인메모리 관계형 데이터베이스에서 효율적인 범위 조인 방법에 대한 연구[1]

한혁*, 강조현*, 진성일**
*㈜리얼타임테크
**충남대학교 컴퓨터공학과
e-mail : hhan@realtimetech.co.kr

A Study on Efficient Range Join For In-Memory Relational Database Management System

Hyeok Han*, Jo-Hyeon Kang*, Sung-Il Jin**
*Realtimetech Inc.
**Dept. of Computer Engineering, Chungnam National University

요 약

Range Join 은 관계형 데이터베이스 시스템에서 제공하는 Join 연산 중에서도 특수한 형태로 비교적 연구 사례가 적고, 동등연산자(“=”)를 사용하는 Equi Join 보다 시간 소모가 많은 Join 중 하나이다. 특히, 대부분의 연구가 Range Join 의 성능을 보장하기 위하여 별도의 신규 색인을 생성하여 처리하는 방법을 제안하고 있다. 본 논문에서는 관계형 데이터베이스 시스템에서 제공하는 기본 자료형 컬럼으로 구성된 Range Join Predicate 과 인메모리 관계형 DBMS 의 기본 제공 색인인 T-Tree 를 활용하여 성능 효율적인 Range Join 방법을 제안한다.

1. 서론

관계형 데이터베이스 시스템의 Join 연산은 데이터베이스 응용 개발 시 가장 빈번하게 사용되는 질의 유형인 동시에 가장 시간 소모가 많은 연산 중 하나이다. 특히, 국내·외 상용 및 오픈 소스 기반의 수 많은 데이터베이스 시스템들 중 하나를 선택하여 응용에 활용하는 경우, 질의 복잡도가 높은 Join 연산의 성능을 핵심 평가 항목에 포함하여 시스템의 완성도 및 응용 적합도를 판단하는 경우를 자주 볼 수 있다.

Join 연산은 결과 집합의 유형에 따라 크게 Inner Join 과 Outer Join 으로 구분하며, Join 조건에 사용하는 비교 연산자에 따라 동등연산자(“=”)를 사용하는 경우는 Equi Join, 그 외(<,>,<=,>=)의 경우는 Non-Equi Join 으로 구분한다. 대부분 응용은 Equi Join 유형의 질의를 가장 많이 사용하나, Non-Equi Join 을 활용하여 질의를 구성하는 경우를 종종 볼 수 있다. 특히, 동일 테이블(S)의 두 컬럼(α, β)을 범위의 시작과 끝으로 사용한 Join predicate(A ≥ S.α AND A ≤ S.β, A BETWEEN S.α AND S.β)을 갖는 경우 이를 Range Join[1]이라고 한다. 예를 들면, 웹 로그 테이블의 접속 IP 에 대응하는 국가 코드를 식별하고자 할 때, IP 주소 영역(Start-IP, End-IP)별 국가 코드를 레코드로 갖는 맵핑 테이블과의 Range Join 을 수행하면 된다.

본 논문은 관계형 데이터베이스 시스템에서 Range Join 질의 처리의 성능 문제를 해결하기 위한 방법에 대해 제안한다.

2. 관련 연구

Range Join 은 관계형 데이터베이스 시스템에서 제공하는 Join 연산 중에서도 특수한 형태로 비교적 연구 사례가 많은 편은 아니다. 센서 네트워크 상에서 통신 효율성에 초점을 맞춘 분산 Range Join 알고리즘에 대한 연구[2]나, Range 와 유사하게 Lower 및 Upper 시간 값을 갖는 Interval 자료형 기반의 Join 방법에 관한 연구[3] 등은 모두 Join 수행에 적합한 새로운 색인을 제안하고 이를 기반으로 Join 알고리즘을 연구한 것으로, B-Tree 와 같이 기본적으로 시스템이 제공하는 색인을 활용하는 방법은 아니다.

PostgreSQL[4]은 “Range Type”이라는 특수한 형태의 자료형을 제공하고 이를 통해 Range Join 을 수행한다. 따라서, 사용자는 해당 자료형을 갖는 테이블 스키마 설계하고 시스템이 제공하는 레코드 삽입 방법에 따른 원시 데이터를 가공 처리하여 테이블에 저장하여야 한다, 특히, 앞선 연구 사례[2][3]와 마찬가지로 성

[1] 본 연구는 2016 년도 우수기술연구센터(ATC)사업 (과제명: 익스트림 트랜잭션 및 분산 확장성을 제공하는 대용량 비휘발성 메모리(SCM) 기반의 차세대 인메모리 빅 데이터베이스 시스템 상용 기술 개발)의 지원을 받았음.

