플래시메모리 저장장치를 위한 데이터베이스 기술

플래시메모리 저장 장치는 성능 특성, 쓰기 방식, 디바이스 내 병렬성 등 여러 가지 측면에서 기존 하드디스크와 많은 차이점을 갖고 있다. 한편, 데이터베이스 엔진은 성능, 데이터 일관성 등 관점에서 하드디스크의 여러 가지 IO 특성들을 가정한 수많은 최적화 기술을 담고 있다. 따라서 플래시메모리를 위한 차세대 데이터베이스 엔진 관점에서는 두 저장장치의 본질적인 차이점에 대한 이해를 바탕으로, 플래시메모리 저장장치의 특성을 활용하는 다양한 최적화의 기회가 존재한다. 본 고에서는 하드디스크와 다른 플래시메모리 저장장치의 특성을 활용한 최신 데이터베이스 기술들을 소개하고, 향후 데이터베이스 기술 및 데이터베이스를 위한 플래시메모리 저장장치의 발전 방향을 간단히 논한다.

이 상 원
성균관대학교

Ⅰ. 서 론

지난 10년 사이에 이루어진 플래시메모리 관련 기술의 획기적인 발전에 따라, 성능, 가격, 전력소모 등 저장 장치 수요자의 관점에서 플래시메모리 기반 저장 장치는 하드디스크와 경쟁 또는 대체할 수 있는 위치로 자리 잡아 가고 있다. 스마트폰으로 대변되는 모바일 디바이스 분야는 말할 것도 없고, 랩탑, PC 등 개인용 컴퓨터와 최근에 구글, 페이스북, 아마존, 애플 등 인터넷 규모의 엔터프라이즈 데이터센터에서도 모든 컴퓨팅노드에 플래시메모리 저장장치를 채택하는 움직임 (all-flash data center)이 점점 더 가속화하고 있다.

한편, 빅 데이터 시대에 플래시메모리에 저장되는 데이터는 대부분 전통적인 데이터베이스 기술인 관계형 DBMS (예: Oracle, MySQL 등) 또는 그 변형인 NoSQL용 Key-Value 저장장치 (예: 카산드라, 몽고DB, Couchbase, Facebook 등)에 의해 유지 관리된다.

그런데, 이들 데이터베이스 관리 소프트웨어와 운영체제, 파일시스템에서 채택하는 많은 내부 기술들은 (단, 최근의 all-flash 패러다임에 따라 플래시메모리 저장장치를 염두에 두고 개발된 페이스북 RocksDB, Aerospike 등의 Key-Value 저장장치는 논외로 한다.) 하드디스크의 특성을 가정하고 개발되었다. 따라서 이러한 전통적인 데이터관리 소프트웨어를 플래시메모리 저장장치 위에서 동작시킬 때 최적의 성능을 얻어내지 못할 수 있다.

이러한 인식 하에서 최근에 플래시메모리 저장장치를 위한 소프트웨어 최적화 기술들이 운영체제, 파일시스템, 데이터베이스 분야를 중심으로 활발히 개발되고 있다. 특히, 데이터베이스 기술은 대용량 데이터, 다양한 데이터 접근 IO 패턴, 데이터의 일관성 보장을 위한 여러 가지 요구 사항들로 인해서 최적화의 기회와 효과가 아주 큰 분야이다.

본 논문에서는 데이터베이스 기술 분야를 중심으로 최근에 개발된 플래시메모리 저장장치 기반의 최적화 기술들을 소개하고, 또한 향후 이 분야의 발전 방향에 대해 간단히 논의한다. 우선 Ⅱ장에서는 데이터베이스 기술에 영향을 미칠 수 있는 플래시메모리 저장장치의 주요한 특성을 하드디스크와 비교해서 논의하고, Ⅲ장에서는 Ⅱ장에서 설명한 주요한 특성을 활용하는 측면에서 본 저자를 포함한 연구진이이 최근 연구 결과의 주요 아이디어를 설명하고, 마지막으로 Ⅳ장에서 이 분야의 향후 연구 방향에 대해 간단히 언급한다.

Ⅱ. 플래시메모리 저장장치 특성

본 장에서는 플래시메모리의 일반적인 특성들과 데이터베이스의 새로운 기술개발을 위한 연구기회에 대해 간단히 설명한다. 본 장에서 설명하는 플래시메모리 저장장치의 일반적인 특성은, eMMC나 PC향, 데이터 센터향, 그리고 엔터프라이즈향 SSD 등 각 제품별로 세부적인 사항들은 다를 수 있으나, 하드디스크에 비해 일반적인 기준으로 차이점을 중심으로 제시하였다. 〈표 1〉은 본 논문에서 소개할 데이터베이스 기술들에 영향을 미치는 플래시메모리 저장장치 특성들을 하드디스크와 간단히 비교하고, 이 특성을 활용한 최근 기술들과 그 참고문헌을 제시하고 있다.

