Concept and Evaluation of X-NAS: a Highly Scalable NAS System

Yoshiko Yasuda, Shinichi Kawamoto, Atsushi Ebata, Jun Okitsu, and Tatsuo Higuchi
Hitachi, Ltd., Central Research Laboratory
{yoshikoy, skawamo, ebata, j-okitsu, higuchi}@crl.hitachi.co.jp

Abstract

X-NAS (eXpandable network attached storage), a highly scalable, distributed file system designed for entry-level NAS, has been developed. It virtualizes multiple NAS systems into a single-file-system view for different kinds of clients. The core of X-NAS is a multi-protocol virtualized file system (MVFS), and its key features—a smart-code wrapper daemon, file-group mapping, and a file-handle cache—improve X-NAS scalability. X-NAS has other key features for improving the manageability on many NAS systems; namely, on-line reconfiguration, autonomous rebalancing, and automatic migration, in which files are migrated automatically and dynamically independently of file-sharing services for clients. To validate the X-NAS concept, an X-NAS prototype was designed and tested according to the NFSv2 implementation. These tests indicate that X-NAS attains a quicker response time and higher throughput than a conventional single NAS, so its cost-performance scalability is also higher.

1. Introduction

The advent of broadband networks is rapidly driving forward the development of services such as business information systems. Under such circumstances, companies are promoting the digitization of their data. Keeping an enormous amount of such digital data means that the importance of storage systems is rising; thus, the investment in storage systems in many companies has reached about three times that for server systems.

Network attached storage (NAS) is a network storage system—connected to an IP network—for efficiently managing digital data. NAS has recently been gaining general acceptance, because it can be managed easily and share files among many clients that run different operating systems using different file systems. Among the various kinds of NAS, an entry-level NAS system is convenient in terms of cost and ease of management for offices and departments with no IT (information technology) experts.

However, an entry-level NAS is not scalable; therefore, if it becomes filled to capacity, clients must buy another one. This means that they then have to administer two NAS systems. Accordingly, the more NAS systems there are to be administered, the more administration costs will increase.

To solve the above-described problems, several scalable distributed file systems have been developed [1,2,3]. These scalable file systems virtualize many distributed file systems into a unified one. However, they can only be used under a UNIX environment. When clients read or write a file on these file systems, they usually use file handles (which are file identifiers). Conventional systems put additional information, which indicates the file system that stores the file entity, into the file handles. When they need to rebalance between several file systems by system reconfiguration, file migration between several file systems is performed according to the administrator’s instructions.

The goal of the present work is to introduce a new concept called X-NAS (where X stands for expandable); namely, a scalable NAS architecture for NAS virtualization for reducing the administration cost of many networked NAS systems. It can be used from different kinds of clients, such as UNIX and Windows, like conventional NAS systems. Several key features for reducing X-NAS overhead can improve the system scalability while maintaining its manageability. Since X-NAS is designed for ease of management, it can easily be reconfigured without stopping file services or changing setting information, even file handles, in the client environment. It can also rebalance the available disk capacity between X-NAS elements automatically and dynamically in the background (i.e., independently of file-sharing services for clients).

This paper also validates an X-NAS concept by using an X-NAS prototype based on NFSv2 implementation. The evaluation results indicate that X-NAS incurs a lower overhead than a conventional entry-level NAS and has better cost-performance and scalability.

2. System overview

Figure 1 shows an overview of X-NAS, which includes one P-NAS (parent NAS) node and many C-NAS (child NAS) nodes. A C-NAS node is equivalent to a single NAS system, which includes an NFS daemon [4] and a data partition. Each file system on the data partition has the same directories tree as that of clients and is used to store file entities. P-NAS has two special functions: a multi-protocol virtualized file system, MVFS for short, and an X-NAS manager. MVFS is a global file system used for the virtualization and ease of management of many C-NAS nodes. It distributes each file entity to all data partitions. The X-NAS manager is responsible for the X-NAS manageability features such as on-line reconfiguration, autonomous rebalancing, and automatic migration facilities (section 4). It adds X-NAS elements to and deletes them from the X-NAS members by on-line reconfiguration. It also moves files from one data partition to the others during X-NAS reconfiguration.

