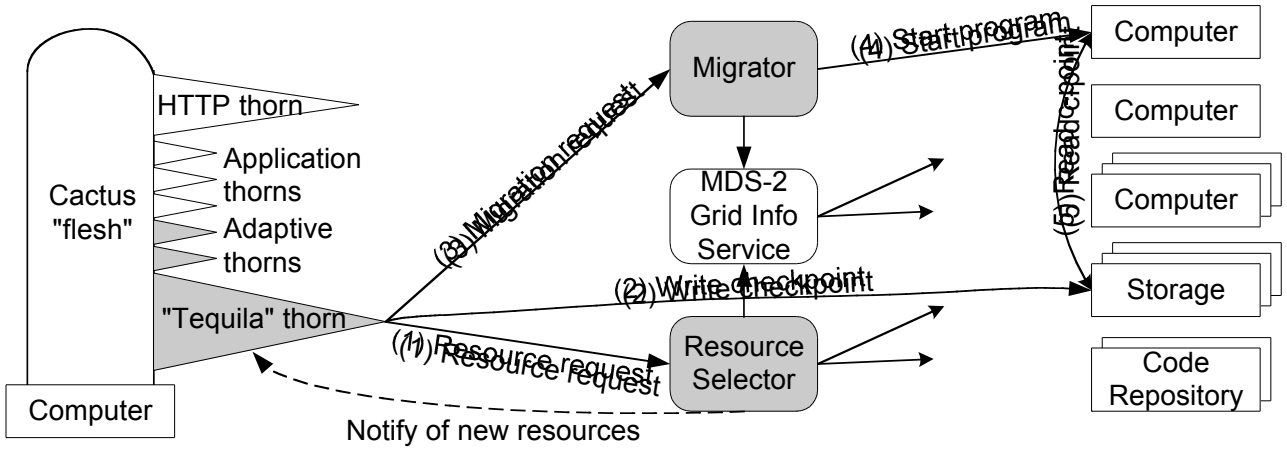


Figure 4.3: Overall architecture of the Grid-enabled Cactus framework, showing in particular the new elements developed in this and related work (shaded), including adaptive thorns, the "Tequila" thorn for managing migration, and the Resource Selector and Migrator services. Resources are discovered, 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.

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 “Tequila” thorn checks the proposed resources against the current resources using a computation performance model. It then checks the amount of time that migration will take to move the application data to the proposed resources from their current location on the current resources, using a migration performance model. If the sum of the migration performance model plus the computation performance model on the proposed resources predicts a smaller amount of real time than the computation performance model on the current resources, the Cactus “Worm” Migration Unit will move the calculations to those proposed resources. 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 Terascale Computing

Even assuming that appropriate resources can be selected and reserved, and that we can reduce sheer complexity by using advanced Grid services, the cited characteristics of the Grid lead to high communication latencies, low bandwidths, and unequal processor powers, which together mean that overall performance tends to be poor. Many specialized techniques can be used to overcome these problems, including Irregular data distributions, Grid-aware communication schedules, redundant computation, and protocol tuning.

We can avoid load imbalances by using irregular data distributions that optimize overall performance. In computing these data distributions, we need information about the application itself, the target computers, and the networks that connect these computers.

We can schedule communication (and computation) so as to maximize overlap between computation and communication, for example, by computing on the interior of a region while exchanging boundary data; we can group communications so as to increase message sizes; or we can organize data distributions or use dedicated communication processors so as to manage the number of processors that engage in inter-machine communication.

We can perform redundant computation to reduce communication costs. For example, increasing the size of the "ghostzone" region in a finite difference code allows us to increase communication granularity significantly, at the cost of otherwise unnecessary computation.

Finally, we can tune a particular protocol based on known characteristics of the application and environment, for example, by selecting TCP window sizes; we can use specialized protocols for wide area communication taking into account application-specific knowledge; or we can compress messages prior to transmission over slow links.

A paper describing the Terascale work was presented at SC'2001 and won the prestigious Gordon Bell award.

