스트림 질의의 동적 최적화를 위한 질의 계획 재구성 기법

이원근° 이상돈
목포대학교 멀티미디어 공학과
wglee2000@hotmail.com°, sdlee@mokpo.ac.kr

Query Plan Reordering Techinque for Dynamic Optimization of Stream Queries

Won-Gun Lee°, Sang-Don Lee
Dept. of Multimedia Engineering, Mokpo National Univ.

요 약

최근 들어 데이터가 연속적으로 생성되므로 인해 디스크에 저장된 형태로 모델링되기 어려운 특성을 갖는 데이터 응용환경에 대한 관심이 증대하고 있다. 스트림 데이터를 대상으로 이루어지는 스트림 질의는 저장된 릴레이션 내의 데이터를 대상으로 한번 적용되고 마는 기존의 데이터 응용에서와는 달리, 한번 등록이 되면 계속적으로 입력 데이터 스트림을 감시하다가 질의를 만족시키는 튜플이 발생될 때마다 결과를 출력하는 연속성을 갖는다.

이러한 데이터 스트림 처리 시스템에서 성능 향상을 위한 질의 계획 최적화에 대한 연구가 이루어지고 있으며, 이를 위한 하나의 방법으로 현재 사용중인 질의 계획에서 질의 계획의 일부를 재구성하기 위해서 최적화 대상 질의 계획으로의 입력을 중단하고 최적화된 새로운 질의 계획으로 바꾸어 임시 저장된 데이터를 새로운 질의 계획에 입력하는 방법이 이용되고 있다. 그러나 이 방법을 사용하는 경우 입력 데이터 버퍼링을 위한 저장공간에 대한 비용이 증가하고, 부정확한 값을 산출을 유발할 수 있는 등 몇 가지 문제점을 안고 있다.

본 논문에서는 이러한 문제를 해결하기 위하여 최적화 대상이 되는 질의 계획을 일시적으로 중복시켜 최적화가 진행되고 있는 과정 중에도 기존의 질의 계획이 입력 스트림을 계속 처리하고, 최적화된 새로운 질의 계획으로 입력 스트림을 처리하도록 하는 일시 중복을 이용한 동적 질의 계획 재구성 기법을 제시하였다.

1. 서 론

최근 들어 전통적인 관계형 데이터가 아닌 가변적인 데이터 스트림(Data Stream)을 처리하기 위한 응용에 대한 연구가 활발히 진행되고 있다. 이러한 응용의 예로는 재무 응용, 네트워크 모니터링, 보안, 원격 통신 데이터 관리, 웹 어플리케이션, 제조업, 센서 네트워크 등이 있다[4]. 이러한 데이터 스트림을 처리하기 위한 응용을 연구하고 있는 프로젝트로는 브라운대학과 MIT에서 Aurora(sensor 모니터링, dataflow)[1,2,3], 스탠포드 대학의 STREAM(범용 DSMS)[4], 버클리 대학의 Telegraph(센서에 대한 적응엔진)[5], OGI/Wisconsin의 Niagara(인터넷 XML database)[6] 등이 진행 중에 있다.

스트림 데이터를 대상으로 이루어지는 스트림 질의는 저장된 릴레이션내의 데이터를 대상으로 한번 적용되고 마는 기존의 데이터 응용에서와는 달리, 한번 등록이 되면 계속적으로 입력 데이터 스트림을 감시하다가 질의를 만족시키는 튜플이 발생될 때마다 결과를 출력하는 연속성을 갖는다[4]. 이러한 스트림 질의를 효과적으로 처리하고 자원을 적절히 배치하기 위해서는 기존의 DBMS와 비슷하게 여러 가지 최적화 기법을 적용할 필요가 있다. 그러나 기존의 DBMS 환경과 달리 스트림 데이터 처리 시스템에서는 연속된 데이터의 입력, 이를 연속적으로 처리해야 하는 연산자의 특성, 그리고 연산자 내에 존재하는 데이터 때문에 기존의 질의 계획 최적화 기법을 그대로 적용하는 것은 불가능하다[1].

