플래시 메모리 저장 장치에서 효율적인 M-트리 기반의 인덱싱 구현

(An Implementation of Efficient M-tree based Indexing on Flash-Memory Storage System)

유 정 수 † 낭 종 호 ††
(Jeongsoo Yu) (Jongho Nang)

요 약 최근 플래시 메모리의 용량이 빠른 속도로 증가하면서 휴대 기기 환경에서 대량의 멀티미디어 데이터를 저장하는 것이 가능하게 되었다. 따라서 플래시 메모리 상에서 인덱스 구조를 통한 데이터 관리 기법이 필요하게 되었다. 여러 인덱싱 방법 중 M-tree는 고차원 거리 공간에 적합하기 때문에 멀티미디어 데이터의 특징 데이터에 대한 인덱싱 방법으로 가장 많이 쓰이고 있다. 그러나 플래시 메모리는 쓰기 연산의 제한을 갖기 때문에, 잦은 쓰기가 발생하는 트리 구조의 인덱싱을 구축 시 심각한 성능 저하가 발생한다. 본 논문에서는 플래시 메모리 상에서 M-tree를 구현함에 있어서 노드 분할 방법을 통하여 쓰기 연산의 횟수를 감소시켜 입출력 성능을 향상시키는 방법을 제안하였다. 실험에 의하면 쓰기 횟수를 약 7%정도로 현저히 감소시킨 것으로 나타났다. 본 논문에서 제안한 방법을 사용하여 플래시 메모리 상에서 대량의 데이터에 대한 인덱싱을 효율적으로 구축할 수 있을 것이다.

키워드 : 플래시 메모리, 인덱싱, M-tree

· 본 연구는 지식경제부 및 한국산업기술평가관리원의 IT산업원천기술개발사업의 일환으로 수행하였음[2009-KI002090, 신뢰성 컴퓨팅(Trustworthy Computing) 기반 기술 개발]
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '플래시 메모리 저장 장치에서 효율적인 M-tree 기반의 인덱싱 구현'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서강대학교 컴퓨터공학과
tool700@korea.com
†† 종신회원 : 서강대학교 컴퓨터공학과 교수
jhnang@sogang.ac.kr

논문접수 : 2009년 8월 13일
심사완료 : 2009년 11월 6일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제1호(2010.1)

Abstract As the storage capacity of the flash memories increased portable devices began to store mass amount of multimedia data on flash memory. Therefore, there has been a need for an effective data management scheme by indexing structure. Among many indexing schemes, M-tree is well known for it's suitability for multimedia data with high dimensional matrix space. Since flash memories have writing operation restriction, there is a performance limitation in indexing scheme with frequent write operation. In this paper, a new node split method with reduced write operation for m-tree indexing scheme in flash memory is proposed. According to experiments the proposed method reduced the write operation to about 7% of the original method. The proposed method will effectively construct an indexing structure for multimedia data in flash memories.

Key words : Flash Memory, Indexing, M-tree

1. 서 론

최근 플래시 메모리는 PMP, 휴대폰, PDA, MP3 플레이어, 디지털 카메라와 같은 휴대용 기기에서 가장 많이 쓰이는 저장 매체이다. 이는 플래시 메모리의 장점인 저전력성, 비휘발성, 물리적 안정성 및 휴대성 등이 휴대 기기의 저장 장치로 매우 적합하기 때문이다. 그러나 플래시 메모리는 덮어 쓰기 연산을 지원하지 못한다는 단점을 가진다. 그래서 플래시 메모리의 덮어 쓰기는 이전 데이터를 빈 페이지에 쓰고 이전 데이터가 쓰여져 있던 페이지를 무효화시키거나 소거 연산을 적용시켜야 하는 과정이 필요하다. 특히 소거 연산은 그 비용이 매우 크고 적용 횟수 또한 수백만 번 정도로 제한되며, 이러한 제약들로 인하여 플래시 메모리는 쓰기 연산이 읽기 연산에 비하여 제약적이다.

