병렬 조인에서 샘플링 기반 비용 예측 기법을 이용한 균등 부하 분산

박 웅 규†

요 약

데이터베이스 시스템에서 조인 연산은 시스템의 성능에 영향을 주는 가장 복잡하고 소모적인 연산이다. 데이터베이스 시스템의 성능 향상을 위한 많은 병렬 처리 알고리즘들이 제안되었으나 기존의 방법들은 AVS(Attribute Value Skew)와 JPS(Join Product Skew) 등과 같은 데이터 편재를 고려하고 있지 않다. 따라서 데이터 편재의 상황에서 기존의 방법들은 조인 연산 중에 노드들 간의 부하 불균형으로 인하여 그 성능이 급격하게 저하된다. 본 논문에서는 병렬 조인 시에 AVS와 JPS를 고려하여 노드간에 균등하게 부하를 분산하는 방법과 이를 이용한 효율적인 병렬 조인 알고리즘을 제안한다. 제안된 알고리즘은 먼저 기존의 샘플링 방법을 이용하여 조인 연산의 입력과 결과 릴레이션의 데이터 분포를 예측하고, 이를 기반으로 데이터 값에 대한 조인 비용을 산출한다. 그리고 히스토그램 균등화 기법을 이용하여 국부적인 조인 과정에서 노드들 간에 부하 균등을 성취할 수 있도록 데이터를 각 노드에 재분배한다. 본 논문에서는 성능 평가를 위하여 제안된 알고리즘과 기존의 대표적인 알고리즘들을 위한 모의 실험 모델을 제시하고 모의 실험 결과를 기술한다. 성능 측정 결과 제안된 알고리즘이 기존의 알고리즘들에 비해서 데이터 편재의 상황에서 성능이 우수한 것으로 나타났다.

Uniform Load Distribution Using Sampling-Based Cost Estimation in Parallel Join

Ung-Kyu Park†

ABSTRACT

In database systems, join operations are the most complex and time consuming ones which limit performance of such system. Many parallel join algorithms have been proposed for the systems. However, they did not consider data skew, such as attribute value skew(AVS) and join product skew(JPS). In the skewness environments, performance of their algorithms can degrade due to load imbalance during join operation processing. In this paper, we propose a framework for a uniform load distribution and an efficient parallel join algorithm using the framework to handle AVS and JPS. In our algorithm, we estimate data distributions of input and output relations of join operations using the sampling methodology and evaluate join cost for the estimated data distributions. Finally, using the histogram equalization method we distribute data among nodes to achieve good load balancing among nodes in the local joining phase. For performance comparison, we present simulation model of our algorithm and other join algorithms and present the result of some simulation experiments. The results indicate that our algorithm outperforms other algorithms in the skewed case.

※ 본 논문은 한국과학재단 1996년 후반기 Post Doc. 연수 지원 결과의 일부임.

† 정 회 원 : 서원대학교 전자계산학과 교수

논문접수 : 1999년 1월 26일, 심사완료 : 1999년 5월 25일

1. 서 론

최근에 멀티미디어 정보 처리의 필요성 대두와 기존의 OLTP(Online Transaction Processing)와 DSS(Decision Support System) 분야 등에서의 복잡한 질의 처리와 대용량 데이터베이스의 신속한 처리의 필요성의 증대에 의하여 고성능 데이터베이스 시스템의 필요성이 요구되고 있다. 고성능 시스템의 구현을 위한 연구는 데이터베이스 시스템의 병렬화 및 분산화를 중심으로 활발하게 진행되어 오고 있다. 그 결과로 Oracle, Sybase, Informix, IBM사 등 주요 데이터베이스 응용 관련 회사들에서는 고성능 데이터베이스 구축을 위한 병렬 및 분산 DBMS를 이미 상용화하고 있는 실정이다.

데이터베이스 시스템에서 조인 연산은 연산 처리의 복잡성과 응용에서의 사용 빈도수가 매우 높은 연산이기 때문에, 고성능 데이터베이스 시스템의 구축을 위하여서는 조인 연산에 관한 효율적인 알고리즘의 개발이 필요하다. 이러한 배경으로 인하여 병렬 조인 알고리즘에 관한 많은 연구들이 진행되었다[1,2,6,8,9,18]. 그 중에서 병렬 해쉬 조인 방법은 병렬 처리가 용이하며, 성능이 우수함이 여러 연구 결과에서 입증되었다[6,9]. 병렬 해쉬 조인 방법의 장점은 해쉬 함수를 적용한 데이터의 균등 분산과 각 노드에서의 국부적인 병렬 조인 과정을 통하여 병렬 처리에 의한 CPU 처리 시간 및 입출력 시간의 단축이다. 그러나 기존의 병렬 해쉬 조인 방법은 데이터가 편재된 실제적인 상황하에서 데이터의 불균형 분산으로 인한 각 프로세서에 할당된 부하의 불균형으로 심각한 성능 저하를 가져온다[1,6,11].

병렬 조인 연산 처리에 영향을 주는 여러 종류의 데이터 편재가 존재한다. 그 중에서 입력 릴레이션의 조인 속성 값의 편재(AVS : Attribute Value Skew)는 각 노드에서의 조인 연산 수행 시에 부하의 불균형을 초래한다는 점에서 병렬 조인 방법의 성능에 영향을 미친다. 한편 데이터베이스의 질의 처리에서 조인 연산의 결과는 다음 연산의 처리를 위하여 디스크에 저장되거나 또는 결과 릴레이션을 사용자에게 보여 주기 위하여 협력 노드(coordinator node)에 전송되는 조인의 최종 과정을 수행하게 된다. 따라서 결과 릴레이션에서의 조인 속성 값 분포의 편재(JPS : Join Product Skew)는 결과 튜플의 생성과 조인의 최종 과정에서 각 노드에서 행하여지는 작업의 양에 영향을 준다[11,14].

이와 같은 배경으로 최근에 데이터의 편재를 고려한 여러 부하 균등 병렬 조인 방법들이 제안되었다[1,5,13,16,17]. 이러한 방법들은 AVS를 고려하여 각 노드에 데이터를 균등하게 분산하여 부하 균등을 이루고자 하였다. 그 중에서 [1,13,17]에서 제안된 방법들은 입력 릴레이션 중에서 구성 릴레이션(building relation)의 AVS만을 고려하였고, 조사 릴레이션(probing relation)의 AVS와 결과 릴레이션의 JPS를 고려하고 있지 않다. 또한 [5,16]에서 제안된 방법은 결과 릴레이션의 JPS를 고려하고 있지 않다. 따라서 기존의 방법들은 데이터가 편재된 실제 상황에서 병렬 조인 시에 노드 사이의 균등한 부하를 얻을 수 없으며, 이로 인한 성능 저하를 초래할 수 있다[14].

한편 [13]에서의 히스토그램 균등화 기법은 구성 릴레이션의 데이터 편재에 의하여 조인 연산을 위한 데이터가 각 노드에 불균등하게 배분되는 것을 방지한다. 하지만 이 방법은 조사 릴레이션의 데이터 분포가 구성 릴레이션의 것과 다른 경우에 기존의 병렬 조인 방법과 마찬가지로 부하의 불균형을 초래할 수 있다. 또한 이 방법은 조인의 결과 릴레이션의 데이터 편재를 고려하고 있지 않기 때문에 결과를 처리하는 데 발생하는 부하의 불균형을 초래할 수 있다.

본 논문에서는 샘플링 기법과 [13]에서 제안된 히스토그램 균등화 기법을 개량하여 조사 릴레이션의 AVS와 결과 릴레이션의 JPS를 고려한 부하의 균등 분산 방법과 이를 이용한 새로운 병렬 조인 알고리즘(SPJA : Sampling based Parallel Join Algorithm)을 제안한다.

