DOI: 10.3745/KIPSTA.2009.16-A.2.89

Ext3 파일 시스템 기반의 편집된 대용량 멀티미디어 파일의 고속 저장 기법

정 승 완[†] · 남 영 진[††] · 서 대 화[†††]

요 약

디지털 기술의 발전과 고화질 미디어의 확산에 따라 휴대폰, 디지털 TV, PMP, 디지털 캠코더, 디지털 카메라 등의 멀티미디어 장치에 대한 사용자 수요가 점차 증가하는 추세이다. 이러한 장치들은 멀티미디어 파일 재생 및 편집 등의 멀티미디어 파일 처리와 관련된 다양한 서비스를 제공하고 있다. 그 중 편집된 대용량 멀티미디어 파일을 저장하는데 있어, 기존 파일 시스템은 많은 시간과 디스크 입출력을 소요하는 성능상의 문제점을 갖고 있다. 본 논문에서는 Ext3 파일 시스템에 아이노드 블록 포인터 재설정을 통한 데이터 블록 공유 기법을 추가하여 편집된 대용량 멀티미디어 파일을 빠르고 효율적으로 저장할 수 있는 기법을 제시한다. 제안된 기법을 적용할 경우 다양한 형태로 편집된 파일에 대한 Ext3 파일 시스템의 저장 성능을 평균 16배 향상시킬 뿐 아니라, 데이터 블록 공유를 통해 사용된 디스크 공간을 획기적으로 줄여줄 수 있음을 실험을 통하여 보여준다.

키워드 : 파일 편집, 멀티미디어 파일, 데이터 블록 공유

A Fast Writing Technique of Large-sized Edited Multimedia Files based on the Ext3 File System

Seung-Wan Jung[†] · Young Jin Nam[††] · Dae Wha Seo[†††]

ABSTRACT

With the advance in digital technologies and the increasing prevalence of high quality multimedia contents, there is a growing user demand for multimedia devices, such as mobile phones, digital TV, PMP, digital camcoders, digital cameras. Such devices provide various services associated with multimedia file manipulation, including multimedia contents playback, multimedia file editing, etc. Conventional file systems exhibit a performance-related drawback that requires considerable amount of time and disk I/Os in order to store large-sized edited multimedia files. This paper proposes a fast, efficient writing technique for large-sized edited multimedia files by using data block sharing with adjustment of inode block pointers. Our experiments show that the proposed scheme not only improves write performance of the Ext3 file system on average by 16 times with various types of edited multimedia files, but also reduces consumed disk space dramatically through the data block sharing.

Keywords : File Editing, Multimedia File, Data Block Sharing

1. 서 론

디지털 기술의 발달로 디지털 TV, 캠코더와 같은 멀티미디어 장치의 사용이 증가하고 있다. 이러한 장치들은 일반적으로 저장장치를 내장하고 있으며, PVR(Personal Video Recorder)[1] 기능을 통해 영상을 녹화하여 저장할 수 있고 장치 내에서 재생할 수 있다. 또한, PVR은 그 기능을 확장해 녹화물에 대해 원하는 부분만을 선택적으로 저장할 수 있는 편집 기능을 제공한다. 사용자는 편집 기능을 이용하여 영상물에서 광고 등 불필요한 부분을 삭제하여 원하는 영상물을 만들 수 있다. 이와 같은 편집 기술은 미디어 컨텐츠의 유통 및 관리에 있어 필수적이며, UCC(User Created Contents)의 보편화 등으로 인해 그 필요성이 점차 증가하고 있다. 최근의 미디어 컨텐츠는 다양한 화질을 가지고 있으며, 같은 내용을 촬영 혹은 녹화하더라도 고화질일수록 더 많은 용량을 차지한다[2]. 디지털 TV, 디지털 캠코더 등과 같은 멀티미디어 장치는 HD급, 풀(full) HD급의 고화질

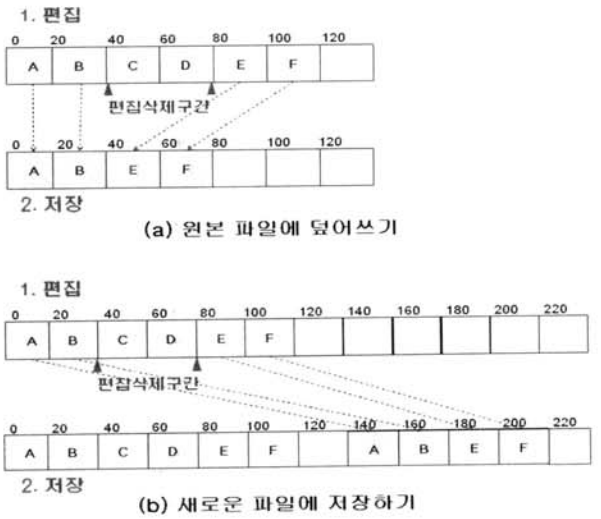
※ 본 연구는 경북대학교 BK21 사업과 지식경제부, 정보통신연구진흥원의 대학 IT연구센터 지원사업과 대구대학교 교내학술연구비 지원을 받아 이루어졌음(IITA-2008-C1090-0801-0045).
† 준 회 원 : 경북대학교 전자전기컴퓨터학부 박사과정
†† 정 회 원 : 대구대학교 컴퓨터IT공학부 조교수(교신저자)
††† 종신회원 : 경북대학교 전자전기컴퓨터학부 교수
논문접수 : 2008년 10월 21일
수 정 일 : 1차 2009년 2월 9일, 2차 2009년 2월 17일
심사완료 : 2009년 2월 17일

미디어 컨텐츠를 제공한다. 풀 HD급 화질의 경우 약 20Mbps의 전송률을 가지며, 한 시간 분량의 영상물에 대해 약 9GB의 용량을 가진다[3, 4]. 먼저 이러한 대용량 멀티미디어 파일에 대한 일반적인 파일 편집 과정과 편집 시 발생하는 성능상의 문제점에 대해 알아본다.

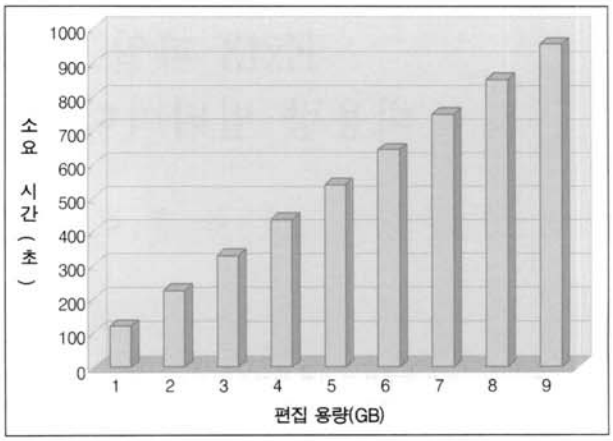
멀티미디어 파일 편집 과정은 (그림 1)에서와 같이 일반적으로 편집 과정과 저장 과정으로 나누어질 수 있다. 편집 과정은 원본 멀티미디어 파일로부터 삭제할 부분을 선택한 후에 원본 파일에 저장할지 혹은 새로운 파일에 저장할지를 결정한다[5]. (그림 1)(a)는 원본 파일에서 C와 D부분을 삭제하고 동일한 원본 파일에 저장하는 과정을 보여주며, (그림 1)(b)는 C와 D부분을 삭제한 후에 새로운 파일에 저장하는 예를 보여주고 있다.

저장 과정은 파일 시스템을 통해 편집 과정에서 삭제되고 남은 부분만을 저장장치에 순차적으로 저장하며, 대용량 멀티미디어 파일의 경우 일반적으로 매우 긴 저장 시간을 필요로 한다. 기존 파일 시스템은 이러한 편집 후 저장 작업을 수행할 때에 효율적인 저장 기법을 제공하지 못한다.

이를 확인하기 위해서 임베디드 시스템에서 사용되는 Ext3 파일 시스템[6]을 이용하여 멀티미디어 파일 편집 후 저장 실험을 수행하였다. 실험에서 멀티미디어 파일은 USB 2.0 인터페이스를 통하여 휴대용 디스크에 저장하도록 설정하였다. (그림 2)는 Ext3 파일 시스템에서 10GB 원본 멀티미디어 파일을 편집 후 남은 용량(실제적으로 저장되는 용량)을 변화시켜가며 파일을 저장할 때 소요되는 시간을 보여주고 있다. Ext3 파일 시스템에서 제공하는 시스템 콜 인터페이스를 이용하여 편집된 파일을 저장할 경우, 일반적으로 편집 저장되는 내용을 원본 파일로부터 순차적으로 읽은 후에 새로운 파일에 순차적으로 저장하는 방식을 사용한다[7]. 이와 같은 방식은 작은 용량의 파일에 대해서는 알고리즘의 단순성을 생각할 때 솔루션이 될 수 있다. 하지만, 고화질 멀티미디어 파일과 같은 대용량 파일의 경우에는 편집 저장되는 파일의 용량이 일반적으로 매우 크기 때문에 많은



(그림 1) 멀티미디어 파일 편집 과정



(그림 2) Ext3 파일 시스템 파일 편집 후 저장 시 소요 시간

시간이 소요될 뿐 아니라 많은 디스크 대역폭을 소모한다[8, 9]. 그림에서 확인할 수 있듯이 9GB 파일을 저장하기 위해서는 약 15분 이상이 소요되는 것으로 파악되었다.

