플래시 메모리를 이용한 다중 버전 기반의 동시성 제어 기법의 성능 평가[1)]

오주형○ 김재명 나갑주 이상원
성균관 대학교 전자전기컴퓨터 공학과
{jhoh○, jam02, factory, swlee}@skku.edu

Performance Evaluation of Multi-Version Concurrency Control using a Flash Memory

Joo-Hyung Oh○, Jae-Myung Kim, Gap-Joo Na, Sang-Won Lee
Electrical and Computer Engineering, Sungkyunkwan University

요 약

데이터베이스 시스템은 전통적으로 트랜잭션의 동시 접근 시 발생할 수 있는 문제들을 해결하고 동시성 향상을 위해 다양한 연구를 진행해 왔다. 다중 버전 기반의 동시성 제어 기법은 데이터 레코드에 대한 여러 버전을 전용 공간에 유지하고 이것을 통해 트랜잭션들의 동시 접근 시 블로킹 없이 원하는 데이터를 읽고 쓸 수 있게 된다. 그러나 데이터 레코드가 포함된 데이터 블록에 대해 여러 개의 트랜잭션이 동시에 데이터를 덮어쓰기 했을 경우 다수의 버전이 생성된다. 그래서 트랜잭션 별로 적합한 데이터 버전을 찾기 위해 버전들이 저장되어 있는 전용공간을 랜덤하게 검색해 데이터 블록을 찾아내야 하므로 많은 시간이 소모된다. 따라서 다중 버전 읽기의 부하를 줄이기 위해 차세대 저장 매체로 부상하고 있는 플래시 메모리를 이용할 경우 랜덤 읽기에 의한 데이터베이스 시스템의 성능저하를 줄일 수 있다. 플래시 메모리는 디스크와 달리 기계적인 파트가 존재하지 않기 때문에 저장된 모든 블록에 대해 빠른 랜덤 읽기를 가능하게 한다. 본 논문에서는 플래시 메모리를 다중 버전 기반의 동시성 기법에 적용했을 경우의 성능 평가를 통해 하드 디스크에 비해 3.5배 이상의 높은 성능을 보임을 증명한다.

1. 서 론

데이터베이스 시스템들에서 다수의 트랜잭션이 동일한 데이터에 접근해서 동시에 작업을 수행할 경우 발생 할 수 있는 다양한 문제들을 해결하기 위해 동시성 제어(concurrency control) 기법을 사용하고 있다[1][2]. 전통적인 데이터베이스 시스템들은 단일 버전(single version)의 데이터를 유지하면서 락킹 (locking)기술을 사용하여 동시성을 제어하는 단일 버전 시스템이었다. 그러나 단일 버전 데이터베이스 시스템의 경우 트랜잭션의 쓰기 요청(write request)이 읽기 요청 (read request)을 블로킹(blocking) 하는 현상이 발생하게 된다. 이는 트랜잭션이 쓰기 작업을 완료 하지 않은 상태의 데이터가 다른 트랜잭션에 의해 읽혀질 경우 발생하는 이상 현상(i.e. Dirty read)을 방지하기 위함이다[3]. 그러나 블로킹은 데이터베이스 시스템의 동시성을 떨어트리는 가장 큰 원인이 된다. 그래서 현재 대부분의 상용 데이터베이스 시스템들은 다중 버전(Multi-Version) 기반의 동시성 제어 기법을 사용한다.

다중 버전 기반의 동시성 제어 기법은 트랜잭션이 데이터 블록에 쓰기(write)작업을 수행하는 동안 다른 트랜잭션들의 읽기 요청을 블로킹하지 않기 위해 롤백 세그먼트(rollback segment)와 같은 전용 공간에 덮어쓰기 이전의 정보를 복구 할 수 있는 언두(undo) 정보를 저장한다. 그래서 쓰기 작업이 수행되는 동안 다른 트랜잭션에 의한 읽기요청은 롤백 세그먼트에 저장된 언두 정보를 이용해 처리된다. 그러나 저장된 언두 정보를 읽기 위해서는 많은 랜덤(random) 블록 읽기가 필요하다. 트랜잭션별로 언두 블록의 집합인 롤백 세그먼트를 각각 사용하기 때문에 트랜잭션에 의해 변경되기 이전의 데이터는 물리적으로 랜덤하게 떨어진 블록에 저장되게 된다.

