전자상거래 데이터의 실시간 분석을 위한 데이터 스트림과 다수의 릴레이션 간의 효율적인 연속 조인 처리 기법

Efficient Processing of Continuous Join Queries between a Data Stream and Multiple Relations for Real-Time Analysis of E-Commerce Data

김해리(Haeri Kim)*, 이기용(Ki Yong Lee)**
haerik11@sookmyung.ac.kr, kiyonglee@sookmyung.ac.kr

초 록

최근 전자상거래 데이터의 실시간 공급이 가능해지면서, 전자상거래 데이터를 실시간으로 분석하기 위한 연속 조인 질의의 처리가 중요해지고 있다. 본 논문에서는 전자상거래 데이터 스트림과 디스크에 저장되어 있는 여러 릴레이션 간의 효율적인 연속 조인 질의 처리 기법을 제안한다. 다양한 실험을 통해, 제안 방법은 기존 방법에 비해 메모리 사용량 및 서비스 처리율을 크게 향상 시킴을 보인다.

1. 서론

최근 전자상거래 데이터의 실시간 공급이 가능해지면서, 전자상거래 데이터를 실시간으로 분석하기 위한 연속 조인 질의의 처리가 중요해지고 있다[1][2][3]. 실시간으로 유입되는 데이터의 스트림(stream)을 S라 하고 디스크에 저장된 릴레이션을 R이라 할 때, S와 R의 연속 조인(continuous join)은 $S \bowtie R$로 표현된다. 그의 사용 예로서, S에 실시간으로 유입되는 각 튜플(tuple) t_s = (*timestamp*, *productID*)이 어떤 시간(*timestamp*)에 어떤 제품(*productID*)이 판매되었는지를 나타내는 데이터라고 하고, 디스크에 저장된 R의 각 튜플 t_r = (*productID*, *price*)이 각 제품(*productID*)의 가격(*price*)을 나타내는 데이터라고 할 때 $S \bowtie R$의 수행결과에 포함된 각 튜플 t_{sr} = (*timestamp*, *productID*, *price*)은 실시간으로 유입되는 각 제품의 가격을 나타낸다.

이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구사업임(2012-001269).

* 숙명여자대학교 컴퓨터과학부 석사과정
** 숙명여자대학교 컴퓨터과학부 조교수

이러한 연속 조인 질의 $S \bowtie R$을 처리하는데 전통적인 중첩 루프 조인(nested loop join)이나 색인 기반 조인(index-based join)을 사용하면, S에 도착하는 매 튜플마다 그와 조인되는 R의 튜플을 찾기 위해 디스크를 여러 번 접근하게 되어 성능이 매우 저하된다. 따라서 이를 극복하기 위해 메시 조인(mesh join)이라는 기법이 최근에 제안되었다[4][5].

하지만 최근 들어 전자상거래 데이터 스트림 S와 다수의 릴레이션 R_1, R_2, …,, R_N 간의 연속 다중(multi-way) 조인 $S \bowtie R_1 \bowtie \ldots \bowtie R_N$에 대한 처리가 점차 중요해지고 있다[2]. 그러나 연속 다중 조인 $S \bowtie R_1 \bowtie \ldots \bowtie R_N$을 처리하기 위해 기존의 메시 조인을 단순히 적용하면 문제점이 발생한다. 2.1절에서 설명할 바와 같이, 메시 조인은 각 릴레이션 R_1, R_2, …,, R_N을 블록(block) 단위로 나눈 뒤, S에 w개의 튜플이 새로 도착할 때마다 R_1, R_2, …,, R_N의 블록을 하나씩 메모리에 읽어들이며 연속 조인 질의를 처리한다. 이 때, 각 릴레이션 $R_i(i = 1, 2, \ldots, N)$의 블록 수를 $B(R_i)$라 하면, 메시 조인을 $S \bowtie R_1 \bowtie \ldots \bowtie R_N$의 처리에 단순히 적용할 경우, 메모리에는 S에 가장 최근에 도착한 $w \cdot B(R_1) \cdot B(R_2) \cdot \ldots \cdot B(R_N)$개만큼의 튜플들을 저장할 공간이 필요하다. (자세한 설명은 2.2절에서 한다.) 즉, 각 릴레이션 R_i의 블록 수를 곱한 수에 비례하는 만큼의 메모리 양이 필요하다. 따라서 릴레이션의 크기가 커지고 릴레이션의 개수가 많아질수록 블록 수의 증가에 따라 요구되는 메모리의 양이 매우 커진다는 단점이 있다. 더욱이 S에 w개의 튜플이 새로 도착할 때마다 S의 튜플들과 R_1, R_2, …,, R_N 각각의 한 블록 씩 간의 조인이 반드시 수행되어야 하므로, 이 시간에 의해 서비스 처리율이 제약을 받는다.

