플래시 메모리 환경을 위한 컨테이너 기반 레코드 관리 방법

(Container-Based Record Management in Flash Memory Environment)

배 덕 호 † 김 상 욱 †† 장 지 웅 †††
(Duck-Ho Bae) (Sang-Wook Kim) (Ji-Woong Chang)

요 약 플래시 메모리는 기존 저장 매체와는 달리 읽기 연산에 비해 쓰기 연산의 수행비용이 매우 크고, 저장된 데이타에 대한 갱신이 제한적인 고유의 특성이 있다. 본 논문에서는 플래시 메모리 환경이 기존의 레코드 관리 방법에 미치는 영향을 분석하고, 기존의 레코드 관리 방법을 그대로 플래시 메모리에 적용하였을 때의 문제점을 지적한다. 이를 기반으로 플래시 메모리 환경을 위한 효율적인 레코드 관리 방법을 제안한다. 제안하는 방법은 컨테이너 구조를 이용하여 레코드 삽입, 삭제, 수정 연산을 수행함으로써 덮어쓰기 연산을 효율적으로 수행할 수 있으며, 이로 인해 소거 연산을 크게 줄일 수 있다. 실험 결과에 의하면, 제안하는 방법은 기존 방법의 성능을 최대 34%까지 향상시키는 것으로 나타났다.

키워드 : 플래시 메모리 DBMS, 레코드 관리 방법, 컨테이너

Abstract Flash memory has its unique characteristics: i.e., (1) the write operation is much more costly than the read operation. (2) In-place updating is not allowed. In this paper, we first analyze how these characteristics affect the performance of record management in flash memory, and discuss the problems with previous methods for record management when they are applied to flash memory environment. Next, we propose a new record management method to be suitable for flash memory environment. The proposed method employs a new concept of a *container* that makes it possible to overwrite data on flash memory several times when performing insertions, deletions, and modifications of records. As a result, this method reduces the number of overwrite operations, and consequently does the number of erase operations. The results of experiments show that our method improves the performance by up to 34%, compared with the previous one.

Key words : Flash memory DBMS, Record management method, Container

· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업(IITA-2008-C1090-0801-0040) 및 2007년도 정부(과학기술부)의 재원으로 한국과학재단(R01-2007-000-11773-0)의 연구비 지원을 받았습니다.
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '플래시 메모리 환경을 위한 컨테이너 기반 레코드 관리 방법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 한양대학교 전자컴퓨터통신공학과
smith@zion.hanyang.ac.kr
†† 종신회원 : 한양대학교 전자컴퓨터통신공학과 교수
wook@hanyang.ac.kr
††† 정 회 원 : 한국산업기술대학교 게임공학과 교수
jwchang@kpu.ac.kr

논문접수 : 2008년 8월 25일
심사완료 : 2008년 12월 12일

Copyright©2009 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 데이타베이스 제36권 제1호(2009.2)

1. 서 론

플래시 메모리는 기존의 저장 매체들과는 다른 고유한 특성을 가진다. 첫째, 플래시 메모리는 덮어쓰기 연산(overwrite operation)이 제한적이다[1]. 데이타가 저장된 페이지를 갱신할 때, 플래시 메모리 소자를 1에서 0으로만 변경하는 갱신이 발생할 경우에만 덮어쓰기 연산을 통해 해당 페이지에 갱신된 데이타를 기록 가능하다. 만약 해당 페이지에 더 이상 덮어쓰기가 불가능하면, 기존의 페이지를 무효화하고, 갱신 사항이 반영된 데이타를 플래시 메모리의 새로운 페이지에 기록하는 방법을 사용한다[2,3]. 둘째, 플래시 메모리에는 오버헤드가 매우 큰 소거 연산이 존재한다. 덮어쓰기 연산을 수행하지 못해 생성된 무효화된 페이지(이하 무효 페이지라고 함)를 다시 사용하기 위해서는 해당 페이지가

포함된 블록에 대한 소거 연산을 먼저 수행하여야 한다[4]. 따라서 플래시 메모리에서는 덮어쓰기 연산을 최대한 수행하여 소거 연산의 횟수를 감소시키는 것이 매우 중요하다.

