병렬처리 대용량 공간자료구조의 연구

방갑산
한성대학교 정보시스템공학과
e-mail:ksbang@hansung.ac.kr

A Study on Parallel Spatial Index Structure Development for Large Data

Kap-San Bang
Dept of Information Engineering, Hansung University

요 약

공간 데이터의 효율적인 처리는 현대의 멀티미디어 데이터베이스에 있어서 대단히 중요한 역할을 하고 있다. 많은 응용분야에서 방대한 양의 공간 데이터는 보조기억장치(예: disk)에 저장이 되어 사용이 되고 공간 색인구조의 처리는 I/O에 대한 의존도가 크므로, I/O 연산의 병렬처리는 공간 색인구조의 질의반응시간을 현저하게 줄일 수 있다. 본 논문에서는 PR-tree라는 병렬형 공간 색인구조를 제안한다. PR-tree는 MXR-tree에 비해 높은 공간활용도와 빠른 처리시간을 보임으로써 공간 데이터베이스를 위한 효율적인 색인구조로 사용이 될 것으로 기대된다.

1. 서론

데이터베이스의 성능에 영향을 주는 많은 요소들은 복합적으로 작용한다. 그러나 가장 큰 영향을 주는 인자로서는 데이터에 접근하는 도구인 자료구조라 할 수 있다. 데이터의 분포와 질의의 형태는 공간 색인구조의 성능에 영향을 준다. 대부분의 공간 색인구조는 질의 연산을 순차적인 방식으로 처리한다. 이러한 방식의 공간 색인구조는 참고문헌[1,2,6]에서 발견할 수 있다. 병렬처리 공간 색인구조의 연구는 비교적 최근에 시작된 분야라 하겠다.

MXR-tree[5]는 최초의 병렬처리 공간 색인구조이다. MXR-tree는 R-tree계열의 구조로써 공간 data object가 그려진 original space를 공간 분할하는 native space indexing방식을 사용한다. MXR-tree는 R-tree의 노드를 여러 개의 디스크에 배치하기 위해 proximity index방식의 heuristic을 사용한다. proximity index의 목적은 새로이 만들어진 노드가 기존의 노드들과 함께 검색될 확률이 가장 적은 디스크에 새 노드를 배치하는 것이다. 이상적인 노드의 분배를 위해서는 같은 레벨에 있는 모든 형제 노드들을 proximity index에서 고려해야 하나 이를 위해서는 너무 많은 디스크 access가 필요하므로 같은 부모 노드를 가진 형제 노드들만을 proximity index계산에 고려한다. 따라서 proximity index heuristic은 경우에 따라서 노드 분배에 불균형을 가질 수 있다.

MXR-tree는 R-tree의 구조적인 특성을 그대로 갖고 있기 때문에 R-tree의 불필요한 검색경로와 같은 문제를 갖고 있다. 병렬처리 공간 색인구조의 질의 연산에서 D(노드배치에 사용된 디스크의 수)개의 process가 만들어지며 각 process는 동시에 할당된 디스크를 검색한다. MXR-tree는 하나의 tree를 가진 구조이며 모든 노드들은 D개의 디스크에 분배된다. 따라서 MXR-tree는 질의 연산 동안에 D개의 process각각이 검색 또는 삭제 영역과 overlap하는 intermediate entry를 찾을 때마다 process간에 communication이 필요하다. Process간에 communication에 필요한 시간은 사용되는 디스크의 숫자에 비례하여 증가한다. 본 논문에서 제시되는 PR-tree는 native space indexing과 disjoint space decomposition(같은 level에 있는 index rectangle들이 서로 overlap하는 것을 허용치 않는 방식)을 사용한다.

