NUMA 구조 기반의 효율적인 해시 조인 알고리즘†

최성준*, 김홍연*, 민준기*
*한국기술교육대학교 (KOREATECH) 컴퓨터공학부
e-mail:fumika@koreatech.ac.kr, zenweird@koreatech.ac.kr,
jkmin@koreatech.ac.kr

An efficient algorithm for hash-join on NUMA architecture

Seongjun Choe*, Hongyeon Kim*, Jun-Ki Min*
*School of Computer Science and Engineering, Korea University of Technology and Education (KOREATECH)

요 약

최근, 불균일 기억 장치 접근 (NUMA) 구조가 부각됨에 따라 NUMA 구조 기반의 관계 연산 기법들이 제안되고 있다. 본 논문에서는 NUMA 구조 기반의 효율적인 해시 조인 기법을 제안한다. NUMA 구조에서 조인 속성 값의 분포가 편중된 릴레이션들 간의 조인은 NUMA 노드들 간의 부하 불균형과 과도한 원격 메모리 접근을 발생시킬 수 있다. 제안 기법에서는 근사 히스토그램을 이용하여 조인 속성 값의 분포를 파악하고, 이를 기반으로 원격 메모리 접근을 줄이는 전달 방안을 제안한다. 실험에서는 입력 릴레이션들에 대해 조인 속성 값의 분포를 변화시키면서 제안 기법에 대한 성능을 평가한다.

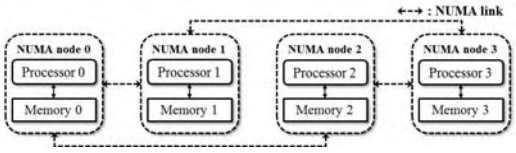
1. 서론

최근, 기존 대칭형 다중 처리 구조 (Symmetric Multi Processing: SMP)의 문제점인 프로세서들의 대기와 병목 현상을 완화시킬 수 있는 불균일 기억 장치 접근 구조 (Non-Uniform Memory Access: NUMA)가 부각되고 있다 [1-3]. NUMA 구조에서는 복수 개의 NUMA 노드 (node)들이 물리적으로 분산되어 있고, 각 노드는 1개 이상의 프로세서들과 하나의 공용 메모리로 구성된다 [4].

그림 1은 4개의 노드들로 구성된 NUMA 구조를 간략화한 예시이다. 그림 1에서 보는 바와 같이, 각 NUMA 노드의 프로세서들은 NUMA link라는 버스를 이용하여 다른 노드의 메모리에 접근할 수 있다. 이 때, 각 노드를 기준으로 자신의 메모리를 지역 메모리 (local memory), 다른 노드의 메모리를 원격 메모리 (remote memory)라고 한다. 여기서 각 NUMA 노드들이 원격 메모리에 접근하는 시간은 지역 메모리에 접근하는 시간보다 느리다 [1].

최근 데이터베이스 관계 연산의 성능을 향상시키기 위해 NUMA 구조를 활용한 연구들이 활발히 진행되고 있다 [1, 6-9]. 본 논문에서는 NUMA 구조 기반의 효율적인 해시 (hash) 조인 기법을 제안한다.

