스트림 환경에서 다중 조인 속성을 위한 멀티웨이 조인 처리기법

백주현* 정성원*
*서강대학교 컴퓨터공학과
e-mail : jhbaek7@mclab.sogang.ac.kr

A Multi-way joins technique for multi join attributes in Stream Environments

Joohyun Baek* Sungwon Jung*
*Dept. of Computer Science and Engineering, Sogang University

요 약

스트리밍 환경에서 조인 연산은 기존의 기법과는 다른 처리 방법을 요구한다. 이런 문제를 해결 하기 위해 기존에 여러 가지의 다양한 기법들이 제안되었다. 하지만 지금까지 제안된 방법들은 두 개의 입력 스트림에 대한 조인만 고려하거나 단일 속성 멀티 스트림 조인에 대해서만 고려해왔다. 하지만 조인 속성이 여러개인 경우에는 한단계로 조인을 수행하는 것은 불가능하다. 이 눈문에서는 이러한 문제를 해결하기 위해서 지금까지 고려되어 왔던 환경에서 더 일반화 된 다중속성을 가지는 조인을 고려한다. 이러한 경우에는 조인이 다단계로 일어나게 되는데 이러한 환경에서는 이전 단계의 조인이 다음 단계의 조인에 영향을 미치게 된다. 그러므로 최종 조인 결과를 빠르게 만들어 내기 위해서는 여러 입력 스트림 중에서 어떤 조인을 먼저 수행하느냐에 따라 전체적인 조인 결과를 만들어내는 속도가 달라지게 된다. 그러므로 전체 조인결과를 빠르게 만들어 내기 위해서 조인이 수행되는 과정에서 여러 입력 스트림중에 어떤 스트림을 먼저 수행할지를 결정함으로써 최종 조인 결과를 빠르게 만들어낼 수 있게 하는 방법을 제안한다.

1. 서론

최근에 스트리밍 환경에서의 질의와 이에 대한 처리 기법들이 많은 관심을 받고 있다. 스트리밍 환경에서는 정해진 데이터 셋을 가지고 처리를 하는 것이 아니라 계속적으로 발생해서 들어오는 데이터를 가지고 처리를 해야 한다. 이러한 환경에서의 문제를 다루기 위해 기존의 relational DBMS 기반이 아닌 DSMS(Data Stream Management System) 개념이 도입되고 있다. DSMS 는 DBMS 와 달리 지속적으로 발생해서 전달되는 스트리밍 데이터에 대한 처리를 위한 다양한 기법들을 도입하고 있다. 이 중에서 조인과 같이 복잡한 연산의 경우, 기존의 방법을 적용할 수 없는 문제를 가지고 있다. 스트리밍 환경에서 조인 연산을 수행하기 위해서는 non-blocking 특성을 가지는 조인 연산 방법이 필요하다. 또한 데이터의 크기가 계속적으로 증가하게 되므로 조인할 데이터가 메모리에 모두 저장될 수 없는 상황에 대한 대처 방안이 필요하게 된다. 이러한 문제를 해결하기 위해 기존에 여러 가지 연구들이 진행되어 왔다. 기존의 연구들은 데이터 셋을 전부 가지지 않은 상태에서도 빠르게 조인 결과를 만들어 줄 수 있는 방법들에 대한 연구가 진행 되었다. 이러한 방법으로 hash join 에 기반한 여러 방법들[1,2,3]이 제안되었다. 또한 이러한 방법들로부터 여러 가지 문제점들을 개선하기 위해서,Hash Merge Join[4], X Join[5] 과 같은 방법들이 제안되었다. X Join 은 하이브리드 해쉬 조인 방법을 기반으로 해서 조인 단계를 세 단계로 구성하여 가능한 한 빨리 결과를 출력할 수 있게 하고, 데이터를 일정한 속도로 전달받지 못하더라도 조인을 계속 수행할 수 있게 한 방법이다. 또한 메모리가 가득찰 경우 디스크와 같은 이차 저장장치를 이용해서 데이터를 저장해 두고 메모리를 비워서 다시 조인을 수행하는 방법으로 동작한다.