본 논문에서는 편집된 대용량 멀티미디어 파일을 Ext3 파일 시스템을 통하여 저장할 때 발생하는 긴 저장 시간과 관련된 성능상의 문제점을 극복할 수 있는 기법을 제시한다. 제안된 기법은 기존 Ext3 파일 시스템에 편집 저장을 위한 새로운 인터페이스를 제공하며, 이를 지원하기 위한 아이노드 블록 포인트 재설정 기법과 공유데이터 블록 관리 기법을 제시한다. 이러한 기법을 Ext3 파일 시스템에 적용하여 대용량 파일 편집 시 소요되는 시간을 대폭 감소시킬 수 있을 뿐 아니라, 데이터 블록 공유를 통하여 소요되는 저장 공간을 획기적으로 감소시켜준다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 공유 데이터 블록 및 대용량 파일 편집 관련 연구를 소개한다. 3장에서는 본 논문의 기본 알고리즘인 아이노드 블록 포인터 재설정 기법과 공유 데이터 블록 관리 메커니즘에 대한 설명과 대용량 파일 편집 시 동작 과정에 대해 다룬다. 4장에서는 성능 평가와 실험 결과를 기술한다. 마지막으로 5장에서는 결론 및 향후 연구에 대해 설명한다.

2. 관련 연구

본 장에서는 본 논문의 중요한 기법 중 하나인 공유 데이터 블록 사용과 관련된 연구와 다양한 파일 시스템에서의 파일 편집 방법들에 대해 설명한다. 먼저 공유 데이터 블록 사용과 관련된 연구에 대해 살펴본다.

공유 데이터 블록을 지원하는 파일 시스템으로는 SAN (Storage Area Network)환경에서 사용하는 GFS(Global File System)[10]와 SANtopia[11] 등이 있다. 이와 같은 파일 시스템들은 여러 개의 서버가 하나의 저장장치를 공유하여 사용할 수 있도록 메타데이터를 통해 공유 데이터 블록을 관리한다. 즉, 다수의 시스템에서 하나의 저장 장치에 기록된 데이터에 접근할 수 있도록 처리해준다. 하지만 위의 파일 시스템에서의 공유 기법은 하나의 시스템에서 파일 간 데이

터 블록을 지원하는데 있어서는 적용하기 힘들다.

VBS(Variable Block Size)[12], XPRESS(eXtendible Portable Reliable Embedded Storage System)[13], Ext3Cow [14]와 같은 파일 시스템에서 파일 편집과 관련된 연구가 이루어졌다. 이러한 파일 시스템들은 기존 FAT, Ext3 파일 시스템에서 사용하는 파일 편집 후 일괄 저장 방식과는 다른 파일 편집 방식을 제공한다. 파일 편집 시에 편집된 데이터의 부분만을 저장할 수 있는 가장 대표적인 방식은 VBS 방식이다. VBS를 지원하는 시스템에서는 파일 시스템 데이터 저장 단위인 블록이 고정되어 있는 것이 아니라 가변적이다. 따라서 VBS는 블록 크기를 조정하여 필요한 데이터만을 저장 가능하다. 하지만 VBS 방식은 아직 연구가 많이 이루어져 있지 않고 실제 구현이 어렵다는 문제점이 존재한다.

XPRESS는 멀티미디어 파일을 기반으로 하는 시스템에서 사용하며, 파일 데이터의 수정 없이 부가적인 메타데이터를 이용하여 파일 편집을 지원하는 파일 시스템이다. 이 방식은 부가적인 메타 데이터를 필요로 하며 부가적인 메타데이터는 데이터베이스로 관리한다. 파일의 범위를 나타내는 메타 데이터는 파일을 읽을 때에 파일의 어느 부분을 읽어야 한다는 것을 파일 오프셋(offset)값을 통해 지정한다. 따라서 파일 편집 시 데이터베이스에 저장된 파일의 범위를 나타내는 메타 데이터를 수정해 주면 실제 파일의 편집 없이 빠르게 편집을 할 수 있다. 하지만 이러한 방식은 파일 편집 후 사용 가능한 디스크 용량이 증가하지 않고 XPRESS의 기반이 되는 Ext3 파일 시스템과 호환성이 없다.

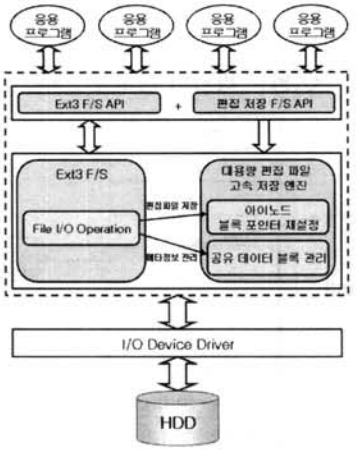
Ext3cow는 기존 리눅스 Ext3 파일 시스템에서 카피-온-라이트(copy-on-write) 기법을 이용하여 스냅샷(snapshot)과 버전닝(versioning)을 제공하는 파일 시스템이다. Ext3cow는 전자 문서에 대해서도 종이 문서와 같이 과거 데이터들을 모두 유지할 수 있는 방법을 제공하기 위해 만들어졌다. 파일 편집 시 Ext3cow는 변경된 데이터 블록에 대해서만 할당하여 새로 쓰고 다른 데이터 블록은 기존 파일과 공유하는 구조를 가진다. 이 방식은 미리 정해진 크기의 파일 편집 시 공유 데이터 블록을 사용하여 디스크 I/O를 줄일 수 있다는 장점을 가진다. 하지만 대용량 멀티미디어 파일 편집에서와 같이 파일의 불필요한 부분을 삭제하여 파일 크기가 변하는 편집에서는 적용될 수 없다.

3. 대용량 편집 멀티미디어 파일 고속 저장 기법

3.1 제안된 구조

대용량 멀티미디어 파일을 메타데이터의 수정만으로 편집 효과를 얻을 수 있게 하는 제안된 시스템은 기존의 리눅스 파일 시스템인 Ext3에 '대용량 편집 파일 고속 저장 엔진'과 '편집 저장 파일시스템 API'을 추가하여 설계한다. (그림 3)은 대용량 편집 멀티미디어 파일 고속 저장 기법 구조를 보여준다.

대용량 편집 파일 고속 저장 엔진은 대용량 멀티미디어



(그림 3) 대용량 편집 멀티미디어 파일 고속 저장을 위한 제안 구조

파일의 메타데이터의 조작을 통해서 파일의 편집을 빠르게 처리해 준다. 대용량 편집 파일 고속 저장 엔진은 기존 파일 시스템의 아이노드 정보를 관리하는 '아이노드 블록 포인터 재설정 모듈'과 새로운 파일을 생성할 수 있는 해 주는 '공유 데이터 블록 관리 모듈'로 구성된다. 아이노드 블록 포인터 재설정 모듈은 데이터 블록의 입출력 없이 메타데이터 수정만을 통한 빠른 편집을 제공한다. 공유 데이터 블록 관리 모듈은 공유 데이터 블록을 가지는 파일과 공유 데이터 블록에 대해 부가적인 별도의 메타데이터를 생성하여 새로 만들어 진 파일의 데이터 블록을 관리한다.

편집 시 대부분의 시간을 데이터 블록 I/O에 소모하던 기존 편집 저장 방식에 비하여, 이 기법은 메타데이터만을 수정하여 대규모 파일을 편집하므로 시간을 절약할 수 있으며, 사용하는 디스크의 대역폭도 크게 줄일 수 있다.

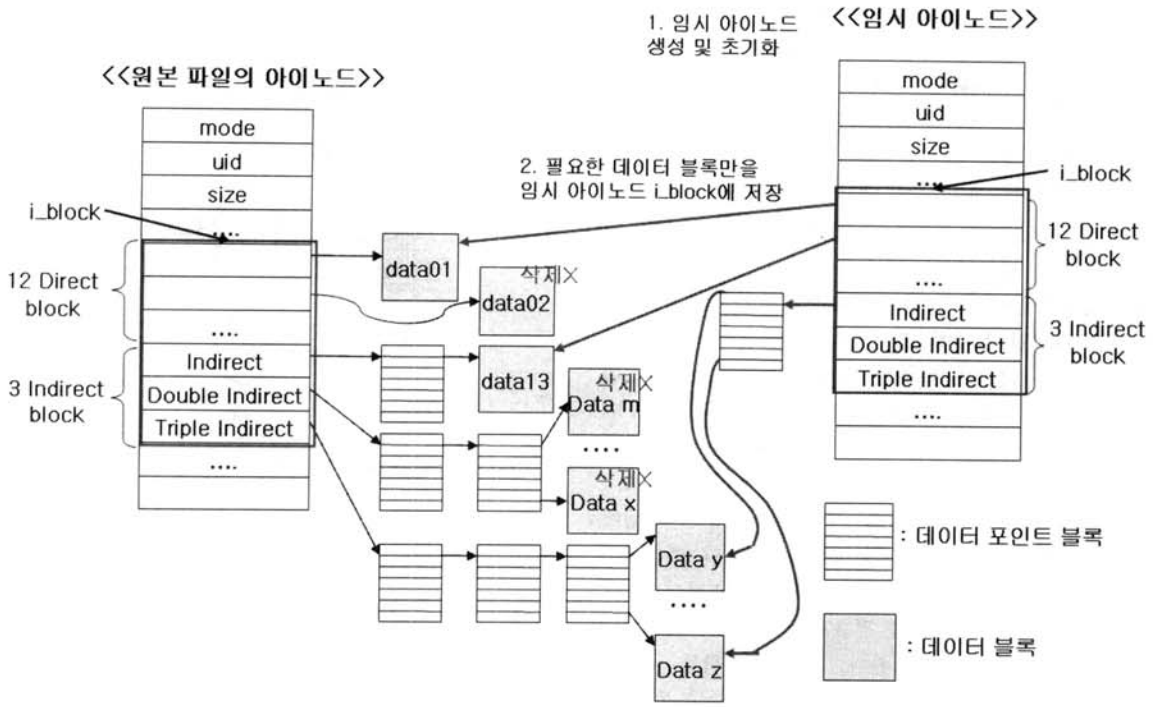
대용량 편집 파일 고속 저장 엔진에 구현된 기능을 손쉽게 이용하기 위하여 편집 저장 파일시스템 API를 제공한다. 편집 저장 파일시스템 API의 기본 프로토타입은 다음과 같다.

