Multimed Tools Appl
DOI 10.1007/s11042-007-0156-1

HERMES: embedded file system design for A/V application

Youjip Won · Doohan Kim · Jinyoun Park · Sichang Lee

Abstract Embedded systems such as PVR, set-top box, HDTV put unique demand on I/O subsystem design. Underlying software, particularly file system, needs to be elaborately designed so that it can meet tight constraints of consumer electronics platform: performance, price, reliability, and etc. In this work, we develop state-of-art file system elaborately tailored for A/V workload. There are two design objectives in our file system: performance and support for logical level abstraction. For performance, we develop a number of novel features: extent based allocation, single level file structure with block index augmentation scheme, aggressive free block allocation to minimize disk fragmentation, elaborate file system meta data layout, separation of name space data and file data and etc. HERMES enables the user to view file as a collection of semantic units (frame or audio samples). HERMES file system encompasses most of state-of-the-art file system technologies published in preceding works. Via extensive physical experiment, we verify that HERMES file system successfully addresses the original issues: good scalability, predictable I/O latency (minimizing variability in I/O latency), efficient disk head movement and etc. This is the result of harmonious effort of large I/O size, aggressive free

Primitive version of this work has been published on *Lecture Note in Computer Science* series, Springer, vol. 2510, pp 172–179, Oct 2002 (Proceedings of EURASIA-ICT).

Y. Won (✉)
ECE Division, Hanyang University, Seoul, Korea
e-mail: yjwon@ece.hanyang.ac.kr

D. Kim
Samsung Electronics, Suwon, Korea

J. Park
LG Electronics, Seoul, Korea

S. Lee
Tellion, Seoul, Korea

block allocation algorithm, data block placement strategy, file organization, layout of HERMES file system and etc. The result of performance experiments indicate that HERMES file system prototype successfully meets the file system constraints for high volume and high bandwidth multimedia application. HERMES file system exhibits superior performance to EXT2 file system (Linux) and XFS file system (SGI).

Keywords Multimedia · A/V workload · File system · Disk scheduling · Embedded system

1 Introduction

1.1 Motivation

Information Appliance for digital video can be thought as a light weight computer system designed to store incoming high quality digital video stream at the local hard disk and/or to play the recorded video clips at user's convenience. Unlike the general-purpose computer, which has abundant computing resources and storage capacity, this type of consumer electronics has stringent resource constraints due to its restriction on power consumption, pricing, acoustic, reliability, and etc. The disk drive in this device is not an exception. It is not feasible to use high performance disk, e.g. SCSI disk, even though the real-time playback and recording of multimedia data puts intense bandwidth demand on the storage device. It is mandatory that the underlying file system is elaborately designed to fully utilize the physical performance of the disk by exploiting the workload characteristics. The fundamental design philosophy of the most commodity file systems, e.g. Unix File System, NTFS, EXT2, FAT32, or etc. are ill-suited for meeting the real-time performance requirement of audio and video retrieval. Small I/O unit size, external fragmentation, complex file organization (e.g. conditional skewed tree structure) and etc. are just a few examples which prohibit the effective utilization of commodity file system in A/V devices.

Efficiency of the underlying file system plays a critical role in supporting real-time A/V application in cost effective manner. To effectively exploit the physical bandwidth of the disk, it is important that file system layout, metadata structure, file organization, file placement, etc. are elaborately tailored so that disk fragmentation is avoided and the time to locate the data block is minimized. Large variation in I/O latency can negatively affect the efficient scheduling and resource allocation of the multimedia data block retrieval. In real-time playback and recording of audio and video data, underlying file system should be able to deliver the requested data in predictable manner.

There is not much debate that the Unix file system is a landmark achievement in modern file system design. However, from A/V application's point of view, there are a number of issues in Unix file system which require further elaboration. Our work, HERMES file system, is aimed at addressing these issues: performance, file system semantics, and reliability. The first issue arises mainly due to organization of file and file system. The Unix (or Unix like) file system is designed for a general purpose usage which needs to incorporate wide variety of data types and wide range of file sizes where most of the files are small. It adopts a skewed tree like file structure. Unfortunately, this design philosophy is ill-suited for meeting the real-time requirement of audio and video data retrieval. Navigating through the multi-level tree structured file can entail a non-trivial amount of disk head movement in visiting the internal nodes of the tree. HERMES file system uses aggressive organization and layout of file system. It also optimizes the free block allocation and data block placement algorithm to avoid disk fragmentation. Meta data structure is designed to achieve this objective. Also, to exploit the characteristics of sequential I/O, HERMES allocates separate region for directory blocks and file data blocks. The second issue is data abstraction. In legacy file system, file is a collection of fixed size physical units, e.g. block or byte. This general abstraction makes the file system simple and efficient while most of the details are left to the application. Semantically, multimedia file is collection of logical units, e.g. video frames or audio samples. Playback, storage and edit operation on multimedia contents deal with video frames and audio samples, rather than blocks. HERMES file system provides block as well as semantic (or logical) level abstraction. It enables the user to view file as a collection of fixed size blocks and in the mean time as a collection of logical units, e.g. video frame. The third issue is reliability. One of the main differences between computer and consumer electronics device is reliability requirement. Consumer electronics device should be much more robust and should be quick in crash recovery. In practice, average consumer electronics user cannot tolerate lengthy file system check and rebuild operation, e.g. `fsck` operation. We elaborately tailor the journaling scheme to satisfy the robustness and crash recovery requirement for consumer electronics platform. Some features of HERMES file system, e.g. extent based allocation, and logical index, have been proposed in preceding works. However, the effectiveness of these features have not been properly explored in the context of real-time mutlimedia workload. One of the important contribution of our work is the comprehensive performance experiment and in depth analysis. Via disk trace level analysis, we closely examine the physical disk head movement in HERMES, EXT2 and XFS. This gives us clear understanding of what we need to consider in designing file system for real-time multimedia application. We find that overall design of HERMES (file system layout, meta data structure, pre-allocation strategy, and etc.) successfully achieves its original philosophy.

1.2 Related works

The file system related issues in multimedia workload have been receiving prime focus since the inception of the multimedia streaming technology. A lot of efforts have been made on devising an algorithm for continuity guarantee. Since legacy SCAN, FIFO, and their bifurcations do not provide bandwidth guarantee, it was not possible to provide continuous flow of the data blocks to the end system. A number of works address these issues and propose the disk scheduling algorithms for the multimedia data retrieval [4, 9, 16, 18, 28]. In practice, there is a need to support real-time multimedia I/O as well as legacy text based I/O in a single disk based framework. In this case, the disk subsystem is required to service I/O requests with different priority requirement and in the mean time, it has to maximize disk performance. A number of disk scheduling algorithm and the prototype system have been proposed to achieve this objective [19, 21, 25, 27]. Recently, Mokbel et al. [15] have proposed new disk scheduling scheme which takes into account a number of scheduling attributes, e.g. deadline, criticality, cylindrical position and etc., and generates a disk schedule based upon the greedy approach. They used the notion of space-fill-curve to devise a schedule for a given set of disk requests.

There are a number of prototype file systems which are designed specifically to handle the multimedia data [2, 10]. XFS [22] is designed for general purpose file system. The most notable mechanism in XFS to increase the scalability of the file system is the B+ tree. XFS uses B+ trees for tracking the free extents, indexing the directory entries, managing the file extent maps, and tracking dynamically allocated i-nodes scattered throughout the file system. XFS is partitioned into regions called allocation groups, and replaces the block oriented bitmaps with an extent oriented structure consisting of a pair of B+ trees for each allocation group to allow for efficient searching for large regions of contiguous space. One of B+ trees is indexed by the starting block of the free extents, and the other is indexed by the length of the free extents. MMFS [5] improves interactive playback performance by supporting intelligent pre-fetching, state-based caching, prioritized real-time disk scheduling, and synchronized multi-stream retrieval. It defines new file system data structure called `mminfo` which carries the streaming specific information, e.g. direction of playback, playback rate, and speed of playback (x2, x4, etc.). MMFS uses existing file organization and file system structure of UFS. Minorca Multimedia file system [24] proposed (a) a new disk layout and data allocation techniques called MOSA that offers a high degree of contiguous allocation for large continuous media files and allows the coexistence of small, non-CM files, and (b) a new read ahead method to optimize the input of the I/O request queue. These techniques aim at increasing disk access locality and at reducing disk seek overhead.

Presto file system [13] introduces the idea of storing the data based on the logical unit. The unit of placement is extent that consists of fixed number of semantic units. This file system can suffer from wastage of space when the actual file size is smaller than the extent size. SMART file system [17] maintains a file as a linked list of extents and thus improves the file size limitation in Presto [13]. Symphony [20] also allows each video file to be accessed either as a sequence of byte or as a sequence of frames. To support two different abstractions in accessing the file, they use two level index structure: index for frame which maps the frame index to byte offset and index for byte which maps the byte offsets to disk block addresses. In Minorca file system and Symphony file system, file is organized using index block and has tree like structure. Particularly, Minorca file system takes B-tree approach. It clusters the index block and the data block together.

In massive scale multimedia server, it is important to reduce the I/O path involved in streaming out the multimedia data. A number of file system have been recently proposed to guarantee the QoS of the system [6, 12]. Zimmermann [30] proposed statistical admission control scheme for multimedia file system. Ahn et al. [1] developed a network interface card which eliminates the unnecessary memory copy on the data path from the disk to network interface card and offloads the network protocol processing.

The rest of the paper is organized follows. Section 2 presents synopsis in Unix file system. Section 3 describes the basic structure of HERMES file system. Section 4 describes the support for logical level abstraction. Sections 5 and 6 presents the for block allocation scheme and journaling approach. Section 7 presents the result of performance experiment. Section 8 concludes the paper.

