RJ10368 (A0507-011) July 12, 2005
Computer Science

IBM Research Report

Glamour: A Wide-Area Filesystem Middleware Using NFSv4

Renu Tewari, Jonathan M. Haswell, Manoj P. Naik, Steven M. Parkes
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Glamour: A Wide-area Filesystem Middleware using NFSv4

Abstract

In this paper we describe the design and implementation of Glamour, a federated filesystem layer that enables clients to seamlessly navigate data that is spread across multiple heterogeneous widely distributed file servers. Glamour is not a globally distributed filesystem. Instead, it enables a set of loosely coupled file servers to behave as one. It provides the common enterprise-wide namespace and data management operations as a wide-area distributed filesystem would, while relying completely on standard off-the-shelf clients, filesystems and client-server protocols. Glamour provides data management with flexible granularity and supports transparent replication and data migration. Using our testbed, we demonstrate the various features of Glamour and report on overheads.

1 Introduction

As enterprises move toward distributed operations spread over several remote locations, multi-site collaboration and joint product development becomes increasingly common. This requires data sharing in a uniform, secure, and consistent manner across the enterprise acceptable performance. While large amounts of data can be easily shared on a local-area network (LAN) using standard file access protocols (NFS[9], CIFS[26]), these mechanisms do not scale well when extended to remote offices connected over a wide-area network). Moreover, deployment of alternate solutions such as a wide-area filesystems geared for global scalability is rarely chosen by enterprises; the cost of maintaining and operating multiple filesystems and protocols for local and wide-area access and integrating data between them is prohibitive.

The need, therefore, is not to build yet another globally distributed filesystem but to group together a set of heterogeneous, multi-vendor, independent, and distributed file servers so that they act as one. It is desirable that data remain where it is, possibly in legacy filesystems or on a variety of single server filesystems. Instead, we want clients to seamlessly navigate the data without the need for additional client-side software and configuration. Performance is not the primary concern here but improvements can be made by replicating "hot", slowly changing data. Other desirable features would be a shared uniform namespace much like AFS[13] along with support for consistent and secure access. Finally, to be commercially viable, the system should be easy to use and deploy.

In this paper we propose Glamour[1], a filesystem middleware framework, that enables clients to seamlessly navigate data that is spread across distributed heterogeneous file servers. Glamour is not a new global filesystem or a new clustered filesystem. Glamour is filesystem agnostic: while it can leverage features of a filesystem, it does not rely on them.

As a first step towards globally distributed file services, Glamour provides a common enterprise-wide namespace across a set of loosely coupled distributed file servers. In order to handle a range of filesystems, from a single server to a large clustered filesystem, Glamour supports flexible data management, both in terms of how much data can be handled as a logical unit and when and how that data can be defined. For easy administration, Glamour relies on a central server to manage namespace operations and trigger data management events. It still maintains, however, the flavor of a federation as each server can operate independently and even completely isolated as might happen in the case of a network partition. To remain easy to deploy, Glamour relies on standard protocols, clients, and filesystems.

Beyond this basic infrastructure, Glamour provides data management services that include replication, non-disruptive migration and persistent server-side caching of data. Additionally, we envision that once this mobility of data is established, Glamour can go a step further to determine where and when data needs to be placed in relation to server and network conditions. This will further work to reduce the overall cost of ownership of the server infrastructure, as per-machine utilization will rise and the cost of administration and tuning will fall.

We have implemented Glamour as a federated file system layer that is built on top of multiple independent filesystems by leveraging the client redirection features of the standard NFSv4 protocol[24]. Currently, Glamour runs on Linux and AIX. Using our testbed infrastructure, we demonstrate how Glamour provides a common namespace and performs replication, load-balancing and fail-over. We detail the Andrew benchmark results and the performance overheads of client redirection both at the server and client.

In this paper we highlight three main contributions of Glamour. First, we demonstrate a commercially-viable architecture of a federated filesystem middleware layer that provides a common namespace and relies only on off-the-shelf client and protocol implementations[2]. Second, we detail the design and implementation of a virtualization layer that supports flexible units of data management on top of a standard filesystem. Finally, we demonstrate how Glamour enhances data mobility and location independence by replicating and migrating data

[1] Glamour was initially an acronym that loosely stood for replication and migration in Grids, now it is simply a word.

[2] Glamour needs the standard but not mandatory feature of client redirection functionality of NFSv4

units without disrupting client applications.

The remainder of this paper is organized as follows. In the next section we briefly summarize the features of wide-area file systems and NFSv4. An overview of Glamour and the primary abstractions is given in Section 3. Glamour fileset operations are described in Section 4. Section 5 discusses data management and fileset mottion. An overview of the archicture of the Glamour implemention is given in Section 6. We evaluate our implementation prototype in Section 7, related work in Section 8, and finally summarize conclusions in Section 9.

2 Wide Area Filesystems and NFSv4

Data and file sharing has long been achieved through traditional file transfer mechanisms such as FTP and distributed file sharing protocols like NFS and CIFS. While the former are mostly ad-hoc, the latter tend to be "chatty" having been designed for LAN environments where clients and servers are located in close proximity. Data sharing can also be facilitated by a clustered filesystem such as GPFS[23], SANFS[16] and Lustre[20]. While these are designed for high performance and strong consistency, they are neither cheap[3] nor easy to deploy and administer. Other filesystem architectures such as AFS and DCE/DFS[15] have attempted to solve the WAN file sharing problem through a distributed architecture that provides a shared namespace by uniting disparate file servers at remote locations into a single logical filesystem. However, these technologies incur substantial deployment expense and have not been widely adopted for enterprise-wide file sharing.

Recently, a new market has emerged to primarily serve the file access requirements of enterprises where knowledge workers are expected to interact across a number of locations. Wide Area File Services (WAFS) is fast gaining momentum and recognition with leading storage and networking vendors integrating WAFS solutions into new product offerings[6][1].

Introduced in 2000, version 4 of the Network File System (NFS)[24] is a distributed file system similar in design to previous versions with additional support for high performance data sharing over a WAN with integrity and security enhancements. Unlike earlier versions, NFSv4 presents a single seamless view of all exported filesystems to a client. A client can traverse the server namespace without regard for the structure of the filesystems on the server. When a server chooses to export a disconnected portion of its name space, it creates a pseudo-filesystem to bridge the unexported portions allowing a client to reach the export points from a single common root. To improve availability, NFSv4 has added features to support filesystem migration and replication. When a filesystem is migrated to a new server, clients are notified of the change by means of a special error code and informed of new locations through a special attribute. It may then access the filesystem on the new server transparently to applications running on the client. This special attribute, `fs_locations`, may also designate alternate locations for a filesystem. If a client finds a filesystem unresponsive or poorly performing, it may choose to access the same data from another location. NFSv4 also introduces *volatile filehandles*[4] which allows a server to expire client state on special events. One such event is migration, where the client will relookup open files using saved pathname components on the new server.

[3] Some may be free but are still expensive to maintain.

3 Glamour

Glamour is a package for providing wide area filesystem federation. The goal of this system is to provide distributed file access across Internet-scale networks, networks that exhibit limited bandwidth, high latency, and low reliability. Glamour provides these services using a unified administration and security model. All administration can be done from a single interface regardless of the scale of the implementation.

In addition to these common features, Glamour includes features found in earlier distributed filesystems such as AFS and DFS. Primarily, these features revolve around the concepts of filesets to facilitate data management and a unified namespace to ease client access.