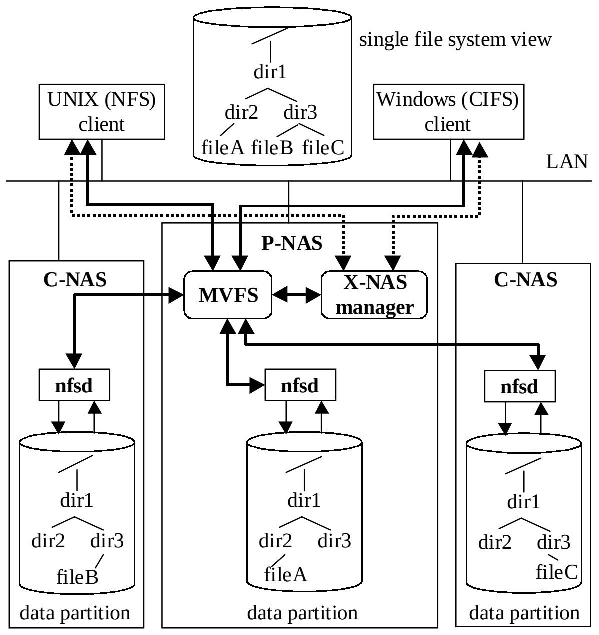


Figure 1. X-NAS overview.

3. X-NAS core

To reduce the administration cost of many NAS systems, a multi-protocol virtualized file system (MVFS) has been developed. The MVFS is based on NFS and distributes all files according to their inode numbers to X-NAS members. These lower-overhead features enable X-NAS to achieve effective cost-performance scalability.

3.1. Multi-protocol virtualized file system

The multi-protocol virtualized file system (MVFS), an X-NAS core, enables the centralized management of many NAS nodes and provides a unified file system view for clients. Figure 2 shows the structure of MVFS. It consists of Xnfsd, Samba [5], the management partition, an X-NAS configuration table, and a virtual partition (VP) mapping table.

Xnfsd is a wrapper of the NFS daemon and completely compatible with NFS (section 3.2.1). It is used for file-access requests from UNIX clients. Samba is used for Windows clients. The management partition provides a unified file system view for clients and is used to specify the C-NAS nodes that store the file entities. The file system on the management partition keeps the same files-and-directories tree as that of clients. However, all files on the management partition are zero-byte-size dummy files as shown in Fig. 2. These dummy files are used for examining the attribute information in the files and directories. They are also used to specify data partitions to store the file entities.

The X-NAS configuration table keeps setting information such as hostnames of X-NAS members and their export points. The VP mapping table is used to specify the data partition that stores files entities (section 3.2.2). These tables are updated during X-NAS reconfiguration.

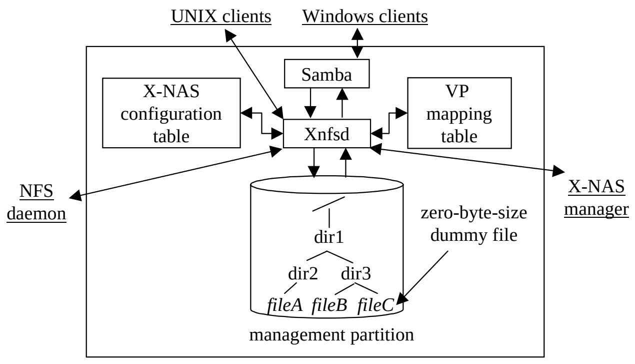


Figure 2. Structure of MVFS.

The file system on the management partition is exported to P-NAS and clients. On the other hand, for security, the file systems on the data partitions are exported to only P-NAS. When UNIX clients access X-NAS, they first mount the export point of the management partition on their mount point directory. As a result, they can get the root file-handle of the mounted file system. When they read or write files and directories by NFS operation, Xnfsd receives the NFS operation in place of the NFS daemon, and then sends it to the appropriate NFS daemon.

For Windows clients, X-NAS exports the directory on which the export point for UNIX clients is mounted. When a Windows client sends a SMB request to X-NAS, Samba receives the SMB request and sends it to the NFS daemon via Xnfsd. This method enables X-NAS to be used by both UNIX and Windows clients at the same time.

3.1.1. Inode-based file-distribution policy. X-NAS distributes all files into all data partitions according to the inode numbers of dummy files on the management partition. The inode number is a unique identifier of UNIX file systems and has one-to-one correspondence with the file. In the case of NFS, the file handle includes the inode number. When Xnfsd receives a file-access request from UNIX clients, it specifies the inode number of the dummy file from the file handle. It then specifies the data partition that stores the file entity by applying a hashing function, namely, a modulo function, to the inode number. Inode numbers of dummy files are not random numbers because they are determined by the file system on the management partition. However, our file-deployment policy specifies the data partition by hashing the inode numbers. All files are therefore distributed among all data partitions pseudo-randomly. Xnfsd also uses the inode number to specify the full path name of the file. The full path name is used to get the local file handle of the file on the appropriate data partition.

