GEMS: User Control for Cooperative Scientific Repositories

Justin M. Wozniak[*], Paul Brenner[†], Santanu Chatterjee[†],
Douglas Thain[†], Aaron Striegel[†], and Jesús Izaguirre[†]
[*] Argonne National Laboratory, Argonne, IL USA
Email: wozniak@mcs.anl.gov
[†] University of Notre Dame, Notre Dame, IN USA
Email: {pbrenne1,schatter,dthain,striegel,jizaguirr}@nd.edu

Opportunistic techniques have been widely used to create computational infrastructures and have demonstrated an ability to deliver computing resources to large applications. However, the management of disk space and usage in such systems is often neglected. While overarching filesystems have been applied, these limit the ability of subgroups within an organization to customize and optimize system behavior. Explicit data placement techniques offer more utility but may bury users with the complexity of system specifics and scripts. New solutions to storage problems will require new approaches to data abstraction, archive survivability, security models, and data delivery to consumers. In this chapter, we describe design concepts used in the GEMS storage system within which users may specify abstract resource structures and policies for their data.

I. Introduction

Data repositories are an integral part of modern scientific computing systems. While a variety of grid-enabled storage systems have been developed to improve scalability, administrative control, and interoperability, users have several outstanding needs: to seamlessly and efficiently 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 evident in the *small virtual organization*, 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 seeks to enable this functionality by creating *ad hoc storage grids*.

Modern commodity hardware used at research labs and university networks ships with an abundance of storage space that is often underutilized, and even consumer gadgets provide extensive storage resources that will not immediately be filled. The installation of simple software enables these systems to be pooled and cataloged into a spacious, parallel ad hoc storage network. While traditional storage networks or tertiary storage systems are isolated behind file servers and firewalls, constricting data movement, we layer the storage service network atop the client consumer network, improving the available network parallelism and boosting I/O performance for data-intensive scientific tasks.

Unfortunately, extending scientific repositories out to volunteer data sources and across administrative boundaries using centralized file services, static replica management systems, or peer-to-peer systems is not viable because of the nature of target systems. Heavy scientific workloads, research-oriented usage patterns, and variably collaborative resource networks require new storage architectures.

Our focus application area, computational chemistry simulations, often produce heavy output data sets, transcripts, etc., compared to amount of metadata that is used to specify the data set, such as the simulation configuration, random seed, file names, and access control policy. Metadata may be used to refer to existing data sets but is typically much smaller than the files themselves. Programmatic access to metadata catalogs creates a *scientific filesystem* that greatly eases access to complex repositories. Our architecture is inspired by typical scientific usage: optimizing system performance for the demands of typical software and interfaces for the mental framework of typical simulation researchers.

The dynamic nature of ad hoc storage networks allows scientific users to easily provide storage resources to the system, insert data sets, and manage access control to resources and data. Hence, researchers - not just computer administrators - can patch together their own communities of software, data, resources and users while integrating with a larger repository system and benefiting from semi-autonomic storage management.

Current systems limit user customizability, which enables locally optimal performance, and ease of utility, which enables maintainable software. Our solution is to develop of flexible systems, providing a variety of access methods and administrative controls. The policies carried out by flexible *ad hoc* 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 privileges will be administered by users on their own data environments. Applications developed on such a system can use this hybridization, enabling system-aware workflow structures.

Our solution to this problem is represented by the GEMS (Grid-Enabled Molecular Simulation) system. Programmatically, GEMS presents a reliable database abstraction atop an uncontrolled network of independent Chirp [1] file servers. GEMS was originally designed for molecular dynamics but has been fully generalized to support any application. In this chapter, we discuss the advanced capabilities of this system for scientific computation and collaboration.

II. GEMS Architecture

The GEMS architecture, shown in Figure 1 is intended to provide a quickly deployable organizational framework for a network of independently owned and operated storage services. In an institutional setting, these machines may already be combined into an opportunistic computing system such as Condor, but the available disk space may go unused. The storage services are available on these machines but not usable because of their individual instability. By structuring and controlling these servers, GEMS gains the utility of the extra disk space as well as the performance offered by a data delivery system that is potentially as scalable as the computing infrastructure.

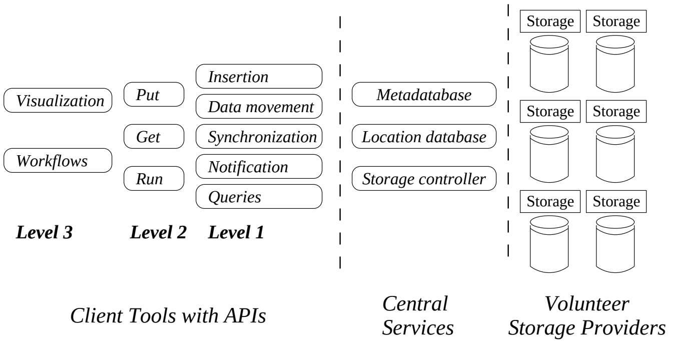


Fig. 1. Overview of the GEMS architecture.

GEMS delivers the utility of these services as a unified resource, and relies on four major concepts: automated replication, awareness of administrative concerns, distributed access control, and centralized browsability. Replication increases the reliability and parallelism of data sources; in GEMS, a semi-autonomic controller [2] detects and responds to subsystem failure in a priority-driven manner. Additionally, administrative matters are managed within the same control system: disk allocation is balanced with respect to the available space volunteered to the system, and data is removed from disks as their owners fill them. GEMS also manages access control lists in the volunteered space and provides an authentication mechanism [3] compatible with existing techniques. Observed deviations from system policy are prioritized and repaired [4] without user intervention, by locating, allocating, and using additional available disk space in accordance with user policy.

The browsable repository created by GEMS essentially implements a parameter sweep database. This metadata may be used to derive a novel parallel programming model that enables synchronized tuple space communication and parallel data movement, called a *data sweep*. This framework may be used to build up parameter sweeps, workflows, interactive parameter space exploration, and related hybrid structures.