The development of Glamour has been targeted towards implementation in an NFSv4 environment. All features of Glamour are delivered to clients via this standard protocol with no client side changes. Aspects of the standard NFSv4 protocol such as volatile filehandles, `fs_locations` attributes, and special protocol return codes are sufficient and used to implement all Glamour functionality.

Standard server to server protocols for this environment do not exist so protocols have been developed for data transfer within Glamour. These protocols leverage existing standards such as LDAP and could be the target of standardization in the future. All protocols, including client-to-server and server-to-server are architecture-independent and have been implemented in Linux-on-Intel and AIX-on-Power environments.

While currently implemented in an NFSv4 context, extensions to Glamour to CIFS environments incorporating Microsoft DFS has begun.

3.1 Filesets

Traditionally, storage management in Unix and Windows is performed at the filesystem level. Filesystems are usually tied one-to-one with partitions (representing either physical or logical volumes), managing all the storage space in that partition. With larger and more complex block storage systems, this management can be course grained and of limited flexibility.

Filesets have been used in the context of distributed filesystems to sidestep these limitations. A fileset can be viewed as a storage abstraction somewhere between a filesystem and a directory. Like a filesystem, a fileset "owns" its contents and

[4] In NFS versions 2 and 3, filehandles were persistent. A client could rely on a filehandle always referring to the same file; if the underlying file object was deleted and replaced by another of the same name, the client was notified of the change through invalidation of any existing filedhandles.

therefore copying a fileset implicitly copies its contents. However, unlike a filesystem, a fileset does not "own" free space. Filesets allocate and free space for files and directories from one or more backing filesystems.

A fileset is a lighter weight object than a filesystem. It is not tied to a particular OS device and can be moved from one filesystem to another relatively easily. Whereas a server will generally have a small number of filesystems, the number of filesets a single filesystem can hold is limited only by the size of the filesystem and the sizes of the filesets themselves. A fileset can be as small as a single empty directory or as large as an entire filesystem. Generally filesets are created to represent a semantic storage relationship. For example, every user could have their home directory in a unique fileset, regardless of size. Because filesets have low overhead, there is little penalty for creating a large number of filesets. Filesets can also be easily created, destroyed, and moved. The analogous filesystem operations are heavy weight. Some operations on filesets, for example, promoting an existing normal directory to a new fileset, have no filesystem analog.

3.1.1 Locations

Filesets in Glamour are actually implemented as two related objects: a fileset object and a fileset location object (see Figure 1). The data for a fileset is actually stored in one or more locations. In the simplest case, Fileset A in the figure, only a single location exists for a fileset and thus the distinction between a fileset and its location is blurred. However, filesets can have multiple (identical) locations as Fileset B does in the figure. In this case, all locations are identical and can be used interchangeably.

This model of filesets and locations is similar to the model of filesets and volumes in AFS[13] and filesets and sites in DFS[15].

Locations are usually spread across multiple servers. Having locations on multiple servers opens up new opportunities for added value;

- *Higher Throughput*: By adding servers, client load can be spread across more compute and storage resources
- *Reduced Latency*: If locations are placed on geographically distributed servers, similarly distributed clients will observe improved latency
- *Higher Reliability*: With multiple servers and automatic failover, tolerance of machine crashes and regional disasters is achieved.

As mentioned above, all locations for a given fileset are identical. In some cases, Glamour can guarantee this. For example, in the case of a read-only replica fileset (see below), the read-only nature of the fileset enables Glamour to guarantee all locations of the fileset are identical. In the case of read-write filesets, however, Glamour relies on external mechanisms, such as a clustered filesystem, to ensure that all locations are consistent.

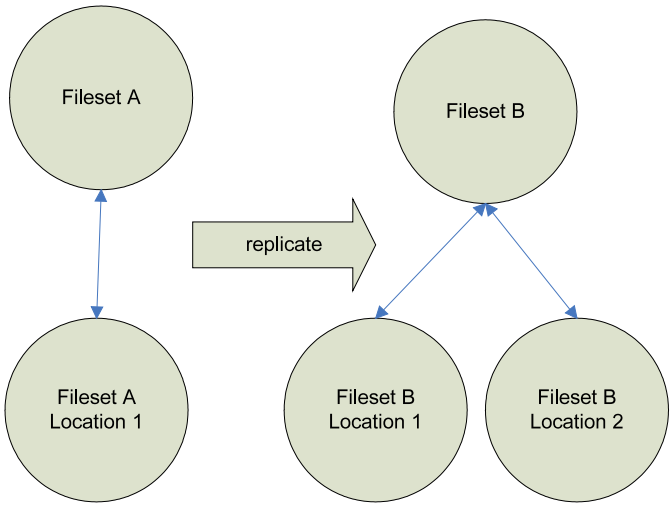

Figure 1: Glamour Filesets and Locations

3.1.2 Fileset Types

Glamour maintains two kinds of filesets: read-write filesets and read-only replica filesets.

Read-write filesets are the most common. They generally have only a single location (unless created in a clustered filesystem environment) and can be read and written by multiple clients just like a normal filesystem.

Glamour also has the concept of a read-only replica of a fileset. Read-only replicas are filesets that are created through replication of another fileset at a particular point in time. Referring back to Figure 1, Fileset B is created as a replica of Fileset A. The contents of Fileset B will reflect the contents of Fileset A at the time the replica was created. Subsequent changes to Fileset A are not reflected in any location of Fileset B until a replica update operation is requested, either manually through Glamour administrative interface (see Section 6.3) or through Glamour-provided automation.

3.2 Mount-points and the Glamour Namespace

Like filesystems, filesets must be mounted before they are visible to clients. Also, like filesystems, filesets are mounted on an existing directory in an existing fileset. The result of mounting a fileset is the creation of a special object called a *referral*.

Each server has a special filesystem which provides the root of the filesystem namespace for local file access. Glamour provides a namespace that serves a similar function for Glamour services. Details of this namespace are provided below.

Glamour mount operations differ from filesystem mounts in two ways. First, Glamour mount operations are persistent, i.e. mounts survive a system reboot. This makes sense since Glamour is a distributed, multi-server system for which the impact of individual system failures should be minimal.

Second, Glamour filesets can be mounted more than once, in multiple places within the namespace. Because Glamour mount points are very lightweight, there is no penalty in al-

lowing multiple mounts.

Note that filesets are mounted, not locations. It is the responsibility of the Glamour servers and NFS clients to determine the best location to use to access data for a fileset with multiple locations.

Glamour also introduces the concept of an external mount point. This represents the mounting of a non-Glamour NFSv4 filesystem within a fileset. Glamour provides the services necessary to create and recognize the mount point which will cause the client to attempt to fetch the data from the external server. Glamour does not manage this external data in anyway and cannot verify the availability or validity of the target data.

3.2.1 Fileset Operations

Once created and mounted, filesets, like filesystems, appear to clients as normal directories, with some restrictions, e.g., a fileset, like a filesystem, cannot be destroyed via a `rmdir` command.

In addition to common filesystem operations, a number of operations unique to filesets are provided:

- An existing fileset can have an additional location created through a *place* operation (with the restrictions mentioned above regarding read-write filesets and cluster filesystems). The place operation indicates to the server where the new location should be created. Multiple locations for the same fileset can be placed on the same server, which can, in some cases, improve performance.
- An existing directory with existing content, either within an existing fileset or not, can turned into a fileset via a *promote* operation. Similarly, a fileset location can be *demote*d back to a normal directory.
- Locations of filesets can be *migrate*d from one server to another. This is similar to creating a new location and removing and old but is done in a way that no client disruption occurs.
- A fileset can be *snapshot*, i.e., a consistent copy made at a point in time. Glamour facilitates this but requires support from underlying filesystems. Otherwise all Glamour copies are not made consistently, i.e., changes can occur during the time it takes Glamour services to walk the contents of a fileset.

