비휘발성 메모리를 이용한 로그 구조 파일 시스템의 성능 향상

(Improving Log-Structured File System Performance by Utilizing Non-Volatile Memory)

강 양 욱 † 최 종 무 ††
(Yangwook Kang) (Jongmoo Choi)
이 동 희 ††† 노 삼 혁 ††††
(Donghee Lee) (Sam H. Noh)

요 약 로그 구조 파일 시스템(Log-Structured File System, LFS)은 변경된 데이타를 메모리에 충분히 모아서 한번에 순차 쓰기로 디스크에 기록함으로써 높은 쓰기 성능을 실현한 파일 시스템이다. 그러나 실제 시스템에서는 여전히 디스크와 메모리 상의 일관성을 위해서 동기화가 발생하며 변경된 데이타를 충분히 메모리에 모으지 못한 채 디스크로 쓰기가 발생하는 모습을 보인다. 자주 발생되는 쓰기는 클리너의 오버헤드를 증가시키고, 더 많은 메타 데이타를 기록하게 한다. 본 연구에서는 비휘발성 메모리를 이용해서 동기화를 없애고, 작은 단위의 쓰기를 효과적으로 활용하도록 LFS와 운영체제의 관련된 서브 시스템들을 변경하였다. 이를 통하여 DRAM만 있는 LFS에 비해서 256M의 NVRAM을 가진 시스템에서 약 2.5배의 성능 향상을 보였다.

· 이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임(No. R0A-2007-000-20071-0)
· 이 논문은 제34회 추계학술대회에서 '비휘발성 메모리를 이용한 로그 구조 파일 시스템의 성능 향상'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 홍익대학교 컴퓨터공학과
ywkang80@gmail.com
†† 종신회원 : 단국대학교 정보컴퓨터학부 교수
choijm@dankook.ac.kr
††† 정 회 원 : 서울시립대학교 컴퓨터과학부 교수
dhlee@venus.uos.ac.kr
†††† 종신회원 : 홍익대학교 컴퓨터공학과 교수
samhnoh@hongik.ac.kr

논문접수 : 2007년 12월 6일
심사완료 : 2008년 4월 5일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제5호(2008.7)

키워드 : 로그 구조 파일 시스템, 비휘발성 램, 쓰기 캐시

Abstract Log-Structured File System(LFS) is a disk based file system that is optimized for improving the write performance. LFS gathers dirty data in memory as long as possible, and flushes all dirty data sequentially at once. In a real system, however, maintaining dirty data in memory should be flushed into a disk to meet file system consistency issues even if more memory is still available. This synchronizations increase the cleaner overhead of LFS and make LFS to write down more metadata into a disk. In this paper, by adapting Non-volatile RAM(NV-RAM) we modifies LFS and virtual memory subsystem to guarantee that LFS could gather enough dirty data in the memory and reduce small disk writes. By doing so, we improves the performance of LFS by around 2.5 times than the original LFS.

Key words : Log-Structured File System, Non-Volatile RAM, Write Cache

1. 서 론

지난 십 수 년 간, 파일 시스템의 성능에 대한 연구들은 탐색 시간과 회전 지연 시간을 지닌 디스크의 물리적인 단점을 적절한 할당 정책과 휘발성 캐시를 바탕으로 해결하는 것에 초점이 맞춰져 왔다. Fast File System(FFS)[1]은 인접한 실린더에 함께 탐색될 가능성이 높은 데이타들을 저장하여 디스크 헤드의 움직임을 최소화하고자 하였으며 로그 구조 파일 시스템(Log- Structured File System, LFS)[2,3]는 큰 단위로 데이타를 모아서 세그먼트란 구조를 만들어 디스크로 쓰게 함으로써 최대한 순차 쓰기가 일어날 수 있게 설계되었다. 읽기 요청시 발생할 수 있는 탐색 오버헤드는 대부분이 메모리 캐시에 의해서 흡수된다. 이렇게 함으로써 LFS는 기존의 파일 시스템들에 비해서 쓰기 성능을 높이고 유사한 읽기 성능을 유지함으로써 더 높은 성능을 얻고자 하였다.