2. PR-tree

효율적인 병렬처리 공간 색인구조는 질의 processing에 있어 total 디스크 access의 수가 적어야함은 물론 access되는 노드가 디스크상에 균일하게 분포되어 있어야 한다. 질의 processing동안에 total 디스크 access의 수를 줄이기 위해서 공간 색인구조는 R-tree[4], R^+-tree[7], R*-tree[3]의 단점을 향상시켜야한다. PR-tree는 data object를 여러 개의 data space에 분배함으로써 intermediate rectangle들 사이의 overlap을 제거하고 leaf 노드에 존재하는 중복된 entry들을 제거한다. PR-tree는 native space indexing과 disjoint space decomposition방식[4, 7]을 사용한다. Leaf 노드에 존재하는 중복된 entry들을 제거하기 위해, PR-tree에서 모든 하위 level rectangle들이 상위 level의 intermediate rectangle에 의해 완전히 포함된다. 이러한 특성을 위해 여러 개의 data space가 사용이 되고, 이러한 data space들은 여러 개의 디스크를 사용하는 병렬처리에 적합하다.

2.1 PR-tree의 구조

PR-tree는 multiple-layer와 multiple-tree의 구조를 가지고 있다. 각 layer는 복수의 트리들로 구성되어 있고, 트리의 개수는 사용되는 디스크의 수와 같다. 각 트리는 height-balanced tree이며 같은 layer에 있는 트리들은 같은 height를 갖는다. 각 data space는 하나의 디스크 상에 위치하는 독립된 하나의 트리와 연관되어있다. 하나의 디스크는 하나이상의 트리를 가질 수 있다. PR-tree에서는 각 트리들이 독립적인 구조를 가지고 있기 때문에 질의 연산 동안에 inter-process communication이 불필요하다. 따라서 모든 디스크의 완전한 병렬처리가 가능하다. 그림1은 2개의 layer와 4개의 트리를 가진 PR-tree의 layout을 보여준다.

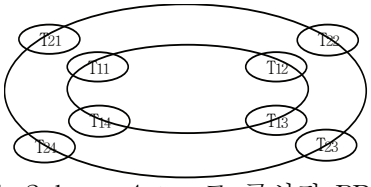

그림1. 2-layer 4-tree로 구성된 PR-tree의 layout

각 트리들은 트리 ID인 T_{ij}(i는 layer의 번호이고 j는 트리의 번호를 나타낸다. 트리번호는 디스크의 번호이기도 하다)에 의해 식별될 수 있다. 디스크 j는 트리 ID T_{ij}(i= 1,2,3,...,L, L은 layer의 수)를 가진 트리들을 포함한다. 예를 들어 그림1에서, 디스크 1은 tree T_{11}과 T_{21}을 포함한다. 따라서 PR-tree의 질의 반응시간은 디스크 access의 최대 수에 비례한다. 만일 P_{ij}가 tree T_{ij}안에 있는 노드들을 처리하는데 걸리는 시간이라면 PR-tree의 질의 반응시간(R)은:

$$R = \max_{j=1\ldots D}\left(\sum_{i=1}^{L} P_{ij}\right)$$

(D는 디스크의 수를, L은 layer의 수를 나타낸다)

그림2는 2차원 data object들에 대한 PR-tree의 object 분배의 예를 보여준다. PR-tree는 R-tree계열의 노드구조를 가진다. 즉 모든 object는 leaf 노드에 저장이 되고, intermediate 노드는 intermediate rectangle들을 나타내는 entry들로 구성이 된다. 그림2에서 PR-tree는 object rectangle 1에서 10까지를 3개의 data space에 entry의 수가 최소인 tree를 선택하는 간단한 heuristic을 사용하여 분배한 것이다.

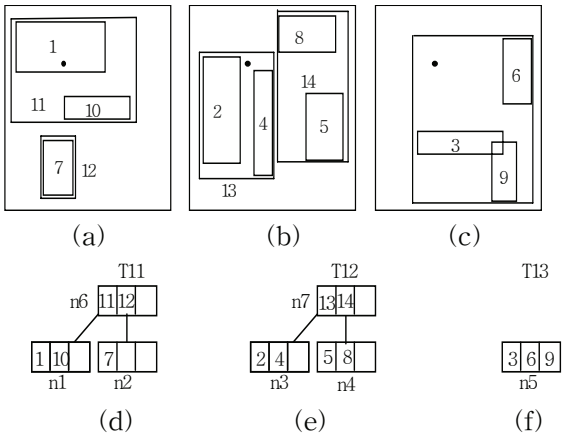

그림2. 점선은 intermediate rectangle, 실선은 object rectangle을 각각 나타낸다: (a)첫 번째 data space (b)두 번째 data space (c)세 번째 data space; (d),(e)(f)는 (a),(b),(c)에 대한 PR-tree 구조