edited_inode *EDIT_IO(original_inode, edited_info{offset1, size1, • • ,offseti, sizei}, NEW_F or SAME_F)

- original_f_inode : 원본 파일에 대한 메타데이터(inode) 정보
- edited_f_info : 편집과 관련된 정보
- NEW_F or SAME_F : 같은 파일에 편집된 내용을 저장할 것인지 혹은 새로운 파일에 편집된 내용을 저장할 것인지를 결정하는 옵션
- edited_f_inode : 편집 저장된 파일의 메타데이터(inode) 정보

3.2 아이노드 블록 포인터 재설정 모듈

아이노드 블록 포인터 재설정 모듈은 파일 아이노드의 i_block 필드를 재설정하여 파일의 블록 단위 삭제나 삽입을 수행한다. 여기서 아이노드는 Ext3 파일 시스템의 파일 정보를 담고 있는 아이노드를 의미하며, i_block 필드는 아이노드 내에서 파일의 데이터 블록 인덱스를 가지고 있다.



(그림 4) 아이노드 블록 포인터 재설정 모듈의 동작 예

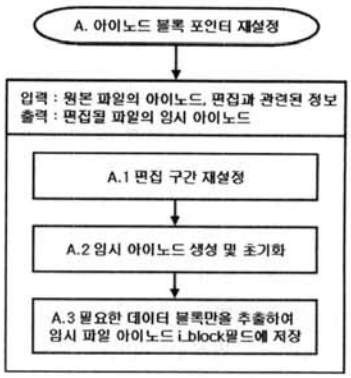
(그림 4)는 아이노드 블록 포인터 재설정 모듈의 동작 과정을 나타낸다.

첫째, 아이노드 블록 포인터 재설정 모듈은 원본 파일의 아이노드, 편집 관련 오프셋과 데이터 크기를 입력받아 편집 구간을 재설정한다. 편집 관련 오프셋은 원본 파일에서 저장해야 할 영역의 시작점과 종료점의 오프셋 값이다. 아이노드 블록 포인터 재설정 모듈은 블록 단위 편집을 하기 때문에 선택된 편집 구간을 Ext3 파일 시스템의 데이터 블록 크기인 4KB 단위로 재설정 한다. 이때 4KB 미만의 데이터 블록 내 손실을 가지고 올 수 있다. 하지만 텍스트 기반 파일이 아닌 대용량 멀티미디어 파일에서 4KB 데이터 블록 내 손실은 멀티미디어 파일 재생에 있어 큰 영향을 미치지 않는다. 이것에 대한 자세한 설명은 4.4.1절에서 다루도록 한다.

둘째, 편집될 동영상 파일의 데이터 블록 인덱스를 저장하기 위한 임시 아이노드를 생성하여 메모리에 저장한다. 이 때, 편집될 동영상 파일의 임시 아이노드는 새로운 값을 저장하기 위해 그 값을 모두 0으로 초기화 시킨다.

셋째, 원본 동영상 파일 아이노드의 i_block 필드에서 재설정된 편집 구간에 속하는 데이터 블록을 추출하여 임시 아이노드 i_block 필드에 순서대로 저장한 후, 편집될 파일의 데이터 블록 인덱스를 저장하고 있는 임시 아이노드를 반환한다.

제안 기법이 적용된 시스템에서 실제 파일 편집 시, 아이노드 블록 포인터 재설정 모듈은 데이터 블록 관리를 위한 작업과 함께 동작하며, 편집한 내용을 원본 파일에 덮어 쓰거나 새로운 파일에 쓸 경우 모두 사용된다. 아이노드 블록 포인터 재설정 모듈은 편집 저장 시 대부분의 시간을 소모하는 데이터 블록의 입출력 없이 메타데이터 수정만을 통한 빠른 편집을 제공한다. (그림 5)는 아이노드 블록 포인터 재설정 모듈의 동작 예를 보여주고 있다. 파일 편집 시 동작 과정은 3.4절에서 자세히 다루도록 한다.



(그림 5) 아이노드 블록 포인터 재설정 모듈 동작 과정

3.3 새 파일 저장을 위한 공유 데이터 블록 관리 모듈

대용량 편집 멀티미디어 파일 고속 편집 기법에서는 데이터 블록 공유를 위하여 부가적인 메타데이터를 사용한다. 데이터 블록 공유를 위한 부가적인 메타데이터는 멀티미디어 파일 중 필요한 부분만을 선택하여 새로운 파일에 저장할 경우 생성된다. 원본 멀티미디어 파일과 편집한 내용을 저장한 새로운 멀티미디어 파일은 데이터 블록을 공유하여 사용한다. 공유 데이터 블록을 사용함으로써 이미 디스크에 기록된 데이터 내용을 중복적으로 쓰는 것을 방지하며 편집한 내용을 새로운 파일에 저장할 때 시간 낭비와 디스크 공간 소비를 줄일 수 있다.

(그림 6)은 공유 데이터 블록 관리를 위해 필요한 메타데이터 구조를 나타낸다. 공유 데이터 블록 관리를 위한 메타데이터는 편집 파일 관리자(EFM), 공유 블록맵 포인터(SBMP), 공유 블록맵(SBM)으로 구성된다. 공유데이터 블록 관리를 위한 메타데이터의 기능과 관리 및 접근 방법을 <표 1>에 요약하였다.

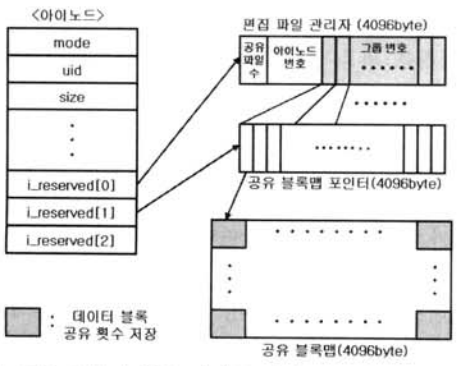
각 메타데이터의 기능 및 파일 편집 시 관리, 접근 방법은 다음과 같다. SBM은 Ext3 파일 시스템의 그룹 내 블록들의 공유 횟수 값을 저장한다. Ext3 파일 시스템의 블록 비트맵은 그룹 내 블록의 사용여부를 한 비트에 0 혹은 1로 나타낸다. 따라서 Ext3 파일 시스템의 블록 비트맵은 데이터 블록 공유 횟수를 나타낼 수 없다. 데이터 블록 공유를 허용하는 제안기법에서 데이터 블록 공유 횟수를 저장하기 위한 부가적인 메타데이터인 SBM이 필요하다. SBM은 블록 비트맵과 같이 그룹 단위의 데이터 블록에 대해 공유 횟수 정보를 저장하며, 저장 단위 크기는 1바이트이고, 하나의 블록 비트맵에 대해 8개의 SBM이 생성된다.

SBMP는 SBM이 저장된 블록 주소를 저장한다. SBMP는 EFM의 그룹 번호 필드에 기록된 그룹 순으로 8개 단위로 SBM을 저장한다. SBMP는 하나의 데이터 블록을 할당받아 저장되며, 공유 파일의 아이노드 i_reserved[1] 필드에 블록 주소를 저장한다.

EFM은 데이터 블록을 공유하고 있는 파일을 관리하고 파일의 데이터 블록이 SBMP에서 자신의 공유 횟수 정보가 저장된 SBM에 접근할 수 있도록 해 준다. 이것을 위해 EFM에는 공유 파일 수, 공유 파일의 아이노드 번호, 공유 데이터 블록이 속한 그룹 번호와 같은 정보를 저장한다.