롤백 세그먼트를 하드디스크에 생성하고 사용하는 데이터베이스 시스템은 랜덤 블록 읽기에 따른 성능 저하를 가질 수밖에 없다. 그러나 하드디스크보다 더 빠른 랜덤 읽기를 저장된 데이터의 위치에 상관없이 일정한 속도로 제공하는 NAND 타입의 플래시 메모리 (이하 플래시 메모리)를 사용할 경우 기존 데이터베이스 시스템의 성능을 향상 시킬 수 있다. 따라서 본 논문에서는 기존 데이

1) 본 연구는 정보통신부 및 정보통신연구진흥원의 IT신성장동력핵심기술개발사업[2006-S-040-01,Flash Memory 기반 임베디드 멀티미디어 소프트웨어 기술 개발]과 정보통신부 및 정보통신연구진흥원의 대학IT연구센터 지원사업(IITA-2006-(C1090-0603-0046))의 일환으로 수행하였음

터베이스 시스템의 언두 정보를 위한 롤백 세그먼트를 하드디스크와 플래시 메모리에 각각 생성해서 사용했을 경우의 성능을 측정하여 비교한다.

본 논문의 기여사항은 다음과 같다. 다중 버전 기반의 동시성 제어 기법을 메모리를 이용해서 사용할 경우 데이터베이스 시스템의 성능을 향상 시킬 수 있다는 것을 증명하고 기존 데이터베이스 시스템의 소스 코드를 수정하지 않고도 플래시 메모리를 적용 할 수 있는 방안을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서 성능 평가를 위한 관련 연구로 다중 버전기반의 동시성 제어 기법과 플래시 메모리에 대해 설명한다. 3장에서는 수행한 성능 평가에 대해 설명하고 결과를 분석한다. 마지막으로 4장에서 결론 및 향후 연구에 대하여 기술한다.

2. 관련 연구

2.1 다중 버전 기반의 동시성 제어 기법

많은 데이터베이스 시스템들이 롤백 세그먼트와 같은 전용 공간에 언두 정보를 저장한다. 데이터베이스 시스템에서 이미 저장된 데이터 블록에 대해 트랜잭션의 덮어쓰기 요청이 수행되는 경우 이전 데이터 블록의 정보를 저장하기 위해 트랜잭션 별로 롤백 세그먼트를 할당받는다. 또한 할당 받은 롤백 세그먼트의 헤더에서 관리하는 프리 블록 풀 (free block pool)에서 새로운 프리 언두 블록을 할당 받아서 언두 정보를 저장한다. 그래서 덮어 쓰기 요청이 수행되는 동안 또는 그 이후에 다른 트랜잭션에 의한 읽기 요청은 롤백 세그먼트의 언두 블록에 저장된 정보를 이용해 처리될 수 있다. 이것은 트랜잭션의 쓰기 작업 동안에 다른 트랜잭션의 읽기 요청을 블로킹하지 않는 다는 것을 의미한다. 그리고 동일한 데이터 블록에 대해 한 트랜잭션이 다른 트랜잭션들의 덮어쓰기 이전에 읽은 데이터를 다른 트랜잭션의 덮어쓰기 이후에 다시 읽기를 원할 경우 Non-Repeatable reads 또는 Phantom reads와 같은 이상 현상이 발생하는 것을 방지하기 위해 읽기 연속성을 제공한다[3].

