# The XP-table: 다중 연속 XPath 질의의 집단 처리를 위한 실행시간 효율적인 영역 기반 구조체

## (The XP-table: Runtime-efficient Region-based Structure for Collective Evaluation of Multiple Continuous XPath Queries)

이 현 호 † 이 원 석 ††

(Hyun-Ho Lee) (Won-Suk Lee)

**요 약** XML 메시지 중계기에서의 주요 이슈들 중 하나는 XML 스트림에 대한 다중 연속 XPath질의를 효율적으로 처리하는 방안이다. 본 연구는 이 문제를 효과적으로 해결하기 위한 시스템을 제안한다. 제안되는 시스템에서는 XPath 질의집합을 XP-table이라는 새로운 영역 기반 데이타구조로 변환한다. XP-table은 대상 질의들의 공통적인 선택조건들을 공유하며, 실행시간 질의 수행 전에 구축된다. XML 스트림은 XP-table과의 효율적 매칭을 위해 스트림 릴레이션(SR)으로 실행시간에 변환된다. 제안된 시스템에서는 XML의 구조적 특성을 반영한 XP-table과 SR 간의 효과적인 매칭 전략이 제시된다. 또한, YFilter나 LazyDFA와 같은 기존 방법론과의 비교를 포함한 일련의 실험들을 통해, 제안된 시스템이 질의 처리의 실행시간 부하를 줄임으로써 시간 효율성이 중요한 스트림 환경에서의 안정적 데이타 처리 능력을 보여준다.

**키워드** : XML 스트림, 다중 연속 XPath 질의, XP-table, 영역 기반 구조체, 스트림 릴레이션 (SR)

***Abstract*** One of the primary issues confronting XML message brokers is the difficulty associated with processing a large set of continuous XPath queries over incoming XML streams. This paper proposes a novel system designed to present an effective solution to this problem. The proposed system transforms multiple XPath queries before their run-time into a new region-based data structure, called an *XP-table*, by sharing their common constraints. An *XP-table* is matched with a *stream relation (SR)* transformed from a target XML stream by a SAX parser. This arrangement is intended to minimize the runtime workload of continuous query processing. Also, system performance is estimated and verified through a variety of experiments, including comparisons with previous approaches such as *YFilter* and *LazyDFA*. The proposed system is practically linear-scalable and stable for evaluating a set of XPath queries in a continuous and timely fashion.

**Key words** : XML stream, multiple continuous XPath queries, *XP-table*, region-based structure, stream relation (SR)

· 본 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임 (No.R0A-2006-000-10225-0)

· 이 논문은 2007 한국컴퓨터종합학술대회에서 'XP-table: 다중 연속 XPath 질의의 집단 처리를 위한 실행시간 효율적인 영역 기반 구조체'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 안양과학대학 컴퓨터정보학부 교수
hhlee@ianyang.ac.kr

†† 종신회원 : 연세대학교 컴퓨터과학과 교수
leewo@database.yonsei.ac.kr

논문접수 : 2007년 9월 27일
심사완료 : 2008년 3월 15일



정보과학회논문지: 데이타베이스 제35권 제4호(2008.8)

## 1. 서 론

데이타 스트림은 빠른 속도로 발생되는 데이타 튜플들의 거대하고 무한한 시퀀스로 정의된다[1]. XML 스트림은 데이타 스트림의 특수한 형태로 XML 메시지나 패킷과 같은 논리적 단위들의 무한 집합으로 구성된다. 본 논문에서는 이를 청크(chunk)라 명한다. 청크는 관계형 데이타 스트림의 튜플과 같이 연속 XPtah 질의의 처리 단위이다. XML 스트림의 대표적 적용 사례로 XML 메시지 중계기를 들 수 있는데, 이는 네트워크 상에서 관련 어플리케이션들이 XML 메시지를 교환할 수 있도록 돕는 역할을 한다[2,3]. 데이타 스트림 관리시스템(DSMS)에 등록된 질의는 발생되는 튜플마다 반복적

으로 수행되므로 연속 질의라 한다[1,4]. 연속 질의는 실행시간에 처리되어야 하며, 이는 엄격한 시공간적 제약을 수반한다. XML 메시지 중계기에서의 주요 이슈들 중 하나 또한 XML 스트림을 구성하는 무한의 청크에 대한 다중 연속 XPath 질의의 효율적으로 처리하는 방안이다[5].

많은 수의 연속 질의들이 DSMS에 미리 등록되기 때문에 이들의 공통 조건을 공유하여 집합적으로 처리하는 것이 보다 효율적이다[6-9]. 이러한 목적으로 본 연구에서는 XPath 질의집합을 XP-table이라는 새로운 구조체로 변환한다. 어떠한 XPath 질의집합도 하나의 접두 트리(prefix tree)[10]로 변환될 수 있다. 이러한 점에 착안하여, 주어진 XPath 질의집합은 접두 트리를 거쳐 XP-table로 변환된다. XP-table은 일련의 구성원 리스트(m-list)로 구성된다. 접두 트리의 루트에서부터 비텍스트 단말(non-text leaf) 노드까지의 경로를 기초 경로(base path)라 부르고, 기초 경로에서 변환된 일련의 조각요소(*fragment*)들을 조각 시퀀스(*f-sequence*)라 부른다. 조각요소란 XML의 단위 요소(element) 또는 속성(attribute)를 가리킨다. f-sequence의 인접한 두 조각요소는 반드시 부모-자식관계를 가져야 하는데, 이는 기초 경로를 XML 구조 정보(DTD 또는 XML Schema)를 참조하여 조각요소들 간의 부모-자식 관계로 풀어냄으로써 얻어진다. XP-table의 m-list는 접두 트리의 기초 경로로부터 변환된 f-sequence에 대응하여 만들어 진다. m-list에서는 기초 경로에 표현된 선택조건 (predicate)들을 근거로 대응되는 f-sequence의 전체 도메인은 일정한 수의 배타적 영역(region)으로 분할하여, 각 영역 마다 미리 계산된 매칭 결과를 유지한다. 주어진 XPath 질의집합은 XP-table이 가지는 여러 개의 m-list를 차례로 수행함으로써 집단적으로 처리된다. 또한, XML 스트림의 각 청크는 스트림 릴레이션(SR)이라 불리는 릴레이션 구조로 변환된다. SR은 청크를 구성하는 각 조각요소의 f-sequence와 관련 정보를 튜플로 저장한다. XP-table과 SR은 f-sequence를 매개로 서로 매칭된다.

본 연구의 공헌은 다음과 같이 요약된다.

- 다중 연속 XPath 질의 처리를 위해, 그들의 선택조건을 물리적으로 공유하고 나아가 대상 질의집합의 부분적으로 미리 계산된 매칭 결과를 가지는 XP-table이라는 구체적 구조체를 제안한다.
- XP-table의 모든 구성요소는 질의 수행 전에 모두 구축되므로, 실행시간 부하를 최소화시키는데 기여한다.
- XP-table의 구성요소인 m-list의 영역 구조는 비교연산을 연산 종류에 관계없이 동일한 방식으로 처리하므로 비 동등 연산도 동등 연산만큼 빠르게 처리할 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서 관련 연구를 기술하고 3장에서 본 논문이 다룰 주요 이슈를 정의하며, 4장에서는 제안되는 시스템의 구성요소와 이를 이용한 질의처리 방안에 대한 자세히 설명한다. 5장에서는 시스템 성능이 일련의 실험들을 통해 검증되고 6장에서 결론을 맺는다.

