Generosity and Gluttony in GEMS:
Grid Enabled Molecular Simulations

J. M. Wozniak, P. Brenner, D. Thain, A. Striegel, J. A. Izaguirre
Dept. of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
{ *jwozniak, pbrenne1, dthain, striegel, izaguirr* } @ *nd.edu*

Abstract

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 reliable 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. The primary design problem is striking a balance between generosity and gluttony. On one hand, storage providers wish to be generous and share resources with their collaborators. On the other hand, an unchecked data producer can be gluttonous and easily replicate data unnecessarily until it fills all available space. To balance generosity and gluttony, 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. We have built a prototype of GEMS on a cluster of workstations and demonstrate its ability to store new data, to replicate within policy limits, and to recover from failures.

1 Introduction

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. A common characteristic for simulations in these disciplines is the ability to generate large amounts of data with complex relationships.

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 system 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.

To meet the data demands of researchers performing biomolecular simulations with PROTOMOL, we introduce the GEMS toolkit: Grid Enabled Molecular Simulations. GEMS is a wide area distributed system for managing the *storage*, *searching*, and *sharing* needs of collaborating researchers. The primary design and engineering problem of GEMS is striking a balance between the *generosity* of storage providers and the *gluttony* of data producers. On one hand, storage owners have many motivations to be generous and share their resources with collaborators. On the other hand, data owners have many competing motivations to be gluttonous and fill all available storage, which would poorly utilize the system and impede progress in the scientific problem at hand.

Consider these forms of generosity in a distributed data system. Independent producers of data may wish to share archived, high value simulation results with others to disseminate their scientific results. Collaborating scientists may be willing to share unused local storage space with each other for scratch space or distributed backups. Of course, generosity has its limits. Resource owners may be willing to allow unused space to be shared, but will demand it back if their needs increase. Few resource owners will wish to share with the world at large: most will want to share only with a well-defined set of collaborators.

If we allow storage owners to express generosity, we must also be prepared for data producers to engage in gluttony. When it is easy to replicate high value data in order to prevent it from loss, we must expect that all users will replicate widely, thus filling up all available storage very quickly. Once storage is filled, we are presented with some difficult questions. In order to make space for new data, what items must be discarded? The users of a system must have some method of expressing which items have high value and which items do not.

GEMS allows users to strike a balance between generosity and gluttony, thus ensuring the accessibility, performance, and survivability of data. To accomplish this, 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.

In this paper, we describe the design and architecture of GEMS. 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 an experimental study, we demonstrate how GEMS is able to deal with changing constraints in a dynamic system. This discussion yields several insights into the behavior of a distributed replica storage system.

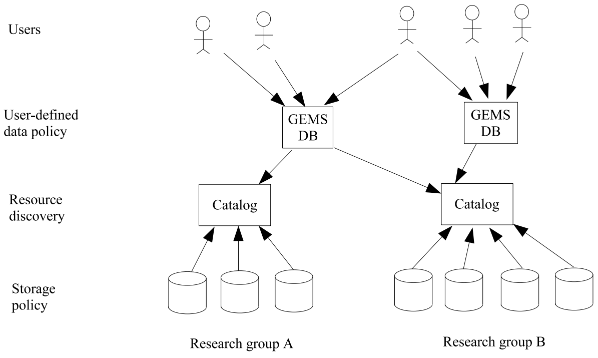

Figure 1. Overview of GEMS Architecture

2 GEMS Architecture

Figure 1 shows the major components of the GEMS architecture that include *storage servers*, *catalog servers*, and *database servers*. The GEMS process begins when the user submits data for storage. With each storage 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.

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 gluttonous replication followed by larger area storage 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.

Similar to public archival, GEMS 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 copying all files locally, new computation is submitted remotely with the new output results owned by the user. Replication of the resulting new data is still the responsibility of the GEMS software.

