비휘발성 메모리의 공간적 효율성을 고려한 파일 시스템의 설계 및 구현

(Design and Implementation of a File System that Considers the Space Efficiency of NVRAM)

현 철 승 † 백 승 재 †† 최 종 무 ††† 이 동 희 †††† 노 삼 혁 †††††
(Choulseung Hyun) (Seungjae Baek) (Jongmoo Choi) (Donghee Lee) (Sam H. Noh)

요 약 최근 차세대 메모리 기술이 급격히 발전하여 FeRAM과 PRAM과 같은 비휘발성 메모리의 상품화가 진행 중이다. 이러한 차세대 비휘발성 메모리(NVRAM)는 메모리와 저장 장치의 속성을 모두 만족시켜 데이타를 영속적으로 저장할 뿐 아니라 빠른 데이타 임의 접근을 가능하게 한다. NVRAM에 자주 변경되는 객체를 영속적으로 저장하기 위해서는 네이밍, 회복, 그리고 공간 관리와 같은 파일 시스템의 핵심 기능이 모두 필요하다. 그렇지만 기존 파일 시스템과 최근에 개발된 NVRAM 용 파일 시스템 모두 공간 효율이 낮으며, 어떤 경우 50% 정도에 불과하다. 따라서 상대적으로 고가인 NVRAM을 활용하기 위하여 공간 효율성이 뛰어난 익스텐트(extent) 기반의 NEBFS (NVRAM Extent-Based File System) 파일 시스템을 설계하였다. 그리고 기존 파일 시스템과 NEBFS의 공간 효율성을 비교 분석하였으며, 아울러 NEBFS를 구현하고 NVRAM이 탑재된 보드 및 NVRAM 에뮬레이션 환경에서 공간 효율성을 측정하여 분석 결과를 검증하였다. 이러한 실험 결과는 NEBFS의 공간 효율이 기존 파일 시스템보다 우수함을 보여 준다.

키워드 : 파일시스템, 비휘발성 메모리, 공간효율성

Abstract Nonvolatile memory technology is evolving continuously and commercial products such as FeRAM and PRAM are now challenging their markets. As NVRAM has properties of both memory and storage, it can store persistent data objects while allowing fast and random access. To utilize NVRAM for general purpose storing of frequently updated data across power disruptions, some essential features of the file system including naming, recovery, and space management are required while exploiting memory-like properties of NVRAM. Conventional file systems, including even recently developed NVRAM file systems, show very low space efficiency wasting more than 50% of the total space in some cases. To efficiently utilize the relatively expensive NVRAM, we design and implement a new extent-based space-thrifty file system, which we call NEBFS (NVRAM Extent-Based File System). We analyze and compare the space utilization of conventional file systems with NEBFS and validate the results with experimental results observed from running the file system implementations on a system with actual NVRAM installed as well as on systems emulating NVRAM. We show that NEBFS has high space efficiency compared to conventional file systems.

Key words : File System, Nonvolatile Memory, Space Efficiency

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

† 학생회원 : 서울시립대학교 컴퓨터통계학과
cshyun@venus.uos.ac.kr
†† 학생회원 : 단국대학교 정보컴퓨터공학부
ibanez1383@dankook.ac.kr
††† 종신회원 : 단국대학교 정보컴퓨터공학부 교수
choijm@dku.edu
†††† 정 회 원 : 서울시립대학교 컴퓨터과학부 교수
dhlee@venus.uos.ac.kr
(Corresponding author임)
††††† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
samhnoh@hongik.ac.kr
논문접수 : 2006년 5월 24일
심사완료 : 2006년 8월 17일

1. 서 론

NVRAM(Nonvolatile RAM)은 기존 RAM과 동일한 성질을 보이면서 디스크와 같이 비휘발성인 차세대 메모리이다. 최근 반도체 기술의 급격한 발전으로 인하여 FeRAM(Ferro-electric RAM), MRAM(Magneto-resistive RAM), PRAM (Phase-change RAM) 등 다양한 형태의 NVRAM이 개발되고 있다[1,2]. 이에 따라 운영체제의 메모리 계층 구조에 대한 변화와 기존 구조상에서 NVRAM을 활용하는 새로운 가능성에 대한 연구가 활발히 진행되고 있다[3-5]. NVRAM의 대용량화와 실용화 추세를 감안하면 대용량의 NVRAM에 정보를 체계적으로 저장하고 관리하는 기술이 필수적이라 하겠다.

지금까지 컴퓨터의 저장 장치로는 RAM과 같은 휘발성 메모리와 디스크와 같은 비휘발성 저장 장치가 널리 사용되고 있다. 휘발성 저장 장치를 효율적으로 사용하기 위해 버디(buddy)나 슬랩(slab)과 같은 기법들이 개발 되었으며[6], 비휘발성 저장 장치에 정보를 효과적으로 저장하기 위해서 다양한 파일 시스템들이 개발 되었다[7-12].

NVRAM은 비휘발성 저장 장치와 휘발성 RAM의 특성을 모두 가지고 있다. 그렇기 때문에 휘발성 저장 장치인 RAM처럼 작은 객체들을 효율적으로 관리할 수 있을 뿐만 아니라 비휘발성 저장 장치인 디스크처럼 큰 파일 데이타도 함께 NVRAM에 효율적으로 저장할 수 있어야 할 것이다. 아울러 당분간은 가격/용량이 상당히 높을 것으로 예상됨에 따라 NVRAM 공간을 최대한 효율적으로 활용하는 관리 기법의 개발이 매우 시급하고 중요하다고 할 수 있다.

본 논문에서는 NVRAM 공간을 효율적으로 활용하면서 NVRAM의 특성 및 장점을 고려하는 익스텐트(extent) 기반의 NEBFS (NVRAM Extent-Based File System) 파일 시스템을 설계하였다. 그리고 NEBFS의 공간 효율성의 우수성을 보이기 위해 공간 비용 분석 기법을 사용하여 기존 파일 시스템들과 공간 효율성을 비교한다. 뿐만 아니라 이들을 리눅스 상에 구현하여 실험을 통한 비교도 수행한다. 실험은 NVRAM의 일종인 FeRAM을 실제 탑재한 EZ-X5[13]에서 수행할 뿐만 아니라 다양한 파일 시스템과 비교를 가능하게 하기 위하여 NVRAM을 에뮬레이션하는 데스크 탑 환경에서도 수행한다. 실험 결과 NEBFS가 공간 효율성 면에서 매우 우수함을 확인할 수 있었다.

이하 논문의 구성은 다음과 같다. 2장에서는 본 연구와 관련된 NVRAM에 대한 과거 연구들을 정리한다. 3장에서는 NVRAM을 지원하는 파일 시스템의 요구 사항들과 함께 제안하는 파일 시스템에 대한 설계에 대해 구체적으로 설명한다. 4장에서는 NEBFS를 비롯하여 기존 파일 시스템들에 대한 공간 효율성을 분석 비교 한다. 5장에서는 NEBFS의 구현과 함께 실험 결과에 대해 설명한다. 마지막으로 6장에서 결론과 함께 향후 연구에 대해 언급하고 마무리한다.

2. 관련 연구

NVRAM이 컴퓨터 시스템에 추가되었을 때 이를 어떻게 활용할 것인가에 대한 연구는 시스템 복구, DRAM과 NVRAM이 동시에 존재하는 시스템에서의 캐시 관리 정책, 파일 시스템에서 캐시로 사용한 경우, 일반 디스크와 동시에 활용되는 하이브리드 형태의 경우, 또는 NVRAM을 저장 장치로 사용한 경우 등 다양하게 진행되었다.

