XML 스트림 데이터에 대한 연속 질의 처리 시스템

한 승 철[†] · 강 현 철[††]

요 약

스트림 데이터 처리는 여러 응용 분야에서 많은 관심을 가지고 활발한 연구가 수행되고 있다. 특히 모니터링, 센서 네트워크 등의 응용 분야에서 끊임없이 생성되는 대량의 스트림 데이터에 대한 효율적인 처리 요구가 높아지고 있다. 본 논문에서는 스트림 데이터에 대한 연속 질의 처리 시스템 모델을 개발하고 성능을 평가한다. 스트림 데이터 모델로 웹상의 데이터 교환 표준으로 자리잡은 XML을 사용하였고 연속 질의는 XQuery에 시구간을 추가한 형태로 표현하였다. 제시된 시스템에서는 질의 처리의 성능 향상을 기하기 위해 질의 결과 값을 백그라운드 처리를 통해 생성하고 결과 값을 실체화하여 후속 질의의 결과 계산에 이용하는 기법을 제공한다. 성능 평가 실험을 통해서 XML 스트림 데이터 처리를 위한 제시한 시스템의 타당성을 보였다.

A Continuous Query Processing System for XML Stream Data

Seungchul Han[†] · Hyunchul Kang[††]

ABSTRACT

Streaming data processing is an area of interest with much research under way. There has been increasing attention on the demands for efficient processing of streaming data produced in the application areas such as monitoring and sensor network. We have developed a continuous query processing system for streaming data and evaluated its performance in this paper. XML, the standard for data exchange on the web, is used as the model for the streaming data and the XQuery appended with a time interval is adopted as the query language for expressing continuous queries. In the proposed system, the query result is produced through background processing and materialized for reuse in subsequent query processing. Through a detailed set of performance experiments, we showed the effectiveness of the proposed system.

키워드 : XML 스트림 데이터(Streaming Data), 연속 질의(Continuous Query), 이벤트 라우팅(Event Routing), 백그라운드 질의 처리(Background Query Processing), 질의 실체화(Query Materialization)

1. 서 론

모니터링, 센서 네트워크 등 연속적인 스트림 데이터를 생성하는 응용 분야가 증가함에 따라 스트림 데이터의 효율적인 처리 요구가 높아지고 있다. 이에 따른 스트림 데이터 처리 시스템에 대한 활발한 연구가 수행되고 있다[1-5]. 그러나 기존의 스트림 데이터에 대한 연구는 대부분 관계 데이터를 기반으로 한다. 본 논문에서는 웹상의 데이터 교환의 표준으로 자리잡은 XML을 데이터 모델로 하여 XML 스트림 데이터에 대한 연속 질의 처리 시스템을 제안하고 개발한다. 또한 성능 평가 실험을 통해서 XML 스트림 데이터 처리를 위해 본 논문에서 제안한 시스템의 타당성을 보인다.

스트림 데이터의 특징은 실시간 처리를 요구하며, 연속적이고, 데이터 양에 한계가 없다는 것이다. 이런 연속적이고 빠르게 발생하는 스트림 데이터를 기존의 DBMS를 이용하여 처리하는 것은 공간 효율 면이나 질의에 대한 응답시간 면에서 매우 비효율적이다[1-5]. 또한 스트림 데이터를 생성하는 응용 분야의 특성상 기존의 DBMS에서 지원하는 질의보다는 연속 질의(continuous query)가 적합하다. 연속 질의는 기존 DBMS 기반의 스탠딩 질의(standing query)와는 많은 차이점이 있다. 스탠딩 질의는 현재 저장된 데이터에 대한 일회성 질의(one-time query)인 반면 연속 질의는 일정 기간 동안 지속적으로 사용될 수 있는 질의를 뜻한다. 모니터링이나 센서 네트워크의 스트림 데이터들은 연속해서 발생하며 이런 데이터로부터 사용자가 얻고자 하는 정보는 계속 변화하는 것이 아니라 고정적인 것이 보통이다. 예를 들어 방안의 온도를 점검하는 센서가 있다면 주기적으로 방안의 온도를 재고 이를 연속해서 스트림 데이터 처리 시스템으로 전송할 것이다. 이때 사용자가 스트림 데이터 처리 시스템을 통해서 얻고자 하는 것은 특정 온도 이상일 때 경고를 해주거나 에어컨을 가동하라는 메시지를 사용자에게

※ 본 논문은 한국과학재단 특정기초연구사업(R01-2003-000-10395-0) 지원으로 수행되었음.
† 준 회 원 : 중앙대학교 대학원 컴퓨터공학부
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
논문접수 : 2004년 7월 16일, 심사완료 : 2004년 8월 26일

알리는 것이다. 즉, 스트림 데이터 처리 시스템에는 온도가 20도 이상일 경우 경고음을 울리라는 질의를 등록하며 시스템은 연속적으로 발생하는 온도에 대한 스트림 정보를 처리하여 사용자에게 결과 값을 전달하는 것이다. 이와 같이 연속 질의의 특징은 미리 등록된 질의(pre-defined query)라는 점과 일정 기간 동안 계속해서 사용한다는 점이다.

본 논문에서 다루는 연속 질의는 XQuery에 시구간을 덧붙인 형태로 표현한다. XQuery는 XML 스트림 데이터에 대한 질의이고 시구간은 연속적으로 발생하는 스트림 데이터에 대해서 질의를 적용할 구간을 나타낸다. 즉, XML 스트림 데이터가 발생한 시간에 따라서 연속 질의의 시구간에 해당하는 XML 스트림 데이터만을 질의 대상으로 한다. 또한 연속 질의 처리를 위해서 본 논문에서는 백그라운 처리를 이용한 이벤트 라우팅 기법을 제안한다. XML 데이터에 대한 질의 처리를 위해서는 XML 문서의 파싱 과정이 필수적이다. 이벤트 라우팅 기법이란, XML 스트림 데이터에 대한 질의 처리를 위해서 SAX 파서를 이용하여 문서를 파싱하고 이때 발생한 이벤트들을 라우팅하여 질의와 매칭되는지를 검토하는 기법이다. 이것을 백그라운드로 처리하여 질의에 대한 결과 값을 스트림 데이터가 발생하는 즉시 처리하여 결과 값을 실체화해 두었다가 질의가 호출될 때 실체화되어 있는 값에 연속 질의의 시구간만을 적용하여 원하는 결과를 빠르게 산출한다. 이때 시구간을 적용하기 위하여 시스템은 타임스탬프를 이용한다. XML 스트림 데이터가 발생하면 발생한 시각의 시스템 타임스탬프를 XML 엘리먼트로 XML 스트림 문서에 삽입함으로써 연속 질의에 대한 시구간을 쉽게 적용할 수 있도록 한다.

2장에서는 스트림 데이터 시스템에 관한 본 논문과의 관련연구를 기술한다. 3장에서는 본 논문에서 제안하는 XML 스트림 데이터 처리 시스템 모델에 대해서 기술한다. 4장에서는 3장에서 제안한 시스템을 구현하고 그에 대한 성능을 평가한 결과를 기술한다. 5장에서는 결론을 맺고 향후 연구 내용을 기술한다.

2. 관련 연구

기존의 스트림 데이터 처리 시스템의 데이터 모델은 대부분 관계 데이터 모델을 기반으로 한 것이다[1-5]. 본 논문에서는 이를 기반으로 관계형 데이터를 XML 데이터 모델로 확장하여 질의 스트림에 적용시키는 데 있어 해결해야 할 기술 요소들을 파악하고 시스템을 개발하여 성능을 평가한다.

