부분매칭 경로질의를 위한 포스트픽스 공유에 기반한 스트리밍 XML 데이타 필터링 기법

(A Filtering Technique of Streaming XML Data based Postfix Sharing for Partial matching Path Queries)

박 석 † 김 영 수 ††
(Seog Park) (Young-Soo Kim)

요 약 센서 네트워크나 유비쿼터스 환경이 보급되면서 최근에는 저장되어 있는 데이타가 아닌 계속적으로 빠르게 지나가는 스트리밍 데이타에 대한 연구가 활발하게 이루어지고 있다. 기존의 Publish-Subscribe 시스템도 인터넷의 발달로 데이타가 실시간으로 빠르게 들어오는 스트리밍 데이타의 형태를 가지게 되면서 스트리밍 데이타 연구에 관심을 가지게 되었고 이 중에서도 웹 환경의 표준으로 많이 사용되는 XML에 관심을 가지게 되었다. Publish-Subscribe 시스템에서 서버에 들어오는 스트리밍 XML 데이타에 대해서 질의에 빠르게 매치(match)되는 것을 찾기 위한 스트리밍 XML 데이타 필터링 기법이 오토마타를 이용해서 연구되었으며, 이중에서 비결정적 오토마타를 사용한 방법이 YFilter이다. 비결정적 오토마타를 사용하는 YFilter의 경우 질의 앞부분의 공통된 오퍼레이터를 한번에 계산하기 위해서 XPath 질의의 공통된 앞부분을 공유하고 질의의 루트부터 처리하는 하향식 방식을 사용하고 있다. 하지만, 부분매칭 경로질의의 경우에는 질의의 앞부분 공유를 방해하고 질의를 루트에서부터 처리할 필요가 없기 때문에 YFilter에서 부분매칭 경로질의가 증가하면 처리량이 떨어지는 문제가 발생한다. 본 논문에서는 이 문제 대해 XPath 질의의 공통된 뒷부분 공유에 기반한 상향식 방식을 사용하는 PoSFilter를 한가지 해결책으로 제시한다. 그리고 YFilter와 PoSFilter의 처리량을 비교를 통해서 PoSFilter의 경우 부분매칭 경로질의가 증가할 때 YFilter보다 좋은 처리량을 나타내는 것을 검증한다.

키워드 : 스트리밍 XML 데이타, 필터링, 질의 뒷부분 공유

Abstract As the environment with sensor network and ubiquitous computing is emerged, there are many demands of handling continuous, fast data such as streaming data. As work about streaming data has begun, work about management of streaming data in Publish-Subscribe system is started. The recent emergence of XML as a standard for information exchange on Internet has led to more interest in Publish - Subscribe system. A filtering technique of streaming XML data in the existing Publish- Subscribe system is using some schemes based on automata and YFilter, which is one of filtering techniques, is very popular. YFilter exploits commonality among path queries by sharing the common prefixes of the paths so that they are processed at most one and that is using the top-down approach.

However, because partial matching path queries interrupt the common prefix sharing and don't calculate from root, throughput of YFilter decreases. So we use sharing of commonality among path queries with the common postfixes of the paths and use the bottom-up approach instead of the top-down approach. This filtering technique is called as PoSFilter. And we verify this technique through comparing with YFilter about throughput.

Key words : Streaming XML data, Filtering, Postfix sharing

· 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10395-0) 지원으로 수행되었음
† 종신회원 : 서강대학교 컴퓨터학과 교수
spark@dblab.sogang.ac.kr
†† 정 회 원 : 삼성전자 정보통신총괄
cosmos0709@msn.com
논문접수 : 2005년 2월 14일
심사완료 : 2005년 11월 8일

1. 서 론

스트리밍 데이타는 실시간으로 빠르게 계속적으로 들어오는 데이타이다. 온라인 뉴스(Online News), 온라인 옥션(Online Auction)과 같은 도메인의 경우에는 사용자가 원하는 정보를 얻기 위해 질의를 등록하면 빠르게

정보를 제공하는 것을 목적으로 하는데 이러한 시스템을 Publish-Subscribe 시스템이라고 한다. 스트리밍 데이타 중에서도 XML의 경우 웹 환경에서 사용할 수 있을 뿐만 아니라 정보 교환에서 중요한 포맷으로 사용하기 때문에 Publish-Subscribe 시스템에서 많이 사용된다. Publish-Subscribe 시스템의 한 부분으로 스트리밍 XML 데이타 필터링 기법은 사용자가 보낸 질의에 대해서 빠르게 매치하는 데이타를 찾는 것을 목적으로 하기 때문에 어떤 질의가 들어오더라도 빠르게 처리할 수 있기를 원한다. 이때 기존의 필터링 기법으로 XFilter, YFilter, XMLTK를 간략히 소개하고 그 중에 메모리를 적게 사용하면서도 좋은 처리량을 나타내고 있는 YFilter를 검토한다.

XPath 질의는 크게 루트에서 시작하는 질의, 부분매칭 경로 질의, predicate를 가진 질의로 나눌 수 있다. 이 중에서 조상-자손 경로(//)로 시작하는 부분매칭 경로질의는 사용자의 편의에 의해 많이 사용될 수 있는 질의 종류이다. 스트리밍 XML 필터링 기법에서는 XPath의 질의 종류 중에서 어떤 질의가 들어올지 모르기 때문에 질의종류에 따라 처리량이 떨어지는 것은 문제가 된다. 하지만 YFilter의 경우에는 질의의 공통적인 앞부분 공유(prefix sharing)에 기반한 하향식(top-down) 비결정적 오토마타를 사용하기 때문에 부분매칭 경로질의가 증가하면 처리량이 떨어지는 문제점이 존재한다. 본 논문은 부분매칭 경로질의에서 처리량이 떨어지는 문제에 대한 새로운 필터링 기법으로 질의 뒷부분 공유(postfix sharing)에 기반한 상향식(bottom-up) 비결정적 오토마타를 사용하는 해결책을 제시한다. 제시한 해결책은 부분매칭 경로질의의 비율이 증가하면 기존의 YFilter보다 좋은 처리량을 나타낸다. 본 논문은 다음과 같이 구성한다. 우선 기존의 스트리밍 XML 데이타 필터링 기법을 살펴보고, 다음에는 연구동기와 제안하는 질의 뒷부분 공유에 기반한 상향식 비결정적 오토마타를 사용하는 PoSFilter에 대해서 살펴보며 마지막으로 제안하는 기법을 이용한 실험을 통해서 기존의 YFilter와 비교 평가하고 결론을 맺는다.

2. 관련연구

이 장에서는 스트리밍 XML 데이타 필터링 기법에 연관된 Publish-Subscribe 시스템에 대해서 살펴보고 기존의 스트리밍 XML 데이타 필터링 기법을 살펴본다.