Finally, GEMS expands upon the virtual data concept of Chimera [10] when offering public archival functionality. Since the majority of the

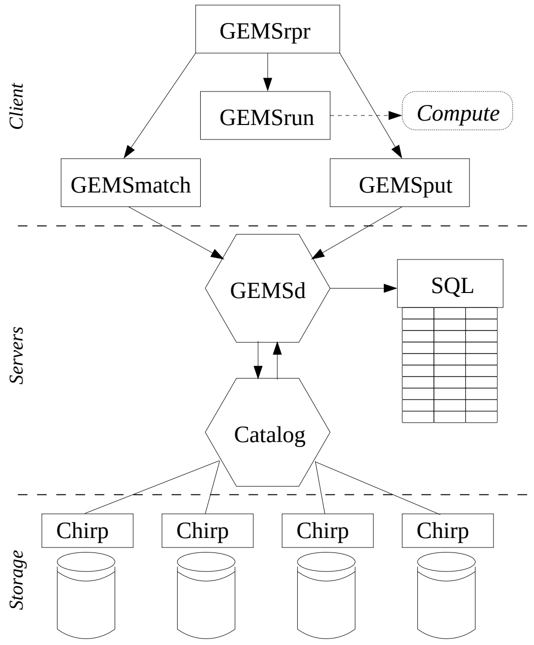

Figure 2. GEMS Toolset Framework.

simulations will create massive output files, it may be less costly to dynamically recompute requested data rather than storing multiple copies of the complete results. Hence, 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 can take advantage of the deterministic nature of many biomolecular simulations to provide auto-regeneration of missing points without re-running the entire simulation. A future paper will discuss GEMS' ability to catalog the structure of deterministic simulations, as well as more complex structures created by protein conformation analysis.

In short, GEMS can use a small amount of knowledge of the scientific value of the data to help solve problems created by the distributed, shared nature of the storage system. The fact that input files are very important and very small allows gluttonous users to replicate heavily and benefit from the system. The fact that output files are large but recomputable is used to protect generous storage providers from being exploited unnecessarily.

3 The GEMS Prototype

Figure 2 shows a conceptual diagram of the GEMS toolset and its interaction with existing resources and grid management tools.

We have constructed a prototype implementation of GEMS using the following tools. The storage server used is the Chirp [22] 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 [15]. 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.

3.1 Client Tools

Users are given client tools to perform three basic operations: *GEMSput*, *GEMSmatch*, and *GEMSrun*.

GEMSput allows the user to specify a completed simulation along with its input and output data files and appropriate metadata. GEMSmatch allows the user to locate existing records that match criteria specified in XML. 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. GEMSrun plugs together GEMS, PROTOMOL, and a batch system in an easy-to-use interface. This tool extracts an existing output set from GEMS, uses it as input to a PROTOMOL job, and then runs it in a batch system.

In addition, users can also submit more complicated requests utilizing all three operations described above through the use of a Result Production Request (rpr) via the GEMSrpr client. If the desired simulation has already been produced, GEMSrpr will detect this by using GEMSmatch to return the location of the outputs. If not, GEMSrun is used to dispatch a job to a batch system and produce the desired data. The outputs will be stored in GEMS with GEMSput and returned to the user in the same way. Thus, GEMSrpr does not contact the server directly, but uses our other software.

3.2 Database Server

The database server is the hub of GEMS. It is responsible for serving users, managing metadata, and periodically scanning storage servers for problems.

Each PROTOMOL computation has a record in the database indicating how the computation could be redone. This includes metadata about the executable, parameters, and input and output files. In addition, each file has metadata describing its initial location, 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 gone 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, the database server views all these failure modes as a loss of data.

Upon discovering the loss of a replica, the database server will search for a surviving copy of that data. If none is found, then permanent data loss has occurred, and this event is logged. If a replica is available, the database server consults the catalog to find available space, and then creates a new replica. By default, the database server handles short term loss quite aggressively. The temporary loss of a storage server due to a reboot or network failure results in rapid replication of the lost data. A data owner may throttle this behavior by requiring a configurable amount of time to pass before a perceived failure results in a replication.