기존의 조인 연구들에서는 원격 메모리 접근 (remote memory access)의 횟수를 감소시키기 위해, 공통적으로



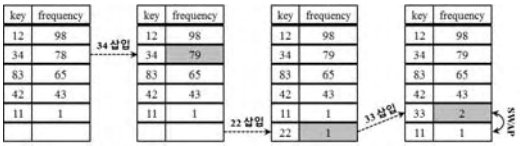
(그림 1) NUMA 구조 예시

수행하는 방안은 데이터 분할 (data partition) 방안이다. 이 때, 기존 기법들은 조인 속성 값의 영역 (domain)을 복수 개의 파티션 (partition)들로 분할하여 각 NUMA 노드가 각 파티션에 속한 튜플들에 대한 조인 연산을 수행한다 [1,2]. 이 때, 각 NUMA 노드는 초기에 자신이 지닌 튜플들이 편중된 조인 속성 값을 갖더라도 자신이 담당한 파티션에 속하지 않는다면, 해당 튜플들을 다른 NUMA 노드들에게 전달한다. 이는 과도한 원격 메모리 접근을 발생시킨다. 또한, 하나의 NUMA 노드에서 복수 개의 스레드 (thread)들이 수행될 수 있으므로 스레드들 간의 부하를 균등하게 배분하기 위해, 스레드 이동 (thread migration) [2] 또는 작업 분배 (work stealing) [8]을 이용한 기법들도 제안되었다.

기존 조인 기법에 반하여, 본 논문에서 제안하는 해시 조인 기법은 각 NUMA 노드들 간의 부하 균형 및 원격 메모리 접근을 최소화하기 위하여 각 튜플의 조인 속성 값이 편중되는지를 판단하고 이에 따라 효율적인 조인 전략을 적용한다.

조인 속성 값의 편중 정도를 판별하기 위해, 본 논문에서는 근사 히스토그램 [5]을 사용한다. 그리고 NUMA 노드들 간의 부하를 균등하게 만들기 위해, 각 조인 속성 값

† 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원의 지원을 받아 수행된 연구임 (R0113-15-005, 대규모 트랜잭션 처리와 실시간 복합 분석을 통합한 일체형 데이터 엔지니어링 기술 개발)



(그림 2) 근사 히스토그램 예시 (k = 6)

의 편중된 정도에 따라 해당 속성 값을 지닌 튜플을 지역 메모리에 유지할지 또는 다른 NUMA 노드로 전달할지를 결정한다.

2. 근사 히스토그램

근사 히스토그램 [5]은 k개의 데이터와 각 데이터의 빈도수 (frequency)를 배열로 유지한다. 여기서 k는 사용자가 정의한 0보다 큰 정수 값이고, 데이터는 빈도수를 기준으로 오름차순 정렬된다.

근사 히스토그램에서 새로운 데이터를 근사 히스토그램에 추가한다고 가정했을 때, 새로운 데이터가 이미 근사 히스토그램에 존재한다면, 해당 데이터의 빈도수를 증가시킨다. 이에 반하여 해당 데이터가 근사 히스토그램에 존재하지 않는다면 가장 낮은 빈도수를 갖는 기존 데이터가 새로운 데이터로 교체되고 빈도수를 1 증가시킨다. 여기서 특정 데이터가 빈번히 발생된다면 해당 데이터는 근사 히스토그램의 상위에 위치하므로 새로운 데이터에 대해서 교체되지 않는다. 또한 빈번히 발생되지 않는 데이터는 근사 히스토그램의 맨 마지막 자리에서 지속적으로 교체될 것이다. 따라서 이러한 방식으로 빈번히 발생하는 데이터를 근사적으로 판별할 수 있다.

그림 2는 k = 6일 때의 근사 히스토그램에 대한 예시이다. 데이터 34는 이미 존재하므로 빈도수만 증가시키고, 데이터 22는 존재하지 않으므로 빈자리에 빈도수 1과 함께 삽입한다. 그리고 데이터 33은 존재하지 않고 빈자리도 없으므로 가장 마지막 자리의 데이터인 22와 교체하고 빈도수를 증가시킨다. 여기서 데이터 33의 빈도수는 2이기 때문에, 데이터 11과 자리를 교환한다.

3. 근사 히스토그램 기반의 해시 조인 기법

본 장에서는 NUMA 구조 기반의 단순 해시 조인 기법과 제안 기법인 근사 히스토그램 기반의 해시 조인 기법에 대해 기술한다. 본 논문에서는 N개의 NUMA 노드들로 구성된 NUMA 구조를 고려하고, 각 NUMA 노드 n_i ($0 \le i < N$)는 하나의 스레드를 수행시킨다고 가정한다. 여기서 i는 n_i의 식별자이다. 그리고 두 노드 n_i와 n_j가 주어졌을 때, n_i의 스레드가 n_j의 지역 메모리에 접근하는 것을 n_i가 n_j로 접근 (access)한다고 표현한다.

