# Toward a Framework for Preparing and Executing Adaptive Grid Programs

Ken Kennedy[α], Mark Mazina, John Mellor-Crummey, Keith Cooper, Linda Torczon
Rice University

Fran Berman, Andrew Chien, Holly Dail, Otto Sievert
University of California, San Diego

Dave Angulo, Ian Foster
University of Chicago

Dennis Gannon
Indiana University

Lennart Johnsson
University of Houston

Carl Kesselman
USC/Information Sciences Institute

Ruth Aydt, Daniel Reed
University of Illinois, Urbana-Champaign

Jack Dongarra, Sathish Vadhiyar
University of Tennessee

Rich Wolski
University of California, Santa Barbara

## Abstract

*This paper describes the program execution framework being developed by* the Grid Application Development Software (GrADS) Project. *The goal of this framework is to provide good resource allocation for Grid applications and to support adaptive reallocation if performance degrades because of changes in the availability of Grid resources. At the heart of this strategy is the notion of a* configurable object program, *which contains, in addition to application code, strategies for mapping the application to different collections of resources and a resource selection model that provides an estimate of the performance of the application on a specific collection of Grid resources. This model must be accurate enough to distinguish collections of resources that will deliver good performance from those that will not. The GrADS execution framework also provides a* contract monitoring *mechanism for interrupting and remapping an application execution when performance falls below acceptable levels.*

## Introduction

The recently-published volume *The Grid: Blueprint for a New Computing Infrastructure* [5] has established a compelling vision of a computational and information resource that will change the way that everyone, from scientist and engineer to business professional, teacher, and citizen uses computation [5,12]. Just as the Internet defines fundamental protocols that ensure uniform and quasi-ubiquitous access to communication, so the Grid will provide uniform access to computation, data, sensors, and other resources. Grid concepts are being pursued aggressively by many groups and are at the heart of major application projects and infrastructure deployment efforts, such as NASA's Information Power Grid (IPG) [7], the NSF PACI's National Technology Grid [12] and Distributed Terascale Facility, the NSF's Grid Physics Network, and the European Union's EU Data Grid and Eurogrid projects. These and many other groups recognize the tremendous potential of an infrastructure that allows one to conjoin disparate and powerful resources dynamically to meet user needs.

Despite the tremendous potential, enthusiasm, and commitment to the Grid paradigm, as well as the sophistication of the applications being discussed, the dynamic and complex nature of the Grid environment poses daunting challenges. Few software tools exist. Our understanding of algorithms and methods is extremely limited. Middleware exists, but its suitability for a broad class of applications remains unconfirmed. Impressive applications have been developed, but only by teams of specialists [3, 4, 5, 6, 8, 9, 11].

Entirely new approaches to software development and programming are required for Grid computing to become broadly accessible to ordinary scientists, engineers, and other problem solvers. In particular, it must be relatively easy to develop new Grid applications. Currently applications are developed atop existing software infrastructures, such as Globus, by developers who are experts on Grid software implementation. Although many useful applications have been produced this way, this approach requires a level of expertise that

[α] Corresponding author: ken@rice.edu

will make it difficult for Grid computing to achieve widespread acceptance.

The *Grid Application Development Software (GrADS) Project* was established with support from the NSF Next Generation Software Program to help address this challenge. In the GrADS vision, the end user should be able to specify applications in high-level, domain-specific problem-solving languages and expect these applications to seamlessly access the Grid to find required resources when needed. Using such environments, users would be free to concentrate on how to solve a problem rather than on how to map a solution onto available Grid resources.

To realize this vision we must solve two fundamental technical problems. First, we must understand how to build programming interfaces that insulate the end user from the underlying complexity of the Grid execution environment without sacrificing application execution efficiency. Second, we must provide an execution environment that automatically adapts the application to the dynamically-changing resources of the Grid. To address this second problem, the GrADS project has designed an execution framework for adaptive Grid applications. The goal of this paper is to elaborate the design of this framework and the motivation behind it.

## The GrADS Framework

Initial efforts within the GrADS project have demonstrated the complexity of writing applications for the Grid and managing their execution. To deal with this complexity, the GrADS project has adopted a strategy for program preparation and execution that revolves around the idea that a program must be *configurable* to run on the Grid. To be configurable in the sense intended by GrADS, a program must contain more than just code—it must also include a portable strategy for mapping the program onto distributed computing resources and a mechanism to evaluate how well that mapped program will run on a given set of resources. The notion of a *configurable object program* is thus at the heart of the GrADS execution framework. Later in this paper, we will discuss tools to help construct mapping strategies and performance models that are part of the configurable object program. For now, we will simply assume that these components exist in executable form.

