이동체 데이타베이스를 위한 통합 색인의 설계 및 구현

(Design and Implementation of Unified Index for Moving Objects Databases)

박 재 관 † 안 경 환 †† 정 지 원 ††† 홍 봉 희 ††††
(Jaekwan Park) (Kyunghwan An) (Jiwon Jung) (Bonghee Hong)

요 약 최근 PDA, 휴대폰, 노트북, GPS, RFID와 같은 모바일 장치의 발달과 범용적인 사용으로 위치 기반 서비스(LBS: Location Based Service)에 대한 요구가 점점 증대되고 있다. 위치 기반 서비스의 핵심 기술로는 이동체의 위치를 저장 및 관리하기 위한 이동체 데이타베이스를 들 수 있다. 이러한 데이타베이스는 이동체 정보를 빠르게 검색하기 위해 색인을 필요로 하며, 이 색인은 다수의 이동체에 의해 갱신되는 업데이트를 관리하고 실시간으로 위치를 추적할 수 있어야 한다. 따라서 이동체 데이타베이스를 위한 색인은 실시간 처리를 위해서 메인 메모리에서 동작하는 색인의 구조를 가져야 하며, 다수 이동체의 위치 정보를 관리하기 위해 색인의 일부분을 메모리에서 디스크로 이동하거나 디스크에서 메모리로 로딩하는 기법을 지원해야 한다.

이 논문에서는 이러한 색인의 요구 조건을 충족시키기 위해서 메인 메모리와 디스크를 연동하는 통합 색인 기법과 메모리 공간 부족 시에 색인의 일부를 디스크로 이동시키는 이주 정책들을 제시하였다. 이주 정책은 디스크 I/O를 줄이기 위해 노드 단위가 아닌 서브트리 단위로 이동하도록 함으로써, 벌크 연산 및 동적 클러스터링의 효과를 얻게 된다. 통합 색인은 이주 정책에 따라 다른 형태로 구성될 수 있으며, 본 논문에서는 Oldest Node 정책과 LRU Buffer 정책을 적용하였다. 또한 통합 색인을 구현하고, 각 이주 정책 별로 실험 평가를 수행하여 성능을 측정하였다.

키워드 : 지리정보시스템, 이동체, 공간 색인

Abstract Recently, the need for Location-Based Service (LBS) has increased due to the development and widespread use of the mobile devices (e.g., PDAs, cellular phones, labtop computers, GPS, and RFID etc). The core technology of LBS is a moving-objects database that stores and manages the positions of moving objects. To search for information quickly, the database needs to contain an index that supports both real-time position tracking and management of large numbers of updates. As a result, the index requires a structure operating in the main memory for real-time processing and requires a technique to migrate part of the index from the main memory to disk storage (or from disk storage to the main memory) to manage large volumes of data.

To satisfy these requirements, this paper suggests a unified index scheme unifying the main memory and the disk as well as migration policies for migrating part of the index from the memory to the disk during a restriction in memory space. Migration policy determines a group of nodes, called the migration subtree, and migrates the group as a unit to reduce disk I/O. This method takes advantage of bulk operations and dynamic clustering. The unified index is created by applying various migration policies. This paper measures and compares the performance of the migration policies using experimental evaluation.

Key words : GIS, Moving Objects, Spatial Index

· 본 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT기술연구사업단)의 지원에 의하여 연구되었음
† 학생회원 : 부산대학교 컴퓨터공학과
jkpack@pusan.ac.kr
†† 정 회 원 : 한국전자통신연구원 텔레매틱스 USN연구단 연구원
mobileguru@etri.re.kr
††† 정 회 원 : 삼성전자 정보통신총괄 무선사업부 개발6그룹 사원
jiwon76.jung@samsung.com
†††† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@pusan.ac.kr
논문접수 : 2005년 7월 11일
심사완료 : 2006년 2월 22일

1. 서 론

최근 PDA, 휴대폰 등과 같은 무선 이동 통신 환경의 발달 및 GPS, RFID와 같은 위치정보 보고 장치가 발달함에 따라 위치 기반 서비스(LBS: Location Based Service)에 대한 요구가 점점 증대되고 있다. 향후에 최근 각광받고 있는 유비쿼터스(ubiquitous) 환경이 본격적으로 도래할 경우, 위치를 기반으로 한 다양한 응용 서비스 개발 및 위치 기반 서비스 플랫폼의 개발이 더욱 필요하게 될 것이다.

이러한 위치기반 서비스에서 차량, 비행기, 선박과 같이 시간에 따라 위치가 변하는 객체를 이동체라 하고, 이들의 위치를 저장하고 관리할 수 있는 시스템을 이동체 데이타베이스 시스템이라 한다. 이동체 데이타베이스의 경우 기존의 데이타베이스와는 다른 다음과 같은 특징을 가진다. 첫째, 이동체의 위치 보고 데이타에 대한 실시간 갱신을 필요로 한다. 많은 이동체들이 주기 또는 비주기적으로 계속해서 위치 보고를 하므로, 주어진 시간 내에 위치 데이타를 데이타베이스 내에 저장해야 한다. 그러나 이동체의 위치에 대한 질의의 경우는 실시간 응답을 필요로 하지 않는다. 둘째, 위치 보고 데이타의 양이 방대하다. 시간이 지남에 따라 계속해서 위치 데이타가 보고 되므로 누적되는 데이타의 양이 점점 많아진다.

이동체 데이타베이스에서 이동체에 대한 현재 위치 및 과거 궤적 정보를 빠르게 검색하기 위해서는 색인을 필요로 하는데 이동체의 특징을 반영하기 위해서는 다음 사항들을 고려한 설계가 필요하다. 첫째, 메인 메모리 기반 색인의 사용이 필요하다. 위치 데이타에 대한 실시간 갱신을 위해서는 예측 가능한 시스템의 동작을 필요로 하는데 기존의 디스크 기반 색인의 경우 캐쉬를 사용하면 교체 전략에 따라 적중(hit) 또는 비적중(miss)될 수 있으므로 성능을 예측하기 어렵다는 문제점이 있다. 둘째, 방대한 위치 데이타에 대한 처리가 필요하다. 계속해서 누적되는 위치 데이타의 경우 모든 데이타를 메모리에 저장하기는 힘들다. 그러므로 검색 가능성이 낮은 과거 궤적 데이타들은 디스크로 이동하는 것이 필요하다.

이 논문에서는 이동체 데이타베이스 색인의 요구 조건을 만족시키기 위한 통합 색인(unified index) 기법을 제시한다(그림 1). 제시하는 통합 색인 기법은 메인 메모리 기반 색인으로 색인의 크기가 주어진 메인 메모리의 용량을 넘어서게 되면 자동으로 색인의 일부를 디스크로 이동시켜 주게 된다. 또한 메모리에서 디스크로의 노드 이동 시에 발생할 수 있는 디스크 I/O를 최소화할 수 있는 방법을 제시한다.

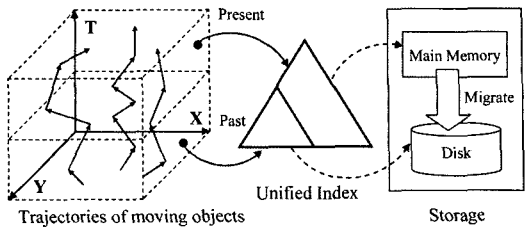


그림 1 통합 색인

통합 색인은 최근 보고된 위치 데이타에 대한 색인 부분이 메모리에 존재하기 때문에 빠른 갱신 속도를 지원할 수 있고, 디스크로의 이동을 지원하기 때문에 계속해서 증가하는 위치 데이타에 대한 저장 문제를 해결할 수 있다. 즉 통합 색인은 메인 메모리 기반 색인의 빠른 갱신 및 검색 속도와 디스크 기반 색인의 방대한 저장 공간의 장점을 모두 가지고 있다.