최근 플래시 메모리의 용량이 증가함에 따라 플래시 메모리에 직접 데이타를 저장, 관리하는 연구가 필요하게 되었다. 그러나 지금까지는 플래시 메모리에 대용량의 데이타를 저장, 관리하는 방법에 대한 연구가 거의 이루어지지 않았다. 대부분의 경우 기존의 디스크 기반 DBMS에서 고안된 방법을 그대로 플래시 메모리에 적용하여 사용하는 수준이었다[3]. 이는 플래시 메모리의 특성을 잘 반영하지 못하여 DBMS의 성능을 저하시키는 원인이 되었다. 그러므로 플래시 메모리의 특성을 고려한 데이타베이스 저장 기술에 대한 연구가 필요하다.

특히, 레코드의 삽입, 삭제, 수정 연산을 결정하는 레코드 관리 방법은 DBMS의 성능에 크게 영향을 미치는 중요한 연구 분야이다. 기존의 레코드 관리 방법은 슬롯을 이용하여 고정 길이 레코드(fixed length record)를 관리하였다[5].

그러나 기존의 고정 길이 레코드 관리 방법은 레코드가 기록되었던 슬롯에 다시 레코드를 기록하는 문제점이 존재한다. 특히, 레코드 수정 시 해당 슬롯에 갱신된 내용을 반영함으로써, 덮어쓰기가 가능한 경우가 매우 드물다. 이렇듯, 기존의 레코드 관리 방법은 많은 무효 페이지를 발생시킨다.

본 논문에서는 이러한 분석들을 바탕으로 플래시 메모리 환경을 위한 컨테이너 기반 레코드 관리 방법을 제안한다. 제안하는 방법은 고정 길이 레코드의 삽입과 수정을 레코드가 기록되었던 슬롯이 아닌 비어있는 새로운 슬롯에 반영하며, 이를 위해 슬롯의 사용 여부가 아닌 레코드의 상태에 따라 슬롯의 상태를 설정한다. 또한, 덮어쓰기 연산을 효과적으로 지원하기 위한 새로운 레코드 저장 구조인 컨테이너(container)를 제안하며, 이를 이용한 레코드 삽입, 삭제, 그리고 수정 방안을 제안한다. 본 논문에서는 다양한 실험을 이용한 성능 분석을 통하여 제안하는 방법의 우수성을 규명한다. 실험 결과에 의하면, 제안하는 컨테이너 방안은 레코드 관리 방법의 성능을 최대 34% 개선시키는 효과를 보인다.

본 논문의 구성은 다음과 같다. 제2절에서는 플래시 메모리의 특성을 분석하고, 레코드 관리 방법에 미치는 영향을 논의한다. 제3절에서는 기존의 레코드 관리 방법을 재조명하여 플래시 메모리 환경에서 새롭게 부각되는 문제점들을 분석한다. 제4절에서는 본 논문에서 제안하는 컨테이너 방안에 관하여 상세히 서술한다. 제5절에서는 성능 평가를 통하여 제안하는 방법의 우수성을 규명한다. 제6절에서는 논문을 요약하고, 결론을 맺는다.

2. 플래시 메모리의 특성과 레코드 관리 방법

플래시 메모리는 기존의 저장 매체와 다른 물리적인 특성이 있다. 본 절에서는 이러한 플래시 메모리 고유의 특징이 레코드 관리 방법에 미치는 영향에 대하여 논의한다.

플래시 메모리의 읽기와 쓰기 연산은 512바이트 크기의 페이지 단위로 이루어진다. 그리고 소거 연산은 연속된 여러 페이지로 구성된 16K바이트 블록 단위로 이루어진다[4]. 표 1은 플래시 메모리, 주기억 장치(DRAM), 그리고 하드 디스크에서 읽기, 쓰기, 소거 연산의 수행 시간을 나타낸다[6]. 플래시 메모리의 쓰기 연산은 읽기 연산에 비해 약 16.7배의 시간이 소요되며, 소거 연산은 쓰기 연산에 비해 약 10배의 시간이 소요된다.

플래시 메모리 소자는 소거 연산을 통해 1로 초기화되며, 쓰기 연산을 통해 특정 소자를 0으로 변경시켜 원하는 데이타를 저장할 수 있다. 저장된 데이타를 갱신할 때, 플래시 메모리 소자를 1에서 0으로만 변경하는 갱신이 발생하면, 쓰기 연산을 통해 해당 페이지에 갱신된 내용을 다시 기록할 수 있다. 이러한 쓰기 연산을 덮어쓰기 연산이라 하며 하나의 페이지는 플래시 메모리가 허용하는 제한된 횟수만큼의 덮어쓰기 연산이 가능하다[1][1].

