mNFS: Multicast-Based NFS Cluster*

Woon-Gwun Lee, Chan-Ik Park, and Dae-Woong Kim

Department of Computer Science and Engineering/PIRL
Pohang University of Science and Technology
Pohang, Kyungbuk 790-784, Republic of Korea
{gamma,cipark,woong}@postech.ac.kr

Abstract. NFS is a distributed file system which is widely used in UNIX environments. Many studies have been worked to improve its I/O performance and scalability. However, existing architectures, which have either single load balancer or meta server, could suffer from single-point-of-failure and unnecessary network delay. We propose a new NFS architecture called mNFS cluster to improve its scalability. Experimental results show that mNFS outperforms both Linux NFS and NFSp by 64% and 99%, respectively.

1 Introduction

NFS is a distributed file system, which is widely used in UNIX operating system. NFS performance depends on network bandwidth, disk I/O throughput and CPU power for processing NFS protocol. Unlike AFS and CODA distributed file system, NFS server was designed to work on a single host resulting in relatively poor scalability. Several research such as NFSp[1] and DirectNFS[2] has been conducted to improve its performance and scalability. Bigfoot-NFS[3] allows multiple NFS file servers to use transparently their aggregate space as a single file system. It applies RAID techniques to network file systems. Bigfoot-NFS uses client-driven approach and runs without any central meta server. The server side is composed of normal NFS servers. Each NFS server has distinct files, that is, files are the unit of interleaving. To provide a single name space, all of the NFS servers have the same directory hierarchy. When multiple files are accessed, I/O loads are distributed into the NFS servers, and I/O performance can be improved. However, according to the result of initial implementation, 24 nodes Bigfoot-NFS performs worse than SunOS kernel-level NFS implementation and single node Bigfoot-NFS.

NFSp is an extension of NFS, which enables the use of the disk space of the nodes of a PC-based beowulf cluster. Early developed distributed file systems, such as AFS, CODA and xFS, have too many features to apply for cluster system

* The authors would like to thank the Ministry of Education of Korea for its support toward the Electrical and Computer Engineering Division at POSTECH through its BK21 program. This research was also supported in part by HY-SDR IT Research Center and in part by grant No. R01-2003-000-10739-0 from the Basic Research Program of the Korea Science and Engineering Foundation.

in the private network. NFSp is divided into several kinds of dedicated servers. A meta server holds file attributes, whereas an I/O node stores the content of files physically. NFSp server acts as a standard NFS server. All I/O requests are sent to the meta server and then forwarded to I/O nodes. Hence, the meta server can be a bottleneck and forwarding I/O requests to I/O nodes causes additional network delay. Moreover, NFSp is based on the NFS protocol version 2 and user-level NFS servers, which limits the I/O performance. An experiment shows that a NFSp system, which have 32 I/O nodes, performs worse than a Linux kernel-level NFS server.

DirectNFS[2] is a hybrid architecture of both NAS and SAN storage system. It extends NFS protocols. DirectNFS is composed of a meta server and SAN disks. Legacy NFS clients are able to access the data in the SAN disks via the meta server which supports the standard NFS protocol. DirectNFS clients, however, have direct connection to SAN and they are able to bypass the meta server, once they obtain meta data from the meta server. Results with the initial implementation shows that DirectNFS performs as good as local file systems, EXT2 and ReiserFS. But DirectNFS system costs high because DirectNFS clients require SAN connectivity.

Bigfoot-NFS, NFSp and DirectNFS are all server driven architectures which require either load balancer or meta server. Therefore, they suffer from single-point-of-failure or additional network delay. To overcome these disadvantages, this paper proposes a multicast-based NFS cluster called *mNFS*. The mNFS is a client driven approach. A mNFS client sends each NFS request to multiple servers via multicasting. Thus, it can perform more efficiently transmission to multiple cluster nodes. The rest of this paper is organized as follows. Sections 2 describes the design of multicast-based NFS cluster and its prototype implementation. Section 3 gives experimental results. Finally, concluding remarks are given in Section 4.

2 mNFS: A Multicast-Based NFS Cluster System

To begin with, mRPC is designed for mNFS and it is a one-to-multiple variant of the normal one-to-one RPC. To support *portmapper* lookup, RPC protocol[4] has one-to-multiple RPC, called broadcast RPC. The broadcast RPC does not decide the set of receiving host and its transmission is limited in local subnet due to the characteristics inherited from broadcasting. mRPC is based on multicasting with IGMP[5] and is able to control the set of receiving hosts owing to the multicast group management facility of IGMP. If all the nodes of a mNFS cluster are joined to a multicast group, than they are capable to receive a NFS request simultaneously. When receiving a NFS request, each cluster node processes it selectively according to request type and load balancing policy of the cluster system.

