제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

효율적인 버킷 분할과 조인 방법을 이용한 공간 해쉬 스트립 조인 알고리즘 설계

심영복*, 이종연**, 정순기*
*충북대학교 컴퓨터공학과
**충북대학교 컴퓨터교육과
e-mail : shimyb26@hotmail.com

Design of a Spatial Hash Strip Join Algorithm using Efficient Bucket Partitioning and Joining Methods

Young-Bok Shim*, Jong-Yun Lee** and Soon-Key Jung*
*Dept of Computer Engineering, Chung-buk National University
**Dept of Computer Education, Chung-buk National University

요 약

본 논문에서는 인덱스가 존재하지 않는 두 개의 입력 릴레이션에 대해서도 최적의 조인 연산을 수행할 수 있는 공간 해쉬 조인 알고리즘을 제안한다. 인덱스가 존재하지 않는 릴레이션의 처리에 사용하는 기존의 공간 해쉬 조인(SHJ: Spatial Hash Join)과 Scalable Sweeping-Based Spatial Join(SSSJ) 알고리즘을 결합하여 SHJ 알고리즘의 단점으로 지적되고 있는 편향된(skewed) 데이터에 대한 조인 연산의 성능저하 문제를 개선할 수 있는 Spatial Hash Strip Join(SHSJ) 알고리즘을 제안한다. SHJ에서 편향된 데이터의 경우 해쉬 버킷의 오버플로우 처리를 위해 버킷 재분할 방법을 사용하고 있는데 반하여 본 논문에서 제안한 SHSJ 알고리즘에서는 버킷의 재분할 처리 대신에 버킷에 데이터를 삽입하고, 조인 연산과정에서 오버플로우가 발생한 버킷에 대하여 SSSJ 알고리즘을 사용함으로써 편향된 입력 릴레이션의 처리 성능을 제고시킬 수 있도록 한다.

1. 서론

공간 데이터베이스 관리 시스템은 GIS, CAD, 의학 데이터베이스, 그리고 멀티미디어 정보 시스템과 같은 여러 분야에서 사용되는 대용량 멀티미디어 데이터(예, 위성 이미지, 분자 구조)의 효율적인 관리에 중점을 두고 있다. 공간 질의 중에서 데이터 집합으로부터 공간 술어를 만족하는 모든 객체 쌍을 검색하는 공간 조인과 겹침(intersection) 연산이 가장 많이 사용되고 있다. 공간 조인 질의의 예로서 다음과 같은 질의를 들 수 있다. "Find all cities that are crossed by a river". 공간 데이터베이스 시스템에 저장된 데이터는 점, 선, 다각형(polygon), 그리고 평면과 같은 간단한 기하학적인(geometric) 객체 타입과 좀더 단순한 기하학적인 객체 타입으로 유도된 홀(hole)을 갖는 보다 복잡한 객체 타입을 포함하고 있다. 공간 데이터베이스 시스템은 이러한 공간 객체에 대한 질의를 효율적으로 지원할 수 있어야 하며, 사용자는 객체간의 공간 관계를 기반으로 하여 두 개의 공간 입력데이터 집합에 대한 결합 연산을 수행한다. 이러한 공간 결합연산을 공간 조인 연산이라고 한다. 공간 조인 연산은 관계형 데이터베이스에서의 조인 연산과 마찬가지로 매우 많은 비용이 소요되는 연산이다. 따라서 효율적인 공간 조인 알고리즘에 대한 연구는 공간 데이터베이스의 질의처리 연구 분야에서 핵심적인 요소이다.

공간 객체는 여러 점들로 표현되기 때문에 그 크기에 따라 공간 조인을 포함하는 공간 연산은 일반적으로 여과 단계(Filter step)와 정제 단계(Refinement step)로 처리된다[1].

