XML 레이블링을 이용한 XML 조각 스트림에 대한 질의 처리 기법

(A Query Processing Technique for XML Fragment Stream using XML Labeling)

이 상 욱 † 김 진 † 강 현 철 ††
(Sangwook Lee) (Jin Kim) (Hyunchul Kang)

요 약 유비쿼터스 컴퓨팅의 실현을 위해서는 이동 단말기의 자원 및 컴퓨팅 파워의 효율적 사용이 필수적이다. 특히, 이동 단말기에 내장된 소프트웨어의 수행에 있어 메모리 효율성, 에너지 효율성, 그리고 처리 효율성이 요구된다. 본 논문은 자원이 제약되어 있는 이동 단말기에서의 XML 데이타에 대한 질의 처리에 관한 것이다. 메모리 용량이 크지 않은 단말기의 경우 대량의 XML 데이타에 대한 질의 처리를 수행하기 위해서는 XML 스트림 질의 처리 기술이 활용되어야 한다. 최근에 제시된 XFrag는 홀-필러 모델을 이용하여 XML 데이타를 XML 조각으로 분할하여 스트림으로 전송하고 처리할 수 있는 기법이다. 이는 메모리가 부족한 이동 단말기에서 조각 스트림으로부터 XML 데이타를 재구성하지 않고 질의 처리를 가능하게 한다. 그러나 홀-필러 모델을 사용할 경우 홀과 필러에 대한 부가적인 정보를 저장해야 하므로 메모리 효율성이 높지 못하다. 본 논문에서는 XML 데이타의 구조 정보를 표현하는 XML 레이블링 기법을 이용하여 XML 데이타를 조각으로 분할하여 처리하는 새로운 기법 XFLab을 제시한다. 구현 및 성능 실험 결과 XFLab이 XFrag보다 메모리 사용량과 처리 시간 양면 모두에서 우수한 것으로 나타났다.

키워드 : XML 조각, 스트림 질의 처리, XML 질의 처리, XML 레이블링, 홀-필러 모델

Abstract In order to realize ubiquitous computing, it is essential to efficiently use the resources and the computing power of mobile devices. Among others, memory efficiency, energy efficiency, and processing efficiency are required in executing the softwares embedded in mobile devices. In this paper, query processing over XML data in a mobile device where resources are limited is addressed. In a device with limited amount of memory, the techniques of XML stream query processing need to be employed to process queries over a large volume of XML data. Recently, a technique called XFrag was proposed whereby XML data is fragmented with the hole-filler model and streamed in fragments for processing. With XFrag, query processing is possible in the mobile device with limited memory without reconstructing the XML data out of its fragment stream. With the hole-filler model, however, memory efficiency is not high because the additional information on holes and fillers needs to be stored. In this paper, we propose a new technique called XFLab whereby XML data is fragmented with the XML labeling scheme which is for representing the structural relationship in XML data, and streamed in fragments for processing. Through implementation and experiments, we showed that our XFLab outperformed XFrag both in memory usage and processing time.

Key words : XML Fragment, Stream Query Processing, XML Query Processing, XML Labeling, Hole-Filler Model

· 본 논문은 한국과학재단 특정기초연구사업(R01-2006-000-10609-0) 지원으로 수행되었음

† 학생회원 : 중앙대학교 컴퓨터공학부
swlee@dblab.cse.cau.ac.kr
jkim@dblab.cse.cau.ac.kr
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
hckang@cau.ac.kr

논문접수 : 2007년 4월 25일
심사완료 : 2007년 10월 16일

정보과학회논문지 : 데이타베이스 제35권 제1호(2008.2)

1. 서 론

XML이 웹에서 데이타 교환의 표준으로 부각된 이래, 유비쿼터스 컴퓨팅, 센서 네트워크, 모니터링 등의 응용 분야에서 XML 스트림 데이타에 대한 효율적인 질의 처리에 관한 연구가 활발히 진행되고 있다. 특히 유비쿼터스 컴퓨팅 환경의 경우, 다수의 사용자들이 PDA나 휴대폰 등의 이동 단말기를 이용하여 정보를 제공받기 때문에, 기존의 요구-응답 방식으로 질의 처리 수행 시 서버 측에 많은 부담을 주게 되어 처리 효율성이 저하된다. 따라서 이와 같은 환경에서는 서버는 데이타를 스트리밍하고, 이에 대한 질의 처리는 이동 단말기(클라이언트)에서 수행하는 기술이 요구되고 있다.

이동 단말기의 자원과 컴퓨팅 파워는 제약이 있으므로, 이동 단말기에 내장된 소프트웨어의 수행에 있어 메모리 효율성, 에너지 효율성, 그리고 처리 효율성이 요구된다. 본 논문은 자원이 제약되어 있는 특히, 가용 메모리 용량이 적은 이동 단말기에서의 XML 데이타에 대한 질의 처리 기술에 관한 것이다. XML 데이타는 계층적 구조를 가지고 있으며 용량이 클 수 있다. 따라서 이동 단말기의 적은 메모리로 대량의 XML 데이타를 처리하려면, XML 데이타를 적절한 조각(fragment)으로 분할(fragmentation)하여 조각 단위로 처리하는 기술이 요구된다[1]. XML 분할에 의해 얻어지는 이점은 다음과 같다. 센서 기반의 환경 등에서 생성되는 XML 스트림 데이타를 구조적으로 분할하여 XML 조각 스트림으로 전송하고 처리하면, 이동 단말기의 메모리 효율성 및 처리 효율성을 제고할 수 있다. 또한 XML 스트림에서 어떤 특정 정보가 갱신될 경우, 전체 XML 데이타를 재전송하지 않고 변화된 조각만 전송함으로써 송수신 비용에서 이득을 얻을 수 있다.

홀-필러(hole-filler) 모델[2,3]은 XML 데이타를 구조적으로 분할하기 위한 것이다. 이를 이용하여 XFrag[4]나 XFPro[5]는 XML 조각 처리 기법을 제시하고 있는데, 홀-필러 모델이 필요로 하는 부가 정보로 인하여 낮은 처리 속도 및 메모리 공간 낭비 등의 문제점이 있다. XFPro는 XFrag의 질의 연산자 파이프라인(pipeline)을 개선하여 속도를 향상시켰으나, 홀-필러 모델이 가지고 있는 문제점에 대해서는 해결 방안을 제시하지 않았다.

본 논문에서는 이와 같은 문제를 해결하기 위해 기존의 홀-필러 모델을 폐기하고, XML 데이타의 구조 정보를 표현하는 XML 레이블링(labeling) 기법을 이용하여 XML 데이타를 조각으로 분할하고 XML 조각 스트림에 대한 질의 처리를 수행하는 기법을 제시한다. 이 기법은 기존의 홀-필러 모델이 필요로 하는 부가 정보의 양을 줄여서 XML 조각을 처리하는 데 소요되는 메모리 양 및 시간을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 살펴본다. 3절에서는 본 논문에서 제시하는 XML 조각 스트림에 대한 질의 처리 기법을 기술한다. 4절에서는 구현 및 성능 평가 결과를 기술한다. 5절에서는 결론을 맺고 향후 연구 내용을 기술한다.

2. 관련 연구

XML 스트림에 대한 질의 처리 기법 연구로는, publish-subscribe, SDI(selective dissemination of information) 등의 응용을 위한 XML 스트림 필터링 기법인 XFilter[6], XTrie[7], YFilter[8], XPush Machine [9], XMLTK [10], FiST [11], AFilter [12] 등이 제안되었다. 또한 저장되어 있는 XML 데이타에 대한 질의 처리가 아닌 스트리밍되는 XML 데이타에 대한 질의 처리 기법으로 x-scan[13], XSM[14], XSQ[15], BEA/XQRL [16], XStreamQuery[17], FluXQuery[18] 등이 제안되었다. 그러나 이들 기법은 XML 문서를 조각들로 분할한 후 임의의 순서로 스트리밍될 때의 질의 처리 기법은 아니다. XML 데이타를 문서 단위 또는 파싱되는 이벤트 단위로 스트리밍한다. 또한 오토마타를 이용하거나 상태 정보의 저장을 위한 많은 양의 메모리를 필요로 하는 등, 본 논문에서 목표로 하는, 질의 처리가 수행되는 이동 단말기에서의 메모리 사용량 절약을 위한 기법은 아니다. XML 조각과 관련된 연구로는, 무선 환경에서 XML 문서를 조각들로 분할하여 임의의 순서로 스트리밍하고 수신 측에서 이를 원래 문서로 재구성하는 Xstream 기법 [19]이 있고, 웹 서비스 응용에서 XML 문서를 구성하는 조각들을 별도로 관리 및 제공하는 active XML 기법 [20] 등이 있다.

XML 문서를 조각으로 분할하여 임의의 순서로 스트리밍하고 이러한 조각 스트림에 대해 질의 처리를 수행하는 기법으로 XFrag[4]가 처음 제안되었다. XFrag는 조각 간의 관계를 기술할 수 있는 기법이 필요한데 이를 위해 홀-필러 모델[2,3]을 사용한다. 이 모델은 조각 사이의 관계를 "홀(hole)"에 대응하는 "필러(filler)"의 관계로 기술하고 있다. 전체 문서는 조각으로 분할될 때 분할될 문서의 위치에 "홀 엘리먼트"를 추가하고 "홀 식별자(hole id)"를 부여한다. 분할되는 문서는 "필러 엘리먼트"로 감싸지고 홀 식별자와 동일한 "필러 식별자(filler id)"가 부여되어 분할된다.

