# The Virginia Tech Computational Grid: A Research Agenda

Calvin J. Ribbens Dennis Kafura Amit Karnik Markus Lorch

Department of Computer Science
Virginia Polytechnic Institute & State University
{kafura,ribbens,akarnik,mlorch}@vt.edu

**Abstract**

An important goal of grid computing is to apply the rapidly expanding power of distributed computing resources to large-scale multidisciplinary scientific problem solving. Developing a usable computational grid for Virginia Tech is desirable from many perspectives. It leverages distinctive strengths of the university, can help meet the research computing needs of users with the highest demands, and will generate many challenging computer science research questions. By deploying a campus-wide grid and demonstrating its effectiveness for real applications, the Grid Computing Research Group hopes to gain valuable experience and contribute to the grid computing community. This report describes the needs and advantages which characterize the Virginia Tech context with respect to grid computing, and summarizes several current research projects which will meet those needs.

## 1 Introduction

A computational grid is a hardware and software infrastructure that provides convenient, efficient, secure, and reliable access to distributed high-end computational and data storage services [14]. The term *grid* is used by analogy with the electrical power grid; it is meant to suggest ubiquity and transparency, i.e., electrical power is available everywhere and the user does not need to know the source of that power. The hardware components of a computational grid are high-speed networks linking computational resources, e.g., supercomputers, clusters, symmetric multiprocessors, networks of workstations, disk farms, etc. The software components of a grid provide an integrated set of services that allow the available computational power to be accessed conveniently for large-scale computations. Grid services utilize the available computational resources so that tasks are run on whatever machine currently has available capacity. A grid also allows a single large computation to be spread across several machines, each of which is executing some portion of the computation. The essential services provided by grids include authentication and security, resource discovery, scheduling, data transfer, and fault tolerance.

Over the last few years interest in grid computing has grown tremendously. Federal funding agencies have supported the development of grid software, including Globus [18, 13] and Legion [29], the most widely used systems. Globus, in particular, is in use at several national laboratories and supercomputing centers as well as at many universities. The *Global Grid Forum* [17] represents a very active international community of researchers and practitioners working on grid technologies. Most of the large computer manufacturers have announced new programs in grid computing as well, e.g., IBM [25], Sun [44], HP [23].

The surge in interest in computational grids is motivated by two main trends—one from the demand side and one from the supply side. On the demand side, computational scientists

always want more powerful resources to tackle new grand challenge problems. Today, problems of national interest are characterized not only by the sheer magnitude of their computational demands, but often these problems require a multidisciplinary approach: multiple mathematical models, multiple length and time scales, multiple computer codes, multiple data sources, multiple investigators. On the other hand, the supply of available computing and network power continues to grow rapidly, especially when viewed in the aggregate. Today's desktop machines are significantly more powerful than a large mainframe of 15 years ago. However, the idea of a computational grid is to harness the computational power of an entire building full of computers—or, indeed, an entire campus, nation, or planet.

The principal challenge standing in the way of successful grid computing is the need for software infrastructure that makes a grid *usable,* in the broadest sense. The purpose of this report is to describe requirements for various types of grid users and stakeholders, and identify several specific research issues we are pursuing in the Grid Computing Research Group in response to these needs. Our focus is on a *campus-wide* grid. There are important specializations that can be made when a computational grid extends only across a single community or enterprise, rather than potentially extending over the whole planet. Hence, we turn first to a brief discussion of issues that are specific to our context at Virginia Tech. Following that, we discuss user scenarios and requirements in Section 3 and sketch several specific research issues in Section 4.

## 2 The Virginia Tech Context

Virginia Tech is extremely well positioned to exploit grid computing and to be a leader in research and development in this area. The university has significant strengths in the research areas that motivate and support grid computing. The primary motivation for grid computing is high-end computational science, a mode of investigation practiced by numerous research groups around campus. For example, at least eleven research centers on campus do large scale computational science and engineering work. These centers include the Virginia Bioinformatics Institute (VBI), the Interdisciplinary Center for Applied Mathematics (ICAM), the Laboratory for Advanced Scientific Computing and Application (LASCA), the Vibration and Acoustics Lab, the Multidisciplinary Analysis and Design Center for Advanced Vehicles, and the Center for Modeling and Simulation in Materials Science. Furthermore, the university has computer science expertise in the research subdisciplines that make grid computing possible, including networking and distributed computing, parallel computing, numerical modeling and scientific computing, software engineering, collaborative computing, human-computer interaction, etc.

Virginia Tech also has a growing collection of computing resources that could be usefully integrated by a grid system. Current resources include the 200 processor CS-LASCA cluster in Torgersen Hall, the 80 processor CS cluster in McBryde Hall, several other clusters in Torgersen and at VBI, numerous shared memory machines (symmetric multiprocessors or SMPs), hundreds of high-end workstations, and perhaps thousands of PCs in teaching classrooms, laboratories, and study areas. While not all applications run well on all resources, there are important problems which can leverage the massive amount of untapped computing power on campus; and there are other problems which can only be addressed if a substantial amount of this computing power can be used as a single, coordinated resource. (See Section 3 for a discussion of some typical user requirements.) Finally, in addition to computers, high-end grid computing also requires high-end networking. The networking infrastructure on campus, along with the expertise at Communications Network Services (CNS), is another clear strength which makes a VT computational grid an attractive proposition.

