DOI: 10.3745/KIPSTD.2009.16-D.1.27

자원 효율적인 XML 조각 스트림 질의 처리를 위한 XML 분할

김 진[†] · 강 현 철[††]

요 약

유비쿼터스 컴퓨팅의 실현을 위해서는 이동 디바이스 등 클라이언트의 제약된 자원을 효율적으로 사용하는 기법이 요구된다. 메모리 용량이 크지 않은 이동 디바이스의 경우, 대용량 XML 데이터에 대한 질의 처리를 수행하기 위해서는 XML 스트림 질의 처리 기술의 활용이 필수적이다. 최근에 서버에서 XML 문서를 XML 조각(XML fragment)으로 분할하여 스트리밍하고 클라이언트에서 이 조각 스트림을 받아 질의를 처리하는 기법들이 제안되었다. XML 조각 스트림 질의 처리에 있어 XML 문서가 분할되는 방법에 따라 자원 사용(질의 처리 시간 및 메모리 사용량) 면에서 큰 차이가 날 수 있기 때문에 효율적인 XML 문서 분할 방법이 요구된다. 본 논문에서는 클라이언트의 질의 처리 시 자원 사용 효율을 높이기 위한 XML 문서 분할 기법을 제시한다. 이를 위하여 먼저 XML 조각 스트림 질의 처리의 비용 모델을 제시하고, 자원 효율적인 XML 문서 분할 알고리즘을 제시한다. 구현 및 성능 평가 결과 본 논문에서 제시한 기법이 기존 기법들에 비해 질의 처리 시간 및 메모리 사용량 양면 모두에서 우수한 것으로 나타났다. 본 논문의 기여는 XML 조각 스트림 질의 처리 기술의 실용화 가능성을 기존 기술에 비해 한 층 더 높였다는 데 있다.

키워드 : XML, XML 조각, 스트림 질의 처리, XML 분할

XML Fragmentation for Resource-Efficient Query Processing over XML Fragment Stream

Jin Kim[†] · Hyunchul Kang[††]

ABSTRACT

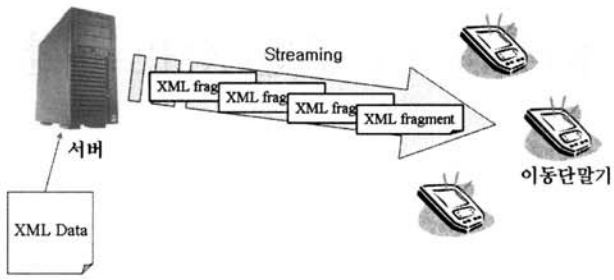
In realizing ubiquitous computing, techniques of efficiently using the limited resource at client such as mobile devices are required. With a mobile device with limited amount of memory, the techniques of XML stream query processing should be employed to process queries over a large volume of XML data. Recently, several techniques were proposed which fragment XML documents into XML fragments and stream them for query processing at client. During query processing, there could be great difference in resource usage (query processing time and memory usage) depending on how the source XML documents are fragmented. As such, an efficient fragmentation technique is needed. In this paper, we propose an XML fragmentation technique whereby resource efficiency in query processing at client could be enhanced. For this, we first present a cost model of query processing over XML fragment stream. Then, we propose an algorithm for resource-efficient XML fragmentation. Through implementation and experiments, we showed that our fragmentation technique outperformed previous techniques both in processing time and memory usage. The contribution of this paper is to have made the techniques of query processing over XML fragment stream more feasible for practical use.

Keywords : XML, XML Fragment, Stream Query Processing, XML Fragmentation

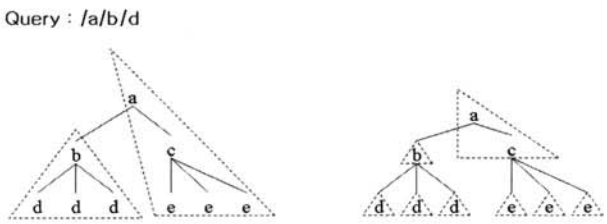
1. 서 론

XML이 웹에서 데이터 교환의 표준으로 부각된 이래 유비쿼터스 컴퓨팅 분야에서도 효율적인 XML 질의 처리에 관한 연구가 활발히 이루어지고 있다. 최근 핸드폰, PDA, 스마트폰 등을 비롯하여 저장 및 처리 능력을 갖춘 이동 디바이스의 보급이 확산됨에 따라 자원이 제약되어 있는 이들 클라이언트에서 효율적인 XML 질의 처리를 수행할 수 있는 기술에 대한 관심이 높아지고 있다. 대용량 XML 데이터는 크기가 클라이언트의 가용 메모리보다 클 수 있으므로, 이들 클라이언트에서 대용량의 XML 데이터를 처리하는 것은 불가능할 수도 있다. 따라서 XML 데이터를 적절한 크기의 XML 조각(XML fragment) [1]으로 분할하여 XML 조각 단위로 스트리밍하고 처리하는 기술이 요구된다. [2]에서는 서버 측에서 XML 데이터를 조각으로 분할하여 방송 기술

※ 본 논문은 한국과학재단 특정기초연구사업(R01-2006-000-10609-0) 지원으로 수행되었음.
† 준 회 원 : 중앙대학교 컴퓨터공학과 공학석사
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
논문접수 : 2008년 9월 8일
수 정 일 : 1차 2008년 12월 23일
심사완료 : 2009년 1월 11일



(그림 1) XML 조각 스트림 질의 처리

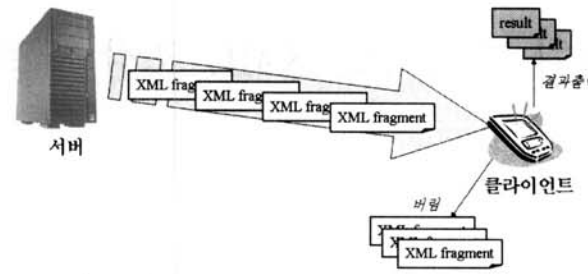


(그림 2) 하나의 XML 문서에 대해 서로 다른 분할을 적용한 예

등을 이용해 스트리밍하고 이동 디바이스와 같은 클라이언트에서 이 조각 스트림을 받아 질의 처리를 수행하는 개념이 제시되었다(그림 1). [3-6]에서는 이를 기반으로 XML 조각 스트림 질의 처리 기법을 제시하였다. [3]에서는 홀-필러(hole-filler) 모델로 조각 간의 관계를 표현하여 조각 스트림 질의 처리를 수행하는 XFrag라는 기법을 제시하였고, [4]에서는 XFrag의 질의 처리 시간을 개선한 XFPro를 제시하였다. [5]에서는 XFrag와 XFPro가 사용하는 홀-필러 모델을 XML 조각 레이블링 기법을 도입하여 대체함으로써 메모리 효율을 높이는 기법을 제시하였고, [6]에서는 [5]의 기법을 동적 조각 스트림에 적용하여 스트림의 크기 증가에도 메모리 효율성을 견지할 수 있는 확장성(scalability)을 제공하는 기법을 제시하였다. 하지만 이들 기법들에서는 클라이언트에서의 XML 조각 스트림에 대한 질의 처리 기술에 대해서만 다루었고, 클라이언트의 자원 효율적인 질의 처리를 지원하기 위한 XML 문서 분할 방법에 대해서는 다루지 않았다.

데이터 분할을 통해 질의 처리 효율성을 제고하는 것은 기존의 분산 관계 데이터베이스에서 널리 사용되어 온 기법으로, select 프리디킷에 의한 수평 분할(horizontal fragmentation), project 연산에 의한 수직 분할(vertical fragmentation), 이들을 혼용하는 혼합 분할(hybrid fragmentation) 등과 같은 기법이 있다. 최근에는 이러한 분할 기법들을 XML에 적용하여 분산 질의 처리의 효율성을 높이고자 하는 연구가 수행되고 있다[7-11]. XML 문서 분할에서는, XML 문서를 구성하는 요소들을 여러개의 XML 조각으로 분할할 수 있는데, XML 문서의 어느 부분들을 같은 조각으로 할당할 것인가에 따라 무수히 많은 분할이 존재할 수 있다. 본 논문은 XML 문서 분할 시 가능한 다양한 분할 중에서 클라이언트에서의 질의 처리 시 효율적인 자원 사용이 가능하게 하는 분할을 생성하는 기법에 관한 것이다.

(그림 2)는 하나의 XML 문서에 대해서 나타날 수 있는 서로 다른 두 개 분할의 예를 나타낸 것이다. 점선으로 표시된



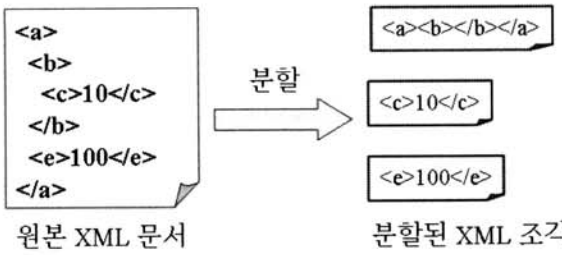
(그림 3) XML 조각 스트림에 대한 질의 처리 모델

부분이 XML 조각이다. 사용자의 질의가 XPath로 "/a/b/d"라고 가정할 때, (그림 2)의 두 분할 중 어떤 것이 이 질의를 [3-6] 등에서 제시된 기법으로 처리할 때 질의 처리 시간이 빠르고 메모리 사용량이 적은지는 직관적으로 판단하기 어렵다. 더욱이 사용자의 질의가 복수 개일 경우에는 이들 질의 간에 자원 사용면에서 상충이 존재할 수 있으므로 하나의 분할 결과가 모든 질의에 대해 최적의 분할일 수 없다. 따라서 각각의 질의가 아닌 전체 질의 집합에 대해 효율적인 분할을 생성하는 기법이 필요하다. 본 논문에서는 이를 위해 먼저, XML 조각 스트림에 대한 질의 처리 비용 모델을 제시하고, 클라이언트의 자원 효율적인 질의 처리를 위한 XML 문서 분할 기법을 제시한다.

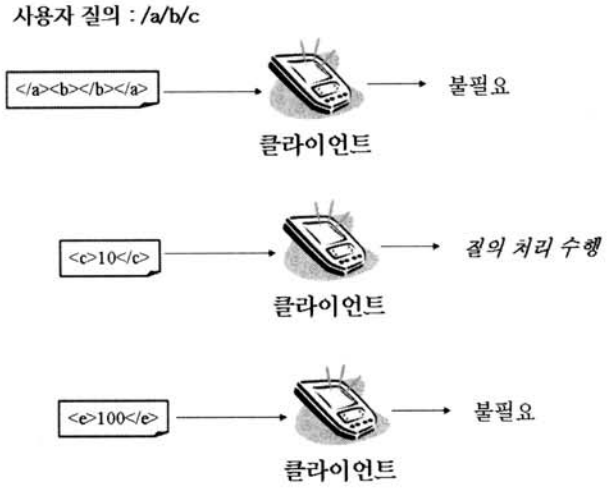
본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 살펴본다. 3절에서는 상기 질의 처리 비용 모델 및 XML 문서 분할 알고리즘을 기술한다. 4절에서는 3절에서 제시한 XML 분할 기법의 구현 및 성능 평가 결과를 기술한다. 5절에서는 결론을 맺고 향후 연구 내용을 기술한다.