그림 1~그림 3은 각각 분할되기 전의 XML 문서, 이를 분할하기 위한 "태그 구조(tag structure)", 그리고 홀-필러 모델 기반의 조각으로 분할된 XML 문서를 나타낸 것이다. 태그 구조는 전체 문서의 구조를 요약하고

```
<commodities>
   <vendor>
      <name> Wal-Mart </name>
        <items>
          <item>
             <name> PDA </name>
             <make> HP </make>
             <model> PalmPilot </model>
             <price currency="USD">315.25<price>
          </item>
          <item>
             <name> Calculator </name>
             <make> Casio </make>
             <model> FX-100 </model>
             <price currency="USD">50.25<price>
          </item>
          ...
        </items>
      </name>
   </vendor>
   ...
</commodities>
```

그림 1 분할 전 XML 문서

```
<stream:structure>
   <tag name="commodities" id="1" filler="true">
      <tag name="vendor" id="2">
         <tag name="name" id="4"/>
         <tag name="items" id="5">
            <tag name="item" id ="6" filler="true">
               <tag name="name" id="7"/>
               <tag name="make" id="8"/>
               <tag name="model" id="9"/>
               <tag name="price" id="10"/>
            </tag>
         </tag>
      </tag>
   </tag>
</stream:structure>
```

그림 2 태그 구조

```
<stream:filler id="1" tsid="1">
   <commodities>
      <vendor>
         <name> Wal-Mart </name>
           <items>
             <stream:hole id="10" tsid="6"/>
             <stream:hole id="20" tsid="6"/>
             ...
      </vendor>
      ...
   </commodities>
</stream:filler>
```

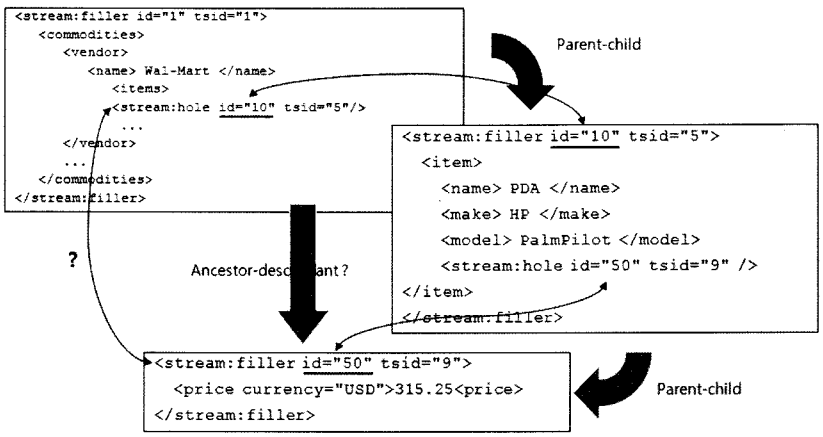
```
<stream:filler id="10" tsid="6">
   <item>
     <name> PDA </name>
     <make> HP </make>
     <model> PalmPilot </model>
     <price currency="USD">315.25<price>
   </item>
</stream:filler>
```

```
<stream:filler id="20" tsid="6">
   <item>
     <name> Calculator </name>
     <make> Casio </make>
     <model> FX-100 </model>
     <price currency="USD">50.25<price>
   </item>
</stream:filler>
```

그림 3 홀-필러 모델을 이용하여 분할된 XML 문서

대상 XML 문서의 모든 엘리먼트에 대해 엘리먼트의 구조적인 위치를 식별할 수 있는 "태그 식별자(tsid)"를 표기하고 있으며 또한 각 엘리먼트가 독립된 조각으로서 XML 문서의 하위 문서를 분할할 것인가를 정의한다. XFrag 연구에서는 이와 같이 분할된 XML 조각 스트림을 받아서 XML 질의 처리를 수행할 수 있는 질의 연산자 파이프라인 처리 기법을 제시하였다.

홀-필러 모델 및 XFrag의 문제점은 다음과 같다. 첫째, 동일한 홀 식별자와 필러 식별자를 두 개의 조각에 표기함으로써 분할된 문서의 크기가 상당히 증가하는 경향이 있다. 이러한 특징은 스트리밍해야 할 XML 조각의 크기가 증가한다는 것을 의미한다. 둘째, 일반적으로 웹 상의 XML 문서는, 트리로 볼 때, 높이는 크지 않고 옆으로 넓게 퍼진 모양을 하고 있다[21]. 이는 XML 트리 내에 자식 노드 또는 자식 서브 트리를 다수 가진 노드가 많다는 것을 의미하고, 이러한 문서를 홀-필러 모델에 의해 분할하게 되면 자연히 매우 많은 홀 식별자를 포함하는 조각이 생길 수 밖에 없다. 이렇게 많은 홀 식별자로 크기가 커진 조각은 이동 단말기가 처리하는 데 많은 비용이 들며 단말기 내에 많은 홀 정보를 저장하여야 하기 때문에 메모리 사용량이 증가한다. 특히, XML 문서에 엘리먼트가 추가되는 상황에서는 추가된 엘리먼트의 홀 식별자를 포함하는 조각과 실제 추가된 조각 모두를 전송하지 않으면 단말기에서 처리할 수 없다. 셋째, 홀-필러 모델은 부모 조각과 자식 조각의 관계만을 기술하므로 홀-필러 모델을 사용하는 XFrag 등의 기존 기법에서는 XPath 질의의 조상-후손 축(//)을 자식 축(/)만으로 구성된 경로로 변환하여 경로상의 모든 조각을 처리하지 않으면 질의 처리를 수행할 수 없다. 이것은 홀-필러 모델이 어떤 조각에 대해 그 조각의 자식 조각과의 관계만을 나타내기 때문이다. 이로 인해 홀-필러 모델은 부모-자식관계가 아닌 임의의 두 조각 간의 조상-후손 관계 파악을 위해서는 두 조각을 잇는 경로 상에 위치한 모든 조각들을 조사하는 것을 필요로 한다. 그림 4는 이러한 문제의 예 보인 것이다. 그림 4의 세 조각 중 좌측 편의 두 조각에 대해, XPath 질의 "/commodities//currency"를 처리하기 위해서는 필러 식별자가 1인 조각의 commodities 엘리먼트와 필러 식별자가 50인 조각의 currency 엘리먼트가 서로 조상-후손 관계라는 사실을 파악해야 하는



XPath 질의:/commodities//currency

그림 4 홀-필러 모델에서 후손 축(//) 처리의 문제점

데, 홀-필러 모델에서는 이를 위해 commodities 엘리먼트에서 currency 엘리먼트까지의 경로 상에 위치하는 모든 조각들을 조사하여야 한다. 그림 4의 예에서는 필러 식별자가 10인 조각이 이러한 경로상의 조각에 해당된다. XFrag에서는 이러한 경로 상의 조각을 처리하기 위해 "/commodities//currency"와 같이 //가 포함된 질의를 "/commodities/vendor/name/items/item/currency"와 같이 자식 축으로만 구성된 질의로 변경하여 처리한다. 이와 같이 질의 처리를 수행할 때 경로 상의 모든 조각들을 조사해야 한다는 것은 경로 상에 위치하는 모든 조각의 정보를 저장해야 한다는 것을 의미하며 이는 메모리 사용량의 증가를 초래한다.

XFPro[5]의 경우, 홀-필러 모델의 태그 구조를 이용하여, 질의 처리 계획을 생성하고 질의 연산자 파이프라인의 최적화를 수행한다. 최적화를 통해 생성된 파이프라인은 기본적으로 XFrag에서 사용하는 파이프라인과 동일하지만, XFrag에 비해 XML 조각을 처리하는 단계가 줄어들게 되어 처리 시간에서 개선 효과를 볼 수 있다. 하지만, XFPro도 홀-필러 모델을 기반으로 하기 때문에 위에서 언급한 문제점들을 그대로 가지고 있다.

본 논문에서는 기존의 홀-필러 모델 기반의 조각 관계 표현 기법을 폐기하고 XML 레이블링 기법을 이용하여 조각 간의 관계를 표현하였다. 홀-필러 모델에서는 중복되지 않는 숫자로 홀 및 필러 식별자를 부여하고 조각 간 부모 자식 관계를 홀 및 필러 식별자 비교로 식별하였는데 반해, 본 논문에서는 XML 레이블링 기법을 사용해서 각 조각에 "조각 식별자(fragment id)"를 부여하고 조각 식별자를 비교함으로써 조각 사이의 부모-자식 관계 뿐 아니라 조상-후손 관계를 식별할 수 있도록 하고, 기존 기법들에서 메모리 사용량 부담의 주요 원인이었던 홀이 (따라서 홀 식별자 및 홀 엘리먼트가) 모두 제거되도록 하였다.

3. XML 레이블링 기법을 이용한 XML 조각 스트림 질의 처리 기법

XML 레이블링 기법은 XML 문서 내의 노드 간의 구조 관계를 표현하여 XML 질의 처리에 이용하기 위한 것이다. 이러한 XML 레이블링 기법은 프리픽스(prefix) 레이블링과 범위(range) 레이블링 기법으로 대별되며 현재까지 가장 진보된 기법들로서 ORDPATH [22], QED[23] 등이 제안되었다. 본 절에서는 프리픽스 레이블링 기법을 이용하여 조각의 구조 관계를 표현하고 이러한 관계 표현을 이용하는 새로운 조각 스트림 질의 처리 기법 XFLab을 제안한다. XML 질의로는 XPath 질의를 고려하였고 XML 레이블링 기법으로는 Dewey order encoding[24]을 사용하였다.

