

# Overdrive Controllers for Distributed Scientific Computation

## PhD Candidacy Proposal Summary

Justin M. Wozniak  
Advisor: Dr. Aaron Striegel

Large distributed computer systems have been successfully employed to solve modern scientific problems that were previously untenable. Such systems create new parallel systems or bind existing resources together to harness the utility of the whole. With the advent of low cost hardware, high speed computers and sizable storage resources are pervasive and often notably underutilized. Modern *middleware* systems seek to congregate the potential of these existing systems into a powerful resource for scientific computation.

A modern perspective on distributed computing studies the ability of systems to respond to heterogenous or low reliability environments, as well as the need to obtain high performance or guaranteed quality of service from these systems. In various forms such as Internet computing, desktop grids, and even “big iron” grids, this philosophy realizes the genuinely unpredictable nature of large scale computing projects. The development effort in this field takes two important forms:

1. it attempts to build aggregate systems that are more reliable than their underlying components, easier to utilize in concert, and effectively unified administratively;
2. it attempts to provide performance that is timely and predictable.

In this proposal, the need to construct reliable, high performance storage from loosely trusted, low reliability components will be addressed. We will investigate methods to catalog and maintain large networks of storage components and present them to scientific users as a unified resource. Additionally, we will integrate this storage resource with computational tools to form a complete system for scientific usage, and we will manage the access permitted to data owners and other users. Computational abstractions and job management architectures will also be evaluated, as we intend to study scientific workloads and the relationship between jobs and data.

The need to obtain good, reliable, predictable performance when performing a new computation will also be addressed. We will investigate methods to obtain job runtime estimate information and use this knowledge to establish probabilistic guarantees on new computation. Methods to regulate systems that rely on runtime estimates will also be discussed, and policy will be developed to prevent abuse of the system. We will apply these techniques to scientific workloads and investigate the results.

A common idea underlines both techniques: the existence of an idealized model represented inside of each managing software system. This model is both queried for historical and predictive information and used by a middleware *grid overdrive controller* to make intelligent management decisions that satisfy user requirements. The design of the controller will borrow from techniques used in a variety of disciplines including mathematical modeling, system theory, and autonomic systems to manage user jobs and data. The result is a synthesis of existing building blocks and new abstractions that will result in efficiency, practicality, and scientific benefit.

# 1 INTRODUCTION

Distributed computer systems seek to overcome natural or economic limits in computer processing speed by linking multiple computers together to create more powerful but less wieldy scientific tools. A variety of existing tools may be used to parallelize numerical computation [1] or archive large data sets [2, 3]. However, the modern phenomenon of overpowered desktops has created university labs, corporate offices, and home gaming machines that contain grossly underutilized computing power and storage space. Software used to access and aggregate such widely distributed resources is an important type of middleware [4], and the creation of new middleware systems that benefit scientific users is an extremely active modern research area. In this proposal, we outline our approach to two important practical problems: the creation of a unified, reliable, secure storage system from existing storage resources and the improvement of future computation performance for batches of parameterized computation tasks.

Scientific researchers employing computational systems to perform numerical processing may implement relatively efficient solutions using ordinary workstations. Potential gains in processing power attract these users to distributed computing, where they ideally undergo a *scientific software design process*. First, they must coordinate processing among the compute sites targeted by the application. This involves practically ensuring that the application can be used in the larger scale computing environment and determining the resources required and time that must be committed to achieving a given goal. Second, they must coordinate data movement and storage before, during, and after computation and integrate the storage resources with the computational framework. This is a necessary step to ensure that the performance of the whole system actually improves over the initial workstation system. Third, they must store the completed results in a permanent manner. In many common situations, scientists may consider using distributed resources to maintain permanent archives of scientific data.

Distributed computing is a mature field in computer science, and a variety of tools are available to perform these basic required operations. However, this has not satisfied scientific users working on monumental problems requiring seemingly unlimited computation. Modern research that seeks to scrap together *whatever resources are available* may be constructed by gluing together existing tools, writing new tools to solve small problems, or building completely new architectures and computational frameworks. Conglomerations of existing software suffer from the obvious problem that while the software may be compatible, a global view of the whole system is not present. Small tools may excel at one aspect of the computing problem and provide fundamental building blocks but do not approach or manage the inherent complexity of larger systems. Completely new frameworks have been used with some success but a glance at nearby desktops or existing user software issues will suggest that successful solutions will need to evolve from the existing software compatibility environment.

Our approach to scalable, widely distributed software systems recognizes the problems associated with solutions that are either too large or too small. Essentially, our view recognizes the permanence of existing systems, dividing the framework into two parts: an existing resource fabric and a new controller: the resource fabric is unchanged, allowing for existing systems and software to function without the controller. For a variety of reasons including simplicity and security, we constrain the controller to perform operations on the resource fabric as an ordinary user. From a software perspective, the controller may be queried or called, allowing new software to be written using controller functionality as methods. The highlight of the framework is that the controller has an internal model of the resource fabric that is used when making policy-based decisions. Such a model will borrow from the mathematical modeling and analysis of computing systems, including operating systems, distributed systems, autonomic systems, and others.

We call such a new system that has these properties a *grid overdrive controller*, and the effort of our research will be investigating and developing this model as well as employing it to solve problems in dis-

tributed storage and job scheduling. While the scope of the project focuses on issues related to distributed computer systems, scientific end users are the ultimate target, and we will emphasize ways in which we may aid such developers undergoing the design process.

The remainder of this document is organized in the following way. Section 2 describes the current need for robust, self-managing systems that interact with scientific users in a reliable, understandable way. In Section 3, we describe our intended areas of study. These include databases for scientific simulation and models for the runtime management of distributed systems. Preliminary results and a review of our work are covered in Section 4. We conclude with a summary of the expected results of our research in Section 5, and a timeline for the remainder of the work in Section 6.

## 2 BACKGROUND

The work proposed in this document draws from a wide range of topics in distributed scientific computing. The emphasis of the work is the impact of systems management tools on the overall performance of complex computing systems, including computation and storage. This section describes previous work in each topic in more detail. We review existing techniques and software systems for distributed computation and storage, the management of computational resources, efficient use of computational resources through scheduling, and modern research in grid computing.