〈표 1〉 플래시메모리 저장장치 특성 및 최근 연구결과

구분	하드디스크 저장장치	플래시메모리 저장장치	최근연구결과
쓰기 방식	덮어쓰기	CoW (Copy-on-Write)	X-FTL[1]
순차 쓰기 vs. 임의 쓰기 성능 차이	순차쓰기 ⟩⟩ 임의쓰기	순차쓰기 ⟩⟩ 임의쓰기	FaCE[2]
내부 병렬성	1	아주 큼 (예: ⟩= 128)	PSync[3] DuraSSD[4]
쓰기버퍼	존재	존재	DuraSSD[4]
순차 읽기 vs. 임의 읽기 성능 차이	순차 읽기 ⟩⟩ 임의읽기	순차읽기 ~ 임의읽기 (70~80% 수준)	인덱스 기반 질의 처리[5]

플래시메모리 저장장치를 위한 소프트웨어 최적화 기술들이 운영체제, 파일시스템, 데이터베이스 분야를 중심으로 활발히 개발 되고 있다.

우선, 플래시메모리 저장장치 내부에서 택하고 있는 쓰기 방식이다. 플래시메모리 칩은 덮어쓰기를 허용하기 않기 때문에, FTL (Flash Translation Layer)라는 소프트웨어 모듈에서는 특정 논리적 데이터페이지 P (예를 들어, 논리 주소 LBA1) 에 대한 쓰기 요청이 올 경우, 플래시메모리 원본 페이지 (예를 들어, 물리주소 PBA1)를 덮어쓰지 않고, 새로운 위치 (예를 들어, 물리주소 PBA2)에 쓰는 Copy-on-Write (CoW) 방식이 보편화되어 있다. 반면, 하드디스크는 데이터에 대한 덮어쓰기를 허용한다.

플래시메모리 저장장치의 두 번째 주요한 특성은, 쓰기연산의 경우, 순차 쓰기와 임의 쓰기의 성능 차이가 아주 크다는 점이다. 최근 FTL 기술 및 충분한 리소스를 활용함으로써 임의 쓰기 연산의 속도가 이전에 비해 상대적으로 많이 빨라졌지만, 지금도 현실적으로 순차 쓰기 처리량에 비해, 임의 쓰기 처리량은 적게는 수 배 많게는 수 십 배의 차이가 난다. 특히, 데이터베이스처럼, 상대적으로 적은 페이지 (예: 8KB)에 대한 임의쓰

기가 빈번한 경우에는 더욱 더 그러하다.

다음으로, 플래시메모리 저장장치에서 택하고 있는 병렬성이다. 하드디스크의 경우, 단일 저장장치 안에 하나의 기계적인 디스크 헤드만 존재하는 반면, 플래시메모리 저장장치는 채널, 웨이, 칩 레벨에서 데이터에 대한 병렬적인 접근을 위한 높은 병렬성을 제공하고 있다. 예를 들어, 최근에 출시되는 SSD들은 페이지 기준으로 128 또는 그 이상의 페이지들을 병렬적으로 읽기 접근하는 것이 가능하다.

네 번째 주요한 특성은, 데이터센터향 또는 엔터프라이즈향 플래시메모리 SSD들은 DRAM 캐시를 배터리 백업함으로써 전원이 나가는 경우에도 DRAM 캐시에 쓰여진 데이터의 지속성을 보장한다[4]. 특히, 최근에는 Tantalum Capacitor와 SSD 펌웨어 기술의 발달로, 저비용의 배터리백업 기반의 쓰기 캐시에 저장된 데이터의 안정성을 보장한다. 반면, 아주 고가의 엔터프라이즈향 스토리지 박스 솔루션을 제외하고는, 배터리백업 방식으로 하드디스크의 쓰기 캐시에 저장된 데이터의 안정성을 보장하는 방식은 사용되지 않고 있다. 이 배터리백업된 쓰기 캐시 기능은 데이터의 안정성에 대한 요구사항이 아주 높은 데이터베이스 애플리케이션의 경우, 성능에 미치는 효과가 아주 크다. 특히, 앞서 언급한 플래시메모리 SSD의 병렬성을 쓰기 연산입장에서 100% 활용하는 면에서도 아주 중요한 특성이다.

마지막 특성은 (이외에도 여러 가지 차이점이 존재하나, 다음 장에서 설명할 최근 데이터베이스 분야 연구에서 활용한 특성 측면에서 마지막 특성), 임의 읽기의 처리량이 순차 읽기 처리량에 아주 근접한다는 점이다. 하드디스크의 경우, 기계적인 디스크 헤드의 움직임이 데이터 접근 속도를 결정하기 때문에, 임의 읽기 속도가 순차 읽기에 비해 아주 느릴 수 밖에 없다. 반면, 플래시메모리 SSD의 경우, 전자적으로 동작하는 특성상 FTL에서 주소 매핑 검색 및 임의읽기 요청 처리를 위한 OS와 커널 모듈의 소프트웨어 부담을 제외하고 큰 속도차이가 없게 된다. 〈표 2〉는 시중에서 널리 유통되는 하디디스크와 SSD를 기준으로 측정한, 순차읽기와 임의읽기의 속도와 순차읽기 대비 임의읽기의 처리량의 상대적인 값을 제시하고 있다. 〈표 2〉에서 알수 있듯이, 순차읽기 대비 임의읽기의 처리량이 하드디스크에서는 아주 낮은 데 반해, SSD의 경우, 거의 근접함을 알 수 있다.

