TB-Tree 를 이용한 이동객체 조인 알고리즘

이재호*, 이성호*
*전자통신연구원
e-mail : snoopy@etri.re.kr

Moving Objects Join Algorithms using TB-Tree

Jai-Ho Lee*, Seong-Ho Lee*
*Electronics and Telecommunications Research Institute

요 약

이동 객체 데이터베이스 시스템에서 시공간 조인 연산은 이동 객체들의 결합을 위한 중요한 연산이며 수행 시간은 이동 객체의 수가 증가함에 따라 기하급수적으로 증가한다. 그러므로 효과적인 시공간 조인 연산이 필수적이다. 본 논문에서는 기존의 공간 조인에서 활용되었던 기법들을 이동객체 조인에 적용하였다. 이동 객체의 궤적에 대한 정보를 잘 유지하고 있는 시공간 색인인 TB-Tree 를 이용한 깊이 우선 탐색 기반과 넓이 우선 탐색 기반 TB-Tree 조인에 대한 알고리즘들을 제시하고 구현한 알고리즘들의 성능 비교한 실험 결과를 제시한다.

1. 서론

도로 위의 차량이나 전장의 군인과 같은 이동 객체들을 관리하는 이동 객체 데이터베이스 시스템(moving objects database systems)은 매우 활발히 연구되고 있으며, 위치 기반 서비스(LBS:Location Based Services), 텔레매틱스 등의 다양한 응용 분야가 있다. 특히 유비쿼트스 환경에서는 많은 수의 이동 객체들을 효과적으로 관리하는 이동 객체 데이터베이스의 기술들의 중요성이 계속 증가하고 있다.

이동 객체에 대한 연구는 이동객체 데이터 모델 및 질의어에 대한 연구와 이동객체 색인에 대한 연구로 크게 구분되고 있다. 이 논문에서는 LBS 응용을 위한 기반 엔진으로부터 연구가 시작되었기 때문에 이동 객체들 중에 이동점(MPoint)[4]을 대상으로 하고 과거 궤적에 대한 질의와 관련된 조인 알고리즘에 대해 초점을 둔다.

이동객체 조인은 시공간적인 속성에 따라 두 집합의 시공간 객체들의 쌍을 만드는 것으로서, 이동 객체 데이터베이스 시스템에서 주요질의 중 하나이다. 예로는, “어제 하루 동안 서로간의 거리가 10 미터이내의 친구들의 쌍을 구하라”이 있다.

이동 객체 조인은 여러 번의 데이터 접근을 요구하는 연산이므로 많은 디스크 I/O 와 처리 시간이 필요하므로 이를 효과적으로 처리하는 것이 필요하다. 그러나 지금까지 이동객체 조인에 대한 연구가 이루어지지 않았다.

조인연산의 대상이 되는 두 집합들에 시공간 색인이 구축되어 있으면 색인을 활용함으로써 더욱 효과적인 조인 연산을 처리할 수 있다. 본 논문에서는 두 개의 집합이 모두 색인을 가지고 있을 경우를 대상으로 하며 이때 사용되는 색인은 과거 시공간 색인인 TB-Tree 를 사용한다. 그리고 기존 개발된 공간 조인 알고리즘들을 살펴보고, 이들을 시공간 조인으로 확장하는 방법에 대해서 기술한다.

2. 관련 연구

두 개의 색인을 사용하는 공간 조인 알고리즘으로는 깊이 우선 탐색 R-Tree 공간 조인(depth-first traversal R-tree spatial join) 알고리즘[2]과 넓이우선 탐색 R-Tree 공간 조인(breadth-first traversal R-tree spatial join) 알고리즘, 그리고 변환공간 뷰 공간 조인(transform-space view based spatial join) 알고리즘이 있다.

깊이 우선 탐색 R 트리 공간 조인은 깊이 우선 탐색을 통해 서로 겹치는 단말 엔트리의 쌍을 찾는다. 이때, 두 비단말 노드에서 겹침이 발생하지 않으면, 그 두 노드에 재귀적으로 연결된 자식 노드들 사이에서도 겹침이 발생하지 않음을 활용하여, 서로 겹치는 노드의 쌍들만을 탐색함으로써 비교 대상을 줄인다.