## 2. 관련 연구

본 연구는 데이타 스트림에 대한 연속 질의 처리의 개념에 근거한다. Aurora[4], TelegraphCQ[7] 그리고 NiagaraCQ[6]와 같은 연속 질의 처리 시스템은 일반적으로 다중 질의에 대한 공유 처리를 지원한다. 본 논문에서 제안되는 기법은 다중 질의의 공유 처리라는 점에서 기존 시스템과 유사성이 있으나, 특별히 XML 스트림에 대한 XPath 질의 처리의 관점에서 연구되었다. 대용량 XML 문서에 대한 다중 XPath 질의 처리에 관한 몇몇 연구들이 있다. XFilter[11]는 각 XPath 질의에 대한 유한 상태 머신(FSM)과 질의 색인을 사용한다. YFilter[12]는 모든 XPath 질의들을 하나의 비결정적 상태 오토메타(NFA)로 표현하고 NFA와 실행시간 스택을 사용하여 질의 처리를 수행한다. Index-Filter[10]는 대용량 XML 문서의 특정 부분을 필터링하기 위해 문서 태그에 대한 인덱스를 구축한다. XTrie[13]는 트라이(trie) 구조에 선택조건을 포함한 XPath를 표현하여 질의를 처리한다. LazyDFA[14]는 대상 XPath 질의들을 집합적으로 표현한 하나의 결정적 오토메타(DFA)를 제안하며, XPush[3]도 이와 비슷하게 결정적 푸시다운 오토메타(PDA)를 사용한다. LazyDFA와 XPush는 공통적으로 그들의 오토메타를 질의 수행 중에 점진적으로 구축한다. 이는 상태 수의 급격한 증가를 막기 위해서이다. [15]에서는 동적으로 변하는 스트림 데이타에 대해 적응력 있게 대처하기 위하여 질의 연산자 라우팅 기법을 통해 동적인 질의 계획을 사용함으로써 적응력 있는 질의 처리를 지원하는 방안을 제안하고 있으며, [16]에서는 XML 데이타의 구조 정보를 표현하는 XML 레이블링 기법을 이용하여 XML 데이타를 조각으로 분할하여 조각 스트림에 대한 질의 처리를 수행하는 기법을 제시한다. 그러나, 어떤 연구도 XML 스트림에 대한 다중 연속 XPath 질의 처리를 위해, 본 연구의 XP-table과 같이 질의 수행 이전에 완성되는 질의 공유 구조를 제시하지는 않았다. 또한, 기존 연구는 XML의 구조적 특징의 하나인 비결정성의 문제를 질의 수행 중에 해결하는데, 이는 실행시간 효율성이 중요한 연속 질의 처리 환경에는 적합하지 않다.

XML 스트림에 대한 질의 처리에 관련된 몇몇 연구들은 XQuery 처리[17-19] 또는 스트림 조인[20,21]과 같이 XPath의 표현 범위 이상의 연산을 처리한다. 그러나, 이러한 연구들도 XPath 처리를 기초하는 기법들이다. 왜냐하면, XML에 대한 모든 확장된 질의들의 기초적인 표현식이 XPath이기 때문이다. 제안된 시스템은 질의 대상 XML 스트림을 튜플 기초의 릴레이션으로 변환하여 질의를 처리함으로써 집단화나 조인과 같은 관계형 연산을 지원하기에 충분한 확장성을 제공한다. 이는 앞으로 본 연구가 나아가야 할 방향이기도 하다.

## 3. 문제 정의

본 논문의 전체적 이슈는 지속적으로 들어오는 XML 스트림에 대한 다중 연속 XPath 질의를 어떻게 실행시간 효율적인 집단적 방법으로 처리하느냐이다. 본 논문의 대상이 되는 XPath 질의의 표현 범위와 몇가지 핵심적인 문제들은 다음과 같다.

**대상 XPath 표현범위** XPath 질의는 일련의 위치 단계(location step)들로 구성된다. 각 위치 단계는 축(axis), 노드 테스트(node test) 그리고 0개 이상의 선택조건(predicate)로 구성된다. 축은 두 연속된 노드 사이의 계층적 관계(부모-자식(parent-child) 관계('/') 또는 자손-또는-본인 (descendant-or-self) 관계('//'))를 나타낸다. 노드 테스트는 조각요소 이름, 와일드 카드(*) 또는 텍스트 값(text())을 가리킨다. 와일드 카드는 어떠한 조각요소와도 매치됨을 의미한다. 선택조건은 하나의 완전한 논리 표현식으로서 대괄호([])로 둘러 쌓인다. 하나 이상의 선택조건을 가진 XPath 질의는 두 개 이상의 기초 경로들로 구성된다. 이러한 질의를 비선형 질의라고 한다.

**비결정적 구성요소 문제** XPath 질의의 자손-또는-본인 관계(//)나 와일드 카드(*)와 같은 비결정적 구성요소의 문제를 풀기 위한 여러가지 방법이 제안되었으나, 어떠한 연구도 질의 수행 전에 이 문제를 완전히 해결하지는 못했다.

**재귀적 조각요소 문제** 재귀적 DTD는 동일한 조각요소의 재귀적인 발생을 허용한다. 재귀적 조각요소를 포함한 f-sequence를 재귀적 f-sequence라 한다. 재귀적 DTD에서 제안된 접근 방법은 적절하지 못한 것처럼 보인다. 왜냐하면, 기초 경로에 존재하는 재귀적 조각요소는 무한대의 재귀적 f-sequence를 발생시키기 때문이다.

**비선형 XPath 질의 문제** XP-table의 m-list는 개개의 기초 경로들의 매칭 결과를 가지고 있기 때문에, 비선형 질의를 구성하는 둘 이상의 기초 경로들의 매칭 결과는 논리적으로 결합되어야 한다. 비선형 질의에서 부분적으로 공유되는 기초 경로들의 분리 지점 상의 요소를 가지치기 요소(branching element)라 부른다. 청크 내에 가지치기 요소가 두 번 이상 나타날 경우, 매칭된 공유 기초 경로들 상의 가지치기 요소들의 실체가 동일한 것인지를 판단해야 한다.

## 4. 시스템 아키텍쳐

### 4.1 시스템 총괄

그림 1에서 보듯이, 제안된 시스템은 다음과 같은 세 개의 하위 요소로 구성된다: 질의측 구성요소, 스트림측 구성요소 그리고 질의 수행기. 그림 2는 예제 XML 스트림 $S$, 스트림 $S$에 대한 DTD 그리고, 5개의 XPath 질의로 구성된 질의집합 $Q$를 보여준다. 그림 1의 질의측 구성요소에서 연속 XPath 질의집합은 XP-tree로 컴파일되고, XP-tree는 다시 XP-table로 변환된다. 스트림측 구성요소에서는 SAX 파서가 XML 스트림의 각 청크를 스트림 릴레이션(SR)로 변환환다. 질의 수행기는 XP-table의 m-list들을 차례로 SR과 매칭시키고,

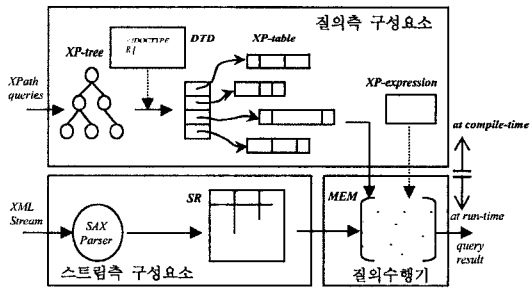



그림 1 시스템 구성도

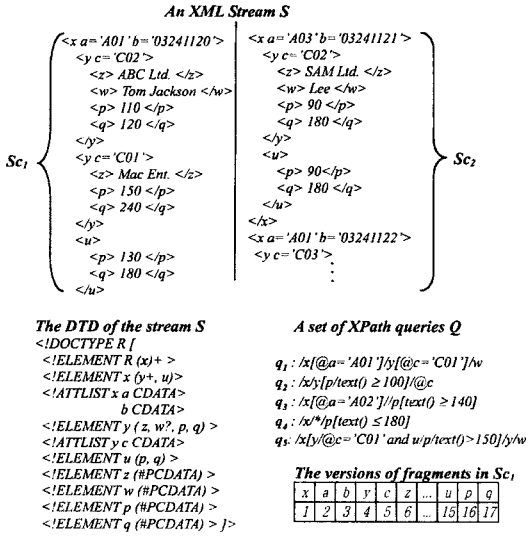



그림 2 예제 XML 스트림과 XPath 질의

매칭 결과를 매칭 연산 매트릭스(MEM)에 저장한다. 각 XPath 질의의 최종 결과는 XP-expression을 연산함으로써 얻어진다. 질의측 구성요소의 모든 데이타구조들은 컴파일 시에 구축되는 반면, 스트림측의 SR은 질의 수행 중에 만들어진다.