〈표 2〉 저장장치 순차 읽기 vs. 임의읽기 처리속도 비교

구분	하드디스크	플래시 SSD
응답속도 (4KB 페이지)	3.5ms	0.2ms
IOPS (4KB 페이지)	600	35,000
순차읽기 처리량	125MB / sec	200MB / sec
순차읽기 대비 임의읽기 처리량 비율	0.02	0.7

Ⅲ. 플래시메모리 기반 최근 데이터베이스 기술

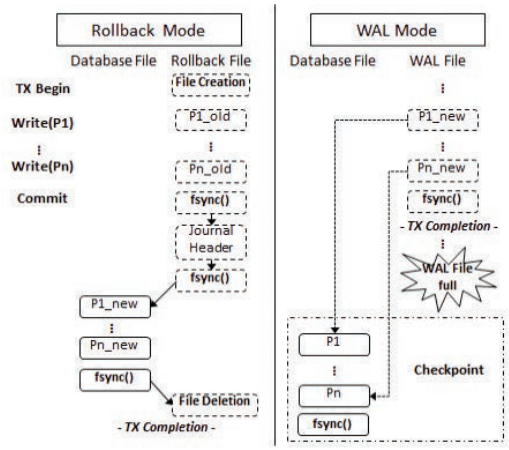
본 장에서는 Ⅱ장에서 설명한 플래시메모리 저장장치의 주요 특성을 활용해서 데이터베이스 성능을 향상시키는 최근 연구결과들을 간단히 소개한다. 지면 제약 등의 이유로 주로 본 저자가 수행한 최근의 연구 결과 중심으로 기술하는 점에 대한 이해를 구한다.

1. X-FTL (Transactional FTL)

데이터베이스에서는 데이터의 일관성이 중요하기 때문에, 하나 또는 복수개의 데이터 페이지를 갱신하는 데이터베이스 트랜잭션의 종료 시 (즉, commit) 해당 페이지들의 내용 전체가 저장장치에 성공적으로 다 쓰이던지 또는 변경 이전의 상태를 보장하는 원자성 (atomicity, 즉, all-or-nothing)이 중요한 요구사항이다. 한편, 덮어쓰기가 허용되는 하드디스크와 최근의 플래시메모리 저장장치의 경우도, 단일 페이지에 대한 쓰기 연산의 원자성을 보장하지 못할뿐더러 다중 페이지에 대한 쓰기 연산의 원자성을 보장하지 못한다.

데이터베이스에서는 관리하는 데이터의 일관성이 중요하기 때문에, 원자성 (atomicity 즉, all-or-nothing) 이 중요한 요구사항이다.

이러한 문제를 해결하기 위해, 데이터베이스 분야에



〈그림 1〉 SQLite 쓰기 연산 원자성 보장을 저널 모드[1]

서는 덮어쓰기를 허용하는 하드디스크를 가정하고, 중복쓰기 (redundant write)에 의한 쓰기 연산의 원자성을 보장하는 방법을 널리 채택하고 있다. 예를 들어, 모바일 디바이스의 실질적인 표준 임베디드 DBMS인 SQLite, 오픈소스 DBMS인 MySQL이나 Posgtres, Sybase SQL Anywhere 같은 제품들이 중복쓰기에 기반한 데이터 페이지 쓰기 연산의 원자성을 소프트웨어적으로 보장하고 있다.

구체적인 예로써, SQLite의 경우, 〈그림 1〉에서처럼, 하나의 트랜잭션에서 페이지들을 쓸 때, 수정되는 페이지들의 원본 페이지들의 복사본을 롤백화일에 보관하고, 해당 페이지들을 덮어쓰는 방식 (이를 롤백 모드라 부름.) 과 새로운 위치에 새로운 페이지들을 성공적으로 다 쓴 후에 원본을 덮어쓰는 방식 (이를 WAL 모드라 부름)을 택하고 있다. 이러한 방식을 통해서 하나 또는 그 이상의 페이지들의 쓰기 연산의 원자성을 보장할 수 있다. 이러한 방식은 하나의 논리적 페이지에 대해 저장장치에는 중복적으로 두 번의 쓰기 연산을 유발하고, 또한 롤백 모드의 경우, 파일 생성, 삭제나 fsync()와 같은 시스템 콜을 빈번하게 호출하게 되므로 성능 저하가 아주 크다.

한편, 플래시메모리는 덮어쓰기를 허용하지 않기 때문에, FTL에서는 페이지 쓰기에 대해 기본적으로 Copy-On-Write 방식을 취함으로써 덮어 쓰이는 페이지의 원본 카피를 보존하고, 새로운 카피는 새로운 위치에 쓰인다. 이러한 특성은 기존 SQLite처럼 저장장치에 중복적인 쓰기를 하지 않고서도, 하나 또는 복수개의 페이지들의 쓰기 원자성을 보장할 수 있는 기회를 제공한다. 이러한 기회 요인을 활용하기 위해 제안된 기법이 X-FTL[1] (Transactional FTL)이다. X-FTL을 지원하는 저장장치에서는 BeginTx, Write(Pi, Tid), Read (Pi, Tid), Commit(Tid) / Rollback(Tid)의 트랜잭션 개념을 갖는 새로운 인터페이스를 제공한다.