Because of the possibility of over-replication, the database server is also responsible for the opposite task of cleaning up unwanted data. Periodically, the database server scans known storage servers for data that is present but unmentioned in the database. We call this task *auditing*. If it represents data already replicated and that puts the user over the replication threshhold, it is deleted. Note that GEMS uses a private directory on each storage server; files unrelated to GEMS will not be removed during auditing.

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 set by the Replicator and Auditor. 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 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, we chose to delegate control authority across the Chirp servers, managed by the GEMSd server. 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 ACL (Access Control List) 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, 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

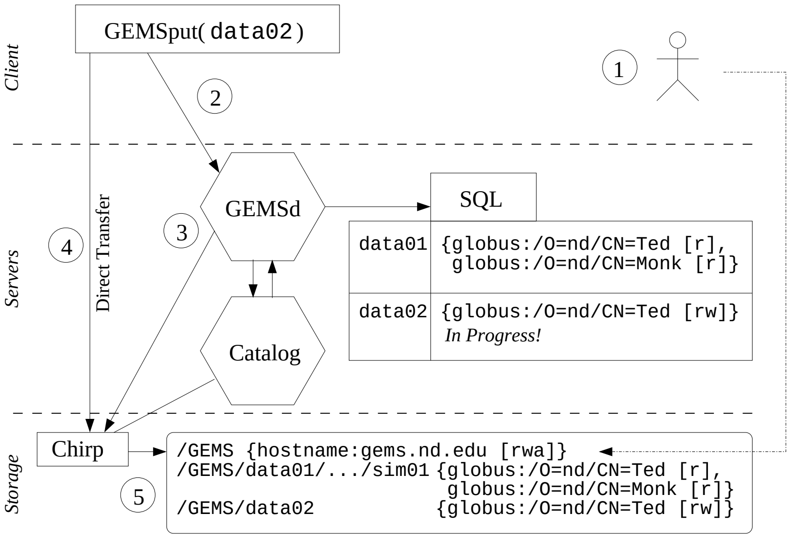
GEMSput(data02)
Client
Servers
Storage
1
2
3
4
5
Direct Transfer
GEMSd
SQL
Catalog
Chirp
data01 {globus:/O=nd/CN=Ted [r],
globus:/O=nd/CN=Monk [r]}
data02 {globus:/O=nd/CN=Ted [rw]}
In Progress!
/GEMS {hostname:gems.nd.edu [rwa]}
/GEMS/data01/.../sim01 {globus:/O=nd/CN=Ted [r],
globus:/O=nd/CN=Monk [r]}
/GEMS/data02 {globus:/O=nd/CN=Ted [rw]}

Figure 3. 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 GEMSd access to administer a directory. 2. The owner of `data02` *states the ACL to be used for protecting the new dataset. 3. GEMSd creates a new directory on the file server and sets the ACL for that dataset. 4. Direct data transfer to the file server begins. 5. As shown for* `data01`*, existing data is directly readable by certain users.*

this fileset is performed with the constraint that the Chirp servers in question accept the authentication method given by the corresponding ACL.

4 Application: Molecular Dynamics Simulation

Molecular dynamics describes 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 [**?**], that we utilize 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 such as a PDB position file (from the national Protein Data Bank repository), a CHARMM force parameter file, and a PSF topology file. The researcher will take the initial conformation in a selected environment, apply a random variation, and allow 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 in protein docking software such as FlexE or AutoDock.

