

## I. Project Activities

The goal of the Grid Application Development Software (GrADS) project is to simplify distributed heterogeneous computing 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 (6/1/02-5/31/03), GrADS research focused on the five inter-institutional efforts described in the following sections: *Program Execution System (PES)*, *Program Preparation System (PPS) & Libraries*, *Applications*, *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 weekly technical teleconferences involving researchers from each of the GrADS sites, PI teleconferences, a PI meeting (11/5-6/02), workshops (11/4-5/02 & 5/7-8/03), and communication via GrADS mailing lists. A driving focus for activities during this reporting period was the creation of demonstrations at the SC 2002 conference (11/17-22/02) and planning additional activities for the SC 2003 conference (11/16-21/03; to be reported next year). These demonstrations are an excellent test of our software, as well as our means of presenting our research to broader audiences.

## **1 Program Execution System (PES)**

Efforts over the last year have continued fundamental research on the components of the GrADS execution system and on enhancement and maintenance of testbeds for experimenting with those components. The development activity produced a successful demonstration of the execution system components at SC2002 in October 2002 in which several programs were launched and executed on the Grid using GrADS components.

### **1.1 Scheduling and Rescheduling**

To achieve and maintain application performance on Computational Grids, it is vital that the GrADS system provides an intelligent matching of application resource requirements and Grid resource conditions. Over the past several years, we have developed a scheduling component that utilizes application-specific models and dynamic resource information from the GrADS information service to automatically identify an application-appropriate set of resources 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 last year and demonstrated at SC2002 as a part of the GrADS Program Execution System.

During the current year, we have focused on adding rescheduling of applications to the GrADS execution framework. To this end, we are pursuing two general strategies. Otto Sievert's M.S. thesis work at UCSD developed a lightweight rescheduling strategy based on swapping active processes onto lightly-loaded processors. Independently, Sathish Vadhiyar's Ph.D. work at UTK performs fully general N to M rescheduling by using the COP performance models to identify profitable opportunities for rescheduling, and implementing a checkpoint-restart mechanism to migrate the application. Both rescheduling mechanisms have been tested with modified versions of the GrADS framework and experiments are underway to determine advantages and disadvantages of each.

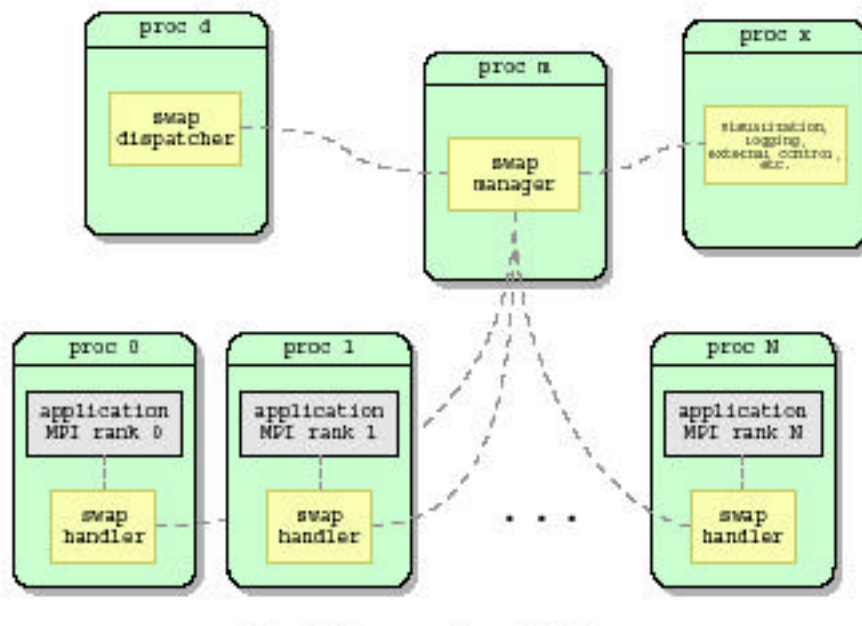
We describe both methods in the subsections below, as well as some more preliminary work on other aspects of rescheduling.

#### **1.1.1 Rescheduling by Process Swapping (N-to-N Rescheduling)**

Recently, GrADS researchers at UCSD (with extensive collaborations at other sites) have embarked on investigations of support for process swapping in GrADSoft. Although we have argued from the beginning that rescheduling is a key capability in any Grid programming system, our early experiments convinced us that there was a need for a lightweight swapping mechanism, rather than migration by parallel heterogeneous checkpoint/restart or full rescheduling with process migration. After extensive discussions, we developed an architecture for swapping MPI-1 processes while the application was running. In 2003, parts of this architecture were implemented successfully in GrADSoft. In addition, some simulation work has been done to evaluate three swapping policies: a greedy policy, a safe policy, and an environmentally friendly policy.

The basic idea behind MPI process swapping is as follows. Consider a parallel iterative application that runs on  $N$  processors. We *over-allocate* processors:  $N$  active processors that (initially) perform the computation and  $M$  (initially) inactive “spares” that give the application the opportunity to swap any or all of the active processors at each iteration. The total number of processors is therefore  $N+M$ . Because entire MPI processes are swapped, this approach means that data redistribution is not allowed, which limits the ability to adapt to fluctuating resources. However, the resulting simple system is a practical solution for many practical situations. In particular, we are primarily concerned with heterogeneous time-shared platforms in which the available computing power of each processor varies throughout time due to external load (e.g. CPU load generated by other users and applications), and where there are only a few simultaneous parallel applications. Although our approach is applicable when resource reclamations and failures occur, to date we have focused solely on rescheduling for performance issues. Our method is also easily incorporated into existing programs, requiring as few as three lines of source code change.

MPI process swapping is implemented as a set of run-time services that interact with a modified MPI library interface. The run-time architecture for a swappable application comprises five main components: the swap-enabled MPI application itself, swap handlers, a swap manager, a swap dispatcher, and the swap tools.



The figure above shows the swap run-time architecture and describes the communication patterns between the swap components. The swap handler modules are transient network services instantiated on the same processor as each MPI process (active and inactive) for the lifetime of the swap-enabled application. The swap handler module is the main communication link between the application and the other swap components, as well as containing performance measurement capabilities. The swap manager (one per application) is the intelligence of the swapping operation. Information from each MPI process and each

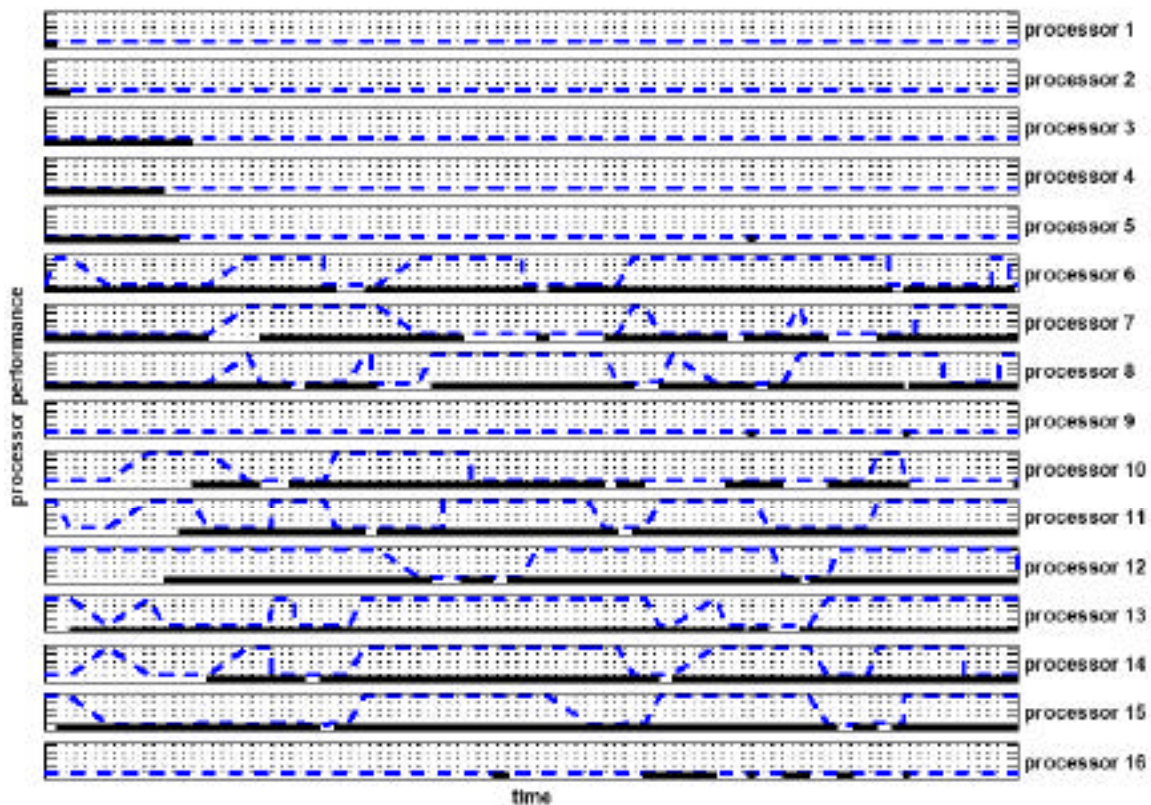
processor is analyzed by the swap manager, which triggers a process swap if one is needed. The swap dispatcher is an always-on remote service at a well-known location (network host/port). The dispatcher fields requests for swapping services and launches the swap manager for each application. The swap tools are a collection of utilities designed to improve the usability of the swap environment, including swap information logging and swap visualization to track an application's progress, and a swap actuator to manually force a swap to occur.

The swap services interact with the MPI application and with each other in a straightforward asynchronous manner. First, a user launches a swap-enabled MPI application on  $N+M$  processors (only  $N$  of which will be active at any given time). MPI process 0 contacts the swap dispatcher during initialization, which launches a swap manager for the application and tells process 0 how to contact the manager. The root process passes this information to all MPI processes in the application, and the swap manager starts a swap handler on the same processor as each MPI process. Once the swap handlers are initialized, the application begins execution. While the application is executing, the swap handlers gather application and environment performance information, both by passive monitoring and by active probes, and feed it to the swap manager. The swap manager analyzes all of this information and determines whether or not to initiate a process swap.

If the swap manager decides that two processes should swap – for example, to move an active process to an unloaded processor – it sends a message to the swap handler that cohabitates with the active root process (the MPI process 0 in the group of active processes). This process periodically contacts its swap handler (through a call to `MPI_Swap()`, one of the few new routines in our system). When this happens after the swap manager's message, the swapping processes exchange information and data, and change their states (from active to inactive and vice versa). Processes not involved in the swap continue to execute the application uninterrupted. In order to minimize the impact to user code, and yet still provide automated swapping functionality, MPI process swapping hijacks many of the MPI function calls through a combination of `#define` macros and function calls. In particular, swap library may transform the destination and source of a message in order to reroute it to the process' new location.

The execution continues in this fashion until the application completes. At that time, each MPI process sends finalization messages to its swap handler before quitting. The swap handler in turn registers a finalization message with the swap manager, and then quits. Once all the swap handlers have unregistered with the swap manager, it sends a quit message to the swap dispatcher, and shuts down.

As an example of the behavior of a swapping application, we show a toy MPI application that was designed to quickly and simply evaluate the implementation robustness of the process swapping services. The application used eight active (out of sixteen total) MPI processes running on time-shared workstations. In the graph below, the blue lines represent processor performance for the application (high is good), while the black bars at the bottom show when the processor was active. Note how swapping gravitates toward the machines with the highest performance, this run also shows the natural dynamism of a typical production environment.



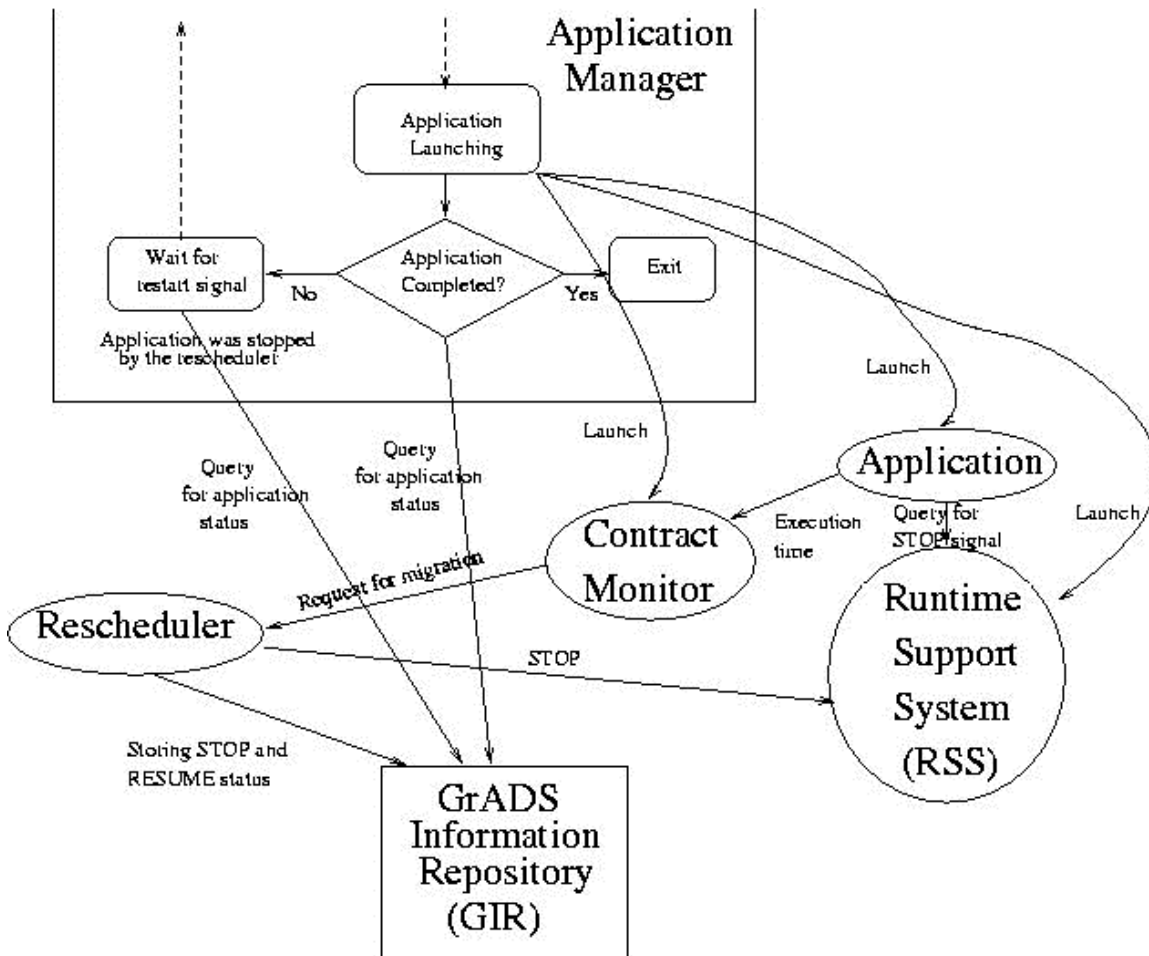
This experiment used a very simple swapping policy with no hysteresis and very simplistic load monitoring. Other experiments performed with this greedy policy, which swaps at any remote indication of a performance improvement, revealed that this policy can be hyperactive. Further experiments compared this policy and two other policies (a risk-averse safe policy that only swaps when a performance increase is all but guaranteed, and an environmentally friendly policy that does not swap to faster processors unless the entire application performance improves) against checkpoint/restart and dynamic load balancing in a simulation environment. While the results are still preliminary (and application-dependent), it appears that swapping provides performance benefits comparable to these other techniques.