〈표 1〉 공유데이터 블록 관리를 위한 메타데이터 기능

구 분	기 능	관리 및 접근
편집 파일 관리자(EFM)	데이터 블록을 공유하는 파일 관리 및 파일의 데이터 블록이 자신의 공유 블록맵에 접근할 수 있는 기능 제공	아이노드 i_reserved[0] 필드
공유 블록맵 포인터(SBMP)	해당 파일이 참조하는 공유 블록맵의 블록 주소를 저장	아이노드 i_reserved[1] 필드
공유 블록맵(SBM)	Ext3 파일 시스템의 그룹 내 블록들의 공유 횟수를 저장	공유 블록맵 포인터



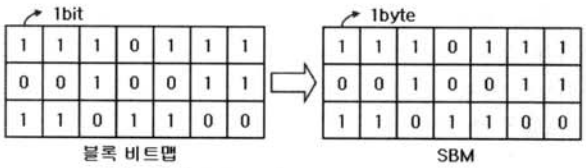
(그림 6) 공유 데이터 블록 관리를 위해 필요한 메타데이터 구조

EFM은 하나의 데이터 블록을 할당받아 저장되며, 공유 파일의 아이노드 i_reserved[0] 필드에 블록 주소를 저장한다.

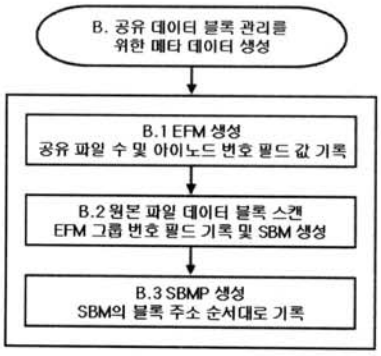
공유 데이터 블록 관리를 위한 메타데이터는 원본 파일에 대해 편집한 내용을 새로운 파일에 저장할 때에 생성되며, 데이터 블록을 공유하는 파일이 삭제되어 데이터 블록을 공유하는 파일이 없을 경우에 공유 데이터 블록 관리를 위한 메타데이터도 삭제된다. 본 논문에서 제안하는 공유 데이터 관리 기법은 파일 단위 데이터 블록 공유를 허용하고 동적으로 생성 및 삭제가 가능하다.

(그림 8)은 공유 데이터 블록 관리를 위한 메타데이터 생성 과정을 나타낸다. 첫째, EFM을 생성하여 공유 파일 수와 파일의 아이노드 번호 필드에 값을 기록한다. 공유 파일 수 필드값은 원본 파일과 새로운 파일이 데이터 블록을 공유하므로 2로 된다. 파일의 아이노드 번호 필드에는 원본 파일과 새로운 파일의 아이노드 번호를 기록한다. 둘째, 원본 파일의 모든 데이터 블록을 스캔하여 데이터 블록이 속한 그룹 번호를 EFM의 그룹 번호 필드에 기록하고 그룹 번호에 해당하는 블록 비트맵을 (그림 7)과 같이 SBM으로 변환한다. SBM은 그룹 개수 * 8개의 데이터 블록을 할당 받아 만들어진다. 셋째, 만들어진 SBM의 블록 주소를 SBMP에 EFM의 그룹 번호 필드에 기록된 그룹 순으로 저장한다.

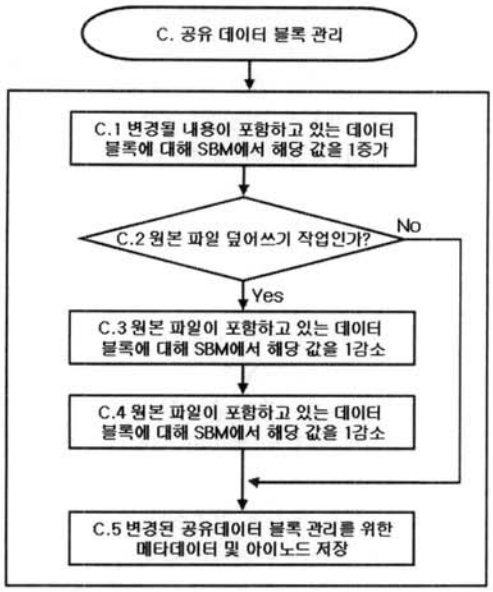
새로운 파일에 저장하기 작업 혹은 원본 파일에 덮어쓰기 작업 시 공유 데이터 블록 관리는 (그림 9)와 같은 절차로 수행된다. 공유 데이터 블록 관리는 공유 데이터 블록 관리를 위한 메타데이터가 생성되어 있을 때 수행된다. 새로운 파일에 저장하기 작업 시 새로운 파일이 포함하는 데이터 블록에 대해 SBM에서 공유 횟수 값을 1 증가 시킨다. EFM의 공유 파일 수와 파일의 아이노드 번호 필드에 새로운 파일 생성에 대한 결과를 반영한다. 즉, 공유 파일 수 필드값을 1 증가시키고, 파일의 아이노드 번호 필드에 새로운 파일의



(그림 7) 블록 비트맵 SBM 변환 과정



(그림 8) 공유 데이터 블록 관리를 위한 메타데이터 생성 과정



(그림 9) 공유 데이터 블록 관리 과정

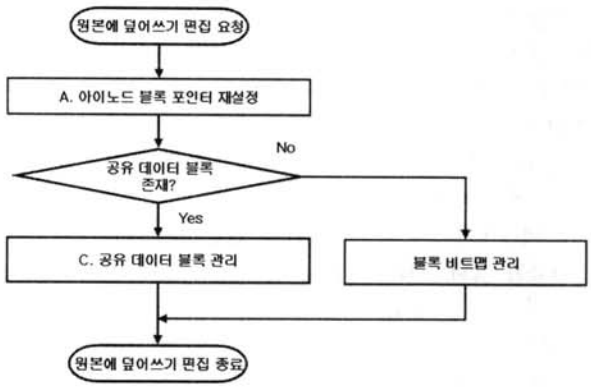
아이노드를 추가한다. 마지막으로 변경된 SBM, EFM 그리고 새로운 파일의 아이노드를 디스크에 기록한다. 원본 파일에 덮어쓰기 작업 시 공유 데이터 블록 관리 방법은 새로운 파일에 저장하기 작업 시 공유 데이터 블록 관리 방법과 차이가 있다. 변경될 내용이 포함하고 있는 데이터 블록에 대해 SBM에서 공유 횟수 값을 1 증가시킨다. 원본 내용이 포함하고 있는 데이터 블록에 대해 SBM에서 공유 횟수 값을 1 감소시키고 그 값이 0으로 변한 데이터 블록에 대해 Ext3 파일 시스템의 블록 비트맵에서 해당 데이터 블록에 대한 값을 0으로 변경시킨다. 마지막으로 변경된 SBM과 아이노드를 디스크에 기록한다.

3.4 제안 기법을 이용한 편집 파일 저장

편집된 내용을 원본 파일에 덮어 쓰거나 새로운 파일에 쓰기와 같은 편집 작업에 대해 고속 저장을 지원하기 위해 아이노드 블록 포인터 재설정 모듈과 공유 데이터 블록 관리 모듈을 추가하였으며, 공유 데이터 블록을 처리할 수 있도록 기존 삭제 방법을 수정하였다.

본 장에서는 앞 절에서 기술한 두 모듈들을 기반으로 멀티미디어 파일 편집 시 동작 과정과 수정된 삭제 기능에 대해 구체적으로 기술한다. 제안기법을 적용한 Ext3 파일 시스템은 편집한 내용을 원본 파일에 덮어 쓰기와 새로운 파일에 쓰기 편집에 대한 고속 저장 기능을 제공한다.

첫째, 편집한 내용을 원본 파일에 덮어 쓸 경우 동작 절차를 (그림 10)에 나타내었다. 편집된 내용을 원본 파일에 덮어쓴다는 것은 원본 파일 중 필요한 부분만을 선택하여 원본 파일에 재 저장한다는 것을 의미한다. 먼저 편집한 내용을 원본 파일에 덮어 쓰기 요청과 함께 앞서 설명한 아이노드 블록 포인터 재설정 모듈을 이용해 편집하고자 하는 데이터 블록만을 재 저장하기 위해 생성된 아이노드 구조체의 포인터 배열에 저장한다. 아이노드 블록 포인터 재설정

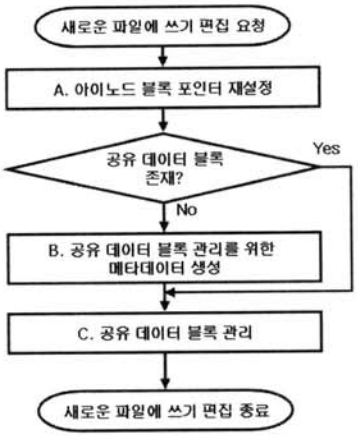


(그림 10) 원본 덮어 쓰기 편집 요청에 대한 동작 절차

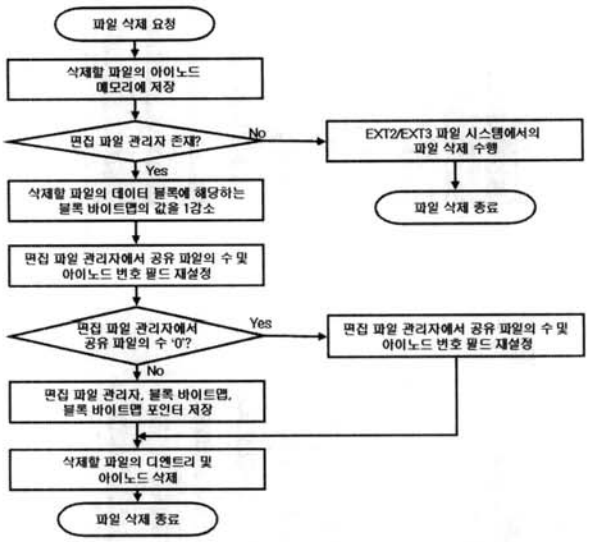
작업이 끝난 후, 원본 파일이 공유 데이터 블록을 가지는지 검사한다. 공유 데이터 블록을 가지는지 여부에 따라 편집 동작이 달라진다. 공유 데이터 블록을 가지는지 여부는 편집 파일 관리자(EFM)의 유무를 통해 알 수 있으며, 원본 파일의 아이노드 구조체에서 i_reserved[0] 필드 값을 통해 알 수 있다. 만약 i_reserved[0] 값이 0일 경우, EFM이 생성되지 않은 상태이고 공유 데이터 블록을 가지지 않는다는 의미이고, i_reserved[0] 값이 0이 아닌 다른 값을 가진다면 그 값은 EFM이 저장된 블록 주소를 나타내고, 공유 데이터 블록을 가진다는 의미이다.

