Global namespace for files

by O. T. Anderson
L. Luan
C. Everhart
M. Pereira
R. Sarkar
J. Xu

We propose a name service that enables construction of a uniform, global, hierarchical namespace, a key feature needed to create a file-system grid. Combined with other grid replication and location-lookup mechanisms, it supports independence of position for users and applications as well as transparency of data location in a scalable and secure fashion. This name service enables federation of individual files as well as file-system trees that are exported by a variety of distributed file systems and is extensible to include non-file-system data such as databases or live data feeds. Such a federated namespace for files can be rendered by network file servers, such as NFS (Network File System) or CIFS (Common Internet File System) servers, proxies supporting the NAS (network-attached storage) protocol, or grid data service interfaces. File access proxies, which handle protocol translation, can also include caching and replication support to enhance data access performance. A uniform namespace with global scope and hierarchical ownership allows sharing file data between and within organizations without compromising security or autonomy.

The major goal of grid computing is to foster sharing of widely distributed resources. All distributed systems are faced with network limitations and scaling. The grid shares these fundamental problems with other distributed systems, but unlike more tightly coupled systems, such as clusters, these problems cannot be eliminated, but must be accepted as part of the environment in which the grid operates. Crucial network problems include large latencies and low bandwidths; the question is not how to improve these parameters as much as how can the system accommodate these limitations. The large scale of grid systems expands the boundaries containing current systems in the number of applications, users, groups, domains, and hosts, the variety of their operating systems, and the diversity of protocols. A key result of this scale is that the security mechanisms of small groups break down, and stronger and more formal measures are needed. Security is also important because owners are more willing to share their resources when they can maintain control over them.

Files represent an especially fundamental resource for collaboration within grid virtual organizations.[1] The usage of file data is strongly impacted by features of the distributed environment, especially latency, bandwidth, and hot spots, to name a few. The grid community has tackled problems affecting the use of file data with design and development in the areas of security, data transport, and replication.

The Grid Security Infrastructure[2] (GSI) provides strong security based on public keys and certificates for authentication. Though GSI does not support groups of users or access control lists (ACLs) to sup-

0018-8670/04/$5.00 © 2004 IBM

port authorization decisions, some separate efforts have begun to address the authorization problem. The development of GridFTP[3] consists of extensions to the venerable FTP (File Transfer Protocol), a reference implementation of a server, a client library, and various utilities to provide secure and high-performance access to files. The Replica Location Service[4] (RLS) has addressed the problem of locating replicas of file data. Collectively, these mechanisms make significant contributions to problems of exchanging files within a grid.

In grid computing, management of distributed data presents challenging problems in data naming and organization. Data grids are implementations of data virtualization services providing uniform access, management, and control mechanisms for distributed data. Several notable products and projects can build data grids across geographically dispersed areas. Among them, the Storage Resource Broker (SRB)[5] is client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. Avaki software[6] provides a single data service that makes data from multiple, distributed, heterogeneous data sources available to grid applications.

One shortcoming of this suite of grid tools is the lack of a namespace to support sharing files. This namespace would allow uniform and global path names to persistently address file data within a grid wherever it is located. Indeed, a suitably open naming architecture could include data from sources other than GridFTP, such as NAS (network-attached storage) file servers, and potentially even non-file sources like databases, thus greatly expanding the scope and utility of the data grid.

We propose a global namespace service (GNS) to provide a naming mechanism to link existing data sources. The term "service" is not intended narrowly as a Web or grid service, but generically to encompass a collection of repositories of namespace data linked to clients realized in various technologies providing ubiquitous access. The GNS is supplemented with security infrastructure, data management systems, location services, and file data sources to create a global namespace. Along with distributed file access protocols and ways of mapping between mismatched clients and servers, this namespace enables a *file-system grid*.[7] This phrase is intended to describe a specific case of a data grid that federates existing exported file systems[8] into a virtual global file system.

The namespace is uniform, global, and hierarchical. *Uniformity* means that it has transparency of position and location. Transparency of position means that the data consumers, users and their applications, can be mobile or have multiple positions. Similarly, location independence allows the data to move from place to place. Providing these capabilities requires an abstraction between the name and location of the data, which allows this mapping to be done dynamically. This separation between the logical-name and physical-location properties of the data allows them to be managed separately. This results in crucial administrative scalability through the delegation that this decoupling makes possible.

Users affect the logical layout of the data by means of namespace operations such as creation, deletion, and renaming. Infrastructure managers are responsible for deploying and retiring servers, adding and removing disks, and configuring networks to optimize a system's ability to handle the demand for data services. The logical layout may be very long-lived, limited only by the lifetime of the owning organization. On the other hand, the physical structure must adapt to short-term changes in load, rapid technological evolution, and organizational changes such as acquisitions and outsourcing.

The global nature of the grid means that logical names must be uniform across administrative domains such as corporations and nations, to support virtual organizations as well as more loosely defined groups. In addition, names are hierarchical, making a path name a descriptive term for them. Path names follow common hierarchical patterns of ownership and control and so make a good model for organizing the namespace.[9]

Path-name prefixes give names to aggregates of file-system data, such as an organization's MS/Dfs (Microsoft Distributed File System) tree of shares or a filer's exports. The GNS can link these existing resources into an enterprise-wide and even a cross-domain namespace that looks the same to everyone everywhere. This federation of file systems is especially valuable if file-access protocol conversion is achieved or multiprotocol servers or clients are available. The namespace also facilitates scalable data management of aggregates by enabling collections of appropriate

size (e.g., servers, file systems, or shares) to be the objects of administrative operations.

The GNS namespace consists of virtual directories that contain other virtual directories and junctions. A *junction* is an object that points to files and subtrees provided by data sources or to namespace servers. Namespace clients traverse path names by performing lookups at a series of GNS instances, yielding a file or file-system reference. This reference is generally a logical name that is mapped by a location service to a physical file server reference such as a URL (Uniform Resource Locator) specifying the protocol, host, and export name. Junctions, in contrast, may instead contain direct physical references. Namespace clients can be implemented in a file-system proxy, a client machine (e.g., using an "automounter" backend),[10] or a server running a protocol supporting referrals (e.g., CIFS [Common Internet File System] or NFS [Network File System] Version 4) that may export any file data or an application library. The best choice of implementation vehicle depends on the environment of each domain in which GNS is deployed.

In the following section, we survey related work in this field. The core of the paper consists of two sections, one detailing the features of GNS and the second describing how the namespace fits into the larger data delivery process. Following this general description, we provide some implementation experience, alternative approaches, and usage scenarios, with the aim of translating the relatively abstract concepts of GNS into a variety of concrete terms and situations. The goal of this mostly functional description of GNS and its proposed usage is to stimulate discussion on the benefits namespaces can provide to the collaborative process and operational efficiency in general.

Related work and technologies

This section presents some of the most significant existing software for distributed file systems and considers their use in the grid environment.

CIFS and NFS. Traditional distributed file systems such as Network File System (NFS) and Common Internet File System[11] (also called SMB [Server Message Block]), collectively called NAS file systems, pose problems in grid and wide-area network (WAN) environments. These protocols were designed for use on fast LANs (local-area networks), and the long latencies of WANs are particularly harmful to their performance. Each has its own naming architecture whose design point was the small workgroup, and their security is often weak for this reason. Their support for replication is limited in many implementations, and support for location independence is almost nonexistent. Collectively, these features now make these protocols seem provincial compared with the enterprise and global-scale demands that distributed file systems now face.

A fourth version of NFS (NFSv4), developed under the auspices of the IETF (Internet Engineering Task Force), has explicitly addressed some of the problems of WAN performance, cross-domain operation, and security. This protocol has reached the draft standard stage as document RFC 3530.[12] It does not provide a global namespace, but relies upon the same automounter mechanisms that supplement naming for earlier versions of the protocol. NFSv4 (and CIFS) provide protocol support for server-side referrals that can be used to stitch individual file systems together, but there are no standards for representing the information that defines the namespace. Unfortunately, many of the WAN-friendly features are optional, and so may not see wide deployment soon; and the substantial complexity of the protocol will slow the availability of high quality implementations. The NFSv4 design effort does share many goals with the grid community, and the protocol has promise as an important component of the grid computing arsenal.

AFS and DFS. Experience with AFS*[13] (Andrew File System) and DFS*[14] (Distributed File System) has demonstrated many of the benefits of a global, uniform, hierarchical namespace. These file systems feature excellent security, good WAN performance, and peerless scalability. Both scale successfully over a wide range of geographies, numbers of users, servers, clients, and data capacities. An unusual feature both provide is that of authorization groups that can be created and maintained by individual users, which improves the convenience and utility of ACLs while reducing the administrative burden. AFS file storage has a proprietary format that DFS improves upon by allowing export of any of the server's local file systems, though many advanced data-management and security features are unavailable unless the proprietary DFS local file system is used.

Neither of these file systems, however, is suitable as a grid file system, though many of the grid ideas and those of other distributed systems have been modeled on them. Both AFS and DFS require complex,

custom, in-kernel software to be developed for each client platform and to be installed on each client machine. Part of the motivation for the present work is to make use of the technologies that have evolved since the inception of AFS and DFS and to provide some of their benefits without requiring extensive in-kernel modifications to every client, by using platform-standard facilities where possible. Authentication and authorization for both AFS and DFS are deeply integrated into their structure, and to the substantial extent that grid facilities are similar but different in detail, their suitability for the grid suffers. Because AFS is now "open source" and is supported by a lively community, it could conceivably evolve one day into a grid file system; at that point it could be integrated into the system we describe. The ownership for DFS is complex and not open, and it is not likely to emerge from this morass of legal entanglements and the absence of support from its few remaining vendors, thus rendering the DFS product a dead end.

