플래시 메모리 기반의 효율적인 공간 인덱스 기법

(An Efficient Spatial Index Technique based on Flash-Memory)

김 정 준[*] 심 희 정[**] 강 홍 구[***] 이 기 영[****] 한 기 준[*****]
(Joung-Joon Kim) (Hee-Joung Sim) (Hong-Koo Kang) (Ki-Young Lee) (Ki-Joon Han)

요 약 최근 무선 인터넷이 발전하고 모바일 단말기 사용이 증가함에 따라 위치 기반 서비스(LBS: Location Based Service)에 대한 요구가 증가되고 있으며, 모바일 단말기 환경에서 효율적인 위치 기반 서비스를 제공하기 위해 공간 데이타를 저장 및 관리하는 공간 인덱스의 연구가 필수적으로 요구되고 있다. 플래시 메모리는 모바일 단말기에서 대용량의 공간 데이타를 효율적으로 저장하기 위한 보조 저장 장치로 많이 사용된다. 그러나 플래시 메모리에 기존 공간 인덱스를 그대로 적용할 경우 빈번한 노드 갱신에 의한 쓰기 연산 증가로 인덱스 성능이 저하된다. 이러한 문제점을 해결하고자 최근 플래시 메모리 기반 공간 인덱스가 연구되고 있지만 버퍼와 플래시 메모리의 공간 활용도가 낮아 효율성이 떨어지는 문제점이 있다. 따라서, 본 논문에서는 기존의 플래시 메모리 기반 공간 인덱스들의 문제점을 해결하기 위해 노드 압축 기법과 쓰기 연산 지연 기법을 적용한 FR-Tree(Flash-Memory based R-Tree)를 제안하였다. FR-Tree의 노드 압축 기법은 공간 데이타의 MBR(Minimum Bounding Rectangle)을 상대 좌표값과 MBR 크기 값을 이용해 압축함으로써 플래시 메모리의 공간 활용도를 높였다. 그리고 쓰기 연산 지연 기법은 공간 데이타의 삽입, 갱신, 삭제시 플래시 메모리에 저장된 공간 인덱스에 바로 반영하지 않고 버퍼에 임시적으로 저장한 후 일괄적으로 플래시 메모리에 반영하여 플래시 메모리의 쓰기 연산 횟수를 줄였다. 특히, 버퍼내 동일한 공간 데이타들의 중복 저장을 방지하여 버퍼의 공간 활용도를 높였다. 마지막으로, 본 논문에서는 다양한 성능 평가를 통해 FR-Tree가 플래시 메모리에서 기존 공간 인덱스들에 비해 성능이 우수함을 입증하였다.

키워드 : 공간 인덱스, 플래시 메모리, MBR 압축, 쓰기 연산 지연

Abstract Recently, with the advance of wireless internet and the frequent use of mobile devices, demand for LBS(Location Based Service) is increasing, and research is required on spatial indexes for the storage and maintenance of spatial data to provide efficient LBS in mobile device environments. In addition, the use of flash memory as an auxiliary storage device is increasing in order to store large spatial data in a mobile terminal with small storage space. However, the application of existing spatial indexes to flash-memory lowers index performance due to the frequent updates of nodes. To solve this problem, research is being conducted on flash-memory based spatial indexes, but the efficiency of such spatial indexes is lowered by low utilization of buffer and flash-memory space. Accordingly, in order to solve problems in existing flash-memory based spatial indexes, this paper proposed FR-Tree (Flash-Memory based R-Tree) that uses the node compression technique and the delayed write operation technique. The node compression technique of FR-Tree increased the utilization of flash-memory space by compressing MBR(Minimum Bounding Rectangle) of spatial data using relative coordinates and MBR size. And, the delayed write operation technique reduced the number of write operations in flash memory by storing spatial data in the buffer temporarily and reflecting them in flash memory at once instead of reflecting the insert, update and delete of spatial data in flash-memory for each operation. Especially, the utilization of buffer space was enhanced by preventing the redundant storage of the same spatial data in the buffer. Finally, we performed various performance evaluations and proved the superiority of FR-Tree to the existing spatial indexes.

Keywords : Spatial Index, Flash-Memory, MBR Compression, Delayed Write Operation

† 본 연구는 국토해양부 첨단도시기술개발사업 - 지능형국토정보기술혁신 사업과제의 연구비지원(07국토정보C05)에 의해 수행되었습니다.

[*] 건국대학교 컴퓨터공학과 박사과정, jjkim9@db.konkuk.ac.kr
[**] KT DataSystems 인프라운영본부 사원, hjsim@db.konkuk.ac.kr
[***] 건국대학교 컴퓨터공학과 강의교수, hkkang@db.konkuk.ac.kr
[****] 을지대학교 의료산업부 교수, kylee@eulji.ac.kr(교신저자)
[*****] 건국대학교 컴퓨터공학과 교수, kjhan@db.konkuk.ac.kr

논문접수 : 2009.02.13 수정일 : 1차 2009.03.16 심사완료 : 2009.03.20

1. 서 론

최근 무선 인터넷의 발전과 PDA(Personal Digital Assistant), HPC(Handheld PC), 스마트 폰(Smart Phone)의 사용 증가로 GIS(Geographic Information System)는 모바일 단말기에서 효율적인 위치 기반 서비스(LBS: Location Based Service)를 제공하기 위한 모바일 GIS로 변화하고 있다. 또한, 대용량의 공간 데이타를 모바일 단말기 상에서 빠르게 검색하기 위한 공간 인덱스에 대한 연구 필요성이 대두되고 있다[1,2,3].

플래시 메모리는 모바일 단말기에서 대용량의 데이타를 저장하기 위한 보조 저장 장치로 많이 사용된다. 하지만 플래시 메모리는 읽기 연산 처리 속도에 비해 쓰기 및 삭제 연산 처리 속도가 상대적으로 매우 느리고, 또한, 플래시 메모리는 제자리 덮어 쓰기(Overwrite)가 불가능하고 페이지 당 삭제 횟수가 제한되는 특징을 가지고 있다[4].

따라서, 플래시 메모리에 기존 공간 인덱스를 그대로 적용하면 빈번한 노드 갱신으로 플래시 메모리의 쓰기 및 삭제 연산 횟수가 증가하고 이에 따른 연산 시간의 증가로 인덱스 성능은 저하된다. 이러한 문제를 해결하기 위해 플래시 메모리의 제약적 특징을 고려한 공간 인덱스 기법이 연구되고 있다. 대표적인 RFTL(R-Tree Flash Translation Layer)은 삽입, 갱신, 삭제시 공간 데이타들을 버퍼에 저장한 후 일괄적으로 플래시 메모리에 저장함으로써 플래시 메모리의 쓰기 및 삭제 연산의 횟수를 줄여 공간 인덱스의 성능을 높였다[5,6]. 그러나 공간 데이타의 중복 저장으로 버퍼와 플래시 메모리의 공간 활용도가 낮아 저장 효율성이 저하되는 문제점이 있다.

