韓國 컴퓨터情報學會 2008 제38차 하계학술발표논문집
第16卷 第1號, 2008. 06.

대용량 데이터 스트림을 처리하기 위한 효율적 이진 조인 처리 기법

박 홍 규*, 이 원 석*

Efficient Binary Join Processing for Large Data Streams

Hong Kyu Park*, Won-Suk Lee*

요 약

최근에 제한된 데이터 셋보다 센서 데이터 처리, 웹 서버 로그나 전화 기록과 같은 다양한 트랜잭션 로그 분석 등과 관련된 대용량 데이터 스트림을 실시간으로 처리하는 것에 많은 관심이 집중되고 있으며, 특히 데이터 스트림의 조인 처리에 대한 관심이 증가하고 있다. 본 논문에서는 조인 연산을 빠르게 처리하기 위한 효율적인 해시 구조와 조인 방법에 대해서 연구하고 다양한 환경에서 제안 방법을 검증한다.

▸ Keyword : 조인 처리(Join Processing), 대용량 데이터(Large Data), 데이터 스트림(Data Stream)

• 제1저자 : 박홍규
* 연세대학교 컴퓨터과학과

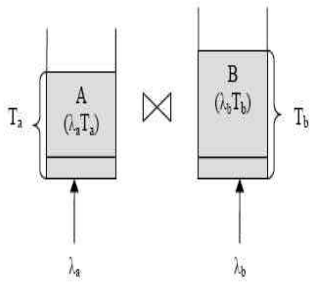
Ⅰ. 서 론

최근에는 제한된 데이터 셋을 처리하는 것이 아닌 대용량 데이터 스트림을 실시간으로 처리하는 것에 많은 관심이 집중되고 있으며, 이는 센서 데이터 처리, 인터넷 트래픽 분석, 주식 거래, 웹 서버 로그나 전화 레코드와 같은 다양한 트랜잭션 로그 분석 등 다양한 분야에서 응용되고 있다.[1,8,9]

데이터 스트림의 처리는 데이터를 모두 저장해놓은 상태에서 한 번 질의를 수행하여 결과를 돌려주는 기존의 방식과는 달리 질의가 미리 등록을 되면 새로운 데이터가 입력될 때마다 즉시 혹은 주기적으로 등록된 질의를 수행하는 연속 질의 형태를 띠게 된다. 이러한 연속 질의는 빠른 속도로 입력되는 스트림을 계속적으로 처리해주어야 하기 때문에 최대한 주어진 시간 내에 많은 스트림을 처리하기 위한 효율적인 데이터 구조가 필수적이다. 본 논문에서는 데이터 스트림 환경에서 조인 연산을 처리하기 위한 방법으로 대부분 사용되고 있는 대칭 해시 조인(Symmetric Hash Join)에 대해 알아보고, 기존에 사용되던 해시 구조에 각 버켓당 헤더 정보를 추가한 HA해시 구조((Header-Added Hash Structure)를 이용하여 보다 빠른 조인 연산이 가능하도록 한다.

Ⅱ. 기존 조인 연산 처리 방법

데이터 스트림에서의 조인 처리는 유한한 메모리 내에서 무한으로 들어오는 스트림을 처리하기 위해서 윈도우 개념을 사용한다. 예를 들어 스트림 R과 S를조인하는데 스트림 A에 들어오는 데이터들 중 최근 t1시간에 들어온 튜플들 만을 저장하고 스트림 B에서 들어오는 스트림도 마찬가지로 최근 t2시간에 들어온 튜플들만을 저장하며, 저장된 튜플들만을 대상으로 조인을 수행하게 된다. 이 조인은 새로운 입력 튜플이 들어올 때마다 새로운 결과를 생성하면서 연속적으로 질의를 수행하게 된다. 즉 A에서 새로운 데이터가 들어오면 B의 유효한 스트림 데이터와 조인을 하여야 하며 B에서 새로운 데이터가 들어오면 A의 유효한 스트림 데이터와 조인을 하여야 한다. 이와 같은 형태의 조인 방법을 대칭 조인(Symmetric Join)이라고 하며, 해시 인덱스를 사용한 방법을 SHJ(Symmetric Hash Join)이라고 한다. 대부분 사용되는 해시 인덱스는 버켓을 사용하며, 버켓 오버플로우 시 오버플로우 버켓을 연결리스트로 연결하여 오버플로우를 처리한다. 그림 1은 윈도우 조인을 그림으로 표현한 것이다.