한편 플래시 메모리는 MLC(multi-level-cell) 기법의 개발로 인하여 빠른 속도로 대용량화 되고 있으며, 이로 인하여 휴대 기기 환경에서 대량의 데이터를 저장하는 일이 점점 일반화 되어가고 있는 추세이다. 따라서 휴대 기기 상에서 DBMS 또는 인덱스 구조를 통한 대량의 데이터 관리 기법의 중요성이 증가하고 있다. 그 동안 수많은 종류의 인덱싱 방법이 개발되어 왔으며, 좌표 공간에 존재하는 데이터에 대한 검색을 목적으로 하는 공간 접근 방법(SAM, spatial access method)과 거리 공간에 대한 방법인 거리 접근 방법(MAM, metric access method)의 두 카테고리로 나눌 수 있다. MAM 기반 방법들은 좌표 시스템을 요구하지 않으며 상대 거리만을 이용하므로 고차원 거리 공간에 존재하는 경우가 많은 멀티미디어 데이터에 대한 특징 벡터에 대한 인덱싱에 주로 사용되며 [1-3]등과 같이 전체 데이터 분포의 특성을 이용하는 정적 방법과 [4-6]등과 같이 삽입

및 삭제가 자유로운 동적 방법으로 나뉜다. 동적 MAM 방법은 가장 포괄적이라는 특징을 가지며 최근 고차원 거리 공간에서의 인덱싱 방법으로 가장 많이 쓰이고 있다. 동적 MAM 방법의 대표적인 인덱스 구조인 M-tree[4]는 bottom-up방식의 균형된 트리 형태이며 고정 크기의 노드 구조를 통해 입출력의 최적화에 초점을 맞추었다는 특성을 갖는다.

M-tree와 같은 균형된 트리 형태의 인덱스 구조는 리프 노드의 갱신이 루트 방향으로 전달되는 형식으로 성장하며 따라서 리프에서 루트까지의 경로상의 잦은 갱신으로 인한 쓰기 비용의 증가 현상이 발생한다. JFFS3[7]에서는 이러한 현상을 “wandering tree”라고 불렀으며, 이러한 현상은 bottom-up 방식의 균형된 트리 형태의 인덱스 구조를 플래시 메모리 상에서 구축 시 성능 저하의 주 요인이 될 수 있다.

본 논문에서는 M-tree를 플래시 메모리 상에서 구축 시 쓰기 횟수를 감소시킴으로써 전체 구축 성능을 향상시키는 방법을 제안하였다. 제안한 방법의 키 아이디어는 M-tree의 노드 갱신 연산의 대부분이 실제로 노드의 일부분만 변경된다는 부분성(partiality)을 이용한 것이다. 즉, 논리적 노드를 작은 조각으로 나누어 출력 영역을 줄이고 갱신 되는 노드 조각을 물리적으로 최대한 같은 페이지에 모아서 출력함으로써 페이지 덮어 쓰기 연산을 줄이는 것을 목적으로 한다.

본 논문의 구성은 다음과 같다. 2장에서는 M-tree의 기본 구조와 플래시 메모리의 특성에 대하여 설명하며, 3장에서는 노드 분할 방법을 통하여 쓰기 횟수를 감소시키는 방법에 대하여 설명한다. 4장에서는 제안한 방법에 대한 실험 결과에 대하여 분석하고, 5장에서는 본 연구의 결론을 내린다.

2. 관련 연구

2.1 M-tree 구조

M-tree는 좌표 공간에 제한하지 않으며 bottom-up 방식으로 트리가 성장하여 동적 삽입, 삭제가 가능하고 균형된 트리 구조와 고정 사이즈의 노드 크기 등 입출력의 최적화에 초점을 맞추었다는 특성을 가지므로 기존의 방법들에 비하여 적합하다고 볼 수 있다. M-tree는 좌표 계를 정의하기 힘든 거리 공간을 지원하기 위하여 피벗과 반지름(covering radius)을 사용한다. 또한 M-tree의 각 노드는 디스크 기반의 인덱싱을 고려하여 고정된 크기를 갖는다. 각 노드는 정해진 양(M 파라미터) 만큼의 엔트리를 포함한다. 노드는 내부 노드와 리프 노드의 두 가지 타입을 갖는다. 그림 1은 M-tree의 기본 구조를 나타내는 예시이다.

