실시간 모바일 GIS 응용 구축을 위한 주기억장치 데이터베이스 시스템 설계 및 구현

강 은 호[†]·윤 석 우[†]·김 경 창[††]

요 약

최근 들어 계속되는 램 가격 하락으로 인해 대용량의 램을 사용하는 주기억장치 데이터베이스 시스템의 구축이 실현 가능하게 되었다. 주기억장치 데이터베이스는 여러 다양한 실시간 응용 분야를 위해 사용되며, 매년 CPU 속도가 60% 정도 증가되고, 메모리 속도가 10% 증가되는 현실에서, 캐쉬 미스(Cache miss)를 얼마나 줄이느냐 하는 문제가 주기억장치 데이터베이스의 검색 성능 측면에서 가장 중요한 문제로 대두되고 있다. 본 논문에서는 이러한 환경을 고려한 실시간 모바일 GIS응용을 위한 주기억장치 데이터베이스 시스템을 설계 및 구현한다. 본 시스템은 크게 PDA를 사용하는 모바일 사용자를 위한 인터페이스 관리기와 가상 메모리 기법을 사용해 전체 데이터를 주기억 장치에 상주시키며 관리하는 주기억 데이터 관리기, 공간 및 비공간 질의를 처리하는 질의처리기, 새롭게 제시하는 공간 데이터를 위한 MR-트리 인덱스와 비공간 데이터를 위한 T-트리 인덱스 구조를 관리하는 인덱스 관리기, 데이터를 디스크에 저장하기 위한 GIS 서버 인터페이스로 구성된다. 새롭게 제시하는 공간 인덱싱을 위한 MR-트리는 노드 분할이 발생될 경우, 입력 경로 상에 하나 이상의 빈 엔트리를 지니는 노드가 존재할 경우에만, 노드 분할을 상위로 전송한다. 그러므로 중간 노드들은 항상 100%에 가깝게 채워져 있게 된다. 본 논문의 실험 결과, 2차원의 MR-트리는 기존의 R-트리에 비해 2.4배 이상의 빠른 검색 속도를 나타냈다. 한편, 주기억 데이터 관리기는 가상 메모리 제공을 위해 전체 벡터 데이터 및 MR-트리, T-트리, 데이터 객체 텍스트 정보를 페이지 단위로 분할하여 관리하고, 간접 주소 기법을 사용하여 디스크로부터의 재로딩시 발생할 수 있는 문제점을 제거하였다.

Design and Implementation of a Main-Memory Database System for Real-time Mobile GIS Application

Eun-Ho Kang[†]·Suk-Woo Yun[†]·Kyung-Chang Kim[††]

ABSTRACT

As random access memory chip gets cheaper, it becomes affordable to realize main memory-based database systems. Consequently, reducing cache misses emerges as the most important issue in current main memory databases, in which CPU speeds have been increasing at 60% per year, compared to the memory speeds at 10% per year. In this paper, we design and implement a main-memory database system for real-time mobile GIS. Our system is composed of 5 modules : the interface manager provides the interface for PDA users ; the memory data manager controls spatial and non-spatial data in main-memory using virtual memory techniques ; the query manager processes spatial and non-spatial query ; the index manager manages the MR-tree index for spatial data and the T-tree index for non-spatial index ; the GIS server interface provides the interface with disk-based GIS. The MR-tree proposed propagates node splits upward only if one of the internal nodes on the insertion path has empty space. Thus, the internal nodes of the MR-tree are almost 100% full. Our experimental study shows that the two-dimensional MR-tree performs search up to 2.4 times faster than the ordinary R-tree. To use virtual memory techniques, the memory data manager uses page tables for spatial data, non-spatial data, T-tree and MR-tree. And, it uses indirect addressing techniques for fast reloading from disk.

키워드 : 주기억장치 데이터베이스(Main-memory Database), 인터페이스 관리자(Interface Manager), 주기억 데이터 관리자(Memory Data Manager), 질의 처리기(Query Processor), 인덱스 관리기(Index Manager), 공간 인덱스(Spatial Index), 비공간 인덱스(Non-Spatial Index), 가상 메모리, 공간 질의(Spatial Query), 비공간 질의(Non-Spatial Query)

1. 서 론

실시간 모바일 GIS와 같은 모든 실시간 응용들의 주 요구사항은 실시간 위치 정보 수정과 질의에 대한 아주 빠른 응답시간이다. 그러나 기존의 디스크 기반 데이터베이스 시스템은 빈번한 디스크 I/O로 인해 이러한 요구사항을 만족시키지 못한다. 주기억 기반의 데이터베이스 시스템은 실시간 응용들의 위와 같은 요구사항을 충족시키기 위해 제안되었으며, 최근 계속되는 램 가격의 하락은 점점 더 주기억

※ 이 논문은 2003학년도 홍익대학교 교내연구비에 의하여 지원되었음.
† 준 회 원 : 홍익대학교 대학원 컴퓨터공학과
†† 정 회 원 : 홍익대학교 컴퓨터공학과 교수
논문접수 : 2003년 5월 10일, 심사완료 : 2003년 10월 2일

장치 데이터베이스 시스템의 구축을 앞당겼다. 최근 연구[8]에서는 10년 이내에 수백 테라바이트(terabyte) 데이터베이스를 위해 테라바이트 단위의 메모리 칩을 제공하는 일이 흔해 질 거라고 예고하고 있다.

요즘의 컴퓨터 시스템들은 CPU와 상대적으로 큰 주기억장치사이에 캐쉬(Cache) 메모리를 사용한다. 이 캐쉬는 작고 빠른 SRAM으로 구성되며, 최근에 접근했던 데이터들을 저장함으로써 주기억 장치보다 더 빠른 접근 시간을 보장한다. 이와 같이 캐쉬와 주기억 장치 사이의 접근 속도 차이가 있음을 감안할 때, 빠른 검색을 위해서는 캐쉬 미스를 줄이는 것이 현재의 주기억장치 데이터베이스 환경에서 가장 중요한 문제점임을 알 수 있다.

과거 주기억장치 데이터베이스 시스템에서의 인덱싱 기법은 인덱스 노드를 접근하는데 걸리는 시간과 노드안의 키들을 비교하는데 걸리는 시간을 같은 정도의 비율로 생각했기 때문에, T-트리를 포함한 그의 변형 모델들이 제안되었다. 그러나 요즘은 CPU의 속도가 매년 60% 정도 증가하는 반면 메모리의 속도가 매년 10%정도 증가하기 때문에, 인덱스안의 키들을 비교하는데 걸리는 시간의 부담이 상대적으로 점점 더 줄어들고 있다[6]. 참고 논문[4]에서는 한 예로 여러 상업용 주기억장치 데이터베이스 관리 시스템들의 성능을 분석했다. 그 결과로 레벨 2 캐쉬 미스 발생 빈도와 레벨 1 명령어 캐쉬 미스 발생 빈도가 실행 시간의 중요 부분을 차지하고 있다는 결론에 이르렀다. 이러한 점을 볼 때 캐쉬 동작은 인덱싱 기술의 가장 중요한 문제점들 중 하나 임을 알 수 있다[3]. 이 말은 요즘의 인덱싱 기법에서는 인덱스 키 비교 시간을 줄이는 일보다 인덱스 노드 접근 시간을 줄이는 일이 더 중요하다는 것을 의미한다. CSB^+-트리(Cache sensitive B^+-트리)[5]는 이러한 환경을 고려해 B+-트리의 변형 모델로 제시되었다. CSB^+-트리는 첫 번째를 제외한 모든 자식 노드들에 대한 포인터들을 제거하기 위해 자식 노드들을 순서대로 메모리 상에 연속적으로 저장시킨다. 포인터 제거로 인해 발생되는 노드내의 빈공간은 인덱스 키들로 채워지기 때문에, 이러한 방법은 B^+-트리의 노드내의 저장 가능한 엔트리 수를 두 배로 늘려 준다. 만약 캐쉬 블록 크기와 같은 인덱스 노드 크기를 사용한다면, 저장 가능 엔트리수가 두 배로 늘어난 인덱스 트리는 트리의 높이를 줄여주고, 트리를 순회할 때 발생 가능한 캐쉬 미스의 수를 최소로 줄여준다.