2.1 Publish-Subscribe 시스템

Publish-Subscribe 시스템[1,2]은 데이타가 서버에 들어올 때 그 데이타들 중에서 원하는 데이타를 얻기 위해서 사용자가 질의를 등록하면 이 질의들은 서버에 저장되고 사용자들이 등록한 질의에 대해서 들어오는 데이타에 매치하는 질의가 있다면 해당 질의를 등록한 사용자에게 매치된 데이타를 보내주는 시스템이다. 본 연구에서는 이 시스템에서 발생하는 데이타가 스트리밍 XML 데이타인 경우인 XML-based Publish-Subscribe 시스템에 대해 연구하고 있으며, 이 경우에 스트리밍 XML 데이타의 형태로 들어온 XML 문서가 사용자가 등록한 XPath 질의에 매치되면 해당 XML 문서를 질의를 보낸 사용자에게 보내준다. Publish-Subscribe 시스템에서 스트리밍 XML 데이타 필터링 기법은 스트리밍 XML 데이타 형태로 들어온 XML 문서가 어떤 질의에 매치되는지 빠르게 알려주기 위한 기법이다. 그림 1은 Publish-Subscribe 시스템의 한 예를 보여주고 있다. 온라인 옥션(Online Auction)에서 실시간으로 새로 들어온 제품에 대한 데이타가 서버로 계속 들어가고 있을 때 사용자는 원하는 데이타를 얻기 위해 서버에 질의를 등록한다. 만약 $User_1$~$User_N$까지 원하는 온라인 옥션 정보를 얻기 위해 질의를 등록하면 서버에 사용자가 보낸 질의가 저장되며 저장된 뒤에 들어

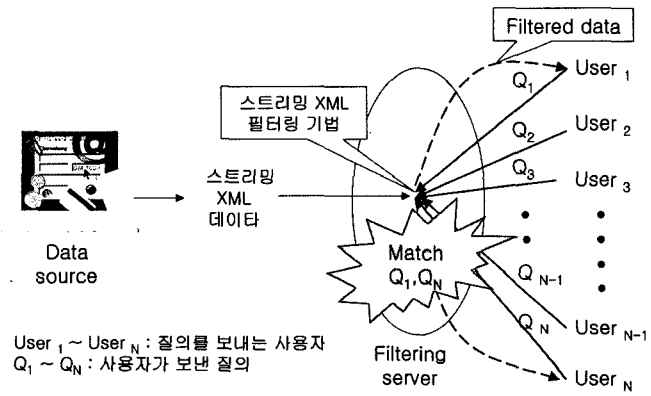


그림 1 Publish-Subscribe 시스템

온 데이타가 $User_1$과 $User_N$이 보낸 질의 Q_1과 Q_N에 매치(match)되면 해당 데이타를 $User_1$과 $User_N$에게 보내게 된다.

2.2 기존의 스트리밍 XML 필터링 방법

기존의 스트리밍 XML 필터링 기법 중에서 오토마타를 사용하는 필터링 기법으로 XFilter[3,4], YFilter[4-6], XMLTK[7,8]가 있다. XFilter는 스트리밍 XML 데이타 필터링 기법 중에서 가장 처음 나온 제안방법으로 사용자에 의해서 보내진 XPath 질의를 유한 오토마타(Finite State Machines)로 생각하여 처리하는 방식이다. 각각의 XPath 질의를 유한 오토마타로 생각하고 오토마타의 마지막 상태에 도달하게 되면 질의가 매치(match) 되었다고 한다. 하지만 XFilter의 경우에는 같은 엘리먼트에 의해 처리될 수 있는데도 불구하고 XPath 각각의 질의에 대해서 계산을 각각 따로 해주어야 하기 때문에 질의의 수가 많아지면 질의 처리를 위한 계산이 증가하기 때문에 처리량이 많이 감소한다. 예를 들면, /a/b와 /a/c라는 질의가 있을 때 /a 부분은 모두 XML 엘리먼트 a에 의해 처리될 수 있는데 /a/b와 /a/c의 /a를 각각 따로 계산해 주어야 한다. 질의 수가 증가하면 처리량이 많이 감소하는 문제를 해결하기 위해서 다음의 YFilter 방법이 제안되었다.

Yfilter는 각각의 XPath 질의를 하나의 비결정적 오토마타(Nondeterministic Finite Automata)로 보고 각 오토마타에 대해서 질의의 공통되는 앞부분을 공유하는 방법을 사용하여 질의들을 하나의 비결정적 오토마타로 만들며 XML의 루트에서부터 처리하도록 하향식(top-down) 방식을 사용하고 있다. 그리고 오토마타의 상태 이동을 위해서 스택을 사용한다. 질의의 앞부분을 공유하고 있기 때문에 같은 엘리먼트에 대해서 질의에 따라 각각 계산을 해주는 것이 아니라 한번만 계산을 해주면 된다. 위의 질의 /a/b와 /a/c에 대해서 질의 앞부분에 공통되는 /a를 한번만 계산하면 된다. XFilter보다 질의의 수가 많을 때 처리량이 좋아지는 장점이 있으며 질의의 변화에 대해서 빠르게 오토마타를 바꿀 수 있고 메모리의 사용량도 적다. 하지만 YFilter의 경우에는 질의의 앞부분을 보고 공유를 하기 때문에 부분매칭 경로질의와 같이 질의의 앞부분이 조상-경로 질의로 시작하는 질의가 증가하면 질의의 앞부분 공유가 방해되기 때문에 처리량이 감소하는 문제가 발생한다.

XMLTK는 비결정적 오토마타로 구성된 XPath 질의들을 결정적 오토마타(Deterministic Finite Automata)로 구성하여 사용하는 방법이다. 비결정적 오토마타의 경우에는 비결정적인 경로(ε, *)가 존재하기 때문에 오토마타 상태 이동 시에 여러 개의 상태를 살펴보아야 하지만 결정적 오토마타를 사용하는 경우에는 경로가 하나로 정해지기 때문에 자신이 가야 할 경로를 쉽게 결정할 수 있는 장점이 있어 처리량이 좋다. 하지만, 질의의 변화가 생기면 다시 비결정적 오토마타에서 결정적 오토마타로 바꾸어야 하기 때문에 질의 변화 시에 처리량이 감소하는 문제가 발생하며 특히 조상-자손 경로와 같이 비결정적 경로가 증가할 경우에는 메모리의 사용이 급격히 증가하는 단점이 있다. YFilter와 XMLTK는 서로 반대의 장, 단점을 갖고 있기 때문에 각각의 응용에 따른 요구에 맞추어 필요한 필터링 기법을 사용할 수 있다.

3. PoSFilter : Postfix sharing에 기반한 필터링 기법

기존의 필터링 기법 중에서 YFilter와 같은 비결정적 오토마타를 사용할 경우 부분매칭 경로질의가 증가할 때 처리량이 감소하는 문제가 생긴다. 이에 대한 제안으로 부분매칭 경로질의에 적합한 필터링 기법으로 질의 뒷부분 공유(postfix sharing)에 기반한 상향식(bottom-up) 비결정적 오토마타를 사용하는 새로운 방법, 즉 PoSFilter를 제안한다.

3.1 연구목표

YFilter는 많은 질의를 처리하기 위해서 질의 앞부분을 공유하는 방법을 사용하였으며 이 방법은 질의의 수가 많아도 처리량을 많이 떨어뜨리지 않는 장점이 있었다. 하지만 YFilter의 경우 질의 종류를 고려하지 않는 문제가 있었다. 스트리밍 XML 데이타 필터링 기법에서 사용자가 질의를 등록할 때 어떤 종류의 질의를 등록할지 알 수 없기 때문에 질의 종류의 비율이 달라질 때 처리량이 떨어진다면 문제가 된다. 특히 YFilter의 경우에는 부분매칭 경로질의에 대해서 처리량이 떨어지는 문제가 발생한다.