2. 관련 연구

XML 조각 스트림에 대한 질의 처리 기법으로는 [3-6] 등이 있다. 이 기법들은 메모리가 제약된 클라이언트에서 대용량의 XML 문서에 대해 질의 처리를 수행하기 위한 것으로, 서버에서 XML 문서를 조각으로 분할하고 스트리밍하면 클라이언트에서는 이 조각 스트림으로부터 질의 결과를 도출한다(그림 3). 클라이언트는 메모리 사용량을 줄이기 위해 서버로부터 받은 XML 조각을 분석(질의 경로 탐색 등)하여 질의 처리에 필요한 정보만 추출하여 저장하고 해당 조각은 버린다(즉, 메모리에서 삭제한다). 즉, 클라이언트는 전체 XML 조각 스트림을 받아 이를 원래의 XML 문서로 재구성하여 질의 처리하는 것이 아니라 조각 단위의 질의 처리를 수행하기 때문에 메모리 사용량을 줄일 수 있다. 또한 XML 질의 처리는 일반적으로 XML 문서의 특정 부분에 대한 정보를 추출하는 것이므로 XML 문서를 조각으로 분할하여 전송하게 되면 질의 처리 시 사용자 질의와 연관된 조각에 대해서만 처리하고 그렇지 않은 부분은 버릴 수 있으므로 메모리 사용량이 줄어든다. 이와 같은 스트림 처리를 위해서는 수신된 조각 간의 관계를 비교하고 다른 조각의 처리 결과도 참조해야 한다. 특히, 서버가 XML 조각을 전송하는 순서와 클라이언트가 수신하는 순서가 다를 수 있



(그림 4) 원본 XML 문서와 이 문서를 조각으로 분할한 예



(그림 5) 이동 디바이스에서 질의 처리에 필요한 조각에 대해서만 처리하는 예

는 점도 고려되어야 한다. 따라서 XML 조각 스트림에 대한 질의 처리의 한 필수 요소로 조각 간의 관계를 명시할 수 있는 기법이 필요하다. [2]에서는 홀-필러 모델을 사용한 조각 간 관계 표현 기법을, [5]에서는 XML 레이블링 기법[12-14]을 응용한 조각 간 관계 표현 기법을 제시하였다.

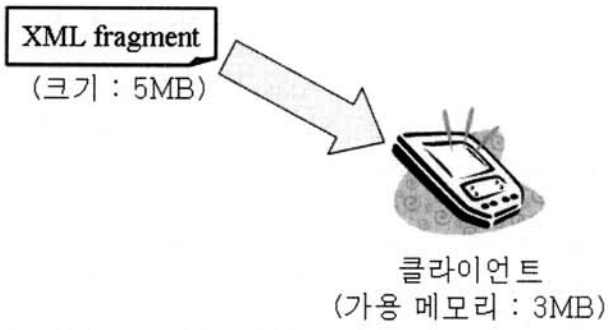
(그림 4)는 원본 XML 문서를 3개의 XML 조각으로 분할한 예를 나타내고 있다. 설명을 간단하게 하기 위해, 분할된 조각에 첨가되는 헤더는 나타내지 않았다 (이에 대해서는 3절에서 설명). (그림 5)는 (그림 4)의 조각들이 스트리밍되었을 때, 클라이언트에서 수신된 조각을 검사하고 처리하는 예를 보여주고 있는 것으로 사용자의 질의가 "/a/b/c"이므로, 질의 처리 시 <c>...</c>을 포함한 조각에 대해서는 질의 결과를 얻기 위한 처리가 필요하며 나머지 조각에 대해서는 더 이상의 처리가 불필요한 것을 나타내고 있다. 이를 위해 클라이언트는 조각이 도착했을 때, 조각의 헤더를 분석하여 (3절 참조) 추가 처리가 필요한 조각과 그렇지 않은 조각을 구분할 수 있어야 한다. [3-5]의 기법들은 홀-필러 모델 또는 XML 레이블링 기법을 바탕으로 그와 같은 조각 스트림 처리 기법을 제시하였다. 그런데 이들 기법들은 스트리밍되는 XML 문서의 전체 크기가 커질 경우 (동적인 정보를 포함하거나 문서 자체의 크기가 매우 큰 경우) 메모리 사용량이 스트리밍되는 문서의 크기에 비례하여 지속적으로 증가하는 경향을 갖는다. [6]에서는 이 같은 문제를 해결할 수 있는 기법들을 제시하고 [3-5]에서 제시한 XML 조각 스트림 질의 처리 기법에 적용하여 XML 문서 크기에 따른 확장성을 제공하였다.

하지만 이들 기법들에서는 XML 문서 분할 방법에 대해서는 다루지 않았다. XML 문서 분할에 관한 연구로는 분산 컴퓨팅 환경에서의 XML 문서 분할 기법을 다룬 것이 있다 [7-11]. [7]에서는 XML 데이터가 일반적으로 웹에서 분산되어 있음에 착안하여, XML 문서를 분할하고 적합한 저장소에 배치함으로써 효율적인 분산 처리 환경을 설계하는 데 중점을 두었다. [8, 9]에서는 관계 데이터베이스에서의 대표적인 분할 기법인 수평 분할 및 수직 분할을 XML 데이터에 적용하는 방법에 대해 다루었다. [9]에서는 [7, 8]의 연구를 보완하여 XML 데이터에 대한 수직, 수평, 혼합 분할 기법에 대해 정의하였고 단일 XML 문서를 대상으로 한 분할뿐만 아니라 여러 개의 XML 문서들로 이루어진 집합에 대한 분할에 대해서도 다루었다. [10]에서는 분산 컴퓨팅 환경에서 질의 처리 시, 중간 결과의 크기가 커지는 문제에 대한 해결책으로 문서의 구조적 특성 (XML 문서를 트리로 나타내었을 때, 서브트리의 크기, 깊이, 너비 등의 구조 정보)을 반영한 문서 분할 기법에 대해 제시하였다. 이외에도 웹 서비스 및 P2P 응용에 적용할 수 있는 XML 분할 기법이 제안되었다[11]. 그러나 이들 연구는 모두 분산 컴퓨팅 환경에서 XML 데이터의 분산 저장 및 처리를 통한 효율성 제고가 목적으로 XML 조각 스트림에 대한 질의 처리에 적용할 수 없다.

XML 데이터를 분할하여 스트리밍하는 연구로는 [15]가 있다. 이 기법에서는 XDU라는 단위로 XML 문서를 분할한다. XDU는 네트워크를 통해 전송할 수 있는 최대 데이터의 양보다 작은 크기를 갖는 문서 분할의 단위이다. 이 기법은 소스에서 대량의 XML 문서를 분할 스트리밍하여 수신측에서 원래의 XML 문서를 복원하는 과정에 있어 네트워킹의 효율성을 높이는 것이 목적으로 본 논문의 기법과는 용도가 다르다.

본 논문에서 제시하는 XML 분할 기법의 비교 대상이 될 수 있는 기법으로는, [5]의 스키마 기반 분할 기법과 [16]의 엘리먼트 질의 빈도 기반 분할 기법이 있다. 전자는, XML 문서를 트리로 나타내었을 때 일반적으로 깊이가 깊지 않고 너비가 넓다는 특징[17]에 착안하여, XML 문서의 DTD에서 반복되는('*' 나 '+'기호가 붙은) 엘리먼트를 하나의 조각으로 따로 분할하는 방식으로 분할의 결과가 질의 처리 시 자원 사용 효율에 미치는 영향은 고려하지 않았다. 후자는 그러한 영향을 고려한 것으로 PFT(Path Frequency Tree)라 불린다. PFT는 XML 문서를 트리로 나타내었을 때 각 노드별로 사용자 질의의 경로가 접근을 요구하는 빈도를 표시하고 이 빈도 값에 따라서 빈도가 낮은 노드는 그 형제 노드가 할당된 조각으로 할당하여 형제 노드 간에 묶고, 빈도가 높은 노드는 그 부모 또는 자식 노드가 할당된 조각으로 할당하여 부모-자식 노드 간에 묶음으로써 질의 처리의 효율을 높이는 휴리스틱을 사용하였다.

이들 두 기존 XML 분할 기법[5, 16]의 근본적인 문제점은 조각의 크기 및 개수 간의 상관관계와 그것이 질의 처리의 효율에 미치는 영향을 고려하지 않았다는 것이다. 첫째,



(그림 6) XML 조각의 크기가 클라이언트의 가용 메모리를 초과하는 경우

분할된 조각의 크기에 대한 제약을 고려하지 않았다. 무엇보다도 질의 처리를 수행하는 클라이언트의 가용 메모리 용량을 고려하지 않았다. 대용량의 XML 문서를 조각으로 분할할 때, 분할된 조각의 크기가 클라이언트의 가용 메모리보다 클 수 있다. 이런 경우, 클라이언트는 해당 조각을 전송받을 수 없으므로 질의 처리가 불가능해지게 된다 (그림 6). 둘째, 분할되는 조각의 개수의 제약에 대해 고려하지 않았다. 조각의 크기를 너무 작게 하는 경우에는 조각의 개수를 증가시키는데 조각의 개수는 클라이언트의 질의 처리 시간에 직접적으로 영향을 미치게 된다. (그림 5)에서 나타낸 것처럼 클라이언트는 질의 처리 시 전송받은 모든 조각에 대해 검사를 수행하는데, 이 시간은 조각의 개수에 비례하기 때문이다. 조각 수가 많으면 조각 간의 관계 파악 부담도 가중시킨다.

3. XML 데이터의 효율적인 분할 기법

본 절에서는 XML 문서 분할이 클라이언트의 자원 사용량에 미치는 영향에 대해 분석하여 XML 조각 스트림 질의 처리의 비용 모델을 제시하고, XML 조각 스트림 질의 처리 시 클라이언트의 자원을 효율적으로 사용하기 위한 XML 문서 분할 기법을 제시한다.

XML 문서를 분할하는 방법은 매우 다양한데 그 수는 XML 문서 내에 존재하는 서로 다른 경로의 개수에 대해 지수적으로 증가한다. 예를 들어, (그림 4)의 XML 문서 내에 존재하는 모든 경로의 수는 "/a", "/a/b", "/a/b/c", "/a/e"의 4개이며 이 4가지를 조합해서 나올 수 있는 서로 다른 분할은 서브 트리 단위의 분할만 고려해도 <표 1>에서 나타낸 것과 같이 8가지이다. 문서의 최상위 엘리먼트로만 구성된 경로 (위의 예에서 "/a")를 제외한 나머지 경로는 그 단말 엘리먼트를 루트로 하는 서브 트리 조각으로 분할해 낼 수도 있고 그 상위 엘리먼트가 포함된 조각에 내포시킬 수도 있다. 따라서, 문서 내에 존재하는 서로 다른 경로의 수를 n이라고 할 때, 서브 트리 단위의 가능한 분할의 총 개수는 2^{n-1}개이다. 예를 들어, XMark benchmark[18]의 xmlgen 프로그램으로 생성한 10.8MB 크기의 auction 문서의 경우, 서로 다른 경로의 개수는 502개이며 가능한 분할의 총 개수는 2^{501}개이다. 이와 같은 경우 최적의 분할을 찾기 위해 모든 가능한 분할에 대해 그 효율성을 점검하는 것은 적합하지 못하므로 최적의 분할보다는 준최적의 분할을 찾아야 한다. 본 논문에서는 조각의 크기(개수) 제약과 사용자 질의를 고려하여 효율적인 분할을 찾는 기법을 제시한다.

