

IBM Scale out File Services: Reinventing network-attached storage

S. Oehme
J. Deicke
J.-P. Akelbein
R. Sahlberg
A. Tridgell
R. L. Haskin

This paper focuses on the challenges of creating a scalable network-attached storage (NAS) filer with hundreds of nodes on top of the IBM General Parallel File System™ (GPFS™). The filer is created as a multiheaded data exporter over Ethernet and uses the Common Internet File System (CIFS) as implemented by the Samba open-source project and a Network File System (NFS) protocol server. Special focus is given to ensure the consistency of data in an efficient manner for applications that have complex access patterns from a large number of nodes. Also considered are the challenges of providing an extremely high-performance facility for exporting data in parallel and the demands of storing very large amounts of data. Introducing new aspects of storage management—such as storage pools and file sets as they are implemented in current file systems—into NAS appliances enables the management of large amounts of data in a single namespace.

Introduction

The efficient management of data is an ongoing struggle between access and scalability. Providing access to file-level data (i.e., those files associated with individual documents, multimedia content, databases, and other applications) becomes more difficult as the number of users with access and the amount of data stored both grow. Also, achieving the scalability needed to respond to the growth of data volume typically results in higher hardware and software costs and greater management challenges.

Network-attached storage (NAS) solutions provide simplicity, manageability, and access, but until now, they lacked a single capability that has kept them from playing a role beyond the departmental level: scalability. Emerging NAS systems and managed storage services that are powered by clustered file system and parallel NAS technologies offer compelling business value to organizations because of their broad spectrum of applicability, competitive price and performance, and scalability to meet growing and evolving data demands.

The remainder of this paper is structured as follows. First, we discuss existing NAS concepts and the NAS

technical history. We then explain the basic technologies that are integrated in IBM Scale out File Services (SoFS). SoFS is a new solution that combines a state-of-the-art clustered file system with highly scalable protocol exporters, which include Samba [1, 2] for the Common Internet File System (CIFS) protocol [3], the Network File System (NFS) [4–6], and other protocols such as File Transfer Protocol [7], Rsync [8], and Hypertext Transfer Protocol [9]. The section “Paradigm shift from scale-up to scale-out” describes the unique differentiators of SoFS, that is, the central concepts of SoFS locking and SoFS end-to-end management. The status of the implementation and some promising performance characteristics are then discussed, and a summary follows.

Existing NAS concepts

Advanced requirements have emerged from various industries for NAS filers, and to some degree, the management and scalability problems with current NAS solutions became industry specific, as illustrated by the following two examples:

©Copyright 2008 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied by any means or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8646/08/\$5.00 © 2008 IBM

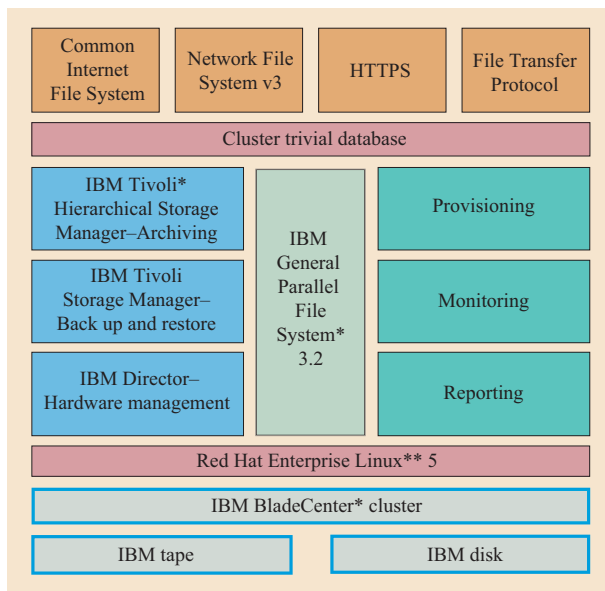


Figure 1

SoFS component view (HTTPS: Hypertext Transfer Protocol over Secure Socket Layer).

1. *Communications and digital media*—The amount of data and the number of objects that need to be stored are rapidly increasing with the move from analog to digital content [10]. Today, most customers in this sector need to read and write data with bandwidths beyond 10 Gb/s, and it is common that they have to achieve this read and write performance from or to a single file from multiple sources or destinations, respectively. Even the most advanced single filer is not able to handle this workload well because its single-file performance is limited by the number of disks available for distributing data blocks and ultimately by the number of data storage devices at the back end. To meet such performance requirements, a solution must be able to spread a single file across as many disks as possible and as many controllers and nodes as possible because a single device is simply not able to handle such a high I/O (input/output) workload. Alternatively, customers in this sector might need to transfer tens of millions of small objects that could potentially fit on a single storage device. In such a scenario, the filer architecture shows fundamental bottlenecks and several combined filers are needed to meet the throughput requirements. Unfortunately, such an approach introduces difficulties with respect to backup performance and maintainability of such a

huge number of objects. In general, this huge number of objects will be divided among various namespaces on several filers, thereby introducing management overhead for the operations team and making less than optimal use of the available total system resources because of an unbalanced I/O demand across the filers.