이 논문의 구성은 다음과 같다. 2장에서 관련 연구에 대해서 기술하고, 3장에서는 통합 색인을 설계할 때 발생할 수 있는 문제점에 대해서 기술한다. 4장에서는 통합 색인의 구조에 대해서 설명하고, 5장에서는 통합 색인의 알고리즘을, 6장에서는 구현 및 실험을 통하여 제안된 통합 색인의 이주 정책들에 대한 성능비교 평가를 보인다. 마지막으로 7장에서 결론을 맺는다.

2. 관련 연구

이 장에서는 관련 연구로서 먼저 디스크 및 메인 메모리 기반 색인에 관한 기존 연구와 이주와 관련된 연구를 설명한다. 디스크 기반의 다차원 색인으로는 R-tree[1]계열을 소개하고, 메인 메모리 기반 색인은 널리 알려진 T-tree[2]를 소개한다. 다음으로 데이타의 양 및 사용 빈도, 제한 시간 내 처리 여부에 따라 데이타를 다른 매체에 분리 저장하는 구조에 관한 연구[3], 데이타의 이동에 관한 연구[4]를 설명한다.

2.1 디스크 및 메인 메모리 기반 색인에 관한 연구

디스크 기반의 대표적인 색인인 R-tree[1]는 저장되는 객체를 최소 경계 박스(MBB: Minimum Bounding Box)로 표현하며 B-tree에 대해서 k차원으로 확장한 모델이다. 다차원 데이타에 대하여 높은 성능을 나타내며 디스크에 적합한 구조로 알려져 있다. 그러나 이동체 데이타베이스 환경에서 위치 데이타에 대한 실시간 갱신을 위해서는 예측 가능한 시스템의 동작을 필요로 하는데 디스크 기반 색인의 경우 캐쉬를 사용하면 교체 전략에 따라 적중(hit) 또는 비적중(miss)될 수 있으므로 성능을 예측하기 어렵다는 문제점이 있다. 또한 많은 이동체가 위치를 보고하는 경우에 디스크 병목 현상이 발생하여 시스템 성능이 시간이 지날수록 저하되는 문

제점을 가지고 있다. 이동체 데이타베이스는 일반적으로 일정한 주기를 가지고 입력되므로, 이동체의 다음 위치 정보가 보고되기 전에 현재 위치 정보의 갱신이 완료되어야 한다. 그러나 이동체 위치 정보는 궤적정보를 표현하는데 사용되므로 주어진 시간 내에 현재 위치의 갱신이 완료되는 것이 보장되어야 한다. 이것은 데이타베이스가 가지는 단위 시간의 처리능력(throughput)의 문제가 아닌 관리 가능한 이동체의 수와 직접적으로 연관되므로, 반드시 주어진 제약시간 내에 변경 트랜잭션 처리가 완료되어야 한다. 즉 실시간 환경에서 많은 이동체의 위치 보고 데이타를 저장해야 하는 이동체 데이타베이스에서 디스크 기반 색인을 사용하는 것은 부적절하다고 할 수 있다.

메인 메모리 기반의 대표적인 색인으로는 1차원 데이타를 위한 T-tree[2]가 있다. T-tree는 기존의 B-tree와 AVL-tree로부터 진화되어 나온 색인으로 대표적인 메인 메모리 색인 구조이다. T-tree는 이진 검색과 높이 균형을 가지는 AVL-tree의 성질을 가지고 있고, 한 노드 안에 여러 개의 데이타를 가지는 B-tree의 성질을 가지고 있다. 이러한 성질로 인하여 빠른 처리 속도와 메모리 사용의 최적화라는 메인 메모리의 특성에 적합한 구조로 알려져 있다. 그러나 이동체 데이타베이스 환경에서 메인 메모리 기반 색인은 이동체의 궤적 정보가 지속적으로 늘어남으로 저장 공간의 한계를 가지고 있다. 그러므로 검색 가능성이 낮은 과거 궤적 데이타들은 저비용, 고용량의 특징을 가진 디스크로 이동하는 것이 필요하다.

2.2 이주(migration)에 관한 연구

데이타를 다른 매체에 분리하여 저장하는 연구로는 Gawlick et al.[3]이 있다. 이는 자주 사용되는 데이타는 메모리 기반 데이타베이스에 저장하고 나머지 데이타는 디스크 기반 데이타베이스에 저장하는 방법을 사용한다. 그러나 한 가지 종류의 데이타가 시간이 지남에 따라서 양이 많아 질 경우, 데이타를 분류하여 저장해야 하기 때문에 자동적으로 저장구조가 상이한 매체간 이동이 지원되지 않는 문제점을 가지고 있다.

데이타의 이동에 관한 연구로는 Time-Split B-tree [4]가 있다. 이 연구는 현재 데이타는 디스크에 저장하고 과거 이력 데이타는 광학 저장 장치로 이동하는 방법에 관한 연구이다. 즉, 한 개의 색인 구조로 현재 데이타와 과거 이력 데이타를 동시에 다룰 수 있다. 그러나 Time-Split B-tree에서는 시간 축으로 노드가 분할될 때 한 번에 하나의 노드씩 광학 저장장치로 이동되기 때문에 이 논문에서 요구하는 디스크 I/O를 줄이기 위한 방법에 적용하기 어려우며, 노드에 중복된 데이타가 존재함으로 인해서 메모리 및 디스크에 적용했을 때 공간을 효율적으로 사용하지 못하는 단점이 있다.

3. 문제 정의

이 논문에서는 이동체가 주기적으로 서버에 위치 정보를 보고하며 이동체 또는 별도 클라이언트는 비주기적으로 질의를 서버에 요청하는 환경을 가정한다. 보고된 이동체의 위치 정보는 서버의 메인 메모리 색인에 저장하고, 메모리 공간이 부족할 경우 과거 이력 데이타는 디스크로 옮겨져야 한다. 이 논문에서는 이것을 색인의 '이주(migration)'로 정의한다. 그러나 기존에 연구된 이동체 색인들은 색인의 이주를 위한 방법론을 제시하지 않고 있다. 따라서 본 논문에서는 최근 이동체 위치 정보는 메인 메모리 색인에 저장하며, 질의에 의한 엑세스의 빈도가 적은 이동체의 과거 이력 정보는 디스크로 이주하여 저장하는 색인을 제안한다. 이 논문에서 제안하는 색인은, 하나의 색인이 메모리와 디스크에 분리 저장 및 관리되는 색인으로써, 통합 색인(Unified Index)이라고 정의한다. 이 장에서는 통합 색인을 설계하기 위해 고려해야 할 사항을 기술한다.

3.1 이주 대상 노드 선정 시 고려사항

통합 색인은 메모리 공간이 부족할 때 색인의 일부를 메모리에서 디스크로의 이주를 지원해야 한다. 이 때, 디스크로 이동할 노드 선택은 데이타 삽입, 질의 등에 영향을 미친다.

