

개선된 공간 해쉬 조인 알고리즘을 이용한 편중 데이터 처리 기법

심 영 복* · 이 중 연**

요 약

지난 수년 동안 공간 데이터의 조인 연산에 대한 많은 연구가 진행되어 왔다. 본 논문에서는 공간 조인연산 시 인덱스가 존재하지 않을 경우, 후보 객체의 여과 단계 처리에 중점을 둔다. 이 분야에 대한 여러 알고리즘들이 제안되었으며 대부분의 경우 공간 데이터의 조인 연산 시 우수한 성능을 나타내고 있다. 하지만, 조인을 위한 입력 테이블의 객체들이 편중되어 분포할 경우 조인 성능이 급격히 저하되는 문제점을 가지고 있으며 이 문제를 해결하려는 연구는 미흡한 실정이다. 따라서, 본 논문에서는 공간 데이터의 편중 문제를 개선하기 위해 기존의 공간 조인 알고리즘 중 Spatial Hash Join 알고리즘과 SSSJ 알고리즘의 장점을 결합한 Spatial Hash Strip Join 알고리즘을 제안한다. 이 알고리즘은 SHJ 알고리즘의 객체 분포에 기반한 공간 분할 특성과 공간 조인 시 SSSJ 알고리즘의 우수한 I/O 특성을 이용한다. 본 논문에서 제안한 SHSJ 알고리즘의 성능 평가를 위해 Tiger/line 데이터를 사용하여 기존 SIJ 알고리즘과 성능을 비교 평가 하였으며 평가 결과 인덱스가 존재하지 않는 입력 테이블에 대한 공간 조인 연산 시 모든 평가 파라미터에 대해 기존의 SIJ 알고리즘보다 우수함이 검증되었다.

Skewed Data Handling Technique Using an Enhanced Spatial Hash Join Algorithm

Young-Bok Shim* · Jong-Yun Lee**

ABSTRACT

Much research for spatial join has been extensively studied over the last decade. In this paper, we focus on the filtering step of candidate objects for spatial join operations on the input tables that none of the inputs is indexed. In this case, many algorithms has presented and showed excellent performance over most spatial data. However, if data sets of input table for the spatial join are skewed, the join performance is dramatically degraded. Also, little research on solving the problem in the presence of skewed data has been attempted. Therefore, we propose a spatial hash strip join (SHSJ) algorithm that combines properties of the existing spatial hash join (SHJ) algorithm based on spatial partition for input data set's distribution and SSSJ algorithm. Finally, in order to show SHSJ the outperform in uniform/skew cases, we experiment SHSJ using the Tiger/line data sets and compare it with the SHJ algorithm.

키워드 : 공간 데이터베이스(Spatial Databases), 공간 조인(Spatial Join), 질의 처리(Query Processing)

1. 서 론

공간 데이터베이스 관리 시스템(Spatial Database Management System)은 GIS, CAD, Bioinformatics, 그리고 멀티미디어 정보 시스템과 같은 여러 분야에서 취급되는 대용량 데이터의 효율적인 관리에 중점을 두고 있다. 공간 데이터에 대해 질의 조건을 만족하는 모든 객체 쌍을 검색하는 공간 조인 및 겹침(intersection) 연산이 가장 많이 사용되고 있다. 예로, "Find all pairs of cities and rivers that intersect."와 같은 공간 질의를 들 수 있다. 공간 데이터베이스에

저장된 데이터는 점, 선, 다각형(polygon), 그리고 평면과 같은 단순한 기하학적(geometric) 객체 타입과 기하학적 객체 타입으로 유도된 홀(hole)과 같은 보다 복잡한 객체 타입을 포함하고 있다. 공간 데이터베이스 관리 시스템은 이러한 공간 객체에 대한 질의를 효율적으로 지원할 수 있어야 하며, 사용자는 객체간의 공간 관계를 기반으로 하여 두 개의 공간 입력 데이터 집합에 대한 결합연산을 수행한다. 이러한 공간 결합연산을 공간 조인(spatial join) 이라고 한다. 공간 조인 연산은 관계형 데이터베이스에서의 조인 연산과 마찬가지로 매우 많은 비용이 소요되므로 효율적인 공간 조인 알고리즘에 대한 연구는 공간 데이터베이스의 질의처리 연구 분야에서 핵심 요소로 취급되고 있다.

공간 객체는 보통 최소 경계 사각형(Minimum Bounding Rectangle: MBR)로 근사화되며 공간 질의는 두 단계로 처

* 이 논문은 2004년도 충북대학교 학술연구지원사업(신진교수 학술연구비: 자연이공)의 연구비 지원에 의하여 연구되었음.

† 준 회원 : 충북대학교 대학원 컴퓨터교육과

** 종신회원 : 충북대학교 컴퓨터교육과 교수, 교신저자

논문접수 : 2005년 1월 13일, 심사완료 : 2005년 3월 3일

리된다[1]. 첫 번째 단계인 여과 단계(Filter step)에서는 질의를 만족하는 모든 MBR의 집합인 후보군을 검색하기 위해 인덱스를 사용한다. 그 후, 두 번째 단계인 정제 단계(Refinement step)에서는 후보군 중에서 실제 질의를 만족하는 객체를 선택하기 위해 객체의 기하 정보를 사용한다.

본 논문에서는 공간 조인 연산 중 조인을 위한 두 입력 테이블에 인덱스가 존재하지 않을 경우의 여과 단계에 중점을 둔다. 이것은 공간 질의처리에서 중간 결과에 대하여 공간 조인 연산을 수행하거나 병렬 처리 환경에서 객체들이 동적으로 분할되어 여러 처리기로 공간 조인 연산을 수행할 경우 발생한다. 이러한 경우 공간 조인 연산 처리를 위한 기존 알고리즘으로는 Spatial Hash Join(SHJ)[2]과 plane-sweep 알고리즘을 확장하여 외부 기억장치의 데이터를 처리할 수 있는 Scalable Sweeping-based Spatial Join(SSSJ)[3] 알고리즘이 있다. 그러나 객체의 공간 분할 기법을 사용하는 SHJ 알고리즘은 편중된(skewed) 공간 데이터에 대한 조인 연산 시 해쉬 버킷의 빈번한 오버플로우가 발생하며 이를 처리하기 위해 버킷을 재분할이 요구되므로 버킷 이용 효율이 감소되는 문제점을 가지고 있다. 따라서 이 논문에서는 기존 SHJ 알고리즘의 문제점을 개선한 Spatial Hash Strip Join(SHSJ) 알고리즘을 제안한다. 알고리즘의 기본 개념은 SHJ 알고리즘의 버킷 재분할 문제를 개선하기 위해 입력 객체의 버킷 할당과정에서 버킷 오버플로우 발생 시 버킷을 분할하지 않고 동일한 버킷 영역에 삽입하며, 해당 버킷의 조인 단계에서 오버플로우가 발생한 버킷에 대해서는 SSSJ 알고리즘을 사용하고, 그렇지 않은 버킷에 대해서는 내부 메모리 plane-sweep 알고리즘을 사용하는 것이다. SHSJ 알고리즘과 기존 SHJ 알고리즘의 성능을 비교 평가하기 위해서 실험 평가 표준 데이터 중 하나인 US Census Bureau의 Tiger/Line 데이터[4]를 사용하였다. 실험 평가 방법으로는 입력 데이터의 개수, 데이터의 편중도와 입출력 버퍼 크기를 달리하고 두 알고리즘의 조인 성능의 평가 요소로는 조인 연산을 위해 요구되는 디스크 페이지 접근 횟수와 조인 연산 수행 시간을 사용한다.