공유 데이터 블록을 가지지 않을 경우 동작에 대해 먼저 살펴본다. 이 경우에 데이터 블록 관리는 Ext3 파일 시스템의 블록 비트맵을 통해 이루어진다. 다음으로 블록 비트맵 관리에 대해 알아본다. 먼저 원본 파일이 가진 데이터 블록에 대해 해당 블록 비트맵 값을 모두 0으로 만든다. 재 저장될 아이노드를 참조하여 필요한 데이터 블록에 대해 해당 블록 비트맵 값을 1로 만든다. 마지막으로 수정된 아이노드 정보를 저장한다. 만약 공유 데이터 블록을 가진다면, 데이터 블록 관리는 공유 데이터 블록 관리를 위한 메타데이터를 통해 이루어지며, 앞서 설명한 공유 데이터 블록 관리 모듈을 통해 동작한다.

둘째, 편집된 내용을 새로운 파일에 쓸 경우 편집 동작 절차를 (그림 11)에 나타내었다. 편집된 내용을 새로운 파일에 쓴다는 것은 원본 파일 중 필요한 부분을 새로운 파일을 생성하여, 새로 만들어진 파일에 그 내용을 저장한다는 것을 의미한다. 편집된 내용을 새로운 파일에 쓸 경우 편집 동작은 원본 덮어쓰기 편집 동작에 비해 간단하다. 전체적인 동작과정은 원본 덮어 쓰기 동작을 절차와 유사하며, 공유 데이터 블록 관리를 위한 메타 데이터 생성 및 관리에 있어 차이가 있다. 먼저 아이노드 블록 포인터 재설정 모듈을 통해 필요한 데이터 블록을 새로운 파일의 아이노드 블록 포인터 배열에 저장한다. 공유 데이터 블록 관리를 위한 메타데이터가 생성되어 있지 않다면, 공유 데이터 블록 관리를 위한 메타데이터를 생성한다. 공유 데이터 블록 관리를 위한 메타데이터가 생성되어 있다면, 공유 데이터 관리 모듈을 수행한다. 이와 같은 과정을 통해 새로운 파일에 편집된 내용을 쓴다. 공유 데이터 블록 관리를 위한 메타데이터 생성과



(그림 11) 새로운 파일에 쓰기 요청에 대한 동작 절차



(그림 12) 파일 삭제 요청에 대한 동작 절차

공유 데이터 블록 관리 모듈은 3.3절에 기술되어 있다.

마지막으로 파일 삭제 요청에 대한 동작 절차를 (그림 12)에 나타내었다. 제안 기법은 여러 파일이 데이터 블록을 공유하여 사용할 수 있다. 기존 Ext3 파일 시스템의 삭제 방법으로는 공유 데이터 블록에 대한 처리를 할 수 없다. Ext3 파일 시스템은 파일 삭제 시 해당 파일이 가진 데이터 블록에 대해 블록 비트맵에서 그 값을 0으로 만들어 사용 가능한 블록으로 만든다. 이 경우, 삭제하려는 파일이 공유 데이터 블록을 가지고 있으면 공유 데이터 블록에 대한 블록 비트맵 또한 0이 된다. 그리고 파일을 생성해서 데이터를 저장할 경우, 공유 데이터 블록은 사용 가능한 블록이므로 그 블록에 새로운 데이터를 기록할 수 있다. 이 경우, 실제 공유 데이터 블록을 참조하는 다른 파일은 잘못된 데이터를 가지게 된다. 그러므로 공유 데이터 블록 처리를 위한 수정된 삭제 방법이 필요하다.

수정된 파일 삭제 동작은 삭제하려는 파일이 공유 데이터 블록을 가질 경우, SBM에서 해당 데이터 블록 공유 횟수 값을 1 감소시켜주고, 그 값이 0일 경우에 해당 블록에 대한 블록 비트맵을 0으로 만든다. 만약 삭제하려는 파일이 공유 데이터 블록을 가지지 않을 경우, 기존 Ext3 파일 시스템의 삭제 방법으로 파일 삭제가 가능하다. 공유 데이터 블록을 가지는 파일을 삭제한 후에 삭제한 파일과 데이터 블록을 공유하던 파일이 더 이상 공유 데이터 블록을 가지지 않을 경우 공유 데이터 블록 관리를 위한 메타데이터인 EFM, SBM, 그리고 SBMP를 삭제해준다. 해당 파일이 더 이상 공유 데이터 블록을 가지지 않는다는 것은 EFM의 공유 파일 수 필드를 통해 알 수 있다.

4. 성능 평가 및 분석

디지털 TV에서 녹화한 영상물을 휴대용 USB 디스크로 복사하여 사용자의 윈도우즈 기반 PC에서 편집하여 저장할 수 있는 기능이 최근 제공되고 있다[15]. 대부분의 디지털 TV에서는 리눅스 운영체제를 사용하고 있으며, 따라서 녹화물도 리눅스 Ext3 파일 시스템으로 포맷된 USB에 저장할 수 있는 기능이 제공된다. 이러한 리눅스 Ext3 파일 시스템 상에 저장된 녹화물을 윈도우즈 기반 PC상에서 동작하는 재생 및 편집 프로그램에서 읽고 쓰도록 하는 기능이 소수에 한해 부가적으로 제공되고 있으며[15], 녹화한 영상물을 PC 상에서 시청하고자 하는 사용자 요구가 증가하고 있다. 본 연구에서는 이러한 사용자 환경을 가정하였으며, 따라서 제안 기법의 성능을 시험하기 위해서 윈도우즈XP 환경의 사용자 레벨에서 동작하는 프로그램 형태로 Ext3 파일 시스템을 구현하고 제안된 기법을 추가적으로 구현하였다. 윈도우즈XP 운영체제에서 사용자 레벨의 Ext3 파일 시스템 형태로 구현하였기 때문에 (그림 1)에서 제시된 새로운 VFS API는 사용자 레벨에서 사용할 수 있는 라이브러리 형태로 구현하였다.