### 2.1 Distributed Computation Systems

Distributed scientific computing builds theoretical or pseudo-experimental workspaces for research atop multiple computer systems connected by a network. Combining computer resources for scientific usage or collaboration was a founding principle for the Internet and a variety of derivative commonly used software tools. An original goal of distributed scientific computing was to employ multiple processors to solve a single problem faster than could be done on any single processor. Programming interfaces were defined that allowed communication among the processors, such as the Parallel Virtual Machine (PVM) [5] or the Message Passing Interface (MPI) [6]. Such approaches clearly indicate to the user that the system is a parallel one. A second goal is to enable multiple researchers to benefit from a shared resource such as a single large parallel computer. Batch schedulers designed for this purpose are described below.

Other work in implementing user software atop multiple processors heavily focused on the single system image model. Systems such as Locus [7], Sprite [8], and Amoeba [9], attempted to pool multiple servers into a unified system that serves multiple users. A more recently developed system called Legion [10] creates a virtual computing environment comprised of software objects that are location independent. These holistic systems intend to provide users with a uniform development and runtime environment that hides the complexity of the distributed hardware.

Job scheduling is a consequence of the intersection of distributed job submission and limited computational resources. Batch schedulers were developed for the first computer systems, and continue their usefulness today as powerful, well understood systems to move jobs through compute servers or clusters. Existing job schedulers include the Portable Batch System (PBS) [11], the Load Sharing Facility (LSF) [12], and the Sun Grid Engine (SGE) [13].

The Condor [14] system enhances the batch scheduling model by gaining the resources that are available on idle workstations. This technique allows existing computation clusters to be augmented by other available computers as they become available.

Application specific job schedulers may be used in an *ad hoc* manner to solve specific scientific problems on unique resources. At Home [15, 16] computing combines potentially millions of computers into a unified computational device. This model typically contains a centralized component, comprised of a job scheduler

and data movement capability. More generalized usage of such large volunteered resources is performed by the Berkeley Open Infrastructure for Network Computing (BOINC) system [17], which allows volunteers to provide computational resources to a wide array of projects from the physical sciences to game theory.

## 2.2 Distributed Storage

An elementary problem in distributed computing is gaining access to a data record on a remote machine. Early methods involved extending the filesystem abstraction to remote directories thus creating a network file system (NFS) [18], unifying multiple remote storage services into resources that may be accessed through the filesystem. Other methods emphasize constructing appropriate naming conventions for storage location and communication protocols [19], and developing drivers to fetch or post data to the relevant server, as in HTTP or FTP. Both access methods emphasize a client-server<sup>1</sup> architecture.

However, in a cooperative model, multiple users with multiple resources attempt to combine them into a unified system. While the traditional methods described above are still required - naming and data access - new problems arise as the system takes on several new properties: the system lacks a central authority [20]; the cooperation stems from application background and is defined by the users, thus requiring administrative abilities to be granted to end users; and the complexity of such large systems requires the use of additional abstractions [21], the ability to resolve dataset names in more complex ways, and the ability to use logical, application-specific lookup procedures instead of physical locations.

Specifically, our approach to scientific computing in a cooperative storage environment involves the construction of a distributed database of datasets, in which user data is replicated over the cooperative network. Such a database is influenced by previous work in replica location, in which physical replica locations must be obtained for a given logical dataset name, as in the Replica Location Service [22]. Additionally, users of such systems typically rely upon meaningful metadata lookups given by a metadata catalog such as the MCAT [23] to obtain the logical dataset of interest.

Many other systems have used a distributed storage fabric to obtain new utility in reliability and performance. Extending the notion and method of disk striping and parity disks [24] to networks of storage services, Zebra [25] stripes data across multiple servers. A further extension is OceanStore [26], which stripes replicated data across a potentially global network of untrusted participating servers. In contrast to disk striping, full file replication is performed on untrusted servers by Farsite [27].

## 2.3 Resource Management and Grid Computing

Resource management is a broad topic in computer systems, however, in this work we are concerned with two areas: providing the user with information about the system and balancing utilization of processing power and disks to avoid overload. This work focuses on models that may be used by the software to manage large distributed systems by properly managing computation, storage, and other resources.

Job scheduling and migration is an elementary component in resource management as it balances computational load by allocating jobs to free resources, however, more fine grained models have been proposed. The Utopia [28] system, for example, uses a variety of metrics to measure the load experienced by a host which may be used to guide load balancing via remote execution. More recently, the MOSIX cluster operating system has been developed to provide adaptive resource sharing and process migration, inheriting from Sprite [29]. Additionally, disk management may be used to balance disk access utilization or disk storage utilization, the former to improve response time and the latter to reduce the risk of disk full conditions or the performance penalties involved in low free space conditions. Intuitively, balancing disk storage space

---

<sup>1</sup>The web structure could also be considered the consequence of hypertext frameworks; this can still be contrasted with cooperative storage.

in many typical usage situations will imply a balancing of disk accesses. Typical methods involve grouping data storage into volumes which are only visible to the administrator; these may be moved from server to server to equalize either of the above load metrics. In AFS [30], for example, these volumes may be moved at run time. Other resources may be automatically managed by the system, such as open network sockets [31] and name service lookups [32].

Grid computing attempts to solve all of these problems and more, including security, importing legacy computing technologies, managing virtual organizations, and high performance. The Globus Alliance is a major source of grid standards and software. At the core of grid computing is the ability to perform remote operations securely through an infrastructure based on public key encryption [33], the Globus Security Infrastructure [34]. As a result, multiple organizations may be coordinated to share and access widely distributed and variously owned resources, using the virtual organization abstraction [35]. Grid computing attempts to provide a high quality of service [36] while managing resources well, that is, providing job scheduling [37], negotiating network management [38], and maintaining high utilization [39] of the available hardware.