3.1 XML 레이블링 기법을 이용한 조각 식별자 할당 및 조각 식별자 관계 비교

XML 문서는 엘리먼트, 애트리뷰트, 텍스트 등을 노드로 하는 트리로 나타낼 수 있는데, 조각으로 분할된 XML 문서 또한 트리로 표현할 수 있다. 분할된 문서의 각 조각은 어떤 엘리먼트 노드를 루트로 하는 원래 XML 트리의 서브 트리이다. 따라서 각 조각은 해당 서브 트리의 루트 노드의 부모 노드가 소속된 조각을 그 부모 조각으로 갖는다. 다시 말해서 분할되어 나온 조각은 자신이 원래 속해 있던 조각을 부모 조각으로 가지게 된다. 결국 부모-자식 간의 관계를 구성하면서 태그 구조에 의해 분할된 조각 간의 관계는 일반적인 트리의

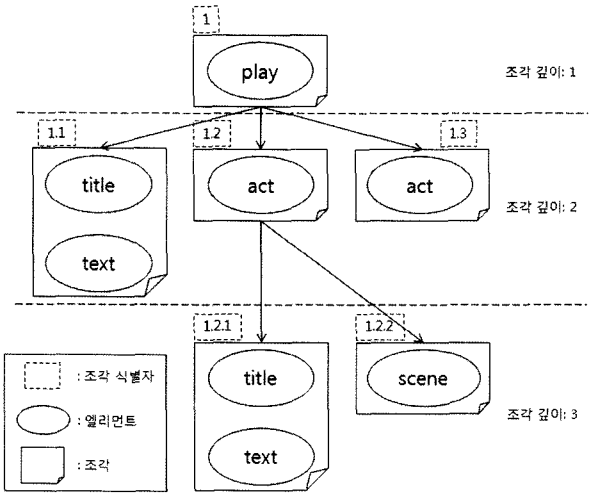


그림 5 Dewey order encoding을 사용하여 조각 식별자를 부여한 예

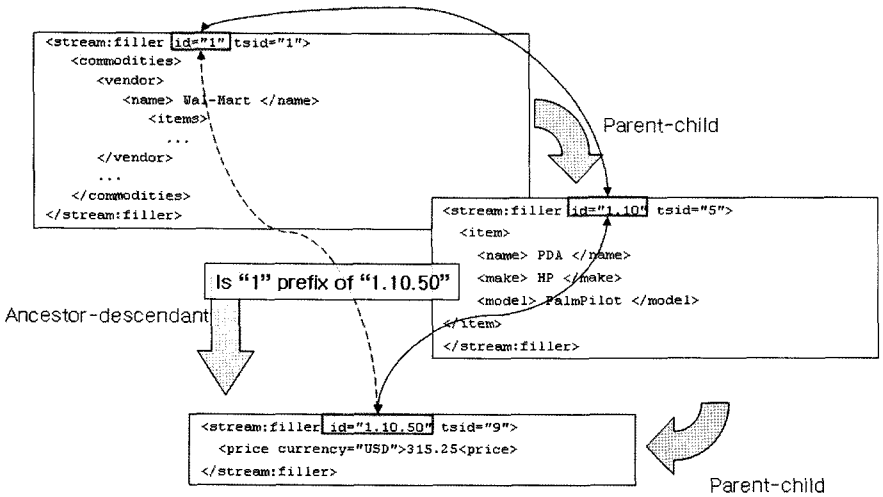


그림 6 XML 레이블링 기법 기반의 XML 문서 분할에서 조각 간 관계 비교

모양이 되며 이를 "조각 트리"라고 한다. 조각 트리의 노드는 XML 조각인데, 조각 트리에서 어떤 노드의 깊이를 "조각 깊이(fragment depth)"라고 정의한다.

분할된 조각은 조각 내 루트 엘리먼트의 태그 식별자와 함께 전체 조각 트리에서 자신의 고유한 위치를 식별하는 조각 식별자를 XML 레이블링 기법에 의해 할당받는다. 그림 5는 Dewey order encoding을 사용하여 XML 조각에 조각 식별자를 할당한 예이다. 각 조각 별로 그 조각 식별자와 조각 깊이를 나타내었다. Dewey order encoding은 루트 노드로부터 현재 노드까지의 경로에 관한 모든 정보를 저장함으로써 전체 트리에서 특정 노드의 위치를 식별하고 또한 조각 간에 부모-자식 관계와 조상-후손 관계를 파악할 수 있게 한다. 그림 6은, 그림 4에서 홀-필러 모델이 갖는 문제점의 하나로 조상-후손 관계의 직접적인 파악이 불가능함을 예시했던 것에 반해 XFLab에서는 조각 식별자로써 부모-자식 관계뿐만 아니라 조상-후손 관계 파악이 가능함을 예시한 것이다.

3.2 XML 문서 분할 요구 사항

XML 문서를 임의로 분할하여 조각 트리를 형성한 후 XML 레이블링 기법에 의해 각 조각들에 조각 식별자를 할당할 경우 문서 내 엘리먼트 간의 포함 관계 표현에 있어 모호성(ambiguity)을 초래할 수 있다. 그림 7은 이와 같은 경우의 예를 나타낸 것이다. 그림 7(a)는 예로 든 XML 문서이고 그림 7(b)는 이것을 트리 구조로 나타낸 것이다. 그림 7(c)와 7(c')는 7(a)의 문서를

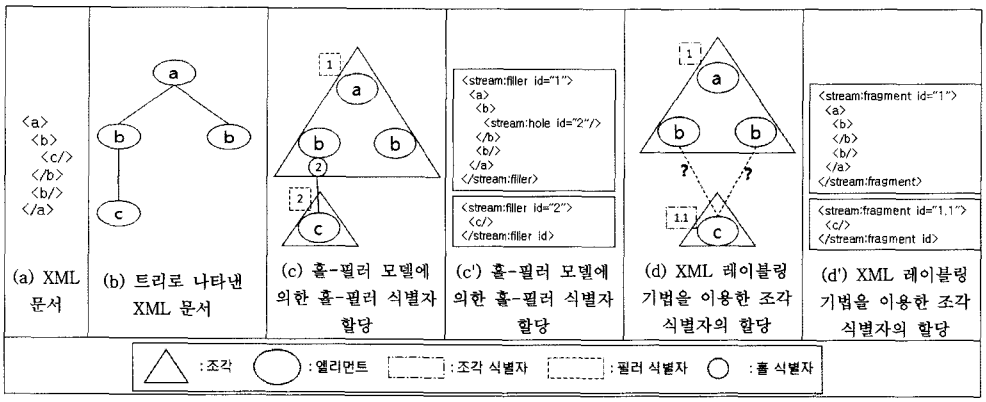


그림 7 레이블링 기법을 통한 조각 식별자 할당으로 관계 식별이 불가능한 예

조각으로 분할한 후 홀-필러 기법에 의해 홀 식별자 및 필러 식별자를 할당한 것을 나타낸 것이고, 그림 7(d)와 7(d′)는 동일한 분할에 대해 XML 레이블링 기법을 이용하여 조각 식별자를 할당한 것을 나타낸 것이다. 그림 7(c) 및 7(c′)에서 엘리먼트 "c"가 첫 번째 "b" 엘리먼트의 자식 엘리먼트임은 필러 식별자 2의 조각 내에 존재하는 홀 엘리먼트 (즉, 그림 7(c′)의 "<stream:holeid ="2"/>")를 통해 알 수 있다. 이와 같이 홀-필러 모델을 이용하여 홀-필러 식별자를 할당할 경우 홀 엘리먼트를 이용하여 조각 내부에 명시적으로 자식 조각의 위치를 지정하므로 엘리먼트 사이의 관계를 완전히 표현할 수 있다. 이에 반해 그림 7(d)와 7(d′)의 경우 조각 1.1의 엘리먼트 "c"가 조각 1의 "b" 엘리먼트의 자식인 것은 태그 구조를 통해 알 수 있지만 조각 1의 두 "b" 엘리먼트 중 어느 것의 자식인지는 알 수 없다.

이러한 문제를 해결하기 위해 본 논문에서 제시하는 XML 레이블링 기법을 이용한 조각 질의 처리 기법에서는 XML 문서를 조각으로 분할할 때 DTD를 참조하여 "*" 또는 "+" 연산자가 명시된 엘리먼트의 각 인스턴스는 서로 다른 조각에 소속되도록 분할한다. 예를 들어, 그림 8(a)의 DTD를 준수하는 문서의 경우 "*" 연산자가 명시된 book 엘리먼트의 인스턴스들은 각각 서로 다른 조각에 소속되도록 분할된다. 즉, 두 개 또는 그 이상의 book 엘리먼트들이 동일한 조각에 소속되는 일이 없도록 하는 것이다. 그림 9는 이와 같은 분할을 통해 얻은 조각 트리에 대해 홀-필러 모델을 이용한 홀 식별자 및 필러 식별자 할당 (그림 9(a) 및 9(a′)) 및 XML 레이블링 기법을 이용한 조각 식별자 할당 (그림 9(b) 및 9(b′))을 나타낸 것이다.

```
<!ELEMENT bib (vendor*)>
<!ELEMENT vendor (name,email,book*)>
<!ATTLIST vendor id ID #REQUIRED>
<!ELEMENT book (title,publisher?, year?, price, author+ )>
<!ATTLIST book ISBN ID #REQUIRED>
<!ATTLIST book related_to IDrefs>
<!ELEMENT author (firstname?, lastname)>
```

그림 8 DTD 예와 반복적으로 나타나는 엘리먼트의 분할 예

DTD에서 "*" 또는 "+" 연산자가 명시된 엘리먼트의 각 인스턴스는 서로 다른 조각에 소속되도록 분할한다는 것은 본 연구에서 제시하는 XML 레이블링 기법을 이용한 XML 문서 분할에 있어 유일한 요건이다. 이러한 요건은 문서 분할의 일반성을 떨어뜨리는 제약이기보다는 대용량 XML 문서의 특징 (높이는 그리 크지 않고 옆으로는 "*" 또는 "+" 연산자에 의해 넓게 퍼짐[21])을 고려하였을 때 일반적으로 자연스러운 분할을 초래한다. 일반적으로 DTD에서 "*" 또는 "+" 연산자가 명시되어 있어 XML 문서에 반복적으로 나타나는 엘리먼트 인스턴스는 각각 실세계의 하나의 개체에 대응되는 것이 보통이므로 본 논문의 문서 분할은 결국 실세계의 각 개체를 별도 조각으로 표현하는 분할에 해당된다.