실제 스트리밍 환경에서 사용되는 응용은 여러 개의 소스에서 들어오는 데이터에 대한 처리를 해야 하는 경우가 많이 존재 한다. 이를 two-way join 을 반복해서 적용하는 방법은 여러 가지 비효율적인 문제를 가지고 있다. 이를 보완하는 방법으로 M join 과 같은 방법이 제안되었다. M Join 은 X Join 에서는 두 개의 입력 스트림에 대한 조인연산만을 고려하는 문제를 개선하여 여러 입력 스트림에 대한 조인연산을 한꺼번에 수행할 수 있도록 하는 방법으로 제안되었다. M Join 에서는 조인 트리를 하나의 레벨로 이루어진 것으로 본다. M Join 은 성능 개선을 위해 조인 수행시 조인 순서를 결정하게 함으로써 불필요한 연산을 줄이도록 한다. 그리고 메모리가 가득찬 경우에 multiway-join 의 특성을 이용하여 효율적으로 메모

리를 비우는 방법을 제안하였다.

M Join 에서는 조인 트리를 모든 소스가 한번에 묶이는 형태의 트리 구조로 생각한다. 즉, 한 단계로 모든 소스에 대한 조인을 수행한다. 이러한 방법으로 조인을 수행하려면 조인 속성이 하나로 통일될 수 있다는 가정이 있어야 한다. 그렇지 않을 경우에는 효율적이지 못하게 된다. 실제 스트리밍 데이터에 대한 조인연산들은 여러 소스에 대한 조인연산을 한번에 묶어서 수행하는 것보다는 여러 단계로 수행해야 하는 경우도 많이 발생하게 된다.

표 1 - 여러 조인 속성을 가지는 조인 질의의 예

```
Select   Rstream(T.id as catID, P.price as price)
    From  ClosedAuction [Now] C, CurrentPrice P,
          Item I, Category T
    Where    C.itemID = P.itemID and C.itemID =
             I.id and I.categoryID = T.id
```

여러 소스를 조인하는 과정에서 한번에 조인을 묶어서 수행하려면 조인 속성이 한가지로 통일되어야 수행이 가능하다. 표 1 은 조인속성이 두개 이상이 되는 조인 질의의 예를 보여주고 있다. 이러한 질의는 실제 환경에서 빈번하게 발생한다.

다중 조인 속성을 가지는 경우 외에도 질의에 따라서는 다단계로 수행하는 것이 효율적인 경우가 많다. 이러한 것은 질의를 수행하는 과정을 정하는 쿼리 optimizer 가 선택하게 되는데 조인 수행을 한 단계로만 수행하는 것은 많은 제한을 가져오게 된다. 즉, 여러 입력 스트림에 대한 조인을 수행하는 경우에 있어서 M Join 에서와 같이 간단하게 한 단계로 수행하지 않고 여러 단계로 나누어 수행하게 되는 경우에 대해서 조인을 효과적으로 수행하기 위한 방법이 기존의 조인 기법에서는 충분히 제안되지 못했다. 여러 단계로 조인을 수행하게 되면 각 단계별로 수행하는 조인이 전체 조인에 미치는 영향이 각각 달라지게 된다. 따라서 조인을 수행하게 되는 순서를 결정해야 하는 문제가 발생하게 된다. 여러 스트림으로부터 데이터 들어올 때 어떤 데이터를 먼저 조인을 수행하느냐에 따라서 조인결과를 만들어내는 속도가 달라지게 된다. 그러므로 조인을 수행하는 순서를 고려함으로써 최종 조인 결과를 빠르게 만들어 낼 수 있게 하는 방법을 제안할 것이다.

2. 제안 방법

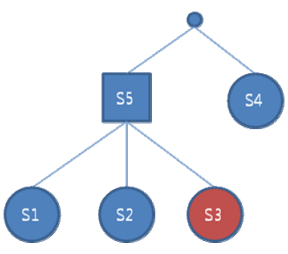

그림 1 - 다단계로 수행되는 조인 트리의 예

여기에서는 그림 1 과 같이 여러 단계로 조인을 수행하는 경우를 고려한다. 그림 1 의 S1~S5 는 각각 하나의 입력 스트림이다. S3 은 특별히 다음 단계에서 조인이 수행될 속성값을 가지고 있는 스트림이다. 기본적인 조인수행방법은 M Join 과 같은 해쉬 조인 방법을 사용한다. 조인할 스트림 데이터가 들어오면 M Join 에서와 같이 해당 서브트리에서 조인을 수행하고 다음 단계의 입력으로 결과를 넘긴다.