3.1 단순 해시 조인 기법

전통적인 해시 조인 기법은 두 릴레이션 R과 S (|R|<|S|)에 대해 Build와 Probe 단계로 구성된다. Build 단계에서는 릴레이션 R의 조인 속성 값을 이용하여 해시 테이블 HT을 생성한다. Probe 단계에서는 릴레이션 S의 조인 속성 값을 이용하여 해시 테이블 HT를 검색하고, 동일한 조인 속성 값을 가지는 릴레이션 R의 튜플을 파악하여 조인 결과로 생성한다.

NUMA 구조 기반의 단순 해시 조인 기법은 데이터 분할 (partitioning) 단계, 지역 해시 테이블 생성 단계, 그리고 지역 조인 단계로 구성된다. 초기에 각 NUMA 노드 n_i는 R와 S의 청크 (chunk)인 $R_i \subseteq R$과 $S_i \subseteq S$를 지역 메모리에 유지한다.

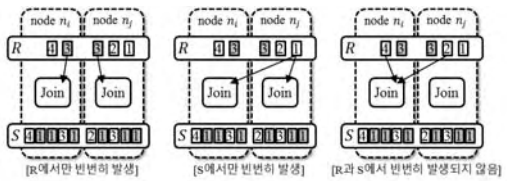
릴레이션 R과 릴레이션 S의 조인 속성이 각각 a와 b라 하자. 이때, 데이터 분할 단계에서 각 NUMA 노드 n_i는 조인 속성 a와 b에 따라서 자신이 담당하는 청크 R_i와 S_i를 분할한다. 즉, R_i에 속한 각 튜플 r은 자신의 조인 속성 r.a의 값에 따라서 대응되는 파티션에 속하게 되며, 청크 S_i에 속한 각 튜플 s 또한 조인 속성 s.b의 값에 대응되는 파티션에 속하게 된다. 데이터 분할을 수행하기 위하여 해시 함수 $H()$를 이용하며 해시 함수의 결과 값 h의 범위는 [0,N)이다. 따라서 조인 속성의 해시 값 h에 따라서 청크 R_i와 S_i는 동일한 해시 값을 지니는 여러 개의 파티션들로 쪼개지며, 각 파티션은 해시 값 h에 따라서 NUMA 노드 n_h ($0 \le h < N$)에 전달된다. 따라서, 각 NUMA 노드 n_h는 동일한 해시 값 h을 지니는 릴레이션 R의 튜플들에 대한 집합 R_h과 릴레이션 S의 튜플들에 대한 집합 S_h를 가진다.

지역 해시 테이블 생성 단계에서, 각 NUMA 노드 n_h는 또 다른 해시 함수 $H'()$을 이용하여 자신이 담당하는 부분 릴레이션 R_h에 대한 지역 해시 테이블 HT_h를 생성한다. 이 후, 지역 조인 단계에서, 각 NUMA 노드 n_h는 S_h의 조인 속성 값을 이용하여 HT_h를 검색하고, 동일한 조인 속성 값을 지니는 R_h의 각 튜플을 파악하여 조인 결과로 생성한다.