Baker와 Sullivan은 시스템의 다운타임을 줄임으로써 시스템의 가용성을 증가시키는 연구를 수행하였다[14]. 이 논문에서는 안정적인 메모리에 시스템의 상태정보를 저장한 후 분산파일 시스템과 POSTGRES 데이타베이스 시스템에 문제가 발생하였을 때 이 정보를 이용하여 빠르게 복구할 수 있음을 보였다. 또한, 일반적으로 파일 시스템에서 속도 향상을 위해 휘발성 메모리를 읽기 및 쓰기 캐시로 사용하게 된다. 캐시를 사용하면 속도 측면에서 많은 이득을 볼 수 있으나 갑작스런 전원 정지 같은 이유로 시스템이 붕괴되었을 때 파일 시스템의 일관성 유지를 위한 복구에 많은 시간이 필요하게 된다. Rio File Cache는 NVRAM을 쓰기 캐시로 사용하여 위와 같이 시스템이 붕괴되었을 때 데이타 일관성을 유지하기 위한 복구를 효과적으로 수행할 수 있음을 보였다[15].

컴퓨터 시스템에 휘발성 메모리와 비휘발성 메모리가 같이 존재하였을 때 캐시 관리정책에 대한 연구가 수행되었다[16,17]. 또한 분산 파일 시스템에서 NVRAM을 쓰기 캐시로 사용하는 상황에 대한 연구를 진행하여 클라이언트에서 서버로 요청되는 쓰기 트래픽을 효과적으로 줄일 수 있으며, 서버에서 디스크 접근을 효율적으로 관리할 수 있음을 보인 바 있다[18].

NVRAM을 다른 미디어와 함께 저장 장치로 사용한 연구로서 Conquest[19]와 HeRMES[20]가 있다. 이 두 논문에서는 NVRAM과 디스크를 같이 사용하는 파일 시스템을 제안하였다. 일반적으로 파일 시스템에 존재하는 파일은 대부분 크기가 작으며, 반면 디스크 용량의 대부분을 차지하는 것은 크기가 큰 파일이라는 정보[21]를 이용하여 파일 시스템에서 자주 접근되는 메타 데이타와 크기가 작은 파일은 NVRAM에 저장하고 크기가 큰 파일은 디스크에 저장하는 방법을 제안하고 있다.

마지막으로 NVRAM만을 저장 장치로 활용하기 위한 연구는 MRAMFS[22]와 PRAMFS[23]이 있다. MRAMFS

에서는 저장되는 정보를 압축하여 용량이 작은 NVRAM의 공간을 효율적으로 사용하는 방법을 제안하였다. 디렉토리들은 해시테이블로 구현되며, 디렉토리 엔트리 객체는 파일의 이름을 표현하기 위하여 고정된 길이의 필드를 포함한다. Inode의 구현은 2-레벨 테이블로 구성된다. 파일 시스템 생성시 1-레벨 테이블(inode 테이블 리스트)은 생성되어 있고, 2-레벨 테이블(inode 블록 테이블)은 필요할 때마다 할당된다. 하나의 inode 블록 테이블에 inode는 1024개까지 포함될 수 있으며, 각 inode는 디렉토리를 가리키는 포인터와 데이타 블록을 가리키는 블록 포인터를 포함하며 최소 256바이트의 길이를 가진다. MRAMFS 논문에 따르면 CPU에 비해 NVRAM으로의 I/O가 상대적으로 느리기 때문에 CPU를 사용하는 압축 및 해제 작업으로 인한 성능 저하는 거의 없으며, 이를 EXT2와 JFFS 파일 시스템과의 비교를 통해 보이고 있다. 그렇지만 압축 및 해제를 하지 않은 경우보다 높은 CPU 이용률을 보이는데, 이는 압축을 하지 않는 파일 시스템과 비교하였을 때 보다 많은 전력을 소비하게 된다. 또한 압축된 데이타의 복사가 수행되고 파일을 탐색하게 되면 일반적인 RAM 기반 파일 시스템보다 성능이 저하될 수 있다. 아울러 압축을 하여 저장하더라도 NVRAM을 블록 장치처럼 사용한다면 블록 크기에 따라 내부 단편화가 발생할 수 있는 문제점을 갖고 있다.

PRAMFS는 EXT2 파일 시스템에서 하나의 블록그룹을 파일 시스템 전체로 확장한 형태의 파일 시스템이다. PRAMFS는 데이타 블록을 가리키기 위해 EXT2 파일 시스템의 디렉토리 엔트리에서 간접 블록과 같은 인덱스 블록을 할당 받고 이 블록이 실제 데이타 블록을 가리키는 구조를 사용한다. 이 경우 인덱스 블록의 낭비로 인해 공간 효율성이 많이 떨어지게 되는 단점이 있다.

3. NEBFS 구조 설계

NVRAM을 블록 저장 장치로 보고 그 상위에 전통적인 파일 시스템을 사용한다면 아주 손쉽게 NVRAM을 저장 장치로 활용할 수 있을 것이다. 그러나 이러한 접근 방법은 디스크의 특성에 최적화되어 있는 파일 시스템을 사용함으로써 NVRAM의 장점을 최대로 활용할 수 없는 문제점이 있다. 또한 기존의 RAM 관리 기법인 버디나 슬랩을 사용할 경우 외부 단편화로 인해 공간 효율성이 54%에서 86% 정도로 매우 낮다는 문제점을 안고 있다[6]. 이 장에서는 NVRAM을 활용하는 파일 시스템이 갖추어야 할 요건들을 나열하고 이를 근거로 개발된 NEBFS (NVRAM Extent-Based File System)의 구조에 대해 구체적으로 설명한다.

3.1 NVRAM을 이용하는 파일 시스템의 요구 조건

NVRAM은 일반적인 메모리의 특성을 가지면서 디스크처럼 데이타의 영속적인 저장이 가능한 비휘발성 메모리이다. 따라서 NVRAM을 효과적으로 사용하기 위해서는 기존 RAM 관리 기법의 특성을 가지면서도 영속적 데이타 저장을 위해 만들어진 파일 시스템의 특성도 함께 갖는 파일 시스템 구조가 필요하다.

NVRAM을 기반으로 하는 파일 시스템은 다음과 같은 속성을 가져야 한다. 첫째, 기존 파일 시스템의 모든 속성을 제공할 뿐 아니라 메모리 관리 기법처럼 다양한 크기의 객체를 효율적으로 저장할 수 있어야 한다. 둘째, 공간 효율성과 성능 면에서 트레이드오프를 제공하여 비휘발성 메모리 기술 발전에 대처 할 수 있어야 한다. 현재 NVRAM은 개발 초기이기에 기존 저장 장치에 비해 가격/용량 면에서 비효율적이지만 반도체 기술의 지속적인 발전에 따라 NVRAM의 대용량화를 예상할 수 있다. 따라서 이러한 비약적 발전에 대비할 수 있도록 설계되어야 한다. 셋째, 현재 사용되고 있는 메모리와 파일 시스템의 활용 형태를 보면 작은 크기의 객체 또는 파일의 개수가 매우 많고 상대적으로 소수의 대용량 파일이 존재하는 것으로 알려져 있다[21]. 이에 따라, 다수의 작은 파일을 저장할 때와 큰 파일을 저장할 때 모두 공간 효율성이 높아야 한다.