A. Runtime Operation

GEMS maintains a replica list for each data file inserted into the system, which asynchronously

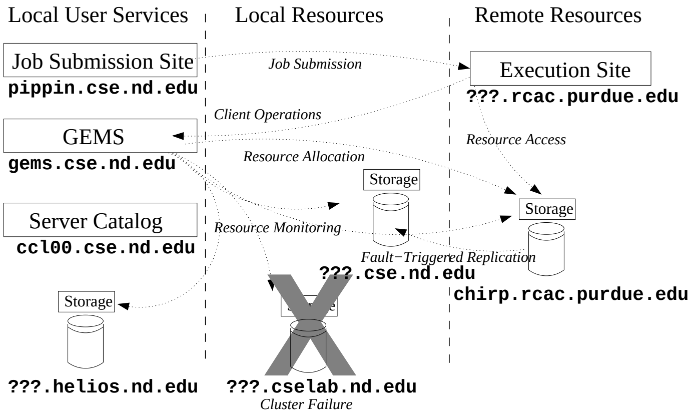


Fig. 2. Example operation of replica maintenance system. Replica shortages are repaired while data services to running jobs continue.

grows as the file is replicated in accordance with user policy. Management aspects include monitoring the underlying storage for failure, file existence, and file corruption via an MD5 comparison. A diagram of example operation is shown in Figure 2.

Limiting the control software system to user level limits its ability to abuse resource providers and enables the opportunistic use of available storage. Whereas a top-down stovepipe solution might intimidate storage providers, resulting in low volunteer participation, the controller relies on an existing intermediary resource discovery service and interacts with registered services as an ordinary user. This approach promotes the healthy use of ordinary policy negotiation among volunteer resource providers and the consumers of the *ad hoc* grid [5].

B. Client Toolkit

While existing tools may be used to access files stored on the underlying system, GEMS enables the orchestration of higher-level operations by providing additional tiered client operations as diagrammed in Figure 1. First-level utilities enable resource reservation and insertion. These allow a client to reserve a given amount of space on a resource - selected by the user or by GEMS - for later use. Data movement utilities may be used to transport the data to that location, in addition to the use of existing clients compatible with the storage services. Upon transfer completion, the state of the files is synchronized with the centralized service, which is then entrusted to manage them. Other clients waiting for notification with respect to this data set may then be notified, or the data set may be found and located later by executing metadata-driven queries.

Second-level utilities provide simplified access to the system, including FTP-like get and put operations as well as a computation utility to bind existing programs to data sources in the repository without code modification. Third-level utilities enable higher level user interaction with the system. Graphical tools include a general-purpose data browser capable of downloading or uploading repository data. Additionally, workflow structures may be built and executed by using complex query and notification combinations, as discussed below.

III. Replica Access

Data replication has multiple benefits. While enhancing data survivability and enabling wide area storage systems as discussed above, it may be combined with computation systems to provide performance benefits. Large-scale computing infrastructures built on load-sharing architectures or Internet peer groups require scalable access to the scientific repositories that enable the application. High-throughput computing requires the elimination of bottlenecks, such as the overuse of a particular file server. A balance in disk usage may intuitively be obtained by balancing data placement: a user-driven process discussed above.

Replica-aware computation has several imporant aspects. Record naming and identification are an essential user tasks that may be greatly eased by appropriate software tools. Data access may be enabled by explicit data transfer or virtual filesystem technologies. Ultimately, efficiency and performance must be delivered by proper reference to the replica system.

A. Data Identification and Access

The underlying complexity of a replica system creates the need for abstract data catalogs that enable scientific data tagging and location. In the case of a simulation repository, for example, each record in the replica system contains the set of data files involved; each record is uniquely identified by a *config number*. The record may also be tagged by a unique set of key-value pairs, which are typically based on the parameters used to generate the data, such as software settings, scientific coefficients, and random number seeds. The *scientific namespace* is the set of possible tag combinations that may be mapped to a record.

While abstract naming systems are required in the presence of dynamic data placement, the use of a virtual filesystem creates an additional translation layer that must be overcome. The combination of a config number and a filename represents an *abstract* name for a file that is location independent with respect to data sources, that is, replica location. The combination of a file server name and a filename represents a *virtual* name for a file. The resolution of abstract name to virtual name is the essential task of a replica location system, but the existence of both names is useful in replica-aware computation.

The GEMS-based encapsulation of a set of scientific data files inside a logically unified, tagged record provides both scientific and technical benefits. Simulation result files are typically used in concert when performing analysis, and the abstract tagging and naming system simplifies the implementation of postprocessing routines. The naming and access scheme additionally helps mask the complex internals of a distributed database, providing conceptual understanding that may be lacking in a plain filesystem. Common tasks such as retrieving yesterday's results, searching for a critical remote file, or deleting a whole record may be performed with confidence.

Technically, the data structures created by the record system provide guidance when performing maintenance activities. For example, resource consumption may be allocated with respect to discernible dataset characteristics or prioritized with respect to user-defined importance.

B. Computation in a Replica Management System

Utilization of network topology information with replica access and data insertion is an important technique. When reading from extended data repositories, simple strategies may be used that benefit from the user-defined storage strategy to obtain access to nearby replicas. In a scientific setting, client software is designed to enable researchers to benefit from fast access to large data sets using a variety of computation tools.

Data access methods may be classed as *explicit* or *implicit* techniques: explicit techniques specify that data is to be moved; implicit techniques reduce the method to one similar to a local data access method and leave the data movement to an underlying system. Explicit methods are exemplified by FTP and SCP, while implicit methods are exemplified by remote filesystems such as NFS or AFS. Additionally, virtual remote filesystems may be exposed to user software using adapters such as Parrot or UFO, allowing implicit access to otherwise explicit services.

Both access methods are expected in extended repositories and may lead to performance improvements because of the available locality and parallelism of data sources. Added complexity and flexibility result when combining traditional local workstation computation with distributed cluster or grid compute engines. Consider the differences in complexity between two potential access patterns: a file retrieval and graphical display on a researcher's local workstation or a batch of hundreds of parallel computation jobs that analyze hundreds of existing simulations to parameterize and instantiate additional simulations. In both cases, the system must be able to provide access to a nearby copy of a data file for read access.

However, in the distributed case, additional optimizations are possible if the data tools are integrated with the computation tools. First, when submitting the job, the abstract name may be used to direct the job submission system to prefer certain computation sites that are more likely to obtain good data access performance. This may be done by inspecting the replica locations that correspond to the abstract record definition upon translation to virtual data names. Next, upon job arrival and start, abstract names are resolved a second time, and data access is optimized with respect to the actual job location.

The implementation of multiple-name resolution provides benefits in performance and reliability. Performance is improved by a two stage process that executes jobs close to their data sources and binds file access to nearby services. Reliability is improved by allowing running jobs to determine replica existence and rebinding the file access with respect to replica availability.

C. Formulating Job Submission for Replica Access

An adapter called Parrot has been previously developed [1] to present a view of multiple remote filesystems as a single, local 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 enables 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; however, doing so increases the complexity faced by the programmer. A client job description framework 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, enabling one to perform computation in a completely application-oriented way, as shown in Figure 3.

```
> 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
```

Fig. 3. Example simulation script using abstract data locations

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 and then to submit the user task, `transmute`, to an available compute system, using the adapter to 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 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.