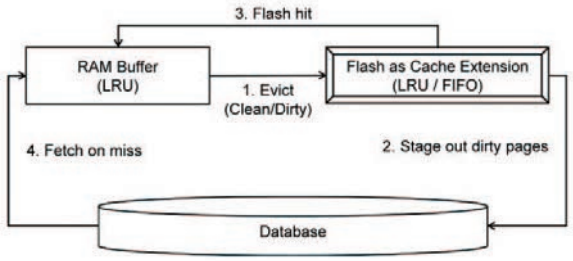
특정 트랜잭션에서 X-FTL 기반으로 N개의 페이지의 쓰기 연산의 원자성 보장을 위해, 우선 BeginTx 명령을 이용해서 새로운 트랜잭션의 시작을 알리고, 그 이후 해당 트랜잭션에서 새로운 페이지를 쓸 때 트랜잭션의 ID, 즉 tid를 인자로 저장장치에 알려주고, 트랜잭션을 종료할 때 Commit(Tid)를 수행하면 Tid에 의해 쓰여진 모든 페이지들의 LBA가 해당 페이지의 새로운 PBA로 매핑을 한꺼번에 바꾸어 줌으로써 모든 페이지를 성공적으로 저장장치에 쓸 수 있게 된다. Commit 성공 이전에 임의의 시점에 발생하는 system crash나 rollback시에는 새로운 페이지 카피본은 무시하고, 이전의 카피에 대한 매핑을 유지함으로써 nothing의 상태로 쉽게 되돌릴 수 있다.

데이터베이스 응용에서 빈번히 접근되는 데이터 페이지들에 대한 시간적 지역성을 고려할 경우, SSD를 DRAM과 하드디스크의 cache로 활용하는 것은 자연스러운 선택이다.

이처럼 X-FTL에서는 플래시메모리 저장장치의 특성을 이용해서 중복쓰기 없이 쓰기 연산의 원자성을 보장함으로써, 다양한 워크로드 대상으로 SQLite 성능을 최소 2배에서 5배 정도 향상시킴을 확인하였다[1].

2. FaCE (Flash as Cache Extension)

플래시메모리 SSD가 하드디스크에 비해서 단위 용량당 가격은 훨씬 비싸지만, IOPS 관점에서 훨씬 더 빠르기 때문에, 데이터베이스 응용에서 빈번히 접근되는 데이터 페이지들에 대한 시간적 지역성 (temporal locality)를 고려할 경우, DRAM과 하드디스크 사이의



〈그림 2〉 FaCE: FlashSSD as Cache Extension[2]

캐시로써 활용하는 것은 아주 자연스러운 선택이다. 〈그림 2〉에서 보여지는 것처럼, 하드디스크에서 읽힌 어떤 페이지가 DRAM 버퍼에서 교체될 때, 이 페이지를 캐시로 사용되는 플래시메모리 SSD에 저장하고, 이후에 이 페이지에 대한 접근이 다시 발생할 때, 느린 하드디스크 대신 SSD에서 읽기 요청을 처리함으로써 시스템 전체의 IO 관련 성능을 획기적으로 높일 수 있다.

높은 병렬성을 갖는 플래시메모리 저장장치의 경우, 복수개의 페이지들에 대한 읽기 연산을 한꺼번에 요청함으로써 성능을 크게 개선시킬 수 있다.

이와 같이 플래시메모리 SSD를 캐시로 사용하는 경우, 가장 중요한 성능지표는 동일한 SSD 공간을 사용했을 때, SSD에서의 hit ratio와 쓰기 연산의 처리량이다. 그리고, 이를 결정하는 주요한 요인은 플래시메모리 SSD 캐시를 버퍼 관리 정책이다. 그런데, 기존의 버퍼 관리 정책은 버퍼 공간을 효율적으로 사용하고 버퍼적중률(hit ratio)을 높이기 위한 LRU (Least Recently Used) 교체 정책의 변형을 주로 사용하였다. 하지만, 이 방법은 SSD에 임의쓰기 패턴을 유발하게 된다.

한편, Ⅱ장 〈표 1〉에서 설명한 바와 같이, 플래시메모리 저장장치의 경우 임의 쓰기에 비해 순차 쓰기가 훨씬 빠르다. 이러한 특성을 활용하기 위해, SSD 캐시에 대한 쓰기 패턴을 임의쓰기 대신 순차쓰기로 바꿀 수 있으면 성능 개선이 가능하다. 이를 위해 SSD 캐시 관리 정책을 기존 LRU에서 단순 FIFO (First-In-First-Out) 방식으로 변경할 수 있다.

