

중첩 윈도우를 가진 데이터 스트림을 위한 효율적인 조인 알고리즘 (An Efficient Join Algorithm for Data Streams with Overlapping Window)

김 현 규 ^{*} 강 우 람 ^{*}
(Hyeon Gyu Kim) (Woo Lam Kang)

김 명 호 ^{††}
(Myoung Ho Kim)

요 약 일반적으로 중첩 윈도우는 스트림 질의에서 흔히 이용된다. 그럼에도 불구하고, 기존의 연구에서는 텁블링 윈도우나 투플-드리븐 윈도우 등의 기본적인 윈도우만을 가정하고 조인 알고리즘을 다루었다. 본 논문에서는 보다 일반화된 윈도우의 형태인 중첩 윈도우상에서 조인을 효율적으로 처리하기 위한 알고리즘을 제안한다. 제안하는 알고리즘은 기본적으로 점증 조인 후 합병하는 방법을 이용한다. 그리고, 대량의 입력으로부터 메모리 오버플로우가 빈번하게 발생하는 상황에서 연속적으로 윈도우 조인 결과를 생성하는 방법에 중점을 두었다. 제안하는 방법은 (1) 점증 조인과 중복을 허용한 완전 조인을 선택적으로 이용하는 방법, (2) 조인 결과의 지연을 최소화하기 위한 교체 대상 선정 방법과 (3) 가용 시간 처리 방법 등을 포함한다. 그리고, 실험을 통해 점증 조인과 완전 조인을 선택적으로 이용하는 것이 하나만 이용하는 기존 방식에 비해 성능이 우수함을 보인다.

* 이 논문은 2007년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. R0A-2007-000-10046-0)

† 이 논문은 제35회 추계학술대회에서 '중첩 윈도우를 기반한 데이터 스트림을 효율적으로 처리하기 위한 조인 알고리즘'의 제목으로 발표된 논문을 확장한 것임

^{*} 학생회원 : KAIST 전산학과
hgkim@dbserver.kaist.ac.kr
wlkang@dbserver.kaist.ac.kr

^{††} 종신회원 : KAIST 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2009년 1월 15일
심사완료 : 2009년 3월 17일

Copyright©2009 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제5호(2009.5)

키워드 : 데이터 스트림, 하이브리드 조인, 중첩 윈도우

Abstract Overlapping windows are generally used for queries to process continuous data streams. Nevertheless, existing approaches discussed join algorithms only for basic types of windows such as tumbling windows and tuple-driven windows. In this paper, we propose an efficient join algorithm for overlapping windows, which are considered as a more general type of windows. The proposed algorithm is based on an incremental window join. It focuses on producing join results continuously when the memory overflow frequently occurs. It consists of (1) a method to use both of the incremental and full joins selectively, (2) a victim selection algorithm to minimize latency of join processing and (3) an idle time processing algorithm. We show through our experiments that the selective use of incremental and full joins provides better performance than using one of them only.

Key words : Data streams, Hybrid join, Overlapping windows

1. 서 론

연속 데이터 스트림의 처리는 유비쿼터스 컴퓨팅 환경의 구현에 있어서 필수적인 요소이다. 이는 주로 위험 이벤트의 감지나 어떤 값의 변경 추이를 실시간으로 관찰하고자 할 때 이용된다[1]. 예를 들어, 망 관리 시스템 (Network management system)에서 각 경로의 패킷 지연 값을 관찰하여 로드 밸런싱(Load balancing)에 이용하거나, 각 패킷의 목적 주소를 체크하여 DDoS 공격과 같은 위험 이벤트를 감지하는데 이용될 수 있다[2].

기존의 데이터베이스와 동일하게, 스트림에 있어서도 조인은 중요한 역할을 차지한다[3]. 위 로드 밸런싱 예에서 세 개의 라우터 A, B, C로 구성된 하나의 경로를 가정해 보자. 해당 경로의 지연 값을 계산하는데 있어 경로를 지나는 패킷을 식별하기 위해, 아래와 같은 질의를 이용할 수 있다. 아래 질의에서 각 패킷은 자신의 아이디를 나타내는 pid를 지닌다고 가정한다.