따라서 본 논문에서는 이러한 문제점을 극복하기 위해, 연속 다중 조인 $S \bowtie R_1 \bowtie \ldots \bowtie R_N$을 효율적으로 처리할 수 있는 기법을 제안한다. 제안 방법은 연속 다중 조인을 파이프라인(pipeline) 방식으로 처리함으로써, 메시 조인에 비해 요구되는 메모리 양을 크게 줄이는 한편 서비스 처리율도 크게 향상시킬 수 있다

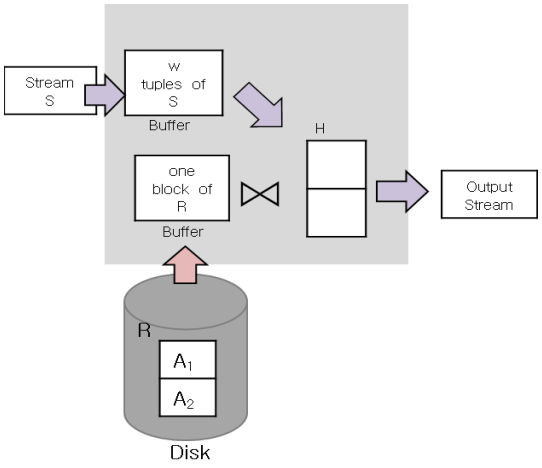
본 논문의 구성은 다음과 같다. 2장에서는 관련 연구와 본 논문의 비교 대상이 되는 메시 조인을 설명하고, 메시 조인을 연속 다중 조인에 단순히 적용했을 때 발생하는 문제점을 기술한다. 3장에서는 제안 방법을 설명하고, 그의 성능을 기존 방법과 비교 분석한다. 4장에서는 실험을 통해 제안 방법이 기존 방법에 비해 좋은 성능을 가짐을 보인다. 5장에서는 결론을 맺는다.

2. 관련 연구

2.1 데이터 스트림과 릴레이션 간의 연속 조인 처리

데이터 스트림 간의 연속 조인 질의 처리에 대해서는 이미 많은 연구가 이루어져왔다[1][2][3]. 하지만 데이터 스트림과 디스크에 저장된 릴레이션 간의 연속 조인 질의에 대해서는 아직까지 많은 연구가 이루어지지 않았다. 데이터 스트림과 디스크에 저장된 릴레이션 간의 연속 조인을 처리하는 가장 간단한 방법은 색인 중첩 루프 조인(indexed nested loop join, INLJ)[4]을 사용하는 것이다. INLJ는 데이터 스트림 S와 디스크 기반 릴레이션 R이 있을 때, S에 새로운 튜플 t가 도착할 때마다 R의 조인 애트리뷰트 상에 생성되어있는 색인을 사용하여 R에서 t와 조인될 튜플들을 찾은 뒤, t

와 그들의 조인 결과를 질의 결과로 내보낸다. 하지만 이 방법의 문제점은 S에 들어오는 매 튜플마다 R에서 그와 조인되는 튜플을찾기 위해 색인을 탐색하는데 따른 여러 번의 임의 접근(random access)이 반복적으로 발생하기 때문에, 디스크 I/O 비용이 매우 커진다는 것이다. 따라서 S에 매우 빠른 속도로 튜플들이 계속해서 들어오게 되면 조인의 성능이 크게 저하된다. 또한 R에 대해 B+-tree와 같은 색인을 유지하는 데 드는 비용 또한 문제가 된다. 이러한 INLJ의 문제점을 보완하기 위해 최근에 메시 조인이라는 방법이 제안되었다[4][5]. 다음 절에서는 본 논문에서 제안하는 방법의 비교 대상이 되는 메시 조인을 자세히 설명한다.



<그림 1> 메시 조인

2.2 메시 조인

본 절에서는 데이터 스트림 S와 하나의 디스크 기반 릴레이션 R 간의 연속 조인 질의 $S \bowtie R$를 효율적으로 처리하기 위해 제안된 기법인 메시 조인을 설명한다. $S \bowtie R$를 처리하기 위해서는 S에 도착하는 각 튜플에 대해 R에서 그와 조인되는 튜플을 찾아 그들 간의 조인 결과를 출력으로 내보내야 한다.