### 1.1.2 Rescheduling by Checkpoint/Restart (N-to-M Rescheduling)

Computational Grids involve large system dynamics such that the ability to migrate executing applications onto different sets of resources assumes great importance. Specifically, the main motivations for migrating applications in Grid systems are to provide fault tolerance and to adapt to load changes on the systems. The University of Tennessee has developed a migration framework for performance oriented Grid systems that implements tightly coupled policies for both suspension and migration of executing applications and takes into account both system load and application characteristics. The main goal of the migration framework is to improve the response times for individual applications. The migration of applications in our migration framework is dependent on the ability to predict the remaining execution times of the applications that in turn is dependent on the presence of execution models that predict the

total execution cost of the applications. The framework has been implemented and tested in GrADS.

The ability to migrate applications in the GrADS system is implemented by adding a component called *Rescheduler* to the GrADS architecture. The migrating numerical application, *migrator*, the *contract monitor* that monitors the application's progress and the *rescheduler* that decides when to migrate, together form the core of the migration framework. The interaction between the different components involved in the migration framework is illustrated below.



The Rescheduling infrastructure

We have implemented a user-level checkpointing library called SRS (**Stop Restart Software**). By making calls to SRS, the application possesses the ability to checkpoint data, to be stopped at a particular point in execution, to be restarted and continued later on a different configuration of processors. The SRS library is implemented on top of MPI at the application

layer and migration is achieved by clean exit of the entire application and restarting the application over a new configuration of machines. The approach followed by SRS allows reconfiguration of executing applications and portability across different MPI implementations. An external component (e.g., the rescheduler) wanting to stop an executing application interacts with a daemon called Runtime Support System (RSS). RSS exists for the entire duration of the application and spans across multiple migrations of the application. Before the actual parallel application is started, the RSS is launched by the application launcher on the machine where the user invokes the GrADS application manager. The actual application interacts with the RSS through the SRS library.

Contract Monitor is a component that uses the Autopilot infrastructure to monitor the progress of the applications in GrADS. An autopilot manager is started before the launch of the numerical application. The numerical application is instrumented with calls to send the execution times taken for the different phases of the application to the contract monitor. The contract monitor compares the actual execution times with the predicted execution times and calculates the ratio between them. The tolerance limits of the ratio are specified as inputs to the contract monitor. When a given ratio is greater than the upper tolerance limit, the contract monitor calculates the average of the computed ratios. If the average is greater than the upper tolerance limit, it contacts the rescheduler, requesting that the application be migrated. If the rescheduler refuses to migrate the application, the contract monitor adjusts its tolerance limits to new values.

Rescheduler is the component that evaluates the performance benefits that can be obtained due to the migration of an application, and initiates the migration of the application. It operates in two modes: *migration on request* and *opportunistic migration*. When the contract monitor detects intolerable performance loss for an application, it contacts the rescheduler requesting it to migrate the application. This is called migration on request. In other cases, if a GrADS application was recently completed, the rescheduler determines if performance benefits can be obtained for an executing application by migrating it to use the resources that were freed by the completed application. This is called opportunistic rescheduling.

The rescheduling framework has been tested on the GrADS testbed. The University of Tennessee plans to integrate the rescheduling framework into the GrADSoft framework.

### **1.1.3 Other Rescheduling Work**

In addition, we have been exploring a new design for the GrADS scheduler that would accommodate two goals. First, we need to support dynamic spawning of new tasks as previous tasks are completed. Such a facility in the scheduler would permit the incorporation of facilities from the Condor DAGMAN scheduler. In addition, dynamic scheduling is a requirement of several of the applications described in Section 4, including GrADSAT and EMAN. Our second goal is to handle applications that use functions provided by libraries that are pre-installed on resources across the Grid. The envisioned facility would be similar to Grid services in that an application could specify that some of its components must run on resources where specialized functionality is preinstalled. This means that the scheduler must be able to select from among those resources at launch time. A key issue is how the

performance models for such applications are constructed. Our goal is to provide performance models for all functions that are preinstalled on a given resource and to integrate these performance models into application performance models at launch time. This topic is discussed in more detail in the program preparation system section. Our plan is to demonstrate a rudimentary version of this facility in the fall.

Research was also conducted in the area of metascheduling in the context of GrADS. The research was oriented towards building scheduling techniques so that the GrADS framework can handle multiple jobs at the same time. The metascheduler that was built receives candidate schedules of different application-level schedulers and implements scheduling policies for balancing the interests of different applications. The goals of the metascheduler include:

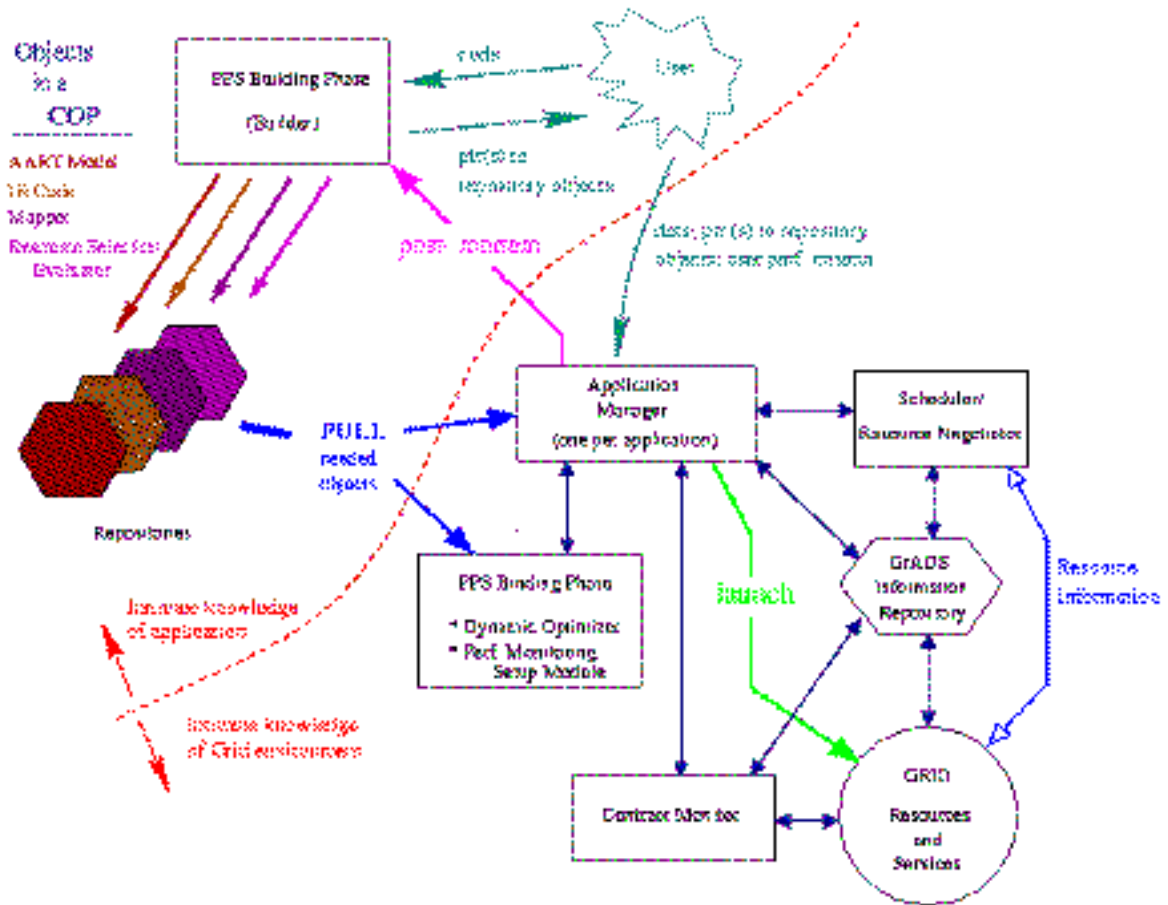
- verifying that the applications made their scheduling decisions based on conditions of the system when competing applications are executing.
- accommodating short running jobs by temporarily stopping long running and resource consuming jobs.
- facilitating new applications to execute faster by stopping certain competing applications.
- minimizing the impact that new applications can create on already running applications.
- migrating running applications to new machines in response to system load changes to improve the performance or to prevent performance degradation.

The metascheduling architecture was tested on the GrADS testbed and provided encouraging results.



## 1.2 Application Manager

The overall program execution and launch facility within GrADSoft is depicted in the figure below.



**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, there must be a process that initiates the resource selection, launches the problem run, and sees its execution through to completion. In the GrADS execution framework, the *application manager* is the process that is responsible for these activities—either directly or through the invocation of other GrADS components or services. In this scenario, individual GrADS components only need to 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, we spent a significant amount of time developing the GrADS application manager, culminating in the successful SC2002 demonstration. The developments include the following:

- The basic Application Manager design and implementation work was completed. The result was used to successfully run demos at SC02 of ScaLAPACK, Cactus, and FASTA applications.
- The application programming interface not only supported application specific COPs for the SC02 demo codes, but proved general enough to make building a set-extended ClassAd translator straight-forward once the ClassAd parser was done. This translator will automatically build the basic COP pieces from an initial ClassAd. Work finished in year 3 on functional attributes was fundamental to building the translator. The translator was used with the Cactus demo at SC02.
- The Application Manager functionality has been extended to support n-to-n rescheduling via Sievert's lightweight rescheduling strategy.
- Necessary Application Manager modifications to support other year 4 work on new binding strategies are being investigated.

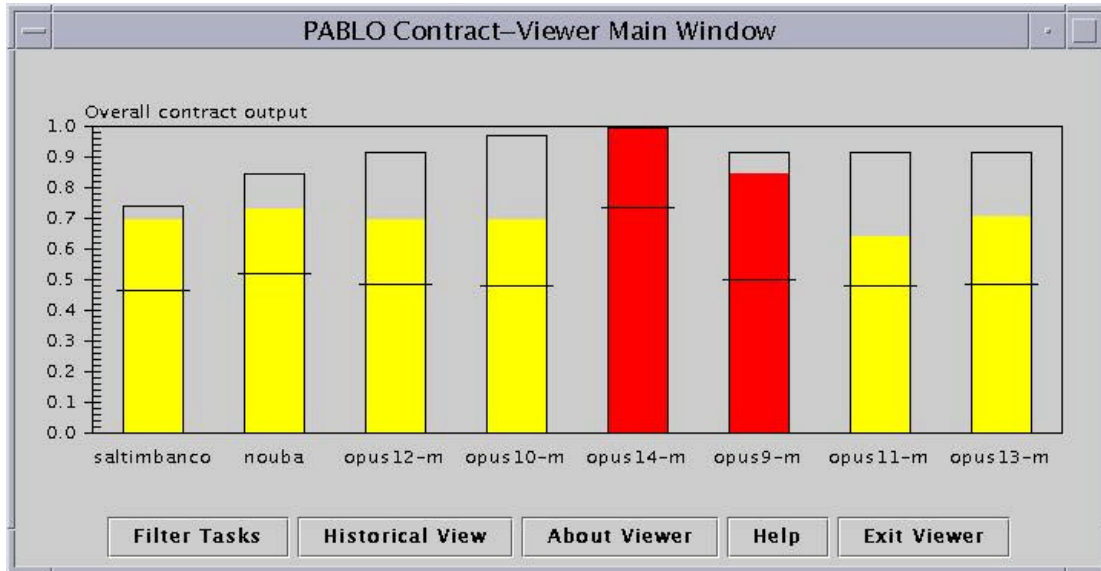
### **1.3 Contract Monitoring**

A performance contract specifies the expected performance of an application on a given set of Grid resources. During execution, application performance measurements are compared to contract specifications by a 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. The expected performance values can be derived from a variety of sources, including application and library developer knowledge, compilers, execution history and user input.

During the reported period, we have significantly expanded and enhanced our contract-monitoring infrastructure, and we more tightly integrated its components with GrADSsoft, the software base being developed in GrADS. The integrated infrastructure can now handle several kinds of applications from different areas. It also includes a real-time visualization tool that can display contract evaluation results and that allows the user to modify contract parameters while application execution proceeds. All these new features of the contract-monitoring facility were shown during the GrADS demonstrations during the Supercomputing conference, in November 2002.

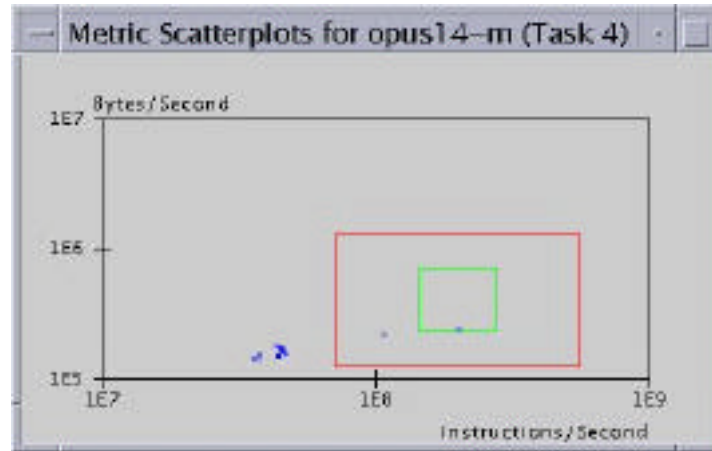
As an example of the new visualization features now supported by our infrastructure, the figure below shows the output of the contract monitor for an execution of the Cactus Wavetoy code on the GrADS testbed. This particular execution was launched on machines located at Illinois and San Diego. Contract outputs are fuzzy variables that can assume any value between 0.0, corresponding to no violation, and 1.0, corresponding to full contract violation. Each colored bar in the figure represents the current contract evaluation in a specific testbed

node, while its color corresponds to the severity of a violation; the bar's envelope indicates the *high-water* mark for that contract. During this execution, an external load was imposed on one of the resources (opus14). This load skews the observed performance –that delay is propagated across all tasks, resulting in some degree of violation on all processors.



