데이터 스트림에서 MJoin을 이용한 다중 조인 질의의 최적화 기법

이헌주° 박 석
서강대학교 컴퓨터학과
{hunz°, spark}@sogang.ac.kr

Optimization of Multiple Join Queries using MJoin over Data Streams

Hunjoo Lee° Seog Park
Department of Computer Science, Sogang University

요 약

센서 네트워크에서 각 센서는 제한된 개수의 속성을 기반으로 한 스키마를 가진다. 사용자는 여러 센서에서 수집된 정보를 종합적으로 살펴보고자 하기 때문에 여러 센서에서 수집된 정보를 조인하는 질의가 필수적이다. 또한, 센서가 수집하는 데이터를 중앙 서버로 보내는 경우 스트림의 형태로 입력되므로 빠른 조인 질의의 질의 수행 계획을 수립해야 한다. 본 논문에서는 기존에 여러 입력 스트림을 조인하는 MJoin을 기반으로 한 다중 조인 질의의 효율적인 최적화 기법을 제안한다. 또한 다중 조인 질의에 대해 기존의 MJoin을 적용한 기법과 본 논문에서 제안하는 다중 조인 질의 최적화 질의 계획 수립 기법을 비교, 분석한다.

1. 서 론

센서 네트워크에서는 각 센서가 데이터를 수집하여 데이터를 최종적으로 수집하는 중앙 서버로 전송한다. 각 센서에서 수집된 데이터는 스트림의 형태로 전송되며 그 데이터가 담고 있는 정보의 종류는 한정적이다[1]. 센서가 수집하는 데이터의 종류가 한정적이기 때문에, 사용자가 총체적인 결과를 얻고자 할 때에는 입력되는 데이터를 기반으로 중앙 서버에 조인 질의를 등록하여야 한다. 또한, 센서 네트워크로부터 전송되는 데이터는 스트림의 형태이기 때문에 그 전체 양은 잠재적으로 무한하다[2]. 이러한 패러다임에서는 기존의 DBMS와는 다르게 대부분의 질의가 연속된 형태로 등록되고 실시간으로 입력 스트림을 메모리상에서 빠르게 처리해야 한다.

다중의 조인 질의가 시스템에 연속 질의 형태로 등록되면 시스템에서는 질의에 대한 질의 실행 계획(query execution plan)을 수립한다. 수립된 질의 실행 계획에 따라 연산 결과 튜플(tuple)이 내보내지는 속도와 시스템이 질의를 처리하기 위해 필요로 하는 저장 공간의 양이 변하게 된다. 질의가 포함하는 여러 연산자들 가운데에서 조인 연산자는 가장 높은 처리 비용을 요구하는 연산자들 가운데에 하나이므로, 조인 연산자 실행 계획을 수립하는 기법에 의해 전체 질의 처리 시간이 큰 영향을 받게 된다. 이와 더불어 조인 연산자는 경험론적인 질의 실행 계획 수립 방법에서 선택(selection) 연산자 이후에 평가되므로 질의가 내포할 수 있는 선택 연산은 조인 연산자를 평가하기 이전에 처리되었다고 가정한다. 따라서 본 논문에서는 질의 처리에 가장 큰 영향을 미치는 조인 연산자에 초점을 맞출 것이다.

전통적인 DBMS 연구 분야에서는 하나의 조인 질의의 효율적인 질의 실행 계획을 수립하기 위해서 지수 시간(exponential time)이 요구 되었다[3]. 이렇게 오랜 시간이 걸리는 기법은 실시간으로 빠르게 스트림이 입력되는 데이터 스트림 관리 시스템(DSMS)에 적용할 수 없다.

그동안 많은 데이터 스트림 연구 분야에서는 비용 및 저장 공간의 효율성 측면에서 조인 질의의 질의 실행 계획을 수립하는 기법을 제안하였는데 대부분 이진 조인(binary join)기반으로 연구가 진행되었다. 이진 조인 연산자는 입력을 두 개로 한정하므로 이진트리(binary tree)의 형태로 질의 계획이 수립된다. 이진 조인을 기반으로 수립된 이진트리 형태인 조인 질의 실행 계획에서 상위의 조인 연산자가 수행되기 위해서는 하위의 조인 연산자의 결과가 끝날 때까지 기다려야 하므로 블록킹(blocking)되는 문제점을 가진다. 또한 대부분의 연구가 하나의 조인 질의에 대해 질의 실행 계획을 수립하거나 [4, 5, 6] 여러 조인 질의에 대해 중간 결과를 공유하는 형태의 질의 실행 계획을 수립하지만 기존의 비효율적인 이진 조인 기반의 기법을 제안하였다[7].

