ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 19-01-12
80

j.inst.Korean.electr.electron.eng.Vol.23,No.1,80~90,March 2019
http://dx.doi.org/10.7471/ikeee.2019.23.1.80

비휘발성 메모리 저장장치를 위한 영속적 페이지 테이블 및 파일시스템 저널링 기법

Persistent Page Table and File System Journaling Scheme for NVM Storage

안 재 형*, 현 철 승, 이 동 희*★

Jae-hyeong Ahn*, Choul-seung Hyun*, Dong-hee Lee*★

Abstract

Even though Non-Volatile Memory (NVM) is used for data storage, a page table should be built to access data in it. And this observation leads us to the Persistent Page Table (PPT) scheme that keeps the page table in NVM persistently. By the way, processors have different page table structures and really operational page table cannot be built without virtual and physical addresses of NVM. However, those addresses are determined dynamically when NVM storage is attached to the system. Thus, the PPT should have system-independent and also address-independent structure and really working system-dependent page table should be built from the PPT. Moreover, entries of PPT should be updated atomically and, in this paper, we describe the design of PPT that meets those requirements. And we investigate how file systems can decrease the journaling overhead with the swap operation, which is a new operation created by the PPT. We modified the Ext4 file system in Linux and experiments conducted with Filebench workloads show that the swap operation enhances file system performance up to 60%.

요 약

최근에 소개된 비휘발성 메모리(Non-Volatile Memory)를 저장장치로 사용하는 경우에도 데이터를 접근하기 위해서는 페이지 테이블이 구축되어야 한다. 이 점에 착안하여 본 논문에서는 페이지 테이블 자체를 비휘발성 메모리에 유지하는 영속적 페이지 테이블 (Persistent Page Table) 기법을 설계한다. 실제 페이지 테이블의 구조는 프로세서마다 다르다. 또한 비휘발성 메모리의 물리주소와 가상주소는 종종 저장장치가 시스템에 연결되기 전까지 알 수 없기 때문에 연결 시점까지는 실제로 동작하는 페이지 테이블을 만들 수 없다. 따라서 영속적 페이지 테이블은 주소와 시스템으로부터 독립적인 구조를 가져야 하며, 저장장치가 동작하는 시점에 영속적 페이지 테이블을 기반으로 시스템 종속적인 페이지 테이블이 생성되어야 한다. 또한 영속적 페이지 테이블 엔트리는 원자적으로 변경되어야 하며, 본 논문에서는 이러한 영속적 페이지 테이블의 설계에 대해 설명한다. 다음으로 파일시스템이 영속적 페이지 테이블이 제공하는 교환 연산을 활용하여 저널링 오버헤드를 감소시킬 수 있음을 보인다. 교환 연산을 활용하도록 Linux Ext4 파일시스템을 변경하였으며, Filebench 워크로드를 이용한 성능 측정 결과를 보면 영속적 페이지 테이블과 교환 연산은 파일시스템의 성능을 최대 60% 향상시킨다.

Key words : Non-Volatile Memory, Ext4 File System, Journaling Overhead, Permanent Page Table, Swap Operation

* Dept. of Computer Science and Engineering, University of Seoul
★ Corresponding author
E-mail : dhl_express@uos.ac.kr, Tel : +82-2-6490-2449
※ This work was supported by the 2017 Research Fund of the University of Seoul.
Manuscript received Mar. 6, 2019; revised Mar. 16, 2019; accepted Mar. 18, 2019.

Ⅰ. 서론

최근에 널리 연구되는 vPCM[1][2], STT-RAM [3], Re-RAM[4][5], 3D-XPoint[6]와 같은 비휘발성 메모리는 비휘발성 속성과 메모리 속성을 모두 가지고 있다. 이들은 바이트 단위 접근이 가능하여 메모리처럼 사용되기도 하고 비휘발성 속성 때문에 저장장치 형태로 사용되기도 한다. 그런데 비휘발성 메모리를 저장장치로 사용하는 경우에도 데이터를 접근하기 위해서는 페이지 테이블이 구축되어야 한다. 일반적으로 운영체제는 시스템 부팅 시점에 또는 비휘발성 메모리 저장장치가 연결되는 시점에 이를 위한 페이지 테이블을 구축한다. 그리고 이 페이지 테이블은 시스템 종료 시점에 또는 저장장치가 분리되는 시점에 사라진다.

본 논문에서는 비휘발성 메모리 저장장치를 위한 페이지 테이블 자체를 비휘발성 메모리 자체에 유지하는 영속적 페이지 테이블(Persistent Page Table) 기법을 제안한다. 영속적 페이지 테이블은 저장장치가 꺼진 후에도 유지되며, 저장장치가 다시 동작하면 사용된다. 그런데 페이지 테이블 구조는 시스템이 사용하는 프로세서에 의존적이다. 따라서 영속적 페이지 테이블이 특정 프로세서에 맞게 작성되면, 저장장치를 프로세서가 다른 시스템에는 연결할 수 없다. 또한 비휘발성 메모리의 물리주소와 가상주소가 결정되어야 페이지 테이블을 만들 수 있는데, 저장장치가 시스템에 연결되기 전까지 이러한 주소는 알기 어렵다. 따라서 영속적 페이지 테이블은 시스템에 독립적이면서 주소에 독립적인 구조를 가져야 한다. 본 논문에서는 시스템과 주소에 독립적인 영속적 페이지 테이블의 설계를 제시한다. 또한 영속적 페이지 테이블의 엔트리는 원자적으로 변경되어야 하는데, 본 논문은 엔트리의 원자적 변경 기법에 대해 설명한다.