〈표 1〉 4가지 경로에 대해 가능한 분할 방법들

분할되는 경로
/a
/a, /a/b
/a, /a/b/c
/a, /a/e
/a, /a/b, /a/b/c
/a, /a/b, /a/e
/a, /a/b/c, /a/e
/a, /a/b, /a/b/c, /a/e

3.1 배경 지식

3.1.1 태그 구조

[3-6] 등 기존의 XML 조각 스트림에 대한 질의 처리 기법들은 모두 [2]의 태그 구조(tag structure)를 사용하여 문서 분할 구조를 표현하였다. 태그 구조는 XML 문서에 존재하는 모든 경로에 고유한 식별자를 부여하고 이 경로에 대한 정보를 XML로 나타낸 것으로 [3, 4]의 기법이 사용한 홀-필러 모델이나 [5, 6]의 기법이 사용한 XML 조각 레이블링 둘 다에 적용 가능하다. (그림 7)은 (그림 4)의 XML 문서에 대한 태그 구조를 나타낸다. (그림 7)에서 fragment="true" 항목은 해당 경로의 엘리먼트를 루트로 하는 서브 트리가 별도의 조각으로 분할됨을 나타낸 것으로, (그림 4)에서 분할된 각 조각의 루트에 해당되는 엘리먼트에 fragment="true"가 명시되어 있다. XML 문서를 분할하게 되면, 분할된 각 조각은 태그 구조에 정의된 정보를 조각의 헤더에 명시한다.

```
<tag stream="figure">
  <tag id="1" name="a" fragment="true">
    <tag id="2" name="b">
      <tag id="3" name="c" fragment="true"/>
    </tag>
  <tag id="4" name="e" fragment="true"/>
</tag>
```

(그림 7) (그림 4)의 XML 문서에 대한 태그 구조

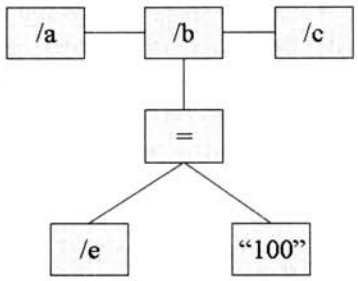
```
<fragment id="1" tsid="1">
  <a><b></b></a>
</fragment>

<fragment id="1.1" tsid="3">
  <c>10</c>
</fragment>

<fragment id="1.2" tsid="4">
  <e>100</e>
</fragment>
```

(그림 8) (그림 4)의 각 조각에 헤더 정보를 추가한 모습

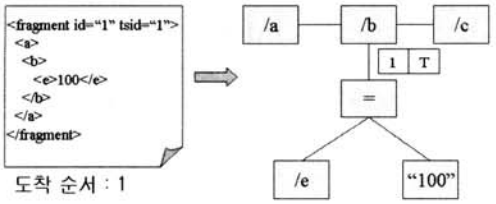
(그림 8)은 (그림 4)에서 나타낸 각 조각들에 헤더를 추가한 모습을 나타낸 것이다. 헤더에는 조각 식별자(fragment id)와 태그 식별자(tsid)가 포함된다. 태그 식별자는 문서의 서로 다른 경로에 고유하게 부여된 값이다. (그림 4)의 XML 문서에서 "a" 엘리먼트를 루트로 하는 조각의 경우, 조각 식별자 값 1과 이 엘리먼트의 태그 식별자 값 1을 조각 헤더에 기록한 것을 볼 수 있다.

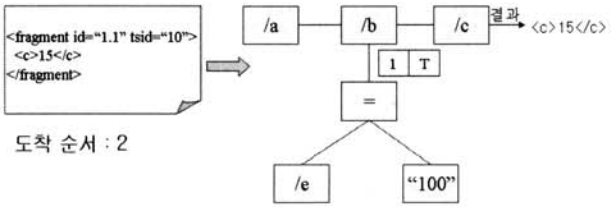
3.1.2 XML 조각 스트림에 대한 질의 처리 구조

XML 조각 스트림에 대해 질의 처리를 수행하기 위해서는 임의의 순서로 스트리밍되는 XML 조각이 원본 문서의 어느 부분에서 분할된 것인지에 대한 정보가 필요하다. XFrag [3]에서는 이를 위해, 서버 측에서는 태그 구조를 전송하여 XML 문서의 구조 및 분할에 대한 정보를 클라이언트가 파악할 수 있도록 하고, 클라이언트는 서버로부터 전송 받은 태그 구조 및 사용자 질의를 기반으로 질의 처리 연산자 파이프라인(pipeline)을 생성한다. 연산자 파이프라인은 XPath 질의의 각 로케이션 스텝(location step)을 처리하는 연산자들을 연결해 놓은 형태이다. 스트리밍된 XML 조각이 연산자 파이프라인에 전달되면, 각 연산자는 해당 조각에서 질의 처리에 필요한 정보를 추출하고 다른 연산자들과 정보를 교환하며 질의 처리를 수행한다. (그림 9)는 사용자 질의 "/a/b[e="100"]/c"에 대한 연산자 파이프라인을 나타내며 (그림 10)은 연산자 파이프라인에서 질의 처리가 이루어지는 과정을 간략히 나타낸 것으로 조각 간 관계 표현은 [5]에서



(그림 9) 질의 처리 연산자 파이프라인





(그림 10) 연산자 파이프라인에서의 질의 처리 과정의 예

제안한 XML 조각 레이블링을 사용하고 있다. 그림 10에서 첫 번째 조각이 도착하면, 이와 관련된 연산자인 "/a", "/b", "/e"에서 이들에 대한 처리를 수행한다. 즉 조건을 비교하고 해당 조각이 질의를 만족하는가의 여부를 (조각 식별자, 처리 결과)의 형태로 저장한다. 첫 번째 조각의 식별자가 1이고 이 조각은 조건을 만족하기 때문에 처리 결과를(1, T)의 형태로 저장한다 (T는 True를 나타냄). 두 번째 조각이 도착하게 되면 이와 연관된 "/c" 연산자에서 이 조각을 처리하게 된다. 조각 식별자가 1.1이기 때문에 (이 예에서는 조각 식별자로 prefix 레이블링 기법인 Dewy order encoding [14]이 사용되었음) 해당 조각의 부모 조각인 식별자가 1인 조각의 처리 결과를 참조하여 결과로 <c>15</c>를 출력한다.

3.2 XML 문서 분할이 자원 사용량에 미치는 영향

XML 조각 스트림 질의 처리 시 XML 문서가 분할된 방법에 따라 클라이언트의 처리 시간 및 메모리 사용량에서 차이를 보이게 된다. 본 절에서는 XML 문서 분할이 질의 처리 시 메모리 사용량 및 처리 시간에 미치는 영향에 대해 예를 통해 기술한다.

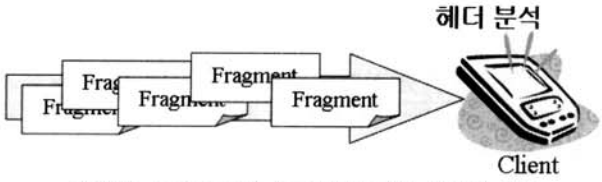
3.2.1 조각의 개수 및 크기

XML 문서 분할 시 분할되는 조각의 크기와 개수는 트레이드오프 관계에 있다. 즉, 각 조각의 크기가 증가하게 되면 조각의 개수는 감소하는 경향을 보이게 된다. 극단적인 예를 들어 XML 문서를 단 하나의 조각으로 분할한다면, 조각의 크기는 XML 문서의 크기에 조각 헤더의 크기를 더한 것과 같다. 이와는 반대 극단으로 모든 엘리먼트를 개별 조각으로 분할하는 경우, 조각의 크기는 엘리먼트 하나의 평균 크기 정도로 감소하지만 조각의 개수는 XML 문서에 존재하는 모든 엘리먼트의 수와 동일하게 증가한다.

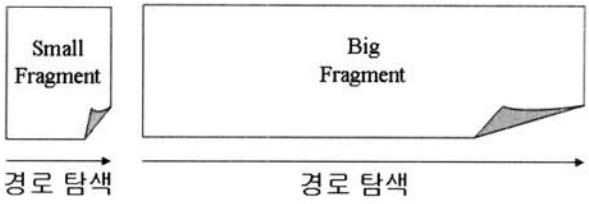
조각의 개수는 헤더 분석 시간과 직접적인 관계를 갖는다. 헤더 분석은 스트리밍된 모든 조각에 대해서 수행되기 때문에, 조각의 개수 증가는 헤더 분석 시간 증가를 가져온다. 따라서 헤더 분석 시간을 감소시키기 위해서는 분할되는 XML 조각의 총 개수를 줄여야 한다. 하지만 조각의 개수가 감소하게 되면 XML 조각의 크기가 증가하게 되며, 이는 조각을 수신하여 저장하기 위한 메모리 사용량을 늘릴 뿐 아니라 연산자 파이프라인에서의 처리 시간을 증가시키는 요인이 된다. 즉 처리 대상 조각의 크기가 전반적으로 커짐으로써 해당 조각에서 각 연산자가 필요로 하는 정보를 찾기 위한 시간이 증가하기 때문이다. (그림 11)은 조각의 개수 및 크기가 질의 처리 시간에 미치는 영향을 나타낸 것이다.

3.2.2 XML 조각 처리와의 자원 사용

분할된 XML 조각이 어떤 엘리먼트를 포함하고 있는가에 따라서 질의 처리 시 자원 사용량이 달라질 수 있다. 질의 처리 시 생성되는 연산자 파이프라인은 각 조각이 도착할 때마다 부분적으로 질의 처리를 수행하게 된다. 질의 처리를 완료할 만큼 충분한 조각을 받기 전까지 각 연산자는 부분 질의 처리 결과(중간 결과)들을 저장하고 있게 된다. (그

