

Backup Server Using Direct I/O Scheme

Kyung Woo HUR*, Won Vien PARK*, Wan Yeon LEE**, Jin KIM*, Young Woong KO*

**Department of Computer Engineering, Hallym University, Chuncheon-si, Gangwon-do, Korea*

***Department of Ubiquitous Computing, Hallym University, Chuncheon-si, Gangwon-do, Korea*

{rapperkw, wonvien, wanlee, jinkim, yuko}@hallym.ac.kr

Abstract— Recently the study of network backup server has developed rapidly and its achievements have become a center of attraction. In this paper we focus on I/O subsystem performance analysis for network backup server. The key issue is to analyse the trade-off between direct I/O and buffered I/O mechanism. We investigate how the two mechanisms affect overall performance of network backup server. We designed and implemented the backup server and conducted several experiments. Experimental result shows overall performance of direct I/O method outperforms traditional buffered I/O mechanism.

Keywords— Backup, Server, Direct I/O, Raw I/O, O_DIRECT

I. INTRODUCTION

Recent advances in Internet technology has led to a convenient network backup, and it has replaced traditional tape backup method. The network backup systems are widely used in current storage market. However, current network backup server has drawbacks.

First, it cannot deal with the large volumes of duplicated files efficiently. The duplicated files waste the capacity of storage devices and network bandwidth. Therefore, to improve the performance of backup server, we have to deal with eliminating duplicated files. Moreover, data duplication problem can be extended to block level because there exist duplicated blocks in a file. The easiest way to find duplicated file is to exploit file metadata information such as file name, size, and date. With this information, we can easily find duplicated version of files in a backup server. There is more powerful mechanism to find duplicated data called hashing such as MD5 [1] and SHA1 [2]. We can easily confirm whether two files are identical or not by simply comparing two hash data. Second, the performance of network backup server is not satisfactory. It is well-known that storage system causes a significant delay for processing I/O requests because almost all storage systems are based on mechanical device. To increase the performance of network backup server, we have to minimize disk I/O latency.

To address these problems, we adapted direct I/O scheme on network backup system to reduce I/O overhead. The well-known direct I/O schemes are raw disk I/O and O_DIRECT which can bypass kernel buffer cache. Generally, if we are performing a backup procedure, the data blocks are transferred to kernel buffer cache from user application memory, and then the data blocks in kernel buffer cache are moved to hard disk

at later. We believe that network backup server does not need to exploit kernel buffer cache because the cached data blocks are rarely reused in backup system. Therefore, direct I/O schemes are much more efficient for network backup server.

In this paper, we described how to reduce I/O latency for network backup server using direct I/O schemes. We analysed the performance of network backup server using direct I/O and compared the performance with traditional buffered I/O scheme. We also conducted several performance experiments to show the detailed performance results.

II. RELATED WORKS

IO-Lite [3] unifies buffering and caching in the general-purpose operating system to prevent duplicated buffer access and minimize data copy. In particular, it allows applications, interprocess communication, the file cache, the network subsystem, and other I/O subsystems to share a single physical copy of the data with safe. IO-Lite can avoid redundant data copy (decreasing I/O overhead) and multiple buffering (increasing effective file cache size). It permits performance optimizations across subsystems (e.g., application-specific file cache replacement and cached Internet checksums).

PVFS is designed as a client-server system in which files are transparently stripped across disks of multiple file servers [4]. PVFS provides concurrent read/write operations from multiple processes or threads, and also distributes strip file to each I/O nodes. PVFS provides good performance but it lacks data consistency when several processes interact with the same files.

A Mosix is a UNIX based cluster operating system that supports preemptive process migration for load-balancing and memory ushering [5]. Mosix supports splitting files to several nodes. It can deliver a high I/O performance by migrating parallel processes to the respective nodes that hold the data. Especially large file is preloaded to the main memory of the clusters nodes.

GPFS is a parallel file system for cluster computers that provides, as closely as possible, the behavior of a general-purpose POSIX file system running on a single machine [6]. GPFS achieves scalable through its shared-disk architecture. GPFS system consists of the cluster nodes, on which the GPFS file system and the applications that use it run, connected to the disks or disk subsystems over a switching fabric.

