데이터베이스시스템을 위한 Hybrid Hash Sort Join 기법

신현광[1*] 최용성[2] 온병원[3] 이인규[4] 최규상[1]

[1]영남대학교 정보통신공학과
[2]경기대학교 컴퓨터과학과
[3]군산대학교 통계 컴퓨터과학과
[4]차세대융합기술연구원 스마트 그리드연구센터

shg3786@naver.com, soonmewar@naver.com, on.byung.won@gmail.com,
inlee3941@gmail.com, castchoi@ynu.ac.kr

The Hash Sort Scheme for Database Management Systems

Hyun kwang shin[1*] Yong sung Choi[2] Byung-Won On[3] In Gyu Lee[4] Gyu Sang Choi[1]

[1]Yeungnam University Information and Communication Engineering
[2]Kyonggi University Computer Science
[3]Kunsan National University Statistics and Computer Science
[4]Advanced Institute of Convergence Technology

요 약

기존의 Hybrid Hash Join 알고리즘은 2차 해싱 단계에서, 하나의 테이블을 해싱을 통하여 해시 디렉토리를 구성한다. 병합단계에서는 다른 하나의 테이블에서 하나의 레코드를 읽어와 해당 해시 구조의 마지막 버킷까지 비교해야 하므로, 병합에 많은 시간이 소비된다. 본 논문에서 기존의 Hybrid Hash Join 기법보다 효율적인 Hybrid Hash Sort Join 기법을 제안한다. 본 논문에서 제안하는 기법은 하나의 테이블의 해시 디렉토리를 구성할 때에, 레코드의 키 값을 기준으로 정렬하도록 구성한다. 따라서 병합 단계에서 해시 디렉토리에 있는 해당 레코드들을 모두 비교할 필요가 없으므로, 병합 시간을 단축할 수 있다.

1. 서 론

대용량 데이터베이스 처리 시스템에서는 효율적인 데이터 처리를 위해, 조인 알고리즘을 사용한다. 조인 알고리즘은 대용량 데이터를 처리하는 데 가장 중요한 알고리즘이며, 반복적으로 데이터를 메모리에 업데이트하는 특징을 가진다[1,2]. 그중 해시 조인은 레코드들을 기억공간에 저장하기 위한 자료구조로서 데이터베이스 저장이나 검색에 잘 적용되는 저장 기법이다. 기존의 해시 조인 알고리즘 중 Hybrid Hash Join은 현재 널리 쓰이고 있는 해시 조인 알고리즘이다.

본 논문에서는 Hybrid Hash Sort Join 알고리즘을 제안하고 기존의 Hybrid Hash Join 알고리즘과 동일한 환경에서 성능을 비교한다.

구성은 다음과 같다. 2장에서는 기존 조인 알고리즘의 설명과 제안하는 Hybrid Hash Sort Join 대해 논의 하며, 3장은 실험환경과 성능을 비교 분석한다. 마지막으로 4장에서는 결론을 맺는다.

2. 본 론

조인 알고리즘은 두 테이블을 조인하기 위해 사용되며, 두 개의 테이블의 크기가 매우 큰 경우에는, 메인 메모리에 두 테이블의 모든 데이터를 한 번에 올릴 수 없게 된다. 따라서 두 테이블의 데이터를 블록 단위로 나누어 메인 메모리로 올린 후, 조인 연산을 수행한다. 현재까지 다양한 종류의 조인 알고리즘들이 사용되고 있다. 현재 사용되고 있는 조인 알고리즘으로는 Block Nested Loop Join, Sort-Merge Join, Grace Hash Join, Hybrid Hash Join 등이 있다[3]. 본 논문에서는 기존의 조인 알고리즘 중 가장 효율적인 Hybrid Hash Join의 성능을 개선한 Hybrid Hash Sort Join 알고리즘을 제시한다.

본 논문에서는 두 테이블은 다음과 같이 가정한다. R 테이블은 <Student_No, Score>을 가지며, S 테이블은 <Class_No, Student_No>의 스키마를 가진다. S 테이블에서 Class_No는 기본 키이고, Student_No는 외래키이다. R 테이블에서 Student_No는 R 테이블의 기본 키이고, Score는 value라고 할 수 있다. 따라서 조인 연산은 Select R.Student_No, S.Student_No, S.Score from R, S where R.Student_No = S.Student의 쿼리로 수행된다.