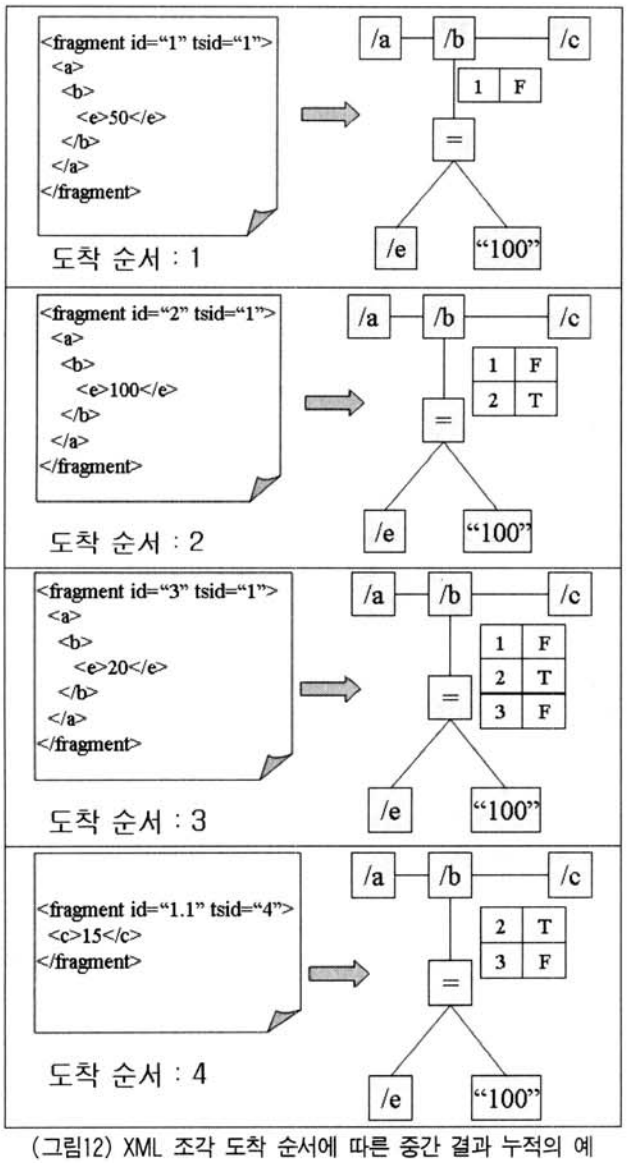


조각의 수가 증가할수록 헤더 분석 시간이 증가



조각의 크기가 증가할수록 조각당 처리 시간이 증가

(그림 11) 조각의 개수 및 크기가 처리 시간에 미치는 영향



(그림12) XML 조각 도착 순서에 따른 중간 결과 누적의 예

림 12)는 질의 처리 과정에서 발생하는 중간 결과의 누적으로 인한 메모리 사용량 증가를 나타낸 것으로 위에서부터 차례로 조각이 도착할 때 연산자 파이프라인에 저장되는 정보를 보여준다. 그림에서 "/b"에 해당하는 연산자는 조각 식별자와 해당 조각에 대한 처리 결과 (True, False 등)의 쌍으로 중간 결과를 저장하고 있다. 조각 식별자가 1, 2, 3인 조각들은 해당 조각의 자식에 해당하는 조각이 오기 전까지 중간 결과를 저장하고 있어야만 한다. 따라서, 4번째 조각(식별자가 1.1인 조각)이 도착하기 전까지 "/b"연산자는 중간 결과들을 모두 저장하고 있다가, 4번째 조각이 도착하게 되면, 조각 식별자가 1인 조각의 중간 결과를 삭제하게 된다. 이런 상황은 실제 응용에서 충분히 발생할 수 있으며, 최악의 경우 중간 결과 누적으로 인해 가용 메모리를 전부 소진하게 되어 질의 처리를 할 수 없게 될 수도 있다.

이와 같은 메모리 누적 현상을 방지하기 위해서는 메모리에 누적되는 질의 처리 중간 결과를 가능한 줄여야 한다. 이를 위해서는 질의 처리를 완료하는 데 필요한 부분이 질의 처리기에 순서대로 도착하게 되는 것이 최선이다. 하지만 서버 측에서 XML 조각을 질의 처리에 적합한 최적의 순서로 스트리밍해도, 방송 환경이나 무선 환경에서는 XML 조각이 임의의 순서로 클라이언트에 도착할 수도 있기 때문에 이 방법에는 한계가 있다.

다른 방법으로는 XML 문서 분할 시 질의에 대한 정보를 반영하는 것이다. 질의를 고려하여 문서를 분할하게 되면 최선의 경우 질의 처리의 어떤 과정에 필요한 정보가 모두 동일 조각에 포함되어 중간 결과를 저장하지 않고 바로 해당 질의 처리 과정을 완료할 수 있어 메모리 누적 현상을 해결할 수 있다. 하지만 질의 처리의 어느 과정에 필요한 정보를 모두 동일 조각에 포함시킬 경우 조각의 크기가 증가할 수 있어, 질의 처리 시간이 길어질 수 있다. 따라서 메모리 사용량과 처리 시간의 관계를 고려한 효율적인 분할 알고리즘이 요구된다.

3.3 질의 처리 비용 모델

다른 분할들 간에 서로 비교하고 더 나은 분할을 찾기 위해서는 해당 분할로써 질의 처리 시 소요되는 비용을 예측할 수 있어야 한다. 본 절에서는 3.2절에서 설명한 조각의 크기 및 개수를 기반으로 질의 처리 비용 모델을 제시한다. XML 문서를 조각으로 분할한 전체 조각의 개수를 n이라 하고, 헤더 검사를 통해 처리 대상으로 결정된 조각의 개수를 m이라고 할 때, 예상되는 질의 처리 시간은 다음과 같이 모델링할 수 있다.

예상 질의 처리 시간
= $\sum_n$조각 처리 시간
= $\sum_n$헤더 분석 시간 + $\sum_m$(경로 탐색 시간 + 평가 및 질의 처리 정보 저장 시간)
= n*1 + $\sum_m$(경로 탐색 시간 + 평가 및 질의 처리 정보 저장 시간)
= n*1 + $\sum_m$(처리 대상 조각에 포함된 엘리먼트 수*1 + 평가 및 질의 처리 정보 저장 시간)

질의 처리 시간은 모든 조각에 대해 헤더를 분석하는 시간과 처리 대상 조각에 대해서 연산자 파이프라인에서의 처리 시간을 모두 더한 것이 된다. 헤더를 처리하는 시간은 엘리먼트 하나에 대해서 처리하는 시간이며, 모든 조각에 대해서 거의 같은 시간이 소요된다. 엘리먼트 하나를 처리하는 데 걸리는 시간을 1로 정규화하면, 모든 조각의 헤더에 대해서 분석하는 데 걸리는 시간은 n*1이다. 연산자 파이프라인에서의 처리는 크게 두 단계로 나누어지는데, 처리 대상 조각에서 각 연산자가 필요로 하는 엘리먼트를 탐색(파싱)하는 경로 탐색과 해당 경로에 대해서 평가하고 결과를 저장하는 것이다. 경로 탐색 시간은 조각의 크기에 비례하며, 이는 조각에 포함된 엘리먼트의 수에 비례한다. 엘리먼트 하나를 처리하는 데 걸리는 시간을 1로 정규화 하였으므로 경로 탐색 시간은 조각에 포함된 엘리먼트의 개수 * 1이다. 평가 및 질의 처리 정보 저장 시간을 k라 할 때, 평가 및 질의 처리 정보 저장은 연산자 파이프라인에서 조각을 처리할 때마다 수행되므로 총 m*k의 시간이 소요된다. 따라서 앞의 식을 정리하면 다음과 같다.

예상 질의처리시간
= n + m(처리 대상 조각에 포함된 평균 엘리먼트 수 + k)

k는 엘리먼트 하나를 처리하는 시간에 비해 평균적으로 약 5배의 시간이 소요되는 것을 [6]의 기법을 이용한 실험을 통해 추정하였다. 이 값은 실험 환경에 따라 변할 수 있는 값으로 정확하지 않을 수 있지만 본 논문에서는 이 값을 5로 설정하고 연구를 진행하였다. 이상에서 제시한 비용 모델은 엘리먼트의 수를 기반으로 비용을 산출한 것으로서, 엘리먼트의 길이는 다를 수 있으며 또한 질의 처리 시 발생하는 부수적인 연산에 대한 비용을 반영하지 않았기 때문에 완벽하지는 않다. 하지만 XML 질의 처리 시 대부분의 시간을 소모하는 부분이 경로 탐색이며 이는 조각에 포함된 엘리먼트의 수에 비례하여 증가한다는 점과 각 엘리먼트의 길이에 따른 처리 시간은 차이가 있지만 그 영향이 크지 않다는 점에 의해 본 논문에서는 엘리먼트의 개수만을 고려하였다.

3.4 XML 문서 분할

본 절에서는 앞서 살펴본 사항들을 바탕으로 XML 문서 분할 기법을 제시한다. 질의 처리 시간을 줄이기 위해서는 XML 문서에서 질의 처리와 연관되지 않은 부분에 대해서는 조각의 크기를 가능한 크게 하여 헤더 검사 시간을 줄이고, 질의 처리와 관련된 부분은 조각의 크기를 작게 하여 연산자 파이프라인에서의 처리 시간을 줄여야 한다. 메모리 효율성 측면에서는 질의 처리와 연관된 엘리먼트들이 한 조각 내에 포함될 수 있도록 하되 크기가 너무 커지지 않도록 한다. 조각이 너무 크면 그 조각을 수신하여 저장하기 위한 메모리가 크게 요구되기 때문이다.

XML 문서는 텍스트로 작성되어 있으며, 크기가 클 수 있으므로 XML 문서를 메모리에 로드하여 문자열을 자르고 이동하는 등의 분할 작업은 비효율적일 수 있다. 따라서 실제 분할에 앞서 분할 스키마를 만들고 이 에 따라 분할을 수행하는 것이 적절하다. 즉 XML 문서의 어떤 경로들을 분할할 것인가를 미리 결정하고, 이 결정에 따라 실제 XML 문서를 분할하는 것이다. 본 논문에서는 태그 구조에 분할을 명시한 후, 이를 기반으로 실제 XML 문서를 분할하는 방법을 사용한다.

분할은 (1) 문서 분석, (2) 크기 기반 분할, (3) 질의 기반 분할, 그리고 (4) 형제 서브트리 병합의 네 단계로 수행된다. 문서 분석에서는, 먼저 클라이언트의 메모리 제약 사항을 고려하여, 조각의 최대 크기를 정한다. 응용 분야에 따라서, 클라이언트의 메모리 용량은 차이가 나기 때문에 본 논문에서는 조각 크기의 상한 값을 임의의 크기로 설정 가능하도록 하였다. 분할된 조각의 크기를 이 설정 값 이하로 제한하기 위해 분할 대상 XML 문서에 존재하는 모든 경로를 대상으로, 각 경로에 해당하는 엘리먼트의 크기 등을 계산한다. 크기 기반 분할에서는 문서 분석에서 계산된 각 엘리먼트의 크기 정보 및 태그 구조를 이용하여, 정의된 조각의 크기 제약을 만족하는 분할을 수행한다. 질의 기반 분할에서는 추가로 사용자 질의를 고려한 분할을 수행하고, 마지막으로 분할된 조각 중 선별적으로 형제 서브 트리 간의 병합을 수행한다.

3.4.1 XML 문서 분석

분할되는 XML 조각의 크기를 제한하기 위해서는 분할 대상 XML 문서의 모든 경로에 해당하는 엘리먼트들의 크기를 먼저 파악해야 한다. 분할 대상 XML 문서 전체를 파싱하면서 각 경로별로 <표 2>와 같은 정보를 추출한다. 이 정보는 분할 시 파생되는 조각의 개수 및 크기 등에 대해 예측하여 XML 문서의 특정 경로에 대해 따로 분할해 낼 것인가의 여부를 결정할 때 사용된다.