스트림 질의 계획 최적화 방법중의 하나로 현재 사용중인 질의 계획에서 일부를 재구성하기 위해 대상 질의 계획으로의 입력을 임시 저장소에 버퍼링하고, 대상 질의 계획을 최적화하여 임시로 저장된 데이터를 최적화된 연속질의에 입력하는 방안이 있다. 이러한 방법은 이미 Aurora[1,2,3]에서 사용되고 있다. Aurora는 박스(box)라 불리는 연산자들을 사용하여 Data-flow 형식의 네트워크에 의해 스트림 질의가 처리되도록 구성되어 있다[1]. 이 네트워크의 일부(이를 서브네트워크라 부른다.)를 선정하여 위에서 설명한 질의 계획 최적화 방법을 적용한다.

그러나 이 방법에는 추가적인 저장공간 요구, 출력 값의 부정확성, 출력값의 지연 발생 등 몇가지 문제점을 가지고 있다.

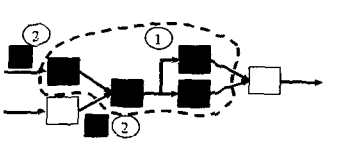
본 논문에서는 이러한 문제점을 해결하기 위해 최적화 대상 질의 계획으로의 입력을 멈추지 않고 질의 계획의 일부를 일시 중복시켜 최적화된 질의 계획으로 재구성하는 동적 질의 계획 재구성 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장은 기존의 동적 질의 계획 재구성 방법에 대하여 알아보고, 3장에서는 기존 접근 방식에서 발생할 수 있는 문제점을 해결하기 위한 일시 중복을 이용한 동적 질의 계획 재구성 기법의 절차와 동기화 방안에 대하여 설명하고, 4장에서 기존의 접근 방법과 본 논문에서 제안하고 있는 일시 중복을 이용한 동적 질의 계획 재구성 기법에 대한 성능 비교를 하고, 5장에서 결론을 맺는다.

2. 기존 접근 방법

Aurora 시스템은 기본적으로 응용프로그램 관리자(application administrator)에 의해 정의된 방식으로 입력 스트림을 처리한다. Aurora는 본질적으로 데이터 흐름(Data-flow) 시스템이며, 프로세스 흐름이나 작업 흐름 시스템에서 사용하는 박스와 화살표 다이어그램을 사용한다. Aurora는 ad-hoc 질의를 지원하기 위해 과거 정보를 저장할 수 있다[1,2,3].

박스는 각각의 연산자를 의미하며, 화살표는 연산자로의 입력과 출력 스트림의 흐름을 나타내고 있다. 이러한 박스와 화살표의 조합은 질의 계획이 되며, Aurora에서는 이를 네트워크라 부른다. 관심 대상이 되는 네트워크의 일부를 서브네트워크라 부르고, Aurora 시스템을 최적화하기 위해 서브네트워크내 박스의 순서를 재구성하는 연구가 진행 중에 있다.



[그림 1] Aurora에서 박스 재구성을 이용한 최적화 절차

이는 전통적인 관계형 데이터베이스에서 질의 최적화를 위해 동일하지만 더 효과적인 형태를 갖도록 연산자를 재배치하는

것과 유사하다. Aurora에서는 두 연산이 상호교환 가능할 때 동일한 기법을 적용할 수 있다[1]. [그림 1]은Aurora에서 박스 재구성을 이용한 최적화 절차를 도식한 것이다. [그림 1]을 살펴보면, Aurora 시스템은 먼저 ①에서와 같이 박스 재구성을 위해 최적화 대상 서브네트워크를 선정하고, 두 번째 단계로 ②와 같이 해당 서브네트워크로의 입력을 버퍼링하여 해당 서브네트워크로의 입력을 차단한다. 세 번째 단계로 대상 서브네트워크내의 데이터를 처리하여 서브네트워크를 비우고, 네 번째로 모든 데이터가 처리되면 ①에서 선정된 서브네트워크를 최적화 성능 모델에 따라 최적화를 수행하고 최적화가 완료되면 마지막으로 ②에서 버퍼링된 내용을 서브네트워크로 보내도록 스케줄러에 지시한다. 이때 이 서브네트워크에 의해 영향을 받는 출력은 응답시간에 일시적 충격(blip)이 발생할 것이다. 그러나 나머지 다른 네트워크는 영향을 받지 않고 데이터 스트림을 처리할 수 있다[1].

