https://doi.org/10.5392/JKCA.2020.20.08.368

비휘발성 메모리를 고려한 고성능 저널링 기법 설계 및 평가

Design and Evaluation of a High-performance Journaling Scheme for Non-volatile Memory

한혁
동덕여자대학교 컴퓨터학과

Hyuck Han(hhyuck96@dongduk.ac.kr)

요약

저널링 파일 시스템은 저널로 알려진 데이터 구조에 커밋되지 않은 파일 시스템의 변경 사항을 관리하여 예기치 않은 장애 발생 시 파일 시스템을 복원한다. 저널링에 필요한 추가 쓰기 연산은 저널링 파일 시스템의 성능에 부정적인 영향을 미친다. 최근 출시된 바이트 수준 접근이 가능한 고성능 비휘발성 메모리는 비휘발성 메모리 공간을 저널용 스토리지로 제공함으로써 저널링 파일 시스템의 성능 문제를 쉽게 해결할 수 있을 것으로 기대되었다. 그러나 고성능 비휘발성 메모리를 사용하더라도 저널링 파일 시스템의 트랜잭션 관리에 내재된 확장성 문제로 성능 문제는 여전히 발생한다. 이 문제를 해결하기 위해 본 논문에서는 파일 시스템 트랜잭션 처리를 위해 확장 가능한 성능을 제공하는 기법을 제안한다. 제안하는 기법은 트랜잭션 처리 상에서 락 프리 자료구조를 사용하고 여러 입출력 채널을 지원하는 고성능 저장 장치에 동시에 입출력 여러 요청들을 처리할 수 있도록 한다. 성능 평가를 위해 제안하는 기법을 ext4 파일 시스템에 구현하였고, 멀티코어 서버에서 구현된 파일 시스템과 기존 ext4 파일 시스템과 최근에 제안된 비휘발성 메모리 기반 저널링 파일 시스템을 여러 벤치마크 프로그램을 사용하여 비교했고, 이를 통해 본 연구에서 구현한 파일 시스템이 ext4 파일 시스템과 최근의 비휘발성 메모리 기반 저널링 파일 시스템보다 각각 2.9/2.3배 더 나은 성능을 보인다는 것을 보여준다.

■ 중심어 : | 저널링 파일 시스템 | 비휘발성 메모리 | 동시 자료구조 |

Abstract

Journaling file systems (JFS) manage changes of file systems not yet committed in a data structure known as a journal to restore the file system in the event of an unexpected failure. Extra write operations required for journaling negatively affect the performance of JFS. The high-performance and byte-addressable non-volatile memory (NVM) was expected to easily mitigate these performance problems by providing NVM space as journal storage. However, even with such non-volatile memory technologies, performance problems still arise due to scalability problems inherent in processing transactions of JFS. To solve this problem, we proposes a technique for processing file system transactions for scalable performance. To this end, lock-free data structures are used and multiple I/O requests are allowed to simultaneously be processed on high-performance storage devices with multiple I/O channels. We evaluate the file system with the proposed technique by comparing the original ext4 file system and the recent proposed NVM-based journaling file system on a multi-core server, and experimental results show that our file system has better performance (up-to 2.9/2.3 times) than the original ext4 file system and the recent NVM-based journaling file system, respectively.

■ keyword : | Journaling File System | Non-volatile Memory | Concurrent Data Structure |

* 이 논문은 2020년도 동덕여자대학교 연구년 제도 지원에 의하여 수행된 것임.