D. Job Submission Methods for Replica Access

In a replica location system, a service maintains a database of replica locations. This service may be queried in three ways: to map metadata tags to data set labels, to map data set labels to a set of file names, or to map a dataset 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*: to effectively perform computation utilizing replica access:

- 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.
- Scheduled Computation on Remote Data; which interfaces with an existing scheduler to create jobs that access data over a virtual filesystem.
- Remote Computation on Remote Data; which utilizes the ability to directly send jobs to remote systems for processing.
- Multiple Name Resolution; which is a more complex scheduled method, guiding the matchmaking process with respect to data locations.

1) 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 on a typical command consisting of an operation on abstract dataset 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 4 in the LCRD frame.

Optimizing this operation is 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.

2) 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 we use these locations in the commands and arguments. The GEMSrun 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 4, 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.

3) 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 (a Chirp feature). The GEMSrun client makes use of this technique by sending jobs directly to a file server for computation. The method is similar to the scheduled SCRD method, but missing two important concepts: external centralized matchmaking and scheduling.

An example of replica access is shown in Figure 4 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.

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.

- In an environment in which replica locations are free to change or fail, replica locations may not be available at job execution time.
- 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 the first property is a harder constraint than the second property, but both represent essential design issues.

A potential solution involves resolving the replica locations twice. The 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, the compute job again resolves replica locations, essentially performing an LCRD operation described above. At this point, all replica locations may change as a result of 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.

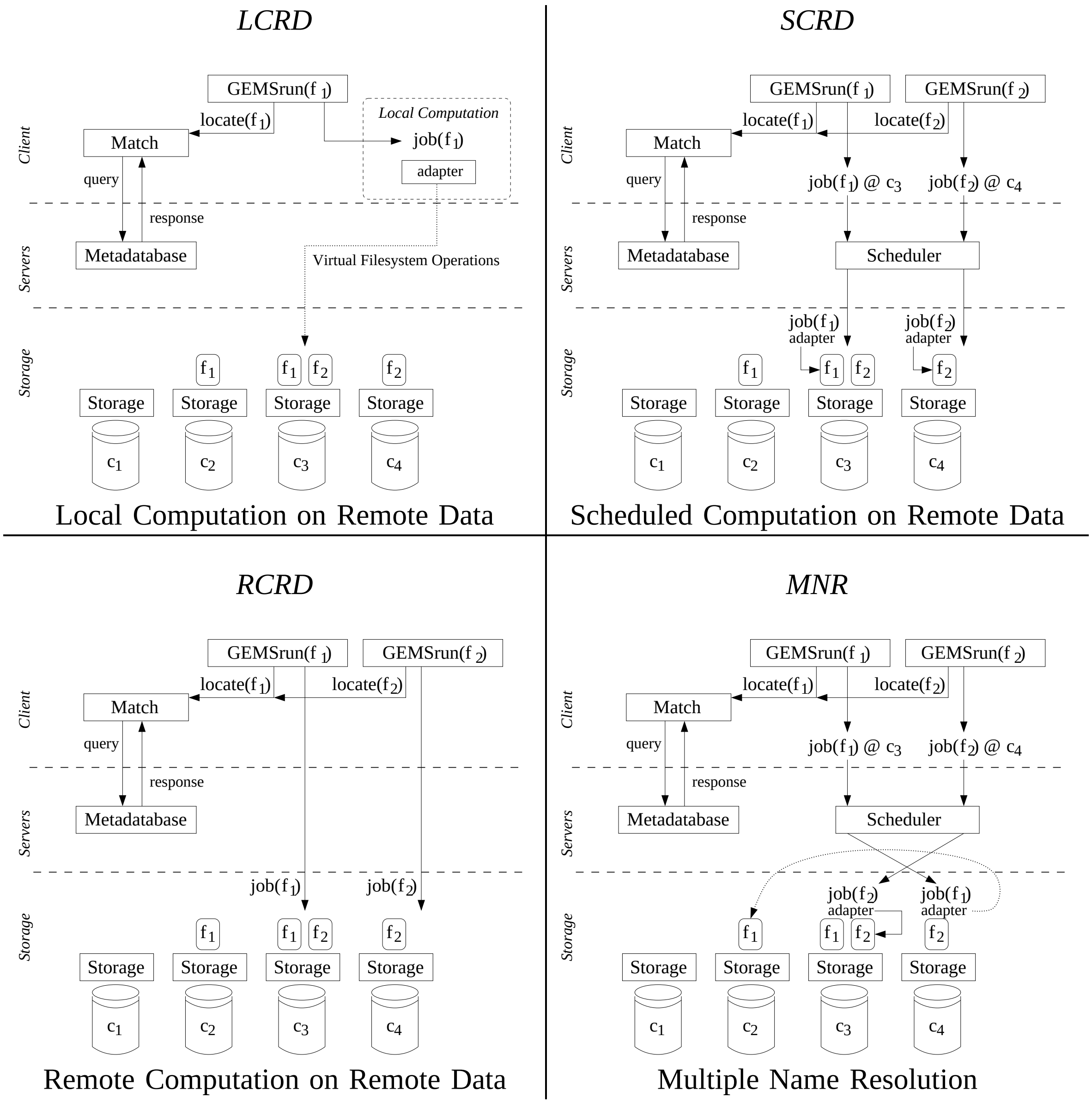


Fig. 4. Computation in a replica management system

Job Submission

The following algorithm is performed by GEMSrun.

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 4 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 $\mathrm{job}(f_2)$ on c_3 first and then places $\mathrm{job}(f_1)$ 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. Using MNR, $\mathrm{job}(f_2)$ utilizes the Late Resolution method and obtains access to the local copy of f_2. Then, $\mathrm{job}(f_1)$ 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.

E. 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. Once the job arrives, it functions as a LCRD job, that is, 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 is 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.

IV. The Repository Model

Scientific repositories create a browsable front end for user-labeled data sets stored in a scalable backend. 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 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 backend. By combining the metadatabase with the management system, a quickly deployable tuple space *and* a survivable and parallelizable data system are 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.

The browsable repository created by GEMS allows users to structure workflows around data sets stored in GEMS. This metadata may be used to derive a novel concurrent programming model that enables synchronized tuple space communication and parallel data movement, called a *data sweep*. This framework may be used to build up parameter sweeps, workflows, and complex interactive parameter space explorations.