Namespace clients traverse path names by performing lookups at a series of GNS instances.

Avaki. The Avaki Data Grid** product provides a uniform namespace built from heterogeneous file systems using a proprietary solution. The system-level architecture at a domain level consists of a network of share, grid, and access servers, as well as a grid domain controller, which collectively manage a global namespace. Share servers export data and meta-data from a variety of native (physical) file systems. Grid servers cache file-system meta-data from multiple share servers and facilitate the creation and management of virtual directories in the global namespace. Access servers provide access to the global namespace via NFS, FTP, and HTTP (HyperText Transport Protocol) protocols and provide data and meta-data caching without enforcing a strong consistency model. The grid domain controller interfaces with LDAP (Lightweight Directory Access Protocol) and NIS (Network Information Service) to authenticate end users accessing the file-system namespace. The system does not address replication or use Globus Toolkit** components such as GridFTP or RLS. Researchers at Avaki proposed a Secure Grid Naming Protocol[15] (SGNP) that has some high-level similarities to GNS. SGNP does not specify mappings to file-system objects.

Storage Resource Broker. Storage Resource Broker (SRB) from the SDSC (San Diego Supercomputing Center) is middleware that provides a comprehensive distributed data-management solution, with features to support the management, collaborative (and controlled) sharing, publication, and preservation of distributed data collections. It has a rich set of APIs (application programming interfaces) to a management layer providing a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. The APIs allow higher-level applications to be built on top of a wide variety of storage systems. SRB, in conjunction with the meta-data catalog, provides a way to access data sets and resources based on their attributes rather than their names or physical locations. This is done by using a logical namespace with a mapping for each data object (file) from a logical to a physical name and location. Each data set stored in SRB has a logical name, which may be used as a handle for data operations. In SRB, files or objects are represented as data sets, or digital entities. Similar to the virtual directory in GNS, the term *collection* indicates a hierarchical tree structure used to group multiple data sets. Although SRB provides a complete solution for distributed data management and access in grid computing, it requires significant deployment efforts. Because SRB is more than simply a namespace, a working SRB system requires a great deal of infrastructure in order to store and access data.

Globus.** The Globus Toolkit contains key features needed for a file-system grid. The GSI provides strong security based on a public key infrastructure using server certificates and user proxies. This arrangement may not scale to global scope without additional mechanisms, but it has many benefits, such as security and autonomy, that provide a stable foundation. Building on that foundation are several authorization systems, for example, the Community Authorization Service,[16] though the group and ACL model of AFS and DFS may be more suitable for file-system use. RLS provides a mechanism for locating replicas by mapping between logical file names and physical file names. The logical file names are flexible enough to allow use as a location service even for unreplicated data. This flexibility results from a lack of struc-

ture that prevents RLS from providing a general solution to the namespace problem. While its focus on file-level replication is a scalability concern, this can be mitigated to some extent by deploying one or more instances per domain.

The file transport standard in Globus is GridFTP, a secure and efficient protocol based on FTP extended to support GSI, parallel data transfers, and other features. Enhancements in Version 3.2 of the toolkit, which provide machine-readable file listings[17] and chmod(),[18] improve support for attributes. Still missing is a means for modifying other attributes and support for synchronization operations such as byte range or open locking. However, it remains uncertain how far GridFTP should be extended in the direction of making it a distributed file system.

Plan 9. The Plan 9[19] operating system uses the file system namespace to represent, control, and monitor virtually all system resources. The file system's message-based interface, called 9P, provides a uniform access method for all these objects. The 9P protocol is easily mapped onto a secure channel and can provide interprocess and network communication. Each process has its own private namespace to hold all the resources that it can access. Resources can be passed between processes and across the network to allow sharing and location-independent access. Plan 9 has been proposed[20] as an ideal grid operating system, called 9grid, because it naturally provides secure, uniform access to resources locally and across the network. Many of the mechanisms necessary for a grid toolkit on other systems are basic features of Plan 9.

The namespace-oriented architecture of Plan 9 extends the UNIX device model and the /proc file-system concept to all accessible resources and shows the power of unified naming and access methods. Each process has its own namespace under its control, which supports privacy, security, and autonomy. Because completely private namespaces would be of limited utility, Plan 9 uses delegation when mounting remote file systems and conventions to assure uniformity of major branches of the namespace. Plan 9 is an operating system, and interoperation relies upon homogeneous nodes and a uniform file-system protocol. However, the ideas and experience are relevant to the present global namespace effort.

Other related technologies. To leverage existing design and development efforts in this area, those implementing GNS should consider existing components. Possibilities to consider include OGSA (Open Grid Services Architecture), DNS (Domain Name System), and LDAP.

OGSA and WSRF. The rapid development of the Internet makes access to file data via Web services a requirement for grid computing. A service-oriented architecture (SOA) is based on the premise that a system can be decomposed into a collection of network-connected components. SOA describes the overall approach of building loosely coupled distributed systems. Grid computing provides an evolving open set of standards for Web services and interfaces that are based on SOA and make services, or computing resources, available over the World Wide Web.

Open Grid Services Architecture (OGSA) is a Global Grid Forum (GGF) standard for building a basic platform to support the plugability and composability of heterogeneous resources, including file systems. The Web Services Resource Framework (WSRF) is a framework that models stateful resources and codifies the relation between Web services and resource participants. WSRF is a convergence standard between grid and Web services. In this framework, the WS-Resource Properties standard defines how data is associated with a stateful resource, and the WS-Resource Lifetime standard allows users to specify the period during which a resource definition is valid. The GNS is designed to work under OGSA and be plugged into WSRF. In particular, the OGSA data service proposed by the GGF DAIS (Database Access and Integration Services) Working Group provides a unified data access and management paradigm based on WSRF. The GNS could be used by an implementation of an OGSA data service to access file data with a global namespace.

Domain Name System. The Domain Name System (DNS) has a long history of robustness in the open and large-scale environment that GNS also targets. The architecture and existing implementations used by DNS may be suitable for holding GNS data, especially for naming organizations near the root of the namespace. The existing infrastructure for managing domain name information, however, is probably not conveniently extended to hold lower-level file system namespace entries. The coordination necessary to share a database used for two such disparate purposes would probably be more trouble than it was worth.

LDAP. Because this is perhaps the most prominent directory service available today, it is natural to ex-

amine the possibility of using it to implement GNS instances.[21] Using existing LDAP deployments in enterprises and organizations can help reduce the management overhead of maintaining GNS and help consolidate the information and technology infrastructure. LDAP defines a communication protocol for accessing and searching a database of entries annotated with attributes. Accordingly, LDAP does not define a directory service but rather the transport and format of messages used by clients to access data in a directory. These messages deal with directory entries, whose structure and relationships are described by a schema.

An earlier system, proposed by Sun Microsystems and standardized by x/Open (now The Open Group), defines an API for federating directory services such as x.500, DNS, and NIS, which is called x/Open Federated Naming[22] (XFN), though it does not address specifically how file systems can be federated into one namespace.

It may be useful at this point to identify the principal characteristics of a directory, as compared to a general-purpose relational database, to better understand the connection between the proposed hierarchical namespace service and existing directory services such as those built using LDAP. A directory is a structure for organizing information about objects that maintains information about each object; a common analogy is a city telephone directory. In the context of LDAP, a directory may be defined as a specialized database serving relatively static information that is optimized for high-volume read operations. In contrast, general-purpose relational databases are more efficient at storing information that changes rapidly. Relational databases typically require a more powerful and complex method of access, such as structured query language (SQL). Directories use a simplified and optimized access protocol that can be used in thin and relatively simple applications. Because directories deal mostly with read requests, they generally do not support transactions. As a result, strict consistency in a distributed environment is usually not practical.

Needless to say, there are a number of LDAP characteristics that fit nicely with those of a namespace service. First, data stored in a GNS should be relatively static under normal circumstances. Secondly, because a GNS offers a specific service, thin and optimized clients are possible and desirable. Other features defined by LDAP that appeal to GNS include security, delegation, and referral mechanisms.

Despite these similar characteristics, a well-designed namespace service has special requirements with respect to hierarchical data management, consistency, and replication. In hierarchical data management, an inherent obstacle with standard LDAP implementations is the inability to modify (e.g., to rename) entries within the hierarchy without affecting nested subentries. This is due to the composition of the hierarchy and how the hierarchy is established and defined by LDAP. The LDAP naming model incorporates a unique name, known as the distinguished name, which unambiguously identifies each entry. Each entry's distinguished name, its fully qualified name, is based on its parent's distinguished name. Hierarchy within LDAP is therefore constructed through the relationship found between entries and their respective distinguished names. This model requires that all subentries, relative to a given entry within the hierarchy, must be modified in the event that an entry is renamed or moved. A relational database, on the other hand, can effortlessly modify an entry's attributes without affecting any of its child entries because entries can easily be indexed by an internally unique identifier that does not serve as an attribute of the entry outside of the relational database. Additional efficiencies can be realized with a relational database when used as a distributed namespace management repository with file-system-oriented requirements, such as renaming, moving, and recursive operations.