There are several factors that make the goal of establishing a campus grid especially interesting, both in terms of research and in terms of practical benefit to the community. Below, we list several points that distinguish a campus grid from the broader field of planet-wide grid computing.

- There are important security and authentication simplifications. For example, the local certification authority could be seamlessly integrated with mechanisms already in place, e.g., Hokie passport and VT pid. Another important example is that the integrity of firewalls that sit between campus grid resources and the outside world can be preserved, e.g., holes can selectively be opened in firewalls in such a way that the main purpose of the firewall is not compromised.
- We can assume better networking connectivity than in the global scale. Because VT controls its network, and because the scale of a campus grid is smaller, communications properties such as reliability, bandwidth, and latency are much more favorable in a campus grid setting than in a larger-scale grid. This has important ramifications for algorithms and middleware, as well as usability.
- A relatively small set of resources means that a campus-wide grid is more static than the planetary grid. By this we mean that the set of (live) resources and the connectivity amongst them does not change as rapidly as it does in an arbitrary wide area grid. Of course, by definition, grids are always dynamic to some extent; but important aspects of grid computing such as resource discovery and load balancing become much easier when the scale of the grid is only campus-wide.
- A relatively small set of users means that a campus-grid can be more responsive to the needs of those users. Even when fully deployed, a VT grid might only be of interest to 50 research groups (however, these research groups probably represent 90% of the computing appetite on campus). And the large-scale applications that these research groups use tend to fall into only a few broad categories (see discussion in Section 3). Therefore, our grid research and implementation can be more focused on making the grid usable for the VT community. In fact, an important goal of our recent research has been to enable grid users to tailor their own problem-specific computing environments for grid computing [40]. We believe this task becomes tractable in our setting.
- There are important legal and economic simplifications. The idea of a global grid raises many serious legal and economic questions, including liability and cost-recovery. While none of these issues vanish in our setting, we believe that there are important practical simplifications because, for example, all grid resources are VT property and are covered by VT terms of service.
- The psychological and sociological challenges to grid computing also have a good chance of being overcome in our context. This university prides itself in applying technology in research and education. The faculty and students are sophisticated and motivated when it comes to using technology well. The level of collaboration among research groups is high, and collaboration is rewarded.
- It seems likely that most grid computing will take place within a single organization or enterprise. Hence, it is not too restrictive to exploit the factors listed above. In fact, it is important to investigate exactly this situation.

Finally, in view of the resources available at Virginia Tech, and leveraging the unique characteristics of our setting, we anticipate the following specific benefits to the university from this research thrust:

- Achieves economic use of existing resources by utilizing otherwise unused computational power.
- Leverages investments in buying small clusters or SMPs by allowing those resources to be combined occasionally with other resources.
- Lowers start-up costs for new users who can gain access to a powerful ensemble with minimal investment.

- Preserves local administrative control since grid scheduling will respect the security and accounting controls defined by the system administrators of local resources.
- Encourages interdisciplinary work and cooperation through the pooling of computational resources without requiring detailed agreements on the nature and administration of local resources.
- Promotes access to national resources at federal research centers which use community-supported grid software, e.g., Argonne National Laboratory, Oak Ridge National Laboratory, Pittsburgh Supercomputing Center.

# 3 User Scenarios and Requirements

Just as the pioneers of the electric system could not have predicted the shape and function of today's power grid, it is difficult today to anticipate all the ways computational grids will be used in the future. However, an important high-level goal of this research is to learn by doing—to deploy a prototype grid and gain experience from real users as they solve problems on our grid. It is helpful to organize our efforts by classifying potential grid users into five broad categories. In this section we define these five user classes, along with a sixth class consisting of important stakeholders such as machine owners and administrators. For each class we discuss requirements likely to emerge from that group. These user classes have increasingly high demands of a grid, i.e., each class requires all of the things required by the previous classes, and more.

### Class 1: Simple applications

The simplest scenario is a user who is just looking for machine cycles. These applications require no special hardware and only simple file access for input and output. There are many projects that meet the needs of most of these users already. The Globus toolkit provides many of these services; others are rapidly becoming available in tools built on top of Globus. The most widely used system in the area of recovering unused cycles and job-migration is Condor [10, 30]. Requirements for this user class include the following:

- Initiating grid participation, authorization.
- Authentication for a session or job, single sign-on.
- Help with compiling for multiple platforms and with staging the application to these platforms. Grid-aware code development environments.
- Help with staging and retrieving data files.
- Convenient job submission, monitoring, management.
- Monitoring of the grid and the user's status with respect to the grid, i.e., resource availability, limits, and accounting.
- Security: user's codes and data cannot be compromised when on other machines.
- Fault tolerance: for long-running applications, support for check-pointing and recovery.
- Resource-brokering: matching job requirements to available resources.
- Allocation, scheduling, load-balancing: locate available machines, migrate jobs if machines that were once lightly loaded become heavily loaded.