## **2.4 Fault Tolerance**

An important concern in the use of these systems is the extent to which an absence of one machine may cause the user to be unable to use the system at all, in the case of a fragile parallel program; determine what is available and what is not, in the case of a single resource image; or access certain data objects, in the case of object distribution. The ability of the remaining functional components of the system to maintain partial operation is called *autonomy* [40], and typically is focused on reducing single points of failure and enabling dynamic construction of operating environments [41]. Additionally, software may be able to automatically adapt to changing conditions by locating a different replica location or utilizing a local caching strategy [42]. If data sources or objects are completely unreachable, the application-level software or user must handle the problem.

System-level fault management takes three forms: fault detection, fault resiliency, and fault recovery. Fault detection on storage devices has a long history, but in the modern era of dynamic grid resources, new methods and software have become important, as ordinary RAID is insufficient [43, 44]. The Globus Heartbeat Monitor [45], for example, employs unreliable failure detectors [46] to detect problems and trigger a correction. The occurrence of faults should not damage user data or inhibit the ability of users to complete tasks. Replica creation and location is employed to prevent the total loss of data by several systems, including the Storage Resource Broker (SRB) [47] and the Grid Data Management Pilot (GDMP) [48]. Fault recovery in storage systems has a similarly long history. Recent work has focused on reducing the cost of recovery, as in the FARM system [49], however, restoring the loss of a given amount of data will always require a transfer of that size. As a result, systems like OceanStore emphasize parity based recovery models [50].

# **3 RESEARCH PLAN**

## **3.1 Databases for Scientific Computation**

Much scientific data may be tabulated and stored using the database model. However, combining these databases with large scale computing systems is not well understood. Users that have massive computation requirements may submit thousands of jobs to a job scheduler, and run hundreds in parallel. Several application areas are currently utilizing Notre Dame resources on this scale, including users in molecular dynamics and image recognition. These users require storage systems that are compatible with their computation

techniques, large enough to provide the necessary amount of storage space, providing sufficient speed of I/O performance to keep up with the large number of jobs, and providing a customizable metadata structure to make the datasets easily understandable from a scientific perspective.

From a practical perspective, these systems must be general enough that they may be easily built up from existing commodity resources and balanced to ensure high utilization. Additionally, these databases are not typically created for a solitary user but for a community of collaborators, thus necessitating robust access control mechanisms.

We propose to employ a new database system that manages scientific data in the database model while coordinating with the requirements of massive computational projects. The new system, called GEMS (Grid-Enabled Molecular Simulation) combines an accessible database of user-specified scientific information with an underlying distributed system for the management of large scientific data sets. Additionally, the system meets the practical design requirements of ease of scientific use, ability to function in a dynamic resource environment, and access control management. Specific contributions of this system are outlined in the following subsections.

### 3.1.1 Enabling Scientific Tasks

A parameter sweep is a common application of parallel computing power. In this method, a single program is selected and a set of potential inputs is formulated. The program and set are specified to a parameter sweep system, which constructs matches or tuples containing an execution site, the program, and the input set. Thus the data size of the parameter information is a small amount of input data and a small amount of output data. Modern computational grids and clusters may be driven by parameter sweeps as they easily generate a large load of jobs that may be executed in a simplified, parallel way.

Data collection for the parameter data is performed in an application-specific way, by calling into system routines. In our approach, the small parameter data is catalogued while all additional large output data sets may also be stored. This combination approach is particularly useful when performing post-analysis of job output data, as the interface provided to the user is a hybrid of the simplified parameter view and the raw output file view. In the case of molecular dynamics, for example, this may include the state of each atom at each timestep, resulting in massive quantities of data *that is being produced in parallel*, that must be stored on multiple physical storage sites. Certain collaborative situations allow users to borrow limited access to remote resources for this scientific usage. Possible constraints include the probability of the removal of this access after a certain period of time.

The construction of a large commodity scientific database of this type is a resource management problem in which a multitude of storage providers must be conglomerated into a unified resource. These unified resources must be managed to prevent the occurrence of disk full conditions in the worst case, and load balanced to prevent overuse of some systems, with respect for the fact that the systems are borrowed. Clients must be able to locate available sites based upon the size requirement and other constraints, obtain a reservation for data submission, be able to utilize this reserved resource. After the committal of the new record, the data must be locatable in a scientific way and accessible in a programmatically convenient way. More generally, managing data that is spread over multiple sites is a difficult problem. The borrowed resources are prone to revocation by the owner or failure for uncontrollable reasons, from the perspective of the remote user. The individual machines that make up the storage layer cannot be trusted to be reliable, or even to stay on. Our approach replicates whole files in response to user requirements to increase the survivability of the data sets.

Replicated data sets are much more difficult for users to manage, so the system has to provide ease of use tools to be usable, while allowing for reasonable default settings. Users must be informed of potential storage

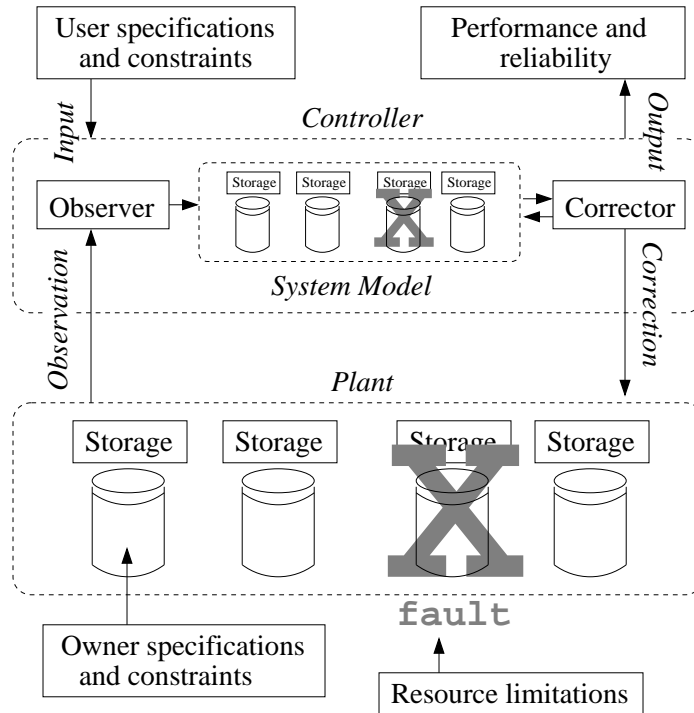


Figure 1: Schematic of a cooperative storage controller. The controller actively observes the state of the system and updates its internal model of the system status. Corrections are made in a well defined way, bringing the the system into a state that satisfies user requirements.

sites for their datasets, so that they may restrict replication to untrusted or undesirable places. Simplified but meaningful abstractions must be provided to allow the proper, efficient use of the distributed system when programming. Replicated data sets also create a serious administrative disk management problem, which will be managed by the new controller. While a great deal of previous work has been done properly managing local disks or coordinating data placement on remote disks, our new system intends to *manage remote disks* to balance storage loads and prevent disk full conditions.