GIS 인덱스 구조를 위해 R-트리를 주기억 인덱싱 기법으로 사용할 경우, 위에서 사용한 포인터 제거 기술만으로는 트리의 높이를 크게 줄여주지 못한다. 이는 MBR(Minimum Bounding Rectangle)이라 불리는 다차원 공간 인덱스 키가 포인터에 비해 훨씬 더 큰 공간을 차지하기 때문이다[1]. 본 논문에서 제시하는 시스템에서는 이러한 컴퓨팅 환경을 고려해 R-트리를 캐쉬 동작에 민감하도록 MR-트리를 제안하고 사용한다. MR-트리는 R-트리와 비교해 뚜렷한 두 가지의 특성을 지닌다. 첫째로 MR-트리는 리프 노드의 분할 발생시, 입력 경로상의 노드들 중 빈 엔트리를 지니는 노드가 존재할 경우에만 노드 분할을 상위 노드로 전달한다. 만약 입력 경로상의 노드들이 빈 엔트리를 지니지 않을 경우, 분할이 발생된 리프 노드는 반-리프(half-leaf) 노드가 되며, 이 노드는 분할로 인해 새롭게 생성된 리프 노드와 데이터 객체들 모두를 자식으로 지닌다. 만약 새로운 객체가 반-리프 노드 또는 반-리프 노드를 부모 노드로 갖는 리프 노드에 계속해서 추가될 경우, 반-리프 노드는 리프 노드들만을 자식으로 갖는 중간 노드(internal node)로 변환된다. MR-트리는 입력되는 객체들의 순서에 따라 중간 노드들의 높이 차이가 증가될 수 있다. 이러한 이유 때문에, 두 번째 특징으로 MR-트리에서는 만약 어떤 노드의 자식 노드들의 높이 차이가 2일 경우 높이 균형화(height balancing) 알고리즘을 수행한다. 이러한 특성으로 MR-트리는 루트에서부터 모든 중간 노드들까지 자식 노드들의 높이 차이가 최대 1로 한정되는 높이 균형 트리 형태를 유지한다.

일반적인 GIS 데이터는 이미지 기반과 벡터 기반으로 나뉠 수 있다. 이미지 기반은 디스플레이를 위한 별도의 오퍼레이션이 필요 없지만 기존 자료를 수정하는데 용이하지 않다는 점과 상대적으로 벡터 데이터에 비해 큰 저장 공간을 필요로 한다는 단점이 있다. 본 시스템은 디스크에 비해 상대적으로 가격이 비싼 주기억 장치에 주요 데이터베이스를 저장해야하기 때문에 벡터 데이터를 지리 정보를 나타내기 위한 기본 구조로 사용한다. 이러한 벡터 데이터는 본 시스템의 주기억 장치 관리자를 통해 페이지 단위로 관리되며, 가상 메모리를 제공함으로써 큰 크기의 벡터 데이터들을 주기억 장치에 저장할 수 있도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구들에 대해 간략하게 언급하고, 3장에서는 본 논문에서 제시하는 주기억장치 데이터베이스 시스템을 각 관리기별로 구분지어 설명한다. 4장에서는 본 시스템에서 사용하는 MR-트리의 성능을 실험을 통해 비교 평가하고, 5장에서는 결론 및 향후 연구과제에 대해 언급한다.

2. 관련 연구

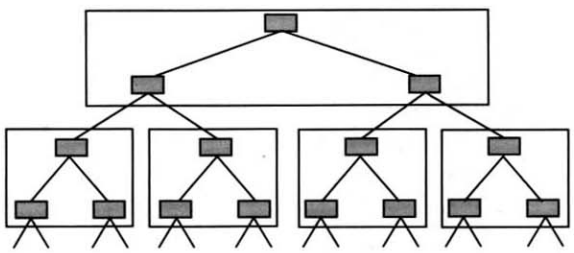
본 장에서는 주기억장치 데이터베이스 시스템의 검색 성능을 좌우하는 인덱스 기법에 대해 제시된 기법들을 조사하여 문제점을 제시하고, 주기억 장치 내의 공간 데이터 저장 기법 제시를 위해 기존의 비공간 데이터 저장 기법에 대한 관련 연구를 수행한다.

2.1 캐쉬 동작에 민감한 인덱싱 기법들

2.1.1 기존의 기법들

클러스터링(Clustering) 기법[2, 7]에서는 동시에 접근되는

데이터 구조들을 하나의 캐쉬 블록 내에 저장한다. 이 기법은 공간적 지역적으로 가까운 데이터들에 대해 한번의 캐쉬 블록 불러오기로 동시에 접근이 가능하기 때문에 암시적인 미리-불러오기(prefetch)를 제공한다. 트리 구조를 클러스터링 할 때 효율적인 방법은 인덱스 노드와 그 서브 노드들을 하나의 캐쉬 블록내에 저장하는 것이다. (그림 1)에서는 이진 트리에서 서브 트리에 대한 클러스터링 기법을 보여준다.



(그림 1) 서브 트리 클러스터링

문자와 숫자와 같은 1차원 데이터 타입을 위한 인덱스 구조를 생성할 때 이러한 클러스터링 기법은 효율적으로 적용된다. 그러나 캐쉬 블록 크기가 32~64바이트임을 감안하면, R-트리에 클러스터링 기법을 사용하는 것은 현실적으로 불가능하다. 만약 2차원 인덱스 구조를 생성한다고 할 경우, 각 차원의 데이터 값을 표시하기 위해서는 4바이트가 필요하기 때문에, 하나의 엔트리를 표현하기 위해서는 각 차원의 최소, 최대 값으로 이루어진 16바이트의 MBR과 4바이트의 포인터 저장 공간이 필요하다. 이와 같이 R-트리에서는 하나의 엔트리를 표현하기 위해 20바이트가 필요하기 때문에, 만약 캐쉬 블록 크기가 128바이트로 주어진다면, 클러스터링 기법을 적용하기 위해서는 하나의 노드에는 단지 2개의 엔트리만을 저장할 수 있다. 그러므로 서브 트리 클러스터링 방법을 R-트리에 적용시킬 경우에 트리의 높이는 심각하게 커질 것이며 이는 결국 캐쉬 미스 발생을 증가시킬 수 있다.

컬러링[2]이란 동시에 접근되는 인덱스 노드들을 캐쉬내에 서로 겹치지 않는 지역에 저장하는 방법이다. 만약 2-컬러링 방법을 사용할 경우, 자주 접근되는 노드들은 첫 번째 캐쉬 지역에 저장하고 나머지 노드들은 다른 캐쉬 지역에 저장한다. 이러한 메핑 방법은 자주 접근되는 노드들이 서로 충돌하지 않고, 자주 접근되지 않는 노드들에 의해 바뀌지 않는 것을 보장한다. 그러나 시스템 커널이 캐쉬 교환 정책을 수행하기 때문에, 이 방법 역시 공간 인덱싱 기법에 즉각적인 적용이 불가능하다.

압축 기법[1, 2]은 하나의 인덱스 노드내에 더 많은 엔트리들을 저장할 수 있게 한다. 이러한 기법은 캐쉬 블록뿐만 아니라 공간 이용율을 증가시켜준다는 장점이 있지만, 키 값들을 압축하고 압축해제하기 위해 더 많은 프로세서 동작이 필요하다는 단점이 있다. 그러나 요즘의 환경에서는 프로세서가 계산하는데 드는 비용이 메모리 접근 비용에 비해 상대적으로 싸기 때문에 상당히 긍정적인 해결책으로 고려할 수 있다. 압축 기법은 키 압축 기법, 포인터 제거 기법 그리고 핫/콜드 노드 분할 기법 등으로 제안되었다.

2.1.2 CR-트리(Cache-conscious R-tree)

CR-트리[1]는 R-트리의 캐쉬 동작에 민감한 변형 버전에 해당한다. 하나의 노드내에 더 많은 엔트리를 저장하기 위해, CR-트리는 QRMBR(Quantized Relative MBR)이라 불리는 손실 발생이 가능한 압축 기법을 사용한다. CR-트리의 인덱스 노드는 노드안 엔트리들을 모두 포함하는 최소 직사각형인 참조 MBR(reference MBR), 그리고 QRMBR 키와 자식 노드들에 대한 포인터로 구성된 엔트리들로 표현된다. 일반적으로 1 바이트 또는 2바이트로 되는 양자화(Quantization) 레벨을 L로 두고, 참조 MBR을 R로 둔다면, 2차원 인덱싱 기법에서 MBR의 x축 최소 경계값 xl은 다음과 같이 표현된다.

$$if\,(MBR.xl \le R.xl)\; QRMBR.xl = 0$$
$$else\; if(MBR.xl \ge R.xh)\; QRMBR.xl = L-1$$
$$else\; QRMBR.xl = \lfloor L(MBR.xl-R.xl)/(R.xh-R.xl)\rfloor$$

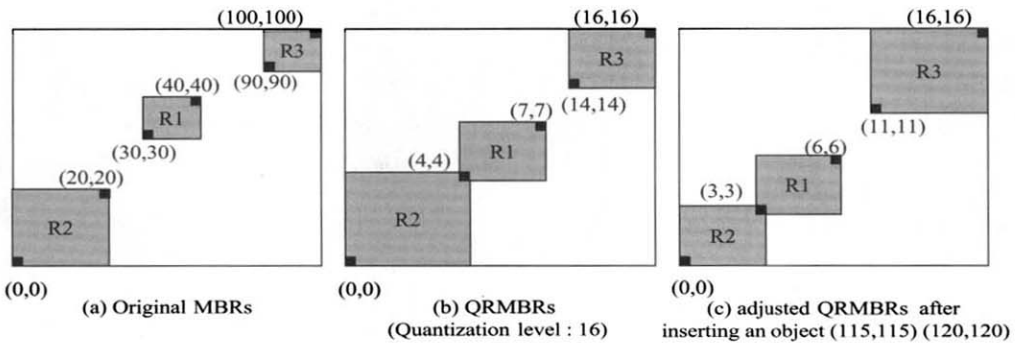
MBR x축의 최대 경계값 xh에 대한 표현은 다음과 같다.