2 Synopsis: Unix file system

2.1 Structure of i-node in Unix file system

In Unix file system, the file management information is kept strictly apart from the file data itself and collected in a separate structure. This structure is called i-node. It contains the file meta data and data block references. For example, i-node of EXT2 file system contains the information of file mode (e.g. rwxrw-r–), user id of owner, file size, and the data references.

Data reference fields in i-node contains the physical location of data blocks. In case of EXT2 file system in Linux, the data references consist of twelve direct references, one indirect reference, two-step indirect reference and three-step indirect reference. With 4 KB data block, up to 48 KB file can be covered with direct references. Single 4 KB block can hold 1,024 block pointers. With single indirect reference, up to 4 MB of data can be covered. With two-step indirect reference, up to 4 GB of data can be stored in a file. Given that multimedia file can easily go beyond tens of mega byte, the data block retrieval for playback entails the retrieval of the intermediate pointer blocks as well. While tree structure based file organization gives greater flexibility in handling wide variety of file sizes, the retrieval of the pointer blocks may entail substantial overhead in real-time playback and recording of multimedia data.

Even with compression, single one hour movie requires from hundreds of mega byte up to several giga byte of disk space. For example, MPEG-1 compressed video for 90 min (1.5 Mb/s) requires approximately 1 GB of storage space. With MPEG-2 compression scheme whose playback bandwidth ranges from 4 to 10 Mb/s, storage requirement becomes heavier substantially. This storage volume and bandwidth requirement becomes further intense in consumer electronics device with the rapid proliferation on HDTV quality digital broadcasting. For example, ATSC format video contents requires 19.2 Mb/s playback bandwidth.

Storing the file from hundreds of mega byte to several giga byte requires two-step or three-step indirect blocks. Thus overhead of accessing i-node block and possibly a number of pointer blocks can be substantial. To access the last data block in 1 GB file, maximum of three additional blocks are accessed to retrieve single data block. These three blocks consist of one i-node block and two indirect blocks (Fig. 1). Even though the i-node and the pointer blocks are in the buffer cache, memory access time can consume significant fraction of CPU cycle. It is very unlikely that the pointer blocks and the data blocks are stored consecutively, especially, when a number of files co-exist in the file system. Thus, meta data update operations, access to indirect pointer blocks and access to actual data block altogether can make the disk head movement very inefficient. We closely examine physical disk head movement in a number of file systems in Section 7.

2.2 Partition layout in Unix file system

Most file systems of modern Unix family operating systems, e.g. Linux, Solaris, NetBSD, etc. adopt the mechanism to place the data blocks consecutively or as

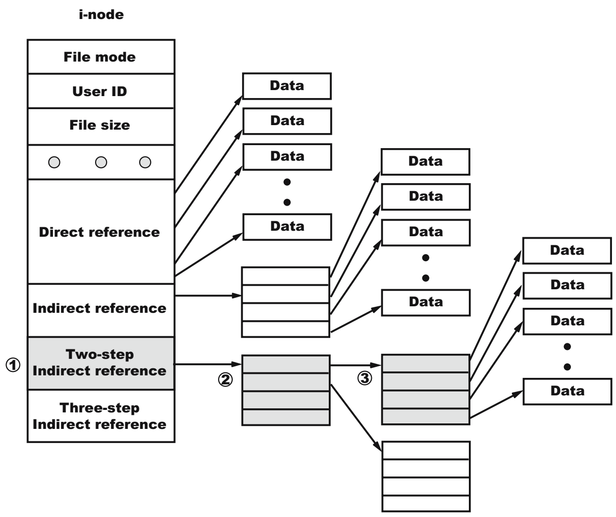


Fig. 1 Block reference structure of EXT2 i-node

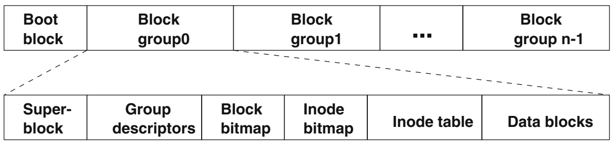
closely as possible. For example, EXT2 file system (Fig. 2), which is the most widely used file system in Linux operating system, uses the concept of block group. The block group is a group of consecutive cylinders. With block group, file system can cluster the file data blocks within relatively closer cylindrical position. However, block group based placement policy still splits the file into different block groups when the file size exceeds the size of the block group and may suffer from significant overhead in disk seek. EXT2 file system partition, consists of multiple block groups. Each group contains the copy of file system superblock (for file system consistency's sake), group descriptor, block bitmap, i-node bitmap, i-node table, and finally data blocks, with the respective order. The size of the block group is mainly constrained by the block bitmap. It is used to identify the free and used blocks within a group. Block bitmap should fit in a block Thus, the maximum number of blocks in a block group corresponds to $8 \times b$ blocks, where b denotes the size of a block in byte. For 4 KB block file system, single block group is as large as 128 MB.

3 HERMES file system structure

3.1 File system organization

Most of the Unix family file systems do not differentiate the file data blocks and directory entry blocks. Multimedia file, e.g. video file, is not an exception. It is possible that multimedia files and the directory files are placed in the disk in interleaved fashion. This can negatively affect the performance of sequential scanning operation on multimedia file. To resolve this issue, HERMES file system maintains the directory block and data block separately. HERMES reserves a certain fraction of contiguous disk space to store directory information. Super block contains the information on the location of directory region and multimedia data region. The size of the directory extent region is fixed when the file system is formatted. Figure 3a illustrates the layout of HERMES file system partition. HERMES partition consists

Fig. 2 Disk layout of EXT2 file system



of the following regions: super block, extent bitmap, i-node bitmap, i-node tables, directory extents and data extents. This file organization was proposed in our earlier work [26].

Superblock is located at very beginning of the file system partition and stores general information of the file system. It contains the information about total number of extents, the number of multimedia extents, the number of free extents, extent size, the number of i-nodes, creation time and etc. Extent bitmap is used to determine whether the respective extent is in use or not. The i-node table consists of predefined number of i-nodes. HERMES adopts extent based structure. In HERMES, extent is the smallest allocation unit. Extent size is determined in file system format phase. Directory entries are stored in directory extent region. After directory extent region, data extent region begins. By separating directory region from the file system data region, we try to reduce the disk seek overhead. Examination on physical disk head movement and the number of I/O requests shows that this design approach successfully improves the head movement overhead and brings substantial performance increase.

3.2 File organization

Information in HERMES file meta data block can be categorized into four sets: legacy file meta data (ownership, most recent access date, etc.), QoS information (frame rate and playback rate), blocks index fields and frame index fields. Figure 3b illustrates the structure of HERMES i-node.

There are total fifteen block index and three frame index fields. Data block access requires the retrieval of pointer block. When the pointer block and the data block are stored apart, it may cause disk seek in retrieving the pointer information and the data block. In fact, most of Unix file system family adopts this approach. In HERMES, we take B-tree like approach and clusters the block index information with data block.

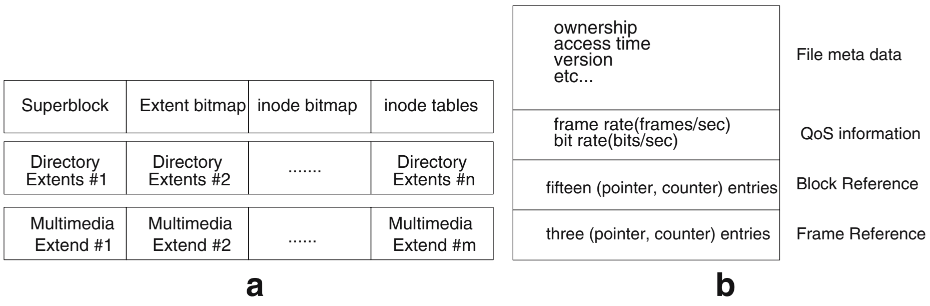


Fig. 3 Structure of HERMES file system. **a** File system layout, **b** i-node structure

Figure 4 illustrates the reference structure of HERMES. UFS adopts skewed tree like file structure for the faster access to small size file and to cover very large files at the same time. However, this skewed multi-level tree structure increases not only the I/O latency but also the variance of I/O latency which negatively affects the I/O predictability. HERMES file organization is designed to address both of these issues: reducing the level of indirection and variance in I/O latency. Reducing the level of indirection requires more pointers in the i-node block. However, we cannot increase the i-node size arbitrarily. We resort to have the pointer to point to *group* of consecutive extents and augment each pointer with the number of consecutive extents. HERMES i-node has total of fifteen (pointer, count) entries. The first twelve entries directly points to group of extents. `i_count` is four byte and thus single cluster of extents can consist of as large as $2^{32} - 1$ number of extents. With 128 KB extent, single (pointer, count) pair can cover theoretically 512 TB data region. While pointer augmentation scheme provides effective solution on reducing the number of pointers and the level of indirections at the same time, it may not behave as expected. When the disk partition is severely fragmented, each pointer may cover only one extent. To cope with this situation, the last three (pointer, counter) entries are designed for indirect access. One of the novel features of HERMES file system is its design of indirect references. As in Fig. 1, legacy Unix family file system manages the data block pointers in separate blocks. Thus, accessing a data block via indirect reference requires the retrieval of pointer blocks as well as data blocks. In HERMES,

