XML 스트림 데이타에 대한 적응력 있는 질의 처리 시스템

(An Adaptive Query Processing System for XML Stream Data)

김 영 현 † 강 현 철 ††

(Younghyun Kim) (Hyunchul Kang)

요 약 센서 네트워크, 모니터링, SDI (selective dissemination of information) 등과 같이 스트림 데이타를 생성하는 응용의 증가로 스트림 데이타에 대한 질의 처리를 효율적으로 지원하기 위한 연구가 활발히 수행되고 있다. 특히 SDI와 같은 웹 환경의 응용은 XML 스트림에 대한 질의 처리를 필요로 하는데, XML은 웹 환경에서 데이타 교환의 표준이므로 이에 대한 연구는 아주 중요하다. 그러나 현재까지 제시된 XML 스트림 질의 처리 시스템들은 정적인 질의 계획을 사용하기 때문에 동적으로 변하는 스트림 데이타에 대해 적응력 있게 대처하지 못하는 문제가 있다. 반면 관계 데이타 스트림에 대한 질의 처리 시스템들은 질의 연산자 라우팅 기법을 통해 동적인 질의 계획을 사용함으로써 적응력 있는 질의 처리를 지원한다. 본 논문에서는 관계 데이타 모델을 사용하는 시스템의 적응력 있는 질의 처리 모델을 적용하여 XML 스트림에 대한 적응력 있는 질의 처리를 수행할 수 있는 시스템을 제안한다. 그리고 기존의 XML을 기반으로 하는 대표적인 시스템인 YFilter와 본 논문이 제안하는 시스템의 성능을 비교, 평가하여 본 논문이 제안하는 시스템의 효율성을 보인다.

키워드 : XML 스트림 데이타, 동적 질의 처리, 연산자 라우팅, XML 스트림 뷰

Abstract *As we are getting to deal with more applications that generate streaming data such as sensor network, monitoring, and SDI (selective dissemination of information), active research is being conducted to support efficient processing of queries over streaming data. The applications on the Web environment like SDI, among others, require query processing over streaming XML data, and its investigation is very important because XML has been established as the standard for data exchange on the Web. One of the major problems with the previous systems that support query processing over streaming XML data is that they cannot deal adaptively with dynamically changing stream because they rely on static query plans. On the other hand, the stream query processing systems based on relational data model have achieved adaptiveness in query processing due to query operator routing. In this paper, we propose a system of adaptive query processing over streaming XML data in which the model of adaptive query processing over streaming relational data is applied. We compare our system with YFilter, one of the representative systems that provide XML stream query processing capability, to show efficiency of our system.*

Key words : XML Stream Data, Dynamic Query Processing, Operator Routing, XML Stream View

1. 서 론

스트림 데이타를 생성하는 응용의 증가로 스트림 데이타에 대한 효율적인 질의 처리에 관한 연구가 활발히 수행되고 있다. 그 응용의 예로는 센서 네트워크[1], 모니터링[2], SDI(selective dissemination of information) [3-5] 등을 들 수 있다. 특히, 웹 환경에서 데이타 교환의 표준으로 XML이 부각된 이래 SDI와 같은 웹 환경의 응용을 위해 XML 스트림 데이타에 대한 질의 처리 시스템의 연구도 활발히 수행되고 있다. XML이 웹 뿐만 아니라 이기종 디바이스 간의 데이타 교환을 위해서 사용되므로 웹 응용뿐만 아니라 모니터링이나 센서 네트워크와 같은 다른 응용에서도 XML을 스트림 데이타 모델로 고려하는 것이 필요하다. 본 논문에서는 XML을

· 본 논문은 한국과학재단 특정기초연구사업(R01-2003-000-10395-0) 지원으로 수행되었음

† 정 회 원 : 삼성전자 디지털미디어 사업부
yhkim@dblab.cse.cau.ac.kr
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
hckang@cau.ac.kr

논문접수 : 2005년 8월 12일
심사완료 : 2006년 3월 4일

데이타 모델로 유동적으로 변하는 **XML 스트림** 데이타에 대한 **적응력 있는 질의 처리**를 지원하는 시스템을 제안한다.

스트림 데이타는 연속적이며 빠르게 생성되며, 데이타의 양에 한계가 없다. 이런 스트림 데이타를 기존의 DBMS를 사용하여 처리하는 것은 질의 응답 시간이나 저장 공간 측면에서 효율적이지 못하다. 그래서 스트림 데이타를 저장하지 않고 스트리밍 되는대로 질의 처리를 수행하는 시스템이 요구된다. 이런 시스템들은 스트림 데이타의 특성 상 기존의 DBMS에서 지원하는 일회성 질의 보다는 스트림 데이타 처리 시스템에 등록되어 계속적으로 존재하는 **연속 질의**를 대상으로 하였으며 연속 질의를 효과적으로 처리하기 위한 다양한 기법에 의해 구성된다. 이들 시스템들은 스트림 데이타의 종류에 따라 두 가지 부류의 시스템으로 나뉠 수 있다.

첫 번째로는 관계 데이타를 스트림 데이타로 입력받는 시스템이다[6-8]. 이런 시스템들은 **동적인 질의 계획**과 **연산자 라우팅** 기법을 사용하여 스트림 데이타를 처리한다. 동적인 질의 처리 기법은 시스템에 등록된 연속 질의에 대한 질의 계획을 실시간으로 변경하면서 질의 처리를 수행하는 것을 말한다. 이런 동적인 질의 처리 기법은 질의에 존재하는 연산자를 개별적으로 관리하면서 스트림 데이타의 특성에 따라 각 연산자의 우선순위를 달리하여 현재 시스템에 들어오는 데이타에 대한 적응력 있는 처리에 초점을 두고 있다.

두 번째로는 XML 스트림 데이타 질의 처리 및 필터링 시스템이다[9,3-5]. XML을 기반으로 하는 시스템은 우선적으로 XML의 구조를 파악하기 위해 XML 문서를 파싱하는 작업을 수행한다. 그리고 XML 문서를 파싱하는 중에 시스템에 등록된 질의를 처리하는 방식으로 대부분 구현되어 있다. XML 문서를 파싱하는 중에 질의 처리를 하기 위해 시스템에 등록된 사용자의 질의들을 **오토마타**로 변환하여 질의 상태를 만족하는지 검사하면서 질의를 처리한다. 하지만 이와 같은 시스템들은 오토마타를 사용하므로 **정적인 질의 계획**으로만 질의를 처리한다. 그렇기 때문에 관계 데이타 기반의 시스템과는 달리 스트림 데이타의 특성을 고려하지 않아 적응력 있는 질의 처리를 할 수 없다는 문제점을 가지고 있다.

본 논문은 이와 같은 문제를 해결하기 위해 기존의 관계 데이타를 기반으로 하는 시스템들의 동적인 질의 처리 모델을 적용하여 XML 스트림 데이타에 대한 적응력 있는 질의 처리를 지원하는 시스템을 제안한다. 제안하는 시스템은 같은 경로를 가지는 XPath 질의들을 서로 다른 연산자로 분리하여 처리하는 방식을 사용하며 이를 바탕으로 질의에 포함되는 연산자를 개별적으로 처리한다. 이와 같은 방식은 기존의 XML 데이타 기반의 시스템이 같은 경로의 연산자를 함께 처리하는 방식과는 다르다. 하지만 본 논문이 제안하는 시스템은 이와 같은 연산자 처리 방식으로 인해서 입력되는 XML 스트림 데이타의 특성에 따라 질의에 포함되는 각 연산자의 수행 순서를 동적으로 변경하여 적응력 있는 질의 처리를 수행할 수 있다. 본 논문에서는 기존의 관계 데이타 스트림에 대한 동적인 연산자 처리 방식을 XML 스트림에 대한 연산자 처리에 적용할 때의 문제점을 파악하고 이를 해결하는 기법을 제안한다. 그리고 기존의 XML 데이타를 기반으로 하는 대표적인 시스템인 YFilter [4]와의 성능 비교, 평가를 통해 본 논문이 제안하는 시스템의 적응력 있는 질의 처리의 효율성을 보인다.

XML 스트림 데이타에 대한 질의 처리 분야에서 본 논문의 기여는 다음과 같다.

- 동적으로 변하는 XML 스트림 데이타에 대해 적응력 있는 질의 처리를 지원하는 시스템을 제안한다. 이를 위해 [6-8]과 같은 기존의 관계 데이타를 기반으로 하는 시스템의 질의 처리 모델을 XML 스트림 데이타 질의 처리 시스템에 적용하는 기법을 제안한다.
- [6-8]과 같은 시스템이 사용하는 질의 처리 모델을 XML 데이타 기반의 시스템에 적용하는 데 있어 발생하는 문제를 파악하고 이를 해결하기 위한 기법을 제안한다.
- 기존의 XML 데이타 기반 스트림 데이타 질의 처리 시스템과의 성능 비교, 평가를 통해 본 논문이 제안하는 시스템의 질의 처리 성능을 검증한다. 이를 위해 대규모 연속 질의를 지원하는 YFilter와 본 논문이 제안하는 시스템의 성능을 비교, 평가하여 본 논문이 제안하는 시스템이 우수하다는 것을 보인다.

본 논문의 구성은 다음과 같다. 2절에서는 스트림 데이타 처리 시스템에 관한 관련 연구를 기술한다. 3절에서는 본 논문에서 제안하는 XML 스트림 데이타에 대한 적응력 있는 질의 처리 시스템에 대해 기술한다. 4절에서는 3절에서 제안하는 시스템의 성능을 YFilter와 비교, 평가한 결과를 기술한다. 끝으로 5절에서는 결론을 맺고 향후 연구 내용을 기술한다.

