공 학 석 사 학 위 논 문

플래시 메모리 기반 임베디드 데이터베이스 시스템을 위한 지연쓰기 기법

*2007*년 *8*월

부 경 대 학 교 대 학 원

컴 퓨 터 공 학 과

윤 승 희

공 학 석 사 학 위 논 문

플래시 메모리 기반 임베디드 데이터베이스 시스템을 위한 지연쓰기 기법

지 도 교 수 송 하 주

이 論文을 工學碩士 學位論文으로 提出함

*2007*년 *8*월

부 경 대 학 교 대 학 원

컴 퓨 터 공 학 과

윤 승 희

이 論文을 尹 勝 熙의 工學碩士

學位論文으로 認准함

2007년 8월

주 심 공학박사 조 우 현

위 원 공학박사 권 오 흠

위 원 공학박사 송 하 주

목 차

그 림 목 차

표 목 차

플래시 메모리 기반 임베디드 데이터베이스 시스템을 위한 지연쓰기 기법

윤 승 희

부 경 대 학 교 대 학 원 컴 퓨 터 공 학 과

요 약

플래시 메모리는 동작 특성상 메모리 영역에 대한 덮어쓰기(overwrite)가 불가능하고 플래시 메모리에 쓰기를 하기 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 연구에서는 플래시 메모리에 대한 삭제 횟수를 줄이기 위해 데이터베이스 페이지에 대한 쓰기 연산을 지연하는 지연쓰기 기법을 제안한다.

지연쓰기 기법은 페이지에 대한 갱신이 일어날 때 페이지 버퍼 내의 해당 페이지에 대해서는 갱신을 수행하되 그것을 유발한 레코드 연산(레코드 삽입, 갱신, 삭제)은 별도의 지연쓰기 버퍼에 기록한다. 그리고 레코드 연산이 지연쓰기 버퍼에 저장되어 있는 동안에는 해당 페이지에 대한 갱신은 보류한다.

만약 해당 페이지를 다시 읽어야 할 필요가 있을 경우에는 지연쓰기 버퍼에 저장된 갱신 정보와 병합하여 갱신된 페이지를 페이지 버퍼에 적재한다.

이는 갱신되는 페이지의 개수와 단일 페이지에 대한 갱신 횟수를 감소시키는 효과를 가져온다. 따라서 플래시 메모리의 삭제 및 쓰기 연산을 감소시켜 데이터베이스 시스템의 수행 성능을 향상 시키게 된다.

Delayed Write Scheme for The Flash Memory Based Embedded Database Systems

Seung Hee Yoon

Department of Computer Engineering Graduate School,

Pukyong National University

Abstract

It is impossible to overwrite a data block in flash memory, and an erase operation have to be performed in advance to write data on a flash memory. Because on erase operation takes a lot of time compared to an read operation, minimizing erase operation is crucial to enhance overall execution performance of the flash memory. In this paper, we propose a delayed write technique that reduces the number of erase operations by delaying write operations.

When a data page is to be modified, proposed scheme updates the data page in page buffer, and keeps the operation (update, insert, delete) records that has caused the modification in a separate storage called delayed write buffer.

The persistent data page is not modified while its operation records are kept in the delayed write buffer.

Later in time, if the modified page has to be loaded into buffer, the persistent data page is loaded into the buffer and merged with the operation records in the delayed write buffer.

By doing so, proposed scheme reduces number of modified pages and write operations on a page. Therefore, it reduces write and erase operations, and enhances overall performance of a flash memory based database system.

Ⅰ. 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 증가 하고 있다.

플래시 메모리는 가볍고 물리적인 충격에 강해 휴대가 용이하며 저전력으로 동작해 배터리 소모량을 줄이기 때문에 하드 디스크를 대체할 매체로 각광받고 있다. 하지만 플래시 메모리는 빠른 읽기 연산 속도에 비해 쓰기 연산 속도[1, 2]가 다른 메모리보다 느리고 하드 디스크처럼 덮어쓰기(overwrite) 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 블록 교체 기법을 적용하여 사용자에게 비교적 수행 시간이 오래 걸리는 삭제 연산에 대하여 이미 삭제 연산을 수행한 빈 블록으로 재사상(remapping)함으로써 플래시 메모리를 기존의 하드 디스크와 같이

사용할 수 있게 한다. 그러나 FTL을 이용하여 덮어쓰기(overwrite)를 지원하는 것은 기존 운영체제 또는 데이터베이스 시스템에 플래시 메모리를 바로 사용할 수 있다는 장점이 있지만, 플래시 메모리의 특성들을 제대로 반영하지 못하고 비정상적인 전원 차단에 대해 신뢰 있는 저장을 보장하지 못한다[3, 5, 13].

이 문제점을 개선한 것이 로그 구조 파일시스템(LFS)[6]을 기반으로 하는 JFFS/ JFFS2[3], YAFFS와 같은 새로운 파일시스템이 제안되었다.

플래시 메모리는 EEPROM(Electrically Erasable Programmable Read Only Memory)의 한 종류로써, 쓰기 전 삭제 연산을 실행해야 한다.

위에서 언급했듯이 연산은 플래시 메모리에서 가장 많은 비용 지출을 요구한다. 지연쓰기 기법을 통해 삭제 연산을 최대한 줄임으로써, 얼마나 향상된 결과를 가져오는지 알아볼 것이다.

2장에서는 플래시 메모리의 주요 개념 및 제안한 기법과 관련된 주요 개념들을 살펴보고, 3장에서는 제안하는 지연쓰기 기법에 대해 상세히 기술한다. 4장에서는 지연쓰기 기법에 대한 구현 및 결과에 대해 분석하며 5장에서는 본 논문의 결론을 맺는다.

Ⅱ. 관련 연구

2.1 플래시 메모리 특성

플래시 메모리는 높은 신뢰성과 집적도를 가진 반도체 형태의 저장 장치로서 비용이 기타 저장 장치에 비해 상대적으로 저렴하여 PDA, 휴대폰과 같은 임베디드 시스템에서 저장 장치로 널리 이용되고 있다. 플래시 메모리는 전원이 끊어져도 데이터가 지워지지 않은 비휘발성 메모리로서 블록 단위로 내용을 지울 수 있고, 다시 프로그래밍 할 수도 있다.

[표 1] 플래시 메모리와 다른 저장 장치와의 비교

종 류	장 점	단 점
DRAM	- 읽기 쓰기가 가능하다	- CPU의 도움 없이 데이터의 영구적인 보관이 어렵다
Flash Memory	- 영구적이고 지속적인 읽고 쓰기가 가능하다	- 쓰기 속력이 느리다 - 쓰기 횟수가 제한적이다
하드 디스크	- 기억 단위 당 가격이 저렴하다 - 호환성이 뛰어나다	- 부피와 소모 전력이 크다 - IDE 인터페이스가 필요하다

표 1에서처럼 임베디드 시스템에서는 소모 전력, 크기 문제로 인하여 DRAM과 플래시 메모리를 기억장치로 가장 많이 사용하고 있다[9].

플래시 메모리는 EEPROM(Electrically Erasable Programmable ROM)의 한 종류로

전원 공급이 없어도 데이터가 보관되는 비휘발성 메모리이다. 하지만 바이트 단위로 지우고 수정할 수 있는 EEPROM과는 다르게 블록 단위로 수정이 되기 때문에 속도가 빠르다. 하지만 플래시 메모리를 기존 데이터가 존재하는 위치에 데이터를 기록하려면 해당 블록을 쓰기 전에 지우기 작업을 해야 한다. 그리고 플래시 메모리의 전기적인 하드웨어 특성 상 플래시 메모리의 블록을 지우는 횟수에 제한이 있어 특정 블록을 고정해 쓰게 되면 플래시 메모리의 수명이 단축된다. 현재 일반적으로 사용되는 플래시 메모리는 NOR형, NAND형 두 가지로 구분한다.

[표 2] 플래시 메모리 특성

	NOR	NAND
직접도	낮음	높음
접근 방식	낮음	높음
쓰기 단위	임의 접근 가능	페이지 단위 접근
읽기 속도	150ns/1Byte 15us/512Byte	10us/1Byte 36us/512Byte
쓰기 속도	211us/1Byte 3.5ms/512Byte	226us/1Byte 266us/512Byte
블록 삭제 속도	1.2/128KByte	2ms/16KByte

NOR형 플래시 메모리는 시스템 버스에 바로 붙일 수 있으며 바이트 단위로 프로그래밍 할 수 있기 때문에 일반적인 램(RAM)처럼 사용할 수 있다. 하지만 NAND형의 플래시 메모리는 바이트 단위의 프로그래밍이 불가능하고 시스템 버스에 바로 붙이더라도 NAND형의 플래시 메모리를 위한 별도의 제어 로직이 필요하다. 두 형태의 플래시 메모리의 공통적인 중요한 특성은 블록 삭제를 실행하면 블록 내의 각 셀 또는 비트가 1로 초기화 되고, 데이터를 쓰는 과정을 통해서 필요한 비트가 0으로 전환된다는 것이다. 또한 삭제 연산은 쓰기 연산의 단위인 페이지보다 큰 블록 단위로만 이루어지게 된다. 따라서 플래시 메모리의 블록의 1 비트만을 수정한다고 해도 해당 블록 전체를 지우는 작업이 필요하다. 또한, 플래시 메모리의 수명은 삭제 주기(erase cycle)로 결정되는데, 평균적으로 10,000회의 삭제 주기를 가진다.

