SQLite DBMS에 IPL 기법 응용
(Applying In-Page Logging to SQLite DBMS)

나 갑 주 † 김 상 우 † 김 재 명 † 이 상 원 ††
(Gap-Joo Na) (Sang-Woo Kim) (Jae-Myung Kim) (Sang-Won Lee)

요 약 플래시메모리는 휴대폰, 디지털카메라를 비롯한 휴대기기의 저장장치로 널리 사용된다. 최근에는 하드디스크와 같은 인터페이스를 가지는 플래시 SSD가 일부 노트북의 하드디스크를 대체하고 있다. 하지만 플래시메모리는 데이타베이스 시스템의 저장장치로 고려되지 못하고 있다. 플래시메모리를 하드디스크처럼 사용하기 위한 플래시 변환 계층이 임의의 영역에 많은 양의 덮어쓰기 연산을 유발하는 데이타베이스의 워크로드에서 나쁜 성능을 보이기 때문이다. 이러한 문제를 해결하기 위해 In-Page Logging (IPL)이라는 기법이 제안되었다. 이 논문에서는 잘 알려진 오픈소스 데이타베이스인 SQLite에 IPL 기법을 적용하여 성능을 평가하였고, 갱신 질의 처리 성능이 30배 향상됨을 보인다.

키워드 : 플래시메모리, 데이타베이스, In-Page Logging, SQLite

Abstract Flash memory has been widely used in mobile devices, such as mobile phone and digital camera. Recently flash SSD(Solid State Disk), having same interface of the disk drive, is replacing the hard disk of some laptop computers. However, flash memory still cannot be considered as the storage of database systems. The FTL(Flash Translation Layer) of commercial flash SSD, making flash memory operate exactly same as a hard disk, shows poor performance on the workload of databases with many random overwrites. Recently In-Page Logging(IPL) approach was proposed to solve this problem. In this paper, we implement IPL approach on SQLite, a popular open source embedded DBMS, and evaluate its performance. It improves the performance by up to 30 factors for update queries.

Key words : Flash Memory, Database, In-Page Logging, SQLite

1. 서 론

플래시메모리는 전원이 차단되어도 저장된 데이타가 지워지지 않는 비휘발성 저장장치로 최근 들어 저장용량의 증가와 가격의 하락으로 인해 디스크를 대체할 수 있는 저장 장치로 각광을 받고 있다. 기계적인 장치에 의해 데이타 접근을 수행하는 디스크와 달리 플래시메모리는 전기적 신호만을 사용한다. 따라서 적은 전력소비와 고속의 접근 속도를 제공하며 강한 내구성과 가벼운 무게 등의 특성을 가진다. 이러한 특성으로 디지털 카메라, MP3 플레이어, 휴대용전화기, PDA 등의 임베디드 기기의 시장에서 플래시메모리는 이미 주류[1]의 저장장치로 사용되고 있으며 최근에는 노트북 컴퓨터등과 같이 임베디드 기기 외에서도 하드 디스크를 플래시메모리로 대체하는 추세가 나타나고 있다[2].

그러나 플래시메모리의 뛰어난 장점들과 달리 하드디스크와 같은 디스크 기반의 저장장치를 직접 플래시메모리로 대체할 경우 몇 가지 제약사항을 고려하여야 한다. 먼저 플래시메모리는 디스크기반의 저장 장치와 달리 덮어쓰기(overwrite) 연산을 수행할 수 없다. 또한, 거의 동일한 연산 속도로 읽기, 쓰기를 수행하는 디스크 장치와 달리 플래시메모리에서는 읽기, 쓰기, 지우기의 세 가지 연산을 사용하며 각 연산을 처리하는 속도(읽

· 본 연구는 지식경제부 및 정보통신연구진흥원의 IT핵심기술개발사업 [2006-S-040-03, Flash Memory 기반 임베디드 멀티미디어 소프트웨어 기술 개발]과 대학 IT연구센터 지원사업(IITA-2008-(C1090-0801-0046)) 의 일환으로 수행하였음

† 학생회원 : 성균관대학교 전자전기컴퓨터공학과
factory@skku.edu
spun@skku.edu
jam02@skku.edu
†† 정 회 원 : 성균관대학교 컴퓨터공학부 교수
swlee@skku.edu

논문접수 : 2007년 9월 20일
심사완료 : 2008년 8월 20일

정보과학회논문지: 데이타베이스 제35권 제5호(2008.10)

기>쓰기>지우기) 역시 다르다. 또한 이러한 연산은 동일한 저장 공간에서 덮어쓰기 연산을 수행할 수 없음을 의미하며 덮어쓰기 연산을 위해서는 먼저 지우기 연산을 수행한 후 쓰기 연산을 수행(erase-before-write) 해야만 한다. 따라서 디스크와 동일한 연산(읽기, 쓰기)을 제공하기 위해 주로 Flash Translation Layer (FTL)이라 불리는 별도의 시스템 소프트웨어를 사용하여 디스크 기반의 장치와 같은 연산을 제공하게 된다[3].

FTL은 플래시메모리를 하드 디스크와 동일하게 동작하도록 설계된 시스템 소프트웨어로 파일시스템과 플래시메모리 장치 사이에 위치한다. FTL을 사용할 경우 디스크 기반으로 설계된 운영체제, 데이타베이스 시스템 등의 대부분의 소프트웨어를 별도의 수정 없이 디스크에서와 동일하게 사용할 수 있다. 그러나 대다수의 응용 소프트웨어에서와 달리 FTL을 통해 데이타베이스 시스템을 구현할 경우 플래시메모리 장점을 충분히 활용하지 못하고 좋지 못한 성능을 보이게 된다. 예를 들어 온라인 트랜잭션 처리 환경(OLTP)과 같은 일반적인 데이타베이스 시스템에서는 다른 소프트웨어와 달리 대용량의 데이타에 대해 매우 임의적인 위치에 대한 데이타 덮어쓰기를 유발한다. 또한 덮어쓰기의 크기가 매우 작기 때문에 덮어쓰기의 단위가 크고 순차적일 때 좋은 성능을 보이는 대부분의 FTL에서 심각한 성능 저하를 유발하게 된다. 즉, 하나의 덮어쓰기 연산은 최소 한 번의 지우기 연산과 다수의 쓰기 연산을 통해 수행하며 이때 덮어쓰기의 단위가 작을수록 불필요한 쓰기 연산이 증가한다. 따라서 FTL의 단위 쓰기 연산의 성능이 저하되어 빈번한 덮어쓰기를 유발하는 데이타베이스 시스템에서는 최악의 경우 하드 디스크를 사용하는 경우에 비해 좋지 못한 성능을 보이게 된다. 따라서 플래시메모리를 사용하는 데이타베이스 시스템에서 발생하는 성능저하를 해결하기 위해서는 플래시메모리에 적합한 데이타베이스 시스템의 저장 구조 및 알고리즘에 대한 연구가 필요하다.

본 논문에서는 최근 새롭게 제안된 In-Page Logging (IPL) 기법[4]을 소개하고 FTL 상의 데이타베이스 시스템과 IPL 기반의 데이타베이스 시스템간의 성능 평가를 수행한다. IPL 기법은 데이타베이스 시스템에서 발생하는 빈번한 덮어쓰기 연산을 적은 양의 로그 정보에 대한 쓰기 연산으로 변환함으로써 성능개선을 수행하는 데이타베이스 저장 기법이다.