$$if(MBR.xlh \le R.xl)\; QRMBR.xh = 0$$
$$else\; if(MBR.xh \ge R.xh)\; QRMBR.xl = L$$
$$else\; QRMBR.xh = \lfloor L(MBR.xh-R.xl)/(R.xh-R.xl)\rfloor$$

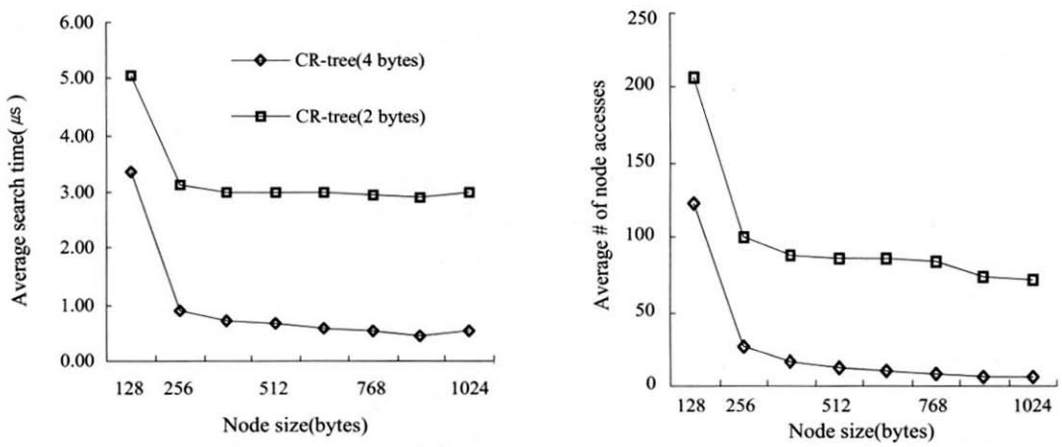
CR-트리는 검색 알고리즘에서 각 방문하는 노드의 참조 MBR을 이용해 검색 사각형을 해당 노드에 해당하는 QRMBR로 변경시킨 다음 검색을 수행한다. 삽입 알고리즘에서는 인덱스 노드를 방문할 때 마다, 새롭게 삽입되는 객체에 대한 MBR은 해당 인덱스 노드의 참조 MBR을 이용해 QRMBR로 변경시킨 다음, 이를 포함시키기 위해 최소의 크기 증가가 필요하는 자식 노드를 찾는다. 만약 이러한 방법으로 리프 노드까지 찾았다면, 우선 이 리프 노드의 참조 MBR은 삽입될 객체의 MBR을 포함한 최소 사각형의 형태가 되도록 수정된다. 그런 다음, 해당 객체를 위한 새로운 엔트리가 노드내에 생성된다. 만약 노드의 참조 MBR이 삽입된 객체로 인해 커졌다면, 이전 참조 MBR를 통해 QRMBR들에 대한 원래의 MBR을 계산 한 다음, 수정된 참조 MBR을 이용해 모든 QRMBR들이 재 계산된다. 만약 삽입시 노드내에 빈 엔트리가 없다면, 해당 노드는 두개의 노드로 분할된다. 이 경우 분할에 따라 참조 MBR이 수정되며, 노드안의 QRMBR들 역시 참조 MBR에 따라 다시 재 계산된다. 이 분할 정보는 필요하다면 루트까지 상위 레벨로 전달된다.

CR-트리의 문제점은 QRMBR이 재 계산 될 때마다 커질 수 있다는 점에 있다. (그림 2)는 그 한 예를 보여준다.

(그림 2)(a)는 어떤 중간 노드의 원래의 MBR 정보를 표현한 것이며, (그림 2)(b)는 이를 QRMBR 형태로 표현한 것에 해당한다. 만약 최소 값으로 (115, 115)를 가지고, 최대 값으로 (120, 120)를 가지는 새로운 객체 MBR이 삽입될 경우, CR-트리는 그림의 R3를 삽입 경로로 선택한다. 그런 다음, CR-트리는 이 노드 경계 MBR이 커졌기 때문에, 참조 MBR과 QRMBR을 재 계산하게 된다. 이 경우 참조 MBR을 재 계산하기위해 우선 CR-트리는 QRMBR 생성 기술을 역으로 적용시켜 원래의 MBR 값을 계산한 다음, 참조 MBR을 조정한다. 그런 다음, 조정된 참조 MBR과 계산해낸 원래의 MBR 값을 이용해, QRMBR을 재 계산한다. (그림 2)(c)는 이러한 일련의 계산을 거쳐 새롭게 표현된 노드의 모습을 나타낸다. 여기서 R1의 경우, CR-트리는 최소 값으로 (25, 25)를, 최대 값으로 (43.75, 43.75)를 원래의 MBR 값으로 계산한 다음, 조정된 참조 MBR을 이용해 (그림 2)(c)와 같은 QRMBR을 계산한다. 그러나 여기서 R1의 바른 QRMBR 최소 값은 (4, 4)로 구성되어야 한다. 이와 같이 잘못되게 커진 QRMBR은 CR-트리의 동작에 나쁜 영향을 미친다. 만약 위의 (그림 2)(c) 상황에서 최소 값으로 (20, 20), 최대 값으로 (23, 23)을 지닌 새로운 객체 MBR이 삽입될 경우, 우선 새로운 MBR은 최소 값으로 (2, 2), 최대 값으로 (4, 4)를 지닌 QRMBR로 변경된다. R-트리의 경우에는 이 새로운 MBR을 삽입하기 위한 경로로 R2를 선택하지만, CR-트리는 R1을 선택할 수 있다. 이는 R1 역시 새로운 객체 QRMBR를 포함하기 위한 최소 직사각형 증가를 가져오기 때문이다. 이러한 이상현상은 루트 노드에 가까운 상위 중간 노드일수록 자주 발생할 수 있다. 왜냐하면 이러한 중간 노드들의 QRMBR은 하위 노드들의 QRMBR보다 상대적으로 더 크고 삽입하려는 객체는 이러한 노드들에서 더욱 작은 QRMBR로 표현되기 때문이다. 이러한 이상 현상은 또 노드의 분할시에도 잘못된 결과를 초래할 수 있다. 이렇게 되면 질의 사각형과 겹치는 객체들을 검색할 경우, 이러한 이상현상은 방문해야 하는 노드의 수를 상당히 증가시킬 수 있다. 결과적으로 CR-트리는 공간 사용 측면에서는 좋은 성능을 항상 보장하지만, 작은 크기의 QRMBR로 인해 검색 성능 저하를 초래할 수 있다. 서로 다른 크기의 QRMBR을 사용하는 CR-트리를 비교하기 위해 본 논문에서는 단위 크기의 정사각형 내에 측면 길이가 0.001인 백만 개의 작은 사각형을 균등 분포를 갖도록 생성했다. (그림 3)은 QRMBR의 크기를 4바이트와 8바이트로 하였을 경우를 비교한 결



(그림 2) CR-트리 이상 현상 예제



(그림 3) 서로 다른 QRMBR 크기를 사용할 경우 검색 성능 및 접근 노드의 수 비교

과이다. 여기서 사용한 질의 검색 사각형의 크기는 전체 1 평방 크기의 0.01%에 해당한다. 이 실험 결과에서는 4바이트의 QRMBR을 사용하였을 경우가 8바이트의 QRMBR을 사용하였을 경우보다 접근해야 되는 노드의 수가 더 많음을 알 수 있으며, 이는 앞의 이상 현상으로 설명할 수 있다.

2.2 주기억 장치 내에 데이터 저장 기법

2.2.1 데이터 압축 및 클러스터링

지리정보와 같은 대용량의 데이터를 제한된 용량의 주기억 장치 내에 효율적으로 저장하기 위한 방안으로 데이터에 대한 압축 기법을 들 수 있다. 만약에 효율적으로 데이터에 대한 압축이 이루어진다면, 이는 부수적으로 네트워크 전송 속도 증가라는 이득을 가져온다. 지금까지 문자의 반복 횟수에 기반을 둔 많은 문자 압축 기법들이 제시되어 왔다[11, 12]. 그러나 이러한 연구들은 문자열에 대한 압축 기법으로 실수 형으로 주어지는 벡터 데이터에는 부적절하고, GNU gzip과 같은 압축 기법을 사용해 실수를 압축할 경우에는 질의 처리시 압축 해제로 인한 속도 저하라는 문제점을 가져온다.