앞으로 계속적인 설명을 위한 예제로 그림 2의 XML 문서를 사용한다. 이 문서는 온라인 옥션을 나타내는 XML 문서이다[9].

우선 부분매칭 경로질의는 XPath 질의가 XML의 루트에서 시작하는 질의가 아닌 조상-자손 경로(//)로 시작하는 질의를 말한다[10]. 만약 3명의 사용자가 온라인 옥션 XML 문서에 대해서 각각 "문서에 물건을 파는 사람의 이름이 나와있는가?"라는 질의 Q1과 "문서에 모든 사람들의 이름이 나와있는가?" 라는 질의 Q2와, "문서에 모든 이름이 나와있는가?" 라는 Q3라는 질의를 던졌다고 하자. 이때 루트에서 시작하는 질의는 그림 3(a)와 같이 나타낼 수 있다.

하지만, 그림 3(a)의 질의 Q1은 seller/person/name의 부분만으로 같은 질의 결과를 가질 수 있다. 질의 Q2는 person/name, 질의 Q3은 name으로 같은 질의

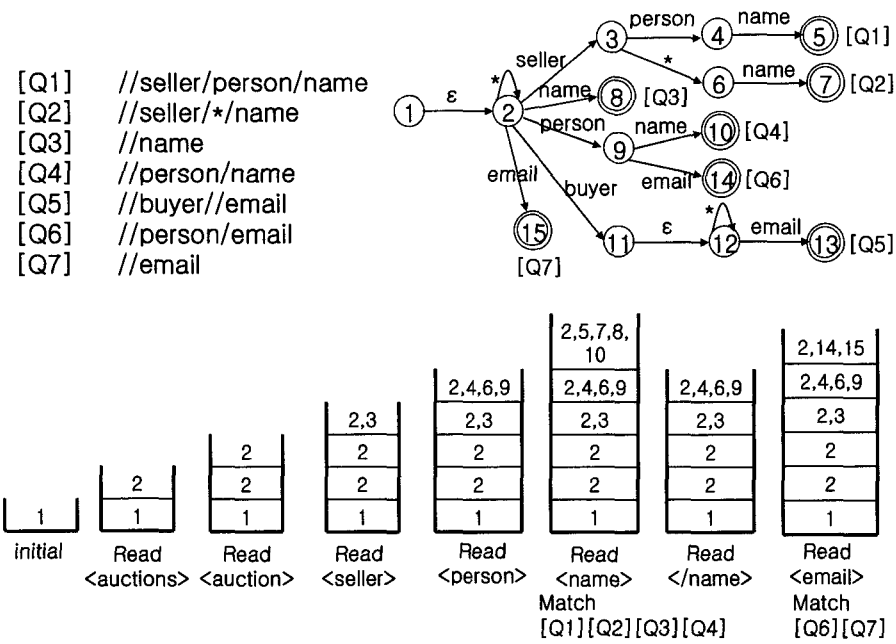


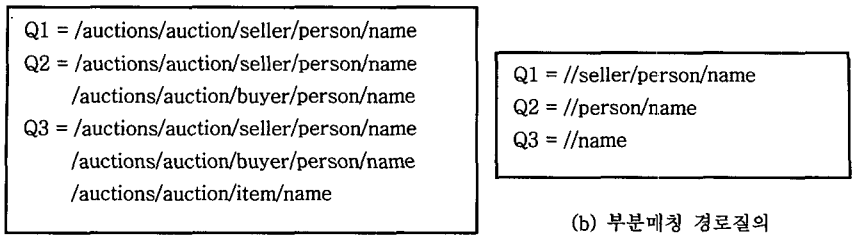
그림 2 YFilter

결과를 가질 수 있다. 이렇게 실제로 필요한 부분들만으로도 충분히 원하는 질의를 구성할 수 있으며 이 경우에 실제로 필요한 부분들은 XML 문서에서 루트의 자손이 되기 때문에 필요한 부분의 앞부분에 조상-자손 기호(//)를 붙여서 그림 3(b)의 질의를 만들 수 있으며 이를 부분매칭 경로질의라고 한다. 부분매칭 경로질의의 경우 XPath 질의에서 루트에서부터 질의를 모두 다 사용할 필요없이 질의 사용자가 원하는 부분만을 써주면 되기 때문에 사용자의 편리성에 의해서 자주 사용될 수 있는 질의이다.

YFilter는 질의 앞부분 공유하는데 부분매칭 경로질의가 들어가게 되면 기존의 경로에서 질의 앞부분 공유가 방해되며 스트리밍 XML 필터링 기법에서 질의는 XML 문서의 내용을 중시하기 때문에 PCDATA군에 접근하는 질의의 뒷부분에 공유하는 질의 노드가 많고 부분매칭 경로질의의 경우 PCDATA에 접근하는 질의 뒷부분을 중시하기 때문에 질의 뒷부분을 공유하는 것이 중요하다. 또한 YFilter처럼 하향식 방법을 사용할 경우에 무조건 XML의 루트에서 시작해야 하기 때문에 XML의 루트부터 시작하지 않는 부분매칭 경로질의의 경우에 처리하지 않아도 되는 엘리먼트를 처리해야 한다는 문제점이 존재한다. 예를 들어 그림 4에서 하향식

```
<auctions>
  <auction>
    <seller>
      <person>
        <name>James</name>
        <email>James@dot.com</email>
      </person>
    </seller>
    <buyer>
      <person>
        <name>Jane</name>
        <email>Jane@dot.com</email>
        <phone>02-333-4444</phone>
      </person>
    </buyer>
    <item>
      <name>toy</name>
      <link>http://www.toy.com</link>
    </item>
    <price>10000</price>
    <payment>
      <date>2004/10/22</date>
      <money>10000</money>
    </payment>
  </auction>
</auctions>
```

그림 3 예제 XML 문서



(a) 루트에서 시작하는 질의 (b) 부분매칭 경로질의

그림 4 루트에서 시작하는 질의와 부분매칭 경로질의

방식을 사용하는 YFilter는 루트에서 오는 엘리먼트인 <auctions> <auction>까지 상태 2에서 연산을 해주어야 하며 또한 질의 Q1, Q2, Q3의 앞부분이 서로 다르기 때문에 질의의 앞부분 공유가 되지 않아 상태 2에서 seller, name, person으로 오토마타의 상태가 나누어지게 되며 뒷부분의 person, name은 공유되지 않는다. 그러므로 스택에서 운영되는 오토마타의 상태 수가 증가하기 때문에 YFilter에서 부분매칭 경로질의가 발생하면 YFilter의 처리량이 떨어지는 결과를 가져오게 된다. 반면에 질의 뒷부분 공유에 기반한 상향식 비결정적 오토마타를 사용할 때에는 질의의 뒷부분을 공유를 하기 때문에 질의 Q1, Q2, Q3의 person, name이 공유되며 <auctions> <auction> 부분을 처리할 필요가 없기 때문에 스택에서 운영되는 오토마타의 상태가 적어진다. 이 방법을 PoSFilter라고 한다.