Using the inode number of the dummy file has two advantages as given below:

(1) The initial assignment of the file is determined when that file is created. It is assumed that users and applications often rename files and directories. Thus, if the initial assignment of the file is chosen according to the path name, the file location between many data partitions may be changed by renaming. In the case of inode-based assignment, even if clients change the file name, the inode number of the file stays the same. The file re-distribution is therefore not needed.

(2) Conventional scalable distributed file systems add special information to the file handle. This information is used to specify the file system that stores that file entity. The conventional systems also support file or volume migration. If file migration or volume migration occurs, the file handle must be revalidated. If clients read or write files by using file handles, each file-handle can be revalidated. However, in the case that the file systems are used for a back-up device, almost all files are accessed only once. In that case, file systems maintain the correspondence between the new file handle and the old file handle permanently. Since disk capacity is limited, it is difficult to keep the correspondence permanently. X-NAS solves this problem because it has no additional information in file handles. Instead of using additional information, it uses the inode number included in the file handle to specify the file location.

3.1.2. NFS operations on X-NAS. NFS operations can be divided into four categories by a file object type and a process type shown in Table 1. The types of file object are categorized as files and directories. The types of process are categorized as read and write.

Table 1: Categories of NFS operations.

Category	File object type	Process type
1	File	Read
2	File	Write
3	Directory	Read
4	Directory	Write

NFS operations belong to categories 1 and 2 are performed on the management partition and one of the data partitions, which store the file entities. The NFS operation of this type is performed as follows. When a client sends a READ operation to X-NAS, Xnfsd receives the READ operation in place of the NFS daemon. First, Xnfsd specifies the data partition that stores the file entity by using the inode number of the dummy file. Second, Xnfsd specifies the full path name by tracing the inode number of the dummy file and the disk blocks. Then, it sends LOOKUP operations with the full path name to the appropriate data partition. As the response to the LOOKUP operations, Xnfsd gets the local file handle of the file on the data partition. It then sends the READ operation to the data partition by using this local file handle. Finally, after Xnfsd gets a response, it sends it back to the client.

NFS operations belong to category 3 are performed on only the management partition. When a client sends LOOKUP operations for a directory to X-NAS, Xnfsd receives the LOOKUP operations in place of the NFS daemon. It reads the attribute information about the directory from the file system on the management partition, and returns its response to the client.

Since all X-NAS members have the same directries tree, NFS operations belong to category 4 are performed on both the management partition and all data partitions. The flow of MKDIR operation is as follows. When a client sends a MKDIR operation to X-NAS, Xnfsd receives the MKDIR operation in place of the NFS daemon. First, Xnfsd creates a directory on the management partition. Next, Xnfsd gets the local file handles from all data partitions by using LOOKUP operations and the full path name. Xnfsd then sends the MKDIR operation with the local file handles to all data partitions. Finally, when Xnfsd gets responses from all NFS servers, it makes one response from all responses and sends it back to the clients.

3.2. Scalability

It is difficult to achieve compatibility between simple-and-unified management and scalability. Accordingly, although the X-NAS architecture has capacity to scale up to about a 64 NAS system, we considered the first X-NAS target market segment to be from entry-level to midrange NAS. This is because midrange NAS is much more expensive than an entry-level NAS system and the cost-and-performance gap between these two NAS systems is wide. To lower the cost compared with the expensive midrange NAS, X-NAS uses entry-level NAS as an X-NAS element. To cover these market segments, X-NAS provides capacity to scale up to several terabytes. This is equivalent to the total capacity of ten entry-level NAS systems. X-NAS can achieve not only capacity scalability but also cost-performance scalability through three key features: a smart-code wrapper daemon, file-group mapping, and a file-handle cache.

3.2.1. Smart-code wrapper daemon. Xnfsd, which is the heart of MVFS, is implemented as a wrapper of an NFS daemon. The wrapper daemon receives a file-access request in place of the NFS daemon from the client, and sends this request to the appropriate NFS server. Although Xnfsd emulates the NFS daemon and is completely compatible with NFS, it has a very simple and smart structure. Since the X-NAS code makes every effort to avoid waste, it has only about 10,000 lines of C code. The quality of X-NAS is therefore higher. Moreover, Xnfsd does not waste CPU power. By using this light-weight wrapper daemon, the X-NAS overhead can thus be reduced.

3.2.2. File-group mapping. Since X-NAS consists of many NAS systems, it has to manage many more files than a single NAS system. As the number of files to be managed increases, more file-mapping information is needed. To solve this problem, X-NAS manages all files as a unit of a virtual file group, namely, virtual partition. Since the virtual partition is a logical unit for managing files, it has no relevance to the files-and-directories tree on the management partition. The relation between the virtual partition and the data partition is stored in a virtual partition (VP) mapping table. The VP mapping table is usually located on the memory of P-NAS. It is copied to the disk space of P-NAS at constant intervals. The VP mapping table is referred to in order to specify the data partition that keeps the file entity. It is updated when the X-NAS configuration is changed (described in section 4.1).

