스토리지 클래스 메모리를 쓰기 버퍼캐시로 활용함에 따른 파일시스템 성능 및 안정성 평가†

(Impact of Exploiting Storage Class Memory as a Write Buffer Cache on the Performance and Reliability of File Systems)

김영진, 도인환, 김은삼, 이동희*, 노삼혁

(Young Jin Kim, In Hwan Doh, Eunsam Kim, Donghee Lee, Sam H. Noh)

홍익대학교, 서울시립대학교*

{yjkim, ihdoh}@necsst.ce.hongik.ac.kr, {eskim, samhnoh}@hongik.ac.kr, dhl_express@uos.ac.kr*

요 약

최근 비휘발성 속성과 램의 속성을 동시에 제공하는 스토리지 클래스 메모리(SCM)의 등장은 시스템 입출력 성능과 안정성 향상을 위한 시스템 소프트웨어 연구에 새로운 형태로의 접근을 가능하게 한다. 이에 본 연구에서는 SCM을 활용하는 쓰기 버퍼캐시에 트랜잭션 단위의 쓰기를 보장함으로써 시스템의 성능과 안정성을 동시에 향상시키고자 한다. 제안된 SCM 쓰기 버퍼캐시 기법은 리눅스 2.6.21에서 저널링 블록 디바이스(JBD)의 트랜잭션 메커니즘을 기반으로 구현되므로 JBD와 같은 수준의 안정성을 보장한다. 또한 실제 시스템에서의 성능 평가 결과는 SCM 쓰기 버퍼캐시를 적용한 EXT3 파일시스템은 비정상 종료 후에도 파일시스템의 일관성을 매번 즉각적(대략 0.2초)으로 복구함을 보여준다. 더불어, 높은 수준의 파일시스템 안정성을 보장함에도 불구하고 최소한의 안정성만을 제공하는 파일시스템보다 더 좋은 입출력 수행 성능을 나타낸다.

1. 서 론

최근 스토리지 클래스 메모리(Storage Class Memory; SCM)의 기술적 성장은 컴퓨팅 시스템의 성능과 안정성 측면에서 획기적인 발전 가능성을 열어준다[1]. 본 연구에서는 입출력 계층 구조에서 SCM을 활용하여 시스템의 안정성과 입출력 성능을 동시에 향상시키고자 한다. 이는 SCM을 파일시스템의 버퍼캐시로 도입하고 해당 버퍼캐시에 트랜잭션 단위의 쓰기를 보장함으로써 가능하다.

지난 수 십 년간, 사용자 데이터에 대한 안정성 보장이라는 이슈는 시스템 소프트웨어 분야에서 꾸준히 연구되고 있는 주제이다. 특히 시스템 붕괴 상황에 대비해서 파일시스템의 일관성을 보장하려는 노력은 다각도로 이루어져 왔으며, 그 결과 다양한 효과적인 기법들이 이미 현실화되었다. 그 예로써, 저널링 파일시스템과 소프트업데이트 기법은 모두 파일시스템의 일관성을 효과적으로 보장한다[2]. 그럼에도 불구하고 기존의 기법들은 시스템의 입출력 성능을 희생하거나 최근에 갱신된 데이터들의 무결성을 희생한다는 측면에서 여전히 개선의 여지가 있다.

비휘발성 램의 한 형태로 분류되는 SCM은 메모리 셀 자체적으로 비휘발성 속성을 제공하며 동시에 고속의 바이트 단위 랜덤 접근을 지원하는 메모리 기술을 총칭한다[1]. 현재 대표적인 SCM으로는 PCM(또는 PRAM), FeRAM, 그리고 MRAM이 주목받고 있다. 특히 PCM은 인텔이나 삼성전자와 같은 주요 반도체 제조사들에 의해서 주도적으로 연구 개발되고 있으며, 최근에 128Mb 칩의 상용화 성공과 그 집적도 그리고 확장성을 고려할 때 가장 유망한 SCM 기술로 평가받고 있다[3].

