Supporting Application-Tailored Grid File System Sessions with WSRF-Based Services

Ming Zhao Vineet Chadha Renato J. Figueiredo
Advanced Computing and Information Systems Laboratory (ACIS)
Electrical and Computer Engineering
University of Florida, Gainesville, Florida
{ming, chadha, renato}@acis.ufl.edu

Abstract

This paper presents novel service-based Grid data management middleware that leverages standards defined by WSRF specifications to create and manage dynamic Grid file system sessions. A unique aspect of the service is that the sessions it creates can be customized to address application data transfer needs. Application-tailored configurations enable selection of both performance-related features (block-based partial file transfers and/or whole-file transfers, cache parameters and consistency models) and reliability features (file system copy-on-write checkpointing to aid recovery of client-side failures; replication, autonomous failure detection and data access redirection for server-side failures). These enhancements, in addition to cross-domain user identity mapping and encrypted communication, are implemented via user level proxies managed by the service, requiring no changes to existing kernels. Sessions established using the service are mounted as distributed file systems and can be used transparently by unmodified binary applications. The paper analyzes the use of the service to support virtual machine based Grid systems and workflow execution, and also reports on the performance and reliability of service managed wide-area file system sessions with experiments based on scientific applications (NanoMOS/Matlab, CH1D, GAUSS and SPECseis).

1. Introduction

Grid systems that allow the provisioning of general-purpose computing as a utility have the potential to enable on-demand access to unprecedented computing power [14]. A key middleware functionality required from such systems is data management – how to seamlessly provide data to applications that execute in wide-area environments crossing administrative domain boundaries. This paper addresses the challenge of data provisioning through the use of a service-oriented architecture for establishing application-tailored Grid file system sessions.

The approach taken in this paper focuses on two application-centric data needs. First, application transparency is desirable to facilitate the Grid-enabling of a wide range of programs. Second, application-tailored performance and reliability enhancements are desirable because applications have diverse requirements, for example in terms of their data access patterns, acceptable caching and consistency policies, and fault tolerance requirements. These two needs are not conflicting, however, and can be addressed by building upon a virtualization layer (providing application-transparent data access [11]) and by enforcing isolation among independent virtualized sessions (allowing for per-application customization). To this end, this paper makes two contributions.

First, we describe a novel WSRF [12] based service middleware architecture that enables the provisioning of data to applications by controlling the configuration, creation and tear-down of virtualized file system data access sessions. The architecture also supports data transfers based on file uploads/downloads. A novel aspect of this approach is the flexibility it provides in controlling caching, consistency and reliability requirements tailored to application needs. The three proposed data management services (data scheduling, file system and data replication) allow Grid users and job schedulers to:

- Create, customize, monitor and destroy virtual distributed file system [11] sessions for applications with complex data access patterns. Specifically, sessions that leverage unmodified, widely available Network File System (NFS [25]) implementations can be configured by the middleware to support: cross-domain identity mapping, encrypted communication, user-level

client caching and weak consistency models; autonomous session redirection to replica servers in the event of a server failure; and checkpointing of file system modifications for consistent application restarts in the event of a client failure.
- Coordinate the movement of whole files for applications with well-defined file transfer patterns, using protocols such as GridFTP [3].

Second, this paper analyzes the performance and reliability enhancements from using this architecture through experiments with a prototype service and benchmark applications. In one experiment, a user-level weak consistency model that overlays NFS kernel clients/servers is investigated. It is shown to improve the performance of read-biased wide-area NFS sessions by speedup factors of up to 5 (CH1D coupled-hydrodynamics simulation and post-processing) and 23 (MATLAB-based NanoMOS nano-electronics simulator with network-mounted software repository).

An experiment using the GAUSS computational chemistry tool shows that user-level copy-on-write (COW), in combination with virtual machine (VM) technologies, supports consistent checkpoint and roll-back of *legacy programs* that operate on *NFS-mounted file systems*, a fault-tolerance capability unique to this approach. Another experiment shows that a running application (SPECseis96) is able to continue execution and complete successfully while a server failure is handled by the service transparently via redirection.

The rest of this paper is organized as follows. Section 2 discusses background and related work. Section 3 describes the service architecture. Sections 4 and 5 describe the application-tailored enhancements and usage examples. Section 6 presents analyses of experimental results and Section 7 concludes the paper.

2. Background and Related Work

Currently there are three main approaches to Grid data management: (a) the use of middleware to explicitly transfer files prior to (and after) application execution [6], (b) the use of application programming interfaces (APIs) that allow an application to explicitly control transfers [3], and (c) the use of mechanisms to intercept and handle data-related events (e.g. system calls [4][22][28] or distributed file system calls [11][30]) implicitly and transparently from applications.

Approach (a) is traditionally taken for applications with well-defined datasets and flows, such as uploading of standard input and downloading of standard output. Approach (b) is taken for applications where the development cost of incorporating specialized APIs is justifiable from a performance standpoint. Approach (c) is chosen when applications do not have well-defined datasets and access patterns, and when application modifications are not possible.

