플래시 메모리 저장 장치에서 멀티미디어 데이터의 실시간 재생

한용철, 양학모, 류연승
명지대학교 컴퓨터소프트웨어학과
e-mail : hanlz1976@yahoo.com.cn, ysryu@mju.ac.kr

Real-Time Retrieval of Multimedia Data from Flash Memory Storage Devices

Lyong-cheol Han, Hak-mo Yang and Yeon-seung Ryu
Department of Computer Software, Myongji University

요 약

Recently, flash memory is becoming popular as storage system to store and retrieve multimedia files. However, there are few researches about multimedia file system for flash memory based storage devices. We have been designing and developing a novel multimedia file systems for flash memory. In this paper, we describe the semantics of real-time retrieval of multimedia data and present scheduling scheme to guarantee the real-time requirements in our multimedia file system.

1. Introduction

With advances in computer hardware and compression schemes, multimedia applications to store and transmit voice, video, and graphics are becoming an part of our daily computational life. Multimedia applications require data to be stored or retrieved at a certain rate. For example, video data compressed using the MPEG-1 requires a playback rate of 1.5 Mbps. Thus, storage system should be able to handle real-time retrieval of multimedia data.

Recently, flash memory is becoming important as nonvolatile storages for digital computing devices because of its superiority in fast access speeds, low power consumption, shock resistance, high reliability, small size, and light weight. Because of these attractive features, flash memory will be widely used in consumer electronics, embedded systems, and mobile computers. Examples are digital cameras, cellular phones, notebooks, hand-held devices, PDAs, etc.

This work was supported by grant No. R08-2004-000-10391-0 from Ministry of Science and Technology.

Table 1. Characteristics of storage media

Operation / Media	Access Time		
	Read	Write	Erase
DRAM	2.56 μs	2.56 μs	
NOR Flash	14.4 μs	3.53 ms	1.2 s (128KB)
NAND Flash	35.9 μs	226 μs	2 ms (16KB)
Micro Drive	21.4 ms	21.4 ms	

Though flash memory has many advantages, its special hardware characteristics impose design challenges on storage systems. For example, flash memory cannot be written over existing data unless erased in advance. To erase the flash memory, the whole flash memory storage space is partitioned into fixed-size erasure units and each erasure unit is the basic unit in erasing the flash memory. Hardware manufacturers define the erasure unit sizes (e.g., 64 Kbytes or 8 Kbytes). The erase operation is slow and consumes comparatively lots of power. Besides, the number of times an erasure unit can be erased is also limited (e.g., 10,000 to 1,000,000 times).

To overcome the limitations from hardware characteristics, special software called flash translation layer has been developed and novel file systems have beed studied.

Recently the decreasing of price and the increasing of capacity, flash memory is used as storage system to store and retrieve multimedia files. However, there are few researches about multimedia file system for flash memory based storage devices. In this paper, we describe the real-time retrieval of multimedia data in file system. The rest of this paper is organized as follows. In Section 2, we review the background that are relevant for this work. In Section 3, we give the overviews of multimedia file system for flash memory. We present the semantics of multimedia file retrieval and its scheduling method in Section 4 and 5, respectively. The conclusions of this paper are given in Section 6.

2. Background

A NAND flash memory is organized in terms of *blocks*, where each block is of a fixed number of *pages*. A block is the smallest unit of erase operation, while reads and writes are handled by pages. The size of page is fixed from 512B to 2KB and the size of block is somewhere between 4KB and 128KB depending on the product. There is a *spare area* appended to every page, where out-of-band data could be written. The typical size of spare area is 16B or 64B depending on the product. When the free space on flash memory is written, the space cannot be updated unless it is erased.

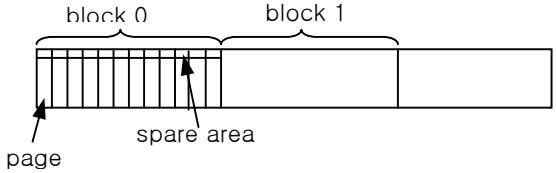

Figure 1. Flash memory organization

To overcome the limitations from hardware characteristics (i.e., bulk erasure, slow erasure, and limited endurance), as to longer flash memory lifetime, attain better system performance, and save power, the erase operations should be avoided as much as possible. The erasure units should be erased evenly to maximize the flash memory lifetime. This is because if the flash memory is not evenly erased, hot spots will soon be worn out, even though the utilization of this flash memory is still low. Besides, due to the disproportionate read/write speed, write access to flash memory should be reduced as possible as well. These are the primary principles in implementing flash-memory based storage systems.

Flash-memory based storage systems are commonly implemented in two ways: either designed from the scratch or constructed by using existing file systems plus new device driver implementations. In the later approach, the file system just generates requests to the drivers in disk block and sector numbers and device drivers then translate the disk block accesses to flash memory addresses. Evidently, no matter which approach is used, it is important to avoid having to erase during every data update. The *non-in-place update* mechanism is generally used to achieve this expectation. Under this mechanism, data updates are written to empty flash memory space and obsolete data are marked invalidated as garbage that a software *cleaner* later reclaims.

