데이터 블록의 변경을 지원하는 사용자 수준 파일시스템 설계 및 구현

공진산*, 정호민*, 고영웅*
*한림대학교 컴퓨터공학과
e-mail:{kongjs, chorogyi, yuko}@hallym.ac.kr

Design and Implementation of User-level File System Supporting File Data Block Update

Jin-San Kong*, Ho-Min Kim*, Young-Woong Ko*
*Dept of Computer Engineering, Hallym University

요 약

현재 디스크 스토리지에서 기존 파일시스템은 파일의 부분 추가 또는 삭제 시 재 정렬하는 오버헤드(re-write)가 발생한다. 이러한 단점을 보완하기위해 블록 단위 변경을 지원하는 사용자 수준(User-level)의 파일시스템을 설계 및 구현하였다. 주요 아이디어는 블록 정렬된 파일에 대하여 향상된 블록 삽입/삭제를 지원하는 API를 지원함으로 블록 단위의 데이터 삭제 시 오버헤드를 줄일 수 있는 기법이다. 실험 결과 제안한 시스템이 파일의 부분 추가 및 삭제 시 수행 시간을 크게 감소시키고 데이터 쓰기 용량을 크게 감소시키는 것을 보였다.

1. 서론

최근 하드웨어의 발달과 UCC의 발달로 인해 멀티미디어 파일과 같이 대용량 파일을 수정하는 일이 빈번해지고 있다. 그러나 현재 널리 사용되고 있는 파일시스템들의 경우 추가나 삭제 시 재 정렬하는 오버헤드가 크다. 예를 들어 TAR 파일의 경우 필요 없는 파일을 TAR에서 제거할 때 파일 위치가 앞에 있을 경우 뒷부분의 데이터를 앞부분으로 옮기기 위해서 디스크 쓰기를 새로 해야 하며 RAR, ZIP 압축 포맷, 동영상 편집기 같은 경우에도 동일한 일이 발생한다. 또한 향후 널리 쓰이게 될 SSD (Solid State Device) 같은 경우에는 쓰기 작업 횟수가 수명과 연관되므로 재 정렬하는 오버헤드를 보완해야 할 필요가 있다. 본 논문에서는 정렬을 위해 발생되는 오버헤드를 줄이기 위해 블록단위로 파일을 수정할 수 있는 API와 이를 지원하는 사용자 수준의 파일시스템을 설계하고 구현하였다.

제안하는 시스템을 설계하기 위해 open, read, write, seek와 같이 기본적인 파일 입출력과 modify라는 블록단위로 파일을 수정할 수 있는 API를 구현했다. 또한 파일시스템을 슈퍼블록, 디렉터리 블록, 할당 테이블 블록, 데이터 블록으로 구분하고 슈퍼 블록에서는 기본적인 파일시스템 정보를 다루고 디렉터리 블록에서는 디렉터리 구조와 파일들의 정보를 저장하며 할당 테이블 블록에서는 데이터 블록에서 실제 블록의 위치를 나타내도록 구현하였다.

본 논문에서 제안한 시스템의 유용성을 검증하기 위해 TAR 포맷 파일의 구간별 추가/삭제 실험을 진행하였으며 실험 결과 수행시간과 전체 쓰기 양이 기존의 파일시스템에서 수행했을 때 보다 줄어드는 것을 확인해 본 논문의 아이디어가 유용함을 증명하였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 관련연구를 살펴보고, 3장에서는 기존 파일시스템 문제점을 알아보고, 4장에서 기존 파일시스템의 문제를 해결하기위해 사용자 수준의 파일시스템의 설계에 대해 설명한다. 장에서는 제안한 시스템에 대한 성능평가, 6장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

현재 디스크에서 널리 사용되고 있는 파일시스템은 EXT[1][2][3], FAT[4] 등이 있다. 최근 보급되는 SSD는 수만 번의 수정으로 소자가 파괴되므로 쓰기 작업 횟수를 줄일 수 있는 웨어 레벨링(wear leveling)이 필요하다. 그러므로 디스크 기반의 파일시스템과 다른 구조가 필요하다. SSD의 대표적인 파일시스템으로 JFFS[5][6], YAFFS[7] 등이 있다.