본 논문에서 제안하는 병렬 조인 알고리즘은 조인하고자 하는 릴레이션들의 데이터 분포를 얻기 위하여 샘플링 기법을 이용한다. 물론 데이터베이스의 카탈로그 테이블에 해당 릴레이션의 데이터 분포 값이 이미 저장되어 있으면 이 과정은 생략될 수 있다. 하지만 그 분포 값이 카탈로그 테이블 내에 저장되어 있지 않다면 정확한 데이터 분포는 해당 릴레이션을 모두 탐색하여 얻을 수 있다. 그러나 일반적으로 데이터베이스 내에 릴레이션의 크기가 매우 크기 때문에, 전체 릴레이션의 탐색은 매우 큰 소모적인 디스크 입력 비용을 요구하기 때문에 비효율적이다. 샘플링 기법은 데이터의 일부분을 탐색하여 소모적인 디스크 입력 비용을 절감하고, 샘플링에 의하여 선택된 데이터 분포를 통하여 전체 데이터 분포를 예측하는 것이다. 또한 예측된 입력 릴레이션의 데이터 분포를 통하여 결과 릴레이션의 조인 속성의 데이터 분포를 예측하고, 이

를 통하여 각 속성 값에 대한 조인의 최종 비용을 산출한다. 그리고 산출된 조인 비용에 대하여 히스토그램 균등화 기법을 적용하여 각 노드가 균일한 조인 비용이 소요되도록 데이터를 분산하여 조인한다.

본 논문의 구성은 다음과 같다. 2장에서 샘플링을 이용하여 조인 비용을 예측하는 기법을 제시한다. 그리고 3장에서는 이를 기반으로 제안하고자 하는 부하의 균등 분배를 위한 방법과 병렬 조인 알고리즘을 기술한다. 4장에서는 성능 평가를 위한 모의 실험 모델을 제시하고, 모의 실험 결과를 5장에서 기술한다. 마지막으로 6장에서 결론을 기술한다.

2. 샘플링을 이용한 조인 비용 예측

대용량 데이터베이스에서의 데이터 크기는 매우 크기 때문에 조인 연산을 위해서 기존 대부분의 부하 균등 알고리즘[1,5,13,16]에서와 같이 데이터의 균등 분배를 위하여 시행되는 전체 릴레이션들의 탐색은 많은 입출력 비용이 발생하므로 문제점이 있다. 샘플링에 의하여 조인 연산의 입력 릴레이션과 결과 릴레이션의 데이터 분포를 비교적 정확하게 예측할 수 있다면 전체 릴레이션에 대한 탐색은 입출력 오버헤드를 고려하여 피해야 한다. 이 장에서는 조인 연산의 입력 릴레이션과 결과 릴레이션을 예측하기 위한 샘플링 기법과, 이 것을 이용하여 조인 연산 시에 각 노드 사이에 부하를 균등하게 분배하기 위하여 필요한 조인 비용을 계산하기 위한 비용 함수를 기술한다.

2.1 시스템 환경

본 논문에서의 병렬 해쉬 조인을 위한 시스템 구조로 대용량 데이터베이스 시스템으로 널리 사용되고 있는 SN(Shared-Nothing) 구조를 가정한다[10]. 조인할 두 릴레이션은 R과 S이며, $|R|$과 $|S|$를 해당 릴레이션의 크기라 하면, $|R| \leq |S|$라 가정한다. 즉, R은 구성 릴레이션이며, S는 조사 릴레이션이다. 조인 연산이 실행되기 전에 R과 S는 모든 노드에 중복되는 것이 없이 균등하게 나누어져 각 노드의 디스크에 저장되어 있다고 가정한다. 또한 시스템은 디스크 I/O, CPU 계산, 그리고 노드간에 데이터 전송이 동시(overlap)에 이루어지도록 독자적인 I/O 프로세서와 통신 프로세서 등을 갖는다[5]. 조인의 결과 튜플들은 결과 릴레이션을 구성하거나 다른 연산의 수행 시에 사용되기 위해서 노드간에 재분배되어 저장된다고 가정한다.

2.2 샘플링에 의한 데이터 분포 예측

조인 연산의 비용은 입력 릴레이션의 데이터 분포에 따라 영향을 받기 때문에 샘플링을 이용하여 입력 릴레이션의 분포를 정확하게 예측할 수 있다면 조인의 결과 릴레이션의 분포와 이에 따른 조인 비용을 정확하게 예측하는 것이 가능하다.

샘플링에 의한 데이터 분포의 예측 기법은 이미 통계학 분야에서 활발히 연구가 이루어진 분야이다[15]. 또한 샘플링 기법을 데이터베이스 시스템에 적용하는 연구도 활발히 진행되어 많은 연구 결과가 발표되었다[1,3,4,7,14]. 본 논문에서는 샘플링을 위한 새로운 방법을 제안하는 것이 아니라, 기존에 발표된 것들 중에 우리의 목적에 부합하고, 성능이 우수한 방법을 사용하고자 한다. [4]에서 기술되어 있는 t-cross 알고리즘은 원래 조인 연산의 선택율(selectivity)을 예측하는 방법으로, 본 논문에서 샘플링 방법을 사용하는 이유가 조인 비용을 정확하게 예측 하고자 함에 있기 때문에 우리의 알고리즘에 사용하기 위하여 적합한 방법이다. 이 방법은 샘플링의 각 단계에서 입력 릴레이션의 튜플들을 임의적(randomly)이고 균등하게(uniformly)하게 선택하여 샘플의 집합을 구하여 각 샘플 집합의 조인 선택율을 계산하여 조인 선택율의 기대값을 구한다. 이와 같은 과정은 각 샘플링 단계에서의 계산한 선택율로 유도되는 선택율의 평균 및 분산 값이 주어진 에러 범위 내에서 변동이 없을 때까지 반복된다[4]. 본 논문에서는 t-cross 방법을 통하여 샘플링이 완료되면 샘플들의 집합으로부터 조인 연산의 입력과 결과 릴레이션의 데이터 분포를 구하고 이를 조인 비용 계산에 이용한다.

다음은 이 기법을 사용하여 결과 릴레이션의 분포를 구하고 각 조인 속성의 값에 의한 조인 비용을 계산하는 방법을 기술한다. 본 논문에서 고려하고자하는 연산은 릴레이션 R과 S의 조인 속성 α에 대한 equi-join이다. 릴레이션 X의 조인 속성 α의 도메인을 $D=\{d_k | 1 \leq k \leq K\}$이라 하고, 값 d_k의 빈도수 $f_X(d_k)$는 α의 값이 d_k인 릴레이션 X에서의 튜플들의 개수를 의미한다고 하자. 이때 조인 연산의 결과 릴레이션 J에서의 d_k의 빈도수는 다음과 같다.

$$f_J(d_k) = f_R(d_k) \times f_S(d_k) \quad (1)$$

버퍼의 갱신을 방법에 의하여 입력 릴레이션의 데이터 분포를 예측할 수 있고, 그 결과를 이용하여 수식 (1)에 의하여 결과 릴레이션의 데이터 분포를 구할 수 있다.

2.3 조인 비용 함수 모델

부하 균등 알고리즘의 목적은 데이터 편재의 영향하에서도 병렬 처리에 참여하는 노드의 부하를 균등하게 하는 것이다. 따라서 이들의 영향을 고려한 조인 비용 함수의 설계가 필요하다. 이 장에서는 데이터 편재의 영향을 고려한 조인 비용 함수 모델을 제시한다. 이 장에서 기술하는 조인 비용 함수는 본 논문에서 제안된 병렬 조인 알고리즘에서의 부하의 균등한 분배를 위하여 사용된다.