File-group mapping has many advantages. One is that the amount of mapping information is much smaller than that of file-level mapping. The second is the short search time due to the small table size on the memory. The third is that the VP mapping table is updated rarely, compared to file-level mapping. These advantages mean that X-NAS is a low-overhead architecture.

3.2.3. File-handle cache. X-NAS has all information for specifying the file location on both the management and data partitions. However, the cost of specifying the full path name of the file object is high because Xnfsd traces the disk blocks on the management partition. The cost of specifying the local file handle on the data partition by using the LOOKUP operations and the full path name is also high. To reduce these costs, X-NAS has a file-handle cache that keeps the correspondence between the file handle of the dummy file, i.e., the global file handle, on the management partition and the local file handle on the data partition. The file-handle cache is generally kept in memory on P-NAS. As a result of file-handle tracing, Xnfsd registers the local file handle on the file-handle cache along with the inode number of the global file handle and the data partition number that keeps the file entity. After that, X-NAS can reuse the local file handle on the memory. The overhead incurred by accessing the management partition can therefore be reduced.

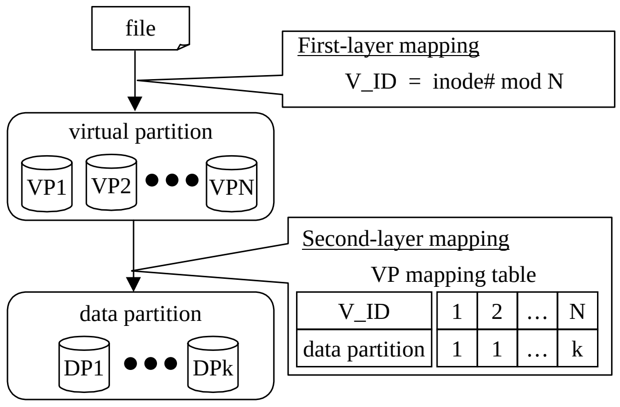
4. Key features

Simple administration is a strong sales point of conventional NAS systems. However, the more NAS systems there are to be administered, the more administration costs will increase. To maintain the manageability of a single NAS while providing a high level of scalability, X-NAS has three key features: on-line reconfiguration, autonomous rebalancing, and automatic migration. These features enable X-NAS reconfiguration to be easily performed without influencing client environments.

4.1. On-line reconfiguration

On-line reconfiguration, which can add or remove NAS nodes easily and transparently without stopping the client file-sharing services, is a strong feature of X-NAS. When one of the NAS nodes in X-NAS is unstable, the administrator removes this NAS node by using a Web browser. After that, X-NAS automatically moves all accessible files from the unstable NAS node to the other nodes.

The core architecture of on-line reconfiguration is two-layer mapping as shown in Figure 3. The first-layer mapping correlates the dummy file on the management partition with the virtual file group, namely the virtual partition. The second-layer correlates the virtual partition with the data partition. The relation between the virtual partition and the data partition is stored in a virtual partition (VP) mapping table.



V_ID	1	2	…	N
data partition	1	1	…	k

Figure 3. File distribution by two-layer mapping.

All files are handled as a unit of virtual partition, which is invisible to clients. The number of virtual partitions N is independent of that of data partitions and is fixed in advance. In terms of object balancing, using a lot of virtual partitions in a few data partitions is useful [8,9]. In X-NAS, N is from approximately 100 to 1,000 times the number of data partitions. A virtual partition ID, V_ID for short, is calculated by using the inode number of the dummy file and the virtual partition number N (as shown in Figure 3). Firstly, each virtual partition is assigned at data partitions equally. Even if the X-NAS manager moves the virtual partition from one data partition to another by on-line reconfiguration, the correspondence between the file and the virtual partition to which the file belongs is the same. This is because the inode number of the dummy file is unchanged and N is a fixed number. On-line reconfiguration is thus easy; it simply involves updating the VP mapping table and the X-NAS configuration table. File migration during on-line reconfiguration is performed as a unit of virtual partition in the background, namely, without disturbing the file-access requests from clients.

4.2. Autonomous rebalancing

The file-distribution policy of X-NAS is based on inode numbers. According to this distribution policy, the number of files may be equally distributed among data partitions. However, each data partition does not use the same amount of disk space, because each file has a different size. Moreover, X-NAS can be used in a heterogeneous environment because the capacity of data partition may change with shipping year and cost. On-line reconfiguration, i.e., adding or deleting NAS nodes, also incurs capacity unbalance. The capacity unbalance between X-NAS members may cause a concentration of file-access requests from clients. In response to these circumstances, X-NAS has an autonomous rebalancing facility that moves files between data partitions automatically and dynamically without any client administration if the available capacity size of each data partition becomes unbalanced.

