HERMES: File System Support for Multimedia Streaming in Information Home Appliance*

Youjip Won[1], Jinyoun Park[1], and Sangback Ma[2]

[1]Div. of Electrical and Computer Engineering, Hanyang University, Korea
{yjwon|jypark}@ece.hanyang.ac.kr
[2]Department of Computer Science, Hanyang University, Korea
sangback@cse.hanyang.ac.kr

Abstract. The HERMES file system is state-of-the-art file system designed to handle multimedia streaming workload in consumer electronics platform. The design objective of HERMES is to minimize the *delay* and the *delay variance* of I/O request in the sequential workload. File organization, meta data structure, unit of storage, or etc. are elaborately tailored to achieve this objective. Further, HERMES provides a number of API's specific to mpeg-4 file format. It can greatly facilitate the development of multimedia applications. For the seamless integration with the existing application, HERMES file system is developed under *virtual file system(VFS)* layer. Prototype of HERMES file system is implemented on Linux operating system. Our benchmark test shows that HERMES file system exhibits suprior performance than EXT2 file system.

Keywords: Multimedia, Streaming, File System, UFS, Scheduling

1 Introduction

1.1 Motivation

Information Appliance for digital video can be thought as a lightweight 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, etc. The disk drive in this device is not an exception. It is not feasible to use high performance disk in this type of device. The realtime playback and the retrieval 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 characteristics of multimedia workload. Unfortunately, the fundamental design philosophy of the

* This work was supported by Korea Research Foundation Grant(KRF-2000-003-E00322.

most commodity file systems, e.g. Unix File System, NTFS, EXT2, FAT32, or etc. is ill-suited for meeting the real-time performance requirement of the audio and video data retrieval. One of the reasons is that navigating through multi-level tree structured file entails non-trivial amount of the disk head movement overhead in visiting internal nodes of the tree.

Efficiency of the underlying file system plays a critical role in providing the streaming service in cost effective manner. To effectively exploit the physical bandwidth of the disk, it is important that the file system layout, meta data structure, file organization, file placement, etc. are elaborately tailored so that disk fragmentation is avoided and the time to locate the data blocks is minimized. Special care needs to be taken in designing the file system to meet the requirement of the underlying workload. There is not much debate that the Unix file system is a landmark achievement in modern file system design. However, regarding the multimedia data retrieval, there are two major issues in Unix file system which requires further elaboration: (i) file structure and (ii) data abstraction. Unix file system abstracts the file as a sequence of byte stream. Mpeg-4 compressed file consists of a set of atoms and each atom may contain video data, audio data, text data, file meta data, or etc. Since HERMES has well defined target workload, it is possible to provide more specific sets of API's for the target application. We provide a set of file system level API's which can extract or record mpeg-4 specific information in HERMES In this work, we present the novel file system which effectively address the above mentioned issues: (i) file system layout, (ii) file organization, and (iii) data abstraction.

1.2 Related Works

Since legacy SCAN, FIFO, and their bifurcations 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 works address these issues and propose the disk scheduling algorithms for multimedia data retrieval[1, 3, 7, 2, 5].

There are a number of prototype file systems which are designed to handle multimedia data[8, 12]. MMFS[10] improves interactive playback performance by supporting intelligent pre-fetching, state-based caching, prioritized real-time disk scheduling, and synchronized multi-stream retrieval. Minorca Multimedia file system[11] proposed (i) a new disk layout and data allocation techniques called MOSA which offers a high degree of contiguous allocation for large continuous media files and allows the coexistence of small, non-CM files, and (ii) 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[4] introduces the idea of storing the data based on the semantic unit. The unit of placement is extent which consists of fixed number of semantic units. The size of file is limited to one extent. SMART file system[6] maintains a file as a linked list of extents and thus improves the file size limitation in Presto[6]. Symphony[9] also allows each video file to be accessed either as a sequence of bytes 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 clusters the index block and data block together, which may look like B tree.

2 Synopsis: Unix File System

In Unix file system, the management information is kept strictly apart from the data and is collected in a separate structure for each file. This structure is called ”i-node” and stores the metadata information of each file as well as data block references for actual data location. Data references consist of twelve direct references, one indirect reference, two-step indirect reference and three-step indirect reference. Given that multimedia file can easily go beyond a tens of mega bytes, data block retrieval in multimedia streaming operation 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, retrieval of pointer blocks gives substantial overhead in the streaming operation.

