pNFS for Everyone: An Empirical Study of a Low-cost, Highly Scalable Networked Storage

Taejin Kim and Sam H. Noh

Computer Engineering Department, Hongik University, Seoul, Korea

Summary
This paper presents empirical experience on a scalable, high-performance storage with two existing low-cost technologies. One technology is Ethernet-Attached Disks (E-AD) which are iSCSI-like disks but with low cost controllers. Another is Network File System (NFS) which is an open standard supported through open source. In particular, NFS version 4.0 (NFSv4.0) has been the de-facto distributed file system protocol for generations. Unfortunately, in this age of mass data, scalability of NFSv4.0 has become an issue. The NFS community has developed the pNFS protocol that takes advantage of parallel accesses to the pool of storage, and announced NFS version 4.1 including the pNFS protocol. We evaluated the scalability and performance of our storage system under various environments and benchmarks. The results show that NFSv4.1 has substantial benefits on large I/O units due to direct disk access and parallelism.

Key words:
Network File System, parallel NFS, E-AD, NFSv4.1

1. Introduction

This paper presents empirical experience showing that scalable, high-performance storage can be achieved by integrating two existing low-cost technologies. In particular, Ethernet-Attached Disks (E-AD), which are iSCSI-like disks but with low cost controllers, and NFS version 4.1 (NFSv4.1)[1,2,3,4], an open standard supported through open source, are used to provide scalable, high performance storage.

NFS version 4.0 (NFSv4.0)[5] has been the de-facto distributed file system protocol for generations. Unfortunately, in this age of mass data, scalability of NFSv4.0 has become an issue[6,7,8,9,10]. With NFSv4.0, as the server serves all client commands and data transfer, the performance is limited by the capability of the server. Furthermore, capacity scalability, that is, the capability to expand storage capacity as needed, is also limited by the storage infrastructure such as DAS (Direct-Attached Storage), NAS (Network-Attached Storage)[11], or SAN (Storage Area Network)[12].

pNFS[13,14,15], introduced as part of NFS version 4.1 (NFSv4.1) has been introduced to overcome the first limitation. NFSv4.1 separates the metadata and dataflow of client requests thereby allowing parallel access to simultaneously accessible storage disks. Unfortunately, pNFS still suffers from capacity scalability limitation as the underlying storage structure is still DAS, NAS, or SAN. Hence, pNFS is generally supported through proprietary implementations. Thus, exploiting the features of pNFS can become a costly endeavour.

The goal of this study is to share our experience in overcoming the performance and capacity scalability limitations of NFSv4.0 through the use of low-cost existing technologies. Through a comprehensive set of experiments that we conduct, we show that by using low-cost Ethernet attached disks, capacity can be expanded as needed at low cost, while obtaining performance scalability as capacity is added. The software platform that we use is based on the unofficial block-based pNFS that was released in Linux 3.XX.

More specifically, we use what we call E-AD (Ethernet-Attached Disk) devices as the disk storage subsystem for an NFSv4.1 setting as our experimental platform. The E-AD device, which was originally developed by Lim et al.[16], consists of a hard disk drive and an ASIC chip that bridges the off-the-shelf hard disk drive directly to the Ethernet. Major advantages of this type of E-AD device used in our experiments include (1) economy, which is a compelling factor in deciding the storage components of today's huge scale storage system where over 1,000TB of storage capacity is becoming the norm, (2) easy expandability, which is a must for the ever-growing storage needs that we are witnessing in today's cloud computing environment and Hadoop[17] ecosystems. Such E-AD devices allow practically unlimited number of disks to the NFSv4.1 system as long as Ethernet ports are available with comparable I/O performance at a fraction of the cost of SAN or iSCSI storage.

The rest of the paper is organized as follows. Section 2 briefly describes related work related to the technology of storage infrastructure and data management. Section 3 describes the E-AD technology and the workings of our system. Section 4 describes the experiment platform. Section 5 and Section 6 evaluate our system and show the

Manuscript received March 5, 2014
Manuscript revised March 20, 2014

results under various environments and benchmarks. Finally, we summarize and conclude this work in section 7.