〈표 2〉 XML 문서 분석 시 각 경로별로 추출하는 정보

항목	설명
총 크기	각 경로의 엘리먼트를 루트로 한 서브트리의 전체 크기(byte 단위)
반복 수	해당 경로가 문서 내에서 나타나는 총 횟수
평균 크기	각 경로의 엘리먼트를 루트로 한 서브트리의 평균 크기(byte 단위)
후손 엘리먼트의 개수	각 경로의 엘리먼트를 루트로 한 서브트리에 포함된 전체 노드의 수

3.4.2 크기 제약을 만족하는 분할

크기 제약을 만족하는 분할이란 XML 문서를 분할할 때 조각의 크기가 정해진 최대 크기보다 작도록 분할하는 것이다. 본 알고리즘은 질의 처리 시 헤더 분석 시간을 최소화할 수 있도록, 분할되는 각 조각의 크기가 크기 제약을 만족하는 범위 안에서 가능한 크게 분할되도록 한다. (그림 13)는 크기 제약을 만족하는 분할의 핵심 요소인 알고리즘 1을 나타낸다. 본 알고리즘은 태그 구조를 트리의 형태로 나타낸 자료 구조를 순회하면서 태그 구조에 분할을 명시한다. 즉 태그 구조의 루트 노드에서 시작하여 각 노드를 루트로 한 서브 트리의 크기가 미리 정의된 크기 제한보다 작은 경우는 알고리즘을 종료하고(1행), 클 경우는 입력 노드의 자식 노드들을 대상으로 본 알고리즘을 재귀적으로 수행한다(2-4행). 재귀 호출 이후에도 입력 노드가 크기 제한을 만족하지 않을 경우, 크기 제한을 만족할 때까지 입력 노드의 자식 노드들을 분할한다(5-9행). 자식 노드들이 개별의 조각으로 분할되어 나가기 때문에 입력 노드의 크기는 점점 줄어들게 되고, 결국 크기 제한을 만족하게 된다. 본 알고리즘은 재귀적으로 수행되기 때문에, 결국 모든 조각이 크기 제한을 만족하도록 분할 구조를 정의하게 된다.

(그림 14)의 알고리즘 2는 태그 구조에 분할을 명시하는 과정 (알고리즘 1의 8행)을 나타낸다. 특정 경로를 분할하게 되면 그 경로를 포함하고 있던 조각은 분할로 인해 영향을 받게 된다. 즉 조각의 전체 크기 및 조각에 속한 노드의 개

알고리즘1:fragmentElementBySize

```
input item : 태그 구조를 트리로 구성하였을 때, 트리의 특정 노드(서브 트리)

/* 입력으로 받은 노드가 크기 제한을 만족하는 경우, 종료 */
1:if(item.getSubtreeSize()<=sizeLimit)return;
/* 입력으로 받은 노드의 자식 노드들을 크기(자식 노드를 루트 노드로 한 서브트리의 최대 크기) 순으로 정렬한 리스트 생성 */
2:Listchildren=item.getSortedChildren();
/* 자식 노드를 인자로 재귀 호출 */
3:for(child:children){
4:        fragmentElementBySize(child);
}
/* 재귀 호출 후에도 현재 노드가 크기 제한을 만족하지 않는 경우, 크기 제한을 만족할 때까지 자식 노드들을 분할한다. */
5:if(item.getSubtreeSize()>sizeLimit){
6:        children=item.getSortedChildren();
7:        for(TreeItemchild:children){
                /* 자식 노드에 분할을 명시한다. */
8:              setFragmentMarkToNode(child);
                /* 현재 노드가 크기 제한을 만족하면 종료 */
9:              if(item.getSubtreeSize()<=sizeLimit)break;
        }
}
```

(그림 13) 크기 기반 분할 알고리즘

알고리즘2:setFragmentMarkToNode

```
input item : 태그 구조를 트리로 구성하였을 때, 트리의 특정 노드(서브 트리)

/* 입력 노드를 포함하는 조각의 루트 노드를 가져온다 */
1:root=item.getFragmentRoot();
2:if(item==root)root=null;
3:if(root!=null){
4:        itemDescendant=item.getTotalDescendant();
5:        itemSize=item.getTotalSize();
        /* 입력 노드의 부모 노드 */
6:        p=item.getParent();
        /* 분할로 인해 변경되는 엘리먼트의 정보를 조각 루트까지 반영시킨다. */
7:        while(p!=root.getParent()){
8:              pDescendant=p.getTotalDescendant();
9:              pSize=p.getSize();
10:             p.setTotalDescendant(pDescendant-itemDescendant);
11:             p.setTotalSize(pSize-itemSize);
12:             p=p.getParent();
        }
}
13:item.setFragment(true);/* 분할을 명시한다. */
```

(그림 14) 태그 구조에 분할을 명시하는 알고리즘

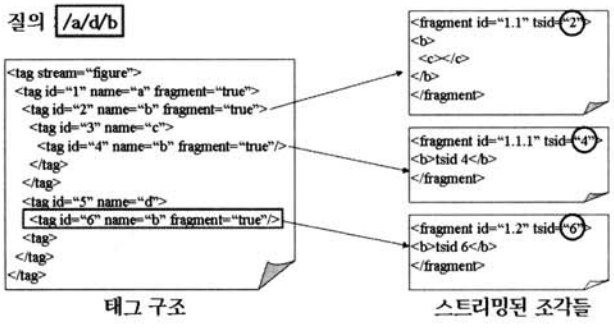
수 등이 변하기 때문에, 이러한 정보를 갱신한다(7-12행).

3.4.3 질의를 고려한 분할

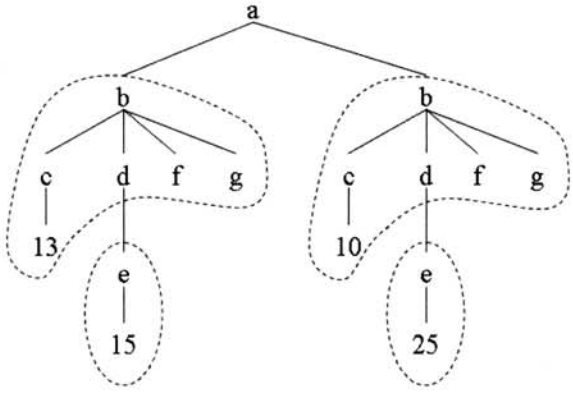
질의를 고려한 분할은 질의 처리 시 처리 시간 및 메모리 효율성을 높이기 위한 분할이다. 이와 같은 분할을 설명하기 위한 배경 지식으로 먼저 태그 구조를 통한 질의 처리 최적화에 대해 설명한다. XPath 질의는 축(axis), 노드 테스트(node test), 프레디킷(predicate)으로 구성된 로케이션 스텝들로 이루어진다. XML 조각 스트림 질의 처리에서는 태그 구조 덕분에 모든 로케이션 스텝들에 대해서 평가할 필요가 없이, 로케이션 스텝에서 프레디킷이 포함된 스텝(이하 '필터 스텝'으로 표기)과 가장 마지막 단계의 스텝(이하 '결과 스텝'으로 표기)에 대해서만 평가를 수행하면 된다. 필터 스텝은 질의 처리 시 해당 조각의 자식 조각들에 대한 필터링을 위한 것이며, 결과 스텝은 질의 처리 결과를 출력 사용자에게 보여주기 위한 것으로, 예를 들어 사용자 질의가 "/a/b[c=10]/d/e"인 경우, "/b[c=10]"과 "/e"에 대해서만 평가를 수행해도 질의 처리가 가능하다. 그 이유는 XML 문서가 조각으로 분할되어 스트리밍될 때, 각 조각의 헤더에는 그 조각의 루트 엘리먼트의 태그 식별자가 명시되어 있기 때문이다[2]. 태그 식별자는 문서의 서로 다른 경로에 고유하게 부여된 값이므로, 예를 들어 XML 문서의 경로 중 "/a/b"와 "/a/d/b"가 있다고 할 때, 이 두 경로는 서로 다른 식별자를 부여받게 된다. 따라서 앞서 예를 든 질의의 경우, "/b[c=10]" 스텝에 대해서 경로가 "/a/b"인 엘리먼트를 식별하는 것이 가능하기 때문에, 모든 스텝에 대한 평가를 하지 않아도 질의 처리의 정확성을 보장할 수 있다. (그림 15)는 분할된 조각이 하나 주어졌을 때 그것의 태그 식별자를 통해 동일한 이름을 가진 서로 다른 세 경로의 "b" 엘리먼트들 (/a/b, /a/b/c/b, /a/d/b)이 구분 가능함을 보여주는 예이다.

헤더 분석을 통해 처리 대상 조각으로 결정된 조각은 질의 처리를 위해 연산자 파이프라인으로 전달된다. 질의를 반영한 문서 분할의 목표는 연산자 파이프라인에서의 조각 처리 시간을 단축시키고, 메모리 사용량을 줄이는 것이다. 이를 위해서는 연산자 파이프라인의 처리 대상 조각에서 질의 처리에 불필요한 엘리먼트들을 배제하여 조각의 크기를 줄여야 한다. 즉 질의를 분석하여 필터 스텝과 결과 스텝의 대상이 되는 엘리먼트가 포함되는 조각의 크기를 가능한 작은 크기로 분할해야 한다. (그림 16)은 이러한 분할을 나타낸 것이다. 그림에서 점선으로 둘러싼 부분이 조각으로 분할되는 부분임을 나타낸다. 필터 스텝은 처리하는 데 필요한 엘리먼트의 경로는 "/a/b"와 "/a/b/c"이므로 이 두 엘리먼트를 하나의 조각으로 분할하였다. 만약, 위의 두 엘리먼트가 각기 다른 조각으로 분할되어 있다면, (그림 16)의 질의를 처리하는 데 있어, "/a/b"에 해당하는 연산자는 "/a/b/c" 엘리먼트를 포함하는 조각이 도착할 때까지 처리가 지연되어 중간 결과를 유지해야만 한다. 이러한 경우를 방지하기 위해 프레디킷이 포함된 스텝에 해당하는 엘리먼트들(위의 경우, b와 c)을 동일한 조각으로 할당하였다. 유사하게 결과 스텝에 대해서도 "e" 엘리먼트 하나만 포함하도록 분할을 수행하였다.

하지만 앞서 제시한 방법으로 분할을 수행하였을 때의 결과가 이 질의의 경우에 최선의 분할인가는 보장할 수 없다. 즉, 결과 스텝에 해당하는 "e" 엘리먼트를 별도 조각으로 독립시킴으로써 연산자 파이프라인에서의 처리 시간을 단축시킬 수는 있으나, 이것은 전체 조각 수를 증가시킴으로써 헤더 검사 시간을 증가시키게 된다. 즉 연산자 파이프라인에서의 처리 시간 단축을 위한 분할이 헤더 검사 시간을 증가시킴으로써 비용 절감 효과가 상쇄될 수 있으며, 혹은 헤더 검사 시간의 증가로 인해 오히려 더 나쁜 결과를 초래할 수도 있다. 따라서 이와 같은 분할 (위의 예에서 e의 독립)을 결정하기 위해서는 3.3절에서 제시한 비용 모델을 이용하여 해당 분할이 분할 전보다 비용을 줄일 때에만 수행하도록 해야 한다. (그림 17)의 알고리즘 3은 이러한 분할 과정을 기술한 것으로 질의를 참조하여 태그 구조 트리에서 어떤 노드를 분할할 것인가를 결정한다. 2-7행은 입력 노드가 질의의 필터 스텝이나 결과 스텝에 해당하는지를 판단하고 비용 기반 분할을 수행한다. 3행은 입력 노드를 분할하기 전의 비용을 측정하며, 5행은 입력 노드를 분할한 후의 비용을 측정한다. 6-7행은 분할 후의 비용이 분할 전보다 더 높은 경우, 분할을 취소하는 과정을 나타낸다. 8-9행은 입력 노드의 자식 노드들을 대상으로 본 알고리즘을 재귀적으로 호출하는 것이다. 따라서 본 알고리즘에 태그 구조 트리의