Since disk needs to access the i-node block and possibly a number of indirect blocks to access the data block, non trivial amount of additional overhead occurs. Even though the i-node and pointer blocks are in the buffer cache, memory access time can consume significant fraction of CPU cycle. It is very unlikely that pointer blocks and data blocks are stored consecutively, especially, when a number of files co-exist in the file system. Disk head needs to travel across the platter to retrieve i-nodes and pointer blocks.

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 closely as possible. EXT2 file system, which is the most widely used file system in Linux operating system, uses the concept of block group. Using the concept of block group, file system can cluster the data blocks for a file within relatively closer cylindrical position. Unfortunately, block group based placement policy still splits the file into different block groups when the size of files exceeds certain limit and may suffer from significant overhead in disk seek. EXT2 file system 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.

3 HERMES: Multimedia File System

UFS (Unix File System) manages both directory file and multimedia file equally. It is possible that multimedia files and the directory files are placed in the disk in interleaved fashion. This can cause substantial overhead in sequential scanning operation on multimedia file. To resolve this issue, HERMES file system maintains the directory block and data block seperately. Fig. 1 illustrates the file system layout of HERMES file system. Its partitions consist of super block, extent bitmap, i-node bitmap, i-node tables, directory extents and multimedia extents.

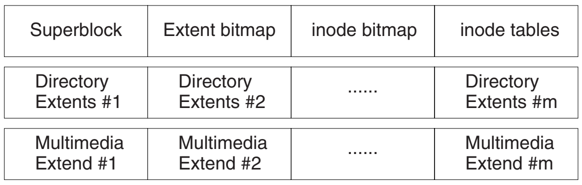


Fig. 1. Layouts of HERMES file system

Superblock is located at the first block of the file system partition and stores general information of the file system. It contains the information about the number of extent, the number of multimedia extent, the number of free extent, size of extent, the number of i-node, creation time, and etc. Extent is the smallest allocation unit and consists of consecutive data blocks. Extent size is determined when formatting file system and cannot be altered unless the file system is re-formatted. The directory entry is stored in directory extent and the multimedia file is stored in multimedia extent. By separating directory entry from multimedia data region, the HERMES file system can reduce the disk seek time and also store and retrieve multimedia data very efficiently. Extent bitmap is used to denote whether the respective extent is in use or not. Free extent is allocated using first fit algorithm. The i-node table consists of a predefined number of i-nodes. All i-nodes have the same size, 128 bytes. Each i-node maintains the file meta-data information: owner ID, group ID, file mode and references of allocated extent, etc. With this design approach, we like to achieve the followings: (i) efficient handling of multimedia playback workload, (ii) minimizing I/O latency, and (iii) supporting wide range of file size.

Fig. 2 illustrates the i-node structure of HERMES file system. We improve the i-node design proposed in [11]. The block reference pointer in UFS i-node points to single block, which can be either data block or pointer block. UFS adopts skewed tree-like file structure to cover large size file. In HERMES, we like to avoid using multi-level indirect reference in locating the data block. In HERMES, each block reference pointer can point to cluster of data blocks. Block reference pointer is augmented with `i_count` which denotes the number of consecutive data blocks as in Fig. 2.

4 Operating System Support for Streaming Operation

4.1 VFS and HERMES

HERMES file system is implemented under VFS layer so that the existing application can use HERMES file system partition without any modification. Fig. 3

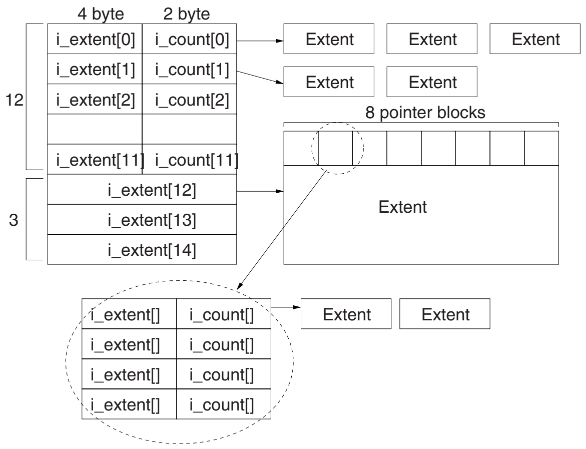


Fig. 2. i-node structure of HERMES file system

illusrates the relationship between operation system kernel, VFS and HERMES. As can be seen in Fig. 3, user can manipulate the files in HERMES in vari-

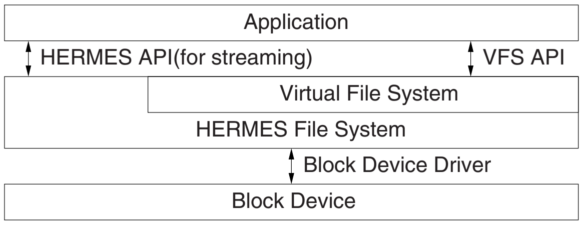


Fig. 3. Structure of HERMES file system

ous ways. It can use the existing API's provided by VFS layer. Also, HERMES provides a number of API's which is dedicated to handle multimedia specific information. The application can directly access these API's to manipulate the multimedia file.

