제21회 한국정보처리학회 춘계학술발표대회 논문집 제11권 제1호 (2004. 5)

TB-Tree를 이용한 이동 객체 궤적에 대한 조인 알고리즘

이재호*, 남광우*, 박종현*
*한국전자통신연구원 텔레매틱스연구단

e-mail : snoopy@etri.re.kr

Algorithms of Joins Using TB-Trees for Moving Object Trajectories

Jai-Ho Lee*, Kwnag-Woo Nam*, Jong-Hyun Park*
*Telematics Research Division, ETRI

요 약

이동 객체 데이터베이스 시스템에서 시공간 조인은 이동 객체들의 결합을 위한 중요한 연산이다. 시공간 조인 연산의 실행 시간은 이동 객체의 수가 증가함에 따라 기하급수적으로 증가한다. 그러므로 효과적인 시공간 조인 연산이 필수적이다. 본 논문에서는 처음으로 이동 객체의 궤적에 대한 정보를 잘 유지하고 있는 시공간 색인인 TB-Tree 를 이용한 시공간 조인에 대한 알고리즘들을 제시하고 구현한 알고리즘에 대한 실험 결과를 제시한다. 먼저 기본적인 알고리즘과 CPU 의 실행 성능 향상을 위한 알고리즘을 기술하고 이동 객체 생성기를 통해 생성된 데이터 집합에 대한 실험을 실시한 결과를 보여준다.

1. 서론

이동 객체에 대한 연구는 이동객체 데이터 모델 및 질의어에 대한 연구와 이동객체 색인에 대한 연구로 크게 구분되고 있다. 이 논문에서는 LBS 응용을 위한 기반 엔진으로 시작되었기 때문에 이동 객체들 중에 이동점(MPoint)에 초점을 맞춘다. 이동점은 일정한 선형 이동 경로를 따라 움직이는 점 객체이며 특정 시간간격별로 샘플링된 유클리드 평면상의 점이다. “어떤 시간 구간 사이에 어떤 공간 영역에 속한 객체들을 찾아라” 와 같은 시공간 질의를 효과적으로 지원하고 고성능으로 처리하기 위해서는 시공간 접근 메소드가 사용되어야 한다. 시공간 접근 메소드는 이동 객체의 현재 위치를 기반으로 미래 위치를 다루는 연구와 과거 위치를 다루는 연구로 구분될 수 있다.

이동 객체 데이터베이스에서의 질의는 두 가지 타입으로 나눌 수 있다. 첫째는 시간 연속(Time Series) 질의와 연속적인 이동 객체(Continuous Moving Objects) 질의로 나눌 수 있다. 연속적인 이동 객체질의는 slice, snapshot, project, join 질의로 구성된다. 시간 연속질의와 연속적인 이동 객체 질의 중 slice, snapshot, project 질의의 경우 한번의 데이터 접근만으로 질의 처리가 가능하다. 그러나 조인 질의의 경우 여러 번의 데이터 객체에 대한 접근이 요구되고 객체의 수에 따라 실행 시간이 기하급수적으로 늘어난다. 일반적으로 관계형 데이터베이스, 공간 데이터베이스에서도 역시 조인의 경우 여러 번의 데이터에 대한 접근이 필요하다. 이동 객체 데이터베이스에서 시공간 조인은 시공간적인 속성에 따라 두 집합의 시공간 객체들의 쌍을 만드는 것이다. 그리고 가장 중요한 시공간 조인 질의는 이동 객체의 궤적에 대한 겹침이다. 시공간 조인의 예를 들면 다음과 같다.

```
Select intersection( r.position, s.position )
From R r, S s
```

위 예제의 질의는 두 이동 객체 데이터 집합에 대

해서 궤적이 교차한 데이터 쌍을 검색하는 조인 연산이다.