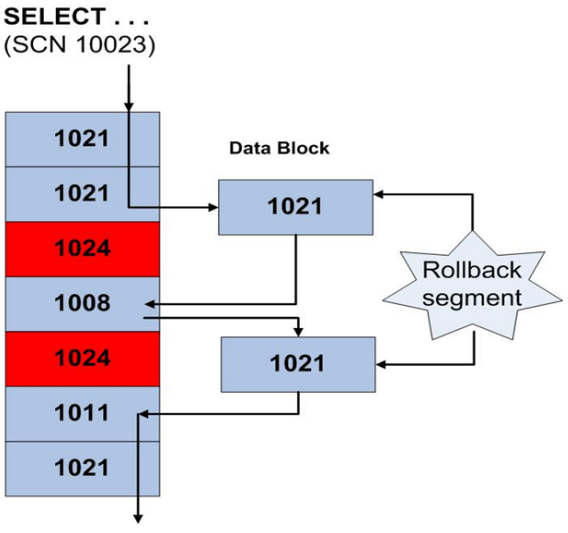

그림 1. 읽기 연속성

그림 1에서 보는 것과 같이 트랜잭션이 읽기 작업을 수행하는 동안에 3번째 블록이 다른 트랜잭션에 의한 덮어쓰기로 변경되었다는 것을 알 수 있다. 이 경우 읽기 연속성을 위해 현재 데이터 블록을 읽지 말고 롤백 세그먼트에 저장된 정보를 검색해서 덮어쓰기 이전의 데이터 블록을 만들어 내서 읽어야 한다. 그러나 다수의 트랜잭션이 동시에 데이터 블록에 쓰기 요청을 할 경우 각 트랜잭션별로 서로 다른 언두 블록을 할당 받아 정보를 기록하게 되므로 읽기 요청을 처리하기 위해 필요한 언두 정보들이 전체 롤백 세그먼트에 걸쳐 저장된다. 또한 트랜잭션별로 할당 받는 언두 블록의 연관성이 없기 때문에 언두 정보 검색을 위해서는 랜덤 블록 읽기가 필요하다.

언두 정보 저장의 경우 시장 점유율이 가장 높은 A사의 저장 패턴을 볼 때 트랜잭션 마다 기존 언두 정보에 항상 정보를 추가 하는 형태의 패턴을 보인다. 결국 다중 버전 기반의 동시성 제어 기법에서 언두 정보는 트랜잭션 별로 저장 공간에 차례로 쓰는 패턴을 보이고 저장된 정보는 랜덤 블록 읽기의 패턴을 보이게 된다.

2.2 플래시 메모리

플래시 메모리는 비휘발성 메모리의 한 종류로 기존 하드디스크에 비해 빠른 데이터 접근 성능을 보장하며, 부피와 전력 소모가 매우 작다. 또한 외부 충격에 강해 휴대용 기기의 저장 장치로 많이 사용되고 있다. 페이지들의 집합인 블록들로 구성된 플래시 메모리는 데이터를 쓰기 위한 기본 단위가 페이지이다. 그러나 이미 써진 페이지에 데이터를 덮어쓰기를 할 경우 해당 페이지가 포함된 블록을 지우고 새로 써야 하는 문제가 발생한다. 그래서 FTL (Flash Translation Layer)과 같은 시스템 소프트웨어를 사용하여 플래시 메모리의 단점을 보완한다.

플래시 메모리는 하드디스크와 많은 차이점을 가지고 있으며 속도 측면에서 표 1과 같은 성능을 보여 준다[5][6].

표 1. 수행 시간 비교 (HDD 와 NAND플래시)

장치	수행 시간		
	읽기	쓰기	지우기
HDD[1]	12.7ms (2KB)	13.7 ms (2KB)	N/A
NAND[2]	80us (2KB)	200 us (2KB)	1.5 ms (128KB)

[1] Seagate Barracuda 7200.7 ST380011A, seek time과 rotation time을 포함한 평균 접근 시간

[2] Samsung K9WAG08U1A 16Gbits SLC

표 1에서 보는 것과 같이 플래시 메모리는 2Kbyte 읽기의 경우 하드디스크에 비해 월등히 우수함을 알 수 있다. 또한 플래시 메모리는 랜덤하게 분포된 섹터에 대한 읽기도 동일한 속도를 보장한다. 이것은 기계적인 파트

가 존재하는 하드디스크와 달리 플래시 메모리는 seek time이나 rotation time이 없기 때문이다. 그러나 플래시 메모리는 하드디스크에서 볼 수 없는 지우기 연산이 필요하다. 앞에서 언급했듯이 플래시 메모리의 경우 이미 사용하고 있는 페이지에 대한 덮어쓰기가 발생했을 경우 해당 페이지를 포함한 블락을 지우고 써야 한다. 그래서 FTL과 파일 시스템에서 플래시 메모리의 단점을 극복하기 위한 다양한 알고리즘이 제안되고 있다. 그 중에서 로그 구조를 활용하는 방법이 가장 널리 사용되고 있다 [5].

