ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 46, No. 9, pp. 885-891, 2019. 9
https://doi.org/10.5626/JOK.2019.46.9.885

압축 기반 파일 시스템 데이터 일관성 유지 기법

(A Compression-based Data Consistency Mechanism for File Systems)

강 동 현 † 이 상 원 †† 엄 영 익 ††
(Dong Hyun Kang) (Sang-Won Lee) (Young Ik Eom)

요 약 데이터 일관성 메커니즘은 시스템 오류 및 정전으로 인해 데이터가 손상되지 않도록 방지하기 위한 파일 시스템의 중요한 컴포넌트이다. 그러나, Ext4 파일 시스템의 기본 저널 모드는 성능상의 이유로 일반 데이터를 제외한 메타 데이터에 대한 일관성만을 보장한다. 즉, 기본 저널 모드는 파일 시스템의 전체 데이터에 대한 일관성을 완벽하게 보장하지 않는다. 이에, 본 논문에서는 파일 시스템의 데이터 일관성을 완벽하게 보장하면서 Ext4의 기본 저널 모드에 비해 향상되거나 유사한 성능을 제공하는 새로운 데이터 일관성 유지 기법을 제안한다. 제안 기법은 압축을 통해서 저널 영역에 요청되는 쓰기 요청의 양을 감소시키고 fsync() 시스템 콜 호출 횟수를 반으로 감소시킨다. 제안 기법을 평가하기 위해, 우리는 jbd2의 일부 코드를 수정하였으며, SSD와 HDD 환경에서 제안 기법의 성능을 Ext4의 두가지 저널 모드와 비교하였다. 실험 결과, 제안 기법이 기본 저널 모드 대비 최대 8.3배 시스템의 성능을 향상시킨다는 사실을 확인하였다.

키워드: Ext4 파일 시스템, 데이터 일관성, 저널 모드, 압축 및 압축 해제

Abstract Data consistency mechanism is a crucial component in any file system; the mechanism prevents the corruption of data from system crashes or power failures. For the sake of performance, the default journal mode of the Ext4 file system guarantees only the consistency of metadata while compromising with the consistency of normal data. Specially, it does not guarantee full consistency of the whole data of the file system. In this paper, we propose a new crash consistency scheme which guarantees strong data consistency of the data journal mode by still providing higher or comparable performance to the weak default journal mode of the Ext4 file system. By leveraging a compression mechanism, the proposed scheme can halve the amount of write operations as well as the number of fsync() system calls. For evaluation of the performance, we modified the codes related to the jbd2 and compared the proposed scheme with two journaling modes in Ext4 on SSD and HDD. The results clearly confirm that the proposed scheme outperforms the default journal mode by 8.3x times.

Keywords: Ext4 file system, data consistency, journal mode, compression, decompression

· 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임.
(No. NRF-2015M3C4A7065696)

† 정 회 원 : 동국대학교 컴퓨터공학과 교수
dhkang@dongguk.ac.kr
†† 종신회원 : 성균관대학교 소프트웨어학과 교수(Sungkyunkwan Univ.)
swlee@skku.edu
yieom@skku.edu
(Corresponding author임)

논문접수 : 2019년 3월 15일
(Received 15 March 2019)
논문수정 : 2019년 5월 1일
(Revised 1 May 2019)
심사완료 : 2019년 5월 26일
(Accepted 26 May 2019)

정보과학회논문지 제46권 제9호(2019. 9)

1. 서 론

Ext4 파일 시스템은 메타 데이터 저널 모드(Ordered Journal Mode)와 데이터 저널 모드(Data Journaling Mode)를 제공하며, 메타 데이터 저널 모드는 Ext4 파일 시스템에서 기본 저널 모드로 사용된다. 이는, 데이터 저널 모드가 데이터의 일관성을 완벽하게 보장하는 대신 메타 데이터 저널 모드보다 2배 많은 양의 쓰기를 수행하기 때문이다. 그리고 이러한 많은 양의 쓰기 명령은 CPU보다 느린 스토리지의 접근을 빈번하게 요구하기 때문에 시스템의 전반적인 성능을 감소시킨다[1,2]. 반면, 메타 데이터 저널 모드는 시스템 오류 및 정전(System Crash 또는 Power Failure)이 발생할 경우, 파일 시스템의 메타 데이터만 복구할 수 있기 때문에 모든 데이터를 완벽하게 복구할 수 없다는 한계를 가지고 있다[1]. 이에, 데이터 자체의 일관성(Data Consistency) 유지가 중요한 시스템(예: 은행, 항공, 등)의 경우, 파일 시스템의 일관성 유지 기법과 함께 어플리케이션의 일관성 유지 기법을 별도로 수행함으로써 데이터의 일관성을 보장한다[3,4].