본 논문에서는 기존의 다중 조인 질의에 대한 공유된 질의 실행 계획 수립 기법의 미비함을 극복하고 효율적인 다중 입력 조인 연산자를 기반으로 여러 조인 질의에 대한 효율적인 공유 질의 실행 계획 수립 기법과 처리 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 이진 조인 기반의 질의 처리에서 발생하는 문제점을 해결하는 MJoin에 대해 간략하게 설명하고 3장에서는 기존의 MJoin을 다중 조인 질의로 확장할 때의 문제점과 제

본 연구는 한국과학재단 특정기초연구(R01-2006-000-10609-0)지원으로 수행되었음.

안하는 기법을 기술한다. 4장에서는 기존의 MJoin과 제안하는 기법을 분석, 비교하며 5장에서는 관련 연구들에 관해 다룬다. 마지막으로 6장에서는 본 논문에서 다루는 연구 결과와 향후 연구 과제를 기술하며 결론을 맺는다.

2. MJoin [4]

이진 조인 기반의 조인 질의는 이진트리의 형태로 질의 실행 계획이 수립되므로 블록킹되는 문제점을 가지고 있다. 데이터 스트림 환경에서는 잠재적으로 무한한 양의 데이터가 시스템에 계속적으로 입력되므로 블록킹되는 질의 실행 계획은 시스템이 가진 메모리양의 한계를 넘어서게 되어 입력 스트림의 샘플링(sampling)이나 로드 쉐딩(load-shedding)을 요구하게 된다. MJoin은 이러한 이진 조인 기반의 형태에서 벗어나 여러 스트림을 입력으로 가질 수 있는 효율적인 조인 처리 기법으로 제안되었다.

MJoin은 전통적인 대칭 해시 조인(symmetric hash join)에서 발전된 형태이다. 즉, 기존의 대칭 해시 조인과 다르게 여러 입력을 가질 수 있으므로 중간 결과를 다음 연산자로 넘기지 않고 여러 스트림과의 조인 결과를 내보낸다.

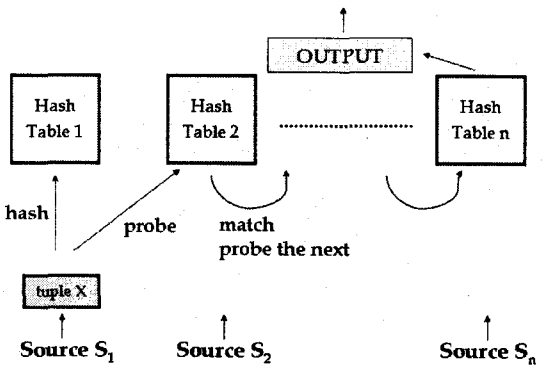


그림 3. MJoin의 처리 구조

위 그림 1은 MJoin의 처리 구조를 나타낸 것이다. 만일 입력 스트림 S1에서 새로운 튜플이 들어오면, S1에 대한 해시 테이블에 들어온 튜플을 삽입하고, 다음 입력 스트림에 대한 해시 테이블을 조사하게 된다. 만일 새로 입력된 튜플이 다른 해시 테이블에 있는 값들과 모두 매치되면 결과를 내보낸다. 하지만 MJoin은 평가 순서에 대한 문제를 가지고 있다. 다시 말해서 위 그림에서 1번 해시 테이블에서 n-1번 해시 테이블까지 조사를 수행하면서 모두 매치가 되었지만 n번 해시 테이블에서 매치가 되지 않는다면 그동안 수행했던 계산이 불필요하게 된다. MJoin에서는 이러한 문제를 그동안의 선택 비율을 유지하면서 해결한다. 즉, 각 입력 스트림에 대해 다른 입력 스트림과의 선택 비율(selectivity)을 오름차순으로 평가한다. 이렇게 되면 초반에 매치 되지 않을 확률이 높은 스트림과 평가가 이루어지게 된다.