접수일자 : 2020년 06월 01일
심사완료일 : 2020년 08월 20일
수정일자 : 2020년 08월 20일
교신저자 : 한혁, e-mail : hhyuck96@dongduk.ac.kr

I. 서 론

현대 파일 시스템은 응용 프로그램에 충돌 일관성을 (crash-consistency) 제공하며, 이를 통해 전력 손실, 시스템 충돌과 같은 장애가 발생하는 경우 정보 시스템의 핵심 소프트웨어들인 데이터베이스 관리 시스템, 키-값 저장소, 분산 데이터 처리 프레임워크와 같은 데이터 처리 프로그램의 복구 절차를 단순하게 해준다. 충돌 일관성을 제공하는 파일 시스템에서 많이 사용되는 기법 중 하나인 저널링은 트랜잭션의 개념을 사용하여 트랜잭션의 일관성을 제공한다. 저널링 파일 시스템의 트랜잭션은 원자적이고 영속적으로 처리되어야 하는 파일 시스템의 변경들로 구성되며 파일 시스템은 변경 사항을 처리하기 위해 데이터를 원래 영역으로 쓰기 전에 저널에 데이터를 중복으로 쓴다(로그 선행 기록/Write-ahead logging). 따라서 데이터를 저널에 추가로 쓰는 연산은 저널링 파일 시스템의 성능에 직접적인 영향을 미친다.

추가 저널 쓰기로 인한 성능 문제를 완화하기 위해 이전 연구에서는 스핀 전달 토크 자기 메모리 (Spin-Transfer Torque Magnetic RAM/STT-MRAM)[1], 상변화 메모리 (Phase-Change Memory/PCM)[2], 3D-Xpoint[3] 등의 비휘발성 메모리(NVM) 기술을 기반으로 하는 저널링 기법을 제안하였다. 이러한 기법들의 핵심 아이디어는 낮은 지연 시간, 높은 대역폭, 비휘발성 등과 같은 장점을 가지는 NVM을 저널 공간으로 활용하여 파일 시스템의 입출력 경로를 최적화하는 것이다. 예를 들어 [4]의 연구에서는 데이터와 메타데이터 입출력 경로를 분리하여 NVM에 메타데이터를 관리하여 메타데이터 작업은 가속화하였다. [5]의 연구에서는 NVM의 바이트 수준 접근성을 활용하여 메타데이터 단위의 저널링을 제안하였고, [6]에서는 NVM에 데이터와 메타데이터를 비용 효율적으로 저널링하는 기법을 제안하였다.

저널링 파일 시스템의 성능을 개선하려는 연구들이 있었지만, 단순히 NVM을 저널 공간으로 사용하는 것은 최근 널리 사용되고 있는 다중 코어를 탑재한 시스템에서 성능 확장성 문제가 존재한다. 성능 확장성 문제의 주요 원인은 로그 선행 기록 원칙을 엄격하게 적용하는 저널링에서 발생한다. 저널링 파일 시스템의 저널링은 여러 개의 쓰레드가 락에 의해 보호되는 공유 데이터 구조에 접근하도록 하며, 하나의 커널 스레드가 저널링을 위한 입출력 작업을 처리한다. 이 아키텍처에서는 다음과 같이 두 가지 성능 문제가 발생할 수 있다. i) 파일 시스템은 저널링을 위해 여러 스레드가 공유 데이터 구조에 접근할 때 락 경쟁으로 인한 확장성 문제에 직면하고 ii) 최신 저장 장치가 지원하는 다중 입출력 채널을 활용하지 못하여 저장 장치의 성능을 충분히 활용하지 못하는 경우가 생긴다[7][8].

이 논문에서 이러한 문제들을 다루기 위해 확장적인 저널링 방법을 제안했다. 제안하는 저널링 기법에서는 NVM 저널 영역에 접근할 때 생길 수 있는 락 경쟁을 제거하기 위해 동시 데이터 구조를 도입한다. 또한 저널링을 위한 여러 개의 커널 수준 입출력 쓰레드들이 저장 장치로의 입출력 작업을 실행할 수 있도록 하여 복수 개의 입출력 채널을 가진 저장 장치의 성능을 최대한 활용할 수 있게 하였다. 제안하는 기법은 Linux 커널 4.1.7의 ext4 파일 시스템 위에 구현되었다. 성능 평가를 위해 Intel P3700 플래시 SSD를 탑재한 24코어 머신에서 실험을 수행하였다. 제안하는 파일 시스템은 기존의 NVM 기반 저널링 파일 시스템에 비해 최대 2.3배 향상된 성능을 보였다. 이 결과는 NVM을 고려해서 새로운 파일 시스템을 설계 및 구현하지 않고 제안하는 기법을 통해 기존 파일 시스템의 기능을 그대로 제공하면서 성능을 개선할 수 있음을 보여준다.