2. *Pharmaceuticals*—Most large-scale pharmaceutical customers have a requirement to store a wide variety of data, sometimes for 30 years or longer to comply with various government regulations [11]. Today, the data is usually made up of hundreds of millions of files spread across multiple filers. The reason for such a solution design as SoFS is not a lack of individual filer capacity, but to manage this number of files effectively in a single-filer system, to integrate it with long-term archival resources, to be prepared for disasters, and to meet other requirements, such as data searches.

As these two brief industry examples illustrate, a paradigm shift in the NAS industry is required. On the one hand, future file-serving solutions need to break the scalability limitations of existing NAS systems, and designers must declare scalability and performance as being central to their objectives. On the other hand, the current mature level of usability and manageability must not be sacrificed when loosening the strong integration provided by NAS filers today. What is required is a fresh look at optimizing NAS systems that takes into account the following system characteristics:

- Usability, including installability and maintainability.
- Scalability and performance.
- Total cost of ownership (i.e., acquisition and maintenance cost).
- Interoperability (i.e., integration in diverse customer environments).

Brief history of NAS

A file system is basically a method for storing and organizing computer files and the data they contain to make it easy to find and access. A network file system is any computer file system that supports the sharing of files as persistent storage over a network link. The first file servers were developed in the 1970s, and in 1985 Sun Microsystems created the NFS, which became the first widely used distributed file system.

It took another 7 years until NetApp (formerly Network Appliance, Inc.) shipped its first product in 1992, a system developed to meet the specific market requirements of data-intensive network environments.

It was customized to one specific task: giving read and write access to a network-attached optimized file server, or *filer*. NetApp founder and executive vice president, David Hitz, said in an interview with the *San Francisco Chronicle* [12], “We built a box to do just one thing, and it goes a hell of a lot faster than a general purpose server.” He was right, and NetApp opened up a new market with its filers—a market that soon became known as the *NAS market*.

Ever since, the NetApp basic approach has been a very efficient combination of general-purpose hardware and a proprietary software stack based on that hardware, which has given NetApp a competitive cost structure for its products (inexpensive hardware) and a competitive technology edge (sophisticated software). The concept of a filer describes a file-based storage device, possibly directly connected to the Internet or a company intranet, that provides fast and easy access to digital information from a variety of source applications. Such a system enables employees to share any kind of document in any format. Filers have been successfully deployed in a variety of other industries; petroleum companies started to use large filers to store seismic data, Internet service providers such as Yahoo! began to use filer systems to store customer e-mail messages, and filers were even used to store the special effects for the blockbuster movie *Titanic* [13].

Other companies, such as EMC and several smaller companies, entered the NAS market [14], and NAS innovation continued to evolve, again primarily led by NetApp. Over the years, many new features have been introduced that improved manageability, made installation easier, and added advanced features such as for disaster recovery or for replication. As NAS products improved in ease of use, the NAS business has flourished.

However, there remained a conceptual limitation on scaling in capacity and the number of file system objects a single NAS filer can handle. Most NAS systems gain performance and scalability improvements from generation to generation simply by leveraging newer processor and I/O technology. This product strategy was well suited to the needs of many customers and helped sustain the growth of the NAS market to the multibillion dollar levels earlier in this decade.

In 1999, an alternative technology was introduced: A small company, Spinnaker Network Solutions, was founded to build a new type of file-serving solution inspired by the Andrew File System (AFS*), one of the major distributed file systems [15, 16]. The solution offered by Spinnaker was arguably not as well integrated as the dominant products available at the time or as user friendly, but it did have outstanding scalability derived from AFS technology.

In November 2003, NetApp acquired Spinnaker [17] and began to integrate the distributed approach into its systems.

Paradigm shift from scale-up to scale-out

If a traditional filer is reaching a scalability or capacity limit, customers upgrade to the next higher filer level until they reach the highest available system—this is called the *scale-up approach*. Instead of scaling up in a single high-end enterprise-class server node or building failover clusters to provide high availability for the filers, the system described below leverages software functions to build a global namespace and exposes to the end user a single virtual cluster node with very high scalability. The system provides file services by implementing a *scale-out approach*, hence the IBM Scale out File Services product name.

SoFS uses the wide-striping capability of IBM General Parallel File System (GPFS*) [18] to spread the blocks of a single file across as many nodes as possible to combine the streaming performance of many midrange back-end storage devices (**Figure 1**). By doing so, a bandwidth of multiple gigabytes per second is possible because the I/O performance of many disks can be combined instead of storing data only to a small RAID (Redundant Arrays of Independent Disks), as filers do today. By fully exploiting the capabilities of GPFS, the integration of a policy-based life-cycle management interface and the possibility to place files in a directory that spans multiple independent storage classes creates a single file system with multiple storage classes—each with its own performance, availability, and cost criteria—on a file basis, not on a filer basis, as has been done in the past.

SoFS utilizes and expands the IBM Tivoli Storage Manager (TSM) and Tivoli Hierarchical Storage Manager (HSM) products to integrate Tape as an additional storage class inside the scale-out NAS. This allows transparent migration of data between disk and tape storage and provides end-to-end life-cycle management of the data.