표 1 여러 가지 저장 장치에서 각 연산의 수행 시간

	읽기 연산	쓰기 연산	소거 연산
Flash Memory	12㎲/512B	200㎲/512B	2㎳/16KB
DRAM	100㎱/1B	100㎱/1B	-
Hard Disk	12.4㎳/512B	12.4㎳/512B	-

하나의 페이지에 수행 가능한 덮어쓰기를 모두 수행하거나, 혹은 덮어쓰기를 수행할 수 없는 경우 기존의 페이지를 무효화하고, 갱신된 데이타를 플래시 메모리의 다른 영역에 기록한다. 무효 페이지를 다시 사용하기 위해서는 먼저 소거 연산을 수행하여야 한다. 소거 연산은 오버헤드가 매우 크며, 하나의 블록에 수행할 수 있는 소거 연산의 횟수는 제한되어 있다[7]. 그러므로 잦은 소거 연산은 플래시 메모리의 수명을 단축시키는 중요한 요인이 된다.

위에서 살펴본 바와 같이 플래시 메모리 환경에서는 소거 연산을 줄이기 위해 덮어쓰기 연산을 최대한 많이 수행하여, 무효 페이지의 생성을 줄이는 방법을 고려하여야 한다. 제3절에서는 이러한 고찰을 기반으로 하여 디스크 기반 DBMS에서 널리 사용되고 있는 레코드 관리 방법에 대하여 분석한다.

1) 삼성 K9F5608U0M 플래시 메모리는 데이타 영역에 대해서 2회, 스페어 영역에 대해서 3회의 덮어쓰기 연산의 횟수를 제한하고 있다.

3. 플래시 메모리 환경에서 기존의 레코드 관리 방법의 고찰

본 장에서는 디스크 기반 DBMS의 레코드 관리 방법을 구체적으로 소개한다. 그리고 이 방법을 그대로 플래시 메모리에 적용했을 때의 문제점에 대해 논의한다.

기존의 레코드 관리 방법에서 하나의 파일은 여러 개의 페이지로 구성된다. 플래시 메모리의 읽기, 쓰기 연산의 단위인 하나의 페이지는 동일한 크기의 논리적 단위인 슬롯들로 구성되며, 하나의 슬롯에는 하나의 고정 길이 레코드를 저장한다[5]. 하나의 슬롯은 레코드가 저장될 공간과 해당 슬롯의 사용 여부를 나타내는 상태 비트로 구성된다. 그림 1은 슬롯 페이지의 구조를 나타낸다. 슬롯에 레코드가 저장되어 있다면, 상태 비트는 0으로 설정되며, 슬롯이 비어있다면, 상태 비트는 1로 설정된다.

그림 2는 기존의 레코드 관리 방법의 레코드 삽입, 삭제, 그리고 수정 절차를 나타낸다.

기존의 레코드 관리 방법은 레코드가 저장되었던 슬롯에 다시 레코드를 저장하는 문제점이 존재한다. 첫째, 새로운 레코드 삽입 시, 레코드가 삭제되었던 슬롯에 다시 레코드를 저장하는 경우가 발생할 수 있다. 레코드가 삭제되었던 슬롯은 기존에 저장된 레코드가 그대로 기록되어 있으며, 슬롯의 상태 비트만 1로 설정되어 있다. 따라서 해당 슬롯에 레코드를 다시 저장할 경우, 플래시 메모리 소자를 1에서 0으로만 변경하여 새로운 레코드를 저장할 수 있는 경우는 매우 드물다.

그림 3은 레코드가 삭제되었던 슬롯에 다시 새로운 레코드를 저장하는 예를 나타낸다. 그림 3(a)에서 슬롯 0에 저장되어 있던 레코드가 삭제된다. 삭제된 슬롯은 더 이상 유효한 레코드가 저장되어 있지 않는 비어있는 슬롯이므로, 그림 3(b)에서 새롭게 삽입되는 레코드는

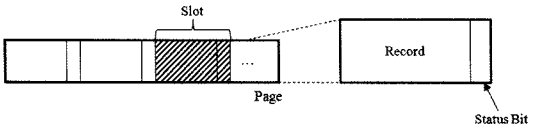
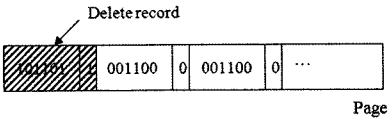


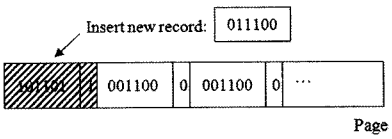
그림 1 슬롯 페이지 구조

단계 1. 레코드를 삽입, 삭제, 수정할 페이지를 탐색한다.
단계 2.
[삽입] 탐색된 페이지의 빈 슬롯에 레코드를 저장한 후, 해당 슬롯의 상태 비트를 0으로 설정한다.
[삭제] 삭제할 레코드가 저장된 슬롯의 상태 비트를 1로 설정한다.
[수정] 수정될 레코드가 저장된 슬롯에 갱신된 새로운 레코드를 저장한다.

