PrimeFilter: 소수 인덱싱 기법에 기반한 효율적 XML 데이타 필터링

(PrimeFilter: An Efficient XML Data Filtering based on Prime Number Indexing)

김 재 훈 † 김 상 욱 †† 박 석 †††
(Jaehoon Kim) (Sangwook Kim) (Seog Park)

요 약 최근 이질적인 시스템 사이에서의 정보교환의 표준으로 널리 사용되는 XML을 사용하는 Publish/Subscribe 시스템의 스트리밍 XML 데이타 필터링 기법이 활발히 연구되었다. 스트리밍 XML 데이타 필터링 기법은 사용자가 등록한 질의에 대해서 신속한 질의-데이타 매칭을 목적으로 하기 때문에 효율적인 질의 처리 메커니즘이 요구된다. 현재까지 대부분의 연구는 질의 경로 표현식의 부분적인 공유를 꾀하거나 프레디킷을 효율적으로 처리함으로써 질의 처리의 시간적, 공간적 효율을 목적으로 하였다. 하지만 만약 질의간의 포함 관계를 알 수 있다면 질의 처리시에 가장 하위의 질의가 매칭되면 그 질의를 포함하고 있는 상위의 질의들은 별도의 처리 과정 없이 매칭됨을 알 수 있게 된다. 이러한 질의 포함 관계를 이용한 질의 처리 방식은 XML 스트리밍 데이타를 처리하는 또 하나의 효율적 방식이 될 수 있다. 본 논문에서는 소수 인덱싱 기법과 목표 질의 노드 중심의 포함 관계 설정에 기반하여 효율적인 스트리밍 XML 데이타 필터링을 수행하는 새로운 방법을 소개한다. 그리고 몇 가지 실험을 통하여 기존 방법과의 비교 분석 및 효율성을 보인다. 비록 각각의 실험은 서로 다른 실험 요소에 대하여 수행되었지만, 모두 제안 방법이 기존의 방법보다 두 배 이상 더 나은 성능을 가짐을 보여 주었다.

키워드 : XML 데이타 필터링, 질의 포함 관계, 소수 인덱싱

Abstract Recently XML is becoming a de facto standard for online data exchange between heterogeneous systems and also the research of streaming XML data filtering comes into the spotlight. Since streaming XML data filtering technique needs rapid matching of queries with XML data, it is required that the query processing should be efficiently performed. Until now, most of researches focused only on partial sharing of path expressions or efficient predicate processing and they were work for time and space efficiency. However, if containment relationship between queries is previously calculated and the lowest level query is matched with XML data, we can easily get a result that high level queries can match with the XML data without any other processing. That is, using this containment technique can be another optimal solution for streaming XML data filtering. In this paper, we suggest an efficient XML data filtering based on prime number indexing and containment relationship between queries. Through some experimental results, we present that our suggested method has a better performance than the existing method. All experiments have shown that our method has a more than two times better performance even though each experiment has its own distinct test purpose.

Key words : XML data filtering, query containment, prime number indexing

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10609-0) 지원으로 수행되었음

† 정 회 원 : 서강대학교 컴퓨터공학과 교수
chris3@sogang.ac.kr
†† 정 회 원 : 삼성전자 정보통신총괄 연구원
rlatd@sogang.ac.kr
††† 종신회원 : 서강대학교 컴퓨터공학과 교수
spark@dblab.sogang.ac.kr

논문접수 : 2008년 3월 19일
심사완료 : 2008년 7월 11일

정보과학회논문지: 데이타베이스 제35권 제5호(2008.10)

1. 서 론

최근 저장되어 있는 데이타가 아닌 끊임없이 연속적으로 빠르게 지나가는 스트리밍 데이타(streaming data)에 대한 연구가 활발하다: YFilter[1], AFilter[2], PosFilter[3]. Publish/Subscribe 시스템[4]은 사용자가 자신의 관심 사항의 프로파일을 등록하면 해당 정보를 각 사용자에게 빠르게 제공하는 것을 목적으로 하는 시스템이다. 예로, 실시간 주식 데이타 모니터링, 온라인 뉴스, 온라인 경매와 같은 도메인에서 사용자가 자신의 관심 있는 주제를 등록하면 해당 정보를 사용자에게 바로 피드백 한다. 이 시스템의 경우 인터넷 등의 발달로 인해 서버에 많은 수의 정보가 실시간으로 빠르게 들어오기 때문에 효율적인 스트리밍 데이타의 처리가 매우 중요하다.

스트리밍 XML 데이타 필터링 시스템은 정보교환의 표준 언어로 널리 사용되고 있는 XML을 대상으로 하는 Publish/Subscribe 시스템의 일종이다. 이러한 시스템에서 사용자의 프로파일은 XPath 언어[5]로 표현되며, 보다 효율적인 데이타 필터링을 위하여 XPath 질의의 부분적인 공유를 활용하거나[1-3], XPath 질의의 프레디킷(predicate)을 효과적 처리하는 기법들[6,7]이 많이 제안되었다.

본 연구에서는 이러한 기존 기법들과는 달리 소수 인덱싱을 이용한 질의들 간의 포함 관계에 기반한 효율적 XML 데이타 필터링 기법을 제안한다. 만약 질의들 간의 포함 관계를 알 수 있으면 가장 하위의 질의에 필터링 된 데이타는 그 질의를 포함하고 있는 상위의 질의들에도 자동적으로 필터링 되게 된다. 따라서 질의들 간의 포함관계를 효율적으로 파악할 수 있다면 질의간의 부분적인 공유 못지 않게 중복된 필터링 처리 시간을 효과적으로 줄일 수 있다. 질의들 간의 포함 관계를 효율적으로 파악하기 위하여 본 논문에서는 소수 인덱싱 기법을 제안한다.

본 논문의 구성은 다음과 같다. 우선 2장에서 스트리밍 XML 데이타 필터링과 관련한 기존 연구들을 살펴본다. 3장에서 본 논문의 연구 동기 및 연구 문제에 관하여 언급하고, 4장에서는 논문의 아이디어를 설명한다. 5장에서는 PrimeFilter의 소수 인덱싱 구성과 필터링 실행에 대하여 설명하며, 6장에서는 기존 YFilter와의 비교 실험 및 결과를 보인다. 마지막으로 7장에서 본 연구의 결론 및 추후 연구를 기술한다.

2. 관련 연구

기존 연구들을 살펴보면 질의 경로 매칭의 부분적인 공유를 통한 처리 효율의 향상과 프레디킷의 효율적인 처리등을 살펴 볼 수 있다. 본 논문에서는 이와 다른 관점인, 소수 인덱싱에 의한 효율적 질의 포함 관계 평가에 기반한 효율적 XML 필터링 기법을 제안한다.

우선 질의 경로 매칭의 부분적인 공유에 대한 대표적 연구로 YFilter[1]를 살펴 볼 수 있다. YFilter는 각 XPath 질의를 하나의 비결정적 오토마타(Nondeterministic Finite Automaton: NFA)로 보고 각 오토마타에 대해서 질의의 공통되는 앞부분을 공유하는 방법 (prefix sharing)을 사용하여 질의들을 하나의 비결정적 오토마타로 만들게 된다. 예로, '/a/b/c/d', '/a/b/c/e', '/a/b/e'의 경우 앞 경로 표현식 '/a/b/c' 혹은 '/a/b'가 공유된다. YFilter는 질의의 앞부분을 공유하고 있기 때문에 같은 엘리먼트에 대해서 질의에 따라 각각 계산을 해주는 것이 아니라 한번만 계산을 해주면 된다. 따라서 처리량이 좋아지며 메모리의 사용량도 줄어든다는 장점이 있다.