### 3.1.2 Data Management Services

A simple replica management system performs basic operations to detect and handle faults. Faults or storage failures may be observed by querying the storage servers. Fault handling is performed by making additional replicas, ensuring that the user specified replica count is maintained. However, in the presence of continuous server appearance and disappearance, called *churn* [51], the amount of data that would need to be transmitted to maintain user-specified replica counts would be very large. Additionally, if a file is reduced to a low replica count, it may soon be lost entirely. Fault handling must be prioritized to prevent the total loss of data and specialized to appropriately handle a variety of network conditions and fault varieties. We intend to provide a robust fault management facility that obtains high utilization over a large number of servers, yet reliably maintains datasets in the presence of churn.

As described above, the systems of concern to this work use a controller model that uses ordinary methods to manage an existing system that is actually used. In the mature field of feedback control, this

existing system is called the *plant* [52]. A *controller* is used to maintain the plant in accordance with user requests. We intend to draw from control models to obtain useful techniques to properly control our underlying storage system, optimizing the state of the system by responding to the external disturbances. A diagram of the control model employed by our system is shown in Figure 3.1.2.

Borrowed resources create a data access control problem: the resource owner - not the replica system - sets the rules regarding how a system may be accessed. Traditionally, when constructing a federated storage system, the access control methods must be defined in advance. However, this limits the ability of users to allow the shared usage of a storage resource or dataset when remote users cannot comply with a local authorization technique. It should be noted that there are three parties to the transactions in a controlled replica system: the client, the storage site, and the replica management system. We seek to construct a system in which users may administer their storage without the need to comply with external authentication techniques, and obtain the benefits of a replica management system on their storage resources without necessarily being able to authenticate to that system directly.

### 3.1.3 High Performance Storage Access

As discussed above, the need for large scale storage systems is a result of the data creation power of highly parallel computational resources. Certain jobs additionally must be able to use the replica system as an input source, necessitating a highly parallel access method and client tools that are convenient to use with existing compute systems. Additionally, to obtain good performance, jobs and data must be co-scheduled to reduce latency and increase bandwidth. Our approach will utilize user supplied network topology information to improve the locality of data sources. We will employ a two-step process to coordinate this functionality with existing job schedulers: first, guide the jobs near the data; second, ensure the job uses the correct replica *for its execution environment*. These techniques may be isolated in environments in which one or the other is unavailable.

A complete and robust method will involve in-depth observation and analysis of the state of the system; effective communication among the client, replica service, and storage sites; and mathematical analysis. The result will be a complex model of the whole system that may be used when planning jobs, allocating resources, and selecting replicas. Such a model has the potential to go far beyond existing techniques which typically focus on only one aspect of the problem, such as matchmaking based on static attributes [53] or network analysis [54].

As discussed above, many existing replica systems create replica locations once [55]. While this does improve storage availability and survivability on grids of relatively reliable systems, this will not be sufficient on systems with high churn rates, e.g., desktop workstations distributed across multiple universities. We intend to observe churn statistics from our current testbed: a network of student-operated university resources and examine the behavior of the dynamic replica placement methods used by GEMS. We intend to defend the dynamic approach against the concern that too much network bandwidth is consumed.

## 3.2 Scheduling and Runtime Issues

Scientific users may often be able to predict performance of their applications on dedicated resources, but after undergoing the implementation of their software in a distributed system, they will typically be unable to maintain this knowledge. Unlike the dedicated computing case, users of loosely connected distributed systems cannot currently allocate blocks of time on computational resources as they lack the essential knowledge and control of the resources to make these allocations. Without allocations, users cannot ensure that their jobs will complete within a given timeframe.

While this problem is well studied in real-time computing, it is particularly difficult to attack on dis-

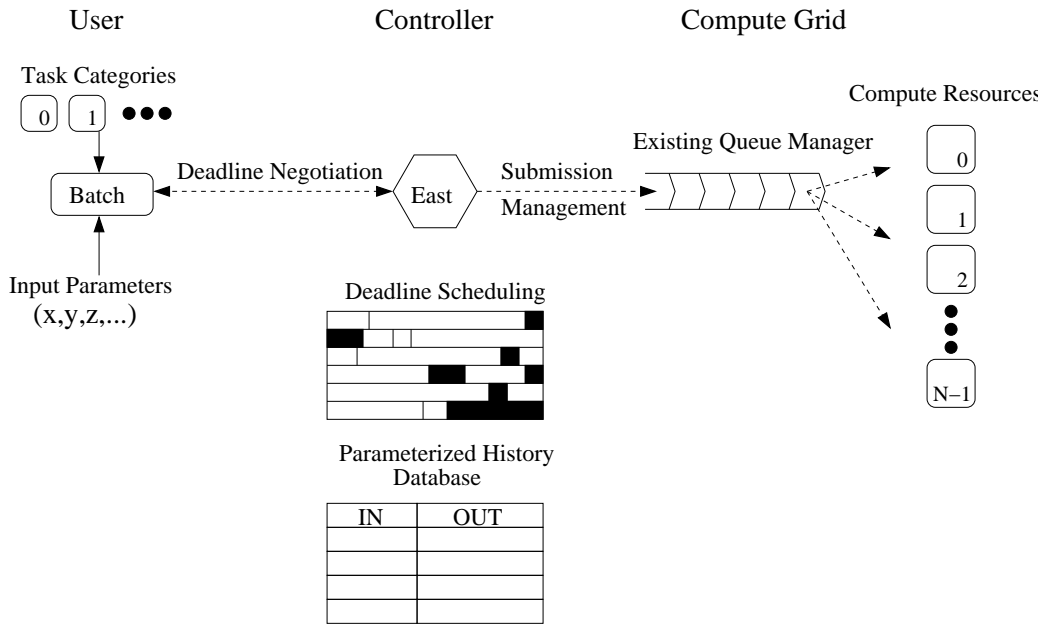


Figure 2: Schematic of a batch-aware, deadline-aware grid controller. The scheduler accepts multiple batches of parameterized jobs from the users and determines whether the schedule will allow them to complete on time. If so, they are distributed to the grid resources for processing.

