DOI: 10.3745/KIPSTA.2010.17A.2.063

대용량 멀티미디어 파일 고속 편집저장을 지원하는 블록 링크 파일 시스템 설계 및 구현

정 승 완[†] · 고 석 영[††] · 남 영 진[†††] · 서 대 화[††††]

요 약

디지털 기술의 발달로 디지털 TV, 캠코더와 같은 멀티미디어 장치의 사용이 증가하고 있다. 이러한 장치들은 멀티미디어 파일 재생 및 녹화, 편집 등의 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 앞서 언급한 문제를 해결하기 위한 기존 연구인 FWAE (Fast Writing after Editing) 기법보다 우수한 성능을 가진 리눅스 Ext2 파일 시스템 기반의 블록 링크 파일 시스템을 제안한다. 블록 링크 파일 시스템은 대용량 파일 편집저장 시, 실제 데이터의 입출력 없이, 메타데이터만을 수정한다. 또한, 멀티미디어 파일 간, 데이터 블록을 공유함으로써, 디스크 사용량을 줄일 수 있으며, 데이터 블록 공유 정보 관리가 기존 FWAE 기법보다 간단하여 시스템 오버헤드를 줄일 수 있다. 제안하는 파일 시스템은 Ext2 파일 시스템을 기반으로 하여 리눅스 환경에 구현하였으며, 유용성은 리눅스 Avidemux 편집 툴에 적용하여 검증하였다.

키워드 : 파일 편집, 멀티미디어 파일, 데이터 블록 공유

Implementation of a Block Link File System Supporting Fast Editing/Writing for Large-sized Multimedia Files

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

ABSTRACT

Recently, the need for multimedia devices, such as digital TV, and camcorder has increased. These devices provide many services on multimedia files such as playing, recording, and editing. Throughout these services, in case of storing edited large-scaled multimedia files, existing file system have several capability problems that are taking too much time and requiring disk I/O. In this paper, we propose the use of Linux Ext2 file system based 'Block-Link file (BL-file) system' in order to solve these problems. For the BL-file system, when editing and storing large-scaled files, there is no data input or output but only modification of the metadata. Additionally, by sharing data blocks between multimedia files, we can save disk spaces. Moreover, because the managing of data block sharing information is more convenient than the existing FWAE technique, we can lessen system overhead. The BL-file system is based on Ext2 file system and implemented in a Linux environment, and the usefulness of the proposed technique is validated by applying the Linux multimedia file-editing tool 'Avidemux'.

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 사용자가 녹화물을 시청하면서 원하는 부분만을 선택적으로 저장할 수 있는 편집 기능을 제공한다. 이와 같은 편집 기능은 녹화한 영상물에서 광고와 같은 불필요한 부분 및 원하지 않는 부분을 삭제하는데 있어 효과적이다. 하지만 삭제 편집 외에 이미지 삽입 및 이미지 변경과 같은 편집 기능은 제공하지 않는다. 이미지 삽입 및 이미지 변경을 위해서는 많은 시간과 시스템 오버헤드를 야기하는 동영상 인코딩 과정을 수행해야 하기 때문이다. 따라서 복잡한 동

※ 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(NIPA-2010-C1090-1031-0009)
† 준 회 원 : 경북대학교 전자전기컴퓨터학부 박사과정
†† 정 회 원 : LIG넥스원 항공연구센터 2팀
††† 정 회 원 : 대구대학교 컴퓨터IT공학부 조교수
†††† 종신회원 : 경북대학교 IT대학 전자공학부 교수
논문접수 : 2009년 11월 17일
수 정 일 : 1차 2010년 1월 11일, 2차 2010년 2월 3일
심사완료 : 2010년 2월 3일

영상 인코딩 과정을 생략하고, 기존 영상에서 불필요한 부분 삭제를 통해 원하는 영상을 만들고자 할 때 이와 같은 편집 기법이 사용될 수 있다. 이와 같은 편집 기법은 실제 동영상 편집 툴에서도 많이 사용되고 있다[5, 20]. 미디어 컨텐츠는 다양한 화질을 가지고 있으며, 같은 내용을 녹화하더라도 고화질일수록 더 많은 용량을 차지한다[2]. 디지털 TV, 디지털 캠코더 등과 같은 멀티미디어 장치는 HD급, 풀(full) HD급의 고화질 미디어 컨텐츠를 제공한다. 풀 HD급 화질의 경우 약 20Mbps의 전송률을 가지며, 한 시간 분량의 영상물에 대해 약 9GB의 용량을 가진다[3, 4].

멀티미디어 파일 편집 과정은 (그림 1)에서와 같이 일반적으로 편집 과정과 저장 과정으로 나누어질 수 있다. 편집 과정은 사용자가 원본 멀티미디어 파일을 재생시키면서 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 원본 파일 'A'에서 C와 D부분을 삭제한 후에 새로운 파일 'B'에 저장하는 예를 보여주고 있다.

멀티미디어 장치에서 촬영 및 녹화된 멀티미디어 파일은 불필요한 부분을 삭제하기 위해 편집 과정을 거치게 된다. 하지만 HD급 고화질 멀티미디어 파일은 일반적으로 대용량의 파일이기 때문에 동영상 편집 후 편집된 내용을 다시 저장하기 위해 많은 시간과 저장 공간 소모가 발생하는 문제점이 있다.

기존 연구 FWAE (Fast Writing After Editing)[6]에서는 편집 구간의 데이터를 새로운 공간에 복사하지 않고 메타데이터를 이용해 데이터를 공유함으로써 이 문제를 개선하였다. 하지만 부가적인 공유 정보 관리를 위한 메타데이터 생성 시 발생하는 많은 시스템 오버헤드는 성능 향상을 반감시켰다. 또한 FWAE 기법은 리눅스 Ext2 파일 시스템을 윈도우 응용 프로그램 환경에서 구현하여 파일 시스템 성능이 떨어지는 최적화 문제가 있었다. 이에 본 논문에서는 FWAE를 개선한 블록 링크 파일 기반의 동영상 고속 편집저장 기법을 제안한다. 이 기법은 공유 정보 관리에서 시스템 부하를 완전히 제거할 뿐 아니라 리눅스 Ext2 파일 시스템에 구현하여 최적화 문제도 해결한다.