PosFilter[3]는 YFilter와는 반대로 질의의 뒷부분을 공유(postfix sharing)한다. 예로, '//a/b/c/d', '//b/c/d', '//e/c/d'의 경우 뒷 경로 표현식 'b/c/d' 혹은 'c/d'를 공유한다. PosFilter는 질의 뒷부분 공유가 빈번한 환경에서의 효율적인 NFA 기반의 XML 필터링 기법을 제안한다.

AFilter[2]는 질의의 앞 공유 및 뒷 공유를 적응적(adaptively)으로 활용하는 방법을 제안한다. 즉, 같은 뒷 경로 표현식을 갖는 모든 질의들을 한꺼번에 경로 매칭 하다가 앞 공유가 유리하다고 판단된 경우에는 각 질의를 다시 앞 경로 표현식에 따라 묶은 후 경로 매칭을 계속해서 수행한다. 하지만 AFilter는 오토마타 기반의 기법이 아니며, 특별히 고안된 메모리상의 자료 구조(AxisView, PRLabel-tree, SFLabel-tree, StackBranch) 및 경로 매칭 알고리즘을 사용한다. 따라서 전통적인 오토마타 기법을 활용하는 YFilter와 PosFilter와는 달리 손쉬운 필터링 시스템의 구현 및 유지 보수 등의 이점이 적다고 할 수 있다.

XTrie[8]는 복잡한 XPath 경로 표현식을 하나의 스트링으로 표현함으로써 단순하며 효과적인 필터링 시스템을 구현한다. 즉, 각 경로 표현식은 스트링으로 표현되고, 모든 스트링이 하나의 트라이(trie) 구조로 공유되며 묶여 진다. XML 데이타 스트림이 입력될 때 이 또한 하나의 스트링으로 표현되며, 트라이 인덱스를 이용한 효율적 부분 문자열(substring) 탐색이 이루어진다.

질의 경로에서의 프레디킷의 효율적 처리 연구로는 TwigM[6]과 FiST[7]를 살펴볼 수 있다. 먼저 TwigM은 재귀적(recursive) 데이타를 처리할 때 pathM과 branchM을 통해서 프레디킷 및 조상-후손 관계의 경로 축(ancestor-descendant axis)을 갖는 질의의 질의 처리 시간을 PTIME으로 감소시켰다. FiST는 XTrie와

같이 XPath를 하나의 스트링 시퀀스로 표현함으로써 질의 경로와 프레디킷을 한 번에 처리함으로써 확장 용이한(scalable) 질의 처리 기법을 제안하였다.

3. 연구 동기 및 문제점

만약 질의간의 포함 관계가 미리 계산되어 있다면, 가장 하위의 질의가 XML 문서에 매칭된다면 그 질의를 포함하고 있는 상위의 질의들은 별도의 처리 과정 없이 XML 문서에 매칭됨을 알 수 있다. 따라서 질의들 간의 포함관계를 파악한다면 YFilter와 PosFilter와 같은 질의의 부분적인 공유 기법만큼 중복된 XML 데이타 필터링의 오버헤드를 크게 줄일 수 있다. 예를 들어 $q1$: '/a/b/c/d', $q2$: '/a/b//d', $q3$: '/a//d', $q4$: '//d'라는 질의들이 있을 때 네 질의는 $q1 \subseteq q2 \subseteq q3 \subseteq q4$와 같은 포함 관계를 가지게 된다. 이 때 XML 문서가 가장 하위의 질의인 $q1$과 매칭된다면 그 상위의 질의들 $q2$, $q3$, $q4$와도 역시 매칭 됨을 자동적으로 알 수 있다. 하지만 이러한 아이디어는 XML 스트리밍 환경에 적용할 때 다음 문제점을 가지고 있다.

문제점: XML 스트리밍 환경에서는 일반적으로 적게는 수백 개에서 많게는 수십만 개의 질의가 등록되어 있다. 이 때 질의가 n개가 등록되어 있다고 한다면 질의들 간의 포함 관계를 알기 위해서는 n^2 번의 비교가 이루어져야 한다. 이렇게 모든 질의를 각각 한 번씩 비교하는 것은 굉장한 비효율을 초래하며 현실적으로 불가능이라 할 수 있다. 따라서 최대한 질의간의 비교를 줄이는 것이 중요하다. 이를 위해 본 연구에서 제안하는 소수 인덱싱 기법을 이용하여 비교할 필요가 없는 질의를 효율적으로 제거할 것이다.

본 XML 필터링 시스템에서 사용 가능한 XPath 질의 유형은 다음과 같다.

(1) 비록 XPath 질의[5]에는 다양한 탐색 축(axis)이 정의되어 있지만, 본 연구에서는 사용자들이 '/', '//', '*', '[]'의 보다 단순한 탐색 축만을 사용하여 XPath 질의를 쉽게 작성하는 경향이 있음을 가정한다. 따라서 XP(/, //, *, [])의 XPath 경로 표현식만을 고려한다.

(2) 프레디킷('[]')과 관련된 XPath 트리 질의에 관하여서는, YFilter[1]에서 제안된 질의 경로 분해(query decomposition) 기법을 이용한 필터링 방법을 가정한다. 예로, 질의 '/a[b]//c[d/e]/f'는 하나의 주경로(main path) '/a//c/f'와 두 개의 프레디킷 경로(predicate path) '/a/b', '/a//c/d/e'로 분해될 수 있다. 세 개의 단일 경로 질의가 XML 필터링 시스템에 각각 등록되고, 만약 유입되는 XML 문서에 대해 세 개의 단일 경로 질의가 모두 매칭될 때, 그 문서는 필터링된다. 따라서 본 연구에서는 단일 경로 질의의 필터링에 초점을 맞춘다.

4. 문제 해결 방안

본 장에서는 3장에서 제시한 문제점에 관하여 질의간의 포함 관계 및 소수 인덱싱 기법을 이용하여 해결하고자 하는 구체적 아이디어를 제시한다. 5장에서는 이러한 아이디어에 기반한 PrimeFilter 시스템의 동작 과정을 설명한다.

4.1 목표 질의 노드 중심의 포함 관계

XML 필터링은 어떤 주어진 XPath 질의의 목표 질의 노드(target query node)가 존재하는 지를 탐색하는 것이므로, 본 연구에서는 목표 질의 노드의 포함 관계를 살펴본다.

정의 1 (목표 질의 노드).

- 목표 질의 노드(target query node)는 어떤 주어진 XPath 질의의 궁극적 질의 결과가 되는 질의의 마지막 노드이다. 예로, '/a//c/d'의 경로 표현식에서 목표 질의 노드는 마지막 노드인 'd'이다.