최근 스트림 데이터를 처리하기 위한 다양한 연속 질의 엔진(continuous query engine)이 연구되고 있다[1-5]. 그 중 본 연구의 기반이 되는 시스템들에는 센서 네트워크 상에서 적응력 있는 스트림 처리를 위한 버클리 대학의 TelegraphCQ [1-2], 범용적인 목적의 스트림 데이터 처리 시스템인 스탠포드 대학의 Stream[3], 모니터링 응용을 위한 스트림 데이터 처리 시스템인 Aurora[4], XML 스트림 데이터 처리를 위한 Tukwila[7] 등이 있다. 또한 오토마타 기법을 이용한 XML 스트림 데이터에 대한 질의 처리 기법으로 XFilter[8], YFilter[9], XPush Machine[10], XSQ[11] 등이 있다.

TelegraphCQ는 스트림 데이터 모델로는 관계형 데이터 모델을 사용하였고 연속 질의 모델로는 SQL에 시구간을 추가하여 확장한 형태의 질의 모델을 사용하였다[1-2]. 또한 데이터와 질의를 모두 스트림 데이터의 형태로 처리하기 위한 자료구조인 SteMs(State Module)를 제안하고 있다. 각각의 SteMs에 데이터 또는 질의 스트림을 저장하기 위해 연속적으로 들어오는 관계형 스트림 데이터를 라우팅하기 위한 Eddy System 또한 TelegraphCQ의 핵심 기술이다[6]. Eddy 시스템은 스트림 데이터로 들어오는 관계형 데이터들을 라우팅하여 미리 등록된 질의들을 만족하는지를 검토하는 시스템이다. 시스템으로 전달된 각 데이터들은 라우팅 정책에 따라서 라우팅되어 등록된 질의를 만족하는지 여부를 검토하여 질의 결과 값을 산출한다. Eddy 시스템의 가장 큰 특징은 여러 개의 질의 처리를 한 번에 수행한다는 것이다. 즉, 하나의 데이터가 등록된 질의들로 한번만 라우팅 되어 최종 결과 값을 산출한다. 또한, TelegraphCQ는 생성된 질의 결과 값을 실체화하여 저장하여 후속 질의 결과에 반영함으로써 빠른 응답시간을 보장한다. Aurora 시스템은 Brown, MIT 등의 대학이 모여서 개발한 스트림 데이터 처리 시스템으로서 가장 큰 특징으로는 QoS를 보장하며 연속 질의와 과거 데이터에 대한 질의를 모두 지원한다는 점이다. Aurora 시스템은 QoS-Driven 방식으로 자원을 관리한다. 즉, 응용에 따라서 원하는 QoS를 등록하여 서비스의 질을 보장한다. 시스템은 등록된 QoS에 따라서 스트림 데이터의 부하 폐기(load shedding)를 통하여 적응력 있는 데이터 처리를 보장한다. 부하 폐기란 모든 스트림 데이터를 처리할 수 없을 때 임의로 스트림 데이터를 버리거나 의미 없는 스트림 데이터를 버리는 것을 뜻한다. 또한 Aurora 시스템은 질의 처리를 위해 connection point를 이용한다. 여러 개의 질의가 등록될 경우 질의 처리 과정 중 공유할 수 있는 부분까지는 공유하고 공유할 수 없는 부분을 connection point를 통하여 분리할 수 있다. 또한 connection point에서는 과거의 데이터를 유지함으로써 다른 스트림 데이터 시스템과는 달리 과거의 데이터에 대한 질의(historical query)를 지원한다[4]. Tukwila는 데이터 통합 시스템으로서 적응력 있는 데이터 처리를 위해 처음 개발되었다. 그 후 XML 스트림 데이터를 처리하기 위한 X-Scan 연산자가 추가되었다. X-Scan 연산자란 오토마타 기법을 이용하여 XML 스트림 데이터가 발생하는 즉시 데이터를 처리하여 결과 값을 반환해 주는 것이다[7]. 오토마타 기법을 이용한 XML 질의 처리 기법으로는 간단한 XPath 질의만을 처리하는 XFilter [8], YFilter[9]가 연구되었고 그 후 이를 확장하여 predicate

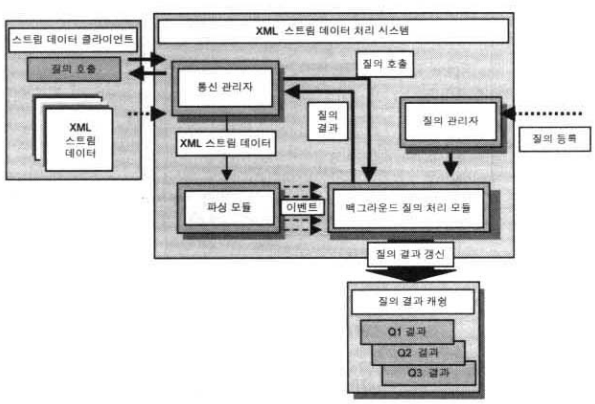
에 대한 처리를 지원하는 XPush Machine[10], XSQ[11] 등이 연구되었다.

본 논문에서 제안하는 시스템과 기존의 시스템의 가장 큰 차이점은 XML을 스트림 데이터 모델로 사용한다는 점과 질의 결과의 점진적 갱신 및 실체화 기법을 이용한다는 점이다. 기존의 시스템들이 주로 관계형 데이터를 데이터 모델로 사용한 데 반해 본 시스템은 XML을 데이터 모델로 사용하여 모니터링, 센서 네트워크 등 여러 가지 XML 데이터에 대한 스트림 응용 분야에서 발생하는 XML 스트림 데이터의 처리를 가능하게 한다. 기존의 XML 스트림 데이터 처리 시스템인 XFilter, YFilter, XPush Machine, XSQ 등과의 본 논문에서 제시한 시스템과의 차이점은 다음과 같다. XFilter는 단순한 오토마타 기법을 이용하여 XML 스트림 데이터에 대한 질의 처리를 하였고, YFilter는 이를 확장하여 질의의 중복되는 하위 경로를 공유하여 보다 빠른 질의 처리를 지원하였다. XPush와 XSQ는 기존의 XFilter와 YFilter에서의 단순한 path 질의 처리에 predicate를 지원하도록 확장하였다. XPush에서는 스택을 이용하여 predicate의 중복된 하위 경로를 공유하는 기법을 사용하였으며 XSQ는 버퍼링을 이용하여 여러 개의 predicate를 가지는 복잡한 질의를 지원하였다. 이와 같이 기존의 시스템들은 오토마타 기법을 이용한 현시점의 XML 스트림 데이터에 대한 질의 처리만을 지원할 뿐 연속 질의에 대한 처리가 불가능하다. 이에 반해 본 논문에서 제안하는 시스템은 Eddy 시스템과 같은 이벤트 라우팅 기법으로 모든 이벤트가 등록된 질의 각각에 대하여 처리된다는 장점이 있다. 또한 연속 질의를 지원하기 위하여 질의 처리 결과값을 실체화하여 저장하고 새로운 XML 데이터가 발생하면 이에 따른 질의 결과를 실체화된 결과값에 점진적으로 갱신하는 기법을 이용하여 변화된 시구간에 대한 연속 질의를 효율적으로 지원한다.