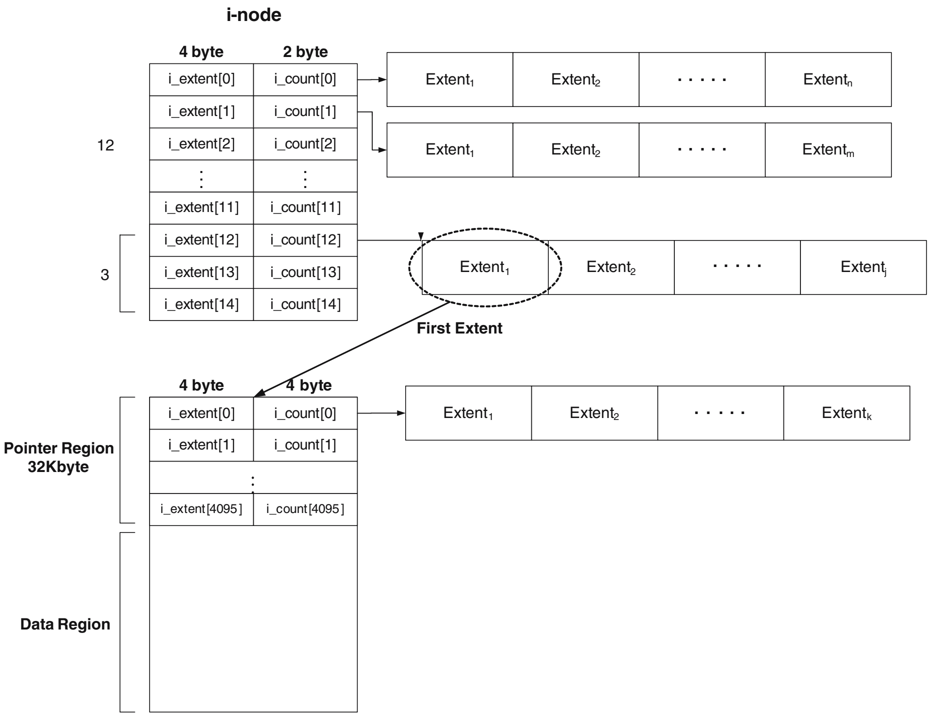


Fig. 4 Block reference structure of HERMES i-node

we cluster the data block and block pointers together. This is to save I/O operations for pointer block retrieval. Figure 4 illustrates the data block reference structure in HERMES file system.

In the extents pointed by these three pointers, the first 32 KB of cluster of extents are designated to contain (pointer, count) entries. There are 4,096 (pointer, count) entries ($\lfloor 32 \times 2^{10}/8 \rfloor$) in this region.

4 File as a collection of frames

Legacy general purpose file systems including Unix family file systems treat file as a collection of blocks (or characters). This approach makes the underlying file system small, simple and flexible. However, as we have more understanding on the file system usage, we can incorporate more elaborate treatment in file system. This approach has been far advanced in DBMS community where file is defined as a collection of records (in case of relational database management systems). HERMES is designed for embedded system for A/V application. We tailor the file system to provide more specific interfaces for real-time multimedia applications.

Legacy file access semantic is (file descriptor, offset) pair and offset is byte distance from the beginning of the file. Multimedia file can be thought as a collection of logical data units, e.g. frames or audio samples. In case of MPEG compression standard, frame has one of three types I-type, P-type or B-type [11]. In multimedia applications' point of view, frame is a smallest unit which has meaningful information, e.g. scene. In VCR operation, normal playback is a sole exception which may not require the prior knowledge on each frame location. Multiple speed playback, reverse playback, skip, scene editing, and etc. all require that the application randomly accesses the arbitrary frame. Beginning of each frame is identified by special symbol called "start code." Without any index information, locating the arbitrary frame real-time is practically infeasible due to its computational overhead involved in pattern matching. In practice multimedia file is normally accompanied by index information. While this approach resolves the problem, the underlying file system still suffers from the limitation of viewing the file as a collection of physical units (byte or block). Most of general purpose file systems is responsible for translating the (*file descriptor, file offset*) to (*device, device offset*), but does not provide any service to locate the logical units, e.g. frames. In database management systems, file system exports elaborate interfaces so that the application can perform complex file operations easily and efficiently. e.g. *insert record*, *delete record*, and *search*. We aim at developing dedicated file system for multimedia application, and thus providing more elaborate treatment for multimedia data. HERMES file system enables the user to view file as a collection of logical units (e.g. frames in video file). It exports a number of interfaces to handle the file in logical units. Application can access the file with (*file descriptor, frame index*).

HERMES i-node bears frame index structure (Fig. 3b). There are two different approaches in designing frame index. Index can contain either physical (device offset) or logical (file offset) location of the respective data unit. If frame index contains physical location of a frame, access to frame will be faster. In this approach, block references and frame references both contain physical location. Redundancy in physical location information makes the meta data update more time consuming. Even worse, crash recovery procedure can be very complicated. If frame index

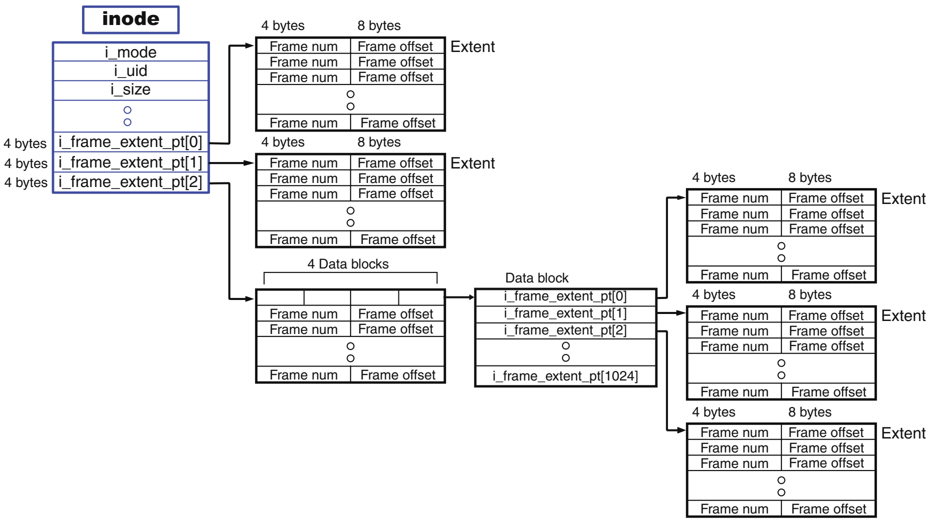


Fig. 5 Logical index structure of HERMES file system

contains logical location (file offset), access to frame becomes less efficient and goes through two phases: access to frame index information and deriving the physical block location from frame offset. However, file system becomes less vulnerable to error and crash recovery procedure is simpler. After careful deliberation, we decided to take the latter approach. In HERMES, frame index contains the byte distance from the beginning of the file. Figure 5 illustrates the structure of frame index in HERMES file system. Each i-node reserves three four byte pointers for logical indexing. The first two pointers point to an extent which contains frame number and frame offset pairs. The third pointer is for indirect access. The third pointer points to an extent whose first 16 KB contains pointers. There are logical 4,096 pointers ($\lfloor 16 \times 2^{10}/4 \rfloor$) in this region. Each pointer points to a data block and each data block contains pointers to extents. Let us assume that data block size and extent size be 4 KB and 1 MB, respectively. Then, single extent can cover 87,318 frames. There are three extents and thus over 260,000 frames can be covered by direct reference. 260,000 frames corresponds to approximately 144 min. In the extent pointed by third pointer, there exist four indirect pointers. It can cover 4,096 extents of frame pointers. Given that single extent can hold 87,318 pointers to frame, it can cover more than 357 million frames which corresponds to 3,300 h of video with 30 frame/s frame rate. HERMES file system exports *addindex* system call. It scans the multimedia file, computes the frame size and initializes the logical index fields.

5 Free block allocation

Free block allocation algorithm is one of the original features of HERMES file system. Since fragmentation brings critical performance degradation in sequential workload, special care needs to be taken to avoid any disk fragmentation. Most of the existing file system adopts some type of pre-allocation scheme to facilitate

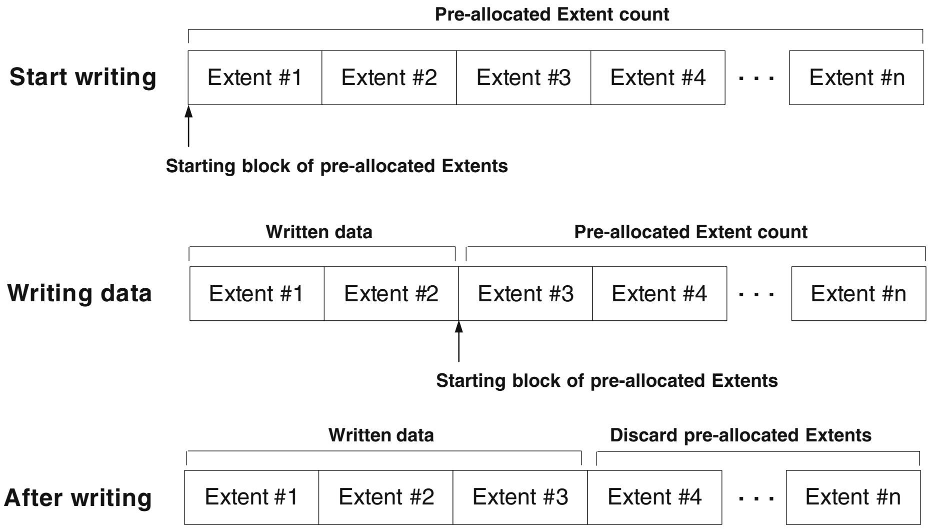

Pre-allocated Extent count
Start writing
Extent #1
Extent #2
Extent #3
Extent #4
Extent #n
Starting block of pre-allocated Extents
Written data
Pre-allocated Extent count
Writing data
Starting block of pre-allocated Extents
Written data
Discard pre-allocated Extents
After writing

Fig. 6 Free block allocation and data placement

the contiguous block placement. In case of EXT2 file system, it reserves eight consecutive blocks by default when it allocates a free block. We take more aggressive preallocation scheme to place the extents in consecutive fashion.