목표 질의 노드 관점에서의 포함 관계는 두 가지로 고려될 수 있다. 하나는 깊이(depth)에 의한 포함 관계이며, 다른 한 가지는 너비(width)에 의한 포함 관계이다. 이 두 가지 관계는 아래의 그림 1을 참고하면 더욱 쉽게 이해할 수 있을 것이다. 예를 들어 $q1$: '/a/b/c/d', $q2$: '/a//c/d', $q3$: '/a/b/c' 세 질의가 있을 때 $q1$과 $q3$의 관계가 깊이에 의한 포함 관계이며, $q1$과 $q2$의 관계가 너비에 의한 포함 관계이다. 하지만 XML 필터링에서 질의 포함 관계를 갖는 두 XPath 질의의 질의 결과

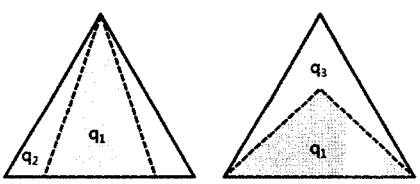


그림 1 목표 질의 노드 관점의 두 질의 포함 관계

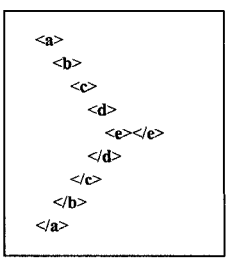


그림 2 XML 예제

는 같아야 하므로(즉, 두 질의의 목표 질의 노드가 같아야 하므로), 본 연구에서는 너비에 의한 포함관계만을 고려한다. 예로 그림 2와 같은 XML 문서의 경우, $q1$과 $q2$는 'd'에서 매칭 되어 질의 처리 결과로 <d><e></e></d>를 얻게 되나, $q3$은 'c'에서 매칭 되어 다른 질의 처리 결과인 <c><d><e></e></d></c>를 얻게 된다.

4.2 소수 인덱싱 기법

소수 인덱싱 기법을 소개하기에 앞서 소수가 가지는 특성을 간단히 살펴보자. 본 연구에서는 Wu et al.[9]에서 연구된 XML 소수 레이블링 기법의 아이디어를 질의 포함 관계에 활용한다. 아래에서 A와 B는 두 정수이고 모두 소수만을 인자로 가진다. 또한 A는 B보다 큰 수이다.

특성 1 (Divisibility).

- 만약 A가 B로 나누어지지 않는다면, B는 A가 가지고 있지 않은 인자를 가지고 있다. 예를 들어 3은 6에 속해있는 소수이지만, 10에는 속해있지 않기 때문에 10은 6으로 나누어지지 않는다.

특성 2 (Inclusion).

- 만약 A가 B에 의해 나누어진다면, A는 B가 가지고 있는 모든 인자를 포함하고 있다. 예를 들어 42와 21이라는 수가 있을 때, 42는 21로 나누어지고, 42는 2, 3, 7을 인자로 가지게 되고, 21은 3, 7을 인자로 가지게 되며 42는 21의 인자인 3과 7을 모두 가지고 있다.

특성 3 (Overlapping).

- 만약 A가 임의의 소수 a로 n번 나누어진다면, A는 a라는 인자를 n개 가지고 있다. 예를 들어 18이라는 정수가 있을 때 18은 3으로 두 번 나누어지고, 인자는 2, 3, 3을 가지므로 3이라는 인자를 2개 가지고 있다.

위의 특성들을 이용해서 효율적 질의 포함 관계 평가를 위한 소수 인덱싱 기법을 소개한다.

정의 2 (XPath 질의 소수 레이블링).

- 임의의 XPath 질의가 등록이 되었을 때 각 질의 노드들은 노드 이름에 따라 고유한 소수 값을 갖게 된다. 이때 각 질의 노드의 소수 레이블 값은 바로 앞 노드의 소수 값에 자기 노드에 부여된 소수값을 곱한 것이다. 조상-후손 경로('//')나 와일드카드('*')는 소수 레이블링과는 무관하다.

예로 $q1$: '/a/b//c/d' 질의가 먼저 등록 된 후 $q2$: '/a/b/e/*/f' 질의가 등록되는 것을 가정하자. 우선 $q1$의 첫 질의 노드 'a'에는 소수 2가 부여된다. 다음 $q1$의 'b'에는 6 (= 2 * 3)이 부여된다. 마찬가지로 $q1$의 'c'는 30 (= 2 * 3 * 5), $q1$의 'd'는 210 (= 2 * 3 * 5 * 7)이 부여된다. 두 번째 질의 $q2$가 들어오면 우선 첫 노드인 'a'는 앞 질의에서 소수 2가 부여되었으므로 2가 부여되고, 마찬가지로 b는 앞 질의에서의 6, 다음 노드인 e가 입력되면 e에 소수 11이 새로이 부여되고 부모 노드의 소수 레이블 값 6과 11을 곱해 66이 부여된다. 마지막 노드인 f 역시 다음 소수 13이 부여되고 소수 레이블 값으로 858 (= 2 * 3 * 11 * 13)이 설정된다. 여기서 목표 질의 노드는 조상 질의 노드들에 부여된 모든 소수들의 곱을 인자로 갖고 있다.

이제 두 질의 간의 포함 관계 평가는 목표 질의 노드의 소수 레이블 값을 바탕으로 이루어질 수 있다. 두 질의 P, Q가 있고 P가 Q를 포함한다면, 두 질의 P, Q간의 포함 관계가 성립할 조건은 다음과 같다.

i) Q의 목표 질의 노드의 소수 레이블링 값은 P의 목표 질의 노드의 소수 레이블링 값으로 나누어 떨어져야 한다.

ii) 두 질의가 공통으로 가진 각 질의 노드의 순서가 같아야 한다.

iii) 또한 공통 질의 노드들의 스키마 구조가 같아야 한다 (이 문제는 질의 경로 축 '//' 와 '*'에 의해서 기인함).

만약 i)이 성립 하지 않는 다면 특성 1 [divisibility] 성질에 의해서 두 질의는 서로 관계가 없다는 것을 알 수 있다. 반대로 i)가 성립하게 된다면 특성 2 [inclusion] 성질에 의해서 아직 포함 관계의 여부는 알 수 없지만 P의 인자들이 Q에 포함된다는 사실을 알 수 있다. 따라서 i)는 두 질의의 포함 관계의 필요조건이라 할 수 있다. 다음으로 i)가 성립하여도 질의 노드의 순서가 다르면 질의 포함 관계를 만족할 수 없다. 예로, '/a/b//*/c'와 '/a/c/b'는 목표 질의 노드 'c'와 'b'가 같은 소수 레이블링 값을 갖지만, 포함 관계에 있진 않다. 마지막으로 i) ii)를 둘 다 만족한다 하더라도, 질의 노드들의 스키마 구조가 다르면 질의 포함 관계를 만족할 수 없다. 예로, '/a/b/*/c'와 '/a/b/c'의 필터링 결과는 다르다.