MJoin은 기본적으로 시스템의 메모리가 가득 찼을 때 디스크에 접근하여 다른 해시 테이블을 구성하는 구조를 가지고 있으나 본 연구에서는 모든 입력 스트림이 슬라이딩 윈도우를 구성하여 매 시간 단위마다 제한된 양의 튜플을 입력으로 가지는 환경을 가정한다. 따라서 MJoin의 해시 테이블이 시스템의 메모리를 넘어서는 경우를 제외하므로 모든 MJoin의 처리는 메모리상에서 이루어진다.

3. 문제 정의 및 제안 기법

다중 조인 질의를 처리할 때 우선적으로 고려해야 할 점은 질의들을 처리할 때 중간 결과를 서로 공유하는 것이다. 이와 더불어 중간 결과를 공유할 때 각 조인 질의가 내포하고 있는 입력 스트림의 서로 다른 윈도우 크기를 고려해야 한다. 또한 새로운 질의가 입력되거나 삭제될 경우 입력 스트림의 처리에 영향을 적게 미치면서 동적으로 공유된 질의 계획을 재구성해야 한다. 마지막으로, 각 조인 질의 평가는 블록킹 되지 않는 형태로 처리되어야 한다.

MJoin은 여러 입력 스트림을 한 번에 처리할 수 있는 조인 연산자로 질의 처리 결과를 블록킹되지 않는 형태로 빠르게 내보낸다. 또한 기존의 이진 조인에 기반한 질의 실행 계획과 다르게, 평가 순서를 선택 비율이 낮은 스트림의 값부터 선택 비율이 높은 스트림의 값으로 정렬하는 것으로 최적화하는 장점을 가진다. 하지만 MJoin은 여러 스트림을 입력으로 가지기 때문에 다른 질의들과 더 많은 중간 결과를 공유하여 처리할 수 없는 문제점이 있다.

본 장에서는 하나의 조인 질의에 대해서 구성되는 MJoin을 나누고 여러 조인 질의로 확장하여 공유 질의 계획을 구성하는 방법에 대해 기술한다.

Q1 = Select * From R[2min] Join S[2min]
Q2 = Select * From R[1min] Join S[1min] Join T[1min]

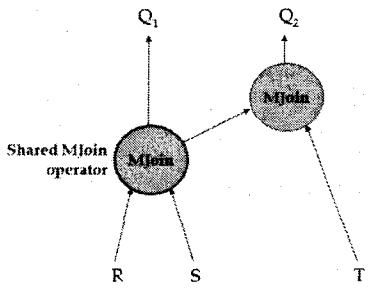


그림 4. 공유된 조인 질의 계획

3.1 공유된 큐에 대한 라우팅

조인 연산의 중간 결과를 공유하기 위해서는 각 질의가 내포하는 윈도우의 크기에 따라 중간 결과를 다음 조

인 연산자로 알맞게 내보내야 한다. 위 그림 2에서 질의 Q1의 결과로 나온 튜플은 Q2가 원하는 윈도우의 크기보다 크기 때문에 모든 결과를 Q2의 입력으로 보내서는 안 된다.