시스템 입출력 계층구조에서 SCM의 효과적인 활용은 전통적으로 트레이드오프 관계로 여겨지는 입출력 성능 향상과 안정성 향상이라는 두 가지 요구사항을 동시에 만족시키는 것을 가능하게 한다. 이에 본 연구에서는 SCM을 버퍼캐시 용도로 도입함으로써 전통적인 파일시스템에서 주기적으로 수행하는 동기식 쓰기를 제거하여 입출력 수행 성능을 향상하고자 한다. 동시에 SCM 버퍼캐시에 트랜잭션 단위의 쓰기를 지원함으로써 이를 적용한 파일시스템에 대해 기존의 저널링 파일시스템 수준의 높은 안정성을 보장하고자 한다.

트랜잭션 단위의 쓰기를 보장하는 SCM 쓰기 버퍼캐시는 SCM 하드웨어 모듈과 이를 활용하는 소프트웨어 모듈로 실현 가능하다. SCM 하드웨어 모듈은 메인 메모리와 같이 물리 메모리 주소 공간으로 사상되어 CPU에 의해서 바이트 단위 접근이 가능하도록 구성된다. 본 논문에서는 리눅스 운영체제에 동적으로 적재가 가능한 커널모듈의 형태로 SCM을 쓰기 버퍼캐시로써 활용하는 기법을 구현한다. 더불어 해당 기법은 버퍼캐시에 트랜잭션 단위의 쓰기를 보장하며, 기존의 파일시스템 독립적인 인터페이스를 지원함으로써 범용성을 제공한다.

실제 시스템 환경에서의 구현을 통한 성능평가 결과는 SCM을 쓰기 버퍼캐시로 활용할 경우에 파일시스템의 성능과 안정성 향상, 그리고 시스템 붕괴 시 파일시스템 일관성 복구시간을 획기적으로 단축할 수 있음을 보여준다. 구체적으로, SCM 쓰기 버퍼캐시를 적용한 EXT3 파일시스템은 시스템 붕괴 후 파일시스템의 일관성을 매번 즉각적(대략 0.2초)으로 복구한다. 또한, SCM 쓰기 버퍼캐시를 적용한 EXT3 파일시스템은 모든 데이터를 저널링하는 EXT3 파일시스템 수준의 안정성을 제공함과 동시에 EXT2 파일시스템은 물론 메타데이터만 저널링하는 EXT3 파일시스템보다 뛰어난 수행성능을 나타낸다.

이후 논문의 구성은 다음과 같다. 본 연구와 밀접한 관련이 있는 기존 연구들을 다음 절에서 언급하고, 3절에서는 본 논문에서 제안하는 SCM을 활용하는 쓰기 버퍼캐시 기법에 대해서 구체적으로 설명한다. 이어서 4절과 5절에서 실험 환경과 성능 평가 결과에 대해서 각각 논의하고, 6절에서 결론을 맺는다.

†이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임 (No. R0A-2007-000-20071-0).

2. 관련 연구

본 연구는 입출력 계층구조에서 SCM을 쓰기 버퍼캐시 형태로 도입하여 파일시스템의 안정성과 성능을 동시에 향상시키려는 시도이다. 이에 본 절에서는 기존의 대표적인 파일시스템 안정성 향상 기법에 대해서 언급한 후, SCM을 포함하여 비휘발성 램을 입출력 계층구조에서 활용하고자 하는 선행연구들을 정리하고자 한다.

기존에 파일시스템의 일관성을 보장하는 대표적인 방법으로는 저널링 파일시스템 기법과 소프트업데이트 기법을 들 수 있다[2]. 저널링 기법은 파일시스템이 갱신될 때마다 해당 사실을 주기적으로 특정 저널공간에 미리 기록한 후에 갱신된 내용을 실제 파일시스템에 반영함으로써 파일시스템 일관성과 데이터에 대한 높은 안정성을 보장하는 기법이다. 이 같은 방식은 추가적인 기록과정으로 인한 파일시스템의 성능 저하를 동반하며, 저널공간이 늘어남에 따라 시스템 붕괴 이후의 복구시간이 길어진다는 한계를 지닌다.