We define a global variable *preallocation-size* to denote the size of preallocation. HERMES i-node contains two fields: *i-preallocation-count* and *i-preallocation-start*. This value is determined at file system format phase and the default value is sixty four. Figure 6 illustrates the free block allocation and data placement scheme of HERMES file system. HERMES file system first searches preallocation-size number of consecutive free extents. If one exists, it reserves these extents. Otherwise, it reserves the largest consecutive extents. *I-preallocation-count* denotes the number of unused extents in the reserved chunk of extents. *I-preallocation-start* is a pointer to extent to store a incoming data (Fig. 7).

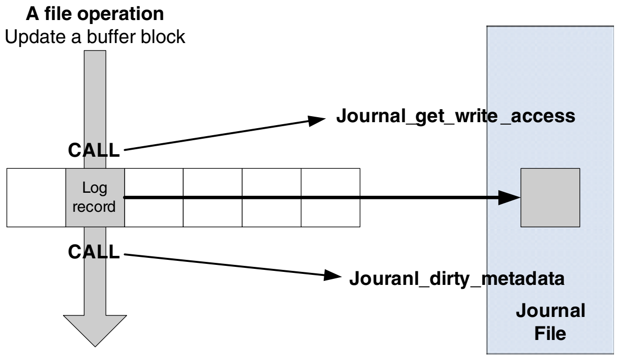
When we write data to the extent, HERMES decreases *i-preallocation-count* and *i-preallocation-start* pointer is updated to the next extent in the pre-allocated extents. After consuming all reserved extents, HERMES file system again searches for another chunk of consecutive extents. After file is closed, HERMES file system frees remaining unused pre-allocated extents. The block placement mechanism of HERMES file system helps to allocate data extents continuously. With this aggressive free block allocation scheme, HERMES file system can reduce disk seek and rotational latency and can guarantee timely transfer of the multimedia data blocks.

6 Journaling in HERMES

6.1 File system journaling

In most of modern operating system design, effort on exploiting disk bandwidth of the underlying I/O subsystem adopts aggressive buffer cache management with

Fig. 7 Logic record



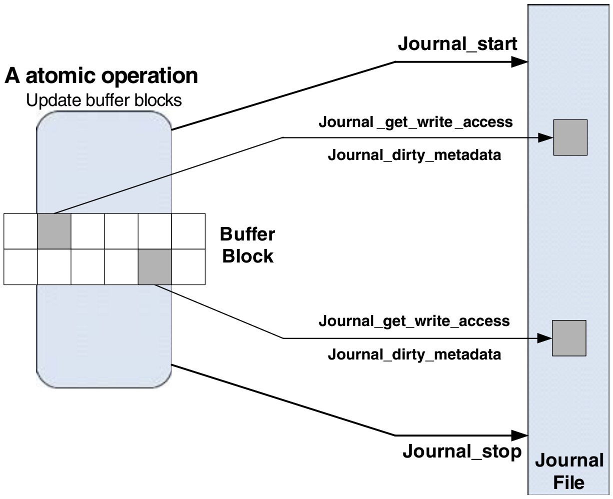
asynchronous I/O. However, this design approach increases the chance that the contents in buffer cache and the contents in the storage are inconsistent and that file system becomes more vulnerable to crash. This applies to both data and meta data. File system provides file system check utility (e.g. fsck) to detect any inconsistencies and to recover the file system. However, as disk size increases file system check operation takes prohibitively long (tens of minutes for hundreds of mega byte partition). Average consumer electronics user cannot tolerate this duration. Inconsistency in data affects only single file while inconsistency in meta-data can collapse the integrity of entire file system partition. Journaling file systems addresses this problem by writing out a special journal file, which keeps track of the transactions to the disk. Updates to the disk are committed automatically. If power is suddenly interrupted, a given set of updates will have either been fully committed to the file system, in which case there is not a problem, and the file system can be used immediately, or will be marked as not yet fully committed, in which case the file system driver can read the journal and fix any inconsistencies. This approach is much quicker than scanning of the entire hard disk, and guarantees that the structure of the file system is always self-consistent, even if power is interrupted or the system crashes at random times.

6.2 Designing journaling scheme for HERMES

The journaling scheme is designed as a part of HERMES file system to eliminate enormously long file system recovery times after an unexpected system crash. The journaling scheme of HERMES file system is very similar to the journaling scheme of EXT3 file system [23]. HERMES file system uses one i-node to store the journal file, and when formatting HERMES file system, the journal file is created and the size of the journal file is determined by the user. If it is not pre-determined, eight extents are used for the journal file. The role of the journal file is to record the new contents of file system metadata blocks. HERMES file system logs all structural file system metadata changes. This includes super block, i-node table blocks, i-node bitmap blocks, extent bitmap blocks, and directory extent blocks.

The journaling scheme of HERMES file system consists of three principles. The first is log record. A journal metadata block contains the entire contents of a single block of file system metadata which is updated. Logging a metadata block is basic journaling principle, which is called a log record. We want to guarantee consistent

Fig. 8 Atomic operation



metadata state after the event of a system crash. Consistent metadata state means a sequence of metadata changes which results from any single file system request made by an application, and contains all of the changed metadata resulting from that request. All log records after one file system request are merged and updated together. This principle is called atomic operation (Fig. 8). For example, the process of creating a new file modifies several metadata structures: i-nodes, i-node bitmaps, extent bitmaps, etc. All these updated metadata blocks are recorded in one atomic operation.

It is quite possible that file operations which are operated near sequence update same metadata blocks. Merging atomic operations lead to performance improvement. We do not have to write separate copies of metadata blocks which are updated frequently. Let us give an example. When we write a data to a file, the file will be expanded and the same bitmap will be updated repeatedly. By default, HERMES file system creates a new transaction every 5 s, and allows all atomic operations added to this transaction. Transaction is flushed to disk periodically by a kernel thread, which writes out to the journal file all metadata blocks which have been updated during one period. After the event of a crash, HERMES file system is restored to a consistent metadata state by replaying the journal file. Rather than examine all metadata by fsck, HERMES file system inspects only those portions of the metadata that have recently changed. Recovery is much faster and recovery time is not dependent on partition size.

7 Performance experiment

We examine the performance behavior of HERMES file system via physical experiment. HERMES file system is implemented on Linux operating system platform (Table 1). We compare the performance of three different file systems: HERMES, EXT2 and XFS(SGI) file system. We use three different benchmark programs: Streaming Workload generator, IOZONE benchmark [7] and LMBENCH bench-

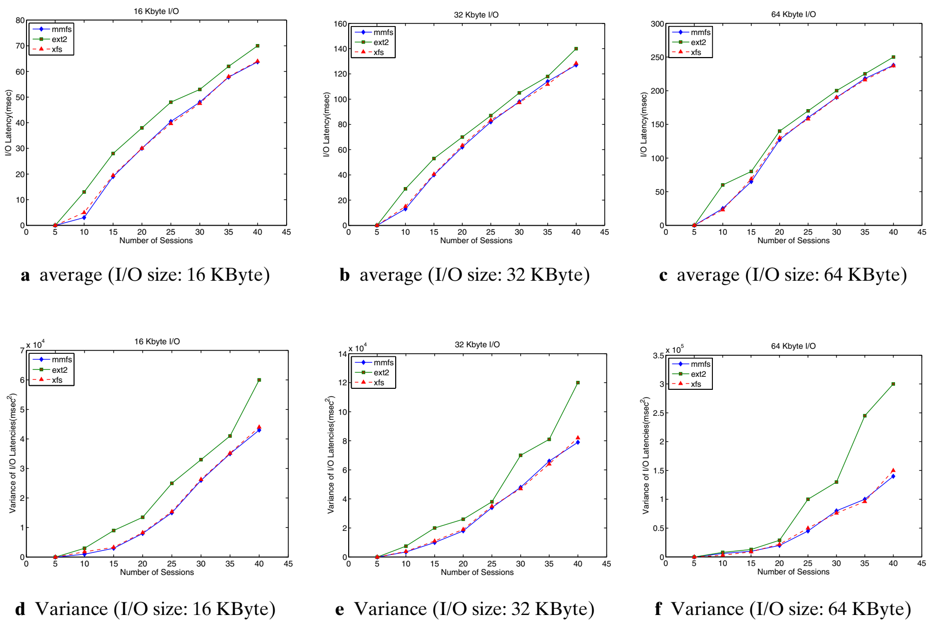
Table 1 Disk profile

Capacity	9.1 GB
Interface	Ultra 160 SCSI
Sector size	512 Byte
Rotational speed	7,200 RPM
Media transfer rate	248–400 Mb/s
Average seek time	6.8 ms
Track to track seek	0.6 ms
Full track seek	1.5 ms

mark [14]. To precisely capture the performance of storage subsystem, we flush the buffer cache before each experiment. Modern disk drive adapts zoning technique to exploit the linear bit density of the magnetic surface. This enhances the capacity of the disk. However, I/O latency can significantly vary dependent upon the location of the requested data block. For fair comparison, each file system partition starts from the same sector. The experiment is performed on dual Pentium III (746 MHz processor with 256 KB cache).