그림 2 레코드 관리 방법의 레코드 삽입, 삭제, 수정 절차



(a) 슬롯 0의 레코드 삭제



(b) 슬롯 0에 레코드 삽입

그림 3 레코드가 삭제되었던 슬롯에 새로운 레코드를 저장하는 예

슬롯 0에 저장된다. 이 때, 기존의 저장된 레코드의 플래시 메모리 소자를 1에서 0으로만 변경하여 새로운 레코드를 저장할 수 있는 경우는 매우 드물다.

둘째, 레코드 수정 시, 기존의 슬롯에 갱신된 내용을 반영한다. 위와 마찬가지로, 기존의 레코드를 1에서 0으로만 변경하여 수정된 레코드를 저장할 수 있는 경우는 매우 드물다.

이와 같이, 기존의 레코드 관리 방법은 덮어쓰기 연산을 거의 활용하지 못하고, 새로운 페이지를 할당받는 경우가 대부분이다. 이는 많은 무효 페이지를 발생시키고, 이로 인한 소거 연산을 빈번히 수행하는 결과를 가져온다.

4. 제안하는 방법

4.1 기본 전략

본 절에서는 기존의 레코드 관리 방법의 문제점을 바탕으로 플래시 메모리의 고유한 특성에 적합한 새로운 레코드 관리 방법의 기본 전략을 기술한다.

첫째, 레코드가 저장되었던 슬롯에는 다시 레코드를 저장하지 않는다. 레코드가 저장되었던 슬롯에 다시 레코드를 저장할 경우, 플래시 메모리의 특성 상 덮어쓰기 연산을 거의 활용할 수 없다. 따라서 제안하는 레코드 관리 방법은 비어있는 새로운 슬롯에 레코드를 저장한다.

둘째, 이를 위해 슬롯의 사용 여부가 아닌 레코드의 상태에 따라 슬롯의 상태를 설정한다. 디스크의 쓰기 연산은 소자의 변경에 제한이 없는 반면, 플래시 메모리의 쓰기 연산은 소자를 1에서 0으로만 변경이 가능하다. 즉, 플래시 메모리는 사용하였던 슬롯을 다시 사용하는 것이 제한되어 있다. 따라서 제안하는 레코드 관리 방법은 레코드가 기록되었던 슬롯을 무효화하여, 해당 슬롯에는 더 이상 새로운 레코드를 저장하지 않는 전략을 사용한다.

더 나아가, 레코드 수정 시, 기존의 레코드를 무효화

하고, 비어있는 슬롯에 수정된 레코드를 저장한 후, 기존의 레코드가 수정된 레코드를 가리키는 방법을 사용한다. 이를 통해 제한된 덮어쓰기 연산을 최대한 수행할 수 있으며, 생성되는 무효 페이지의 수를 크게 줄일 수 있다.

4.2 컨테이너 기반 레코드 관리 방법

컨테이너는 하나의 페이지를 동일한 크기로 나눈 논리적인 구조이며, 하나의 컨테이너에는 하나의 고정 길이 레코드가 저장된다.[2] 컨테이너는 레코드를 저장하는 데이타 부분과 컨테이너에 저장된 레코드를 관리하기 위한 컨테이너 메타데이타 부분으로 구성된다.

컨테이너 메타데이타는 해당 컨테이너 안에 저장된 레코드의 상태를 나타내는 상태 비트(status bits)와 수정이 발생하였을 경우 수정된 레코드가 저장된 컨테이너의 위치를 표시하기 위한 이동된 주소 비트(moved address bits)로 구성된다. 그림 4는 제안하는 컨테이너 방법의 구조를 나타낸다.

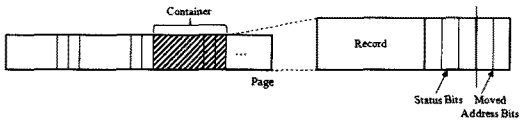


그림 4 컨테이너 구조

컨테이너의 상태 비트는 총 3비트로 구성되며, 컨테이너의 상태는 레코드가 저장되지 않은 비어있는 상태(free status), 유효한 레코드가 저장된 유효 상태(valid status), 삭제에 의해 무효한 레코드가 저장된 무효 상태(invalid status), 그리고 수정에 의해 해당 페이지의 다른 컨테이너로 이동된 상태(moved status) 등 총 4가지의 상태가 존재한다.