이동 객체에 대한 질의는 다양하고 처리가 매우 복잡하다. 따라서 이동 객체의 특성이 잘 반영된 질의 처리 방법이 필요하다. 특히 이 논문에서는 시공간 색인을 사용하여 시공간 조인을 위한 알고리즘을 디자인하고 구현하는 문제에 대해서 초점을 맞춘다. 또한 이미 고안된 시공간 색인을 이용하여 효과적인 알고리즘을 제시하고자 한다. 시공간 조인을 수행하기 위해서 시공간 색인을 이용하는 이유는 공간 조인에서 공간 색인을 이용하는 것과 유사하다. 그래서 여러 시공간 색인 중 TB-Tree[3]를 시공간 조인을 위한 시공간 접근 방법으로써 고려하였다. TB Tree 를 시공간 색인으로 선택한 이유는 TB-Tree 의 노드에 저장된 노드 또는 엔트리(entry)가 항상 시간 순으로 배열되어 있는 구조적인 특성을 이용하면 조인 알고리즘이 간소화되고 성능을 높일 수 있다. 또한 TB-Tree 의 단말 노드(leaf node)에 저장된 속성 정보를 이용하여 객체에 직접 접근 없이 색인에 대한 접근 만으로도 질의를 수행하기 위한 일반적인 두 단계(filter step, refinement step)를 모두 수행할 수 있다.

2. 관련 연구

TB-tree 는 이동체의 위치를 이산적으로 추출하고, 추출된 이동체의 연속적인 두 위치를 선분으로 표현하고 연결된 선분들의 집합을 궤적으로 표현한다. 또한 하나의 단말 노드에 동일한 이동체의 궤적만을 저장하는 궤적 보존 정책을 사용하며, 동일한 궤적을 저장하고 있는 단말 노드간의 링크가 존재한다. 따라서 TB-tree 는 단말 노드에 동일한 이동체의 궤적만 저장하도록 궤적 번들 기법과 동일한 궤적이 저장된 단말 노드들을 링크로 연결시켜 궤적 추출에 효과적이며 복합 질의(combined Query)에 빠른 성능을 나타낸다. TB-tree 는 R-tree 의 분할 방식이 아닌, 가장 최근에 삽입된 엔트리를 새로운 노드에 두는 방식을 사용하여 높은 공간 활용도를 가진다.

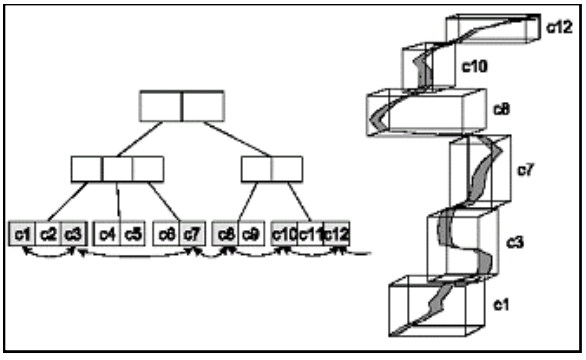

그림 1. TB-tree 구조

3. 시공간 조인

3.1 TB-Tree를 이용한 시공간 조인의 기본적인 방법

TB-Tree 를 이용한 시공간 조인의 기본적인 알고리즘은 하위 노드의 최소 경계 사각형(Minimum Bounding Box)을 형성하고 있는 입방체(cube)를 사용하는 것이다. 3 차원 입방체의 각 축은 공간 축인 X,Y 축과 시간 축인 t 축으로 이루어져 있다. 두 노드내의 엔트리(entry) Er 과 Es 의 입방체들이 공통된 교차 영역이 없다면 Er 의 하위 트리에 속한 CubeR 과 Es 의 하위 트리에 속한 CubeS 로 이루어진 (CubeR, CubeS) 의 쌍은 존재하지 않는다. 반면에 교차 영역이 존재한다면 하위 트리 중에 교차하는 입방체의 쌍이 존재한다. 이후부터는 조인의 대상으로 하는 TB-tree R 과 S 가 동일한 높이를 가진 트리라고 가정한다. 아래는 TB-Tree 를 이용한 시공간 조인의 기본적인 알고리즘이다.

```
SpatioTemporalJoin( R, S : TB_Node)    (* height of R is equal height of S *)
    SortedIntersectionTest( R,S, Seq );
    For I = 1 To Seq.legth Do
        (Er, Es) = Seq[I];
        if( Both R and S is a leaf page )
            // Refinement
            if( RealIntersectionTest(Er,Es) )
                output(Er,Es)
        Else
            ReadPage( Er.rrn );
            ReadPage( Es.rrn )
            SpatioTemporalJoin( Er.rrn, Es,rrn )
        End
    End
End
```

```
SortedIntersectionTest( Rseq, Sseq: Sequence of cube; output : sequence of pair of cube );
(*Rseq and Sseq are sorted *)
    output = null;
    I = 1; j= 1;
    While( I <= Rseq.Length  and j <= Sseq.Length ) Do
        If Rseq[I].StartTime < Sseq[I].StartTime Then
            InternalLoop( Rseq[I], j, Sseq, output )
            I = I + 1;
        Else
            InternalLoop( Sseq[I], I, Rseq, output )
            J= J + 1;
        End
    End
End
```

```
InternalLoop( t : Cube, unmarked : Cardinal, Sseq : Sequence of Cube, output : Sequence of Pair of cube )
    K = unmarked
    While( k <= Sseq.Length and Sseq[k].StartTime <= t.EndTime ) do
        If( SpatialIntersectionTest( t, Sseq[k] ) ) Then
            Append( Output, (t,Sseq[k]) )
        End
        K = K + 1
    End
End
```