본 논문에서는 기존 파일 시스템이 얼마나 효율적으로 다양 크기의 객체를 표현하고 있는지 분석하고, 분석 결과를 통해 기존 파일 시스템 중 공간 효율성이 뛰어난 FAT 파일 시스템을 기반으로 성능과 공간 효율성 간에 트레이드오프를 제공하는 NEBFS를 설계하였으며 그 특성은 다음과 같다.

가. 최소한의 할당 단위를 사용한 익스텐트(extent) 기반의 할당

NVRAM은 RAM의 특성을 가지고 있기 때문에 바이트 단위로 접근할 수 있다. 이러한 장점을 활용하기 위해 블록을 가변적으로 할당한다면 내부단편화가 발생하지 않는다. 그러나 그 관리가 어렵고 외부단편화에 의해 저장 장치 이용률은 심각하게 떨어지게 된다[6]. 반면, 고정된 블록 단위로 할당한다면 내부 단편화가 발생하게 된다. 만약 고정된 블록 단위로 공간을 할당하면서 최소 할당 단위를 작게 하면 내부 단편화로 인한 낭비는 줄어들지만 관리를 위한 메타 데이타 양은 증가한다. 반대로 최소 할당 단위를 크게 하면 메타 데이타 양은 감소하지만 내부 단편화로 낭비되는 양은 증가한다.

NEBFS는 가변 블록의 장점과 고정 블록 할당의 장점을 취하기 위하여 블록 단위로 공간을 할당하면서 아울러 최소 할당 단위를 매우 작게 설정할 수 있도록 하였다. 그리고 최소 할당 단위를 작게 하면서 증가하는

메타 데이타 양을 감소시키기 위하여 익스텐트 기반 할당 정책을 사용하였다[7,11,12]. 익스텐트 기반 할당 정책은 메타 데이타를 표현할 때 연속적인 블록을 하나의 익스텐트로 표현하기 때문에 단편화 정도를 잘 관리하면 파일 시스템의 메타 데이타 양을 감소시키는 데 도움이 된다.

나. 파일 시스템에서 고정된 메타 데이타의 최소화

디스크 기반 파일 시스템에서는 I/O의 최적화를 위해 파일 시스템을 포맷할 때 메타 데이타를 위한 공간을 미리 확보한다. 그러나 NVRAM인 경우 디스크에 비해 I/O 절차가 단순하고 빠르기 때문에 NEBFS에서는 파일 시스템을 포맷할 때 파일 시스템 구조를 유지하기 위한 최소한의 메타 데이타만 확보하며 파일을 위해 필요한 메타 데이타는 필요할 때마다 할당 한다.

다. 메타 데이타의 집중 관리

메타 데이타는 파일의 속성 정보와 파일 데이타 블록을 가리키기 위한 인덱스 정보로 구성된다. 이 두 종류의 메타 데이타를 한 곳으로 모으면 파일과 관련된 모든 정보를 한 번에 접근할 수 있는 장점을 얻을 수 있다. 예를 들어 메타 데이타가 저장 장치의 여러 부분에 흩어져 저장되어 있는 것보다 일정 영역에 존재한다면 파일 데이타보다 접근이 많이 이루어지는 메타 데이타의 특성상 저장 장치 접근 횟수를 크게 줄이게 된다. 이러한 장점은 RAM의 속성을 가지면서도 블록 단위로 접근해야 하는 NVRAM용 파일 시스템에서도 나타나며, NEBFS는 파일을 접근하는데 필요한 메타 데이타들이 한 곳으로 모아 한 번에 접근할 수 있도록 설계하였다.

라. 성능과 공간 효율성의 트레이드오프

파일 시스템의 성능을 높이기 위해서는 저장 장치의 기본 접근 단위인 블록의 크기를 크게 하면 된다. 그러나 블록의 크기를 크게 하면 NVRAM의 낭비가 증가하여 공간 효율성이 떨어지게 된다. NEBFS에서는 사용목적에 따라 유동적으로 블록의 크기를 조절할 수 있도록 하여 성능과 공간 효율성 간에 우선순위를 조절할 수 있도록 설계하였다.

3.2 NEBFS의 내부 구조

NEBFS의 내부 구조는 FAT 파일 시스템과 유사하다. FAT 파일 시스템은 MBR(Master Boot Record)에 해당되는 섹터 다음에 FAT(File Allocation Table) 테이블, 그 다음에 루트 디렉토리가 존재한다. NEBFS는 FAT 파일 시스템에서 FAT 테이블을 비트맵으로 대체한 구조이다. 또한 FAT 파일 시스템에서는 파일 데이타 블록을 가리키기 위한 정보를 FAT 테이블이 가지고 있지만, NEBFS에서는 이러한 정보를 디렉토리 내의 엔트리에 파일의 속성 정보와 함께 익스텐트 형태로 저장한다. 이러한 구조는 초기 메타 데이타를 최소화 시킬 수 있고, 메타 데이타 정보를 한 곳으로 집중시킬 수 있게 한다.

NEBFS에서 디렉토리 엔트리의 크기는 64바이트이다. 디렉토리 엔트리는 파일 이름, 파일의 속성 정보, 그리고 파일 데이타 블록을 가리키는 익스텐트들로 구성된다. 속성은 기존 파일 시스템의 것과 차이가 없으며 크기는 30바이트이다. 익스텐트는 4바이트 블록 번호와 2바이트 오프셋으로 구성되며, 디렉토리 엔트리에 총 5개의 익스텐트가 존재한다. 익스텐트는 지정된 블록 번호로부터 시작하여 연속으로 몇 개의 블록이 할당되었는지를 가리킨다. 만약 한 블록이 128바이트이고 블록이 모두 연속으로 할당된다면 하나의 익스텐트는 최대 8M바이트의 공간을 가리킬 수 있으며, 따라서 하나의 디렉토리 엔트리는 최대 40M바이트의 블록을 가리킬 수 있다. 파일이나 서브디렉토리가 커져 하나의 디렉토리 엔트리로 표현하지 못하면 추가로 인덱스 엔트리를 할당받아 확장할 수 있으며, 디렉토리 엔트리와 인덱스 엔트리들은 각 엔트리의 마지막 4바이트 필드를 사용하여 체인을 형성한다. 예를 들어 그림 1에서 File B의 마지막 4바이트에 '6'이라는 값이 있는데, 이는 추가 엔트리 위치 6을 가리키며, 6번째 엔트리에는 File B의 인덱스 엔트리가 존재한다. 추가 인덱스 엔트리는 총 9개의 익스텐트를 가져 최대 표현 범위는 72M 바이트가 된다. 인덱스 엔트리의 나머지 바이트들은 속성 정보를 표현하기 위한 1 바이트, 다음 인덱스 엔트리를 가리키는 마지막 4 바이트, 그리고 예약 영역인 나머지 바이트들로 구성된다.

이러한 구조는 블록 하나 크기의 공간부터 익스텐트가 허용하는 최대 크기까지의 공간을 하나의 익스텐트로 표현하기 때문에 가상적으로 가변 크기의 블록을 할당하는 것과 같은 효과를 낸다. 또한 용량이 큰 파일이

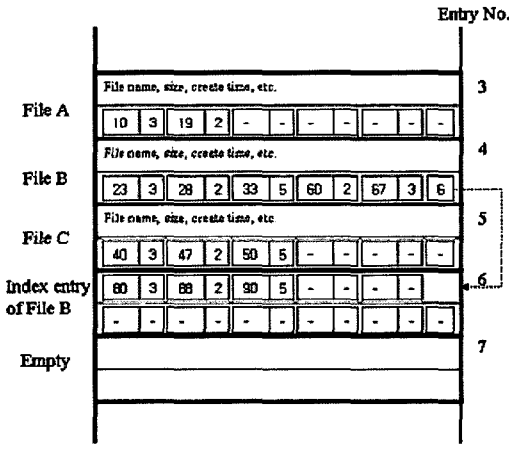


그림 1 NEBFS 내부구조

나 작은 파일 모두 효과적으로 기록할 수 있는 장점이 있다.

