플래시 메모리 환경을 위한 이단계 인덱싱 방법

(A Two-level Indexing Method in Flash Memory Environment)

김 종 대 † 장 지 웅 ††
(Jong-Dae Kim) (Ji-Woong Chang)
황 규 정 ††† 김 상 욱 ††††
(Kyu-Jeong Hwang) (Sang-Wook Kim)

요 약 최근 플래시 메모리 용량이 증가함에 따라 대용량의 데이타를 빠르게 검색하기 위한 효율적인 인덱싱 방법의 필요성이 증가하였다. 플래시 메모리는 기존 저장매체와 다른 여러 가지 하드웨어적인 특성이 있다. 특히, 쓰기 연산과 소거 연산은 비용이 매우 크고, 덮어쓰기 연산이 불가능하다. 본 논문에서는 플래시 메모리에 저장되는 데이타에 대해여 발생하는 잦은 쓰기 연산을 감소시켜 다양한 연산을 효율적으로 처리하는 인덱스 구조를 제안한다. 본 논문에서는 성능 평가를 통해 제안하는 인덱싱 방법의 우수성을 보인다.

키워드 : 플래시 메모리 DBMS, 인덱싱 방법

Abstract Recently, as the capacity of flash memory increases rapidly, efficient indexing methods become crucial for fast searching of a large volume of data stored in flash memory. Flash memory has its unique characteristics: the write operation is much more costly than the read operation and in-place updating is not allowed. In this paper, we propose a novel index structure that significantly reduces the number of write operations and thus supports efficient searches, insertions, and deletions. We verify the superiority of our method by performing extensive experiments.

Key words : Flash memory DBMS, Indexing Method

1. 서 론

플래시 메모리는 비휘발성 저장 장치로 가볍고 물리적인 충격에 강하며 전력 소비가 작아서 모바일 기기 등에서 데이타 저장 장치로 널리 사용되고 있다[1]. 플래시 메모리가 많이 사용되고 플래시 메모리의 용량이 증가함에 따라 플래시 메모리에 저장된 대용량의 데이타에서 원하는 데이타를 빠르게 검색하기 위한 효율적인 인덱스 구조의 필요성도 함께 증가하고 있다.

플래시 메모리는 기존의 하드 디스크와는 다른 플래시 메모리만의 고유한 특성을 가진다. 플래시 메모리에서 읽기와 쓰기 연산은 페이지 단위로 이루어지는데 빠른 읽기 연산 속도에 비해서 쓰기 연산의 속도가 매우 느리고 덮어쓰기(in-place update) 연산이 지원되지 않는다[2,3]. 플래시 메모리에서 덮어쓰기 연산을 수행하기 위해서는 기존의 데이타를 무효화하고, 갱신 사항이 반영된 새로운 데이타를 플래시 메모리의 다른 영역에 기록하는 방법을 사용한다[2]. 무효화된 페이지를 다시 사용하기 위해서는 먼저 소거(erase) 연산을 수행하여야 한다.

소거 연산은 연속된 여러 페이지로 구성된 블록 단위로 이루어지는데 어떤 블록에 대하여 소거 연산을 수행하기 위해서는 먼저 해당 블록에 저장된 유효한 데이타를 다른 블록으로 복사한 후 소거 연산을 수행하여야 하므로 플래시 메모리 연산 중 가장 많은 비용을 요구한다[4]. 소거 연산은 오버헤드가 매우 큰 연산으로 잦은 쓰기 연산에 의해 발생한다[5]. 따라서 플래시 메모리 환경에서는 쓰기 연산을 감소시키는 것이 매우 중요하다.

위와 같은 플래시 메모리의 특성으로 인하여 기존에 널리 사용되는 인덱스 방법인 B+트리 구조는 플래시 메모리 환경에서는 적합하지 않다. 디스크 기반 DBMS에서 주로 사용되는 트리 형태의 인덱스 구조는 레코드를 삽입하거나 삭제할 때 트리 구조의 균형을 유지하기 위해서 추가적인 쓰기 연산이 발생할 수 있기 때문이다.

인덱스 구조의 인덱스 페이지는 인덱스 엔트리들을 저장하는 단위이며 인덱스 페이지의 집합이 인덱스이다.