Moreover, since relational database management systems (RDBMS) generally support transactional operations, it is naturally easier to ensure stricter data consistency between read and write operations. Most RDBMSs incorporate optimized mechanisms for data replication, with transaction-based features, in a distributed environment. In fact, certain vendor products, such as IBM's eNetwork LDAP directory, use a relational database for their implementation. As a result of examining the architectural purpose and function of various technologies currently available, we find that the most efficient, scalable, standards-based solution for a distributed topology of GNS services would be comprised of a request-handling engine backed by a relational database that incorporates LDAP, among others, as the communication protocol. This does not, however, imply that a relational database is required to implement a GNS, but rather suggests that greater efficiencies can be realized when a GNS is implemented as described.

Table 1 Junction types

Type	Example	Description
GNS junction	gns://naming.ibm.com	GNS instance
Logical file-system name	rls://rls.arc.ibm.com/storage	Subtree located via location service
Logical file name	rls://rls.arc.ibm.com/8493802	File located via location service
Physical file-system name	gsiftp://cvs.shark.tuscon.ibm.com/src	Subtree on specific server
Physical file name	http://grid.almaden.ibm.com/arc/info/papers.zip	File on specific server

Table 2 GNS mapping table

Node	Type	Target
/gfs/globus.org	GNS junction	gns://gns.globus.org
/gfs/ibm.com	GNS junction	gns://gfs.ibm.com
/gfs/ibm.com/SG/	Virtual Directory	
/gfs/ibm.com/SG/shark	Physical file-system name	gsiftp://cvs.shark.tuscon.ibm.com/src
/gfs/ibm.com/SG/empData	Logical file-system name	rls://rls.hr.ibm.com/TucsonEmployeeData
/gfs/ibm.com/ARC	GNS junction	gns://namespace.almaden.ibm.com
/gfs/ibm.com/ARC/csstorage	Logical file-system name	rls://rls.arc.ibm.com/arc_cs_storage
/gfs/ibm.com/ARC/ais	Virtual Directory	
/gfs/ibm.com/ARC/ais/fileX	Logical file name	rls://rls.arc.ibm.com/8493802

Table 3 Location service mapping table

Logical name	Physical location
TucsonEmployeeData	cifs://nas3.tucson.ibm.com/employeeData
arc_cs_storage	gsiftp://storage.almaden.ibm.com nfs://nas1.almaden.ibm.com/csstg
8493802	gsiftp://gridftp1.almaden.ibm.com/infofiles/8493802 https://w3.almaden.ibm.com/arc/info/8493802

GNS features

This section presents the features of the GNS, including its directory structure, meta-data use, and interfaces.

Directory structure and junctions. A GNS instance holds a tree of virtual directories originating at a root directory. The root and each subsidiary directory contain named entries that have one of two types: directory or junction. Both types of objects may have other attributes, but a junction is particularly characterized by its target, which takes the form of a URL. The target URL addresses a GNS instance or a file-system object.

Path names are evaluated from the GNS root by traversing intermediate subdirectories until the path name is exhausted or a junction is reached. A junction in a GNS or file-system directory represents a delegation of authority for evaluating subsequent path name components. Both types of junctions insert subtrees into the namespace. Thus, junctions construct the namespace from various directory services.

File-system trees are generally terminals of the namespace because most file systems do not contain junctions. Junctions could be inserted into file systems, however, to allow nesting of file-system trees within other file systems. Such junctions could be implemented several ways: using a special file-system object (e.g., a specially marked symbolic link or directory), using a separate database, or with GNS virtual directories. Interpreting these junctions would

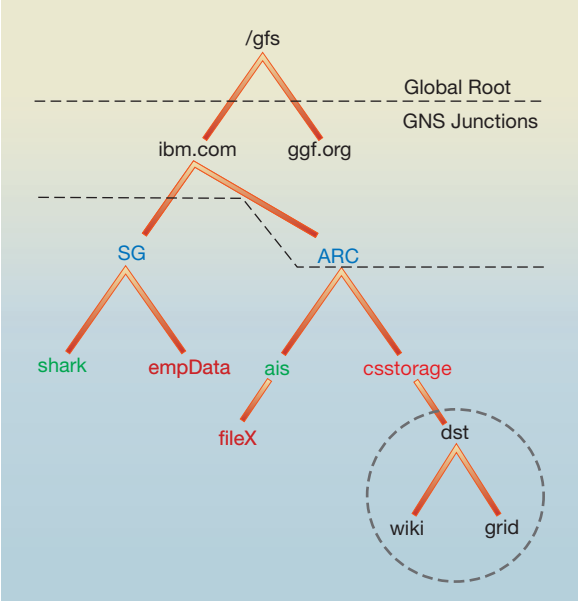
require logic in the client or server code to handle the referral to the target service.

A GNS junction target conceptually has one of five types, summarized in Table 1. In one type, the target is another GNS instance. The other four file-system target types may be either logical or physical and point to files or directories. The term *file junction* is used for logical or physical junctions whose target is a file; similarly, the target of a *directory junction* is a file-system directory. The four types are logical file name, logical file-system name, physical file name, and physical file-system name.

The URL for logical targets indicates both the location service address and the resource name for the file or subtree that is used as a lookup key in the location service. The result of the location lookup is a physical target. A physical target is one or more URLs that include the information necessary to contact the file service. This information includes protocol, interface, host, export, and path prefix. Replicated data is indicated by multiple URLs and therefore requires a selection step to pick the most suitable replica. The name service only returns the junction target; the tasks of contacting the location service and replica selection are the responsibilities of the GNS client.

An example of GNS use is depicted in Figure 1 and annotated in Tables 2 and 3. Under the global root directory, /gfs, two GNS junction points are shown, which provide the junctions to the namespaces maintained separately by two different organizations ("ibm.com" for IBM and "ggf.org" for the Global Grid Forum). Within ibm.com, SG is a virtual directory, and ARC is another GNS junction point. Only virtual directories are represented in GNS. They do not reside in any physical file systems, but they allow physical file systems or files to be hierarchically organized in the virtual namespace. Under SG, shark is a junction to the physical directory subtree available at the GridFTP server cvs.shark.tucson.ibm.com, while empData is a junction to the logical name TucsonEmployeeData, which is resolved to a CIFS share via a location service at rls.hr.ibm.com. Under ARC, csstorage is a junction to a logical directory subtree with the name arc_cs_storage. This logical directory subtree name is resolved through a lookup to a location service running on rls.arc.ibm.com. This logical directory subtree is shown served out of a GridFTP server csstorage.almaden.ibm.com and an NFS file system csstorage on nas1.almaden.ibm.com. A junction to a logical file, fileX, (in ais, a virtual directory) is also shown under the ARC virtual directory. GNS maps file X to a logical file name 8493802, which is used for lookup at the location service and mapped to a GridFTP server-based copy of the file and a secure HTTP-based version.

Figure 1 A GNS example



Meta-data and access control. A key property of any file system is that it attaches meaning to path names by mapping the visible paths to file contents. This is no less true of the namespace constructed using GNS. Therefore the security of reads and writes is crucial to ensuring that the meaning of a path name assigned by a writer is preserved for all subsequent readers. Each file system has mechanisms to define permissions for operations and privileges for users. Similarly, GNS associates an ACL with each entry that controls creation and deletion operations that modify the namespace as well as the reads and lookups used for accessing it.

The meta-data required for the main tasks of the GNS consists of directory contents, junction targets, and ACLs. Other attributes might also be useful in some circumstances. Directory junctions delegate meta-data handling, especially for subsequent name resolution, to a physical file system. On the other hand, file junctions allow the GNS to perform authorization and provide meta-data.

A GNS virtual directory containing file junctions could replace a file-system directory and supplement file-system meta-data for those files. Authorization is shared between the GNS and the file system. The GNS controls lookup rights while the file system controls read and write access. Control of creation and deletion operations is held jointly; for example, creating a file requires inserting the file junction in the GNS virtual directory as well as creating the underlying file object on an appropriate server. This split responsibility can lead to consistency problems when access to both the GNS and the physical file system are unconstrained. One approach to controlling access is for the GNS to issue capabilities that must be used when communicating with the file server. A capability is a concept from operating-system security research in which authorization is encapsulated in a token whose holder is granted certain rights to a specific object or service. Capabilities allow the processes of authentication and authorization, which GNS is well-placed to perform, to be decoupled from the enforcement of access control by the file server. Such a file system could dispense with its own directories and most other meta-data and operate as an object store. This object file-system model is one possible use of the GNS; in other situations the GNS would only contain directory junctions to large file systems.

A similar circumstance arises when the GNS is used to store junctions pointing to files and subtrees within physical file systems. In this case, GNS virtual directories overlay file-system directories and either the file server or the GNS-aware client must merge them together. This arrangement may be useful because it allows unified administration of an organization's namespace both at a high level, where diverse file-system resources are linked together, and at the lower level where these resources are assembled from individual file-system trees. On the other hand, as mentioned above, this split responsibility can lead to consistency problems. Storing junctions as special objects in the file system, as in AFS or DFS, helps maintain consistency but can be difficult to manage (and may be infeasible for some file systems). The best solution for file-system-to-file-system junctions may be a judicious combination of these approaches.