여러 튜플이 입력 스트림에 동시 다발적으로 들어오는 경우 여러 개의 튜플이 조인 수행이 되기를 기다리고 있게 된다. 이때, 대기중인 튜플들중에 어떤 스트림에서 들어온 튜플을 먼저 조인할 것인지를 결정해서 순서를 정해서 조인을 수행하도록 한다. 이를 위해서 조인을 할 수 있는 튜플이 여러 개인 경우에 어떤 것을 먼저 수행하는 것이 전체 수행과정에 있어서 빠른 결과를 만들어내는지 판단하기 위해서 조인을 수행한 결과의 개수와 그 결과가 얼마나 다음 단계에서 쓰일지를 추정해야 한다. 그림 1 에서 S4, S5 의 조인은 하위 레벨의 조인 결과가 S5 쪽으로 들어와야 조인이 가능하게 된다. 이 때에 하위 레벨에서의 조인이 만들어낸 결과가 상위 레벨에서 조인이 가능한 것이어야 계속적으로 조인 연산이 상위레벨로 이어질 수 있다. 조인이 성공적으로 수행되면 하나 이상의 조인 결과가 발생된다. 이 결과들이 상위레벨로 전달되었을 때 연속적으로 조인을 수행할 수 있어야 효율적으로 수행을 지속해 나갈 수 있다. 만약 조인 결과가 다음단계에 전해졌지만 다음단계에서 조인이 되지 않는다면 메모리의 부담을 가중시킬 뿐 효율적으로 조인을 수행할 수 없게 만든다. 또한 상황에 따라서 하위 레벨의 조인을 수행하는 것보다 상위 레벨에서 조인을 먼저 수행하는 것이 결과를 빠르게 만들어 낼 수 있다. 상위 레벨의 경우 최종 조인 결과에 근접해 있는 것들이기 때문에 하위 레벨에서의 조인이 상위레벨에서 결과를 만들어내는데 효과적으로 이용되지 못한다면 상위레벨을 먼저 수행하는 것이 메모리의 부담을 줄이면서 결과를 빨리 만들어내는 데에 유리하다.

이러한 점에 착안해서 조인을 수행했을 때 그 결과가 효과적인지를 판단하기 위해 유효조인결과개수를 추정하여 그 값이 가장 큰 것을 먼저 수행하게 한다. 유효조인결과개수는 아래 표 2 와 같이 계산한다.

표 2 - 유효조인결과개수 계산 방법

깊이가 1 인 경우
유효조인결과개수= 조인 결과 개수^2

깊이가 2 이상인 경우
유효조인결과개수 = 조인 결과 개수 * 다음 단계 조인 추정개수

조인 결과 개수는 어떤 튜플이 조인되었을 때 만들어 낼 수 있는 결과의 개수를 의미한다. 조인 결과의 개수는 조인을 위해 비교할 소스들의 해쉬 테이블의 해쉬 버켓에 몇 개의 튜플이 있는지를 보고 추정할 수 있다. 결과개수는 트리 깊이에 따라서 다르게 계

산한다. 깊이가 1 인 경우 결과 개수를 추정하여 제곱한 값으로 정한다. 깊이가 2 이상인 경우 결과가 다음 단계에서 얼마나 쓰일지를 추정하여 곱해준다. 만약 S1 의 튜플로 해쉬 함수 값이 2 인 튜플이 들어왔을 경우의 조인 결과 개수는 S2, S3 의 2 번째 해쉬 버켓의 크기의 곱으로 구할 수 있다.

다음 단계에 얼마나 쓰일 것인지를 판단하기 위해서는 조인 결과가 만들어졌을때 다음 단계의 조인 속성값으로 어떤 값을 가지는지 알아야 할 필요가 있다. S3 와 같이 자기 스스로 다음 단계 조인 속성값을 가지고 있는 경우에는 문제가 되지 않지만 S1, S2 는 실제 조인을 수행해 보아야만 알 수 있다. 이를 해결하기 위해 linked list 를 미리 구성해두는 방법을 이용한다. Linked list 에는 S3 의 해쉬 테이블에 저장된 튜플들이 가졌던 다음 단계 조인 속성값이 상위 레벨의 해쉬 함수값으로 어떤 값을 가지는지를 저장해 둠으로써 S1, S2 가 다음 단계에서 얼마나 쓰일지 쉽게 알아낼 수 있게 한다.