그림2(d), 2(e), 2(f)는 3개의 data space에 상응하는 트리를 보여준다. 그림2(d), 2(e), 2(f)에서 트리 T_{ij}에 있는 모든 노드들은 디스크 j에 저장이 된다. PR-tree의 leaf level에는 중복된 entry가 존재하지 않는다. 그림2에서 PR-tree는 1-layer/3-tree의 구조를 갖는다. 그림3은 그림2의 예에서 주어진 point 질의를 처리하기 위한 PR-tree의 디스크 access 시간을 보여준다. 그림3에서 수평축의 한 칸은 하나의 노드를 access하는데 걸리는 디스크 access 시간을 나타낸다. 하나의 노드를 access하는데 걸리는 디스크 access 시간이 상수이고 한 노드의 크기가 1 page라 가정한다. 그림2에서 오직 data object 1만이 주어진 검색 point와 overlap한다. 그림3에서 주어진 point 질의를 처리하기 위한 PR-tree의 디스크 access의 수는 5이고, 이 5개의 노드를 병렬로

access하는데는 2 unit time이 걸린다. PR-tree에서 각 process는 하나의 디스크와 연계되어, 오직 주어진 검색 영역과 overlap하는 노드들만 access하고 inter-process communication이 없다.

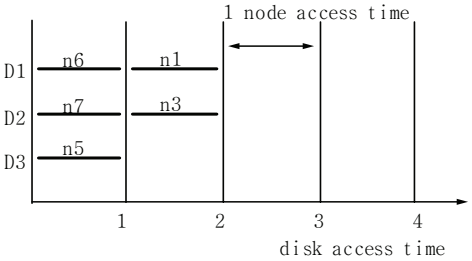

그림3. PR-tree의 병렬처리 디스크 access 시간

2.2 PR-tree의 알고리즘

2.2.1 PR-tree의 삽입 연산

새로운 entry(NEW)를 삽입하기 위해 PR-tree의 삽입 알고리즘은 object distribution heuristic을 사용하여 하나의 tree를 선택을 한다. Object distribution heuristic은 새로운 entry를 받아들일 수 있으면서 PR-tree의 특성(예: disjoint intermediate rectangle과 상위 level rectangle에 의한 완전한 포함)을 만족시키는 하나의 트리를 선택한다. 공간 data object는 임의의 크기와 모양을 갖는다. 또한 이러한 object들은 서로 overlap될 수 있고 공간상에 임의의 위치에 분포하므로, data space의 어느 한 부분이 다른 부분에 비해 data object의 밀도가 아주 높거나 낮을 수 있음을 의미한다. 삽입 경로를 따라 노드의 분할의 가능성이 큰 tree는 보다 많은 노드(intermediate 노드와 leaf 노드)를 만들려는 경향을 가지고 있다. 결과적으로, 높은 노드 분할의 가능성을 가졌던 삽입 경로에 해당하는 영역에 대한 검색 연산에서는 보다 많은 노드가 access되어야한다. Absolute crowd index(AC)에 의한 object distribution은 새로운 entry인 “NEW”를 삽입하기 위한 삽입경로를 따라서 최소한의 노드 분할 가능성을 가진 하나의 트리를 찾는 것이다. AC distribution heuristic은 노드 분할의 가능성을 결정하기 위해 삽입경로를 따라서 노드의 복잡도(crowdness)를 측정을 한다.

$$\sum_{j=1}^{h} (W \times j \times N_j \rightarrow EntryNum)$$

(h는 트리의 order(reverse height), W는 weight 상수, N_j는 삽입 경로의 order가 j인 노드, EntryNum은 노드 N_j안의 entry의 수이다)

