임베디드 멀티미디어 시스템을 위한 파일 시스템의 설계 및 구현

이 민 석*

A File System for Embedded Multimedia Systems

Minsuk Lee*

Abstract

Nowadays, we have many embedded systems which store and process multimedia data. For multimedia systems using hard disks as storage media such as DVR, existing file systems are not the right choice to store multimedia data in terms of cost, performance and reliability.

In this study we designed a reliable file system with very high performance for embedded multimedia applications. The proposed file system runs with quite simple disk layout to reduce time to initialize and to recover after power failures, uses a large data block to speed up the sequential accesses, incorporates a time-based indexing scheme to improve the time-based random accesses and boosts reliability by backing up the important meta data on a small NVRAM. We implemented the file system on a Linux-based DVR and verified the performance by comparing with existing file systems.

Keywords : File System, Embedded System, Multimedia, Linux

논문접수일 : 2004년 11월 1일 논문게재확정일 : 2005년 1월 10일

※ 이 연구는 2003년도 한성대학교 교내 연구비 지원 과제임.

* 한성대학교 컴퓨터공학과

1. 서 론

오늘날 IT 발전은 초고속 네트워크를 기반으로 하는 멀티미디어 서비스와 멀티미디어 가전을 중심으로 이루어지고 있다. 이에 따라 VOD (Video On Demand) 서비스를 위한 멀티미디어 서버나 멀티미디어 데이터를 적극적으로 저장하고 재생하는 DVR(Digital Video Recorder), PVR(Personal Video Recorder), 홈 미디어 서버 등이 이미 시장에서 중요한 위치를 차지하고 있으며, 캠코더와 같은 휴대용 영상 기기도 자기 테이프 대신 하드 디스크나 대용량 플래쉬 메모리와 같은 임의 접근이 가능한 저장 장치를 점차 많이 사용하는 추세이다. 이는 이러한 장치들에 대용량의 멀티미디어 정보를 신뢰성 있으면서도 높은 성능으로 저장하고 재생할 수 있는 파일 시스템이 필요하다는 것을 의미한다.

많은 멀티미디어 임베디드 기기들은 다양한 운영체제를 사용하고 있지만 시스템 제조업체 입장에서는 하드웨어의 특성과 응용 프로그램의 기능에 따라 pSOS, VxWorks와 같은 실시간 운영체제 또는 임베디드 리눅스, 윈도우 CE와 같은 고기능 운영체제를 사용하게 되며, 선택한 운영체제가 제공하는 파일 시스템들로 파일 시스템 선택이 제한되는 상황을 겪게 된다. 그러나 기존의 운영체제가 제공하는 파일 시스템들은 여러 사용자와 엄청나게 많은, 그리고 작은 파일들을 유지하는 대형 서버를 고려하여 설계되어 있거나, 불과 수십에서 수백 메가 바이트의 저장 공간만을 가지고 있는 시스템들에 최적화되어 있어서 대량의 멀티미디어 데이터를 효율적으로 저장하고 재생하는 응용에는 적합하지 않은 경우가 많다. 또 많은 임베디드 시스템들은 불시에 전원이 차단될 가능성이 높은 환경이며, 메인 메모리 용량이나 CPU의 성능과 같은 시스템 자원에 대한 여러 제약도 가지고 있는데, 이런 부분들이 기존 파일 시스템의 설계에서는 잘 고려되지 않았었다. 또 서버에 사용될 것을 목적으로 만들어진 파일 시스템들은 소프트웨어적으로 매우 복잡하고 소스 및 바이너리 크기가 커서, 하드웨어 원가와, 소프트웨어의 보수 및 유지에 소요되는 비용의 상승을 가져와 임베디드 시스템 환경에 적용하기 어려웠다.

본 연구에서는 멀티미디어 파일을 저장하고 재생할 때 높은 성능을 보이며, 동시에 불시의 전원 차단에 대한 안정성도 높은 새로운 멀티미디어 파일 시스템을 설계, 구현하고, 다양한 파일 시스템 참조 패턴에 대한 성능 평가를 실시하여 구현된 파일 시스템의 성능을 검증하였다. 개발된 파일 시스템은 리눅스 환경에서 구현되어 시험 및 성능 평가 되었으며, pSOS와 같은 실시간 운영 체제에도 이식되어, 현재 DVR과 같은 실제 응용에 성공적으로 적용되고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 연구 배경으로 기존 파일 시스템에 대한 고찰을 하고, 대형 멀티미디어 파일이 저장되는 시스템에서 요구 사항과 기존 파일 시스템들의 문제점을 언급한다. 3장에서는 설계된 파일 시스템의 구조 및 구현 내용을 기술하며, 개발된 파일 시스템의 모체가 된 이전 버전의 파일 시스템과의 설계 차이점과 개선점을 설명한다. 4장에서는 설계된 파일 시스템의 성능을 기존 리눅스 파일 시스템 및 이전 버전의 파일 시스템의 성능과 비교, 평가하기 위한 환경을 기술하고 성능 평가 실험 결과를 분석한다. 끝으로 5장에서는 논문의 결론을 맺는다.

2. 연구 배경

2.1 파일 시스템 관련 연구

파일 시스템은 운영체제의 일부로서 컴퓨터의 하드 디스크에 데이터를 저장하고, 읽어내는

논리적인 수단이다. 즉, 파일의 자료 구조, 데이터가 디스크에 저장되는 방식, 파일을 읽고, 쓰는 연산 등 파일에 관련된 모든 것을 관리한다[1, 2]. 본 절에서는 최근 임베디드 시스템에서 널리 사용되고 있는 리눅스 운영체제를 중심으로 소스가 공개된 파일 시스템들의 특징을 살펴본다.

1991년 리눅스가 탄생한 이래 리눅스 파일 시스템도 지난 십여 년 동안 지속적인 발전을 거듭해 왔다. 그 중 오늘날 가장 널리 사용되는 파일 시스템은 리눅스의 표준 파일 시스템인 EXT2와 저널링 파일 시스템들인 EXT3, JFS, XFS, ReiserFS 등이 있다.

EXT2(The Second Extended File System)는 Way Davidson에 의해 설계된 리눅스 표준 파일 시스템으로 최근까지 리눅스 시스템에서 주로 사용되었으며, 일반 파일, 디렉터리, 디바이스 파일, 심볼릭 링크의 표준 유닉스 파일 타입을 지원한다. 또한 4TB까지의 대용량 파티션 지원이 가능하며 255 문자까지의 긴 파일 이름을 지정할 수 있고 필요에 따라 1012 문자까지 확장이 가능하다[1, 3]. EXT3(The Third Extended File System)는 Stephen Tweedie가 설계한 것으로 EXT2에 저널링을 도입한 EXT2의 확장 파일 시스템이다[4]. 따라서 EXT3는 EXT2와 똑같은 디스크 포맷과 메타 데이터를 가지고 있으며 메타 데이터 및 데이터 저널링을 지원하고 순차적인 파일 이름 찾기가 가능한 블록 기반 파일 시스템이다[5, 6]. 현재 EXT3는 리눅스의 표준 파일 시스템으로서 서버, 워크스테이션에서 널리 사용되고 있다.

ReiserFS는 Namesys의 Hans Reiser와 그가 이끄는 팀이 개발한 파일 시스템으로 작은 파일에 대한 참조 성능 향상에 초점을 맞추어 설계되었다. ReiserFS는 메타 데이터 저널링을 하며 디렉터리, 파일, 데이터를 구성하는 방법으로 B* 트리를 사용한다[6]. JFS(Journaled File System)는 IBM 사에서 만들어진 파일 시스템으로 현재 IBM의 엔터프라이즈 서버에서 사용되고 있다. JFS는 서버 환경에서의 높은 처리량을 위해 설계된 파일 시스템으로서 메타 데이터 저널링을 지원하며 메타 데이터를 B+ 트리로 관리한다. 또한, 동적 아이노드 할당 방법을 사용하여 사용하지 않는 아이노드는 해제하여 디스크 저장 공간을 절약할 수 있다[6-8]. XFS는 SGI 사에 의해 개발되었으며 IRIX 운영체제에 사용되고 있다[9]. 대용량 파일을 저장하는 서버를 위해 설계된 XFS는 블록 크기를 512B에서 64KB까지 지정할 수 있으며 메타 데이터 저널링을 지원한다. 또 메타 데이터 관리를 위해 B* 트리를 사용하며 완전한 64 비트 파일 시스템으로 수백만 테라 바이트의 용량을 지원한다[1, 6, 10].