그러나 메타 데이타와 같은 파일 시스템의 안정성과 관련된 데이타들을 보호하기 위해서 실제 시스템의 구현에서는 동기화가 일어날 수 밖에 없다. LFS는 메타데이타들의 기록은 비동기화 되어 있지만, 메타 데이타 연산이 요청되면 가능한 빠르게 디스크로 모든 변경된 데이타를 기록한다. 또한 LFS는 덮어 쓰기가 없기 때문에 한번 디스크 기록이 일어나게 되면 순차 쓰기를 위해 만들어지는 세그먼트 정보와 같은 부가적인 메타 데이타들이 늘어나고, 또한 이전 블록들을 재활용해야 하는 클리너의 부담도 늘어나게 된다. 따라서 실제 환경에서 디스크의 사용량이 적을 때는 다른 파일 시스템에 비해서 높은 성능을 보이고 있으나 디스크의 사용량이 늘어남에 따라 점차 성능이 하락하는 경향을 보이게 된다.

LFS가 디자인대로의 최대한의 성능을 보이게 하려면 쓰여질 데이타를 충분히 메모리에 모아두어 캐시로서의 효과를 얻고, 쓰기가 일어난 경우엔 세그먼트 크기 이상의 큰 단위로 I/O가 발생하도록 하여야 한다. 그러나 파일 시스템의 안정성과 일관성의 문제 때문에 휘발성 메모리를 전제로 한 현대의 운영체제에서는 LFS가 목표했던 것처럼 동작하는데 한계가 있다. 이러한 한계를 극복하기 위한 방안으로 차세대 비휘발성 메모리(Non-Volatile Memory, NVRAM)의 활용을 고려할 수 있다.

차세대 비휘발성 메모리는 DRAM과 유사한 I/O 성능을 가지면서도 별도의 전원 없이 데이타를 영속적으로 저장할 수 있다. 기존의 배터리에 기반한 연구는 오랫동안 연구되어 왔지만[4], 최근 반도체 기술의 발전으로 배터리의 도움이 필요 없는 다양한 종류의 NVRAM들이 등장하고 있다. 용량 면에서도 주요 반도체 회사 중 하나인 삼성전자는 2006년 512Mb NVRAM을 개발하였고, 2008년부터 양산을 시작하여 매년 2 배씩 용량을 늘려갈 계획에 있다[5]. 앞으로 점차 NVRAM이 현재의 휘발성 메모리의 일부를 대체해가게 될 것으로 예상한다.

본 연구는 이러한 상황에서 NVRAM을 이용하여 LFS의 부분 세그먼트 쓰기 문제를 해결하여 I/O 횟수를 줄이고 클리너의 오버헤드를 줄임으로써 성능을 높인다. 본 연구팀의 기존 연구[6]의 설계를 개선하고 성능을 향상시켰다. TPC-C 트레이스 실험 결과 NetBSD 3.1의 기존 LFS 성능 대비 2.5배의 성능 향상 있었다.

이하 논문의 구성은 다음과 같다. 2장에서는 비휘발성 메모리를 이용한 LFS의 설계에 대해서 설명하며, 3장에서는 BSD 커널 상에 비휘발성 메모리를 활용하는 LFS의 구현, 4장에서는 실험 결과를 소개하며, 5장에서는 결론 및 향후 연구 내용을 소개한다.

2. 비휘발성 메모리를 이용한 LFS의 설계

LFS는 세그먼트 단위로 메타 데이타와 파일 데이타를 기록한다. 세그먼트는 파일의 메타 데이타, 데이타 블록과 그 세그먼트가 포함하고 있는 블록의 사용 내역, 파일들의 위치 등의 정보를 포함하는 세그먼트 메타데이타 블록(Segment Summary Block)을 포함한다. 세그먼트의 크기는 보통 512KB이나 1MB 단위인데, 메모리 상에 모인 데이타들을 이 단위로 나누어 적절한 시점에 기록하게 된다. 그러나 쓰여져야 할 데이타의 크기가 이 세그먼트 크기보다 작을 때 부분 세그먼트 쓰기가 발생한다. 이러한 부분 세그먼트 쓰기가 많을수록 더 많은 메타데이타들이 디스크로 기록되게 되며 디스크 사용률이 높아짐에 따라 클리너의 부담을 가중시키게 된다. 비휘발성 메모리를 활용하는 LFS의 설계는 이러한 부분 세그먼트 쓰기를 효과적으로 감소하는데 있다.