4.4 Experiences

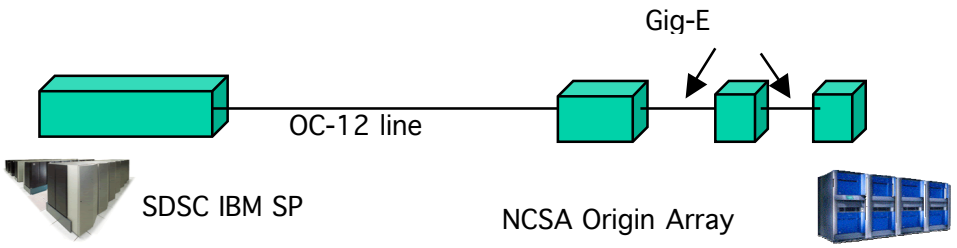
We have demonstrated successful contract violation detection, resource selection, and migration. We have also studied the effectiveness of the procedures and have optimization them. We are currently in the process of completing the intelligent migration using computation and migration performance models. We are also currently in the process of replacing the simple contract violation detection with sensors that will be fed into Pablo, coding the computation migration model into the COP structure, and integrating Cactus into the GrADS ApplicationManager.

These dynamic adaptation techniques were utilized on a collection of a 1024-CPU IBM Power-SP at SDSC and a 256-CPU SGI Origin2000 and two 128-CPU SGI Origin2000 systems at NCSA. The Origins each have two 250 MHz R10000 CPUs per node, and Blue Horizon has eight 350 MHz Power-3 CPUs per node. The machines could allocate 512 MB of memory per CPU. Single-processor performance of our application on the two platforms was

measured to be 306 MFlop/s on the SDSC SP2 and 168 MFlops/s on the NCSA Origins.

All machines had a high-speed inter-processor inter-connection switch providing O(200 MB/s) intra-machine communication bandwidth. Inter-machine communication performance varied. Machines at NCSA were locally connected using (almost dedicated, no competing traffic) Gigabit Ethernets achieving up to 100 MB/s — comparable to intra-machine communication performance.

Between SDSC and NCSA, we achieved a measured application-level bandwidth of at most 3 MB/s, as measured by the `mpptest` program, over what is actually an OC12 (622 Mb/s) network. We have not yet determined the reason for this low performance. Even with such poor performance, the various techniques described below allowed us achieve efficiency of up to 88% on runs across the two sites for this tightly coupled simulation. The scaling was only about 14% without these dynamic adaptation techniques.



Processor Topology for Terascale run

4.5 Education, Papers, and Presentations

Several graduate students at U. Chicago have been involved in the research, notably Matei Ripeanu, Anda Iamnitchi, Chuang Liu, and Lingyun Yang. These students have participated in GrADS meetings and written papers on the research. Two of these papers were presented at Europar'2001 and SC'2001. One of these papers will be presented at HPDC-11, and one appeared in the *International Journal of High Performance Computing Application.* The paper that was presented at SC'2001 won the prestigious Gordon Bell award.

A paper was published in the *International Journal of High-Performance Computing Applications* describing the dynamic adaptation using migration techniques.

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

5 MacroGrid & Infrastructure

The primary goal of the MacroGrid is to provide an environment to support the development of GrADS-specific software and components and to provide a large scale, heterogeneous, real-world experimental execution environment for testing GrADS applications.

The MacroGrid continues to include a heterogeneous collection of hardware including Intel/AMD workstations and clusters, IBM SPs, SGI Origin 2000s, and Suns running a wide range of operating systems including Linux, AIX, Solaris etc. This past year, MacroGrid has matured and provided the required capabilities for three separate and distinct experiments to be run, namely Cactus, ScaLAPACK and the UCSD Resource Selector. New applications (such as FASTA) are being evaluated for deployment and testing across the MacroGrid.

Main activities during this past year included:

- Continued operation of the MacroGrid and clearinghouse,
- Continued operation of the GrADS VO servers,
- Enhancement of VO support and servers,
- Design of the transition plan of the MacroGrid to Globus 2.0,
- Enhancement of the schema about GrADS software objects, and
- Optimization of the interface between the GrADS information service and other GrADSoft tools,
- Investigation of the various design and deployment factors retarding GrADS application performance.