플래시 메모리에서는 특정 블록에 지우기 작업이 집중되어 해당 블록만이 수명이 다하는 것을 방지하기 위하여, 전체 블록에 삭제 주기가 고루 분산되도록 관리되어야 하는데 이러한 과정을 마모도 평준화(wear leveling)이라고 한다.
NOR형 플래시 메모리는 SDRAM과 같은 일반적인 메모리처럼 시스템의 주소 버스와 데이터 버스에 바로 연결될 수 있는 버스 형태의 외부 인터페이스를 가진다. 따라서 보통의 메모리와 같은 방식으로 직접 액세스(access)가 가능하며 코드를 직접 실행하는 것도 가능하다. 또한 바이트 단위 프로그래밍이 가능하며 빠른 속도의 임의 접근이 가능하므로 시스템에서 주로 코드를 저장하고 실행하는 용도로 사용된다[10].

NAND형 플래시 메모리는 NOR형 플래시 메모리와 마찬가지로 삭제 연산의 기본 단위인 블록으로 나누어진다. 블록은 페이지의 집합이며 크기는 모두 같다. 블록들은 읽기/쓰기 연산의 기본단위인 페이지로 구성된다. 각 페이지 마다 8, 16, 64 바이트의 잉여 영역을 가지는데, 이 잉여 영역(space area)은 많은 응용에서 메타데이터(metadata)나 에러 정정 코드(error correction code, ECC)를 저장하는데 사용된다.

[표 3] NOR, NAND 플래시 메모리의 장단점

	NOR	NAND
장 점	- 빠른 임의 접근 - 바이트 단위 쓰기가 가능	- 빠른 쓰기 - 빠른 지우기 - 작은 블록 크기
단 점	- 느린 쓰기 - 느린 지우기	- 느린 임의 접근 - 바이트 단위 쓰기 불가능
응용 분야	- 코드용 메모리 - BIOS, 휴대전화 , PDA	- 데이터용 메모리 - USB 드라이브, 디지털 카메라, MP3 플레이어

[표 4] NOR, NAND 플래시 메모리의 비교

	NOR	NAND
Interface	BUS	I/O
Cell size	Large	Small
Cell cost	High	Low
Read time	Fast	Slow
Program Time	Fast	Slow
Erase Time	Slow	Fast
Power Consumption	High	Low

플래시 메모리의 기본 동작은 읽기(read), 쓰기(write), 그리고 지우기(erase)이다. 추가적으로 동작대기(suspend operation)가 있다[11].

▣ 읽기 - 일반 RAM과 동일하다.

▣ 지우기 - 기본적으로 플래시 메모리가 지워진 상태에서 데이터 내용은 0xFF 이다. 지우는 것은 비트를 '0'에서 '1'로 바꾸는 과정이다. 그리고 각각의 데이터들을 독자적으로 지울 수 없고 반드시 블록 단위로 지워야 한다. 블록의 크기는 플래시 메모리 마다 다르며 대게 8KB에서 128KB 사이이다. 예를 들어 플래시 메모리의 한 비트가 '0'이라면 이 비트를 지우거나 '1'로 기록하기 위해서는 그 비트가 포함된 전체 블록을 지워야 한다. 이 지우는 과정은 해당 블록의 모든 '0'들을 '1'로 바꾼다. 플래시 메모리 드라이버에서 가장 중요한 일 중의

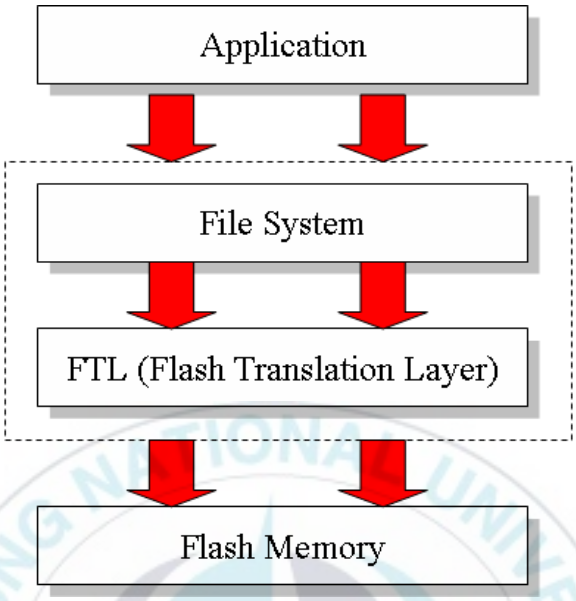
하나가 데이터를 포함하는 블록을 지우는 것이다.

▣ 동작대기 - 대부분의 플래시 메모리는 지우기와 프로그램 오퍼레이션을 대기시키는 것이 가능하다. 이는 플래시 메모리에 쓰기, 혹은 지우기 오퍼레이션을 하는 동안 시스템이 시간적인 제약이 있는 코드나 데이터에 접근하려고 하는 것을 가능하게 한다. 플래시 메모리가 대기 명령을 받으면 현재 오퍼레이션을 적당한 시간동안 대기하게 한다. 이 시간을 대기 지연이라고 부른다. 이 때 오퍼레이션의 현재 상태를 저장하고 플래시 메모리를 읽기가 가능한 상태로 만든다. 이 기간이 끝나면 대기된 오퍼레이션을 다시 실행한다.

대기, 복귀 사이클은 블록 삭제와 같은 긴 오퍼레이션 동안 반복적으로 수행된다.

2.2 FTL(Flash Translation Layer)

FTL[4]은 플래시 메모리의 삭제 연산을 감추기 위한 미들웨어로 호스트 시스템의 파일시스템과 플래시 메모리의 사이에 위치한다. 삭제 연산은 쓰기 연산 시에 파일시스템이 생성한 논리 주소를 플래시 메모리상의 이미 삭제 연산을 수행한 영역에 대한 물리 주소로 변환함으로써 감춰진다.



[그림 1] 플래시 메모리 시스템 구조

비교적 수행 시간이 오래 걸리는 삭제 연산을 감추고 I/O를 하나의 단위(Atomic Operation)로 처리하므로 하드디스크와 같은 단일 저장장치를 구성하므로, 상단에서 일반 파일시스템을 사용해서 플래시 메모리를 효율적으로 제어할 수 있다.

2.3 임베디드 데이터베이스 시스템 특성

임베디드 시스템은 시스템 목적과 특성상 저장장치로 하드디스크 보다 플래

시 메모리를 주로 사용한다. 임베디드 시스템에서 사용되는 데이터베이스 시스템은 시스템의 특성에 맞추어 small foot-print 데이터베이스 코드(1MB 내외)를 가진 경량화 된 데이터베이스 시스템이다.

임베디드 데이터베이스 시스템은 자립형(stand-alone) 데이터베이스 시스템 구조이다. 애플리케이션 코드와 데이터베이스 라이브러리(library)가 합쳐진 방식으로 데이터베이스가 하나의 파일로 구성된다. 일반적인 데이터베이스 시스템과 같이 클라이언트-서버(Client-Sever) 형식을 사용하지 않으며 서버 데몬(Server Daemon)이 없다.

또한 트랜잭션 커밋(commit) 시 FORCE 전략을 사용해서 트랜잭션 수행 도중 갱신된 모든 페이지를 저장장치에 기록한다.

서버 데몬이 없으므로 로깅(Logging) 기법을 사용하기 곤란한 단점이 있다.

2.4 플래시 메모리 기반 데이터베이스

플래시 메모리가 대용량화 되면서 저장장치로 각광받게 된 것은 극이 최근의 일이다. 따라서 플래시 메모리에 기반으로 한 데이터베이스 시스템에 대한 연구가 아직은 그리 많지 않은 편이나 일부 데이터베이스 색인(Index)에 대한 연구 결과가 발표된 바 있다[7,8].