**Contract monitor visualization of Cactus-Wavetoy execution on GrADS testbed**

The contract outputs above are computed as a fuzzy combination of individual contract outputs for each monitored performance metric. Thus, one can also analyze individual metrics to understand the causes of contract violations. The figure below shows a combined scatterplot view of the two most affected metrics for the same processor. The bounded regions in the metric space corresponding to the borders of the contract are codified in the contract rule base. Points inside the inner rectangle correspond to no contract violation, and points outside the outer rectangle correspond to a total violation. Points that lie between the two rectangles represent partial contract violations for those metrics. By manipulating the rectangle borders, users can interactively redefine the contract parameters to desired acceptance thresholds.



**Contract regions and observed performance for two Wavetoy metrics**

In the recent months, we have migrated this infrastructure to the latest Globus version installed in the GrADS testbed, and expanded its scope of applicability to other architectures. We have completed the porting to IA-64, and we have implemented support for a variety of compiler versions. This effort led to a new release of our Autopilot toolkit (release 2.4), which is now available at the Illinois website.

We also began adapting some of the contract monitor components, targeting rescheduling goals. We implemented an API to the contract monitor, which provides current contract values to applications. Using such values, one can build applications that are self-adaptive and change their behavior based on runtime conditions. Meanwhile, we are adding support to enable execution migration across different testbed nodes. We have also begun to create contracts with temporal components, where violation decisions are based on current as well as on observed status.

#### **1.4 GrADS Program Execution System Demonstration**

In November 2002, at SC2002, we successfully demonstrated a coordinated GrADSoft system including prototypes of all major components of the GrADS architecture. The prototype GrADSoft system was demonstrated on ScaLAPACK, Cactus, and a genomic sequencing application based on FASTA. 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. In addition, we demonstrated GrADSAT, a satisfiability application on a modified version of the scheduler.

The demonstration prototype system included simple versions of most of the components mentioned in the Application Manager section, 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 consisted of simple interfaces to MacroGrid services such as NWS.

We are now in the process of planning our new demonstration for SC2003 (November 2003). New features planned for this demonstration include rescheduling based on contract violations and a new approach to scheduling and binding of applications that use pre-installed library components. In addition, the GrADSoft prototype will schedule applications on architecturally heterogeneous resources, including at least IA-32 and IA-64 based system, along with those that involve Linux. We may also be able to demonstrate GrADSoft working on different operating systems.

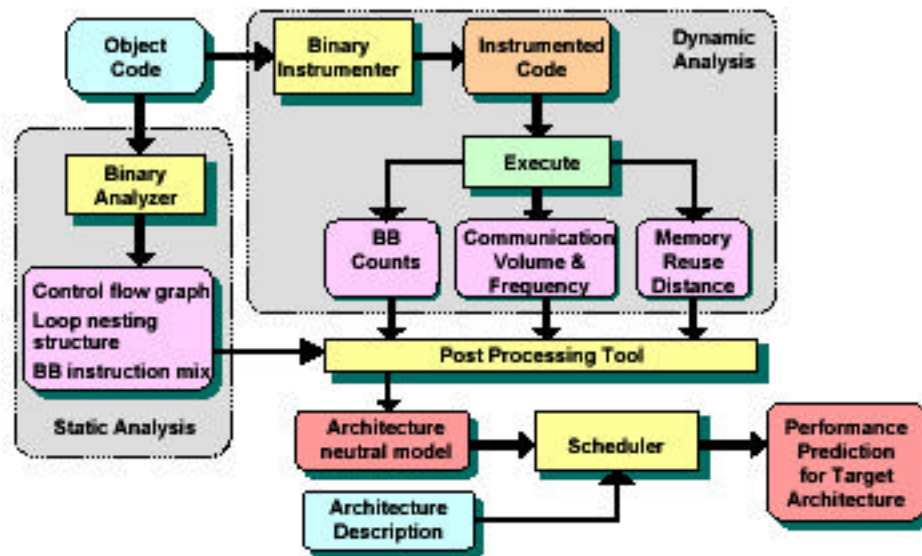
## **2 Program Preparation System (PPS) & Libraries**

The goal of the GrADS Program Preparation System (PPS) 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 focused in five areas: developing software technology for semi-automatically constructing scalable performance models for parallel applications, automated generation of mappers, 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. Previous work showed that manual construction of accurate models was quite difficult. Accordingly, a major thrust of our recent program preparation system work is 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.

Building accurate performance models for parallel applications is difficult. Simply knowing the number of floating-point operations a scientific application executes provides little indication of its performance. Scientific codes rarely achieve peak performance. On a single node, memory hierarchy latency and bandwidth are significant limiting factors. Also, an application's instruction mix can dramatically affect performance; today's superscalar processors can execute multiple instructions in parallel if they are provided with the right mix of instructions. For parallel programs, communication frequency, communication bandwidth and serialization complicate the situation further. Our approach aims at building parameterized models of black-box applications in a semi-automatic way by using architecture-neutral Application Signatures. We build models using information from both static and dynamic analysis of an application's binary. We use static analysis to construct the control flow graph of each routine in an application and to look at the instruction mix inside the most frequent executed loops. We use dynamic analysis to collect data about the execution frequency of basic blocks, information about synchronization among processes and reuse distance of memory accesses. By looking at application binaries instead of source code, we are able to build language-independent tools and we can naturally analyze applications that have modules written in different languages or are linked with third party libraries. By analyzing binaries, the tool can also be useful to both application writers and to compiler developers by enabling them to validate the effectiveness of their optimizations. The overall structure of our performance instrumentation and prediction system is shown in the figure below.

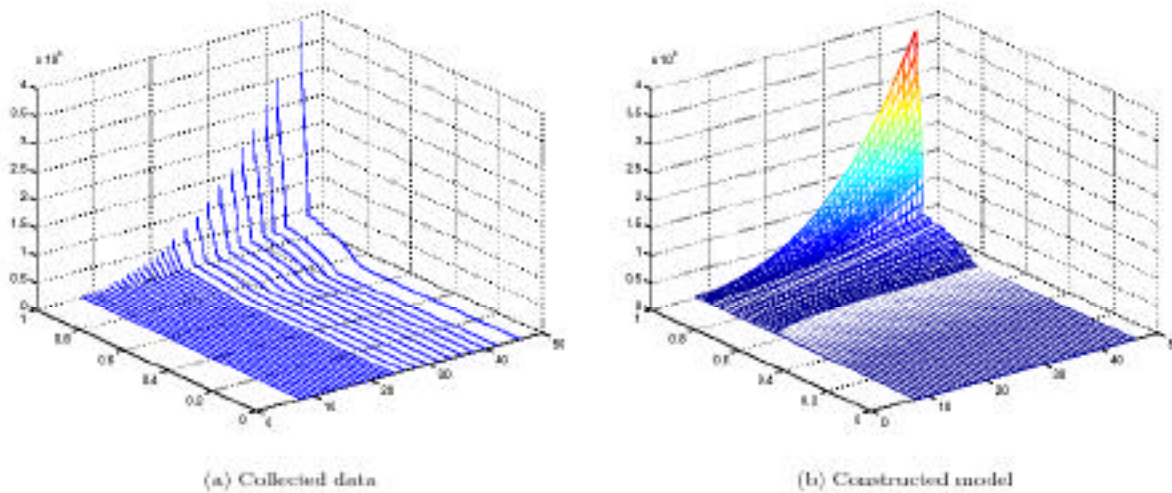


To semi-automatically build an architecture-neutral model of a black-box parallel application, we collect several types of dynamic information by instrumenting the binary using a tool based on the EEL library. The instrumenter analyzes each routine by building its control flow graph, computing Tarjan intervals on the graph (to recover loop nesting structure). Using this information, the instrumenter places instrumentation on selected flow edges such that the estimated overhead for executing instrumentation code is minimized. The inserted instrumentation consists of a data collection routine, from a shared library, to increment a histogram of basic blocks executed between synchronization points, to characterize memory behavior, or to record the communication partner and the amount of data sent and/or received at a synchronization point. A post-processing phase uses the counts on instrumented edges to reconstruct counts for all edges and basic blocks, from which we derive counts of executed instructions and reuse distance values (i.e. the number of distinct memory locations touched since the last access).

Once the instrumentation is inserted, the principal challenge is to assimilate the data that can be collected into accurate models of application performance. To build parameterized models that can predict performance for data sizes that we haven't measured, we collect data from multiple runs with different and determinable input parameters. While building a model for an entire parallel application is our ultimate aim, to date we have concentrated on building parameterized models for single node executions, a hard problem in its own right. From the static analysis and dynamic measurements, we create a control flow graph annotated with execution frequencies of each basic block. An instruction schedule analysis tool converts these paths to generic RISC instructions, computes their execution cost for a specific architecture (which may be different from the architecture where the performance data was collected), and predicts computation execution time (ignoring the effect of the memory hierarchy). We are currently working on translating our data on memory reuse distance into an estimation of latency for a given target memory hierarchy.

Our plan is to build a model for the execution frequency of each loop, parameterized by the

inputs, by fitting polynomial curves to the data from several independent runs. A similar approach models the behavior of each memory instruction and to predict the fraction of hits and misses for a given problem size and cache configuration. We have found that accurate memory hierarchy prediction requires analyzing a complete histogram of reuse distances collected by our binary instrumentation. We divide the memory accesses for an instruction into multiple bins and then compute two polynomials for each bin, one for the number of accesses that are part of that bin and one for how the average reuse distance of those accesses changes with problem size. While this is still an area of active research by our group, one can see qualitatively from the graphs below that the current models have reasonable quality.



## 2.2 Automatic Mapper Generation for HPF Programs

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 focused on construction of mappers from program task graphs, which can be constructed from MPI programs or from programs written in high-level languages like HPF.

We built a prototype tool to build a mapper that maps HPF applications onto the Grid using the GrADS infrastructure. A substantial portion of the work leverages infrastructure from the NSF POEMS project to automatically generate a “task graph” from HPF application source. The mapper uses the SPMD task graph representation as the basis for mapping application processes to Grid nodes. Using the mapper generated, we have been able to launch an HPF application, namely *tomcatv*, on the Grid. To our knowledge this is the only instance of an HPF application running on the Grid. We compared the automatically generated mapper with the generic site-aware mapper in GrADSoft. The results show that, for Grid runs of the application on 16 processors distributed over three clusters, our mapper has performance comparable to the generic site-aware mapper in GrADS. On the average, the mapper generated by the tool performs about 5% better than the generic site-aware mapper.



### 2.2.1 HPF Application

The HPF application we chose for the purpose of demonstration was *tomcatv*, a vectorized mesh generation program. It is one of the programs in the Spec CFP'95 benchmark suite. It generates a 2D boundary fitted coordinate system around general geometric domains. The application calculates residuals, finds the maximum value of the residuals and then solves a tri-diagonal system in parallel.

### 2.2.2 HPF Mapper Tool Design

The tool was built on the POEMS task graph construction infrastructure. The task graph captures the static parallel structure of the application. It is a directed graph representing not only control flow of the application but also communication, synchronization and computation partitioning of the application. Each node of the graph denoting a task may represent one of the following types: control flow nodes for loops and branches, procedure calls, communication, or pure computation. Edges between nodes may denote control flow within a processor or synchronization between different processors due to communication tasks. A key aspect of the static task graph is that each node in the STG actually represents a set of instances of the task, one per process that executes the task at runtime. Similarly an edge in the STG represents a set of edge instances connecting pairs of dynamic node instances. Symbolic integer sets are used to describe the set of instances for a given node. The edge mappings enable precise symbolic representations of arbitrary regular communication patterns. A Communication Event Descriptor (CED), kept separate from the STG, captures all information about a single communication event. It captures the communication pattern for a communication event.

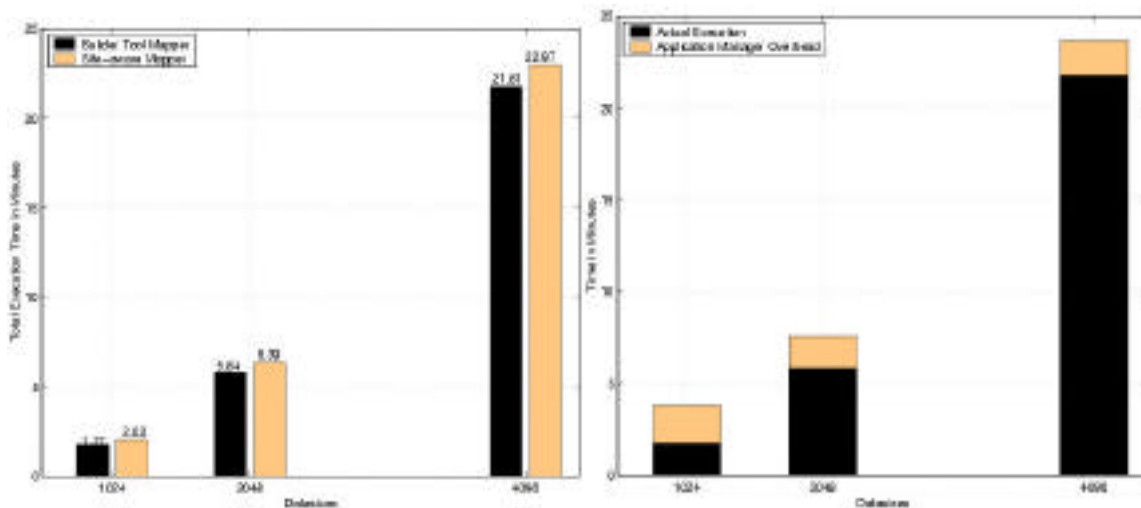
Once we have the task graph representation for the HPF application and the CEDs, we use them to discover the communication performance model of the application. This performance model is represented by a complete graph of process nodes with edges between the nodes denoting communication characteristics between two processes. Aside from the application performance model, the mapping algorithm also needs information about the network characteristics and state of the available compute and storage resources. These are obtained from NWS and MDS through the GrADS infrastructure. The mapping algorithm uses a best-fit heuristic to map a process onto a real processor. It basically maps process pairs having high communication between them to a processor pair having the best latency/bandwidth metric between them.

