

Implementation and Evaluation of EXT3NS Multimedia File System

Baik-Song Ahn[†], Sung-Hoon Sohn[†], Chei-Yol Kim[†], Gyu-Il Cha[†],
Yun-Cheol Baek[‡], Sung-In Jung[†], Myung-Joon Kim[†]

[†] Electronics and Telecommunications
Research Institute
Daejeon, 305-350 Korea
+82-42-860-1660
{bsahn, sonsh, gauri, gicha, sijung,
joonkim}@etri.re.kr

[‡] Division of Computer Software
Sangmyung University
Seoul, 110-743 Korea
+82-2-2287-5074
ybaek@smu.ac.kr

ABSTRACT

The EXT3NS is a scalable file system designed to handle video streaming workload in large-scale on-demand streaming services. It is based on a special H/W device, called Network-Storage card (NS card), which aims at accelerating streaming operation by shortening the data path from storage device to network interface. The design objective of EXT3NS is to minimize the delay and the delay variance of I/O request in the sequential workload on NS card. Metadata structure, file organization, metadata structure, unit of storage, etc. are elaborately tailored to achieve this objective. Further, EXT3NS provides the standard API's to read and write files in storage unit of NS card. The streaming server utilizes it to gain high disk I/O bandwidth, to avoid unnecessary memory copies on the data path from disk to network, and to alleviate CPU's burden by offloading parts of network protocol processing. The EXT3NS is a full functional file system based on the popular EXT3. The performance measurements on our prototype video server show obvious performance improvements. Specifically, we obtain better results from file system benchmark program, and obtain performance improvements in disk read and network transmission, which leads to overall streaming performance increase. Especially, the streaming server shows much less server's CPU utilization and less fluctuation of client bit rate, hence more reliable streaming service is possible.

Categories and Subject Descriptors

D.4.3 [Operating Systems]: File Systems Management – *access methods, Directory structures, Distributed file systems, File organization.*

General Terms

Measurement, Performance, Design, Experimentation

Keywords

Streaming, File system, Multimedia, Video server

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MM'04, October 10–16, 2004, New York, New York, USA.
Copyright 2004 ACM 1-58113-893-8/04/0010...\$5.00.

1. INTRODUCTION

In this paper we develop a scalable streaming server architecture that supports large-scale real-time multimedia service which demands tremendous I/O bandwidth requirement. Considering state-of-the-art streaming server architecture, the streaming server must have the following features: (1) increase in disk I/O bandwidth, (2) data transmission at gigabit-speed with low CPU overhead, (3) minimizing the number of memory copies while reading data from disk and transmitting it via network. For this purpose, we introduce a *streaming acceleration* H/W device called Network-Storage card (NS card). NS card consists of a disk controller (with disks), TCP Offload Engine (TOE) network interface, a memory controller (with PCI memory called PMEM), and a PCI bridge that connects major components of the card to system I/O bus. Software modules related to disk controller and network interface can directly access PMEM in order to avoid unnecessary memory copies.

We also design and implement a new file system called EXT3NS. The EXT3NS is a file system built on top of NS card. It enables applications to access fast-path I/O of NS card via standard read and write system call interfaces. It provides legacy VFS layer interface for existing file system related applications as well. The file system can easily accommodate large block size defined by disk striping driver of NS card. It has a disk layout to efficiently store large numbers of multimedia files as well.

A lot of efforts were made to optimize the data path from disk to network in networked file servers. [2] proposed and implemented some enhancements to 4.4 BSD compliant UNIX for throughput improvements and QoS guarantee on the data path from the disk to network in Multimedia-On-Demand(MOD) storage servers of MARS project. [11] proposed a mechanism to provide kernel support for data relaying operations in order to reduce context switching and data copying across protection boundaries. Modern UNIX clone operating systems provide a system call, called *sendfile*, to eliminate data copies between user mode and kernel mode [1].

Earlier works in multimedia streaming server have focused mainly on guarantee of real-time retrieval, buffering and caching of multimedia data, etc. Since legacy disk scheduling algorithms do not provide bandwidth guarantee, it is not possible to provide continuous flow of data blocks from the disk to the end system. A number of researches address these issues and propose the disk scheduling algorithms for multimedia data retrieval [3][9][12].

There are numerous prototype file systems which are designed to handle multimedia data. Tiger Shark is a scalable, parallel file system designed to support interactive multimedia applications, particularly large-scale ones [7][6]. MMFS [4] improves interactive playback performance by supporting intelligent prefetching, state-based caching, prioritized real-time disk scheduling, and synchronized multi-stream retrieval. Minorca multimedia file system [14] proposed a new disk layout and data allocation techniques to offer a high degree of contiguous allocation for large continuous media files and a new read ahead method to optimize the input of the I/O request queue. Presto File System [10] introduces the idea of storing the data based on the semantic units. The size of file is limited to one extent. Symphony [13] supports diverse application classes that access data with heterogeneous characteristics.

The main contribution of this paper is the deployment of NS card and the development of EXT3NS file system in designing large-scale video streaming service. The streaming server utilizes *EXT3NS on NS card* to gain high disk I/O bandwidth, to avoid unnecessary memory copies on the data path from disk to network, and to alleviate CPU's burden by offloading parts of network protocol processing. As a result, one can greatly increase the number of concurrent users in streaming service at the same cost (the same number of disks, the same memory buffer, etc.)

