플래시 메모리 기반 임베디드 데이터베이스 시스템을 위한 지연쓰기 기법

윤승희°, 송하주
부경대학교 전자컴퓨터정보통신공학부
sexyshaq@lycos.co.kr°, hjsong@pknu.ac.kr

Delayed Write Scheme for The Flash Memory based Embedded Database Systems

Seung-Hee Yun°, Ha-Joo Song
Divison of Electronic, Computer, and Telecommunication Engineering
Pukyong National University

요 약

플래시 메모리는 동작 특성상 메모리 영역에 대한 덮어쓰기(overwrite)가 불가능하고 메모리 쓰기를 위해서는 삭제(erase) 연산을 반드시 먼저 수행해야 한다. 삭제 연산은 읽기 연산에 비해 많은 시간이 소요되므로 될수록 줄이는 것이 플래시 메모리의 수행 성능 향상에 유리하다. 본 논문에서는 플래시 메모리에 대한 삭제 횟수를 줄이기 위해 데이터베이스 페이지에 대한 쓰기 연산을 지연하는 지연쓰기 기법을 제안한다. 이 기법은 페이지에 대한 갱신이 일어날 때 페이지캐시 내의 해당 페이지에 대해서는 갱신을 수행하되 그것을 유발한 레코드 연산(레코드 삽입, 갱신, 삭제)은 별도의 지연쓰기 큐에 기록한다. 그리고 레코드 연산이 지연쓰기 큐에 저장되어 있는 동안에는 해당 페이지에 대한 갱신은 보류한다. 만약 해당 페이지를 다시 읽어야할 필요가 있을 때에는 지연 쓰기 큐에 저장된 갱신 정보와 병합하여 갱신된 페이지를 페이지 캐시에 적재한다. 이는 갱신되는 페이지의 개수와 단일 페이지에 대한 갱신 횟수를 감소시키는 효과를 가져온다. 따라서 플래시 메모리의 삭제 및 쓰기 연산을 감소시켜 데이터베이스 시스템의 수행성능을 향상시키게 된다.

1. 서 론

최근 들어 컴퓨팅 분야의 기술발전이 유비쿼터스 기반의 컴퓨팅으로 변환됨에 따라 휴대폰, PDA, 스마트카드 디지털 카메라와 같은 이동 컴퓨팅 장치의 사용이 급증하면서 플래시 메모리에 대한 관심이 증가하고 있다 플래시 메모리는 가볍고 물리적인 충격에 강해 휴대가 용이하며 저전력으로 동작해 배터리 소모량을 줄이기 때문에 하드디스크를 대체할 매체로 각광받고 있다 하지만 플래시 메모리는 빠른 읽기연산 속도에 비해 쓰기연산 속도[1.2]가 다른 메모리보다 느리고 하드디스크처럼 덮어쓰기 연산이 지원되지 않는 단점을 가지고 있다[2].

플래시 메모리 상에서 덮어쓰기를 수행하기 위해서는 먼저 해당 블록(block)에 대해 플래시 메모리의 연산 중 가장 비용이 많이 요구되는 삭제 연산을 수행해야만 한다. 이러한 문제점을 극복하기 위해 플래시 메모리는 파일 시스템과 플래시 메모리 사이에 위치하는 플래시 변환 계층(flash memory translation layer, FTL)이란 시스템 소프트웨어를 사용한다. 플래시 변환 계층은 다양한 블록교체기법을 적용하여 사용자에게 비교적 수행시간이 오래 걸리는 삭제 연산에 대하여 이미 삭제 연산을 수행한 빈 블록으로 재사상(remapping)함으로써 플래시 메모리를 기존의 하드디스크와 같이 사용할 수 있게 한다. 그러나 FTL을 이용하여 덮어쓰기를 지원하는 것은 기존 운영체제 또는 데이터베이스 시스템에 플래시 메모리를 바로 사용할 수 있다는 장점이 있는 반면 실행 성능이 매우 떨어진다. 이에 플래시 메모리에서 효율적으로 동작하는 파일 시스템에 대한 연구가 진행되었으며 그 결과 로그 파일 시스템[6]에 기초한 JFFS, JFFS2, YAFFS와 같은 새로운 파일 시스템들이 제안되었다.

