Row-지향과 Column-지향 데이터베이스의 조인 질의 처리 비용 비교

오병중*, 안수민*, 김경창*
*홍익대학교 컴퓨터공학과
e-mail : lightclassic@nate.com, happydaygirl@nate.com, kckim@hongik.ac.kr

Comparision of Join Query Processing Cost in Row-Oriented and Column-Oriented Databases

Byung-Jung Oh*, Soo-Min Ahn*, Kyung-Chang Kim*
*Dept of Computer Engineering, Hong-Ik University

요 약

데이터 레코드를 가로(row-wise)로 저장하는 기존의 데이터베이스를 Row-지향 데이터베이스, 세로(column-wise)로 저장하는 데이터베이스를 Column-지향 데이터베이스라 정의하자. 본 논문에서는 Row-지향 데이터베이스와 Column-지향 데이터베이스에서 분석 workload 형태의 조인 질의를 처리하여 비교 우위 성능을 보이는 데이터베이스 시스템을 고찰하고자 한다. 객관적인 성능 실험을 위해 분석적 모델인 스타 스키마 벤치마크를 이용하였다. Nested Loop 조인과 Sort Merge 조인 기법을 사용한 실험에서 Column-지향 데이터베이스의 성능이 우수하게 나타났음을 확인할 수 있다.

1. 서론

데이터 레코드를 가로로 저장하는 기존의 Row-지향 데이터베이스 시스템에서는 데이터의 입출력과 수정이 쉬운 장점을 가지고 있었다. 하지만 조인 질의 처리 관점에서 보았을 때, 조인에 참여하지 않는 데이터까지 읽어와 처리를 하게 되므로 디스크 I/O 비용 측면에서 비효율적이다[4]. 상대적으로 데이터 레코드를 세로(column-wise)로 저장하는 Column-지향 데이터베이스에서는 조인에 참여하는 컬럼만을 읽어와 조인 질의를 처리하므로 저비용으로 훨씬 빠른 처리 속도를 구현할 수 있다[5].

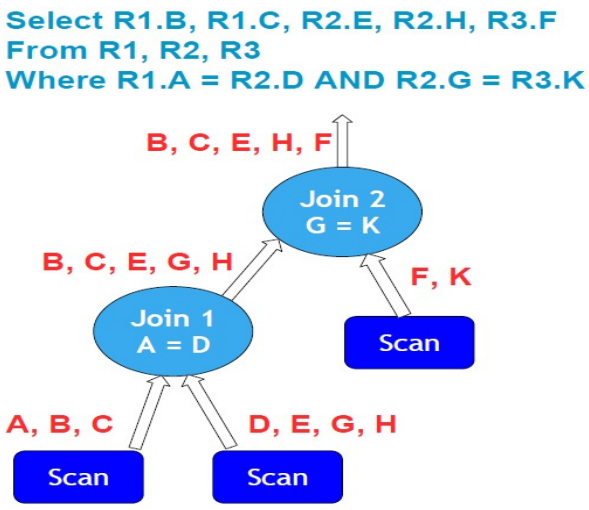
최근에는 MonetDB[3], 그리고 SybaseIQ와 같은 Column-지향 데이터베이스 시스템들이 출현되어 사용되고 있다. 이들 시스템을 만든 개발자들은 주로 읽기(read-intensive) 연산만을 실행하는 데이터웨어하우스, Decision Support System(DSS) 및 Business Intelligence(BI) 응용에서는 Column-지향 데이터베이스 시스템이 기존의 시스템 보다 몇 십 배의 성능 향상이 가능하다고 주장하였다.

본 논문의 구성은 다음과 같다. 2절에서는 Column-지향 데이터베이스에서 사용가능한 두 가지 조인 기법에 대해 알아본다. 3절의 성능 분석에서는 먼저 분석적 모델인 스타 스키마 벤치마크에 대해 간략하게 설명하고 이 벤치마크를 이용하여 Nested Loop Join과 Sort Merge Join 알고리즘을 적용한 Row-지향 데이터베이스 시스템과 Column-지향 데이터베이스 시스템의 I/O 비용을 알아본다. 그리고 4절에서는 결론에 대해 기술한다.

2. Column-지향 데이터베이스에서의 조인 처리 기법

본 절에서는 Column-지향 데이터베이스의 조인 기법에 대해 설명한다. 일반적으로 조인 기법에는 해당 컬럼(column)을 먼저 튜플로 구성하여 조인을 수행하는 조기 실체화(early materialization)와 해당 컬럼에 대해 먼저 조인을 수행하는 지연 실체화(late materialization) 두 가지 방법이 있다.