메시 조인은 R을 $m(m \geq 1)$개의 블록 B_1, B_2, …, B_m으로 나눈다. 이와 함께 S에 가장 최근에 도착한 $w \cdot m$개의 튜플들을 메모리 저장공간 H에 저장한다. 메시 조인은 S에 w개의 튜플이 새로 도착할 때마다, 새로 도착한 w개의 튜플을 H에 추가하고, 가장 오래된 w개의 튜플을 H에서 제거한다. 그리고 R의 한 블록 B_i를 메모리에 올린다. R의 블록은 S에 w개의 튜플이 새로 도착할 때마다 B_1, B_2, …, B_m 순서대로 메모리에 올라간다. B_m 다음에는 다시 처음으로 돌아가서 B_1 차례가 된다. 그 후 메시 조인은 H에 저장된 S의 튜플들과 현재 메모리에 올라온 R의 블록 B_i간의 조인을 수행하고, 그 결과를 출력으로 내보낸다.

<그림 1>은 메시 조인의 수행 예를 보여준다. R은 두 개의 블록 A_1, A_2로 나뉘며, 메모리에는 S에 가장 최근에 도착한 $w \cdot 2$개의 튜플을 저장할 저장공간 H가 있다. S에 w개의 튜플이 처음으로 새로 도착하면, 이들을 H에 추가한다. 그리고 R의 첫 블록 A_1을 메모리로 올려 H와 A_1 간의 조인 $H \bowtie A_1$을 수행하고 그 결과를 출력으로 내보낸다. 그 후 S에 w개의 튜플이 다시 새로 도착하면, 이들을 H에 추가한다. 그리고 R의 다음 블록 A_2를 메모리로 올려 H와 A_2 간의 조인 $H \bowtie A_2$를 수행하고 그 결과를 출력으로 내보낸다. 그 후 S에 w개의 튜플이 다시 새로 도착하면, 이들을 H에 새로 추가하고, 가장 오래된 w개의 튜플을 H에서 제거한다. 그리고 R의 첫 블록 A_1을 다시 메모리로 올려 H와 A_1 간의 조인 $H \bowtie A_1$을 수행하고 그 결과를 출력으로 내보낸다. 이렇게 조인을 수행하면 S에 도착한 각 튜플들은 H에서 제거되기 전에 R의 모든 블록 A_1, A_2와 한 번씩 조인이 수행됨이 보장된

다. 따라서 S에 w개의 튜플이 새로 도착할 때마다 R의 블록을 한번씩만 읽으면 되므로 디스크에 접근하는 비용이 크게 줄어든다. 그 결과 $S \bowtie R$를 매우 효율적으로 처리할 수 있게 된다.

2.3 메시 조인의 문제점

2.2절에서 설명한 메시 조인은 데이터 스트림 S와 단일 릴레이션 R 간의 연속 조인 $S \bowtie$ R을 처리하기 위해 제안된 기법이다. 메시 조인을 본 논문에서 다루는 문제인 연속 다중 조인 $S \bowtie R_1 \bowtie \ldots \bowtie R_N$의 처리에 단순히 적용하면 다음과 같이 된다.

메모리에는 각 릴레이션 R_i에 대해 그의 블록을 하나씩 저장할 수 있는 저장공간이 있다. 그리고 각 릴레이션 R_i의 블록 수를 $B(R_i)$라 할 때, S에 가장 최근에 도착한 $w \cdot B(R_1) \cdot B(R_2) \cdot \ldots \cdot B(R_N)$개의 튜플을 저장할 메모리 상의 저장공간 H가 필요하다.

메시 조인으로 $S \bowtie R_1 \bowtie \ldots \bowtie R_N$를 처리하기 위해서는, S에 w개의 튜플이 새로 도착할 때마다, (1) 새로 도착한 w개의 튜플을 H에 추가하고, 가장 오래된 w개의 튜플을 H에서 제거한다. (2) 그리고 각 R_i에서 한 블록씩 뽑았을 때 발생 가능한 모든 블록 조합이 한 번씩 차례대로 나타날 수 있도록, R_1, R_2, …, R_N 중에서 하나를 선택하여 그의 다음 블록을 메모리에 올린다. (3) 그 후 H에 저장된 S의 튜플들과 현재 메모리에 올라온 R_1, R_2, …, R_N의 블록들 간의 조인을 수행하고, 그 결과를 출력으로 내보낸다. 이렇게 조인을 수행하면 S에 도착한 각 튜플은 H에서 제거되기 전까지 R_1, R_2, …, R_N에서 한 블록씩 뽑았을 때 발생 가능한 모든 블록 조합과 한 번씩 조인이 수행됨이 보장된다.