Weight 상수인 W(>1)는 높은 order의 노드에 보다 많은 weight를 주기 위해서 사용이 된다. Order가 높은 노드는 entry NEW의 삽입에 의해 보다 쉽게 분할하려는 경향이 있다. 왜냐하면, 새로이 삽입 되는 entry NEW는 leaf 노드에 삽입되고 leaf 노드는 가장 높은 order를 가지고 있기 때문이다. 가장 낮은 AC index값을 가진 트리가 entry NEW의 삽입을 위해서 선택된다. 만일 leaf 노드가 NEW를 받아들일 수 있는 상태라면 tree의 status는 leaf 노드 내의 entry의 수에 의해 A(accept)또는 S(accept and 분할)로 set된다. Status가 A 또는 S인 트리들에 대해서 알고리즘은 4가지의 criteria를 사용을 해서 하나의 트리를 선택을 한다. 이들 4개의 criteria의 적용 우선 순위는:

1. tree order: minimum order를 가진 트리를 선택
2. index 값: minimum AC index값을 가진 tree를 선택
3. tree status: status A를 가진 트리를 status S를 가진 트리에 우선해서 선택
4. 트리에 속한 entry의 수: 최소 숫자의 entry를 가진 트리를 선택

만일 현재의 layer에 어떤 트리도 새로운 entry인 NEW를 받아들일 수 없다면, 현재의 layer를 하위의 layer로 설정을 하고 선택 process를 반복한다. 만일 하위의 layer가 존재하지 않는다면 새로운 layer가 만들어진다.

2.2.2 PR-tree의 노드 분할

삽입연산 동안에 노드가 overflow하는 경우에 노드는 둘로 분할되어야 한다. 각 차원에 대해, overflowing 노드내의 entry들을 나타내는 rectangle들의 시작과 끝 좌표 값을 하나의 배열, Split_Pos[], 에 저장을 하고 정렬한다. 분할 알고리즘은 노드내의 entry의 개수가 capacity(CAP)를 초과할 때 호출되므로, overflowing 노드, N내의 rectangle들의 시작과 끝 좌표 값의 개수는 총 2(CAP+1)이다. Split_Pos[j], j= 1, 2, 3, ..., 2(CAP+1), 안에 있는 각 좌표에 대해, 분할 알고리즘은 overflowing 노드내의 entry를 나타내는 rectangle의 주어진 차원에 대한 시작과 끝의 두 좌표 값이 모두 Split_Pos[j]보다 작거나 같은 rectangle의 숫자를 계산해서 배열 LO_count[j]에 저장한다. 또한 Split_Pos[j], j= 1, 2, 3, ..., 2(CAP+1), 안에 있는 각 좌표에 대해, overflowing 노드내의 entry를 나타내는 rectangle의 주어진 차원에 대한 시작과 끝의 두 좌표 값이 모두 Split_Pos[j]보다 크거나 같은 rectangle의 숫자를 계산해서 배열 UP_count[j]에 저장한다. Split_Pos[j], j= 1, 2, 3, ..., 2(CAP+1), 안에 있는 각 좌표에 대해, overflowing 노드내의 entry를 나타내는