The Portable Parallel File System (PPFS) allows applications to control caching, pre-fetching, data distribution and file sharing policies [7]. Files are divided into variable size records, called segments. Each segment is managed by a single I/O server. It is implemented as a user level library portable across several parallel file systems.

Cluster file uses the same data representation for file layouts, access patterns, and the mappings between each other [8]. The Key issue of Cluster file is as follows; Parallel applications can physically partition a file in arbitrary patterns. They can also set arbitrary views on a file. Views hide the parallel structure of the file and ease the programmer's burden of computing complex access indices. The intersections between views and layouts are computed by a memory redistribution algorithm. Read and write operations are optimized by pre-computing the direct mapping between access patterns and disks.

NFSv4 departs from the stateless model to support exclusive opens called share reservations, mandatory locking, and file delegations [9]. The NFSv4 server must store some information about clients, users, and files, as well as information about the outstanding share reservations, locks, and delegations. The need to manage consistency of state information on multiple nodes fetters NFSv4's ability to export an object via multiple servers. Partitioning the file space among multiple NFS servers can work around this limitation to an extent, but increases management cost and fails to address scalable access to a single file or directory.

III.SYSTEM DESIGN

Almost all UNIX systems support buffered disk I/O and raw disk I/O mechanism. Buffered disk I/O mechanism is used commonly in UNIX environment that utilize kernel buffer cache for disk I/O. It settles disk latency using the kernel buffer cache. When there are block requests, the data block copied to kernel buffer cache from user-level memory. If there fill enough data blocks on buffer cache or there is a flush request from application, then volumes of data blocks are writing on the disk storage.

Raw disk I/O writes data blocks from user buffer to disk storage directly by bypassing kernel buffer cache. I/O subsystems designed with specific purpose can use its own buffer cache mechanism. This specific subsystem can manage disk I/O policy for its purpose. The well-known example of direct I/O is DBMS (Database Management System). Generally, to copy the volume of data blocks to buffer cache cause high CPU overhead and limits throughput of server. Also, it consumes memory therefore buffered I/O can cause increasing memory waste. The Raw disk I/O scheme does not use kernel buffer cache because it writes directly from user buffer to disk storage. The advantage of raw disk I/O is to reduce the number of data copy to kernel buffer cache and overhead by data copy. But raw disk I/O cannot exploit file system semantic; it should have file management policy and mechanism. The DBMS, for example, has its own buffer cache, so it can avoid repeated use of its buffer cache and kernel buffer cache. Therefore, the performance of DBMS is

better than using kernel buffer cache. Figure 1 shows the difference between Buffered I/O and Raw I/O scheme.

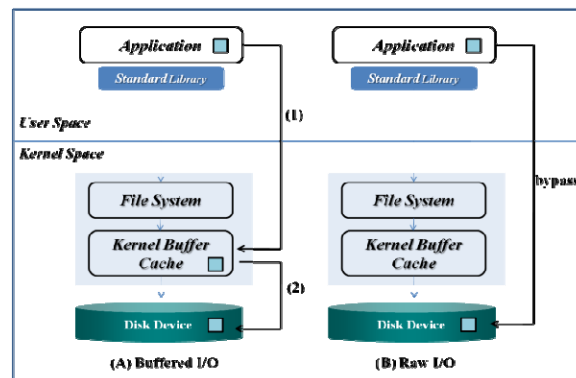


Figure 1. Buffered I/O vs. Raw I/O

As you can see in figure 1-(a), in buffered I/O environment, application access disk drive through buffer cache however, in figure 1-(b), application directly access disk drive for processing disk I/O with raw I/O mechanism. To use direct disk I/O scheme for easy way, we can use O_DIRECT flag since kernel 2.4(Linux kernel version). With O_DIRECT flag, application writes on the disk storage directly without using kernel buffer, so, it is more convenient way than implementing raw disk I/O. We can use direct I/O scheme by adding O_DIRECT flag when we try to use file open system call.