2. Related Work

In this section, we discuss the storage system layers relevant to our work. We first discuss the storage infrastructure layer. Then, the data management layer is discussed. Finally, we discuss the status of pNFS, which is most relevant to our study.

2.1 Storage Infrastructure

Storage systems are composed of two levels of technology. One is the storage infrastructure used by the storage system. DAS (Direct-Attached Storage), NAS (Network-Attachted Storage)[11], SAN (Storage Area Network)[12] are typical forms of storage infrastructure that are used today. DAS, as its name implies, attaches storage disks directly to the local system. With DAS, the capacity can be expanded only to the number of device slots that are available within the system.

In contrast to DAS, NAS and SAN are storage connected through the network. A NAS unit is a storage appliance that provides file-based services to clients requesting service. NAS units are generally composed of a (fixed) number of disks and uses file-based protocols such as NFS[18] or CIFS[19]. Expanding NAS capacity can only be done in NAS units. Storage provided via SAN, on the other hand, provides block-based services generally using the Fibre Channel protocol[20], leaving the file management issue on the client side. Expanding the capacity of SAN storage is limited by the number of Fibre Channel switch ports, and may also require expanding the Fibre Channel network, which can be costly. Our current definition for a NAD (Network-Attached Disk) device includes all storage systems that exhibit the following properties: direct client-drive data transfer in a networked environment, asynchronous oversight by the high level filesystem, cryptographic support for the integrity of requests, storage self-management opportunities derived from a more abstract and independent role for storage systems, the ability to extend the feature set of a NAD for the purpose of application as well as for the client operating system.

2.2 Data Management

The other layer of technology involved with storage systems is the data management layer. Numerous experimental and real implementations for data management of mass distributed storage have been proposed. Lustre[21], PVFS2[22], Google File System[23], Ceph[24], GPFS[25] are some notable research and real world approaches. Each of these systems present unique approaches to sharing, moving, and integrating data stored in distributed storage systems. Due to their uniqueness (and their proprietory and high cost nature), portability is the main stumbling block for wide deployment of these systems.

In contrast, NFS (Network File System) is an open standard distributed file system protocol that has been widely accepted and deployed as an industry standard since the 1980's. Currently, NFS version 4.0 (NFSv4.0) is most widely deployed. However, with current trends of mass data production and usage, NFSv4.0 is having difficulty sustaining scalability in terms of performance and deployment as all service requests and data movement must pass through the NFS server of which the basic underlying storage infrastructure is either a NAS or a SAN. Hence, IETF (Internet Engineering Task Force), the open standards organization looking over Internet standards, proposed NFS version 4.1 (NFSv4.1) with the aim to support scalable parallel access to clustered storage servers for higher performance[1,2,3,4]. This form of parallel access support is also referred to as pNFS[13,14,15]. NFSv4.1 supports file-, block-, and object-based storage systems.

Figure 1 shows the organization of pNFS (parallel NFS). The key difference of pNFS from NFSv4.0 is its separation of file control and management from the data transfer. First, the NFS client and meta-data server (MDS), which holds the data layout information, communicates the control and management information of files through the pNFS protocol. The data layout information exchanged depends on the way the data is stored, i.e., file-, block-, or object-based. For example, in a system supporting block-based data layout, the starting block number and the number of bytes to transfer would be provided to the client. Once the client receives the data related information from the MDS, actual transfer of data is done in parallel in cooperation with the data servers (DS), independent of the MDS. NFS client-side data layout related drivers have been available in Linux since version 2.6.x. A few studies have been conducted in regards to NFSv4.1, with a focus on file-based storage systems[6,7,8,9,10]. Moreover, the source code of block-based pNFS server has not been supported on pNFS development site since version 3.4.