일반 하드디스크의 EXT(EXTended filesystem)는 리눅스 파일시스템 중 하나이고, 4가지의 버전이 존재한다. 데이터 저장에 있어 EXT는 블록으로 나뉘어져 있고, 이 블록은 블록 그룹으로 나뉜다. 각각의 블록 그룹은 블록 그룹 디스크

본 연구는 교육과학기술부와 한국연구재단의 지역혁신인력양성사업과 기초연구사업(No.2009-0076520)의 지원을 받은 결과물임을 밝힙니다.

립터 테이블과 슈퍼블록을 포함하고 있으며, 모든 블록 그룹은 블록 비트맵, 아이노드 비트맵, 아이노드 테이블을 포함하고 있고, 마지막으로 실제 데이터 블록을 포함한다.

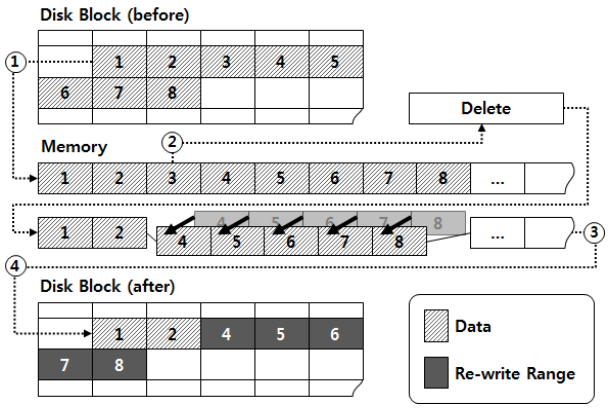
FAT(File Allocation Table)은 대부분의 메모리 카드와 수많은 컴퓨터 시스템에 널리 쓰이는 컴퓨터 파일시스템 구조이다. 그리고 FAT16, FAT32, exFAT 등 여러 버전이 존재한다. FAT은 크게 부트 레코드, 예약된 영역, FAT영역, 데이터 영역으로 구분된다. FAT영역은 클러스터들을 관리하는 테이블이 모여 있는 공간이다. 데이터 영역은 파일 또는 디렉터리가 저장되어 있고, 클러스터라고 불리는 논리적인 단위로 데이터를 보관한다.

SSD의 JFFS(the Journaling Flash File System)는 LFS(Log-structured File System)방식의 플래시 메모리 파일시스템으로써 우수한 안정성때문에 널리 사용되고 있다. 버전에 따라 JFFS, JFFS2 JFFS3가 존재하고, JFFS2부터 NAND 플래시 메모리에 적합한 파일시스템으로 개발되었다. JFFS에서는 저널링 노드를 기반으로 데이터를 저장하며, 가비지 컬렉션 기법을 통해 웨어 레벨링 기능을 수행한다.

YAFFS(Yet Another Flash File System)는 파일을 쓰기 위해 파일의 메타데이터를 저장하는 페이지를 먼저 쓰고, 그 파일에 속한 모든 페이지의 잉여 영역에 공통된 파일 ID를 기록하며, RAM에 저장된 트리를 통해 이들 페이지의 연관관계를 파악한다. 만일 파일에 속한 어떤 페이지가 더 이상 유효한 값을 갖고 있지 않다면, 그 페이지의 잉여 영역이 'Dirty'로 표시된다. 블록에 속한 모든 페이지가 'Dirty'로 표시 된 경우나 플래시 메모리에 가용 공간이 없을 경우, 유효한 페이지를 다른 블록으로 복사한 후 해당 블록을 삭제한다.

3. 기존 파일시스템의 문제점

기존 파일시스템에서 특정 파일에 대해 디스크에 입출력하는 작업은 빈번하게 일어나게 된다. 이때 기존의 파일시스템에서는 디스크 내 특정 파일의 일정 구간을 삭제하였을 때 파일의 빈 공간을 처리하기위해 재 정렬하는 오버헤드(re-write)가 발생한다.