3.3 XFLab 질의 연산자 파이프라인

본 절에서는 3.1절에서 기술한 XML 조각 간 관계 표현을 바탕으로 XFLab의 질의 연산자 파이프라인의 개요를 기술한다. XFLab의 질의 연산자 파이프라인 생성은 XFrag의 그것보다 간단하다. XFrag가 사용하는 홀-필러 모델에서는 부모-자식 관계만을 파악할 수 있으므로 조상-후손 축이 포함된 XPath 질의 처리를 위해서는 질의의 조상-후손 축을 자식 축만으로 구성된 경로로 변환해야 하였으나, XFLab은 조상 후손의 관계를 직접 확인할 수 있으므로 그러한 질의 변환이 불필요하다. 따라서 XFLab에서는 이러한 질의 변환 없이 XPath 질의의 각 스텝(XPath location step)별로 연산자를 구

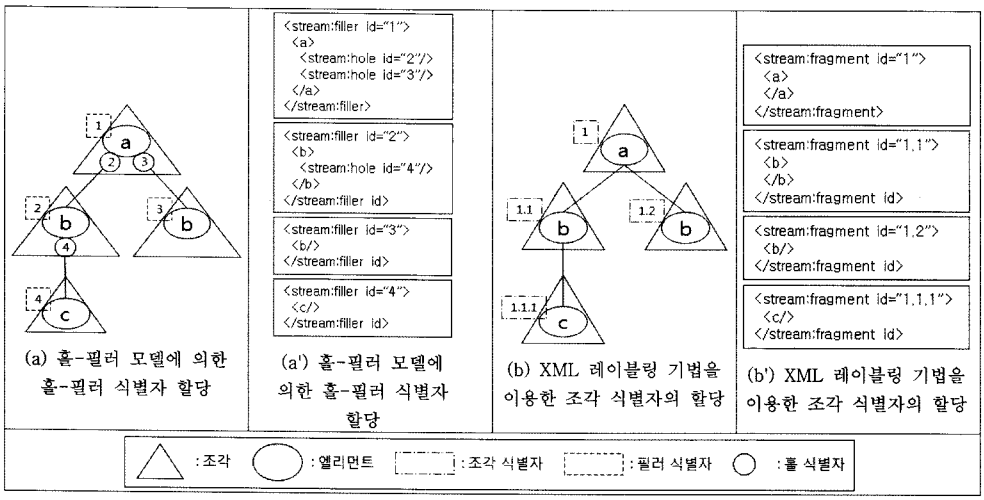
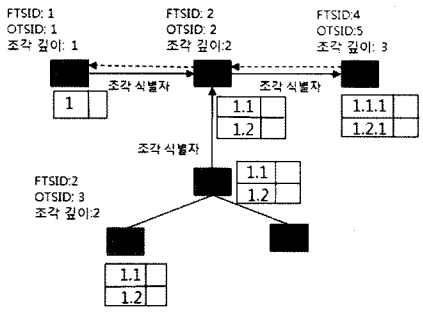


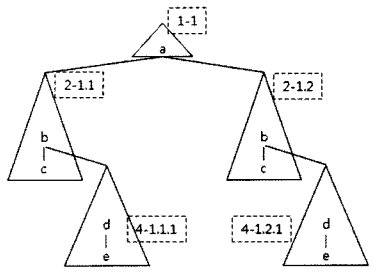
그림 9 3.2절의 분할 요건을 적용한 분할 예

성하고, 각각의 연산자에 XPath 질의의 이전 스텝과 다음 스텝에 해당하는 연산자들을 참조할 수 있는 링크를 생성하여 질의 연산자 파이프라인을 생성한다. 연산자는 연산자로 변환되기 전의 XPath 질의 스텝과 매치되는 엘리먼트에 대한 질의 처리를 수행하게 되는데 이 엘리먼트를 연산자의 "타겟 엘리먼트(target element)"라고 한다. 질의 처리 수행 시 연산자는 자신의 타겟 엘리먼트들에 대한 평가 결과를 해당 엘리먼트가 포함된 조각의 조각 식별자와 함께 "연관 테이블(association table)"에 저장한다. 연관 테이블에 저장되는 조각 식별자와 해당 조각에 포함된 타겟 엘리먼트 평가 결과 쌍을 "테이블 엔트리(table entry)"라 한다. 타겟 엘리먼트 평가 결과는 참, 거짓, 미결정(undecided)의 값을 가질 수 있다. 연산자에는 타겟 엘리먼트를 식별하기 위해 타겟 엘리먼트의 태그 식별자 값이 할당되는데 이것을 "연산자 태그 식별자(Operator TSID, 이하 OTSID)"라고 하며, 연산자의 타겟 엘리먼트가 포함된 조각을 연산자의 "타겟 조각(target fragment)"이라 한다. 그런데 XML 문서 분할 시에 조각에 기록되는 태그 식별자는 조각의 루트 엘리먼트의 태그 식별자 값이므로 OTSID 값만으로는 타겟 조각을 식별할 수 없다. 이와 같은 이유로 연산자의 타겟 조각을 식별하기 위해 OTSID와는 별도로 타겟 조각의 태그 식별자 값을 연산자에 할당하는데, 이 값을 "타겟 조각 태그 식별자(target Fragment TSID, 이하 FTSID)"라고 한다. 또한 연산자에는 타겟 조각의 조각 깊이 값도 할당된다.

XFLab 질의 연산자 파이프라인은 XML 조각이 도착하면 조각의 TSID와 동일한 FTSID를 가지는 연산자를 찾아 해당 연산자의 연관 테이블에 기록하고 연산자가 조각에 대한 평가(evaluate)를 수행하도록 한다. 그림 10은 XPath 질의 "/a/b[c=3]//e"에 대한 연산자 파이프라인, 스트리밍되는 조각 트리 및 이 조각 트리가 모두 스트리밍되었을 때 연산자 별로 할당된 연관 테이블에 저장되어 있는 조각 식별자 값의 예를 나타낸 것이다. 그림 10(a)는 XPath 질의 "/a/b[c=3]//e"에 대해 생성된 XFLab 연산자 파이프라인을 나타낸 것이다. XPath 질의의 경로 스텝(path step)은 경로 연산자(그림 10(a)의 "/a", "/b", "/c", "//e")로 변환되어, XML 문서에서 해당 연산자의 OTSID와 동일한 태그 식별자를 갖는 엘리먼트에 대한 평가를 수행한다. XFLab 연산자 파이프라인에서는 그림 10(a)의 연산자 "/c", "//e"의 경우와 같이 OTSID와 FTSID가 다를 수도 있다. 이때는 "c", "e" 엘리먼트가 각각 연산자의 FTSID로 식별된 조각 내에 위치하는 경우이기 때문으로 문제가 되지 않는다. XPath 질의의 프레디킷(predicate) 표현식은 조건 연산자(그림 10(a)의 "=")로 변환되어 프레디킷에 명시된 조건을 평가하게 된다. 각각의 연산자는 하위 연산자(XPath 질의의 다음 스텝에 해당하는 연산자)에게 XML 조각을 넘겨줄 수 있는 링크(link)를 지닌다. 이는 XML 조각에 포함된 엘리먼트가 단일 엘리먼트가 아닌 경우, 해당 조각에 대한 평가가 하위 연산자에서도 이루어질 수 있기 때문이다. 그림 10의 예에서, "b" 엘리먼트와 "c" 엘리먼트가 동일한 조각에 포함되므로, 이 조각은 질의 처리 시 "/b" 연산자와 "/c" 연산자 모두에서 평가가 이루어져야 한다. 또한 각각의 연산자는 자신의 상위 연산자를 참조할 수 있는 링크를 가지게 되는데, 이는 연산자에서 엘리먼트에 대한 평가를 수행할 때, 해당 엘리먼트가 포함된 조각의 조상 조각에 대한



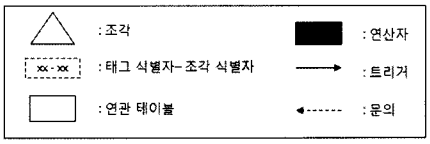
(a) 질의 연산자 파이프라인



(b) 조각 트리

```
<tag stream="ex">
  <tag id="1" name="a" filler="true">
    <tag id="2" name="b" filler="true">
      <tag id="3" name="c" filler="false"/>
      <tag id="4" name="d" filler="true">
        <tag id="5" name="e" filler="false"/>
      </tag>
    </tag>
  </tag>
</tag>
```

(c) 태그 구조



(d) 범례

그림 10 XFLab 질의 연산자 파이프라인 및 연관 테이블의 상태 예

평가 결과를 참조해야 하기 때문이다.

그림 10(b)는 스트리밍되는 조각 트리를 나타낸 것이다. 각 조각에는 조각의 태그 식별자로서 조각의 루트 엘리먼트의 태그 식별자와 조각 식별자를 "-"로 구분하여 표시하였으며 조각들의 내부에는 각 조각이 어떠한 엘리먼트를 포함하는지를 나타내었다. 그림 10(b)의 조각들이 모두 전송되었을 때 연산자 파이프라인의 각 연산자에 할당된 연관 테이블에 저장되는 조각 식별자 값을 그림 10(a)에 나타내었다. 그림 10(c)는 이 예에서 사용한 태그 구조를 나타낸 것이고, 그림 10(d)는 범례이다.

3.4 XFLab 질의 연산자 파이프라인 처리 알고리즘

본 절에서는 XFLab 파이프라인의 질의 처리 알고리즘을 설명한다. XFLab 파이프라인에서 "평가(evaluate)"와 "조상 문의(ancestor inquiry, 이하 문의)", "후손 트리거(descendant trigger, 이하 트리거)"는 연산자 파이프라인을 통해 질의를 처리하는 데 있어 핵심적인 동작이다. 문의는 주어진 XML 조각을 처리하는 데 있어 현재의 연산자에서 상위 연산자 즉, XPath 질의의 이전 스텝에 해당되는 연산자와의 연관성을 파악하여 필요한 작업을 수행하기 위한 과정이며, 트리거는 하위 연산자 즉, 이후 스텝에 해당되는 연산자와의 연관성을 파악하여 필요한 작업을 수행하기 위한 과정이다. XFLab 질의 연산자 파이프라인 처리 알고리즘의 개괄적인 동작은 다음과 같다. 먼저 XFLab 질의 연산자 파이프라인은 조각이 도착하면 도착한 조각이 질의 처리와 유관한 조각인지를 연산자들에 할당된 FTSID를 조사하여 확인한다. 만약 어떤 연산자의 FTSID가 도착한 조각의 TSID와 일치한다면, 이 연산자에게 조각을 전달하면서 조각의 처리를 수행하도록 한다. 조각을 전달받은 연산자가 조각을 처리하는 과정을 그림 11의 알고리즘 1에 나타내었다. 조각을 전달받은 연산자는 먼저 타겟 엘리먼트에 대해 조각을 파싱하기 위해 OTSID 값으로 파싱 핸들러를 설정한다. 파싱이 완료되면 조각 식별자 값과 타겟 엘리먼트의 값으로 이루어진 연관 테이블 엔트리를 얻고 이것을 연관 테이블에 저장한다. 연산자는 연관 테이블에 저장한 엔트리의 값을 평가(evaluate)하는 작업을 수행하기 전에 그것이 필요한 동작인지 상위 연산자에게 문의한다. 이때 엔트리의 조상 조각 정보를 기록한 엔트리를 상위 연산자의 연관 테이블에서 찾아내기 위해 조각 식별자를 사용한다. 문의의 결과에 따라 조각에 대한 평가가 수행되고 평가 결과에 따라 트리거를 수행하는데, 트리거의 경우에도 연산자가 트리거하려는 후손 엔트리를 식별하기 위해 조각 식별자를 사용한다. 최하위 연산자(XPath 질의의 마지막 스텝에 해당하는 연산자)는 타겟 엘리먼트에 대한 평가가 수행되더라도 평가 결과를 저장하지 않고 타겟 엘리먼트의 내용을 저장한다. 이것은 최하위 연산자는 조각에 대한 평가 결과가 참으로 결정되면 해당 엘리먼트를 결과로 출력해야 하기 때문이다. 이상에 대한 자세한 알고리즘은 아래에서 기술한다.