본 논문에서는 공간 연산의 여과 단계에 중점을 두며, 두 개의 입력 릴레이션에 인덱스가 존재하지 않을 경우를 해결하는 공간 조인 알고리즘을 제안한다. 입력 릴레이션에 인덱스가 존재하지 않을 경우는 복잡한 질의의 중간 결과가 조인 연산의 입력 릴레이션이 되거나, 또는 병렬 데이터베이스 환경에서 릴레이션이 다중 프로세서로부터 입력되는 경우에 발생한다. 이러한 문제를 해결하기 위한 연구 일환으로 SHJ[2]와 외부 plane-sweep 알고리즘을 확장한 SSSJ[3] 알고리즘이 제안되었다. SHJ 알고리즘은 편향된(skewed) 데이터의 처리에 단점을 가지고 있다. 두 입력 릴레이션이 상이하게 편향된 데이터 분포를 갖을 경우 해쉬 버킷에 오버플로우가 발생되며, 이를 처리하기 위해 버킷을 재분할해야 한다. 따라서 SHJ 알고리즘은 버킷 재분할로 인해 버킷 효율이 감소하는 단점이 있다. 본 논문에서는 입력 데이터의 편향 문제를 해결하기 위해 버킷 오버플로우를 우선 허용하며, 해당 버킷의 조인 연산 단계에서 오버플로우가 발생했던 버킷은 SSSJ 알고리즘을 사용하고, 그렇지 않은 버킷에 대해서는 내부 메모리 plane-sweep 알고리즘을 사용하는 방법을 제안한다.

논문의 구성은 다음과 같다. 제 2장에서는 관련 연구로서 SHJ와 SSSJ 알고리즘을 살펴보고, 제 3장에서는 SHJ 알고리즘의 단점을 보완할 수 있는 SHSJ 알고리즘에 대해 기술한다. 제 4장에서는 결론 및 향후 연구과제에 대해 기술한다.

2. 관련 연구

관련 연구와 알고리즘의 기술에 사용한 기호는 표 1과 같다.

표 1. 표현 기호

기 호	의 미
A, B	SHJ의 입력 릴레이션
$b_{A,i}$	릴레이션 A에 대한 버킷 i
$b_{B,i}$	릴레이션 B에 대한 버킷 i
P, Q	SSSJ의 입력 릴레이션
r	입력 데이터 사각형
r^x_{min}, r^x_{max}	입력 사각형 r의 x-축 최소값과 최대값
r^y_{min}, r^y_{max}	입력 사각형 r의 y-축 최소값과 최대값
L	sweep line 리스트
L_P, L_Q	각 릴레이션 P, Q의 데이터에 대한 sweep line
L_P^{ij}, L_Q^{ij}	스트립 i~j 구간에 대한 sweep line
N	입력 데이터의 총 개수

2.1 SHJ 알고리즘

SHJ[2] 알고리즘은 관계형 데이터베이스에서 사용하는 Grace 해쉬 조인 알고리즘과 유사하며[4], 인덱스가 존재하지 않는 두 공간 릴레이션 A와 B의 조인 연산에 사용된다. 알고리즘의 처리과정은 다음과 같다. 우선 입력 사각형에 대한 초기 공간 분할 집합을 결정하고, 데이터 집합 A와 B를 분할된 버킷에 할당한다. 연산 결과로서 해쉬 테이블을 생성되며, 각 버킷은 해당 파티션의 사각형으로 버킷 영역을 나타낸다. 해싱을 수행하고, 동일한 파티션을 갖는 A와 B로부터 각 버킷 쌍들을 읽어드려 공간 조인 연산을 수행한다. 그림 1은 SHJ의 공간 분할 과정을 나타낸다.

릴레이션 A에 대한 초기 버킷 영역의 생성에는 bootstrap seeding[5] 알고리즘을 사용하며, R*-tree[6]와 같은 삽입 알고리즘을 이용하여 릴레이션 A의 데이터를 각 버킷에 할당한다. A로부터 모든 객체의 할당을 통해 공간 파티션의 영역 변경과 파티션들의 겹침 현상이 발생할 수 있다. 데이터 집합 B의 삽입시 데이터 집합 A의 최종 버킷 영역을 이용하여 입력 데이터와 겹치는 모든 버킷 영역에 삽입한다.