지금까지 데이터 일관성 유지와 시스템의 성능 한계를 동시에 해결하기 위해 다양한 연구가 진행되었다[5-13]. 예를 들어, 데이터 저널 모드의 2번 쓰기 요청을 1번의 쓰기 요청으로 변경하여 시스템의 성능을 향상시키고 주소 재 매핑(Address Remapping)을 이용하여 메타 데이터와 데이터의 일관성을 모두 보장하는 연구가 대표적인 최신 연구이다[5]. 또한, 비 휘발성 메모리(Non-volatile Memory)을 이용하여 저널 모드의 쓰기 요청을 완화시키는 연구도 활발하게 진행되었다[6,7]. 그러나, 기존 연구에서 제안한 기법들은 모두 특별한 하드웨어 기능을 요구하기 때문에 범용적으로 사용하기에는 아직 어려움이 있다[5-7]. 이에, 본 논문에서는 컴퓨터 시스템이 일반적으로 많이 사용하는 압축 기법 이용하여 데이터 일관성을 보장하고 시스템의 성능을 향상시킬 수 있는 새로운 데이터 일관성 유지 기법을 제안한다. 특히, 본 논문에서 제안하는 일관성 유지 기법은 순차 쓰기(Sequential Write) 패턴을 활용함으로써, 데이터 저널 모드의 쓰기 오버헤드(Overhead)을 최대한 완화시킨다. 또한, 본 논문에서 우리는 다양한 파일 시스템 워크로드를 이용하여 제안 기법이 Ext4 파일 시스템의 기본 저널 모드에 비해 우수함을 보인다. 실험 결과, 제안 기법이 Ext4의 기본 저널 모드보다 최대 8.3배 시스템의 성능을 향상시킬 수 있음을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 Ext4 파일 시스템의 메타 데이터 저널 모드와 데이터 저널 모드를 자세히 살펴보고 두 저널 모드의 성능 차이를 확인한다. 3장에서는 본 논문에서 제안하는 기법에 대해서 상세하게 설명하고 4장에서는 제안 기법의 성능 결과를 다양한 실험군과 함께 보여준다. 마지막으로 5장에서 본 논문의 결론을 맺는다.

2. 연구 배경

본 장에서는 Ext4 파일 시스템의 두 가지 저널 모드의 동작 과정을 살펴보고 최신 스토리지 기반의 성능 평가를 통해 두 저널 모드의 성능 특성을 알아본다.

2.1 Ext4 파일 시스템의 저널 모드

Ext4 파일 시스템은 현재 가장 널리 사용되고 있는 파일 시스템 중 하나이며, 지금까지 Ext4 파일 시스템의 데이터 저널 모드에 따른 성능 하락을 완화시키기 위해 학계와 산업계는 다양한 연구를 진행해왔다[1-13].

데이터 저널 모드(Data Journal Mode)는 저널 커밋(Journal Commit)이 발생하면 변경된 파일 시스템의 메타 데이터와 데이터를 모두 저널 영역(Journal Area)에 기록한다. 그리고 저널 커밋 과정에서, 저널 영역 내부에서의 쓰기 순서를 보장하기 위해 두번의 fsync() 시스템 콜을 호출하여 업데이트된 메타 데이터와 데이터의 영속성(Persistency)을 보장한다. 저널 영역에 기록이 완료된 후, Ext4 파일 시스템의 데이터 저널 모드는 페이지 캐시의 영역이 부족하거나, 저널 체크 포인트(Checkpoint)가 발생할 때, 변경된 데이터를 파일 시스템의 원본 위치에 덮어쓴다(Overwrite 또는 In-place Update). 그 결과, 데이터 저널 모드의 경우, 동일한 데이터를 저널 영역과 원본 영역에 각각 2번씩 기록하게 되고 이는 시스템의 전체 성능을 감소시킨다. 반면, 메타 데이터 저널 모드(Ordered Journal Mode)의 경우, 저널 커밋이 발생할 때 변경된 데이터에 속하는 메타 데이터만 저널 영역에 기록하고 데이터는 원본 위치에 바로 덮어쓴다. 다시 말해, 파일 시스템의 변경된 메타 데이터는 저널 영역과 원본 영역에 각각 2번 기록되지만 데이터는 파일 시스템의 원본 영역에 1번만 기록된다. 그러므로, 성능이 느린 스토리지(예: HDD)와 블록 디바이스 인터페이스(예: SATA2) 환경에서는 데이터 저널 모드보다 쓰기 요청이 적은 메타 데이터 저널 모드가 시스템의 성능 측면에서 유리하다. 그러나, 메타 데이터 저널 모드는 갑작스러운 시스템 오류가 발생할 경우, 파일 시스템의 메타 데이터는 복구할 수 있지만 데이터를 복구할 수 없다는 데이터 일관성 관점에서의 단점을 가지고 있다.