현대의 운영체제는 휘발성 메모리를 전제로 설계되어 있기 때문에 특성이 다른 비휘발성 메모리를 활용하기 위해서는 우선 기존의 운영체제의 가상 메모리 관리자 및 파일 시스템에 관련된 서브 시스템들의 설계 및 정책상 변화가 필요하다. 각 서브 시스템들의 정책을 결정하는데 있어 몇 가지 중요한 이슈가 있다.

- NVRAM을 버퍼 캐시의 일부로서 활용할 것인가, 페이지 캐시의 일부로 활용할 것인가.
- NVRAM의 할당, 해제, 관리는 언제 어떻게 발생하는가.
- NVRAM에 포함되어야 할 데이타는 무엇인가.
- NVRAM의 데이타는 언제 디스크로 쓰여져야 하는가.

이 장에서는 LFS의 관점에서 이러한 디자인 이슈들을 바탕으로 변경되어야 할 운영체제 서브 시스템들의 변화를 설명한다.

2.1 비휘발성 메모리의 활용

대부분의 배터리에 기반한 비휘발성 메모리 관련 기존 연구들에서 비휘발성 메모리는 그림 1(a)와 같이 추가적인 쓰기 캐시로서 활용되었다. FFS와 같은 파일 시스템에서 이 방법은 FFS를 비롯한 많은 파일 시스템들에서 이러한 방식을 통해서 메타데이타의 동기화 연산으로 인한 작은 쓰기 오버헤드를 효과적으로 줄이는데 도움이 되었다. 그러나 LFS에서는 메타 데이타 또한 일반 데이타와 같이 한번에 모아서 기록되므로, 상대적으로 매우 작은 단위의 쓰기가 적기 때문에 다른 파일 시스템에 비해서 LFS에 비휘발성 메모리 디스크 캐시의 도입은 이득이 적다.

그림 2(b)와 같이 페이지 캐시의 일부로서 NVRAM을 활용하게 되면 이러한 문제를 해결할 수 있다. 변경된 모든 데이타들이 NVRAM 상에 존재하게 되기 때문에 메모리와의 일관성 문제를 발생하지 않아 동기화를 제거할 수 있다. 이 때문에 메모리에서 충분히 많은 양의 데이타를 모아서 기록하게 됨으로써 부분 세그먼트 쓰기 문제를 완화한다.