3.4.1 문의(inquiry)

연산자에서 XML 조각을 평가할 때, 각각의 XML 조각은 전체 XML 문서 내에서 특정 부분에 해당하므로

알고리즘 1: processing_fragment

```
입력 : fragment 평가 대상 조각
변수 : handler 파서의 핸들러
       parser XML 파서
       association_table 연산자의 연관 테이블

   /*파싱 핸들러가 현재 연산자의 타겟 엘리먼트에 대해 파싱하도록 OTSID를 핸들러에 할당 */
1: handler.setTSID( OTSID );
   /* 질의 처리 연산자 파이프라인에게 넘겨 받은 조각을 파싱 */
2: parser.parse( fragment, handler );
   /* 파싱 결과로 얻어진 조각 식별자와 타겟 엘리먼트의 내용을 받음
3: table_entry = handler.table_entry;
   /* 연관 테이블에 저장 */
4: association_table.put( table_entry.fragment_ID, table_entry );
   /* 상위 연산자에 문의 */
5: inquiry_result = inquiry( fragmentID );

   /* 문의 결과와 함께 조각의 평가를 수행 */
6: eval_result = evaluate( inquiry_result, fragment_ID );
   /* 트리거 수행 */
7: IF (eval_result != UNDECIDED)
8:       trigger( eval_result, fragment_ID );    /* 평가 결과가 미결정이 아니라면 트리거를 수행 */
   END IF
```

그림 11 알고리즘 조각 처리

알고리즘 2: inquiry

```
입력 : 평가 대상 조각의 식별자(fragment_ID)
출력 : 조상 조각의 평가 결과(TRUE, FALSE, UNDECIDED)
변수 : predecessor 상위 연산자
       association_table 연산자의 연관 테이블

  /* 상위 연산자가 존재하는 경우 */
1: IF (predecessor != null) THEN

       /* 상위 연산자와 현재 연산자가 다른 조각에 포함되어 있는 경우 */
2:     IF (FTSID != predecessor.FTSID) THEN
                /* 상위 연산자의 조각 깊이 할당 */
3:              depth = predecessor.fragment_depth;

                /* 현재 연산자에 도착한 조각의 조각 식별자를
                   상위 연산자의 조각 깊이 값에 맞는 조각 식별자로 변환 */
4:              fragment_ID = convert_fragment_ID( fragment_ID, depth );
       END IF

       /* 해당 조각 식별자로 연관 테이블 엔트리를 가져온다 */
5:     table_entry = predecessor.association_table.get( fragment_ID );
6:     IF (table_entry == null) THEN              /* 해당 조각이 도착하지 않은 경우 */
7:              result = UNDECIDED;
       ELSE
8:              result = table_entry.value;     /* 반환할 해당 조각의 결과 */
       END IF
   ELSE
9:     result=TRUE;                               /* 최상위 연산자인 경우 */
   END IF
10: return result;
```

그림 12 알고리즘 문의(inquiry)

XPath 질의 처리를 위해서는 조각 간의 평가 결과를 참조해야 할 필요가 있다. 예를 들어, 연산자에서 특정 XML 조각의 평가 결과가 거짓(false)으로 결정되었다면, 해당 조각의 후손에 해당하는 XML 조각들은 평가할 필요가 없으며, 값은 거짓으로 결정된다. 따라서 연산자에서 조각을 평가하기에 앞서 해당 조각의 조상에 해당하는 조각의 평가 결과를 확인해야만 한다. 그림 12의 알고리즘 2는 이와 같은 과정을 나타내고 있다.

알고리즘 2는 입력으로 후손 조각의 식별자를 받아서 해당 조각의 부모 혹은 조상에 해당하는 조각의 평가 결과를 출력한다. 상태는 참, 거짓, 미결정(undecided)의 세 가지로 나누어진다. 참은 조상 조각이 조건식을 만족하였거나 경로 스텝에 해당하는 경우이며, 이 경우 후손 조각들에 대한 평가는 수행되어야 한다. 거짓은 조상 조각이 조건식을 만족하지 못하였거나 그것의 조상에 해당하는 조각의 평가 결과가 거짓인 경우로, 후손 조각들은 평가될 필요 없이 항상 거짓이다. 미결정 상태는 문의를 요청한 조각의 조상이 아직 도착하지 않아서 평가되지 못했음을 의미한다. 이 경우 조각은 조상 조각의 결과가 나오기 전까지 평가될 수 없는데 이는 상위 연산자의 참 거짓에 따라 평가 결과가 달라지기 때문이다. 이런 이유로 문의 결과가 미결정 상태라면 질의 평가는 보류되고 조상 조각 도착에 의한 상위 연산자의 트리거를 기다리게 된다. 상위 연산자에게 평가 결과를 문의하는 과정에서 핵심적인 동작은 후손 조각의 식별자를 변환하여 조상 조각을 찾아내는 것이다. 기존의 XFrag나 XFPro 기법은 문의를 요청한 조각의 부모 조각을 찾기 위해 연관 테이블 외에 추가적인 자료 구조를 유지한다. 이 자료 구조는 자식 조각의 식별자와 이를 포함하는 부모 조각의 식별자를 쌍으로 저장하는 맵(map)으로, 자식 조각의 식별자를 이용해 부모 조각을 빠르게 찾을 수 있는 인덱스의 역할을 한다. 하지만 이러한 자료 구조를 유지함으로써 연관 테이블에 저장된 엔트리 및 홀의 개수에 비례하여 메모리 사용량이 증가하게 된다. XFLab은 추가적인 자료 구조의 유지 없이 연산자들의 타겟 조각 깊이 차이를 이용해서 후손 조각의 식별자를 조상 조각의 식별자로 변환할 수 있다. 본 논문에서 이용한 레이블링 기법은 구분자(separator) '.'를 이용하여 조각의 구조적 위치를 나타내고, 이를 이용하여 조상 조각과 후손 조각 간의 관계를 표현한다. 조각 식별자는 조각의 깊이를 n이라 하였을 때 분리자 개수가 n-1이므로 이를 이용하여 연산자가 평가하는 조각의 조각 식별자를 상위 연산자의 조각 깊이에 해당하는 조각의 식별자로 변환한다. 예를 들어 조각 식별자로 "1.2.1.3"을 가지는 조각 깊이 4의 조각이 있을 때, 조각 깊이 2인 연산자에 문의를 수행하기 위해서는 두 번째 구분자 전까지의 코드를 선택하여 조각 식별자를 "1.2"로 변환한다. 이와 같이 변환된 식별자를 이용하여 조상 조각의 평가 결과를 알아낸다.

3.4.2 평가(evaluate)

각 연산자는 조각을 평가할 때 앞 절에서 설명한 문의 과정을 통하여 해당 조각을 평가할 것인가에 대한 결정을 내린다. 그림 13의 알고리즘 3은 연산자에서 수행하는 조각의 평가 과정을 나타낸다. 문의 결과가 거짓이거나 미결정이고 최하위 연산자가 아니라면 해당 조각의 결과를 조상 조각의 결과와 동일하게 설정하고 연관 테이블 엔트리에 기록한다. 이것은 해당 조각을 루트로 한 트리의 후손들을 질의 처리 대상에서 제외(거짓)시키거나 보류(미결정)시키는 결과를 가져온다. 문의 결과가 참이라면 해당 조각에 대한 평가를 수행하게 되는데 이 과정에서 조건이 있는 경우와 없는 경우로 나누어진다. 조건이 있는 경우는 조건식의 평가 결과(참, 거짓, 미결정)를 반영하고, 그렇지 않은 경우는 결과를 참으로 설정한다. 평가를 수행하는 연산자가 최하위 연산자이고 평가 결과가 참이라면 해당 연산자와 동일한 태그 식별자를 갖는 엘리먼트를 결과로 출력한다. 최하위 연산자가 아니라면 현재 조각에 대한 평가 결과를 연관 테이블 엔트리에 기록하고 평가를 종료한다.

3.4.3 트리거(trigger)

조각 평가 시 어떤 조각들은 평가 결과가 미결정으로 분류되어 값이 결정되기를 기다리는 조각들이 있다. 예를 들어, 조상 조각이 아직 도착하지 않았거나 프레디킷 표현식에 해당하는 조각이 도착하지 않아서 평가를 보류한 조각들의 경우이다. 이런 조각들은 미결정 상태를 유발한 조각이 도착하여 평가가 이루어지면, 자신의 상태를 결정지을 수 있다. XFLab에서는 조각에 대한 평가를 수행한 후, 결과가 미결정 상태가 아니라면 하위 연산자에서 대기 중인 조각들에 대한 평가를 수행하도록 한다. 그림 14의 알고리즘 4는 이와 같은 과정을 나타낸다.