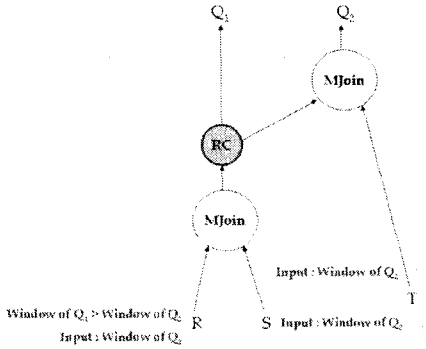


그림 5. 라우팅 연산이 추가된 공유 질의 계획

라우팅 연산은 조인 연산과 비교하여 상대적으로 매우 낮은 계산 비용을 가져야 하므로 간단한 전략을 사용한다. 공유 연산자는 입력으로 그 결과를 공유하는 질의들이 사용하는 윈도우 크기 중 가장 큰 것을 입력으로 취한다. 또한 각 튜플은 시스템에 입력될 때의 시간을 알려주는 타임스탬프를 가지고 있는데, 조인 결과로 나온 튜플은 조인에 참여한 튜플 중 가장 오래된 타임스탬프를 자신의 타임스탬프로 한다. 위 질의 Q1은 모든 입력 스트림에서 최근 2분 동안의 튜플을 윈도우로 사용하고 Q2는 모든 입력 스트림에서 최근 1분 동안의 튜플을 윈도우로 사용하고 있으므로 라우팅 연산을 수행하는 RC에서는 현재시간 - 튜플_타임스탬프 <= 2분에 해당하는 튜플을 Q1의 결과로 내보내고, 현재시간 - 튜플_타임스탬프 <= 1분에 해당하는 튜플을 Q2의 입력으로 보낸다. 따라서 위 예에서는 Q1로 모든 결과를 내보내며 Q2의 입력 튜플은 현재시간 - 튜플_타임스탬프 <= 1분에 만족하는 튜플을 내보낸다.

3.2 연산자 공유 조건

연산자를 공유하는 것은 대부분의 경우에 공유하지 않는 경우보다 저장 공간과 계산 비용 측면에서 유리하다. 하지만 연산자를 공유하면서 입력 윈도우의 크기도 커져 최악의 경우에 공유하지 않는 경우보다 조인 처리 과정에서 더 많은 양의 튜플을 사용할 수 있다. 따라서 연산자를 공유하여 처리할 경우에 그렇지 않은 경우와 비교하는 연산 과정이 필요하다. 공유된 연산자가 사용하는 입력 스트림에서 사용하는 윈도우 크기는 다음과 같다고 하자.

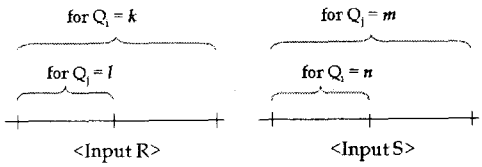


그림 6. 공유 연산자에서 사용하는 입력 윈도우

위 그림 4는 질의 Q_i가 입력 R에 대해 k만큼의 윈도우를, S에 대해 n만큼의 윈도우를 사용하고, 질의 Q_j가 입력 R에 대해 l만큼의 윈도우를, S에 대해 m만큼의 윈도우를 사용하는 예를 나타낸 것이다. 만일 위 질의들이 공유되지 않고 처리되는 경우에 Q_i의 조인 연산자는 $k*n$만큼의 튜플들을 평가하고, Q_j의 조인 연산자는 $l*m$만큼의 튜플들을 평가하게 된다. 이와 반대로 공유된 조인 연산자는 해당 입력에서 가장 큰 윈도우의 튜플들을 입력으로 사용하므로 $k*m$만큼의 튜플들을 사용하게 된다.

따라서 어떤 공유할 연산자의 중간 결과를 m개의 질의 Q_1, Q_2, ..., Q_m이 이용 가능하고, k개의 입력 스트림 R_1, R_2, ..., R_k이 해당 공유 연산자에서 사용되며, 각 질의 Q_i가 입력 R_j에 대해 W_{ij}의 윈도우 크기를 가지질 때, 각 입력 R_j에 대해 가장 큰 윈도우 크기가 W_{j*}라면 다음과 같은 조건을 만족 시킬 때 공유 가능하다.

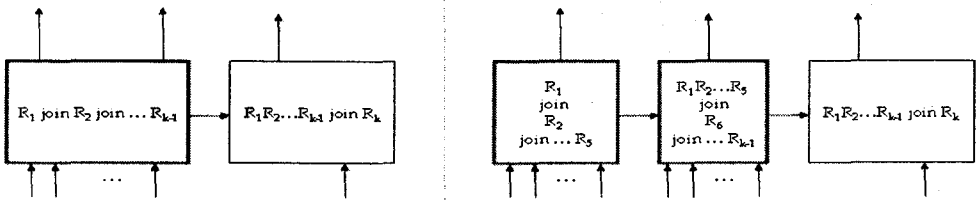
$$\sum_{i=1}^{m}\left(\prod_{j=1}^{k} W_{ij}\right) \geq \prod_{j=1}^{k} W_{j*}$$

