이동체 데이터베이스를 위한 통합 색인의 성능 평가[1]

정지원* 안경환° 홍봉희*

부산대학교 컴퓨터공학과*, 한국전자통신연구원 텔레매틱스 연구단 LBS 연구팀°

{whitegem, bhhong}@pusan.ac.kr*, mobileguru@etri.re.kr °

Evaluation of Unified Index for Moving Object Databases

Jiwon Jung* Kyunghwan Ahn° Bonghee Hong*

요 약

이동체 데이터베이스에서 이동체의 과거 궤적을 저장하기 위해 메인 메모리 기반 색인을 이용하면 시간이 지남에 따라 데이터의 방대함으로 인해 주어진 메모리 용량이 부족하게 될 수 도 있다. 이를 해결하기 위해서는 메인 메모리에 상주하는 색인의 일부를 계속해서 디스크로 이주하는 정책이 필요하다. 이런 이주 정책을 지원하는 메인 메모리 이동체 색인이 통합 색인이다. 기존 통합 색인의 색인 이주 정책인 이동 서브트리 정책은 시간 축으로 가장 오래된 엔트리를 seed 노드로 선정하여 이동 서브트리를 구성한다. 이때 항상 시간적으로 가장 오래된 노드만을 디스크로 옮김으로써 과거에 대한 질의 시 비효율적이라는 문제점을 가진다.

본 논문에서는 이주를 위한 서브트리 구성에 필요한 seed 노드를 선택하기 위해, 질의 및 삽입 시에 참조되는 단말 노드들을 유지하는 LRU 버퍼를 이용한 색인 이주 정책을 제시한다. 이를 바탕으로 메인 메모리 기반 색인의 장점과 메모리 용량 부족의 문제를 해결한 통합색인을 구현하고, 다양한 성능 평가를 통하여 제시된 이주 정책이 기존의 이주 정책에 비해 삽입 성능뿐만 아니라 영역 질의에서도 우수함을 보인다.

1. 서 론

이동 통신 기술의 발달로 인하여 휴대폰이나 PDA와 같은 무선 이동 기기의 사용이 보편화되면서 LBS(Location Based Service)의 요구가 나날이 증대되고 있다. 이와 같은 위치 기반 서비스에서 시간에 따라 위치가 변하는 객체를 이동체라고 하며 이를 저장하고 관리할 수 있는 시스템을 이동체 데이터베이스 시스템이라고 한다. 이동체 데이터베이스 시스템은 기존의 데이터베이스와 다른 다음과 같은 특징을 가진다. 첫째, 이동체의 위치 보고 데이터에 대한 실시간 갱신을 필요로 한다. 둘째, 시간이 지남에 따라서 계속해서 위치 데이터가 보고 되므로 위치 보고 데이터의 양이 방대하다.

이런 특징을 가진 이동체의 현재 위치를 실시간으로 저장 및 검색하기 위해서는 이동체 데이터베이스를 메인 메모리 기반의 데이터베이스로 구현할 필요가 있다. 그러나 계속해서 누적되는 위치 데이터를 모두 메모리에 저장하기는 힘들다. 그러므로 검색 가능성이 낮은 과거 궤적 데이터들은 디스크로 이동하거나 삭제하는 것이 필요하다.

상기의 문제점들을 해결하기 위한 접근

[1] 본 연구는 교육부에서 주관하는 "차세대물류IT기술연구사업단"에 의해 지원 받은 연구임.

방법으로 메인 메모리 색인과 디스크 기반 색인의 특징을 결합한 통합 색인(Unified Index)에 관한 연구가 필요하다.