To provide a single file system image, each cluster node maintains identical metadata of virtual file system for the mNFS cluster where each node has its own local disk and its local file system.

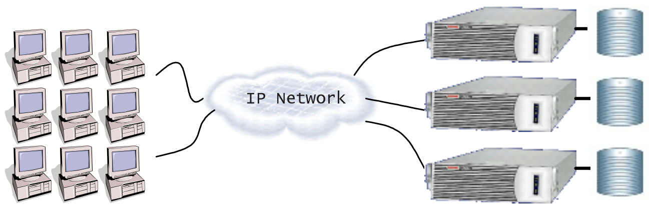


Fig. 1. Dedicated Disk Architecture

The metadata write operations like MKDIR, RMDIR and CREATE, must be sent to and processed by all cluster nodes. In the case of metadata read operations such as READDIR and GETATTR, it is enough for only one cluster node to handle the request and return the result to a client. A cluster node which is responsible for each file block I/O is determined by the offset of the block. The mNFS cluster system stripes data among cluster nodes. When a block read or write request is issued, a given file offset determines which cluster node should handle the block request.

2.1 Considerations of mNFS

Single Filesystem Image. A NFS client gets NFS server-side file system attribute information like Table 1 through a *STATFS* request. When a *STATFS* request is sent by a client, a corresponding cluster node returns virtual file system's attribute information configured by the mNFS cluster.

Metadata Operations. There is no control node in a mNFS cluster system, and each cluster node runs autonomously in co-operation with each others. Decision for I/O load balancing is made by each cluster node independently. Hence, the mNFS cluster must have a predefined load balancing policy to resolve conflict between which node is going to handle an I/O request. On the mNFS prototype, the RPC transaction ID is used as a unique identifier for round-robin load balancing policy.

Table 1. Information of File system attributes

Attribute	Description
bsize	block size of file system
blocks	the total number of blocks
bfree	the number of free blocks
bavail	the number of blocks available to non-privileged users
files	the number of used*inode*
ffree	the number of free *inode*

File I/Os. Performance improvement of the mNFS cluster system depends mainly on file I/O operations. The load balancing of file related operations can't be achieved by simple round-robin policy with RPC transaction ID because cluster node does not handle the whole of a file. A file is divided into several chunks of a fixed length, and they are distributed and stored into the cluster nodes. Therefore, the mNFS cluster can determine which cluster node are responsible for a (read or write) operation of a chunk of data as follows:

$$\begin{aligned} chunk &= \lfloor \mathit{offset} \;/\; su \rfloor \\ i &\equiv (chunk + ino) \bmod n \end{aligned}$$

In the above equation, *offset* refers to a location of data block in a file, *chunk* refers to a processing unit of contiguous data blocks, *su* is the size of a chunk, *ino* is an inode number, and *i* refers to cluster node ID.

Inode Number Manipulation. The prototype mNFS cluster is based on dedicated disk architecture. So, each cluster node has its own disks and local file systems. In this environment, each file should be assigned a unique identifier which is common over the multiple cluster nodes. In the NFS protocol, an inode number, *ino* is used as an unique identifier of a NFS I/O operation. Even though each cluster node makes use of its own local file system, if it uses the same kind of disk and file system, then the inode number of newly created file becomes common over all cluster nodes. On the Linux, EXT2[6] is one of the most widely used local file systems. In EXT2 file system, the inode number of a file is allocated sequentially and can be synchronized easily among cluster nodes. In the case of directory, inode numbers are allocated on a less occupied block group. And each cluster node may have different block group usages due to the policy of file interleaving. Consequently, different inode numbers can be allocated for the same directory. Thus, in the prototype, we removed dynamic factors of inode number allocation for directories.

File Write and Inode Object. The metadata update operation such as *SETATTR*, *CREATE*, *REMOVE*, *MKDIR*, *RMDIR*, is achieved by the mRPC. The mRPC is performed at the same time and maintains the identity of a node metadata in all cluster node. But the mNFS cluster needs to update additional metadata changes. If file write operation occurs on a chunk of file data, then it implicitly updates the corresponding inode's fields such as file length and modification time. In a mNFS system, a file write request is dealt with only in a cluster node which is selected by load balancing policy. However, the change of inode object should be updated by all cluster nodes to keep identical metadata over the system.

2.2 The Pros and Cons of mNFS

The mNFS has the following advantages owing to multicast-based clustering technique.

