비휘발성램을 사용하는 플래시 파일 시스템의 효율적인 메타데이터 관리기법

여해동 ○ 원유집
한양대학교
{valkun, yjwon}@ece.hanyang.ac.kr

Metadata management in hierarchical file system

Haedong Yeo ○ Youjip Won
Hanyang university

요 약

JFFS나 YAFFS와 같은 플래시 메모리를 사용하는 파일 시스템들은 in-place업데이트가 불가능한 NAND플래시 메모리의 특성 때문에 주로 로그 구조 기반으로 되어 있기 때문에 마운트 시간이 길어 지는 단점을 가지고 있다. 최근에는 이러한 문제점을 극복하고 파일 시스템의 성능을 높이기 위해 NVRAM을 이용하는 방법이 연구 되고 있으나, 현재 양산 되는 NVRAM의 크기는 제한 되어 있는데 반해 이러한 연구들이 성능 향상을 위해 필요로 하는 NVRAM의 크기가 너무 크기 때문에 현실적으로 사용하기 어렵다는 문제점을 가지고 있다. 본 논문에서는 이러한 문제점을 극복하기 위해서 로그 구조 기반의 파일 시스템이 메인 메모리에 생성하는 메타데이터를 경량화 및 압축하여 NVRAM에 저장함으로써, 저비용으로 파일 시스템의 마운트 성능을 개선 하는 방법을 연구 하였다. 이를 통해 마운트 시간을 획기적으로 줄이는 동시에 필요한 NVRAM의 크기를 플래시 메모리의 0.15% 수준으로 줄일 수 있었다.

1. 서 론

차세대 메모리로 주목 받고 있는 FRAM(Ferroelectric RAM), MRAM(Magnetic RAM), PRAM(Phase change RAM)과 같은 바이트 단위의 접근이 가능한 비휘발성 메모리(NVRAM)는 구동에 필요한 전력이 매우 낮으며, 바이트 단위로 접근 가능하고, in-place 업데이트가 가능한 RAM의 장점을 가지고 있으면서 비휘발성이라는 저장소의 특성을 가지고 있어 기존 운영체제에 많은 변화를 가져 올 것으로 보인다. 특히 임베디드 시스템 및 메모리 기반의 파일 시스템 분야에 현재 널리 쓰이고 있는 NAND 플래시 메모리가 페이지 단위로 읽고 쓸 수 있고 블록 단위로 삭제해야 해야 하기 때문에 생기는 문제점을 가지고 있기 때문에 이런 한 문제점이 없는 NVRAM은 향후 메모리 기반의 저장 매체 시장에도 큰 영향력을 행사 할 수 있을 것으로 기대된다. 그러나 아직까지 플래시 메모리에 비하여 용량이 작고 가격이 비싼 편이며, 앞으로도 당분간은 NVRAM의 CPM(Cost per MB)이 플래시 메모리 보다 낮아 지기는 힘들 것으로 예측됨으로 성능과 비용을 고려할 때 NVRAM만을 사용하는 시스템 보다는 플래시 메모리와 NVRAM을 같이 사용하는 시스템이 널리 쓰일 것으로 생각된다.

YAFFS[1]나 JFFS[2]와 같이 in-place 업데이트가 불가능한 NAND 플래시 메모리의 특성을 반영한 로그 구조 기반의 파일 시스템의 가장 큰 단점은 마운트 시간이 오래 걸려 시스템의 부팅시간이 오래 걸린다는 점이다. 파일 시스템에서 NVRAM을 사용하여 마운트 시간을 개선 하려는 기존의 연구는 몇몇 존재하지만 성능 개선을 위해서 필요로 하는 NVRAM의 크기의 크기가 커서 비용이 많이 든다는 단점이 있다.

본 논문에서는 NAND 플래시 메모리와 NVRAM을 같이 사용하는 로그 구조 기반의 파일 시스템에서 NVRAM에 저장되는 파일 시스템의 메타데이터의 효율성을 개선하여 필요한 NVRAM의 크기를 줄여 시스템의 비용대비 성능을 개선하고자 한다

2. 배 경

2.1 NAND 플래시 메모리

NAND 플래시 메모리는 NOR 플래시 메모리와는 달리 바이트 단위로 읽고 쓰는 것이 불가능 하고 페이지 단위로 읽고 쓰고, 블록 단위로 지우기만 가능하다는 단점이 있으나 CPM이 낮아 각종 임베디드 시스템, MP3 플레이어, 최근에는 SSD까지 메모리 기반의 저장 매체로 가장 널리 쓰이고 있다.