3.1.1 노드의 포인터

통합 색인에는 메모리 및 디스크 양쪽에 노드가 존재할 수 있으므로 메모리 포인터 및 디스크 포인터가 모두 존재할 수 있다. 이들을 포인터의 방향에 따라서 분류해 보면 다음과 같다(표 1).

표 1 노드 내의 포인터 종류

포인터의 방향	포인터 이름
memory node points to memory node	m2m 포인터
memory node points to disk node	m2d 포인터
disk node points to memory node	d2m 포인터
disk node points to disk node	d2d 포인터

m2m 포인터는 메모리 노드가 다른 메모리 노드를 가리키는 경우로 순수 메모리 기반의 색인과 같이 메모리 주소를 가지게 된다. d2d 포인터의 경우는 디스크 기반 색인의 포인터와 같은 종류로 디스크 노드가 디스크 노드를 가리킬 때를 나타내는 포인터이다. m2d와 d2m 포인터의 경우 기존의 색인 기법에서는 존재하지 않는 포인터로써 각기 다른 타입의 노드를 가리키는 포인터이다.

3.1.2 d2m 포인터의 문제점

그림 2는 통합색인에서 나타날 수 있는 일반적인 구조를 보여준다. 회색의 노드는 디스크로이동된 노드들이며, 흰색의 노드는 메모리에 존재하는 노드이다. 이때 디스크 노드 A의 자식 노드들 중에 노드 B처럼 메모리에 존재하는 노드를 가리키는 경우가 존재할 수 있으며, 이것을 d2m 포인터라고 한다. 이러한 d2m 포인터는 두 가지 성능상의 문제를 야기시킨다. 첫째, 노드 B가 디스크로 이동되어야 하는 경우 부모 노드의 포인터 갱신을 위해서 노드 A를 다시 메모리로 읽어 들여야 한다. 둘째, 노드 B가 질의에 의해 참조되어야 한다면 노드 A가 먼저 메모리로 읽혀져야 한다. 즉, 메모리 노드를 접근하기 위해서 Disk I/O를 발생시키는 것은 바람직하지 못하다. 이를 방지하기 위해서는 이주 대상 노드의 선정 정책에서 d2m 포인터가 발생하지 않도록 고려되어야 한다.

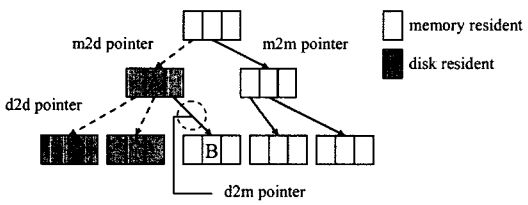


그림 2 d2m 포인터의 문제점

3.2 이주 노드의 디스크 배치

디스크 배치(disk placement)는 통합 색인의 질의 성능에 영향을 미친다. 메모리 공간 부족으로 노드의 이동이 일어날 때, 디스크로 이동할 노드들이 어떤 순서로 디스크에 저장되는 가에 따라 질의 성능이 결정된다. 그림 3은 R-tree[1]가 생성되는 과정에서 각 노드들이 디스크 상에 어떠한 순서로 배치되는 것인가를 나타낸 것이다. R-tree 계열의 색인이 구성될 때 그림 3과 같이 노드가 생성된 순서대로 페이지들이 디스크 상에 할당되므로 논리적으로 근접한 노드들이 디스크상에 물리적으로 근접해서 저장되지는 않는다.

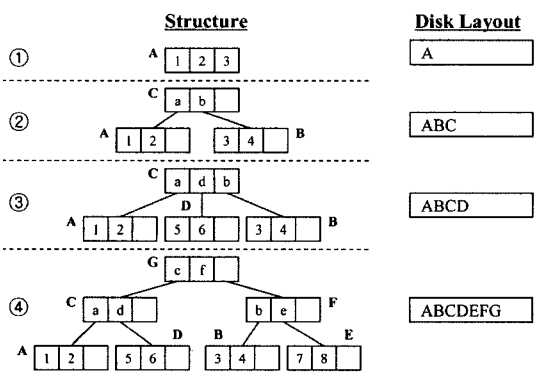


그림 3 R-tree에서의 디스크 배치

이러한 디스크 배치는 통합 색인에서 질의가 수행되면서 디스크에 저장된 노드를 엑세스 할 경우, 메모리로 로딩하기 위한 Disk I/O의 증가를 유발시킨다. 따라서 메모리에서 디스크로 이주될 때 이주 대상 노드들이 물리적으로 근접한 곳에 저장하기 위한 디스크 배치 정책이 적용될 필요가 있다.

3.3 노드 이주 시점 및 방법

통합 색인에서 노드의 이주가 발생하는 시점은 메모리 공간의 부족으로 인하여 새로운 위치 정보를 저장할 노드를 생성할 수 없을 때이다. 이것은 식 (1)로 정의할 수 있다.

$$\sum_{n=0}^{h-1}(F\times U)^n > \frac{M}{S_p} \quad (1)$$

수식 1에서 첫 번째 항의 h는 통합 색인의 깊이를 나타내고, F는 노드의 Fanout, U는 노드의 Utilization을 의미한다. 두 번째 항에서 M은 메모리 공간의 크기를 의미하고 S_p는 페이지의 크기를 의미한다. 즉, 수식 1은 색인 노드 수가 메모리 공간상에 저장될 수 있는 노드 수보다 많을 때를 체크할 수 있는 근거를 제시하며, 이것은 통합 색인의 이주가 메모리에 새로운 노드를 만들 공간이 없을 때 이주가 발생함을 의미한다.

노드의 이주 시점이 되면, 하나의 노드씩 이동하기 보다는 향후 이주 대상 노드가 될 노드들을 묶어 하나의 그룹으로 이동하면 잦은 이주 발생을 줄일 수 있으며, 이로 인한 디스크 엑세스도 줄일 수 있다. 따라서 메모리에서 디스크로 이동될 때 이주 대상 노드를 복수 개로 선정하고 동적 클러스터링 방법을 도입하여 디스크 상에 위치 시키는 것이 필요하다. 이것은 질의에 의해 다시 메모리로 로딩될 때 이주에서 적용된 단위로 벌크 로딩으로 처리되어야 함을 암시한다. 이러한 요구사항은 기존 색인에서 발생하지 않지만, 메모리와 디스크의 통합적인 색인으로 구축될 때 요구되는 문제이다.

4. 통합 색인

이 논문에서는 이동체를 점으로 과거 궤적은 폴리라인으로 모델링한다. 통합 색인의 구조는 노드가 4가지 포인터 타입을 가지는 특징을 제외하면 R-tree의 구조와 유사하다. 통합 색인의 형태 및 성능은 어떤 노드를 메모리 또는 디스크에 저장할 것인지와 관련된 이주 정책에 따라 다르게 나타나게 된다. 따라서 이 논문에서는 노드의 이주 정책들과 색인의 구조적인 특징을 중심으로 설명한다. 제시되는 이주 정책은 R-tree기반의 여러 색인들에 적용될 수 있으므로, Original R-tree에 이주 정책을 적용한 통합 색인을 설명한다.