이 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 설명하고 3장에서는 제안하는 기법을 설명한다. 4장에서는 실험 결과를 보여주며 5장에서는 향후 연구 방향과 함께 논문의 결론을 맺는다.

II. 관련 연구 및 배경

1. 관련 연구

NVM을 고려하여 파일 시스템의 성능을 개선하기 위한 파일 시스템 최적화 연구들이 수행되어 왔다. BPFS[9]는 프로세서의 메모리 버스에 직접 연결된 NVM에 최적화된 파일 시스템이다. 메모리 버스를 통

해 대량으로 데이터를 전송하지 않고 작은 단위의 랜덤 쓰기에 적합한 파일 시스템 업데이트 연산을 수행한다. SCMFS[10]은 스토리지 클래스 메모리(SCM)을 위한 파일 시스템이다. 운영체제의 메모리 관리 및 파일 시스템을 통합하여 파일 별로 연속적인 공간을 유지하여 입출력 성능을 향상시킨다. PMFS[11]은 NVM의 바이트 수준 접근성을 활용한다. 이를 위해 mmap 인터페이스를 통해 NVM에 직접 액세스할 수 있도록 한다. NOVA[12]는 하이브리드 메모리 시스템(DRAM 및 NVM)의 성능을 극대화하도록 설계된 로그 구조 파일 시스템이다. 이러한 연구들에서는[9-12] 로그 및 파일 데이터에 NVM이 사용되며, NVM의 바이트 수준 접근성과 낮은 대기 시간을 이용하도록 설계된 새로운 파일 시스템이다. 이러한 연구들은 성능을 위해 NVM을 고려해서 파일 시스템을 새롭게 설계 및 구현한 것이며 현재 가장 많이 사용되고 있는 ext4 파일 시스템과 같은 기존 파일 시스템의 성능을 최적화한 것은 아니다.

NVM을 고려하여 기존 파일 시스템을 최적화하는 연구들 또한 수행되어 왔다. UBJ[13]는 NVM에 캐싱 및 저널링의 기능을 통합한 버퍼 캐시 아키텍처다. FSMAC[4]은 데이터 및 메타데이터 액세스 경로를 분리하여 메타데이터 액세스를 가속화하는 최적화하는 파일 시스템이다. SJM[14]은 쓰기 감소를 위한 NVM 기반의 저널링 기법을 제안하였으며 [15]의 연구에서는 모바일 장치에 최적화된 델타 저널링(DJ)이라는 저널링 기법을 제안했다. DJ는 저널 블록을 작은 크기의 NVM을 사용하기 위해 변경된 부분만을 압축하여 저장한다. [10]의 연구에서는 NVM용 메타데이터 저널링 메커니즘을 제안하고 변경된 메타데이터만 포함하는 NVM용 저널링 기법을 설계하였다. 이러한 연구들은 다중 코어를 고려하지 못 하고 있어서 성능 확장성 문제를 일으킬 수 있다. 즉, 락 기반 처리 기법에 의존하며 고성능 저장 장치에 동시에 입출력 요청하는 것은 고려하지 못하고 있다. 따라서 본 논문에서는 기존 저널링 파일 시스템의 저널 공간으로 NVM을 사용하고 락 프리 자료구조와 최적화된 저널링 입출력 구조를 사용하여 다중 코어 및 고성능 저장 장치 환경에서 성능을 최적화하고자 한다.