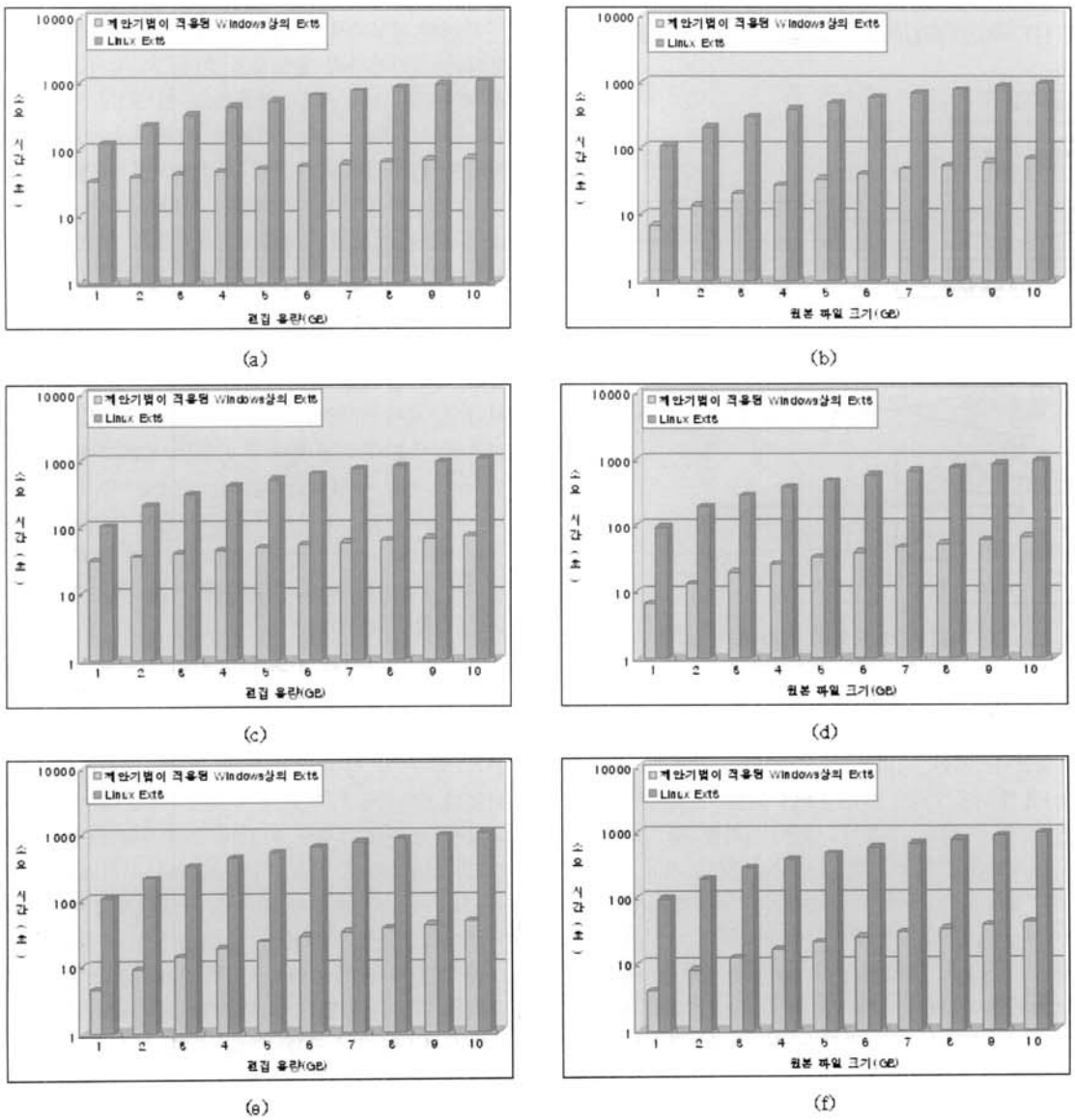
성능 관점에서는 윈도우즈 사용자 레벨에서 응용프로그램으로 동작하는 Ext3 파일 시스템의 기본적인 읽기 및 쓰기 성능은 리눅스 커널 레벨에서 동작하는 Ext3 파일 시스템의 읽기 및 쓰기 성능에 비해 약 1.5배 정도 낮은 것으로 측정되었다. 즉, 제안 기법이 리눅스 커널 내에 직접 포함되어 구현될 경우의 성능은 아래에서 제시하는 성능보다 더 높을 것으로 예상된다. 성능 비교 대상인 실제 Ext3 파일 시스템은 커널 2.6.18 버전의 리눅스 운영체제에서 동작하며, 다음과 같은 동일한 하드웨어 실험 환경을 이용한다.

실험 환경은 인텔 Core2 CPU 1.86GHz * 2, DDR2 1GB 메모리를 가진다. USB 2.0 환경에서 Calmee MOON FKL-GP3 230GB USB 디스크를 Ext3 파일 시스템으로 포맷한 뒤에 대용량 멀티미디어 파일을 복사한 후, 제안기법이 적용된 윈도우즈 환경의 Ext3(이하 제안기법)와 실제 리눅스 Ext3 파일 시스템에서 파일 편집하고 저장할 경우의 성능을 비교한다. 성능 측정 요소로 편집된 파일이 저장될 때 소요되는 시간과 사용되는 디스크 공간의 크기를 이용하였다. 또한, 부가적으로 수정되어 저장된 파일을 삭제할 때 소요되는 시간도 측정하여 비교하였다.

4.1 편집된 파일 저장 시 소요시간 비교

파일 편집 후 저장 시에 Ext3 파일 시스템은 편집된 내용을 메모리로 읽어온 후, 모든 내용을 디스크에 다시 기록한다. 제안 기법은 전체 데이터에서 불필요한 데이터 블록을 제외하고 필요한 데이터 블록만을 아이노드에 재 기록한다. 편집 후 저장할 데이터가 많을수록 Ext3 파일 시스템과 본 제안기법에서 파일 저장 시 소요되는 시간은 양쪽 모두 증가한다. Ext3 파일 시스템은 데이터를 새로 디스크에 모두 쓰는 반면에 제안기법은 실제 데이터 I/O 없이 메타데이터만을 재설정한다는 점에서 용량이 증가함에 따라 저장 시 소요되는 시간 차이는 더욱 커진다. 편집한 대용량 파일을 저장하는데 소요되는 시간 비교 실험은 편집한 내용을 원본 파일에 덮어쓸 경우와 새로운 파일에 쓸 경우로 나누어 수행한다. 각 기능에 대해 원본 파일 용량을 일정하게 고정시키고 편집하여 저장할 용량을 변화시켜가며 소요 시간을 측정한다. 그리고 원본 파일 용량을 변화시켜가며, 원본 파일 용량의 90%를 편집하여 저장할 때 소요 시간을 측정한다. 이 실험을 통해 원본 동영상 파일 크기에 따른 파일 편집 후 저장 시 성능을 알 수 있다. 편집하여 저장하는 비율을 90%로 설정한 이유는 디지털 TV에서 녹화된 영상에서 광고 등 불필요한 부분이 10%정도 포함된다고 가정했기 때문이다.

(그림 13)은 편집 후 저장 시 소모되는 시간 비교 결과를



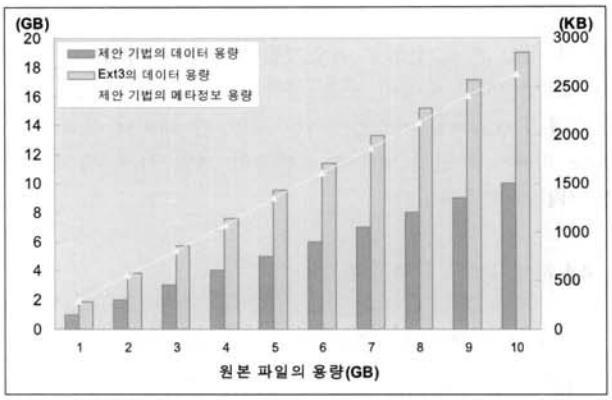
(a) 원본 파일을 10GB로 고정시켜놓고 원본 덮어쓰기를 하였을 때 소요시간 비교
(b) 원본 파일 크기의 90%만을 편집하여 원본 덮어쓰기를 하였을 때 소요시간 비교
(c) 제안기법이 공유 데이터 블록을 가지지 않을 때, 원본 파일을 10GB로 고정시켜놓고 새로운 파일에 저장할 때 소요시간 비교
(d) 제안기법이 공유 데이터 블록을 가지지 않을 때 원본 파일의 크기의 90%만을 편집하여 새로운 파일에 저장할 때 소요시간 비교
(e) 제안기법이 공유 데이터 블록을 가질 때, 원본 파일을 10GB로 고정시켜놓고 새로운 파일에 편집한 내용을 저장할 때 소요시간 비교
(f) 제안기법이 공유 데이터 블록을 가질 때 원본 파일의 크기의 90%만을 편집하여 새로운 파일에 저장할 때 소요시간 비교

(그림 13) 파일 편집 후 저장 시 제안기법과 Ext3 파일 시스템의 소요 시간 비교

보여주고 있다. 편집 후 저장 시 소모되는 시간은 모든 실험결과에 대해 제안기법이 적용된 윈도우즈 상의 Ext3가 리눅스 Ext3 파일 시스템보다 평균 16배 우수한 성능을 보였다. (그림 13) (a)와 (b)는 원본 덮어쓰기 시 소요되는 시간 비교 결과이다. 공유 데이터 블록을 가지지 않는 파일에 대해 편집한 내용을 원본에 덮어쓸 경우, 원본 파일이 가진 데이터 블록의 블록 비트맵을 0으로 설정해 주어야 하므로 부가적인 시간이 소모되었다. 부가적인 시간은 파일의 삭제 시간만큼 소모된다. 이러한 삭제 작업은 Ext3 파일 시스템과 제안기법을 적용한 윈도우즈 상의 Ext3 양쪽 모두 발생한다. 제안기법의 삭제 성능이 Ext3 파일 시스템에 비해 좋지 않는데도 불구하고 소요시간이 평균 14배 정도 단축되었다. (그림 13) (c)와 (d)는 공유데이터 블록 관리를 위한 메타데이터가 생성되어 있지 않을 때, 편집한 내용을 새로운 파일에 저장할 경우 소요되는 시간 비교 결과이다. 제안기법을 적용한 윈도우즈 상의 Ext3가 파일 편집 후 저장하는데 소요하는 시간은 (그림 13) (a)와 (b)와 유사하다. 새로운 메타데이터인 SBM을 만드는 작업이 삭제 작업과 같이 해당 파일의 데이터 블록이 속한 그룹의 블록 비트맵을 스캔해야 하기 때문이다. 공유 블록을 위한 메타데이터 생성 시간은 편집 후 저장될 파일 용량과 관계없이 원본 파일 용량에 비례한다. 반면 Ext3 파일 시스템은 부가적으로 원본 파일의 삭제 작업을 수행하지 않아도 되기 때문에 삭제 시간만큼의 시간을 절약할 수 있다. 제안기법이 Ext3 파일 시스템에 비해 소요시간이 평균 13배 정도 단축되었다. (그림 13) (e)와 (f)는 공유데이터 블록 관리를 위한 메타 데이터가 생성되어 있고, 편집한 내용을 새로운 파일에 저장할 경우 소요되는 시간 비교 결과이다. 부가적인 삭제 작업 및 메타데이터 생성 작업을 할 필요가 없기 때문에 제안기법의 성능이 제일 우수함을 볼 수 있다. 그 결과 제안기법을 적용한 윈도우즈 상의 Ext3가 리눅스 Ext3 파일 시스템에 비해 소요시간이 평균 23배 정도 단축되었다.