3. XML 스트림 데이터에 대한 연속 질의 처리 시스템 모델

본 장에서는 본 논문에서 제안한 XML 스트림 데이터 처리 시스템의 모델과 시스템 동작 과정을 설명하고 각각의 모듈이 사용하고 있는 기법과 자료구조 등에 대해서 설명한다. (그림 1)은 XML 스트림 데이터에 대한 연속 질의 처리 시스템 모델을 나타낸다. XML 스트림 데이터 처리 시스템은 크게 다섯 모듈로 구성된다. 각각은 통신 관리자(Communication Manager), 질의 관리자(Query Manager), 문서 관리자(Document Manager), 파싱 모듈(Parsing Module) 그리고 백그라운드 질의 처리 모듈(Background Query Processing Module)이다.

통신 관리자는 스트림 데이터 클라이언트와 통신을 하는 역할을 수행한다. 즉, 스트림 클라이언트로부터 XML 스트림 데이터를 수신하고 연속 질의를 등록 받는다. 또한 클라이언트가 질의를 호출할 경우 해당 질의에 대한 결과를 클라이언트로 전송하는 역할을 수행한다. XML 스트림 데이터는 스트림 데이터 처리의 특성상 데이터가 발생한 시간과 데이터들 간의 순서가 매우 중요하기 때문에 각각의 문서에 시스템으로 들어온 시간을 표시하여야 한다. 문서 관리자가 이런 역할을 담당한다. 문서 관리자는 스트림 클라이언트로부터 받은 XML 스트림 데이터에 시스템 타임스탬프를 이용하여 시간을 부여한다. 각각의 XML 스트림 문서는 발생한 시간 순서에 따라 현재 시간을 타임스탬프 태그로 묶어 XML 엘리먼트로 문서의 마지막에 삽입한다. 질의 관리자는 등록된 질의를 관리하는 역할을 한다. 본 시스템은 연속 질의를 사용하기 때문에 질의를 미리 등록하고 클라이언트에서 결과를 얻고 싶을 때 질의를 호출하여 해당 질의에 대한 결과 값을 얻어낸다. 질의 관리자는 효율적으로 질의 관리와 질의 처리를 위해서 QS(Query Structure)라는 특별한 자료구조를 사용한다. 등록된 질의는 각각 QS에 저장되고 질의 관리자는 실질적으로 QS의 집합을 관리하고 XML 스트림 데이터와 QS를 매칭시키는 역할을 수행한다.



(그림 1) XML 스트림 데이터에 대한 연속 질의 처리 시스템 모델

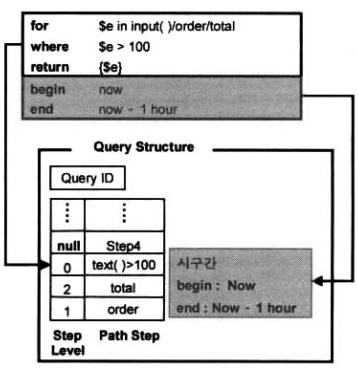
일단 질의가 등록되고 XML 스트림 데이터를 수신하면 수신된 데이터는 통신 관리자를 통해서 파싱 모듈로 넘겨진다. 파싱 모듈에서는 XML 스트림 데이터를 SAX 파서를 이용해 파싱하여 문서에 대한 정보를 이벤트 형태로 큐에 저장한다. 파싱 모듈에서 이벤트 큐에 이벤트들을 삽입하기 시작하면 백그라운드 질의 처리 모듈은 이벤트들을 하나씩 큐에서 꺼내 등록된 질의와 매칭 여부를 검사한다. 데이터와 질의가 매칭될 경우는 해당 질의에 대한 결과를 실체화하여 유지하였다가 이후 시구간 질의가 호출되었을 때 빠른 응답시간을 보장할 수 있도록 한다.

이후 본 절의 구성은 다음과 같다. 3.1절에서는 본 시스템에서 제안하는 연속 질의 모델에 대해서 기술하고 3.2절에서는 파싱 모듈과 백그라운드 질의 처리 모듈을 통한 질의 처리 기법인 이벤트 파싱 기법과 라우팅 마스크에 대하여

기술한다. 그리고 마지막으로 3.3절에서는 질의 결과의 실체화 및 점진적 갱신 기법을 통한 효율적인 질의 처리 기법에 대해서 기술한다.

3.1 질의 모델

XML 스트림 데이터에 대한 연속 질의 처리 시스템은 연속 질의를 처리하기 위해서 (그림 2)와 같은 QS 자료구조를 사용한다. 연속 질의는 일정 생명주기 동안 연속적으로 사용되어야 하기 때문에 시스템에 미리 정의되어야 하며 질의 대상 구간에 따라 landmark 질의와 sliding window 질의로 나눌 수 있다[1]. 즉, [1시, 현재]와 같은 기준 시간으로부터 변화하는 값을 구간으로 갖는 landmark 질의와 [현재부터 한시간 전, 현재]와 같이 구간의 양끝이 시간의 변화에 따라 함께 변화하는 sliding window 질의를 연속 질의의 예로 들 수 있다. 본 시스템에서 제안하는 연속 질의 모델은 (그림 2)와 같이 XQuery의 FLWR 구문에 음영 표시 부분의 begin-end 시구간 구문을 추가한 형태이다.



(그림 2) QS 자료구조

연속 질의 처리는 XQuery의 간단한 path 질의를 for절에서 바인딩하고 where절에서 predicate를 설정하도록 하는 가장 기본적인 질의만을 우선 고려하였다. 시구간은 begin-end 절을 이용하여 연속 질의를 지원하도록 한다. 질의 관리자는 (그림 2)와 같이 연속 질의가 들어오면 QS를 생성하고 for절과 where 절을 이용하여 QS에 XQuery를 바인딩 한다. 또한 시구간은 별도로 저장하여 타임스탬프의 적용이 용이하도록 한다. QS는 질의 식별자(Query ID), path 정보(step level, path step), 시구간으로 구성된다. 질의 식별자는 각각의 등록된 질의를 식별하기 위한 식별자로 유일한 값이 부여된다. for절과 where절의 path 정보는 각 step의 루트로부터의 level을 나타내는 step level과 각 step의 path를 나타내는 path step의 두 가지로 표현된다. path step은 XML 문서 내의 path 정보와 매칭되어 질의를 만족하는 결과 값을 얻어내는 데 사용되며 step level은 각각의 path step이 문서 내에서 루트로부터의 몇 번째 자식인지를 나타낸다. 단 텍스트 엘리먼트는 0으로 표시한다. 이는 효율적인 질의 처리를 위해서 이벤트를 필터링하기 위한 라우팅 마스크를 셋팅하는 데 사용된다.

시구간의 begin 절에는 질의 대상 데이터에 대한 구간의 시작 시간을, end 절에는 끝마침 시간을 저장한다. 시구간은 스트림 클라이어트가 질의를 호출할 경우 해당 질의에 대한 실체화된 결과 값에 적용되어 클라이언트가 원하는 구간의 결과만을 추출하는 데 사용된다. 결과 값의 실체화에 대해서는 3.4절에서 기술한다. 연속 질의를 위와 같이 QS에 분할 저장하는 것은 백그라운드 질의 처리를 효율적으로 지원하기 위한 것이다. QS는 질의를 저장하기 위한 자료구조일 뿐만 아니라 효율적인 질의 처리를 위한 XML 문서의 엘리먼트와 질의의 매칭 여부를 쉽게 알 수 있도록 질의를 유지 관리하는 자료구조이다. 질의 관리자는 등록된 질의의 수만큼의 QS를 유지하며 하나의 XML 엘리먼트가 등록된 모든 질의에 대해 타당한지를 효율적으로 검사할 수 있도록 한다.