- Standard NFS Protocol.
  Both server-side and client-side of the mNFS cluster requires some modification. But it does not modify any NFS protocol and not define any additional protocol. Thus the mNFS can inherit the stability from the standard NFS protocol.
- Improvement of File I/O Throughput.
  Generally, when processing a request in network file systems such as NFS and CIFS, the disk time is longer than the request transmission time. For a file read/write request, NFS does not consider an order of I/O requests. So the mNFS is able to improve I/O performance because file I/O requests are distributed into multiple cluster nodes.
- Scalability.
  System scalability has been restricted by its physical scalability. If a system is composed of multiple hosts, it could be more scalable than a single host system. Besides the mNFS cluster is based on IP multicast. Therefore the number of cluster nodes does not affect the time of NFS request transmission.
- Elimination of Additional Network Delay.
  If it has any control node for storage cluster such as load balancer or meta server in NFSp, at first, an I/O request is sent to a control node, and then the real I/O node, which is responsible for the request, is determined. At the end, the I/O request is handled at the I/O node. So, it requires two network transmission to deliver an I/O request to the corresponding I/O node. Whereas, mNFS cluster sends an I/O request through a multicasting, this mechanism delivers I/O request to all cluster nodes with just one transmission. And then each cluster node determines whether it should handle the request or not. In this way, there is no additional network delay.

Transmitting NFS requests via mRPC forces a mNFS client to be modified, increasing the maintenance cost of system. When a block write request is issued, the request is sent to all cluster nodes through multicasting and valuable network bandwidth is wasted by the multiple copies of the data block. Also, because metadata write operations must be performed synchronously by all of the cluster nodes, its response time are affected by the synchronization overhead.

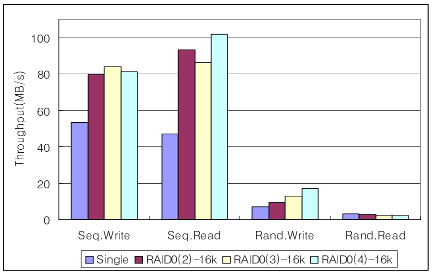
3 Evaluation

A prototype of the mNFS cluster is based on the NFS implementation included in Linux kernel 2.4.20. Ext2, which is widely used in the Linux environments, is used as the local file system of mNFS cluster nodes. The mNFS cluster server consists of up to three cluster nodes. The cluster nodes and a single client are connected by gigabit ethernet. The multicast group address for the mNFS cluster is configured with *224.100.1.1*. The cluster nodes and client are used with PC-based host as shown in Table 2. Each host has two disks one for operating system and the other for storage space.

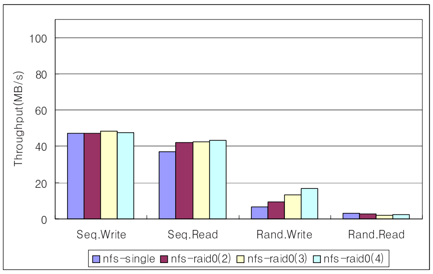
The Intel NetStructure 470T gigabit ethernet switch has been used, providing sufficient network bandwidth. The NetStructure has six 1000base-T ethernet

Table 2. Testbed specification

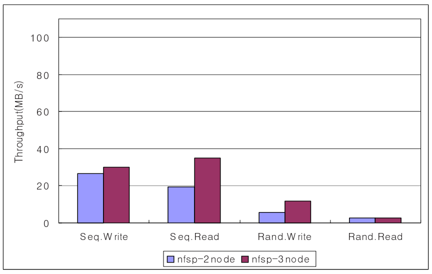
Processor	Intel Pentium 4 2.0GHz
Main Memory	256MB DDR SDRAM
HDD	IBM Deskstar 80GB ATA HDD
RAID Controller	Promise 20276 ATA RAID Controller
NIC	Intel 82540EM Gigabit ethernet adapter



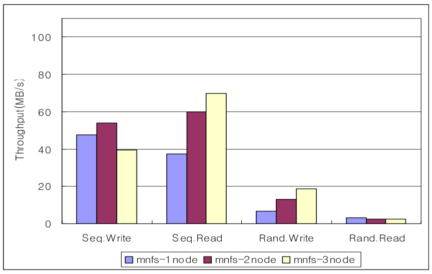
(a) Local disk



(b) Linux NFS



(c) NFSp



(d) mNFS

Fig. 2. I/O throughput

ports. Theoretically, it can support 21.2Gbps of internal bandwidth and forward about 1.4 million packets per second. Also, to support more efficient multicasting, it has the packet filter capability through *IGMP Snooping*[7].