### 2.2.3 Experiments

The testbed for the experiments was the GrADS MacroGrid that consists machines from clusters at three GrADS sites – UCSD, UIUC and UTK. As a proof of concept, we have been able to launch *tomcatv* across three clusters on the GrADS testbed using 4, 8 and 16 processors across the three clusters. This is the first instance of an HPF application running on the Grid. We launched the same using the GrADS Application Manager and the whole GrADSsoft infrastructure. We have also run the experiments with different data sizes - matrices of size 1024, 2048 and 4096. All these experiments were repeated for two mapping

cases; one that uses the generic GrADSoft site-aware mapper and other that uses the mapper generated by the Mapper tool.

We validated the mapper generated by the tool by comparing it with the generic site aware-mapper already existing in GrADSoft. The figure on the left below shows execution time plots for the three problem sizes, each running on 16 processors. For each problem size, the two bar diagrams correspond to two different mappers used. The left bar corresponds to the total execution time for the Builder tool Mapper and the right bar corresponds to the generic GrADSoft Mapper. From the plot, it is clear that the mapper generated by the tool performs about 5% better than the generic mapper. The figure on the right below shows that the GrADSoft overhead on the application running time is a small constant time.



### 2.3 Binder/Dynamic Optimizer

This work at Rice University 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.

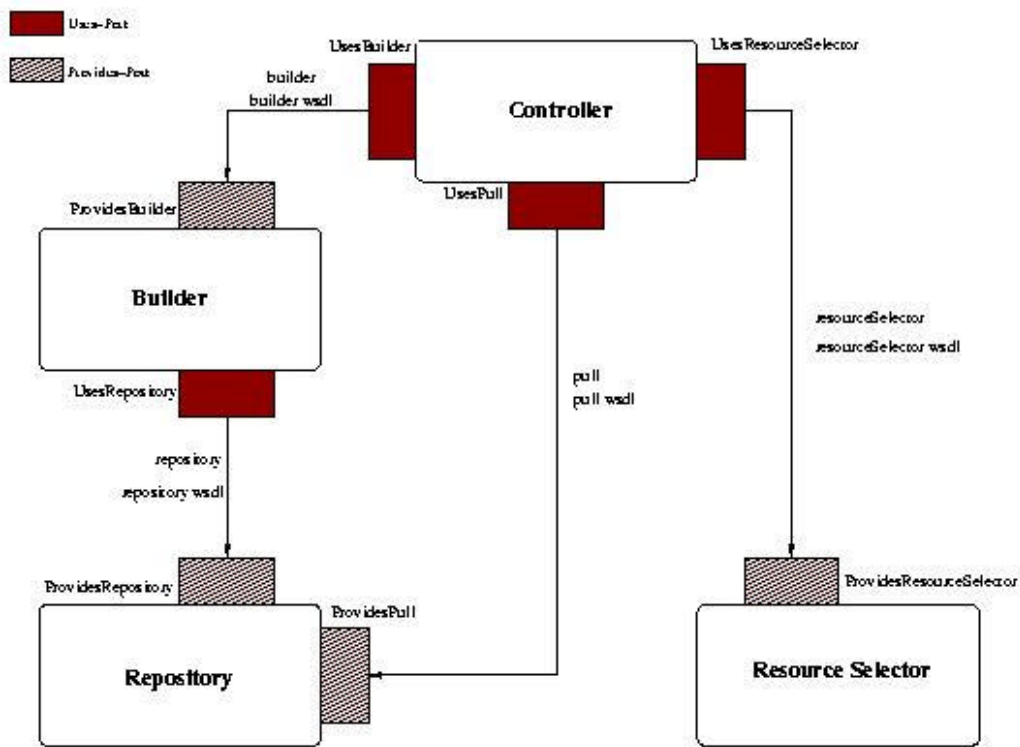
On top of this analysis infrastructure, we have been developing tools for x86 to x86 binary translation. This work has two aims. First, it will serve as the basis for the tool that inserts probes and monitoring code into application binaries to track their execution progress and performance. In year 4, we successfully fielded a binder that inserted the required initialization and finalization calls to the contract monitor into our demo applications. This simplified binder was successfully used for the demonstrations at the SC 2002 conference. Second, it will also serve as the basis for optimization of binaries including low-level value numbering and re-scheduling. This work was the primary basis of a Master's thesis this year by Anshuman Dasgupta.

Vizer, the software described in Dasgupta's thesis, is a framework for conducting high-level optimizations on binary programs. Vizer analyzes binary files and reconstructs data structures and control flow that were present in the high-level source code used to create the binary. In the GrADS context, this information is used to instrument x86 executables on Grid computing environments. In other work, the same information can be used to implement optimizations that are otherwise not possible in binary optimizers. Vizer conducts one such optimization: the vectorization of Intel x86 object code.

## **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 Grid-enabled Library Development

GrADS library work has concentrated on two libraries:

- The UHFFT library (University of Houston Fast Fourier Transform)
- The ScaLAPACK library (Scalable Linear Algebra Package)

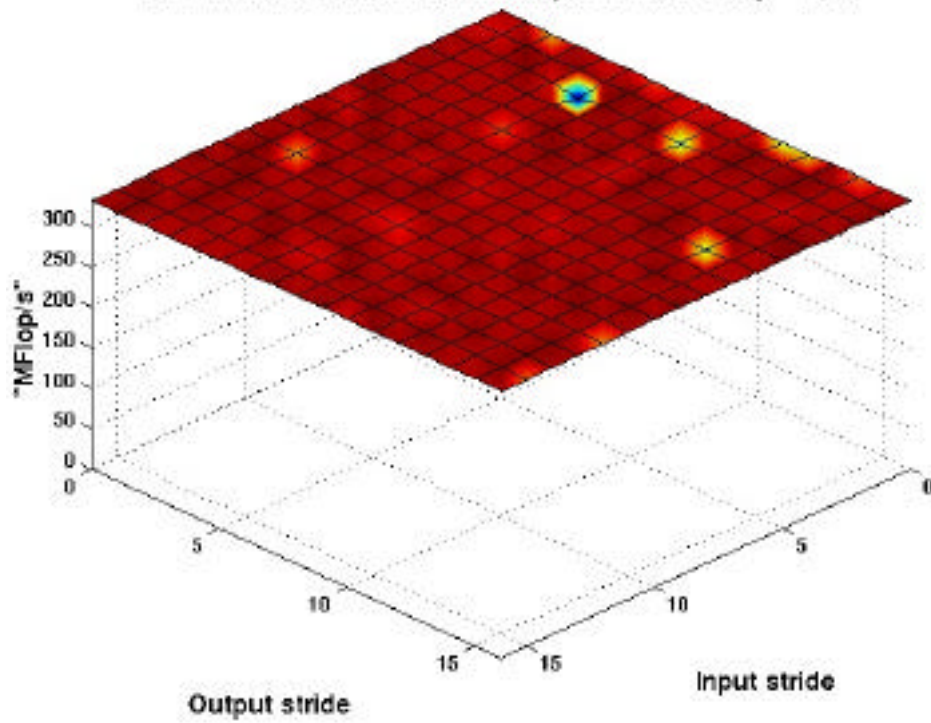
We describe each below.

### 2.5.1 UHFFT Library

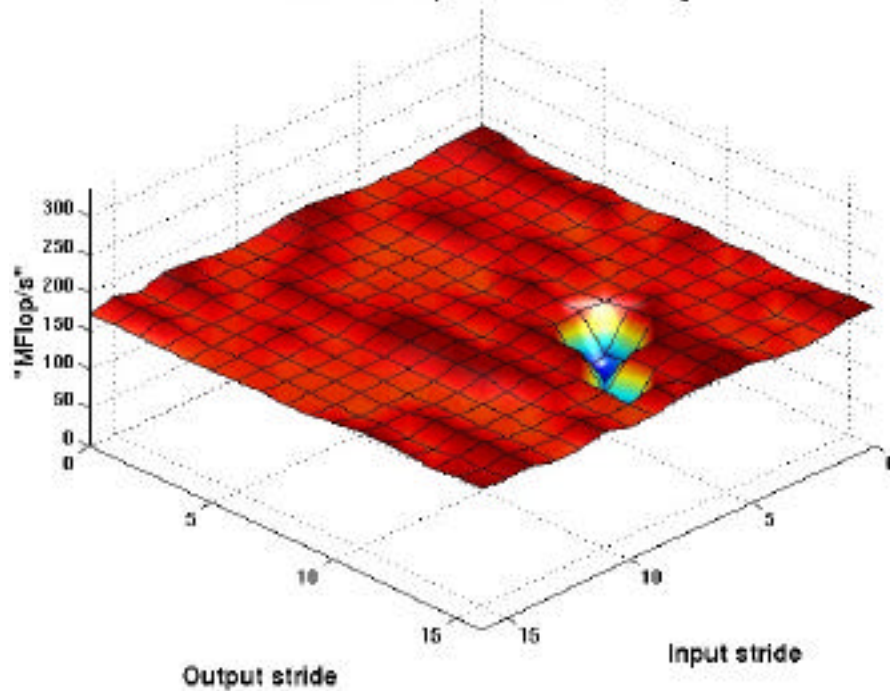
Libraries research at the University of Houston has focused on the UHFFT library for Fast Fourier Transformations. In the GrADS context, this work has focused on research in the architecture of high-performance scientific library software consistent with the GrADS software architecture of separating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

The UHFFT performance has been compared with other FFT libraries and is very competitive with the best-known public domain libraries and even some vendor libraries. The benchmarking of the UHFFT within the context of the GrADS project isn't made just to demonstrate competitive performance, but more so to understand the performance characteristics and eventually being able to model it in a form amenable for executable and composable performance models. An important aspect of most applications is access stride information though in most applications that is implicit and a function of the problem size. In an adaptive software package, such as the UHFFT, this information is important not only for assessing expected run time, but also for making choices of what building blocks to choose and how to compose them. The significance of this issue is illustrated in the figures on the following pages in which the execution rate is plotted as a function of the input and output strides for the Itanium2 and the UltraSparc III architecture. Though the difference in clock speeds is only 20%, the difference in execution rates is significant even when cache size is sufficient. The effect of the smaller cache on the Sparc processor is also highly apparent (and significant). For the Opteron processor its higher clock frequency compared to the Itanium2 processor is showing a clear performance advantage for small output strides, but for larger strides the Opteron performance may be considerably worse. For a good selection of composable code blocks it is important to properly capture these differences and being able to predict the behavior of composed blocks with good accuracy. This is a key aspect of our current effort.

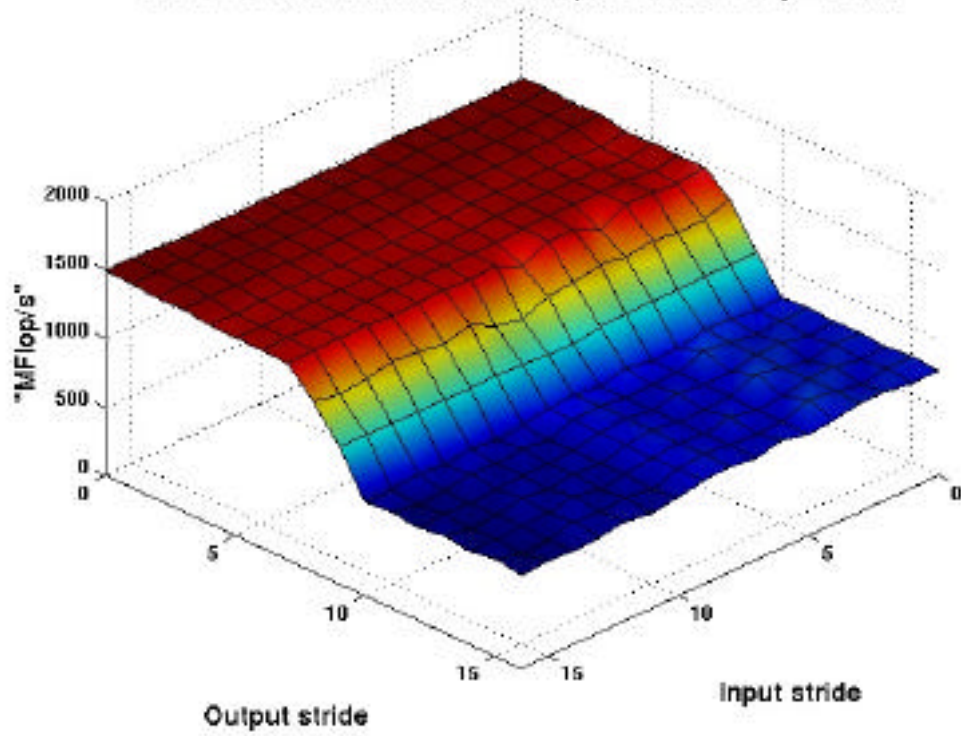
UHFFT Radix-2 Itanium 2 (McKinley) 900 MHz,  $P_{avg} = 332.5$



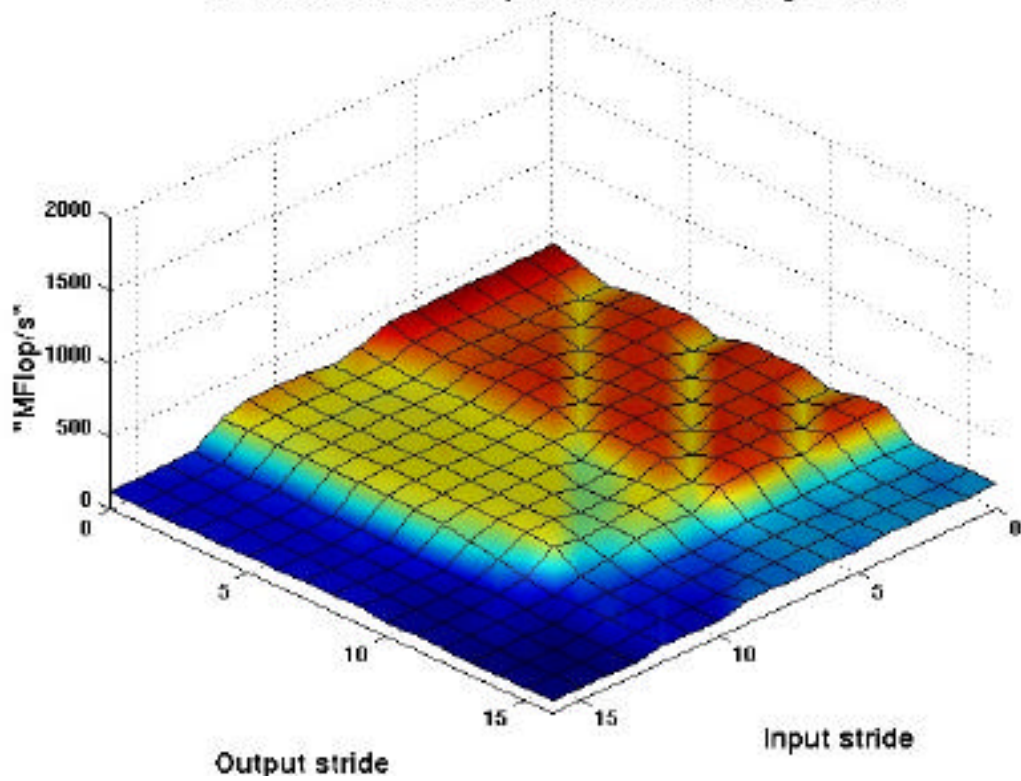
UHFFT Radix-2 UltraSparc III 750 MHz,  $P_{avg} = 175.1$



UHFFT Radix-64 Itanium 2 (McKinley) 900 MHz,  $P_{avg} = 1122.8$



UHFFT Radix-64 UltraSparc III 750 MHz,  $P_{avg} = 229.0$





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. We plan to incorporate these models in a form suitable for use in GrADSoft, and to distribute them in the same way that models for ScaLAPACK are for general use.