Each entry e in the GEMS metadatabase contains a tuple of parameter tags and values, formatted as m equations: $e.\mathrm{tuple} = \{p_i = v_i, i = 1..m\}$ (see Figure 3). A query set 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." The entry e also contains a file management data structure, $e.\mathrm{files}$, which may be used to obtain file information and replica locations.

A. Workflows in a Tuple Space

Experimental scientific workloads are more often driven by adaptive parameter investigations than well-known static sequences. Popular grid programming models include the parameter sweep and the

[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.

workflow. In a *parameter sweep*, the same computation element is independently performed by 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 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, 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 the following:

- 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 the state variable x_i.

Workflows built within the GEMS framework are supported by 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:

- *Encapsulation:* Operations performed within a workflow task form a logical group.
- *Clarity:* The task dependency structure may be described by a simple graph, and the state of a running instance may be similarly diagrammed.
- *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.

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 GEMS method 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.

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.

B. Dynamic Creation of Workflow Targets

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 5a).

More complex, interactive workflow-like structures may also be simply instantiated within this model. Consider the same molecular simulation

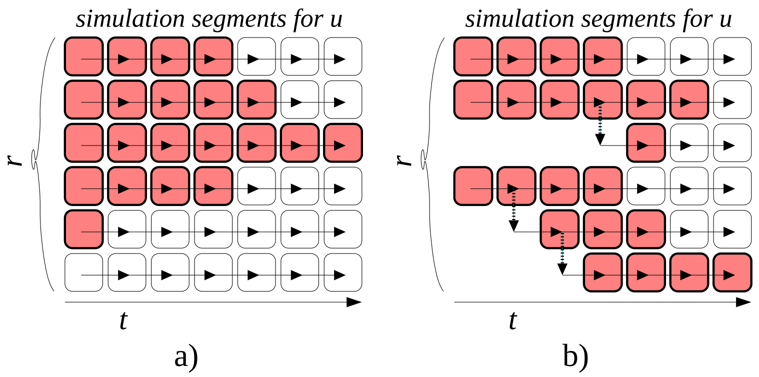


Fig. 5. (a) In a parameter sweep [6], [7], user jobs typically fill a square parameter space. (b) To allow for interactive parameter execution, the system has to allow the dynamic user creation of execution branches.

example with the addition of user steering: the user may modify the force fields applied within the experiment, thus forking a new trajectory through the parameter space as shown in Figure 5b). 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

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

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.

V. Computing in a Repository of Replicas

The data-driven grid models a compute grid as a set 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* [8]. Users of a opportunistic grid of arbitrary size may have *temporary* access to a large variety of existing storage resources, accessible over standard APIs [9], [10]. 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.

A. Data Placement

While approximately optimal solutions have been attempted for the job/data co-scheduling problem, we focus on the simple case of users attempting to enable grid computing in small collaborative projects. Our model relies on an external, flat list of available, independently administered storage sites. Users are then capable of organizing the sites into logical clusters, making up a storage map. This reusable per-record process is technically less difficult than typical job placement matchmaking. The maps result in a mentally tangible data environment in which the storage, access control, and data movement policies may be carried out.

Policy negotiation lends itself to existing work on resource matchmaking [11], however, we extend this model in the storage case by enabling users to specify cluster topology along with the eligibility criterion, called a *storage map*. This additional information deepens the semantics offered by the given policy information, by labeling the eligible sites as well as defining how to use them. Given a list of available storage sites, the user can select eligible locations and group them into clusters. In typical use cases, whole swaths of available machines at collaborating universities can be pooled together, with simple categorization.

For example, using traditional matchmaking a controller could be instructed to store two replicas on a given list of eligible sites; with the storage map, one replica would be stored at each of two clusters, and data access operations from a client at either cluster would be directed to the nearest replica. Thus the simple augmentation can be used to reduce the occurrence of wide-area data access performance penalties and increase the availability of replicas in the event of cluster outages. Other consequences of the storage map concept are discussed below.

B. Data Access

To satisfy these problems, we offer the *data sweep* as a programming model. This model allows users to operate within a *data landscape* by performing database-like operations. Application checkpoints, job dependencies, user or automatic branches, and ordinary stored data records may be encapsulated and utilized by the researcher. While the high-level approach works with abstract data records stored within the storage map, the underlying system still operates on the distributed file servers, allowing the use of existing application codes.

Each data record in the system is correlated to a storage map indicating the cluster topology used by the record. Combining the data sweep computation model with this information allows running jobs to access replicas that are *nearest* to the computation site with respect to the map. Thus, while not globally optimal, the user-level replica placement flexibility offered allows users to create locally optimal systems for their work.

Compiled user codes are unable to accommodate the variety of new and experimental APIs. Thus, 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. In these examples, we consider data access methods for scientific users launching large batches of jobs that run on a compute network in which an opportunistic storage network is embedded. Experimental cases were performed on a simple archive creation workload and in an actual molecular hyperdynamics application.

C. Archive Creation

The first experiment measured the time taken to create a simple tar file from a local data set used by a real world scientific application, the hyperdynamics experiments. The resulting archive is 3.7 GB and contains 13,934 files. The data storage sites ran Linux 2.6.9 on dual 2.4 GHz 64-bit AMD machines, all on the same institutional network. The GEMS server was located on a dual 2.8 GHz AMD system running Linux 2.4.27. The three methods were run and profiled; results are shown in Figure 6.

- *G/P*
  GEMS provides a data repository for running scripts through traditional `get` and `put` operations. 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 it also relies on significant local disk usage and data copying.
- *PIPE*
  GEMS creates data *sources* and *sinks* that may be targeted by input and output streams. For example, users may reserve larger amounts of space in the GEMS system than are 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. Again, it uses only the GEMS client tools, but it requires interprocess communication connected by the user through UNIX pipes or a similar technique.
- *VFS*
  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 operating within the replica system. While this solution offers the most functionality, it is available only on Linux.

The *G/P* method created the archive and stored it with the `GEMSput` repository insertion tool in separate steps. The *PIPE* method consisted of three steps: creating a repository reservation with

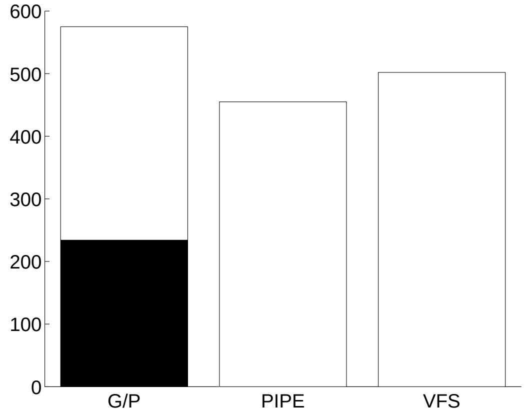


Fig. 6. Archive creation times via various methods. In the *G/P* case, the turnaround time consists of two components, local archive creation (black) and archive tranfer to the repository (white). The other methods perform both operations concurrently.