### 4.2 질의측 구성요소

연속 XPath 질의집합 $Q$에 대한 접두 트리 $T(Q)=(V_T,E_T)$에서, 함수 $prefix_q(x)$는 질의 $q\in Q$ 상의 루트 요소 $R$부터 조각요소 $x$까지의 일련의 위치 단계를 반환하고, 함수 $branching_q(x)$는 $prefix_q(x)$ 상의 가지치기 요소들의 집합을 반환한다. XPath 질의 $q$=*/a[@b>10]/c[d/text()<20]/e*에 대한 몇가지 예는 다음과 같다: $prefix_q(@b)$=*/a/@b*, $prefix_q(c)$=*/a/c*, $branching_q(@b)=\{a\}$ 그리고 $branching_q(e)=\{a,c\}$. 또한, 함수 $qset(v)$는 노드 $v$와 연관된 질의들을 반환하고, 함수 $vset(q)$는 질의 $q$와 연관된 노드들을 반환한다. 함수 $label(v)$와 $path(v)$는 노드 $v$에 대응되는 조각요소와 노드 $v$의 기초 경로를 각각 반환한다. 제안된 시스템의 질의측 구성요소 중 XP-tree의 정의는 다음과 같다.

**정의 1. (XP-tree)** 연속 XPath 질의집합 $Q$에 대한 접두 트리 $T(Q)=(V_T,E_T)$가 다음 조건을 만족한다면, $XP\text{-}tree(Q)$로 정의된다.

- 단말(leaf) 노드 $v\in vset(q)$는 $qov_set(v)$로 정의되는 $(q,op,val)$ 집합을 유지한다. 단, $q\in qset(v)$이고 $op$와 $val$은 각각 조각요소 $label(v)$에 대한 연산자와 피연산자를 뜻한다.
- 가상 루트 노드 $v_R$을 제외한 비단말(non-leaf) 노드 $v'\in vset(q)$은 $bran_set(v')$으로 정의되는 $(q,V)$ 집합을 유지한다. 단, $label(v')$는 $prefix(label(v))$ $(v\subset V)$ 상의 가지치기 요소들 중 하나이다. 즉, $label(v')\in branching_q(label(v))$. □

그림 3에서 질의 $q_4$와 $q_5$ 상에서의 노드 $v_4$는 $prefix(lable(v_4))$가 대상 청크에 존재하는지 여부만을 파악하므로 $qov_set(v_4)$의 각 $op$와 $val$은 $DC$로 정의된다. $DC$는 *"Don't Care"*의 의미이다. 노드 $v_2$는 $bran_set(v_2)=\{(q_1,\{v_3,v_4\}),(q_2,\{v_4,v_5\})\}$를 가진다. $(q_1,\{v_3,v_4\})$는 $prefix_{q1}(c)$ 상의 요소 $y$의 실체와 $prefix_{q1}(w)$ 상의 요소 $y$의 실체가 같아야 함을 의미한다.

**비결정적 구성요소 문제 해결** $XP\text{-}tree(Q)$의 단말 노드 $v$에 대한 $path(v)$는 DTD를 참조하여 대응되는 f-sequence $fseq(v)$로 변환된다. $path(v)$ 상에 자손-또는-본인 관계나 와일드카드 노드가 존재한다면, $path(v)$를 만족하는 모든 가능한 f-sequence들 찾기 위해 DTD를 깊이우선(depth-first) 방식으로 탐색해야 한다. 반대로, 서로 다른 단말 노드의 기초 경로가 동일한 조각요소를 가리킨다면, 하나의 f-sequence로 변환된다. 이러한 변환은 모두 컴파일 시에 수행된다.

**예제 1.** 그림 3(a)의 노드 $v_6$에 대한 $fseq(v_6)$은 두 개의 멤버 $xyp$와 $xup$를 갖는다. 반대로, 노드 $v_5$, $v_6$ 그리고 $v_8$은 동일한 f-sequence $xyp$로 변환된다. □

**재귀적 조각요소 문제 해결** 재귀적 DTD에 대하여, 목적 XML 스트림의 최대 깊이와 같이 변환될 f-sequence 수를 제한하는 특정한 제약조건이 존재하지 않는다면, 본 연구에서는 최근 처리된 XML 스트림 청크 집합을 근거로, 변환될 f-sequence의 수를 한정함으로써 주어진 질의집합의 정확한 매칭 결과를 근접한다. 한정 f-sequence 집합은 주기적으로 갱신된다. 이 주기를 f-sequence 갱신 주기 $\delta$로 정의한다. f-sequence의 길이는 그것을 구성하는 조각요소의 개수로 정의된다. 그것은 청크에 존재하는 해당 f-sequence에 매칭되는 경로의 깊이와 같다. $max_l$를 가장 긴 f-sequence의 길이라 하고, $max_s$를 최근 f-sequence 갱신 주기 $\delta$ 동안 처리된 청크 집합에서 가장 깊은 경로의 깊이라 하자. 주기 $\delta$가 경과할 때 마다, 한정 f-sequence 집합은 다음과 같은 절차로 갱신된다.

**If** $(max_l < max_s)$ **Then**

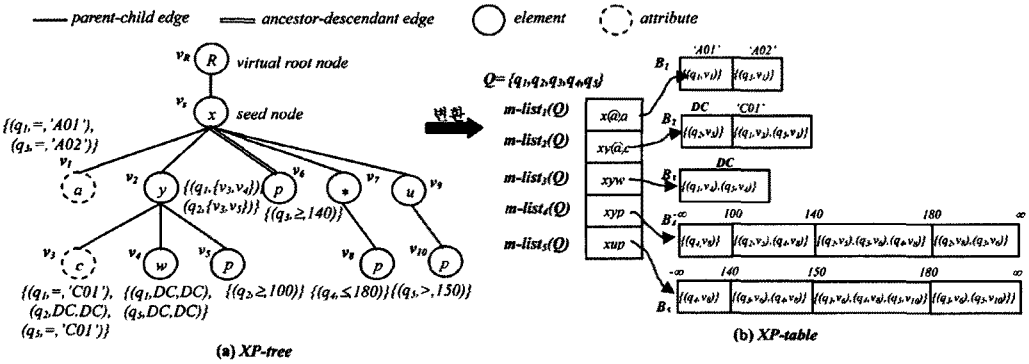



그림 3 그림 2의 다섯 질의에 대한 XP-tree와 XP-table

*new recursive f-sequences whose lengths are shorter than $max_s$ are added to the current set of f-sequences.*

**Else if** *($max_f > max_s + \varepsilon$)* **Then** /* A tolerance parameter $\varepsilon$ controls the frequency of the refinement process. */

*recursive f-sequences whose lengths are longer than $max_s + \varepsilon$ are eliminated from the current f-sequence set.*

**Else**

*there is no change.* □

변환된 f-sequence는 자신만의 m-list를 구축하며, 이러한 m-list들이 XP-table을 형성한다. XP-table은 다음과 같이 정의된다.