2.1. Hybrid Hash Join 알고리즘 [4]

첫 번째 단계에서 R 테이블은 데이터를 블록단위로 나눈다. R 테이블의 첫 번째 블록을 1차 해시 함수를 사용하여 각 레코드

를 해당 버킷에 할당한다[5]. 위 방식으로 R 테이블의 다른 블록들을 1차 해싱을 통해 각 버킷에 레코드를 할당 및 하드디스크에 저장한다. S 테이블의 블록들도 동일한 과정을 거쳐 해싱을 수행하며, 버킷들을 하드디스크에 저장한다.

두 번째 단계에서는 S 테이블의 버킷에서 외래키를 메모리에 읽어와 R 해시 테이블의 기본키와 병합 연산을 수행한다.
단, R 테이블과 S 테이블의 첫 번째 버킷은 하드디스크에 저장하지 않고, 2차 해시 함수를 통해 메모리에서 조인을 수행한다.

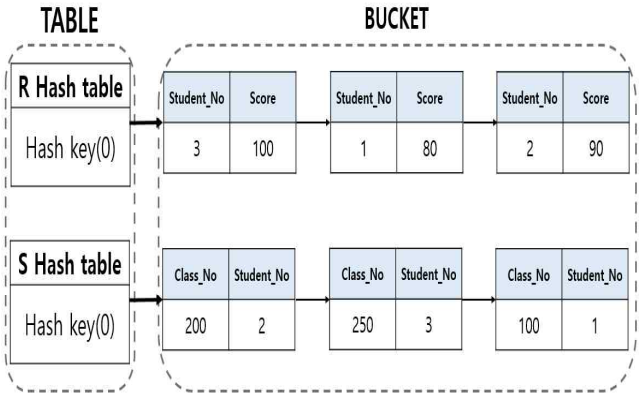

그림 1 Hybrid Hash Join 알고리즘의 예시

그림 1은 Hybrid Hash Join 알고리즘의 예시이다. R 해시 테이블과 S 해시 테이블의 해시 키는 하나의 디렉토리에 여러 개의 버킷으로 구성된다. R 테이블은<Student_No, Score>, S 테이블<Class_No, Student_No>로 구성되며, 랜덤 값을 가진다. 버킷 하나 당 한 개의 레코드로 이루어져 있다. 본 논문은 Hybrid Hash Join과 Hybrid Hash Sort Join과 1차 해싱이 동일함으로 첫 번째 단계는 생략한다.

두 번째 단계에서 2차 해시 함수를 통해 메모리에 있는 S 해시 테이블의 버킷 중 첫 번째, 레코드 Student_No와 R 해시 테이블의 모든 버킷의 Student_No와 순차적으로 비교한다. 비교 과정에서 S 해시 테이블의 첫 번째 레코드 Student_No와 R 해시 테이블의 Student_No 레코드 중 동일한 값을 가지게 되면 병합단계를 수행한다.
그림 1은 S 해시 테이블의 첫 번째 Student_No의 외래키 2와 R 해시 테이블의 세 번째 레코드 Student_No의 기본키 2가 같으므로 병합단계를 수행한다. 병합테이블은 Score, Student_No 그리고 Class_No로 순으로 생성한다. 이 과정을 S 해시 테이블의 모든 레코드 Student_No에 대해 반복 수행한다.

이 알고리즘의 경우 S 해시 테이블의 첫 번째 버킷의 Student_No와 R 해시 테이블의 모든 버킷의 Student_No에 대해 반복적으로 비교하므로, 불필요한 비교연산을 수행하는 단점이 존재한다. 이 논문에서 제안하는 알고리즘은 이러한 단점을 개선하여 불필요한 비교 연산을 최소화시켜 시스템의 성능을 향상한 알고리즘으로 2.2장에서 설명한다.