Experience with network-computing environments has shown that there are many applications that need solutions based on approach (c) [1][18]. In particular, distributed file system-based techniques are key to supporting applications that *must be deployed without modifications to source code, libraries or binaries*. Examples include commercial, interactive scientific and engineering tools and VM monitors that operate on large, sparse datasets [10][19][26][31].

Wide-area distributed file systems for shared Grid environments are desirable, but need to be considered in a context where modifications tailored to Grid applications are unlikely to be implemented in kernels. Nonetheless, recent work has shown the feasibility of applying user-level loop-back proxies to build wide-area file systems on top of existing O/S kernel implementations [15][24]. Examples of systems that use NFS distributed file system clients to mount Grid data are found in the middleware of PUNCH [11][18], In-VIGO [31][1], Legion [30] and Avaki's Data Grid Access Servers (DGAS) [16].

This paper builds upon related developments in NFS proxy-based Grid-wide distributed Virtual File Systems (GVFS). Previous work has investigated the core mechanisms *within* a GVFS session to support Grid-enabled data flow. This paper, in contrast, focuses on a service-oriented model to control *creation, configuration and management of customized independent data sessions.*

NeST [5] is a related storage appliance that services requests for data transfers supporting a variety of protocols, including NFS and GridFTP. However, only a restricted subset and anonymous accesses for NFS are available. Furthermore, the system does not integrate with unmodified kernel NFS clients, a key requirement for application transparency. The BAD-FS system [4] has also recognized the advantages of exposing caching, consistency and fault tolerance to middleware for application-tailored decisions. However, because it is based on libraries and interposition agents, it does not support important applications, including binaries that are not dynamically-linked or POSIX-compliant. In contrast, the techniques described in this paper enable NFS-mounted application-tailored Grid file systems.

WSRF-based Grid middleware has also been implemented in [29][8]. The system described in this paper focuses on data management and is unique in support for dynamic and customizable sessions.

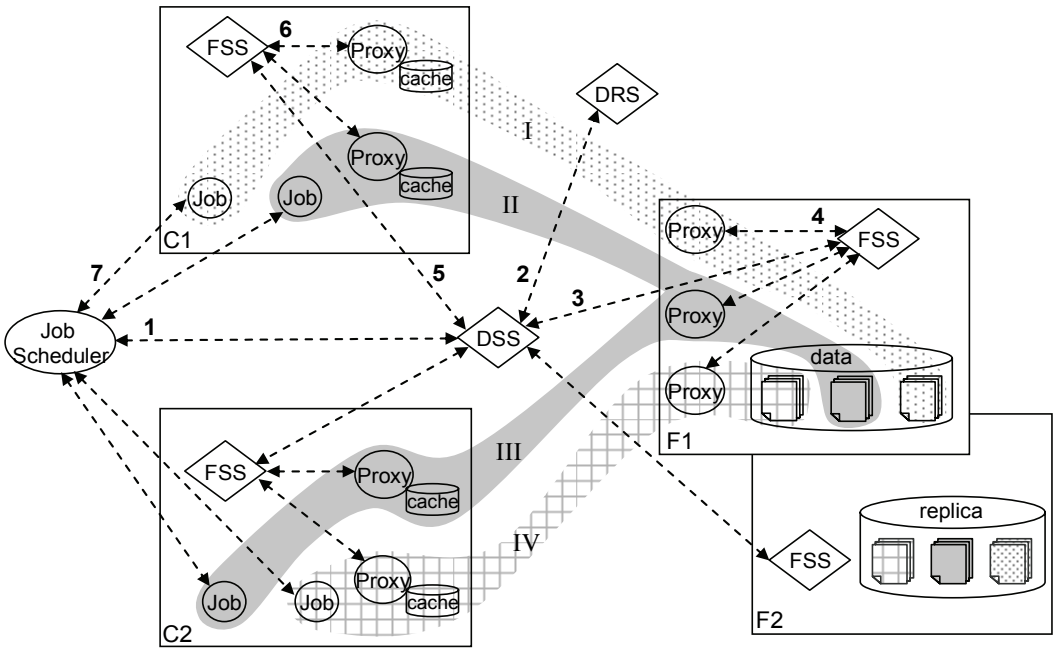

Figure 1: Example of Grid file system sessions established by the data management services on compute servers (C1, C2) and file servers (F1, F2). In step 1, the job scheduler requests the DSS (Data Scheduler Service) to start a session between C1 and F1; step 2, the DSS queries the DRS (Data Replication Service) for replica information; it then requests in step 3 the FSS (File System Service) on F1 to start the proxy server (step 4). The DSS also requests the FSS on C1 to start the proxy client and mount the file system (steps 5, 6). The job scheduler can then start a task in C1 (step 7), which will have access to data from server F1 through session I. Sessions II, III and IV are isolated from session I.

3. Services for Session Management

3.1 Overview

Figure 1 illustrates the overall architecture proposed in this paper. It supports on-demand creation of data access sessions by means of WS-Resources (the *control flow*, dashed lines), and virtualized distributed file systems [1] (the *data flow*, shaded regions). The figure shows examples of data sessions established by the data management services. Sessions are independently configured and mounted on separate directories at the client. Multiple sessions can share the same dataset (e.g., sessions II and III in Figure 1).