트리거 알고리즘의 핵심은 부모 조각의 식별자를 이용하여 자식 조각을 찾는 것이다. 기존의 XFrag에서는 연관 테이블 엔트리에 자식 조각에 대한 정보(홀 식별자)를 기록하고 있으므로, 홀 식별자와 동일한 필러 식별자를 갖는 조각에 대하여 평가를 수행하도록 하위 연산자에게 지시를 한다. 하지만 XFLab은 자식 조각에 대한 정보를 기록하지 않으므로 이런 방법을 적용할 수 없다. 대안으로 제시할 수 있는 간단한 방법은 하위 연산자의 연관 테이블에 저장된 엔트리들을 순회하며 조각 식별자를 비교하는 것이다. 하지만 이 방법은 연관 테이블에 저장된 엔트리의 개수에 비례하여 처리 시간이 증가하는 단점이 있어 전체적인 처리 효율성을 떨어뜨릴 수 있다.

```
알고리즘 3: evaluate
입력 : 평가 대상 조각의 식별자(fragment_ID)
       평가 대상 조각의 문의 결과(inquiry_result)
출력 : 평가 대상 조각의 평가 결과(TRUE, FALSE, UNDECIDED)
변수 : condition 조건 연산자
       association_table 연산자의 연관 테이블

   /* 문의 결과가 참일 때 */
1:  IF (inquiry_result == TRUE)
        /* 입력으로 받은 조각 식별자와 일치하는 연관테이블 항목을 가져옴 */
2:        table_entry = association_table.get( fragment_ID );
3:        IF (condition != null) THEN
4:                result = condition.predicate_result( fragment_ID ); /* 조건식 평가 결과를 반영 */
          ELSE
5:                result = TRUE;                    /* 조건이 없다면 TRUE */
          END IF
   ELSE  /* 문의 결과가 거짓 일 때 */
6:        result = inquiry_result;
   END IF

  /* 최하위 연산자인 경우 */
7:  IF (successor == null)
8:        IF (result==TRUE)
9:                display_result( table_entry.value );      /* 결과를 출력 */
          END IF
   ELSE  /* 최하위 연산자가 아닌 경우 */
10:       table_entry.value = result;              /* 평가 결과를 연관 테이블에 기록 */
11:       association_table.put( fragment_ID, table_entry );

   END IF
12: return result;
```

그림 13 알고리즘 평가(evaluate)

```
알고리즘 4: trigger
입력 : 평가 대상 조각의 식별자(fragment_ID)
       평가 결과(eval_result)
변수 : successor  하위 연산자
       association_table 연산자의 연관 테이블

  /* 상위 연산자와 하위 연산자가 다른 조각에 포함되어 있는 경우 */
1: IF (FTSID != sucessor.FTSID)

        /* treemap을 이용하여 부모 조각의 후손에 해당하는 조각 식별자 리스트를 얻어온다 */
2:      children = successor.association_table.getSubmap( fragment_ID );
3:      child = children.first();

        /* 해당 조각의 모든 후손 조각에 대해 트리거 */
4:      WHILE (child != null)
5:              successor.evaluate( eval_result, child.fragment_ID );
6:              child = child.next();
        END WHILE
  ELSE    /* 상위 연산자와 하위 연산자가 같은 조각에 포함되어 있는 경우 */
7:      successor.evaluate( eval_result, fragment_ID );
  END IF
```

그림 14 알고리즘 트리거(trigger)

본 논문에서는 자식(후손) 조각을 트리거하는 새로운 방법을 사용하였다. XFLab에서 각 조각의 식별자는 조상 조각의 식별자를 프리픽스로 갖는다. 이런 특징을 이용하여 연관 테이블 엔트리를 정렬된 트리 구조로 저장하

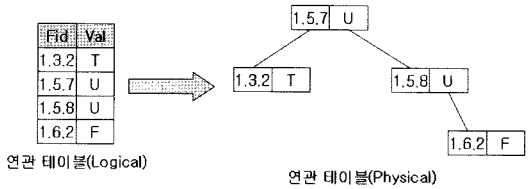


그림 15 연관 테이블의 논리적 구조와 물리적 구조

```
<tag stream="ex">
  <tag id="1" name="a" filler="true">
    <tag id="2" name="b" filler="true"/>
    <tag id="3" name="c" filler="true"/>
  </tag>
</tag>

질의 : /a[b=10]/c
```

면 자식(후손) 조각을 효율적으로 찾아낼 수 있다. 그림 15는 특정 연산자 내에 저장된 연관 테이블 구조의 예를 보인 것이다. 그림 15의 왼쪽은 연관 테이블을 논리적인 구조로 나타낸 것으로 조각 식별자를 기준으로 정렬된 상태를 나타내며, 오른쪽은 테이블 엔트리가 트리의 형태로 저장된 모습을 나타낸 것이다. 연관 테이블이 정렬된 상태로 저장되기 때문에 후손을 찾는 것은 조상의 식별자를 프리픽스로 갖는 엔트리들을 범위 기반으로 추출함으로써 찾을 수 있다. 찾아낸 연관 테이블 엔트리들에서 후손 조각의 식별자를 이용하여 하위 연산자에게 해당 조각에 대한 평가를 수행하게 한다.

3.5 질의 처리 예

본 절에서는 앞 절에서 설명한 알고리즘들을 기반으로 한 질의 처리 동작을 그림 16에서 예를 들어 설명한다. 예에서 사용된 XPath 질의 및 태그 구조는 다음과 같다. 태그 식별자가 1이고 조각 식별자가 1인 조각이 도착하면 "/a" 연산자는 조각에 대한 평가를 수행한다. 이 연산자는 해당 조각과 관련된 조건이 아직 평가되지 않았기 때문에 그림 16의 (a)와 같이 연관 테이블에 조각 평가 정보를 미결정으로 기록한다. 이 조각에 대한 정보는 연관 테이블에 기록하였으므로 이 시점에서 조각을 삭제한다. 태그 식별자가 3이고 조각 식별자가 1.2인 조각이 도착하면, "/c" 연산자는 이 조각에 대한 처리를 시작한다. 이 연산자는 최하위 연산자로 결과 출력을 담당한다. 하지만 문의 결과가 미결정 상태이므로, 이 연산자는 결과를 바로 출력하지 않고 그림 16의 (b)와 같이 연관 테이블에 결과를 기록해 둔 후에 상위 연산자에서 트리거할 때까지 결과 출력을 보류한다.

태그 식별자가 2이고 조각 식별자가 1.1인 조각이 도착하면 "/b" 연산자는 해당 조각에서 b 엘리먼트의 값을 추출하여 연관 테이블에 기록한 후 조건 연산자를 트리거한다. 이 조각은 조건식을 만족하기 때문에 조건

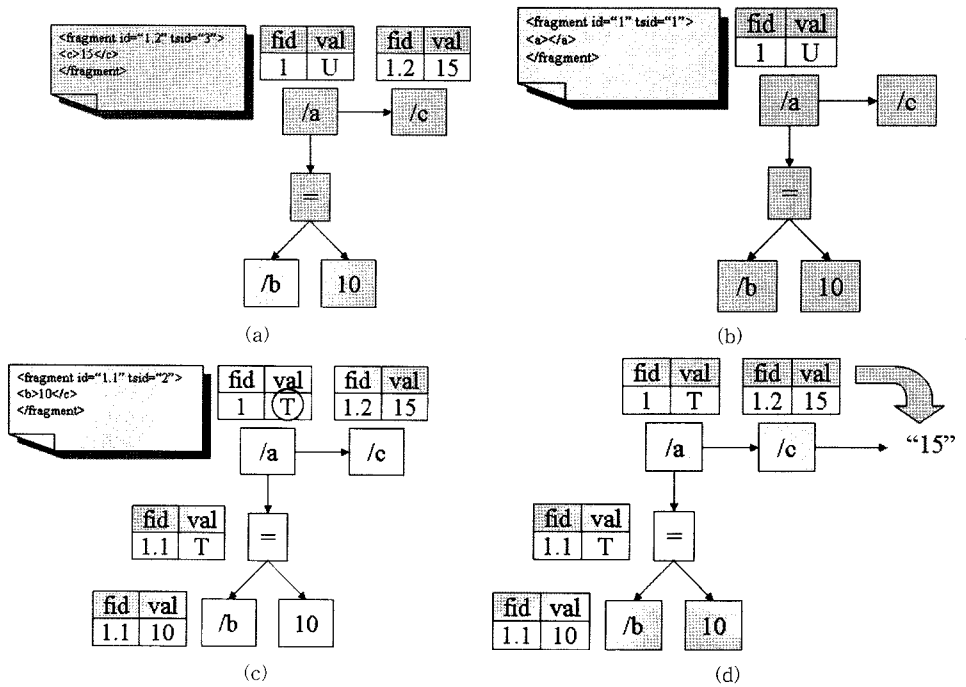


그림 16 XFLab 질의 처리 예

연산자는 그림 16의 (c)와 같이 연관 테이블에 평가 정보를 참으로 기록하고, "/a" 연산자를 트리거한다. "/a" 연산자는 조각 식별자가 1인 조각이 프레디킷에 명시된 조건을 만족하였으므로 이 조각에 대해 연관 테이블의 값을 참으로 갱신하고 "/c" 연산자를 트리거한다. "/c" 연산자는 조상 조각의 결과가 참으로 결정되었기 때문에 연관 테이블에서 조각 식별자가 1.2인 조각의 값 "15"를 출력한다.

4. 구현 및 성능 평가

4.1 개요

본 논문에서 제시한 XFLab 기법을 J2SE Development kit 5.0 update 6을 사용하여 Java로 구현하였다. 성능 실험은 Windows XP Professional 운영체제에서 2.4GHz 듀얼 코어 CPU를 사용하고 메모리는 2GB인 시스템에서 수행하였다. 성능 평가에 사용된 실험 데이타는 XMark benchmark[25]의 xmlgen 프로그램으로 생성한 세 가지 경매(auction) 문서로 크기는 각각 22.8 MB, 56.2MB, 113MB이다. 홀-필러 모델을 이용하여 위 문서를 분할하였을 때는 분할된 문서 크기가 27.2MB, 67.2MB, 135MB가 되었으며, XFLab의 경우는 분할된 문서 크기가 25MB, 61.9MB, 124MB가 되었다. 실제 응용과 비슷한 환경을 시뮬레이션하기 위해 XML 조각을 전송하는 서버를 구현하여 XML 조각을 스트리밍하였으며, 이동 단말기를 시뮬레이션하는 XML 조각 스트림 처리기에서 이를 받아 처리하였다. 본 실험에서는 질의 처리 시 문서의 크기 및 전송 순서에 따른 메모리 사용량 및 처리 시간을 측정하였으며 실험에 사용된 XPath 질의는 모두 10가지로 (Q1~Q10) 표 1에 열거하였다. 문서의 전송 순서로는 preorder, top-down, 그리고 bottom-up의 세가지를 고려하였는데 이는 각각 분할된 XML 조각 스트림을 전송하는 순서로서, 그림 17에 이들 세가지 전송 순서의 예를 나타내었다. preorder란, 조각 트리를 preorder 방식으로 탐색하면서 지나게 되는 노드의 순서로 전송하는 것이다. 그림 17의 예에서 preorder 방식의 전송은 1->2->3->4->5->6->7이 된다. top-down과 bottom-up 방식은 노드의 레벨(level)을 기준으로 전송하는 것으로 top-down의 경우는 1->2->5->6->3->4->7이 되고 bottom-up의 경우는 3->4->7->2->5->6->1이 된다.