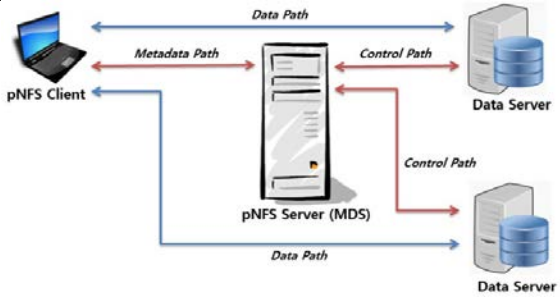

Fig. 1 Overview of pNFS protocol

2.3 Network Attached Disks

There are a number of network attached disks systems such as FC SAN [12], iSCSI[26], NASD[27], AoE[28], and FCoE[29]. E-AD used in our experiments distinguishes itself from others. E-AD implements its own re-transmission mechanism at its transport layer protocol without relying on TCP as in iSCSI in order to achieve higher effective data transfer rate by exploiting faster re-transmission. E-AD disk drive provides economic storage because it is not required to have a CPU or memory while most other network attached systems require a CPU and memory. E-AD adopts a virtual HBA (Host Bus Adapter) in the host computers, i.e., the device driver software installed in host computers provide a virtual HBA without having a physical HBA for E-AD. Although E-AD does not use a physical HBA, the E-AD system achieves high data I/O bandwidth by realizing I/O commands that transfer bigger chunks of data than the commands in ATA , SCSI, or FC. Since E-AD uses generic off-the-shelf Ethernet switches it has more applicability than others that are required to use special network hubs as in FC SAN.

3. Cost Effective, Scalable pNFS

3.1 E-AD Technology

The E-AD storage devices, originally developed by Lim et al.[16], are basically SATA hard disk drives directly attached to Ethernet through a bridging ASIC, as shown in Figure 2. The protocol used by this device can be regarded as a variation of iSCSI, Ethernet based SAN, or AoE (ATA over Ethernet). However, the particular proprietary protocol suite implemented here emphasizes economy and performance over iSCSI and AoE by not employing any DRAM or CPU and by adopting a fast and reliable packet re-transmission mechanism. The host computer, i.e. NFSv4.1 MDS and clients, issues I/O commands similar to those of iSCSI to the E-AD devices. The controller in the ASIC translates the host computer issued I/O commands into SATA I/O operation commands for the SATA disk drives, and vice versa.

The effective protocol of E-AD allows multiple hosts to issue I/O commands at the same time to a multitude of the E-AD devices in parallel. Since the host computers view the devices as their own local disks attached to their own internal bus, they issue block-level I/O commands directly to the devices as they would to local drives and all existing software can run without any modification to make use of the E-AD devices.

In addition, the fact that the protocol does not require an IP address for each individual device makes it possible to expand the storage space to practically an unlimited capacity as long as Ethernet ports are available to accommodate the devices.

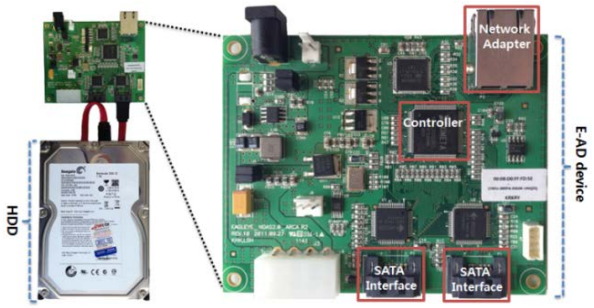

Fig. 2 E-AD storage device: SATA hard disk drives directly attached to Ethernet through a bridging ASIC

3.2 Overview of the Workings

Figure 3 shows the workings of our system. As shown in the figure, it is essentially a typical NFSv4.1 SAN configuration without the costly SAN Fibre Channel. The NFS server in the figure serves as the MDS and data is stored in E-ADs. The MDS uses the LVM (Logical Volume Manager) to configure the E-ADs into a configuration of choice, say RAID0. This configuration becomes a single logical volume, and the MDS exports this volume as a block device and a mount point to the clients.

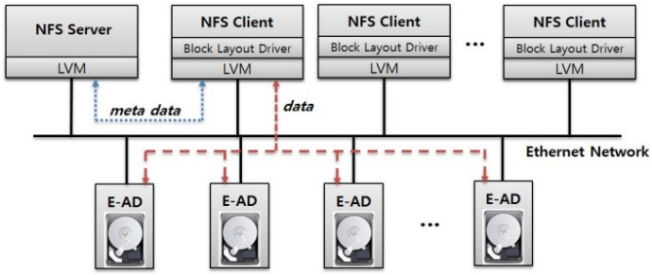

Fig. 3 Workings of our system

To store or retrieve data, the NFS client initially mounts the MDS exported volume. In so doing, the client generates a logical volume of its own based on the server exported configuration, whereupon the client sets up the device information. This set of device information is an extra step not found in NFSv4.0 and is used to directly access the devices from the client.