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 용이하

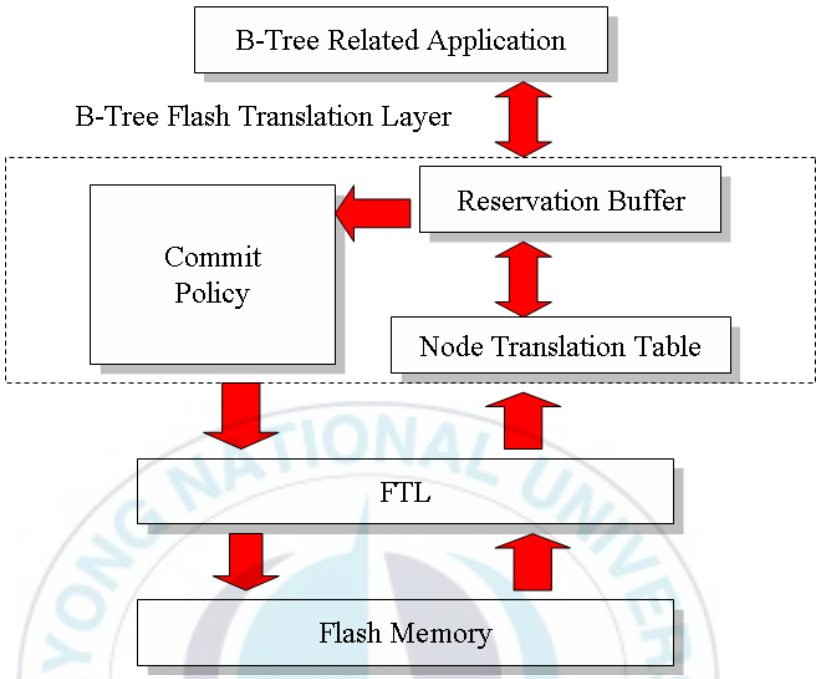
며 확장성이 우수한 색인 기법으로 유명하다.

하지만 하드 디스크에서와 달리 플래시 메모리 상에서 구현한 B-트리는 빈번한 페이지 갱신에 따른 덮어쓰기 증가로 인해 색인 구축비용이 매우 커진다는 문제가 있다. 따라서 플래시 메모리 환경에 적합한 트리 구현 기법이 필요하다.

2.4.1 BFTL(B-Tree Flash Translation Layer)

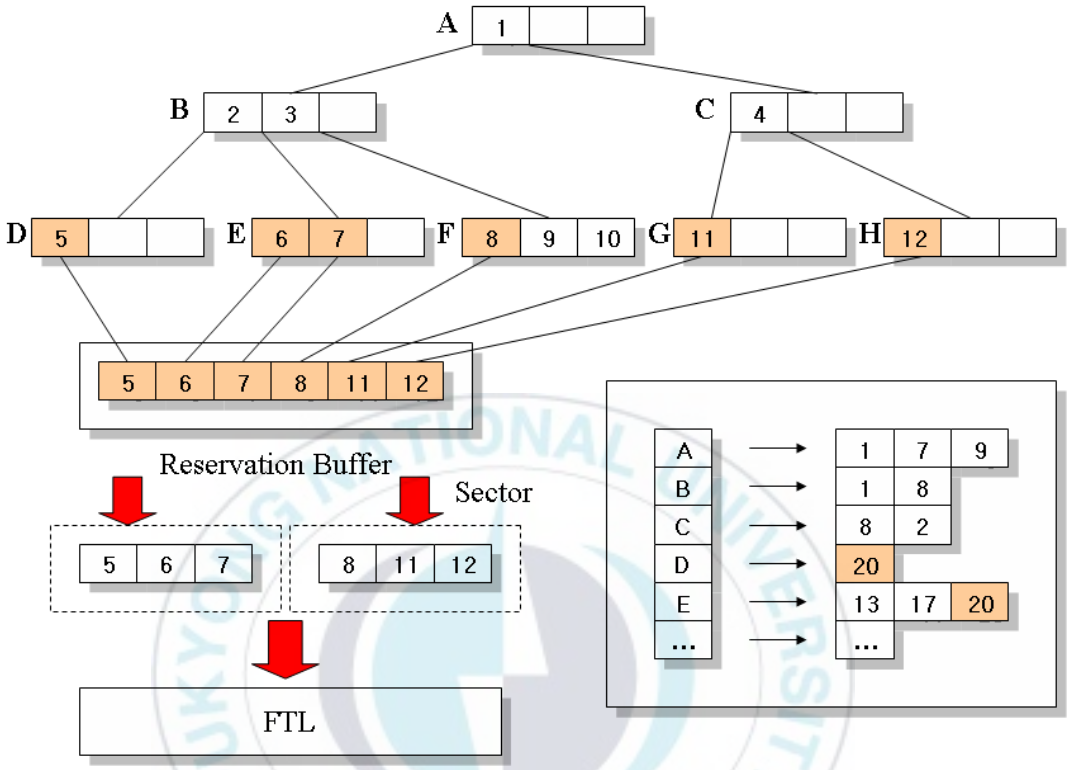
그림 2은 BFTL[7] 기법의 구성 및 구현 방법을 도식화한 그림이다. BFTL은 새로 입력된 색인 유닛(Index Unit)을 임시로 저장하는 유보 버퍼(Reservation Buffer), 플래시 메모리에 저장된 색인 유닛의 논리 섹터 주소를 기록한 노드 변환 테이블(Node Translation Table), 유보 버퍼 내에 있는 색인 유닛을 플래시 메모리로 저장하기 위한 수용정책(Commit Policy)으로 구성 되어져 있다.

새로운 키 값이 입력되면 유보 버퍼와 노드 변환 테이블의 해당 노드 리스트를 이용하여 키 값이 삽입될 단말 노드(leaf node)를 구성하고 키 값을 해당 노드에 삽입하여 색인 유닛을 생성한다. 생성된 색인 유닛은 일시적으로 유보 버퍼에 저장되며 유보 버퍼 내에 빈 공간이 없을 경우 수용 정책에 의해 쓰기 연산이 수행되어 플래시 메모리에 저장된다.



[그림 2] BFTL(B-Tree Flash Translation Layer)

BFTL 기법은 각 색인 유닛이 수용 정책에 의해 플래시 메모리로 저장될 때 유보 버퍼 내에 서로 다른 노드에 속하는 색인 유닛들이 플래시 메모리상의 동일한 섹터에 저장될 수 있다. 따라서 플래시 메모리상에 흩어진 색인 유닛들을 관리하기 위해서 RAM과 같은 휘발성 메모리에 노드 변환 테이블을 추가하여 색인 유닛의 저장된 논리 섹터 주소를 리스트로 유지한다. BFTL 기법은 B-트리를 구축 시 유보 버퍼를 사용함으로써 쓰기 연산을 감소 시켰지만 노드 구성



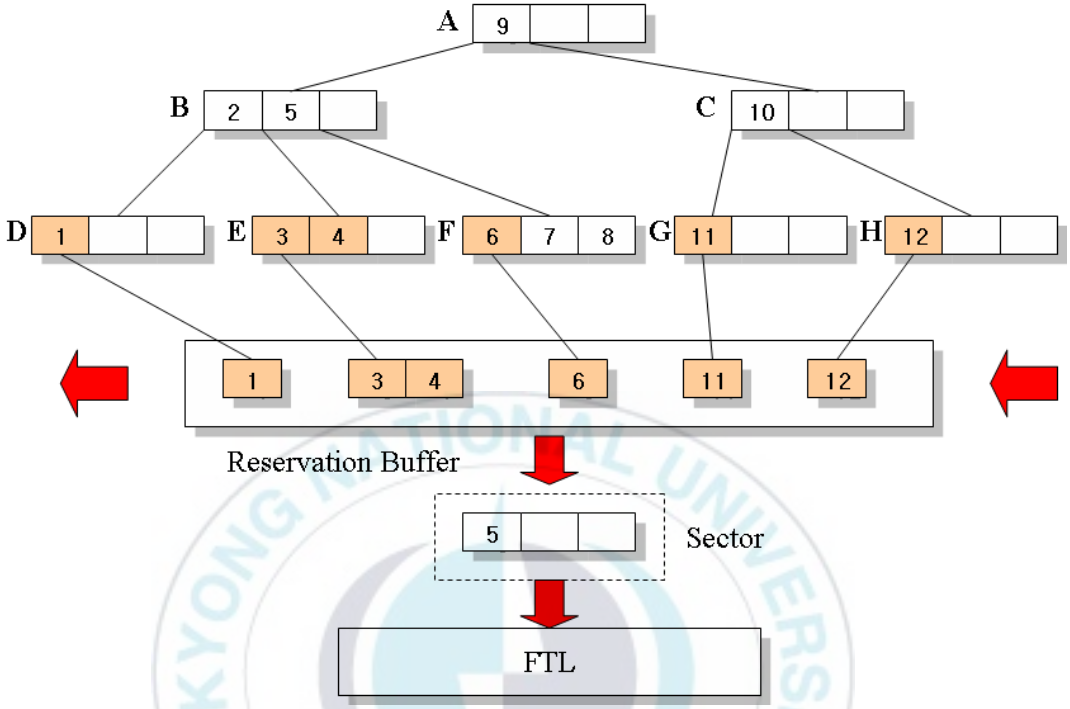
[그림 3] BFTL 노드 변환 테이블

과 검색 시 노드 변환 테이블의 리스트를 참조하기 때문에 읽기 연산이 많이 발생하여 검색 비용이 증가하게 된다. 또한 유보 버퍼와 노드 변환 테이블을 추가하기 위한 하드웨어 비용이 증가 한다. 또한 유보 버퍼에서 플래시 메모리에 데이터를 저장할 때 버퍼 교체 전략을 사용하지 않는다는 문제점이 있다.