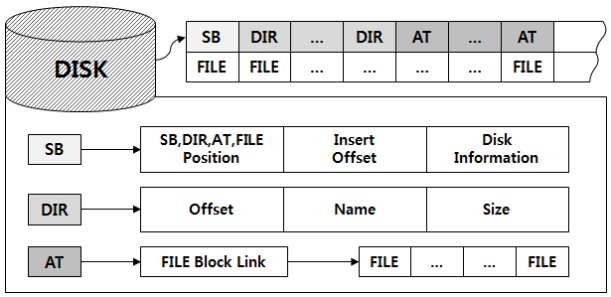


(그림 1) 기존 파일시스템의 데이터 수정 오버헤드

(그림 1)은 블록에 순차적으로 저장되어 있는 특정 파일에 대하여 삭제가 일어났을 때 오버헤드가 발생하는 단계를 나타낸다. 원호 안에 숫자는 기존 파일시스템의 삭제 순서를 나타내고 있다. ①은 디스크에 파일이 순차적으로 블록에 저장되어 있고, 삭제 명령 시 메모리에 올라가는 것을 보여준다. ②는 삭제 시 메모리로 올라온 파일에서 파일의 일정 구간(3번째 블록)이 삭제된 것을 의미하고, ③은 파일의 일정 구간(3번째 블록)이 삭제되었을 때 그 구간을 채우기 위해서 삭제 구간 뒤에 있는 블록들이 재 정렬되어지는 것을 보여준다. ④는 재 정렬된 블록들을 디스크에 다시 써지는 것을 의미한다. 이 때 디스크에서는 삭제된 부분을 제외한 나머지 부분에 대하여 다시 쓰는 오버헤드가 발생되며 파일의 용량이 커지거나 파일의 삭제 구간이 앞쪽에 있을 때 처리 시간이 길어지게 된다.

4. 제안하는 파일시스템 설계

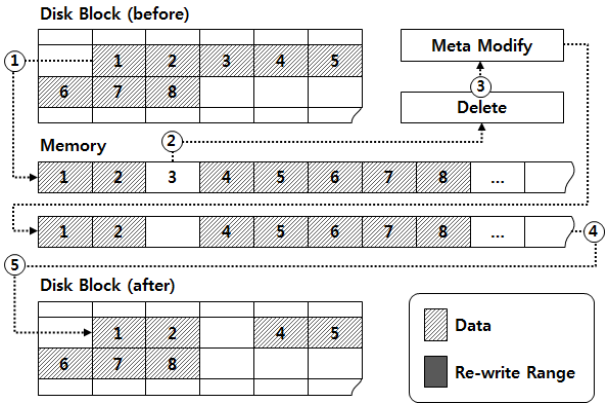
제안하는 파일시스템은 리눅스 상에서 구축한 사용자 수준의 파일시스템이다. 주요 모듈은 SB(Super Block), DIR(Directory), AT(Allocation Table), 데이터 블록으로 이루어진 구조를 가지며, 블록 단위로 정렬되어 있는 파일이 메타 데이터를 통하여 고정 블록 단위로 디스크에 저장되고, 그 테이블을 유지할 수 있도록 하는데 목적이 있다.



(그림 2) 제안하는 파일시스템 구조

(그림 2)에서는 제안하는 파일시스템을 구성하는 요소들을 보여준다. 구축한 사용자 수준의 파일시스템은 다른 파일시스템과 마찬가지로 블록 형태의 데이터 입출력이 가능하며, 각 블록의 위치는 테이블 형태의 자료구조에서 링크되어 관리한다. SB은 블록의 위치 인덱스, 에러 블록 인덱스, 디스크의 상태 등의 전체적인 정보들을 가지며, DIR은 파일시스템에 담고 있는 파일에 대한 위치, 크기, 이름 등의 정보를 가지고 테이블 형태로 유지한다. AT는 데이터 블록과 일대일 연결 관계를 저장하고 있으며 최초의 블록 접근은 DIR의 링크로부터 시작한다.

또한, 파일 입출력을 위한 기본적인 API들을 제공한다. 파일을 생성하거나 열기위한 open과 입출력을 위한 read, write, 그리고 재 정렬 오버헤드를 피하기 위하여 추가된 block_modify 함수가 있다. 본 연구에서 제공하는 API는 가상 파일시스템을 실제 파일시스템과 동일하게 사용가능하도록 해주는 역할을 한다.