The read procedure of NFSv4.1 is as follows:

1. The client sends a read request (which includes the file descriptor, the start address, and length) to the MDS.
2. Based on the file inode information, the MDS replies with a layout that contains the device number and the set of extents, i.e. offset and length pair, that comprise the file.

3. Using the layout received, the client reads the disks directly based on the logical volume configuration information.

The write procedure of NFSv4.1 is as follows:

1. The client sends a write request (which includes the file descriptor, the start address, and length) to the MDS.
2. The MDS allocates (as consecutively as possible) the requested size in a minimal number of extents and sends the layout, which comprises the device number and the set of extents, to the client.
3. Using the layout received, the client directly writes to the disks based on the logical volume configuration information.
4. Upon completion, the client sends a commit to the MDS.
5. The MDS modifies its meta-data information based on the commit information.

Table 1: Settings of experimental environment

	Features
Meta-data Server	Linux 3.3.0, i3 CPU, 8G RAM, 1Gbs Network Adapter, HDD(500GB/7200rpm/16M Cache)
Client Machine	Linux 3.3.0, i7 CPU, 8G RAM, 1Gbs Network Adapter, HDD(3TB/7200rpm/64M Cache)
E-AD	SATA2 Interface ASIC HDD(3TB/7200rpm/64M Cache)
Network Switch	10/100/1000Mbps (autosensing)

4. Experimental Platform

In this section, we describe the platform in which all our experiments were performed. Table 1 shows the hardware platform with which the experiments were conducted. To support NFSv4.1, there is one meta-data server (MDS) and a maximum of 16 E-AD devices. The E-AD devices are configured in various ways as experiments are performed as we will describe in the next two sections. A maximum of 6 client machines, individually an i7 machine running Linux 3.3.0, generate the workloads to the storage system. A maximum 1Gbps network switch connects all the machines. The maximum network bandwidth of the network switch limits the throughput of the storage system. The software that we use in our experiments are originally the pNFS source code of the Linux kernel 3.3 maintained by Benny Halevy obtained from the kernel NFS git repository[30]. The downloaded source includes the block-based pNFS server code that was modified from the file-based pNFS server code spnfs developed by EMC[31].

However, this block-based server code has not been patched since kernel 3.4. The downloaded code, which had a few bugs, was debugged as needed. Specifically, we list a few significant problems that were corrected.

1) The storage system was not being properly recognized when two or more disks were configured as RAID0.

2) Originally, when the client requested a file write, the layout request was made only in 4K page sizes. This resulted in significant communication overhead between the MDS and the client that obviated the benefits of writing directly to the storage devices, especially when the write size was large. We modified this code to allow layout requests of any size that we desire.

3) Originally, the client periodically issued commits to the server during file writes to storage. This was a major bug that resulted in disastrous consequences. To elaborate, to write a file to storage, the client requests for a layout for a write of a particular size from the MDS. The MDS responses with one, and the client starts to write to disks. In the middle of writing, the client may decide to commit, sending the committed layout to the server. Upon receiving this committed layout, the server takes its own layout that it had originally sent to the client and modifies it according to the committed layout. This results in the client and server layouts being different; the client still has the original layout and the server has the new (partial) committed layout.

At this point, if the client tries to commit once again after writing more, the server simply ignores the request as the two layouts are not in sync. A similar problem arises when a read of the written data is requested. To remedy this problem, we modified the code so that a commit will happen only after the write is complete.

5. Scalability Experiments

In this section, we look into the scalability aspect of pNFS. In the following set of experiments, we conduct a series of copy operations (using the “cp” command) from the local disk to the E-AD devices, and vice versa. The results are denoted as “To E-AD” and “From E-AD”, the former referring to local disk to E-AD and the latter to E-AD to local disk copies. In the case of more than one E-AD device, the devices are configured as RAID0 so that the devices are accessed in parallel.

