https://doi.org/10.5392/JKCA.2021.21.07.157

# 저널링 파일 시스템을 위한 비휘발성 메모리 기반 병행적 저널링 기법의 설계 및 구현

## Design and Implementation of NVM-based Concurrent Journaling Scheme

**박수희, 이은영, 한혁**
동덕여자대학교 컴퓨터학과

**Suehee Pak(pak@dongduk.ac.kr), Eunyoung Lee(elee@dongduk.ac.kr), Hyuck Han(hhyuck96@dongduk.ac.kr)**

**요약**

파일 시스템에서 하나의 쓰기 연산은 여러 데이터를 수정할 수 있지만, 이러한 파일 시스템의 변경들은 원자적으로 디스크에 써지지 않는다. 따라서 파일 시스템의 일관성을 위해 기존의 저널링 기법은 시스템 성능을 저하시키는 대신 충돌 일관성을 보장한다. 비휘발성 메모리를 저널 공간으로 사용하면 비휘발성 메모리의 낮은 지연 시간과 바이트 수준 접근성으로 성능 저하를 완화시킬 수 있다고 알려졌다. 그러나 비휘발성 메모리를 고려한 저널링 기법 중에서 확장성을 제공하는 것은 없다. 본 논문에서는 확장적 저널링을 위해 비휘발성 메모리상의 저널 공간을 여러 영역으로 분할하여 한 영역에 집중된 연산을 분산시킨다. 또한, 저널 영역별로 입출력 쓰레드를 두어 저장 장치에 데이터 쓰기 연산을 가속화한다. 제안된 기법을 JFS에 적용하여 고성능 저장장치를 탑재한 멀티코어 서버에서 이를 평가한다. 평가 결과는 제안된 기법이 기존의 NVM 기반 저널링 파일 시스템의 기법보다 성능이 우수함을 보여준다.

■ 중심어 : | 충돌일관성 | 파일 시스템 | 비휘발성 메모리 | 저널링 | 멀티코어 확장성 |

**Abstract**

A single write operation in a file system can modify multiple data, but these changes in the file system are not atomically written to disk. Thus, for the consistency of the file system, conventional journaling guarantees crash consistency instead of sacrificing the system performance. It is known that using non-volatile memory as a journal space can alleviate performance degradation due to low latency and byte-level accessibility of non-volatile memory. However, none of the journaling techniques considering non-volatile memory provide scalability. In this paper, journal space on non-volatile memory is divided into multiple regions for scalable journaling, thus dispersing concentrated operations in one region. Second, the journal area-specific operator structure is used to accelerate data write operations to storage devices. We apply the proposed technique to JFS to evaluate it on multi-core servers equipped with high-performance storage devices. The evaluation results show that the proposed technique performs better than the existing technique of the NVM-based journaling file system.

■ keyword : | Crash Consistency | File System | Non-Volatile Memory | Journaling|multicore Scalability |

* 이 논문은 2019년도 동덕여자대학교 학술연구비 지원에 의하여 수행된 것임.
접수일자 : 2021년 03월 23일
심사완료일 : 2021년 06월 21일
수정일자 : 2021년 06월 21일
교신저자 : 한혁, e-mail : hhyuck96@dongduk.ac.kr

## I. 서 론

파일 시스템은 파일 추상화를 통해 스토리지 디바이스를 가상화하여 사용자에게 데이터 액세스의 간단한 방법을 제공한다. 사용자는 파일을 논리적으로 연속된 저장소로 볼 수 있으며 파일 시스템은 메타데이터를 유지 및 관리하여 물리적 스토리지 장치의 공간을 관리할 수 있다. 그러나 범용 스토리지 장치가 파일의 다중 업데이트에 대한 원자적 쓰기 속성을 지원하지 않기 때문에 일관성에 관해 제한된다는 것은 알려져 있다. 따라서 파일 업데이트 사이에 정전과 같은 일이 발생하면 파일 시스템이 일관되지 않은 상태로 유지되는 경우가 생길 수 있다.

