제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

MMDB에서의 실내 환경에 적합한 BITMAP INDEX와 탐색기법

전현식, 박현주
한밭대학교 정보통신공학과
e-mail:ngelmaum@empal.com

BITMAP INDEX and Searching Strategies On MMDB Adapt To Indoor Environment

Hyeon-Sig Jeon, Hyun-Ju Park
Information Communication Engineering, HANBAT National University

요 약

공간 질의 및 색인에 관한 기존 연구는 주로 실외 환경에 기반을 두고 있다. 실내 환경은 실외 환경과는 달리 질의 특성 및 환경적 요소가 다르다. 실내 환경 질의의 대표적인 특징은 객체의 현재 위치를 파악하고 즉시 응답해야하며, 질의 범위도 지역적으로 제한되어 있는 점이다. 본 논문에서는 기존 연구가 가진 문제점을 해결하기 위해 메인 메모리 기반의 DBMS를 사용하며, 실내 환경에서 객체의 위치 탐색시 효율적으로 적용할 수 있는 비트맵 인덱스 기법을 제안한다.

1. 서론

현재 세계적으로 많은 국가와 기업들이 공간 영역 질의 분야에 많은 관심을 가지고 연구하고 있다. 연구 초기에는 위치 추적 기술과 저장 및 질의 기술 등 기반 기술에 초점을 두었으나 현재는 관련 기술을 이용하는 어플리케이션 영역도 확대되어 다양한 서비스를 제공하기 위해 해당 어플리케이션에 최적화된 영역 질의를 필요로 한다.

초기 어플리케이션의 대표적인 예로 이동통신사에서 제공하는 컨텐츠 중 '친구 찾기' 기능을 들 수 있다. 이 서비스는 실외 환경에서 기지국의 TDOA (Time Difference of Arrival) 기술을 이용하거나 GPS 기술을 이용하여 위치를 파악해 고객에게 제공한다. 연구 초기에는 이와 같이 실외 환경에서 적용할 수 있는 기반 기술과 어플리케이션에 초점을 맞추었으나 현재는 실내 환경에 적용 가능한 기반 기술과 어플리케이션에도 많은 관심을 두고 연구하고 있다.

기존 실외 환경에서의 연구를 실내 환경에 적용할 때 나타나는 대표적인 문제점은 다음과 같다. 첫째, 디스크 기반 DBMS를 사용하기 때문에 실내 환경에서 우선시되는 빠른 응답성을 보장하기 어렵다. 실내 환경에서의 질의 특성으로 인해 객체의 이동 경로 및 현재 위치를 요청한 즉시 실시간으로 응답해야 하기 때문에 메인 메모리 DBMS를 사용하는 것이 유리하다. 둘째, 환경적 요소와 질의 특성이 다르기 때문에 기존 저장 구조 및 색인 구조는 비효율적인 구조를 가진다. 간단한 예로 실내 환경은 질의 영역이 적은 공간으로 한정되고 세부적으로 정확히 표현해야한다. 하지만 기존 저장 구조와 색인 구조는 실내 환경의 고유한 특성을 제공하기 어려운 구

조이다.

이러한 문제점을 해결하기 위해 본 논문에서는 메인 메모리 데이터베이스를 사용하여 빠른 응답성을 보장하고, 비트맵 인덱스를 사용하여 메인 메모리 데이터베이스에 효율적으로 적용할 수 있는 탐색기법에 대해 제안한다.

2. 관련연구

본 절에서는 실내 환경과 실외 환경의 차이점과 MMDB(Main Memory Database)의 특징을 살펴보고, MMDB에서 사용하는 대표적인 다차원 색인구조인 해시기반 인덱스와 트리기반 인덱스에 대해 간단히 살펴본다. 마지막으로 본 논문에서 활용하는 비트맵 인덱스의 특징에 대해 간단히 살펴본다.

2.1 실내 환경과 메인 메모리 데이터베이스

실내 환경과 실외 환경의 대표적인 차이점은 다음과 같다. 첫째, 질의하고자 하는 범위와 특성이 다르다. 실내 환경은 작은 공간에서 정확한 질의를 목표로 하고, 실외 환경은 넓은 공간에서의 위치 측정을 목표로 둔다. 둘째, 유지하는 정보와 특성이 다르다. 실내 환경에서는 거의 모든 객체에 대한 각각의 일련의 정보를 유지하고 이동 객체와 고정 객체를 구분하는 것이 효율적이지만, 실외 환경에서는 꼭 필요한 정보만 추출하여 유지해야 한다. 셋째, 요청하는 정보의 특성이 다르다. 실내 환경에서는 대화식(interactive) 환경을 구축해야하기 때문에, 빠른 응답성을 보장해야 한다. 또한 각 객체에 대해 업데이트가 빈번히 발생하기 때문에 이를 효과적으로 다룰 수 있는 메커니즘이 필요하다.

