OVERDRIVE CONTROLLERS FOR DISTRIBUTED SCIENTIFIC COMPUTATION

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Justin Michael Joseph Wozniak, B.Sc., MMath

Aaron Striegel, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana

October 2008

OVERDRIVE CONTROLLERS FOR
DISTRIBUTED SCIENTIFIC COMPUTATION

Abstract

by

Justin Michael Joseph Wozniak

Large distributed computer systems have been successfully employed to solve modern scientific problems that were previously untenable. With the advent of low cost hardware, high speed computers and sizable storage resources are pervasive and often notably underutilized. Modern *opportunistic middleware* systems seek to congregate the potential of these existing systems into a powerful resource for scientific computation. Within distributed computing, the study of these systems considers the ability of middleware to respond to heterogeneous or low reliability environments, as well as the need to obtain high performance or guaranteed quality of service. 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.

Justin Michael Joseph Wozniak

In this dissertation, the need to construct reliable, high performance storage from loosely trusted, low reliability components is addressed. We investigate methods to catalog and maintain large networks of storage components and present them to scientific users as a unified resource. The methods developed in this area resulted in the implementation of the GEMS replica management system.

The need to obtain good, reliable, predictable performance when performing a new computation is also addressed. We investigate methods to obtain job runtime estimate information and use this knowledge to establish probabilistic guarantees on the completion time of new computation. The methods developed in this area resulted in the implementation of the East scheduling simulator.

A common idea underlines both techniques: the use of a higher level managing software system that operates around an internal model of the controlled resources. 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 borrows from techniques used in a variety of disciplines including mathematical modelling, system theory, and autonomic systems to manage user jobs and data. The result is a synthesis of existing building blocks and new abstractions that results in efficiency, practicality, and scientific benefit.

DEDICATION

ad familiam meam

CONTENTS

FIGURES vii

TABLES ix

ACKNOWLEDGMENTS x

CHAPTER 1: INTRODUCTION 1
1.1 Challenges in Grid Computing 3
1.1.1 Data Management 3
1.1.2 Resource Management 4
1.1.3 Policy Management 6
1.2 Scientific Applications 6
1.2.1 Molecular Dynamics 7
1.2.1.1 Transition Path Sampling 7
1.2.1.2 Hyperdynamics 8
1.2.2 Simulation of Computer Systems 8
1.2.3 Other Applications 9
1.3 Motivation for a New Model 10
1.4 Application to Computer Systems 13
1.5 Autonomic Approaches 14
1.6 Overview of the Overdrive Controller 16
1.6.1 Model Definitions 16
1.6.2 Applications to Grid Challenges 19
1.6.2.1 Replica Management Systems 19
1.6.2.2 Policy-aware Schedulers 20
1.7 Outline 20

CHAPTER 2: GRID COMPUTING 23
2.1 Overview 23
2.2 The Computation Grid 26
2.2.1 Scheduling 27

2.2.2 Parameter Sweeps . . . 28
2.2.3 Workflow Systems . . . 30
2.3 The Storage Grid . . . 32
2.3.1 Distributed Filesystems . . . 33
2.3.2 Replica Systems . . . 34
2.3.3 Grid-Enabled Services . . . 35
2.4 Security on the Grid . . . 37
2.5 The Opportunistic Grid . . . 37
2.6 Scientific Grids . . . 38
2.6.1 Production Grids . . . 38
2.6.2 Application-Specific Grids . . . 39
2.7 Grid-Enabling Software . . . 40
2.7.1 Java . . . 40
2.7.2 Relational Databases . . . 41
2.7.3 Chirp . . . 42
2.8 Point of Comparison: SRB and GEMS . . . 42

CHAPTER 3: SCIENTIFIC REPOSITORIES . . . 46
3.1 Scientific Storage . . . 46
3.2 The GEMS System . . . 47
3.2.1 Overview . . . 47
3.2.2 Discussion . . . 48
3.2.3 Architecture . . . 49
3.2.4 A General-Purpose Metadata Catalog . . . 51
3.2.5 Comparison with Existing Systems . . . 52
3.3 Prototype . . . 54
3.3.1 Client Tools . . . 55
3.3.2 Database Server . . . 57
3.3.3 Distributed Access Control . . . 59
3.3.4 Scientific Benefits . . . 61
3.3.5 Toolset Summary . . . 63
3.4 Applications . . . 63
3.4.1 Transition Path Sampling . . . 64
3.4.2 Hyperdynamics . . . 66
3.5 Performance . . . 73
3.5.1 Programmatic Repository Access . . . 73
3.5.2 Simple Performance Experiments . . . 75
3.6 Summary . . . 77

CHAPTER 4: DATA SURVIVABILITY 78
4.1 Overview 78
4.2 Initial Experiments 79
4.2.1 Simple Replication Experiment 80
4.2.2 Discussion 82
4.3 Fault Prioritization and Feedback Control 82
4.3.1 Discussion 83
4.3.2 Control System Architecture 83
4.3.3 Control Model 84
4.4 Fault Management Philosophy 85
4.4.1 Definitions 86
4.4.2 System Response Analysis 87
4.4.3 Optimal System Response 90
4.4.4 Determining a Priority System 91
4.5 Experiments with Prioritization 94
4.6 Summary 96

CHAPTER 5: GRID INTEGRATION AND DERIVATION 97
5.1 Overview 97
5.2 Data Services for Simulation 100
5.2.1 Overview 100
5.2.2 Discussion 101
5.2.3 Replica-aware Computation 103
5.2.4 Applications 112
5.2.4.1 Operating Environments 113
5.2.4.2 Experimental Results 114
5.2.4.3 Bandwidth Analysis 117
5.2.4.4 Queueing Theoretic Analysis 120
5.3 Parameterized Workflows 125
5.3.1 Workflow Formulation 128
5.3.2 Scalable Parameterized Job Submission 130
5.3.3 Workflow Performance Analysis 132
5.4 Grid Integration 136
5.5 Grid Derivation 137
5.6 The Rendition Protocol for Access Control 139
5.6.1 Overview 140
5.6.2 Discussion 141
5.6.3 Assumptions in Shared Commodity Systems 142
5.6.4 Properties of Access Control in a Replica System 144
5.6.5 System Specifics 146
5.6.6 Application 149

5.6.7 Summary 154
5.7 Summary 155

CHAPTER 6: SCHEDULING POLICY 156
6.1 Overview 156
6.2 Case Studies 159
6.2.1 Applications 159
6.2.2 Grid Middleware 159
6.3 A Model for Deadline-Driven Grids 160
6.4 Simulation 163
6.4.1 Low-Quality Estimates 164
6.4.2 Grace Periods 165
6.4.3 Probabilistic Enforcement 166
6.4.4 Protecting Users from Bad Estimates 166
6.5 Summary 167

CHAPTER 7: CONCLUSION 173
7.1 Summary 174
7.2 Next Generation Grids 175
7.3 Future Work 177
7.3.1 Opportunistic Storage 178
7.3.2 Timeliness in Distributed Computing 180

APPENDIX A: GEMS DEVELOPMENT 182

APPENDIX B: VITA 184

BIBLIOGRAPHY 185

FIGURES

1.1 Diagram of a generic overdrive grid controller. 18

2.1 Scripts for Nimrod (simplified). 30
2.2 Scripts for APST (simplified). 31
2.3 SRB software overview. 43

3.1 GEMS architecture. 50
3.2 Example usage of GEMS client tools. 56
3.3 The GEMSview browser. 57
3.4 Access control example. 60
3.5 Performance ratio R (Equation 3.1). 67
3.6 Graphical user interface representation of hyperdynamics. 70
3.7 Archive creation times via various methods. 76

4.1 Example of fault tolerance over time. 80
4.2 GEMS replica control loop. 84
4.3 System response to induced server faults. 88
4.4 The Problem object in GEMS. 92
4.5 Priority queue performance. 95

5.1 Grid data access models. 103
5.2 Example GEMSrun script with abstract data locations. 105
5.3 Computation in a replica management system. 108
5.4 Total job turnaround time for GEMS jobs. 116
5.5 Symbols used in bandwidth analysis of data consumption jobs. . . 119
5.6 Total job turnaround time for the analytical model. 120
5.7 Queueing theoretic model for data consumption jobs. 121
5.8 Discretized Markov model for data consumption jobs. 122

5.9 Cox distribution model for data consumption jobs. 123
5.10 Parameter sweep and workflow diagrams. 129
5.11 Bottlenecked job submission model. 133
5.12 Condor usage and database load levels for the all paths hyperdynamics experiment. 134
5.13 Performance results for the hyperdynamics workflow. 135
5.14 Read-only access controlled by local storage servers. 145
5.15 Metadata layout with ACL. 147
5.16 Outline of the rendition protocol. 150
5.17 User authentication at a rendezvous point. 152
5.18 Simple three-domain collaboration. 153
5.19 Trust chain construction. 154

6.1 Deadline-driven grid computing model. 168
6.2 Acceptance and guarantee ratios for batches of jobs without enforcement. Batches of PolyDevice jobs. 168
6.3 Acceptance and guarantee ratios for batches of jobs with hard enforcement. Batches of PolyDevice jobs. 169
6.4 Acceptance and guarantee ratios for batches of jobs without enforcement. Batches of NS-Device jobs. 169
6.5 Acceptance and guarantee ratios for batches of jobs with hard enforcement. Batches of NS-Device jobs. 170
6.6 Acceptance and guarantee ratios for batches of jobs with enforcement level K. Batches of PolyDevice jobs. 170
6.7 Acceptance and guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs. 171
6.8 Acceptance and guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs. 171
6.9 Guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs, N = 32, QoE = 50%. 172

A.1 Lines of code in GEMS over time. 182

TABLES

3.1 OUTPUT BANDWIDTH FOR WW DOMAIN SIMULATION . . 65
3.2 INTERACTIVE HYPERDYNAMICS RESULTS 72
6.1 COMPUTATION TIME DEVICES IN THE EAST SIMULATOR 164

ACKNOWLEDGMENTS

I would like to first thank my advisor, Dr. Aaron Striegel, for his support and advice throughout my four years at Notre Dame. He created opportunities for me to get involved in promising projects and to travel to conferences, which provided motivational experiences and a head start for my dissertation investigations.

I offer thanks to all of the collaborators and investigators on the GEMS project. Dr. Paul Brenner kept the project relevant by focusing on real-world problems faced by simulation researchers. Much underlying software was provided or administered by Dr. Douglas Thain, who was helpful in attacking software issues. The principal investigators, Dr. Jesús Izaguirre and Dr. Jeffrey Peng, are to be thanked for their contribution to the project. I also thank Santanu Chatterjee for his collaboration on the hyperdynamics application.

I acknowledge the funding and resource providers for this work. The GEMS project was funded by NSF DBI-0450067. My education at Notre Dame was sponsored by the Arthur J. Schmitt Foundation. Additionally, computational hardware was made available by the University of Notre Dame.

I particularly thank my parents, Susan and Dr. Wayne Wozniak, for their enthusiasm for science and education which continually renewed my efforts.

My wife, Venus, deserves special credit for being a loving companion during our stay at Notre Dame.

CHAPTER 1

INTRODUCTION

Scientific computing is the use of artificial systems to manipulate mathematical representations of observed processes to arrive at a desired result. Early examples include predicting the motion of celestial objects, the birth of modern numerical methods [61], and predicting the outcome of sexual reproduction in flowers, the birth of modern genetic analysis [86]. These two example sciences have evolved over the years and been refined and integrated into the study of biomolecular dynamics, a system that enables insight into biochemical processes by applying numerical methods based on classical physical models for bonded and nonbonded interatomic structures.

When these and other scientific fields quickly became too complex for an individual to analyze, teams of *computers* - human specialists in arithmetic - were employed to mechanize the operations required. The ability to distribute work among multiple workers was due to the increase in the data size of the underlying problems: different aspects of the problem could be analyzed by different workers.

The development of the electronic computer once again enabled individual researchers to perform complex computations in addition to their academic research. However, almost as soon as computing became a viable tool for scientific applications, capability limits were reached due to limited technology. A solution

was found in *parallelism*: configuring teams of computers in a synchronized, modular way to gain better performance. These parallel computers became important tools for scientific computing and other challenging applications.

As computers became an ingrained part of the scientific process, communication among them became as important as communication among researchers. National scale communication networks were developed to link machines together, for reasons including the ability to access and use computers remotely [102], the inherent diasporic nature of academic communities, the importance of shareable data repositories [21], and computational approaches to the management of remote devices [6].

These applications diverge from the modular parallel model, and thus are grouped under the more general category of *distributed computing*. This field recognizes users, computers, networks, and other resources and entities in a complex composite system. The model is general enough to include solutions to a variety of large and small scale computing problems.

By the end of the 20th century, a variety of developers had attempted to employ complex computing infrastructures consisting of tightly-coupled, high speed parallel computers loosely linked together over long distance Internet connections. Ideally, such an architecture would enable single users to gain access to computing power limited only by societal agreements. The intrinsic technical challenges were great, so a concerted research and development effort would be required to actualize this computing potential. This new field would attempt to provide the computing benefit of large, flexible computing infrastructures to users in accordance with multiple-stakeholder policy frameworks in the form of *grid computing*.

1.1 Challenges in Grid Computing

Individual challenges in the original grid concept have developed into subfields for new research. Three of these will be investigated in this dissertation, and are introduced below with their technical and social implications.

1.1.1 Data Management

The ability of multiple computers to work together with respect to a given data set is a classical problem in distributed computing. Network-enabled filesystems, an early practical solution, enable multiple computers to share access to a common resource - a directory structure - in a manner analogous to that enabled by a multitasking operating system. Similarly, databases may be accessed over networks through protocols comparable to those used by interprocess communication on a single machine. However, as users gained access to larger numbers of high powered processors, single server data systems were not able to keep up in terms of speed or size. The need for distributed data architectures that aggregate the performance and capacity of multiple storage sites resulted in cluster filesystems and distributed databases. Grid-enabled data services enable access to widely disparate data sources over common interfaces and through scalable techniques, as discussed further in Section 2.3.

Data services provided by multiple remote entities create a resource management problem that matches or exceeds the difficulties found in the computing case considered below. First, data storage and movement capacity must be allocated to users and groups in accordance with societal principles. However, a second concern is that data is expected and required to last over time, resulting in long term agreements among users and resource managers. Additionally, data must

be protected from damage due to unauthorized accesses and system failures more than a running job. For example, a failed job requires a restart, but a permanently damaged disk could erase the results of many lengthy jobs.

Our work in grid data management focuses on the design and construction of a scientific data repository geared toward biomolecular simulation. The motivating premise is that independent stakeholders should be able to work together as users to delegate limited access to their resources to a central repository system, creating an aggregate system with high capacity, parallelizable data access, and centralized browseability. The central system controls data placement in accordance with user policy, promotes data survivability through automatic replication and fault response, and enforces access controls. Each of these features considers end users and resource providers during operation. The system has been implemented in the GEMS software and has proved itself as a viable resource for recent work in two molecular dynamics applications.

1.1.2 Resource Management

While simple technologies may be used to grant remote login access to computation services, the large scale of the potential grid user base makes simple per-computer timesharing techniques insufficient. In the presence of many computation sites, scheduling is performed at the granularity of a whole site for a significant length of time, not just time quantums, to avoid the performance penalties due to context switches. Grid resource management systems enable operating system-like scheduling facilities that typically allocate whole resources to long-running user jobs with respect to system policies, all at a macro-scale. This intermediate brokering layer promotes ease of access to a variety of disparate, het-

erogeneous resources at the whole job level. Complex software systems have been constructed to perform these services, including Globus GRAM [37] and Condor [79], as discussed further in Section 2.2.

Resource management also involves human problems, such as questions regarding who should gain access to the grid at all, and if multiple users have access, who should gain access first. These questions are complicated by the large scale user base that can request use of grid resources, as well as the presence of organizational users: entities that access the grid as an organization, not as an individual. The mix of individuals and composite entities can greatly complicate the accounting process. These questions must first be answered through the proper application of societal rules, perhaps with the aid of appropriate information systems. Then the systems must enable policy application and enforcement.

Our work in resource management focuses on a specific subproblem: the inability of users to provide accurate estimates for job runtimes necessary for good scheduling guarantees. If users could consistently provide accurate estimates, even when working with complex grid systems, schedule guarantees could be provided in a manner analogous to that found in real-time computing. However, the complexity and experimental nature of existing grid systems results in a great deal of unpredictability. The approach in this work embraces unpredictability by policing the system in a probabilistic manner for users that provide the worst estimates, thus providing better guarantees for users with better estimates. This approach leads to a game theoretic situation in which users compete to provide the best estimates, obtaining good results for themselves and enabling the system to run with better predictability and high utilization. Simulations conducted with the East software demonstrated that a simple policing technique can provide better

results to users with better estimates.

1.1.3 Policy Management

Grid policy is the imprint of societal principles, agreements, priorities and goals applied to grid infrastructures. While grid computing intends to provide more computational capabilities in a general sense, hard decisions must be made to determine which user will gain access to what resource when contention exists. Both users and resource managers have an impact on policy. When presented with a great deal of resource choices, users must specify how their applications will access the underlying components of a complex distributed system. Resource managers must protect the integrity of their systems, and typically desire the capability to grant and restrict access to different potential clients. Much work in grid access control and policy has considered the basic security problem, as discussed further in Section 2.4.

Our work in grid policy focuses on the complex interface between combining and selecting resources for use in a given application. By starting with a data-driven computing model, we show how the data environment provides a landscape in which users can corral jobs and data in an effective manner.

1.2 Scientific Applications

The technical problems that make grid computing difficult are outweighed by the scientific benefit of addressing challenging applications. Several applications of scientific problems are considered and used as motivation in this work. We provide some background on them below.

1.2.1 Molecular Dynamics

Molecular dynamics is the study of the Born-Oppenheimer approximation to chemical structures [75]. This ball-and-stick model provides useful insight into the motion of molecules while reducing the complexity inherent in a full model that includes subatomic particles. The computational exploration of these models gained legitimacy [119] in the mid-1980's as quantum mechanical ideas from the 1930's and biomolecular breakthroughs in the 1960's combined with powerful computers.

Molecular modelling provides valuable insight for chemists developing specialized compounds. Structure databases and visualization techniques allow for the rapid evaluation of the geometric properties of potential solutions. Once possibilities have been chosen, the activity of these molecules over time may be investigated computationally by molecular dynamics. Additionally, these simulations may be monitored and sampled to obtain statistical properties such as entropy or free energy.

Our work in this dissertation has been applied to two particular subtopics in molecular dynamics. Each of these was investigated with the help of modified versions of the ProtoMol [84] molecular simulator.

1.2.1.1 Transition Path Sampling

The first, transition path sampling (TPS) [24], charts the movement of a molecule as it changes from one shape to the next. The state of the system is modelled as a point in phase space, and TPS attempts to chart the path taken between one low energy state and another. TPS requires a multitude of similar but largely independent parallel simulations of the same molecular system, testing

the limits of computing systems, while generating a great deal of particle position data as a transcript for analysis. Results from the TPS application are described further in Sections 3.4.1 and 5.2.4.2.

1.2.1.2 Hyperdynamics

The second, hyperdynamics [145, 146], intends to greatly increase the speed at which a simulation progresses, sacrificing some accuracy for a fuller sampling of the states encountered by a molecular system. The method applies additional bias forces to force the system state out of regions that have been sampled adequately and into regions of interest that are rarely encountered. This enables a better understanding of the simulated molecules as the rare states often have critical properties. Hyperdynamics does not benefit from heavy parallelism but is notable for its complex workflow structure as bias forces are added. Experimental results in hyperdynamics are presented in Sections 3.4.2 and 5.3.3.

1.2.2 Simulation of Computer Systems

Computer simulations are often used to provide simulated performance results for hardware that does not exist, or for design features that are too expensive to construct for experiment. SimpleScalar [27], for example, allows the user to specify an instruction set for simulation, possibly tweaking some architectural parameters. A compiled program is then fed into the simulator, which simulates the program execution and returns a variety of performance results. An example use of SimpleScalar on a distributed computing system would formulate a large set of test programs and architectural changes, then run each experiment in parallel on a different computation site. The end results could then be accumulated and

design features could be evaluated.

Distributed systems themselves may also be studied *via* simulation. Computer networks are complex systems that often result in unpredictable behavior. Building networks of routers and cable is not an affordable method to test new networking techniques, thus network simulators are used to provide insight into how new applications, routing algorithms, or network architectures may perform. A popular network simulator, ns-2 [95], may be used to determine the expected performance of these components by simulating their behavior. Like SimpleScalar experiments, large batches of ns-2 computations may be distributed in a large computing system, after which performance evaluation of the simulated network can begin.

1.2.3 Other Applications

A variety of non-simulation scientific computing projects serve as reference points for our work. Scientific data is typically stored in large repositories and distributed to clients for analysis. A modern example of a scientific database of this type is the Sloan Digital Sky Survey (SDSS) [45], which makes volumes of astronomical data available to the public. The current release of the data set [4] includes 10.0 terabytes of observations, including 287 million individual objects. As a reminder of the scale and complexity of such projects, it is notable that as recently as the year 2000, "express courier" was part of the data movement architecture for SDSS data.

Archives of microcosmic data are also important motivators and provide a variety of computing challenges. A particle physics grid is being constructed to process, store and distribute high energy physics data from CERN [26]. The

project as a whole will produce about 10 petabytes of observations per year, requiring the power of over 100,000 desktop-class computers.

Image processing is another application that combines numerical computing with the employment of large data sets. Here at Notre Dame, the Computer Visualization Research Laboratory (CVRL) has produced a photographic gallery of human faces on the order of 100 gigabytes. While the size of the data set does not match the projects mentioned above, distributed algorithms that intend to process or search the whole database [33] encounter job placement and data movement challenges.

1.3 Motivation for a New Model

Distributed computing is a well-studied field in computer science, and a variety of tools are available to perform the 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. This model borrows from the mathematical modelling and analysis of computing systems, including operating systems, distributed systems, autonomic systems, and areas.

We call a system that has these properties a *grid overdrive controller*, and the overarching effort of the research presented here investigated and developed this model as well as employed it to solve problems in distributed 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 use the model to aid such developers undergoing the design process.

The software solution to these types of problems presented here takes the form of a running service that responds to user requests and manages the underlying system in question. Designing such a system in an *ad hoc* manner is problematic. The services provided to the end user - its programming interface - is fundamen-

tally dependent on the mingling of state between the existing system and the new service. Additionally, the form and methodology of its management services is very vague, and is not easily conceptualized as a service.

Our response to these challenges may be described through the use of an accessible, extensible model for a variety of solution systems. The controller provides well-defined inputs and outputs for end users as a service. Additionally, it defines a management model whereby certain prescribed observations about the underlying system result in consequent corrections that are applied by the controller.

Unlike a procedure call-like service, controllers manage the state of the underlying system and guide it toward a steady state as specified by user policy in an autonomic manner. This convergent framework invites comparison to feedback control system theory and existing work on controllers. In the model, state is clearly separated into that of the controlled system, which may be indirectly manipulated through well-defined existing interfaces, and the controller, which primarily models the existing system as directed by well-defined observations.

We are thus motivated to denote an abstract model for systems that handle these types of situations for the following reasons:

1. It will demonstrate similarities among systems that were created to perform different tasks;

2. It will clarify the operation of existing systems; and

3. It will provide a starting point for reasoning about new systems and formulating their design.

1.4 Application to Computer Systems

Wide area distributed storage systems pose large challenges to the application developer. For example, employing hundreds of unreliable devices as a storage layer is not a feasible way to begin a chemistry project. But the elementary task of "just put this somewhere" on such a fabric involves a variety of intermediate steps, many of which may be automated. Additionally, the ongoing task of keeping the data alive in the presence of churn cannot be left to the user, so an automatic system must be deployed.

Additionally, basic job schedulers are not equipped to deal with the limitless range of possible user access requirements. Users may wish to automate job placement among several schedulers, a process called *metascheduling*. Similarly, users may wish to create scheduling policies for which the scheduler was not designed. A solution to this problem is to wrap the scheduler inside a thin higher level policy aware scheduler, encapsulating the pre-existing functionality through relatively minor translations of query methods. The task of scheduling new jobs with respect to the policy is delegated to this new system which is aware of the state of the pre-existing system. As in the storage case above, the ongoing task of maintaining the policy over time must be automatically handled.

Finally, grid controllers derive from the observation that compute grids are not fully mastered by any individual. Users often cannot change the underlying systems upon which they rely, for administrative or technical reasons. Users cannot monitor system performance around the clock. And users cannot calculate optimum system utilization without proper tools. The model presented herein will address these concerns and more.

1.5 Autonomic Approaches

Over the past few years, the *autonomic computing paradigm* has been proposed as a solution to addressing the management of processes in distributed computing, particularly with respect to performance optimization and fault tolerance. This model emphasizes the ability of a system to take its own state into account when responding to user requests. For example, a simple autonomic system could evaluate its response time for a given user computation, and attempt to swap algorithms until the fastest response time is obtained. Often, the system is designed to promote the successful application of *emergent behavior*: a complex system response to a relatively simple set of user instructions. An example application could be a self-optimizing peer-to-peer network in which a simple gossip protocol is employed to simply maximize local performance, but the aggregate global result could be turbulently complex and seemingly unpredictable.

An introduction into autonomic computing on the grid starts with an extension of middleware, as the tasks required of autonomic systems exceed the capabilities [44] of conventional middleware systems. Starting from the middleware model, *adaptivity* is added. This allows the system to continue to function without human management when nearby resources and user demands are rapidly changing. This implies the ability to be *fault tolerant*: it continues to function when nearby resources fail. This is often achieved through the use of a higher-level registry of low-level redundant resources [139] which may be swapped out upon failure detection. Next, the system is given the ability to *manage* resources automatically, optimizing their utilization or enforcing a system policy. Finally, the system is exported to the grid as an open service compatible with grid tools [115].

An example application of autonomic management of a distributed computing

system is for soft real time applications. In this setting, computations performed on a variety of computation sites are constrained to complete within a given amount of time. An autonomic scheduler can adjust load levels on the computation cluster to prevent overloading [56], monitor the state of the tasks and their runtime constraints, and possibly pre-empt low-importance tasks if required to satisfy high-importance tasks.

The theory of control presented here draws from control system theory [120], a methodology for controlling an existing device through the construction of a regulating master device. A control system operates by observing feedback information from the controlled device and comparing it to the desired state mandated by the external user, making corrections to the controlled device to bring the two signals together. An commonly encountered example control system is the autopilot feature of an airplane. The pilot states the desired flight path of the plane, and the autopilot maintains this course in spite of turbulence, mechanical fluctuations, bird strikes, and other perturbations by making proportionate corrections to the course, until the pilot resumes control.

Feedback control methods in this framework take on a heavily numerical flavor. Their implementation relies on the ability to make measurement of state deviation from the user requirements and to implement proportionate responses over time. Thus we focus on a relatively narrow class of system operation, parameter tuning, and broadly expand its application.

Other models for autonomic response to distributed computing problems have been proposed. For example, the framework offered by AutoMate [7] provides a grid-enabled autonomic component suite built on existing standard grid software systems. The higher level model behind the motivating research project takes on

a heavily biological flavor [99] as it proposes a broad range of delivered functionality, including self-awareness and self-healing. Other systems take a rule-oriented approach: if certain conditions are met, a prescribed response is taken. Rudder [7] enables the user to present these rules and actions to enable component swapping. In the case of partially conflicting rules, fuzzy logic may be applied, as in the AutoPilot [142] system.

1.6 Overview of the Overdrive Controller

The model presented here constitutes a significant reduction of the realm of functionality promised by an autonomic system. The controller model here is essentially a software design framework that promotes clarity in the information flow through the system.

1.6.1 Model Definitions

Overdrive grid controllers are defined as a class of software systems that control an external software system called the *plant*. They are characterized by five properties: separation, internalization, vigilance, proportionality, and convergence. These are in addition to the properties of the underlying grid system which the controller intends to enhance, including scalability, standardization, and collaboration. Definitions of these characteristics are given below:

- **Separation:** The controller is separate in design from the plant, and no plant modifications are required to implement the controller functionality. Thus, in accordance with software engineering principles, the new system is limited to interact with the plant only through established interfaces.

The public interfaces employed by the controller are less likely to change over time, resulting in better maintainability for the controller. Additionally, some underlying systems may not be modified for lack of access or backward compatibility.

- **Internalization:** The controller employs an internal model of the plant to make decisions. This characteristic is notable because it addresses the ability of software systems to perform complex calculations with respect to a simulated or idealized plant that exceed human ability; additionally, it indicates that all automatic decisions are made with view of the plant that is always slightly out-of-date.

- **Vigilance:** The controller observes the state of the plant to update the internal model over time. Additionally, it uses maintains user requirements while operating asynchronously and autonomously in the presence of faults and contention for resources. Thus control actions are performed to achieve user objectives, which are observed again in an ongoing process.

- **Proportionality:** At each decision point, the controller attempts to make appropriate, timely state corrections to the plant by evaluating the user value on the elements upon which the controller is ultimately acting. This characterizes the numerical view on the system. For example, fault tolerance corrections to a data system could be prioritized with respect to the user worth of the data sets in peril.

- **Convergence:** The system is driven toward a state at which no further action need be taken to achieve a user objective barring external changes. This, of course, precludes the active monitoring of the environment for user

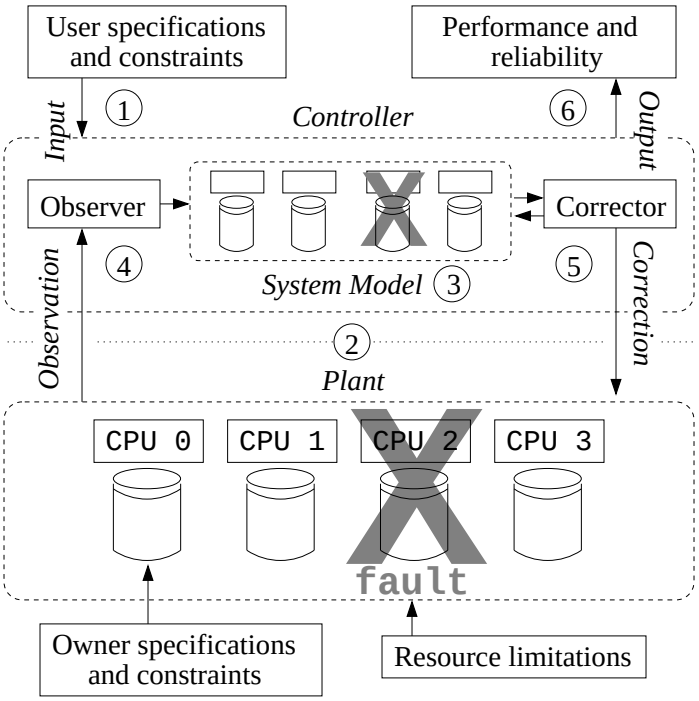


Figure 1.1. Diagram of a generic overdrive grid controller.

input or other external changes such as unpredictable partial system failures. For example, a replica placement system adhering to this model would terminate active operation once replicas have been satisfactorily placed after some finite time period. The system would then only consume additional disk space or network resources in significant quantities upon the occurrence of an activating event.

Figure 1.1 illustrates an abstract overdrive grid controller. The autonomic system responds primarily to user requests, as shown at the top ①. User calls to the controller are processed entirely independently from potential user calls to the underlying plant, and the systems are encapsulated from each other ②, maintaining the principle of separation. An internal system model ③ is supplied

with ongoing updates from an observant component ④; abnormal states may result in corrective action ⑤ that is asynchronous from the user perspective, thus providing vigilant services. Finally, the end results of the system ⑥ are returned to the user in accordance with system properties; the convergence constraint ensures that the automatic processes will wind down given enough time and the absence of additional events.

1.6.2 Applications to Grid Challenges

In this section, we consider how this model may be applied to the problems addressed in this work in the areas of distributed storage and scheduling policy systems.

1.6.2.1 Replica Management Systems

Much of the work in this document was invested in the GEMS replica management system. The system was designed to control a pre-existing network of independent storage sites for use by a relatively small set of users. Since the underlying Chirp servers were not to be modified, the principle of separation was maintained: the only access to these systems was over the Chirp network protocol. GEMS maintains an internal status of the replicas as an intersection of information obtained from the servers as well as from the Chirp catalog. Correction may be automatically made to the system by allocating and creating additional replicas, thus bringing the system back to one allowed by user requirements. In the idle state, the system makes no additional replicas and moves no significant data, thus converging to a user-allowed state. Thus the system actualizes a distributed storage controller.

1.6.2.2 Policy-aware Schedulers

Similarly, the hypothetical policy enforcement system studied by the East scheduler in this work also implements the model. The system intends to implement additional properties of benefit to contending users on a limited supply of pre-existing computation resources. Thus the metascheduling functionality is separated from the underlying resources. An internal model of the system is represented as a combination of the deadline calendar as well as the state of computation sites and their jobs. Since the jobs may be submitted and executed any time before the deadline, autonomous job submission, monitoring, and possible termination are desirable capabilities. These capabilities may be contrasted with the "try-to-do-it-now" strategy typically implemented by schedulers. When the system encounters few errors, corrective action is minimal, and upon completion of all requested batches, the system simply retires to an idle state.

1.7 Outline

The controller model developed as a common aide for reasoning about and solving practical problems. In the remainder of this dissertation, we will present these background projects before presenting the abstract form of the controller model.

In short, the product of this research may be categorized into *technical* research and development, *applications* support and innovations, and *conceptual* contributions to the autonomic software development process. The bulk of this dissertation is concerned with technical work in distributed systems, including algorithms and policies such as the *priority queue for replication* (Section 4.3.2), the *rendition protocol* (Section 5.6.5), and *probabilistic enforcement* (Section 6.3).

Applications work includes the enabling technologies used in transition path sampling (Section 3.4.1) as well as progress in computational hyperdynamics such as the *performance ratio* (Equation 3.1) and the *interactive hyperdynamics data sweep* (Section 5.3.1). These topics are integrated within the overdrive controller conceptualization, which inspired and guided these implementations and evinces our interpretation of the meaning of the grid (Sections 5.4 and 5.5).

The remainder of this dissertation is structured as follows:

- **Chapter 2** reviews prior work on specific topics in grid computing as well as descriptions of important grid software systems.

- **Chapter 3** presents the opportunistic grid-enabled scientific repository as exemplified by our software, GEMS. Building a usable, reliable repository involves the consideration of features targeted toward human scientific users such as data organization and programmatic data access. The architecture described in this chapter thus integrates user-oriented features as well as solutions to basic storage federation questions. We present two example real-world research applications that benefited from GEMS functionality, and we describe the basic performance characteristics of the software.

- **Chapter 4** addresses data survivability issues on a resource fabric of unreliable, independent storage sites. GEMS originally employed a simple replication system that was developed into a robust fault detection, prioritization, and correction system capable of keeping data available in the presence of a great deal of system outages. Chapters 3 and 4 round out a solution to the data management challenge described above.

- **Chapter 5** describes how a data system may be used as a foundation for a

complete grid system, augmenting the ability of schedulers to work well with complex data systems, and managing policy as defined by users and administrators. Programmatic techniques for working with replicated data files and parameterized data sets are presented in the context of real-world, computationally intensive molecular simulations. Additionally, a novel access control method is used to develop a comprehensive framework for resource federation and delegation. This chapter constitutes our response to the policy management challenge.