2.2.2 Dali 주기억 장치 관리 시스템

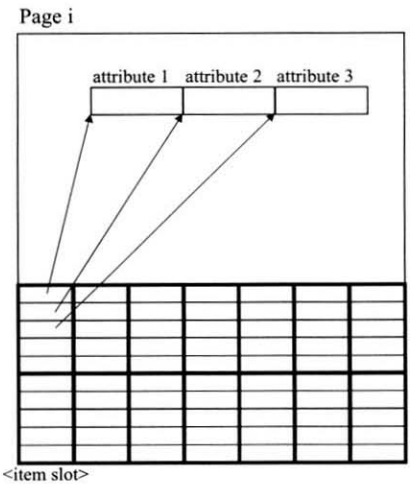
Dali[9, 10] 시스템은 AT&T Bell 연구소에서 개발한 주기억 DBMS 시스템으로 전체 데이터베이스를 여러 개의 데이터베이스 파일로 구성한다. 데이터베이스 백업은 데이터베이스 파일 단위로 수행하며, 주기억 장치로 로드시에는 mmap이라는 Unix 표준 시스템 함수를 이용해 페이지별로 로드시킨다. 그리고 이 mmap Unix 표준 함수는 가상 기억장치를 제공하는 기능으로도 사용된다.

이 시스템에서 하나의 데이터베이스 파일은 여러개의 파티션(partition)들로 구성되며, 각 파티션은 파티션 테이블, 메타 데이타 파티션, 데이터 파티션, 오버플로우 공간, 빈 공간으로 구성된다. 메타 데이타 파티션에는 데이터 파티션에 저장되는 데이터 레코드에 대한 타입, 크기, 데이터 파티션내의 위치정보, 록킹여부 정보를 저장한다. 오버플로우 공간에는 메타 데이터 파티션과 데이터 파티션에서 오버플로우가 발생할 경우 이를 저장한다.

Dali 시스템에서의 데이터베이스 파일의 식별자는 4바이트의 기계위치 주소와 4바이트의 기계내의 지역 주소로 나뉘어 관리된다. 즉 전체 데이터베이스가 주기억 장치내의 특정 지역에서만 유지되는 것이 아니라, 데이터베이스가 여러 데이터베이스 파일들로 나뉘어 동적으로 여러 지역에 분산되어 저장 유지될 수 있도록 한다. 데이터 레코드에 대한 접근을 위해 DBPtr과 itemID라는 2가지 타입의 데이터베이스 포인터를 제공한다. DBPtr은 데이터베이스 파일내의 식별자와 데이터베이스 파일내의 간접 주소로 구성되어, 사용자에게 데이터 접근을 제공한다. itemID는 데이터 레코드 헤더에 대한 포인터로 간접 데이터 접근을 제공한다.

2.2.3 Starburst 주기억 장치 상주 시스템

Starburst[13] 시스템에서는 전체 메모리를 논리적 단위인 세그멘트 단위로 나누고, 하나의 테이블에 속한 데이터 아이템들은 하나의 세그먼트내에 저장된다. 하나의 세그먼트는 다시 여러 개의 페이지들로 구성되는데 각 페이지는 (그림 4)에서와 같이 아이템 슬럿들의 배열과 빈 공간으로 구분된다.

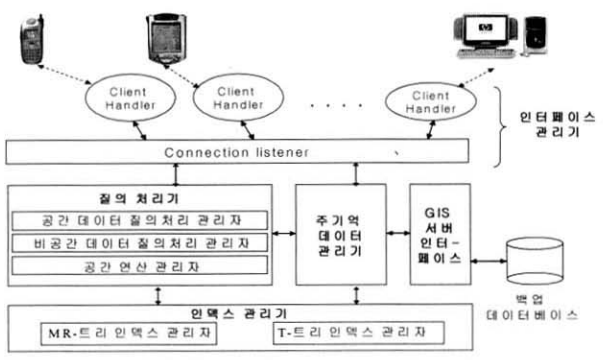


(그림 4) Starburst 페이지 구조

Starburst 시스템에서는 가변 길이의 데이터 아이템을 지원하기 위해 우선 해당 테이블의 아이템들이 저장된 세그멘트의 페이지들에서 빈 아이템 슬럿과 실제 아이템 값이 저장될 공간을 찾는다. 그런 다음, 선택된 페이지의 빈 슬럿에 아이템 구조를 저장하고, 각 애트리뷰트의 값들은 같은 페이지의 빈 공간에 삽입한다. 그런 후, 아이템 슬럿에서 애트리뷰트의 값들을 가리키도록 포인터를 연결한다.

3. 실시간 모바일 GIS를 위한 주기억장치 데이터베이스 시스템

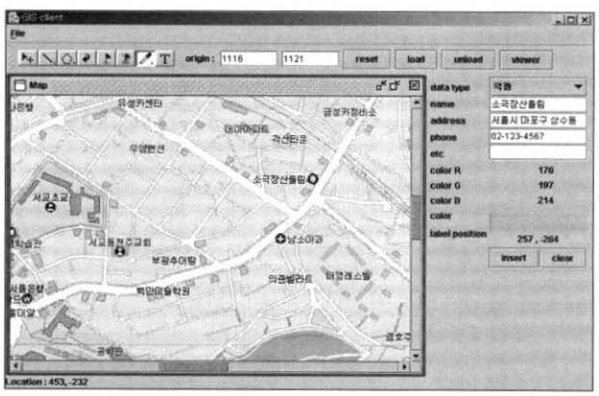
본 논문에서 설계한 주기억장치 데이터베이스 시스템은 (그림 5)에서와 같이 크게 5개의 관리기로 구성된다.



(그림 5) 주기억장치 데이터베이스 시스템 구성도

(그림 5)의 주기억장치 데이터베이스 시스템은 실시간 모발 지리 정보 시스템을 위한 시스템으로 자료의 입력 및 다양한 지리정보 관련 검색을 지원한다. 지도 데이터는 벡터 데이터를 사용하며, 데이터베이스 시스템 관리자는 유선으로 연결된 PC급 컴퓨터를 이용해 벡터 데이터와 지명, 주소와 같은 관련된 지리 정보를 입력한다. 입력된 자료는 인터페이스 관리자를 통해 주기억 데이터 관리기로 전송되며, 주기억 데이터 관리기는 페이지 단위로 전송받은 벡터 데이터와 텍스트 형태의 지리 정보 데이터를 저장한다. 인덱스 관리기는 주기억 데이터 관리기로부터 요청받아 학교, 은행, 음식점과 같은 각각의 지리 정보 타입별로 벡터 데이터에 대한 MR-트리 인덱스와 지리 정보에 대한 T-트리 인덱스를 생성한다. 여기서 MR-트리와 T-트리 인덱스 역시 백업 및 로딩 시간을 최소화하기 위해 주기억 데이터 관리기를 통해 페이지 단위로 인덱스 저장 공간을 할당받아 저장하고, 주기억 데이터 관리기는 GIS 서버 인터페이스를 통해 벡터 및 지리정보, 인덱스 자료를 백업 데이터베이스에 저장한다.

질의 처리기에 의해 수행되는 지리 정보 검색은 크게 지명을 이용한 검색, 디스플레이된 화면내의 지명 타입을 이용한 검색 그리고 사용자의 자기 위치 정보를 이용한 검색을 지원한다. 이를 위해 무선 모바일 환경의 클라이언트는 우선 자신의 위치 정보 또는 검색어를 입력해 서버로 전송하고, 서버의 인터페이스 관리기는 이를 받아 질의 처리기로 전달한다. 질의 처리기는 인덱스 관리기를 통해 검색하고자 하는 벡터 데이터와 지리 정보 데이터에 대한 주소값을 알아 낸 다음, 이를 이용해 주기억 데이터 관리기로부터 원하는 데이터 값을 얻어, 인터페이스 관리기로 전달한다. 인터페이스 관리기는 각 무선 모바일 클라이언트들을 대신해 자신의 버퍼에 결과 지도를 직접 드로우(draw)한 다음, 각각의 클라이언트로 전송해 디스플레이하게 한다.



(그림 6) GIS 지도 입력

실례로 (그림 6)과 (그림 7)은 본 논문에서 구현한 공간 및 비공간 데이터 입력화면과 비공간 질의를 위한 클라이언트측 GIS 뷰어 화면을 보여준다. (그림 6)의 지도 입력화면을 통해 사용자는 지리 정보를 저장, 삭제, 갱신하며, (그림 7)의 GIS 뷰어 화면을 통해 지리 정보를 검색한다. (그림 7)은 기본적인 확대, 축소 기능과 뷰어 화면에서 마우스로 드래그하면 위치를 이동할 수 있는 기능, 그리고 상세 찾기 메뉴 등의 기능들을 통해서 지리 정보를 검색할 수 있도록 하는 기능으로 구성되어 있다.