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 becomes the Application Manager's responsibility.

The application launch and execution process is illustrated in Figure 1. We will step through this process discussing the role of each component in the execution launch sequence.

## Application Execution Scenario

A Grid user, or a problem solving environment (PSE) on behalf of the user, provides source code (which may be annotated with resource selection or run-time behavior information) or a handle to an existing IR Code object previously created for the user. This is given to a component called the *Builder*, which is the part of the program preparation system responsible for producing a configurable object program (COP). An overview of how the Builder accomplishes its task will be provided in a later section.

The Builder will construct any required objects and return a handle to a configurable object program, which includes the IR Code, the mapping strategy (or *Mapper)*, and the performance model, which we will refer to as the *Resource Selection Evaluator (RSE).* In addition, the Builder will provide a model of the resource space needed for execution of the application. This is called an *Application Abstract Resource and Topology (AART) Model*. An AART Model provides a structured method for encapsulation of application characteristics and requirements in an input-data-independent way. This information is in the form of a collection of descriptive and parametric resource characteristics along with a description of the topology connecting these resources. The purpose of the AART Model is to kick-start the resource selection process and to provide part of the information needed by the Mapper and the Resource Selection Evaluator.

Next, the user starts the Application Manager. This may be a standard GrADS Application Manager or a specialized manager designed by the user. The Application Manager needs the handle to the COP, I/O location information, the problem run size information (specifically, information to allow calculation of memory requirements), plus any desired resource selection criteria and other run-specific parameters desired or required.

The Application Manager retrieves the pieces of the COP. The AART Model is combined with the problem run information, resulting in the Resource Selection Seed Model. This produces the preliminary state necessary for the Mapper and the Resource Selection Evaluator to start being useful.

Once these components are available, the application manager invokes the *Scheduler/Resource*