4.1 통합 색인의 이주 정책

4.1.1 이주 시점

통합 색인의 노드들은 메모리 또는 디스크 포인터를 가질 수 있다. 이동체에 대한 위치 데이타가 삽입될 때에는 메모리 노드에 저장되며 시간이 지남에 따라 노드의 분할에 의해 메모리 공간이 부족하게 될 경우, 통합 색인에서는 색인의 일부를 디스크로의 이동하는 것이 필요하다. 이 때 이주 시점은 다음 조건(식 (2))이 만족될 때 발생한다.

$$\sum_{n=0}^{h-1}(F\times U)^n \geq \frac{M}{S_p} - N_s \times \alpha \qquad (\alpha \geq 1) \tag{2}$$

식 (2)에서 첫 번째 항의 h는 통합 색인의 깊이를 나타내고, F는 노드의 Fanout, U는 노드의 Utilization을 의미한다. 두 번째 항에서 M은 메모리 공간의 크기를 의미하고 S_p는 페이지의 크기를, N_s는 이동 그룹의 노드 개수를 의미한다. 이 수식이 의미하는 바는 통합 색인의 노드 개수가 메모리 공간에 저장될 수 있는 노드의 개수에서 이동 그룹의 노드 개수의 배수(α : 1보다 큰 값)를 뺀 값보다 같거나 크게 될 때가 이주 시점임을 의미한다. 위 수식에서 이동 그룹의 배수를 빼는 이유는 이동 그룹을 디스크로 이동하는 동안 계속해서 삽입되는 위치 정보를 메모리에 저장하기 위해서이다.

4.1.2 이주 대상 노드 선정 정책

이 논문에서는 이주 시점에 디스크로 이동할 노드를 선정하기 위해 두 가지 정책, 생성된 시간이 오래된 궤적을 가진 노드를 대상으로 선정하는 정책과 질의 및 삽입 시에 참조되는 단말 노드들에 대한 LRU buffer를 이용하여 대상을 선정하는 정책을 제시한다.

4.1.2.1 Oldest Node 정책

통합 색인에서 디스크로 이동할 노드를 선정할 때 삽입 시에 참조될 가능성이 있는 노드들과 질의에 의해서 참조될 가능성이 높은 노드들은 선정에서 제외하면 좋은 성능을 얻을 수 있다. 이 논문에서는 이동체 궤적이 시간에 따라 참조될 확률이 결정된다고 가정한다. 즉, 현재와 가까운 궤적 정보일수록 더 참조될 가능성이 높고 오래된 궤적일수록 참조될 확률이 적음을 가정한다. Oldest Node 정책은 메모리에 존재하는 노드들 중에서 가장 오래된 위치 정보를 가진 노드를 이주의 대상으로 선정하는 방법이다. 통합 색인의 경우 노드의 최소 경계 사각형(Minimum Bounding Box, MBB)의 요소중 가장 적은 시간 값을 가지는 노드를 선정하며 단말 노드부터 우선 선택하여 점점 상위 레벨의 비단말 노드를 이주 대상을 확대한다.

R-tree 계열의 색인은 색인의 좌측에 시간적으로 오래된 궤적정보가 저장되고, 노드의 분할이 일어남에 따라 우측에 새롭게 생성된 노드들이 존재하게 된다. 이

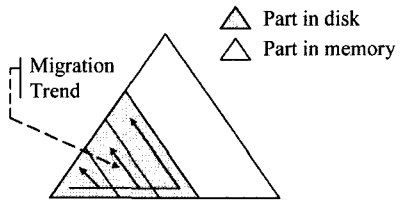


그림 4 Oldest Node 정책에서 디스크로의 이동 순서

때 이주 노드의 선정은 그림 4와 같이 색인의 좌하단에서 점진적으로 우상단 방향으로 발생한다.

4.1.2.2 LRU buffer 정책

시간적으로 가장 오래된 노드를 선정하는 정책은 최근에 새롭게 갱신이 이뤄진 부분에 질의가 많을 때 좋은 검색 성능을 보일 수 있다. 그러나 이동체 궤적에 대한 질의가 과거 궤적부터 현재 궤적까지 다양하게 이뤄진다고 할 때는 좋은 성능을 기대하기 어렵다. LRU 버퍼를 이용한 정책은 삽입 및 질의 시에 참조되는 노드들로 구성된 LRU 버퍼에서 가장 오래 전에 참조된 노드를 이주 대상으로 선정하는 방식이다.

그림 5에서 보는 바와 같이, LRU buffer를 이용한 정책의 노드 선정 방향은 buffer의 정렬 순서에 따라 결정된다.

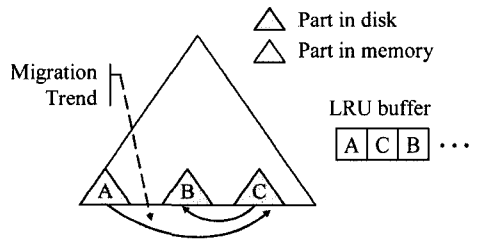


그림 5 LRU buffer를 이용한 정책에서 디스크로의 이동 순서

4.1.3 메모리에서 디스크로의 이동

노드 선정 정책에 의해서 선택된 노드들을 하나 씩 디스크에 쓰게 된다면 디스크 I/O 비용이 많이 들게 되므로, 이를 일정 단위로 묶어서 디스크에 한 번에 저장하는 벌크 연산이 필요하다. 이 때 관련 있는 노드들을 가까이 배치하기 위한 동적 클러스터링으로써 질의 시에 다시 디스크에 접근하는 경우에 발생하는 디스크 I/O를 줄일 수 있다.

이 논문에서는 벌크 연산과 동적 클러스터링의 두 가지 목적을 달성하기 위해서 이동 서브트리 결정 레벨(migration subtree decision level)과 이동 서브트리(migration subtree) 개념을 정의한다.

정의 1. *이동 서브트리 결정 레벨(migration subtree*

decision level) : 이주 대상 서브트리의 크기를 결정하기 위한 서브트리의 높이

정의 2. *이동 서브트리(migration subtree)* : 이주 노드의 그룹으로써 전체 트리에서 이동 서브트리 결정 레벨의 높이를 가진 서브트리

디스크로 이동이 일어날 때, 이동 대상 노드의 선정 정책에 따라 이동 서브트리의 구성 방법과 이동 순서가 바뀌게 된다. 따라서 노드 선정 정책에 따른 색인의 이주 과정을 아래에서 기술한다.

그림 6은 시간적으로 가장 오래된 노드를 seed로 사용하여 이동 서브트리 단위로 메모리 노드를 디스크로 이동하는 예를 보여준다. 이동을 위한 노드 선정 시점이 되었을 때, 색인의 루트 노드로부터 MBB의 시간축 값이 가장 작은 노드로 탐색하여 내려가면 그림 6에서와 같이 단말 seed 1 노드가 선택되게 된다. 이 때 이 seed를 이용하여 이동 서브트리 결정 레벨에 해당하는 만큼 대상 노드들을 확장시키면 이동 서브트리 ①이 선택되게 되고, 이것을 디스크에 기록하게 된다. 두 번째로 이동 시점이 되었을 때 앞에서 했던 방식과 같이 MBB의 시간축에 따라 색인을 탐색하게 되는데, 이때 디스크 포인터가 있을 경우는 비교대상에서 제외시키면서 탐색하면 단말 노드 seed 2 노드가 선택된다. 동일한 방법으로 이동 서브트리 ②가 디스크로 쓰여지게 되고, 이동 서브트리 ③의 경우도 이동 서브트리 ②와 같은 방법으로 기록된다. 그런데 다시 노드 선정 시점이 되면 비단말 노드가 seed로 선택되게 되고 서버트리 ⑥이 이동 서브트리로 구성된다. 그러나 서버트리 ⑥이 디스크로 이동되면 ⑥의 단말노드가 메모리 노드인 ④, ⑤를 가리키게되는 d2m 포인터가 발생하게된다. 따라서 이러한 경우를 피하기 위해서는 형제 노드가 메모리 포인터를 가지지 않는 노드일 때만 seed로 선택하는 것이 바람직하다. 결과적으로 그림 6의 색인 구조의 경우 이동 서브트리는 ①, ②, ③, ④, ⑤, ⑥의 순서로 구성 및 이동된다. 이러한 이동 서브트리 구성을 위한 seed가 선택되는 조건은 다음과 같다. 첫째, seed가 단말 노드이다. 둘째, seed가 비단말 노드일 경우 seed를 이용하여 구성한 이동 서브트리의 모든 단말노드가 디스크 포인터를 가져야 한다.