소프트업데이트 기법은 항상 파일시스템의 일관성이 보장되도록 파일시스템에서 갱신된 정보들이 저장매체에 반영되는 순서를 조정한다. 이 경우 저널링 기법과 달리 추가적인 기록으로 인한 오버헤드는 없지만 쓰기 순서 결정을 위하여 복잡한 자료구조를 메모리상에 유지하여야 하므로 실제로 구현이 힘들다. 또한 비록 파일시스템의 일관성은 보장되지만 시스템 붕괴 시 램에 캐시된 모든 자료가 손실된다는 단점이 있다. 본 연구는 SCM을 도입하여 이들 두 기존 방식들의 단점은 배제하고 장점만을 취하고자 한다는 점에서 새롭다고 할 수 있다.

비휘발성 램을 입출력 계층구조에 도입하려는 시도는 다양한 방식으로 진행되어오고 있다. Baker 등은 네트워크 파일시스템 환경에서 배터리의 지원을 필요로 하는 비휘발성 램을 사용하는 쓰기 캐시가 쓰기와 관련된 사용자 처리량을 크게 개선할 수 있음을 제시한 바 있다[4]. Chen 등은 무정전 전력공급 장치(UPS)의 지원으로 전체 시스템 메모리가 비휘발성 속성을 가질 경우에 파일캐시의 안전한 관리 및 복구 방법을 제안하고 트랜잭션 개념이 도입된 비휘발성 파일캐시의 사용으로 인한 성능과 안정성 측면에서의 이득을 정량화한 바 있다[5]. 본 연구는 SCM을 비휘발성 램으로 고려하며, 휘발성 램 버퍼캐시와 SCM버퍼캐시가 공존하는 환경을 고려하면서 트랜잭션 쓰기를 지원한다는 점에서 기존의 연구들과 차별된다.

최근에 SCM을 입출력 계층구조에서 활용하는 연구가 활발하게 진행되고 있다. Miller 등은 메타데이터 저장소로 MRAM을 활용하는 디스크 기반 파일시스템을 제안한 바 있다[6]. Doh 등은 메타데이터를 SCM에 유지하고 파일데이터만 플래시 메모리에 저장하는 플래시 파일시스템을 제안한 바 있다[7]. Kim 등은 파일시스템과 플래시 변환계층의 메타데이터를 저장하는 용도로 PCM의 사용을 고려한 바 있다[8]. 본 연구는 파일시스템의 메타데이터를 포함하는 전체 데이터에 대한 쓰기 캐시를 고려하고 있다는 점에서 구별된다.

3. TranSCM: 트랜잭션 단위의 쓰기를 지원하는 SCM 쓰기 버퍼캐시 관리 모듈

TranSCM의 설계 목표는 메타데이터와 데이터의 일관성 보장을 통한 파일시스템의 안정성 향상과 극단적인 지연 쓰기를 통한 파일시스템의 성능 향상을 동시에 보장하는 것이다. 본 절에서는 SCM 쓰기 버퍼캐시의 성능과 안정성 측면에서의 효과에 대해서 개괄적으로 살펴본 다음, TranSCM을 구성하는 컴포넌트들과 그 동작방식에 대하여 설명한다.

