공간 조인의 성능 향상을 위한 동적 그리드 공간 분할 알고리즘

박 재 형*, 정 병 수
경희대학교 전자계산공학과

Dynamic Grid Space Partition Algorithms for Improve the Performance of Spatial Joins

Jae Hyung Park*, Byung Soo Jeong
Dept. of Computer Science, Kyunghee University

요 약

지리 정보 시스템에서 공간 객체들은 크기가 가변적이고 객체의 형태가 일정하지 않을 뿐만 아니라 공간 객체의 분포 또한 일정하지 않다. 공간 조인은 이러한 공간 객체들의 특성으로 인해 비용이 많이 들고, 공간 객체의 분포에 따라 특정 영역에서의 공간 조인 비용이 많이 들 수 있다. 이 논문에서는 공간 객체들의 분포에 따라 한 번의 Disk 접근으로 공간 객체들을 적재할 수 있는 크기로 셀을 동적 분할하는 알고리즘을 제안한다. 제안된 알고리즘을 수행한 후에 생성된 다양한 크기의 셀을 기반으로 공간 조인을 수행한다. 또한 정제 단계에서 공간 객체를 메모리로 적재하는 Disk I/O를 줄이기 위한 방법도 알아본다.

1. 서론

지리 정보 시스템이나 컴퓨터 지원 설계에서 공간 분석을 위한 중요한 연산중의 하나인 공간 조인은 대상이 되는 공간 객체의 수가 많고, 공간 객체의 분포가 다양하며, 객체의 크기가 가변적이다. 또한 공간 조인은 조인 연산에 드는 비용뿐만 아니라 조인 연산에 필요한 공간 객체를 적재하는 비용도 상당히 많이 든다. 공간 객체들에 대한 효율적인 공간 조인을 위한 여러 가지 방법들이 연구되었다[1,2,5,6,7]. 이 논문에서는 공간 조인을 수행할 전체 영역을 한 번의 Disk 접근으로 적재할 수 있는 가변 크기의 셀(cell)로 나누는 동적 영역 분할 기법을 제안하였다 제안된 기법은 공간 객체의 분포에 따라 셀의 분할을 수행함으로서 전체적으로 Disk I/O 횟수를 감소시키고 조인 연산 시간을 줄일 수 있는 다양한 크기의 셀을 정의한다. 이 논문에서는 제안한 동적 영역 분할 기법을 이용하여 여과 단계에서 전체 영역을 한 번의 Disk 접근으로 적재할 수 있는 단위인 셀(cell)로 분할하고 각 셀을 이용하여 Disk I/O를 줄일 수 있는 공간 조인 연산을 수행한다. 또한 정제 단계에서 Disk에서 메모리로 공간 객체를 적재할 때 Disk I/O 수를 줄이기 위한 방법을 알아본다