2. 관련 연구

기존의 관계 데이타 모델을 기반으로 하는 시스템들은 동적인 질의 처리 모델을 지원한다[6-8]. 그 중 본 연구의 기반이 되는 시스템은 센서 네트워크 상에서 적응력 있는 스트림 데이타 처리를 수행하는 TelegraphCQ [6]이다. TelegraphCQ의 데이타 모델은 관계 데이타 모델이며 연속 질의 모델은 SQL에 시구간을 추가한 형

태의 것이다. 또한 SteMs를 사용하여 질의와 데이타를 모두 스트림 데이타의 형태로 처리한다. 그리고 Eddy [6, 10]를 사용하여 연속적으로 들어오는 스트림 데이타를 라우팅한 후 SteMs에 저장한다. Eddy는 시스템에 들어오는 스트림 데이타를 라우팅하여 시스템에 등록된 질의 조건을 만족하는지 검사하는 기법이다. 또한 Eddy는 스트림 데이타에 대한 라우팅 정책에 따라 질의 처리를 수행한다. Eddy의 라우팅 정책으로는 Back-pressure와 Back-pressure에 티켓 정책을 더한 기법이 있다. 이 기법들을 사용하여 질의에 존재하는 연산자들을 스트림 데이타의 특성에 맞게 라우팅한다. Back-pressure는 각각의 연산자들에 입력되는 스트림 데이타의 수와 연산자에 의해 처리된 스트림 데이타의 수를 측정하여 입력된 스트림 데이타 대비 처리된 스트림 데이타의 비율을 계산하여 질의 처리 속도가 빠른 연산자와 느린 연산자를 구별한다. 그래서 빠른 연산자를 느린 연산자보다 먼저 처리한다. 그리고 Back-pressure에 티켓 정책을 더한 기법은 연산자에 의해 처리가 완료된 스트림 데이타가 질의 결과를 만족하지 않아 버려지게 되는 경우가 발생하면, 해당 연산자에게 티켓을 부여하는 기법이다. 티켓은 연산자에 계속적으로 누적되며, 누적된 티켓의 수로 각각의 연산자들의 우선 순위를 정할 수 있다. 많은 티켓을 가지는 연산자는 낮은 선택률을 가지는 연산자라고 할 수 있다. 이 두 기법을 사용하여 Eddy는 연산자들의 처리 속도와 선택률을 계산할 수 있다. 이렇게 계산된 값들을 사용하여 빠른 질의 처리를 보이는 연산자와 낮은 선택률을 가지는 연산자를 선택하여 질의 처리를 수행한다. 낮은 선택률을 가지는 연산자를 먼저 처리하기 때문에 다음으로 선택되는 연산자는 이전의 연산자보다 적은 수의 데이타를 처리하게 된다. 이와 같은 기법을 사용하여 TelegraphCQ는 스트림 데이타의 특성에 맞게 각각의 연산자를 동적으로 배치하여 적응력 있는 질의 처리를 수행한다.

XML을 기반으로 하는 기존의 시스템들[9,3-5]은 이와는 달리 정적인 질의 처리 모델을 사용한다. 그 예로 XFilter[3], YFilter[4], XMLTK[5] 등의 XML 스트림 필터가 있다. XFilter는 시스템에 등록된 각 XPath 질의마다 개별적으로 유한 오토마타(Finite State Machine)를 생성하여 스트리밍되는 XML 데이타에 적용한다. 시스템에 등록된 질의의 수만큼 유한 오토마타가 존재하기 때문에 XFilter의 경우 질의 수가 많아지게 되면 확장성이 떨어진다는 단점이 있다. XFilter의 이런 문제를 해결하기 위해 YFilter가 제안되었다. YFilter는 시스템에 등록된 모든 질의를 하나의 비결정적 유한 오토마타(Nondeterministic Finite Automata)로 변환하여 질의 간의 공통된 경로에 대한 처리를 공유하는 방식을 취한다. 모든 질의가 하나의 비결정적 오토마타로 공유되기 때문에 같은 XPath 경로를 가지는 질의들의 경우 XFilter와 달리 한번의 연산 과정으로 처리를 마칠 수 있다는 장점이 있다. 하지만 '//' 또는 '*' 와 같은 비결정적 경로가 등록된 XPath 질의들에 다수 존재하게 되면 오토마타의 상태 수가 증가하여 확장성이 떨어지는 단점이 있다. XMLTK에서는 YFilter의 비결정적 경로에 대한 문제를 해결하고자 결정적 오토마타(Deterministic Finite Automata)를 사용하는 기법을 연구하였다. 결정적 오토마타는 비결정적 오토마타에 비해 수행 효율은 단연 뛰어나지만 비결정적 오토마타에 비해 상태 수가 기하급수적으로 증가하여 실용성이 없으므로 기존 연구들에서 의도적으로 회피했다. XMLTK에서도 시스템에 등록된 모든 질의를 하나의 결정적 오토마타(이를 eager-DFA라 부른다)로 변환하는 것이 아니라 비결정적 오토마타를 먼저 구성한 후 결정적 오토마타의 필요한 부분만 실행 시간(run time)에 구성해 가면서 처리한다. 이를 lazy-DFA라 부른다. XMLTK는 결정적 오토마타를 사용하여 비결정적 오토마타가 가지는 문제를 XML 스트림 데이타와 등록된 질의에 따라 일부 해결할 수 있음을 보였지만, 비결정적 오토마타에 비해 요구되는 메모리 용량이 현격히 증가한다는 단점이 있다. 따라서 XMLTK는 상태 수의 현저한 증가를 초래하는 조건 연산(predicate)을 지원하지 못하는 한계도 있다. 이상으로 살펴본 XML 데이타 기반의 시스템들은 모두 오토마타 기반의 정적인 질의 처리 모델을 사용한다. 이와 같은 정적인 질의 처리 모델은 동적으로 변하는 XML 스트림 데이타에 대하여 적응력 있는 질의 처리를 수행하지 못한다는 단점이 있다.

본 논문에서 제안하는 시스템은 앞서 살펴본 기존의 시스템과는 몇 가지 차이점이 있다. 첫째, 기존의 대표적인 스트림 데이타 처리 시스템[6-8]들이 주로 관계 데이타 모델을 사용하는 데 반해 본 논문이 제안하는 시스템은 웹 데이타 교환의 표준으로 자리 잡은 XML을 데이타 모델로 사용하여 다양한 응용에서 발생할 수 있는 XML 스트림 데이타에 대한 질의 처리를 가능하게 한다는 것이다. 둘째, XML을 기반으로 하는 XFilter, YFilter, XMLTK 등과 같은 시스템이 가지는 정적인 질의 처리 모델과는 달리 관계 데이타를 기반으로 하는 시스템의 동적인 질의 처리 모델을 사용한다는 것이다. 동적인 질의 처리 모델은 시스템에 입력되는 동적으로 변하는 스트림 데이타에 대하여 적응력 있는 질의 처리를 가능하게 한다는 장점을 가진다.

3. XML 스트림 데이타에 대한 적응력 있는 질의 처리 시스템

본 절에서는 본 논문이 제안하는 XML 스트림 데이타에 대한 적응력 있는 질의 처리 시스템의 동작 과정과 각각의 모듈이 사용하는 기법에 대해서 설명한다. 우선 본 논문이 제안하는 시스템의 개요를 기술하면 다음과 같다.

기존의 XML 데이타 기반 시스템은 오토마타를 사용하여 같은 XPath 경로를 가지는 질의를 함께 처리하는 기법을 사용한다. 하지만 본 논문이 제안하는 시스템에서는 입력되는 XML 스트림 데이타에 대한 등록된 질의에 포함되는 XPath 질의의 결과를 XML 스트림에 실체뷰로 추가한다. 이를 스트림 뷰라고 부른다[11]. XML 스트림 데이타에 대한 이와 같은 전처리 작업은 시스템에 등록된 질의에 포함되는 XPath 질의의 개별적인 접근을 가능하게 해주며 관계 데이타 기반의 동적인 질의 처리 기법을 적용할 수 있게 해준다.

본 논문의 XML 스트림 데이타 처리 시스템은 XML 스트림 데이타에 스트림 뷰를 더하는 **Stream Mediator** 모듈과 실제적인 질의 처리를 담당하는 **StreamSlimer** 모듈로 구성된다. 그리고 StreamSlimer 모듈은 사용자로부터 입력받은 질의를 분석하는 **QueryAnalyzer(질의 분석기)**, 분석된 질의가 저장되는 **QueryRepository(질의 저장소)**, 시스템에 등록된 질의의 질의 계획을 생성하는 **QueryPlanner(질의 계획기)**, 생성된 질의 계획에 따라 질의를 처리하는 **QueryProcessor(질의 처리기)**로 구성된다. 전체적인 시스템의 구조는 그림 1과 같다.

사용자로부터 입력되는 질의는 질의 분석기를 거쳐 질의 저장소에 등록된다. 질의 저장소는 등록된 질의에 한하여 질의 처리 과정 중에 필요한 XPath 질의에 시스템 내부의 식별자를 부여한다. 질의 처리 과정 중에 필요한 XPath 질의와 식별자는 Stream Mediator 모듈에 전달되며, Stream Mediator 모듈은 이 값을 사용하여 시스템에 들어오는 XML 문서에 스트림 뷰를 더한다. 스트림 뷰가 더해진 XML 스트림 데이타가 **Stream Buffer Manager**에 저장되면 질의 처리기는 질의 계획기가 생성한 정적인 질의 계획을 사용하여 질의 처리를 시작한다. 질의 처리기는 질의 처리의 결과를 토대로 하여 각각의 질의 처리 연산자들의 **선택률(selectivity)**을 계산한다. 질의 처리기는 연산자들이 가지는 선택률을 비교하여 연산자들 사이의 처리 순서(질의 계획)를 결정하며, 선택률을 기반으로 생성된 질의 계획(**동적인 질의 계획**)을 사용하여 질의 처리를 진행한다. 질의 처리기를 통해 처리된 질의 결과는 **Result Manager**를 거쳐 최종적으로 사용자에게 전달된다. 만약 XML 스트림의 필터링만 수행할 경우에는 결과 생성 과정은 생략된다.

