XML 레이블링 기법을 이용한 XML 조각 스트림에 대한 질의 처리*

이상욱°, 김진, 강현철
중앙대학교 컴퓨터공학부
{swlee, jkim}@dblab.cse.cau.ac.kr, hckang@cau.ac.kr

Query Processing over XML Fragment Stream Using an XML Labelling Scheme

Sangwook Lee°, Jin Kim, Hyunchul Kang
School of Computer Science and Engineering
Chung-Ang University

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 데이터에 대한 질의 처리 기술에 관한 것이다. 메모리 용량이 크지 않은 단말기의 경우 대량의 XML 데이터에 대한 질의 처리를 수행하기 위해서는 XML 스트림 질의 처리 기술이 활용되어야 한다. 최근에 제시된 XFrag 기법은 홀-필러(hole-filler) 모델을 이용하여 XML 데이터를 XML 조각(fragment)으로 분할(fragmentation)하여 스트림으로 전송하고 처리할 수 있는 기법이다. 이는 메모리 효율성이 요구되는 이동 단말기에서 전체 XML 문서를 재구성하지 않고 XML 데이터에 대한 질의 처리를 가능하게 한다. 그러나 홀-필러 모델을 사용할 경우 홀과 필러에 대한 부가적인 정보를 저장해야 하므로 메모리 효율성이 높지 못하다. 본 논문에서는 XML 데이터의 구조 정보를 표현하는 XML 레이블링(labelling) 기법을 이용하여 XML 데이터를 조각으로 분할하여 조각 스트림에 대한 질의 처리를 수행하는 기법을 제시한다. 구현 및 성능 실험 결과 본 논문에서 제시한 기법이 기존의 XFrag 기법보다 메모리 사용량과 처리 시간 양면 모두에서 우수한 것으로 나타났다.

1. 서 론

XML이 웹에서 데이터 교환의 표준으로 부각된 이래, 유비쿼터스 컴퓨팅, 센서 네트워크, 모니터링 등의 응용 분야에서 XML 스트림 데이터에 대한 효율적인 질의 처리에 관한 연구가 활발히 진행되고 있다. 특히 유비쿼터스 컴퓨팅 환경의 경우, 다수의 사용자들이 PDA나 휴대폰 등의 이동 단말기를 이용하여 정보를 제공 받기 때문에, 기존의 요구-응답 방식으로 질의 처리 수행 시 서버 측에 많은 부담을 주게 되어 처리 효율성이 저하된다. 따라서 이와 같은 환경에서는 서버는 데이터를 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 데이터에 대한 질의 처리 기술에 관한 것이다. XML 데이터는 계층적 구조를 가지고 있으며 용량이 클 수 있다. 따라서 이동 단말기의 적은 메모리로 대량의 XML 데이터를 처리하려면, XML 데이터를 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 데이터를 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 데이터를 재전송하지 않고 변화된 조각만 전송함으로써 송수신 비용에서 이득을 얻을 수 있다.

최근에 제시된 홀-필러(hole-filler) 모델[2,3]은 XML 데이터를 구조적으로 분할하는 기법을 제시하고 있다. XFrag[4]나 XFPro[5]는 홀-필러 모델을 사용한 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 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 데이터의 구조 정보를 표현하는 XML 레이블링(labelling) 기법을 이용하여 XML 데이터를 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 처리하는데 소요되는 메모리 양 및 시간을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 살