<그림 1> 윈도우 조인

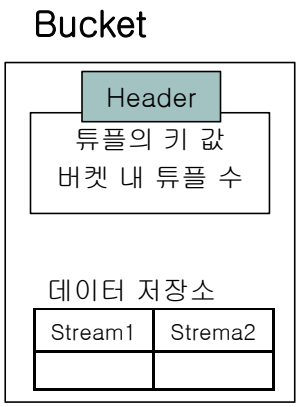
Ⅲ. HA 해시 구조를 이용한 조인 처리

새로운 입력 튜플이 들어왔을 때 조인 연산을 수행하기 위한 연산은 크게 3가지로 나누어 볼 수 있다. 즉 새로운 입력 튜플을 삽입하는 연산(삽입 연산), 정의된 윈도우를 벗어난 유효하지 못한 데이터를 삭제하는 연산(삭제 연산), 그리고 조인 결과(조사 연산)를 생성하기 위해서 상대 스트림을 스캔하는 연산으로 나눌 수 있다.[2] 이 3가지의 연산 중에 상대적으로 많은 접근이 필요하기 때문에 조사 연산이 가장 큰 부하를 가지게 된다. 그러므로 본 논문에서는 조사 연산의 부하를 줄일 수 있는 해시 구조를 HA 해시 구조를 제안하고, 이를 이용한 조인 처리 기법을 제안한다.

3.1. HA(Header-Added) 해시 구조

일반적으로 해시는 원하는 데이터에 접근하기 위해 O(1)의 탐색이 필요하지만 실제적으로는 다른 키 값을 가진 데이터들이 하나의 버켓에 저장되어 있는 경우가 많다. 이러한 현상은 불필요한 데이터들까지 접근하는 원인이 된다. 본 논문에서는 각 버켓에 들어온 데이터들에 대한 키 값을 헤더에 저장하고 비록 해시 함수를 통해 같은 버켓에 들어온 튜플이라 할지라도 다른 버켓에 저장함으로써 보다 효율적으로 조인을 처리할 수 있는 HA 해시 구조를 제안한다.

각 버켓들은 각각의 헤더를 가지고 있으며, 헤더에는 저장된 튜플의 키 값(조인 속성의 값)과 각 버켓 내에 몇개의 튜플이 저장되어 있는지를 저장한다. 튜플의 키 값은 조인 속성의 값이 다른 튜플들을 검사하는 것을 방지하여 조사 연산의 효율을 높인다. 데이터 저장소는 헤더에 있는 튜플의 키 값을 가진 튜플들만 저장되는 공간으로써 일반적인 버켓의 형태와 동일한 역할을 하는 곳이며 동일한 조인 속성으로 조인이 되는 경우 그림 2와 같이 버켓을 스트림의 갯수 만큼 가지고 있는다.



<그림 2> HA 해시의 버켓 구조

3.2. HA 해시 구조를 이용한 조인 처리 방법

이 장에서는 조인 연산의 처리는 삽입 연산, 삭제 연산, 조사 연산으로 이루어져 있으며, 각 연산들이 HA 해시 구조를 이용하여 어떻게 수행되는지에 대해 간략히 설명한다.