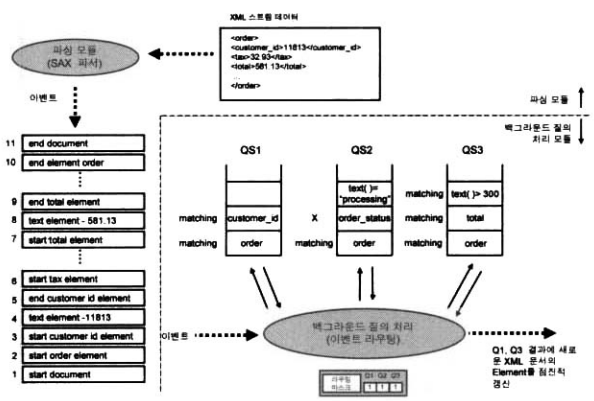
3.2 질의 처리

(그림 3)은 백그라운드 질의 처리 과정을 나타낸 것이다. 스트림 데이터는 연속적으로 끊임 없이 발생한다. 이런 특성을 갖는 스트림 데이터를 원활하게 처리하기 위해서는 기존의 DBMS처럼 데이터를 모두 저장하여 처리하는 것은 저장공간 효율과 질의 처리 효율 어느 측면에서도 좋은 접근 방법이 아니다. 백그라운드 질의 처리 모듈은 XML 스트림 데이터를 처리하기 위해 데이터가 들어오는 대로 원하는 정보만을 뽑아내 저장하고 쓸모없는 데이터는 버리는 역할을 수행한다. 질의 처리는 빠른 응답시간을 위해서 백그라운드로 수행될 뿐만 아니라 이벤트 라우팅 기법과 라우팅 마스크를 이용한 필터링 기법을 사용한다. 라우팅 마스크는 이벤트를 라우팅하여 질의를 처리할 때 좀 더 효율적으로 처리하기 위해서 불필요한 라우팅을 하지 않게 필터링하는 기능을 수행한다.

이벤트 라우팅 기법은 파싱 모듈에서 파싱된 XML 스트림 데이터에 대한 이벤트를 이용해 등록된 질의를 처리하여 결과 값을 얻는 과정이다. (그림 3)은 웹상의 전자상거래 벤치마킹 문서인 TPC-W[14]의 order문서를 XML 스트림 데이터로 백그라운드 질의 처리를 하는 과정을 예로 들어 나타낸 것인데, 질의 처리를 위해서 파싱 모듈에서 생성된 이벤트들을 QS로 라우팅하여 질의 결과 값을 산출하는 과정이다. <표 1>과 같은 세 가지 XQuery가 연속 질의로 등록되어 있다고 가정하자. 이에 따라 시스템은 세 개의 QS를 생성하고 XQuery의 for절과 where절의 path를 QS의 path step에 저장한다. 라우팅 마스크는 등록된 질의의 개수만큼의 정수값을 유지한다. 각각의 마스크는 해당 질의가 다음에 라우팅되어서 들어올 이벤트의 step level을 가리킨다. 즉, 라우팅 마스크가 나타내고 있는 step level과 이벤트의 루트로부터의 level 값이 일치할 경우만 이벤트를 해당 QS

로 라우팅 하여 매칭 여부를 확인하고 그렇지 않은 경우는 필터링하여 다음 QS로 라우팅한다. 질의 처리 과정은 다음과 같다.

① XML 스트림 데이터의 파싱된 이벤트를 가져 온다.
② 가져온 이벤트의 루트로부터의 level과 등록된 질의의 라우팅 마스크가 가리키는 step level이 일치할 경우 일치하는 QS로 이벤트를 라우팅한다. 일치하지 않을 경우 다음 라우팅 마스크에 대해 ②번 과정을 다시 수행한다.
③ 라우팅 마스크를 통과하여 QS로 라우팅된 이벤트는 각각의 path step과 이벤트가 매칭되는지를 검사한다. 매칭될 경우 마지막 path step이 아니면 라우팅 마스크를 다음 path step의 step level로 셋팅한다. 마지막 path step이 매칭되었을 경우, 해당 엘리먼트의 end 태그에 대한 이벤트가 들어올 때까지 이벤트들을 XML 엘리먼트 형태로 결과 값으로 저장한다. 매칭되지 않을 경우 다음 QS의 라우팅 마스크로 라우팅하여 ②번 과정부터 다시 수행한다.
③ QS의 마지막 path step과 매칭되어 최종 결과가 산출될 경우 결과 값을 실체화하고 더 이상 같은 문서에 대하여 이벤트를 라우팅 받지 않기 위해서 end document 이벤트가 들어올 때까지 해당 QS의 라우팅 마스크를 null로 유지한다.



(그림 3) 백그라운드 질의 처리 과정

〈표 1〉 (그림 3)의 QS에 등록된 XQuery

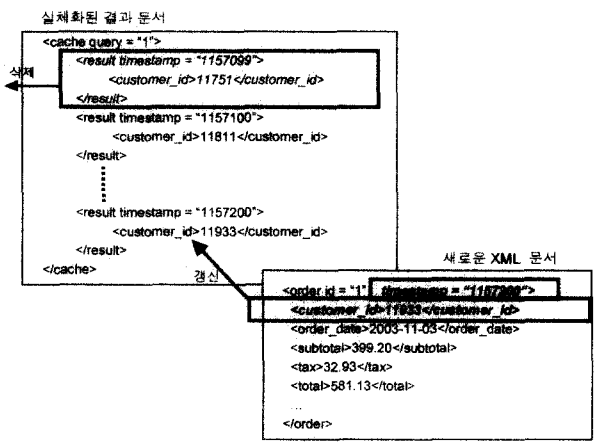
QS	XQuery
QS1	For $e in input()/order/order_id Return {$e}
QS2	For $e in input()/order/order_status Where $e = "processing" Return {$e}
QS3	For $e in input()/order/total Where $e > 300 Return {$e}