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:

- *Auto-regeneration of data:* By virtue of the deterministic state of the simulations, missing data points can be regenerated without rerunning the entire simulation. For example, data point DP_{N+1} can be regenerated by running the original executable using the program inputs and DP_N. Unlike checkpointing, special libraries are not employed for this functional-

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

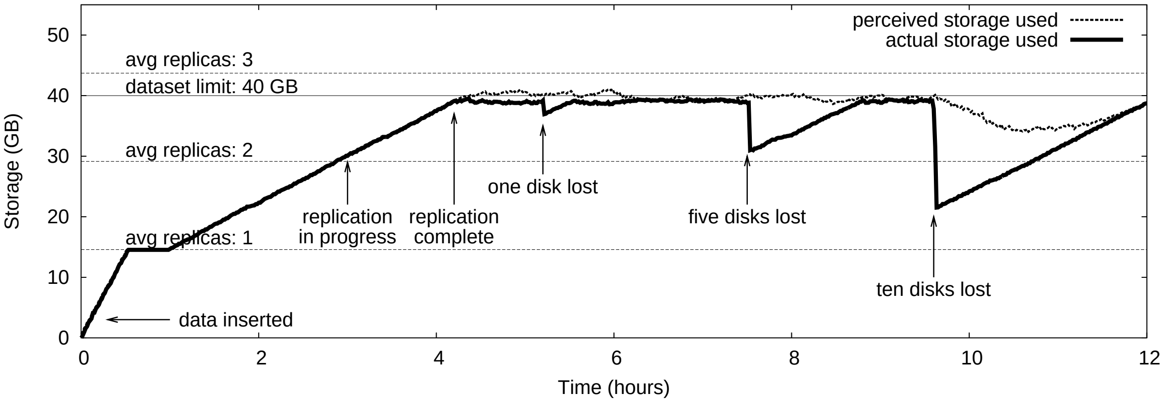

Figure 4. Example of Fault Tolerance over Time

This figure shows the first twelve hours in the lifetime of a 14 GB dataset entered into GEMS for safekeeping. Out of a storage cluster of 20 disks, the system policy has allotted GEMS 40 GB to store this dataset. 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.

ity and only a subset of the simulation state is necessary for these intermediate states.

- *Multiple priority levels:* Since data can be automatically regenerated by the system, it becomes possible to offer different levels of redundancy between the different types of data. While critical components such as the original executable and inputs must be preserved, the final output and intermediate data points can be stored at a lower redundancy level. In addition, data that exists primarily for archival use may be further reduced in redundancy provided that the initial conditions for creation are not lost.

- *Search for existing data:* The ability to store massive amounts of data becomes useless if the data cannot be retrieved in a meaningful fashion. In order to enable evolving research, it must be possible to employ both search queries based on well-defined data structures as well as user-customized fields.

The most 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 viewing data blocks as simply striving towards a uniform redundancy level, the importance of the data can be taken into account when reacting to inevitable failures. The gluttonous nature of GEMS will attempt to maximize redundancy levels of critical data but yet the generosity aspect will yield space as necessary 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 such as BioSimGrid [21].

5 Experiments

In this section, we demonstrate the usefulness of GEMS on actual simulation data. GEMSd relies upon several other resources that are used for data and metadata storage, as well as compute hosts. In our tests, we used 20 storage servers and a dual-processor GEMSd and metadatabase server, all of which were running Linux. The server used PostGreSQL 7.4.6 [14] for the metadatabase.

We present a 12 hour experiment during which all machines remained available to other non-GEMS users. We chart the storage space utilized as reported by GEMS along with the actual physical storage utilized. 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 users.

By monitoring global system storage knowledge, GEMS can make adaptive replication count decisions. In this test, we assume all stored data is output data and is subject to tight replication limits. The Replicator and Auditor components then react appropriately, replicating if available storage permits and deleting as storage becomes limited.

Figure 4 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 Replicator component of GEMS is not running. To start, multiple GEMSputs 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 Replicator 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. The Auditor and Replicator then 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 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. Interestingly, for the smaller storage losses GEMS records little to no deviation in total capacity. The Replicator continues from its current point in the metadata 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. In case of disk failure, or if a storage donor suddenly leaves the system, GEMS will recover by recopying the backup replicas.

6 Related Work

Figure 5 shows how GEMS relates to other work in distributed storage. It lies at the middle of two axes in design: database/filesystem and centralized/peer-to-peer.