이 논문의 구성은 다음과 같다. 제 2장에서는 공간 조인 연산을 위해 제안된 기존 알고리즘을 기술한다. 제 3장에서는 SHJ 알고리즘의 단점을 보완할 수 있는 SHSJ 알고리즘을 설계하고, 제 4장에서는 이 논문에서 제안한 SHSJ 알고리즘과 기존 SHJ 알고리즘의 성능을 비교하여 분석한다. 마지막으로 제 5장에서는 이 논문의 결론 및 향후 연구과제에 대해 기술한다.

2. 관련 연구

지난 수년 동안 공간 데이터베이스 분야에서는 효율적인 공간 질의처리 기법에 대한 많은 노력을 기울여왔다. 기존 연구의 대부분은 공간 조인의 여과 단계에 중점을 두었으며, 이 기법들은 입력 테이블의 공간 인덱스에 기반한 조인 알고리즘과 단일 인덱스에 기반한 알고리즘 그리고 두 입력

테이블 모두 인덱스가 존재하지 않을 경우를 처리하기 위한 알고리즘의 세 그룹으로 분류할 수 있다.

인덱스 기반 조인 알고리즘에는 조인을 위한 두 입력 테이블 모두 R-tree[5]로 인덱스가 존재한다는 것을 전제로 하며 조인 처리를 위해 교차하는 엔트리 쌍에 대해 두 트리를 동기 순회하는 기법인 R-tree Join(RJ) 알고리즘이 있다[6, 7].

단일 인덱스 조인 기법의 가장 간단한 알고리즘은 Indexed Nested Loop Join(INLJ)[8] 방법이다. 기본 개념은 인덱스가 존재하지 않는 테이블의 각 객체를 R-tree로 구성된 테이블에 대해 윈도우 질의를 수행하는 방법으로 처리한다. 또한 인덱스가 존재하지 않는 테이블에 대해 R-tree를 구성하는 Seeded-Tree Join(STJ) 알고리즘이 있다[9, 10]. 가장 최근에 소개된 알고리즘으로 STJ와 SHJ를 결합한 Slot Index Spatial Join(SISJ) 알고리즘이 있다[11].

두 입력 테이블 모두 인덱스가 존재하지 않을 경우에 대한 공간 조인 알고리즘은 과거에 많이 연구되었다. 예로 가장 기본적이고 원시적인 알고리즘인 중첩 루프 조인(nested loop join)이 있다[8]. 또한 단일 인덱스에 대한 조인 알고리즘 중 하나인 STJ 알고리즘을 확장하여 두 입력 테이블 모두 인덱스가 존재하지 않을 경우를 처리하는 Seeded-tree 알고리즘이 제안되었다[12]. 이와는 별도로 두 입력 테이블의 공간을 규칙적으로[13, 14] 혹은 불규칙적으로[2] 분할하고 분할의 결과로 정의된 버킷으로 객체를 할당하는 기법이 있다. 이 절에서는 이 논문에서 제안한 알고리즘의 관련 연구로서 SHJ 알고리즘과 SSSJ 알고리즘에 대해 기술한다.

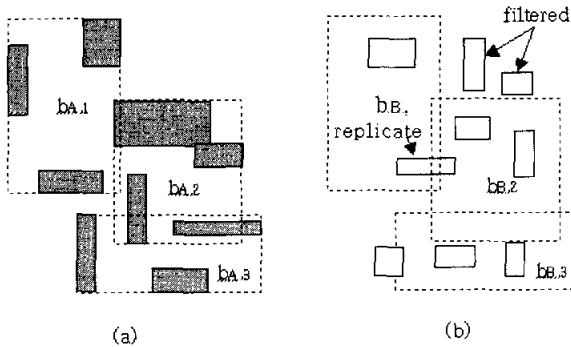
2.1 Spatial Hash Join(SHJ)

SHJ 알고리즘의 기본 개념은 사각형으로 공간 분할된 공간 해쉬 버킷 집합을 정의하고 데이터 집합 A와 B를 생성된 버킷 사각형에 할당하고 같은 영역을 갖는 A와 B로부터의 각 버킷 쌍들은 조인 처리를 위해 디스크로부터 읽어 들여 공간 조인 연산을 수행하는 기법이다. 해쉬 버킷의 영역은 데이터 집합 A에 의해 결정된다. SHJ 알고리즘에서 버킷 정의의 기준은 작은 영역, 적은 겹침, 그리고 할당될 A로부터의 객체가 거의 균일하게 분포할 수 있는 영역을 결정하는 것이다. 영역의 수 s 는 한 버킷에 할당되는 최소경계 사각형(MBR)의 평균수가 메모리에 적재가능하고 영역을 나누는 동안 버퍼 쓰래싱(thrashing)을 피할 수 있게 정의된다. 데이터 집합에는 MBR 분할에 사용할 수 있는 정보가 없으므로 데이터 집합 A의 샘플에 의해 두 단계로 계산된다. 우선 s 개의 MBR을 샘플로 선택하고, 선택된 MBR의 중심 값이 초기 버킷의 영역으로 정의된다. 그 후, 다른 입력 MBR들은 그들의 중심 값이 가장 근접한 영역으로 할당되며, 이때 버킷의 영역 확장이 필요할 때 영역 중심 값의 갱신이 요구된다.

해쉬 단계 동안 A로부터의 각 객체 r_A 가 어떤 영역에도 포함되지 않을 경우 그 객체의 삽입을 위해 영역 확장이 최소로 요구되는 버킷에 할당된다. A로부터의 모든 객체가 할당되고 난 후 각 버킷의 영역은 변하게 되고 버킷들은 서로

겹침이 발생할 수도 있다.

그 후, 데이터 집합 B의 버킷 할당을 위해 A의 버킷 영역과 동일한 영역을 사용하여 버킷에 할당되지만 삽입과정에서 버킷의 영역은 고정적으로 유지되고 객체는 그것과 교차하는 모든 버킷에 삽입 된다. 또한, 어떤 객체는 하나 이상의 버킷에 삽입되고(replicated), 어떤 객체는 어떠한 버킷에도 삽입되지 않는다(filtered). (그림 1)은 SHJ 알고리즘의 조인 예이다.



(그림 1) SHJ 알고리즘의 분할 단계: (a) 세 개의 영역에 분할된 데이터 집합 A; (b) 데이터 집합 B의 필터링과 중복

(그림 1)에서 데이터 집합 B에 대해 해쉬를 수행한 후, 조인은 A로부터의 각 버킷 $b_{A,i}$ 와 B로부터의 해당 버킷 $b_{B,i}$ (같은 영역을 갖는 버킷)의 부합에 의해 수행된다. 만약 한 버킷이 메모리에 적재가능하면 그 버킷을 메인 메모리로 읽어오고 다른 한 버킷의 객체들은 적재된 객체에 대해 단일 스캔 방식으로 처리되며 이 방법은 I/O 비용을 최적화한다. 만약 조인을 위한 두 버킷의 객체가 동시에 메모리로 적재하여 조인 연산이 가능하면 조인 성능이 우수한 plane-sweep 알고리즘[15]을 이용하거나, 한 버킷의 객체를 벌크 로드 R-tree[16]를 생성하여 INLJ 알고리즘을 사용하여 조인 연산을 수행한다.