이러한 아이디어에 의해 제안된 플래시메모리 SSD 캐시 관리 기법이 FaCE (Flash as Cache Extension) 기법[2] 이다. FaCE에서는, DRAM 버퍼에서 특정 페이지가 교체될 때, 플래시메모리 SSD 캐시에 항상 순차쓰기 패턴이 유발되게 한다. 이를 위해, 캐시를 원형 버퍼 형태로 유지하면서, 가장 오래전에 반입된 페이지를 순차적으로 덮어쓰고 해당 페이지가 하드디스크에서 읽혀진 이후에 수정된 경우 (즉, dirty page), 그 페이지를 하드디스크로 쓴다. FIFO 방식을 유지하면서, 순차쓰기를 하기 때문에, 쓰여지는 페이지의 이전 카피가 SSD 캐시 내에 존재하는 경우에 덮어쓰지 않고 새 카피를 추가하는 방식이다. 따라서 제한된 SSD 캐시 공간의 활용도는 조금 떨어지고 따라서 SSD 캐시 적중률도 낮게 된다.

이러한 쓰기 처리양의 효율성과 SSD 캐시 적중률의 trade-off를 고려할 경우, 데이터베이스의 대표적인 워크로드인 TPC-C의 경우, 캐시 적중률을 조금 손해보더라도 임의쓰기 대신 순차쓰기 방식으로 SSD 캐시를 관리하는 FaCE 방식이 기존 방식에 비해 최대 3배 이상의 성능 향상을 보였다.

3. PSync (트리 병렬 탐색)

데이터베이스나 NoSQL Key-Value 저장장치에서는 주어진 키 값에 대응하는 데이터 레코드나 문서에 대한 접근을 B+트리와 같은 인덱스를 이용해서 접근하는 경우가 아주 빈번하다. 한편, 주어진 키 값에 대응하는 데이터 레코드가 두 개 이상이고 이들 레코드가 클러스터링되어 있지 않고 서로 다른 위치에 저장된 경우도 아주 일반적이다.

이러한 경우에 기존의 데이터베이스에서는 한번에 하나의 페이지를 순차적으로 읽기 연산을 통해서 접근하게 된다. 하드디스크의 경우, Ⅱ장 〈표 1〉에서 설명한 바와 같이, 저장장치 내에서 병렬성이 제한되어 있기 때문에, 이와 같이 순차적으로 데이터 페이지를 하나씩 접근하는 방식을 택해도 해당 질의 입장이나 시스템 전체 처리량 관점에서 큰 차이가 없다.

하지만, 아주 높은 병렬성을 가진 플래시메모리 저장

장치의 경우, 인덱스를 통한 데이터페이지 접근 시 이미 조건을 만족하는 페이지들의 주소를 미리 알 수 있다. 따라서 이 경우, 복 수개의 페이지들에 대한 읽기 연산을 한꺼번에 요청함으로써 플래시메모리 저장장치 내의 병렬성을 한꺼번에 구동함으로써 성능을 크게 개선할 수 있다. 예를 들어, 128개의 서로 다른 페이지를 한 번에 하나씩 읽기 요청하면, 128 * 1페이지 접근 시간 만큼의 시간이 소요되는 반면, 128의 병렬성을 가진 플래시메모리 저장장치에 128개 페이지가 균등하게 흩어져 있는 경우, 1페이지 접근 시간 동안 128개의 페이지를 읽어낼 수 있다.

PSync[3] 기법은 이와 같은 동기에서 개발된, 플래시 메모리 SSD 내의 병렬성을 극대화하는 트리 구조의 데이터 탐색 기법으로 단일 질의 입장에서 응답시간은 획기적으로 개선할 수 있다. 한편, 동시에 수행되는 데이터베이스 질의 수가 많아지는 경우에 PSync의 상대적인 효용성은 낮아질 수 있는데 (즉, 별도로 기존 방식처럼 데이터페이지를 하나씩 접근하는 질의가 수백개 존재하는 경우도 병렬성이 활용될 수 되기 때문에), 한 질의에서 운영체제의 커널 인터페이스를 통해서 여러 개의 페이지를 요청하는 방식이 보편화될 경우, 소프트웨어 스택의 부담 (예를 들어, 운영체제 context switch 부담)을 줄일 수 있기 때문에 제안 기법의 유용성은 여전히 유효하게 된다.

4. DuraSSD (비휘발성 DRAM 캐시를 가진 SSD)

데이터베이스에서 관리하는 데이터는 system crash에도 불구하고 복구 (recovery)가 가능해야 하기 때문에, 데이터베이스에서 갱신된 페이지를 저장장치에 쓰는 경우 (즉, write() 시스템 콜을 사용해서), 운영체제에서 제공하는 fsync 시스템 콜을 사용해서, 파일 시스템 버퍼캐시의 데이터를 저장장치에 빈번하게 내리게 된다. 한편, 저장장치가 쓰기 연산의 속도를 개선할 목적으로 큰 용량의 DRAM 캐시를 갖는데, fsync 연산의 경우, IO 스케줄러나 system crash에서 쓰기 연산의 순서 보장 등의 이유로, 저장장치의 DRAM 캐시에 쓰여진 페이지를 하드디스크 매체나 플래시메모리 칩으로 안전하게 내리게 된다. 이때 주로 사용되는 메커니즘이 커널에서 저장장치로 불려지는 write barrier 명령이다. 즉, write barrier 명령을 받으면, 저장장치 펌웨어는 DRAM 캐시에 저장된 모든 페이지들을 비휘발성 저장 매체로 안전하게 쓰게 된다.