The performance measurements on our prototype video server show obvious performance improvements. Specifically, we obtained better results from the basic file system benchmark program, and obtained performance improvements in disk read and network transmission, which leads to overall streaming performance increase. Especially, the streaming server shows much less CPU utilization, and less fluctuation of client bit rate, hence more reliable streaming service is possible.

The remainder of this paper is organized as follows. In section 2, we describe the overall server architecture assumed in this project. The detailed design and implementation of EXT3NS based on the NS card is explored in section 3. In section 4 and 5, we present detailed performance evaluations of the file system and streaming service. In section 6, we summarize our conclusions.

2. Streaming Acceleration H/W: Network-Storage Card

The file system and video streaming service developed in our project are based on an innovative streaming server architecture called SMART (Services of Multimedia Applications for Residential communiTies). The SMART has a CDN-like architecture, where an edge node of the CDN is in charge of actual streaming service.

A streaming server (in edge node) has one or two processors, a system memory, a memory controller hub (or bridge), an I/O controller hub, several PCI/PCI-X controller hubs, a local hard disk, and a remote management adapter card. For efficient storage, retrieval, and transmission of large amount of video data, the server node employs a new H/W device called Network/Storage card (NS card for short). The NS card includes both a disk storage interface and a Gigabit Ethernet interface all together. It also includes a buffer memory to temporarily store the retrieved video data to be transmitted.

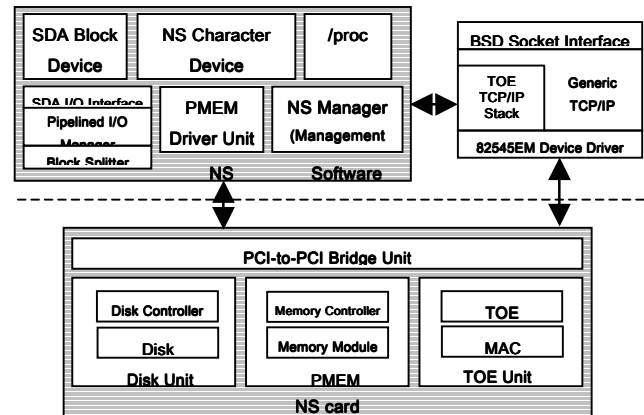


Figure 1 Hardware and software components of NS card

NS card is devised to rectify some limitations of existing file I/O for networked multimedia, such as unnecessary data copying, lack of guaranteed storage access and overhead of network protocol processing. An NS card can support up to 250 concurrent clients, providing up to 1 Gbps stream for each client. This is equivalent to MPEG-2 streaming service with the bit rate of 20 Mbps for 50 concurrent users. In order to satisfy this performance requirement, the NS card introduces the following performance enhancements to existing streaming architectures: (1) a new disk striping driver to increase disk I/O bandwidth, (2) on-board memory to avoid unnecessary data copy to/from main memory, and (3) network interface with protocol offload capabilities.

Figure 1 illustrates the architecture of NS hardware and software modules. The prototype hardware consists of a PCI-to-PCI Bridge, a disk controller with disks, a memory controller with a memory module, and a TOE with MAC devices. Tundra Tsi310 is used as PCI-to-PCI bridge and 64bits/66Mhz PCI bus as secondary PCI Bus. Maximum five Ultra320 15000rpm SCSI disk is attached to QLogic Ultra160 SCSI controller via dual channel Ultra160 SCSI bus. 512 megabytes of SDRAM is controlled by Intel 80321 I/O processor. Data bus is 64 bits wide. Intel 82545M is used as TOE engine for NS card.

The software modules of NS card comprises an NS manager, a stream disk array driver, a PMEM driver, and a TOE driver.

- *Stream disk array (SDA)* The stream disk array is a special-purpose disk array optimized for large sequential disk accesses. Similar to existing RAID intuitively, the SDA provides much higher performance compared with existing RAID by achieving parallelized I/O, pipelined I/O, and low CPU consumption. The SDA software disk striping driver provides at least 1 Gbps bandwidth at the worst-case block distribution. The storage capacity of a NS card is approximately 500 Gbytes, which corresponds to 50 HDTV-quality MPEG-2 files of 10 Gbytes (about 2 hours of video and audio data). The stream disk array driver consists of a stream disk array I/O interface, a request queue, a pipelined I/O manager, and a block splitter. The stream disk array interface provides a way to use *fast path I/O* which avoids unnecessary memory copies. The pipelined I/O manager is in charge of internal/external transfer of disks so that the external bus utilization remains very high at the worst case of block distribution. The block splitter divides one logical block into several split blocks, which are allocated to each disk

separately (this is similar to RAID-3 without a parity disk). This new allocation method splits blocks into all of the disks consisting of the disk array, whereas RAID interleaves blocks into disks. A logical block is divided by the number of disks consisting of disk arrays. The logical disk block size of SDA is 256 Kbytes ~ 2M bytes.