파일 시스템의 불일치를 해결하기 위해 이전 연구에서는 여러 업데이트들에 대해 원자성을 보장하는 기법을 제안하였다. 예를 들어 널리 사용되는 기법 중의 하나인 저널링은 파일의 모든 수정사항을 단일 트랜잭션으로 그룹화한다. 그런 다음 저장 장치에 데이터 변경 사항을 쓰기 전에 전용 저널 공간에 수정 사항을 기록한다. CoW (Copy-on-Write)[1] 방법은 파일 시스템의 변경을 원래 위치가 아닌 다른 위치로 변경 사항을 쓰고 변경 작업을 파일 시스템의 루트까지 전파한다. 따라서 CoW 기반 기술은 불가피하게 새로운 데이터와 오래된 데이터를 관리하기 위해 트리와 같은 데이터 구조를 필요로 한다. 즉, 저널링 및 CoW 기반 기술들은 모두 충돌 일관성을 제공하기 위해 추가 쓰기가 필요하며, 추가 쓰기는 파일 시스템에서 쓰기 증폭을 야기한다.

충돌의 일관성을 보장하기 위한 추가 쓰기는 시스템 연구자들로부터 많은 관심을 받아왔으며, 최근에는.바이트 주소 지정성 및 낮은 지연 시간과 같은 장점 때문에 위상 변경 메모리(PCM)[2] 및 3D-XPOINT와 [3] 같은 새로운 비휘발성 메모리(NVM) 기술을 최대한 이용하려는 연구 및 시스템들이 제안되었다. 예를 들어, ext4-DAX는 [4]NVM 공간을 파일 시스템 공간으로 사용하며 메타데이터 저널링을 제공한다. NVM을 활용한 다른 저널링 파일 시스템들이 제안되었지만 이러한 시스템들은 NVM을 사용하지만 다중 코어를 최대로 활용하지 못 하는 단점이 있다.

확장성 문제의 주요 원인은 파일 변경들에 대해 반영되어야 하는 순서를 엄격히 적용하고 락에 의해 보호되는 공유 데이터 구조를 사용하는 중앙 집중식 설계이다. 중앙 집중식 설계에는 두 가지 성능 문제가 발생할 수 있다. 첫째, 파일 시스템은 수정사항을 그룹화하거나 직렬화하기 위해 공유 데이터 구조에 대한 경합이 커질 때 다중 코어 확장성 문제에 직면한다. 둘째, 최신 스토리지 장치에서 지원되는 여러 I/O 채널을 활용하지 못하는 경우가 많다. 이 확장성 문제를 해결하기 위해 분산형 설계를 기반으로 하는 파일 시스템들이[5][6] 소개된다. 그러나 이러한 파일 시스템들은 고성능의 파일 입출력을 보여주지만 완전히 재설계된 파일 시스템이기 때문에 기존의 다른 파일 시스템에 적용할 수 없다.

본 연구에서는 충돌 일관성을 보장하는 저널링 파일 시스템을 위한 고성능 저널링 기법을 제안한다. 이를 위해 저널링 작업의 동시성을 향상시키기 위한 NVM 기반 기법을 제안한다. 제안하는 기법은 충돌 일관성을 보장하면서 변경 데이터들을 유지하는 작업을 분산시키기 위해 1) 락을 배재한 복수 개의 NVM 로그 버퍼 연산, 2) 고성능 저장 장치의 성능을 최대로 활용하기 위한 병렬 I/O의 두 가지 구성 요소로 구성된다. 제안된 기법을 리눅스의 JFS에서 구현하였으며 44 프로세서 코어, 768 GiB 메모리, NVMe 플래시 SSD를 갖춘 멀티 코어 시스템에서 성능을 평가했다. 평가 결과에 따르면 기존 NVM 기반 저널링 파일 시스템보다 처리량이 최대 5.09배 더 우수하다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구와 관련된 다른 연구에 대해 설명한다. 3장에서는 제안하는 분산형 저널링 기법을 설명한다. 4장에서는 구현한 파일 시스템의 성능 결과를 보여주며 5장에서는 논문의 결론을 맺는다.