Fundamentally, the goal of this architecture is to enable flexible, secure resource sharing. This involves the establishment of relationships between providers and users that are complex (and often conflicting) in distributed environments. From a user's standpoint, resources should ideally be customizable to their needs, regardless of their location. From a provider's standpoint, resources should ideally be configured in a single, consistent way. Otherwise, sharing is hindered by a provider's inability to accommodate individual user needs (and associated security risks) and by the user's inability to effectively use systems over which they have limited control.

To this end, the proposed service-oriented approach builds upon two key aspects of the WS-Resource framework: *interoperability* in the definition, publishing, discovery and interactions of services [13][12][20], and *state management* for controlling data access sessions that persist throughout the execution of an application. It also builds upon a virtualized data access layer that supports user-level customization. As a result, the services are deployed once by the provider, and can then be accessed by authorized users to create and customize independent data access sessions.

The services are intended for use by both end-users and middleware brokers (e.g. job schedulers) acting on their behalf. In either case, it is assumed that the client can authenticate to the service host, directly or indirectly through delegation, leveraging authentication support at the WSRF layer, and obtain access to a local user identity on the host (e.g. via GSI-

[1] The service also supports file-based data transfers for the data flow, as described in Section 4.

based Grid-to-local account mappings, or via middleware-allocated, "logical" user accounts [17][2]).

The following techniques are used to enforce isolation among data sessions established by the service. On the server side, the kernel server "exports" one or more base directories to the service's loop-back proxies. Per-session export files are created by the service; proxies use these files to enforce that only a directory sub-tree authorized to be used for a session can be exported. The server-side proxy authenticates RPC requests based not only on RPC credentials (as conventional NFS servers do) but also by matching a 128-bit session key that is piggy-backed by the client-side proxy with an RPC payload. Finally, client/server requests are encrypted and tunneled through SSH. These techniques are in place to prevent IP spoofing and snooping of file handles. More details on session isolation techniques are presented in [9].

The prototype has been built using WSRF::Lite, a Perl-based WSRF implementation that provides transport layer security through HTTPS. Session information databases (which are maintained independently by each service) have been implemented using MySQL. The remaining of this section presents each service component in detail.

3.2 File System Service (FSS)

The File System Service runs on every compute and file server and controls the local file system proxies. It essentially implements the establishment and customization of file system sessions. The proxy processes are the resources to the service, and the service provides the interface to start, configure, monitor and kill them. Their properties are stored in files on local disk. A client-side proxy is associated with a single session; a server-side proxy, however, can be involved in more than one session (Figure 1).

The service customizes a proxy via configurations defined in a file and can signal it to dynamically reconfigure itself by reloading the file. The configuration file holds information including: disk cache parameters, cache consistency model and data replica location. They are represented as WS-Resource Property and can be viewed and modified with standard WSRF operations (getResourceProperty and setResourceProperty). When the FSS receives a request for a session's status, it signals the proxy to report the accumulated statistics (number of RPC calls, resource usage etc.) and to issue an NFS NULL call to the server to check whether the connection is alive.

3.3 Data Scheduler Service (DSS)

The Data Scheduler Service is in charge of creation and management of Grid file system sessions. These sessions are associated to the service as its WS-Resources, and their properties are stored in a database. The service supports the operations of creating, configuring, monitoring and tearing down of a session.

A request to create a session needs to specify the two endpoint locations (IP address, client mount point, server file system export path) and the desired configurations of the session (e.g. caching enabled/disabled, copy-on-write enabled/disabled, weak consistency model timeouts, as described in Section 4). The DSS firstly checks its information about other sessions to resolve sharing conflicts. For example, if the same dataset is accessed by another session with write-delay enabled at its client side, the service interacts with the corresponding FSS to force the session to write back and disable write delay.

When there is no conflict, the DSS can proceed to start the session (Figure 1). It asks the server-side FSS to start the proxy server and the client-side FSS to start the proxy client and then establishes the connection. Before sending a request to the client-side FSS, the DSS also queries the DRS (a service described below). If there are replicas for the dataset, their locations are also sent along with the request, so that in case of failure the session can be redirected to a backup server.

Note that a session is set up for a particular task. If there is an irresolvable conflict when scheduling a session (e.g. the dataset is currently under exclusive access by another session), the DSS does not establish the session and returns an error to the requestor. Cache parameters and consistency models can be reconfigured during a session. Upon such a request, the DSS also needs to resolve possible conflicts with other sessions. The DSS associates the endpoint reference (EPR) of a session with the EPRs of the proxies. When a request to monitor the session is received, the DSS asks the FSSs to monitor the proxies.

3.4 Data Replication Service (DRS)

The Data Replication Service is responsible for managing data replication. Its WS-Resources are data replicas. The service exposes interfaces for creating, destroying and querying a given dataset's replicas. The state of resources is implemented with a relational database, which facilitates the query and manipulation of information about replicas. The service can be queried with the location of a dataset (primary or backup one), and it returns the locations of the replicas.

A request to create a replica needs to specify the location of the data and the desired replica. If a replica does not already exist at the requested location, the DRS then interacts with the DSS to schedule a session between the source and the destination, and have the data replicated. Whenever a replica is created or destroyed, the DRS updates the database accordingly.