- **Chapter 6** considers our problem of interest in scheduling. Complex calendaring systems that schedule jobs far into the future and make guarantees about the timeliness of delivered computational results rely on good user estimates of job completion, which is difficult for users to provide, or forced job termination, which results in low utilization and lost work. We present a metascheduling controller that implements a novel probabilistic policing solution to the deadline-driven computing problem, and demonstrate that it provides an incentive for users to provide good estimates, while avoiding the adverse effects of strict enforcement. This realizes an application of policy that is autonomically applied to a grid system by a higher level controller, providing another example application of the controller model - as a response to the resource management challenge.

- **Chapter 7** concludes with a final conspectus of this work. Additionally, we review how the overdrive controller features were applied to the practical data management, scheduling, and policy problems that motivate this dissertation. Additionally, some further potential investigations are described.

CHAPTER 2

GRID COMPUTING

The work described 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 Overview

By the end of the second millennium, scientific computing had entered into an exciting era. Desktop computers offered sizable computation ability. Relatively complex statistical analysis could be performed by tools written for typical office hardware. The Internet had become a common tool for rapid communication and data transfer around the world. Researchers could access their computational resources over wide area networks, obtain results, and ship them anywhere quickly. Results could be posted on the World Wide Web, enabling browsing and searching.

However, scientific computing had not yet solved every remaining problem. More computational power was needed to perform numerical computations on ex-

panding numbers of variables, more storage space was needed to retain the growing bulk of computational results and instrument readings, and faster network connections were needed to enable large scale collaborative projects. Additionally, the possibility of collaboration with colleagues at remote locations was tantalizingly close. Despite the presence of global network connectivity, users were restricted by a variety of limitations including inflexible security architectures, the inability to seamlessly move jobs to diverse sites, the complexity of accessing and managing data at remote locations, difficulty navigating the web of cables connecting users and resources with efficiency, and scalability limits in existing systems.

Early grid architects framed these problems with a power grid analogy [48, 49]: users should be able to gain access to remote resources by plugging into a standard service. The authors frame complex distributed computing systems as an infrastructure that promotes societal progress in a manner analogous to other historical investments, such as the railroad or interstate highway systems. By inviting book chapters from a variety of authors, they demonstrate the wide appeal and impact of the new technologies.

The breadth of the grid concept could lead to a bloat of possible conceptions of what the grid is about. In a simple article entitled "What is the Grid?" [47], Ian Foster attempted to crystallize the conception of the grid into a checklist of three simple bullet points. He defines a grid as a system which

- integrates systems across administrative boundaries, exceeding the capabilities of traditional authentication and access control systems, enabling new forms of remote collaboration, resource sharing, and resource aggregation;
- employs standardized network communication protocols for system operations, reducing the grid to an interface which may be implemented by a

variety of new software systems;

- provides better quality of service to users as measured by a variety of metrics.

Further inspection led to the dissection of the grid into its component parts, which were found to be social elements. Collections of collaborating institutions were defined as virtual organizations (VOs) [53]: organizations of users and administrators working on common computing problems, possibly for a temporary period. The architecture of the grid, then, is simply a widely understood and spoken communication protocol which enables these social structures. More specifically, technical grid entities are represented by grid services that make use of open technologies such as the Open Grid Services Architecture [51], a web services [63] design that enables event-driven port-to-port programming models [59].

More recently, Heinz Stockinger took a different approach to the grid definition problem by surveying [125] a variety of active systems researchers for definitions of grid computing. His results include a mish-mash of definitions; for example, the respondents differ over whether grid computing is a subfield or superfield of distributed computing. However, the work is notable for its breadth of discussion and attempt to include a diverse community in a complex dialogue.

Regardless of the specific technical definitions, the motivation for the original grid concept and the justification for expensive production grids available today is large scale scientific computing. Various applications in this area struggled to make use of the complex distributed infrastructure, but common solutions can be found to provide the operating system-like services required. These include three major features, such as

1. Access to processors, process scheduling, and job management (Section 2.2);
2. Access to storage devices, files, and replica sets (Section 2.3); and
3. Protection of user and system resources from disruption (Section 2.4).

Additionally, grids must be assembled and presented to scientific users through the use of existing software systems. Our background investigations thus include

4. Assembling grids using opportunistic methods, that is, harvesting available spare resources to create a grid (Section 2.5);
5. Using grids to accomplish real-world scientific tasks (Section 2.6); and
6. A background on grid-enabling software used in this work (Section 2.7).

2.2 The Computation Grid

Distributed scientific computing systems construct theoretical or pseudo-experimental workspaces for research atop multiple computers 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) [127] or the Message Passing Interface (MPI) [149]. 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 [148], Sprite [97], and Amoeba [131], attempted to pool multiple servers into a unified system that serves multiple users. A more recently developed system called Legion [65] 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, providing grid-enabled data access [154] as well as a scalable file system [155].

Computing on the grid, however, is essentially a resource management problem. Resources must be coordinated via a process called *metacomputing*. In this model, decisions about where to execute user computation are made by a scheduler. This centralized system delegates tasks for execution to sites that advertise their computational services and respond to understood queries.

The Globus Resource Allocation Manager (GRAM) [37] provides a front end over existing local job queues. GRAM has been extended to handle advanced reservation [123], resource co-allocation [38], and dynamic virtual environments [72].

2.2.1 Scheduling

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) [68], the Load Sharing Facility (LSF) [167], and the Sun Grid Engine (SGE) [87]. These schedulers often

operate as the workhorses of a compute grid, providing the majority of the cycles available to the user. When pooled into grid-enabled superclusters *via* the GRAM abstraction and cluster federation, they may be employed *en masse* to tackle huge problems.

Simulation of computational grid resources is an important tool when testing scheduling algorithms and techniques. The Bricks system [130] has been used by several researchers to test algorithms and grid performance. For example, to effectively schedule parameter sweep jobs and associated file staging on an internetwork of compute hosts, Bricks was used [29] to evaluate various scheduling algorithms.

2.2.2 Parameter Sweeps

While various toolsets such as the Java CoG kit [144] and others [60] have reduced the complexity associated with interfacing with the grid, supporting interfaces must still be developed to bridge the final interface to the application. For parameter-driven simulation environments with large parameter spaces, the development gap between the existing tools and the actual results can be quite significant.

A very common use of the grid is to provide a solution to a so-called "embarrassingly parallel" problem. In this problem, one has a set of data points to evaluate, each of which is generated by the same basic simulation with a few different input parameters and none of which depends on the intermediate output of another task, implying no communication among tasks. There are a great many uses for this model by researchers in a variety of disciplines when conducting experiments by simulation. Examples include simple cases where a range of random

number seeds should be input to a simulation, or one of the more complex case studies discussed below. The size of the user pool and the inherent complexity of the grid as a computing resource have led to the design and construction of a variety of tools that implement a solution to the above parallel problem.

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. The parameter sweep model is represented by important implementations such as Nimrod [2] or the Parameter Sweep Tookit (APST) [30].

Nimrod/G provides several basic services to grid programmers through a simple scripting tool and a set of shell programs. To prepare a set of tasks for execution in Nimrod/G, the researcher writes a script to specify the variable parameters and the list of commands to be executed, which may include node-to-node file copies, substitutions, and other programs. The provided tools must then be used to build up a database of computational resources, connecting these to the task. Nimrod/G builds up a task list by varying the parameters in the domain specified by the user, and the user then executes the task list, and may observe its progress by examining the database. An additional tool that builds upon Nimrod/G is Nimrod/O, which provides more advanced functionality for parameter optimization. An example use of Nimrod is shown in Figure 2.1.

Nimrod scripts:

This file runs `sim` with inputs {0, 1, 2}.

```
parameter x from 0 to 2 step 1
task main
  node:execute sim $x
end task
```

Set up a compute resource and submit.

```
nimrod generate sim.pln
nimrod resource computer.edu
nimrod portalapi addrun sim G
nimrod addserver sim computer.edu
nimrod portalapi startexp sim
```

Figure 2.1. Scripts for Nimrod (simplified).

A different approach to the same problem is taken by the AppLeS Parameter Sweep Template (APST) [78]. In this framework, the researcher must know all the tasks and parameters in advance, or produce this information by a separate script. This information is written into an XML file that provides the input for APST. The XML file also contains the requested computational and storage resources, input and output files, and other system information. The user then executes the APST client, which automatically executes the tasks on the various resources, copying files as necessary. An example use of APST is shown in Figure 2.2.

2.2.3 Workflow Systems

Computational workflows allow large computations to be split into a partially ordered set of workflow elements, each of which results in a record in stable storage. The workflow model is appropriate for experimental computing systems because the process is restartable in the presence of failure. Original scalar workflow

APST scripts:

This file sets up a compute resource and runs `sim` with inputs {0, 1, 2}.

```
<apst>
 <compute><host id='compute'>
  <ssh server='compute.edu'/>
 </host></compute>
 <tasks>
  <task executable='sim'
        arguments='0'>
  <task executable='sim'
        arguments='1'>
  <task executable='sim'
        arguments='2'>
 <tasks>
</apst>
```

Submit.

```
apstd sim.xml
```

Figure 2.2. Scripts for APST (simplified).

systems were implemented to aid the compilation of complex software, such as Make [46] or Ant [14].

Workflow systems for the grid include the suite of software that comprises the GriPhyN Virtual Data System (VDS) [166]. This system originated with Chimera [54], a virtual data system that codified the workflow elements, and Pegasus [41], which maps virtual transformations to actual grid tasks scheduled on real resources. GridAnt [143] extended the robust workflow model of Ant to general grid computations, and DAGMan [138] integrates workflow computing with the Condor opportunistic scheduler described below.

Modern work seeks to manage the complexity of the workflow on an opportunistic grid [40] by automating complex decision making when faced with changing resources or failures. Workflow construction is thus an interdisciplinary process, combining the skills of the application researchers with computer scientists and systems experts, resulting in an iterative software design process [62].

Additionally, data movement strategies may be influenced by the workflow paradigm. GriddLeS [3] supports data abstraction in workflows by using Bypass [134] and multiplexing various data access methods through common underlying clients. With similar goals, the Batch-Aware Distributed File System (BAD-FS) [17] co-schedules data placement and job activity while managing failures, preventing resource overload and thrashing, and building cooperative remote caches.

2.3 The Storage Grid

An elementary problem in distributed computing is gaining access to a data record on a remote machine. Other methods emphasize constructing appropriate naming conventions for storage location and communication protocols [85], and

developing drivers to fetch or post data to the relevant server, as in HTTP or FTP. Both access methods emphasize a client-server[1] architecture.

2.3.1 Distributed Filesystems

Early methods for distributed data access involved extending the filesystem abstraction to remote directories, thus creating a network file system (NFS) [117], unifying multiple remote storage services into resources that may be accessed through the filesystem. The Andrew distributed filesystem (AFS) [118] extended the model by allowing clients to cache open files, enhancing scalability, and eased management by enabling distributed system administration of autonomous installations. While caching filesystems improve the scalability of the whole system, individual applications may be limited by network latency for small operations, therefore the recently developed BlueFS [94] system employs a speculative technique to promote job progress in the presence of latent responses to predictable operations.

Centralized network file services can become a bottleneck when the number of clients becomes large. Serverless network file systems [13] were built to increase the scalability of network file services by distributing the workload among multiple multi-purpose servers. The ability of desktop workstations to function as powerful local processing and visualization tools *as well as* cache services for cooperating users has been exploited by the Freeloader system [141]. By distributing cache fragments among nearby hosts, the system reduces communication with the centralized data store.

[1] The web structure could also be considered the consequence of hypertext frameworks; this can still be contrasted with cooperative storage.

2.3.2 Replica Systems

Replicating user data improves file survivability and enables parallelizable data services. The pioneering distributed data system Zebra [67] extended the notion and method of disk striping and parity disks [100] to networks of storage services by striping data across multiple servers. The replica storage system was extended to the grid with the advent of the Replica Location Service (RLS) [35], which provides the ability to map logical file names to physical file locations, thus providing a building block for distributed replica storage. Giggle [34] builds on the RLS and other Globus tools to create a complete replica identification, location and movement solution. Important current research areas on storage systems focus on optimal replica placement [103], co-scheduling data movement and job execution [110], and long term data survivability [15].

Another important replica system is the Storage Resource Broker (SRB) [104], which provides the user with an abstraction layer over a variety of underlying storage systems. SRB provides an "all-hosts" replication technique and a user-controlled technique, and manages the metadata catalog in a relational database. SRB is compared to our project, GEMS, in more detail below. Similarly, the the Grid Data Management Pilot (GDMP) [126] combines an object description catalog with a replica table to manage replicated data sets in an application-friendly way.

Many other systems have used a distributed storage fabric to obtain new utility in reliability and performance. A further extension is OceanStore [121], 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 [5].

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* [114], and typically is focused on reducing single points of failure and enabling dynamic construction of operating environments [73]. Additionally, software may be able to automatically adapt to changing conditions by locating a different replica location or utilizing a local caching strategy [71]. 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 [8, 164]. The Globus Heartbeat Monitor [124], for example, employs unreliable failure detectors [31] 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. 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 [165], 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 [153].

2.3.3 Grid-Enabled Services

A primary requirement of grid-enabled data systems is the front-end presented to the storage client, a running job. Often this takes the form of a reference site used for data staging. Another Globus project, GASS [22], exemplifies the data staging model of job submission, using cache management strategies to reduce network bandwidth consumption.

The Condor project has produced multiple data access and movement systems. Data access is abstracted over various protocols and policies in the NeST [18] storage appliance. NeST additionally provides a flexible server configuration, with multiple concurrency models, differential scheduling, and advance storage reservations. Data movement within workflows is promoted with Stork [74], becoming a first-class operation supported by restartability and resource overload prevention.

Additional services of use to running jobs include the ability to effectively employ data locality when performing a computation [140]. GFARM [132], for example, addresses possible collocation opportunities to enhance the I/O performance of a fully POSIX capable grid distributed filesystem. To meet their performance and scalability objectives, efficient replication algorithms are employed to improve data preservation and collocation over the wide area.

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 - distribution and data access - new problems arise as the system takes on several new properties: the system lacks a central authority [39]; 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 abstrac-

tions [133], the ability to resolve data set names in more complex ways, and the ability to use logical, application-specific lookup procedures instead of physical locations.

2.4 Security on the Grid

Grid integration techniques up to this point have focused on combining a globalized authentication scheme with accessible communication protocols. The Globus Security Infrastructure (GSI) [52] has an important presence in modern grid computing. This model stresses the primality of local access control mechanisms, and maps global users to local users. Other systems integrate authentication methods, such as the Generic Authorization and Access-control API (GAA-API) [116]. Open grid access is a hallmark of the Open Grid Services Architecture (OGSA) project [50].

A comprehensive model for large-scale virtual computing system is the Legion system [155]. The data services in Legion are designed to scale, making massive use of parallelism to provide enormous aggregate bandwidth to jobs running in the object-based system. Objects in Legion are responsible for their own access control, which is similar to the model presented here. Legion provides a single system image, which is different from our model in which users specify a subset of the system to use: thus, Legion relies heavily on encryption to protect objects from unauthorized access.

2.5 The Opportunistic Grid

Grids may be assembled by investing in large quantities of new hardware, however, existing development *and* production grids are often assembled *opportunisti-*

cally: that is, by discovering underutilized existing systems and integrating them into a resource network for external use. The Condor [79] system, for example, 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.

At Home [12, 98] 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 [11], which allows volunteers to provide computational resources to a wide array of projects from the physical sciences to game theory.

2.6 Scientific Grids

Real-world grids implemented over the last ten years typically state their applications in advance. They consist of geographically dispersed resources and a corresponding social network of computer specialists and application stakeholders. This structure promotes funding opportunities as research problems in computer science are combined with the natural sciences, and focuses attention on the applications.

2.6.1 Production Grids

As grid computing is a relatively new concept and software systems are still in the development phase, most grids should be considered experimental due to their probationary nature. A few grids have surpassed this status and are reliable

and understood well enough to be used for practical application purposes. These grids are an important part of active research in their target application fields, and are termed *production grids*. Reliable systems like these are still of interest to computer researchers; for example, traces of usage activity may be produced to assist the development of future systems [76].

The Grid2003 Project was the first large scale production grid. It targeted a handful of scientific applications, including high energy physics, astronomy, and astrophysics. Making heavy use of existing tools mentioned above, including Condor, the Virtual Data Toolkit, grid monitoring tools, and others, the fundamental ability to maintain 2500 remote CPUs and run 1000 simultaneous jobs demonstrated the viability of the new architecture. This project lives on as the Open Science Grid [108], which continues to serve the Grid2003 application areas.

University resources have been successfully combined into production grids, an example is the TeraGrid [82]. Combining resources from the University of Illinois, Purdue, the University of Indiana, the University of Texas, and other institutions, the infrastructure has promoted research areas such as computational fluid dynamics [58], computational chemistry [42], and computer systems [150].

Similarly, modern work on the EGEE grid in the European Union offers large scale resources for high energy physics. The current system offers over 41,000 CPUs, all running the same operating system. The complex geography and interconnections among EU grid sites offers a rich environment for grid usage and maintenance strategies [103].

2.6.2 Application-Specific Grids

Some production grids have been designed for a single target application. A storage system designed for the application area of molecular dynamics is BioSimGrid [128]. Centering on a simulator-independent scientific database, BioSimGrid provides tools to perform analysis on its libraries of simulation data. The software architecture combines a standard database with an underlying SRB storage system. Computation in BioSimGrid is centered around application-specific analysis tools which are run on the centralized system. BioSimGrid conducted investigations examining the performance of flat files versus relational databases for biomolecular data [92] and cluster scheduling procedures [156].

Likewise, the National FusionGrid focuses on nuclear fusion research. This project is notable for its combination of experiment and simulation, while producing very large data sets. Constrained by tight access control requirements, progress was made in grid security [28]. Collaboration on running experiments is aided by modern shared display technology [1].

2.7 Grid-Enabling Software

Grid computing is a composite technology that combines a variety of pre-existing technologies. We provide an overview of some software systems used in our work here.

2.7.1 Java

The need for portable network programming became clear in the early 1990's as cheap local area networks and other distributed systems became commonplace. In 1991, Sun developed the Java programming language with the widely publicized

goals of wide portability, however, it was also notable for its convenient APIs for network sockets and remote access to objects and methods [151]. Over time, Java became an extremely common glue language for distributed computing, as exemplified by the Java CoG [144]. Web services have become an important aspect of grid computing with Java, at it provides standardized XML APIs and web service technologies [64].

2.7.2 Relational Databases

The elementary data source on UNIX-oriented computing systems is the filesystem, typically accessed programmatically through the POSIX [101] API, which essentially operates on a byte by byte basis. However, concurrent and distributed programming models, among others, often benefit from additional functionality offered by the data source, such as record by record operations, exclusion among multiple clients, typed data elements, etc. Relational databases offer this functionality and more [36] by offering a tabular structure and stronger consistency rules, and the cost of greater complexity in the data service.

Relational databases aid the development of grid computing in several ways. First, the tabular, typed structure aids scientific programmers organize large data sets in a systematic way. Second, databases provide synchronization for common operations in a simpler way for concurrent access. For example, an *insert record into list* operation in a file would require programmatic synchronization or the use of a centralized controller, where databases automatically ensure the atomicity of this simple operation. Third, databases offer a network-accessible interface independent of any existing network file system, often in a portable, language-flexible way. For example, the PostgreSQL system [66] used in this project is

accessible through APIs for over ten programming languages, including JDBC [96].

Grids may be modeled around the concept of the database, such as BioSim-Grid described above. GridDB [80] provides a general relational abstraction for workflow-like tasks running on grid resources. User requests, formatted as familiar database operations, result in the creation abstract workflows. Results are memoized for efficient access and retrieval. Other grid systems benefit from a database back end, including computing systems and data systems. Quill [111] stores job submission information from the Condor scheduler in a centralized database, creating a valuable provenance system. Quill uses a PostgreSQL back end. The MCAT metadata catalog [106] isolates user metadata from system and replica metadata, providing a new API and scalable implementation for scientific workflows, and may be run on a variety of underlying database systems.

2.7.3 Chirp

New grid controllers rely on the foundation laid by existing systems such as computation and storage services. A critical file server employed in this project is the Chirp file server [137]. This server extends the POSIX API over the network with addition of new performance-enhancing functions [136]. A virtual filesystem adapter called Parrot [135] may be used to access Chirp servers by trapping and transforming system calls made by existing codes, in addition to the ability to access other storage services. Chirp servers are visible for discovery via a catalog service to which they report on startup. The catalog service reported 277 available Chirp services at Notre Dame consisting of 21.0 terabytes of available storage on October 25, 2007.

2.8 Point of Comparison: SRB and GEMS

A popular data source for large scale computing is the Storage Resource Broker (SRB), which, as noted above, has previously been used to store data sets from biomolecular simulation. SRB was motivated by the challenges posed by data-intensive computing [89], used by researchers when attempting to gain new information by reprocessing large amounts of existing data. In the targeted grid setting, this may involve accessing up to a petabyte per day from widely distributed, heterogeneous data sources. The noteworthy performance aspect of this computing paradigm is that data movement time is the dominant response time constraint.

To enable to aggregation of large numbers of storage resources, SRB starts by implementing the Grid Brick [104] abstraction, defined as a commodity storage appliance providing standard data services. Each brick represents a three-level storage abstraction service [152] tied to underlying storage systems such as databases, filesystems, and tertiary archives. The abstraction service consists of a top-level communication layer accessible to client requests, a high-level logical data name layer which relies on the MCAT metadata catalog [106] to translate logical names into physical representations, and an underlying driver system to provide the required POSIX-like operations on the actual underlying data system. These basic features are diagrammed in Figure 2.3.

The SRB installations can then be used to provide rich functionality for scientific archives [105]. User files may be logically organized into containers, protected and accessed *via* wide area authentication and authorization techniques, and tagged with meaningful metadata. Data files may be replicated to improve fault tolerance or cached near to a processing unit for improved performance. This

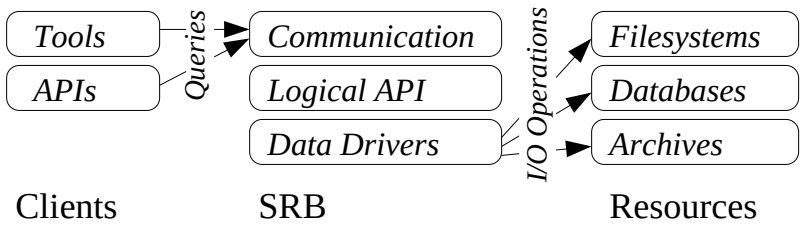


Figure 2.3. SRB software overview.

functionality is then presentable to the grid as it implements a virtual organization [104] of virtualized data [88] integrated with ownership and access mechanisms. Ultimately, SRB resources may then be easily federated [107] to provide the required performance and functionality through the construction and integration of SRB zones, which provide metadata and data replication.

SRB shares with GEMS the goal of providing high performance, flexible, abstract data services to scientific users in a grid environment. However, the assumptions and contributions made by each system differ. SRB intends to operate on a relatively small number of relatively large disks, improving client access and administrative functionality [104]. SRB zones may then be federated in centrally defined structures [107] to provide a variety of required functionality. Access control is managed through centrally registered user groups and, optionally, tickets, a capability system. SRB has thus been able to support a large number of scientific projects [105] by managing hierarchical collections of user data.

Contrarily, GEMS intends to opportunistically operate on a relatively large number of relatively small disks, improving the utility of possibly volatile existing disk space for distributed computation and dynamically managing replicated

tertiary archives. The underlying file servers require single command line configuration to start up and integrate with GEMS, which then controls them using semi-autonomic techniques to provide robust data survivability and services. By dropping in a GEMS controller, individual users may structure their own data grids, specifying replica layout topologies and access control lists in a per-record fashion. The GEMS replica control system thus contributes to data survivability in difficult environments, and its parameterized, tabular metadata system provides scientifically friendly data organization, primarily serving molecular dynamics workloads.

CHAPTER 3

SCIENTIFIC REPOSITORIES

Computational research involves the creation of data sets containing the results of scientific computation or instrumental data. The storage location for a collection of such records is commonly called a *scientific repository*, and in this chapter we discuss the results of our research in this area. In this chapter, we will describe our approach to the construction of a scientific repository on unreliable resources, focusing the tools and data model presented to the user. Our research in this area was driven in part by the development of the Grid-Enabled Molecular Simulation (GEMS) system.

The concept of the GEMS scientific repository was developed in 2005 [158].

3.1 Scientific Storage

Scientific data often has certain properties that differentiate it from other data:

1. It is independent of the technical nature of the physical computer system;
2. It is typically highly structured.

A common need in scientific computation is simply a non-volatile location to store data for future access. However, there is a difference between scientific *secondary storage* and *tertiary storage*. Secondary storage, such as a local hard drive

is used to quickly store data over the short term, such as over system restarts or virtual memory accesses. Tertiary storage is typically intended to store important data for the long term safely and economically, often relying on tape or optical disk. Large scientific archives have tertiary characteristics, emphasizing the long term safe storage of data sets, possibly at the cost of slower access performance.

This duality is problematic for high performance applications that access or create large scale data sets at high rates. Complex cache systems such as the Raster Data Management database (RasDaMan) [112] improve the apparent performance by interposing online hard disks between the client software and the essentially offline back end.

Offline storage can then be managed for its own sake. For example, a storage management system called Lots Of Copies Keeps Stuff Safe (LOCKSS) [83] emphasizes the ability to ensure data integrity over time periods on the order of hundreds of years. Data access rates are not a constraint to the system design, and in fact throwaway computations are performed to create inertia and resistance to system change.

3.2 The GEMS System

This section provides a high level view of major GEMS features as a preface for the detailed development in later chapters.

3.2.1 Overview

Biomolecular simulations produce more output data than can be managed effectively by traditional computing systems. Researchers need distributed systems that allow the pooling of resources, the sharing of simulation data, and the reli-

able publication of both tentative and final results. To address this need, we have designed GEMS, a system that enables biomolecular researchers to store, search, and share large scale simulation data. GEMS offers a technical solution to data storage problems such as metadata management and file replication, and also addresses the social dynamic that results from the opportunistic use of volunteered data storage resources. Collaborating storage providers may share resources with their collaborators, but must be protected from unchecked data producers who may replicate data unnecessarily until it fills all available space. To mitigate the risks to all stakeholders, GEMS allows both storage providers and data producers to state and enforce policies on the consumption of storage and the replication of data. By taking advantage of known properties of simulation data, the system is able to distinguish between high value final results that must be preserved and low value intermediate results that can be deleted and regenerated if necessary.

3.2.2 Discussion

For a large number of scientific disciplines, grid computing offers the capability for inexpensive computing and storage on scales previously reserved for the domain of supercomputing. Hence, researchers involved in simulation-driven scientific studies such as chemistry, physics, and biology have been naturally drawn to the promise of cheap, large scale computing.

Producers of such large and complex data need system support for managing their experimental work. A single user of ProtoMol can generate so many variations on the same simulation that a database-like index is needed to simply keep track of the work already accomplished. Since the total amount of data generated can easily exceed the storage available in any single device, researchers need a sys-

tem that can be expanded or reconfigured while running. Allied researchers often explore simulations in related areas and would like to be able to index and share results with each other. Many simulation modes are iterative; computational work can be saved if intermediate outputs of older simulations can be recovered and re-used. Because of the high value of some simulations and the potential for data loss in any computing environment, users would like to replicate their data both in the local area for performance and across the wide area as insurance against disaster.

Designed to meet these data-intensive demands, GEMS was developed as a wide area distributed system for managing the *storage*, *searching*, and *sharing* needs of collaborating researchers. GEMS allows both storage providers and data owners to exercise control over system policies. Each owner of a storage device sets a policy dictating who may use it and how much space may be consumed. Likewise, each data owner is able to dictate the location and replication factor of data placed into the system. The design and architecture of GEMS is such that each component of the system includes a strong policy component that defends the interests of its owner. A prototype of GEMS is currently operating at the University of Notre Dame. Through experimental studies, we have demonstrated how GEMS is able to deal with changing constraints in a dynamic system. This has yielded several insights into the behavior of a distributed replica storage system.

3.2.3 Architecture

Figure 3.1 shows the major components of the GEMS architecture that include *storage servers*, *catalog servers*, and *database servers*, as shown in Figure 3.1. The GEMS process begins when the user submits data for storage. With each storage

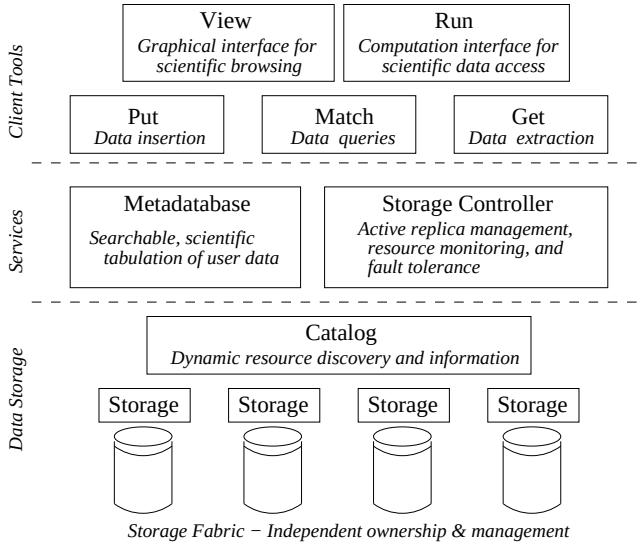


Figure 3.1. GEMS architecture.

placement, the user includes metadata for both the file itself and indexing in the database. In turn, the database server determines where to place the data based on resource discovery information from the catalog server. Storage servers are required to report their presence to a local catalog server for discovery.

Storage servers thus double as short-term secondary storage disks for running jobs, but upon data committal, are pooled into a managed tertiary storage system that emphasizes long term reliability. *GEMS takes unreliable, volunteered remote disks and creates both a parallel, localized data system* and *a highly distributed replica archive.* This is delivered by controlling the metadata and data flow through the system, considered in this chapter, as well as a potentially volatile resource network, considered in the next chapter.

Functionality provided by GEMS can be categorized into fault tolerance support through replication, dependent computing, and virtualized data. First, the primary function of GEMS is to provide appropriate replication of data placed

onto the grid. While one can, with reasonable reliability, replicate data on storage entities owned by the user, no guarantee is provided when the storage of other groups is used. GEMS emphasizes aggressive replication in order to tolerate multiple modes of failures. The user needs only to specify the requested redundancy levels leaving GEMS to manage how the data is placed and replicated.

GEMS additionally offers the ability to conduct dependent computing. For instance, suppose that a given user wishes to conduct a new algorithm for protein docking from a previous set of runs on a remote site. Rather than downloading all files for local computation, a job is submitted remotely, utilizing computational *and* data resources in the network. The resulting new data is then turned over to the GEMS software for replication.

Finally, GEMS expands upon the virtual data concept of Chimera [54] when offering public archival functionality. GEMS data is virtualized in that it may or may not exist for immediate consumption. Since GEMS is tuned toward certain explorations in molecular dynamics and protein conformations, GEMS users can take advantage of the deterministic nature of many biomolecular simulations to provide auto-regeneration of missing points without re-running the entire simulation.

3.2.4 A General-Purpose Metadata Catalog

Scientific repositories create a browseable front end for user-labeled data sets stored in a scalable back end. With the primary intent of creating a searchable repository, GEMS presents a database-like abstraction over the file sets stored among the storage providers. User applications could access these existing volunteer services directly, but since they are independently managed, their presence

in the system is unreliable. Technically, the GEMS system creates a centralized parameter sweep database of application specific metadata backed up by a churn-aware file replica management back end. By combining the metadatabase with the management system, a quickly deployable tuple space *and* a survivable and parallelizable data system is therefore implemented. The resulting system is thus a merger of repository tags and parameter sweep entries, which programmatically represents a shared tuple space in which running jobs may communicate, with linkage to a replica location service.

Each entry e in the GEMS metadatabase contains a tuple of parameter tags and values, formatted as m equations: $e.\text{tuple} = \{p_i = v_i, i = 1..m\}$. These entries are called *configs*, as they traditionally represented the parameter information found in a configuration file for a single simulator run. A query set to find matching configs may be formed by creating a similar tuple $\{q_i \mathbf{R}_i w_i, i = 1..n\}$, where $\mathbf{R}_i$ is some relation. The metadatabase responds to a query set by returning the set $\{e : \forall q_i \exists p_j : p_j = q_i \wedge v_j \mathbf{R}_i w_i\}$. Thus range queries over the metadatabase may be simply constructed, for example, a user may request "all entries of type *simulation* with temperature above 300." e also contains a file management data structure e.files, which may be used to obtain file information and replica locations.

3.2.5 Comparison with Existing Systems

GEMS has commonality with distributed file systems, databases, and peer-to-peer sharing systems. Users of these systems have different expectations when it comes to error states, and internally, faults are treated differently. Users of file systems and databases generally expect all-or-nothing responses. File system users expect that if one file on their machine is present and correct, then all will

be be there. The locality assumption comes from the standard assumption that when a physical storage device fails, all of its data is permanently lost, and that partial failures are handled by the operating system and not exposed to the user. This carries over into expectations of network file systems, which often attempt to emulate the behavior of a local file system.

While GEMS stores whole files and their directory information, it does not attempt to provide file system behavior. Files that are expected to be found in the same directory may be found on different hosts, destroying any assumptions about locality in the delivery of file sets. This affects the design of GEMS clients, which have utilize lists of replica locations. While GEMS does not expose internal faults to the user, the user should be aware of the actions that GEMS may take in the case of data loss, which include contacting various remote hosts.

NFS [117] is a standard point of comparison for file retrieval latency and API semantics for remote storage. GEMS is not designed to compete with a finely tuned cluster storage system, and since it is not a filesystem, it does not attempt to mimic any system's API semantics. GEMS does function in an RPC fashion from the client's perspective, and does not maintain important volatile state. AFS [69] provides remote file access as well as some additional fault handling functionality. Read-only replicas of stored data may be easily configured, and roll back to previous data is built in. However, AFS servers must maintain cache consistency for their clients. GEMS does not require complex caching strategies because of its write-once characteristic, greatly reducing the server's responsibilities.

Relational database systems similarly are expected to provide all-or-nothing responses. GEMS, however, may only be able to partially fulfill a query and will resort to partial delivery. GEMS does provide a metadatabase as described above,

and normal database user expectations apply. When querying for output data files, the returned result is a fault-tolerant plan on how to retrieve the requested information from remote storage devices.

The most obvious point of comparison for a redundant data storage system is a local RAID [100] installation, or a network-based RAID, such as Zebra [67]. In a typical hardware RAID setting, faults may be detected as blocks are read from the device, where in GEMS, errors are actively probed by a server component. Zebra detects faults over the network, which is a technically difficult observation to make. GEMS and RAID both benefit from hot-pluggable hardware. As devices are added, both systems can discover and begin utilizing new storage. When a new server is added to GEMS, it may immediately begin receiving data from a client, just as some RAID types may immediately use a new disk. Removing a disk from either may be performed while the system is running. GEMS offers some additional benefits in that it has knowledge of the importance and replication status of the stored data, so upon replica loss, replication does not necessarily immediately consume network bandwidth for all lost data.