세 조건을 확인하기 위한 방법은 아래와 같다.

i) Q의 목표 질의 노드 소수 레이블링 값을 P의 목표 질의 노드 소수 레이블링 값으로 나누어 본다. 나머지가 0이 아닐 경우 포함 관계가 없다는 것을 알 수 있다.

ii) 우선 두 질의 P, Q가 있을 때 목표 질의 노드의 소수 레이블링 값이 작은 P의 질의 노드들을 기준으로 질의를 분해한다. 이 경우 P의 질의 노드를 찾고 그에 해당하는 Q의 질의 노드를 순서대로 찾는다. 만약 P에 해당하는 질의 노드들이 모두 Q의 질의 노드들과 순서상 매치되는 지를 확인한다. 만약 조상-후손 경로 축('//')으로 시작하는 질의가 올 경우, 두 질의 앞에 임의의 루트 노드를 설정하여 처리하도록 한다. 예를 들어 $q1$: '/a/b/c', $q2$: '//c'라는 두 질의가 있을 때 임의로 질의를 $q1'$: '/r/a/b/c', $q2'$: '/r//c'로 변형하여 처리하

도록 한다.

iii) 아래와 같이 두 질의 P, Q의 공통 질의 노드들의 스키마 매칭을 수행한다. 먼저 P와 Q의 공통 질의 노드를 다음과 같이 표현하자.

$P = \{x_1, x_2, x_3, \cdots, x_{n-1}, x_n\}$

$Q = \{y_1, y_2, y_3, \cdots, y_{m-1}, y_m\}$

그러면 P의 질의 노드들을 기준으로 부분 경로를 계산할 수 있다. 즉, x_1에서 x_2까지의 경로를 p_1으로 설정하고 x_2에서 x_3까지의 경로를 p_2로 설정하고 같은 방식으로 x_n까지의 설정을 하고, 질의 Q 역시 P의 질의 노드들을 기준으로 $q_1, q_2, \cdots, q_{n-1}$으로 설정할 수 있다.

$P = \{p_1, p_2, p_3, \cdots, p_{n-2}, p_{n-1}\}$

$Q = \{q_1, q_2, q_3, \cdots, q_{n-2}, q_{n-1}\}$

예로, 그림 3의 $q1$: 'a/b/*/c/d//e/*/f'와 $q2$: 'a//d//*//f'를 살펴보자. 우선 $q2$의 질의 노드들이 $q1$의 질의 노드들에 포함되기 때문에 $q2$를 기준으로 부분 질의 경로를 나누게 된다. $q2$의 경우 질의 노드 {a, d, f}를 가지기 때문에 부분 경로 {a-d, d-f}의 두 구간을 고려할 수 있다. $q1$ 역시 {a, d, f}에 따라 두 구간으로 나누어진다.

다음으로 각 부분 경로 구간에 대하여 다음을 계산한다. 만약 부분 경로 구간에 조상-후손 경로 축 ('//')이 오게 되면, 조상-후손 경로 축은 0개 이상의 엘리먼트를 자유롭게 포함할 수 있으므로, n을 부여한다. 조상-후손 경로 축이 여러 개 존재할 경우, 개수에 상관없이 n으로 계산한다. 와일드카드 ('*')의 경우 한 개의 엘리먼트가 자유롭게 올 수 있으므로 1의 수만큼 더해준다. Q의 경우 만약 공통 질의 노드 외에 다른 질의 노드가 존재할 경우 와일드카드처럼 1로 계산한다. P를 기준으로 계산된 부분 경로 깊이를 Q에 비교하여 질의 포함 관계를 판별한다.

예로 그림 3에서, 우선 d-f구간을 보면 $q1$의 경우 경로 표현식 '//e/*'를 가지며 조상-후손 경로 축을 n으로 계산하고 와일드카드, 'e'를 각각 1로 계산하여 n+2가 된다. n은 0이상의 엘리먼트가 올 수 있다는 의미이고 따라서 $q1$의 d-f구간은 두 개 이상의 엘리먼트가 올 수 있게 된다. $q2$의 경우 '//*//'를 가지며 두 개의 조상-후손 경로 축을 n으로 계산하고 와일드카드를 1로 계산하면 n+1이 된다. 이것은 $q2$의 d-f구간이 1개 이상의 엘리먼트를 가질 수 있음을 의미한다. 따라서 $q2$는 'e'를 포함하지 않는 XML 경로 또한 부분 매칭할 수 있으므로 $q1 \subseteq q2$의 관계가 성립한다. a-d구간을 살펴보면 $q1$은 3이 되고 $q2$는 n이 된다. 역시 $q2$의 a-d구간은, 'b'와 'c'를 포함하지 않는 XML 경로를 부분 매칭할 수 있으므로, $q1$의 a-d구간을 포함하게 된다. 질의의 모든 부분 경로에서 $q2$가 $q1$을 포함하고 있기 때문에 최종적으로 질의 $q2$와 질의 $q1$의 스키마 구조가 일치함을 알 수 있다.

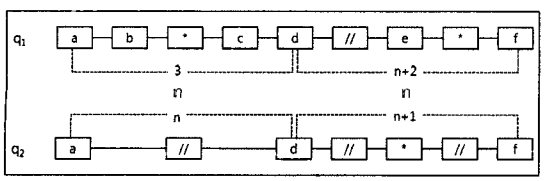


그림 3 두 질의 간의 부분 경로 구간별 질의 포함 관계

5. PrimeFilter

5.1 XPath 질의 등록

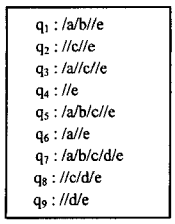
5.1.1 NFA를 이용한 각 XPath 질의의 목표 질의 노드의 레이블 값 계산

임의의 XPath 질의가 등록될 경우 XPath 질의의 목표 질의 노드의 소수 레이블 값이 계산될 필요가 있다. 목표 질의 노드의 레이블 값 계산을 위하여 NFA를 사용한다. PrimeFilter의 NFA 구성은 YFilter [1]의 NFA 구성과 유사하다. 단지 NFA의 각 오토마타 상태(state) 노드에 정의 2에서 소개한 각 질의 노드의 소수 레이블 값이 부여된다. PrimeFilter의 NFA를 구성하는 알고리즘은 그림 4와 같으며, 그림 5의 예를 가지고 설명하도록 하겠다.

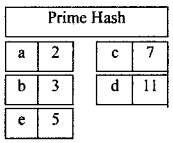
먼저 $q1$: '/a/b//e'라는 질의가 들어오면 'a'에 소수 2가 부여되며 그림 5(c)와 같이 새로운 상태(state) 2가 생성된다. 이때 그림 5(b)의 소수 해쉬 테이블에 (a, 2)가 저장된다. 계속해서 'b'에는 소수 3이 부여되고 2와 3의 곱인 6을 갖는 상태가 생성된다. 다음으로 조상-후손 경로축 ('//')에 대해서는 YFilter와 같이 ε-상태 전이(state transition) 및 셀프-루프(self-loop)를 갖는 새로운 상태가 생성된다. 이때 새로운 상태에는 이전 레이블 값인 6을 그대로 가져오게 된다. 마지막으로 'e'가 들어오면 소수 5가 부여되고 이 전과 같은 방식으로 상태

```
for (x in Query_Nodes(q_i))  // 어떤 XPath 질의 q_i의 각 질의 노드 x에 대하여 수행
{
    Get the prime number of x from Prime_Hash;
    if (Prime number of x is null){
        if(x != '//' && x != '*'){
            Register the next prime number into Prime_Hash;
            Create new state of NFA;
        }else
            Create new state of NFA;
    }else{
        Check the state transition by x
        if (there is the state transition by x)
            Follow the state transition;
        else{
            Create new state of NFA
        }
    }
}
```