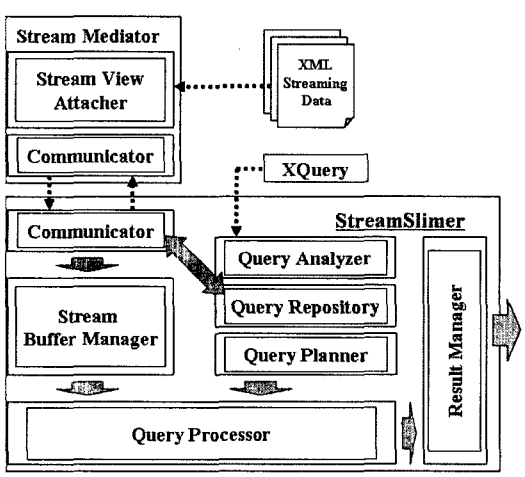

그림 1 XML 스트림 데이타에 대한 적응력 있는 질의 처리 시스템의 구조

3.1 XML 스트림 뷰

본 논문이 제안하는 시스템은 XML 스트림 뷰를 사용하여 동적인 질의 처리를 수행한다. 스트림 뷰는 스트림에 대한 뷰로서 즉, 스트리밍 되는 데이타에 대한 질의의 결과 스트림을 의미한다. XML 스트림 뷰는 XML 문서에 대한 XML 질의의 결과 엘리먼트 정보를 문서의 헤더 형태로 함께 저장한 것이다. 본 시스템에서는 XQuery 질의와 관련된 XPath 질의를 모두 스트림 뷰로 등록하여 질의 처리를 수행할 수 있다. 질의와 관련된 XPath 질의가 모두 스트림 뷰로 등록되어 있기 때문에 질의 처리 과정 중에 각각의 질의 처리 연산자가 필요로 하는 XPath 질의 결과에 개별적으로 접근할 수 있다. 이런 개별적인 접근은 질의 처리 연산자들의 독립성을 보장할 수 있게 한다. 그리고 독립성을 가지는 연산자들은 정해진 순서 없이 시스템에 들어오는 스트림 데이타의 특성에 따라 재배치할 수 있다는 장점을 가진다.

그림 2를 보면 기존의 XML 데이타 기반의 시스템과 본 시스템의 차이를 알 수 있다. 질의의 조건 연산자 A, B, C에 대해서 기존의 XML 데이타 기반의 시스템들은 예를 들어 (A->B->C)와 같이 정해진 순서로 질의 처리를 수행할 수 밖에 없지만 XML 스트림 뷰를 사용할 경우 (A->B->C), (C->B->A), (B->A->C) 등의 다양한 순서로 질의 처리를 수행할 수 있다.

본 시스템에 입력되는 XML 스트림 데이타는 Stream Mediator 모듈을 거쳐 StreamSlimer 모듈로 전달된다. 이때 Stream Mediator 모듈의 **Stream View Attacher**는 XML 스트림 데이타에 스트림 뷰를 더하는 과정을 수행한다. 그림 3은 XML 스트림 데이타에 스트림 뷰를 더하는 과정을 나타낸 것이다. 사용자 질의가 시스템에 등록되면 질의 저장소에 질의에 대한 정보가 담겨진다.

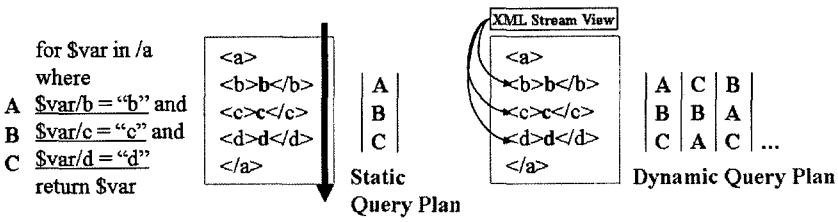

그림 2 기존의 XML 데이타 기반 시스템(좌)과 본 시스템(우)과의 질의 처리 모델 비교

그리고 질의 저장소는 질의 처리에 필요한 XPath 질의에 시스템 내부의 식별자를 할당한다. 질의 처리에 필요한 각각의 XPath 질의와 해당하는 식별자는 해시 맵을 사용하여 저장되며 이렇게 저장된 데이타는 시스템 내부의 Communicator를 통하여 Stream Mediator 모듈에 전달된다. Stream Mediator 모듈은 Communicator를 통해 질의 처리에 필요한 XPath 질의와 각각의 식별자를 전달받는다. Stream Mediator 모듈은 전달 받은 값을 기반으로 시스템 내부로 전달되는 XML 문서를 파싱하면서 질의 처리에 필요한 XPath 질의 결과 엘리먼트 정보를 원본 XML 문서에 더해준다. 이와 같은 과정을 거쳐서 원본 XML 문서에 XML 스트림 뷰가 더해지게 된다.

XML 문서에 더해지는 스트림 뷰는 pathId, beginPos, endPos의 세 가지 값을 가진다. pathId는 해시 맵을 통해 XPath 질의와 매핑되며 이 값을 사용하여 질의 처리 과정 중에 필요한 XPath 질의의 결과를 접근할 수 있다. 또한 pathId를 통해 스트림 뷰에 접근하여 (beginPos, endPos) 값을 얻을 수 있다. 이 값은 해당 XML 문서에 대한 XPath 질의 결과의 바이트 오프셋으로 beginPos는 결과의 시작 부분 바이트 오프셋을 의미하며 endPos는 끝 부분의 바이트 오프셋을 의미한다

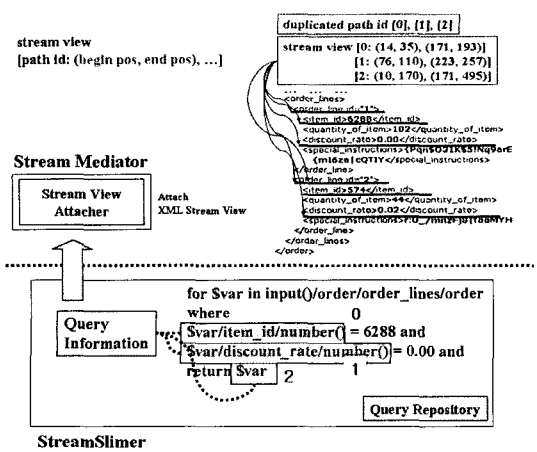

그림 3 스트림 뷰를 더하는 과정

[12]. 그리고 duplicated path id 리스트는 XQuery 질의에 포함되는 XPath 질의 가운데 중복하여 존재하는 XPath 질의의 경로에 해당하는 pathId를 저장한다.

3.2 질의 계획

본 시스템은 앞 절에서 살펴본 XML 스트림 뷰를 통해 각 연산자들의 개별적인 접근을 보장한다. 그리고 이를 바탕으로 질의 처리 과정 중에 각 연산자의 우선 순위를 판단하여 연산자의 실행 순서를 실시간으로 변경하여 처리한다. 이와 같이 질의 연산자를 동적으로 선택하여 수행하는 기법을 **연산자 라우팅**이라고 한다. 이러한 동적인 질의 처리가 가지는 장점은 스트림 데이타의 변화에 따라 질의 연산자들에 대한 질의 계획을 변경해 가면서 적응력 있는 질의 처리를 수행할 수 있다는 것이다. 연산자 라우팅 기법의 예를 들면 다음과 같다. 우선 XML 문서 상에 '/a/b/c > 100' 인 경우가 100%의 확률로 존재하고 '/a/b/d < 150' 인 경우는 20%의 확률로 존재한다고 가정하자. 그리고 XML 문서를 순차적으로 파싱한다면 '/a/b/c' 가 '/a/b/d' 보다 우선하여 파싱 작업이 이루어진다고 하자. 이런 가정 하에 조건이 '/a/b/c > 100 and /a/b/d < 150' 인 질의가 들어왔다고 하면, 정적인 질의 처리를 수행하는 시스템에서는 '/a/b/c' 가 '/a/b/d' 보다 먼저 파싱되기 때문에 항상 '/a/b/c > 100' 인 조건을 처리한 후 '/a/b/d < 150' 을 처리한다. 그러나 만약 '/a/b/d < 150' 을 먼저 처리한다면 '/a/b/c > 100' 은 처리하지 않고 질의 수행을 마칠 확률이 80%나 된다. 이러한 질의 최적화는 XML 데이타 기반의 XFilter [3], YFilter [4], XMLTK [5] 등의 정적인 질의 처리 기법에서는 불가능하다. 따라서 본 논문에서는 이와 같은 기능을 제공하고자 XML 데이타 기반의 시스템에 연산자 라우팅 기법을 적용하여 동적인 질의 처리를 가능하게 하였다. 위의 예와 같은 질의 조건을 가진 질의가 본 시스템에 등록되면 '/a/b/c > 100', '/a/b/d < 150' 과 같은 문자열이 각 질의 연산자를 구별하는 식별자가 되며, XML 스트림 데이타에 대해 각각의 질의 연산자 처리를 수행한다.

시스템에 등록된 질의에 대한 질의 계획을 그림 4의 예를 들어 설명하면 다음과 같다. 그림 4의 질의 Q0가

등록되면 질의 분석기를 통해 질의가 분석된 후 질의 저장소에 저장된다. 단계 1에서 생성한 질의 계획은 정적이지만 질의 처리 과정 중에 각 질의 연산자의 선택률을 계산하여 동적으로 질의 계획을 변경한다. 다음으로 질의 Q1이 질의 분석기를 통해 질의 저장소에 등록되면 질의 계획 생성기는 이전에 등록된 질의 Q0와 Q1이 공유할 수 있는 연산자가 있는지 검사한다. 검사 후 공유할 수 있는 연산자가 있기 때문에 그림 4의 단계 2에서와 같은 질의 계획이 생성된다. 마지막으로 질의 Q2가 질의 분석기를 통해 질의 저장소에 등록된다. 질의 Q2는 공유되는 연산자가 존재하지 않기 때문에 그림 4의 단계 3과 같은 질의 계획이 세워진다. 본 시스템은 단계 3의 질의 계획을 사용하여 실질적으로 질의 처리를 하며 XML 스트림 데이타를 처리하는 과정 중에 각 질의 연산자들의 선택률을 측정한다. 그리고 측정한 선택률을 기반으로 연산자들의 실행 순서를 실시간으로 변경해 질의를 처리한다.