## II. 관련 연구 및 배경

NVM을 위한 파일 시스템 최적화에 대한 많은 연구가 진행되었다. SCMFS는[7] 스토리지 클래스 메모리를 위한 파일 시스템이며, 운영 체제에 메모리 관리와 파일 시스템을 통합하여 I/O 성능을 향상시킬 수 있도

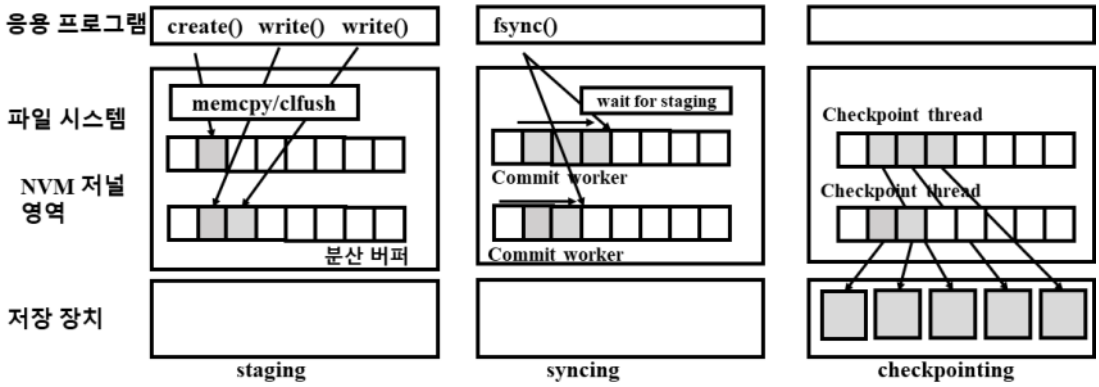


**그림 1. 제안하는 저널 최적화 기법의 시스템 구조 (NVM 버퍼를 2개로 분할하여 사용하는 예)**

록 파일마다 인접 공간을 관리한다. BPFS는[8] 프로세서의 메모리 버스에 직접 연결된 NVM에 최적화된 파일 시스템이며, 파일 업데이트를 위해 대량의 데이터를 메모리 버스로 전송하지 않고 작은 양의 데이터를 전송한다. PMFS는[9] 블록 스토리지와 관련된 오버헤드를 줄이는 POSIX 파일 시스템이다. NVM의 바이트 주소 지정성을 통해 애플리케이션이 mmap 인터페이스를 통해 NVM에 직접 액세스할 수 있도록 한다. NOVA는 [5] 하이브리드 메모리 시스템(DRAM 및 NVM)에서 성능을 극대화하도록 설계된 로그 구조 파일 시스템이다. 파일 데이터와 로그를 분리하여 성능을 극대화시킨다. 이러한 연구들에서[5][7][8] 새롭게 설계 및 개발된 파일 시스템들은 NVM을 로그 및 파일 데이터에 사용하며, NVM의 바이트 주소 지정성과 짧은 지연 시간을 활용하여 성능을 개선한다.