2.2 Scalable Sweeping-Based Spatial Join(SSSJ)

<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_{min}^x, r_{max}^x | 입력 사각형 r의 x-축 최소값과 최대값 |
| r_{min}^y, r_{max}^y | 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 |

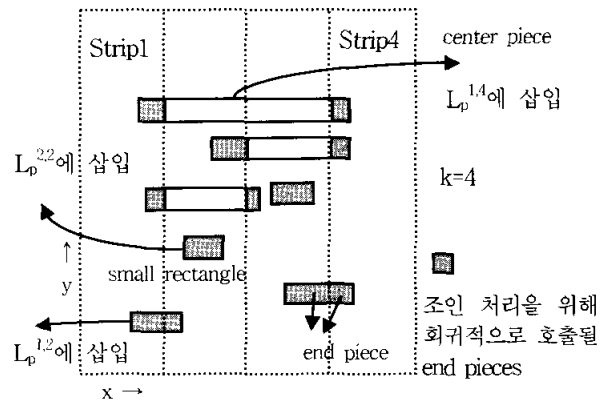
SSSJ 알고리즘은 plane-sweep 알고리즘을 외부 조인이 가능하도록 확장한 알고리즘으로 이 논문에서 제안한 SHSJ 알고리즘의 버킷 조인 단계를 처리한다.

SSSJ 알고리즘을 이용한 2-차원의 조인 처리에 대해서, 각 입력 테이블 $r \in P$ 혹은 $r \in Q$ 는 x-축에서 하한-경계값 r_{min}^x 과 상한-경계값 r_{max}^x 에 의해, 그리고 y-축에서는 하한-경계값 r_{min}^y 과 상한-경계값 r_{max}^y 에 의해 정의되며, 입력 테이블 P와 Q의 데이터 집합 사이의 교차 쌍을 검색하는 조인 처리 알고리즘은 (그림 2)와 같다.

```

Algorithm Rectangle_Join (MBRList itemListP,
                          MBRList itemListQ)
// itemList : MBR수, BufferSize : I/O 크기
Step 1 // 메모리 상주 가능한 k개의 strip으로 분할
SplitTimeToStrip(itemListP, itemListQ);
Step 2 // MBR은 small과 large로 구분
// small : MBR이 하나의 스트립에만 포함
// large : MBR이 둘 이상의 스트립에 포함
// center : large에서 한 스트립 전체에 걸쳐
// 있는 MBR
Step 2.1 /* 먼저 P와 Q로부터의 가운데 조각 사이의
모든 교차 계산 */
IntersectWithCenterPiece(itemListP.center,
                          itemListQ.small);
IntersectWithCenterPiece(itemListQ.center,
                          itemListP.small);
Step 2.1 /* P와 Q 사이의 끝 혹은 소형 조각과
사이의 모든 교차 계산 */
IntersectWithSmallPiece(itemListP.small,
                        itemListQ.small);
    
```

(그림 2) Rectangle_Join 알고리즘



(그림 3) 2-차원 조인 알고리즘을 이용한 분할의 예

```

Scalable Sweeping-Based Spatial Join:
Step 1 // y좌표에 대해 정렬
Sort(itemListP, itemListQ);
Step 2 PlaneSweep(itemListP, itemListQ);
Step 3 //overflow 발생, i는 조인을 위한 버킷 순서쌍
if (itemListPi + itemListQi > out of memory)
    Rectangle_Join (itemListP, itemListQ)
    
```

(그림 4) 테이블 P와 Q의 조인을 위한 SSSJ 알고리즘

(그림 3)은 SSSJ 알고리즘의 strip 분할의 예이며 그 알고리즘은 (그림 4)에서 기술한다. SSSJ 알고리즘의 초기 정

릴 수행 후 조인 문제를 해결하기 위해 직접 plane-sweep을 시도한다. Rectangle_Join(RJ) 알고리즘에서의 수직 분할 단계는 단지 plane-sweep 알고리즘의 수행에서 요구되는 sweeping 구조가 메인 메모리의 크기보다 클 경우에만 수행된다. 대안으로, 보증된 신뢰 범위를 갖는 데이터 집합의 겹침을 측정하기 위해 랜덤 샘플링을 사용할 수 있고, 그 후 이 정보를 사용해서 입력에 대한 분할 여부를 결정하게 된다. 초기 정렬 후 sweeping 구조가 메인 메모리 적재가 가능하다고 가정하고 단순한 내부 plane-sweep 알고리즘을 시작한다. sweeping을 수행하는 동안 sweeping 구조의 크기를 검사하고, 만약 사전 정의된 한계 값에 도달하면 Rectangle_Join 알고리즘을 호출한다.

현재 시스템에 대규모의 메인 메모리 크기가 주어질 때 대부분의 경우 데이터 집합은 내부 plane-sweep 알고리즘으로 수천억 개의 사각형을 처리할 수 있다. 하지만, 만약 데이터의 양이 아주 방대한 경우 SSSJ는 수행 시간을 적당히 증가시키는 수직 분할을 호출한다.

3. SHSJ 알고리즘의 설계

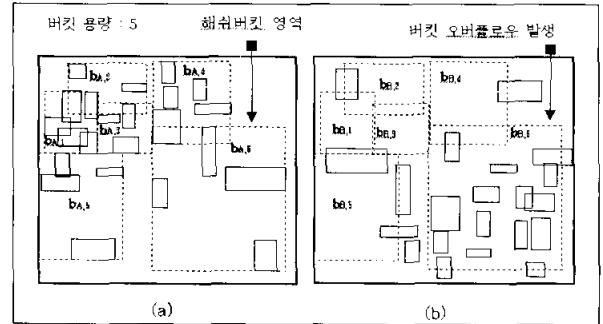
앞 장에서 언급한 것처럼 SHJ 알고리즘은 편중된 입력 테이블에 대해 외부 데이터 집합을 버킷에 할당시 버킷 오버플로우가 빈번하게 발생하며 기존의 SHJ에서는 오버플로우가 발생한 버킷에 대해서 메인 메모리에 적재 가능한 크기로 버킷을 재분할하는 방법을 사용하여 해결했다. 하지만 버킷 재분할 방법은 편중 데이터의 경우 빈번한 버킷 오버플로우로 인해 오버헤드가 크다는 단점을 가지고 있다. 본 장에서는 이 문제를 효율적으로 처리할 수 있는 SHSJ 알고리즘을 설계한다.

3.1 SHSJ 알고리즘의 특성

관련 연구에서 기술한 것처럼 SHJ와 SSSJ 알고리즘은 조인의 대상이 되는 두 입력 테이블 모두 인덱스가 존재하지 않는 경우의 조인 처리를 위해 제안된 기법이다. 두 입력 테이블 모두 인덱스가 존재하지 않을 경우는 복잡한 공간 질의 연산의 중간 결과에 대한 공간 조인 연산 시 발생할 수 있다. 두 테이블의 조인을 수행하기 전에 각각의 테이블에 대해서 선택(select) 연산을 선행한 후 선택 연산의 결과로 얻어진 튜플에 대해서 조인 연산을 수행하기 때문에 두 데이터 집합의 조인 연산을 위해 기존의 인덱스를 사용할 수 없게 된다.