이와 같이, 쓰기 관점에서 데이터베이스는 데이터 일관성 및 복구 기능으로 인해, 아주 빈번한 fsync()와 그에 수반되는 write barrier 명령을 사용하게 된다. 그런데, 빈번한 write barrier는 쓰기 관점의 병렬성을 아주 낮게 만든다. 〈표 3〉은 fsync가 하드디스크와 플래시메모리 저장장치의 쓰기 IOPS에 미치는 영향을 평가하기 위해 fio IO 성능평가 도구를 이용해서 실험한 결과를 보여 주고 있다. 단일 쓰레드에서 쓰기 연산 회수별 fsync 주기를 조절하면서, 4KB 페이지의 IOPS 값을 측정하였다.

데이터베이스에서 관리하는 데이터는 system crash에도 불구하고 복구가 가능해야한다.

〈표 3〉에서 보여 지듯이, fsync 시스템 콜을 호출하지 않는 경우, 쓰기 IOPS가 아주 높은데 이는 DRAM 쓰기 버퍼에 페이지를 가득 채워 쓰기 연산과 관련된 플래시메모리 SSD의 병렬성을 극대화함으로써 얻어지는 성능이다. 한편, 매 쓰기 연산마다 fsync를 호출하는 경우, 쓰기 IOPS가 225로 엄청나게 낮아짐을 알 수 있다. 이는 매 write마다 플래시메모리 SSD내의 수많은 병렬성 경로 중 한 경로만 실제 쓰기 연산을 수행하기 때문이 처리량 입장에서 효용성이 아주 낮아지기 때문이다.

따라서 데이터베이스 응용의 입장에서 fsync에 따른

〈표 3〉 fsync()의 주기와 쓰기 IOPS[4]

구분	Fsync당 쓰기 연산 수			
	1	8	256	no fsync
하드디스크	59	184	381	387
SSD	225	1556	12,647	15,319

부담을 낮추고, 쓰기 IOPS 입장에서 플래시메모리 저장장치의 병렬성을 극대화하는 하나의 대안은, DRAM 캐시를 배터리백업해서 비휘발성으로 만듦으로써, 비록 fsync가 아주 자주 오는 경우에도 병렬성을 극대화하는 방안이다. 이러한 배경에서 만들어진 플래시메모리 SSD가 DuraSSD[4]이다. 이를 통해 실제 다양한 데이터베이스 벤치마크를 상용 DBMS, 오픈소스 DBMS, 그리고 NoSQL 시스템을 수행했을 때, 일반 SSD에 비해 10배 이상의 성능 향상을 얻을 수 있음을 확인하였다.

5. 인덱스 기반 질의 처리

Ⅱ장 〈표 2〉에서 설명한 바와 같이, 플래시메모리 저장장치에서는 임의읽기의 처리량이 순차 읽기 처리량에 거의 근접한다. 한편, 하드디스크의 경우, 두 읽기 방식에는 큰 차이가 있다. 따라서 하드디스크 기반의 데이터베이스에서는, 특정 테이블에 인덱스가 있는 경우에도 인덱스를 통해서 접근하는 데이터 페이지의 개수가 테이블 전체의 개수에 비해 아주 적을 때 (예를 들어, 1% 이하), 인덱스 기반의 질의 처리가 테이블 전체를 스캔하는 방식에 비해서 더 비용 효율적이다. 따라서 순차 읽기와 임의 읽기의 처리량의 격차가 점점 더 커진 하드디스크 저장장치의 경우, 데이터베이스 질의처리 방식은 인덱스 기반의 질의처리 방식에 비해 테이블 전체를 순차적으로 스캐닝하는 방법이 더 보편화되어왔다.

반면, 임의읽기의 처리량이 순차읽기 처리량에 근접하는 플래시메모리 저장장치의 경우, 데이터베이스 질의처리에서 전체 테이블 스캔 방법과 인덱스 스캔 방법에 대한 재고찰을 필요로 한다. 이러한 점에 착안한 최근의 연구 결과[5]에 따르면, 인덱스를 통해 접근되는 페이지 수가 테이블 전체 페이지 수의 50% 이상에 해당하는 경우에도, 인덱스 기반의 질의처리가 테이블 전체 스캔 방법보다 성능이 더 낮다.