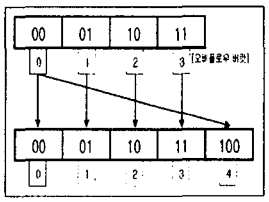
이런 환경적인 특성을 만족하기 위해 메모리 기반 DBMS를 사용하는 것이 유리하다. 메인 메모리는 CPU와 아직도 많은 속도차를 가지지만 차세대 RAM(RDRAM과 DDR-SDRAM) 등의 개발로 CPU와 메모리의 대역폭을 비슷하게 유지하고 있다.[1] 또한 가장 큰 문제점으로 지적되었던 메모리 비용이 감소하고 고성능의 메모리가 빠른 속도로 보급되고 있다.

2.2 대표적인 다차원 색인구조

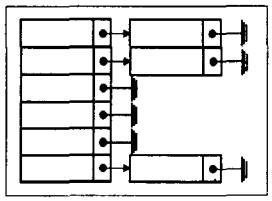
2.2.1 해시기반 다차원 인덱스

해싱에 기반을 둔 색인은 키 값으로부터 직접 객체의 위치를 계산하기 때문에 트리기반의 색인구조에 비해 일치하는 질의를 처리할 때 뛰어난 질의 성능을 보이지만 범위 질의를 처리할 때는 성능이 떨어진다.

먼저 [Fig 1]과 같은 선형 해싱(Linear Hashing)이 있다.[2] 선형 해싱은 물리적으로 인접한 공간에 데이터 페이지를 유지하기 때문에 디렉토리 없이 계산에 의해 처리하므로 페이지 어드레싱 방식을 단순하게 만들 수 있다. 그리고 데이터 페이지 분할이 필요할 때 미리 정의된 방법으로 데이터 페이지를 분할하기 때문에 높은 공간 효율을 가질 수 있다. 하지만 체인에 오버플로우가 발생하면 성능이 떨어지는 문제점을 가진다. 다음으로 [Fig 2]와 같은 고정크기의 해시 테이블을 사용하는 체인-버킷 해싱(Chained-Bucket Hashing)이 있다. 체인-버킷 해싱은 해시 테이블이 최적의 사이즈를 가질 때 매우 뛰어난 탐색 성능을 보이지만 해시 테이블이 많은 기억공간을 낭비하고, 최적 사이즈를 계산하는 것이 매우 어렵다.



[Fig 1] 선형 해싱

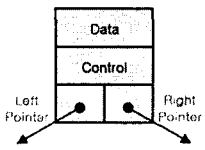


[Fig 2] 체인-버킷 해싱

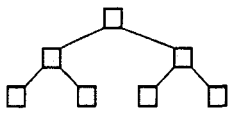
2.2.2 트리기반 다차원 인덱스

트리에 기반을 둔 색인은 객체의 위치를 정하기 위해 트리를 순회하기 때문에, 일치하는 질의를 처리할 때 해싱에 기반을 둔 색인보다 떨어지는 성능을 보인다. 그러나 트리가 정렬되어 있고 색인 인덱스를 가지기 때문에 범위 질의에서 매우 뛰어난 성능을 보인다.

트리기반 인덱스의 대표적인 구조로 균형 이진 트리인 AVL-Tree와 MMDB에서 주로 사용하는 T-Tree에 대해 살펴본다.

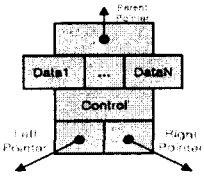


[Fig 3] AVL-Tree Node

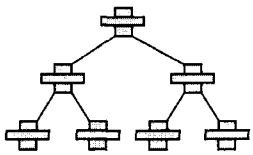


[Fig 4] AVL-Tree

[Fig 3]과 [Fig 4]는 AVL-Tree의 노드와 Tree 구조를 보여준다.[3] AVL-Tree는 이진 탐색 트리와 유사하지만 높이의 균형을 체크하여 유지한다. 트리 높이의 차수가 2이상이 되면 로테이션 연산을 수행하여 높이의 균형을 유지하지만 저장 효율이 떨어지는 문제점을 가진다.