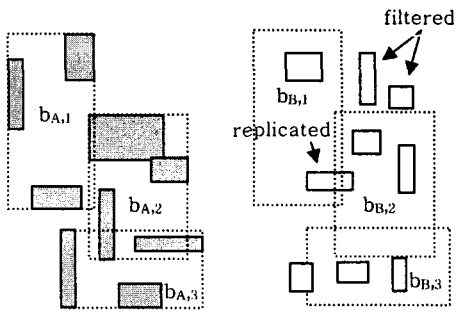


그림 1. SHJ 알고리즘의 분할 단계
(a) 세 개의 파티션에 분할된 데이터 집합 A
(b) 데이터 집합 B의 필터링과 중복

데이터 집합 B에 대해 해쉬을 수행하고, A로부터의 각 버킷 $b_{A,i}$와 B로부터의 해당 버킷 $b_{B,i}$ (같은 영역을 갖는 버킷)간의 부합(match)에 의한 조인 연산을 수행한다. 만약 한 버킷의 메모리 적재가 가능하다면 그것을 읽어서 다른 버킷의 객체들과 함께 단일 스캐닝 방법으로 처리하여 I/O 비용을 최적화한다. 만약 두 버킷 모두의 메모리 적재가 가능하다면 조인 단계의 연산 속도를 향상시키기 위해 plane-sweep 알고리즘을 이용하거나, 또는 한 버킷의 객체를 벌크(bulk) 로드하여 R-tree[7]를 생성하고 인덱스 중첩 루트 조인(INLJ: Indexed Nested Loop Join)을 수행한다.

2.2 SSSJ 알고리즘

본 논문에서 제안한 SHSJ 알고리즘을 이용한 버킷 조인에 사용한 SSSJ 알고리즘의 구조는 다음과 같다. 2-차원 조인 연산시 각 사각형 r∈P 혹은 r∈Q는 x-축에서 하한-경계값 r^x_{min}과 상한-경계값 r^x_{max}에 의해, 그리고 y-축에서 하한-경계값 r^y_{min}과 상한-경계값 r^y_{max}으로 정의된다. 사각형 P와 Q 사이의 교차 영역을 검색하기 위해서는 다음과 같은 알고리즘을 이용한다(그림 2).

Algorithm Rectangle_Join
Step(1) Partition the two-dimensional space into k vertical strips.
Step(2) /* Not necessarily of equal width. At most 2N/k rectangles start or end in any strips, for some k to be chosen later */
if rectangle is contained in a single strip
the rectangle is called small.
else
the rectangle is called large.
Intersection(P(center piece), Q(center piece));
Intersection(P(center piece), Q(small rectangle));
Intersection(P(small rectangle), Q(center piece));
for each strip /* recursively */
Intersection(P(end piece or small rectangle), Q(end piece or small rectangle));

그림 2. 2-차원 rectangle join

x-좌표의 끝점을 중심으로 영역이 정렬되므로 Step(1)에서는 각 스트립의 경계를 계산하고, 정렬된 영역 리스트를 스캐닝 한다. 회귀적으로 처리되는 각 레벨에서의 반복적인 정렬 연산에 소요되는 부하를 감소시키기 위해 정렬된 다수의 작은 영역 리스트로의 분할을 고려해야한다. 리스트 L에 대해서도 마찬가지 경우이다. Step(2)에서의 회귀적인 처리는 전체적인 하위 문제(subproblems)가 메모리에 적재될 때 결정되며, 각 문제의 처리에 내부 plane-sweeping 알고리즘을 이용한다. 각 레벨에서의 회귀적인 처리시 입력 사각형의 전체 수는 분할된 모든 구간은 최대 두 끝점을 가질 수 있으므로 최대 2N이 된다.

Step(2)에서 함수 Intersection()의 구현에 대해 기술한다. Step(2)의 스트립 내에서 가운데 조각(center piece)이 포함되는 교차 계산 문제는 내부 메모리 조인의 경우와 유사하다. 특히, 어떤 가운데 조각은 스트립의 경계에서 시작과 동시에 끝난다. 이것은 스트립 i에 포함된 소형 사각형(small rectangle) r과 스트립 i 에 포함된 가운데 조각 s에 대해 단지 구간 (r^y_{min}, r^y_{max})와 (s^y_{min}, s^y_{max})가 교차된다면 두 사각형 r과 s는 교차한다는 것을 의미한다.