제안하는 컨테이너 기반 레코드 관리 방법은 플래시 메모리 소자를 1에서 0으로의 변경만으로 모든 컨테이너의 상태를 표시할 수 있다. 그림 5는 컨테이너의 상태 변화에 따른 메타데이타 부분의 변화를 나타낸다. 그림 5(a)에서 모든 비트가 1로 설정되어 있으면, 비어 있는 상태를 나타낸다. 그림 5(b)에서 해당 컨테이너에 새로운 레코드가 삽입되어 유효 상태가 되면, 첫 번째 비트를 0으로 설정한다. 그림 5(c)에서 레코드가 삭제되어 무효 상태가 되면, 두 번째 비트를 0으로 설정한다. 그림 5(d)에서 유효한 레코드가 수정되면, 세 번째 비트를 0으로 설정하고 이동된 주소 비트에 수정된 레코드가 저장된 컨테이너의 위치를 기록한다.

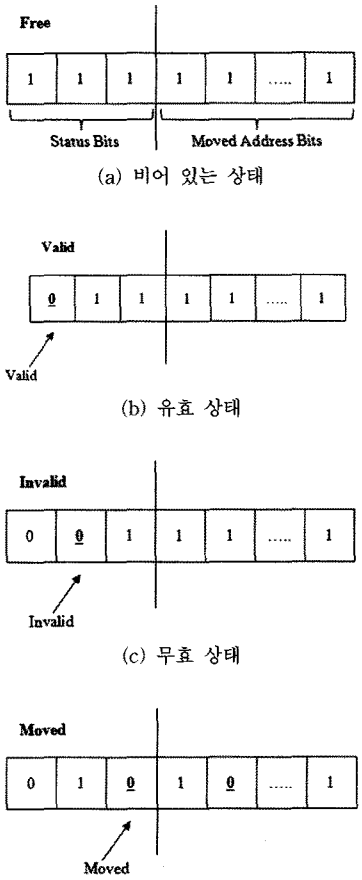


그림 5 컨테이너의 상태 변화에 따른 메타데이타 부분의 변화

이동된 주소 비트는 수정 전 레코드가 저장된 컨테이너에 수정된 레코드가 저장된 컨테이너의 위치를 표시하기 위해 사용된다. 이동된 주소 비트는 하나의 페이지 안에 존재하는 컨테이너를 표시할 수 있을 만큼의 비트로 구성된다. 식 (1)은 이동된 주소 비트의 개수를 구하는 식이다. 예를 들어, 하나의 페이지가 8개의 컨테이너로 구성되어 있을 경우, 이동된 주소 비트는 3비트가 된다.

$$numofmovedaddressbits = \lceil \log_2^{numofcontainer} \rceil \quad (1)$$

제안하는 컨테이너 기반 레코드 관리 방법의 레코드 삽입, 삭제 절차는 다음과 같다. 새로운 레코드 삽입 시, 비어있는 컨테이너에 레코드를 저장한 후, 해당 컨테이너의 상태를 유효 상태로 설정한다. 레코드 삭제 시, 삭제할 레코드가 저장된 컨테이너의 상태를 무효 상태로 설정한다.

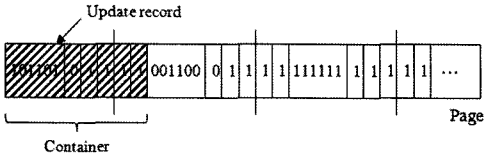
그림 6은 컨테이너 기반 레코드 관리 방법의 레코드 수정 절차를 나타낸다.

2) 제안하는 컨테이너 기반 레코드 관리 방법은 기존의 기법들과 동일하게 페이지 헤더를 사용하여 페이지 내의 레코드의 개수와 같은 페이지 관련 메타데이타들을 관리한다[5].

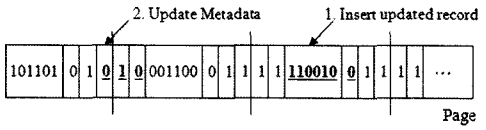
단계 1. 레코드를 수정할 페이지의 비어있는 컨테이너 C_2에 수정된 레코드를 저장하고, 컨테이너 C_2의 상태를 유효 상태로 설정한다.
단계 2. 수정 전 레코드가 저장된 컨테이너 C_1의 상태를 이동된 상태로 설정하고, 컨테이너 C_1의 이동된 주소에 컨테이너 C_2의 위치를 표시한다.

