韓國 컴퓨터情報學會 論文誌
第12卷 第3號, 2007. 7.

2007-12-3-1-6

대용량 메모리를 가진 병렬 데이터베이스 시스템의 조인 연산

박 영 규*

Join Operation of Parallel Database System with Large Main Memory

Young-Kyu Park*

요 약

확장성에서 장점을 가지고 있는 비공유 병렬 프로세서 구조는 병렬 데이터베이스 시스템에서 많이 적용되고 있는 구조이다. 그러나 비공유 병렬 프로세서 구조는 데이터의 분포가 전체 프로세서에게 균일하게 분포되어 있지 않을 경우에는 일부 프로세서에게 부하가 집중되고 이로 인한 성능의 감소가 불가피하게 되는 단점이 있다. 특히 부하의 불균형 정도가 심한 경우에 조인 연산을 수행할 때 이런 성능 감소의 단점은 두드러진다. 본 논문은 비공유 병렬 프로세서 구조에서 부하의 불균형 정도가 심한 경우에도, 조인 연산을 실시하기 전에 부하 불균형을 고려함으로써 성능 감소를 최소화하고, 메모리의 대용량화를 이용하여 성능을 높인 조인 알고리즘을 제시한다. 또한 알고리즘의 성능 분석을 위한 분석 모델을 제시하며, 분석 모델을 통하여 데이터 불균형 문제를 해결하기 위한 다른 알고리즘과의 성능을 비교한다.

Abstract

The shared-nothing multiprocessor architecture has advantages in scalability, this architecture has been adopted in many multiprocessor database systems. But, if the data are not uniformly distributed across the processors, load will be unbalanced. Therefore, the whole system performance will deteriorate. This is the data skew problem, which usually occurs in processing parallel hash join. Balancing the load before performing join will resolve this problem efficiently and the whole system performance can be improved. In this paper, we will present an algorithm using merit of very large memory to reduce disk access overhead in performing load balancing and to efficiently solve the data skew problem. Also, we will present analytical model of our new algorithm and present the result of some performance study we made comparing our algorithm with the other algorithms in handling data skew.

▸ Keyword : 조인 알고리즘(Join Algorithm), 데이터 불균형(Data Skew), 해시(Hash)

• 제1저자 : 박영규
• 접수일 : 2007.5.29, 심사일 : 2007.6.20, 심사완료일 : 2007. 7.20.
* 가야대학교 컴퓨터정보학과 교수