7.1 Streaming workload

We examine the I/O latencies of HERMES, EXT2 and XFS file system. Real-time multimedia workload is characterized by its sequential access pattern and exhibits higher degree of spatial locality. We examine two main metrics in this experiment:



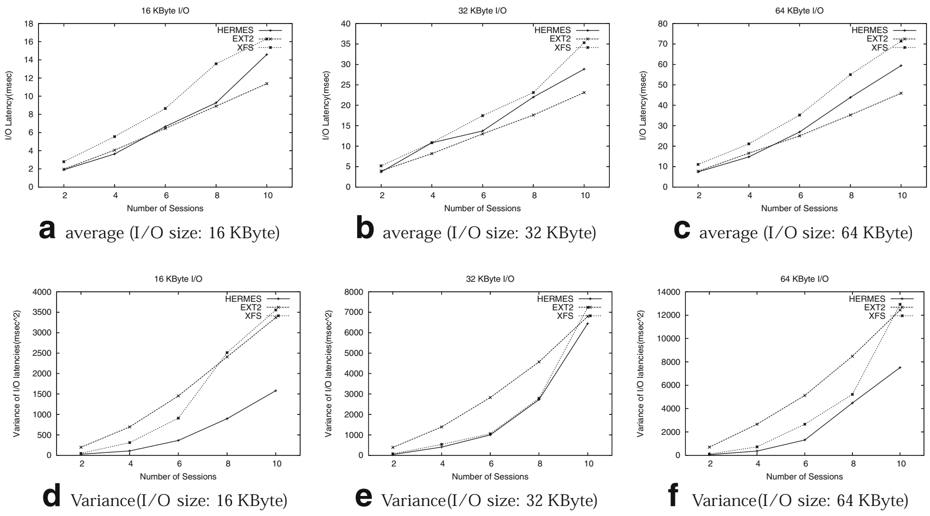
a average (I/O size: 16 KByte) **b** average (I/O size: 32 KByte) **c** average (I/O size: 64 KByte)

d Variance (I/O size: 16 KByte) **e** Variance (I/O size: 32 KByte) **f** Variance (I/O size: 64 KByte)

Fig. 9 I/O Latency under varying number of sessions: small size contents (30 MB each)

average and variance of I/O latency. We adjust the I/O size and number of sessions. We generate forty MPEG-2 files. Each I/O session sequentially scans the file. Modern disk drive adopts *zoning* technique to exploit linear bit density. When there exist a number of file system partitions in a single disk, it is possible that the partition in the outer side of the disk platter exhibit better I/O performance since it can store larger number of data in a track. Special care has been taken to create the HERMES, EXT2 and XFS partition in the identical cylindrical position. We examine the I/O behavior for small size contents for mobile streaming as well as large size contents for HDTV quality local playback. In case of the experiment for small size contents, forty 30 MB files are created in HERMES, EXT2, and XFS file system partitions, respectively. In case of the experiment for large size contents, ten 6 GB files are created in HERMES, EXT2, and XFS file system partitions.

Figures 9 and 10 illustrate the average and variance of I/O latency for small size contents and large size contents. We measure the I/O latency under three different I/O unit sizes: 16, 32, and 64 KB. X-axis is the number of concurrent sessions, and Y-axis is I/O latency. It enables us to examine if a certain file system behaves better under particular I/O unit size. We find the I/O latency increases with the I/O unit size in sub-linear fashion. We presume that this is because a certain fraction of I/O latency is from the operational overhead of the hard disk drive. As is shown, I/O latency in HERMES file system is approximately 60% of I/O latency in EXT2 file system in most I/O unit sizes. This performance benefit is result of harmony of simple file structure, aggressive block placement, and separation of directory blocks and file data blocks in HERMES file system. As the number of concurrent streams increases, the overhead of reading the indirect block continues more dominant fraction of the elapsed I/O time in the Linux file system. On the contrary, HERMES file system minimizes the disk seek overhead by prohibiting the usage of multi-level indirect reference and block placement mechanism. I/O latency in XFS file system is



a average (I/O size: 16 KByte) **b** average (I/O size: 32 KByte) **c** average (I/O size: 64 KByte)

d Variance(I/O size: 16 KByte) **e** Variance(I/O size: 32 KByte) **f** Variance(I/O size: 64 KByte)

Fig. 10 I/O latency under varying number of sessions: large contents (6 GB each)

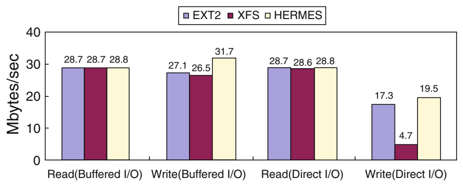
approximately similar to the latency in HERMES file system. In small size contents, HERMES and XFS exhibit similar performance while EXT2 exhibit slightly worse performance. In large size contents, the performance between HEMES and XFS becomes more clear. Aggressive preallocation scheme and large size I/O enables the HERMES to outperform XFS when relatively large number of streams are on-going with larger I/O unit.

The variance of I/O latency is important to maintain I/O latency more predictable. We can observe that the variance of HERMES file system is approximately 30% of EXT2 file system. In EXT2 file system, complex i-node structure along with multi-level data block organization and block group oriented placement strategy can make the latency of data block vary widely. On the other hand, HERMES file system has relatively flat structure and I/O latency remains relatively uniform. Variance of I/O in XFS file system is similar to that of HERMES file system under small size contents. With large size contents, the variance of I/O latency in HERMES becomes much smaller than the one in XFS.

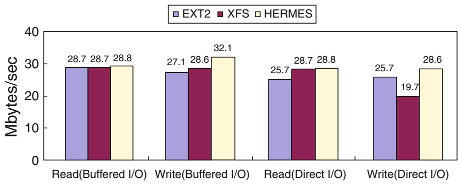
We can draw several conclusions from these results. As is shown, HERMES file system is superior to EXT2 file system. It can support A/V workload very efficiently. Another interesting result is that HERMES file system reduces the variance of I/O latency. It is the result of extent based structure, i-node structure of HERMES file system by avoiding multi-level indirect reference in locating the data block, and aggressive free block allocation strategy.

7.2 IOZONE benchmark test

The Iozone [7] benchmark tests how fast a file system can perform various file manipulation operations. It includes the sequential read/write, re-read/write, random read/write, reverse read, and etc. Sequential I/O tests reads and writes 2 GB file. We use two different I/O sizes: 4 and 64 KB. We also examine the performance effect of reading (or writing) the data blocks into buffer cache. Buffer cache is used to increase the chance that subsequent I/O requests are serviced from the buffered data block. We can save the memory traffic if we omit the process of copying the data blocks from I/O device to buffer cache and copy the data blocks directly to user space. There are pros and cons of neglecting buffer cache. HERMES file system is to be used in A/V device, e.g. set-top box, PVR, camcoder and etc. In these environments, application exhibits highly sequential workload characteristics. Therefore, it may not be unreasonable to assume that the advantage of using buffer cache is marginal. Memory traffic on system bus is another important issue. Copying the data blocks to



a I/O size = 4 KByte



b I/O size = 64 KByte

Fig. 11 Result of IOZONE benchmark, file size: 2 GB

Table 2 Summary of performance comparison: EXT2, XFS and HERMES

File system	I/O latency		Buffered I/O		Direct I/O	
	Average	Variance	Read	Write	Read	Write
EXT2	Poor	Poor	Good	Good	Good	Fair
XFS	Good	Good	Good	Good	Fair	Poor
HERMES	Good	Good	Good	Good	Good	Good

buffer cache increases the memory traffic compared with the case of bypassing it. This certainly increases the burden on the system bus. In some situation, this overhead has rather serious impact on overall system performance. Usually, embedded device is equipped with very limited hardware resources and it is not uncommon that the device does not even have DMA (direct memory access) controller or is equipped with low-end DMA controller. In this case, CPU should take the burden of moving the data blocks from I/O device to memory or between memory via iteratively performing load/store instructions.

Figure 11 illustrate the results of the Iozone benchmark. In all these test, HERMES exhibits superior performance. Performance of sequential buffered read in HERMES exceeds slightly the performance in EXT2 and XFS. The result of sequential Buffered write in HERMES is approximately 15% higher than in case of EXT2 and XFS. Result of direct read operation is similar with the result of buffered I/O. This is because both cases do not return until the DMA (direct memory access) operation completes. In case of direct write test, HERMES exhibits superior performance to using EXT2 and XFS. This phenomenon becomes more dominant in smaller I/O size. Buffered write exhibits higher performance than direct write operation. Table 2 presents the summary of comparisons of HERMES against two

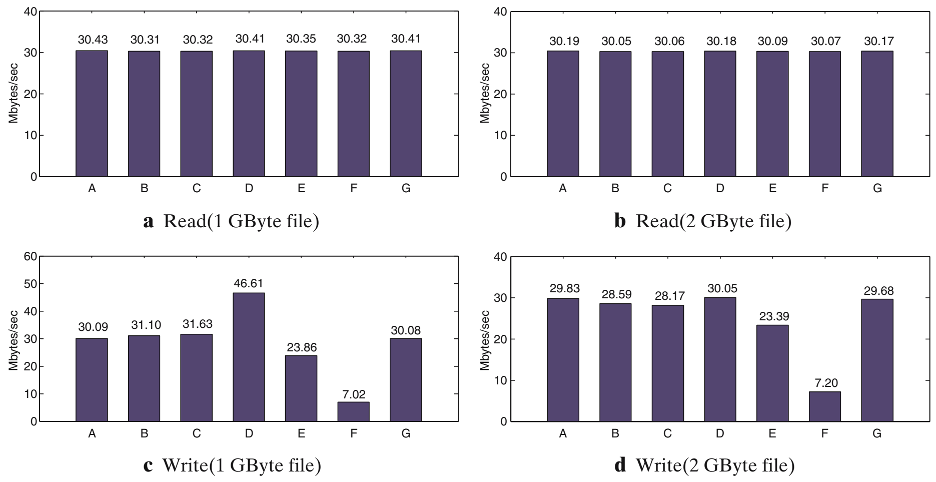


Fig. 12 Result of LMBENCH (I/O size: 8 KB); **a** raw disk, **b** EXT2 (buffered I/O), **c** XFS (buffered I/O), **d** HERMES (buffered I/O), **e** EXT2 (non-buffered I/O), **f** XFS (non-buffered I/O), **g** HERMES (non-buffered I/O)

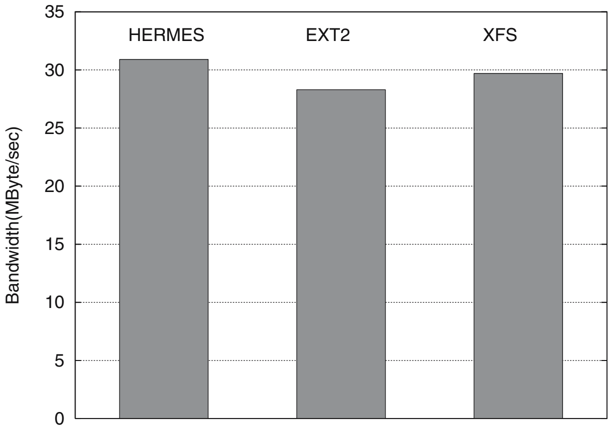


Fig. 13 Fast-forward like operation

popular file systems, EXT2 and XFS. As can be seen, HERMES yields very good performance in all aspects of the given file system performance metrics.