(그림 15) 태그 식별자를 통한 경로 식별의 예

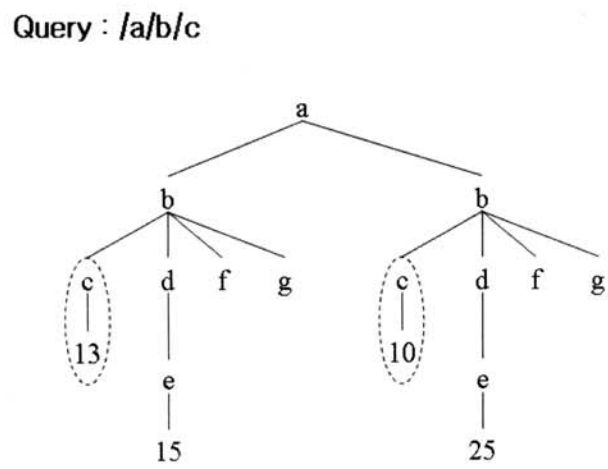


(그림 16) 질의를 고려한 XML 문서 분할의 예

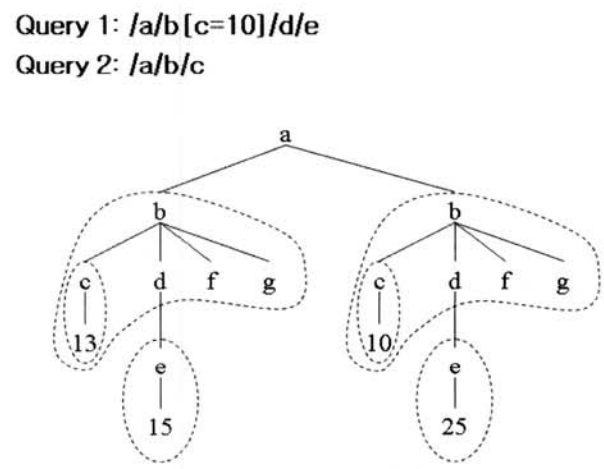
알고리즘3:fragmentElementByQuery

```
input item : 태그 구조를 트리로 구성하였을 때, 트리의 특정 노드(서브 트리)

/* 입력으로 받은 노드의 자식 노드들을 크기(자식 노드를 루트 노드로 한 서브트리의 크기) 순으로 정렬한 리스트 생성 */
1:Listchildren=item.getSortedChildren();
/* 질의의 결과에 해당하는 노드 및 프레디킷을 포함하고 있는 노드 분할 */
2:if(item.isRelatedToQuery()){
        /* 분할 전 비용 측정 */
3:      doubleprevTimeCost=estimateProcessingTime();
4:      setFragmentMarkToNode(item);
        /* 분할 후 비용 측정 */
5:      doublecurrTimeCost=estimateProcessingTime();
        /* 입력 노드를 분할하기 전이 분할한 후보다 비용이 작은 경우 */
6:      if(currTimeCost>prevTimeCost)
                /* 입력 노드의 분할을 취소 */
7:              revertFragmentMarkToNode(item);
}
/* 자식 노드가 있는 경우, 자식 노드를 인자로 재귀 호출 */
8:for(child:children){
9:      fragmentElementByQuery(child);
}
```

(그림 17) 질의 기반 분할 알고리즘



(그림 18) 질의를 고려한 XML 문서 분할의 다른 예



(그림 19) 두 개 질의에 대한 분할을 조합한 예

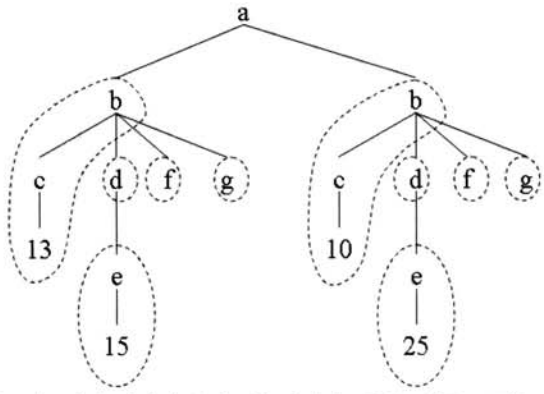
루트 노드를 입력하면 태그 구조 트리를 순회하면서 질의를 고려한 분할을 수행하게 된다.

알고리즘 3은 한 개 질의뿐만 아니라, 다수의 질의에 대해서도 효율적인 분할을 수행하는 데 이용된다. (그림 18)은 (그림 16)에 나타난 XML 문서에 대해 다른 질의를 고려한 분할을 나타낸다. (그림 18)에서는 사용자 질의가 "a/b/c"로 결과 스텝에 해당하는 c 엘리먼트를 개별 조각으로 분할하게 된다. 하지만 (그림 16)의 예에서는 c 엘리먼트와 b 엘리먼트를 동일한 조각으로 할당하게 되므로 이 두 분할은 서로 상충된다. 따라서 두 분할을 조합하거나 (그림 19) 두 분할 중 하나를 선택해야 한다. 이들 가능한 분할 후보 별로 평균 질의 처리 비용을 계산하여 가장 낮은 비용을 갖는 분할을 선택한다.

3.4.4 형제 서브 트리 병합

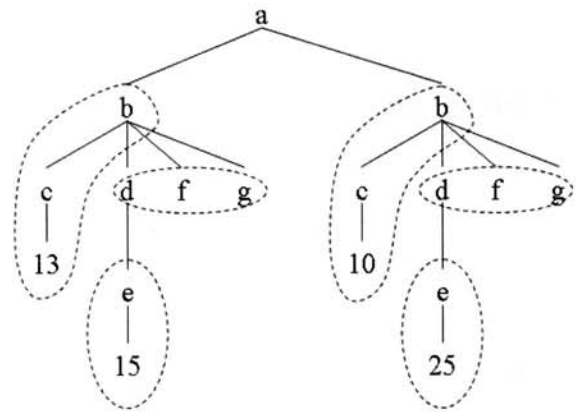
크기 제약을 만족하는 분할 및 질의를 고려한 분할을 수행하게 되면, 대부분의 질의에 대해서 처리 효율성이 높아지게 된다. 하지만 여전히 연산자 파이프라인에서의 처리 과정에는 비효율성이 존재할 수 있다. 예를 들어 (그림 16)의 분할을 볼 때, b를 루트로 한 조각에는 c, d, f, g의 자식 엘리먼트들이 포함된다. 이 중에서 (그림 16)의 질의 (/a/b[c=10]/d/e) 처리에 직접적으로 관련된 엘리먼트는 프레디킷의 조건에 해당하는 c 엘리먼트로 나머지 엘리먼트는 질의 처리의 결과에 영향을 주지 않는다. 하지만 이런 엘리먼트들이 동일 조각에

Query : /a/b[c=10]/d/e



(그림 20) 질의 처리에 불필요한 엘리먼트들을 다른 조각으로 분할한 예

Query : /a/b[c=10]/d/e



(그림 21) 서브 트리 병합을 적용한 예

알고리즘4:mergeSiblingSubtrees

```
input item : 태그 구조를 트리로 구성하였을 때, 트리의 특정 노드(서브 트리)

1: children = item.getChildren();
2: beforeCost = estimateProcessingTime();
3: for (TreeItem child : children) {
       /* 질의와 관련이 없고, 다른 조각에 포함되지 않은 자식 노드의 경우
        형제 서브 트리 병합의 대상이 된다.*/
4:     if (!child.isRelatedToQuery() && !child.hasChildFragment() && !child.isFragmentRoot()) {
5:             child.setSiblingSubtreeMerge(true);
               /* 병합되는 조각의 크기가 크기 제한을 넘는 경우 */
6:             if (item.getMergeRoot().getSize() > sizeLimit) {
                   /* 해당 노드의 병합을 취소한다. */
7:                 child.setSiblingSubtreeMerge(false);
}}}
   /* 형제 서브 트리 병합의 대상이 되는 노드들 중, 첫 번째 노드를 가져온다. */
8: mergeRoot = item.getMergeRoot(); if (mergeRoot==null) return;
9: mergeDesc = mergeRoot.getTotalDescendant();
10: mergeSize = mergeRoot.getSize();
11: item.setTotalDescendant(item.getTotalDescendant() - mergeDesc);
12: item.setTotalSize(item.getTotalSize() - mergeSize);
13: afterCost = estimateProcessingTime(false);
   /* 형제 서브 트리 병합를 했으나 비용이 더 나빠진 경우 되돌린다. */
14: if (beforeCost < afterCost)
15:     revertMerge(firstChild);
```

(그림 22) 비용 및 크기 기반의 형제 서브 트리 병합 알고리즘

포함됨으로써 경로 탐색 시간을 증가시키게 된다. 따라서 이러한 엘리먼트들을 다른 조각으로 분할해 내는 것이 처리 시간을 단축시키는 데 유용하다. (그림 20)은 (그림 16)의 분할에서 불필요한 엘리먼트들을 각각 다른 조각으로 분할한 모습이다.

하지만 (그림 20)의 분할에도 문제점은 잔존한다. 왜냐하면, d, f, g 엘리먼트들이 각각 다른 조각으로 분할됨에 따라 헤더 검사 시간이 증가하기 때문이다. 이런 문제점을 해결하기 위해 서브 트리 단위의 분할에 국한되지 않고 형제 서브 트리들의 병합을 고려할 수 있다. (그림 21)은 (그림 20)의 분할을 형제 서브 트리 병합을 적용하여 수정한 결과를 나타낸다. 형제 서브 트리 병합을 적용하면 질의 처리에 불필요한 엘리먼트들을 모두 하나의 조각에 포함시킬 수 있으므로, 각각을 별도 조각으로 분할하는 것보다 처리 효율성을 높일 수 있다. 그러나 이 방법도 다수 질의에 대한 분할 시에는 질의 별로 득실이 다를 수 있어 최선이 아닐 수 있다. 따라서 형제 서브 트리 병합도 3.3절의 비용 모델에 의거하여 평균 질의 처리 비용이 감소하는 경우에만 적용한

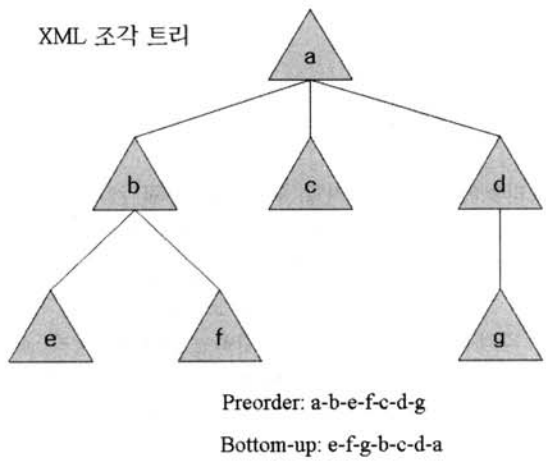
다. 또한 이러한 병합이 조각의 크기 제한을 충족할 때에만 적용 가능하다. (그림 22)는 이러한 비용 및 크기 기반의 형제 서브 트리 병합을 수행하는 알고리즘 4를 기술한 것이다.

4. 구현 및 성능 평가

4.1 개요

본 절에서는 본 논문에서 제안한 기법의 구현 및 실험을 통한 기존 기법들과의 성능 비교 평가 결과를 기술한다. 본 논문에서 제안한 XML 문서 분할 기법은 조각의 크기 및 질의를 고려한 것이므로 크기/질의 기반 XML 분할(XML Fragmentation by Size and Query)라 부르고 (이하 SQ로 표기), 이를 [5]에서 제시한 스키마 (DTD) 기반의 분할 기법 (이하 D로 표기) 과 [16]에서 제안한 PFT 기법과 비교하였다.