### Class 2: Embarrassingly parallel applications

The typical user in this class wants to run many instances of the same task or application, with little or no communication between tasks. A standard example is parameter sweep or brute-force search algorithms, where the same code is run over and over again but with different

inputs. The goal is to find any solution, or perhaps a best solution, according to some measure, e.g., factoring large integers [43], searching for signs of extraterrestrial life [42], or optimizing the configuration of a drug molecule [5]. The master-worker paradigm is typically used to organize parallelism inherent in these applications. The Search for Extraterrestrial Intelligence (SETI@home) project [42] is a popular instance of an embarrassingly parallel application. Several companies have been founded recently to attempt to commercialize this approach to large scale computing, e.g., DataSynapse, Entropia, Parabon, Avaki. Additional requirements for this user class include the following:

- A higher-level user interface that controls the overall process. This could be anything from a scripting language to a graphical user interface (GUI). The interface needs to support definition of the cases to run, monitor progress, re-schedule if cases fail, and assimilate the results, e.g., compute the global minimum.
- More sophisticated scheduling algorithms. For example, the scheduler could assign tasks in batches of size greater than one, or could use performance results from previous runs to more efficiently schedule future runs.
- More sophisticated fault-tolerance schemes. A typical approach is for the master to reschedule a given worker task if the first attempt to complete that task fails to return an answer in some predicted time. Another approach simply schedules more than one instance of each task, using the first one that returns with an answer and ignoring any that return later.

### Class 3: Traditional parallel applications

This class consists of users with distributed-memory parallel applications, e.g., a code that uses the MPI message passing library [21]. Two scenarios can be imagined. A relatively simple case is that of a parallel code that runs on a single homogeneous parallel resource, e.g., a cluster, SMP, or network of workstations. This user wants to run many instances of the code, perhaps, and is simply looking for available cycles. This case is essentially a parallel version of Class 1. The second case is a parallel application so large that no single homogeneous resource suffices. We focus primarily on the second case here, although important research is done by investigators whose work falls into the first scenario as well. Requirements for the second case include the following:

- MPI over the grid, including job startup and termination. It must be possible to run an MPI job across heterogeneous resources, e.g., across two clusters. The first attempts at supporting MPI applications over the grid are just now emerging, namely MPICH-G2 [12] and PACX-MPI [4].
- Algorithms are extremely important for this class of users. Minimally, algorithms must be adaptive enough to work on different numbers of processors. Latency tolerant, coarse-grained algorithms are preferred. Resource-aware, resource-adaptive algorithms will be important as well. For many computations a choice of algorithm and/or implementation must be made for each type of resource, e.g., SMP, tightly-coupled cluster, loosely-coupled network of workstations. Using the wrong algorithm can have a significant negative impact on performance. There is a need for algorithm recommendation systems that can automatically choose the best algorithm as a function of where a particular application is sent on the grid.
- There is also a need for a resource reservation mechanism and for co-scheduling, i.e., users need to reserve and schedule multiple resources at once. This is particularly challenging in a grid environment where it may be necessary to marshal resources from multiple administrative domains, controlled by multiple local schedulers.

- Performance debugging, modeling, measurement and tuning is particularly important for large parallel jobs. Users require very sophisticated tools to support this kind of activity in the grid context.
- Fault-tolerance for Class 3 jobs is much more challenging than for previous classes. When hundreds of processors are committed to a single application, the odds that one of them will fail are good, especially if the application runs for days or weeks. Furthermore, the cost of restarting the entire computation is enormous. Ideally, one would like only the failed process(es) to be restarted, while the healthy processes continue computing. Unfortunately, current message passing software layers do not provide a convenient solution to these problems.

### Class 4: Heterogeneous parallel applications

These applications typically consist of several components linked together at a high level. A component might be a simulation code, a data source, a visualization component, etc. These are the kinds of applications that motivate the largest computational grid projects at US and european national laboratories. Well-known examples include global climate modeling and multidisciplinary design optimization for aicraft design. Several software frameworks are being developed to meet the needs of this class, including Harness [9], the Common Component Architecture (CCA) [3], and Babel [11]. Requirements include the following:

- Software infrastructure for compositional modeling. Model development tools for defining control-flow and data-flow, and for encapsulating legacy resources.
- Component frameworks, supporting plug-and-play and interoperability among libraries, solvers, components, etc.
- Representation schemes for sharing data among components. Different components may model and represent data differently.
- Data-management. Simpler classes need help with managing data too, but it is a major requirement for this class, since these applications almost always produce and consume vast quantities of data. Other critical issues include access to data, data locality, and interfacing with databases and pre- and post-simulation tools.

### Class 5: Problem solving communities

The emphasis here is not so much on the type of problem being solved, but on who is solving it. It is important to keep the real context in mind. Focusing too much on a single user running a single application is a mistake. Computational science is done by groups of scientists and engineers. No simulation is done in isolation—many are part of an ensemble of simulations, and every simulation builds on previous results and suggests future questions. This perspective suggests the following requirements when thinking about middleware and grid computing environments:

- Management of grid communities. It should be easy to add or remove users from particular grid communities, e.g., by controlling access to particular machines, codes, data sets. Convenient mechanisms for authentication, authorization, and delegation are needed here, including support for short-lived collaborations.
- Collaboration tools. Physically separated users must be able to communicate and share resources, both synchronously and asynchronously. This is a very broad category and an active research area in its own right, e.g., see [22, 2].
- Simulation management. This is another very broad category. Previous classes may need assistance here as well. Tools are needed to coordinate simulations done by a group of researchers, and to manage and exploit the results of previous simulations.