7.3 LMBENCH benchmark

The results of the LMBENCH [14] test are encouraging. LMBENCH benchmark consists of tools for system performance analysis. We use *lmdd* command which copies a specified input file to a specified output file. It is mainly for measuring disk and file system performance. We test read and write performance under different file sizes (1 and 2 GB) with 8 KB I/O. These tests are performed with two different I/O type: buffered I/O and direct I/O. We compare the I/O performance in following environments: raw disk partition, HERMES, EXT2, and XFS. For HERMES, EXT2, and XFS, we examine the performance of buffered I/O and non-buffered I/O.

Figure 12a and b illustrate the results of LMBENCH read test. The result of HERMES file system is slightly better than EXT2 and XFS file system. We can find that HERMES well exploits raw disk bandwidth. The results of write tests are

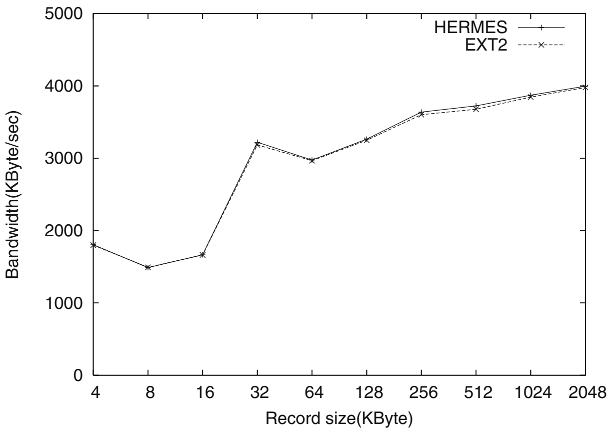


Fig. 14 Strided read in IOZONE

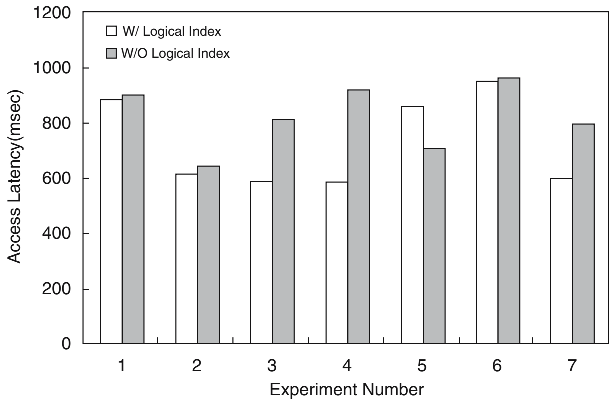


Fig. 15 Access latency with and without logical index

illustrated in Fig. 12c and d. In write test, HERMES file system exhibits dominant performance against to EXT2 and XFS. HERMES file system exhibits nearly 30 MB/s write performance in non-buffered write. On the other hand, EXT2 and XFS exhibits 23 and 7 MB/s write performance, respectively. It is found that meta data update and logging operation for XFS significantly affects its write performance. In case of buffered-write, the write performance can sometimes exceed the write performance of the physical device. The efficiency of HERMES file system is the result of smartly designed extent structure, aggressive block allocation and i-node structure. Details will be discussed in Section 7.6.

7.4 Performance of VCR-like operation

VCR operation include multiple speed reverse and forward playback (Fig. 13). We examine the performance of HERMES file system under VCR-like workload. We use two different workloads. First, VCR-like operation is simulated via repetition of read 1 MB and seek 5 MB operations. Second, we examine the performance under

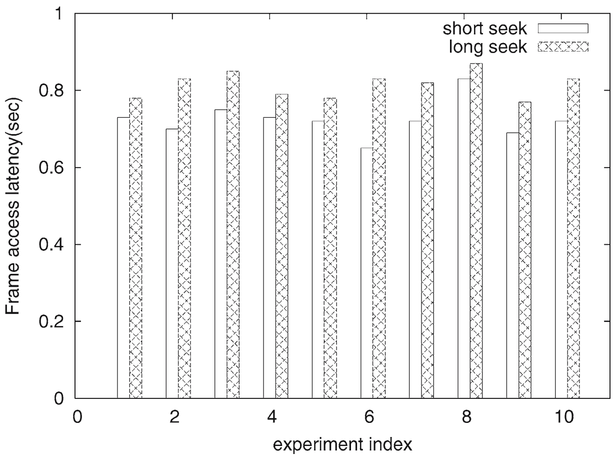
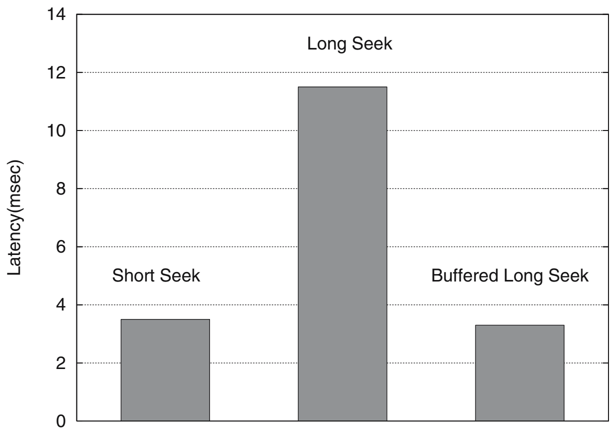


Fig. 16 Access latency of short and long seek with logical index

Fig. 17 Effect of buffered pointer blocks in indirect access

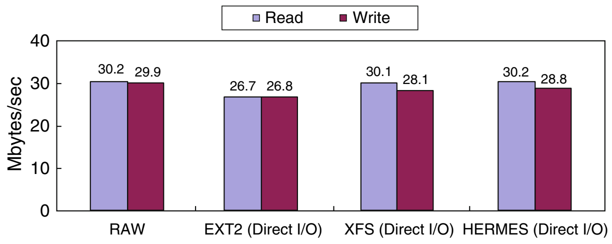


strided read test in IOZONE benchmark. *Strided read* test of IOZONE (Fig. 14) repeats reading a record and forward the file offset by 200 KB. *Strided operation* is also used to examine the performance of VCR-like operation. HERMES file system outperforms EXT and XFS file system.

7.5 Effectiveness of logical index and journaling

There are two main reason to support the logical index structure in HERMES. It can facilitates the fine grain access of the frame. In addition to that incorporating the logical index in the file system makes the disk head movement more efficient. Multimedia contents can be supplied with the index information. In this case, the application needs to access the contents as well as the index file. This can give rise to excessive seek overhead. We examine the effectiveness of the logical index of HERMES. We use the commodity MPEG player called XINE [29]. It can play MPEG-1 and MPEG-2 encoded files. The player has slide bar in the bottom part of the player. It enables the user to randomly access the content. When the user moves the slide bar, the player software estimates the respective offset in the file. With the offset, it locates the closest frame and starts playback starting from that frame. HERMES file system contains frame offsets in its meta data structure. We modify the XINE player to use the frame index information in random access. We measure the latency from the slide bar is set till the player starts to play from the respective frame. This latency includes I/O latency to access the respective frame, time to decode the respective frame, and time to render the decoded images on the screen. This latency is actually much larger than the I/O latency itself. We repeat the same experiment seven times. This is very practical indicator for effectiveness of the logical index. Figure 15 illustrates the result of the experiment. The average latency is 720 and 825 ms for with the logical index and without the logical index. Logical index improves the access speed by 15% on the average. Latency of accessing data block is subject to the number of intermediate pointer blocks. In HERMES file system, upto 270,000 frames can be accessed directly and the frames beyond that are accessed indirectly via intermedia pointer block. We examine the latency of short and long distance frame seek. From the beginning of a file,

Fig. 18 LMBENCH (file size: 2 GB, I/O unit size: 512 KB)



we access the 100,000th frame (short distance seek) and 300,000th frame (long distance seek) via logical index, respectively. We use the same method to measure the latency as in Fig. 15. Figure 16 illustrates the result of the experiment. User experiences approximately 10% longer latency in long distance seek. Once the pointer block is read into buffer cache, the latency of long distance seek decreases significantly. The latency of short distance seek and the long distance seek becomes approximately the same. Figure 17 illustrates the result of the experiment. Initially, short and long seek takes 3.3 and 11.2 ms, respectively. When we perform long seek operation again to different data block with pointer block being the same, then the latency of long seek drops to 3.3 ms. This is because the pointer block is cached into memory. Therefore, the overhead of access the indirect block is insignificant. HERMES journaling scheme successfully makes the file system robust against unexpected system crash. After system crashes, HERMES uses journaling information to examine the file system integrity and quickly recovers important meta data.