4.2 디스크 소모 용량 비교

(그림 14)는 원본 파일의 내용을 편집하여 새로운 파일에 저장했을 때, 제안기법이 적용된 윈도우즈 상의 Ext3와 리눅스 Ext3 파일 시스템이 원본 파일과 새로운 파일에 대해 사용하는 디스크 용량 비교를 나타낸다. Ext3 파일 시스템에서 새로운 파일에 편집한 내용을 저장할 경우, 바뀐 모든 데이터 내용을 디스크에 다시 기록한다. 하지만 제안기법의 경우에 원본 파일에 속하는 데이터 블록을 새로운 파일이 공유하여 사용한다. 데이터 블록을 공유하여 사용하기 위해 공유 데이터 블록 관리를 위한 메타데이터 용량이 부가적으로 소모되지만 그 용량은 원본 파일의 용량의 0.03%정도에 불과하다. 공유 데이터 블록 관리를 위한 메타데이터 용량은 원본 파일의 데이터 블록이 속한 그룹의 수에 비례한다. 제안기법이 사용하는 디스크 용량은 데이터 블록을 공유하는 파일의 수가 많아짐에 따라 Ext3 파일 시스템에서 사용하는 디스크 용량에 비해 그 사용량이 줄어든다.



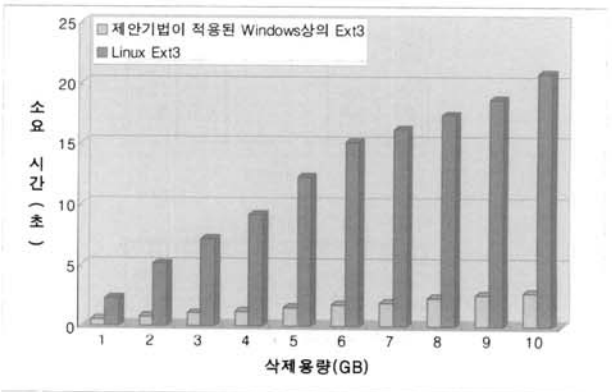
(그림 14) 새로운 파일 쓰기 시 용량 비교

4.3 편집 저장된 파일 삭제 시 소요시간 비교

편집 저장된 파일 삭제 시 성능 비교 결과는 (그림 15)와 같다. 실험은 제안기법에서 공유 데이터 블록을 가지는 파일에 대해 삭제를 하였을 때 소요되는 시간과 Ext3 파일 시스템에서 파일을 삭제할 때 소요되는 시간을 파일의 용량에 따라 비교하였다. 파일 삭제 시 평균 6.5배의 시간 단축이 있었으며, 성능 향상 원인은 다음과 같다.

첫째, 디스크 접근 시간을 줄일 수 있다. Ext3 파일 시스템에서는 파일 삭제 시 파일의 데이터 블록이 속한 그룹의 블록 비트맵에서 해당 데이터 블록의 비트맵 값을 1에서 0으로 만들어 준다. 삭제 파일이 클 경우에 해당 그룹의 블록 비트맵은 디스크에서 분산되어 저장되어 있기 때문에 해당 디스크 블록에 접근하는데 걸리는 시간이 길어진다. 하지만 제안기법은 공유 데이터 블록을 가진 파일 삭제 시, SBM에 접근하여 데이터 블록에 해당하는 공유 횟수 값을 1만 감소시키면 된다. SBM은 원본 파일의 데이터 블록과 인접한 연속된 블록에 할당되기 때문에 접근 시간이 짧다.

둘째, 파일 삭제 시 연산을 줄일 수 있다. Ext3 파일 시스템에서는 파일 삭제 시 파일의 데이터 블록이 삭제 될 때마다 슈퍼블록의 free_block_count 값과, 그룹 디스크립터의 free_block_count 값을 각각 1씩 증가시켜주어야만 했다. 하



(그림 15) 파일 삭제 시 소요시간 비교

지만 제안기법은 위와 같은 연산을 필요로 하지 않는다. 삭제 시 성능 비교 실험은 제안기법의 경우에 삭제되는 파일의 모든 데이터 블록이 공유 블록이라는 가정 하에 이루어졌으며, Ext3 파일 시스템에서는 파일 간 데이터 블록의 공유를 지원하지 않기 때문에 일반적인 파일 삭제 시 소요되는 시간을 측정하였다.

4.4 제안기법의 문제점 분석

4.4.1 편집 저장 시 발생하는 블록 내 데이터 손실

제안 기법의 아이노드 블록 포인터 재설정 모듈은 블록 단위 편집을 하기 때문에 선택된 편집 구간을 Ext3 파일 시스템의 데이터 블록 크기인 4KB 단위로 재설정 한다. 이때 4KB 미만의 데이터 손실을 가지고 올 수 있다. 따라서 본 절에서는 4KB 미만의 데이터 손실이 멀티미디어 파일에 미치는 영향에 대해 살펴본다. HD급 화질을 가지는 멀티미디어 파일은 MPEG2 형식으로 인코딩되어 있다. MPEG2에서 규정하는 프래임 형태는 I, B, P 픽처 세가지 형태가 있다. I(Intra) 픽처는 다른 픽처들의 참조 픽처로서의 역할을 하며 움직임 보상을 사용하지 않기 때문에 시간상으로 문제되는 오류를 방지할 수 있다. P(Predictive) 픽처는 가장 최근의 I 픽처나 P 픽처를 참조하여 움직임 추정 기술을 사용해서 부호화 한 것이다. B(Bidirectionally predictive) 픽처는 쌍방향 예측 부호화 영상으로 순방향 예측뿐만 아니라 역방향 예측가지 포함한 픽처이다. 4KB 미만의 데이터 손실로 인해 I, P 픽처가 손실을 입으면, 다음 I 픽처가 나올때 까지 영향을 미친다. 이때 데이터 손실로 인해 화면 일그러짐 현상등이 나타난다. 화면 변화가 심한 동영상일 경우 손실로 인해 미치는 영향이 시간적으로 짧으며, 움직임이 거의 없는 영상의 경우 미치는 영향이 시간적으로 길다. 하지만 일반적으로 MPEG2에서는 픽처 에러로 인한 파급효과를 줄이기 위해 초당 2개 이상의 I 픽처를 가지며, 그 영향은 1초를 넘기지 못한다[16, 17].

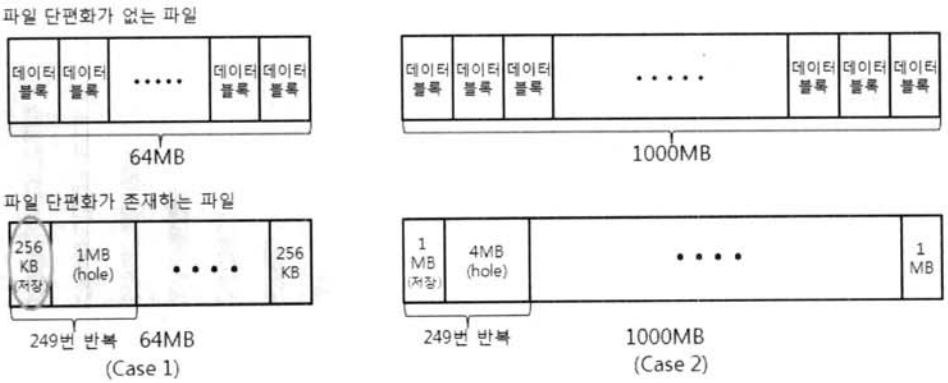
4.4.2 공유 데이터 블록을 위한 메타데이터 저장에 따른 부가 디스크 공간

제안 기법은 공유 데이터 블록 관리를 위해 부가적인 메타데이터를 생성한다. 부가적인 메타데이터는 파일 편집 저장 시 새로운 파일에 쓸 경우 생성되며, 용량은 원본 파일의 데이터 블록이 속한 그룹의 수에 비례한다. (그림 14)는 새로운 파일 쓰기 시 제안 기법이 적용된 Ext3와 Ext3의 사용하는 디스크 용량 비교를 나타낸다. 공유 데이터 블록 관리를 위한 부가적인 메타데이터는 원본 파일 크기에 비례하는 것을 알 수 있고 그 사용량은 원본 파일 크기의 0.03%에 불과하다.

4.4.3 아이노드 블록 포인터 재설정에 따른 파일 단편화 현상

제안 기법은 파일 편집 시 아이노드 블록 포인터 재설정을 통해 실제 데이터의 이동 없이 메타데이터 수정만을 통해 빠른 편집을 제공한다. 하지만 이로 인해 파일 단편화 현상을 일으키고, 디스크 검색 시간을 부가적으로 일으켜, 파일 읽기 성능 저하를 일으킨다. 따라서 본 절에서는 파일 단편화에 따른 읽기 성능을 측정하여, 멀티미디어 파일 재생에 미치는 영향에 대해 설명한다.