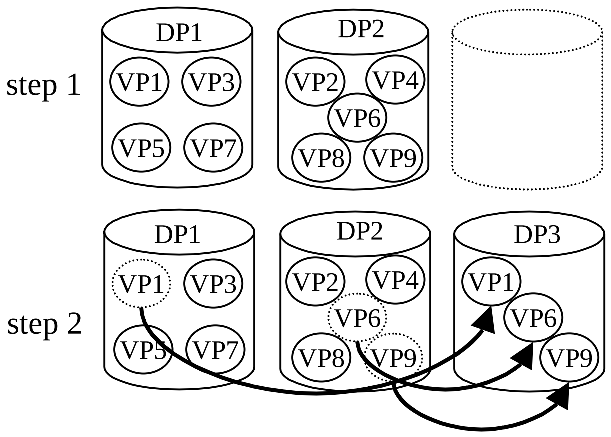


Figure 4. Example of autonomous rebalancing.

The X-NAS manager checks the available capacity size of each data partition at constant intervals. It moves some of files in this data partition when the available capacity size in any data partition is lower than the threshold chosen in advance. Since files are managed as a unit of virtual partition, it selects some virtual partitions randomly on the full data partition according to the VP mapping table. It then moves files in these virtual partitions to the other data partitions by NFS operations such as READ and WRITE. After moving them, the X-NAS manager updates the VP mapping table. Figure 4 shows an example of autonomous rebalancing. Firstly, the number of X-NAS members is only two, and all virtual partitions (in this case N is nine) are mapped to DP1 and DP2 (step 1). Secondly, X-NAS adds DP3 by on-line reconfiguration. When the X-NAS manager recognizes the capacity unbalance, it selects some virtual partitions (VP1, VP6, and VP9) on both DP1 and DP2 and moves them to DP3 (step 2). Although there are several algorithms for selecting which virtual partition to move, the simplest way is random selection to balance the available capacity between all data partitions. Since this autonomous rebalancing facility rebalances data placement between the X-NAS elements in the background, clients can continuously access their files on the X-NAS.

4.3. Automatic migration

Conventional entry-level NAS is not scalable, and it has no MVFS or on-line reconfiguration functions. Clients with existing entry-level NAS systems can not therefore expand their systems. On the other hand, X-NAS can expand the capacity of the existing NAS by keeping the files-and-directories tree on the existing NAS system.

When the X-NAS manager receives an automatic migration request from clients, X-NAS stops file-sharing services on the existing NAS system and mounts the file system of the existing NAS on X-NAS. Then X-NAS traces the files-and-directories tree on the existing NAS and copies this tree to the file system on the management partition. Just after this copying processing, the VP mapping table indicates that all virtual partitions belong to the existing NAS. After X-NAS makes the files-and-directories tree on the management partition, it restarts the file-sharing service for clients and moves some virtual partitions corresponding to the existing NAS to another data partition in the background. For example, when using a capacity-rebalancing policy, X-NAS moves the virtual partitions to balance capacity availability between all data partitions.

5. Performance evaluation

To validate the X-NAS concept, we implemented an X-NAS prototype on several 1U Linux servers. (Note that the X-NAS prototype is now based on the NFSv2 implementation.) All X-NAS functions are provided as a user-mode process on Linux. Since X-NAS is independent of the Linux kernel, it is portable.

Evaluation of the overhead and the scalability of X-NAS were conducted by using the industry-standard SPECsfs97 benchmark program [6], which measures the performance of the NFS server, on the X-NAS prototype. The evaluation index of SPECsfs97 is throughput, that is, the number of executed NFS operations per second when the same number of NFS operations is offered. Since implementation of the X-NAS prototype is based on NFSv2, an NFSv2 working set was used in the test. Note that the number of mount points was only one.

5.1. Experimental environment

Figure 5 shows the experimental environment. Since a C-NAS only has one data partition, it is the equivalent to the original NFS server. However, P-NAS has a management partition in addition to the data partition. The number of disk accesses to P-NAS is therefore twice that to C-NAS. To examine the effect by accessing the management partition, two test cases were performed: one is that P-NAS has both a management partition and a data one; the other is that P-NAS has only a management partition, that is, a redirector.