2.4.2 BOF(B-Tree On Flash Memory)

BOF 기법[12]은 BFTL 기법과 동일한 색인 유닛을 사용한다. 검색비용을 줄이기 위해서 같은 노드에 속하는 유보 버퍼내의 색인 유닛들을 플래시 메모리 상의 동일한 섹터에 저장한다. 한 섹터에 한 노드를 저장함으로써 특정 노드를 구성할 때 한 번의 읽기 연산을 수행하여 노드를 구성 할 수 있으며, 노드 변환 테이블을 유지하기 위한 추가적인 하드웨어 비용이 필요 없다.

입력된 키 값을 해당 노드에 삽입하기 위해선 삽입될 단말 노드를 검색한 후 플래시 메모리에 저장된 해당 노드 데이터와 유보 버퍼안의 색인 유닛과의 노드 병합(node merge)을 수행한다. 마지막으로 병합된 노드에 새로 입력된 키를 삽입함으로써 B-트리 노드를 구성하게 된다. 만약 병합된 노드에 빈 공간이 없는 경우 노드 분리(node split)가 발생하는데 노드 분리가 발생하면 버퍼관리에 의해 새로 생성된 노드와 기존의 노드를 플래시 메모리에 저장한다. 노드 분리 후 유보 버퍼에 삽입하지 않고 플래시 메모리에 직접 저장함으로써 노드 분리가 발생할 때 기존 노드의 쓰레기 값을 제거한다.



[그림 4] BOF(B-Tree On Flash Memory)

그러나 노드 분할 등 노드에 중복적인 쓰기가 발생할 경우 (ex : 0, 0, 0, 0) 데이터 관리가 취약하며 플래시 메모리의 특징을 반영하지 않은 단순한 버퍼 교체 전략(FIFO)을 사용한다는 단점이 있다.

Ⅲ. 지연쓰기 기법

제안하는 기법은 모바일 환경과 같은 임베디드 데이터베이스 시스템을 대상으로 한다. 임베디드 데이터베이스 시스템은 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 동작한다.

▣ 동시성 제어(Concurrency Control)는 데이터베이스 단위와 같이 비교적 큰 단위의 잠금 단위(Locking Granularity)를 사용한다.

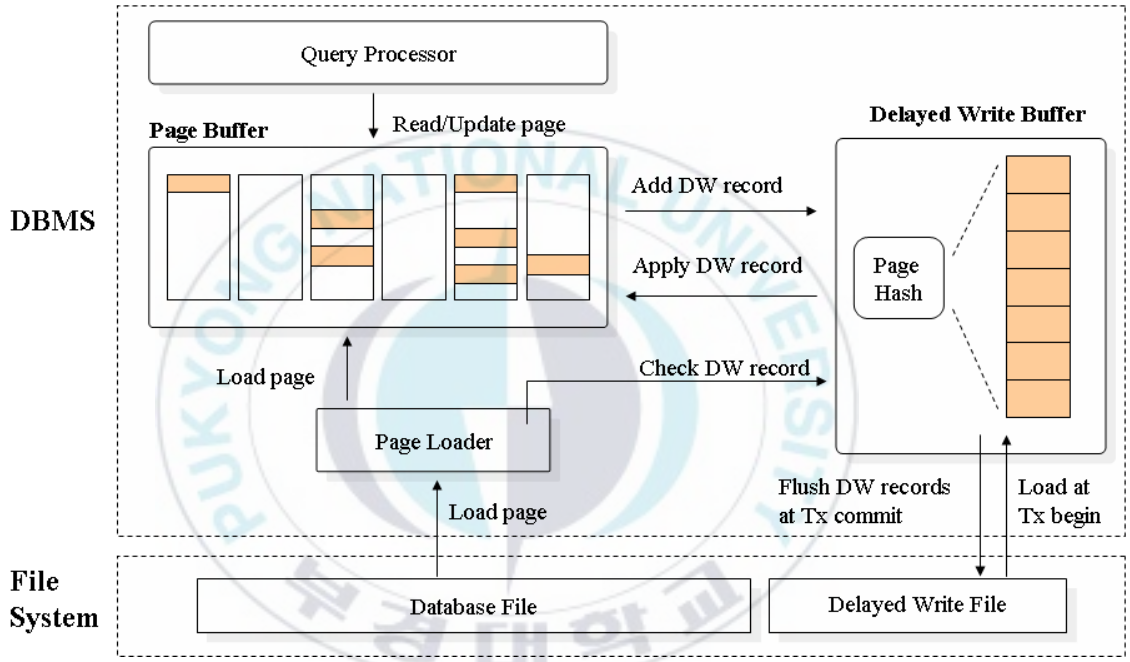
3.1 지연쓰기 기법 설계

제안하는 기법은 이러한 임베디드 데이터베이스 시스템 특성에 대한 가정을 바탕으로 하여 단일 데이터에 대해서는 오직 하나의 프로세스만이 갱신 연산을 수행할 수 있는 것으로 한다. 아울러 쓰기 지연의 대상이 되는 데이터베이스 페이지는 인덱스 페이지와 작은 레코드(small record[1])들 저장하는 페이지만이

1) 레코드의 크기가 데이터베이스의 페이지 크기보다 작은 레코드

해당되며 BLOB(Binary Large Object)의 데이터를 저장하기 위한 페이지는 해당되지 않는다.

그림 5는 지연쓰기 기법의 데이터베이스 시스템 구조를 나타낸 것이다.



[그림 5] 지연쓰기를 위한 데이터베이스 시스템 구조

지연쓰기 기법의 데이터베이스 시스템은 데이터베이스 시스템에 페이지 버퍼(Page Buffer)와 지연쓰기 버퍼(Delayed Write Buffer)가 있으며, 파일시스템에 데이터베이스 파일(Database File)과 지연쓰기 파일(Delayed Write File)이 존재한다.

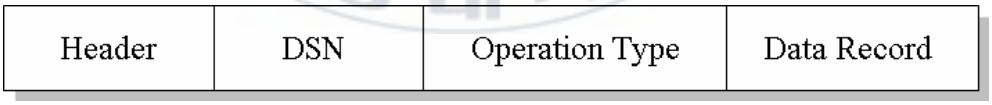
제안하는 기법은 데이터베이스 페이지에 대한 쓰기를 다음과 같은 과정으로 수행한다.

▣ Step 1. 지연쓰기 버퍼에서 갱신 페이지에 대한 지연쓰기 레코드의 개수 및 그 크기의 합을 계산한다.

▣ Step 2. 만약 지연쓰기 레코드의 개수가 일정개수(*THcount*) 이상이거나 지연쓰기 크기의 합과 새로 갱신될 크기의 합이 일정 크기(*THsize*)이상이면 지연쓰기 버퍼에서 해당 페이지와 관련된 지연쓰기 레코드를 모두 제거하고 Sync_Page로 표시한다. Step 4로 진행한다.

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

▣ Step 4. 해당 페이지를 갱신한다.



[그림 6] 지연쓰기 레코드 구조

그림 6은 지연쓰기 레코드의 구조를 나타낸 것으로 각 필드는 다음과 같이 구성된다.

▣ Header : 레코드 헤더

▣ DSN : 단조 증가하는 일련번호

▣ Operation Type : 삽입/삭제/갱신

▣ Data Record : 바이트 배열로 표현된 실제 저장될 레코드

페이지 캐시의 상위 계층(예 : 질의 처리기)에 의해 데이터베이스 페이지를 플래시 메모리에서 읽어 와야 하는 경우는 다음과 같이 동작한다.

▣ Step 1. 플래시 메모리로부터 지정된 페이지 영역을 읽어 들여 페이지 버퍼에 적재한다.

