차세대 비휘발성 메모리를 활용한 플래시 파일 시스템 연구*

박세은[O], 최종무[¶], 이동희[†], 노삼혁[‡]
단국대학교 정보컴퓨터학부[O¶]
서울시립대학교 컴퓨터과학부[†]
홍익대학교 정보컴퓨터공학부[‡]
pse1002@dankook.ac.kr

A Status Report on the Implementation of a Flash File System that Exploits Non-Volatile RAM

Se Eun Park[O], Jongmoo Choi[¶], Donghee Lee[†], Sam H. Noh[‡]
Division of Information and Computer Science, Dankook University[O¶]
Department of Computer Science, University of Seoul[†]
School of Computer & Information Engineering, Hongik University[‡]

요 약

차세대 비휘발성 메모리(Non-Volatile RAM, 이후 NVRAM)의 사용이 현실화 되면서 이를 활용한 저장 장치의 성능 개선 연구가 활발히 진행되고 있다. 본 논문에서는 NVRAM을 이용한 플래시 파일 시스템의 성능 향상 방법을 제안한다. 우선 자주 갱신 되는 정보를 NVRAM에 유지시켜 플래시 메모리의 덮어쓰기(overwrite)로 인한 성능 저하 문제를 개선한다. 또한 NVRAM에 파일시스템의 메타 정보 위치를 유지하여 파일 시스템을 마운트할 때 요구되는 플래시 메모리의 탐색 공간을 줄인다. 실험 결과 마운팅 시간이 줄고 플래시 메모리의 접근 횟수가 감소함을 확인하였다.

1. 서 론

디지털 융합(digital convergence)이 활발히 진행되면서 이동형 장치(Mobile Device)는 더욱 대용량, 고성능의 비휘발성 메모리를 요구하고 있다. 예를 들어 이전의 휴대폰은 전화 기능과 PIMS(Personal Information Management System)을 위한 소량의 비휘발성 저장 공간만 있으면 충분했지만, 요즘 휴대폰은 MP3, 디지털 카메라, 게임 등의 용량을 위한 대용량의 저장 공간을 요구하고 있다. 최근 Video Streaming, 캠코더 등의 기능이 휴대폰에 추가되고 있으며, 이때는 대용량뿐만 아니라 고성능 저장 공간을 요구한다.

현재 이동형 장치의 저장 공간으로 많이 사용되는 것은 플래시 메모리이다. 하지만 플래시 메모리는 덮어 쓰기(overwrite)가 허용되지 않는다는 하드웨어적인 특성이 있다. 즉 플래시 공간을 삭제(erase)한 후 처음 쓰기는 허용되지만, 쓰여진 데이터의 갱신은 허용되지 않는 것이다.

이를 극복하기 위해 flash file system은 트랜잭션 기법을 이용한 원자적 삭제 후 쓰기 기법이나 다른 위치에 쓴 후 사상(mapping) 정보의 수정 등의 기법을 사용한다. 하지만 이러한 작업은 쓰기 시간의 지연을 야기하며 결국 플래시 메모리의 성능을 저하시킨다. 한편, 플래시 파일 시스템은 마운트될 때 플래시 메모리 공간 전체를 탐색하도록 구현되어 있으며, 이 때문에 야기되는 마운팅 시간의 지연은 대표적인 플래시 파일 시스템의 문제점으로 지적되고 있다[4].

본 논문에서는 FRAM, PRAM, MRAM과 같은 차세대 비휘발성 메모리(NVRAM)를 이용해 플래시 파일 시스템의 성능 향상을 시도한다. 우선 플래시 메모리의 갱신 빈도가 높은 데이터를 NVRAM에 유지하여, 플래시 메모리로 전달되는 갱신 요청의 횟수를 줄인다. 또한 파일 시스템을 마운팅 할 때 필요한 정보가 저장된 플래시 위치를 NVRAM에 저장하여, 플래시 메모리 공간 전체를 탐색할 필요성을 없앤다.

본 논문에서는 제안된 기법을 64MB NAND 유형의 플래시 메모리와 4MB FRAM(Ferro-electric Random Access Memory)이 탑재된 내장형 보드(EZ-X5 보드)에서 실험하였다. 보드에는 리눅스가 동작하며, 플래시 메모리는 YAFFS 플래시 파일 시스템으로 관리한다. 본 연구진은 YAFFS의 내부에 FRAM 관리 모듈을 추가하였다. 한편 FRAM이 없을 경우 SDRAM으로 FRAM을 에뮬레이션 할 수 있는 인터페이스도 구현하였다. 실험 결과 플래시 파일 시스템의 마운팅 시간이 줄고, 수행 중에 플래시 메모리의 접근 횟수가 감소하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를, 3장에서는 파일 시스템의 설계 및 구현을 설명한다. 4장에서는 성능평가 결과를 보여주며, 향후 연구 내용을 5장에서 정리한다.