3.3 선택률 기반 연산자 라우팅

앞 절에서 살펴본 질의 계획은 질의 처리 중에 실시간으로 계산된 연산자들의 선택률을 사용하여 선택률이 낮은 연산자 순서로 처리하는 것이다. 연산자의 선택률은 해당 연산자의 처리 결과에 의해 계산한 값으로 각 연산자들이 질의 처리 중에 데이타를 선택하는 비율을 의미한다. 같은 질의에 포함된 연산자들의 순서를 선택률이 가장 낮은 순서로 배치하게 되면 낮은 선택률을 가지는 연산자에 의해 높은 비율로 데이타가 걸러지기 때문에 다음 연산자가 처리해야 하는 데이타의 수가 그만큼 줄어든다. 그리고 줄어든 데이타의 수만큼 질의 처리 성능 향상을 기대할 수 있다. 예를 들어 그림 4의 단계 3과 같은 질의 계획에서 질의 Q0의 A 연산자의 선택률이 가장 낮아서 먼저 수행했다고 하자. 이때 A 연산자의 질의 처리 결과가 거짓이라면 Q0의 A 연산자 다음의 B 연산자를 수행하지 않아도 되는 이점과 A 연산자를 공유하는 질의 Q1의 질의 처리를 중단해도 되는 이점을 얻을 수 있다.

한편, 선택률을 계산하는 방법은 다음과 같다. 앞으로 시스템에 들어올 스트림 데이타를 예측할 수 없기 때문에 이미 질의 처리를 끝낸 데이타를 토대로 연산자들의 선택률을 측정해야 한다. 각 연산자의 선택률을 계산하는 방법은 그림 5의 공식과 같다. 각 연산자가 처리한 스트림 데이타 중에 연산자의 질의 처리를 만족하는 데이타의 수에 연산자가 처리한 데이타의 수를 나눈 값이 해당 연산자의 선택률이다. 이 값을 사용하여 XQuery 질의에 포함되는 질의 연산자들의 순서를 동적으로 변경해 선택률이 낮은 연산자 순서로 질의를 처리한다. 하지만 이와 같은 방법은 시스템에 등록된 질의에 대해

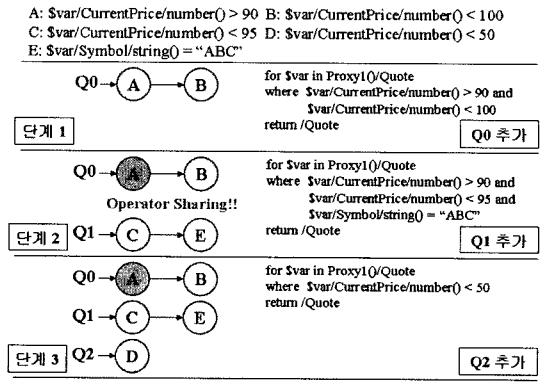

그림 4 질의 계획

$$\text{각 연산자의 Selectivity} = \frac{\text{해당 연산자의 질의 처리 결과가 참인 경우의 개수}}{\text{해당 연산자가 처리한 스트림 데이터의 개수}}$$

그림 5 각 연산자의 선택률을 구하는 공식

고정적인 선택률을 가지는 스트림 데이타가 입력될 때만 적용할 수 있는 기법으로 가변적으로 변하는 선택률을 가지는 스트림 데이타에 대해서는 적용할 수 없다는 단점을 가진다.

이와 같은 문제를 해결하여 시스템에 입력되는 스트림 데이타에 대해 적응력 있게 연산자의 선택률을 측정하는 기법은 다음과 같다. 앞서 살펴본 선택률 기반 연산자 라우팅 기법은 각 연산자에 대해 변화하는 선택률을 가지는 스트림 데이타가 시스템에 입력될 경우 적응력 있는 연산자 라우팅 성능을 보여주지 못한다(그림 20 참조). 이와 같은 문제의 원인은 각 연산자들의 선택률이 계속적으로 누적된다는 데 있다.

예를 들어, 그림 6의 단계 1의 질의 계획은 각 연산자의 선택률을 반영하여 (A->B)의 질의 계획을 사용하여 입력되는 스트림 데이타를 처리하고 있음을 알 수

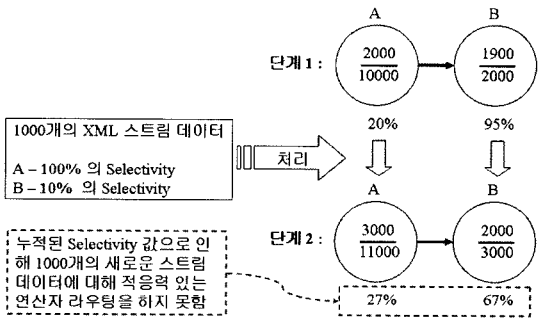

그림 6 선택률 기반 연산자 라우팅 기법의 문제점

있다. 하지만 그림 6과 같이 각 연산자에 대해 새로운 선택률을 가지는 스트림 데이타가 입력될 경우 선택률 기반 연산자 라우팅 기법은 단계 2가 되어도 (A->B)의 질의 계획을 가지고 스트림 데이타를 처리함을 알 수 있다. 좀 더 나은 질의 처리 성능을 보장하기 위해서는 단계 1에서 단계 2로 가는 도중에 새롭게 입력되는 스트림 데이타의 선택률을 반영하여 (B->A)와 같은 질의 계획으로 전환해야 한다. 본 논문에서는 이와 같은 기능을 제공하기 위해 선택률 기반 연산자 라우팅 기법에 각 연산자의 선택률을 **갱신**하는 기법(**Refresh**)을 다음과 같이 추가하였다.

예를 들어, 그림 7을 보면 그림 6의 단계 2와는 달리 (B->A)의 질의 계획으로 전환하여 새롭게 입력되는 스트림 데이타를 처리하는 것을 볼 수 있다. 이와 같은 질의 계획의 동적인 전환이 가능한 이유는 그림 7의 단계 1에서 단계 2로 가면서 선택률의 갱신이 수행되기 때문이다. 갱신 기법은 선택률 기반 연산자 라우팅 기법에 일정한 주기를 두어 현재까지 누적된 각 연산자의 선택률을 갱신하는 작업을 말한다. 각 연산자의 선택률을 모두 갱신하되 선택률을 단순히 0%로 리셀하는 것이 아니라 기존의 선택률을 유지하면서 갱신한다. 모든 연산자의 선택률이 0%로 리셀되면 현재까지 적절한 계획으로 처리되던 질의 처리 과정에 영향을 줄 수 있기 때문에 각 연산자의 기존 선택률은 유지하면서 스트림의 변화에 빠른 적응을 할 수 있게 선택률을 구성하는 값들을 변경하는 것이다. 그림 7을 보면 단계 1에서 단계 2로 가면서 연산자 A와 B의 선택률을 각각 기존의 20%, 95%로 유지하면서 갱신하기 위하여 처리된 데이타와 각 연산자를 만족하는 스트림 데이타의 수를 각각 2/10, 9.5/10 으로 변경하였다. 그 이후 1,000개의 XML 스트림 데이타가 스트리밍되는 동안 A, B 연산자의 선택률이 각각 100%, 10% 였다고 하자. 그러면 이 중 불과 9개의 데이타가 처리된 직후인 단계 3에 이르러 A, B 연산자의 선택률이 역전되어 질의 계획을 (B->A)로 바꾸게 된다. (즉, 이때의 A, B 연산자의 선택률은 각각 (2+9*1.0)/(10+9)=11/19=58%, (9.5+9*0.1)/(10+9)=10.4/19=55%이 되어 B 연산자의 선택률이 A 연산자의 선택률보다 더 낮아진다.) 계속하여 나머지 991개의 데이타가 처리되어 단계 4에 이르면 B 연산자의 선택률은 (10.4+991*0.1)/(19+991)=109.5/1010=11%가 되고, A 연산자의 선택률은 B 연산자에 의해 991개의 데이타 중 그 10%만 처리하여 100%를 선택하므로 (11+99.1*1.0)/(19+99.1)=110.1/118.1=93%가 된다.

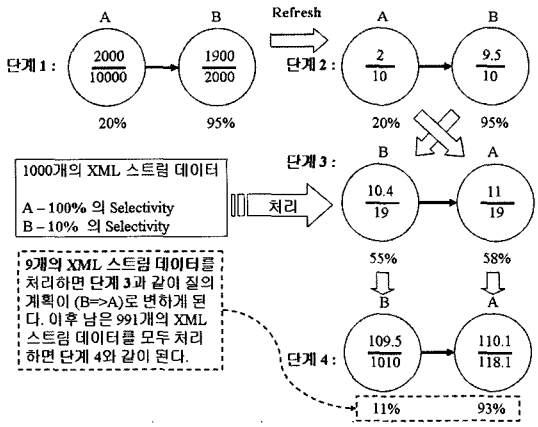

그림 7 적응력 있는 선택률 기반 연산자 라우팅 기법

3.4 질의 처리

본 절에서는 전절에서 살펴본 선택률 기반 연산자 라우팅에 의한 질의 처리 과정을 그림 8의 예를 들어 설명한다. 그림 8은 왼쪽에 기술된 XQuery 질의의 처리 과정을 순차적으로 나타낸 그림이다. XQuery 질의에 포함되는 질의 연산자는 A, B, C 세 가지이며 각 연산자의 처리 결과에 의해 해당 연산자의 선택률을 측정한다. 그리고 데이타가 스트리밍되는 동안 선택률의 변화를 반영해 실시간으로 질의 계획을 변경하게 된다.