그러나 이 방법에는 몇가지 문제점을 갖는다. 첫째, 서브네트워크로의 입력을 버퍼링하기 위해서는 일정량의 저장공간이 필요하다. 만약 시스템으로의 입력 데이터 스트림의 양이 많은 경우 메모리 버퍼의 크기를 초과하게 되면 저장소관리자(Storage Manager)는 해당 데이터를 디스크에 저장하게 되고, 다시 입력을 수행할 때 디스크에서 해당 데이터 스트림을 읽어오게 된다. 이로 인해 필요 이상의 디스크 I/O를 발생하게 되므로 영향을 받게될 출력에 대한 충격이 예상치 보다 커질 수 있다. 또한 서브 네트워크로의 입력 스트림의 수가 많을수록 입력 스트림을 위한 버퍼링 공간이 많이 필요하게 된다. 둘째, 서브네트워크로의 입력을 차단하고 서브네트워크에 존재하는 데이터를 처리하는 과정에서 특정 연산자의 경우 일정 크기 이상의 튜플이 존재하여야 연산을 정상 처리할 수 있게 된다. 즉 연산자가 필요로 하는 데이터의 양이 부족하여 원하는 출력결과를 산출할 수 없게 되는 상황이 발생할 수 있다. 셋째, 위에서 대상 서브네트워크의 영향을 받는 출력은 응답시간에 일시적인 충격이 발생할 수 있다고 하였다. 만약, 대상 서브네트워크의 크기가 매우 크거나 해당 서브네트워크가 전체 네트워크의 상류에 위치하는 등의 이유로 영향을 미치게 되는 범위가 크게 되면 거의 모든 결과에 대한 응답시간에 영향을 미치게 되어 출력의 QoS를 만족시키지 못하는 상황이 발생할 수 있다.

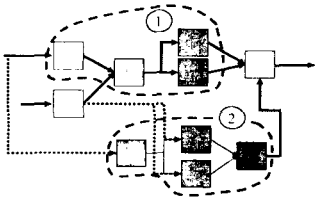
위에서 나열한 것과 같은 문제점들을 해결하기 위한 방법으로 본 논문에서는 일시 중복을 이용한 동적 질의 계획 재구성 기법을 제안한다.

3. 일시 중복을 이용한 동적 질의 계획 재구성 기법

질의 계획을 재구성하기 위해 대상 질의 계획의 입력을 차단하고 최적화를 실행하는 방법이 갖는 문제점을 해결하기 위해 본 논문에서는 대상 질의 계획을 일시적으로 중복시켜서 처리하는 기법을 제안한다.

3.1 일시 중복을 이용한 동적 질의 계획 재구성 절차

[그림 2]는 일시적 중복을 이용한 동적 질의 계획 재구성 절차를 설명하고 있다.



[그림 2] 일시적 중복을 이용한 동적 질의 계획 재구성 절차

1단계에서 ①과 같이 최적화 대상이 되는 질의 계획을 식별한다. 2단계에서 최적화 성능 모델을 이용하여 최적화된 새로운 질의 계획을 생성한다. 이렇게 생성된 질의계획이 [그림 2]에서 ②에 해당한다. 이때 기존의 서브네트워크는 계속 활성화되어 있으며, 입력 데이터 스트림을 처리하고 있다. 3단계에서 기존의 질의 계획에 의한 출력과 새로 생성된 질의 계획에 의한 출력간의 동기화방법 결정을 결정한다. 4단계에서 동기 정보와 함께 새로운 서브네트워크로 데이터 스트림을 입력한다. 5단계에서는 3단계에서 지정된 동기화 방법에 따라 기존의 질의 계획에 의한 출력과 새로운 질의 계획에 의한 출력을 동기화한다. 6단계에서 이제 동기화 되었으므로 기존의 질의 계획을 삭제하고 새로운 질의 계획에 의해서만 질의를 처리하도록 한다.