기존의 파일 시스템들은 주로 서버 환경에서 최적화가 되도록 파일 시스템이 설계 및 구현되어 있으며, 파일 시스템과 주변 유틸리티 코드를 포함하여 수 만에서 수 십만 라인의 소스 코드로 구현된 매우 복잡한 소프트웨어들이다. 따라서 임베디드 시스템에서는 적절한 보수 유지가 어려울 뿐만 아니라, 실행 바이너리 저장을 위한 플래쉬 메모리 공간을 많이 차지함으로써 제품의 원가 상승을 가져온다. 또한 기존 파일 시스템들은 엄청나게 많은 파일이 저장되는 서버 환경을 고려하여, 파일 및 디렉터리를 검색하기 위한 복잡한 알고리즘을 수행하며, 서버가 가지고 있는 대용량의 메인 메모리 상에 유지되는 버퍼 캐쉬에 그 성능을 의존하고 있어 메인 메모리의 용량이 상대적으로 작은 임베디드 시스템 환경에서의 성능은 그리 좋지 않다. 동시에, 크기가 작은 많은 파일들이 만들 디스크 공간 조각화를 줄이기 위해서 최대 수십 킬로 바이트의 단위 데이터 블록을 사용하고 있기 때문

에 디스크에서 데이터 블록 배치의 연속성이 줄어들고, 데이터 블록의 위치를 찾기 위한 비용도 증가하게 된다.

현재 리눅스에서 가장 많이 사용되고 있는 파일 시스템들이 대형 멀티미디어 파일을 지원하는데 있어서의 구조적인 문제점은 디스크 상의 데이터 배치로 인한 문제, 데이터의 트리 구조 관리로 인한 문제, 버퍼 캐쉬 사용으로 인한 문제 등의 세 가지로 요약할 수 있다.

멀티미디어 데이터를 효과적으로 읽고 쓰기 위해서는 디스크 헤드의 이동 시간(seek-time)을 최소화하는 것이 가장 중요하다. 디스크 헤드의 이동 시간은 데이터 블록의 배치 방식과 메타 데이터의 구조에 따라 결정되며 이는 파일 시스템이 설계될 때 정의된다. 데이터 배치 방법과 데이터 블록의 접근 방법, 트리 구조의 디렉터리, 파일 계층 구조로 인한 성능에 관하여는 많은 연구가 이루어져 있다[1, 11].

멀티미디어 데이터를 위한 기존 파일 시스템의 또 다른 중요한 문제는 버퍼 캐쉬의 사용으로 인해 발생한다. 버퍼 캐쉬는 데이터의 재사용 가능성을 고려하여 메인 메모리의 일부를 디스크 블록의 캐쉬로 사용하는 정책이다. 이는 여러 프로그램이 같은 파일을 공유하거나, 같은 파일이 응용 프로그램에 의해 자주 재사용되는 경우, 시스템의 성능을 향상시키는데 크게 도움을 준다. 그러나 단일 프로그램이 순차적으로 데이터를 엑세스함으로써 디스크 블록의 재사용이 잘 보장되지 않는 멀티미디어 시스템의 경우 오히려 메모리를 낭비하고 잦은 스왑핑으로 인하여 시스템 성능을 저하시키는 요인이 되기도 한다. 또한, 서버와 달리 불시에 전원이 차단되는 사례가 빈번하게 발생하는 가전 형태의 멀티미디어 임베디드 시스템의 경우 버퍼 캐쉬에 남아 미처 디스크에 기록되지 못한 많은 데이터가 한 순간에 사라져 파일 시스템 데이터의 무결성을 보장하지 못하는 원인이 된다. 심각한 경우 파일 시스템 전체가 망가져 복구가 안되거나, 복구에 오랜 시간이 걸리게 된다. 로그 파일 시스템[12] 기반인 ReiserFS, XFS, JFS와 같은 저널링 파일 시스템을 사용하면 복구 속도와 안정성, 그리고 XFS의 경우 순차적 참조 성능은 어느 정도 올라가지만 저널링을 위한 메타 데이터의 주기적 기록 때문에 시스템 성능이 전반적으로 떨어진다. 실제로 권우일 등이 행한 연구[7]에서 DVR과 같은 멀티미디어 환경에서 성능 실험을 수행한 결과, 저널링 파일 시스템들은 EXT2 파일 시스템보다 전반적으로 낮은 읽기 성능을 보였다.

위와 같은 문제는 기존 리눅스 파일 시스템이 대용량, 순차적 쓰기/읽기, 읽기 연산 중심이라는 멀티미디어 파일의 특성, 기능이 제한된 응용 프로그램, 불시 전원 차단 가능성이라는 개인용 시스템의 특징들을 충분히 고려하지 못한 채 설계되었기 때문이며 이는 기존 리눅스 파일 시스템이 대형 멀티미디어 파일을 주로 사용하는 개인용 멀티미디어 시스템에 적합하지 않음을 의미한다.

본 연구 결과 설계된 파일 시스템에서는 파일 시스템의 안정성 확보를 위하여 수 킬로 바이트의 적은 용량을 가진 NVRAM(Non-Volatile RAM)을 사용한다. 순수하게 파일 시스템의 안정성을 향상시키기 위한 방법으로 리눅스에서 NVRAM을 사용하는 실용적인 파일 시스템은 없으며, NFS(Network File System)에서 로그를 저장하기 위하여 NVRAM을 사용한 연구[13], 성능 향상을 위하여 NVRAM의 일종인 MRAM(Magnetic RAM)을 사용한 연구[14] 등이 있다. 하지만 양쪽 모두 파일 시스템이 서버 환경에서 사용될 것을 가정한 연구로서 메가 바이트 단위의 큰 NVRAM을 사용하도록 설계되어 제품 가격이 문제가 되는 소형 임베디드 시

스템에서는 적용하기 어렵다. 하드 디스크가 없는 임베디드 시스템에서는 파일 저장을 위해 NAND 또는 NOR 형태의 플래쉬 메모리를 이용하는 것이 보통인데, 리눅스에서는 JFFS2 (Journalling Flash File System 2)[15] 등이 개발되어 있고, 마이크로소프트도 자신의 운영체제를 위하여 이전부터 플래쉬 파일 시스템을 제시하고 있다[16].

2.2 멀티미디어 환경에서 새로운 파일 시스템의 필요성

멀티미디어 파일이라 함은 숫자나 문자 위주의 데이터를 벗어나 텍스트, 그래픽, 이미지, 오디오, 비디오, 애니메이션 등 여러 미디어 데이터 정보가 디지털 방식으로 융합된 형태의 파일을 말한다. 멀티미디어 파일은 다음과 같은 특징을 갖는다.