▣ Step 2. 페이지 식별자(Page Identifier, PID)와 일치하는 Sync_PID를 가지는 지연쓰기 레코드(Delayed Write Record)가 존재하는 지를 검사한다.

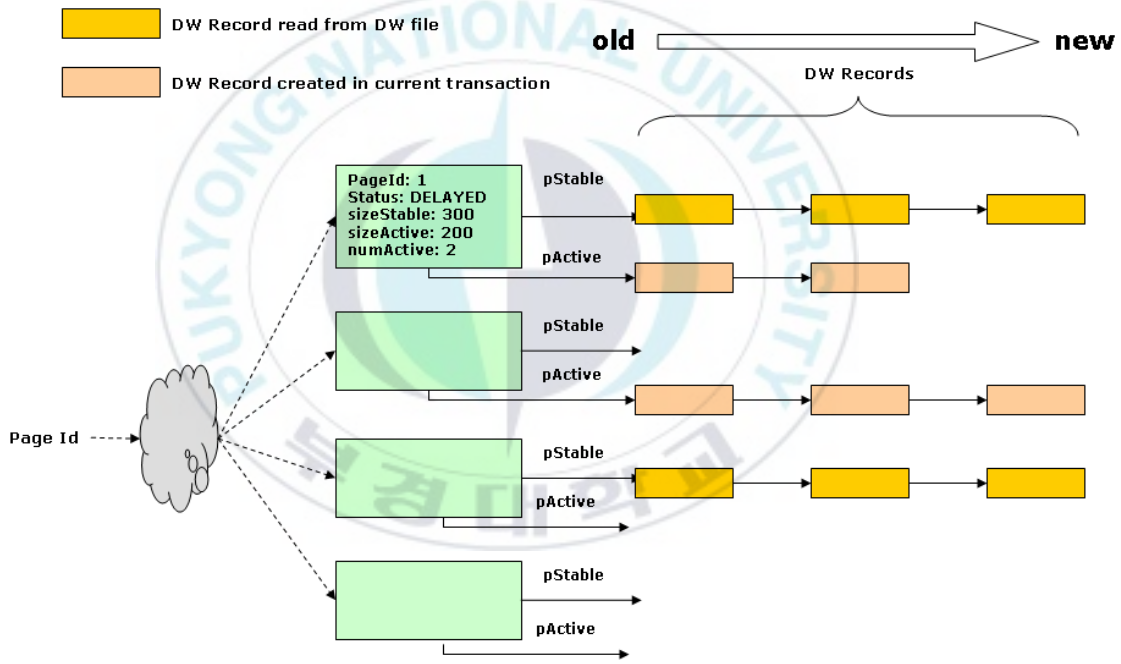
▣ Step 3. 만약 해당 페이지와 관련된 지연 쓰기 레코드가 존재하면 해당되는 지연쓰기 레코드를 사용하여 병합(merge)을 수행한다. 만약 다수의 지연쓰기 레코드가 해당되면 버퍼 내의 순서대로 병합을 수행한다.

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

2) UNDO를 위해 스왑영역(swap area)을 사용하는 것으로 가정한다.

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

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



[그림 7] 지연쓰기 버퍼의 구조

지연쓰기 버퍼가 일정 크기(Buffer Size) 이상이 되면 지연쓰기 레코드들을 페이지 식별자 별로 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 버퍼에 적재하고 지연된 쓰기 연산을 모두 수행한다.

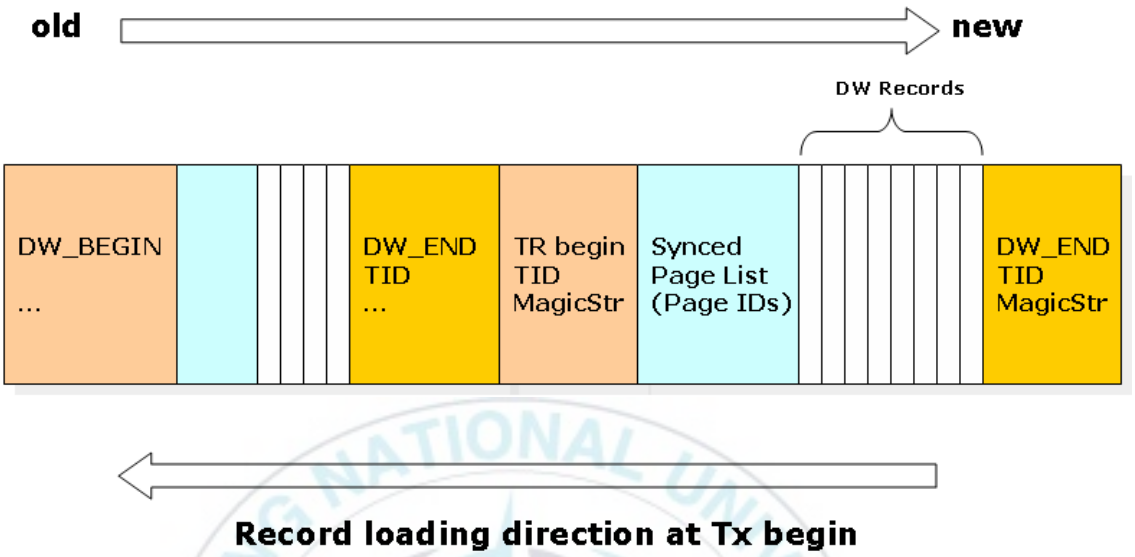
트랜잭션을 시작 할 경우에는 다음과 같이 동작한다.

▣ Step 1. 지연쓰기 버퍼를 초기화 한다.

▣ Step 2. 데이터베이스에 연관된 지연쓰기 파일이 존재하는지 검사한다.

▣ Step 3. 지연쓰기 파일을 끝에서 시작 방향으로 순차적(sequential)으로 읽으면서 지연쓰기 레코드를 지연쓰기 버퍼에 적재한다.

▣ Step 4. Synced Page List의 페이지들에 대해서 PURGE_STATE로 표시한다.



[그림 8] 지연쓰기 파일 구조

트랜잭션 커밋을 시작 할 경우에는 다음과 같이 동작한다.[3)]

▣ Step 1. DW_BEGIN 레코드를 지연쓰기 파일에 기록한다.

▣ Step 2. 페이지 버퍼는 트랜잭션 수행 도중 갱신된 페이지(Dirty Pages)들을 찾아낸다.

▣ Step 3. 갱신된 페이지 중에 SYNC_PAGE로 표시된 페이지 들을 데이터베이스에 모두 기록한다.

▣ Step 4. SYNC_PAGE의 리스트를 지연쓰기 파일에 기록한다. (Synced Page

3) 커밋 시 페이지 버퍼는 FORCE 전략(strategy)을 사용하는 것으로 가정한다.

List)

▣ Step 5. SYNC_PAGE로 표시되지 않은 Dirty Page는 각각의 Active List에 포함된 지연쓰기 레코드를 지연쓰기 파일에 기록한다.

▣ Step 6. DW_END 레코드를 기록한다.

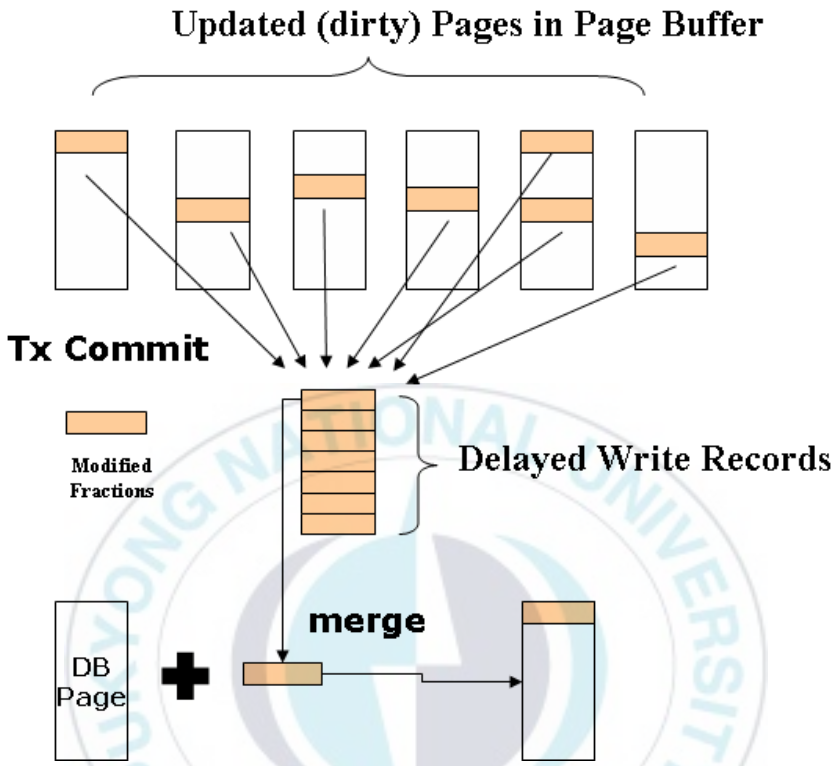
저장된 지연쓰기 레코드들은 데이터베이스 시스템이 재시작 되는 경우 플래시 메모리에서 메인 메모리로 읽혀지게 된다.