- *Peripheral Memory (PMEM)* The NS card is equipped with DRAM memory modules called PMEM of 512 Mbytes. The PMEM is dedicated to temporary buffer on disk-to-network data path. Using fast path I/O interface, user can read data from SDA disk to PMEM. This procedure is also done by local DMA, it is not related to DMA between device and host's main memory. The EXT3NS also provides a user with the PMEM management library. This library makes PMEM memory mapped into the user address space according to the application's request. The library also provides mechanisms to allocate and release PMEM memory.
- *Protocol offload network interface* Existing TCP/IP protocol stack cannot transmit video data from PMEM to network interface directly, since it assumes that their payload data are in main memory, not in PMEM. Consequently, the protocol stacks are modified in order to access data in PMEM and to transmit streaming data from PMEM to network directly, while existing TCP/IP stacks are still working on main memory. Additionally, we offload the calculation of TCP/UDP/IP checksum and implement Scatter/Gather I/O.

3. Design and Implementation of EXT3NS

In this section, we describe various design aspects of EXT3NS, such as disk layout, data block addressing, file I/O interface, and PMEM interface.

3.1 Design Objectives

As a multimedia file system, the EXT3NS has been developed considering the following design criteria.

- For programming convenience, the EXT3NS must provide applications with the standard read and write system call interfaces to use SDA fast-path disk I/O.
- In addition to fast-path I/O operation, legacy buffered I/O must be supported as well.
- The EXT3NS must support large block sizes defined by block splitting driver of SDA.
- The limit of file size must be large enough to accommodate MPEG-2 video files of 2 hour length
- For availability reason, the EXT3NS must be a journaling file system. Journaling is quite useful since NS card is usually equipped with tens of Terabytes disks which results in long file system check time without journaling.
- Users can use conventional file system commands and utilities, such as cp, mv, etc., on EXT3NS.
- The EXT3NS must provide basic file system administration tools such as mkfs, fsck, etc.
- The file system must be implemented as a kernel module so as not to modify existing kernel blocks like VFS layer.

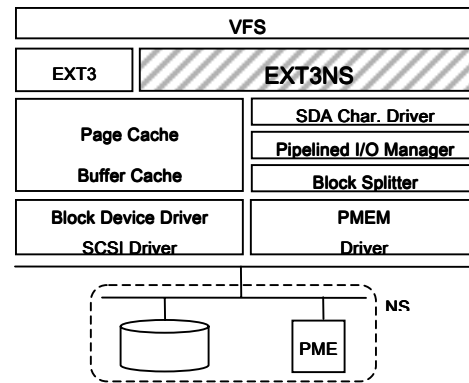


Figure 2 EXT3NS file system in Linux Kernel

3.2 Operations of EXT3NS File System

Figure 2 describes the location of EXT3NS filesystem in the Linux kernel and how it interacts with the kernel. When the application read data from disk via read system call, EXT3NS determines whether it is read operation to PMEM area of NS card or to the system main memory using the argument value of user buffer address. If it is to PMEM area, EXT3NS performs the fast-path I/O by using NS device driver. If it is to main memory, EXT3NS performs the legacy page cache I/O. In case of legacy page cache I/O, EXT3NS just read from disk to main memory by calling generic read function provided by the memory management module of Linux kernel. In this case, requested data block is read from disk to the page cache of Linux kernel through the system PCI bus and copied to the user buffer. Data in page cache can be reused if request for the same data occurs in the near future.

In case of SDA fast-path I/O, EXT3NS calls the read method of SDA device driver. At this time, requested disk block is read to the PMEM block of NS card. Data path of this operation only includes local PCI bus of NS card connecting SDA disk array and PMEM, which does not use system's main PCI bus. From the viewpoint of overall system it could be called as true zero-copy operation, because it does not include memory-to-memory copy and does not use the bus cycle of main PCI bus.

The biggest obstacle for EXT3NS in coexisting with other filesystems under the VFS layer is that disk block size of SDA is much larger than that of generic block devices. x86 architectures allows only 512, 1024, 2048 and 4096 bytes as block size [1], whereas block size permitted by SDA driver is from 256Kbytes to 2Mbytes. So a solution must be made to handle this discrepancy problem properly. We solved it as follows:

- Metadata is stored and accessed on a 4KB basis. There are six kinds of metadata in EXT2 and EXT3: superblocks, group block descriptors, inodes, blocks used for indirect addressing (indirect blocks), data bitmap blocks, and inode bitmap blocks. They are allocated in 4 KB unit and accessed via buffer cache. This can be easily accomplished due to the fact that most of metadata reside in the head of EXT3NS partition and are kept strictly apart from the data and is collected in a separate structure for each file.
- For six metadata, indirect block must be taken special care since it resides in the data block area, not in the metadata

block area. How the EXT3NS handles this kind of metadata will be covered in the next section.

- The unit of allocation for data blocks depends on the configuration of SDA driver. When reading and writing data blocks, applications can use either buffered read/write or fast-path read/write. Buffered read/write is the same as the existing mechanism using main memory cache. On the other hand, fast-path I/O utilizes PMEM as a buffer when reading and writing SDA device.

The EXT3NS inherits the journaling features from EXT3. The EXT3NS provides only ‘ordered’ and ‘writeback’ journaling modes and does not support ‘journal’ mode. The SDA block size is too large to support journal mode due to performance degradation and the lack of disk space for journaling. Most of modern journaling file systems, such as ReiserFS, XFS, and JFS, also limit themselves to log the operations affecting metadata.