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

```
<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

펴본다. 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

전체 XML 문서를 조각으로 분할하여 전송하고 이러한 조각에 대해 질의 처리를 수행하는 기법으로 XFrag[4]가 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 엘리먼트를 추가하고 홀 식별자(hole id)를 부여한다. 분할되는 문서는 필러 엘리먼트로 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 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 연산자 파이프라인을 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 깊이가 깊지 않고, 옆으로 넓게 퍼진 모양을 하고 있다[6]. 이러한 문서를 홀-필러 모델에 의해 분할하게 되면 매우 많은 홀 식별자를 포함하게 되는데 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 축을 자식 축으로 구성된 경로로 변환하지 않으면 질의 처리가 불가능하다. 그림 4는 이 문제의 예를 보이고 있다. 이러한 처리 방식은 문서 내에 조상-후손 축으로의 질의에 일치하는 엘리먼트가 다수 존재할 때 성능 저하의 요인이 될 수 있다.

XFPro[5]의 경우는 홀-필러 모델의 태그 구조를 이용하여, 질의 처리 계획을 생성하고 최적화를 수행한다. 최적화를 통해 생성된 파이프라인은 기본적으로 XFrag에서 사용하는 파이프라인과 동일하지만, XFrag에 비해 XML 조각을 처리하는 단계가 줄어들게 되어 속도 면에서 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

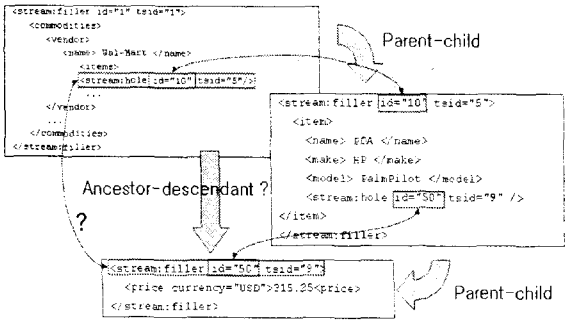

그림 4 홀-필러 모델의 문제점

관계를 표현하였다. 홀-필러 모델이 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 있다.

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 ORDPATH[7], QED[8] 등의 가장 진보된 기법들이 제안되었다. 본 절에서는 기본적인 프리픽스 레이블링 기법을 이용하여 조각의 구조 관계를 표현하고 이러한 관계 표현을 이용하는 조각 스트림 질의 처리 기법 XFLab을 제안한다. XML 질의로는 XPath 질의를 고려하였고 XML 레이블링 기법으로는 *Dewey order* encoding[9]을 사용하였다.

3.1 XML 문서 분할 요구 사항

본 논문에서 제시한 XML 레이블링 기법을 이용한 조각 질의 처리 기법에서는 XML 문서를 조각으로 분할할 때 DTD를 참조하여 "*", "+" 연산자가 정의된 엘리먼트는 별도의 조각으로 분할한다. 예를 들어, 그림 5의 DTD를 준수하는 문서의 경우 vendor, book, author 엘리먼트는 각각 개별의 조각으로 분할된다.

그림 5 DTD 예