그림 8의 초기단계에서는 XQuery 질의를 분석하여 조건 연산들을 질의 연산자 A, B, C로 차례로 추출하고 이들의 결과 큐를 초기화하였다. 초기단계에서는 해당 연산자가 처리한 스트림 데이타가 없었기 때문에 각 연산자의 선택률은 모두 0%이며 연산자가 추출된 순서대로 (A->B->C)의 질의 계획을 세운다. 이후 타임스탬프가 0인 스트림 데이타가 전달되면 초기단계에 세워진 질의 계획을 사용하여 연산자들을 하나씩 순차적으로 처리한다. (이때 각 연산자가 필요로 하는 XPath 질의 결과는 XML 스트림 뷰를 통해 바로 접근할 수 있다. XML 스트림 뷰에 대한 설명은 3.1절을 참조.) 먼저 질의 연산자 A를 처리한다. A 연산자의 처리가 끝나게 되면 그 결과를 결과 큐에 적게 되며 그림 5의 선택률을 구하는 공식을 사용하여 A 연산자의 선택률을 구하게 된다. 질의 연산자 A가 가지는 조건을 입력된 스트림 데이타가 만족하므로 A 연산자의 선택률은 100% (=1/1) 가 된다. 질의 연산자 A에 대한 처리 과정이 종료되어 다음 연산자 B를 처리한다. B 연산자의 처리 결과는 거짓으로 우선 결과 큐에 그 결과를 적은 후 B 연산자의 선택률을 구하게 된다. 즉, B 연산자의 선택률은 0% (=0/1)가 된다. 질의 연산자 B의 처리 결과가 거짓이기 때문에 다음 연산자인 C는 처리하지 않고 타임스탬프가 0인 스트림 데이타에 대한 처리를 모두 종료하게 된다. 이후 다음 차례의 스트림 데이타를 처리할 때는 타임스탬프가 0인 스트림 데이타를 처리한 결과로 얻은 각 연산자의 선택률에 의해 선택률이 낮은 연산자

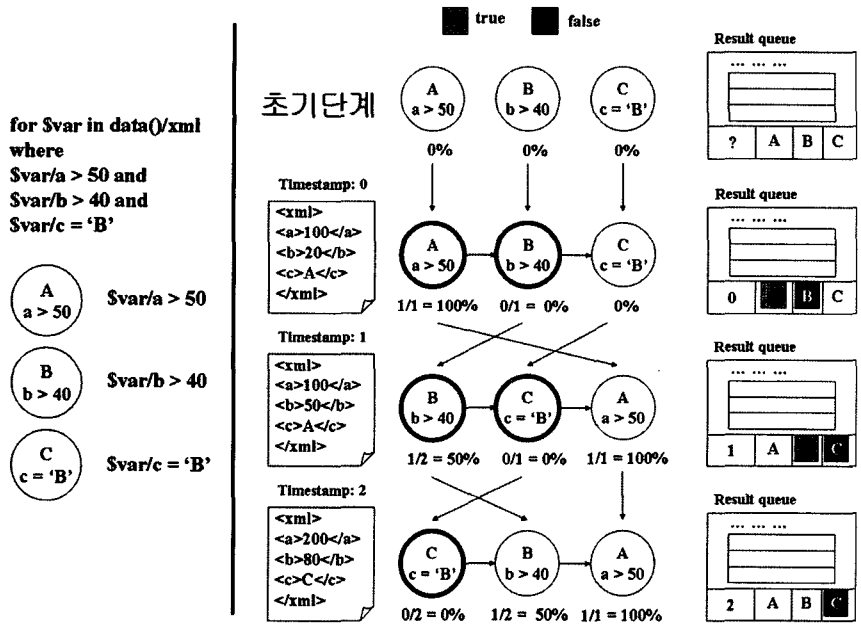

그림 8 연산자 라우팅을 통한 적응력 있는 질의 처리 과정

부터 처리한다. 이는 타임스탬프가 1인 스트림 데이타를 처리하는 부분에 나타내었다. 초기단계에서는 질의 계획이 (A->B->C) 이었지만 타임스탬프가 1인 스트림 데이타를 처리할 때의 질의 계획은 (B->C->A) 로 변경된 것을 볼 수 있다. 이는 연산자 라우팅의 결과이다. (B->C->A)로 변경된 질의 계획을 사용하여 타임스탬프가 1인 스트림 데이타를 이전과 같은 과정을 통해 처리한다. 그 결과 각 연산자의 선택률은, A는 100%, B는 50%, C는 0%로 된다. 따라서 질의 계획이 (C->B->A)로 변경되고 이와 같은 순서로 타임스탬프가 2인 스트림 데이타에 대해 질의 연산자를 전과 같은 과정으로 처리하게 된다. 복수개의 질의가 시스템에 등록되어 있으면 각 질의 별로 이상에서 기술한 질의 처리를 수행한다.

3.5 연산자 마커

본 논문이 제안하는 시스템에서는 등록된 질의들의 처리에 있어 질의 간 처리 공유의 단위는 연산자이다. 반면 오토마타를 사용하는 기존의 XML 기반 시스템들은 질의 간 처리 공유의 단위는 경로이다. 즉, 동일한 경로를 가지는 다수의 질의의 경우 경로 단위로 처리를 공유한다. 하지만 본 논문의 시스템은 관계 데이타 기반의 시스템과 같은 방식을 취해 질의 연산자가 가지는 XPath 경로 외에 조건 연산을 포함하여 연산자의 단위로 사용한다. 이와 같은 차이점으로 인해 본 시스템은 질의 처리 과정 중에 다수의 같은 XPath 경로를 공유하는 질의의 경우에도 조건 연산에 사용된 값이 다르다면 이를 서로 다른 연산자로 구분한다. 따라서 질의 간 처리를 공유하는 정도가 기존의 XML 데이타 기반의 시스템들에 비해 떨어지게 되며 이는 등록된 질의의 개수가 많아질수록 성능이 저하된다는 것을 의미하므로 이를 해결하는 기법이 필요하다.

그림 9는 위에서 설명한 성능 저하 현상의 원인을 본 논문이 제안하는 시스템과 기존의 YFilter와 비교하여 도식화한 것으로, 질의의 조건 연산들과 그에 대한 두 시스템에서의 질의 계획을 각각 나타내었다. 왼쪽의 시스템은 YFilter이고 오른쪽의 시스템은 본 논문이 제안하는 시스템에서 상기 문제를 해결하지 않았을 경우의 시스템이다. 두 시스템의 가장 큰 차이는 질의 간 처리를 공유하는 방식이다. YFilter에서는 B 연산자들이 그 조건에 나타난 값에 무관하게 경로만을 고려하였을 때 동일하므로 한 번만 처리되는 반면, 본 시스템은 같은 경로를 가지는 B 연산자들의 조건에 사용된 값이 각각 다르기 때문에 (즉, B = 0, 1, 2, ...) 이들을 각각 별도의 연산자로 간주한다. 이와 같은 차이로 인해서 본 시스템의 경우 질의 수가 많아지게 되면 성능이 저하된다 (그림 24 참조).

이러한 성능 저하 현상을 해결하기 위해 본 시스템에서는 연산자 마커(Operator Marker)라는 기법을 고안하여 사용하였다. 연산자 마커는 현재 처리된 연산자의 XPath 질의를 기억하여 추후 동일한 XPath 경로를 가

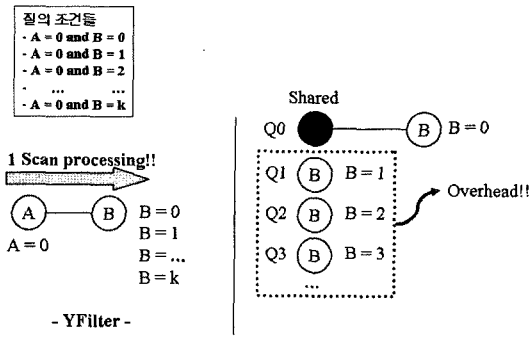

그림 9 YFilter와 본 시스템에서 연산자 마커를 사용하지 않을 경우의 질의 처리 비교

지는 연산자의 불필요한 처리를 방지하는 기능을 담당한다. 그림 9에서처럼 연산자 마커를 사용하지 않은 본 시스템의 경우 질의 Q0에서 이미 B 연산자의 처리를 마친 후에도 다른 질의의 B 연산자들을 계속 처리하게 된다. 하지만 이미 Q0의 처리 과정에서 B 연산자가 필요로 하는 XPath 질의 결과를 확인한 상태이기 때문에 질의 Q1부터의 B 연산자 처리가 필요한지 아닌지를 판단할 수 있다.

질의의 조건 연산이 '='로 구성되어 있는 경우, Q0의 B 연산자의 처리 결과가 참인지 거짓인지에 따라 두 가지의 연산자 마커가 생길 수 있다. 우선 Q0의 B 연산자의 처리 결과가 참일 경우에는, Q0의 후속 질의에 포함된 B에 대한 다른 연산자의 처리를 모두 생략할 수 있다. 그 이유는 예를 들어 Q0에 'B = 0' 인 연산자가 존재한다면 그 연산자는 이미 다른 질의의 'B = 0' 연산자와 공유되므로 'B = 0' 인 연산자의 처리는 더 이상 필요 없고 처리를 기다리는 다른 B 연산자의 조건에 사용된 값은 0이 아니므로 그 결과는 모두 거짓이 될 것이기 때문이다. Q0의 B 연산자의 결과가 거짓인 경우에는, 결과가 참인 경우와는 다소 틀리다. 우선 Q0의 B 연산자를 수행하면서 현재 처리 중인 XML 문서 내의 B 연산자가 필요로 하는 XPath 결과 값을 연산자 마커에 설정한다. 그림 10을 보면 이 값을 'X'라고 표시하였다. 그리고 질의 Q1부터 존재하는 B 연산자에 대해서는 연산자 마커를 사용하여 'B = X'인 경우의 연산자만을 처리하고 그렇지 않은 경우에는 모두 생략하면 된다. 'B = X'인 연산자도 하나만 존재하기 때문에 이 연산자를 제외한 모든 B 연산자가 생략된다.