Q1. SELECT *

```
FROM A [RANGE 4 mins, SLIDE 1 min]
      B [RANGE 4 mins, SLIDE 1 min]
      C [RANGE 4 mins, SLIDE 1 min]
WHERE A.pid = B.pid AND B.pid = C.pid
```

위 예에서 볼 수 있듯이, 중첩 윈도우(Overlapping window)는 스트림 질의에서 흔히 이용된다[4]. 그러나 MJoin[5], PJoin[6], Golab et al.[7] 등 대부분의 기존 조인 방법은 윈도우 간의 중첩이 없는(RANGE 값과 SLIDE 값이 같은) 텁블링 윈도우(Tumbling window)나 투플이 들어올 때마다 윈도우를 갱신하는 (RANGE

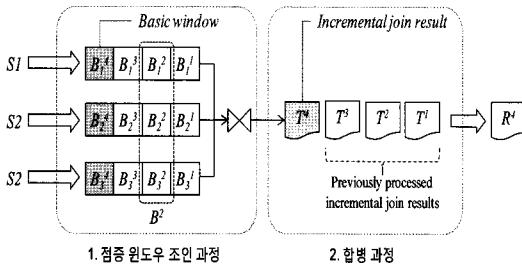


그림 1 중첩 윈도우를 처리하기 위한 점증 조인 방법

n time, SLIDE 1 tuple로 정의되는) 형태[1]의 기본적인 윈도우를 가정한 후 조인 문제를 다루었다.

본 논문에서는 보다 일반화된 윈도우의 형태인 중첩 윈도우를 가정하고, 이를 효율적으로 처리하기 위한 조인 알고리즘을 소개한다. 중첩 윈도우는 기본적으로 중복 처리를 내재한다. 위 질의 예에서는 각 스트림마다 매 3분 길이의 입력에 대해 조인을 중복으로 처리하게 된다. 이러한 중복 처리를 피하기 위해, 가장 쉽게 생각할 수 있는 방법은 점증 윈도우 조인(Incremental window join)을 수행한 후 이전에 생성된 여러 결과를 합병(Merge)하는 방법이다(그림 1). 먼저 점증 윈도우 조인 과정에서는 새로운 1분 길이의 입력에 대해서만 조인을 수행한다. 다음으로 합병 과정에서는 새로운 결과와 이전에 생성된 3개의 결과를 합쳐서 조인 결과를 생성한다. 이와 같은 방법을 간단히 점증 조인이라 부른다.

점증 조인을 처리하기 위해서는 하나의 윈도우를 중첩되지 않는, 기본 윈도우(Basic window)[9]라 불리는 소단위로 구분해야 한다. 즉 처리의 단위가 하나의 윈도우가 아닌, 기본 윈도우로 축소된다. 기본 윈도우 단위의 하나의 입력은 여러 조인 출력에 영향을 주게 된다. 예를 들어 그림 1에서 메모리 오버플로우에 의해 B^2 가 디스크로 이동하면 T^2 부터 T^5 까지 생성될 수 없으며, 이는 다수의 조인 결과를 생성할 수 없게 만든다. 중첩 윈도우의 이러한 특성은 기존의 조인 알고리즘에서 다루지 않았던 부분이며, 제안하는 방법에서는 이와 관련하여 조인 결과의 지연을 최소화하기 위한 (1) 교체 대상 선정(Victim selection) 방법과 (2) 가용 시간(Idling time) 처리 방법을 소개한다.

또 하나 주목해야 할 점은 점증 조인이 윈도우의 모든 튜플을 매번 조인하는 방법에 비해 항상 나은 성능을 보장하지는 않는다는 사실이다. 편의상 모든 튜플을 조인하는 방법을 완전 조인(Full join)이라 명명한다. 예를 들어, 이전 조인 결과의 집합 T 를 저장하기 위한 메모리가 부족할 경우, T 를 디스크에 유지할 수 있다. 이 경우 점증 조인을 통해 결과를 출력하기 위해서는, 새로운 점증 조인 결과를 디스크에 쓰고 T 의 일부를 다시 읽어

들이기 위한 디스크 접근 비용이 요구된다. 만약 조인 선택율(Join selectivity)이 크다면 T 의 크기가 커지므로 디스크 접근 비용이 증가하며, 이 비용은 완전 조인을 통해 T 를 생성하는 비용보다 더욱 커질 수 있다.