(그림 7) GIS 뷰어 화면

3.1 인터페이스 관리기

인터페이스 관리기는 클라이언트 핸들러(Client Handler)와 연결 수신자(Connection listener)로 구성된다. 클라이언트 핸들러는 지리정보 입력 역할을 수행하는 유선 환경의 PC급 컴퓨터와 지리정보 검색 역할을 수행하는 PDA들에 대해 각각 별도로 생성되어 유지되며 연결 수신자를 통해 질의 처리기와 주기억 장치 관리기와 연결된다.

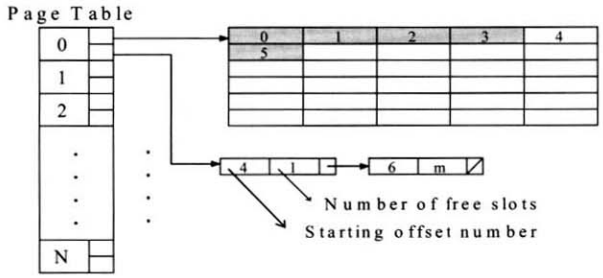
유선 환경의 클라이언트 컴퓨터는 지리 정보에 대한 벡터 데이터와 해당 지명 등의 위치정보에 대한 자료를 입력받아 서버로 전송하고, 클라이언트 핸들러는 이를 받아 연결 수신자를 통해 주기억 데이터 관리기로 전송하여 자료 저장을 지시한다. 그러므로 유선 환경의 클라이언트 PC를 위한 클라이언트 핸들러는 주기억 데이터 관리기와 단순한 연결 기능만을 수행한다.

PDA와 같은 무선기기 클라이언트를 위한 플랫폼으로는 j2me(Java2 Platform, Micro Edition)를 사용한다. j2me는 자바 기술 중의 하나로써 j2se(Java2 Platform, Standard Edition)의 모든 스펙을 지원하지 않는데, 이는 기기 자체의 제한된 자원으로 인한 플랫폼의 축소가 필연적이기 때문이다. 따라서 j2me 플랫폼에서의 GUI 기능은 많이 축소되어 있어서 기본적인 사용자의 입력을 받을 수 있는 정도로 구성되어 있으며, 특히 Graphics 객체는 j2se에 비해 많이 제한되어 있다. 본 시스템에서는 이러한 PDA와 같은 클라이언트의 제한성을 감안하여 서버 측의 클라이언트 핸들러가 대신 데이터를 검색하고 버퍼에 저장하여 직접 메모리에 draw해서 이미지를 클라이언트에 전송하는 방식으로 설계했다. 그러므로 PDA를 위한 클라이언트 핸들러는 각각의

데이터 버퍼 공간을 유지한다.

3.2 주기억 데이터 관리기

주기억 데이터 관리기는 가상 메모리 기법을 사용해 지리 정보에 대한 데이터와 인덱스 구조들을 메모리내에 상주시킨다. 이를 위해 본 논문의 주기억 데이터 관리기는 4개의 페이지 테이블들을 관리하는데, 이는 벡터 데이터를 위한 페이지 테이블, 벡터 데이터에 대한 정보에 해당하는 텍스트 형태의 지리 정보를 위한 페이지 테이블, 벡터 데이터에 대한 인덱스 구조인 MR-트리를 위한 페이지 테이블, 그리고 텍스트 정보를 위한 T-트리를 위한 페이지 테이블로 구성된다.



(그림 8) 페이지 테이블

각각의 페이지 테이블은 (그림 8)과 같이 일정 크기의 레코드들의 배열로 구성되며, 각 페이지 내에는 빈 레코드들에 대한 정보를 페이지내의 빈 레코드 시작 오프셋과 빈 레코드의 개수 형태로 이루어진 연결된 리스트 형태로 관리한다. 만약 새로운 데이터나 인덱스 자료 입력으로 인해 빈 레코드가 필요할 경우, 주기억 데이터 관리기는 페이지 테이블에서 빈 레코드가 있는 페이지 번호와 해당 페이지 내의 레코드 위치에 해당하는 오프셋 정보를 이용해 새로운 데이터를 저장한다. 그러므로 질의 처리기를 통해 자료를 검색할 경우 해당 데이터에 대한 주소 값은 (페이지 시작주소+오프셋 값) 형식으로 찾아가는 간접주소 기법을 사용한다. 이는 가상 메모리를 사용하고, 지리 정보 자료에 대한 백업 및 재 로딩을 위해 고안되었다. 만약 직접 주소 기법을 사용한다면, 디스크로 백업받은 데이터를 재 로딩할 경우, 해당 데이터에 대한 메모리 위치는 동적으로 공간을 할당하는 운영체계에 의해 결정되므로 재 로딩시 데이터에 대한 인덱스 정보 역시 재 로딩시 마다 수정이 필요하다는 문제점이 발생하지만, 본 논문과 같은 간접 주소 기법을 사용하면 메모리로 재 로딩시마다 페이지의 시작 주소만이 동적으로 변경되므로 인덱스 정보의 재수정이 필요 없다는 장점이 있다.

벡터 데이터의 경우에는 하나의 지리정보 객체를 표현하기 위해서는 여러 개의 레코드들이 필요할 수 있는데, 이를 위해 각 레코드들은 서브 레코드에 대한 주소 값을 간접 주소 형태로 가짐으로써 이를 표현한다.

MR-트리와 T-트리의 경우, 하나의 노드 크기 단위의 메모리를 할당받아 인덱스 엔트리들을 저장하는 기법은 메모리 사용측면에서 더 효율적이라고 할 수 있지만, 가상 메모리를 제공하고, 인덱스 자료를 백업 및 재 로딩하는데 많은 시간이 소요된다는 문제점이 있다. 그러므로 MR-트리와 T-트리 인덱스 구조 역시 위의 벡터 데이터의 경우와 마찬가지로 페이지 단위로 할당해 관리한다.

3.3 질의 처리기

질의 처리기는 크게 공간 데이터를 위한 질의 처리 관리자와 비공간 데이터를 위한 질의 처리 관리자 그리고, 공간 연산자로 구성된다. 공간 데이터 질의 처리 관리자는 클라이언트측 PDA에 붙여진 GPS를 통해 입력받은 사용자의 위치정보를 인터페이스 관리기를 통해 입력 받은 다음, 이를 이용해 현재 사용자 위치에 근접한 벡터 데이터를 검색한다. 비공간 데이터 질의 처리 관리자는 클라이언트 단말기로부터 입력받은 주소, 지명과 같은 조건에 일치하는 공간 데이터와 비공간 데이터를 검색하며, 공간 연산자는 현재 사용자의 위치로부터 가장 근접한 특정 타입의 객체를 검색한다.

3.4 GIS 서버 인터페이스 관리기

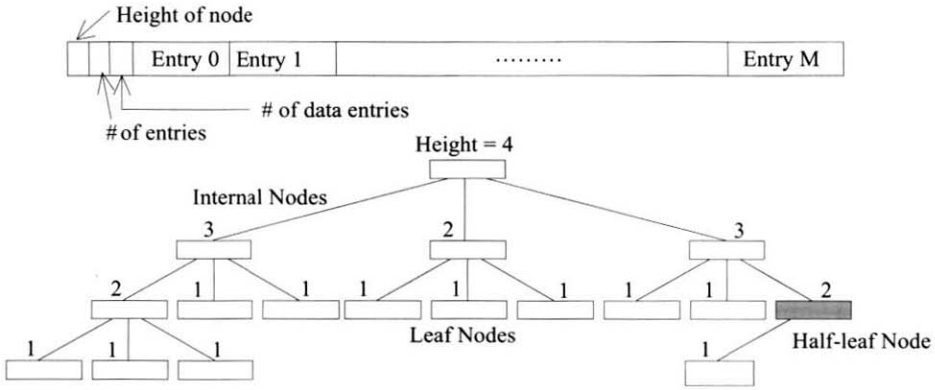
GIS 서버 인터페이스 관리기는 주기억 데이터 관리기로부터 지리 정보에 대한 벡터 데이터값 및 텍스트 데이터값 그리고 이들에 대한 인덱스 정보를 디스크에 저장시키고, 로딩시키는 인터페이스 역할을 수행한다.

3.5 인덱스 관리기

본 논문의 주기억장치 데이터베이스 시스템은 2종류의 인덱스를 관리한다. 입력된 객체에 대해 이 위치정보에 대한 벡터 데이터를 위해서는 기존의 R-트리를 케쉬 동작에 민감하도록 변경시킨 MR-트리를 제안하고 사용하며, 객체 자체의 텍스트 형태 정보를 위해서는 기존 연구들에서 주기억 인덱싱 기법으로 제시된 T-트리[14]를 사용한다.