위의 단계 중에서 1단계의 최적화 대상 질의 계획 식별은 이미 기존의 방식에 의해 완료된 상태이며, 2단계에서의 성능 모델은 이미 결정되어 새로운 질의 계획의 형태는 이미 정해진 상태로 가정한다. 또한, 최적화 대상이 되는 기존의 질의 계획의 출력과 최적화가 완료된 새로운 질의 계획의 출력은 동일하며, 질의 계획내의 모든 연산자는 동기 정보를 모두 인식하며, 처리 방법을 알고 있다고 가정한다.

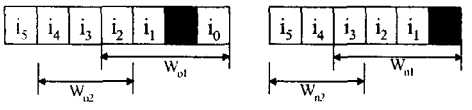
3.2 동기화 방안

두 개의 질의 계획에서의 출력을 동기화 방법으로는 별도의 동기 정보를 이용하는 방법과 입력 스트림내의 타임스탬프를 이용하는 방법으로 나눌 수 있다.

첫번째, 별도의 동기 정보를 이용하는 방법은 생성된 동기 정보를 데이터 스트림에 추가하고 이 동기 정보를 출력부에서 확인하여 두 개의 스트림을 동기시키는 방법이다. 이 방법은 연산자의 종류에 관계없이 사용할 수 있기 때문에 광범위하게 사용할 수 있는 장점이 있다. 그러나 동기 정보를 추가로 생성하기 위한 비용과 입력스트림에 동기 정보를 삽입하는데 추가적인 비용이 소요된다.

두번째, 입력스트림의 타임스탬프의 값을 동기 정보로 활용하는 방법은 입력 스트림의 특정 시점의 타임스탬프를 동기 정보로 선정하고 출력부에서 이를 검출하여 동기시키는 방법이다. 이 경우에는 동기 정보를 추가로 생성하지 않고, 입력스트림에 대한 정보를 변경하지 않기 때문에 이로 인한 추가적인 비용은 발생하지 않는 장점을 갖는다. 그러나, 별도의 동기 정보를 이용한 동기화 방법과 달리 타임스탬프 동기 정보를 이용하는 경우에는 질의 계획내 연산자의 종류에 따라(예, Filter 연산자, Drop 연산자) 동기 정보로 설정된 타임스탬프를 삭제하거나 변경하게 되는 현상이 발생할 수 있다. 따라서 타임스탬프를 이용한 동기화 방법은 질의계획 내에서 해당 타임스탬프의 값에 변경/삭제가 발생하지 않을 것이 보장되는 경우에만 사용하여야 한다.

동기 정보는 질의 계획 내에서 사용되는 연산자가 윈도우 연산자를 포함하는 경우에 몇가지 문제점을 발생시킬 수 있다. 즉 질의 계획이 윈도우 연산자를 포함하는 경우에는 기존의 질의 계획과 새로운 질의 계획의 출력이 서로 달라져서 동기화시 문제가 발생할 수 있다. [그림 3]은 윈도우 연산자에 의해 각각이 질의 계획의 출력이 달라지는 예를 보인 것이다.



a. 기존 질의 계획내에 포함된 윈도우 연산자

b. 새 질의 계획내에 포함된 윈도우 연산자

[그림 3] 윈도우 연산자를 포함하는 질의 계획의 출력

[그림 3]은 윈도우의 크기가 3이며, 2씩 옮겨가는 슬라이딩 윈도우를 포함하는 질의 계획의 경우를 나타내고 있다. 각각의 질의 계획에는 동기 정보를 시작으로 하는 데이터 스트림이 입력되기 시작한다고 가정한다. 이 경우 기존의 질의 계획에서는

[그림 3]의 a에서 보는 바와 같이 윈도우 W_{o1}과 W_{o2}를 만들고 해당 투플에 대한 연산이 이루어진다. 여기에서 i_0는 동기 정보를 수신하기 전에 이미 윈도우에 입력된 투플을 나타낸다. 동일한 시점에서 입력 스트림을 수신한 새로운 질의 계획에서 만들어진 윈도우 W_{n1}과 W_{n2}내에 포함된 투플들은 W_{o1}과 W_{o2}내의 투플들과 서로 일치하지 않아 서로 다른 값을 출력하게 되고 결과적으로 동기화에 실패하게 된다.