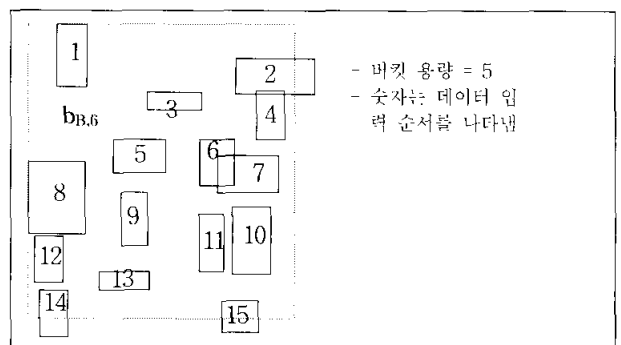
본 절에서는 인덱스가 존재하지 않는 편중 데이터의 조인 문제를 해결하기 위해 기존의 SHJ 알고리즘의 단점을 개선한 SHSJ 알고리즘을 제안한다. (그림 5)는 SHJ 알고리즘을 사용한 두 입력 테이블의 버킷 분할 과정을 도시한 것이다. (그림 5) (a)는 해쉬 버킷을 생성한 후 첫 번째 입력 데이터 집합이 할당된 상태이며 (그림 5) (b)는 첫 번째 데이터 집합에 의해 생성된 해쉬 버킷 영역으로 두 번째 입력 데이터 집합을 할당한 예이다. (그림 5)에서 보는 것처럼 두 입력 데이터 집합의 분포가 매우 상이하게 편중되어 있으므로

(그림 5) (a)에서와 같이 대부분의 데이터들이 좌측 상단에 분포하는 반면 (그림 5) (b)에서는 우측 하단에 집중되어 분포하고 있다. (그림 5)에서 해쉬 버킷의 용량을 5로 정의했을 때 두 번째 데이터 집합을 버킷에 할당 시 버킷 $b_{B,6}$ 는 오버플로우가 발생하며 이 문제를 해결하기 위한 기존 연구인 SHJ 알고리즘에서는 버킷 재분할 기법(bucket repartitioning method)을 사용한다.



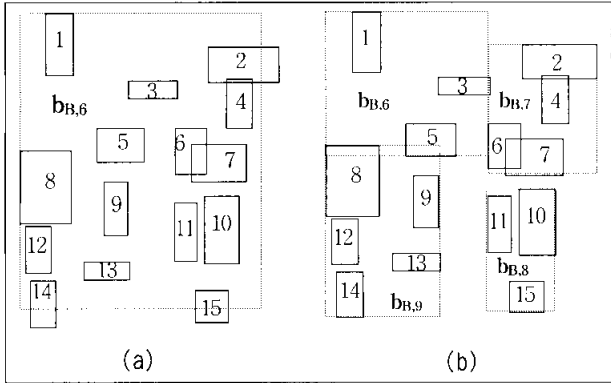
(그림 5) SHJ 알고리즘을 이용한 버킷 분할의 예:
(a) 첫 번째 입력 데이터 집합의 버킷 할당;
(b) 두 번째 입력 데이터 집합의 버킷 할당

본 절에서는 인덱스가 존재하지 않는 편중 데이터의 조인 문제를 해결하기 위해 기존의 SHJ 알고리즘의 단점을 개선한 SHSJ 알고리즘을 제안한다. (그림 5)는 SHJ 알고리즘을 사용한 두 입력 테이블의 버킷 분할 과정을 도시한 것이다. (그림 5) (a)는 해쉬 버킷을 생성한 후 첫 번째 입력 데이터 집합이 할당된 상태이며 (그림 5) (b)는 첫 번째 데이터 집합에 의해 생성된 해쉬 버킷 영역으로 두 번째 입력 데이터 집합을 할당한 예이다. (그림 5)에서 보는 것처럼 두 입력 데이터 집합의 분포가 매우 상이하게 편중되어 있으므로 (그림 5) (a)에서와 같이 대부분의 데이터들이 좌측 상단에 분포하는 반면 (그림 5) (b)에서는 우측 하단에 집중되어 분포하고 있다. (그림 5)에서 해쉬 버킷의 용량을 5로 정의했을 때 두 번째 데이터 집합을 버킷에 할당 시 버킷 $b_{B,6}$ 는 오버플로우가 발생하며 이 문제를 해결하기 위한 기존 연구인 SHJ 알고리즘에서는 버킷 재분할 기법(bucket repartitioning method)을 사용한다.



(그림 6) 버킷 $b_{B,6}$ 의 데이터 삽입 순서

번째 (그림 6)은 (그림 5) (b)에서 두 번째 테이블의 객체들을 버킷 할당 시 오버플로우가 발생한 버킷 $b_{B,6}$ 을 나타내며, 숫자는 입력 데이터의 삽입 순서이다.

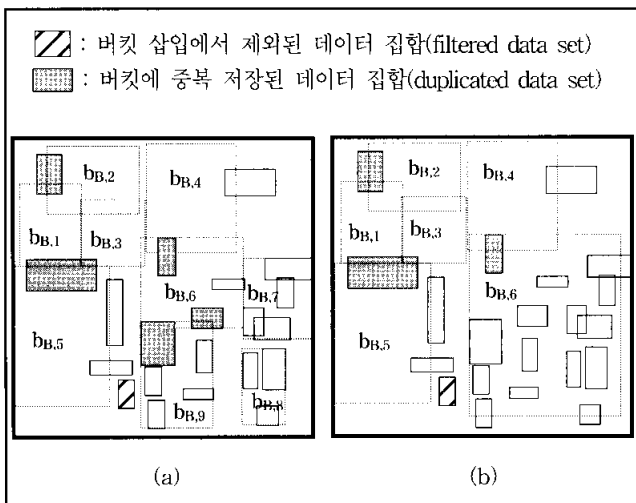


(그림 7) SHJ 알고리즘에 의한 버킷 오버플로우의 처리 결과: (a) 버킷 $b_{B,6}$ 의 입력 데이터; (b) SHJ의 최종 결과

SHJ의 버킷 재분할 알고리즘은 R-tree의 노드 분할 기법과 동일한 알고리즘을 사용하여 수행된다. (그림 6)의 버킷 $b_{B,6}$ 을 SHJ 알고리즘을 사용하여 구성한 최종 해쉬 테이블의 결과는 (그림 7) (b)와 같다.

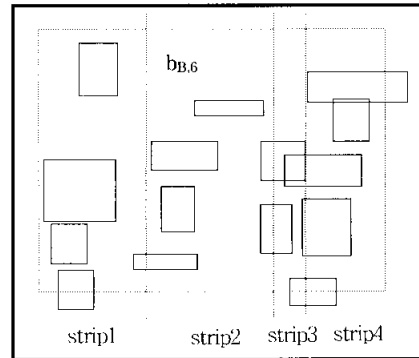
디스크 접근 횟수를 증가시키며 버킷 내의 데이터의 교차 비교 횟수도 증가하게 되므로 CPU 비용이 증가된다. 또한, 객체의 중복 저장을 위한 공간 비용도 증가하게 된다. 이 문제는 두 입력 데이터 집합의 공간 분포가 매우 상이하게 편중되어 있을 경우 더욱 심각해진다. 이러한 단점을 개선하기 위해 SHSJ 알고리즘에서는 버킷 분할 단계에서 버킷 오버플로우 발생 시 버킷을 재분할하지 않고 동일 영역으로 삽입하는 방법을 사용한다. 버킷 조인 단계의 처리를 기존 SHJ 알고리즘과 유사하지만 버킷 용량을 초과하여 입력된 버킷의 조인 연산 시에는 Plane-sweep 알고리즘 대신 조인 연산 시 I/O 성능이 우수한 SSSJ 알고리즘을 사용한다. 따라서 기존 SHJ 알고리즘의 버킷 할당 과정에서 버킷 분할 처리와 이로 인해 중복 저장되는 객체가 증가하는 문제점을 개선할 수 있다.