2. 저널링 파일 시스템

저널링 파일 시스템들은 파일 시스템의 변경 사항을 디스크에 쓰기 전에 로그라는 단위로 만들고 변경 사항들을 트랜잭션으로 처리하여 저널 영역에 쓴다. 저널 쓰기 연산이 끝나면, 변경 사항을 디스크 상의 원래 위치에 쓴다 (체크포인팅). 이를 통해 저널링 파일 시스템들은 시스템의 장애 상황에서 복구하여 응용 프로그램에 충돌 일관성을 제공한다. 본 논문에서는 ext4 파일 시스템이 일반적으로 많이 사용되기 때문에 ext4 파일 시스템에 그리고 저널링 모드는 기본 모드인 ordered 모드에 초점을 맞춘다.

Ext4 파일 시스템은 저널 영역을 위해 고정된 크기의 공간을 할당한다. 저널 영역에 순차적으로 써진 파일 시스템의 변경 사항은 파일 입출력 연산 간의 순서를 의미한다. 저널 영역과 관련된 두 가지 연산은 i) 커밋과 ii) 체크포인트이며, 커밋 연산에서 파일 시스템은 저널 영역에 변경된 페이지들을 쓴다. 체크포인트 연산에서 파일 시스템은 해당 페이지들을 저장 장치 상의 원래 위치에 쓴다. 이를 통해 체크포인트 처리가 완료된 저널 영역의 페이지들을 삭제할 수 있어서 저널 영역의 공간을 재사용할 수 있다.

커밋 연산은 실행 중인 트랜잭션을 종료하고 트랜잭션에 포함된 파일 시스템의 변경 사항을 저널 영역에 기록한다. 응용 프로그램의 프로세스에 의해 명시적으로 호출되거나 (fsync 시스템 호출) 커널 자체에 의해 주기적으로 호출된다. 저널 스레드는 커밋 연산을 수행하기 위해 먼저 트랜잭션 상태를 실행에서 커밋으로 변경한다. 그리고 트랜잭션에 포함된 페이지들의 전체 이미지들을 저널 영역에 쓴다. 이 작업이 완료되면 트랜잭션 커밋의 종료를 표시하기 위해 커밋 블록을 저널 영역에 쓴다. 따라서 커밋 블록은 파일 시스템이 복구될 때 트랜잭션의 원자성을 보장한다. 그 후에 트랜잭션의 상태를 커밋에서 커밋 완료 상태로 변경하고 트랜잭션을 체크포인트 리스트의 추가한다. 이 연산에서 트랜잭션의 상태를 변경하거나 트랜잭션의 내의 리스트 등을 접근하는 연산은 모두 락으로 보호된다.

저널 영역을 완전히 사용한 경우에는 파일 시스템은 파일 시스템의 어떠한 수정도 처리할 수 없다. 이 경우에는 저널 영역을 회수하기 위해 커밋된 트랜잭션의 체

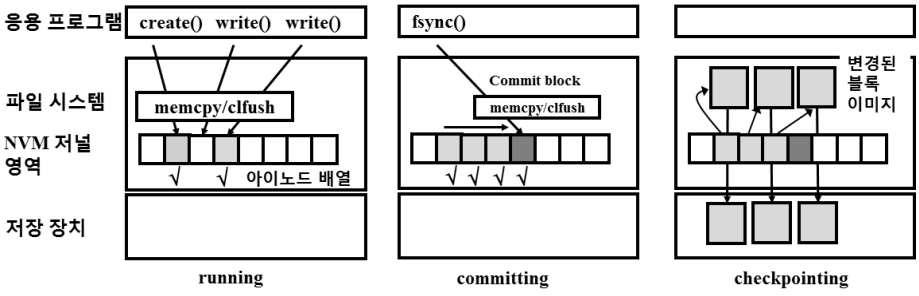

그림 1. 제안하는 기법의 전체 구조 (√ : 복사 완료 플래그)

크포인트를 수행한다. 이를 위해 체크포인트 리스트에 접근하여 체크포인트 입출력 연산을 수행하고, 다른 쓰레드들은 체크포인트 작업이 완료될 때까지 기다린다.