이러한 현상을 방지하기 위해서는 각 질의 계획이 포함하는 윈도우 연산자 내에 포함되는 투플들을 일치시킴으로써 해결할 수 있다. 본 논문에서는 윈도우 연산자 내에 포함되는 투플을 일치시키기 위해 윈도우 연산자가 동기 정보를 수신하게 되면 윈도우 내에 존재하는 기존의 투플을 이용한 연산을 모두 처리하고, 동기 정보 이후의 첫 번째 투플을 시작으로 하는 새로운 윈도우를 만들도록 한다. 이렇게 하면 윈도우는 항상 동기 정보 이후 첫 번째 투플로 시작하게 되므로 새로운 질의 계획내의 윈도우 연산자와 기존의 질의 계획내의 윈도우 연산자 내에 포함되는 투플을 서로 일치시킬 수 있게 된다.

4. 성능비교

본 장에서는 기존의 동적 질의 계획 재구성 기법과 임시 중복을 이용한 동적 질의 계획 재구성 기법의 성능을 출력 지연과 메모리 사용량에 대하여 비교한다.

먼저 출력 지연에 대한 성능 비교를 위하여 각각의 방법에서 최적화 이후 새로 입력된 투플이 처음으로 출력되는데 까지 걸리는 시간을 T 라고 한다. 기존의 방법을 이용하는 동적 질의 계획 재구성 기법에서 최적화 이후 새로 입력된 투플이 처음으로 출력되는데 까지 걸리는 시간 T_o는 다음과 같다.

T_o = *(대상 질의 계획 내의 투플 배출 시간) + (최적화 시간) + (최적화된 질의 계획의 데이터 처리 시간) + (버퍼링 데이터 처리 시간)*

여기에서 대상 질의 계획 내의 투플 배출 시간은 대상 질의 계획 내의 연산자들이 일부 데이터로 연산을 수행할 수 있는지 없는지에 따라 값이 차이가 날 수 있으나, 일반적으로 윈도우 연산자를 포함하는 질의 계획은 일부 투플로의 연산 결과가 의미가 없기 때문에 거의 대부분의 경우에 고려 대상이 된다. 일반적으로 이 인자는 대상 질의 계획의 크기에 비례하여 증가한다. 최적화 시간은 각각의 연산자의 순서를 성능 모델을 기반으로 재배열하는데 걸리는 시간을 의미하며, 주로 전체 성능에는 영향을 미치지 않는다고 가정할 수 있다. 최적화된 질의 계획의 데이터 처리 시간은 질의 계획 최적화가 이루어지고 버퍼링되고 있던 최초의 입력 데이터가 첫 번째 연산자를 거쳐 마지막 연산자에 의해 연산이 끝마쳐지는데 까지 걸리는 시간을 의미한다. 이 인자 또한 대상 질의 계획의 크기에 비례하여 증가한다. 버퍼링 데이터 처리시간은 최적화를 위해 입력을 차단한 시점부터 입력 버퍼에 저장된데이터를 모두 처리하는데 까지 소요되는 시간을 의미한다. 이 시간은 입력 스트림의 속도가 매우 빠른 경우에는 매우 큰 인자로 작용될 수 있다.

이번에는 임시 중복을 이용한 동적 질의 최적화 기법에서 최적화 이후 결과 투플이 처음으로 출력되는데 까지 걸리는 시간 T_n을 살펴보자.

T_n = *(각 연산자에서 동기 정보 처리 시간의 합) + (동기화 소요 시간)*

여기에서 각 연산자에서 동기 정보 처리 시간의 합은 기존의 방법을 사용하는 경우에는 불필요한 추가적인 연산에 필요한 시간이다. 동기화에 걸리는 시간은 두 개의 질의 계획(기존의 질의 계획과 최적화된 질의 계획)에서의 출력에서 동기정보를 추출하고 이 정보를 이용하여 출력을 동기화 시키는데 소요되는 시간을 의미한다.