(그림 3) 개선된 파일시스템의 파일 입출력

(그림 3)은 본 논문에서 제안하는 핵심 아이디어에 대한 흐름을 보여준다. 디스크에 각 블록의 연결 관계 정보를 가지는 AT를 가지고 있고, 파일의 삭제 기능 수행 시 AT의 수정만을 통해 이루어진다. 원호 안에 숫자는 파일 시스템의 삭제 순서를 나타내고 있다. ①은 파일시스템에 저장된 파일이 삭제 시 메모리에 올라가는 것이고, ②는 전체 파일의 일정 구간(3번째 블록)삭제를 의미한다. ③은 AT를 수정하여 블록들의 링크를 다음 블록에 연결하는 것이다. ④는 AT 수정을 통하여 재 정렬 오버헤드가 발생하지 않음을 의미하고, ⑤는 위 과정을 통하여 일정 부분 삭제된 파일이 디스크에 저장되는 것을 보여준다.

결과적으로 본 시스템에서는 블록 삭제 시 디스크 내에서 오버헤드가 발생하지 않으며 입출력에 대한 시간 비용을 감소시킬 수 있다.

5. 성능 평가

실험에 사용한 컴퓨터는 CPU 클록 3.0GHz, 2GB RAM의 하드웨어 스펙을 가지며 운영체제는 Fedora Core 9이다. 디스크 스토리지의 파일시스템은 EXT4를 사용하였다. 실험 데이터는 기존 파일시스템과 제안하는 파일시스템의 비교를 위해 여러 파일을 묶을 수 있고, 블록 정렬된 파일 포맷인 TAR를 사용했고, TAR로 묶인 1.2GB에 132개의 서브파일을 가진 파일이다.

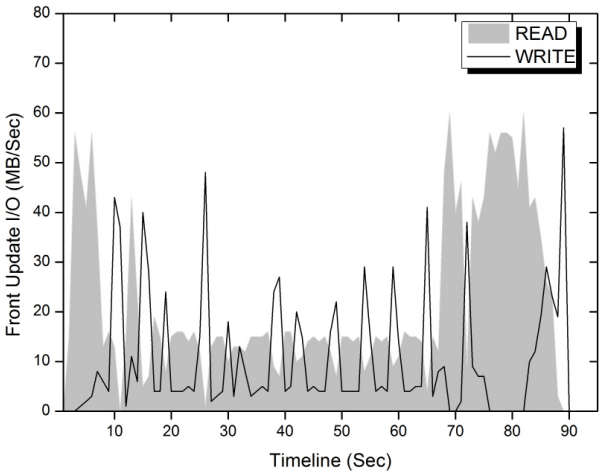
기존 파일시스템에서 재 정렬 오버헤드를 확인하기위해 TAR의 구간별로 서브파일을 삭제하는 실험을 진행하였다.

<표 3> 파일의 구간 별 삭제 시 읽기 오버헤드(MB)

	처음부분	중간부분	끝부분
1차	1256.20	493.37	10.85
2차	1263.01	516.91	0.23

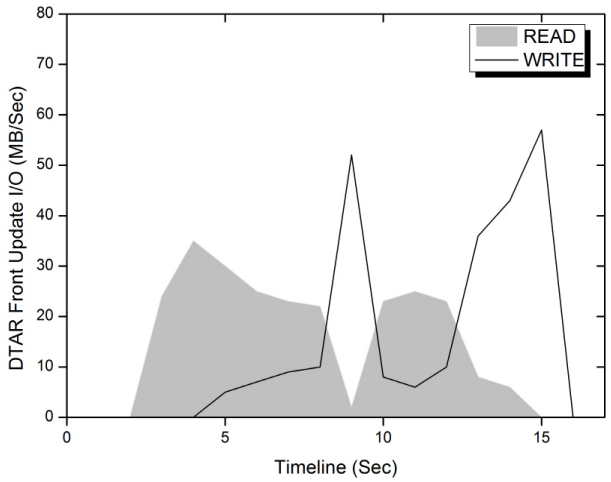
<표 3>은 TAR파일에 대하여 처음, 중간, 끝 부분을 삭제함으로써 얼마나 많은 재 정렬 오버헤드가 발생하는지 실험한 것이다. 실험 결과 현재 사용되는 파일시스템에 TAR파일의 구간 삭제를 진행하였을 때, 첫 부분이 삭제되면 블록이 파일 크기만큼 다시 써지는 것을 볼 수 있다. 중간 부분이나 끝 부분이 삭제되었을 때는 구간 변경이 이루어진 시점까지만 블록 재 정렬을 하기 때문에 다시 쓰는 블록의 크기가 처음 부분 삭제에 비해 적어졌다. 실험 시간 또한 오버헤드와 비례하여 감소했다.