· 본 연구는 2007년도 정부(과학기술부)의 재원으로 한국과학재단(R01-2007-000-11773-0)과 2007년도 정부재단(교육인적자원부 학술연구조성사업비)으로 학술진흥재단(KRF-2007-314-D00221) 및 지식경제부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업(IITA-2008-C1090-0801-0040)의 연구비 지원을 받았습니다.
· 이 논문은 제34회 추계학술대회에서 '플래시 메모리 환경을 위한 이단계 인덱싱 방법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 한양대학교 전자컴퓨터통신 연구원
jdkim@agape.hanyang.ac.kr
†† 정 회 원 : 한국산업기술대학교 게임공학과 교수
jwchang@kpu.ac.kr
††† 정 회 원 : 한양대학교 전자컴퓨터통신 연구원
lingohkc@agape.hanyang.ac.kr
†††† 종신회원 : 한양대학교 전자컴퓨터통신 교수
wook@hanyang.ac.kr

논문접수 : 2008년 1월 3일
심사완료 : 2008년 6월 10일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제7호(2008.10)

인덱스 페이지는 다수의 인덱스 엔트리들을 포함하며, 인덱스 엔트리는 <레코드의 키 값, 레코드의 위치 정보>형식으로 구성된다.

본 논문에서는 플래시 메모리 환경에서 효율적으로 동작하는 새로운 인덱싱 방법을 제안한다. 제안하는 인덱싱 방법에서는 플래시 메모리의 인덱스 페이지에 인덱스 엔트리들은 키 값에 의해 정렬된 순서로 저장하고, 메인 메모리에 파일 맵 형태로 존재하는 디렉토리를 이용하여 인덱스 페이지들에 대한 정렬된 순서로 관리한다. 제안하는 인덱싱 방법은 레코드의 삽입, 삭제에 의한 인덱스 구조의 동적인 변경을 최소화하여 인덱스 구조 변경으로 인한 플래시 메모리의 쓰기 연산을 최소화하였다.

2. 연구 배경

플래시 메모리는 물리적으로 기존의 저장 매체와 다른 특성을 가진다. 플래시 메모리는 읽기, 쓰기, 소거 연산 간에 수행 시간의 편차가 크고, 덮어쓰기 연산을 지원하지 않는다. 본 장에서는 이러한 플래시 메모리 고유의 특성이 기존의 인덱싱 방법에 미치는 영향에 대해서 논의한다.

기존의 디스크 기반의 DBMS에서 가장 대표적인 인덱스 구조로는 B+트리가 있다. B+트리는 디스크의 입출력이 용이하고 확장성이 뛰어나 현재 디스크 기반 DBMS에서 가장 널리 쓰이는 인덱스 구조이다. 그림 1은 B+트리 인덱스 구조의 예를 나타낸 것이다.

그림 1에서 하나의 B+트리 노드에 저장되는 인덱스 엔트리의 개수는 4개이므로 각 노드에는 최소한 2개 이상의 인덱스 엔트리가 저장되어야 한다. 만일 어떤 노드에 50% 미만의 인덱스 엔트리만이 남아있게 되면 해당 노드는 이웃 노드와 병합되거나 재분배를 수행한다. 반대로 어떤 노드가 가득차서 더 이상 인덱스 엔트리를 삽입할 수 없는 경우에는 분할을 수행한다. 분할과 병합은 상위 노드에 영향을 미쳐서 연쇄적인 분할 또는 병합이 발생할 수 있으며, 이 때 인덱스의 구조가 유기적으로 변하게 된다.

플래시 메모리에서 B+트리를 그대로 사용하는 경우, 레코드가 삽입 또는 삭제될 때 플래시 메모리 페이지는 비록 여유 공간이 존재하더라도 동일한 페이지에 덮어쓸 수 없기 때문에 새로운 페이지를 할당하여 해당 노드의 변경된 상태를 저장해야 한다. 이 때 분할 또는 병합 현상이 발생하면 이는 상위 노드에 영향을 미친다. 즉, 상위 노드에 대응되는 플래시 페이지에서도 새로운 페이지의 할당 및 복사 연산이 이루어져야 한다. B+트리에서는 이러한 분할과 병합이 리프 노드로부터 루트 노드에 이르기까지 연쇄적으로 발생할 수 있다. 이와 같이 데이타의 삽입과 삭제로 잦은 덮어쓰기를 유발시키는 B+트리는 플래시 메모리의 물리적 특성으로 인하여 인덱스에서 발생하는 다양한 연산을 수행함에 있어서 디스크 기반의 DBMS에서와 같은 성능을 기대하기 어렵다.