[Fig 5] T-Tree Node



[Fig 6] T-Tree

다음으로 AVL-Tree의 단점인 저장 오버헤드를 해결한 T-Tree가 있다. [Fig 5]와 [Fig 6]은 T-Tree의 노드와 구조를 나타낸다. T-Tree는 현재 메인 메모리 데이터베이스의 색인 구조로 가장 보편화된 트리로 AVL-Tree와 유사한 구조를 가지지만 차이점으로 노드 내부의 다수의 엔트리에 값을 저장하여 각 키 엔트리의 저장 오버헤드를 줄인다.[4]

2.3 비트맵 인덱스

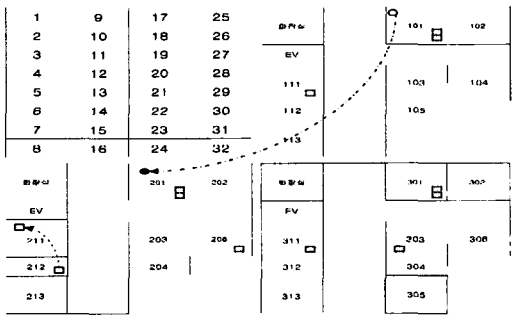
비트맵 인덱스는 해당 데이터 값이 기준에 만족하면 비트 값이 1이고, 만족하지 않으면 0을 가지고 데이터의 순서([예] 테이블의 행 또는 열)와 같은 순서로 저장하는 인덱스 구조로 테이블이 많은 행을 가지고, 키 열은 적은 카디널리티를 가질 때 사용하는 것이 유리한 구조로 비트맵 전체를 탐색해 해당 객체의 위치를 파악하는 색인구조이다.[5]

3. MMDB에서 효율적인 탐색을 위한 비트맵 인덱싱

본 절에서는 MMDB에서 효율적으로 질의할 수 있는 탐색 기법과 비트맵 인덱스 구조를 제안한다.

3.1 제안기술

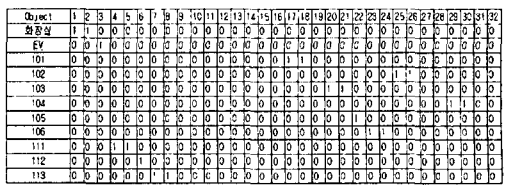
[Fig 7]은 제안하는 비트맵 인덱스를 적용하기 위한 블록과 공간정보를 표현한다. 이해를 간단히 하기 위해 [Fig 7]의 왼쪽 상단의 블록 사이즈에 각 건물구조가 대칭적으로 구성되어 있다. 대부분의 실내 환경의 객체는 사각형의 형태로 이루어져 있으며, 비슷한 블록 사이즈로 구성되어 있다. 하나의 블록 사이즈를 기준으로 두어 공간을 확대할 수 있다. 이런 식으로 구성하는 경우 각 공간을 쉽게 구성할 수 있는 장점이 있다. 단, 특수한 형태의 경우 ([예] 삼각형이나 원형구조)에는 특수한 MBR 조정 알고리즘이 필요하지만 본 논문에서는 고려하지 않는다.



[Fig 7] 층 평면도와 블록 구성도

[Fig 7]은 총 4개로 구성되어 있다. 왼쪽 상단 부분은 각 건물의 내부와 매칭되는 블록을 나타내고 있으며, 오른쪽 상단에는 건물의 1층 평면도를 나타낸다. 왼쪽 하단 부분은 2층 평면도, 오른쪽 하단 부분은 3층 평면도를 나타내고 있다. 각 평면도의 공간 정보는 비트맵 인덱스를 사용하여 유지한다. 고정 객체와 이동 객체를 하나의 인덱스에 유지하게 되면 업데이트 연산이 발생할 때 추가적인 비용이 소요되기 때문에 따로 유지한다. 마지막으로 자주 요청되는 정보는 캐싱 테이블을 생성하여 효율적인 질의를 제공할 수 있게 구성하였다.

[Fig 8]은 1층 평면도를 매칭하여 구성된 비트맵 인덱스로서 고정 객체의 정보를 담고 있다. 비트맵을 살펴보는 방법은 화장실을 예로 들어 왼쪽의 매칭 테이블과 1과 2가 매치되기 때문에 비트맵 인덱스는 1과 2의 부분에 1로 설정하고 나머지 부분은 0으로 채워진다. 다른 정보도 위와 같은 방법으로 매칭되어 비트맵 인덱스에 유지된다.

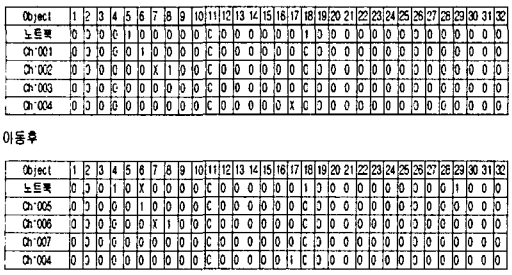


[Fig 8] 1층 평면도의 비트맵 인덱스 [고정 객체]

[Fig 8]에서 Object는 객체의 아이디를 나타내고 고정 객체인 방 정보를 나타낸다. 다음으로 1~32의 숫자는 매칭되는 공간의 영역을 나타낸다. 마지막으로 비트맵 정보가 유지되는 곳의 0은 객체를 포함되지 않음을 의미하고, 1은 객체를 포함하는 것을 의미한다.