리프 노드에 삽입된 엔트리가 노드의 반지름을 변경

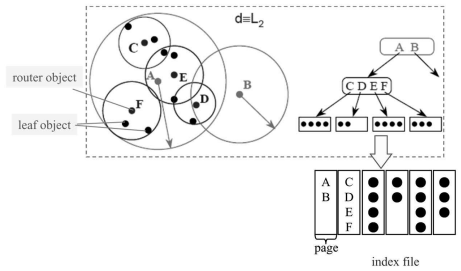


그림 1 M-tree의 기본 구조

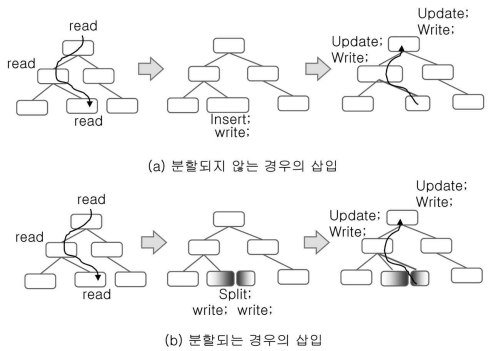


그림 2 M-tree 노드의 쓰기 연산

표 1 플래시 메모리 상에서 M-Tree 삽입 비용
(MLC-2 SD Card, 10,000 오브젝트, L1-256)

총 삽입 시간	쓰기 시간	읽기 시간	거리 계산 시간
2,608,305ms	2,395,703ms	79,953ms	15,724ms

할 경우 상위 노드에 존재하는 포인터 엔트리의 r^c 값이 변경되고, 이와 같은 변경은 루트 노드까지 위 방향으로 발생한다. 또한 리프 노드에서 넘침 현상이 발생할 경우 노드는 분할하며 부모 노드에는 기존의 포인터 엔트리가 새로 분할된 두 노드에 대한 포인터 엔트리로 대체된다. 이때도 마찬가지로 다시 부모 노드가 넘침 현상이 발생할 경우 분할되며 루트 노드까지 위 방향으로 전달된다. 이와 같이 M-tree는 트리 기반 인덱싱이므로 그림 2와 같이 리프의 변경으로 인한 경로 상의 노드들에 빈번한 입출력이 발생한다. 표 1은 MLC-2 4G SD카드에서 L1-256차원의 데이터 10,000개에 대하여 M-tree의 총 삽입 시간을 측정한 것으로서, 총 삽입 시간의 약 85%정도가 쓰기 시간이 차지한다.

2.2 플래시 메모리의 특성

플래시 메모리는 블록이란 단위로 구성된다. 또한 블록은 페이지(섹터라고도 불리움) 단위로 구성되며 읽기, 쓰기는 페이지 단위로 수행된다. 그림 3은 2GB, 2KB SLC NAND Flash구조의 예이다. 플래시 메모리의 쓰기속도는 읽기에 비하여 매우 느리다. 이는 페이지 단위