본 논문에서는 이러한 문제점을 해결하기 위해 버퍼와 플래시 메모리의 공간 활용도를 증가시켜 공간 인덱스의 성능을 높인 FR-Tree(Flash-Memory based R-Tree)를 제안한다. FR-Tree에서는 R-Tree에 노드 압축 기법과 쓰기 연산 지연 기법을 적용하였다. 노드 압축 기법은 MBR 크기를 줄여 플래시 메모리의 공간 활용도를 높이기 위해 공간 데이타 MBR(Minimum Bounding Rectangle)을 RSMBR (Relative-Sized MBR)로 변환하여 저장하는 압축 기법이다. RSMBR은 MBR 좌하점과 우상점을 상대 좌표값과 MBR 크기로 표현한다[7].

쓰기 연산 지연 기법은 노드 갱신에 따른 플래시 메모리의 쓰기 및 삭제 연산 횟수를 줄여 플래시 메모리의 공간 활용도를 높이기 위해 삽입, 갱신, 삭제시 공간 데이타를 버퍼에 저장한 후 일괄적으로 플래시 메모리에 저장함으로써 쓰기 및 삭제 연산을 지연시키는 기법이다. 삽입될 공간 데이타는 리스트나 R-Tree 구조로 버퍼에 저장되고, 갱신 및 삭제될 공간 데이타는 플래시 메모리에 저장된 R-Tree의 노드를 참조하는 포인터와 같이 버퍼에 저장된다. 버퍼가 가득 차게 되면 버퍼에 저장된 공간 데이타는 플래시 메모리에 저장된 R-Tree에 반영된다. 특히, 갱신 및 삭제 버퍼의 공간 데이타는 포인터를 이용해 노드에 직접 접근하므로 갱신 및 삭제 연산을 빠르게 처리할 수 있다. 그리고, 버퍼내 존재하는 공간 데이타에 대해 갱신 및 삭제 연산이 발생하면 버퍼내 기존 공간 데이타를 갱신 및 삭제함으로써 공간 데이타가 중복 저장되는 것을 방지한다.

본 논문의 구성은 다음과 같다. 제2장에서는 관련 연구로 플래시 메모리, R-Tree, RFTL에 대해 분석한다. 제3장에서는 본 논문에서 제시한 FR-Tree의 정의, 인덱스 구조, 연산 과정, 알고리즘에 대해 설명한다. 제4장에서는 기존의 공간 인덱스와 FR-Tree의 성능을 비교 분석한다. 마지막으로, 제5장에서는 본 논문의 결론에 대해 언급한다.

2. 관련 연구

본 장에서는 플래시 메모리, R-Tree, RFTL에 대해 분석한다.

2.1 플래시 메모리

플래시 메모리는 비휘발성 메모리로 디스크에 비해 속도가 빠르고 충격에 강하며 휴대성이 높아 모바일 단말기의 보조 저장 장치로 사용된다. 플래시 메모리는 다음과 같은 제약적인 특징을 가지고 있다. 첫째, 플래시 메모리는 읽기 속도에 비해 상대적으로 쓰기 및 삭제 속도가 느리다. 표 1은 플래시 메모리에서 각 연산의 처리 속도를 보여준다[8,9].

표 1. 플래시 메모리의 연산 속도

(삼성 K9K4G08U0Mm 8Gbit 기준)

연산 구분	읽기	쓰기	삭제
연산 단위	페이지 (2KB)	페이지 (2KB)	블록 (128KB)
연산 시간	25μs	200μs	2ms

표 1에서 보듯이 페이지 단위로 수행하는 읽기와 쓰기 연산 중 쓰기 연산은 약 8배의 더 많은 시간을 소요한다. 또한 블록 단위로 수행되는 삭제 연산은 읽기 연산에 비해 약 80배의 시간을 소요해 연산 속도가 가장 느리다.

둘째, 플래시 메모리는 블록 당 삭제 횟수가 최소 100,000에서 최대 1,000,000번까지로 제한된다. 만약, 제한된 삭제 횟수를 초과하면 해당 블록은 더 이상 데이타를 저장할 수 없게 된다. 셋째, 플래시 메모리는 디스크와 다르게 제자리 덮어쓰기(Overwrite)를 지원하지 않는다. 그 대신 새로운 페이지에 데이타를 저장하여 갱신하는 외부 덮어 쓰기(Out-Place Update)를 지원한다.

2.2 R-Tree

대표적인 디스크 기반 인덱스인 R-Tree는 다차원의 공간 데이타를 검색하기 위해 B-Tree를 변형시킨 공간 인덱스이다[10]. R-Tree는 빈번한 노드 갱신으로 인해 동일한 섹터(Sector)를 자주 갱신하게 된다. 따라서 R-Tree를 플래시 메모리에 반영할 경우 노드 갱신에 의해 외부 덮

어 쓰기가 증가하고 결국 플래시 메모리의 쓰기 연산 횟수가 증가되기 때문에 인덱스 성능이 저하된다. 그림 1은 R-Tree에서 공간 데이타 삽입 예를 보여준다.

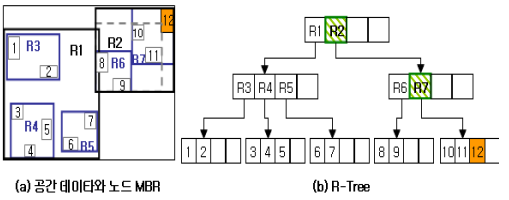

그림 1. R-Tree에서 공간 데이타의 삽입

그림 1(a)에서 보듯이 1번에서 12번까지의 실선으로 표시된 사각형은 공간 데이타 MBR이고, R1에서 R7까지의 사각형은 R-Tree의 노드 MBR을 나타낸다. 그림 1(b)는 그림 1(a)의 MBR을 R-Tree 구조로 나타낸 것이다. 이 중 12번 공간 데이타가 삽입되면, 12번 공간 데이타가 속한 단말 노드와 R2와 R7이 속한 비단말 노드까지 연쇄적으로 갱신이 일어난다. 이렇듯 R-Tree에서는 한 번의 연산으로 다수의 노드가 갱신될 수도 있으며, 빈번한 노드 갱신은 플래시 메모리의 쓰기 연산 횟수를 증가시키기 때문에 인덱스 성능을 저하시킨다. 따라서 플래시 메모리의 제약적 특징을 고려한 공간 인덱스 기법에 대한 연구가 필요하다.

2.3 RFTL(R-Tree Flash Translation Layer)

RFTL은 R-Tree에 대해서 버퍼(Buffer)를 사용해 플래시 메모리의 쓰기 연산 횟수를 감소시킨 공간 인덱스이다[5,6]. 그림 2는 RFTL의 구조를 보여준다.

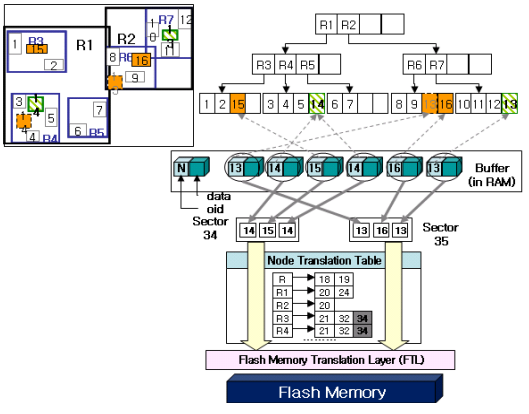

그림 2. RFTL의 구조

그림 2에서 보듯이 RFTL은 공간 데이타를 임시로 저장하는 버퍼와 노드를 구성하기 위한 노드 변환 테이블(Node Translation Table)로 구성된다. RFTL에서 삽입, 갱신, 삭제될 공간 데이타들은 플래시 메모리에 저장되기에 앞서 연산이 발생한 순서대로 버퍼에 저장된다. 이 때, 버퍼가 가득 차서 더 이상 공간 데이타를 저장할 수 없게 되면 버퍼내 공간 데이타들은 플래시 메모리의 페이지 크기 단위로 묶여 플래시 메모리에 저장된다. 그리고 공간 데이타가 저장된 플래시 메모리의 논리 주소를 노드 변환 테이블에 추가한 뒤 버퍼를 초기화하여 다음 연산을 위한 버퍼 공간을 확보한다.