The MacroGrid testbed has been operational for the past two years and has been quite stable and mature with few disruptions of service.

The Clearinghouse (http://www.isi.edu/grads) continues to be an important resource for the GrADS community and applications, providing critical information about points of contact, procedures and policies of the MacroGrid. Two GrADS specific Virtual Organization servers are operated as part of the MacroGrid. These servers, located at ISI and UIUC, provide information on the real-time availability of resources hardware and software, software location, versions and patch levels, etc. This information has been customized for GrADS, enabling researchers and applications to find resources in the heterogeneous testbed. Web pages of the Clearinghouse are the first place to consult, for any researcher interested in accessing the MacroGrid. The VO servers speak LDIF protocol and can be queried by applications.

During the past year, we upgraded the VO servers a few times to address stability and reliability issues in the underlying software. These upgrades were performed to existing MDS 2.0 servers, transparent to the rest of the GrADS community. We have developed a plan to

upgrade the VO servers to current release of Monitoring and Detection Service (MDS 2.1). MDS 2.1 provides more stability, built-in security services (such as authentication), and a substantial enhancement in performance. We anticipate completing this task by the middle of June 2002.

The current deployment of the MacroGrid is based on Globus 1.1.3 for grid services. With the release of Globus 2.0, we have developed a transition plan to upgrade the entire testbed to Globus 2.0. Several sites have already upgraded to this new version and we expect to complete this process by the middle of June 2002. The GrADS project required substantial enhancements to Globus Toolkit™ release 2.0 prior to deployment on the MacroGrid. GrADS experiments running MPICH-G2 and Pablo (for contract monitoring) demonstrated that the underlying MPI implementation is not thread safe. To accommodate the requirements of the GrADS project, special callback mechanisms were developed and made available. We are convinced that these additional mechanisms will be beneficial to other applications and projects.

A significant part of the GrADS VO service is a set of information services (IS) that characterizes the performance characteristics of each node, both in terms if static information (O.S. type, processor clock speed, installed memory capacity, etc.) and dynamically changing information (CPU load, available real memory, network performance). This information is used by components of the GrADSoft toolkit to make initial scheduling and contract decisions.

Because the dynamic information (which is gathered at the bottom layer of the GrADS software stack by the NWS) is performance critical for GrADSoft schedulers, we have developed a new performance abstraction for GrADS—the VO-Grid—that can serve as a high-performance interface to the GrADS-IS.

VO-Grids allow schedulers to select dynamically subsets of the total resource pool that they wish to consider during resource selection. The GrADSoft VO-Grids interface then caches the necessary data near the scheduler, refreshing the information as it becomes available. When a scheduler queries the GrADS-IS for dynamic information, the latest cached data is returned immediately, yielding greatly improved schedule performance over an on-demand, non-cached query. Additionally, the VO-Grid serves the data as a multi-dimensional array that can be indexed by the scheduler. This format is the one used internally by all current GrADSoft resource selectors. By studying GrADS resource selection techniques, we have been able to "push" the task of forming the abstraction down into the IS as part of its interface. Because the IS is responsible for managing this abstraction, the performance and reliability are greatly improved while at the same time the interface is standardized.

We plan to incorporate the VO-Grids interface as part of the "production" release of the NWS for NPACI and to make it available to the wider NWS and Globus user communities. As such, VO-Grids are an example of how GrADS research has come to influence lower-level infrastructure.

During this past year, we have also worked on extending the schema representing the GrADS software objects to meet new GrADS requirements. In addition to software name, patch

levels, location, and installation date, we are developing capabilities to store information about databases location and attributes. These new capabilities will be required by the new applications being planned for the MacroGrid (such as FASTA).

6 MicroGrid

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, and are expected to culminate in the release of a robust, scalable MicroGrid system in the upcoming year.

6.1 Validation/Demonstration of the Current MicroGrid System

We continue to make progress in validating the MicroGrid tools as a high fidelity modeling tool for Grid applications and resource environments. In previous periods, we have validated that the MicroGrid simulation tools can successfully run existing Globus 2.0 applications unchanged (virtualization of the underlying Grid information services, network, and compute resources). Beyond simply executing the Globus 2.0 applications, we have further demonstrated that the emulation/simulation provided by MicroGrid can accurately model performance using examples of the NAS Parallel Benchmarks and the Cactus PDE solver application.