```
<!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)>
```

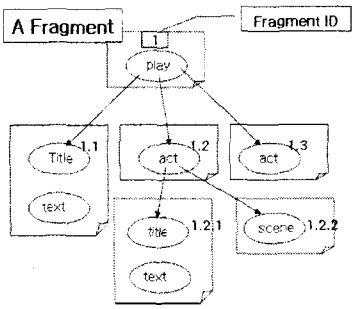

그림 6 Dewey order encoding을 사용하여 조각 식별자를 부여한 예

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

XML 문서는 트리로 나타낼 수 있는데, 조각으로 분할된 XML 문서 또한 트리로 표현할 수 있다. 각 조각은 자신이 분할되어 나온 조각을 부모 조각으로 가지게 되며 이때 분할된 조각은 자식 조각이 된다. 이렇게 부모-자식 간의 관계를 구성하면서 결과적으로 태그 구조에 의해 분할된 조각 간의 관계는 일반적인 트리의 모양이 된다.

이렇게 분할된 조각은 전체 조각 트리에서 자신의 고유한 위치를 식별하는 식별자를 XML 레이블링 기법에 의해 할당받는다. 그림 6은 Dewey order encoding을 사용하여 XML 조각에 조각 식별자를 할당한 예이다. Dewey order encoding은 루트 노드로부터 현재 노드까지의 경로에 관한 모든 정보를 저장함으로써 전체 트리에서 특정 노드의 위치를 식별하고 또한 부모-자식 관계와 조상-후손 관계를 파악할 수 있게 한다. 그림 7은 조각 식별자로 부모-자식 관계 및 조상-후손 관계를 비교하는 것을 예시한 것이다.

3.3 XFLab 파이프라인

본 절에서는 3.2절에서 기술한 XML 조각 간 관계 표현을 바탕으로 연산자 파이프라인의 동작을 설명한다. 본 논문에서 제시하는 이러한 새로운 기법을 XFLab이라고 명명하였다. XFLab 연산자 파이프라인은 XFrag를 기반으로 하되 트리거(trigger)와 문의(inquiry) 및 각종 연관 테이블(association table) 엔트리(entry) 매칭을 수행하는 부분을 수정하여 작성되었다. 따라서 파이프라인을 구성하는 연산자의 기본 동작은 XFrag와 동일하다. 기존의 홀-필러 모델에서는 부모-자식 관계만을 파악할 수 있었으므로 XPath 질의의 조상-후손 축을 자식 축으로 구성된 경로로 변환해야 하였으나 XFLab은 조상-후손의 관계를 직접 확인할 수 있으므로 그러한 질의 변환이 불필요하다. 따라서 질의 변환 없이 연산자를 구성하고 파이프라인을 생성할 수 있다.

XFrag 파이프라인에서 문의와 트리거는 파이프라인이 질의를 처리하는 데 있어 핵심적인 동작이다. 연산자들은 연관 테이블에 저장하고 있던 특정 엔트리의 값을 계산하기 전에 그것이 필요한 동작인지 부모 연산자에게 문의한다. 이때 엔트

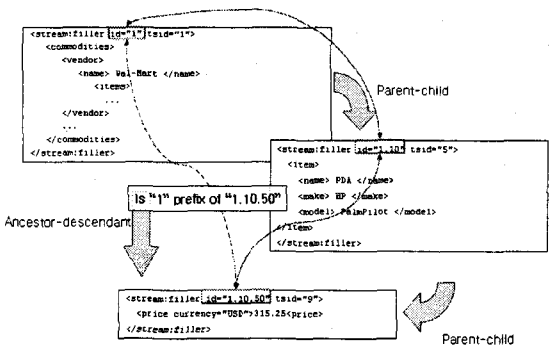

그림 7 XML 레이블링 기법 기반의 XML 문서 분할에서 조각 간 관계 비교

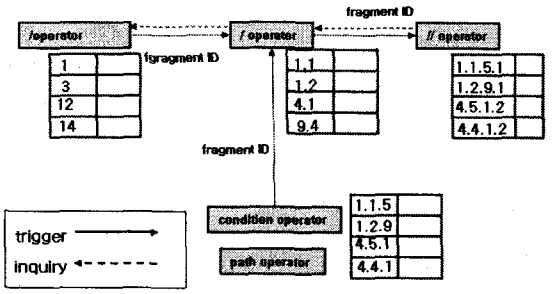

그림 8 연산자와 연관 테이블의 상태 예

리의 부모 조각 정보를 기록한 엔트리를 상위 연산자의 연관 테이블에서 찾아내기 위해 조각 식별자를 사용한다. 마찬가지로 트리거의 경우 부모 연산자가 트리거하려는 엔트리를 식별하기 위해 조각 식별자를 사용한다. 그림 8은 연산자들과 연산자 별로 할당된 연관 테이블에 저장되어 있는 조각 식별자의 예를 나타낸 것이다.

4. 구현 및 성능 평가

4.1. 개요