To provide a single global namespace to the user, SoFS uses virtualization and redirection technologies available within GPFS clusters. Each SoFS cluster node has internal access to all data blocks simultaneously and is able to assume the responsibility for each of the other nodes in a cluster. This provides the means for a transparent, nondisruptive protocol failover for client applications. As none of the existing cluster technologies provides a transparent failover of network-based file systems, SoFS introduces an extremely scalable cluster suite dealing with the specifics of file systems to provide a high degree of semantic correctness while being able to scale to very high bandwidths.

IBM GPFS

At the heart of SoFS is the IBM GPFS cluster file system. GPFS provides the nodes of the SoFS cluster with a single-system view of the file system. GPFS is based on the shared-disk model: All nodes share read and write access to a group of block devices (e.g., RAID volumes accessed by means of a Fibre Channel switch). GPFS implements a POSIX** (Portable Operating System Interface Standard) file system on these volumes such that each SoFS (NFS or CIFS) server sees a fully coherent view of the shared file system. The shared disks contain both data and metadata.

Unlike most cluster file systems, GPFS has no distinct metadata server; the GPFS client and SoFS server nodes manipulate metadata directly synchronized by the GPFS lock manager. The GPFS lock manager performs two basic functions in SoFS: It manages cache consistency locks in order to synchronize concurrent updates by multiple nodes and it manages NFS locks in a central place that allows NFS clients to reacquire their locks during recovery from a server node failure. GPFS locks grant rights to the object that they cover (e.g., on a file or byte-range level) until they are relinquished by the holder or revoked by the lock manager. These long-lived locks (or tokens) enable the nodes to prefetch and write behind, as well as to cache data and metadata with minimal interaction with the lock manager.

GPFS has a number of features that support high availability and fault tolerance. Normally, fault tolerance for storage is handled by hardware (redundant paths to dual-controller RAID), but GPFS contains its own block replication in the file system that can be used instead of (or in addition to) RAID. To handle node failure, GPFS stores a journal of all metadata updates to a shared disk. When a node fails (as perceived by a quorum of surviving nodes), it is first fenced from the shared disks to prevent it from writing. Then one of the surviving nodes replays the journal of the failed node to redo all partially completed metadata updates. Finally, its locks are released, allowing other nodes to continue operation. There are only a few centralized GPFS services: configuration, locking, quorum, quota, and a few others. They fail transparently to other nodes. All management commands (such as adding and removing nodes from the cluster, adding and removing storage, and rebalancing data across new storage) are performed online without having to take down the cluster. Long-running commands (e.g., rebalance) can be restarted. GPFS supports rolling upgrades so new code can be installed one node at a time without taking down the cluster.

GPFS is designed to scale to very large file systems. A single mount point can comprise up to 2,000 volumes, each of which can be, for example, a RAID parity group. In 2007, the largest GPFS file system in the field was 2

petabytes. GPFS uses extendable hashing for directories, so that the performance of lookup scales even for large numbers of files. The GPFS block size is typically a full RAID parity stripe, which optimizes performance for sequential and quasi-sequential workloads, such as digital media and high-performance computing. To optimize the use of storage for metadata, small files, and the last partial block of large files, GPFS packs multiple such fragments into full blocks as required.

Samba CIFS

The CIFS file server component of SoFS is based on the open-source Samba project. There have been many efforts over the years to make Samba work well in a clustered environment, but they all suffered from poor performance, incoherent CIFS protocol semantics, or both [19].

These issues have been solved for SoFS by taking a new approach to clustering access for the CIFS protocol. To understand this new approach, we start with a description of the main architectural features of Samba and the problems faced by previous clustering attempts.

Samba is a CIFS service designed to provide accurate CIFS (and thus Microsoft Windows** operating system) semantics in an environment based on a POSIX file system such as GPFS. The core function provided by Samba is semantic mapping, that is, mapping the rich semantics expected by Windows operating system-based clients when talking to CIFS file servers to the rather less rich semantics provided by POSIX file systems. To enable this semantic mapping, Samba employs a number of internal metadata databases that store the additional data associated with the currently active resources used to map between the required CIFS semantics and the POSIX semantics. A typical Samba installation will use approximately 12 databases, each dedicated to one of the aspects of the required semantic mapping.

An example of this metadata is the open file database (known internally in Samba as `opendb`). This database holds the additional information from file open requests in the CIFS protocol that cannot be represented in POSIX. For example, the CIFS protocol `NTCreateX` operation (which is the most commonly used file open operation in modern Windows operating system-based clients) takes 13 parameters, whereas the POSIX open operation takes three parameters. The additional parameters in `NTCreateX` control such things as whether another client opening the same file with another set of parameters should be allowed to open the file at the same time. This additional information is stored in `opendb` and referred to in any future open operations before the POSIX open operation is tried, which effectively allows Samba to provide accurate CIFS semantics on top of a POSIX file system.