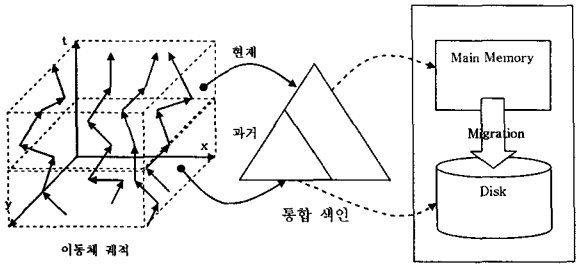

그림 1 통합 색인

본 논문에서는 이동체 데이터베이스 색인의 요구 조건을 만족시키기 위해 그림 1과 같이 통합 색인에 대한 이동 정책을 연구하고자 한다. 이동체 궤적이 계속 누적되어, 메인 메모리 기반 색인이 주어진 메모리의 용량을 넘어서게 되면 자동으로 색인의 일부를 디스크로 이주(migration)시켜 주게 된다. 그리고 메모리에서 디스크로의 색인 이주 시에 발생할 수 있는 디스크 I/O를 최소화할 수 있는 방법에 대해서도 연구하고자 한다. 더불어, 이러한 통합색인을 위한 LRU buffer를 이용한 이주 정책을 제시하고, 실제 통합색인을 구현하고 실험 성능 평가를 통하여 제시한 이주 정책이 기존의 이주 정책에 비해 우수함을 보인다.

이 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구를 그리고 3장에서 대상환경 및 문제정의를 기술한다. 4장에서는 새로운 이주정책을 제시하고, 5장에서 통합색인의 실제 구현 및 실험 평가를 통하여 논문에서 제시한 방안을 기존의 연구와 비교한다. 마지막으로 6장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

기존에는 데이터의 양 및 사용 빈도에 따라 데이터를 다른 매체에 분리 저장하는 구조에 관한 연구[1], 색인의 이동에 관한 연구[2]와 데이터를 디스크 상에서 군집화(clustering)하는 연구[3]가 있었다.

데이터를 다른 매체에 분리하여 저장하는 연구로는 [1]이 있는데, 이는 자주 사용되는 데이터는 메모리 기반 데이터베이스에 저장하고 자주 사용되지 않는 데이터는 디스크 기반 데이터베이스에 저장하는 방법을 사용한다. 예를 들면, 은행에서 계좌정보와 고객정보를 색인으로 구축한다고 할 때, 계좌정보는 메모리에 고객정보는 디스크에 저장하는 방식을 취할 수 있다. 그러나 한 가지 종류의 데이터가 시간이 지남에 따라서 양이 많아 질 경우, 데이터를 분류하여 저장해야 하기 때문에 자동적으로 저장구조가 상이한 매체간 이동이 지원되지 않는 문제점을 가지고 있다.

색인의 이동에 관한 연구로는 [2]가 있다. 이 연구는 현재 사용되고 있는 데이터는 디스크에 저장하고 과거의 사용되지 않는 데이터는 광학 저장 장치로 이동시키는 방법에 관한 연구이다. [2]에서 제시한 Time-Split B-tree에서는 시간축으로 노드가 분할될 때 한 번에 하나의 노드씩 광학 저장장치로 이동되기 때문에 이 논문에서 요구하는 디스크 I/O를 줄이기 위한 방법으로 적용하기 어려우며, 노드에 중복된 데이터가 존재함으로 인해서 메모리 및 디스크에 적용했을 때 공간을 효율적으로 사용하지 못하는 단점이 있다.

데이터를 디스크 상에서 클러스터링하는 연구로는 [3]이 있다. 이는 데이터 접근 시 원하는 페이지로의 디스크 헤드 접근 시간이 페이지당 전송 시간보다 훨씬 크기 때문에 디스크 I/O 시간을 줄이기 위해 데이터를 적절히 배치하는 연구이다. 그러나 이는 정적인 데이터를 클러스터링하는 것으로서 실시간 대용량 이동체 환경에서 적용하기 어렵다는 문제점을 가지고 있다.

3. 대상 환경 및 문제 정의

3.1 대상 환경

이 논문에서는 이동체가 주기적으로 서버에 자신의 위치 정보와 필요시 질의 정보를 전송한다고 가정한다. 서버에서는 위치 정보를 메인 메모리에 저장하고, 메모리 공간이 부족할 경우 과거 이력 데이터는 디스크로 옮겨진다. 클라이언트에서 질의 요청이 오면, 저장하고 있는 위치 정보를 이용하여 응답하는 환경을 대상으로 하고 있다.