다음 단계 조인 추정 개수는 이렇게 구성해 둔 linked list 를 이용해서 구한다. 만약 S1 의 입력 튜플이 해쉬 함수 값으로 1 을 가지는 경우 1 번째 linked list 에 저장된 값들을 참고한다. 이 값들을 이용하면 다음 단계에서 조인 결과가 몇번째 해쉬 버켓에 있는 튜플들과 조인이 될지 알 수 있다. 다음 단계의 해당 해쉬 버켓의 크기를 보면 몇 개의 튜플들과 조인이 될 지 알 수 있고 이중 최대값으로 다음 단계 조인 추정개수를 정한다. 깊이가 1 인 경우에는 결과가 바로 최종 조인 결과가 되므로 자기 자신을 제곱한 값을 사용한다.

전체 수행과정의 예를 들기 위해서 그림 1 에서 S1~S3 의 해쉬 함수를 mod 4, S4 와 S5 의 해쉬 함수를 mod 5 라고 가정하자. 우선순위 결정을 위해서 각 조인 서브 트리마다 하나의 linked list 를 구성하여 유지한다. Linked list 는 다음 단계에서 조인 결과가 얼마나 조인될 것인지를 간단하게 추정하기 위해서 사용된다.

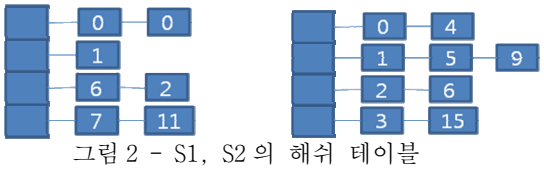

그림 2 - S1, S2 의 해쉬 테이블

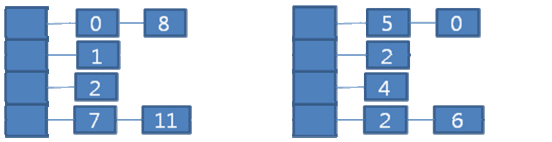

그림 3 - S3 의 해쉬 테이블과 다음 단계 조인 속성값

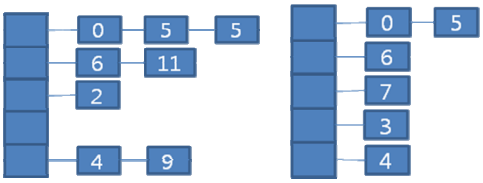

그림 4 - S4, S5 의 해쉬 테이블

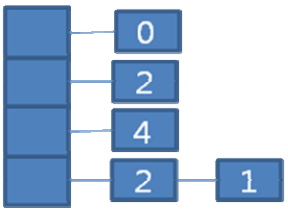

그림 5 - S3 에 의해 유지되는 Linked List

위의 그림 2~4 는 그림 1 에서의 각 node 의 해쉬 테이블의 예를 보여주는 것이다. S1~S3 의 해쉬 함수는 mod 4 라고 가정하고 S4~5 의 해쉬 함수는 mod 5 라고 가정한다. 그림 5 는 이 예에서 유지하는 linked list 의 예를 보여주고 있다.

Linked list 의 개수는 해쉬 버켓의 개수와 같게 한다. Linked list 로 유지되는 값은 다음 단계에 쓰일 조인 속성값이 가질 다음 level 의 hash index 이다. 예를 들면, S3 로 새로 들어온 tuple t1 이 다음 단계에서 조인될 속성값으로 'X' 라는 속성을 가진다고 할 때 'X' 가 다음 단계에서 해쉬 함수에 의해 가지는 값이 2 라면 2 를 list 에 저장하게 된다. 저장하는 위치는 t1 이 S3 의 첫번째 해쉬 버켓에 저장된다면 linked list 에서도 같은 순서의 list 에 저장한다. 즉, 위의 그림 5 에서 보면 S3 의 첫번째 해쉬 버켓에 저장된 두 튜플은 다음 단계 속성값으로 5, 0 을 가지는데 이 값들은 다음 단계의 해쉬 함수값으로 0 로 가지게 된다. 이 값을 첫번째 list 에 저장하게 된다. 만약 값이 중복된다면 한번만 저장한다.