위 알고리즘에서 TB_Node 는 TB-Tree 내의 노드 타입이며 각 노드는 엔트리(entry)들의 집합을 포함하고 있다. 하나의 엔트리 E 는 RRN(relative record number) 과 입방체인 MBB 로 구성되어 있다. RRN 은 하위 노드에 대한 정보이다.

TB Tree 의 구조적인 특징으로 인하여 부모 노드에 속한 자식 노드가 시간적 정렬되어 저장되어 있다. *SortedIntersectionTest* 는 이러한 특성을 이용하여 교차(intersections) 연산을 위한 일반적인 기술인 Plane sweep 방법을 적용하였다.

또한 질의 처리의 경우 색인을 통해 여과 단계(filter step)를 처리하는 것이 일반적이나 TB Tree 의 단말 노드에 저장된 orientation 정보를 활용하여 정제 단계(refinement step) 까지 처리된 결과를 얻을 수 있다. 지면 관계상 실제 궤적이 교차하는 연산(*RealIntersectionTest*)은 생략하였다.

3.2 CPU-Time 성능 향상

CPU 처리 시간은 조인 조건(ex. 두 입방체 또는 두 궤적이 교차하는지 테스트)을 연산하는데 요구되는 부동 소수점 비교 연산의 수에 비례한다. 따라서 시공간 조인함수 실행 시에 비교 횟수를 줄임으로써 CPU-Time 을 줄여 성능을 향상 시킨다.

CPU-Time 을 줄이기 위해서 노드들의 쌍들에 대해서 조인 시에 검색 영역을 제한한다[1]. 즉 *SpatioTemporalJoin2* 내에서 노드의 엔트리 중 연산에 필요한 엔트리의 수를 작게 한다. 알고리즘 *SpatioTemporalJoin* 은 하나의 노드에 속한 각각의 엔트리에 대해서 다른 노드의 모든 엔트리에 대해 조인 조건을 테스트한다.

조인 조건을 만족하는 두 노드의 엔트리를 Er, Es 라고 한자. 그러면 Er.cube ∩ Es.cube ≠ ∅ 식이 성립하고 Er.rrn, Es.rrn 이 *SpatioTemporalJoin* 의 입력으로서 호출될 것이다. 이러한 특성은 CPU-Time 을 줄이기 위한 중요한 요소를 가지고 있다.

Er.cube 과 Es.cube 이 겹치는 영역과 겹치는 Er.rrn 과 Es.rrn 의 엔트리들만이 공통적으로 겹치는 영역을 가질 수 있다. 이러한 특성을 이용하여 조인의 성능을 향상 시킨다.