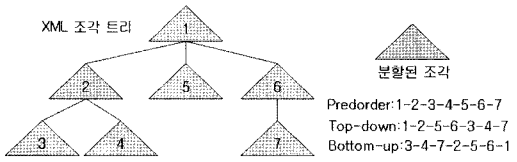


그림 17 XML 조각 전송 순서

측정된 메모리 사용량은 조각 스트림에 대한 질의 처리가 모두 완료될 때까지 클라이언트의 단말기가 필요로 했던 최대 메모리 용량을 의미한다. 즉, 최소한 그만큼의 메모리가 있어야 질의 처리가 가능했음을 의미한다. 질의 처리 시간은 서버와의 통신 시간을 제외한 클라이언트 단말기에서의 XML 조각 처리 시간만을 측정하였다.

4.2 실험 결과

그림 18은 본 논문에서 제시한 XFLab의 메모리 사용량을 XFrag의 그것과 비교 평가한 결과 그래프이다. 그래프의 x축은 표 1의 질의들을 나타내고, y축은 메모리 사용량을 MB 단위로 나타낸 것이다. 실험 결과를 보면 XFLab이 XFrag 에 비해서 10가지 질의 모두 메모리 사용량에서 우수한 성능을 보이고 있음을 알 수 있다. XFLab은 특히 조상-후손 축이 포함된 질의(Q2, Q4, Q6, Q9) 처리 시 뛰어난 성능을 보이는데 이는 3.4절에서 설명한 것과 같이 XPath 질의의 각 스텝이 연산자로 변환되므로 XFLab이 XFrag에 비해서 적은 수의 엔트리를 저장하는 연관 테이블을 유지하기 때문이다. 예를 들어 Q6의 질의 처리 시 XFLab에서는 "/site", "/open_auctions", "/open_auction", "/initial", "//increase"가 연산자로 생성되는 반면, 기존 기법에서는

표 1 실험에 사용된 질의

질의 번호	질의
Q1	/site/open_auctions/open_auction[initial>"10"]/interval/start
Q2	/site/open_auctions/open_auction[initial>"10"]//start
Q3	/site/open_auctions/open_auction/bidder/increase
Q4	/site//increase
Q5	/site/open_auctions/open_auction[initial>"10"]/bidder/increase
Q6	/site/open_auctions/open_auction[initial>"10"]//increase
Q7	/site/open_auctions/open_auction/bidder[increase>"200"]
Q8	/site/open_auctions/open_auction[initial>"0"]/bidder
Q9	/site/regions/africa/item[location="United States"]//from
Q10	/site/regions/africa/item[location="United States"]/mailbox/mail/from

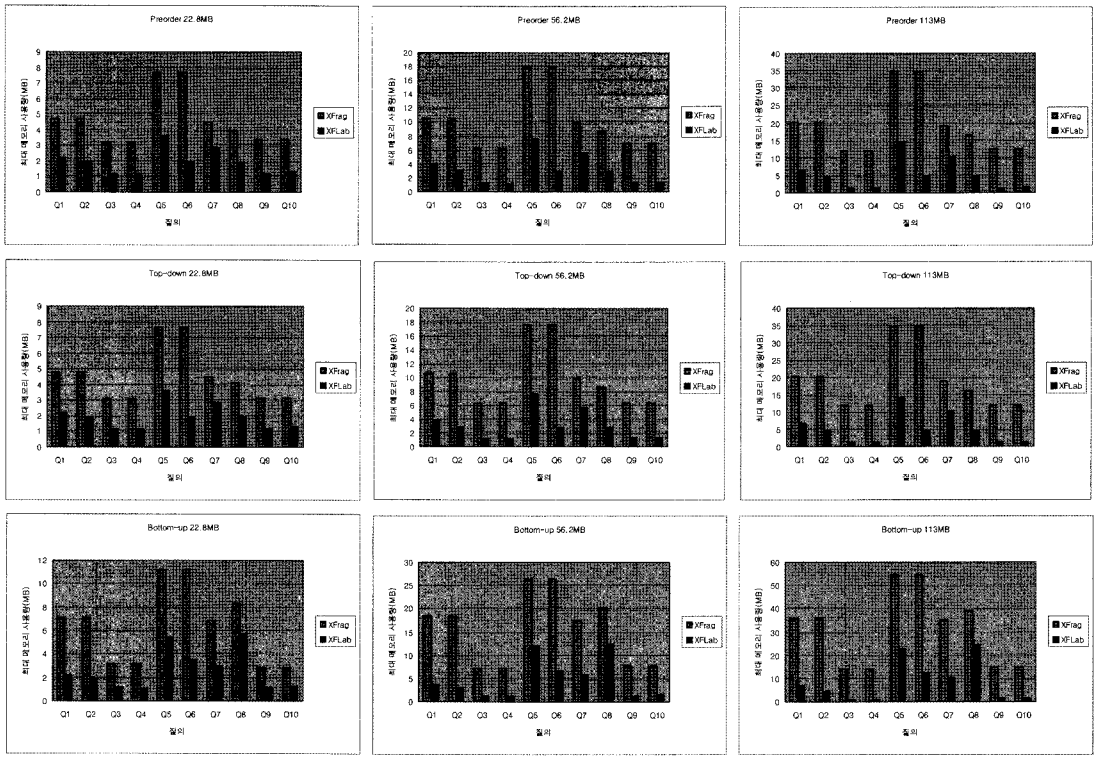


그림 18 문서 크기 및 전송 순서에 따른 메모리 사용량 측정 결과

XPath 질의의 조상-후손 축을 부모-자식 축으로 변환해야 하므로 변환된 질의는 Q5의 질의와 동일해지고 이를 연산자로 변환 시 "/bidder" 연산자를 추가로 생성해야 한다. 따라서 "/bidder" 연산자에서 유지해야 하는 연관 테이블 엔트리의 개수(113MB 기준 59,486개)만큼 메모리를 절약하게 된다.

XFLab은 문서의 크기 증가에 비례한 메모리 사용량 증가가 XFrag에 비해서 덜 민감한 특징을 갖는다. 홀-필러 모델을 사용하여 XML 문서를 분할하게 되면, 문서의 크기와 비례하여 홀의 개수도 많아지는 경향을 보이게 된다. 이는 XML 문서의 보편적인 구조[21]를 고려할 때 당연한 결과이다. XFrag는 질의 처리 시 연관 테이블 엔트리에 홀 정보를 기록해야 하므로 문서의 크기 증가에 비례하여 메모리 사용량이 급격히 증가하는 경향을 보인다. 예를 들어 Q5의 질의는 홀-필러 모델 적용 시 저장해야 하는 홀 정보의 개수가 22.8MB 옥션 문서에서는 12,097개, 56.2MB 옥션 문서에서는 29,629개, 113MB 옥션 문서는 59,486개이다. 또한 많은 홀을 포함한 조각은 그 자체로 큰 용량을 차지하기 때문에 질의 처리 시 더 많은 메모리를 사용하게 된다. 반면에 XFLab에서는 홀 정보를 저장하지 않으며, 분할된 XML 조각의 크기가 홀-필러 모델을 적용한 것에 비해 작으므로 문서의 크기 증가에 비례한 메모리 사용량 증가가 XFrag에 비해서 완만한 경향을 보인다. 특히 Q3, Q4, Q9, Q10 질의의 경우 XFLab은 문서 크기의 증가에 따른 메모리 사용량 증가가 거의 없는 것을 알 수 있다. 이는 Q3과 Q4 질의의 경우 질의에 프레디킷이 포함되지 않아서 질의 처리 시 연관 테이블에 정보를 저장하지 않기 때문이며, Q9와 Q10 질의의 경우 프레디킷은 있으나 연관 테이블에 저장되는 엔트리의 수가 적기 때문이다. 이런 상황은 XFrag에서도 동일하게 일어나지만 앞에서 언급했듯이, 홀-필러 모델을 적용한 XML 문서 분할은 분할된 XML 조각의 크기가 문서의 크기에 비례하여 증가하는 경향을 보이기 때문에 질의 처리 시 많은 메모리를 사용하게 된다.