2.2 Ext4 저널 모드의 성능 특성

오늘날의 스토리지 기술은 NAND 플래시 스토리지(예: eMMC, SD Card, SSD, NVMe SSD)의 등장으로 상당한 발전을 거듭하였다. 특히, 최근에 출시되는 스토

리지의 (예: HDD, SSD, NVMe SSD) IO 성능은 과거에 비해 상당히 향상되었으며, 이를 지원하기 위한 인터페이스 기술(예: SATA3, NVMe) 역시 향상되었다. 예를 들어, SATA3 인터페이스는 현재 6 Gbps의 I/O 전송 속도를 제공하며 이는 SATA2 인터페이스에 비해 2배 향상된 성능이다[14]. 이에, 본 논문에서는 최신 스토리지와 인터페이스 환경에서 Ext4의 각 저널 모드에 대한 성능 특성을 다시 확인하고 이를 비교하고자 한다. 실험을 위해 우리는 최신 Seagate HDD(2TB)와 Samsung SSD850 PRO (256GB)를 실험 스토리지 장치로 사용하였으며, 두 스토리지 장치는 SATA3 인터페이스를 통해 연결하였다. 또한, IO 기반의 실험을 위해 Filebench(1.5 버전)의 파일 시스템 워크로드인 Varmail과 File Server 워크로드를 사용하였다[15,16].

그림 1은 Filebench(1.5 버전)의 최신 스토리지 환경에서 메타데이터 저널 모드와 데이터 저널 모드의 IO 처리량(Throughput)을 비교하여 보여준다(메타 데이터 저널 모드와 데이터 저널 모드는 각각 OJ와 DJ로 표기되었음). 우리가 예상하였듯이, 빈번하게 fsync()을 호출하는 Varmail 워크로드의 경우, 데이터 저널 모드(DJ)가 메타 데이터 저널 모드(OJ)보다 향상된 성능을 보여준다. 이러한 실험 결과는 데이터 저널 모드가, 저널 커밋 시점에 저널 영역에 데이터(즉, 메타데이터와 데이터)을 모두 순차적으로 기록하기 때문이다. 즉, 빈번하게 저널 커밋이 발생할 때, 순차 쓰기(Sequential Write) 패턴은 시스템의 전체 성능을 향상시킬 수 있는 기회를 시스템과 스토리지에 각각 제공한다. 첫 번째 기회는 순차 쓰기 패턴이 호스트 소프트웨어(Software)의 IO을 처리 오버헤드를 줄여주는 것이다. 즉, 호스트 소프트웨어는 스토리지에 블록(Block) IO을 요청하기 위해, 기본적으로 Bio 구조를 이용하며, 순차 쓰기 패턴은 생성되는 Bio를 단순하고 쉽게 생성할 수 있도록 도와준다. 그 결과, Bio의 생성 개수를 감소시킬 수 있다. 두 번째 성능 향상 기회는 스토리지 내부에서 발생한다. 잘 알려져 있듯이, HDD와 SSD는 순차적인 쓰기 패턴에 최적화된 스토리지 장치이다. HDD는 데이터를 영구적(Persistent)으로 기록하기 위해 실린더(Cylinder)와 디스크 암(Arm)을 이동시키는 기계적인 움직임을 수행하는데 순차적인 패턴의 쓰기 요청이 이러한 기계적인 움직임이 최소화시키기 때문이다. 또한, NAND 기반 스토리지인 SSD 역시 순차 쓰기 패턴이 SSD 스토리지 내부의 병렬성(Parallelism)을 최대화하고, 가비지 컬렉션(Garbage Collection) 오버헤드(Overhead)을 감소시키기 때문에 순차적인 쓰기 패턴에 유리하다는 사실은 이미 잘 알려져 있다[17,18].

한편, fsync()을 거의 호출하지 않는 Fileserver 워크로드의 경우, 데이터 저널 모드(DJ)의 성능이 메타 데이터 저널 모드(OJ)의 성능보다 상당히 감소하는 것을 확인할 수 있다. 이러한 패턴은 메타 데이터 저널 모드를 Ext4의 기본 저널 모드로 사용하는 이유를 잘 설명해 준다. 이러한 성능 감소의 원인을 좀 더 자세히 확인하기 위해, 우리는 데이터 저널 모드의 동작 과정을 상세히 분석하였다. 그 결과, 데이터 저널 모드가 파일 시스템의 메타 데이터와 데이터를 모두 저널 영역에 저장하기 때문에 포-그라운드 체크 포인트(Foreground Checkpoint)을 메타 데이터 저널 모드에 비해 상당히 많이 호출한다는 사실을 확인할 수 있었다. 포-그라운드 체크 포인트는 저널 영역이 부족할 때 저널 내부에서 여유 공간(Free-space)을 생성하기 위해 강제적으로 수행되는 체크 포인트이며, 저널과 연관된 IO 동작을 일시적으로 모두 멈추기(Suspend) 때문에 시스템의 성능을 상당히 감소시킨다[19].