최근 주로 사용되고 있는 라지 블록(Large Block) NAND 플래시 메모리의 경우에 그림1과 같이 한 블록은 128KB의 데이터를 저장 할 수 있으며 4KB의 스페어 영역을 가지고 있다. 각 블록은 64개의 페이지와 스페어 영역으로 구성 되어 있으며 각 페이지의 크기는 2KB, 스페어 영역의 크기는 64B이다. 스몰 블록(Small Block) NAND 플래시의 경우는 하나의 블록 (16KB +

512B)이 32개의 페이지를 포함하고 있으며 각 페이지는 16KB, 스페어 영역은 16B로 구성되어 있다. NAND 플래시 메모리는 소자의 특성상 데이터가 존재하는 페이지에 데이터를 쓰기 위해서는 먼저 삭제를 해야 하고, 삭제 명령의 경우는 블록 단위로만 가능하기 때문에 in-place 업데이트를 하는 것이 불가능하게 된다.

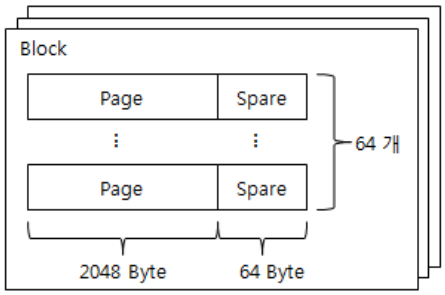

그림1. 라지 블록 NAND 플래시 메모리 구조

2.2 NAND 플래시 파일 시스템

다수의 NAND 플래시 파일 시스템은 위에서 언급한 in-place업데이트가 불가능한 소자의 특성 상 로그 구조를 사용하고 있으며 이러한 로그 기반의 파일 시스템은 사용시간이 길어지면 데이터가 플래시 메모리의 전반에 흩어 지게 되어 파일 블록을 찾는데 시간이 오래 걸리게 되게 때문에 일반적으로 파일 객체와 각 파일 객체의 실제 데이터 블록이 위치하는 물리적 주소 정보인 PAT(Physical Address Translation)를 메인 메모리 상에 유지하고 있다.

로그 구조 기반의 NAND 플래시용 파일 시스템은 ext[3]와 같은 일반 적인 파일 시스템이 파일 시스템의 메타데이터를 통해 각 파일 시스템 객체(파일, 디렉토리, 심볼릭 링크, 하드 링크)의 실제 데이터 블록의 위치를 알 수 있는 것과는 반대로 각 데이터 페이지가 그 페이지들이 해당하는 파일 시스템 객체의 정보를 가지고 있는 역사상(reverse mapping)방식을 사용하고 있으며, 각 페이지가 어떤 파일 객체의 어떤 부분의 데이터를 가지고 있는지의 정보를 저장하기 위해서 스페어 영역을 사용하고 있다.

그림2와 같이 YAFFS와 같은 로그 구조의 기반의 파일 시스템에서는 마운트시에 플래시 메모리의 스페어 영역을 모두 읽어 들여 각 페이지가 어떤 객체의 데이터를 포함하는지를 알아 낸 후 PAT를 구성하고 각 객체의 리스트 및 객체의 해시 테이블을 메인 메모리 상에 구성하는 작업을 하게 되는데, 이때 걸리는 시간이 사용되는 플래시의 용량에 비례하여 선형적으로 증가함으로 용량이 큰 경우는 큰 문제점이 된다. 파일 시스템의 마운트 작업은 주로 시스템의 부팅시점에 이루어 지기 때문에 이런 낮은 확장성(scalability)이 로그 구조 기반의 파일 시스템의 사용을 보편화 하는데 큰 걸림돌이 되고 있다.

전체 크기가 T인 NAND 플래시 메모리의 마운트 과정에서 읽어 들여야 하는 스페어 영역의 전체 크기(Ts)를 라지 블록 플래시 메모리와 스몰 블록 플래시 메모리의 경우를 나누어 다음과 같이 계산 할 수 있다.