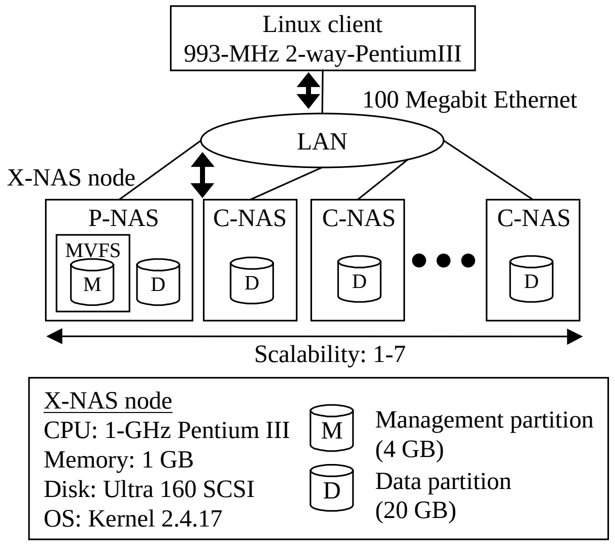


Figure 5. Experimental environment.

5.2. Experimental results

The performance results from the SPECsfs97 test in Figure 6 shows the relationship between the offered load and the average response time of all NFS operations. The response time of four-way X-NAS is slower than that of the original NFS server. This is because X-NAS must access the data partitions on C-NAS via the network. In the case of a higher offered load, the response time of four-way X-NAS is dramatically increased. This is because there is a bottleneck in the disk accesses to P-NAS. In the case of seven-way X-NAS, the response time is faster than that of the original NFS because the disk accesses are distributed between many C-NAS nodes.

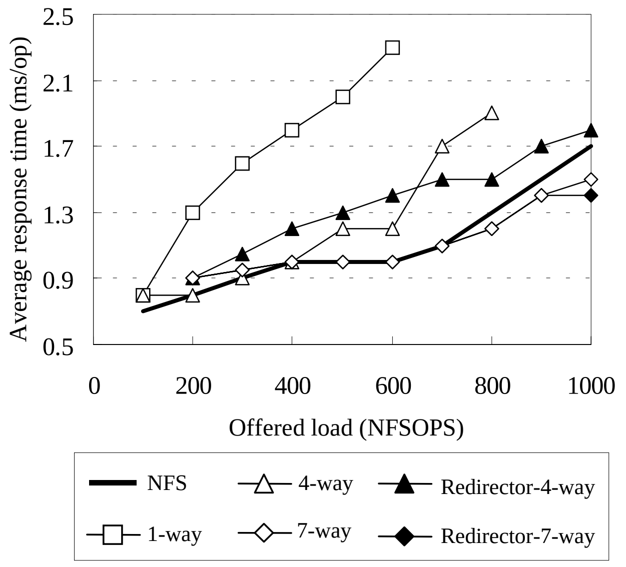


Figure 6. Average response time.

Figure 7 shows the relationship between offered load and delivered load. The delivered load indicates whether the system can process all NFS operations of an offered load within a benchmarking time. If the delivered load is equal to the offered load, it indicates that no bottleneck occurs in the system. As shown in Figure 7, the peak delivered load of the original NFS is 700 NFSOPS. On the other hand, the one- and four-way X-NASs have lower performance than the original NFS server. However, when the number of data partitions increases, the delivered load on X-NAS becomes higher than that on the original NFS. When the offered load is 1000 NFSOPS, the delivered load is sustained (i.e., load is saturated).

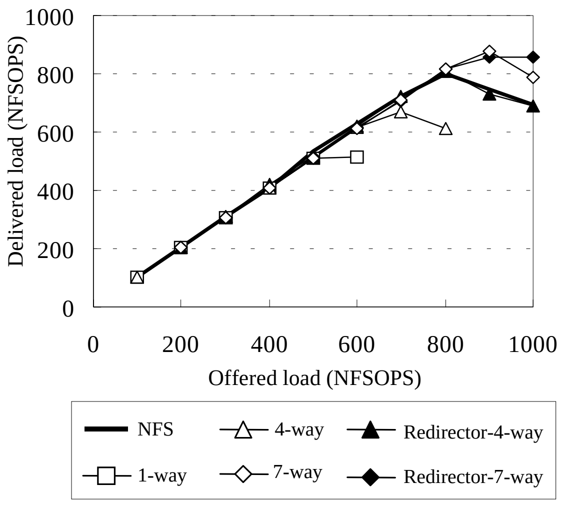


Figure 7. Throughput.

5.3. Discussion

To investigate the above-mentioned load saturation, the response times for LOOKUP and READ operations were analyzed. Figure 8 shows the average response time for LOOKUP operations. It is clear that the average response times of the one- and four-way X-NASs dramatically increase. This indicates that the bottleneck is caused on P-NAS by accessing the management partition. On the other hand, the response time of seven-way X-NAS is constant. This is because there is no bottleneck in the disk accesses to P-NAS. On the other hand, the average response time of READ operations (shown in Figure 9) gets longer as the load increases. However, at 1000 NFSOPS, the average response time for READ operations saturates. This means that the bottleneck is not disk access.