**정의 2. (XP-table)** 질의집합 $Q$에 대한 *XP-table(Q)*는 일련의 m-list들로 정의된다. $i$번째 m-list *m-list$_i$(Q)*는 자신의 f-sequence $fseq_i$와 영역배열 $B_i$로 구성된다. *XP-table(Q)*는 다음과 같이 표현된다.

$$XP\text{-}table(Q) = \mathop{\rho}_{i=1}^{r} m\text{-}list_i(Q) = \mathop{\rho}_{i=1}^{r}(fseq_i, B_i) \quad (1)$$

단, $\rho$는 시퀀스이고 $r$은 $Q$에 존재하는 f-sequence의 개수다. 영역배열 $B_i$의 $j$번째 영역 $b_{ij}$는 영역의 범위 정보와 $(q,v)$ 집합 $qv_pair_{bij}$를 유지한다. 각 $(q,v)$는 현 청크에 나타난 f-sequence $fseq_i$의 자식 *text()* 값이 영역 $b_{ij}$에 포함되어 있다면 질의 $q$의 *path(v)*가 현 청크에 의해 만족됨을 뜻한다. □

**정리 1.** $fseq_i$에 대한 선택조건에 포함된 비교상수의 개수를 $m$이라 하면 $fseq_i$의 도메인 $dom(fseq_i)$는 많아야 $(2m+1)$개의 배타적 영역으로 분할된다. 비교상수 자체가 하나의 영역을 이루므로 $m$개의 영역이 생기고, 나머지 도메인은 $m$개의 비교상수에 의해 많아야 $(m+1)$개의 구간영역이 생긴다. □

m-list의 각 영역이 상호 배타적이므로, 목적 스트림 청크에 존재하는 해당 f-sequence의 어떠한 자식 text() 값도 오직 하나의 영역에 속한다. 예외적으로, XP-tree의 *qov_set()*에 있는 $DC$만이 DC-region이라는 자신의 영역을 만드는데, DC-region은 f-sequence의 전체 도메인을 포함한다. DC-region은 소속 $(q,v)$ 들이 모든 영역에 중복적으로 존재함으로써 생기는 공간낭비를 막아준다. m-list가 DC-region을 포함하고 있다면, 청크당 m-list는 많아야 두 개의 영역이 방문된다.

**예제 2.** 그림 3(b)의 *m-list$_2$(Q)*에서, f-sequence *xy@c*가 대상 청크 내에 존재한다면 DC-region 내의 질의 $q_2$를 속성 $c$ 값에 관계없이 만족시킨다. 나아가, 속성 $c$ 값이 *'C01'*이면 질의 $q_2$와 더불어 질의 $q_1$, $q_5$도 만족시킨다. *m-list$_1$(Q)*는 그림 3(a)의 노드집합 $V=\{v_5, v_6, v_8\}$으로부터 변환된다. *m-list$_1$(Q)*는 각각의 범위가 *( - ∞,100), [100,140), [140,180]* 그리고 *(180,∞)* 인 네 개의 영역을 가진다. 이는 *qov_set(V)* = $\{(q_2, \geq,100), (q_3, \geq,140), (q_4, \leq,180)\}$에 의해 전체 도메인이 *100, 140, 180*을 경계로 네 개의 배타적 영역으로 분할된 결과이다. □

**비선형 XPath 질의 문제 해결** 비선형 XPath 질의에서의 가지치기 요소 실체의 일치성 여부를 파악하기 위해서, 질의 수행기는 그림 2와 같이 대상 청크의 각 조각요소 실체에 version이라는 유일한 전역 일련번호를 부여한다. 그림 2에서 질의 $q_1$이 청크 $Sc_1$에 의해 만족되지 못한다는 사실은 해당 청크 내에서 매칭된 속성 $c$의 부모 요소 $y$의 version과 매칭된 요소 $w$의 부모 요소 $y$의 version이 서로 같지 않음을 보임으로써 증명된다. 전자는 *4*이고, 후자는 *10*이다. 각 XPath 질의에 대해서 XP-expression이라는 논리표현식을 정의한다. 비선형 질의에 대한 XP-expression은 해당 질의를 구성하는 기초 경로의 매칭 결과를 논리적으로 조합하는 것뿐만 아니라 version을 통한 가지치기 요소의 version 일치성도 파악한다. XP-expression은 다음과 같이 정의된다.

**정의 3. (XP-expression)** 질의집합 $Q$에 대한 *XP-tree(Q)* = $(V_T, E_T)$가 주어졌을 때, 질의 $q \in Q$에 대한 *XP-expression(Q)*는 다음 두 개의 하위 표현식으로 구성된다: *node_exp(q)*와 *ver_comp(q)*. *node_exp(q)*는 질의 $q$의 모든 기초 경로들의 매칭결과를 조합한다. *ver_comp(q)*는 목적 청크에 매칭된 질의 $q$의 기초 경로들 상의 가지치기 요소의 version을 비교한다. 각 표현식은 다음과 같이 정의된다.

$$node\_exp(q) = \mathop{\varphi}_{v \in vset(q)} M(q,v) \quad (2)$$

노드 $v$는 *XP-tree(Q)*의 단말 노드이고 연산자 $\varphi$는 *AND($\wedge$), OR($\vee$)* 또는 *NOT($\neg$)* 등의 논리연산자를 가리킨다. *M(q,v)*는 *path(v)*가 목적 청크 내에 존재하면서, 자식 *text()* 값이 도메인 *dom(q,v)* 내에 존재할 때 *true*를 반환한다.

$$ver\_comp(q) = \{(|fseq(v')|, V)|(q,V) \in bran\_set(v')\} \quad (3)$$

노드 $v'$은 가지치기 노드이고, $|fseq(v')|$는 $fseq(v')$의 길이로서 $v'$의 위치를 의미한다. $|fseq(v')|=i$라면, f-sequence 집합 $\{fseq(v) \mid v \in V\}$에 속하는 f-sequence들의 $i$번째 요소의 version들이 서로 같은지 비교된다. □

**예제 3.** 그림 3(a)의 질의 $q_1$에 대해, *node_exp($q_1$)*

$= M(q_1,v_1) \wedge M(q_1,v_3) \wedge M(q_1,v_4)$이고 $ver_comp(q_1) = \{(2,\{v_3,v_4\})\}$이다. $ver_comp(q_1)$은 $fseq(v_3)$의 두번째 요소 $y$의 version과 $fseq(v_4)$의 두번째 요소 $y$의 version이 같아야 함을 의미한다. □

### 4.3 스트림측 구성요소

XML 스트림의 각 청크는 SAX 파서에 의해 스트림 릴레이션(SR)로 변환된다. SR은 다음과 같이 정의된다.

**정의 4. (Stream Relation)** 청크 $S_c$에 대한 스트림 릴레이션 $SR(S_c)$는 $S_c$을 구성하는 각 조각요소에 대응되는 튜플(tuple)을 가진 릴레이션이다. $SR(S_c)$의 조각요소 $f$의 튜플은 다음과 같은 속성을 가진다: <f-sequence>, <val> 그리고 <ver-sequence>. <f-sequence>는 $f$에 대한 f-sequence를 저장하고, <val>은 $f$의 자식 text() 값을 저장하며, <ver-sequence>는 <f-sequence>를 구성하는 조각요소들의 version시퀀스를 저장한다.□

SR을 구축하기 위해 SAX 파서는 대상 청크의 모든 조각요소들과 그들의 자식 *text()* 값들을 하나씩 차례로 읽는다. 또한, SAX파서와 더불어 두 개의 전역 변수 $fseq_s$와 $verseq_s$가 사용된다. 변수 $fseq_s$는 파서에 의해 읽혀진 현재의 조각요소에 대한 f-sequence를 저장한다. 동시에, 변수 $verseq_s$는 $fseq_s$에 저장된 f-sequence의 각 조각요소들에 대응되는 version들의 시퀀스를 저장한다. 이들을 이용하여SR을 구축하는데 다음과 같은 세 가지 SAX 파서의 이벤트 함수들이 사용된다: *startElement()*, *characters()* 그리고 *endElement()*. 읽혀진 현재의 조각요소를 $f$라고 했을 때, 함수 *startElement()*는 변수 $fseq_s$에 조각요소 $f$를 추가하여 하나 확장된 f-sequence를 $fseq_s$에 저장하고, 변수 $verseq_s$에 조각요소 $f$에 대한 version을 추가하여 하나 확장된 version 시퀀스를 $verseq_s$에 저장한다. 그리고 나서, *startElement()*는 현 청크의 SR에 새로운 튜플 $(fseq_s, NULL, verseq_s)$를 삽입한다. 함수 *characters()*는 새로 입력된 튜플의 두번째 속성 <*val*> 값을 현 조각요소 $f$의 자식 *text()* 값으로 수정한다. 마지막으로, 함수 *endElement()*는 현재의 $fseq_s$와 $verseq_s$에서 마지막 조각요소와 그것의 version을 각각 제외시킴으로써 하나 짧아진 $fseq_s$와 $verseq_s$를 만든다. 그림 4는 그림 2의 청크 $Sc_1$에 대한 SR을 보여준다. SAX 파서의 이벤트 함수들은 $O(1)$ 시간에 처리되므로 SR을 구축하는데 필요한 시간은 $O(1)$으로 수렴된다.