- 멀티미디어 파일은 주로 음악, 영상 등의 정보를 저장하므로 한 파일이 적어도 수 메가 바이트에서 수십 기가 바이트까지 매우 크다. 또한 DVR과 같은 멀티미디어 기기에서는 파일이라는 개념보다는 몇 개의 채널 단위의 스트림 데이터가 전체 디스크를 사용하는 경우도 있다.
- 파일에 대한 읽기, 쓰기 연산이 주로 순차적으로 일어난다. 대개의 멀티미디어 파일은 비디오, 오디오, 데이터 등 여러 가지 형태의 데이터가 섞여 있지만 이 파일에 대한 연산은 재생, 빠른 재생, 저장 등 주로 각 미디어 데이터에 대한 순차적인 읽기, 쓰기의 형태이다.
- 읽기 참조 중심이다. 대표적인 멀티미디어 파일인 영화, 애니메이션, 음악 파일들은 많은 경우 한번 디스크에 기록된 후, 여러 번 읽혀 재생되는 형태로 참조된다.
- 시간 동기성을 갖는다. 오디오와 비디오, 애니메이션 데이터 등 멀티미디어 데이터는 개별적으로 또는 여러 미디어 데이터가 연관된 복합적인 형태로 시간적인 순서에 따라 저장 및 재생되어야하는 동기화 특성이 존재한다. 이는 응용에 따라 다르지만 DVR, PVR 등과 같이 VBR(Variable Bit Rate) 방식으로 인코딩된 멀티미디어 데이터를 기록하는 시스템에서는 파일에서의 바이트 위치가 아닌 시간을 기준으로 한 인덱싱이 더 유용하다는 것을 의미한다.

위의 특성들은 모두 파일 시스템의 구조에 따라 파일 시스템의 성능에 큰 영향을 끼칠 수 있는 것들이며, 이 논문에서는 이러한 특성을 고려하여 파일 시스템을 설계하고 구현하고자 한다. 이 연구는 멀티미디어 파일을 저장하기 위해 설계된 이전 버전의 파일 시스템[17]을 바탕으로 성능과 신뢰성을 크게 개선하는 차원에서 이루어졌다.

3. 멀티미디어 시스템을 위한 파일 시스템의 구현

이 장에서는 새로 설계 및 구현된 파일 시스템의 구조와 특성을 기술하며, 비슷한 요구 사항을 만족하기 위해 저자가 속한 연구 팀에서 이전에 설계했던 파일 시스템과 구조적인 차이에 대한 비교를 수행한다.

3.1 개인용 멀티미디어 파일 시스템의 요구사항

이 절에서는 멀티미디어 시스템을 위한 파일 시스템이 갖추어야 할 중요한 요구 사항을 성능

과 안정성을 기준으로 살펴본다.

우선 성능 기준으로는 대형 멀티미디어 파일에 대한 순차 참조에 최적화되어야 하며, 파일의 절대적 위치보다는 시간 기준의 탐색 성능이 좋아야 하며, 쓰기보다는 재생이라는 형태로 사용자와 교감하는 읽기 부분에서의 성능이 더 중요성을 가진다. 또 시스템 관리 측면에서는 디스크에 파일 시스템을 초기화하는 시간, 불시의 전원 차단 후에 수행하는 파일 시스템 검사 시간 등이 중요한 요소이다. 특히 새 디스크를 초기화하고 파일 시스템을 복구하는 시간은 사용자의 편의성을 고려하여 많은 가전기기 형태의 임베디드 멀티미디어 기기에서 수 초 이내에 수행되어야 하며, 이는 단순한 디스크 데이터 배치 구조를 통해서만 달성될 수 있다.

안정성 기준이란 불시의 전원 차단에 대한 내성을 의미하는 것으로 먼저 불시의 전원 차단 후에도 파일 시스템이 정상적으로 복구되어 사용될 가능성을 높이는 것과, 복구된 경우에도 데이터의 손실을 최소화함으로써 얻어진다. 기존의 파일 시스템은 전원에 대한 불시 차단 반복 실험에서 저널링을 하지 않는 경우 파일 시스템 복구가 불가능한 현상을 자주 관찰할 수 있으며, 저널링을 하는 경우에도 복구가 불가능해지는 현상이 드물지만 발생하고 있다. 또 파일 시스템 자체는 복구가 되더라도 상당히 많은 데이터가 손실되는 것이 보통이다. 이러한 복구 가능성 및 복구 수준에 관한 문제는 파일 시스템을 적용한 임베디드 시스템 제품에서 매우 중요한 요소로서 반드시 파일 시스템 설계에 고려되어야 한다.

3.2 설계, 구현된 파일 시스템의 구조

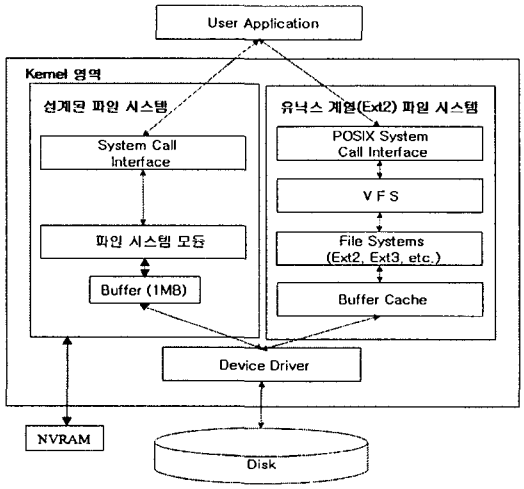
3.2.1 파일 시스템의 계층 구조

이 논문에서 제안된 파일 시스템은 리눅스 운영체제 상에서 우선 구현, 시험되었으며, pSOS와 같은 실시간 운영 체제에도 성공적으로 이식되었다. 설계된 파일 시스템은 DVR, PVR 등 멀티미디어 기기를 고려한 데이터 저장 구조를 갖는다.

운영 체제에서 파일 시스템의 전체 구조는 <그림 1>과 같다. 설계된 파일 시스템은 유닉스 계열 파일 시스템과 달리 자체 시스템 콜 인터페이스를 제공한다. 이는 유닉스 계열 운영체제가 제공하는 VFS(Virtual File System) 환경이 가정하는 디스크 데이터 배치 구조와 설계된 파일 시스템의 데이터 배치 구조가 매우 다르기도 하며, VFS가 관리하는 버퍼 캐쉬를 사용하지 않기 때문이다. 하지만 응용 프로그램을 위하여 VFS 환경에서 제공되는 *open*(), *close* (), *read*(), *write*(), *lseek*() 등 거의 모든 POSIX 인터페이스를 응용 프로그램에 제공함으로써 간단한 매크로 처리로서 소스 프로그램상의 호환성이 상당히 유지된다.

일반 유닉스 계열 파일 시스템은 디스크 블록에 대한 재사용을 고려한 성능 향상을 위해 버퍼 캐쉬를 파일 시스템과 디스크 사이에 두어 디스크를 직접 참조하지 않고 버퍼 캐쉬를 통하여 참조한다. 대부분의 운영 체제에서는 시스템 내의 모든 비할당 메모리를 버퍼 캐쉬로 적극 활용하여 디스크 블록의 재사용, 쓰기의 지연 효과 등을 추구함으로써 성능 향상을 꾀하고 있지만, 순차 참조 특성을 가지는 멀티미디어 데이터 환경에서는 그다지 성능 개선 효과가 없으며, 쓰기 버퍼의 사용은 불시 전원 차단 때 데이터 무결성을 보장하지 못하는 중요한 원인이 된다. 따라서 설계된 파일 시스템에서는 기존 운영체제와는 달리 사용자 공간에 있는 응용 프로그램과의 인터페이스를 위한 최소한의 버퍼만을 유지하며, 가능한 모든 데이터를 하드 디스크에 바로 기록하고, 읽도록 하고 있다.

<그림 1>에서 NVRAM은 불시에 전원이 차단된 후나 그 밖의 다른 이유로 파일 시스템의 복구가 필요할 때, 주요한 메타 데이터를 저장하기 위한 공간이다.