예를 들어, (그림 3)의 order XML 문서 스트림에 대한 질의 처리 과정은 다음과 같다. 새로운 문서가 파싱 모듈을 통해서 파싱되어 이벤트 큐에 이벤트들이 순차적으로 쌓이게 된다((그림 3)의 왼쪽 1번에서 11번 까지의 내용). 처음 발생하는 이벤트는 [1.start document] 이벤트로서 문서의 시작을 알린다. [1.start document] 이벤트가 들어오면 라우팅 마스크는 모두 1로 셋팅되어 루트 엘리먼트가 들어오기를 기다린다. 두 번째 이벤트인 [2.start order element]가 들어오면 order 엘리먼트는 루트 엘리먼트이기 때문에 라우팅 마스크의 값이 1인 QS로 라우팅을 시작한다. 3개의 라우팅 마스크 모두 루트 엘리먼트를 기다리고 있기 때문에 QS1부터 QS3까지 차례로 라우팅되어 매칭된다. 매칭된 QS에 대해서는 두 번째 path step의 step level로 라우팅 마스크를 셋팅한다. 즉, 다음 step이 모두 루트 엘리먼트의 자식을 기다리고 있기 때문에 모두 2로 셋팅된다. 이어서 들어오는 [3.start customer id element] 이벤트는 QS1로 라우팅 되어 매칭되고 QS2와 QS3과는 매칭되지 않고 버려진다. QS1은 마지막 path step과 매칭되었기 때문에 들어온 문서에 대해서 1번 질의에 대해서 만족하므로 라우팅 마스크를 null로 셋팅하고 3번 이벤트에 대한 end element 이벤트가 들어올 때까지의 이벤트들을 결과 엘리먼트로 실체화한다. 즉, <customer_id>11812</coustomer_id>의 값이 결과 값으로 실체화된다. 3번 이벤트가 종료된 후의 라우팅 마스크 QS1은 null, QS2와 QS3은 2를 가리키므로 4번~6번 이벤트는 라우팅 마스크에 의해서 필터링 되거나 QS로 라우팅 되어 매칭되지 않고 버려진다. [7.start total element]가 들어오면 QS3이 매칭되고 텍스트 엘리먼트를 가리키는 0으로 라우팅 마스크가 셋팅된다. 다음 [8.text element - 581.13]이 들어오면 텍스트 엘리먼트이기 때문에 QS3으로 라우팅 되고 해당 질의의 조건대로 300보다 크기 때문에(<표 1>의 QS3의 where절 참조) QS3의 마지막 step level을 만족하고 결과 값으로 실체화된다. [11.end document] 이벤트가 들어오면 문서에 대한 백그라운드 질의 처리가 모두 끝나고 실체화된 결과를 메모리에 저장된 실체화 문서의 마지막에 삽입한다. 이와 같이 질의 처리 과정은 이벤트 라우팅을 백그라운드로 처리하여 이루어진다. 또한 라우팅 마스크를 사용하여 불필요한 라우팅을 줄이고 매칭 가능성이 있는 질의만을 검사하여 질의 처리를 효과적으로 수행한다.

3.3 질의 결과의 실체화 및 점진적 갱신

백그라운드 질의 처리로 산출된 각 질의에 대한 결과 값은 질의 식별자를 이용하여 각각의 질의 별로 분류되어서 XML 문서 형태로 메모리에 저장된다. 질의 결과 값은 (그림 4)와 같이 해당 문서가 발생한 시간의 타임스탬프를 속성으로 묶어서 XML 엘리먼트로 실체화된다. 이때 후속 데이터에 대한 결과 값은 계속해서 생성되며 이것은 결과 문서의 마지막에 엘리먼트로 삽입되어 실체화된 문서가 점진

적으로 갱신되는 효과를 가져온다. 실체화된 문서의 엘리먼트들은 자연히 타임스탬프 순서대로 배치되며 이것은 질의 호출시 시구간의 적용을 용이하게 한다. 실체화된 결과 값은 해당 질의가 호출 될 경우 시구간에 반영하여 실제 질의에 대한 결과 값으로 반환된다. 즉, 현재 시간의 타임스탬프를 적용하여 시구간을 정하고 실체화된 결과 엘리먼트들 중에 해당 시구간 안에 있는 엘리먼트들만을 결과 값으로 선택하여 반환한다. 결과 값을 질의 결과로 반환하면 반환된 값은 다시 결과 문서로 실체화된다. 즉, 해당 시구간에 반영되지 않은 엘리먼트들은 모두 삭제되고 시구간 안에 존재하는 엘리먼트들만 결과 값으로 다시 실체화된다. 결과 문서의 실체화 및 점진적인 갱신 기법은 질의 수행을 위해 무한한 양의 스트림 데이터를 모두 저장하는 것이 아니라 질의 처리에 필요한 데이터들만을 일정 시구간 동안 유지함으로써 스트림 데이터에 대한 연속 질의를 효율적으로 처리한다.



(그림 4) 질의 결과의 점진적 갱신

4. 구현 및 성능 평가

본 장에서는 3장에서 제시한 기법들을 기반으로 XML 스트림 데이터에 대한 연속 질의 처리 시스템을 구현하고 성능을 평가하여 본 논문에서 제시한 기법들이 스트림 데이터 처리에 효율적인 접근 방법임을 보인다. 구현은 크게 XML 스트림 데이터를 처리하는 XML 스트림 데이터 처리 시스템과 XML 스트림 데이터를 전송하는 XML 스트림 클라이언트의 두 부분으로 나누어 수행하였다. 후자는 실험을 위하여 구현된 것인데 이에 대해서 먼저 간단히 소개한다.

```
<? xml version = "1.0" encoding = "UTF-8"?> <!--generated by ToXgeneVersion
1.1a on Mon Dec 01 14 : 21 : 32 KST 2003 →
<order id = "1">
<customer_id> 11813 </customer_id>
<order_date> 2003-11-03 </order_date>
<subtotal> 399.20 </subtotal>
<tax> 32.93 </tax>
<total> 581.13 </total>
<ship_type> AIR </ship_type>
<ship_date> 2003-11-04 </ship_date>
<bill_address_id> 39187 </bill_address_id>
<ship_address_id> 3522 </ship_address_id>
<order_status >PROCESSING </order_status>
<credit_card_transaction>
<credit_card_type> MASTERCARD </credit_card_type>
<credit_card_number> 1508275354460218 </credit_card_number>
<name_on_credit_card> dugouts impress sin </name_on_credit_card>
<expiration_date> 2005-02-22 </expiration_date>
<authorization_id> APv4Jpq - ?!p#Yf </authorization_id>
<transaction_amount> 581.13 </transaction_amount>
<authorization_date> 2003-11-04 </authorization_date>
<transaction_country_id> 75 </transaction_country_id>
</credit_card_transaction>
<order_lines>
<order_line id = "1">
<item_id> 6288 </item_id>
<quantity_of_item> 102 </quantity_of_item>
<discount_rate> 0.00 </discount_rate>
<special_instructions> {Pqn$O31K$5!Nq9arEFF}2, G ~ (:_PmWb6Jz1(Q ~ {2#Nt+}
GuCm$U</special_instructions>
</order_line>
<order_line id = "2">
<item_id> 574</item_id>
<quantity_of_item> 44 </quantity_of_item>
<discount_rate> 0.02 </discount_rate>
<special_instructions> r : U_/mnzFj8|taaMYHu4 : e{UQemg=L/9xj*}3 </special_in-
structions>
</order_line>
</order_lines>
</order>
```

(그림 5) TPC-W의 Order 문서

4.1 XML 스트림 데이터 클라이언트

XML 스트림 데이터 클라이언트는 실험을 위한 것으로서 주기적으로 시스템에 XML 스트림 데이터를 전송한다. 스트림 클라이언트는 XML 스트림 데이터 유입률(XML stream data arrival rate)과 실행 시간 또는 전송할 문서 개수를 파라미터로 입력받아 해당 실행 시간 또는 전송할 문서 개수만큼의 문서를 하나씩 읽어 들여 데이터 유입률에 따라서 전송한다. 100개의 데이터를 0.5초의 유입률로 전송한다면 0.5초에 문서 1개씩을 전송하는 것을 뜻한다. 즉 50초 동안 100개의 문서를 XML 스트림 데이터 처리 시스템으로 전송한다. 이는 XML 스트림 데이터가 데이터 유입률에 따라서 얼마나 적응력있게 데이터를 처리할 수 있는 가에 대한 실험을 위한 것이다.