2.2 비휘발성 메모리 페이지의 할당과 해제

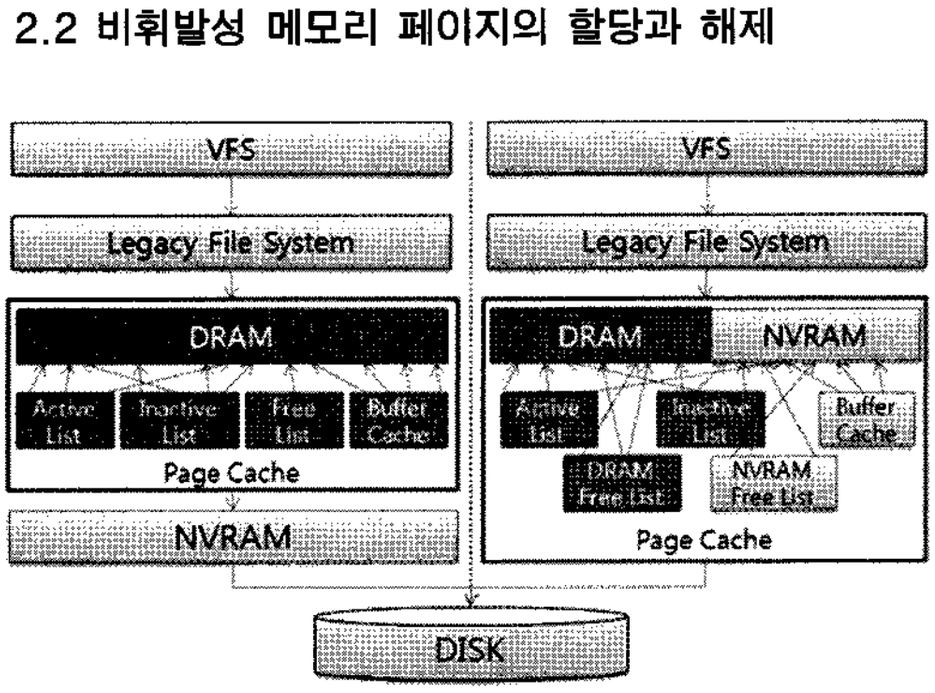

그림 1 비휘발성 메모리의 레이아웃

LFS에서 NVRAM이 필요한 시점이 되면 가상 메모리 관리자에 NVRAM 페이지 할당 요청을 보낸다. 이때 빈 NVRAM 페이지가 남아있지 않다면 페이지 데몬은 사용중인 페이지를 해제하여 재사용할 수 있게 한다. 기존의 DRAM 외에도 NVRAM 페이지들을 효과적으로 관리하기 위해서 페이지 데몬은 현재 메모리 내에 빈 NVRAM 페이지들을 일정 수준으로 유지한다. 빈 페이지를 유지하는 비율에 따라서 이를 활용하는 파일 시스템의 성능 또한 변화하지만, 본 연구에서는 기존의 DRAM 페이지 관리 정책과 동일한 비율로 NVRAM 빈 페이지를 유지하도록 관리한다.

NVRAM 빈 페이지들을 효과적으로 관리하기 위해서 그림 2와 같이 새롭게 가상 메모리 관리자 내에 NVRAM 페이지 리스트를 추가하였다. 현재 널리 사용되고 있는 리눅스나 BSD와 같은 범용적인 운영체제가 분류하는 Active, InActive, FreeList외에도 새롭게 NVRAM FreeList가 추가된다. NVRAM 페이지가 가득 차면 LFS는 변경된 데이타들을 기록하여 페이지 데몬에 의해 의한 쓰기가 빈번히 발생하지 않도록 한다.

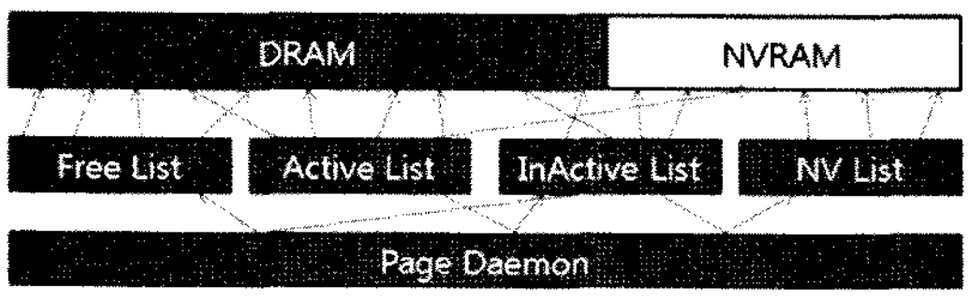

그림 2 비휘발성 메모리 페이지 관리

2.3 비휘발성 메모리에 포함되는 데이타

LFS는 기록시에 파일의 메타 데이타와 데이타가 같은 위치에 기록되므로 물리적 위치가 다른 곳에 기록되는 파일 시스템들에 비해서 메타 데이타만 NVRAM에 유지하는 것은 디스크의 탐색 시간에 영향을 주지 못한다. 따라서 메타 데이타뿐 아니라 실제 데이타 또한 NVRAM에 기록하여 한번에 NVRAM의 모든 데이타가 같이 쓰여질 수 있게 한다.

읽기와 쓰기 요청에 따른 NVRAM 할당 정책이 달라질 수 있다. 쓰기 요청된 LFS 블록들만 NVRAM 페이지에 할당하고 읽기 요청된 블록들은 DRAM 페이지에 구분하는 방법과 모든 읽기와 쓰기 구분 없이 요청된 LFS 블록들을 NVRAM 페이지에 저장하는 방법이다. 전자의 방법은 쓰기 요청을 NVRAM 페이지가 가능한 만큼 미루어 디스크로 쓸 수 있기 때문에 디스크 I/O를 최대한 줄일 수 있다는 장점이 있다. 그러나 이 방법은 읽기 요청에 의해 캐시 상에 올라온 블록을 변경하고자 할 때 이 블록들이 다시 NVRAM으로 복사되어야 하기 때문에 페이지 복사 오버헤드가 추가로 발생한다. 또한 읽기 요청이 다수인 경우 빈 NVRAM 페이지들이 있음에도 DRAM 페이지 부족으로 인해 페이지 데몬이 빈번히 호출됨으로써 이에 따른 오버헤드가 발생하게 된다. 후자의 방법은 읽기, 쓰기 요청을 모두 NVRAM에서 활용하기 때문에 페이지 복사 오버헤드가 발생하지는 않는다. 그러나 쓰여지는 시점에 읽기 요청이 쓰기 요청에 비해 NVRAM 공간을 많이 차지하여 작은 크기의 쓰기가 나타날 수 있다.