이를 정리하면, 대상 질의 계획의 크기가 증가함에 따라, 그리고 입력 스트림의 양이 증가하면 할수록 T_o는 T_n에 비하여 더 빠른 속도로 증가할 것임을 알 수 있다.

다음은 각각의 동적 질의 계획 재구성 기법의 메모리 소모량에 대하여 평가한다. 먼저 기존의 동적 질의 계획 재구성 기법에서 메모리를 필요로 하는 항목은 입력 스트림을 버퍼링하는데 필요한 저장 공간이다. 입력 스트림을 버퍼링하는데 필요한 저장공간은 대상 질의 계획으로의 입력 스트림의 수에 비례하여 증가한다.

한편, 일시 중복을 이용한 질의 계획 재구성 기법에서 필요로 하는 메모리 항목은 동기 정보를 생성하고 설정하는데 필요한 공간, 일시적으로 중복된 두 개의 질의 계획을 유지하는데 소요되는 공간과 동기화 연산에 필요한 공간이다. 중복되는 질의 계획을 유지하는데 필요한 메모리 양을 제외하면 무시할 만한 양이다. 그리고 중복되는 질의 계획을 유지하는데 소요되는 메모리의 양은 대상 질의 계획의 크기에 비례하지만 대상 질의 계획으로의 입력의 개수와는 무관하며, 필요한 메모리의 양은 최대값은 기존 질의 계획 유지에 필요한 메모리양의 2배이다.

5. 결론

현재 사용되고 있는 데이터 스트림 처리를 위한 질의 계획을 최적화 방법에는 몇가지 문제가 있음을 보였다. 첫째로 입력 데이터 버퍼링을 위한 저장공간에 대한 비용이 증가할 수 있으며, 둘째로 대상 질의 계획내의 데이터를 배출하는 과정에서 일부 연산자의 특성에 의해 부정확한 값을 산출할 수 있으며, 셋째로 출력에 가해지는 일시적 충격(blip)이 최적화 대상 서브네트워크의 위치에 따라 시스템 네트워크의 많은 부분에 영향을 줄 수 있다는 것이다.

이러한 문제점들을 해결하기 위하여 본 논문에서는 최적화 대상 질의 계획으로의 입력을 차단하지 않고 최적화된 새로운 질의 계획를 생성하여 질의 계획을 일시 중복시켜 입력 스트림을 처리하도록 하였다. 그리고 이 과정에서 중복되는 질의 계획간의 출력을 동기화하는 방안에 대하여 설명하였다. 임시 중복을 이용한 동적 질의 계획 재구성 기법은 버퍼링으로 인해 야기되는 문제점을 해결하고, 서브네트워크내의 데이터 배출과정에서 부정확한 값 산출을 최소화할 수 있으며, 출력에 가해지는 일시적 충격을 해소하였다.

참고문헌

[1]D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik. Monitoring Streams: A New Class of Data Management Applications. In proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02), August 20-23, Hong Kong, China.

[2]M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, S. Zdonik. Scalable Distributed Stream Processing. In proceedings of the First Biennial Conference on Innovative Database Systems (CIDR'03), Asilomar, CA, January 2003.

[3]D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, , Y. Zing, R.Yan, , S. Zdonik. Aurora: A Data Stream Management System. Demo proposal.

[4]Brian Babcock, Shivnath Badu, Mayur Datar, Rajeev Motwani, Jennifer Widom, "Models and Issues in Data Stream Systems", Proc. of the 2002 ACM Symp. on Principles of Database Systems (PODS 2002), June 2002

[5]Joseph M. Hellerstein, Michael J. Franklin, Sirish Chandrasekaran, Amol Deshpande, Kris Hildrum, Sam Madden, Vijayshankar Raman, Mehul Shah, "Adaptive Query Processing: Technology in Evolution.", IEEE Data Engineering Bulletin 23(2): 7-18 2000.

[6]Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases" SIGMOD Conference 2000.