수식 1. 연산자 공유 조건

3.3 조인 하위 집합 공유 문제

여러 조인 질의가 데이터 스트림 관리 시스템에 등록되면 조인 질의들 가운데에 공유할 연산자를 찾아야 한다. 공유할 연산자를 찾게 되면 최종 공유 질의 계획을 구성하여 입력 스트림에 대해 평가를 수행한다. 공유되는 연산자의 수와 형태에 따라 전체 질의들의 평균적인 질의 평가 속도도 영향을 받는다. 그림 5에서와 같이 하나의 연산자를 공유시키면 쪼개어 지는 수가 적어지므로 MJoin의 특성 상 평가 순서(probing sequence)가 본래 MJoin을 이용하여 평가하는 최적 순서에 근접할 수 있다. 하지만 공유 연산자를 사용하는 질의의 수가 적어지게 된다. 반대로 두 개의 공유 연산자를 사용하면 더 많은 질의들이 사용할 수 있으나 본래의 튜플 평가 순서에서 크게 벗어나게 된다. 본래의 평가 순서를 위배하는 경우보다 많은 질의에 걸쳐 연산자가 공유되면 저장 공간의 측면과 중간 결과를 한번만 계산하면 된다는 장점으로 인해 더 많은 이득을 얻을 수 있으므로 본 논문에서는 많은 공유 연산자를 찾아내는 것에 초점을 맞출 것이다.

우리는 이 문제를 조인의 하위 집합 공유 문제(Join Subset Sharing Problem)로 명명하였다. 조인의 하위 집합 공유 문제는 조인 질의를 하나의 집합으로 가정하고 입력 스트림을 해당 집합의 원소로 본다. 전체 조인



<1 shared sub-MJoin> <2 shared sub-MJoins>

그림 7. 하나의 공유 연산자를 가진 조인 질의 계획과 두개의 공유 연산자를 가진 조인 질의 계획

집합들 중 여러 조인 집합에 포함되는 가능한 많은 하위 집합들을 찾는 것이 목적이다. 이렇게 찾아낸 하위 집합은 최소한 두개 이상의 원소를 가져야 하며 최소한 두 개 이상의 조인 집합에 속해야 한다.

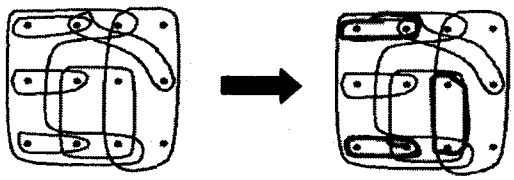


그림 8. 조인 하위 집합 공유 문제

위 그림 6에서 테두리는 조인 집합의 경계를 나타내는 것이고 검은 점은 입력 스트림인 원소를 나타낸 것이다. 만일 위 그림에서 왼쪽과 같이 조인 집합이 주어졌을 때, 공유 하위 집합을 최대한의 개수로 찾아낸 것이 오른쪽과 같다. 그림 6의 오른쪽에서 굵게 표시한 부분이 공유되는 연산자를 나타낸 것이다.

3.4 다중 조인 질의에서 공유 기법

본 논문에서는 각 조인 질의를 집합의 형태로 표기한다. 예를 들어 다음과 같은 조인 질의가 시스템에 등록되어 있다고 가정하자.

Q1 : select * from R join S
Q2 : select * from R join S join T join U

위 질의들은 다음과 같이 표기한다.

Q1 = {R, S}
Q2 = {R, S, T, U}

위와 같이 등록된 조인 질의들 중 공유가 가능한 조인의 하위 집합을 가장 많은 수로 추출해 내는 가장 단순한 방법은 모든 가능한 경우를 평가하는 방법이다. 하지만 이 방법은 지수 시간(exponential time)의 시간 복잡도를 가지므로 입력 스트림이 끊임없이 시스템에 들어오는 환경에 적용 불가능하다. 현재까지는 이 문제를 다항 시간(polynomial time)에 푸는 알고리즘을 찾지 못하였다. 본 문제는 전통적인 집합 커버 문제(Set Cover Problem)와 유사하며 집합 커버 문제는 이미 대표적인 NP-Complete문제로 증명되었다[8]. 따라서 본 논문에서는 최적의 결과에 근접할 수 있는 경험론적 욕심쟁이 방법(heuristic greedy method)을 제안한다.