| <frag_seq> | <val> | <ver_list> |
|---|---|---|
| x | | 1 |
| x@a | 'A01' | 1-2 |
| x@b | '03241120' | 1-3 |
| xy | | 1-4 |
| xy@c | 'C02' | 1-4-5 |
| xyz | 'ABC Ltd' | 1-4-6 |
| xyw | 'Tom Jackson' | 1-4-7 |
| xyp | 110 | 1-4-8 |

| <frag_seq> | <val> | <ver_list> |
|---|---|---|
| xyq | 120 | 1-4-9 |
| xy | | 1-10 |
| xy@c | 'C01' | 1-10-11 |
| xyz | 'Mac Ent' | 1-10-12 |
| xyp | 150 | 1-10-13 |
| xyq | 240 | 1-10-14 |
| xu | | 1-15 |
| xup | 130 | 1-15-16 |
| xuq | 180 | 1-15-17 |

그림 4 그림 2의 청크 $Sc_1$에 대한 SR

### 4.4 질의 수행기

제안된 시스템의 질의 수행기는 목적 스트림의 각 청크에 대해 연속 XPath 질의집합을 집단적으로 수행한다. 그림 5는 질의 수행기에서 XML 스트림 $S$에 대한 XPath 질의집합 $Q$의 매칭 알고리듬을 보여 준다. 그림 5의 함수 *visit_table()*에서는 목적 청크 $S_c$에 대하여 *XP-table(Q)*의 모든 m-list들을 차례로 방문하면서, 속성 <*f-sequence*> 값이 방문된 m-list의 f-sequence와 일치하는 $SR(S_c)$의 튜플을 찾고, 찾은 튜플의 <*val*> 값을 포함하고 있는 해당 m-list의 영역을 스캔한다. 해당 m-list가 DC-region을 포함하고 있다면, 이 영역 또한 스캔된다. 질의집합 $Q$의 각 질의가 목적 청크 $S_c$에 의해 만족되는지 여부를 알기 위해 매칭 연산 매트릭스 (Matching Evaluation Matrix, MEM)이라는 희소 매트릭스를 사용한다. MEM에는 스캔된 영역(들)에 있는 각 $(q,v)$에 대응되는 저장공간이 할당되며, 매칭된 모든 튜플들의 <*ver-sequence*> 값들이 저장된다. MEM은 다음과 같이 정의된다.

**정의 5. (Matching Evaluation Matrix)** 질의집합 $Q$에 대한 *XP-table(Q)*와 목적 청크 $S_c$가 주어졌을 때, $Q$에 대한 $MEM(Q)$는 다음과 같이 정의된다.

$$MEM(Q, S_c) = \rho_{q\in Q} \rho_{v\in vset(q)} (q, v, verseqs(q,v)) \quad (4)$$

단, $\rho$는 시퀀스이고 $verseqs(q,v)$는 청크 $S_c$에서 질의 $q$의 단말 노드 $v$에 매칭된 튜플의 <ver-sequence> 값들의 집합이다. $SR(S_c)$의 튜플 $t$에서 $fseq_t$, $val_t$ 그리고 $verseq_t$를 각각 속성 <f-sequence>, <val> 그리고 <ver-sequence>의 값이라고 하면, $verseqs(q,v)$는 다음과 같이 표현된다.

$verseqs(q,v) = \{verseq_t | fseq_t = fseq(v)$ and $val_t \in dom(q,v)$ for a leaf node $v \in V_T\}$. □ (5)

MEM에 XP-table의 m-list와 SR의 튜플 간의 매칭 결과를 저장한 후, 그림 5의 함수 *eval_expr()*에서는 질의집합 $Q$에 속하는 각각의 질의 $q$에 대해 *XP-expression(q)*의 두 하위 표현식 *node_exp(q)*와 *ver_comp(q)*를 연산한다. *node_exp(q)*의 각 $M(q,v)$는 $MEM(Q,S_c)$의 요소 $(q,v,verseqs(q,v))$에서 $verseqs(q,v) \neq \phi$이면 *true*를 반환한다(그림 5의 23번째 줄). *node_exp(q)*가 *true*이면, *ver_comp(q)*를 연산하고 청크 $S_c$에 대한 질의 $q$의 최종 매칭 결과를 생산한다(그림 5의 24-30번째 줄).

Algorithm ***ExecuteXPathQuery(Q,S)***

***match_qs(f, $S_c$)*** *returns the tuples of $S_c$ whose <f-sequence> is consistent with the f-sequence f.*
***ret_region(B, val)*** *returns the region b of the list B satisfying val $\in [l_b,u_b]$.*
***η(ρ, p)*** *returns the $p^{th}$ versions in a set of versions sequences ρ.*

```
1  while (1) do
2    visit_table(Q, S_c) ; /* visits each m-list in the XP-table(Q) */
3    eval_expr(Q, S_d) ; /* evaluates XP-expressions for the queries in Q */
4    S_c = the next S_c ;
5  end while
6  visit_table(Q, S_c)
7  begin
     /* |m-lists(Q)| returns the number of m-lists in the XP-table(Q) */
8    for (k=1; k<=|m-lists(Q)|; k++) do
9      f = the f-sequence of m-list_k(Q) ;
10     for each tuple t of match_qs(f, S_d) do
11       val = the attribute <val> of the tuple t ;
12       b_1 = the DC-region of m-list_k(Q) ;
13       b_2 = ret_region(m-list_k(Q), val) ;
14       /* scans a matched region */
15       for any pair (q, v) ∈ (qv_pair_b1 ∪ qv_pair_b2) do
16         update the element (q, v, verseqs(q,v)) to (q, v, verseqs(q,v) ∪ {< ver-sequence>|the < ver-
sequence> of the tuple t} in the MEM(Q, S_d)) ;
17     end for each
18   end for
19 end visit_table()
20 eval_expr(Q, S_d)
21 begin
     /* |Q| is the number of queries consisting of the query set Q */
22   for (k=1; k<=|Q|; k++) do
23     r_k = ⊗_{v∈vset(q_k)} M(q_k,v) for the MEM(Q, S_d) ; /*node_exp(q_k)*/
24     if (r_k == true) then /* evaluates ver_comp(q_k) */
25       for any pair (p,V) ∈ ver_comp(q) do
26         for any node v ∈ V do
27           if the same version m, such that m ∈ η(verseqs(q_k,v),p), doesn't exist then
28             r_k = false ;
29         end for any
30     end if
31     output r_k ;
32   end for
33   clear MEM(Q, S_d) ;
34 end eval_expr()
```

그림 5 XML 스트림에 대한 XPath 질의 매칭 알고리듬

**MEM**

| *query* | $v_1$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_8$ | $v_{10}$ |
|---|---|---|---|---|---|---|---|
| $q_1$ | 1-2 | 1-10-11 | 1-4-7 | | | | |
| $q_2$ | | 1-4-5<br>1-10-11 | | 1-4-8<br>1-10-13 | | | |
| $q_3$ | | | | | 1-10-13 | | |
| $q_4$ | | | | | | 1-4-8<br>1-10-13<br>1-15-16 | |
| $q_5$ | | 1-10-11 | 1-4-7 | | | | |

**XP-expression**

| *query* | *node_exp* | *ver_comp* | ***Final Result*** |
|---|---|---|---|
| $q_1$ | $M(q_1,v_1) \wedge M(q_1,v_3) \wedge M(q_1,v_4)$ | $(2, \{v_3,v_4\})$ | → |
| $q_2$ | $M(q_2,v_3) \wedge M(q_2,v_5)$ | $(2, \{v_3,v_5\})$ | → |
| $q_3$ | $M(q_3,v_1) \wedge M(q_3,v_6)$ | | → |
| $q_4$ | $M(q_4,v_8)$ | | → |
| $q_5$ | $M(q_5,v_3) \wedge M(q_5,v_4) \wedge M(q_5,v_{10})$ | | → |