따라서 제안하는 방식에서는 점증 조인과 완전 조인을 함께 이용한다. 점증 조인과 완전 조인은 다시 메모리 상에서 수행되는 원-패스(One-pass) 조인과 디스크를 함께 이용하는 투-패스(Two-pass) 조인으로 각각 나뉘어지며, 제안하는 알고리즘은 시스템의 메모리 상황에 따라 4가지 조인 방법을 적절히 활용한다.

본 논문에서는 논의의 편의를 위해 시간-기반 슬라이딩 윈도우(Time-based sliding window)[1]에서 동작하는 동등-조인(Equi-join)의 경우에 한해 다룬다. 그리고 윈도우 정의에 있어서 각 스트림의 RANGE 값이 같고, 역시 각 스트림의 SLIDE 값이 모두 같은(예를 들어 Q1과 같은) 경우를 다룬다. 이러한 제약은 중첩 윈도우를 다룰 때, 하나의 윈도우를 기본 윈도우로 쉽게 구분할 수 있게 하여 처리 모델을 단순화시키는 잇점이 있다. 또한 대칭 해시 조인(Symmetric hash join)[5]이 조인의 기본 알고리즘으로 사용된다고 가정한다.

본 논문의 구성은 다음과 같다. 2장에서는 조인 알고리즘에 대해 설명하고 3장에서는 관련 실험 결과를 설명한다. 4장에서는 관련 연구에 대해 간략히 소개하고, 5장에서 결론으로 마무리한다.

2. 조인 알고리즘

이 장에서는 우선 조인 알고리즘의 설명에 앞서 우선 메모리가 충분하지 못할 경우 디스크로 보낼 교체 대상을 선정하는 알고리즘을 소개한 다음, 시스템의 메모리 상황에 따라 두 방법을 함께 이용하는 하이브리드 조인 알고리즘을 소개한다.

2.1 교체 대상 선정 알고리즘

제안하는 조인 알고리즘은 기본적으로 점증 조인을 이용한다. 점증 조인이 동작하기 위해서는 그림 1에서 보듯이, 윈도우 내용을 저장하기 위한 공간과 이전에 생성된 점증 조인 결과를 저장하기 위한 공간이 필요하다. 편의상 전자를 윈도우 버퍼, 후자를 조인 버퍼(Join extent buffer)라 명명한다.

각각의 버퍼에서 어떤 입력을 교체 대상으로 선정할지는 자명하다. 가능한 오래된 입력을 선택해야 자연이 최소화된다. 단지 고려되어야 할 점은 윈도우 버퍼와 조인 버퍼 중 어느 버퍼의 입력을 먼저 교체 대상으로 선택할지 결정하는 문제이다. 즉 윈도우 버퍼의 B^x 와 조인 버퍼의 T^y 중 어느 것을 선택해야 자연이 적어지는지 알아야 한다.

Algorithm 1. Victim selection

```

Input: size  $s$  (a required memory size)

If  $s \leq /T_m/$ 
    Choose victims in  $T^x_m$ 's, not in  $T^y_m$ 's ( $x \leq y$ )
Otherwise,
    Choose  $T_m$  as a part of the victim, and then
        For the remaining part whose size is  $s - /T_m/$ ,
            Choose victims in  $B^x$ 's, not in  $B^y$ 's ( $x \leq y$ )

```

그림 2 교체 대상 선정 알고리즘

이 문제의 해답은 다음의 예를 통해 직관적으로 알 수 있다. R 이 생성되기 위해서는 T 가 필요하고, T 가 생성되기 위해서는 W 가 필요하다는 것을 이용한다. 예를 들어 그림 1에서 조인 베퍼 중 가장 지연을 크게 하는 T^3 이 교체 대상으로 선택된다면, R^2 이 첫 조인 결과로 생성될 수 있다(R^6 을 생성하기 위해서는 T^3, T^4, T^5, T^6 이 필요하므로). 이에 반해, 윈도우 베퍼 중 지연을 가장 작게 하는 B^1 이 선택된다면 이로부터 W^4 까지는 만들어질 수 없으며, 마찬가지로 T^4 역시 만들어질 수 없다. 이는 T^4 가 교체 대상으로 선택된 것과 동일한 효과를 가져오며 이 경우 R^2 까지는 생성되지 못하므로(이는 R^6 을 생성하기 위해서는 T^3, T^4, T^5, T^6 이 필요하기 때문) R^8 이 첫 조인 결과로 생성될 수 있다. 즉 조인 베퍼 중 최악의 입력을 교체 대상으로 선택하더라도 윈도우 베퍼의 최선의 입력을 선택한 것 보다 지연이 적어진다는 것을 알 수 있다. 따라서 윈도우 베퍼에 비해 조인 베퍼에서 교체 대상을 선택하는 것이 항상 효율적이다.