2.3.1 데이터 편재의 영향

조인 연산의 비용이 데이터의 편재에 영향을 받는다는 것은 여러 연구의 결과로 입증되었다[6,11]. 특히, 병렬 해쉬 조인의 성능은 데이터 편재의 영향하에서 그 성능이 급격하게 저하된다는 것이 일반적으로 알려져 있다[5,6,13]. 그것은 데이터 편재가 각 노드에서 처리되는 튜플 수의 불균형을 초래하여, 전체 병렬 처리의 성능이 가장 과부하가 걸린 노드의 수행 시간에 의하여 결정되기 때문이다. 조인 연산의 부하 불균형을 초래하는 데이터 편재는 다음과 같은 두 가지로 분류할 수 있다[4,11].

● AVS :

AVS는 입력 릴레이션의 조인 속성의 불균형 데이터 분포에 의하여 발생된다. 불균형 데이터 분포는 데이터의 불균형 할당 및 분산을 초래하여, 조인 연산 시에 노드 사이에 불균등한 작업 부하의 할당, 편중된 데이터들이 하나의 노드에 할당되는 경우에 발생하는 버켓 오버플로우, 그리고 특정 노드에 데이터 전송이 집중되는 네트워크 부하의 증가를 초래한다.

● JPS :

JPS는 서로 다른 노드에서 처리되는 데이터의 조인 선택율(join selectivity)들이 같지 않음으로 인해서 발생한다. JPS는 AVS에 의하여 영향을 받을 수 있지만, 일반적으로 AVS와는 무관하게 발생할 수 있다. 이것은 조인 연산의 결과 처리를 위하여 조인 릴레이션을 분할하는 방법에 의하여 조인 튜플들을 생산하는 특정 노드에 CPU 처리비용, 디스크로의 출력, 또는 네트워크를 통한 통신비용에 의한 과중한 부하를 초래한다. 결국 JPS는 조인 연산 시에 특정 노드에 의한 부하 불균형을 초래하여 성능의 저하를 초래한다.

2.3.2 비용 함수를 위한 매개변수

조인 연산의 성능에 영향을 주는 매개변수들은 다음과 같으며, 이 매개변수들은 4장에서의 조인 연산의 성능 측정을 위한 모의 실험 모델을 기술할 때에도 같은 의미로 사용된다.

● 작업부하에 관한 매개변수

$|R|$: 릴레이션 R의 크기(단위 : 튜플 수)

$|S|$: 릴레이션 S의 크기(단위 : 튜플 수)

t : 튜플의 크기(단위 : 바이트 수)

● 시스템에 관한 매개변수

P : 노드(프로세서)의 수

M : 각 노드의 메모리 용량(단위 : 튜플 수)

μ : 프로세서의 처리율(단위 : MIPS)

ω_{io} : 디스크의 I/O 대역폭(단위 : Bytes/초)

ω_{comm} : 각 노드의 통신 채널 대역폭(단위 : Bytes/초)

I_{cpu} : 조인 연산의 임의의 과정에서 한 튜플을 처리하기 위한 CPU 명령어 개수

2.3.3 조인 비용 함수

AVS를 고려한 부하 균등 알고리즘에 관한 여러 연구의 결과들이 발표되었지만[1,5,13], JPS를 고려한 알고리즘에 관한 연구는 미비한 실정이다. 각 노드에서의 최종적인 조인 연산의 수행 시간은 AVS와 JPS의 영향을 받기 때문에, 이들의 영향을 고려한 조인 비용 함수의 설계가 필요하다. 이 절에서 기술하는 조인 비용 함수는 본 논문에서 제안하고자 하는 병렬 조인 알고리즘에서의 부하의 균등한 분배를 위하여 필요하다. 조인 비용 함수 $T_{join}(d)$는 속성 값이 d인 튜플들을 할당받아 처리하는 노드에 해당 튜플들에 의한 조인 연산 비용을 의미한다.

$T_{join}(d)$는 구성 릴레이션 R의 해쉬 테이블을 구성하는 시간과 조사 릴레이션의 S의 해쉬 테이블 탐색

시간, 그리고 조인의 결과 튜플을 처리하는 시간의 합으로 구성된다. 즉,

$$T_{join}(d) = T_{join\_hash}(d) + T_{join\_probe}(d) + T_{join\_result}(d).$$

$T_{join}(d)$를 구성하는 각 처리 시간을 유도하기 위한 과정은 다음과 같다.

(1) 해쉬 테이블을 구성하는 시간

본 논문에서는 각 단계에서의 CPU 시간과 통신 시간, 그리고 I/O 시간의 중첩(overlap)이 완벽하다고 가정하였기 때문에, $T_{join_hash}(d)$는 CPU 처리 시간과 디스크 I/O 시간 중에 최대 처리 시간을 가진 값으로 한다. 즉,

$$T_{join\_hash}(d) = \max(T_{join\_hash\_io}(d),\ T_{join\_hash\_cpu}(d)).$$

여기서, $T_{join\_hash\_io}(d) = \dfrac{f_{R(d)} \cdot t}{\omega_{io}}$,

$$T_{join\_hash\_cpu}(d) = \frac{f_R(d) \cdot I_{cpu}}{\mu} \text{ 이다.}$$

(2) 해쉬 테이블 탐색 시간

조사 릴레이션 S의 해쉬 테이블 탐색 시간은 다음과 같다.

$$T_{join\_probe}(d) = \max(T_{join\_probe\_io}(d),\ T_{join\_probe\_cpu}(d)).$$

이때 I/O 시간과 CPU 처리 시간은 AVS에 의해서 구성 릴레이션에서 속성 값 d를 갖는 튜플들의 크기가 메모리 크기를 초과하는 버켓 오버플로우가 발생하는 경우와 그렇지 않는 경우로 나누어 질 수 있다.

- 버켓 오버플로우가 발생하는 경우($M < f_R(d)$)

$$T_{join\_probe\_io}(d) = \frac{\lceil \frac{f_R(d)}{M} \rceil \cdot f_s(d) \cdot t}{\omega_{io}},$$

$$T_{join\_probe\_cpu}(d) = \frac{\lceil \frac{f_R(d)}{M} \rceil \cdot f_s(d) \cdot I_{cpu}}{\mu}$$

- 버켓 오버플로우가 발생되지 않는 경우($M \geq f_R(d)$)

$$T_{join\_probe\_io}(d) = \frac{f_s(d) \cdot t}{\omega_{io}},$$

$$T_{join\_probe\_cpu}(d) = \frac{f_s(d) \cdot I_{cpu}}{\mu}$$

(3) 조인 결과 튜플 처리 시간

각 노드에 국부적으로 조인된 결과 튜플들은 결과 릴레이션을 구성하거나, 다른 연산에서의 사용을 위해서 네트워크를 통하여 적절한 노드에 전송되어 저장된다. 즉,

$$T_{join\_result}(d) = \max(T_{join\_result\_cpu}(d),\ T_{join\_result\_comm}(d)).$$

여기서, $T_{join\_result\_cpu}(d) = \dfrac{f_J(d) \cdot I_{cpu}}{\mu}$,

$$T_{join\_result\_comm}(d) = \frac{f_J(d) \cdot t}{\omega_{comm}}$$

3. 병렬 조인 알고리즘

이 장에서는 AVS와 JPS를 고려한 부하의 균등 분산을 위한 방법과 이를 이용한 새로운 병렬 조인 알고리즘인 SPJA를 제안한다. 조인 연산을 각 노드에서 국부적으로 수행하기 위하여 노드간의 부하를 균등히 하도록 데이터를 재분배하는 것이 필요하다. 이 장에서는 먼저 데이터 재분배를 위한 데이터 분산 틀(data distribution framework)을 설명한다. 그리고 데이터 분산 틀(framework)을 이용하여 부하 균등 조인 연산을 수행하기 위한 알고리즘의 과정을 기술한다.