그림 6 그림 2의 청크 $Sc_1$에 대한 다섯 XPath 질의수행 결과

**예제 4.** 그림 6에서, $node_exp(q_1)=M(q_1,v_1) \wedge M(q_1,v_3) \wedge M(q_1,v_4)=true$이다. 그러나, $\eta(verseqs(q_1,v_3),2) \neq \eta(verseqs(q_1,v_4),2)$ 이므로 $ver_comp(q_1)$은 만족되지 않는다. 결과적으로, 질의 $q_1$의 최종 매칭결과는 $false$이다. □

## 5. 실 험

제안된 시스템은 C와 Java로 구현되었다. 모든 실험은 1 Ghz CPU와 1 GM 메인 메모리를 가진 Pentium IV에서 수행되었다. 운영체제는 Linux 9.0을 사용하였다. 실험에는 다음과 같은 두 종류의 실제 데이타셋이 사용되었다: 6MB Protein 데이타셋 $D_1$ (pir.georgetown.edu)과 9MB NASA 데이타셋 $D_2$ (xml.gsfc.nasa.gov). 데이타셋 $D_1$은 비재귀적 DTD를 가졌으며, 최대 깊이는 7이다. 데이타셋 $D_2$는 재귀적 DTD를 가

졌으며, 최대 깊이는 *8*이다. XML 스트림이 무한대의 청크들의 집합이라는 개념을 보다 정확하게 적용하기 위해서, 사용 데이타셋의 구조를 조금 수정하였다. 즉, $D_1$의 요소 <ProteinEntry>와 $D_2$의 요소 <dataset>을 각각의 데이타셋을 구성하는 청크의 최상위 요소의 역할을 하게 하고, $D_1$과 $D_2$ 공히 2,400개의 청크로 구성되게 하였다. 목적 XPath 질의집합의 선택률(selectivity)를 다양하게 변화시키기 위해서 각 데이타셋의 text() 값들은 특정 범위를 가진 수치들로 대치되었다. 사용되는 질의집합은 YFilter(yfilter.cs.berkeley.edu)에 의해 제공되는 XPath 발생기의 수정된 버전에 의해 생성된 인공적 XPath 질의들을 사용하였다.

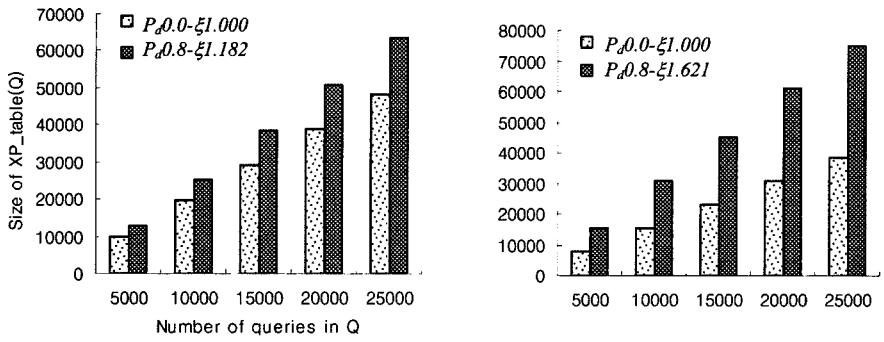
효과적인 실험을 위해서 몇 가지 패라미터들을 정의한다. 접두 트리에서 $k$개의 단말 노드를 가지는 XPath 질의를 *k-degree* XPath 질의라고 한다. $n$개의 *k-degree* XPath 질의집합 $Q = \{q_1, \cdots, q_n\}$에 대해, $Q$에 속한 질의들의 전체 구성요소 중 비결정적 구성요소의 비율을 $p_d$라 한다. *XP-table(Q)*의 각 m-list가 동일한 영역 개수를 가진다는 가정 하에 m-list가 가지는 영역의 수를 $c$라 한다. 그리고, *XP-tree(Q)*의 한 단말 노드가 복수 개의 f-sequence들로 변환되는 정도를 측정하기 위해서, 변환 중복 계수(transformation redundancy coefficient) $\xi$을 정의한다. $\xi$은 *XP-tree(Q)*에 있는 단말 노드의 수에 대한 이것으로부터 변환된 f-sequence의 수의 비율로 정의된다. 더불어, 목적 스트림 청크의 크기의 상태적 측정을 위해 조각요소 반복계수(fragment repetition coefficient) $\omega$를 정의한다. 이는 DTD에 정의된 조각요소들의 수에 대한 청크에 나타난 조각요소의 수의 비율로 정의한다. 두 개수의 차이는 선택적 조각요소와 반복적 조각요소로 인해 발생한다. 실험에 쓰이는 데이타셋 $D_1$의 청크들의 평균 $\omega$는 *0.92*이고, 데이타셋 $D_2$의 평균 $\omega$는 *1.49*이다. 각 실험에 사용되는 패라미터 값은 표 1에 있다.

그림 7은 질의집합 $Q$에 속한 질의 개수 $n$을 변화시키면서 *XP-table(Q)*의 크기를 측정한 결과이다. XP-table의 크기는 이를 구성하는 m-list들의 영역들에 저장된 $(q,v)$의 총 개수로 측정한다. 그림에서 보듯이, $D_2$에 대한 $Q$의 변환 중복 계수 $\xi$가 $D_1$에 대한 $\xi$보다 비결정적 구성요소비율 $p_d$에 더 민감하다. 이는 $D_2$가 재귀적 구조를 가졌기 때문이다. 따라서, $D_2$에 대한 *XP-table(Q)*의 크기가 $p_d$ 값에 따라 더 크게 변동한다. 그러나, 어떤 경우에도 질의의 비결정적 요소가 XP-table 크기에 재앙적 결과를 초래하지는 않는다.

그림 8에서는 $D_1$과 $D_2$의 청크를 질의처리 할 때 필요한 *MEM(Q,S$_c$)*의 평균 크기를 측정하였다. MEM의 크기는 이를 구성하는 각 요소에 저장되는 version 시퀀스

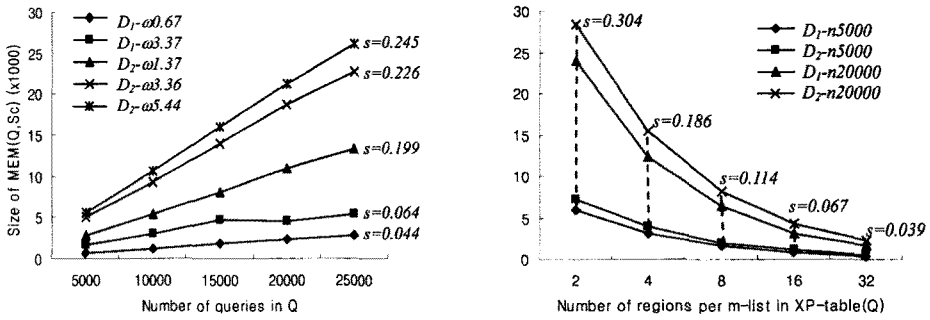
표 1 실험 패라미터

| | $n$ | $c$ | $k$ | | $p_d$ | | $\xi$ | | $\omega$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $D_1$ | $D_2$ | $D_1$ | $D_2$ | $D_1$ | $D_2$ | $D_1$ | $D_2$ |
| 그림 7 | *Varying* | *20* | *2.15* | *1.84* | *Varying* | | *Varying* | | *0.92* | *1.49* |
| 그림 8(a) | *Varying* | *20* | *2.30* | *1.40* | *0.15* | *0.15* | *1.15* | *1.40* | *Varying* | |
| 그림 8(b) | *5K*<br>*20K* | *Varying* | *2.60* | *1.49* | *0.15* | *0.15* | *1.15* | *1.40* | *0.92* | *1.49* |
| 그림 9 | *Varying* | *20* | *2.22* | *1.40* | *0.15*<br>*0.30* | *0.15*<br>*0.30* | *Varying* | | *0.92* | *1.49* |
| 그림 10 | *5K*<br>*20K* | *Varying* | *2.60* | *1.49* | *0.15* | *0.15* | *1.15* | *1.40* | *0.92* | *1.49* |