III. 최적화 기법

1. 저널 처리 최적화

[그림 1]은 제안하는 저널링 처리 기법의 전체적인 구조를 보여주며 왼쪽부터 오른쪽으로 차례로 트랜잭션의 실행, 커밋, 체크포인트 과정을 나타낸다. 저널 공간은 NVM에 위치하고 있다. Ext4 파일 시스템 저널링 모드 중에서 ordered 모드의 경우 파일 시스템의 메타데이터 변경만을 트랜잭션으로 관리하여 저널링을 수행한다. 그리고 파일 혹은 디렉터리의 메타데이터는 아이노드로 표시되는데 4KB 페이지 블록에 16개의 아이노드가 포함된다. [5]의 연구와 같이 본 논문에서도 NVM 저널 영역에 4KB 페이지 블록 이미지를 복사하는 것이 아니라 변경된 아이노드만 복사를 하도록 하였다. 따라서 NVM 저널 영역에 아이노드들이 복사되기 때문에 NVM 저널 영역은 아이노드 배열로 구성될 수 있다.

create(), write() 함수와 같이 파일 시스템을 변경할 때, [그림 2]에 표현된 절차대로 저널 처리를 수행한다. 1) 페이지 블록 내에서 변경된 아이노드만 추출한다. 2) 원자적 연산을 이용하여 NVM 저널 영역의 사용한 가능한 공간을 할당한다. 즉, 아이노드 배열의 인덱스 값의 원자적 더하기 연산을 이용하여 저널 영역에서 사용 가능한 공간을 락 연산 없이 계산한다. 3) 변경된 아이노드를 할당받은 NVM 저널 공간에 memcpy() 함수 등을 이용하여 복사한다. 4) clflush() 함수와 같은 CPU 캐시 플러시 연산을 수행하여 복사하고자 하는 아이노드를 NVM 메모리 모듈에 영속적으로 쓴다. 이 단계까지 완료해야 NVM 저널 영역에 데이터가 영속적으로 써짐이 보장된다. 5) 복사된 NVM 저널 공간에 복사가 완료되었음을 플래그를 통해 표시한다. 예를 들어 [그림 1]의 왼쪽에서는 3개의 파일 시스템 변경이 동시에 발생하여 저널 처리를 수행하고 있는데 3개 중 1개는 아직 메모리 복사 및 플러시 연산을 완료하지 못 하고 있음을 나타낸다.

```
Global Variables
    nvm_inode_index;
    nvm_journal[];

1: procedure PROCESS_JOURNAL(modified_block)
2:     modified_inode = get_modified_inode(modified_block);
3:     i = atomic_add(&nvm_inode_index, 1);
4:     memcpy(nvm_journal[i].storage, modified_inode);
5:     clflush(nvm_journal[i].storage);
6:     nvm_journal[i].flag = true;
7: end procedure
```

그림 2. 저널 처리 절차

2. 커밋 및 체크포인트 처리의 최적화

이 장에서는 커밋 및 체크포인트 연산의 최적화 기법에 대해 설명한다. 커밋 연산은 응용 프로그램 혹은 커널이 호출하여 실행된다. 여러 쓰레드들이 동시에 커밋 연산을 실행할 수 있기 때문에 커밋 연산들 사이의 순서를 맞추기 위해 커밋 순서를 기록하는 배열을 유지하며 이 배열에는 커밋 블록이 저장되는 아이노드 배열의 인덱스 값이 저장된다.

```
Global Variables
    nvm_inode_index;
    nvm_journal[];
    commit_order_index;
    commit_orders[];

1:  procedure PROCESS_COMMIT
2:      commit_idx = atomic_add(&commit_order_index, 1);
3:      while(commit_orders[commit_idx -1] == 0);
4:      last_commit_idx = commit_orders[commit_idx -1];
5:      commit_block_idx = atomic_add(&nvm_inode_index, 1);
6:      commit_orders[commit_idx] = commit_block_idx;
7:      while(nvm_journal[last_commit_idx].flag == false);
8:      for i ranges(last_commit_idx+1,commit_block_idx) do
9:          while(nvm_journal[i].flag == false);
10:     end for
11:     memcpy(nvm_journal[commit_block_idx].storage, com-
    mit_block);
12:     clflush(nvm_journal[commit_block_idx].storage);
13:     nvm_journal[commit_block_idx].flag = true;
14: end procedure
```