Ⅳ. 향후 연구 방향

앞 장에서 비록 제한적이기는 하지만, 플래시메모리 저장장치의 서로 다른 여러 가지 특성들을 활용한 새로운 데이터베이스 기술들을 적용함으로써 다양한 데이터베이스 응용에서 성능을 획기적으로 향상할 수 있는 기회가 많음을 살펴보았다. 이러한 플래시메모리 저장장치 기반의 데이터베이스 기술 개발은 여전히 태동단계였고, 개발된 기술들을 모태로 향후에 모바일 및 엔터프라이즈향 오픈소스, 상용 데이터베이스에 적용하는 과정, 데이터베이스 분야를 위한 플래시메모리 저장장치 기술 분야 등 본격적인 연구가 시작될 것으로 예상된다. 본 저자 입장에서 예상되는 몇 가지 연구 방향은 다음과 같다.

플래시메모리 저장장치 기반의 데이터베이스 기술은 향후에 모바일 및 엔터프라이즈향 오픈소스, 상용 데이터베이스에 적용하는 과정, 데이터베이스 분야를 위한 플래시메모리 저장장치 기술분야 등 본격적인 연구가 시작될 것으로 예상 된다.

우선, 산업체에서 지능형 SSD (Smart SSD) 기술 개발에 대한 관심이 많은 관계로, 저장 장치가 단순한 읽기, 쓰기 인터페이스만을 지원하는 블록디바이스를 넘어서 데이터베이스를 포함한 호스트 소프트웨어에서 필요하고 중요한, 그리고 플래시메모리 저장장치 콘트롤러의 리소스를 이용해서 효율적으로 지원 가능한 기능들을 잘 정의해서 새로운 인터페이스 형태로 지원하는 흐름이 있을 것이다. 앞서 소개한 X-FTL과 유사한 형태의 상대적으로 단순한 새로운 인터페이스 추가부터 데이터베이스의 질의 처리 기능을 플래시메모리 저장장치에서 지원하는 방안[6]까지 넓은 스펙트럼의 시도가 가능하리라 예상한다. 특히, 멀티코어 및 대용량의 DRAM 캐시를 갖춘 플래시메모리 SSD의 경우 점점 더 많은 기능을 호스트로부터 저장장치로 내리는 방식 (off-loading)을 가능케 할 것이다.

다음으로, 앞서 설명한 PSync[3]의 개념처럼, 이미 DBMS나 NoSQL 시스템에서는 한 프로세스에서 인덱스를 이용해서 복 수 개의 임의의 페이지에 대한 읽기 연산을 요청하는 애플리케이션은 이미 존재하거나 향후 더 많아 질 것이다. 하지만, 운영체제 커널, 디바이스

드라이브, 그리고 플래시메모리 저장장치 인터페이스 차원에서 이 개념을 직접 지원하기 위해서는 아직 준비가 덜 된 상태이다. 이러한 use case를 바탕으로 호스트부터 저장장치 단까지 병렬적인 데이터 접근을 가능케 하는 기술 스택의 개선이 필요하다.

또한, NVMe (Non-Volatile-Memory Express) 와 같은 새로운 저장장치 인터페이스 출현으로 인해 많은 연구의 기회가 존재할 것이다. NVMe는 인텔을 중심으로 플래시메모리를 포함한 비휘발성 메모리 기반 저장장치를 위한 차세대 저장장치 인터페이스로, 현재 삼성전자에서 이를 지원하는 최초의 제품이 시장에 출시되기 시작하고 있다. 이 인터페이스에서는 단순 읽기 쓰기 이외에의 여러 가지 인터페이스 기능을 제공하고 있고, 이보다 더 중요하게 호스트에서 저장장치의 리소스를 직접적으로 그리고 효율적인 접근을 허용할 여지가 있다. 따라서 이는 데이터베이스의 여러 가지 기능들을 획기적인 방식으로 새롭게 구현할 수 있는 기회를 제공할 것이다.

또한, 현재 많은 반도체 회사들이 투자하고 있는 플래시메모리 이외의 비휘발성 메모리 기술들의 활용 방안과 그 기술의 현실성에 관한 검증이 필요하다. 비록 플래시메모리가 하드디스크에 비해 성능 관점에서 현격하게 빠르나, CPU와 DRAM에 비해서는 여전히 느린 것도 사실이다. 따라서 메모리 계층구조 관점에서 DRAM과 플래시메모리 저장장치 사이를 메울 새로운 저장장치 기술로, PRAM, MRAM 등 많은 기술들에 개발이 이루어지고 이를 데이터베이스 등의 소프트웨어에서 활용하는 방안들이 제시되고 있다[7]. 하지만, 이 흐름과 관련해서 근본적인 질문은, 이러한 새로운 메모리 소자들의 현실적인 성능 (특히, 쓰기 성능) 과 가격을 고려했을 때, 데이터베이스를 포함한 응용 소프트웨어 입장에서 얻어지는 실질적인 성능이 투자 가치가 있느냐는 점이다. 특히, 데이터베이스 엔진처럼 다수의 클라이언트가 수 십 또는 수백 개의 멀티스레드 형태로 동작할 경우, 단일 프로그램의 응답시간이 아닌 전체 시스템의 처리율 관점에서 있어, 현재의 플래시메모리 SSD만으로도 전체 시스템을 충분히 균형 상태로 (즉, CPU와 저장장치 사용률이 모두 100%인 상태) 만 들 수 있다. 또한, 저장장치의 배터리 백업된 DRAM 캐시 기술을 고려할 때[4], 데이터베이스 분야에서는 이러한 차세대 비휘발성 메모리 저장장치의 존재 필요성에 대한 재검토가 반드시 필요하다.