3.3 Prototype

We have constructed a prototype implementation of GEMS using the following tools. The storage server used is the Chirp [133] personal file server. Each Chirp server periodically sends a message with its status and available space via UDP to a catalog server. The catalog server makes the state of a set of storage servers available via HTTP in the form of XML or Condor's ClassAds [109]. The database server, called GEMSd, is a custom Java server that accepts client connections and stores the state and location of each file in a PostgreSQL database. A variety of

client tools allow users to insert, use, and query the system for data.

3.3.1 Client Tools

Users are given client tools to perform three basic operations: *GEMSput*, *GEMSmatch*, and *GEMSget*, as well as a higher-level tool called *GEMSrun*. GEMSput allows the user to specify a completed simulation, along with its input and output data files and appropriate metadata, for insertion into the system. GEMSmatch allows the user to locate existing records that match criteria specified by the parameter framework described above. GEMSmatch contacts the database server, performs a query, and then returns a list of locations where the matching data may be accessed directly on a storage server. GEMSget simply downloads required GEMS files or whole data sets by using replica location information; nearby copies are tried first, followed by all possible locations. Examples of these command-line operations are shown in Figure 3.2.

GEMSrun provides a comprehensive interface for user jobs that intend to work with GEMS data. The user process is treated as a consumer and producer of GEMS data, thus requested data sources are linked in as input files and output data sinks are created for new files. The client makes use of the Parrot adapter externally, orchestrating data locations at a high level. GEMSrun is described in more detail in Section 5.2.

A graphical user tool to enable web browser-like searches, downloads, and uploads with GEMS has been constructed in the GEMSview client, as shown in Figure 3.3. GEMSview allows searches based on keywords, arbitrary key/value pairs, or an identifying config key. Users may also submit data to the system through the GUI by specifying key/value pairs and a list of file names for upload.

```
①  > GEMSput name=sorin app=sim seed=149
           --file sim.in --file sim.out --key
   4189
②  > GEMSmatch name=sorin app=sim --keys
   9980
   4189
   3415
③  > GEMSmatch --config 4189 --params --files
   4189
   name = sorin
   app  = sim
   seed = 149
   sim.in  1.3 KB (4/5)
   sim.out 421 MB (3/3)
④  > GEMSmatch --config 4189 --locate sim.out --hosts
   myhost.mycampus.edu
   host13.nearby.edu
   host41.faraway.edu
⑤  > GEMSmatch --locate /4189/sim.out
   /chirp/myhost.mycampus.edu/GEMS/4189/sim.out
```

Figure 3.2. Example usage of GEMS client tools.
① First, the user submits files and parameter metadata, obtaining a config key (4189). ② The user can obtain this and other configs by specifying a query that returns a multiple-entry result set. ③ GEMSmatch may be used to examine a variety of config details, including file sizes and replica counts (actual/requested). ④ Replica servers of a given file may be obtained and ordered with respect to the cluster topology (Section 5.5). ⑤ The nearest available replica may be obtained in a Parrot-compatible path by resolving the abstract path (Section 5.2.3).

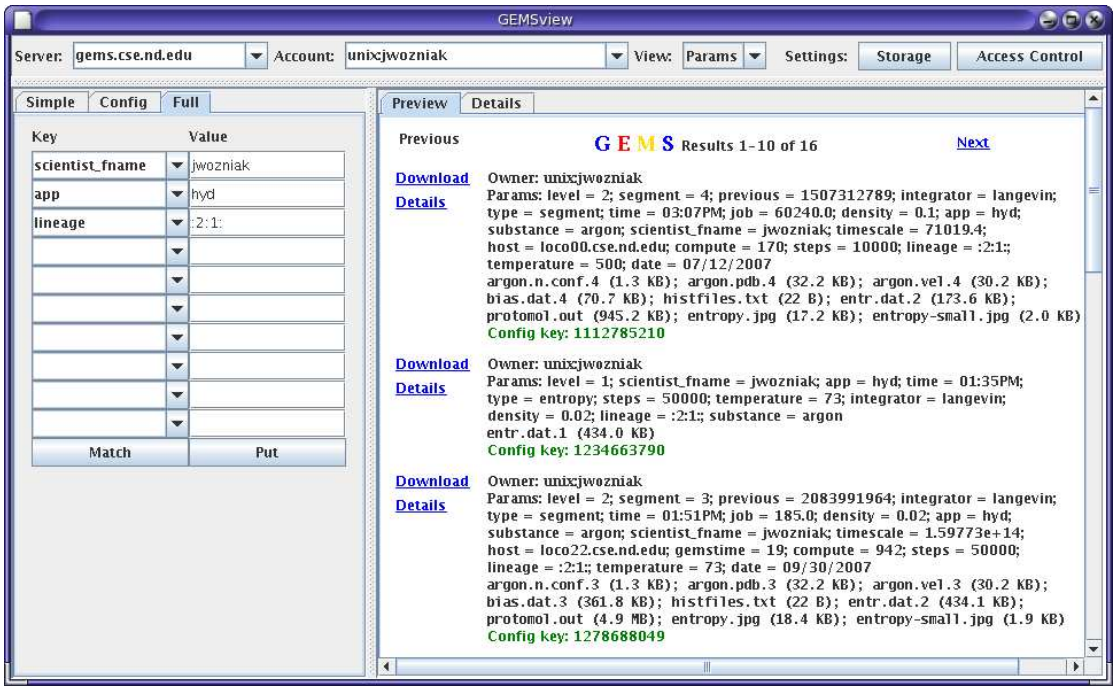


Figure 3.3. The GEMSview browser.

GEMSview users may select their GEMS server and desired authentication method (Section 5.6.6) from the top toolbar. Using the left hand pane, a search method is selected; in this case, a set of hyperdynamics segments (Section 3.4.2) is selected. The right hand results pane shows a preview of the matching config results including their owners, parameters, files, and identifying keys. Configs may be viewed in more detail or downloaded.

All operations are implemented atop the GEMS API that support the command-line clients.

3.3.2 Database Server

The GEMSd database server is the hub of GEMS. It is responsible for serving users, managing metadata, and periodically scanning storage servers for problems. This includes metadata about the executable, parameters, and input and output files. In addition, each file has metadata describing its size, desired replication

level, and sites where it may be found. This information is used by the client when retrieving data for another GEMSmatch or GEMSrun operation and also by the server when checking the status of storage servers.

The database server is responsible for maintaining the replication count of the data that it tracks. Periodically, the database server scans its database and then probes the necessary storage servers to make sure that the expected files are still there. The files might be unavailable for several reasons: the storage server may have failed, the resource owner might have evicted the files, or the network may be temporarily partitioned. Regardless of the underlying reason, the database server views all these failure modes as a loss of data. Data survivability techniques are then applied as detailed in Chapter 4. The database server is also responsible for the opposite task of cleaning up unidentifiable data that exists on Chirp servers in the space allocated to GEMS, in a garbage collection process. For example, failed GEMSput insertion operations could leave broken files for which GEMS has no legitimate information; these are iteratively deleted over time.

GEMS incorporates disk utilization management as a fundamental feature. The GEMS server is configured at startup with the permitted storage size. Although replication counts are requested per file by GEMSput, the GEMSd server may allocate less space if disk utilization runs too high. Statistics regarding current utilization are obtained by comparing disk utilization information in the Chirp catalog, as well as information in the database. The replication and garbage collection components consult the system utilization state before making changes to the system, and make adjustments to a file's replication count as they progress.

3.3.3 Distributed Access Control

A distributed storage system relies heavily on the contributions of storage space and access control offered by the individual servers. The storage servers are expected to respond to requests for storage, retrieval, and replica management, as well as enforce access control as specified by the central server and the storage owner. Within this framework, there are several options. If the data is very public, each server could provide read access to all its data, with write access and replica control strictly managed by the central authority. Extremely sensitive data sets could be managed exclusively by the central server: the storage servers would only listen to the central authority, so all client data transfers would have to hop through the central machine. While the first approach offers more access than many researchers would like, the second is obtuse and would exhaust the central machine.

In GEMS, control authority is delegated across the Chirp servers, but managed by the GEMSd server, as shown in Figure 3.4. This supports our objective of protecting storage providers, while allowing for the automatic replica and metadata management provided by GEMSd. Storage providers advertise storage space to the system through the Chirp Catalog, which is read by GEMSd as discussed above. As an additional step, the server must allow GEMSd administrative access to a `/GEMS` directory, otherwise the server will be ignored. This allows GEMSd to allocate storage for new data and maintain appropriate permissions on the filesets during replication.

In this system, the Chirp access control list (ACL) for a given fileset is specified at the time GEMSput is used. GEMS uses the Chirp form `method:name` for authentication, which allows a variety of authentication methods including Unix,

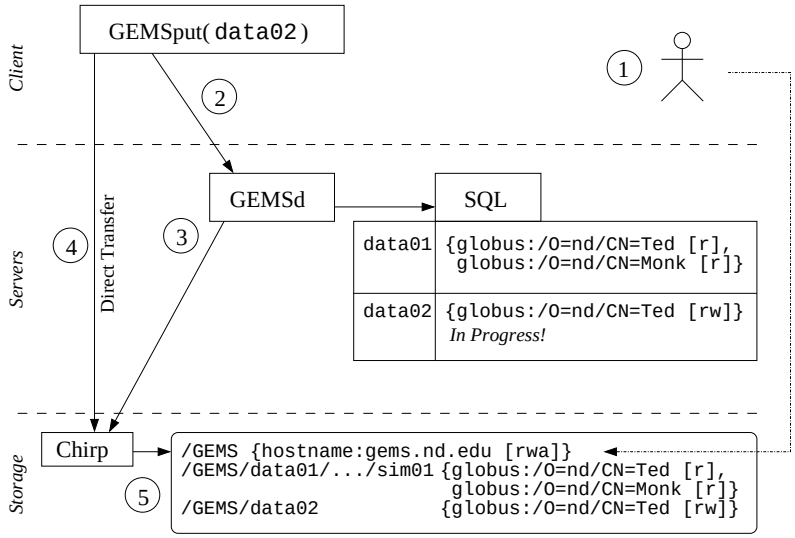


Figure 3.4. Access control example.

This example shows how GEMS protects data within the constraints of all the participants. ① The owner of a file server gives the GEMS server access to administer a directory. ② The owner of `data02` states the ACL to be used for protecting the new data set. ③ GEMSd creates a new directory on the file server and sets the ACL for that data set. ④ Direct data transfer to the file server begins. ⑤ As shown for `data01`, existing data is directly readable by certain users.

hostname, Kerberos, and Globus. A GEMSput user is authenticated by a rendezvous protocol with a trusted Chirp server. Upon success, the ACL is stored with the simulation metadata. Then GEMSd creates a writeable directory on a satisfactory Chirp server for the client, and the data is uploaded from the client. When the client notifies GEMSd that the transfer is complete, GEMSd locks out the write permissions on the directory, activates the entry in the metadatabase, and sets read permissions for the appropriate users. File replication for this fileset is performed with the constraint that the Chirp servers in question accept the authentication method given by the corresponding ACL.

3.3.4 Scientific Benefits

A primary target application for the GEMS system was molecular dynamics, which models a molecular system as a function of time based on integration of equations of motion and interacting forces. The running time of these simulations are typically dominated by the force calculation between the various atoms in the simulation. ProtoMol is a generic, object oriented molecular dynamics simulator [84], that is utilized to assist in the detection of new conformations of proteins[1]. Finding new conformations helps in understanding the functions of proteins in living tissue and aids in current biochemical areas of research such as pharmaceuticals.

An example ProtoMol conformational sampling computation proceeds as follows. First, the researcher retrieves appropriate positional, relational, and physical parameter files for the protein of interest. The researcher will take the initial conformation in a selected environment, apply a random variation, and allow

[1] A conformation is an energetically minimal geometric configuration that is significantly different from other known conformations.

molecular dynamics to proceed. If the new state of the system meets certain physical properties, then it may be declared a new conformation. The new set of conformation ensembles can then be further analyzed *via* additional computation.

As the simulation runs, large intermediate files are created regarding the state of the system in addition to a final set of unique conformation candidates. Researchers are interested in not only the final result but also intermediate results as the simulation progresses in a deterministic manner. From the operation of ProtoMol and other biomolecular simulations, we note several important characteristics that GEMS addresses:

- Data browseability;
- Reproducibility; and
- Data valuation and prioritization.

GEMS is intended to enable evolving methods for biomolecular research, thus, the data sets must be easy to share, discover, and access by users concerned only with data descriptions. Additionally, these data sets should be reproducible given the appropriate descriptors, adding to survivability in the case of loss as well as simple scientific provenance information.

A notable feature of GEMS is that it allows to user to specify critical data files through metadata and hence prioritize the target redundancy levels for the system. Rather than simply striving towards a uniform redundancy level, the importance of the data can be taken into account when reacting to inevitable subsystem failures. GEMS will attempt to maximize redundancy levels of critical data but will yield space on overloaded disks provided that minimum redundancy is preserved on critical data.

Finally, the incorporation of XML metadata for both placement and searching allows GEMS to build on existing work and arbitrarily extend search sophistication as necessary. A critical aspect in the development of ProtoMol is the ability to improve simulation speed. Hence, it would often be necessary to insert arbitrary tags to denote algorithm approaches and internal notes. In addition, GEMS is also capable of including already well-defined characteristics of results in the manner of BioSimGrid [93].

3.3.5 Toolset Summary

The GEMS toolset design meets the needs of researchers working in the computationally exhaustive and data centric field of biomolecular simulation. The suite of programs provides a new method for users to find, use, and store large data files. This is accomplished by implementing a novel distributed data storage model which combines autonomous storage resources, an appropriate metadata specification, automatic storage allocation and replication policies, and an interface for distributed computation. The prototype implementation has been shown suitably functional to demonstrate the model's potential as a production system.

3.4 Applications

GEMS has been used as a scientific repository in two major projects over the last four years: transition path sampling and hyperdynamics. The work on transition path sampling was originally presented in Spring 2007 [25]. The work on hyperdynamics was carried out over Summer and Fall 2007.

3.4.1 Transition Path Sampling

The successful implementation of the transition path sampling (TPS) method relies on the computation of numerous trials - trajectories - executed in parallel. Transition paths are rare events that occur when a trajectory initiated at a known point in phase space passes nearby to a target point. The vast majority of trials fail, but those that are successful are used to obtain information about the state transition of a molecule.

In an experiment run atop the GEMS system, 2000 trajectories were attempted on the WW protein domain of the PIN1 enzyme, employing ProtoMol simulations. At 65 hours per simulation, the total CPU time consumed totals 130,000 hours on a Pentium 4 system, executed on heterogeneous Condor resources. The output required 1.6 million files, which were replicated to improve survivability. The total storage was over 1 TB for this run. After execution, postprocessing of the data files was performed in another distributed and data-intensive process. Performance analysis for the postprocessing is broken down thoroughly in Section 5.2.4.2.

During the PIN1 experiment, multiple particular strengths of the GEMS system were identified.

- Parameterized file management;
- Distributed storage access;
- Workflow management;
- Secure authentication;
- Cooperation and dynamism; and

TABLE 3.1

OUTPUT BANDWIDTH FOR WW DOMAIN SIMULATION

Output Resolution	**Number of concurrent simulations**		
	64	128	256
1000fs	0.05 MB/s	0.1 MB/s	0.21 MB/s
100fs	0.53 MB/s	1.06 MB/s	2.12 MB/s
10fs	5.3 MB/s	10.6 MB/s	21.2 MB/s

- High throughput data generation.

File management and the metadata catalog enabled the large number of files to be distributed among a large network of over 100 file servers to be automated, allowing for application-specific workflow programming. Security and cooperation were required to maintain the integrity of the computation results on a large-scale multiple user, multiple resource provider system. Dynamism is exemplified by the ability of the system to continue operation as resources come and go, a common occurrence in such large runs.

Additionally, the massive available parallelism of the underlying storage network enables high frequency observation data output from the parallel tasks, improving the scientific quality of the results. As shown in Table 3.1, high output frequencies for large numbers of jobs writing to independent files could overwhelm a typical file server, even in a non-data-intensive case.

3.4.2 Hyperdynamics

Another algorithm commonly used to investigate transitions from one metastable state to another is hyperdynamics. Hyperdynamics does not require prior special knowledge of the system, instead, the method uses statistics obtained from simulation progress and uses them to modify the potential energy surface in a controlled way. This is represented by a bias potential that lifts the simulation out from a stable region and into saddle regions of interest.

Critically, applying additional bias levels poses hazards. If the bias is too high within a metastable region, the system will not reach a local equilibrium in the biased trajectory and the timescale of the trajectory will be incorrect. On the other hand, if we apply fewer bias levels, a longer simulation will be required for the system to move from one metastable region to another.

As a demonstration of the difficulty involved in this process, Figure 3.5 diagrams the timescale performance as a function of branch location on the time axis. Thus, the researcher controlling the simulation must monitor the histograms above for error and smoothness while selecting branch points that maximize simulation efficiency in terms of timescale. Since the application of additional bias levels increases computation time per segment, we employ a *performance ratio* (R) that indicates efficiency: work done measured by timescale per unit of CPU time, or

$$R = \frac{\texttt{timescale (simulated femtoseconds)}}{\texttt{CPU time (real seconds)}}. \tag{3.1}$$

This simulation on a 400 atom Argon system was executed on Intel processors running at 1.4 GHz. Langevin dynamics [122] were performed with no bias for 50 picoseconds (ps), in 10 segments of 5ps each, with temperature $T = 300\,K$. One segment at bias level 1 was performed. After that, the simulation was carried out

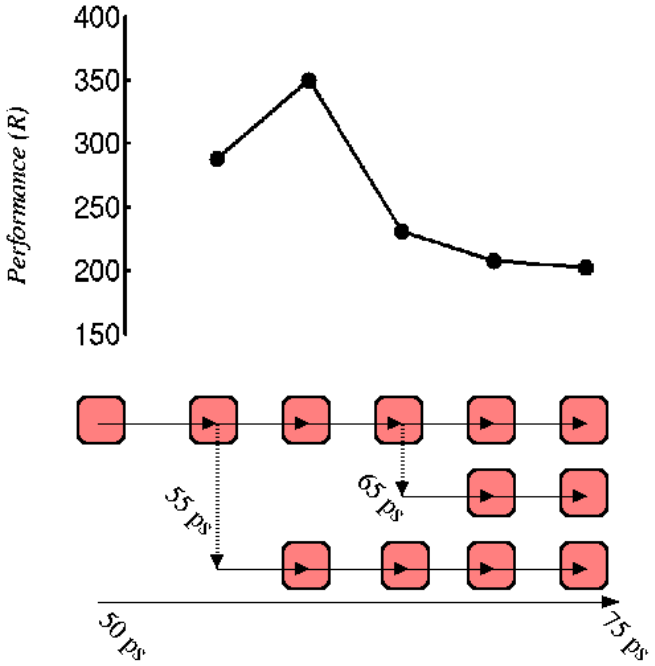


Figure 3.5. Performance ratio R (Equation 3.1).

The ratio is plotted above a diagram indicating two illustrated example branches, 55ps and 65ps. R is plotted as a function of branch time (ps).

after branching to bias level 2 after each of 4 possible segments. The performance ratio was obtained for each case, including the case where the simulation stayed at level 1 for the entire experiment. As shown in Figure 3.5, branching from level 1 to level 2 at 60ps results in the best performance, a total timescale of 282,851. This may be compared to the timescale obtained without the use of hyperdynamics: 75,000.

In summary, selecting optimal branch points for an arbitrary number of bias levels is not possible to do in advance. In our present application, viewing output on the fly is not enough. The timescale and entropy histogram indicate if the simulation has progressed to the point at which applying another bias potential level would be beneficial and free from serious error. As the algorithm under study is new and there is no analytical method to make this determination, tools to enable *ad hoc* exploration of the parameter space are required. Managing the complexity of such an exploration is covered in more detail in Section 5.3.

Each execution of the ProtoMol molecular simulator [84] is considered a GEMS transformation mapping input files to output files, which altogether make up a segment. Each segment is stored as a GEMS *config* and is tagged with user-defined parameters, formatted as parameter sweep-compatible variable names and values. While the underlying files are managed, replicated and migrated due to the potential of host churn, the GEMS server maintains the relatively small amount of metadata required to perform the operations needed by a scientific filesystem. Most importantly, configs may be queried for existence, retrieved in full, used as input to new jobs via a virtual filesystem, or created by specifying files and tags.

Configs conflate concepts such as workflow node, application checkpoint, and staged input data. For example, in this application, a complete segment stored as

a config primarily represents the completion of a workflow element. Since the data files are not directly returned to the user but are asynchronously replicated over the storage network, they form a layer of potential pre-staged computation sites. The system supports the use of network topology information to optimize replica access and job placement. Additionally, these segments may act as checkpoints to restart linear jobs, such as in a simple but long molecular dynamics trajectory.

Configs may be viewed graphically with simple custom GUIs. As an example, Figure 3.6 shows a combination of parameter tags from the central database as well as a view of plotted data files. These distributed hyperdynamics executions were followed remotely by Matlab postprocessing, generating and storing simulation output and plots on the remote cooperative system. Parameters include the execution *host* and output *timescale*, a hyperdynamics-specific indicator of work performed and the benefit of the new algorithm. This client also uses parameter tags to arrange the tiles in the frame, and pulls image files from the storage network to provide a high level view of workflow progress. This provides an intuitive, geometric feel to the workflow state as opposed to the flat view offered by the generic GEMSview browser.

The database-like abstraction created by the system enables typical table-oriented queries to be performed, such as "How many jobs at temperature 300 Kelvin have completed?", or "Show me the CPU performance for all Argon simulations on the Helios cluster." The parameter queries may cover scientific and system data or a combination as discussed below. Since the system also stores the actual data files used by the simulator on the cooperative storage/computation network, it enables and orchestrates high bandwidth, parallel access to large data files for existing codes. This combination thus exceeds the utility offered

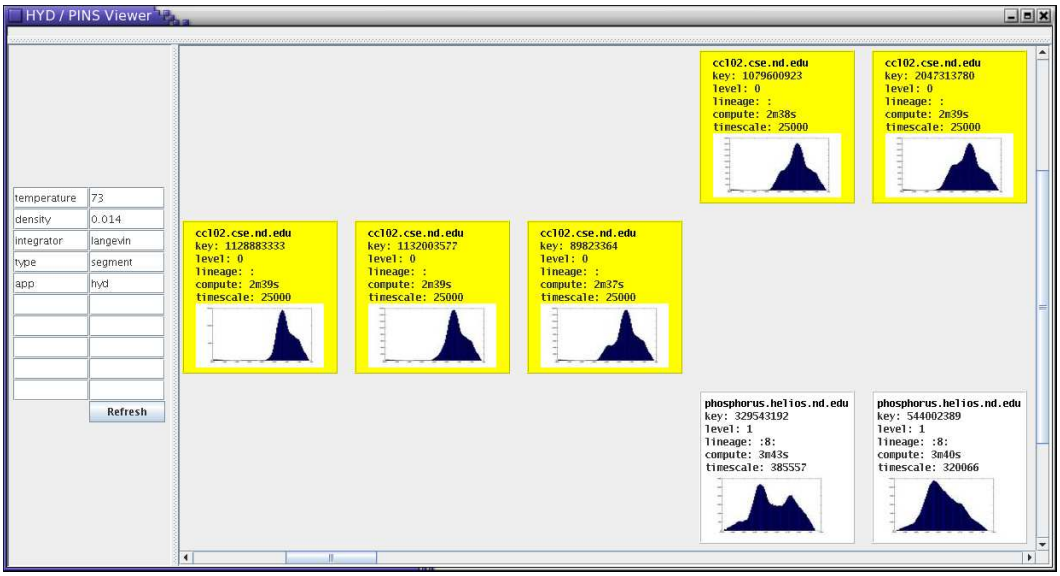


Figure 3.6. Graphical user interface representation of hyperdynamics.

The tool automates simulation branch restarts by setting up a hyperdynamics bias. Thumbnails show that entropy distributions in segment 3 and 4, level 0 are similar. Therefore, one can branch from segment 3 to level 1. At level 1, the bias will push the system away from the conformational space already traversed in level 0.

by systems-specific tools or unified SQL databases.

A hyperdynamics simulation trajectory can be viewed as a collection of smaller segments of trajectories corresponding to different bias levels. The user needs to decide when it is appropriate to branch to the next level by visually inspecting the distribution of entropies S. A branch is appropriate from one level to the next when the distribution of entropies (S) reaches a steady state in one area of the entropy space of the system. Our client-side tool helps the researcher make these decisions effectively and finish simulations faster. Figure 3.6 shows an example where branching is performed after running 3 short segments in level 0. The shape of the distribution of S does not change significantly from segment 3 to segment 4. This is an indication that the system might be trapped in an entropy-dominated region. Branching to level 1 biases the system out of the region already visited. The system will still visit the biased region and reach local equilibrium. However, part of the trajectory inside the biased region is coarse-grained. The tool allows researchers to decide from which segment to branch by clicking on the thumbnails. One can also look at an enlarged plot of the distribution or perform job control operations.

As an informal demonstration of the effectiveness of this approach, a user attempted using the GEMS framework for interactive hyperdynamics. The user based parameter exploration choices based on the knowledge of hyperdynamics properties and graphical feedback from the client tools in an attempt to observe the condensation of a droplet from a gas. A 1000 atom gas phase Argon system was simulated at $T = 73$ Kelvin, using a Langevin impulse integrator [122]. The droplet was observed at segment 17, level 3. The total number of segments created was 70, totalling 6.2 CPU hours on 5 hosts. However, the turnaround time was

TABLE 3.2

INTERACTIVE HYPERDYNAMICS RESULTS

Method	**Total (hours)**	**Turnaround (hours)**
Single long run	1.8*	1.8*
Full search	5800*	1.4*
Interactive search	6.2	1.4

*Result extrapolated from experiment.

only 1.4 hours.

By extrapolating from the results achieved in this experiment, we can estimate the cost of performing the experiment with another method. Results are tabulated in Table 3.2.

- Employing a single long ProtoMol run would have required running a single job with traditional dynamics to achieve the target timescale at which the droplet was observed. This would result in 1.8 hours of single processor computation.

- Additionally, a full search of all pathways of length 17 could have been performed, one of which would have resulted in the observed droplet. This would rely on the implementation of perfect parallelism on 2^{16} processors for the final leaves, and ignores the scalability issues mentioned in the previous sections. This idealized computation would result in total resource consumption of 5800 CPU hours. However, since the path length is still 17, the turnaround time is again only 1.4 hours.

3.5 Performance

The data-driven grid repository models a the computational environment as a landscape of data sources and sinks of interest, laid out as a potential workspace. Mobile jobs, submitted by data-aware schedulers, interact with the *data landscape*, consuming existing data objects and storing output data records through unitary operations called *transforms* [54]. Users of a opportunistic grid of arbitrary size may have *temporary* access to a large variety of existing storage resources, accessible over standard APIs. However, individual users would typically have difficulty organizing these sites into usable categories, ensuring data survivability in the presence of churn, and efficiently using the resources as data sources and sinks. Thus our work emphasizes usability and control in repository data management, combined with performance.

3.5.1 Programmatic Repository Access

Compiled user codes are unable to accommodate the variety of new and experimental APIs so new systems must provide tools to connect them to new services. Our approach to this problem builds on previous work that creates abstractions within UNIX-like structures. Since GEMS is primarily a grid controller, it does not deal with these structures directly but provides tools to orchestrate the connections. Additionally, our solutions reinforce the generally held conception that coarse grained, suboptimal performance is achievable by easily portable software while better tools meet portability challenges. In these examples, we consider data access methods for scientific users launching large batches of jobs which run on a compute network in which an opportunistic storage network is embedded.

- *G/P*

  GEMS provides a data repository for running scripts through traditional `get` and `put` operations (GEMSget and GEMSput). Interleaving these within script operations results in a data staging model similar to that used by Condor and other systems. However, since data records and replicas are stored among the compute sites, there is great potential for data parallelism and locality. This solution is highly portable as it relies only on the GEMS client toolkit, a Java implementation, but relies on significant local disk usage and data copying.

- *PIPE*

  Second, GEMS creates data *sources* and *sinks* that may be targeted by input and output streams. For example, users may reserve larger amounts of space within the GEMS system than is available on a local client machine, and then use a reference to the reserved location as a streamed output target, enabling the creation of large archives without large local space or data copying. This method offers relatively high performance and an intermediate level of portability and complexity - it again uses only the GEMS client tools but requires interprocess communication connected by the user through UNIX pipes or a similar technique.

- *VFS*

  Third, GEMS offers client tools to manage file naming conventions used by the Parrot virtual filesystem. This user space adapter provides system call translation to reformulate ordinary data access methods into distributed filesystem RPCs. GEMS clients simplify user access to this tool when op-

erating within the replica system. While this solution offers the most functionality, it is only available on Linux.

3.5.2 Simple Performance Experiments

Experimental cases were performed on an simple archive creation workload, measuring the time taken to create a simple tar file from a local filespace used by a real world scientific application, the hyperdynamics experiments. The resulting archive is 3.7 GB and contains 13,934 files. The client creation site runs Linux 2.6.9 on 3.2 GHz Intel machine[2] and the data storage sites ran Linux 2.6.9 on dual 1.8 GHz 64-bit AMD machines[3], all on the same institutional network. The GEMS server was located on a dual 2.8 GHz AMD system[4] running Linux 2.4.27. The three methods from Section 3.5.1 were run and profiled; results are shown in Figure 3.7.

The *G/P* method created the archive and stored it with the GEMSput repository insertion tool in separate steps. The method *PIPE* consisted of three steps: creating a repository reservation with GEMSreserve, piping the output of `tar` into a Chirp I/O forwarding tool, and on completion, committing the repository record with GEMScommit. Each GEMS method took less than 2 seconds so the time consumed is not visible. The *VFS* method created the record using GEMSrun to drive `tar` execution inside a Parrot environment, and managed record construction internally.

Results showed that streaming output methods are slightly better than the two-step method, and that moving data through the pipe was slightly faster than moving data through the virtual filesystem. In general, the concurrency gained

[2] `caravaggio.helios.nd.edu`
[3] `sc0-*.cse.nd.edu`
[4] `gems.cse.nd.edu`

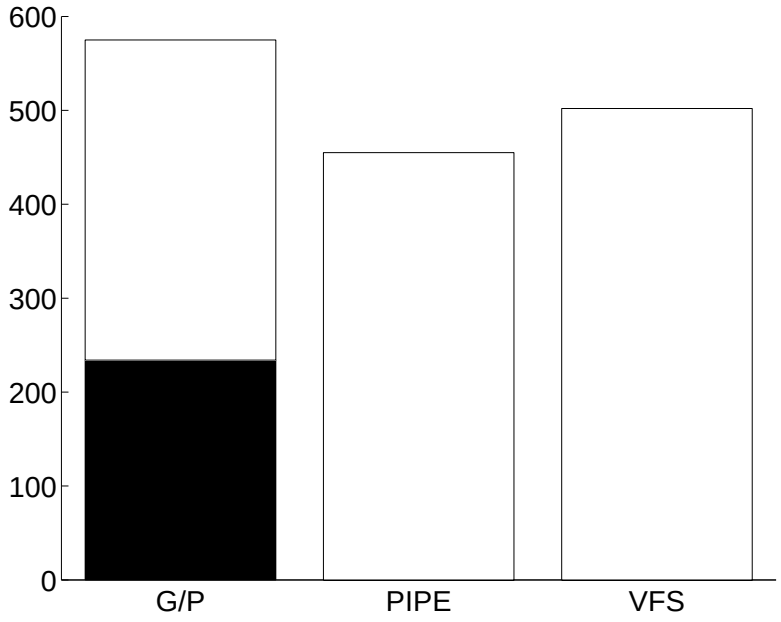


Figure 3.7. Archive creation times via various methods.

In the *G/P* case, the turnaround time consists of two components, local archive creation (black) and archive transfer to the repository (white). The other methods perform both operations concurrently.

by using the more complex *PIPE* or *VFS* methods may improve performance for processes that interleave computation with data movement, as shown here for data insertion.

Data retrieval performance is analyzed in detail in Section 5.2.4.2: since GEMS is designed as a repository capable of serving multiple readers in parallel, the analysis of this functionality must be taken in the context of the grid-enabling runtime toolkit.

3.6 Summary

Using the aforementioned techniques, GEMS implements the basic functionality required by a scientific repository: the ability to store, publish, and access large data sets. However, the ability to use partially trusted, unreliable volunteered resources depends on the ability to handle storage faults appropriately without data loss. In the next chapter, we attack the data survivability problem.

CHAPTER 4

DATA SURVIVABILITY

Using a cooperative, unreliable network of heterogeneous volunteer storage resources to build a distributed system results in many challenges. Providers are free to leave the system, and lack of reliability requires that the system be equipped to continue operating in an ongoing state of partial failure. Specifically, data management in such an infrastructure requires replication, parity techniques, or some other use of redundancy. In this chapter, we discuss our approach to data management with respect to fault tolerance.

The fault management system in GEMS was developed in 2006 [160, 161].

4.1 Overview

Large, shared storage systems create new problems. As users increasingly depend on greater numbers of more distant remote systems for storage, the underlying systems become less trusted and reliable for a variety of reasons. A storage provider could revoke storage by filling the storage device, evicting users, or simply turning the machine off. Additionally, the scale of the number of hardware components involved implies that individual components will fail regularly.

In a large distributed computer system, the process of machines joining and leaving the system is called *churn*; the churn of a system may be studied to obtain

qualities about the lifetime of a system member. Churn models often assume that once a resource leave the network, its state is invalidated so it cannot return as the same member. Systems that allow rejoining after failure use the concept of *availability*; a resource may be unavailable during a reboot but is expected to come back to the system with its data intact.

Churn is often studied in the context of peer-to-peer systems [77] in which the primary job of the system is to keep its fast search structure intact. Rapidly moving large data objects to keep them alive in a wide area peer-to-peer system with a high churn rate is not feasible, but a fast search structure allows the system to quickly locate members and objects that do exist. In a system with availability challenges, data objects may be replicated to improve the probability of their availability. GEMS thus assumes that large data files can be kept alive on infrastructures with availability challenges but relatively low churn, such as university networks. The Notre Dame network is used for a variety of projects that are feasible with moderate levels of unavailability [90].

4.2 Initial Experiments

The original GEMS replication method in 2005 used a simple 3-step technique:

1. Find a file with missing replicas;
2. Create an additional replica for this file;
3. Repeat.

GEMS performed this method successfully enough to manage small amounts of real-world simulation data on the Notre Dame network.