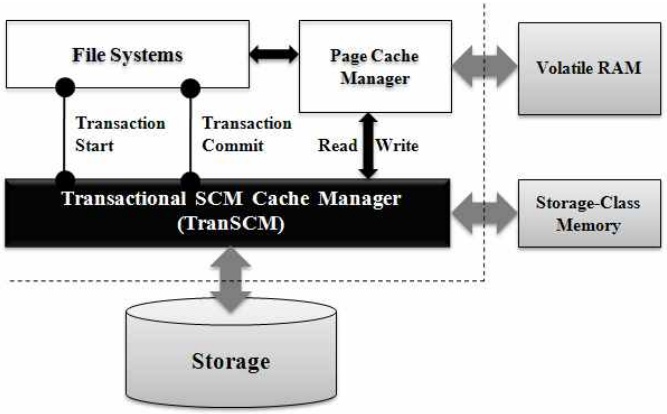


그림 1. TranSCM을 포함하는 시스템 입출력 계층구조

3.1 전체적인 구조

SCM 쓰기 버퍼캐시는 기존의 파일시스템 독립적인 추가적인 소프트웨어 모듈인 TranSCM에 의해서 관리되며, TranSCM은 기존의 파일시스템들이 트랜잭션 단위 쓰기 캐시 기능을 사용할 수 있도록 특정 인터페이스를 제공한다. 그림 1은 TranSCM을 포함하는 입출력 계층구조를 간략하게 보여준다. 논리적으로 TranSCM은 파일시스템과 블록장치 드라이버 사이에 위치하며, 파일시스템에게 트랜잭션의 시작과 종료(commit), 그리고 트랜잭션 단위의 쓰기 요청 연산에 대한 인터페이스를 제공한다.

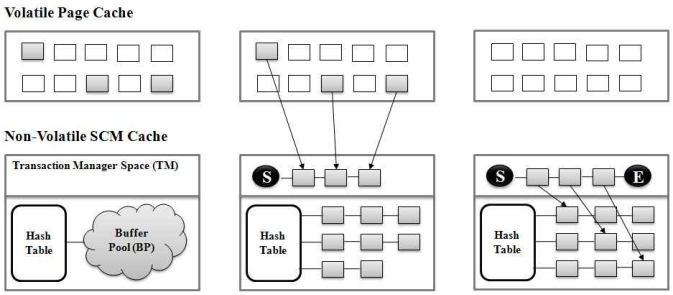
TranSCM이 제공하는 인터페이스를 통해서 기존의 파일시스템들은 트랜잭션 단위로 파일데이터와 메타데이터를 일관성 있게 그리고 안정적으로 갱신하는 것이 가능하다. 트랜잭션 시작 요청이후에 갱신된 파일시스템의 모든 정보들은 SCM 상에서 하나의 트랜잭션으로 구성되어 트랜잭션 종료 요청이 발생하면 SCM 쓰기 버퍼캐시로 즉각 반영된다. 파일시스템의 관점에서 수정되는 모든 데이터들은 부분 쓰기의 문제없이 트랜잭션 단위로 저장매체로 반영되므로 파일시스템의 일관성이 보장되며, 종료되지 않은 마지막 트랜잭션의 갱신된 정보를 제외한 모든 데이터는 안정적으로 유지된다.

TranSCM은 파일시스템의 모든 갱신된 정보들을 비휘발성 SCM에 유지하기 때문에, 전통적으로 수행되어온 휘발성 램 페이지 캐시상의 갱신된 페이지들을 주기적으로 디스크에 기록하는 과정은 불필요하게 된다. 쓰기를 위한 디스크 접근은 오직 SCM 버퍼캐시의 유효 공간을 확보하는 과정에서만 발생한다. 따라서 SCM 버퍼캐시의 크기가 충분히 크거나 효과적으로 유효 공간을 확보하는 것이 가능한 경우에, TranSCM을 적용한 파일시스템은 기존의 안정성을 고려하지 않는 파일시스템보다 더 좋은 성능을 발휘하는 것이 가능하다.

3.2 내부 설계

TranSCM은 SCM 공간을 트랜잭션 처리를 위한 영역(Transaction Manager; TM영역)과 순수 버퍼캐시 영역(Buffer Pool; BP영역)으로 나누어서 관리한다. 그림 2는 TranSCM이 트랜잭션 단위로 갱신된 데이터를 캐시하는 과정에서 SCM 내부의 각 영역에서 수행되는 일련의 동작들을 보여준다.