※ 본 논문은 정통부 및 정보통신연구진흥원의 정보통신선도기반기술 개발사업의 연구결과로 수행되었습니다

2. 관련 연구

플래시 메모리가 대용량화되면서 저장장치로 각광받게 된 것은 극히 최근의 일이다. 따라서 플래시 메모리에 기반한 데이터베이스 시스템에 대한 연구가 아직은 그리 많지 않은 편이나 일부 데이터베이스 색인에 대한 연구 결과가 발표된 바 있다[3,7]. 일반적으로 데이터베이스 시스템이 효과적으로 동작하기 위해서는 해시 테이블(hash table) 또는 검색 트리(search tree) 와 같은 색인 기법들이 필요하며 특히 B-트리(B^+, B^* 포함) 색인 구조는 데이터의 효율적인 삽입 삭제 검색이 용이하며 확장성이 우수한 색인기법으로 유명하다. 하지만 하드디스크에서와 달리 플래시 메모리 상에서 구현한 B-트리는 빈번한 페이지 갱신에 따른 덮어쓰기 증가로 인해 인덱스

구축비용이 매우 커진다는 문제가 있다. 따라서 플래시 메모리 환경에 적합한 트리 구현기법이 필요하다. 플래시 메모리 상에서 트리를 구현하기 위해 기존 연구[3]에서 BFTL(B-Tree Flash Translation Layer: BFTL) 기법이 제안되었으나 거대한 사상 테이블을 유지하기 위한 별도의 SRAM 영역이 필요하며 트리 검색 시 읽기연산이 다수 발생하여 검색속도가 떨어지는 단점을 가지고 있다. 이외에도 일반 데이터베이스 시스템의 고장회복(recovery) 성능을 높이기 위해 플래시 메모리를 활용하는 방법에 대한 일부 연구가 있다.

3. 지연쓰기 기법

제안하는 기법은 모바일 환경과 같은 임베디드 데이터베이스 시스템을 대상으로 한다. 이러한 시스템들은 일반적인 대규모 데이터베이스 시스템과는 달리 다음과 같은 특징이 있다.

- 데이터베이스 응용 프로그램과 데이터베이스 시스템이 별개의 프로그램(또는 프로세스)로 구분되지 않고 두 부분이 단일 프로그램으로 결합되어 동작한다. 따라서 데이터베이스는 라이브러리 형태로 제공된다.
- 동시성제어(concurrency control)는 데이터베이스 단위와 같이 비교적 큰 단위의 잠금 단위 (locking granularity)를 사용한다.

제안하는 기법은 이러한 가정을 바탕으로 하여 단일 데이터에 대해서는 오직 하나의 프로세스만이 갱신연산을 수행할 수 있는 것으로 한다. 아울러 쓰기지연의 대상이 되는 데이터베이스 페이지는 인덱스 페이지와 작은 레코드(small record[1])들 저장하는 페이지만이 해당되며 BLOB(binary large object)의 데이터를 저장하기 위한 페이지는 해당되지 않는다.

제안하는 기법은 데이터베이스 페이지에 대한 갱신을 다음과 같은 과정으로 수행한다.

Step 1. 지연쓰기 큐에서 갱신 페이지에 대한 지연쓰기 레코드(deferred write record)의 개수 및 그 크기의 합을 계산한다.

Step 2. 만약 지연쓰기 레코드의 개수가 일정 개수 (TH_{count}) 이상이거나 지연 쓰기 크기의 합과 새로 갱신될 크기의 합이 일정 크기(TH_{size}) 이상이면 지연쓰기 큐에서 해당 페이지와 관련된 지연쓰기 레코드를 모두 제거한다. Step 4로 진행한다.

Step 3. 그렇지 않은 경우, 갱신 연산을 유발한 데이터베이스 연산(레코드 삽입, 삭제, 갱신)에 대한 정보를 포함하는 지연쓰기 레코드를 생성하고 지연쓰기 큐에 삽입한다.

Step 4. 해당 페이지를 갱신한다.

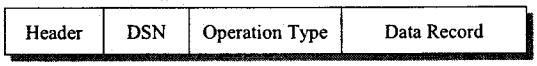

그림 3 지연쓰기 레코드의 구조

그림2는 지연쓰기 레코드의 구조를 나타낸 것으로 각 필드는 다음과 같이 구성된다.