2.2. Hybrid Hash Sort join 알고리즘

첫 번째 단계는 Hybrid Hash Join 알고리즘과 동일하다. 하지만, 두 번째 단계 2차 해시 함수에서, R 테이블은 삽입 시 Student_No를 기준으로 삽입 정렬을 수행하며, R 테이블의 레코드는 Student_No 기본키를 기준으로 정렬된다. 그러므로 R 해시 테이블의 기본키와 S 해시테이블의 외래키를 비교하는 과정에서 모든 Student_No에 대한 비교 연산을 줄일 수 있다.

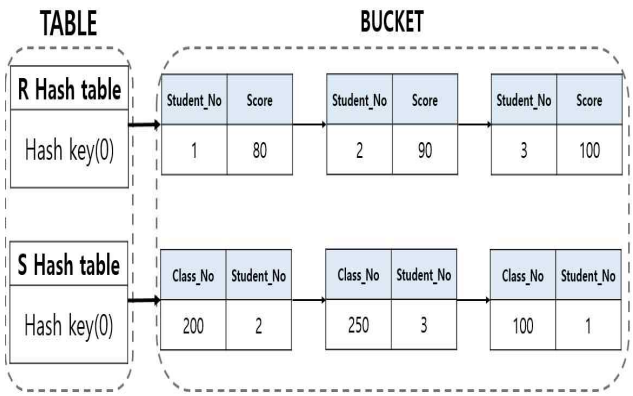

그림 2 Hybrid Hash Sort Join 알고리즘의 예시

그림 2는 Hybrid Hash sort Join 알고리즘의 예시로서 구성은 그림 1의 해시 테이블과 동일하지만 R 해시 테이블의 레코드들은 기본키 Student_No를 기준으로 정렬된 상태이다. S 해시 테이블의 첫 번째 레코드 Student_No와 R 해시 테이블의 모든 레코드의 Student_No들 비교하는 과정에서 R 해시 테이블의 첫 번째 레코드의 Student_No와 동일하지 않으므로 다음 레코드의 Student_No와 비교한다. R 해시 테이블의 두 번째 레코드의 Student_No는 S 해시테이블의 첫 번째 레코드 Student_No와 동일하므로 병합테이블에 쓴 뒤, 다음 R 해시테이블의 레코드 Student_No를 비교해 Student_No가 동일하지 않을 경우 마지막 레코드까지 연산을 수행하지 않고, S 해시테이블의 두 번째 레코드의 Student_No를 앞서 말한 방식과 동일하게 수행한다. Hybrid Hash sort Join 알고리즘 경우, 동일한 Student_No가 있는 레코드를 찾은 후, 나머지 레코드에 대해 모두 비교할 필요성이 없다.

3. 실 험 결 과

본 논문의 실험은 1차 해싱을 생략한 2차 해싱 단계에서의 성능을 비교한다. 왜냐하면, 1차 해싱은 기존의 Hybrid Hash Join과 Hybrid Hash Sort Join과 동일하기 때문이다. 또한, 다중 디렉토리를 사용하지 않았으며, 단일 디렉토리와 Disk I/O를 제외한 메모리 공간에서 실험했다.

두 테이블은 다음과 같이 가정한다. R 테이블은 <Student_No, Score>을 가지며, S 테이블은 <Class_No, Student_No>의 스키마를 가진다. S 테이블에서 Class_No는 기본키이고, Student_No는 외래키이다. R 테이블에서 Student_No는 R 테이블의 기본키이며, Score는 value다.