이렇게 RFTL은 삽입, 갱신, 삭제될 노드의 엔트리들을 버퍼에 임시로 저장한 후 일괄적으로 플래시 메모리의 페이지 크기에 맞게 저장하여 쓰기 연산의 효율성을 높였다. 그러나 RFTL은 노드 변환 테이블 운영에 따른 메모리 공간의 사용으로 인덱스 운영비용이 증가한다. 그리고 버퍼와 플래시 메모리에 공간 데이타들이 중복 저장되므로 버퍼와 플래시 메모리의 공간 활용도가 낮아져 인덱스의 저장 효율성이 저하되는 문제점을 가지고 있다.

본 논문에서는 이러한 문제를 해결하기 위해 버퍼와 플래시 메모리의 공간 활용도를 증가시킨 플래시 메모리 기반 공간 인덱스를 제안한다.

3. FR-Tree(Flash-Memory based R-Tree)

본 장에서는 본 논문에서 제시한 플래시 메모리 기반 공간 인덱스 기법인 FR-Tree에 대해 설명한다. 우선 FR-Tree의 정의, 구조, 연산에 대해 설명한 뒤 마지막으로 알고리즘에 대해 기술한다.

3.1 FR-Tree의 정의

FR-Tree는 플래시 메모리의 제약적 특징을 고려해 R-Tree를 확장한 플래시 메모리 기반 공간 인덱스이다. FR-Tree는 버퍼와 플래시 메모리의 공간 활용도를 높이기 위한 노드 압축 기법과 쓰기 연산 지연 기법을 사용한다.

3.1.1. 노드 압축 기법

FR-Tree에서는 플래시 메모리의 공간 활용도를 높이기 위해 공간 데이타의 MBR 좌표값 크기를 줄이는 RSMBR(Relative-Sized MBR) 압축 기법을 적용한다. RSMBR 압축 기법은 MBR 좌표값 크기를 줄이기 위해 MBR 좌하점은 상대 좌표값으로 표현하고 우상점은 MBR 크기로 표현한다[7]. 그림 3은 RSMBR 압축 기법을 적용한 예를 보여준다.

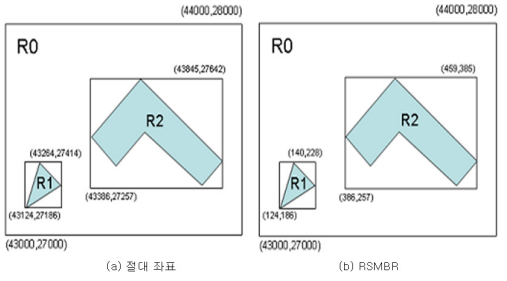

그림 3. MBR 압축 기법

그림 3(a)는 공간 데이타 MBR을 절대 좌표 MBR로 표현한 것이고, 그림 3(b)는 공간 데이타 MBR을 RSMBR로 표현한 것이다. 그림 3(b)에서 객체 R1의 MBR 좌하점 상대 좌표값은 (124, 186)이고 MBR의 X축 크기는 140이며, Y축 크기는 228이다. 그리고 객체 R2의 MBR 좌하점 상대 좌표값은 (386, 257)이고 MBR의 X축 크기는 459이며, Y축 크기는 385이다. 따라서, RSMBR 압축 기법에 의해 객체 R1은 (124, 186), (140, 228)으로 표현되고, 객체 R2는 (386, 257), (459, 385)로 표현된다. 그림 4는 RSMBR의 구조 및 실제 좌표값이 저장된 예를 보여준다.

2byte	3bit	4-16bit
좌하점의 상대 좌표	우상점의 길이 플래그	우상점의 실제 값
124	010	10001100

그림 4. RSMBR의 구조 및 예

그림 4에서 보듯이 RSMBR의 구조는 좌하점의 상대 좌표, 우상점의 길이 플래그, 우상점의 실제 값을 가진다. 우상점의 길이 플래그는 실제 값이 차지하는 bit의 크기를 나타낸다. 즉, 길이 플래그 값 1, 2, 3, 4는 각각 실제 값이 4bit, 8bit, 12bit, 16bit의 크기를 가지고 있다는 것을 의미한다. 그러므로 각 bit 단위 마다 값은 15, 255, 4095, 65535의 크기까지 표현할 수 있다. 예를 들어, 객체 R1의 RSMBR 우상점 좌표값 중 140은 길이 플래그 010과 실제 값 10001100로 표현된다. RSMBR에서 각 (X, Y) 좌표값은 최소 6byte, 최대 10byte로 표현 가능하기 때문에 MBR의 저장 용량이 감소한다. 대부분의 공간 데이타 MBR 크기는 255 이하이기 때문에 1byte로 표현 가능하며, 한 개의 RSMBR이 6byte를 넘는 경우는 그 빈도수가 절대적으로 낮다.

본 논문에서는 이러한 노드 압축 기법을 FR-Tree에 적용함으로써 MBR의 정확도를 유지하면서 공간 인덱스의 저장 용량에서 가장 큰 비중을 차지하는 MBR 크기를 줄여 플래시 메모리의 공간 활용도를 높였다.

3.1.2 쓰기 연산 지연 기법

FR-Tree에서는 버퍼와 플래시 메모리의 공간 활용도를 높이기 위해 버퍼를 이용하는 쓰기 연산 지연 기법을 적용한다. 이를 위해 FR-Tree는 플래시 메모리에 저장된 R-Tree인 대상 트리 외에 추가적으로 삽입, 갱신, 삭제 버퍼로 구성된다. 삽입, 갱신, 삭제될 공간 데이타는 연산에 따라 버퍼에 저장된 후 일괄적으로 대상 트리에 반영되기 때문에 플래시 메모리에 대한 쓰기 연산을 지연시킬 수 있다.

FR-Tree에서는 효율적인 공간 데이타의 일괄 삽입을 위해 공간 데이타를 삽입 버퍼에 저장하기 전에 대상 트리에서 공간 데이타 MBR을 포함하는 노드를 선택한다. 이때 선택된 노드를 삽입될 공간 데이타를 구분하기 위한 분류 노드라고 한다. 그림 5는 삽입될 공간 데이타의 분류 노드를 선택하는 과정을 보여준다.

1) 삽입될 공간 데이타 MBR이 대상 트리의 루트 노드 MBR에 포함되는지 확인한다.
2) 1)에서 포함되지 않는 경우 분류 노드 선택 과정을 종료한다.
3) 1)에서 포함되는 경우 삽입될 공간 데이타 MBR이 자식 노드 MBR에 포함되는지 반복적으로 확인한다.
4) 3)에서 삽입될 공간 데이타 MBR을 포함하는 자식 노드가 더 이상 없을 경우 마지막으로 비교하였던 자식 노드를 분류 노드로 선택하고 3)에서 단말 노드에 도달했을 경우 부모 노드를 분류 노드로 선택한다.