성능 평가를 위해 최근 임베디드 데이타베이스 시스템 라이브러리로 널리 사용되는 오픈 소스 기반의 SQLite[5]를 사용한다. 본 논문에서는 FTL상에서의 성능과 IPL을 적용할 경우의 성능에 대한 평가를 수행하며, IPL을 적용 시 하나의 갱신 질의에 대해 약 30배의 성능향상이 발생함을 보인다.

논문의 구성은 다음과 같다. 2장에서는 플래시메모리의 특징과 기존의 플래시메모리 관련 연구들을 소개한다. 3장에서는 본 논문에서 평가하는 In-Page Logging 기법에 대해 소개한다. 4장에서는 IPL기법 구현에 사용한 데이타베이스인 SQLite에 대해 소개하고 5장에서 SQLite에 IPL기법을 응용하는 과정에 대해 기술하고 6장에서 성능평가를 위한 시뮬레이션 과정을 설명한다. 마지막으로 7장에서 결론과 향후 연구를 설명한다.

2. Flash Memory 기술 소개 및 관련연구

플래시메모리 기반에서 효율적으로 동작하는 데이타베이스 설계를 위해서는 플래시메모리의 동작 특성과 관련기술에 대한 이해가 필요하다. 이번 장에서는 플래시메모리에 대한 기본적인 특성에 대해 설명하고 플래시메모리를 기존의 디스크와 동일하게 작동하도록 하는 펌웨어인 FTL에 대해 소개하고 플래시메모리 데이타베이스 연구 분야 최근 기법들에 대해 소개한다.

2.1 Flash Memory

플래시메모리는 EEPROM, 디스크 등과 같이 전원이 꺼져도 기록된 데이타가 지워지지 않는 비휘발성 저장장치이다. 기계적인 장치를 사용하는 기존의 디스크와 달리 전기적인 신호에 의해 동작하기 때문에 기계적인 지연이 없다. 표 1은 플래시메모리와 하드 디스크의 성능을 보여주고 있다[4]. 표 1에서 알 수 있듯이 지연시간에서 플래시메모리는 하드 디스크에 비해 읽기의 경우 약 158배, 쓰기의 경우 약 68배의 뛰어난 성능을 지니며 이러한 빠른 연산속도로 인해 디스크를 대체할 수 있는 저장장치로 주목 받기 시작하였고 작은 크기와 강한 내구성, 저 전력 등의 장점으로 이미 임베디드 분야에서 플래시메모리는 하드디스크 보다 우위를 차지하고 있다.

플래시메모리는 제작 형태와 설계 방식에 따라 NOR형 플래시메모리와 NAND형 플래시메모리로 분류된다. NOR형의 경우 주로 코드 수행용으로 사용하며, NAND형 플래시메모리는 데이타 저장을 위해 사용한다. 또한 NAND형 플래시메모리는 소 블록(small block) 구조의 플래시메모리와 대 블록(large block) 구조의 플래시메모리로 분류가 가능하며 대 블록 구조가 소 블록 구조

표 1 접근 속도 : 하드 디스크 VS. NAND 플래시

저장 장치	동작 속도		
	읽기	쓰기	지우기
하드 디스크	12,700μs (2KB)	13,700μs (2KB)	N/A
NAND 플래시	80μs (2KB)	200μs (2KB)	1,500μs (128KB)

에 비해 대용량을 가질 수 있다. 현재 일반적으로 사용되는 플래시메모리는 NAND형의 대 블록 구조를 가진다. 따라서 본 논문에서는 대 블록 구조의 NAND형 플래시에 대해서만 고려하며 이후 플래시메모리는 NAND형 대용량 플래시메모리로 가정한다.

플래시메모리는 내부적으로 블록 단위로 나뉘며, 각 블록(128KB-64개 페이지)은 페이지 단위(2K-4섹터)로 구성되며 하나의 페이지는 4개의 섹터(512B)로 나뉜다. 블록은 지우기 연산의 기본 단위가 되며 페이지는 읽기와 쓰기 연산의 기본단위가 된다. 또한 쓰기 연산의 경우에는 섹터단위의 연산도 수행이 가능하며(Number Of Partial writes, NOP), 이는 플래시메모리의 종류에 따라 지원여부가 다르다.

이와 같이 플래시메모리의 기본 동작은 읽기/쓰기/지우기 연산으로 구분된다. 읽기 연산이 가장 빠른 속도로 수행되며 쓰기, 지우기 연산 순이다. 따라서 플래시메모리를 사용할 경우 지우기 연산의 횟수를 최소화 시키는 것이 중요하다. 디스크에서와 달리 지우기 연산이 필요한 이유는 플래시메모리에서는 덮어쓰기를 수행할 수 없기 때문이다. 즉 플래시메모리의 모든 쓰기 연산은 지우기 연산이 수행된 깨끗한 영역에 대해서만 가능하다(erase-before-write). 따라서 덮어쓰기 연산을 처리하기 위해서는 별도의 과정이 필요하다. 가장 기본적인 방법은 덮어쓰기가 수행될 블록을 메모리로 읽은 후 덮어쓰기를 수행하는 방법이다. 즉, 읽어드린 블록에 대해 메모리에서 덮어쓰기를 한 후 기존 블록을 지우고 메모리상의 변경된 블록을 다시 쓰는 방법이다. 이는 대블록 NAND 플래시의 경우 64번의 읽기, 1번의 지우기, 64번의 쓰기 연산을 유발한다. 또한 이러한 방법의 덮어쓰기 연산은 플래시의 지우기 연산을 수행한 이후 갑작스럽게 전원이 꺼지거나 메모리의 내용이 지워질 경우 블록의 일부 또는 전체의 데이타가 유실될 수 있다. 따라서 일반적으로 다음과 같은 과정으로 덮어쓰기 연산을 수행하며, 이는 128번의 읽기/쓰기와 2번의 지우기 연산이 필요하다.

1) 새로운 블록을 할당 받는다.
2) 덮어쓰기를 수행할 기존 블록을 할당된 새 블록으로 복사한다.(읽기&쓰기)
3) 기존 블록에 지우기 연산을 수행한다.
4) 변경될 페이지에 대해 쓰기 연산을 수행하고 나머지 페이지를 할당된 블록에서 복사한다. (읽기&쓰기)
5) 할당되었던 블록을 지운다.

플래시메모리는 덮어쓰기를 수행할 수 없는 문제 이외에도 각 블록에 대한 지우기 연산이 특정 횟수로 제한되는 문제점이 있다. 즉, 각 블록에 대한 지우기 연산은 정해진 횟수 이상을 수행할 경우 오류에 대해 보장하지 않는다. 따라서 플래시메모리에 대한 동작이 여러 블록에 최대한 균등한 형태로 수행(wear-leveling)되도록 해야 한다.