알고리즘 1은 이와 같은 아이디어를 의사 코드로 기술한 형태이다(그림 2). 위 알고리즘에서 T_m 은 T 에서 메모리에 유지되는 부분을 나타내며, T_m^x 과 T_m^y 을 포함한다. 알고리즘 1에 의해 선택된 교체 대상은 교체 이후 생성될 조인 결과의 지연을 최소화한다.

2.2 기본 조인 알고리즘

본 논문에서 제안하는 조인 알고리즘은 기본적으로 점증 조인을 이용한다. 그러나 앞에서도 언급하였듯이, T 를 위한 메모리가 부족할 경우 이를 디스크에 유지하고 사용하기 위한 비용을 고려해야 한다. 만약 이 비용이 완전 조인을 이용해 T 를 생성하는 비용보다 크다면, 점증 조인보다는 완전 조인을 이용하는 것이 더욱 효율적이다. 따라서 두 비용을 추정하여 이를 바탕으로 점증 조인과 완전 조인 중 어느 조인을 이용할 것인지 선택하는 것이 제안하는 알고리즘의 기본 아이디어에 해당한다. 편의상 T 를 디스크에 유지하고 사용하는 비용을 C_{reuse} , 그리고 T 를 완전 조인을 통해 생성하는 비용을 C_{join} 이라 지칭한다. 두 비용은 조인 선택율과 스트림의 입력 비율로부터 추정 가능하다.

Algorithm 2. One-time hybrid join processing

```

Input:  $B^i$  (a set of new basic windows)

If  $/W^i/ + /T^i/ \leq m$  (a total memory size)
    One-pass incremental join, then one-pass merge
Else if  $/W^i/ \leq m \wedge /W^i/ + /T^i/ \leq C_{reuse}$ 
    If  $C_{reuse} < C_{join}$ 
        One-pass incremental join, then two-pass merge
    Otherwise, one-pass full join
Else
    If  $C_{reuse} < C_{join}$ 
        Two-pass incremental join, then two-pass merge
    Otherwise, two-pass full join

```

그림 3 기본 하이브리드 조인 알고리즘

제안하는 알고리즘은 시스템의 메모리 상황에 따라 점증 조인과 완전 조인을 적절히 활용한다. 시스템의 메모리 상황은 (1) T 를 위한 메모리가 충분한 경우, (2) T 를 위한 메모리는 불충분하나 W (모든 입력 윈도우)를 위한 메모리는 충분한 경우, 그리고 (3) W 를 위한 메모리 또한 불충분한 경우로 나눌 수 있다.

우선 첫째 경우는 기존에 설명한 점증 조인의 수행을 통해 조인 결과를 얻을 수 있다. 나머지 경우는 T 를 위한 메모리가 부족한 경우이므로, C_{reuse} 와 C_{join} 을 비교해야 한다. 둘째 경우에서 만약 C_{reuse} 가 적다면 점증 조인을 수행하며, 단 합병에 있어서 T 의 일부분이 디스크에 존재하므로 이를 접근하기 위한 노력이 든다. 반대로 C_{join} 이 적다면 메모리 상에서 완전 조인을 수행하여 조인 결과를 생성한다.

첫째와 둘째 경우는 W 가 메모리에 있으므로 원-패스(점증 혹은 완전) 조인 알고리즘을 이용한다. 그러나 마지막 경우는 W 의 일부분이 디스크에 존재하는 상황으로, 투-패스 조인 알고리즘을 이용해야 한다. 이 경우 만약 C_{reuse} 가 적다면 투-패스 점증 조인을 수행하며, 합병 역시 T 가 디스크에 존재하므로 투-패스 합병 알고리즘이 이용된다. 반대로 C_{join} 이 적다면 투-패스 완전 조인을 수행하여 조인 결과를 생성한다.