3.1 데이터 분산 틀

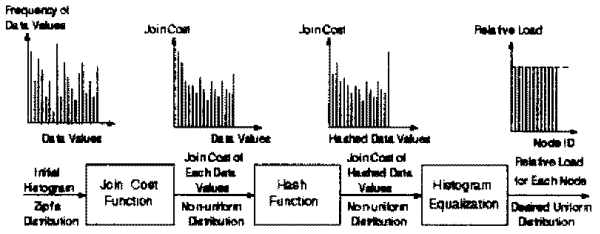
본 논문에서의 데이터 분산 틀은 [13]에서 제안한 히스토그램 균등화 기법을 이용한다. 하지만 [13]에서 제안한 방법은 구성 릴레이션의 AVS 만을 고려하였고, 구성 릴레이션 전체를 탐색하여야 하므로 비효율적이다. 본 논문에서는 구성 릴레이션의 AVS 뿐만 아니라 조사 릴레이션의 AVS와 JPS를 고려한 효율적인 데이터 분산 틀을 제안하고자 한다. 제안한 방법은 또한 샘플링 기법을 적용하여 조인 연산의 수행 중에 발생할 수 있는 소모적인 디스크 탐색 시간을 줄인다.

[13]에서 제안한 히스토그램 균등화 기법을 소개하면 다음과 같다. $\bar{x}$를 이산확률변수(discrete random variable) 그리고 $g(x)$를 단조증가 변환함수라 하자. 그리고 입력확률변수 $\bar{x}$가 $x_1, x_2, \cdots, x_J (x_1 \leq x_2 \leq \cdots \leq x_J)$와 같은 값을 갖고, 출력확률변수 $\bar{y}$가 $y_1, y_2, \cdots, y_K$

$(v_1 \le v_2 \le \cdots \le v_K)$와 같은 값을 가질 때 히스토그램 균등화 과정은 변환 $\bar{y}=g(\bar{x})$에 의하여 출력확률분포가 균등분포(uniform density)가 되도록 $\bar{x}$가 $\bar{y}$로 사상되는 것을 의미한다. $H_{\bar{x}}(x)(x=x_1,x_2,\cdots,x_j)$를 입력 값의 발생빈도의 비율이라 하면, 히스토그램 균등화를 위한 변환 함수는 다음과 같다.

$$y=g(x)=(y_K-y_1)\sum_{m=x_1}^{x} H_{\bar{x}}(m)+y_1 \qquad (2)$$

식 (2)의 의미는 임의의 히스토그램 분포를 갖는 입력 변수 x를 변환 함수를 이용하여 출력 변수 y로 사상하게 되면, 출력 변수 y의 히스토그램은 균등하게 분포됨을 의미한다.



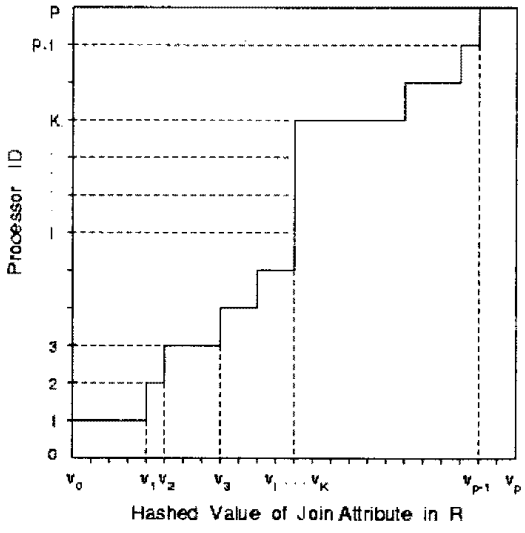
(그림 1) 부하 균등 분산을 위한 데이터 분산 틀

본 논문에서 제안하는 데이터 분산 틀의 전체적인 구도는 (그림 1)과 같다. 먼저 샘플링 기법을 이용하여 입력 릴레이션 R과 S의 데이터 분포를 얻는다. 그리고 이 분포를 2장에서 제시한 조인 비용 함수에 적용하여 각 데이터 값에 대한 조인 비용을 계산한다. 그리고 각 데이터 값에 해쉬 함수를 적용하여 각 해쉬 값에 대한 조인 비용을 얻는다. 그리고 최종적으로 [13]에서 제안한 히스토그램 균등화 기법을 적용하여 각 노드에 균등화된 부하를 제공하도록 데이터를 재분배한다. 즉, 임의의 입력 데이터 분포가 주어지면, 우리는 히스토그램 균등화 변환 방법을 이용하여 노드 사이에 부하를 균등하게 하도록 하는 데이터 분산을 얻는다. 이때 입력확률변수는 $\bar{x}$는 h개의 조인 속성의 해쉬 결과 값을 가질 수 있으며, 출력확률변수 $\bar{y}$는 노드의 $id(1,2,\cdots,P)$를 의미한다. 이때 히스토그램 균등화 함수는 식 (2)로부터 다음과 같이 얻을 수 있다.

$$y=g(x)=\lceil P\sum_{m=0}^{x} H_{\bar{x}}(m) \rceil \qquad (3)$$

3.2 튜플 분산을 위한 경계값 추출

식 (3)의 변환함수가 단조증가 함수이기 때문에 i번째 노드를 위한 상위 경계값 v_i는 $g^{-1}(i)$를 통해서 구할 수 있다. 즉, 릴레이션 R의 조인 속성의 가장 작은 해쉬 결과 값을 v_0라하고, P개의 경계값을 $v_1,v_2,\ldots,v_P$라 하면, 조인 속성의 해쉬 결과 값이 a인 튜플이 $v_{i-1}<a\le v_i\ (1<i<P)$인 경우에 그 튜플은 노드 i로 분배된다. (그림 2)는 변환 함수로부터 경계값을 결정하는 예를 보인다. 만일 조인 속성의 해쉬 결과 값이 경계값에서 많은 중복이 발생하는 경우에 두 개 이상의 노드가 같은 경계값을 갖는 경우가 발생할 수 있다. 즉, $i\neq j$일 때 $g^{-1}(i)=g^{-1}(j)$인 경우가 발생할 수 있다. 이때에는 같은 경계값을 갖는 노드가 균등하게 튜플들을 분배하는 비율로 그 경계값을 갖는 튜플들은 할당된다.



(그림 2) 히스토그램 균등화를 위한 변환 함수

3.3 알고리즘 수행 과정

새로운 병렬 조인 알고리즘인 SPJA는 다음과 같은 세 단계의 수행 과정으로 이루어진다.

샘플링 단계 :

- 단계 1.1 각 노드는 R과 S를 샘플링 하여 얻어진 국부적인 히스토그램을 방송하고, 다른 프로세서로부터 전송된 히스토그램을 더하여 R과 S의 전체적인 히스토그램을 예측한다.
- 단계 1.2 각 노드는 조인 비용 함수를 이용하여 조

인 속성 값에 대한 조인 비용을 계산하고, 조인 속성에 해쉬 함수 H_1을 적용하여 해쉬 결과 값에 대한 조인 비용을 구한다.

- 단계 1.3 각 노드는 히스토그램 균등화 함수를 이용하여 각 노드에 튜플의 분배를 위한 경계값을 구한다.

분할 단계 :

- 단계 2.1 각 노드는 국부적 디스크에 저장된 릴레이션 R과 S의 부분을 차례로 디스크로부터 읽어서 각 튜플의 조인 속성에 해쉬 함수 H_1을 적용한다.
- 단계 2.2 단계 2.1의 해쉬 결과값과 단계 1.3의 경계값을 이용하여 두 릴레이션을 분할하고 해당 노드에 전송한다. 즉, 이 단계의 수행 결과 각 노드 i는 분배된 두 릴레이션 R_i와 S_i로 재구성된다.

조인 단계 :

각 노드 i는 자신에게 전송된 두 릴레이션 R_i와 S_i에 GRACE 해쉬 조인 방법을 적용하여 국부적으로 조인 연산을 수행한다.