그림 7은 LRU buffer를 이용한 이주 정책에서 이동 서브트리 단위로 메모리 노드를 디스크로 이동하는 예이다. 이동 서브트리 ①은 이미 디스크로 이동한 서브트리로 가정하고 다시 메모리 공간이 부족하여 이동을 위한 노드 선정 시점이 되면 LRU buffer에서 가장 오래전에 참조된 노드를 선택하게 되며 그림 7에서는 seed 1이 선택된다고 가정한다. 이 때 이 seed를 이용하여 이동 서브트리 결정 레벨에 해당하는 만큼 선정 노드들을 확장시켜서 이동 서브트리 ②를 만들게 된다. 이때 이동 서브트리로 구성되어 디스크로 옮겨진 노드들 중에 LRU buffer에 있었던 노드는 LRU buffer에서 삭제시킨다. 왜냐하면 다음 이동 시점이 되었을 때, 이미 디스크로 이동한 노드를 seed 선정대상에서 제외하기 위함이다. 또다시 이동 시점이 되었을 때 앞에서 했던 방식과 같이 LRU buffer에서 seed로서 단말 노드 seed 2가 선택된다고 가정하면, 이동 서브트리 ③이 디스크로 쓰여지게 된다. 이 때 ①, ②, ③의 부모 노드는 비단말 노드이지만 자식 노드들이 모두 디스크로 옮겨진 상태이고 메모리 포인터를 가지지 않으므로 LRU buffer에 새롭게 포함시킨다. 이는 다시 노드 선정 시점이 되었을 때, 비단말 노드 또한 seed로 선정되도록 하기 위함이다. LRU buffer에 존재할 수 있는 조건을 일반화 시켜보면 다음과 같다. 첫째, LRU buffer에 존재하는 모든 노드는 메모리 상에 존재해야 한다. 둘째, 단말 노드이

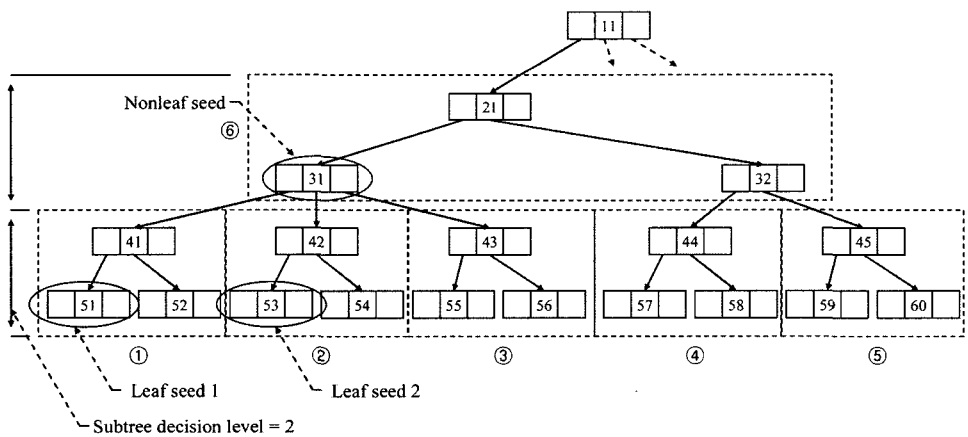


그림 6 Oldest Node를 이용한 이동 서브트리 구성

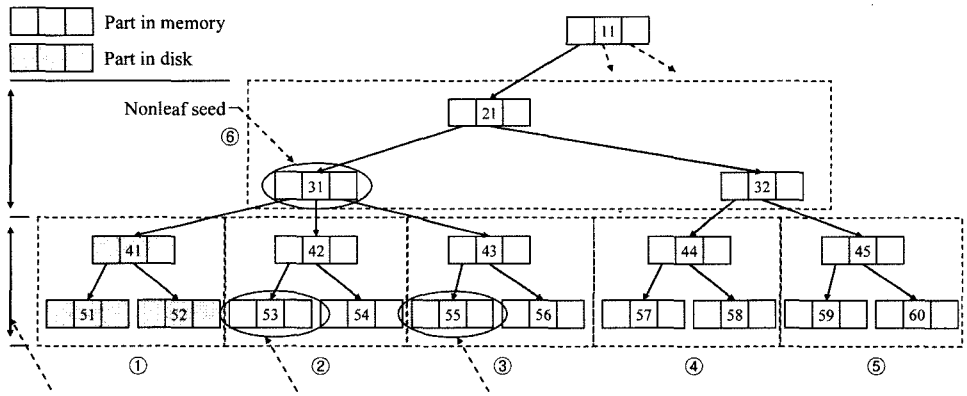


그림 7 LRU buffer를 이용한 이동 서브트리 구성

다. 셋째, 비단말 노드일 경우, 모든 자식 노드의 포인터들이 디스크 포인터를 가져야 한다.

LRU buffer를 이용한 이주 정책은 Oldest Node를 이용한 이주 정책에 비해 다음과 같은 장점을 가진다. 삽입과 질의 시에 가장 오래 전에 참조된 노드를 seed로 선택함으로써, 자주 참조되지 않는 노드가 seed로 선택되게 되어, 오래된 궤적들에 대한 질의가 다수 발생하는 조건에서는 단순히 시간적으로 오래된 노드를 seed로 선정하는 것보다 질의 성능이 뛰어나다.

4.1.4 디스크에서 메모리로의 이동

디스크에 이동된 노드가 질의에 의해서 참조될 경우, 다시 메모리로 로딩되어야 한다. 통합 색인에서는 디스크로의 이주 방법과 유사하게 이동 서브트리 단위로 다시 메모리로 로딩한다. 이러한 벌크로딩 방법은 논리적은 근접한 노드들을 함께 읽어들임으로써, 특히 영역질의를 수행할 때 디스크 I/O를 줄일 수 있다.

4.2 통합 색인의 노드 구조

그림 8은 통합 색인의 단말 및 비단말 노드의 구조를 보여준다. 색인 탐색 시에 노드에 대한 접근은 그림에서와 같이 공통 인터페이스인 identifier를 사용해서 메모리 노드와 디스크 노드를 접근할 수 있다. 노드의 구별은 identifier의 type에 의해서 이루어지고, type의 값에 따라 메모리 페이지 주소 또는 디스크 페이지 주소로 변환된다. 단말 노드의 구조는 그림과 같이 기존의 R-tree의 노드 구조와 비슷하다. 차이점은 추가적으로 엔트리에서 궤적을 위한 segment id와 3차원 내에서의 방향 정보인 orientation을 가지는 것이다. 그리고 단말 및 비단말 노드 모두 헤더 정보로서 자신의 identifier와 MBB를 가지며 identifier는 비단말 노드의 엔트리와 동일하게 address와 type을 가진다.