**Class 6: Other stakeholders**

This class includes machine owners, administrators, and maintainers of the grid infrastructure. Requirements include the following:

- Control. Owners must be able to determine when cycles are offered to the grid, and to whom.
- Management. Who bears the burden of adding a machine to the grid and taking it off again? Who manages the grid? What kinds of management are required?
- Accounting. Monitoring grid use, accountability, e.g., identifying the largest users.
- Security. Grid applications must not compromise machines or data.
- Psychological and economic issues. There must be some motivation for a machine owner to contribute a machine to the grid.

# 4 Research Issues and Approach

In this section we identify several current emphases in the Grid Computing Research Group at Virginia Tech. We focus on computer science research issues rather than on the computational science and engineering application areas that we support. Applications are crucial to our success, however. Collaboration with grid users is a critical component of each research thrust described in this section.

The field of grid computing is growing very rapidly. It is a significant challenge to keep up with the pace of change and to work strategically, identifying projects that fit well with the needs of Virginia Tech and with our own expertise and interests, and that leverage work being done elsewhere. Hence, this list of issues will change frequently.

## 4.1 Architecture for Grid Computing Environments

Until recently, most of the research and development in grid computing has been on relatively low-level tools and frameworks such as Globus. This bottom-up approach makes sense, since without the basic set of grid and data services provided by these tools, we would not be able to build higher-level tools and frameworks. Now however, with Globus rapidly becoming the de facto standard for low-level grid services, and with increasing stability in Globus itself, more attention is being paid to higher level issues. One of the research groups in the Global Grid Forum is focused on grid computing environments (GCEs) [19]. Emerging tools include frameworks [16, 28], toolkits [20, 45], and portals [26, 35].

Effective grid computing requires much more than raw computational power and low-level grid services. Usability concerns dictate that high-level tools and environments are needed to support large-scale computational science research. This recognition is fundamental to our research agenda, and it is consistent with the approach taken in our work in Problem Solving Environments (PSEs) [1, 38]. The PSE Research Group at Virginia Tech has for many years been studying how to design and build PSEs for multidisciplinary grid communities. The emphasis of this research has been on leveraging high-level problem-solving context to design architectures and tools that scientists and engineers can use to build their own computing environments.

The type of virtual organization [15] we are targeting is the computational science research group. In working with several such groups, we have identified three themes that are crucial to effectively supporting this community (see Ramakrishnan et al. [40] for details):

- *collaboration*: computational science is virtually always done in groups.
- *compositional modeling*: building large computational models from many parts.
- *context*: computational science tools and environments must be aware of the specific problem-solving context.

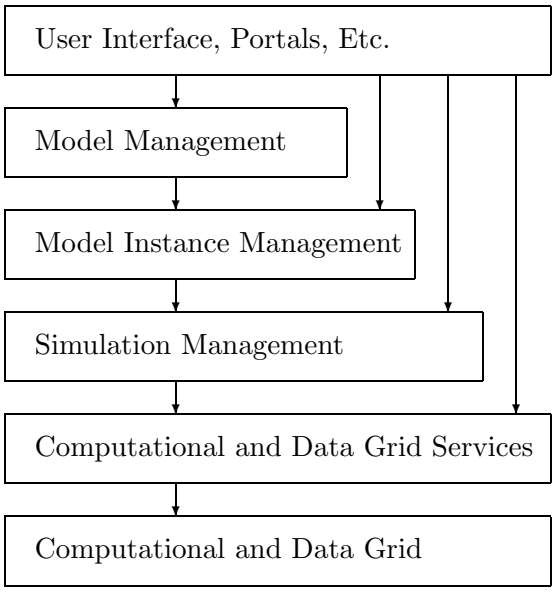


Figure 1: An architecture for data-centric computing environments for scientific grid communities. Adapted from Ramakrishnan et al. [40].

The third theme, exploiting context, is perhaps the most critical in the case of grid computing. In building grid computing environments, we believe there is a significant requirement for exploiting the larger context of the particular scientific application. There is also great opportunity in leveraging this context. The 'context' we refer to here should be understood in a very general sense; it is defined by the activities of the problem-solving community. Consider these simple examples: (1) a research group has been doing computational simulations for months, and a database of previous results is used to improve the efficiency of current simulations or to avoid computation if a desired result is already available; (2) a simulation is being run as part of a higher-level problem-solving strategy, e.g., in an optimization loop or under the guidance of a recommender system; (3) a performance database is used by a scheduler to assign grid resources intelligently; (4) a scientific result depends on the completion of a large ensemble of simulations, and scheduling and simulation management tools do sophisticated load balancing given this information. A primary goal of our work is to design an architecture for grid computing environments that exploits problem-solving context, broadly defined.

Figure 1 shows our high-level architecture for GCEs. This view is not a complete architecture in the full sense of the word, e.g., with clearly defined interfaces between layers. Instead, it is a design framework that factors out conceptually independent layers of abstraction corresponding to different functions that must be represented in an effective GCE. The layers in Figure 1 are based on three entities, defined as follows:

**Model:** A model is a directed graph of specific executable pieces defining the control-flow and data-flow in a computation. Note that we distinguish between a model and its representation in a GCE; the representation might involve only the model's name or it might involve a more sophisticated representation of each node in the digraph. Although models consist of ready-to-run pieces of code, these pieces may be parameterized.