라지 블록 플래시 메모리 (S_i = 4KB, β = 128KB)

스몰 블록 플래시 메모리 (S_i = 512B, β = 16KB)

$$\alpha = \frac{T}{\beta}, \qquad Ts = \sum_{i=1}^{\alpha} S_i$$

라지 블록 플래시와 스몰 블록 플래시에서 동일하게 전체 플래시 메모리의 크기가 같으면 스페어 영역의 크기가 같은 것을 알 수 있다.

시스템에 1GB의 플래시 메모리가 있다면 전체 스페어 영역의 크기 만 32MB가 되는 것이다.

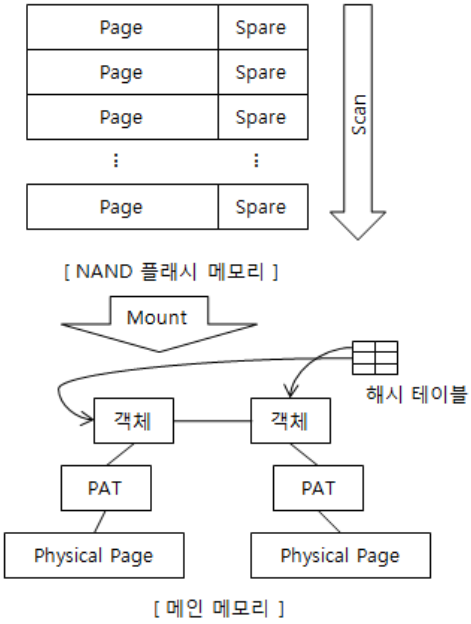

그림2. NAND 플래시용 파일 시스템의 마운트 과정

3. 관련연구

파일 시스템에서 NVRAM을 사용하여 성능을 향상 시키고자 하는 연구는 주로 디스크 기반의 파일 시스템에서 이루어 졌으나, 최근 들어 NVRAM과 NAND 플래시 메모리를 결합한 파일 시스템에서 이루어 지고 있다.

Conquest[4]는 파일 시스템에 지속성 램(Persistent RAM)을 도입한 하이브리드 파일 시스템으로 자주 접근되는 파일 시스템의 메타데이터와 작은 크기의 데이터 블록을 지속성 램에 저장함으로써 성능을 개선 하였고, MRAMF[5]는 NVRAM의 직접도가 낮아 용량이

낮다는 점에 주목하고 디스크 기반의 파일 시스템에서 NVRAM을 적용하고 데이터를 실시간으로 압축하여 효율적으로 NVRAM을 사용하려는 시도를 한 파일 시스템이다. HeRMES[6]는 파일 시스템 접근의 50% 이상이 메타데이터에 대한 접근임에 착안해 속도가 빠른 MRAM에 메타데이터를 저장하여 성능을 향상 시키는 동시에 MRAM을 쓰기 버퍼 및 저장소로 사용하는 등 MRAM을 다양한 용도로 사용하고자 하는 시도를 하였다.

FRASH[7]와 MiNVFS[8]는 NVRAM과 플래시 메모리를 이용하는 파일 시스템에서 모든 메타데이터를 NVRAM에 저장하여 마운트 속도와 파일시스템 접근 성능을 향상시키고 있다.

PFFS[9]는 파일시스템 메타데이터를 ext2와 유사하게 이중 간접 인덱스(double indirect index)까지 구성하여 NVRAM에 저장함으로써 성능을 향상 시키는 동시에 NVRAM으로 사용하고 있는 PRAM의 수명을 고려하여 PRAM에 데이터를 저장 할 때도 웨어레벨링(wear-leveling)를 하도록 설계 하였다.

위 언급한 연구들은 다양한 방법으로 NVRAM을 사용하여 파일 시스템의 성능을 개선하려 노력하였지만 요구되는 NVRAM의 크기가 너무 크다는 문제점을 공통적으로 가지고 있다.

4. 설 계

본 파일 시스템의 목적은 로그 구조 기반의 파일 시스템에서 메인 메모리상의 파일 시스템 데이터를 조작하여 NVRAM에 저장 함으로써 마운트 시간을 줄이고자 할 때 NVRAM을 효율적으로 사용하여 필요한 NVRAM의 크기를 줄이는 데에 있다. 이를 위해서 마운트 과정에 필요한 데이터들이 어떤 것들이 있으며 이들을 어떻게 구성 하여 NVRAM에 저장하는 것이 효율적인지 살펴 보는 것이 중요하다.