플래시 메모리를 물리적 기반으로 하는 인덱스 구조로는 B-트리를 플래시 메모리의 특성에 맞게 개선한 BFTL[6]이 있다. BFTL에서는 인덱스 엔트리들을 정렬된 순서로 저장하지 않는다. 그러므로 논리적인 인덱스 노드와 물리적인 인덱스 페이지 간에 불일치가 발생한다. 이러한 문제를 해결하기 위해서 별도의 자료구조를 사용한다.

첫째, 동일한 인덱스 노드에 속하는 인덱스 엔트리들의 물리적 저장 위치를 식별할 수 있도록 노드 변환 테이블을 이용한다. 노드 변환 테이블은 메인 메모리에 상주하며 노드의 수가 증가함에 따라 메인 메모리의 사용량도 증가하고 검색연산 시 다수의 서로 다른 페이지를 접근해야 하기 때문에 검색 성능은 감소한다.

둘째, 플래시 메모리에 저장된 인덱스 엔트리들은 각기 자신이 소속된 노드를 나타내는 추가 정보를 포함한다. 만약, 인덱스 노드의 분할 또는 합병이 발생하면 이 추가 정보로 인하여 다수의 페이지에 분산되어 저장된

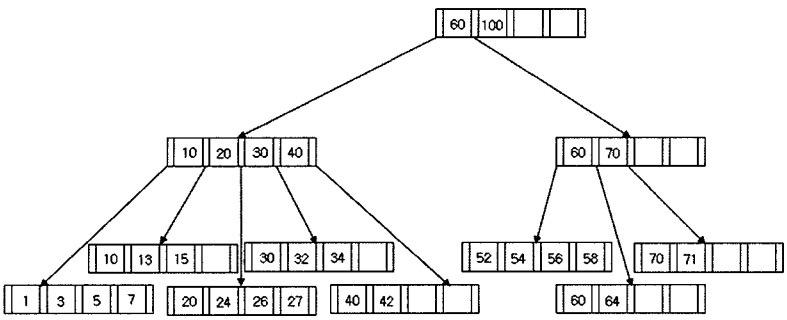

그림 1 B+트리 인덱스의 예

인덱스 엔트리들의 갱신이 유발된다. 그 결과 대상 인덱스 엔트리들을 포함하는 모든 플래시 페이지들은 덮어쓰기 연산이 발생한다. 플래시 메모리 환경에서의 B-트리를 구현하기 위해 BFTL 기법이 제안되었으나 BFTL 구조상 많은 메인 메모리의 사용이 필요하며, 검색 시 많은 읽기 연산이 발생하여 검색 시간이 오래 걸린다는 단점이 있다.

이와 같이 플래시 메모리 환경에서 B-트리와 BFTL은 인덱스 구조로서 보안해야할 부분이 있다. 본 논문에서는 플래시 메모리 환경에서 효율적으로 동작하는 새로운 인덱싱 방법을 제안한다.

3. 이단계 인덱싱 방법

본 논문에서는 플래시 메모리의 특성을 고려하여 쓰기 연산을 최소화하는 인덱싱 방법를 제안한다. 제안하는 인덱싱 방법은 인덱스 엔트리가 저장되는 플래시 메모리 페이지들을 저장된 인덱스 엔트리의 키 값에 의해 정렬하여 관리하며 인덱스 엔트리의 삽입, 삭제에 의한 인덱스 구조의 동적인 변경을 최소화하고, 구조의 변경이 발생하더라도 국지적으로 일어나도록 하여 플래시 메모리의 쓰기 연산을 횟수를 최소화 하였다.

제안하는 인덱스 구조는 크게 인덱스 페이지와 디렉토리로 구성된다. 그림 2는 제안하는 인덱스 구조의 예이다.

인덱스에서 논리적으로 연속적인 인덱스 페이지들이 플래시 메모리 내에서 물리적으로 연속적일 필요는 없다. 각각의 인덱스 페이지는 포함하는 인덱스 엔트리의 수에 따라서 빈 공간이 존재할 수 있으며, 인덱스 페이지 내의 인덱스 엔트리들은 키 값의 크기에 따라 정렬되어 저장된다.