4. Application-Tailored Data Sessions

The data management services are capable of creating and managing dynamic Grid file system sessions. Unlike traditional distributed file systems which are statically set up for general-purpose data access, each Grid file system session is established for a particular task. Hence the services can apply application tailored customizations on these sessions to enhance Grid data access in the aspects of performance, consistency and fault tolerance (Figure 2). The following three subsections describe the choices that can currently be made on a per-application basis.

4.1 Grid Data Access and File Transfer

FTP-based tools can often achieve high performance for large-size file movements [3], but the application's data access pattern needs to be well defined to employ such utilities. For applications which have complex data access patterns and for those that operate on sparse datasets, the generic file system interface and partial-data transfer supported by GVFS are advantageous. Both models are supported by the data management services.

The FSS can configure data access sessions based on file system proxies. According to the information about the logical user accounts provided by the DSS, the FSS dynamically sets up cross-domain identity mappings (e.g. remote-to-local Unix IDs) on a per-session basis. The FSS can also configure the GVFS session with disk caching to exploit data locality, and SSH tunneling to provide encrypted data transfer. It is capable of dynamically reconfiguring a file system session based on changed data access patterns, for example, when a session's dataset becomes shared by multiple sessions, as discussed in the next section.

The services can also employ high-performance data transfer mechanisms (e.g. GridFTP, SFTP/GSI-OpenSSH) if it is known in advance that applications use whole-file transfers. This scenario can be dealt with in two different ways. In the conventional way, a user authenticates through the DSS, which requests the FSS to transfer files on behalf the user: downloading the required inputs and presenting them to the application before the execution; uploading the specified outputs to the server after the execution.

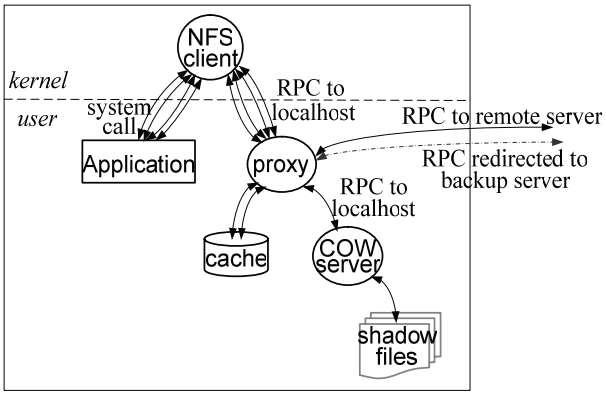

Figure 2. Application tailored customizations for a GVFS session. Read requests are satisfied from the remote server or the proxy cache. Writes are forwarded to the loopback COW server and stored in shadow files. When a request to the remote server fails it is redirected to the backup server.

The FTP-style data transfer can also be exploited by GVFS *while maintaining the generic file system interface*. The proxy client uses this functionality to fetch the entirely needed large files to a local cache, but the application still operates on the files through the kernel NFS client and the proxy client in a block-based fashion. In this way, the selection of data transfer mechanism becomes transparent to applications and can be leveraged by unmodified applications. Such an application-selective data transfer session has been shown to improve the performance of instantiating Grid VMs [31] and can also be used to support other applications through the use of DSS/FSS services.

4.2 Cache Consistency Models

Different applications can benefit from the availability of different caching policies and consistency models. The DFS and FSS services enable applications to select well-suited strong or weak consistency models by dynamically customizing file system sessions. Different cache consistency models are *overlaid* upon the native NFS client polling mechanism by the user-level proxies. For instance, an overlay invalidation polling mechanism can substantially improve performance of wide-area GVFS sessions by handling attribute revalidation requests at the client side. Other models that focus on stronger consistency guarantees rather than higher performance can also be realized in this overlay model, e.g. through the use of inter-proxy call-backs for cache invalidation.

Typical NFS clients use per-file and per-directory timers to determine when to poll a server. This can lead to unnecessary traffic if files do not change often and timers are set to too small a value on one hand, and long delays in updates if timers have large values on the other hand. Across wide-area networks, revalidation calls contribute to long application-perceived latencies. In contrast, the overlaid model customizes the invalidation frequency or disables the consistency checks on a per file system session basis.

Because the data management services dynamically establish sessions that can be independently configured, the overlaid consistency model can be selected to improve performance when it is applicable. Two examples where overlaid consistency models can improve performance are described below:

Single-client sessions: when a task is known to the scheduler to be independent (e.g. in high-throughput task farm jobs), client-side caching can be enabled for both read and write data, and write-back caching can be used to achieve the best possible performance. As writes are delayed on the client, the data may become inconsistent with the server. But from the session's point of view, its data can be guaranteed to be consistent by the DSS. Consistency actions that apply to a session are initiated through the DSS in two occasions: 1) when the task finishes and the session is to be terminated, the cached dirty data is automatically submitted to the server; 2) when the data is to be shared with other sessions, the DSS reconfigures the session by forcing it to write back cache contents and disable write-delay henceforth. In either case, the DSS waits for the write-back to complete before it starts another session on the same data.