The UHFFT library is built automatically making extensive use of a code generator. Though the code generator has properly supported generation of code for the most common algorithms (mixed radix, split radix, Rader, and Prime factor) for complex-to-complex transforms it has been revised so it now fully support these functions also for transforms on real data.

### **2.5.2 ScaLAPACK Library**

University of Tennessee has been building a library of performance models of the ScaLAPACK routines for distribution to the general users. The library is based on the experiences in building the GrADSoft architecture and uses the data structures developed in GrADSoft. The performance model library is mainly intended for the users of ScaLAPACK who want to determine the near-optimal system configuration and input parameters for problem solving. The users of ScaLAPACK normally do not possess the adequate information for determining the appropriate set of end resources in the programming environment to solve ScaLAPACK problems of different problem sizes. By utilizing the GrADS scheduling technologies, the performance model library helps determine the optimal set of resources for problem solving.

The user of the library first calls a function for initializing different parameters of the entire set of resources. These parameters include the CPU load of the machines, the memory capacity, the network information between the machines etc. The user then calls the function `getExecutionCost`.

e.g.: `getEexecutionCost("dgesv", M, N, indices, &cost)`

In the above example, the user determines the execution cost of the ScaLAPACK `dgesv` problem for a matrix of `M` rows and `N` columns. The "indices" point to the candidate set of resources for which the execution cost of the problem is to be determined. In order to help the user to pass only select set of candidate resources to the function, the performance model library also provides functions for determining the properties of the different resources.

By utilizing the GrADSoft architecture, the performance model library provides the option of providing resource information either statically through a cache file or dynamically from NWS. We also plan to provide mechanisms for reading the resource information in other standard formats including the execution trace files available at the supercomputing centers and also the trace files used in other simulation environments including SimGrid, GridSim etc.



### **3 Applications**

From the beginning, GrADS has used sample applications from a variety of sources as motivations for our research. These driving applications ensure that our work is relevant to realistic systems, provide important benchmarks for our performance, and help us establish partnerships with other research groups. They also form the basis for most public demonstrations of our work, as at the annual SC'XY conference. We report here on six applications: ScaLAPACK, Cactus, FASTA, GrADSAT, EMAN, and two approaches to NetSolve. Of these, ScaLAPACK and Cactus have been extensively covered in past annual reports; FASTA and GrADSAT were developed in 2002; and the others began serious development in calendar year 2003. ScaLAPACK, Cactus, FASTA, and GrADSAT were demonstrated at SC 2002. We expect that all will be available for demonstration at SC 2003.

#### **3.1 ScaLAPACK (Linear Systems)**

ScaLAPACK was used as the first application to guide the development of GrADSoft architecture. The University of Tennessee helped to build interfaces for the ScaLAPACK developers to integrate performance models of the ScaLAPACK routines into the GrADSoft infrastructure. Performance model templates are provided to the library writers that can be filled for specific applications. A generic testing routine was also provided to enable the library writer to test the integrity of the performance model of his application. The performance model routines are then compiled into a dynamic performance model library that forms an integral component of Configurable Object Program (COP). The location of the dynamic library is specified by the GrADSoft user and loaded dynamically during runtime. The performance model routines are then initialized with the problem and resource properties and are used in the GrADSoft scheduler routines.

University of Tennessee, through the ScaLAPACK application, also helped in the development of other important components of the GrADSoft application manager, namely, the binder and the launcher. For binding, where the ScaLAPACK binary is dynamically instrumented, University of Tennessee helped identify the locations in the binary for dynamic instrumentation by manual instrumentation of the ScaLAPACK application. For application launching, a ScaLAPACK wrapper routine was written to read the problem parameters from configuration files staged to the end resources for problem solving. In a truly collaborative effort, ScaLAPACK was successfully integrated into the GrADSoft framework and demonstrated at the SC 2002 conference.

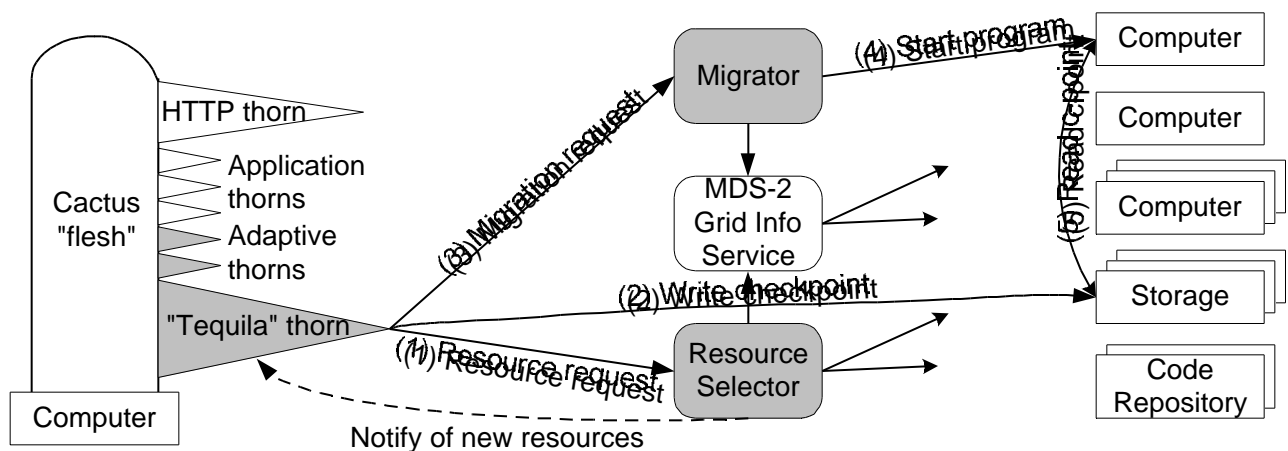
#### **3.2 Cactus (Partial Differential Equations and Astrophysics)**

As described in previous reports, GrADS researchers at the University of Chicago 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.

Previously, the GrADS project and Cactus code team collaborated to produce a Grid-enabled version of the powerful Cactus framework for the construction of parallel solvers for partial differential equations. Use in a Grid environment required Cactus to incorporate new modules for dynamic data distribution, latency-tolerant communication algorithms, and automatic detection of and adaptation to application slowdown. GrADS accomplished these goals by leveraging services provided by the Globus Toolkit for security, resource discovery, and resource access, and also providing new Resource Locator and Migrator services. These features contributed to Cactus winning the prestigious Gordon Bell prize in 2001 for an astrophysics computation. They also formed the basis for successful demonstrations at SC 2002.

The structure of the Cactus application is shown in the figure below. The “Tequila” thorn essentially implements the functions of the GrADS application manager – detecting available resources, computing the mapping of application processes to those resources, and monitoring the execution. In particular, this thorn communicates with the GrADS contract monitor to detect the need for remapping the computation. The “Resource Scheduler” seeks to identify suitable alternative resources from among those discovered and characterized by the selector, as does the standard GrADS scheduler. We define an application-Resource Selector communication protocol in the Set Extended ClassAds language. The “Migrator” implements the returned schedule, in the same spirit as the regular GrADS binder.



**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**

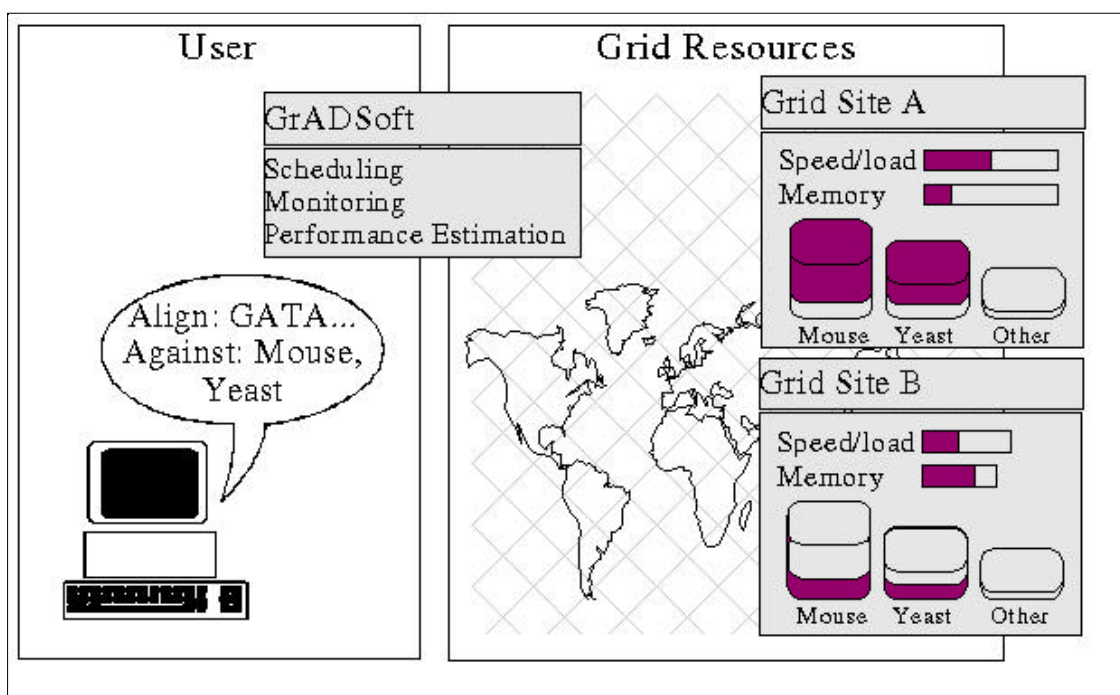
**migration, and the Resource Selector and Migrator services. Resources are discovered, monitored, and accessed using protocols and libraries provided via the Globus Toolkit. The steps involved in a migration operation are shown, numbered according to the order in which they are performed.**

In past years we demonstrated successful contract violation detection, resource selection, and migration and studied the effectiveness of the procedures and have optimization them. More recent work has extended the intelligent migration using computation and migration performance models. We also replaced the simple contract violation detection with sensors that fed into Pablo, coding the computation migration model into the COP structure, and integrating Cactus into the GrADS Application Manager. These advances were demonstrated on the exhibits floor at SC 2002.

### **3.3 FASTA (Sequence Matching)**

Protein and genome sequence matching is one of the basic operations in bioinformatics. Query sequences are compared against reference sequence databases, to find the most similar sequence and to generate optimal alignments. The similarity and alignment problem is similar to problem of calculating the edit distance between two strings and can be completely solved using dynamic programming techniques. However, these techniques tend to be computationally too expensive, especially given the current size of the reference databases, and the rate at which they are growing. Several heuristic techniques have been developed to speed up the similarity comparison and FASTA is one of the best known. FASTA is a protein and genome sequence alignment application, developed and implemented by William Pearson of the University of Virginia. The original parallel implementation of FASTA is a master-worker MPI code, where the reference databases are distributed by the master to all the workers. The query sequences are then sent to each worker, who computes the similarities with its proportion of the reference data, and return the result to the master.

At the University of Tennessee, we have been working on using FASTA as a test application for GrADSoft, to guide its development for large database applications and to demonstrate the use for biological applications. We adapted the original FASTA code so that the databases are maintained and used locally at the workers, without requiring full replication. The GrADSoft framework obtains database locality information from the GrADS Information Service, and uses this along with the current system information to schedule the FASTA application on a near-optimal set of Grid nodes. This application is an example of large data applications where we do not wish to move the database over a wide area network, so computation is scheduled at the location of the data.



**A pictorial overview of FASTA on the Grid; system status and database locality information are used in making scheduling decisions.**

A Performance Model was developed for the FASTA application based on experimental runs, so the predictions of the performance model are only expected to be accurate within the parameters of those runs. However, for scheduling purposes, all that the model needs to do is to distinguish between two execution schedules that are presented to it, giving a faster execution time to the better of the two schedules. Based on the Performance Model, a Mapper was developed that consists of a simple linear approximation that can be used by a linear programming approach to assign work to the hosts. The Mapper tries to assign work to hosts in order to balance and minimize the computation time, however, since the databases need not be replicated at all hosts, the workload can end up being uneven over the Grid nodes.

In creating a schedule, the GrADSoft scheduler provides a list of hosts and their associated system status (load, connectivity, etc) to the FASTA Performance Model. The GrADS Information Service was extended to allow the grid resources to be filtered by desired software requirement (e.g., the required databases). The Performance Model calls the Mapper, which uses the linear programming approximation to assign work to the hosts based on the current system status. The Performance Model takes the load assignment and computes a more accurate estimate of the execution time. The GrADSoft scheduler can use this estimate in selecting a near-optimal schedule. GrADSoft provides a simple framework that hides the details of scheduling and executing the FASTA application on a distributed Grid consisting of varying resources.

This application was demonstrated at Supercomputing 2002, running on the GrADS MacroGrid using resources at UTK, UIUC, and UCSD. This work was also presented as a

talk entitled “Sequence Alignment on the Computational Grid “ at the 2003 UT-ORNL Bioinformatics Summit, March 28 2003.

### 3.4 GrADSAT (Propositional Satisfiability)

Propositional satisfiability (SAT) is an important problem in computer science from a theoretical perspective. It is also pivotal for a wide range of practical applications. Such applications include circuit design, FPGA layout, Artificial Intelligence and scheduling. SAT solvers represent a powerful tool for solving problem instances for these applications. In particular, SAT solvers are used in the verification of circuit designs resulting in accelerated development of new circuits. Because of the importance of SAT results to the engineering community, a suite of benchmark problems (some with known solutions, some not) has been developed to test the efficacy of solver programs and an annual competition – most recently, the SAT 2002 Competition. (<http://www.satlive.org/satcompetition/>) – is held in which different solver implementations are compared head-to-head. Because of the inherent difficulty of the problem (it is, after all, the canonical NP-complete problem) and the specific complexities of available heuristics and solvers (including “learning” techniques for compiling partial solutions and unpredictability of partitioning the search space), parallel SAT solvers are particularly troublesome. They thus make a challenging driving application for GrADS.