Nonclustered versions of Samba use a very lightweight memory mapped database called a *trivial database* (TDB) to store this metadata. A TDB is a simple key and value database that allows for extremely fast shared access (multireader, multiwriter) to database contents between processes on a POSIX system. Each CIFS client connection in Samba is served by a single `smbd` daemon process, and each of these processes communicates with each other by means of the TDB databases. This architecture has proved itself to be robust and very scalable.

At first glance it may seem that clustering CIFS could therefore be achieved very simply by storing these databases on a clustered coherent distributed file system such as IBM GPFS. As GPFS provides POSIX semantics, the cluster Samba should operate with only very minor modifications as long as the TDB databases are placed on shared storage. This approach is the basis for several failed attempts at clustering Samba. It does work, but it is unacceptably slow. A typical performance result on a widely used benchmark (NBENCH) with this approach is that the performance with a two-node cluster is between 10 and 100 times worse than that with a single node as tested. This is not the sort of result that excites users about scalability. The problem is that the costs of the coherence protocols in the cluster file system are far too high for the I/O patterns needed by a TDB. For a long time the developers of a number of cluster file systems tried to reduce these overheads, but with very little success. The approach to clustering in Samba needed to be rethought.

It is fundamental to cluster file systems that they not lose data. They operate under the constraint that once they have acknowledged a write, the data will not be lost—even if the node on which the write has occurred completely fails. Therefore, a cluster file system must either write the data to shared storage or send the data to every node in the cluster (as $N - 1$ nodes could fail at any time). This is not true for metadata in Samba, as it can afford to lose data under some circumstances without the loss of any information necessary to perform accurate semantic mapping from CIFS to POSIX.

For example, in the `opendb` database discussed above, Samba stores a piece of data per currently open handle for that file across the system. All of the data for a particular file in the file system is held in a single record in the database. When node N adds another handle to that record, the extra data is associated with a handle that is owned only by node N . If node N fails immediately after adding that data, then two things happen: First, all open handles on that node are implicitly closed, and second, the data associated with the newly opened handle on node N is lost from the database.



Figure 2

Clustered Samba dispatcher daemon.

Thus, all that is required to be done is to ensure that when a node fails, the system is able to recover all pieces of each record in the `opendb` database that are not associated with handles opened on the failed node. An easy way to achieve that is for each node to locally store the last version of each record that it has itself written and to associate with that record a globally increasing sequence number. When a node fails, the system must scan the remaining nodes and for each record choose the instance with the highest sequence number as the record that will be used for the remaining nodes.

In this paper we do not attempt to prove that all the databases in Samba have this property, although we hope that the simplicity of the above argument provides a clear idea of how this can proceed. The developers found it was indeed a simple matter to show that all nonpersistent metadata databases in Samba have the property that all TDB metadata stored on a node can be lost without compromising correct CIFS semantics. Thus, it is possible to design a system that requires no shared storage for TDB metadata contents and no replication of data on write.

The result was a new cluster-aware version of TDB called *cluster trivial database* (CTDB) [20]. The CTDB system involves a distributed protocol for database operations between a `ctdbd` daemon on each node of the cluster, plus a local protocol for communication between Samba daemons on each node and the `ctdbd` on each node. The CTDB documentation [17] describes these protocols in some detail and describes the application programming interface that is used to talk to the CTDB daemons. Figure 2 shows a high-level design of the CTDB communication.

The core design of CTDB uses a simple two-level location system for records. The first is a *location master* for each record that is determined by a fixed hash on the database key. The second is a roving *data master* that,

Table 1 Performance on the NBENCH CIFS benchmark for a four-node cluster consisting of state-of-the-art Intel Dual-Core blades with a small GPFS file system back end. (CTDB: cluster trivial database.)

<i>No. of nodes</i>	<i>Pre-CTDB (MB/s)</i>	<i>With CTDB (MB/s)</i>
1	95.0	109
2	2.1	210
3	1.8	278
4	1.8	308

in the current implementation, is the last node to write to a given record. The location master always knows where the record is located (the identity of the data master), and the data master stores the most current copy of the record. Each record is augmented by additional metadata in the form of a header that contains the record sequence number (RSN), a 64-bit integer that is incremented whenever the data master moves between nodes. The RSN is used to recover the data in the database when a node enters or leaves the cluster by the simple mechanism of scanning the database (while globally locked) and choosing the instance of each record with the highest RSN. In addition to these database facilities, CTDB offers an integrated Internet Protocol (IP) failover mechanism, as is traditionally provided by high-availability clustering systems. This consists of public IP assignment, Address Resolution Protocol updates, local service initiation, and system monitoring functions.

The result when CTDB was incorporated into Samba was quite dramatic. **Table 1** shows the performance on the NBENCH CIFS benchmark for a four-node cluster consisting of state-of-the-art Intel Dual-Core blades with a small GPFS file system back end. The four results shown are for a load spread across one, two, three, and four nodes. With the old approach (TDB on shared storage), a dramatic slowdown is seen as soon as an attempt is made to use more than one node in the cluster. With the new approach, good scaling results.

The limit of the scaling in the above example is the total disk throughput that the system is capable of (only a small disk subsystem back end was available for testing). Even so, the improvement from the pre-CTDB approach demonstrates the utility of moving the metadata off the cluster file system.