그림 6 제안하는 방법의 레코드 수정 절차

그림 7은 컨테이너 구조를 이용하여 레코드를 수정하는 예를 나타낸다. 그림 7(a)에서 컨테이너 C_0에 저장된 레코드의 수정이 발생하였다. 그림 7(b)에서 수정된 레코드는 비어있는 컨테이너인 컨테이너 C_2에 저장된다. 또한, 컨테이너 C_0의 상태 비트는 이동된 상태로 수정되고, 이동된 주소 비트는 컨테이너 C_2의 위치인 2로 설정된다. 이렇듯, 제안하는 컨테이너 구조를 이용하면, 덮어쓰기를 활용하여 레코드의 삽입, 삭제, 수정을 처리할 수 있다.



(a) 컨테이너 C_0에 레코드 수정 요청



(b) 컨테이너 C_2에 수정된 레코드 삽입 및 컨테이너 C_0의 메타데이타 부분 수정

그림 7 컨테이너 구조를 이용한 레코드 수정

만약 레코드 삽입, 삭제, 수정 시 한 페이지에 수행 가능한 덮어쓰기를 모두 수행하였을 경우, 기존의 페이지를 무효화하고, 해당 페이지의 데이타들을 플래시 메모리의 다른 페이지에 저장한다.

제안하는 컨테이너 기반 레코드 관리 방법은 해당 페이지에 빈 공간이 존재함에도 불구하고, 무효 상태의 컨테이너들이 많이 존재하여 새로운 레코드나 수정된 레코드를 삽입하지 못하는 경우가 발생한다. 이를 해결하기 위해 레코드를 삽입하지 못할 경우, 해당 페이지의 모든 무효 컨테이너들을 비어있는 상태로 설정하고, 해당 컨테이너들의 데이타 부분의 비트를 모두 1로 설정한다.

또한, 이동된 컨테이너와 기존의 컨테이너의 병합을 수행한다. 컨테이너의 병합은 수정된 레코드를 기존의 컨테이너에 복사하여 기존의 컨테이너를 유효 컨테이너로 변경한 후, 수정된 레코드가 저장되어 있던 컨테이너는 비어있는 상태로 설정하고, 해당 컨테이너의 데이타 부분의 비트를 모두 1로 설정한다. 본 논문에서는 이러한 작업을 페이지 압축(page compaction)이라고 부르며, 이를 통해 레코드가 삽입될 컨테이너를 확보할 수 있다.

이와 같이, 제안하는 컨테이너 구조는 제한된 덮어쓰기 연산을 최대한 수행할 수 있으며, 이로 인한 소거 연산의 횟수가 감소하여 성능이 향상된다. 본 논문에서는 실험을 통하여 제안하는 방법의 우수성을 규명한다.

5. 성능 비교 실험

본 장에서는 기존의 레코드 관리 방법과 제안하는 방법 간의 성능 비교를 통하여 제안하는 방법의 우수성을 보인다.

5.1 실험 환경

본 실험에서는 플래시 메모리 개발 프레임 워크의 플래시 메모리 에뮬레이터[8]를 사용한다. 에뮬레이터는 플래시 메모리의 동작을 모방할 뿐 아니라 모의 성능 측정 기능을 제공하여 플래시 메모리 소프트웨어의 개발 및 성능 최적화에 유용하게 사용될 수 있다. 에뮬레이터의 플래시 메모리 용량은 삼성 2G NAND 플래시 메모리[9]를 기준으로 설정한다. 플래시 메모리의 하나의 페이지의 크기는 2K이며, 하나의 블록의 크기는 128KB이다. 한 페이지에 수행 가능한 덮어쓰기 연산의 횟수는 데이타 영역은 2회, 스페어 영역은 3회로 설정한다.

전체적인 실험의 구성은 다음과 같다.[3] 각 실험에서는 5만 개의 레코드를 벌크로드 한 후, 5만 개의 레코드 삽입, 삭제, 수정 연산을 수행한다. 벌크로드를 수행할 때, 각 페이지의 70%만 채운다. 연산에 사용되는 레코드의 키는 1에서 1000만 사이에서 임의로 생성하며, 레코드의 크기는 100KB로 고정한다.