본 논문의 시스템에서는 연산자 마커를 사용하여 같은 XPath 경로를 가지는 연산자들의 불필요한 처리를 피했기 때문에 전체적인 질의 처리 성능이 향상되는 것을 확인할 수 있었다(그림 24 참조). 연산자 마커는 해시 구조를 사용하여 구현하였으며 질의와 관련된 모든

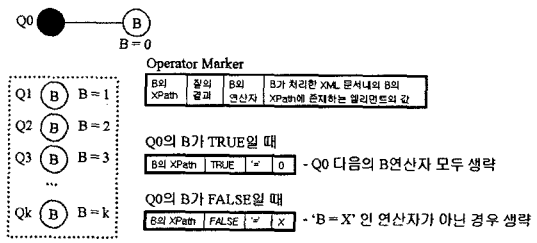

그림 10 연산자 마커

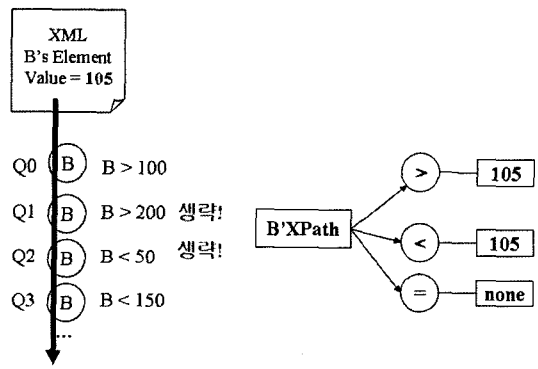

그림 11 연산자 마커의 구조와 >, <의 처리

XPath 경로에 대해서 마커가 생겨난다. 질의 수가 많아질수록 연산자 마커의 크기도 증가하는 문제점이 있지만 한 XML 문서에 존재하는 XPath 경로의 수에는 한계가 있기 때문에 연산자 마커의 크기는 질의 수의 증가에 따라 계속 증가하지 않는다.

그림 10에서는 '=' 에 대한 연산자 마커 만을 고려했는데 '>', '<' 등과 같은 다른 연산자도 질의에 사용되기 때문에 '>', '<' 등에 대한 연산자 마커도 고려해야 한다. 그림 11을 참고하여 보면 Q0의 B 연산자를 처리할 때 연산자 마커에는 현재 처리 중인 XML 문서의 B의 XPath 질의 결과가 기록된다. 이렇게 기록된 값을 통해 다음 연산자를 실행할 때 '>', '<' 등을 가지는 B 연산자를 다시 만나게 되면 연산자 마커에 '>', '<' 등에 대한 값이 존재하기 때문에 Q1의 B 연산자 같은 경우 연산자 마커에 기록된 '>' 연산자의 값이 200보다 작은 105라는 것을 확인한 후 Q1의 B 연산자를 생략하게 된다. 그리고 Q2의 B 연산자의 경우는 앞의 경우와는 반대로 '<' 연산자의 값이 50보다 큰 105이기 때문에 생략하게 된다. 이와 같은 방식으로 현재 처리하고 있는 XML 문서 내에 존재하는 B 연산자가 요구하는 값을 기록하게 되면 불필요한 연산자의 처리를 피할 수 있다.

4. 성능 평가

본 논문에서 제안한 XML 스트림 데이타 질의 처리

시스템을 jdk1.4.2를 사용하는 JAVA 환경에서 구현하였으며, 성능 실험은 Windows 2000 Server 상에서 3.6 GHz CPU를 사용하고 메모리는 2GB인 시스템에서 수행하였다. 실험은 모두 세 가지로 구성하였다. 첫 번째 실험의 목적은 본 논문이 제안하는 선택률 기반 연산자 라우팅 기법의 성능에 대한 평가이다. 선택률 기반 연산자 라우팅 기법의 성능 평가는 시스템에 등록된 질의에 대해 고정된 선택률을 가지는 스트림 데이타를 사용하여 수행하였으며, 등록된 질의가 입력되는 스트림 데이타를 모두 처리하기 위해 평균적으로 몇 개의 질의 연산자를 수행하는지를 측정하였다. 두 번째 실험은 첫 번째 실험에서 사용한 선택률 기반 연산자 라우팅 기법에 갱신 기법을 추가하여 수행하였고 질의 연산자에 대해 유동적인 선택률을 가지는 스트림 데이타를 사용하였다. 이는 시스템에 등록된 질의에 대해 유동적인 선택률을 가지는 스트림 데이타에 대해 본 논문이 제안하는 연산자 라우팅 기법이 얼마나 적응력 있는 연산자 라우팅을 지원하는지를 알아보기 위한 것이다. 마지막으로 세 번째 실험은 YFilter와 본 시스템의 성능을 비교, 평가하기 위한 것이다. YFilter는 등록된 질의들을 오토마타로 변환하여 스트리밍 되는 XML 문서에 대한 정적인 질의 처리를 수행함으로써 XML 문서를 필터링하는 대표적인 시스템으로 SDI 응용에 적용할 수 있도록 인터넷 스케일의 다수 질의를 지원한다. 그에 반해 본 논문이 제안하는 시스템은 YFilter와 같은 필터링 기능 외에 등록된 다수의 XML 질의에 대해 그 처리 결과를 반환할 수 있는 시스템이지만 YFilter와의 비교를 위해 XML 문서 필터링에 소요되는 시간만을 평가하였다. 각 실험에서 사용한 XML 문서는 그림 12와 같다.

	선택률 기반 연산자 라우팅 성능 평가	적응력 있는 연산자 라우팅 성능 평가	질의 처리 성능 평가
문서	NiagaraCQ's "Quote.xml" [20]	Synthetic Data	TPC-W Order docs[18-19]
용량 & 문서 개수	전체: 2MB 4518 개	전체: 1.5MB 16000 개	전체: 8.2MB 5000 개

그림 12 각 실험에 사용한 XML 문서 스트림

4.1 선택률 기반 연산자 라우팅의 성능 평가

연산자 라우팅에 대한 성능 평가에서는 입력되는 스트림 데이타에 대해 그림 13과 같은 XML 문서와 그림 14와 같은 선택률을 가지는 질의를 사용하였다. 즉, 질의 연산자 A~E의 조건을 만족하는 값이 문서 상에 각각 100%, 80%, 60%, 40%, 20% 로 존재한다. 이러한 선택률에 의한 최적의 질의 계획은 그림 14의 best case로 표시된 것이고 최악의 질의 계획은 worst case로 표시된 것이다. 연산자 라우팅 기법은 그림 14의 최악의 질의 계획이 초기단계에 수립되어도 질의 처리 과정 중에 계산한 연산자 A~E의 선택률에 의거하여 best case의 질의 계획과 동일하거나 그와 유사한 질의 계획으로 전환해 가도록 해야 한다.

```
<?xml version="1.0"?>
<Quote>
<Symbol>CLHB</Symbol>
<Sector>SEC0</Sector>
<Industry>IND01</Industry>
<Time>
<Year>1999</Year>
<Date>Thu Oct 28 22:45:23 1999</Date>
</Time>
<CurrentPrice>147.54</CurrentPrice>
<OpenPrice>145.20</OpenPrice>
<Change>1.61</Change>
<PrevClosePrice>144.72</PrevClosePrice>
<Volume>484766</Volume>
<DayRange>
<Low>88.65</Low>
<High>270.89</High>
</DayRange>
<Week_52_Info>
<Low>71.18</Low>
<High>807.72</High>
<Average>206.12</Average>
</Week_52_Info>
</Quote>
```

그림 13 NiagaraCQ's "Quote.xml"

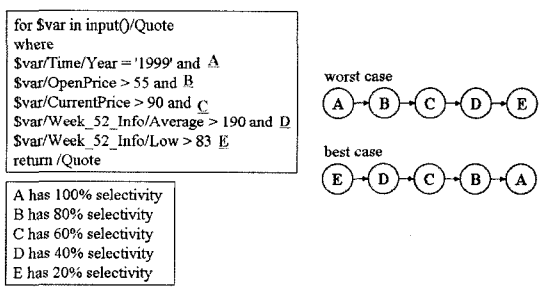

그림 14 연산자 라우팅 실험 환경

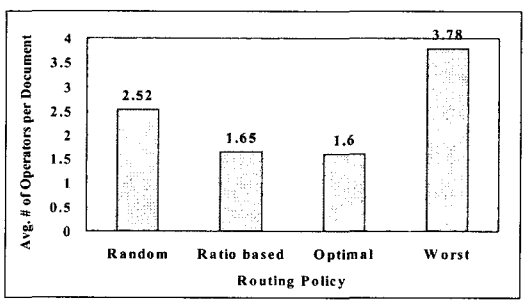

그림 15 선택률 기반 연산자 라우팅의 성능 평가

연산자 라우팅의 성능 평가는 다음과 같이 네 가지의 라우팅 기법을 비교한 것이다(그림 15 참조). x축의

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
<a>11</a>
<b>21</b>
<c>31</c>
<d>0</d>
<e>0</e>
</data>
```

그림 16 Synthetic XML 데이타