트랜잭션 단위로 데이터를 캐시하는 과정은 트랜잭션 구성과 SCM 버퍼캐시 갱신의 두 단계로 나누어서 생각할 수 있다. 참고로, TranSCM이 트랜잭션을 구성하고 관리하는 세부 메커니즘은 리눅스의 저널링 블록 디바이스(JBD)의 트랜잭션 메커니즘을 그대로 차용하므로 상세한 설명은 생략하며, SCM 버퍼캐시 갱신을 중심으로 설명한다[9].



(a) Page dirty (b) Transaction copy (c) Cache update

그림 2. 트랜잭션 단위 쓰기 보장을 위한 내부 동작

TranSCM이 적용된 파일시스템은 Create(), Write()와 같은 파일시스템 단위 연산 시에 TranSCM이 제공하는 인터페이스를 포함하므로 TranSCM은 파일시스템에 의하여 수정된 정보를 각 연산 단위로 유지하고 관리할 수 있다. 그림 2(a)는 파일시스템 연산에 의하여 일부 페이지가 수정되었음을 보여준다. TranSCM에 의하여 연산 단위로 관리되는 갱신된 페이지들은 파일시스템의 명시적인 트랜잭션 요청이나 주기적인 트랜잭션 요청에 의해서 하나의 트랜잭션으로 구성된다. 구성된 트랜잭션은 시작(**S**tart)과 끝(**E**nd)을 명시하여 SCM의 TM영역으로 복사된다. 그림 2(b)는 갱신된 페이지가 하나의 트랜잭션으로 구성되어 SCM의 TM영역으로 복사됨을 보여준다. SCM 버퍼캐시의 갱신은 하나의 트랜잭션이 완벽하게 SCM의 TM영역으로 복사되었을 때만 일어난다. 이는 시스템 붕괴 시 버퍼캐시의 부분 갱신 문제를 해결하기 위함이다. 그림 2(c)는 하나의 완전한 트랜잭션이 버퍼캐시로 반영되는 과정을 나타낸다.

비휘발성 SCM 버퍼캐시를 복구하는 절차, 다시 말하면 TranSCM이 적용된 파일시스템의 복구 절차는 간단하게 이루어진다. SCM 버퍼캐시의 갱신은 그림 2(c)에서 볼 수 있듯이 하나의 새로운 트랜잭션이 시작과 끝을 명시하여 SCM 공간으로 완전히 복사 되었을 때만 수행된다. 시스템 붕괴가 발생 했을 때 SCM의 새로운 트랜잭션은 시작과 끝이 명시된 완전한 상태 혹은 끝이 명시 되지 않은 불완전한 상태에 있을 수 있다. 만약 트랜잭션이 완전한 상태에 있다면 BP영역에 갱신 문제가 있을 수 있으므로 해당 트랜잭션에 속한 모든 버퍼를 BP영역으로 반영하는 것으로 복구가 완료되며, 트랜잭션이 불완전한 상태라면 단순히 트랜잭션을 폐기하는 것으로 복구된다.

4. 실험 환경

본 절에서는 TranSCM이 적용된 파일시스템의 수행성능과 안정성 평가를 위하여 본 연구에서 구현한 소프트웨어들과 실험이 수행된 하드웨어 시스템 환경에 대해서 살펴본다.

4.1 시스템 소프트웨어 환경

기본적으로 리눅스 2.6.21 커널에서 TranSCM를 커널 모듈 형태로 구현하고, 대표적인 저널링 파일시스템인 EXT3 파일시스템이 TranSCM를 운용하도록 수정한다. 이후 논의의 편의를 위해서 EXT3-TranSCM이라 칭한다. TranSCM은 SCM 관리자, SCM 버퍼캐시 관리자 그리고 트랜잭션 캐시 API로 이루어진 3개의 상호 연동하는 모듈로 구성되며, 트랜잭션을 처리하는 부분은 JBD를 기반으로 구현되어 있다. SCM 관리자는 SCM의 할당과 해제를 담당하며 시스템 메모리인 휘발성 램의 일부를 SCM 영역으로 에뮬레이션 하는 것이 가능하다. SCM 버퍼캐시 관리자는 갱신된 버퍼들을 LRU 형태로 관리하며 SCM 버퍼캐시가 갱신된 버퍼로 가득 찬 경우 전체 갱신된 버퍼의 30%를 디스크로 기록해 유효 공간을 확보한다. 트랜잭션 캐시 API모듈은 파일시스템의 갱신된 버퍼들을 트랜잭션으로 모아 SCM 버퍼캐시 모듈로 전달하는 역학을 담당한다.