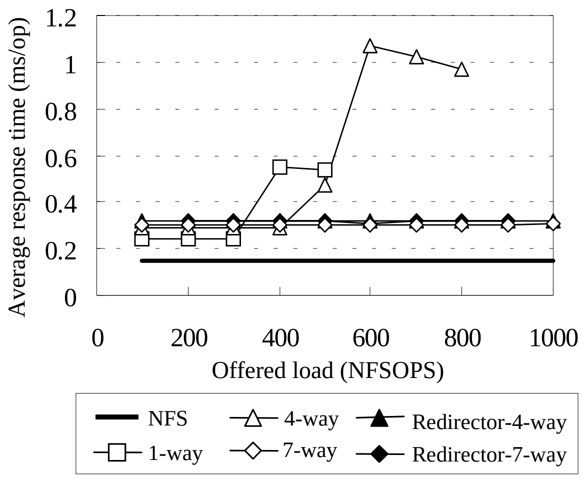


Figure 8. Average response time for LOOKUP operations.

The transfer rate of the network was calculated by using the workload mix and the average file size to be accessed on the SPECsfs97 benchmark program. As a result, it was found that the bottleneck is the network transfer rate. The evaluation environment consisted of a 100-Mbit Ethernet as the network. However, if the Gigabit Ethernet were used, this network bottleneck could be reduced.

Results of the performance evaluation described above demonstrate that X-NAS attains a quicker response time and a higher throughput than a single NFS server. This result verifies that the X-NAS has good cost-performance scalability.

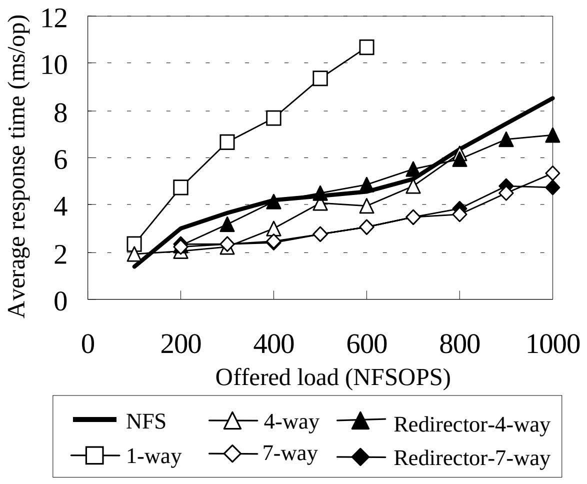


Figure 9. Average response time for READ operations.

6. Related work

There are several studies on scalable distributed file systems based on NFS. In particular, DiFFS [1,2] and Slice [3] resemble X-NAS in terms of their design principles. DiFFS delivers high performance because it is distributed among many partitions. It puts additional information, i.e., the partition ID, into a file handle. File handles must therefore be revalidated in the case of on-line reconfiguration. Slice, developed at Duke University, is also close to X-NAS in terms of its partition approach and use of directory servers. The partition used in Slice is a single server. Slice also puts a file ID as a routing key in each file handle, so it can be easily reconfigured. However, it must revalidate the file handles when the system is reconfigured. Furthermore, these scalable distributed file systems do not correspond to Windows clients.

As for on-line reconfiguration, several file relocation algorithms has been proposed. For example, LH* which is a generalization of Linear Hashing [7], distributes files among many disks by hashing primary keys of file objects. When an administrator wants to add a new disk to a cluster, it moves the half of objects on one disk to the new disk. The optimal algorithm proposed by Honicky [8,9] uses a “set” that is similar to a virtual partition of X-NAS. In this algorithm, each file is mapped to a set that contains many file objects. When administrators add a new disk, this algorithm specifies all of the sets that will be moved from an existing disk, and moves them to the new one. However, these two algorithms have to send messages about the update configuration to clients.

Lunar Flare NAS, developed at Tricord, is a Windows-based scalable NAS system [10]. It can be used both on UNIX and Windows clients. Its key feature is its Illumina software, which performs NAS aggregation. Since Illumina is distributed to all elements, Lunar Flare NAS has high scalability. However, since it is based on the unique file system, the members of Lunar Flare NAS are limited to products by Tricord.