3.5.1 MR-트리

MR-트리의 인덱스 노드들은 (그림 9)와 같이 중간 노드(internal node)들, 리프 노드(leaf node)들, 그리고 반-리프 노드(half-leaf node)들로 분류된다. 여기서 반-리프 노드라 함은 리프 노드들과 데이터 객체에 대한 엔트리들을 모두 지니는 노드를 의미한다. MR-트리의 노드 구조는 R-트리의 노드 구조와 구별되는 두 가지의 추가적인 필드가 존재한다. 하나는 노드의 높이(Height of node)를 나타내는 필드로 자식 노드들의 높이들 중 가장 큰 높이의 값에 1을 더한 값을 지니며, MR-트리의 불균형 상태를 검색하는데 사용된다. 반-리프 노드에 의해 사용되는 데이터 엔트리 개



(그림 9) MR-트리 구조

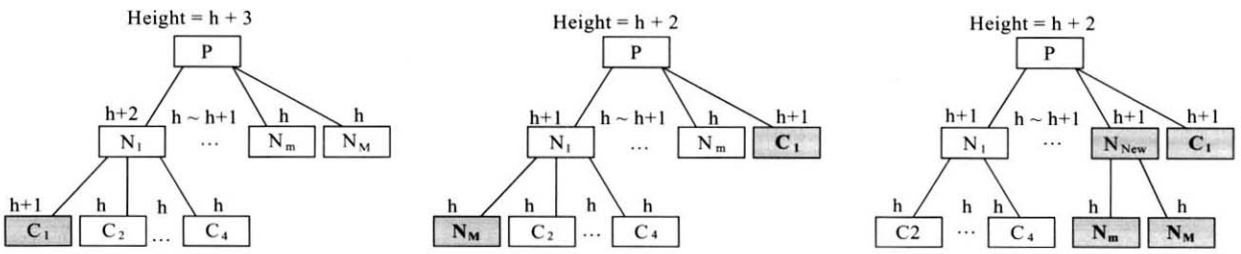
수(# of data entries) 필드는 노드 안에 저장하고 있는 데이터 객체를 가리키는 엔트리의 개수를 저장한다. 한편, 엔트리 개수(# of entries) 필드는 노드 안자식 노드를 가리키는 엔트리의 개수를 의미한다. 반-리프 노드내의 엔트리들이 자식 노드를 위한 엔트리인지, 데이터 객체를 위한 엔트리인지를 추가적인 필드 없이 식별하기 위하여, MR-트리에서는 데이터 엔트리들을 자식 노드를 위한 엔트리들 다음에 배치시킨다. 그러므로 데이터 객체들을 위한 첫 번째 엔트리 번호는 엔트리의 시작 번호를 0으로 할 때 노드안 엔트리 개수(# of entries) 필드에 해당한다.

검색 알고리즘은 R-트리[7]의 검색 알고리즘과 비슷하다. 단지 차이점이라면 반-리프 노드를 검색할 경우 질의 사각형을 자식 노드를 위한 MBR들 뿐만 아니라 데이터 객체를 위한 MBR들과도 비교를 한다는 점을 들 수 있다.

삽입 알고리즘에서는 루트에서 리프 노드까지 내려오면서 MR-트리는 각 내부 노드가 빈 엔트리를 지니는지 여부를 검사한다. 만약 그 중 하나 이상의 노드가 빈 엔트리를 지닌다면, 노드 분할 발생시 이를 상위로 전달한다. 만약 모든 내부 노드들의 엔트리가 가득 차있다면, 분할된 노드는 새롭게 생성된 노드와 데이터 객체들을 위한 엔트리를 지니는 반-리프 노드가 된다. MR-트리는 삽입 순서에 따라 트리의 높이가 커지고, 중간 노드들의 높이 차이가 증가할 수 있다. MR-트리는 R-트리와 같은 트리조정(AdjustTree) 알고리즘[7]을 사용하는데 만약 MR-트리 내에서 어떤 내부 노드의 서브 트리들 간의 높이차이가 2가 되는 서브 트리를 발견하면, 높이균형(HeightBalance) 알고리즘이 호출되고, 이는 중간 노드의 서브 트리들 간의 높이 차이가 최대 1이 되도록 MR-트리를 균형화 시킨다.

(그림 10)은 MR-트리의 균형화 과정을 두 가지의 예제를 통해 보여준다. 우선 (그림 10)(a)는 높이 균형화 작업 전의 MR-트리를 보여준다. 여기서 P의 자식 노드들의 높이 차이가 2가 되기 때문에 MR-트리는 불균형 상태를 감지한다. (그림 10)(b)와 (그림 10)(c)는 (그림 10)(a)로부터 균형화 과정을 수행한 다음, 가능한 변경된 트리 구조들이다. (그림 10)(b)의 경우에는 트리 불균형을 유발시킨 노드 C_1과 높이가 h인 P의 자식 노드들중 하나(N_M)를 교환하는 방법이며, (그림 10)(c)의 경우에는 높이가 h인 P의 자식 노드 2개를 자식으로 갖는 새로운 노드(N_{NEW})를 생성하고, 이로 인해 생긴 P의 빈 엔트리에 C_1을 채워 넣는 방법에 해당한다.

```
Algorithm HeightBalance(P, N)
  // P는 불균형 트리 노드, N은 높이가 P.height-1인 P의 자식 노드 //
  H1. EC ←<entry in N,where E.child -> Height + 1 == N.height> ;
  H2. Create a temp entry ET ; Copy EC to ET Remove EC from N ;
  H3. N.height-- ; P.height-- ;
  H4. for (each pair of entries <Ei, Ej>
            from the entries in P including ET){
```



(a) 높이 균형화전의 MR-트리

(b) 높이 균형화 수행후의 가능한 MR-트리 구조

(c) 높이 균형화 수행후의 가능한 다른 MR-트리 구조

(그림 10) MR-트리 균형화 과정 예제

```
    NEi ← Ei.child ; NEj ← Ej.child ;
    if (NEi.height == P.height - 2 &&
          (NEj.height == P.height - 2 !! NEj has empty rooms)) {
          Make a rectangle J including Ei.MBR and Ej.MBR ;
          Calculate dij = area(J) - area(Ei.MBR) - area(Ej.MBR) ;
    }
  }
H5. Choose the pair <Ei, Ej> with the smallest dij
H6. if (both NEi and NEj have same Heights) {    // P.height-2
    Create a node NNew          // fig. 3(c)
    Add entries for NEi and NEj to NNew ;
    Create an entry ENew for NNew ;
    Remove the entry Ej from P ; Add ENew to P ;
H7. } else {                    // NEi.height == NEj.height-1
H8. Add Ei to NEj               // fig. 3(b)
    Enlarge Ej.MBR so that it encloses Ei.MBR ;
    }
H9. Remove Ei from P ; Add ET to P ;
```

MR-트리의 삭제 알고리즘은 R-트리 계열과 같은 알고리즘을 사용하지만, 재 삽입 수를 줄이기 위해 R-트리의 트리압축(CondenseTree)[7]과 다른 방식을 사용한다. (그림 10)은 그 예를 보여준다.

만약 R-트리의 트리압축 알고리즘이 (그림 11)(a)와 같은 상황을 만난다면, 노드 C는 단순히 Q로 삽입되고, 마지막 단계에서 노드 D1과 D2는 재 삽입 과정을 거치게 된다. 그러나 MR-트리의 트리압축 알고리즘에서는 우선 노드 P안에 빈 엔트리가 있는지 여부를 검사한다. 만약 빈 엔트리가 있다면, 다시 노드 C의 자식 노드들에 대한 높이를 조사한다. (그림 11)(a)에서 D1의 높이는 h-1이고 D2의 높이는 h이기 때문에, 노드 D2는 재삽입 없이 P의 자식 노드로 연결이 가능하다. 이는 P와 D2의 높이 차이가 2에 불과하기 때문이다. 그러므로 트리압축 알고리즘은 (그림 11)(b)에서와 같이 노드 D2를 P의 자식 노드로 지정하고, 단지 D1만을 Q로 삽입한다. 이와 같은 방법으로 MR-트리는 R-트리에 비해 재 삽입되어야 할 엔트리의 수를 최소화함으로써 삭제 시간을 줄인다.

4. 실 험

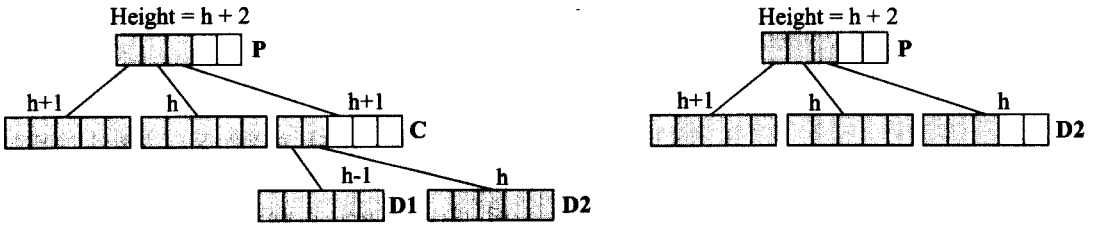
본 장에서는 시스템의 성능 평가를 위해서 새롭게 제시하는 인덱스 구조와 기존의 인덱스 구조들을 구현하여 검색 및 갱신 성능을 비교한다. 이를 위해서 본 실험에서는 R-트리, PE R-트리[1], CR-트리, PE CR-트리[1] 및 새롭게 제시하는 MR-트리, CCMR-트리를 2차원 인덱스 구조로 구현 하였다. CCMR(Cache Conscious MR)-트리는 CR-트리에서 사용하는 QRMBR 기술을 리프 노드에만 적용시키는 구조로써, 만약 참조 MBR의 변경으로 인해 리프 노드의 QRMBR들이 변경될 경우, QRMBR은 데이터베이스에 저장된 객체 MBR를 이용해 재계산 된다. 이 모든 인덱스 구조들은 GNU의 gcc로 컴파일 되도록 하였으며 하드웨어 환경은 1.6GHz의 CPU와 256K의 L2 캐쉬, 128바이트의 캐쉬 블록 크기를 사용하는 펜티엄 4 컴퓨터를 사용하였다.