이들 세 기법들을 J2SE Development Kit 5.0 update 11을 사용하여 구현하였다. 성능 실험은 Intel Pentium D 2.66GHz CPU 및 1.5GB의 램을 탑재한 Windows XP 플랫폼의 시스템에서 수행하였다. 성능 평가에 사용된 실험 데이터는 XMark benchmark[18]의 xmlgen 프로그램을 사용하여 생성하였으며, 크기는 10.8MB이다(xmlgen 프로그램의 scaling factor 인자로 0.1을 사용하였으며, 생성된 문서에서 불필요한 띄어쓰기(space), 탭(tab), 그리고 줄바꿈 문자를 제거하였다). XML 조각 스트림 질의 처리 알고리즘으로는 [6]의 것을 이용하였다. 클라이언트가 XML 조각 스트림을 전송 받는 순서는 서버에서 전송하는 순서와 다를 수 있으므로, 본 실험은 조각이 XML 문서 트리 기준으로 preorder 순서로 도착하는 최선의 경우와 bottom-up 순서로 도착하는 최악의 상황을 모두 고려하였다. (그림 23)은 XML 문서를 분할한 각 조각을 트리 구조로 나타내고, preorder 전송 순서와 bottom-up 전송 순서를 예시한 것이다. 성능 척도로는 질의 처리가 수행된 클라이언트에서의 질의 처리 시간과 메모리 사용량을 이용하였다. 모든 실험은 동일한 환경에서 10회 반복 수행하였고, 이에 대한 평균값을 취하여 나타내었다.



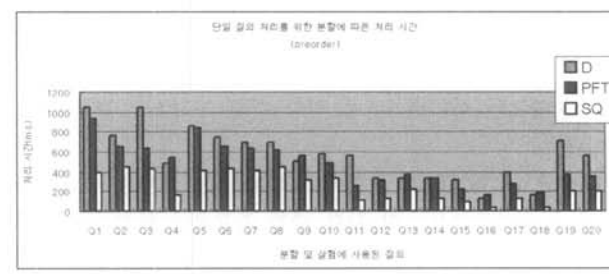
(그림 23) 실험에 사용된 조각의 전송 순서

4.2 단일 질의 처리를 위한 XML 분할의 평가

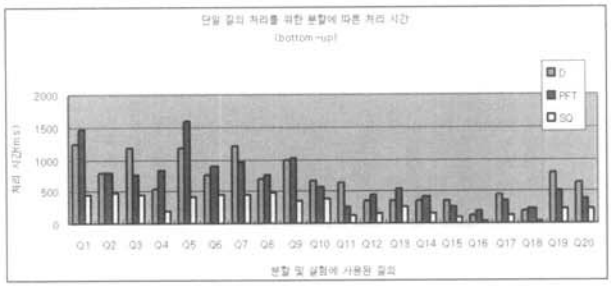
본 절에서는 단일 질의를 대상으로 XML 분할 알고리즘을 적용하여 얻어진 분할로 해당 질의를 처리한 성능 평가 결과를 기술한다. <표 3>은 본 실험에 사용된 질의를 나타낸다. Q1-Q20의 각 질의를 대상으로 SQ, D, 그리고 PFT로 분할하였다. SQ에서는 조각 크기의 상한을 20KB로 설정하였다. 이는 D나 PFT로 분할된 조각의 최대 크기가 약 20KB이었기 때문이다. (그림 24)는 각 분할에 대해 질의 처리에 걸린 시간을 질의별로 나타낸 그래프이다 (조각 전송 순서는 preorder). 실험 결과를 보면 본 논문의 SQ가 D 및 PFT에 비해 20가지 질의 모두 처리 시간 면에서 우수한 성능을 보임을 알 수 있다. (그림 25)는 조각의 전송 순서가 bottom-up일 때의 처리 시간을 나타낸 그래프이다. SQ는 이러한 최악의 경우에도 다른 기법에 비해서 모든 질의에 대해 우수한 처리 성능을 보인다. 실제로 SQ의 경우, 조각의 전송 순서가 최선인 경우와 최악인 경우의 질의 처리 시간이 거의 동일하게 나타났다.

〈표 3〉 단일 질의를 대상으로 한 분할에 사용된 질의

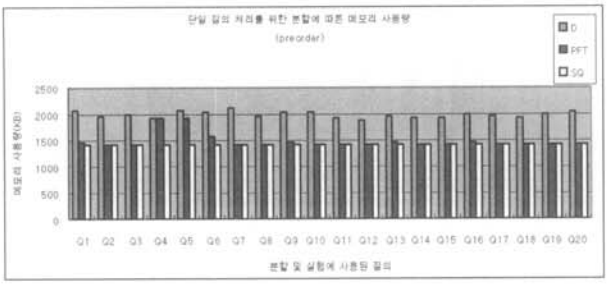
질의 번호	질의
Q1	/site/open_auctions/open_auction[initial>"10"]//increase
Q2	/site/open_auctions/open_auction/bidder[increase>"200"]
Q3	/site/open_auctions/open_auction[initial>"0"]/bidder
Q4	/site/open_auctions/open_auction[initial>"10"]//start
Q5	/site/open_auctions/open_auction[initial>"200"]/bidder/time
Q6	/site/open_auctions/open_auction/bidder[increase>"200"]/time
Q7	/site/open_auctions/open_auction[initial>"500"]/bidder[increase>"200"]/time
Q8	/site//increase
Q9	/site/people/person[name="Claudine Nunn"]/watches/watch
Q10	/site/people/person[name="Torkel Prodrodmidis"]/profile
Q11	/site/people/person/homepage
Q12	/site/closed_auctions/closed_auction[price>"100"]/type
Q13	/site/closed_auctions/closed_auction[price>"200"]/annotation/author
Q14	/site/closed_auctions/closed_auction[price>"40"]/itemref
Q15	/site/closed_auctions/closed_auction/buyer
Q16	/site/regions/africa/item[location="United States"]/mailbox/mail/from
Q17	/site/regions/namerica/item[payment="Creditcard"]
Q18	/site/regions/australia/item/description
Q19	/site/regions//item
Q20	/site//emailaddress



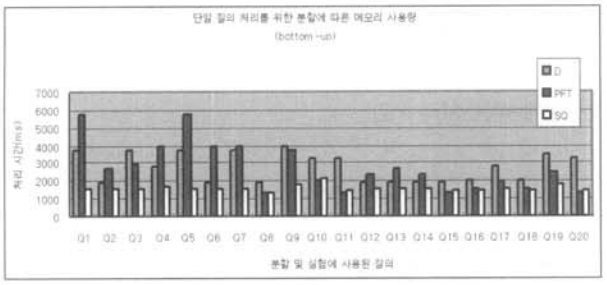
(그림 24) 단일 질의 처리를 위한 분할에 따른 처리 시간(preorder)



(그림 25) 단일 질의 처리를 위한 분할에 따른 처리 시간(bottom-up)



(그림 26) 단일 질의 처리를 위한 분할에 따른 메모리 사용량(preorder)



(그림 27) 단일 질의 처리를 위한 분할에 따른 메모리 사용량(bottom-up)

(그림 26)은 각 분할 기법에 따른 메모리 사용량을 질의별로 나타낸 그래프로 조각 전송 순서는 preorder이다. 실험 결과를 보면, SQ는 D보다 메모리 사용량이 적다. 반면 SQ와 PFT는 모든 질의에 대해 거의 동일한 메모리 사용량을 보이고 있다. D는 분할에 있어 질의를 고려하지 않는데 반해 SQ와 PFT는 질의를 고려하기 때문에 메모리 사용 효율이 차이가 났다. (그림 27)은 조각 전송 순서가 bottom-up일 때의 메모리 사용량을 나타낸 그래프이다. 실험 결과를 보면 조각이 bottom-up으로 전송되는 최악의 상황에서도 SQ는 모든 질의에 대해서 다른 기법에 비해 우수한 성능을 보이고 있다.

4.3 복수 질의 처리를 위한 XML 분할의 평가

본 절에서는 복수 질의를 대상으로 XML 문서를 분할하고 성능 실험한 결과를 설명한다. <표 3>에서 나타낸 질의들을 3가지의 서로 다른 질의 집합 1, 2, 3으로 구성하여, 각 질의 집합을 대상으로 성능 평가를 수행하였다. 실험에 사용된 질의 집합은 <표 4>-<표 6>과 같으며, 각각 SQ, D, 그리고 PFT로 분할을 수행하였다. 전절의 실험에서와 동일하게 SQ의 조각 크기 제한은 20KB로 설정하였다.

〈표 4〉 질의 집합 1

질의 번호	질의
Q1	/site/open_auctions/open_auction[initial>"10"]//increase
Q2	/site/open_auctions/open_auction[initial>"10"]//start
Q3	/site/open_auctions/open_auction[initial>"500"]/bidder[increase>"200"]/time
Q4	/site/people/person[name="Claudine Nunn"]/watches/watch
Q5	/site/closed_auctions/closed_auction[price>"100"]/type
Q6	/site/closed_auctions/closed_auction/buyer
Q7	/site/regions/australia/item/description

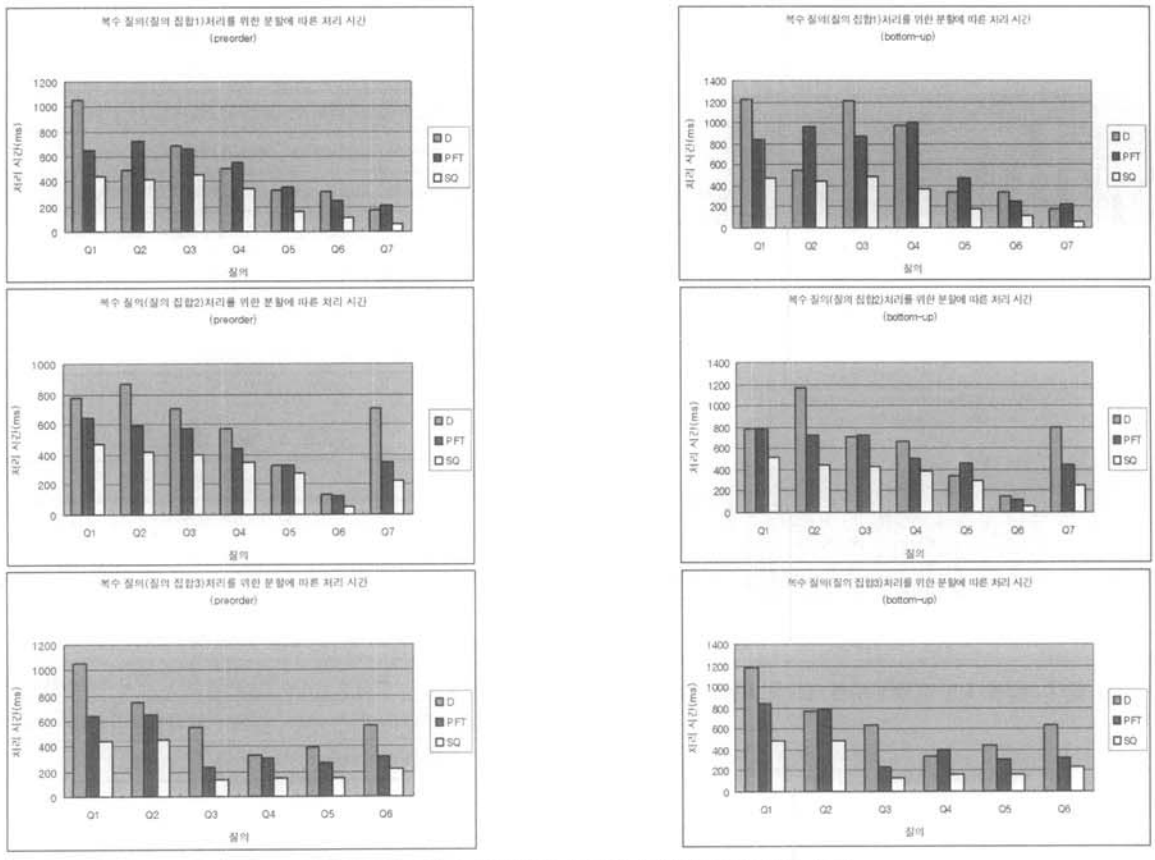
〈표 5〉 질의 집합 2

질의 번호	질의
Q1	/site/open_auctions/open_auction/bidder[increase>"200"]
Q2	/site/open_auctions/open_auction[initial>"200"]/bidder/time
Q3	/site//increase
Q4	/site/people/person[name="Torkel Prodrodmidis"]/profile
Q5	/site/closed_auctions/closed_auction[price>"200"]/annotation/author
Q6	/site/regions/africa/item[location="United States"]/mailbox/mail/from
Q7	/site/regions//item