성능 평가 대상으로는 EXT3 파일시스템과 EXT3 파일시스템의 저널링 공간으로 SCM을 사용하도록 JBD를 수정한 파일시스템(이후, EXT3-JbdSCM이라 명명)을 채택한다. 성능평가에 사용된 벤치마크는 PostMark 버전 1.51이며, 생성된 워크로드에서 쓰기 요청된 총 용량은 3022MB이다[10].

4.2 하드웨어 환경

TranSCM이 적용된 파일시스템의 평가를 위해서 임베디드 시스템 환경과 PC 환경을 함께 사용한다. 실험에 사용된 임베디드 시스템은 실제 SCM의 한 형태인 FeRAM을 탑재하고 있음에도 불구하고 임베디드 시스템이 가지는 열악한 하드웨어 수준을 고려할 때 파일시스템의 수행성능을 충분히 평가하는데 한계가 있다. 이에 추가적인 실험 환경으로 SCM 영역을 메인 메모리의 일부로 에뮬레이션한 PC 환경을 사용한다.

임베디드 시스템은 PXA255 ARM RISC 400MHz MCU, 64MB SDRAM, 64MB NAND Flash를 갖추고 있다. 또한 최대 64MB의 FeRAM이 가용하며 USB 1.0 인터페이스를 통하여 디스크를 부착할 수 있다. 실험 PC는 Intel Core2 Duo E4400 2.00GHz CPU, 2GB RAM, Seagate Barracuda 7200.10 400GB 디스크를 갖추고 있다. PC환경에서 진행된 모든 실험에서 2GB의 시스템 메모리 중 768MB가 SCM으로 에뮬레이션 되므로 PC환경에서의 시스템 메모리는 1280MB로 볼 수 있다.

5. 성능 평가 결과

본 절에서는 TranSCM이 적용된 파일시스템의 수행성능과 복구시간에 초점을 맞춘다. TranSCM은 리눅스의 JBD와 동일한 트랜잭션 메커니즘을 차용하여 JBD가 제공하는 수준의 안정성을 제공하므로 안정성에 대한 논의는 생략한다.

5.1 파일시스템 복구

실험을 위해 임베디드 시스템 환경에서 벤치마크 수행도중 임의의 시점에서 전원을 차단하여 시스템을 붕괴시킨 후 파일시스템 복구과정을 수행한다. EXT3 파일시스템의 디스크 저널 크기, EXT3-JbdSCM의 SCM 저널 크기, EXT3-TranSCM에 사용된 SCM 버퍼캐시의 크기는 모두 64MB로 동일하다.

표 1은 시스템 붕괴 이후 각 파일시스템의 복구시간을 보여준다. 비정상적인 종료 이후 파일시스템 마운트의 경우 EXT3은 디스크의 저널 공간전체를 탐색하고 필요한 트랜잭션을 메인 파일시스템에 기록하여야 하므로 50초 내외로 가장 느리게 복구된다. EXT3-JbdSCM의 경우 SCM 저널 공간을 탐색하고 필요한 트랜잭션을 메인 파일시스템으로 기록하는데, 메인 파일시스템으로 기록해야 하는 트랜잭션의 양에 따라서 0.2초에서 38초 내외로 복구 시간에 많은 차이를 보여준다. EXT3-TranSCM의 경우 SCM상의 마지막 트랜잭션을 분석하고 버퍼캐시로 반영 또는 미반영 하는 것으로 복구 과정이 완료되므로 매번 0.2초로 즉각적인 복구가 가능하다.