그리고 같은 노드들이 디스크에서 여러 번 읽히는 것을 방지하기 위해 local plane sweeping 과 pinning 과 같은 휴리스틱 방법들을 사용한다. 그러나 이들 휴리스틱 방법들은 조인 대상인 전체 노드들의 액세스 순서는 고려하지 않고 겹침 관계에 있는 단 하나의 비단말 노드 쌍이 가리키는 자식 노드들의 액세스 순서만을 고려하므로, 전역 최적값(global optimum)을 찾지 못하는 단점을 가진다.

넓이우선 탐색 R-Tree 공간 조인 알고리즘에서는 넓이우선 탐색을 통해 겹침 관계에 있는 노드 쌍들의 정보를 중간 조인 색인(IJI: intermediate join index)에 저장하고, IJI 내의 모든 조인 노드 후보들의 액세스 순서를 고려하므로 전역적 최적 조인 시퀀스를 생성한다. 그러나, 이동 객체 데이터베이스와 같이 겹침 관계에 있는 노드 쌍들이 많다면, IJI 의 크기가 지수적으로 증가하게 되어 오히려 깊이 우선 탐색 R 트리 공간 조인보다도 성능이 현저하게 떨어진다.

변환공간 뷰 공간 조인 알고리즘은 한 쪽 트리 R 의 단말 노드들을 공간상에서 인접한 순서대로 정렬하고, 정렬된 단말 노드들을 한 번에 하나씩 차례대로 다른 트리 S 에 영역 질의를 수행한다. 이렇게 함으로써, 이전 질의에서 액세스된 노드들이 다음 질의에도 버퍼에서 최대한 존재하도록 하여 버퍼 활용률을 향상시킨다. 이때, 트리 R 의 단말 노드들의 정렬은 변환공간에서 공간 채움 곡선을 사용하여 이뤄지는데, 정렬시에 색인을 활용함으로써 정렬 비용을 최소화한다. 이 방법은 가장 최근에 제안된 것으로서 가장 성능이 우수한 것으로 알려져 있다. 그러나, 공간 색인에만 적용이 되므로 이에 대한 확장 방법이 필요하다.

3. 깊이 우선 탐색 TB-Tree 조인 알고리즘

깊이우선 탐색 TB-Tree 조인 알고리즘은 기존의 R 트리에서 개발된 깊이 우선 탐색 조인 알고리즘을 TB-Tree 에 맞게 확장한 것이다.

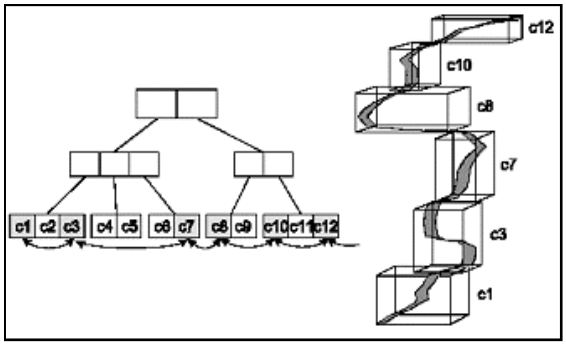

그림 1. TB-tree 구조

그림 1은 TB-Tree 의 구조를 나타낸다. 그림에서 보듯이 하나의 궤적은 여러 개의 세그먼트로 분리되어 저장된다. 그리고, 근사 포함 박스 내에서 세그먼트의 방향에 대한 정보를 추가로 포함하고 있어서 정확한 시공간 정보를 색인 내에서 유지하고 있다.[1]

이러한 TB-Tree 에 대한 깊이 우선 탐색 TB-Tree 조인 알고리즘의 기본 아이디어는 다음과 같다. 두 개의 디렉토리 엔트리들의 3D 포함 박스가 겹치지 않는다면, 그 둘 사이에는 어떠한 조인 쌍도 발생하지 않는다는 것이다. 이렇게 함으로써 트리의 상위 레벨에서 조인 가능성이 없는 조인쌍들은 검색하지 않음으로써 성능을 향상시킨다.

```
SpatioTemporalJoin1( 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 )
                output(Er,Es)
            Else
                ReadPage( Er.rrn );ReadPage( Es.rrn )
                SpatioTemporalJoin1( Er.rrn, Es,rrn )
            End
        End
End
SortedIntersectionTest( Rseq, Sseq: Sequence of mbb;
                    var output : sequence of pair of MBB );
        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
```