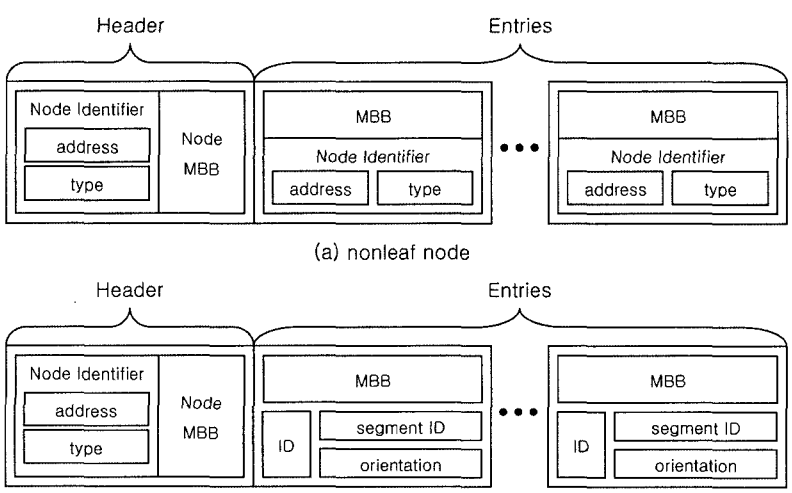


그림 8 통합 색인의 노드 구조

5. 통합 색인의 알고리즘

이 장에서는 Oldest Node를 이용한 이주 정책과 LRU buffer를 이용한 이주 정책의 알고리즘, 질의 알고리즘 및 삽입 알고리즘을 제시한다.

5.1 이주 알고리즘

메모리에서 디스크로의 노드 이동 알고리즘은 아래와 같이 크게 세 부분으로 이루어진다.

가) 이동 대상이 되는 이동 서브트리를 선정하는 단계

나) 이동 서브트리가 디스크에 이미 쓰여졌는지 유무에 따라 취소 또는 디스크로 플러싱을 수행하는 단계

다) 메모리 포인터를 디스크 포인터로 변환하는 단계

```
Algorithm Migration (N:NodePointer, P:MigrationPolicy)
begin
/* 이주 정책에 따라 subtree를 만들기에 적합한 seed를 리턴 */
if(P is OldestNode) then  /* Oldest Node 이주 정책일 때 */
  T = PickSeedNode_OldestNode(N)
else /* LRU buffer를 이용한 이주 정책일 때 */
  T = PickSeedNode_LRU()
end if
  S = SelectSubTree(T) /* 선택된 subtree의 root를 리턴 */
if(S resides in disk) then /* 이미 disk에 존재할 경우, 메모리에서 삭제함 */
   ptr = DiscardSubTree(S)
else
  /* flushing 후 subtree root 노드의 주소를 얻음*/
  /* 이주 정책이 LRU를 이용한다면, LRU_list에서 디스크로 이동한 노드들은 삭제함 */
  ptr = FlushSubTree(S, P)
end if
P = FindSubTreeParent(S)
TranslatePointer(P, S, ptr) /* subtree의 부모 노드의 포인터를 갱신 */
end
```

알고리즘 1 이주 알고리즘

Migration 알고리즘에서 이동 서브트리를 선택하기 위해 먼저 수행하게 되는 seed 선정 알고리즘의 경우 이주 정책에 따라 시간적으로 제일 오래된 데이타를 가리키는 노드를 선정하거나, 질의 및 삽입에 대한 LRU buffer에서 가장 오래 전에 참조된 노드를 선정하게 된다.

알고리즘 2는 seed를 선택하기 위해 Oldest Node를 이용한 이주 정책을 수행할 때 적용되는 알고리즘이다. 단말 노드의 경우 시간적인 조건만 만족할 경우 seed로서 선택될 수 있지만 비단말 노드의 경우는 이동 서브트리를 형성했을 때 이동 서브트리의 단말 노드가 메모리 포인터를 가지고 있으면 d2m 포인터가 발생할 수 있으므로 선정대상에서 제외시킨다.

이동 서브트리 선택을 위한 seed 선정 알고리즘 3은 LRU buffer를 이용한 이주 정책으로써, LRU_list에는 색인 구축 전에 설정값에 따라 삽입 또는 질의에 의해 참조되는 단말 노드들이 LRU 정책에 따라 정렬되어 있다. 이동 시점이 되었을 때, 이 LRU_list에서 가장 오래 전에 참조된 노드를 한 개 꺼내서 이동 서브트리를 형성한다. 이 때 이동 서브트리의 단말 노드에 해당하는 노드들이 모두 디스크 포인터를 가진다면, 이 노드를 seed로 반환하게 된다.

```
Algorithm PickSeedNode_OldestNode(N:NodePointer)
begin
  min = maximum possible value
if(N is a leaf) then
  return N
else
  if(N has memory pointer) then
  /* nonleaf node이고 memory pointer가 있을 경우 */
  /* 시간축으로 최소값을 가진 pointer를 탐색 */
    for each e in N do
        if((e.ptr is memory pointer) and
           (e.ptr is not in reject_list)) then
          if(min-t(e.mbb) < min) then
            min = min-t(e.mbb)
            ptr = e.ptr
          end if
        end if
      end for
      PickSeedNode_OldestNode(ptr)
    else
    /* nonleaf node이고 memory pointer가 없을 경우 */
    /* subtree의 모든 leaf가 memory pointer를 가지고 있지 않은지 검사 */
      if(TestSubTree(N)) then
        return N
      else
      /* subtree의 leaf 중 memory pointer가 존재 */
      /* 다음에 다시 선정되지 않도록 reject-list추가 */
        reject_list += N
        PickSeedNode_OldestNode(GetParent(N))
      end if
  end if
end if
end
```

알고리즘 2 PickSeedNode_OldestNode 알고리즘

```
Algorithm PickSeedNode_LRU()
begin
/* LRU_list : 삽입 및 질의에 의해 참조된 단말 노드들에 대한 list */
N = LRU_list.first()
while(N is not NULL)
/* subtree의 모든 leaf가 memory pointer를 가지고 있지 않은지 검사 */
  if(TestSubTree(N)) then
    return N
  else
  /* subtree의 leaf 중 memory pointer가 존재 */
  /* LRU_list에서 다음 노드를 가져옴 */
    N = LRU_list.next();
  end if
end while
return NULL
end
```

알고리즘 3 PickSeedNode_LRU 알고리즘

5.2 질의 및 삽입 알고리즘

기본적인 질의 알고리즘은 R-tree 계열의 그것과 동일하고, 단지 노드 탐색 시에 디스크 포인터가 있을 경우 이동 서브트리를 벌크 로딩(bulk loading)하는 처리 과정이 추가로 존재한다는 점이 다르다. 알고리즘 4는 통합 색인에서 영역 질의를 위한 알고리즘을 보여주고 있다.

```
Algorithm Search(N:NodePointer, R:MBB)
begin
  result = {}
  if(N is a disk pointer) then /* disk pointer일
    경우 bulk loading을 수행 */
```

```
    ptr = BulkLoadSubTree(N)
    P = GetParent(N)
    TranslatePointer(P, N, ptr)
  end if
  if(N is a leaf) then
    return N
  else
    for each e in N do
      if(e.mbb contains R) then
        result += Search(e.ptr, R)
      end if
    end for
  end if
  return result
end
```

알고리즘 4 Search 알고리즘

삽입 알고리즘은 기존의 R-tree계열의 insert 알고리즘을 적용할 수 있다. 단 ChooseSubtree 알고리즘[1], 즉 삽입할 노드를 찾기 위한 알고리즘 부분에서 디스크 포인터가 있는 엔트리들은 제외하는 부분이 추가되었다. 이는 삽입 시에 디스크에 대한 접근을 하지 않음으로 인해서 실시간 삽입의 성능을 보장해 주기 위해서이다.