영속적 페이지 테이블은 기존 저장장치에서는 불가능한 교환 연산을 가능하게 한다. 교환 연산은 페이지 테이블의 두 엔트리를 바꾸어 이들이 가리키는 두 블록의 내용을 바꾸는 연산이다. 이러한 교환 연산은 파일시스템의 저널링 오버헤드를 줄이는데 사용될 수 있다. 저널링 기법은 크래시로부터 데이터를 보호하고 일관성을 빠르게 회복하기 위하여 동일한 데이터를 두 번 기록한다[7]. 그런데 교환 연산은 단 한 번의 기록과 교환 연산으로 저널링을 가능하게 한다. 본 논문은 리눅스 Ext4 파일시스템이 교환 연산을 활용하기 위해 어떻게 변경되어야 하는지를 설명한다. 특히 파일시스템의 오류 회복 절차에 대한 정교한 분석이 필요하며, 본 논문은 이러한 분석과 오류 회복 기법의 설계를 제시한다. 교환 연산을 이용하도록 Ext4 파일시스템을 변경하였으며 성능 향상 정도를 측정하였다. 측정 결과에 의하면 교환 연산 활용은 파일시스템 성능을 2~60% 정도 향상시킨다.

본 논문의 구성은 다음과 같다. 2절은 관련 연구를 제시하며, 3절은 영속적 페이지 테이블의 설계 및 원자적 변경을 설명한다. 4절은 교환 연산을 이용하기 위해 리눅스 Ext4 파일시스템을 어떻게 변경해야 하는지 보여주며, 5절은 벤치마크 프로그램을 이용하여 측정한 성능 향상 정도를 제시한다. 마지막으로 결론은 6절에서 제시된다.

Ⅱ. 관련연구

차세대 메모리 겸 스토리지로 주목받고 있는 비휘발성 메모리(NVM)는 메모리와 저장장치의 속성을 모두 가지고 있다. NVM은 바이트 단위로 접근이 가능하고 프로세서-메모리 버스에 장착되어 프로세서는 Load/Store와 같은 메모리 접근 명령으로 NVM의 데이터에 접근한다. 이러한 속성 때문에 NVM을 시스템 메모리 확장에 사용하려는 연구가 수행되었다[8]-[12]. NVM은 또한 비휘발성 속성을 가지고 있어 저장장치로 사용이 가능하며 이를 위한 연구가 수행되었다[13]-[17].

파일시스템의 데이터와 메타데이터 일관성을 지키기 위한 노력으로 저널링 기법이 개발되었다[7]. 그런데 저널링 기법은 파일시스템 일관성을 유지하기 위해 데이터를 두 번 기록한다. 당연히 이러한 두 번 쓰기는 파일시스템 성능을 저하시키는 요인이며, 저널링 오버헤드를 감소시키고 파일시스템 성능을 향상시키기 위한 많은 연구가 있었다[17]-[22]. 특히 오버헤드를 줄이기 위해 메타데이터 저널링을 블록 단위가 아닌 바이트 단위로 수행하는 연구가 수행되었다[23]. 파일을 특정 가상주소로 사상하

는 메모리 맵 파일 기능을 사용하려면 파일을 위한 페이지 테이블을 만들어야 한다. 자주 사상하는 파일의 페이지 테이블을 NVM에 저장하여 나중에 다시 사용하는 연구가 존재한다[24]. 그러나 이 연구는 NVM 저장장치를 위한 영속적 페이지 테이블을 만드는 것이 아니며, 또한 Ext4 파일시스템이 교환 연산을 이용하도록 변경하는 것과는 관련이 없다.

III. 영속적 페이지 테이블 설계

1. 시스템 독립적인 영속적 페이지 테이블

프로세서는 페이지 테이블을 이용하여 가상주소를 물리주소로 바꾸고, 물리주소를 이용하여 메모리에 접근한다. 운영체제는 비휘발성 메모리 저장장치가 연결될 때 또는 부팅할 때 비휘발성 메모리를 위한 페이지 테이블을 주기억장치에 구축한다. 특히 리눅스 운영체제의 경우 비휘발성 메모리 저장장치를 위한 디바이스 드라이버가 페이지 테이블 생성을 담당한다.

본 논문의 목적은 비휘발성 메모리 저장장치를 위한 영속적 페이지 테이블을 설계하고 영속적 페이지 테이블을 비휘발성 메모리 자체에 유지하는 것이다. 그런데 페이지 테이블 구조는 프로세서에 종속적이다. 따라서 페이지 테이블은 다른 종류의 프로세서를 가진 시스템에서는 인식되지 못한다. 결국 다양한 시스템에 장착되는 저장장치의 특성을 고려하면 영속적 페이지 테이블은 특정 프로세서에 종속되지 않는 구조를 가져야 한다.

영속적 페이지 테이블을 설계할 때 고려해야 할 사항이 하나 더 있다. 페이지 테이블을 구성하기 위해서는 비휘발성 메모리의 물리주소와 가상주소가 결정되어 있어야 한다. 그런데, 저장장치가 시스템에 고정되어 미리 결정되어 있는 경우가 아니라면, 이러한 주소는 저장장치가 시스템에 연결될 때 운영체제에 의해 결정된다. 결국 저장장치가 시스템에 연결될 때까지 프로세서가 바로 사용할 수 있는 페이지 테이블 생성은 불가능하다.

이러한 문제들을 해결하기 위해 본 논문에서는 시스템 독립적인 영속적 페이지 테이블을 설계하였다. 이 테이블은 또한 주소로부터도 독립적이어서, 비휘발성 메모리가 시작하는 물리주소와 가상주소를 모두 0번지로 가정하여 작성된다. 앞으로 시스템 독립적이면서 주소 독립적인 영속적 페이지 테이블을 L2P 맵이라 부른다. 이 L2P 맵은 비휘발성 메모리 자체에 저장되며, 영속적으로 유지된다. 저장장치가 연결되어 비휘발성 메모리 영역의 주소가 결정되면 운영체제는 이 L2P 맵을 이용하여 시스템 종속적인 페이지 테이블을 메모리에 다시 구축한다.