그림 5. 분류 노드 선택 과정

그림 6은 분류 노드 선택 과정의 예를 보여준다.

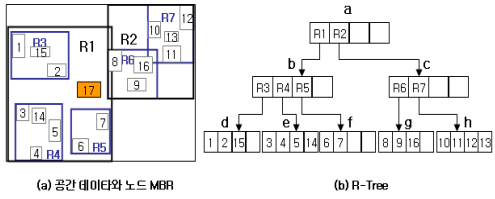

그림 6. 분류 노드 선택 예

그림 6(a)에서 보듯이 17번 공간 데이타가 삽입될 경우 17번 공간 데이타의 MBR이 루트 노드 a의 R1에 포함되므로 자식 노드 b를 검색하게 된다. 노드 b에서는 17번 공간 데이타 MBR을 포함하는 엔트리 MBR이 더 이상 없으므로 노드 b를 분류 노드로 선택한다.

삽입 버퍼의 공간 데이타들은 대상 트리에 삽입시 노드 갱신 비용을 줄이기 위해 분류 노드가 동일한 공간 데이타끼리 R-Tree로 구성되어 저장되고, 대상 트리의 분류 노드에 자식 노드로 삽입된다. 이 때 삽입 버퍼내 존재하는 R-Tree를 입력 트리라고 한다.

FR-Tree에서는 갱신 및 삭제될 공간 데이타를 갱신 버퍼와 삭제 버퍼에 저장한 후 일괄적으로 플래시 메모리에 반영함으로써 플래시 메모리의 쓰기 연산을 지연시킨다. 갱신 및 삭제 버퍼에는 갱신 및 삭제될 공간 데이타와 대상 트리에서 해당 공간 데이타가 속하는 노드를 참조하는 포인터가 저장된다. 그리고 버퍼에 동일한 공간 데이타가 존재한다면 버퍼에서 선행 처리를 하여 공간 데이타의 중복 저장을 방지한다. 그림 7은 갱신 버퍼와 삭제 버퍼의 구조를 보여준다.

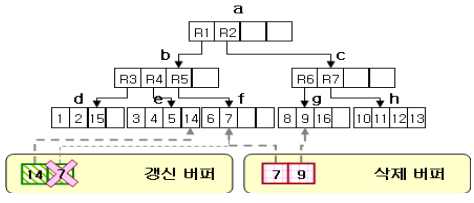

그림 7. 갱신 버퍼 및 삭제 버퍼 구조

그림 7에서 보듯이 7, 14번 공간 데이타가 갱신되면 각 공간 데이타가 속한 대상 트리의 노드 e와 f를 참조하는 포인터와 공간 데이타가 함께 갱신 버퍼에 저장된다. 그리고 7, 9번 공간 데이타가 삭제되면 먼저 갱신 버퍼에 저장된 7번 공간 데이타를 삭제하고 7, 9번 공간 데이타가 속한 대상 트리의 노드 f와 g를 참조하는 포인터와 공간 데이타가 함께 삭제 버퍼에 저장된다.

3.2 인덱스 구조

FR-Tree는 플래시 메모리에서 공간 인덱스의 성능을 향상시키기 위한 목적으로 R-Tree를 확장한 공간 인덱스이다. 그림 8은 FR-Tree의 전체 구조를 보여준다.

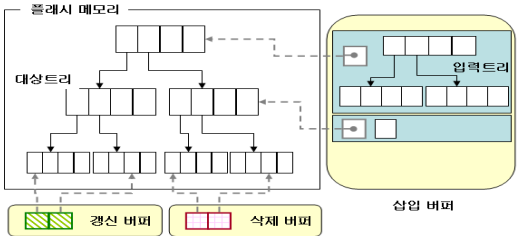

그림 8. FR-Tree의 구조

그림 8에서 보듯이 FR-Tree는 플래시 메모리에 저장된 대상 트리와 메모리상에 존재하는 삽입 버퍼, 갱신 버퍼, 삭제 버퍼로 구성된다. 그림 9는 FR-Tree에서 대상 트리와 입력 트리의 노드 및 엔트리 구조를 보여준다.

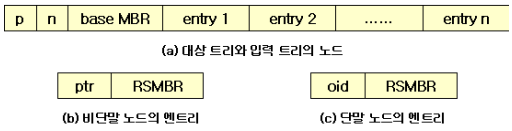

그림 9. FR-Tree 노드 및 엔트리 구조

그림 9(a)에서 보듯이 대상 트리와 입력 트리의 노드는 노드의 레벨 p, 노드의 엔트리 개수 n, 각 엔트리의 압축 기준이 되는 base MBR, 그리고 엔트리들로 구성된다. 그리고 엔트리는 비단말 노드와 단말 노드에 따라 두 가지의 구조를 가진다. 그림 9(b)는 비단말 노드의 엔트리 구조로 자식 노드를 가리키는 포인터 ptr과 엔트리의 RSMBR로 구성된다. 그림 9(c)는 단말 노드의 엔트리 구조로 실제 공간 데이타를 참조하기 위한 식별자 oid와 실제 공간 데이타의 RSMBR로 구성된다. 그림 10은 갱신 버퍼와 삭제 버퍼내 엔트리의 구조를 보여준다.

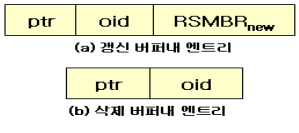

그림 10. 갱신 및 삭제 버퍼내 엔트리 구조

그림 10에서 보듯이 갱신 버퍼와 삭제 버퍼의 엔트리는 대상 트리의 단말 노드를 가리키는 포인터 ptr, 공간 데이타를 참조하기 위한 식별자 oid, 그리고 갱신될 공간 데이타의 새로운 RSMBRnew로 구성된다.

3.3 연산

본 절에서는 FR-Tree에서의 삽입, 갱신, 삭제 그리고 검색 연산에 대해 설명한다.

3.3.1 삽입 연산

FR-Tree에서 삽입될 공간 데이타들은 선택된 분류 노드에 따라 구분되어 삽입 버퍼에 R-Tree나 리스트 구조로 저장된 후, 삽입 버퍼가 가득 차게 되면 대상 트리에 삽입된다. 그림 11은 공간 데이타의 삽입 예를 보여준다.

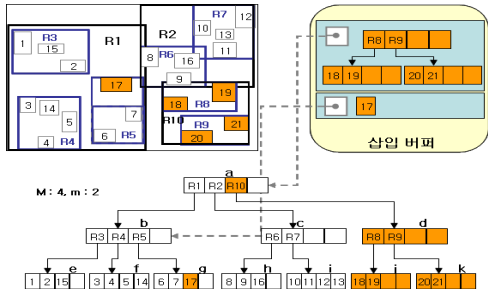

그림 11. 공간 데이타 삽입 예

그림 11에서 보듯이 17번 공간 데이타는 그림 6에서 보여준 분류 노드 선택 과정에 따라 b를 분류 노드로 선택하여 삽입 버퍼에 저장된다. 18번에서 21번까지의 공간 데이타는 a를 분류 노드로 선택하고 입력 트리 구조로 버퍼에 저장된다. 이 때 삽입 버퍼가 꽉 차게 되면 입력 트리는 대상 트리인 루트 노드 a의 자식 노드로 삽입되고, 17번 공간 데이타는 일반 R-Tree의 삽입 방법으로 대상 트리에 삽입된다.