2.2 Flash Translation Layer (FTL)

하드 디스크를 직접 플래시메모리로 대체하기 위해서는 앞에서 언급한 덮어쓰기 연산에 대한 문제점을 해결해야 하며 파일 시스템을 사용하는 대부분의 소프트웨어의 경우 플래시메모리의 읽기/쓰기/지우기 연산을 덮어쓰기가 가능한 읽기/쓰기의 연산으로 변환 시켜주어야 한다. 이는 플래시메모리의 사용 용도와 응용 프로그램간의 동작 특징에 따라 해결 방법이 다를 수 있으며 일반적인 소프트웨어 동작 환경(운영체제/파일시스템)에서는 Flash Translation Layer(FTL)[3,6-8]라는 펌웨어를 통해 이를 해결한다. FTL의 또 다른 주요 역할은 플래시메모리에서 사용되는 연산단위를 하드 디스크의 연산단위로 변환 시켜주는 작업이다. 즉, 플래시메모리에서 사용되는 물리적인 연산 단위 주소(블록, 페이지)를 하드 디스크의 논리적인 연산 단위 주소(섹터)로 변환 시켜주는 역할을 수행함으로써 기존의 디스크 기반의 소프트웨어가 플래시메모리에서 동작하도록 처리하여 준다.

그림 1은 플래시메모리를 사용하는 Compact Flash (CF)카드의 내부 구조이다. 사용자는 CF카드의 인터페이스를 통해 논리 주소에 대한 데이타를 요청하고, CF카드의 동작 관리자(controller)는 적합한 데이타를 플래시메모리에서 읽고 반환해 준다. 이때 논리주소와 실제 물리주소간의 변환이 필요하며 ROM에 저장된 FTL에 의해 수행된다.

일반적인 응용 소프트웨어들의 경우 효율적인 FTL을

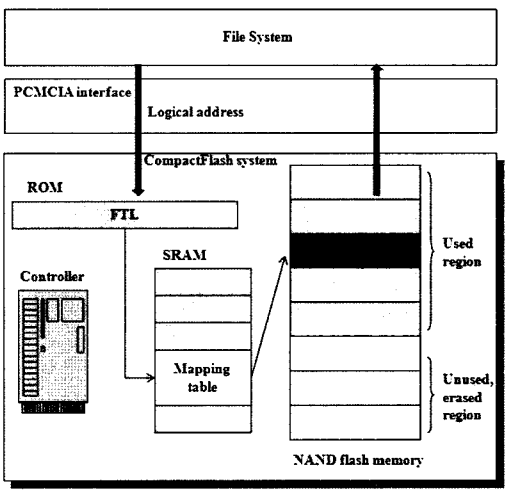

그림 1 Compact Flash Memory 구조

사용할 경우 플래시메모리의 제약 사항을 극복하고 디스크에 비해 뛰어난 성능을 보장하지만, 좋지 못한 FTL을 사용할 경우 오히려 디스크에 비해 취약할 수 있다. 효율적인 FTL이란 수행되는 소프트웨어에서 발생하는 쓰기 연산에 대한 처리과정에 따라 결정된다. 즉, 덮어쓰기를 유발하는 쓰기 연산에 대해 FTL이 어떠한 정책을 사용하는가에 따라 플래시메모리 저장장치의 전체 성능이 좌우된다. 따라서 대부분의 FTL의 경우 덮어쓰기 연산의 수를 최소화하기 위한 기법을 사용하며 주로 Log-structured File System[9]과 같이 로그 기반의 설계를 통해 수행한다. [7,8]은 대표적인 로그 기반의 FTL로 플래시메모리 상에 특정 블록을 로그 영역으로 사용하여 쓰기 연산 및 내부 갱신 연산(덮어쓰기 연산) 수행 시 변경될 데이타를 로그 영역에 기록함으로써 지우기 연산을 최대한 회피한다.

그러나 로그 기법을 적용할 경우에는 로그 영역과 기존 데이타 영역에 대한 병합 연산(merge)이라는 새로운 연산이 발생한다. 병합 연산이란 한정된 로그 영역에 더 이상 새로운 데이타를 기록할 수 없을 경우 발생한다. 병합 연산의 수행 과정은 먼저 기존의 데이타 블록의 과거 데이타와 로그 블록의 새로운 데이타를 병합하여 이를 새로운 데이타 블록에 기록하고 과거 데이타 블록과 사용된 로그 블록에 대한 지우기 연산을 수행하는 과정으로 이루어진다. 따라서 병합 연산은 최소 2번의 지우기 연산과 1블록에 대한 쓰기 연산을 유발하며 이는 로그 기법을 활용한 FTL에서 가장 큰 비용을 유발한다.

그림 1과 같이 대부분의 FTL은 파일시스템을 사용하는 환경에 적합한 구조로 설계된다. 따라서 상위 단계에서 발생하는 쓰기 연산에 논리적인 정보를 알 수 없는 구조를 가진다. 그러나 데이타베이스 시스템의 경우 쓰기 연산의 논리적인 요청(삽입/삭제/갱신)을 알 수 있으며 이러한 정보를 FTL상에서 활용이 가능하게 된다. 그러나 이러한 특징을 활용하기 위해서는 FTL에서 논리적인 요청에 대한 별도의 동작을 처리할 수 있어야 한다.

2.3 플래시메모리 데이타베이스 연구 분야

현재 플래시메모리를 활용한 데이타베이스 분야의 연구는 미미한 편이다. 이는 데이타베이스 연구 분야에서 아직 플래시메모리에 대한 이해가 부족하고, 파일시스템을 이용한 직접적인 구현이 어렵기 때문으로 분석된다. 플래시메모리 기반의 데이타베이스에 대한 대표적인 연구는 다음과 같다.

2.3.1 Btree Flash Translation Layer(BFTL)

Btree Flash Translation Layer[10]은 데이타베이스 시스템의 필수 요소인 B-Tree 인덱스를 플래시메모리 환경에서 적합하도록 설계된 기법이다.

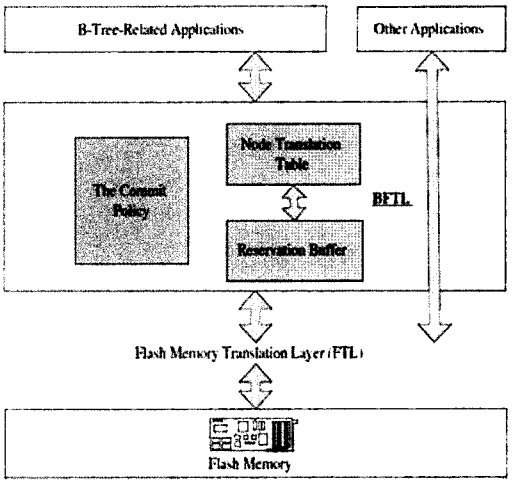

그림 2 BFTL의 기본 구조

BFTL은 플래시메모리의 빠른 읽기 동작의 횟수를 늘리고 쓰기 연산의 횟수를 줄임으로써 B-Tree 인덱스 동작의 성능을 높이는 기법이다. 그림 2와 같이 B-Tree 인덱스의 쓰기 연산을 수행할 경우 예약 버퍼(Reservation Buffer)를 통해 쓰기 연산의 단위를 높이고 하나의 물리적인 페이지에 여러 노드의 인덱스 유닛을 저장한다. 노드에 대한 읽기 동작은 노드 변환 테이블(Node Translation Table)을 활용하여 여러 물리적인 페이지에 흩어진 인덱스 유닛을 읽는다. 따라서 BFTL은 로그를 기반의 구조를 가지며 쓰기 연산에 대해 최적화된 기법이다.