또한 하드 디스크와 플래시 메모리는 읽기와 쓰기 패턴에 따라 다른 성능을 낼 수 있다.

그림 2. 읽기와 쓰기 패턴에 따른 성능

그림 2에서 보는 것과 같이 하드 디스크는 순차 쓰기와 순차 읽기에서 좋은 성능을 내는 반면 플래시 메모리는 랜덤 읽기와 순차쓰기에서 좋은 성능을 낼 수 있다. 따라서 기존 다중 버전 기반의 동시성 제어 기법에서 언두 정보 저장, 읽기 연속성 그리고 동시성 제어를 위해 필요한 많은 랜덤 블록 읽기 작업을 위해 플래시 메모리를 이용할 경우 데이터베이스 시스템의 성능을 향상 시킬 수 있다.

3.1 실험 환경 및 설명

플래시 메모리를 이용한 다중 버전 기반의 동시성 제어의 성능 측정을 위한 실험 환경은 표2와 같다.

표 2. 실험 환경

Database	A사
Database buffer	150 MB
Redo buffers	2 MB
Rollback segment size	1 GB

정확한 실험을 위해 하드디스크와 플래시를 이용한 SSD (Solid State Disk)를 각각 로우 디바이스(raw device)로 데이터베이스 시스템에 연결해 언두 정보를 위해 사용하였다. 또한 성능평가를 위한 스크립트를 작성해 동시에 다수의 트랜잭션들이 데이터 블록을 변경하는 상황을 재현하였다. 스크립트의 내용은 다음과 같다.

- 트랜잭션 A는 100M의 데이터를 삽입(Insert) 한 후 커밋(commit)을 수행한다. 또한 트랜잭션 A의 고립도 레벨(Isolation level)을 serializable로 변경한 다음 삽입한 데이터에 대해 읽기 작업을 수행한다.
- 읽기가 완료 된 후에 트랜잭션 B는 A가 삽입한 레코드들을 덮어쓰기 한 후 커밋을 한다.
- 이후 트랜잭션 C, D가 차례로 덮어쓰기를 수행하고 작업이 완료 된 다음 트랜잭션 A는 읽기 연속성을 위해 처음 삽입했던 데이터에 대해 읽기 작업을 수행한다.

위 스크립트를 수행하게 되면 A 트랜잭션이 삽입한 100Mbyte의 데이터들은 데이터 블록에 저장된다. 삽입이 완료 된 후 트랜잭션의 고립도 레벨을 serializable로 변경하는 것은 이 후 다른 트랜잭션에 의해 A가 삽입한 데이터가 변경되더라도 변경되기 이전의 데이터를 읽을 수 있어야 한다는 것을 의미한다. 고립도 레벨 변경에 이어 읽기 작업이 완료 된 다음 트랜잭션 B에 의해 덮어쓰기가 수행되어 질 경우 데이터 블록의 데이터들은 B가 덮어쓰기를 한 데이터들로 바뀌게 된다. 이때 트랜잭션 A에 의해 처음으로 삽입된 데이터들은 롤백 세그먼트 중 하나를 할당 받아 롤백 세그먼트가 관리하는 언두 블록에 저장된다. 또한 트랜잭션 C의 업데이트가 수행되게 되면 데이터 블록은 C가 덮어쓰기를 수행한 데이터들로 바뀌게 되고 마찬가지로 트랜잭션 B가 덮어쓰기를 수행한 데이터들은 롤백 세그먼트를 하나 할당 받아서 언두 블록에 저장되게 된다. 트랜잭션 D의 덮어쓰기도 위와 같다. 모든 트랜잭션의 덮어쓰기가 수행된 후 트랜잭션 A가 다른 트랜잭션에 의한 덮어쓰기 이전의 데이터들을 읽고자 할 경우 데이터베이스 시스템은 롤백 세그먼트를 검색해서 이전 데이터를 생성해 낸다. 결국 스크립트를 통해 다중 버전 기반의 동시성 제어 기법에서 발생하는 많은 랜덤 블록 읽기를 재현해 낼 수 있다.

3.2 실험 결과 분석

스크립트를 이용하여 하드디스크와 SSD에서 10번의 실험을 수행한 결과 그림 3과 같은 그래프를 얻을 수 있다.