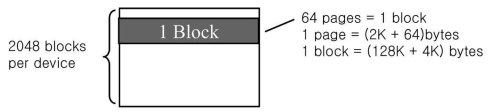


그림 3 2GB, 2KB Page SLC NAND 구조

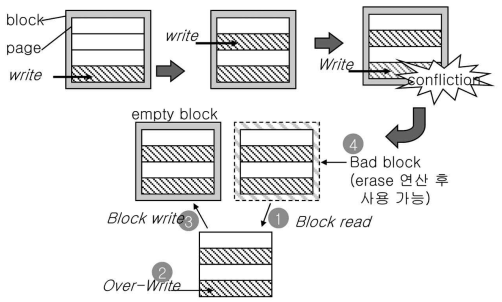


그림 4 플래시 메모리의 블록 병합 연산

의 쓰기 연산 시 한번 쓴 페이지에 대한 덮어 쓰기(페이지 충돌)가 불가능하며, 이때 블록 전체에 대하여 소거 연산을 거친 후에야 쓰기가 가능하기 때문이다. 블록당 소거 연산은 약 2ms[8]정도의 매우 긴 시간을 요구한다. 따라서 플래시 메모리의 성능 향상을 위해서는 쓰기 연산으로 인한 소거 연산을 줄이는 것이 매우 중요하다. 예를 들어 [9]에서 읽기, 쓰기, 소거연산 비용의 각 비율은 1:7:63이다. 이러한 특성으로 인해 쓰기 연산 과정은 쓰기 연산이 수행 될 페이지가 포함된 블록을 읽은 후 새로 쓸 내용과 병합한 후 미리 소거되어 있는 빈 블록을 찾아서 쓰는 리다이렉트 과정이 필요하다. 그림 4는 병합 과정의 예시이다. 그리고 이러한 병합 과정 때문에 플래시 메모리를 사용할 경우 논리적 페이지와 물리적 페이지의 주소 매핑 기능이 제공되어야 한다. 또한 블록의 소거 횟수는 제한되어 있으며 이는 플래시 메모리의 수명에 영향을 주기 때문에 빈 블록을 찾을 시 이러한 특성도 고려되어야 한다. 플래시 변환 계층(FTL)이라고 불리는 소프트웨어는 이와 같은 주소 매핑 기능을 수행하는 소프트웨어 모듈이며, FTL은 주소 매핑 외에도 wear-leveling(블록의 소거 횟수를 균일하게 분포하여 bad 블록의 수를 최소화하는 알고리즘), garbage collection(즉시 소거하지 않고 invalid 블록으로 남긴 후 한꺼번에 소거연산을 처리하는 방법)등의 기능을 수행하기도 한다. 주소 매핑 방법은 페이지 단위 매핑, 블록 단위 매핑, 하이브리드 매핑 방법 등이 있으며, 특히 로그-기반(log-based) FTL은 몇 개의 로그 블록에 임시로 쓰기 연산을 함으로써 페이지 충돌로 인한 블록 소거 횟수를 줄이는 방법을 사용한다.

이와 같은 플래시 메모리의 특성으로 인하여 기존의 디스크 기반 인덱싱 방법들을 그대로 수행할 경우 빈번한 소거 연산 등 심각한 문제점들이 발생한다. [10-12]는 플래시 메모리에 적합한 인덱싱 방법을 연구하였다. 그러나 [10]는 플래시 메모리의 물리적 특성인 NOP (Number Of Partial page programming)이 1인 경우 적용이 불가능하며(최근에 주로 사용되는 MLC 플래시는 대부분 NOP가 1이다), [11,12]는 각각 B-Tree, R-Tree에 대하여 노드의 효과적인 페이지 할당을 통해 소거 연산을 줄이는 방법들을 제안하였지만 M-Tree처럼 이미 노드의 크기를 페이지 크기를 고려하여 고정시키는 인덱싱 방법에 대해서는 그 이점이 크지 않다고 볼 수 있다. 따라서 본 논문에서는 입출력 최적화에 목적을 둔 M-tree를 대상으로 플래시 메모리상에서 쓰기 횟수를 감소시키는 방향으로 연구를 진행하였다.

3. 노드 분할을 통한 쓰기 횟수 감소 방법

M-tree의 노드 쓰기 연산은 노드에 새 오브젝트가 삽입되거나 노드의 정보가 변경되는 경우, 또는 분할로 인한 새 노드의 생성시에 발생하며 이 때 대부분 노드의 일부분만 변경이 된다(partiality). 예를 들어 그림 5와 같이 4KB 노드에 한 개의 오브젝트가 추가될 경우 파일시스템의 클러스터 사이즈가 4KB이고 플래시의 블록사이즈가 2KB라고 가정한다면 단 한 개의 오브젝트가 추가되더라도 두 개의 블록에 대한 쓰기 연산이 발생하게 된다. 특히 이 때 두 블록은 덮어쓰기로 인한 블록 병합 및 소거 연산을 불러일으키기 때문에 입출력 비용을 크게 증가시키는 원인이 된다.