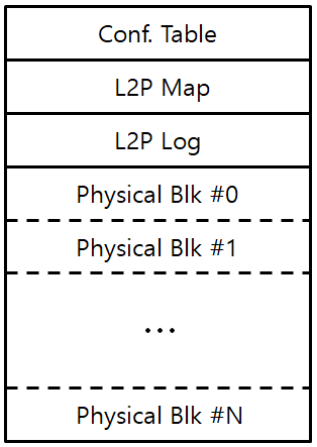

Fig. 1. NVM Layout for Persistent Page Table.
그림 1. 영속적 페이지 테이블을 위한 NVM 레이아웃

L2P 맵을 설명하기 전에 먼저 그림 1을 보면서 비휘발성 메모리 공간의 전체 구성을 설명한다. 영속적 페이지 테이블 기법은 비휘발성 메모리 공간을 네 부분으로 나누는데, 제일 앞부터 구성 테이블(Configuration Table), L2P 맵, L2P 로그 영역, 그리고 데이터 영역이 존재한다. 다시 데이터 영역에는 4KB 크기의 물리 불럭들이 존재한다. 각 블록은 물리 블록 번호와 논리 블록 번호를 가지는데 번호는 모두 0부터 시작한다. 물리 블록 번호는 고정되어 있지만 논리 블록 번호는 변할 수 있다. 4KB 크기로 분할한 이유는 펜티움 프로세서의 페이지 크기가 4KB이기 때문이며, 각 블록은 결국 페이지 단위가 되어 가상메모리 시스템에서 페이지 형태로 사용된다. 앞으로 블록과 페이지를 근본적으로 동일한 단위로 취급한다.

비휘발성 메모리의 제일 앞부분에 존재하는 구성 테이블은 뒤따라오는 세 영역의 시작 위치와 크기를 가진다. L2P 맵은 논리 블록 번호를 물리 블록 번호로 변환하는 역할을 수행하며, 영속적 페이지

테이블의 핵심이라고 할 수 있다. L2P 맵에는 물리 블록 번호만 저장되고 논리 블록 번호는 별도로 저장되어 있지 않으며, 맵 엔트리를 접근할 때 사용하는 인덱스가 논리 블록 번호를 의미한다. L2P 로그 영역은 L2P 맵을 원자적으로 변경하기 위해 사용하며 다음 절에서 자세히 설명한다.

2. 시스템 종속적 페이지 테이블

비휘발성 메모리 저장장치가 시스템에 연결되면 운영체제의 디바이스 드라이버는 구성 테이블에서 L2P 맵을 찾는다. 그리고 L2P 맵을 기반으로 시스템 종속적인 페이지 테이블을 생성한다. 물론 페이지 테이블을 생성하기 전에 비휘발성 메모리의 물리주소와 가상주소가 결정되어 있어야 하며, 이들의 시작 주소를 각각 PADDR과 VADDR이라고 하자. 이 주소가 주어지면, 디바이스 드라이버는 시스템 종속적인 페이지 테이블의 생성을 시작한다. 먼저 데이터 영역의 모든 블록에 대해 논리 블록 번호와 물리 블록 번호를 가지고 가상주소와 물리주소를 계산한다. 데이터 블록의 논리 블록 번호를 LBN, 물리 블록 번호를 PBN이라 부르겠다. 특히 PBN은 L2P[LBN]와 같이 배열 형태를 가진 L2P 맵으로부터 얻어질 수 있다. 이들을 이용하여 논리 블록 0번부터 시작하여 모든 블록의 가상주소와 물리주소를 계산하는데, 논리 블록 번호 LBN을 가진 블록의 가상주소와 물리주소 계산식은 다음과 같다.

LBN 블록의 가상주소 = VADDR + LBN*4KB
LBN 블록의 물리주소 = PADDR + PBN*4KB

각 블록의 가상주소와 물리주소가 결정되면 가상주소가 물리주소로 사상되도록 시스템 종속적인 페이지 테이블 엔트리에 등록하면 된다. 등록을 마치면 가상 메모리 하드웨어에 의해 가상주소가 물리주소로 변환되어 가상주소로 이에 대응되는 블록에 접근할 수 있다. 그런데 현대 프로세서와 운영체제는 다단계 페이지 테이블을 사용하고 있다. 구체적으로 리눅스 운영체제는 가상 메모리 주소 하나를 물리주소로 변환하기 위해 PGD, PMD, PTE로 이어지는 여러 단계의 페이지 테이블을 검색한다. 따라서 필요한 경우 최하위 페이지 테이블 이외에도 중간 단계의 페이지 테이블을 생성해야 한다. 리눅스 커널에 있는 페이지 등록 함수는 필요한 경우 중간 단계의 페이지 테이블도 생성한다.

가상주소를 물리주소로 변환하기 위해 여러 단계의 페이지 테이블을 검색하는 것은 메모리 참조 성능을 떨어뜨린다. 현대 프로세서는 이러한 다단계 페이지 테이블 검색으로 인한 성능 감소를 최소화하기 위하여 내부에 TLB(Translation Lookaside Buffer)를 가지고 있다. TLB에는 최근에 변환된 가상주소와 물리주소 쌍이 저장되며, 페이지 테이블을 검색하지 않고 가상주소를 물리주소로 바로 변환할 수 있게 한다. 이처럼 TLB는 성능 향상에 많은 도움을 주지만, 페이지 테이블 엔트리가 변경되면 TLB 플러시 명령을 사용하여 TLB의 내용을 모두 무효화해야 한다. 그리고 주소가 변환될 때마다 해당 내용이 다시 TLB에 기록되어 나중에 동일한 주소가 TLB를 통해 바로 변환되도록 한다.

다음 절에서 설명하는 교환 연산은 페이지 테이블의 두 엔트리를 교환하는 연산이다. 그런데 교환 연산은 페이지 테이블을 변경하므로 TLB 플러시 명령으로 TLB를 무효화 하며, 이로 인한 성능 하락이 존재한다. 그러나 성능 측정 결과에 따르면 TLB 플러시로 인한 성능 하락은 무시할 정도로 작다. 그러나 교환 연산으로 인한 성능 이득은 이보다 훨씬 크다.