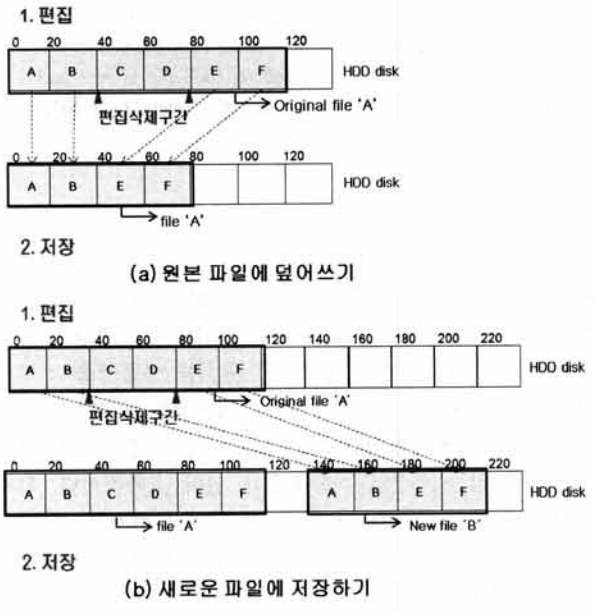
본 논문의 구성은 다음과 같다. 2장에서는 기존 동영상 파일의 편집저장 기법을 소개하고 문제점을 제시한다. 3장에서는 새로운 파일 형태인 블록 링크 파일을 제안하고, 4장에서는 블록 링크 파일 시스템과 기존의 FWAE 기법을 비교한다. 5장에서는 실험을 통해 본 논문에서 제시한 기법과 기존 Ext2 및 FWAE을 여러 측면에서 성능을 비교한다. 마지막으로 6장에서는 결론과 향후 과제를 제시한다.

2. 관련 연구

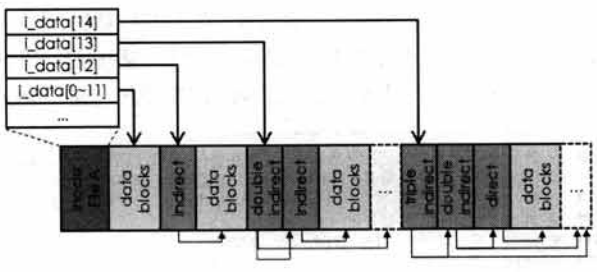
본장에서는 먼저 공유 데이터 블록과 관련된 연구에 대해 알아보고, 제안하는 블록 링크 파일 시스템의 기반이 되는 리눅스 Ext2 파일 시스템에 대해 설명하고, 기존 동영상 파일의 편집저장 기법인 VBS, XPRESS, FWAE의 특징 및 장단점을 살펴본다.

공유 데이터 블록을 지원하는 파일 시스템으로는 SAN (Storage Area Network)환경에서 사용하는 GFS(Global File System)[7]와 SANtopia[8], Venti[9], LBFS(Low-Bandwidth Network File System)[10]이 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 데이터 블록을 지원하는데 있어서는 적용하기 힘들다.

먼저, Ext2 파일 시스템[11]에서 파일 데이터 블록의 할당 방법에 대해 설명한다. (그림 2)는 Ext2의 파일 구조를 나타낸다. Ext2 파일 시스템의 파일은 파일 정보를 담고 있는 아이노드와 실제 데이터 블록의 물리 주소를 저장하고 있는 간접 블록, 실제 데이터를 담고 있는 데이터 블록들로 구성된다. 아이노드는 12개의 직접 블록과 3개의 간접 블록



(그림 1) 멀티미디어 파일 편집 과정



(그림 2) Ext2 파일 구조

들로 이루어지며, 파일의 크기가 커지면 간접 블록을 사용하여 데이터를 저장한다. Ext2 파일 시스템은 파일 편집저장 시, 편집된 내용을 저장장치의 빈 공간에 모두 새로 기록하고, 수정된 메타데이터를 기록한다[12]. 이와 같은 편집 방법은 대용량 파일 편집저장 시, 많은 데이터 I/O를 발생시켜 상당한 시간이 소요된다는 단점이 있다[13, 14].

VBS (Variable Block Size)[15], XPRESS (eXtendible Portable Reliable Embedded Storage System)[16]와 같은 파일 시스템과 FWAE 기법에서 파일 편집과 관련된 연구가 이루어졌다. 이러한 파일 시스템들은 기존 Ext2 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 메타 데이터는 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.

FWAE 기법에서는 멀티미디어 파일 편집 시, 실제 데이터 수정 없이 아이노드와 간접 블록과 같은 메타데이터만을 수정한다. 또한, 원본 멀티미디어 파일을 편집하여 새로운 파일에 저장할 경우, 원본 파일과 새로운 파일은 데이터 블록을 공유하여 사용한다. 이를 위해 공유되는 데이터 블록을 관리하기 위한 부가적인 메타데이터가 생성된다. 생성된 부가적인 메타데이터에는 공유되는 데이터 블록마다 몇 개의 파일에서 공유가 되는지 공유 횟수 정보를 저장한다. FWAE 기법은 데이터 블록 공유와 메타데이터 수정만을 통해 기존 Ext2에 비해 데이터 I/O가 급격히 감소하는 효과를 가져왔다.

하지만 공유 정보를 생성에 많은 시간이 소요되는 문제점이 있었다. 예를 들어 8GB 파일을 공유한다면, 약 200백만 개의 데이터 블록에 대한 공유 횟수 정보를 각각 생성 및 관리해야 하기 때문에 많은 시스템 오버헤드가 생기게 된다. 또한 FWAE 기법은 구현 기반이 되는 Ext2 파일 시스템을 일반 PC 유저의 응용 프로그램 접근을 용이하게 하고자 Ext2Read[21] 오픈소스를 기반으로 하여 윈도우 응용 프로그램 수준에서 구현되어 성능상의 최적화가 이루어지지 않았다.

3. 블록 링크 파일 시스템

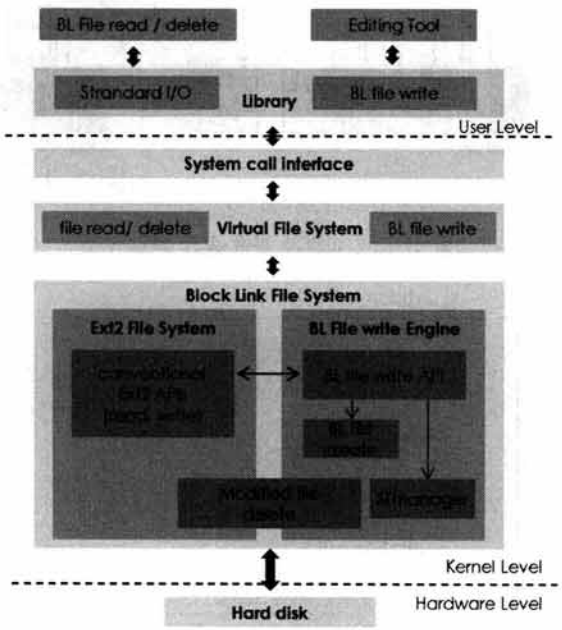
본 장에서는 블록 링크 파일 시스템의 전체 구조와 새로운 파일 형태인 블록 링크 파일, 블록 링크 파일을 이용한 대용량 멀티미디어 파일 편집저장 절차 및 블록 링크 파일 시스템 각 모듈에 대해서 기술한다.