최적의 노드 크기를 찾기 위해, 실험에서는 노드의 크기는 128바이트에서 1,024바이트까지 변경시켰다. 한편, R-트리와 MR-트리를 위한 MBR의 크기는 16바이트를 사용했으며, CR-트리와 PE CR-트리를 위해서는 QRMBR의 크기를 8바이트로 사용했다. 이는 앞의 (그림 5)에서 설명한 것과 같이 너무 작은 QRMBR의 크기는 오히려 성능 저하를 나타냄을 보였기 때문이다. 그러나 CCMR-트리에 대해서는 CR-트리에서 발생 가능한 어떤 이상 현상도 발생하지 않기 때문에 4바이트의 QRMBR을 사용한다. 본 실험에서는 이 모든 인덱스 구조들에 원래의 R-트리에서 사용했던 선형-비용(linear-cost) 분할 알고리즘을 적용 시켰다.

본 실험에서는 [1]과 같이 2개의 데이터 집단을 인위적으로 생성 시켰다. 이는 단위 정사각형 안에 위치한 백만 개의 작은 사각형으로 구성된다. 첫 번째 데이터 집단은 단위 정사각형 안에서 균등 분포를 나타내며, 두 번째는 중심점 (0.5, 0.5)와 표준편차 0.25를 갖는 가우스 분포를 나타낸다. 각각의 데이터 사각형의 평균 측면 길이는 0.001로 지정하였다. 한편, 모든 인덱스 구조의 성능이 입력 순서에 영향을 받기 때문에, 본 실험에서는 데이터 집단들안 데이터 객체 MBR의 입력 순서를 난수 함수를 사용해 설정했다.

4.1 검색 성능

본 실험에서는 여러 인덱스 구조들에 대한 검색 성능을 비교하기 위해, 2차원의 사각형 검색 영역에 대한 처리 시간을 비교했다. 이를 위해 본 실험에서는 가우스 분포와 균

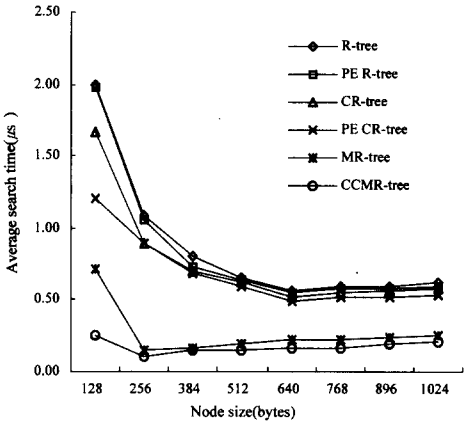


(a) 삭제 실행전 MR-tree

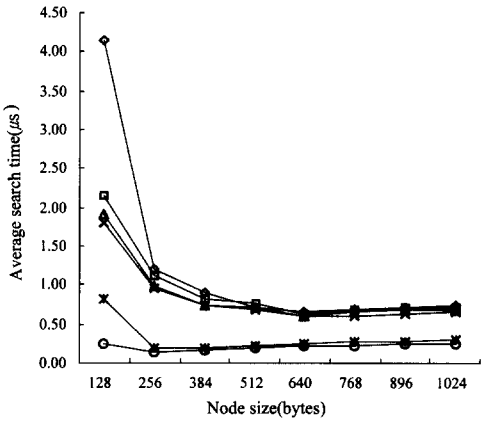
(b) 재삽입전의 MR-tree
(D1만이 재삽입 Q에 추가됨)

(그림 11) MR-트리 삭제시 트리 압축과정 예제

등 분포를 갖는 각각의 10,000개의 서로 다른 질의 사각형을 생성했다. 각각의 데이터 집합을 위한 질의 사각형은 단위 정사각형의 0.01%에서 1%로 다양화 하였다.



(a) 검색 사각형의 크기 = 0.01%



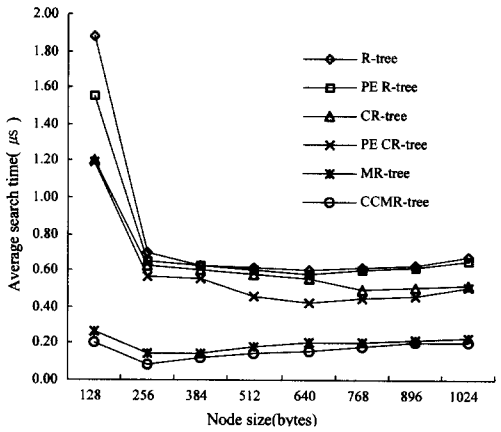
(b) 검색 사각형의 크기 = 0.1%

(그림 12) 검색 성능 비교(평균 〈0.5, 0.5〉, 표준편차 0.25의 가우스 분포)

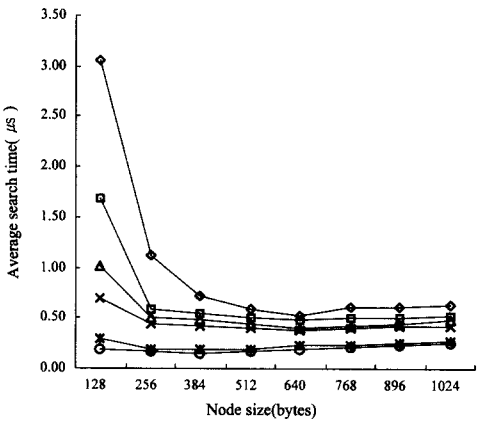
(그림 12)은 가우스 분포를 갖는 데이터 집단에 대해 여러 비교 인덱스들의 평균 검색 시간을 나타내며, (그림 13)는 균등 분포를 갖는 데이터 집단에 대한 평균 검색 시간을 나타낸다. 이 그래프에 대해 다음과 같은 관찰 결과를 얻을 수 있다.

- 노드의 크기가 커지면서, 검색시간은 빠르게 최소 상태에 도달하며, 다시 천천히 증가함을 알 수 있다. 이러한 경향은 검색 사각형의 크기를 크게 했을 경우에도 같은 결과를 보여준다.
- MR-트리와 CCMR-트리는 R-트리, PE R-트리, CR-트리, PE CR-트리와 비교해 더 빠른 검색 성능을 보여준다. 특히, 본 논문에서 제시하는 두 기법은 데이터 집단이 가우스 분포를 가질 경우와 검색 사각형이 작을수록 더 좋은 검색 성능을 보여 준다.

본 논문에서는 이밖에도 여러 불균형한 분포를 갖는 데이터 집단들에 대해 같은 실험을 수행했지만, (그림 12)과 (그림 13)로부터 확연하게 구별되는 특별한 차이점을 찾지 못했다.



(a) 검색 사각형의 크기 = 0.01%



(b) 검색 사각형의 크기 = 0.1%

(그림 13) 검색 성능 비교 (균등 분포)

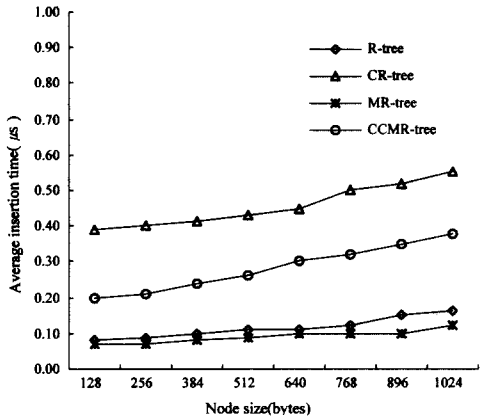
4.2 갱신 성능

갱신 성능을 비교하기 위해, 본 실험에서는 백만 개의 균등분포 데이터 집단을 로딩한 후, 10,000개의 데이터 객체를 삽입하고, 전체 데이터 집단으로부터 임의로 선정한 10,000개의 데이터 객체를 삭제했다.