2.1 Early Materialization Join



(그림 1) Early Materialization Join 절차

Early Materialization 조인 기법에서는 먼저 필요한 컬

럼들을 가져와서 row-기반으로 생성한 다음 조인 연산을 수행한다. 기존의 Row-지향 데이터베이스와 다른 점은 조인을 위해 모든 튜플을 가져오는 것이 아니라 조인을 위해 필요한 컬럼들만 튜플로 구성한다는 것이다. 질의 처리 계획의 가장 앞 단계에서 Column-지향 데이터베이스에 저장되어 있는 컬럼들을 Row-지향 데이터베이스 형태로 구성하는 것이다. 다음의 그림 1은 Early Materialization의 절차를 나타내고 있다[4].

그림 1에서 볼 수 있듯이 Early Materialization에서는 질의 처리 첫 단계에서 해당 컬럼을 디스크에서 스캔해와 Row-기반의 형태로 구성한 후에 조인을 실행한다. 첫 단계에서는 릴레이션 R1, R2의 해당 컬럼들을 스캔하여 Row-기반 형태로 구성한다. 다음 단계로 R1.A = R2.D의 조인 질의을 수행하여 결과로써 A=D인 B, C, E, G, H 컬럼들을 반환받는다. 다음으로 릴레이션 R3의 F, K 컬럼을 스캔하여 Row-Store로 구성하여 전 단계에서 결과로 반환받은 B, C, E, G, H로 구성된 릴레이션과 두 번째 조인 조건 R2.G = R3.K의 조인 질의를 수행하여 최종 결과로써 컬럼 B, C, E, H, F로 구성된 릴레이션을 표시한다.

2.1 Late Materialization Join

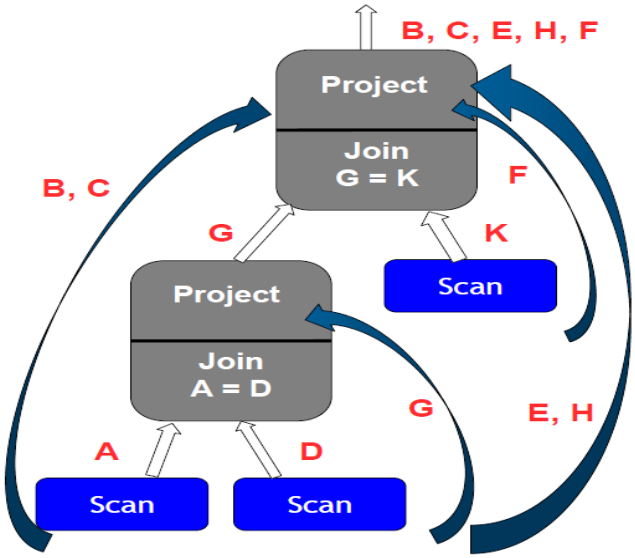
Late Materialization 조인 기법에서는 먼저 조인에 필요한 컬럼만을 가져와서 조인을 수행한 후에 질의 결과를 얻기 위해 필요한 컬럼들을 나중에 구성한다. Late materialization 조인 기법이 Early Materialization 조인 기법과 다른 점은 먼저 해당 컬럼들에 대해서 조인을 수행한 후에 조인을 만족하는 컬럼의 튜플 위치(tuple position)을 가지고 최종 projection에 참여하는 컬럼들을 가져온다.

따라서 Late Materialization 조인 기법에서는 가능한 질의 처리 계획의 나중 단계에서 컬럼의 프로젝션(Projection)을 위해 구성하는 방법이다[6]. 그림 2에서는 그림 1의 질의를 Late Materialization 기법으로 조인 질의를 처리하는 절차를 나타내고 있다[4].

Late Materialization 조인 처리에서는 가장 첫 단계로 먼저 수행되는 조인 질의 R1.A = R2.D를 위해 각 릴레이션 R1과 R2에서 컬럼 A와 컬럼 D를 스캔해온다. 조인 질의 R1.A = R2.D를 수행하고 반환받는 결과는 컬럼 내의 조인 질의에 부합하는 값이 위치한 포지션이다. 이 때, 포지션으로는 ranges, lists, bitmap 등의 여러 형태로 받을 수 있다[6].

다음 단계로 두 번째 질의 R2.G = R3.K를 수행하기 위해 전 단계에서 프로젝션된 포지션과 함께 컬럼 G와 K를 스캔하여 두 번재 질의를 수행하고 마찬가지로 조인 질의에 부합하는 컬럼 내 값을 가진 포지션을 프로젝션한다. 이제 첫 번째와 두 번째 질의를 통해 결과로써 받은 두 포지션을 position-wise AND 연산을 하여 최종적으로 두 조인 질의에 부합하는 포지션을 반환받는다.

마지막 단계로 이 position-wise AND 연산의 포지션을 SELECT 절에서 요구하는 각 컬럼 B, C, E, H, F 절에 매칭하여 해당이 되는 값만을 가져와 최종 결과를 보여주기 전에 Row-Store의 형태로 Stitch하여 표현하게 된다.