하지만 이 방법은 다음과 같은 문제점을 가진다. (1) 메모리 과다 사용: H에 저장된 S의 튜플들은 R_1, R_2, …, R_N 에서 한 블록씩 뽑았을 때 발생 가능한 모든 블록 조합과 한 번씩 조인이 수행되기 전까지는 H에서 제거될 수 없다. 따라서 H는 S에 가장 최근에 도착한 $w \cdot B(R_1) \cdot B(R_2) \cdot \ldots \cdot B(R_N)$개의 튜플을 저장하고 있어야 한다. 따라서 릴레이션의 크기가 커지고 릴레이션의 개수가 많아질수록 블록 수의 증가에 따라 요구되는 메모리의 양이 매우 커진다. (2) 잦은 조인 연산 수행: S에 w개의 튜플이 새로 도착할 때마다 H에 저장된 S의 튜플들과 R_1, R_2, …, R_N 각각의 한 블록 씩 간의 조인이 반드시 수행되어야 하므로, 이 시간에 의해 서비스 처리율이 제약을 받게 된다.

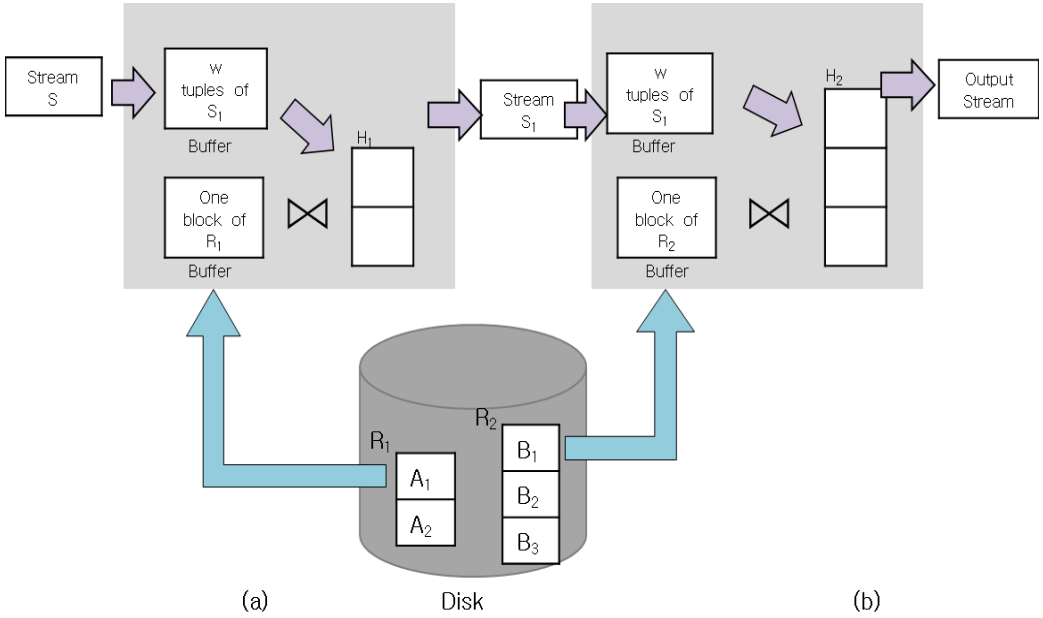
3. 제안 방법

3.1 개요

제안 방법은 S에 w개의 튜플이 새로 도착할 때마다 매번 모든 릴레이션 R_1, R_2, …, R_N을 조인에 참여시키는 대신, R_1, R_2, …, R_N을 순차적으로 조인에 참여시키는 파이프라인(pipeline) 방식으로 질의를 처리한다.