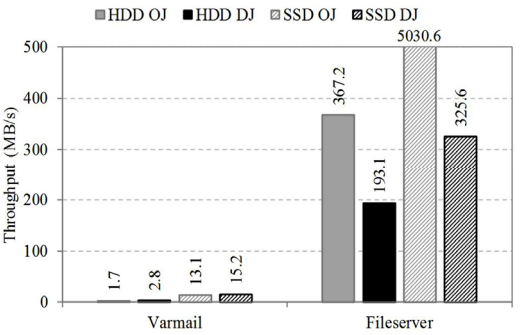

그림 1 Ext4의 두가지 저널 모드의 성능 비교
Fig. 1 Performance Comparison of Two Journal Mode on Ext4

3. 제안 기법: 압축 기반 데이터 일관성 유지 기법

2장에서 우리는 Ext4의 두 저널 모드의 동작 과정과 성능을 비교하였으며, 데이터 저널 모드(Data Journal Mode)을 사용하여 Ext4 파일 시스템의 데이터 일관성을 완벽하게 보장하며, 시스템의 성능을 향상시킬 수 있는 가능성을 확인하였다. 이에, 본 장에서는 Ext4의 데이터 저널 모드를 확장한 새로운 저널 기법을 소개한다. 특히, 본 논문에서 제안하는 기법은 저널 영역에 저장하는 저널 데이터의 크기를 줄이기 위해 압축 및 압축 해제(Compression/Decompression) 기술을 이용한다. 그림 2는 제안 기법의 전반적인 처리 절차를 보여준다.

3.1 비 손실(Data Lossless) 압축 기반 저널 커밋

제안 기법은 파일 시스템의 데이터를 완벽하게 보장하기 위해 Lempel-Ziv와 Fixed Huffman 기반의 데이터 비 손실(Lossless) 압축 기법을 이용하였다[20]. 제안

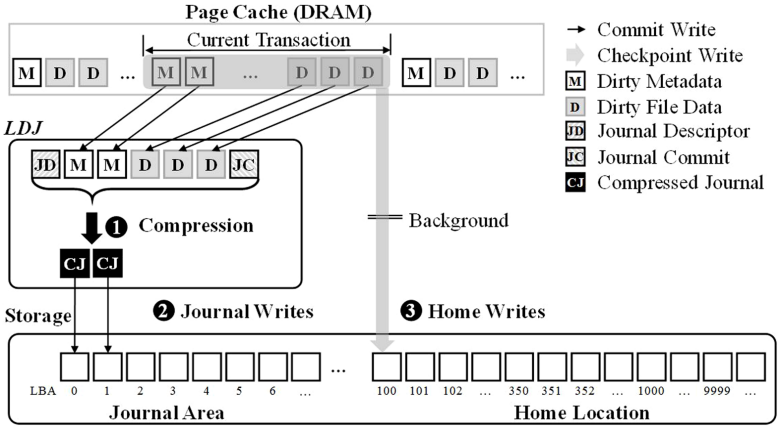

그림 2 제안 기법의 저널 과정
Fig. 2 Journal Process of the Proposed Scheme

기법에서 사용한 비 손실 압축(Compression) 기법은 데이터에서 중복된 문자열을 찾고 이를 사전(Dictionary)에 기록함으로써 중복된 문자열을 제거하는 방식으로 전체 데이터의 크기를 데이터 손실 없이 감소시킬 수 있다[20]. 압축된 데이터는 압축될 때 저장된 사전의 데이터를 기반으로 압축 해제(Decompression)을 시도한다. 그러나, 이때 저장된 사전과 압축된 데이터의 문자열이 일치하지 않는다면 압축 해제는 실패한다. 다시 말해, 제안 기법에서의 압축된 데이터는 전체 데이터가 압축 해제되거나 해제되지 않기 때문에 데이터 일관성에 중요한 요소 중 하나인 원자성(Atomicity)을 완벽하게 보장한다. 특히, 제안 기법은 저널 커밋 시점에 저널 디스크립터(Journal Descriptor) 블록, 변경된 메타 데이터와 데이터 블록, 그리고 저널 커밋(Journal Commit) 블록을 모두 하나로 압축하기 때문에 저널 내부에서 순서를 보장하기 위해 필요했던 첫 번째 fsync()을 생략할 수 있다(그림 2의 ❶). 즉, 전통적인 저널 방식에서 사용하던 두번의 fsync()호출을 한번의 fsync()호출로 감소 시킬 수 있다. 게다가, 제안 기법의 압축된 저널(Compressed Journal) 블록의 개수는 기본 데이터 저널 모드의 저널 블록 개수보다 적기 때문에 스토리지 쓰기 요청을 상당히 감소시킨다(그림 2의 ❷). 압축된 저널의 크기를 저널 데이터 블록의 0번째 오프셋(Offset)에 4 바이트 (Byte)의 크기로 저장한 후 압축된 저널 데이터 블록을 저널 영역에 기록하였다. 이는 데이터 복구 시점에 압축된 저널의 위치(Offset)를 확인하기 위해 사용된다.