5.1 Effect of disk parallelism

Figure 4 shows the average throughput results (in MB/sec) as the number of E-AD devices is increased when copying 10 files each of size 1GB. Contrasting the results of copying to and from E-AD, we see that writing to E-AD shows better throughput. The main reason behind this is

because writes are asynchronous and acknowledgements arrive at the client even before the data is fully written to the device. The difference of overhead incurred by messages exchanged between the server and the client during read and write operation also influences these results. The results also show that throughput improves as the number of devices used, that is, parallelism, is increased saturating the network bandwidth with an only a few devices.

5.2 Effect of the number of clients

We repeat the experiments conducted in the previous subsection as we increase the number of client machines. Figure 5 shows the (aggregate and average) throughput as each client copies a 1GB file from (and to) local disk to (and from) 16 E-AD devices. The results show that the aggregate throughput scales with increased number of clients.

5.3 Effect of file size

Figure 6 shows the effect of the file size on performance when the number of E-AD devices is 16 starting from one 1MB file to one 10000MB file. Each data point is reported is an average of five executions with the system rebooted after each execution. We see that the throughput increases as the file size increases saturating at maximum network throughput.

5.4 Effect of the number of files

We now take a 1MB file and increase the number of files to see the effect of the number of files on performance. In this experiment the lone client sequentially copies the specified number of 1MB files to (from) the 16 E-AD devices from (to) the local disk. The results are presented in Figure 7. We see that with the increase of the number of files, there is no deterioration of performance even though the MDS is consulted for each data service request. However, we also see that the performance does not saturate the network bandwidth as each file is only 1MB.

6. Benchmark Performance

In this section, we present results for the Filebench and FIO benchmarks. We present results comparing NFSv4.1 with NFSv4.0. The experimental environment is the same as the scalability experiments, except that there is only one client.