2. 관련 연구

최근 NVRAM이 삼성, 인텔 등의 회사에서 발표되고 있다[2]. NVRAM의 장점은 (i) DRAM 공정과 호환성이 높아 고집적화가 가능하며, (ii) 정보의 쓰기, 소거가 고속이고, (iii) 비휘발성이며, (iv) 쓰기 횟수에 제한이 거의 없고, (v) 소비전력이 적다[1].

* 본 연구는 한국과학재단 특정기초연구(R01-2004-000-10188-0)지원으로 수행되었음.

	FLASH	FRAM	MRAM	PRAM
Non Volatility	Yes	Yes	Yes	Yes
Read speed	~100ns	~100ns	~100ns	~100ns
Write speed	10us(program) 1ms(erase)	~100ns	~100ns	~500ns
Stand by Current	20~100uA	~10uA	~10uA	~10u

표 1. NVRAM의 특징

표 1은 대표적인 NVRAM의 종류와 특징을 비교한 것이다[3]. 결국 NVRAM은 DRAM과 달리 비휘발성을 제공하고, 플래시와는 달리 덮어 쓰기가 가능하며, 쓰기 속도가 빠르다.

한편, 플래시 메모리를 위한 대표적인 파일 시스템에는 JFFS2와 YAFFS가 있다. JFFS2는 LFS(Log-structured File System)처럼 플래시 메모리에 대한 갱신 연산을 추가 연산으로 변형하여 처리하며, 이를 통해 플래시 메모리의 덮어 쓰기가 허용되지 않는 문제를 해결하였다. 하지만 플래시 메모리의 이용률(utilization)이 커지면 쓰기 속도가 저하되고, 마운트 시간이 오래 걸리며, DRAM 메모리를 많이 사용하는 단점이 있다. 이런 단점을 해소하기 위해 YAFFS가 개발되었으며, YAFFS는 마운트 속도와 NAND 유형 플래시 메모리의 입출력 속도에서 JFFS2보다 좋은 성능을 보인다[4].

3. 설계 및 구현

그림 1은 YAFFS의 구조와 동작 원리를 도시한 것이다. NAND 유형 플래시 메모리는 512Bytes의 페이지와 16Bytes의 스페어 영역으로 구성된다. 그리고 몇 개의 페이지들이 모여 블록을 구성한다(1세대 NAND 유형 플래시는 32개의 페이지들이, 2세대는 64개의 페이지들이 하나의 블록을 구성). 플래시 메모리에서 읽기와 쓰기는 페이지 단위로 수행되며, 삭제는 블록 단위로 수행된다.

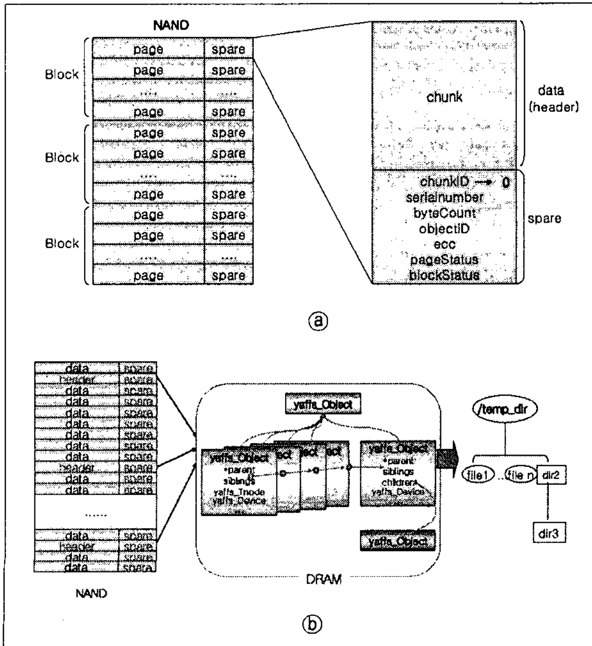

그림 1 YAFFS 동작 원리

YAFFS는 페이지와 스페어를 각각 "chunk"와 "tag"라는 용어로 관리한다. Chunk에는 파일의 내용(data)과 파일 속성 정보(header)가 저장되고, tag에는 chunk에 저장된 내용의 관리 정보가 저장된다. 파일 속성 정보는 파일 이름과 파일 크기, 수정 시간, 상위 디렉터리 등의 내용들로 구성된다. Tag의 관리 정보는 chunk에 저장된 내용이 어떤 파일 것인지에 대한 식별자(objectID), 파일 내부에서 chunk의 위치(chunkID), 갱신 연산시 증가되는 일련 번호(serialnumber), chunk내에 유효한 바이트 수(byteCount), 에러 검출을 위한 ECC 등으로 구성된다. ChunkID가 0이면 chunk에는 header 즉 파일의 속성 정보를 유지하며, 0보다 큰 수이면 chunk에는 파일의 실제 데이터가 저장되어 있다.