3.3.2 갱신 연산

FR-Tree에서는 갱신 연산시 기존 공간 데이타가 저장된 위치에 따라 다른 갱신 연산을 수행한다. 표 2는 갱신 연산 방법을 보여준다.

표 2. 갱신 연산 방법

노드 / 저장 위치	동일한 경우 갱신 연산	상이한 경우 갱신 연산
삽입 버퍼	- 삽입 버퍼내의 기존 공간 데이타를 갱신한다.	
갱신 버퍼	- 갱신 버퍼내의 기존 공간 데이타를 갱신한다.	- 갱신 버퍼내 기존 공간 데이타를 삭제한다. - 삭제 버퍼에 기존 공간 데이타를 삽입한다. - 삽입 버퍼에 갱신될 공간 데이타를 삽입한다.
플래시 메모리 (대상 트리)	- 갱신 버퍼에 갱신될 공간 데이타를 삽입한다.	- 삭제 버퍼에 기존 공간 데이타를 삽입한다. - 삽입 버퍼에 갱신될 공간 데이타를 삽입한다.

표 2에서 보듯이 갱신 연산은 기존 공간 데이타의 MBR이 속한 노드와 갱신될 공간 데이타의 MBR이 속한 노드의 동일 여부와 기존 갱신 공간 데이타가 저장된 위치에 따라 다르게 수행된다. 그림 12는 공간 데이타의 갱신 예를 보여준다.

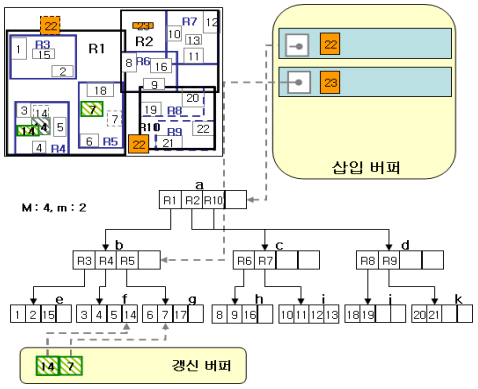

그림 12. 공간 데이타 갱신 예

그림 12에서 보듯이 삽입 버퍼에 저장된 22번 공간 데이타가 점선으로 표시된 사각형에서 실선으로 표시된 사각형으로 갱신될 경우 삽입 버퍼내 공간 데이타가 바로 갱신된다. 그리고 대상 트리에 저장된 7, 14번 공간 데이타가 갱신될 경우 기존 공간 데이타의 엔트리가 속한 노드 MBR에 계속해서 포함되므로 갱신 버퍼에 저장된다. 이 때, 갱신될 공간 데이타는 기존 공간 데이타가 속한 노드를 참조하는 포인터와 함께 저장된다. 또한, 14번 공간 데이타가 동일한 노드 안에서 재갱신되면 갱신 버퍼내 공간 데이타를 갱신하여 공간 데이타의 중복 저장을 방지한다. 갱신 버퍼가 꽉 찬 경우 갱신 버퍼내 공간 데이타들의 포인터를 이용하여 대상 트리는 신속하게 갱신된다.

3.3.3 삭제 연산

FR-Tree에서는 삭제 연산시 기존 공간 데이타가 저장된 위치에 따라 다른 삭제 연산을 수행한다. 표 3은 삭제 연산 방법을 보여준다.

표 3. 삭제 연산 방법

저장 위치	삭제 연산
삽입 버퍼	- 삽입 버퍼내 기존 공간 데이타를 삭제한다.
갱신 버퍼	- 갱신 버퍼내 기존 공간 데이타를 삭제한다. - 삭제 버퍼에 삭제될 공간 데이타를 삽입한다.
플래시메모리 (대상 트리)	- 삭제 버퍼에 삭제될 공간 데이타를 삽입한다.

표 3에서 보듯이 삭제될 공간 데이타가 저장된 위치에 따라 다르게 삭제 연산이 수행된다. 그림 13은 공간 데이타의 삭제 예를 보여준다.

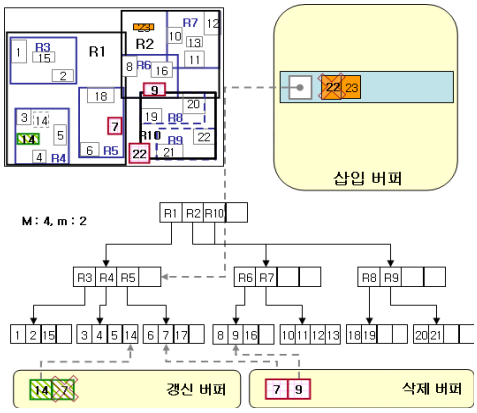

그림 13. 공간 데이타 삭제 예

그림 13에서 보듯이 삽입 버퍼에 저장된 22번 공간 데이타가 삭제될 경우 삽입 버퍼내 공간 데이타가 바로 삭제된다. 그리고 갱신 버퍼에 저장된 7번 공간 데이타가 삭제될 경우 갱신 버퍼에 존재하는 공간 데이타를 삭제한 후 삭제 버퍼에 삽입한다. 또한, 대상 트리에 저장된 9번 공간 데이타가 삭제될 경우 삭제 버퍼에 삽입한다.

3.3.4 검색 연산

FR-Tree에서는 검색시 대상 트리 외에 삽입, 갱신, 삭제 버퍼의 공간 데이타들도 함께 검색된다. 표 4는 FR-Tree에서의 검색 연산 방법을 보여준다.

표 4. 검색 연산 방법

저장 위치	검색 방법
플래시 메모리 (대상 트리)	① 질의 윈도우에 겹치는 결과 집합을 생성한다.
삭제 버퍼	② ①의 결과 집합에서 oid가 동일한 질의 결과는 삭제한다.
갱신 버퍼	③ ①의 결과 집합에서 oid가 동일한 질의 결과는 삭제한다. ④ 갱신 데이타가 질의 윈도우에 겹치면 ①의 결과 집합에 추가한다.
삽입 버퍼	⑤ 삽입 데이타가 질의 윈도우에 겹치면 ①의 결과 집합에 추가한다.

표 4에서 보듯이 먼저 플래시 메모리에 저장된 대상 트리를 일반 R-Tree 검색 방법에 따라 검색하여 질의 윈도우(QW: Query Window)에 겹치는 공간 데이타들로 구성된 결과 집합을 생성한다. 이 질의 결과 집합에 삭제, 갱신, 삽입 버퍼내 공간 데이타들을 표 4와 같이 순서대로 반영해 최종 질의 결과 집합을 완성한다. 그림 14는 공간 데이타 검색 예를 보여준다.