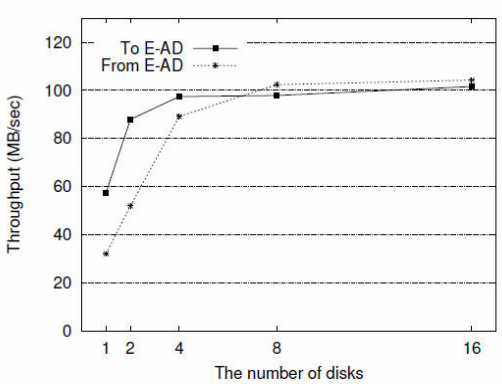

Fig. 4 Disk scalability

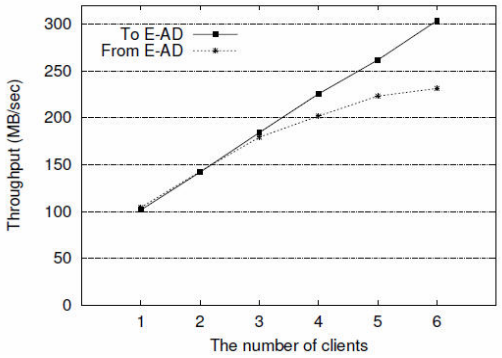

Fig. 5 Client scalability

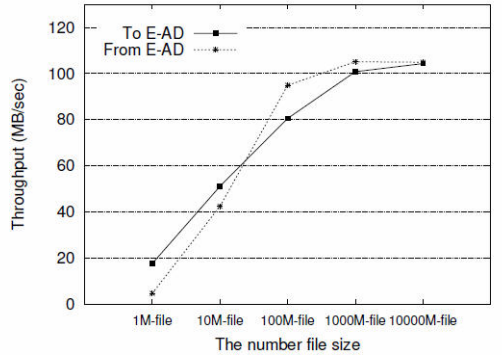

Fig. 6 File size scalability

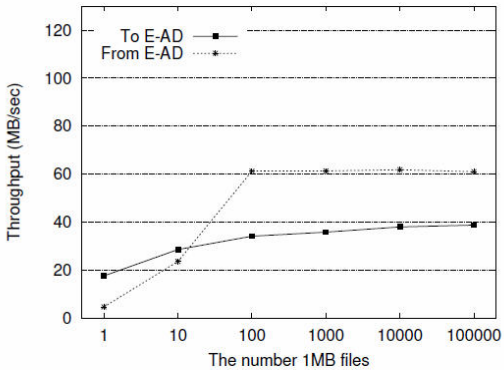

Fig. 7 Number of files scalability

6.1 Filebench

Filebench is a file system and storage benchmark that generates a large variety of workloads. Table 2 shows the feature of the workloads used in our experiments.
Figure 8 shows substantial performance improvements with NFSv4.1 on fileserver and videoserver workloads using mass data. This result shows that the bigger I/O unit allows for greater benefits for pNFS as it increases the effectiveness of direct disk access. The results for workloads with many small reads are different in that NFSv4.0 performs better or in par with NFSv4.1. The first reason for this is that in case of NFSv4.0, most I/O requests are serviced through the cache of the local file system in the NFS server because the I/O unit is small. Another reason is that the effectiveness of direct disk access in NFSv4.1 is small because the I/O unit is small.

Table 2: Filebench workloads

Workload	Mean file size	Read: write ratio
Fileserver	128K	1:2
Varmail	16K	1:1
Webserver	16K	10:1
Videoserver	1G	Read only

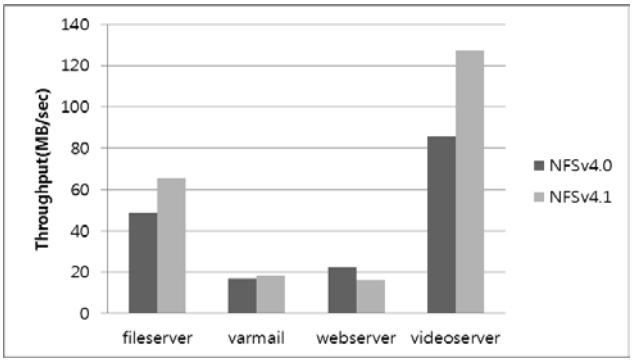

Fig. 8 Filebench benchmark

6.2 FIO

FIO is an I/O tool meant to be used both as a benchmark and for stress/hardware verification. In this benchmark, we conducted sequential write/read and random write/read. Figures 9, 10, 11 and 12 show that the performance of NFSv4.1 is better than NFSv4.0, except for random read requests smaller than 128Kbytes. The reason is similar to the case of workloads with many small reads for the Filebench benchmark.