Ⅰ. 서 론

시스템의 메모리 용량이 커지면서 데이터베이스를 메인 메모리에 적재하여 데이터베이스 시스템의 성능을 높이는 MMDBMS(Main Memory Database Management System)에 관한 연구가 계속되고 있다. MMDBMS는 데이터 관리가 간단해지고 성능이 높아지는 장점이 있으나[1], 데이터베이스의 크기가 아주 크거나 데이터베이스의 크기가 증가하는 응용에는 사용하기 힘든 한계가 있다. 이를 극복하기 위한 방법으로 디스크 상주형 데이터베이스와 MMDBMS를 하나의 제품으로 구현한 하이브리드 MMDBMS제품도 등장하고 있다[2]. 디스크 상주형 데이터베이스 시스템 관점에서 볼 때 메모리의 크기가 대량화 되는 것을 이용하여 시스템의 성능을 높이기 위한 연구가 더욱 필요하다고 생각된다. 디스크 상주형 데이터베이스 시스템의 구조를 논하자면 비공유 구조(Shared Nothing), 디스크공유 구조(Shared Disk), 전체공유 구조(Shared Everything) 그리고 이들의 혼합 구조인 하이브리드 구조(Hybrid)로 크게 분류할 수 있다. 이들 구조 중 비공유 구조는 확장성과 유용성 면에서 다른 구조에 비하여 장점을 가지고 있는 것으로 알려져 있다. 조인 연산은 데이터베이스 연산 중 가장 빈번히 사용되면서 비용이 아주 많이 드는 연산중 하나이며, 이런 이유로 성능향상을 위한 연구가 집중되고 있다[3]. 병렬 해시 조인 알고리즘은 비공유 구조의 데이터베이스에 적합한 알고리즘이다[4]. 이 알고리즘은 릴레이션들의 튜플들을 해시하여 버킷들을 구성하고, 프로세서들에게 할당을 한다. 각 프로세서들은 자신에게 할당된 버킷들의 튜플들을 조인하게 된다. 이러한 조인 알고리즘은 데이터가 프로세서들 간에 균등하게 분포가 되어 있을 경우 높은 성능을 보이게 되지만, 프로세서들 간에 데이터가 불균등하게 분포된 경우에는 성능이 떨어지게 된다[5]. 이는 각 프로세서들이 처리해야할 데이터의 양이 다르므로 다른 노드들이 데이터 처리를 완료한 후에도 일부 프로세서는 데이터 처리를 계속해야하는 현상이 발생하기 때문이다. 이럴 경우, 각 프로세서가 처리하는 데이터의 양을 최대한 균등하도록 조절함으로써 각 프로세서들이 같은 시간에 연산을 끝내도록 하는 것이 조인 알고리즘의 성능에서 아주 중요해 진다. 각 프로세서가 처리하는 데이터의 양을 균등하도록 조절하는 조인 알고리즘들은 이미 많이 발표되어있으며, 이러한 알고리즘으로는 ABJ+(Extend ed Adaptive Load Balancing Parallel Hash Join)알고리즘[6], ADJ (Analysis and Distribution Parallel Hash Join)알고리즘[7] 등이 있다. 이들 알고리즘들은 조인 이전 단계에서 프로세서간의 데이터양을 균등하게 만들기 위한 디스크 접근 때문에 릴레이션의 크기가 커질수록 성능이 다소 떨어지는 단점이 있다. 본 논문에서는 데이터의 불균형 분포에 따른 성능 감소를 해소하고, 대용량 메모리를 이용하여 디스크 접근회수를 줄임으로써 성능을 향상시킨 알고리즘을 제시하는데 이 알고리즘을 ATJ (Analysis and Tuning Parallel Hash Join)알고리즘이라고 한다. ATJ 알고리즘은 버켓들에 대한 정보를 분석하고, 그 정보를 바탕으로 데이터를 조절하여 분산함으로써 불균형 분포를 해소한다. 또한, 버킷을 분산할 때와 조인을 수행할 때 발생하는 디스크 입출력의 횟수를 줄여서 성능을 개선한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 연구에 대하여 언급하며, 3장에서는 제안하는 알고리즘에 대하여 설명한다. 4장은 분석 모델에 대한 설명을 다루고, 5장에서는 알고리즘들의 성능을 비교하고, 6장에서 결론을 다룬다.

Ⅱ. 관련 연구

병렬 프로세서 데이터베이스의 기본 구조로 비공유 구조와 전체공유 구조 그리고 디스크 공유 구조, 그리고 이들의 혼합 구조인 하이브리드 구조를 들 수 있다[8][9]. 전체 공유 구조는 모든 프로세서들이 메모리와 디스크를 공유하며, 디스크 공유 구조는 모든 프로세서가 직접 접근할 수 있는 디스크와 공유되지 않는 자신만의 로컬 메모리를 가지는 구조이다. 데이터베이스 응용 면에서 볼 때 이들 두 구조는 확장성이 약한 단점이 있다. 비공유 구조는 프로세서마다 자신의 디스크와 메모리를 가지고 있고, 서로 공유하지 않는 구조이다. 이 구조에서는 수십에서 수백 개의 노드들 간에 서로의 간섭 없이 확장시킬 수 있는 장점을 가지고 있는 반면, 데이터의 분포가 균형을 이루지 못할 경우 성능이 낮아질 수 있다. 하이브리드 구조는 이들 구조의 특성들이 혼합된 구조이다. 데이터베이스에서 수행되는 연산중에서 조인 연산이 아주 비용이 많이 드는 대표적인 연산이며, 이런 조인 연산이 데이터 분포가 불균형을 이루고 있는 비공유 구조에서 수행 될 경우 비용이 상승하게 되는 문제점이 발생하게 된다. 비공유 병렬 데이터베이스 시스템에서 제시되는 여러 가지 조인 알고리즘 중에서 해시에 의해 버킷을 구성하여 조인을 수행하는 병렬 해시 조인 알고리즘이 많이 이용되고 있으며, GRACE 해시 조인 알고리즘이 대표적인 해

시 조인 알고리즘으로 알려져 있다[10]. GRACE 해시 조인 알고리즘의 경우는 비공유 구조가 아닌 디스크 공유 구조에서도 병렬화한 PGHJ라는 병렬 해시 조인알고리즘이 발표되었었는데, 이 알고리즘에서도 노드간의 병렬처리로 인한 성능향상을 보이고 있다[11].