실험에 사용한 프로세서는 1GB, 2MB, 그리고 4KB 페이지 크기를 선택적으로 사용할 수 있으며, 리눅스는 비휘발성 메모리 영역에 대해서는 1GB 페이지 크기를 사용한다. 그런데 앞에서 설계한 영속적 페이지 테이블은 4KB 페이지 크기를 사용한다. 비휘발성 메모리 영역에 대해 1GB가 아닌 4KB 페이지 크기를 사용하면 TLB 미스율이 높아져 성능이 떨어질 가능성이 있다. 실제로 1GB 페이지 크기를 4KB로 변경한 후 다음 절에서 설명할 세 개의 벤치마크 워크로드를 사용하여 실험한 결과는 평균 1.2% 성능 하락을 보여주었다. 그러나 4KB 페이지 크기를 사용하면서 교환 연산을 사용할 때 얻어지는 성능 이득이 이보다 훨씬 커서 전체적으로 성능이 향상되었다. 5절에서 제시하는 성능 향상 정도는 4KB 페이지 사용으로 인한 성능 하락을 상쇄하고 얻어진 성능 이득이다.

3. 페이지 테이블의 원자적 변경

시스템 종속적인 페이지 테이블이 변경되면, 동

시에 L2P 맵도 변경되어야 한다. 그런데 L2P 맵은 영속적이므로 변경 도중 시스템 크래시가 발생하면 맵의 일관성이 훼손된 채로 남아있게 된다. 이러한 문제를 방지하기 위해 맵 엔트리는 원자적으로 변경되어야 한다. 시스템 종속적인 페이지 테이블의 변경은 원자적일 필요가 없는데, 왜냐하면 이 테이블은 저장장치가 연결될 때 L2P 맵으로부터 다시 생성되기 때문이다. L2P 맵을 원자적으로 변경하기 위해 맵 변경 전에 L2P 로그 영역에 트랜잭션 시작 로그를 남기도록 하였다. 그 다음 맵의 변경을 UNDO 할 수 있는 UNDO 로그들을 기록한다. UNDO 로그가 기록되면 실제로 L2P 맵(그리고 시스템 종속적인 페이지 테이블)을 변경하며, 변경이 성공적으로 완료되면 트랜잭션 커밋 로그를 남긴다(그림 2).

저장장치 디바이스 드라이버는 시스템이 부팅할 때 또는 저장장치가 연결될 때 L2P 로그 영역을 조사한다. 그리고 커밋 되지 않은 트랜잭션이 존재하면 UNDO 로그에 따라 L2P 맵을 변경 이전 상태로 되돌린다. 모두 되돌리면 L2P 로그에서 트랜잭션을 삭제한다. 트랜잭션 시작 로그, UNDO 로그, 그리고 트랜잭션 커밋 로그들은 자체적으로 체크섬 필드를 가지고 있다. 체크섬이 일치하지 않는 로그는 로그 기록 도중 시스템 크래시가 발생하였다고 가정하여 무시한다.

Ⅳ. 교환 연산과 구현

영속적 페이지 테이블 기법을 사용하면 기존 저장장치에서는 불가능한 교환 연산을 만들 수 있다. 교환 연산은 L2P 맵(그리고 시스템 종속적인 페이지 테이블)의 두 엔트리를 서로 교환하는 것이다. 그 결과 저장장치에 있는 두 블록의 내용이 뒤바뀐다. 이러한 교환 연산은 특히 동일한 데이터를 두 번 기록하는 파일시스템의 저널링 기법 오버헤드를 줄이는데 사용될 수 있다. 본 절에서는 리눅스 Ext4 파일시스템에 이러한 교환 연산을 어떻게 적용할 수 있는지에 대해 설명한다.

교환 연산을 적용하기 위해서는 파일시스템의 상세한 분석이 필요하다. 특히 파일시스템에는 정교한 오류 복구 절차가 있으며, 이러한 오류 복구 절차와 조화를 이루도록 교환 연산을 추가해야 한다. 지금부터 Ext4 파일시스템의 저널링 기법에 대해 설명하며, 다음으로 교환 기법을 활용하기 위한 Ext4 파일시스템의 변경 사항에 대해 설명한다.

1. Ext4 파일시스템의 저널링 기법

저널링 기법은 파일시스템의 데이터 일관성을 지키기 위한 기법이다[7]. 저널링 기법은 시스템 크래시로부터 데이터를 보호하기 위해 동일한 데이터를 두 번 기록한다. 먼저 저널 영역에 변경된 데이터를 트랜잭션 단위로 기록하고, 다시 원래 위치에 기록한다. 그리고 저널 영역에 트랜잭션 커밋을 기록한다. 리눅스 Ext4 파일시스템의 저널링 데몬인 JBD2는 세 가지 저널링 모드를 가지고 있다. 구체적으로 writeback, ordered, journaled 모드가 존재한다. Writeback 모드는 메타데이터에 대해서만 저널링을 수행한다. Ordered 모드 역시 메타데이터만 저널링하지만, 메타데이터를 커밋하기 전에 이와 관련된 일반 데이터 블록들을 디스크에 기록한다. 마지막으로 journaled 모드는 메타데이터 뿐만 아니라 일반 데이터 블록에 대해서도 저널링을 수행한다. 모든 모드에서 한 번 기록된 메타데이타는 손실되지 않는다. Journaled 모드인 경우 일반 데이터도 손실되지 않는다. 그러나 writeback 모드의 경우 일반 데이터는 시스템 크래시 상황에서 손실될 가능성이 있다. (그렇지만 시스템은 주기적으로 Sync를 수행하여 손실 가능 시간을 제한하고 있다.) Ordered 모드에서는 커밋된 일반 데이터는 손실되지 않지만 그 이외의 일반 데이터는 손실될 수 있다.