Cleaning policies that determine the behavior of the cleaner such as which erasure units to clean, when to clean them, and how to clean them, severely affect how flash memory is worn and the amount of erasure. Thus, cleaning policies are key to flash memory management.

Two major concerns regarding cleaning policies are the *segment selection algorithm* that determines the erasure units to be cleaned and the *data reorganization method* that determines how to migrate valid data in the selected erasure units. The data reorganization method has the most important impact. Therefore, the challenge is how to effectively reorganize data, such that hot data are separated from cold data while remaining low processing overhead. Clustering hot data together in the same erasure units for a better cleaning effectiveness has been shown in previous literature.

3. Multimedia File System for flash memory

Multimedia applications usually require data to be stored and retrieved at a certain rate. For example, video data compressed using MPEG-1 requires a transfer rate of about 1.5 Mbps. Thus, multimedia file system supports real-time storage and retrieval of multimedia data. Moreover, several multimedia applications can exist concurrently, sharing the storage device. However, the I/O bandwidth of storage device is limited. To ensure that an application's real-time request is handled properly, an admission control scheme must be devised to restrict the number of concurrent real-time requests being serviced at any given time. The admission controller decides on whether or not to accept a new request. Once a request is admitted, the corresponding application is serviced with a guarantee that it will be able to transfer the media data continuously at the requested rate.

Most multimedia files tend to be voluminous. For example, a 100 minute MPEG-1 compressed video requires more than a giga bytes of storage space. The file system should handle properly the deleted files. The blocks that belong to deleted files become garbage and are cleaned for reuse. Because of slow write/erase speed of flash memory and large number of blocks to be cleaned, it could affect the real-time data retrieval. Therefore, file system must consider efficient garbage collection procedure for real-time data retrieval.

4. The Semantics of Multimedia File Retrieval

We assume that clients access multimedia files in sessions. The client might be a user-level program that uses system calls to access files. A session is a contract between file system and its client. That is, a session guarantees that its behavior will obey certain rules. When requesting a session establishment, clients make a system call with parameters such as session type (read or write), buffer and data rate (bytes per second). If file system cannot be accepted the session because of some reasons such as lack of the buffer space or large requested data rate, an error code is returned. Otherwise, a session is established. In case of the read session, we assume that the client can read data in advance by requesting the amount of workahead at session setup time in order to improve the system performance.

```
open_session(
   int type,              /* read or write */
   char* filename,        /* file name */
   void* buffer,          /* data buffer */
   int workahead,         /* amount of workahead (≥0) */
   int rate)              /* data rate (bytes per second) */
```

For example, in applications such as playback of digital audio, I/O begins on session acceptance and proceeds at a client's requested data rate. Data is transferred to the client via buffer. Once a session has been accepted, file system will write data sequentially into the buffer, and client will remove data sequentially.

Definition : A multimedia file F has parameters R and S, where R is maximum data rate and S is block size, respectively. A file F is defined as a *limited-rate* file if the ith byte of F has a timestamp $T(i) \geq 0$ such that $T(i) \leq T(j)$ for $i < j$, and

$$i - j \leq R \cdot (T(i) - T(j)) + S$$

for all $i > j$.

The timestamps may be explicit or implicit. This Definition means that the amount of data in a time interval of length of T is at most $R \cdot T + S$. See figure 2. This example represents a file of 30 frames/second video data with a varying number of bytes per frame. Each frame is a vertical segment of the stairstep. The number of bytes with timestamps in an interval of length T cannot exceed $R \cdot T + S$.

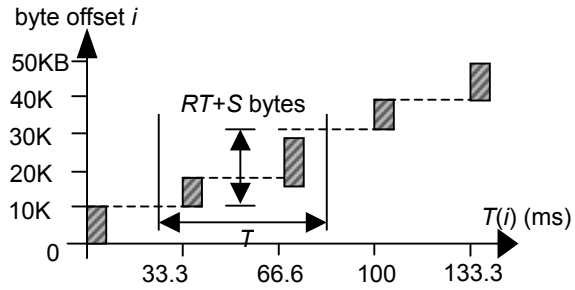

Figure 2: Example of a limited-rate file

For describing the semantics of sessions, we define some notations. Let R_i be the request rate (bytes per second) at which the file is to be read or written of a session i. Let W_i be the workahead parameter of a session i. Let B_i be the size of the buffer of a session i. Let bt_i be the start time of a session i. At time t, $G_i(t)$ is defined as the offset of the next byte to be removed from the buffer by the client (read session) or file system (write session) in session i. We also define $P_i(t)$ at time t as the offset of the next byte to be put into the buffer by client (write session) or file system (read session) in session i. Let $L_i(t)$ be a logical clock defined for $t \geq bt_i$. Then the following conditions must hold:

$$L_i(bt_i) = 0$$

$$\frac{dL_i(t)}{dt} = R_i \quad if \; L_i(t) \prec G_i(t)$$

$$\frac{dL_i(t)}{dt} = 0 \quad if \; L_i(t) = G_i(t)$$

In other words, the logical clock advances at rate R_i whenever it trails $G_i(t)$, and stops otherwise.