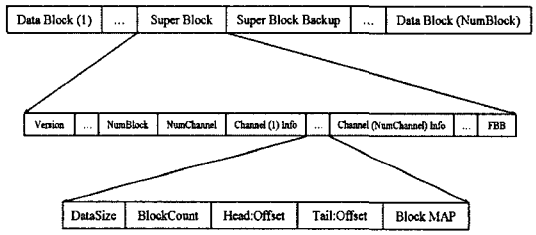


<그림 1> 파일 시스템의 운영 체제 상의 위치

3.2.2 디스크 상의 데이터 배치

설계된 파일 시스템은 순차적 쓰기/읽기 특징의 멀티미디어 특성을 반영하여 <그림 2>와 같은 단순한 데이터 배치 구조를 갖는다. DVR, PVR 등 각 스트림의 채널별 데이터를 저장하는 시스템을 고려하여 설계된 이 파일 시스템에서 디렉터리 구조는 없으며, 기존 파일 시스템의 파일에 해당하는 채널이 정의되어 있으며 데이터도 당연히 채널 별로 저장된다.

설계된 파일 시스템은 메타 데이터(슈퍼 블록)로 디스크 크기 및 파일 시스템 구성 정보, 각 채널의 기본 정보 및 데이터 블록 할당 정보, 데이터 블록의 사용 여부를 기록한 비트맵 정보 등을 가지고 있으며, 이 모든 메타 데이터는 파일 시스템 복구를 위해 하드 디스크의 슈퍼 블록 백업 영역에 주기적으로 백업된다. 메타 데이터는 하드 디스크의 논리적 주소 (Logical Block Address)를 기준으로 중간에 위치하며, 그 앞뒤로 각 데이터 블록이 위치한다. 각 단위 데이터 블록의 크기는 최소 1MB에서 최대 256MB까지로, 파일 시스템 초기화 때 지정할 수 있다. 한 데이터 블록의 크기를 이와 같이 크게 유지하면 내부 단편화 문제가 발생하지만, 이 연구가 목표 시스템으로 정한 DVR, PVR은 수십에서 수백 기가 바이트의 하드 디스크 공간에 몇 개 채널의 멀티미디어 데이터가 연속적으로 저장되는 환경이기 때문에 채널당 수 메가 바이트의 단편화는 그리 문제가 되지 않는다.



<그림 2> 파일 시스템의 디스크 데이터 배치

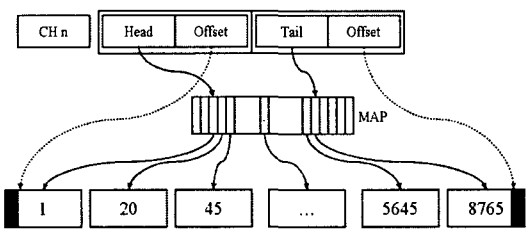
<그림 2>의 디스크 데이터 배치 구조는 매우 단순하기 때문에 최대 채널 수와 단위 데이터 블록의 크기만을 지정하여 초기화 되는데, EXT2 등 기존 파일 시스템을 초기화할 때와는 달리 불과 1 초 이내에 초기화가 완료되며, 불시의 전원 차단 후에도 대부분의 경우 수 초 이내에 파일 시스템 검사와 복구가 가능하다.

설계된 파일 시스템의 메타 데이터 구조는 <표 1>과 같다. 메타 데이터 가운데 *BlockSize*와 *NumChannel* 값은 파일 시스템을 하드 디스크에 초기화할 때 사용자가 지정하는 값이며, *FirstIndexTime*, *LastIndexTime*은 이 파일 시스템이 DVR에서 사용될 때 채널별, 시간별 인덱스를 유지하기 위한 사용하는 인덱스 채널에 대한 시간 정보이다. 각 채널에는 전체 디스크 크기에 해당되는 데이터 블록 할당 맵이 있고 <그림 3>과 같이 그 맵의 시작과 끝 위치가 저

장되어 임의의 데이터 위치를 찾을 수 있도록 되어있다. 각 채널의 데이터 시작 위치는 블록 할당 맵의 시작 엔트리 번호(*Head*)와 그 블록 안에서의 시작 위치(*HeadOffset*)로 지정되며, 끝 위치는 블록 할당 맵의 마지막 엔트리 번호(*Tail*)와 그 블록 안에서의 끝 위치(*TailOffset*)로 지정된다. 이는 시작 위치가 바뀔 것을 가정하는 것으로 다음 절의 인덱싱 방법에서 기술된다.

〈표 1〉 주요 메타 데이터

BlockSize		단위 데이터 블록 크기
NumBlock		전체 데이터 블록 수
NumFreeBlock		빈 데이터 블록 수
NumChannel		데이터 채널 수
채널별	*DataSize*	총 바이트 수
	BlockCount	데이터 블록 수
	Head	시작 위치
	HeadOffset	맵에서의 시작 블록의 오프셋
	Tail	마지막 위치
	TailOffset	맵에서의 마지막 블록의 오프셋
	Map	데이터 블록 할당 맵
FirstIndexTime		데이터가 최초로 기록된 시간
LastIndexTime		마지막으로 데이터가 기록된 시간
FBB		Free 데이터 블록 비트맵

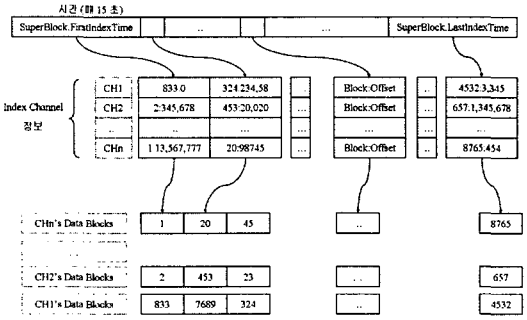


〈그림 3〉 각 채널의 데이터 블록 지정 방법

3.2.3 데이터에 대한 시간 기준 인덱스 구조

설계된 파일 시스템에는 카메라에 의해 촬영된 동영상과 음성, 사용자에 의해 지정되는 정지 화상, 로그 등을 기록하는 DVR에 적용하기 위하여 파일 이름이나 절대적인 위치에 의한 방법이 아니라, 각 채널에서 시간을 기준으로 데이터의 위치를 찾는 기법이 적용되었다. 이를 위하여 스트림 데이터가 아닌 인덱스 정보만을 저장하는 인덱스 채널을 따로 정의하였으며, 이 인덱스 채널에는 매 15 초마다 각 채널의 데이터가 어떤 데이터 블록의 어느 위치에 있는지를 모든 채널에 대하여 기록한다. 이 인덱스 정보를 이용하여 DVR에서는 사용자가 지정하는 임의의 시간에 해당하는 비디오, 오디오 정보에 바로 접근할 수 있다. <그림 4>는 파일 시스템이 DVR 응용을 위해 유지하는 인덱싱 기법을 그림으로 나타낸 것이다.

또 파일 시스템에는 채널 별로 데이터가 연속적으로 저장되며, 데이터가 점점 차서 하드 디스크에 빈 공간의 양이 일정 수준 이하로 감소하면 각 채널 데이터의 앞 부분, 즉 가장 오래된 데이터를 삭제(보통은 한 시간 분량의 데이터를 삭제)하여 빈 데이터 블록을 확보한 뒤 기록하는 리사이클 기능을 구현하였다.



〈그림 4〉 DVR을 위한 인덱싱 기법

위의 인덱싱 방법에 따라 설계된 파일 시스템은 데이터를 읽을 때, 순차 접근과 바이트 간격에 의한 임의 접근, 시간 정보에 따른 임의 위치 접근이 모두 가능하다. 순차 접근은 슈퍼 블록에 유지되는 채널 정보에 따라 각 채널의 첫 데이터 블록의 시작 위치로부터 *read*() 동작을 수행함으로써 가능하고, 바이트 수를 인자로 하

는 *lseek*()를 통하여 임의 접근도 가능하며, 또한 인덱스 채널을 참조함으로써 시간에 의거한 임의 접근도 가능하다.

설계된 파일 시스템은 POSIX 스타일의 API에 추가하여, 시간 기준 인덱싱 기법을 지원하기 위하여 <표 2>와 같은 특별한 API들을 구현하였다.