Multiple-client, read-biased sessions: For file system sessions where exclusive write access to data is not necessary, the scheduler can apply relaxed cache consistency models on these sessions to improve performance. One approach currently implemented by GVFS proxies is based on an invalidation polling scheme. The basic idea is to have the proxy server record the file handles of potentially modified files in an invalidation buffer, and the proxy clients poll the buffer periodically. Then a proxy client can find out what files have possibly been modified by the other clients during the last period, and invalidates the cached contents of these files.

Such a model proves effective when modifications to the file system are infrequent and need to be quickly propagated to clients, for instance, in a scenario where a software repository is shared among clients. For sessions where data changes more often, the invalidation frequency can be set to a higher value; the frequency can also adaptively self-adjust in a specified range. Such polling time parameters can be customized on a per-session basis through the FSS.

4.3 Fault Tolerance

Reliable execution is crucial for many applications, especially long-running computing and simulation tasks. The data management services currently provide two techniques for improved fault tolerance: client-side COW assisted checkpointing, and server replication and session redirection.

Copy-on-write file system: The services can enable COW on a file system session, so all file system modifications produced by the client are transparently buffered in local stable storage. In such a session, the client proxy splits the data requests across two servers: reads go to the remote main server, and writes are redirected to a local COW server[2]. The approach relies on the opaque nature of NFS file handles to allow for virtual handles that are always returned to the client, but map to physical file handles at the main and COW servers. A file handle hash table stores such mappings, as well as information about client modifications made to each file handle. Files whose contents are modified by the client have "shadow" files created by the COW server in a sparse file, and block-based modifications are inserted in-place in the shadow file.

When an application is checkpointed, the FSS can request the checkpointing of all buffered modifications in the shadow file system. Then, when recovery from a client-side failure is needed, as the application is rolled back to the previous saved state, the FSS can also roll back the corresponding data state. Without the COW mechanism, when the application rolls back the modifications on the files since the last checkpointing are already reflected on the server. Thus the data state becomes inconsistent with the application state, and the recovery may not proceed correctly. For instance, files deleted on the server may be touched by the client when a checkpointed application is rolled back, causing the application to fail. A number of checkpointing techniques can be employed in this approach, including [23][7]. One particular case is checkpointing of an entire VM when the application is inside it, which is discussed in details in Section 5.1.

Server replication and session redirection: Replication is a common practice for fault tolerance. The data management services can support replication at the server-side, and transparent failure detection and recovery for GVFS sessions as follows. When the DSS

[2] Reads of file objects that have been modified by the client are routed to the COW server, instead of the main server.

requests the FSS to start a proxy client, it also asks the DRS for information about existing data replicas (the address of a replica server and the file handle of a replica) and passes it to the FSS. During the session, if the proxy client notices a RPC times out (the timeout value is adjustable at the proxy), it then decides on whether to redirect the call to the replica server.

The proxy tries at first to reestablish the connection to the server, in case the failure is caused by a transient network or server error, or a closed SSH tunnel. If it still fails, the proxy then connects to the replica server, and forwards the failed call and the following ones through the new connection. It is important to handle NFS clients that cache file handles in memory. Hence, for each redirected RPC call, the proxy client maps the old file handle inside the message to the new one[3]. Therefore the application does not even notice the failure[4], and the recovery is handled transparently.

The consistency among the replicas can be dealt with in two ways. An active-style replication scheme can be used, where each modification request on the data is processed by all the replicas. The advantage is that recovery can be very fast but it causes extra traffic and load on each server. Another scheme is to integrate the COW technique described above with the replication scheme, so no propagation of modifications is necessary, and server failure can be quickly recovered by switching to the replica server.

5. Usage Examples

5.1 VM Based Grid Computing

VMs have been demonstrated as an effective way to provide secure and flexible application execution environments in Grids [10]. The dynamic instantiation of VMs requires efficient data management: both VM state and user/application data need to be provided to the machine running a VM, and may be stored in remote data servers. Previous work has described a VMPlant Grid service to support the selection, cloning and instantiation of VMs [21]. The data management services provide functionality that complements VMPlant to support VM-based Grid systems.

In this model, the VMPlant service is in charge of managing and instantiating VMs, including the VMs used for computing (execution of applications), and data (storage of application and user data). To instantiate a compute VM, the VMPlant service requests the DSS to schedule a GVFS session between the VM state server and the VM host, and the VM state can be transferred in the way discussed in [31]. After the VM is instantiated, the VMPlant service requests the DSS to schedule another session between the compute VM and the data VM, for access to the application and user data. Then the application can be started inside the compute VM.

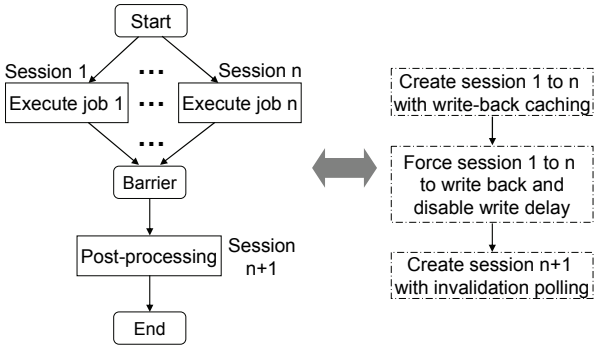

Figure 3. A Monte-Carlo workflow and the corresponding data flow supported by the data management services.