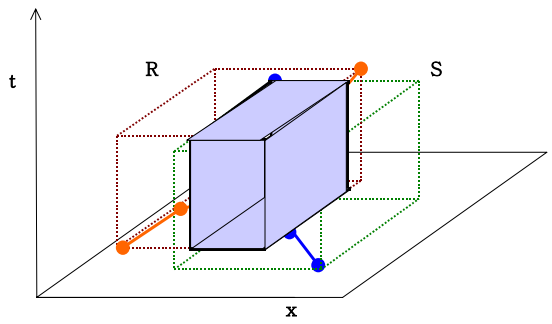

그림 2 조인의 검색 영역의 제한

그림 2에 보인 예제를 보면 알고리즘 *SpatioTemporalJoin* 은 16 번의 조인 조건 검색을 한다. 그러나 새로운 *SpatioTemporalJoin2* 알고리즘은 9 번의 조인 조건 검색을 한다. *SpatioTemporalJoin2* 알고리즘은 아래와 같다.

```
SpatioTemporalJoin2(R, S : TB_Node, cube : Cube)
    R = { Ei | (Ei ∈ R) and ( Ei.cube ^ cube <> null ) }
    S = { Ei | (Ei ∈ S) and ( Ei.cube ^ cube <> null ) }
    SortedIntersectionTest( R,S, Seq );
    For I = 1 To Seq.legth Do
        (Er, Es) = Seq[I];
        if( Both R and S is a leaf page )
            // Refinement
            if( RealIntersectionTest(Er,Es) )
                output(Er,Es)
        Else
            ReadPage( Er.rrn )
            ReadPage( Es.rrn )
            SpatioTemporalJoin2( Er.rrn, Es.rrn, Er.cube ^
                                                Es.cube)
        End
    End
End
```

4. 실험 및 비교 평가

TB-Tree 를 이용한 시공간 조인을 수행하기 위해 이동 객체들의 궤적에 대한 2 가지 집합에 대해서 고려하였다. 하나는 TB-Tree R, 다른 하나는 TB-Tree S 이다. R 과 S 사이의 시공간 조인에 대한 실행 시간을 측정함으로써 CPU-time 과 I/O-time 에 대한 실험을 하고자 한다. [1]에서의 공간 조인에 관한 성능 실험에서와 유사하게 시공간 조인의 경우 역시 조인 조건을 검사하는 연산이 매우 비용이 크다. 따라서 좋은 성능 측정 실험을 하기 위하여 디스크 접근 횟수와 조인 조건 비교 횟수 모두를 측정하는 것으로 하였다.

시공간 조인의 I/O time 에 대한 성능 실험은 일반적인 디스크의 접근 횟수로 측정하고 시공간 조인의 CPU time 은 비교 횟수로써 측정한다. 비교는 위에서 언급한 것과 같이 조인 조건의 검사(두 입방체 또는 궤적이 교차하는지 여부)를 위한 연산을 말한다.

시공간 조인의 실행 시간을 측정하기 위한 실제 데이터를 얻는 것이 매우 현실적으로 힘들다. 그래서 본 논문에서는 이동 객체 데이터 생성기 중에 City Simulator 에 의해 생성된 2 가지의 시공간 데이터 집합을 대상으로 하였다. 두 데이터 집합은 생성시 설정 옵션을 달리 하였으나 모두 1000 개의 이동 객체에 대해서 1000 번 보고한 데이터를 생성하였다. R 과 S 의 시공간 조인의 결과로 교차하는 궤적의 305 쌍이 검색되었다. 다양한 페이지 크기에 대하여 TB-Tree R 과 S 의 속성들을 아래 표로 보여준다.

Page size	M	TB-Tree R			TB-Tree S		
		높이	노드 수	데이터수	높이	노드 수	데이터수
1KB	16	5	68,273	1,009,000	5	68,273	1,009,000
2KB	33	3	31,970	1,009,000	3	31,970	1,009,000
4KB	67	3	16,245	1,009,000	3	16,245	1,009,000
8KB	136	2	8,060	1,009,000	2	8,060	1,009,000

표 1 : TB-tree R 과 S 에 대한 정보

아래 표 2 와 표 3 는 각각 위의 TB-Tree R 과 TB-Tree S 를 이용하여 *SpatioTemporalJoin* 알고리즘과 *SpatioTemporalJoin 2* 알고리즘을 테스트한 결과를 보여주고 있다. 페이지 사이즈 별 조인 조건에 대한 비교 횟수와 디스크 접근 횟수를 나타낸다. RealIntersectionTest 호출에 의해서 실제 궤적이 교차했는지에 대한 비교 횟수는 두 알고리즘 모두 252,060 번으로 동일하다.