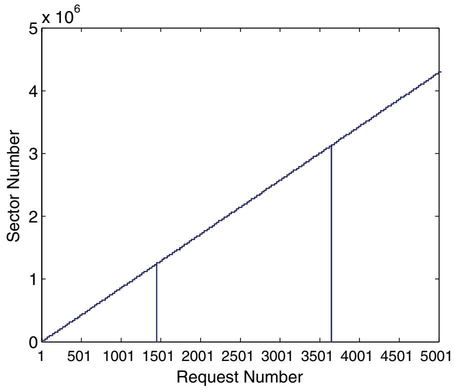
7.6 Anatomy of disk behavior

The result of several benchmark tests show that HERMES file system has superior performance to EXT2 and XFS file system. We examine why HERMES file system works better. We first visualize the disk head movement in each file system and perform trace driven simulation to closely examine the disk performance behavior. We perform LMBENCH test (2 GB file and 512 KB I/O) and extract I/O trace using Trace Tool [3]. Each disk trace record contains read/write flag, device major number, device minor number, number of requested sectors, requested sector number, and time of request in 10 ms unit. With this information, we can visualize the disk head movement. We feed the disk trace to Disksim [8] simulator and examine the various disk behavior. In particular, Disksim generates disk seek time, disk rotational

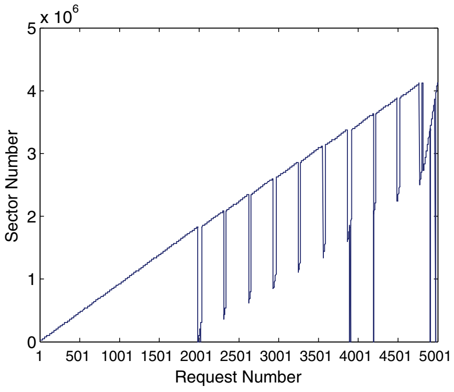
Table 3 Number of I/O requests

	Read (non-buffered I/O)	Write (non-buffered I/O)
LMBENCH	4,096	4,096
EXT2 (disk trace tool)	5,140	5,197
XFS (disk trace tool)	4,120	4,272
HERMES (disk trace tool)	4,098	4,103

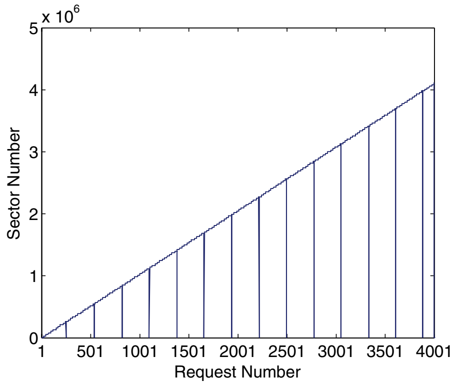
latency, disk positioning time and various system statistics. Each statistic consists of the average value, the standard deviation, and the maximum value.



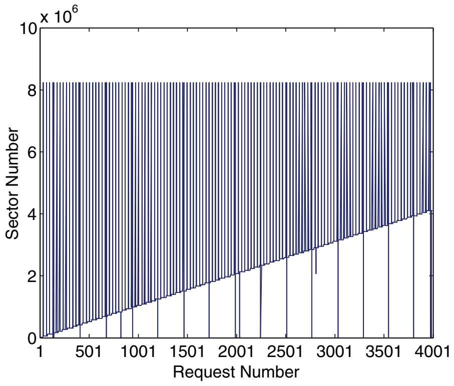
a EXT2(Read)



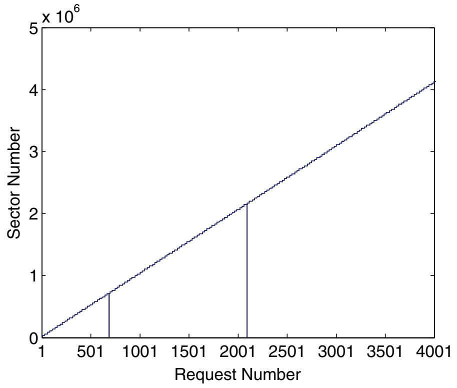
b EXT2(Write)



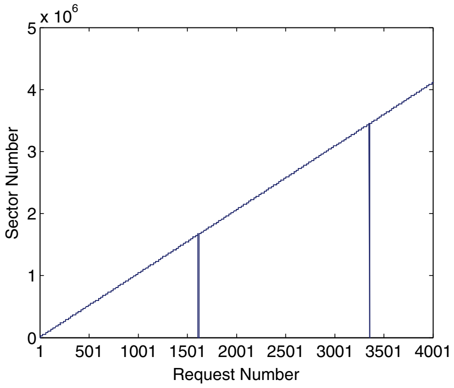
c XFS(Read)



d XFS(Write)



e HERMES(Read)



f HERMES(Write)

Fig. 19 Disk head movement for non buffered I/O

Figure 18 illustrates the result of LMBENCH benchmark of non-buffered I/O test. LMBENCH generates total of 4,096 I/O requests. However, we can find that more than 4096 requests are issued to disk (Table 3). This is due to meta data update operations. We can find that file system with complex meta data structure, e.g. EXT2 file system, splits one request in user level to two or more requests in disk I/O device. EXT2 file system issued more than five thousand requests to disk I/O device. HERMES file system, on the other hand, does not generate as many requests as EXT2 or XFS. This is because file system and meta data structure is simple and therefore writing data block does not accompany as many meta data update as in the case of XFS and HERMES.

Figure 19 illustrates disk head movement. X-axis denotes the request number and Y-axis denotes the respective sector number. As can be seen, most of the sector numbers linearly increase as I/O proceeds. However, in EXT2 file system, there exists occasional glitches. These glitches are at the beginning of the disk or beginning of the block group. EXT2 file system partitions the entire partition into group of cylinders called block group. Metadata information within block group is kept at the beginning of each block group. That is for the file meta data and file system meta data

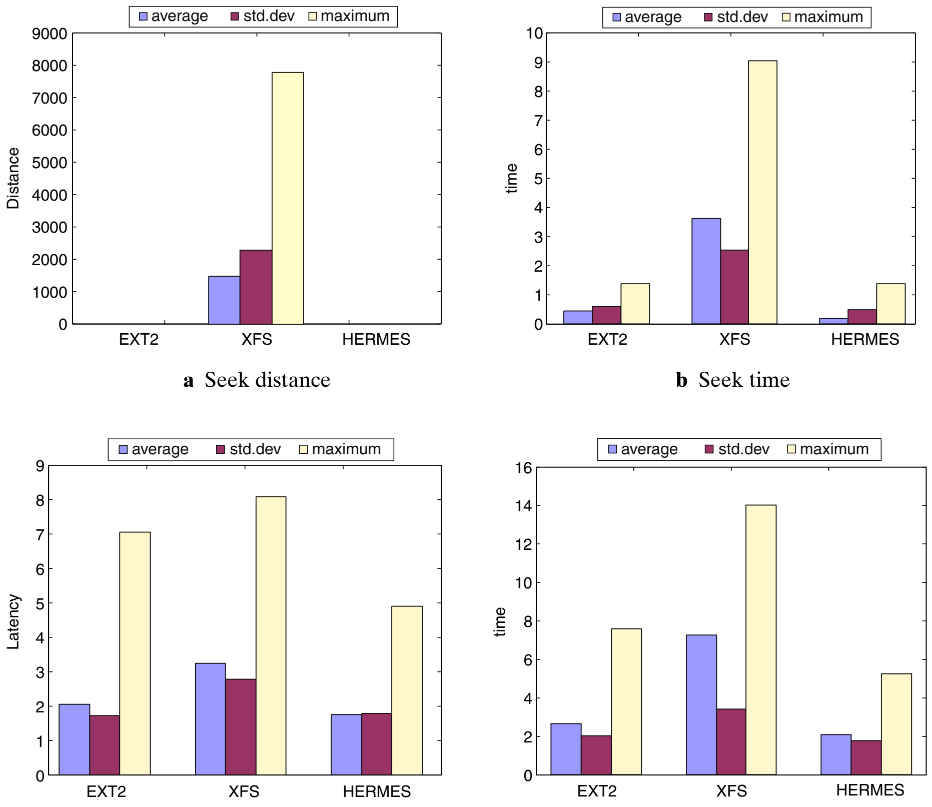
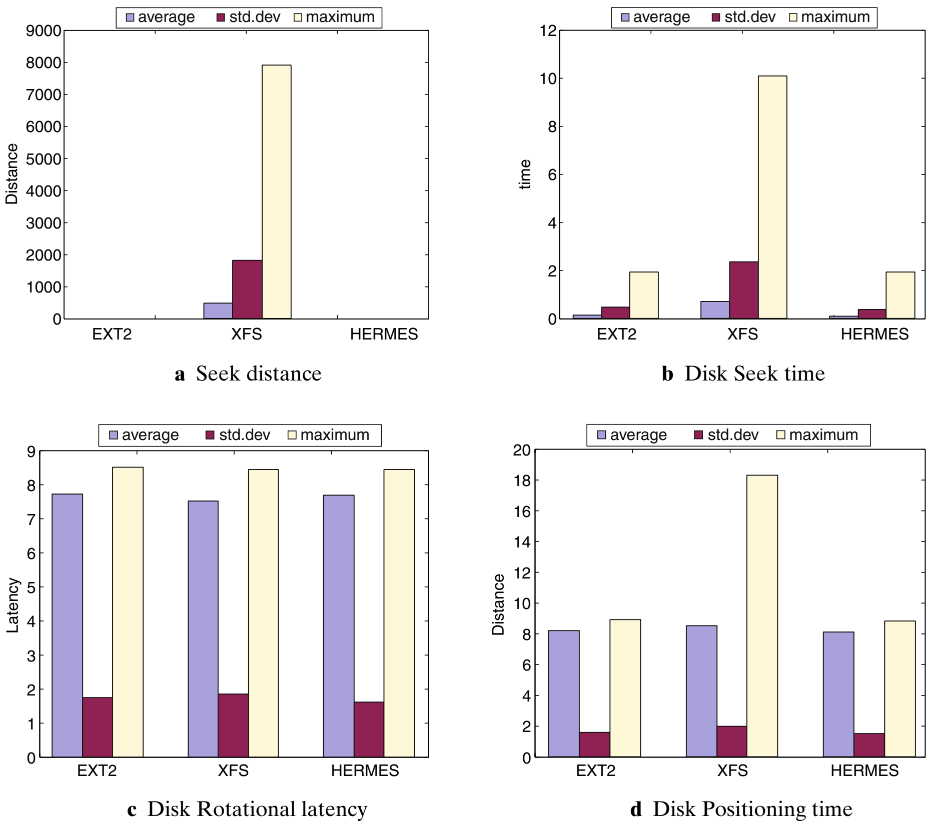


Fig. 20 Disk performance analysis: non-buffered read



a Seek distance **b** Disk Seek time **c** Disk Rotational latency **d** Disk Positioning time

Fig. 21 Disk performance analysis: non-buffered write

update. This is why position of meta data increases with the I/O requests. In XFS, it occasionally accesses the beginning of the partition as well as the end of the partition.