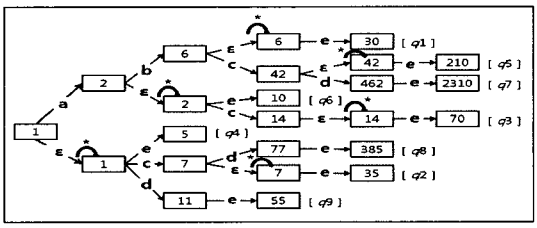
그림 4 PrimeFilter NFA 구성 알고리즘



(a)



(b)



(c)

그림 5 소수 레이블 상태 값을 갖는 NFA 구성

30이 생성된다. 상태 30은 질의 $q1$의 마지막 상태(final state)이므로 해당 질의 번호가 저장된다. 그림 5(b)의 소수 해쉬 테이블에는 (b, 3)과 (e, 5)가 추가적으로 저장되었다. 두 번째 질의 $q2$: '//c//e'가 들어오면 처음에 '//'가 고려되고 이전 상태 1을 그대로 가져온다. 다음 'c'에 대해선 소수 7이 부여되며 상태 7 (= 1 × 7)이 생성 된다. 그림 5(b)의 소수 해쉬 테이블에 (c, 7)이 새로이 저장된다. 다음 '//'에 대해선 ε-상태 전이 및 셀프-루프를 갖는 상태 7이 생성된다. 'e'에 대해선 소수 해쉬 테이블에 이미 할당된 소수 값 5가 존재하므로, 이를 이용하여 마지막 상태 35 (= 7 × 5)가 생성된다. 다음 $q3$: '/a//c//e'가 들어오면 처음 노드인 'a'는 이전에 오토마타 상태를 구성했기 때문에 해당 상태 2로 이동하게 된다. 다음으로 '//'에 대해서 상태 2가 새로이 생성되며, 'c'에 대해서 상태 14, '//'에 대해서 상태 14, 'e'에 대해서 마지막 상태 70(= 14 × 5)이 생성된다. 나머지 질의 예제도 위와 같은 과정으로 구성됨을 알 수 있다.

5.1.2 목표 질의 노드들의 마지막 상태들에 대한 소수 인덱싱

본 절에서는 PrimeFilter의 핵심 구성 요소인 NFA에서의 마지막 상태들에 대한 소수 인덱싱(혹은 소수 인덱싱 그래프라고 불림)에 대하여 설명하도록 하겠다. 소수 인덱싱 그래프는 앞서 소개한 PrimeFilter NFA 구성에서 목표 질의 노드들이 동일한 NFA 경로의 포함 관계에 해당하는 지를 신속히 알기 위하여 미리 구성한 인덱스라고 볼 수 있다.

예로, 그림 6(e)는 그림 5(a)의 샘플 질의들에 대한 목표 질의 노드 'e'에 대한 소수 인덱싱 그래프를 보여준다(사실 그림 5(a)의 샘플 질의들은 모두 목표 질의 노드 'e'를 갖는다). 그림 5(c)에서 $q2$:'//c//e'는 마지막 상태 35를 갖고, $q8$:'/c/d//e'는 마지막 상태 385를 갖는다. 그림 6(e)에서 385는 35로 나누어지므로 35의 후손 노드이고, $q2$와 $q8$은 너비에 의한 질의 포함 관계를 갖는 것을 알 수 있다. 즉, 385의 모든 조상 노드들은 $q8$과 너비에 의한 질의 포함 관계를 갖는다. 따라서 입력되는 XML 문서에서 일련의 XML 엘리먼트 <c><d><e>를 읽을 경우, 'e'에 대한 소수 인덱싱 그래프에서 385를 찾고 385를 포함한 모든 조상 노드들의 해당 XPath 질의가 매칭 질의임을 한번에 알 수 있다.

지금부터는 소수 인덱싱 그래프의 생성 과정을 설명한다. 먼저 입력된 질의와 기존 소수 인덱싱 그래프에 등록된 질의들과의 포함 관계를 고려할 때 두 질의 사이에서의 포함 관계 경우의 수는 표 1과 같이 정리된다. 사실 이러한 내용은 앞서 4.2절에서의 두 질의 P와 Q간

표 1 입력 질의와 소수 인덱싱 그래프에 등록된 질의 간의 비교

pn(P)와 *pn*(Q)의 소수 값의 비교	*pn*(P) mod *pn*(Q)	P, Q 사이의 질의 포함 관계
pn(P) > *pn*(Q)	*pn*(P) mod *pn*(Q) = 0	P ⊆ Q
		질의 포함 관계 없음.
	pn(P) mod *pn*(Q) ≠ 0	질의 포함 관계 없음.
pn(P) = *pn*(Q)	*pn*(P) mod *pn*(Q) = 0	P ⊆ Q
		P ⊇ Q
		질의 포함 관계 없음.
pn(P) < *pn*(Q)	*pn*(P) mod *pn*(Q) = 0	P ⊇ Q
		질의 포함 관계 없음.
	pn(P) mod *pn*(Q) ≠ 0	질의 포함 관계 없음.

P: 새로 입력된 XPath 질의
Q: 이미 소수 인덱싱 되어 있는 XPath 질의
pn(P), *pn*(Q): P와 Q의 NFA에서의 마지막 상태의 소수 레이블 값
before_Q: Q 이전에 포함 관계를 비교한 질의
next_Q: Q 이후에 포함 관계를 비교한 질의
bflag: P의 before_Q가 확정되었음을 표시하는 상태 변수, 초기 값은 true이며 확정되었을 때 false 값을 가짐.

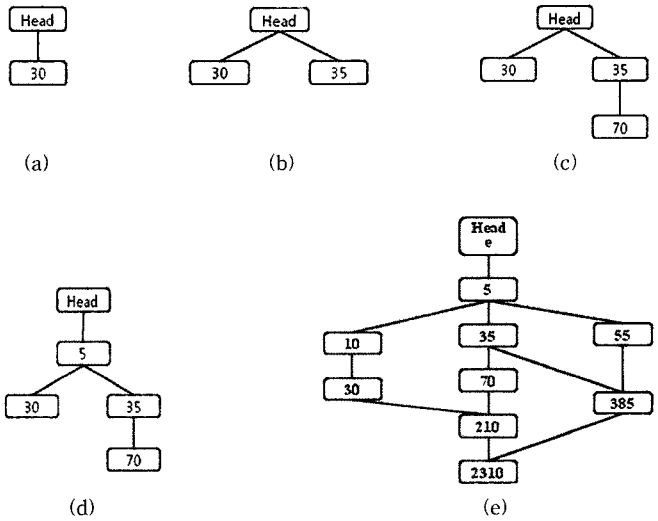


그림 6 마지막 상태 노드들에 대한 소수 인덱싱 그래프 생성

의 포함 관계 성립 조건에서 이미 설명된 것이다.

표 1에서 첫 번째 열은 새로 입력된 질의와 소수 인덱싱 그래프 노드들과의 소수 레이블 값의 비교이다. 두 번째 열은 두 소수 레이블 값이 나누어지는 가를 따지는 것이며 세 번째 열은 질의 포함 관계를 따지는 것이다. 이를 위해 4.2절의 질의 포함 관계 성립 조건 ii), iii)에서 설명한 방법을 수행한다. 최종 P ⊆ Q, 포함 관계 없음, P ⊇ Q가 판정되었을 경우 소수 인덱싱 그래프의 새로운 노드 추가를 위하여 다음 과정이 수행된다.