Clustering NFS

When used in a cluster featuring the newly developed CTDB technology, the system attempts to present the NFS cluster IP connection to clients as if the cluster was

a single multihomed host. It is important to make sure that the IP address used by a client when mapping a share of the NFS server will stay the same, even after a cluster recovery has been performed that might have migrated that IP address over to a different physical node. NFS contains very little state and is reasonably easy to cluster. Besides keeping the IP address, it is important to also keep the same port the client was using for the initial connect and transfer of data.

When a client is connected to an IP address that migrated to a different server inside the cluster to a different cluster member, it is important to retain the same ports used by related services, such as NFS lock manager, NFS status daemon, and NFS quota daemon. This is because some clients assume that the ports used by these services may never change, which effectively lets these clients fail to rediscover the service if the port has changed. This issue is easily solved by locking these services down to using only a dedicated port.

The issue of transferring Transmission Control Protocol (TCP) connections is more difficult to solve, but TCP connections are preferably used with NFS because they perform better than User Datagram Protocol (UDP) connections. One consideration with TCP on Linux is that a TCP connection that is in an established state will continue to exist on the host even after the interface with the matching IP address is temporarily removed. Thus, if a TCP connection is established to an IP address on the server, the TCP connection can survive across multiple cluster reconfigurations while the IP address has been migrated off the node and later back to the node. Because many NFS clients reuse the same local port or small range of ports, TCP connections can get out of sync on the client, leading to many unnecessary packets between client and server, a situation referred to as an *ack-storm*.

This situation can occur, for example, when the following happens:

1. A client is connected to an IP address on the server.
2. The cluster is reconfigured, and the IP address is migrated onto a different node. However, the TCP connection remains on the original node.
3. The client reestablishes the TCP connection to the new node and, when doing so, negotiates a completely new sequence and *ack* (acknowledge) number for the TCP connection.
4. Another cluster reconfiguration occurs, and the IP address is migrated back onto the old node.

Because the TCP connection in step 1 can survive and remain on the old cluster node until step 4, and as the client is reusing the same client-side port when it reestablishes connections, it ends up in a situation in

which two different TCP connections (one established in step 1 and a second established in step 3) use the same identical socket pairs but different sequence and `ack` numbers.

One way to prevent this from happening is to shut down and restart the NFS service every time an IP address configuration change occurs on a node, but this would be expensive and disruptive because it leads to an outage of several seconds for all clients connected to the node and—because a node can and often will serve multiple addresses at the same time—not only those connected to the affected IP address.

In a SoFS environment, when an IP address is being migrated off the node, CTDB will attach this address to the local loopback interface instead of completely removing the IP address. This makes the IP address still reachable from processes running on the local node and allows CTDB to explicitly terminate individual TCP connections. Once an IP address is migrated off the node, a dedicated tool, `ctdb killtcp` is executed to send a constructed TCP `RST` to the server side, which terminates the TCP connection, thus avoiding the potential problem of an `ack`-storm while also avoiding an expensive restart of the NFS service.

Another issue with TCP is the time it may take a client to recover from a loss of connection. If a cluster reconfiguration occurs and the affected IP address is migrated onto a different node at a time in which a client is transmitting an NFS protocol data unit to the server across the TCP connection, it is possible that the client will enter a TCP retransmission timeout state waiting for TCP `acks` for the sent data. This timeout uses an exponential back-off and may result in the client waiting for many seconds before retransmitting the data and detecting the loss of the TCP connection. In order to speed up recovery on the clients so they quickly recover lost TCP connections and to short-circuit any TCP retransmission timeouts in the client, the use of a technique called *TCP tickles* is needed.

When a CTDB node takes over an IP address during cluster reconfiguration, it knows about all previously existing TCP connections that were established prior to the failover and sends out an artificial TCP `ack` with a 0 sequence number. When the client receives this TCP `ack`, the client will immediately respond by sending back its own `ack` to the server because the TCP sequence number is invalid. The server TCP stack receives this `ack`, but because it does not match any existing TCP connections, the server TCP stack will send a TCP `RST` back to the client with the current client sequence number. This triggers the client to immediately reestablish the TCP connection and short-circuits any TCP retransmission timeouts.

Comparison of NFS and CIFS locking

There are several differences in the locking semantics between NFS and CIFS, both for actual locking and also regarding the way locks are recovered during a server restart.

In CIFS, locking is mandatory and enforced on all read and write operations. This, combined with CTDB-Samba trying to achieve single-image semantics, poses additional performance problems for a multinode cluster. If a byte-range lock on a file is taken out on one file on a cluster node, all other nodes in the cluster must immediately become aware of this lock so that the tests for read and write operations can be enforced. Failure to do so would otherwise result in incorrect CIFS byte-range locking semantics.

To address this correctness issue, CTDB provides a clustered temporary database where all byte-range locks held by CIFS clients are stored. This database distributes the records across the cluster nodes on a per-record basis where one record contains all locking information for one single file.

Because Samba needs access to these records very frequently (every time a client performs a single read or write operation), it is critical that fetching and accessing these records can be done as fast as possible.