(그림 2) Late Materialization Join 절차

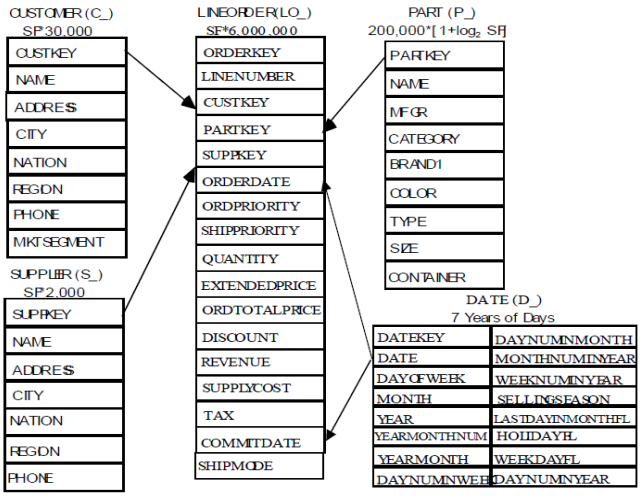
이미 기존의 Materialization 기법 연구에서는 Early Materialization 기법보다 Late Materialization 기법이 더욱 좋은 성능을 내고 있음을 설명하였다[4, 6]. 다음 절에서는 Row-지향 데이터베이스와 Column-지향 데이터베이스에서 조인 기법을 적용하여 디스크 I/O 비용의 효율적 측면에서 비교 분석하고자 한다.

3. 성능 분석

Column-지향 데이터베이스와 Row-지향 데이터베이스의 조인 질의 처리 디스크 I/O 비용의 객관적 비교 분석을 위해 데이터베이스 모델은 스타 스키마 벤치마크(Star Schema Benchmark)를 사용하였다[1]. 스타 스키마 벤치마크는 TPC-H를 기반으로 한 데이터 웨어하우징에서 복잡한 정보를 모델링하는 표준형 기술로써 중심이 되는 Fact Table을 중심으로 Dimension 테이블이 붙어있는 형태이다[2].

3.1 분석 환경

성능 분석에 사용된 스타 스키마 벤치마크의 스키마는 그림 3과 같고, 이용한 질의 Query flight 1.1과 이에 대응하는 FF(Filter Factor)는 <표 1>에 수록하였다[1]. 본 논문의 분석에서는 스타 스키마 벤치마크의 SF(Scale Factor)를 0.01로 하여 Lineorder의 총 튜플 수는 60000 튜플로 설정하였다.



(그림 3) 스타 스키마 벤치마크

<표 1> Query Flight 1.1

Query	SELECT sum(lo_extendedprice*lo_discount) as revenue FROM lineorder, date WHERE lo_orderdate = d_datekey AND d_year = 1993 AND lo_discount between 1 and 3 AND lo_quantity < 25;
FF	d_year = 1993 → FF is 1/7 lo_discount between 1 and 3 → FF is 0.5 lo_quantity < 25 → FF is 3/11

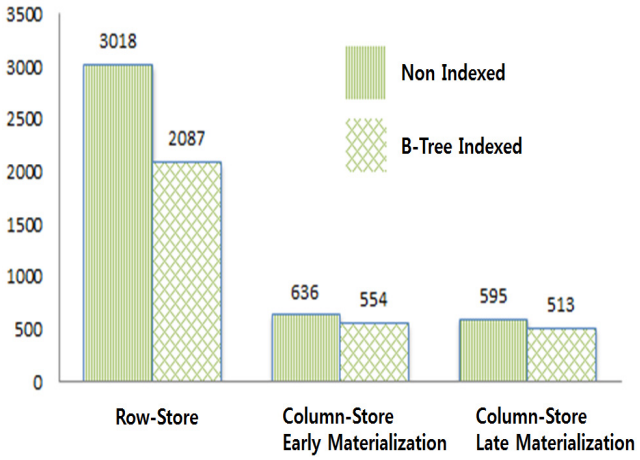
Row-지향 데이터베이스의 레이아웃을 그대로 Column-지향 데이터베이스로 가져왔으며, 이 때 릴레이션을 Column-지향 데이터베이스로 매핑하기 위해 MonetDB의 아키텍처를 사용하였다[3].