제안하는 인덱스 구조의 디렉터리는 인덱스 페이지를 논리적 순서에 따라 직접 접근이 가능하도록 하는 역할을 담당한다. 디렉터리를 파일 맵의 형태로 파일 시스템이 제공한다. 단, 이 디렉터리는 효율적인 검색을 위하여 중간에 삽입, 삭제가 가능하다. 디렉토리는 다수의 디렉토리 엔트리를 포함하며, 디렉토리 엔트리는 <인덱스 페이지에 저장된 최소 키 값, 인덱스 페이지의 위치 정보>형식으로 구성된다.

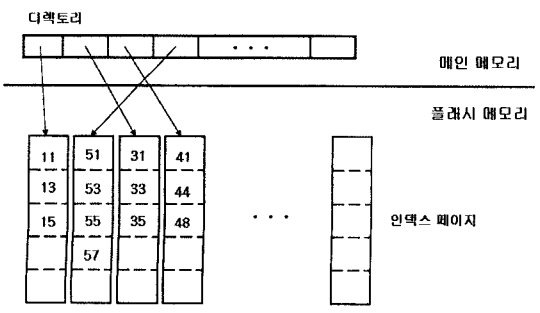

그림 2 제안하는 인덱스 구조의 예

제안하는 인덱스의 디렉토리는 메인 메모리에 존재하기 때문에 인덱스 엔트리의 삽입, 삭제에 의한 플래시 메모리의 쓰기 연산을 최소화 하였다. 디렉토리를 구성하는 엔트리는 키 값과 인덱스 페이지에 대한 식별자(4Byte)로 구성된다. 하나의 디렉토리 엔트리는 하나의 인덱스 페이지와 대응되므로 디렉토리의 전체 크기는 디렉토리 엔트리의 크기와 인덱스 페이지의 개수의 곱이 되며, 일반적인 경우 그 크기는 별로 크지 않다. 그러나 인덱스 페이지가 매우 많은 경우에는 검색 성능을 약간 희생하더라도 디렉토리에 인덱스 페이지에 대한 키 값을 저장하지 않고 인덱스 페이지에 대한 식별자만을 저장하는 방법으로 디렉토리를 위한 메인 메모리 사용량을 감소시킬 수 있다. 초기 시스템 시작 시 디렉토리 구성은 시스템을 종료할 때에 플래시 메모리에 저장한 디렉토리를 메인 메모리로 읽어 들이는 방법으로 구성한다.

따라서 제안하는 인덱스 구조는 갱신 연산이 빈번하나 전체 데이타의 양이 많지 않은 모바일 기기와 같은 환경에서 유용하다.

제안하는 인덱싱 방법의 인덱스 엔트리 삽입 수행 절차는 다음과 같다.

단계 1. 삽입할 인덱스 엔트리의 키 값을 가지고 디렉토리를 통해서 인덱스 엔트리가 저장될 인덱스 페이지를 검색한다.

단계 2. 검색된 인덱스 페이지에 삽입할 인덱스 엔트리를 저장할 충분한 공간이 있을 경우, 해당 인덱스 페이지에 인덱스 엔트리를 저장한다.

단계 3. 검색된 인덱스 페이지에 인덱스 엔트리를 저장할 수 없는 경우, 새로운 인덱스 페이지를 할당 받아 인덱스 엔트리들을 분할하여 저장하고 디렉토리에 반영한다.

제안하는 인덱싱 방법은 플래시 메모리 환경에서 다음과 같은 구조적인 특징을 갖는다. 첫째, 플래시 메모리의 물리적 특성을 고려하여 인덱스 엔트리의 삽입, 삭제 연산에 따라 유기적으로 인덱스 구조가 변경되지 않으므로 플래시 페이지의 쓰기 연산과 소거 연산의 횟수를 감소시킨다. 둘째, 플래시 메모리에 저장된 데이타에 대하여 인덱스 엔트리들을 효과적으로 클러스터링 함으로써 빠른 검색을 지원한다.