STJ1(SpatioTemporalJoin1)은 깊이 우선 탐색 기반 TB-Tree 조인 알고리즘을 나타낸다. TB-Tree 의 노드 내의 엔트리(Entry)들이 시간적인 순서대로 정렬되어 있는 특성을 가지고 있다. 알고리즘을 살펴보면, 트리 R 과 S 의 두 노드를 입력으로 받아 local plane sweeping 기법을 적용하여 조인 쌍들을 결과로서 반환한다. 즉, R 의 모든 엔트리 Er 과 S 의 모든 엔트리 Es 들이 시간 상으로 정렬되어 있으므로 시간 복잡도를 줄일 수 있다.

이외에도 CPU 시간 및 IO 시간을 줄이기 위해 최적화를 수행하기 위해 사용된 기법으로는 탐색 공간 제약과 Local plane-sweep order with pinning 기법이 있다.

탐색 공간 제약이란 STJ1 의 알고리즘의 경우 한 쌍의 노드가 주어질 때, 각 노드의 엔트리수의 곱만큼의 비교가 필요하다. 그러나, R 의 모든 엔트리들에 대해서 S 의 모든 엔트리들을 비교하기 전에 R 의 각 엔트리들이 S 자체와 겹침이 발생하는지 먼저 검사하고, S 의 모든 엔트리들에 대해서도 동일한 작업을 수행한다면, 비교하는 횟수를 굉장히 많이 줄일 수 있다.

그림 2 에 보인 예제를 보면 알고리즘 STJ1 은 R 영역내의 4 개의 궤적과 S 영역 내의 4 개의 궤적 모두가 연산의 대상이 된다. 그러나 새로운 STJ2(SpatioTemporalJoin2) 알고리즘은 R 영역내의 3 개의 궤적과 S 영역내의 3 개의 궤적이 연산의 대상이 됨에 따라 결과적으로 조인 조건에 대한 비교 연산의 횟수가 감소 하게 된다. 그림에서 보듯이 탐색 공간 제약을 적용하지 않는 경우 카티션 프로덕트만큼의 비교횟수가 발생하는 반면, 탐색 공간 제약을 적용하는 경우 비교 횟수가 상당히 줄어듦을 알 수 있다.

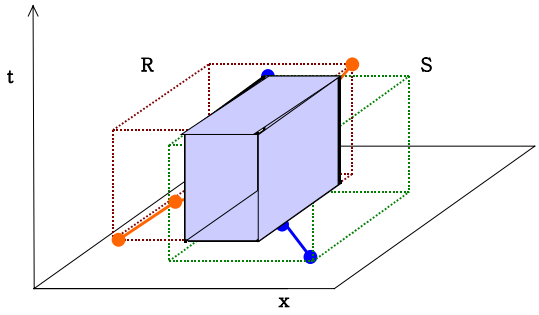

그림 2 조인의 검색 영역의 제한

```
SpatioTemporalJoin2( R, S : Node,
                     mbb : cube_Parallelepiped   )
   R = { Ei | (Ei ∈ R) and ( Ei.mbb ^ mbb <> null ) }
   S = { Ei | (Ei ∈ S) and ( Ei.mbb ^ mbb <> 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 )
           output (Er,Es)
       Else
           ReadPage( Er.ref );ReadPage( Es.ref )
           SpatioTemporalJoin2(  Er.rrn,  Es,rrn,  Er.mbb  ^
Es.mbb)
       End
   End
End
```

Local plane-sweep order with pinning 기법의 개념은 다음과 같다. 평면 스위핑에 의해 엔트리의 쌍 (Er,Es)의 순서를 결정한다. 첫번째 엔트리의 쌍에 대해서 Er.rrn 의 서브 트리와 Es.rrn 의 서브 트리에 대해서 조인 연산을 수행한 다음 두 엔트리의 degree 를 계산한다. 엔트리 E 의 입방체에 대한 degree 는 최소 경계 사각형 E.mbb 와 다른 트리의 아직 처리되지 않은 엔트리의 최소 경계 사각형 사이의 교차 영역이 존재하는 수를 나타낸다. 가장 큰 degree 를 가지는 엔트리를 메모리상에 유지하고 이 엔트리와 교차하는 다른 엔트리들과의 시공간 조인 연산을 수행한다. 두 엔트리의 degree 가 모두 0 일 경우 다시 평면 스위핑 순서에 따라 다음 엔트리의 쌍에 대해서 조인 연산을 수행한다.