tributed, dynamic resources. We intend to investigate the use of probabilistic runtime estimates combined with new probabilistic resource usage policies to improve the user experience when running jobs on remote resources. This will allow the system to obtain, guarantee, support and enforce job runtime estimates and resource allocations.

Better knowledge of job runtimes aids both users and system designers. Allocating processors to jobs in large scale systems is a complex task that requires a scheduling system to maintain certain system properties. User requirements or utilization optimizations may create a deadline constraint on the execution of a job. When users or system tools are able to provide estimates on the runtime of a job, batch of jobs, or workflow, the system may be able to provide new properties and enhanced utilization, if the deadlines are met.

When deadlines are not met, the system may be adversely affected, resulting in extremely poor utilization. Additionally, this may cause other deadline misses. To prevent this occurrence, *policed schedulers* may be employed to kill jobs and release resources that are being abused by intentional underestimation, software defects, or error. However, the effects of these policies have not been fully treated in a modern grid environment.

A great deal of insight into the performance of a scheduler policy may be obtained by relatively simple coarse-grained simulation of tasks running on a virtual multi-processor system or grid. A simulator will be constructed to examine the impact of various deadline enforcement policies in our target environment. Such a simulator will be able to rapidly prototype new policies, make use of real world experiments when creating and processing virtual workloads, and report useful statistics on throughput, turnaround time, guarantee ratio, and other data relevant to the study of deadline driven computing. Comparison and validation of simulated results are essential. In our case, we will have the opportunity to compare the performance of simulated workloads with equivalent real workloads on available campus hardware.

### 3.3 Functionality Tradeoffs in Policed Systems

A familiar tradeoff in large scale computing is that of high performance versus greater utilization, as exemplified by the Condor system which attempts to grant some resources to all users without, for example, explicitly allocating resources to solve high performance problems for any one task.

Similarly, a tradeoff exists in deadline driven computing. In a system in which the underlying system is variable and performance cannot be guaranteed, users cannot be expected to provide tight estimates, and thus should not be penalized for narrowly missing deadlines, lest the utilization of the system drop due to job eviction. For example, a user running a thousand jobs over a period of several days would be shocked to find that data had been lost due to an runtime underestimation of one second. In this example, to accurately represent performance, system utilization would be decremented by the amount of lost work.

Research in this area through simulation will provide deeper insight into the properties of this tradeoff. By experimenting with various system policies - and usage strategies - we will be able to evaluate existing policing techniques.

A tradeoff between tight enforcement and high utilization must then be reached. As the performance of the underlying system is variable, guarantee ratios for batches of user jobs are inherently variable, and this flexibility allows for the creation of new scheduling policies which are probabilistic in nature. New policy principles could include background information regarding the baseline quality and reliability of service by the underlying system, the quality of estimates, and other input data. The system must also take into account the user expectation that good estimates will provide better performance and utilization and that poor or malicious system usage will result in poor performance.

As described above, such new methods must be evaluated for utility. If, for example, a policy is easy for users to manipulate, all users may suffer poorer performance and lower utilization. If a policy makes bad decisions, it may also result in underutilization. New policies will be developed and implemented for simulation and tested with the construction of a deadline enforcement system.

While simulating scheduling policy does produce valuable insight, we will construct a deadline driven job submission system that will allow users to obtain probabilistic guarantees for batches of jobs submitted with runtime estimates. The functionality will use the existing Chirp server-side execution facility or the existing Condor functionality. However, a new prototype controller will be built to provide the new job management methodology and enforce a system policy.

This controller will thus satisfy the constraints of a grid overdrive controller as described above. It will attempt to build new functionality atop an existing system to provide better results than the former system, while actively maintaining its control level and adapting to changes on the underlying fabric.

## 4 PRELIMINARY RESULTS

These include work in both small and large databases for scientific computation, software systems under development, and models for management, security, and policy.

### 4.1 Scientific Databases

We began our research in this area by investigating scientific databases for simple, massively parallel batches of parameter sweep tasks [56]. This work allowed for a deeper understanding of typical scientific usage of grid systems, as well as representation of cataloged application data.

A cooperative scientific database called GEMS, for Grid-Enabled Molecular Simulation, has been constructed and is installed on three servers at the University of Notre Dame. Each system makes use of the existing Chirp system and thus has access to around 250 storage sites. The software is currently used for