(a) Dataset $D_1$ (b) Dataset $D_2$

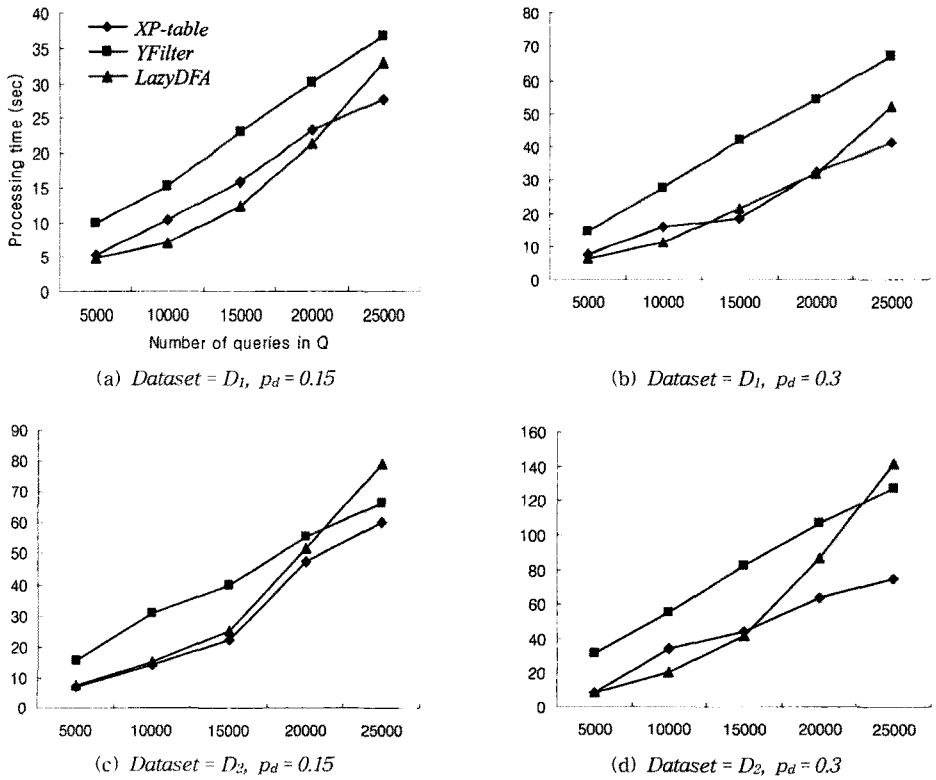
그림 7 *XP-table(Q)*를 위한 필요 공간




(a) varying $n$ and $\omega$ (b) varying $c$ and $n$

그림 8 *MEM-table*($Q_c$,$S_c$)를 위한 필요 공간

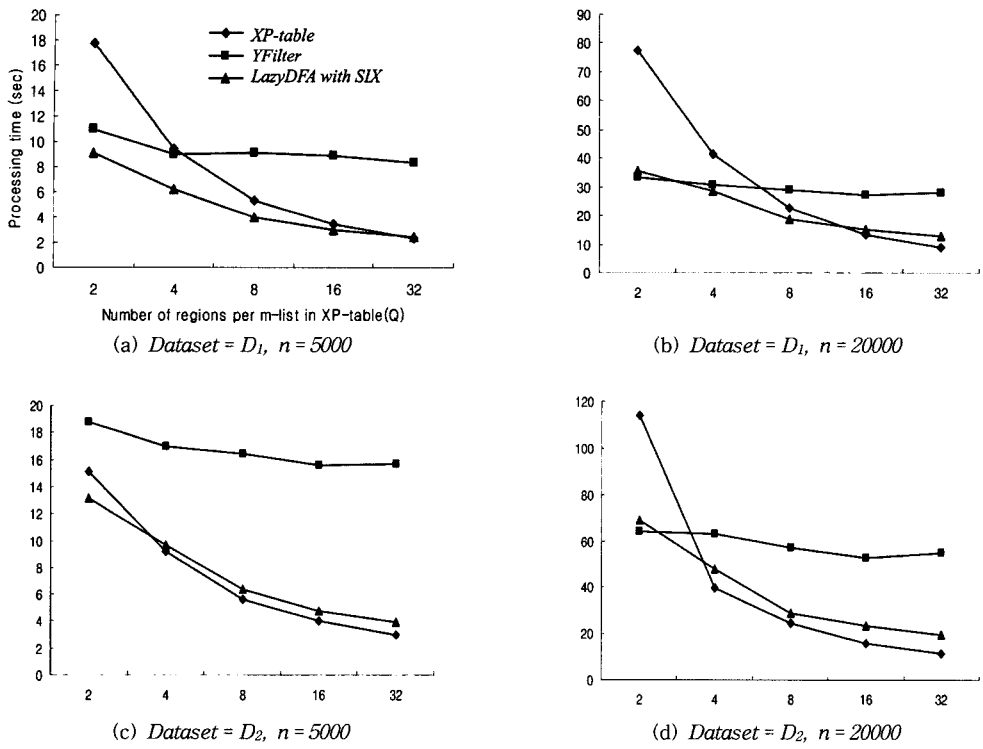
의 총 개수로 측정한다. 그림 8(a)는 $Q$에 속한 질의 개수 $n$을 변화시키면서, 데이타셋과 조각요소 반복 계수 $\omega$에 따른 $MEM(Q,S_c)$의 크기와 질의 선택률 $s$를 보여준다. 그림에서 보듯이 $\omega$는 질의 수행 시 필요공간과 질의 선택률에 중대한 영향을 미친다. 왜냐하면, $\omega$ 값이 크다는 것은 목적 청크에서 다양한 자식 *text()* 값을 가지고 반복적으로 나타나는 조각요소가 많다는 것이고 이는 $MEM(Q,S_c)$ 내의 version 시퀀스의 수를 증가시키며 질의 선택률의 증가로 이어지기 때문이다. 그림 8(b)는 *XP-table(Q)*의 영역 개수 $c$를 변화에 따른 $MEM(Q,S_c)$의 크기와 질의 선택률 $s$의 변화를 보여준다. 그림에서 보듯이, 이 두가지 측정치는 $c$ 값에 반비례한다. 왜냐하면, $c$ 값의 증가는 영역 당 $(q,v)$ 개수를 감소시키고 이는 곧 $MEM(Q,S_c)$가 가지는 version 시퀀스의 수를 감소시키기 때문이다. 데이타셋 사이의 MEM 크기의 격차는 그들에 대한 선택률 차이와 관련이 있다.




(a) *Dataset* = $D_1$, $p_d = 0.15$ (b) *Dataset* = $D_1$, $p_d = 0.3$

(c) *Dataset* = $D_2$, $p_d = 0.15$ (d) *Dataset* = $D_2$, $p_d = 0.3$

그림 9 질의처리시간 (변수: $p_d$ and $n$)




(a) *Dataset* = $D_1$, $n$ = 5000 (b) *Dataset* = $D_1$, $n$ = 20000

(c) *Dataset* = $D_2$, $n$ = 5000 (d) *Dataset* = $D_2$, $n$ = 20000

그림 10 질의처리시간 (변수: $n$ and $c$)

그림 9와 그림 10은 공통적으로 데이타셋 $D_1$과 $D_2$에 대하여 질의집합 $Q$를 처리하는데 걸리는 시간을 측정하고, 이것을 동일한 조건 하에서 기존 방법론인 YFilter [12], LazyDFA[14]와 비교했다. 그림 9는 비결정적 요소 비율 $p_d$를 변화시키면서 $Q$에 속한 질의 개수 $n$에 따른 질의처리시간을 보여준다. 그림에서 보듯이, 질의처리시간을 질의 개수 $n$과 비율 $p_d$에 비례하여 증가한다. 또한, 제안된 방법은 YFilter 보다는 전반적으로 우수한 성능을 보인다. LazyDFA와 비교에서는 상대적으로 작은 $n$ 값과 낮은 $p_d$ 비율 하에서는 제안된 방법이 성능 상 열세에 있다. 그러나, $n$과 $p_d$의 값이 증가될수록 제안된 방법이 LazyDFA 보다 더 효율적으로 안정화됨을 볼 수 있다. 이는 제한된 메모리 공간에서 LazyDFA는 질의 수행 중 DFA를 구축하기 위한 부하가 $n$과 $p_d$ 값이 커질수록 급격하게 증가되는 반면, XP-table은 실행시간 전에 구축이 완료되므로 상대적으로 $n$과 $p_d$ 값의 증가에 덜 민감한 것으로 해석된다.