For a read session, the following conditions must hold for all $t \geq bt_i$:

$$P_i(t) - G_i(t) \leq B_i$$

$$P_i(t) - L_i(t) \geq W_i$$

$$G_i(t) \leq P_i(t)$$

These conditions mean that file system does not overflow the buffer, file system allows the client to read ahead of the logical clock by up to W bytes, and the client does not read beyond the write point. Therefore, file system can provide a guaranteed data rate.

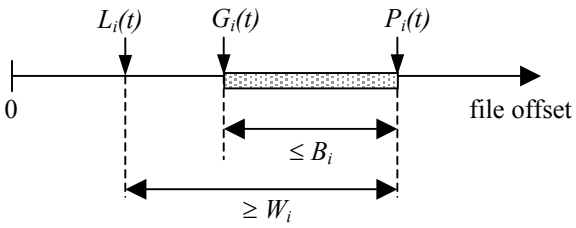

Figure 3: Semantic of a read session. (The shaded rectangle represents data in the buffer)

5. Scheduling the retrieval sessions

File system accepts a new session only if its data rate requirements, together with those of existing sessions, can be guaranteed. Suppose that sessions S_1, ... S_n read files F_1, ... F_n at rates R_1, ... R_n. File system uses a round robin scheduler, in which sessions are serviced in cycles. Let C be the length of the cycle.

Let S_p and T_p be the size of a page and time to read a page from flash memory, respectively. Then the maximum number of pages that can be retrieved for session S_i is

$$\left\lceil \frac{C \cdot R_i}{S_p} \right\rceil + 1$$

Thus, the maximum latency time for servicing sessions S_1, ... S_n is

$$L(n) = \sum_{i=1}^{n} \left(\left\lceil C \cdot R_i / S_p \right\rceil + 1 \right) \cdot T_p$$

Let ρ be a fraction of each cycle is used for servicing real-time data retrieval sessions. The remainder of the cycle, (1-ρ)C, is reserved for background tasks such as garbage collection.

Then a new session is admitted if the following condition is satisfied

$$L(n) \leq \rho \cdot C .$$

and there is at least $2CR_i$ bits of the bufer space available.

6. Conclusion

Recently, flash memory is widely used as storage system to store and retrieve multimedia files. However, there are few researches about multimedia file system for flash memory based storage devices. In this paper, we describe the semantics of real-time retrieval of multimedia data and present scheduling scheme to guarantee the real-time requirements. We are currently designing and developing a novel multimedia file system for linux. For future work, we are planning to study performance issues in our implemented file system.

References

[1] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer, "Non-Volatile Memory for Fast, Reliable File Systems," *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1992.

[2] M. L. Chiang and R. C. Chang, “Cleaning Policies in Mobile Computers Using Flash Memory,” *Journal of Systems and Software*, Vol. 48, No.3, pp. 213-231, Nov. 1999.

[3] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, "Using Data Clustering to Improve Cleaning Performance for Flash Memory," *Software Practice & Experience*, Vol. 29, No.3, pp. 267-290, Mar. 1999.

[4] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber, "Storage Alternatives for Mobile Computers," *Proceedings of the 1st Symposium on Operating Systems Design and Implementation*, 1994.

[5] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," *Proceedings of the 1995 USENIX Technical Conference*, Jan. 1995.

[6] B. Marsh, F. Douglis, and P. Krishnan, "Flash Memory File Caching for Mobile Computers," *Proceedings of the 27 Hawaii International Conference on System Sciences*, 1994.

[7] M. Rosenblum and J. K. Ousterhout, “The Design and Implementation of a Log-Structured File System,” *ACM Transactions on Computer Systems*, Vol. 10, No. 1, 1992.

[8] P. Torelli, "The Microsoft Flash File System," *Dr. Dobb's Journal*, Feb. 1995.

[9] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994.

[10] B. Ozden, R. Rastogi, and A. Silberschatz, A Framework for the Storage and Retrieval of Continuous Media Data, *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, May. 1995.

[11] D. J. Gemmel, H. M. Vin, D. D. Kandler, P. V. Rangan, and L. A. Rowe, "Multimedia Storage Servers : A Tutorial," *IEEE Computer*, pp. 40-49, May. 1995.

[12] Intel Corporation, "Understanding the Flash Translation Layer Specification," http://developer.intel.com

[13] T. Chung, D. Park, Y. Ryu and S. Hong, “LSTAFF: System Software for Large Block Flash Memory,” *Lecture Notes in Computer Science*, Vol. 3398, Feb. 2005.

[14] L. Chang and T. Kuo, “An Efficient Management Scheme for Large-Scale Flash Memory Storage Systems,” *Proc. of ACM Symposium on Applied Computing*, 2004.