3.2 가정 및 환경

본 연구에서는 스트리밍 XML 데이타 필터링 기법을 위한 질의를 다음과 같이 가정한다.

- 스트리밍 XML 데이타 필터링 기법에서 가정하는 질의는 지금 들어오고 있는 스트리밍 XML 문서에 대한 질의만을 가정한다. 즉, 지나간 문서는 저장되지 않기 때문에 질의 처리를 하지 않는다.
- XPath의 위치경로 탐색을 우선적으로 고려하기 위해서 XPath 질의에서 predicate를 제외한 질의로 가정한다.
- 스트리밍 XML 데이타 필터링 기법은 스트리밍 XML 데이타의 내용을 중요시 하기 때문에 PCDATA군에 접근하는 질의들이다.

첫 번째 가정에서 Publish-Subscribe 시스템에서는 이미 지나간 데이타는 저장해두지 않기 때문에 더 이상 등록된 질의에 대해서 데이타가 매치되는지 살펴보지 않는다. 그리고 두 번째 가정에 있어서 제안하는 필터링 기법의 경우에는 우선 XPath의 위치 경로의 탐색을 줄이는 방법을 연구하고 있기 때문에 predicate의 처리에 대한 연구는 추후연구로 남겨놓는다. 세 번째 가정에서 Publish-Subscribe 시스템에서 사용자는 XML의 내용(content)을 중요시하는데 내용 중에서도 사용자가 원하는 정보는 대부분 XML의 PCDATA에 있다. 그러므로 사용자가 스트리밍 XML 필터링 기법을 사용할 때 등록하는 XPath 질의들은 PCDATA군에 접근하는 질의를 등록하게 된다.

3.3 제안한 PoSFilter 구성방법

3.3.1 XPath 질의의 상향식 비결정적 오토마타 구성단위

우선 PoSFilter를 구성하기 전에 사용자가 등록한 XPath 질의를 각각 하나의 비결정적 오토마타로 생각한다. PoSFilter의 경우 질의의 뒷부분에서 시작하는 상향식 방법을 사용하기 때문에 그림 5처럼 XPath 각각의 질의를 상향식 비결정적 오토마타로 나타낸다. 여기에서 * (wildcard)는 어떤 노드가 들어와도 상관없다는 노드로 XML에서 어떤 엘리먼트가 들어와도 경로를 지나갈 수 있다는 것을 나타내며 ε는 오토마타의 빈 문자열로 조상-자손 경로를 나타낼 때 사용한다. 단, 조상-자손 경로가 질의의 맨 앞에 있는 경우는 생략할 수 있다.

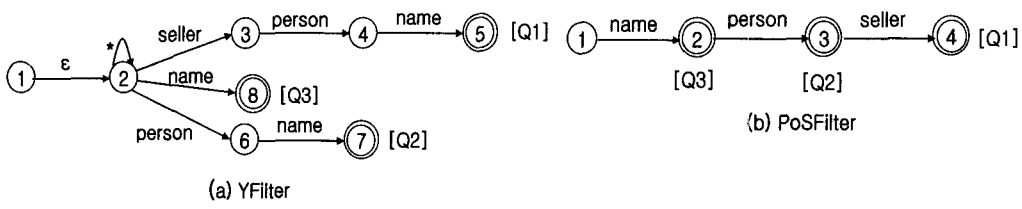


그림 5 하향식 방식과 상향식 방식 비교

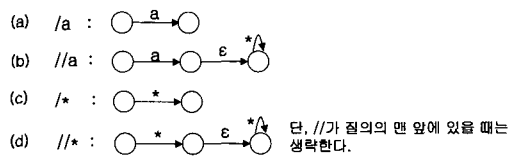


그림 6 XPath 질의의 상향식 비결정적 오토마타 구성 단위

3.3.2 PoSFilter 구성방법

사용자가 보낸 XPath 질의들 각각에 대해서 상향식 비결정적 오토마타를 구성했다면 이들 오토마타들을 공유를 통해 하나의 오토마타로 구성할 수 있다. 이 질의 각각에 대한 오토마타를 NFA_p로 나타내고 들어온 스트리밍 XML 데이타의 엘리먼트에 대해서 오토마타의 상태를 이동했을 때 오토마타 NFA_p의 마지막 상태(Final state)에 도착하면 이때 질의가 스트리밍 XML 데이타로 된 XML 문서에 매치되었다고 한다. 이 질의 오토마타 NFA_p를 하나의 커다란 질의 뒷부분 공유에 기반한 상향식 비결정적 오토마타로 구성하고 이를 PoSFilter라 한다. PoSFilter는 질의의 뒷부분부터 시작하며 무조건 오토마타의 상태 1에서 시작해야 한다.

• PoSFilter 구성방법

가) 우선 맨 처음 질의가 들어온 경우에는 공유할 오토마타 상태가 없기 때문에 새로운 오토마타 상태를 생성하며 새로운 오토마타 상태를 생성할 경우 이 상태들을 나타내기 위해 생성하는 상태의 순서에 따라 번호를 매긴다.

나) 다음에 질의가 들어오면 우선 기존에 들어온 XPath 질의 오토마타의 마지막 상태들을 저장한 리스트를 보고 이 안에서 들어온 질의가 기존의 질의들과 질의의 앞부분이 공유될 수 있는지 살펴본다. 질의의 앞부분이 공유될 수 있다면 공유하여 최대 공유될 수 있는 오토마타 상태를 저장해둔다.

다) 다음에는 질의의 뒷부분을 보는데 질의의 뒷부분은 항상 오토마타의 상태 1에서 시작하며 이때 기존의 질의들과 질의의 뒷부분이 공유될 수 있는지 살펴본다. 이 때 공유된다면 기존의 질의와 공유하고 아닌 경우에는 새로운 오토마타 상태를 생성한다.

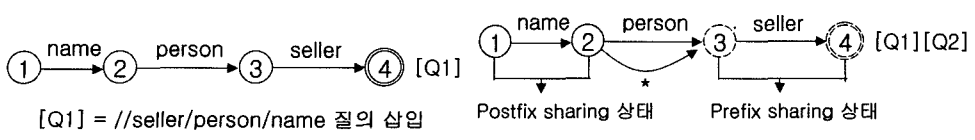
라) 질의의 뒷부분에서 질의를 공유하거나 새로운 상태를 만들다가 만약 질의의 앞부분에 공유되는 오토마타 상태가 있었다면 이 오토마타 상태와 연결한다.

• PoSFilter 구성 알고리즘

위의 구성방법에 대해서 예를 들어 다음의 4개의 질의가 있을 때 이 질의들에 대해서 PoSFilter를 구성하는 방법을 살펴보자.