그러나 예약 버퍼와 노드 변환 테이블이 시스템의 메모리에서 관리되고 있기 때문에 메모리의 효율을 떨어뜨리고 전원이 차단되는 상황에서는 메모리의 모든 내용이 지워지므로 복구가 불가능하게 된다. 또한, 실제적인 B-Tree가 메모리상에서 유지되기 때문에 부팅 시 플래시메모리의 모든 영역에 대한 읽기 연산을 수행하여야 하므로 부팅 시간이 길어지는 단점이 있다.

이러한 단점을 해결하기 위한 최근 연구로 [11]이 있으며, [11] 기법의 경우 BFTL과 달리 물리적인 주소 정보를 유지하여 전원 오프에 관한 문제점도 해결하고 있다. 그러나 센서환경과 같이 쓰기 연산이 읽기 연산보다 많을 경우에 대한 최적화를 수행하고 있으며, 따라서 읽기 연산에 대한 성능저하가 발생한다.

2.3.2 임베디드 환경을 위한 Flash 기반 DBMS

최근에 발표된 Flash 기반의 임베디드 DBMS로 LGeDBMS[12]와 Polyhedra[13]등이 있다. Polyhedra의 경우, 덮어쓰기 문제를 해결하기 위해서 쉐도우페이징(shadow paging)을 사용하고 있는데, 이 방법의 경우 고장 회복(recovery) 측면에서 문제점을 안고 있다

[14]. LGeDBMS의 경우, LFS[9]의 아이디어를 활용하여 논리 페이지 단위의 버전관리 기법(versioning scheme)을 사용한다. 이런 측면에서 Polyhedra와 비슷한 방법을 사용한다고 볼 수 있다. 하지만, 이 두 방법 모두 각 논리페이지의 쓰기 연산마다, 논리-물리 페이지 맵핑 테이블에 대한 쓰기 연산을 유발하게 된다. 그런데, 이 맵핑 정보의 변경은 결국 그 맵핑 정보를 포함하고 있는 블록에 대한 병합 연산을 필요로 하기 때문에 데이타블록에 대한 성능보다도 맵핑 정보의 변경이 더 성능에 문제가 된다. 또한, 논리적으로 하나의 테이블에 대한 연속된 논리 블록들이 여러 물리 블록에 흩어져 있고, 메모리 재활용(garbage collection)의 시점과 공간 낭비 등의 문제가 유발된다.

3. In-Page Logging (IPL)

일반적으로 디스크 기반의 데이타베이스 시스템에서 데이타베이스의 갱신 연산 (insert, update, delete)은 버퍼 관리자를 활용하여 임의의 위치에 대한 데이타 처리를 순차적인 위치에 대한 연산으로 변경하여 수행한다. 이는 임의의 위치에 대한 접근에 비해 순차적인 접근에 유리한 디스크의 특성을 활용하기 위함이다. 즉 디스크의 물리적인 기계장치의 움직임을 최소화하기 위한 방안으로 버퍼 관리자를 활용한다. 또한 하나의 레코드에 대한 갱신일 경우에 레코드가 포함된 데이타 페이지 전체(수 블록)에 대해 갱신을 수행하여 데이타 페이지 단위로 디스크에 갱신연산을 수행한다. 따라서 요청되는 갱신 연산의 범위가 임의적이고 여러 데이타 페이지에 흩어져서 저장된 레코드에 대한 요청일 경우 실제 변경되어야 할 데이타보다 훨씬 많은 양의 덮어쓰기 연산을 수행하여야 한다. 이러한 특징으로 인해 플래시메모리에 직접 디스크 기반의 데이타베이스 시스템을 구현할 경우 플래시메모리의 성능저하의 요인인 덮어쓰기 연산이 빈번하게 발생하게 된다.

본 장에서는 플래시메모리의 효율적인 데이타베이스 시스템 구현 방안인 In-Page Logging(IPL) 기법을 소개한다. IPL은 데이타베이스 시스템에서 발생하는 갱신 연산에 대해 변화를 로그로 표현하여 저장장치내의 변경 내용을 최소화시키고 페이지단위의 덮어쓰기 연산을 각 블록의 로그 영역에 대해 로그단위(섹터 최대 512B)의 붙여 쓰기 연산으로 대체하여 플래시메모리에 유리한 데이타베이스 저장 관리 기법이다.

3.1 IPL 기본 구조

그림 3은 IPL의 기본 구조를 나타낸다. 그림에서 데이타베이스에서 사용하는 최소 단위인 데이타 페이지는 8KB 단위로 관리하며, 데이타베이스의 버퍼영역에서 각 데이타 페이지는 로그 정보를 기록하는 512B 크기의 내부 로그 섹터를 가진다. 로그 정보는 갱신(insert, delete, update) 연산이 발생할 경우 기록되며 데이타 페이지에 대해 논리적인 의미를 가진 물리적(physiological) 로그[14] 형태로 최소한의 변경 사항을 저렴한 붙여 쓰기 연산으로 대체 수행 한다.

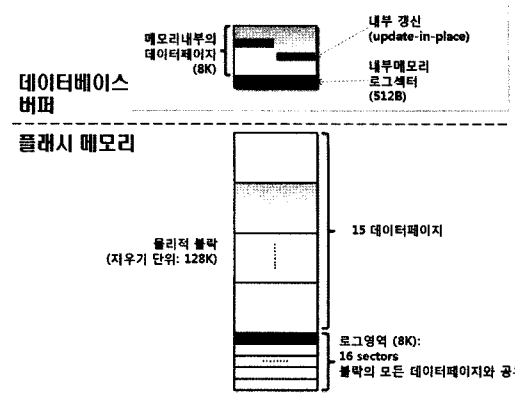

그림 3 IPL의 기본 구조

또한 플래시메모리내의 저장구조는 대 블록 구조의 NAND 플래시메모리의 경우 하나의 블록(128KB)을 기준으로 15개의 데이타 페이지와 1개의 로그 페이지로 나뉜다. 데이타 페이지와 로그페이지 모두 8KB의 크기를 가지며 로그페이지의 경우 같은 블록에 포함된 15개의 데이타 페이지가 공유하여 사용한다. 또한, 로그페이지에 대한 연산은 섹터단위로 수행되며, 이는 버퍼에 위치하는 개별 데이타 페이지에 해당되는 로그 섹터에 기록된 정보가 저장된다.

3.2 IPL 동작 특성

디스크 기반 데이타베이스의 경우 버퍼 관리자는 논리적인 데이타 페이지단위로 데이타를 관리한다[15]. 사용자의 요청에 대해 버퍼 관리자는 버퍼 내부에 요청된 데이타 페이지가 있는지 확인 하고 존재하지 않을 경우 저장 장치에서 해당 데이타 페이지를 읽어 들인다. 사용자 요청으로 데이타 페이지 변경이 발생할 경우에는 버퍼 내부에서 변경을 수행하고 페이지 교체정책 혹은 사용자의 완료 연산(commit)을 통해 저장 장치에 해당 데이타 페이지를 저장한다.

반면 IPL은 데이타 페이지의 변경이 발생할 경우 버퍼에 존재하는 로그섹터에 변경사항을 기록한 후 데이타 페이지의 변경을 수행하며 플래시메모리에 저장 시 로그섹터만을 로그페이지에 기록한다. 또한, 요청된 데이타 페이지를 버퍼로 읽을 경우 로그 페이지의 변경내역을 적용하여 최신의 데이타 페이지를 복원하여 버퍼에 위치시킨다.