**Model Instance:** A model instance is a model with all parameters specified. Note that some parameters may not be specified until runtime. Thus, while there might not exist a static conversion from models to model instances, the distinction between model instances and

models is still useful. For example, using two different input data sets with the same model corresponds to two different model instances.

**Simulation:** A simulation is a model instance assigned to and run on a particular computational resource. It is useful to distinguish between a model instance and a simulation because, for example, a single model instance can be run (and re-run) many times using different computational resources or different random number sequences; each of these would be a new simulation by our conventions.

Given these definitions, the framework summarized in Figure 1 can be used to organize the functions which should be supported in a GCE for a computational science grid community. The Model Management layer contains tools for creating and modifying models. The primary activity in the Model Instance Management layer is parameter definition, which turns models into model instances. The core service at the Simulation Management layer assigns model instances to grid resources. The lowest two levels in Figure 1 correspond to software and hardware resources that make computational grids possible.

Since our emphasis is on high-level, problem-specific issues, we have focused primarily on the top four levels in Figure 1. For example, in our interaction with computational scientists we find that we can make an important computation at the Model Instance Management level. In the simplest case, users assign parameter values to a model and ask a simulation management tool to execute the simulation. This level also includes tools for generating sets of model instances, e.g., ensemble generators and parameter sweep tools [7, 8]. High-level problem solving strategies and activities belong at this level as well. For example, scripting languages are used to 'program' model instance generation in the context of an outer optimization loop, or adaptation and caching strategies are employed, e.g., adjusting algorithm parameters or using a recently computed result as an initial guess for a subsequent computation. Another high-level strategy found at this level is a 'database query' mode, where scientists describe sets of model instances in an SQL-like language, and previously computed results are retrieved or new results computed, as needed. Finally, model instance management includes tools that analyze or reason about model instances. An emerging example is the use of recommender systems [24, 39] which use data-mining strategies to select problem or algorithm parameters, based on a database of previous results.

The Simulation Management level also includes a wide variety of activities. In the most straightforward case, users define a simulation by selecting resources, and use grid and data services to interact with those resources. Middleware such as Globus provides the tools needed to perform these tasks, including authentication, staging, running, and retrieving of results. More interesting possibilities for simulation management arise when we consider assigning compute resources to sets of model instances. For example, better scheduling and load balancing strategies are possible when we know more of the context (e.g., number of model instances, dependencies between instances, and requirements of each) or when we have performance results from previous simulations. More sophisticated modes of interacting with running simulations also belong at this level, including collaborative monitoring and steering of simulations, fault tolerance, and job migration. Also found at this level are services that allow interaction with a data grid, e.g., insertion, retrieval, and mining.

It is clear from even the brief discussion above that a wide variety of context-aware tools and environments will be needed to support grid computing communities. The challenge will only increase as grid computing gains in popularity among scientists outside the current generation of grid 'power users.' Flexibility, usability, and extensibility will be of paramount concern. Building vertically integrated, problem-specific environments from scratch for each problem-solving context is not a good idea; it requires too much effort and results in complex and inflexible systems. Hence, many research groups are working to define abstractions, architectures, and frameworks that allow environments to be built out of simpler, pre-existing components. Projects such as CCA [3], Harness [9], Cactus [6], and many others are making good progress in this direction. These approaches rely on carefully defined interfaces, strictly enforced via object oriented tech-

nology. Such a strategy is certainly useful *within* each of the layers illustrated in Figure 1, e.g., in defining complicated models by composing simpler models. However, we believe a different approach is needed in order to allow easy construction of GCEs spanning several of the layers in Figure 1.

Our approach does not attempt to define precise interfaces and component models for each layer of abstraction. Instead, we take a more flexible, data-centric view. By 'data-centric' we mean that *everything* in the problem-solving context can be viewed as data. We are building tools which treat as 'data' things such as a name (e.g., an executable, a machine, a user's data file), a model of a physical system, an ensemble of simulations, a binding between simulation components, a load-distribution, or a set of performance results. Data can also include activities, such as constructing a parameterized model, searching a database for previous results, running a set of simulations. By taking this approach, we believe a wide variety of tools can be built and combined to form flexible, powerful, context-aware GCEs.

## 4.2 Security

Secure access to distributed resources is another key issue for grid computing, and is an area that has received considerable attention from the grid infrastructure community. We can leverage this work in our context. However, the technology is still evolving, and there are contributions that we can make, especially in the area of supporting the kinds of research groups that will be the dominant users of the Virginia Tech campus grid.

We envision that much grid use will involve small, dynamic working groups for which the ability to establish transient collaboration with little or no intervention from resource administrators is a key requirement. Current grid security mechanisms support individual users who are members of well-defined virtual organizations. Recent research seeks to provide manageable grid security services for self-regulating, stable communities. Our prior work with component-based systems for grid computation [31] demonstrated a need to support spontaneous, limited, short-lived collaborations. Such collaborations most often rely on shared or delegated fine-grained access privileges to data and executable files as well as to grid compute resources. We are developing mechanisms that focus on management and enforcement of fine-grained access rights. Our solution employs standard attribute certificates to bind rights to users (or their surrogates) and enables high level management of fine-grained privileges, allowing them to be freely delegated, traded, and combined. Our privilege management and enforcement mechanisms enable the usage of fine-grained rights, leverage other work in the grid computing and security communities, reduce administrative costs to resource providers, enable ad-hoc collaboration through incremental trust relationships, and can be used to provide improved security service to long-lived communities.