4. 파일 시스템의 공간 효율성 분석

파일 시스템에서 파일을 생성하는데 드는 비용은 수식 (1)에서 표현한 것과 같이 고정 비용(C_f)와 가변 비용(C_v)로 나눌 수 있다.

$$C = C_f + C_v \quad (1)$$

C_f는 파일 시스템을 처음 구축할 때 고정적으로 드는 공간 비용이고 C_v는 파일을 생성할 때마다 가변적으로 소모되는 공간이다. C_v는 파일을 관리하기 위한 메타 데이타와 파일 데이타 자체를 저장하기 위해 필요한 공간으로 구성된다. NEBFS의 경우 MBR과 비트맵 블록 이외에는 별도의 C_f가 존재하지 않는다. 따라서 전체 비용 C를 통해 파일 생성에 필요한 비용을 파일 시스템별로 비교하면 초기 C_f가 작은 NEBFS의 비용이 대부분 좋게 나온다. 따라서 기존 파일 시스템과 NEBFS를 비교하기 위하여 C_f의 비용을 각 파일 당 비용으로 변환하여 비교하도록 한다[24].

4.1 파일 시스템의 공간 비용 분석

NEBFS에서 파일이 하나 생성될 때 필요한 공간 비용 모델은 식 (2)와 같은 형태로 표현할 수 있다.

$$C = D + Bitmap + File \quad (2)$$

$D = Ceiling((\lceil Ac / E_{avg} + 5 \rceil / 9) \times 64, B) / N_d$

단, $1 < Ac \leq 5$ 일 경우 $D = B$

Ceiling(X,Y): X를 Y의 배수에 가장 가까운 정수로 올림하는 함수

A_c : 파일에서 사용한 블록 수

Bitmap: $\lceil A_c / 8 \rceil$

B: 블록 크기

File: $B \times A_c$

E_{avg}: 평균 익스텐트 크기

N_d: 동일한 디렉토리 내에 존재하는 파일의 수

NEBFS에서 하나의 파일을 생성할 때 비용은 디렉토리 비용과 비트맵 비용, 그리고 파일 데이타로 나타낼 수 있다. 위 수식에서 A_c는 파일에서 사용한 블록 수이고 E_{avg}는 평균적으로 하나의 익스텐트가 표현하는 블록 개수를 나타내며, B는 블록 크기이다. 그리고 N_d는 같은 디렉토리에 포함되어 있는 파일의 개수이다. NEBFS에서 파일이 생성되면 디렉토리 엔트리가 할당되며, 디렉토리 엔트리에는 파일 이름, 파일 속성 필드, 그리고 파일 데이타 블록을 가리킬 수 있는 익스텐트가 5개가 존재한다. 파일이 커져 추가의 익스텐트가 필요하게 되면 익스텐트 필드가 9개인 64 바이트 크기의 인덱스 엔트리를 할당하게 된다. 따라서 디렉토리 비용을 위와 같이 나타낼 수 있다.

또한 파일 시스템 분석을 통해 수식으로 표현한 FAT 파일 시스템의 공간 비용 모델은 다음과 같다.

$$C = D + FAT + File \quad (3)$$

$D = Ceiling(N_d \times 32, B) / N_d$

FAT = FAT size $\times A_c$

FAT 파일 시스템의 비용은 디렉토리 엔트리 비용과 FAT엔트리 비용, 그리고 파일 데이타로 구성된다. FAT size는 FAT 테이블에서 FAT 엔트리의 크기를 나타낸다. 예를 들어 FAT12이면 12비트이고 FAT16이면 16비트이며 FAT32일 때는 32비트이다. FAT 파일 시스템에서 하나의 디렉토리 블록을 다수의 파일/서브디렉토리가 공유하고 하나의 디렉토리 엔트리의 크기는 32 바이트이므로, 한 파일이 차지하는 디렉토리 공간은 디렉토리 블록의 크기를 디렉토리에 존재하는 파일/서브디렉토리 개수(사용 중인 디렉토리 엔트리 개수)로 나누면 구할 수 있다.

다음으로 EXT2 파일 시스템의 비용 모델은 다음과 같다.

$$C = D + Inode + Bitmap + F/Nf + File \quad (4)$$

$D = Ceiling(Sum(Ceiling(4, 8+namelen(i))), B) / N_d$

Inode = 128 bytes

F / N_f: 고정비용

EXT2 파일 시스템의 비용은 디렉토리 비용과 inode 비용, 비트맵 비용, 고정 비용, 그리고 파일 데이타를 위한 비용으로 나타낼 수 있다. EXT2 파일 시스템에서는 하나의 파일이나 디렉토리가 생성되었을 때 메타 데이타를 저장하기 위해 하나의 inode를 할당한다. 이 inode의 크기는 128 바이트로 고정되어 있다. 디렉토리 비용은 이름의 길이에 따라 달라지게 되며, 4 바이트로 배치되기 때문에 위에서 표현한 수식과 같이 나타낼 수 있다.

F/N_f는 파일 시스템을 생성할 때 지불된 EXT2 파일 시스템의 고정 비용을 파일 시스템 내에 존재하는 파일의 개수로 나눈 값이다. EXT2 파일 시스템은 다른 파일 시스템에 비하여 고정비용이 크다. 이 비용은 파일에 대한 접근을 효율적으로 하기 위한 비용과 시스템이 비정상적으로 종료되었을 때 파일 시스템을 복구를 위한 데이타를 저장하기 위한 비용이다.

마지막으로 PRAMFS의 공간 비용 모델은 다음과 같다.

$$C = D + Bitmap + File \quad (5)$$

D = Inode + index block

PRAMFS는 EXT2 파일 시스템에서 하나의 블록그룹을 파일 시스템 전체로 확장한 구조이다. 그렇기 때문에 하나의 파일을 생성할 때 역시 하나의 inode가 할당된다. 또한 블록 지정 방식으로 EXT2 파일 시스템과 유사하게 간접 블록을 사용하므로 디렉토리 비용을 Inode와 인덱스 블록의 합으로 나타낼 수 있다.