A. Text Font of Entire Document

The proposed system is composed of three main modules: client, MDS(Meta-data server) and backup server. The behaviour of proposed system is as follow. In the client module, the file is divided into fixed size blocks (about 16 Kbytes) and the each block is converted to hash data (about 160 bit). The hash data is grouped to hash list and then transmitted to MDS. The MDS has several roles. The MDS checks duplication blocks by examining overall hash list in hash database, and generates duplication block list. The result is transmitted to the client reversely. Finally, the client sends non-duplicating blocks to backup server. The backup server write data blocks to disk with direct I/O scheme.

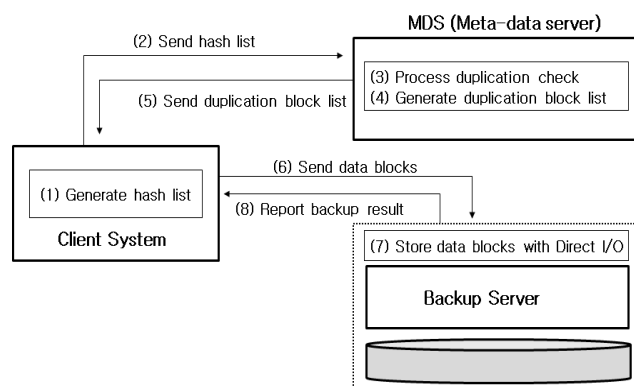


Figure 2. The layout of proposed architecture

In this work, we designed and implemented Deduplication based backup system to reduce data storage capacity. The Deduplication scheme is widely used in backup system [11,12,13,14,15,16]. The client module provides an interface for selecting backup files and directory for a user. If a user selects the file or directory, then the backup task is scheduled on a specified time or immediately. Figure 3 shows the proposed system's client interface.

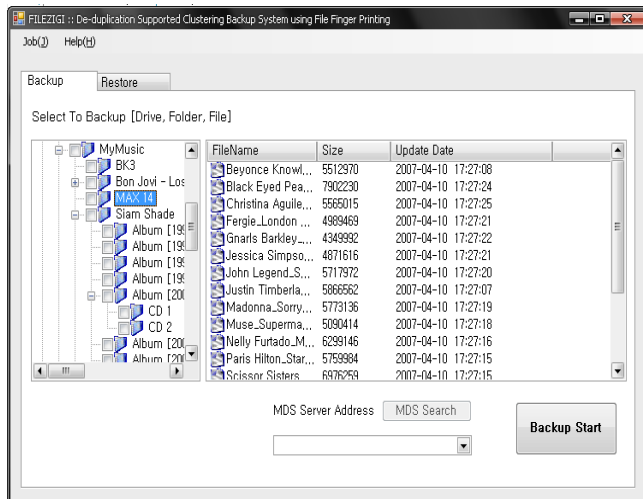


Figure 3. User interface for backup system

While a backup is processing, the client module creates a file list which includes backup files, directories and hash data. The hash data is generated by a SHA1 hash algorithm that processes about 80 rounds arithmetic computation (add, and, or, xor, rotl, etc) for a one block data. The computation result of hash algorithm is a 160 bit binary data and stored with a plain hex code in the file list. Figure 3 shows the deduplication steps in the proposed system.

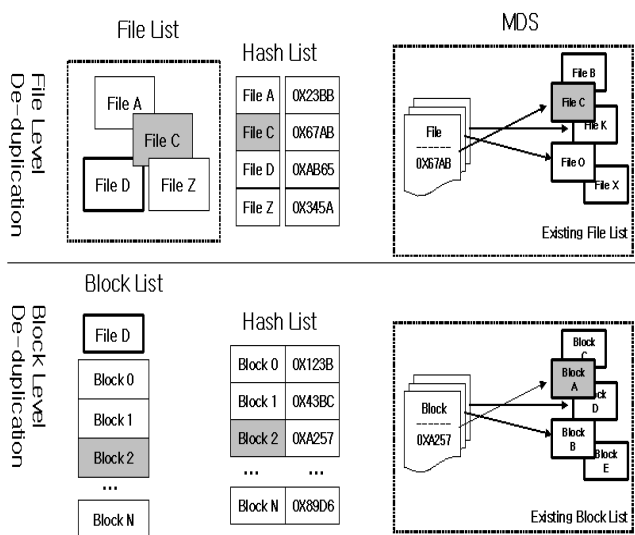


Figure 4. Deduplication mechanism for backup server

In figure 4, with the file and hash list, we can find duplicated files in MDS, therefore we actually send backup