정상 동작 중에 응용 프로그램에 의해 트랜잭션이 취소되는 경우에는 플래시 메모리에 기록된 지연쓰기 레코드의 DSN중 최대값(*DSNfmax*)보다 큰 DSN을 가지는 지연쓰기 레코드를 메모리 내의 지연쓰기 버퍼에서 제거한다. 그리고 페이지 버퍼내의 페이지들에서 갱신된 페이지들을 검색한다. 만약 갱신된 페이지의 DSN이 *DSNfmax* 보다 작으면 해당 페이지는 버퍼에 계속 보존하고 그렇지 않은 경우에는 무효화 시킨다.

그림 9는 지연쓰기 기법의 쓰기 및 읽기의 방법에 대해 도식화한 그림이다.

일반 데이터베이스 시스템은 페이지의 일부가 갱신되더라도 전체 페이지를 갱신한다. 쓰기 전 삭제 연산을 실행해야 하는 플래시 메모리를 사용하는 DBMS의 경우 큰 오버헤드(overhead)로 작용된다.

지연쓰기 기법은 여러 페이지에 갱신해야 할 데이터를 지연쓰기 버퍼에 단일 페이지로 저장함으로써 갱신되는 페이지의 수를 크게 감소시킬 수 있다. 이는



[그림 9] 지연쓰기 기법 쓰기 및 읽기

데이터 레코드의 크기가 100 바이트라고 가정하고 페이지의 크기가 4K 바이트 또는 8K 바이트 정도라고 할 경우 최대 1/40 ~1/80 수준으로 페이지 쓰기를 줄일 수 있음을 의미한다.

또한 지연쓰기 페이지들은 연속된(contiguous) 페이지들로 저장할 수 있다. 연속된 페이지들에 대한 쓰기 연산은 디스크에서 뿐만 아니라 플래시 메모리에서도 임의(random) 쓰기보다 좋은 성능을 나타내므로 부가적인 입출력 성능을 향상

시킬 수 있다.

갱신 영역이 큰 페이지는 지연쓰기를 하지 않고 페이지 전체를 플래시 메모리에 저장한다.

갱신이 반영되지 않은 페이지를 추후 읽을 경우 페이지 로더(Page Loader)가 지연쓰기 버퍼를 검사해서 해당 페이지를 검색한 다음 플래시 메모리에 저장된 페이지와 지연쓰기 버퍼에 존재하는 페이지를 병합한다. 지연쓰기 기법은 CPU 사용 측면과 메모리 사용 측면에서 약간의 오버헤드가 따른다. 페이지를 읽을 때마다 지연쓰기 버퍼를 페이지 식별자를 이용하여 검색하고 만약 지연쓰기 된 페이지로 판명되면 지연쓰기 레코드를 이용하여 페이지를 최신 상태로 변경해야 한다. 여기서 페이지 식별자를 이용한 검색 과정에 소요되는 시간은 초기에 별로 크지 않으며 코드 최적화를 통해 상당 부분 줄일 수 있다. 그러나 캐시 내의 페이지를 최신 상태로 변경하는 과정은 상당 시간이 소요 될 수 있다. 이를 위해 제안하는 기법에서는 지연쓰기가 일정 횟수(*THcount*) 또는 일정 크기 이상(*THsize*)이 되면 지연쓰기를 하지 않고 플래시 메모리의 페이지에 갱신하도록 하였다.

부가적으로 소용되는 메모리 공간 중 가장 큰 부분은 지연쓰기 버퍼와 페이지 해시에 사용되는 SDRAM이다. 지연쓰기 버퍼를 200으로 하고 레코드의 크기가 100 바이트라면 약 20K 바이트 정도의 SDRAM과 플래시 메모리를 사용하게 된다.

지연쓰기 기법의 경우 플래시 전용 파일 시스템을 사용하지 않더라도 데이터베이스 시스템 수준에서 적용이 가능한 장점이 있다.

3.2 기존 연구와의 비교

BFTL 기법과 BOF 기법의 유보 버퍼(Reservation Buffer)와 제안하는 기법의 지연쓰기 버퍼는 페이지에 대한 갱신 연산을 덮어쓰기로 수행하지 않고 별도의 플래시 공간에 저장한다는 점에서 유사하다고 할 수 있다.
반면 제안하는 기법에서는 페이지가 캐시에 적재될 때 지연쓰기 버퍼에 저장된 레코드를 적용하므로 페이지 캐시에는 갱신된 페이지가 사용되는 반면 BFTL에서는 갱신 이전의 페이지와 갱신 데이터가 서로 독립적으로 존재하므로 데이터의 중복이 발생한다. 유보 버퍼 측에서 데이터를 검색하면 페이지 버퍼의 데이터는 사용되지 않는다는 점에서 차이가 있다.

로그 버퍼 FTL 기법(Log Buffer FTL)[14]은 플래시 메모리의 섹터에 대해 갱신이 일어나는 경우 플래시 메모리의 일부 영역을 로그 버퍼로 지정하고 여기에 변경된 데이터를 저장하는 FTL 기법이다. 제안하는 기법과 로그 버퍼 기법 모두 동일한 페이지가 연속적으로 갱신되는 경우에 유리하다는 점에서는 유사한 특징을 가진다. 반면 동작하는 수준이 섹터와 페이지라는 점에서의 차이가 있

으며 제안하는 기법은 페이지의 내용 중 변화된 부분만을 유지하는 반면 로그 버퍼 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 고장 회복 단계에서 커밋된 데이터가 데이터베이스에 반드시 반영되는 용도로 사용되는 것인 반면 지영쓰기 레코드는 순전히 페이지 갱신을 지연하기 위한 용도이다.
따라서 고장 회복 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 고장 회복 단계가 아닌 정상적인 동작 중에 사용하는 것을 주요한 목적으로 한다는 점에서 차이가 있다.

Ⅳ. 실험 구현 및 결과

4.1 실험 구현 환경

제안한 기법의 성능을 평가하기 위하여 다음 조건에서 실험하였다. 성능 평가에 사용된 하드웨어 재원은 CPU - 인텔 펜티엄 듀얼 코어 2 와 Main Memory - 1GB RAM을 사용하였다. 데이터베이스는 대표적인 임베디드 데이터베이스 시스템인 SQLite 버전 3을 사용하였다. SQLite의 경우 오픈 소스로 개발자가 임의대로 소스 코드 변경 후 사용할 수 있는 장점이 있다.

표 5 성능 평가 환경

구 분	성능 평가 사양
CPU	Intel Pentium Core 2 Duo 1GHz
OS	Windows XP
Main Memory	1GB RAM
File System	FAT File System
Embedded DBMS	SQLite 3.3.10 Embedded DBMS

파일 시스템으로는 FAT을 사용하였다. 제안한 지연쓰기 기법은 본 논문 3.2장에서 밝혔듯이 플래시 전용 파일시스템을 사용하지 않더라도 데이터베이스 시스템 수준에서 사용이 가능하다. JFFS/JFFS2, YaFFS와 같은 플래시 전용 파일시스템을 사용하지 않고 구현하더라도 별 문제가 없다.

성능 평가 조건으로 데이터베이스 페이지 크기를 4K로 했으며 데이터베이스에 쓰는 레코드의 수를 1,000,000개로 했다. 또한 레코드의 크기(byte)를 16 바이트와 256 바이트로 나누어서 성능을 평가하였다.

표 6 성능 평가 테이블 크기

(레코드의 수 : 1,000,000개)

레코드 크기(byte)	데이블의 크기(MB)
16	22
256	268

트랜잭션 당 1, 4, 16개의 레코드를 삽입하여 쓰기 성능 평가를 비교 하였다. 또한 읽기 성능을 평가하기 위해 전체 레코드를 검색한다.