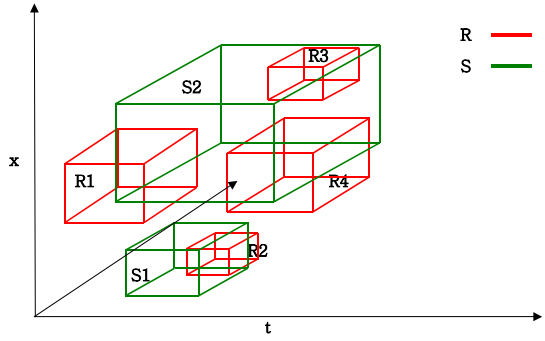

그림 3 시공간 조인 상황의 예제

그림 3의 예제에서 R1 과 S2 의 조인 연산이 수행된 후 각각의 degree 를 계산하면 degree(R1) = 0 와 degree(S2) = 2 라는 결과를 얻을 수 있다. 따라서 S2 와 R3, R4 에 대해서 다시 조인 연산을 수행하게 된다. <R1, S2, R4, R3, S1, R2>의 순서로 디스크에 접근하게 된다. 이와 같이 엔트리의 degree 를 이용한 아이디어는 이미 0에서 소개되었다.

아래의 STJ3(SpatioTemporalJoin3)는 위 단락의 내용을 구체적인 알고리즘으로써 기술하고 있다. 시공간 조인 시 많이 요구되는 페이지를 메모리 상에 유지함으로써 동일에 페이지에 대한 반복적인 접근을 제거하여 성능을 향상하였다.

```
SpatioTemporalJoin3(    R,    S    :    Node,    mbb    :
    Cube_Parallelepiped   )
R = { Ei | (Ei ∈ R) and ( Ei.mbb ^ mbb <> null ) }
S = { Ei | (Ei ∈ S) and ( Ei.mbb ^ mbb <> null ) }
SortedIntersectionTest( R,S, Seq );

Do
    (Er, Es) = Seq[0];
    if( Both R and S is a leaf page )
        output(Er,Es)
    Else
        ReadPage( Er.rrn );   ReadPage( Es.rrn )
        SpatioTemporalJoin( Er.rrn, Es,rrn, Er.mbb ^ Es.mbb)
    End
    Seq.Remove(0);
    DegR = Degree( Er );          DegS = Degree( Es );

    If( DegR = 0 and DegS = 0 )
            // go to first of loop
    Else If( DegR >= DegS )
        SeqIntersectR = { (Ex,Ey) | (Ex,Ey) ∈ Seq and ( Er =
Ex ) or (Er = Ey) }
        For I = 1 To SeqIntersectR.length Do
            (Ex, Ey) = SeqIntersectR[I];
            SpatioTemporalJoin3(  Ex.rrn,  Ey.rrn,  Ex.mbb  ^
Ey.mbb)
            Seq.Remove( Ex,Ey );
        End
    Else
        SeqIntersectS = { (Ex,Ey) | (Ex,Ey) ∈ Seq and ( Es =
Ex ) or (Es = Ey) }
        For I = 1 To SeqIntersectS.length Do
            (Ex, Ey) = SeqIntersectS[I];
            SpatioTemporalJoin3(  Ex.rrn,  Ey.rrn,  Ex.mbb  ^
Ey.mbb)
            Seq.Remove( Ex,Ey );
        End
    End
While( Seq is not Empty )
End
```

4. 넓이우선 탐색 TB-Tree 조인 알고리즘

넓이 우선 탐색 TB-Tree 조인 알고리즘은 기존의 R-Tree 공간 조인 알고리즘을 확장한 형태이다. 이 알고리즘은 기존의 깊이 우선 탐색 조인 알고리즘이 탐색 공간 제약과 plane sweeping 과 같은 휴리스틱에 기반한 방법이므로 지역적 최적화밖에 되지 않는다는 문제점을 해결하는 전역적 최적화 방법으로서 제안되었다.