files except "File C". In this manner, a file-level deduplication reduces the number of files to be backed up. Into the bargain, we can reduce more data blocks by block-level de-duplication mechanism. For example, although "File D" has to be transferred to file server, not all blocks are transferred. The "File D" is composed of lots of blocks and there exists identical blocks with existing blocks in MDS.

B. Backup Server Procedure

We have implemented the backup system with direct I/O scheme and buffered I/O. The overall backup procedure is as follows.

- 1) The client creates file finger print with SHA1 hash function.
- 2) File finger print is delivered to MDS, which process duplication check for the file and send the duplication result to MDS server.
- 3) The client divides the non-duplication file to several fixed blocks, creates block hashes for each block and finally sends the result to MDS. The MDS also checks duplication for each block and send duplicate block list to client.

The client the non-duplicate blocks to cluster node.

The backup server receives data blocks and writes to disk using direct I/O.

- 1) If user wants to retrieve backup data, we have to restore original file from backup server. The detailed restore procedure is as follows.
- 2) The user request file restore with the specified option list such as backup date/time, file name and directory.
- 3) The client request meta-data for restoring file/directory. The MDS construct the restore information include directory and file list and send the result to the client.
- 4) The client retrieves blocks of the file to be restored and send to client. If all the blocks are transmitted, the client assembles the blocks and creates the restore file.

IV. EXPERIMENT RESULT

In this paper, we try to show the effectiveness of direct I/O scheme for backup server. Therefore we compared Direct I/O based backup module to buffered I/O module. As you can see in the Figure 5, we experiment the performance of existing method such as Buffered I/O, Raw I/O, and O_DIRECT I/O for find suitable I/O method which can improve network backup server. When backup system is working, we record the log of system behaviours such as CPU and I/O. The experiment environment of this paper organize a Pentium 4 3.0 GHz, 512 MB memory, 160GB HDD, two 250GB HDD, and operating system is CentOS 5.2.

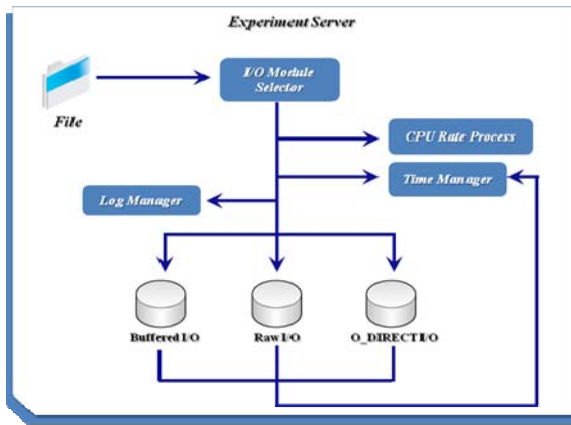


Figure 5. Architecture of experiment server

To get correct experiment result, we measure each I/O works execution start and end time and calculate elapsed time. Also, we measure average CPU rate of use during progress I/O operation and iowait value.

- 1) Experiment on Buffered disk I/O using typical kernel buffer cache in UNIX.
- 2) Experiment on raw disk I/O that record directly on disk storage without kernel buffer cache.
- 3) Experiment on disk I/O using O_DIRECT flag that support since kernel 2.4.

In this experiment, we used 4Gbyte file as backup data. And we varied the buffer size from 512byte to 4Mbyte and measure execution time and average CPU rate. We repeated five times for correct result and we rebooted the backup system to initialize memory system. This prevents buffer cache from being effected by previous experiment.

We show the CPU rate of each I/O scheme. The system CPU rate reflects how much works was done in kernel mode and the user CPU rate reflects how much works was done in user mode, respectively.