GRACE 해시 조인 알고리즘은 분할단계와 일치단계로 구성되어 있다. 분할단계에서는 두개의 릴레이션들을 동일한 수의 버킷들로 나누고 각 버킷들을 서로 다른 프로세서의 디스크에 저장한다. 이때 한 버킷의 크기가 메모리의 용량을 초과하지 않도록 하기 위해 가능한 많은 개수의 버킷을 사용하게 된다. 일치단계에서는 두 릴레이션의 일치된 버킷을 메모리에 적재하여 조인을 실행하며, 조인한 결과를 모으게 된다. 버킷을 메모리에 적재하여 조인을 실행할 때, 성능을 높이기 위하여 메모리의 용량에 맞추어서 되도록 많은 양의 버킷을 메모리에 동시에 적재를 하는데 이 과정을 버킷 튜닝(Bucket Tuning)이라고 한다[10]. 버킷 생성을 위하여 해시를 할 때 릴레이션의 조인 속성의 값이 균등하게 분포되어 있지 않다면 버킷의 크기는 불균일하게 될 것이고, 노드에 따라 튜플의 수가 특별히 많아지는 경우가 발생하게 될 것이다. 이를 데이터 불균형이라고 하며 조인의 성능을 떨어트리는 요인이 된다. 이런 데이터 불균형 분포로 인한 성능저하를 해결하기 위한 알고리즘들이 발표되었으나 대부분의 알고리즘은 성능의 저하를 어느 정도 완화하였을 뿐 해결하지는 못하였다. 그러나 ABJ+ 알고리즘은 데이터 불균형 분포에 관계없이 일정한 성능을 보이는 알고리즘으로 데이터의 불균형으로 인한 문제를 해결하였다[6]. 그리고 ADJ 알고리즘은 데이터의 불균형으로 인한 문제를 해결하면서 ABJ+ 알고리즘의 성능을 능가하는 알고리즘이다. ADJ 알고리즘은 데이타의 불균형 분포에 영향을 받지 않으며 기존 알고리즘 보다 디스크 접근 횟수를 감소함으로써 성능을 높이는 특징을 가지고 있다. 그러나 데이터 불균형 정도가 낮은 경우는 GRACE 해시 조인 알고리즘에 비하여 성능이 낮게 나타나는 문제점이 있다.

III. 제안 알고리즘

ATJ 알고리즘은 데이터의 불균형 분포에 영향을 받지 않는 알고리즘으로, 기존의 알고리즘들보다 디스크 접근 횟수를 감소시켜 성능을 한 단계 더 높인 알고리즘이다. 그리고 데이터 불균형의 정도가 낮더라도 ADJ와 달리 성능이 크게 낮아지지 않는 장점을 가지고 있다. 이 알고리즘은 비공유 구조의 해시 조인 알고리즘으로서 빌딩 릴레이션을 R, 프르빙 릴레이션을 S라 가정할 때, 다음과 같은 단계들을 거쳐 수행된다.

단계 1 : 릴레이션 분석

각 노드들은 두 릴레이션을 디스크로부터 읽어 들이면서 적절한 해시 함수들을 이용하여 해시를 수행한다. 해시를 하여서 각 버킷 크기 정보를 저장하게 되는데 이를 버킷 카운터라고 한다. 이 단계에서는 버킷을 생성하지는 않으며 각각의 해시 함수에 대한 버킷 카운터 값만 생성, 저장하게 된다.

단계 2 : 버킷 조정

각 노드는 버킷 카운터 값들을 미리 정해진 조정노드로 보낸다. 조정 노드는 이 값들을 이용하여, 각 노드가 처리할 버킷들을 결정하는데 노드마다 버킷을 할당하기 위하여 최적적합전략(Best-fit strategy)인 아래 두 식 (3.1)과 (3.2)를 만족하는 n개의 버킷들을 선택한다.

$$\sum_{j=1}^{n} |\text{버켓}_{ij}| \leq \frac{|R|+|S|}{\text{노드수}} \quad \cdots\cdots (3.1)$$

$$\sum_{j=1}^{n+1} |\text{버켓}_{ij}| > \frac{|R|+|S|}{\text{노드수}} \quad \cdots\cdots (3.2)$$