3.2 문제 정의

통합색인에는 메모리와 디스크 전반에 걸쳐 노드가 존재하여 다양한 형태의 포인터가 존재한다. 이들을 포인터의 방향에 따라 아래와 같이 분류할 수 있다.

포인터의 방향	포인터 이름
memory ⇨ memory	m2m 포인터
memory ⇨ disk	m2d 포인터
disk ⇨ memory	d2m 포인터
disk ⇨ disk	d2d 포인터

표 1 통합 색인의 포인터

그런데 d2m 포인터[4]의 경우 메모리에 존재하는 노드를 접근하기 위해 디스크 I/O가 발생한다는 문제점을 가진다. 이를 방지하기 위해서는 노드의 이동 단계에서 d2m 포인터가 발생하지 않도록 하는 이동 정책이 필요하다.

메모리에서 디스크로 이동할 때에는 두 가지 사항이 고려되어야 한다. 첫째, 디스크 I/O를 줄이고, 데이터 삽입 중에 시스템의 블로킹(blocking)이 발생하지 않도록 하는 알고리즘이 필요하다. 둘째, 디스크에 저장될 때 관련성이 있는 노드들에 대한 동적 클러스터링(dynamic clustering)을 적용하여 디스크상에 위치시키는 것이 필요하다. 이는 디스크 기반 데이터베이스에서는 발생하지 않는 메인 메모리 기반 데이터베이스에만 존재하는 문제이다[5].

반면, 디스크로 이동한 노드들이 질의에 의해서 참조될 경우에 디스크 기반 색인과 같이 노드들을 하나씩 접근하여 메모리로 이동하는 방법은 많은 디스크 I/O를 발생시킨다. 그러므로 벌크 로딩(bulk loading)과 같은 디스크 I/O를 줄이기 위한 알고리즘이 필요하다.

이와 같은 고려사항에 따라 기존 연구에서는 이동 서브트리 결정 레벨이라는 개념을 도입하고 이동 서브트리(migration subtree) 단위로 디스크 연산을 사용하였다[4]. 기존의 이동 서브트리 정책은 노드 선정 시점이 되었을 때, 루트(root)로부터 MBB의 t축 값이 가장 작은 노드로 탐색하여 내려가면서 단말 seed를 선택한다. 이 때 이 seed를 이용하여 이동 서브트리 결정 레벨에 해당하는 만큼 선정 노드들을 확장시켜나가게 되면 이동 서브트리가 선택되게 된다.

그런데 이동 서브트리 구성을 위한 seed 선정에 있어 단지 색인 상에서 시간적으로 오래된 노드만을 선택하게 되어 항상 색인의 과거부분이 디스크로 이동하게 된다는 문제점을 가진다. 즉, 질의 처리시에 과거부분에 대해서는 항상 디스크에서 읽어와야 하므로 추가적인 디스크 I/O가 불가피하다.

4. LRU buffer를 이용한 이주 정책

LRU buffer를 이용한 이주 정책도 기존의 연구와 마찬가지로 벌크 연산과 동적 클러스터링의 두 가지 목적을 달성하기 위해서 이동 서브트리 결정 레벨(migration subtree decision level)이라는 개념을 도입하고 이동 서브트리(migration subtree) 단위로 디스크 연산을 수행한다. LRU buffer는 삽입과 질의 시에 참조되는 단말 노드들로 구성되어 있으며, seed 선정 시에 이 LRU buffer로부터 가장 오래 전에 참조된 노드를 seed로 선정하여 이동 서브트리를 구성한다.