4.2 구현 및 실험 환경

XML 스트림 데이터 처리 시스템과 스트림 클라이언트는 모두 JDK1.4.2 를 사용하는 JAVA 환경에서 구현되었다. 파싱 모듈에서 사용한 SAX 파서는 apache에서 제공하는 SAX 파서를 이용하였다[15]. 실험 환경은 다음과 같다. XML 스트림 데이터 처리 시스템은 1.13GHz CPU 2개를 사용하고 메모리는 2304MB인 Windows 2000 Server 시스템에서 실행하였고 스트림 클라이언트는 1.8GHz CPU와 768MB 메모리

를 사용하는 Windows 2000 Server 시스템에서 실행하였다. 실험 데이터로는 TPC-W[14]에서 제공하는 order 문서를 사용하였다. order 문서는 XBench[12-13]에서 사용한 XML 문서 생성기를 이용하여 생성하였다. 25000개의 order 문서를 생성하여 실험에 사용하였으며 각각의 order 문서의 크기는 대략 2KB 정도이다. order 문서는 웹상에서의 전자상거래 주문 문서로서 (그림 5)와 같이 상품 주문에 대한 자세한 정보를 포함하고 있다.

실험은 흔히 백화점, 체인점 또는 쇼핑몰 등의 분산된 구매환경에서 발생한 주문 문서에 대하여 중앙 부서에서 각종 통계 자료를 뽑는 것을 도메인으로 하였다. 즉, 분산된 환경에서 매초 발생하는 주문에 대한 정보를 XML 스트림 데이터로 생성하여 중앙 부서의 XML 스트림 데이터 처리 시스템으로 전송하여 통계자료 또는 주문 현황을 파악하는 것을 가정하고 실험하였다. 예를 들어, XML 스트림 데이터 처리 시스템을 이용하여 시구간에 따라서 발생한 주문 문서를 취합하여 시간 때 별로 고객층, 상품 선호도 등을 모니터링 하거나 전체 판매량 또는 한달간의 판매량 등을 질의로 등록하여 구매 정보를 분석할 수 있다.

실험에 사용된 연속 질의의 경로 집합으로는 <표 2>과 같은 path 경로들을 사용하였다. 모든 문서에 존재하는 order_date를 뽑는 Q1을 비롯하여 order 문서에 대해서 다양한 질의가 처리되도록 하였다.

4.3 실험 결과

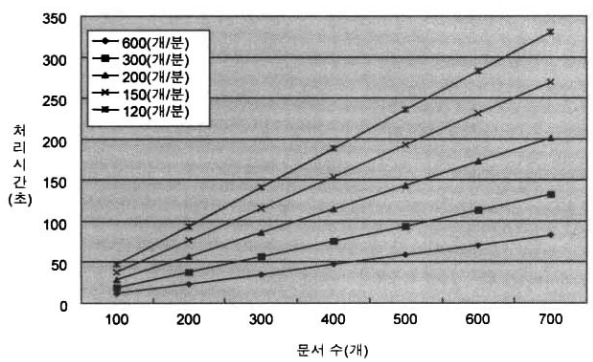
실험에서는 스트림 클라이언트에서 order 문서를 하나씩 읽어 데이터 유입률에 따라서 XML 스트림 데이터 처리 시스템으로 전송하고 시스템은 <표 2>의 질의 경로를 등록하여 데이터를 처리하여 실체화하는 데 걸린 시간, 스트림 클라이언트의 질의 요청에 응답하는 시간 등의 주요 XML 스트림 처리 시스템의 성능을 평가할 수 있는 요인들을 측정하였다.

〈표 2〉 실험에 사용된 질의 경로

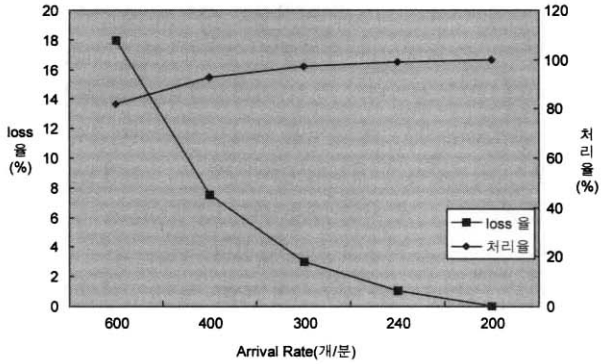
질 의	질의 경로
Q1	order/order_date
Q2	order/total/text( ) > 25.3
Q3	order/subtotal/text( ) < 250.33
Q4	order/tax/text( ) > 30.55
Q5	order/transaction_amount/text( ) > 600.5
Q6	order/authorization_date
Q7	order/transaction_country_id/text( ) > 30
Q8	order/credit_card_transaction/credit_card_type
Q9	order/ship_type

4.3.1 스트림 데이터 유입률에 따른 XML 문서 처리

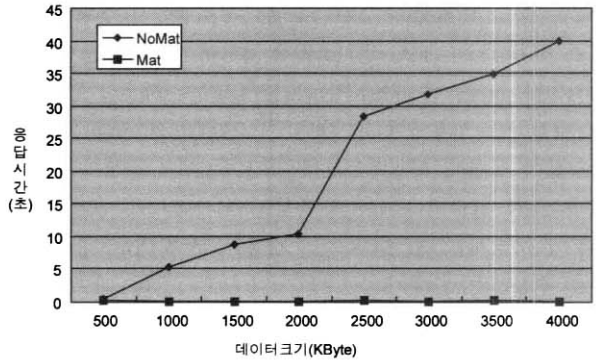
XML 스트림 데이터 처리 시스템은 끊임없이 발생하는 XML 문서를 처리하기 위한 시스템이기 때문에 문서의 발생빈도에 따른 문서 처리율과 처리량이 성능에 중요한 요인이 된다. (그림 6)은 데이터 유입률의 증가에 따른 XML 스트림 데이터 처리 시스템의 문서 처리시간을 나타낸 것이다. 데이터 유입률은 0.1초에 문서를 하나씩 전송하여 1분에 600개의 문서를 전송하는 것부터 0.05초씩 시간 간격을 증가시켜 0.3초에 하나의 문서를 전송하여 1분에 200개의 문서를 전송하는 유입률을 사용하였다. 데이터 유입률의 증가에 따라서 XML 스트림 처리 시스템의 XML 문서 처리량도 증가하는 것을 볼 수 있다. 그러나 데이터 유입률이 과도하게 증가하면 문서 처리 속도보다 문서의 유입 속도가 빨라지며 결과적으로 스트림 클라이언트 측에서 전송했으나 시스템 측에서 처리하지 못하는 문제점이 발생한다. (그림 7)의 그래프는 XML 스트림 처리 시스템의 스트림 처리 한계를 알아보기 위한 것으로 각각의 데이터 유입률에 따라서 문서의 처리율을 나타내었다. 왼쪽의 Y축은 데이터 loss율로 스트림 클라이언트에서는 전송했으나 XML스트림 처리 시스템에서는 처리하지 못한 문서의 비율을 나타낸다. 오른쪽 Y축은 스트림 클라이언트에서 전송한 문서 중 XML 스트림 처리 시스템에서 처리된 문서의 비율을 나타낸다. 그래프를 보면 분당 600개의 문서를 전송하였을 경우 데이터의 loss율은 대략 18% 정도이고 처리율은 82% 정도이다. 즉, 스트림 클라이언트에서 초당 10개의 문서를 전송하였을 때 XML 스트림 데이터 처리 시스템은 1분에 492개의 문서를 처리하고 나머지 108개의 문서는 처리하지 못하는 문제점이 발생한다. 데이터 유입률을 감소시킬수록 loss율은 낮아지고 처리율은 높아지는 그래프를 확인할 수 있다. 유입률이 200개/분이 되었을 때 즉 0.3초에 1개의 문서를 전송하여 1분에 200개의 문서를 전송할 경우 스트림 데이터에 대한 처리율은 100%가 되고 loss율은 0%가 된다. 200개/분 이하의 유입율을 사용할 경우 스트림 데이터 처리 시스템이 클라이언트가 보내는 모든 스트림 데이터를 처리할 수 있는 것이다. 그래서 후속 실험에서는 1초에 2개의 문서를 전송하는 120개/분의 유입율을 사용하여 실험하였다.