3.1 전체 구조

(그림 3)은 블록 링크 파일 시스템의 전체 구조를 보여준다. 블록 링크 파일 시스템은 기존의 Ext2 파일 시스템에 동영상 고속 편집 기능을 가지는 BL file write Engine이 추가되었다. BL file write Engine은 고속 편집저장 기능을 제공하기 위한 BL file create 모듈과 SI manager 모듈 그리고 데이터 블록을 공유하는 파일 삭제를 위해 Modified file delete 모듈이 존재한다. BL file create 모듈은 블록 링크 파일 생성 작업을 수행하며, SI manager 모듈은 파일 간 데이터 블록 공유 정보를 관리한다. 각 모듈에 대한 자세한 설명은 다음 장에서 한다.

동영상 고속 편집 기능을 제공하는 BL file write API를 유저 레벨에서 사용하기 위해 BL file write 라이브러리 함수, 시스템 호출, 가상 파일 시스템 함수를 추가하였다. 유저 레벨에서 요청된 BL file write 라이브러리 함수는 시스템 호출과 가상 파일 시스템을 거쳐 블록 링크 파일 시스템에 구현된 BL file write API로 연결된다.

블록 링크 파일의 읽기와 삭제는 기존 라이브러리 함수를 그대로 사용한다. 하지만 블록 링크 파일 삭제 시에는, 기존 Ext2 파일 시스템의 파일 삭제 함수인 ext2_delete_inode()에 블록 링크 파일 삭제 기능이 추가된 Modified file delete 모듈이 수행된다.



(그림 3) 블록 링크 파일 시스템 구조

3.2 블록 링크 파일

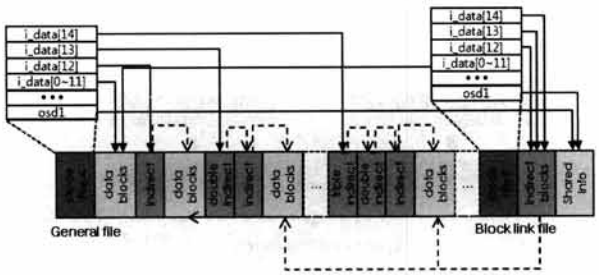
본 장에서는 대용량 멀티미디어 파일의 고속 편집을 위해서 블록 링크 파일 시스템에서 사용하는 새로운 파일 형태인 블록 링크 파일(Block Link File)에 대해 설명한다.

대용량 멀티미디어 파일 편집 시, 편집된 블록 링크 파일로 생성된다. 블록 링크 파일은 편집 대상 파일의 데이터 블록을 공유하고 데이터 블록들의 주소 정보만 가지고 있다.

블록 링크 파일은 (그림 4)와 같이 아이노드, 간접 블록, 공유정보(Shared Information, SI) 등의 메타데이터로 구성된다. 블록 링크 파일은 실제 데이터 블록을 가지지 않는다. 다만 편집 대상 파일과 공유하는 데이터 블록의 주소 정보를 아이노드와 간접 블록에 저장한다. 그리고 새로운 메타데이터인 SI는 원본 파일의 데이터 블록 공유 상태를 나타내고 원본 파일이 다른 파일과 데이터 블록을 처음 공유할 때 생성된다.

(그림 5)는 SI 구조체를 보여준다. SI 구조체는 원본 파일의 아이노드 번호, 총 편집 구간 개수, 각 편집 구간 정보로 구성된다.

SI 구조체에서 편집 구간 정보는 원본 파일과 블록 링크 파일이 데이터를 공유하는 구간의 시작과 끝 오프셋을 원본 파일의 논리 오프셋을 기준으로 기록된다. 편집 구간 정보를 통해 원본 파일과 블록 링크 파일들 간의 데이터 공유 구간을 알 수 있다. SI 구조체는 하나의 빈 데이터 블록을 할당 받아 저장되며, SI 구조체가 저장된 데이터 블록의 물리 주소를 원본 파일과 블록 링크 파일의 아이노드 구조체 osd1 필드에 저장한다. 아이노드 osd1 영역은 아이노드 구조체에 할당되어 있지만 어떠한 목적으로 사용되지 않는 reserved 영역이다. 이를 통해 원본 파일과 블록 링크 파일이 쉽게 SI 구조체 접근 가능하다.



(그림 4) Ext2 파일과 블록 링크 파일 구조

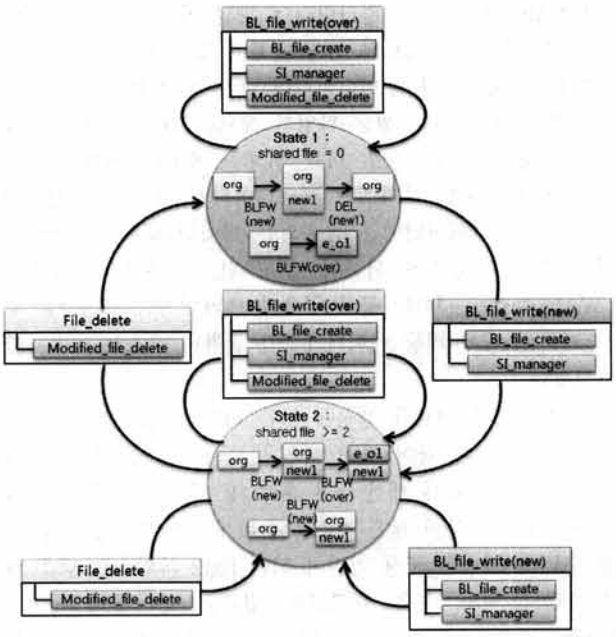
```
struct EDIT
{
    u32 i_num;
    u32 start_offset
    u32 end_offset
};
struct SHARE_INFO
{
    u16 total_edits
    u32 original_file_inum;
    struct EDIT edits[300]
};
```

(그림 5) SI 구조체

3.3 BL File write 엔진 전체 동작 과정

본 장에서는 대용량 멀티미디어 파일 편집 저장 시, 전체 동작에 대해 살펴본다. 멀티미디어 파일 편집 시, 편집 저장 방법으로는 편집한 멀티미디어 파일을 원본 파일에 덮어쓰기와 새로운 파일에 쓰기 2가지 방식이 있다. 블록 링크 파일 시스템은 2가지 편집 저장 방식에 있어 모두 빠른 편집 저장을 지원하기 위해 BL file write 엔진을 사용한다. 대용량 멀티미디어 파일 편집 시, BL file write 엔진에서의 빠른 편집 저장 동작에 대해 살펴보도록 한다.