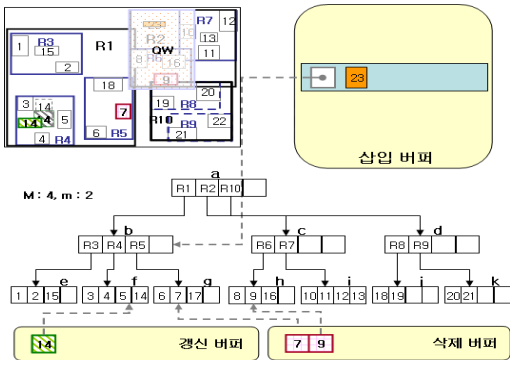

그림 14. 공간 데이타 검색 예

그림 14에서 보듯이 사각형 QW는 질의 윈도우를 나타내며, QW에 겹치는 공간 데이타를 검색하고자 한다. 우선 일반 R-Tree 검색 방법에 따라 대상 트리를 검색하여 8, 9, 10, 16번의 공간 데이타로 구성된 결과 집합을 생성한다. 이 중에서 삭제 버퍼에 존재하는 9번 공간 데이타는 질의 결과 집합에서 제외되고, 삽입 버퍼에 존재하는 23번 공간 데이타가 추가되어 최종 질의 결과 집합 8, 10, 16, 23을 생성한다.

3.4 알고리즘

본 절에서는 FR-Tree의 삽입, 갱신, 삭제, 검색 알고리즘에 대해 설명한다.

3.4.1 삽입 알고리즘

삽입 알고리즘의 입력은 공간 데이타의 *id*와 *mbr*이고, 출력은 삽입 성공 실패 여부를 반환한다. 그림 15는 FR-Tree의 삽입 알고리즘을 보여준다.

ALGORITHM : Insert(ID *id*, MBR *mbr*)

```
BEGIN
 NODE  target ← NULL
 IF(baseMBR of Root Node contains mbr)
  target ← findClassifiedNode(mbr,target_tree.rootnode)
 END IF
 IF(target == NULL)
  RETURN addExceptedList(id, mbr)
 END IF
 RSMBR rsmbr ← encordingRSMBR(target.mbr, mbr)
 IF(!addClssifiedList(target, id, rsmbr))
  RETURN FALSE
 END IF
 IF(isFullBuffer())
  writeBufferToFlash()
 END IF
END
```

그림 15. 삽입 알고리즘

그림 15에서 보듯이 삽입 알고리즘의 수행 과정은 다음과 같다. 먼저, 삽입될 공간 데이타 MBR이 대상 트리의 루트 노드 MBR에 포함되는 경우 삽입될 공간 데이타의 분류 노드를 검색한다. 그리고 분류 노드를 기준으로 MBR을 RSMBR로 압축하여 분류 노드에 따라 삽입 버퍼에 저장한다. 마지막으로 삽입 버퍼가 꽉 찬 경우 삽입 버퍼내 모든 공간 데이타를 대상 트리에 삽입한다.

3.4.2 갱신 알고리즘

갱신 알고리즘의 입력은 공간 데이타의 *id*와 *mbr*이고, 출력은 갱신 성공 실패 여부를 반환한다. 그림 16은 FR-Tree의 갱신 알고리즘을 보여준다.

ALGORITHM : update(ID *id*, MBR *mbr*)

```
BEGIN
 INTEGER save_place ← findSavePlace(id)

 IF(save_place is InsertBuffer)
  RETURN modifyInsertBuffer(id,mbr)
 END IF
 IF(save_place is UpdateDataBuffer)
  IF(mbr is contain MBR of node)
   RETURN modifyUpdateBuffer(id,mbr)
  ELSE IF(delUpdateBuffer(id, mbr))
   IF(addDeleteBuffer(id))
    RETURN addInsertBuffer(id,mbr)
   END IF
  END IF
 ELSE
  IF(mbr is contain MBR of node)
   RETURN addUpdateBuffer(id, mbr)
  ELSE IF(addDeleteBuffer(id))
   RETURN addInsertBuffer(id,mbr)
  END IF
 END IF
 RETURN FALSE
END
```

그림 16. 갱신 알고리즘

그림 16에서 보듯이 갱신 알고리즘의 수행 과정은 다음과 같다. 먼저, 기존 공간 데이타의 저장 위치가 삽입 버퍼이면 삽입 버퍼내 기존 공간 데이타를 갱신한다. 그리고 저장 위치가 갱신 버퍼이고 MBR이 기존 공간 데이타의 엔트리가 속한 노드 MBR에 계속해서 포함되면 갱신 버퍼내 기존 공간 데이타를 갱신한다. 만약, MBR이 기존 공간 데이타의 엔트리가 속한 노드 MBR을 벗어나면 갱신 버퍼내 기존 공간 데이타를 삭제하고, 삭제 버퍼에 삭제할 공간 데이타를 저장한 뒤 갱신될 공간 데이타를 삽입 버퍼에 삽입한다. 또한, 저장 위치가 대상 트리이고 MBR이 기존 공간 데이타의 엔트리가 속한 노드 MBR에 계속해서 포함되면 갱신 버퍼에 갱신될 공간 데이타를 저장한다. 만약, MBR이 기존 공간 데이타의 엔트리가 속한 노드 MBR을 벗어나면 삭제 버퍼에 삭제될 공간 데이타를 저장하고, 갱신될 공간 데이타를 삽입버퍼에 삽입한다.

3.4.3 삭제 알고리즘

삭제 알고리즘의 입력은 삭제될 공간 데이타의 *id*이고, 출력은 삭제 성공 실패 여부를 반환한다. 그림 17은 FR-Tree의 삭제 알고리즘을 보여준다.

```
ALGORITHM : delete(ID id)
BEGIN
 INTEGER save_place ← findSavePlace(id)
 IF(save_place is InsertBuffer)

  RETURN delInsertBuf(id)

 ELSE IF(save_place is UpdateDataBuffer)
  IF(delUpdateBuf(id))
   RETURN addDeleteBuf(id)
  END IF
 ELSE
   RETURN addDeleteBuf(id)
 END IF
 RETURN FALSE
END
```

그림 17.삭제 알고리즘

그림 17에서 보듯이 삭제 알고리즘의 수행 과정은 다음과 같다. 먼저, 저장 위치가 삽입 버퍼이면 삽입 버퍼내의 기존 공간 데이타를 삭제하고, 갱신 버퍼이면 갱신 버퍼내 기존 공간 데이타를 삭제하고 삭제 버퍼에 삭제될 공간 데이타를 삽입한다. 그리고 저장 위치가 대상 트리이면 삭제 버퍼에 삭제될 공간 데이타를 삽입한다.

3.4.4 검색 알고리즘

검색 알고리즘의 입력은 검색하고자 하는 질의 *window* 이고, 출력은 최종 질의 결과 집합을 반환한다. 그림 18은 FR-Tree의 검색 알고리즘이다.

```
ALGORITHM : search(MBR window)
BEGIN
 RESULT_SET resultSet ←
getResultFromTargetTree(window)
 resultSet ← getResultExceptDelete(resultSet, window)
 resultSet ← getResultAplliedUpdate(resultSet, window)
 resultSet ← resultSet +
getResultFromInsertBuffer(window)
 RETURN resultSet
END
```

그림 18. 검색 알고리즘

그림 18에서 보듯이 검색 알고리즘의 수행 과정은 다음과 같다. 먼저, 플래시 메모리에 저장된 대상 트리의 질의 결과 집합을 생성한다. 그리고 이 질의 결과 집합에서 삭제 버퍼에 존재하는 공간 데이타를 제외한 질의 결과 집합을 다시 생성하고, 갱신 버퍼에 존재하는 공간 데이타의 질의 결과를 반영한다. 마지막으로 삽입 버퍼내 질의 결과를 결과 집합에 추가하여 최종 질의 결과 집합을 생성한다.