실험의 성능 척도로는 모든 연산을 수행한 후, 페이지 읽기, 쓰기, 소거 연산의 총 횟수를 측정한다. 또한, 읽기, 쓰기, 소거 연산의 횟수에 각각의 가중치(읽기: 1, 쓰기: 16.7, 소거:167)를 곱한 값을 더해 전체 연산의 비용을 계산한다[6].

본 실험은 크게 두 가지로 구성된다. 실험 1에서는 기존의 레코드 관리 방법과 제안하는 방법의 각 연산의 성능을 측정한다. 실험 2에서는 기존의 레코드 관리 방법과 제안하는 방법의 전반적인 성능을 측정한다.

5.2 실험 결과

실험 1. 삽입, 삭제, 수정 연산의 성능 비교

3) 본 논문에서는 실험 설계 과정에서 [10]을 참조하였다.

실험 1에서는 기존의 레코드 관리 방법과 제안하는 방법의 각 연산의 성능을 측정한다. 이를 위하여 삽입, 삭제, 수정 연산을 각각 수행하며 성능을 측정하였다. 실험 1의 결과는 그림 8을 통해 알 수 있다. 그래프 x축은 삽입, 삭제, 수정 연산을 나타내고, y축은 측정한 전체 연산의 비용을 나타낸다.

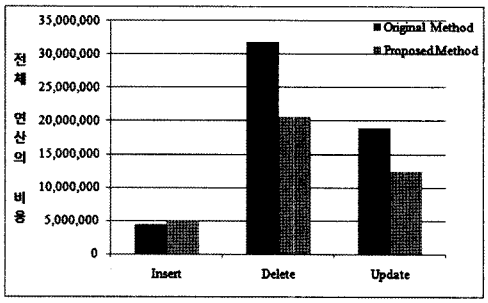


그림 8 각 연산의 성능

실험 결과, 제안하는 방법의 삭제 연산과 수정 연산의 성능은 기존의 레코드 관리 방법에 비해 우수하였다. 이는 기존의 레코드 관리 방법의 경우, 레코드 삭제, 수정 시 덮어쓰기 연산을 거의 수행할 수 없어 이로 인한 소거 연산을 많이 수행하기 때문이다.

반면에, 삽입 연산의 경우, 기존의 레코드 관리 방법의 성능이 제안하는 방법에 비해 우수하였다. 레코드 삽입 시, 기존의 레코드 관리 방법과 제안하는 방법 모두 덮어쓰기 연산이 수행 가능하다. 그러나 제안하는 방법의 경우, 컨테이너를 관리하기 위한 메타데이타 부분이 추가적으로 필요하여 하나의 페이지에 삽입 가능한 레코드의 개수가 기존의 레코드 관리 방법에 비해 적다. 이로 인한 쓰기 연산의 횟수가 많아져 기존의 레코드 관리 방법에 비해 성능이 낮아졌다.

실험 2. 전반적인 성능 비교

실험 2에서는 기존의 레코드 관리 방법과 제안하는 방법의 전반적인 성능을 측정한다. 이를 위하여 전체 연산 중 삭제 연산의 비율을 20%로 고정한 후, 나머지 80%의 삽입, 수정 연산 중 삽입 연산의 비율을 20%, 40%, 60%, 80%로 증가시켜가며 성능을 측정하였다. 실험 2의 결과는 그림 9를 통해 알 수 있다. 그래프 x축은 매개 변수인 삽입 연산의 비율 변화를 나타내고, y축은 측정한 성능 척도를 나타낸다.

그림 9는 삽입 연산의 비율을 증가시켜가며 측정한 전체 연산의 비용 변화를 나타낸다. 실험 결과, 제안하는 방법의 성능이 기존의 레코드 관리 방법에 비해 우수하였다. 이는 제안하는 방법이 덮어쓰기 연산을 많이 수행하여, 이로 인해 소거 연산의 수가 크게 줄었기 때문이다.

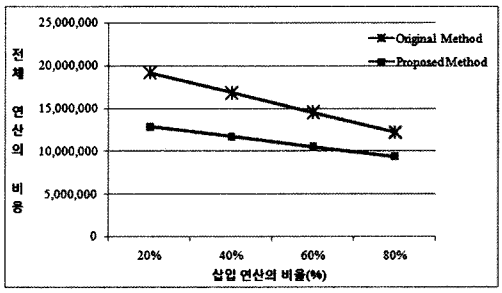


그림 9 삽입 연산의 비율 변화에 대한 실험 결과

6. 결 론

플래시 메모리는 읽기, 쓰기, 소거 연산의 수행 속도의 차이가 크다. 또한, 덮어쓰기 연산은 수행에 제한이 있으며, 이로 인해 소거 연산이 발생한다. 본 논문에서는 이러한 특성을 기반으로 플래시 메모리 환경에서 기존의 레코드 관리 방법의 문제점을 분석하고, 이를 바탕으로 새로운 레코드 관리 방법을 제안하였다.