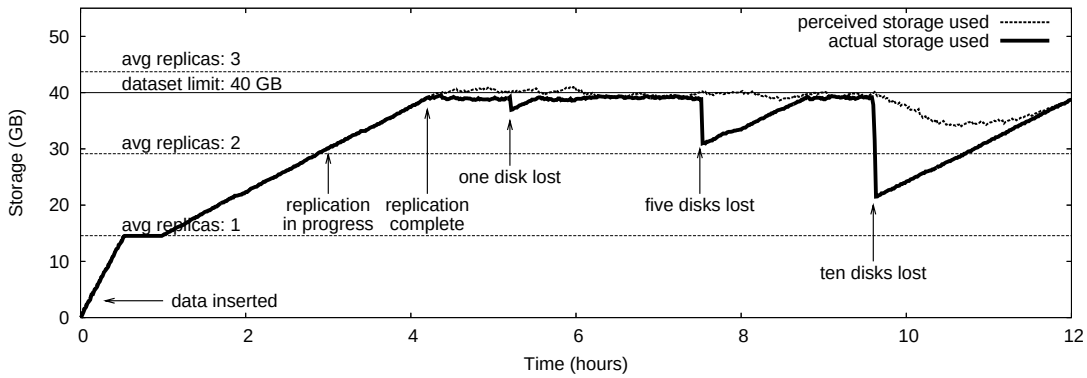


Figure 4.1. Example of fault tolerance over time.

The figure shows the first twelve hours in the lifetime of a 14 GB data set entered into GEMS for safekeeping. Out of a storage cluster of 20 disks, the system policy has allotted GEMS 40 GB to store this data set. Three failures are induced, but GEMS detects and re-replicates lost data. The discrepancy between actual and perceived storage indicates the time needed to discover failures via file scanning.

4.2.1 Simple Replication Experiment

In this section, we present the original replica method in GEMS on actual simulation data. GEMS relies upon several other resources that are used for data and metadata storage, as well as compute hosts. In these tests, we used 20 heterogeneous Chirp storage servers and a the dual-processor GEMS server[1], all of which were running Linux.

During this 12 hour experiment, all machines remained available to other non-GEMS users. The storage space utilized as reported by GEMS along with the actual physical storage utilized was recorded and plotted. An important practical aspect of this test is that storage availability fluctuates not only as a function of the storage utilized by GEMS but also the storage utilized by all other non-GEMS

[1] `gems.cse.nd.edu`

users. In this test, we assume all stored data is output data and is subject to tight replication limits. The GEMS components then react appropriately, replicating if available storage permits and deleting as storage becomes limited.

Figure 4.1 presents the system's functionality during this test with key events labelled. At the outset the distributed storage pool contains no GEMS data and the replication component of GEMS is not running. To start, multiple GEMSput operations are executed. For simplicity, in each GEMSput the number of replications requested for each file was set to 3. Once the GEMSput operations completed, we turned on the replication method and it attempted to provide all replications requested. Near hour 4, GEMS runs out of space to create additional replicas, so the usage level flattens out. GEMS processes iteratively adjust the storage to make sure that no record has 3 replicas while another record has not been given the 2 replicas it is permitted.

Once replication levelled off, we deleted the GEMS files from an increasing number of clients to simulate resource owners who have decided to temporarily evict GEMS from their system or experienced a hardware failure and replaced it with equivalent but empty storage device, simulating churn. Interestingly, for the smaller storage losses GEMS records little to no deviation in total capacity. The replication process continues from its current point in the file list and restores the missing copies as it discovers them.

Generally speaking, this test shows that GEMS can greedily consume the amount of resources that it has been granted, then safely protect storage resources from being overused.

4.2.2 Discussion

While this method showed some signs of success, its simplicity results in potential problems:

1. Replication is a blocking operation that impedes progress through the file list;

2. Faults are handled in the order observed, without respect to severity; and

3. If fault handling fails temporarily due to unavailability of existing replicas or other problems, the method must abort the fault handling and continue the loop.

A more advanced method is need to separate fault observation from repair, improving concurrency and adding flexibility. Faults should be prioritized to quickly prevent total replica loss and ensure that files more important to users are handled first. Fault handling failures themselves should also be handled appropriately.

4.3 Fault Prioritization and Feedback Control

The common technique to gain storage reliability over a long period of time is the creation of data replicas on multiple servers, but in the presence of server failures, ongoing corrective action must be taken to prevent the loss of high value *and* low value data. Such a system is difficult to control, and replica management is typically handled in an *ad hoc* manner, as shown above. In this section, we claim that repairing prioritized faults is a scheduling problem, founded on the need to minimize a risk-based error function, E. Citing experiments on a prototype replica system for molecular simulations, we apply concepts from control system theory to analyze and handle the application of corrective action.

4.3.1 Discussion

To achieve fault-tolerance and data preservation over long periods of time, replicas of each datum stored in the system are created and spread across large numbers of devices. The creation of replicas amplifies the size and scope of the data management problem. Additional replicas demand additional storage space. Appropriate metadata regarding the status of the replicas must be constantly maintained to keep the system intact. When data is to be retrieved, existing replicas must be located, and access to the data must be granted to the user. In a writable system, changes to the data in one replica must be propagated to all copies in a consistent way. Additionally, the system is dynamic. Appropriate response actions must be taken if a file can no longer be accessed. Analogously, action must be taken if a whole storage device or file server becomes unresponsive.

In this section, we describe our solution to the replica management problem. Our architecture is presented in Section 4.3.2, and in Section 4.3.3 we provide our model for replica systems in general, and show how it can be used to analyze the actions of a replica management system.

4.3.2 Control System Architecture

GEMS is an active replication system. Over time, the server processes actively probe for problems and determine a response, typically creating additional replicas or garbage collecting storage for later use. The active maintenance in GEMS consists of three components, the Auditor, Replicator, and Garbage Collector. The Auditor uses the metadatabase as a guide, and contacts the storage servers to determine whether the actual system is consistent with the information in the database. If a fault is detected, relevant information is compiled into a Problem

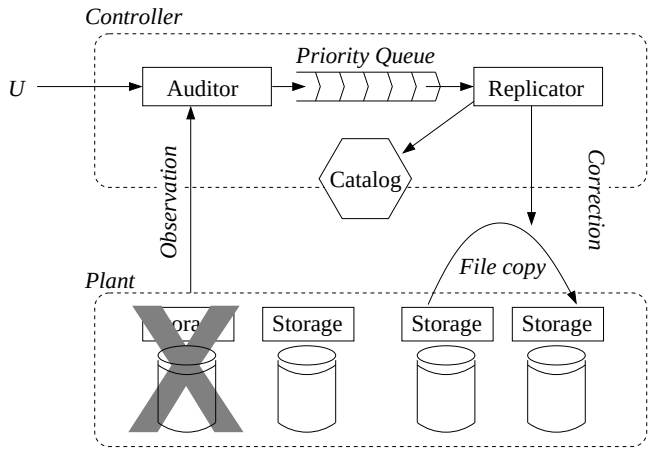


Figure 4.2: GEMS replica control loop.

object, ranked with a priority value, and enqueued in a *Priority Queue*. The Replicator monitors the Priority Queue, removes the Problem object representing the highest ranked fault, and determines a response.

In the case of a typical server failure, the lost file must be identified by consulting the metadatabase. Existing copies of this file are looked up, free space is located on a Chirp storage server, and the file is copied to restore the replica count to its desired level. If files are over-replicated, or otherwise inserted into the GEMS storage space inconsistently with the metadatabase, they will be detected by the Garbage Collector component and deleted to free space.

The maintenance process is shown in Figure 4.2. The *Observation* step in the control loop represents the active probing of the Auditor component for faults. Corrective action is shown as the *Correction* signal from the controller.

4.3.3 Control Model

In following, we discuss some principles to allow the use of control system theory to analyze our replica management system. We begin with basic definitions

and symbols, and then describe some basic tools from control system theory that prove useful when analyzing the GEMS system.

4.4 Fault Management Philosophy

The GEMS framework is subject to many of the same error modes that plague its cousin grid distributed systems. Sample sources of faults include: an errant user, the desktop hardware, the OS, the TCP connection, the local switch, numerous routers, the storage server software and its hardware, etc. Even though the apparent severity of one of the mentioned failures may differ by orders of magnitude (the loss of one node due to hard drive failure versus the loss of an entire DNS domain due to router failure), the GEMS design embraces the spectrum of failure with one unified resolution policy. *Observed fault handling is a routine maintenance operation.*

To give this concept substance we introduce the concept of a Problem. A Problem is an object in GEMS which can be assigned a priority based on its severity and queued for resolution. Problems are generated through Auditors which continuously audit the system state with respect to the metadata, physical reality, and user requests as reflected in the metadata. On the failure side of the Problem continuum, one example would be the loss of a local switch providing access to ten nodes, say, *Cluster A*. The Auditor would recognize that the metadata pointing to *Cluster A* is now inconsistent and for each unavailable file a Problem will be instantiated with all of the pertinent information necessary for calculation of a Problem priority and resolution. On the maintenance side of the Problem continuum, consider a client who submits a record of ten files into the database with a requested replication of 5 each. The replica will make sure the first copy

of each file is submitted to stable storage by the client tool but the remaining requested copies will be handled through the instantiation of Problems as the metadata shows an insufficient number of replications with respect to the users request. Hence, Problems can be subtyped to provide a useful abstraction for system operations.

4.4.1 Definitions

Fundamental to the analysis of a replica system is the number of replicas of a given file. The system contains N files, where each file f_i has x_i replicas, $f_{i1}...f_{ix_i}$. Each replica must be stored on a server c, which means that for each f_{ij} in existence, $\exists k : f_{ij} \in c_k$. We loosely say $f_i \in c_k$ if $\exists j : f_{ij} \in c_k$. The replica count is constantly changed as files are lost due to storage loss, or as files are replicated, so we have $x_i(t)$ representing the number of replicas for f_i at time t. If $x_i = 0$, then f_i is permanently lost. Each file has a size in bytes, s_i. The total storage consumed by the system can be expressed as:

$$S(t) = \sum_{i=1}^{N} s_i x_i(t). \tag{4.1}$$

The number of storage servers and the amount of storage offered is also constantly changing over time. Of the M storage servers, d_j represents the amount of disk space offered by server c_j. Thus physical storage limit of a server represents a constraint on the files that may be stored there:

$$\forall c_j, \sum_{i:f_i \in c_j} s_i < d_j. \tag{4.2}$$

The amount of available storage C in the system at a given time t may be

expressed as:

$$C(t) = \sum_{j=1}^{M} d_j(t). \tag{4.3}$$

A clear consequence of (4.2) is that $C \geq S$.

At the time of data insertion into the system, the user may specify how many replicas are requested for a given file, which we call u_i. If $x_i < u_i$, then the user's desired replication level is not being met, and replication should occur. The total storage requested by all users is:

$$U = \sum_{i=1}^{N} s_i u_i. \tag{4.4}$$

We consider U to be the reference signal to the GEMS system; the GEMS system is constructed to keep the underlying storage in line with the requested replica level.

4.4.2 System Response Analysis

We begin this discussion with an example: an experimental use of the system over a short period of time, as shown in Figure 4.3.

This experiment proceeded over a period of 5 hours. A GEMS installation was configured, and access was granted to Chirp storage servers running on 19 machines. The system was configured such that the amount of apparently available storage for GEMS was near 350 GB. The given diagram plots the amount of storage apparently available as observed internally by GEMS, and the amount of storage apparently consumed as observed by GEMS. Both observations are significantly delayed behind real time.

For the first 1.5 hours, data was inserted into the system using GEMSput.

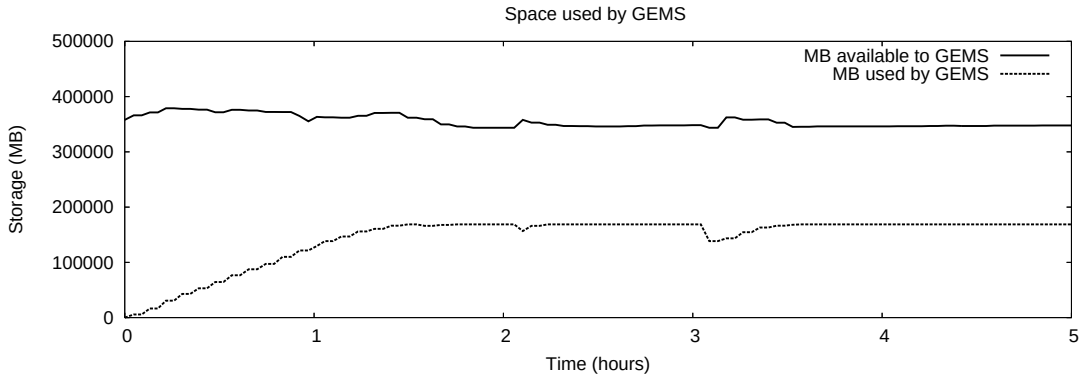


Figure 4.3: System response to induced server faults.

The input data consisted of 50 files, each 337.6 MB in size, each replicated 10 times. At hour 1.5, the system contained 500 files. Shortly after data insertion, a fault was induced on one of the storage servers, causing it to lose all GEMS data. This means that those replicas were permanently lost, but that the amount of storage available was not reduced. The effect is almost imperceptible on this plot. At hour 2, faults were induced on 4 servers, causing a noticeable bump in the amount of storage apparently consumed. At hour 3, faults were induced on 7 servers, causing an even more significant bump.

This type of diagram may be compared to a diagram describing the response of a system to an impulse. In this simple experiment, since all the files are equivalent, the input signal can be described as $U = 500 \times 337.6$MB, requiring the system to maintain 500 files of 337.6 MB each. Deviation from U may be measured over time to determine how poorly the system is able to respond to the forced change in state. A variety of error functions may be developed to quantify the performance of the system.

A simple metric is simply the storage used by the system. GEMS should not leave free space unused when additional file replicas are requested. For t in the

runtime of the experiment, the error function in (4.5) simply measures how well GEMS is filling the available space with data.

$$E = \int_t \min(C(t), U(t)) - S(t)\ dt. \tag{4.5}$$

Equation (4.5) offers no information about what is filling that space, whether it is high priority data, or even whether some files are over-replicated. If the system is full, i.e. $U \geq C$, this equation is a equivalent to:

$$E = \int_t \sum_{i=1}^{N} (u_i - x_i) s_i\ dt. \tag{4.6}$$

Since the system should not be rewarded for over-replication, we have:

$$E = \int_t \sum_{i=1}^{N} (\max(u_i - x_i), 0) s_i\ dt. \tag{4.7}$$

For many purposes, including fairness, file size is almost irrelevant to the worth of data. In many cases, if a single file is lost, of any size, the amount of work that the researcher would have to do to diagnose the problem and rescript the simulation run is similar. Strictly in terms of per-file replication counts, one could measure unweighted replica shortages as:

$$E = \int_t \sum_{i=1}^{N} (\max(u_i - x_i), 0)\ dt. \tag{4.8}$$

A related measurement is the number of files that have been completely lost due to the loss of all of their replicas. We measure this as:

$$Z(t) = (\text{The count of files permanently lost.}) \tag{4.9}$$

4.4.3 Optimal System Response

Most users of storage systems can specify which files are more important than others. GEMS makes expressing this evaluation easy, and provides a few ways of specifying data worth to the system. As discussed in Section 3.2.4, GEMS maintains a great deal of metadata about the data sets it stores which can be used.

The performance of a replica management system should be measured in terms of what the users require from the system. This is partially captured in the difference $u_i - x_i$, but this does not capture the value of the data.

A simple observation from the experience of running simulations is that input files are more valuable than output files. This is especially true in the GEMS environment, which is designed to promote the sharing of input files: it is convenient to have pre-computed output files, but if lost they may always be recomputed by combining the input files and the execution information from the GEMS meta-database. Additionally, other factors may weight the value of data files.

Data value would not be important if GEMS had unlimited ability to respond to faults, but this is not the case. The typical response to a fault, as described in our experiment, is to create an additional replica. The ability to create additional replicas is limited by several factors, including the availability of source servers, destination servers, and the network. GEMS limits the stress on servers by ensuring that the replication process never makes use of a given server for more than one task at a time, i.e., a server is either transmitting a single file, receiving a single file, or idle. GEMS does not currently explicitly limit its consumption of network resources.

The result of these observations and constraints is that faults in a large system

such as GEMS must be prioritized: they can not all be handled or repaired at once, and they do not all represent a threat to data of the same value. This line of thinking led to the creation of the Priority Queue in the GEMS controller, between the Auditor and Replicator in Figure 4.2.

The architecture necessary is simple enough, but the priority assignment scheme has not been dealt with. Given a certain set of faults, what is the appropriate action to take?

If we assume that we have an error function E that is correct, then we simply must minimize E. At each opportunity, the system must take the action that will likely result in the minimum value of E. Since future values of the cost function are affected by unknown, unpredictable events such as server failures, we must make standard assumptions about the future status of the system. For example, we can assume that all servers are equally likely to fail, regardless of the content that is stored on them. This implies that we may minimize E by locally minimizing the cost function. If the cost function E is easy to understand, this is easy to do.

Our cost function is simply the sum of a set of values that result from observed faults. Each fault is prioritized in some way. So the set of actions that we may take is made up of the set of corrective actions we may take to repair an observed fault, and minimizing the cost function is equivalent to repairing the highest priority fault first.

4.4.4 Determining a Priority System

The remaining problem is to find a method to pick an appropriate E. This is now equivalent to the problem of prioritizing faults. The priority of a fault is then some function applied to the available information about that fault.

Field	Type
RequestedReplicas	`integer`
CurrentReplicas	`integer`
Duration	`date`
Size	`integer`
InputFile	`boolean`
FileMetadata	`object`
Priority	`integer`

Figure 4.4. The Problem object in GEMS.

In a typical replica management system, there are several available statistics about a fault. We have the number of requested replicas from the user, the number of replicas intended to be allocated by the system, the number of observed existing replicas on storage servers, the time elapsed since the fault was detected, and the size of the affected file. In the GEMS system, we also have information about whether the file was an input file to a simulation. GEMS represents each observed fault as a Problem object, with fields summarized in Figure 4.4.

Each Problem observed by the Auditor represents a file that is below its requested replica count and thus is in need of additional replication. The *RequestedReplicas* field indicates how many replicas of this file were requested by the user, and indicates indirectly how valuable the user feels this file is. The *CurrentReplicas* field indicates how many replicas currently exist in the system, which is a volatile observation. The system stamps each Problem with the time of observation, stored under *Duration*. The size of the file is stored as *Size*. File size may affect the resulting response in a variety of ways, for example, the system may choose to delay replicating a very large file to allow hundreds or thousands of smaller file copies to complete first: performance which may be desirable under

many of the error definitions given above. The *InputFile* field is *true* if the user has indicated that the given file is an input file to a simulation. *FileMetadata* is a compound field that provides the filename and replica locations, and is needed to perform the repair. Based upon all of this information, the *Priority* field can then be used to determine which Problem to process next, highest *Priority* first.

Each Problem object contains a set of members which are necessary for the Problem's resolution and which also serve as the parameters for the priority calculations. From the set of priority members a priority is calculated internally by the Problem on instantiation. Of key interest is the ability for the Problem to recalculate its own priority. For example, the priority queue could trigger all Problems in the queue to recalculate their priority (stale Problem member data is addressed in the following section).

A good demonstration of the priority recalculation utility leads us into the discussion of priority function parameters. Classical job starvation concerns would prompt us to include "time in queue" as a parameter with varies in proportion to an increased priority. Problems involving a host failure which has only been recognized for a short period are not handled immediately, which gives the host system a chance to recover, eliminating an unnecessary file transfer. A second parameter of importance for our scientific user base is file type. Whereas an output file may be automatically regenerated because the metadata contains enough information to derive the output files, the input configuration files are irreplaceable without human intervention. To again reduce the probability of permanent data loss we increase priority in with respect to the number of remaining replications. Additional parameters such as file size allow us the ability to fine tune the prioritization for improved response to massive failure scenarios. An important observation in

biomolecular simulation data sets is that input files tend to be small, and output files are large. This allows us to aggressively replicate the input files while being more cavalier about output files, which saves storage space and bandwidth.

In GEMS, the priority function (P) is related to the above error functions but is also influenced by more general notions of fairness, as well as known observations about the underlying storage system. For example, it is known that many storage servers that are unavailable will eventually come back, so a delay (D) component is introduced, and the priority is based around the duration in minutes (M) of the Problem, scaled by the risk of total file loss (K). We adopt the convention that Problems with $P \leq 0$ are left in the queue. Currently:

$$P = MK - D \tag{4.10}$$

4.5 Experiments with Prioritization

In this experiment, we demonstrate that the new replicator functionality improves upon the previous functionality, which is similar to functionality obtained in a system with limited access to simulation metadata. Such a data-agnostic system could only repair errors in the order observed, one at a time, and could not respond dynamically to a rapidly changing storage fabric. Our prototype can emulate this behavior if we make all the priorities equivalent, and perform in-order responses.

Our example shows that the prototype can respond to faults in the order specified by the priority function. In this case, small files have priority over large files, which is typical in a biomolecular simulation environment where the input files and configuration files are irreplaceable, and the output files represent derived

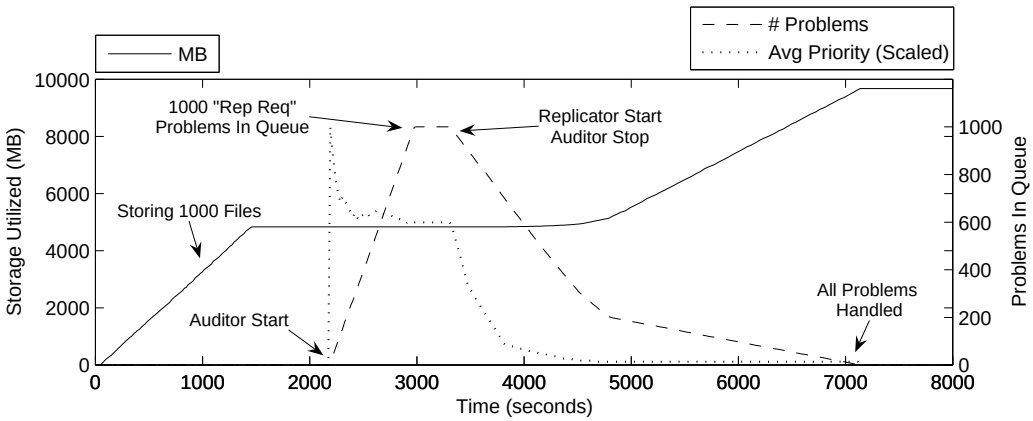


Figure 4.5. Priority queue performance.

data and are only required to speed the retrieval time.

The underlying experimental setup consisted of the client machine, the GEMS server, and an underlying storage network of various heterogeneous Linux machines from the Notre Dame campus network running the Chirp file server. The client machine, a 900MHz AMD Linux system, controlled the experiment by communicating with the GEMS server [2], *via* the GEMS clients.

As demonstrated in Figure 4.5, we start by disabling the GEMS storage management components, the Auditor, Replicator, and GarbageCollector. The simulation data sets, about 3 GB total, are added to the system, so each file has one copy on some Chirp server. The Auditor is then turned on and replica shortages are detected: the files need to be replicated up to the requested level. The graphic shows the average priority of problems, which are scored by, in this case, the size of the file and the replication count. The Replicator is turned on, and the average

[2] `gems.cse.nd.edu`

priority quickly drops as high priority problems are quickly handled by replicating these small files on other Chirp hosts. The average priority slowly levels off as larger, low priority output files are copied over the network, increasing the disk usage level.

4.6 Summary

Replica control systems have complex dynamics that must be controlled carefully to minimize the risk of the loss of high and low valued data. There is a need to quantify the state of the system and its current level of risk in a cost function. The resulting error function for the system may be then be used to control the system in a systematic way, as opposed to an *ad hoc* manner.

Specifically, our approach solves the three problems mentioned in Section 4.2.2. Concurrency is enabled by the division of the Replicator and Auditor components and parallel use of the network connections among storage providers; a comprehensive prioritization scheme is employed that utilizes much available information; and faults that cannot be immediately repaired may be simply requeued.

The replication model in GEMS evolved from a simple, automated find-and-fix agent to a multicomponent, prioritized, parallelized controller with multiple levels of fault management. This control model enables the construction of repositories on unreliable resources which would otherwise be unsuitable for data services, and better utilizes resources that often go untapped: the storage devices and peer-to-peer network connections in opportunistic computing networks. These networks can be used as a fabric for the computation and administration techniques explored further in the next chapter.

CHAPTER 5

GRID INTEGRATION AND DERIVATION

Data repositories are an integral part of modern scientific computing systems. While computer processing power has continued to increase, available disk bandwidth and network performance have not kept up - necessitating proper attention to the design of distributed storage systems that do not restrict performance for scientific users. Grid computing poses additional design constraints on storage systems, insisting on scalability, interoperable administrative control, and flexible architectures. While a variety of grid storage systems have been developed, users have several outstanding needs: to work with replicated data sets, to customize system behavior within a grid, and to quickly tie together remotely administered grids or independently operated resources. This is particularly true in the *small virtual organization* case, in which a subset of possible users seek to coordinate subcomponents of existing grids into a workable collaborative system. Our approach to this problem starts with the storage system and intends to enable this functionality by creating *ad hoc storage grids*.

5.1 Overview

Previous work on large scale storage has taken several forms, including building globally rooted filesystems, distributed databases, or interface description for

explicit data movement. While the filesystem model enables existing software, it restricts user ability to customize the system by masking system configuration and data placement decisions. Database models excel at representing scientific data in application-oriented ways, but similarly restrict user decision making. Explicit data movement technologies allow for optimal performance and customization, but also require expertise and time investment in scripting and programming.

A cooperative grid storage system consists of a network of storage resources which have been provided to for common usage. These systems provide an economical method to increase the utilization of existing resources or to donate resources to prioritized projects without the administrative commitment that is often required. For example, reporting underutilized desktop workstation hard drives to a common catalog and allowing remote access enables other users to store large data sets they would otherwise not be able to maintain. Additionally, replication may be performed in this setting to increase survivability or improve data movement parallelism as discussed below.

The next generation grid will overcome the limitations of these systems - both in customizability, which enables locally optimal performance, and in ease of utility which enables maintainable software. We claim that the solution to these problems involves flexible systems that are hybrids on several design axes, including the filesystem vs. database axis and the explicit vs. implicit data location axis. The administration of these systems will be reconfigurable at the user level, allowing for integration of grid resources into larger systems and derivation of smaller grids; likewise, enhanced or limited user privileges will be administered by users on their own data environments. Applications developed on such a system will accommodate these hybridizations, enabling system-aware workflow structures.

This chapter reports grid-enabling design concepts developed in conjunction with the GEMS system. While the system originated as a repository for molecular dynamics simulations, it evolved into a general-purpose storage integrator, enabling storage resource aggregation, replica management and maintenance, high level user control over storage policy and access control, and an application-friendly computation environment. Additionally, we will describe the programming model used to access this metaresource, a high level database-driven process called a *parameterized workflow*. We then describe how the unstructured volunteer fabric may be integrated into a viable back end for these workflows by integrating and deriving grids from swaths of individual services.

Existing "big iron" grid systems have provided data access [22] to grid resources and integrated complex I/O systems with user programs [3]. Contrarily, GEMS tackles unreliable storage resources with an opportunistic approach to repository construction, enabling stakeholders to share physical and data resources securely at the user level. However, GEMS implements many important grid design concepts, offering three major grid-enabling features:

1. **Integration of existing resources:** GEMS corrals free disk space into a unified resource, increasing utilization of these systems. This meets the primary grid design feature of enabling arbitrary aggregation of the resources available to the user. These storage resources are used as a basis for a parallelizable data service in support of a complex, distributed computing system.

2. **Abstraction of heterogeneous underlying systems:** GEMS relies on underlying portable software that allows many different systems to be represented. Additionally, GEMS-specific software is portable and relies on web

service-like [147] communication protocols. This enables a variety of access techniques for data clients.

3. **Organization-delegated authentication:** Grid systems must accommodate scalable authentication techniques by delegating control to underlying stakeholder organizations. GEMS offers multiple contributions in this area, including an indirect authentication technique and the capacity for the administrative delegation of user privileges.

In the following sections, we will describe how these features allow users to access *GEMS as a grid*. This represents a comprehensive case study of a the integration of system built around the overdrive controller concept with real-world distributed computing issues such as data movement, resource management, and authentication.

5.2 Data Services for Simulation

This section describes the use of GEMS in conjunction with a distributed computing service.

5.2.1 Overview

Cooperative storage systems have been assembled to provide users with access to large amounts of useful space, but locating, accessing, and efficiently employing such distributed data sets in scientific simulation or analysis is another monumental problem. In this section, we describe simple techniques compatible with existing software that may be used on clusters or grids to access cataloged, replicated data sets: software tools that allow user code to use a replica system as

a scientific file system. The importance of operating in-place, close to the data, is demonstrated by the performance improvements gained by using data locations as a guide when scheduling computation. A new replica management system called GEMS is proposed to improve the research capabilities of two important computing infrastructures: university networks (groups of clusters) and volunteer computer resources (Internet computing).

5.2.2 Discussion

Modern shared computational clusters and grids offer users a great deal of capacity for code execution. When multiple independent jobs are executed in such systems, the total system may obtain a performance speedup that is linear in the number of compute nodes *after input data has been distributed to the nodes and before output data is recovered*. The data movement mechanism - the manner in which data is moved among file servers and compute resources - is limited by the network, and individual file servers are limited by performance specifications and network connectivity. Even on small compute clusters, batches of jobs that operate on sizable files will quickly find that a simple centralized storage device will become the bottleneck. Larger scientific tasks that require large batches of thousands of jobs or more will overwhelm a centralized service, resulting in poor performance for all jobs.

A potential solution to this problem is to utilize multiple data storage services at different locations in the network. Developing such a distributed file service involves dealing with several challenges, including logical name resolution, efficient matchmaking, and data movement. First, in order to gain access to data, jobs must be able to resolve given logical file names into physical file locations, i.e.,

where the actual file may be obtained. Jobs must also be able to store new data in the system in a permanent way - not just in temporary space on the compute node - somewhere in the distributed system. Second, jobs gain significant a performance improvement from being *close to the data*. If the compute network and the storage network occupy the same physical machines, a significant performance improvement can be obtained by matching jobs to machines that have the required data sets. Third, a mechanism for *data movement* must exist to support such a system, which allows a running job to gain access to the data associated with a physical location, either by staging whole files or through a virtual filesystem.

Our alternative approach *recognizes the replica creation process as an asynchronous data pre-staging process*, in which data sets are moved to potential compute sites in advance. In our model, the exploitation of compute and storage site collocation allows reads and writes directly to the local file server. Upon job completion, output data is committed to the system, replicated by the system in an asynchronous manner, and actively maintained by auditing services. This overlay system allows users to employ a vast network of independently operated storage servers *reliably* and *without explicit data staging*. Such an approach effectively builds a computation system atop an existing replica location system, thus adding the benefits of parallelism to the fault tolerance benefits of data replication.

The replica creation process offers much, yet its implementation incurs costs that must be justified to scientific users. In this section, we offer three contributions to motivate the use of this type of system. First, we characterize computing models that employ replica sites as data sources, while considering the programmatic interface to these methods. Second, we present an experiment from a real-world biomolecular dynamics application employing a significant number of

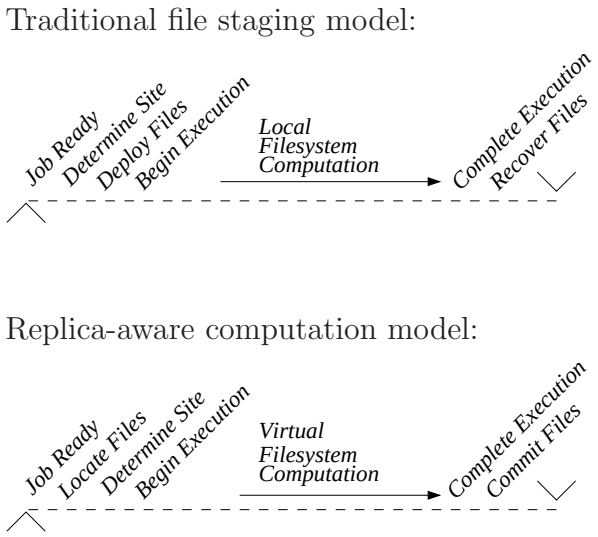


Figure 5.1. Grid data access models.

In the traditional file staging model, the computation site is determined before data files are considered, because all data is assumed to come from a centralized source. In the distributed, replica-based computation model, the replica locations *are* considered when selecting the computation site, and a virtual filesystem is used to connect jobs to data services, because the files are not explicitly copied over the network.

processors and data sources. Third, we explore a reusable mathematical model that predicts job turnaround time as a function of obtainable system specifics such as data bandwidth and job runtime.

5.2.3 Replica-aware Computation

Shared replica management systems have been designed to meet the storage requirements of users requiring a variety of functionality. Users benefit from increased storage space: especially short term storage space, as the shared workspace may increase utilization of the underlying systems. Additionally, replicated data sets are more resilient to hardware failure or loss, as replicas serve as backup

copies. User groups that desire to publish their data inside a virtual organization benefit from cataloged replica systems, which provide a searchable catalog of metadata and allow for data sharing and reuse.

The Parrot adapter has been previously developed [133] to treat the whole distributed file system as a single filesystem. By trapping file operations from a running process, the adapter may allow the process to access files from remote servers in a transparent way. This adapter allows for the user to choose the *compute site*, the *input source site*, and the *output destination site* independently. The replica management system is combined with these tools to locate data. A new external system is needed to locate input sites, find sites to safely store outputs, and obtain a compute site that is not already occupied. To obtain good performance, the three sites should be collocated.

Automated resolution allows clients of the replica management system to map file locations in the searchable, database-like namespace to the physical namespace of the virtual filesystem. In this work, we call the entries in replica namespace the *abstract* filenames, as opposed to the *virtual* filenames compatible with the adapter. *Name resolution* maps abstract names in a `/label/path/file` format to virtual filenames in a `/protocol/host/path/file` format. The replica management system provides the additional ability to obtain data set labels by searching over the metadata, providing the ability to perform computation in a completely application-oriented way, as shown in Figure 5.2.

The first line of the script uses the client toolkit to locate the data set label required to obtain the necessary input files for the `transmute` task. The second line invokes the toolkit to resolve the abstract file names to virtual file names compatible with the adapter by:

```
> KEY=$( GEMSmatch reagents=acidbase )
> GEMSrun --input HCl /$KEY/hcl
          --input NaOH /$KEY/naoh
          --output NaCl salt
          --output HOH water
          reagents=saltwater
          --exec transmute HCl NaOH
                 to NaCl HOH
```

Figure 5.2. Example GEMSrun script with abstract data locations.

1. Defining input tokens for abstract files and output tokens for new files (e.g. the output file `salt` is represented by `NaCl`);

2. Substituting the virtual file names in place of the tokens (e.g. `NaOH` → `/protocol/host/path/naoh`); and

3. Submitting the user task (`transmute`) to an available compute system, which uses the adapter to actually perform the file operations.

The result of this script is a new entry in the replica system, containing two files, `salt` and `water`, which may be located by using the tag-value pair: `reagents=saltwater`.

Clearly, more complex tasks would involve very lengthy command lines. Since existing job schedulers already require users to explicitly define input and output files, simple extensions to the syntax of these job scheduler scripts may be preprocessed by the GEMSrun client to provide a more familiar syntax for job submitters.

In a replica location system, a service maintains a database of replica locations. This service may be queried in three ways for the purposes of this paper: to map

metadata tags to data set labels, to map data set labels to a set of file names, or to map a data set label and file name to a storage site. Once the site has been obtained, the data source may be accessed as described above.