(그림 6) 데이터 유입률에 따른 문서 처리시간



(그림 7) 데이터 유입률에 따른 XML 스트림 데이터 처리율



(그림 8) 실체화에 따른 성능 향상

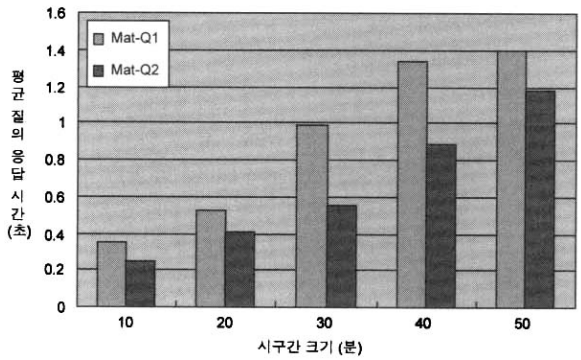
4.3.2 질의 결과의 실체화에 따른 응답시간

(그림 8)은 실체화에 따른 성능향상 그래프를 보여주고 있다. 질의 결과를 실체화하지 않는 시스템과 실체화하는 시스템을 사용하여 질의 응답시간의 차이를 측정하였다. 질의 결과를 실체화하는 것을 Mat, 실체화하지 않는 것을 No-Mat로 표기하였다. 즉, Mat 버전은 결과 값을 백그라운드 처리로 실체화하여 유지하고 있는 시스템이다. 반면 No-Mat 버전은 스트림 클라이언트로부터 수신한 스트림 데이타에 대하여 파싱 모듈에서 파싱만을 거친 상태로 이밴트를 저장한다. 따라서 Mat 버전의 응답시간은 질의 호출시 이미 백그라운드 프로세싱에 의해 생성된 결과 값에 현재 시스템 타임스탬프를 이용해 시구간을 적용하여 결과 값을 산출하는데 걸린 시간이 된다. 반면, No-Mat 버전의 경우 질의 호출시 현재 이밴트 큐에 쌓여있는 데이터들에 대해 질의 처리 과정을 거쳐 결과 값을 생성하고 시구간을 적용하여 최종 결과 값을 얻는 데 소요된 시간이 응답시간이 된다. 실험은 1초에 두개의 order문서를 스트림 클라이언트에서 전송하였으며 각 Mat 버전과 No-Mat 버전에 Q1을 등록하여 질의 응답시간을 측정하였다. 또한 Q1의 시구간 크기는 10분으로 동일하게 설정하여 수행되었다. 그래프를 보면 전송된 데이터의 크기가 클수록 결과를 실체화하지 않는 시스템에서는 응답시간이 급격히 증가하는 것을 볼 수 있다. 이것은 많은 양의 데이터에 대해서 질의를 처리하고 다시 시구간을 적용하여 결과를 뽑아내는 데 걸리는 소요 시간이 매우 크기 때문이다. 반면에 실체화하여 질의 결과를 미리 생성한 시스템은 전송된 데이터의 크기에 상관없이 결과를 백그라운 프로세싱으로 미리 생성하기 때문에 질의 호출시 0.5초 이내의 빠른 응답시간을 유지한다. 결과 값의 실체화는 빠른 응답속도를 보장하는 것 뿐만 아니라 데이터 크기에 상관없이 일정한 응답시간을 보장한다는 장점이 있다. 즉, 전송된 데이터의 양이 중요한 것이 아니라 등록된 질의의 시구간의 크기에 따라서 응답시간이 결정되는 것이다. 그러므로 본 논문에서 제안하는 실체화 기법은 스트림 데이터와 같이 무한한 양의 데이터에 대한 등록된 연속 질의를 처리하는 응용분야에 매우 적합함을 알 수 있다.

4.3.3 시구간 크기에 따른 질의 응답시간

시구간의 크기는 질의 응답시간을 결정하는 매우 중요한 요인으로 작용한다. 시구간 크기에 따른 질의 응답시간의 실험은 다음과 같이 수행되었다. 스트림 클라이언트에서 1초에 2개의 order 문서를 XML 스트림 데이터 처리 시스템으로 전송하며 XML 스트림 데이터 처리 시스템은 <표 2>의 Q1과 Q2를 등록하여 스트림 데이터를 처리하였다. 시구간의 크기는 10분부터 50분까지 증가시키며 실험하였다. 1초에 2개의 문서를 보낼 경우 스트림 데이터 처리 시스템에서는 1분에 120개의 문서를 수신하며 10분 동안은 1200개의 문서를 처리하여야 한다. 이것은 order문서의 한 개의 크기가 2Kbyte이기 때문에 2.4M 정도의 데이터가 된다. (그림 9)는 시구간 크기에 따른 질의 응답 속도를 보여주는 그래프이다. 그래프는 Q1과 Q2의 질의를 주기적으로 호출하여 결과 값을 얻는 데 걸린 시간의 평균값을 나타낸다. 그래프는 시구간의 크기가 증가할수록 질의 응답시간 또한 증가하고 있다. 이것은 시구간의 크기가 증가하면 시구간을 적용해야 하는 실체화한 질의 결과 데이터도 증가하기 때문이다. Q1과 Q2 중 Q1이 Q2보다 질의를 처리하는 성능이 좋은데 이것은 실체화된 결과 값의 크기에 따른 것이다. Q1의 질의는 order문서의 order_date를 모두 뽑는 것으로 oder_date 엘리먼트는 모든 문서에 존재하기 때문에 전송된 문서의 수만큼의 데이터가 실체화된다. 반면 질의 Q2는 필터연산을 이용하여 total 값이 25.3 이상인 엘리먼트만을 질의 결과로 실체화하기 때문에 실체화된 데이터의 크기가 상대적으로 Q1보다 적다. Q1과 Q2의 응답시간의 차이에서 나타나듯이 질의의 복잡도보다는 질의 결과물의 크기가 질의 응답시간에 더 많은 영향을 미친다. 질의 처리는 백그라운드 프로세싱으로 이루어지기 때문에 질의의 복잡도는 질의 응답시간에 크게 영향을 미치지 않는다. 그러나 질의 결과로 실체화된 데이터의 양은 질의 호출시 시구간을 적용하기 위해 하나씩 시구간 내에 있는지 여부를 확인해야 하기 때문에 실체화된 데이터 크기가 증가함에 따라 성능도 떨어지게 된다. 따라서 시구간의 크기가 증가하면 질의 결과로 유지하는 데이터의 크기도 증가하기 때문에 응답시간도 증가하는 결과

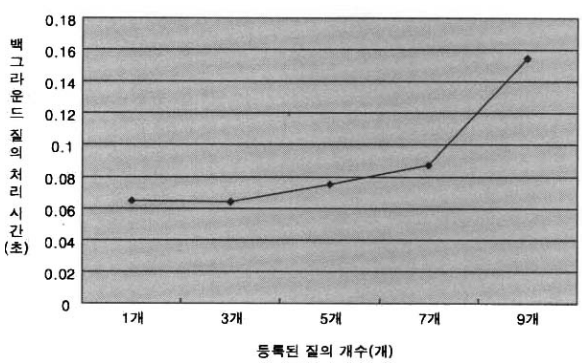
를 보인다.