Ext4 파일시스템의 저널링을 구체적으로 설명하면 다음과 같다. JBD2 데몬은 일정 시간동안 생성 또는 변경된 일반 데이터와 메타데이터를 저널 트랜잭션에 등록한다. 구체적으로 JBD2 데몬은 트랜잭션에 속한 데이터의 블록 번호와 이들이 기록될 저널 영역의 블록 번호를 저널 디스크립터에 기록한 후 변경된 데이터를 저널 영역에 기록한다. Ordered 모드이면 트랜잭션의 inode 리스트를 참조하여 트랜잭션과 관련된 inode들의 일반 데이터 역시 디스크에 기록하도록 요청한다. 저널 영역에 변경된 데이터가 기록되면 JBD2 데몬은 커밋 블록을 디스크에 적는다. 커밋이 디스크에 기록되면 저널 영역의

데이터는 완전하게 기록된 상태가 된다. 다음으로 저널 시작 번호, 마지막 커밋된 트랜잭션 등의 정보가 적힌 저널 슈퍼 블록(Journal Super Block)을 업데이트한다. 그리고 커밋이 완료된 트랜잭션과 데이터를 체크포인트 리스트에 등록하여 나중에 체크포인트를 수행할 수 있도록 한다.

JBD2 데몬은 체크포인트를 미루다가 저널 영역에 빈 공간이 부족할 때 체크포인트를 수행한다. 체크포인트의 기본 동작은 저널 영역에 기록된 데이터들을 원래 자신의 위치에 다시 기록하는 것이다. 그리고 저널 슈퍼 블록을 갱신하여 저널 영역을 무효화 하고 이곳에 기록된 트랜잭션과 데이터를 폐기한다.

비정상적으로 시스템이 종료되면 JBD2 데몬은 파일시스템 마운트 시점에 복구 루틴을 수행한다. JBD2 데몬은 먼저 저널 슈퍼 블록을 참조하여 정상적으로 체크포인트 된 마지막 트랜잭션 번호를 확인한다. 그리고 그 이후 트랜잭션에 대해 복구 루틴을 수행한다. 구체적으로 커밋이 완료되지 않은 트랜잭션은 폐기하고, 커밋이 완료된 트랜잭션의 데이터는 원래 자신의 위치에 기록한다. 그 후 후 저널 슈퍼 블록을 갱신하여 저널 영역을 무효화 한다.

2. 교환 연산을 이용한 저널링 기법

저널링 기법에 따르면, 데이터는 커밋 시점에 저널 영역에, 그리고 체크포인트할 때 원래 자신의 위치에 기록된다. 이러한 두 번 쓰기는 저널링 파일시스템의 성능을 떨어뜨리는 요인이 된다. 그런데 영속적 페이지 테이블이 제공하는 교환 연산을 사용하면 한 번 쓰기만으로 저널링을 수행할 수 있다. 구체적으로 체크포인트를 수행할 때 저널 영역 블록과 원래 데이터가 위치할 데이터 블록을 교환하도록 하면 저널 영역에 기록된 최신 데이터가 원래 자신의 위치에 존재하게 된다. 그렇지만 저널 영역에는 이전 데이터가 존재하게 된다. 이는 일시적인 것으로 체크포인트가 완료되어 저널 수퍼 블록이 갱신되면 저널 영역은 무효화되고 데이터는 사라진다. 부작용을 없애기 위해 교환 연산을 호출할 때 버퍼 캐시나 페이지 캐시에 캐싱된 이전 데이터는 무효화 해야 한다.

위에서 설명한 교환 연산 사용 시나리오는 그러나 시스템 크래시가 발생하는 경우 잘 동작하지 않는다. 예를 들어 체크포인트가 교환 연산을 수행하여 이전 데이터가 저널 영역에 그리고 최신 데이터가 데이터 영역에 존재한다고 하자. 다음으로 저널 수퍼 블록을 갱신하는데, 갱신을 완료하기 전에 시스템 크래시가 발생하면 Ext4 파일시스템은 마운트 시점에 체크포인트를 다시 수행한다. 그리고 체크포인트는 교환 연산을 또 다시 수행하며, 그 결과 데이터 영역에는 이전 데이터가, 저널 영역에는 최신 데이터가 존재하게 된다. 그리고 체크포인트가 완료되어 저널 수퍼 블록이 갱신되면 최신 데이터는 손실되고 데이터 영역에는 이전 데이터가 존재하게 된다.

이 문제를 해결하기 위해 두 가지 방법을 고려하였다. 첫 번째는 교환 연산이 아니라 중복 연산을 사용하는 것이다. 중복 연산은 L2P 맵(그리고 시스템 종속적인 페이지 테이블)의 두 엔트리가 동일한 영역을 가리키게 하여 두 블록의 내용을 같게 하는 것이다. 즉 체크포인트시 저널 영역의 블록을 가리키는 엔트리를 데이터 영역의 블록을 가리키는 엔트리에 복사하여 두 블록이 모두 최신 데이터를 가지게 한다. 이러한 중복 연산은 체크포인트를 여러 번 수행해도 부작용을 일으키지 않는다. 그러나 중복 연산을 사용하면 L2P 맵 또는 페이지 테이블에 등록되지 않은 블록이 존재하며, 이 블록은 결국 저널 영역 블록을 재사용할 때 다시 맵에 등록되어야 한다. 이를 위해 미사용 블록과 중복 블록에 대한 정보를 비휘발성 메모리에 유지하고, 이 정보도 원자적으로 갱신해야 한다. 이러한 추가적인 문제

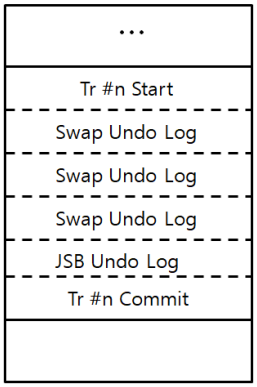

Fig. 2. Transaction records in L2P Log.
그림 2. L2P 로그에 기록되는 트랜잭션 항목들

점들은 결국 중복 연산을 사용하지 않는 이유가 되었다.