X-NAS is a scalable architecture based on a standard file system such as NFS. It can provide file-sharing services for many kinds of conventional client OS. To specify the file location, it uses the inode number instead of using additional information in each file handle. An autonomous rebalancing facility checks the available capacities of each data partition at constant intervals. It moves files as a unit of virtual partition when capacity unbalancing occurs or administrators add or delete NASs. These file-migration processes are performed dynamically and automatically in the background (i.e., independently of file-sharing services for clients). In addition, X-NAS allows its elements to be many kinds of NAS systems with NFS services, since X-NAS elements need no special information about X-NAS. It can also expand the capacity of the existing NAS system by keeping the existing files-and-directories tree; therefore system reconfigurations are easily performed.

7. Future work

X-NAS has key features for scaling up its capacity while keeping its performance. It can thus achieve good cost-performance scalability in our target market segment, in which there are about ten NAS systems. However, when the number of NAS systems to be managed by X-NAS ranges from about 32 to 64, the performance of current X-NAS is not sufficient. This is because the file-access requests on the current X-NAS are managed by one P-NAS. To solve this problem, two approaches are being considered: hardware and software improvements to eliminate the network bottleneck; and raising the CPU frequency and on-chip multiprocessor to eliminate the CPU bottleneck. In the former approach, applying Gigabit Ethernet and changing the transmitting policy, which directly returns the responses of the file-access requests from the data partitions to the client, are promising. If the network bottleneck were eliminated, a CPU bottleneck would occur. However, the CPU usage is less than 30% on the current P-NAS when the number of X-NAS members is up to eight. The CPU bottleneck thus can be reduced by the latter approach.

As for petabytes scalability of X-NAS, a hierarchical approach is essential. Since there are several methods for a hierarchical approach, it is not described here in detail. However, grid architecture will be useful in terms of security.

Regarding X-NAS reliability, a single point of failure is also a problem. However, it can be solved by X-NAS clustering, which will be discussed in detail in another paper.

8. Conclusions

A new concept, named X-NAS, a highly scalable NAS system, has been developed. The X-NAS core, named MVFS, provides a single file system view for clients that run on the different operating systems and enables administrators to manage many NAS elements easily. Three key features of MVFS can improve the X-NAS cost-performance scalability. Firstly, a smart-code wrapper daemon with low overhead in place of a NFS daemon distributes file-access requests into many NFS servers. The file-distribution policy based on inode numbers is useful for renaming of files and directories and the system reconfiguration. Secondly, file-group mapping can reduce the amount of file-distribution information. By this mapping, the overhead to search for the file location can be reduced. Thirdly, the file-handle cache, which keeps the correspondence between the global file handle and the local file handle, enables the overhead for accessing MVFS to be reduced.

To improve the manageability of many NAS systems, X-NAS's on-line reconfiguration enables administrators to add or remove X-NAS elements without stopping file-sharing services for clients. In addition, autonomous rebalancing can rebalance the available disk capacity by moving files between X-NAS members automatically and dynamically independently of file-sharing services for clients. Furthermore, automatic migration expands the capacity of the existing NAS node by keeping the existing files-and-directories tree.

An X-NAS prototype, which is based on NFSv2 implementation, delivers a quicker response time and a higher throughput compared to a single original NFS server running the SPECsfs97 benchmark program. X-NAS not only maintains the performance of the single NAS system but also provides a unified file system view for both UNIX and Windows clients. Consequently, X-NAS has good cost-performance scalability.

References

[1] Christos Karamanolis et al.: DiFFS: a Scalable Distributed File System, Technical Report HPL-2001-19, HP Laboratories Palo Alto, 2001.

[2] Christos Karamanolis et al.: An Architecture for Scalable and Manageable File Services, Technical Report HPL-2001-173, HP Laboratories Palo Alto, 2001.

[3] Darrell C. Anderson et al.: Interposed Request Routing for Scalable Network Storage, ACM Transactions on Computer Systems, Vol. 20, No. 1, February 2002.

[4] Brent Callaghan: NFS Illustrated: Addison-Wesley Professional Computing Series: Addison-Wesley, 2000.

[5] Chris Hertel: Samba: An Introduction: http://us1.samba.org/samba/docs/SambaIntro.html

[6] Standard Performance Evaluation Corporation: SFS3.0 Documentation Version 1.0: http://www.spec.org

[7] Witold Litwin et al.: LH*: a scalable, distributed data structure, ACM Transactions on Database Systems, Vol.21, No. 4, 1996.

[8] R. J. Honicky et al.: An Optimal Algorithm for Online Reorganization of Replicated Data, Technical Report UCSC-CRL-02-36, University of California, Santa Cruz, November 2002.

[9] R. J. Honicky et al.: An Optimal Algorithm for Online Reorganization of Replicated Data, In Proceedings of the 17th International Parallel and Distributed Processing Symposium (To be published).

[10] Tricord systems: http://www.tricord.com