Interface. The GNS has one or more interfaces used to access and modify the namespace. Operations refer to GNS objects by path name. The paths are relative to the GNS instance's single root. The subtree of objects is location independent and could be visible via multiple absolute paths in the global namespace. Namespace cycles cannot be prevented because individual instances are autonomous; there is no global authority to prevent, detect, or eliminate cycles. Therefore, GNS clients must be prepared to detect loops and handle such path names appropriately. The lookup operation can traverse multiple components, terminating at a virtual directory when the input path is exhausted. If traversal reaches a junction, the GNS client must apply the unused path components to the service indicated by the junction's target. It is also possible to perform lookups one component at a time. The strategy of using multiple components can save communication time (i.e., round trips), but provides the GNS with more information than it needs. This information leakage can have security implications.

The GNS supports a standard file-system-like interface but only implements naming and meta-data operations, such as lookup, creation, and setting attributes. Because readers of the GNS are likely to be far more numerous than writers, it may be reasonable to support only read-only operations using a file access interface and perform updates using an interface more suited to data management or administrative operations. This separation of read and write functions into different interfaces could have substantial benefits for system integration and maintenance while having minimal impact on users. A junction target in URL form can be accessed as an extended attribute or like a symbolic-link target. Other extra attributes used by GNS objects include ACLs. The names of directory entries are case-insensitive, which does not reduce their utility and improves compatibility with Microsoft Windows** and DNS interfaces.

Synchronization is needed to coordinate the reads and writes of the namespace. This simple statement belies the complexity of the subject because the requirements range across many different parts of the namespace. In addition, synchronization also serves the purpose of maintaining consistency for caches of namespace data. Near the root of the global namespace, readers vastly outnumber writers and performance requirements are stringent. On the other hand, closer to the leaves of the namespace, updates may be nearly as frequent as lookups, and the benefits of explicit synchronization mechanisms may be worth their expense. At the top of the namespace, a time-to-live[23] approach is perfectly adequate, while at the bottom, the trade-offs are more difficult, and one or more appropriate mechanisms need to be identified.

Operational aspects

This section presents design aspects of the GNS that must be considered in creating a GNS implementation.

Namespace organization. The global namespace can be divided into three sections: the root, the GNS, and physical file systems. Names in the root directory represent the origins of control for various namespace owners. The GNS defines global prefixes for files and subtrees. File systems export terminal files, subtrees, and file systems.

The root directory presents very significant control and performance problems, and it is constructed accordingly to balance the conflicting needs of uniformity and autonomy. The use of conventions in its definition provides sufficient global uniformity for practical purposes. Autonomy is preserved by composing the root directory locally from an ordered list of sources, such as a configuration file for local and domain-level names and DNS for global names. Local names might be used for scratch storage and private names for defining convenient aliases for frequently used data. Domain names might be used to address proprietary organizational information. Global names provide access to resources anywhere on the network. Using DNS records (e.g., type TXT or AFSDB) allows the use of existing name ownership to provide a well-defined structure for the root of the global namespace. By using an ordered list of sources for the root, local configuration can override names defined later in the list, while allowing most names to be the same everywhere. In effect, the root directory is a special GNS instance fabricated by the GNS client based on local configuration information.

Many GNS instances define the top levels of the global namespace. Each owner of a name in the GNS root (e.g., "ibm.com") operates an instance for direct path name lookups to appropriate resources. Entities with a decentralized internal structure may operate subsidiary GNS instances (e.g., "ibm.com/research" or "ibm.com/research/almaden"). This hierarchy of GNS instances may mirror an organization's DNS hierarchy or may follow a different logic. When the structures are compatible, the DNS names may replace one or more levels of the GNS hierarchy. Within an organization, the GNS is used to assemble a large namespace from the smaller namespaces of individual file servers. This can occur in cases of large granularity, with a few junctions to large monolithic file systems, or at a finer grain, with smaller subtrees being assembled into a structure relatively independent of servers and file systems.

Global path names eventually reach a junction to a file system. Within a file system, path name evaluation occurs normally, with file and directory access using a protocol appropriate for the hosting file server. File systems and protocols that support junctions can further extend the namespace by assisting the GNS in lacing together individual subtrees into a namespace shared by all users.

Each GNS instance and file server owns a portion of the namespace. As the owner, it can delegate, with junctions, responsibility for lower portions of the namespace. The result is that each part of the namespace has well-defined ownership. A consequence of this is that unique names for resources such as files can be defined easily.

Security. The GNS depends upon mutual authentication of both services and requestors to inform each end of a communication channel as to the identity, or principal, of the other party. Authentication of requestors is necessary for access control. The authentication of services to clients validates the results of the path name evaluation that maps from path name to file contents. This validation is necessary to assure that data consumers see the same data the data producers intended. This is the most basic contract a namespace service has with its users. However, this assurance depends on factors beyond the basic security of the authentication process.

Several services are involved in a path name traversal, and additional services are involved in mapping from logical to physical targets. During namespace traversal, junctions have a delegation function, and the creator of the junction is implicitly asking the client to extend his trust in the current service to the target service. When contacting the location service, trust is again required to ensure that the logical name and the physical names represent the same data. This trust can be conveyed in two ways. The principal name for the service may be an X.509 subject that is assembled from the service name, server name, and other characteristics and verified by a certificate signed by a certificate authority. Alternatively, the service's public key may be embedded in the junction when it is inserted into the namespace. The latter approach is safer, as it tightly binds the junction's path name to the service to be used for path names with that prefix. Ensuring that the junction keys re-

main valid, however, even as data moves or is replicated, is a significant challenge.

Authorization is an important aspect of GNS, and it should be compatible with that used by both file systems and administrative tools. While other approaches are possible, the ACL mechanism for describing and evaluating access control has long been used in both file systems and other secure applications. ACLs benefit greatly from the use of a group abstraction that bundles authentication principals into named collections. Groups allow ACLs to be more compact and expressive, and the resulting security is more reliable and transparent. Scalability requires that group creation and maintenance is something ordinary users can do. Otherwise, the administrative burden of creating and updating groups inhibits their use by making ACL usage needlessly cumbersome, to the detriment of the whole system's security. A global namespace requires an authentication and authorization system of similar extent and scope. More work remains to be done in this area.

Authorization is an important aspect of GNS, and it should be compatible with that used by both file systems and administrative tools.

The security of the entire file system is limited by its weakest link. Because the security of the namespace underpins the security of the entire system, it is crucial that the namespace use the strongest practical security. For example, if GSI were used for the namespace, file access protocols such as GridFTP, which also uses GSI, or NFSv4, which can utilize the GSSAPI (Generic Security Services Application Programming Interface) based on the same authentication mechanisms, would be able to perform securely.

On the other hand, NFSv2/v3 is notoriously insecure, and even now implementations based on a secure foundation are not widely available.[24] To mitigate the security weakness in environments where an insecure remote NFSv2/v3 server needs to be accessed across a wide-area network, a local NFSv2/v3 server can potentially be used to serve a replica or a proxy cache.[25] The replication or caching mechanism can use a secure protocol or channel, such as IPsec. The GNS allows such a replica or proxy cache to be recorded in the location service as described earlier in this paper, making it visible to the clients. When the local NFS server serves files out of a replica or cache, access control decisions may require user ID mapping if the user IDs on the local machine belong to a different user domain from that of the remote server. This does not solve the fundamental problem of NFSv2/v3's trust on client machines in presenting user IDs, but it does contain these issues to the local network. It is also possible to reduce the trust requirement in NFSv2/v3 clients through some out-of-band authentication mechanism.

For example, a custom logon utility program on a UNIX NFS client machine can be used to authenticate a user with a local UNIX user ID to an NFS server using Kerberos or some other challenge-response mechanism. The NFS server then maps the user ID from the client machine to the authenticated user identity on the server. Subsequent NFS request messages with this user ID are considered to be from the authenticated user, and an NFS client machine is only trusted to present user IDs recently authenticated from the client machine (i.e., within a configurable period of time, such as 30 hours). Note that this authentication mechanism is still vulnerable to IP spoofing, which can be a bigger threat in wide-area networks. For NFSv2/v3 clients, the namespace is not a panacea, but with a proxy or replica server, valuable improvements in security can be achieved by containing client accesses to local networks.

While GNS supports strong security, the namespace does not require it of file systems. Deploying a secure namespace that enables collaboration by using secure file systems may encourage improvements in other file systems and the authentication and authorization infrastructure they require. For example, it may foster wider and quicker adoption of GSI, NFSv4, and other secure components.

In general, security is a large topic that goes beyond authentication and authorization mechanisms to include communication systems, protocols, software integrity, physical protection, managerial issues, economic trade-offs, and even philosophic questions of what trust really means. Securely deploying a global namespace requires integration with many existing systems, which will necessarily be difficult and imperfect at first. Federating multiple file access protocols also has security implications because mis-

matches in authentication and authorization mechanisms must be bridged while still preserving necessary security. The namespace's global scope means that principals from one domain will need a representation in other domains, or else security must be disabled or sharing prohibited. Real collaboration requires a middle road. While this paper addresses the necessary naming component, there are significant issues to address in the security domain as well.

Interoperability interfaces. The namespace constructed by GNS needs to be made available via ordinary file-system interfaces in order to encourage adoption and ease of deployment in environments with existing file-system clients. The mapping of read-only namespace functions to file-system directory and meta-data operations is straightforward. A GNS wrapper for each file-system protocol could accomplish this mapping. The protocol support could be integral to the GNS instance or implemented as an external translator acting as a file-system server and as the GNS client of one or more GNS instances. A file-system server that exports only GNS data and no actual file data acts as a GNS proxy.