그림 3. 커밋 처리 절차

커밋 연산 [그림 3]과 같이 다음의 순서로 수행된다. 1) 원자적 더하기 연산을 이용하여 커밋 순서 배열의 비어 있는 공간을 할당받는다. 할당 받은 공간의 이전에는 커밋 순서 상 직전 커밋 연산의 커밋 블록이 저장될 아이도느 배열의 인덱스 값이 기록된다. 2) NVM 저널 영역의 사용한 가능한 공간을 할당한다. 할당받은 공간에 커밋 연산의 마지막에 트랜잭션의 원자성을 보장하기 위한 커밋 블록을 쓰게된다. 2) 커밋된 가장 최근 트랜잭션의 커밋 블록과 1)에서 할당한 영역 사이의 아이노드들이 영속적으로 쓰여졌는지 여부를 복사 완료 플래그를 통해 연속적으로 확인한다. 3) 만약 특정 아이노드의 복사 완료 플래그가 켜져 있지 않다는 것은 현재 저널 처리 중이라는 것을 의미하며 이 경우에는 이 작업이 끝나는 것을 기다린다. 4) 모든 아이노드가 영속적으로 NVM 모듈에 써졌음을 확인한 후에는 커밋 블록을 1)에서 할당받은 영역에 쓰고 캐시 플러시 연산을 수행한다.

이와 같은 방법의 장점은 커밋 연산중에도 가용한 NVM 저널 공간이 있다면 응용 프로그램들이 저널 처리 연산을 락 경쟁 없이 수행할 수 있다. 또한 응용 프로그램들이 동시에 fsync() 함수를 호출하더라도 커밋 블록의 인덱스를 통해 fsync() 함수들 사이의 순서를 보장할 수 있다.

체크포인트 연산은 커널 수준 입출력 전용 쓰레드들이 수행한다. 커널 수준 입출력 쓰레드들은 마지막으로 체크포인트된 트랜잭션의 커밋 블록과 가장 최근에 커밋된 커밋 블록 사이의 트랜잭션 단위로 다음의 절차들을 반복 수행한다. 1) 입출력 전용 쓰레들은 특정 아이노드를 원자적으로 (atomic_cas 연산) 체크포인트를 할 아이노드로 선택한다. 2) 만약 선택에 성공한다면 선택된 아이노드를 포함된 페이지 블록을 저장장치에 쓴다. 3) 선택에 실패하면 저널 공간의 다음 아이노드에 대해 다시 원자적으로 선택 연산을 수행한다. 4) 선택하고자 하는 저널 공간에 커밋 블록이 저장되어 있다면 선택에 성공한 쓰레드가 커밋 블록을 쓰고 나머지 쓰레드들은 모두 커밋 블록이 써지기를 기다린다. 따라서 커밋 블록은 다른 블록들이 써진 후에 써지기 때문에 파일 시스템이 제공하는 원자성을 보장할 수 있다. 5) 커밋 블록이 써진 후에는 커밋된 다음 트랜잭션에 대해 1)부터 체크포인트 연산을 수행한다.

이와 같은 방법의 장점은 다중 입출력 채널을 가지고 있는 고성능 플래시 SSD와 같은 저장장치의 성능을 최대로 활용할 수 있다는 것이다. 기존의 ext4 파일 시스템은 한 번에 하나의 체크포인트 입출력 연산을 처리하였다면 제안하는 기법은 복수개의 커널 쓰레드가 동시에 체크로인트 입출력 연산을 수행하기 때문에 고성능 저장장치의 최대 성능을 락 경쟁 없이 활용할 수 있다.