Page size	비교횟수	디스크접근횟수
1KB	195,573,208	1,063,115
2KB	102,352,608	419,733
4KB	70,735,540	359,571
8KB	72,216,669	342,171

표 2 : *SpatioTemporalJoin* 의 디스크 접근과 비교 횟수

Page size	비교횟수	디스크접근횟수
1KB	155,644,663	1,063,115
2KB	88,991,603	419,733
4KB	57,201,341	359,571
8KB	47,681,498	342,171

표 3 : *SpatioTemporalJoin2* 의 디스크 접근과 비교 횟수

Page size	STJ	STJ2	성능향상
1KB	195,573,208	155,644,663	1.3
2KB	102,352,608	88,991,603	1.2
4KB	70,735,540	57,201,341	1.2
8KB	72,216,669	47,681,498	1.5

표 4 : 검색 영역에 대한 제약의 유무에 따른 비교

테이블 4 는 검색 영역에 대한 제약의 여부에 따른 비교 횟수를 비교한 표이다. STJ 는 *SpatioTemporalJoin* 알고리즘을, STJ2 는 *SpatioTemporalJoin2* 알고리즘을 나타낸다. 위의 표 4 의 결과를 보면 검색 영역에 대한 제약을 줌으로써 성능이 향상 되었다는 것을 알 수 있다.

5. 결론 및 향후 연구

이동 객체 데이터베이스에서 여러 질의들 중 조인 질의에 대한 처리를 위해 이동 점 객체들만의 궤적을 저장하고 있는 시공간 색인(TB-Tree)을 이용한 조인 알고리즘을 제시하였다. 기본적인 알고리즘과 CPU 성능을 향상 시키기 위하여 검색 영역을 제한하는 등의 공간 조인에서 활용되었던 기법들을 시공간 색인에 효과적으로 적용하였다. 제시한 알고리즘을 구현하고 이동 객체 생성기를 통해 생성된 데이터 집합을 이용하여 실험을 실시하였다.

향후 CPU 실행 시간 뿐만 아니라 디스크의 접근 시간을 줄이기 위한 알고리즘에 대한 연구의 진행이 필요하고 가상 데이터가 아닌 실제 데이터에 대한 실험이 요구된다. 또한 시공간 데이터의 크기가 다른 일반적인 데이터와 비교할 때 대용량의 데이터이기 때문에 더욱 효과적인 조인 처리를 위한 방법에 대한 연구가 절실히 필요하다.

참고문헌

[1] Brinkhoff , Hans-Peter Kriegel , Bernhard Seeger, Efficient processing of spatial joins using R-trees, Proceedings of the 1993 ACM SIGMOD international conference on Management of data, p.237-246, May 25-28, 1993, Washington, D.C., United States

[2] Ming-Ling Lo, Chinya V. Ravishankar: Spatial Joins Using Seeded Trees. SIGMOD Conference 1994: 209-220

[3] Dieter Pfoser, Christian S. Jensen, Yannis Theodoridis: Novel Approaches in Query Processing for Moving Object Trajectories. VLDB 2000: 395-406

[4] Preparata F. P., Shamos M. I.: 'Computational Geometry', Springer, 1988

[5] Cotelo L., J. A. Forlizzi,, L, Guting, R. H. , Nardelli, E., and Schneider, M., "Algorithms for Moving Objects Databases", FernUniversitat Hagen, Informatik-Report 289, October 2001.

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

[7] Schneider, L. M and Vazirgiannis, M., "A Foundation for Representing and Querying Moving Objects," ACM Transactions on Database Systems, Vol. 25, pp.1-42, 2000.

[8] Moreira, J., Ribeiro, C., and Abdessalem, T., "Query operations for moving objects database systems," ACM-GIS Conference, pp.108-114, 2000.