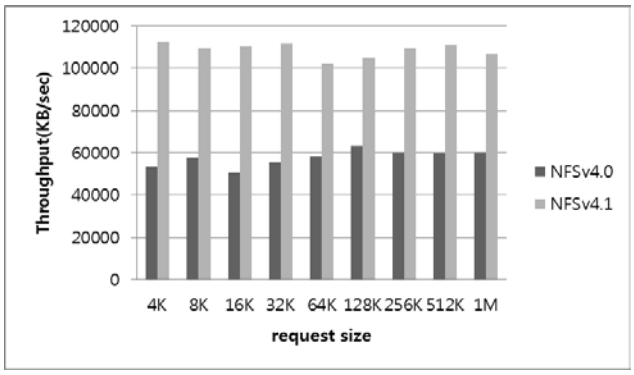

Fig. 9 FIO benchmark: sequential write

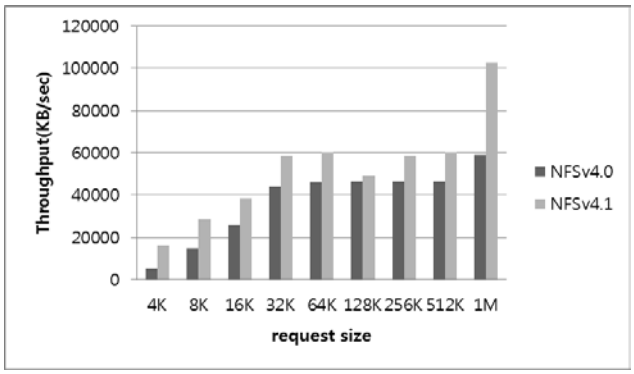

Fig. 10 FIO benchmark: random write

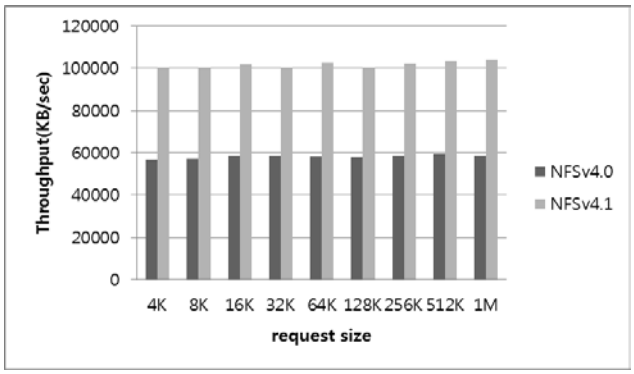

Fig. 11 FIO benchmark: sequential read

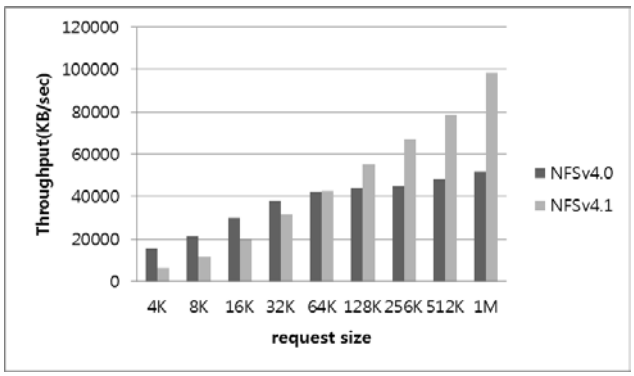

Fig. 12 FIO benchmark: random read

7. Summary and Conclusion

In this work, we implemented and studied the feasibility of a cost effective large scale disk storage system with E-AD and pNFS technology. E-AD device has the feature of cheaper and easier installation than other network disk devices such as NAS and SAN. The scalability and performance of our storage system was shown through experiments. The results showed that NFSv4.1 has substantial benefits for large I/O units due to direct disk access and parallelism. In other words, the results show that the NFSv4.1 server is more suitable for large scale I/O servers such as video streaming servers and cloud storage servers.

For future work, we plan to reinforce the NFSv4.1 protocol. The first reinforcement is to have NFSv4.1 act as NFSv4.0 for small sized requests. Through this study, we found that the weakness of NFSv4.1 is in servicing small requests. We plan to modify NFSv4.1 so that the client is able to select NFSv4.0 or NFSv4.1 depending on the request size. The second reinforcement is to have NFSv4.1 work for various disk array configurations. The main feature of NFSv4.1 is direct disk access and parallelism. The disk configuration of current NFSv4.1 can only use RAID0. We plan to modify NFSv4.1 so that it can work on various disk array configurations.

Acknowledgments

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2012R1A2A2A01045733).