In 2002, GrADS researchers at the University of California, Santa Barbara developed GrADSAT, a distributed SAT solver based on zChaff developed by Sharad Malik’s group at Princeton University. The zChaff solver is a “complete” solver; i.e. it is guaranteed to find an instance of satisfiability if the problem is satisfiable, or to terminate proving that the problem is unsatisfiable. The zChaff solver itself implements the Davis-Putname-Longemann-Loveland (DPLL) algorithm with learning, one of the fundamental SAT methods. GrADSAT has two important innovations that, together, result in an automatic SAT solver that outperforms the best previously known solvers as measured by the SAT2002 competition:

- **Scalable Distributed Learning:** GrADSAT includes an effective method for distributed learning and sharing of automatically deduced clauses amongst a large set of networked hosts.
- **Adaptive Resource Scheduling:** GrADSAT incorporates an adaptive scheduling methodology that enables high-performance SAT solutions using the GrADS testbed or other shared Grid resources that are widely dispersed geographically.

The latter result is a direct result of GrADS project support. The former result benefits from the GrADS infrastructure for its implementation, but was not a focus of GrADS research *per se*.

#### 3.4.1 GrADSAT Implementation

Like other DPLL-based solvers, GrADSAT is a backtracking search algorithm. At each search level, a decision heuristic picks a variable that does not have an assigned value, speculatively assigns it a value, propagates the value through all clauses that contain the variable, and checks for unsatisfiable clauses. If a conflict is found, the algorithm analyzes it to create a new “learned” clause (i.e. proof that a particular variable assignment is invalid and

need not be considered further), and the algorithm backtracks to the appropriate level. Eventually the algorithm terminates, either by producing a set of variable assignments where all clauses evaluate to *true* (if the problem is satisfiable), or by backtracking completely to the first decision level (if the problem is unsatisfiable).

GrADSAT uses a master-worker programming model to implement this backtracking. The execution starts at the master, which reads the problem file, manages the worker processes, and generates the final output results. The master uses the GrADS information server to rank possible target execution sites for workers according to processing power and memory capacity. It then starts execution on a single worker process on the highest-ranked execution site. The worker executes the search while it has sufficient resources.

While executing, a worker monitors its memory usage and the amount of time it has spent on a particular subproblem. If it uses more than a threshold percentage of memory even after deleting all unnecessary clauses, it is then risking running out of memory and thus will eventually stop. Therefore, its problem must be split, and the worker notifies the master. Upon receipt of a notification the master searches within the resource pool for the highest ranked idle resource. It also splits the worker's problem into independent subproblems based on complementary assignments of some free variables. A new worker is initiated initialized from the spawning worker's problem set. In order to alleviate memory usage inconsequential clauses are removed. The set of clauses in both subproblems include all clauses that do not evaluate to *true* because of the associated assignment stack.

Similarly, a time out period is given every worker. When this time period expires, the worker requests more resource from the master to help solve the current sub-problem on the assumption that a long running problem will continue to be a long running problem. The scheduler must attempt to balance the benefit of extra processing power against the expense of communicating the necessary state. Using the GrADSoft tools, the scheduler can determine how fast each machine is, how much memory is available, and the performance of the network connectivity between machines. It uses this information to determine which resource to acquire once a decision to split is made.

Learned clauses from a worker when shared with other workers can help prune a part of their search space. On the other hand, sharing clauses limits the kind of simplifications that can be made. GrADSAT merges new learned clauses from workers before choosing the next variable assignment at the topmost decision level. This allows for simpler implementation, and insures that clauses are merged in batches. The exact effect of sharing clauses is not yet known. In addition, when a large number of workers are sharing even a few clauses the total communication overhead becomes significant. Therefore GrADSAT workers only share "short" clauses in order to minimize communication cost.

### **3.4.2 Experimental Results**

The experimental results were obtained by running 42 test problems selected from the SAT2002 competition for SAT solvers, on the GrADS testbed. Our benchmarks included industrial, hand-made and randomly-generated instances, some of which were designated as

*challenging* (i.e. deemed hard by all solvers in the 2002 competition). The GrADS testbed was not dedicated to running GrADSAT, but rather was being used by various GrADS researchers at the time of the experiment. As such, other applications might have shared the computational resources with our application. If it had been possible to dedicate all of the GrADS resources to GrADSAT, we believe that the results would be better. As they are, they represent what is currently possible using non-dedicated Grids in a real-world compute setting; that is, they present a more realistic scenario for most users.

The results are presented in the table on the next page. The second column represents the instance solution. A question mark (?) means that the solution was unknown before we attempted to solve it with GrADSAT. The last column shows the maximum number of active clients during the execution of an instance. For all instances this number starts at one and varies during the run. The maximum it could reach is 34, the number of hosts in the testbed, but the scheduler may choose to use only a subset. Speedup is measured as the ratio of the fastest sequential execution time of zChaff (on the fastest dedicated machine available to us) to the time recorded by GrADSAT. We compared against zChaff both because our solver is based on it and because it was the overall winner of the SAT2002 competition. Entries above 1.0 mean that GrADSAT was faster than zChaff, while entries below 1.0 are slowdowns, typically due to excessive data sharing on small problems.

The problem instances are split into three categories. The first category is the set of instances that were solved by both zChaff and GrADSAT. The second category is the set of files that GrADSAT was able to solve while zChaff either timed-out or ran out of memory in our test. (The time-out was set at 12000 seconds for both programs in this test.) Only three out of the ten files in this category were solved by any other solver during the SAT2002 competition, where the time-out was set at 6 hours (21600 seconds), and three others were part of the challenging benchmark for which results were originally unknown. The other four had known answers, but no automatic generalized solver had been able to correctly generate them. The final set of input files are the SAT problems that were solved by neither GrADSAT nor zChaff. These problems might be solvable by either program with more resources or longer time.

File name	SAT/UNSAT /UNKNOWN	zChaff (sec)	GrADSAT (sec)	Speed- Up	Max # of clients
<b>Problems solved by both zChaff and GrADSAT</b>					
6pipe.cnf	UNSAT	6322	4877	1.23	34
avg-checker-5-34.cnf	UNSAT	1222	1107	1.10	9
bart15.cnf	SAT	5507	673	8.18	34
cache_05.cnf	SAT	1730	1565	1.11	34
cnt09.cnf	SAT	3651	1610	2.27	12
dp12s12.cnf	SAT	10587	532	19.90	8
homer11.cnf	UNSAT	2545	1794	1.42	10
homer12.cnf	UNSAT	14250	4400	3.24	33
ip38.cnf	UNSAT	4794	1278	3.75	11
rand_net50-60-5.cnf	UNSAT	16242	1725	9.42	20
vda_gr_rcs_w8.cnf	SAT	1427	681	2.10	15
w08_15.cnf	SAT	14449	1906	7.58	34
w10_75.cnf	SAT	506	252	2.01	2
Urquhart-s3-b1.cnf	UNSAT	529	526	1.01	4
ezfact48_5.cnf	UNSAT	127	196	0.65	1
glassy-sat-sel_N210_n.cnf	SAT	7	68	0.10	1
grid_10_20.cnf	UNSAT	967	3165	0.31	12
hanoi5.cnf	SAT	2961	1852	1.60	33
hanoi6_fast.cnf	SAT	1116	831	1.34	4
lisa20_1_a.cnf	SAT	181	243	0.75	2
lisa21_3_a.cnf	SAT	1792	337	5.32	4
pyhala-braun-sat-30-4-02.cnf	SAT	18	84	0.21	1
qg2-8.cnf	SAT	180	224	0.80	2
<b>Problems solved by GrADSAT only</b>					
7pipe_bug.cnf	SAT	TIME OUT	5058	-	34
dp10u09.cnf	UNSAT	TIME OUT	2566	-	26
rand_net40-60-10.cnf	UNSAT	TIME OUT	1690	-	30
f2clk_40.cnf	UNSAT(?)	TIME OUT	3304	-	23
Mat26.cnf	UNSAT	MEM OUT	1886	-	21
7pipe.cnf	UNSAT	MEM OUT	6673	-	34
comb2.cnf	UNSAT(?)	MEM OUT	9951	-	34
pyhala-braun-unsat-40-4-01.cnf	UNSAT	MEM OUT	2425	-	34
pyhala-braun-unsat-40-4-02.cnf	UNSAT	MEM OUT	2564	-	34
w08_15.cnf	SAT(?)	MEM OUT	3141	-	34
<b>Remaining problems</b>					
comb1.cnf	(?)	TIME OUT	TIME OUT	-	34
par32-1-c.cnf	SAT	TIME OUT	TIME OUT	-	34
rand_net70-25-5.cnf	UNSAT	TIME OUT	TIME OUT	-	34
sha1.cnf	SAT	TIME OUT	TIME OUT	-	34
3bitadd_31.cnf	UNSAT	TIME OUT	TIME OUT	-	34
cnt10.cnf	SAT	TIME OUT	TIME OUT	-	34
glassybp-v399-s499089820.cnf	SAT	TIME OUT	TIME OUT	-	34
hgen3-v300-s1766565160.cnf	(?)	TIME OUT	TIME OUT	-	34
hanoi6.cnf	SAT	TIME OUT	TIME OUT	-	34

#### GrADSAT and zChaff SAT2002 Benchmark Results on GrADS testbed



### 3.5 EMAN (Electron Microscopy)

We have decided to take on a new application for validation of the GrADS approach to computationally demanding applications. Electron microscopy is an important tool in the determination of 3-D structure of large macromolecular complexes (referred to as particles) made up of multiple molecular components with a total mass exceeding millions of Daltons. Electron cryomicroscopy is a promising methodology because it can reveal not only the folds of individual molecular components, but also their interactions responsible for the normal and abnormal states of the complex. So far this technology is applied successfully to virus pathogens with icosahedral symmetry. However, its potential applicability is immense for complexes with and without symmetry extracted through the Proteomics research in all branches of biology. Furthermore, the biologists are interested to solve the structures not only in one but also multiple functional states of the complex. Currently, a major bottleneck in this approach is computational in data processing, structure mining and sharing. Biochemical and electron cryomicroscopy improvements in combination with Grids should enable this advance in macromolecular structure determination as a high throughput and high accuracy technology for structural Proteomics research.

Data gathered from electron cryomicroscopes either through CCD cameras or digitized photographic films is the primary input for structure determination. Data collection typically produces ~250 image frames/day, giving ~10 GB of raw data per day per microscope. Thus, each microscope in a multiple-microscope facility will produce several terabytes per year. Data from the microscopes is deposited in a database for access by a suite of software tools designed to perform reconstruction and visualization of 3-D structures.

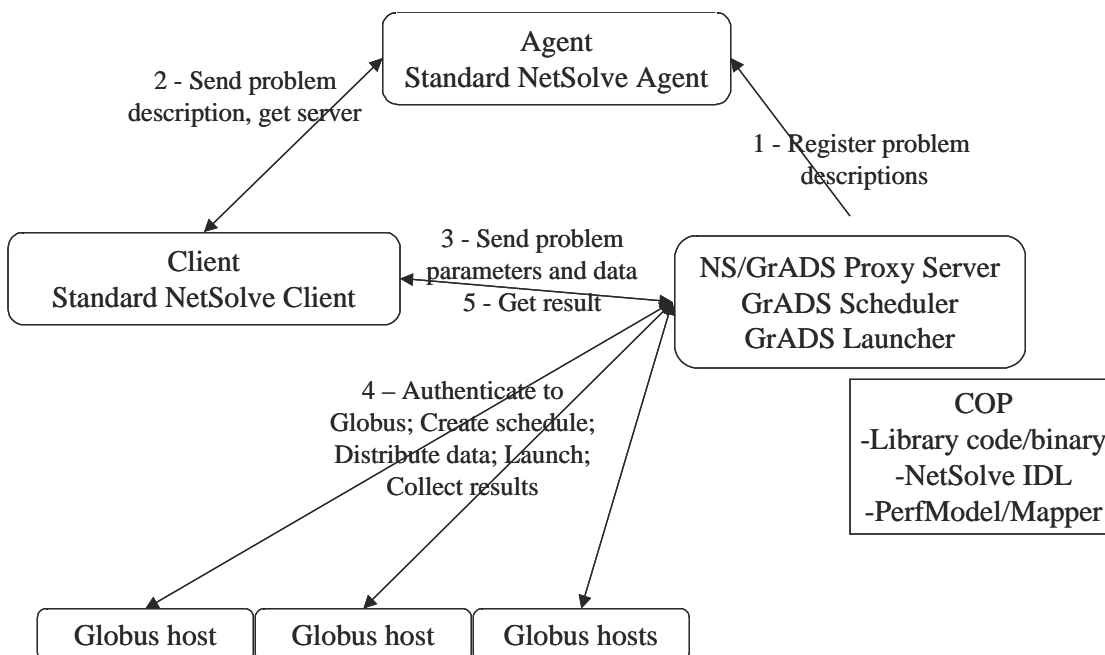
Data requirements for a given project can vary by several orders of magnitude depending on several variables including the size and symmetry of the particle and the desired target resolution. Typical projects will require from tens to tens of thousands of image frames, with 50-1000 particle images in each frame. Particles are represented as arrays of floats, currently ranging from ~128x128 to 1024x1024. Structure determination will require between 10,000 and millions of individual particle images, depending on the macromolecular complex and desired resolution. The final structure is represented as a 3-D density map the same size as the input image, i.e. - 1k x 1k particles will produce a 1k x 1k x 1k volumetric structure. Typical reconstructions today require ~1-10 petaflops of total computation, but due to the scaling of processing with particle size, large projects may easily require 10-1000x more processing power. The processing steps for structure determination are: localization of particle images within frames, power spectrum characterization on a per/frame basis, initial structure determination based on the selected particle images, and a complicated iterative refinement process. Multiple iterative cycles of image deconvolution and structure refinement is necessary, with the number of cycles varying depending on the structure and desired resolution.

We are currently using the EMAN package developed by Dr. Steven Ludtke at Dr. Wah Chiu's National Center for Macromolecular Imaging at Baylor College of Medicine

(<http://ncmi.bcm.tmc.edu/>). Unlike other single particle processing packages, EMAN was designed to automate as much of the single particle reconstruction process as possible. Once a set of boxed out particle images have been prepared, the entire reconstruction proceeds with almost no human intervention. The EMAN suite consists of GUI programs and high-level reconstruction routines both layered on top of a comprehensive C++ image processing library. EMAN also contains a number of tools for post-processing the reconstructed volumes, such as foldhunter for docking x-ray crystal structures into the final reconstruction, and helixhunter for localizing alpha-helices of protein within a subnanometer resolution structure. These tools also require substantial enough computational resources to warrant a grid approach.