The DRS allows for replication of data VMs for improved reliability. VM instances can be checkpointed/resumed using the techniques available in existing VM monitors (e.g. VMware suspend/resume, scrapbook UML, Xen 2.0). With COW enabled in the GVFS session, buffered data modifications introduced by the application are also checkpointed as part of the VM's saved state. Upon failure of the compute VM, a session can be resumed from the last checkpoint to a consistent state with respect to the data server.

5.2 Workflow Execution

A workflow typically consists of a series of phases, where in each phase a job is executed using inputs that may be data-dependent on other phases. Workflow data requirements can be managed by the DSS with a file system session per phase, and each session can be tailored to suit the corresponding job. Furthermore, the control over enabling and disabling the consistency models and synchronizing client/server write-back copies is available via the service interface. Hence scheduling middleware can select and steer consistency models during the lifetime of a session.

For instance, a typical workflow in Monte-Carlo simulations consists of executing large numbers of independent jobs. Outputs are then post-processed to provide summary statistics. This two-phase workflow's execution can be supported by the data management services with a data flow (Figure 3) such

[3] Proxy has a file handle to path mapping on stable storage. An old file handle is mapped to the new one by the proxy parsing the path with LOOKUP calls to the replica server.

[4] The session is hard-mounted.

Table 1. Experimental Setup

	VM	VM Configuration	Host Configuration	Network Between the VMs
1	Compute VM	256MB memory, 4GB disk, Linux RedHat 7.3	Dual-2GHz Xeon processors, 1.5 GB memory	WAN between NWU and UFL, VNET[27] used between the VMs
	Data VM		Dual-2.4GHz Xeon processors, 1.5 GB memory	
2	Compute VM	256MB memory, 4GB disk, Linux RedHat 7.3	Dual-3.2GHz Xeon processors, 2.5 GB memory	WAN between LSU and UFL, SSH tunneling used
	Data VM			
3	Compute VM	256MB memory, 3GB disk, Linux Debian 3.1		
	Data VM			

that (1) a session is created for each independent simulation job with an individual cache for read/write data, (2) each session is forced to write back and then disable write delay as the simulation jobs complete, and (3) a new session with invalidation polling consistency is created for running the post-processing jobs that consume the data produced in step (1).

Such a workflow can be supported by the In-VIGO system [1], where a configuration file is provided by the installer to specify the data requirement and preferred consistency model for each phase. When it is requested by a user via the In-VIGO portal, the virtual application manager interacts with the resource manager to allocate the necessary resources, interacts with the data management services to prepare the required file system session, and then submits and monitors the execution, for each phase of the workflow.

6. Evaluation

The service-managed Grid file system sessions have been investigated with experiments based on the execution of applications in Grid VMs. The VMs are based on VMware GSX 2.5; detailed configurations are shown in Table 1. The "Compute VM" is the data client, and the "Data VM" is the file system server. Wide-area setups between University of Florida (UFL) and both Northwestern University (NWU) and Louisiana State University (LSU) are considered.

The choice of VM-based environments is motivated by two factors. First, experiments in wide- and local-area networks with consistent execution environments can be easily set up by transferring VMs. Second, file system checkpointing is a powerful complement to a VM monitor's native checkpointing capability.

6.1 Overlay Weak Cache Consistency

Two benchmarks are considered in this experiment with the VMs described in Setup 1 of Table 1. The NanoMOS benchmark models the usage of a shared software repository. It runs the parallel version of NanoMOS, a 2-D simulator for n-MOSFET transistors. The execution requires MATLAB, including the MPI toolbox (MPITB), which is read-shared among WAN users and also maintained by a LAN user, the administrator. A WAN user runs NanoMOS for 8 iterations, while between the 4th and 5th run the administrator performs an update in the repository.

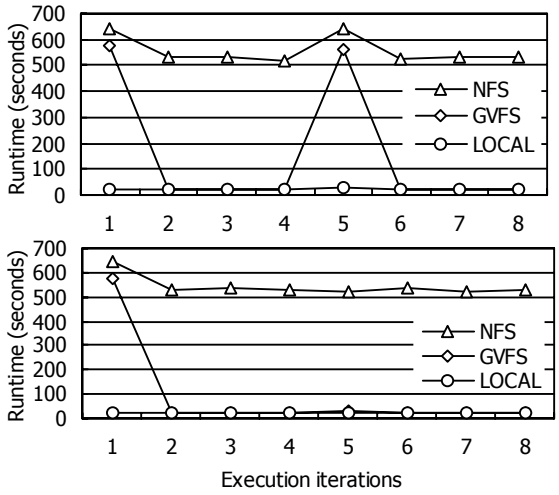

Figure 4. NanoMOS benchmark runtimes of 8 iterations performed across WAN via native NFS, and GVFS with 30 seconds invalidation period, and on local disk. Between the 4th and 5th run another user updates the software, where in (a) (top graph) the entire MATLAB is updated, and in (b) (bottom graph) only the MPITB is updated.