두 번째로 고려한 방법은 L2P 로그 기법을 확장하여 교환 연산과 저널 수퍼 블록 갱신을 원자적으로 수행하는 것이다. 먼저 저널 수퍼 블록의 기록이 체크포인트의 완료와 미완료를 구분하는 경계선이라는 점을 주목할 필요가 있다. 특히 블록을 교환하는 교환 연산들과 저널 수퍼 블록의 기록이 원자적으로 수행되어 모두 다 완료되거나 모두 다 수행되지 않으면 시스템 크래시 상황에서도 문제가 발생하지 않는다. 이러한 점에 주목하여 본 논문에서는 L2P 로그 기법에 저널 수퍼 블록을 원자적으로 기록하는 기능을 추가하였다. 체크포인트를 위해 처음으로 교환 연산을 수행하면 L2P 로그 영역에 트랜잭션 시작을 기록한다. 그리고 맵 엔트리 변경을 되돌리기 위한 UNDO 로그를 기록한다. 뒤따르는 교환 연산은 모두 UNDO 로그를 남긴다. 그리고 마지막으로 저널 수퍼 블록을 기록할 때 L2P 로그 영역에 이에 대한 UNDO 로그(그림 2에서 JSB Undo Log)를 기록한다. 성공적으로 저널 수퍼 블록이 쓰여지면 L2P 로그 영역에 트랜잭션 커밋을 기록하여 모든 변경이 반영되도록 한다.

L2P 로그 영역에 트랜잭션 커밋이 기록되기 전에 시스템 크래시가 발생하였다고 하자. 그러면 저장장치가 시스템에 연결될 때 실행하는 디바이스 드라이버는 L2P 로그 영역을 검색하고, 커밋되지 않은 트랜잭션에 대해 UNDO 작업을 수행한다. 트랜잭션에 남겨진 맵 UNDO 로그에 따라 맵을 원래 상태로 되돌린다. 저널 수퍼 블록에 대한 UNDO 로그가 존재하면 저널 수퍼 블록 기록 도중 오류가 발생하였다고 가정하여 성공적으로 기록되지 않은 저널 수퍼 블록 내용을 지운다. 모든 UNDO 작업을 완료하면 L2P 로그에 남겨진 트랜잭션 기록을 삭제한다. 디바이스 드라이버가 작업을 마치면 운영체제는 Ext4 파일시스템을 마운트 한다. Ext4 파일시스템은 저널 수퍼 블록이 갱신되지 않았기 때문에 체크포인트를 다시 실행한다. 교환 연산이 하나도 수행되지 않은 상태이므로 체크포인트를 다시 수행하여도 아무런 문제가 발생하지 않는다.

이 방식을 사용하면 기존 파일시스템의 변경을 최소화 하면서 교환 연산을 적용할 수 있다. 그러나 이 방식은 단점도 가지고 있는데, 특히 디바이스 드라이버에 파일시스템의 저널링을 위한 특별한 기능을 추가하였다는 점에서 디바이스 드라이버와 파일시스템 간에 계층화를 약화시킨다. 그러나 이를 사용하지 않고 파일시스템에 교환 연산을 적용하려면 많은 부분의 코드를 변경해야 하고 그 검증이 매우 어렵다. 이 점들을 감안하여 본 논문에서는 교환 연산과 저널 수퍼 블록의 기록을 원자적으로 실행하는 방법을 사용하였다.

V. 실험 및 성능 평가

1. 실험 환경

Intel에서 개발한 비휘발성 메모리 시뮬레이터인 PMEM 디바이스 드라이버를 사용하여 실험을 수행하였다. PMEM 드라이버 상단에 존재하는 Ext4 파일시스템은 비휘발성 메모리 저장장치를 기존 블록 디바이스와 동일하게 사용할 수 있다. 이 경우 Ext4 파일시스템이 제공하는 세 가지 저널링 모드를 모두 사용할 수 있다. 또는 비휘발성 메모리 저장장치를 기존 블록 디바이스와는 다르게 사용할 수도 있는데, 이를 위해서는 Ext4 파일시스템의 설정을 바꾸어 DAX(Direct Access eXited) 기능을 켜야 한다. DAX 기능은 비휘발성 메모리 저장장치의 특성을 고려하여 페이지 캐시를 거치지 않고 저장장치에서 유저 영역으로 그리고 그 반대 방향으로 바로 데이터를 전송한다. DAX 기능을 사용할 경우 메타데이터뿐 아니라 일반 데이터까지 저널링하는 journaled 모드는 이용하지 못한다. 실험에서는 Ext4 파일시스템이 journaled 모드와 ordered 모드를 사용할 때 성능을 측정하였으며, 특히 journaled 모드를 사용할 때는 비휘발성 메모리 저장장치를 기존 블록 디바이스와 동일하게 취급하고, ordered 모드를 사용할 때는 DAX 기능을 켰다.

실험에 사용된 서버는 16개의 Intel Xeon E5-2623 프로세서를 가지고 있으며 운영체제는 Ubuntu 16.04 이다. 그리고 160GB의 메모리를 가지고 있는데 이 중 32GB는 메인 메모리로, 128GB는 PMEM 영역으로 사용하였다. 파일시스템 성능을 측정하기 위하여 Filebench 워크로드[25]를 사용하였다. Filebench 워크로드는 다양한 서버 응용 프로그램의 I/O 패턴을 시뮬레이션 하며, 기존 Ext4 파일시스템과 교환 연산

을 사용하는 파일시스템을 비교하기 위해 사용되었다. 특히 실험에서는 Fileserver, Varmail, Webproxy 벤치마크 프로그램을 사용하였다. Filebench는 공통적으로 1,000,000개의 디렉토리를 생성하고 그 아래 파일을 생성한다. 그리고 500개의 스레드가 CREATE, READ, WRITE, DELETE 등으로 구성된 연산을 실행한다. Fileserver 워크로드는 세 개의 워크로드 중에서 데이터 쓰기가 가장 많다. Varmail 워크로드는 데이터 읽기와 쓰기의 비율이 약 1:1이며, 메타데이터 쓰기가 가장 많다, Webproxy 워크로드는 쓰기에 비해 읽기의 비중이 크며 작은 크기의 데이터를 기록한다.