3.3 Disk Layout of EXT3NS

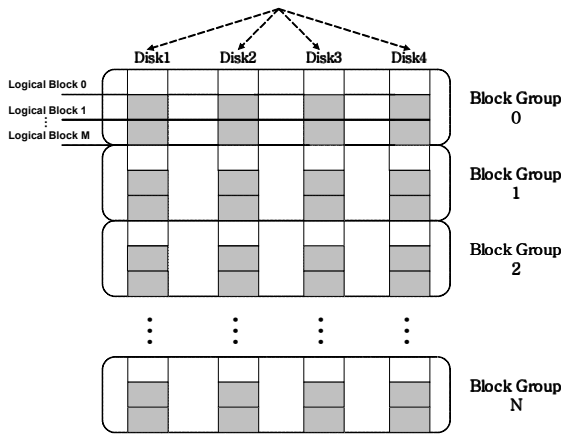


Figure 3 Disk layout of EXT3NS

As shown in Figure 3, disk layout of EXT3NS filesystem is very analogous to that of existing EXT3 filesystem. As in EXT3, it consists of several block groups and each block group consists of metadata blocks (a superblock, block group descriptors, data block/inode bitmaps, inode group) and a chunk of data blocks. Superblock and group descriptors are duplicated to every block group for improving availability, and only the superblock and group descriptors of first block group are used in normal times.

But EXT3NS has some different features from EXT3 in many aspects. The most notable features are difference of block size, block allocation scheme / use of indirection block, addition of some fields in metadata, etc. Some reasons for modifications are:

- Disk block size of EXT3NS is much larger than that of EXT3.
- For a given disk size, the number of block groups is much smaller than that of EXT3.
- EXT3 usually favors small files and is optimized for small and random read/write access, whereas EXT3NS usually deals with large files with sequential read/write pattern.

3.4 Data Block Addressing

The size of metadata differs from that of data block. As we have said above, metadata size of EXT3NS is the same as that of EXT3, which is one of 1KB, 2KB and 4KB in order to use the existing buffer cache mechanism of Linux. On the other hand, data block size of EXT3NS complies with that of SDA, one of 256KB, 512KB, 1MB and 2MB.

EXT3NS manages disk partition as several block groups for improving disk read/write performance. It includes data blocks for a file within the same block group for locating them on physically close position in disk. But if the file size exceeds certain limit, data blocks of that file may be located in different block groups. For EXT3NS, there is little possibility for that case because the size of one block group is much larger than that of EXT3 and the number of block groups is much smaller. For example, there are only 9 block groups when making EXT3NS filesystem in 270GB disk. In this case the size of one block group is approximately 30GB. Furthermore, disk blocks of a file are allocated in sequential manner within the same block group.

File block addressing in EXT3NS uses the same concept as indirection block like EXT2 and EXT3. The size of indirection blocks in EXT3NS is the same as SDA block size, since it resides in data block area. But indirection block access is performed on a 4KB basis when file offset is transformed into disk block number. That is, for a file offset, only one 4KB area of indirection block corresponding to that offset is accessed via buffer cache.

The upper limit of file size is about 2^{53} bytes, although upper layers like VFS do not fully permit this value. It is well known that indirection blocks may incur non-trivial amount of additional overhead, since disk needs to access the inode block and possibly a number of indirect blocks to access the data block. However, EXT3NS uses very large size of indirection block (which is the same as data block size). So it is not greatly affected by the overhead of tree-based file organization, because it can handle files of several gigabytes within only the first-level indirection block. (Actually the maximum size of file which can be handled in first-level indirection is 274,889,441,280 bytes.)

3.5 Fast-path Interface and Page Cache I/O Interface

EXT3NS supports both page cache I/O and SDA fast-path I/O. This can be possible due to the fact that SDA device support both fast-path I/O interface and standard block device driver. The EXT3NS just calculates the logical disk block number correctly for a given file block number. Moreover, it is possible to use fast-path I/O of NS card via standard read/write system call interface. We also support special I/O interfaces for retrieving some information of SDA by means of user level library.

3.6 PMEM interface

EXT3NS filesystem provides user with library-level interface for users to use PMEM memory of NS card. Users can access PMEM area of NS card via user address space using PMEM library. PMEM library maps the physical address of PMEM memory to user address space through mmap() system call. Currently one NS card embeds 512MB PMEM memory and maximum 4 NS cards could be installed at the same time, so maximum 2GB PMEM memory could be mapped to user address space.

PMEM management also provides applications with the mechanism allocating and releasing the PMEM memory. NS software module keeps the free and allocating list of PMEM blocks, and translates virtual address into physical address.

4. Comparison on Basic Streaming Operations

In this section, we experimentally evaluate the proposed streaming server architecture and the file system. The focus of this section is mainly on the performance of reading disk blocks and sending data to network.