알고리즘 2는 이와 같은 아이디어를 의사 코드로 기술한 형태이다(그림 3). 이 알고리즘은 입력 B^i 가 준비되었을 때 메모리 상황을 고려해 어떤 조인 및 병합 알고리즘을 이용할지 판단한다.

2.3 연속 조인 알고리즘

알고리즘 2는 입력 B^i 가 준비되었을 때, 한번의 조인 결과를 생성하기 위한 알고리즘이다. 이는 조인 결과 R^i 가 항상 윈도우 단위($W_1^i \times W_2^i \times \dots \times W_n^i$)로 이루어져도록 보장한다. 이 알고리즘은 연속 입력에 대해 연속으로 조인 결과를 낼 수 있도록 설계되어야 한다. 또한 연속 처리에 있어서 역시 윈도우 단위 조인 결과(Window-basis join results)를 생성하도록 보장해야 한다.

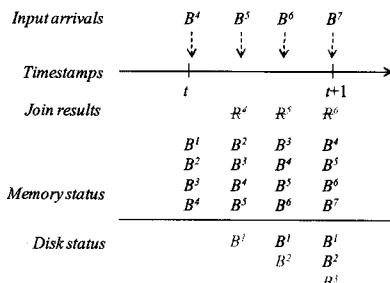


그림 4 이벤트 기반 연속 조인 알고리즘

연속으로 조인 결과를 안정적으로 생성하기 위해서는 다음의 고려 사항을 반영해야 한다.

1. 알고리즘 2의 수행 중 메모리 오버플로우가 일어나는 경우의 처리
2. 입력 전달이 이루어지지 않는 가용 시간(idle time)의 처리

위 사항을 반영하기 위해, 제안하는 알고리즘에서는 이벤트 기반 처리 방식을 이용한다.

먼저 입력이 전달되었을 때는 알고리즘 2를 이용하여 조인 결과를 생성한다. 만약, 조인 결과 생성 중 메모리 오버플로우가 발생한다면, 알고리즘 2의 수행을 잠시 중단한 후 알고리즘 1을 수행하여 교체 대상을 선정한 후 디스크로 보낸다. 만약 교체 대상이 T 에 속한다면 기존 입력이 메모리에 그대로 유지된 상태이므로, 중단한 알고리즘 2의 수행을 재개한다. 이에 반해 교체 대상이 W 에 속한다면 기존 입력이 바뀐 경우에 해당하므로, 중단된 알고리즘 2의 수행을 취소(Abort)한다. 이 때 이전에 출력되던 조인 결과가 불완전함을 알리기 위해 구분자(Delimiter)를 별도로 생성하여 전달한다.

예를 들어 그림 4에서 4개의 기본 윈도우가 하나의 윈도우를 구성하며, T 를 위한 메모리는 없다고 가정하자. t 시간에 B^1 부터 B^4 가 메모리에 준비되었을 경우 조인을 수행하여 R^4 를 생성하기 시작한다. 생성 도중 메모리 오버플로우를 유발하는 B^5 가 전달될 경우, 알고리즘은 B^1 을 디스크로 보내고 구분자(Delimiter)를 출력하여 생성되던 R^4 가 불완전함을 알린다. 그림에서는 취소선(R^i)을 이용해 불완전한 결과를 표시하였다. 마찬가지로 메모리 오버플로우를 유발하는 B^6 과 B^7 이 전달될 경우, B^2 와 B^3 을 디스크로 보내고 생성되던 조인 결과 R^5 와 R^6 에 대한 구분자를 출력한다.

만약 조인 결과의 생성이 성공적으로 이루어지고 새로운 입력이 아직 전달되지 않았다면, 가용 시간 처리(idle time processing) 모드로 전환한다. 해당 모드에서는 메모리 오버플로우에 의해 디스크로 옮겨진 입력들을 메모리로 읽어와 조인을 수행한다. 이 때 디스크 내 가장 최근 입력부터 오래된 입력의 역순으로 읽어와 조인을 수

행한다. 따라서 조인 결과 역시 역순으로 생성된다.