여기서 i는 노드 번호, $|\text{버켓}_{ij}|$는 i 노드의 j번째 버켓의 크기, $|R|$는 R 릴레이션의 크기, $|S|$는 S 릴레이션의 크기를 나타낸다. 조정노드는 이런 정보들을 모든 노드에 전달한다. 이 단계에서는 각 노드의 버킷들 중 버킷 저장용 메모리에 저장할 버킷들도 결정해야 한다. 메모리는 버킷 저장용를 수행하기 위한 메모리 영역과 버킷을 저장하기 위한 영역으로 나누어져 있다. 버킷 저장용 메모리에 저장된 버킷들은 단계3에서 디스크에 저장되는 과정이 생략이 되고, 단계 4에서 수행하는 조인단계에서 디스크에서 읽어 들이는 과정이 생략되므로, 성능을 향상시킬 수 있다.

단계 3: 릴레이션의 분산

각 노드는 디스크로부터 R과 S릴레이션들의 튜플들을 읽어 들여 해시함수를 이용하여 해시한다. 조정노드에 의해 설정된 버킷할당에 따라 튜플들을 해당 노드로 상호 네트워크를 통해 전송하게 된다. 각 노드들은 자신에게 할당되는 튜플들을 2단계에서 생성한 정보에 따라 디스크로 저장하거나 버킷 저장 메모리에 저장을 한다.

단계 4 : 조인

각 노드는 메모리 용량에 맞게 버킷들을 조정하고 로컬조인을 수행한다.

ABJ+ 알고리즘은 두 릴레이션 R과 S에 대하여 6회의 입출력이 발생한다. ADJ 알고리즘에서는 이를 5회로 줄이기 된다. ATJ 알고리즘에서는 이를 다시 버킷저장용 메모리의 크기만큼 입출력을 줄인다. 메모리의 전체크기에 대한 버킷 저장 메모리의 크기 비율이 커지면 해시용 메모리가 줄어들므로 해시비용은 높아지게 되고, 대신 메모리의 입출력 비용이 줄어지게 되는데 전체적인 성능은 개선이 된다.

VI. 성능 측정 모델

이 장에서는 ATJ 알고리즘의 성능을 측정하기 위한 모델을 제시한다. 먼저 매개변수를 설명하고 다음에 알고리즘에 대한 비용함수를 설명한다.

4.1 비용 함수의 매개 변수

- **작업 부하에 대한 매개 변수**

$|R|$: R 릴레이션의 크기

$|S|$: S 릴레이션의 크기

σ : 데이타 불균형 분포의 정도

$$\sigma = \frac{|P_s| - |P_u|}{|P_s|}$$

t : 튜플의 크기

- **시스템 매개 변수**

M_g : 각 노드의 해시용 메모리 용량

M_s : 각 노드의 버킷 저장 메모리 용량

$$M_s = \alpha \cdot \frac{|R|+|S|}{N} \cdot t$$

α : 버킷 저장 메모리의 릴레이션에 대한 비율

M_{tot} : 각 노드의 메모리 용량

$M_{tot} = M_g + M_s$

N : 시스템 내의 노드 수

Icpu : 한 튜플을 처리하기 위한 명령어의 수

μ : CPU의 처리율(단위, MIPS)

ω_{io} : 디스크의 I/O대역폭(단위, Bytes/Sec)

ω_{comm} : 노드당 통신 채널 대역폭(단위, Bytes/Sec)

- **측정 매개 변수**

T_{atj} : ATJ 알고리즘의 조인 비용

T_{anal} : 한 노드의 릴레이션 분석 소요시간

T_{comm} : 한 노드의 릴레이션 전송 소요시간

T_{cord} : 한 노드의 릴레이션 조정 소요 시간

T_{cpu} : 한 노드의 릴레이션 처리 소요 시간

T_{dist} : 한 노드의 릴레이션 분산 소요 시간

T_{io} : 한 노드의 릴레이션 디스크 I/O소요 시간

T_{read} : 한 노드의 릴레이션 일기 소요시간

T_{join} : 조인 단계에서의 소요 시간

T_{gjoin} : 범용 메모리 조인 소요 시간

T_{sjoin} : 버킷 저장 메모리 조인 소요 시간

$T_{g\text{-}hash}$: 범용 메모리에 해시테이블 구축시간

$T_{g\text{-}proc}$: 범용 메모리에 해시테이블 검색시간

$T_{s\text{-}hash}$: 버킷 저장 메모리에 해시테이블 구축시간

$T_{s\text{-}proc}$: 버킷 저장 메모리에 해시테이블 검색시간

4.2 비용함수

이 논문에서 제시한 모델에는 두 가지의 가정을 하였다. 하나는 알고리즘 각 단계 내에서의 중첩이 완벽하다는 것이며, 또 하나는 각 단계 사이에서는 중첩이 허용되지 않는다는 것이다.