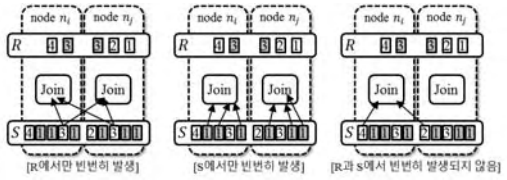
단순 해시 조인 기법에서 릴레이션 R 또는 S의 조인 속성이 편중된 분포를 갖는다면, 특정 해시 값 h를 갖는 튜플들의 수가 증가하여 특정 파티션의 크기가 커지게 된다. 이에 따라서, 각 NUMA 노드가 자신이 담당하는 청크를 파티션들로 분할할 때, 편중된 분포에 대응되는 파티션은 그 크기가 다른 파티션에 비해서 크기 때문에 원격 메모리 접근 횟수가 증가된다. 즉, 한 NUMA 노드가 다른 NUMA 노드에 파티션을 전달하는 비용이 커지게 된다. 또한 편중된 분포에 대응되는 파티션을 담당하는 NUMA 노드에서 지역 해시 테이블 생성 단계 또는 지역 조인 단계의 처리 비용이 다른 NUMA 노드의 비용보다 커지게 되어 전체적인 조인 성능의 저하를 유발한다.

3.2 제안 기법

기존 단순 해시 조인 기법에서는 조인 속성 값의 빈도수와 상관없이 데이터 분할 단계를 수행하여 튜플들을 각 NUMA 노드에 전달한다. 그러나 본 논문에서 제안하는 NUMA 구조 기반의 해시 조인 기법에서는 원격 메모리



a. $r_x \in R_i$에 대한 수행



b. $s_x \in S_i$에 대한 수행

(그림 3) 제안 기법의 수행 예시 (N=2)

접근 횟수를 줄이기 위하여 릴레이션 R과 S의 속한 튜플들의 조인 속성 값의 발생 빈도에 따라서 선별적으로 튜플들을 다른 NUMA 노드들에게 전달한다.

어떤 조인 속성 값 x를 가지는 청크 R_i ($\subseteq$R)의 튜플을 r_x라 하고 청크 S_i ($\subseteq$S)의 튜플을 s_x라 하자 (즉, $r_x.a = s_x.b = x$). 값 x가 릴레이션 R에서는 빈번히 발생하고 릴레이션 S에서는 빈번히 발생하지 않는다면, 각 NUMA 노드 n_i는 r_x를 다른 NUMA 노드에게 전달하지 않고 자신의 지역 메모리에 적재한다. 그리고 각 NUMA 노드 n_j는 s_x를 전체 NUMA 노드들에게 전파 (broadcasting)한다. 이에 따라서, 지역 메모리에 r_x를 적재한 각 NUMA 노드 n_i는 n_j로부터 전송받은 s_x를 이용하여 조인을 수행할 수 있다. 유사하게, 값 x가 릴레이션 R에서는 빈번히 발생하지 않고 릴레이션 S에서는 빈번히 발생한다면, 각 NUMA 노드 n_i는 r_x를 전체 NUMA 노드들에게 전파한다. 그리고 각 NUMA 노드 n_j는 s_x를 다른 NUMA 노드에게 전달하지 않고 자신의 지역 메모리에 적재한다. 이에 따라서, 지역 메모리에 s_x를 적재한 각 NUMA 노드 n_j는 n_i로부터 전송받은 r_x를 이용하여 조인을 수행할 수 있다. 이를 통하여 한 쪽 릴레이션에서만 빈번히 발생되는 값 x를 가지는 튜플들의 전송에 따른 원격 메모리 접근을 줄인다.

이에 반하여, 값 x가 릴레이션 R과 S에서 모두 빈번히 발생하지 않거나 또는 릴레이션 R과 S에서 모두 빈번히 발생한다면, 기존 단순 해시 조인 기법과 동일하게 각 NUMA 노드 n_i는 r_x와 s_x를 해시 함수 $H()$를 이용하여 조인 속성 값 x를 담당하는 NUMA 노드 n_h에 전달하여 조인을 수행한다.