FLEX는[10] 기존 파일 I/O 요청을 가로채서 NVM에 메모리 매핑된 I/O를 수행하도록 하여 사용자 공간에서 파일 I/O 연산을 제공하고 이를 통해 NVM을 최대로 활용한다. 또한 세분화된 저널링을 사용하여 기존 애플리케이션에 더 나은 성능을 제공한다. UBJ[11] 연구는 캐싱 및 저널링 기능을 NVM과 통합하는 버퍼 캐시 아키텍처를 제안하였다. FSMAC[12] 연구는 데이터와 메타데이터 처리 경로를 분리하여 메타데이터 처리를 최적화하는 방법을 제안하였다. [13]의 연구에서는 모바일 기기용 델타 저널링(DJ) 기술을 제안했다. DJ는 저널 블록을 압축 델타(compressed delta)로 저장하여 작은 양의 NVM을 사용한다. [14]의 연구에서는 NVM을 고려하여 세분화된 메타데이터 저널링 기법을 제안하고 블록이 아닌 업데이트된 메타데이터만 포함하는 새로운 저널 포맷을 설계했다. 또 다른 세분화된 저널링 방법인 NJS는 메타데이터를 기록하고 데이터를 NVM에 덮어쓰도록 하여 비용 효율적인 저널링 방법을 구현하였다. 이러한 연구들은 기존 저널링 파일 시스템의 충돌 일관성을 보장하기 위해 NVM을 사용한다는 측면에서 본 연구와 유사하다. 그러나 이전 연구들은 다중 코어 확장성 문제를 일으킬 수 있는 락으로 보호되는 중앙 집중식 로깅 기술에 의존하며 체크포인트 쓰레드가 1개여서 고성능 저장 장치를 최대로 활용할 수 없는 단점이 있다.

## III. 저널 처리 최적화 기법

### 1. 저널 처리

제안하는 기법은 파일 시스템의 변경 사항을 블록 스토리지에 반영하기 전에 일시적으로 저장하기 위해 [그림 1]과 같이 분산형 NVM 버퍼를 사용한다. 파일 시스템의 변경 사항이 블록 식별자에 의해 특정 NVM 버퍼에 저장된다. 복수 개의 작업 쓰레드들이 파일 시스템이 일관된 상태에 도달할 수 있을 때만 NVM 버퍼에 저장된 변경 사항들을 블록 스토리지로 쓴다. 기존의 저널링 파일 시스템의 중앙 집중식 설계가 확장성 문제

를 가지고 있기 때문에, 제안하는 기법은 이 영역을 여러 파티션으로 분할하여 여러 쓰레드들이 버퍼에 접근할 때 생기는 경합을 분산시킨다.

```
1: procedure PROCESS_STAGING(block)
2:     s_buffer = get_nvm_staging_buffer(block.id);
3:     buf_idx = atomic_add(s_buffer.index, 1);
4:     s_buffer[buf_idx].block_id = block.id;
5:     memcpy(s_buffer[buf_idx].data, block);
6:     clflush(s_buffer[buf_idx]);
7:     s_buffer[buf_idx].flag = true;
8: end procedure
```

**그림 2. 분산 NVM 버퍼를 활용한 저널 처리**

동기화 시점(예: fsync 호출)에서 프로세스는 이전 동기화부터 현재 동기화 시점 사이에 속하는 모든 수정이 NVM 버퍼에서 완전히 커밋되는지 확인한다. 또한 체크포인트 쓰레드들은 스토리지에 커밋된 수정들을 차례대로 반영한다. 제안하는 기법에서는 고성능 체크포인트 작업을 위해 여러 개의 쓰레드가 체크포인트 입출력 작업을 수행하게 한다. 이를 통해 최신 고성능 스토리지 장치에서 일반적으로 지원되는 복수 개의 입출력 채널들을 최대로 활용한다. 체크포인트가 진행된 NVM 버퍼들은 다시 회수되어 변경 사항을 다시 저장하는데 사용된다.