앞에서 기술한 내용을 기반으로 각 노드에서 병렬로 실행되는 SPJA의 가상코드를 표현하면 (그림 3)과 같다.

```
SPJA Algorithm /* parallel algorithm on each node */

Begin
 /* Sampling phase */
 calls t-cross procedure;
  /* sampling algorithm for estimating data distributions
     of the local partition of R and S */
 broadcast the estimated local data distribution;
 evaluate global data distributions of R and S;
 evaluate the join cost of each data value from the
   distributions;
 evaluate the join cost for each hashed data value using
   a hash function H1;
 select the boundary values for tuple redistribution;
   /* using the histogram equalization algorithm */

 /* Partitioning phase */
 read the local partition of R and S, page by page;
 For every tuple t in the partition
      hashed_value = H1(t);
          /* apply a hash function H1 to t */
          /* distribute the tuples among the nodes using
             the boundary values */
      if( (Id =Find_Node(hashed_value)) == My_Node)
          write t into the local disk;
      else
          transmit t to Node Id;
 End_For
 write the tuples transmitted from the other nodes into
 its local disk;

 /* Joining phase */
 perform the local join using the GRACE join algorithm;
end
```

(그림 3) SPJA의 가상 코드

SPJA는 기존의 부하 균등 알고리즘[5,13,16,17]들에 비하여 다음과 같은 특징을 갖는다. 기존의 알고리즘들은 전체 릴레이션의 소모적인 탐색을 통하여 데이터를 분산하기 위한 준비 단계를 수행한다. 그러나 SPJA는 샘플링 기법을 이용하기 때문에 준비 단계에서의 디스크 I/O 및 CPU 시간을 기존의 방법에 비하여 절약할 수 있다. 또한 기존의 방법은 최종적인 조인을 위하여 각 노드에 재분배되는 데이터의 양을 균등하게 하는 접근 방법을 사용한다. 이와 같은 방법을 이용한 기존의 알고리즘들의 문제점은 AVS에 의하여 발생하는 버켓 오버플로우와 각 노드에 할당된 데이터의 조인 선택율의 차이에서 발생하는 성능 저하를 처리할 수 없다는 것이다. 그러나 SPJA는 해쉬 결과 값에 대한 조인 비용을 예측하고 히스토그램 균등화 기법을 이용하여 각 노드 사이에 부하가 균등하게 되도록 데이터를 재분배한다. 따라서 병렬 조인 시에 어떠한 종류의 데이터 편재에도 영향을 받지 않고 기존의 방법에 비하여 우수한 성능을 제공한다.

4. 모의 실험 모델

데이터 편재 하에서 부하 균등을 고려하지 않은 병렬 조인 알고리즘 중에서 대표적인 방법이 GRACE 병렬 해쉬 조인 알고리즘(PHJA : Parallel Hash Join Algorithm)이다. 그리고 AVS를 고려한 방법들 중에서 성능이 우수하다고 알려진 방법은 [5]에서 제안된 부하 균등 알고리즘(LBJA : Load Balancing Join Algorithm)이다. 따라서 이 장에서는 데이터 편재를 가정하여 기존의 병렬 해쉬 조인 알고리즘(PHJA)과 기존의 대표적인 부하 균등 알고리즘(LBJA), 그리고 본 논문에서 제안된 SPJA의 성능을 구하기 위한 모의 실험 모델을 제시한다.

병렬 조인 알고리즘들의 성능을 조인 연산에 소요되

총 수행 시간으로 나타낼 수 있다. 모의 실험을 통한 성능 측정은 병렬 조인 알고리즘을 단계별로 실제로 수행하여 각 성능 요소 값을 구하고, 그 값을 비용 함수에 대입하여 조인 연산에 소요되는 총 수행 시간을 계산함으로써 이루어진다. 따라서 병렬 조인 알고리즘의 총 수행 시간은 각 단계별 수행 시간의 합으로 나타낼 수 있다. 그리고 각 단계별 수행 시간은 CPU 사용 시간, 디스크 사용 시간, 그리고 통신에 필요한 시간에 의해서 결정된다. 본 논문에서 각 단계에서의 CPU 시간과 통신 시간, 그리고 I/O 시간의 중첩(overlap)이 완벽하다고 가정하였기 때문에, 단계 i의 수행 시간 $T_i(d)$는 CPU 처리 시간, 디스크 I/O 시간, 통신 시간 중에 최대 처리 시간을 가진 값으로 한다. 즉, $T_i = \max(T_{i_io}, T_{i_cpu}, T_{i_comm})$이며, 총 수행 시간은 $T = \sum T_i$이 된다.

모의 실험 모델의 비용 함수에 사용되는 성능 요소 및 매개변수들은 2.3절의 2.3.2절에서 정의한 것을 같은 의미로 사용한다. 그리고 모의 실험 과정 중에 측정되는 매개변수는 다음과 같이 정의한다.

● 모의 실험에 의한 측정 매개변수

R_{max_init} : 조인 연산 전에 각 노드에 저장되어 있는 R의 튜플 수 중에서 최대값

S_{max_init} : 조인 연산 전에 각 노드에 저장되어 있는 S의 튜플 수 중에서 최대값

$N_{max\_io\_split}$: 분할 단계에서 가장 많은 디스크 I/O가 발생하는 노드의 I/O 튜플 수

$N_{max\_comm\_split}$: 분할 단계에서 가장 많은 통신이 발생하는 노드에서 송·수신되는 튜플 수

$N_{max\_io\_part}$: 분배 단계에서 가장 많은 디스크 I/O가 발생하는 노드의 I/O 튜플 수

$N_{max\_comm\_part}$: 분배 단계에서 가장 많은 통신이 발생하는 노드에서 송·수신되는 튜플 수

R_{max_final} : 분배 단계 후에 가장 많은 데이터가 분배된 노드에 분배된 R의 튜플 수

S_{max_final} : 분배 단계 후에 가장 많은 데이터가 분배된 노드에 분배된 S의 튜플

N_{max_join} : 가장 많은 데이터가 분배된 노드에서 조인이 이루어진 튜플 수

N_{max_samp} : 가장 많은 튜플을 샘플링한 노드에서의 샘플링 튜플 수

모의 실험을 통해서 비교하고자 하는 병렬 조인 알고리즘들의 비용 함수들은 다음과 같이 유도된다.

4.1 PHJA의 비용 함수

PHJA의 전체 처리 시간은 릴레이션을 읽어 해쉬 함수를 적용하여 각 노드에 데이터를 분산하는 분산 시간과 각 노드에서 분산된 데이터를 국부적으로 조인하는 시간의 합으로 구성된다. 즉, 총수행 시간은 $T_{PHJA} = T_{split} + T_{join}$이고, 각 단계별 수행 시간은 다음과 같다.

4.1.1 분할 단계의 수행 시간

분할 단계의 수행 시간은 릴레이션을 읽어서 해쉬 함수를 실제로 적용하고 릴레이션을 분할하여서, 각 노드에 분배되는 튜플의 수를 구하여 얻을 수 있다. 즉,

$$T_{split} = \max(T_{split\_io}, T_{split\_cpu}, T_{split\_comm}).$$

여기에서, $T_{split\_io} = \frac{N_{max\_io\_split} \cdot t}{\omega_{io}}$,

$$T_{split\_cpu} = \frac{(R_{max\_init} + S_{max\_init}) \cdot I_{cpu}}{\mu},$$

$T_{split\_comm} = \frac{N_{max\_comm\_split} \cdot t}{\omega_{comm}}$ 이다.

4.1.2 조인 단계의 수행 시간

조인 단계의 수행 시간은 2.3절에서 조인 비용 함수를 구한 과정과 유사한 과정을 통하여 유도할 수 있다. 각 노드에서의 국부적 조인을 위한 수행 시간은 구성 릴레이션 R의 튜플들로부터 해쉬 테이블을 구성하는 시간, 조사 릴레이션 S의 튜플들의 해쉬 테이블 탐색 시간, 그리고 조인의 결과 튜플들을 처리하는 시간의 합으로 구성된다. 즉,

$$T_{join} = T_{join\_hash} + T_{join\_probe} + T_{join\_result}.$$

T_{join}을 구성하는 각 처리 시간을 유도하기 위한 과정은 다음과 같다.