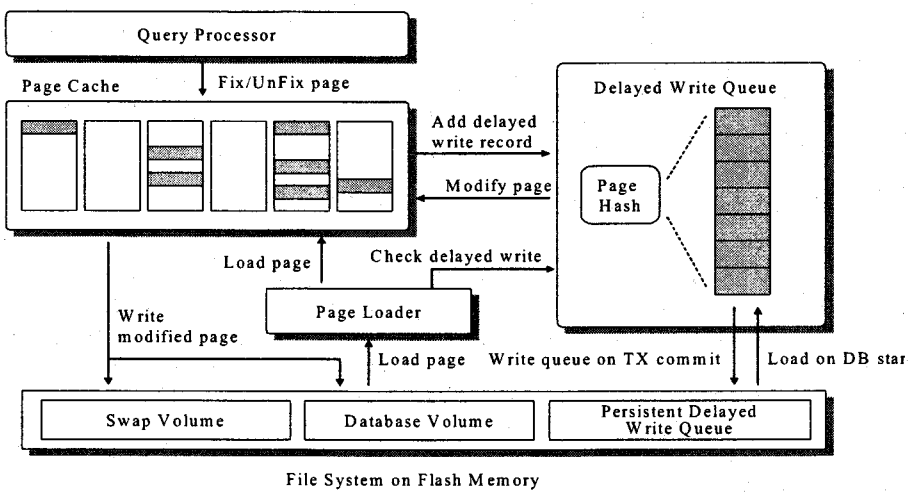

그림 2 지연쓰기를 위한 페이지 캐시 구조

1) 레코드의 크기가 데이터베이스의 페이지 크기보다 작은 레코드

- Header : 레코드 헤더
- DSN: 단조 증가하는 일련번호
- Operation Type: 삽입/삭제/갱신
- Data Record: 바이트 배열로 표현된 실제 저장될 레코드

페이지캐시의 상위 계층(예. 질의처리기)에 의해 데이터베이스 페이지를 플래시 메모리에서 읽어 와야 하는 경우에는 다음과 같이 동작한다.

Step 1. 플래시 메모리로 부터 지정된 페이지 영역을 읽어 들여 페이지 캐시에 적재한다.

Step 2. 페이지의 페이지식별자(Page Identifier, PID)를 사용하여 지연쓰기 큐를 검사한다. 이 과정에서 Page Hash를 사용된다.

Step 3. 만약 해당 페이지와 관련된 지연쓰기 레코드가 존재하면 해당되는 지연쓰기 연산을 수행한다. 만약 다수의 지연쓰기 레코드가 해당되면 큐에 삽입된 순서대로 지연쓰기 연산을 수행한다.

트랜잭션 커밋 이전에 갱신된 페이지가 페이지 캐시에서 교체(replace)되어야 하는 경우에는 다음과 같이 동작한다.[2)]

Step 1. 해당 페이지에 대한 지연쓰기 레코드가 존재하는 경우에는 페이지 캐시에서 해당 페이지를 무효화(invalidate)한다. 이 페이지가 다시 페이지 캐시에 적재되는 경우에는 이전에 설명된 페이지 읽기 연산을 거치게 된다.

Step 2. 그렇지 않은 경우에는 해당 페이지를 스왑영역에 기록하고 무효화한다음 스왑아웃(swap-out)되었음을 기록한다.

지연쓰기 큐가 일정 크기(queue size) 이상이 되면 지연쓰기 레코드들을 페이지식별자별로 정렬하고 동일 페이지에 속한 레코드의 수가 많은 것들을 우선해서 선택한다. 만약 동일한 우선순위에 속한 페이지가 다수인 경우에는 소속된 레코드들의 크기의 합이 큰 것을 우선해서 선택한다. 이 과정을 통해 선택된 지연쓰기 레코드들이 속한 페이지를 페이지 캐시에 적재하고 지연된 쓰기 연산을 모두 수행한다.

트랜잭션 커밋시(commit)에는 다음과 같이 동작한다[3)]

Step 1. 페이지캐시는 트랜잭션 수행도중 갱신된 페이지(dirty pages) 들을 찾아낸다.

Step 2. 갱신된 페이지에 관계된 지연쓰기 레코드가 존재하는 지를 검사한다.

Step 3. 존재하는 경우에는 갱신된 페이지의 DSN과 해당 페이지의 지연쓰기 레코드들 중 DSN이 최대인 것과 비교한다. 페이지의 DSN이 작은 경우에는 페이지에 지연된 쓰기 연산을 모두 수행한 다음 페이지를 플래시에 기록한다.

Step 4. Step3에서 제외된 경우 그리고 Step2에서 관련된 지연쓰기 레코드가 존재하지 않는 페이지들은 플래시 메모리에 기록하지 않는다.

Step 5. 지속성 지연쓰기 큐를 위한 파일 공간(그림1의 Persistent Delayed Write Queue 공간)에 저장되어 있는 레코드들을 삭제하고 지연쓰기 큐에 존재하는 모든 레코드들을 저장한다.