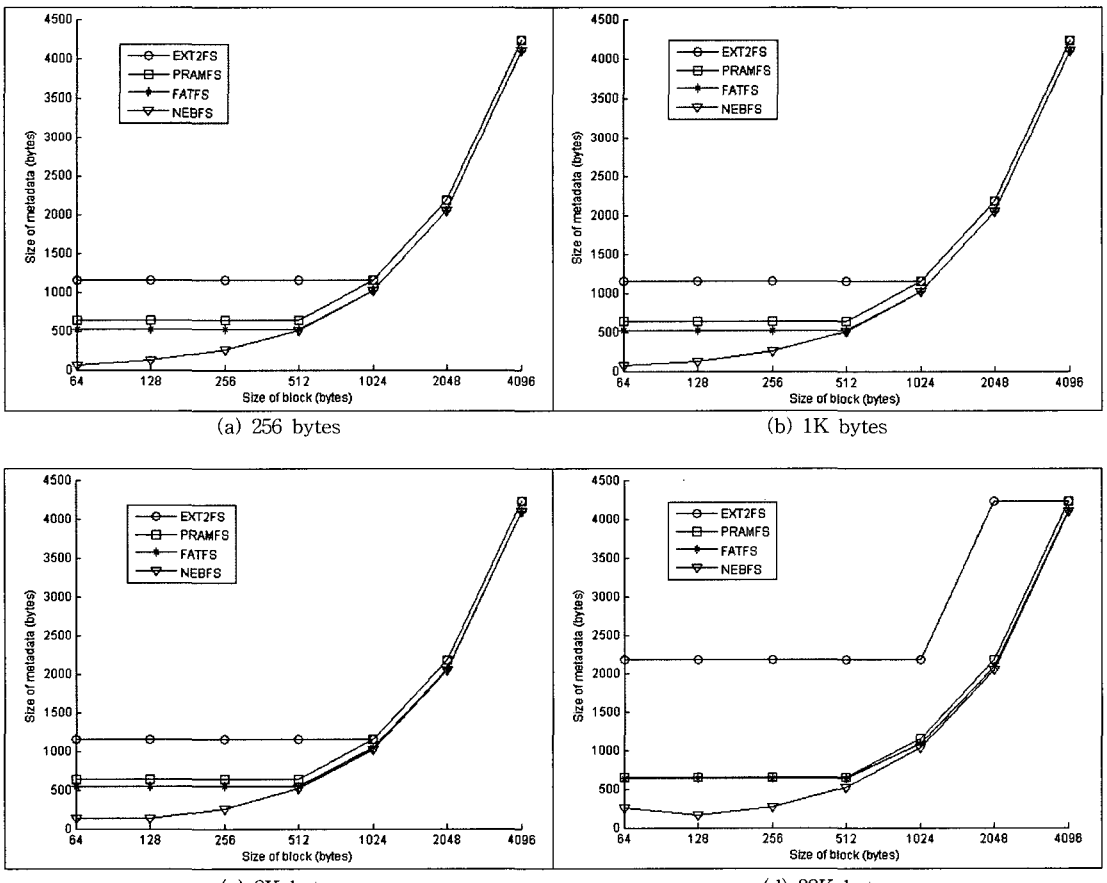
4.2 파일 시스템의 공간 비용 비교

4.1절의 수식 (2), (3), (4), (5)에서 계산된 파일 비용 모델을 이용하여 파일 시스템에서 파일 생성에 따르는

비용을 평가하였다. 블록 기반 파일 시스템에서 마지막 블록이 낭비되는 크기는 모두 동일하기 때문에 이는 무시하였으며, 메타 데이타의 비용 즉, 디렉토리 비용을 비교하였다. 또한 각 파일 시스템마다 지원하는 이름의 길이가 다르기 때문에 비교에서는 이름의 길이를 11 바이트로 설정하였으며, FAT 파일 시스템에서 FAT 엔트리 크기는 FAT16으로 가정하고 비교를 수행하였다. 그림 2는 비교 결과를 보여준다.

그림 2에서 보이고 있는 4개의 그래프는 각각 파일 크기를 256B~32KB까지 변화시켰을 때, 블록크기의 변화에 따른 비용 그래프이다. Y축의 비용은 파일을 생성할 때 필요한 메타 데이타 공간, 즉 D 값을 나타낸다. 그래프에서 볼 수 있듯이 파일 생성 시 공간 효율성은 NEBFS, FAT, PRAMFS, EXT2의 순으로 좋다. 또한 파일 크기가 변함에 따라 EXT2 파일 시스템의 비용이 다른 파일 시스템보다 높아지는 현상을 보인다. 이는 파일의 크기가 커짐에 따라 EXT2 파일 시스템은 데이타 블록을 가리키기 위한 인덱스를 확보하기 위해 간접블록, 2중/3중 간접블록을 사용하게 되는데 이 비용이 추가되기 때문이다. 그래프 상에서 PRAMFS가 FAT이나 NEBFS보다 좋지 않은 이유 역시 간접블록이 하나의 파일이 생성될 때마다 할당되기 때문이다. 다시 말해 PRAMFS에서 파일마다 파일 데이타의 인덱싱을 위한 블록이 할당되므로, 하나의 파일마다 파일 데이타의 마지막 블록과 인덱싱을 위한 블록이 평균적으로 반씩 낭비되어 전체적으로 하나의 블록이 낭비되는 것과 같다.

FAT 파일 시스템과 NEBFS를 비교하였을 때 디렉토리 엔트리 자체의 크기는 FAT이 NEBFS보다 작다. 그리고 FAT 파일 시스템에서 FAT 엔트리의 사용 개수는 파일의 크기에 따라 달라지는데, FAT 엔트리의 사용 개수가 NEBFS에서 익스텐트로 표현하는 데이타 블록 개수보다 작을 때 FAT 파일 시스템이 더 좋은 효율성을 보일 수 있다. 그러나 FAT 파일 시스템에서 클러스터 크기를 작게 하면 FAT의 크기가 증가하여 더 많은 고정비용을 소모하게 되므로, 파일 시스템 전체로 봤을 때 비효율적이 된다.



(a) 256 bytes (b) 1K bytes

(c) 8K bytes (d) 32K bytes

그림 2 비용 분석 모델에 따른 파일 시스템 별 메타 데이타 공간 활용 비교

NEBFS는 수식(2)에서 확인 할 수 있듯이 익스텐트의 크기가 커질수록, 즉 E_{avg}의 값이 커질수록 메타 데이타의 양이 줄어들게 된다. E_{avg}를 일정한 수준으로 유지시켜 준다면, 공간 효율성 면에서 NEBFS 구조가 최적의 선택이 된다. 이렇게 NEBFS의 경우 E_{avg}를 적절한 크기 이상으로 유지하는 것이 공간 효율성과 성능에 중요한 영향을 미치며, 따라서 단편화를 적절히 제어하여 E_{avg}를 적절한 크기로 유지할 필요가 있다.

5. 구현 및 실험 결과

NEBFS를 리눅스에서 구현하여 기존 파일 시스템들과 비교 평가하였다. 비교는 크게 세 개의 플랫폼에서 수행되었다. 우선 그림 3에서 보여진 것과 같이 NVRAM의 한 종류인 FeRAM이 탑재되어 있는 EZ-X5 테스트 보드 상에서 실험하였고, 실제 FeRAM 대신 같은 보드 상에서 FeRAM을 SDRAM으로 에뮬레이트하면서 실험하였으며, 마지막으로 호스트 컴퓨터 상에서 SDRAM으로 FeRAM을 에뮬레이트하면서 실험하였다. 이렇게 세 가지 플랫폼에서 구현하고 실험한 이유는 보드에 FeRAM을 2M 바이트만 탑재할 수 있는 상황이어서 다양한 실험을 할 수 없었기 때문이다. 따라서 실제 FeRAM에 탑재된 보드에서의 실험은 본 구현의 신뢰성과 이후 에뮬레이션 환경에서 실시한 실험의 신뢰도를 검증하기 위해서 실시되었다.

그림 4는 FeRAM이 2M 바이트일 때 위 세 가지 실험 환경에서 실시한 실험 결과를 보여주고 있다. X축은 NEBFS 실험 시 활용한 블록의 크기를 나타내며 y축은 메모리 이용률로 UNIX의 df 명령을 이용하여 측정하였다. 이용률이 낮을수록 파일 시스템이 NVRAM을 효율적으로 활용했다고 해석할 수 있다. 실험은 Postmark 벤치마크를 사용하여 수행되었으며, 벤치마크의 설정 값은 파일의 크기가 500바이트에서 20K바이트, 그리고 평균 파일 크기가 10K인 파일을 초기에 100개를 생성한 후 100번의 트랜잭션을 수행하도록 하였다.