```
for $var in input()/data
where
$var/a/number() > 10 and
$var/b/number() > 20 and
$var/c/number() > 30 and
$var/d/number() > 40 and
$var/e/number() > 50
return $var
```

A: $var/a/number() > 10
B: $var/b/number() > 20
C: $var/c/number() > 30
D: $var/d/number() > 40
E: $var/e/number() > 50

	연산자 A	연산자 B	연산자 C	연산자 D	연산자 E	최적의 질의 계획
구간 W	100%	80%	60%	40%	20%	E->D->C->B->A
구간 X	5%	15%	25%	35%	45%	C->E->A->B->D
구간 Y	100%	90%	80%	70%	60%	C->E->D->A->B
구간 Z	10%	30%	50%	70%	90%	A->B->C->D->E

그림 17 적응력 있는 연산자 라우팅 실험 환경

Optimal은 그림 14의 best case 질의 계획을 의미하며 Worst는 그림 14의 worst case 질의 계획을 나타낸 것이다. Random은 질의 처리 과정 중에 연산자 A~E를 임의로 선택하여 라우팅한 기법을 나타낸 것이며 Selectivity-based는 본 논문에서 제안하는 선택률 기반의 연산자 라우팅 기법을 나타낸 것이다. y축은 시스템에 등록된 그림 14의 질의가 모든 스트림 데이타를 처리하는 데 A~E 연산자를 몇 번 수행했는지 계산하여 XML 문서 하나당 몇 개의 연산자를 평균적으로 수행했는지를 측정한 결과이다. 실험 결과를 보면 본 시스템이 사용하는 선택률 기반의 연산자 라우팅 기법이 최적의 질의 계획이 보여주는 값과 유사한 값을 가짐을 알 수 있다.

4.2 적응력 있는 선택률 기반 연산자 라우팅의 성능 평가

전 절에서 살펴본 선택률 기반 연산자 라우팅 기법의 성능 평가에서는 질의 연산자에 대해 고정된 선택률을 가지는 스트림 데이타를 사용하였다. 하지만 실제 환경에서는 질의 연산자에 대해 유동적인 선택률을 가지는 스트림 데이타가 발생한다. 본 절에서는 이와 같은 환경을 고려하여 유동적인 선택률을 가지는 스트림 데이타를 사용하여 선택률 기반 연산자 라우팅 기법에 갱신 기법을 추가하여 수행한 성능 평가 결과를 기술한다. 실험에 사용한 문서는 그림 16과 같은 형태를 가지는 합성된 XML 문서이며 프로그램에 의해 임의의 내용으로 생성하였다. 그리고 생성한 XML 문서는 그림 18과 같은 구간 구분에 따라 시스템에 입력된다. 입력되는 스트림 데이타에 대해 그림 17의 질의를 사용하여 실험을 하였으며 그림 18의 각 구간은 각 연산자 A~E에 대해 그림 17의 표에 나타낸 선택률을 가진다. 그리고 선택률을 갱신하는 주기는 시스템에 입력되는 스트림 데이타의 개수가 100의 배수가 될 때로 설정하였다. 선택률을 갱신하는 주기를 이와 같이 짧게 설정한 이유는 갱신 작업 자체가 비용이 크지 않음을 보이기 위한 것이며 갱신 주기가 짧아도 시스템에 입력되는 스트림 데이타의 각 구간에 따라 적응력 있는 선택률 측정을 한다는 것을 검증하기 위함이다.

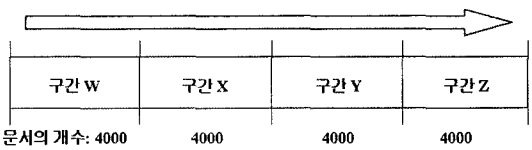

그림 18 시스템에 입력되는 XML 스트림 데이타의 구간

선택률 기반 연산자 라우팅 기법에 갱신 기법을 추가한 기법에 대한 평가는 그림 17의 구간 마다 최적의 질의 계획으로 수행한 실험 결과와 비교를 통해 수행하였다. 실험 결과는 그림 19~21과 같으며 모두 세 가지의 실험에 의한 것이다. 그림 19는 구간 W~Z 에 대해서 그림 17에 나타낸 최적의 질의 계획을 사용하여 수행한 경우의 실험 결과이며, 그림 20은 갱신 기법을 사용하지 않은 선택률 기반 연산자 라우팅 기법의 실험 결과이다. 그리고 마지막으로 그림 21은 선택률 기반 연산자 라우팅 기법에 갱신 기법을 추가한 실험 결과이다. 그림 19~21의 x축은 시스템에 스트리밍된 XML 문서 수를 의미하며, y축은 해당 문서를 처리한 시점의 질의 처리 연산자의 선택률을 측정한 것이다.

그림 19의 실험 결과를 보면 질의 처리 연산자 A~E가 입력되는 스트림 데이타에 대해 각 구간마다 그림 17과 거의 유사한 선택률을 가진다는 것을 알 수 있다.

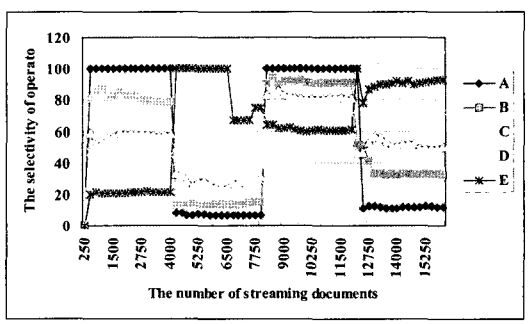

그림 19 각 구간에 대해 최적의 질의 계획으로 질의 처리를 수행한 결과

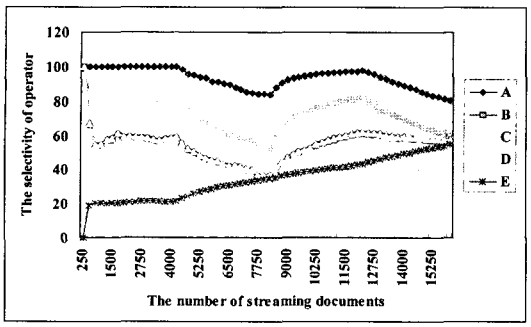

그림 20 선택률 기반 연산자 라우팅을 사용하여 질의 처리를 수행한 결과

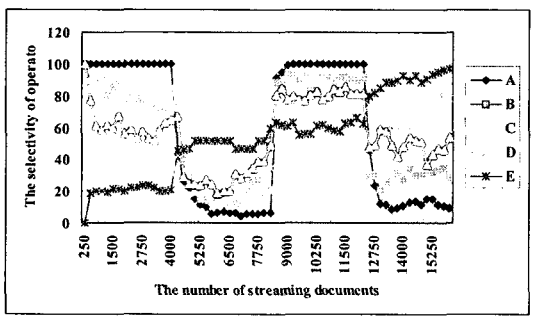

그림 21 선택률 기반 연산자 라우팅에 갱신(Refresh) 기법을 추가한 기법의 수행 결과

하지만 그림 20의 선택률 기반 연산자 라우팅의 실험 결과를 보면 구간 W의 경우 그림 19의 W 구간과 비슷한 결과가 나왔지만 구간 W를 지나 구간 X를 처리하게 되면서 3.3절에서 언급한 각 연산자의 선택률이 누적되는 문제점 때문에 질의 연산자 A~E가 그림 19에 해당되는 결과를 보이지 않음을 알 수 있다. 그림 20의 실험 결과를 사용하여 질의 계획을 세우게 되면 구간 X부터 (E->D->C->B->A)의 질의 계획을 수립하게 된다. 이와 같은 질의 계획은 구간 X~Z에 한해서 최적의 질의 계획이 되지 못한다.

선택률 기반 연산자 라우팅에 갱신 기능을 추가한 기법에 대한 그림 21의 실험 결과를 살펴보면 그림 19의 실험 결과에 해당되는 최적의 경우와 같은 결과는 아니지만, 각 구간에서 측정한 질의 연산자 A~E의 선택률을 사용하여 질의 계획을 세워보면 그림 17의 구간 W~Z에 해당하는 최적의 질의 계획과 동일함을 알 수 있다. 그림 21의 실험 결과에서 구간 X의 경우 연산자 A가 X 구간에 해당하는 스트림 데이타를 처리하는 초기에는 다른 연산자와 유사한 선택률을 가지지만 점차 구간 X에 연산자 A가 가지는 선택률인 5%에 적응해 간다는 것을 알 수 있다. 그림 21의 실험 결과 그래프

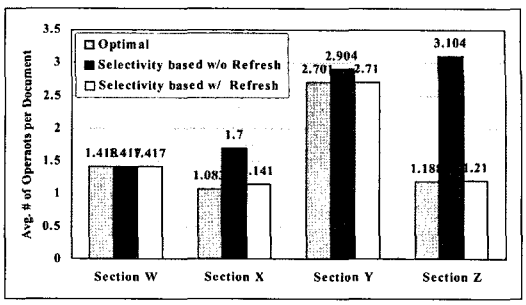

그림 22 구간 W~Z에서 평균적으로 수행된 질의 연산자의 수

자체는 고르지 못하지만 각 질의 연산자마다 각 구간에서 입력되는 스트림 데이타의 선택률에 해당하는 값으로 적응해 나간다는 것을 확인할 수 있다. 그림 22는 그림 19~21의 실험 결과를 사용하여 질의 처리를 수행할 경우 시스템에 입력되는 스트림 데이타를 처리하기 위해 문서당 평균적으로 몇 번의 질의 연산자를 처리하는지를 나타낸 것이다. 각 구간에 따라 측정하였으며 선택률 기반 연산자 라우팅에 갱신 기능을 추가한 기법이 그림 19의 최적의 경우와 유사한 결과를 보인다는 것을 알 수 있다.

4.3 스트림 필터링을 위한 질의 처리 성능 평가

YFilter와의 성능 비교에 사용된 문서는 TPC-W [13]에서 제공하는 주문 정보(Order) 문서이다. Order 문서는 XBench[13,14]에서 사용한 XML 문서 생성기를 사용하여 생성하였으며, 생성한 Order 문서의 전체 크기는 대략 5MB 정도이다. 그리고 각각의 문서 크기는 약 2KB 정도이다. Order 문서는 웹상의 전자상거래 주문 문서로서 그림 23과 같이 상품 주문에 대한 자세한 정보를 포함하고 있다. 성능 평가 실험 결과는 디스크에 존재하는 Order 문서를 하나씩 읽어 본 논문이 제안하는 시스템과 YFilter시스템에 전달하여 모든 Order 문서를 처리하는 데 소요된 총 시간을 측정한 것이다. 실험에 사용된 질의는 표 1과 같다.

그림 24는 YFilter와 본 논문이 제안하는 시스템의 성능을 비교, 평가한 결과이다. 질의 Q1~Q12는 그림 24의 x축에 나타낸 질의 개수 내에서 고른 분포로 존재한다. 그리고 표 1의 질의 조건에 존재하는 %1, %2, %3, %4 는 임의의 값으로 설정하였다. 연산자 마커를 사용하지 않는 경우에는 본 시스템이 YFilter보다 확장성(scalability)은 약간 좋으나 전체적으로 성능이 저하됨을 볼 수 있다. 이와 같은 결과의 원인은 3.5절에서 언급하였다. 연산자 마커 기법을 추가한 후의 실험 결과를 보면 본 시스템의 성능이 YFilter보다 우수할 뿐만 아니라 확장성이 현저히 개선됨을 알 수 있다. 이는 연

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by ToXgene Version 1.1a on Mon Dec 01 14:21:32 KST 2003 -->
<order id="1">
<customer_id>11813</customer_id>
<order_date>2003-11-3</order_date>
<subtotal>399.20</subtotal>
<tax>32.93</tax>
<total>581.13</total>
<ship_type>AIR</ship_type>
<ship_date>2003-11-4</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>2003-11-4</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$5!Nq9arEFF}2,G~(:_PmWb6Jz1(Q~){2#Nt+}GuCm$UU:T
{mI6ze|cQTIY</special_instructions>
</order_line>
<order_line d="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_instructions>
</order_line>
</order_lines>
</order>
```

그림 23 TPC-W의 Order 문서

산자 마커를 통해 불필요한 연산자의 중복 처리를 방지했다는 것과 시스템에 등록된 질의의 개수가 많아짐에 따라 Q10~Q12와 같은 질의의 분포가 증가하는 것에서 찾을 수 있다.