본 논문에서 제시한 XFLab 기법을 J2SE Development kit 5.0 update 6을 사용하는 JAVA 환경에서 구현하였다. 성능 실험은 *Windows XP Professional* 운영체제에서 *2.6GHz* CPU를 사용하고 메모리는 1GB인 시스템에서 수행하였다. 성능 평가에 사용된 실험 데이터는 XMark benchmark[10]의 xmlgen 프로그램으로 생성한 22.8MB 크기의 auction 문서이다. XFrag 기법을 이용하여 위 문서를 분할 시 분할된 문서 크기가 30.4MB가 되었으며, 본 논문에서 제시한 XFLab의 경우는 분할된 문서 크기가 27.0MB가 되었다. 실제 응용과 비슷한 환경을 시뮬레이션하기 위해 XML 조각을 전송하는 서버를 구현하여 순차적으로 XML 조각을 전송하였으며, 이동 단말기를 시뮬레이션하는 XML 조각 스트림 처리기에서 이를 받아 처리하였다. 실험에 사용된 질의는 [4]에서 사용된 아래 질의와 같다.

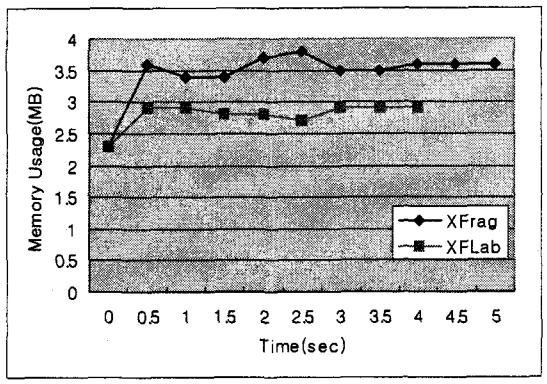

그림 9 질의 1 실험 결과

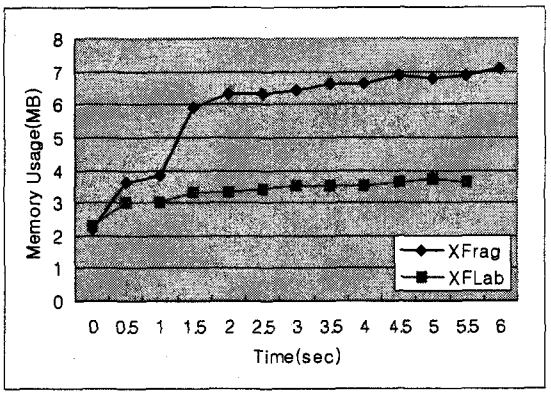

그림 10 질의 2 실험 결과

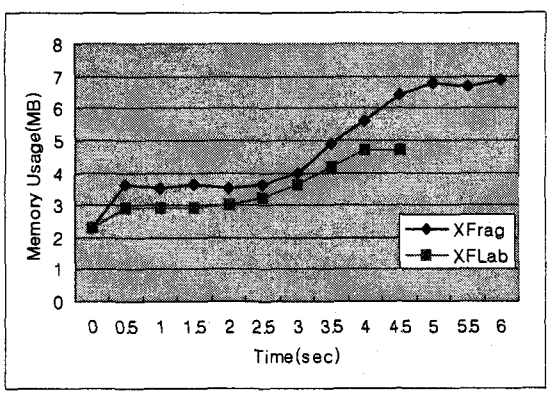

그림 11 질의 3 실험 결과

질의 1: doc("auction.xml")/site/open_auctions//increase

질의 2: doc("auction.xml")/site/open_auctions/open_auction[initial>"10"]/bidder

질의 3: doc("auction.xml")/site/open_auctions/open_auction/bidder[increase>"200"]

이동 단말기의 메모리 사용량은 EclipseProfiler[11] plugin를 사용하여 측정하였으며, 처리 시간은 서버와의 통신 시간을 제외한 XML 조각 처리 시간만을 측정하였다. 실험 결과는 각 질의에 대하여 30번 처리를 수행한 평균값을 나타내었다.

4.2. 초기 실험 결과

그림 9-그림 11은 XFrag와 XFLab으로 동일한 질의를 수행한 결과 그래프이다. 그래프의 x축은 처리 시간을 초 단위로, y축은 시간대별 메모리 사용량을 MB 단위로 나타낸 것이다. 실험 결과를 보면 XFLab이 XFrag에 비해서 세 질의 모두 메모리 사용량 및 속도 면에서 우수한 성능을 보이고 있음을 알 수 있다. 질의 1은 프레디킷(predicate)이 없어서 XML 조각 간의 관계를 유지할 필요가 없다. 따라서 두 기법 모두 XML 조각이 도착하는 즉시 결과를 도출하고 해당 XML 조각을 버린다. 질의 처리 동안 메모리 사용량의 최고값을 비교하였을 때, XFLab이 XFrag에 비해 800KB 정도의 메모리를 절약 (XFrag 대비 약 24% 절약)하고 있는 이유는 XML 레이블링 기법을 사용하여 XML 데이터를 분할한 경우 XML 조각의 크기가 홀-필러 모델을 이용하여 분할한 경우보다 작기 때문에 질의 처리를 위해 이동 단말기에서 사용되는 공간이 작기 때문이다. 처리 시간은 1초 정도 빠르다 (XFrag 대비 약 20% 단축).

질의 2의 경우 XFLab의 메모리 효율성이 훨씬 좋은 특성을 보인다. 이 질의는, open_auction 엘리먼트에 프레디킷이 있으므로 open_auction 엘리먼트가 포함된 조각과 bidder 엘리먼트가 포함된 조각 간의 구조 관계를 유지해야만 한다. XFLab은 XFrag에 비해서 적은 정보만으로 조각 간의 관계를 식별할 수 있으므로, 이 질의의 경우 약 3.5MB 정도의 메모리를 절약 (XFrag 대비 약 48% 절약)하고 있으며 처리 시간은 0.5초 정도 빠르다 (XFrag 대비 약 8% 단축).