3.2 Lazy 저널 체크 포인트

제안 기법은 저널 데이터를 압축함으로써 저널 영역의 공간적 활용도를 기존 데이터 저널보다 향상시켰다. 그러나, 저널 영역이 부족하여 발생하는 포-그라운드 체크 포인트(Foreground Checkpoint)의 발생 가능성은 여전히 메타 데이터 저널 모드보다 높다. 이에, 제안 기법에서는 기본 저널 크기(128MB)를 확장함으로써, 포-그라운드 체크 포인트를 백-그라운드 체크 포인터(Background Checkpoint)로 변경하기 위한 Lazy 체크 포인트(Checkpoint)를 이용하였다[19](그림 2의 ❸). 물론, Lazy 체크 포인트는 기본 저널 크기의 확장함으로써, 체크 포인트 시점에 더 많은 덮어쓰기 요청을 발생시킨다. 그러나, 이러한 덮어쓰기는 모두 백-그라운드로 수행될 수 있기 때문에 시스템의 전체 성능에 직접적인 영향을 거의 주지 않는다.

3.3 잘못된 복구(False Recovery) 방지 기법

시스템 오류 및 정전(System Crash 또는 Power Failure)이 발생한 후 제안 기법은 저널 영역에서 압축된 저널(Compressed Journal)을 읽은 후 압축해제 절차를 진행함으로써 데이터를 안전하게 복구한다. 그러나, 저널 영역은 라운드-로빈(Round-robin) 방식으로 재사용되기 때문에 잘못된 복구(False Recovery)의 가능성이 존재한다. 그림 3은 과거에 저장된 압축 데이터가 라운드-로빈 방식에 의해 현재 압축된 데이터의 사전(Dictionary)과 우연히 일치하는 경우 발생할 수 있는 잘못된 복구의 예제를 보여준다. 그림 3의 (a)는 제안 기법을 통해서 압축된 저널 블록 3개 (CJ1, CJ2, CJ3) 중 첫 번째 블록(CJ1)이 저장되지 않은 상태에서 시스템 복구가 수행된 경우를 보여준다. 그리고, 그림 3의 (b)와 (c)는 압축된 저널 블록 3개 중 두번째 그리고 세 번째 블록이 저장되지 않았지만 우연히 일치하는 경우를 각각 보여준다. 이렇게 저널 블록이 순서와 상관없이 스토리지에 저장되는 것은 스토리지 내부의 상황에 따

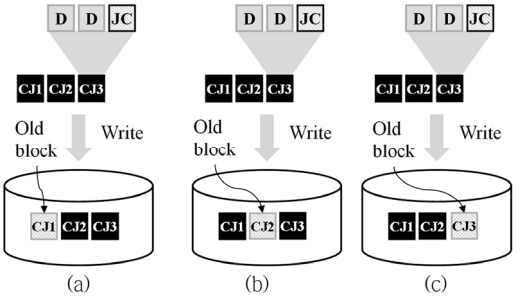

그림 3 제안 기법의 잘못된 복구 예제
Fig. 3 An Example of False Recovery on the Proposed Scheme

라서 언제든지 발생할 수 있다. 이에, 잘못된 복구 피하기 위해서 우리는 체크 포인트가 완료된 저널 블록의 데이터를 지우는 동작을 백-그라운드로 수행한다(즉, 해당 저널 블록에 “0x00”을 기록함).

4. 실험 및 평가

본 장에서는 제안 기법의 성능을 평가하기 위한 실험 환경과 워크로드에 대해서 간단하게 정리하고 실험 결과를 보여준다.

4.1 실험 환경

본 논문에서 제안하는 기법을 평가하기 위해 최신 Seagate HDD(2TB)와 Samsung SSD850 PRO(256GB)를 실험 장비로 사용하였으며, 리눅스(커널 버전 4.9) 환경에서 Ext4 파일 시스템과 jbd2 코드를 수정함으로써 제안 기법을 구현하였다. 특히, 제안 기법에서 필요한 압축 및 압축 해제 동작을 런타임으로 수행하기 위해, 본 논문에서는 Lz4 압축 알고리즘을 사용하였으며, jbd2의 커밋 시점에 Lz4 알고리즘을 수행할 수 있도록 코드를 추가 및 수정하였다.