2. 관련연구

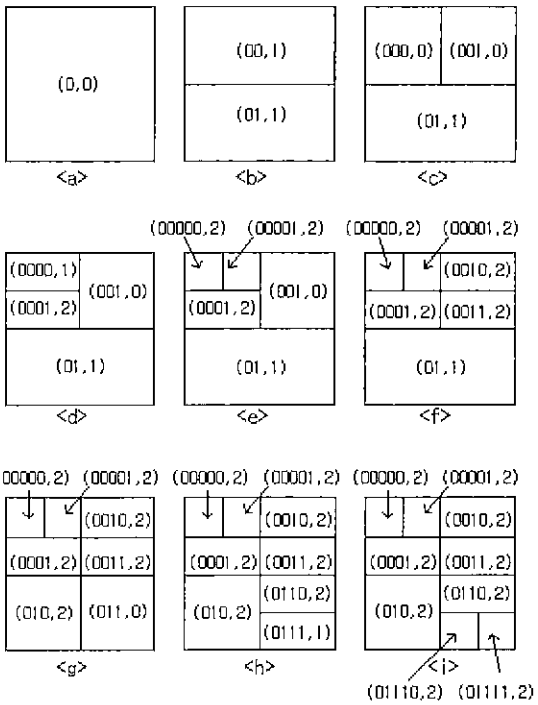
[6]에서는 공간 객체가 어느 특정 셀에 몰려있을 경우 해당되는 특정 셀에 대해 공간 조인을 수행하는 비용이 많이 들기 때문에 균형적인 공간 객체들의 적재를 위해 전체 영역을 일정한 크기의 작은 tile들로 잘게 쪼개어 round robin이나 hashing 기법을 이용하여 tile들을 각각의 파티션에 할당한다. 그러나 이러한 방법은 전체 영역을 작은 tile들로 나누어 각 파티션에 할당하기 때문에 굉장히 많은 수의 공간 객체가 각각의 파티션에 중복 저장되게 된다. 그러므로 이것은 공간 조인 연산과 Disk I/O에 상당한 영향을 미치게 된다. [3]에서는 공간 데이터에 효율적인 집합적 접근을 지원하기 위한 기법을 제안하였는데 scene단위로 공간적 접근을 처리한다. 이것은 R*-tree를 사용하여 공간적으로 인접한 객체들의 집합을 물리적으로 연속된 저장 장소에 매핑하는 방법이다. [4]에서는 하나의 그리드 셀이 가리키는 데이터 페이지가 하나의 I/O 단위이다. 여기에서는 이전에 사용되었던 질의에 따라 클러스터를 구성한다. 그러나 이 방법은 하나의 그리드 셀에 대해 질의 속도를 빠르게 하기 위해 셀을 분할하고 합병할 경우에 다른 그리드 셀에 대해서는 분할 및 합병을 수행하기 이전보다 더 좋지 못한 결과를 초래할 수 있다는 문제점이 있다

3. 동적 영역 분할 기법을 이용한 공간 조인

기존의 연구들에서 공간 조인에서의 클러스터링은 고려되지 않았다. 여기에서는 전체 영역을 한 번의 Disk 접근으로 가져

올 수 있는 클러스터의 크기로 셀을 구성하는 동적 영역 분할 기법을 제안한다 이것은 다른 셀에 영향을 미치지 않고 분할이 필요한 셀들만 분할하는 방법을 취한다. 분할의 기준은 셀 내의 있는 모든 공간 객체들의 MBR(Minimum Bounding Rectangle)의 면적의 합이다. 즉, 셀 내에 있는 모든 공간 객체들의 MBR의 면적의 합이 시스템에 따라 정의된 임계값보다 작으면 한번의 Disk 접근으로 가져올 수 있는 단위로 간주하고 더 이상 분할하지 않지만 셀 내의 모든 공간 객체들의 MBR의 면적의 합이 임계값보다 크면 셀을 분할하고, 분할된 셀들 내의 각 공간 객체들의 MBR의 면적의 합이 임계값보다 작아질 때까지 재귀적으로 셀 분할을 수행한다. 각 셀들의 id(cid)와 분할 축(flag)을 지정하기 위해 <cid, flag> 쌍을 정의한다. flag는 분할 축 지정뿐만 아니라 분할을 수행할지의 여부도 함께 나타낸다 셀 id(cid)의 최대 비트 스트링 길이를 지정하기 위해 전역 변수인 mbs(maximum bit string length)를 정의한다

flag가 0이었다면 수평 분할이 이루어지고, 분할되어 새로 생성된 상위 셀과 하위 셀에 대해 cid를 수정하고 flag switching을 수행한 후에 각 셀에 대해 다시 재귀적으로 분할이 이루어진다 flag가 1이었다면 수직 분할이 이루어지고, 분할되어 생성된 왼쪽 셀과 오른쪽 셀에 대해 cid 수정과 flag switching이 이루어진 후, 각 셀에 대해 재귀적 분할이 이루어진다. 결국 모든 셀이 한 번의 디스크 접근으로 적재될 수 있는 크기로 분할된다.