`GEMSreserve`; piping the output of `tar` into a Chirp I/O forwarding tool; and on completion, committing the repository record with `GEMScommit`. Since each GEMS method took less than 2 seconds, the time consumed is not visible. The `VFS` method created the record, using `GEMSrun` to drive `tar` execution inside a `parrot` environment, and managed the record construction internally.

Results show that streaming output methods are slightly better than the two-step method and that moving data through the pipe is slightly faster than moving data through the virtual filesystem.

D. Hyperdynamics

The next experiment measured the overhead of the GEMS model on a complex computation, molecular dynamics (MD). MD simulation [12] is a powerful and widely used tool to study molecular motion. Insight into chemical properties of a molecule may be obtained by observing rare conformational transitions from one metastable region of the simulated potential energy surface into another. Typical systems stay in one metastable state for a long time before making a transition into another metastable state. Thus considerable computational resources must be allocated to achieve the simulated timescale necessary to reach the transitions. Recently, many approximate methods have been developed to extend the timescale of molecular simulations. These methods include transition path sampling, the kinetic Monte Carlo method, the finite-temperature string method, and the hyperdynamics method [13].

TABLE I
MD ENERGY SURFACE EXPLORATION APPROACHES

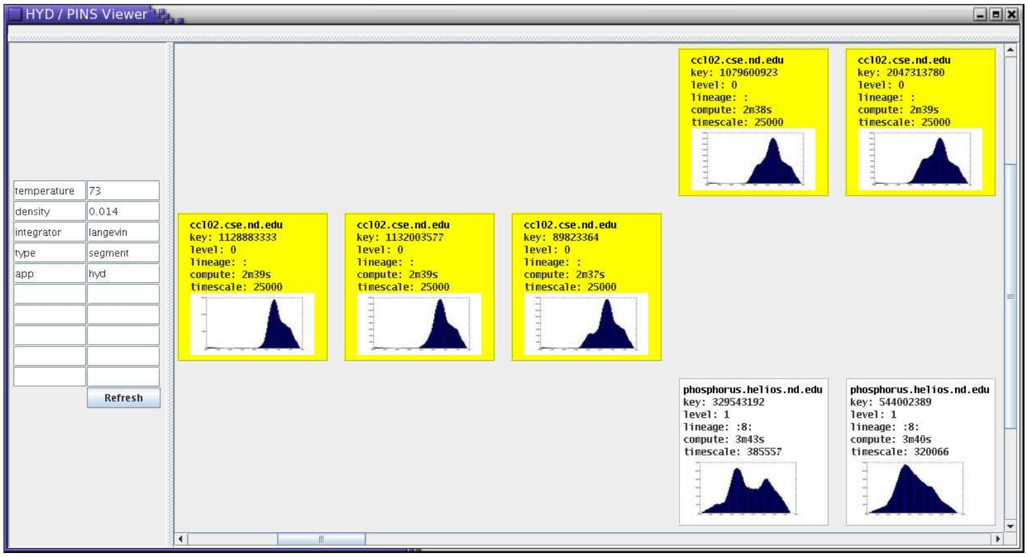
Name	Approach	Challenge
`HYD-DEPTH`	Long run	Numerical performance of scalar system
`HYD-BREADTH`	Branch search	Scalability of distributed system
`HYD-EXPLORE`	Interactive	Human interaction with distributed system

In this section, we apply GEMS to the hyperdynamics method. In hyperdynamics simulation, enhanced storage organization and rapid data access for spacious parameter sweeps are insufficient features for the effective investigation of system behavior. This steered method allows the user to bias the simulation into areas of conformational space yet to be explored.

The hyperdynamics method treated here essentially consists of a search through a parameter space of simulation parameters, including the level of hyperdynamics bias force applied. The method attempts to observe a rare long timescale event quickly by applying additional bias forces to the system. The application produces timescale and entropy histograms that indicate whether the simulation has progressed to the point at which applying another bias potential level would be beneficial and free from serious error. Since there is no analytical method to make this determination, tools to enable *ad hoc* exploration of the parameter space were required.

Our target application involved the generation of simulated molecular trajectories over a range of input parameters, including temperature (T) and density (ρ). These trajectories are independent and thus are an ideal application for parameter sweep toolkits. Figure 5a) diagrams this method by indicating simulation *segments* - restartable chunks of simulation progress - as functions of the (T, ρ) input and time, t. Each *trajectory*, shown in the figure as a row, is a sequence of segments over time, each containing the simulated molecular state over a given segment length and encapsulated in the storage system. Applications described above such as transition path sampling (TPS) have already been implemented in such a way [14].

The application described in this paper differs in that only a subset of the whole parameter space is



temperature	73
density	0.014
integrator	langevin
type	segment
app	hyd

```
cc102.cse.nd.edu
key: 1079600923
level: 0
lineage: :
compute: 2m38s
timescale: 25000

cc102.cse.nd.edu
key: 2047313780
level: 0
lineage: :
compute: 2m39s
timescale: 25000

cc102.cse.nd.edu
key: 1128883333
level: 0
lineage: :
compute: 2m39s
timescale: 25000

cc102.cse.nd.edu
key: 1132003577
level: 0
lineage: :
compute: 2m39s
timescale: 25000

cc102.cse.nd.edu
key: 89823364
level: 0
lineage: :
compute: 2m37s
timescale: 25000

phosphorus.helios.nd.edu
key: 329543192
level: 1
lineage: :8:
compute: 3m43s
timescale: 385557

phosphorus.helios.nd.edu
key: 544002389
level: 1
lineage: :8:
compute: 3m40s
timescale: 320066
```

Fig. 7. Graphical user interface representation of hyperdynamics segments. The tool automates simulation branches and 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.

explored: it is too large to fully explore, and only a part is of interest. However, the area of interest is not known in advance and must be determined by statistical analysis of previously computed segments. The interesting areas of the parameter space are entered by *branching* from the existing segments, creating a new trajectory that differs from the unmodified sequence in that a new bias potential, described above, is applied. Thus, additional metadata must be stamped on each segment to record the location of the segment in the search tree, as diagrammed in Figure 5b).