(1) 해쉬 테이블 구성 시간

해쉬 테이블을 구성하는 시간 T_{join_hash}는 CPU 처

내 시간과 디스크 I/O 시간들 중에서 최대 처리 시간을 가진 값으로 한다. 즉,

$$T_{join\_hash} = \max(T_{join\_hash\_io}, T_{join\_hash\_cpu}),$$

여기에서, $T_{join\_hash\_io} = \frac{R_{max\_final} \cdot t}{\omega_{io}}$,

$T_{join\_hash\_cpu} = \frac{R_{max\_final} \cdot I_{cpu}}{\mu}$ 이다.

(2) 해쉬 테이블 탐색 시간

조사 릴레이션을 튜플들에 해쉬 함수를 적용하여 그 결과 값을 가지고 메모리 해쉬 테이블을 탐색하는 시간 T_{join_probe}는 다음과 같다.

$$T_{join\_probe} = \max(T_{join\_probe\_io}, T_{join\_probe\_cpu}).$$

이때 I/O 시간과 CPU 처리 시간은 AVS에 의해서 구성 릴레이션의 버켓들의 크기가 메모리 크기를 초과하는 버켓 오버플로우가 발생하는 경우와 그렇지 않은 경우로 나누어 질 수 있다.

● 버켓 오버플로우가 발생하는 경우

구성 릴레이션 R의 오버플로우 버켓의 수를 k개라고 하자. 또한 $R_i (i=1,2,\cdots,k)$를 릴레이션 R의 오버플로우 버켓이라 하고 S_i를 R_i에 대응하는 릴레이션 S의 버켓이라고 하자. 그리고 $|R_i|$와 $|S_i|$를 각각 R_i와 S_i에 포함된 튜플 수라 하자. 이때 R_i는 메모리 용량을 초과하기 때문에 R_i에 대응되는 조사를 위한 버켓 S_i에 대한 처리는 $\lceil \frac{|R_i|}{M} \rceil$ 번 반복되어 수행된다. 즉,

$$T_{join\_probe\_io} = \frac{S_{max\_final} \cdot t}{\omega_{io}} + \frac{(\sum_{i=1}^{k} |S_i| (\lceil \frac{|R_i|}{M} \rceil - 1)) \cdot t}{\omega_{io}},$$

$$T_{join\_probe\_cpu} = \frac{S_{max\_final} \cdot I_{cpu}}{\mu} + \frac{(\sum_{i=1}^{k} |S_i| (\lceil \frac{|R_i|}{M} \rceil - 1)) \cdot I_{cpu}}{\mu}.$$

● 버켓 오버플로우가 발생하지 않는 경우

$$T_{join\_probe\_io} = \frac{S_{max\_final} \cdot t}{\omega_{io}},$$

$$T_{join\_probe\_cpu} = \frac{S_{max\_final} \cdot I_{cpu}}{\mu}.$$

(3) 조인 결과 튜플 처리 시간

각 노드에서 국부적으로 조인된 결과 튜플들은 결과 릴레이션을 구성하거나, 다른 연산에서의 사용을 위해서 네트워크를 통하여 적절한 노드에 전송된다. 즉,

$$T_{join\_result} = \max(T_{join\_result\_cpu}, T_{join\_result\_comm}).$$

여기서, $T_{join\_result\_cpu} = \frac{N_{max\_join} \cdot I_{cpu}}{\mu}$,

$T_{join\_result\_comm} = \frac{N_{max\_join} \cdot t}{\omega_{comm}}$.

4.2 SPJA의 비용 함수

SPJA의 전체 처리 시간은 샘플링 단계, 분할 단계, 그리고 조인 단계의 각 단계별 처리 시간의 합으로 구성된다. 즉, 총 수행 시간은 $T_{SPJA} = T_{samp} + T_{split} + T_{join}$ 이고, 각 단계별 수행 시간은 다음과 같다.

4.2.1 샘플링 단계의 수행 시간

각 노드에서의 샘플링 단계의 수행 시간은 샘플링을 위하여 샘플 데이터를 디스크로부터 읽어들이는 I/O 시간과 CPU 처리 시간으로 구성된다. 여기에서 샘플링에 의하여 구성된 각 노드에서의 국부적 데이터 히스토그램을 전송하는 통신 시간과 히스토그램 균등화를 위한 CPU 처리 시간은 전체 처리 시간에 비해서 매우 작은 시간이기 때문에 무시한다. 따라서 샘플링을 위한 수행 시간은 $T_{samp} = \max(T_{samp\_io}, T_{samp\_cpu})$ 이다.

여기서, $T_{samp\_io} = \frac{N_{max\_samp} \cdot t}{\omega_{io}}$,

$T_{samp\_cpu} = \frac{N_{max\_samp} \cdot I_{cpu}}{\mu}$ 이다.

4.2.2 분할 단계의 수행 시간

PHJA의 분할 단계의 비용 함수와 동일하다.

4.2.3 조인 단계의 수행 시간

PHJA의 조인 단계의 비용함수와 동일하다.

4.3 LBJA의 비용 함수

LBJA 조인 방법의 전체 처리 시간은 분할 단계, 분배 단계, 버켓 튜닝 단계, 조인 단계의 각 단계별 처리 시간의 합으로 구성된다[5]. 그런데 버켓 튜닝 단계의 수행 시간은 다른 단계의 시간에 비해서 매우 작은 시간이므로 무시한다. 즉, 총 수행 시간은 $T_{LBJA} = T_{split} + T_{part} + T_{join}$ 이고, 각 단계별 수행 시간은 다음과 같다.

4.3.1 분할 단계의 수행 시간

분할 단계에서 각 노드는 디스크로부터 해당 릴레이션을 읽어, 해쉬 함수를 적용해 버켓을 분할하여, 분할된 버켓을 디스크에 다시 저장한다. 분할 단계의 수행 시간은 다음과 같다.

$$T_{split} = \max(T_{split\_io}, T_{split\_cpu}).$$

$$T_{split\_io} = 2 \cdot \frac{(R_{max\_init} + S_{max\_init}) \cdot t}{\omega_{io}},$$

$$T_{split\_cpu} = \frac{(R_{max\_init} + S_{max\_init}) \cdot I_{cpu}}{\mu}.$$

4.3.2 분배 단계의 수행 시간

분배 단계에서는 파티션 튜닝 과정을 통하여 노드 사이에 버켓의 분배가 일어난다[5]. 분배 단계 수행시간은 다음과 같다.

$$T_{part} = \max(T_{part\_io}, T_{part\_comm}).$$

$$T_{part\_io} = \frac{N_{max\_io\_part} \cdot t}{\omega_{io}},$$

$$T_{part\_comm} = \frac{N_{max\_comm\_part} \cdot t}{\omega_{comm}}.$$

4.3.3 조인 단계의 수행 시간

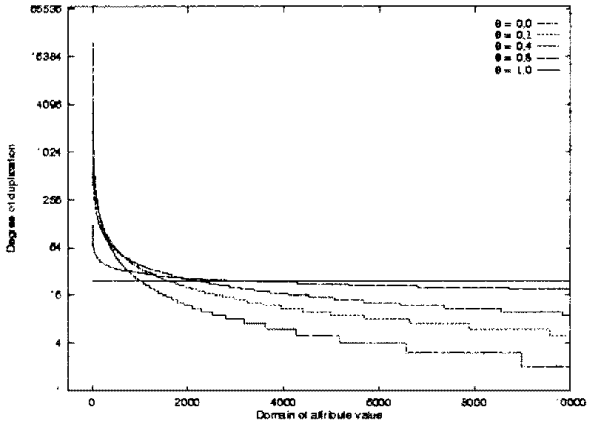
PHJA의 조인 단계의 비용함수와 동일하다.

