정렬을 통한 해시 조인 탐색 단계에서의 캐시미스 감소 기법[1]

오기환*, 김재명*, 강운학*, 이상원*
*성균관대학교 컴퓨터공학과
e-mail : wurikiji@skku.edu

Reducing Cache Misses in Hash Join Probing Phase By Pre-sorting Strategy

Gi-Hwan Oh*, Jae-Myung Kim*, Woon-Hak Kang*, Sang-Won Lee*
*Dept. of Computer Engineering, Sungkyunkwan University

요 약

메모리 가격이 저렴해 짐에 따라 대용량의 데이터베이스 연산이 메모리 안에서 처리될 수 있다. 그에 반해 메모리의 접근속도는 과거에 비해 크게 향상되지 않았기 때문에, 효율적인 캐시 활용이 전체 성능을 결정하는 중요한 요소가 된다. 멀티코어 환경에서 효율적 캐시와 높은 동시성을 모두 만족시키기는 쉽지 않다. 이 논문에서는 알려진 메모리 기반 해시 알고리즘을 비교하고, 각각에 대해 탐색 단계에서 조인 키를 기준으로 정렬 알고리즘을 적용하여 수행 시간과 캐시 미스 감소를 비교한다.

1. 서론

메인 메모리의 용량(capacity)은 점점 커지고 있다. 과거에는 메모리가 제한된 자원이었지만, 단위 가격의 하락은 테라-바이트(Terabytes, 10^{12} bytes)급의 메모리를 가능하게 하였다. 메모리 용량은 증가하였지만, 프로세서의 처리 속도에 비례한 성능은 지속적으로 감소하였다. 예를 들어 1994 년 메모리 접근 속도는 15~25 CPU 사이클이 소요되었지만 [3], 이 논문에서 사용된 Intel 2.8GHz CPU 에서 DDR3-1333 메모리를 접근하는 경우, 100~110 CPU 사이클이 소요된다. 이는 프로세서의 속도를 기준으로 메모리 접근 속도가 4 배~7 배 감소하였음을 의미한다.

단일 CPU 처리 속도 역시 한계에 이르러 프로세서 제조사들은 멀티코어(multi-core) CPU 로 그 한계를 극복하고 있다. 응용프로그램이 멀티코어 CPU 를 최대한 활용하기 위해서는 병렬수행이 되도록 코드가 수정되어야 한다. 서로 다른 코어가 공유되는 자원에 대해 경합을 벌이는 경우 실질적 성능 향상이 없기 때문에, 효과적인 병렬화를 위해서는 프로세서 아키텍처에 대한 이해가 필요하다. 모든 개발자들이 효율적인 프로그램을 개발하는 것을 보장하기 어렵다. 반면, 데이터베이스 시스템은 복수의 쿼리를 동시에 처리하기도 하고, 하나의 쿼리를 병렬로 처리하는 기능을 제공한다. SQL 과 같은 선언적 질의언어(declarative query language)를 이용한 개발 편의와 더불어, 멀티 코어 환경에서 병렬성 (degree of parallelism)을 극대화 하기 위한 도구로 활용될 수 있다.

데이터 베이스 시스템에서 처리해야 하는 데이터의 양은 폭발적으로 증가 (information explosion) 하고 있으며, 메모리 환경에서 실시간 데이터 분석에 대한 요구사항 역시 급증하고 있다. 이러한 요구사항을 만족하기 위해서는 메모리 상에서 대용량 데이터 베이스 연산을 수행하여야 한다.

해시 조인은 대표적인 조인 연산 중 하나인데, 조인 키에 대해 값 비교 및 상수 횟수의 메모리 참조만을 통해 조인을 병렬로 수행한다. 하지만, 해시 테이블의 크기가 증가하면, 생성된 해시 테이블을 참조하며 조인을 수행하는 단계(probe phase)에서 캐시 미스로 인한 성능 저하가 심각하다. 이는 다중 쓰레드의 동시 참조로 더욱 악화된다.