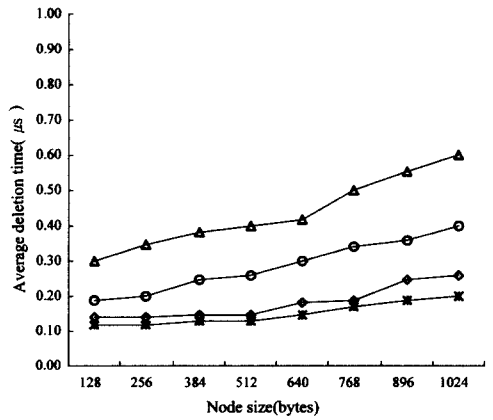
(그림 14)(a)와 (그림 14)(b)는 각각 삽입 및 삭제에 대한 평균 처리 시간을 나타낸다. 인덱스들의 삽입시간에서는 R-트리와 MR-트리에 비해 CR-트리와 CCMR-트리가 더 나쁜 성능을 보여준다. 이것은 노드 분할이 발생하거나, 참조 MBR의 변경에 있어서 CR-트리와 CCMR-트리은 QRMBR에 대한 재 계산 과정이 필요하기 때문이다. 한편, MR-트리와 CCMR-트리는 R-트리 및 CR-트리와의 비교에 있어서 각각 더 좋은 성능을 나타낸다. 이는 다음과 같이 설명된다. MR-트리와 CCMR-트리의 경우 삽입 경로상의 중간

노드에 빈 엔트리가 있을 경우에만 리프 노드의 분할이 상위 노드로 전달된다. 그러므로 MR-트리와 CCMR-트리는 전체 노드 분할 수를 확연히 줄일 수 있다. 그러나 MR-트리와 CCMR-트리는 삽입 성능에 악영향을 미는 높이 균형화 과정이 발생할 수 있다. 본 실험에서는 두개의 데이터 집단에 대해 삽입시의 높이 균형화 발생수를 측정하였고, 그 결과 이는 노드 분할 수에 비해 극히 미미했기 때문에 전체 삽입 성능에 별다른 영향을 미치지 않는다는 점을 발견했다. 이러한 점은 불균등한 데이터 집단에 대한 실험에서도 같은 결과를 얻었고, 최악의 경우, 높이 균등화 발생은 R-트리의 중간 노드 분할과 비슷한 수로 발생되었다.

인덱스 삭제의 경우 역시, CR-트리와 CCMR-트리가 R-트리와 MR-트리에 비해 더 나쁜 성능을 나타낸다. 이 이유는 삽입의 경우와 같은 방식으로 설명할 수 있다. MR-트리의 경우 R-트리와 비교해서, 서로 다른 트리압축(CondenseTree) 알고리즘의 사용으로 인해 재 삽입되어야 하는 노드의 수를 확연히 줄인다. 그러므로 MR-트리와 CCMR-트리는 R-트리와 CR-트리에 비해 각각 더 좋은 성능을 나타낸다.



(a) 10,000 데이터에 대한 삽입시간



(b) 10,000 데이터에 대한 삭제시간

(그림 14) 갱신 성능 비교(균등 데이터 분포)

5. 결 론

본 논문에서는 실시간 모바일 GIS 응용을 위한 주기억장치 데이터베이스 시스템을 설계 및 구현하였다. 본 시스템은 크게 유/무선 클라이언트와의 통신을 위한 인터페이스 관리기와 공간 및 비공간 질의 처리를 위한 질의 처리기, 벡터 데이터 및 비공간 데이터, MR-트리와 T-트리 인덱스 구조들을 페이지단위로 관리하는 주기억 데이터 관리기, 그리고 본 논문에서 새롭게 제시하는 공간 질의 처리를 위한 MR-트리와 비공간 질의 처리를 위한 T-트리를 관리하는 인덱스 관리기로 구성된다.

새로 제안된 MR-트리에서는 삽입 경로상의 중간 노드들 중 하나 이상이 빈 엔트리를 지닐 때만, 노드 분할을 상위 노드로 전달한다. 만약 빈 엔트리가 전혀 없다면, 새롭게 생성된 리프 노드는 단지 분할된 노드의 자식 노드가 된다. 이러한 노드는 데이터 객체를 위한 엔트리들뿐만 아니라 자식 노드들을 위한 엔트리들 역시 지니고 있기 때문에 반-리프 노드라 부른다. MR-트리의 높이는 데이터의 삽입 순서에 영향을 받는다. 그러므로 어떤 중간 노드의 자식 노드들의 불균형을 발견할 경우, 높이 균형화 알고리즘이 실행되어 자식 노드들의 높이를 균형화 시킨다.

본 논문의 실험에서는 MR-트리가 R-트리와 CR-트리에 비해 검색 시간에서 2.4배 이상의 좋은 성능을 나타냈다. MR-트리는 검색 사각형의 크기가 작을수록, 그리고 데이터 집단이 가우스 분포를 형성할 때 더 좋은 성능을 나타냈다. 수정 성능 측면에서 역시 MR-트리가 더 좋은 성능을 나타냈다. 향후 연구과제로 구현한 본 시스템에 실제 대량의 지리 정보 자료를 입력하여 그 성능을 분석하고, 대량의 지리 정보 자료에 대한 효율적인 압축 기법을 연구하며 인덱스 구조 및 각 관리기를 최적화 시킬 계획이다.

참 고 문 헌

[1] K. Kim, S. K. and Cha, K. Kwon, "Optimizing Multidimensional Index Trees for Main Memory Access," *Proceedings of ACM SIGMOD Conference*, pp.139-150, 2001.

[2] Trishul M. Chilimbi, Mark D. Hill and James R. Larus, "Making Pointer-Based Data Structures Cache Conscious," *IEEE Computer*, TBD, pp.67-74, 2000.

[3] P. Bones, S. Manegold and M. Kersten, "Database Architecture Opimized for the New Bottleneck : Memory Access," *Proceedings of VLDB Conference*, pp.54-65, 1999.

[4] A. Ailamaki, D. J. DeWitt, M. D. Hill and D. A. Wood, "DBMSs on a Modern Processor : Where Does Time Go?," *Proceedings of VLDB Conference*, pp.267-277, 1999.

[5] J. Rao, K. A. Ross, "Cache Conscious Indexing for Decision-support in Main Memory," *Proceedings of VLDB Conference*, pp.78-89, 1999.

[6] J. Rao, K. A. Ross, "Making B+-trees Cache Conscious in Main Memory," *Proceedings of ACM SIGMOD Conference*, pp.475-486, 2000.

[7] A. Guttman, "R-tree : A Dynamic Index Structure for Spatial Searching," *Proceedings of ACM SIGMOD Conference*, pp.47-57, 1984.

[8] Phi Bernstein, et al., "The Asilomar report on database research," Sigmod Record, 27(4), 1998.

[9] Bohannon, P., D. Lieuwen, R. Rastogi, A. Silberschatz and S. Sudarshan, "The Architecture of the Dali Main Memory Storage Manager," *The International Journal on Multimedia Tools and pplications*, 4, 2, March, 1997.

[10] H. V. Jagadish, Daniel Lieuwen, Rajeev Rastogi, Avi Silberschatz, "Dali : A High Performance Main Memory Storage Manager," Proc. of the 20th conf. on VLDB, Santiago, Chile, 1994.

[11] L. Felician and A. Gentili, A nearly optimal Huffman technique in the microcomputer environment, *Inf. Sys. 12*, 4, 37, 1987.

[12] R. G. Gallager, Variations on a Theme by Huffman, *IEEE Trans. on Inf. Theory IT-24*, 6, 66, 1978.

[13] Tobin J. Lehman, Eugene J. Shekita, Luis-Felipe Carera "An Evaluation of Starburst's Memory Resident Storage Componet," *IEEE Trans. on Knowlege and Data Engineering*, Vol.4, December, 1992.

[14] Tobin J. Lehman, Michael J. Carey, "A Study of Index Structures for Main Memory Database Management Systems," *Procedings 12th Int'l. conf. on Very Large Databases*, kyoto, pp.294-303, Aug., 1986.

강 은 호

e-mail : ehkang@cs.hongik.ac.kr

2002년 홍익대학교 컴퓨터공학과(학사)

2002년~현재 홍익대학교 컴퓨터공학과 재학(석사)

관심분야 : 실시간 데이터베이스, 지리정보 시스템, 모바일 데이터베이스

윤 석 우

e-mail : sw0305@hanmail.net

1994년 홍익대학교 컴퓨터공학과(학사)

1996년 홍익대학교 전자계산학과(석사)

2002년~현재 홍익대학교 컴퓨터공학과 재학(박사)

관심분야 : 실시간 데이터베이스, 지리정보 시스템, 모바일 데이터베이스

김 경 창

e-mail : kckim@cs.hongik.ac.kr

1978년 홍익대학교 전자계산학과(학사)

1980년 한국과학기술원 전산학과(석사)

1990년 University of Texas at Austin 전산학과(박사)

1991년~현재 홍익대학교 정보컴퓨터공학부 교수

관심분야 : 객체지향 데이터베이스, 주기억 데이터베이스, OLAP 및 데이터 웨어하우징, Web 데이터베이스