Most operations on files are localized in both time and node domain; this allows for a few optimizations on how to access the records. When Samba needs to access a specific record, Samba first locks this record in a local copy of the database that is stored in memory shared between the Samba processes and the local CTDB daemon. This record also contains information about whether the local stored version has the latest valid content. If the local copy of the record is current, Samba can use the content in the database record and proceed with its internal processing. In this scenario there is no need for any interaction between Samba and CTDB, and accessing the clustered database to lock, access, and unlock the record costs about the same as for the normal behavior or for a nonclustered version of Samba.

Only when Samba tries to access a local record that is not current will Samba need to invoke the CTDB service. If this happens, the local CTDB service will locate the most current version of the record in the cluster and migrate it over to the local node so it can be stored in the local copy of the database. This operation would commonly require no more than one or two network roundtrips between one or two pairs of nodes. In the vast majority of cases it is possible for Samba to access the records locally without any involvement of CTDB. Tests have shown that the cluster overhead for this database is marginal compared to a single instance of a nonclustered Samba daemon.

There is very little lock recovery in CIFS after a client or a server reboot. When a client or a server in CIFS has failed and restarted, all associated locks on files are lost on both the client and the server. Neither the CIFS client redirector nor the CIFS server will attempt any recovery of locks once the host has been restarted. If locks are to be recovered at all, it is the responsibility of the individual applications on the CIFS client to do so.

In NFS locking, locks are advisory and not part of the NFS protocol itself. Thus, an NFS server can service read and write operations from clients without regard to whether the file is locked. Instead, locking is implemented in a separate daemon or protocol called *network lock manager* (NLM). Having no association between file-sharing protocols and lock-management protocols makes it easier to implement clustering the NFS service. There is no need to store any lock information in a CTDB database or to modify the NFS and NLM services to provide a clustered service. The lock interaction between the NLM service in the kernel and the underlying file system driver is sufficient if the data is exported using a clustered file system such as GPFS.

In NFS, clients and servers will try to recover any locks across a failure and reboot in a way transparent to clients. This requires limited involvement from the CTDB service. Every time a cluster recovery or a reconfiguration of the CTDB occurs, the CTDB daemon will restart the NLM daemon on all available nodes in the cluster. Additionally, CTDB will send out status notification messages to all clients that have any locks held on the server to ensure that all clients know that they need to reclaim any locks. To cope with client-side implementation differences, these notifications are sent twice to each node, once specifying the cluster name and once specifying the IP address through which the client mapped the share as the server name.

From components to end-to-end management

The components described above—which consist of an extensible number of nodes, GPFS as the shared-file system, clustered versions of the server-side network protocol implementations for such file systems as CIFS and NFS, and the connecting hardware—can be assembled individually today. In high-performance computing, this has been demonstrated in many highly scalable installations. High-end examples of these systems can be found in the TOP500** Supercomputer list [21] over the past few years. All of these supercomputers are assembled as unique systems, especially at the very high end. A custom supercomputer such as these is not easy to administer and requires a lot of highly skilled operational effort. Bringing the benefits of a highly scalable shared-file system and the underlying hardware infrastructure into other sectors, such as

communications, digital media, pharmaceuticals, and for company-wide storage architectures in large enterprises, requires a different sort of administration. Indeed, operators in such companies are seeking an end-to-end view for managing this infrastructure, as they are used to for managing systems in a single box. Nowadays, all components exist for building a NAS system based on a highly scalable shared-file system that can provide the required I/O bandwidth by easily scaling the number of nodes. Managing the complexity of such hardware and software combinations becomes the issue preventing the introduction of a highly scalable NAS into organizations lacking deep skills and knowledge of the internal structures and operations. Another challenge is the management of a large number of end users requiring access to such a system if all storage in a large organization or company is consolidated into one extremely scalable system.

The resulting requirement is for a management infrastructure capable of presenting the entire complex infrastructure as a system that appears to the user as an appliance in a single console. Storage management front ends today are designed for managing single boxes or open infrastructures with servers, storage, and network devices [22], as well as the applications running on those devices. Compared with open infrastructure management, the complexity, connectivity, and localization of each component should be visible only to the level required for managing it as a NAS box. The difference from a classic NAS box is the ability to scale out the number of components to achieve the required levels of I/O bandwidth, capacity, and other parameters.

In addition to the requirement for a single control point, such an appliance should implement different user roles for different tasks. Typical roles include the following:

- *End users*—Being allowed to create and manage the size, life cycle, and service levels of spaces.
- *Power users or stewards*—Being allowed to monitor and manage spaces and end users.
- *Administrators*—Being able to manage all hardware and software components.

Figure 3 maps the Storage Networking Industry Association (SNIA) shared storage model [23] to SoFS.