이러한 문제는 이미 디스크 기반의 외부 해시 조인(external hash join)의 임의의(random) 디스크 IO 유발상황과 유사하다. 이에 대한 해결책 역시 GRACE 해시 조인[5]에서 사용한 사고의 흐름과 유사한데, 과거에 디스크 IO 를 최소화 하기 위한 파티션 기법이 캐시 미스를 최소화 하기 위한 것으로 발전하였다. [4]

하지만 [2]에서는 위의 결론과 다르게 파티션을 고려하지 않은 기본적인(naïve) 해시 조인(이하 NO PARTITION)이 전반적으로 더 좋은 성능을 보인다고 주장하였다. 게다가 [4]의 경우 파티션 정도에 따라 성능 기복이 있지만, NO PARTITION 의 경우 더 균일한 성능과 구현 편의성이 있다고 주장하였다.

[1] 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었음(NIPA-2012-(H0301-12-3001)).

이 논문에서는 [2]의 접근을 다시 연구하였다. 실험을 통해 논문의 주장대로 PARTITION 에 비한 장점을 확인하였지만, 탐색(probe) 단계가 수행시간의 대부분을 차지하고 그 원인이 해시 테이블 참조 시 캐시 미스임을 확인하였다.

캐시 미스가 발생하는 주요한 요인은 다음과 같다. 1) 해시 테이블의 크기가 캐시 크기에 비해 충분히 (10 배 이상) 크다. 2) Probe 측 테이블의 조인 키는 임의의 값을 가지기 때문에 해시 테이블에 랜덤 읽기를 유발한다. (캐시 미스) 이를 해결하기 위해 우리는 탐색 테이블 측의 조인 키에 대해 부분적인 정렬을 수행하였다.

이 논문의 공헌은 다음과 같다. 1) NO PARTITION 의 캐시 미스 유발 현상에 대해 분석 및 검증하였고, 2) 이를 완화하기 위한 정렬 알고리즘을 제안하였다.

제안한 방식으로 성능이 최대 31% 향상되었다. 정렬에 필요한 연산 비용을 고려하면 캐시미스에 따른 성능 저하가 심각성을 알 수 있다. 또한 알고리즘에 따른 캐시 미스를 분석한 결과 캐시 미스가 20% 수준으로 감소함을 확인하였다.

논문은 다음과 같이 구성되었다. 2 장에서는 해시 조인에 대한 관련연구를, 3 장에서는 해시 조인에서 캐시 미스를 줄이기 위한 사전 정렬 기법에 대해 소개한다. 4 장에는 기존의 알고리즘 별 사전정렬 기법의 효과를 측정하고, 5 장에서 이 논문을 마무리 한다.

2. 해시조인

이 장에서는 이 논문에서 비교하는 기존 해시 조인 알고리즘에 대해 설명한다.

해시 조인 알고리즘은 두 개의 테이블의 관계를 형성하는데 사용한다. 조인할 두 개의 테이블 중 크기가 상대적으로 더 작은 테이블을 기준 테이블(build table)로 선택하여 해시 테이블을 생성하고, 다른 테이블 전체를 탐색 테이블(probe table)로 선택하여 테이블 전체를 탐색하면서 키 값의 일치 여부를 확인 한다.

해시 조인은 크게 파티션 단계(partition phase), 해시 테이블 생성 단계(build phase)와 탐색 단계(probe phase)로 나뉘며, 파티션 단계는 해시 조인 알고리즘에 따른 부가적인 단계이다. 파티션 단계는 일반적으로 기준 테이블이 메모리 내에 모두 저장될 수 있는 경우에 수행한다. 각각의 파티션이 CPU 캐시 사이즈와 동일한 크기를 가질 경우 해시 테이블 생성 단계와 탐색 단계에서의 캐시 미스가 현저히 줄어들게 된다. 해시 테이블 생성 단계에서는 먼저 빈 해시 테이블을 생성한 뒤 기준 테이블의 모든 튜플을 읽어 온 뒤, 조인에 사용할 키 값을 추출하여 해시를 적용한다. 해당 해시 키 값에 해당하는 해시 버킷의 끝에 튜플을 추가한다. 탐색 단계에서는 탐색 테이블의 튜플을 읽어와 조인에 사용할 키 값을 추출하여 해시를 적용한 뒤, 해당 해시 버킷에 탐색 대상 튜플의 키 값이 존재하는지의 여부를 판별한 뒤 존재할 경우 조인 결과를 출력한다.