While a replica management system does not spawn computation itself, we demonstrate a tool to interface with existing computation systems to access, analyze, and create data in a compatible way. In this section, we outline four *computation modes*, that is, ways to effectively perform computation utilizing replica access, including:

1. Local Computation on Remote Data; which allows the local workstation to access remote data sources and create new data in the replica system over a virtual filesystem.

2. Scheduled Computation on Remote Data; which interfaces with an existing scheduler to create jobs that access data over a virtual filesystem.

3. Remote Computation on Remote Data; which utilizes the ability to directly send jobs to remote systems for processing.

4. Multiple Name Resolution; which is a more complex scheduled method, guiding the matchmaking process with respect to data locations.

Local Computation on Remote Data (LCRD)

In many common cases, the user simply desires to run a single job on the local workstation. The LCRD model enables users to start new jobs that require data access to the replica management system. This mode is based upon a typical command consisting of an operation on abstract data set identities, comprising the input and output locations. These abstract arguments, which do not specify the actual data location, are translated by the replica system into the physical

file locations required by the virtual filesystem adapter, in a manner analogous to shell parameter parsing and expansion. Thus a task that requires access to remote, abstracted data sets may be translated into a local task operating on data that is virtually local. An example execution of this method is shown in Figure 5.3, in the LCRD frame.

Optimizing this operation is quite simple. First, the output data location is determined. The preferred output location is a server on the local host, but if this is not available or not allowed by the relevant access control policy, a server that is not currently busy will be selected. In the worst case, a remote, busy machine will be selected. If any required input files from the system have replicas on a local server, these hosts are selected as data sources. Otherwise, a remote, idle service will be selected to provide the file, and if this is not possible the worst case behavior of a busy remote server will be selected.

Scheduled Computation on Remote Data (SCRD)

We augment the specification for scheduled remote jobs by allowing users to specify input and output locations that reside in the replica management system, and use these locations in the commands and arguments. The client translates this augmented submission script into a submission script compatible with the replica system by making necessary substitutions: resolving the abstract data locations into virtual filesystem locations, and ensuring that the resulting job does in fact run atop the adapter. The data sets required by the resulting job are then location-independent because of the adapter, i.e., no data staging is necessary.

As shown in the SCRD frame of Figure 5.3, jobs are sent to the scheduler with requested destination hosts, illustrated by the "@" markup. The scheduler honors the request and submits the job to the appropriate compute site.

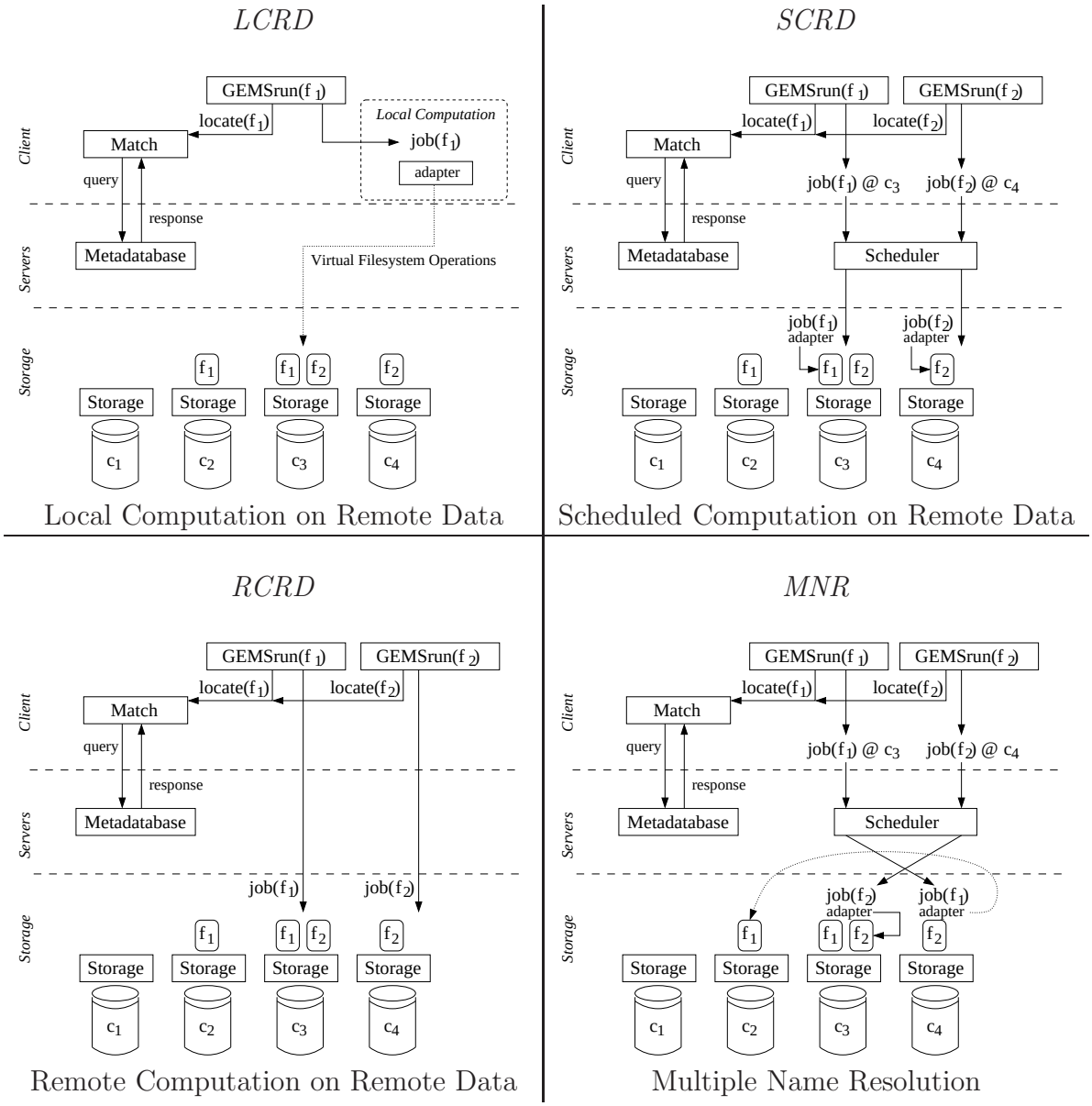


Figure 5.3: Computation in a replica management system.

Remote Computation on Remote Data (RCRD)

The file server used in this work allows the user to execute jobs inside a server-side execution environment. The user makes use of this technique by sending jobs directly to a file server for computation. The method is similar to the scheduled SCRD method, however, we are missing two important concepts: external centralized matchmaking and scheduling.

An example of replica access is shown in Figure 5.3, in the RCRD frame. Two jobs are submitted to the system by the user. Each specifies a target file, f_1 or f_2. The requests are translated into replica location operations. The request for f_1 is received first, and the response indicates that the file can be accessed on host c_3. This host is then marked "busy". The client then submits the job to host c_3. The second response arrives at the server, which locates the file on c_3 and c_4. Since c_3 is busy, the job is sent to c_4.

Multiple Name Resolution (MNR)

As seen above, the local computation method selects replicas relative to the given computation site, which is immutable. The scheduled and remote computation methods select replicas relative to a variety of potential compute sites, presenting challenges of interest to designers of grid computing systems.

1. In an environment in which replica locations are free to change or fail, replica locations may need to be re-selected after job deployment.

2. Specifically for scheduled jobs, if the job is not deployed to the specific host that optimizes the file transfer, it may be beneficial to re-select replica locations to minimize transfer relative to the actual computation site.

Clearly property (1) is a harder constraint than property (2) but both represent essential design issues.

A potential solution involves resolving the replica locations twice. A first resolution is performed by the job submission routine, which now selects only the computation site. The computation site is chosen in such a way that the resulting file transfers will be minimized. Then, after deployment, we have a second resolution: the compute job again resolves replica locations, essentially performing an LCRD operation described above. At this point, all replica locations may change due to storage server failure or computation site surprises, but the selection of compute site is fixed, greatly simplifying the choice. This may produce very good throughput at the small but significant cost of a second query to the centralized replica location service[1].

In summary, we have a scheduled method similar to the SCRD but more robust and efficient because of the complex handling of replica locations. Below, the algorithm for the Multiple Name Resolution method is outlined:

[1] A second query is not absolutely necessary: results from the first query could be packaged, deployed, and reused. There is a fault-tolerance trade off here that is outside the current focus.

Job Submission

The following algorithm is performed by the client.

1. For each potential compute host c_i, compute the network transfer n_i required to perform computation on that host.
2. Compute appropriate ranks and submit the Late Resolution algorithm as a job to the scheduler.

Late Resolution

The following algorithm is performed by the job upon arrival on a compute host.

1. Determine the host this task is occupying.
2. Locate all required files, and prefer locations that are on this host.
3. Resolve abstract file locations to virtual file locations in the user job argument string.
4. Execute the user job atop the adapter.

This algorithm is illustrated in Figure 5.3, in the MNR frame. In a manner similar to the SCRD illustration, the user submits jobs, the replica management system suggests appropriate hosts, and the jobs are sent to the external scheduler. However, the external scheduler places job(f_2) on c_3 first, then places job(f_1) on c_4. Simply applying the SCRD method here would cause two network file accesses: each job would begin accessing files found on a different host. However, using the MNR, job(f_2) utilizes the Late Resolution method and obtains access to the local copy of f_2. job(f_1) utilizes the Late Resolution method, and cannot locate a local copy of f_1 or the originally preferred copy on c_3, but is able to locate the copy on c_2. The net result is that one job obtains access to a local file, and one job must access a file over the network.

Summary

LCRD jobs can be executed in any environment in which a replica location service is available, creating a useful and practical prototyping tool for running

simulation in the presence of any replica location system. They even can be submitted to job schedulers, implicitly creating an "unguided" MNR method.

SCRD jobs provide a useful and often requested additional functionality to existing job schedulers: they allow matchmaking based on replica location. However, once the job arrives, it functions as a LCRD job, meaning that if a different compute site is allocated by the scheduler, all file access must occur over the network, because name resolution has *permanently* occurred[2].

RCRD jobs require the user to have compute access to the remote machine, over a system such as SSH or Chirp. A special case that could benefit from such a system are Internet computing applications as discussed below, because such applications often require an application-specific job scheduling policy. The RCRD method would allow such applications to use the data locations as an additional guide in the process.

The MNR is a robust and complex method for job scheduling. By both guiding the job to an appropriate compute site and making corrections upon arrival, it gains the benefits of the replicated data sources and the global view of the scheduler. Practically, it relies upon the ability to package additional code as a wrapper around the user job, which may be a constraint in some environments.

5.2.4 Applications

This section describes how these concepts may be applied to the target application, molecular dynamics. We first discuss two environments in which replica storage systems and job schedulers may be combined using the above principles to increase utilization of existing resources and benefit users by creating useful and

[2]A variation would be to instruct the scheduler that a given job is only eligible to be run on the specified host, and must otherwise wait- which would provide good collocation but would fill job queues in many applications.

efficient virtual workspaces. We then provide experimental results that show how the system performs in these settings, and interpret them with a mathematical model.

5.2.4.1 Operating Environments

The University Network (LAN)

Many universities have clusters or laboratories with a variety of machines connected over a relatively fast internal local area network. These clusters may be combined into useful high throughput, high utilization systems with the appropriate software. For example, an engineering building at the University of Notre Dame, which contains over 200 Linux and Solaris machines, has been combined into a Condor [79] system. The default Condor installation package used on campus includes the Chirp [133] file services, totalling over 7 TB of available distributed storage. Additionally, the University is served by a centralized AFS [69] installation. Users have a large pool of computing and storage resources at their convenience, however, permanent storage and physical file location management must be handled by the individual users.

New jobs submitted via Condor require that input data must be immediately staged to the host, either explicitly by file transfer or implicitly by AFS. While the local network is capable of serving data for a handful of compute bound jobs, cases have arisen in which a small number of output heavy simulation jobs have caused debilitating effects on the network.

Our strategy replicates the data files in GEMS. Users then submit a batch of GEMSrun requests to the GEMS server, generating Condor submit files via automated client tools, so that the computation proceeds on a host which can

serve that data locally, thus eliminating *ad hoc* file transfer.

Internet Computing (WAN)

A variety of applications are suited for the "@ Home" computing model, in which volunteers allow their computers to be used for large-scale scientific projects when they would otherwise be idle. Popular examples include SETI @ Home [11, 12] and Folding @ Home [98]. The range of potential applications is limited because of the perceived data movement bottleneck, so applications are typically chosen only if the amount of data transfer is relatively small. Applications that require heavy amounts of input and output present a list of challenges, including how to move the data to a compute host in the first place.

GEMS offers a strategy to approach these problems. First, by increasing the number of data servers, we decrease the load on the central data server. Original input data may then be served once to a set of volunteer hosts, and the replicas will be automatically maintained. Data can then be managed with fairness and load balancing as described previously [158]. Output data would be written to the local disk, and asynchronously replicated among the volunteer machines. Subsequent post-processing of the output data would be performed by locating existing replicas of the data and computing on the storage sites, without any additional data transfers.

5.2.4.2 Experimental Results

In this section, we describe performance experiments utilizing the replica management system GEMS combined with the Condor job scheduler. We intend to show improvement in the amount of useful work completed for our target application by collocating data and computation with GEMS and performing compu-

tation atop the Parrot personal filesystem.

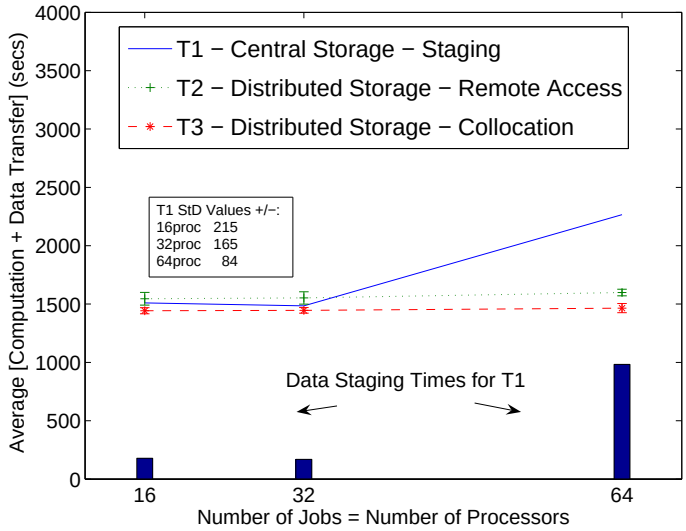
In our experiment, we use VMD [70] to postprocess a 350 MB ProtoMol [84] output trajectory file, performing an RMSD based clustering calculation. We first tested the university network LAN environment as typified by low latency, high bandwidth connections among tightly clustered compute hosts or clusters of hosts. Our experimental testbed[3] typifies this setting, consisting of 32 dual processor Intel Pentium III 1.4 GHz machines with 1 GB of RAM, connected by a dedicated 1000Mb interconnect, and running Linux 2.6.9. In this environment we analyzed the average runtime performance of three competing computation and storage models: "Staging", "Remote Access", and "Collocation".

Periodic executions of the same postprocessing task were submitted to the cluster and runtimes were averaged with standard deviations indicated by the error bars. As shown in the Cluster Configuration graph of Figure 5.4, we measured three computation techniques on this testbed.

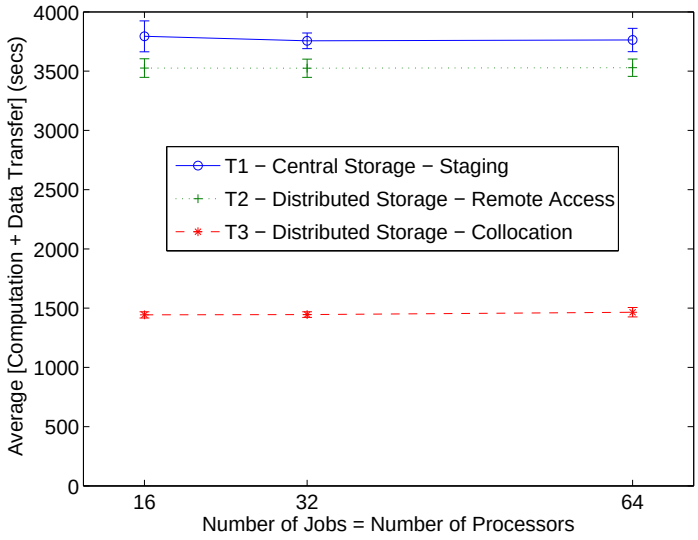
The "Staging" model ships the input file from a centralized storage server for each execution of the task. "Remote Access" indicates that each VMD job ran atop the adapter, with the constraint that data must be obtained from a *remote* replica location. A third test was performed using "Collocation", as discussed in Section 5.2.3. In this case, jobs are scheduled to only execute on hosts that have the required data file. Performance for the replica-based methods are superior as the number of concurrent jobs increases, due to the increased scalability of the data service.

A second battery of tests was performed on the same cluster, tuned to perform as an Internet Computing system. The file server on each machine was "throttled" to limit the data bandwidth from any given file server to 1 MB/s.

[3] `loco*.cse.nd.edu`



Cluster Configuration



Internet Configuration

Figure 5.4. Total job turnaround time for GEMS jobs.

While this testbed does not model all important properties of WAN performance or "@ Home" computing, it is useful to illustrate the idealized performance of the three computation methods.

As shown in the Internet Configuration graph of Figure 5.4, "Staging" and "Remote Access" perform similarly, as job execution is limited by access to data. The performance of "Collocation" is equivalent to that obtained in the "Cluster Configuration", yielding a significant speedup of greater than 2 when compared to the Staging and Remote Access models.

We see that an SCRD algorithm, as described in Section 5.2.3 becomes crucial in order to obtain good performance. On the wide-area network, *jobs must be close to the data*.

5.2.4.3 Bandwidth Analysis

Since an important performance goal of a replica system is to increase the available bandwidth parallelism in the system, it is of interest to consider the relationship between the parallelizable and sequential tasks that a system may perform. In this subsection we will consider the above experiment more formally. Our analysis essentially follows Amdahl's rule [9] with bandwidth considerations. Each processor in this framework alternates between two significant phases: a sequential phase and a parallel phase. The parallel phases are independent and perfectly parallelizable, for example, the computation done by each independent job. The sequential phases are tied to some centralized resource such as a single file server. Additional system configurations may be modeled by assuming the data access is parallel and the replica location operations are serial, such as in the case of a replica system.

The following model is diagrammed in Figure 5.5. First, we consider the case of a single data server with n independent processing jobs which periodically issue large file reads. If S is the file size and B' is the apparent bandwidth, with a fixed processing time of J, the the turnaround time for a single job is

$$T = \frac{S}{B'} + J. \tag{5.1}$$

If there are r jobs currently reading and the total available bandwidth is B, the apparent bandwidth is $B' = \frac{B}{r}$. The apparent number of jobs contending for bandwidth, may be denoted as the number of reading jobs:

$$r = 1 + \frac{(n-1)\frac{S}{B'}}{\frac{S}{B'} + J}. \tag{5.2}$$

Thus a reading job must suffer from the contention caused by itself and the fraction of the $(n-1)$ other jobs that are in the read state.

To obtain the modeled job turnaround time, we must solve Equation (5.2) for B' to determine our apparent data access rate.

$$B' = \frac{-Sn + JB + \sqrt{S^2n^2 - 2SnJB + J^2B^2 + 4JBS}}{J} \tag{5.3}$$

To obtain the turnaround time for the job, we use the access rate from equation (5.1). That is, given the appropriate S, n, B, J, we can obtain the apparent bandwidth B' for a job and plug in to obtain job turnaround time T, taking into account the deteriorating performance of the centralized component as n increases. Analytical results are shown in Figure 5.6 along with experimental job runtimes with these parameters. It should be noted that the job phase cycles are not in lockstep with other processors due to small scale variations in system performance.

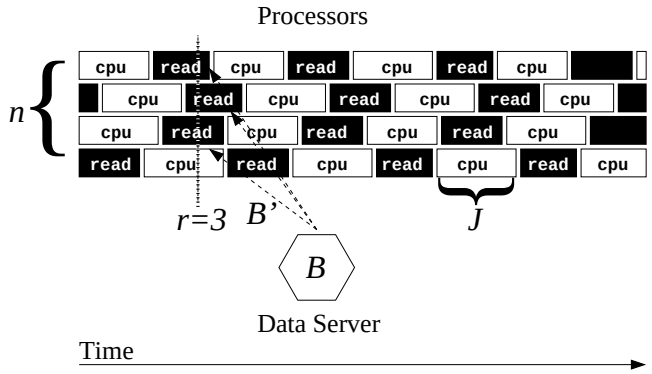


Figure 5.5. Symbols used in bandwidth analysis of data consumption jobs.

Graphical depiction of the symbols used in the system analysis, shown as a Gantt chart. The system consists of $n = 4$ processors, r of which are expected to be reading at any given time. The data server has a maximum bandwidth of B, only B' of which is available to any given process. The job length is J seconds. At the instant shown, $r = 3$ so $B' = \frac{B}{3}$.

Performance heterogeneity among processors is not considered.

The resulting figure indicates that for our simulated workload, we can expect good multiprocessor speedup until about 20 processors are utilized, after which bandwidth constraints increasingly cause sequential performance.

More generally, the model may be applied to the replica case. Assuming reasonable load balancing that could be obtained through the random selection of replica sites, the available bandwidth B would be multiplied by the number of replica sites. So given a workload with a bandwidth dependent I/O phase and a parallelizable phase, the model may be applied to predict the turnaround performance.

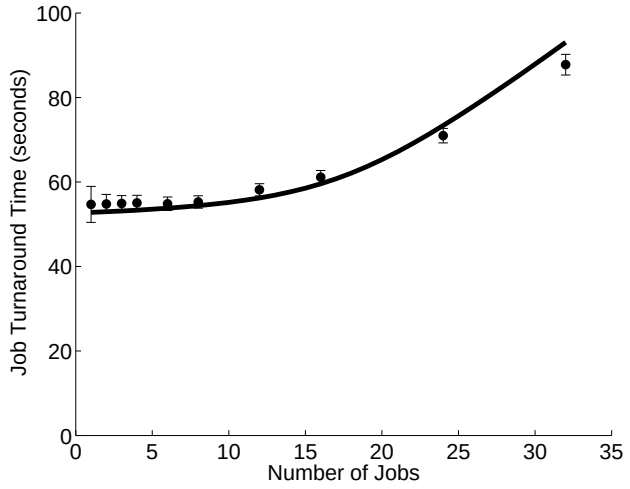


Figure 5.6. Total job turnaround time for the analytical model.

System parameters: $J = 50 \pm 10$ seconds, B=10 MB/s, and S=280 MB. The model is plotted as a curve; actual runtimes were averaged at the points shown. Error bars indicate a 95% confidence interval.

5.2.4.4 Queueing Theoretic Analysis

Our model for the data consuming application here essentially models a set of jobs which alternate between two states: reading and processing. This type of model is commonly analyzed through the use of queuing theory, a form of mathematical modelling focused on the abstract behavior of systems of customers and services. This technique originated in telecommunications networks, where it is commonly employed today, and is also useful in economic models, workflow analysis, and other applications. In a general queueing system, customers move from state to state as described by a deterministic or probabilistic routing protocol. At each state, customers wait for service in accordance with a given queueing discipline. Queuing theory allows the analyst to form and answer useful questions about the behavior of the system, such as:

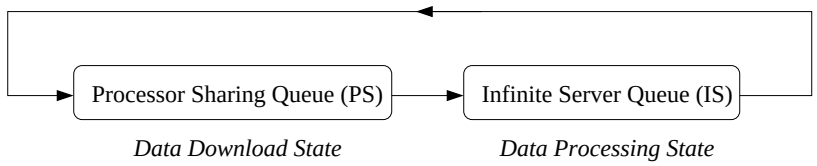


Figure 5.7. Queueing theoretic model for data consumption jobs.

- Where do the customers spend most of their time?

  This question may be answered by obtaining the steady-state probability vector (π).

- What is the mean number of customers in a given state?

  This question may be answered by obtaining the marginal probability that a given service state has a given number of customers waiting, and applying averaging.

The turnaround time measurement above may be analyzed by applying a queue model as shown in Figure 5.7. In this framework, the customers are user jobs that progress around the closed loop system, moving from the data download state to the data processing state and back. The data download state is modelled as a processor sharing (PS) queue: the download rate is shared among the jobs in that state, thus the service time is proportional to the number of jobs currently downloading data. The data processing state is modelled as an infinite server (IS) queue, thus the service time is deterministic.

Queueing system analysis is often based around exponential service times and standard First-Come-First-Serve (FCFS) queues. Using Kendall's notation, the PS queue is a M/G/1-PS queue and the IS queue is M/G/∞. However, these exponential service times only roughly model the semi-deterministic behavior of predictable user downloads and processing tasks. In the remainder of this section

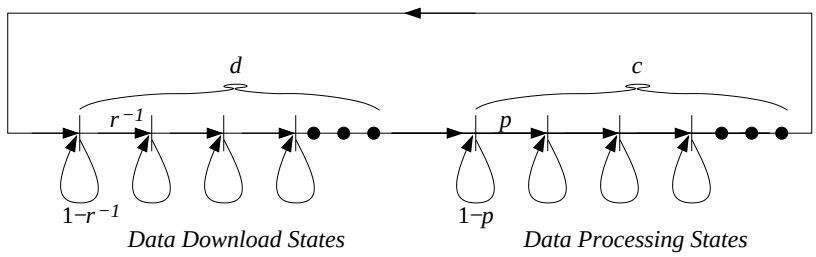


Figure 5.8. Discretized Markov model for data consumption jobs.

we address these approximation problems and demonstrate how models for the data consumption application may be obtained using known techniques.

Markov Modelling

In this first model, we apply a discrete time Markov chain (DTMC) [23] to the application. Each job moves forward through a Markov chain, progressing from discretized download states to discretized computing states and back, as shown in Figure 5.8. There are d download states and c computing states for each of the n jobs. This state space may be simply multiplied out in an "n-dimensional" Cartesian manner. To simulate the effect of random access to shared network, a downloading job makes progress with probability r^{-1}, and does not with probability $1 - r^{-1}$. Note that r may be easily obtained at each potential state by inspection. To simulate the effect of a user job competing for CPU time against ambient local tasks, a job makes progress in a given timestep with probability p and does not make progress with probability $1 - p$.

The expected turnaround time for jobs in this network may then be found by obtaining the probability distribution vector of the DTMC system. Then, the mean value of r may be obtained by summing to obtain the probabilities of each possible value of r and applying averaging. However, obtaining the probability distribution would require powering the DTMC transition probability matrix until

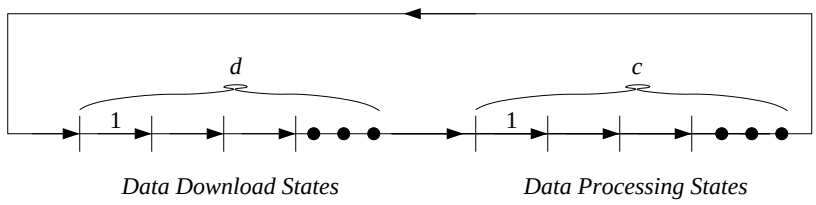


Figure 5.9. Cox distribution model for data consumption jobs.

convergence. This square matrix is $m \times m$, where $m = (d + c)^n$. The number of nonzero entries in each row is $O(2^n)$ as each job may or may not make progress. Thus, this may be seen to be a very expensive technique for this application.

BCMP Networks

The previous approach is problematic for its arbitrary discretization of the substates as well as its large computational cost. A continuous time Markov chain (CTMC) may be applied to avoid the arbitrary discretization while still obtaining the rich state probability information. This more accurate model may be obtained through the use of the highly generalized BCMP theorem (named for the authors) [16], which allows for multiple queue types and provides closed form probability results. To obtain our near-deterministic state transition times in this framework, we use a Cox distribution function as shown in Figure 5.9. In this service time distribution, the download substate passes from $d_i \rightarrow d_{i+1}$ at in an exponential service discipline with mean service time 1. The mean total service time through the d substates is then d, with variation smaller than 1. A similar distribution is given for the c compute states.

The probability distribution π for the possible states of jobs in the system is then obtainable [23] by evaluating

$$\pi(S_1, S_2) = f_1(S_1) f_2(S_2), \text{where} \tag{5.4}$$

$$f_1(S_1) = k_1! \prod_{l=1}^{d} \frac{1}{k_{1l}!}, \tag{5.5}$$

$$f_2(S_2) = \prod_{l=1}^{c} \frac{1}{k_{2l}!}, \tag{5.6}$$

and where S_i represents the number of jobs in each substate of state i, k_i is the number of jobs in state i and k_{il} is the number of jobs in state i, substate l.

This method may be used to obtain a more accurate model of the probabilities but still samples each possible job/state situation and requires heavy sampling of π to obtain aggregate performance information. As shown above, only the expected value of r is required to obtain meaningful system performance prediction.

Mean Value Analysis

Our third queueing theoretic approach emphasizes the need to simply obtain mean values of important performance characteristics at the expense of the full probability distribution. The mean value analysis (MVA) [113] allows the mean number of jobs in each state to be obtained *via* a recursive technique. The framework models the mean value of the turnaround time or total service time ($\bar{T}$) observed by a job in a state to be equal to the service time of that job alone plus the service time of the mean number of jobs at that state ($\bar{K}$) in a system with $n-1$ jobs.

Following the MVA outline in the simpler case of exponential service times [23], the method starts by formulating the form of the mean service time at each

state, in our case,

$$\bar{T}_1(n) = \mu_1^{-1}(1 + \bar{K}_1(n-1)) \tag{5.7}$$

$$\bar{T}_2(n) = \mu_2^{-1}, \tag{5.8}$$

where μ_i indicates the mean service rate at state i. Then the throughput (λ) of the loop system is formulated as

$$\lambda(n) = \frac{n}{\bar{T}_1(n) + \bar{T}_2(n)}. \tag{5.9}$$

Finally, the form of $\bar{K}(n)$ is obtained, by multiplying

$$\bar{K}_i(n) = \lambda(n)\bar{T}_i(n). \tag{5.10}$$

This method thus obtains the mean system observations without analyzing the whole Markov system or obtaining all state probabilities. However, the method is considered memory intensive [32] and must be additionally extended to handle more general queueing disciplines.

5.3 Parameterized Workflows

Managing data on a complex, dynamic underlying infrastructure could be very difficult in the large scale applications covered in previous sections. The GEMS approach enables the combination of its general-purpose metadata catalog as described in Section 3.2.4. GEMS seamlessly combines metadata programming with distributed job/data parallelism as described above in Section 5.2.3. Experimental scientific workloads are more often driven by adaptive parameter investigations

than well-known static sequences. Existing popular grid programming models include the parameter sweep and the workflow. In a *parameter sweep*, the same computation element is independently performed using each eligible point of some parameter domain as input. In a *workflow*, a partially ordered set of data movement and computation elements is run to completion. An existing example of a hybrid model is the *parameter optimization*, in which the output of the computation is optimized within a given parameter domain.

While these models are extremely powerful and encapsulate a variety of applications, they fail to capture other important computations such as postprocessing analysis, query-based computing, and interactive computing. Given a computation $y \leftarrow P(x)$, examples for these cases include:

- "For each completed simulation in category C, compute the average of state variable y_i and store it."
- "For each completed simulation $P_C(x)$, if the output matches Q, re-run the simulation with parameter modification ∂x."
- "Plot the current state of each simulation. Restart a user-selected set of simulations from their last checkpoint after altering state variable x_i."

Workflows built within the GEMS framework are supported by a the ability to parameterize workflows, thus seeding the resulting execution. While existing programming tools such as shells support all of the operations performed by workflow tools, workflows are useful for three software engineering-related reasons:

1. **Encapsulation:** Operations performed within a workflow task form a logical group.

2. **Clarity:** The task dependency structure may be described by a simple graph and and the state of of a running instance may be similarly diagrammed.

3. **Restartability:** Partially completed workflows may be restarted based on previously completed, safely stored work. The ability to quickly determine the minimum amount of new work that must be performed to renew an attempt to achieve a target or explore a new target limits the damage done by faults and enables exploratory interaction.

Restartability is of critical importance to large scale batches executing on unreliable resources, as exemplified by the TPS case in Section 3.4.1. Clarity and encapsulation are beneficial when implementing complex batches from applications such as hyperdynamics in Section 3.4.2.

Any new workflow system functionality should not impede these abilities: without them the user might as well use a fully functional programming language. Thus, the parameterized workflow alters this model in only a few well-defined ways First, workflow targets are parameter tuples, not files or operations. The existence of these may be easily queried as shown above. Second, workflow targets may be parameterized, resulting in function call-like tasks. This results in several problems that must be solved by the new framework. Task parameters must be arithmetically manipulated and propagated into the actual task execution. Additionally, operations within workflow targets must be properly parameterized. Inputs and output file locations are also parameterized values that must be linked into the user computation.

5.3.1 Workflow Formulation

Consider a batch of scientific simulations S, over which several random seeds may be used, and the simulation time is broken into manageable, checkpointable segments. Each evaluation $S(u, r, t)$ of the simulation is defined by a user name u, a random seed r, and a time segment t, for m random seeds and n segments. Each evaluation depends on the previous evaluation with respect to time but within the same random seed group.

In a static workflow system the user would have to generate a Makefile-like script containing $m \times n$ workflow tasks with hand-parameterized filenames and other objects. In a parameterized workflow system this may be framed by simply stating that $S(u, r, t) : S(u, r, t - 1)$; that is, each segment of execution time is dependent on the previous. A base case $S(u, r, t = 0)$ is inserted into the system as an initial simulation state or base case. A target $S(sorin, 312, 100)$ may then be specified, the system would generate the 100 resulting tasks and execute them. Parameters are passed into user code by filtering a user-specified configuration file with `sed`-like operations, then the user task is executed as specified by the workflow node. Thus each workflow task must contain a header, a dependency list, a mapping from header parameters to metadata values, a mapping from configuration file tokens to header parameters, and an execution string. The resulting example workload fills a rectangular parameter space but includes dependency information as shown in Figure 5.10 a).

More complex, interactive workflow-like structures may also be simply instantiated within this model. Consider the same molecular simulation example with the addition of user steering: the user may modify the force fields applied withing the experiment, thus forking a new trajectory through the parameter space as

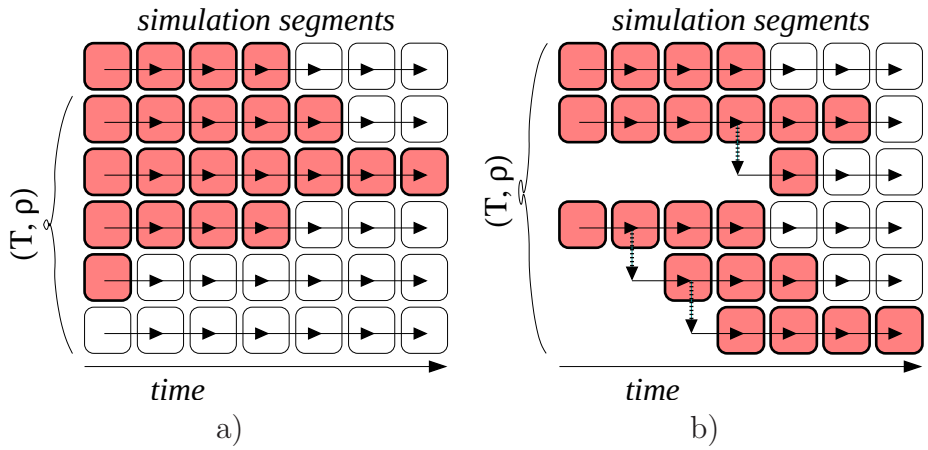


Figure 5.10. Parameter sweep and workflow diagrams.

a) In a parameter sweep [2, 25], user jobs typically fill a square parameter space. b) To allow for interactive parameter execution, the system has to allow the dynamic user creation of execution branches.

shown in Figure 5.10 b). This model satisfies the complexity required by the data flow inherent in the hyperdynamics method, which requires the propagation of information from multiple ancestor computations.