(그림 6)은 BL file write 엔진의 전체 동작을 상태 다이어그램으로 표현한 것이다. 2가지 상태가 존재한다: 데이터 블록을 공유하는 파일이 존재하지 않는 상태1과 데이터 블록을 공유하는 상태2. 어떤 데이터도 공유하지 않는 원본 파일은 상태1에서 시작한다. 멀티미디어 파일이 BLFW (BL-File Write) 명령 후 상태1을 유지하기 위해서는 BLFW (over)(BLFW with overwrite mode) 명령을 실행한다. 이때 BLFW(over)은 BL file create 모듈, SI manager 모듈, modified file delete 모듈을 차례로 호출한다. BLFW(over) 명령은 빠른 편집 저장을 지원하기 위해 원본 파일에 덮어쓸 파일을 블록 링크 파일 형태로 생성한다. 그리고 원본 파일과 원본 파일에 덮어쓸 파일의 공유 정보를 기록한 다음 원본 파일을 삭제한다. 초기 상태에서 BLFW(new) (BLFW with new file mode) 명령을 실행할 경우, 상태1에서 상태2로 상태 전이가 일어난다. 이때 원본 멀티미디어



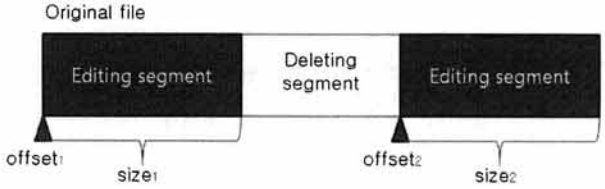
(그림 6) BLFW 엔진 동작 상태 전이 다이어그램

파일과 편집된 내용이 저장되는 새로운 파일이 데이터 블록을 서로 공유한다. BLFW(new) 명령 수행 시, BL file create 모듈과 SI manager 모듈이 차례로 호출된다. 상태2에서 BLFW(over), BLFW(new) 명령을 실행하면 파일은 여전히 상태2를 유지한다. 상태2에서 파일 삭제 명령 수행 후, 데이터 블록을 공유하는 파일이 더 이상 존재하지 않을 경우, 상태2에서 상태1로 상태 전이가 일어난다. 파일 삭제 명령 수행 시, Modified file delete 모듈을 호출한다.

3.4 BL file create 모듈

BL file create 모듈은 블록 링크 파일을 새롭게 생성할 때 실행된다. 블록 링크 파일은 BLFW 명령이 실행될 때 만들어진다. 블록 링크 파일은 앞서 3.2에서와 같이 아이노드, 간접 블록과 같은 메타데이터와 부가적인 공유정보(SI)로 구성된다. BL file create 모듈에서는 편집되어 저장되는 파일 정보에 맞게, 아이노드를 생성하여 저장한다. 또한, 원본 파일 중 불필요한 데이터 블록을 제외한 편집된 파일에 저장되는 데이터 블록만을 뽑아내어 간접 블록에 데이터 블록의 물리 주소를 기록한다. 마지막으로 공유정보(SI) 구조체를 생성하고 초기화 한다.

BL file create 모듈은 원본 파일의 아이노드 정보인 inode_src, 편집된 내용이 저장될 파일의 아이노드 정보인 inode_dest, 편집 정보를 가지고 있는 edit_info를 입력 값으로 받는다. edit_info는 편집되어 저장되는 영역의 시작 오프셋과 저장 사이즈를 하나의 셋 (set) 으로 하여 다수의 셋 (set) 으로 구성된다. (그림 7)은 edit_info에 저장되는 정보를 알기 쉽게 보여준다. 먼저 원본 파일에서 편집 구간에 해당하는 데이터 블록의 물리주소를 읽어와, 블록 링크 파일의 메타데이터에 저장한다. 이 때 원본 파일의 메타데이터에서 물리주소를 가져오고, 블록 링크 파일의 메타데이터에 저장하기 때문에 데이터 블록 복사와 관계된 I/O는 발생하지 않는다. 편집된 내용이 저장되는 파일의 데이터 블록 맵핑 작업을 마친 후, 블록 링크 파일의 구성 요소 중 하나인 공유 정보(SI)를 생성하고 그 값을 초기화한다. 생성된 공유 정보는 하나의 데이터 블록을 할당 받아 저장되며, 해당 데이터 블록의 주소값은 원본 파일과 편집된 내용이 저장되는 블록링크 파일의 아이노드 osd1 영역에 저장된다. 마지막으로 원본 파일의 아이노드 정보와 새로운 파일의 아이노드 정보 그리고 공유 정보를 디스크에 저장한다. 이와 같은 절차로 BL file create 모듈은 실행하고, 다음으로 SI manager 모듈을 실행한다.



(그림 7) edit_info에서 편집 구간

3.5 SI manager 모듈

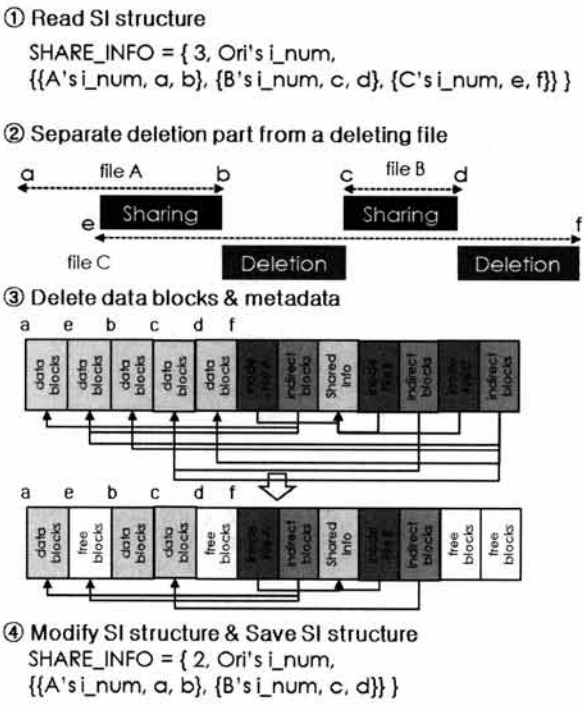
SI manager 모듈은 파일 간 공유하는 데이터 블록의 공유 구간 정보를 저장한다. (그림 7)에서와 같이 원본 파일을 편집하여, 편집한 내용을 원본 파일에 덮어쓰거나 새로운 파일에 저장할 때, editing segment에 속한 시작과 끝 데이터 블록의 논리적 블록 번호(원본 파일 기준)를 SI의 edits 배열에 기록한다. 기록된 SI를 통해, 공유 파일 삭제 시 파일 간 데이터 블록을 공유하지 않는 영역만을 선별하여 데이터 블록 삭제를 할 수 있다.

SI manager 모듈의 동작 과정은 다음과 같다. 먼저, BL file create 모듈에서 생성되었던 SI 정보를 읽어온다. 파일의 편집 정보가 담겨있는 edit_info 입력 값으로부터 SI의 edits 배열에 편집 구간(Editing segment)의 시작과 끝 논리 블록 번호를 저장한다. 수정된 SI를 디스크에 저장한다.