[Fig 9]는 이동 객체의 정보를 유지한 비트맵 인덱스이다. 이동 객체는 매우 빈번히 움직이기 때문에 고정 객체와 따로 유지하는 것이 효율적이다. [Fig 9]에서 위의 비트맵 인덱스는 1층의 정보를 유지한 비트맵 인덱스이고 아래에 위치한 비트맵 인덱스는 2층의 정보를 유지한 비트맵 인덱스이다. [Fig 9]에서 Object는 객체의 아이디를 나타내고 아이디는 미리 등록을 하는 방법으로 유지한다. 고정 객체와는 다르게 이해를 돕기 위해 추가적으로 'X'는 이동 객체의 이동 전의 위치를 나타낸다.

[Fig 9]에서 보듯이 빔 프로젝트의 경우 같은 건물의 5영역(111호의 아래쪽)에서 4영역(111호의 위쪽)으로 이동했다는 것을 나타내고 있으며 객체 Ch1004의 경우 1층의 17영역(101호)에서 2층의 17영역(201호)로 이동했음을 나타내고 있다.



[Fig 9] 이동 객체를 유지한 비트맵 인덱스

마지막으로 실내 환경에서는 자주 요청되는 객체([EX] 빔프로젝트) 또는 위치([EX] 화장실)가 있다. 이런 객체나 위치를 질의시 전체 비트맵을 탐색하는데 많은 비용을 소비하기 때문에 여분의 공간을 둬 캐시 테이블을 유지하여 질의 효율을 높일 수 있다. 캐시테이블 유지정책은 필드에 카운터 값을 둬 값이 기준 값 이상이 되었을 때 추가하고 이하가 되었을 때 삭제하여 최신의 정보를 유지한다.

3.2 시나리오

논문에서 제안하는 비트맵 인덱스의 장점을 기술하기 위해 하나의 시나리오를 이용하여 탐색 기법을 설명하고, 비트맵 인덱스의 탐색 이점을 기술한다. 시나리오의 예는 다음과 같다.

노트북이 필요한 사용자가 노트북을 이용하려고 할 때, 건물에서 임대 가능한 노트북과 노트북의 위치를 찾고자 하는 경우이다. 이 때 각 층의 1호는 실습준비실로 사용가능한 노트북이 위치한다고 가정한다. [Fig 7]과 [Fig 9]를 참조한다.

① PDA에 이용 가능한 노트북을 찾기 위해 입력한다.

② 서버에서는 사용자의 요청된 질의를 SQL로 변환하여 BITMAP INDEX에서 해당하는 값을 탐색한다. 이 경우 변환된 SQL의 예는 다음과 같다. [예] 2층일 경우.

```
[SQL] Select obj from Info, object where object.장비
      ="노트북" AND Info.호수="201";
```

[결과] 비트맵의 '18' 리턴

③ 탐색된 객체의 비트맵 인덱스 값에 해당하는 메모리 주소로 이동하여 결과를 리턴한다.

: 메모리 주소에는 [건물정보, 층 정보, X좌표, Y좌표, 해당 건물 호수 등 정보 유지]

④ 결과를 리턴 받은 후 X좌표와 Y좌표를 이용하여 찾고자 하는 위치를 디스플레이 한다.

위의 질의에서 보는 것과 같이 SQL에서 많은 조인 연산을 포함한다. 조인연산은 높은 비용을 요구하기 때문에 본 논문에서는 조인연산의 문제점을 해결하기 위해 비트맵 인덱스를 사용하였다.

4. 결론

본 논문에서는 메인 메모리 기반의 DBMS를 사용한 실내 환경에서 위치 탐색시 효율적으로 적용할 수 있는 비트맵 인덱스 기법을 제안하였다. 앞으로 본 비트맵 인덱스 기법의 유효성을 증명하기 위해 성능평가를 실시해야하며, 일관적인 블록 사이즈가 아닌 다양한 블록 사이즈에 적용할 수 있도록 블록 사이즈 확장에 대한 연구가 필요하다.

참고문헌

[1] J.L. Hennessy and D.A. Paterson, Computer Architecture: A Quantitative Approach, 2nd Edition, Morgan Kaufmann, 1996.

[2] W. Litwin, "Linear Hashing: A New Tool For File and Table Addressing", Proc. Intl. Conf. on VLDB, 1980.

[3] D. Knuth, The Art of Computer Programming, Addison-Wesley, 1973.

[4] T. Lehman and M. Carey, "A Study of Index Structures for Main Memory Database Management System", Proc. Intl. Conf. on VLDB, 1986.

[5] Chee-Yong Chan, Yannis E. Ioannidis, "Bitmap index design and evaluation", ACM SIGMOD, 1998.