능을 위해 별도의 GiST(Generalized Search Tree)[5] 색인을 생성하여야 한다.

본 논문에서는 관계형 데이터베이스 시스템에서 제공하는 기본 자료형 컬럼으로 구성된 Range Join Predicate 과 인메모리 관계형 DBMS 의 기본 제공 색인인 T-Tree 를 활용하여 성능 효율적인 Range Join 방법을 제안한다.

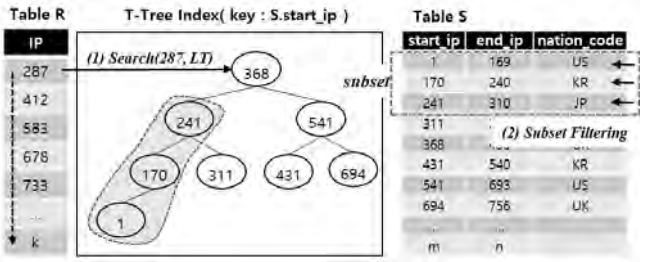
3. 기존 시스템의 색인 기반 Range Join

[그림 1]은 테이블 R 과 S 에 대한 관계형 데이터베이스 시스템의 전형적인 Range Join SQL 문이다. 이해를 돕기 위해, 테이블 R 은 웹 접속 로그 정보 테이블, 테이블 S 는 IPv4 영역별 국가코드가 기록된 맵핑 정보 테이블로 가정하며, Range Join 시 Index 를 활용하기 위하여 사전에 B.start_ip 컬럼을 Key 로 갖는 T-Tree 색인을 생성하였다고 가정한다.

```
SELECT B.nation_code FROM R
JOIN S ON R.ip >= S.start_ip AND R.ip <= S.end_ip;
```

[그림 1] 전형적인 Range Join SQL 문

Range Join 연산은 Index Search Join 과 유사한 과정을 거친다. Join 조건에 참여하는 테이블 S 의 컬럼(S.start_ip)을 Key 로 하는 색인을 먼저 식별하고, (1)테이블 R 의 각 레코드 값(R.ip)을 색인의 입력 값으로 하여 "R.ip >= S.start_ip" 조건을 만족하는 색인의 시작 Entry 검색을 통해 최종 검색 레코드를 포함하는 색인 Subset 을 식별한다. (2)식별된 색인 Subset 에 포함된 각 레코드에 대해 순차 접근을 통하여 두번째 Join 조건(R.ip <= S.end_ip)을 만족하는 레코드를 찾을 때까지 수행을 계속한다.[그림 2]



[그림 2] 기존 색인 기반 Range Join 방법

이러한 방식의 Range Join 수행 비용은 아래와 같이 정의 할 수 있다

Range Join 비용 = *Tcount(R) * (Icost + Fcont(**n**))*

- *TCount(R)* : 테이블 R 의 레코드 개수
- *Icost* : 테이블 R 의 각 레코드 단위 색인 비용
- *Fcost(n)* : **subset 의 각 레코드 단위 필터 비용**

4. 새로운 전용 색인 연산자를 활용한 Range Join

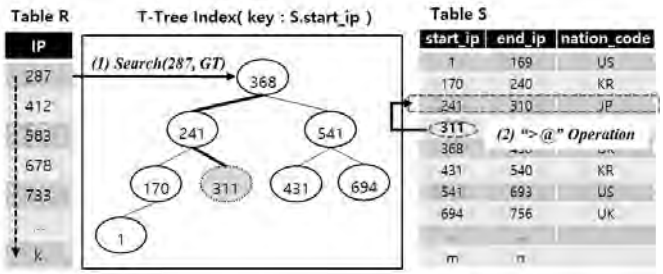
본 논문의 제안 방법은 기존 Range Join 이 색인을 통해 Subset 을 특정한 후 검색된 Subset 의 모든 레코드를 대상으로 Filtering 을 수행하는 것과 달리, (1)Range Join 조건 중 시작 조건(R.ip >= S.start_ip)을 반전하여 만족하는 첫 번째 색인 Entry 를 찾고, (2)해당 색인 Entry 의 직전 Entry(-1) 레코드에 접근하여 끝 조건(R.ip <= S.end_ip)을 평가하는 방식으로 수행한다.

본 논문에서는 관계형 색인의 기본 연산자 (=,<,>,<=,>=) 이외에 효율적인 Range Join 을 위한 비교연산자 " >@ "를 정의하여 [그림 3]과 같이 Range Join SQL 문을 수정하였다.

```
SELECT B.nation_code FROM R
JOIN S ON R.ip >@ S.start_ip AND R.ip <= S.end_ip;
```

[그림 3] 신규 Range Join 연산자 기반 SQL 문

Join 조건에 신규로 정의한 색인 연산자 ">@"을 조인 조건으로 명시한 Range Join 의 처리 과정은 [그림 4]와 같다.