SSSJ 알고리즘은 그림 4와 같다. SSSJ는 앞장에서 설명한 최적화된 Rectangle_Join 알고리즘과 효율적인

plane-sweeping 기법의 결합으로 구성된다. SSSJ에 의해 초기 정렬 연산을 수행하고, 조인 문제를 해결하기 위해 직접 plane-sweeping을 시도한다. Rectangle_Join에서 수직 분할 단계는 단지 plane-sweep에서 사용되는 sweeping 구조의 크기가 메인 메모리보다 클 경우에만 수행된다.

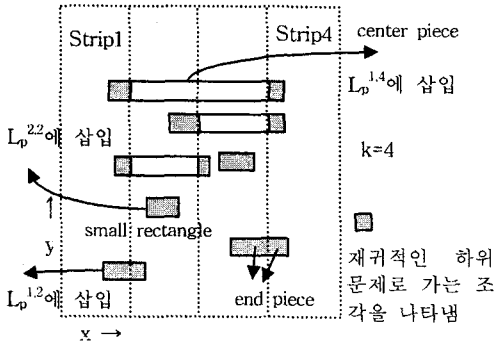


그림 3. 2-차원 조인 알고리즘을 이용한 분할의 예

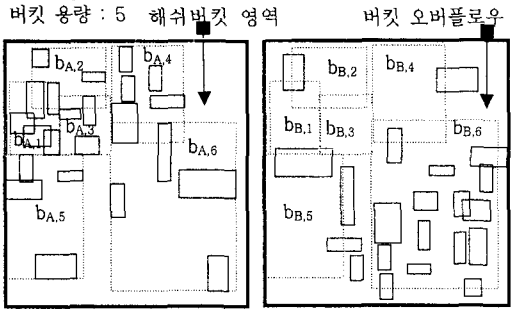
Algorithm SSSJ
Sort sets P and Q on their lower y-coordinates.
Initiate an internal-memory plane-sweep.
if the plane-sweep runs out of main memory
perform one level of partitioning using Rectangle_Join recursively call SSSJ on each subproblem.

그림 4. SSSJ 알고리즘

초기 정렬 후에 sweeping 구조를 메인 메모리에 적재할 수 있는 것으로 가정하고, 단순히 내부 plane-sweeping 알고리즘을 구동 시킨다. sweep이 수행되는 동안 sweeping 구조의 크기를 검사하고, 만약 사전 정의된 임계치에 도달하면 Rectangle_Join 알고리즘을 호출한다. 워크스테이션상의 대용량 메인 메모리를 사용할 경우 내부 plane-sweeping 알고리즘은 천억 단위의 사각형을 처리할 수 있다. 그러나 대용량 데이터나 혹은 매우 불균형한 분포도를 갖는 데이터 집합의 경우 SSSJ는 수행 시간을 최소로 증가시키는 수직 분할 방법을 사용한다.