Some complications in mapping semantics between a file-system protocol and the GNS interface exist, such as simulating unsupported attributes, representing junctions, and providing synchronization. File-system attributes that do not have analogues in GNS can generally be synthesized or replaced with some appropriate constant value. However, some attributes, such as NFS permission bits, may be more difficult to handle. Synchronization of writers with readers is handled in file systems by using byte-range and session locks, and various cache consistency mechanisms, such as OpLocks,[26] time-to-live, and callbacks.[27] Approximate matching of these operations is usually acceptable. However, when more exact semantics are required, matching is difficult. More study is needed in this area.

Handling junctions is critical to efficient operation of the namespace, as they allow the namespace and the GNS proxy to get out of the data path once path name traversal is completed. The NFSv4, CIFS, and HTTP protocols support referrals, which can be used to return information about junction targets to these file-system clients. These clients expect physical target information, so proxies for these protocols will act as GNS clients and contact the location service to translate logical names to physical ones. The physical target, or list of targets in the case of replicated data, is translated by the proxy into the NFSv4 parameter "fs_location," CIFS parameter "DFS_REFERRAL," or HTTP redirect responses, as appropriate. The client can then access the file-system data directly without further contact with the GNS proxy. File-system clients for older NFS versions and FTP, however, will require proxies or one of the techniques described later in this paper to access data referred to by global path names.

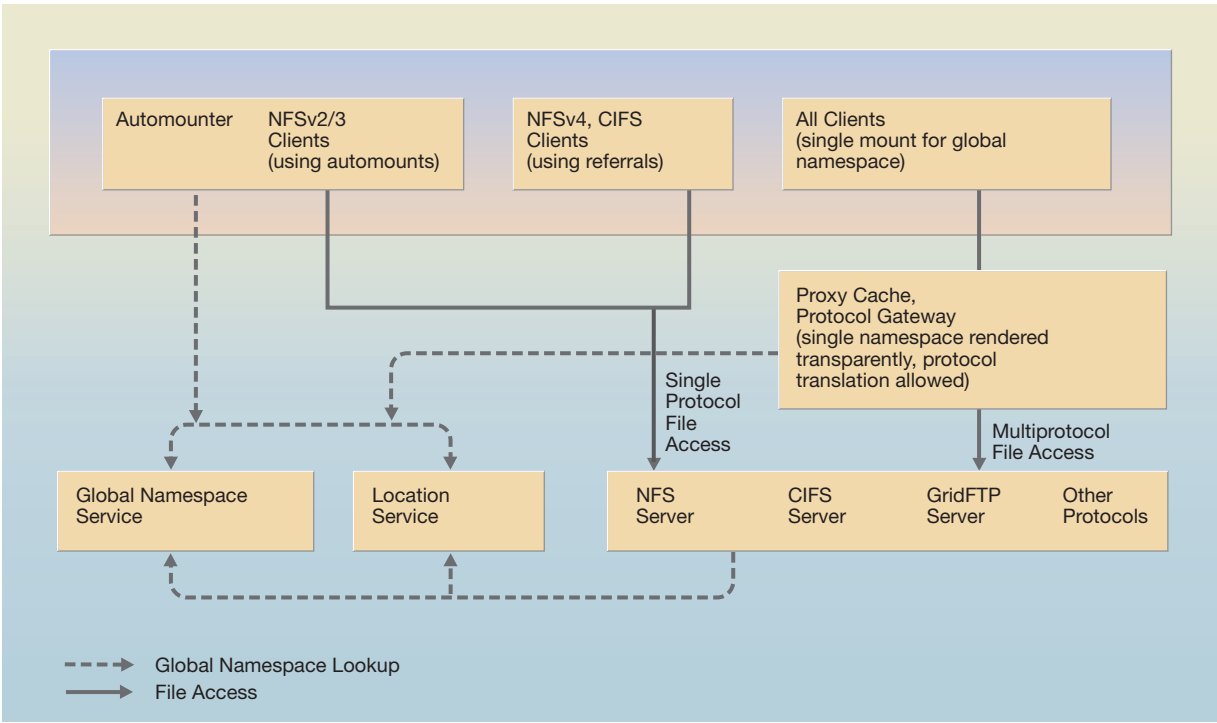
GNS clients. Path evaluation requires requests to GNS for name lookups and to a location service for physical targets. The data are then accessed using a protocol appropriate for the target. There are trade-offs between different approaches for implementing this combination of functionality. If clients and servers share a protocol supporting referrals, then using a GNS proxy is sensible. Multiprotocol servers, such as several NAS appliances, are increasingly common, and make this approach attractive. Another approach is possible for multiprotocol clients for which both Microsoft Windows and UNIX have some support. In the UNIX case, for example, the automounter could be used to translate GNS junctions into suitable parameters for the mount command and provide the client with access to the global namespace. On the other hand, joining clients to servers that have no protocols in common requires a file-system proxy such as a gateway or caching appliance. Finally, GNS support can be added to a client or application with a user or kernel library that can handle GNS lookups, location mappings, and multiprotocol file-system access. This approach can have advantages but at a greater cost than the other approaches.

Proxies behave as servers for one file-system protocol while operating as clients of the GNS, the location service, and the target's file-system protocol. Such proxies can provide protocol translation, enabling federation and shared caching that benefits performance.

In addition, they can support data replication and migration, allowing centralization of data management tasks, while hiding the complexities of these operations from the file-system clients. Thus, in addition to providing necessary support for some protocols, proxies may have utility for clients of all protocols. Figure 2 illustrates the paths used in GNS accesses.

Scalability and performance. In order to provide a distributed global namespace service, the GNS must

Figure 2 GNS access paths



employ a scalable and extensible architecture. The GNS proposes to leverage the existing infrastructure whenever possible by utilizing existing DNS and LDAP deployments. This approach offers the obvious advantages of resource management overhead reduction, resource investment consolidation, and reduced requirements for installation and deployment. Moreover, this approach enables the GNS to assume the scalability and performance characteristics exhibited by these globally deployed architectures.

GNS service discovery and representation of the top-level namespace, as mentioned in the subsection on namespace organization, proposes to employ standard DNS. DNS is a time-tested architecture with truly global scalability and proven performance. DNS has been scaling in operation, with considerable reliability, availability, and ample performance, since 1983. DNS currently services an estimated 301 million unique domain names worldwide, according to the Internet Domain Survey of January 2004,[28] acting as a worldwide, distributed database.

DNS clearly offers a reasonable solution for the top-level namespace of GNS. However, it is not suitable for the storage, management, and security aspects of GNS at lower levels of the service. As previously discussed, LDAP is a promising candidate for providing GNS services from a server perspective. Due to differences in the nature of their operation and application, it is expected that LDAP would not perform as well as DNS. LDAP does, however, provide a directory service which is optimized for read operations on large sets of data. This characteristic represents a desirable quality appropriate for GNS services. Furthermore, some LDAP implementations can store and serve over a billion objects. While different LDAP server implementations perform and scale differently, one study revealed that an LDAP server was capable of delivering 3175 message operations per second using 10 clients.[29] For that same server, the search rate with 10 clients yielded 3147 operations per second.

Extensibility is an even more notable aspect of LDAP scalability and performance. LDAP installations can begin small, both in terms of size and price, and offer virtually limitless expandability. This expandability can often be realized easily without retooling or application modification.

Examples

This section supplies concrete examples in the implementation of the GNS.

Usage scenarios. Imagine a loosely collaborative scientific enterprise, consisting of 10 large organizations, each with 50 file servers. Each organization has perhaps 250 local users, who mostly use the files on their local servers. Even within a single organization, our proposed uniform namespace helps those users navigate among the changing set of file servers. Each user mounts just one root of the namespace and is potentially connected to any of the files for which he or she is authorized; if the user moves to another workstation, the names of the files do not change. If a user wants to identify a file to be shared with a colleague at one of the other nine organizations, a simple file name will suffice—and that file name is exactly the same file name by which the user already identifies the file. The remote colleague's workstation also uses the same global namespace, identifying the correct file server at the first organization and the correct file within that server. Path names in the global namespace thus act as unique identifiers to content, similar to the capability provided by HTTP URLs in the World Wide Web. This enables a more efficient mode of information sharing by simple mechanisms such as e-mail, where a path name is exchanged in the body of the e-mail instead of using a bulky attachment. As long as the recipient of the e-mail has a suitable file-system client installed locally, such information sharing via file-system namespace references is simple. This is a common paradigm for information sharing in AFS and DFS environments where the sender and receiver both have AFS clients installed. Our heterogeneous global namespace solution enables the same capability with standard clients supporting NFS and CIFS protocols.

Suppose that one of these large organizations, in addition to its 250 local users, has 100 satellite users, each with small laboratories and requirements for local file storage. Small file servers can be installed in the small laboratories, and they can be included in the global namespace for that organization. Workstations in the lab can refer to files in the same lab, at the central site, or at the remote site, all simply by naming those files, and the file names will not change regardless of where the user sits.

In both of these examples, the global namespace forms an abstraction layer that hides the actual location of the files in addition to making the location of the user irrelevant. The namespace serves as a useful abstraction for additional file and file-system services: files can be replicated to multiple locations, but all the replicas can share a single name, and workstations can make use of the nearest replica to satisfy requests for the replicated file. File systems can be rehosted on different servers, and by updating the location server, the namespace that pointed to the old location can be made to point to the new location. Such operations allow for files to be migrated from old to new systems without disrupting the expectations of users, achieving the dream of the transparent upgrade. The global namespace we propose can accomplish all this.