본 연구에서 서브파일의 추가/삭제 실험을 위해 기존 파일시스템과 제안한 파일시스템 각각 동일한 동작을 하는 업데이트 루틴을 사용했다. 이 업데이트 루틴은 서브파일들의 오래된 버전을 삭제하고, 버전업된 파일을 추가하는 동작으로 이루어졌다. 실험 파일은 TAR로 묶인 1.2GB에 8개의 서브파일을 가진 파일이고, 파일의 업데이트 루틴 수행동안 읽기/쓰기 오버헤드를 초단위로 측정했다.



(그림 4) 기존 파일시스템의 업데이트 오버헤드

(그림 4)는 기존 파일시스템에서 TAR의 처음 부분에 위치한 서브파일에 대해 업데이트 오버헤드를 측정한 것이다. 70초 이전의 읽기/쓰기 오버헤드는 기존 파일시스템의 재 정렬 오버헤드이고 그 후의 읽기/쓰기는 파일의 추가에 대한 오버헤드이다.



(그림 5) 제안한 파일시스템의 업데이트 오버헤드

(그림 5)는 본 논문에서 제안한 파일시스템에서 TAR의

처음 부분에 위치한 서브파일에 대해 업데이트 오버헤드를 측정한 것이다. 서브파일의 삭제 시 파일시스템의 메타 영역만을 수정하므로 삭제에 대한 읽기/쓰기 오버헤드가 거의 발생되지 않은 것을 확인할 수 있었고, 추가에 대한 읽기/쓰기 오버헤드만 발생한 결과를 얻었다. 또한 기존 파일시스템과 업데이트 수행 시간을 비교하였을 때, 기존 파일시스템은 재 정렬 오버헤드로 인해 90초가 걸린 반면 제안하는 파일시스템은 17초가 걸렸다. 실험 결과를 통해 제안하는 파일시스템에서 동작하는 블록 정렬된 서브파일이 디스크 스토리지 활용에서 큰 효과가 있을 것을 확인했다.

6. 결론 및 향후연구

본 논문에서는 기존 파일시스템에서 파일 추가 및 삭제 시 발생하는 오버헤드를 줄이기 위해, 블록 정렬된 파일을 사용하는 사용자 수준의 파일시스템을 설계하고 구현한 결과를 기술하였다. 주요 아이디어는 블록 정렬된 파일에 대해 일정 구간 삭제 시 기존 파일시스템과 다르게 메타 데이터 영역만을 접근하고, 링크의 수정을 통하여 데이터 블록의 변경에 대한 오버헤드를 최소화하는 것이다. 실험 결과 본 논문에서 기존 파일시스템과 제안한 파일시스템의 업데이트 읽기/쓰기 오버헤드를 측정하였고 입출력 수행시간이 대폭 감소함을 보였다. 향후 연구로는 제안한 파일시스템을 실제 리눅스 커널에 구현하고, 그림이나 동영상이 첨가된 파일(ppt 등 문서 파일)을 블록 정렬된 파일 포맷으로 변환하는 API를 구현하고, 이러한 파일을 지원하는 향상된 파일시스템에 대하여 구현할 계획이다.

참고문헌

[1] Ts'o, T.Y. and Tweedie, S. "Planned extensions to the Linux EXT2/EXT3 filesystem" USENIX 2002 Annual Technical Conference, Freenix Track, 235-244, 2002.

[2] Cao, M. and Tso, T.Y. and Pulavarty, B. and Bhattacharya, S. and Dilger, A. and Tomas, A. "State of the art: Where we are with the ext3 filesystem" Proceedings of the Ottawa Linux Symposium (OLS), 69-96, 2005.

[3] Mathur, A. and Cao, M. and Bhattacharya, S. and Dilger, A. and Tomas, A. and Vivier, L. "The new ext4 filesystem: current status and future plans" Ottawa Linux Symposium, 2007.

[4] Microsoft Corporation, "Microsoft Extensible Firmware Initiative FAT32 File System Specification," 2000.

[5] D.Woodhouse "JFFS: The Journaling Flash File System" Ottawa Linux Symposium, 2001.

[6] LOU, Y. and ZHANG, Y. "Analysis and Improvement of the JFFS3" Science Technology and Engineerin, Volume 9, 605-610, 2009.

[7] Manning, C. "How YAFFS works" Retrieved April Volume 6, p2011, 2010