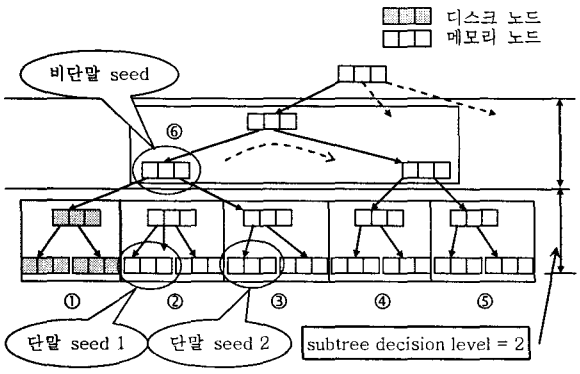

그림 2 LRU buffer를 이용한 이주 과정

그림2는 LRU buffer를 이용한 이주 정책에서 이동 서브트리 단위로 메모리 노드를 디스크로 이동하는 예이다. 이동 서브트리 ①은 이미 디스크로 이동한 서브트리이다. 메모리 공간이 부족하여 이동을 위한 노드 선정 시점이 되었을 때, LRU buffer에서 가장 오래 전에 참조된 노드를 선택하게 되며 그림에서는 seed 1이 선택되게 된다. 이 때 이 seed를 이용하여 이동 서브트리 결정 레벨에 해당하는 만큼 선정 노드들을 확장시켜서 이동 서브트리 ②를 만들게 된다. 이 때 이동 서브트리를 flush daemon에 인자로 넘겨주게 되면 flush daemon이 디스크에 이동 서브트리를 쓰게 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. 두 번째로 이동 시점이 되었을 때 앞에서 했던 방식과 같이 LRU buffer에서 seed로서 단말 노드 seed 2가 선택된다면, 이동 서브트리 ③이 디스크로 쓰여지게 된다. 이때 그림의 비단말 seed는 이동 서브트리 ①②③의 부모이지만 자식 노드들이 모두 디스크로 옮겨진 상태이다. 즉, 그림의 비단말 seed는 전혀 메모리 포인터를 가지지 않으므로 LRU buffer에 새롭게 포함시킨다. 이는 다시 노드 선정 시점이 되었을 때, 비단말 노드 또한 seed로 선정되도록 하기 위함이다. 이와 같은 방식으로 디스크로의 이주가 일어 나게 된다.

LRU buffer에 존재할 수 있는 조건을 일반화 시켜보면 다음과 같다.

- LRU buffer에 존재하는 모든 노드는 메모리 상에 존재해야 한다.
- 단말 노드이다.
- 비단말 노드일 경우, 모든 자식 포인터들이 디스크 포인터를 가져야 한다

LRU buffer를 이용한 이주 정책은 다음과 같은 장점을 가진다. 첫째, 이동 서브트리를 이용하므로 벌크 연산으로 인한 디스크 I/O 비용을 줄일 수 있다. 둘째, 이동 서브트리를 하나로 묶어서 I/O를 할 경우 질의 시에 디스크 I/O를 줄일 수 있다. 셋째, 삽입과 질의 시에 가장 오래 전에 참조된 노드를 seed로 선택하므로써, 단순히 시간적으로 오래된 노드를 seed로 선정하는 것보다 질의 성능이 뛰어나다.

다음은 이주 정책으로서 LRU buffer를 이용할 때, 통합 색인의 이주 과정을 기술한 것이다.

1. seed node selection
   - LRU buffer에서 가장 오래 전에 참조된 노드를 선택
   - 선택된 노드가 아래의 조건을 만족하지 않는다면, LRU buffer에서 다음으로 오래 전에 참조된 노드를 선택
     - ➢ Root node 제외
     - ➢ Memory pointer를 가진 노드 제외
     - ➢ Root로부터 subtree decision level이내의 노드 제외
2. subtree selection
   - seed 노드에서 subtree decision level 상위의 부모를 공통으로 가지는 모든 자식 노드를 선택
3. flushing
   - 검색을 통해 메모리에 존재하는 서브트리의 경우 제외
   - Flush buffer에 node를 배열
   - Flush daemon이 disk로 flush
4. pointer 변환
   - 서브트리의 부모 노드의 포인터를 m2d