3.2 분석 결과

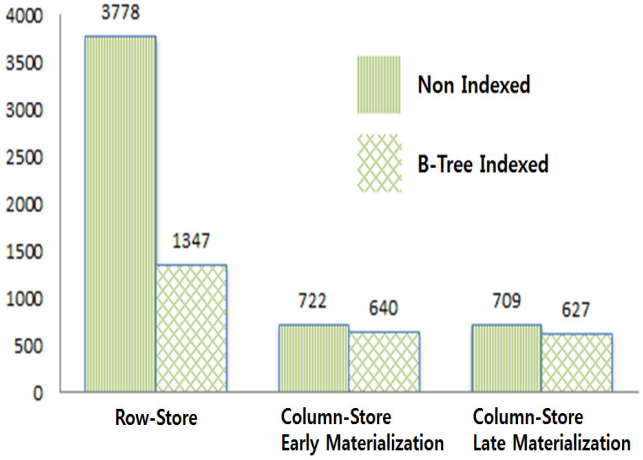
사용한 질의 처리 기법은 Page-Oriented Nested Loop Join과 Sort Merge Join을 사용하였다. 각각의 조인 기법을 적용하기에 앞서 Lineorder, Date 테이블에 Selection을 위한 인덱스가 없는 경우와 B-Tree 인덱스를 가지고 있는 경우를 가정하여 처리하였다. B-Tree 인덱스를 가지고 있는 가정에는 Lineorder 테이블의 discount 컬럼과 Date 테이블의 year 컬럼에 3-계층의 B-Tree 인덱스를 가지고 있다고 가정하였으며, 각 d_year = 1993, 1≤lo_discount≤3의 서술부에 부합하는 데이터를 가져오는 I/O 비용은 1로 산정하였다.

Column-지향 데이터베이스에서는 Early Materialization 기법과 Late Materialization 기법에 따라 I/O 비용이 얼마나 차이가 나게 되는가에 대한 분석도 병행하였으며, 이에 대한 결과는 그림 4, 5과 같이 나타났다.

분석 결과의 수치는 디스크 I/O 비용만을 따진 것이며, 메모리상에서의 연산과 CPU 비용은 고려하지 않았다.



(그림 4) Page-Oriented Nested Loop Join 처리 결과



(그림 5) Sort Merge Join 처리 결과

그림 4와 그림 5의 그래프에서 볼 수 있듯이 Row-지향 데이터베이스에서와 Column-지향 데이터베이스에서의 조인 질의 처리를 위한 디스크 I/O 비용을 비교 분석해 보았을 때, Row-지향 데이터베이스보다 Column-지향 데이터베이스에서 몇 배 낮은 비용이 드는 것을 알 수 있다. 또한 Column-지향 데이터베이스에서는 Early Materialization 조인 기법 보다 Late Materialization 조인 기법이 더 적은 비용이 필요로 함을 확인하였다.

4. 결론

본 논문에서는 기존의 Row-지향 데이터베이스와 Column-지향 데이터베이스의 디스크 I/O 비용적인 측면을 바탕으로 스타 스키마 벤치마크의 스키마와 일부 조인 질의 상에서 Page-Oriented Nested Loop Join과 Sort Merge Join을 수행하여 어떠한 데이터베이스 시스템이 조인 처리에 효율적인가에 대해 알아보았다. 이를 통해 데이터웨어하우스 환경에서는 같은 질의를 처리하더라도 Row-지향 데이터베이스 보다는 Column-지향 데

이터베이스 시스템이 더욱 비용 효율적이었음을 알 수 있었다. Selection 조건을 명시하는 컬럼에 B-tree 인덱스가 존재하는 디자인이 비용적인 측면에서 큰 효율을 거둘 수 있었다.

또한 Column-지향 데이터베이스 시스템 내에서도 조인에 참여하는 컬럼들을 질의 처리 앞 단계에서 모두 Row-Store 형태로 구성하는 Early Materialization 기법보다는 모든 질의 처리가 끝나고 마지막 단계에서 구성하여 결과를 표현해주는 Late Materialization 기법이 디스크 I/O 비용 측면에서 효율적인 성능을 보임을 알 수 있었다.

참고문헌

[1] Patrick E. O'Neil, Elizabeth J. O'Neil, and Xuedong Chen. "The Star Schema Benchmark (SSB)". Revision 3, June 5, 2009.

[2] 김도근. 롯데정보통신 IS 사업팀 DBA. "성능 항목별 DBMS 3종 비교 분석". http://www.oracle.com/technology/global/kr/pub/columns/dbms.html.

[3] P. A. Boncz. "Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications". Ph.d. thesis, Universiteit van Amsterdam, May 2002.

[4] Daniel J. Abadi, Peter A. Boncz, and Stavros Harizopoulos. "Column-oriented Database Systems". VLDB 2009 Tutorial.

[5] Peter Boncz, Marcin Zukowski, Niels Nes. "MonetDB/X100: Hyper-Pipelining Query Execution". CIDR Conference. 2005.

[6] Daniel J. Abadi, Daniel S. Myers, David J. DeWitt, Samuel R. Madden. "Materialization Strategies in a Column-Oriented DBMS". IEEE. 2007.