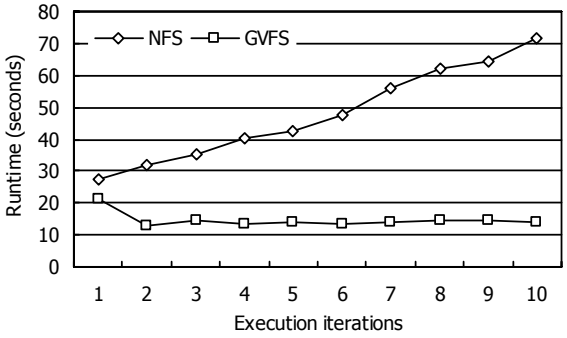

Figure 5. CH1D benchmark runtimes for 10 iterations on the input data accessed across WAN via native NFS, and GVFS with 30s invalidation period. Each run has a new data directory generated on the data VM and consumed by the post-processing program on the compute VM.

Two situations are considered: a major update, where the entire MATLAB is updated, and a minor update, where only the MPITB is updated.

Figure 4 shows the runtimes of the benchmark when the repository is mounted from the data VM via NFS/GVFS, or stored on local disk. With the relaxed cache consistency model the GVFS session achieves 23-fold speedup when its caches are warm, compared to native NFS. When updates happen, performance is affected depending on the amount of necessary invalidations: in (a), the invalidations triggered by the major update almost completely flush the cache, so iteration 5 only performs 3% better than iteration 1. In (b), the iteration after the minor update is still 14-fold faster than native NFS. In the common case (in the absence of updates), the performance of conventional NFS over the WAN is very poor, while the performance of the GVFS session with weak consistency is very close to local-disk performance.

Another benchmark used is based on CH1D, a hydrodynamics modeling application. It models a scenario where real-time data are generated on coastal observation sites and processed on off-site computing centers. CH1D outputs data into a sequence of directories on the data VM, which become the inputs to a post-processing program executed on the compute VM. The program runs 10 iterations, where in each run a new data directory is generated and then consumed by the post-processing program. The experiment results are shown in Figure 5. It is evident that as the input dataset grows the penalty caused by consistency checks also grows almost linearly in native NFS, but it remains practically constant in GVFS. The 10th run of GVFS is already 5 times faster than native NFS.

6.2 File System Checkpointing/Recovery

This experiment models a scenario where a VM running an arbitrary application is checkpointed, continues to execute, and later fails. Before failing, the application changes the state of the file server irreversibly – e.g. by deleting temporary files. This case is tested with the Gaussian computational chemistry application running on the compute VM and data mounted from the data VM (Setup 2 in Table 1). The experiments show that, in native NFS, when the compute VM is resumed to its previous checkpointed state, the NFS reports a stale file handle error and the application aborts. In contrast, with the application-tailored checkpointing GVFS session, the application has been recovered successfully after the VM is resumed from the same checkpoint.

Although it is arguable that for this particular example, saving the temporary files on the compute VM's local disk instead of on GVFS can also include a consistent data state in the checkpointed VM, it is difficult for applications whose temporary data generation pattern is not explicitly available or controllable. The COW assisted checkpointing is important because it can be applied to provide failover from client failure for a more general scenario. In fact, in combination with a VM, it supports checkpointing of *legacy programs using data from NFS-mounted file systems*, a capability unique to this approach.

6.3 Error Detection and Data Redirection

In this section, the application of the FSS-based error detection and data redirection is evaluated with a data session established for the SPECseis96 benchmark application. During its execution, a failure is injected by powering off the data VM. The failure is detected when a RPC call times out, and is immediately recovered by establishing a new connection to the replica VM and redirecting the calls.

The experiment is conducted with the VMs described in Setup 3 (Table 1). The benchmark finishes successfully, without being aware of the server failure and recovery during its execution. The elapsed time of such a run (268 seconds) is compared with the execution time of the benchmark in a normal GVFS session (without injected failure, 258 seconds), and the results show that the overhead of the error detection and the redirection setup is 5 seconds (plus the timeout value - 5 seconds, specified on the proxy). Considering a long-running application, the overhead is negligible.

7. Conclusions and Future Work

Application-transparent data management and the capability of improving upon a native distributed file system at the user level are key to supporting a variety of applications in Grid environments. Previous work has shown that virtualization techniques provide a framework for establishing isolated data access sessions dynamically. This paper shows that a WSRF-oriented architecture can be used to provide an interoperable interface for managing such sessions, while supporting configuration of data access/transfer styles, caching and consistency, checkpointing and replication based on application requirements. Results show that performance enhancements due to user-level caching and consistency policies, and reliability enhancements due to file system checkpointing and redirection are enabled by the service.

The current service framework can collect application profiling information such as NFS RPC call statistics. Future work will further leverage this

information to help optimize Grid data sessions with application-tailored consistency models, replication management and load balancing schemes.

Acknowledgements

Effort sponsored by the National Science Foundation under grants EIA-0224442, ACI-0219925, EEC-0228390 and NSF Middleware Initiative (NMI) grants ANI-0301108 and SCI-0438246. The authors also acknowledge a gift from VMware Inc., SUR grants from IBM, and resources available from the SCOOP prototype Grid. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, IBM, or VMware. The authors would like to thank Peter Dinda at Northwestern University for providing access to resources, and Justin Davis and Peter Sheng for providing access to resources and applications.