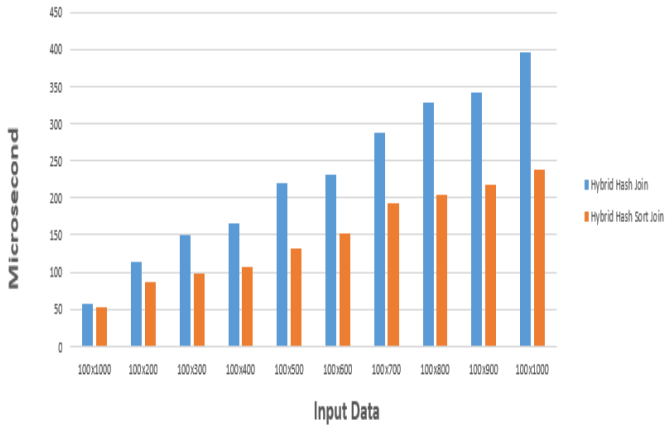

그림 3 S테이블 데이터 크기에 따른 성능비교

그림 3은 Hybrid Hash Join과 Hybrid Hash sort Join의 2차 해싱 단계의 수행 시간이다. R 테이블의 데이터의 크기는 100으로 가정했으며, S 테이블은 100~1000으로 실험했다. Hybrid Hash Join과 Hybrid Hash sort Join은 S 테이블의 데이터가 늘어남에 따라, 최대 66.7%의 성능이 향상되었다.

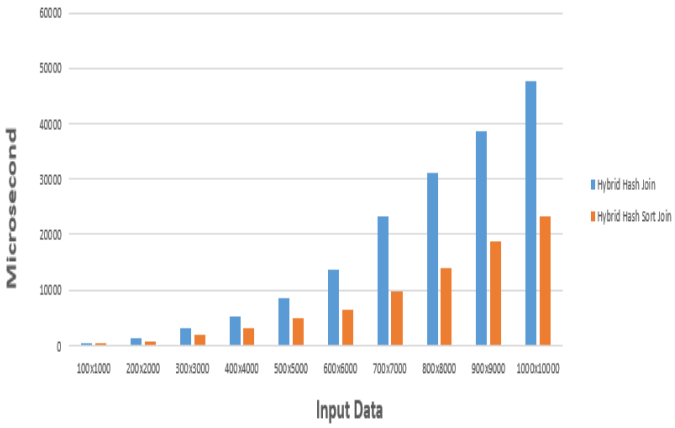

그림 4 R테이블과 S테이블의 데이터에 따른 성능비교

그림 4 역시 그림 3의 실험과 동일하다. 단 R 테이블은 100~1000, S 테이블은 R 테이블의 크기의 10배로 실험했다.
R 테이블과 S 테이블의 데이터가 증가함에 따라, 기존의 Hybrid Hash Join 알고리즘보다 Hybrid Hash sort Join 알고리즘의 성능이 최대 105%의 성능이 향상되는 것을 볼 수 있다.

4. 결 론

본 논문에서는 Hybrid Hash Join과 Hybrid Hash sort Join의 2차 해싱 단계 성능을 비교하여 실험하고 Hybrid Hash sort Join의 성능이 기존의 Hybrid Hash Join의 성능에 비하여 최대 105%를 개선하였음을 그림4. 통해 검증했다. 그 이유로 Hybrid Hash Join에서는 Student_No를 찾기 위해 모든 버킷에 대해 반복 연산을 수행하지만, Hybrid Hash Sort Join 알고리즘은 동일한 Student_No가 있는 레코드를 찾은 후, 나머지 레코드에 대해 모두 비교할 필요성이 없기 때문이다. 현재 본 논문에서는 단일 디렉토리에 버킷들이 연결되어 있다. 추후 단일 디렉토리를 다중 디렉토리로 확장하고 SSD, HDD등 다양한 저장 매체에서 성능 측정 및 연구 분석을 할 것이다.

참고문헌

[1] Y. Choi, B. On, G. S. Choi, I. Lee, A Comparative Study of PRAM-based Join Algorithms, Journal of KIISE, Vol.42, No. 3, pp.379-389, March. 2015.

[2] D. Kang, D. Jung, J. Kang and J. Kim, μ-tree: An ordered index structure for NAND flash memory, Proc. of the 7th ACM/IEEE International Conference on Embedded Software, pp. 144-153, 2007.

[3] J. Do and J. Patel, Join processing for Flash SSDs: Remembering past lessons, Proc. of 5th International Workshop on Data Management on New Hardware, Providence, Rhode-Island, USA, Jun. 2009.

[4] J. Patel, M. Carey and M. Vernon, Accurate modeling of the hybrid hash join algorithm, ACM SIGMETRICS Conference, Nashvile, 1994.

[5] A. Silberschatz, H. Korth and S. Sudarshan, Database system concepts, McGraw-Hill Education, 6th Edition, 2011.