표 1. 시스템 붕괴 이후 복구시간

File Systems	Recovery time(sec)
EXT3	49.897
EXT3-JbdSCM	0.248 ~ 38.697
EXT3-TranSCM	**0.217**

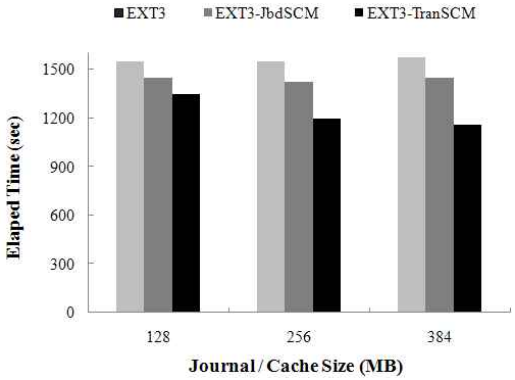


그림 3 파일시스템 성능 비교

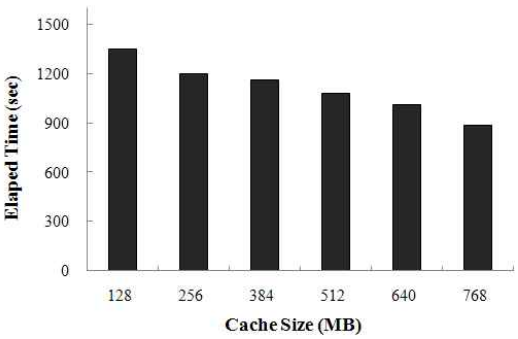


그림 4. 캐시에 대한 EXT3-TranSCM의 성능

5.2 수행 성능

그림 3은 PC 실험 환경에서 디스크 저널 크기, SCM 저널 크기 및 SCM 버퍼캐시의 크기 변화에 따른 각 파일시스템의 벤치마크 수행성능을 보여준다. 실험에서 EXT3은 JBD가 지원하는 세 가지 저널링 모드 중 수행속도가 가장 빠르지만 메타데이터의 일관성만 보장할 수 있는 Writeback 모드로 동작하고, EXT3-JbdSCM과 EXT3-TranSCM는 메타데이터의 일관성과 데이터의 안정성까지 모두 보장할 수 있는 Full Journal 모드에서 동작 한다.

그림 3에서 EXT3-JbdSCM과 EXT3-TranSCM는 안정성 보장을 위하여 Writeback 모드로 동작하는 EXT3보다 더 많은 양의 데이터를 기록함에도 불구하고 모든 설정에서 EXT3보다 좋은 수행성능을 보여주고 있다. EXT3과 EXT3-JbdSCM의 성능 차이는 저널 공간으로 사용되는 매체가 디스크에서 SCM으로 교체됨에 따른 매체의 접근 속도 차이에서 기인된다. EXT3-JbdSCM은 EXT3에 비하여 전반적으로 좋은 성능을 보여주지만 EXT3과 마찬가지로 저널 공간 크기의 증가로 인한 성능 변화는 미미함을 확인할 수 있다.

EXT3-TranSCM와 EXT3-JbdSCM과의 성능 차이는 시스템 구조적 측면의 차이에서 기인되며 EXT3-TranSCM는 사용되는 SCM의 크기에 비례하여 성능이 증가하는 것을 확인할 수 있다. 그림 4는 위의 실험 설정에서 SCM 버퍼캐시 크기 증가에 따른 EXT3-TranSCM의 수행 성능을 보여준다. 그림 3의 결과와 마찬가지로 SCM 캐시 사이즈에 비례하여 성능이 좋아지고 있는 것을 확인할 수 있으며, 이는 단순히 빠른 저널링 매체를 사용함으로써 얻을 수 있는 성능 이득의 한계를 보여줌과 동시에 본 연구에서 제안하는 EXT3-TranSCM의 성능상의 효과를 잘 나타내고 있다.