XFLab은 문서의 전송 순서에 따른 메모리 사용량에서도 타 기법에 비해 우수한 성능을 보인다. 특히 XML 조각의 전송 순서가 bottom-up인 경우 메모리 사용량에서 현저한 차이를 보이게 되는데 이유는 다음과 같다. bottom-up 방식으로 XML 조각을 받아서 질의를 처리할 경우, 연산자 내의 연관 테이블은 미결정 상태의 엔트리들로 채워지게 된다. 미결정 상태인 엔트리들은 상위 연산자에서 결과가 결정될 때까지 연관 테이블 내에 유지되어야 하기 때문에 메모리 사용량 증가를 초래한

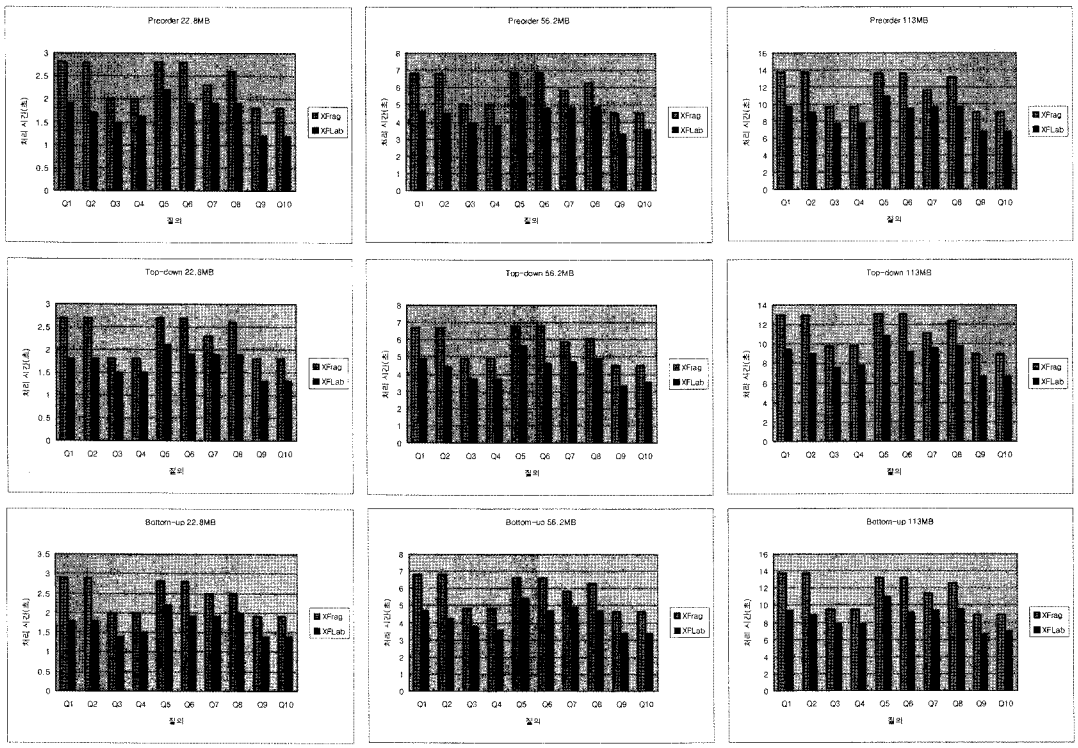


그림 19 문서 크기 및 전송 순서에 따른 실행 시간 측정 결과

다. 질의에 따라서는 미결정 상태인 엔트리들 없이 바로 처리가 되기도 하지만(Q3, Q4 질의) 다른 질의에서는 앞에서 언급한 과정이 진행된다. 이 과정은 XFrag에서도 동일하게 일어난다. 하지만 홀-필러 모델을 이용하여 옥션 문서를 분할하고 조각 트리를 구성했을 때, 트리의 상부에 해당하는 조각들은 대체로 많은 수의 홀을 포함하고 있어 크기가 커지게 된다. 이러한 조각을 질의 처리 과정의 후반부에서 처리해야 하는 XFrag는 많은 양의 메모리를 필요로 하게 된다. XFLab은 이러한 조각이 존재하지 않으므로 메모리 효율 면에서 우수한 성능을 보인다.

그림 19는 XFLab의 질의 처리 시간을 XFrag와 비교 평가한 결과 그래프이다. 그래프의 x축은 표 1의 질의들을 나타내고, y축은 처리 시간을 초단위로 나타낸다. XFLab은 XFrag에 비해서 10가지 질의 모두에서 빠른 처리 성능을 보이고 있다. XFLab은 질의 처리 과정에서 홀에 대한 처리를 할 필요가 없으며 처리해야 하는 조각 자체의 크기가 XFrag에 비해서 작기 때문에 처리 시간 면에서도 우수한 성능을 보이게 된다. 질의 처리 과정에서 문의, 트리거 등의 알고리즘 수행에 걸리는 시간보다 조각을 파싱(parsing)하는 데 걸리는 시간이 더 크기 때문에 조각의 크기는 전체적인 처리 성능에 큰 영향을 미치게 된다.

조상-후손 축(//)이 포함된 질의(Q2, Q4, Q6, Q9 질의)의 경우, XFLab은 질의 변환이 필요 없으므로 XFrag보다 적은 수의 연산자들을 가지고 질의 처리를 하게 된다. XFrag는 Q1-Q2, Q3-Q4, Q5-Q6, Q9-Q10의 각 쌍이 모두 동일한 성능을 보인다. 이에 비해 XFLab은 조상-후손 축이 포함된 질의 처리가 그렇지 않은 경우(Q1, Q3, Q5, Q10)에 비해서 처리 시간이 동일하거나 단축되는 것을 볼 수 있다.

5. 결 론

본 논문에서는 유비쿼터스 컴퓨팅 환경에서 XML 조각 스트림을 처리하는 데 있어 XML 레이블링 기법을 이용한 조각 관계 표현 및 이를 기반으로 하는 XFLab 기법을 제시하였다. 제시한 기법에서는 기존의 XFrag, XFPro 등의 기법이 사용하는 홀-필러 모델을 폐기하고 XML 레이블링 기법을 도입하여 XML 조각 간 관계를 표현하는 데 이용하였다. XML 레이블링 기법을 도입하여 홀-필러 모델에서 부가적인 정보였던 홀 식별자를 제거함으로써 질의 처리 시 메모리 및 처리 효율성을 큰 폭으로 개선할 수 있었다. 또한 조상-후손 축을 포함한 질의 처리 시에도 질의를 자식 축 기반의 질의로

변환하지 않고도 처리가 가능하여 처리 효율성 면에서도 이득이 된다. 뿐만 아니라 분할된 XML 조각의 크기가 홀-필러 모델의 경우에 비하여 작고, 부가 정보의 양이 적기 때문에 이동 단말기에서의 에너지 효율성에도 이득이 된다. 구현 및 실험을 통하여 본 논문에서 제시한 XFLab 기법이 기존의 XFrag 기법보다 메모리 및 처리 효율성 양면 모두에서 우수함을 확인하였다.

향후 연구 과제는 다음과 같다. 첫째, 서버의 XML 데이타가 클라이언트의 이동 단말기에서 질의 처리가 이루어지는 과정 중에도 계속 생성되면서 스트리밍되는 동적인 환경으로 본 논문의 XFLab을 확장하는 연구가 필요하다. 둘째, XML 데이타의 분할 과정에서 파생되는 정보를 이동 단말기의 질의 처리 과정에서 활용하여 메모리 및 처리 효율을 제고하기 위한 기법의 연구가 필요하다. 셋째, 기존의 XML 레이블링 기법을 확장하여 XML 조각 간 구조 관계 표현뿐만 아니라 이동 단말기에서의 질의 처리 및 메모리 관리에도 활용할 수 있는 부가 정보를 제공하게 하여 메모리 및 처리 효율을 제고하기 위한 연구가 필요하다.

참 고 문 헌

[1] "XML Fragment Interchange," W3C Candidate Recommendation 2001.

[2] S. Bose, L. Fegaras, D. Levine, and V. Chaluvadi, "A Query Algebra for Fragmented XML Stream Data," DBLP 2003.

[3] L. Fegaras, D. Levine, S. Bose, and V. Chaluvadi, "Query Processing of Streamed XML Data," CIKM 2002, pp. 126-133.

[4] S. Bose and L. Fegaras, "XFrag: A Query Processing Framework for Fragmented XML Data," Web and Databases 2005.

[5] H. Huo, G. Wang, X. Hui, R. Zhou, B. Ning, and C. Xiao, "Effiecient Query Processing for Streamed XML Fragments," Proc. Int'l Conf. on DASFAA, 2006.

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

[7] C. Chan et al., "Efficient Filtering of XML Documents with XPath Expressions," Proc. Int'l Conf. on Data Eng., 2002.

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

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

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

[11] J. Kwon, P. Rao, B. Moon, and S. Lee, "FiST: Scalable XML Document Filtering by Sequencing Twig Patterns," Proc. of Int'l Conf. on VLDB, 2005, pp. 217-228.

[12] K. Candan, W. Hsiung, S. Chen, J. Tatemura, and D. Agrawal, "Afilter: Adaptable XML Filtering with Prefix-Caching and Suffix-Clustering," Proc. of Int'l Conf. on VLDB, 2006, pp. 559-570.

[13] Z. Ives, A. Levy, and D. Weld, "Efficient Evaluation of Regular Path Expressions on Streaming XML Data," Tech. Rep. UW-CSE-2000-05-02, U. of Washington, 2000.

[14] B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou, "A Transducer-Based XML Query Processor," Proc. Int'l Conf. on VLDB, 2002.

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

[16] D. Florescu, C. Hillery, D. Kossmannn, P. Lucas, F. Riccardi, T. Westmann, M. Carey, A. Sundararajan, and G. Agrawal, "The BEA/XQRL Streaming XQuery Processor," Proc. Int'l Conf. on VLDB, 2003.

[17] L. Fegaras, "The Joy of SAX," Proc. Int'l Workshop on XQuery Implementation, Experience, and Perspectives, 2004.

[18] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier, "FluXQuery: An Optimizing XQuery Processor for Streaming XML Data," Proc. Int'l Conf. on VLDB, 2004.

[19] E. Wong, A. Chan, and H. Leong, "Efficient Management of XML Contents over Wireless Environment by Xstream," Proc. ACM Symp. on Applied Computing, 2004, pp 1122-1127.

[20] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda, "Lazy Evaluation for Active XML," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2004.

[21] L. Mignet, D. Barbosa, and P. Veltri, "The XML Web: a First Study," WWW 2003.

[22] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: Insert-Friendly XML Node Labels," SIGMOD 2004.

[23] C. Li and T. Ling, "QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates," CIKM 2005.

[24] I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and Querying Ordered XML Using a Relational Databse System," SIGMOD 2002.

[25] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse, "XMark: A Benchmark for XML Data Management," Proc. Int'l Conf. on VLDB, 2002, pp. 974-985.

이 상 욱

2006년 2월 중앙대학교 컴퓨터공학과 졸업(학사). 2006년 3월~현재 중앙대학교 대학원 컴퓨터공학과 재학 중. 관심분야는 XML 데이타베이스, XML 스트림 데이타 처리 등

김 진

2006년 8월 중앙대학교 컴퓨터공학과 졸업(학사). 2006년 9월~현재 중앙대학교 대학원 컴퓨터공학과 재학 중. 관심분야는 XML 스트림 데이타 처리, 웹 데이타베이스 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, Stream 데이타 관리, 센서네트워크 데이타베이스 등