사용자 프로세스가 파일 시스템의 데이터를 변경하면 [그림 2]와 같이 분할된 NVM 버퍼를 이용하여 변경 사항을 저장하며 그 절차는 다음과 같다. 우선 변경된 데이터를 포함하는 페이지의 블록 식별자를 기준으로 분할된 NVM 버퍼들 중에서 특정 NVM 버퍼를 지시하는 포인터를 얻는다. 특히 이 연산은 같은 블록 식별자를 가지면 동일한 NVM 버퍼에서 버퍼 블록을 할당 받을 수 있도록 나머지 연산자를 활용한다. 그리고 NVM 버퍼 영역의 사용 가능한 버퍼 블록 식별자를 할당받기 위해 원자적 덧셈 연산을 이용한다. 이를 통해 NVM 버퍼 블록의 할당을 락 없이 수행할 수 있다. 이 연산을 수행한 후에 메모리 복사 함수를 이용하여 변경된 블록을 할당받은 NVM 버퍼 블록에 복사하고, NVM에 완전히 써지는 것을 보장하는 clflush 함수를 수행하여 변경된 블록을 NVM 메모리 모듈에 영속적으로 쓴다. clflush 함수의 수행까지 완료해야 NVM 버퍼 영역에 변경된 데이터가 영속적으로 써졌음을 보장할 수 있다. 그리고 커밋 처리를 위해 복사된 NVM 버퍼 공간에 저널 처리가 완료가 완료되었음을 표시한다. 이러한 과정이 락을 사용하지 않고 수행되기 때문에 락 경합에 의한 성능 하락을 방지할 수 있다.

```
1: procedure PROCESS_COMMIT
2:     commit_ts_id = atomic_add(commit_ts_buf_idx);
3:     i = 0;
4:     for s_buffer ∈ all staing buffers do
5:         commit_ts[i++] = atomic_get(s_buffer.index);
6:     end for
7:     commit_ts_buffer[commit_ts_id] = commit_ts;
8:     i = 0;
9:     for s_buffer ∈ all staing buffers do
10:        while s_buffer.last_check ≤ commit_ts[i] do
11:        end while
12:        i++;
13:    end for
14: end procedure
15: procedure COMMIT_WORKER
16:    s_buffer = get_nvm_staging_buffer(worker_id);
17:    while true do
18:        while s_buffer[j].flag == false do
19:        end while
20:        atomic_set(s_buffer.last_check,j);
21:        j++;
22:    end while
23: end procedure
```

**그림 3. 분산 NVM 버퍼를 활용한 커밋 처리**

## 2. 커밋 및 체크포인트 처리

커밋 연산은 응용 프로세스가 fsync와 같은 함수를 호출할 때 수행되는 연산이며 커밋 처리는 [그림 3]과 같이 처리된다. 커밋 처리는 커밋 요청을 하는 프로세스 혹은 쓰레드 외에도 저널 처리 작업의 완료를 연속적으로 검사하는 커밋 작업 쓰레드들이 커밋 작업을 수행한다. 커밋 요청을 하는 응용 프로세스 혹은 쓰레드는 [그림 3]의 PROCESS_COMMIT과 같은 순서로 작업을 수행한다. 우선 더하기 연산을 이용하여 여러 프로세스 혹은 쓰레드들이 호출하는 커밋 연산 사이의 순서를 맞춘다 (2 줄). 그리고 원자적 읽기 연산을 이용하여 모든 NVM 버퍼들에서 가장 최근에 할당된 버퍼 블록 인덱스들을 읽어서 commit_ts라는 배열 변수에 저장한다. (3-6줄) 이후의 커밋 처리 연산은 commit_ts까지 모든 파일 시스템 변경 사항들이 NVM 버퍼 블록들에게 영속적으로 써짐을 확인하는 것이고 확인이 끝나면 커밋 처리는 완료된다. (9-13줄) 그리고 commit_ts를 commit_ts_buffer에 저장하여 체크포인트 시에 순서를 맞출 수 있게 한다. (7줄)