그림 4는 IPL의 쓰기 연산 과정을 나타낸다. 사용자

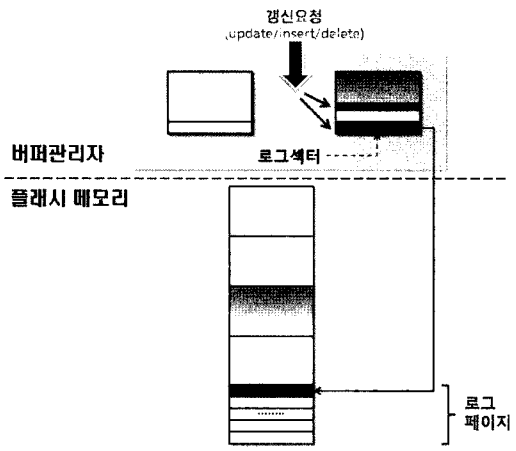

그림 4 IPL 동작 원리 : 쓰기 연산

에 의한 페이지 갱신 요청이 들어올 경우 버퍼관리자는 버퍼내의 로그 섹터에 변경 내용을 기록한다. 이때 논리적의미를 가진 물리적인 로그 형태로 최소한의 크기를 유지하여 기록한 후 데이타 페이지의 변경을 수행한다. 만약 로그 섹터가 꽉 차거나 완료 연산, 페이지 교체 정책 등이 발생할 경우 데이타 페이지에 대한 쓰기 연산을 수행하지 않고 로그 섹터에 기록된 로그 내용만 소속된 블록의 로그 영역에 기록한다. 이때 로그 정보는 512B단위의 섹터단위로 쓰기 연산을 수행하므로 하나의 블록(15개의 데이타 페이지)은 16번의 쓰기 연산을 수행 할 수 있다.

그림 5는 IPL의 읽기 연산의 수행 과정이다. 읽기 연산은 기본적으로 플래시메모리 상의 데이타 페이지를 버퍼로 읽어오는 과정으로 수행된다. 이때 데이타 페이지가 소속되어 있는 플래시메모리 상의 해당 블록의 로그페이지에서 현재 데이타 페이지에 대한 변경사항(로그) 유무를 확인하고 변경 사항이 있을 경우 기록된 변경사항을 데이타 페이지에 적용하여 최신의 데이타 페

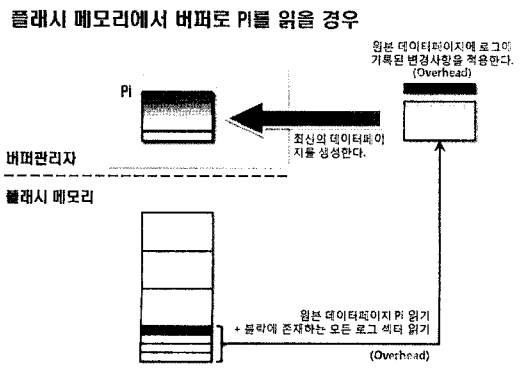

그림 5 IPL의 동작 원리 : 읽기 연산

이지를 버퍼에 위치시킨다. 읽기 연산의 경우 그림과 같이 매번 새롭게 페이지를 버퍼로 읽을 때 로그페이지에 대한 검색을 수행해야 하는 추가 연산 비용과 원본 데이타 페이지에 로그섹터의 변경사항을 적용해야 하는 오버헤드가 발생한다. 그러나 플래시메모리의 특성(빠른 읽기 속도, 80μs)을 고려할 경우 발생되는 추가 연산 비용이 성능에 영향을 미치지 않는다.

IPL은 다수의 덮어쓰기 연산을 로그에 대한 쓰기 연산으로 대체하므로 로그 기법을 기반으로 하고 있으며 읽기/쓰기 연산 이외의 병합 연산이 발생하게 된다. 즉, 만약 로그 페이지의 공간이 더 이상 존재 하지 않을 경우 병합 연산이 발생한다. 병합 연산의 경우 병합을 수행할 블록에 포함된 데이타 페이지에 대해 로그 정보를 적용하여 새로운 블록에 기록 한다 후 기존의 블록에 대한 지우기 연산을 수행하는 과정으로 이루어진다.

4. SQLite

이번 장에서는 IPL을 적용할 데이타베이스 시스템인 SQLite[5]에 대해 소개한다.

SQLite는 오픈 소스의 임베디드 SQL 데이타베이스 엔진으로 별도의 서버 구동 없이 동작이 가능한 임베디드 기반의 데이타베이스 관리 시스템이다. SQL92 표준을 대부분 지원하며 각 트랜잭션은 원자성(atomicity), 일관성(consistency), 격리성(isolation), 신뢰성(durability)을 보장한다. 전체 시스템의 구성은 그림 6과 같다.

SQLite는 동작특성에 따라 크게 SQL문을 해석하는 Parser 영역(SQL Complier), 각 실행 코드에 대한 실행 계획을 세우는 영역(Core), 버퍼관리자 및 저장 장치 관리자(Beckend) 영역으로 나뉜다. Parser 영역은 사용

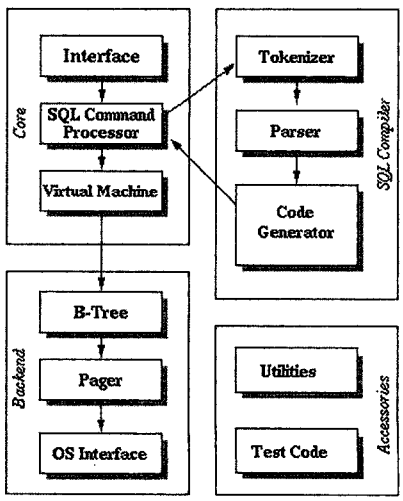

그림 6 SQLite의 기본 구조

자의 질의를 데이타베이스 엔진이 이해할 수 있는 명령어로 변경하는 작업을 수행하며 변경된 명령어를 활용하여 최적의 수행계획을 세우게 된다. 이때 저장 장치와의 동작은 저장관리자를 통해 수행되며 저장 장치를 통해 읽어드린 데이타는 버퍼관리자를 통해 관리된다.

버퍼관리자의 관리 기법은 저장 장치에 대한 쓰기연산 수행 정책에 따라 force/no-force 나뉜다[14]. Force 정책은 모든 완료 연산이 수행될 경우 버퍼의 존재하는 데이타 페이지에 대한 쓰기 연산이 수행된다. 반면 no-force 정책은 완료 연산이 발생되었을 경우라도 실제 저장 장치에 기록이 되지 않는 경우가 발생하며 만약 데이타베이스 복구를 수행할 경우 기록되지 않은 데이타를 기록하기 위해 리두 로그(redo log)를 기록하여야 한다.

또한, 버퍼의 페이지교체 정책에 따라 버퍼 관리 기법은 steal/no-steal로 나뉜다. steal을 사용할 경우 완료 연산 시점과 상관없이 페이지교체정책으로 인해 저장 장치에 쓰기 연산이 수행된다. 반면 no-steal의 경우 버퍼의 데이타 페이지는 오직 완료 연산이 발생할 경우에만 쓰기 연산이 수행된다. 따라서 수행 취소(rollback) 연산이 발생할 경우 이전 데이타 페이지에 대한 복구를 위해 언두 로그(undo log)를 사용하여야 한다.