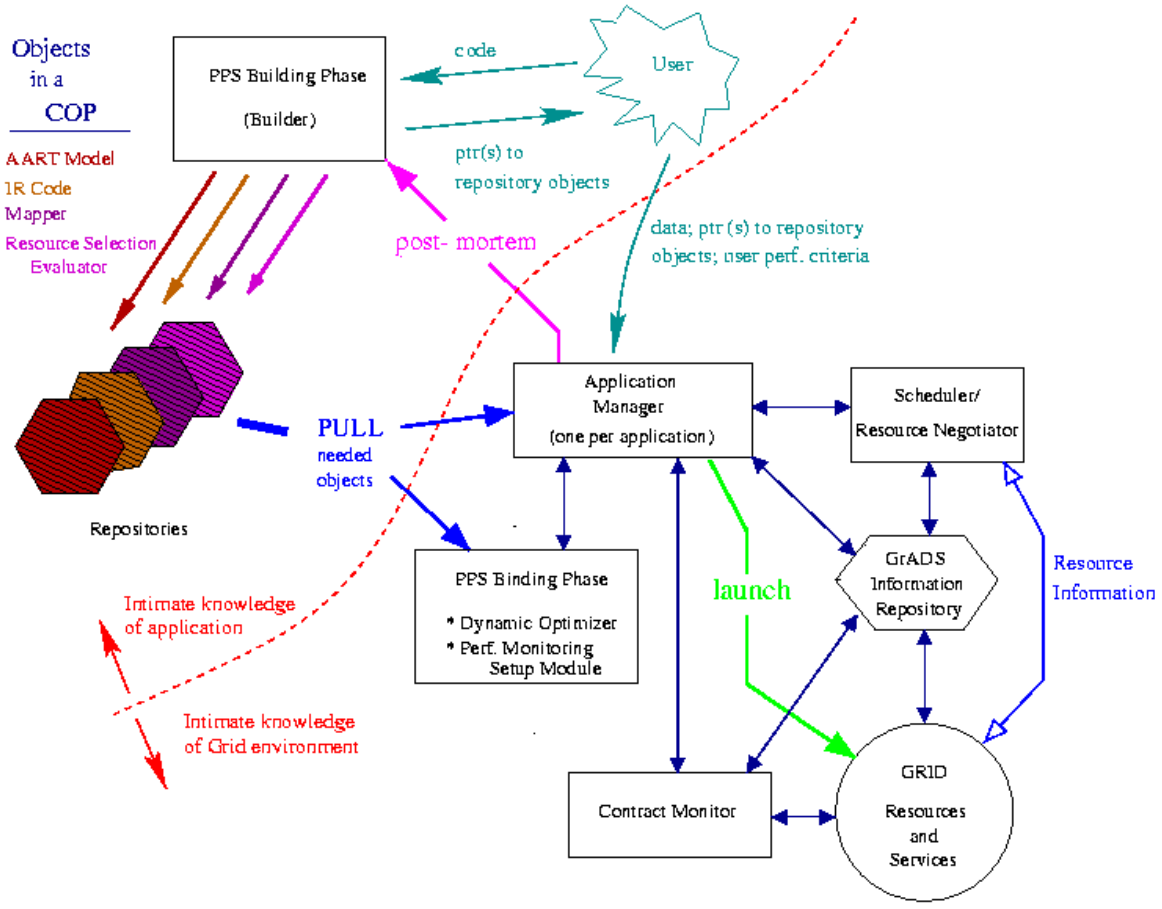


**Figure 1: The GrADS Application Launch and Execution Process**

*Negotiator (S/RN)* and provides it with the Resource Selection Seed Model. The Scheduler/Resource Negotiator is the component responsible for choosing Grid resources appropriate for a particular problem run based on that run's characteristics and organizing them into a proposed virtual machine. In GrADS, the S/RN is basically an optimization procedure that searches the space of acceptable resources looking for the best fit according to the application's needs as determined by using the Resource Selection Evaluator as an objective function.

The Scheduler/Resource Negotiator then invokes the Grid Information Service to determine the state of Grid resources and determine what resources are available that satisfy the characteristics required by the Resource Selection Seed Model. In other words, the Resource Selection Seed Model defines a feasible resource space for application execution. Once sets of feasible sets of resources are identified, they are organized into a collection according to the proposed Grid virtual machine. The Scheduler/Resource Negotiator then searches the collection of feasible sets of resources to find the one with the best performance on the given application, using the Resource Selection Model provided by the Application Manager as the objective function.

Once a collection of resources has been identified, the Application Manager begins the launch sequence. First, it stores state (basically a checkpoint) on the impending problem run (i.e. application + data) in the GrADS *Program Execution System (PES) Repository*, which is used to keep track of where each component of the application is executing and provide sufficient information to restart the application in the case of a catastrophic component failure. The Application Manager then invokes the *Program Preparation System (PPS)*

*Binding Phase*, passing it the COP handle, selected virtual machine, and the user's run-time information.

The PPS Binding Phase invokes the Mapper to perform the actual data layout and creates optimized binaries using a component called the Dynamic Optimizer, which performs tailoring of the program components to the specific computational resources on which they will run. The Binding Phase also inserts monitoring sensors needed by the performance-monitoring component of the execution environment, which is referred to as the *Contract Monitor*. The Contract Monitor is responsible for identifying egregious violations of the performance assumptions that led to the original resource mapping and initiation a reallocation of resources if necessary. The definition of what sensors are needed is provided by the Performance Monitoring Setup Module, which is invoked from within the PPS Binding Phase.

For some Grid-aware libraries, the PPS Binding Phase may need to arrange for dynamic linking to pre-built libraries for specific platforms. Handles to the optimized problem run binaries are passed back to the Application Manager, which again checkpoints its state to the GrADS PES Repository.

The Application Manager starts the Contract Monitor and then launches the binaries by invoking the GrADS Launcher, a service that is constructed on top of the Globus middleware layer. While the Contract Monitor is initializing, code inserted by the PPS in the application binaries may be positioning data on the resources making up the virtual machine.

As the code runs, the Contract Monitor gathers sensor data and uses the contract monitoring performance model(s) and violation thresholds provided by the Performance Monitoring Setup Module to determine if the application is delivering an acceptable level of performance. In addition, the Contract Monitor may try to make some determination of the cause of the poor performance. It reports its findings, together with summary monitoring information, to the Application Manager.

The evaluation of acceptable levels of performance and determination of the cause of violations is the shared responsibility of the Contract Monitor Component and the Application Manager, with the final decision to signal a violation coming under the domain of the Application Manager. The distribution of the decision making effort between the components will vary as appropriate for the given application structure, contract monitoring performance model granularity, and violation type.