### 3.6 NetSolve and GrADS Integration (Mathematical Libraries)

NetSolve is a project at University of Tennessee that provides numerical computation on remote resources. It is a RPC based client/agent/server system that allows one to remotely access both hardware and software components. It provides an easy-to-use interface to numerical libraries and front-ends exist in Matlab, Mathematica, C and Fortran. For example, using the Matlab front end to call a ScaLAPACK LU solver routine is as easy as calling “[x,z,info] = netsolve(‘pdgesv’,A,b)”. This frees the end user from a lot of the tedious work involved in setting up and using complex mathematical libraries.



**NetSolve and GrADSoft integration is shown with the information flow between the server, agent and client.**

At the University of Tennessee, we are working on an integration of NetSolve with GrADS. NetSolve has three distinct components that interact to provide library writers and users with a simple interface. Library writers add problems using an interface definition for a subroutine, which is compiled by NetSolve into a service. Servers register the problems that they can handle with an agent. A client takes a problem request and contacts an agent with

that request. The agent informs the client about servers that can fulfill that request. The client then contacts the server who handles the actual computation.

We have implemented a new NetSolve-GrADS proxy server (effectively the GrADS AppMgr with an additional service wrapper) that can take the NetSolve clients request and execute it using the GrADSoft framework. We have inherited a large portion of the NetSolve, which makes it easier to add additional mathematical libraries by using an interface language to define the calling sequences for additional routines. Minimal changes were required on the part of the NetSolve client. The protocol had to be enhanced to support Globus authentication and proxy transfer, so that the client could run on the remote Grid services under their correct Globus identity. The NetSolve agent remains unchanged from the standard distribution.

The initial version of a NetSolve-GrADS proxy server has been implemented, making it possible to use Matlab as front end to access the GrADSoft framework. Thus, using a Matlab call such as “[x,z,info] = netsolve(‘pdgesv’,A,b)” we can solve a matrix on the GrADS testbed via the GrADSoft infrastructure using the LU solve routine from the ScaLAPACK package. Other NetSolve interfaces, such as C and Fortran, are also supported.

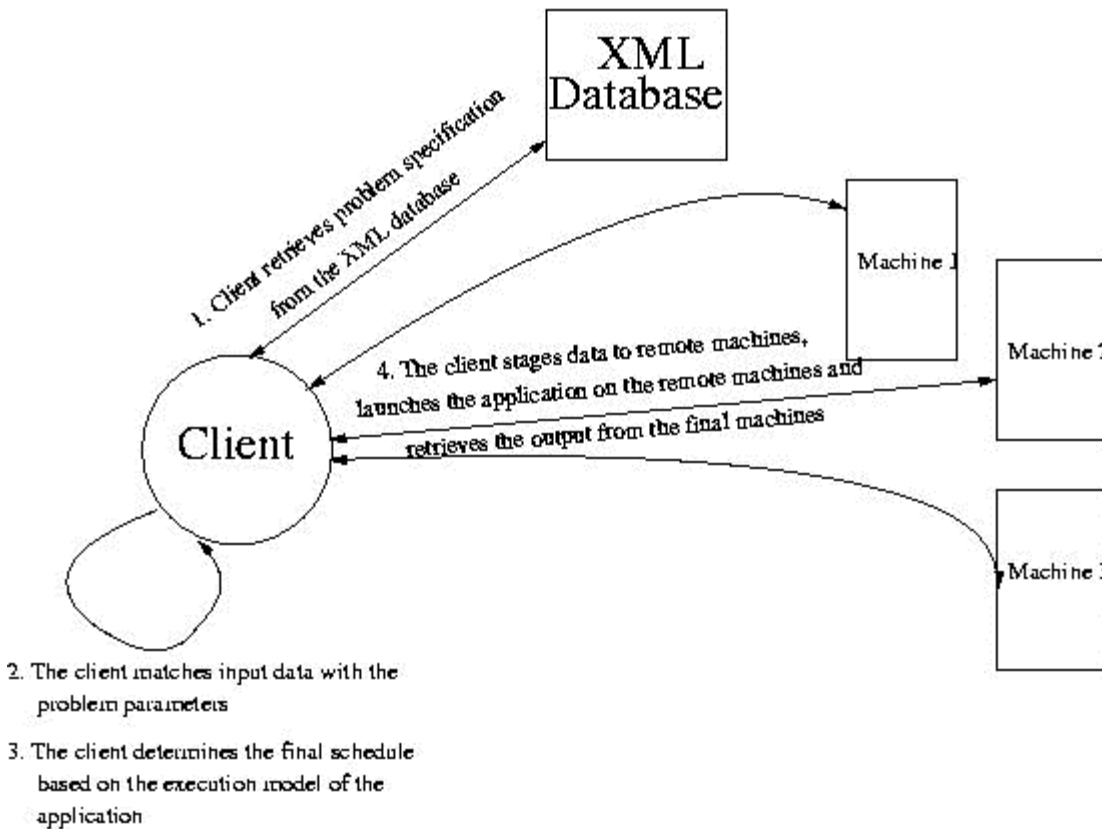
The current version of the NetSolve-GrADS proxy server copies the data from the client to the server and then to the compute nodes. We are currently examining techniques to handle the data staging from the client to the GrADS compute nodes more gracefully.

This integration provides GrADS with simple user interfaces and a possible route for distribution. In the current NetSolve distribution, parallel MPI jobs are run on fixed sets of machines, so there is no dynamic scheduling based on problem size or system status. This integration has enabled NetSolve to dynamically schedule parallel jobs on distributed Grid resources taking into account problem information as well the status of the system. Additionally, NetSolve has gained a Globus-based authentication and authorization system, which enables jobs to execute on remote resources under the identity of the end user.

### **3.7 GrADSolve (Mathematical Libraries)**

Another approach is also being followed at University of Tennessee to integrate the simple RPC mechanisms of NetSolve with the powerful scheduling strategies of GrADS. GrADSolve is a RPC system that supports execution of parallel applications over Grid resources. In GrADSolve, the resources used for the execution of parallel application are chosen dynamically based on the load characteristics of the machines. Also GrADSolve stages the user's data to the end resources based on the data distribution used by the end application. Finally, GrADSolve allows the users to store execution traces for problem solving and use the traces for subsequent solutions.

The figure below shows the general overview of GrADSolve.



### Overview of GrADSolve

At the core of the GrADSolve system is a XML database implemented with Apache Xindice. Since XML is mostly useful for storing metadata and transferring compatible documents across the network, GrADSolve uses XML as a language for storing information about different Grid entities. This database maintains four kinds of tables - *users*, *resources*, *applications* and *problems*. The *users* table contains information about the different users of the Grid system, namely the home directories of the users on different resources. The *resources* table contains information about the different machines in the Grid, namely the names of the machines, the clusters to which the machines belong, the architecture and the operating system in the machines, the peak performance of the machines etc. The *applications* table contains information about different applications, namely the name and owner of the application, if the application is sequential or parallel, the language in which the application is written, the number of input and output arguments, the data type and size of the arguments, the location of the binaries of the applications on each of the resources etc. Finally, the *problems* table maintains information about the individual problem runs due to the invocation of the remote applications by the end users. All the above-mentioned information is stored in the XML database in the form of XML documents. The Xindice implementation of the XML-RPC standard was used for storing and retrieving information to and from the XML database.

The library writer uploads his application into the Grid system specifying the problem description of the application using an Interface Definition Language (IDL). The GrADSolve

system creates a wrapper for the application, compiles the wrapper along with the application and transports the executable application to the different resources of the Grid system using the Globus GridFTP mechanisms. The library writer also has the option of adding an execution model for the application. The information regarding the locations of the end applications on the resources are stored in the Xindice XML database.

The end user writes a client program in C or Fortran to execute applications over the Grid. The GrADSolve client accesses the XML database, retrieves the problem specification for the application and matches the user's data with the parameters of the problem. The GrADSolve client also downloads the execution model of the application from a remote resource if the application possesses an execution model. For the resources that contain the application, the GrADSolve client retrieves the various performance characteristics including the peak performance of the resources, the load on the machines, the latency and the bandwidth of the networks between the machines and the free memory available on the machines from the Network Weather Service (NWS).

Based on the resource characteristics of the machines and the execution model of the application, the GrADSolve client determines an application-level schedule for application execution by employing scheduling heuristics. The application-level schedule consists of the list of Grid resources for the execution of end application. After determining the final application-level schedule, the GrADSolve client partitions the user's input data and stages the appropriate blocks of data to the different resources using the Globus GridFTP mechanisms. The client then spawns the application on the set of resources using MPICH-G. Similar to the staging of the input data, the client gathers the different blocks of output data from different resources using GridFTP and copies the data to the user's memory.

## 4 MacroGrid & Infrastructure

During the past year, there have been several significant research activities associated with the Macro Grid. A complete upgrade of all infrastructure software required the project to overlay parallel installations at different version levels on some of the Macro Grid resources. The management and administration of parallel Grid infrastructures enabled new tools to be integrated and tested using the new installation while the old installation supported application-driven experiments. By carefully coordinating the co-location of Grid software versions, we have been able to expand the testbed from approximately 50 included machines to approximately 130 without interrupting the research that uses them. Similarly, the Grid software infrastructure has been upgraded through two versions during the same period.

The primary goal of the MacroGrid continues to be 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 comprises of 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 flavors of Linux, AIX, IRIX, Solaris etc along with the operation of a Clearinghouse <http://www.isi.edu/grads> providing information and details on available resources. As stated earlier, we have transparently upgraded the services of the MacroGrid to include over 130 computing units, this year. Due to changes in the portions of the underlying grid software, substantial effort was put into porting the GrADSoft framework to run on the MacroGrid.

Main activities during this past year included:

- Continued operation of the MacroGrid and clearinghouse,
- Continued operation/enhancement of the GrADS VO servers,
- Transparent transition of the MacroGrid to be based on Globus Toolkit 2.2.4
- Transition to use MPICH-G2 (version 1.2.5.1a).
- Transition to use PAPI 2.3.3 and 2.3.4
- Enhancement of VO support and servers
- Enhancement of the schema about GrADS Information Providers
- Development of GrADS Info Provider deployed on a per-node basis
- Expanding the GrADS MacroGrid testbed
- Porting GrADSoft environment to an IA-64 environment

The online Clearinghouse at <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. A GrADS specific Virtual Organization server is being operated as part of the MacroGrid. This server, located at ISI provides 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 server to the current release of Monitoring and Detection Service (MDS 2.2). MDS 2.2 provides more stability, built-in security services (such as authentication), and a substantial enhancement in performance. In addition, we rewrote and deployed the new version of the custom information provider for GrADS based on extensions to Grid Information Services developed at UC Santa Barbara. We have also improved the VO-Grid interface, implementation and support for this data abstraction within GrADSoft.

During August and September of 2002, the University of Houston (UH) procured a 20 node IA-64 cluster with partial support (5%) from the GrADS project. The procured cluster is built with HP zx6000 dual processor, 900 MHz, Itanium2 processors using SCI technology for node interconnection. SCALI software is used for communication. An interprocessor communication bandwidth of up to 386 MB/sec and latency as low as 4.7 microseconds have been measured. The cluster was installed in a room previously used for student offices, and needed installation of both power and cooling given the rather high power consumption and heat generation of the Itanium2 workstations. Resolution of the cooling need is still to be completed.

There are currently 65 machines deployed on the GrADS MacroGrid at the University of Tennessee, Knoxville. GrADS researchers at UTK are in the process of deploying the Neo cluster (Sparc Architecture), the Boba Cluster (32 Dual P4 Xeons), and their HP Cluster (Itanium2) on the GrADS MacroGrid. This should be completed by mid-June. Once this deployment is complete, there will be more than 100 machines from UTK on the GrADS MacroGrid.

Prior to the introduction of the UH IA-64 cluster into the GrADS MacroGrid the MacroGrid had been entirely based on IA-32 processor architectures. The effort in bringing up the GrADSoft environment on the UH cluster did not pose any difficulties with the GrADSoft environment other than the “normal” issues with the underlying middleware and firewalls, since the UH cluster is behind a firewall. Other issues that had to be addressed in the porting were related to Open PBS that is used on the UH cluster for scheduling. UH researchers are currently in the process of installing a suite of performance tools, including HPCView, in support of their performance modeling effort within the GrADS project.

## 5 MicroGrid

In the past year, we have made significant progress in the following areas: 1) perform a range of application and middleware experiments as well as validation experiments using the Microgrid, 2) extend and enhance the network emulation system for greater scalability, 4) developing a library of Grid and network resource configurations to support experimentation, and 5) integrating and documenting the system sufficiently to enable an external software release of the Microgrid in February 2003. These efforts are summarized below, and are expected to enable a broad range of experimentation with MicroGrid and the addition of a new set of capabilities of the MicroGrid system in the upcoming year.

### 5.1 Extensive Application and Middleware Experiments

We used the MicroGrid to perform an extensive evaluation of the GrADS scheduler by emulating its behavior over a range of virtual Grid testbeds and application workloads. This improves significantly on the previous evaluation of this scheduler which involved only simple kernel applications and a single grid resource configuration. Using the capabilities of the MicroGrid to emulate a wide range of resource configurations, and the availability of several larger GrADS grid applications, we are able to perform a much more thorough evaluation of the GrADS scheduler. This more thorough evaluation produces new insights into the accuracy of the performance models, and the effectiveness of the scheduler for future Grid environments such as TeraGrid (DTF and ETF) and OptIPuter systems.

We evaluated the robustness of the scheduler in a range of different environments by performing the following experiments: (1) Run the scheduler and an application both on a real testbed and a virtual MicroGrid testbed. By compare the two running results, we show the effectiveness/validity of our method of using the MicroGrid. (2) Run the scheduler and different applications on a range of virtual Grids. By showing that the running time of the applications on scheduled resources, we test the robustness of the scheduler. Currently, the scheduler needs one performance model for each application to provide application appropriate scheduling. We use manually-built performance models by the GrADS system. As the GrADS system matures, we hope to obtain such components automatically.

We use the following five applications from the GrADS project for our experiments. These applications were integrated into the GrADSoft framework by the following efforts: ScaLAPACK [Petitet2001], Jacobi [Dail2003], Game of Life [Dail2003], Fish [Sievert2003], and the FASTA effort was led by Asim YarKhan and Jack Dongarra of University of Tennessee at Knoxville. These applications are either of significant interest to a scientific research community (ScaLAPACK and FASTA), or are representative of application classes of interest (Game of Life, Jacobi, and Fish). They also each provide non-trivial characteristics from a scheduling perspective. All are SPMD MPI applications.