SQLite의 경우 임베디드 환경에 적합한 데이타베이스로 force와 no-steal정책을 사용한 버퍼 관리기법을 사용한다. 따라서 언두 로그와 리두 로그에 대한 관리를 별도로 수행하지 않는다. 또한 SQLite의 완료 연산 수행 시 신뢰성을 보장하기 위한 방법으로 저널 파일을 사용한다. 저널 파일은 변화가 발생하는 논리 페이지에 대한 원본 데이타를 임시로 저장하여 수행 취소 연산 혹은 시스템 완료 연산 시 발생하는 시스템 고장에 대한 복구를 위해 사용되는 파일이다. 다음은 저널 파일을 사용하고 force, no-steal 정책을 사용하는 SQLite의 완료 연산의 수행 과정이다.

1) 격리성을 보장하기 위해 데이타베이스(파일)를 잠근다.
2) rollback 저널 파일을 생성한다.
3) 변화가 발생한 데이타 페이지들(dirty 페이지)을 저널파일에 복사한다.
4) 데이타베이스(파일)에 변경 내용을 쓴다.
5) 생성된 rollback 저널파일을 삭제한다.
6) 데이타베이스(파일)의 잠금을 해제한다.

SQLite에서는 변경된 데이타에 대한 수행 취소 연산이나 복구를 위해 저널파일을 사용하고 있으나 앞장에서 언급한 IPL의 저장 기법을 활용할 경우 기존의 데이타가 보전되므로 별도의 저널 파일에 대한 연산이 불필요하게 되며 플래시메모리 상에서 저널 파일 관리를 위한 성능저하를 개선할 수 있게 된다.

5. SQLite에 IPL 기법 응용

IPL 기법은 3장에서 설명한 것과 같이 IPL기법을 적용하기 위해서는 논리적인 저장 관리 기법과 물리적인 플래시메모리의 동작에 대한 두 부분의 설계가 동시에 이루어지어야 한다. 그러나 대부분의 플래시메모리 저장장치(USB저장 장치, SD 카드, CF 메모리 등...)는 이미 FTL이 적용된 일반적인 블록 장치로서 인식이 된다. 따라서 IPL 기법을 완벽하게 구현하기 위해서는 개발보드 상의 내장 플래시메모리에 대한 동작을 직접 수정하여야 하거나 플래시메모리 저장장치의 FTL을 새롭게 수정하여야 한다. 본 장에서는 직접 플래시메모리에 대한 연산이 가능한 개발보드상의 적용과정과 플래시메모리 저장 장치에서의 성능평가의 문제점에 대해 언급하고 본 논문에서 SQLite에 IPL 기법을 응용한 과정에 대해 설명한다.

5.1 SQLite에 IPL 기법 응용 방안

내장 플래시메모리 기반에서 SQLite에 IPL기법을 적용하기 위해서는 운영체제의 장치 관리자(device driver), SQLite의 저장 관리자에 대한 재 설계가 필요하다. 일반적인 운영체제의 경우 블록 장치에 대한 읽기/쓰기 연산은 블록 장치 관리자에 등록된 동작 정의를 통해 수행된다. 따라서 읽기/쓰기 외에 지우기 연산을 사용하는 IPL의 경우 지우기 연산을 정의하는 장치 관리자에 대한 수정이 필요하다. 또한 장치 관리자의 경우 운영체제의 버퍼 캐시를 사용하기 때문에 버퍼 캐시 정책에 대한 정의도 새롭게 수행해야 한다. SQLite의 저장 관리자 측면에서 수정사항은 우선 파일 기반의 동작 대신 직접 블록 장치에 대한 연산을 수행할 수 있게 수정 되어야 한다. 즉 하나의 파일이 하나의 데이타베이스로 사용하는 방식 대신 플래시메모리의 물리적 주소에 대한 파티션을 통해 데이타베이스를 관리하는 방식으로 설계되어야 한다. 내장 플래시메모리를 활용할 경우의 또 다른 고려사항은 개발 보드내의 데이타 전송의 성능에 대한 고려를 하여야 한다. NAND 플래시메모리의 경우 물리적인 단계에서 일반적으로 8bit 단위로 주소 및 데이타 정보를 주고받는다. 따라서 개발 보드내의 데이타 버스의 대역폭 및 전송 속도에 따라 플래시메모리의 성능이 좌우되기도 한다.

USB 스틱, CF 카드등과 같은 플래시메모리 저장 장치를 사용할 경우는 FTL을 통해 플래시메모리의 모든 동작이 이루어짐으로 각 동작은 읽기/쓰기의 두 가지만을 지원하게 된다. 따라서 플래시메모리 자체의 특성 및 사용자 프로그램의 효율적인 설계 뿐만 아니라 FTL의 설계방식에 따라 성능이 좌우되며, FTL에 의해 플래시메모리 내부의 동작이 감춰지므로 명확한 성능평가를

수행하기 힘들다. 2장에서 언급한 BFTL[10]과 Polyhedra[13]의 경우 이러한 환경에서 개발된 형태로 플래시메모리의 특성을 충분히 활용하지 못하며 FTL에 의존적인 성능을 보이게 되므로 성능평가의 기준이 될 수 없다. 또한 LGeDBMS[12]의 경우 휴대전화에 적용된 데이타베이스로 FTL 및 내부 저장 장치의 성능에 의존적으로 동작하게 될 것이다.

5.2 성능 평가를 위한 실험 과정

앞서 살펴본 바와 같이 IPL의 동작에 대한 성능평가를 위해서 본 논문에서는 내장된 플래시메모리를 사용하는 환경을 가정하고 IPL 기법을 SQLite의 버퍼 관리자와 저장 관리자의 읽기/쓰기 동작에 대해 적용한다. 적용된 SQLite는 실험 질의를 수행한 후 저장 장치 관리자 측면의 동작(읽기/쓰기/지우기)을 수집하여 이에 대한 성능평가를 시뮬레이션 한다. 또한 수정 되지 않는 SQLite는 디스크 환경에서 실험 질의를 수행하고 저장 관리자 계층에서 동작(읽기/쓰기)을 수집한 후 기본적인 기능을 수행하는 FTL상의 성능평가를 시뮬레이션 하여 이 둘의 성능을 비교 분석 한다.

그림 7과 그림 8은 SQLite에 IPL을 적용할 경우(SQLite-IPL)의 질의 처리 과정으로 그림 7은 삽입(insert) / 수정(update) / 삭제(delete) 질의에 대한 과정이고 그림 8은 검색(select) 질의에 대한 과정이다.

그림 7과 같이 갱신 연산에 대한 요청이 발생할 경우 Pager 계층에서는 변경에 대한 로그를 IPL 관리자로 전달하고, Pager내의 데이타 페이지를 수정한다. IPL 관리자는 전달받은 섹터단위의 로그를 실제 플래시메모리의 로그영역에 기록한다.

SQLite의 경우 갱신 연산이 발생할 경우 저널 파일을 생성하여 기존의 데이타페이지의 복사본을 저장한다. 시뮬레이션을 위한 SQLite-IPL의 경우 갱신 연산이 발생할 경우 저널 파일 대신 각 블록에 대한 로그 파일을 생성한 후 그림 7의 로그 섹터의 쓰기 연산에 대한 로그를 블록에 대한 로그 파일에 붙여 쓰기를 수행한다.