ATJ 해시 조인 알고리즘에 대한 비용함수는 다음과 같이 계산된다.

- 해시조인을 위한 전체 처리 시간

$T_{atj} = T_{anal} + T_{cord} + T_{dist} + T_{join}$

: R과 S릴레이션을 읽어서 버킷들이 분산되어 있는 정도를 분석하는 시간(T_{anal})과 버킷들을 조정하고 각 노드에게 버킷을 할당하는 시간(T_{cord}), 각 노드로 버킷을 분산시키는 시간(T_{dist}), 그리고 두 릴레이션을 조인하는 시간 (T_{join})의 합으로 구성된다.

• 버킷 분산도를 분석하는 시간

$T_{anal} = \max(T_{read} , T_{proc})$

튜플들을 읽고 해시를 위한 처리는 동시에 발생하고 이들은 중첩된다는 가정에 따라 최대 처리 시간이 버킷 분산 시간이 된다.

-. 튜플을 읽는 시간

$$T_{read}=\frac{|R|+|S|}{N}\cdot\frac{t}{\omega_{io}}$$

: 각 프로세서가 디스크로부터 릴레이션 R과 S를 읽는 시간이다.

-. 튜플을 해시하는 시간

$$T_{proc}=\frac{|R|+|S|}{N}\cdot\frac{I_{cpu}}{\mu}$$

: 각 프로세서가 릴레이션 R과 S의 튜플을 해시하는 시간이다.

0. 버킷들을 조정하고 각 노드에게 할당하는 시간

$T_{cord}\approx 0$

: 조정 노드에서 버킷들을 조정하고 각 노드에게 버킷을 할당하는데 약간의 CPU시간이 소용되는데 이 수치는 너무 작아서 무시하였다.

0. 버킷을 분산시키는 시간

$T_{dist} = \max(T_{io} , T_{proc} , T_{comm})$

: 튜플들의 입출력, 해시를 위한 CPU처리 그리고 버킷 이동은 동시에 발생하고 이들은 중첩된다는 가정에 따라 최대 처리시간이 버킷 분산 시간이 된다.

-. 튜플들을 입출력하는 시간

$$T_{io}=2\cdot\frac{|R|+|S|}{N}\cdot\frac{t}{\omega_{io}}-\frac{M_s}{\omega_{io}}$$

: 부하 균등이 고려되어 각 노드에게 버킷이 할당되었으므로 모든 노드간의 입출력이 거의 동일하며, 버킷저장용 메모리의 입출력 시간이 제외된다.

-. 튜플들을 해시하는 시간

$$T_{proc}=\frac{|R|+|S|}{N}\cdot\frac{I_{cpu}}{\mu}$$

: R과 S릴레이션의 튜플들을 해시하기 위한 CPU처리 시간이다.

-. 튜플의 분산을 위한 통신시간

$$T_{comm}=\frac{N-1}{N}\cdot\frac{|R|+|S|}{N}\cdot\frac{t}{\omega_{comm}}$$

: 부하가 균등하도록 버킷이 할당되었으므로 각 노드들은 자신이 조인할 부분을 제외한 튜플들을 해당 노드에 분산하게 되며 이 시간은 노드마다 거의 동일하다.

• 버킷들의 조인을 수행하는 시간

$T_{join} = T_{gjoin} + T_{sjoin}$

: 버킷들을 조인하는 시간, 범용 메모리의 튜플을 조인하는 시간(T_{gjoin})과 버킷 저장용 메모리의 튜플을 조인하는 시간(T_{sjoin})의 합이다.

-. 범용 메모리 조인시간

$$T_{gjoin}=\lceil(\frac{(|R|+|S|)\cdot t}{N}-M_s)/M_g\rceil\cdot(T_{g-hash}+T_{g-proc})$$

: 범용 메모리에 디스크에 있는 버킷들을 읽어 들여서 조인하는 시간이다. 전체 버킷의 튜플에서 버킷 저장용 메모리에 있는 버킷의 튜플을 뺀 부분만 조인하는 시간이다.