그리고, 플래시메모리 저장장치가 데이터베이스 아키텍처에 미치는 영향에 대한 검토가 필요하다. 최근 메인메모리 DBMS가 SAP, MS 등을 중심으로 활발하게 개발되고 있는데, 이는 하드디스크의 가격대비 아주 낮은 IOPS 성능으로 인해서, 비싼 DRAM 가격과 전력을 소모하더라도 모든 데이터를 메인메모리에 저장하는 것이 더 효율적이기 때문에 활성화된 측면이 있다. 하지만, IBM, Oracle 등의 DBMS 업체는 전통적인 DBMS에 플래시메모리 SSD를 캐시 또는 하드디스크를 대체하는 형태로도 응용에서 필요한 성능을 충분히 달성한다고 판단하고 있다. 따라서 이 두 접근 방식 중에서 어느 패러다임이 시장에서 주류가 될 지에 대한 판단은 시간이 필요하다. 다만, 이를 위한 이론적이고 체계적인 분석에 관한 연구도 필요하다.

이 외에도, 버퍼관리 기술[8], 질의 처리 및 최적화 등 전통적이면서 중요한 데이터베이스 기술 분야에 대한 연구도 여전히 필요하다.

Ⅴ. 맺음말

이상에서 플래시메모리 저장장치의 여러 가지 특성을 활용해서 데이터베이스 시스템 성능을 획기적으로 향상시키는 최근의 기술들을 소개하고, 향후 유망한 연구 방향에 대해 개략적으로 제시하였다..

빅데이터와 all-flash 패러다임을 고려할 때, 이 연구 분야는 여전히 획기적인 연구를 필요로 하고 있다. 특히, 플래시메모리 저장장치에 저장된 데이터를 사용하는 주된 소프트웨어는 데이터베이스이고, 이 데이터베이스의 데이터 접근 패턴이나 필요로 하는 기능, 요구되는 시맨틱을 고려하면, 운영체제나 파일시스템 계층보다 훨씬 더 많은 연구 기회가 있다. 따라서 데이터베이스 엔진 계층에 대한 이해를 바탕으로, 운영체제,

파일 시스템, 커널, 디바이스 드라이버는 말할 것도 없이, 플래시메모리 저장장치 내부 아키텍처, 펌웨어 로직, 저장장치 인터페이스 설계까지를 수직적으로 재고찰하는 연구가 필요하다.

참 고 문 헌

[1] W. H. Kang, S. W. Lee, B. Moon, G. H. Oh and C. Min, "X-FTL: Transactional FTL for SQLite Databases," in Proc. of ACM SIGMOD, pp. 97-109, New York, USA, June 2013.

[2] W. H. Kang, S. W. Lee and B. Moon, "Flash-based Extended Cache for Higher Throughput and Faster Recovery," in Proc. of VLDB, pp. 1615-1626, Istanbul, Turkey, August 2012.

[3] H. Roh, S. Park, S. Kim, M. Shin and S.W. Lee, "B+-tree Index Optimization by Exploiting Internal Parallelism of Flash-based Solid State Drives," in Proc. of VLDB, pp. 286-297, Istanbul, Turkey, August 2012.

[4] W. H. Kang, S. W. Lee, B. Moon, Y. S. Kee and M. W. Oh, "Durable Write Cache in Flash Memory SSD for Relational and NoSQL Databases," in Proc. of ACM SIGMOD, Snowbird, Utah, USA, June 2014 (to appear).

[5] E. M. Lee, S. W. Lee and S. Park, "Optimizing Index Scans on Flash Memory SSDs," SIGMOD Records, Vol. 40, no. 4, pp. 5-10, December 2011.

[6] S. Kim, H. Oh, C. Park, S. Cho and S. W. Lee, "Fast, Energy Efficient Scan inside Flash Memory," ADMS, Istanbul, Turkey, August 2012

[7] K. Kim, S. W. Lee, B. Moon, C. Park and J. Y. Hwang, "IPL-P: In-Page Logging with PCRAM," in Proc. of VLDB (demo track), pp. 1363-1366, Seattle, USA, September 2011.

[8] S. Y. Park, D. Jung, J. U. Kang, J. S. Kim and J. Lee, "CFLRU: A Replacement Algorithm for Flash Memory," in Proc. of ACM CASES (International Conference on Compilers, Architecture and Synthesis for Embedded Systems), pp. 234-241, Seoul, Korea, October 2006.

이 상 원

1991년 2월 서울대학교 컴퓨터공학 학사
1994년 2월 서울대학교 컴퓨터공학 석사
1999년 2월 서울대학교 컴퓨터공학 박사
1999년 1월~2001년 2월 한국오라클 기술과장
2001년 3월~2002년 2월 이화여자대학교 연구교수
2002년 3월~현재 성균관대학교 정보통신대학 교수

〈관심 분야〉
플래시메모리 데이터베이스 기술