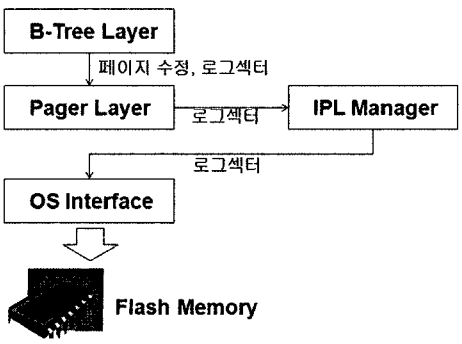

그림 7 SQLite-IPL의 Update 연산

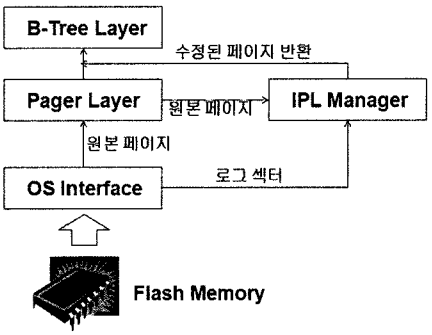

그림 8 SQLite-IPL의 Select 연산

이때 각 블록의 로그 정보에 대해 16번째 마다 병합 연산을 위한 사항을 기록 한다.

검색 질의인 경우 그림 8과 같이 플래시메모리에 저장된 원본 데이타 페이지를 Pager 계층으로 전달하고 로그섹터를 IPL 관리자로 전달한다. 또한 Pager 계층은 전달받은 원본 데이타 페이지를 IPL 관리자에게 전달한다. IPL 관리자는 원본 데이타 페이지에 로그 섹터의 변경내용을 적용하여 상위 계층으로 수정된 데이타 페이지를 반환하는 과정으로 수행된다. 시뮬레이션을 위한 SQLite-IPL에서는 데이타페이지의 읽기와 함께 데이타 페이지가 소속된 블록 로그 파일을 동시에 읽고 읽어드린 데이타페이지에 블록 로그에 저장된 로그들을 적용하는 과정으로 시뮬레이션을 수행한다.

시뮬레이션을 위한 실험 데이타는 다음과 같이 설정하고 각 레코드들의 키는 1~1000000 범위의 임의의 정수를 가지며 값은 130byte의 임의의 문자열을 가진다. 실험은 1000000건의 레코드를 먼저 SQLite와 SQLite-IPL에 저장한 후 임의의 레코드에 대한 검색 연산을 각 1000번, 임의의 레코드에 대한 수정 연산을 각 1000번씩 수행하였다. 실험에 대한 결과와 분석은 다음 장에 설명한다.

Data Page : 8KB Total row : 1000000건 Total Page : 2329 MAX ROW in a Page :59

[실험 데이타 테이블 정보]

6. 시뮬레이션을 통한 성능 평가

6.1 실험 결과

실험은 SQL로 작성된 수정 질의와 검색 질의에 대해 수행한다. 각 질의는 모두 1000번씩 수행하며 각 질의는 임의의 페이지 요청을 발생시킨다. 또한, 수정 질의의 경우 매 건마다 완료 연산이 발생하도록 하여 디스크의 입/

출력을 유발하고 실제 발생하는 입/출력의 횟수를 구하였다.

표 2 1000번의 수정 질의 수행 결과

	읽기 연산	쓰기 연산	쓰기연산 (JOURNAL)	쓰기연산 (LOG)
SQLite	4002	2000	2000	N/A
SQLite-IPL	4563	N/A	N/A	1000

표 2는 수정 질의에 대한 실험 결과이다. 위에서 알 수 있듯이 수정 질의 시에도 해당되는 페이지를 찾기 위해 읽기 연산이 발생하였다. 순수 SQLite에서 1000개의 데이타 페이지에 대한 수정 질의 임에도 4002번의 읽기 연산이 발생한 이유는 SQLite의 테이블이 B-Tree 형식으로 구현되므로 말단 노드를 찾기 위한 읽기 연산으로 해석된다. 쓰기 연산의 경우 이전 장에서 설명한 완료 연산 정책으로 인해 수정 질의 횟수 보다 많은 쓰기 연산이 발생하였다. 1000건의 연산에 대해 각 데이타 페이지의 헤더 수정을 위한 1000번의 쓰기와 실제 데이타 페이지에 대한 1000번의 쓰기 연산이 발생하였다. 또한, 저널파일에 대한 쓰기 연산도 발생하였으며 최초 저널 파일의 생성 시 1000번의 쓰기 연산이 발생하였고 원본 페이지에 대한 복사를 위해 1000번의 쓰기 연산이 추가로 발생하였다.

SQLite-IPL의 경우 SQLite와 같이 갱신될 데이타 페이지의 읽기 연산이 4002번 발생하였고, 로그섹터에 대한 읽기 연산이 추가로 561번 발생하였다. 반면 쓰기 연산의 경우 각 수정 질의에서 발생시킨 로그섹터에 기록 횟수로 1000번의 쓰기 연산만 발생하였다.

표 3 1000번의 검색 질의 수행 결과

	읽기연산	읽기연산(LOG)
SQLite	4002	N/A
SQLite-IPL	4002	356

표 3은 검색 질의에 대한 읽기 연산 횟수이다. 검색 질의 역시 임의의 레코드에 대해 1000번을 수행하였다.

SQLite와 SQLite-IPL 모두 데이타 페이지에 대한 읽기 횟수는 같고 SQLite-IPL의 경우 로그영역에 대한 읽기가 356번 추가로 발생하였다. 이는 총 356개의 로그영역에 로그정보가 기록되어있음을 의미한다.

실험 결과에서 알 수 있듯이 IPL을 적용하지 않은 경우와 IPL을 적용한 경우의 입/출력 횟수가 크게 차이가 발생함을 알 수 있었다. 이러한 결과만으로도 IPL을 적용할 경우의 효율적임을 알 수 있다. 다음 절에서는 수정 질의에 대한 시뮬레이션을 수행하여 플래시메모리 환경에서의 성능을 분석한다.

6.2 성능 평가

본 논문의 목적은 기존의 FTL을 이용하는 플래시메모리 환경과 IPL을 사용하는 환경에 대한 성능 평가를 수행하는 것이다. 본 절에서는 실험에서 수행한 수정 질의에 대해 FTL을 사용한 경우와 IPL을 사용한 경우에 대한 시뮬레이션을 수행 하고 이를 분석해 본다.