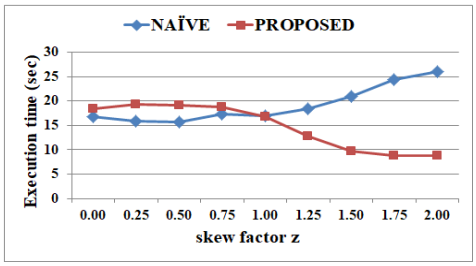
그림 3은 본 논문에서 제안하는 해시 조인 기법의 수행 방법에 대한 예시이다. 그림 3에 나타난 NUMA 구조는 2개의 NUMA 노드들 n_i와 n_j로 구성되어 있으며, 각 튜플의 조인 속성 값이 짝수이면 n_i로, 홀수이면 n_j로 각각 전달된다. 그리고 조인 속성 값으로 3을 갖는 튜플들은 릴레이션 R에서, 조인 속성 값으로 1을 갖는 튜플들은 릴레이션 S에서 각각 빈번히 발생된다고 가정한다. 그림 3에서 조인 속성 값 3을 갖는 튜플들은 릴레이션 R에서만 빈번히 발생되므로, 각 청크 R_i에서 조인 속성 값 3을 갖는 튜플들은 각 NUMA 노드 n_i의 지역 메모리에 적재되고, 각 청크 S_i에서 조인 속성 값 3을 갖는 튜플들은 전체 NUMA 노드들 n_i와 n_j에게 전파된다. 이와 다르게, 조인 속성 값 1을 갖는 튜플들은 릴레이션 S에서만 빈번히 발생되므로, 각 청크 S_i에서 조인 속성 값 1을 갖는 튜플들은 지역 메모리에 적재되고, 각 청크 R_i에서 조인 속성 1을 갖는 튜플들은 전체 NUMA 노드 n_i와 n_j에게 전파된다. 그리고 조인 속성 값 2와 4를 갖는 튜플들은 R과 S에서 모두 빈번히 발생되지 않으므로, 전체 NUMA 노드 n_i와 n_j에게 전파된다.

본 논문에서 제안하는 해시 조인 기법은 다음과 같은 3단계-근사 히스토그램 생성, 지역 해시 테이블 생성, 지역 조인-로 구성되어 진다.

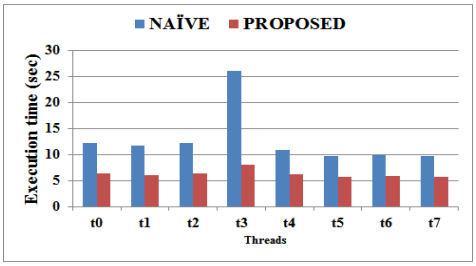
근사 히스토그램 생성 단계: 이 단계에서, 각 NUMA 노드 n_i에서 청크 R_i과 S_i에 속한 튜플들의 조인 속성 값들에 대한 분포를 파악한다. 이를 위하여, 각 NUMA 노드 n_i는 청크 R_i와 S_i에 대해 샘플링을 수행하고, 샘플링된 튜플들의 조인 속성 값들에 대해 R_i와 S_i에 대한 지역 (local) 근사 히스토그램들 hist(R_i)와 hist(S_i)를 각각 생성한다. 이 때, 사용자 정의 값 λ 이상의 빈도수를 지니는 조인 속성 값들을 *편중된 조인 속성 값*들이라 하고 sk(R_i), sk(S_i)로 표기한다. 이 후, 모든 NUMA 노드들이 편중된 조인 속성 값들의 정보를 공유하기 위하여, 각 NUMA 노드 n_i에서 파악된 sk(R_i)와 sk(S_i)를 하나의 NUMA 노드 n_k에서 수집한다. n_k에서는 전체 NUMA 노드들로부터 전송받은 편중된 조인 속성 값들을 병합하여 각 릴레이션 R과 S에서 편중된 조인 속성 값들을 sk(R) ($=\cup_{0\le i<N}$ sk(R_i))과 sk(S) ($=\cup_{0\le i<N}$ sk(S_i))를 생성한다. 그리고 n_k는 sk(R)과 sk(S)를 전체 NUMA 노드들에게 전파한다. 이 후, 각 NUMA 노드 n_i는 sk(R)과 sk(S)를 이용하여 특정 조인 속성 값 x가 편중된 조인 속성 값인지 또는 아닌지를 판별할 수 있다.