파티션 단계에서 수행하는 알고리즘에 따라 파티션을 하지 않는 해시 조인(No Partitioning), 파티션을 만드는 해시 조인(Independent Partitioning) 등으로 나눌 수 있다. 이 논문에서는 [2]에서 소개된 해시 조인 알고리즘 중 두 가지를 사용하였으며, 각각의 해시 조인 알고리즘의 특징은 다음과 같다.

2.1 No Partitioning Hash Join Algorithm

파티션을 만들지 않는 기본적인 해시 조인 알고리즘이다. 해시 테이블 생성 단계에서 모든 쓰레드가 공유하는 하나의 공유 해시 테이블을 생성한다. 각 쓰레드간의 독립성 보장을 위해 래치가 필요하다.

2.2 Independent Partitioning Hash Join Algorithm

모든 쓰레드가 기준 테이블과 탐색 테이블 모두에 대해 파티션을 만들어서 사용한다. 각 쓰레드마다 고유의 파티션 영역을 가지고 해시 조인 알고리즘을 수행한다.

3. 캐시 미스 감소를 위한 사전 정렬 기법

일반적으로 탐색 테이블내의 튜플의 키 값이 정렬되어 있지 않기 때문에, 연속된 튜플들이 서로 다른 해시 키 값을 갖게 되고 이로 인해 서로 다른 해시 버킷을 참조하게 된다. 하지만, 대부분의 경우 기준 테이블의 크기가 캐시 사이즈를 초과하기 때문에 탐색 단계에서 해시 버킷을 참조할 때 많은 캐시 미스를 유발하게 된다. 이러한 탐색 단계에서 발생하는 캐시미스는 대부분의 해시 조인 알고리즘에서 피할 수 없는 부분이다.

이러한 상황에 기초하여, 본 논문에서는 해시가 적용된 키 값을 탐색 단계 이전에 미리 정렬을 하여 지역성을 높여 탐색 단계에서의 캐시 미스를 줄이는 알고리즘을 제안한다. 이 알고리즘에서는 정렬을 통해 얻어지는 성능 향상이 정렬하는데 사용되는 비용에 비해 얼마나 큰가에 따라 성능이 달라지게 된다. 정렬 기법의 최적화를 한번에 모든 데이터를 정렬하지 않고 캐시 크기만큼의 데이터만 정렬하여 탐색 단계를 적용 함으로써 정렬하는 동안 생길 수 있는 캐시 미스를 방지하였다.

4. 성능 평가

이 장에서는 각 해시 알고리즘 별 성능을 측정하고, 탐색 단계에서 캐시 미스를 분석한다.

<표 1> 실험 환경

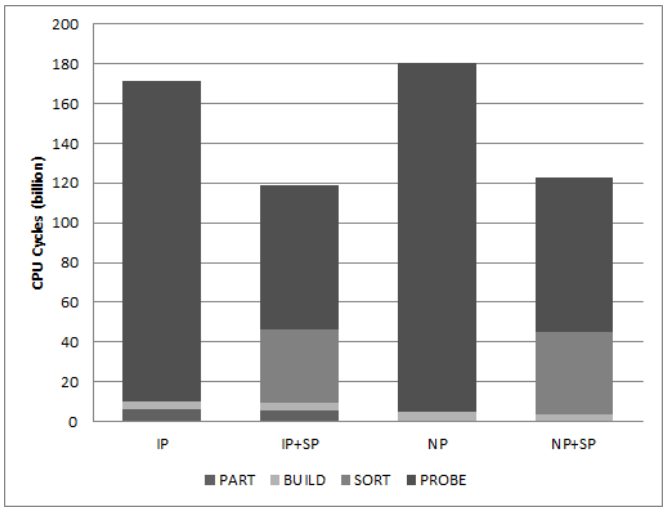
CPU	Intel Core i7-860 (4 x 2.8GHz)
Private Cache	256KB
Shared Cache	8MB
Main Memory	12GB (3 x 4GB)
OS	Linux 2.6.32
Compiler	GNU g++ 4.4.6

실험 환경은 <표 1>과 같다. 멀티코어 환경을 위해 Intel Quad-Core 2.8GHz 프로세서를 사용하였다. 프로세서에는 각 코어 별로 256KB 의 캐시를 가지며 4 개의 코어는 8MB 의 캐시를 공유한다. 메인 메모리는

12GB 이며, Linux Kernel 2.6, 컴파일러는 GNU g++ 4.4.6 버전을 활용하였다.