3.2.1. 삽입 및 삭제 연산

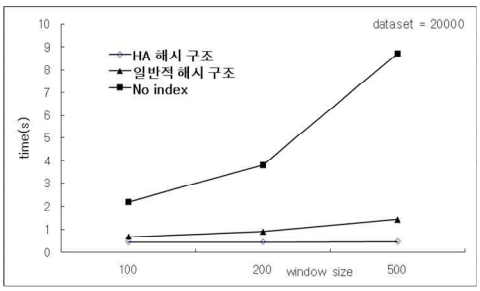
새로운 튜플 t1이 들어왔을 때 삽입 및 삭제 연산이 이루어진다. 우선 t1의 조인 속성의 값을 해싱 함수를 통하여 해싱값을 구하고, 이와 관련된 해시 버켓으로 접근한다. 그리고 나서 해당 버켓의 헤더에 저장되어 있는 튜플의 키 값과 t1의 조인 속성의 값을 비교하여 값이 같다면 해당 버켓에 튜플을 저장하고 버켓 내 튜플수를 하나 증기시킨다. 만약 버켓의 헤더에 있는 튜플의 키 값과 t1의 값이 동일하지 않다면 기존의 해시 구조에서 오버플로우가 생겼을 때 오버플로우 버켓을 생성하듯이 새로운 버켓을 생성하고 헤더를 초기화한 후 t1을 관련된 스트림의 데이터 저장소에 저장한다. 삭제 연산은 두 가지 방법으로 수행될 수 있는데, 활동적 삭제(Active Invalidation) 방법과 풀린 삭제(Loose Invalidation) 방법이 있다.[5] 활동적 삭제 방법은 윈도우의 적용을 엄격하게 하여, 유효한 데이터만을 저장하는 방법으로 매 튜플이 생성될 때마다 삭제 연산이 이루어져야 한다. 풀린 삭제 방법은 윈도우의 적용을 느슨하게 하여 삭제 연산에 대한 부하를 줄이는 대신 유효하지 않은 데이터들을 포함하고 있을 가능성이 있다. 본 연구에서는 최대한 빠른 시간 안에 대용량의 데이터 스트림을 처리하는 것을 목적으로 하고 있기 때문에 풀린 삭제 방법을 사용한다. 즉, 새로운 튜플이 들어올 때마다 삭제 연산을 수행하는 것이 아니라 조사 연산을 수행할 때 같이 삭제 연산을 수행하게 된다. 조사 연산을 수행할 때 상대 스트림에 대한 결과를 모두 조사하는 과정에서 삭제 연산을 수행하기 때문에 삭제 연산에 드는 다른 부하는 하나도 없게 된다.

3.2.2. 조사 연산

새로운 튜플 t1이 들어왔을 때 삽입 연산을 수행한 후에 조인 결과를 생성하기 위하여 조사 연산을 수행한다. 기존의 해시 구조의 경우 t1이 들어오면 t1의 조인 속성을 해싱 함수를 통하여 해싱 값을 알아낸 후에 상대 스트림의 해당 버켓에 가서 모든 튜플들과 속성 값들을 비교하고 같은 값일 경우에만 결과를 생성하여야만 한다. 하지만 HA 해시 구조를 이용하면 속성 값의 비교 없이 상대 스트림의 데이터 저장소에 있는 튜플들과 합쳐서 결과를 내어주는 역할을 한다. 이 것은 하나의 버켓에는 같은 속성 값을 가지는 것을 보장하기에 가능한 것이다. 하지만 이전 절에서 말했듯이 풀린 삭제 방법을 사용하면 윈도우의 범위를 벗어난 유효하지 않은 튜플이 있을 수 있기 때문에 윈도우에 대한 검사는 하여야 한다.

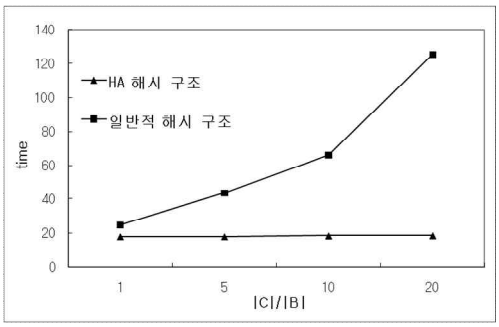
IV. 실험 결과

본 논문에서 제안한 방법의 성능을 검증하기 위하여 인조 데이터셋(Synthetic Dataset)를 생성하여 실험하였다. 스트림의 모든 속성은 숫자 형태로 정의된다. 속성의 값은 각 속성의 카디널리티를 정하고 0부터 카디널리티까지의 숫자 중 랜덤하게 생성하여 인조 데이터셋을 생성하였다. HA 해시 구조를 이용한 조인 처리 방법의 대한 전반적인 성능을 평가하기 위하여 윈도우의 크기를 변화시켜가면서 일반적인 해시 구조를 이용한 조인 처리와 비교를 하였다. 그림 3에서 알 수 있듯이 윈도우의 크기가 커진다는 것은 그 만큼 하나의 버켓에 많은 양의 튜플이 들어감을 의미하기 때문에 윈도우의 크기가 커질수록 제안된 방법이 보다 좋은 성능을 나타냄을 알 수 있다.