커밋 작업 쓰레드는 [그림 3]의 COMMIT_WORKER와 같은 순서로 작업을 수행한다. 커밋 작업 쓰레드는

분할된 NVM 버퍼의 개수만큼 수행된다. 즉, 버퍼 당 한 개의 쓰레드가 지속적으로 저널 처리의 완료를 연속적으로 확인한다. 커밋 작업 쓰레드는 담당해야할 NVM 버퍼를 찾는다. (16줄). 그리고 버퍼 블록들이 모두 영속적으로 플러시 되었는지 차례로 확인한다. (17-22줄) 이를 위해 매 버퍼마다 저널 처리 시에 기록하는 플래그를 확인한다. (18-19줄) 그리고 동시에 커밋을 수행하고 있는 응용 프로세스 혹은 쓰레드들이 마지막으로 저널 처리가 완료된 버퍼 블록을 확인할 수 있도록 last_check라는 변수를 두어 확인이 끝날 때마다 확인이 끝난 버퍼의 인덱스를 저장한다. (20줄).

체크포인트 연산은 운영체제 수준 체크포인트 쓰레드들이 작업을 수행한다. 체크포인트 쓰레드들은 분할된 NVM 버퍼의 수 만큼 생성이 되어 NVM 버퍼 전용으로 체크포인트 작업을 수행한다. 체크포인트 쓰레드들은 마지막으로 체크포인트된 commit_ts와 그 다음 커밋된 commit_ts 사이의 변경된 블록들을 모두 블록 스토리지 장치에 쓴다. 이 때 체크포인트 작업마다 쓰레드들 사이에 동기화가 이루어져야 하며 이를 위해 다음과 같은 절차를 수행한다.

1) 각 쓰레드들은 해당 체크포인트 작업이 완료되면 특정 변수를 원자적으로 1만큼 더하게 되면 마지막에 체크포인트 작업을 수행한 쓰레드들은 그 변수 값을 쓰레드의 개수와 같음을 확인할 수 있다.

2) 마지막에 체크포인트를 완료한 쓰레드는 체크포인트할 커밋이 있는지 확인하고 커밋이 있다면 전역 변수를 0으로 초기화하여 다른 쓰레드들이 다음 작업을 수행할 수 있게 한다.

최근 고성능 SSD의 경우에 입출력 채널이 복수 개이고 본 연구에서 제안한 체크포인트 연산은 체크포인트 입출력 연산을 수행하는 쓰레드가 여러 개라는 점에서 고성능 플래시 SSD 성능을 최대로 활용할 수 있다. 기존의 저널링 파일 시스템은 체크포인트 연산을 수행하는 쓰레드가 1개이며, 제안된 기법은 복수 개의 쓰레드가 체크포인트 연산을 수행하기 때문에 성능 향상 효과가 크다.