본 연구에서는 NVRAM에 LFS의 읽기 쓰기 요청을 모두 포함하는 방법을 선택하였다. NVRAM 페이지 복사 오버헤드를 줄일 수 있고, 다른 서브 시스템들과 함께 사용되는 DRAM 량의 의존성을 낮출 수 있어 NVRAM의 효과를 살펴보기에 효과적이다.

2.4 파일 시스템 일관성 유지

파일 시스템이 변경한 페이지들이 모두 NVRAM에 보관되기 때문에 더 이상 일관성 유지를 위해서 디스크로 기록하지 않는다. NVRAM에 쓰는 순간 이미 디스크에 쓰여진 것과 같기 때문이다. 따라서 FSYNC와 같은 응용 프로그램의 요청을 무시할 수 있으며, 메타데이타 연산으로 인한 잦은 플러시를 줄 일 수 있다. 복구 과정 또한 디스크를 스캔하는 과정 없이도 메모리 상의 데이타를 다시 재배치 하는 것으로 완료할 수 있다.

NVRAM에 저장된 파일 데이타와 메타데이타들을 부팅 과정에서 시스템이 인식할 수 있도록 하기 위해 각 NVRAM 페이지들이 자신의 Inode와 파일 오프셋에 대한 정보를 가지도록 NVRAM 페이지 헤더에 복구를 위한 정보들을 추가하였다.

시스템이 부팅되면 메모리 상의 NVRAM 페이지들 중 변경된 페이지들의 Inode와 오프셋을 이용해 각 Vnode의 변경된 페이지 리스트에 추가할 수 있도록 하고, LFS를 한번 플러시 해줌으로써 변경된 모든 데이타가 꺼지기 전에 플러시 되었던 것과 같은 효과를 줄 수 있다.

3. 비휘발성 메모리를 활용하는 LFS의 구현

본 연구에서는 NetBSD 3.1 커널 상에 NVRAM을 활용하는 LFS를 구현하였다. NetBSD는 LFS가 기본적으로 커널과 함께 배포되는 유일한 운영체제이다. NetBSD에서 물리 메모리는 BSD의 가상 메모리 관리자인 UVM[7]에 의해서 관리되며, UVM과 디스크간의 데이타 교환은 UBC(Unified Buffer Cache)[8]에 의해서 이루어진다. 구현은 크게 4부분으로 나누어 질 수 있다.

- NVRAM 영역의 할당 - UVM이 물리 메모리의 일부를 NVRAM으로 가정하고 별도로 관리한다.
- NVRAM 페이지 관리 - NVRAM 페이지의 할당, 해제와 재사용이 가능하다.

- 동기화 메커니즘의 제거
- LFS에서의 NVRAM 활용 - LFS는 쓰기 요청을 NVRAM 페이지를 할당 받아서 처리한다. NVRAM 공간이 가득 차는 경우를 체크하여 디스크로 내보낸다.

3.1 NVRAM 영역의 할당

실제 NVRAM 하드웨어가 페이지 캐시에 사용될 만큼 큰 용량으로 상용화되지 않았기 때문에 DRAM의 일부 공간을 NVRAM 영역으로 설정하였다. 그림 3은 DRAM 공간의 일부를 NVRAM 공간으로 할당하는 방법을 나타낸 그림이다. 시스템 부팅 시에 DRAM 공간의 일부를 떼어내고, 이 영역을 페이지 크기로 나누어 NVRAM 페이지 리스트로 저장하였다. NVRAM 영역의 앞 부분은 헤더를 위해서 사용하는데, Super Block과 페이지 헤더 구조체들의 리스트로 나뉘어진다. Superblock은 NVRAM Page List를 가지고 있으며, 페이지 리스트의 상태 정보 등을 담고 있다. 페이지 헤더 구조체는 NetBSD의 Page 구조체에 데이타의 복구를 위해서 디바이스 번호와 아이노드 번호, 파일의 오프셋, 데이타의 크기를 추가하여 관리하고 있다.