(그림 9) 시구간 크기에 따른 질의 응답시간

4.3.4 등록된 질의 개수에 따른 질의 처리

질의 처리는 백그라운드 처리로 이루어지기 때문에 등록된 질의의 개수는 질의 처리의 성능에 큰 영향을 미친다. 실험은 다음과 같이 이루어졌다. 120개/분의 데이터 유입률로 1초에 두개의 문서를 스트림 클라이언트를 통해 전송하며 <표 2>의 질의 경로를 순서대로 등록하여 실험하였다. 즉, 질의 한 개의 등록 경우에는 Q1을, 질의 다섯 개의 등록 경우에는 Q1부터 Q5를 등록하여 실험하였다. 질의 처리 시간의 측정은 스트림 데이터로부터 받은 문서 하나가 파싱되어 등록된 모든 질의에 대한 처리를 끝마쳐 결과 값으로 실체화될 때까지의 처리 시간을 측정하였다. (그림 10)은 등록된 질의의 개수에 따른 백그라운드 질의 처리 속도를 나타내는 그래프이다. 세 개의 질의를 등록할 때 까지는 질의 처리에 별다른 성능 차이가 없었으나 그 이상 질의를 등록할 경우 질의 개수의 증가에 따라서 질의 처리 소요 시간도 증가하는 형태의 그래프를 보였다. 이것은 등록된 질의 개수가 증가할수록 하나의 이벤트를 처리하는 시간이 증가하기 때문에 전체적으로 질의를 처리하는 시간도 증가하기 때문이다. 또한 등록된 질의가 많아질수록 결과 값으로 실체화되는 데이터의 양도 증가하며 많은 양의 데이터를 실체화하기 위한 처리 시간 역시 증가한다.



(그림 10) 등록된 질의 개수에 따른 질의 처리 시간

5. 결 론

본 논문에서는 XML 스트림 데이터를 처리하기 위한 XML 스트림 데이터 처리 시스템 모델을 제안하고 개발하여 성능을 평가하였다. 또한 스트림 데이터 모델로 XML을 사용하였기 때문에 XML 스트림 데이터에 대한 연속 질의를 처리하기 위한 기법으로 SAX 파서를 이용한 엘리먼트 라우팅 기법을 사용하였고 질의 처리의 성능 향상을 위하여 질의 결과의 점진적 갱신을 이용한 실체화 기법을 제안하였다. 실험 결과에서 나타나듯이 XML 스트림 데이터를 처리하기 위해서는 기존의 DBMS와 같은 데이터 처리 기법으로는 적합하지 않다. 스트림 데이터의 특성상 모든 데이터를 저장하여 처리하는 것은 비효율적이며 좋지 않은 성능을 보인다. 따라서 이를 처리하기 위해서는 본 논문에서 제안하는 것과 같은 백그라운드 질의 처리를 이용한 실체화 기법이 필요하다. 본 논문에서는 XML 스트림 데이터에 대한 질의 처리를 백그라운 처리를 이용한 실체화 기법을 통해서 질의 호출 전에 수행함으로써 빠른 질의 응답시간을 보장하며 또한 전송된 데이터의 크기에 상관 없이 시구간의 크기에 따라서 일정한 응답시간을 보장한다.

향후 연구 과제는 다음과 같다. 아직 XML 스트림 데이터에 대한 연속 질의 처리를 위해 결과 값의 실체화 기법을 이용하는 스트림 처리 시스템에 대한 연구 사례가 없었다. 향후 본 논문의 시스템과 유사한 XML 스트림 데이터 처리 시스템에 대한 연구 결과가 발표되면 본 논문의 시스템과의 비교가 필요하다. 또한 다중 사용자 환경으로의 시스템 확장, 질의 처리 과정 중 불필요한 연산을 줄이는 최적화된 질의 처리 라우팅 기법, 디스크로 백업된 과거 스트림 데이터에 대한 질의 처리를 지원하기 기법들도 향후에 연구할 주제이다.

참 고 문 헌

[1] S. Madden, M. Shah, J. Hellerstein and V. Raman, "Continuously Adaptive Continuous Queries over Streams," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.49-60, 2002.

[2] S. Chandrasekaran and M. J. Franklin, "Streaming Queries Over Streaming Data," Proc. of Int'l Conf. on VLDB, 2002.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom, "Models and Issues in Data Stream Systems," Proc. of Symp. on PODS, 2002.

[4] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman., M. Stonebraker, N. Tatbul and S. Zdonik, "Monitoring Streams - A New Class of Data Management Applications," Proc. of Int'l Conf. on VLDB, 2002.

[5] J. Chen, D. J. DeWitt, F. Tian, Y. Wang, "NiagraCQ : A

Scalable Continuous Query System for Internet Database," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.379-390, 2002.

[6] R. Avnur and J. Hellerstein, "Eddies : Continuously Adaptive Query Processing," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.261-272, 2000.

[7] Z. G. Ives, D. Florescu, M. Friedman, A. Levy and D. S. Weld, "An Adaptive Query Execution System for Data Integration," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1999.

[8] M. Altinel and M. J. Franklin, "Efficient Filtering of XML Document for Selective Dissemination," Proc. of Int'l Conf. on VLDB, pp.53-64, 2000.

[9] Y. Diao, P. Fischer, M. J. Franklin and R. To, "YFilter : Efficient and Scalable Filtering of XML Documents," Proc. ICDE, 2002.

[10] A. K. Gupta and D. Suciu, "Stream Processing of XPath Queries with Predicates," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.419-430, 2003.

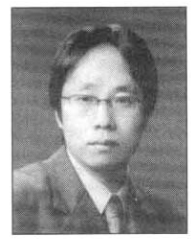
[11] F. Peng and S. S. Chawathe, "XPath Queries on Streaming Data," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.431-442, 2003.

[12] B. B. Yao, M. T. Ozsu, "Xbench-A Family of Benchmarks for XML DBMSs," Proc. EEXTT 2002 and DiWeb 2002.

[13] http://db.uwaterloo.ca/~ddbms/projects/xbench/.

[14] http://www.tpc.org.

[15] http://xml.apache.org/xerces2-j/.



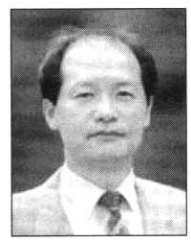
한 승 철

e-mail : schan@dblab.cse.cau.ac.kr

2003년 중앙대학교 컴퓨터공학부(공학사)

2003년~현재 중앙대학교 컴퓨터공학부 석사과정 재학 중

관심분야 : XML Stream Data Management, XML Update, XML 등



강 현 철

e-mail : hckang@cau.ac.kr

1983년 서울대학교 컴퓨터공학과(공학사)

1985년 U. of Maryland at College Park, Computer Science(M.S.)

1987년 U. of Maryland at College Park, Computer Science(Ph.D.)

1988년~현재 중앙대학교 컴퓨터공학부 교수

관심분야 : XML 데이타베이스, 웹 데이타베이스, 스트림 데이터 관리