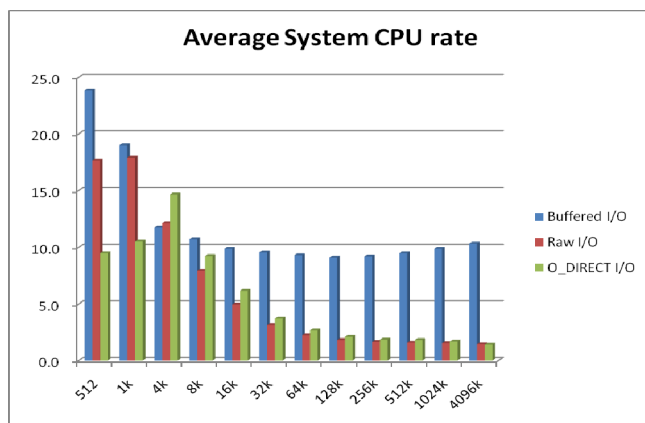


Figure 6. Average CPU rate (kernel mode)

In Figure 6, Buffered I/O shows that it consumes about 10 % of CPU work in kernel mode when it performs disk I/O. However, Raw I/O and O_DIRECT I/O shows 3% of CPU

work. The kernel has to copy data blocks from user mode to kernel mode, which is usually processed by CPU memory copy instruction. The direct I/O scheme does not need to copy user data blocks to kernel buffer, therefore the kernel can be user in more idle state than buffered I/O scheme. We can conclude that buffered I/O is more CPU consuming scheme in kernel mode compared to raw disk I/O and O_DIRECT I/O.

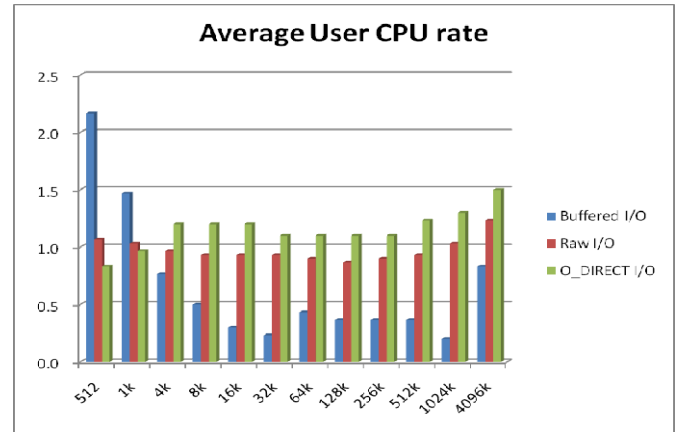


Figure 7. Average CPU rate (user mode)

In figure 7, the experiment result shows the CPU load of user mode when we process disk I/O job with each I/O scheme. The backup server has to control disk drive directly without kernel intervention. To process disk I/O, backup system have to use *mmap* system call and copy data blocks with DMA mechanism. This procedure needs CPU work therefore the overall CPU load in user mode is higher than buffered I/O. However, the difference of CPU load is only 1 %. Therefore, buffered I/O scheme consumes more CPU work than direct I/O scheme.

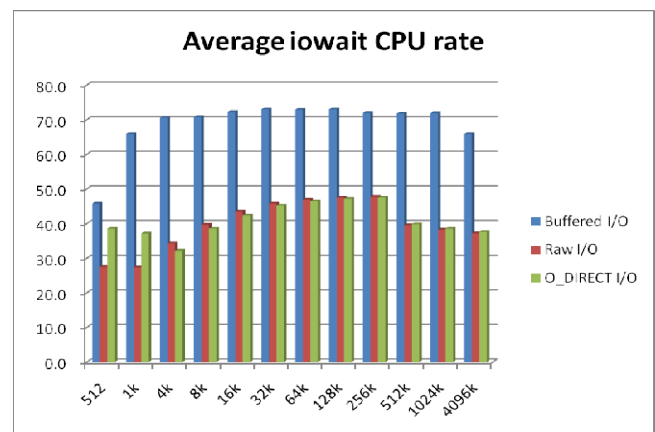


Figure 8. Average iowait CPU rate

Figure 8 shows average iowait values for each I/O scheme. As you can see in this figure, buffered I/O scheme shows high iowait value. The experiment result explains that the network backup server with buffered I/O is much more overloaded than direct I/O.