표 3 - 입력 스트림 예

S1	S2	S3	S4	S5
1	-	-	-	-
-	3	2/3	2	-
-	-	-	-	5

표 3 은 그림 1 에서 나타낸 상황에서의 입력 스트림의 예이다. 표의 값은 조인 속성 값을 나타낸다. S3 의 경우 / 뒤의 값은 다음 단계 조인 속성값을 나타낸다. 튜플이 들어온 순서는 표의 위에서부터 순서대로 들어왔다고 가정한다. 메모리에 그림 2~5 와 같이 저장되어 있다고 할 때 위의 표 3 과 같이 입력 스트림으로 들어왔다고 가정하면, 각 소스의 기대 효과값을 계산하면 다음과 같다.

표 4 - 각 소스의 기대 효과 값

S1	S2	S3	S4	S5
3	4	4	1	9

S1 은 S2 와 S3 의 1 번째 해쉬 버켓에 튜플이 3 개와 1 개가 있으므로 결과 개수 값은 3 X 1 = 3 이 된다. 다음 단계 조인 추정개수는 linked list 의 1 번째 list 를 보면 2 이므로 S5 의 2 번째 해쉬 버켓의 크기를 본다. 이 값이 1 이므로 기대 효과는 3 X 1 X 1 = 3 이 된다. S5 의 경우 결과 개수 값이 3 이고 이를 제곱한 9 가 된다. 이 경우 S5 를 선택해서 먼저

조인을 수행하게 된다. 만약 튜플이 들어온 순서대로 수행한다면 S1 을 먼저 수행하게 된다. 이 경우 S5 를 먼저 수행하는 것이 훨씬 효율적이라고 볼 수 있다.

3. 결론 및 향후 과제

이 논문에서는 조인속성이 여러가지인 경우의 조인에 대해서 조인 수행의 순서를 결정하는 방법을 도입함으로써 전체 조인 수행과정을 효율적으로 진행할 수 있도록 하는 방법을 제안하였다. 이 방법을 이용해서 여러 단계로 진행되어야 하는 조인과정에서 초기 결과값을 빠르게 내어줄 수 있을 뿐 아니라 지속적으로 전체적인 효율성을 고려하여 조인을 수행하기 때문에 더 많은 결과를 내어 줄 수 있다. 이 방법에서는 현재 메모리에 저장된 데이터만을 고려하여 조인순서를 결정하였지만 미래에 들어올 데이터 패턴을 고려하여 더 효율적으로 추정할 수 있는 방법에 대해서 추가적으로 연구할 필요가 있다.

조인을 여러 단계로 수행함에 있어서 조인 결과를 정하는것과 더불어 중간 결과 값을 저장하게 되는 문제가 발생한다. 이로 인한 메모리 오버헤드를 줄이는 방법에 대해서 향후 추가로 연구할 필요가 있다. 그리고 여러 단계로 조인을 수행하게 되면 한 단계로의 조인에서와는 달리 데이터가 스왑되는 것이 조인 연산에 미치는 영향이 달라지게 된다. 이런 경우에 어떤 데이터를 선택하여 스왑할 것인지를 결정해야 하는 문제를 향후 추가로 연구할 필요가 있다.

참고문헌

[1] D. DeWitt, R.Katz, F. Olken, L. Shapiro, M. Stonebraker, and D. Wood. Implementation Techniques for Main Memory Database Systems. In SIGMOD, pages 1-8, 1984.

[2] Z. G. Ives, D. Florescu, M. Friedman, A. Y. Levy, and D. S. Weld. An Adaptive Query Execution System for Data Integration. In SIGMOD 1999 pages 299-310, 1999.

[3] G. Luo, C. J. Ellmann, P. J. Hass, and J. F. Naughton. A scalable hash ripple join algorithm. In SIGMOD, pages 252-262, 2002

[4] M. Mokbel, M. Lu, and W. Aref. Hash-Merge Join : A Non blocking Join Algorithm for Producing Fast and Early Join Results. In ICDE 2004, pages 251-263, 2004.

[5] T. Urhan and M. Franklin. XJoin : A Reactively Scheduled Pipelined Join Operator. IEEE Data Engineering Bulletin, 23(2):7-18, 2000.

[6] S. Viglas, J. Naughton, and J. Burger. Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources. In VLDB 2003, pages 285-296, 2003.