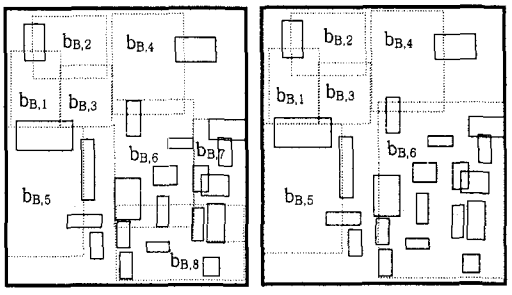
3. SHSJ 알고리즘의 제안

앞 장에서 언급한 것처럼 SHJ 알고리즘은 그림 5에서와 같이 편향된 입력 릴레이션에 대해 외부 데이터를 버킷에 할당시 버킷 오버플로우가 발생하면 메인 메모리에 상주가 가능한 크기로 버킷을 재분할하는 방법을 사용하였다. 따라서 외부 데이터 집합의 입력과정에서 재분할 처리에 추가 비용이 소요된다. 또한 버킷을 분할하게 되면 그림 6(a)에서와 같이 버킷 수가 증가할수록 데이터 집합 B의 중복 저장에 소요되는 데이터가 증가하게 된다. 따라서 중복 데이터에 대한 공간 비용과 조인 연산시 비교 횟수의 증가를 초래하게 된다. 그러나 SHSJ 알고리즘은 기존 SHJ 알고리즘과 동일하지만, 단지 버킷 오버플로우가 발생했을 때 버킷 분할방법 대신에 오버플로우가 발생한 버킷에 계속해서 데이터를 저장하게 된다. 그림 6(b)와 같이 버킷 오버플로우가 발생해도 버킷 분할을 수행하지 않고 데이터를 계속 입력하게 되므로 버킷 분할에 대한 추가 비용이 발생하지 않으며, 중복 저장이 필요한 데이터의 양도 감소하기 때문에 공간 비용과 조인 단계에서 처리 시간도 감소하게 된다. 버킷 조인 단계에서는 기존의 SHJ와 같이 동일한 영역을 갖는 버킷들을 조인하게 되며, 이때 오버플로우가 발생한 버킷에 대해서는 I/O 성능이 우수한 SSSJ 알고리즘을 이용하여 처리한다. 오버플로우가 발생하지 않은 버킷은 내부 plane-sweep 알고리즘으로 처리함으로써 기존 SHJ 알고리즘에 비해 효율적인 버킷 조인 연산을 수행할 수 있다.



(a) 첫 번째 입력 데이터 집합의 버킷 분할 (b) 두 번째 입력 데이터 집합의 버킷 할당

그림 5. 편향 데이터 집합에 대한 버킷 오버플로우



(a) SHJ (b) SHSJ

그림 6. SHSJ를 이용한 버킷 오버플로우 처리

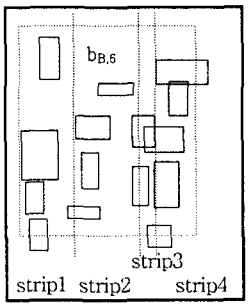


그림 7. 오버플로우가 발생한 버킷 조인을 위해 SSSJ 알고리즘 적용

그림 7은 그림 6(b)에서 버킷 오버플로우가 발생한 버킷 $b_{B,6}$의 조인 처리를 위한 SSSJ의 스트립 sweep 알고리즘의 예를 나타내며, 그림 8은 SHSJ 알고리즘을 나타낸다.

Algorithm SHSJ

Step(1) Obtain initial bucket extents for the inner dataset using bootstrap seeding.
Assign each inner object to one bucket based on the bucket extents and the assignment criteria.

Step(2) Set the outer bucket extents to the final inner bucket extents.
Assign a copy of each outer object to every outer bucket whose extent overlaps the object.
If *a bucket overflow occurs*
Do not perform bucket partitioning and directly assign the rectangle into overflowed bucket.

Step(3) Join corresponding bucket pairs to produce result.
If the bucket was overflowed
call **Rectangle_Join** of SSSJ;
else
Initiate an internal-memory plane-sweep.

그림 8. SHSJ 알고리즘

그림 8의 단계(1)은 기존의 SHJ와 동일한 방법으로 첫 번째 입력 데이터 집합을 공간 해쉬 테이블에 할당하고 단계(2)에서도 버킷 오버플로우가 발생했을 때 분할하지 않고 그대로 삽입한다는 점만 제외하면 기존 SHJ 알고리즘과 동일하게 수행된다. 단계(3)은 단계(1)과 (2)에서 생성된 두 해쉬 테이블에 대해 동일한 영역을 갖은 해쉬 버킷을 메인 메모리로 읽어들여 plane-sweep 알고리즘을 사용하여 조인 연산을 수행한다. 이때 두 번째 해쉬 테이블 생성과정에서 오버플로우가 발생했던 버킷은 SSSJ 알고리즘을 사용하여 처리한다.