표 7 트랜잭션의 종류

INS1	트랜잭션 당 1개의 레코드 삽입
INS4	트랜잭션 당 4개의 레코드 삽입
INS16	트랜잭션 당 16개의 레코드 삽입
SCAN	전체 레코드를 검색

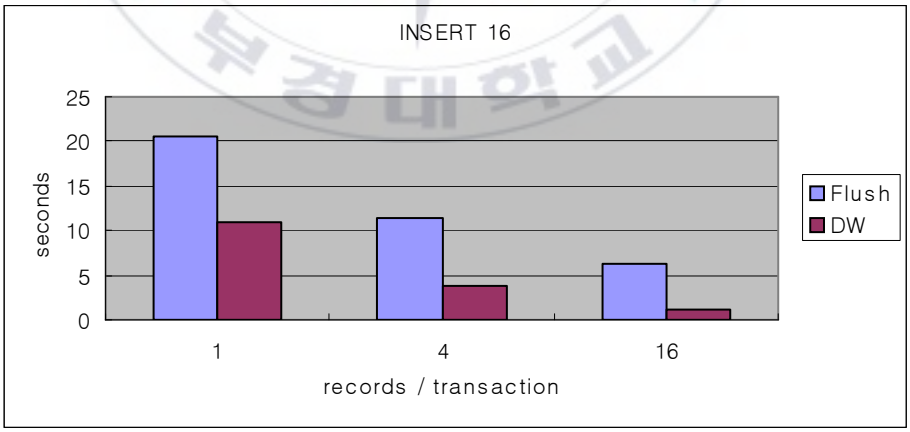
성능 평가에 대한 성능 측정 단위는 트랜잭션 당 소요시간과 트랜잭션 당 페이지의 쓰기의 수로 성능을 측정한다.

표 8 성능 측정 단위

성능 측정 단위
트랜잭션 당 소요시간 (sec/transaction)
트랜잭션 당 페이지의 쓰기의 수 (page writes/transaction)

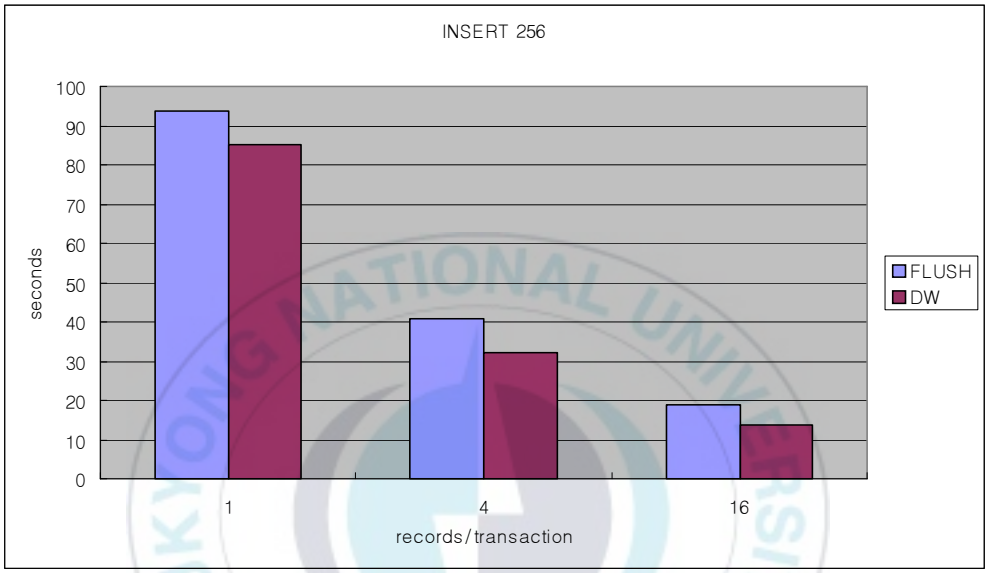
4.2 실험 구현 결과

그림 10은 16 바이트 크기의 레코드를 삽입한 경우의 결과를 나타낸 그림이다.



[그림 10] 성능 평가 결과 - 쓰기 트랜잭션(16 byte)

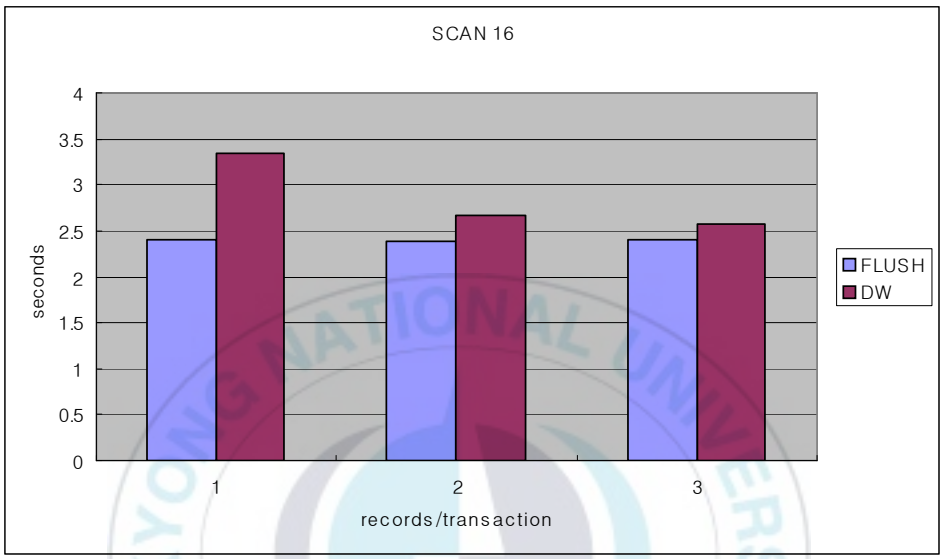
그림 11은 크기가 256 바이트인 레코드를 삽입한 경우의 결과를 나타낸 그림이다.



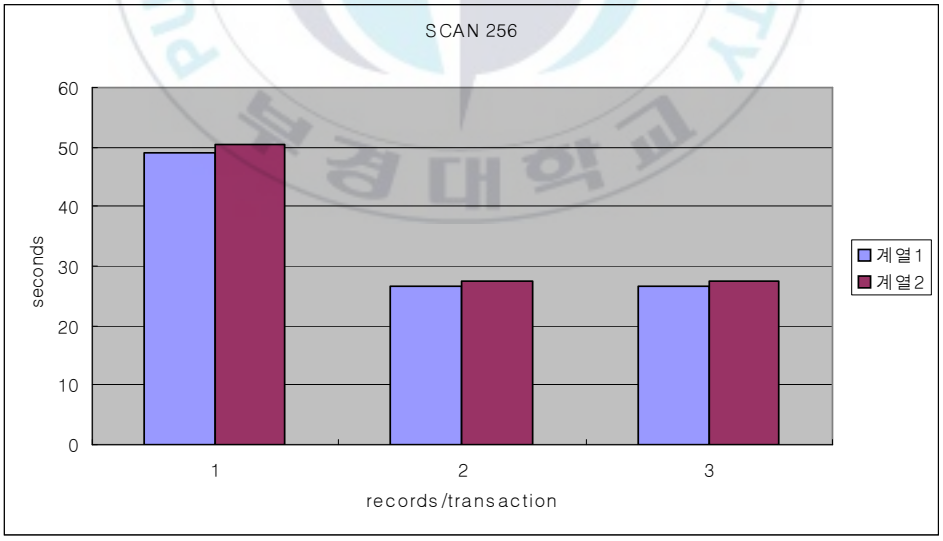
[그림 11] 성능 평가 결과 - 쓰기 트랜잭션(256 byte)

그림 10과 11을 함께 비교해 볼 때 일반적인 임베디드 데이터베이스 시스템에서 쓰기 기법보다 지연쓰기 기법이 트랜잭션 커밋 시 쓰기 성능 향상이 있음을 보여준다.

그림 12, 13은 기존의 기법과 지연쓰기 기법의 읽기 성능을 비교한 그림이다.