PoSFilter를 구성하기 위해서 가장 첫 번째 질의를 고려한다. 그림 7에서 첫 번째 질의인 //seller/person/name의 경우에는 처음 질의가 들어왔기 때문에 다른 질의와 공유할 필요없이 새로운 오토마타의 상태를 생성한다(알고리즘 41~42). 오토마타의 상태 번호는 오토마타가 들어와서 새로운 상태가 구성되는 순서를 입력하면 된다. //seller/person/name의 경우에는 맨 처음에 무조건 질의의 가장 뒷부분을 상태 1에서 시작하고 다른 상향식 질의 오토마타와 공유되는 부분이 없기 때문에 새로운 오토마타 상태를 만들면서 순서대로 상태 번호 2, 3, 4를 붙인다. 그림 7에서 두 번째 질의 //seller/*/name의 경우에는 우선 첫 번째 질의의 마지막 상태인 4로부터 뒤로 읽으면서 질의의 앞부분 부분에 공유가 될 수 있는 부분이 있는지 살펴본다(알고리즘 3~9). 첫 번째 질의가 //seller로부터 시작되었기 때문에 두 개의 질의 중에서 앞부분 seller는 공유될 수 있다(알고리즘 21~26). 앞부분에 공유되는 상태는 3과 4가 되고 이중에서 마지막 공유 부분인 3을 저장해둔다. 그리고 질의의 앞부분에 더 이상 공유되는 질의 노드가 없으면 그 다음에는 질의의 가장 뒷부분을 공유하는데(알고리즘 27~31) 상태 1에서 시작하며 두 개의 질의가 name으로 질의 뒷부분이 같기 때문에 이 부분을 공유한다. 그러면 name과 seller사이에 *가 남게 되는데 뒷부분은 상태 2까지 공유되고 앞부분은 상태 3에서 공유되기 때문에 전이 *(wildcard)는 상태 2와 상태 3을 연결한다(알고리즘 32~34).

세 번째 질의는 그림 8에서 보듯이 //name의 경우에는 질의가 하나의 노드로 이루어져 있기 때문에 곧바로 뒷부분 공유를 하는데 질의의 뒷부분이 기존의 오토마타와 상태 1과 상태 2에서 공유되기 때문에 그대로 공유시킨다. 네 번째 질의 //person/name의 경우에는 그



[Q1] = //seller/person/name 질의 삽입

[Q2] = //seller/*/name 질의 삽입

그림 7 질의 Q1과 Q2 삽입 방법

• PoSFilter 구성 알고리즘

```
BU_NFAp : XPath 질의의 상향식(bottom-up) 비결정적 오토마타
FinalList : 마지막 상태들이 들어있는 리스트
FFinalState : 들어온 상향식 오토마타 질의와 질의의 앞부분이 공유될 수 있는 오토마타의 마지막 상태
FirstState : 모든 상향식 오토마타가 처음 시작하는 초기 상태
PreState : 질의의 앞부분 공유가 어떤 상태까지 공유되는지 알기 위해 저장한 상태
PostState : 질의의 뒷부분 공유가 어떤 상태까지 공유되는지 알기 위해 저장한 상태
TempState : 임시적인 상태 저장을 위한 상태
Struct state {
        Beforestate_id; 현재 상태의 이전 상태
        Nextstate_id; 현재 상태의 다음 상태
        element; 현재 상태의 엘리먼트들
}

1 Composeautomata(BU_NFAp){
2       if(BU_NFAp_is_not_First){
3               for(FinalState ∈ FinalList){
4                       if(FilnalState.Beforestate_id.element==
5                               FinalState_of_BU_NFAp.Beforestate_id.element){
6                               FFinalState=FinalState;
7                               Break;
8                       }
9               }
10              if(FFinalState==NULL){
11                      FirstState=FirstState_of_BU_NFAp;
12                      TempState = FirstState;
13                      for(TempState.element == element_of_BU_NFAp){
14                              Postfix_Sharing;
15                              TempState = TempState.NextState_id;
16                      }
17                      Make New_State;
18                      FinalList.insert(FinalState);
19              }
20              else{
21                      TempState = FFinalState;
22                      for(TempState.Beforestate_id.element==
23                                      BU_NFAp.Beforestate_id.element){
24                              Prefix_Sharing;
25                              PreState=TempState.BeforeState_id;
26                      }
27                      TempState = FirstState;
28                      for(TempState.NextState_id.element== element_of_BU_NFAp){
29                              Postfix_Sharing;
30                              PostState=TempState.NextState_id;
31                      }
32                      if(BU_NFApof_element_is_not_Sharing){
33                              Make New_State;
34                      }
35                      else{
36                              PreState_and_PostState_Sharing;
37                      }
38              }
39      }
40      else{
41              Make New_State;
42              FinalList.insert(FinalState);
43      }
}
```

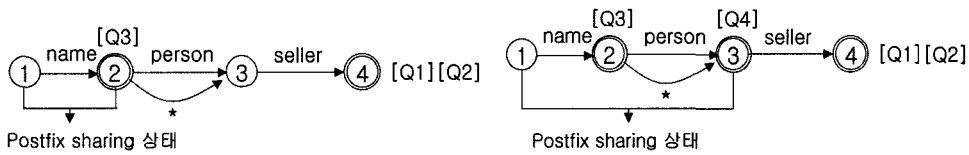


그림 8 질의 Q3과 Q4의 삽입 방법

림 8에서 질의의 마지막 상태가 2와 4의 두 개이기 때문에 여기에서부터 person이 공유될 수 있는지 찾는다. 하지만 상태 2는 name이 있고 상태 4는 seller가 있기 때문에 공유할 부분이 없다. 그러므로 질의의 뒷부분인 상태 1에서 시작해서 공유될 부분을 찾는데 이때 name과 person이 공유될 수 있으므로 질의의 뒷부분에서 name과 person이 공유된다. YFilter와 PoSFilter에서 비결정적 오토마타로 구성된 여러 개의 XPath를 하나의 비결정적 오토마타로 구성하는데 드는 비용을 비교해본다. 여기에서 총 XPath 질의의 수를 N이라고 하고 각 XPath를 구성하는 노드의 개수를 n이라고 하며 XPath 질의들에 대해서 마지막 상태를 저장한 리스트 안의 마지막 상태 개수를 k라고 하자. 이 경우 YFilter의 경우 하나의 비결정적 오토마타를 구성하는 경우에 O(N×n)이 사용되지만 PoSFilter의 경우 XPath 질의에서 마지막 상태들에서 앞부분이 매치되는 경우가 있는지 살펴보아야 하기 때문에 O(N× (k+n))의 비용이 든다. PoSFilter의 경우 YFilter의 경우보다 비결정적 오토마타 구성 시에 좀 더 많은 비용이 들지만 이미 구성된 후의 처리량의 경우에는 PoSFilter가 좀 더 좋은 성능을 나타낼 수 있다.

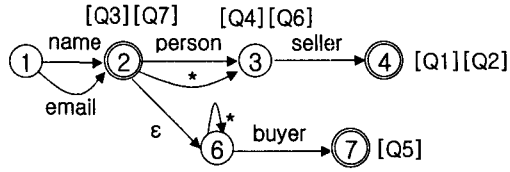
3.3.3 PoSFilter 실행

위에서 여러 개의 XPath 질의를 질의 뒷부분 공유에 기반한 상향식 비결정적 오토마타인 PoSFilter로 구성하였다. PoSFilter를 실행하기 위해서 스트리밍 XML 데이타의 각 엘리먼트가 들어올 때마다 오토마타의 상태를 어떻게 이동해야 하는지 알아야 한다. 이렇게 엘리먼트에 대한 오토마타의 상태의 이동을 알기 위해서 스택을 사용한다. 스트리밍 XML 데이타는 스트리밍 XML 데이타가 시작되는 부분(Start of Document), 스트리밍 XML 데이타의 시작 엘리먼트 부분(Start of Element), 스트리밍 XML 데이타의 끝 엘리먼트 부분(End of Element)으로 나눌 수 있다.