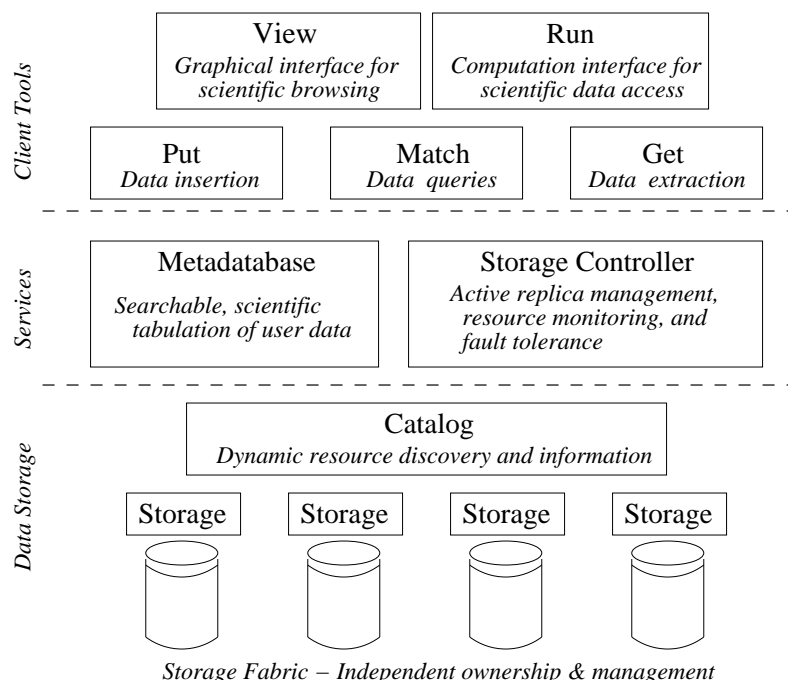


Figure 3: Key components of the GEMS system. The three levels isolate user tools that allow data insertion, retrieval, and runtime usage. Automatic services include metadata queries, replica location, and management. The underlying system essentially consists of external grid-enabled file servers.

molecular dynamics research but could be easily and effectively used by other data intensive computational tasks.

The current system meets many of the requirements previously discussed. From a scientific perspective, it offers a unified view of the storage network to enhance the ability to utilize a variety of remote storage resources in an abstract way, and provides a public, searchable metadata tagging system that may be used to catalog application-specific input and output parameters. It eases the user management of distributed data sets by implementing a “fire-and-forget” replication model in which users may submit data and neglect to fine-tune the replication process, performs active replica monitoring and management, and promotes resource load balancing by observing the state of the storage layer when creating replica sites.

GEMS enhances the ability of users to create distributed applications with a novel combination of functionality. It provides a transaction model for the importation or creation of new data by creating reservations and accepting data set committal at a later date, and it integrates with a personal virtual filesystem [21] to create a convenient I/O subsystem. Access control is handled in a flexible, user-specified manner. Building upon similar internal data structures, GEMS manages replica locations and may derive locality from user-specified topology information on a record-by-record basis. Most importantly for users of distributed job schedulers, GEMS supports computation through client utilities that may create jobs in a variety of methods, including local execution, active server execution, or job scheduler execution.

Additionally, the system meets the needs of both storage providers and storage consumers [57]. It allows users to gain access to remote machines with survivability delivered through the use of replication. It also respects storage providers by allowing them to restrict access to unknown users, evicting data, and

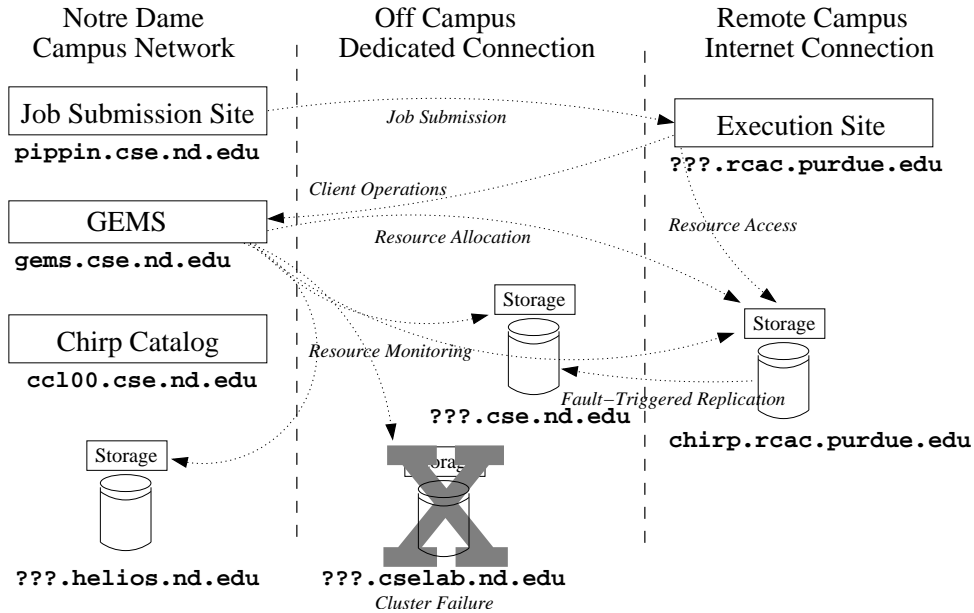


Figure 4: Usage scenario of the GEMS system. Local, off-site, and remote campus resources are monitored for failures and automatic replica creation results to ensure dataset survival. Remote jobs executing in a Condor environment obtain free storage space or nearby dataset replica locations via GEMS client tools.

monitoring the usage of their system.

## 4.2 Storage Support for Scientific Simulation

As GEMS relies upon a borrowed, relatively unmanaged storage fabric, storage faults must be handled on a continuous basis. The loss of storage sites or individual files must be detected and promptly rectified to prevent the complete loss of a record. GEMS currently uses an architecture inspired by feedback control systems [58] to handle storage anomalies, which are treated as perturbations from the desired system state.

The response to a storage failure is the creation of additional replicas to prevent total data loss: an expensive operation in the management of large scientific datasets. Additionally, while resources are consumed when performing large replications, other smaller datasets may be lost. Thus, replication must be both prioritized, efficient, and broad to prevent data loss but conservative to avoid overconsumption of resources in the presence of short term outages. We have adapted a model from control system theory using probabilistic concepts to derive a ranking system to handle observed storage problems. Using these priorities, we are able to treat replica creation as a scheduling problem.

As discussed above, grid computing is defined by the ability to work across administrative boundaries, requiring an ability to gain access to remote resources over new protocols. The GEMS system implements an extremely abstract representation of user identities, allowing the storage providers and data owners to coordinate resource sharing and collaboration without having to directly authenticate to GEMS [59]. A rendition protocol was introduced by which a client may authenticate to the system indirectly through a storage site in a situation in which authentication would otherwise be impossible. This allows for a greatly enhanced administration environment in which users which are essentially unknown to GEMS may configure and share new storage networks, maintaining their own access policies, while gaining the benefits of

replication and storage management provided by GEMS.