Q10~Q12의 질의는 3개 이상의 조건을 가진다. YFilter의 경우 질의에 존재하는 조건에 대해 순차적인 처리를 하지만 본 시스템의 경우 연산자 라우팅 기법을 사용하여 선택률이 낮은 연산자를 먼저 처리하는 방식을 취하기 때문에 조건식이 많은 질의를 처리함에 있어 YFilter 보다 우수한 성능을 발휘한다. 본 실험에서 본

표 1 실험에 사용된 질의

질의 번호	질의
Q1	/order[@id=%1]/order_lines
Q2	/order[order_date[text()=%1]]/order_lines
Q3	/order[ship_date[text()=%1]]/order_lines
Q4	/order[order_date[text()=%1]][ship_date[text()=%2]]/order_lines
Q5	/order[ship_type[text()=%1]][order_status[text()=%2]]/credit_card_transaction/credit_card_type
Q6	/order[credit_card_transaction/credit_card_type[text()=%1]]/order_lines
Q7	/order[order_date[text()=%1]][credit_card_transaction/credit_card_type[text()=%2]]/credit_card_transaction/credit_card_number
Q8	/order/order_lines/order_line[discount_rate[text()=%1]]
Q9	/order[order_date[text()=%1]][ship_date[text()=%2]][credit_card_transaction/credit_card_type[text()=%3]]/order_lines
Q10	/order[order_date[text()=%1]][ship_date[text()=%2]][order_lines/order_line/discount_rate[text()=%3]]/order_lines/order_line
Q11	/order[order_status[text()=%1]]/order_lines/order_line[discount_rate[text()=%2]]
Q12	/order[order_date[text()=%1]][ship_date[text()=%2]][order_lines/order_line/discount_rate[text()=%3]]/order_lines/order_line[item_id[text()=%4]]

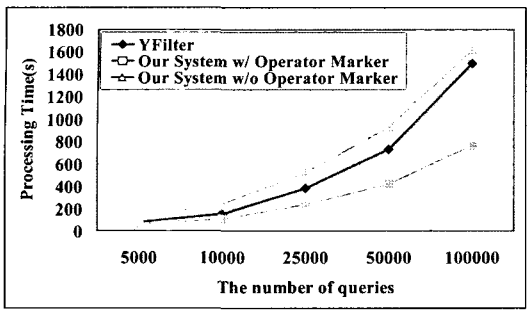

그림 24 XML 스트림 필터링을 위한 질의 처리 성능 평가

시스템의 경우 질의에 포함된 XPath 질의 결과를 스트림 뷰에 등록하는 시간까지 포함하여 YFilter와 성능을 비교하였다. 만약 XML 스트림 데이타에 스트림 뷰가 이미 존재한다면 본 시스템의 성능은 그림 24의 실험 결과보다 훨씬 더 향상될 것이다.

5. 결 론

본 논문에서는 동적으로 변화하는 XML 스트림 데이타에 대한 적응력 있는 질의 처리를 수행하는 시스템을 제안하고 성능을 평가하였다. 적응력 있는 질의 처리를 하기 위해 관계 데이타를 기반으로 하는 시스템의 동적인 질의 처리 모델을 사용하였으며 XML 데이타 스트림에 대한 질의 처리에 이를 적용하기 위해서 해결해야 하는 문제들을 파악하고 그 해결 기법들을 제안하였다.

기존의 XML을 기반으로 하는 스트림 질의 처리 시스템들이 XML 문서에 나타나는 순서로 질의 연산자에 접근하는 데 반해 본 논문에서 제안하는 시스템은 XML 스트림 뷰를 사용하여 질의 연산자를 독립적으로 접근할 수 있게 하였다. 질의 연산자의 독립성을 보장함으로 인해 각 질의 연산자에 대해서 임의의 접근이 가능하여 동적인 질의 처리를 수행할 수 있는 질의 처리 모델을 개발하였다. 또한 임의의 접근이 가능한 연산자들의 선택률을 실시간으로 측정하여 선택률이 낮은 연산자를 우선하여 처리하는 선택률 기반의 연산자 라우팅 기법을 제안하였고 이와 같은 연산자 라우팅 기법에 갱신(Refresh) 기법을 추가하여 시스템에 입력되는 스트림 데이타의 특성에 맞게 연산자의 선택률을 적응력 있게 측정하는 기법을 제안하였다. 또한 연산자 마커(Operator Marker)로 시스템에 등록된 다수의 질의 간의 연산자 처리를 공유하는 기법을 제안하였다. 다양한 성능 실험 결과 본 논문이 제안하는 XML 스트림 질의 처리 시스템이 유동적인 스트림 데이타의 변화에 대해 잘 적응함을 알 수 있었고, 그 결과 인터넷 스케일의 다수 질의에 의한 XML 필터링을 위한 질의 처리에 있어서도 기존의 대표적인 시스템인 YFilter보다 더 좋은 성능과 확장성을 보인다는 것을 확인할 수 있었다.

향후 연구 과제는 다음과 같다. 본 시스템에서 사용하는 XPath 경로 표현식은 '//'와 '*'를 지원하지 못하는 한계가 있다. 이와 같은 기능을 포함한 완전한 경로 표현식을 지원하기 위한 시스템으로의 확장 연구가 필요하다. 또한 본 시스템에서는 질의 연산자에 대해 현재까지 스트리밍된 XML 데이타에 대하여 참으로 판정된 비율로 선택률을 계산하였는데 보다 더 정교한 선택률 계산을 위하여 기존의 관계 데이타베이스의 질의 최적화에서 사용되는 선택률을 현재까지 스트리밍된 XML 데이타에 대하여 적용하는 연구도 필요하다. 한편, 본 시스템에서 사용된 스트림 뷰 기법은 P2P와 같은 분산 환경에서 더욱 유용하므로 본 논문의 시스템을 P2P와 같은 분산 환경에서의 XML 스트림 질의 처리를 위해 확장하는 데 있어 스트림 뷰의 모델, 선택 기법, 그리고 효율성 평가에 대한 연구가 필요하다.

참 고 문 헌

[1] Hellerstein, J., Hong, W., and Madden, S., "The Sensor Spectrum: Technology, Trends, and Requirements," ACM SIGMOD Record, Vol. 32, No. 4, pp. 22-27, 2003.

[2] Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., and Zdonik, S., "Monitoring Streams: A New Class of Data Management Applications," Proc. of Int'l Conf. on VLDB, pp. 215-226, 2002.

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

[4] Diao, Y., Altinel, M., Franklin, M., Zhang, H., and Fischer, P., "Path Sharing and Predicate Evaluation for High-Performance XML Filtering," ACM Transactions on Database Systems, Vol. 28, No. 4, pp. 467-516, 2003.

[5] Green, T., Miklau, G., Onizuka, M., and Suciu, D., "Processing XML Streams with Deterministic Automata and Stream Indexes," ACM Transactions on Database Systems, Vol. 29, No. 4, pp. 752-788, 2004.

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

[7] The STREAM Group, "STREAM: The Stanford Stream Data Manager," IEEE Data Engineering Bulletin, Vol. 26, No. 1, pp. 19-26, 2003.

[8] Abadi, D., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul,

N., and Zdonik, S., "Aurora: A New Model and Architecture for Data Stream Management," VLDB Journal, Vol. 12, No. 2, pp. 120-139, 2003.
[9] Chen, J., DeWitt, D. J., Tian, F., and Wang, Y., "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," Proc. of ACM SIGMOD Int'l Conf. on Management of Data, pp. 379-390, 2000.
[10] Avnur, R. and Hellerstein, J., "Eddies: Continuously Adaptive Query Processing," Proc. of ACM SIGMOD Int'l Conf. on Management of Data, pp. 261-272, 2000.
[11] Gupta, A., Halevy, A., and Suciu, D., "View Selection for XML Stream Processing," Proc. Int'l Workshop on the Web and Databases, pp. 83-88, 2002.
[12] Yoshikawa, M., Amagasa, T., Shimura, T., and Uemura, S., "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," ACM Transactions on Internet Technology, Vol. 1, No. 1, pp. 110-141, 2001.
[13] Yao, B. and Ozsu, M., "XBench-A Family of Benchmarks for XML DBMSs," Lecture Notes in Computer Science, Vol. 2590, pp. 162-164, 2002.
[14] XBench - A Family of Benchmars for XML DBMSs, http://db.uwaterloo.ca/~ddbms/projects/xbench
[15] Transaction Processing Performance Council, http://www.tpc.org
[16] NiagaraCQ's Repository of Docuements, http://www.cs.wisc.edu/niagara/data

김 영 현

2004년 2월 중앙대학교 컴퓨터공학과 졸업(학사). 2006년 2월 중앙대학교 대학원 컴퓨터공학과 졸업(석사). 2006년 5월~현재 삼성전자 디지털 미디어 영상디스플레이 사업부 연구원. 관심분야는 XML 데이타베이스, XML 스트림 데이타 처리 등

강 현 철

1983년 서울대학교 컴퓨터공학과 졸업(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 관심분야는 XML 및 웹 데이타베이스, 이동 데이타베이스, Stream 데이타 관리 등