질의 3은 bidder 엘리먼트에 프레디킷이 있으므로, bidder가 포함된 조각의 정보를 유지하고 있어야 한다. 따라서 초기에는 메모리 사용량이 일정한 편이었다가 bidder가 포함된 조각이 도착하면서 메모리 사용량이 늘어나게 된다. XFLab이 XFrag에 비해 2.5MB 정도의 메모리를 절약 (XFrag 대비 약 32% 절약)하고 있으며, 처리 시간은 1.5초 정도 빠르다 (XFrag 대비 약 25% 단축).

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 기법이 기존의 XFrag 기법 보다 메모리 및 처리 효율성 양면 모두에서 우수함을 확인하였다.

향후 연구 과제는 다음과 같다. XML 데이터 분할 과정에서 파생되는 정보를 이동 단말기(클라이언트)의 질의 처리 과정에서 활용하여 메모리 및 처리 효율을 제고하기 위한 기법의 연구가 필요하다. 또한 XML 레이블링 기법을 확장하여 XML 조각 간 구조 관계 표현뿐만 아니라 이동 단말기의 질의 처리에 활용할 수 있는 부가 정보도 제공하게 하여 이동 단말기의 처리 효율을 제고하기 위한 연구가 필요하다.

참 고 문 헌

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

[2] Sujoe Bose, Leonidas Fegaras, David Levine and Vamsi Chaluvadi, "A Query Algebra for Fragmented XML Stream Data," DBLP 2003.

[3] Leonidas Fegaras, David Levine, Sujoe bose and Vamsi Chaluvadi, "Query Processing of Streamed XML Data," CIKM 2002, pp. 126-133.

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

[5] Huan Huo, Guoren Wang, Xiaoyun Hui, Rui Zhou, Bo Ning, and Chuan Xiao, "Effiecient Query Processing for Streamed XML Fragments," DASFAA 2006.

[6] Laurent Mignet, Denilson Barbosa, Pierangelo Veltri, "The XML Web: a First Study," WWW 2003.

[7] Patric O'Neil, Elizabeth O'Neil, Shankar Pal, Istvan Cseri, Gideon Schaller, Nigel Westbury, "ORDPATHs: Insert-Friendly XML Node Labels," SIGMOD 2004.

[8] Changqing Li, Tok Wang Ling, "QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates," CIKM 2005.

[9] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, Chun Zhang, "Storing and Querying Ordered XML Using a Relational Databse System," SIGMOD 2002.

[10] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu and Ralph Busse. "XMark: A Benchmark for XML Data Management," VLDB 2002, pp. 974-985.

[11] http://eclipsecolorer.sourceforge.net/index_profiler.html