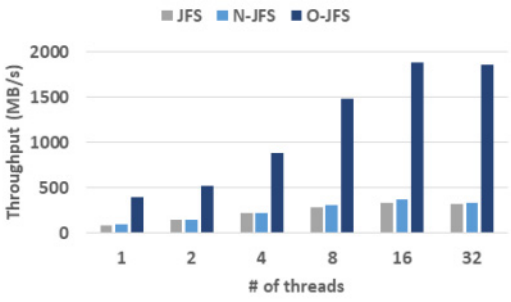


그림 4. Varmail 성능 결과

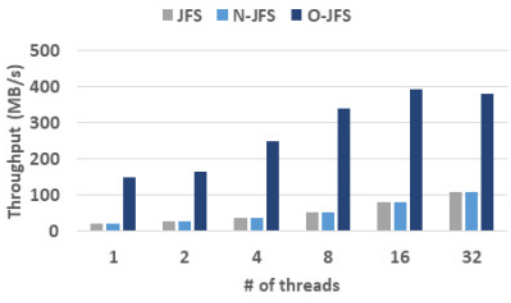


그림 4. Tokubench 성능 결과

## IV. 실험 및 평가

분산형 저널링 처리 기법을 평가하기 위해 2개의 Intel(R) Xeon(R) Gold 6152 CPU와 768GB의 메모리를 장착한 서버를 사용하였다. 이 서버 시스템은 총 44개의 코어를 가지고 있으며 운영체제는 Linux 커널 4.1.7을 사용하였다. 블록 스토리지 장치를 위해 2.0 TiB Intel DC P4510 flash SSD를 사용하였고, NVM 버퍼를 위해 DRAM 영역의 일부를 (1GB) 사용하였다. 본 연구에서 제안한 기법을 IBM의 JFS에 구현하였다. (O-JFS) JFS는 저널 공간을 별도의 블록 스토리지로 설정할 수 있으며 DRAM의 일부 공간을 블록 스토리지로 설정[1]하여 JFS의 저널 공간으로 사용한 시스템과 비교하였다. (N-JFS) 성능 평가를 위해 Varmail, Tokubench 벤치마크를 사용하였다. Varmail과 Tokubench 벤치마크에서 사용하는 파일의 수는 각각 100,000개와 3,000,000개이며 파일의 평균 크기는 각각 16KB와

1 이를 위해 다음과 같은 Linux 커널 부트 파라미터를 추가하였다. GRUB_CMDLINE_LINUX="memmap= 1G!16G"

4KB이다.

[그림 4]는 Varmail 벤치마크의 성능 결과이다. Varmail은 메일 서버에서 생성되고 읽히고 지워지는 파일 연산들을 수행하며, 파일 입출력 연산 사이에 동기화 연산이 수행된다. 제안하는 분산형 기법을 구현한 O-JFS는 16 쓰레드일 때 1879MB/s의 처리량을 보인다. 반면 SSD만을 사용하는 기본 JFS는 328MB/s의 처리량을 보이고, DRAM 공간의 일부를 블록 스토리지로 설정하여 JFS의 저널 공간을 단순 대체한 N-JFS는 369MB/s 정도의 처리를 보인다. 이를 통해 Varamail 벤치마크에서는 제안하는 기법이 JFS보다는 5.72배, N-JFS는 5.09배 더 좋은 성능을 보인다는 것을 확인하였다. 이것은 제안하는 기법이 1) 기존 기법이 저널을 처리할 때 락을 사용하는 부분을 제거하고, 2) 락은 제거한 후에 버퍼에 접근할 때 생기는 경합을 완화하기 위해 분산형 저널링 기법을 사용하였고, 3) 고성능 저장장치를 최대로 활용하기 위해 복수 개의 체크포인트 쓰레드를 사용하였기 때문이다.

[그림 5]는 Tokubench 벤치마크의 성능 결과이다. Tokubench은 많은 양의 파일을 생성하는 작업을 수행 한다. O-JFS는 16 쓰레드일 때, 392 MB/s의 처리량을 보였으며 JFS와 N-JFS는 80 MB/s의 성능을 보였다. 이를 통해 O-JFS는 N-JFS보다 4.9배 더 좋은 성능을 보임을 확인하였다. JFS와 N-JFS 성능 결과는 저널링을 위한 장치를 파일 시스템 수정 없이 초고성능의 저장 장치로 단순 대체하는 것은 성능 상의 이득이 크지 않거나 없을 수도 있음을 보여준다. 이것은 저널링 장치의 성능이 좋아질수록 저널링 관련 소프트웨어 오버헤드가 부각되기 때문이다.

## V. 결론

본 연구에서는 초고성능 비휘발성 메모리를 고려한 분산형 저널링 기법을 제안하였다. 제안하는 기법은 저널링 파일 시스템의 성능을 위해 저널링 처리에 있어서 락을 배재하였고 저널 버퍼 접근 시에 발생할 수 있는 자원 경쟁을 완화하기 위해 저널 버퍼를 분할하였다. 그리고 최신 고성능 SSD의 성능을 최대로 활용하기 위해 분할된 버퍼마다 체크포인트 쓰레드를 생성하여 입출력 연산을 수행하도록 하였다. 제안하는 기법을 실증하기 위해 Linux의 JFS에 구현하여 성능을 평가하였다. 성능 평가를 통해 제안하는 기법이 기존의 방법보다 최대 5.09배 좋아짐을 확인하였다. 또한, 저널링을 위해 성능이 좋은 저장 장치를 단순 대체하는 것은 저널링 관련 소프트웨어 오버헤드가 오히려 부각되어 성능 상의 이득이 크지 않음을 확인하였다. 향후에는 저널링 파일 시스템의 변경 사항을 페이지 혹은 블록 단위가 아닌 작은 단위로 추적하는 기술을 적용하여 성능 평가를 하고 CoW 기법에도 분산형 버퍼를 적용하여 성능을 높이는 연구를 계속 하고자 한다.