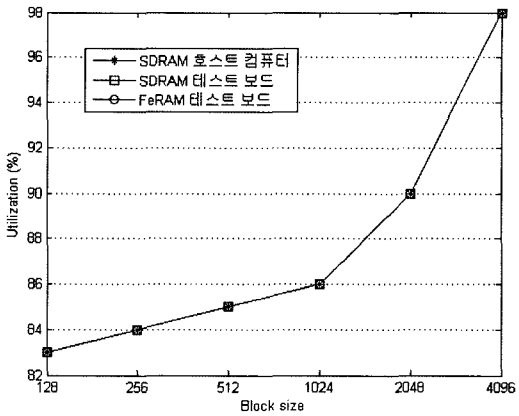


그림 4 세 가지 실험 환경 즉, FeRAM 탑재 보드, SDRAM 탑재 보드, SDRAM 호스트에서의 실험 결과 비교

그림 4에서 보듯이 세 가지의 실험 환경 모두에서 완전히 일치하는 공간 효율성을 보이고 있어 선이 마치 하나처럼 보인다. 또한, 별도의 그래프로는 보이고 있지 않지만 세 가지의 실험 환경 모두 1초 이내에 수행이 완료되는 사실을 관찰 할 수 있었다. 이렇게 구현된 NEBFS가 세 가지 환경 모두에서 일치되는 결과를 보임에 따라 이후의 실험은 호스트에서 128MB의 SDRAM으로 FeRAM을 에뮬레이션하는 환경에서 실행되었다.

그림 5는 128M 바이트 FeRAM을 에뮬레이션하는 호스트 환경에서 0.5K~4K 바이트까지 블록의 크기를 변화시키면서 실험한 결과이다. 실험에서 파일 크기를 500~20K 바이트로 하고 초기 500개 파일을 생성한 후 15000번의 트랜잭션을 수행하도록 Postmark를 설정하였다.

블록 크기를 0.5K~4K 바이트로 변경할 때 모든 블록 크기에서 NEBFS가 다른 파일 시스템에 비해 좋은 공간 효율성을 보이고 있으며, 블록 크기가 커질수록 공간 효율성의 차이가 커지고 있음을 확인할 수 있다. 결과에서 일부 나타내지 못한 점들은 이들의 NVRAM 이

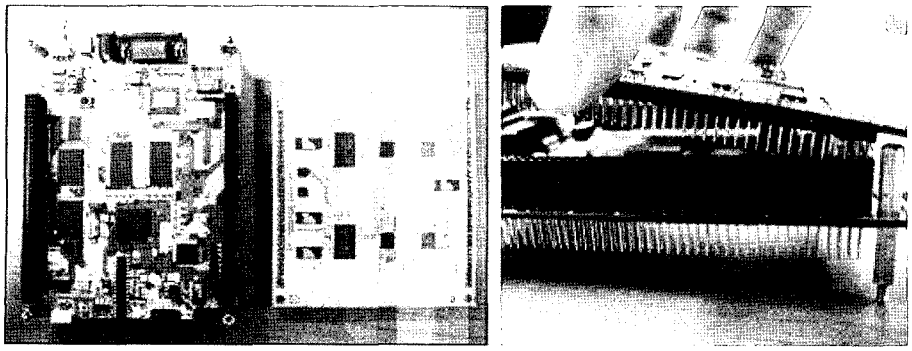
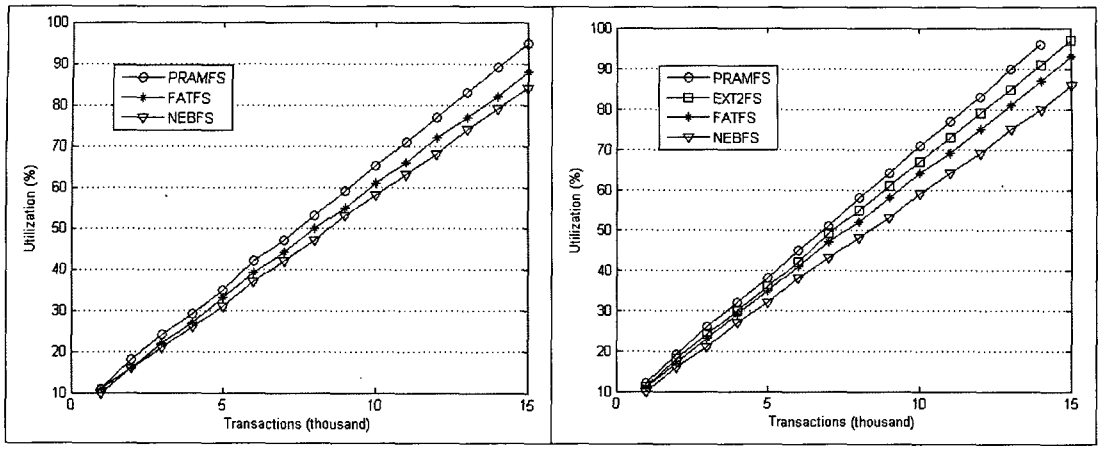
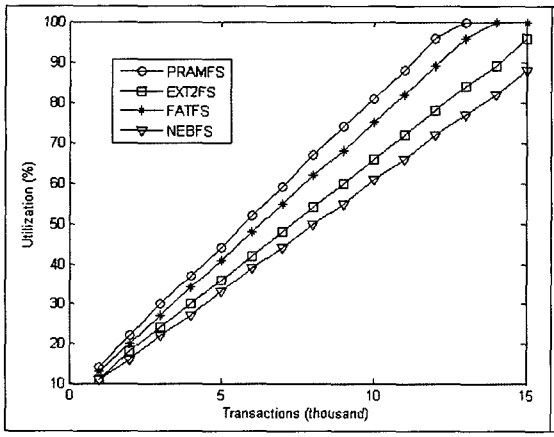