제안 방법은 S에 w개의 튜플이 새로 도착하면 2.2절에서 설명한 메시 조인을 이용하여 S와 R_1 간의 조인을 수행하고 그의 결과를 스트림 S_1으로 내보낸다. S_1에 w개의 튜플이 새로 도착하면, 제안 방법은 S에 w개의 튜플이 새로 도착했을 때와 마찬가지로 메시 조인을 이용하여 S_1과 R_2 간의 조인을 수행하고 그의 결과를 스트림 S_2로 내보낸다. 이와 같은 식으로 마지막으로 S_{N-1}과 R_N 간의 조인이 수행되면 그 결과가 최종 질의 결과가 된다.

<그림 2>는 제안 방법을 사용하여 연속 다중 질의 $S \bowtie R_1 \bowtie R_2$ 를 수행하는 예를



<그림 2> 제안 방법

보여준다. R_1은 두 개의 블록 A_1, A_2로 나뉘며, R_2는 세 개의 블록 B_1, B_2, B_3으로 나뉜다. 메모리에는 S에 가장 최근에 도착한 w·2개의 튜플을 저장할 저장공간 H_1이 있다. S에 w개의 튜플이 도착할 때마다, 메시 조인을 사용하여 H_1와 R_1의 한 블록 간의 조인을 수행하고 그 결과를 S_1으로 내보낸다. 따라서, S_1은 $S \bowtie R_1$의 조인 결과를 나타내는 스트림으로 볼 수 있다. 메모리에는 S_1에 가장 최근에 도착한 w·3개의 튜플을 저장할 저장공간 H_2가 있다. 만약 S_1에 w개의 튜플이 새로 도착하면, 메시 조인을 사용하여 H_2와 R_2의 한 블록 간의 조인을 수행하고 그 결과를 최종 질의 결과로 내보낸다.

이렇게 조인을 수행하면 S에 도착한 각 튜플들은 R의 모든 블록 A_1, A_2와 각각 한 번씩 조인이 수행됨이 보장되며, S_1에 도착한 각 튜플들(즉, $S \bowtie R_1$의 조인 결과에 포함된 각 튜플들)은 R_2의 모든 블록 B_1, B_2, B_3와 각각 한 번씩 조인이 수행됨이 보장된다. 따라서 $S \bowtie R_1 \bowtie R_2$의 결과가 올바르게 출력됨이 보장된다.

3.2 제안 방법의 분석

이 절에서는 $S \bowtie R_1 \bowtie \ldots \bowtie R_N$의 처리에 메시 조인을 단순하게 적용한 방법(2.2절)과 제안 방법(3.1절)을 비교하여 분석한다. 이를 위해 (1) 두 방법에서 요구하는 메모리의 양과, (2) 두 방법의 서비스 처리율을 비교한다. 편의를 위해 제안 방법을 PL(PipeLined Join), 2.2절에서 설명한 방법을 MESH로 각각 표기한다.

먼저 MESH에서 사용하는 메모리 사용량 M_{MESH}는 다음과 같이 계산된다.

$$M_{MESH} = N \cdot B + w \cdot v_S + (w \cdot \prod_{i=1}^{N} B(R_i)) \cdot v_S$$

여기서 $N \cdot B$는 한 블록의 크기를 B라 할 때, 각 릴레이션 R_1, R_2, …,, R_N의 블록을 하나씩 저장하는데 사용되는 메모리 사용량을 나타낸다. $w \cdot v_S$는 S의 한 튜플의 크기를 v_S라

할 때, S에 가장 최근에 도착한 w개의 튜플을 저장하는 버퍼의 크기를 나타낸다. 마지막 항은 2.3절에서 설명한 대로 S에 가장 최근에 도착한 $w \cdot B(R_1) \cdot B(R_2) \cdot \ldots \cdot B(R_N)$개의 튜플을 저장할 메모리 상의 저장공간 H의 크기를 나타낸다.

반면에 PL에서 사용하는 메모리 사용량 M_{PL}는 다음과 같이 계산된다.

$$M_{PL} = N \cdot B + N \cdot w \cdot v_S + (w \cdot \sum_{i=1}^{N} B(R_i)) \cdot v_S$$

여기서 $N \cdot B$는 한 블록의 크기를 B라 할 때, 각 릴레이션 R_1, R_2, ...,, R_N의 블록을 하나씩 저장하는데 사용되는 메모리 사용량을 나타낸다. $N \cdot w \cdot v_S$는 스트림 S, S_1, ..., S_{N-1} 각각에 대해 가장 최근에 도착한 w개의 튜플을 저장하기 위한 버퍼 크기의 총합을 나타낸다. 마지막 항은 저장공간 H_1, H_2, ⋯, H_N의 크기의 총합을 나타낸다.