가용 시간 처리 모드에서는 작업을 수행하기 전의 메모리 상태를 유지하면서 조인을 수행한다. 예를 들어, 작업을 수행하기 전 생성된 결과를 R_i 라 하면, 이에 필요한 $N - 1$ 개의 기존 B_x ($i - N - 1 < x \leq i$)는 메모리에 항상 유지한다. 이는 새로운 입력 B_{i+1} 이 전달되었을 때, 새로운 조인 결과 R_{i+1} 를 지연 없이 생성하기 위함이다. 기존 스트림 연구에서 언급하듯이, 스트림 응용에서는 최신의 데이터를 가능한 실시간에 처리하는 것이 중요하다[1]. 따라서 제안하는 방법에서는 가용 시간 처리 모드에서 최신 데이터를 지연 없이 생성하도록 최적화하고자 하였다.

그림 4의 예에서 B_7 이 전달된 후 알고리즘이 R_7 을 성공적으로 생성했다고 가정하자. 이 후 가용 시간 처리 모드에서는 R_6 을 생성하기 위해 디스크로부터 B_3 을 읽어온다. 이 경우 기존의 메모리 내용 중 B_5 부터 B_7 까지는 다음에 생성될 결과 R_8 을 위해 유지한다. 따라서 B_3 을 메모리로 읽어 들이기 위해서는 기존의 B_4 를 디스크로 옮겨야 한다. 즉 가용 시간 처리 모드에서는 투-패스 조인 알고리즘이 사용될 가능성이 높다.

만약 조인 결과의 생성이 성공적으로 이루어지고 난 후 처리해야 할 새로운 입력이 존재한다면, 알고리즘 2를 수행하게 된다. 이 때 주의해야 할 점은, 만약 디스크에 B_i 가 유지된다면 추후에 조인 결과를 생성하기 위해서는 B_i 외에 W_x ($x = i + N - 1$) 속하는 나머지 B_y ($i - i < y \leq x$)도 디스크로 전달되어야 한다는 점이다. 즉, B_i 하나만으로는 윈도우가 성립되지 않으므로 조인 결과를 생성할 수 없으며, 윈도우를 구성하기 위한 나머지 B_y 역시 디스크에 저장해야 추후에 조인 결과의 생성이 가능하다.

그림 4의 예에서 만약 R_7 을 생성한 후 바로 B_8 이 전달되었다면, 가용 시간 처리 모드로 들어가지 않고 다시 알고리즘 2가 호출된다. 이 때 디스크에 있는 내용으로부터 추후에 R_4 부터 R_6 을 생성하기 위해, B_4 부터 B_6 을 디스크에 저장한다.

3. 실험 결과

메모리가 부족하지 않은 일반적인 상황에서는 점증 조인이 완전 조인보다 빠르게 수행된다. 본 논문이 대상으로 하는 메모리가 부족한 환경에서도 이런 결과를 나타내는지 실험을 통해 확인하고자 하였다.

그림 5는 조인 선택율을 0.05로 고정시켰을 때 베이직 윈도우 크기에 따른 각 조인 방법의 소요 시간을 나타낸다. 베이직 윈도우 크기가 작을 때는 점증 조인이 완전 조인에 비해 빠르지만 베이직 윈도우 크기가 커질수록 급격히 소요 시간이 증가한다.