2. 벤치마크 결과

벤치마크 프로그램을 이용하여 영속적 페이지 테이블과 교환 연산을 활용하도록 수정한 Ext4 파일시스템을 기존 Ext4 파일시스템과 비교하였다. 성능을 비교할 때 영속적 페이지 테이블과 교환 연산을 사용하도록 수정한 Ext4 파일시스템을 PPT 기법이라 부른다.

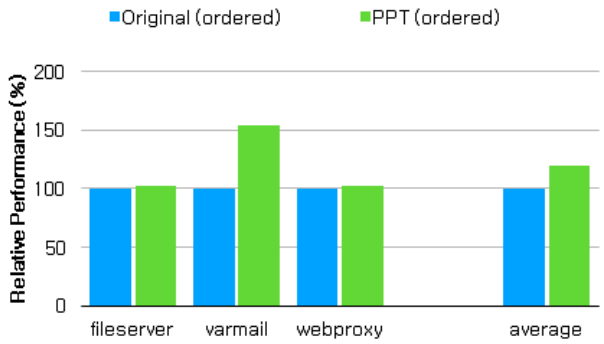

Fig. 3. Filebench performance of Ext4 using ordered mode.
그림 3. Ext4에서 ordered 모드일 때 Filebench 성능

그림 3은 DAX 설정을 켜고 저널 모드는 ordered 일 때 원래 파일시스템과 PPT 기법의 상대적 성능을 보여준다. 그림의 Y축은 원래 파일시스템의 성능을 100%라고 할 때 상대적 성능을 의미한다. 그림을 보면 PPT 기법은 세 가지 워크로드에서 2%~54%의 성능개선을 보여준다. 특히 Varmail 워크로드는 성능이 54% 향상되었다. Varmail 워크로드의 성능이 다른 워크로드보다 크게 향상된 이유는 메타데이터 연산의 비중이 높기 때문이다. Ordered 모드는 메타데이터만 두 번 쓰기를 수행하며, 일반 데이터는 한 번만 기록한다. 따라서 ordered 모드인 경우 메타데이터 연산만 영속적 페이지 테이블과 교환 연산으로 성능 이득을 볼 수 있으며, 메타데이터 연산 비중이 높은 Varmail 워크로드의 성능 이득이 다른 워크로드보다 상대적으로 크다.

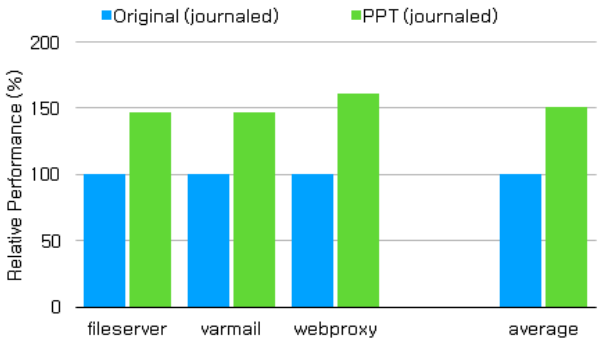

Fig. 4. Filebench performance of Ext4 using journaled mode.
그림 4. Ext4에서 Journaled 모드를 사용할 때 Filebench 성능

그림 4는 메타데이터와 일반 데이터 모두에 대해 저널링을 수행하는 journaled 모드를 사용할 때 기존 파일시스템과 PPT 기법의 상대적 성능을 보여준다. Journaled 모드인 경우, PPT 기법은 Fileserver 워크로드의 성능을 46% 향상 시킨다. 그리고 Varmail 워크로드는 46%, Webproxy 워크로드는 61% 성능이 향상되었다. 이러한 실험 결과는 당연한 것으로 PPT 기법이 두 번이 아닌 한 번 쓰기만으로 저널링을 수행하기 때문에 모든 데이터에 대해 저널링을 수행하는 경우 큰 폭의 성능 향상이 나타난다. 결국 저널링 오버헤드를 줄이는 PPT 기법은 안전성을 높이기 위해 메타데이터 뿐만 아니라 일반 데이터까지 저널링하는 journaled 모드의 사용 기회를 늘려줄 수 있다.

Ⅵ. 결론

비휘발성 메모리로 만들어진 저장장치에 접근하기 위해서는 페이지 테이블이 만들어져야 한다. 본 논문은 페이지 테이블 자체를 비휘발성 메모리에 저장하는 영속적 페이지 테이블과 영속적 페이지 테이블이 가능하게 하는 교환 연산을 제시하였다. 그리고 리눅스의 Ext4 파일시스템이 교환 연산을 사용하도록 변경하였다. 벤치마크 프로그램 결과를 보면 제안하는 기법은 저널링 오버헤드를 줄여 파일시스템의 성능을 향상시킨다. 특히 메타데이터와

일반 데이터 모두 저널링하는 journaled 모드를 사용할 때 성능을 크게 향상시킨다. 영속적 페이지 테이블과 교환 연산이 제공하는 이득은 파일시스템 이외에 다른 응용프로그램에게도 제공될 수 있다. 특히 시스템 호출 형태로 교환 연산을 제공하면 데이터베이스와 같은 응용 프로그램이 이를 이용하여 자체적으로 수행하는 저널링 오버헤드를 줄일 수 있으며, 앞으로 이에 대한 연구를 수행할 계획이다.

References

[1] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lan, "Phase-change random access memory: A scalable technology," *Journal of Vacuum Science & Technology B*, vol. 28, no. 2, pp. 223-262, 2010. DOI: 10.1147/rd.524.0465

[2] S. Gao, J. Xu, B. He, B. Choi, and H. Hu, "PCMLogging: Reducing Transaction Logging Overhead with PCM," In *Proc. of the 20th ACM International Conference on Information and Knowledge Management (CIKM 11)*, pp. 2401-2404, 2011. DOI: 10.1145/2063576.2063977

[3] T. Kawahara, "Scalable Spin-Transfer Torque RAM Technology for Normally-Off Computing," *IEEE Design & Test of Computers*, vol. 28, no. 1, pp. 52-63, 2011. DOI: 10.1109/MDT.2010.97

[4] L. Chua, "Resistance switching memories are memristors," *Appied Physics A* vol. 102, issue 4, pp. 765-783, 2011. DOI: 10.1007/s00339-011-6264-9

[5] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *Proc. of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA 16)*, 2016.
DOI: 10.1109/ISCA.2016.13

[6] Intel. "3D XPoint Unveiled－The Next Breakthrough in Memory Technology," http://www.intel.com/con-tent/www/us/en/architecture-and-technology/3d-xpoint-unveiled-video.html.