We run these applications on a range of different grid resources environments, including a current real grid test bed (the GrADS MacroGrid), a real cluster, and MicroGrid models for the MacroGrid, the real cluster, the future TeraGrid, and the three future OptIPuter



configurations with latencies of 5ms, 60ms, and 10-40ms. For brevity, we omit detailed description of these systems.

### 5.1.1 Validating the MicroGrid Tools

To validate the MicroGrid emulation tools, we use all five applications; for each one we select a sample schedule and run the application with that schedule on both the real GrADS testbed and on the MicroGrid emulation of the testbed. Additionally, we perform the same tests for a cluster-based subset of the GrADS testbed (see Figures 8 and 9). The main observation is that the Microgrid emulated application execution time tracks real application performance in a cluster environment well, with the exception of the Game of Life application, which we will investigate further for the final version of this paper. The results obtained for the wide-area platform are however disappointing. We attribute them to the instability of the real platform

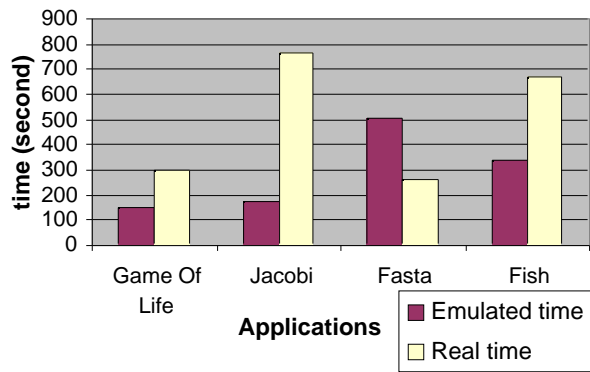


Figure 8. Wide-area experiments

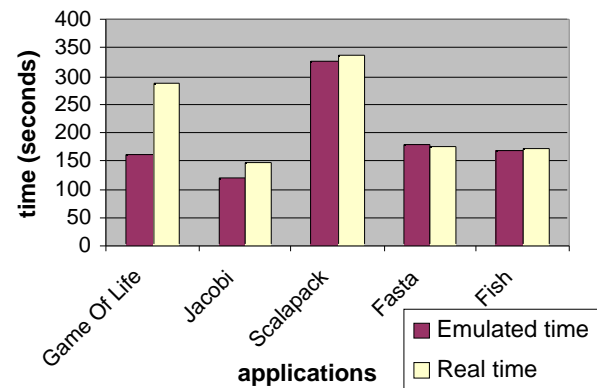


Figure 9. Cluster testbed

during the experiments, which the MicroGrid does not emulate (e.g fluctuating available bandwidth), and limitations of our TCP window size modeling. We are currently investigating these two issues.

### 5.1.2 Validating the Scheduler

We have performed validation the experiments on several virtual future Grid resources:

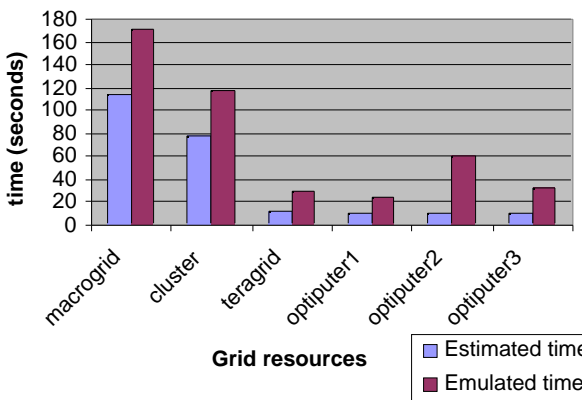


Figure 10. Jacobi runs on Virtual Grid:

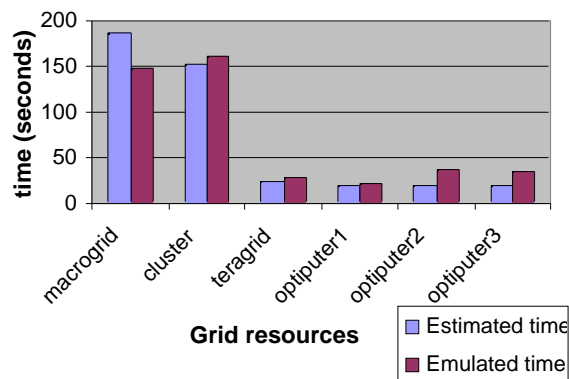


Figure 11. GameOfLife runs on virtual Grid:

TeraGrid, Optiputer, and a Grid with mixed resources on Optiputer and Internet-2.

The above two graphs show that the execution times estimated by the scheduler are tightly dependent on CPU resources and network bandwidth, while being less related to network delay. As showed in results, when the bandwidth is large enough (TeraGrid, Optiputer1, Optiputer2, and Optiputer3), the scheduler will give almost same estimation on running time. By contrast, the emulated results reflect the impact of network latency. In wide-area environment, the actual network bandwidth is tightly related to network latency [20]. Also in high bandwidth high latency area, small data transfer cannot benefit much from the large network bandwidth due to the slow start mechanism used for TCP window size. These results show an important limitation of the GrADS scheduler's performance models: they should model latency explicitly to be effective in high latency high bandwidth environments.

Figure 12 shows experiments for the FASTA application, for which the scheduler selects different number of nodes on different topology. In the three Optiputer topologies, we let the scheduler select from twelve nodes in each experiments. The twelve nodes include four nodes from each of the three clusters. An interesting fact is that it selects all the twelve nodes in Optiputer1, selects four nodes (from same cluster) in Optiputer2, and selects eight nodes in Optiputer3. These scheduling results are reasonable, because the clusters are close to each other in Optiputer1 (5 ms between each other), the clusters in Optiputer2 are far away from each other (60 ms), while in Optiputer3 two clusters are close to each other and the third cluster is from relatively further and slower Internet-2. But of course the scheduling for

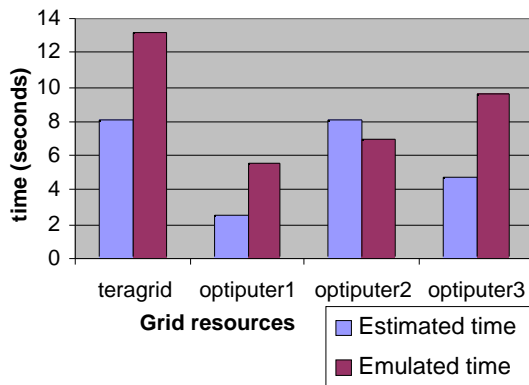


Figure 12. FASTA runs on Virtual Grids

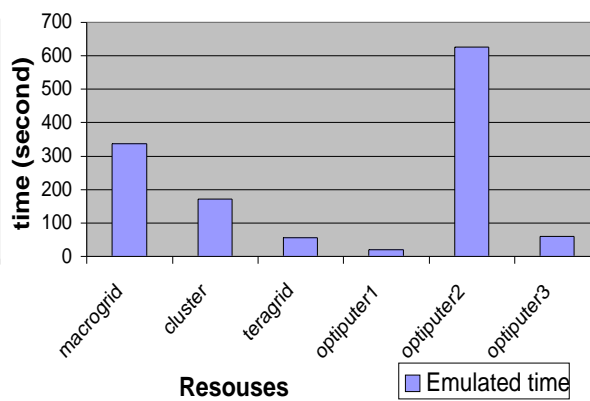


Figure 13. Fish on virtual Grids

Optiputer3 is not optimal, because the real case is that even four nodes (in Optiputer2) run faster than eight nodes. These results allow us to verify that the GrADS scheduler makes indeed sensible decisions.

Figure 13 shows experiments results of the Fish application. Fish does not come with a performance model. Instead, the resource requirement is provided to the scheduler using ClassADS resource description language and the scheduler will select a set of Grid resources which satisfy the resource requirement. The scheduler doesn't predict performance. The results showed above are from the emulation on following resources respectively: twelve MacroGrid nodes, eight nodes in UTK's cluster, ten nodes from five TeraGrid sites, and the Optiputer results are emulated on twelve nodes in three sites. The results show that network

latency is a major factor for performance, as the MacroGrid and Optiputer2 performance is dramatically worse than the low latency OptIPuter systems. Once again they show that if a performance model is to be developed for Fish it must include network latencies explicitly.

In the coming year, we will continue to broaden the range of experiments in the dimensions of range of grid resource configurations, number of resources, and number of applications. This will provide invaluable insight into the dynamics of grid resources, software, and applications.

## **5.2 Evaluate and Improve Scalability of the Network Emulation System**

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. By harnessing scalable compute resources, the MicroGrid system and applications together are an interesting parallel application for parallel and Grid resources. We consider a key problem for scaling such studies, the load balance of network emulation.

The load balance problem has received much attention in a wide range of parallel and distributed applications because good solutions are critical to achieving good speedups. For network emulation, it is a challenging problem because the networks studied have irregular structure, and the actual network traffic determines the required emulation work. As a result, each virtual network element (router, etc.) poses an unpredictable load. No previous network emulation projects have provided systematic techniques to achieve load balance (and therefore good scaling). The majority of these research projects provide no automated solution, depending on users to manually partition the network to achieve good performance. These approaches are not acceptable large scale network emulation involving thousands of network entities.

We have evaluated three approaches to network partitioning and mapping problem, using a range of static and dynamic information. Using a large-scale network emulation system we have built in MicroGrid, called MaSSF, we evaluate each of these three methods for partitioning: static topology information, combining topology and application static information, and combining topology and application profile data. These studies show that static topology and application information can achieve good load balance and profiling based algorithm further improves the achieved load balance for even larger scale network emulations. The specific results include:

- formulating the emulation load balance as a graph partition problem and the application of multi-object partition algorithms to solve it,
- developing a metric based on network topology and static application placement information to estimate total network traffic (application + background),
- develop and implement a scheme to profile network traffic efficiently in the emulator and then, use the profiled data to estimate total network traffic, and evaluating three approaches to network partition for load balance (topology, topology+application, profile-based), on several grid workloads and demonstrate that topology alone gives poor

load-balance, adding application information gives adequate load-balance, but adding profile information significantly further improves load balance.

### 5.2.1 Network Partition and Mapping Approaches

We explore three different approaches to define the objectives for the network mapping. These approaches exploit network topology, a combination of network topology and application traffic, and a profile-based approach which makes use of the simulation work (i.e. network traffic) from previous runs to predict future runs. Results from application of these approaches to two application workloads for a range of network topologies is included below.

#### 5.2.1.1 Network Topologies

We use three network topologies. The first two represent real networks, such as the TeraGrid (see <http://www.teragrid.org/>) and a section of a university campus network. To explore more complex network structures, our third network topology is created by a generic topology generator, which creates Internet-like topologies (the BRITE toolkits for network generation and also background traffic support).

Network Topology	Router	Host	Emulation Engine Node
Campus	20	40	4
TeraGrid	27	150	6
Brite	160	132	11

Table 1. Network Topology Setup

#### 5.2.1.2 Evaluation of Load Balance and Emulation Time

The emulation time of both applications is shown in Figures 6 and 7. For ScaLapack, the use of application-placement based partitioning improves overall emulation time significantly. Use of the profiling-based partitioning then further increased performance by about 10-20% (though it gets worse on Teragrid). For the GridNPB workload, the emulation time is nearly the same for all three partitioning algorithms. As we have mentioned before, the emulation time is not a directly measurement of the load imbalance, and because the execution time of GridNPB is compute dominated, not network dominated by the network communication, improvement of the emulator gives little overall runtime benefit.

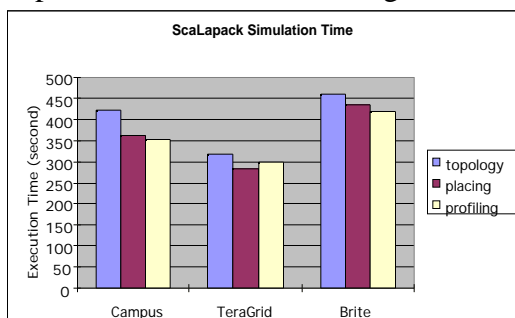


Figure 6 Emulation Time for ScaLapack

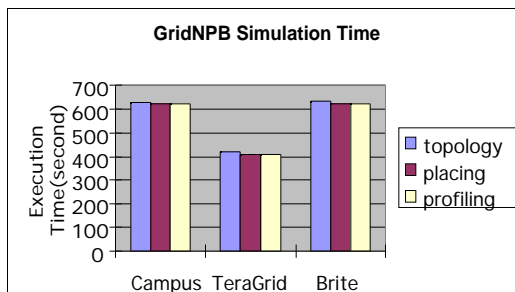


Figure 7 Emulation Time for GridNPB

To provide further insight, we show the fine-grained load imbalance of the Campus network emulation. We collect actual load simulation engine nodes in each time interval (2 second) and calculate the load imbalance for that period. As shown in Figure 8, the load imbalance of profiling-based algorithm is actually greatly improved compared to the topology-based partition algorithm, even the overall execution time is not significantly improved.

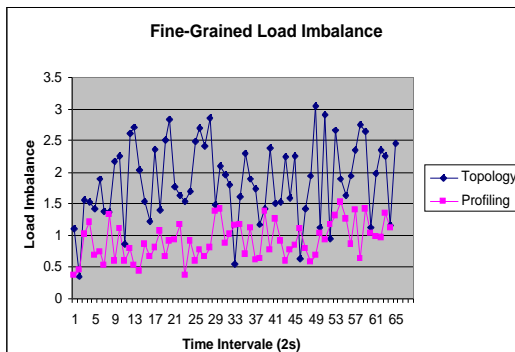


Figure 8 Fine-Grained Load Imbalance of GridNPB

### 5.3 Developing a library of Grid and Network Resource Configurations

As can be seen in the previous sections, we are developing a set of resource configurations which are interesting, realistic, and large scale. At this point, they include cluster (a simple LAN cluster of workstations), campus (a large campus network configuration with a backbone, several levels of routers, and a collection of LANs), GrADS MacroGrid (a model of the GrADS testbed which include several sides connected over Internet2), TeraGrid (a model of the NSF's emerging Distributed Terascale Facility and Extended Terascale Facility), and a range of OptIPuter configurations (which are based on high speed optical networks and a diversity of latencies).

In addition, we have enabled the MicroGrid system to accept topologies generated by an Internet topology-generator, Brite. We used this to generate one of the network configurations used in the network emulation scalability experiments, but it could also be used to generate a much larger collection of configurations.

### 5.4 Integration, test, and Documentation

We performed significant integration, test, and documentation in the reporting period. This enabled use to produce an alpha-release of the MicroGrid system (made generally available) in February 2003. We are currently revising the system based on input from external users, research extensions, and issues that have become clear to us in the intervening period.