그림 3은 NAND 플래시 메모리를 사용하는 로그 구조 기반의 파일 시스템에서 데이터를 어떻게 플래시 메모리에 배치하는지를 나타낸 그림이다.

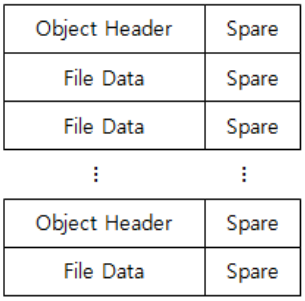

그림3. NAND 플래시 메모리 상의 데이터 구조

플래시 메모리의 페이지는 객체의 상세 정보를 담고 있는 오브젝트 헤더와 객체의 실제 데이터를 담고 있는 데이터 페이지로 나뉘어 지며, 오브젝트 헤더에 담긴 객체의 상세 정보는 객체의 종류, 아이디, 이름, 액세스 권한, 액세스 시간 등이고, 각 페이지의 스페어 영역에는 해당 페이지가 오브젝트 헤더를 포함하는지, 특정 객체의 데이터를 담고 있는지에 대한 정보와 그 페이지가 어떤 객체에 해당 되는 페이지 인지에 대한 정보가 있다.

파일에 대한 접근이 있을 때 마다 플래시에서 파일의 메타데이터를 읽는 일 없이 빠르게 파일 시스템을 사용하기 위해선 플래시 메모리의 데이터를 마운트시에 읽어 들여 메인 메모리상에 자료 구조를 생성 하여야 하며, 이 자료구조는 크게 오브젝트 헤더로부터 읽어 드린 객체에 대한 상세 정보와 스페어 영역으로부터 읽어 드린 각 객체의 실제 데이터를 포함 하고 있는 페이지의 물리적 주소 정보인 PAT, 이 두 가지로 나누어 볼 수 있다.

위에서 언급한 네 가지 메타데이터를 NVRAM으로 관리 함으로써 파일 시스템의 성능을 향상 시킬 수 있을 것으로 보고 각각을 살펴 보면 오브젝트 헤더와 스페어 영역을 NVRAM에 저장 할 경우 마운트 시간을 줄 일 수 있는 장점 외에도 메타 데이터에 대한 접근 속도가 빨라져 파일 액세스 성능까지 높일 수 있는 장점이 있으나, 이를 위해서 NVRAM상에 존재해야 하는 데이터의 크기가 파일의 개수에 따라 달라 질 수 있는 오브젝트 헤더를 제외 하더라도 2.2에서 계산 한 바와 같이 1GB NAND 플래시 메모리 당 적어도 32MB 이상이 됨으로 현재 양산 되는 NVRAM의 크기[10]를 고려 하면 요구되는 NVRAM의 크기가 너무 커지게 된다.

본 시스템에서는 메인 메모리 상의 구조체들을 NVRAM에 저장하여 마운트 속도를 높이는 스냅샷 기법을 응용하되, 구조체들의 크기를 경량화 하여 필요한 NVRAM의 크기를 최소화 하는데 중점을 두어 설계 하였다.

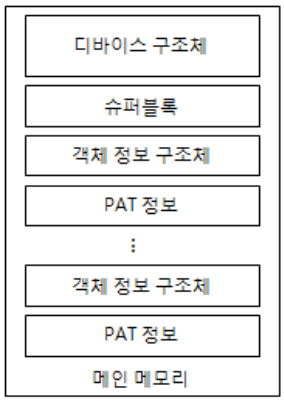

그림4. 메인 메모리 상의 구조체 구조

메인 메모리상의 구조체들은 그림4와 같이 NAND 플래시 메모리의 페이지 크기, 블록당 페이지의 개수,

스페어 영역의 크기 정보 등의 플래시 메모리의 물리적 특성과 전체 블록들의 상태, 파일 시스템의 슈퍼 블록 등을 포함 하는 디바이스 구조체와 파일 시스템 상의 객체를 나타내는 객체 구조체, 각 객체의 물리 데이터 블록의 주소를 나타내는 PAT로 구성되어 있다.