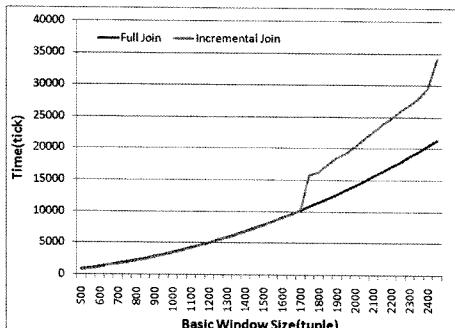


그림 5 베이직 윈도우 크기에 따른 소요 시간 변화

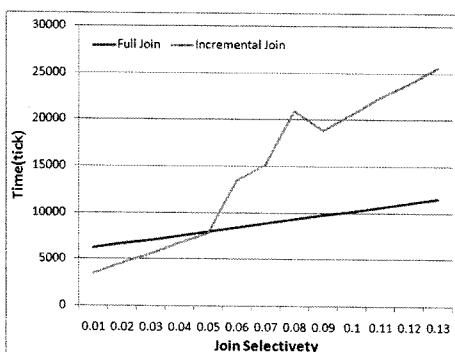


그림 6 조인 선택율에 따른 소요 시간 변화

그림 6은 베이직 윈도우 크기를 1500으로 고정시켰을 때 조인 선택율에 따른 각 조인 방법의 소요 시간을 나타낸다. 조인 선택율이 낮을 때는 디스크에 쓰는 양이 작기 때문에 점증 조인이 좋은 성능을 나타내지만 선택율이 높아질수록 점증 조인의 성능이 나빠진다.

위의 실험 결과들로 볼 때, 하이브리드 방식을 사용한 것이 타당함을 알 수 있다.

4. 관련 연구

점증 조인 알고리즘은 웹 기반 응용을 위해 제안되었으며, XJoin[8], Hash-merge join[9] 등 다양한 알고리즘이 제시되었다. 이를 알고리즘의 특징은 유한한 스트림을 입력으로 받아들이며, 윈도우의 개념이 없다는 점에서 본 논문의 점증조인과는 다르다. 이를 알고리즘은 사용자에게 가능한 빨리 조인 결과를 제공하는데 중점을 두며, 새로운 입력에 대해 메모리에 있는 입력만을 조인 처리한다. 디스크에 위치한 입력은 입력 전달이 모두 끝난 후 처리된다. 즉 입력 순서에 따라 조인 처리를 하지 않는다.

이와는 달리, 연속 스트림에서 결과를 “연속하여” 출력하기 위한 방법으로 MJoin[5], PJoin[6], Golab et al. [7] 등 다수의 알고리즘이 소개되었다. 이를 알고리즘은

무한한 스트림을 연속으로 처리하기 위해 윈도우를 이용한다. 그러나 이들은 텀블링 윈도우(Tumbling window)나 튜플이 들어올 때마다 윈도우를 갱신하는 형태 [1]의 기본적인 윈도우를 다루었으며, 보다 일반화된 형태인 중첩 윈도우에 대해서는 다루지 않았다. 이런 측면에서 볼 때 우리가 대상으로 하는 응용에 적합하지 않다.

한편, 조인 관점에서 부하 제거(Load shedding)나 스케줄링 기법 관련 역시 다양한 연구가 되었다. Das et al.[10] 등을 포함한 이들 방법의 핵심은 입력 튜플 중 조인 결과에 포함되지 않거나 적게 포함되는 튜플을 제거하거나 스케줄링하지 않는 방법이다.

5. 결 론

본 논문에서는 보다 일반화된 윈도우의 형태인 중첩 윈도우를 가정하고, 이를 효율적으로 처리하기 위한 조인 알고리즘을 제안하였다. 제안하는 알고리즘은 메모리의 상황에 따라 점증 조인과 완전 조인을 적절히 활용하며, 중첩 윈도우의 특성을 반영하여 조인 결과의 지연을 최소화하기 위한 교체 대상 선정 방법과 가용 시간 처리 방법을 제시하였다.

참 고 문 현

- [1] Babcock et al., Models and Issues in Data Stream Systems, Proc. of ACM PODS, pp. 1-16, 2002.
- [2] Cranor et al., Gigascope: A Stream Database for Network Applications, Proc. of ACM SIGMOD 2003.
- [3] Harmad et al., Scheduling for Shared Window Joins over Data Streams, Proc. of VLDB 2003.
- [4] Li et al., Semantics and Evaluation Techniques for Window Aggregates in Data Streams, Proc. of ACM SIGMOD, pp. 311-322, 2005.
- [5] Viglas et al., Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources, Proc. of VLDB, pp. 285-296, 2003.
- [6] Ding et al., Joining Punctuated Streams, Proc. of EDBT, pp. 587-604, 2004.
- [7] Golab et al., Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams, Proc. of VLDB, pp. 500-511, 2003.
- [8] Urban et al., XJoin: A Reactively-Scheduled Pipelined Join Operator, IEEE Data Engineering Bulletin 2000.
- [9] Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results, ICDE 2004.
- [10] Das et al., Approximate Join Processing over Data Streams, Proc. of SIGMOD, pp. 40-51, 2003.