6. 실 험

이 장에서는 본 논문에서 제시된 색인의 구조 및 알고리즘에 따라 구현된 통합 색인에 대하여 이동체 데이타 셋, 질의 데이타 셋에 의한 이주 정책들의 성능 결과를 실험하였다. 색인의 이주 정책 간의 성능 비교 기준은 삽입 또는 질의 시 노드에 접근할 때 마다 발생하는 디스크 I/O 회수이다. 더불어 이동 서브트리 결정 레벨 및 한 번에 이동되는 이동 서브트리 수를 다양하게 변화시킴에 따른 디스크 I/O의 변화를 관찰하였다. 제안된 통합 색인은 C++로 구현되었으며, 펜티엄 IV 3.06GHz 프로세서, 1GB 메모리와 Windows 2003 Server를 사용하여 실험을 수행하였다.

6.1 실험 환경

일반적으로 색인의 성능 검증을 위해서는 다양한 데이타 환경에서 색인 성능을 평가할 수 있는 테스트 환경이 필요하다. 이 논문에서는 이러한 연구 중 이동체 데이타 생성에 대한 대표적인 도구인 GSTD(Generate Spatio-Temporal Data) 생성기[11]를 사용하여 생성된 데이타 셋에 대하여 실험을 하였다. 이동체 데이타베이스 환경을 고려하여 통합 색인의 성능을 평가하기 위해서 표 2와 같은 삽입 데이타를 생성하였다. 이동체 위치 데이타를 생성하기 위한 매개 변수로, 이동체의 초기 분포는 가우시안 분포이며 이동체의 이동은 랜덤이다. 표 2와 같이 이동체의 총 보고 횟수가 10만 개, 50만 개, 100만 개 데이타로서 실험을 하였다.

그림 9는 1,000개의 이동체가 각각 1,000번씩 보고할 때, 일정 시간 간격에 따른 이동 분포를 나타낸 것이다.

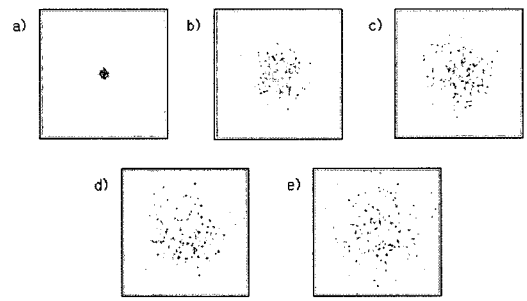


그림 9 이동체 1,000개의 시간에 따른 이동 분포

이동체 데이타베이스 환경에서 가장 일반적인 질의인 영역 질의에서 빠른 응답 시간을 제공하기 위해 디스크 상에 있는 디스크 노드로의 접근을 최소화해야 한다. 색인에서 3차원 영역 질의의 성능은 질의 영역에 관계가 있으므로 질의 영역의 크기를 변화시키면서 색인의 검색 성능을 평가하는 것이 필요하다. 본 논문에서는 생성된 색인의 시공간 영역에 대해서 1%, 5%, 10%의 크기를 갖는 질의 1,000 개와 10,000 개를 생성하여 각 색인에서 수행함으로써 검색 시 비교하는 디스크 노드 수를 측정한다. 그림 10은 전체 영역에 대하여 1%, 5%, 10%의 영역을 가지는 랜덤(random)하게 생성한 1,000 개의 질의 데이타를 나타낸 것이다.

이 논문에서 제안된 이동 서브트리를 이용한 이주 정책들의 성능 비교를 위해 노드 단위로 이동하는 정책과

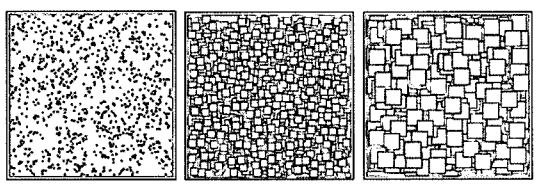


그림 10 질의 데이타

표 2 GSTD 데이타의 실험 요소

총 데이타 수(개)	이동체 수(개)	보고 회수(번)	비고
10만	100	1,000	100개의 이동체가 1,000번 보고
50만	500	1,000	500개의 이동체가 1,000번 보고
100만	1,000	1,000	1,000개의 이동체가 1,000번 보고

몇 가지 변형된 이주 정책들을 추가하여 실험을 진행한다. 표 3은 실험 평가에 사용된 5가지 이주 정책들이다. Oldest Node와 LRU for insert and search는 본 논문에서 제안된 이주 정책들이다. Brute force는 통합 색인의 이주 정책과 대비되는 정책으로서, 이주 시점에 이동 서브트리 단위가 아닌 노드 단위로 디스크로 이동이 일어나는 이주 정책이다. 그리고 LRU for insert와 LRU for search는 LRU buffer를 이용한 이주 정책의 같은 계열로서, 삽입과 질의를 개별적으로 고려하여 LRU list를 구성하는 방법이다.

6.2 성능 평가

6.2.1 이주 정책 간의 질의 성능 비교

이 실험에서는 이주 정책들의 성능을 평가하기 위해 이동체 데이타의 수를 변화시키면서 다양한 크기의 영역 질의를 수행하였다. 이동체의 수를 변화시킬 때 사용한 이동체의 수는 100, 500, 1,000 개이고, 이동체의 보고 횟수는 1,000 번을 사용하였다. 질의 개수는 1,000 개이며, 질의 영역은 1%, 5%, 10%이다.

전반적으로 LRU buffer를 이용한 이주 정책의 성능이 Oldest Node 정책에 비해 2~5% 정도 우수하였다. 이동 서브트리 방식을 적용하지 않는 Brute force 방식은 노드 단위로 이동이 일어나므로 잦은 디스크 I/O를 발생으로 성능이 저하되는 특징이 나타났다. 또한 질의 셋을 최근 업데이트가 이뤄진 데이타에 대한 질의가 주로 일어나도록 설정한 일부 실험에서는 Oldest Node 정책의 성능이 LRU buffer 보다 좋은 결과를 보였다.

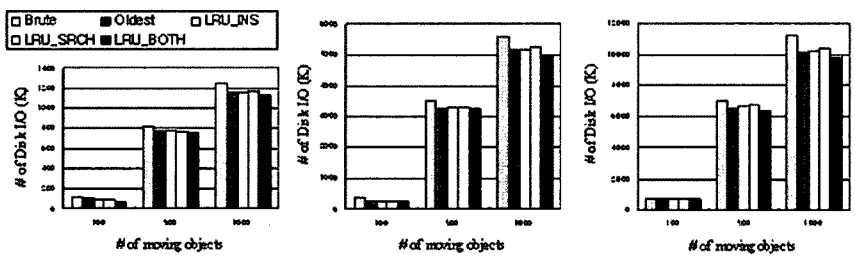
6.2.2 이동 서브트리 결정 레벨에 따른 질의 성능 비교

이 실험에서는 이동체 수를 1,000 개, 보고 횟수는 1,000 번, 그리고 질의 횟수를 1,000 번으로 고정시켰을 때, 이동 서브트리 결정레벨을 변화시키면서 이동 서브트리 방식을 사용한 이주 정책 간의 성능을 비교하였다. 이동 서브트리 결정레벨은 이동 서브트리의 높이를 결정하는 중요한 인자로서 높이가 높아질수록 디스크로 이동하는 노드의 수가 증가한다는 특징을 가진다.