## 4.3 Simulation Management

Many grid computing research issues fit in the general category of simulation management, which includes everything needed to submit, monitor, and retrieve results from grid-based simulations. Given our emphasis on exploiting the broad context of computational science problem solving, we are also interested in managing and leveraging results from previous simulations, perhaps run on different parts of the grid. Progress in this broad area is critical to the adoption of grid computing as a viable option for high-end computational science; hence, much of our work will be focused here. Our initial efforts are in the area of job submission tools, resource brokering, and scheduling.

The architecture described in Section 4.1 allows for a variety of job submission tools to be developed independent of lower-level details of a particular computational grid. For example, we have built a simple tool that allows users to submit jobs to any resource using grid services supplied by Globus [27]. However, the same tool can also be used to submit jobs to a non-Globus resource, e.g., a Condor pool or a 'private' workstation. Users can use the tool directly through

a command-line interface or through a simple GUI, or the tool can be plugged into a more complicated grid computing environment. For example, our core job submission facility is used in prototypes of a parameter-sweep definition tool (which generates many simulations based on a parameterized data file), and of a database query tool (where simulations are generated only as needed, if results for the required cases are not already available in a database of previously computed results).

The resource brokering problem is to select computational resources for a particular job, based on the needs of that job and current availability of resources. Resources include computers and networks, but might also include data files, executable codes, and special purpose devices. In a grid environment, current availability is limited not only by traditional factors such as machine load, but by distributed computing issues such as authorization and data locality. Also, in a grid the potential list of resources can be enormous. What is needed is a flexible framework for resource brokering in the context of grids. The matchmaking paradigm has been a successful approach to the resource brokering problem for Condor [41]. We are investigating generalizations of this approach that will work across a grid, and will allow secure, configurable, and extensible matchmaking modules appropriate to a given grid context.

In addition to resource brokering, another aspect of scheduling that we are focused on is sometimes called 'co-scheduling.' In the simplest case, it is the problem of acquiring enough computers at the same time to run a single large parallel application. Classical scheduling algorithms are based on mapping a task graph to a graph of resources (processors), and assume complete information about the resource graph. Unfortunately, in a grid context there are many practicalities that make classical approaches problematic. For example, different administrative domains might not give complete information about their resource structure. Furthermore, since scheduling is a combinatorial problem, scheduling decisions over a huge grid of resources becomes unmanageable. We are designing an architecture by which distributed global scheduling can be done in a scalable manner, while at the same time respecting administrative domains and policies [37]. The scheduling framework is offered as a web service and does not require modification of existing local schedulers.

### 4.4 Fault Tolerant Message Passing

Fault tolerant algorithms, implementations, and systems are critical to successful adoption of large-scale grid computing. As jobs are scheduled across a diverse and widely scattered pool of resources, the chances that those jobs will encounter a problem increases. This is especially true for large parallel jobs. Imagine a single parallel job running on 200 machines at the Math Emporium for one week—the chance that at least one of those machines goes down is very high. For Class 2 (embarrassingly parallel) jobs, the usual approach to fault tolerance is straightforward: if a single task fails to return for some predetermined time, assume it is lost and resubmit somewhere else. However, for tightly-coupled parallel applications (Class 3), the task is much harder. User-level checkpointing is one approach that has been used for many years. This requires code modification, and also requires more simulation monitoring from the user than is reasonable in a grid setting, e.g., 'Which of my 150 simulations died and where was it running when it did so?' Removing this burden from the user is a high priority. However, until now there has been no general, system-level solution to the checkpoint and restart problem for MPI message-passing parallel codes. We are investigating a new approach to transparent checkpointing that will allow restart and migration of MPI codes without user intervention. We are collaborating with colleagues in the Computing Systems Research Lab who are developing the *Weaves* programming framework. The approach relies on a user-level implementation of TCP/IP and Weaves support for object-based composition of unmodified code modules. Note that a system that supports checkpoint/restart/migrate as a strategy for handling faults can also be used for handling poor performance. If a simulation is not making the kind of progress we expect on some other available resource, it can be checkpointed and moved.

## 4.5 Algorithms

Effective algorithms are always critical to the success of any computing effort. Finding good algorithms in the grid context is especially critical because of the large problem-sizes and substantial computing resources involved. The stakes are higher in the sense that the cost of using a poor algorithm can be very large. The challenge of finding good algorithms is great too, since grid applications may have to execute on a wide variety of machines. Our research agenda in the area of algorithms consists of two main thrusts, one in latency-tolerant algorithms and the other in software infrastructure for dynamic algorithm selection.

A parallel algorithm is latency-tolerant if it performs well even when the computation is spread across a relatively wide area network. The most latency-tolerant algorithms typically consist of many large independent computational steps and relatively few global communication steps. Embarrassingly parallel applications (Class 2) all fall into this category. Deriving latency-tolerant algorithms for Class 3 and 4 applications is much more challenging, however. For applications that require the numerical solution of partial differential equations (PDEs), we have been studying one approach to hiding latency, namely 'domain decomposition' algorithms, for many years [32, 33, 36]. Other application areas call for other approaches. But common themes include carefully distributing data and work to multiple processors, overlapping communication with computation, and trading off computation for communication. We will continue to investigate these strategies in the context of problems of interest to our computational science colleagues at Virginia Tech.