〈표 2〉 시간 기준 인덱스와 관련된 API들

API	기 능
dvr_locate_index()	시간을 입력으로 채널에서의 데이터 위치 찾기
dvr_iseek()	위치 정보를 이용하여 데이터 스트림에 임의 접근하기
dvr_tell()	채널의 현재 읽기 또는 쓰기 위치 (블록 번호, Offset) 알아내기
dvr_recycle()	지정된 시간 이전의 데이터를 파일 시스템에서 삭제

파일 시스템에 데이터를 쓰는 경우에는 오직 순차 접근만 가능하며 쓰기가 진행되는 동안 인덱스 채널에 매 15초마다 각 채널의 쓰기 위치가 파일 시스템에 의해 자동으로 기록되어 시간 인덱스가 유지된다.

3.2.4 데이터 무결성의 보장

설계된 파일 시스템에서는 보통의 운영체제에서 사용하는 버퍼 캐쉬를 쓰지 않기 때문에, 데이터를 디스크에 바로 기록함으로써 불시의 전원 차단에도 높은 데이터 안정성을 보인다. 가전용으로 만들어지는 많은 멀티미디어 시스템은 사용자가 불시에 전원을 끄는 경우를 빈번하게 당하게 될 가능성이 높으며, 보안을 위해 사용하는 DVR의 경우 침입자가 의도적으로 전원을 차단할 수도 있다. 기존의 파일 시스템처럼 버퍼 캐쉬를 적극적으로 사용하는 경우 전원 차단과 함께 시스템의 메인 메모리의 버퍼 캐쉬에 있던 데이터가 미처 디스크에 기록되지 못하여 데이터가 손실되거나, 최악의 경우 파일 시스템 자체가 망가지기도 한다.

설계된 파일 시스템은 데이터를 디스크에 기록할 때 버퍼 캐시를 통하지 않고 바로 디스크에 쓰도록 하여 불시의 전원 차단 등의 경우, 데이터 손실 가능성을 크게 줄이고 있으며, 간단한 메타 데이터 구조 때문에 파일 시스템을 복구하기 위한 파일 시스템 검사 시간이 매우 짧다.

그에 더하여, 이전 파일 시스템의 구현 경험에서, 소프트웨어적으로는 디스크에 데이터를 썼지만 하드 디스크 내부에서 유지되고 있는 버퍼에 남아 있다가 전원이 차단되면 디스크 표면에 기록되지 않고 손실이 되는 경우가 많다는 점이 관찰되었다. 그렇게 손실된 정보가 각 채널의 데이터 블록 링크나 매우 중요한 정보인 시간별 인덱스 정보인 경우에는, 이차적으로 더 많은 데이터를 복구할 수 없거나 인덱싱이 불가능해지는 상태가 되기도 한다. 따라서 매우 중요한 최소한의 정보를 적극적으로 보호할 필요성이 있다.

새로 설계된 파일 시스템에서는 각 채널의 데이터 블록 할당 맵의 시작 부분과 끝 부분의 일부를 포함한 슈퍼 블록 내의 주요 정보, 최근 수분 가량의 인덱스 정보 등을, 내장 배터리로 전원 차단 후에도 데이터가 보존되는 NVRAM에 저장하며, 이 NVRAM에 저장된 정보는 전원이 복구되면 파일 시스템의 복구 때 우선 사용된다.

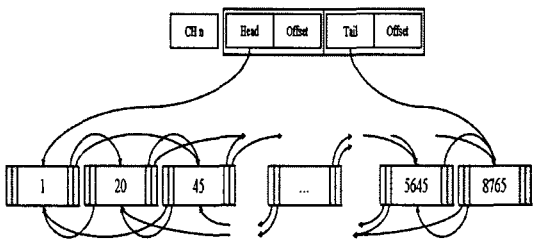
또 불시에 전원이 차단될 때 하드 디스크 내부의 버퍼에 있던 내용이 손실되는 것을 최소화하기 위하여 15초마다 인덱스 정보가 기록될 때, 하드 디스크의 내부 버퍼에 있던 쓰기 데이터들을 디스크 면에 실제 기록하도록 하는 캐쉬 플러쉬 명령을 IDE 드라이버를 통해 하드 디스크에 내리도록 하였다.

3.2.5 미리 읽기

파일 시스템에서는 읽기 속도를 높이고자 효과적인 미리 읽기(read-ahead)를 수행한다. 멀티미디어 시스템은 대부분 읽기 위주의 참조를 수행한다. 본 파일 시스템이 적용되는 DVR에서도 각 채널에 대한 쓰기가 진행되는 동시에 사용자가 디스크에 기록된 데이터를 GUI를 통해 읽고(정속 또는 2배속, 4배속의 속도로), CD-R과 같은 백업 매체로 전송하며, 또 동시에 네트워크를 통해 저장된 정보를 읽기도 한다. 이는 쓰기 성능도 중요하지만, 사용자의 조작에 대하여 실시간 응답을 해야 한다는 관점에서 디스크로부터 데이터를 읽는 성능이 더 중요할 수 있음을 의미한다. 따라서 설계된 파일 시스템에서는 응용 프로그램이 디스크로부터 읽기를 수행하면 매번 32 킬로 바이트의 데이터를 읽을 때마다 버퍼가 허용하는 한도 내에서 단위 버퍼 크기의 네 배까지의 데이터를 미리 읽어 읽기 버퍼에 유지한다. 따라서 일단 읽기를 수행하면 다음의 읽기를 위해 이미 충분한 데이터를 버퍼에 유지하게 되므로 순차 읽기 성능이 획기적으로 개선된다.

3.2.6 이전 버전과의 비교

이 파일 시스템은 DVR 용으로 제작된 것으로 이전에 있던 파일 시스템[17]의 데이터 배치 구조를 개선한 것이다. 새 파일 시스템에서는 이전 파일 시스템에서 한 채널에 속한 각 데이터 블록 리스트를 <그림 5>와 같이 4개의 링크로 유지하던 방식을 버리고, <그림 3>과 같은 데이터 블록 맵 방식으로 바꾸었다. 따라서 매번 한 채널에 데이터 블록이 새로 추가될 때 기존 방식에서는 각 데이터 블록의 시작 위치에 있는 정보 블록을 수정하였지만, 새 버전에서는 수퍼 블록의 채널의 데이터 블록 할당 맵을 수정함으로써 하드 디스크의 헤드가 더 먼 거리를 이동할 수는 있으나, 하드 디스크 내부의 헤드 스케줄 기법의 도움으로 성능 상의 차이를 거의 느낄 수 없었으며, 이전 방식에서 데이터 블록 연결 링크가 중간에서 끊어져 많은 데이터 손실이 일어날 수 있는 가능성을 없앴다. 또 데이터 블록 할당 맵의 오류에 대비하여 NVRAM을 이용하여 신뢰도 보강을 하였다. 이러한 데이터 블록 할당 맵을 사용함으로써 *lseek*()와 같은 순차 진행 참조 속도가 개선되었으며, 이는 결국 인덱스 채널의 참조 시간을 개선하여 임의 시간에 대한 데이터 접근 속도가 혁신적으로 향상된 것을 실험으로 확인할 수 있었다.



〈그림 5〉 이전 버전에서의 채널 데이터 블록 지정 방법

4. 파일 시스템 성능 평가

본 연구에서는 새로 설계된 파일 시스템의 성능을 리눅스에서 널리 사용되는 파일 시스템인 EXT2, 그리고 이 파일 시스템의 모델이 된 이전 버전과 비교하였다. 성능 평가는 파일 시스템에 대한 순차 읽기, 임의 접근 읽기, 순차 쓰기, 읽기와 쓰기 혼합 동작에 대한 응답 시간으로 하였다.

4.1 성능 평가 실험 환경

성능 평가를 실시할 시스템의 환경은 <표 3>과 같다.