그림 10은 XP-table의 영역 개수 $c$에 따른 질의처리시간의 변화를 보여준다. 그림 8(a)에서의 실험과 같은 이유로 질의처리시간은 $c$ 값에 반비례한다. 즉, $c$ 값의 증가는 질의 선택률 $s$를 떨어뜨리고 이것이 질의처리시간 감소로 나타난다. YFilter의 경우는 질의 선택률 $s$의 변화가 처리시간에 큰 영향을 미치지 않는다. 이는 YFilter가 선택-연기(selection-postponed) 전략을 사용하기 때문이다. 그러므로, 제안된 방법이 낮은 선택률 환경에서 YFilter보다 더 효율적이다. SIX라는 스트림 인덱스를 가진 LazyDFA의 경우는 제안된 방법과 유사한 결과를 보인다. 상대적으로 높은 선택률 하에서는 LazyDFA가 제안된 방법보다 더 효율적으로 나타난다. 그러나, 선택률 $s$가 낮아지면서 제안된 방법의 효율이 LazyDFA보다는 앞선다. 결론적으로, 제안된 방법이 질의 선택률의 변화에 더 민감하게 반응한다.

## 6. 결 론

본 논문은 XML 스트림 상에서의 다중 연속 XPath 질의를 효율적으로 처리하기 위한 데이타구조와 알고리듬을 제안하였다. XPath 질의의 비결정적 구성요소 문제를 DTD를 참조한 질의 변환을 통해 컴파일 시에 해결하였다. 재귀적 DTD문제는 가장 최근에 처리한 스트림 청크들의 특성을 이용하여 근접(approximation) 방법으로 해결하였다. 또한, 선택조건을 가진 비선형 질의의 문제는 가지치기 요소의 실체에 대한 version 비교라는 독특한 방법으로 해결하였다. 제안된 시스템은 XQuery 처리[17-19], 스트림 조인[20,21] 그리고 집단

화(aggregation)[4]와 같은 좀 더 진전된 주제를 다루기 위한 기초를 제공한다. 제안된 시스템은 질의처리를 위해 XML 스트림을 튜플에 기초한 릴레이션으로 변환하기 때문에 조인과 집단화와 같은 관계형 연산 적용이 용이하다. 향후 연구는 이러한 연산을 적용하기 위한 시스템 확장에 초점을 맞출 것이다.

## 참 고 문 헌


[1] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Mandu, C. Olston, J. Rosonstein, and R. Varma, Query Processing, Resource Management, and Approximation in a Data Stream Management System, Proc. of CIDR Conf., Asilomar, CA, USA, pp.245-256, 2003.

[2] Y. Diao and M. Franklin, Query Processing for High-Volume XML Message Brokering, Proc. of VLDB Conf., pp.261-272, 2003.

[3] A. K. Gupta and D. Suciu, Stream Processing of XPath Queries with Predicates, Proc. of SIGMOD Conf, San Diego, CA, USA, pp.419-430, 2003.

[4] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management, VLDB Journal, vol.12(2), pp.120-139, 2003.

[5] J. Clark and S. DeRose, XML Path Language (XPath) Version 1.0, W3C Recommendation, http://www.w3.org/TR/1999 /REC-xpath-19991116, 1999.

[6] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases, Proc. of SIGMOD Conf., Dallas, Texas, USA, pp.379-390, 2000.

[7] S. R. Madden, M. A. Shah, and J. M. Hellerstein, Continuously Adaptive Continuous Queries over Streams, Proc. of SIGMOD Conf, Madison, Wisconsin, USA, 2002.

[8] K. Munagala, U. Srivastava, and J. Widom, Optimization of Continuous Queries with Shared Expensive Filters, Proc. of VLDB Conf., Seoul, Korea, 2006.

[9] 윤은원, 이원석, 데이타 스트림에서 다중 연속질의의 선택 조건에 대한 실행 순서 결정, 한국정보과학회 2007 한국컴퓨터종합학술대회 논문집 제34권 제1호 (C), pp. 25-28, 2007.

[10] N. Bruno, L. Gravano, N. Koudas, and D. Srivastava, Navigation- vs. Index-Based XML Multi-Query Processing, Proc. of ICDE Conf., Bangalore, India, pp.139-150, 2003.

[11] M. Altinel and M. J. Franklin, Efficient Filtering of XML Documents for Selective Dissemination of Information, Proc. of VLDB Conf., Cairo, Egypt, pp.53-64, 2000.

[12] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer, Path matching and predicate evaluation for high-performance XML filtering, ACM Transactions on Database Systems, vol.28(4), pp.467-516, 2002.

[13] C. Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi, Efficient Filtering of XML Documents with XPath Expressions, VLDB Journal, vol.11, pp.354-379, 2002.

[14] T. J. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu, Processing XML Streams with Deterministic Automata and Stream Indexes, ACM Transactions on Database Systems, Vol. 29, Issue 4, pp.752-788, 2004.

[15] 김영현, 강현철, XML 스트림 데이타에 대한 적응력 있는 질의 처리 시스템, 한국정보과학회, 정보과학회논문지 : 데이타베이스 제33권 제3호, pp. 327-341, 2006.

[16] 이상욱, 김진, 강현철, XML 레이블링 기법을 이용한 XML 조각 스트림에 대한 질의 처리, 한국정보과학회, 한국정보과학회 학술발표논문집 한국정보과학회 2006 가을 학술발표논문집 제33권 제2호(C), pp. 113-117, 2006.

[17] V. Josifovski, M. Fontoura, and A. Barta, Querying XML Streams, VLDB Journal, vol.14, pp.197-210, 2004.

[18] Z. G. Ives, A. Y. Halevy, and D. Weld, An XML Query Engine for Network-Bound Data, VLDB Journal, vol.11(4), pp.380-402, 2002.

[19] H. Su, J. Jian and E. A. Rundensteiner, Raindrop: a uniform and layered algebraic framework for XQueries on XML streams. Proc. of CIKM Conf., New Orleans, Louisiana, USA, pp.279-286, 2003.

[20] L. Ding, E.A. Rundensteiner, and G. Herneman, MJoin: A Metadata-Aware Stream Join Operator, Proc. of DEBS Conf., San Diego, CA, USA, 2003.

[21] T. Urhan and M. J. Franklin, XJoin: A Reactively-Scheduled Pipelined Join Operator, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol.23(2), pp.27-33, 2000.



이 현 호

1993년 연세대학교 전산과학과 졸업(이학사). 1995년 연세대학교 컴퓨터과학과 졸업(이학석사). 1996년~1999년 육군사관학교 교수부 전산학과 전임강사. 2000년~2001년 ㈜엔코아정보컨설팅 선임컨설턴트. 2001년~현재 안양과학대학 컴퓨터정보학부 조교수. 2007년 연세대학교 컴퓨터산업시스템공학과 졸업(공학박사). 관심분야는 데이타 스트림, XML 스트림, 연속 질의 처리, 질의 최적화

이 원 석

1985년 미국 보스턴대학교 컴퓨터공학과(공학사). 1987년 미국 퍼듀대학교 컴퓨터공학과(공학석사). 1990년 미국 퍼듀대학교 컴퓨터공학과(공학박사). 1990년~1992년 삼성전자 선임연구원. 1993년~1999년 연세대학교 컴퓨터과학과 조교수 1999년~2004년 연세대학교 컴퓨터과학과 부교수. 2004년~현재 연세대학교 컴퓨터과학과 교수. 관심분야는 분산데이타베이스, 미디에이터시스템, 데이타마이닝, 데이타스트림