[7] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis and Evolution of Journaling File Systems," in *Proc. of USENIX Annual Technical Conference (ATC 05)*, pp. 106-120, 2005.

[8] S. Kannan, A. Gavrilovska, and K. Schwan, "pVM: Persistent Virtual Memory for Efficient Capacity Scaling and Object Storage," In *Proc. of the European Conference on Computer Systems (EuroSys 16)*, pp. 13:1-13:16, 2016.
DOI: 10.1145/2901318.2901325

[9] J.-Y. Jung and S. Cho, "Memorage: Emerging Persistent RAM based Malleable Main Memory and Storage Architecture," In *Proc. of the ACM International Conference on Supercomputing (ICS 13)*, 2013. DOI: 10.1145/2464996.2465005

[10] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "NV-Heaps: Making Persistent Objects Fast and Safe with Next-generation, Non-Volatile Memories," In *Proc. of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 11)*, pp. 105-118, 2011. DOI: 10.1145/1950365.1950380

[11] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight Persistent Memory," *Proc. of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 11)*, pp. 91-104, 2011. DOI: 10.1145/1950365.1950380

[12] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, C.-Y. M. Wang, "NVM duet: Unified Working Memory and Persistent Store Architecture," *Proc. of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 14),* pp. 455-470, 2014. DOI: 10.1145/2541940.2541957

[13] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System Software for Persistent Memory," In *Proc. of the 9th European Conference on Computer Systems (EuroSys 14)*, pp. 15:1-15:15, 2014.

DOI: 10.1145/2592798.2592814
[14] J. Xu and S. Swanson, "NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories," In *Proc. of the 14th USENIX Conference on File and Storage Technologies (FAST 16)*, pp. 323-338, 2016.
[15] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O Through Byte-Addressable, Persistent Memory," In *Proc. of the Symposium on Operating Systems Principles (SOSP 09)*, pp. 133-146, 2009.
DOI: 10.1145/1629575.1629589
[16] J. Ou, J. Shu, and Y. Lu, "A High Performance File System for Non-Volatile Main Memory," In *Proc. of the 11th European Conference on Computer Systems (EuroSys 16)*, pp. 12:1-12:16, 2016.
DOI: 10.1145/2901318.2901324
[17] S. Zheng, L. Huang, H. Liu, L. Wu, and J. Zha, "HMVFS: A Hybrid Memory Versioning File System," In *Proc. of the 32nd Symposium on Mass Storage Systems and Technologies (MSST 16)*, 2016. DOI: 10.1109/MSST.2016.7897079
[18] H. Wan, Y. Lu, Y. Xu, and J. Shu, "Empirical Study of Redo and Undo Logging in Persistent Memory," In Proc. of the 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA 16), pp. 1-6, 2016.
DOI: 10.1109/NVMSA.2016.7547178
[19] M. Liu, M. Zhang, K. Chen, X. Qian, Y. Wu, and J. Ren, "DudeTM: Building Durable Transactions with Decoupling for Persistent memory," In *Proc. of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 17)*, pp. 329-343, 2017. DOI: 10.1145/3037697.3037714
[20] A. Memaripour, A. Badam, A. Phanishayee, Y. Zhou, R. Alagappan, K. Strauss, and S. Swanson, "Atomic In-place Updates for Non-Volatile Main Memories with Kamino-Tx" In *Proc. of the 12th European Conference on Computer Systems (EuroSys 17)*, pp. 499-512, 2017.
DOI: 10.1145/3064176.3064215
[21] Y. Zhang, J. Yang, A. Memaripour, and S. Swanson, "Mojim: A Reliable and Highly-Available Non-Volatile Memory System," In *Proc. of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 15)*, pp. 3-18, 2015.
DOI: 10.1145/2694344.2694370
[22] W.-H. Kim, J. Kim, W. Baek, B. Nam, and Y. Won, "NVWAL: Exploiting NVRAM in Write-Ahead Logging," In *Proc. of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 16)*, pp. 385-398, 2016.
DOI: 10.1145/2872362.2872392
[23] C. Chen, J. Yang, Q. Wei, C. Wang, and M. Xue., "Optimizing File Systems with Fine-grained Metadata Journaling on Byte-addressable NVM," *ACM Transactions on Storage*, vol. 13, Issue 2, pp. 13:1-13:25, 2017. DOI: 10.1145/3060147
[24] J. Choi, J. Kim, and H. Han, "Efficient Memory Mapped File I/O for In-Memory File Systems," in *Proc. of the 9'th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, 2017.
[25] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A Flexible Framework for File System Benchmarking," *;login: USENIX Magazine*, vol. 41, no. 1, 2016.

BIOGRAPHY

Jae-hyeong Ahn (Member)

2016 : BS degree in Computer Science & Engineering, University of Seoul.
2018 : MS degree in Computer Science & Engineering, University of Seoul.
2018~current : Manager, Telechips Inc.

Choul-seung Hyun (Member)

2001 : BS degree in Communication & Computer Engineering, Jeju Nat'l University.
2007 : MS degree in Computer Science & Engineering, University of Seoul.
2012 : PhD degree in Computer Science & Engineering, University of Seoul.
2017~current : Research Professor, University of Seoul.

Dong-hee Lee (Member)

1989 : BS degree in Computer Engineering, Seoul Nat'l University.
1991 : MS degree in Computer Engineering, Seoul Nat'l University.
1998 : PhD degree in Computer Engineering, Seoul Nat'l University.
2001~current : Professor, University of Seoul.