4. 결론 및 향후 연구과제

본 논문에서는 두 입력 릴레이션 모두 색인이 존재하지 않는 경우의 공간 조인 연산 알고리즘인 SHSJ 알고리즘을 제안했다. 이 알고리즘은 기존 연구인 SHJ 알고리즘과 SSSJ알고리즘을 결합하여 기존의 SHJ 알고리즘의 편향 데이터에 대한 단점을 개선하였다. 그림 6은 그림 5(b)의 오버플로우에 대한 SHJ와 SHSJ 알고리즘의 처리 예를 보여준다. 그림 6(a)에서는 보는 것처럼 SHJ 알고리즘은 $b_{B,6}$ 버킷을 메모리 상주 가능한 3개의 버킷($b_{B,6}$, $b_{B,7}$, $b_{B,8}$)으로 분할하여 처리하였다. 이 경우 버킷 분할을 위한 추가 비용이 요구되며 버킷의 수가 증가할수록 입력 사각형과 겹침이 발생하는 버킷의 수가 늘어나기 때문에 중복 저장해야 하는 공간 비용도 증가하게 되므로 중복 정장되는 사각형이 많으면 많을수록 버킷 조인 단계에서 교차 비교하는 회수가 증가하게 된다. 그러나 본 논문에서 제안한 SHSJ 알고리즘은 두 번째 데이터 집합의 입력 과정에서 편향 데이터의 경우 버킷 오버플로우가 많이 발생하는데 이를 처리하기 위해 버킷을 분할하지 않고 버킷 영역을 유지하면서 버킷의 데이터를 파일에 출력한다. 두 번째 데이터 집합에 대한 해쉬 버킷에 입력이 끝나면 같은 영역을 갖는 해당 버킷 쌍을 plane-sweep 알고리즘을 통해 처리한다. 이때, 메인 메모리 상주 가능한 버킷 쌍은 내부 plane-sweep 알고리즘을 통해 처리하고 버킷 오버플로우가 발생한 버킷 쌍은 그림 7의 예와 같이 SSSJ 알고리즘을 사용하여 메모리 상주 가능한 여러 스트립으로 분할하여 각 스트립 별로 plane-sweep 알고리즘을 사용해서 조인 연산을 수행하게 된다. 따라서 SHSJ 알고리즘을 사용함으로써 버킷 분할 비용이 존재하지 않으며 중복 저장 비용 및 조인 비교 회수가 감소하는 결과를 얻는다.

향후 연구로는 SHSJ 알고리즘을 TIGER/Line[8] 데이터를 이용해 편향 데이터를 비롯한 다양한 특성에 대해 실험 평가하고 기존의 SHJ와 SSSJ 알고리즘과 비교 분석할 계획이다.

참고문헌

[1] J. A. Orenstein, "A Comparison of Spatial query Processing Techniques for Native and Parameter Spaces," Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic, NJ, pp.343-352, May, 1990.

[2] M. L. Lo and C. V. Ravishankar, "Spatial Hash-Joins," Proceedings of the 1996 ACM SIGMOD International Conference Management of Data, pp.209-220, May 1996.

[3] L. Arge, O. Procopiuc, S. Ramaswami, T. Suel, and J. Vitter. "Scalable Sweeping Based Spatial Join," Proceedings of the 24th Annual International Conference on VLDB, pp.570-581, Aug. 1998.

[4] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, McGraw-Hill, 1997.

[5] M.-L. Lo and C. V. Ravishankar, "Generating seeded trees from data sets," the 4th International Symposium on Large Spatial Databases (Advances in Spatial Databases: SSD '95), Portland, Maine, pp.328-347, Aug. 1995.

[6] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," Proceedings of the 1990 ACM SIGMOD International Conference Management of Data, pp.322-331, May 1990.

[7] S. T. Leutenegger, J. Edgington, and M. A. Lopez, "STR: A Simple and Efficient Algorithm for R-Tree Packing," Proceedings of the 13th International Conference on Data Engineering, pp.497-506, Apr. 1997.

[8] U. S, Bureau of the Census, Washington. DC. "2000 TIGER/Line Files," http://txsdc.tamu.edu/txdata/tiger/tiger2k/tigerfiles.php.