(그림 9)는 (그림 8) (b)에서 오버플로우가 발생한 버킷 $b_{B,6}$ 의 조인 연산을 수행하기 위한 SSSJ의 strip sweep 알고리즘의 예이며, 한 스트립의 최대 용량이 5인 경우를 도시한 것이다. 버킷의 객체들을 메모리에 적재 가능한 4개의 스트립으로 분할하고 각 스트립에 대해 내부 plane-sweep 알고리즘을 사용하여 공간 조인 연산을 수행하는 전략을 사용한다.



(그림 8) 버킷 오버플로우의 처리 결과 비교: (a) SHJ의 오버플로우 처리 결과; (b) SHSJ의 버킷 오버플로우 처리 결과

(그림 8) (a)는 (그림 5)의 두 번째 입력 테이블의 객체를 SHJ 알고리즘을 사용하여 버킷에 할당했을 경우를 나타내면 (그림 8) (b)는 동일한 데이터를 SHSJ 알고리즘을 사용하여 버킷에 할당한 결과이다. (그림 8)에서 보는 것처럼 SHSJ 알고리즘은 SHJ 알고리즘과 달리 버킷 오버플로우가 발생했을 때 재분할 처리 없이 우선 오버플로우를 허용하므로 (그림 8) (a)와 같이 버킷 재분할을 수행할 경우 여러 버킷에 중복 저장되는 데이터가 증가하게 된다. 중복 저장되는 데이터 집합의 증가는 버킷 조인 단계에서 디스크에 저장되어 있는 해당 버킷의 데이터를 읽어 들이는 과정에서



(그림 9) SSSJ 알고리즘을 이용한 오버플로우 버킷 조인

3.2 SHSJ 알고리즘

이 논문에서 제안하는 SHSJ 알고리즘은 (그림 10)과 같다. (그림 10)에서 Step1은 첫 번째 입력 테이블에 대한 초기 해쉬 버킷 영역을 정의하고 각 입력 객체를 버킷 영역과 할당 기준에 따라 하나의 해쉬 버킷에 삽입하는 과정이다. 이 단계에서 초기 해쉬 버킷 영역을 정의하기 위해 기존의 SHJ 알고리즘에서 사용했던 bootstrap seeding[2] 방법을 사용한다. 이 때 초기 버킷 영역은 버킷의 중심점으로 정의되며 객체 삽입 시 그 영역의 갱신이 요구된다. 각 입력 객체의 버킷 할당 기준은 기존의 SHJ 알고리즘과 동일하지만 버킷 오버플로우가 발생할 경우 버킷 분할을 하지 않고 버킷 용량을 초과하여 입력되는 데이터를 연결 리스트를 사용하여 동일 버킷 영역에 계속 삽입하는 방법을 사용한다.

Step2에서는 두 번째 입력 테이블의 객체들을 해당 버킷에 할당한다. 이 단계는 기존 SHJ 알고리즘에서와 같이 해쉬 테이블의 버킷 영역은 첫 번째 입력 테이블의 객체들을

모두 할당한 후 최종적으로 생성된 버킷의 영역을 두 번째 입력 테이블의 객체들을 할당하기 위한 버킷 영역으로 사용한다. 입력 객체들을 버킷에 할당 시 그것과 겹침이 발생하는 모든 버킷으로 할당되며 어떤 버킷과도 겹침이 발생하지 않는 객체는 버킷 할당에서 제외된다. 객체 할당 방법은 기존 SHJ 알고리즘과 동일하며 단지 두 번째 입력 테이블에 대한 해쉬 버킷의 오버플로우가 발생할 경우 입력클러 버퍼 크기에 맞게 분할하지 않고 연결 리스트를 사용하여 동일 버킷에 할당하는 전략을 사용한다는 것이 기존 SHJ 알고리즘과의 차이점이다.

Step3은 Step1과 2에서 생성된 두 해쉬 테이블에 대해 동일한 영역을 갖은 해쉬 버킷의 데이터를 버퍼 사이즈만큼 메인 메모리로 읽어 들인 후 plane-sweep 알고리즘을 사용하여 조인 연산을 수행한다. 이 때 입력 테이블의 객체들에 대한 버킷 할당 과정에서 오버플로우가 발생하지 않는 버킷은 내부 메모리 plane-sweep 알고리즘을 사용하고 오버플로우가 발생한 버킷은 SSSJ 알고리즘을 사용하여 조인 연산을 수행한다.

```

Algorithm SHSJ
Step1 /*초기 버킷 영역을 정의 (Bootstrap seeding
      알고리즘 사용)*/
GenerateInitialBucket(itemListP);
//MBR 집합 P로 해쉬 테이블 생성
MakeBucketAsP(itemListP);
Step2 // MBR 집합 Q로 두 번째 해쉬 테이블 생성
      //겹침이 발생하는 모든 버킷에 할당
MakeBucketAsP(itemListQ);
If (itemListQ.bucket[i] > out of memory)
버킷 영역의 분할 없이 동일 버킷에 삽입
Step3 /* 테이블 P와 Q의 동일한 버킷 영역을 공유
      하는 버킷 쌍 조인*/
If ( itemListP.bucket[i] + itemListQ.bucket[i] >
out of memory )
ResultSet = RectangleJoin( itemListP.bucket[i],
itemListQ.bucket[i] );
else
ResultSet = PlaneSweep(itemListP.bucket[i],
itemListQ.bucket[i]);
Step4 // 중복된 결과 제거
RefineResult(ResultSet);
    
```

(그림 10) Spatial Hash Strip Join 알고리즘

Step4에서는 Step3의 결과로 얻어진 객체의 MBR 쌍 중에서 중복되는 결과 쌍을 제거하는 단계이다. 이 과정이 필요한 이유는 Step3에서 두 번째 입력 테이블의 객체 할당 시 겹침이 발생한 모든 버킷으로 할당하였으므로 처리 결과인 MBR 쌍들은 중복이 발생할 수 있으며 중복된 MBR 쌍은 질의 처리 결과에서 원하는 값이 아니므로 중복 객체 쌍을 제거하는 과정이 요구되기 때문이다. 이 연산은 기존 SHJ 알고리즘과 동일하며 MBR 쌍들을 정렬 후 한 번의 순차 스캐닝을 통해 간단하게 수행할 수 있다.

4. 실험 및 결과분석

이 장에서는 앞 장에서 제안한 SHSJ 알고리즘의 성능 평가를 위한 실험 환경과 방법 및 실험 결과를 기술한다. 본 실험은 Intel Xeon 550MHz CPU, 1GB RAM, 40GB 하드디스크의 Solaris 8 환경에서 수행하였으며 SHSJ 알고리즘의 구현을 위해 GNU C 컴파일러(gilbc 2.95.3)를 사용하였다.

4.1 실험 방법

이 논문에서 실험 평가의 비교 대상 알고리즘으로는 SHJ 알고리즘을 선택하였으며, 두 알고리즘 모두 인덱스가 존재하지 않는 입력 테이블의 조인 연산을 처리하기 위한 알고리즘이다.