메인 메모리상의 구조체 각각의 크기를 살펴 보면, 디바이스 구조체의 크기는 3648바이트, 객체 구조체의 크기는 124바이트이며 PAT정보는 트리 형태로 이루어져 있으며 파일의 크기에 따라 전체 트리의 크기가 변할 수 있다. 파일 시스템 내의 파일의 개수에 따라 메인 메모리상의 구조체 크기의 합은 달라 질 수 있지만 NVRAM에 저장하기에는 너무 큰 크기임으로 이를 줄이기 위해서 그림5와 같이 각 구조체의 내용 중 플래시 메모리의 스페어 영역을 모두 스캔하지 않고 오브젝트 헤더 페이지나 슈퍼 블록만 읽어 오면 값을 얻어 올 수 있는 부분을 모두 제외 하고 경량화 한 컴팩트 구조체들을 정의하고 언마운트 시점에서 메인 메모리상의 구조체들을 컴팩트 구조체로 변경하여 NVRAM에 저장하게 하였다.

컴팩트 디바이스 구조체	컴팩트 객체 구조체
- 블록의 상태 정보	- 객체의 종류
- 가비지 콜렉터를 위한 지워진 파일 정보	- 객체 및 부모의 아이디
- 웨어레벨링을 위한 블록 시퀀스 정보	- 크기
	- 시리얼 번호

그림5. 압축 전 컴팩트 객체의 구조

또한 변환된 컴팩트 구조체들을 압축하여 NVRAM을 보다 효율적으로 사용할 수 있게 하였다. 일반적으로 리눅스 커널 내부 및 외부에서 일반적인 용도로 널리 쓰이고 있는 압축 방식인 ZLIB[11]의 경우 압축률은 높지만 속도가 느리고 압축 시 메모리를 많이 사용하고 있어, 본 시스템에서는 압축률은 ZLIB 보다 낮지만 상대적으로 압축 속도가 빠르고 압축 시 요구되는 메모리의 크기가 적어 본 논문이 제안하는 파일 시스템이 주로 적용되는 임베디드 시스템에 사용하기 적합한 LZO[12] 압축기를 사용하였다. LZO 압축기는 특히 압축을 푸는 속도가 빨라 마운트시에 NVRAM에 압축된 메타데이터를 이용해도 마운트 속도에 영향을 적게 주면서 NVRAM의 사용량을 줄이는데 가장 적합하다고 할 수 있다.

블록 단위의 압축 알고리즘인 LZO 알고리즘의 단위 블록 크기를 결정하기 위하여 블록의 크기를 변화 시키면서 압축하는 실험 결과(그림6)를 반영하여 본 파일 시스템에서는 컴팩트 객체들의 전체 크기를 하나의 블록으로 처리하여 압축을 하였다.