저장된 지연쓰기 레코드들은 데이터베이스 시스템이 재시작되는 경우 플래시 메모리에서 메인 메모리로 읽혀지게 된다.

정상 동작 중에 응용프로그램에 의해 트랜잭션이 취소되는 경우에는 지속성 저장장치에 기록된 지연쓰기 레코드의 DSN중 최대값(DSN_{fmax})보다 큰 DSN을 가지는 지연쓰기 레코드를 메모리 내의 지연쓰기 큐에서 제거한다. 그리고 페이지 캐시내의 페이지들에서 갱신된 페이지들

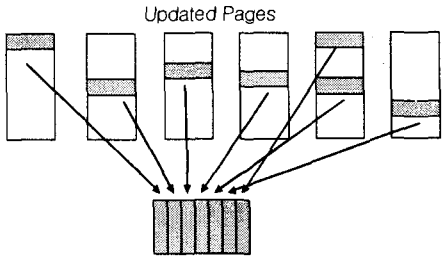

그림 4 페이지 쓰기의 감소

을 검색한다. 만약 갱신된 페이지의 DSN이 DSN_{fmax} 보다 작으면 해당 페이지는 캐시에 계속 보존하되 그렇지 않은 경우에는 무효화시킨다.

그림3은 제안하는 기법이 우수한 성능을 나타낼 수 있는 경우를 나타낸 그림이다. 이 경우 다수의 페이지에 걸쳐 소량의 갱신이 이루어지는 경우를 나타낸 것이다. 제안된 기법을 사용하지 않는 경우 페이지의 캐시 여부와는 무관하게 언젠가는 해당 페이지들은 플래시 메모리 상에서 갱신이 이루어져야 한다. 반면 제안한 기법은 다수의 페이지 갱신하는 대신 지연쓰기 레코드들만을 플래시에 기록하면 되므로 실제 갱신되는 페이지는 크게 줄어들 수 있다. 이는 데이터 레코드의 크기가 100 바이트 라고 가정하고 페이지의 크기가 4K 또는 8K 정도라고 할 때 최대 1/40 ~ 1/80 수준으로 페이지 쓰기를 줄일 수 있음을 의미한다. 또한 이 때, 지연쓰기 페이지들은 연속된(contiguous) 페이지들로 저장할 수 있다. 연속된 페이지들에 대한 쓰기 연산은 디스크에서뿐만 아니라 플래시 메모리에서도 임의(random) 쓰기보다 좋은 성능을 나타내므로 부가적인 입출력 성능의 향상을 꾀할 수 있다.

2) UNDO를 위해 스왑영역(swap area)를 사용하는 것으로 가정한다.
3) 커밋시 페이지 캐시는 FORCE 전략(strategy)을 사용하는 것으로 가정한다.

제안하는 기법은 CPU 사용 측면과 메모리 사용 측면에서 약간의 오버헤드가 따른다. 페이지를 읽을 때 마다 지연쓰기 큐를 페이지식별자를 이용하여 검색하고 만약 지연쓰기된 페이지로 판명되면 지연쓰기 레코드를 이용하여 페이지를 최신 상태로 변경해야 한다. 여기서 페이지식별자를 이용한 검색 과정에 소요되는 시간은 애초에 그다지 크지 않으며 코드 최적화를 통해 상당부분 줄일 수 있다. 그러나 캐시 내의 페이지를 최신 상태로 변경하는 과정은 상당 시간이 소요될 수 있다. 이를 위해 제안하는 기법에서는 지연쓰기가 일정 횟수(TH_{count}) 또는 일정 크기 이상(TH_{size})이 되면 지연쓰기를 하지 않고 플래시상의 페이지를 갱신하도록 하였다. 최근 CPU의 성능이 급격히 향상되고 있음을 감안했을 때 이 과정에서 소요되는 시간은 더욱 줄어들 수 있을 것으로 예상한다.

부가적으로 소요되는 메모리 공간 중 가장 큰 부분은 지연쓰기 큐와 페이지 해시에 사용되는 SRAM이다. 지연쓰기 큐를 200으로 하고 레코드의 크기가 100바이트라면 약 20K 바이트 정도의 SRAM과 플래시 메모리를 사용하게 된다.