V. CONCLUSIONS

In this paper, we focus on I/O subsystem performance analysis for network backup server. The key issue is to analyse the trade-off between direct I/O and buffered I/O mechanism. We investigate how the two mechanisms affect overall performance of network backup server. The reason why we adapt direct I/O is that backup server usually access disk for writing only, so kernel buffer cache doesn't anything for performance improvement. With direct I/O scheme, the number of data copy times can be reduced, which contributes to the overall backup server performance. This approach enables us to use storage capacity. We conducted several experiments including I/O processing time, average CPU rate and iowait to measure various aspects of network backup server. Experiment result shows overall performance of direct I/O method outperforms traditional buffered I/O mechanism.

ACKNOWLEDGMENT

This research was financially supported by the Ministry of Education, Science Technology (MEST) and National Research Foundation of Korea(NRF) through the Human Resource Training Project for Regional Innovation and by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0016143).

REFERENCES

- [1] R. L. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, Internet Activities Board, 1992.
- [2] RFC 3174, "US Secure Hash Algorithm 1 (SHA-1)"
- [3] VIVEK S. PAI, PETER DRUSCHEL, and WILLY ZWAENEPOEL "IO-Lite: A Unified I/O Buffering and
- [4] Caching System", ACM Transactions on Computer Systems, Vol. 18, No. 1, February 2000
- [5] P.H. Carns, W.B. Ligon III, R.B. Ross and R. Thakur. "PVFS: A Parallel File System for Linux Clusters." Proc. 4-th Annual Linux Conference, Atlanta, GA, 2000.
- [6] L. Amar, A. Barak, A. Eizenberg and A. Shiloh. The MOSIX Scalable Cluster File Systems for Linux. <http://www.MOSIX.org>, July 2000.
- [7] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters" First USENIX Conference on File and Storage Technologies (FAST'02), Monterey, CA, January 28-30, 2002.
- [8] James V. Huber, Cristopher L. Elford, Daniel A. Reed, Andrew A. Chien, David S. Blumenthal, "PPFS: A High Performance Portable File System." In Proceedings of the 9th ACM International Conference on Supercomputing, Barcelona 1995.
- [9] F. Isaila and W. Tichy, "Clusterfile: A flexible physical layout parallel file system." Third IEEE International Conference on Cluster Computing, 2001.
- [10] Peter Honeyman and Dean Hildebrand and Dean Hildebrand. "Scaling NFSv4 with Parallel File Systems." in Proc. of Cluster Computing and Grid, 2005.
- [11] KyoungSoo Park, Sunghwan Ihm, Mic Bowman, and Vivek S. Pai, Supporting Practical Content-Addressable Caching with CZIP Compression, In Proceedings of the USENIX Annual Technical Conference, Santa Clara, CA, June 2007
- [12] C. Polcironiades and I. Pratt, Alternatives for detecting redundancy in storage systems data, In Proceedings of USENIX Annual Technical Conference, 2004
- [13] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson, Reliability and security in the CoDeeN content distribution network, In Proceedings of the USENIX Annual Technical Conference, 2004
- [14] D. Bobbarjung, Suresh Jagannathan, C. Dubnicki. Improving Duplicate Elimination in Storage Systems, ACM Transactions on Storage, November 2006
- [15] Purushottam Kulkarni, Fred Douglass, Jason LaVoie, and John M. Tracey, Redundancy Elimination Within Large Collections of Files, In Proceedings of 2004 USENIX Technical Conference, Boston, Massachusetts, USA, 2004.
- [16] Satoshi, N. Mutsuo, S. Katsuo, I. "Realtime Network Backup to Existing Servers Based on Stackable Filesystem," International Conference on Digital Object Identifier, pp 415-420, 2009.