Table I provides an overview of potential approaches to this method. The first, `HYD-DEPTH`, represents the control case of using only traditional molecular dynamics. The second, `HYD-BREADTH`, automatically explores all possible applications of the method. While this method may employ massive parallelism to obtain a given result quickly, we demonstrate here that it wastes a great deal of resources. A solution is offered in the `HYD-EXPLORE` method, which allows for the controlled use of parallelism to explore promising paths as well as to selectively prune useless searches. This method relies on the technical solution of distributed computing problems such as data management and organization, user feedback and programmability, and job control. Our results demonstrate that user-steered hyperdynamics (`HYD-EXPLORE`) can improve on traditional dynamics (`HYD-DEPTH`) as long as the resource consumption of brute-force techniques (`HYD-BREADTH`) is restrained.

As an example, Figure 7 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 it pulls image files from the storage network to provide a high-level view of workflow progress.

In our present application, viewing output on the fly is not enough. The timescale and entropy histogram indicate whether the simulation has progressed to the point at which applying another bias potential level would be beneficial and free from serious error. Because 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.

As a demonstration of the nontriviality of this process, Figure 8 diagrams the timescale performance as a function of branch location on the time axis. Thus, the researcher controlling the simula-

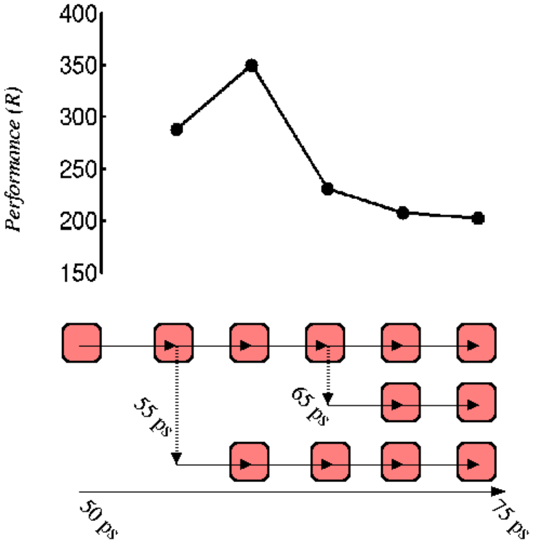


Fig. 8. Performance ratio R for various branch points. The ratio is plotted above a diagram indicating two illustrated example branches. R is plotted as a function of branch time. A higher ratio indicates more efficient exploration of the simulated conformation space. The performance ratio is obtainable only as the result of a simulation segment, necessitating interactive workflow control.

tion must monitor the output histograms for error and smoothness while selecting branch points that maximize simulation efficiency in terms of timescale. Critically, applying additional bias levels poses hazards. If the bias is too high in 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. It is not known *a priori* how long one will need to simulate in order to achieve correct distribution of S for the next bias level. Typically, one would like to run a set of independent hyperdynamics simulations, each with a different set of parameters, such as temperature, density, and bias settings.

Since the application of additional bias levels increases computation time per segment, we use a *performance ratio* (R) that indicates efficiency: work done measured by timescale per unit of CPU time, or

$$R = \frac{\texttt{timescale}\ (\texttt{simulated femtoseconds})}{\texttt{CPU time}\ (\texttt{real seconds})}. \qquad (2)$$

In this perspective, a set of normal single-processor hyperdynamics simulations is individually

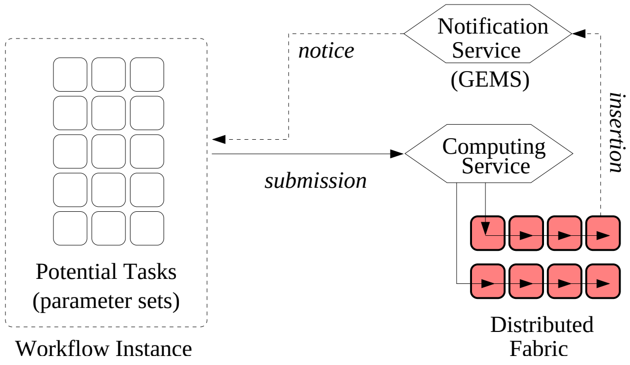


Fig. 9. Notification/submission loop for parameterized workflow instances in the GEMS framework.

run at a baseline performance level. GEMS offers the ability to easily exploit the available parallelism in the method on an opportunistic computing/storage system. Any method to parallelize the set of sequential runs will accumulate some lost communication time; in this section we consider the communication time consumed as a whole, that is, the time that would not be consumed if running all tasks on a sequential system.

The GEMS framework was used to create a *notification-based dependency loop*, enabling ordinary workflow programming on parameterized GEMS records, implemented as a data sweep over the required search targets, as diagrammed in Figure 9. Since additional search jobs are based on statistics obtained in previous jobs, a complex dependency structure is automatically built from the the user dependency rule outlined in Equation (1).

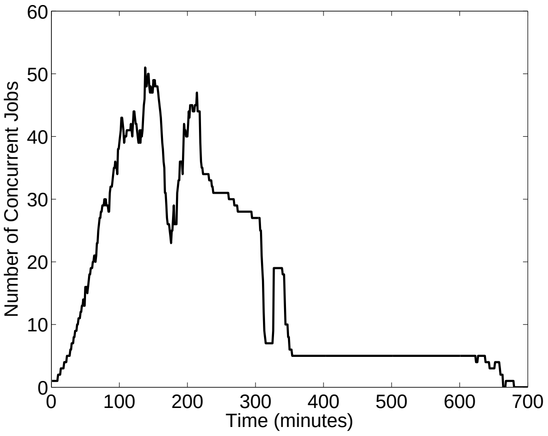


Fig. 10. Job parallelism over time in the hyperdynamics method.

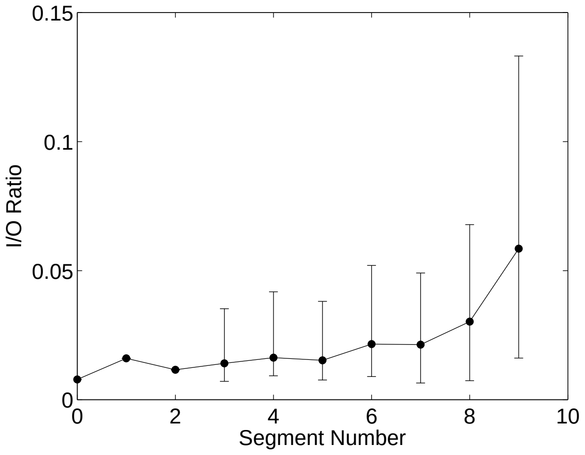


Fig. 11. I/O Ratio I (Equation 3) per segment. Averages are reported with 95% confidence intervals where appropriate.