〈표 3〉 성능 측정 환경

시스템	CPU	PowerPC (166Mhz)
	메인 메모리	32MB (27MB)
	OS 버전	Linux 2.4.2
HDD (Seagate ST340016A)	IDE 인터페이스	UDMA 100
	디스크 회전수	7200rpm
	평균 탐색 시간	9ms
	용량	40GB

성능 평가 실험에 사용된 환경은 실제 DVR로서 5 채널의 화상, 2 채널의 오디오, 1 채널의 정지 화상 그리고 1 채널의 시스템 로그를 저장하는 데이터 채널들과, 매 15초마다 데이터 위치와 해당 15초 동안 각 데이터 채널에 데이터가 저장되어있는지 유무를 기록하는 인덱스 정보를 3개 채널에 저장하는 시스템이다. 실험에서는 파일 시스템의 성능만을 평가하기 위하여 DVR의 주요 기능에 해당되는 비디오 및 오디오 채널의 인코딩 및 디코딩 기능을 정지시키고 인위적으로 생성된 데이터를 저장하고 읽는 방법으로 시험을 하였다. 시스템의 메인 메모리는 32MB이고 시스템에서 부팅 후 5MB 가량을 램 디스크로 사용하기 때문에 실제 시스템의 가용 메인 메모리는 27MB가 된다. 이 공간 가운데 응용 프로그램과 커널이 사용하고 남은 메모리 영역을 EXT2 파일 시스템에서는 적극적으로 버퍼 캐쉬로 활용하며, 설계된 파일 시스템에서는 <그림 1>에서처럼 그 가운데 오직 1MB을 디스크 참조를 위한 읽기, 쓰기 버퍼(즉, VFS의 버퍼 캐쉬에 해당하는 버퍼)로 활용하고 있다.

파일 시스템의 성능을 측정하기 위해 사용된 부하는 <표 4>와 같다.

실제 파일 시스템 API를 통해 전달되는 단위 데이터는 성능 측정 프로그램의 글로벌 데이터 영역에 존재하는 128KB의 데이터이다. 테스트 프로그램은 쓰레드 수를 1개, 4개, 8개로 유지하고, 단위 데이터 크기로 해당 채널 (EXT2의 경우는 해당 파일)에 순차 또는 임의의 쓰기, 읽기를 1024회 반복 수행하게 된다.

〈표 4〉 성능 시험 용 부하

응용 프로그램 버퍼 크기	128KB
반복 회수	1024회
단일 쓰레드의 읽기/쓰기 크기 (버퍼크기) x (반복 회수)	128MB
쓰레드 수	1개, 4개, 8개

테스트 프로그램은 멀티미디어 파일에 대한 성향을 잘 반영하고 있어야 한다. 개인용 멀티미디어 시스템의 경우 주로 순차적인 쓰기, 읽기 연산이 대부분이기 때문에 테스트 프로그램도 이와 같은 기능을 갖도록 구현되었으며, 또 시간에 의한 임의 접근을 비교하기 위하여 임의 접근 읽기 기능도 구현하였다. EXT2의 경우에는 시간에 의한 임의 접근 기능이 없기 때문에 난수를 발생하여 파일의 임의 위치로 *lseek*()를 한 뒤에 그 위치에서 데이터를 읽는 것으로 실험으로 하였으며, 이전 버전의 파일 시스템과 새로운 파일 시스템에서는 임의의 시간을 난수로 얻어낸 뒤 그 시간에 해당하는 인덱스 채널 정보를 먼저 읽은 다음 *dvr_iseek()* 함수로 데이터 채널을 읽는 방법으로 사용하였다.

성능 측정은 쓰레드 1, 4, 8 개가 모두 읽기 또는 쓰기만을 수행하는 경우와, 읽기와 쓰기 쓰레드 개수가 1 : 3, 1 : 7, 2 : 6, 4 : 4, 7 : 1, 6 : 2, 3 : 1인 경우데 대하여 모든 쓰레드가 작업을 종료하는 총 실행 시간을 *gettimeofday*() 함수를 이용하여 측정하였으며, 동일한 실험을 4회 수행하여 그 평균값을 결과로 채택하였다.

VFS가 제공하는 버퍼 캐쉬 기능을 적극적으로 사용하는 EXT2 파일 시스템의 경우, 매회 실험 때마다 이전 실험의 버퍼 캐쉬의 내용이 사용될 가능성이 있고 이 경우 파일 시스템에 대한 정확한 성능 평가가 이루어질 수 없기 때문에 이에 대한 선행 처리를 해주는 것이 필요

하다. 버퍼 캐쉬의 효과를 없애는 일반적인 방법으로는 시스템에 메인 메모리보다 큰 파일을 쓰는 방법과 파일 참조 때 *O_DIRECT* 모드를 사용하는 방법, 시스템을 재부팅하는 방법 등이 있다[6]. 본 실험에서는 EXT2 파일 시스템의 경우, 매번 실험 시작 전에 시스템을 재부팅하여 버퍼 캐쉬의 효과를 완전히 없앴으며, 설계된 파일 시스템의 경우에는 리눅스의 커널 모듈 내에 버퍼가 유지되므로, 파일 시스템을 구현하고 있는 커널 모듈을 리눅스의 *rmmod, insmod* 명령을 이용하여 모듈 자체를 다시 설치함으로써 모든 버퍼를 초기화하여 실험하였다.

성능 측정에 앞선 하드 디스크는 EXT2의 경우 모든 인자를 기본 값으로 하여 *mkfs* 명령을 내림으로써 초기화 되었으며, 설계된 파일 시스템의 경우에는 16MB의 데이터 블록 크기를 가지도록 초기화되었다. 또 설계된 파일 시스템에서는 모두 시간 기준 인덱싱 기능을 켠 채로 성능 측정이 이루어 졌으며, 새로 설계된 파일 시스템의 성능 측정 실험에서는 NVRAM도 활성화하였다. 실제 물리적인 NVRAM은 8비트 인터페이스를 통하여 참조되는 느린 메모리로서, 채널의 데이터 크기가 변한다거나 하는 거의 모든 단위 쓰기 동작 때 매번 수정되고, NVRAM 정보의 무결성을 위해 체크섬을 수행하는 등, 적지 않은 성능 저하 요인으로 작용하지만, 새로 설계된 파일 시스템의 설계 목적 가운데 하나인 신뢰성 향상을 위해 NVRAM 기능을 켜고 실험하였다.

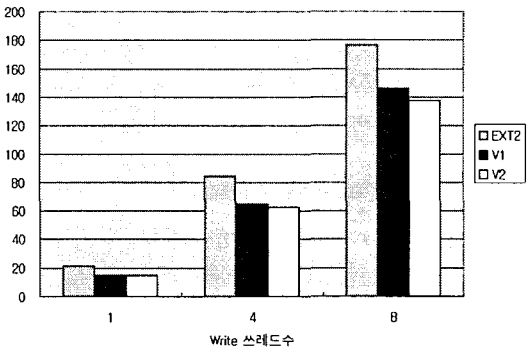
4.2 성능 평가 결과

이 절에서는 성능 평가 결과를 보이고, 그 결과를 분석한다. 성능 평가 결과는 모두 막대 그래프로 표시되며, EXT2의 경우 EXT2 파일 시스템을, V1의 경우 이전 버전 파일 시스템을 V2의 경우 이 연구에서 새로 설계된 파일 시스템을 의미한다. 모든 그래프의 Y축은 초 단위 시간을 의미한다.