rectangle의 주어진 차원에 대한 좌표 값이 Split_Pos[j]와 overlap하는 rectangle의 숫자를 계산해서 배열 OV_count[j]에 저장한다. LO_count[j], UP_count[j], OV_count[j]의 합은 CAP+1이다. 따라서 OV_count[j]의 값은 (CAP+1)-(LO_count[j]+UP_count[j])에 의해 얻어질 수 있다. 각 LO_count[j]와 UP_count[j] (j= 1, 2, 3, ..., 2(CAP+1))에 대해 둘 중에 적은 수를 택하여 배열 Temp[j]에 저장한다. Balancing factor m은 분할 이후 두 개의 노드 N과 SP사이에 entry수에 있어 균형을 주기 위해 사용이 된다. 만일 어떤 Temp[j]도 m보다 크거나 같은 갖지 않는다면, m값은 MAX(Temp[j], j= 1, 2, 3, ..., 2(CAP+1))로 set된다. 모든 j에 대하여 Temp[j]의 값이 m보다 크거나 같은 것들 중에서 분할 알고리즘은 최소의 OV_count[j]의 값을 가진 하나의 j를 선택한다. 만일 최소의 OV_count[j]의 값이 하나 이상 존재한다면 분할 위치인 Split_Pos[j]에 대해 분할 한 뒤에 노드 N과 SP를 나타내는 intermediate rectangle의 면적 합이 최소인 j를 선택을 한다. 분할 알고리즘은 balancing factor m이 0 이 아닌 이상, 각 차원에 대해서 하나의 분할 위치와 그 분할 위치에 대해 overlap 하는 entry의 개수를 얻는다. 만일 balancing factor m이 0 이라면, PR-tree의 노드 capacity가 증가되어야 한다. Disjoint space decomposition method를 사용하는 공간 색인구조의 노드 capacity는 maximum entry overlap보다 커야 한다. 그렇지 않은 경우 노드의 분할이 불가능하다. 분할 알고리즘은 균형 있는 entry 분배를 위해 가장 큰 balancing factor m을 가진 차원을 선택한다. 만일 balancing factor m이 같다면 분할 position에서의 entry의 overlap이 최소인 차원을 선택한다. Entry의 overlap역시도 동일한 것이 있다면 분할 이후 두 group에 대한 면적을 비교해 최소 값을 주는 차원이 선택된다. 하나의 차원과 분할 위치를 선택한 후, 분할 알고리즘은 CAP+1개의 entry를 두 개의 노드 N, SP로 분배한다. 분할 위치와 overlap하는 leaf 노드 entry는 제거되어 새로이 삽입 알고리즘에 의해 insert된다. 분할 position과 overlap하는 intermediate level entry들은 선택된 분할 차원과 위치에 대해 재귀적으로 sub-divide된다. 분할 위치보다 좌표 값이 작거나 같은 entry들은 노드 N에 저장이 되고, 좌표 값이 크거나 같은 entry들은 노드 SP에 저장이 된다.

3. 결론

본 논문에서 새로운 병렬처리 공간 색인구조인 PR-tree가 소개되었다. PR-tree는 여러 개의 디스크를 사용하여 질의 연산에 따르는 디스크 I/O를 병렬 처리함으로써 질의 processing을 향상시켰다. 공간 data object를 디스크 상에 균일하게 분배하기 위해서, PR-tree는 object distribution heuristic과 연산 알고리즘을 제시하였다. PR-tree는 여러 개의 data space에 data object를 disjoint space decomposition과 하위 level rectangle의 완전포함 특성을 유지하면서 분배함으로써, 질의 속도를 향상시키고 leaf 노드의 불필요한 entry의 중복 또한 제거하였다. PR-tree와 MXR-tree에 대한 성능의 비교에서 PR-tree는 우수한 질의 response time과 공간활용도를 갖고 있는 것으로 분석된다. PR-tree는 공간 database를 위한 효율적인 공간 색인구조로서 사용될 것으로 기대된다.

참고문헌

[1]Bang, K. S. and Lu, Huizhu, SMR-tree: an efficient 색인구조 for Spatial Databases, Proc. of the 1995 ACM Symposium on applied Computing, Nashville, February, pp. 46-50, 1995.

[2]Bang, K. S. and Lu, Huizhu, An Efficient 색인구조 for Spatial Databases, Journal of Database Management, Vol. 7, No. 3, Summer, pp. 3-15, 1996.

[3]Beckmann, N and Kriegel, H. P., The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, ACM SIGMOD, pp. 322-331, 1990.

[4]Güttman, A., R-trees: A Dynamic 색인구조 for Spatial Searching, Proc. of the ACM SIGMOD, pp. 47-57, 1984.

[5]Kamel, I. and Faloutsos, C., Parallel R-tree, ACM SIGMOD, pp. 195-204, 1992.

[6]Lomet, D.B., A Review of Recent Work on Multiple attribute Access Methods, ACM SIGMOD Record, Vol. 21, No. 3, pp. 56-63, 1992.

[7]Sellis, T., Roussopoulos, T. and Faloutsos, C., R^+-tree: A Dynamic Index for Multi-dimensional objects, Proc. of the 13th VLBD Conference, pp. 507-518, 1987.

[8]Understanding GIS, The Arc/Info Method, 1994.