In this case, user requests simply create new targets and tasks are launched with respect to the dependency state of the whole workflow. The tree-like search structure is a simple consequence of a minor change to the parameterized dependency rule, such as

$$\begin{aligned} &S(u, r, t, \text{branch} = p) : \\ &\qquad S(u, r, t-1, \text{branch} = p) \\ &\qquad \textbf{or } S(\text{identity} = p, \text{time} = t-1). \end{aligned} \tag{5.11}$$

Thus a branch parameter is added and the segment identity code is referenced. Segments depend on either the previous segment in time *or* the branch point

when appropriate. Thus a simple, automated *data sweep* may be made through the parameter space to fulfill the user targets.

5.3.2 Scalable Parameterized Job Submission

In this section we consider a real-world approach to the parameter exploration problem. In this approach, a branch-without-bound algorithm is applied to explore all possible paths through the hyperdynamics parameter space. This algorithm tests the architecture of the system as it consumes resources exponentially over time as the branches expand. While the computation is easily parallelizable in our application, a distributed scientific repository is employed to store the simulation results (segments) for reuse in the parameter exploration as well as archival purposes, any limits of which create a bound on the branchability of the search.

While data movement in GEMS is entirely parallelized, certain operations within the system rely on a centralized component that ensures synchronization and a consistent, searchable scientific namespace. Four potential operations must be considered: data insertion, metadata search, retrieval, and notification. The creation of a new config in hyperdynamics can involve multiple executions of each, depending on the segment concerned and the state of the computing site.

Data insertion and retrieval are indexed operations that allow for the use of unique database keys. Insertion simply inserts the file and application metadata into their respective tables and refers the client to an appropriate reserved storage location. Data movement proceeds while the server is free to perform other work: upon completion, the server is notified and verifies the file statistics. The ensuing data replication is an asynchronous server process that does not significantly compete with client operations. Likewise, data retrieval is an indexed operation

that refers a client to a storage site at which the requested files may be accessed.

Parameter search, however, is a non-indexed search of simple application-formatted metadata tags and values. This more costly operation is a highlight of our system as it enables the scientific filesystem abstraction used above, however, it exacts a centralized cost on the distributed algorithms used. Notification allows a client to be notified when a record exists matching a given parameter query; our framework provides a disconnected, scalable callback mechanism. While this operation is also non-indexed it is less of a constraint in practice as the number of outstanding notification requests is small.

Notification and job start control create resource management issues when exploring a non-rectangular parameter space (Figure 5.10). In our application, as a job proceeds, resulting data records enable additional computational parallelism. Several potential job start mechanisms could be employed:

1. *Node wait:*

   A job is submitted for each node in the search space (or, as an optimization, for each horizontal row). However, in the exponential search case, more than half of the nodes would spend more than half of the time waiting to be notified, reducing system utilization by more than 50%.

2. *Row wait:*

   A job is submitted for the first row. Upon completion, jobs are recursively submitted for each possible branch generated by that row. This job-oriented approach is supported by systems like DAGMan. However, each node must be represented by a job, again clogging the scheduler and reducing the bulk in-place processing enabled by the row-oriented approach, multiplying data movement costs by a factor equal to the number of segments. Alternatively,

the job could wait until the row is complete before spawning branch rows, however, this would reduce the available parallelism by a factor equal to the number of segments.

3. *Client wait:*

   The client manages all notifications. The client spawns a blocked local task for each possible node; upon notification the task proceeds to submit the appropriate job. However, this simply transfers the scalability questions from server-side procedures to client side tools.

In our framework, we employed a simple client-side tool that enqueued potential blocked local tasks, and a small number of these tasks were actualized in a bootstrapping step. Tasks were promoted into and out of the queue upon any notification into a small finite pool of actual blocked tasks, in an order consistent with expected notification time (based on segment number). When blocked tasks received their notification, they proceeded to job submission. This approach prevented the construction of a complex multi-channel client-side service, while enabling a great deal of computational parallelism. This bottlenecking procedure is diagrammed in Figure 5.11.

5.3.3 Workflow Performance Analysis

In this section, a scalability experiment is reported using the GEMS replica management system, the Condor computing system, an underlying network of Chirp file servers, and the HYD molecular hyperdynamics toolkit developed for this work. The implementation of a full search of all possible paths through the parameter space was submitted to Condor, using the GEMS/Chirp storage system for all algorithmic data operations.

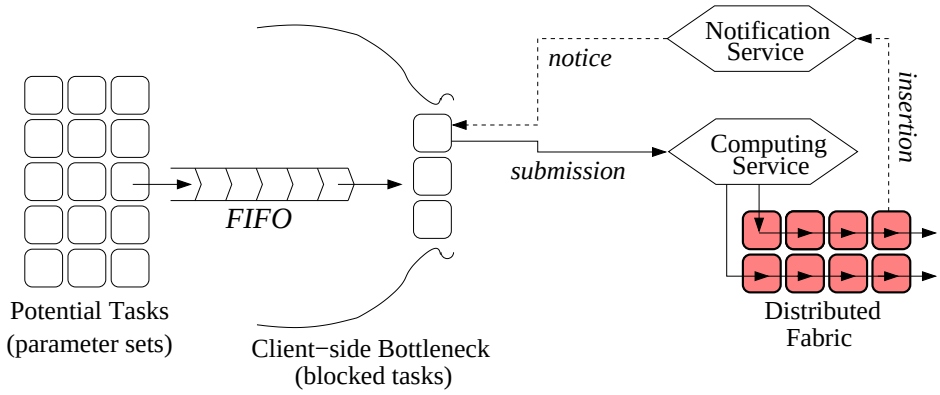


Figure 5.11. Bottlenecked job submission model.

Local tasks depend on a record-based notification service and are released to block at the bottleneck at which point they register for notification in a controlled manner.

The computing infrastructure at Notre Dame upon which GEMS and Condor rely is a widely heterogeneous group of 540 Intel and SPARC processors, of which only Intel were used for this experiment. Reported MIPS performance ranged from 346 to 5696, and data movement was coordinated with job location in accordance with GEMS techniques over on-campus LAN connections. The GEMS server, the potential centralized constraint on distributed performance, operates on a Postgres 8.2.4 installation of two 2.4 GHz Opteron processors[4].

A full search of 50ps segments at all possible bias levels was performed up for a total path length of 10 segments. The simulator operated on a 400 atom gas phase Argon system at $T = 73$ Kelvin, $\rho = 0.1066$ g/cm^3. This method produces 2^{10} segments as each possible branch is taken, with each segment totalling about 6 MB. Condor usage and database load levels for Postgres processes are shown in Figure 5.12; both metrics were sampled at 1 minute intervals.

[4] `gems.cse.nd.edu`

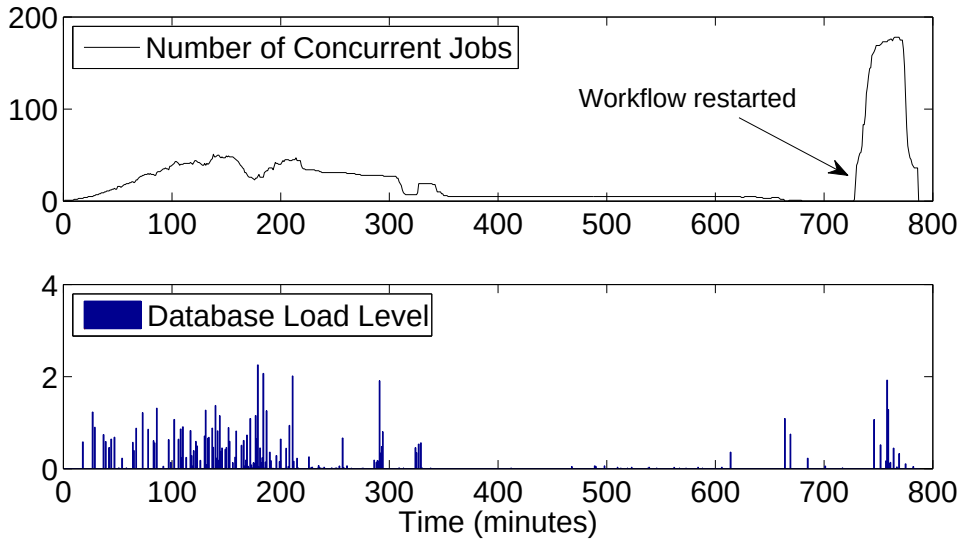
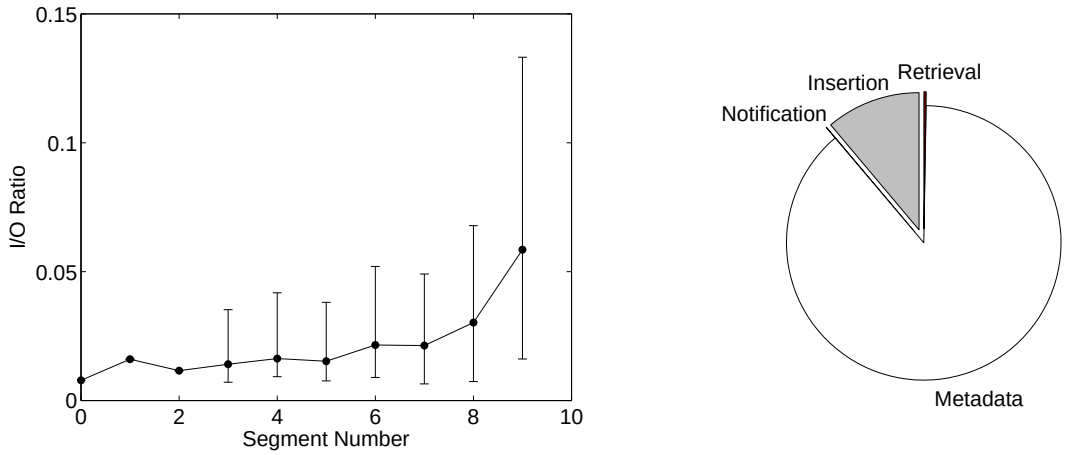


Figure 5.12. Condor usage and database load levels for the all paths hyperdynamics experiment.

Upon termination of the original workflow at minute 700, 40% of the required segments were stored. Note the slow growth in parallelism due to branch jobs entering the ready state over time, an irregular process due to compute site heterogeneity and other factors. The workflow was restarted the next day to finish off the set. The preponderance of branch tasks were immediately sent to the computing system as their dependencies were already satisfied, however, a great deal of metadata processing was required to probe, locate and obtain the requisite data segments. Since little or no processing interleaved these metadata operations, the metadata server was overwhelmed, forcing jobs to retry[5], timeout, and fail.

Running tasks alternate between I/O operations and computation. Here, I/O operations consist of centralized metadata operations as well as data movement

[5]GEMS clients employed a simple retry mechanism whereby they retried after 1, 2, 4, and 8 seconds, then failed.



a) b)

Figure 5.13. Performance results for the hyperdynamics workflow.

a) I/O Ratio I (Equation 5.12) per segment. Averages are reported with 95% confidence intervals where appropriate. b) Centralized time consumed on the replica management server per operation.

operations. A checkpoint, for example, consists of a metadata insertion, a reservation, parallel data movement, and record committal. Computation is fully independent per node. An important measure of system congestion is thus the *I/O Ratio* (I), computed as:

$$I = \frac{\texttt{Time spent in GEMS clients}}{\texttt{Time spent in computation}}, \tag{5.12}$$

where computation includes ProtoMol and Matlab operations. This ratio is plotted *per segment* in Figure 5.13 a). Thus, the scalability in terms of number of leaves on the search tree is considered. Plotting the ratio in the segment domain indicates the scalability of the search algorithm.

Additionally, individual operations were profiled for performance for a similar, shorter run up to segment 4. The server was already managing results from

previous users, totalling about 75,000 configs consisting of over 7,000,000 replica locations. The relative server response time as measured by the server for each operation is shown in Figure 5.13 b). Metadata operations were the most common operation during the run and were also the most expensive.

This experiment demonstrates that while a centralized metadata system may be employed to efficiently manage workflows that interleave metadata operations with computation, when all tasks request intense synchronization and metadata information simultaneously, performance is greatly affected. Thus while a full automated search of the parameter space is potentially possible under nominal conditions, selective pruning and interaction with user-identifiable parameter regions should be attempted to gain better resource utility on practical systems.

5.4 Grid Integration

A founding design feature of grid computing is the ability to allow access resources to users across administrative domains. An administrative domain may be commonly conceived as a local UNIX installation managed by a system administrator. Grid construction allows users outside the local system to run jobs or allocate storage on the resource. A system administrator may enable this by first creating a local user that has access to these services, then installing a grid software system that runs as this user. The new system maintains an independent authentication and authorization scheme intended to scale to many users. For example, the Globus system includes a public key authentication system combined with a virtual organization model to authorize grid operations.

This solves the basic grid problem but has certain limitations. Positively, it allows a new abstraction layer - the grid security system - above the operating system

layer in a scalable way. However, it shoehorns users into a single authentication scheme orchestrated by administrators, not users. This adds to administrative workloads while restricting the ability of users to share their access to resources with others. Typical systems do not allow users to grant privileges to another user without compromising their own account, by revealing a password or private key. Access control lists allow the addition user names and permissions but are an authorization scheme. GEMS provides a method to turn arbitrary access control lists into a globally visible authentication scheme as well.

Consider a case in which research group leaders from distant universities desire to construct a relatively secure cooperative database, building upon existing (grid-enabled) resources. We call this the *grid integration* problem, which is concerned with constructing new grids from existing grids or their fragments, ultimately creating a grid-of-grids. In the pre-grid era, administrators would have to agree on a global user list and propagate it out. Using grid tools, they could establish a global security authority and ensure that all components agree to use it. Both solutions are problematic, as global user lists are difficult to manage, and a globalized security system may be overwhelming, limiting the ability of users to create subgroups or employ previous methods (such as UNIX or DNS authentication).

5.5 Grid Derivation

For any of these complex tasks, the user must be able to specify at some level where they will run and where the data will be stored. The controller model takes a high level view on this process. In the distant university case, the construction of the cooperative distributed database should not be taken to mean that within the database all records are equal from a security perspective. In fact, many cases

could arise in which users could agree to share resources, say, in a pair-wise way within the greater structure. This would involve selecting certain resources for use in the derived system and ensuring the security system works for the users and systems involved: a local procedure which should have no global side effects. This process again creates a *data landscape* in which a subgroup of users have access to a subset of possible resources *and* data records. We call this the *grid derivation* problem, which creates a record-specific grid-within-grid abstraction.

Grid derivation starts with the well-known process of matchmaking, a process analogous to selecting rows from a database table. The process is augmented because it results in a new environment: one in which other users can participate. Our simplified implementation begins with a small scale user process: the selection of the requested resources. Resources may be organized into clusters reflecting network topology or geographical distance to enable certain functionality described below. This information is organized into a *storage map* and may be stored for later use.

The creation of a data landscape begins when a user combines a data record containing data files and metadata tags with a storage map and access control list (ACL), resulting in a *config*. A typical use of a config is the input and output files of a simulator program, combined with metadata such as the options passed to the program. This config is instantiated registered with the greater system and entrusted to its control.

The storage map defines the derived grid resources upon which the data files will reside, defining the data landscape in terms of system-level security and data movement performance. The ACL is propagated to these storage sites and applied to the appropriate files. While this may be performed with existing tools, an

important challenge remains: how does information from the config propagate up to the greater system? Users must be able to administer their data at the global level. For example, they must be able to delete a config and its underlying widely replicated files. To do this they must authenticate to the greater system. The ability to provide meta-grid control of diverse resources will constitute our solution to the grid integration and derivation problems.

5.6 The Rendition Protocol for Access Control

The process by which a user performs authenticated operations within the config data landscape is called the *rendition protocol*. This protocol applies in systems that implement the grid controller model, in which a controller manages the global policy but is agnostic with respect to system specifics such as a password list. In this case, the system may only interact in an authenticated manner with underlying physical storage sites that implement direct authentication protocols. Upon this fabric we intend to build an indirect protocol that enables the controller to authenticate a channel for a certain user with respect to a config.

Thus we reiterate our data-driven focus: operations in the system change the data landscape. Critical operations at the controller level must be authenticated through a storage site for a config, because only here is the ACL enforceable and the data stored and protected. The controller model explicitly allows users to use old protocols to interact with the underlying system, relying on direct authentication methods.

The rendition protocol was first presented in 2006 [159].

5.6.1 Overview

Distributed computation systems have become an important tool for scientific simulation, and a similarly distributed replica management system may be employed to increase the locality and availability of storage services. While users of such systems may have low expectations regarding the security and reliability of the computation involved, they expect that committed data sets resulting from complete jobs will be protected against storage faults, accidents and intrusion. We offer a solution to the distributed storage security problem that has no global view on user names or authentication specifics. Access control is handled by a rendition protocol, which is similar to a rendezvous protocol but is driven by the capability of the client user to effect change in the data on the underlying storage.

The rendition protocol solution allows users to obtain an initially anonymous channel to the controller. The user may then request access to modify a config, at which point the controller creates a challenge which must be satisfied for the execution of the operation. This challenge typically takes the form of a file operation with respect to the storage site and ACL in question, such as the creation of a numbered marker file in a directory from which all users are restricted except those known to be authorized for the operation by the ACL. Upon the hand over of the file, the controller is notified to inspect the satisfaction of challenge, carry out the operation, and return the appropriate notice.

This protocol may be thought of as similar to other existing indirect protocols that delegate authentication to an external authority, but in fact rendition offers greater efficacy. The simplicity of the scheme allows ordinary users to delegate authentication to the whole storage fabric, a diverse array of heterogeneous sites, each of which may implement a subset of the available physical authentication

protocols. This derivation of responsibility for security enables users to make use of local system knowledge to create *ad hoc* collaborative systems without global consequences.

5.6.2 Discussion

A variety of options exist to parallelize and distribute storage over clusters or grids, but several technical and organizational issues must be considered. Current computational environments span multiple administrative domains, such as two or more university clusters combined into a unified resource to increase the utilization of compute resources. A current authorization strategy in such systems is to execute these jobs as a nearly privilegeless anonymous user that has access only to pre-specified remote files over the network.

Distributing storage across the compute system in such a setting presents additional difficulties. Users may be willing to allow the anonymous third-party system to execute jobs on their behalf, but will need to be able to access the storage directly upon data creation. A strategy to solve this problem would be to create a master user list and replicate it to all storage sites, allowing user access over a pre-defined protocol. Managing users in such a system becomes extremely burdensome as the user list would contain members from all collaborating institutions.

More advanced methods have been proposed including grid authentication protocols that take into account the existence of distributed administrative domains. However, choosing one such protocol implies that all users must agree to the protocol, and would find it difficult to fall back on simpler, localized, pre-existing schemes.

Consider the case of an *ad hoc* collaboration between researchers at two dis-

tant universities. Each contributes storage servers to the project, and one of the researchers installs a replica management system to synchronize data sets between the sites. This researcher may be unwilling or unable to provide accounts for all eligible users from the other university to interact with the management database.

Similar problems arise in many experiences with cooperative computing, and the premise of the approach presented here originates from typical assumptions and properties of this situation. *Users and authentication methods employed are considered secondary to the ability of users to modify the data sets and servers involved, and the ability to authenticate via a given protocol as a given user is less important than the ability to access a given data set at a given site.* This observation motivates a system that operates at high level, independent of protocols and user names, and can stay within its purpose: the management of distributed data sets. The centralized service then acts as little more than a guide, coordinating interaction among users and storage sites. These user-site pairs handle security in a pairwise way, and system changes are propagated up to the management system.

As a solution to our considered problem, client tools could be employed that interact with the *local* storage service as a method of proving their identity to the greater system. This allows researchers to simply administer their own machines instead of a grid, and allows the replica system to simply manage replicas instead of users.

5.6.3 Assumptions in Shared Commodity Systems

Shared replica management systems have been designed to meet the storage requirements of users requiring a variety of functionality. Users benefit from increased storage space: especially short term storage space, as the shared workspace

may increase utilization of the underlying systems. Additionally, replicated data sets are more resilient. User groups that desire to publish their data inside a virtual organization benefit from cataloged replica systems, which provide a searchable catalog of metadata and allow for data sharing and reuse.

Such systems have a certain typical set of user requirements and assumptions. First, users must basically trust the machines that they are borrowing, the network, and the administrators from whom they obtain these resources. Specifically, the model here assumes that if a user is willing to delegate computation to a site, then that site may be trusted to serve the output data. Additionally, there is an assumption that the system will be in a partial failure state at all times, with some machines unavailable for a variety of reasons. Middleware computation systems attempt to build this fragile infrastructure into a useful resource through fault-tolerant checkpointing and job restarts; storage solutions such as ours presented here attempt to create a viable, scalable, and secure storage solution through replication and replica management. Finally, the data sets involved are of relatively low value, for example, data transmission is typically performed in the clear. Note that this does not allow us to assume that the data sets are intended to be publicly readable. These assumptions correspond to those taken by users of commonly available opportunistic computation systems such as Condor.

The protocol described in this section attempts to utilize this fundamentally unreliable and uncontrollable resource fabric into a secure system for communication between previously unauthenticable users and a replica management database while keeping data secure and providing exceptional flexibility.

5.6.4 Properties of Access Control in a Replica System

Our relevant system architecture consists of a network of storage devices widely distributed and independently maintained. The storage sites may be configured in various ways, deploying different subsets of the available connection protocols. The storage providers desire to collaborate - to obtain the benefits of a replica system - and thus allow limited access to a centralized service replica management system (RMS). Users may be able to authenticate to some subset of the available storage sites using one or more protocols. The set of users may be multiplied in a set of pairs $(protocol, name)$, called the set of subjects. Each real-world user corresponds to multiple subjects, based on the accounts and account types available. However, the RMS is not aware of any global list of subjects in advance. Additionally, users may desire to access the data that they may store in the system over more than one protocol.

Centrally, the RMS catalogs a large number of storage devices as they advertise their resources to the system, but cannot manage the lists of users from various domains. The services offered are unable to authenticate users over the network. The immediate result is a simple read-only lookup service that guides users to data sources.

The storage sites implement a variety of authentication protocols, and are drawn from multiple administrative domains. Users may access data in the system by following a metadata lookup to the RMS with a connection to an appropriate replica site, after which they may obtain the required data file. In a large, multiple organization system, users desire to gain access to multiple domains using multiple protocols, which is acceptable because the user/server relationships may be defined by the users.

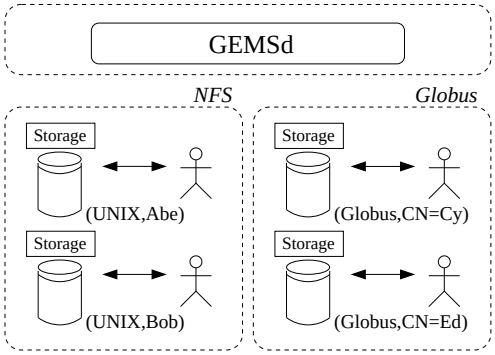


Figure 5.14. Read-only access controlled by local storage servers.

To effect change in the system, such as to delete a record or to modify a metadata entry for a record, two systems must be protected: the metadatabase and the servers. This represents a challenge to the multiple domain model, because the metadatabase and the storage server may have different conceptions of who the user is. For example, a user may be able to authenticate to a nearby storage site that contains a replica of a data set that the user wishes to delete, but the user is unable to authenticate to the centralized server- a necessary ability to delete a replica set that may be owned by another user and distributed widely.

No global list of users is available to the RMS, since the RMS is unable to implement an authentication test for each subject. However, a user may demonstrate the ability to modify a stored data set by interacting with a storage site. Since the RMS is able to observe such interaction, an indirect method of authentication is possible. By demonstrating the ability to change the storage system, the user has authenticated in a meaningful way to the RMS.

5.6.5 System Specifics

In this section, we develop the specific capabilities of the software components relevant to this work. The data structure of a data set stored in the system is shown as a diagram in Figure 5.15. Each data set, or *config*, is indexed by a numeric key, but is commonly accessed by a unique metadata lookup query. Configs have an owner and ACL as shown: owners always have full access to the config and may grant full or limited access to other subjects. The config consists of any number of files with their path information, as in an archive. The *storage map* indicates which storage sites are eligible to receive and serve the data files. Additionally, the map indicates the cluster topology to the replication service. The replication service uses this information to split replica locations among available clusters. The map is also used when retrieving data to obtain a nearby replica. This structure increases the performance of data access, and increases data survivability when whole clusters may be offline.

The map takes on critical importance in the security of the data set. Users may use the replica servers as rendezvous locations when gaining access to the config in question, so the servers must be trusted *by the specific user* to enforce the ACLs appropriately.

The system allows the users to create virtual workspaces in which they create storage and grant access control to other users. *The benefit of the methods described here is that the centralized service does not have to maintain a list of these users, or even be able to authenticate them directly over their desired protocol.*

Different operations require different types of authentication in this system. There are four authentication methods that are implemented by the RMS server.

- **none**, to access the public search facility;

```
Metadata     Config Key     Config Entry

                            owner   (UNIX,Abe)
name=Abe
sim=chem  } 4321 {          acl     (Other,Jim,read)
p=3                         map     my_cluster
                                    └──── abe*
                                    jims_cluster
                                    ├──── jim01
                                    └──── jim02
                            files   bin/chem   (2)
                                    ├──── abe01
                                    └──── jim02
                                    data/stats (3)
                                    ├──── abe01
                                    ├──── jim02
                                    └──── jim01
```

Figure 5.15. Metadata layout with ACL.

A user-specified number of copies of each file are stored across the storage map, all are protected by the same ACL.

- **insert**, to insert a new config into the system;
- **config**, to modify or delete a config;
- **admin**, to rewrite the configuration of the entire system.

The most common use of an archival system is to consume data from it to perform new computation. While the data files are protected on the storage servers, locating this data is a publically available service. Searches may be used to map metadata to configs, or configs to file replica locations. This is equivalent to simply observing a resource catalog.

Data insertion is performed by client tools after authentication using the rendition protocol. The client simply contacts the metadatabase with a request for a challenge, and specifies an desired authentication method. The response to the challenge takes is a rendezvous location on a storage server that the metadatabase administrator and the user trust to allow insertion into the system. The

rendezvous location takes the form of a tuple containing an appropriate storage server, a new config key to uniquely identify this data set, and randomly generated code number for this transaction: (*host*, *config*, *code*). The server creates a directory on the given host named `/<config>/RDVS/<code>`, and sets the ACL on this directory to allow the client write access. The client then contacts the storage server and creates a marker file at the given path. Upon completion, the metaserver is notified over the channel, the result is verified, and the channel may be thought of as authenticated.

One the config has been created, the ACL is generalized to meet the owner's request allowing access to other eligible subjects. Such subjects do not need to authenticate to the centralized service as described above. The replication process, coordinated by the metadatabase, propagates the ACL to other eligible storage sites along with the data files.

Metadata modification or config deletion is performed upon satisfaction of the config rendition protocol. In this protocol, the client contacts the metadatabase server and indicates which config is to be affected. The reply takes the form given above, but the host given is selected from the storage servers that currently contain replicas of the stored data. Additionally, the rendezvous directory is placed within the directory allocated for that config. Thus the protocol takes on an element of realism: clients that can demonstrate access to the stored data files are granted access to modify the metadata associated with the config.

The authenticated subject is not a full representation of the user subject in the traditional sense of a login. Since the subject has been authenticated *for* a certain config, the authenticated subject must be considered a limited subject (*protocol*, *name*, *config*), as access has only been granted by the system for the

config in question.

If a certain storage site is authorized to serve replicas of a given config, the storage site effectively acts as an authentication authority for the user with respect to the configs which it stores. Thus the site effectively *speaks for* the user with respect to that config. As a result, users must construct storage maps appropriately, weighing the benefits gained from "broader", more distributed maps with the security risks involved in spreading data far and wide. In typical cases, this is equivalent to or easier than constructing a matchmaking script like those used in common computation systems. When the storage network *is* the computation network this is elementary.

A last protocol is available to allow administrators to modify behavior of the whole system during operation, allowing for remote administration. The system configuration specifies a rendezvous location that may be used to satisfy the protocol, and upon satisfaction of the protocol, a new configuration file may be fed into the server.

A generic form of the protocol is outlined in Figure 5.16.

5.6.6 Application

Again, we consider the university network as a commonly used tool for scientific research. Here we provide an example to motivate the usefulness of the new system. The Notre Dame GEMS and Condor installations are employed as the testbed. Since all the storage hosts in question share a network file system, the UNIX authentication protocol is a common choice for a GEMS authentication method. Users will typically allow hostname authentication as a fallback access method, and Globus authentication is possible via a local certificate authority.

Rendition Procedure for Storage Access

	Client operations	*Metaserver operations*
1.	Client obtains an anonymous secure channel to the metaserver.	
2.	Client requests access to config c as user n.	
3.		Metaserver issues a rendezvous challenge involving a host h in the map associated with c.
4.	Client authenticates directly to host h using method m.	
5.	Client satisfies challenge.	
6.	Client notifies metaserver that challenge is satisfied.	
7.		Metaserver inspects host h for completion of challenge.
8.		Metaserver notifies client that inspection was successful; the channel is authenticated as $m : n$ *for* c.
9.	Client transmits metadatabase operation regarding config c to metaserver.	

Figure 5.16: Outline of the rendition protocol.

The Notre Dame Condor system has been combined with Condor systems at nearby universities to increase the opportunities for resource sharing and collaboration. To unify the storage into a single logical system as well, users would expect to have to be able to authenticate under some uniform protocol. However, the GEMS protocol allows users to interact with the system by authenticating through only the subset of servers that are relevant to the data set in question, relieving the administrative burden of managing users in a global way.

The GEMS system allows replicas of user data sets to be automatically replicated to remote computation sites. Users may use a simple storage map to inform the system that the `*.nd.edu` and, for example, `*.purdue.edu` computation and storage resources are eligible to be used. The map would be used to ensure that data access would be localized to a campus network.

In practice, the GEMS installation could not be immediately configured to authenticate users from multiple universities. However, as individual stake holders allow each other access to storage sites at their respective locales and develop a complex system of user groups and virtual organizations, GEMS is able to grow in parallel with the system, relying on storage providers to serve as the authority on storage access, which is the objective all along.

A more abstract example of the protocol in action is shown in Figure 5.17. In this special case, we demonstrate the additional ability of the system to authenticate users to whom access has been granted on a pre-existing config. In this case, user Abe has stored a config on his server, but has deployed an ACL that allows user Guy full access using the "Other" protocol. Abe's environment is capable of authenticating UNIX users and Other users, however, the metadatabase is unable to authenticate Other users. The config in question has been replicated, perhaps

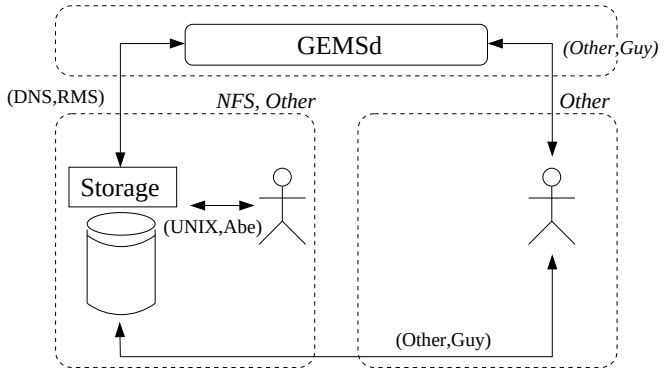


Figure 5.17. User authentication at a rendezvous point.

to other domains, so deleting the config must be performed at the RMS (GEMSd) level. Since Abe has allowed Guy full access, Guy should be able to perform deletion but is unable to gain config access at the RMS level. However, the client tool that performs deletion using the rendition protocol is able to render a marker file at a rendezvous location on the storage site. This is checked by the RMS, which authenticates over a hostname protocol. Thus the connection between the RMS and Guy is indirectly authenticated, as indicated by the italicized subject label *(Other, Guy)*.

As a second example, consider the simple collaboration shown in Figure 5.18. In this case, user B inserts a data record with a config policy that allows storage and access at domain C. Collaboration and shared data administration are possible even though each user is unable to authenticate at a remote site. Moreover, a job submitted by user B running in domain C may access data, perhaps using a simple DNS based authentication. This reinforces the notion that data replication may be viewed as an asynchronous job pre-staging process.

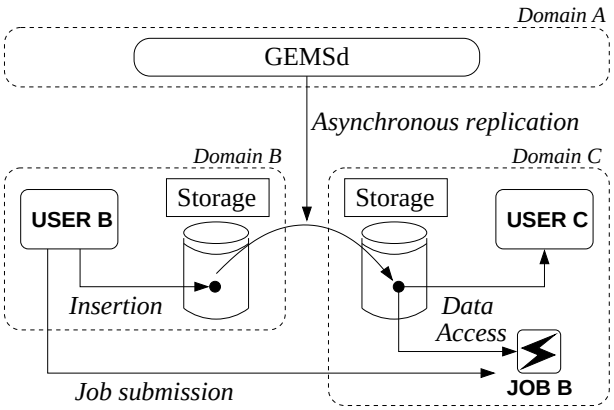


Figure 5.18. Simple three-domain collaboration.

The ability of users to create secure, derived data landscapes enables the elementary creation of a data workspace in which a foreign user may be permitted to operate by a previous user. Since the controller needs no *a priori* knowledge of the user names, an existing user can simply add foreign users to the ACL on a set of permitted records. The foreign user may then use their permission to access the system by performing the rendition protocol through a config controlled by a previously trusted user. Thus a trust chain is established by ordinary users to both

1. securely, incrementally, and locally expand the user base of the system, and

2. construct a system-manageable trust and accounting structure.

A diagram of example operation is shown in Figure 5.19. The rendition-based trust chain infrastructure solves a common problem in trust delegation: trust delegates must not receive too much information from the delegator. Passwords and other credentials simply cannot be passed to a third party. Likewise, the authentication system cannot be globally affected by ordinary users. Yet users

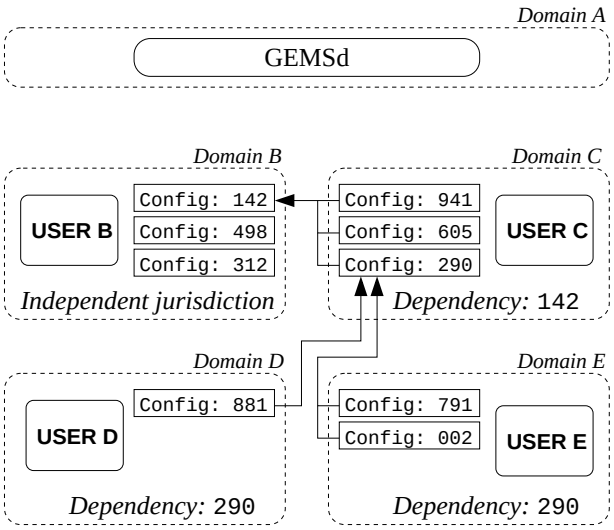


Figure 5.19. Trust chain construction.

are able to delegate access to records via an ACL, and the config data structure provides a physical authentication test. Practically speaking, outside users then gain insertion privileges to the data landscape in a dependent way.