IV. 성능 평가

제안된 기법을 평가하기 위해 2개의 Intel Xeon CPU E5-2697 CPU를 장착한 서버를 사용하였다. 이 서버 시스템은 24개의 코어와 256GB 메인 메모리를 가지고 있으며, 이 시스템의 운영체제는 Linux 커널 4.1.7 버전이다. 본 실험에서 사용되는 저장 장치는 Intel SSD DC P3700 1.8TB이며 NVM을 위해 DRAM 256MB를 NVM으로 가정하고 실험을 수행하였다. 실험에 사용한 벤치마크는 Tokubench[16], Filebench의 Varmail이며[17] 실험을 위한 파라미터는 [표 1]과 같다. 성능 비교를 위해 ext4 파일 시스템, [5]의 NVM 기반 저널링 기법을 ext4에 구현한 파일 시스템(fg-ext4), 그리고 본 논문에서 제안하는 기법을

ext4에 구현한 파일 시스템(o-ext4)을 사용하였다.

표 1. 벤치마크 파라미터

벤치마크	파일 수	평균 파일 크기
Varmail	100,000	16 KB
Tokubench	3,000,0000	4 KB

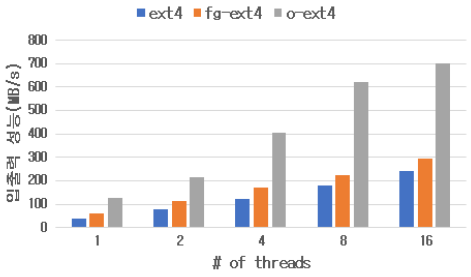

그림 4. Varmail 성능 평가 결과

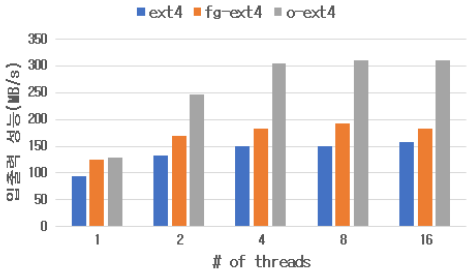

그림 5. Tokubench 성능 평가 결과

[그림 4]는 메일 서버의 입출력을 모사하는 varmail의 성능 평가 결과이다. Varmail은 파일 생성 및 데이터 추가, 동기화, 그리고 파일 삭제 작업을 수행하는 쓰기 연산 위주의 워크로드이다. 따라서 파일 생성/삭제/동기화 연산들을 수행하는 쓰레드가 많아질 때, 메타데이터 저널링 전반에 걸쳐서 성능 오버헤드가 관찰될 수 있다. 제안하는 파일 시스템은 (o-ext4)는 16 쓰레드일 때 700 MB/s의 최대 성능을 보이며 기존 ext4 파일 시스템 대비 191%, 최신 NVM 기반 저널링 파일 시스템(fg-ext4) 대비 138%의 성능 개선을 보여주었다.

[그림 5]은 Tokubench의 성능 평가 결과이다. Tokubench는 파일을 생성하는 작업을 모사한다. 파일을 생성할 때 아이노드가 생성되므로 주로 저널 처리의 오버헤드가 관찰 될 수 있다. 작업 쓰레드의 수가 16일 때 성능 향상 폭이 가장 컸다. 이 때, ext4 파일 시스템은 156.9 MB/s, fg-ext4는 183.5 MB/s, o-ext4는 309.6 MB/s의 성능을 보였다. 제안하는 기법이 기존 NVM 기반 저널링 기법보다 1.68배 정도 성능 향상을 가져왔다. 기존 파일 시스템은 저널링 및 커밋/체크포인트 연산을 수행할 때 락 경합이 발생하고 하나의 쓰레드가 체크포인트 입출력 연산을 수행하는 반면에 제안하는 파일 시스템은 이 연산들을 락 없이 수행하고, 다수의 쓰레드가 입출력 연산을 동시에 처리함에 따라 성능이 개선될 수 있었다.