The global namespace abstraction will also fit naturally with database-managed files. IBM's DB2* DataLinks[30] is one such mechanism. It uses a new SQL type—"DATALINKS"—the content of which is represented as a URL providing the path name and server name (or cell name for DFS). A DB2 server that supports the DataLinks mechanism interacts with a DataLinks file manager running on a file server to implement referential integrity and coordinated backup. A path name in our global namespace scheme could be included as a reference in a DataLinks field, though providing the enhanced services of the DataLinks file manager would require additional integration work. In a way, DataLinks provides a database view of files as data objects in the same way that GNS creates a hierarchical file-system view of files using file junctions. The ambitious approach taken by DataLinks addresses the problems of consistency between the GNS and file systems arising when file junctions are used, but at the cost of deploying a DataLinks file manager for each supported file system. File-system junctions represent a clearer delegation of authority presenting simpler consistency issues.

Deployment options. The following subsection details available alternatives for legacy distributed-file-system clients to utilize the global namespace.

GNS proxies. One approach enabling legacy distributed file-system clients to access the global namespace is via GNS proxies. A GNS proxy for a given file-system protocol uses GNS access methods to traverse the global namespace of virtual directories and junctions and present information to the legacy client (e.g., CIFS, NFSv2/v3, NFSv4) in a form that would be understandable to that client. Such a proxy would require GNS client functionality and access to the lo-

cation service for translating logical to physical targets, but it would not export ordinary file-system data. Virtual directories in the namespace that are discovered by the proxy by querying one or more GNS servers would be returned as directory objects to the legacy clients. On the other hand, a reference to a physical file system that is located as a leaf node of the global namespace could be returned to the legacy client either as a file system referral (if supported, e.g., for CIFS and NFSv4), or in a special representation that can be used by the legacy client, when enhanced appropriately, to mount the new file system. Various options to achieve the latter for NFSv2/v3 are described below.

The location service is conceptually an independent database accessible via some network protocol. The database stores mappings from logical names for files and file systems to the physical addresses at which they can be found. Each administrative domain maintains its own location service to track the location of its storage resources. RLS is one possible realization of the location service. While RLS was developed to support replica location, clearly the degenerate case of a single replica provides the desired location services as well. Another possibility is to use an LDAP server with a suitably designed schema. Indeed, the location service could be collocated with the domain GNS, and the same database engine could store both databases.

CIFS. Microsoft Corporation's MS/Dfs leverages the referral capability of CIFS (SMB) to support the creation of a global namespace. An MS/Dfs global namespace is constructed from various exported file systems (called shares) by an administrator, who creates MS/Dfs links that map a path in the MS/Dfs namespace to a separate file system by using the Universal Naming Convention (UNC) name of the form \\servername\sharename. MS/Dfs links are thus junctions that can be used to construct nested file systems. An MS/Dfs namespace can be traversed by an unmodified CIFS client using standard operations; when it traverses an MS/Dfs link, the MS/Dfs server returns a "PATH_NOT_COVERED" error code, the CIFS client returns to the server with a "GET_DFS_REFERRAL" request, and the server responds by returning a referral (or more than one if there are multiple replicas). Each referral structure provides the UNC name of the server to contact and the path name to start with. There is support in CIFS for referring a client to a file system exported by a different protocol (e.g., the Netware** example is well documented), but support for multiprotocol referrals has a critical dependency on the presence of the appropriate type of redirector in the accessing client.

This capability of CIFS can be very easily used by a dataless CIFS server in the role of a GNS proxy to provide a legacy CIFS client access to the multiprotocol global namespace constructed with GNS and location services. This server is dataless in the sense that it does not export local file-system shares. Such a GNS proxy (e.g., gnsCifsProxy1.raleigh.ibm.com) would be the root server for legacy CIFS clients, and it could export the root of the global namespace (/gfs) as some arbitrarily named share (e.g., "gnsroot"). Referring to the example of Figure 1, if a CIFS client referenced the UNC name \\gnsCifsProxy1.raleigh.ibm.com\gnsroot\ibm.com\SG\empData, it would get an error code of "PATH_NOT_COVERED," and on requesting an MS/Dfs referral, would get a response back to the CIFS server at Tucson, as the UNC name \\nas3.tucson.ibm.com\employeeData. In returning such a referral, the GNS proxy has to consult the location service because CIFS referrals have to be physical targets. The CIFS client can then access this CIFS server directly, bypassing the proxy and the entire GNS-based infrastructure. In this example, a naming convention for the CIFS-based physical target is assumed, namely, that the share name follows the server name. Note that a file system can be an MS/Dfs namespace containing nested file systems, and normal MS/Dfs referral mechanisms handle that portion of the namespace traversal.

NFS versions 2 and 3. The same global namespace can be rendered for use by legacy NFS clients (i.e., NFS versions 2 and 3). Assumptions built into NFS and UNIX introduce the restriction that only file system junctions are usable, and junctions to single files are not visible. Because legacy NFS lacks the concept of a file-system referral, it is more complicated to render the GNS for such clients, but there are at least three methods that can accomplish it by using the automounter. The use of the automounter can allow multiprotocol support in machines configured with appropriate client file systems. For example, a UNIX system with both NFS and Samba support could access file systems on both NFS and CIFS servers. All such methods presuppose that the file-system junctions in GNS can be known to the automounter. Possibilities include defining an executable map that queries the GNS, using local maps updated by scanning the GNS periodically, or having the automounter use a network data source that is backed by queries to the GNS. One automounter network source is LDAP holding NIS data as defined by RFC 2307.[31] Because

the GNS may be implemented using LDAP, some commonality may be possible, as with DNS, but the schemas are likely to differ such that LDAP automounter maps would be a transformed representation replicating the relevant GNS junctions.

Furthermore, for junctions from one file system to another, the assumption is that the NFS server can be modified to render the junctions in a specific way. This assumption is the basis under which we simultaneously export the same file system via multiple protocols, so that junctions are rendered one way via a CIFS server, another way via an NFSv2 or NFSv3 server, and yet another way via an NFSv4 server. On the server's file system, the junction might be represented as a symbolic link of a given format, such as a symbolic link to gns://abc in which abc is the logical name for the target file system. We then recognize this pattern in each of the file exporters for a given file system. Although it would be possible to record physical rather than logical names in the file systems so that one file system could refer to another without requiring a logical-to-physical evaluation, we prefer to use logical names in this role to allow for future changes to that logical-to-physical mapping without having to change the data in all the parent file systems.

The use of the automounter can allow multiprotocol support in machines configured with appropriate client file systems.

The first legacy rendering method uses a dataless GNS proxy for NFS, much like the one described for use with CIFS, so that names and directories wholly within the GNS are rendered as NFS names and directories, but in which file-system junctions are represented as empty directories. Simultaneously, these empty directories are represented to the automounter as trigger nodes, directing clients to mount NFS file systems at those points in the tree. Thus, as legacy NFS clients navigate their way through the global directory tree, the GNS proxy and the automounter cooperate to make the appropriate file systems appear as needed. In GNS, file systems themselves can also contain junctions to other file systems. The NFSv2 and NFSv3 servers can represent these junctions with empty directories, and the automounter can know about all the nested junctions as well, so that child file systems are mounted atop these empty directories by the time applications actually traverse through them.

The second legacy rendering replaces the dataless GNS proxy for NFS with what some automounter versions call *offset mounts.*[32] This allows the automounter to create the appropriate virtual directories and names up to the point of a file-system junction, which the automounter would cause the legacy client to mount as usual. Offset mounts are not widely supported, though. File-system-based junctions to other file systems could be rendered as above, with empty directories overlaid by the automounter.

The third legacy rendering uses the dataless GNS proxy for NFS, but renders junctions to logical file-system names not with a direct or trigger mount, but rather as a symbolic link to a well-known path-name pattern. For example, if a server file system contained a junction to the logical name "xyz," this could be rendered to the legacy NFS client as a symbolic link to /.gns/xyz. The reserved /.gns directory would itself be served by the automounter. Perhaps via an executable map, the location service would be consulted for the physical locations of file system xyz, and then the automounter would mount that file system atop /.gns/xyz. Client applications would follow the symbolic link, whereupon they would be led into the appropriate target file system. Thus the need for the automounter to know the exact locations of the file-system-to-file-system junctions before they are discovered in the server file systems is removed. As mentioned above, this rendering makes it most advisable to use logical names rather than physical ones in the file system's local representation of junctions.

NFS version 4. NFS version 4 is highly analogous to CIFS: the version 4 protocol defines referrals directly. NFSv4 servers can refer clients to other servers, though this is done with an optional part of the NFSv4 protocol. Clients that do not implement this part of the protocol can be integrated into the GNS in the same way as legacy NFSv2/v3 clients. Clients that do implement this part of the protocol work as CIFS does. Thus, a dataless NFSv4 server can be a proxy for the high-level GNS namespace, and that proxy can refer NFSv4 clients to file-system trees (though not to individual files).