3.6 Modified file delete 모듈

블록 링크 파일 시스템은 기존의 Ext2 파일 시스템과 달리, 파일 간 데이터 블록 공유가 가능하다. 따라서 공유 파일 중 하나가 삭제 될 때, 삭제할 파일에 속한 데이터 블록이 공유 블록인지 아닌지 판단하여 공유 블록이 아닌 데이터 블록만을 선별적하여 삭제할 필요가 있다. 이와 같은 이유로 기존 Ext2 파일 삭제 기능을 수정한 Modified file delete 모듈이 필요하다.

Modified file delete 모듈은 먼저 삭제하려는 파일의 아이노드 osd1 필드를 검사하여 공유 파일인지 아닌지 검사한다. 만약 공유 파일이 아니라면 기존 Ext2의 파일 삭제 방법으로 파일을 삭제한다. 만약 공유 파일이라면 (그림 8)과 같은 순서로 파일을 삭제한다. (그림 8)은 파일 A, 파일 B,



(그림 8) Modified file delete 모듈 동작 예

파일 C가 데이터 블록을 공유하고 있을 때, 공유 파일 중, 파일 C를 삭제할 때의 동작 과정을 나타낸다. 먼저, 삭제하려는 파일의 아이노드 osdl 필드에 기록된 주소값에서 SI 구조체를 읽어온다. 그리고 삭제할 파일이 포함하고 있는 데이터 블록 중 다른 파일과 공유하지 않는 삭제 영역(deletion part)만을 추출한다. 삭제 영역에 속한 데이터 블록만을 삭제하고 삭제하려는 파일의 메타데이터를 삭제한다. 마지막으로 SI 구조체에서 삭제하려는 파일과 관련된 구간 정보값을 삭제하고 수정된 값을 디스크에 저장한다.

4. FWAE 기법과 블록 링크 파일 시스템 차이점 비교

본 장에서는 블록 링크 파일 시스템과 블록 링크 파일 시스템의 기반이 되는 FWAE 기법의 동작 기능 및 공유 데이터 관리 방법에 대해 비교한다. <표 1>은 FWAE 기법과 블록 링크 파일 시스템의 차이점을 비교 정리한 것이다. 먼저, FWAE 기법과 블록 링크 파일 시스템은 멀티미디어 파일 고속 편집 기능을 제공한다는 점에서 동일하다. 하지만, 공유 데이터 관리 방법과 구현 환경에 있어 그 차이가 있다. 먼저 FWAE 기법은 공유 데이터를 데이터 블록 단위로 관리한다. 따라서 공유 데이터 관리를 위한 메타데이터 생성 시, 원본 파일에 속한 모든 데이터 블록의 공유 횟수 정보를 만들어야 한다. 예를 들면, 원본 파일의 크기가 8GB일 경우, 200만개 데이터 블록의 공유 횟수 정보를 만들어야 한다. 반면 블록 링크 파일 시스템은 공유 데이터를 파일의 논리 오프셋을 이용하여 세그먼트 단위로 관리한다. 따라서 SI 구조체에 공유 세그먼트의 시작과 끝 오프셋만을 기록하면 된다. 즉, 블록 링크 파일 시스템은 FWAE 기법에 비해 공유 데이터 관리 단위를 크게 함으로써, 복잡하고 불필요한 공유 데이터 관리 연산 및 생성 과정을 없앨 수 있다.

블록 링크 파일 시스템은 리눅스 Ext2 파일 시스템에 직접 구현함으로써, 임베디드 기기의 루트 파일 시스템으로 탑재될 수 있고, 파일 시스템 차원에서 멀티미디어 파일 고속 편집 기능을 제공할 수 있다. 또한 파일의 읽기, 쓰기 성능 측면에서 윈도우즈 응용 프로그램 형태로 구현된 FWAE 기법과는 달리 최적화가 이루어져 있는 장점을 가진다.

5. 성능 평가

본 장에서는 블록 링크 파일 시스템의 BLFW 명령 수행 시 성능을 Ext2 파일 시스템과 FWAE 기법과 비교, 분석한다. 블록 링크 파일 시스템은 Linux 커널 2.6.11 버전에서 Ext2 파일 시스템을 수정하여 구현하였다. 또한 블록 링크 파일 시스템의 API를 Avidemux[5]에 적용하여 실제 멀티미디어 파일 편집 실험을 수행하였다. 고화질 대용량 멀티미디어 파일을 실험 대상으로 하기 위해 MPEG2와 H.264 압축 포맷을 가진 파일을 대상으로 실험을 수행하였다. 또한 AVI, MPEG, ASF, WMV 등 다양한 확장자 명을 가진 멀티미디어 파일을 대상으로 실험을 하였고 모두 빠른 편집 저장이 가능했다. 비교대상인 FWAE는 윈도우 기반 응용 프로그램 형태로 동작하며, 기존 Ext2 파일 시스템은 리눅스 운영체제에서 동작한다. 블록 링크 파일 시스템, FWAE와 Ext2가 동작하는 플랫폼은 2.8GHz * 2 Intel P4 CPU와 2GB 메인 메모리로 구성되어 있으며, Calmee 230GB 외장 하드 디스크를 이용하여 실험하였다. <표 2>는 실험 환경에 대해 요약하여 보여준다.

실험을 통해 멀티미디어 파일 편집 저장 성능, 편집된 파일 읽기 성능, 수정된 파일 삭제 성능, 그리고 블록 링크 파일 시스템의 문제점에 대해 평가한다.

〈표 2〉 실험 환경

	Block link	FWAE	Ext2
OS	Linux Kernel 2.6.11	Windows XP	Linux Kernel 2.6.11
CPU	Intel P4 2.8GHz * 2		
Main Memory	Samsung 2GB DDR2		
USB disk	FUJITSU Calmee MOON FKL-GP3 230GB		