(그림 16)은 파일 단편화에 따른 성능 하락 정도를 측정하기 위한 실험에 사용할 파일을 나타낸다. 최악의 상황을 고려해 249번의 단편화가 발생한 파일을 생성하였고, case 1과 case 2와 같이 연속된 저장 공간의 크기를 변화시켜가며 파일 읽기 성능을 측정하였다. case 1과 같이 단편화가 일어난 파일에서 연속적인 데이터 저장 영역이 256KB일 때, 35%의 읽기 성능 저하가 있었으며, 그 속도는 13.06MB/s였다. case 2와 같이 단편화가 일어난 파일에서 연속적인 데이터 저장 영역이 1MB일 때, 3.7%의 읽기 성능 저하가 있었으며, 그 속도는 26.74MB/s였다. 실험 결과를 통하여 파일 단편화가 있더라도 연속적인 데이터 저장 역역의 크기가 크면 클수록 파일 단편화에 따른 성능 저하의 영향을 적게 받는다는 것을 알 수 있었다. 또한 제안 기법이 대상으로 하는 멀티미디어 파일 편집 시 연속적인 데이터 저장 영역의 크기는 편집 형태에 따라 달라지지만, 최소 수 MB에서 수 GB로 실험에서 보여준 연속적인 데이터 저장 영역보다 더 크다. 따라서 파일 단편화로 인한 성능 저하의 영향을 적게 받는다. 또한 최악의 경우, 연속적인 데이터 저장 영역이 case 1과 같이 작다고 하더라도 13.06MB/s의 읽기 성능을



(그림 16) 파일 단편화 영향을 알아보기 위한 비교 파일 예

가진 단편화 파일은 2.5MB/s의 재생 속도를 충분히 만족시킬 수 있다.

5. 결론 및 향후 연구계획

멀티미디어 영상물에 대한 사용자의 수요가 늘어남에 따라서 멀티미디어 파일 재생 및 멀티미디어 파일 처리에 있어 다양한 서비스가 제공되고 있다. 하지만 임베디드 장치의 파일 시스템으로 널리 사용되는 기존 Ext3 파일 시스템은 대용량 파일 편집 시 시간이 많이 걸리고 디스크 I/O가 많은 문제를 가지고 있다. 본 논문에서는 기존 Ext3 파일 시스템의 문제를 해결하기 위해 대용량 파일을 기본단위로 하는 시스템에서 편집으로 인한 부분적인 삭제가 발생할 때 빠르고 효율적인 편집 및 저장 방법을 제공해준다. 이를 위해 데이터 I/O 없이 CPU 연산만으로 파일 편집을 할 수 있는 아이노드 블록 포인터 재설정 기법과 데이터 블록을 공유하여 사용할 수 있는 공유 데이터 블록 관리 기법을 제안하고 기존 Ext3 파일 시스템에 통합 구현하였다. 성능측정 결과 제안기법이 적용된 Ext3 (사용자 수준의) 파일 시스템이 기존 리눅스 Ext3 파일 시스템보다 파일 편집 후 저장 시에 소요되는 시간 측면에서 평균 16배가량 우수한 성능을 보였다. 또한, 실제 사용하는 디스크 사용량에서도 공유 데이터 블록 사용으로 인해 절약 가능하다는 것을 알 수 있었다. 제안기법은 멀티미디어 대용량 파일을 기본단위로 하는 임베디드 시스템이나 일반 PC의 파일 시스템에 적용하여 대용량 멀티미디어 파일의 편집 후 저장 시 유용하게 사용될 수 있을 것으로 기대한다.

향후 연구계획으로는 제안한 기법을 리눅스 Ext2 파일 시스템에 구현한다. 최근 들어 디지털 카메라 내에 포함된 플래시 메모리의 크기가 대용량화 되면서 고품질의 동영상 촬영 및 저장이 가능하게 되었다. 또한, 디지털 카메라 자체 동영상 편집 프로그램을 이용하여 저장된 동영상에 대한 가공도 가능하게 되었다. 본 연구에서 제기한 편집파일 저장 시 발생하는 성능상의 문제는 쓰기 성능이 읽기에 비해서 상대적으로 낮은 플래시 메모리에서 더욱더 심각해진다는 것을 확인하였다. 현재 본 연구의 후속으로 제안기법에 적용된 아이디어를 낸드 플래시 메모리를 저장장치로 하는 YAFFS[16], CFFS[17]와 같은 낸드 플래시 전용 파일 시스템에 적용하는 연구를 진행하고 있다.

참 고 문 헌

[1] S. Y. Lim, J. H. Choi, J. M. Seok, and H. K. Lee, "Advanced PVR architecture with segment-based time-shift," International Conf. on Consumer Electronics, pp.1-2, Jan., 2007.

[2] Y. Ninomiya, "High definition television systems," Proc. IEEE, Vol.83, No.7, pp.1086-1093, Jul., 1995.

[3] M. Sadiku and S. Nelatury, "High definition television in detail," IEEE potentials, Vol.26, No.1, pp.31-35, Jan.-Feb., 2007.

[4] L. Zong and N. Bourbakis, "Digital video and digital TV: a comparison and the future directions," Proc. 1999 International Conf. on Information Intelligence and Systems, pp.470-481, Nov., 1999.

[5] Avidemux, http://fixounet.free.fr/avidemux/.

[6] D. Bovet and M. Cesati, Understanding the LINUX KERNEL. 3rd edition, O'Reilly, pp.738-774, 2006.

[7] M. McKusick, W. Joy, S. Leffler, and R. Fabry, "A fast file system for UNIX," ACM Trans. Computer Systems, Vol.2, No.3, pp.181-197, 1984.

[8] T. Ts'o, "Planned extensions to the Linux Ext2/Ext3 filesystem," Proc. of the 2002 USENIX Ann. Technical Conf., pp.235-243, 2002.

[9] "Whitepaper: Red Hat's new journaling file system: ext3," http://www.redhat.com/support/wpapers/redhat/ext3/ext3.pdf

[10] K. Preslan et al., "A 64-bit, shared disk file system for linux," Proc. of the 16th IEEE Mass Storage Systems Symp., pp.22-41, Mar., 1999.

[11] Y. K. Lee, S. W. Kim, G. B. Kim, and B .J. Shin, "Metadata management of the SANtopia file system," Proc. 8th International Conf., pp.492-499, Jun., 2001.

[12] 신용주, 김정원, 김영주, 정기동, "대용량 파일들의 편집, 저장을 위한 효율적인 파일 관리 기법", 한국정보과학회 가을 학술발표논문집, Vol.55, No.7, 1997.

[13] J. Y. Hwang, J. K. Bae, A. Kirnasov, M. S. Jang, and H. Y. Kim, "A reliable and portable multimedia file system," Proc. of the Linux Symp., Jul., 2006.

[14] Z. Peterson and P. Burns, "Ext3cow: A time-shifting file system for regulatory compliance," ACM Trans. Storage, Vol.1, No.2, pp.190-212, 2005.

[15] LG Electronics, "Xcanvas Player," http://xcanvas.co.kr.

[16] P. N. Tudor, "MPEG-2 video compression," IEE journal, Vol.7, pp.257-264, Dec., 1995.

[17] 이호석, 김준기, "알기 쉬운 MPEG-2", 홍릉과학출판사, 2002.

[18] S. H. Lim and K. H. Park, "An efficient NAND flash file system for flash memory storage," IEEE Trans, on Computers, Vol.55, No.7, Jul., 2006.

[19] Aleph One Ltd, Embedded Debian, "Yaffs: A NAND-flash filesystem," http://www.aleph1.co.uk/yaffs, 2002.

정 승 완

e-mail : tmdrod@ee.knu.ac.kr

2006년 경북대학교 전자전기컴퓨터학부(학사)

2008년 경북대학교 전자전기컴퓨터학부(공학석사)

2008년~현 재 경북대학교 전자전기컴퓨터학부 박사과정

관심분야 : 임베디드 SW, 파일 시스템, 센서 네트워크 등

남 영 진

e-mail : yjnam@daegu.ac.kr

1992년 경북대학교 전자공학과(학사)

1994년 포항공과대학교 전자전기공학과(공학석사)

2004년 포항공과대학교 컴퓨터공학과(공학박사)

1994년~1998년 한국전자통신연구원 컴퓨터연구단

2004년~현 재 대구대학교 컴퓨터IT공학부 조교수

관심분야 : 네트워크 스토리지, 임베디드 SW, 무선 네트워크 등

서 대 화

e-mail : dwseo@ee.knu.ac.kr

1981년 경북대학교 전자공학과(학사)

1983년 한국과학기술원 전산학과(공학석사)

1993년 한국과학기술원 전산학과(공학박사)

1983년~1995년 한국전자통신연구원 컴퓨터 S/W 연구실

2004년~현 재 경북대학교 임베디드소프트웨어 연구센터장

1998년~현 재 경북대학교 전자전기컴퓨터학부 교수

관심분야 : 임베디드 SW, 병렬처리, 분산운영체제 등