실험을 위해 [2]에 소개된 두 가지 해시 조인 알고리즘인 No Partitioning (NP) 알고리즘과 Independent Partitioning (IP) 알고리즘을 [2]에서 제공하는 C++ 프로그램을 활용하였다. 또한 각각의 알고리즘의 탐색 단계(probe phase) 이전에 정렬을 적용한 알고리즘을 구현하였다. 정렬을 적용한 알고리즘은 각각 NP+SP, IP+SP 로 표현한다. 프로그램의 수행 순서는 모든 데이터를 메인 메모리로 읽어 온 뒤 해시 조인을 하는 순서이다. 실험에서는 메인 메모리로 데이터를 읽어오는 것은 결과에 포함하지 않았다. 실험 장비가 4 개의 코어를 가지고 있으므로, 위의 알고리즘은 모두 4 개의 쓰레드에서 동시에 수행된다.

실험에 사용한 두 개의 테이블은 각각 16M 개, 256M 개의 튜플을 가지고 있으며 각 튜플의 사이즈는 16B 이다.[2] 해시 테이블을 생성하는 테이블(Build Table)은 모두 서로 다른 정수 키를 가지며, 탐색 측의 테이블 (Probe Table)은 조인 되는 테이블과 1:16 비율로 균일한 분포를 가진다.



(그림 1) 단계별 소모 사이클

(그림 1)은 단계별 소모 사이클. 결과를 확인하면 경우 IP+SP 가 가장 적은 사이클을 소모하는 것을 볼 수 있다. SP 를 적용한 알고리즘의 결과를 살펴보면 정렬에 드는 시간이 전체 수행시간의 30% 이상을 차지 하지만 탐색 단계의 소모 사이클이 SP 가 적용되지 않은 알고리즘에 비해 55% 이상 감소하여 총 소모 사이클이 31% 이상 줄어든 것을 볼 수 있다.

<표 2>탐색 단계의 캐시 미스 (최종 공유 캐시)

알고리즘	캐시 미스
IP	381,788,161
IP + SP	83,773,673
NP	408,160,467
NP + SP	112,712,619

<표 2>는 탐색단계에서 각 알고리즘 별 캐시 미스 유발 정도를 나타낸다. IP 와 NP 모두 SP 를 적용하기 전에는 많은 캐시 미스를 유발하지만 SP 를 적용한 후에는 캐시 미스가 23% 수준으로 떨어지는 것을 확인할 수 있다.

5. 결론

이 논문에서는 탐색 단계에서의 많은 캐시 미스를 알아보고 이를 완화하기 위한 정렬 알고리즘을 제안하고 실험하였다. 실험 결과 정렬 알고리즘을 적용한 해시 조인이 그렇지 않은 해시 조인에 비해 31%이상의 성능 향상을 얻을 수 있음을 알았다. 또한 정렬 알고리즘을 적용한 결과 캐시 미스가 정렬을 하지 않은 것에 비해 23% 수준으로 감소하는 것을 확인하였다.

참고문헌

[1] Kim, Sedlar, and Chhungani, “Sort Vs. Hash Revisited: Fast Join Implementation On Modern Multi-Core CPUs”, In VLDB, August 2009

[2] Blanas, Li, and M. Patel, “Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs”, In ACM SIGMOD, June 2011

[3] Ambuj Shatdal, Chander Kant , and Jeffrey F. Naughton, “Cache Conscious Algorithms for Relational Query Processing”, In Proceedings of the Very Large Database Conference, 1994.

[4] S. Manegold, P. A. Boncz, and M. L. Kersten, “Optimizing main-memory join on modern hardware”, IEEE Trans. Knowl. Data Eng., 14(4):709–730, 2002.

[5] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka, “Application of Hash to Data Base Machine and its Architecture”, New Generation Computing, 1(1):63-74, 1983.