-. 범용 메모리에 해시 테이블을 구축하는 시간

$$T_{g-hash} = \max(\frac{M_g}{\omega_{io}}, \frac{M_g}{t}\cdot\frac{I_{cpu}}{\mu})$$

: 튜플들의 입출력작업과 조인하기 위해 해시 테이블을 구축하는 CPU 처리 작업이 동시에 발생하고 이들은 중첩된다는 가정에 따라 최대 처리 시간이 해시 테이블 구축 시간이 된다.

-. 버킷 저장용 메모리의 해시 테이블을 검색하는 시간

$$T_{g-proc} = \max(\frac{M_g}{\omega_{io}}, \frac{M_g}{t}\cdot\frac{I_{cpu}}{\mu})$$

: 튜플들의 입출력과 해시 테이블 검색이 동시에 수행되고 이들은 중첩된다는 가정에 따라 최대 처리 시간이 버킷 검색 시간이 된다.

-. 버킷 저장용 메모리 조인시간

$$T_{sjoin} = T_{s-hash} + T_{s-proc}$$

: 버킷 저장용 메모리에 저장된 버킷들을 조인하는데 소요되는 시간이다.

-. 버킷 저장용 메모리에 해시 테이블을 구축하는 시간

$$T_{s-hash} = \frac{M_s}{t} \cdot \frac{I_{cpu}}{\mu}$$

: 버킷 저장용 메모리에 있는 튜플들에 대한 처리이므로 입출력작업이 필요 없으며, 해시 테이블을 구축하는 CPU 처리 작업만이 해시 테이블 구축 시간이 된다.

-. 버킷 저장용 메모리의 해시 테이블을 검색하는 시간

$$T_{s-proc} = \frac{M_s}{t} \cdot \frac{I_{cpu}}{\mu}$$

: T_{s-hash} 와 마찬가지로 버킷 저장용 메모리에 있는 튜플들에 대한 검색이므로 입출력작업이 필요 없다. 조인하기 위해 해시 테이블을 검색하는데 필요한 CPU 처리 시간이 버킷 검색 시간이 된다.

V. 성능 측정 결과

4장에서 제시한 비용 함수를 이용하여 성능 측정한 결과를 제시하기 위하여 매개 변수의 값을 나타낸다. 비교를 위하여 GRACE 알고리즘과 ADJ 알고리즘의 성능도 같이 측정하였다.

R 릴레이션의 크기 (|R|) : 10,000,000 ~ 20,000,000튜플/노드 (디폴트 : 10,000,000)
S 릴레이션의 크기 (|S|) : 10,000,000 ~ 20,000,000튜플/노드 (디폴트 : 10,000,000)
튜플의 크기 (t) : 1000 Byte/튜플
노드의 수 (N) : 64)
메모리 용량 (M) : 100 ~ 1000 MByte/노드 (디폴트 : 100)
버킷 저장 메모리의 릴레이션에 대한 비율(α) : 0 ~ 0.3 (디폴트 : 0.2)
CPU처리율 (μ) : 2000 MIPS
I/O 대역폭 (ω_{io}) : 노드당 400 MByte/Sec
통신 채널 대역폭(ω_{comm}) : 노드당 400 MByte/Sec
명령어 패스길이 (I_{cpu}) : 1,000 명령어

5.1 불균형 분포도에 따른 성능비교

그림1은 데이터 불균형 분포도의 변화에 대한 각 알고리즘의 성능을 측정한 도표이다. GRACE 알고리즘을 보면 데이터의 불균형 분포도가 증가함에 따라 조인 비용이 급격히 증가되지만 ATJ 알고리즘과 ADJ 알고리즘은 변화가 발생하지 않는다. 이는 두 알고리즘이 각 노드에 할당된 버킷의 전체 크기가 균등하므로 데이터의 불균형 분포에 영향을 받지 않고 조인 비용이 일정해 진다는 것을 알 수 있다. ATJ 알고리즘과 ADJ 알고리즘을 비교해 보면 ATJ 알고리즘의 성능이 훨씬 나은 것을 볼 수 있는데 이는 ATJ 알고리즘의 장점인 버킷을 분산하는 단계부터 버킷저장용 메모리에 저장된 튜플에 대한 디스크 출력과, 조인단계에서 그 튜플들에 대한 디스크 입력을 생략하였기 때문이다.
불균형 분포의 정도가 낮은 경우는 GRACE 알고리즘이 더 높은 성능을 보이는데 이는 부하를 균등하게 분포시키려는 ATJ 알고리즘과 ADJ 알고리즘의 오브헤드 때문이며, ATJ 알고리즘은 ADJ 알고리즘에 비하여 그 차이가 상당히 줄어 있음을 볼 수 있다.