전반적으로 LRU buffer를 이용한 이주 정책이 Oldest Node 정책에 비해 좋은 성능을 보였다. 그러나

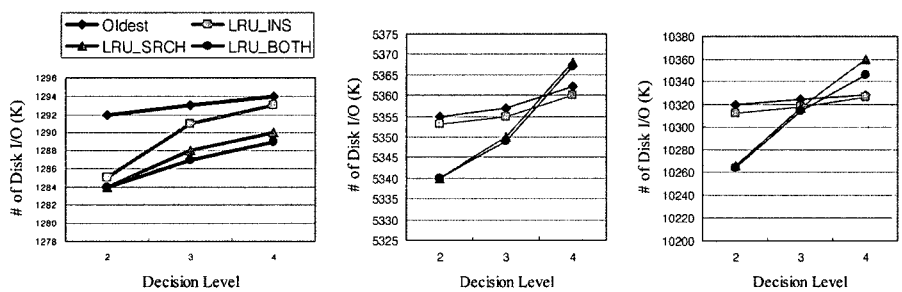
표 3 실험 평가에 사용된 이주 정책

이주 정책	설 명
Brute force	가장 오래된 한 개의 노드를 선택하여 디스크로 이동
Oldest Node	가장 오래된 노드를 이동 서브트리 구성을 위한 seed로서 선택
LRU for insert and search	삽입과 질의 시에 참조되는 노드들로 seed 선정을 위한 LRU list를 구성
LRU for insert	삽입 시에 참조되는 단말 노드들로 seed 선정을 위한 LRU list를 구성
LRU for search	질의 시에 참조되는 단말 노드들로 seed 선정을 위한 LRU list를 구성



(a) Search (1%) (b) Search (5%) (c) Search (10%)

그림 11 이주 정책들의 질의 성능



(a) Search (1%) (b) Search (5%) (c) Search (10%)

그림 12 이동 서브트리 결정 레벨에 따른 질의 성능

Oldest Node 정책의 경우, decision level이 커지더라도 성능 감소가 거의 없다. 이는 색인의 특정 부분(과거 이력)을 집중적으로 디스크로 이동시킴으로써, 이동 서브트리의 크기가 증가하더라도 색인의 전반적인 모습의 변화는 거의 없기 때문이다.

7. 결 론

이동체 데이타베이스는 이동체의 위치를 저장 및 관리하기 위한 저장소이다. 이러한 데이타베이스는 이동체 정보를 빠르게 검색하기 위해 색인을 가져야 하며, 이 색인은 다수의 이동체에 의해 갱신되는 업데이트를 관리하고 실시간으로 위치를 추적할 수 있어야 한다. 따라서 이동체 데이타베이스를 위한 색인은 실시간 처리를 위해서 메인 메모리에서 동작하는 색인의 구조를 가져야 하며, 다수 이동체의 위치 정보를 관리하기 위해 장기간 엑세스되지 않는 정보를 디스크로 이동시켜 저장할 수 있어야 한다.

이 논문에서는 이러한 요구 조건을 충족시키기 위해서 메인 메모리와 디스크를 통합한 통합 색인을 구현하기 위해서 고려되어야 할 사항인 d2m 포인터의 문제점과 해결책, 디스크로 이주하기 위한 시기 및 정책들을 제안하였다. 또한 디스크 I/O를 줄이기 위한 이동 서브트리 및 이동 서브트리 결정 레벨을 정의하여 벌크로딩 방법을 도입하는 방법을 기술하였다. 마지막으로, 제시된 통합 색인에 대한 구현 및 실험을 통하여 각 이주 정책들의 성능을 평가하였다. 향후, 색인의 메모리 부분과 디스크 부분의 노드 크기에 따른 성능 변화 및 최적값 도출과 같은 다양한 성능 평가가 필요하며, R-tree 이외의 이동체 색인에 본 논문에서 제시하는 정책들을 적용하는 방안도 연구되어야 한다. 마지막으로 통합 색인에 적용될 수 있는 추가적인 이주 정책에 관한 연구도 필요하다.

참 고 문 헌

[1] A. Guttman, "R-trees: A dynamic index structure for spatial searching," Proc. of the ACM SIGMOD, pp. 47-54, 1984.

[2] T. J. Lehman and M. J. Carey, "A Study of Index Structures for Main Memory Management Database Systems," Proc. of VLDB Conference, pp. 294-303, 1986.

[3] D. Gawlick and D. Kinkade, "Varieties of concurrency control in IMS/VS Fast Path," IEEE Database Eng., Vol. 8. No. 2, pp. 3-10, June 1985.

[4] D. Lomet and B. Salzberg, "Access Methods for Multiversion Data," Proc. of ACM SIGMOD, pp. 315-324. 1989.

[5] M. Stonebraker, "Managing persistent objects in a multi-level store," Proc. of ACM SIGMOD, pp. 2-11, 1991.

[6] N. Beckmann and H. P. Kriegel, "The R*-tree: An efficient and robust access method for points and rectangles," Proc. of ACM SIGMOD, pp. 332-331, 1990.

[7] B. Seeger, P.A. Larson, and R. McFayden. "Reading a set of disk pages," Proc. of VLDB Conference, pp. 592-603, 1993.

[8] H. Garcia-Molina and K. Salem, "Main memory database systems: An overview," IEEE Trans. Knowledge Data Eng., Vol. 4, No. 6, pp. 509-516, 1992.

[9] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects," In Proc. of the VLDB Conference, pp. 395-406, 2000.

[10] Güting, R., Böhlen, M., Erwig, M., Jensen, C. S., Lorentzos, N., Schneider, M., and Vazirgiannis, M. "A Foundation for Representing and Querying Moving Objects," ACM Transactions on Database Systems, Vol. 25, No. 1, pp. 1-42. 2000.

[11] Y. Theodoridis, J. R. O Silva and M.A Nascimento, "On the Generation of Spatiotemporal Datasets," Proc. of Symposium on Advances in Spatial Databases, pp. 147-164, 1999.

박 재 관

1999년 부산대학교 컴퓨터공학과 졸업(공학사). 2001년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사). 2003년 부산대학교 대학원 컴퓨터공학과 박사과정 수료. 2001년~현재 ㈜사이버맵웰드 부설 연구소 선임연구원. 관심분야는 지리정보시스템, RFID 미들웨어, 모바일 GIS, 이동체 색인

안 경 환

1997년 부산대학교 컴퓨터공학과 졸업(공학사). 1999년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사). 2004년 부산대학교 대학원 컴퓨터공학과 졸업(공학박사). 2004년~현재 한국전자통신연구원 LBS연구팀 연구원. 관심분야는 LBS, 이동체 데이타베이스, 스트림데이타처리기

정 지 원

2003년 부산대학교 컴퓨터공학과 졸업(공학사). 2005년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사). 2005년~현재 삼성전자 정보통신총괄 무선사업부 사원. 관심분야는 LBS, 이동체 색인

홍 봉 희

정보과학회논문지 : 데이타베이스 제 33 권 제 1 호 참조