To end users, GEMS is something like a database and also something like a filesystem. It provides a database-like abstraction for inserting, indexing, and managing replicated data. But, the objects stored are filesystem components and can be accessed through an ordinary filesystem interface. This duality allows GEMS to take advantage of aspects of both models. The database aspect allows GEMS to easily handle issues of replication, fault tolerance, and consistency, while the filesystem aspect allows for native interaction with existing tools and applications.

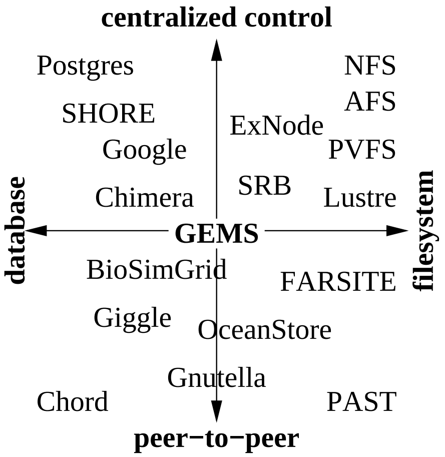

Figure 5. Related Work

In the distribution of control, GEMS is a mix of centralized and peer-to-peer. Each storage server in GEMS is independent and has its own policies that control who is allowed to consume space. Every resource owner may contribute as little or as much to the system as they like while retaining the capability to retract storage at any time. However, the GEMS catalogs and databases used to locate existing space and available data are shared among many users. In a typical configuration, many users would share a small number of catalogs and databases, but conceivably each could have their own.

Other distributed storage systems inhabit other positions in this coordinate system, many at the extremes of one axis or the other. We begin in the upper right hand corner and proceed clockwise around the graph. For example, a conventional distributed filesystem such as NFS [18] has completely centralized administrative control. Over time, research in distributed filesystems has slowly pushed the boundaries of control structures outward. AFS [11] permits clients to exercise more control over cache consistency. PVFS [7] and Lustre [20] spread data and metadata across multiple hosts in a cluster to achieve high throughput and expandable capacity. However, these systems all retain central administrative control. In contrast, each storage server in GEMS is completely independent.

Some systems have explored complete independence. FARSITE [1], OceanStore [12], and PAST [17] all provide a (nearly) unmodified filesystem interface on top of a set of completely untrusted storage servers. The complete distrust of storage devices requires these systems to take expensive and extreme measures such as Byzantine agreement in order to provide some degree of confidence in an unreliable environment. In contrast, GEMS assumes some degree of cooperation between users and does not guard against malice.

If a system is willing to sacrifice some of the filesystem interface, it can achieve greater availability and fault tolerance by providing a more database-like functionality. For example, a fully peer-to-peer file sharing service such as Gnutella [16] only provides the ability to store and fetch files. Even more database-like is a distributed hash table such as Chord [19], which only provides the ability to map hash keys to storage locations. GEMS provides much of the lookup capability of these systems, but retains a filesystem interface to local data.

As we move toward the upper left hand corner, we approach systems that look more like a traditional database. A search engine such as Google [5] is essentially a database of individual files. Although Google has an enormous scale, it still represents a single administrative domain in which all components work cooperatively. Even closer is SHORE [6], which combines both structured and unstructured data into a single hierarchical view. Due to the complete independence of its storage devices, GEMS cannot provide the fine-grained consistency semantics of a traditional database.

There are four systems similar to GEMS in the grid computing domain: Giggle, SRB, BioSimGrid, and Chimera. Each provides varying degrees of independence and data abstraction. All of these systems may potentially need to deal with issues of gluttony and generosity.

Giggle [8] is a replication location service that indexes existing storage archives and allows users to locate the nearest copy of a known data item. SRB [3] brings both filesystems and databases together into a federated system that allows for uniform indexing and access of multiple data sources. GEMS builds upon the concepts of Giggle and SRB by making each index itself partially responsible for replication and management decisions. The user decides what steady state is to be maintained, and then delegates that job to the database server.