3.2 NVRAM 페이지의 관리

페이지의 할당은 UVM내 uvm_pagealloc 함수에 의해서 이루어진다. read나 write 시스템 콜을 호출하면 LFS는 해당 페이지가 메모리 상에 있는지 확인하고 필요한 경우 uvm_pagealloc 함수를 이용해서 페이지 할당을 요청한다.

파일 시스템에 필요한 NVRAM 페이지를 할당하기 위해서 uvm_pagealloc 함수에 NVRAM_PAGE_REQ 플래그를 추가하고. 이 함수가 빈 NVRAM 페이지 리스트로부터 NVRAM 페이지를 할당할 수 있도록 수정하였다.

쓰기 요청이 들어오면 우선 빈 NVRAM 페이지수를 체크하고 충분한 빈 공간이 있다면 NVRAM 페이지들을 할당한다. 요청된 쓰기에 필요한 페이지 수보다 빈 페이지 수가 작다면, LFS의 플러시 루틴를 통해서 모든 NVRAM 상의 변경된 데이타를 디스크로 기록하고, 빈 NVRAM 페이지들을 확보한다. LFS는 다른 파일 시스템과는 달리 VM에서 제공하는 버퍼 캐시 알고리즘이나 디스크로 쓰기 위해서 vnode pager를 이용하지 않기 때문에 이러한 변경은 VM이 아닌 LFS 파일 시스템의 Write 함수 내에서 이루어졌다.

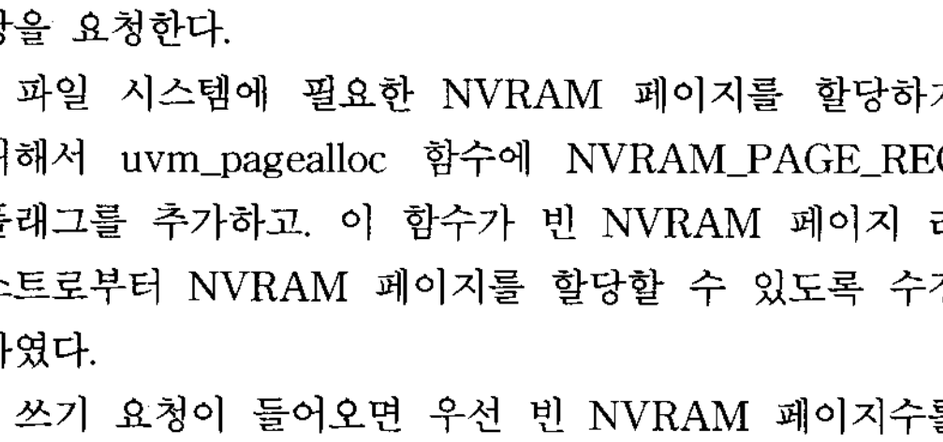

그림 3 비휘발성 메모리 영역(NVRAM)의 할당

빈 NVRAM 페이지 수가 어느 기준 이하로 내려가면 NVRAM 페이지 데몬이 실행된다. NVRAM 페이지 데몬은 더 이상 사용되지 않거나 변경되지 않았거나 낮은 우선 순위의 NVRAM 페이지들을 회수하여 빈 페이지를 확보한다. NVRAM 페이지 데몬이 실행되는 기준은 NetBSD에서와 같이 4%~6% 이하로 빈 NVRAM 페이지가 줄어든 경우로 하였다.

3.3 동기화 메커니즘의 제거

NetBSD LFS에서 동기화는 syncer 모듈에 의한 30초 마다 일반 데이타를 동기화하도록 되어있으며, 메타데이타의 경우 이보다 더 짧은 주기로 하드 디스크로 동기화하도록 하고 있다. NVRAM을 활용하는 LFS에서는 동기화의 위험이 없게 되므로, 모든 종류의 데이타에 대해서 동기화를 동작시키지 않는다. LFS에서 디스크로 쓰는 경우가 발생하는 것은 NVRAM 공간이 부족한 경우만 있게 된다.

4. 성능 평가

동기화를 제거한 효과를 알아보기 위해서 TPC-C 트레이스를 수행하여 응답 시간 및 NVRAM 크기에 따른 성능 변화를 측정하여 보았다. 파일 시스템 성능에 미치는 다른 요소들을 최소화하기 위해서 원격의 수집 서버를 구축하여 커널 내외의 정보를 전송하도록 하였다.