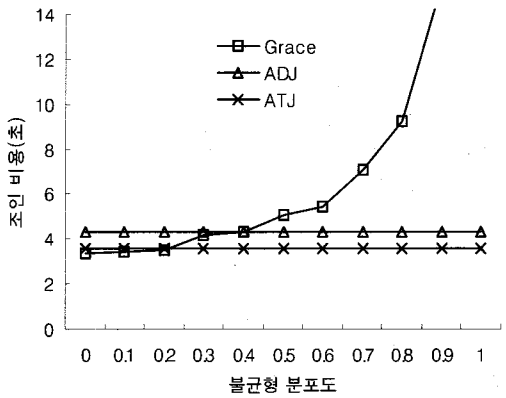

그림 1. 불균형 분포에 따른 성능 변화
Fig. 1 Skew Effect on join algorithms

즉, ADJ 알고리즘이 가지고 있는 문제점인 불균형 분포가 적은 경우 성능이 떨어지는 점도 ATJ 알고리즘은 거의 문제가 되지 않는다. 불균형 분포의 정도가 커지면 GRACE 알고리즘의 성능은 떨어지게 되지만 ATJ 알고리즘의 성능

은 떨어지지 않고 좋은 성능을 보인다.

5.2 릴레이션의 크기에 따른 성능 측정

릴레이션의 크기는 조인하는 릴레이션에 따라 바뀌게 되므로 릴레이션의 크기에 따른 성능분석도 의미가 있다고 본다. 그림 2의 경우 불균형분포를 0.3으로 하였을 경우를 측정한 것으로 릴레이션의 증가에 알고리즘사이에 조인비용이 약간씩의 변화가 있지만 거의 선형적으로 증가하고 있다. 그림 1에서 보면 알 수 있는 것처럼 알고리즘간의 성능차이는 별로 없는 불균형 분포도가 0.3일 경우에 ATJ 알고리즘이 가장 좋고, Grace 알고리즘이 ADJ 알고리즘보다 성능이 좋게 나타나고 있다.

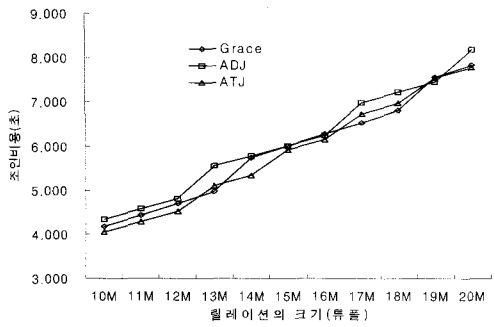

그림 2. 릴레이션 크기에 따른 성능변화
Fig. 2 Effect of relation size on join algorithms

5.3 메모리 용량 변화에 따른 성능 측정

ATJ 알고리즘만으로 버킷저장용 메모리의 용량을 증가시키며 성능 변화를 보는 것이 그림3이다. 이 비는 버킷 저장용 메모리의 릴레이션 크기에 대한 비율이며, 메모리 용량은 고정시키고 측정한 값이다.