• P ⊆ Q

P가 before_Q와 연결되었을 때 연결을 삭제한다. 그리고 새로이 Q를 P의 before_Q로 만든다. 소수 인덱싱 그래프에서의 다음 노드에 대한 Q와 계속적으로 비교해 나간다.

• 포함 관계 없음

Q의 before_Q의 next_Q가 P가 되도록 한다. 이때 P의 *bflag*는 false가 된다. 소수 인덱싱 그래프에서의 다음 노드에 대한 Q와 계속적으로 비교해 나간다.

• P ⊇ Q

Q를 P의 next_Q로 만든다. Q의 before_Q가 P를 next_Q로 갖도록 만든다. 계속해서 next_Q의 하위 노드들을 탐색하여 P를 next_Q로 갖는 연결을 삭제한다.

예로, 그림 5(a)에서의 첫 질의 $q1$: 'a/b//e' (30)가 입력되면 'e'에 대한 소수 인덱싱 그래프가 아직 생성이 되어 있지 않기 때문에 그림 6(a)와 같이 헤드 노드를 만들고 $q1$을 연결한다. 다음 질의 $q2$: '//c//e' (35)가 입력되면 소수 인덱싱 그래프에 연결된 $q1$과 비교한다: 여기서 $q2$는 P이고 $q1$은 Q이며, $q1$의 before_Q는 헤드 노드이다. 두 소수 레이블 값, 30과 35는 서로 나누어지지 않으므로 포함 관계가 없는 질의이므로, Q의 before_Q 즉 헤드 노드가 P를 가리키도록 한다(그림 6(b)를 참조). 세 번째 질의 $q3$: '/a//c//e' (70)이 들어오면 앞서의 $q2$와 마찬가지로 $q1$과 비교하고 포함 관계가 없기 때문에 Q의 before_Q 즉 헤드 노드가 P인 $q3$을 가리키도록 한다. 계속적으로 소수 인덱싱 그래프에서의 다음 노드 Q인 $q2$와 비교가 이루어진다. $q3$은 $q2$로 나눠지고 P ⊆ Q 포함관계에 있기 때문에 P = $q3$은 헤드 노드, 즉 before_Q와 연결을 끊고 Q = $q2$를 P의 before_Q로 만든다(그림 6(c) 참조). 네 번째 질의 $q4$: '//e' (5)가 들어오면 우선 $q1$과 비교한다. $q4$는 $q1$을 포함하므로 P ⊇ Q 과정에 의해서 Q = $q1$를 P = $q4$의 next_Q로 만들고 $q1$이 연결되어 있던 Q의 before_Q, 즉 헤드 노드와의 연결을 해제한다. 이때 Q=$q1$의 before_Q, 헤드 노드의 next_Q가 P = $q4$가 되도록 한다. 이러한 과정이 소수 인덱싱 그래프의 모든 Q 노드에 대하여 재귀적으로 수행된다. 그림 6(e)는 그림 5(a)의 샘플 질의에 대해 만들어진 최종 소수 인덱싱 그래프이다.

5.2 XML 필터링 실행

XML 문서가 들어오면 시작 엘리먼트를 읽을 때마다 해당 엘리먼트를 목표 질의 노드로 갖는 소수 인덱싱 그래프가 존재하는 지를 확인한다. 이러한 확인은 목표 질의 노드들에 대한 해쉬 인덱스를 사용하여 신속히 처리할 수 있다. 만약 존재할 경우 현재까지 입력된 일련의 XML 엘리먼트에 대한 소수 레이블 값을 계산하여, 해당 소수 인덱싱 그래프에서의 상태 노드를 상향식(bottom-up)으로 검색한다. 검색된 상태 노드의 모든 조상 상태 노드들에 대한 질의가 입력 데이타에 대한 매칭 질의이다.

예로 그림 2의 <a><c><d><e></e></d></c>의 XML 엘리먼트 입력을 고려하자. 'a', 'b', 'c', 'd' (462 = 2 × 3 × 7 × 11)를 읽을 때까지는 해당 소수 인덱싱 그래프가 존재하지 않는다. 'e' (2310 = 462 × 5)를 읽을 경우 그림 6(e)의 'e'를 목표 질의 노드로 갖는 해당 소수 인덱싱 그래프가 존재하므로, 그래프에서 2310의 상태 노드를 검색한다. 2310은 그림 5(a)에서 *q7*: '/a/b/c/d/e' 질의이고, 소수 인덱싱 그래프에서의 질의 포함 관계에 의하여 그림 5(a)의 모든 질의가 매칭 질의임을 한번에 파악할 수 있다.

6. 성능 평가

본 장에서는 XPath 질의의 부분적인 공유의 대표적인 방법인 YFilter와의 성능 비교 실험 결과를 제시한다. 관련 연구에서 소개한 다른 XML 필터링 방법의 경우 최적화된 소스 코드를 구할 수 없었기 때문에 YFilter를 통한 본 제안 방법의 효율성을 가늠해 본다. PosFilter[3]의 경우 YFilter의 소스 코드를 일부 변형하여 구현된 것이므로 이의 성능 비교는 YFilter를 통하여 정확히 가늠해 볼 수 있다.

6.1 실험 환경

본 논문에서 제안하는 기법은 모두 자바로 구현되었으며 비교 대상의 YFilter의 경우 공개된 소스코드[10]가 있기 때문에 본 실험에서는 그것을 그대로 사용했다. 실험은 인텔 코어2 6300 1.86GHz 프로세서와 1GB DDR2 메모리가 장착된 마이크로소프트 윈도우 XP 운영체제에서 자바 가상 머신 1.5.0을 이용하여 수행되었다. 또한 자바 가상 머신이 사용하게 될 가상 메모리의 최초 및 최대 힙(heap) 크기를 512MB 로 설정하여 2차 보조기억장치의 I/O가 발생하지 않도록 하였다. 이를 위해 모든 실험에서는 마이크로소프트 윈도우 XP의 작업관리자를 통해 페이지 파일 사용량과 실제 메모리 사용량을 확인하여 2차 보조기억장치의 I/O가 발생하지 않음을 명백히 하였다. 마지막으로, 별도의 쓰레드(thread)를 생성하여 성능 결과를 측정하였다.

• 시험 부하 생성

표 2 NITF, DBLP DTD의 특성

	NITF	DBLP
엘리먼트 이름의 수	123	36
전체 속성의 수	510	14
엘리먼트의 반복 (recursion)	존재	존재

PrimeFilter나 YFilter 모두 XML 문서와 질의 매칭을 위해 XML 스키마 정보가 필요 없다. 다만 본 실험에서는 스키마 정보를 실험을 위한 데이타 생성에 사용한다. XML 스키마는 YFilter 실험에서 사용된 NITF (News Industry Text Format)를 주로 사용하였다. 또한 추가 실험 데이타로 DBLP DTD를 사용하였다. 본 실험에서 사용한 NITF, DBLP DTD의 특징은 다음 표 2와 같다.