pointer로 변환
- 메모리 상의 서브트리를 삭제

5. LRU buffer 갱신
- 디스크로 이동한 노드는 LRU buffer에서 삭제 및 갱신
- 비단말 노드 중에서 모든 자식 노드들이 디스크로 이동한 노드는 LRU buffer에 삽입

5. 실험 평가

5.1 실험 요소

통합 색인을 평가하기 위해서 다음과 같은 삽입과 검색 데이터를 사용하여 삽입 및 검색 시 일어나는 디스크 I/O 횟수를 측정하였다. 삽입은 10만개(100*1000:100개의 이동체가 1000번 보고)의 데이터를 가지고 수행하였으며 검색은 전체영역의 1%, 5%, 10%의 크기를 가진 10,000개의 영역 질의(Range Query)를 삽입이 일어나는 과정 중에 계속해서 수행하였다. 또한 제안된 이주 정책과의 질의 성능 비교를 위해 기존의 이주 정책과 몇 가지 이주 정책을 추가하여 테스트를 진행하였다.

다음은 실험 평가에 사용된 5가지 이주 정책들이다.

- •Brute force : 이주 시점에 시간축으로 가장 오래된 한 개의 노드를 선택하여 디스크로 이동
- •Oldest node(이동서브트리) : 기존 연구에서 제시된 이주 정책으로서 이동서브트리 구성을 위한 seed로서 시간축으로 가장 오래된 노드를 선택
- •LRU for insert and search : 본 논문에서 제시한 이주 정책으로 삽입과 질의 시에 참조되는 단말 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

5.2 실험 파라미터

이 논문에서 사용하는 노드의 크기는 일반적인 디스크 기반 색인의 크기와는 다르게 디스크 페이지 보다 작게 만들어 줌으로써 더 나은 성능을 얻을 수 있다. 따라서 미리 수행해 본 실험에서의 결과를 토대로 노드 용량을 512byte로 줄여서 실험을 수행하였다. ([6]에서 메모리 기반 색인들의 성능평가에서 작은 노드의 크기가 더 좋은 성능을 나타내고 있다.)

5.3 실험 세부 사항

색인은 C언어로 구현되었으며, windows 2003 운영체제, 1GB의 메인 메모리, CPU는 Pentium IV 3.06Ghz의 사양을 가진 컴퓨터에서 실험하였다. 데이터 집합은 GSTD 생성기[7]를 사용하여 생성하였다.

5.4 실험 결과

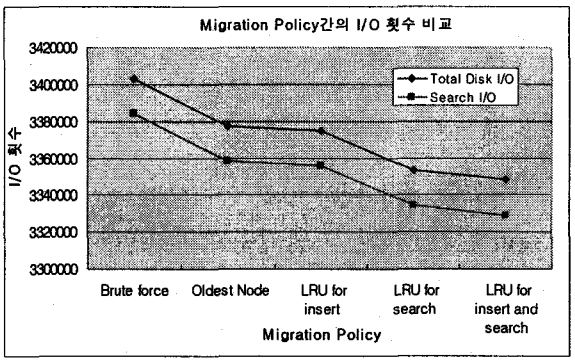

그림 3 이주 정책 간의 I/O 횟수 비교

그림 3에서 보는 바와 같이 Brute force 방식은 매 이주시점마다 한 개 씩의 노드를 디스크로 이동시킴으로써, 많은 디스크 I/O가 일어나고 있다. 또한 기존 이주 정책인 Oldest node 방식은 Brute force 방식에 비해 좀 더 나은 성능을 보이고 있지만, 본 논문에서 제시한 LRU buffer를 이용한 이주 정책보다 성능이 떨어짐을 알 수 있다. 이는 제시된 이주 정책이 이동 서브트리 구성을 위한 seed 선정을 위해 삽입과 질의를 동시에 고려하여 삽입과 질의 시에 메모리에서 참조가 자주 발생하기 때문이다.