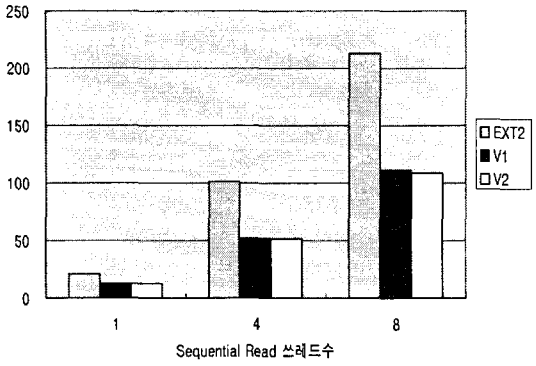
4.2.1 순차 쓰기 성능

<그림 6>은 쓰레드가 1, 4, 8 개일 때의 순차 쓰기 성능을 그래프로 나타내고 있다. 쓰기의 경우 쓰기 쓰레드가 하나인 경우 V1, V2 모두 EXT2에 비하여 31.0%의 성능 개선이 있었으며, V1, V2의 차이는 거의 없었다. 쓰레드 수가 많아져도 역시, EXT2 보다는 월등히 좋은 성능을 보이고 있다. 이는 EXT2에 비하여 설계된 파일 시스템이 훨씬 증가된 순차성으로 인하여 디스크 헤드의 움직임이 줄었기 때문이며, 쓰레드가 많아질수록 여러 채널에 의한 헤드 움직임 때문에 성능 개선 효과가 줄어들지만, 쓰기 쓰레드가 8개일 때도 V2는 EXT2에 비하여 22.1%의 성능 우위에 있음을 볼 수 있다.

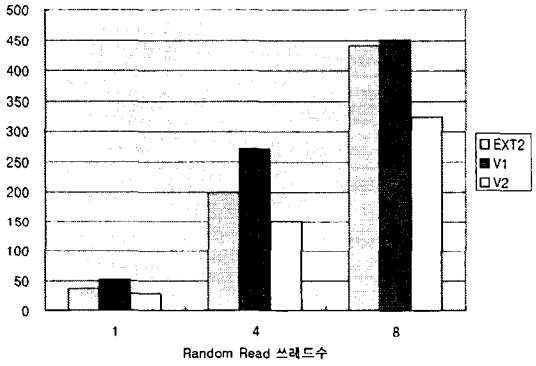
또 쓰기 쓰레드 수가 많아질수록 새로운 파일 시스템, V2의 성능의 이전 버전 V1에 비하여 개선되고 있음을 확인할 수 있다. 이는 V1의 경우 데이터 블록 리스트가 4개의 링크로 유지되기 때문에 여러 데이터 블록을 옮겨 다니며 링크를 수정해야하는 반면에, V2는 슈퍼 블록에 모든 데이터 블록 할당맵이 모여 있기 때문에 디스크 헤드의 움직임이 약간 줄어든 효과로 해석된다. 이러한 차이는 하드 디스크 내부의 헤드 스케줄링 기법에 의해 상당이 무마되기 때문에 정확한 분석이 현실적으로 어렵다.



〈그림 6〉 순차 쓰기 성능



〈그림 7〉 순차 읽기 성능



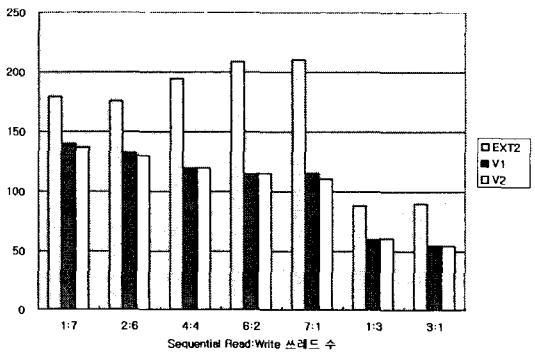
〈그림 8〉 임의 읽기 성능

4.2.2 순차 읽기 성능

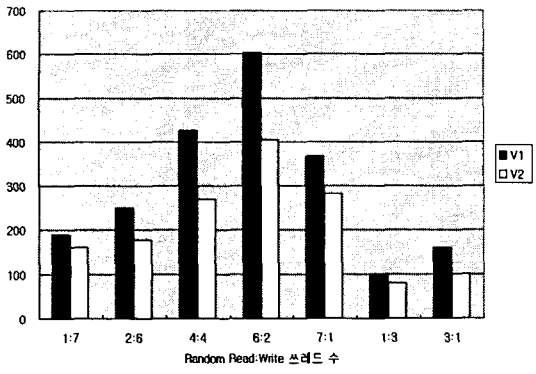
<그림 7>은 쓰레드가 1, 4, 8개 일 때의 순차 읽기 성능을 그래프로 나타내고 있다. 순차 읽기의 경우에도 설계된 파일 시스템은 EXT2에 비하여 쓰레드가 1 개인 경우 39.5%, 쓰레드가 4, 8 개일 때 각각 49.0%, 48.9%의 성능 개선 효과가 있었다. 이는 모든 메타 데이터를 마운트할 당시에 메모리에 읽어 놓고 시작하는 V2의 경우, 다음 데이터 블록을 찾기 위하여 메타 데이터를 디스크에서 읽을 필요가 없기 때문이며, V1의 경우에도 마찬가지로 다음 데이터 블록을 찾기 위해 읽고 있던 데이터 블록의 정보에 있는 다음 블록 위치를 참조할 뿐, 디스크에서 부가적인 메타 데이터를 읽을 필요가 없기 때문에 거의 같은 성능을 보이고 있다.

4.2.3 임의 읽기 성능

<그림 8>은 쓰레드가 1, 4, 8개 일 때의 임의 읽기 성능을 그래프로 나타내고 있다. 임의 읽기의 경우, 설계된 파일 시스템은 V1, V2 모두, 매번 128KB를 읽을 때마다 난수로 생성된 시간을 기준으로 인덱스 채널을 먼저 읽고 위치 정보를 확인한 뒤, 데이터 채널을 읽게 된다. V2는 이 실험에서도 EXT2에 비하여 읽기 쓰레드 수가 1, 4, 8개일 때 각각 25.4%, 24.4%, 25.5% 성능이 좋은 것으로 나타났다. 실제 시간 기준이 아닌 임의의 위치를 난수로 생성하여 *lseek*()를 한 뒤에 데이터를 읽는 EXT2의 경우와 시간 기준 인덱싱을 하는 설계된 파일 시스템은 직접 비교하는 것은 공정하지 않지만 참고적으로 성능을 비교하였다. EXT2로 시간 기준 인덱싱을 구현하기 위해서는 두 개의 파일을 참조해야하는 한다는 점을 고려할 때 이 실험에서의 성능 측정 방식은 EXT2에 매우 유리한 것임에도 불구하고, 단순한 *lseek*()만을 수행하는 EXT2보다 V2의 성능이 월등히 좋은 것은 매우 고무적이다. 이 실험에서 V1의 성능이 EXT2보다도 나쁜 이유는 매 128KB을 읽을 때마다 시간 인덱스를 찾는 과정에서 인덱스 채널에 대한 *lseek*()를 하게 되는데 이 과정에서 인덱스 채널의 데이터 블록의 링크를 따라 가면서 다음 블록의 위치를 찾기 때문에 여러 번의 데이터 블록 헤더를 읽는 과정에서 발생한 성능 저하이다.



〈그림 9〉 순차 읽기 및 쓰기 혼합 성능



〈그림 10〉 임의 읽기 및 쓰기 혼합 성능

4.2.4 순차 읽기 및 쓰기 혼합 성능

<그림 9>는 순차 읽기와 쓰기가 혼합된 쓰레드가 4 또는 8개 일 때의 성능을 그래프로 나타내고 있다. 순차 읽기와 쓰기가 혼합된 경우에도 설계된 파일 시스템의 성능이 좋은 것으로 나타나고 있다. 앞의 순차 읽기 실험 결과를 반영하듯이, 순차 읽기 쓰레드 수가 상대적으로 많아질수록 EXT2와의 성능 격차가 벌어지는 것으로 실험 결과가 나오고 있으며, V2는 EXT2에 비하여 실험 조합에 따라 최소 23.9% (읽기 : 쓰기 비율이 1 : 7인 경우)에서 최대 47.7%(읽기 : 쓰기 비율이 7 : 1인 경우)의 차이를 보였다. V1과 V2의 경우 순차 읽기, 순차 쓰기의 경우에서처럼 거의 유사한 성능을 보이고 있음을 알 수 있다.