5. 모의 실험 결과

실제적인 데이터베이스에서 불균일한 데이터 분포가 Zipf 분포를 갖는다는 것이 여러 연구의 결과로 알려져 있기 때문에 모의 실험을 위한 샘플 데이터베이스는 Zipf 분포에 따르도록 생성하였다[5,6,13]. 이 분포에서는 N개의 가능한 값 중에 i $(1 \le i \le N)$번째 값이 중복된 값을 가질 확률이 다음 식으로 주어진다.

$$p_i = \frac{c}{i^{1-\theta}}, \text{ 여기서 } c = \frac{1}{H_N^{1-\theta}}, H_N^{1-\theta} = \sum_{i=1}^{N} \frac{1}{i^{1-\theta}} \quad (4)$$

위 식에서 변수 θ 값을 변화시킴으로써 불균일의 정도를 증감시킬 수 있다. 즉, $\theta = 1$이면 균일 분포를 나타내며, θ가 0에 가까워 질 수록 불균일의 정도가 점점 증가한다. (그림 4)는 θ 값의 변화에 대한 Zipf 분포의 예를 보인다.



(그림 4) Zipf 분포의 예

모의 실험은 $\theta = 0$(심한 불균일), $\theta = 0.2$(완화된 불균일), 그리고 $\theta = 1$(균일한 분포)의 세 가지 경우의 데이터 분포를 갖는 샘플 데이터베이스를 만들어 수행하였다. 즉, 모의 실험을 위한 샘플 데이터베이스는 수식 (4)를 사용하여 무작위 수를 생성하여 조인 속성값을 얻음으로써 생성하였다. 모든 실험에서 두 릴레이션은 같은 Zipf 데이터 분포를 갖는다고 가정하였다.

모의 실험은 SUN사의 SPARCstation 20 시스템 상에서 수행되었고 모의 실험을 위한 코드는 C 언어로 작성하였다. 모의 실험은 실행 기반(execution-driven) 방식으로 진행되었다. 즉, Zipf 분포에 맞는 샘플 데이터베이스를 구성하고 각 단계 별로 알고리즘을 실제로 실행하여 4장에서 기술한 모의 실험에 의한 측정 매개변수 값을 구하고, 이를 다시 4장에서 기술한 비용 함수에 대입하여 조인의 전체 처리 시간을 계산하였다. 모든 모의 실험은 같은 조건하에서 5번 반복 실행하여 평균값을 계산하는 방식으로 진행하였다.

모의 실험을 위한 매개 변수 값은 다음과 같다. 릴레이션 R의 크기는 (그림 5)을 제외한 나머지 실험에

서 100만개의 튜플이며, 릴레이션 S의 크기는 모든 실험에서 릴레이션 R의 크기의 10배로 하였다. 또한 두 릴레이션에서 조인 속성의 가능한 값의 개수는 10000개로 하였다. 또한 각 노드는 하나의 프로세서로 구성된다고 가정하였다. 모의 실험에서 사용한 2.3절의 2.3.2절에서 정의한 매개 변수 값들은 다음과 같으며, 이 값들은 [5]에서의 모의 실험에서 사용한 값과 같다.

● 모의 실험에서의 매개변수 값

t : 200 바이트

P : (그림 6)을 제외한 다른 실험에서 20개

M : (그림 6)을 제외한 다른 실험에서 20000 튜플

μ : 20 MIPS

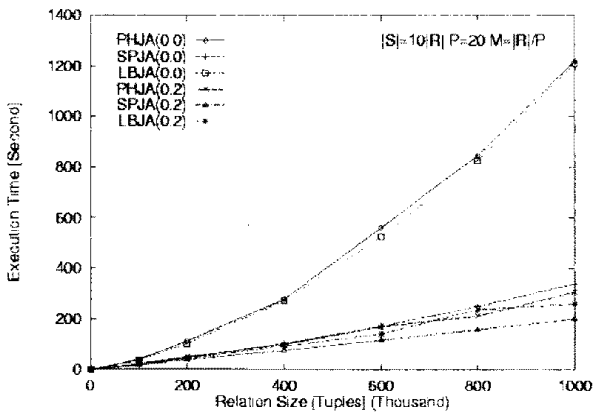
ω_{io} : 4 MBytes/초

ω_{comm} : 2.22 MBytes/초

I_{cpu} : 1000개 CPU 명령어

모의 실험은 PHJA, LBJA, 그리고 본 논문에서 제안된 SPJA의 성능을 비교하기 위하여 시행되었다. (그림 5)와 (그림 6)에서 괄호 안의 숫자는 θ값을 의미한다.

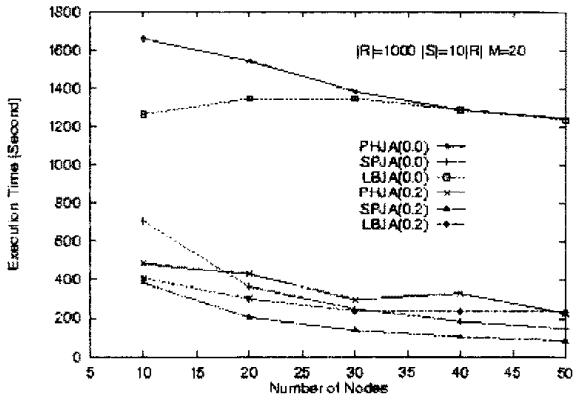
(그림 5)는 릴레이션의 크기 증가에 대한 수행 시간의 변화를 데이터 편재의 두 가지 상황(심한 불균일, 완화된 불균일)에 대하여 관찰한 모의 실험 결과를 그래프로 나타낸 것이다. 모든 경우에서 SPJA는 PHJA 및 LBJA에 비하여 우수한 성능을 보인다. 또한 SPJA는 불균일의 정도에 관계없이 릴레이션의 크기 증가에 비례하여 수행 시간이 선형적으로 증가한다. 그러나 PHJA 및 LBJA는 불균일의 정도가 커질 수록 SPJA에 비해서 상대적으로 수행 시간이 매우 크게 증가함



(그림 5) 릴레이션 크기의 변화에 대한 성능

을 알 수 있다. 이는 릴레이션의 크기의 증가에 의하여 데이터의 편재에 의한 영향을 SPJA는 거의 받지 않고 있으나, PHJA와 LBJA는 매우 민감하게 영향을 받고 있음을 알 수 있다. 이와 같은 현상은 릴레이션 크기 증가에 따라 AVS에 의한 버켓 오버플로우에 의한 비용과 JPS에 의한 조인 비용이 릴레이션의 크기 증가 비율 이상으로 증가하기 때문에 발생한다.

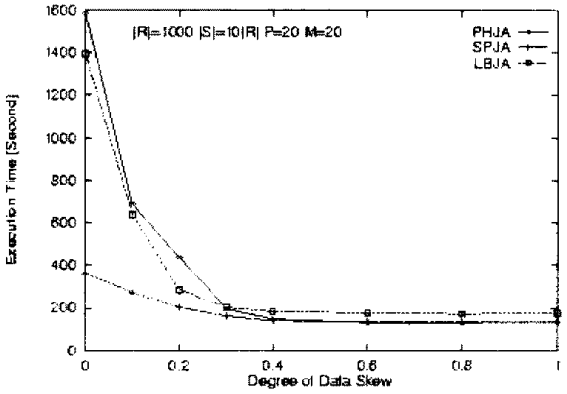
(그림 6)은 노드 수의 변화에 따른 수행 시간의 변화를 보이고 있다. (그림 5)에서와 같이 모든 경우에서 SPJA는 PHJA 및 LBJA에 비하여 우수한 성능을 보인다. 그리고 SPJA는 다른 알고리즘에 비해서 상대적으로 노드 수 증가에 따라 수행 시간이 단축되고 있음을 알 수 있다. 실제로 (그림 6)에서 심한 불균일의 경우에 SPJA는 노드 수가 10개에서 50개로 5배 증가함에 따라 수행 시간이 약 4.8배 단축됨을 보이고 있다. 이는 SPJA가 노드 수에 거의 비례하는 성능 향상을 얻을 수 있음을 의미한다. 즉, SPJA는 부하의 균등 분배를 통하여 다른 알고리즘에 비하여 노드 수의 증가에 따라 높은 병렬성을 갖는다. LBJA는 심한 불균일의 경우에 노드 수의 증가에 따라 성능이 향상되지 않음을 알 수 있다. 이는 데이터 양의 균등 분배를 기반으로 하는 기존의 부하 균등 방법이 심한 불균일 경우에서 AVS에 의한 버켓 오버플로우와 JPS에 의한 영향에 의하여 매우 낮은 병렬성을 가진다는 것을 의미한다.