NITF, DBLP DTD에 따른 XML 문서의 랜덤 생성을 위해 본 실험에서는 IBM의 XML generator를 사용했다. 데이타 생성에는 두 가지의 매개 변수를 입력하는데 하나는 최대 깊이이고 다른 하나는 엘리먼트의 반복 횟수이다. 엘리먼트 반복 횟수의 경우 범위를 3으로 제한하였다. XPath 질의 생성은 YFilter에서 구현한 XPath generator[10]를 사용하였다. 표 3은 XPath 질의와 XML 문서 데이타 생성을 위한 실험 매개 변수이다. 실험에서 사용하는 일반 질의는 최대 깊이가 10이며 평균 깊이는 5이다. 그리고 조상-후손 경로('//')와 와일드카드('*')의 비율은 10%로 설정하였다.

• 실험 측정 기준

본 실험의 측정 단위는 YFilter에서 정의한 다중-질의 처리 시간(multi-query processing time, MQPT)이며 그 정의는 다음과 같다.

정의 3 (다중-질의 처리 시간, MQPT).

필터링 시스템에 XML이 입력되는 시점을 *tstart*, 입력된 XML의 SAX 파싱이 완료된 시점을 *tparsing*, SAX 파싱 이벤트를 이용하여 구조 질의와 XML 데이타가 최종 매칭되는 시점을 tend라 하자. 그럼 MQPT = *tend* - *tparsing*이다.

이러한 다중-질의 처리 시간을 올바로 이해하기 위해 주의할 점은 필터링 시간의 정의인 tend-tstart와 달리

표 3 XPath 질의와 XML 문서 데이타 생성을 위한 실험 매개 변수

매개 변수	범위	의미
Q	1000 ~ 100000	질의 수
D	6 ~ 10	최대 질의 깊이
AD	3 ~ 10	평균 질의 깊이
W	0 ~ 1	XPath 경로상의 임의의 위치에서의 wildcard "*" 발생 빈도
DS	0 ~ 1	XPath 경로상의 임의의 위치에서의 조상-후손 경로 축 "//" 발생 빈도
P	0 ~ 20	질의당 프레디킷의 수
RP	2, 3, 5	동일 엘리먼트 반복 횟수

MQPT에는 XML을 파싱하는데 걸리는 시간이 포함되지 않는다는 것이다.

6.2 실험 1: 총 질의 수에 따른 데이타 처리 시간 비교

본 실험을 통해 질의의 수가 증가함에 따라 PrimeFilter와 YFilter의 질의 처리 시간이 어떻게 변하는지를 알아본다. 실험에서 사용한 질의의 최대 깊이는 10이며 평균 깊이는 5이다. 그리고 조상-후손 경로('//')와 와일드카드('*')의 비율은 10%로 설정하였다. 질의의 수는 10,000~100,000 개의 질의를 1,000 단위로 측정하였다. 그림 7을 보면 시간에 따른 증가 폭은 두 기법 모두 비슷하다. PrimeFilter의 경우 YFilter보다 좋은 MQPT값을 나타낸다.

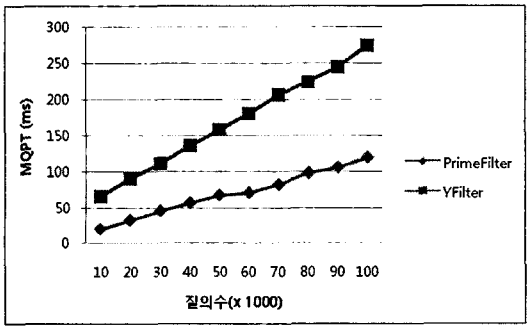


그림 7 질의 수에 따른 MQPT 비교(NITF, D = 10, W = 0.1, DS = 0.1)

6.3 실험 2: 질의 등록 및 XML 필터링

본 논문의 제안방법은 질의 등록 과정이 YFilter에 비해 비효율적이라는 단점이 있다. 이는 질의 간의 포함 관계 설정을 위한 과정이 요구되기 때문이다. 따라서 본 실험에서는 질의 등록 시간과 필터링 시간을 총괄적으로 측정하였다. 입력되는 XML 문서의 수는 50개이며, 질의 수 Q = 10,000 또는 50,000개로 하여 실험하였다. 그림 8의 결과를 보면 질의 등록 시간은 YFilter에 비해 PrimeFilter가 크지만 필터링 시간은 적음을 알 수 있다. 또한 전반적 필터링 시스템의 성능은 PrimeFilter가 앞섬을 알 수 있다.

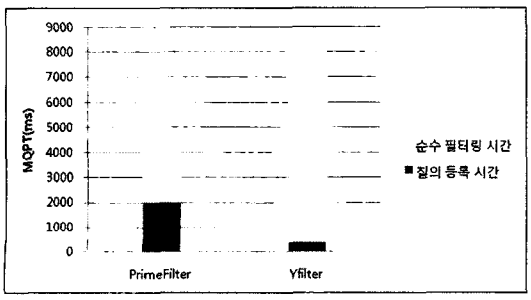


그림 8 질의 등록 및 필터링 처리 시간 비교(NITF, Q = 50,000)

6.4 실험 3: 평균 질의 깊이에 따른 필터링 시간 비교

이 실험을 통해 질의의 평균 깊이에 따라 PrimeFilter와 YFilter의 MQPT값이 어떻게 변하는지 알아본다. 실험에서 사용한 XPath 질의는 질의 깊이가 3과 10사이이고, 조상-후손 경로 축('//')과 와일드카드('*')의 비율은 각각 10%로 설정되었다.

그림 9를 보면 YFilter의 경우 평균 질의 깊이가 깊어질수록 MQPT의 값이 증가한다. 반면 PrimeFilter의 경우 평균 질의 깊이가 깊어질수록 MQPT의 값이 감소한다. 이는 PrimeFilter의 경우 어떤 질의가 매칭될 경우 포함 관계에 있는 나머지 질의의 매칭 관계도 한번에 파악되기 때문이다. 사실 이러한 이점 때문에 PrimeFilter는 평균 질의 깊이에 영향을 받지 않음을 확인할 수 있었다.

재미있는 사실은 PrimeFilter의 경우 평균 질의 깊이가 증가할수록 MQPT의 값이 감소하는데, 그 원인은 질의의 깊이에 상관없이, 평균 질의 깊이가 낮을 경우 XML 문서에 매칭되는 질의의 수가 보통 증가하며 반대로 평균 질의 깊이가 클 경우 매칭 질의는 적게 존재하기 때문이다.