4.2 Interface for Multimedia Streaming

HERMES defines a set of application programming interfaces to handle multimedia streaming workload. These interfaces facilitate the handling of multimedia data in more efficient manner. It achieves the following objective: (i) organizing the file as a collection of LDU(logical data unit), (ii) supporting QoS related data structure and (iii) management of streaming operation specific meta-data.

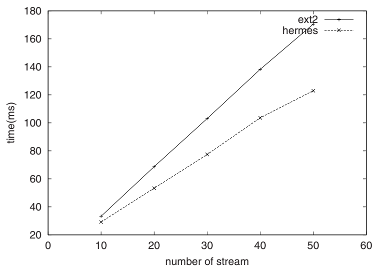
HERMES defines `mminfo` structure to convey the meta information related to streaming operation. The information includes the direction(forward or backward) or speed(x1, x2,...) of playback. The information of `mminfo` is used for scheduling the data retrieval operation in HERMES file system. The `mp4track` structure stores the information about the multimedia data track. The information shows track ID, creation time, modification time, the duration and time scale. The HERMES file system offers a set of kernel level API's. Table **??** illustrates the API's. When opening a file, `mp4open` enables the application to specify the information related to mutlimedia playback, e.g. QoS, playback rate, playback speed, etc. This information is carried in `mminfo` structure. `mp4GetMovieIOD` retrieves *IOD(Initial Object Descriptor)* in MPEG-4 file and store this information into `iniitalOD`. The size of IOD is stored into `pIoDLenght`. `int mp4GetTrackCount` retrieves the number of tracks in MPEG-4 file and stores into nCount. `mp4GetMovieTrack` retrieves information of the specified track in MPEG-4 file. `mp4read` reads the samples in a track. The sample can be video frames or the sequence of audio samples. The sample data and its size are read into `buffer` data structure. `mp4write` stores the multimedia samples in a track.

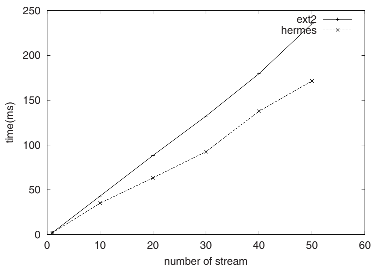
5 Performance Experiment

In this section we examine the performance behavior of the proposed file system. The HERMES file system is implemented on the Linux operating system. Performance of the file system is measured via experiments with a streaming workload. The experiment is performed on dual Pentium III (Coppermine) 746MHz processor with 256 Kbyte cache. The system has the 4 Ultra-Wide SCSI hard disks, each with 9.1 Gbytes disk space. The disk model is IBM Ultra star 36LP. A simulation program is written to sequentially scan the entire file. We vary the number of concurrent streams in the experiments.

We compare the I/O latency between HERMES and EXT2 file system. Since the streaming workload usually scans the file sequentially, it exhibits higher degree of spatial locality. Fifty MPEG-4 files with 50Mbyte each are created in both EXT2 file system partition and HERMES file system partition. In EXT2 file system, 570Mbytes file consist of 12 direct reference and 143 single indirect pointer blocks. We measure the the I/O latency between HERMES file system and EXT2 file system varying the number of concurrent streams. Fig. 4 illustrates the result of experiment. As is shown, I/O latency in the HERMES file system is approximately 70% of the latency in EXT2 file system. This is because as the number of concurrent sessions increases, the overhead of reading the indirect block constitutes 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.

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. In contrast, HERMES file system has relatively flat structure in organizing the data blocks. Thus, I/O latency remains relatively uniform. We measure the variance of I/O lantecy under different number of





(a) I/O Unit size: 16 KByte (b) I/O Unit Size: 64 KByte

Fig. 4. Scalability Test: I/O latency of the streaming operation

concurrent streaming session. Fig. 5 illustrates the result of experiment. We can observe that the variance of HERMES is much smaller than EXT2 file system.