(그림 6) 노드 수의 변화에 대한 성능

불균일 정도의 변화에 따른 수행 시간의 변화는 (그림 7)에 나타나고 있다. 불균일의 정도가 심해 질 수록 즉, θ값이 0.3 이하에서 PHJA와 LBJA의 수행 속도는 급격히 증가함을 알 수 있다. 이는 버켓 오버플로우와

JPS에 의한 영향이 얼마나 심각하게 PHJA와 LBJA의 성능에 영향을 주는 지를 알 수 있다. 데이터 편재가 있는 경우(θ값이 0.3 이하인 경우)에 SPJA 및 LBJA의 성능이 PHJA의 성능 보다 매우 우수함을 알 수 있다. 또한 데이터 분산이 균등 분포에 가까워지면서 LBJA는 준비 단계에서의 전체 릴레이션의 탐색 시간 때문에 성능이 가장 나쁨을 알 수 있다. 하지만 SPJA는 θ값이 0.3 이상인 경우에도 PHJA와 매우 근사하게 조금 큰 수행 시간을 보인다. 이는 데이터가 균등 분포로 가면서 샘플링의 비용이 절감되어 작은 수의 샘플로부터 데이터 분포를 정확하게 예측하는 것이 가능하기 때문에 성능이 PHJA와 거의 같기 때문이다.



(그림 7) 불균일 정도(θ)의 변화에 대한 성능

6. 결 론

본 논문은 SN 구조상에서의 AVS와 JPS 등과 같은 데이터가 편재된 상황하에서 부하의 균등 분산을 위한 데이터 분산 방법과 이를 기반으로 효율적인 병렬 조인 알고리즘을 제안하였다. 제안된 알고리즘은 샘플링 방법을 이용하여 데이터의 분할을 위한 준비 단계의 수행 시간을 최소화하였고, 각 조인 속성의 해쉬 값에 대한 조인 비용을 예측하고, 부하의 균등 분배를 통하여 데이터의 편재에 의한 상황하에서도 높은 병렬성에 의하여 조인 비용을 최소화하였다. 그리고 제안된 알고리즘의 성능 평가를 위하여 데이터 편재에 의한 부하 불균형과 버켓 오버플로우 문제를 고려한 모의 실험 모델을 제시하였다. 그리고 성능 평가를 위하여 Zipf 분포를 갖는 샘플 데이터베이스를 생성하여 기존의 병렬 해쉬 조인 알고리즘 및 기존의 대표적인 부하 균등 알고리즘과 제안된 알고리즘에 대하여 모의 실험을 수행하였다. 실험 결과 데이터의 편재가 심할 수록 제안된 알고리즘의 성능이 기존의 방법들보다 매우 우수함을 알 수 있었다.

참 고 문 헌

[1] D.J. DeWitt, J.F. Naughton, et. al., "Practical Skew Handling in Parallel Joins," in Proc. of the Int. Conf. on VLDB, pp.27-40, 1992.

[2] D.J. DeWitt and R. Gerber, "Multiprocessor Hash Based Join Algorithms," in Proc. of the Int. Conf. on VLDB, pp.151 164, 1985.

[3] S. Ganguly, P.B. Gibbons, and et. al., "Bifocal Sampling for Skew-Resistant Join Size Estimation," in Proc. of the ACM SIGMOD Conf., pp.271 281, 1996.

[4] P.J. Haas and A.N. Swami, "Sampling-Based Selectivity Estimation for Joins Using Augmented Frequent Value Statistics," in Proc. of Data Engineering, pp.522-531, 1995.

[5] K.A. Hua, C. Lee, and C.M. Hua, "Dynamic Load Balancing in Multicomputer Database Systems Using Partition Tuning," IEEE Trans. on Knowledge and Data Engineering, Vol.7, No.6, Dec., pp.968-983, 1995.

[6] M.S. Lakshmi and P.S. Yu, "Effectiveness of Parallel Joins," IEEE Trans. Knowledge and Data Engineering, Vol.2, No.4, pp.410-424, 1990.

[7] Y. Ling and W. Sun, "An Evaluation of Sampling-Based Size Estimation Methods for Selection in Database Systems," in Proc. of Data Engineering, pp.532-539, 1995.

[8] P. Mishra and M.H. Eich, "Join processing in relational databases," ACM Computing Surveys, pp.63 113, 1989.

[9] D.A. Schneider and D.J. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," in Proc. of the ACM Sigmod Conf., pp.110-121, 1989.

[10] S.Y.W. Su, *'Database Computers : Principles, Architectures, and Techniques,'* McGraw-Hill, 1988.

[11] C.B. Walton, A.G. Dale, and R.M. Jenevein, "A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins," in Proc. of the Int. Conf. on VLDB, pp.537-548, 1991.

[12] A. Papoulis, *'Probability, Random Variables, and Stochastic Processes,'* Mcgraw-Hill, 1991.

[13] U.K. Park, H.K. Choi, and T.G. Kim, "Uniform partitioning of relations using histogram equalization framework : An efficient parallel hash-based join," Information Processing Letters, Vol.55, No.5, pp.283-289, 1995.

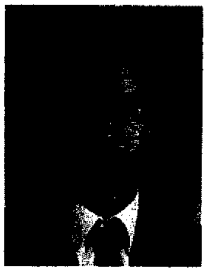
[14] V. Poosala and Y.E. Ioannidis, "Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balancing," in Proc. of the Int. Conf. on VLDB, pp.448-459. 1996.

[15] 한국통계학회 조사통계연구회 역, "Designing and Conducting Survey Research", 자유아카데미, 1996.

[16] 김용걸 외 4인, "비공유 병렬 데이터베이스 시스템에서 불균형 분포를 위한 부하균등 해쉬 조인 알고리즘", 한국정보과학회 논문지, 제22권, 제8호, pp. 1174-1182, 1995.

[17] 조석봉, 원영선, 홍만표, "하이퍼 큐 정렬을 이용한 병렬 결합 알고리즘", 한국정보과학회 논문지, 제24권, 제6호, pp.629-639, 1997.

[18] 임영모 외 4인, "하이퍼큐브 시스템을 위한 Cube-Robust 병렬 결합 알고리즘", 한국정보과학회 논문지, 제21권, 제3호, pp.528-537, 1994.



박 웅 규

e-mail : ukpark@dragon.seowon.ac.kr

1984년 서강대학교 전자공학과(공학사)

1986년 한국과학기술원 전기및전자공학과(공학석사)

1995년 한국과학기술원 전기및전자공학과(공학박사)

1986년~1990년 한국전자통신연구소 컴퓨터개발부 연구원

1991년~현재 서원대학교 전자계산학과 부교수

1997년~1998년 미국 Univ. of Central Florida 전자계산학과 교환교수

관심분야 : 병렬 데이터베이스, 멀티미디어 정보 처리, 인터넷 응용