〈표 1〉 FWAE 기법과 블록 링크 파일 시스템의 차이점 비교

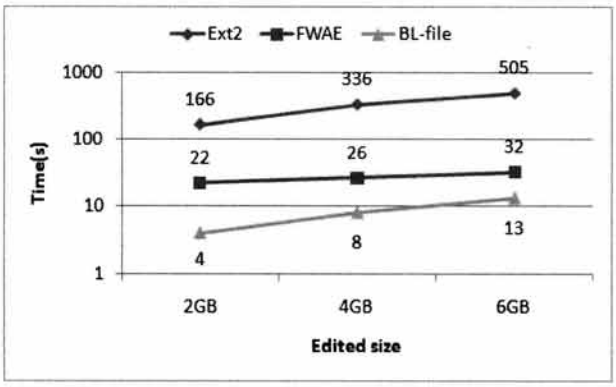
	FWAE 기법	블록 링크 파일 시스템
멀티미디어 파일 고속 편집 기능	멀티미디어 파일 중 불필요한 부분 삭제 • 원본 덮어쓰기 • 새로운 파일에 쓰기	멀티미디어 파일 중 불필요한 부분 삭제 • 원본 덮어쓰기 • 새로운 파일에 쓰기
공유 데이터 관리 방법	공유 데이터 블록마다 공유 횟수 정보 저장 • 새로운 파일에 편집한 데이터 저장할 때 원본 파일에 속한 모든 데이터 블록에 대해 공유 횟수 정보 생성 • 새로운 파일에 편집한 데이터 저장할 때 편집한 파일에 속하는 데이터 블록의 공유 횟수를 1증가 • 공유 파일 삭제 시, 삭제 파일에 속하는 데이터 블록의 공유 횟수를 1감소시키고 공유 횟수가 0이 되면 해당 데이터 블록 삭제	공유 데이터를 파일의 논리 오프셋을 이용한 세그먼트 단위로 관리 • 새로운 파일에 편집한 데이터 저장할 때 SI 구조체를 만들어 공유 데이터의 시작과 끝 오프셋을 세그먼트 단위로 기록 • 최대 300개의 세그먼트 관리 가능 • 공유 파일 삭제 시, 삭제 파일에 속하는 데이터를 SI 구조체에 저장된 공유 세그먼트의 범위와 비교하여 공유되지 않는 부분만을 선별하여 삭제
구현 환경	윈도우즈 응용 프로그램 • Ext2Read 오픈소스 기반 • mplayer 동영상 재생 툴과 연동	리눅스 Ext2 파일 시스템 • Ext2 파일 시스템에 BLFW API 추가 • Avidemux 동영상 편집 툴과 연동

5.1 멀티미디어 파일 편집저장 성능

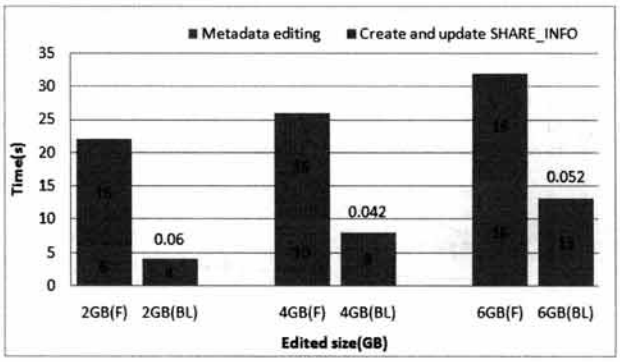
본 실험에서는 멀티미디어 파일 편집 저장 시 소요되는 시간, 편집 저장 세부 과정별 소요 시간, 그리고 편집 저장을 위해 사용하는 디스크 사용량에 대해 성능을 측정한다.

먼저, (그림 9)는 멀티미디어 파일 편집 저장 시 소요되는 시간을 비교하여 나타낸다. 8GB 원본 멀티미디어 파일에 대해 편집 저장 용량을 달리하여, 새로운 파일에 저장했을 때 결과를 나타낸다. 블록 링크 파일 시스템이 가장 우수한 성능을 보였다. 기존의 Ext2 파일 시스템은 편집된 내용을 디스크 공간에 모두 새로 쓰기 때문에 데이터 블록 I/O가 많이 일어나 많은 시간이 소요된다. 반면 FWAE와 블록 링크 파일 시스템은 데이터 블록을 공유하고 메타데이터 편집만을 통해 파일을 생성하므로 데이터 블록 I/O가 없어져 Ext2에 비해 성능이 큰 폭으로 향상 된다. 하지만 블록 링크 파일 시스템은 FWAE에 비해 공유 정보 관리를 위한 메타데이터 생성 과정이 간단하고 Windows 기반 응용 프로그램 형태로 동작하는 FWAE에 비해 최적화가 잘 되어 있어 약 3배 빠른 성능을 보여준다.

다음으로 FWAE와 블록 링크 파일 시스템의 멀티미디어 파일 편집 저장 시 성능차이를 명확히 살펴보기 위해 세부 과정 별 소요 시간에 대해 알아본다. 세부 과정으로는 필요한 데이터 블록만을 추려내어 데이터 블록 인덱스를 재구성하는 메타데이터 편집 과정과 공유 정보 생성 및 관리 과정으로 나눌 수 있다. (그림 10)에서 FWAE와 블록 링크 파일 시스템의 편집 저장 시 세부 과정별 소요시간을 보여준다. 메타데이터 편집에서 블록 링크 파일 시스템이 1.25배 우수한 성능을 보였다. 그 이유는 FWAE는 개발환경에 따른 최적화 문제로 데이터 I/O 성능이 떨어지기 때문이다. 또한, 공유 정보 생성 및 관리에 있어서도 블록 링크 파일 시스템이 우수한 성능을 보였다. FWAE는 데이터 블록 단위로 공유 정보를 관리하기 때문에 8GB 멀티미디어 파일에 대해 약 200만개의 데이터 블록 정보를 관리하여야 한다. 반면 블록 링크 파일 시스템은 편집 구간별로 공유 정보를 관리하므로 공유 정보 생성 및 관리에 있어 소요되는 시간이 훨씬 적다.