따라서 PL은 MESH에 비해 S_1, ..., S_{N-1} 각각에 대해 가장 최근에 도착한 w개의 튜플을 저장하기 위한 버퍼가 부가적으로 필요하다. 하지만 MESH에서는 H를 위해 $w \cdot B(R_1) \cdot B(R_2) \cdot \ldots \cdot B(R_N)$에 비례하는 메모리를 사용하는 반면, PMJ는 H_1, H_2, ⋯, H_N을 위해 $w \cdot (B(R_1) + B(R_2) + ... + B(R_N))$에 비례하는 저장공간만을 사용함으로써 총 메모리 사용량을 크게 줄임을 알 수 있다.

다음은 두 방법의 서비스 처리율을 예측한다. 서비스 처리율은 시간당 처리되는 S의 튜플 수를 의미한다. 먼저 MESH의 서비스 처리율 μ_{MESH}는 대략 다음과 같이 예측된다.

$$\mu_{MESH} \approx \frac{w}{c \cdot \sum_{i=1}^{N} \frac{B}{v_{R_i}}}$$

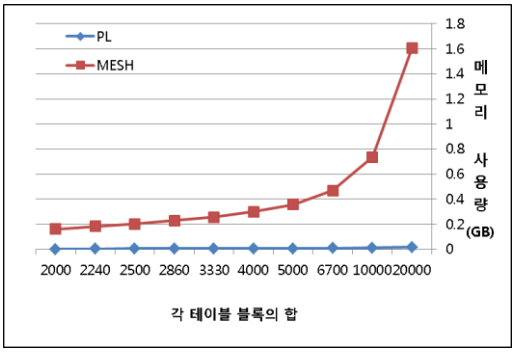
위 식에서 분모는 S에 w개의 튜플이 도착했을 때, H와 현재 메모리에 올라온 R_1, R_2, ...,, R_N의 블록들 간의 조인을 수행하는 데 걸리는 시간을 나타낸다. 여기서 v_{Ri}는 릴레이션 R_i의 튜플 하나의 크기를 나타내며, c는 비례상수이다. H와 현재 메모리에 올라온 R_1, R_2, ...,, R_N의 블록들 간의 조인을 해시 조인으로 처리하는 경우, 각 R_i의 블록에는 B/v_{Ri}개 만큼의 튜플이 있으므로 각 튜플들을 해시 테이블에 넣어 조인하는 비용은 $B/v_{R1} + B/v_{R2} + \cdots + B/v_{RN}$에 비례한다.

반면에 PL의 서비스 처리율 μ_{PL}은 대략 다음과 같이 예측된다.

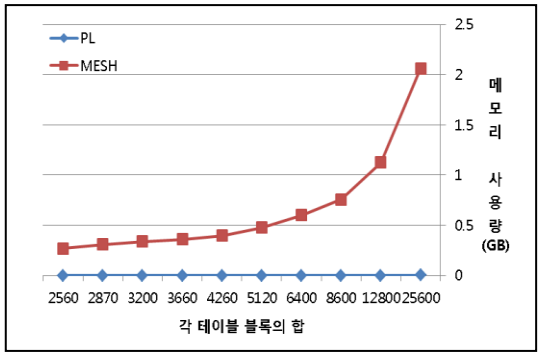
$$\mu_{PL} \approx \frac{w}{c \cdot \sum_{i=1}^{N} (\frac{B}{v_{R_i}} \cdot \prod_{j=1}^{i-1} \frac{1}{B(R_i)})}$$

즉, S에 w개의 튜플이 도착했을 때, H_1와 현재 메모리에 올라온 R_1의 블록 간의 조인을 해시 조인으로 처리하면 B/v_{R1}에 비례하는 시간이 걸린다. 그 결과 S_1에는 평균적으로 $w/B(R_1)$개의 튜플이 새로 유입되므로 (외래 키(foreign key) 조인 가정), H_2와 현재 메모리에 올라온 R_2의 블록 간의 조인은 평균적으로 $(B/v_{R2}) \cdot (1/B(R_1))$에 비례하는 시간이 걸린다. 이와 같이 계산하여 PL에서 각 H_1, H_2, ⋯, H_N에 대해 수행되는 조인 연산의 수행 시간을 모두 더하면 위 식의 분모와 같이 된다.