References

[1] S. Adabala et al., "From Virtualized Resources to Virtual Computing Grids: The In-VIGO System", *Future Generation Computing Systems*, special issue on Complex Problem-Solving Environments for Grid Computing, Vol 21 No. 6 (April 2005).

[2] S. Adabala et al., "Single Sign-On in In-VIGO: Role-based Access via Delegation Mechanisms Using Short-lived User Identities", In *Proc. of 18th IPDPS*, 2004.

[3] B. Allcock et al., "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing", *IEEE Mass Storage Conf.*, 2001.

[4] J. Bent et al., "Explicit Control in a Batch-Aware Distributed File System", In *Proc. of the First USENIX Symposium on Network Systems Design and Implementation (NSDI)*, pp365-378, 2004.

[5] J. Bent et al., "Flexibility, Manageability, and Performance in a Grid Storage Appliance", In *Proc. of HPDC-11*, Edinburgh, Scotland, July 2002.

[6] J. Bester et al., "GASS: A Data Movement and Access Service for Wide Area Computing Systems", In *Proc. of 6th IOPADS*, Atlanta, GA, May 1999.

[7] M. Bozyigit and M. Wasiq, "User-Level Process Checkpoint and Restore for Migration", *Operating Systems Review*, 35(2):86-95, 2001.

[8] P. V. Coveney et al., "Introducing WEDS: a WSRF-based Environment for Distributed Simulation", *UK e-Science Technical Report*, number UKeS-2004-07.

[9] R. Figueiredo, "VP/GFS: An Architecture for Virtual Private Grid File Systems", In *Technical Report TR-ACIS-03-001*, ACIS, ECE, Univ. of Florida, 05/2003.

[10] R. Figueiredo, P. Dinda, J. Fortes, "A Case for Grid Computing on Virtual Machines", In *Proc. of 23rd IEEE Intl. Conf. on Distributed Computing Systems*, 2003.

[11] R. Figueiredo, N. Kapadia and J. Fortes, "The PUNCH Virtual File System: Seamless Access to Decentralized Storage Services in a Computational Grid", In *Proc. of HPDC-10*, San Francisco, CA, August 2001.

[12] I. Foster (ed) et al., "Modeling Stateful Resources using Web Services", White paper, March 5, 2004.

[13] I. Foster et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", OGSI WG, GGF, June 22, 2002.

[14] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Intl. J. Supercomputer Applications*, 15(3), 2001.

[15] K. Fu, M. Kaashoek, D. Mazières, "Fast and Secure Distributed Read-Only File System", *ACM Transactions on Computer Systems (TOCS)*, 20(1), pp 1-24, Feb. 2002.

[16] A. S. Grimshaw, M. Herrick, M, A. Natrajan, "Avaki Data Grid", In *Grid Computing: A Practical Guide To Technology And Applications*, Ahmar Abbas, editor.

[17] N. H. Kapadia et al., "Enhancing the Scalability and Usability of Computational Grids via Logical User Accounts and Virtual File Systems", In *Proc. of IEEE Heterogeneous Computing Workshop (HCW)*, 2001.

[18] N. Kapadia, J. Fortes, "PUNCH: An Architecture for Web-Enabled Wide-Area Network-Computing", *Cluster Computing*, 2(2), 153-164 (Sept. 1999).

[19] M. Kozuch, M. Satyanarayanan, "Internet Suspend/Resume," *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, NY, 2002.

[20] H. Kreger, "Web Services Conceptual Architecture", White paper WSCA 1.0, IBM Software Group, 2001.

[21] I. Krsul et al., "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing", In *Proc. of Supercomputing*, 2004.

[22] M. Litzkow et al., "Condor: a Hunter of Idle Workstations", In *Proc. of ICDCS-8*, June 1988.

[23] M. Litzkow et al., "Checkpoint and Migration of Unix Processes in the Condor Distributed Processing System", *Technical Report 1346,* U. of Wisconsin-Madison, 1997.

[24] D. Mazières, "A toolkit for user-level file systems", In *Proc. of the 2001 USENIX Technical Conf.*, June, 2001.

[25] B. Pawlowski et al., "NFS Version 3 Design and Implementation", In *Proc. of USENIX Summer Technical Conference*, 1994.

[26] C. Sapuntzakis et al., "Virtual Appliances for Deploying and Maintaining Software", In *Proc. of the 17th Large Installation Systems Administration Conf.*, October 2003.

[27] A. Sundararaj and P. Dinda, "Towards Virtual Networks for Virtual Machine Grid Computing", *3rd USENIX Virtual Machine Research and Technology Sym.*, 2004.

[28] D. Thain et al., "The Kangaroo Approach to Data Movement on the Grid", In *Proc. of HPDC-10*, 2001.

[29] G. Wasson and M. Humphrey, "Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments", In *Proc. of 19th IPDPS*, 2005.

[30] B. White et al., "LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications", In *Proc. of Supercomputing*, 2001.

[31] M. Zhao, J. Zhang and R. J. Figueiredo, "Distributed File System Support for Virtual Machines in Grid Computing", In *Proc. of HPDC-13*, 06/2004.