3.3 Glamour namespace

Glamour facilitates fileset organization by providing a common root namespace that all clients see. The namespace contains no data. It only serves as a place to mount other user-created filesets. In this way, it functions analogously to the NFSv4 psueo-filesystem. However, where the NFSv4 psuedo-filesystem is configured independently for all NFSv4 servers, the Glamour root namespace is shared by all clients. The scope of the namespace as well as other Glamour objects is described below.

3.4 Cell

The primary organizational unit of Glamour is a *cell*. All Glamour objects such as filesets, fileset locations, and the root namespace, are associated with a cell. Cells are independent and non-interacting with each other so that a single organization can create cells in a way that best meets their business needs of security and performance. Cells can be as small as a workgroup or as large as an enterprise.

Note that the cell is a logical construct and that the Glamour architecture (see Section 6) allows multiple cells to be serviced by a single host.

In addition to maintaining all the information necessary to manage filesets, Glamour cells provide a range of other services:

- *Security*: Glamour security revolves around the cell. Cell services allow the authorization of users and groups as well as adding and removing data severs.
- *Automation*: Glamour provides automation services to facilitate maintenance of filesets. Chief among these is the scheduled update of a fileset replica from a source fileset.

Glamour cells are administered through a single administrative user interface (see Section 6.3).

3.5 Relationship to NFSv4

Glamour provides added value to an NFSv4 infrastructure without enforcing additional requirements. For example, an existing NFSv4 server could be added to a Glamour cell without having to disrupt any preexisting normal NFSv4 services that server was providing.[5]

When a NFSv4 server as added to a cell, Glamour will place a copy of the cell namespace on that server and export the namespace as "*/cellname*". An NFSv4 client need only mount the cell (or the root of the server psuedo-filesystem, if desired) in order to access any data within the Glamour cell, regardless of where the data resides.

Glamour does not provide a truly global namespace, one that crosses all organizational boundaries, e.g., a web URL. Efforts along these lines are being considered in the NFSv4 working group and elsewhere and include such techniques as using DNS records to resolve the top level, universal namespace. Glamour should fit within such a framework, managing data once an directory level suitable for a cell is traversed.

4 Implementing Fileset Operations

Glamour's data management framework aims at seamlessly replicating, migrating and navigating through filesets distributed across a federation of wide-area distributed

The fileset abstraction can be natively supported by the underlying physical filesystem (PFS). With native PFS support a fileset looks like a filesystem from the NFSv4 client and server

[5]The NFS server code itself must have been modified to include Glamour functionality as described later. No client changes are required

perspective. Except that the fileset masquerading as a filesystem can magically appear and disappear. Without PFS support, filesets can be supported by Glamour adding a virtualization layer. All we need is support in the operating system to go through this layer for fileset management and hooks in the NFSv4 server to query this layer for fileset information. Finally, we need some magic such that a vanilla NFS client sees the fileset and recognizes it as a data management unit.

In this section we will discuss the details of how this virtualization is implemented.

4.1 Implementing Filesets for NFSv4

While Glamour can define arbitrary filesets at the server, we still need a vanilla NFSv4 client to recognize fileset boundaries. As a client traverses the server namespace, it needs to detect filesystem transition (crossing mount points) for various reasons, including obtaining replica locations and other metadata information. Typically, in the absence of filesets the client detects a filesystem boundary when the value of the `fsid` attribute returned by the server changes during traversal. Each filesystem returns a unique `fsid` value which, in most cases, is based on the device major and minor numbers of the underlying device. Supporting fine-grained filesets, would then require support from the underlying filesystem to return a unique `fsid` value per fileset.

Note that from a client's perspective, fileset boundaries are solely defined by changes in `fsid` values. Essentially, all Glamour needs to do is add hooks in the NFSv4 server to query Glamour and return a different `fsid` per fileset. It would seem that a simple mapping table between the fileset boundary and a virtual `fsid` would suffice. However, every object within the fileset has to return the same `fsid` on a GETATTR request. It would be impossible for Glamour to identify the fileset for any given object unless it tracked every object in its mapping table and monitored them as they were created and deleted. One option would be to walk up the directory tree (by looking up ".."), on every GETATTR request, to check if any ancestor directory happens to be a fileset boundary. While this works for directories with significant overheads, it would still not work for files. This is because, given a filehandle in the client request, it may be impossible for the server to determine the directory containing the file especially in the presence of hard links.

In Glamour we have designed an approach that is scalable, and also requires minimal state to be maintained while adding only nominal performance overhead. The key idea is to embed the fileset information in the filehandle that is exchanged between the client and the server, instead of maintaining it for object at the server . Firstly, every fileset is assigned a unique `filesetid` per VFS. Secondly, a mapping is maintained between the `fileid` of the fileset root and the `filesetid`. Finally, an object filehandle is enhanced with the associated `filesetid` The client will later, in a PUTFH call, return the filehandle containing the embedded `filesetid`. Since a client can only access an object is by traversing the directory tree leading upto it from the server root, the `filesetid` will pass through the successive filehandles that are exchanged between the client and the server. Whenever the client steps into a directory that is the root of new fileset, the filehandle is changed to reflect the new `filesetid`. On a GETATTR `fsid` request by the client, the associated `filesetid` is used to create a unique fsid value that is returned to the client. The effect of supporting fine-grained filesets on the handling of the various NFSv4 operations is summarized below:

- *GETFH* The server embeds the object's associated `filesetid` in the filehandle that is returned to the client. The `filesetid` is either the one that was in the incoming filehandle (from the earlier PUTFH call) or a newly created one if the current object happens to be the root of a fileset.
- *PUTFH* The `filesetid` in the incoming filehandle is stored as the default value for the current object.
- *LOOKUP* The current object is checked against a mapping table to determine if it is the root of a fileset. If no entry exists, the default value that was set by the `PUTFH` operation is used. If multiple `LOOKUP`s are requested in a `COMPOUND` request, the `filesetid` flows through appropriately.
- *GETATTR* The affected attributes in a `GETATTR` request are: `ATTR_FILEHANDLE` and `ATTR_FSID`.
- *LOOKUPP* Here, we need to find the `filesetid` of the parent directory. It may be possible that the parent directory belongs to a different fileset, hence the default value cannot be used.
- *Others* `SAVEFH`, `RESTOREFH`, `PUTROOTFH`, `PUTPUBFH`, `OPEN`, `READDIR`: As with the earlier operations, the `filesetid` is appropriately handled.
- *RENAME, LINK* When a rename or hard link crosses a fileset boundary we return the `NFS4ERR_XDEV` error.

4.1.1 Fileset Promotion and Demotion

So far we have discussed how Glamour provides the necessary support to handle fine-grained filesets. The next step is to determine when a fileset can be defined. In the simplest case, filesets are carved out as an administrative task before the NFSv4 server is online (i.e., before filesystems are exported). The client, in this case, does not have any state (e.g., filehandles) associated with the objects in a fileset and can be provided the "virtual" fsid whenever it crosses a fileset boundary. The filesets in this case could be *static*, as long as the server was online. For an operational system with rare down times, it is not always prudent to take the server offline for any customer desired changes in fileset boundaries. In Glamour, therefore, we added the flexibility of *dynamic filesets* – new fileset boundaries can be established after the server is online and clients have previously accessed data. An existing directory in an exported filesystem can be "promoted" to be the root of a new fileset. The only restriction imposed is that there should be no existing hard links that cross the new fileset boundary. Note

that if a new directory is created and marked as a fileset, it is similar to a static fileset as the client has no state associated with it.