## 참 고 문 헌

[1] R. A. Lorie, “Physical integrity in a large segmented database,” ACM Transactions on Database Systems(TODS), Vol.2, No.1, pp.91-104, 1977.

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

[3] Intel, 3D-Xpoint Memory, https://www.intel.com / content/www/us/en/architecture-and- technology/optane-dc-persistent-memory.html

[4] Linux Foundation, ext4-dax, https://www.kernel.org/ doc/Documentation/filesystems/dax.txt

[5] J. Xu and S. Swanson, “NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories,” USENIX FAST, 2016.

[6] Y. Kwon, H. Fingler, T. Hunt, S. Peter, E. Witchel, and T. Anderson, “*S*trata: A cross media file system,” ACM SOSP, 2017.

[7] X. Wu and A. L. N. Reddy, “SCMFS: A File System for Storage Class Memory,” IEEE/ACM SC, 2011.

[8] 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.

[9] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson,

"System Software for Persistent Memory," ACM EuroSys, 2014.

[10] J. Xu, J. Kim, A. Memaripour, and S. Swanson. Finding and fixing performance pathologies in persistent memory software stacks, ACM ASPLOS, 2019.

[11] E. Lee, H. Bahn, and S. H. Noh, "Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory," USENIX FAST, 2013.

[12] J. Chen, Q. Wei, C. Chen, and L. Wu, "FSMAC: A file system metadata accelerator with non-volatile memory," IEEE MSST, 2013.

[13] J. Kim, C. Min, and Y. I. Eom, "Reducing excessive journaling overhead with small sized NVRAM for mobile devices," IEEE Transactions on Consumer Electronics, Vol.60, No.2, pp.217-224, 2014.

[14] C. Chen, J. Yang, Q. Wei, C. Wang, and M. Xue, "Fine-grained metadata journaling on NVM," IEEE MSST, 2016.

## 저 자 소 개

**박 수 희(Suehee Pak)** 정회원

- 1989년 2월 : 서울대학교 계산통계학과(학사)
- 1991년 8월 : 미국 University of California, San Diego, Dept. of Computer Science(공학 석사)
- 1995년 2월 : 미국 University of California, San Diego, Dept. of Computer Science(공학 박사)
- 1995년 9월 ~ 현재 : 동덕여자대학교 컴퓨터학과 조교수/부교수/교수

〈관심분야〉 : 소프트웨어공학, 멀티미디어 및 디지털 콘텐츠

**이 은 영(Eunyoung Lee)** 정회원

- 1996년 2월 : 고려대학교 컴퓨터학과(이학사)
- 1998년 8월 : 고려대학교 컴퓨터학과(이학석사)
- 2005년 1월 : 미국 Princeton University, Dept. of Computer Science(이학박사)
- 2005년 3월 ~ 현재 : 동덕여자대학교 컴퓨터학과 조교수/부교수/교수

〈관심분야〉 : 클라우드 컴퓨팅, 병렬처리 시스템, 소프트웨어 보안

**한 혁(Hyuck Han)** 정회원

- 2003년 8월 : 서울대학교 컴퓨터공학부(공학사)
- 2006년 2월 : 서울대학교 컴퓨터공학부(공학 석사)
- 2011년 2월 : 서울대학교 컴퓨터공학부(공학 박사)
- 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