[그림 4] 신규 색인 연산자 활용 Range Join

본 논문에서 제안하는 신규 색인 연산자 기반의 Range Join 수행 비용은 아래와 같이 정의 할 수 있다

Range Join 비용 = *Tcount(R) * (Icost + Fcont(**1**))*

- *TCount(R)* : 테이블 R 의 레코드 개수
- *Icost* : 테이블 R 의 각 레코드 단위 색인 비용
- *Fcost(1)* : **상수 비용. 항상 한 레코드만 필터 수행**

5. 실험

실험을 위해 신규 색인 연산자는 소스 수정이 가능한 인메모리 DBMS 를 이용하여 구현하였으며, 실험은 기존 Range Join, 제안 기술 적용한 Range Join 그리고 PostgreSQL 의 Range Type[4]을 이용한 Range Join 질의로 나누어 성능을 평가하였다.[그림 5/6/7]

성능 실험은 3 절에서 언급한 웹 접속 로그 정보 테이블(R)과 국가 코드 맵핑 테이블(S) 기반의 Range Join 질의를 수행하는데 걸리는 시간을 측정하며, 테이블 R 은 원본 웹 로그 파일의 데이터 양을 증가 시키면서(100MB, 500MB, 1GB) 질의를 수행하였으며, 테이블 S 에 사용되는 데이터는 약 14 만 레코드의 IPv4 범위 값(start_ip, end_ip)과 국가코드를 갖는 테이블로 구성하였다.

```
SELECT S.nation_code FROM
(SELECT DISTICT ip FROM R) A
JOIN S ON A.ip >= S.start_ip AND A.ip <= S.end_ip);
```

[그림 5] 기존 시스템의 Range Join 질의

```
SELECT S.nation_code FROM
(SELECT DISTICT ip FROM R) A
JOIN S ON A.ip >@ S.start_ip AND A.ip <= S.end_ip);
```

[그림 6] 신규 연산자를 적용한 Range Join 질의

```
SELECT S.nation_code FROM
(SELECT DISTICT ip FROM R) A
JOIN S ON A.ip <@ S.ip_range;
```

[그림 7] PostgreSQL 의 Range Type 을 이용한 Range Join 질의

[표 1] Range Join 수행 성능 비교 결과(단위:초)

구분	100MB 테이블	500MB 테이블	1GB 테이블
기존 시스템 Range Join	101.88	204.32	387.92
신규 연산자 Range Join	1.92	6.87	17.8
PostgreSQL Range Type 기반 조인	1.47	6.25	17.2

실험 결과는 [표 1]과 같다. 본 논문에서 제안한 방법의 Range Join 방법이 기존 시스템 대비 약 20 배 이상 성능이 빨라진 것을 확인 할 수 있고, Range Type 을 이용하여 Range Join 을 수행한 PostgreSQL 와 동등한 성능을 내는 것을 확인할 수 있다.

6. 결론

기존 시스템의 Range Join 방법은 색인으로 식별된 subset 내의 모든 레코드를 순차 접근하면서 두 번째 조인 조건을 평가하나, 신규로 제안한 색인 연산자를 사용하는 경우 Equi Join 처럼 단 건의 최종 레코드를 대상으로 조인 조건이 평가되므로 훨씬 빠른 성능의 색인 기반 Range Join 을 구현할 수 있었다.

다만, 본 제안은 범위 테이블의 값이 중첩(Overlap) 되지 않는 경우의 알고리즘으로 향후 범위 값이 중첩 되어 있는 테이블을 대상으로 하는 다차원 색인 기반의 Range Join 연구를 추가적으로 추진할 예정이다.

참고문헌

[1] https://community.dev.hpe.com/t5/Vertica-Blog/What-Is-a-Range-Join-and-Why-Is-It-So-Fast/ba-p/223413

[2] Pandit, Aditi, and Himanshu Gupta. "Communication-efficient implementation of range-joins in sensor networks." *International Conference on Database Systems for Advanced Applications*. Springer Berlin Heidelberg, 2006.

[3] Enderle, Jost, Matthias Hampel, and Thomas Seidl. "Joining interval data in relational databases." *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 2004.

[4] https://www.postgresql.org/docs/current/static/rangetypes.html

[5] Joseph M. Hellerstein, Jeffrey F. Naughton and Avi Pfeffer. Generalized Search Trees for Database Systems. Proc. 21st Int'l Conf. on Very Large Data Bases, Zürich, September 1995, 562–573.