그림 3 EZ-X5 보드와 FeRAM 보드

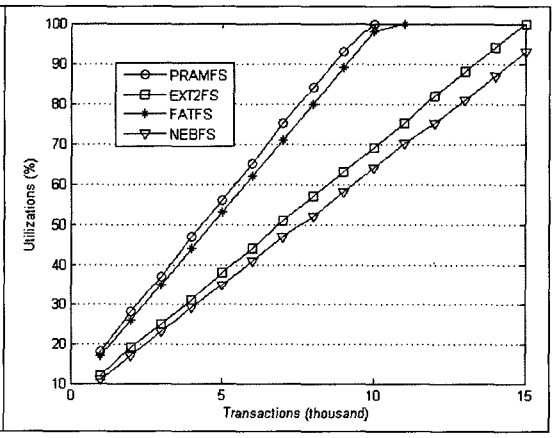


(a) 0.5K

(b) 1K



(c) 2K



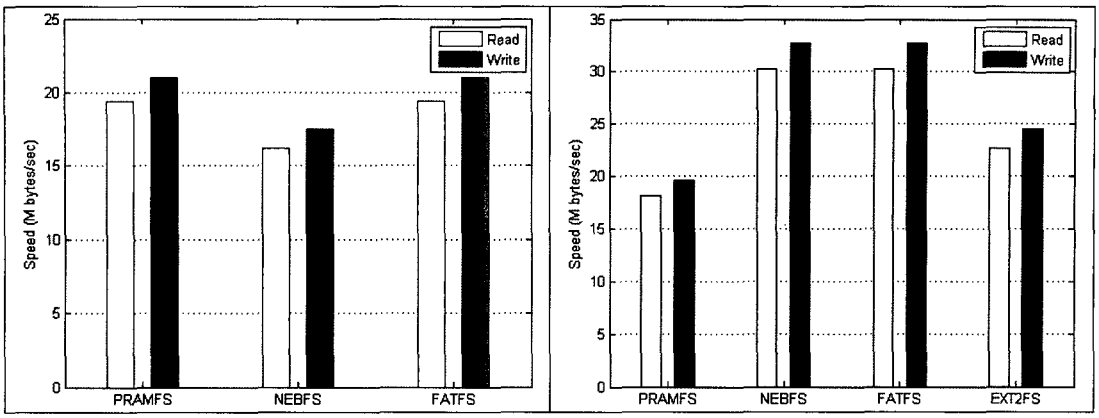
(d) 4K

그림 5 블록 크기에 따른 파일 시스템의 공간 효율성 비교

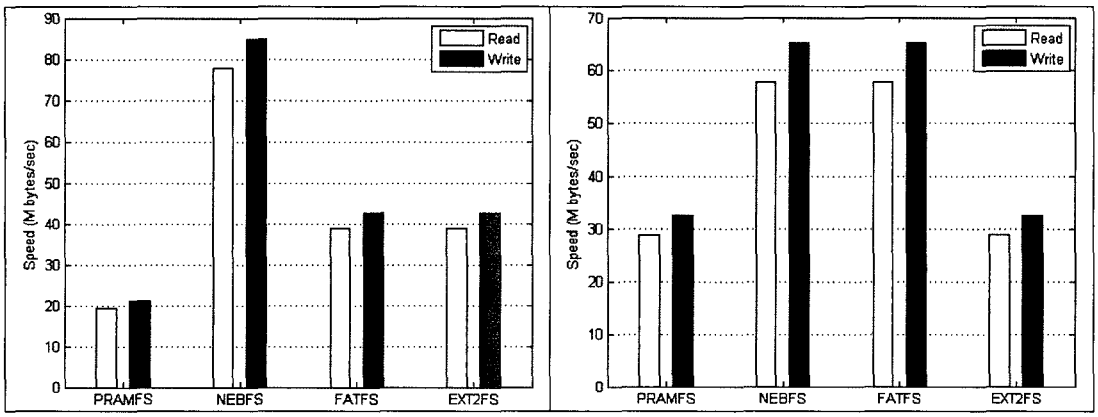
용률이 100%에 도달하면서 실험을 더 이상 진행할 수 없는 상황에 이르렀기 때문이다. 또한 EXT2에서 지원하는 최소 블록 크기가 1K 바이트이기 때문에 블록 크기가 0.5K인 그림 5(a)에는 EXT2 파일 시스템의 결과를 보이지 않았다. 뿐만 아니라 NEBFS를 제외한 다른 기존 파일 시스템에서는 256 바이트 이하의 블록 크기는 지원하지 못하는 실정이다.

PRAMFS인 경우, 블록의 크기가 커질수록 공간 효율성이 급격하게 떨어지는데 이는 파일의 개수가 늘어날수록 파일 데이타를 가리키기 위해 필요한 간접블록의 개수도 증가하기 때문이다. 또한 FAT 파일 시스템과 EXT2 파일 시스템의 경우 블록 크기가 1K 바이트인 경우 FAT 파일 시스템이 좋은 공간 효율성을 보여주고 있지만, 블록 크기가 2K 바이트 이상이 되면 EXT2 파일 시스템이 더 좋은 공간 효율성을 보이고 있다. 이는 분석에 의해 도출된 그림 2와 상반되는 결과이다. 이러한 상반된 결과에 대한 정확한 파악은 아직 하지 못하고 있는 상황이나 다음과 같은 이유에서 도래되었을 것으로 추측하고 있다. FFS[10]나 EXT2 파일 시스템의 경우 블록의 크기가 2K 이상이면 다수의 파일의 마지막 데이타들은 동일한 블록을 공유하여 저장된다. 이렇게 블록을 다수의 파일이 공유하는 기능이 활성화되면서 FAT 파일 시스템보다 EXT2가 더 좋은 공간 효율성을 보이고 있지 않나 추측된다. 앞으로 이에 대해 분석이 필요하며 검증을 해 볼 예정이다.

그림 6은 에뮬레이션 환경에서 측정한 각 파일 시스템의 수행 속도 결과를 보여준다. 수행 속도는 NEBFS, FATFS, EXT2FS, PRAMFS의 순으로 좋은 것으로 나타났다. 파일 시스템의 성능은 알고리즘의 최적화 정도에 많은 영향을 받으며, 다양한 최적화를 적용하여 NEBFS의 성능을 매우 향상시킬 수 있었다. 그리고 모든 파일 시스템에서 공통적으로 블록의 크기가 커지면

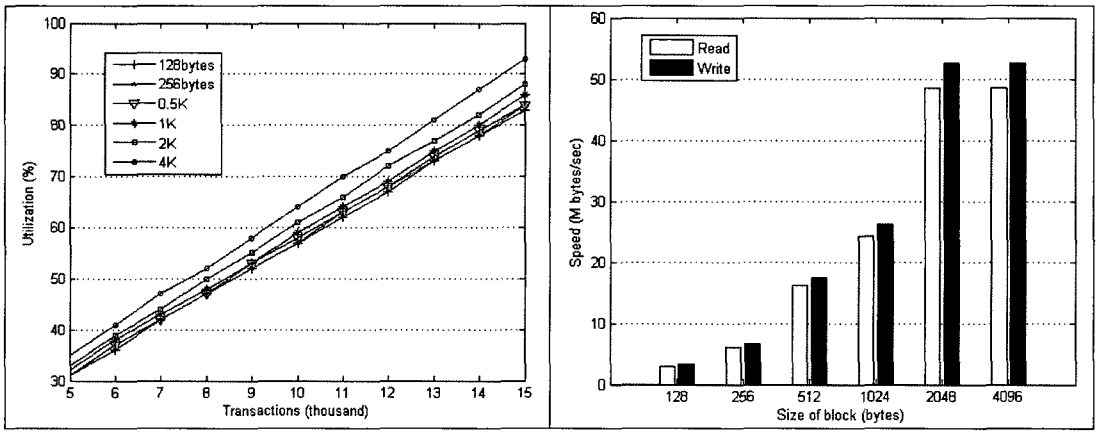


(a) 0.5K (b) 1K



(c) 2K (d) 4K

그림 6 블록 크기에 따른 파일 시스템의 수행 속도 비교



(a) 공간 효율성 (b) 속도

그림 7 NEBFS의 공간 효율성 및 속도

공간적으로는 손실이 커지지만 속도는 향상됨을 확인할 수 있다.

이러한 결과는 NEBFS만을 대상으로 한 실험에서 다시 확인할 수 있다. 그림 7은 NEBFS에서 블록 크기에

따른 공간 효율성과 실행 속도를 보여준다. 여기서 블록의 크기가 작아질수록 공간 효율성은 좋아지지만 실행 속도는 느려짐을 알 수 있다. 특히 NEBFS에서 블록의 크기가 작아지면 디렉토리를 구성하는 블록의 개수가 많아지며, 파일을 생성/삭제할 때 디렉토리 엔트리 검색 시간이 커지면서 성능이 매우 떨어짐을 확인하였다. 따라서 향후 이러한 부분에서 최적화가 필요하다.