4. 성능 평가

본 장에서는 본 논문에서 제시한 FR-Tree와 R-Tree와 RFTL의 성능을 비교 분석한다.

4.1 성능 평가 환경

본 논문에서는 FR-Tree의 성능 평가를 위해 Java 1.5.0으로 FR-Tree, RFTL, R-Tree를 구현하였다. 실험 환경은 Intel PXA270 624MHz, RAM 64MB, Microsoft Windows Mobile 5.0의 HP IPAQ hx2790 PDA이며, JVM은 Mysaifu JVM 0.3.9를 사용하였다. 그리고 공간 인덱스를 저장하기 위한 저장 장치로 플래시 메모리 Transcend SD Card 4G(읽기:0.03ms, 쓰기:0.4ms, 삭제:0.375ms)를 사용하였다. 또한 R-Tree의 노드 크기는 2KB로 설정하였고 FR-Tree와 RFTL의 버퍼 크기는 10KB로 동일하게 설정하였다.

본 논문에서 제시한 FR-Tree와 RFTL, R-Tree의 성능 비교를 위해 서울시 건물 데이타를 테스트 데이타로 사용하였다. 서울시 건물 데이타의 객체 수는 248,115개이고, 데이타 크기는 36MB이다.

4.2 성능 평가 결과

본 절에서는 FR-Tree, RFTL, R-Tree의 삽입, 갱신, 삭제, 검색 연산과 인덱스 크기에 대한 성능 평가 결과에 대해 분석한다.

4.2.1 삽입 연산 성능

삽입 연산의 실험을 위해 서울시 건물 데이타 중 20,000~100,000개의 객체를 임의로 선정하여 삽입 연산을 수행하였다. 그림 19는 삽입 연산의 성능 평가 결과를 보여준다.

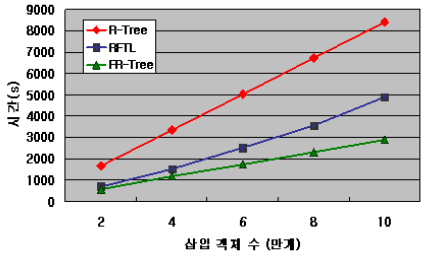

그림 19. 삽입 연산 성능 비교

그림 19에서 보듯이 FR-Tree는 R-Tree와 RFTL에 비해 높은 삽입 성능이 나타났으며, 삽입 객체 수가 증가할수록 더욱 좋은 성능을 보여주고 있다. FR-Tree와 RFTL은 버퍼를 사용한 삽입 연산으로 R-Tree에 비해 플래시 메모리의 쓰기 및 삭제 연산 횟수가 줄어 성능이 향상되었다. 특히, FR-Tree는 입력 트리 삽입에 따른 대상 트리의 노드 갱신 비용 감소와 RSMBR 압축 기법으로 인한 플래시 메모리의 공간 활용도 증가로 가장 좋은 삽입 평가 결과를 보여주었다.

4.2.2 갱신 연산 성능

갱신 연산의 실험을 위해 100,000개의 서울시 건물 데이타를 임의로 삽입한 뒤, 그 중 10~50%의 객체를 임의로 선정하여 갱신 연산을 수행하였다. 그림 20은 갱신 연산의

성능 평가 결과를 보여준다.

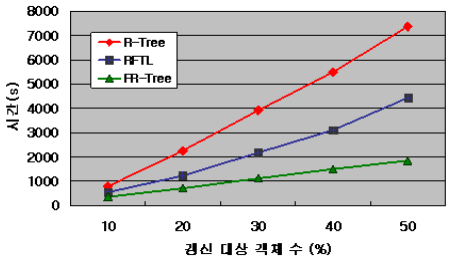

그림 20. 갱신 연산 성능 비교

그림 20에서 보듯이 FR-Tree는 R-Tree와 RFTL에 비해 갱신 성능이 높게 나타났으며, 갱신 객체 수가 증가할수록 더욱 좋은 성능을 보여주고 있다. FR-Tree와 RFTL는 버퍼를 사용한 갱신 연산으로 R-Tree에 비해 플래시 메모리의 쓰기 및 삭제 연산 횟수를 줄여 성능이 향상되었다. 특히, FR-Tree는 갱신 대상 공간 데이타가 속한 노드의 직접 접근으로 갱신 연산을 수행하므로 가장 좋은 갱신 성능 평가 결과를 보여주었다.

4.2.3 삭제 연산 성능

삭제 연산의 실험을 위해 100,000개의 서울시 건물 데이타를 임의로 삽입한 뒤, 그 중 10~50%의 객체를 임의로 선정하여 삭제 연산을 수행하였다. 그림 21은 삭제 연산의 성능 평가 결과를 보여준다.

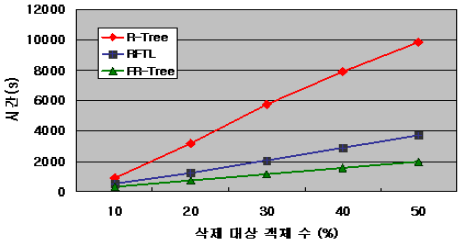

그림 21. 삭제 연산 성능 비교

그림 21에서 보듯이 FR-Tree는 R-Tree와 RFTL에 비해 삭제 성능이 높게 나타났으며, 삭제 대상 객체 수가 증가할수록 더욱 좋은 성능을 보여주고 있다. FR-Tree와 RFTL는 버퍼를 사용한 삭제 연산으로 R-Tree에 비해 플래시 메모리의 쓰기 및 삭제 연산 횟수를 줄여 성능이 향상되었다. 특히, FR-Tree는 삭제 대상 공간 데이타가 속한 노드의 직접 접근으로 삭제 연산을 수행하므로 가장 좋은 삭제 성능 평가 결과를 보여주었다.

4.2.4 검색 연산 성능

검색 연산의 실험을 위해 100,000개의 서울시 건물 데이타를 임의로 삽입한 뒤, 서울시 영역의 10~50%에 해당하는 크기의 윈도우 질의를 가지고 검색 연산을 수행하였다. 그림 22는 검색 연산을 수행한 결과를 보여준다.

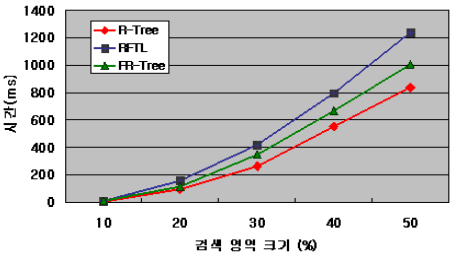

그림 22. 검색 연산 성능 비교

그림 22에서 보듯이 FR-Tree는 R-Tree에 비해 낮은 검색 성능을 보였으나, RFTL에 비해서는 높은 검색 성능을 보였다. 이는 FR-Tree가 R-Tree와는 다르게 삽입, 갱신, 삭제 버퍼에 대한 추가적인 검색 시간이 소요되었기 때문이다. 그리고 RFTL은 노드 변환 테이블을 이용하여 플래시 메모리에 흩어져 저장되어 있는 데이타 페이지들을 읽어 노드를 구성해야 하므로 가장 낮은 검색 성능 평가 결과를 보여주었다.