6. 결 론

본 연구에서는 입출력 계층구조에 SCM을 파일시스템의 버퍼캐시로 도입하고 해당 버퍼캐시에 트랜잭션 단위의 쓰기를 지원하는 기법을 제안하였다. 구현을 통한 실제 시스템 환경에서의 성능 평가 결과는 제안된 SCM 쓰기 버퍼캐시의 활용을 통해 파일시스템 안정성을 희생하지 않으면서 입출력 성능을 향상하는 것이 가능함을 보여준다.

본 연구가 비록 의미 있는 실험 결과를 제시함으로써 SCM의 효과적인 활용을 통한 시스템의 획기적인 발전 가능성을 보여주고 있지만 아직까지 해결되어져야 할 다수의 문제점을 포함하고 있는 초기 연구임에는 틀림없다. 이에 향후 연구과제로 다음과 같은 사항들을 남겨둔다.

먼저, 다양한 형태의 파일시스템 워크로드를 사용하여 광범위한 실험을 진행하고자 한다. 이를 통해서 SCM 쓰기 버퍼캐시를 적용한 파일시스템이 수행 성능 측면에서 이득을 얻게 되는 원인들을 분석하고, 분석된 자료를 바탕으로 다시 SCM 쓰기 버퍼캐시를 개선하고자 한다. 둘째, 현재 SCM을 적용한 파일시스템으로써 EXT3 파일시스템만을 고려하고 있지만 다른 파일시스템으로 확장하지 않을 이유는 전혀 없다. 더 나아가 데이터베이스 시스템에서의 SCM 적용에 대해서도 살펴보고자 한다. 셋째, 전통적으로 쓰기 캐시의 성능에 결정적인 영향을 끼치는 요소로써 캐시의 유효 공간을 확보하고자 저장매체에 갱신된 페이지들을 쓰는 정책(디스테이징 정책)이 중요하게 여겨진다. 이에 트랜잭션 개념을 저해하지 않는 범위 내에서의 효과적인 디스테이징 정책에 대한 연구도 진행될 것이다.

참고 문헌

[1] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, “Overview of candidate device technologies for storage-class memory,” IBM Journal of Research and Development, vol. 52, no. 4/5, 2008.

[2] M. I. Seltzer, G. R. Ganger, M. K. McKusick, K. A. Smith, C. A. N. Soules, and C. A. Stein, “Journaling versus Soft Updates: Asynchronous Meta-data Protection in File Systems,” In Proceedings of the 2000 USENIX Annual Technical Conference (USENIX’00), 2000.

[3] R. Freitas, W. Wilcke, B. Kurdi, G. Burr, “Storage Class Memories,” Tutorial at the 7th USENIX Conference on File and Storage Technologies (FAST’09), 2009.

[4] M. Baker, S. Asami, E. Deprit, J. Ouseterhout, and M. Seltzer, “Non-Volatile Memory for Fast, Reliable File Systems,” In Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), 1992.

[5] P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell, “The Rio File Cache: Surviving Operating System Crashes,” In Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS-VII), 1996.

[6] E. L. Miller, S. A. Brandt, and D. D. E. Long, “HeRMES: High-Performance Reliable MRAM-Enabled Storage,” In Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS-VIII), 2001.

[7] I. H. Doh, J. Choi, D. Lee, and S. H. Noh. “Exploiting Non-Volatile RAM to Enhance Flash File System Performance,” In Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT’07), 2007.

[8] J. K. Kim, H. G. Lee, S. Choi, and K. I. Bahng. “A PRAM and NAND Flash Hybrid Architecture for High-Performance Embedded Storage Subsystems,” In Proceedings of the 7th ACM International Conference on Embedded Software (EMSOFT’08), 2008.

[9] K. Sovani, “Linux: The Journaling Block Device,” Kernel Trap (http://kerneltrap.org/node/6741), 2006.

[10] J. Katcher, “PostMark: a New Filesystem Benchmark,” Network Appliance, Tech. Rep. TR3022, 1997.