STJ4(SpatioTemporalJoin4)는 넓이우선 탐색 기반 TB-tree 조인 알고리즘을 나타낸다. 입력으로는 두개의 TB-tree R, S 를 입력으로 받고, 출력으로는 조인쌍을 출력한다. 사용되는 자료구조로는 중간 조인 색인(intermediate join index: IJI)를 사용한다. 먼저, 두 개의 루트 노드에 대해서 SortedIntersectionTest 를 호출하여 조인 쌍들을 IJI[0]에 기록한다. 그리고, 자식

노드들로 내려가면서, IJI[i]번째에 있는 조인 쌍들을 이용하여 다음 조인 쌍들을 찾음으로써 전역적으로 조인 쌍들을 고려하게 된다.

```
SpatioTemporalJoin4( R, S : Node)
// R,S are two TB-trees, hR= hS
// IJI[i] intermediate join index at level i

    IJI = null;
    IJI[0] = SortedIntersectionTest( R,S );
    Int i=0;
    While( i < hR-1 do )
        foreach <r_node,s_node> in IJI[i] Do
            IJI[i+1] = IJI[i+1]∪
                    SortedIntersectionTest( r_node,s_node )
        End
        i=i+1
    End
    output IJI[i]
End
```

5. 성능 비교

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 처리 시간과 I/O 처리 시간에 대한 실험을 하고자 한다. [2]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 처리 시간에 대한 성능 실험은 일반적인 디스크의 접근 횟수로 측정하고 시공간 조인의 CPU 처리 시간은 비교 횟수로써 측정한다. 비교는 위에서 언급한 것과 같이 조인 조건의 검사를 위한 연산을 말한다. 조인 조건에 대한 검사는 두 노드의 최소 경계 사각형(MBB : Minimum Bounding Box)이 교차하는지 여부에 대한 검사이다.

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.

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

표 1 TB-tree R 과 S 에 대한 정보

표 2는 검색 영역에 대한 제한의 여부에 따른 비교 횟수를 비교한 표이다. 각 페이지 크기 별 조인 조건의 비교 횟수를 보여 준다. 표 2의 결과를 보면 검색 영역에 대한 제약을 줌으로써 모두 성능이 향상되었다는 것을 알 수 있다.

Page size	STJ1	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 2 조인 조건 비교 횟수의 비교

아래 표 3는 STJ1, STJ2 와 STJ3 알고리즘의 디스크 접근 회수에 대한 실험 결과를 보여 준다. 결과를 보면 STJ3 알고리즘에 의해서 디스크 접근 횟수가 상당히 줄어든 것을 알 수 있다.

Page size	STJ1, STJ2	STJ3	성능향상
1KB	2,126,230	1,219,160	1.7
2KB	839,466	512,915	1.6
4KB	719,142	427,169	1.6
8KB	684,342	387,041	1.7

표 3 STJ1,STJ2, STJ3 알고리즘의 디스크 접근 횟수

6. 결론 및 향후 연구

이동 객체 데이터베이스에서 많은 디스크 입출력과 처리 시간을 필요로 하는 조인 연산에 대한 효과적인 처리가 필요하다. 그러나 지금까지 이동객체 조인에 대한 연구는 이루어지지 않았다.

본 논문에서는 시공간 색인(TB-Tree)을 이용한 조인에 대해 연구하였다. 기존의 공간 조인에서 활용되었던 기법들을 이동 객체 조인에 적용하였다. 깊이 우선 탐색 기반 TB-Tree 조인과 넓이 우선 탐색기반 TB-Tree 조인 알고리즘을 제시하였고 각 알고리즘들의 성능 비교한 결과를 보였다.

대용량의 데이터이기 때문에 향후 좀더 효과적이고 최적화된 조인 알고리즘에 대한 연구의 진행이 필요하고 가상 데이터가 아닌 실제 데이터에 대한 실험이 요구된다.

참고문헌

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

[2] 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

[3] Farshad Fotouhi, Sakti Pramanik: Optimal Secondary Storage Access Sequence for Performing Relational Join. IEEE Trans. Knowl. Data Eng. 1(3): 318-328 (1989)

[4] Martin Erwig, Ralf Hartmut Güting, Markus Schneider, Michalis Vazirgiannis: Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. GeoInformatica 3(3): 269-296 (1999)