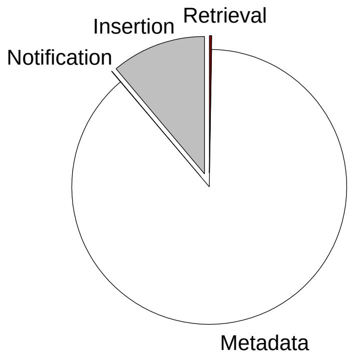


Fig. 12. Centralized time consumed on the replica management server per operation.

E. Interpretation of Hyperdynamics Results

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

$$I = \frac{\texttt{Time spent in GEMS clients}}{\texttt{Time spent in computation}}, \qquad (3)$$

where computation includes PROTOMOL and Matlab operations. This ratio is plotted *per segment* in Figure 11. 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 relative server response time as measured by the server for each operation is shown in Figure 12. Metadata operations were the most common operation during the run and were also the most expensive.

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 II.

- Employing `HYD-DEPTH` 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. Since the path length is still 17, the turnaround time is again only 1.4 hours, however.

This experiment demonstrates that while a centralized metadata system may be used 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.

TABLE II
EXPERIMENTAL RESULTS.

Name	Total (hours)	Turnaround (hours)
`HYD-DEPTH`	1.8*	1.8*
`HYD-BREADTH`	5800*	1.4*
`HYD-EXPLORE`	6.2	1.4

*Result extrapolated from experiment.

VI. Collaboration and Security

Extensible repositories combine user-defined naming schemes, storage strategies, and externally owned and operated storage services into a viable scientific resource. Each operation in the system combines metadata operations on the centralized system with data operations on distributed volunteer systems and funnels a wide, potential usage space through enabling client functionality and tools.

Simple data tagging-frameworks are doomed to eventually expire. Different communities of researchers will emphasize different tags because of the varying importance of simulation parameters, local conventions in usage patterns, or obvious application-related factors. An extensible repository will provide the minimum data-tagging tools to make collaboration possible, allowing for new user groups to import existing data sets and naming schemes into the greater system.

The ability to add and remove external storage services during normal system operation from the system is a fundamental feature. Users must be able to determine storage sites that are eligible to store their data. Consequentially, extensible repositories cannot impose data placement strategies on users.

A. *Scientific Collaboration*

Extensible repositories are motivated by the concept of collaborating scientists creating a common catalog and management system while maintaining local resource autonomy. Thus, a common ground is instantiated at the intersection of client requests and authorized system services. User and site management is a monumental problem in a large scale system, and must be localized by data set owners. The global set of users must be able to operate without global data definitions, user lists, host lists, and so forth, as these are likely to change rapidly. Administrative data structures are thus created on a per-record basis and represent a usage and management policy followed by the greater system but enforced by the underlying systems trusted by the users. A repository extended over existing resources such as volunteered desktops must be able to respect the ownership of the resource as well as the security of the data objects.

The system allows for the interaction of users in various roles, including *data consumer*, *data provider*, *storage provider*, and *replica manager*. Note that a person may take on multiple roles. Each has limited knowledge of other users and limited ability to affect the whole system.

Individual users may find it complex to manage dynamic replica locations, monitoring of user-defined network topology maps, authentication schemes, and large-scale user groups. Thus, simplified concepts are applied and a generalized authentication scheme is used to enable scientific data sharing when desirable, and tight access restriction otherwise.

B. *User Management and Access Control*

Scientific systems operators who wish to share data sets and storage resources with remote users will find that allowing authorized access and prohibiting unauthorized access is of primary importance. These problems are commonly treated as security issues. All users who attempt to access system components must be authenticated. In practice, superficial tests are performed to determine whether the user can satisfy a given test based on the subject name claimed by the connecting party. Upon satisfaction, the system determines whether the subject is eligible to perform the requisite operation by consulting an access control list (ACL).

Wide-area extended repositories are designed to support a large number of volatile user groups, data record, and storage devices. The construction and maintenance of a global static list of users and their authentication abilities are not practical. Thus the authentication process is pushed away from the centralized system and onto the intermediate storage systems with which the users interact. An authentication test may be performed on a storage site relevant and trustworthy with respect to the user and record at hand. The successful completion of the test indicates that the operation is acceptable, and resulting changes are propagated up to the centralized replica management system.

C. *Semi-autonomous Regions*

This functionality may be enabled by a careful consideration of all system stakeholders and through the use of appropriate procedures and data structures. *Data providers* may specify access control rules use a commonly used tool, the ACL, without referring to a global user list or authentication protocol. Additionally, they may specify which *storage*

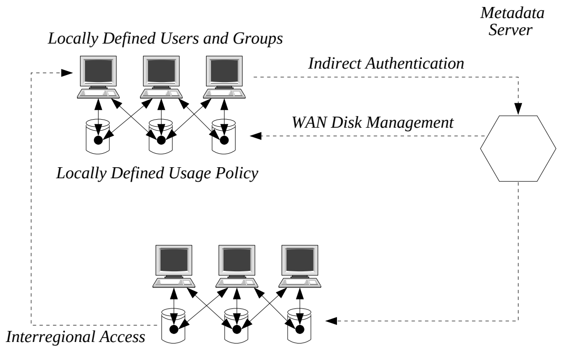


Fig. 13. User-specified cluster topology information creates semi-autonomous regions on a per record basis. Users and storage providers interact over authenticated connections. Although users have locally defined identities, changes are propagated up to the metadata level as authorized by the record owner.

providers are eligible to host their data sets. Since data placement is dynamically controlled by the *replica manager*, pattern matching-based specification structures are used to include or exclude servers and clusters for individual data records, creating a manageable per-record storage network topology.

The intersection of data and storage providers creates a conceptual *semi-autonomous region* in which data lives. These constructs are autonomous in that the allowable authentication techniques and user lists may be maintained without reference to a global center such as the replica manager or another region.

Data consumers must be able to navigate the distributed data structure created by the data and storage providers. Access to a given record requires satisfaction of the record ACL with respect to a given storage site. Thus read access is enabled in a pairwise manner, without requiring regulation by the replica manager. Users from multiple regions may interact via these direct connections by regulating access control and consulting the centralized metadata catalog, creating the possibility for efficient, direct data connections and blackboard-style data publication. This creates the potential for simplified collaboration.

D. Repository Access on the Grid

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 these capabilities 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 [15].

This solves the basic grid security problem but has certain limitations. 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.