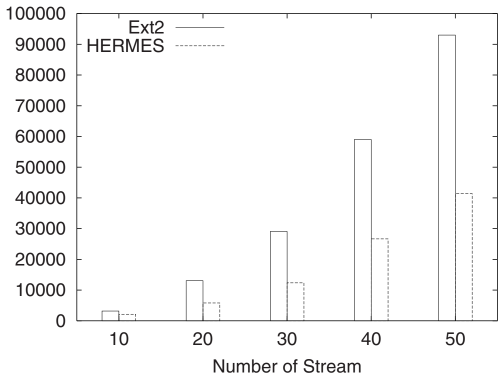


Fig. 5. Variance of I/O latency, 64 KByte I/O

6 Conclusion

In this work, we focus our effort in devising efficient file system for streaming operation and analyze its performance behavior under streaming workload. There are two design objectives in our file system. First, it should be able to support sequential access efficiently. To achieve this objective, we avoid using multi-level tree like structure. Single file is organized as a collection of data unit groups. Data unit group contains a fixed number of data units. Instead of using small size data unit(block), each data unit group consists of a collection of semantic data units. Second, the file organization is developed to handle relatively large files(tens of Mbyte), which are commonly found in multimedia applications. Our file system is implemented on the Linux platform. We examine the performance of the given file system under streaming workload and compares it with the performance of the EXT2 file system. There are a number of distinctive features which deserves attention: HERMES file system has more predictable behavior and thus is more robust against jitter. It services the I/O request with more evenly distributed latency compared to legacy UFS. We found that HERMES file system is more scalable compared to legacy UFS. The performance gap between HERMES and EXT2 becomes more dominant as there are more number of concurrent. The result of performance experiments indicates that the HERMES file system prototype successfully meet the file system constraints for multimedia streaming application.

References

1. Mon-Song Chen, Dilip D. Kandlur, and Philip S. Yu. Optimization of the grouped sweeping scheduling(gss) with heterogeneous multimedia streams. In *ACM Multimedia '93*, pages 235 – 242, 1993.
2. D. Gemmell, H. Vin, D. Kandlur, P. Rangan, and L. Rowe. Multimedia Storage Servers: A Tutorial. *COMPUTER*, 28(5):40–49, May 1995.
3. D.R. Kenchammana-Hosekote and J. Srivastava. Scheduling Continuous Media on a Video-On-Demand Server. In *Proc. of International Conference on Multi-media Computing and Systems*, Boston, MA, May 1994. IEEE.
4. Wonjun Lee, Difu Su, Duminda Wijesekera, Jaideep Srivastava, Deepak Kenchammana-Hosekote, and Mark Foresti. Experimental evaluation of pfs continuous media file system. In *Proceedings of CIKM*, pages 246–253, Las Vegas, Nevada, USA, 1997.
5. B Ozden, A. Biliris, R. Rastogi, and Avi Silberschatz. A Low-Cost Storage Server for Movie on Demand Databases. In *Proc. of VLDB '94*, 1994.
6. Jinyoun Park, Youjip Won, and Jaideep Srivastava. Smart: Yet another file system for multimedia streaming. In *Proceedings of International Conference on Distributed Multimedia Systems*, Taipei, Taiwan, Sep. 2001.
7. P. Rangan, H. Vin, and S. Ramanathan. Designing an on-demand multimedia service. *IEEE Communication Magazine*, 30(7):56–65, July 1992.
8. R.L.Haskin. Tiger shark-a scalable file system for multimedia. *IBM Journal of Research and Development*, 42:185–197, 1998.
9. Prashant J. Shenoy, Pawan Goyal, Sriram S. Rao, and Harrick M. Vin. Symphony: An integrated multimedia file system. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking(MMCN'98)*, pages 124–138, San Jose, CA, USA, Jan 1998.
10. T. Chiueh T.H. Niranjan and G. A. Schloss. Implemenation and evaluation of a multimedia file system. In *Proceedings of International Conference On Multimedia Computing and Systems*, 1997.
11. C. Wang, V. Goebel, and T. Plagemann. Techniques to increase disk access locality in the minorca multimedia file system. In *Proceedings of the 7th ACM Multimedia*, 1999.
12. R.P.Fitzgerald W.J.Bolosky and J.R.Douceur. Distributed schedule management in the tiger video fileserver. *ACM SIGOPS Operating Systems Review*, 31, 1997.