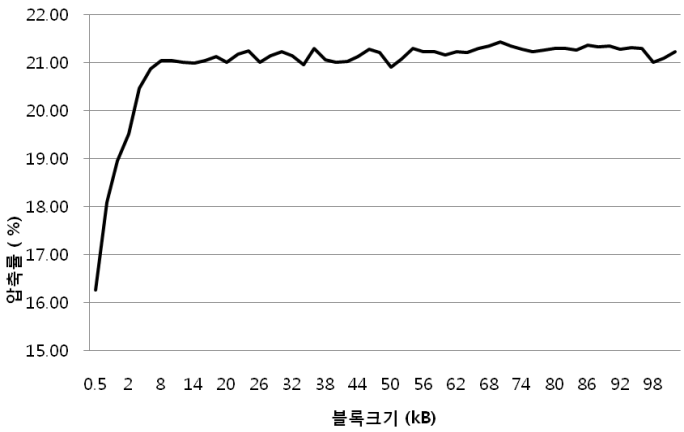

그림6. 블록 크기 별 압축률 변화

본 시스템에서 메타데이터를 NVRAM에 저장하는 모듈을 전체적으로 요약하면 그림7과 같은 구조를 가지고 있으며, 그 절차를 간단히 요약하면 메인 메모리상의 파일 시스템 메타데이터 객체들을 경량화기(Object Compactor)를 통해 경량화 하여 컴팩트 객체로 만든 후 압축기(Metadata compressor)를 통해 전체 컴팩트 객체를 압축하여 NVRAM에 저장하는 절차를 거치게 된다. 또한 NVRAM 시뮬레이터를 두어 메인 메모리를 사용해 데이터를 저장하고 언마운트 한 이후에 메인 메모리의 데이터를 다른 파일 시스템으로 덤프하게 하여 NVRAM이 부착되어 있지 않은 시스템에서도 NVRAM을 사용하는 것과 같은 실험을 진행 할 수 있게 하였다. 이러한 절차는 파일 시스템의 마운트와 언마운트시에 일어나며, 압축된 메타데이터를 기록하는 중에 시스템의 크래시가 일어나 NVRAM상에 불완전한 데이터가 있더라도 다음 마운트 시점에서 플래시의 스페어 영역으로부터 데이터를 읽어 들여 마운트 작업을 마칠 수 있게 된다.

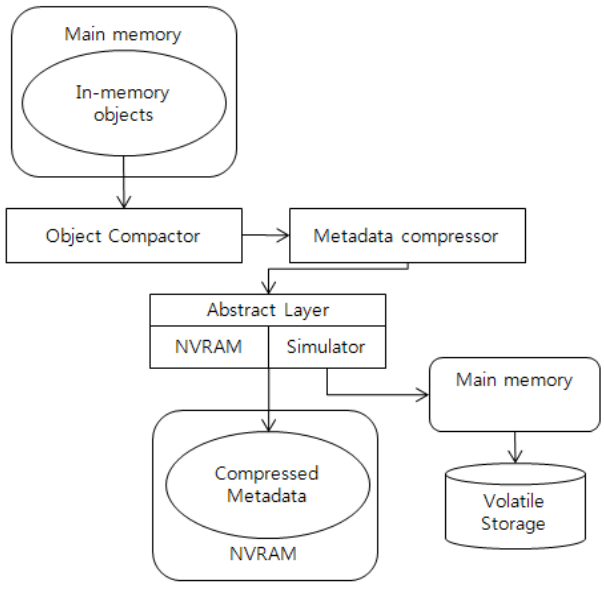

그림7. 메타데이터 관리기의 구조

5. 실 험

본 논문에서 제안한 파일 시스템(CMFS: Compressed Metadata File System)의 구현에는 Marvell PXA320 코어와 128MB의 SDRAM, 128MB의 라지 블록 NAND 플래시 메모리가 부착 되어 있는 임베디드 시스템을 사용하였으며 플래시 메모리를 두 개의 MTD 파티션으로 분할하여 63.5MB의 플래시 메모리를 CMFS의 영역으로 설정하고, NVRAM이 부착되어 있지 않은 상기 임베디드 시스템에서 실험을 진행하기 위해서 CMFS의 NVRAM 시뮬레이터를 사용하였다.

시스템을 리눅스 커널을 사용하고 애플리케이션 툴킷으로 QT를 사용하는 MP3 플레이어라고 가정하고 파일 시스템상에 리눅스 루트 파일 시스템, QT 라이브러리 및 QT기반의 애플리케이션, MP3 파일을 저장해 놓고 실험을 진행하였다.

그림7은 파일 시스템의 크기 별로 NVRAM에 저장되는 메타데이터의 크기를 측정한 그래프이다. 파일 시스템의 크기가 커짐에 따라 같은 비율로 파일의 개수도 증가 시켜본 결과 파일 시스템의 크기에 선형적으로 필요로 하는 NVRAM의 크기가 커지는 것을 확인 할 수 있으며, 필요로 하는 NVRAM의 크기는 전체 파일 시스템 크기의 약 0.13%~0.14% 정도로 개선되었음을 확인 할 수 있다. 약 1.3MB~ 1.4MB정도의 NVRAM으로 1GB의 NAND 플래시 메모리를 사용하는 파일 시스템의 마운트 성능을 향상 시킬 수 있는 것이다.