4.1.2 Filehandle Expiration

With dynamic promotion, filehandles of objects that a client has previously seen that now belong to the new fileset have to be expired by returning the `NFS4ERR_FHEXPIRED` error. For this, we need to relax the persistence attribute of filehandles. The server should indicate to the client that filehandles are volatile so that the client can prepare to maintain the necessary state required to rebuild them on expiration[6]. As filehandle expiration is generally expensive, the server needs to determine which filehandles need to be expired as we do not want to expire all filehandles. This is tricky as the fileset information is carried in the filehandle and the server cannot determine which filehandles belong to the new fileset without traversing up the tree. To manage expirations, we attach a generation number to each fileset. Whenever a new fileset is promoted, the generation number of its parent fileset is incremented. This generation number is also embedded in the filehandle along with the filesetid. Whenever the server is presented with a filehandle that has a valid `filesetid`, it checks the validity of the corresponding fileset generation. If the generation is not current, the filehandle is expired. With this the filehandle expiration is contained within the outer fileset. A similar approach is taken when a fileset is demoted. In this case, however, expiration can be limited to the demoted fileset only.

4.2 Fileset Replication

Replica filesets are generally assumed to be read-only, with only the original fileset being read-write. Any operation that would result in a modification at the replica fileset will result in `NFS4ERR_ROFS`. Replica consistency is administratively controlled, i.e., an administrator can specify when, and how often, a replica needs to updated. For example, a replica could be a daily backup of the original fileset. In this case, the replica update mechanisms could include taking a snapshot of the source fileset at midnight and updating all the replicas.

Replica updates pose an interesting problem. Although replicas are read-only, they need to be modified on an update, while clients may be accessing its files. A simple approach is to proceed with updates ignoring client accesses. While this may work in most cases, it could result in client inconsistencies for open files. For a consistent updates, snapshot support is required. A snapshot based replica update requires that the filesystem snapshot be taken from the source fileset and then populated at the replica location. The clients are then redirected to the new snapshot while preserving the filehandles. A snapshot based in-place replica update requires that the filesystem not only support fileset level snapshots but also guarantee that fileid namespace be preserved across snapshots. This will ensure that the clients see the same filehandle and attributes for a given object that it was previously accessing.

The data transfer between the source fileset and the replicas happens using an out of band server-to-server protocol. The Glamour infrastructure allows different protocols to be plugged in, the default being a differential compression scheme similar to *rsync*. This is discussed in the next section.

Replication in Glamour is useful for both load balancing and failure handling. For load balancing, the server returns different server locations (the `fs_locations` attribute in NFSv4) for the directory that is the boundary of a replica fileset. The locations returned could be based on the geographic or network location of the client, the load on the server or a combination of other factors. To enforce load balancing, Glamour can steer a client to different replica locations dynamically. The client would, in such cases, need to handle volatile filehandles and different fileids for the same objects on a different server.

Multiple replica locations are also useful to mask failures. When the client detects that the server providing a replica is unresponsive or poorly performing, it can connect to another server from the list of locations for that replica. Fail-over behaves somewhat similar to migration except that the server has failed and no state can be recovered.

4.3 Fileset Migration

A fileset, in Glamour, can be physically migrated from one server to another. NFSv4 protocol has a method of providing filesystem migration with the use of the special `fs_locations` attribute. Migration is typically used for read-write, single copy filesystems and usually employed for load balancing and resource reallocation. For the purpose of migration, a filesystem is defined as all files that share a given fsid. This allows a Glamour fileset to be physically migrated from one server to another with no noticeable impact to client applications. It is important to note that a fileset migration does not impact the namespace which is designed to be location transparent. Once a fileset is migrated, it appears as a referral at its previous location, i.e. all future accesses of the fileset on the original server will result in client redirection to the new location. Some clients that did not previously communicate with the original server, or did not have cached state pertaining to files from the migrated fileset, will encounter a pure referral when traversing the server namespace that includes the migrated fileset. Existing clients, however, potentially have outstanding state on the original server that should[7] be transferred between the participating servers. There are three design choices in migrating client state:

- *No state transfer* - the client starts afresh at the new server. If the client presents state information from the original server, it gets stale errors. In this case, the client should be prepared to recover all state as in case of server fail-

[6] A server specifies this by setting the filehandle expiration type to be `FH4_VOLATILE_ANY` and, optionally, `FH4_NOEXPIRE_WITH_OPEN`.

[7] SHOULD per [24]

ure. While this is a simple approach from a server implementor's point of view, it can be rather disruptive to client applications.

- *Complete state transfer* - the client sees a truly transparent migration. Since all client state is transferred between the servers, the client can continue to use the state assigned by the original server. In addition, clients can use persistent filehandles if the immigrating filesystems can recognize each other's filehandles. While this may be improbable to implement in a heterogeneous multi-vendor environment, it may be plausible in more tightly coupled homogeneous systems.
- *Some state transfer* - client starts afresh at the new server *except* for files that it has open[8]. All state pertaining to open files is migrated to the new server. While client expects filehandle expiration for other files, it can continue to use existing filehandles for open files on the new server[9]. This requires the server to recognize and service *foreign* filehandles specially. Other client state (including clientid and lease information) may also need to be migrated[10]

Apart from state management, migration also requires data transfer in case of single-copy read-write file systems. How the data is transferred is also open to several design choices.

- *No data transfer*: The remote server acts as a proxy reading the data on demand and in the background. Clients in this case are instantaneously redirected to the new server. All reads and writes and metadata operations happen at the new server. Apart from the performance concerns of the client having to cross two servers to get the data, there are data integrity concerns. On a network partition between the two servers neither server will have a fully consistent copy of the fileset.
- *Complete data transfer*: In this case the data transfer is complete before the client redirection happens. However, we can't delay all the updates from clients to the old server before the data transfer completes. The idea is to use a series of snapshots each with lesser data to be transferred. Finally, when the remaining updated data is sufficiently small, the clients are paused for a while until the all the data is moved over. The client redirection happens after that. Although this is a much slower approach, data integrity is maintained. In Glamour we use this approach.

Although we were tempted to explore a "transfer some data" middle ground approach we realized that it was the worse of the two. It was neither fast nor provided data integrity.

[8]This is supported by the NFSv4 protocol with filehandle expiration type of FH4_NOEXPIRE_WITH_OPEN.

[9]It is possible that a client has a lot of open files on the server especially if the server is offering delegations. It is possible to simplify the server implementation by recalling delegations before a migration event occurs.

[10]We intend to consider this approach to migration in Glamour, but it is a work in progress.

5 Data Management

Glamour has two major data manipulation components. The client-server fileset component was discussed in Section 4. The other component is the data management services component which manages server to server data operations. These services are responsible for maintaining the configuration state for all filesets within a Glamour cell and for transferring filesets between systems.

Data management encompasses management of filesets within a server, i.e., exclusive of client-server aspects covered in Section 4 and between servers.

Most fileset data management operations occur as the result of an administrative action, for example, a request to create a new fileset or to place a new copy of a replica fileset on a new server. Data management services also support the NFS server in responding to requests for `fs_locations` attribute. Since these attributes represent cell-wide information, they are the domain of the data management services.

5.1 Data Management Fileset Services

Once a fileset exists on a host operating as an NFSv4 server, the fileset services described in Section 4 are used by the server in responding to client requests. However, these services do not encompass such functions as those necessary to actually create a fileset, to migrate a fileset from one server to another, or to make create a new location for a fileset. Data management fileset services provide this functionality.