While enhanced replica maintenance and fluid access control are useful in a static repository, GEMS is intended to be utilized by scientific jobs as they execute on distributed scheduled resources. Current work focuses on refining the methods used to access GEMS storage by jobs running in these existing system.

The emphasis here is that GEMS simplifies the distributed storage problem to the point that storage servers may be as widely distributed as the compute servers. By scheduling jobs to access on-site replicas we eliminate the turnaround penalty caused by data staging, thus, replication now has a double benefit: storage reliability and data pre-staging.

### **4.3 Task Runtime Management and Analysis**

We have begun our investigation of scheduling systems by developing a simulator environment that allows the user to create a computational cluster, a scheduler, a policy enforcer, supply the system with a workload, and observe the results. This will allow for the rapid prototyping of schedulers for distributed systems.

A simulator called East has been constructed to evaluate the impact of the quality of estimates given to a deadline driven multiprocessor scheduler. Additionally, the simulator allows for the user to provide experimentally obtained runtime information so that actual workloads and results may be recreated. Additionally, various policy enforcement techniques may be plugged into the simulator to observe the effect of different levels of enforcement. The output of the simulator includes a great deal of statistical information that may be used when evaluating or visualizing the results.

We have offered a new method for deadline enforcement that is based on the reality of variable performance of existing computation grids. This method accepts jobs and gives probabilistic guarantees, and has preliminarily been shown to improve the results for users that provide better estimates of execution time while rarely evicting jobs that narrowly exceed their estimate. Such a model encourages users to provide good estimates, thus resulting in better performance for all users, while avoiding the utilization penalties that result from strict enforcement.

## **5 SUMMARY**

The following list summarizes the projected contributions of this work.

- An enhanced understanding of dynamic replica systems, mathematical models that may be used to interpret them, policy that makes effective use of them, and software that manages them through the maintenance of the system model;
- A new framework for grid computing that allows models of task history analysis, system performance, and enforcement policy to be utilized at runtime to obtain reasonable gains in job timeliness and completion predictability;
- A complete model for overdrive grid controllers that enhance the utility of existing systems by providing new abstract functionality and actively maintaining underlying grid resources;
- New software packages in the GEMS and East project domains.

GEMS is available at <http://sourceforge.net/projects/gems-nd>.

## 6 PROJECT TIMELINE

Spring 2007	<ul style="list-style-type: none"><li>• Draft journal paper on dynamic databases for science using GEMS system. Target: Philosophical Transactions of the Royal Society.</li><li>• Conference paper submission on rendition protocol for distributed database administration. Target: International Conference on Very Large Databases (Due March 21, 2007).</li></ul>
Summer 2007	<ul style="list-style-type: none"><li>• Draft journal paper for high level feedback control of distributed storage systems. Target: Control Journal.</li><li>• Outline of dissertation and completion of data collection.</li></ul>
Fall 2007	<ul style="list-style-type: none"><li>• Conference paper submission on probabilistic enforcement. Target: Real-Time and Embedded Technology and Applications Symposium.</li><li>• Development of dissertation.</li></ul>
Spring 2008	<ul style="list-style-type: none"><li>• Draft journal paper for probabilistic policy enforcement. Target: Real Time Systems Journal.</li><li>• Dissertation defense in March.</li></ul>

## References

- [1] Antoine Petit, Susan Blackford, Jack Dongarra, Brett Ellis, Graham Fagg, Kenneth Roche, and Sathish Vadhiyar. Numerical libraries and the Grid. *The International Journal of High Performance Computing Applications*, 15(4), 2001.
- [2] Randolph Y. Wang and Thomas E. Anderson. xFS: A wide area mass storage file system. In *Workshop on Workstation Operating Systems*, 1993.
- [3] Zeyad Ali and Qutaiba Malluhi. NSM: A distributed storage architecture for data-intensive applications. In *Proc. Mass Storage Systems and Technologies*, 2003.
- [4] Philip A. Bernstein. Middleware: A model for distributed services. *Communications of the ACM*, 39(2), 1996.
- [5] Vaidy S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4), December 1990.
- [6] David Walker. The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, 20(4), 1994.
- [7] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The LOCUS distributed operating system. In *Proc. Symposium on Operating Systems Principles*, 1983.
- [8] John K. Ousterhout, Andrew R. Cherenson, Frederick Douglass, Michael N. Nelson, and Brent B. Welch. The Sprite network operating system. *IEEE Computer*, 21, 1988.
- [9] Andrew S. Tanenbaum, Robbert van Renesse, Hans van Staveren, Gregory J. Sharp, and Sape J. Mullender. Experiences with the Amoeba distributed operating system. *Communications of the ACM*, 33(12), 1990.
- [10] Andrew Grimshaw and William A. Wolf. Legion - a view from 50,000 feet. In *Proc. High Performance Distributed Computing*, 1996.
- [11] Robert L. Henderson and David Tweten. Portable batch system: Requirement specification. Technical report, NAS Systems Division, NASA Ames Research Center, 1998.
- [12] Songnian Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proc. Cluster Computing*, 1992.
- [13] Sun Microsystems. Sun Grid Engine. <http://gridengine.sunsource.net>.
- [14] Michael Litzkow, Miron Livny, and Matt Mutka. Condor - A hunter of idle workstations. In *Proc. International Conference of Distributed Computing Systems*, 1988.
- [15] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11), 2002.
- [16] Vijay Pande et al. Atomistic protein folding simulations on the submillisecond time scale using world-wide distributed computing. *Biopolymers*, 68, 2003.