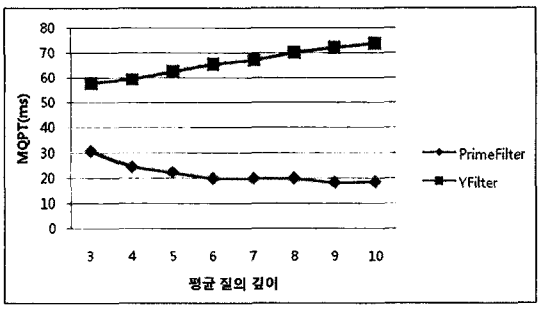


그림 9 질의 깊이에 따른 MQPT 비교(NITF, W = 0.1, DS = 0.1)

6.5 실험 4: 조상-후손 경로 축과 와일드카드의 비율에 따른 필터링 시간 비교

본 실험을 통해 질의에서의 '//'와 '*'의 비율에 따른 PrimeFilter와 YFilter의 MQPT 변화를 알아본다. 본 실험의 목적은 YFilter가 조상-후손 경로 축과 wild-card의 비율 증가에 따라 영향을 받지만, PrimeFilter의 경우 영향을 받지 않는 것을 보여주기 위함이다. 이는 YFilter의 NFA의 경우 '//'과 '*'에 의해서 스택에서의 오토마타 상태 수의 폭발적 증가 현상이 발생하지만, PrimeFilter는 목표 질의 노드의 소수 인덱싱 그래프의 존재 여부만을 확인하기 때문이다. 실험에서 사용한

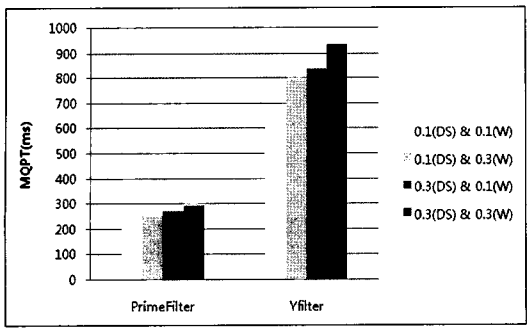


그림 10 DS와 W의 비율에 따른 MQPT 비교(NITF, D = 10, AD = 5)

XPath 질의의 최대 깊이는 10이며 평균 깊이는 5이다. 그리고 조상-후손 경로 축과 와일드카드의 비율은 각각 10%와 30%를 조합하여 설정하였다.

그림 10을 살펴보면 YFilter의 경우 DS와 W의 비율이 증가함에 따라 MQPT의 값도 증가함을 알 수 있다. 이는 조상-후손 경로 축('//')이 비결정적 오토마타에 들어 있을 경우 셀프 루프에 의해 계속 스택에 쌓이는 상태의 폭발적 증가 현상이 일어나기 때문이다. 또한 와일드카드('*')가 비결정적 오토마타에 들어 있을 경우도 잘못된 경로의 전이가 증가하기 때문에 역시 처리 시간이 증가하는 문제를 일으킨다. PrimeFilter의 경우 포함 관계를 이용하기 때문에 조상-후손 경로 축과 와일드카드의 비율은 필터링 시간에 크게 영향을 주지 않는다.

6.6 실험 5: DBLP 데이타에 대한 추가 실험

NITF DTD 외에 DBLP DTD를 이용해 실험을 수행하였다. 실험인자로 RP는 5로 지정하였으며, D = 10, AD = 5이다. 그리고 W = 0.1, DS = 0.1이며, 질의의 수 Q는 10,000~100,000 개의 질의를 1,000 단위로 측정하였다. 결과는 실험 1과 비슷한 양상을 보였다. 이는 본 실험의 결과가 NITF에 국한되지 않고 다양한 데이타에서 비슷한 결과를 보여줄 수 있음을 나타낸다.

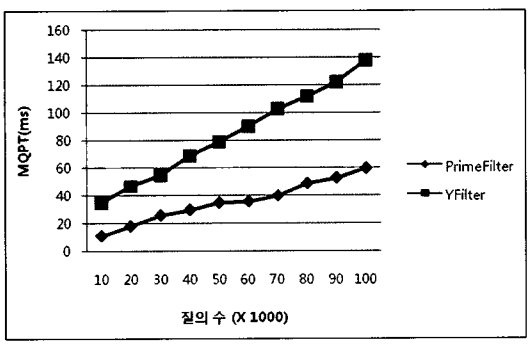


그림 11 질의 수에 따른 MQPT 비교(DBLP, D = 10, W = 0.1, DS = 0.1)

7. 결론 및 향후 연구 과제

본 논문이 제안한 PrimeFilter는 그 동안 많은 연구가 진행되었던 스트리밍 XML 데이타 필터링 기법들과 비교하여 다음과 같은 의미를 갖는다. 질의의 부분적인 공유나 프레디킷을 효과적으로 처리하려는 기존 연구들과 달리, 하위의 질의가 매칭되면 상위의 질의는 반드시 매칭된다는 질의 포함 관계의 성질을 이용한 새로운 XML 필터링 아이디어를 제시하였다. 또한 포함 관계 설정을 효율적으로 수행할 수 있는 소수 인덱싱 기법을 제안 소개하였다. 다만 PrimeFilter는 프레디킷을 제외한 단일 경로 질의를 처리하는데 중점을 두었기 때문에 프레디킷 처리 부분에 대한 연구는 미비하다. 앞으로의 향후 연구과제는 본연구에서 제안된 소수 인덱싱에 기반한 질의 포함관계기법을 프레디킷까지 적용하여 처리할 수 있는 방법을 모색하는 것이다.

참 고 문 헌

[1] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer, "Path sharing and predicate evaluation for high-performance XML filtering," ACM Transactions on Database Systems, Vol.28, No.4, pp. 467-516, 2003.

[2] K. S. Candan, W. Hsiung, S. Chen, J. Tatemura, D. Agrawal, "AFilter: Adaptable XML filtering with prefix-caching and suffix-clustering," Proc. 32th VLDB, Seoul, Korea, pp. 559-570, 2006.

[3] J. Kim and S. Park, "PosFilter: An efficient filtering technique of XML documents based on postfix sharing," Proc. 24th BNCOD, Glasgow, Scotland, pp. 70-81, 2007.

[4] P.Th. Eugster, P. Felber, R. Guerraoui, A. M. Kermarrec, "The Many Faces of Publish/Subscribe," ACM computing surveys, 2003.

[5] J. Clark, S. DeRose, "XML Path Language (XPath) Version 1.0," http://www.w3.org/TR/xpath, November, 1999.

[6] Y. Chen, S. Davidson, and Y. Zheng, "An efficient XPath query processor for XML streams," Proc. of the 22nd International Conference on Data Engineering (ICDE), pp. 77, 2006.

[7] J. Kwon, P. Rao, B. Moon, and S. Lee, "Fist: scalable XML document filtering by sequencing twig patterns," In Proc. of the 31th VLDB, pp. 217-228, 2005.

[8] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi, "Efficient Filtering of XML Documents with XPath Expressions," Proc. of the 18th International Conference on Data Engineering (ICDE), pp. 235-244, Feb. 2002.

[9] X. Wu, M. L. Lee, W. Hsu, "A Prime Number Labeling Scheme for Dynamic Ordered XML Trees,"

Proc. of the 20th International Conference on Data Engineering (ICDE), pp. 66-78, 2004.
[10] M. Franklin, Y. Diao, S. Rizvi, A. Edakkunni, M. Altinel, P. Fischer, R. To, and P. Hwang, "YFilter 1.0 release," 2003, available at http://yfilter.cs.umass.edu/code_release.htm

김 재 훈
정보과학회논문지 : 데이타베이스
제 35 권 제 2 호 참조

김 상 욱
2005년 8월 서강대학교 컴퓨터공학과 공학사. 2008년 2월 서강대학교 컴퓨터공학과 공학석사. 2008년 3월~현재 삼성전자 정보통신총괄 통신연구소 연구원 관심분야는 XML, 모바일 애플리케이션, 웹 데이타베이스, 데이타베이스 보안 임

박 석
정보과학회논문지 : 데이타베이스
제 35 권 제 2 호 참조