These services are implemented via a purpose-built server to server protocol. As described in Section 6, a Glamour agent is instantiated on every host running Glamour services. This agent is responsible for accepting requests from either an administrator or from other peer agents.

In addition to implementing fileset maintenance operations, the data management services are responsible for managing fileset allocation to backing filesystem storage. When a fileset location is created, it can be created at a specific location in the local filesystem hierarchy or optionally allocation can be left to data management services which maintain a list of pools of filesystems where filesets are created if not otherwise specified.

5.2 Data Management Protocols

As part of the data management fileset services, a rich set of copy and replication services is provided. These services are provided in a plug-in library that anticipates servers providing different combinations of these protocols.

All server to server communications run over ONCRPC and can be configured to use different security mechanisms such RPCSEC_GSS and AUTH_SYS depending on the environmental needs of identity verification and privacy.

5.2.1 Fileset Motion

A primary responsibility of the data management services is movement of filesets from server to server. This can happen, for example, in response to an administrative request to add a

place a new fileset location, to update existing replica locations from a source fileset, or to migrate a fileset location from one server to another.

When placing a new replica location or migrating a fileset, a copy operation takes place. Different copy implementations exist which can provide optional compression for low bandwidth lines or no compression for higher performance where network bandwidth allows.

5.2.2 Fileset Replication

One of the most common data management operations is replica fileset update. In this case, a higher performance rsync[5]-like protocol is available which only transmits fileset differences between servers. The data management protocol selection process will drop back to a simple copy where the rsync protocol is not available.

When fileset replication is to occur, it is desirable that the replication represent a point-in-time copy or snapshot. This ensures that the data in the new or update replica is consistent with the state of the source at some point in time. Without point in time copies, the contents of the resulting fileset may not represent the contents of the source at any single time but rather a mixture of the contents of the source at different times. Whether this is acceptable or not depends on the use of the contents of the fileset.

Glamour does not implement snapshot support but can utilize that support where provided by the underlying filesystem. For example, if a fileset exists within a filesystem that provides snapshot functionality, when a fileset snapshot is requested, Glamour can snapshot the entire filesystem which is sufficient (but not necessary) to snapshot the fileset location. This snapshot can then be used to populate or update a replica location. When the fileset snapshot is no longer required, it can be destroyed, at which point Glamour will release the filesystem snapshot.

Various performance trade-offs exist when using filesystem level snapshots. Generally snapshots can be quickly created with low overhead. However, there may be limitations on the number of snapshots that can exist concurrently and there may be a performance impacts on all filesystem activity while the snapshot exists. For this reason, the impacts of snapshot usage must be manually considered. In some cases, they can be forgone and the simpler inconsistent method used.

5.2.3 Location Selection

In addition to managing fileset motion, the data management services maintain and respond to requests for fileset location information. The most common source of this request is in response to an NFS client request for the fs_locations attribute. While the kernel Glamour components (see Section 6) maintain the necessary information to handle fileset requests for fileset locations on the current server, the kernel does not maintain fileset information for locations not on the local server. The data management services maintain a small, custom database of all locations for all filesets in the Glamour cell.

The data management services provides a service for requesting location data for a given fileset. The service is designed with a flexible interface for selecting among possible locations. In general, the service will not return all locations for a requested fileset since this can lead to a large amount of useless data transfer. Moreover, it is unclear how useful a long list of fileset locations is to a client. A client has severely limited information on the organization of a Glamour cell and thus is not a good position to rank the utility of different locations for a fileset. The server on the other hand has a complete list of all filesets, locations, and servers within the cell. Potentially this information can be augmented with status about server and network capacity and load, maintained via new Glamour intra-cell services. For this reason, the data management services rank the list of candidate replica locations and return an ordered list of the top n candidates, where n is configured per cell. To facilitate this, some information about the client request, for example, the client IP address, is passed to the ranking function. From the list of locations returned to the client, the client can select any in an implementation-defined manner. Some existing client implementations select the first location and only switch to another on server failures. Others use round-robin selection.

5.3 Cell namespace support

The data management services are responsible for maintaining the Glamour cell namespace in response to administrative mount and unmount requests. These services take two parts: maintaining a fileset representing the namespace and managing server exports of all filesets including the namespace fileset.

5.3.1 Namespace filesets

The Glamour cell namespace is implemented as by two normal Glamour filesets. A read-write fileset is created and maintained as described below. This fileset exists on the administration server (see Section 6). A replica of this fileset is also created and a fileset location for this replica is placed on every data server in the cell. This read-only replica thus becomes the root of the Glamour namespace. The Glamour data management services are completely responsible for modifying and updating both of these filesets in response to the various administrative operations.

The namespace fileset plays a role analogous to that of the standard NFSv4 pseudo-filesystem. Its role is only to provide a place to mount other filesets. Thus is contains mount points only and no data.

5.3.2 Export management

Glamour manages all exporting of fileset locations required for proper operation. No manual NFS exports are required.

The primary Glamour export is that of the namespace. The local location of the namespace replica fileset is exported to "/cellname" in the NFSv4 pseudo-namespace. Thus any server in a Glamour cell resolve the root of the Glamour namespace.

Each server also exports all the fileset locations it contains. The issue arises where these locations are exported to in the Glamour namespace. The locations are *not* exported directly into the namespace. For example, given a cell `mycell` and fileset mounted on /mycell/myfileset, the fileset location is *not* exported into the NFSv4 namespace at /mycell/myfileset.

This may seem counterintuitive but recall that a fileset mount operation results in a mount point be created in the appropriate fileset, in this case, at the name myfileset in the fileset representing the root namespace. This mount point triggers the NFS server to respond to the NFS client with a moved error at which point the client will request the fs_locations attribute for myfileset. Glamour then responds with locations for this fileset, including the fs_root component. Because this component is not exposed to the client, Glamour has complete freedom in where within the pseudo-namespace it exports filesets as long as the exports and the data in the locations attributed is consistent.

Glamour exports fileset locations into a hidden directory (nominally ".hidden") in the directory that implements the cell. This makes the directory available to the client but not obvious to the user.

5.4 Server Management

The data management services are responsible for managing all fileset-related state, whether it be in-kernel-memory state or filesystem resident state. In particular, the data management initializing services are used to reload kernel information after a reboot or subsystem reset and to check for fileset consistency similar to fsck.

6 Architecture

The overall Glamour architecture is shown in Figure 2. An instance of the Data Management Server and kernel services will run on every server in the cell providing file services. In addition, on one server in the cell, an administration server is run. This server communicates with the data management servers on all data servers in the cell. In the simplest case, all servers and components may be run on a single machine.

The Glamour architecture facilitates fault tolerance by making each data server independent of each other. Inter-server communication occurs via reliable queues that can hold messages should a server fail or a network partition occur. When the server is restored or the partition healed, the queued messages will be sent bringing the server back to full currency.

The administration server is not duplicated in Glamour and could be run on a server configured for high-availability for added reliability. Note that Glamour data services run independently of the administration server. If the administrating server fails or a network partition causes communication with the server to be lost, all data services are sill available, only administration functions cannot be performed until the server is restored or the network healed.

The Glamour architecture supports multiple cells. Each data server can hold filesets for multiple servers and an administration server can be used to configured multiple cells. This allows cells to be relatively lightweight, making them more useful, more configurable to application needs. A desire to create a new cell for administration and security purposes does not require new hardware.