We are now working with the new capabilities of the MicroGrid (see other sections below), and more advanced, dynamic GrADS applications to validate accurate performance modeling in these more challenging dynamic environments. In particular, these efforts involve effectively virtualizing dynamic information services, the GrADS runtime environment, and accurately modeling more complex dynamic application structure (e.g. run, monitor, adapt, run, etc.). To date, we have demonstrated virtualized of the dynamic information services, and successful execution of the decision procedures in the GrADS runtime environment. We are currently still working on successfully modeling the entire dynamic application.

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. Previously existing capabilities of the MicroGrid include: processor modeling, network virtualization, Grid information service virtualization, static and dynamic emulation rates, “limited” network modeling, and Compaq Alpha Linux platform support. In the reporting period, we have added the following additional capabilities – file system emulation, x86 Linux support, and major progress on scalable network emulation. These respectively increase the completeness, accessibility, and provide a foundation for more complete modeling of application and Grid resource behavior.

6.2.1 File System Emulation

In addition to the emulation of computing, network, and Grid information sources already enabled in the MicroGrid, the increasing interest in Data Grids make the modeling of file systems critical. We have built a system for modeling the file-system performance of file servers in a Grid system. This system allows each individual "virtual" MicroGrid resource to have file system latency, transfer bandwidth, etc. specified as a model parameter and modeled accurately. These resources can then be composed into a data grid simulation, using the remainder of the MicroGrid functionality to study complex data-intensive configurations and experiments. The file system models were designed, implemented, and validated as a component model for the MicroGrid. The validation used a range of standard filesystem benchmarks developed by the operating systems community.

6.2.2 x86 Linux Support

The MicroGrid requires a collection of machines to power both application execution and modeling. Previously, we supported Compaq Alpha machines running Red Hat Linux. However, due to the lesser popularity of that configuration, we began work on an x86 version. We now have a robust, working version of the MicroGrid for x86 Red Hat Linux, and in fact have moved over to that platform as our primary development system. For experimental purposes, we have configured a cluster of 30 machines. None of this hardware was purchased with funding from this grant. This x86 platform will enable the more effective transfer of the MicroGrid tools to the GrADS project team and the overall community.

6.2.3 Scalable Network Emulation

To meet the needs of scalable Grid modeling, we have been researching novel techniques and building a scalable network emulators. These efforts are a critical part of the MicroGrid effort, as the network emulation/simulation tools available do not meet our scalability and performance requirements. Our approach is to build a scalable network emulator built on distributed discrete event simulation engine to 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 efforts: 1) exploratory work to study and analyze the alternative synchronization algorithms and ways to exploit network topology, 2) to build a prototype and test these alternatives, and 3) to validate the emulator and use it for performance analysis of some real world applications.

So far, we have studied the currently available synchronization algorithms such as null-message algorithm, global barrier, and optimistic algorithms. We believe that null-message-like algorithm is good a large scale distributed simulation, but its success will depend on how much we can limit the overhead of null-messages. We are implementing this algorithm and adapting it to our emulator's requirements. The global barrier algorithm is also promising, considering the emergence of low-latency cluster interconnection. We will also experiment with it. And we will also try to mix both of them to achieve better performance. The software of network emulators mainly consists of three parts, the first is build a distributed simulation

engine, the second one is implement the network protocol model, such as TCP/IP, OSPF, BGP protocols; the third one is the emulation functionality and dynamic simulation rate parts.

In the past year, we have completed the simulation engine, and the implementation of two synchronization algorithms. We have also completed the network protocol modeling (a significant number of protocol modules). We are currently proving out the scalability of the system (robustness and number of simulated resources and simulating resources) on our cluster systems (32 nodes). We plan to explore scalability on larger systems such as the TeraGrid systems being deployed by NPACI/The Alliance. We have also completed basic emulation functionality. In the coming year, we will be able to focus on both integration of these tools with the MicroGrid infrastructure, and application of its capabilities to do large-scale Grid simulation experiments.