6. 결론 및 향후 연구

급격한 기술 발전으로 NVRAM의 상품화가 빨라짐에 따라 이러한 NVRAM에 정보를 체계적으로 저장하고 관리하는 기술이 필수적이라 하겠다. 그러나 NVRAM은 기존 저장 장치에 비해 가격/용량 면에서 아직 비효율적이라는 단점이 있다. 따라서 기존 저장 장치를 위한 파일 시스템이나 메모리 관리자들을 NVRAM에 그대로 적용시킬 수 없으며 NVRAM을 효율적으로 관리하고 사용하기 위한 새로운 구조가 필요하다.

본 논문에서는 이러한 요구 조건을 만족시킬 수 있도록, 데이타를 저장할 때 낭비되는 공간을 최소화 시키는 동시에 다양한 크기의 객체를 효율적으로 저장할 수 있는 새로운 구조의 파일 시스템을 설계 하였다. 그리고 파일 생성 비용 모델에 따른 분석을 통해 제안한 파일 시스템의 공간 효율성을 분석하였으며 파일을 생성하기 위한 비용을 기존의 파일 시스템과 비교하였다.

제안한 파일 시스템의 공간 효율성은 평균적인 익스텐트의 크기에 가장 큰 영향을 받게 된다. 향후 연구는 익스텐트 크기를 효과적으로 유지시킬 수 있는 할당정책과 온라인 조각 모음 기법에 대한 연구를 수행할 것이다. 또한 블록의 크기가 작아졌을 때 속도가 떨어지는 문제점을 해결하기 위한 연구를 진행할 것이다.

참 고 문 헌

[1] H. H. Kim, Y. J. Song, and S. Y. Lee, "Novel Integration Technologies for Highly Manufacturable 32Mb FRAM," In Proceedings of the 2002 Symposium on VLSI Technologies, Digest of Technical Paper, pp. 210-211.

[2] S. J. Ahn, Y. N. Hwang, and Y. J. Song, "Highly Reliable 50nm Contact Cell Technology for 256Mb PRAM," In Proceedings of the 2005 Symposium on VLSI Technologies, Digest of Technical Paper, pp. 98-99.

[3] F. Wang, "A Modified Architecture for High-Density MRAM," ACM SIGARCH Computer Architecture News, Vol.29, pp.16-22.

[4] H. G. Lee and N. Chang, "Energy-Aware Memory Allocation in Heterogeneous Non-Volatile Memory Systems," In Proceedings of the 2003 International Symposium on Low Power Electronics and Design, pp.420-423.

[5] S. Ivanov, "Solid State Disk Architecture on the Basis of FRAM," In Proceedings of the International Conference on Computer Systems and Technologies, pp.32-36.

[6] U. Vahalia, UNIX Internals, Prentice-Hall, 1996.

[7] L. W. McVoy and S. R. Klieiman, "Extent-like Performance from a UNIX File System," In Proceedings of the Winter 1991 USENIX Conference, 1991.

[8] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, Vol. 10, No. 1, pp.22-52, 1992.

[9] M. I. Seltzer, K. Bostic, M. K. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," In Proceedings of the 1993 USENIX Winter Conference, pp.307-326, 1993.

[10] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A Fast File System for UNIX," ACM Transactions on Computer Systems, Vol. 2, No. 3, pp.181-197, 1984.

[11] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the XFS File System," In Proceedings of the USENIX 1996 Annual Technical Conference, pp.1-14, 1996.

[12] Z. Zhang and K. Ghose, "yFS: A Journaling File System Design for Handling Large Data Sets with Reduced Seeking," In Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp.59-72, 2003.

[13] EZ-X5, http://falinux.com/zproducts/ex-x5.php

[14] M. Baker and M. Sullivan, "The Recovery Box: Using Fast Recovery to Provide High Availability in the UNIX Environment," In Proceedings of the 1992 USENIX Summer Conference, pp.31-44, June 1992.

[15] P. M. Chen, W. T. Ng, G. Rajamani, and C. Aycock, "The Rio File Cache: Surviving Operating System Crashes," In Proceedings of the Architectural Support for Programming Languages and Operating Systems, pp.74-83, 1996.

[16] S. Akyurek and K. Salem, "Management of Partially Safe Buffers," IEEE Transactions on Computers, Vol. 44 No. 3 pp.394-407, 1995.

[17] T. Haining and D. D. E. Long, "Management Policies for Non-Volatile Write Caches," In Proceedings of the 1999 IEEE International Performance, Computing and Communications Conference, pp. 321-328.

[18] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer, "Non-volatile Memory for Fast, Reliable File Systems," In Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System,

Oct. 1992.
[19] A. A. Wang, P. Reiher, G. J. Popek, and G. H. Kuenning, "Conquest: Better Performance through a Disk/Persistent-RAM Hybrid File System," In Proceedings of the USENIX Annual Technical Conference, 2002.
[20] E. L. Miller, S. A. Brandt, and D. D. E. Long. "HeRMES: High-Performance Reliable MRAM-Enabled Storage," In Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII), pp.83-87, Germany, May 2001.
[21] D. Roselli, J. R. Lorch, and T. E. Anderson, "A Comparison of File System Workloads," In Proceedings of USENIX Annual 2000 Technical Conference, pp 41-54, 2000.
[22] N. K. Edel, D. Tuteja, E. L. Miller, and S. A. Brandt "MRAMFS: A Compressing File System for Non-Volatile RAM," In Proceedings of the 12th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.
[23] PRAMFS, http://pramfs.sourceforge.net
[24] A. Forin and G. R. Malan, "An MS-DOS File System for UNIX," In Proceedings of the Winter 1994 USENIX Conference, 1994.

현 철 승
2001년 제주대학교 통신컴퓨터공학부(학사). 2005년~현재 서울시립대학교 컴퓨터과학부 석사과정. 관심분야는 운영체제, 내장형시스템, 저장장치

백 승 재
2005년 단국대학교 컴퓨터공학과(학사) 2004년~현재 비트 컴퓨터 강사. 2005년 현재 단국대학교 정보컴퓨터과학과 석사과정. 관심분야는 운영체제, 내장형 시스템

최 종 무
1993년 서울대학교 해양학과(학사). 1995년 서울대학교 컴퓨터공학과(석사). 2001년 서울대학교 컴퓨터공학과(박사). 2001년~2003년 유비쿼스 주식회사 책임연구원. 2003년~현재 단국대학교 정보컴퓨터학부 조교수. 2005년~2006년 UC Santa Cruz 교환교수. 관심분야는 운영체제, 내장형시스템, 스토리지시스템

이 동 희
1989년 서울대학교 컴퓨터공학과(학사) 1991년 서울대학교 컴퓨터공학과(석사) 1998년 서울대학교 컴퓨터공학과(박사) 1998년~1999년 삼성전자 중앙연구소 선임연구원. 1999년~2001년 제주대학교 통신컴퓨터공학부 조교수. 2002년~현재 서울시립대학교 컴퓨터과학부 부교수. 관심분야는 운영체제, 플래시메모리소프트웨어, 내장형시스템

노 삼 혁
1986년 서울대학교 컴퓨터공학과(학사) 1993년 메릴랜드대학교 컴퓨터과학과(박사). 1993년~1994년 조지워싱턴대학교 객원 조교수. 1994년~현재 홍익대학교 정보컴퓨터공학부 교수. 관심분야는 운영체제, 플래시메모리소프트웨어, 내장형시스템, 차세대저장장치