V. 결론

본 논문에서는 고성능 비휘발성 메모리를 위한 저널링 기법을 제안하였다. 제안하는 기법에서 저널링, 커밋, 체크포인트 연산을 수행할 때 락 경쟁으로 인한 성능 하락을 최소화하기 위한 동시 자료 구조와 체크포인트 연산을 병렬로 수행하도록 했다. 제안하는 기법과 기존 기법을 ext4 파일 시스템에 구현하고 2개의 벤치마크를 이용하여 성능을 평가하였다. 실험 결과를 통해 제안하는 기법이 최대 138% 정도 성능을 향상시킬 수 있음을 보였다. 향후 연구에는 ext4 파일 시스템의 기본 저널링 모드 외에도 데이터 저널링 모드를 위한 최적화 기법을 제안하고 구현하고자 한다.

참 고 문 헌

[1] A. Ahari, M. Ebrahimi, F. Oboril, and M. Tahoori, “Improving reliability, perfor-mance, and energy efficiency of STT-MRAM with dynamic write latency,” IEEEICCD, 2015.

[2] J. Yue and Y. Zhu, “Accelerating Write by Exploiting PCM Asymmetries,” IEEE HPCA, 2013.

[3] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R.

Dulloor, J. Zhao, and S. Swanson, “Basic Performance Measurementsof the Intel Optane DC Persistent Memory Module,” CoRR, abs/1903.05714, 2019.
[4] J. Chen, Q. Wei, C. Chen, and L. Wu, “FSMAC: A file system metadata accelerator with non-volatile memory,” IEEE MSST, 2013.
[5] C. Chen, J. Yang, Q. Wei, C. Wang, and M. Xue, “Fine-grained metadata journaling on NVM,” IEEE MSST, 2016.
[6] X. Zhang, D. Feng, Y. Hua, and J. Chen, “A cost-efficient nvm-based journaling scheme for file systems,” IEEE ICCD, 2017.
[7] Changwoo Min, Sanidhya Kashyap, Steffen Maass, Woonhak Kang, and Taesoo Kim, “Understanding manycore scalability of file systems,” USENIX ATC, 2016.
[8] Yongseok Son, Sunggon Kim, Heon Young Yeom, and Hyuck Han, “High-performance transaction processing in journaling file systems,” USENIX FAST, 2018.
[9] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, “Better I/O Through Byte-addressable, Persistent Memory,” ACM SOSP, 2009.
[10] X. Wu and A. L. N. Reddy, “SCMFS: A File System for Storage Class Memory,” IEEE/ACM SC, 2011.
[11] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, “System Software for Persistent Memory,” ACM EuroSys, 2014.
[12] J. Xu and S. Swanson, “NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories,” USENIX FAST, 2016.
[13] E. Lee, H. Bahn, and S. H. Noh, “Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory,” USENIX FAST, 2013.
[14] L. Zeng, B. Hou, D. Feng, and K. B. Kent, “SJM: An SCM-based Journaling Mechanism with Write Reduction for File Systems,” DISCS, 2015.
[15] J. Kim, C. Min, and Y. I. Eom, “Reducing excessive journaling overhead with small sized NVRAM for mobile devices,” IEEE Transactions on Consumer Electronics, 2014.
[16] J. Esmet, M. A. Bender, M. Farach-Colton, and B. C. Kuszmaul, “The tokufs streaming file system,” USENIX HotStorage, 2012.
[17] V. Tarasov, E. Zadok, and S. Shepler, “Filebench: A flexible framework for filesystem benchmarking,” login, Vol.41, No.1, 2016.

저 자 소 개

한 혁(Hyuck Han) 정회원

- 2003년 8월 : 서울대학교 컴퓨터공학부(공학사)
- 2006년 2월 : 서울대학교 컴퓨터공학부(공학석사)
- 2011년 2월 : 서울대학교 컴퓨터공학부(공학박사)
- 2011년 3월 ~ 2012년 8월 : 서울대학교 컴퓨터공학부 박사후 연구원
- 2012년 9월 ~ 2014년 2월 : 삼성전자 메모리 사업부 책임연구원
- 2014년 3월 ~ 현재 : 동덕여자대학교 컴퓨터학과 조교수/부교수

〈관심분야〉 : 데이터베이스 시스템, 병렬 프로그래밍, 분산 시스템