YAFFS는 마운팅할 때 플래시 메모리에서 모든 header chunk들을 DRAM으로 읽어 파일 시스템 구조를 생성한다. 각 header chunk마다 yaffs_Object라는 객체가 DRAM에 만들어지며, 이 객체들은 서로 연결되어 파일 시스템의 트리 구조를 형성한다. 또한 각 객체는 자신이 관리하는 파일에 속한 data chunk들의 위치를 yaffs_Tnode라는 계층 구조로 관리한다. 결국 특정 파일을 open()하면 객체 간에 트리 구조를 이용해 해당 객체를 찾고, 이후 read()/write() 요청은 객체의 Tnode를 이용해 서비스된다. 만일 파일 내용이 수정되면 수정된 data chunk와 header chunk는 즉시 플래시 메모리의 프리 페이지에 쓰이며, 위치와 일련 번호 등의 관리 정보들이 갱신된다.

지금까지 설명에서 우리는 YAFFS의 두 가지 단점을 발견할 수 있다. 첫째, 마운팅 할 때 yaffs_Object라는 객체를 생성하기 위해 플래시 메모리의 모든 chunk들을 순차적으로 탐색해야 한다는 것이다. 둘째, header chunk는 파일 속성 정보를 유지한다는 특성상 갱신 빈도가 높은데, 갱신할 때 마다 신뢰성을 위해 플래시 메모리에 써야 한다는 것이다.

본 논문은 플래시 메모리와 DRAM 계층 사이에 FRAM이라는 NVRAM을 활용하여 위 두 문제의 해결 방법을 제안한다. 그림 2는 본 논문에서 제안한 시스템 구조를 보여준다.

FRAM에는 최근에 수정된 header chunk들이 유지된다. FRAM의 크기와 플래시 메모리의 크기, 그리고 파일 시스템에 존재하는 파일의 개수에 따라 파일 시스템에 존재하는 모든 header chunk가 FRAM에 존재하도록 설계할 수도 있으며, 일부만 FRAM에 존재하도록 설계할 수도 있다. 만일 전자의 방법으로 설계하면 플래시 메모리에는 data chunk들만 존재하게 된다. 반면 후자의 방법으로 설계하면 header chunk들의 일부는 플래시 메모리에, 다른 일부는 FRAM에 존재하게 된다. 본 연구에서는 확장성을 위해 후자의 방법을 선택하였다. 따라서 최근에 갱신된 일부 header chunk들만 FRAM에 존재하게 되며, 이 공간이 임계치 이상 사용되면 header chunk들을 플래시 메모리에 쓰도록 하였다. 리눅스 커널이 cpu_idle()을 수행할 때 FRAM에서 플래시 메모리로 header chunk들의 쓰기를 수행하면 큰 부하 없이 이 작업의 수행이 가능하다. 또한 대부분의 header chunk의 갱신을 플래시 메모리가 아닌 FRAM에서 서비스할 수 있기 때문에 파일 연산의 지연시간을 줄일 수 있다.

한편, 본 논문에서는 플래시 메모리에서 header chunk들의 위치를 FRAM에서 관리한다. 그림 2에서 "Location in NAND"

가 이 정보를 의미한다. 그럼 파일 시스템을 마운팅할 때 플래시 메모리 전체를 탐색할 필요 없이 FRAM의 정보만으로 모든 header chunk들의 위치를 발견할 수 있다. 물론 Tnode를 구성하기 위해 data chunk들의 위치를 탐색해야 할 필요는 여전히 있지만, 본 연구진은 NVRAM을 활용하여 이 탐색 또한 없앨 계획이다. 이렇게 되면 마운팅할 때 플래시 메모리 전체 공간을 탐색할 필요성이 완전히 없어진다.

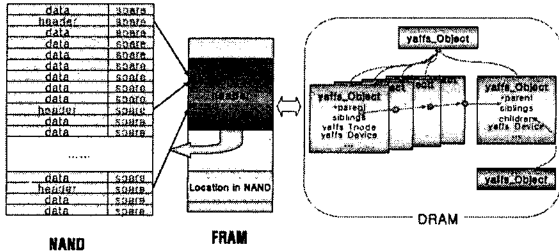

그림 2 FRAM을 활용한 파일 시스템 구조

4. 실험 결과

본 논문에서 제안한 기법을 YAFFS에 추가하였다. YAFFS는 리눅스의 모듈 프로그래밍 기법을 이용해 구현되었으며, MTD를 사용해 플래시 메모리를 접근한다. 현재 본 연구진은 EZ-X5라는 내장형 보드에 FRAM을 통합하는 작업을 진행 중이며, 본 논문에 기술된 결과는 EZ-X5에서 SDRAM을 FRAM으로 에뮬레이트한 환경에서 실험 결과이다. 실험에서 FRAM에 유지되는 header chunk의 최대 개수는 30개이다.