4. 기존 연구와의 비교

BFTL 기법의 예약버퍼(reservation buffer)는 제안하는 기법의 지연쓰기 큐는 페이지에 대한 갱신 연산을 덮어쓰기로 수행하지 않고 별도의 플래시 공간에 저장한다는 점에서는 유사하다 할 수 있다. 반면 제안하는 기법에서는 페이지가 캐시에 적재될 때 지연쓰기 큐에 저장된 레코드를 적용하므로 페이지 캐시에는 갱신된 페이지가 사용되는 반면 BFTL에서는 갱신 이전의 페이지와 갱신 데이터가 서로 독립적으로 존재하고 예약버퍼 측에서 데이터를 검색하면 페이지 캐시의 데이터는 사용되지 않는다는 점에서 차이가 있다.

로그버퍼 FTL 기법(log buffer FTL)[5]은 플래시 메모리의 섹터에 대해 갱신이 일어나는 경우 플래시 메모리의 일부 영역을 로그버퍼로 지정하고 여기에 변경된 데이터를 저장하는 FTL 기법이다. 제안하는 기법과 로그버퍼 기법 모두 동일한 페이지가 연속적으로 갱신되는 경우에 유리하다는 점에서는 유사한 특징을 가진다. 반면 동작하는 수준이 섹터와 페이지라는 점에서의 차이가 있으며 제안하는 기법은 페이지의 내용 중 변화된 부분만을 유지하는 반면 로그버퍼 기법은 변경된 전체 섹터를 유지한다는 점에서 차이가 있다.

개념적으로 제안하는 기법과 가장 유사한 것은 레코드 레벨 REDO 로깅 방법이라 할 수 있다. 양측 모두 페이지에 갱신이 이루어질 때에 갱신을 유발한 레코드 수준에서 로그 레코드를 생성하고 트랜잭션 커밋시에 이것을 지속성 저장장치에 기록하는 면에서 동일하다. 반면 REDO 로그는 데이터베이스 시스템 재가동시 고장회복 단계에서 커밋된 데이터가 데이터베이스에 반드시 반영되는 용도로 사용되는 것인 반면 지연쓰기 레코드는 순전히 페이지 갱신을 지연하기 위한 용도이다. 따라서 고장회복 단계에서도 페이지가 갱신은 되었으되 REDO 로그 레코드가 적용되지는 않았더라도 만약 그 갱신 내역이 지연쓰기 레코드에 기록되어 있다면 해당 페이지를 갱신하지 않아도 된다. 또한 지연쓰기 기법은 고장회복 단계가 아닌 정상적인 동작 중에 사용하는 것을 주요한 목적으로 한다는 점에서도 차이가 있다.

5. 결론 및 향후 연구

쓰기와 소거 연산은 플래시 메모리의 연산중에 가장 큰 시간이 소요된다. 제안하는 기법은 지연쓰기 레코드를 이용하여 단일 페이지에 다수의 갱신이 일어나는 경우 그리고 소량의 갱신이 여러 페이지에 걸쳐 일어나는 경우 플래시 메모리에 대한 쓰기와 소거 연산을 감소시킨다. 제안하는 기법은 대표적인 프리웨어(freeware) 임베디드 데이터베이스 시스템 중의 하나인 SQLite Version 3에 구현되고 있다. 추후에는 플래시 메모리의 하드웨어 구조에 근거하여 쓰기/읽기 연산을 효율적으로 수행할 수 있는 입출력 방식에 대한 연구가 수행될 예정이다.

참고문헌

[1] Michael Wu, and Willy Zwaenpoel, "eNVy: A Non-Volatile, Main Memory Storage System", In proceeding of 6th Symposium on Architectural Support for Programming Languages and Operating System, pp.86-87, 1994.

[2] Intel Corporation, Understanding the Flash Translation Layer(FTL) Specification, http://developer.intel.com/.

[3] Chin-Hsien Wu, Li-Pin Chang, Tei-Wei Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems", Real-Time Computing Systems and Applications, 2003.

[4] Intel corporation, 3 Volt Synchronous Intel StrataFlash Memory, http://www.intel.com/.

[5] J.Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, A Space-Efficient Flash .Translation Layer for Compact Flash System, IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp.366-375, 2002.

[6] M. Rosenblum, and J.K. Ousterhout, The Design and Implementation of a Log-Structured File System, ACM Transaction on Computer Systems, 1992.

[7] Christophe Bobineau, Luc Bouganim, Philippe Pucheral, Patrick Valduriez, "PicoDBMS: Scaling down Database Techniques for the Smart Card", In proceeding of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000.

[8] Michael Owebs, "The Definitive Guide to SQLite", Apress, 2006.