References

[1] S. Shepler, M. Eisler, and D. Noveck, “Network File System (NFS) Version 4 Minor Version 1 Protocol,” 2010. RFC5661.
[2] S. Shepler, M. Eisler, and D. Noveck, “Network File System (NFS) Version 4 Minor Version 1 External Data Representation (XDR) Description,” 2010. RFC5662.
[3] D. Black, S. Fridella, and J. Glasgow, “Parallel NFS (pNFS) Block/Volume Layout,” 2010. RFC5663.
[4] B. Halevy, B. Welch, and J. Zelenka, “Object based Parallel NFS (pNFS) Operations,” 2010. RFC5664.
[5] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, “Network File System (NFS) Version 4 Protocol,” 2003. RFC3530.
[6] D. Hildebrand, L. Ward, and P. Honeyman, “Large files, small writes, and pNFS,” in Proceedings of the 20th Annual International Conference on Supercomputing, ICS ’06, (New York, NY,USA), pp. 116-124, ACM, 2006.
[7] D. Hildebrand and P. Honeyman, “Exporting Storage Systems in a Scalable Manner with pNFS,” in Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, MSST ’05, (Washington, DC, USA), pp. 18-27,IEEE Computer Society, 2005.
[8] D. Hildebrand, P. Honeyman, and W. A. A. Adamson, “pNFS and Linux: Working Towards a Heterogeneous Future,” in Proceedings of 8th LCI International Conference on High-Performance Cluster Computing, 2007.
[9] D. Hildebrand and P. Honeyman, “Direct-pNFS: scalable, transparent, and versatile access to parallel file systems,” in Proceedings of the 16th International Symposium on High Performance Distributed Computing, HPDC ’07, (New York, NY, USA), pp. 199-208, ACM, 2007.
[10] L. Chai, X. Ouyang, R. Noronha, and D. K. Panda, “pNFS/PVFS2 over Infiniband: early experiences,” in Proceedings of the 2nd International Workshop on Petascale Data Storage: held in conjunction with Supercomputing ’07, PDSW ’07, (New York, NY, USA), pp. 5-11, ACM, 2007.
[11] NAS. http://en.wikipedia.org/wiki/Network-attached_storage.
[12] SAN. http://en.wikipedia.org/wiki/Storage_area_network.
[13] B. Welch, B. Halevy, D. Black, A. Adamson, and D. Noveck, “pNFS Operations Summary.” Internet Draft, draft-welch-pnfsops-00.txt, 2004.
[14] G. Gibson, B. Welch, G. Goodson, and P. Corbett, “Parallel NFS Requirements and Design Considerations.” Internet Draft, draftgibson-pnfs-reqs-00.txt, 2004.
[15] Parallel Network File System. http://www.pnfs.com/.
[16] H.-K. Lim, J.-H. Han, and D.-K. Jeong, “A Network Storage LSI Suitable for Home Network,” IEEE Communications Magazine, vol. 43, no. 5, pp. 141-148, 2005.
[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), MSST ’10, (Washington, DC, USA), pp. 1-10, IEEE Computer Society, 2010.
[18] Sun Microsystems Inc., NFS: Network File System Protocol Specification, 1989. RFC1094.
[19] Common Internet File System. http://technet.microsoft.com/en-us/library/cc939973.aspx.
[20] FC. http://en.wikipedia.org/wiki/Fibre_Channel.
[21] Lustre a network clustering FS. http://wiki.lustre.org/.
[22] Parallel Virtual File System -Version2.” http://www.pvfs.org/pvfs2.
[23] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” in Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP ’03, (New York, NY, USA), pp. 29-43, ACM, 2003.
[24] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, “Ceph: a scalable, high-performance distributed file system,” in Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI ’06, (Berkeley, CA, USA), pp. 307-320, USENIX Association, 2006.
[25] F. Schmuck and R. Haskin, “GPFS: A Shared-Disk File System for Large Computing Clusters,” in Proceedings of the 1st USENIX Conference on File and Storage Technologies, FAST ’02, (Berkeley, CA, USA), USENIX Association, 2002.

[26] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)," 2004. RFC3720.
[27] G. A. Gibson, D. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka, "File Server Scaling with Network-Attached Secure Disks," in SIGMETRICS, pp. 272-284, 1997.
[28] AoE. http://en.wikipedia.org/wiki/ATA_over_Ethernet.
[29] FCoE. http://en.wikipedia.org/wiki/Fibre_Channel_over_Ethernet.
[30] B. Halevy, "Linux pNFS server development." git://git.linux-nfs.org/projects/bhalevy/linux-pnfs.git.
[31] D. Muntz, M. Sager, and R. Labiaga, "spNFS: A Simple pNFS Server." http://www.con-nectathon.org/talks08/dmuntz-spnfs-cthon08.pdf.
[32] Filebench benchmark. http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Main_Page.
[33] FIO benchmark. http://git.kernel.dk/?p=fio.git;a=summary/.

Taejin Kim received the B.S. degree in computer engineering from Hongik University in 1998, and the M.S. degree in computer science from Hongik University in 2000. He has been in the Ph.D. program at Hongik University since 2010. His areas of research include networked storage systems and high speed large scale big data systems.

Sam H.(Hyuk) Noh received the BS degree in computer engineering from the Seoul National University, Korea in 1986, and the PhD degree from the Department of Computer Science, University of Maryland at College Park in 1993. He is a Professor in the School of Information and Computer Engineering. His current research interests are in operating system issues pertaining to embedded/computer systems. (Refer to http://next.hongik.ac.kr for details.) Dr. Noh is a member of the IEEE, the ACM, USENIX, and KIISE.