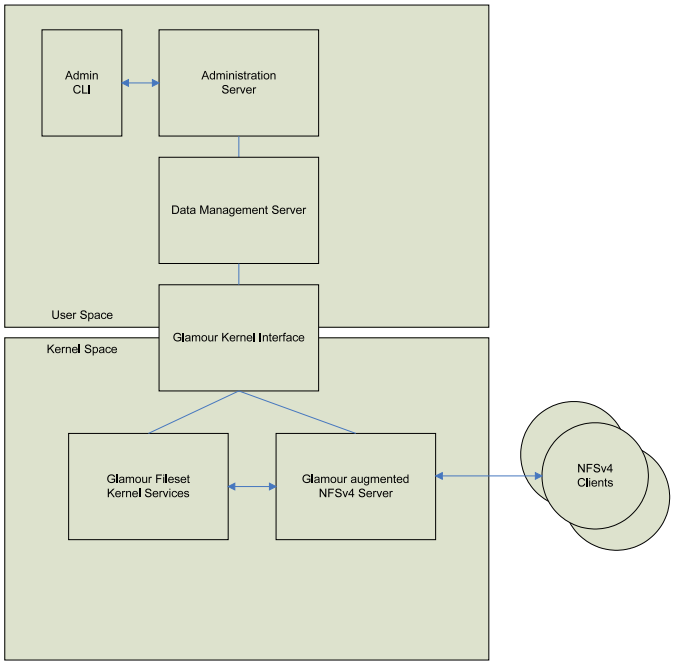

Figure 2: Glamour Architecture

6.1 Kernel Components

Glamour contains three groups of kernel services, generally implemented as loadable kernel modules.

6.1.1 Fileset Kernel Services

The fileset kernel services module implements the functions for filesets, e.g., create, modify, etc. These services are implemented via native filesystem calls for each operating system. These file services are generally called from the NFS server and the data management server as described below.

6.1.2 Kernel Interface Module

The kernel interface module is used by the various other components to communicate between user and kernel components. Communication occurs in both direction, e.g., from user to kernel space when the data management server requests fileset services and from kernel to user space when the NFS server requests location data for a fileset. Downcalls and upcalls may be implemented in different ways and varies across OSes although the API used the other components is the same in all cases.

6.1.3 NFS Server

The Glamour NFS server is a modified standard NFS server supporting version 4 of the NFS protocol.

The server is modified to be fileset-aware and to maintain the state necessary to implement fileset semantics (including multiple locations, failover, etc.) in a way invisible to the client. Implementation details were described in Section 4.

The modified server queries the fileset services component to determine fileset information such as boundaries, identifiers, etc. It queries the data management server via the kernel interface module to resolve client requests for fileset location information.

6.2 Server Components

6.2.1 Data Management Server

The data management server runs on each host in the cell providing file services. It responds to requests from the administration server to execute fileset operations. In response, it maintains a small local database of configuration information and uses the fileset kernel services via the kernel interface module to actually perform fileset operations. It communicates with other data management servers in the cell to perform fileset motion operations.

The data management server also responds to request, generally from the NFS server via the kernel interface, to look up locations for given filesets.

6.2.2 Administration Server

The administration server provides a single point of coordination for all Glamour administration functions. It maintains a complete database of all configuration of the cell and information within it, including servers. filesets, locations, users, groups, automation jobs, etc.

The administration server receives user requests and takes the appropriate action, updating its local database and forwarding commands on to the affected data management servers in the cell. All administration server communications, both with the administration client and the data servers, is implemented via a purpose-built protocol run over ONCRPC with support for RPCSEC_GSS and AUTH_SYS. The protocols are built on LDIF.

6.3 Administration Client

Administrative action occurs via an administrative client which talks to the administration server. Currently the client is implemented as a command line interface (CLI) which simply encodes and decodes user commands for processing by the administration server. The CLI can run on any host that can access the administration server via ONRPC with one of the allowed security protocols.

7 Evaluation

In this section, we evaluate the performance of Glamour using different workloads with an emphasis on measurement of overheads and scalabity of following referrals, client load balancing through server redirection, and client handling of server failure in the presence of alternate replica locations. We compare vanilla NFS systems against those with of a Glamour enhanced NFS server. We present two benchmarks: a micro-benchmark to determine the performance of individual common metadata operations that are affected the most by this work, and the Andrew benchmark to show overall performance on common filesystem operations.

7.1 Experimental Setup

Although our Glamour prototype has been built on both Linux and AIX, we present our Linux results only. We ran our experiments between a Linux NFS client and NFS servers in the following configurations:

- **Vanilla**: A vanilla setup using an unmodified Linux client and server. Results from these tests give us baseline numbers to evaluate the overheads and advantages of Glamour.
- **Glamour**: A standard[11] Linux client interacting with Glamour-enabled NFSv4 servers.

All experiments were conducted using identical IBM eServer xSeries 330 machines using 1.266GHz Intel Pentium III CPU, 2GB RAM and 36GB 7200rpm hard disk, and running Linux kernel 2.6.12.

7.2 Micro-benchmarks

Although not utilized in the performance evaluation since it does not yet support NFSv4, SpecSFS[27] was used to create the filesystem hierarchy and populate the files, creating 1 500 directories and 60 000 files totalling 1GB of data. All micro-benchmarks were run using this fileset. We also use OProfile[3], a system-wide profiler for Linux, to identify performance bottlenecks.

7.2.1 Namespace Traversal

We evaluate the overhead of following referrals to a different server and the effect of client redirection on application response time. To measure this, we perform a simple recursive listing using `ls -lR` on a tree created at Server A that is mounted by an NFSv4 client. At several points in the directory tree, mount points were created, each pointing to another server, Server B.

Figure 3 shows the response time for `ls -lR` with multiple directories on server A being set as mount point referrals. In the "multiple locations" case, each referral on Server

[11] Current Linux client implementation does not support replication and migration features of the protocol which are optional. Although we have modified the Linux client to enable these features, only changes as allowed by the protocol have been made.

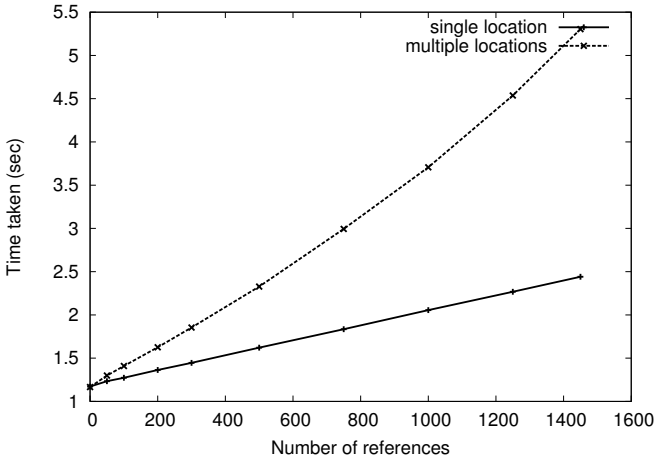

Figure 3: ls -lR traversal through 1500 directories with multiple reference points

A points to a different location on Server B. As the client traverses the namespace, it is redirected upon hitting each referral. On following each referral, the client performs the equivalent of an NFS mount operation, creating a superblock for each `fsid` transition. On completion, the client would have as many mounts as the number of referrals followed. In the "single location" case, all referrals on Server A point to the same location on Server B. In this case, since the root paths for all referrals are identical, the client needs to create only a single mount for all referrals. Profiling shows that the client incurs substantial overhead in superblock management as the number of mounts increases as it maintains a singly-linked list of all superblocks that makes lookups expensive. This is shown in OProfile numbers in Tables 7.2.1-7.2.1.