전체 조인 집합이 주어졌을 때, 먼저 가장 작은 원소 개수를 포함한 집합을 선택한다. 만일 가장 작은 원소 개수를 가진 집합이 복수 개 존재한다면 많은 질의에 포함되는 집합을 선택한다. 그리고 선택된 집합의 원소를 모두 포함하는 집합들에서 각 원소들을 묶어 하나의 원소로 치환한다. 하나의 원소만을 가진 집합은 이미 조인 질의 계획이 수립된 것을 의미하므로 다음 단계에서 제외된다. 위 과정을 모든 질의가 제외될 때까지 계속하여 반복한다. 이를 Pseudo Code로 나타낸 것이 알고리즘 1과 같다.

```
if (there is only 1 query in initial step){
  Terminate algorithm;
} else {
  int k, m; //k : k-th min value, m : # of shared queries
  Set ss[], os;
  for (there are no queries){
    k := m := 1;
    while ((m<=1)&&(k<the # of remaining queries)){
      ss[] := subsets with k-th min # of elements;
      os : = one of ss[] involved in max # of queries;
      m := the number of queries involving os;
      if ((m<=1)||(not satisfy join sharing condition))
        k++; //select next smallest join subset
    }
    replace elements in queries to os;
    if (there are query sets only one element)
      exclude the query sets;
  } }
```

알고리즘 1. 다중 조인 질의 공유 알고리즘

예를 들어 다음과 같은 질의가 등록되어 있다고 가정하자.

Q1 = {R, S}
Q2 = {R, T}
Q3 = {S, T}

Q4 = {R, S, T}
Q5 = {R, S, T, U}

첫 번째 단계에서 Q1, Q2, Q3이 가장 작은 수의 원소를 가지고 있으며 모두 3개의 집합에 속할 수 있으므로, 어떤 것을 선택해도 무방하다. 여기에서는 가장 먼저 등록된 Q1을 선택한다. 이후 R, S 두 원소를 모두 포함하는 질의에서 해당하는 두 원소가 하나의 원소로 치환된다. 여기에서는 Q1 = {{R, S}}, Q4 = {{R, S}, T}, Q5 = {{R, S}, T, U}로 치환된다. Q1은 하나의 원소만을 가지게 되어 다음 단계에서 제외된다. 두 번째 단계에서 Q4가 두개의 원소만을 가지고 두개의 질의 속할 수 있으므로 Q4가 선택된다. 모든 질의가 제외될 때까지 이 과정을 반복하면, 다음과 같은 결과를 얻게 된다.

Q1 = {{R, S}}
Q2 = {{R, T}}
Q3 = {{S, T}}
Q4 = {{{R, S}, T}}
Q5 = {{{{R, S}, T}, U}}

위 결과는 최종 조인 질의 계획이고, Q1의 질의 결과는 Q4와 Q5에서 사용가능하며 Q4의 중간 결과를 Q5에서 사용하게 됨을 의미한다.

3.5 시스템 구성

우리는 위에서 제안한 기법으로 시스템 프레임워크를 설계하였다.

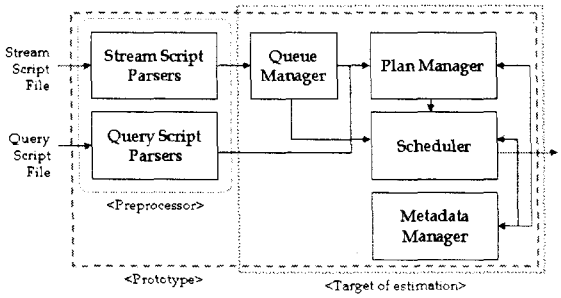


그림 9. 다중 조인 질의 처리 시스템

위 그림 7은 전체 시스템 구조를 나타낸 것이다. 스트림 데이터의 스크립트 파일과 질의 파일을 읽어 들이는 부분은 실험을 위한 전처리 과정으로 실제 시스템의 성능에 영향을 미치지 않는다. 질의 처리 부분은 크게 질의 계획 생성 및 관리자, 스케줄러, 메타데이타 관리자로 구성되어 있다. 질의 계획 관리자에 의해 다중 조인 질의의 최적화가 이루어지고, 스케줄러는 해당 타임 단위당 하나의 조인 연산자를 실행한다. 본 시스템에서는 [9]에서 제안된 Chain-flush 기법을 적용한다. 마지막으로 메타데이타 관리자는 연산자를 실행한 후 변화된 선택 비율 등의 정보를 저장한다.