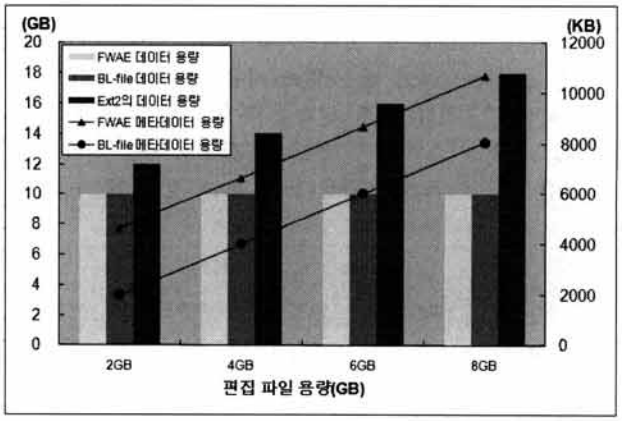
(그림 9) 편집 용량별 편집저장 소요 시간



(그림 10) 편집저장 세부 과정별 소요시간

5.2 디스크 소모 용량 비교

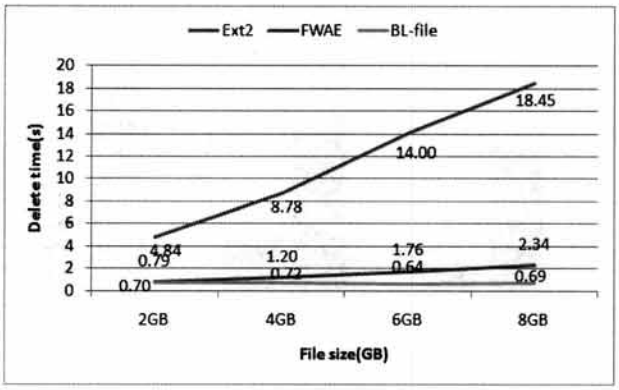
(그림 11)은 원본 파일을 편집하여 새로운 파일에 저장할 때, Ext2, FWAE 그리고 블록 링크 파일 시스템에서 원본 파일과 새로운 파일에 대해 사용하는 디스크 용량 비교를 나타낸다. 원본 파일의 크기는 10GB 이고 새로운 파일에 편집하여 저장하는 용량을 달리할 때, 실제 데이터 저장량과 부가적으로 생성되는 메타데이터 크기를 구분하여 실험하였다. Ext2 파일 시스템에서 새로운 파일에 편집한 내용을 저장할 경우, 바뀐 모든 데이터 내용을 디스크에 다시 기록한다. 반면 FWAE, 블록 링크 파일 시스템은 파일 간 데이터 블록을 공유하여 사용하기 때문에 저장 공간 소모가 급격히 줄어든다. 또한 블록 링크 파일 시스템은 공유 구간별로 공유 정보를 관리하기 때문에 FWAE에 비해 메타데이터 저장 공간 소모가 더 적다. FWAE와 블록 링크 파일 시스템은 데이터 블록을 공유하는 파일의 수가 늘어남에 따라 Ext2 파일 시스템에 비해 소모하는 디스크 공간 사용율이 더욱 줄어든다.



(그림 11) 편집 용량에 따른 필요한 저장 공간

5.3 BL 파일 삭제 성능

블록 링크 파일 시스템의 수정된 삭제 성능을 실험하기 위해 Ext2와 FWAE의 삭제 성능과 비교한다. 삭제 파일은 Ext2 파일 시스템에서는 파일 공유를 허용하지 않으므로 일반 파일을 삭제하고 FWAE와 블록 링크 파일 시스템에서는

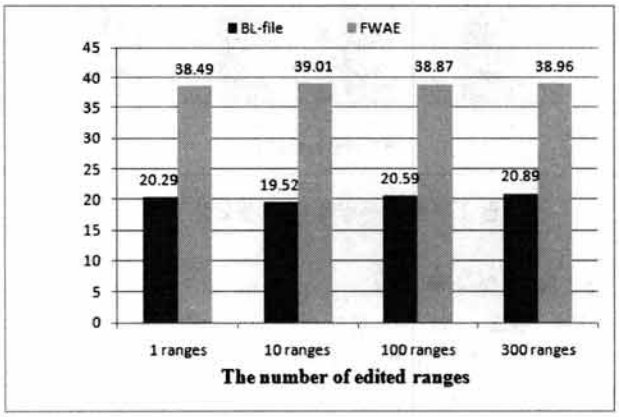


(그림 12) 편집된 파일 용량별 파일 삭제 성능

데이터 블록을 공유하는 파일을 삭제한다. (그림 12)는 편집된 파일 용량별 삭제 성능을 나타낸다. FWAE와 블록 링크 파일 시스템에서는 데이터 블록의 물리 주소가 적혀있는 간접 포인트 블록이 연속적으로 디스크에 적힌다. 그리고 Ext2 파일 시스템에서는 간접 포인트 블록들이 디스크 공간에 분산되어 있다. 따라서 블록 링크 파일 시스템과 FWAE는 Ext2 파일 시스템에 비해 디스크 seek time이 줄어들어 더 우수한 파일 삭제 성능을 보인다. 한편 블록 링크 파일 시스템은 FWAE에 비해 공유 정보 관리가 간단하기 때문에 FWAE에 비해 약 44% 더 우수한 성능을 보인다.

5.4 편집 구간 개수에 따른 성능 비교

블록 링크 파일 시스템은 편집 구간이 많아질수록 SI 구조체의 edits 배열에 저장하는 데이터가 많아지고 관리해야 하는 공유 데이터 영역이 많아진다. (그림 13)은 10GB 원본 파일을 8GB로 편집하여 저장할 때 편집 구간 개수에 따른 FWAE 기법과 블록 링크 파일 시스템이 필요로 하는 시간을 나타낸다. 블록 링크 파일 시스템은 현재 최대 300개의 편집 구간을 저장할 수 있다. 파일 아이노드의 osd2, osd3 영역과 같이 reserved 된 영역을 사용하면 최대 1000개 정도로 확장 가능하다. 실험 결과를 통해 FWAE 기법 및 블록 링크 파일 시스템 모두 편집 시 필요로 하는 시간이 편집 구간 개수에 크게 영향을 받지 않는 것을 알 수 있다.



(그림 13) 편집 구간 개수에 따른 성능 비교

실제 블록 링크 파일 시스템은 편집 구간 정보가 1개 증가할수록 12byte의 정보를 더 저장해야 한다. 하지만 메타데이터 생성 과정이 모두 메모리에서 일어나고 만들어진 메타데이터가 하드 디스크에 저장될 때 약 4KB 만 저장하면 되므로 편집 구간이 증가한다고 해서 심각한 오버헤드가 발생하지 않는다.

5.5 제안기법의 문제점 분석

5.5.1 편집 저장 시 발생하는 블록 내 데이터 손실

블록 링크 파일 시스템은 BLFW 명령 수행 시, 블록 단위 편집을 하기 때문에 선택된 편집 구간을 Ext2 파일 시스템의 데이터 블록 크기인 4KB 단위로 재설정 한다. Ext2 파일 시스템은 파일 가운데 4KB 미만의 데이터 블록 삽입을 할 수 없다. 4KB 미만의 데이터를 파일 가운데 삽입할 경우, 유효한 데이터 부분뿐만 아니라 4KB 데이터 블록 내에서 유효하지 않은 영역까지 유효한 데이터로 판단하기 때문이다. 따라서 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 프레임 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 나올 때 까지 영향을 미친다. 이때 데이터 손실로 인해 화면 일그러짐 현상 등이 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[17, 18].

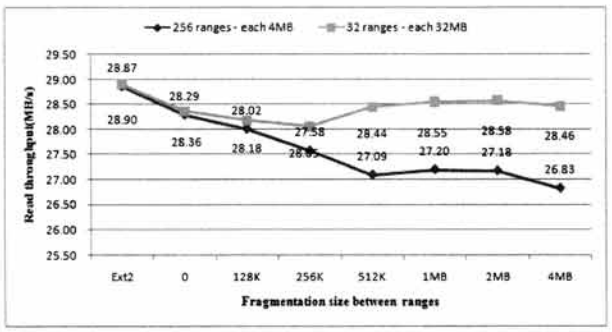
4KB 미만 데이터의 무결성을 보장하기 위해서는 특정 데이터 블록 내 데이터의 일부분만을 유효한 데이터로 판단하고, 그 부분만을 읽어오도록 하기 위한 방법이 필요하다. 예를 들어 10KB 데이터가 편집되었는데 이것이 3KB, 4KB, 3KB 공간을 차지하고 있다면, 가장 앞 쪽의 3KB 유효 데이터가 있는 데이터 블록을 손실 시키는 것이 아니라 파일을 읽을 때 첫 번째 데이터 블록의 3KB의 유효 데이터만 읽어오도록 만드는 것이다. 실제 이러한 방식은 관련 연구에서 소개한 데이터 베이스 기반인 파일 시스템인 XPRESS에서 사용되고 있다.