An NFSv4 server can render the common junction representation (e.g., a symbolic link to gns://somename) by consulting the location service to determine the corresponding physical name (or names) for the referral, and then returning them to the client as fs_locations information. There have been suggestions that fs_locations should be extended to provide a logical name for evaluation by the client.[33]

Generally, with all these file-system renderings of the GNS, a given client will be able to see only that portion of the global namespace that is represented by file systems exported by protocols that are understood by that client. For example, if a CIFS-only client is navigating a global namespace made up of file systems exported only via CIFS or NFS, then those portions of the namespace exported only by NFS will be inaccessible from that client. However, it is straightforward to export file systems using more than one protocol at a time, as many NAS products do, and it is not uncommon for clients to be capable of navigating multiple protocols. (If nothing else, clients can often handle multiple versions of NFS interchangeably.)

Distributed SAN File System. The Distributed Storage Tank (DST) is an ongoing IBM research project that extends the IBM TotalStorage* SAN file system for heterogeneous and distributed file sharing, including sharing in wide area networks. The aim of this research project is to demonstrate how SAN (storage area network) file systems can be extended to allow one SAN file-system cluster to interoperate with other SAN file-system clusters as well as various other types of file storage systems, such as NAS appliances and data grids. The DST provides its clients with a unified file access interface to these distributed and heterogeneous data sources. File data from globally distributed sources are rendered with SAN speed through server-managed caching and replication.

To demonstrate global namespace management and rendering functionalities in the DST, the project has developed working prototypes of the GNS server—with an RDBMS back end, graphical service management tool, and file-system client. The file-system client currently functions as a component of the DST, rendering the GNS namespace via the SAN File System meta-data server, making the global namespace immediately available to all SAN- and gateway-connected clients without any modification to the clients. Although the GNS originated from the DST project, and the prototype implementation is demonstrated by the DST, its application is not limited to the DST. In fact, the namespace service architecture and naming model proposed by the GNS offer an interoperable namespace solution that is neutral to file-system protocols and technologies.

When a GNS client is deployed within a proxy, as is the case in our prototype implementation, a number of additional valuable features are realized. All clients of the file-system server or proxy experience a transparent view of the global namespace. The DST operates as a cache proxy, rendering the global namespace and caching the remote data represented within it. All referral connections and data transports are performed by the server or cache proxy, thereby relieving the file-system clients from the responsibilities of namespace service intelligence as well as protocol support for remote data sources. The prototype implementation demonstrates the feature of a cache proxy that supports multiple protocols, including NFS, FTP, and GridFTP, and so provides transparent access to global data. When data is scattered over heterogeneous sources, this is certainly an attractive capability.

The prototype implementation demonstrates the ability to mount the global namespace at the root of a file-system client's directory tree or file-system view. Once a user or application enters this special global namespace mount, all file-system requests are directed to the cluster meta-data server (MDS) component responsible for handling dynamic global namespace requests. This component employs the previously discussed GNS client, which is responsible for communicating with the namespace service and converting the results presented by the service into file-system-specific entities that have meaningful representation in the local/clustered file system. For example, a service response message that contains a number of virtual directory entries accompanied by respective attributes is converted into file-system directories with corresponding attributes. As a result, if the file-system client issues a directory listing command ("ls") at the root of the global namespace, the MDS handles the request by first querying the namespace service. This typically returns a number of virtual directories (at the root level). The virtual directories are converted to file-system directories. After the newly created directory entries are stored in the MDS, their listing is returned to the requesting client and appears to the file-system client as standard directories. The more exciting example is when a file system client lists the contents of a directory that is actually a namespace junction to a remote data source. In this case, the MDS com-

ponent responsible for handling global namespace requests identifies that the result of its GNS query represents a junction to a remote file system. A local file system container is dynamically constructed and commissioned to serve as a cache of the remote file system, and a connection to the remote file server is established. The meta-data of the remote directory is cached in the corresponding local container and subdirectory, and file entries are created in it and are then returned to the requesting file-system client. With data read requests, the same process is followed, with the addition of data transfer taking place before returning to the file-system client.

Deployment comparison. It is instructive to compare the facilities that may be rendered with each of the protocols or systems described in the previous section.

CIFS can render the parent GNS space as well as file-system-to-file-system junctions. The targets of CIFS referrals can in principle be file systems exported over a variety of protocols, though it is not clear how well this works in practice. GNS clients are used both in the CIFS server and in the dataless proxy that renders the GNS.

NFSv2 and v3 require substantial help from the automounter to navigate both the parent GNS space and any file-system-to-file-system junctions. Automounter mounts can be a choice of NFS protocol versions. The NFS automounter can mount only file systems, not individual files. GNS clients are used both in the automounter data source and in the dataless proxy that renders the GNS.

NFSv4 supports the same style of interaction that CIFS does. The representation of referrals in version 4 points only to other NFSv4 servers, but subsequent revisions might extend that to enable the description of referrals to a variety of file-server protocols.

Distributed SAN File System is capable of handling all GNS features and facilities, including both file and file-system junctions. It renders GNS as a file-system client component of DST within a proxy and imposes the requirement of using the proxy as an intermediary, which may or may not be convenient in a given deployment and may or may not provide adequate consistency assurances for the task at hand. But the rendering is complete, and the proxy is the only mechanism that ensures the ability to traverse a GNS made up of heterogeneous protocols, in which different file systems are accessible only through different protocols (CIFS, NFSv2/v3, and NFSv4).

Conclusion

We have defined a global, uniform, hierarchical namespace service that transparently links file data to users and provides crucial functionality to enable a file-system grid. A suite of strategies has been presented for accessing file data residing on diverse sources using commonly available clients. This federated approach provides the promise of a universal file system that looks the same to all users everywhere, providing a boon to collaboration. The namespace also allows decoupling logical and physical resources, which improves the ability to manage the increasingly complicated information infrastructure.

This federated approach provides the promise of a universal file system that looks the same to all users everywhere.

Future work includes efforts to reduce more of these ideas to practice and to deploy them more widely. This will provide valuable experience about what combination of approaches works best in a given environment. In addition, we hope to extend the namespace to include nonfile data such as databases and real-time feeds. The effort needed to join data currently fragmented into numerous isolated systems is large, but we have shown a few manageable steps that can start the process. Once the process begins, the tremendous benefits to usability and administration will provide the impetus necessary to complete the unification.

Acknowledgments

Carl Burnett reviewed an early version of this paper, and we received helpful comments from several anonymous reviewers.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Avaki Corporation, The University of Chicago, The Open Group, Novell, Inc., or Microsoft Corporation.

Cited references and notes

1. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications* **15,** No. 3 (2001), http://www.globus.org/research/papers/anatomy.pdf.
2. *Security Documentation*, The Globus Alliance, http://www.globus.org/security/.
3. W. Allcock, Editor, *GridFTP: Protocol Extensions to FTP for the Grid*, Global Grid Forum Draft Standard (April 2003), http://www-isd.fnal.gov/gridftp-wg/draft/GridFTPRev2.pdf.
4. A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripenu, B. Schwartzkopf, H. Stocking, K. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services," *Proceedings of the IEEE/ACM Supercomputing SC2002 Conference*; IEEE Computer Society Press, Washington, D.C. (November 2002), http://www.isi.edu/~annc/rls/chervenakFinalSC2002.pdf.
5. A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen, and R. Olschanowsky, "Storage Resource Broker—Managing Distributed Data in a Grid," *Computer Society of India Journal*, Special Issue on SAN **33,** No. 4, 42–54 (October 2003), http://www.npaci.edu/DICE/Pubs/CSI-paper-sent.doc.
6. A. Grimshaw, "Enterprise Data Grids," Tutorial presentation at Global Grid Forum 7 (GGF7), Tokyo, Japan (March 2003), http://www.gridforum.org/Meetings/ggf7/tut1.htm.
7. L. Luan and T. Anderson, "Grid Namespace for Files," (June 2003), http://www.gridforum.org/Meetings/ggf10/GGF10%20Documents/GridNamespaceforFiles.pdf.
8. The term "file system" has two distinct meanings. One refers to a body of software, concepts, and interfaces and the other refers to a collection of files, directories and other objects linked in a hierarchy to a root directory. The latter meaning is sometimes given other names, such as a CIFS "share" or AFS "volume," but in typical usage, also followed in this paper, "file system" is used for both. Sometimes for clarification we will say "subtree" of "file-system tree" to make the meaning explicit.
9. A path name is a sequence of components, separated by "/" characters - for example, /usr/joe/statement.txt. Each pathname component represents a lookup in a directory, so to interpret the path name of this example, the key "usr" is looked up in the directory "/", and presumably leads to another directory. The key "joe" is looked up in that directory, leading to yet another directory, in which "statement.txt" is looked up. File path names are looked up at GNS instances. In some cases, a GNS can fully resolve a path name and provide the client a reference to the file system or file the path name represents. In other cases, a GNS instance may only be able to resolve a prefix of a path name provided by the client and provide the client a reference to another GNS instance where the client should do the lookups to resolve the remaining portion of the path name.
10. *Automounter* is a mechanism for managing UNIX** file systems that automates the process of mounting them when they are referenced and unmounting them when they are no longer being used. *Mounting* attaches the root of a file system to a directory in the local file-system namespace according to parameters that describe how the file system can be accessed (e.g., a disk device or an NFS server name) and related options. The file systems to be managed by automounter can be specified in a number of ways, including configuration files and helper programs that provide the local path name where the file system is to appear and the parameters necessary to mount it..
11. *Common Internet File System Technical Reference*, Storage Networking Industry Organization, http://www.snia.org/tech_activities/CIFS/.
12. S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network File System (NFS) Version 4 Protocol", RFC3530, Internet Engineering Task Force (April 2003), http://www.ietf.org/rfc/rfc3530.txt.
13. *OpenAFS*, http://www.openafs.org/.
14. *DFS Administration Guide*, IBM Corporation (2000), http://www.ibm.com/software/stormgmt/dfs/library/docs/31manuals/AdminGd/duagd002.htm.
15. J. Apgar, A. Grimshaw, S. Harris, M. Humphrey, and A. Nguyen-Tuong, "Secure Grid Naming Protocol (SGNP): Draft Specification for Review and Comment," (2002) http://www.gridforum.org/Meetings/ggf4/bofs/SGNP%20-%20GWD-R%202002.02.05.pdf.
16. *Community Authorization Service (CAS) GT3 Notes,* http://www.globus.org/security/CAS/GT3/.
17. R. Elz and P. Hethmon, *Extensions to FTP*, Internet Engineering Task Force, FTPEXT Working Group Internet Draft (September 2002), http://www.ietf.org/internet-drafts/draft-ietf-ftpext-mlst-16.txt.
18. The Unix function chmod() changes the access rights of a file system object using the simple 12-bit UNIX permission model.
19. R. Pike, D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom, "The Use of Name Spaces in Plan 9," *Operating Systems Review* **27,** No. 2, 72–76 (April 1993), http://www.cs.bell-labs.com/sys/doc/names.html.
20. A. Mirtchovski, R. Simmonds, and R. Minnich, *Plan 9—An Integrated Approach to Grid Computing*, The 9Grid, http://www.9grid.net/papers/ipdps-04/plan9-grid.pdf.
21. H. Johner, L. Brown, F.-S. Hinner, W. Reis, and J. Westman, *Understanding LDAP—Design and Implementation*, IBM Redbook SG24-4986-00 (June 1998), http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg244986.html.
22. *Federated Naming: The XFN Specification*, The Open Group (July 1995), http://www.opengroup.org/public/pubs/catalog/c403.htm.
23. "Time-to-live" is a simple cache consistency mechanism used by DNS, HTTP, and many other systems where the server tells the client how long the returned information can be expected to remain valid. The client can use and cache the information for this period and then should request the data again in case it has changed.
24. Stronger authentication support such as Kerberos may start to appear more in vendors' products as a byproduct of their NFSv4 development, where Kerberos-based authentication support is mandatory.
25. The file-system-proxy cache function is not available in typical file-system products today. An emergence of a global namespace for files and file-system grids, however, will likely encourage file server vendors to develop file-system-proxy cache products.
26. The CIFS/SMB protocol uses "OpLocks" to tell a client that it may perform certain operations on a file or directory safely without contacting the server for each one. This represents a temporary delegation of authority from the server to the client allowing it to cache data, which may be "recalled" (i.e., broken) by the server when it can no longer assure the safety of the client's independent operation.
27. "Callback" is a term used by AFS to designate the recall of the server's extension of caching rights to the client. The call-