4. 제안하는 기법의 분석 및 비교

기존의 MJoin은 하나의 질의에 대해 하나의 연산자로 질의 실행 계획을 수립한다. 이는 매우 짧은 시간에 이루어지며 새로운 질의가 등록되거나 삭제되는 경우에도 기존의 질의 실행 계획에는 변화가 없다. 본 논문에서 제안하는 기법은 다중 조인 질의에 대해 경험론적 욕심쟁이 방법으로 공유된 형태의 질의 실행 계획을 수립한다. 시스템에 등록된 질의의 수가 m개이고 입력 스트림의 수가 n개 라면 조인 질의가 입력으로 가질 수 있는 최대 입력은 n개가 된다. 질의 실행 계획을 수립하기 위해서 먼저 각 질의가 가진 입력 스트림의 수를 평가해야 하는데, 시스템에는 m개의 질의가 있고 각 질의는 최대 n개의 입력을 가질 수 있으므로 $O(mn)$의 시간이 걸린다. 또한 각 질의에서 선택된 원소들을 하나의 원소로 치환해야 하는데, m개의 질의를 n번 탐색해야 하므로 $O(mn)$의 시간이 소요된다. 마지막으로 하나의 원소만을 가지는 질의를 다음 단계에서 제외시켜야 하므로 하나의 원소만을 가지는 질의는 $O(m)$ 시간 내에 찾을 수 있다. 따라서 각 단계 마다 $O(mn)+O(mn)+O(m)=O(mn+m)$의 시간이 요구된다. 이 단계는 최악의 경우 모든 질의가 공유하는 부분이 없다고 가정하면 매 단계에서 제외되는 질의는 하나의 뿐이므로 총 m번 반복된다. 따라서 전체 알고리즘의 수행 시간은 $O(m)\times O(mn+m)=O(m^2n+m^2)$이 된다. 이는 기존의 MJoin이 모든 질의에 대해 질의 실행 계획을 수립하는 시간보다 오래 걸리지만 초기 질의 실행 계획이 수립된 후 입력 스트림을 처리하는 시간과 메모리양이 줄어들게 된다.

질의 처리를 할 때 각 튜플이 계산되는 시간을 a, 각 스트림i의 윈도우 크기를 W_i, 각 스트림 i의 선택비율을 s_i라 가정하면 하나의 연산자에 대해 다음과 같이 비용 계산을 할 수 있다.

$$\sum_{i=1}^{n} aW_n + a\prod_{i=1}^{n} W_n s_i$$

위 수식에서 왼쪽 부분은 해시 테이블에 입력되는 전체 튜플에 대한 비용이고 오른쪽 부분은 연산 과정에서 요구되는 처리 비용을 나타낸 것이다. 다음은 서로 다른 1000개의 질의에 대해 전체 질의 처리 비용을 비교한 결과이다. 본 분석에서 최대 입력 스트림의 수는 20개로 제한하였다. 편향정도(Skewness)는 전체 질의에서 사용하는 입력 스트림이 치우친 정도를 나타내는 척도로 수치가 커질수록 어떤 스트림을 여러 질의에서 사용한다는 의미이다.

Skewness	MJoin	제안 기법	비용개선비율(%)
0	12583	9891	21.39
1	12110	8920	26.34
2	12381	8259	33.29
3	12247	7972	35.72

표 1. 사용하는 입력 스트림에 따른 1000개의 질의에 대한 비용 비교 표

위 표1에서 알 수 있듯이 여러 질의에 걸쳐 사용되는 입력 스트림이 한쪽으로 치우칠수록 여러 질의들 가운데에 공유될 수 있는 부분이 많아지므로 제안하는 기법은 더 적은 질의처리 비용을 가지는 질의 실행 계획을 반환한다. 사용되는 입력 스트림의 치우치는 정도가 고르게 분포하는 경우에도 공유되는 조인 연산자가 존재하므로 기존의 MJoin보다 비용이 21% 줄어드는 것을 확인하였다. 질의 실행 계획을 수립하기 위한 시간을 감수하여 기존의 MJoin보다 더 적은 비용으로 질의 처리를 수행할 수 있다.

5. 관련 연구

공유된 조인 연산자를 사용하는 경우 서로 다른 윈도우를 가지는 질의로 적절한 튜플을 라우팅 하는 기법은 [10]에서 제안되었으나 이진 조인을 기반으로 하여 두 개의 입력 스트림만을 고려하기 때문에, 두 개 이상의 스트림을 입력으로 가지는 MJoin에 대해서 작동하지 않는다. 또한 튜플을 여러 질의로 스케줄링하기 위한 계산 비용이 비교적 크다는 문제를 가지고 있다.