< 그림 1. 동적 공간 분할 기법의 예 >

```
static int mbs = 1;
Partition(cid, flag)
{
    if (All_Object_MBR_Area(cid) > threshold) {
        if ( flag == 0 ) {
            horizontal division.                // 수평 분할.
            if (created new cell == upper cell) {  // 상위 셀 처리
                cid = cid left-shift with 0; // 상위 셀의 cid 지정
                flag = 1;                   // flag switching.
                Partition(cid, flag).       // 상위 셀에 대해
            }                               // 재귀 호출.
            else {                          // 하위 셀 처리.
                cid = cid left-shift with 1;
                flag = 1;
                Partition(cid, flag).
            }
        }
        if ( flag == 1) {
            vertical division;                  // 수직 분할.
            if (created new cell == left cell) { // 왼쪽 셀 처리
                cid = cid left-shift with 0, .
                flag = 0;
                Partition(cid, flag);
            }
            else {                              // 오른쪽 셀 처리.
                cid = cid left-shift with 1;
                flag = 0;
                Partition(cid, flag).
            }
        }
    }
    else {
            flag = 2;
            if ( bit_string_length(cid) > mbs )
               mbs = bit_string_length(cid);
    }
    return;
}
```

< 표 1. 동적 공간 분할 알고리즘의 의사 코드>

셀 경계선에 걸쳐있는 객체는 중복 저장한다 공간 분할의 예가 <그림 1>에 기술되었고, 동적 영역 분할 알고리즘의 의사 코드가 <표 1>에 기술되었다.

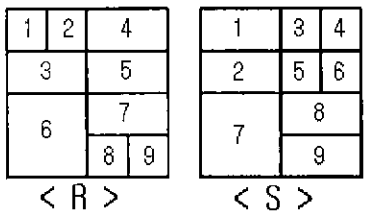
전체 영역이 한번의 Disk I/O로 access될 수 있는 셀로 모두 분할된 후 공간 조인을 수행할 셀 id의 구분을 위해 각 셀의 cid를 재순서화한다. 재순서화 방법은 각 셀 cid의 bit string length를 mbs와 비교하여 mbs와 같으면 십진수로 치환하고, 셀 cid의 bit string length가 mbs보다 작으면 그 차(mbs - cid's bit string length)만큼 cid의 bit string에 0을 삽입한 후에 십진수로 치환하여 셀 들을 정렬하고 오름차순으로 cid를 1

부터 재순서화한다. <그림 1>의 최종 분할 영역인 i번째 공간 영역에 대한 재순서화의 예가 <표 2>에 예시되었다.

cid(bit 수)	mbs(5) - bit 수	수정된 cid	재정렬된 cid
00000 (5)	0	00000	1
00001 (5)	0	00001	2
0001 (4)	1	00010	3
0010 (4)	1	00100	4
0011 (4)	1	00110	5
010 (3)	2	01000	6
0110 (4)	1	01100	7
01110 (5)	0	01110	8
01111 (5)	0	01111	9

< 표 2. (그림 1)의 i번째 공간 영역에 대한 cid 재순서화 >

전체 영역의 분할이 완료되고, cid가 순서대로 정렬된 후에 두 공간 객체 집합 R과 S의 공간 조인이 수행된다. <그림 2>의 두 공간 객체 집합에서 집합 R과 집합 S의 공간 조인을 수행하는 셀의 쌍은 <R_1, S_1>, <R_2, S_1>, <R_3, S_2>, <R_4, S_3>, <R_4, S_4>, <R_5, S_5>, <R_5, S_6>, <R_6, S_7>, <R_7, S_8>, <R_8, S_9>, <R_9, S_9>의 총 11번의 조인 연산만이 필요로 된다. 이것은 기존에 제시되었던 공간 조인 알고리즘보다 적은 횟수의 연산을 수행을 수행하고, 적은 Disk I/O를 가져오며 특히, 셀 <R_3, S_2>, <R_6, S_7>, <R_7, S_8>쌍의 경우 매우 좋은 성능을 보여준다.