우리는 제안 기법의 성능을 평가하기 위해, 서버 환경의 IO 워크로드를 생성하는 Filebench (1.5 버전)[15,16]와 Postmark 벤치마크[21]를 선정하였으며, Filebench에서는 fsync() 명령이 빈번하게 수행되는 Varmail과 빈번하게 수행되지 않는Fileserver 워크로드를 실험 대상으로 선택하였다. 또한, 공정한 실험 결과를 얻기 위해, 우리는 fdisk, format, mount 명령을 모든 실험 전에 순차적으로 수행하였다.

4.2 실험 결과

본 논문에서 제안하는 기법은 Ext4 파일 시스템을 기반으로 구현되었기 때문에 Ext4의 메타 데이터 저널 모드 모드(OJ)와 데이터 저널 모드(DJ)을 제안 기법과 함께 비교하였다.

그림 4와 그림 5는 Filebench 워크로드를 SSD와 HDD에서 수행한 실험 결과를 보여준다. 그림에서 4와

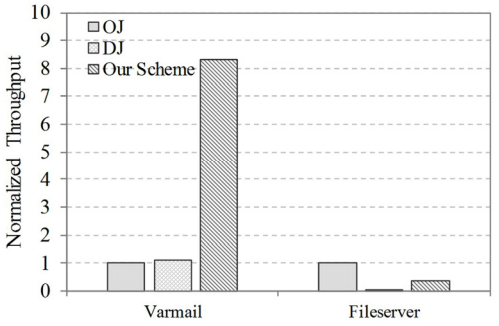

그림 4 Filebench 수행 결과(SSD)
Fig. 4 Filebench Results on SSD

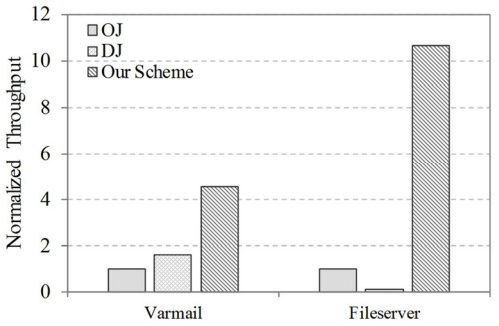

그림 5 Filebench 수행 결과(HDD)
Fig. 5 Filebench Results on HDD

그림 5에 보여주듯이, 제안 기법은 fsync() 명령이 빈번하게 수행되는 Varmail 워크로드에서 메타 데이터 저널 모드에 비해 8.3배(SSD), 4.5배(HDD) 각각 성능이 향상되는 것을 확인할 수 있다. 이는 제안 기법의 경우, 압축을 통해서 커밋 과정에서 발생하는 쓰기 요청 양과 fsync()의 호출을 한 번으로 감소시켰기 때문이다. Fileserver 워크로드의 경우, HDD에서는 제안 기법이 메타 데이터 저널 모드보다 10.6배 향상된 성능을 보여주는 반면 SSD에서는 메타 데이터 모드보다 성능이 감소하는 것을 확인할 수 있다. 이는 HDD의 경우, 쓰기 패턴과 IO 성능이 밀접한 연관성이 있으며 저널 모드의 순차 쓰기 패턴이 메타데이터 저널 모드의 임의 쓰기 패턴보다 IO 성능 측면에서 유리하다는 사실을 보여준다. 좀 더 정확한 실험 결과를 얻기 위해, 우리는 Postmark 벤치마크의 실험을 진행하였으며, 그림 6과 그림 7은 Postmark 벤치마크의 실험 결과를 SSD와 HDD에서 각각 보여준다. 우리는 본 실험에서 10,000개의 파일 생성, 25,000 서브디렉토리, 50,000 트랜잭션, 그리고 1MB 크기의 읽기/쓰기 명령을 수행하도록 Postmark의 실험 환경을 설정하였다. 또한, 저널 모드에 따른 성능 부하(Overhead)를 확인하기 위해, Postmark 벤치마크 툴이