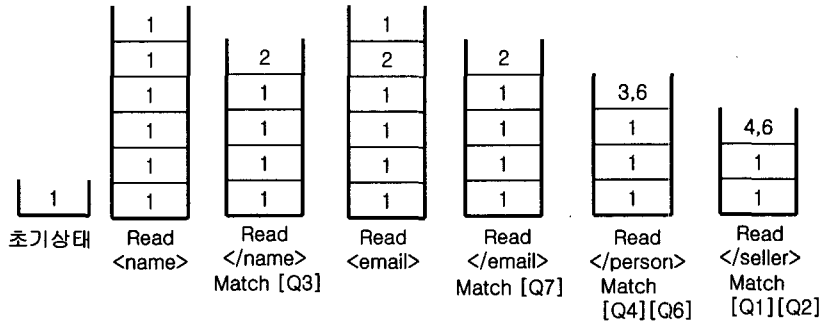
- 스트리밍 XML 데이타의 시작부분(Start of Document)
  PoSFilter의 오토마타 상태 이동을 알기 위한 스택을 준비한다. 그리고 그 스택에는 PoSFilter의 상태 이동에서 가장 먼저 시작하는 상태 1을 저장해둔다.
- 스트리밍 XML 데이타의 시작 엘리먼트 부분 (Start of Element)
  PoSFilter의 상태 1을 스택에 푸쉬(push)해준다.
- 스트리밍 XML 데이타의 끝 엘리먼트 부분(End of Element)
  PoSFilter는 상태 이동을 위해 끝 엘리먼트를 이용한다. 스택의 탑(top)에 있는 오토마타의 상태를 읽어서 다음에 이동할 상태를 구하고 팝(pop)을 한 뒤에 다음에 이동할 상태를 스택의 탑에 넣는다. 이때 새로운 오토마타 이동 상태를 넣을 때 이전에 다른 상태도 존재한다면 그 상태도 보존하면서 다음에 이동할 상태를 저장한다. 단, 이때 상태 1의 경우는 그 상태를 보존할 필요 없다. 만약 다음에 이동할 상태를 구했을 때 이 오토마타 상태가 질의의 마지막 상태인 경우에는 매치하는 질의를 구한 뒤에 이 마지막 상태에 더 이상 오토마타의 전이가 존재하지 않는다면 이 상태는 지워버린다. 그리고 만약 오토마타의 상태가 셀프 루프(self loop)를 가지고 있는 경우에는 상태 이동 시에 계속적으로 함께 이동한다.

그림 9(a)는 사용자가 보낸 질의들이고 이 7개의 질의들을 가지고 그림 9(b)의 PoSFilter를 만들었을 때 이 오토마타의 상태가 스트리밍 XML 데이타에 의해서 어떻게 바뀌는지 스택의 상태를 그림 9(c)가 보여주고 있다. 우선 스트리밍 XML 데이타가 시작되면 스택을 그림 9(c)의 초기 상태로 만들며 이때 항상 PoSFilter는 오토마타의 상태 1에서 시작하기 때문에 초기 상태에는 항상 상태 1을 저장해둔다. 스트리밍 XML 데이타가 시작되면 스트리밍 XML 데이타의 엘리먼트가 들어오기 시작한다. 그림 2에서 시작 엘리먼트 <auctions> <auction> <seller> <person> <name>이 차례대로 들어오면 오토마타 상태 1만 계속 스택에 푸쉬한다. 그러면 <name> 엘리먼트가 들어올 때까지 그림 9(c)의 두 번째 상태가 된다. 그 다음에 처음으로 XML 문서의 끝 엘리먼트 </name>가 들어오게 되고 우선 스택의 탑에 있는 오토마타의 상태를 찾아본다. 이 때 스택의 탑에는 오토마타의 상태 1이 들어있으며 오토마타의 상태 1에서 시작하여 name 엘리먼트로 오토마타에서 이동할 수 있는 상태를 PoSFilter에서 있는지 살펴본다. 그림 9(b)의 오토마타에서 상태 1에서 name이 들어오면 상태 2

Q1 = //seller/person/name
Q2 = //seller/*/name
Q3 = //name
Q4 = //person/name
Q5 = //buyer//email
Q6 = //person/email
Q7 = //email
(a) XPath queries



(b) 질의 뒷부분 공유에 기반한 상향식 비결정적 오토마타 (PoSFilter)



(c) 실행스택 (stack)

그림 9 PoSFilter 실행 예제

로 이동하는 것을 알 수 있다. 그러면 기존의 스택의 탑에 있던 오토마타의 상태 1을 팝(pop)하고 스택의 탑에 새로운 오토마타의 상태 2를 넣는다. 이 때 상태 2는 질의의 마지막 상태로 이때는 오토마타에서 매치되는 질의가 무엇인지 찾는다. 이 때 name 엘리먼트가 들어왔기 때문에 질의 3에 매치된다. 상태 2는 마지막 상태이긴 하지만 그 다음에 연결된 오토마타 상태들이 더 있기 때문에 그대로 스택의 탑에 남겨둔다. 그 다음에 <email>가 들어오면 스택에 다시 상태 1을 푸쉬하고 </email>이 들어오면 스택의 탑의 상태 1에서 email 엘리먼트로 어디로 이동할 수 있는지 살펴본다. 이 때 또 다시 상태 2로 이동하게 된다. 이렇게 스택을 이용하여 스트리밍 XML 엘리먼트가 들어올 때마다 오토마타의 어느 상태로 이동할 수 있는지 알게 됨으로 해서 매치되는 질의가 무엇인지 결과를 얻을 수 있게 된다.

스택에 운영되는 오토마타의 상태 수가 많다는 것은 그만큼 XPath 질의에 대해서 계산해야 하는 오퍼레이터의 수가 많아진다는 것을 의미하고 스택에 운영되는 오토마타의 상태 수가 적어진다는 것은 XPath 질의에 대해서 계산해야 하는 오퍼레이터의 수가 적어진다는 것을 의미한다. YFilter의 경우에는 XPath 질의들의 앞부분의 공통적인 부분이 있는지 보고 공유하기 때문에 질의의 앞부분이 달라지면 새로운 오토마타 상태를 만들기 때문에 질의 뒤에 공통적인 부분이 있더라도 공유를 하지 못한다. 그러므로 YFilter에서 부분매칭 경로질의가 증가하는 경우 스택에 운영되는 오토마타의 상태 수가 증가하게 된다. 반면에 PoSFilter의 경우에는 질의가 있을 때 우선 질의의 앞부분에 공유할 수 있는 부분이 있는지 살펴 공유할 수 있는 상태를 찾아내고 다음에 질의의 뒷부분에도 공유할 수 있는 부분이 있는지 살핀다. 이로 인해 PoSFilter의 경우 부분매칭 경로질의가 증가할 때 스택에 운영되는 오토마타의 상태 수가 YFilter의 경우보다 적어진다. 그림 2의 YFilter와 그림 9의 PoSFilter를 비교해보면 둘다 시작 엘리먼트에서는 상태를 push하고 끝 엘리먼트에서는 상태를 pop하기 때문에 스택의 크기는 동일하지만 스택에서 운영하는 오토마타의 상태 수는 YFilter가 16이고 PoSFilter는 6인 것을 볼 수 있다. 이렇게 PoSFilter의 경우에는 기존의 질의 앞부분 공유 뿐만 아니라 질의 뒷부분 공유까지 하기 때문에 부분매칭 경로질의가 증가하여 질의 앞부분 공유를 방해하더라도 YFilter의 경우처럼 오토마타의 상태수가 증가하지 않기 때문에 처리량이 YFilter의 경우보다 높아질 수 있다.