5.6.7 Summary

> **ren · di · tion** *n.* 1. The act of submitting for approval [10];
> 2. An explanation of something that is not immediately obvious [157].

The protocol presented in this section results from several observations about the properties of widely distributed replica management systems: the wide variety of users, access methods, and administrative domains; the distinction between centralized metadata and user data files; and an implicit trust of storage providers, that storage providers will keep the data sets secure. The purpose of replication

here is not for security purposes but for reliability, availability, and performance purposes. This model is consistent with the security model of opportunistic computation systems because in such systems users must trust some number of sites to create correct data sets.

The new method presented herein attempts to authenticate users in a limited way, avoiding difficulties caused by squeezing users into a single authentication protocol, as well as the administrative challenge of managing all users and resources as a whole. Thus the user identity of a client connection to the system is never "immediately obvious", and an explanation is provided in the form of a file operation submitted for approval.

5.7 Summary

This chapter has laid out the GEMS system as an example implementation of a grid infrastructure. GEMS offers high performance scalable data services to running grid jobs, offers an organized programming model, enables the construction and customization of workspaces, and manages access control through a novel technique.

The main topic of this work has been the construction of a model for grid controllers. This chapter has emphasized that the solution techniques used along the way were in accordance with that model. In the next chapter, another grid problem is presented and solved using the grid controller paradigm.

CHAPTER 6

SCHEDULING POLICY

Researchers conducting computer simulations can often provide estimates of computation time for a given type of simulation, which may be used by the compute cluster to aid in resource allocation and scheduling. However, the low quality of these estimates can cause deadline misses and unpredictable behavior. This problem is exacerbated on complex compute resources, clusters, and grids. Past-deadline jobs may be killed to provide resources for others, but the effect on the throughput of whole batches not fully understood. In this chapter, we examine models for simple job schedulers and examine the quality of the deadline guarantee given. Simulation results based on actual runtimes are provided and discussed.

The scheduling policy techniques described in this chapter were originally reported in 2007 [162].

6.1 Overview

The emergence of commodity compute clusters and grids has provided researchers with an important tool for simulation. Batch systems such as PBS [68] and LSF [167] provide users with a cluster of compute hosts to which jobs may be submitted, while grid engines such as Condor-G [55] and Globus GRAM [37] create an access point for widely distributed compute systems. Although such

resources often suffer from poor internal communication speed compared to a multiprocessor shared memory machine, they have been used with considerable success by researchers who need to submit a batch of independent jobs for processing. The vast majority of this work has focused on how to successfully share and complete computing tasks such as computation and storage to achieve a large scientific objective.

However, an often unaddressed aspect of grid computing is the notion of deadline-driven scheduling [129]. Unlike traditional deadline scheduling from the realm of real-time computing [81], the problem of deadline scheduling in the grid context is significantly more difficult. The critical difference that emerges in grid computing is the dynamic nature of the grid resources themselves.

Previous work in the area of grid deadline scheduling considered only the impact of low quality estimations on throughput with regards to deadline scheduling [129]. The effects of *policing* when coupled with variable quality deadline estimations in the context of grid computing have not been fully investigated. In addition, previous work in grid scheduling has typically focused on each task being completely independent. For many types of scientific simulation such as parameter sweeps and parameter explorations, this is not the case. In these cases, tasks can often be grouped into a batch of related tasks that while computationally independent, the utility of the final scientific result is dependent upon the completion of all tasks.

Thus, the motivation of our investigation here is to answer the following question: *Given an environment of related tasks with low quality information, how strictly should policing be enforced and what effects will result on throughput and deadline guarantees?* In this topic, we make several key contributions. First,

we offer insight into the effects of policing when the system contains low quality run-time estimations. We offer a broad set of simulation studies incorporating both real world data as well as synthetic data. Second, we formalize a model whereby batches of jobs can be specified with differing deadlines, and applied this to real world scenarios. Finally, we offer insight into how to police. In short, when multiple users offer bad estimates, who should pay the price?

Our solution is based on the control model proposed in the introduction to this dissertation. In the deadline-driven case here, relatively small system perturbations can lead to relatively large batch completion unpredictability. We apply an overdrive metascheduler with new policy principles to control an existing, abstract batch system. The controller ameliorates the scheduling challenges by taking into account both the magnitude of the run-time perturbations as well as the value of user jobs. Thus, probabilistically proportionate responses may be applied to reduce system unpredictability. Additionally, we provide an incentive for users to strive for good run-time estimates, creating a game theoretic environment in which users are anticipated to be motivated to enhance their estimates, promoting better system utility.

The remainder of this chapter is organized as follows. Next, Section 6.2 motivates our work by providing cases that exemplify the trade offs directly considered in this work. Then, Section 6.3 formalizes the system model used in Section 6.4 for our simulation studies based on both synthetic and real world data.

6.2 Case Studies

6.2.1 Applications

For our first example case, a CPU design was optimized through simulation with SimpleScalar [27], a tool for simulating the performance of real programs on a range of modern processors utilizing execution-driven simulation. Tasks varied in run-time from as shown in Table 6.1. Most importantly, the utility of the results themselves was dependent upon the completeness in that no points were missing from the results. In the second simulation case, network simulations were conducted using the NS-2 [95] simulator. NS-2 is a discrete event simulator that provides packet-level granularity for simulating networks. During execution of the above simulations, the run-time statistics were sampled.

6.2.2 Grid Middleware

Over the past few years, the GIPSE (Grid Interface for Parameter Sweeps and Exploration) toolkit has been developed [163]. Rather than exposing the task-centric nature of the grid, the tool allows the user to solve problems using a research-oriented interface. GIPSE manages the creation and monitoring of jobs on the compute grid, as well as maintaining a database of all the relevant data about previously completed jobs, including input parameters and output results, job metadata such as the executable and resources used, job computation time, and other useful information.

It is this requirement for timeliness the large body of work in real-time multi-processor scheduling can be brought to bear. In essence, the problem can be reduced to an admission control and scheduling problem. For the researcher, the question is simple: given a group of tasks, can they be finished on time?

However, it is the nature of grid computing that makes this problem significantly more difficult than the traditional multi-processor scheduling problems faced in real-time.

6.3 A Model for Deadline-Driven Grids

As discussed above, there is a great need for *ad hoc* compute clusters and grids that can provide reasonable schedule guarantees. In this section, we lay out a model to meet such demands, as shown in Figure 6.1.

The object of the system is to complete several batches of jobs, where each batch has a deadline given by the user. For each batch $B_i \in B$, $B_i = \{J_{ijk}\}$, where each job J has three indices, the batch number, the task type, and a uniquifier. Each batch has an independent deadline, $B_i.deadline$. The system can identify a job J_{ijk} as an instance of task $T_j \in T$, belonging to batch B_i, and different from all other jobs in the system.

To obtain a response from the scheduler as to whether B_i is feasible given the current system state, each job is given a computation time estimate, $J_{ijk}.est$. The user also provides each job with a corresponding input set I_{ijk}, which is a set of input parameters valid for task T_j, which is the parameter space for T_j, denoted $T_j.I$. Each parameter is indexed, so $I_{ijk}[0], I_{ijk}[1], ...I_{ijk}[m-1] \in I_{ijk}$. The actual computation time is obtained empirically by the grid; in this model, each task has a *device* that may be applied to an input set to determine computation time. Hence, once submitted to a compute resource, the computation time $J_{ijk}.c$ is obtained by evaluating $J_{ijk}.c = T_j.device(I_{ijk})$.

A compute grid G consists of N homogeneous processing hosts, that are each capable of executing any job in a non-preemptive, single-processing manner. The

jobs currently running on the grid at time t are said to be in the set R_t. If a job J is added to R_t, then $J.start = t$. If more jobs are submitted to the grid than available hosts, they are queued in the FIFO wait queue W. When a job J_i completes execution, $R_{t+1} := R_t - \{J_i\}$, an event is triggered back to the user, and if W is not empty, it is popped to obtain job J_j, which is added to R_{t+1}.

The scheduler S accepts a batch B_i from the user at any time, and can determine whether the batch has a feasible deadline in several heuristic ways. In the FIFO model, which we use[1], S builds up a calendar into the future, stacking onto R all jobs in $W \cup \{B_i\}$. If each job meets the deadline, success is returned for the batch, but since the guarantee is based upon the accuracy of the task estimates provided, there is the possibility of deadline misses.

The value of the batch in a simulation research setting represents, for example, data points on a plot. This work assumes the researcher cannot accept partially complete data sets. Since the user penalty for having a rejected batch is less than that from a late or incomplete batch, it is better to force the user to negotiate: to adjust their submission to meet a deadline than to provide a less accurate guarantee, providing poor results.

This is based on the assumption that the user can supply per task estimates where the actual runtime is centered around the estimated computation time, but offset by a random value that is within a given percentage of the actual time. We call this value the Quality of Estimate (QoE) to avoid conflict with [43]. For example, if the simulated user can always provide the scheduler with a time estimate such that the actual runtime is within 40% of the estimate, and not

[1]For simplicity and compatibility with our experimental results on a Condor system.

biased higher or lower, we say the QoE is 40%. So we assert:

$$\left|\frac{J_{ijk}.est - J_{ijk}.c}{J_{ijk}.est}\right| \leq QoE. \tag{6.1}$$

For any set of batches B, after submission to the scheduler, we have an acceptance ratio that indicates the size of the subset of B that was accepted. Additionally, we use the definition of guarantee ratio as given in [91], on batches, so the ratio represents the number of batches in which all jobs met the deadline for that batch, divided by the number of batches accepted by the scheduler.

To help free up space for a new batch that arrives at a given scheduling event, the scheduler may be permitted to kill jobs that have been running in R for longer than they were allocated based on their estimate. So we say that J is eligible to be killed at time t if:

$$t - J.start > J.est. \tag{6.2}$$

To demonstrate an intermediate approach as motivated in Section 6.4, we can instruct the scheduler to kill jobs that have exceeded a certain threshold K, meaning we kill jobs when:

$$t - J.start > J.est \times (1 + K). \tag{6.3}$$

For example, if $K = 0.5$, jobs are allowed to exceed their estimate by 50% before becoming a possible victim.

Choosing K is difficult, because it is not easily possible to choose a value that provides the users with flexibility for errors and still promotes fairness and deadline guarantees. To help select jobs for termination, and ameliorate these issues, we may assign a probability p that a job will be killed at a scheduling

event at time t, as:

$$p = 1 - \frac{J.est}{t - J.start}. \tag{6.4}$$

For example, a job that had exceeded its estimate by a factor of 2 at a given scheduling event would have a 50% chance of being killed.

6.4 Simulation

To investigate the properties of the system presented in Section 6.3, a simulator was written that corresponds to that model. The results in this section demonstrate the performance of scheduler policy under idealized and experimental workloads.

The simulator, designed to allow the study of computation time *estimates* over *time* is named East. The program allows the researcher to set up a virtual compute grid and scheduler combination. Virtual tasks may be defined, assigned input parameters, grouped into batches, and sent to a virtual compute cluster. A table of computation time devices is given in Table 6.1. The experimental run times were obtained from the Condor system as discussed in Section 6.2. `SS` refers to a table of runtime information from the SimpleScalar cases and may be used to lookup the runtime for a job given an input set, I. Similarly, `NS` refers to a table of runtimes for NS-2.

Each of the following tests is based on a varying QoE, as defined above in (6.1). For each QoE value, 10 tests were run, and the results were averaged. The standard deviation of the tests is shown by the bars around each data point. For each job, the estimate given to the scheduler was randomly selected from the set of estimates that satisfy (6.1). Batches of fixed size arrived at the scheduler between random, uniformly distributed intervals around a given mean value. The input

TABLE 6.1

COMPUTATION TIME DEVICES IN THE EAST SIMULATOR

Type	**Name**	**Device**	**Median**	**Mean**	**Min**	**Max**
Ideal	SumDevice	$\sum I[i]$	22	22	0	45
Ideal	PolyDevice	$\sum k_i I[i]$	123	198	0	804
Experimental	SS-Device	$\texttt{SS}[I]$	218	237	137	615
Experimental	NS-Device	$\texttt{NS}[I]$	297	426	87	3442

Note: `SS` and `NS` refer to database tables.

set for each job was randomly, uniformly selected from the space of valid input for the appropriate device. The simulated compute resources comprise a simple N node cluster. The batch size was scaled up as the number of hosts increased to simulate more complex systems.

6.4.1 Low-Quality Estimates

In this test, the acceptance and guarantee ratios for the scheduler were measured against the QoE. As shown in Figure 6.2, increasing the error in the estimate has a significant effect on scheduler acceptance, especially in complex systems. This was observed in many cases to be due to a single large over-estimation of the runtime of a long job, which forces the schedule past the deadline, and results in rejection. Additionally, the guarantee ratio for whole batches is perfect when the estimates are exact, but even when they vary widely, the effect is that the guarantee ratio rarely drops. This is because the deadlines are somewhat permissive, but tight enough that not all batches are accepted. However, in the complex case with 128 hosts, a heavy job rate, and very bad estimates, almost no jobs

are accepted, because nearly all batches will have at least one extremely long job with an overestimated runtime, forcing batch rejection. In these extreme cases, the Batch Guarantee Ratio is shown as 0 because no batches were considered.

Overall, while the guarantee ratio is nearly perfect, the acceptance ratio is not. This indicates that users could intentionally provide underestimates to increase their acceptance ratio, hogging the system.

In our next test, Figure 6.3, we attempt to police the system fairly and increase the scheduler acceptance rate by killing jobs that have exceeded their estimate. However, when the scheduler is instructed to kill jobs that have exceeded their estimate in order to accept future jobs, many fewer batches are able to complete. This is aggravated by the fact that we are measuring the Batch Guarantee Ratio, so killing one job that is slightly over time results in a loss of the whole batch. In addition, there is little to no gain in scheduler acceptance. This is because many of the jobs that were killed were close to completion at the time of kill, which means that very little schedule time was freed.

Similar results are obtained when using runtimes from real NS-2 runs as shown in Figures 6.4 and 6.5, in which we repeat the above experiments using experimentally observed NS-2 runtimes.

6.4.2 Grace Periods

The fact that so many jobs are killed near their completion time motivates the implementation of grace periods. The simple technique of granting jobs a threshold, or grace period, before killing them was also simulated for the PolyDevice tasks. For a range of values of K, the same simulation was performed, where jobs were killed according to the method described above. Although the results in

Figure 6.6 show that larger grace periods result in more throughput as measured by the batch guarantee ratio, this is an unfair, easily manipulated system policy.

6.4.3 Probabilistic Enforcement

Applying the probabilistic policy, as describe in Equation (6.4) and shown in Figure 6.7, acceptance ratios match up with ratios as given in the previously shown hard enforcement and non-enforcement policies. The batch guarantee ratio is much improved. This is due to the policy, which rarely kills jobs that are near their estimate. However, throughput as measured by the batch guarantee ratio still does not approach that of the unpoliced system.

6.4.4 Protecting Users from Bad Estimates

The intent of a policed system is to provide better results for users overall, especially those that use the system properly. Users that intentionally abuse the system, for example, by providing low estimates to increase the probability of batch acceptance, should receive poor results, if this is necessary to continue to provide good results to other users.

In this experiment, we compare the results obtained by two groups of users: Group A, which submits NS-Device tasks with estimates centered on the correct value, and Group B, which always submits an underestimate. Estimates for Group B were generated by randomly selecting an estimate inside the given QoE range, until an underestimate was obtained. Acceptance and guarantee ratios are shown in Figure 6.8. These results show that Group A users obtain better results: more of their accepted batches complete on time, because very few of their jobs are killed, compared to the Group B users.

6.5 Summary

While real-time computing and grid computing both emphasize scheduling, their ultimate goals are often different and result in trade offs. Strictly enforcing computation time estimates on the grid can greatly reduce throughput in heavily loaded, complex systems. However, failure to enforce a schedule is unfair to the other users of a shared compute system, and necessitates policy to prevent users from cheating the system by providing misleading estimates.

Tests performed with a new grid scheduling simulator showed that killing jobs that exceed their estimates can greatly reduce throughput, especially in complex environments. A more balanced approach is required. We offered a probabilistic policy that reduces the throughput penalty by probabilistically forgiving users extra time for over-running jobs. The new policy is also difficult to manipulate, and offers the best results to users that provide the best estimates.

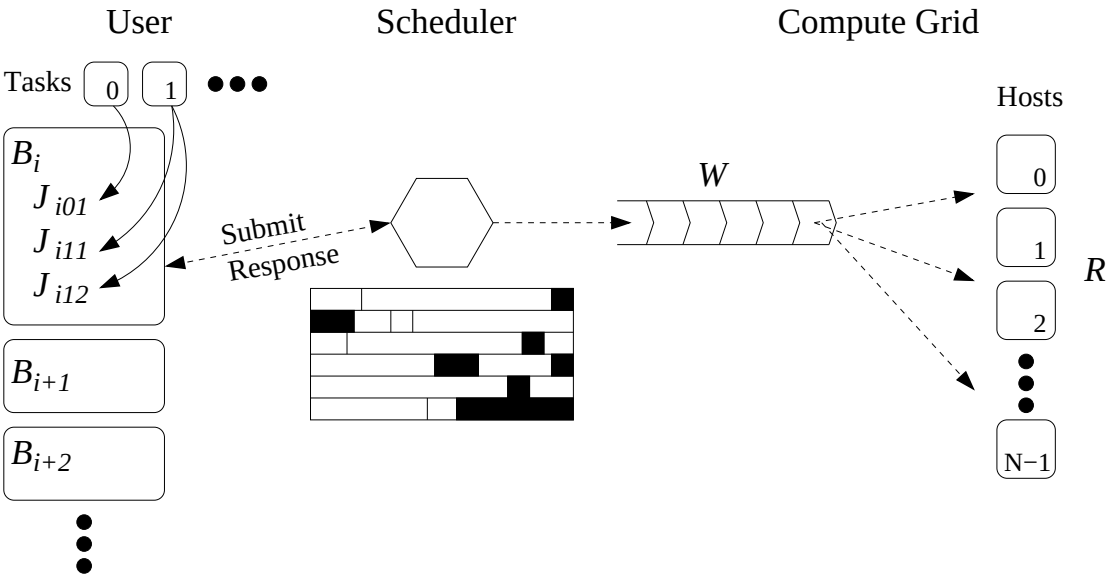


Figure 6.1. Deadline-driven grid computing model.

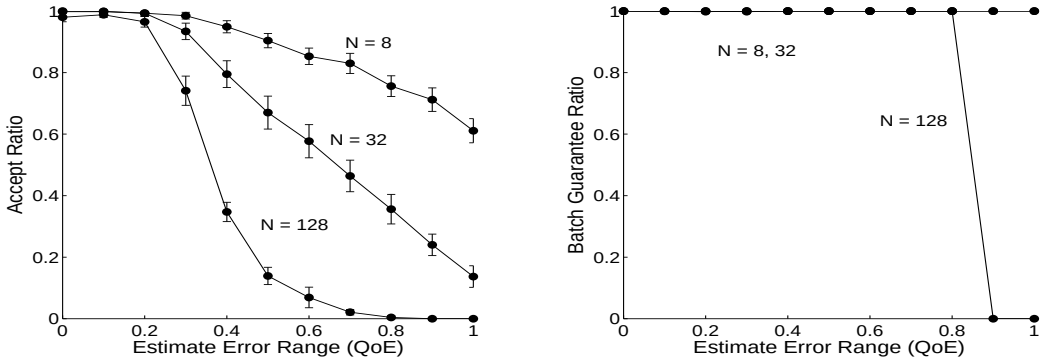


Figure 6.2: Acceptance and guarantee ratios for batches of jobs without enforcement. Batches of PolyDevice jobs.

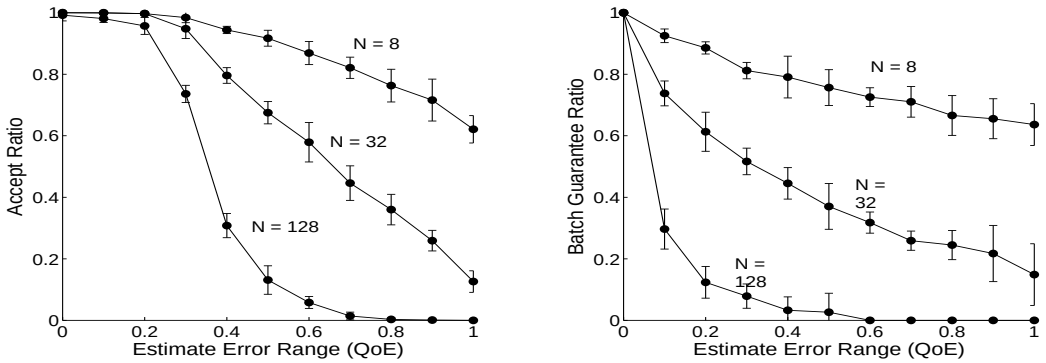


Figure 6.3: Acceptance and guarantee ratios for batches of jobs with hard enforcement. Batches of PolyDevice jobs.

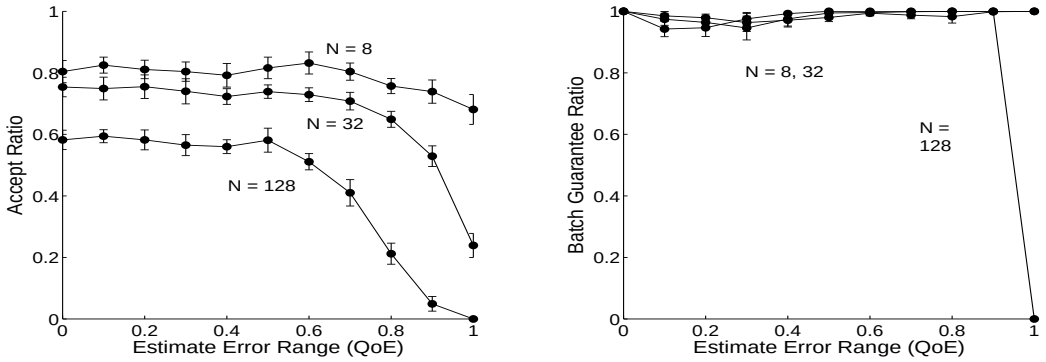


Figure 6.4: Acceptance and guarantee ratios for batches of jobs without enforcement. Batches of NS-Device jobs.

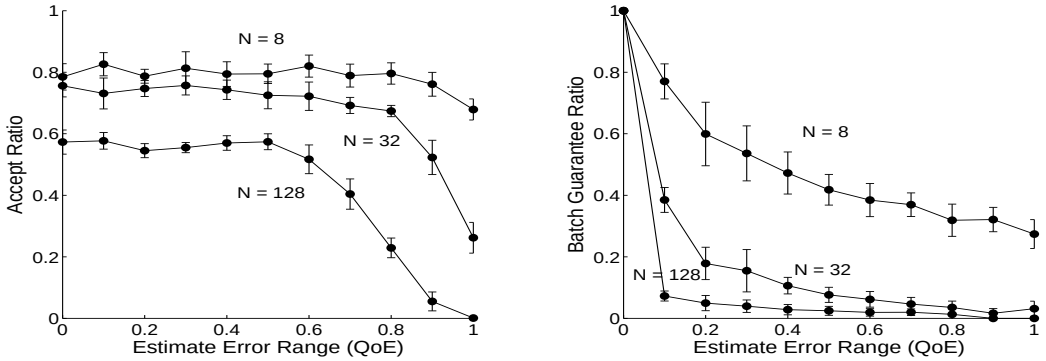


Figure 6.5: Acceptance and guarantee ratios for batches of jobs with hard enforcement. Batches of NS-Device jobs.

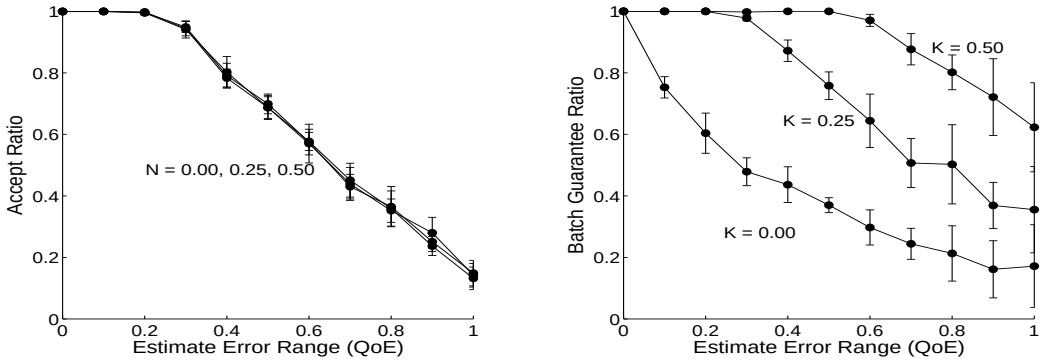


Figure 6.6: Acceptance and guarantee ratios for batches of jobs with enforcement level K. Batches of PolyDevice jobs.

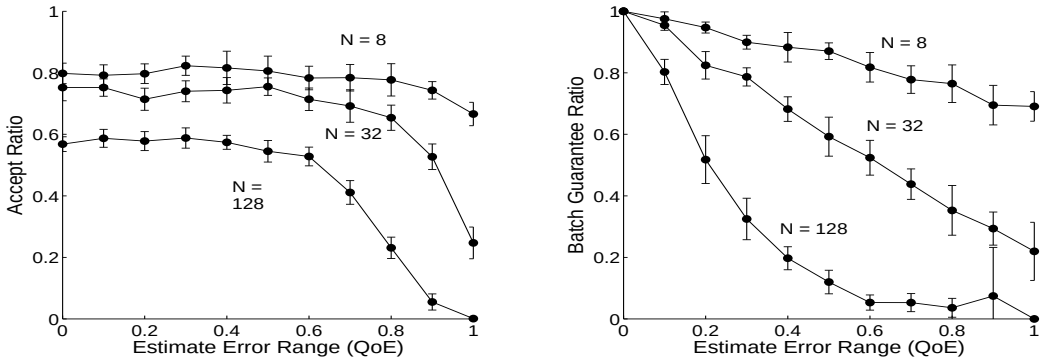


Figure 6.7: Acceptance and guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs.

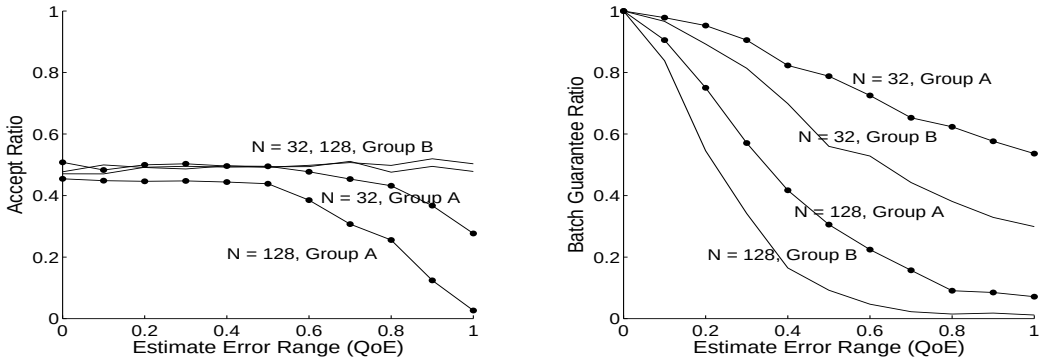


Figure 6.8: Acceptance and guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs.
Data with dots indicates the Group A users, who provided estimates centered on the correct running time, undotted lines indicate Group B users, who provided consistent underestimates. This enforcement method demonstrates an intermediate approach between hard enforcement and no enforcement, with intermediate guarantee ratios as a result. As shown, if users can keep their running times within 50% of the estimate, they achieve similar acceptance ratios to users that deliberately underestimate, while obtaining much better guarantee ratios on their batches. (Standard deviation information removed for clarity.)

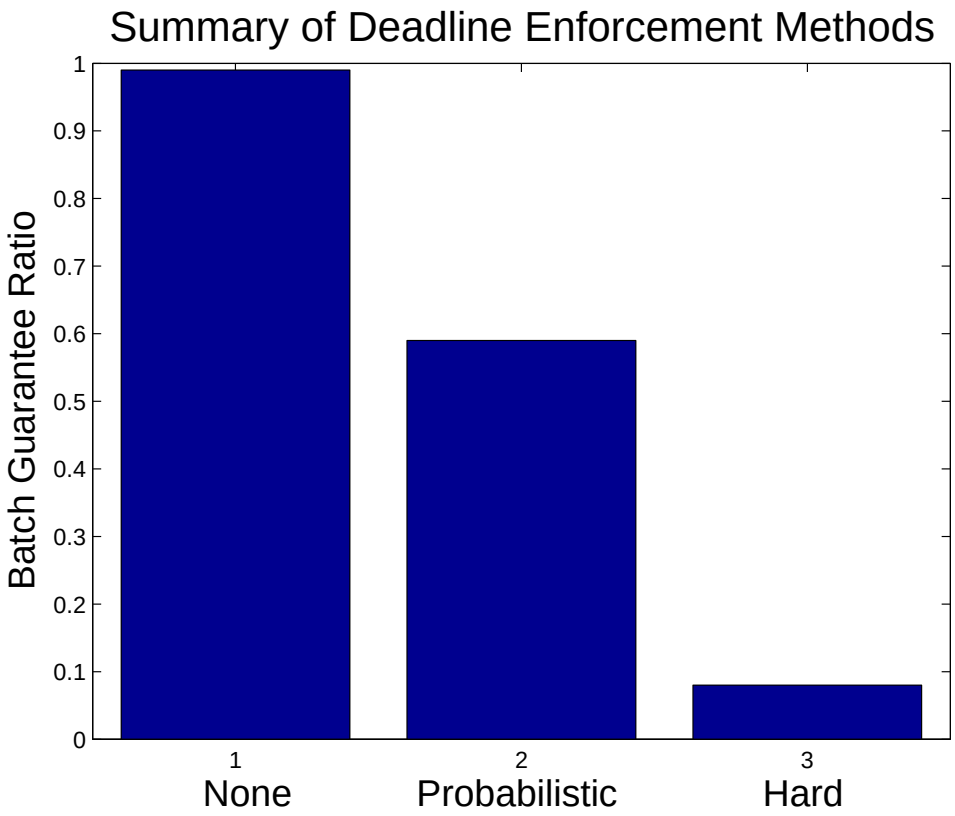


Figure 6.9: Guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs, N = 32, QoE = 50%.
Data compiled from previous diagrams. All users Group A: unbiased estimates. In summary, no enforcement allows for most jobs to complete on time, because of necessary over-estimation for worst case input parameter sets. Since this policy is easily abused, this policy is compared against the probabilistic enforcement method and the hard enforcement method. This illustrates the trade-off between deadline enforcement and high throughput.

CHAPTER 7

CONCLUSION

Scientific computing systems are expected to increase in size and complexity, creating an ever more pressing need for control abstractions and management solutions. These world-wide computing systems will continue to pose challenges until appropriate controls, methods, software are developed, implemented, and communicated to administrators and end users.

Programming the grid is ideally as simple as possible, however, users currently must be aware of many technical details of the resources utilized. Additionally, users must often orchestrate complex operations before and after jobs are executed. Interacting with a higher level controller may be a viable technique to change the semantics used by the programmer from an imperative model to a declarative framework. The shift emphasized here is one from a systems programming model to a policy programming model. If complex user tasks can be represented simply as policy rules, autonomic agents can perform the underlying systems operations. While this basic concept has been previously pioneered *via* matchmaking, brokering, or other techniques, we have presented a general solution based on a feedback loop model.

Additionally, managing the grid should be elementary. In fact, managing continental scale infrastructures is not easy and automatic tools must be developed to enable the fluid deployment of grid resources. Our work has attempted to

push administrative tasks away from site administration, treating resources as immutable objects. We have provided a model whereby swaths of these sites can be managed by user level controllers, which ultimately report to application users. Thus underlying sites can remain unaware of grids into which they are integrated, simplifying resource requirements.

7.1 Summary

We have provided the motivation, description and two examples of overdrive controllers for scientific computing on the grid. Each example remained based in its original domain - data storage or scheduling - but was shown to benefit in control and usability through the addition of an overdrive system.

The GEMS system started as a design requirement for a repository built upon existing volunteer resources. Chapter 3 described how GEMS meets the basic definition of a simulation repository, enabling simplified management of common data tasks in scientific applications, including metadata and basic integration with computational tools. However, higher level control of the system was required to keep user files alive on the unreliable underlying storage fabric, requiring the construction of a control loop layer as described in Chapter 4. GEMS was subsequently expanded into a comprehensive grid system in Chapter 5, enabling complex job/data coordination, grid construction and usability features, and a new authentication technique.

Similarly, East started as an deadline-driven computing application based on the availability of job metadata in grid systems. Heavy-handed estimate enforcement could be employed to attempt to improve the quality of the system guarantees, but was shown to be oppressive in unpredictable computing environments. A

new metascheduling layer was added, as described in Chapter 6, to make decisions about which jobs to prioritize based on the quality of user-provided information. This technique thus attempts to control system response to user requests as well as control user behavior in the game theoretic sense by a stabilizing preference for honest users.

In summary, the overdrive controller model provides a framework for understanding and reasoning about the coarse grained operation of distributed systems. Additionally, it provides a numerical flavor to analyzing the aggregate effects of system behavior. It can be used to describe points of interest in replica management systems, metaschedulers, and potentially other systems as well.

7.2 Next Generation Grids

How do we get more from conglomerations of existing systems without modifying their internals?

Achieving future goals in grid computing will require new software frameworks in addition to resources and investments. To propagate the full benefits of grid-enabled technologies to more users and stakeholders, several problems must be addressed.

1. **Usability:**

   Programming on the grid or any distributed system is different from working with the local machine, which is acceptable if reasonable usability is in place. Systems designers must present a tangible programming model that encapsulates scalable aggregate functionality without hiding important performance details.

2. **Manageability:**

   The automatic management of distributed systems can be thought of as usability for administrators or systems managers. Management and optimization can be grouped under the category of parameter tuning, an automated iterative process that selects the magic numbers that make a system work well - or work at all.

   Appropriate administrative tools lower the risk involved in grid installation or integration. New systems can act as a front-end to existing systems (overdrive), an alternative parallel route to perform system operations (short-circuit), or as a technique that necessarily replaces the existing system (disruptive technology).

3. **Visibility:**

   The physical remoteness of large distributed systems reduces the ability for users to obtain information about the system behavior. Beyond debugging broken software, system visibility enables users to realize the emergent properties of complex systems.

The overdrive controller model provides potential solutions for these abstract design requirements that will shape future work with the model.