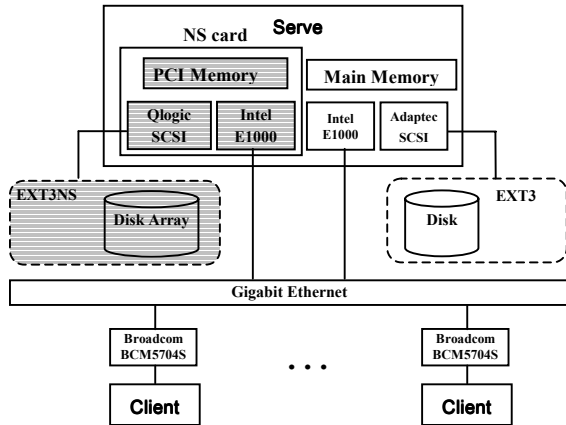


Figure 4 Configuration of testbed system

Figure 4 shows a detailed view of our testbed system. The testbed consists of one server and numbers of clients interconnected by gigabit ethernet network. The server has dual Intel Xeon 2.4 GHz processors and 1GB DDR RAM. The conventional EXT3 file system is created in a 73GB Seagate Cheetah Ultra320 SCSI disk (15,000 RPM) attached to the Adaptec SCSI controller embedded in the main board. The EXT3NS file system is created in four Seagate SCSI disk array (which is the same as the one used in EXT3) attached to the Qlogic SCSI controller embedded in NS card. Ethernet controllers embedded in NS card (used for EXT3NS) and main board (used for EXT3) are all Intel E1000 gigabit network adaptor. The operating system of the server is Redhat Linux 8.0 (kernel version 2.4.18-14) with NS card driver modules. NS card driver, EXT3NS and EXT3 file systems are all configured as a kernel module.

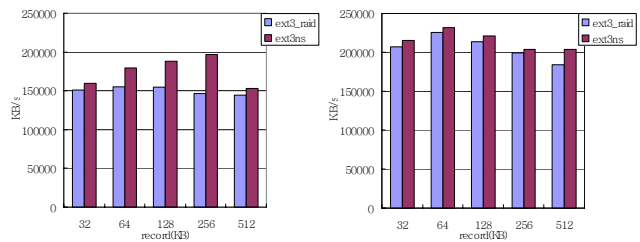
The performance of the EXT3NS file system is measured via experiments with a streaming workload. We use a simple streaming server and client program called *SerCIE*, which is an in-house streaming service benchmark tool. The popularity of video in *SerCIE* is modeled according to a zipf distribution. The skew factor value is selected by [5] to tune the distribution to match empirical video rental patterns for the most popular 100 videos of the time. In order to compare the performance of the EXT3NS file system with the existing EXT3, we build two versions of ‘*SerCIE*’, one of which uses PMEM of the NS card as I/O buffer, and the other uses system main memory. From now on we call each version of *SerCIE* as *SerCIE-NS* and *SerCIE-MM*. *SerCIE-NS* reads data from disk to PMEM, whereas *SerCIE-MM* reads data from disk to main memory.

4.1 Basic Filesystem Performance

In this experiment the testbed has a disk array of four SCSI disks. In order to ensure that EXT3 and EXT3NS file system have the same disk capability, EXT3 file system is configured on the software raid level 0 of four disks. We use the Iozone [8] benchmark tool for measuring file system performance¹. We mainly presents the results of *read* test and *write* test. Read test measures the performance of reading an existing file sequentially. Write test measures the performance of writing a new sequential file. When a new file is written, not only does the data need to be stored but also the overhead information for keeping track of where the data is located on the storage media.

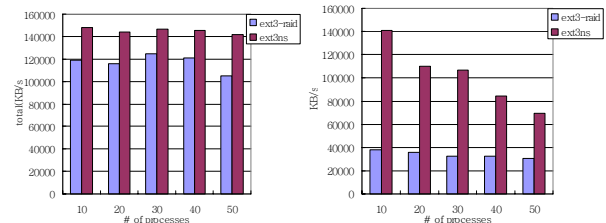
4.1.1 Page cache I/O

In this section, we measures the performance of page cache I/O of EXT3NS and EXT3.



(a) write (b) read
Figure 5 EXT3NS using main memory buffer

Figure 5 compares the results of read and write test for single stream case. The size of test file is 1 Gbytes and the record size is 128KB (record is a chunk of data to be read or written in a single read/write system call). The performance gap between the two file systems mainly comes from the difference of data block size of those file systems. The data block size of EXT3NS is equal to the block size of PMEM of NS card. In this experiment, data block size of EXT3NS is 1MB. EXT3 needs 256 times data block allocation for 1MB free disk space when using 4KB block size. But EXT3NS just needs to allocate data block once for 1MB free block space.



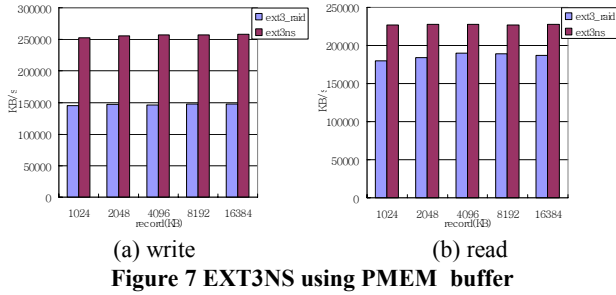
(a) write (b) read
Figure 6 EXT3NS performance when multiple processes access with main memory