% time	Function name
10.34	`nfs_lookup_revalidate`
6.90	`decode_getfattr`
6.90	`nfs_readdir`
6.90	`nfs_update_inode`

Table 1: Percentage of total running time of the experiment used by individual functions on a Vanilla system.

% time	Function name
8.77	`nfs_lookup_revalidate`
7.02	`nfs_idmap_id`
5.26	`decode_compound_hdr`
3.51	`decode_getfattr`

Table 2: Percentage of total running time of the experiment used by individual functions with all referrals pointing to the same replica location.

It is noteworthy that once a client has traversed the namespace and resolved all referrals by creating appropriate superblock state, this information is cached so all future namespace traversals do not incur these overheads. This is shown in Figure 4 where the same experiment has been repeated after the client has already traversed the namespace once.

% time	Function name
91.09	`nfs4_compare_super`
0.65	`decode_getfattr`
0.37	`nfs_idmap_id`
0.28	`nfs_revalidate_inode`

Table 3: Percentage of total running time of the experiment used by individual functions with each referral pointing to a different replica location.

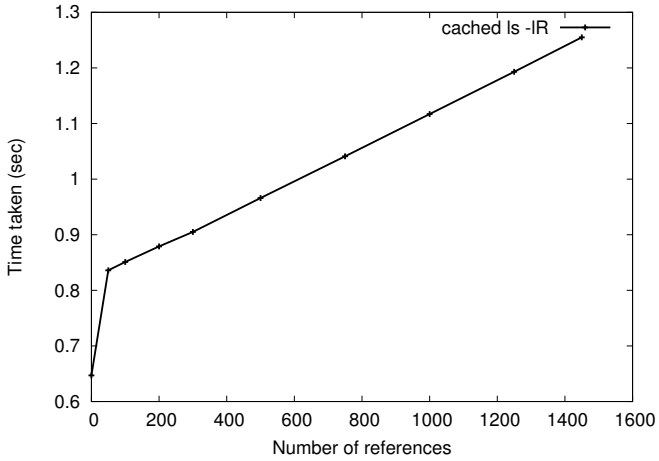

Figure 4: ls -lR traversal through 1500 directories with multiple referrals after attributes have already been cached

7.2.2 Replica Load balancing

To demonstrate the benefits of load balancing through replication and client redirection, we ran experiments that allow a client to access parts of the same fileset through multiple locations in parallel. Using the same fileset created earlier, we compare the time it takes a client to archive the entire fileset when reading from a single server with the time it takes for the same operation when spread across multiple servers. For this, we created a simple parallel version of GNU `tar` with multiple processes running in parallel each of which archives a portion of the directory. Each process operates on a Glamour fileset and the client is redirected to use all the replica locations. The simplest case where each replica location corresponds to a different server is shown in Figure 5.

7.2.3 Failure recovery

In a traditional NFSv4 setup with no replica locations, a client detects server failure and resumes normal operations after server recovery by re-establishing state with the server. Client operations ongoing when the server is rebooting are delayed (and retried) until the server is functional again. On the other

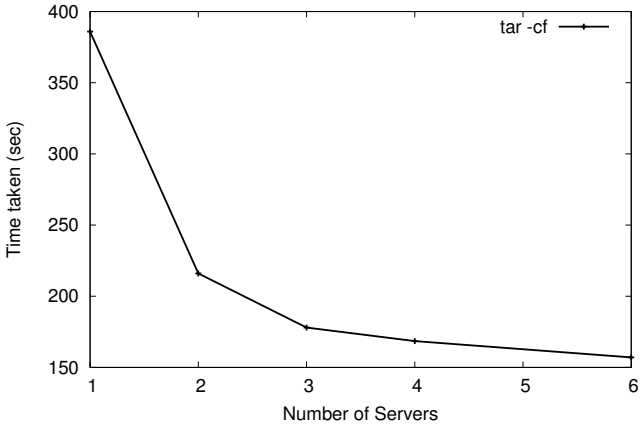

Figure 5: Archival of data replicated across multiple servers.

hand, with replication support, an NFSv4 client can detect unresponsive or poorly performing filesystems and choose to access the same data from alternate replica locations. A client can greatly reduce application response times on server failure by automatically redirecting to a replica. We demonstrate this in Glamour with a simple experiment where a client continuously traverses a directory on a server. We initiate a server reboot and compare the times it takes for the client to resume operations with a standalone server, and a server with replicas. The time taken for a client to recover in a Glamour-enabled environment depends on the timeout settings on the client that are configurable. In the experiment, a client marks a server that has been unresponsive for 10 seconds as temporarily unavailable. We conclude that Glamour can match the response time of a vanilla system failure and recovery with 8 concurrent replica failures.

Operation	Time (sec)
Vanilla with no failure	00:24
Vanilla with failure	04:14
Glamour with Client Redirection	00:34

Table 4: Time taken to traverse a directory with `ls -lR`. A vanilla system experiences delay on server reboot. A Glamour-enabled system allows the client to automatically failover the operation to a replica.

7.2.4 Filehandle Expiration

As mentioned in Section 4.1.2, a Glamour-enabled NFSv4 server will expire client filehandles in the events of fileset promotion and demotion. This requires the client to prepare itself to recover from expiration of volatile filehandles. It does this by storing the component names leading upto and including the filesystem object in question, and redoing object lookups by starting at the root of the server's filesystem namespace. While fileset operations like promote and demote are typically scheduled at times of low activity, an administrator is also allowed to dynamically create filesets. We experiment with the overheads of expiring filehandles and its effects on application response time. We repeat the previous `ls -lR` experiment to allow a client to cache the entire directory metadata. After all filehandles are cached, we trigger a fileset promote event that results in all filehandles being expired. We then measure the time it takes for the client to recover all the filehandles (Table 7.2.4).

Operation	Time (sec)
Vanilla	3.7
Glamour with Filehandle Expiration	11.16

Table 5: Time taken to traverse a directory with `ls -lR`. After expiration, client incurs additional processing to recover all filehandles from the cached component names.

7.3 Modified Andrew Benchmark

This section evaluates Glamour's performance by emulating a software development workload using a modified version of the Andrew Benchmark[14]. We show that Glamour has acceptable performance in the presence of reference points in the namespace in a environment of sequential workload with no data sharing. We also show that if the workload can take advantage of multiple replica locations by running operations in different filesets in parallel, Glamour demonstrates better performance as compared to a single server.

The benchmark works with an existing fileset (source tree). Phase I (*mkdir* of the benchmark creates directories in the filesystem being tested; phase II (*copy*) copies the files from the fileset into the directories created; phase III (*stat*) recusively lists all the directories; phase IV (*grep*) scans each copied file; and phase V (*make*) performs a compilation of the source tree. We created a variant of the Andrew benchmark to demonstrate the effectiveness of data access from multiple replica locations:

- We use, as input, the Linux kernel source[12] which contains 1076 directories, 17361 files and approximately 230MB of data.
- We create mount points at various places in the directory tree during the *mkdir* phase so that subsequent operations are spread over two servers through client redirection[13]. We compare this scenario with a single server with no replica locations.
- In the *make* phase, we run multiple jobs simulataneously (with `make -j [number of servers]`) to allow the compilation to take advantage of replica locations.

Table 7.3 shows the results of running the benchmark using a single client with a vanilla system, and with 2 and 4 servers. Since the first four phases employ a sequential metadata workload, there is no advantage of using multiple replica locations. In fact, the response times are slightly worse due to client redi-

[12]Kernel version 2.6.12

[13]Replica locations are created by observing parallelism demonstrated by `make -j`

rection on following referrals. The benefits are clear in phase V where operations can be run in multiple filesets in parallel[14].