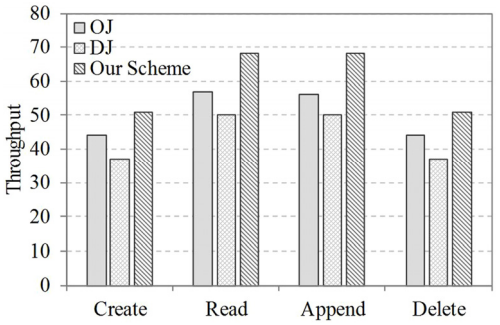

그림 6 Postmark 수행 결과(SSD)
Fig. 6 Postmark Results on SSD

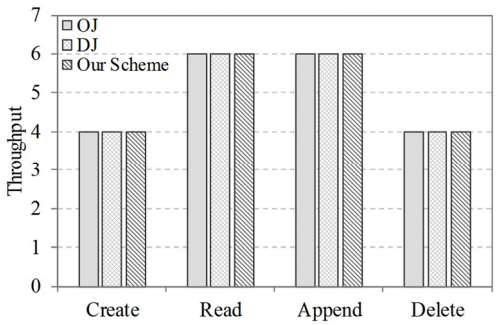

그림 7 Postmark 수행 결과(HDD)
Fig. 7 Postmark Results on HDD

워킹 셋(Working Set)을 생성할 때마다 fsync() 시스템 콜을 호출하도록 Postmark벤치마크의 코드를 수정하였다.

그림 6에서 보여주듯이, SSD의 경우, Postmark 실험 결과는 Filebench의 실험 결과와 비슷한 패턴을 보였으며, 기존 메타데이터 저널 모드 (OJ)보다 제안 기법이 최대 21% 향상된 성능을 보여준다. 반면, HDD의 경우 (그림 7), 저널 모드의 쓰기 양, fsync() 시스템 콜의 호출 횟수와 관계없이 3개의 저널 모드가 모두 동일한 성능을 보여준다. 이는 많은 양의 쓰기 요청과 빈번한 fsync()로 인해 스토리지 병목 현상(Bottleneck)이 발생하기 때문이다.

마지막으로, 우리는 제안 기법에서 사용하고 있는 압축 기법에 대한 성능 부하를 분석하고자 한다. 이는 저널 커밋 시점에 발생하는 성능 부하가 제안 기법의 성능에 상당한 영향을 미칠 수 있기 때문이다. 본 실험을 위해 우리는 jbd2에서 Lz4 압축 알고리즘을 수행하기 전과 후에 ktime_get() 명령어를 추가하였다. 실험 결과, Varmail 워크로드는 107us, Fileserver 워크로드는 109ms, 그리고 Postmark 워크로드는 169us의 평균 압축 시간이 소모 되었음을 확인할 수 있었다. 본 실험 결과에서 보여주듯이, 압축에 소모되는 시간이 Fileserver 워크로드의 경우에는 적지 않다. 그러나, 압축을 통해 스토리지로 요청하는 쓰기 요청 양을 줄일 수 있기 때문에 CPU의 압축 시간에 따른 성능 부하는 이해할 수 있는 수준이다. 특히, 그림 5에서 보여주듯이 느린 IO 성능을 가진 HDD의 경우에는 압축으로 인한 성능 부하보다 쓰기 요청 양의 감소가 전반적인 성능 측면에서 더 효과적이라는 사실을 확인할 수 있다.

5. 결 론

본 논문에서는 효율적인 데이터 일관성 유지 기법을 제안하기 위해, 기존 Ext4 파일 시스템의 데이터 저널 모드와 메타 데이터 저널 모드의 동작 방식을 간단하게 확인하였으며, 각 저널 모드의 성능을 SSD와 HDD 에서 각각 측정함으로써, 저널 모드의 성능 특성을 분석하였다. 또한, 분석한 성능 특성을 기반으로 압축을 이용하여 fsync() 시스템 콜의 호출을 한 번으로 줄이고 저널 되는 데이터의 양을 줄이는 새로운 데이터 일관성 유지 기법을 제안하였다. 마지막으로, 제안 기법의 평가를 위해 가장 널리 사용되고 있는 Filebench와 Postmark 벤치마크 툴을 사용하였으며, SSD와 HDD에서의 성능 평가를 각각 진행하였다. 실험 결과 제안 기법의 성능이 기존 메타 데이터 저널 모드 대비 최대 8.3 배 향상되는 것을 확인할 수 있었으며, 향후 연구로는, 실제 데이터를 기반으로 실험을 확장하여 제안 기법을 좀 더 자세히 검증하고자 한다.

References

[1] V. Chidambaram, T. Sharma, A. C. Arpaci-dusseau, and R. H. Arpaci-dusseau, "Consistency Without Ordering," *Proc. of the 10th USENIX Conference on File and Storage Technologies*, pp. 1-16, 2012.

[2] V. Chidambaram, T. S. Pillai, A. C. Arpaci-dusseau, and R. H. Arpaci-dusseau, "Optimistic Crash Consistency," *Proc. of the 24th ACM Symposium on Operating Systems Principles*, pp. 228-243, 2013.

[3] W.-H. Kim, B. Nam, D. Park, and Y. Won, "Resolving Journaling of Journal Anomaly in Android I/O: Multi-Version B-tree with Lazy Split," *Proc. of the 12th USENIX conference on File and Storage Technologies*, pp. 273-285, 2014.