4. 실험 및 평가

YFilter는 질의의 수에 따른 처리량만을 중요시 하였다. 하지만 스트리밍 XML 데이타 필터링 기법의 경우에는 어떤 종류의 질의가 들어올지 모르기 때문에 질의

의 종류에 대해서도 고려해야 한다. 본 논문에서 제시한 PoSFilter는 부분매칭 경로질의에 대해서 효과적으로 스트리밍 XML 데이타를 처리하기 위한 방법이다. 이번 장에서는 YFilter와 제시한 PoSFilter의 처리량에 대해서 비교해 본다.

• 실험 환경

실험을 위한 스트리밍 XML 데이타로 XMark[8]에서 제공하는 온라인 옥션을 위한 DTD를 사용하였다. 여기에서 가정하는 온라인 옥션은 옥션의 사용자가 자신이 원하는 물건의 정보를 얻기 위해서 XPath 질의를 서버에 등록하고 이 옥션에서는 물건을 팔거나 산다는 정보를 스트리밍 XML 데이타 형태로 실시간으로 받아들인다. 여기에서는 옥션의 사용자가 등록한 XPath 질의를 크게 루트에서 시작하는 질의와 부분매칭 경로질의의 두 가지로 나눈다. 처리량은 초당 처리할 수 있는 엘리먼트를 나타내며 처리량이 높은 필터링 기법이 좋은 성능을 나타낸다. 실험을 위해서 CPU Pentium4 1.7GHz, 512M인 컴퓨터에서 Java를 사용하여 구현하였다.

• 부분매칭 경로질의 수에 따른 처리량 비교

이 실험에서는 사용자가 등록한 모든 질의가 부분매칭 경로질의이고 질의의 수가 증가할 때 YFilter와 PoSFilter의 처리량의 변화를 알아보려고 한다. 그림 9는 질의가 모두 부분매칭 경로질의이며 질의의 개수가 100, 500, 1000일 때의 결과를 나타내고 있다. 질의가 모두 부분매칭 경로질의의 경우에 YFilter의 경우보다는 PoSFilter의 처리량이 높은 것을 알 수 있다. YFilter의 경우에는 질의의 앞부분을 공유하기 때문에 질의의 뒷부분 공유를 중시하는 부분매칭 경로질의만 있을 경우에 질의 앞부분 공유가 방해를 받기 때문에 YFilter의 처리량은 감소할 수 밖에 없다. 하지만 PoSFilter의 경우에는 질의 앞부분 공유뿐만 아니라 YFilter에서는 공유해 줄 수 없었던 질의 뒷부분까지 공유해 주기 때문에 스택에 운영되는 오토마타의 상태 수가 적어지고 그러므로 처리량이 YFilter보다 좋게 된다.

• 부분매칭 경로질의 비율에 따른 처리량 비교

이 실험에서는 정해진 질의 수에 대해서 부분매칭 경로질의 비율이 변할 때 그에 따른 두 개의 필터링 기법의 처리량을 비교하기 위한 실험이다. 질의의 수는 100이고 부분매칭 경로질의의 비율이 0%, 20%, 40%, 60%, 80%, 100%로 변경된다. 그림 10을 보면 YFilter의 경우에 모두 루트로 시작하는 질의일 경우에 가장 좋은 처리량을 보이고 있다. 이것은 질의가 루트에서부터 시작할 경우에 모든 질의의 앞부분이 공유될 수 있어서 질의 앞부분 공유가 증가하기 때문이다. 하지만 YFilter의 경우 부분매칭 경로질의가 20%만 되어도 처리량이 떨어지는 결과를 보인다.

이 이유는 루트에서 시작하는 질의의 앞부분과 부분매칭 경로질의의 앞부분이 서로 다르기 때문이다. 루트에서 시작하는 질의는 부모-자식 경로(/)로 시작하지만 부분매칭 경로질의의 경우에는 조상-자손 질의(//)로 시작하기 때문에 서로 앞부분이 공유될 수 없다. 그래서 이전에 0%일 때는 앞부분이 공유되었던 질의들이 부분매칭 경로질의로 바뀌면서 이전의 질의들과 공유되지 못하고 따로 떨어져나가게 된다. 이렇게 해서 YFilter는 20%일 때부터 급격히 처리량이 감소하고 그 이후에도 부분매칭 경로질의가 증가하면 서서히 처리량이 감소한다. 반면에 PoSFilter의 경우에는 상향식 방식으로 처리하기 때문에 루트가 모두 있는 경로 질의에 대해서는 처리량이 YFilter보다 떨어졌지만 질의 앞부분 공유와 질의 뒷부분 공유를 모두 지원하기 때문에 부분매칭 경로질의가 증가하더라도 이전에 있는 질의들과 공유되기 때문에 처리량이 별로 변하지 않는 특징을 보인다. 그러므로 부분매칭 경로질의가 20%만 넘더라도 PoSFilter가 더 좋은 처리량을 보이는 것을 알 수 있다.

• 부분매칭 경로질의 비율에 따른 스택의 오토마타 상태 수 비교

위에서 부분매칭 경로질의의 비율에 따라서 처리량이 달라지는 결과를 볼 수 있었다. 이것은 YFilter와 PoSFilter에서 스택 안에서 운영하는 오토마타의 상태 수에 영향을 받기 때문이다. 스택 안에 오토마타의 상태 수가

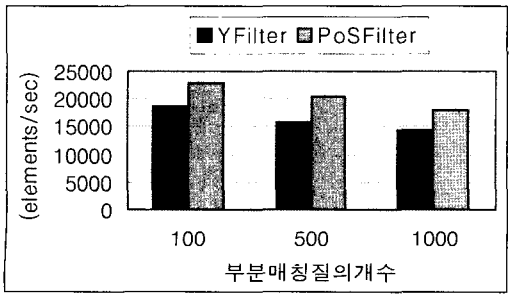


그림 10 부분매칭 경로질의 수에 따른 처리량 비교

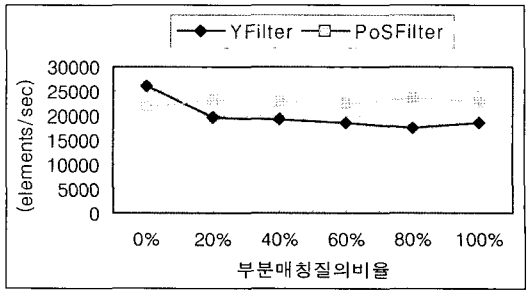


그림 11 부분매칭 경로질의 비율에 따른 처리량 비교

많다는 것은 스트리밍 XML 데이타의 엘리먼트가 들어왔을 때 그 다음에 처리해야 할 오토마타의 상태 수가 많아진다는 것을 의미한다. 그러므로 스택 안에 상태 수가 많아지면 처리량이 증가하는 문제가 발생한다.