실험 데이터로는 표준 평가 데이터 중 하나인 Tiger/line 데이터를 사용하였으며, 이 가운데 Connecticut(CT), New Jersey(NJ), New York(NY) 그리고 Rhode Island(RI)의 4개 주에 대한 도로(roads)와 수로(hydrographic) 데이터를 사용하였다. 이 논문에서는 공간 질의 처리 단계 중 여과단계에 중점을 두었으므로 Tiger/line 데이터의 도로와 수로에 대한 데이터를 추출한 후 이 데이터들을 MBR 객체로 변환하여 조인 연산에 이용한다. <표 2>는 실험에서 사용할 Tiger/line 데이터 중에서 4개의 주에 대한 도로와 수로에 대한 MBR 객체수와 그 용량을 나타낸다. <표 3>은 알고리즘의 성능 분석에 사용한 평가 항목들의 크기를 나타낸다.

<표 2> 실험 데이터의 특성

| 항 목 | | RI | CT | NJ | NY |
|-----|----------|--------|--------|--------|--------|
| 도로 | 공간 객체 수 | 70058 | 197066 | 443472 | 972525 |
| | 객체 집합 크기 | 3.15MB | 8.71MB | 20MB | 43.9MB |
| 수로 | 공간 객체 수 | 6475 | 27547 | 48202 | 157793 |
| | 객체 집합 크기 | 298KB | 1.22MB | 2.18MB | 7.14MB |

<표 3> 평가 항목의 크기

| 평가 항목 | 내 용 | | | |
|------------|-----------|------|------|------|
| 입력 테이블의 크기 | RI | CT | NJ | NY |
| 입력 객체의 편중도 | 25% | 50% | 75% | 90% |
| 버퍼 크기 | 512 bytes | 1 KB | 2 KB | 4 KB |

<표 3>과 같이 평가 항목으로는 입력 테이블의 크기, 편중도 그리고 버퍼 크기의 세 가지 요소로 한정한다. 입력 테이블의 크기는 <표 1>에 나타낸 것처럼 입력 데이터 집합의 크기가 서로 다른 네 개 주의 객체 집합을 사용하여 평가한다. 입력 객체의 편중도 실험을 위해 <표 1>의 Rhode Island 주에 대한 데이터 집합을 Rhode Island 주 전체 영역 기준으로 8개의 동일한 공간으로 분할한다. 8개의 영역 중 한 영역의 객체 수에 대한 8개 영역 전체의 객체

수 비율을 편중도라고 정의하였으며 그 비율은 각각 25%, 50%, 75% 그리고 90%를 사용한다. 이 비율은 원본 데이터를 재구성하여 만든 것이다.

마지막으로, 버퍼 크기는 512, 1024, 2048 그리고 4096 bytes로 각각 다르게 주어지고 각 버퍼 크기에 대한 각 알고리즘의 조인 성능 특성을 분석한다. 입력 테이블의 크기와 편중도를 평가 항목으로 사용할 때 테이블의 크기에 대한 영향만을 고려하기 위하여 버퍼 크기는 512 bytes로 고정한다. 또한, 편중도에 대한 평가와 버퍼 크기에 대한 평가를 위해 RI 데이터를 사용한다.

성능 비교 기준으로는 위의 평가 항목에 대한 조인 연산 수행 시 요구되는 디스크 페이지 접근 횟수와 조인 연산 시 소요되는 총 수행 시간을 사용한다. 본 실험 평가에서 입력 데이터에 비해 메인 메모리의 크기가 너무 커서 모든 데이터가 메모리에 상주 가능하므로 동시에 메모리에 적재될 수 있는 데이터의 크기를 2Mbytes로 제한하였다.

4.2 결과 분석

본 절에서는 4.1절의 실험 데이터를 SHJ와 SHSJ 알고리즘에 적용한 조인 연산 결과를 바탕으로 각 알고리즘의 성능을 비교 분석한다.

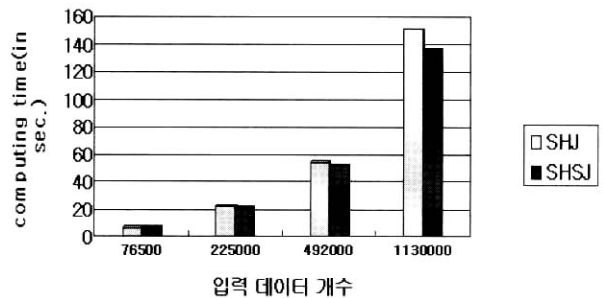
4.2.1 입력 테이블의 크기에 대한 평가

입력 테이블의 크기에 대한 성능 평가를 위해 버퍼 크기는 1024 bytes로 고정하고, <표 2>와 같이 데이터 집합의 크기가 각각 다른 네 개의 주, Rhode Island(RI), Connecticut(CT), New Jersey(NJ) 그리고 New York(NY)에 대한 도로와 수로의 객체 집합의 조인을 위한 두 입력 테이블을 사용한다. <표 4>와 같이 SHJ와 SHSJ 알고리즘에 조인 연산의 결과를 비교한다.

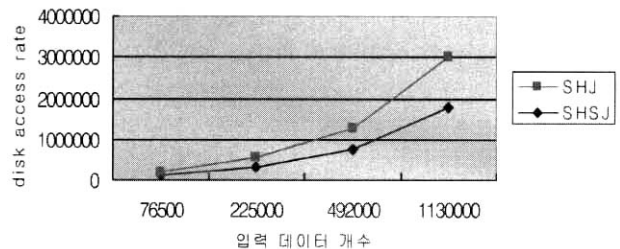
<표 4>는 크기가 서로 다른 네 개의 입력 데이터 집합으로 조인을 수행한 조인 시 요구되는 버킷 수와 두 번째 데이터 집합의 객체가 두 개 이상의 버킷과 겹침이 발생하여 중복 저장에 요구되는 객체의 수, 그리고 조인 수행 전 중복 저장된 객체로 인해 증가한 데이터 크기를 나타낸다.

SHSJ 알고리즘은 버킷의 오버플로우를 허용하므로 버킷 수가 적기 때문에 중복 저장되는 객체의 수가 감소하게 된다.

(그림 11)은 네 개 주에 대한 입력 데이터 집합의 조인 수행 시간을 나타낸다. RI와 같이 데이터 집합의 크기가 작을 경우 버킷의 수가 적고 오버플로우가 거의 발생하지 않았으므로 두 알고리즘은 거의 동일한 성능을 보인다. NY와 같이 입력 데이터 집합의 크기가 조인 수행이 정의된 임출력 버퍼 크기보다 매우 클 경우 SHSJ 알고리즘이 SHJ 알고리즘보다 효율적인 성능을 갖는다. (그림 11)은 데이터 집합의 크기가 다른 네 입력 데이터 집합의 조인 시 요구되는 디스크 접근 횟수(disk access number)를 나타낸다. SHSJ 알고리즘의 경우 버킷 오버플로우 발생 시 분할처리 과정이 없으므로 분할을 위한 추가 디스크 접근이 요구되지 않으므로 SHJ 알고리즘보다 효율적이며 특히, 입력 데이터 집합의 크기가 클수록 우수한 성능을 갖는다.