4.2.5 임의 읽기 및 쓰기 혼합 성능

<그림 10>은 임의 읽기와 쓰기가 혼합된 쓰레드가 4 또는 8개 일 때의 성능을 그래프로 나타내고 있다. 이 실험은 임의 읽기와 순차 쓰기를 혼합한 경우이다.

EXT2로 시간 기준 인덱싱을 구현하기 위해서는 시간 인덱싱 정보가 들어있는 파일을 먼저 읽어 위치 정보를 얻은 뒤에, 실제 데이터 파일에 대하여 *lseek*()를 먼저 수행한 뒤 읽기를 수행해야 하며, 마찬가지로 쓰기 때도 데이터 파일과 시간 인덱스 정보가 저장된 파일 등 두 파일에 동시에 써야한다. 이렇게 하는 것은, 이 논문에서 제안하는 것처럼 시간 인덱싱 기법을 파일 시스템에서 직접 지원하는 방식과는 너무 상이한 방법이기 때문에, 임의 읽기 및 쓰기 혼합 성능 측정에는 EXT2를 제외하고 시간 기준 인덱싱 기법이 구현된 V1과 V2의 성능만을 측정하여 비교하였다.

성능은 임의 읽기 실험 결과가 반영되어 읽기 쓰레드의 수가 많을수록 V2의 성능이 상대적으로 좋게 나타났다. 역시 시간 기준으로 인덱스를 검색하는 과정에서 V1은 적지 않은 오버헤드가 있었으며, V2에서는 그 부분이 데이터 배치 구조 설계에서 개선되었기 때문이다. 읽기 쓰레드만 있었던 경우보다는 쓰기 쓰레드의 효과에 따라 성능의 차이가 약간 줄었지만 그럼에도 불구하고 실험 조합에 따라 최소 15.5% (읽기 : 쓰기 비율이 7 : 1인 경우)에서 최대 36.8% (읽기 : 쓰기 비율이 4 : 4인 경우)의 차이를 보였다.

5. 결 론

기존 리눅스 파일 시스템은 대용량 파일 처리를 위한 기능 미흡, 느린 읽기 속도, 부적절한 인덱스 구조, 데이터 무결성에 대한 보장 미약 등의 문제로 인하여 대형 멀티미디어 파일이 저장되고 재생되는 개인용 멀티미디어 시스템을 지원하기 어려웠다. 본 연구에서는 위와 같은 문제점을 해결할 수 있는 새로운 파일 시스템을 설계하고 구현하였다.

설계된 파일 시스템은 데이터의 순차성 보강, 단순한 메타 데이터 구조, 채널별 데이터 관리, 대용량 데이터 블록의 사용, 효과적인 미리 읽기, 데이터 무결성 보장을 위한 NVRAM의 사

용, 시간 기준 인덱싱 등을 통해 개인용 멀티미디어 시스템에 필요한 성능과 안정성을 확보하였다. 또 본 연구에서는 설계된 파일 시스템을 리눅스의 커널 모듈 형태로 구현하고, 성능 평가를 수행하여 EXT2보다 순차 읽기와 쓰기의 혼합 시험에서 최대 47.7% 높은 성능을 보임을 검증하였으며, 이 파일 시스템의 기초가 되었던 이전 버전의 파일 시스템보다 임의 읽기와 쓰기 혼합 시험에서 최대 36.8%의 성능 향상을 얻는 등, 좋은 성능을 보임으로서 멀티미디어 파일 시스템으로서의 적합성을 검증하였다.

이 파일 시스템은 앞으로도 지속적인 발전이 예상되는 멀티미디어 시스템을 위한 파일 시스템의 구현을 위한 좋은 기초 자료로 활용될 수 있으며, 리눅스나 그 밖의 운영 체제를 사용하는 멀티미디어 상품에 쉽게 적용될 수 있다. 현재 이 파일 시스템은 임베디드 리눅스 및 실시간 운영체제인 pSOS 상에도 이식되어 실제 상용 DVR에 적용되고 있으며, DVR 뿐만 아니라 PVR, 캠코더, 방송용 녹화, 편집, 저장 시스템, 대용량 데이터 로깅 시스템 등에도 적용될 수 있다.

참 고 문 헌

[1] Bar, Moshe, *Linux File Systems*, McGraw Hill Companies, 2001.

[2] 권수호, *Linux Kernel Programming Bible 2nd Edition*, 글로벌, 2002.

[3] Card, R., Ts'o, T. and Tweedie, S., *Design and Implementation of the Second Extended Filesystem*, First Dutch International Symposium on Linux, 1994.

[4] Tweedie, Stephen, EXT3, Journaling Filesystem, the Ottawa Linux Symposium, http://olstrans.sourceforge.net/release/OLS2000-EXT3/OLS2000-EXT3.html, July, 2000.

[5] Robbins, Daniel, 고급 파일 시스템 개발자 가이드, Part 7 EXT3, IBM, http://www-903.ibm.com/developerworks/kr/linux/library/l-fs7.html.

[6] Bryant, R., Forester, Ruth and Hawkes, John, *Filesystem Performance and Scalability in Linux 2.4.17*, 2002 USENIX Conference, 2002.

[7] 권우일, 윤미현, 이동준, 장재혁, 양승민, *DVR 시스템을 위한 저널링 파일 시스템의 성능평가*, 2002 추계 한국정보과학회 논문지, 2002.

[8] Journaled File System Technology for Linux, html document of IBM, http://oss.software.ibm.com/jfs/.

[9] XFS html document of SGI, http://oss.gi.om/projects/xfs/.

[10] Sweeney, A., Doucette, D., Hu, W., Anderson, C. Nishimoto, M. and Peck G., *Scalability in the XFS File System*, 1996 USENIX Conference, 1996.

[11] 원유집, 박진연, *정보가전용 멀티미디어 파일 시스템 기술*, 멀티미디어와 정보화 사회, 한국과학기술진흥재단, http://211.40.179.13/ok_file/ke28/ke028-index.htm.

[12] Czezatke, Christian and Ertl, M.A., *A Log-tructured Filesystem for Linux*, 2000 USENIX Conference, 2000.

[13] Hitz, D., Lau, J. and Malcom, M., *File system design for an NFS file server appliance*, In Proceedings of the Winter 1994 USENIX Technical Conference, pp. 235-246, San Francisco, CA, Jan. 1994.

[14] Miller, E.L., Brandt, S.A. and Long, D.D.E., *HeRMES : High-performance reliable M-RAM-nabled storage*, In Proceedings of the 8th IEEE Workshop on Hot Topics in

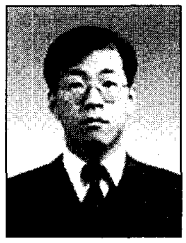
Operating Systems(HotOS-VIII), pp. 83- 87, Schloss Elmau, Germany, May 2001.

[15] Woodhouse, D., *The journalling flash file system*, In Ottawa Linux Symposium, Ottawa, ON, Canada, July 2001.

[16] Levy, M., *Memory Products*, chapter Interfacing Microsoft's Flash File System, Intel Corporation, 1993, pp. 4318-4325.

[17] 이민석, "대형 멀티미디어 파일을 위한 파일 시스템 구현", *Journal of Informaion Technolgy Application & Management*, Vol. 10, No. 4, 2003, pp. 169-183.

저자소개



이 민 석

1986년 서울대학교 컴퓨터공학과 학사, 1988년 서울대학교 컴퓨터공학과 석사, 1995년 서울대학교 컴퓨터공학과 박사를 마치고 1999년~2002년 (주)팜팜테크 CTO를 지내고, 1995년~현재 한성대학교 컴퓨터공학부 부교수로 재직중이다. 관심 분야로는 실시간 시스템, 임베디드 시스템, 임베디드 리눅스 등이다.

◈ 이 논문은 2004년 11월 1일 접수하여 1차 수정을 거쳐 2005년 1월 10일 게재확정되었습니다.