지역 해시 테이블 생성 단계: 이 단계에서는 각 NUMA 노드 n_i는 sk(R)과 sk(S)를 참조하여 각 튜플 r$\in R_i$의 조인 속성 값 r.a가 릴레이션 R과 S에서 편중되는지 판별한다. 만약, r.a가 R에서는 편중되고 S에서는 편중되지 않는다면, r은 지역 해시 테이블 HT_i에 추가된다. 이와 반대로, r.a가 R에서는 편중되지 않고 S에서는 편중된다면, r은 전체 NUMA 노드들에게 전파된다. 그리고 r.a가 R과 S에서 모두 편중되지 않거나 R과 S에서 모두 편중된다면, 각 NUMA 노드 n_i는 해시 함수 $H()$를 이용하여 r을 NUMA 노드 n_h에 전달한다. 이 후, 각 NUMA 노드 n_h는 다른 NUMA 노드들로부터 전달받은 R의 튜플들을 지역 해시 테이블 HT_h에 추가한다.

지역 조인 단계: 이 단계에서는 지역 해시 테이블 생성 단계와 유사하게, 각 NUMA 노드 n_i가 sk(R)과 sk(S)를 참조하여 각 튜플 s$\in S_i$의 조인 속성 값 s.b가 릴레이션 R과 S에서 편중되는지 판별한다. 만약, s.b가 R에서는 편중되고 S에서는 편중되지 않는다면, s는 다른 NUMA 노



(그림 4) z의 증가에 따른 조인 수행 시간



(그림 5) 각 스레드들의 조인 수행 시간

드들에게 전파된다. 이와 반대로, s.b가 R에서는 편중되지 않고 S에서는 편중된다면, 각 NUMA 노드 n_i는 지역 해시 테이블 HT_i을 검색하고 s.b와 동일한 조인 속성 값을 지니는 R의 튜플들을 파악하여 조인한다. 그리고 s.b가 R과 S에서 모두 편중되지 않거나 R과 S에서 모두 편중된다면, 각 NUMA 노드 n_i는 해시 함수 $H()$를 이용하여 s를 NUMA 노드 n_h에 전달한다. 이 후, 각 NUMA 노드 n_h는 다른 NUMA 노드들로부터 전달받은 S의 각 튜플 s에 대해 HT_h를 검색하고 조인 속성 값 s.b와 동일한 조인 속성 값을 지니는 R의 튜플들을 파악하여 조인한다.

4. 실험 및 평가

본 논문에서 제안 기법의 성능 평가를 위해 2개의 NUMA 노드들로 구성된 컴퓨터를 사용하였으며, 각 NUMA 노드의 CPU는 Intel Xeon E5-2609 v3이고 지역 메모리의 크기는 32 Gbyte이다. 그리고 각 NUMA 노드는 4개의 스레드들을 수행시켜 조인을 수행한다. 제안 기법은 ubuntu 16.04 환경에서 C++로 구현하고 gcc version 5.1.0 컴파일러를 이용하여 컴파일 하였다.

본 논문의 실험에서는 합성 데이터를 이용하였으며 릴레이션 R의 튜플 수는 $1.0*10^5$개이고 조인 속성 값의 분포는 균등 분포이다. 릴레이션 S의 튜플 수는 $1.0*10^9$개이고 조인 속성 값은 zipfian 분포를 따르며 skew factor 값 z는 0에서 2.0까지의 값을 가진다. 지역 근사 히스토그램 생성하기 위한 샘플 비율은 각 릴레이션 튜플 수의 1%, 근사 히스토그램의 크기 k는 128, 그리고 빈번한 조인 속성 값을 판별하기 위한 사용자 정의 임계값 λ는 각 릴레이션에 속한 튜플 수의 0.01%이다.