Next we measure the performance of file systems when multiple processes access the file systems simultaneously. The file size is 100MB and record size is 128KB. The difference in Figure 6 is not because of the low cpu-utilization of EXT3NS but efficient data block allocation of EXT3NS

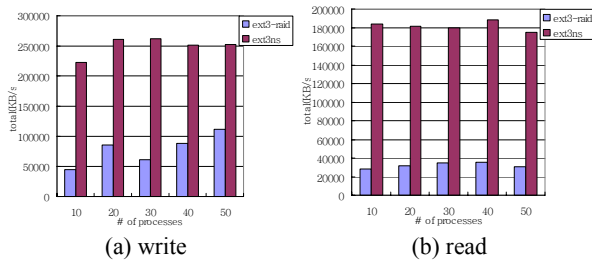
¹ We were obliged to modify Iozone’s source code in order to run it on NS card. It obviously violates Iozone’s run rules. We tried to keep the modifications as small as possible.

4.1.2 Fast path I/O

In this section, we measure the performance of fast path I/O of EXT3NS against the performance of EXT3.



In Figure 7, EXT3NS uses SDA fast-path I/O interface to read and write data. The size of test file is 1 Gbytes and the record size is 1 MB. The variation of record size affects much less than Figure 5. The read performance of EXT3NS is slightly better than that of EXT3. When only single process accesses the file system, the performance gap is rather small, but when several processes access the file systems at the same time, EXT3NS outperforms EXT3 file system because SDA I/O of EXT3NS is much more efficient to handle large data chunk at once.



In Figure 8, multiple processes use PMEM as a buffer to read from and write to disk. The file size is 100MB and the record size is 1MB. EXT3NS shows tremendous performance enhancement compared with EXT3 file system. There are two reasons for this result. First, the SDA fast-path I/O can easily handle large chunk of data such as one megabyte buffer. EXT3NS just need to call command only once to read and write 1MB chunk. But EXT3 has to call command for 256 times. Second, the SDA fast-path I/O does not need much CPU cycles compared with legacy EXT3.

4.2 Network Performance

Recall from Section 2 that the NS card sends data in PMEM directly to the network. It bypasses copying data to the TCP buffer in kernel memory, which requires additional time and processor power. To demonstrate this behavior, we run the two versions of SerCIE on our testbed systems and measure the time spent in sending data to the network. SerCIE-NS checks the time spent in sending data from PMEM of NS card to the network adapter, and SerCIE-MM checks the time spent in sending data from main memory to the network. The test is made for various number of client requests, from 50 to 200, and the size of each data file is 700 megabytes. We measure the time spent in send system call by using `gettimeofday`, and calculate the average time to send one megabytes of data to the network for all client requests.

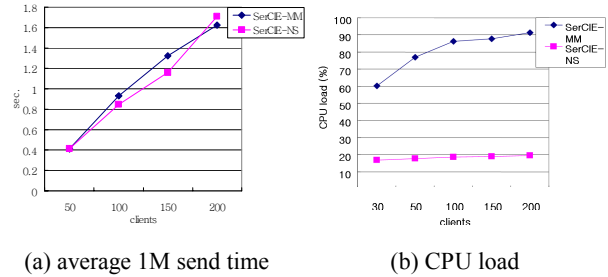


Figure 9 shows the average send time for one megabyte data in two versions of SerCIE for various number of client requests. The send performance of SerCIE-NS is almost the same as SerCIE-MM. The reason is that the pure performance of network drivers used in NS card and on-board NIC is almost the same. But by using NS card, lots of process power could be saved with the help of zero-copy scheme and protocol offloading in NS card, so the NS card shows much better performance in overloaded circumstances (Figure 9 (b)).

4.3 Streaming Acceleration Performance

The streaming performance mainly depends on the read performance from disk and send performance to network. Thus we measure the overall streaming acceleration performance, which consist of reading time from disk to memory buffer and sending time from buffer to the network. As in Section 4.2, we use two versions of SerCIE for benchmarking EXT3NS and EXT3 again. we use `gettimeofday` system call to measure the time spent in each read/send system calls and average number of bytes read and sent in one second for all client requests.

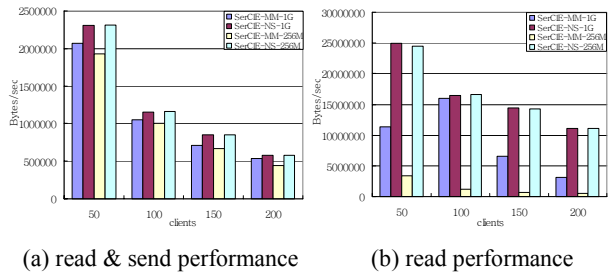


Figure 10(a) shows the average bytes of data for reading from disk and sending to the network in one second for each version of SerCIE. Our SerCIE-NS shows better performance than SerCIE-MM from 8 to 17 percent. Actually network performance of NS card is not so much better than conventional Gigabit NIC as seen in Section 4.2, but disk read performance of EXT3NS is better than EXT3, which result in more streaming performance gap between the two versions of SerCIE.