The second broad area of our current focus in algorithms for grid computing is in the area of dynamic algorithm and software selection. This is a higher-level issue than designing a single efficient algorithm for a particular application on a particular compute platform. Instead, the assumption is that there may be several algorithms and implementations to choose from for a given computational step (e.g., solving a linear system of equations); we need to select a suitable algorithm and implementation at run-time and seamlessly integrate it into the application code, preferably with little or no intervention from the user. In a grid environment it is often impossible at development time to know on what computing resource a particular code will run. In fact, with load balancing and code migration schemes, a single simulation may move from machine to machine during its lifetime. What is needed is a strategy and a software framework to support run-time selection of algorithms and integration of codes into running simulations. The selection can be based on problem and resource characteristics known only at run-time, and can leverage scientific and performance results from previous runs using data-mining techniques. The goal is performance portability across a wide range of grid resources.

## 4.6 Problem Specific Grid Computing Environments

Grid computing research is pointless unless it yields substantial new computational power for users and enables new insight into science and engineering problems. Hence, we are committed to working closely with applications researchers in all phases of our research. Our approach to an architecture and middleware for building GCEs stresses modularity, separation of concerns, and re-use. A primary goal of this strategy is to allow users to tailor their own environment to the specifics of their problem-solving context. In order to motivate and test our work, it is important to build several 'end-to-end' systems for particular user communities. This application-oriented strategy has been a successful one in the PSE work at Virginia Tech [1] for many years, in terms of funding, in progress in computer science research issues, and in enabling our science and engineering colleagues to tackle larger and more complex problems.

Our closest collaborations thus far in the grid computing work have been with colleagues in the Department of Physics and in the Virginia Bioinformatics Institute (VBI). We are building tools and a grid computing environment to support the work of Massimiliano Di Ventra and his research group in physics. Di Ventra's group uses first-principles atomistic calculations to predict the electronic, optical and transport properties of molecular electronics devices. The

computational requirements of this research effort are enormous, requiring hundreds or thousands of simulations, each of which can require days of compute time on a small cluster. The primary simulation code used by Di Ventra and colleagues is a typical Class 3 parallel application. Bioinformatics is another source of large problems which can be attacked via grid computing. We are working with Allan Dickerman and others at VBI to build GCEs for genomic sequence analysis using widely-used tools such as BLAST (Basic Local Alignment Search Tool). These applications mostly fall into Class 1 and 2; but there are very large data files involved, so concerns of data locality and database interaction are very important. We are also working with Pedro Mendes and his group at VBI. Mendes' research is in the simulation and analysis of biochemical networks. Applications of Class 1, 2, and 3 are all of interest to this group, which makes it a particularly interesting motivating example for scheduling and load-balancing middleware. We are currently designing a grid-based version of the COPASI [34] system for modeling biochemical pathways.

# 5 Conclusions and Next Steps

Developing a usable computational grid for Virginia Tech is an attractive project from many perspectives. It leverages distinctive strengths of the university context, including a multitude of distributed compute resources, high networking capacity, and sophisticated users. The project can help meet the research computing needs of many of the most demanding users. And from our perspective, grid computing generates a host of interesting computer science research questions—from networking to security to algorithms to user interfaces. By deploying a grid and demonstrating that it is effective for real applications, our research group will gain valuable experience and exposure in the grid computing community.

In the near term, our emphasis will be on expanding the set of machines, users and applications for the campus grid. Identifying users from Classes 4 and 5 will be especially important in terms of motivating future research. Collaborations with external partners (e.g., nearby universities and government labs) should also be pursued. Eventually, leadership and administration for grid computing should transition from our research group to university support. Our research group will continue to provide leadership on campus through presentations, workshops, and other collaborations as well.


## Acknowledgments

This work was supported in part by the Virginia Commonwealth Information Security Center and by Virginia Tech Information Systems and Computing. The authors also thank Prachi Bora, Jeevak Kasarkod, Mrinmayee Kulkarni, Sandeep Prabhakar, and Satish Tadepalli for their contributions.


# References


[1] M. Abrams, D. Allison, D. Kafura, C. Ribbens, M. B. Rosson, C. Shaffer, and L. T. Watson. PSE research at Virginia Tech: An overview. Technical Report 98–21, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA, 1998.

[2] D. A. Agarwal, S. R. Sachs, and W. E. Johnston. The reality of collaboratories. *Computer Physics Communications*, 110:134–141, 1998.

[3] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computation*. IEEE, 1999.

[4] T. Beisel, E. Gabriel, and M. Resch. An extension to MPI for distributed computing on MPPs. In Marian Bubak, Jack Dongarra, and Jerzy Wasniewskii, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 75–83. Springer, 1997.

[5] R. Buyya, K. Branson, J. Giddy, and D. Abramson. The virtual laboratory: Enabling molecular modeling for drug design on the world wide grid. *Concurrency and Computation: Practice and Experience*, 2003. to appear.

[6] Cactus Project. www.cactuscode.org.

[7] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLes parameter sweep template: User-level middleware for the grid. In *Proceedings of SC2000*, 2000.