이러한 현상을 해결하기 위해서는 노드의 변경된 부분만을 다시 쓰는 방식이 필요하며, 이는 논리적 노드를 물리적으로 작은 단위로 나누어 저장함으로써 가능하다. 그러나 플래시 메모리의 특성 상 노드를 나누더라도 페이지 덮어쓰기가 불가능한 제한으로 인하여 블록 단위의 병합 연산을 발생시키므로 그 효과가 없어진다고 볼 수 있다. 예를 들어 그림 5의 예에서 노드를 작은 단위로 나누어 입출력 한다고 가정하면 o5의 삽입 시 두 번

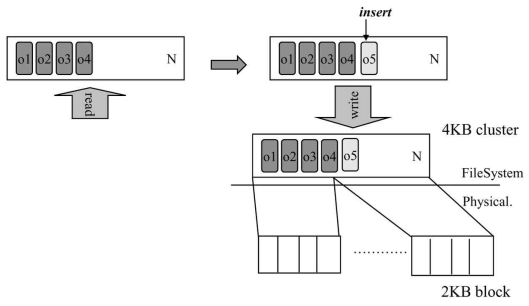


그림 5 Node Partiality 현상

째 블록의 첫 번째 섹터만 출력하는 것이 가능할 것이다. 하지만 뒤이어 o5가 다시 갱신될 경우 이미 o5가 기록되어 있는 두 번째 블록의 첫 번째 섹터는 다시 쓰여지지 못하므로 결국 두 번째 블록 전체에 병합 연산이 가해지고 노드 분할의 효과를 상실하게 된다. 따라서 노드를 분할하여 작은 노드 조각단위로 저장하는 방법뿐만 아니라 작은 노드 조각들을 최대한 같은 물리적 섹터에 모아서 저장하는 방법이 필요하다. 이러한 경우 서로 다른 노드들의 노드 조각들을 동일한 물리적 섹터에 모아서 저장하기 위한 섹터 쓰기 버퍼와 논리적 노드와 물리적 노드 조각들이 서로 물리적 위치가 달라지게 되므로 매핑 테이블 및 노드 조각들에 대한 물리적 섹터의 할당 기능이 필요하게 된다. 그림 6은 이와 같은 노드 분할 방법의 예시로서, 4KB의 논리적 노드를 4조각으로 분할하고 플래시의 물리적 섹터의 크기가 2KB, 2섹터 크기의 버퍼를 사용할 경우이다. 그림 6의 의 예에서 현재 노드 N_1의 크기는 2KB이며, 만일 N_1의 마지막 엔트리의 정보가 변경될 경우 f_1^1의 쓰기 연산이 버퍼 B^1에 발생하며 이미 P^3에 쓰여진 값은 BAD가 되고 $Table^{N_1}$은 $Table^{N_1} : B_0^1\, B_1^1\, \varnothing\, \varnothing$이 된다. 또한 버퍼가 찼으므로 이 후 버퍼가 플러시될 경우 새 섹터(P_5 or P_1)에 쓰여지며 $Table^{N_1}$의 값도 갱신된다. 그림 7은 이러한 노드 분할 방식의 효과를 예시한 것으로써 그림 7(a)의 경우 두 번의 섹터 쓰기와 한 번의 블록 병합이 발생하였지만 그림 7(b)의 노드 분할 방식의 경우 단 한번의 섹터 쓰기만 발생하는 것을 나타낸다.

앞에서 설명한 바와 같이 이러한 노드 분할 방법을 사용할 경우 매핑 테이블과 섹터 할당이라는 오버헤드가 발생한다. 먼저 매핑 테이블의 공간 비용은 만일 전체 사진 데이터가 10000개, M값(M-Tree Node Size)이 30일 경우 전체 노드 개수는 대략 500개 정도 생성된다고 본다면 분할 수가 4일 때 500×4×4byte=8KB 정도이다. 그리고 각 물리적 노드 조각들에 대한 상태를 {FREE,