실험에 사용된 TPC-C 트레이스는 1200만개의 동기화된 읽기와 쓰기 명령으로 이루어져 있으며, TPC-C의 각 명령 당 응답 시간을 수집하였다. TPC-C를 수행하면서 커널 내 LFS의 각종 내부 상태를 관찰할 수 있도록 LFS 내의 플러시 횟수, 새로 생성된 세그먼트 수 등의 파라미터들 역시 수집 서버로 전송하였다. 실험은 768M의 물리 메모리를 갖는 펜티엄4 급 PC에서 수행하였으며, 수집 서버 역시 같은 사양의 PC와 MSSQL 데이타베이스를 사용하였다.

TPC-C 트레이스의 실행 결과 표 1에 나타난 것처럼 DRAM 768M를 사용하는 NetBSD LFS의 경우 총 31.567 시간이 소요되었으며, 이 중 32M를 NVRAM으로 활용한 경우 18.181 시간이 소요되었다. 256M까지 NVRAM을 활용한 경우엔 12.274 시간으로 2.5 배 이상 수행 시간이 빨라졌다.

이러한 수행 성능의 향상은 NVRAM으로 인해 부분 세그먼트 쓰기가 줄면서 플러시의 발생 횟수가 감소한 데 그 영향이 있다. 표 1의 세그먼트 플러시 발생 횟수는 TPC-C 수행 시간과 같은 모습을 띄며 감소되었다. 디스크로 기록된 실제 블록수의 변화 또한 비슷한 비율로