삽입, 삭제 연산에 따라 유기적으로 변경되지 않는 인덱스 구조를 사용하는 경우에는 삽입, 삭제 연산이 발생할 때마다 즉시 인덱스에 반영하는 것 보다는 모아서

일괄 처리한다면 성능을 개선 할 수 있다. 동일한 인덱스 페이지에 속하는 인덱스 엔트리들에 대하여 삽입, 삭제 연산이 반복적으로 발생하는 경우 이 연산의 결과를 일괄적으로 인덱스에 반영함으로써 쓰기 연산의 횟수를 감소시킬 수 있기 때문이다.

메인 메모리에 구성된 이단계 인덱싱 방법의 디렉토리와 2장 연구 배경에서 언급한 BFTL의 노드 변환 테이블을 비교하면 디렉토리는 하나의 유효한 인덱스 페이지당 하나의 디렉토리 엔트리가 고정적으로 구성된다. 반면에 BFTL의 노드 변환 테이블의 경우 BFTL을 구성하는 노드의 수와 하나의 노드에 속하면서 분산 저장된 페이지의 개수의 평균을 곱한 크기만큼 메인 메모리를 사용하여 구성된다. BFTL은 노드 변환 테이블의 링크 수를 제한하는 방법으로 그 크기를 조절하고 있지만 이단계 인덱싱 방법의 디렉토리와 비교하여 메인 메모리 사용면에서 보다 비효율적이라고 할 수 있다.

결과적으로 제안하는 인덱싱 방법은 플래시 메모리 환경의 물리적인 특성을 고려하여 균형 잡힌 트리 형태의 인덱스 구조를 지양하고, 인덱스 페이지에 여분 공간을 허용함으로써 분할, 병합, 재분배의 수행 횟수를 감소시키는 새로운 인덱스 구조를 제안하였다.

4. 성능 평가

본 장에서는 플래시 메모리 환경에서 B+트리 인덱스와 제안하는 인덱싱 방법에 대하여 실험을 통한 성능 분석을 수행하여 본 논문에서 제안하는 인덱싱 방법의 우수성을 규명한다.

4.1 실험 환경

실험은 플래시 메모리 개발 프레임 워크의 플래시 메모리 애뮬레이터[7]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. 삼성 2 G byte NAND 플래시 메모리를 기준으로 에뮬레이터 상의 플래시 메모리 용량을 설정한다.

연산에 사용되는 인덱스 엔트리의 키 값은 1에서 1000만 사이에서 임의로 생성하되 중복되지 않고 균등 분포에 따라 생성한다.

실험의 성능 척도로는 모든 연산을 수행한 후, 페이지 읽기, 쓰기, 소거 연산의 총 횟수를 측정한다. 또한, 읽기, 쓰기, 소거 연산의 횟수에 각각의 가중치를 곱한 값을 더해 전체 연산의 비용을 계산한다. 이 때, 쓰기 연산은 13, 소거 연산은 130의 가중치를 부여한다[4].

4.1 실험 결과

본 실험에서는 플래시 메모리 환경에서 동작하는 B+트리와 제안하는 인덱싱 방법의 검색, 삽입, 삭제 성능을 측정한다. 그림에서 그래프의 x축은 실험을 수행한 데이타의 수를 나타내고, y축은 측정한 성능 척도를 나타낸다.

그림 3은 검색 연산의 횟수를 증가 시켜가며 측정한 전체 연산의 비용을 나타낸 것이다. 전체 연산이 읽기 연산의 횟수만 나온 것은 검색 연산 시 쓰기, 소거 연산이 일어나지 않기 때문이다. 검색 연산 시 수행된 성능의 차이는 제안하는 인덱싱 방법이 B+트리에 비해 약 3배 정도 우수하다. 제안하는 인덱싱 방법과 B+트리와의 성능차이는 B+트리의 경우 인터널 노드들이 플래시 메모리에 존재하기 때문에 각 인터널 노드들을 접근하는 비용이 발생한다. 반면에 제안하는 인덱싱 방법은 디렉토리를 검색하여 인덱스 페이지에 접근하므로 검색 성능이 우수하다. 검색 성능의 결과는 B+트리의 높이에 따라 성능의 차이가 발생한다. 본 실험에서 벌크로드 후의 B+트리의 높이는 3으로 제안하는 인덱싱 방법에 비해 약 3배의 성능 차이를 보인다.