In this evaluation, the throughput of the mNFS prototype is measured and compared with those of Linux NFS and NFSp. Comparison measures are file I/O throughput via NFS protocol and scalability when the number of cluster nodes or disks increases The mNFS and NFSp allocates a single disk to each cluster node. On the other hands, Linux NFS has a RAID level 0 disk array which has the same number of disks as cluster systems. IOzone[8] was used as a performance benchmark program. To exclude the effects of the difference in the total memory size between the experiments, size of experiment data sets are adjusted, that is, the size of data sets are three times of the total memory size.

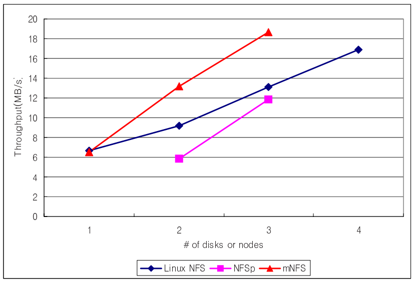


Fig. 3. Scalability of Random write

Table 3. Throughput of Linux NFS and mNFS when they contains 3 disks in total

	Linux NFS		mNFS	
Sequential Write	48.264	1	39.478	0.82
Sequential Read	42.567	1	69.648	1.64
Random Write	13.132	1	18.669	1.42
Random Read	2.11	1	2.557	1.21

In Figure 2(a) and 2(b), nevertheless the number of disks increases, a sequential I/O performance of Linux NFS is almost not changed. But the performance of random I/O is similar to that of local disk. Thus, a sequential I/O performance is limited by both gigabit ethernet and NFS protocol overheads.

For the convenience of implementation, NFSp is based on a user-level NFS server implementation. Due to frequent memory copy between user memory space and kernel memory space, it is difficult to expect good I/O performance. And it supports only NFS protocol version 2 where all file writes are handled synchronously to provide stability of data. Figure 2(c) shows that I/O performance improves when the number of server nodes or disks increases, But the performance of 3 nodes NFSp is lower than that of Linux NFS.

Figure 2(d) shows the result of mNFS experiments. In the case of sequential write, 2 nodes mNFS performed better than a single node. But 3 nodes mNFS performed worse than 2 nodes system and even a single node system. As the number of node increases, file write requests are prone not to be delivered in order. Thus sequential writes can not be processed sequentially at the I/O nodes and performance degradation can occur. In the case of random I/O, mNFS outperforms both Linux NFS and NFSp as shown in Figure 3. mNFS and NFSp seems to have similar scalability but the throughput of mNFS is much better than that of NFSp. Table 3 shows the throughput of Linux NFS and mNFS when they contains 3 disks in total. It is shown that mNFS provides better sequential write and read performance than Linux NFS by 18% and 64%, respectively. In random read and write case, mNFS performs better than Linux NFS by 21% and 42%, respectively.

As a whole, A mNFS cluster provides high scalability and good I/O performance.

4 Conclusions

The mNFS cluster system is a storage system which enhances the NFS that is an internet standard distributed file system. Through multicast RPC of client side and clustering of server side, the mNFS improves I/O performance and scalability. The mNFS cluster does not need any additional hardware. On the existing TCP/IP network, it can achieve good price/performance ratio. We show a prototype implementation of the mNFS. The mNFS does not have any dedicated control node, but it is controlled by autonomous co-operation between cluster nodes. Because the mNFS does not have any single-point-of-failure node, it can provide much higher availability. Although there are overheads of meta data update, the mNFS can improve file I/O performance and scalability. We expect that a mNFS system can be used successfully as a virtualized high-end storage system.

References

1. Lombard, P., Denneulin, Y.: nfsp: A distributed nfs server for clusters of workstations. In: Proc. of International Parallel and Distributed Processing Symposium. (2002)
2. Bhide, A., Engineer, A., Kanetkar, A., Kini, A.: File virtualization with directnfs. In: Proc. of the 19th IEEE Symposium on Mass Storage Systems and Technologies. (2002)
3. Kim, G.H., Minnich, R.G., McVoy, L.: Bigfoot-nfs: A parallel file-striping nfs server (1994)
4. Sun Microsystems, Inc.: RFC 1057: RPC: Remote procedure call protocol specification, version 2 (1988)
5. Fenner, W.: RFC 2236: Internet Group Management Protocol, version 2 (1997)
6. Bovet, D.P., Cesati, M.: Understanding the LINUX KERNEL. O'Reilly (2001)
7. Intel Corp.: Intel netstructure 470 switch user guide, 2nd edition (2001)
8. Norcott, W.: Iozone filesystem benchmark. URL: *http://www.iozone.org/* (1998)