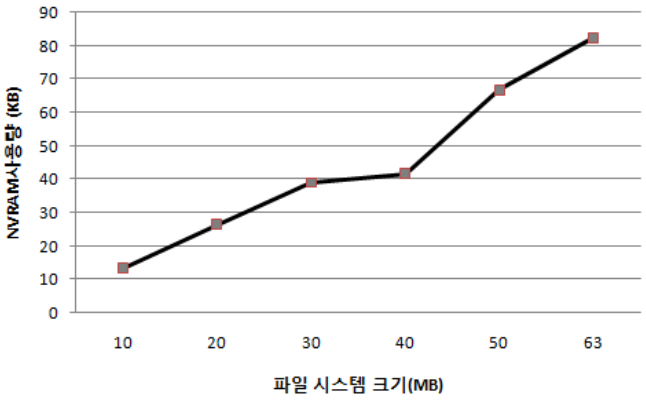

그림7. 파일 시스템 크기 별 NVRAM사용량

다음으로 압축된 메타데이터를 사용할 때 마운트 및 언마운트 속도의 향상 정도를 알아 보기 위한 실험을 진행 하였다. 측정된 시간은 마운트와 언마운트를 각각 10회씩 실시하여 얻은 평균 값으로 최소 10 마이크로세컨드까지 측정 하였으며 비교를 위하여 본 논문에서 제안하는 CMFS와 NAND 플래시 메모리용으로 가장 많이 쓰이고 있는 YAFFS와 속도를 비교 하였다. 그림8은 CMFS와 YAFFS의 마운트 속도를 비교한 결과이다.

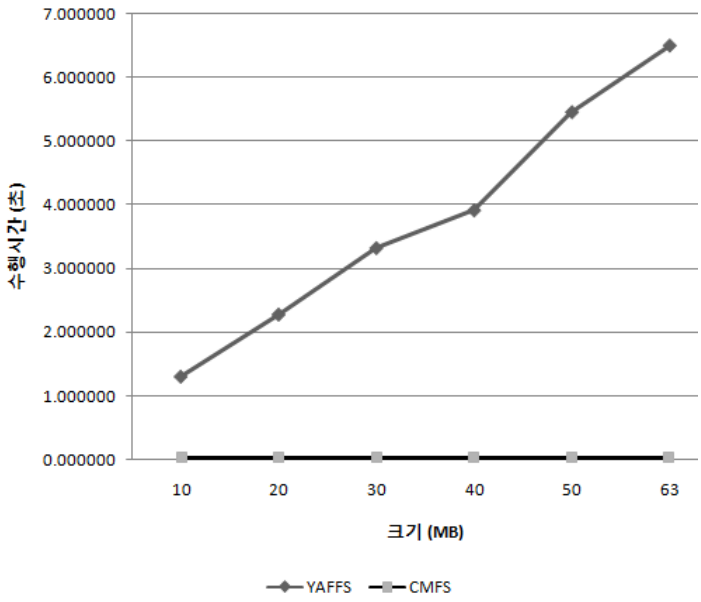

그림8. 파일 시스템 크기에 따른 마운트 시간 비교

크기	10MB	20MB	30MB	40MB	50MB	63MB
(ms)	227.84	239.05	248.55	254.28	273.79	282.64

표1. CMFS의 파일 시스템 크기에 따른 마운트 시간

표1은 CMFS의 마운트 속도 측정 결과를 좀더 자세히 표시한 것이다. CMFS 역시 파일 시스템의 크기가 10MB 커질 때 마다 약 5~10ms 정도로 마운트 시간이 증가하고 있지만 YAFFS에 비해서 마운트 시간이 크게 개선 되었음을 확인 할 수 있다.