<그림 2. 공간 분할을 수행한 후의 두 공간 객체 집합>

4. 정제 단계에서 Disk I/O를 줄이기 위한 기법

영역 분할 단계에서 셀의 경계면에 걸쳐있는 공간 객체를 중복 저장하였기 때문에 여과단계에서 공간 조인을 수행한 후의 결과물인 후보 객체 쌍들은 중복된 값을 가질 수 있다. 또한 여과 단계에서 공간 객체의 MBR을 사용하여 조인 연산을 수행하였기 때문에 정제 단계에서 중복되는 후보 쌍을 제거하고 실제 공간 객체를 사용하여 조인 속성을 만족하는지를 조사해야만 한다. 이 논문에서는 공간 객체를 메모리로 적재하는 비용을 줄이기 위해 정렬키 스위칭 기법을 이용한다[8]. 정렬키 스위칭 기법은 조인을 수행할 두 집합 중에서 하나의 집합에 중점을 두고, 공간 객체를 버퍼 캐쉬로 적재하고, 처리를 끝내면 다음 집합을 중심으로 공간 객체를 적재하는 방법이다. 정렬키 스위칭 기법의 예를 <표 3>에 기술하였다.

여과 단계에서 산출된 후보 공간 객체쌍
(r1, s1)
(r3, s1)
(r1, s2)
(r5, s1)
(r1, s3)
(r7, s1)
(r2, s5)
(r2, s2)
(r2, s3)
(r2, s4)
(r4, s4)
(r6, s4)

후보쌍	캐쉬내의 내용	적재되는 객체수
(r1,s1)	(r1,s1)	2
(r1,s2)	(s1, r1, s2)	1
(r1,s3)	(s1, s2, r1, s3)	1
정렬키 S로 변경		
(r3, s1)	(s2, s3, s1, r3)	1
(r5, s1)	(s3, r3, s1, r5)	1
(r7, s1)	(r3, r5, s1, s7)	1
정렬키 R로 변경		
(r2, s2)	(r5, r7, r2, s2)	2
(r2, s3)	(r7, s2, r2, s3)	1
(r2, s4)	(s2, s3, r2, s4)	1
(r2, s5)	(s3, s4, r2, s5)	1
정렬키 S로 변경		
(r4, s4)	(s3, s5, s4, r4)	1
(r6, s4)	(s5, r4, s4, r6)	1

< 표 3. 정렬키 스위칭 기법 >

5. 결론

본 논문에서는 한 번의 Disk 접근으로 메모리로 적재할 수 있는 크기로 셀을 동적으로 분할하는 알고리즘을 제안하였다. 제안된 알고리즘은 균등 분할을 수행하는 기존의 방법보다 공간 조인의 연산 비용과 Disk I/O를 줄임으로서 전체 공간 조인의 비용을 줄이는데 기여하였다. 향후 연구 과제로는 cid의 재순서화를 수행하지 않고도 cid만으로 조인할 셀을 찾는 방법과 공간 객체들의 다양한 분포에 대한 다양한 성능 평가를 수행하는 것이다.

6. 참고문헌

[1] A Guttman "R-trees : a dynamic index structure for spatial searching" In Proceeding of ACM-SIGMOND International Conference on Management of Data, Aug. 1984.

[2] T Brinkhoff, H P Kriegel, and B Seeger "Efficient Processing of Spatial Joins Using R-trees". In Proceedings of ACM-SIGMOND Conference Washington, DC, May 1993

[3] T. Brinkhoff, H. P. Kriegel, R. Schneider, "Scene Organization - A Technique for Global Clustering in Spatial Database Systems", Technical report 9322, University of Munich, 1993.

[4] Gisbert Droege, Hans-Jorg Scheck "Query-adaptive data space partitioning using variable-size storage clusters", Advances in Spatial Databases, Springer Verlag, 1993

[5] T Brinkhoff, H P Kriegel, R Schneider, B. Seeger "Multi-step Processing of Spatial Joins". In Proceedings of the 1994 ACM-SIGMOND Conference, Minneapolis, May 1994.

[6] J. M. Patel, D. J. DeWitt, " Partition based spatial-merge join" In Proceeding of the 1996 ACM-SIGMOND Conference, Montreal, Canada, June 1996

[7] M. L. Lo, C. V. Ravishankar. "Spatial Hash-Joins" In Proceedings of the 1996 ACM-SIGMOND Conference, Montreal, Canada, June 1996

[8] D. J. Abel, V. Gaede, R A Power, X. Zhou, "Resequencing and Clustering to Improve the Performance of Spatial Joins", ISS-37, December 1996.