하나의 조인 질의에 대한 최적화된 질의 실행 계획을 수립하는 기법은 [5]에서 제안되었다. 하지만 이는 데이터 스트림의 특성, 네트워크 대역폭, 시스템의 계산 능력이 변하는 동적이 환경이 아닌 정적인 환경에 국한된 기법으로 계산 비용이 매우 비싸기 때문에 동적인 데이터 스트림 환경에는 부적절하다.

동적인 데이터 스트림 환경에서 다중 조인 질의에 대한 공유된 질의 실행 계획은 [7]에서 제안되었다. 하지만 이진 조인 연산자를 기반으로 수행되며 질의 실행 계획을 수립하는 시간이 전통적인 질의 실행 계획의 수립 기법과 비교하여 매우 떨어지는 수행 시간을 보여준다. [11]에서는 여러 질의를 정확하게 공유하여 처리하는 조건을 제시하였다. 하지만 다중 조인 질의에 대한 공유 질의 실행 계획의 수립 기법을 제시하지 못하는 문제점을 가지고 있다.

6. 결론 및 향후 연구

본 논문에서는 기존의 MJoin을 다중 조인 질의에 적용하는 기법을 제안하였다. 또한 MJoin을 다중 조인 질의로 확장하는 가운데에 파생되는 문제점을 해결하기 위해 간단한 비교 연산만을 요구하는 추가적인 라우팅 방법을 제안하였다. 마지막으로 다중 조인 질의 계획을 구성할 때 공유 가능한 연산자의 조건을 공식화 하였다. 제안하는 기법은 다중 조인 질의가 등록되어 있는 환경에서 기존의 MJoin을 각각의 질의에 적용하는 방법보다 더 적은 계산비용과 저장 공간이 요구됨을 분석하였다.

추후 연구로써 데이터 스트림 관리 시스템에 새로운 질의가 등록되거나 기존의 질의가 삭제될 경우 빠르게 새로운 질의 계획을 수립하는 기법을 연구할 예정이며, 제안하는 기법을 다각도로 평가할 시스템을 구현 중이다.

참고문헌

[1] Sven Schmidt, Marc Fiedler, Wolfgang Lehner, "*Source-aware Join Strategies of Sensor Data Streams*", In Proceedings of the 17th International Conference on Scientific and Statistical Database Management, pp. 123-132, 2005.

[2] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom, "*Models and Issues in Data Stream Systems*", In Proceedings of PODS, pp. 1-16, 2002.

[3] Anant Jhingran, Sriram Padmanabhan, Ambuj Shatdal, "*Join Query Optimization in Parallel Database Systems*", In Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems, 1993.

[4] Stratis D. Viglas, Jeffrey F. Naughton, Josef Burger, "*Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources*", In Proceedings of the 29th VLDB Conference, pp. 285-296, 2003.

[5] Ahmed M. Ayad, Jeffrey F. Naughton, "*Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams*", In Proceedings of the 2004 ACM SIGMOD, pp. 419-430, 2004.

[6] Stratis D. Viglas, Jeffrey F. Naughton, "*Rate-Based Query Optimization for Streaming Information Sources*", In Proceedings of the 2002 ACM SIGMOD, pp. 37-48, 2002.

[7] Jianjun Chen, David J. DeWitt, "*Dynamic Re-grouping of Continuous Queries*", In Proceedings of the 28th VLDB Conference, pp. 430-441, 2002.

[8] Michael R. Garey and David S. Johnson, "*Computers and Intractability : A Guide to the Theory of NP-Completeness*", W. H. Freeman & Co., pp.221-222, 1979.

[9] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, "*Operator scheduling in data stream systems*", VLDB Journal Special Issue on Data Stream Processing, 2004.

[10] M. Hammad, M. Franklin, W. Aref, and A. Elmagarmid. "*Scheduling for shared window joins over data streams*", In Proceedings of the 29th VLDB Conference, pp. 297-308, 2003.

[11] Sailesh Krishnamurthy, Michael J. Franklin, Joseph M. Hellerstein, Garrett Jacobson, "*The Case for Precision Sharing*", In Proceedings of the 30th VLDB Conference, pp. 972-986, 2004.