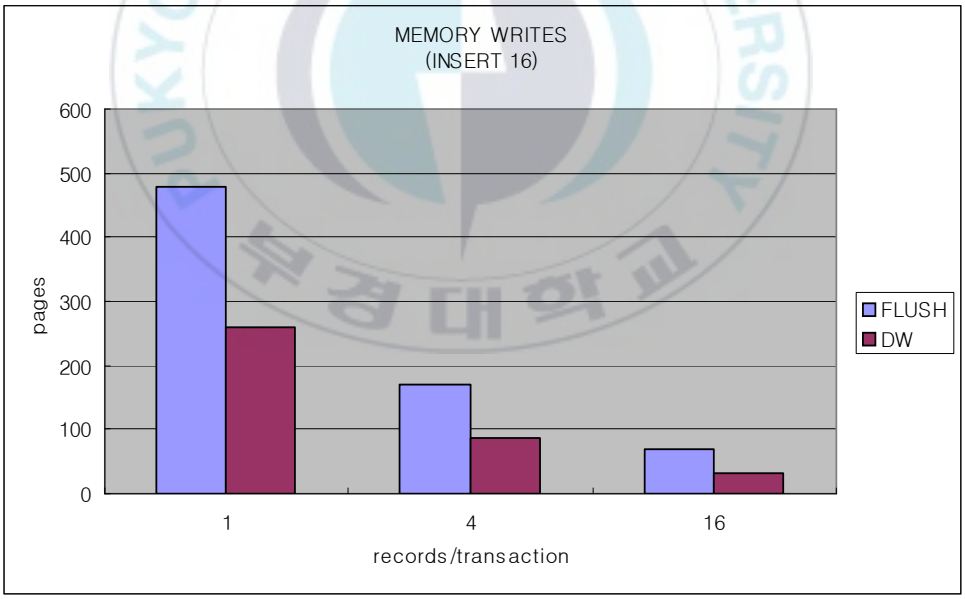
[그림 12] 읽기 성능 평가(16 byte)



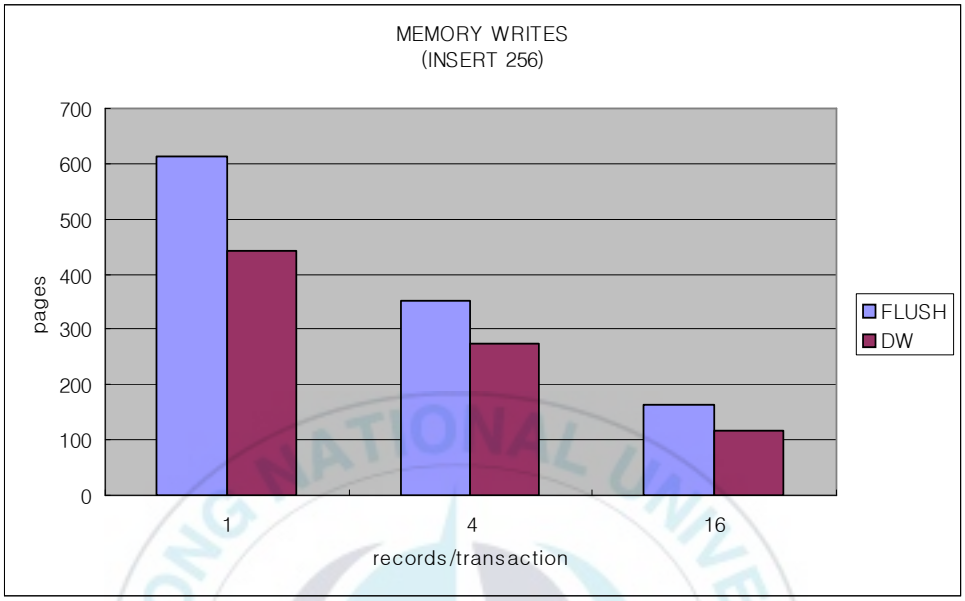
[그림 13] 읽기 성능 평가(256 byte)

그림 12, 13을 보면 지연쓰기 기법이 기존의 기법보다 읽기 성능이 6% - 40% 정도 성능이 감소한 것을 볼 수 있다. 지연쓰기 기법은 갱신이 반영되지 않은 페이지를 추후 읽을 때는 지연쓰기 정보를 지연쓰기 버퍼에서 읽어 와서 병합을 하므로 병합에 따른 오버헤드가 발생한다. 그러나 본 논문에서의 읽기 성능 평가의 경우 전체 레코드의 수가 1,000,000개의 이르기 때문에 실제 데이터베이스 시스템을 운용할 경우 이 보다 훨씬 작은 레코드를 운용하므로 읽기 성능 감소는 훨씬 줄어들 것으로 예상된다.

그림 14, 15는 쓰기 연산의 횟수를 비교 한 것이다.



[그림 14] 쓰기 연산의 횟수 비교 (16 byte)



[그림 15] 쓰기 연산의 횟수 비교 (256 byte)

그림 14, 15와 같이 지연쓰기 기법은 쓰기 페이지의 수를 감소시킨다. 지연쓰기 기법은 쓰기 연산의 횟수를 줄여 플래시 메모리의 마모도를 줄이므로 플래시 메모리 내구성을 향상 시킨다.

Ⅴ. 결론

플래시 메모리는 디스크 형태의 저장 장치에 비해서 충격에 강하고 적은 전력을 소모하기 때문에 디지털 카메라의 저장장치 및 내장형 장치의 보조 기억장치로 널리 사용되고 있다. 일반 메모리와 같은 빠른 읽기와 쓰기를 가지기 때문에 데이터의 변경이 적은 동영상 플레이어나 MP3 플레이어와 같은 멀티미디어의 장비들에 매우 적합하다.

플래시 메모리는 기존 디스크 장치와는 다르게 덮어쓰기가 되지 않으며, 쓰기 전 삭제 연산을 수행해야 하는 특성 때문에 기존의 데이터베이스 시스템의 기술을 그대로 사용하기에 어려움이 있다.

본 논문에서는 플래시 메모리 데이터베이스 환경에서 새로운 지연쓰기 기법을 제안하였다.

플래시 메모리에서 쓰기 연산과 소거 연산은 플래시 메모리의 연산중에서 가장 큰 비용을 소요한다. 지연쓰기 기법은 플래시 메모리의 특성을 고려하여 지연쓰기 레코드를 이용하여 단일 페이지에 다수의 갱신이 일어나는 경우 그리고 소량의 갱신이 여러 페이지에 걸쳐 일어나는 경우 플래시 메모리 쓰기 연산을 그룹핑하여 쓰기 연산의 횟수를 감소시켜 쓰기 성능 및 소거 연산을 향상 시킨다. 이러한 성능 향상을 통해 플래시 메모리의 마모도를 줄여 노화(aging)를 지연시킨다.

향후에는 플래시 메모리 파일 시스템에서의 테스트가 필요하며, 플래시 메모리 어웨어(aware) 버퍼 관리 기술 및 쿼리 프로세싱(query processing) 연구를 수행 할 예정이다.

참 고 문 헌

[1] Michael Wu, and Willy Zwaenpoel, "eWay: 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] David Woodhouse, "The Journalling Flash File System", RedHat Inc, http://sources.redhat.com/jffs2/, 2001.

[4] Tae-Sun Chung, Stein Park, Myung-Jin Jung and Bumsoo Kim, "STAFF : State Translation Applied Fast Flash Translation Layer", ARCS 2004 LNCS 2981, pp.199-212.

[5] 김정기, 박승민, 김채규, “임베디드 플래시 파일 시스템”, 정보처리학회지, 제 9권, 제 1호, pp.43~49, 2002.1.

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

[7] 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.

[8] 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.

[9] 박재호 “임베디드 리눅스”, 한빛 미디어, 2002.

[10] Samsung Electronics, SmartMedia Data Book, 2003.

[11] MSDN, "Flash Memory Operations", Microsoft. Inc, 2002.

[12] 남정현, 박동주 “플래시 메모리 상에서 효율적인 B-트리 설계 구현” 한국정보과학회 2005 추계 학술 발표 논문집(2), pp 55-57, 2005.

[13] Vipin Malik, "The Linux MTK, JFFS HOWTO", ftp://ftp.linux.org.uk, 2001.

[14] 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 Translations on Consumer Electronics Vol.48, No.2, pp.366-375, 2002.

감사의 글

부족한 저를 지속적인 보살핌과 배려, 그리고 성심을 다해 지도와 가르침을 주신 송하주 교수님께 진심으로 감사드립니다. 또한 부족한 저의 논문을 심사해주신 조우현 교수님, 권오흠 교수님께도 진심으로 감사드립니다. 그리고 학위 과정 동안 저에게 많은 가르침을 주셨던 학과의 여러 교수님들께도 감사드립니다.

낯설었던 대학원 생활에 많은 도움을 주신 길호 선배, 주현 선배, 바쁜 연구실 생활에도 불구하고 1년 동안 항상 챙겨주고 조언해준 충기 선배, 대학원 동기 현동형, 득룡, 진호, 규희, 영선누나에게도 감사의 마음을 전합니다.

연구실에서 동거 동락한 정모, 인철, 상우, 동완, 근기, 태경, 보성, 규동, 성진, 주미, 그리고 연구실 석사과정 후배 화춘, 종민 이하 선,후배 동기에게도 감사의 마음과 앞으로 열심히 해서 좋은 성과 있기를 기원합니다.

끝으로 저에게 무한한 사랑, 애정, 가르침과 함께 무한 부담까지 주신 세상에서 가장 사랑하고 존경하는 古人이 되신 祖父, 命 길고 福 많이 받기를 항상 기원해 주시며, 저를 키워주신 祖母, 사랑하고 존경하는 부모님, 사랑하는 여동생 나영, 저의 혈육, 친지 여러분께도 감사의 마음을 전합니다.

2007년 8월 윤 승 희 드림