(그림 11) 입력 데이터의 크기에 대한 조인 수행 시간



(그림 12) 입력 데이터의 크기에 대한 디스크 접근 횟수

<표 4> 입력 데이터의 크기에 대한 조인 수행 결과

| 입력 데이터 | 알고리즘 | 전체버킷수 | 분할버킷수 | 중복객체수 | 해쉬 수행전 입력데이터 크기 | 해쉬 수행후 입력데이터 크기 |
|--------|------|-------|-------|--------|-----------------|-----------------|
| RI | SHJ | 8 | 1 | 5656 | 3.44MB | 3.58MB |
| | SHSJ | - | - | 7342 | 3.44MB | 3.50MB |
| CT | SHJ | 14 | 2 | 24054 | 9.93MB | 11.7MB |
| | SHSJ | - | - | 20602 | 9.93MB | 10.4MB |
| NJ | SHJ | 42 | 10 | 65310 | 22.2MB | 23.8MB |
| | SHSJ | - | - | 47780 | 22.2MB | 23.3MB |
| NY | SHJ | 91 | 25 | 194253 | 55.1MB | 55.9MB |
| | SHSJ | - | - | 179335 | 55.1MB | 55.4MB |

〈표 5〉 입력 데이터의 편중도에 대한 조인 수행 결과

| 편중비율 | 알고리즘 | 전체버킷수 | 분할버킷수 | 중복객체수 | 해쉬 수행전 입력데이터 크기 | 해쉬 수행후 입력데이터 크기 |
|------|------|-------|-------|-------|-----------------|-----------------|
| 25% | SHJ | 18 | 4 | 9606 | 3.44MB | 3.49MB |
| | SHSJ | | - | 4656 | 3.44MB | 3.55MB |
| 50% | SHJ | 22 | 8 | 7957 | 3.44MB | 3.63MB |
| | SHSJ | | | 8150 | 3.44MB | 3.64MB |
| 75% | SHJ | 27 | 13 | 12031 | 3.44MB | 3.76MB |
| | SHSJ | | - | 9384 | 3.44MB | 3.67MB |
| 90% | SHJ | 29 | 15 | 12216 | 3.44MB | 3.77MB |
| | SHSJ | | - | 8521 | 3.44MB | 3.66MB |

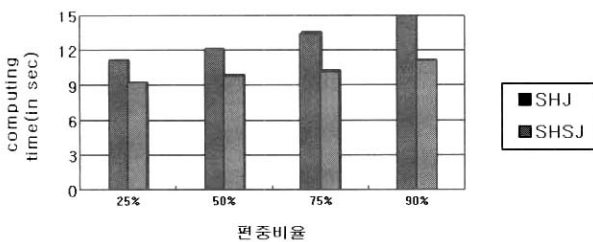
4.2.2 편중도에 대한 평가

〈표 3〉과 같이 각 입력 테이블의 편중도를 각각 25%, 50%, 75% 그리고 90%의 비율로 주어지고, 버퍼는 512 bytes, 입력 데이터를 RI 주의 데이터를 사용하여 평가한다.

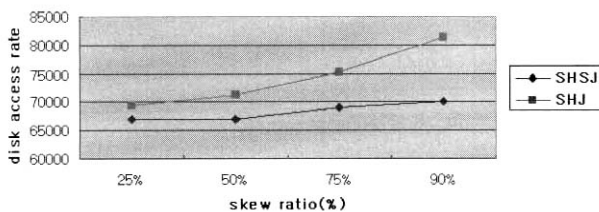
〈표 5〉는 입력 테이블의 편중도에 대한 조인 수행 결과를 나타낸다. 객체의 편중도가 높을수록 해쉬 버킷을 구성하는 동안 발생하는 버킷 오버플로우의 수와 중복 저장되는 객체의 수가 증가하는 특성을 갖는다. (그림 13)은 편중도에 대한 조인 수행시 요구되는 시간을 나타내며, 편중도가 증가할수록 SHSJ 알고리즘이 SHJ 알고리즘 보다 우수한 성능을 갖는다. (그림 14)는 편중도에 대해 조인 수행 시 요구되는 디스크 접근 횟수를 나타내며 편중도가 증가할수록 세 알고리즘 모두 페이지 접근 횟수가 증가한다. SHJ와 SHSJ 알고리즘의 경우 객체의 편중도가 높을수록 여러 버킷에 중복 저장되는 객체의 수가 증가하기 때문이다. SHSJ 알고리즘의 경우 편중도가 높을 경우 버킷 오버플로우가 발생하더라도 버킷을 분할처리하지 않기 때문에 디스크 접근 횟수의 증가율이 작으며 편중비율이 높을수록 SHJ 알고리즘보다 디스크 접근 횟수가 감소됨을 알 수 있다.

4.2.3 버퍼 크기에 대한 평가

입력 데이터 집합인 RI 주의 도로와 수로의 데이터에 대해 〈표 3〉에서 정의한 것과 같이 버퍼 크기를 512, 1024, 2048, 4096 bytes로 변경하면서 두 알고리즘의 성능을 평가한다. 〈표 6〉은 버퍼 크기가 각각 다를 경우 동일한 데이터에 대한 조인 수행 결과 분할된 버킷의 수, 중복 저장된 객체의 수, 해쉬 테이블을 구성한 후 입력 데이터 집합에 대한 크기의 변화를 나타낸다. 버퍼 크기가 작을 경우(512 bytes), 해쉬 버킷의 수가 증가하게 되므로 중복 저장되는 객체의 수가 증가하여 조인 성능이 저하된다. SHJ 알고리즘의 경우 〈표 6〉에서와 같이 버퍼 크기가 작을수록 분할이 발생하는 버킷이 급격히 증가하게 되므로 중복 저장되는 객체의 수는 버퍼가 512 bytes인 경우 4096 bytes일 경우보다 약 70% 정도 증가한 반면 SHSJ 알고리즘의 경우 증가폭이 훨씬 작다. (그림 15)는 버퍼 크기에 대한 조인 수행 시간을 나타내며, 버퍼 크기가 작을 경우, 빈번한 버킷 오버플로우가 요구되는 SHJ 알고리즘보다 SHSJ 알고리즘이 효율적인 성능을 갖는 반면, 버퍼의 크기가 4096 bytes로 클 경우에는 SHJ 알고리즘도 버킷 분할이 요구되는 수가 적으므로 SHJ와 SHSJ 알고리즘은 유사한 조인 성능을 갖는다.



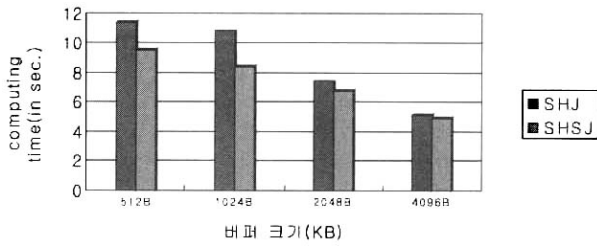
(그림 13) 입력 데이터의 편중도에 대한 조인 수행 시간



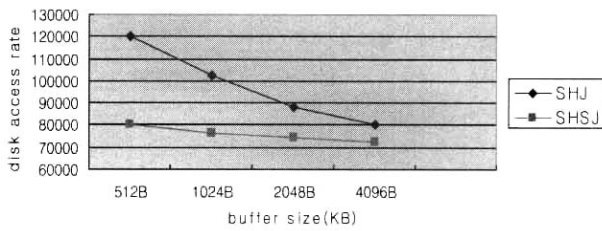
(그림 14) 입력 데이터의 편중도에 대한 디스크 접근 횟수

〈표 6〉 버퍼크기에 대한 조인 수행 결과

| 버퍼크기 (bytes) | 알고리즘 | 전체 버킷수 | 분할 버킷수 | 중복 객체수 | 해쉬 수행전 입력데이터 크기 | 해쉬 수행후 입력데이터 크기 |
|--------------|------|--------|--------|--------|-----------------|-----------------|
| 512 | SHJ | 28 | 14 | 8782 | 3.44MB | 4.06MB |
| | SHSJ | | - | 6630 | 3.44MB | 3.60MB |
| 1024 | SHJ | 22 | 8 | 9399 | 3.44MB | 3.68MB |
| | SHSJ | | - | 6238 | 3.44MB | 3.63MB |
| 2048 | SHJ | 27 | 3 | 12778 | 3.44MB | 3.79MB |
| | SHSJ | | - | 5658 | 3.44MB | 3.58MB |
| 4096 | SHJ | 15 | 1 | 5656 | 3.44MB | 3.58MB |
| | SHSJ | | - | 7342 | 3.44MB | 3.56MB |