본 논문의 공헌은 다음과 같다. 첫째, 플래시 메모리의 물리적인 특성이 기존의 레코드 관리 방법에 미치는 영향을 분석하고, 플래시 메모리 환경에서 나타나는 기존의 레코드 관리 방법의 문제점을 지적하였다. 둘째, 분석한 내용을 기반으로 플래시 메모리 환경에 적합한 레코드 저장 구조인 컨테이너 구조를 제안하고, 이를 이용한 새로운 레코드 관리 방법을 제안하였다. 셋째, 실험을 통해 제안하는 방법의 우수성을 규명하였다.

참 고 문 헌

[1] J. Jeong , S. Noh, S. Min and Y. Cho, "A Design and Implementation of Flash Memory Simulator," *Journal of Korean Information Science: C*, Vol.8, No.1, pp. 36-45, 2002.

[2] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," *ACM Computing Surverys*, Vol.37, No.2, pp. 138-163, 2005.

[3] C. Wu, L. Chang, and T. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," In *Proc. Int'l. Conf. on Real-Time and Embedded Computing Systems and Applications*, RTCSA, Vol. LNCS 2968, pp. 409-430, 2003.

[4] S. Lee and B. Moon, "Design of Flash-Based DBMS: An In-Page Logging Approach," In Proc. *ACM Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 55-66, 2007.

[5] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1995.

[6] K. Yim, "A Novel Memory Hierarchy for Flash Memory Based Storage Systems," *Journal of Semiconductor Technology and Science*, Vol.5,

No.4, pp. 262-269, 2005.
[7] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," In *Proc. USENIX Technical Conf. on Unix and Advanced Computing Systems*, pp. 155-164, 1995.
[8] S. Kim et al., "A Development Framework for Reliable Flash Memory Software," *SK Telecommunications Review*, Vol.15, No.4, pp. 638-646, 2005.
[9] Samsung, 2G NAND Flash Memory, http://www.samsung.com/products/semiconductor/NANDFlash/, 2008.
[10] J. Rao and K. Ross, "Making B+-Trees Cache Conscious in Main Memory," In *Proc. ACM Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 475-486, 2000.

배 덕 호

2006년 2월 한양대학교 정보통신대학 컴퓨터전공(학사). 2008년 2월 한양대학교 전자컴퓨터통신공학과(석사). 2008년 3월~현재 한양대학교 전자컴퓨터통신공학과(박사과정). 관심분야는 임베디드 DBMS, 플래시 메모리 DBMS, 이동 객체 DBMS, 데이타 마이닝, 사회연결망분석

김 상 욱

1989년 2월 서울대학교 컴퓨터공학과(학사). 1991년 2월 한국과학기술원 전산학과(석사). 1994년 2월 한국과학기술원 전산학사(박사). 1991년 7월~1991년 8월 미국 Stanford University, Computer Science Department, 방문 연구원. 1994년 3월~1995년 2월 KAIST 정보전자연구소 전문 연구원 1999년 8월~2000년 8월 미국 IBM T.J. Watson Research Center, Post-Doc. 1995년 3월~2003년 2월 강원대학교 정보통신공학과 부교수. 2003년 3월~현재 한양대학교 정보통신대학 정보통신학부 교수. 관심분야는 데이타베이스 시스템, 저장 시스템, 트랜잭션 관리, 데이타 마이닝, 멀티미디어 정보 검색, 공간 데이타베이스/GIS, 주기억장치 데이타베이스, 이동 객체 데이타베이스/텔레매틱스, 사회 연결망 분석, 웹 데이타 분석

장 지 웅

1993년 2월 연세대학교 전산과학과 졸업(학사). 1995년 2월 한국과학기술원 전산학과 졸업(석사). 2001년 8월 한국과학기술원 전자전산학과(전산학 전공) 졸업(박사). 2001년 9월~2002년 2월 첨단정보기술연구센터 연수 연구원. 2002년 3월~현재 한국산업기술대학교 게임공학과 부교수. 관심분야는 데이타베이스 시스템, 저장 시스템, 트랜잭션 관리, GIS, 주기억장치 데이타베이스, 플래시메모리 데이타베이스