5.5.2 파일 단편화 현상

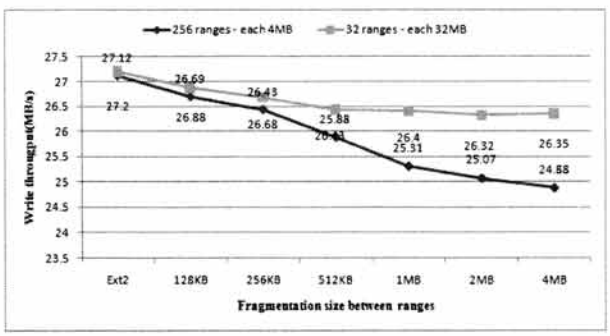
파일 단편화로 인한 성능 저하 정도를 좀더 상세히 알기

위해 파일 단편화 개수와 파일 단편화 구간간의 물리적 간격을 조정하여 읽기 및 쓰기 성능을 측정하였다. 단편화가 일어난 파일의 읽기 성능 측정을 위해 2GB 크기의 멀티미디어 파일을 BLFW(over) 명령을 이용하여 각각 연속적인 데이터가 4MB인 256개의 세그먼트와 연속적인 데이터가 32MB인 32개의 세그먼트로 만들어 단편화 구간간의 간격을 다르게한 파일을 만들었다. 그리고 각각의 파일에 대해 읽기 성능을 측정하였다. 그 결과는 (그림 14)와 같다. 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 멀티미디어 파일 편집 시 연속적인 데이터 저장 영역의 크기는 편집 형태에 따라 달라지지만, 최소 수 MB에서 수 GB로 실험에서 보여준 연속적인 데이터 저장 영역인 4MB, 32MB보다 더 크다. 따라서 파일 단편화로 인한 성능 저하의 영향을 적게 받는다. 또한 최악의 경우, 실험에서와 같이 단편화 파일이 26.83MB/s의 읽기 성능을 가진다하더라도 2.5MB/s의 HD급 멀티미디어 파일의 재생 속도를 충분히 만족 시킬 수 있다.

(그림 15)는 단편화로 인한 쓰기 성능 저하 실험 결과이다. 읽기 성능 실험과 같은 단편화 파일을 사용하였다. 실험을 위해 하드 디스크 저장 공간에 빈 블록이 남지 않게 데이터를 기록한다. 그리고 BLFW(over) 명령을 통해 단편화 파일을 만든 후 남은 빈 공간에 데이터를 기록하며 성능을 측정한다. 실험 결과 읽기 성능과 비슷한 결과가 나왔으며, 2.5MB/s로 재생되는 HD급 멀티미디어 파일의 재생 속도보다 빠르게 녹화가 가능하다.



(그림 14) 단편화로 인한 읽기 성능 저하



(그림 15) 단편화로 인한 쓰기 성능 하락

블록 링크 파일 시스템은 파일의 단편화가 일어나더라도 읽기 및 쓰기 속도가 HD급 멀티미디어 파일의 재생속도인 2.5MB/s보다 빠르면 사용하는데 있어 문제가 없다.

6. 결론 및 향후 연구 계획

본 논문에서는 대용량 멀티미디어 파일을 고속 편집 저장하기 위한 블록 링크 파일 시스템을 제안하였다. 블록 링크 파일 시스템은 실제 데이터 블록 I/O없이 메타데이터 수정만을 통해 빠른 편집이 가능하게 해준다. 또한 파일간 데이터 블록 공유를 통해 디스크 공간 사용량을 줄이고 편집한 내용을 새로운 파일에 저장할 경우에도 빠른 저장을 가능하게 한다. 본 논문에서는 FWAE의 공유 정보 생성 및 관리 부분을 개선하였고 리눅스 Ext2에 직접 구현하여 최적화 문제를 해결하였다. 또한 이 기능을 실제 동영상 편집 툴인 Avidemux에 적용하였고 실험을 통해 기존 Ext2에 비해 약 42배, FWAE에 비해 약 3배 빠른 동영상 편집저장 성능을 보여주었다. 이와 같이 블록 링크 파일시스템은 Ext2의 안정성과 멀티미디어 응용 기능 등을 제공해준다.

향후 연구로 Ext4 파일 시스템[19]의 익스텐트(Extent) 블록 매핑 방식을 이용하여 Ext4 파일 시스템에서 편집저장 성능을 더욱 개선하는 연구를 진행할 계획이다.

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] 정승완, 남영진, 서대화, "Ext3 파일 시스템 기반의 편집된 대용량 멀티미디어 파일의 고속 저장 기법", 한국정보처리학회 논문지A, V.16A, No.2, pp.89-100, April, 2009.

[7] 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.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] 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.
[9] Sean Quinlan and Sean Dorward, "Venti: a new approach to archival storage," *Proc. of USENIX Conf. on File and Storage Technologies*, Jan., 2002.
[10] A. Muthitacharoen, B. Chen, and D. Mazi'eres, "A low-bandwidth network file system," *Proc. of the 18th ACM Symp. on Operating Systems Principles*, Oct., 2001.
[11] D. Bovet and M. Cesati, *Understanding the LINUX KERNEL*. 3rd edition, O'Reilly, pp.738-774, 2006.
[12] 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.
[13] T. Ts'o, "Planned extensions to the Linux Ext2/Ext3 filesystem," *Proc. of the 2002 USENIX Ann. Technical Conf.*, pp.235-243, 2002.
[14] "Whitepaper: Red Hat's new journaling file system: ext3", http://www.redhat.com/support/wpapers/redhat/ext3/ext3.pdf
[15] 신용주, 김정원, 김영주, 정기동, "대용량 파일들의 편집, 저장을 위한 효율적인 파일 관리 기법", 한국정보과학회 가을 학술발표논문집, Vol.55, No.7, 1997.
[16] 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.
[17] P. N. Tudor, "MPEG-2 video compression," *IEE journal*, Vol.7, pp.257-264, Dec., 1995.
[18] 이호석, 김준기, "알기 쉬운 MPEG-2", 홍릉과학출판사, 2002.
[19] A. Mathur and etc., "The new ext4 filesystem: cururrent status and future plans," *Proc. of Linux Symp.*, Jun., 2007.
[20] LG Electronics, "Xcanvas Player," http://xcanvas.co.kr
[21] SourceForge, Inc., Ext2Read, http://sourceforge.net/projects/ext2read.

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

고 석 영

e-mail : seokyoungko@lignex1.com
2007년 경북대학교 전자전기컴퓨터학부(학사)
2009년 경북대학교 전자전기컴퓨터학부(공학석사)
2008년~현 재 LIG넥스원 항공연구센터 2팀
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 IT대학 전자공학부 교수
관심분야 : 임베디드 SW, 병렬처리, 분산운영체제 등