BioSimGrid [21] and Chimera [10] are systems for indexing and sharing experimental data between multiple researchers. Chimera in particular allows a user to define how data may be realized automatically, and then does so upon request. GEMS builds upon these two systems by allowing the participating storage sites complete technical independence. In GEMS, each storage site is free to join, participate, and withdraw from the system within their own local constraints. Thus, the placement of newly-generated data within GEMS must be done with some consideration of the data value and the owner of the storage space.

A data management system can be built using a variety of underlying storage elements. GridFTP [2] provides secure, parallel-stream access to legacy FTP systems. IBP [13] makes storage accessible through a malloc-like interface with access control via capabilities. NeST [4] provides unified access to grid storage through a variety of protocols. GEMS could potentially use any of these storage devices as the underlying fabric. We have employed Chirp [22] as the basic storage element because it permits fine-grained data sharing policies as well as direct runtime access to filesystem fragments.

7 Conclusion

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.

This paper did not discuss the fundamental design capabilities of GEMS to catalog and store data dependencies specific to our target application, molecular dynamics and protein conformations. We also did not thoroughly present the simulation metadata specific to the scientific problem, which is important for these capabilities, as well as for comparison with systems such as BioSimGrid [21]. An area to investigate in the future is the interface to the compute grid. We currently use the shell to spawn computation, but are targeting Condor [9] and GIPSE [23]. These topics will be the subject of other papers.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Operating Systems Design and Implementation*, Boston, MA, Dec. 2002.

[2] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, and S. Tuecke. Protocols and services for distributed data-intensive science. In *Proceedings of Advanced Computing and Analysis Techniques in Physics Research*, pages 161–163, 2000.

[3] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC storage resource broker. In *Proceedings of CASCON*, Toronto, Canada, 1998.

[4] J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Flexibility, manageability, and performance in a grid storage appliance. In *Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing*, Edinburgh, Scotland, July 2002.

[5] S. Brin and L. Page. The anatomy of a large scale hypertextual search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[6] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. Shoring up persistent applications. In *ACM SIGMOD Management of Data*, June 1994.

[7] P. H. Carns, W. B. Ligon, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Annual Linux Showcase and Conference*, 2000.

[8] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B, Schwartzkopf, H, Stockinger, K. Stockinger, and B. Tierney. Giggle: A framework for constructing scalable replica location services. In *Supercomputing*, November 2002.

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

[10] I. Foster, J. Voeckler, M. Wilde, and Y. Zhou. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.

[11] 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):51–81, February 1988.

[12] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Architectural Support for Programming Languages and Operating Systems*, 2000.

[13] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In *Proceedings of the Network Storage Symposium*, 1999.

[14] The PostgreSQL Global Development Group. *PostGreSQL Documentation.* `www.postgresql.org/docs/` .

[15] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 1998.

[16] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *International Conference on Peer-to-peer Computing*, August 2001.

[17] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent, peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, October 2001.

[18] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network filesystem. In *Proceedings of the USENIX Summer Technical Conference*, pages 119–130, 1985.

[19] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashock, and H. Balakrishman. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *ACM SIGCOMM*, pages 149–160, August 2001.

[20] C. F. Systems. Lustre: A scalable, high performance file system. White paper, November 2002.

[21] K. Tai, S. Murdock, B. Wu, M. H. Ng, S. Johnston, H. Fangohr, S. J. Cox, P. Jeffreys, J. W. Essex, and M. S. P. Sansom. BioSimGrid: Towards a worldwide repository for biomolecular simulations. *Organic Biomolecular Chemistry*, 2:3219–3221, 2004.

[22] D. Thain. Chirp: An architecture for cooperative storage. Technical Report 2005-02, University of Notre Dame, Computer Science and Engineering Department, February 2005.

[23] J. M. Wozniak, A. Striegel, D. Salyers, and J. A. Izaguirre. GIPSE: Streamlining the management of simulation on the grid. In *Proceedings of the 38th Annual Simulation Symposium.* IEEE Computer Society, 2005.