그림 4는 삽입 연산의 수를 증가시켜가며 측정한 전체 연산의 비용을 나타낸 것이다. 제안하는 인덱싱 방법이 B+트리에 비해 삽입 성능이 약 14.5% 우수하다. 그 이유로는 B+트리의 경우 리프 노드의 분할에 의해서 상위 노드의 추가적인 분할이 일어날 수 있다. 반면에 제안하는 인덱싱 방법은 B+트리에서의 리프 노드에 해당하는 인덱스 페이지의 분할에 의해 상위 노드의 분할과

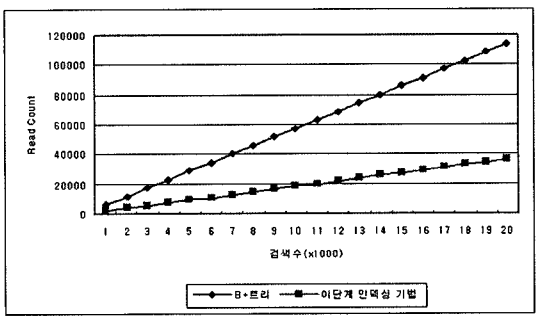

그림 3 검색 시 수행된 읽기 연산 횟수

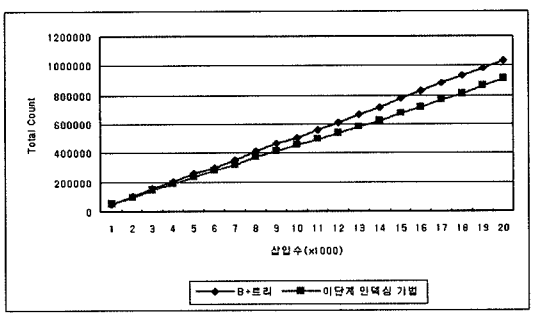

그림 4 삽입 연산 시 수행된 총비용

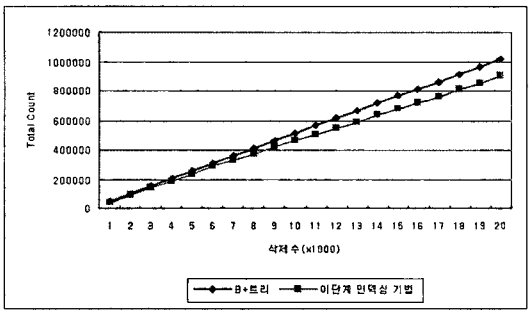

그림 5 삭제 연산 시 수행된 총 비용

같은 추가적인 비용이 발생하지 않는다. 삽입 성능의 결과는 B+트리의 높이가 커질수록 제안하는 인덱싱 방법과의 성능 격차는 커진다.

그림 5는 삭제 연산의 수를 증가시켜가며 측정한 연산의 비용을 나타낸 것이다. 이 경우 성능의 결과와 원인은 삽입 연산의 경우와 매우 비슷하다. 단, 삽입 연산의 분할 비용에 비해서 삭제 연산의 병합 비용이 적기 때문에 총 비용이 삽입 실험의 비용을 넘지는 않는다.

5. 결 론

최근 들어, 플래시 메모리는 모바일 기기의 저장 장치뿐만 아니라 하드 디스크를 대체할 매체로 주목 받고 있다. 플래시 메모리의 용량이 증가함에 따라 플래시 메모리에 저장된 데이타를 효율적으로 검색하기 위한 인덱싱 방법이 요구된다. 하지만 플래시 메모리의 물리적인 특성으로 인해 기존의 방법을 그대로 적용하면 심각한 성능 저하를 가져올 수 있다. 따라서 본 논문에서는 플래시 메모리 환경에서 효율적으로 동작하는 새로운 인덱싱 방법을 제안하였다. 제안하는 인덱싱 방법은 인덱스 페이지를 논리적인 순차로 구성하여 인덱스 구조의 동적인 변경을 최소화하여 인덱스 구조 변경에 따른 플래시 메모리의 쓰기 연산의 횟수를 줄여서 플래시 메모리 환경에서 효율적으로 동작하는 인덱싱 방법을 제안하였다.

참 고 문 헌

[1] M. Wu and W. Zwaenpoel, "eNYy: A Non-Volatile, Main Memory Storage System," In *Proc. ACM Symp. on Architectural Support for Programming Languages and Operating System*, ACM ASPLOS, pp. 86-87, 1994.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Samsung, 2G NAND Flash Memory, http://www.samsung.com/products/semiconductor/NANDFlash/, 2007.