FTL을 통해 한 번의 버퍼 상에 존재한 데이타 페이지에 대한 수정 질의 비용은 다음과 같다.

```
수정 질의 비용
= 데이타 페이지 읽기 비용(물리적 : 4 페이지)
+ 저널 페이지 쓰기 비용(물리적 : 8 페이지)
+ 데이타 페이지 갱신 비용(물리적 : 4 페이지)
+ 메타 페이지 갱신 비용(물리적 : 4 페이지)
+ 저널 페이지 삭제 비용(물리적 : 1 블록)
```

[FTL을 사용한 SQLite 수정 질의 수행비용]

저널 페이지의 쓰기의 경우 최초 저널 생성 시 저널 헤더를 위해 한 개의 데이타 페이지 공간 사용하므로 실질적으로 2개의 페이지에 대해 쓰기 연산이 발생한다. 또한 사용이 완료된 저널 페이지는 지우기 연산을 수행하므로 플래시의 지우기 연산인 블록 연산이 발생하며 데이타 페이지 및 메타 페이지의 갱신 연산은 다음과 같은 비용이 발생한다.

```
데이타 페이지 갱신 비용
= 기존 블록을 새로운 블록으로 복사
(물리적 : 60 페이지 읽기 및 쓰기)
+ 변경된 페이지 쓰기 (물리적 : 4 페이지)
+ 기존 블록 삭제 (물리적 : 1블록)
+ 주소 맵핑 정보 변경 (α)
= 60*80μs + 60*200μs + 4*200μs + 1.5ms + α =
  19.1ms + α
```

[페이지 갱신비용]

위의 식에서 맵핑 정보 변경은 기존의 블록의 주소를 새로운 블록으로 바꾸기 위해 발생하며, 대부분의 FTL의 경우 맵핑 정보 역시 플래시메모리 상에서 관리한다. 따라서 맵핑 정보 변경 역시 하나의 갱신 연산으로 볼 수 있으며 IPL에서 역시 주소 맵핑을 위한 동작이 발생하므로 무시하도록 한다.

위에서 계산한 갱신비용을 적용하여 하나의 수정 질의 수행에 대한 비용은 다음과 같이 41.62ms이 된다.

```
수정 질의 비용
= 4 * 80 + 8 * 200 + 19.1ms + 19.1ms + 1.5ms
= 41.62ms
```

[SQLite의 수정 질의 수행비용]

반면 IPL의 수정 질의 수행비용은 다음과 같다.

```
• 수정 질의 수행비용
= 해당 로그 영역에 로그 정보를 기록 (물리적 : 1섹터)
+ 병합 연산 비용 * 1/16

• 병합 연산 비용
= 기존 블록의 데이타 페이지 읽기(물리적: 60 페이지)
+ 로그 페이지 읽기 (물리적 : 4페이지)
+ 새로운 블록에 데이타 페이지 쓰기(물리적: 60페이지)
+ 기존 블록 삭제 (물리적 : 1블록)
= 60 * 80 μs + 4 * 80 μs + 60 * 200 μs + 1.5ms
= 18.62ms

• 수정 질의 수행비용
= 200 μs + 18.62 ms * 1/16 = 1.363ms
```

[SQLite-IPL의 수정 질의 수행비용]

IPL을 사용할 경우 로그페이지에 로그섹터가 16개까지 들어가므로 병합 연산은 데이타 페이지에 대해 17번째 덮어쓰기 연산이 수행될 경우 발생한다. 따라서 하나의 수정 질의에서는 1/16의 확률로 발생하며 계산 결과와 같이 연산 비용은 1.363ms이 된다.

이와 같이 플래시메모리 상에서 데이타베이스를 수행할 경우 덮어쓰기 연산을 유발하는 수정 질의에 대한 성능은 약 30배의 성능차이를 보이며, 주소 맵핑을 위한 덮어쓰기 연산을 고려할 경우 더욱 큰 차이를 보이게 된다.

따라서 플래시메모리 기반의 데이타베이스 시스템을 위해서는 별도의 저장 구조와 알고리즘이 필요함을 알 수 있었고 IPL 기법이 효율적으로 동작함을 알 수 있었다.

7. 결론 및 향후 연구

본 논문에서는 플래시메모리에 적합한 데이타베이스 시스템 구현을 위한 IPL 기법을 소개하고 성능 평가를 수행하였다. 일반적인 소프트웨어에 적합한 FTL기법에 비해 IPL을 사용할 경우 하나의 갱신 연산에 대해 약 30배의 성능 향상이 있음을 알 수 있었다. 따라서 플래시메모리의 뛰어난 성능을 효과적으로 이용하기 위해서는 데이타베이스 시스템에 적합한 IPL 기법을 사용하는 것이 보다 효율적임을 알 수 있다.

향후 연구로는 IPL에서 사용하는 트랜잭션 관리 기법과 복구 기법에 대한 성능평가를 수행할 것이다. 또한, 로그영역에 기록되는 논리적 의미를 가진 물리적 로그 정보에 대한 최적화 방안에 대한 연구도 수행할 것이다.

참 고 문 헌

[1] Linda Dailey Paulson. "Will Hard Drives Finally Stop Shrinking?" IEEE Computer, 38(5):14-16, May 2005.

[2] Mark Hachman. New Samsung Notebook Replaces Hard Drive With Flash. http://www.extremetech.com, May 2006.

[3] Intel. Understanding the Flash Translation Layer (FTL) Specification. Application Note AP-684, Intel Corporation, December 1998.

[4] Sang-Won Lee, Bongki Moon, "Design of Flash-Based DBMS: An In-Page Logging Approach," ACM SIGMOD, June 2007.

[5] SQLite, http://www.sqlite.org

[6] T. S. Chung, D. J. Park, S. W. Park, D. H. Lee, S. W. Lee, and H. J. Song, "System Software for Flash Memory: A Survey," EUC 2006, August 2006.

[7] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho, "A Space-Efficient Flash Translation Layer for CompactFlash Systems," IEEE Transactions on Consumer Electronics, 48(2):366 - 375, May 2002.

[8] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sang-Won Park, and Ha-Joo Song, "A Log Buffer based Flash Translation Layer using Fully Associative Sector Translation," ACM Transactions on Embedded Computing Systems, 6(3): Article 18, July 2007.

[9] Mendel Rosenblum, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, 10(1):26-52, February 1992.

[10] C. H. Wu, L. P. Chang, and T. W. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," RTCSA 2003, pp. 409-430, 2004.

[11] Suman Nath, Aman Kansal, "FlashDB: Dynamic Self-tuning Database for NAND Flash," IPSN 2007, April 2007.

[12] Gye-Jeong Kim, Seung-Cheon Baek, Hyun-Sook Lee, Han-Deok Lee, and Moon Jeung Joe. "LGe-DBMS: A Small DBMS for Embedded System with Flash Memory," In Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006.

[13] ENEA, Ploynedra FlashLite Datasheet, http://www.enea.com, 2006.

[14] C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, and Peter M. Schwarz. "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM Transactions on Database Systems, 17(1):94 - 162, 1992.

[15] R. Ramakrishnan, J. Gehrke, "Database Management Systems, 3rd Ed.," McGraw-Hill, 2004.

나 갑 주

2003년 한국항공대학교 항공전자공학과(학사). 2006년 성균관대학교 컴퓨터공학과(석사). 2006년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정. 관심분야는 flash memory DBMS, index structure

김 상 우

2007년 부경대학교(학사). 2007년~현재 성균관대학교 전자전기컴퓨터공학과 석사과정. 관심분야는 flash memory DBMS, storage system

김 재 명

2006년 성균관대학교(학사). 2008년 성균관대학교(석사). 2008년~현재 (주)알티베이스 연구개발본부 연구원. 관심분야는 데이타베이스 시스템 구조 및 개발, 플래시 메모리

이 상 원

1991년 서울대학교 컴퓨터학과(학사). 1994년 서울대학교 컴퓨터학과(석사). 1999년 서울대학교 컴퓨터학과(박사). 1999년~2001년 한국 오라클. 2001년~2002년 이화여대 BK21 계약교수. 2002년~현재 성균관대학교 정보통신공학부 부교수. 관심분야는 flash memory DBMS