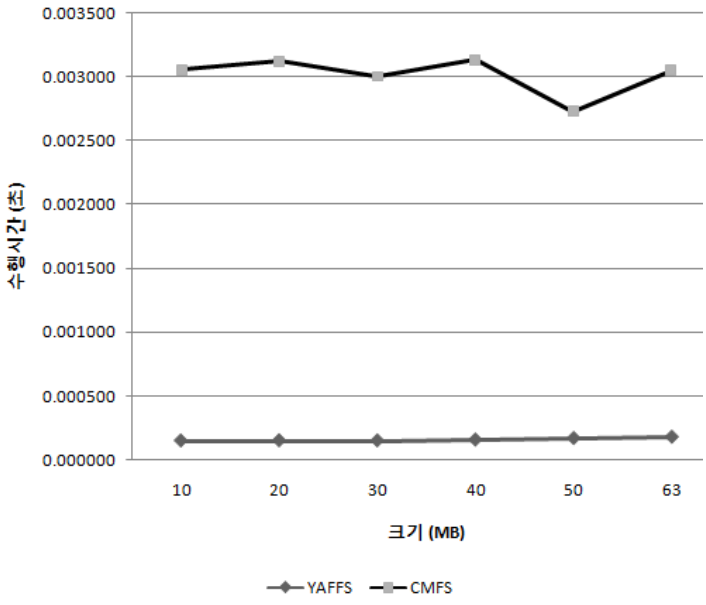

그림9. 파일 시스템 크기에 따른 언마운트 시간 비교

그림9는 CMFS와 YAFFS의 언마운트 시간을 측정한 결과이다. 언마운트 시점에 메모리상의 객체를 경량화하고 압축하는 절차를 거치기 때문에 YAFFS와 비교할 때 속도가 30ms 정도 느리다. 그러나 파일 시스템의 크기에 따라 다를 수 있지만 마운트 시에 얻을 수 있는 성능의 개선 효과가 크기 때문에 언마운트 시에 발생하는 약간의 성능 감소는 큰 문제가 아닌 것으로 볼 수 있다.

6. 결 론

RAM의 장점과 저장 매체로서의 장점을 동시에 가진 NVRAM의 특성상 NVRAM의 다양한 용도로 사용 될 수 있을 것이나, 아직까지 집적도가 낮아 용량이 작고 가격이 비싸다는 단점이 있다. 본 논문에서는 이러한 NVRAM을 효율적으로 사용하기 위한 한 방법으로 NAND플래시 메모리를 사용하는 로그 구조 기반의 파일 시스템의 긴 마운트 시간이라는 단점을 보완하기 위해 NVRAM을 사용하였으며, NVRAM에 저장되는 메타데이터를 최대한 경량화하고 압축하여 NVRAM에 저장함으로써 제한된 리소스인 NVRAM을 효율적으로 사용하게 하였다. NVRAM을 사용하는 기존의 연구와 달리 제한된 용량의 NVRAM을 효율적으로 사용하는데 연구의 초점을 두어 플래시 메모리 크기의 0.15%에 해당하는 크기의 NVRAM만을 사용하면서도 마운트 성능을 크게 개선할 수 있었다.

참고문헌

[1] Aleph One Company, “The Yet Another Flash Filing System (YAFFS)”, http://www.yaffs.net

[2] D. Woodhouse, “JFFS: The Journaling Flash File System”, Proceeding of the Ottawa Linux Symposium, RedHat inc. 2001.

[3] R. Card, T. Ts'o, and S. Tweedie, “Design and Implementation of the Second Extended Filesystem,” The HyperNews Linux KHG Discussion, http:/www.linuxdoc.org, 1999.

[4] A.-I.A. Wang et al., “Conquest: Better Performance through a Disk/Persistent-RAM Hybrid File System,” Proceedings 2002 USENIX Ann. Technical Conf., June 2002.

[5] Nathan K. Edel , Deepa Tuteja , Ethan L. Miller , Scott A. Brandt, “MRAMFS: A Compressing File System for Non-Volatile RAM”, Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), p.596-603, October 04-08, 2004.

[6] Ethan L. Miller , Scott A. Brandt , Darrell D. E. Long, “HeRMES: High-Performance Reliable MRAM-Enabled Storage”, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, p.95, May 20-22, 2001.

[7] Eun-ki Kim, Hyungjong Shin, Byung-gil Jeon, Seohhee Han, Jaemin Jung, Youjip Won, “FRASH: Hierarchical File System for FRAM and Flash”, ICCSA 2007, Malaysia, pp. 238-251, 2007

[8] In Hwan Doh, Jongmoo Choi, Donghee Lee, Sam H. Noh, “Exploiting non-volatile RAM to enhance flash file system performance”, Proceedings of the 7th ACM & IEEE international conference on Embedded software, 2007.

[9] Youngwoo Park, Seung-Ho Lim, Chul Lee, Kyu Ho Park, “PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash”, Proceedings of the 2008 ACM symposium on Applied computing, 2008.

[10] RAMTRON Datasheet http://www.ramtron.com/products/nonvolatile-memory/parallel.aspx

[11] Jean-loup Gailly and Mark Adler. Zlib http://www.gzip.org

[12] Markus F.X.J Oberhumer. LZO data compression library http://www.oberhumer.com/opensource/lzo/