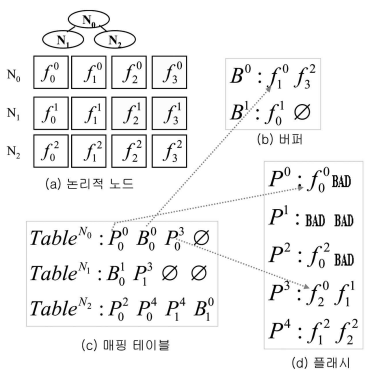
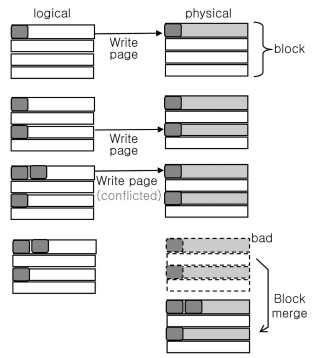
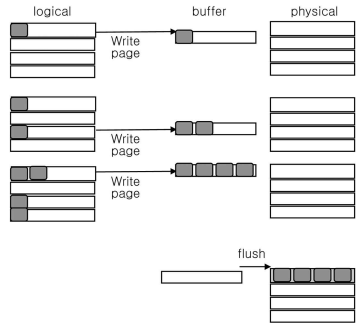


그림 6 노드 분할 방법의 구조



(a) 노드 분할 적용 전 노드 입출력
(3번의 페이지 쓰기와 1번의 블록 병합 발생)



(b) 노드 분할 적용 후 노드 입출력
(1번의 페이지 쓰기 발생)

그림 7 노드 분할 효과의 예

BAD, ALLOCATED}로 표시하는 비트 벡터를 사용하여 버퍼 플러시때 새로 쓸 섹터를 간단히 결정할 수 있다. 그러나 노드 분할 시 발생하는 가장 큰 문제점은 섹터를 읽을 때 분할 수 이하 만큼의 인다이렉션이 필요하다는 점이다. 예를 들어 그림 6의 경우에서 노드 N2를 읽으려면 총 3개의 섹터를 읽어야 한다. 그러나 플래시 메모리의 특성 상 읽기의 성능 저하보다 쓰기의 성능 향상 폭이 훨씬 더 크기 때문에 이와 같은 노드 분할 방법은 전체 입출력 성능을 향상시킬 수 있다고 볼 수 있다. 그리고 노드 읽기 인다이렉션 양을 줄이기 위해서 논리적 노드가 물리적 노드 조각들로 흩어진 정도를 최소화하도록 지연된 쓰기를 통해 쓰기 스케줄링을 하는 방법도 가능하다. 노드 조각의 흩어짐을 최소화하는 방법은 매우 일반적인 파일 입출력 관련 연구 분야이기 때문에 본 논문의 범위에서는 직접 다루지 않는다.

4. 실험 및 분석

그림 8은 M=4KB, Flash Sector Size=2KB, 분할수=4일 경우 각 연산 횟수를 측정한 것이고 그림 9는 M=8KB, Flash Sector Size=4KB, 분할수=8일 경우 측

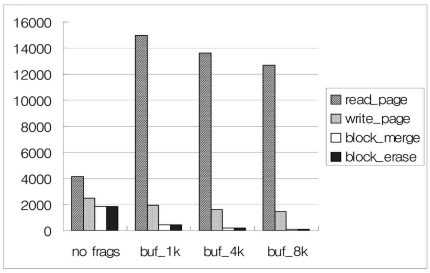


그림 8 노드 분할 성능 측정
(M=4KB, Flash Sector=2KB, 분할수=4)

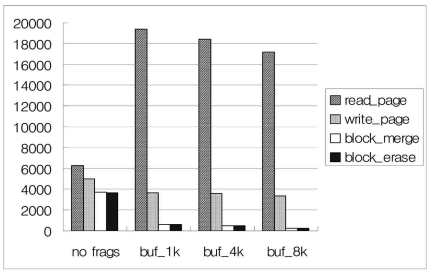


그림 9 노드 분할 성능 측정
(M=8KB, Flash Sector=4KB, 분할수=8)