<그림 3> 윈도우 크기에 따른 성능 비교

HA 해시 구조를 이용한 조인 처리 방법은 하나의 해시 버켓에 서로 다른 속성 값을 가진 튜플들이 저장되어 있을 경우에 좋은 성능을 기대할 수 있다. 이에 관한 성능 차이를 알아보기 위하여 조인 속성의 카디널리티(|C|)와 해시 테이블의 크기(|B|)에 따른 비교를 하였다. 예를 들어 |C|가 10이고 |B|가 5라면, 버켓의 갯수는 5개이고, 조인 속성은 1~10까지의 값을 가지기 때문에 한 버켓당 2개의 서로 다른 값을 가진 튜플이 들어가게 된다. 만약 |C|가 100이고 |B|가 10이라면 한 버켓당 10개의 서로 다른 값을 가진 튜플이 들어갈 것이다. 그러므로 |C|/|B|의 값이 커질수록 제안된 기법은 기존 기법보다 더 좋은 성능을 나타낸다. 그림 4에서 보듯이 HA 해시 구조를 이용한 조인 처리 방법은 |C|/|B|의 값이 커져도 하나의 버켓에는 같은 속성값을 가진 튜플만 저장되기 때문에 조인 처리 시간이 거의 차이가 없었다. 하지만 서로 다른 속성값을 가진 만큼 오버플로우 버켓으로 생성하여야 하기 때문에 메모리는 더 많이 사용할 것이다.



<그림 4> |C|/|B|에 따른 성능 비교

V. 결론

본 논문에서는 실시간으로 무한히 많은 데이터가 들어오는 데이터 스트림 환경에서 주어진 시간 내에 최대한 많은 데이터를 처리 하기 위한 해시 구조를 제안하고 이를 이용한 윈도우 조인 기법을 제시하였다. 이 방법은 삽입 연산에서는 기존의 일반적 해시 방법에 비해 많은 연산을 하지만, 조사 연산을 최대한 단순화함으로써 계속적으로 조인 결과를 생성해야 하는 연속 조인 질의의 경우에 매우 적합하다. 하지만 기존의 해시 방법에 비해 오버플로우 버켓이 많아질 수 있기 때문에 이에 대한 연구를 향후 더 해야할 것이다.

참고문헌

[1] The STREAM groups STREAM: The Stanford Stream Data Manager (short overview paper) IEEE Data Engineering Bulletin, March 2003

[2] J. Kang, J. F. Naughton, and S. Viglas. Evaluating window joins over unbounded streams. In Proc. of the 2003 Intl. Conf. on Data Engineering, Mar. 2003

[3] Michael Cammert, J¨urgen Kr¨amer, Bernhard Seeger, Sonny Vaupel A Cost model for adaptive Resource Management in data stream systems

[4] A. Krishnamurthy, H. boral, and C. Zaniolo. Optimization of nonrecursive queries. In proc. Of the 1986 Intl. Conf. ln Very Large Data Bases, pages 128-137, Aug. 1986.

[5] Stratis D. Viglas Jeffrey F. Naughton Josef Burger. Maximizing the output rate of multi-way join queries over streaming information sources

[6] Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In VLDB, 2003.

[7] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, J. Widom, Adaptive ordering of pipelined stream filters, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, 2004, pp. 407{418.

[8] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In Proc. Of the 2002 ACM SIGMOD Intl. Conf. on Management of Data, pages 49-60, June 2002.

[9] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In ACM SIGMOD, 2000.

[10] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems Invited paper in Proc. of PODS 2002, June 2002