Concurrently, the Contract Monitor output, as well as the original sensor output, can be archived for later use to refine models, adjust thresholds, or guide future executions. In addition, the application, Contract Monitor, and Application Manager may adjust the contract monitoring performance models and violation thresholds throughout the application lifetime in response to evolving application patterns and resource volatility.

If the Application Manager determines that the application is not making reasonable progress (or alternately, if the system becomes aware of more suitable execution resources), the *Rescheduler* is invoked. Using knowledge of the current execution, the Rescheduler determines the best course of action in order to improve progress. Examples of rescheduling actions are replacing particular resources, redistributing the application workload/tasks on the current resources, and adding or removing resources; or doing nothing (continuing execution with the current VM).

If the Rescheduler constructs a revised VM, the Application Manager builds new optimized executables, checkpoints the application, reconfigures and re-launches the application. The application reads in the checkpoint information and continues program execution. Once the application finishes, the Application Manager makes certain that the relevant collected performance data is fed back (i.e. archived) into the Program Preparation System and shuts down the Contract Monitor.

## Constructing Configurable Object Programs

Clearly, for this execution scenario to work, we must have a reasonable performance model and mapping strategies for each application. In fact, the performance model depends on a preliminary mapping provided by the mapping strategy, so these two components are intimately tied together. In our preliminary research [10], we discovered that performance models for non-homogeneous collections are extremely difficult for even sophisticated developers to construct.

As a result, we have adopted a strategy of providing within the program preparation system a collection of components and tools to assist in the development of the requisite performance models and mapping strategies. These tools will use three general strategies for constructing reasonably accurate performance models:

1. Expert knowledge about performance of components, particularly on different classes of homogeneous parallel processors.
2. Trial execution to determine run times of important components, with estimates of communications costs based on information from the Grid Information Service.
3. Integration of whole-application performance models from accurate models for individual components, based on the topology of the application.

The design and evaluation of these tools is a subject of ongoing research. However, our preliminary studies

indicate that there is strong promise that these three strategies can combine to provide enough accuracy to make the resource selection process effective [1,10].

## Project Status

Preliminary versions of the execution model described in this paper have been prototyped in the context of two demonstration applications: ScaLAPACK [10] and Cactus [1]. We are currently working toward an implementation that includes generic versions of these components that can be used with any configurable object program.

## Bibliography


1. G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *International Journal of High Performance Applications and Supercomputing* 15(4), Winter, 2001.
2. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Applications and Supercomputing* 15(4), Winter, 2001.
3. T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I–WAY: Wide--Area Visual Supercomputing. *The International Journal of Supercomputer Applications and High Performance Computing* 10(2):123–130, Summer/Fall 1996.
4. I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software Infrastructure for the I–WAY Metacomputing Experiment. To appear in *Concurrency: Practice & Experience*.
5. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1998.
6. E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed Computing in a Heterogenous Computing Environment. In *Proc. EuroPVMMPI'98*. 1998.
7. W. E. Johnston, D. Gannon, and B. Nitzberg. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In *Proceedings of the 8th IEEE Symposium on High-Performance Distributed Computing (HPDC)*, IEEE Computer Society Press, 1999.
8. T. Kimura and H. Takemiya. Local Area Metacomputing for Multidisciplinary Problems: A Case Study for Fluid/Structure Coupled Simulation. In *Proc. Intl. Conf. on Supercomputing*, pages 145–156. 1998.
9. P. Lyster, L. Bergman, P. Li, D. Stanfill, B. Crippe, R. Blom, C. Pardo, and D. Okaya. CASA Gigabit Supercomputing Network: CALCRUST Three–Dimensional Real–Time Multi–Dataset Rendering}. In Proceedings of Supercomputing '92, Minneapolis, Minnesota, November 1992 (Poster session).
10. A.Petitet, S.Blackford, J.Dongarra, B.Ellis, G.Fagg, K.Roche, and S.Vadhiyar. Numerical Libraries and the Grid: The GrADS Experiment with ScaLAPACK. *International Journal of High Performance Applications and Supercomputing* 15(4), Winter, 2001.
11. T. Sheehan, W. Shelton, T. Pratt, P. Papadopoulos, P. LoCascio, and T. Dunigan. Locally Self Consistent Multiple Scattering Method in a Geographically Distributed Linked MPP Environment. *Parallel Computing* 24, 1998.
12. R. Stevens, P. Woodward, T. DeFanti and C. Catlett. From the I–WAY to the National Technology Grid. *Communications of the ACM* 40(11): 50–60, November 1997.