- [17] David P. Anderson. BOINC: A system for public-resource computing and storage. In *Proc. Workshop on Grid Computing*, 2004.
- [18] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the Sun Network File System. In *Proc. USENIX*, 1985.
- [19] Michael Mealling and Ray Denenberg. Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and recommendations. *IETF RFC 3305*, August 2002.
- [20] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proc. Symposium on Operating Systems Principles*, 2001.
- [21] Douglas Thain, Sander Klous, Justin Wozniak, Paul Brenner, Aaron Striegel, and Jesus Izaguirre. Separating abstractions from resources in a tactical storage system. In *Proc. Supercomputing*, 2005.
- [22] Ann L. Chervenak, Naveen Palavalli, Shishir Bharathi, Carl Kesselman, and Robert Schwartzkopf. Performance and scalability of a replica location service. In *Proc. High Performance Distributed Computing*, 2004.
- [23] Gurmeet Singh, Shishir Bharati, Ann Chervenak, Ewa Deelman, Carl Kesselman, Mary Manohar, Sonal Patil, and Laura Pearlman. A metadata catalog service for data intensive applications. In *Proc. Supercomputing*, 2003.
- [24] David Patterson, Garth Gibson, and Randy Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. Management of Data*, 1988.
- [25] John Hartman and John Ousterhout. The Zebra striped network file system. In *Proc. Symposium on Operating System Principles*, 1993.
- [26] Emil Sit, Andreas Haeberlen, Frank Dabek, Byung-Gon Chun, Hakim Weatherspoon, Robert Morris, M. Frans Kaashoek, and John Kubiatawicz. Proactive replication for data durability. In *Proc. International Workshop on Peer-to-Peer Systems*, 2006.
- [27] Atul Adya, William Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John Douceur, Jon Howell, Jacob Lorch, Marvin Theimer, and Roger Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proc. Symposium on Operating Systems Design and Implementation*, 2002.
- [28] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software - Practice and Experience*, 23(12), 1993.
- [29] Fred Douglass and John Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. *Software: Practice and Experience*, 21, August 1991.
- [30] John Howard, Michael Kazar, Sherri Menees, David Nichols, Mahadev Satyanarayanan, Robert Sidebotham, and Michael West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1), 1988.
- [31] Yiu-Fai Sit, Cho-Li Wang, and Francis Lau. Socket cloning for cluster-based web server. In *Proc. IEEE International Conference on Cluster Computing*, 2002.

- [32] Thomas Brisco. DNS support for load balancing. *IETF RFC 1794*, August 1995.
- [33] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, and Frank Siebenlist. X.509 proxy certificates for dynamic delegation. In *Proc. PKI R&D Workshop*, 2004.
- [34] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. *Conference on Computers and Security*, 1998.
- [35] Laura Pearlman, Von Welch, Ian Foster, Carl Kesselman, and Steven Tuecke. A community authorization service for group collaboration. In *Proc. International Workshop on Policies for Distributed Systems and Networks*, 2002.
- [36] Ian Foster. What is the Grid? A three point checklist. *GRIDToday*, 2002.
- [37] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459, 1998.
- [38] Ian Foster, Markus Fidler, Alain Roy, Volker Sander, and Linda Winkler. End-to-end quality of service for high-end applications. *Computer Communications*, 27(14), 2004.
- [39] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3), 2002.
- [40] Matei Ripeanu and Ian Foster. A decentralized, adaptive, replica location service. In *Proc. High Performance Distributed Computing*, 2002.
- [41] Kate Keahey, Matei Ripeanu, and Karl Doering. Dynamic creation and management of runtime environments in the grid. In *Workshop on Designing and Building Grid Services*, 2003.
- [42] Larry B. Huston and Peter Honeyman. Partially connected operation. *Computing Systems*, 8(4), 1995.
- [43] Qin Xin, Ethan Miller, T. Schwarz, Darrell D. E. Long, Scott A. Brandt, and Witold Litwin. Reliability mechanisms for very large storage systems. In *Proc. Mass Storage Systems and Technologies*, 2003.
- [44] Guillermo A. Alvarez, Walter A. Burkhard, and Flaviu Cristian. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In *Proc. International Symposium on Computer Architecture*, 1997.
- [45] Paul Stelling, Cheryl DeMatteis, Ian T. Foster, Carl Kesselman, Craig A. Lee, and Gregor von Laszewski. A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2), 1999.
- [46] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2), 1996.
- [47] Arcot Rajasekar, Michael Wan, Reagan Moore, George Kremenek, and Tom Guptill. Data grids, collections and grid bricks. In *Proc. Mass Storage Systems and Technologies*, 2003.
- [48] Heinz Stockinger, Asad Samar, Bill Allcock, Ian Foster, Koen Holtman, and Brian Tierney. File and object replication in data grids. In *Proc. High Performance Distributed Computing*, 2001.

- [49] Qin Xin, Ethan L. Miller, and S.J. Thomas J. E. Schwarz. Evaluation of distributed recovery in large-scale storage systems. In *Proc. High Performance Distributed Computing*, 2004.
- [50] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. Workshop on Peer-to-Peer Systems*, 2002.
- [51] Bram Cohen. Incentives build robustness in BitTorrent. In *Proc. Workshop on the Economics of Peer-to-Peer Systems*, 2003.
- [52] Stanley Shinnars. *Modern Control System Theory and Design*. Wiley Interscience, 1998.
- [53] Sudharshan Vazhkudai, Steven Tuecke, and Ian Foster. Replica selection in the globus data grid. In *Proc. Cluster Computing and the Grid*, 2001.
- [54] C. Lee, R. Wolski, I. Foster, C. Kesselman, and J. Stepanek. A network performance tool for grid computations. In *Proc. Supercomputing*, 1999.
- [55] Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys*, 38(1), 2006.
- [56] Justin M. Wozniak, Aaron Striegel, David Salyers, and Jesus A. Izaguirre. GIPSE: Streamlining the management of simulation on the grid. In *Proc. Annual Simulation Symposium*, 2005.
- [57] Justin M. Wozniak, Paul Brenner, Douglas Thain, Aaron Striegel, and Jesus A. Izaguirre. Generosity and gluttony in GEMS: Grid-Enabled Molecular Simulation. In *Proc. High Performance Distributed Computing*, 2005.
- [58] Justin M. Wozniak, Paul Brenner, Douglas Thain, Aaron Striegel, and Jesus A. Izaguirre. Applying feedback control to a replica management system. In *Proc. Southeastern Symposium on System Theory*, 2006.
- [59] Justin M. Wozniak, Paul Brenner, Douglas Thain, Aaron Striegel, and Jesus A. Izaguirre. Access control for a replica management database. In *Proc. Workshop on Storage Security and Survivability*, 2006.