4.2.5 인덱스 크기 성능

인덱스 크기 실험을 위해 100,000개의 서울시 건물 데이타를 임의로 삽입하여 인덱스를 생성한 뒤 각 인덱스 크기(플래시 메모리에 저장된 인덱스)를 비교하였다. 그림 23은 인덱스 크기를 비교한 결과를 보여준다.

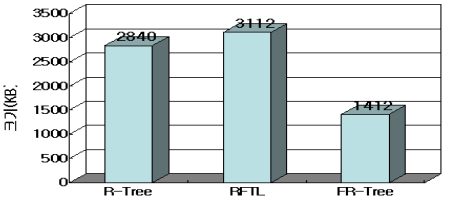

그림 23. 인덱스 크기 성능 비교

그림 23에서 보듯이 FR-Tree는 RFTL와 R-Tree에 비해 가장 적은 인덱스 크기를 사용함을 알 수 있다. 이는 FR-Tree가 RSMBR 압축 기법을 적용하여 인덱스 크기의 가장 큰 부분을 차지하는 MBR 크기를 줄였기 때문이다. 그에 반해 RFTL은 추가적 연산 플래그와 공간 데이타의 중복 저장으로 플래시 메모리의 공간 활용도가 가장 낮은 성능 평가 결과를 보여주었다.

5. 결 론

최근 무선 인터넷의 발전과 모바일 단말기 사용이 증가함에 따라 위치 기반 서비스를 효율적으로 제공하기 위한 모바일 GIS가 활발하게 연구 개발되고 있다. 특히, 그 중에서도 모바일 단말기에서 공간 데이타를 효율적으로 검색하기 위한 공간 인덱스의 연구가 필수적으로 요구되고 있다.

본 논문에서는 노드 압축 기법과 쓰기 연산 지연 기법

의 적용으로 버퍼와 플래시 메모리의 공간 활용도를 높인 FR-Tree를 제안하였다. FR-Tree는 노드 압축 기법을 적용함으로써 MBR의 크기를 줄여 플래시 메모리 공간 활용도를 향상시켰다. 또한, 쓰기 연산 지연 기법을 적용함으로써 플래시 메모리의 쓰기 및 삭제 연산 횟수를 감소시켜 버퍼와 플래시 메모리의 공간 활용도를 크게 향상시켰다. 그리고, 버퍼내 저장되어 있는 공간 데이타가 갱신이나 삭제된 경우 버퍼에 존재하는 기존 공간 데이타를 갱신 혹은 삭제함으로써 버퍼내 동일한 공간 데이타의 중복 저장을 방지하였다.

마지막으로, 본 논문에는 FR-Tree의 성능을 측정하기 위해 RFTL, R-Tree와의 성능을 비교 평가하였다. 성능 측정 결과 FR-Tree가 다른 인덱스에 비해 삽입 성능은 최대 190%, 갱신 성능은 최대 300%, 삭제 성능은 최대 330% 향상되었음을 확인할 수 있었다. 인덱스 크기도 비교 대상 인덱스에 비해 최대 55%까지 감소해 플래시 메모리의 공간 활용도가 가장 우수함을 확인하였다.

참 고 문 헌

[1] Forlizzi, L., Gueting, R.H., Nardelli, E., and Schneider, M., "A Data Model and Data Structures for Moving Objects Databases," Proc. of the ACM SIGMOD Conference, 2000, pp. 319-330.

[2] Niedzwiadek, H., "OpenLS-1 Interoperability Project: An Overview," LBS Forum Foundation Meeting and Celebration Seminar, 2002, pp. 30-51.

[3] 이기영, 윤재관, 한기준, "HBR-Tree를 이용한 실시간 모바일 GIS의 개발," 개방형지리정보시스템학회 논문지, 제6권 제1호, 2004, pp. 75-85.

[4] Byun, S.W., "An Index Rewriting Scheme Using Compression for Flash Memory Database Systems," Journal of the Information Science, Vol.33 No.4, 2007, pp. 398-415.

[5] Wu, C.H., Chang, L.P., and Ku, T.W., "An Efficient R-Tree Implementation over Flash-Memory Storage Systems," Proc. of the ACM International Symposium on Advances in Geographic Information Systems, 2003, pp. 17-24.

[6] Wu, C.H., Chang, L.P., and Kuo, T.W., "An Efficient B-Tree Layer for Flash-Memory Storage Systems," Proc. of the International Conference on Real-Time and Embedded Computing Systems and Applications, 2003, pp. 409-430.

[7] 김정준, 강홍구, 김동오, 한기준, "메인 메모리 다차원 인덱스를 위한 효율적인 MBR 압축 기법," 한국공간정보시스템학회 논문지, 제9권 제2호, 2007, pp. 13-23.

[8] SAMSUNG Electronics, 512M × 8 Bit / 1G × 8 Bit NAND Flash Memory, 2005.

[9] 이수관, 민상렬, 조유근, "플래시 메모리 관련 최근 기술 동향," 정보과학회지, 제24권 제12호, 2006, pp. 99-106.

[10] Guttman, A., "R-Trees: a Dynamic Index Structure for Spatial Searching," Proc. of the ACM SIGMOD Conference, 1984, pp. 47-54.

[11] Lee, T.W., Moon, B.K., and Lee, S.H., "Bulk Insertion for R-trees by Seeded Clustering," Proc. of the Data and Knowledge Engineering, Vol.59 No.1, 2006, pp. 86-106.

김 정 준

2003년 건국대학교 컴퓨터공학과(공학사)
2005년 건국대학교 대학원 컴퓨터공학과(공학석사)
2005년~현재 건국대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 공간 데이타베이스, GIS, USN

심 희 정

2004년 건국대학교 컴퓨터공학과 졸업(공학사)
2009년 건국대학교 컴퓨터공학과 졸업(공학석사)
현재 KT DataSystems 인프라운영본부 사원
관심분야 : 모바일 시공간 데이타베이스, 공간 인덱스

강 홍 구

2002년 건국대학교 컴퓨터공학과(공학사)
2004년 건국대학교 대학원 컴퓨터공학과(공학석사)
2009년 건국대학교 대학원 컴퓨터공학과(공학박사)
2009년~현재 건국대학교 컴퓨터공학부 강의교수
관심분야 : 공간 데이타베이스, GIS, LBS, USN

이 기 영

1984년 숭실대학교 전자계산학과(공학사)
1988년 건국대학교 대학원 컴퓨터공학과(공학석사)
2005년 건국대학교 대학원 컴퓨터공학과(공학박사)
1984년~1991년 한국해양연구원 연구원
1996년~1998년 한국컴퓨터정보학회 이사 및 서울동부지회장
1991년~현재 을지대학교 의료산업학부 교수
관심분야 : 공간 데이타베이스, GIS, LBS, USN, 텔레매틱스

한 기 준

1979년 서울대학교 수학교육학과(이학사)
1981년 한국과학기술원(KAIST) 전산학과(공학석사)
1985년 한국과학기술원(KAIST) 전산학과(공학박사)
1985년~현재 건국대학교 컴퓨터공학부 교수
1990년 Stanford 대학 전산학과 Visiting Scholar
2000년~2002년 한국정보과학회 데이타베이스연구회 운영위원장
2004년~2006년 한국공간정보시스템학회 회장
2004년~2008년 한국정보시스템감리사협회 회장
관심분야 : 공간 데이타베이스, GIS, LBS, 텔레매틱스, 정보시스템 감리