Usability is a difficult target, at once requiring the conservation of existing frameworks while also requesting new, simpler, labor-saving innovations. The overdrive controller leaves existing systems intact, allowing a very conservative approach. However, the new framework offered is considerably different, migrating from an command-driven `do-this` model to a hands-off `this-must-be-done` model. For example, in replica management, user operations were augmented

from file movement operations to high-level cluster definition and replica request operations, resulting in complex management operations carried out by the controller.

Manageability is a similarly difficult topic. Controlling existing software requires the installation of more software. However, in the limited context of parameter tuning, controllers may offer a great deal of benefit. The model is designed to correct itself, pointing back to a sustainable steady state. For example, in the context of user quotas, GEMS automatically selects the maximum file replica count - and implements appropriate system changes - when available space becomes restricted. This number is iteratively updated as user requests increase and the available disk space ebbs and flows.

Since input and output are well-defined features of the overdrive controller model, an ideal controller would also improve system visibility. The internal system model could also be published to authorized clients, demonstrating the state of the system in as much as the controller is aware of it. However, controllers increase the asynchronicity of the system, making query results more difficult to understand, as the perceived system state and controller activity are delayed.

7.3 Future Work

Scientific computing systems have continued to function as social projects, combining teams of scientists, engineers, and technicians. Proposed petaflop capacity supercomputers, networks of university desktop machines, and production grids that span continents are modern infrastructures that rely on users and administrators collaborating effectively. While modern hardware provides more computing power than ever and interoperability standards have made heterogeneous

computing a reality, users are still unable to shape computing environments to their short term needs. Systems designers must enhance scientific control of computing environments, allowing for the dynamic creation of composite or derivative computing frameworks customized and optimized for target applications- a form of scientific ergonomics.

The GEMS and East project domains have a great amount of potential for future work in these areas. Both emphasize the ability to customize the behavior of existing resources to satisfy more demanding requirements. Additionally, the controller model itself could be applied to other problems in large-scale distributed computing.

7.3.1 Opportunistic Storage

Current work with GEMS has enabled researchers to quickly integrate storage infrastructures and carve out scientific workspaces in desktop grid environments. Future work in distributed storage will study the manner in which small groups of researchers interact through data sharing across large infrastructures, and attempt to address their needs. While these systems have been approached in the past as global optimal replica placement problems, mathematical models will be applied to study performance for the real world usage patterns of collaborating users. Future work will additionally provide comprehensive access delegation techniques inspired by indirect authentication methods and develop applications to large-scale scientific computing, Internet computing, and other areas.

A first topic to consider in shared storage over wide-area networks is the nexus of locality, survivability, flexibility, and cost. If an opportunistic storage fabric is used, much replica management must be performed to combat churn, thus increas-

ing the bandwidth cost for the benefit of greater flexibility. This greater flexibility could increase locality, by allowing the replica manager to migrate replicas to a user-created storage server, as discussed in Chapter 5. Real-world wide-area applications and user groups could be studied as they patch together user-level solutions for these problems within greater existing grid.

A second topic is data interoperability. GEMS currently provides three API-like interfaces that must be addressed by users:

1. The parameterized metadatabase, which provides a somewhat low-level, application-independent key/value tagging system;

2. A web services-like XML interface to the central GEMSd component, accessible over the network or through the provided clients;

3. Additionally, replica location information obtained from the clients may be used to obtain direct access to files on the underlying Chirp servers over the Chirp wire protocol.

Each of these features was designed to be as user-friendly as possible, providing higher-level functionality for the common scientific tasks of data set submission and retrieval. This interface operates quite differently from, for example, a POSIX interface, but the POSIX interface excels at portability as exemplified by implementations such as Parrot or FUSE [57]. Since GEMS does not implement a commonly used filesystem or database interface, users must invest time and effort to integrate GEMS with their workloads.

The data set tagging framework could be augmented with an optional interface and tools that mimics existing programmatic interfaces to data sources such as the Protein Data Bank [20] or GenBank [19]. This would ease the introduction of

GEMS data sources into existing operating environments.

Additionally, the central metadatabase should allow transactions as a standardized web service. This would allow document and data archives to use GEMS as a back end for emerging, standardized data services in a variety of application areas. Underlying data stored on file servers in raw formats could be converted and merged into the XML data stream by forwarding connections through intermediate translation services, located centrally on the GEMS server or dispersed throughout the storage network. Data movement and access delegation in this two-hop transfer/translate/transfer setup would be an interesting area for study as well.

7.3.2 Timeliness in Distributed Computing

The East simulator was developed to investigate the effects of a deadline computing framework to promote predictable behavior when resources are congested. The simulator is currently being extended to investigate job migration strategies for algorithms that suffer from barrier synchronization delays.

A multiprocessor barrier operation is a programmatic step which must be passed by all processes at the same time. This method may used to begin a synchronized all-to-all computation or, in a Monte Carlo setting, to allow intermediate processing to interleave rounds of parallel computation. Executing barriers on the grid is challenging for a variety of reasons, including

1. heterogeneity of computation resources;
2. resource unreliability; and
3. the potentially high cost of job migration.

For example, previous work [156] scheduling a barrier-dependent molecular dynamics computation in an opportunistic computing system used dedicated clusters of faster processors to advance jobs that were lagging behind or encountered resource failure.

Our future work will generalize the barrier scheduling problem for an arbitrary case in which multiple competing users schedule these workloads, and using East, we will investigate several related problems. First, we will show that the greedy approach of always holding onto processors while waiting for a barrier to pass is deadlock-prone, complicating simple strategies and resulting in considerable job migration or staging costs. Second, assuming a small number of high performance processors are available for use, we will investigate architectural questions such as

1. How to integrate these with the rest of the opportunistic system;
2. How many to reserve for the catch-up case;
3. How to promote jobs to the catch-up cluster; and
4. What the throughput costs are of specialized strategies compared to a traditional raw throughput case.

Additionally, parameterized computation histories may be tapped to enhance user understanding of computational resources required by their parameter explorations. Studying these histories will require the application of computational geometry and applying them to new runs will need high dimensional interpolation techniques.

APPENDIX A

GEMS DEVELOPMENT

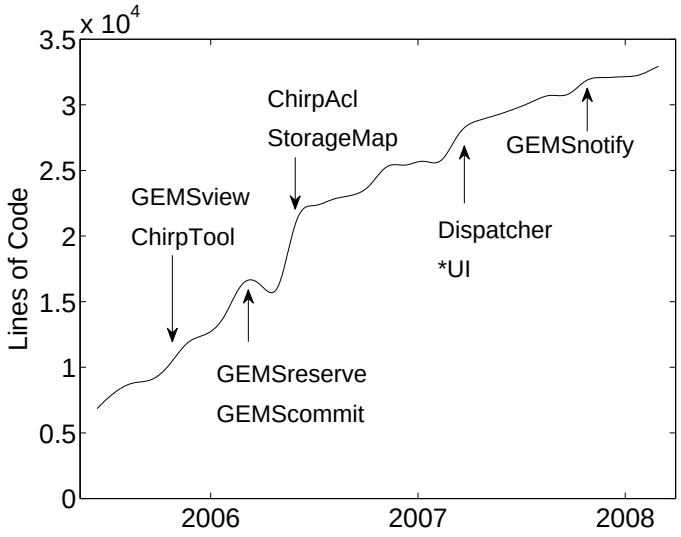


Figure A.1. Lines of code in GEMS over time.

The graphic above plots lines of code in the GEMS CVS repository as a splined function of time. Denoted milestones indicate classes of interest that were introduced to satisfy new concepts and requirements.

- **GEMSview, ChirpTool**: GEMSview (Section 3.3.1) provided a graphical data browser for GEMS, and the ChirpTool implemented a Chirp command-line tool in the Java platform, together increasing user accessibility to the system.

- **GEMSreserve, GEMScommit**: Implemented a two-phase data insertion procedure for GEMS data sets. The resulting code overhaul resulted in some code size reduction as GEMSput was reimplemented atop these operations.

- **ChirpAcl, StorageMap**: Increased the ability of both GEMS services and clients to implement storage and access control policy in the resource network. Covered in Chapter 5.

- **Dispatcher, *UI**: A variety of graphical user interfaces were added to GEMS, accessible through GEMSview. Much of the formerly command-line client functionality is now accessible through point-and-click tools.

- **GEMSnotify**: Provides client notification upon matching data set insertion. Enables the parameterized workflow paradigm discussed in Section 5.3.

GEMS is an open-source project available at:

`http://sourceforge.net/projects/gems-nd`

APPENDIX B

VITA

Justin Michael Joseph Wozniak is a native of the state of Illinois in the USA. He graduated from St. Edwards High School, Elgin, IL, in 1996. His Bachelor of Science in Mathematics and Computer Science with minors in Latin and Chemistry was awarded by the University of Illinois at Urbana-Champaign in 2000. His Master of Mathematics in Computer Science was awarded by the University of Waterloo, ON, in 2003. Upon completion of the doctoral requirements at the University of Notre Dame, he will take a postdoctoral appointment at Argonne National Laboratory, residing in Chicago, IL, with his wife Venus and daughter Gracie Day.

BIBLIOGRAPHY

1. G. Abla, G. Wallace, D. Schissel, S. Flanagan, Q. Peng and J. Burruss, Shared display wall based collaboration environment in the control room of the DIII-D national fusion facility. In *Proc. Workshop on Advanced Collaborative Environments* (2005).

2. D. Abramson, J. Giddy and L. Kotler, High performance parametric modeling with Nimrod/G: Killer application for the global grid. In *Proc. International Parallel and Distributed Processing Symposium* (2000).

3. D. Abramson and J. Kommineni, A flexible I/O scheme for grid workflows. In *Proc. International Parallel and Distributed Processing Symposium* (2004).

4. J. K. Adelman-McCarthy, The sixth data release of the Sloan Digital Sky Survey. *ArXiv e-Prints*, 707 (2007).

5. A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer and R. Wattenhofer, Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proc. Symposium on Operating Systems Design and Implementation* (2002).

6. D. Agarwal, S. Sachs and W. Johnston, The reality of collaboratories. *Computer Physics Communications*, 110(1-3) (1998).

7. M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen and M. Parashar, AutoMate: Enabling autonomic applications on the grid. In *Proc. Autonomic Computing Workshop* (2003).

8. G. A. Alvarez, W. A. Burkhard and F. Cristian, Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In *Proc. International Symposium on Computer Architecture* (1997).

9. G. M. Amdahl, Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings* (1967).

10. The American Heritage Dictionary of the English Language. Houghton Mifflin Company, Fourth Edition.

11. D. P. Anderson, BOINC: A system for public-resource computing and storage. In *Proc. Workshop on Grid Computing* (2004).

12. D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer, SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11) (2002).

13. T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli and R. Y. Wang, Serverless network file systems. In *Proc. Symposium on Operating System Principles* (1995).

14. Apache Ant. Web site, `http://ant.apache.org`.

15. M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. J. Giuli and P. Bungale, A fresh look at the reliability of long term digital storage. In *EuroSys* (2006).

16. F. Baskett, K. M. Chandy, R. R. Muntz and F. G. Palacios, Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2) (1975).

17. J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau and M. Livny, Explicit control in a batch-aware distributed file system. In *Proc. USENIX Symposium on Networked Systems Design and Implementation* (2004).

18. J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. C. A. Dusseau, R. H. Arpaci-Dusseau and M. Livny, Flexibility, manageability, and performance in a grid storage appliance. In *Proc. High Performance Distributed Computing* (2002).

19. D. Benton, Recent changes in the GenBank on-line service. *Nucleic Acids Research*, 18 (1990).

20. H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov and P. E. Bourne, The protein data bank. *Nucleic Acids Research* (2000).

21. F. C. Bernstein, T. F. Koetzle, G. J. Williams, J. Edgar F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi and M. Tasumi, The protein data bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, 112(3) (1977).

22. J. Bester, I. Foster, C. Kesselman, J. Tedesco and S. Tuecke, GASS: A data movement and access service for wide area computing systems. In *Proc. I/O in Parallel and Distributed Systems* (1999).

23. G. Bolch, S. Greiner, H. de Meer and K. S. Trivedi, *Queueing Networks and Markov Chains*. Wiley Interscience, second edition (2006).

24. P. G. Bolhuis, D. Chandler, C. Dellago and P. L. Geissler, Transition path sampling: Throwing ropes over rough mountain passes, in the dark. *Annual Review of Biophysics and Biophysical Chemistry*, 53 (2002).

25. P. Brenner, J. M. Wozniak, D. Thain, A. Striegel, J. W. Peng and J. A. Izaguirre, Biomolecular path sampling enabled by processing in network storage. In *Proc. Workshop on High Performance Computational Biology* (2007).

26. D. Britton, A. Cass, P. Clarke, J. Coles, A. Doyle, N. Geddes, J. Gordon, R. Jones, D. Kelsey, S. Lloyd, R. Middleton, S. Pearce and D. Tovey, GridPP: Meeting the particle physics computing challenge. In *UK e-Science All Hands Conference* (2005).

27. D. Burger, T. M. Austin and S. Bennett, Evaluating future microprocessors: the SimpleScalar tool set. Technical Report 1308, University of Wisconsin, Madison, WI (1996).

28. J. Burruss, T. Fredian and M. Thompsonc, Security on the US Fusion Grid. *Fusion Engineering and Design*, 81(15-17) (2006).

29. H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, Heuristics for scheduling parameter sweep applications in grid environments. In *Proc. Heterogeneous Computing Workshop* (2000).

30. H. Casanova, G. Obertelli, F. Berman and R. Wolski, The AppLeS parameter sweep template: User-level middleware for the grid. In *Proc. Supercomputing* (2000).

31. T. D. Chandra and S. Toueg, Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2) (1996).

32. K. M. Chandy and D. Neuse, Linearizer: a heuristic algorithm for queueing network models of computing systems. *Communications of the ACM*, 25(2) (1982).

33. K. I. Chang, K. W. Bowyer and P. J. Flynn, An evaluation of multi-modal 2D+3D face biometrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4) (2005).

34. A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt and M. Ripeanu, Giggle: A framework for constructing scalable replica location services. In *Proc. Supercomputing* (2002).

35. A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman and R. Schwartzkopf, Performance and scalability of a replica location service. In *Proc. High Performance Distributed Computing* (2004).

36. E. F. Codd, A relational model of data for large shared data banks. *Communications of the ACM*, 13(6) (1970).

37. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith and S. Tuecke, A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459 (1998).

38. K. Czajkowski, I. Foster and C. Kesselman, Resource co-allocation in computational grids. In *Proc. High Performance Distributed Computing* (1999).

39. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris and I. Stoica, Wide-area cooperative storage with CFS. In *Proc. Symposium on Operating Systems Principles* (2001).

40. E. Deelman, T. Kosar, C. Kesselman and M. Livny, What makes workflows work in an opportunistic environment? *Concurrency and Computation: Practice and Experience*, 18 (2006).

41. E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gila, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13 (2005).

42. R. Dooley, K. Milfield, C. Guiang, S. Parmidighantum and G. Allen, From proposal to production: Lessons learned developing the computational chemistry grid cyberinfrastructure. *J. Grid Computing*, 4(2) (2006).

43. L. Dunning and S. Ramakrishnan, A heuristic cost estimation method for optimizing assignment of tasks to processors. In *Proc. Symposium on Applied Computing* (1999).

44. H. A. Duran-Limon, G. S. Blair and G. Coulson, Adaptive resource management in middleware: A survey. *Distributed Systems Online*, 5(7) (2004).

45. D. G. Y. et. al., The Sloan Digital Sky Survey: Technical summary. *The Astronomical Journal*, 120 (2000).

46. S. I. Feldman, Make - a program for maintaining computer programs. *Software - Practice and Experience* (1979).

47. I. Foster, What is the Grid? A three point checklist. *GRIDToday* (2002).

48. I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, first edition (1999).

49. I. Foster and C. Kesselman, editors, *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, second edition (2003).

50. I. Foster, C. Kesselman, J. Nick and S. Tuecke, Grid services for distributed system integration. *Computer*, 35 (2002).

51. I. Foster, C. Kesselman, J. Nick and S. Tuecke, The physiology of the grid: An open grid services architecture for distributed systems integration. Open Grid Service Infrastructure Working Group, Global Grid Forum (2002).

52. I. Foster, C. Kesselman, G. Tsudik and S. Tuecke, A security architecture for computational grids. In *Proc. Conference on Computers and Security* (1998).

53. I. Foster, C. Kesselman and S. Tuecke, The anatomy of the grid. *J. Supercomputer Applications*, 15 (2001).

54. I. Foster, J. Voeckler, M. Wilde and Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proc. Scientific and Statistical Database Management* (2002).

55. J. Frey, T. Tannenbaum, I. Foster, M. Livny and S. Tuecke, Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3) (2002).

56. Y. Fu, H. Wang, C. Lu and R. S. Chandra, Distributed utilization control for real-time clusters with load balancing. In *Proc. Real Time and Embedded Technology and Applications Symposium* (2006).

57. FUSE. Web site, `http://fuse.sourceforge.net`.

58. K. Gaither, Visualization's role in analyzing computational fluid dynamics data. *IEEE Computer Graphics & Applications*, 24(3) (2004).

59. D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu and N. Rey-Cenvaz, Programming the grid: Distributed software components, P2P and grid web services for scientific applications. *Cluster Computing*, 5(3) (2002).

60. D. Gannon, G. Fox, M. Pierce, B. Plale, G. von Laszewski, C. Severance, J. Hardin, J. Alameda, M. Thomas and J. Boisseau, Grid portals: A scientist's access point for grid services (draft 1). *Global Grid Forum* (September 2003), Work in Progress.

61. C. F. Gauss, *Disquisitiones Arithmeticae*. Fleischer (1801).

62. Y. Gil, P. A. Gonzlez-Calero and E. Deelman, On the black art of designing computational workflows. In *Proc. Workshop on Workflows in Support of Large-Scale Science* (2007).

63. S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura and R. Neyama, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Pearson Education, first edition (2001).

64. S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura and R. Neyama, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*. Pearson Education (2001).

65. A. Grimshaw and W. A. Wolf, Legion - a view from 50,000 feet. In *Proc. High Performance Distributed Computing* (1996).

66. T. P. G. D. Group, *PostgreSQL Reference Manual - Volume 1 SQL Language Reference*. Network Theory Ltd. (2007).

67. J. Hartman and J. Ousterhout, The Zebra striped network file system. In *Proc. Symposium on Operating System Principles* (1993).

68. R. L. Henderson and D. Tweten, Portable batch system: Requirement specification. Technical report, NAS Systems Division, NASA Ames Research Center (1998).

69. J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham and M. West, Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1) (1988).

70. W. F. Humphrey, A. Dalke and K. Schulten, VMD – Visual Molecular Dynamics. *J. Molecular Graphics*, 14 (1996).

71. L. B. Huston and P. Honeyman, Partially connected operation. *Computing Systems*, 8(4) (1995).

72. K. Keahey, K. Doering and I. Foster, From sandbox to playground: Dynamic virtual environments in the grid. In *Proc. Workshop on Grid Computing* (2004).

73. K. Keahey, M. Ripeanu and K. Doering, Dynamic creation and management of runtime environments in the grid. In *Workshop on Designing and Building Grid Services* (2003).

74. T. Kosar and M. Livny, Stork: Making data placement a first class citizen in the grid. In *Proc. International Conference on Distributed Computing Systems* (2004).

75. A. R. Leach, *Molecular Modelling: Principles and Applications*. Addison Wesley Longman (1996).

76. H. Li and M. Muskulus, Analysis and modeling of job arrivals in a production grid. *SIGMETRICS Performance Evaluation Review*, 34(4) (2007).

77. J. Li, J. Stribling, T. M. Gil, R. Morris and M. F. Kaashoek, Comparing the performancs of distributed hash tables under churn. In *Proc. Workshop on Peer-to-Peer Systems* (2004).

78. W. Li, R. Byrnes, J. Hayes, V. Reyes, A. Birnbaum, A. Shabab, C. Mosley, D. Pekurowsky, G. Quinn, I. Shindyalov, H. Casanova, L. Ang, F. Berman, M. Miller and P. Bourne, The encyclopedia of life project: Grid software and deployment. *J. New Generation Computing on Grid Systems for Life Sciences* (2004).

79. M. Litzkow, M. Livny and M. Mutka, Condor - A hunter of idle workstations. In *Proc. International Conference of Distributed Computing Systems* (1988).

80. D. Liu and M. Franklin, GridDB: A data-centric overlay for scientific grids. In *Proc. Very Large Data Bases* (2004).

81. J. Lopez, M. Garcia, J. Diaz and D. Garcia, Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. *Proc. Euromicro Conference on Real-Time Systems* (2000).

82. M. Maisel and G. Wells, Teraflops tackle terabytes on the teragrid. *Earth Observation Magazine*, 13(5) (2004).

83. P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, M. Bakerand and Y. Muliadi, Preserving peer replicas by rate-limited sampled voting. In *Proc. Symposium on Operating Systems Principles* (2003).

84. T. Matthey, T. Cickovski, S. Hampton, A. Ko, Q. Ma, M. Nyerges, T. Raeder, T. Slabach and J. A. Izaguirre, ProtoMol, an object-oriented framework for prototyping novel algorithms for molecular dynamics. *ACM Transactions on Mathematical Software*, 30(3) (September 2004).

85. M. Mealling and R. Denenberg, Uniform resource identifiers (URIs), URLs, and uniform resource names (URNs): Clarifications and recommendations. *IETF RFC 3305* (2002).

86. G. Mendel, Experiments on plant hybridization. *J. Royal Horticultural Society*, 26 (1901).

87. S. Microsystems, Sun Grid Engine. `http://gridengine.sunsource.net`.

88. R. Moore, Evolution of data grid concepts. In *Proc. Global Grid Forum Data Area Workshop* (2004).

89. R. W. Moore, C. Baru, R. Marciano, A. Rajasekar and M. Wan, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 5. Morgan Kaufmann, first edition (1999).

90. C. Moretti, T. C. Faltemier, D. Thain and P. J. Flynn, Challenges in executing data intensive biometric workloads on a desktop grid. In *Proc. Workshop on Large Scale and Volatile Desktop Grids* (2007).

91. C. S. R. Murthy and G. Manimaran, *Resource Management in Real-time Systems and Networks*. MIT Press (April 2001).

92. M. H. Ng, S. Johnston, S. Murdock, B. Wu, K. Tai, H. Fangohr, S. Cox, J. W. Essex, M. Sansom and P. Jeffreys, Efficient data storage and analysis for generic biomolecular simulation data. In *Proc. UK e-Science All Hands Meeting* (2004).

93. M. H. Nga, S. Johnston, B. Wuc, S. E. Murdock, K. Tai, H. Fangohr, S. J. Cox, J. W. Essex, M. S. Sansom and P. Jeffreys, BioSimGrid: Grid-enabled biomolecular simulation data storage and analysis. *Future Generation Computer Systems*, 22(6) (2006).

94. E. B. Nightingale, P. M. Chen and J. Flinn, Speculative execution in a distributed file system. *ACM Transactions on Computer Systems*, 24(4) (2006).

95. NS-2 software package. `http://www.isi.edu/nsnam/ns`.

96. J. O'Donahue, A. X. Yang and K. Mittal, *Java Database Programming Bible*. Wiley Publishing, Inc. (2002).

97. J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson and B. B. Welch, The Sprite network operating system. *IEEE Computer*, 21 (1988).

98. V. Pande, I. Baker, J. Chapman, S. P. Elmer, S. Khaliq, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. Snow, E. Sorin and B. Zagrovic, Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Biopolymers*, 68 (2003).

99. M. Parashar and S. Hariri, Autonomic computing: An overview. *LCNS*, 3566 (2004).

100. D. Patterson, G. Gibson and R. Katz, A case for redundant arrays of inexpensive disks (RAID). In *Proc. Management of Data* (1988).

101. IEEE/ANSI Std. 1003.1. Portable operating system interface (POSIX)-Application Program Interface (API) [C language] (1996).

102. J. Postel and J. Reynolds, Telnet protocol specification. *IETF RFC 854* (1983).

103. R. M. Rahman, K. Barker and R. Alhajj, Study of different replica placement and maintenance strategies in data grid. In *Proc. Cluster Computing and the Grid* (2007).

104. A. Rajasekar, M. Wan, R. Moore, G. Kremenek and T. Guptill, Data grids, collections and grid bricks. In *Proc. Mass Storage Systems and Technologies* (2003).

105. A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen and R. Olschanowsky, Storage Resource Broker - Managing distributed data in a grid. *Computer Society of India Journal*, 33(4) (2003).

106. A. K. Rajasekar and R. W. Moore, Data and metadata collections for scientific applications. In *Proc. European High Performance Computing* (2001).

107. A. Rajsekar, M. Wan, R. W. Moore and W. Schroeder, Data grid federation. In *Proc. Parallel and Distributed Processing Techniques and Applications* (2004).

108. A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers and M. Samidi, Scheduling data-intensive workflows onto storage-constrained distributed resources. In *Proc. Cluster Computing and the Grid* (2007).

109. R. Raman, M. Livny and M. H. Solomon, Matchmaking: Distributed resource management for high throughput computing. In *Proc. High Performance Distributed Computing* (1998).

110. K. Ranganathan and I. Foster, Simulation studies of computation and data scheduling algorithms for data grids. *J. Grid Computing*, 1(1) (2003).

111. C. F. Reilly and J. F. Naughton, Exploring provenance in a distributed job execution system. In *Proc. International Provenance and Annotation Workshop* (2006).

112. B. Reiner and K. Hahn, Optimized management of large-scale data sets stored on tertiary storage systems. *Distributed Systems Online*, 5(5) (2004).

113. M. Reiser and S. S. Lavenberg, Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2) (1980).

114. M. Ripeanu and I. Foster, A decentralized, adaptive, replica location service. In *Proc. High Performance Distributed Computing* (2002).

115. T. Roblitz, F. Schintke, A. Reinefeld, O. Barring, M. B. Lopez, G. Cancio, S. Chapeland, K. Chouikh, L. Cons, P. Poznanski, P. Defert, J. Iven, T. Kleinwort, B. Panzer-Steindel, J. Polok, C. Rafflin, A. Silverman, T. Smith, J. V. Eldik, D. Front, M. Biasotto, C. Aiftimiei, E. Ferro, G. Maron, A. Chierici, L. Dellagnello, M. Serra, M. Michelotto, L. Hess, V. Lindenstruth, F. Pister, T. M. Steinbeck, D. Groep, M. S. O. Koeroo, W. S. de Cerff, G. Venekamp, P. Anderson, T. Colles, A. Holt, A. Scobie, M. George, A. Washbrook and R. A. G. Leiva, Autonomic management of large clusters and their integration into the grid. *J. Grid Computing*, 2 (2004).

116. T. Ryutov, G. Gheorghiu and C. Neuman, An authorization framework for metacomputing applications. In *Proc. Cluster Computing* (1999).

117. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh and B. Lyon, Design and implementation of the Sun Network File System. In *Proc. USENIX* (1985).

118. M. Satyanarayanan, Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5) (May 1990).

119. T. Schlick, *Molecular Modeling and Simulation - An Interdisciplinary Guide*. Springer-Verlag, New York, NY (2002).

120. S. Shinners, *Modern Control System Theory and Design*. Wiley Interscience (1998).

121. E. Sit, A. Haeberlen, F. Dabek, B.-G. Chun, H. Weatherspoon, R. Morris, M. F. Kaashoek and J. Kubiatowicz, Proactive replication for data durability. In *Proc. Workshop on Peer-to-Peer Systems* (2006).

122. R. D. Skeel and J. A. Izaguirre, An impulse integrator for Langevin dynamics. *Molecular Physics*, 100(24) (2002).

123. W. Smith, I. Foster and V. Taylor, Scheduling with advanced reservations. In *Proc. International Parallel and Distributed Processing Symposium* (2000).

124. P. Stelling, C. DeMatteis, I. T. Foster, C. Kesselman, C. A. Lee and G. von Laszewski, A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2) (1999).

125. H. Stockinger, Defining the grid: a snapshot on the current view. *J. Supercomputing*, 42(1) (2007).

126. H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman and B. Tierney, File and object replication in data grids. In *Proc. High Performance Distributed Computing* (2001).

127. V. S. Sunderam, PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4) (December 1990).

128. K. Tai, S. Murdock, B. Wu, M. Ng, S. Johnston, H. Fanghor, S. J. Cox, P. Jeffreys, J. W. Essex and M. S. P. Sansom, BioSimGrid: Towards a worldwide repository for biomolecular simulations. *Organic and Biomolecular Chemistry*, 2 (2004).

129. A. Takefusa, H. Casanova, S. Matsuoka and F. Berman, A study of deadline scheduling for client-server systems on the computational grid. *Proc. High Performance Distributed Computing* (2001).

130. A. Takefusa, S. Matsuoka, H. Nakada, K. Aida and U. Nagashima, Overview of a perfomance evaluation for global computing scheduling algorithms. In *Proc. High Performance Distributed Computing* (1999).

131. A. S. Tanenbaum, R. van Renesse, H. van Staveren, G. J. Sharp and S. J. Mullender, Experiences with the Amoeba distributed operating system. *Communications of the ACM*, 33(12) (1990).

132. O. Tatebe, N. Soda, Y. Morita, S. Matsuoka and S. Sekiguchi, Gfarm v2: A grid file system that supports high-performance distributed and parallel data computing. In *Proc. Computing in High Energy and Nuclear Physics* (2004).

133. D. Thain, S. Klous, J. Wozniak, P. Brenner, A. Striegel and J. Izaguirre, Separating abstractions from resources in a tactical storage system. In *Proc. Supercomputing* (2005).

134. D. Thain and M. Livny, Bypass: A tool for building split execution systems. In *Proc. High Performance Distributed Computing* (2000).

135. D. Thain and M. Livny, Parrot: Transparent user-level middleware for data-intensive computing. In *Proc. Workshop on Adaptive Grid Middleware* (September 2003).

136. D. Thain and C. Moretti, Efficient access to many small files in a filesystem for grid computing. In *Proc. Conference on Grid Computing* (2007).

137. D. Thain, C. Moretti and J. Hemmes, Chirp: A practical global file system for cluster and grid computing. Technical Report 2007-04, University of Notre Dame (2007).

138. D. Thain, T. Tannenbaum and M. Livny, Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience* (2004).

139. Y. Tohma, Incorporating fault tolerance into an autonomic-computing environment. *Distributed Systems Online*, 5(2) (2004).

140. S. Vazhkudai, S. Tuecke and I. Foster, Replica selection in the Globus data grid. In *Proc. Cluster Computing and the Grid* (2001).

141. S. S. Vazhkudai, X. Ma, V. W. Freeh, J. W. Strickland, N. Tammineedi, T. Simon and S. L. Scott, Constructing collaborative desktop storage caches for large scientific datasets. *ACM Transactions on Storage*, 2(3) (2006).

142. J. S. Vetter and D. A. Reed, Real-time performance monitoring, adaptive control, and interactive steering of computational grids. *International Journal of High Performance Computing Applications*, 14(4) (2000).

143. G. von Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton and A. Rossi, Gridant: A client-controllable grid workflow system. In *Hawaii International Conference on System Science* (2004).

144. G. von Laszewski, J. Gawor, P. Lane, N. Rehn, M. Russell and K. Jackson, Features of the Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 14(13-15) (2002).

145. A. Voter, A method for accelerating the molecular dynamics simulation of infrequent events. *J. Chem. Phys.*, 106(11) (1997).

146. A. F. Voter, Hyperdynamics: accelerated molecular dynamics of infrequent events. *Phys. Rev. Lett.*, 78 (1997).

147. Web services architecture (2004), W3C Web Services Architecture Working Group.

148. B. Walker, G. Popek, R. English, C. Kline and G. Thiel, The LOCUS distributed operating system. In *Proc. Symposium on Operating Systems Principles* (1983).

149. D. Walker, The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, 20(4) (1994).

150. E. Walker, T. Minyard and J. Boisseau., GridShell: A login shell for orchestrating and coordinating applications in a grid enabled environment. In *Proc. Computing, Communications and Control Technologies* (2004).

151. A. Walsh, J. Couch and D. H. Steinberg, *Java 2 Bible*. IDG Books (2000).

152. M. Wan, A. Rajasekar, R. Moore and P. Andrew, A simple mass storage system for the SRB data grid. In *Proc. Mass Storage Systems and Technologies* (2003).

153. H. Weatherspoon and J. Kubiatowicz, Erasure coding vs. replication: A quantitative comparison. In *Proc. Workshop on Peer-to-Peer Systems* (2002).

154. B. S. White, A. S. Grimshaw and A. Nguyen-Tuong, Grid-based file access: The Legion I/O model. In *Proc. High Performance Distributed Computing* (2000).

155. B. S. White, M. Walker, M. Humphrey and A. S. Grimshaw, LegionFS: A secure and scalable file system supporting cross-domain high-performance applications. In *Proc. Supercomputing* (2001).

156. C. J. Woods, M. H. Ng, S. Johnston, S. E. Murdock, B. Wu, K. Tai, H. Fangohr, P. Jeffreys, S. Cox, J. G. Frey, M. S. P. Sansom and J. W. Essex, Grid computing and biomolecular simulation. *Philosophical Transactions of the Royal Society A*, 363(1833) (2005).

157. WordNet 2.0.

158. J. M. Wozniak, P. Brenner, D. Thain, A. Striegel and J. A. Izaguirre, Generosity and gluttony in GEMS: Grid-Enabled Molecular Simulation. In *Proc. High Performance Distributed Computing* (2005).

159. J. M. Wozniak, P. Brenner, D. Thain, A. Striegel and J. A. Izaguirre, Access control for a replica management database. In *Proc. Workshop on Storage Security and Survivability* (2006).

160. J. M. Wozniak, P. Brenner, D. Thain, A. Striegel and J. A. Izaguirre, Applying feedback control to a replica management system. In *Proc. Southeastern Symposium on System Theory* (2006).

161. J. M. Wozniak, P. Brenner, D. Thain, A. Striegel and J. A. Izaguirre, Making the best of a bad situation: Prioritized storage management in GEMS. *Future Generation Computer Systems*, 24(1) (2007).

162. J. M. Wozniak, Y. Jiang and A. Striegel, Effects of low-quality computation time estimates in policed schedulers. In *Proc. Annual Simulation Symposium* (2007).

163. J. M. Wozniak, A. Striegel, D. Salyers and J. A. Izaguirre, GIPSE: Streamlining the management of simulation on the grid. In *Proc. Annual Simulation Symposium* (2005).

164. Q. Xin, E. Miller, T. Schwarz, D. D. E. Long, S. A. Brandt and W. Litwin, Reliability mechanisms for very large storage systems. In *Proc. Mass Storage Systems and Technologies* (2003).

165. Q. Xin, E. L. Miller and S. Thomas J. E. Schwarz, Evaluation of distributed recovery in large-scale storage systems. In *Proc. High Performance Distributed Computing* (2004).

166. Y. Zhao, M. Wilde, I. Foster, J. Voeckler, J. Dobson, E. Glibert, T. Jordan and E. Quigg, Virtual data grid middleware services for data-intensive science. In *Proc. Middleware* (2004).

167. S. Zhou, LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proc. Cluster Computing* (1992).

This document was prepared & typeset with LaTeX 2_ε, *and formatted with* NDdiss2_ε *classfile (v1.0[2004/06/15]) provided by Sameer Vijay.*