그림 11을 보면 그림 10의 결과와 마찬가지로 YFilter는 루트로 시작하는 질의의 경우에는 스택의 오토마타 상태 수가 적지만 부분매칭 경로질의 비율이 20%만 되더라도 스택의 오토마타 상태수가 급격하게 증가하는 것을 볼 수 있다. 반면에, PoSFilter의 경우에는 부분매칭 경로질의의 비율이 변하더라도 스택의 오토마타 상태 수가 많은 차이를 보이지 않는 것을 알 수 있다. 그러므로 YFilter의 경우 부분매칭 경로질의가 20%만 되도 처리량이 떨어지지만 PoSFilter의 경우에는 부분매칭 경로질의가 증가하더라도 거의 비슷한 스택 오토마타 상태 수를 보이기 때문에 처리량이 유사하다.

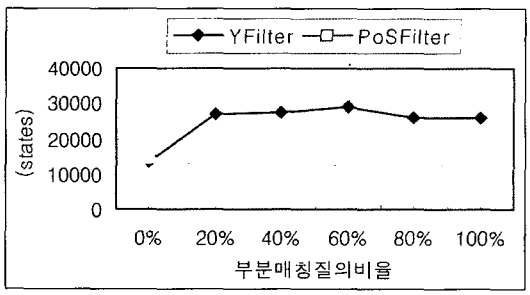


그림 12 부분매칭 경로질의 비율에 따른 스택의 오토마타 상태 수 비교

5. 결 론

센서 네트워크나 유비쿼터스 환경이 보급되면서 최근에는 저장되어 있는 데이타가 아닌 계속적으로 빠르게 지나가는 스트리밍 데이타에 대한 연구가 활발하게 이루어 지고 있다[11-13]. 기존의 Publish-Subscribe 시스템의 스트리밍 XML 데이타 필터링 기법은 오토마타를 이용한 처리에 대해서 주로 연구가 진행되었으며 연구들 중의 하나가 YFilter이다. 하지만, YFilter의 경우 부분매칭 경로질의가 증가하면 처리량이 떨어지는 문제점이 존재했다. 본 논문에서는 이런 문제점을 해결하기 위해서 새로운 스트리밍 XML 필터링 기법으로 PoSFilter를 제안하고 있다. 기존의 YFilter는 질의의 앞부분을 공유하는 질의 앞부분 공유와 하향식 방식을 사용하므로 부분매칭 경로질의와 반대의 성격을 가지기 때문에 부분매칭 경로질의에 대해서 처리량이 떨어질 수 밖에 없었다. 본 논문에서 제시한 PoSFilter는 이런 부분매칭 경로질의의 특성을 반영함으로 해서 부분매칭 경로질의가 증가할 경우에 YFilter는 처리량이 많이 감소하는 반면에 PoSFilter는 일정한 처리량을 갖고 부분매칭 경로질의가 20%이상만 되어도 더 좋은 처리량을 보였다. 다만 이 방법을 사용할 경우에 전체 질의 중에서 부분매칭 경로질의의 비중이 작을 경우에는 YFilter보다 처리량이 떨어지는 결과를 나타낼 수 있기 때문에 부분매칭 경로질의 뿐만 아니라 루트에서 시작하는 질의가 있을 경우에도 좋은 처리량을 나타낼 수 있도록 두 가지 필터링 기법을 결합하는 연구가 필요하다. 또한 본 연구에서는 XPath의 위치 경로의 탐색을 줄이는 부분만을 우선적으로 연구하고 있기 때문에 가정에서 predicate를 고려하지 않았다. 그러므로 제시한 PoSFilter에 predicate를 포함한 질의까지 처리할 수 있는 추가적인 연구가 필요하다.

참 고 문 헌

[11] B. Babcock, S. Babu, M. Datar, R. MotWani, J. Widom, "Models and Issues in Data Stream Systems," In Proceeding of the PODS, 2002, pp.1-16.

[3] Mehmet Altinel, Michael J. Franklin, "Efficient Filtering of XML Documents for selective Dissemination of Information," In Proceeding of the VLDB, 2000, pp.53-64.

[5] Yanlei Diao, Michael J. Franklin, "High-Performance XML Filtering: an overview of YFilter," Bulletin of the IEEE, 2003, pp.1-8.

[6] Yanlei Diao, Peter Fischer, Michael J. Franklin, Raymond To, "YFilter: Efficient and scalable Filtering of XML Documents," In Proceeding of the ICDE, 2002.

[4] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, Peter Fischer, "Path Sharing and Predicate Evaluation for High-Performance XML Filtering," ACM Transcations on Database Systems, 2003, pp. 467-516.

[7] Todd J. Green, Gerome Miklau, Makoto Onizuka, and Dan Suciu, "Processing XML Streams with Deterministic Automata," In Proceeding of the LNCS, 2003, pp. 173-189.

[8] Todd J. Green, Gerome Miklau, Makoto Onizuka, and Dan Suciu, "Processing XML Streams with Deterministic Automata and Stream Indexes," In Proceeding of the TODS, 2004.

[9] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, Ralph Busse, "XMark: A Benchmark for XMl Data Management," In Proceeding of the VLDB, 2002, pp.974-985.

[1] Tim Furche, "Optimizing multiple queries against XML streams," http://www.pms.ifi.lmu. de/publikationen/diplomarbeiten/Tim.Furche/mqspex.pdf

[10] Chin-Wan Chung, Jun-Ki Min, Kyuseok Shim, "APEX: An Adaptive Path Index for XML data,"

In Proceeding of the SIGMOD, 2002, pp.121-132.
[12] S. Babu and J. Widom, "Continuous Queries over data Streams," In Proceeding of the SIGMOD, 2001, pp.109-120.
[13] Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet databases," In Proceeding of the SIGMOD, 2000, pp.379-390.
[2] P.Th. Eugster, P.Felber, R. Guerraoui, A. M. Kermarrec, "The Many Faces of Publish/Subscribe," ACM Computing Serveys, 2003, pp.114-131.

박 석

1978년 서울대학교 계산통계학과(이학사). 1980년 한국과학기술원 전산학과(공학석사). 1983년 한국과학기술원 전산학과(공학박사). 1983년 9월~현재 서강대학교 컴퓨터학과 교수. 1989년~1991년/2002년~2003년 University of Virginia 방문교수. 1997년 2월~현재 한국정보보호학회 이사. 2005년. 한국정보과학회 부회장, 2004년 1월~2005년 12월 한국정보과학회지 편집위원장 1999년~현재 DASFAA Steering Committee 멤버. 관심분야는 데이타베이스 보안, 실시간 시스템, 트랜잭션 관리, 데이타웨어하우스, 웹과 데이타베이스, XML, XML 데이타베이스 시스템에서의 필터링 기법, 역할기반 접근제어 임

김 영 수

2003년 서강대학교 컴퓨터학과(공학사) 2005년 서강대학교 컴퓨터학과(공학석사). 2005년 2월~현재 삼성전자 정보통신총괄 무선사업부 소프트웨어 팀원. 관심분야는 XML 데이타 필터링 기법과 데이타 베이스