따라서 위 두 식을 비교하면 PL은 MESH에 비해 릴레이션의 블록 수가 많아질 수록 더 좋은 서비스 처리율을 가짐을 알 수 있다. 이는 S에 w개의 튜플이 새로 도착할 때마다 무조건 모든 릴레이션 R_1, R_2, ...,, R_N의 한 블록씩을 조인에 참여시키는 MESH



<그림 3> 메모리 사용량
(릴레이션이 4개일 때)



<그림 4> 메모리 사용량
(릴레이션이 5개일 때)

에 비해, $S \bowtie R_1 \bowtie \ldots \bowtie R_i$의 결과가 w개 만큼 생성되어야만 $S \bowtie R_1 \bowtie \ldots \bowtie R_{i+1}$를 수행하는 PL이 조인 연산에 더 적은 시간을 사용함을 의미한다.

4. 성능 평가

4.1 실험 환경

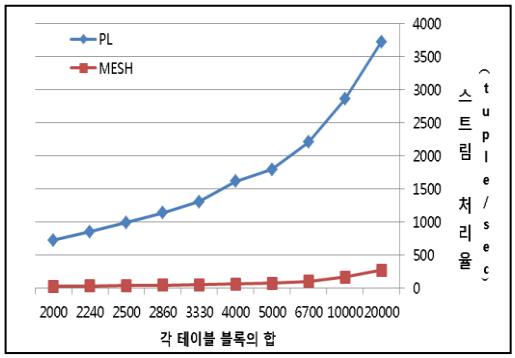
본 절에서는 PL와 MESH의 성능을 비교하기 위한 실험 환경 및 방법을 기술한다.

(1) 실험 환경: 데이터 스트림과 조인되는 릴레이션의 개수가 각각 4개일 때와 5개일 때, 릴레이션의 블록 개수의 합을 증가시키면서 실험을 수행하였다. 각 릴레이션의 조인 애트리뷰트(attribute)의 값의 범위는 기존 메시 조인[4]의 실험 방법과 동일하게 각각 $[1, 7.2\times10^5]$, $[1, 3.0\times10^5]$, $[1, 4.8\times10^5]$, $[1, 5.0\times10^5]$, $[1, 6.0\times10^5]$으로 하였으며, 균등 분포(uniform distribution)로 값을 생성하였다. 모든 릴레이션의 튜플 하나의 크기는 400 bytes로 동일하다고 가정하였으며, 각 블록의 크기는 200 Mbytes로서 한 블록에는 2000개의 튜플이 들어가도록 하였다. 그리고 데이터 스트림에 도착하는 튜플들의 조인 애트리뷰트 값은 위의 조인 애트리뷰트 값의 범위 내에서 임의로 생성하였다. 실험은 윈도우 7 운영체제가 설치되었으며 CPU가 Intel Core i7-2600이고 메모리는 4GB인 컴퓨터에서 수행되었다.

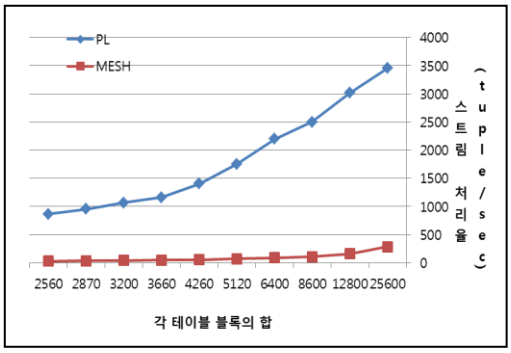
(2) 실험 방법: 본 실험에서는 PL와 MESH의 성능을 메모리 사용량과 서비스 처리율 두 가지 척도로 측정하였다. 메모리 사용량은 각 방법이 수행될 때 사용한 메모리의 총량을 측정하였다. 서비스 처리율은 초당 처리된 데이터 스트림의 튜플 수로써, 각 방법을 10분간 수행하여 첫 1분 예열 시간을 제외하고 나머지 시간 동안 처리된 S의 튜플 수를 총 수행 시간으로 나누어 구하였다. 각 척도에 대해 릴레이션의 개수가 각각 4개일 때와 5개일 때, 릴레이션의 블록 개수의 합을 증가시키면서 실험을 수행하였다.