(그림 15) 버퍼 크기에 대한 조인 수행 시간



(그림 16) 버퍼 크기에 대한 디스크 접근 횟수

(그림 16)은 버퍼 크기에 대한 디스크 접근 횟수를 나타내며, 버퍼의 크기가 감소할수록 두 알고리즘 모두 조인 성능이 향상되며 버퍼 크기가 4096 bytes일 경우 SHSJ 알고리즘이 SHJ 알고리즘보다 조금 우수하지만 512 bytes일 경우 SHJ 알고리즘보다 월등히 우수한 성능을 나타낸다.

5. 결 론

이 논문에서는 공간조인 연산을 위한 입력 테이블에 인덱스가 존재하지 않을 경우 조인 연산의 효율적인 처리를 위한 SHSJ 알고리즘을 제안하였다. 이 경우의 처리를 위한 기존 연구인 SHJ 알고리즘의 편중 데이터에 대한 성능 저하 문제를 개선하기 위해 버킷 할당과정에서 버킷 영역의 분할 없이 분배되고 버킷 조인 단계에서 두 버킷의 크기가 동시에 메모리 적재가 불가능할 경우 SSSJ 알고리즘을 이용하여 조인 성능의 향상을 시도하였다.

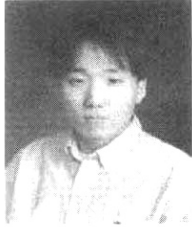
Tiger/Line 데이터를 이용하여 기존 SHJ 알고리즘과 SHSJ 알고리즘의 성능을 실험 평가한 결과는 다음과 같다. SHSJ 알고리즘의 조인 성능은 기존 SHJ 알고리즘에 비해 조인을 위한 입력 테이블의 크기가 클수록 그리고 버퍼 크기가 작을수록 우수한 성능을 나타냈으며 특히, 입력 데이터가 편중되어 분포할 경우 더 효율적 이었다. 반면, 버퍼 크기에 비해 입력 테이블의 크기가 충분히 작은 경우에는 SHJ 알고리즘과 성능이 유사했다. 따라서 이 논문에서 제안한 SHSJ 알고리즘은 공간 조인을 위한 입력 테이블에 인덱스가 존재하지 않으며 편중된 분포를 갖는 경우 공간 조인이 요구되는 GIS 어플리케이션에 효율적으로 이용될 수 있을 것으로 사료된다.

향후 연구 과제로는 SHSJ 알고리즘과 SSSJ 알고리즘, 색인이 존재할 경우 사용되는 기존 RJ 알고리즘과의 성능 분석이 요구된다.

참 고 문 헌

- [1] J.A. Orenstein, "Redundancy in Spatial Databases," In Proceeding of ACM SIGMOD International Conference on Management of Data, pp.294-305, June, 1989.
- [2] M. L. Lo and C. V. Ravishankar, "Spatial Hash-Joins," In Proceedings of ACM SIGMOD International Conference on Management of Data, pp.209-220, May, 1996.
- [3] L. Arge, O. Procopiu, S. Ramaswami, T. Suel, and J. Vitter, "Scalable Sweeping Based Spatial Join," In Proceedings of International Conference on Very Large Data Bases, pp.570-581, Aug., 1998.
- [4] U. S. Bureau of the Census, "2002 Tiger/line Files," 2002.
- [5] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," In Proceedings of ACM SIGMOD International Conference on Management of Data, pp.47-57, Jun., 1984.
- [6] L. Becker, K. Hinrichs, and U. Finke, "A New Algorithm for Computing of Spatial Joins Using R-trees," In Proceedings of the Ninth International Conference on Data Engineering, pp.190-197, Vienna, Austria, Apr., 1993.
- [7] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger, "Multi-Step Processing of Spatial Joins," In Proceedings of ACM SIGMOD International Conference on Management of Data, pp.197-208, Jun., 1994.
- [8] R. Elmasri and S. B. Navathe, Fundamental of Database systems, 3rd edition, Addison-Wesley Publishers, pp.594-600, 2000.
- [9] M. L. Lo and C. V. Ravishankar, "Spatial joins using seeded trees," In Proceedings of ACM SIGMOD International Conference on Management of Data, Minneapolis, MN, pp. 209-220, May, 1994.
- [10] M. L. Lo and C. V. Ravishankar, "Generating seeded trees from data sets," In the Fourth International Symposium on Large Spatial Databases (Advances in Spatial Databases: SSD '95), Portland, Maine, pp.328-347, Aug., 1995.
- [11] N. Mamoulis and D. Papadias, "Slot Index Spatial Join," IEEE Transactions on Knowledge and Data Engineering, Vol.15, No.1, Jan./Feb., 2003.
- [12] M. L. Lo and C. V. Ravishankar, "The Design and Implementation of Seeded Trees: An Efficient Method for Spatial Joins," IEEE Transactions on Knowledge and Data Engineering, Vol.10, No.1, pp.136-151, 1998.
- [13] J. M. Patel and D. J. DeWitt, "Partition Based Spatial-Merge Join," In Proceedings of ACM SIGMOD International Conference on Management of Data, pp.259-270, Jun., 1996.
- [14] N. Koudas and K. Sevcik, "Size Separation Spatial Join," In Proceedings of ACM SIGMOD International Conference Management of Data, pp.324-335, May, 1997.
- [15] R. H. Buting and W. Schilling, "A Practical Divide-and-Conquer Algorithm for the Rectangle Intersection Problem," Information Sciences, Vol.42, No.2, pp.95-112, July, 1987.

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



심 영 복

e-mail : young@himskorea.co.kr
2002년 삼척대학교 정보통신공학과(공학사)
2004년 충북대학교 대학원 컴퓨터공학과
(공학석사)
2004년~현재 충북대학교 대학원 컴퓨터
교육과 박사과정

2004년~현재 HIMS Korea(주) 소프트웨어연구소 연구원
관심분야 : 질의 최적화, 시공간 데이터베이스, GIS, Embedded
System 등.

이 종 연



e-mail : jongyun@chungbuk.ac.kr
1985년 충북대학교 전자계산기공학과
(공학사)
1987년 충북대학교 대학원 전자계산기공
학과(공학석사)
1999년 충북대학교 대학원 전자계산학과
(이학박사)

1989년 비트컴퓨터(주) 개발부
1990년~1994년 현대전자산업(주) 소프트웨어연구소 주임연구원
1994년~1996년 현대정보기술(주) CIM사업부 책임연구원
1999년~2003년 삼척대학교 정보통신공학과 조교수
2003년~현재 충북대학교 컴퓨터교육과 조교수
관심분야 : 질의 최적화, 시공간 데이터베이스, 바이오 데이터
마이닝, GIS, CIM 등.