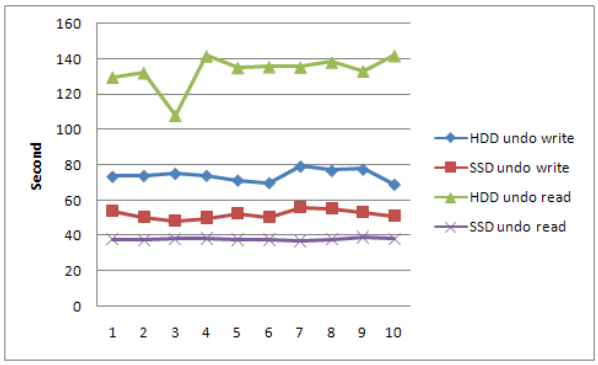

그림 3. 언두 정보 저장과 읽기 성능 비교

그림 3을 보면 언두 정보 저장과 읽기를 위해 플래시 메모리를 사용했을 경우 성능향상이 되었음을 알 수 있다. 특히 언두 블록 읽기의 경우 SSD가 하드디스크에 비해 3.5배 이상의 성능을 보이는 것을 확인 할 수 있다. 그러나 플래시 메모리의 특성을 고려해 볼 때 10배 이상의 빠른 속도를 보일 것으로 기대 된다. 위 실험 결과는 읽기 연속성을 위해 롤백 세그먼트만 검색을 하는 것이 아니라 최근 데이터가 저장된 하드디스크에 대한 검색이 포함되어 있기 때문이다. 그러나 본 성능평가에서는 롤백 세그먼트만을 교체한 후 테스트했기 때문에 원하는 결과를 얻기 위해서는 기존 데이터베이스 시스템의 구조를 변경해야 한다. 이것은 데이터베이스 시스템의 소스코드를 변경하지 않고 플래시 메모리를 사용해 성능향상을 하고자 하는 본 논문의 방향에 위배되므로 고려하지 않는다.

따라서 그림3에서 얻은 결과는 3.5배 이상의 성능향상을 이룰 수 있다는 것을 의미한다.

4. 결론

대부분의 데이터베이스 시스템들은 다중 버전 기반의 동시성 제어를 위해 언두 정보를 하드디스크에 저장하고 사용한다. 그러나 저장된 언두 정보를 랜덤하게 읽는 경우가 대부분이기 때문에 하드디스크기반의 기존 시스템은 동시에 접근하는 트랜잭션의 수가 늘어날수록 더 많은 성능저하를 야기한다. 또한 언두 정보는 전용 공간에 항상 추가 되는 형태의 저장 패턴을 보이게 된다. 따라서 저장된 데이터의 위치에 상관없이 동일한 속도로 랜덤 읽기가 가능하고 빠른 순차 쓰기가 가능한 플래시 메모리를 다중 버전 기반의 동시성 제어 기법을 위해 사용할 경우 뛰어난 성능을 얻을 수 있다. 본 논문에서 실험을 통해 디스크 대신 플래시를 이용해 다중 버전 기반의 동시성 제어 기법을 사용 했을 경우 디스크에 비해 3.5배 이상의 성능 향상을 이룰 수 있다는 것을 증명하였다.

향후 과제로 다중 버전 기반의 동시성 제어 기법을 더 연구하여 더 많은 성능 향상을 할 수 있도록 연구해야 한다.

참고 문헌

[1] P. A. Bernstein and N. Goodman, "Multiversion Concurrency Control-Theory and Algorithms", ACM Transaction on Database Systems, Vol.8, No.4, 1983

[2] D. Majumdar, "A Quick Survey of MultiVersion Concurrency Algorithms", 2006

[3] P. A. Bernstein, V. Hadzilacos and N. Goodman, Chapter 5: Multiversion Concurrency Control", Concurrency Control and recovery in Database Systems, Addison-Wesley Publishing Company, 1987

[4] ANSI X3.135-1992, American National Standard for Information Systems — Database Language — SQL, November, 1992

[5] J. S. Kim, J. M. Kim, S. H. Noh, S. L. Min and Y. Cho, "A Space-Efficient Flash Translation Layer for CompactFlash Systems. IEEE Transactions on Consumer Electronics, 48(2):366-375, 2002.