4.2 실험 결과

<그림 3>과 <그림 4>는 각각 릴레이션의 개수가 4개와 5개일 때, 릴레이션의 블록 개수를 증가시키면서 PL과 MESH의 메모리 사용량을 측정한 결과이다. 가로축은 릴레이션에 포함된 블록 수의 총합을 나타내며, 세로축은 메모리 사용량(GB)를 나타낸다. 3.2절에서 분석한 바와 동일하게, 블



<그림 5> 서비스 처리율
(릴레이션이 4개일 때)



<그림 6> 서비스 처리율
(릴레이션이 5개일 때)

록 수가 증가할 수록 메모리 사용량이 두 기법 모두 증가하는 것을 볼 수 있다. 다만 MESH의 메모리 사용량은 각 릴레이션의 블록 수의 곱에 비례하여 매우 크게 증가하는 반면, PL의 메모리 사용량은 각 릴레이션의 블록 수의 합에 비례하여 증가하기 때문에 상대적으로 거의 변화하지 않는 것처럼 보인다. 따라서 제안 방법은 기존 방법에 비해 메모리 사용량을 큰 폭으로 감소시킴을 알 수 있다.

<그림 5>와 <그림 6>는 각각 릴레이션의 개수가 4개와 5개일 때, 릴레이션의 블록 개수를 증가시키면서 PL과 MESH의 서비스 처리율을 측정한 결과이다. 가로축은 릴레이션에 포함된 블록 수의 총합을 나타내며, 세로축은 초당 처리된 S의 튜플 수(tuples/sec)를 나타낸다. 그림에서 볼 수 있듯이 PL은 MESH 보다 훨씬 높은 서비스 처리율을 보이고 있다. 이것은 3.2절에서 분석한 것처럼, PL은 MESH에 비해 S에 w개의 튜플이 도착할 때마다 수행되는 조인 연산의 비용이 더 적기 때문이다. 또한 블록 개수의 합이 증가할 수록 서비스 처리율이 증가하는 것을 볼 수 있다. 이것은 블록 수가 증가할 수록 S에 w개의 튜플이 도착했을 때마다 수행되는 조인 연산의 비용이 줄어들기 때문이다. 즉, MESH에서는 H와 R_1, R_2, …,, R_N의 한 블록씩들을 조인할 때 블록 수가 증가할 수록 조인의 중간 결과 크기가 작아지며, PL에서는 $S \bowtie R_1 \bowtie \ldots \bowtie R_i$의 수행 결과로 다음 S_i로 유입되는 결과 튜플의 수가 블록 수의 곱에 비례하는 만큼 큰 폭으로 줄어들기 때문이다. 따라서 릴레이션이 증가하거나 릴레이션의 블록 수가 증가할 수록 더 많은 스트림 튜플을 처리할 수 있게 된다. 하지만 <그림 4>와 <그림 5> 모두 PL이 훨씬 좋은 서비스 처리율을 보여주고 있음을 알 수 있다.

5. 결론

본 논문은 실시간으로 유입되는 전자상거래 데이터 스트림과 다수의 릴레이션간의 연속 다중 조인을 효율적으로 처리하는 기법을 제안하였다. 그리고 이를 기존의 방법을 단순하게 적용했을 때와 비교 분석하였다. 또한 실험을 통해, 본 논문에서 제안하는 기법이 기존 방법에 비해 메모리 사용량과 서비스 처리율 측면에서 모두 우수한 성능을 보이고 있음을 증명하였다.

참고문헌

[1] A. Karakasidis and I. Hellas. ETL queues for active data warehousing. In Proc. Int.Workshop on Information Quality in Informational Systems (IQIS), pages 28-39, 2005.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Processing sliding window multi-joins in continuous queries over data streams. In Proc. ACM SIGMOD-SIGACTSIGART Symposium on Principles of Database Systems (PODS), pages 1-16, Madison,Wisconsin, USA, June 2002.

[3] C. White. Intelligent business strategies: Real-time data warehousing heats up. DM Review, 2002.

[4] Hector Garcia-Molina, Jeffrey D.Ullman, Jennifer Widom. DATABASE SYSTEMS: The complete Book: International Edition, 2/E. pages 718-745 (2009)

[5] Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N.: Meshing Streaming Updates with Persistent Data in an Active Data Warehouse. IEEE Trans. on Knowl. And Data Eng. 20(7), pages 976-991 (2008)

[6] Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N. E.: Supporting Streaming Updates in an Active Data Warehouse. In: IEEE 23rd International Conference on Data Engineering, ICDE 2007, Istanbul, Turkey, pages 476-485 (2007)