Conclusion

This paper has presented a new approach to scalable NAS services that combines existing scalable file system and file server technologies with a new scalable distributed database system for Samba (CTDB) and adds an integrated storage management infrastructure. While the combination of GPFS, Samba (including CTDB), and NFS provides the foundation of highly scalable file

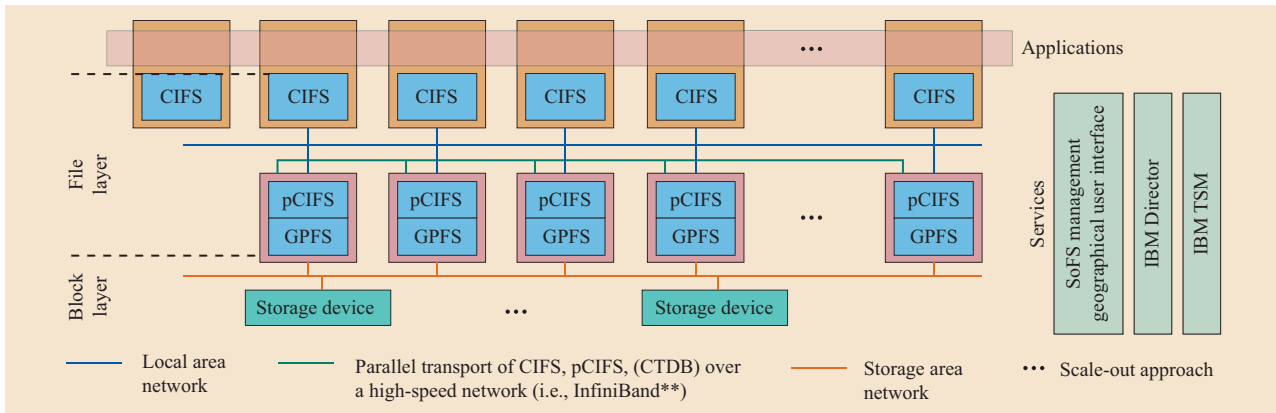


Figure 3

SNIA shared storage model mapped to SoFS. (CIFS: Common Internet File System; GPFS: General Parallel File System; CTDB: cluster trivial database; TSM: Tivoli Storage Manager.)

serving, the complementary management infrastructure addresses user requirements with respect to usability.

Integrating those technologies effectively creates a new, easy-to-manage system capable of providing SoFS. SoFS [24] is a scalable and manageable file service that provides a unique combination of highly scalable multiprotocol NAS storage with a highly available cluster architecture in an all-active configuration. The resulting system has the potential to fully meet the highly demanding storage needs of a wide range of industry sectors.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Linus Torvalds, Institute of Electrical and Electronics Engineers, Inc., Microsoft Corporation, Top500.org, or InfiniBand Trade Association in the United States, other countries, or both.

References

1. Samba: An Introduction; see <http://us1.samba.org/samba/docs/SambaIntro.html>.
2. 10 Years of Samba! see <http://us1.samba.org/samba/docs/10years.html>.
3. SNIA Storage Networking Industry Association, "Common Internet File System (CIFS) Technical Reference," Rev. 1.0; see http://www.snia.org/tech_activities/CIFS/CIFS-TR-1p00_FINAL.pdf.
4. Sun Microsystems, Inc., "NFS: Network File System Protocol Specification," IETF Request for Comments 1094, Network Working Group (March 1989); see <http://www.ietf.org/rfc/rfc1094.txt>.
5. B. Callaghan, B. Pawlowski, and P. Staubach, "NFS Version 3 Protocol Specification," IETF Request for Comments 1813, Network Working Group (June 1995); see <http://www.ietf.org/rfc/rfc1813.txt>.
6. S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network File System (NFS) Version 4 Protocol," IETF Request for Comments 3530, Network Working Group (April 2003); see <http://www.ietf.org/rfc/rfc3530.txt>.
7. J. Postel and J. Reynolds, "File Transfer Protocol (FTP)," IETF Request for Comments 959, Network Working Group (October 1985); see <http://www.ietf.org/rfc/rfc0959.txt>.
8. Rsync; see <http://rsync.samba.org>.
9. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol—HTTP/1.1," IETF Request for Comments 2616, Network Working Group (June 1999); see <http://www.ietf.org/rfc/rfc2616.txt>.
10. IDC, "The Expanding Digital Universe: A Forecast of Worldwide Information Growth through 2010," white paper (2007); see http://www.emc.com/about/destination/digital_universe/pdf/Expanding_Digital_Universe_IDC_WhitePaper_022507.pdf.
11. U.S. Food and Drug Administration, Welcome to Compliance References; see http://www.fda.gov/ora/compliance_ref/default.htm.
12. D. Einstein, "Doing One Thing Well: Network Appliance Has Carved Niche in File-Server Market," *San Francisco Chronicle*, August 22, 1998, sec. D, p. 1; see <http://www.sfgate.com/cgi-bin/article.cgi?f=/c/a/1998/08/22/BU38257.DTL&hw=david+hitz+netapp&sn=001>.
13. NetApp, Inc.; see <http://www.answers.com/topic/network-appliance-inc?cat=biz-fin>.
14. K. Komiega, "Microsoft and EMC Partner for APIs, New NAS Box," *SeachStorage.com*; see http://searchstorage.techtarget.com/originalContent/0,289142,sid5_gci896249,00.html.
15. AFS Frequently Asked Questions; see <http://www.angelfire.com/hi/plutonic/afs-faq.html>.
16. N. Zeldovich, Rx Protocol Specification (Draft); see <http://mit.edu/kolya/afs/rx/rx-spec>.
17. NetApp, Inc. (November 4, 2003). *Network Appliance Agrees to Acquire Spinnaker Networks*. Press release; see <http://www.netapp.com/us/company/news/news-rel-20031104.html>.
18. IBM Corporation, Storage Systems—Projects—GPFS; see <http://www.almaden.ibm.com/StorageSystems/projects/gpfs/>.
19. V. Lendecke, S. Oehme, A. Bokovoy, A. Fedoseev, and A. Tridgell, Samba & Clustering; see http://wiki.samba.org/index.php/Samba_%26_Clustering.
20. Samba CTDB, Welcome to the CTDB Web Pages; see <http://ctdb.samba.org/>.
21. Top500.org, TOP500 Supercomputer Sites; see <http://www.top500.org>.