Phase	Vanilla	2 servers	4 servers
mkdir	00:36	0:34	0:32
copy	37:11	37:41	38:15
stat	00:50	00:58	01:06
grep	01:07	01:11	01:19
make	17:34	13:05	12:40
Total	57:18	53:29	53:52

Table 6: Times in seconds for different phases of the Andrew benchmark when run on a vanilla system, with 2 servers having 5 filesets each, and with 4 servers having 4 filesets each.

8 Related Work

A number of previous research prototypes and commercial systems have explored some aspects and featuers of Glamour. Most notably AFS [2, 13, 14], which is a globally distributed filesystem, introduced a number of concepts that we refine or reuse in Glamour. AFS introduced the concept of a cell as an administrative domain and supports a global namespace. AFS also introduced the volumes [25] abstraction for data management. AFS has extensive client-side file caching for improving performance and supports cache consistency through call backs. AFS allowed read-only replication that was useful for improving performance. The successor to AFS was the IBM DFS [15] filesystem which had most of the features of AFS but also integrated with the OSF DCE platform. DFS provided better load balancing and synchronization features along with transparency across domains within an enterprise for easy administration. There were other AFS related filesystems such as Coda [18] that dealt with replication for better scalability while focusing on disconnected operations.

In contrast to the global scale of AFS, NFS [9], a widely deployed distributed file system, was designed to be simple, stateless and operating system independent using a simple client-server model. Glamour ofcourse relies on NFSv4 [24] for its operation. Unlike earlier NFS versions, the new NFSv4 protocol integrates locking, windows-style share semantics, stronger security, compound operations and client delegations. NFSv4 is stateful as it supports locking and file delgations and is firewall friendly as it uses a well known port and TCP. Moreover, v4 supports a stateful OPEN/CLOSE operation with share semantics similar to Windows. It also eliminates all secondary protocols that were used in v3(e.g., mount and NLM). NFSv4 also adds a host of new features that are leveraged by Glamour. These include volatile filehandles, client redirection, replication and migration support, and a pseudo namespace.

Similar to NFS, in the MS Windows world, CIFS [26] provides the remote server file access support. The Windows DFS [21] protocol extends CIFS to provide a common namespace using server directories and can redirect clients to other CIFS servers similar to NFSv4.

Recently there has been some work on leveraging the features of NFSv4 to provide global naming and replication support. In [31] the focus is on providing a global namespace and read-write replica synchronization. Other related efforts are geared toward improving performance by using parallel data access [11, 12]. Numerous IETF drafts proposal highlight the design considerations for NFSV4 naming [29, 10], the issues related to replication and migration [30] and provide an implementation guide for the NFSv4 referral support [22]

Orthogonal to the distributed file system work, there has been a number of commercial clustered file systems clustered filesystems [23, 4, 16, 28]. These are all geared for high-performance solutions using high-speed network connections and tightly coupled servers.

A large assortment of research protoypes have explored grouping together servers for a common file service. The xFS file system [8] decentralized the storage services across a set of cooperating servers in a local area environment. In contrast, the Oceanstore [19] project is an archival system, aimed at storing huge collections of data using world-wide replica groups with security and consistency guarantees. Another effort that focuses on security and byzantine faults, is Farsite [7] where a loose collection of untrusted insecure servers are grouped together to establish a virtual file server that is secure and reliable. Archipelago [17] couples islands of data for scalable internet services.

9 Conclusions and Future Work

In this paper, we demonstrated a commercially-viable architecture of a federated filesystem middleware layer that provides a common namespace and relies only on off-the-shelf client and protocol implementations. We detailed the design and implementation of a virtualization layer that can support flexible fileset granularities on top of any standard filesystem. Finally, we demonstrated how Glamour enhances data mobility and location independence by replicating and migrating data units without disrupting the client and client applications.

We described the Glamour implementation that runs on Linux and AIX. Using our testbed infrastructure, we demonstrated how Glamour provides a common namespace and performs replication, load-balancing and fail-over. We detailed the performance overheads of client redirection both at the server and at the client and the Andrew benchmark results.

We envision that once the Glamour infrastructure and the mobility of data is established, we can go a step further to automatically create filesets and place them at different locations and perform automatic load balancing with regard to the server and network conditions. We also plan to extend the implementation to handle read-write replication, implement a stateful data migration across different filesystems, and better understand how Glamour can be used to leverage the features of

[14]The paralellism in make decreases when using 4 simultaneous jobs instead of 2 which is why `make` with 4 servers is not much better.

clustered filesystems.

References

[1] DiskSites: http://www.disksites.com.

[2] OpenAFS: http://www.openafs.org.

[3] OProfile: http://oprofile.sourceforge.net.

[4] PanFS: http://www.panasas.com/panfs.html.

[5] Rsync http://rsync.samba.org.

[6] Tacit: http://www.tacit.com.

[7] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[8] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. *ACM Trans. Comput. Syst.*, 14(1):41–79, 1996.

[9] B. Callaghan, B. Pawlowski, and P. Staubach. NFS version 3 Protocol Specification. RFC 1813.

[10] C. Fan, D. Noveck, and M. Wurzl. Nfsv4 global namespace problem statement. IETF Draft, `http://www.ietf.org/internet-drafts/draft-fan-nfsv4-global-namespace-00%.txt`.

[11] G. Gibson, B. Welch, G. Goodson, and P. Corbett. Parallel NFS requirements and design considerations. IETF Draft.

[12] D. Hildebrand and P. Honeyman. Exporting storage systems in a scalable manner with pNFS. Technical Report TR-05-1, CITI, 2005.

[13] J. Howard and et al. An overview of the andrew filesystem. In *Usenix Winter Techinal Conference*, Februrary 1988.

[14] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst.*, 6(1):51–81, 1988.

[15] IBM. DFS Administration Guide.

[16] IBM. IBM Storage Tank – A Distributed Storage System.

[17] M. Ji, E. W. Felten, R. Wang, and J. P. Singh. Archipelago: An island-based file system for highly available and scalable internet services. In *Proc. of the 4th USENIX Windows Systems Symposium*, August 2000.

[18] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *SOSP '91: Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 213–225, New York, NY, USA, 1991. ACM Press.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 190–201, New York, NY, USA, 2000. ACM Press.

[20] Lustre. The lustre storage architecture.

[21] Microsoft. DFS Technical Reference.

[22] D. Noveck and R. C. Burnett. Implementation guide for referrals in nfsv4. IETF Draft.

[23] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. of the First Conference on File and Storage Technologies (FAST)*, pages 231–244, Jan. 2002.

[24] S. Shepler and et al. NFS version 4 Protocol. RFC 3530.

[25] R. Sidebotham and et al. Volumes— the andrew file system data structuring primitive. In *European UNIX System User Group Conference*, September 1986.

[26] SNIA. Common Internet file System (CIFS) Technical Reference. SNIA.

[27] spec.org. SpecSFS: http://www.spec.org/sfs97r1.

[28] A. Sweeney and et al. Scalability in the XFS file system. In *USENIX conference*, San Diego,CA, 1996.

[29] R. Thurlow. A namespace for NFS version 4. IETF Draft, `http://www.ietf.org/internet-drafts/draft-thurlow-nfsv4-namespace-00.tx%t`.

[30] R. Thurlow. A server-to-server replication/migration protocol. IETF Draft.

[31] J. Zhang and P. Honeyman. Naming, migration, and replication in nfsv4. Technical Report TR-03-2, CITI, 2003.