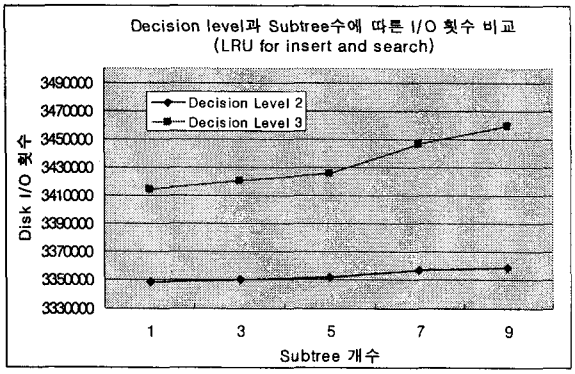

그림 4 Decision level과 Subtree 수에 따른 I/O 횟수 비교

그림 4는 본 논문에서 제시된 이주 정책에서 이동 서브트리 결정 레벨(Decision level)과 이주 시점에 디스크로 한번에 옮겨지는 이동 서브트리 개수의 변화에 따른 디스크 I/O의 변화를 나타낸 것이다. 이동 서브트리 결정 레벨을 한 단계 높임으로써 보다 많은 노드가 디스크로 이동하게 되어 전체적으로 I/O가 증가하였음을 알 수 있다. 또한 이주 시점에 디스크로 이동하는 서브트리의 수를 증가시킴으로써 잦은 디스크로의 이주를 막을 수 있었지만, 질의 시에 디스크 참조가 증가하여 디스크 I/O가 증가하였음을 알 수 있다. 참고로 Brute force 방식을 제외한 다른 모든 이주 정책들에서도 그림 5와 비슷한 양상을 보였다.

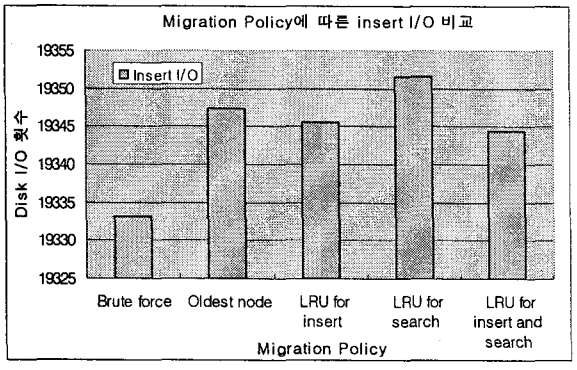

그림 5 삽입 I/O 횟수 비교

각 이주 정책에 따른 삽입 I/O 변화를 측정하였다(그림 5). Brute force는 노드 단위로 이주하기 때문에 가장 적은 디스크 I/O를 보였다. 그러나, 나머지 이주 정책들은 이동 서브트리 단위로 대량의 노드들을 이주시키므로 삽입 시에 디스크 I/O가 일어날 확률이 높다. 특히, 질의에 의해 참조되는 노드만을 LRU buffer에 유지하는 정책(LRU for search)은 삽입에 사용되는 노드들은 전혀 고려하지 않고 이주를 수행하였기 때문에 삽입 시에 조금 더 많은 디스크 I/O가 일어났다.