The amount of buffer cache mainly depends on the amount of main memory in the server. Large buffer cache leads to the increase of cache hit rate, which results in improvement of read performance. Actually it is not the real performance of disk, so the test should be made for small amount of main memory in order to minimize the effect of buffer cache and compare the pure disk performance. Thus we decrease the memory size from one

gigabyte to 256 megabytes and run two versions of SerCIE program. Figure 10(a) also shows the average bytes of data for reading from disk and sending to the network in one second for various number of client requests in 256 megabytes of main memory. The performance of SerCIE-NS is not so bad compared with the result of one gigabyte memory test, but the performance of SerCIE-MM is far worse than that of one gigabyte memory test. I/O operation using NS card does not use buffer cache in kernel memory, so decrease of main memory cannot affect the performance of SerCIE-NS. But conventional disk read uses buffer cache, which results in worse performance of SerCIE-MM.

To compare disk read performance more precisely, we measure pure disk read performance in SerCIE-NS and SerCIE-MM. Figure 10(b) shows the average bytes of data for reading from disk in one second in two versions of SerCIE. As the number of clients increases, the performance of SerCIE-MM becomes rapidly worse, while the performance of SerCIE-NS is almost constant regardless of client numbers. The performance gap is much larger in small memory size. Notice that the performance of SerCIE-MM becomes a little better when number of clients increases from 50 to 100. The reason is that more requests converge to one or two contents because of zipf distribution characteristic, which results in the increase of buffer cache hit rate. Figure 10(b) also shows the read performance of two versions of SerCie in 256 megabytes of memory. Performance of SerCIE-MM degrades linearly, while the performance of SerCIE-NS is still constant.

5. Comparison on Throughput

In this section, we evaluate the throughput of EXT3NS file system to show how well it can handle streaming requests from the viewpoint of clients. We use the two versions of SerCIE as before, and check the bitrate of client side to determine if it is enough for seamless streaming service.

5.1 Number of Concurrent Users

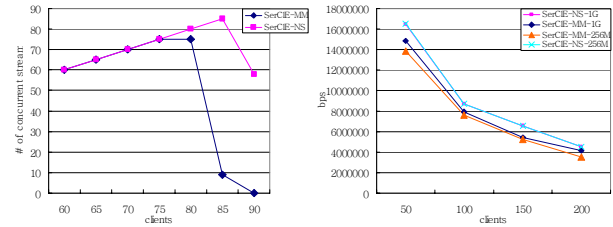
In order to serve seamless streaming, the server must keep constant bitrate during the services. So a test is carried out to determine how well the server can transmit data to clients keeping the minimum required bitrate for seamless streaming. Recall that our project aims at high quality streaming services, so the bitrate constraints have to be very strict. Minimum 10Mbps must be satisfied for seamless streaming of DVD-quality MPEG-2 streaming, and 20Mbps for HDTV-quality streaming.

We run the two versions of SerCIE again, and check the average bitrate of each client at every 5 seconds. Then we count the number of successful time intervals (keeping the minimum bitrate constraints) and failed time intervals. If the ratio of failed count to total count is smaller than certain percentage, we consider it as 'valid', else we consider it as 'invalid'. And we examine the maximum number of 'valid' streams in each file systems.

Table 1 shows the number of valid streams for each bitrate constraint; 10Mbps and 20Mbps. EXT3NS can service 6~9 more clients than EXT3 in 10Mbps stream and 12~23 more clients in 20Mbps.

Table 1 Maximum number of valid streams

benchmark	SerCIE-MM		SerCIE-NS	
	bps	%	bps	%
10Mbps	5	44	50	
	10	53	61	
	20	60	69	
20Mbps	5	7	30	
	10	14	35	
	20	23	35	



(a) Number of valid streams (b) Client bitrate

Figure 11 Average client bitrate

Figure 11(a) shows the number of valid streams in EXT3 and EXT3NS. Both can serve all of its clients very well until 75. However the difference between the number of valid streams for two file systems rapidly increases as the number of clients exceeds 80. Figure 11(b) shows average bitrate of each client for various client numbers. The EXT3NS outperforms EXT3 by 9~21 percents. To reduce the buffer cache effect, we resize the main memory of server to 256 megabytes and did the same experiments again. Figure 11(b) also shows average bitrate of each client in this case. Decreased buffer cache size leads to decrease of buffer cache hit rate, which increases read system call execution time of EXT3 file system. So the aggregated bitrate of the server decreases and the bitrate of each client would be also decreased.

5.2 Variance of Bitrates

We must consider bitrate variances in streaming service, because it definitely affects streaming quality of client side. Clients usually stop streaming service when the bitrate is dropped below the minimum required bitrate of media file. Such conditions often happen when variance of bitrate is high because of large fluctuation in bitrate variation. As a result, client needs more buffer memory in order to minimize the bitrate fluctuation effect. So we measure the bitrate of each client at every 5 seconds as in Section 5.1 and calculate the difference of highest and lowest bitrate for every client. Then we took the average of the differences of all clients.