E. Use Cases

Consider a case in which research group leaders from distant universities desire to construct a relatively secure cooperative database, building on existing (grid-enabled) resources. This problem is concerned with constructing new grids from existing grids or their fragments, ultimately creating a grid-of-grids. In the pre-grid era, they 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 extremely 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).

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 pairwise way within the greater structure. This would involve selecting certain resources for use in the derived system and ensuring that the security system works for the users and systems involved: a local procedure that should have no global side effects. This process again creates a *data landscape* in which a subgroup of users has access to a subset of possible resources *and* data records. This use case creates a record-specific grid-within-grid abstraction.

F. Implementation

The creation of a semi-autonomous region 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 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 approach 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 constitutes our solution to the grid integration problem.

G. Rendition: An Enabling Technology

The process by which a user performs authenticated operations within the config data landscape is called the *rendition protocol* [3]. 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 interact in an authenticated manner only 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 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 that 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. When the file is handed over, the controller is notified to inspect the satisfaction of the challenge, carry out the operation, and return the appropriate notice.

This protocol may be thought of as similar to other 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 heterogenous 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.

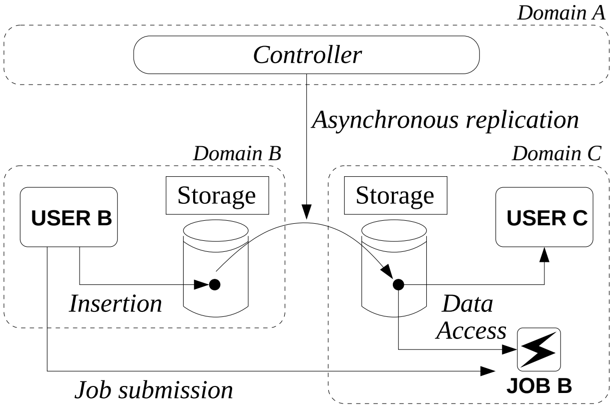


Fig. 14. Simple three-domain collaboration.

As an example, consider the simple collaboration shown in Figure 14. 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 case reinforces the notion that data replication may be viewed as an asynchronous job pre-staging process.

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 are able to delegate access to records via an ACL, and the config data structure provides a physical authentication test.

VII. Conclusion

Observations and design features in this paper are based on experiences with the GEMS system, a replica management system originally designed as a cooperative chemistry simulation repository. The GEMS model provides new practical solutions for active research problems such as autonomous grid control of disparate components, a flexible security model driven by user needs, and services for data access from a variety of possible clients.

GEMS enables the rapid construction of *ad hoc* storage grids that augment the usability of opportunistic computing systems. The workflow techniques presented here, integrated in the GEMS toolkit, improve the ability to browse intermediate simulation results and guide workflow progress.

The new contribution here is the application of GEMS concepts to general problems in distributed scientific computing, framed as grid integration and derivation problems. While a variety of tools exist to approach these problems, the ad hoc storage grid as presented here proposes a comprehensive solution packaged in a tangible, dynamic data landscape. Additionally, the collaboration concepts provide a unified methodology for previously fragmented archetypes of matchmaking and access control - providing a platform upon which solutions may be built.

Acknowledgments

This research is supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy under Contract DE-AC02-06CH11357. Work is also supported by DOE with agreement number DE-FC02-06ER25777.

References

[1] Douglas Thain, Sander Klous, Justin Wozniak, Paul Brenner, Aaron Striegel, and Jesus Izaguirre, "Separating abstractions from resources in a tactical storage system," in *Proc. Supercomputing*, 2005.

[2] Justin M. Wozniak, Paul Brenner, Douglas Thain, Aaron Striegel, and Jesus A. Izaguirre, "Applying feedback control to a replica management system," in *Proc. Southeastern Symposium on System Theory*, 2006.

[3] Justin M. Wozniak, Paul Brenner, Douglas Thain, Aaron Striegel, and Jesus A. Izaguirre, "Access control for a replica management database," in *Proc. Workshop on Storage Security and Survivability*, 2006.

[4] Justin M. Wozniak, Paul Brenner, Douglas Thain, Aaron Striegel, and Jesus A. Izaguirre, "Making the best of a bad situation: Prioritized storage management in GEMS," *Future Generation Computer Systems*, vol. 24, no. 1, 2008.

[5] Justin M. Wozniak, Paul Brenner, Douglas Thain, Aaron Striegel, and Jesus A. Izaguirre, "Generosity and gluttony in GEMS: Grid-Enabled Molecular Simulation," in *Proc. High Performance Distributed Computing*, 2005.

[6] David Abramson, Jon Giddy, and Lew Kotler, "High performance parametric modeling with Nimrod/G: Killer application for the global grid," in *Proc. International Parallel and Distributed Processing Symposium*, 2000.

[7] Paul Brenner, Justin M. Wozniak, Douglas Thain, Aaron Striegel, Jeff W. Peng, and Jesus A. Izaguirre, "Biomolecular path sampling enabled by processing in network storage," in *Proc. Workshop on High Performance Computational Biology*, 2007.

[8] Ian Foster, Jens Voeckler, Michael Wilde, and Yong Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *Proc. Scientific and Statistical Database Management*, 2002.

[9] Arcot Rajasekar, Michael Wan, Reagan Moore, George Kremenek, and Tom Guptill, “Data grids, collections and grid bricks,” in *Proc. Mass Storage Systems and Technologies*, 2003.

[10] William Allcock, John Bresnahan an Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster, “The Globus striped GridFTP framework and server,” in *Proc. Supercomputing*, 2005.

[11] Rajesh Raman, Miron Livny, and Marvin H. Solomon, “Matchmaking: Distributed resource management for high throughput computing,” in *Proc. High Performance Distributed Computing*, 1998.

[12] Tamar Schlick, *Molecular Modeling and Simulation - An Interdisciplinary Guide*, Springer-Verlag, New York, NY, 2002.

[13] X. Zhou, Y. Jiang, K. Kramer, H. Ziock, and S. Rasenmassen, “Hyperdynamics methods for entropic systems: Time-space compression and pair correlation function approximation,” *Physical Review E*, vol. 74, 2006.

[14] Paul Brenner, Justin M. Wozniak, Douglas Thain, Aaron Striegel, Jeff W. Peng, and Jesus A. Izaguirre, “Biomolecular committor probability calculation enabled by processing in network storage,” *Parallel Computing*, vol. 34, no. 11, 2008.

[15] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, and Frank Siebenlist, “X.509 proxy certificates for dynamic delegation,” in *Proc. PKI R&D Workshop*, 2004.

Notice

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (”Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.