[8] A. DeVivo, M. Yarrow, and K. McCann. A comparison of parameter study creation and job submission tools. Technical Report NAS-01-002, NASA Ames Research Center, Moffett Field, CA, 2001.

[9] J. Dongarra, G. E. Fagg, A. Geist, J. A. Kohl, P. M. Papadopoulos, S. L. Scott, V. S. Sunderam, and M. Magliardi. HARNESS: Heterogeneous adaptable reconfigurable networked systems. In *Proceedings of HPDC 1998*, pages 358–359, 1998.

[10] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12:53–65, 1996.

[11] T. Epperly, S. Kohn, and G. Kumfert. Component technology for high-performance scientific simulation software. In *Proceedings of the Working Conference on Software Architectures for Scientific Computing Applications*, Ottowa, Ontario, Canada, October 2000. International Federation for Information Processing.

[12] I. Foster and N. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *SC'98 Proceedings*, New York, NY, 1998. ACM Press.

[13] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11:115–128, 1997.

[14] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, Orlando, FL, 1999.

[15] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. Supercomputer Appl.*, 15(3):200–222, 2001.

[16] D. Gannon and A. Grimshaw. Object-based approaches. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 205–236. Morgan Kaufmann, Orlando, FL, 1999.

[17] Global Grid Forum. www.gridforum.org.

[18] Globus Project. www.globus.org.

[19] Grid Computing Environments Reseach Group (GGF). www.computingportals.org.

[20] Grid Portal Development Kit. doesciencegrid.org/projects/GPDK.

[21] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. MIT Press, Cambridge, MA, second edition, 1999.

[22] J. Grudin. Computer-supported cooperative work. *CACM*, 32:30–34, 1991.

[23] Hewlett-Packard Grid Computing. www.hp.com/techservers/grid.

[24] E. Houstis, A. Catlin, J. R. Rice, V. S. Verykios, N. Ramakrishnan, and C. E. Houstis. PYTHIA-II: A knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*, 26:227–253, 2000.

[25] IBM Grid Computing. www-1.ibm.com/grid.

[26] IPG Launch Pad. portal.ipg.nasa.gov.

[27] A. Karnik and C. J. Ribbens. Data and activity representation for grid computing. Technical Report 02-13, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA, 2002.

[28] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C. Kesselman, R. Aydt, D. Reed, J. Dongarra, S. Vadhiyar, and R. Wolski. Toward a framework for preparing and executing adaptive grid programs. In *Proceedings of NSF Next Generation Systems Program Workshop (International Parallel and Distributed Processing Symposium 2002), Fort Lauderdale, FL, April 2002*, 2002.

[29] Legion Project. legion.virginia.edu.

[30] M. Litzkow, M. Livny, and M. Mutka. Condor—a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, Los Alamitos, CA, 1988. IEEE Computer Society Press.

[31] M. Lorch and D. Kafura. Symphony—a Java-based composition and manipulation framework for computational grids. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 21–24, Berlin, 2002.

[32] G. Mateescu and C. J. Ribbens. An iterative substructuring preconditioner for collocation with hermite bicubics. In C.-H. Lai, P. E. Bjorstad, M. Cross, and O. Widlund, editors, *Proceedings of the Eleventh International Conference on Domain Decomposition Methods*, pages 73–81, Bergen, 1999. DDM.org.

[33] G. Mateescu, C. J. Ribbens, and L. T. Watson. A domain decomposition algorithm for hermite collocation problems. *Num. Meth. PDEs*, 2003. to apear.

[34] P. Mendes. Biochemistry by numbers: simulation of biochemical pathways with Gepasi 3. *Trends Biochem. Sci.*, 22:361–363, 1997.

[35] NPACI HotPage. hotpage.npaci.edu.

[36] G. G. Pitts, C. J. Ribbens, and L. T. Watson. Domain decomposition and high order finite differences for elliptic PDEs. In *Parallel Processing for Scientific Computing*, pages 727–731, Philadelphia, PA, 1993. SIAM.

[37] S. Prabhakar, C. Ribbens, and P. Bora. Multifaceted web services: An approach to secure and scalable grid scheduling. Technical Report 02–26, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA, 2002.

[38] PSE research group. research.cs.vt.edu/pse.

[39] N. Ramakrishnan and C. J. Ribbens. Mining and visualizing recommendation spaces for elliptic PDEs with continuous attributes. *ACM Trans. Math. Softw.*, 26:254–273, 2000.

[40] N. Ramakrishnan, L. T. Watson, D. G. Kafura, C. J. Ribbens, and C. A. Shaffer. Programming environments for multidisciplinary grid communities. *Concurrency and Computation: Practice and Experience*, 2003. to appear.

[41] R. Raman, M. Livny, and M. H. Solomon. Matchmaking: an extensible framework for distributed resource management. *Cluster Computing*, 2:129–138, 1999.

[42] SETI@home Project. setiathome.ssl.berkeley.edu.

[43] R. D. Silverman. Massively distributed computing and factoring large integers. *Communications of the ACM*, 34:95–103, 1991.

[44] Sun Grid Technology. www.sun.com/grid.

[45] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke. CoG kits: A bridge between commodity distributed computing and high-performance grids. In *ACM Java Grande 2000 Conference*, pages 97–106, San Francisco, CA, 2000.