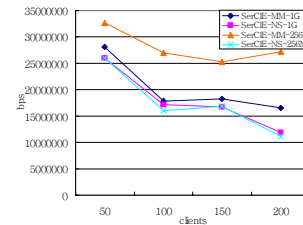


Figure 12 Average bitrate variance

Figure 12 shows average bitrate variances for each number of client requests. For all cases, SerCIE-NS shows lower bitrate variation than SerCIE-MM. As shown in Figure 12, bitrate variance of EXT3NS is smaller than that of EXT3 from 4 to 28 percents according to circumstances. We resize the main memory size of server to 256 megabytes and do the same experimentation again. Figure 12 also shows the average bitrate variance when server run in 256 megabytes of main memory. Result of EXT3NS is not so much different from that of 1 gigabytes main memory environment, but variance of EXT3 is much greater than result of 1 gigabytes memory. It is because buffer cache hit rate of server decrements as the memory size reduced, which results in large variation of read system call execution time between cache hit and cache miss.

To compare the bitrate patterns of the two versions more precisely, we measure the bitrate trace of one client for every 5 seconds. Bitrate of EXT3NS is relatively stable as compared with that of EXT3 and keep at least 300,000 bps for most of the time. But bitrate of EXT3 even drops to zero occasionally. Client needs more buffer memory to minimize the bad effect of large bitrate fluctuation. Therefore, when server uses EXT3NS file system, clients can get streaming service with smaller system resources than conventional EXT3 file system.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we focus our effort on devising high-performance file system for streaming operation and analyze its behavior under streaming workload. Our file system has the following features: (1) For programming convenience, the EXT3NS provides applications with the standard read and write system call interfaces to use SDA fast-path disk I/O. (2) In addition to fast-path I/O operation, legacy buffered I/O is supported as well. (3) The EXT3NS supports large block sizes which are defined by fast-path disk I/O driver. (4) The limit of file size is large enough to accommodate high quality video. (5) For availability reason, the EXT3NS is a journaling file system. (6) Users can use conventional filesystem-related commands and utilities on EXT3NS. (7) The EXT3NS provides basic file system administration tools such as mkfs, fsck, etc.

We examine the performance of the given file system under streaming workload and compares it with the performance of the EXT3 file system. There are a number of distinctive features which deserves attention: Disk read performance improves by 3~118%, transmission performance improves by 0.6~14%. As a result, the overall streaming performance increases by 8~30%, and especially, our file system shows much less CPU utilization by 43~72% than EXT3 file system.

The EXT3NS has a lot of enhancement possibilities. Firstly, asynchronous I/O is very desirable for large-scale video servers. Currently SDA driver have some primitives for asynchronous I/O. It is possible to submit disk I/O asynchronously and get the result via signal. We are planning to support asynchronous fast-path I/O on the EXT3NS. Secondly, we are designing PMEM caching. As of current version of EXT3NS, PMEM is merely used as a buffer area in streaming operations. According to our analysis, PMEM as a cache will greatly improve SDA I/O latency.

7. REFERENCES

- [1] Bovet, D. P. and Cesati, M. *Understanding the Linux Kernel*, 2nd Ed.
- [2] Buddhikot, M. M., Chen, X. J., Wu, D., Parulkar, G. M. Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, Austin, Texas, June 1998.
- [3] Chen, M. S., Kandlur, D. D., and Yu, P. S. Optimization of the Grouped Sweeping Scheduling (gss) with Heterogeneous Multimedia Streams. In *Proceedings of ACM Multimedia '93*, 1993, 235-242.
- [4] Chiueh, T., Niranjana, T. H., and Schloss, G. A. Implementation and Evaluation of a Multimedia File System. In *Proceedings of International Conference on Multimedia Computing and Systems*, 1997.
- [5] Dan, A., Sitaram, D., and Shahabuddin, P. Dynamic Batching Policies for an On-Demand Video Server. *Multimedia Systems*, 4, 3 (1996), 112-121.
- [6] Fitzgerald, R. P., Bolosky, W. J., and Douceur, J. R. Distributed Schedule Management in the Tiger Video Fileserver. *ACM SIGOPS Operating Systems Review*, 31, 1997.
- [7] Haskin, R. L. Tiger Shark – A Scalable File System for Multimedia. *IBM Journal of Research and Development*, 42, (1998), 185-197.
- [8] Iozone.org. iozone. <http://www.iozone.org>.
- [9] Kenchammana-Hosekote, D. R. and Srivastava, J. Scheduling Continuous Media on a Video-On-Demand Server. In *Proceedings of International Conference on Multimedia Computing and Systems*, Boston, May 1994.
- [10] Lee, W., Su, D., Wijesekera, D., Srivastava, J., Kenchammana-Hosekote, D., and Foresti, M. Experimental Evaluation of PFS Continuous Media File System. In *Proceedings of Conference on Information and Knowledge Management*. Las Vegas, Nevada, 1997, 246-253.
- [11] Maltz, D. and Bhagwat, P. TCP Splicing for Application Layer Proxy Performance, *IBM RC 21139*, Mar. 1998.
- [12] Rangan, P., Vin, H., and Ramanathan, S. Designing an On-Demand Multimedia Service. *IEEE Communication Magazine*, 30, 7 (July 1992), 56-65.
- [13] Shenoy, P. J., Goyal, P., Rao, S. S., and Vin, H. Symphony: An Integrated Multimedia File System. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking*. San Jose, Jan 1998, 124-138.
- [14] Wang, C., Geobel, V., and Plagemann, T. Techniques to Increase Disk Access Locality in the Minorca Multimedia File System. In *Proceedings of the 7th ACM Multimedia*, 1999.