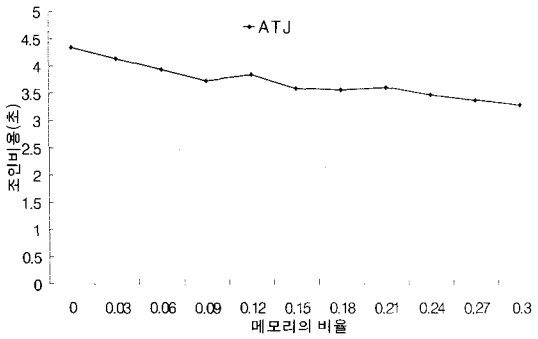

그림 3. 메모리 용량변화에 따른 성능 변화
Fig. 3 Effect fo memory capacity on ATJ

비율이 증가할수록 성능이 높아지는데, 이는 비율이 증가되면 버킷을 디스크로 저장하지 않고 메모리에서 보유하는 양이 늘어나면서 디스크의 입출력 횟수를 줄이는 효과가 있기 때문이다. 물론 그 양이 커지면 해시를 하기 위한 메모리의 양이 줄어들기 때문에 성능이 감소하는 면이 발생하나 디스크 입출력이 성능에 차지하는 비중이 크므로 그 감소가 성능에 큰 영향을 미치지 못하고 전체성능을 높이게 된다. 그래프가 선형적으로 감소만 하지 않는 이유는 바로 해시를 위한 메모리의 양이 줄어든 것이 반영된 것이다. 전체적으로 버킷 저장용 메모리의 양이 커지면 성능이 높아지게 된다. 그리고 그 성능향상의 정도가 높은 편이다.

VI. 결론

병렬 프로세서 데이터베이스 구조중 하나인 비공유 구조는 그 특성상, 노드간의 부하 분포가 불균형을 이루면, 노드간의 작업량이 달라지므로 이로 인하여 성능이 저해될 수 있다. 이는 데이터베이스에서 수행하는 연산 중에서 비용이 아주 많이 드는 대표적인 연산인 조인 연산을 수행할 때 잘 드러나며, 그림1의 GRACE 알고리즘의 예에서 잘 알 수 있다.

본 논문에서는 비공유구조의 병렬 프로세서 데이터베이스 시스템에서 노드간의 데이터 분포가 불균형을 이루더라도 성능이 저해되지 않는 알고리즘을 제안하였다. 이 알고리즘은 데이터 불균형 분포를 해소하고, 메모리의 대용량화를 이용하여 디스크 입출력 횟수를 감소시킴으로써 성능을 향상시켰다. 그리고 성능 분석 모델을 제시하였으며, 세 가지 모델에 대하여 데이터 불균형 분포에 따른 성능 변화를 보였다. 메모리 용량의 변화가 알고리즘에 미치는 영향도 같이 보였으며, ATJ 알고리즘이 불균형 분포가 심한 조인에 효과가 크고, 메모리의 크기가 커질수록 성능이 향상된다는 것을 알 수 있었고 불균형 분포가 낮은 경우에도 장점이 있는 알고리즘이라는 것도 알 수 있었다.

참고문헌

[1] Hector Garcia-Molina, "Main Memory Database Systems: An Overview", IEEE Transaction on knowledge and data engineering vol 4, No 6 December 1992

[2] 원훈, "하이브리드 MMDBMS의 도래", 컴퓨터 월드, pages 106-113, 2005

[3] Donovan A. Schneider, David J. DeWitt, " A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", Proceeding of ACM SIGMOD, pages 110-121, 1989

[4] M. Kitsuregawa. Hidehiko Tanaka and Tohru Moto-oka, "Architecture and Performance of Relational Algebra Machine GRACE", In Processing International Conference of Parallel Processing, pages 241-250, 1984

[5] M. Kitsuregawa and Yasushi Ogawa, "Bucket Spreading Parallel Hash : A new, robust parallel hash join method for data skewness in the super database computer", In Proceedings of 16th International Conference on Very Large Data Bases, pages 210-221. Brisbane, Australia, August 1990.

[6] Kien A. Hua and Chiang Lee, "Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning", In Processing of 17th International Conference on Very Large Data Bases, pages 525-535, 1991

[7] 김용걸, 김성혜, 박영규, 진성일, 임기욱, "비공유 병렬 데이터베이스 시스템에서 데이터 불균형 분포를 위한 부하 균등해시 조인 알고리즘", 정보과학회 논문지 제22권 8호

[8] P. Valduriez, "Parallel Database Systems : the case for shared-something", Proc. 9th Int. Conf. on Data Engineering, 460-465, 1993

[9] David J. DeWitt and Jim Gray, "Parallel Database System : The Future of High Performance Database Systems", Communication of the ACM, pages 85-98, June 1992

[10] Priti Mishra and Margraret H. Eich, "Join Processing In Relational Databases", ACM Computing Surveys. Vol. 24. No. 1, March. 1992

[11] 김창현, 조행래, "데이타베이스 공유 시스템에서 병렬 해쉬 조인 알고리즘의 구현", 정보과학회 학술 발표 논문집, pages 43-45, 2002

저 자 소 개

박 영 규

1997년 : 충남대학교 컴퓨터과학과 박사수료

1997년-현재 : 가야대학교 컴퓨터 정보학과 교수