[4] C. Min, W.-H. Kang, T. Kim, S.-W. Lee, and Y. I. Eom, "Lightweight Application-Level Crash Consistency on Transactional Flash Storage," *Proc. of the USENIX Annual Technical Conference*, pp. 221-234, 2015.

[5] D. H. Kang, G. Oh, D. Kim, H. Doh, C. Min, S.-W. Lee, and Y. I. Eom, "When Address Remapping

Techniques Meet Consistency Guarantee Mechanisms," *Proc. of the 10th USENIX Conference on Hot Topics in Storage and File Systems*, pp. 1-5, 2018.
[6] W. Lee et al., "WALDIO: Eliminating the filesystem journaling in resolving the journaling of journal anomaly," *Proc. of the USENIX Annual Technical Conference*, pp. 235-247, 2015.
[7] G. Oh, S. Kim, S.-W. Lee, and B. Moon, "SQLite optimization with phase change memory for mobile applications," *Proc. of 41th ACM Very Large Data Bases*, pp. 1454-1465, 2015.
[8] D. H. Kang and Y. I. Eom, "TO FLUSH or NOT: Zero Padding in the File System with SSD Devices," *Proc. of the 8th ACM Asia-Pacific Workshop on Systems*, pp. 1-9, 2017.
[9] Q. Chen, L. Liang, Y. Xia, H. Chen, and H. Kim, "Mitigating Sync Amplification for Copy-on-write Virtual Disk," *Proc. of the 14th USENIX conference on File and Storage Technologies*, pp. 241-247, 2016.
[10] D. Park and D. Shin, "iJournaling: Fine-Grained Journaling for Improving the Latency of Fsync System Call," *Proc. of the USENIX Annual Technical Conference*, pp. 787-798, 2017.
[11] S. Park, T. Kelly, and K. Shen, "Failure-atomic Msync(): A Simple and Efficient Mechanism for Preserving the Integrity of Durable Data," *Proc. of the 8th ACM European Conference on Computer Systems*, pp. 225-238, 2013.
[12] T. S. Pillai, R. A. L. Lu, V. Chidambaram, A. C. Arpacid-usseau, and R. H. Arpaci-dusseau, "Application Crash Consistency and Performance with CCFS," *Proc. of the 15th USENIX Conference on File and Storage Technologies*, pp. 181-196, 2017.
[13] K. Shen, S. Park, and M. Zhu, "Journaling of Journal Is (Almost) Free," *Proc. of the 12th USENIX conference on File and Storage Technologies*, pp. 287-293, 2014.
[14] Serial ATA, [Online] Available: https://en.wikipedia.org/wiki/Serial_ATA
[15] Filebench (version 1.5), [Online] Available: https://sourceforge.net/projects/filebench
[16] V. T. Asov, E. Zadok, and S. Shepler, "Filebench: A Flexible Framework for File System Benchmarking," *:login:Magazine* 41, pp. 6-12, 2016.
[17] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "SFS: random write considered harmful in solid state drives," *Proc. of the 10th USENIX conference on File and Storage Technologies*, 2012, pp. 139-154.
[18] D. H. Kang, C. Min, and Y. I. Eom, "An Efficient Buffer Replacement Algorithm for NAND Flash Storage Devices," *Proc. of the IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pp. 239-248, 2014.
[19] A. Aghayev, T. Ts'o, G. Gibson, and P. Desnoyers, "Evolving Ext4 for Shingled Disks," *Proc. of the 15th USENIX conference on File and Storage Technologies*, pp. 105-119, 2017.
[20] J. Ziv, and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory*, Vol. 24, No. 5, pp. 530-536.
[21] J. Katcher, "Postmark: A New File System Benchmark," *Technical Report (TR3022)*, 1997.

강 동 현
2000년 한국산업기술대학교 컴퓨터공학과 학사. 2008년 성균관대학교 전자전기컴퓨터공학과 석사. 2018년 성균관대학교 전자전기컴퓨터공학과 박사. 2018년~2019년 삼성전자 책임연구원. 2019년~현재 동국대학교 경주캠퍼스 조교수. 관심분야는 스토리지 시스템, 시스템 소프트웨어, 운영체제

이 상 원
1991년 서울대학교 컴퓨터학과 학사. 1994년 서울대학교 컴퓨터학과 석사. 1999년 서울대학교 컴퓨터학과 박사. 1999년~2001년 한국오라클. 2001년~2002년 이화여대 BK21 계약교수. 2002년~현재 성균관대학교 소프트웨어학과 교수. 관심분야는 플래시 메모리 DBMS

엄 영 익

정보과학회논문지
제 46 권 제 7 호 참조