This detailed examination shows that the file system organization and file organization of HERMES are effectively designed to reduce the disk overhead. Figures 20 and 21 illustrates various statistics from Disksim simulator for read and write test. As is shown, HERMES file system yields the most efficient disk head movement overhead. In all statistics, HERMES file system is superior to EXT2 and XFS file system.

8 Conclusion

Embedded systems such as PVR, set-top box, HDTV put unique demand on I/O subsystem design. The underlying software, particularly file system, needs to be elaborately designed so that it can meet tight constraints of consumer electronics platform: performance, price, reliability, and etc. In this work, we develop state-of-art file system elaborately tailored for embedded environment of A/V workload.

There are two design objectives in our file system: performance and logical level abstraction. For performance optimization, we use extent based allocation, single level file structure with block index augmentation scheme and aggressive free block allocation. HERMES enables the user to view file as a collection of semantic units (frame or audio samples). Frame reference structures are elaborately designed so that it can encompass sufficient number of frames. HERMES file system exports a number of interfaces for frame level file treatments.

Via extensive physical experiment, we verify that HERMES file system successfully addresses a number of issues. HERMES file system exhibits very good scalability. To efficiently support several sequential I/O sessions, we carefully design the file system organization, free block allocation algorithm, data block placement strategy and file organization and placement. Performance experiment shows that that HERMES file system becomes more efficient as the number of sessions increases. HERMES file system bears relatively more predictable I/O latency. Skewed tree like file structure in legacy Unix family file system negatively affects the variance in I/O latency. We develop single level tree structure with pointer augmentation scheme. Experimental results confirm that HERMES file system yields much more predictable I/O behavior than other file systems. We closely examine the disk head movement behavior in HERMES file system. This is to analyze the meta data, file data, journaling data access patterns. This microscopic analysis and trace driven simulation reveals that the good performance of HERMES file system is originated from its efficient file system organization and layout. The result of performance experiments indicate that HERMES file system prototype successfully meet the file system constraints for high volume and high bandwidth multimedia application.

Acknowledgements In addition to authors, there a number of people who have made significant contribution in making HERMES file system into existence. Authors would like to thank Hyungkyu Jang and Jaemin Ryu for their dedicated work for HERMES file system. This work was supported by grant No. R08-2003-000-11104-0 from the Basic Research Program of the Korea Science & Engineering Foundation, SDR Research Center at Hanyang University and SRCCS at Seoul National University.

References

1. Ahn B-S, Sohn S-H, Kim C-Y, Cha G-I, Baek Y-C, Jung S-I, Kim M-J (2004) Implementation and evaluation of extns multimedia file system. In: Proceedings of ACM multimedia conference, New York, NY, USA, pp 588–595 (Oct)
2. Bolosky WJ, Fitzgerald RP, Douceur JR (1997) Distributed schedule management in the tiger video fileserver. ACM SIGOPS Operat Syst Rev, 31
3. Brigham Young University Performance Evaluation Laboratory. Dtb: Linux disk trace buffer. Available: http://traces.byu.edu/new/Tools/
4. Chen M-S, Kandlur DD, Yu PS (1993) Optimization of the grouped sweeping scheduling (GSS) with heterogeneous multimedia streams. In: ACM Multimedia '93, pp 235–242
5. Chiueh T, Niranjan TH, Schloss GA (1997) Implemenation and evaluation of a multimedia file system. In: Proceedings of international conference on multimedia computing and systems
6. Dimitrijevic Z, Rangaswami R (2003) Quality of service support for real-time storage systems. In: Proceedings of international IPSI-2003 conference (October)
7. File system benchmark tool. Available: http://www.iozone.org
8. Ganger GR, Worthington BL, Patt YN (1998) The disksim simulation environment. Technical report CSE-TR-358-98, Dept. of Electrical Engineering and Computer Science, Univ. of Michigan (February)

9. Gemmell D, Vin H, Kandlur D, Rangan P, Rowe L (1995) Multimedia storage servers: a tutorial. Computer 28(5):40–49 (May)
10. Haskin RL (1998) Tiger shark-a scalable file system for multimedia. IBM J Res Dev 42:185–197
11. Jeon J, Won Y, Ahn S (2001) Performance analysis of non-stationary model for empirical VBR process. In: IEEE Globecom, pp 2435–2439
12. Kim T, Won Y, Koh K (2005) Apollon: file system support for qos augmented i/o. In: Proceedings of pacific rim conference on multimedia (PCM '05) also in lecture note in computer science series from Springer, Jeju, Korea (Dec)
13. Lee W, Su D, Wijesekera D, Srivastava J, Kenchammana-Hosekote D, Foresti M (1997) Experimental evaluation of pfs continuous media file system. In: Proceedings of CIKM, Las Vegas, Nevada, USA, pp 246–253
14. McVoy L, Staelin C (1996) Lmbench: portable tools for performance analysis. In: Proceedings of USENIX 1996 annual technical conference, San Diego, California (Jan)
15. Mokbel MF, Aref WG, Elbassioni K, Kamel I (2004) Scalable multimedia disk scheduling. In: Proceedings of the 20th international conference on data engineering, pp 498–509 (March)
16. Ozden B, Biliris A, Rastogi R, Silberschatz A (1994) A low-cost storage server for movie on demand databases. In Proc. of VLDB '94
17. Park J, Won Y, Srivastava J (2001) SMART: yet another file system for multimedia streaming. In: Proceedings of international conference on distributed multimedia systems, Taipei, Taiwan (Sep)
18. Rangan P, Vin H, Ramanathan S (1992) Designing an on-demand multimedia service. IEEE Commun Mag 30(7):56–65 (July)
19. Rompogiannakis Y, Nerjes G, Muth P, Paterakis M, Triantafillou P, Weikum G (1998) Disk scheduling for mixed-media workloads in a multimedia server. In: Proceedings of ACM multimedia '98, Bristol, UK, pp 297–302
20. Shenoy PJ, Goyal P, Rao SS, Vin HM (1998) Symphony: an integrated multimedia file system. In: Proceedings of SPIE/ACM conference on multimedia computing and networking (MMCN'98), San Jose, CA, USA, pp 124–138 (Jan)
21. Shenoy PJ, Vin HM (1998) Cello: disk scheduling framework for next generation operating system. In: Proceedings of ACM SIGMETRICS, Madison, WI, USA, pp 44–55
22. Sweeney A (1996) Scalability in the xfs file system. In: Proceedings of USENIX annual technical conference, San Diego, CA, USA (Jan)
23. Tweedie S (1998) Journaling the linux ext2fs filesystem. In: LinuxExpo '98
24. Wang C, Goebel V, Plagemann T (1999) Techniques to increase disk access locality in the minorca multimedia file system. In: Proceedings of the 7^{th} ACM multimedia
25. Wijayaratne R, Reddy ALN (2001) System support for providing integrated services from networked multimedia storage servers. Presented at ACM multimedia
26. Won Y, Park J, Ma S (2002) Hermes: file system support for multimedia streaming in internet home appliance. Lect Notes Comput Sci 2510:484–500 (Oct)
27. Won Y, Ryu YS (2000) Handling sporadic tasks in multimedia file system. In: Proc. of ACM multimedia conference '00, Los Angelses, CA, USA
28. Won Y, Srivastava J (2000) SMDP: minimizing buffer requirements for continuous media servers. Multimedia Syst 8(2):105–117
29. xine player. http://xinehq.de
30. Zimmermann R, Fu K (2003) Comprehensive statistical admission control for streaming media servers. In: Proceedings of 11^{th} ACM multimedia conference, Berkeley, CA, USA, pp 75–85

Youjip Won received the B.S. and the M.S. degree in Computer Science from Department of Computer Science and Statistics, Seoul National University, Korea in 1990 and 1992, respectively and Ph.D in Computer Science from the University of Minnesota, Minneapolis in 1997. After graduation, he worked for Server Architecture Lab, Intel as Server Performance Analyst till 1999. Since 1999, he has been on board of faculty members in Dept. of Electrical and Computer Engineering, Hanyang University, Seoul, Korea, where he is now Associate Professor. His research interests include Operating System, Computer Networks, Multimedia, Performance Analysis.

Doohan Kim received the B.S. degree in Electronics Engineering from KyungWon University in 2002. and M.S. degree in Dept. of Electrical and Computer Engineering from Hanyang University in 2004. He is currently working for Samsung Electronics GSM Mobile Telecommunication Business as a development engineer. His research interests include Operating System, Embedded System and Computer Architecture.

Jinyoun Park received the B.S. degree in Electronics Engineering from Hanyang University in 2000. and M.S. degree in Dept. of Electrical and Computer Engineering from Hanyang University in 2002. He is currently working for LG Electronics Digital Media Lab as a embedded system engineer. His research interests include embedded system and wireless network.

Sichang Lee received the B.E. degree in Computer Engineering from Department of Electronic Computer Engineering, Dankook University, Korea in 2003 and the M.S. degree in Computer Science from Department of Electronic Computer Engineering, Hanyang University, Korea in 2006. He is currently working for Network R & D Center, Tellion, Seoul, Korea, where he is now junior engineer. His research interests include Computer Architecture, Operating System, Computer Networks.