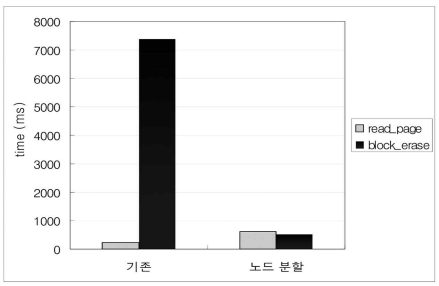


그림 10 노드 분할 방식 적용 시 출력 시간의 감소

정한 것으로써, 블록 병합 및 소거 연산 횟수가 현저하게 감소한 것을 나타낸다. 읽기 연산의 횟수는, 3장에서 설명한 바와 같이 노드 조각의 흩어짐으로 인한 인다이렉션의 증가로 인하여, 분할 수가 4일 경우 약 2.7배, 분할 수가 8일경 우 약 6배로 증가하였지만 그림 10과 같이 블록 소거의 감소로 인한 쓰기 비용에 대한 이득이 훨씬 크기 때문에 플래시 메모리에서 인덱싱을 구축할 때 매우 효과적인 방법이라고 볼 수 있다.

5. 결 론

본 논문에서는 플래시 메모리 상에서 M-tree를 구축 시 플래시 메모리의 높은 쓰기 비용으로 인한 노드 출력 비용을 줄이기 위하여, 노드 분할 방식을 적용하여 쓰기 횟수를 기존의 약 7% 정도로 현저히 감소시키는 효과를 확인하였다. 본 논문에서 제안한 방법을 사용하여 플래시 메모리 상에서 대량의 데이터에 대한 인덱싱을 효율적으로 구축할 수 있을 것이다.

참 고 문 헌

[1] Yianilos, P. N., "Excluded middle vantage point forests for nearest neighbor search," *Tech. rep., NEC Research Institute, 1999. Presented at the Sixth DIMACS Implementation Challenge: Near Neighbor Searches workshop*, Jan. 1999.

[2] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of Fourth Annu. ACM-SIAM Symp. Discrete Algorithms*, pp.311-321, 1993.

[3] T. Bozkaya and Z. M. Ozsoyoglu, "Distance-based indexing for high-dimensional metric spaces," in *Proceedings of ACM-SIGMOD international conference on Management of data*, pp.357-368, 1997.

[4] P. Ciaccia, M. Patella, F. Rabitti, and P. Zezula, "Indexing metric spaces with M-tree," in *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 1997)*, pp. 426-435, 1997.

[5] C. Traina, Jr., A. J. M. Traina, B. Seeger, and C. Faloutsos, "Slim-trees: High performance metric trees minimizing overlap between nodes," in *proceedings 7th EDBT 2000*, pp.51-65, 2000.

[6] X. Zhou, G. Wang, J.Y. Xu, G. Yu, "M+-tree: A new dynamical multidimensional index for metric spaces," in *Proceedings of the 14th Australasian Database Conference (ADC'03)*, pp.161-168, 2003.

[7] A. B. Bityutskiy. JFFS3 design issues. http://www.linux-mtd.infradead.org.

[8] K. Han-Joon and L. Sang-goo, "A new flash memory management for flash storage system," in *Proceedings of the Computer Software and Applications Conference (COMPSAC 1999)*, pp.284-289, 1999.

[9] Samsung Electronics. *Nand flash memory & smartmedia data book*, 2004.

[10] Siwoo Byun, "F-Tree : Flash Memory based Indexing Scheme for Portable Information Devices," in *Proceedings Journal of Information Technology Applications & Management*, vol.13, no.4, pp. 257-271, 2006.

[11] D. Kang, D. Jung, J.-U. Kang, and J.-S. Kim, "μ-tree: an ordered index structure for nand flash memory," in *Proceedings of the 7th ACM & IEEE international conference on Embedded software (EMSOFT '07)*, pp.144-153, 2007

[12] Chin-Hsien Wu, Li-Pin Chang, Tei-Wei Kuo, "An efficient r-tree implementation over flash-memory storage systems," in *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pp.17-24, 2003.