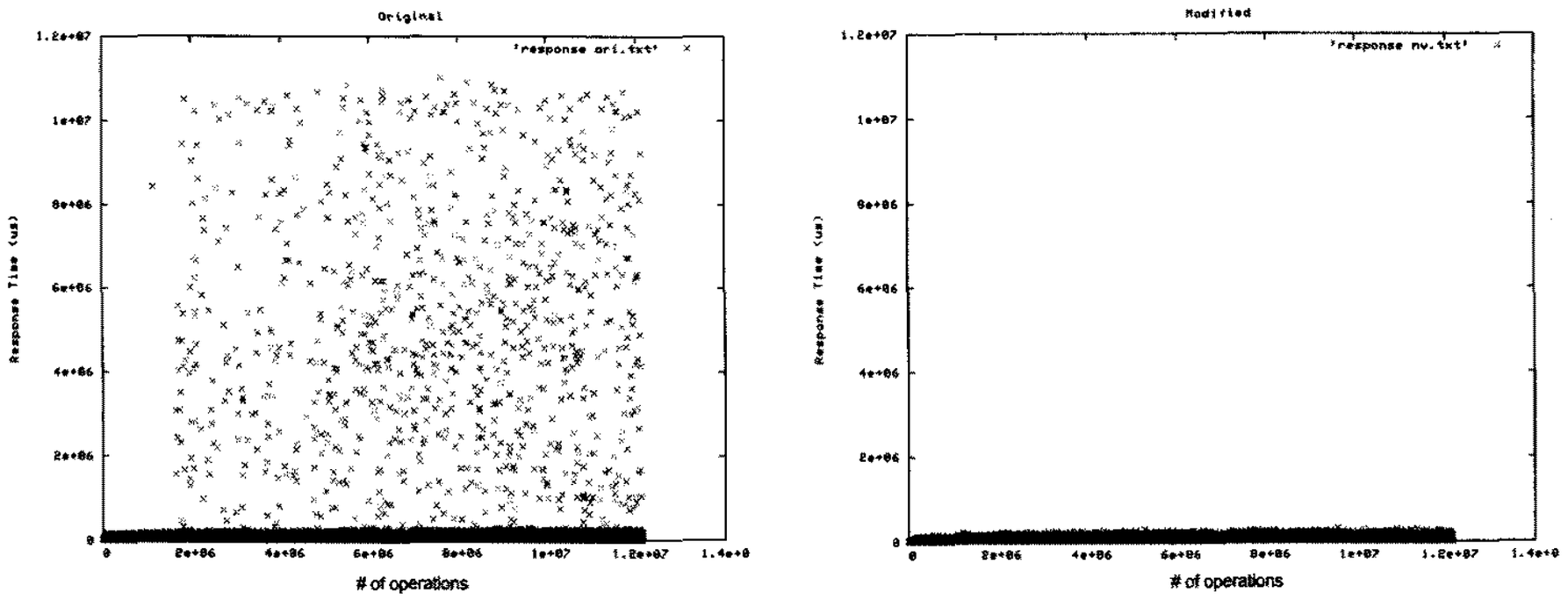

그림 4 (a) NetBSD LFS의 응답 시간 분포 (b) 수정된 NetBSD LFS의 응답 시간 분포

표 1 TPC-C 트레이스 실험 결과

	Original	NVRAM 32M	NVRAM 256M
플러시 횟수	1869696	599393	393729
쓰기 연산 횟수	3785135	539117	399493
응답시간 평균	8195 us	5374 us	3624 us
응답시간 표준편차	52509 us	16309 us	12473 us

감소하고 있다. 이렇게 실제로 쓰여진 블록수가 줄어든 것은 클리너에 의해 처리되어야 할 세그먼트 수가 줄어들게 되는 효과가 있다. 클리너에 의해 쓰여진 블락 수는 32M NVRAM을 사용했을 때 31756개에서 256M일 때 28401개로 감소한 결과를 보여주고 있다. 실제 측정된 응답 시간의 분포 그림 4의 경우와 같이 NVRAM을 사용한 경우가 고른 결과를 보여준다. 각 NVRAM크기에 따른 평균 응답 시간은 표 1에 나타난 것과 같이 각 요청당 평균 4571 us (66%) 만큼 감소하였다.

5. 결론 및 향후 연구

본 연구는 LFS가 휘발성 메모리를 사용함에 따라서 피할 수 없었던 성능 상의 이슈를 해결하기 위해 차세대 NVRAM을 활용하는 LFS를 설계하고 구현하였다.

이를 통하여 부분 세그먼트 쓰기를 감소시키고, LFS가 생성하는 자료구조의 수를 감소시킴으로써 클리너의 오버헤드를 감소하고 TPC-C 트레이스 실험 결과 수행 성능을 2.5배 향상시킬 수 있었다. 그러나 이 실험에서 최대한 간략하게 결정된 디스테이징 정책, 페이지 재사용 정책 등 여러 정책들의 개선을 통하여 NVRAM의 잠재적인 성능을 더 높일 수 있을 것으로 생각한다. 적절한 디스테이징 정책은 적은 양의 NVRAM으로 더 큰 용량의 NVRAM 만큼의 효과를 낼 수 있을 것으로 기대한다. 이러한 연구 과제들을 중심으로 한정된 메모리 공간 안에서 디스테이징 시점을 바꾸어가며 효과적인 디스테이징 정책을 찾는 연구를 진행하고 있다.

참 고 문 헌

[1] Marshall Kirk McKusick, Marshall K. Mckusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry, "A Fast File System for UNIX," Computer Systems, Vol.2, No.3, pp. 181-197, 1984.

[2] Margo Seltzer, Keith Bostic, Marshall Kirk McKusick, Carl Staelin, "An Implementation of a Log-Structured File System for UNIX," In the Proceedings of the Winter 1993 USENIX Conference, pp. 307-326, 1993.

[3] Mendel Rosenblum and John K. Ousterhout, The Design and Implementation of a Log-Structured File System, ACM Transactions on Computer Systems, Vol.10, No.1, pp. 26-52, 1992.

[4] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, Margo Seltzer, "Non-volatile memory for fast, reliable file systems," In the Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 10-22, October 1992.

[5] http://www.etnews.co.kr/newswire/press_view.html?id=0184587, April, 2007.

[6] Yangwook Kang, Jongmoo Choi, Donghee Lee, Sam H. Noh, "Design and Implementation of the Log-Structured File System Utilizing Nonvolatile Memory," In the Proceedings of the Korean Computer Conference (B) Vol.34, No.1, pp. 310-314, June 2007.

[7] Charles D. Cranor and Gurudatta M. Parulkar, "The UVM Virtual Memory System," In the Proceedings of the USENIX Annual Technical Conference, pp. 117-130, 1999.

[8] Chuck Silvers, "UBC: An Efficient Unified I/O and Memory Caching Subsystem for NetBSD," In the Proceedings of the Freenix 2000 USENIX Annual Technical Conference, pp. 285-290, 2000.