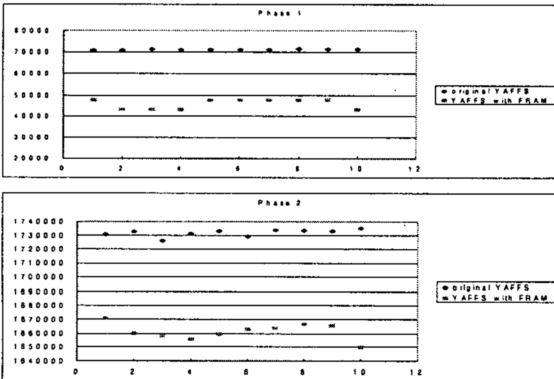

그림 3 MAB 수행시간

그림 3은 MAB(Modified Andrew Benchmark) 수행 결과이다[5]. MAB는 디렉터리 생성, 파일 복사, 파일 검색, 컴파일 등의 단계로 구성된다. 하지만 제한된 성능의 CPU를 사용하는 내장형 환경에서 컴파일 같은 작업은 거의 수행되지 않는다. 따라서 본 실험에서는 디렉터리 생성과 파일 복사를 수행하는 단계 1과 2만을 수행하였다. 실험은 모두 10번 진행하였으며, 그림 3과 그림 4에서 x축은 각 실험을 의미하고 y축은 각 실험의 수행 시간을 의미한다. 실험 결과 NVRAM을 활용한 파일 시스템이 기존의 파일 시스템에 비해 좋은 성능을 보임을 알 수 있었다. 구체적으로 그림 3의 단계 1(Phase 1)에서 약 20000㎲, 단계 2(Phase 2)에서는 약 70000㎲의 성능 향상을 얻을 수 있었다. 이것은 header 내용을 갱신할 때 플래시 메모리가 아닌 FRAM을 접근하였으며, 결국 플래시 메모리의 갱신 연산 부하를 줄였음을 의미한다.

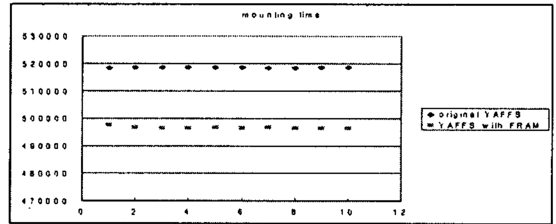

그림 4 파일시스템 마운트 시간

그림 4는 파일 시스템 마운트 시간을 측정한 것이다. MAB를 수행한 후 원래 YAFFS와 본 논문에서 수정한 파일 시스템을 실험하였다. 이 실험에서도 NVRAM을 활용한 파일 시스템이 약간 좋은 성능을 보임을 알 수 있다. 성능향상은 header chunk의 일부를 플래시 메모리가 아닌 FRAM에서 읽어 왔기 때문에 얻어진 것이다. 하지만 Tnode 구성을 위해 수정된 파일시스템에서도 플래시 메모리 전체 영역에 대한 탐색이 수행되며, 이 때문에 본 연구진은 기대했던 것 만큼의 성능향상을 얻지 못하였다. 현재 이 부분에 대해서는 header chunk에 Tnode를 구성할 수 있는 정보를 추가적인 부하 없이 유지하여 플래시 메모리 전체 영역에 대한 탐색이 필요하지 않도록 수정 중이다.

5. 결론

본 연구는 NVRAM을 활용한 플래시 파일 시스템의 성능 향상 기법을 제안하였다. 자주 접근하는 데이터와 메타 정보의 위치를 NVRAM에 유지하여, 파일 접근 시간과 마운팅 시간이 단축됨을 확인하였다. 향후 마운트할 때 플래시 메모리 전체 영역의 탐색을 완전히 없애는 방법과 NVRAM을 데이터 캐시로 확장하는 방안을 연구할 것이다.

참고 문헌

[1] SIGDA Technical News, http://www.acm.org/da_technews/articles/2003-1/0710t.html
[2] Hongsik Jeong, New Memory Technology , http://semiplaza.snu.ac.kr/lecture/samsung/sam_7.html
[3] 유병곤, 류상욱, 윤성민 유비쿼터스용 유니버설 메모리 기술, 전자통신경향분석, 제 20권, 1호, 130~138, 2005
[4] http://www.alephl.co.uk/yaffs/yaffs.html, YAFFS Spec.
[5] J. Ousterhout, Why aren't operating systems getting faster as fast as hardware?, In Proceedings of the Summer USENIX Technical Conference, pages 247-56, Anaheim, CA, Summer 1990. USENIX