그림 4는 skew factor 값의 변화에 따른 제안 기법(PROPOSED)과 단순 해시 조인 기법 (NAIVE)의 해시 조인 수행 시간을 나타낸다. 그림 4에서 skew factor 값이 1.0 미만일 때, 제안 기법은 근사 히스토그램 생성의 부하로 인하여 단순 해시 조인 기법보다 다소 느린 수행 시간을 보이나 skew factor가 증가함에 따라, 제안 기법이 단순 해시 조인 기법보다 우수한 성능을 보인다.

릴레이션 R에 속한 튜플들은 skew factor 값과 상관없이 균등 분포를 따르나 skew factor가 증가함에 따라 릴레이션 S에 속한 튜플들의 조인 속성 값이 편중되게 된다. 따라서 단순 해시 조인 기법에서는 릴레이션 S에 속한 편중된 조인 속성 값을 지닌 튜플들을 한 NUMA 노드의 지역 메모리로 전송하기 위한 원격 메모리 접근이 빈번히 발생되어 skew factor 값이 증가함에 따라서 전체적인 조인 성능이 저하된다. 이에 반하여 제안 기법에서는 릴레이션 S에 속한 편중된 조인 속성 값을 지닌 튜플들을 각 NUMA 노드의 지역 메모리에 유지하고 대신 빈번히 발생되지 않는 릴레이션 R에 속한 튜플들을 각 NUMA 노드에 전파하여 원격 메모리 접근을 줄임으로서 skew factor 값이 증가함에 따라서 조인 시간이 감소하게 된다. 또한 각 스레드 간의 작업량을 균등하게 유지하는 효과도 얻을 수 있다. 그림 5는 skew factor의 값이 2.0일 때, 각 기법들의 스레드 별 수행시간을 나타낸다. 그림 5에서 보는 바와 같이, 단순 해시 조인 기법에서 스레드 t3의 수행 시간이 다른 스레드들보다 월등히 큰 것을 볼 수 있다. 따라서 전체적인 조인 수행 성능이 저하된다. 이에 반하여 제안 기법은 스레드들에 대한 수행 시간이 균등하여 전체적인 조인 성능이 향상된다.

5. 결론

본 논문의 제안 기법에서는 NUMA 구조를 고려한 효율적인 해시 조인 기법을 제안하였다. 본 논문의 실험에서 조인 속성 값에 대한 분포가 편중될수록 제안 기법의 원격 메모리 접근 횟수는 감소하므로, 단순 해시 조인 기법의 비하여 제안 기법의 성능이 뛰어남을 보였다.

[참 고 문 헌]

[1] Martina-Cezara Albutiu et. al., “Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems”, Proceedings of VLDB Endowment, Vol. 5, No. 10, pp. 1064-1075, 2012

[2] Yinan Li et. al., “NUMA-aware algorithms: the case of data shuffling”, Conference on Innovative Data Systems Research (CIDR), 2013

[3] Li Wang et. al., “NUMA-Aware Scalable and Efficient In-Memory Aggregation on Large Domains”, IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol. 27, No. 4, pp. 1071-1084, 2015

[4] Yuichiro Yasui et. al., “Fast and Scalable NUMA-based Thread Parallel Breadth-first Search”, International Conference on High Performance Computing & Simulation (HPCS), pp. 377-385, 2015

[5] Graham Cormode et. al., “Methods for finding frequent items in data streams”, Vol. 19, No. 1, pp. 3-20, 2010

[6] Orestis Polychroniou et. al., “A Comprehensive Study of Main-Memory Partitioning and its Application to Large-Scale Comparison- and Radix-Sort”, Proceedings of ACM SIGMOD, pp. 755-766, 2014

[7] Paratanu Roy et. al., “Low-Latency Handshake Join”, Proceedings of VLDB Endowment, Vol. 7, No. 9, pp. 709-720, 2014

[8] Viktor Leis et. al., “Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age”, Proceedings of ACM SIGMOD, pp. 743-754

[9] Cagri Balkesen et. al., “Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited”, Proceedings of VLDB Endowment, Vol. 7, No. 1, 2013