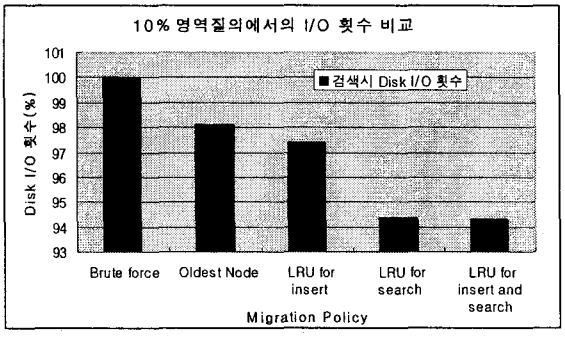

그림 6 10% 영역 질의에서의 디스크 I/O 횟수 비교

좀 더 큰 영역 질의에서의 이주 정책 간에 어떤 변화가 생기는지 알아보았다(그림 6). 이는 10%의 영역 질의를 수행함으로써 검색 성능을 측정하고, brute force 방식을 기준으로 삼아 다른 정책과의 관계를 백분율로 표현한 것이다. 그림에서 보는 바와 같이, LRU buffer를 이용하는 정책들이 전반적으로 좋은 성능을 보였다. 특히 질의 영역이 커짐으로써, 삽입 시의 디스크 I/O와 질의 시의 디스크 I/O 사이의 격차가 커져서, `LRU for search`와 `LRU for insert and search`간의 차이가 거의 없어졌음을 알 수 있다.

실험 결과에서 보듯이, 제안한 LRU buffer를 이용한 이주 정책은 기존의 이동 서브트리 방식을 사용함으로써, 벌크 로딩(bulk loading)의 효과뿐만 아니라, 시공간적으로 인접할 가능성이 높은 노드들을 하나로 묶어서 I/O하므로 동적 클러스터링 효과를 가진다. 더불어, 삽입 및 질의 시에 일어나는 참조에 대한 LRU buffer를 이용하여 이주를 위한 seed를 선택하였기 때문에, 삽입 및 질의 성능에서 다른 이주 정책보다 우수하다.

6. 결론 및 향후 연구

메인 메모리 기반 색인을 이용하여 이동체의 과거 궤적을 저장함으로써, 시간이 지남에 따라 데이터의 방대함으로 인해 주어진 메모리 용량이 부족하게 될 수 있다. 이를 해결하기 위해서 메인 메모리에 상주하는 색인의 일부를 계속해서 디스크로의 이주를 지원하는 통합 색인이 개발되었다. 그러나 기존 통합 색인의 이주 정책은 과거에 대한 질의 시 비효율적이라는 문제점을 가진다. 본 연구에서는 이동 서브트리 구성에 필요한 seed 노드를 선택하기 위해, 질의 및 삽입 시에 참조되는 단말 노드 들을 유지하는 LRU 버퍼를 이용한 색인 이주 정책을 제안하였다. 또한 이를 바탕으로 통합색인을 구현하고 성능 평가를 통하여 제안된 이주 정책이 기존의 이주 정책에 비해 삽입 성능뿐만 아니라 영역 질의에서도 우수함을 보인다.

향후 연구로는 색인 구축 환경에 따른 적절한 이동 서브트리 결정 레벨의 선택 기준에 관한 연구가 필요하다. 또한 이주 시점에 얼마나 많은 이동 서브트리를 옮기는 것이 통합 색인의 성능에 영향을 줄 수 있는지에 대한 다양한 실험 평가에 관한 연구도 필요하다.

참고 문헌

[1]D.Gawlick and D.Kinkade, " Varieties of concurrency control in IMS/VS Fast Path," Data Eng. Bull., vol.8. no.2, pp.3-10, June 1985.

[2]D. Lomet and B. Salzberg, "Access Methods for Multiversion Data", Proc. of SIGMOD, 1989

[3]D.S. Cho, B.H. Hong, "Optimal Page Ordering for Region Queries in Static Spatial Databases," 11th International Conference DEXA, pp 366-375, 2000.

[4]K.H. Ahn, B.H. Hong, "이동체 데이터베이스를 위한 통합 색인 기법," 한국정보과학회 데이터베이스 연구(KDBC 2003) 19권 2호 pp 36-43, 2003. 5.

[5]Hector Garcia-Molina and Kenneth Salem. Main memory database systems: An overview. IEEE Trans. Knowledge Data Eng., 4:509-516, 1992

[6]T.J.Lehman and M.J.Carey," A Study of index structures for main memory database management systems," in Proc.12th Conf. on VLDB, Aug.1986

[7]Y. Theodoridis, J. R. O Silva and M.A Nascimento, " On the Generation of Spatiotemporal Datasets" , SSD, Hong Kong, LNCS 1651, Springer, p147-164, 1999.