22. M. Lovelace, M. Defiebre, H. Gunatilaka, C. Neal, and Y. Xu, "TotalStorage Productivity Center V3.3 Update Guide," IBM Redbook, IBM Corporation, April 23, 2008; see <http://www.redbooks.ibm.com/abstracts/sg247490.html>.
23. Storage Networking Industry Association, The SNIA Shared Storage Model; see http://www.snia.org/education/storage_networking_primer/shared_storage_model/.
24. IBM Corporation, Storage Optimization and Integration Services—Scale out File Services; see <http://www.ibm.com/services/us/its/html/sofs-landing.html>.

Received October 1, 2007; accepted for publication February 7, 2008; Internet publication June 3, 2008

Sven Oehme *IBM Deutschland GmbH, Hechtsheimer Strasse 2, 55131 Mainz, Germany (oehmes@de.ibm.com)*. Mr. Oehme started at IBM in 1993 as a team leader for a manufacturing infrastructure support team and later assumed the role as the development lead architect for Stonehenge/Open Enterprise System Virtualization, a scalable virtualization appliance. He is currently a subject-matter expert on a worldwide level for file systems. He is also the lead architect for the File System Competence Center Development group in the Linux Technology Center working on the development of extreme scalable file server solutions.

Juergen Deicke *IBM Deutschland GmbH, Hechtsheimer Strasse 2, 55131 Mainz, Germany (deicke@de.ibm.com)*. Dr. Deicke received Dipl.-Ing. and Ph.D. degrees in electrical engineering from the Darmstadt University of Technology, Germany. He joined IBM in May 2000 and has been working both as a software architect and as a development manager. Since February 2006, he has headed the IBM Systems and Technology Group systems software development organization in Mainz, Germany, where most software development projects are focusing on Linux, mainframes, tape management, and file systems. He lectures on innovation management at the Darmstadt University of Technology. Dr. Deicke is a member of the IEEE.

Jens-Peter Akelbein *IBM Deutschland GmbH, Hechtsheimer Strasse 2, 55131 Mainz, Germany (akelbein@de.ibm.com)*. Dr. Akelbein received a Dipl.-Ing. in electrical engineering from the Technical University of Braunschweig, Germany. For his research work on intelligent mass storages devices based on file-level protocols, he received a Ph.D. degree in electrical engineering from the Helmut-Schmidt-University, Germany. He joined IBM in 1999 working on the development team for the IBM TSM, mainly focusing on HSM. He managed departments for IBM supply-chain management applications and for TSM. He currently leads the Architecture and Development of Open Systems department in Mainz, developing the IBM Integrated Removable Media Manager on IBM System z* and solutions for SoFS.

Ronnie Sahlberg *IBM Australia, 8 Brisbane Avenue, Canberra ACT 2600, Australia (sahlberg@au1.ibm.com)*. Mr. Sahlberg is a member of the Samba team and a core developer for the wireshark protocol analyzer. He received a B.S. degree in mathematics from Linköping University, Sweden. His current

work focuses on Samba and scalable NAS systems making use of CTDB.

Andrew Tridgell *IBM Australia, 8 Brisbane Avenue, Canberra ACT 2600, Australia (tridgell@au1.ibm.com)*. Dr. Tridgell is one of the principal developers of the Samba software suite. After originally studying theoretical physics, he received a Ph.D. degree in computer science from the Australian National University, specializing in network protocols and distributed algorithms. He currently works for the Advanced Linux Response team in the IBM Linux Technology Center.

Roger L. Haskin *IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120 (roger@almaden.ibm.com)*. Dr. Haskin manages the File Systems department at the IBM Almaden Research Center. He received a Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign. Since joining IBM Research in 1980, he has pursued a wide variety of interests and has published in the areas of full-text retrieval, database, distributed systems, file systems, and multimedia technology. He originated and led the development of GPFS, perhaps the most widely used parallel file system available for high-performance computing. Dr. Haskin leads a number of other projects in IBM Research, including ones to investigate networked storage (NFS4 and pNFS) and to define the storage architecture for PERCS (Productive Easy-to-use Reliable Computer Systems), the IBM architecture for the Defense Advanced Research Projects Agency High Productivity Computing Systems program.