back message from the server notifies the client that a file or directory may have been modified by another client and it should invalidate its cache of that object's data.
28. Internet Systems Consortium, Inc., http://www.isc.org/.
29. J. Snyder, "Sizing up LDAP servers," *Network World* (05/15/00), http://www.nwfusion.com/reviews/2000/0515rev2.html.
30. R. Michel, A. Arora, K. Crooks, A. Lalla, and D. Shields, *Data Links: Managing Files Using DB2*, IBM Redbook SG24-6280-00 (December 2001), http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246280.html.
31. L. Howard, *An Approach for Using LDAP as a Network Information Service*, Internet Engineering Task Force, RFC2307 (March 1998), http://www.ietf.org/rfc/rfc2307.txt.
32. B. Callaghan, *NFS Illustrated*, Addison-Wesley, Reading, MA (2000).
33. J. Zhang and P. Honeyman, *Naming, Migration, and Replication in NFSv4*, University of Michigan CITI (Center for Information Technology Integration) Technical Report (2003), http://www.citi.umich.edu/techreports/reports/citi-tr-03-2.pdf.

Accepted for publication May 27, 2004.

Ted Anderson *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (ota@almaden.ibm.com).* Mr. Anderson is a member of the Almaden Research Center (ARC) and works in the Distributed Storage Tank group, whose aim is providing consistent and transparent access to remote data for users of the SAN File System and other file systems. Before joining ARC he worked at Transarc, both before and after its acquisition by IBM, on Episode, a fast-restart physical file system providing fileset support, as well as other aspects of DFS/DCE. He was also a developer of AFS, both at Transarc and at Carnegie Mellon University (CMU). Before coming to CMU in 1988, he worked on the S-1 project at Lawrence Livermore National Laboratory after obtaining a bachelor's degree from the Massachusetts Institute of Technology in 1979.

Leo Luan *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (luan@almaden.ibm.com).* Dr. Luan is a research staff member at the Almaden Research Center and manager of the Grid and IDC (Internet Data Center) Storage Systems department. He received a B.S.E.E. degree from National Tsinghua University, Hsin-Chu, Taiwan, ROC, in 1982, and a Ph.D. degree in electrical engineering from the University of Maryland in 1990. In 1990, Dr. Luan started consulting with the IBM Federal Systems Division in distributed computing security, and led the design and implementation of the audit subsystem for OSF (Open Software Foundation) DCE. He joined the IBM Almaden Research Center in 1994 where he has been working on storage systems. He developed ADSM (now TSM) solutions for AFS and DFS backup and pioneered the development of the LAN-free backup solution for TSM. Dr. Luan also led a research project called UFiler to explore Internet-based universal file-storage solutions. His current research focus includes distributed file systems and storage solutions for grid computing. He leads the Distributed Storage Tank project, which aims at building key technology components for heterogeneous distributed file systems and demonstrating their application in wide-area and grid file sharing.

Craig Everhart *IBM Systems and Technology Group, 500 Park Offices Drive, Highway 54, Research Triangle Park, NC 27709 (craigev@us.ibm.com).* Dr. Everhart is a Senior Technical Staff Member currently working in the development of the SAN File System product, as well as focusing on strategic file system and storage issues. Prior to joining IBM with the completion of the acquisition of Transarc, he was the chief architect for DCE/DFS, among other roles there. Before joining Transarc, he designed and built advanced e-mail systems at the Information Technology Center at Carnegie Mellon University, from which he received his Ph.D. degree in 1985.

Manuel Pereira *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (mvp@almaden.ibm.com).* Mr. Pereira is a member of the Almaden Research Center and works in the Distributed Storage Tank (DST) group as a major contributor to its design and development. He has developed a functional GNS prototype implementation consisting of a GNS server, SAN-File-System-integrated GNS client, and GNS administrative client. He was the technical mentor of the IBM Extreme Blue project at Almaden in Spring 2003 where he led the exploration of grid enablement of DST. In addition to the global namespace features of DST, he has been instrumental in the development of DST's consistency management framework. Prior to his involvement in DST, Manuel was the principal developer of the UFiler project, where he developed an API and corresponding libraries that enable secure Java/Web-based access to and management of AFS; this work has since been adopted by the Stonehenge project in Germany, and it will be included in their product. His AFS development experience has included authorship in the OpenAFS community. Before joining ARC he was the chief developer and Web application architect for the IBM Edge extranet application, which served IBM's largest OEM and reseller customers, channeling over $2 billion of revenue in its first full year of deployment. He joined IBM after obtaining a bachelor's degree from the California State University at Fresno in 1998.

Ronnie Sarkar *IBM Systems and Technology Group, 500 Park Offices Drive, Highway 54, Research Triangle Park, NC 27709 (sarkar@us.ibm.com).* Dr. Sarkar has been with IBM for 15 years. He is currently a Senior Technical Staff Member in the Storage Software Architecture department of the Systems and Technology Group, where he has been focusing on the SAN File System, policy-based storage management, and parallel NAS issues. Prior to this position, he was in the Software Group (AIM division) where he led development teams building Web-based host integration products. He spent the first seven years in the Networking Software Division working on communication protocols and protocol converters. Before joining IBM, he spent three years in an Indian startup company building PC compilers, while getting his bachelor's degree in electrical engineering and his master's degree in computer science from the Indian Institute of Technology, and his Ph.D. degree in computer science from Ohio State University in 1989.

Jane Xu *IBM Systems and Technology Group, 5600 Cottle Road, San Jose, California 95193 (jxu@us.ibm.com).* Dr. Xu is a Senior Technical Staff Member in the Storage Software Architecture department in the Systems and Technology Group. She is also a member of the IBM Academy of Technology. Dr. Xu is currently responsible for policy-based life-cycle management of the SAN File System, grid storage, and file-system strategy for on demand systems. She actively participates in the Global Grid Forum (GGF), representing IBM in the data area and serving as a co-chair of the grid-file-system-service working group. She is a mem-

ber of the IBM on demand design council and leads the file-system working group. Dr. Xu joined IBM Storage Systems division in 1990 working on storage architecture. Joining the IBM Software Group, she delivered the DB2www, Net.Data, and DB2 XML Extender products as the technical leader. After a one-year assignment to assist the Vice President of Research in 2001, she worked on advanced technology for storage software. Dr. Xu has received several Outstanding Technical Achievement Awards. She holds eleven issued patents and has another eleven pending. Dr. Xu received her Ph.D. degree in computer science from the University of Southern California in 1990, and her B.S. and M.S. degrees in computer science from the University of Hawaii at Manoa, in 1985 and 1986 respectively.