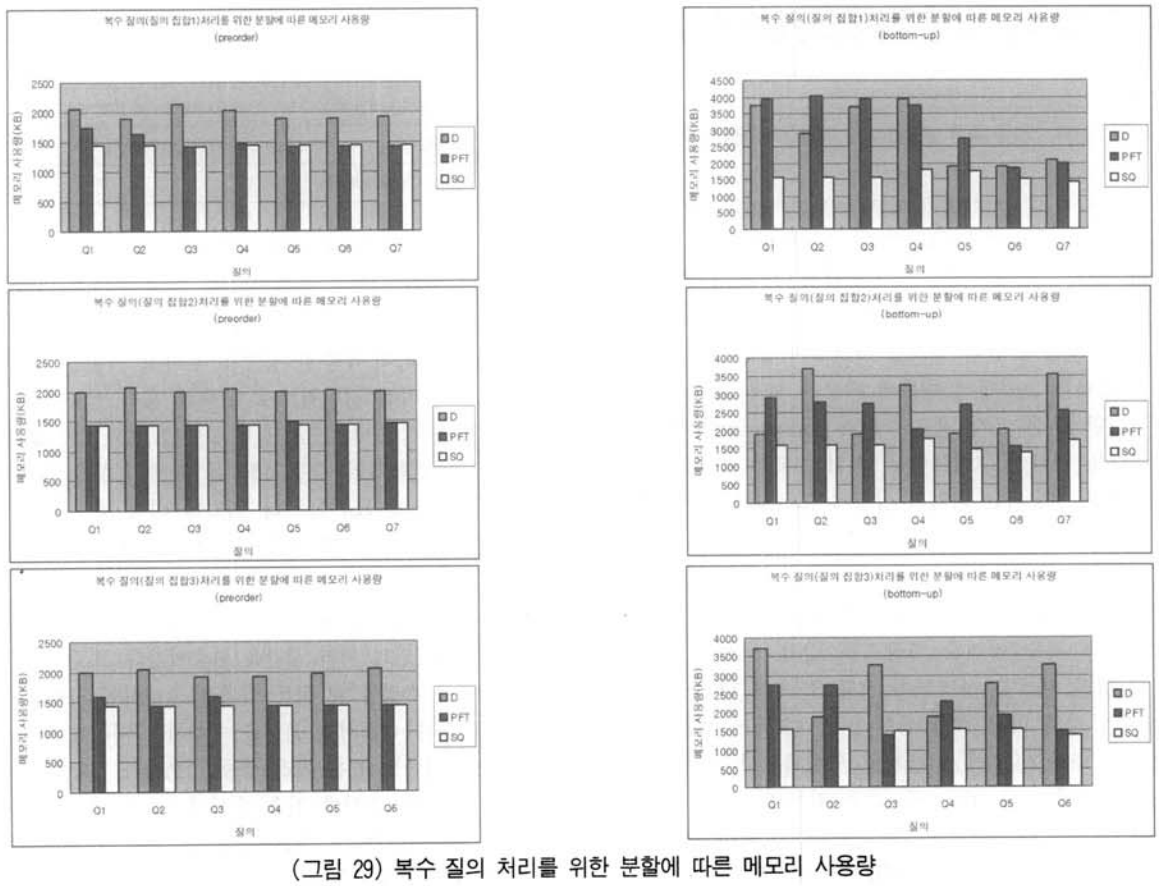
〈표 6〉 질의 집합 3

질의 번호	질의
Q1	/site/open_auctions/open_auction[initial>"0"]/bidder
Q2	/site/open_auctions/open_auction/bidder[increase>"200"]/time
Q3	/site/people/person/homepage
Q4	/site/closed_auctions/closed_auction[price>"40"]/itemref
Q5	/site/regions/namerica/item[payment="Creditcard"]
Q6	/site//emailaddress

(그림 28)의 왼편 세 그래프는 조각의 전송 순서가 preorder일 때, 각 질의 집합에 대한 질의별 처리 시간을 나타낸 것이다. 실험 결과를 보면 SQ가 세 종류의 질의 집합에 대해서 다른 기법에 비해 우수한 성능을 보이고 있음을 알 수 있다. PFT는 D에 비해 전반적으로 우수한 처리 성능을 보이지만, 그렇지 않은 질의(예: 질의 집합 1의 Q2, Q4, Q5, Q7)도 존재한다. 질의 집합 1의 경우, PFT를 통해 분할을 수행한 결과 조각의 개수는 51,857개로 타 기법에 비해 훨씬 많은 조각이 생성된다(D는 32289 개, SQ는 13,144개). SQ는 질의와 무관한 부분에 대해 가능한 큰 크기로 분할을 수행하여 헤더 검사 시간을 줄이고, 질의에 연관된 부분은 작은 크기로 분할하여 연산자 파이프라인의 처리 시간을 단축시킨다. 또한 특정 서브트리의 분할을 결정할 때 모든 질의를 고려하여 전체 성능이 좋아지는 경우에 대해서만 분할을 수행하기 때문에 복수 질의에 대한 분할 시에도 높은 처리 성능을 보였다. (그림 28)의 오른편 세 그래프는 조각의 전송 순서가 bottom-up일 때의 처리 시간을 나타낸 것이다. SQ는 bottom-up의 경우에서도 처리 성능이 가장 우수함을 볼 수 있다. bottom-up 조각 전송은



(그림 28) 복수 질의 처리를 위한 분할에 따른 처리 시간



(그림 29) 복수 질의 처리를 위한 분할에 따른 메모리 사용량

〈표 7〉 SQ의 기존 기법 대비 평균 처리 시간 단축율

분할 기법 / 질의 집합	D	PFT
질의 집합 1(preorder)	44%	42%
질의 집합 2(preorder)	46%	28%
질의 집합 3(preorder)	53%	31%
질의 집합 1(bottomup)	56%	54%
질의 집합 2(bottomup)	48%	37%
질의 집합 3(bottomup)	59%	41%

〈표 8〉 SQ의 기존 기법 대비 평균 메모리 사용량 감소율

분할 기법 / 질의 집합	D	PFT
질의 집합 1(preorder)	27%	4%
질의 집합 2(preorder)	28%	0%
질의 집합 3(preorder)	28%	2%
질의 집합 1(bottomup)	44%	50%
질의 집합 2(bottomup)	39%	35%
질의 집합 3(bottomup)	45%	29%

앞서 설명하였듯이 조각 스트림 처리에서 최악의 상황으로 볼 수 있다. 타 기법들이 bottom-up에서 처리 시간이 크게 증가하는 것에 비해, SQ는 처리 시간 증가가 크지 않다.

(그림 29)는 동일한 실험에 대한 메모리 사용량을 나타낸 것이다. 조각의 전송 순서는 preorder일 때 단일 질의를 위한 분할 실험과 마찬가지로 SQ와 PTF는 모든 질의에 대해서 거의 동일한 메모리 사용량을 보이고 D보다 우수함을 알 수 있다. 조각의 전송 순서가 bottom-up일 때 SQ는 조각의 전송 순서가 bottom-up인 경우 타 기법에 비해 특히 우수한 성능을 보였다. 특히 주목할 것은 PFT는 bottom-up으로 조각이 전송될 때 메모리 사용량이 현저히 증가하는 것에 비해 SQ는 메모리 사용량의 증가가 크지 않다는 것이다.

<표 7>과 <표 8>은 SQ의 D 및 PFT 대비 성능 향상 평균값을 질의 집합 및 조각 전송 순서 별로 요약한 것이다.

5. 결 론

XML이 웹에서 데이터 교환의 표준으로 부각되면서 유비쿼터스 컴퓨팅 환경에서도 효율적인 XML 데이터 처리에 대한 연구가 활발히 진행되고 있다. 이동 디바이스와 같은 클라이언트의 제약된 저장 공간 및 컴퓨팅 파워를 이용하여 대량의 XML 데이터에 대한 질의 처리를 수행하기 위해서는 질의 처리기에서의 처리 시간 효율성 및 메모리 효율성이 요구된다. 최근에는 XML 문서를 조각으로 분할하여 스트리밍하고, 클라이언트는 이들 분할된 XML 조각 스트림을 받아 질의를 처리하는 기법들이 제시되었다.

본 논문에서는 XML 조각 스트림에 대한 질의 처리 시 효율적인 자원 사용을 위한 XML 문서 분할 기법을 제안하였다. XML 문서를 분할하는 시점에서 질의 처리기의 처리 시간 및 메모리 효율성을 고려하기 위해 XML 조각 스트림 처리기가 질의를 처리하는 과정을 분석하여 질의 처리 비용 모델을 제시하였다. 본 비용 모델을 기반으로 사용자의 질의를 고려하여 XML 문서를 분할하는 알고리즘을 제시하였다. 구현 및 실험을 통해 본 논문에서 제시하는 기법이 기존의 기법들에 비해 처리 시간 및 메모리 효율성 양면 모두에서 우수함을 확인하였다.

향후 연구 과제는 다음과 같다. 첫째, 이동 디바이스의 에너지 효율성을 제고하기 위한 연구가 필요하다. 에너지 절약은 이동 디바이스가 활동 모드와 수면 모드 간에 적절히 전환하도록 함으로써 가능하다. 이를 위해서 XML 조각 스트림의 인덱싱에 대한 연구가 필요하다. 둘째, 이동 디바이스의 평균 질의 응답 시간을 단축시키기 위한 연구가 필요하다. 이동 디바이스의 평균 질의 응답 시간은 스트리밍되는 XML 데이터의 양에 의존적이다. 그러나 XML 문서 내의 서로 다른 경로들 중 hot 데이터에 해당되는 것들을 포함하는 XML 조각들을 그렇지 않은 cold 데이터와 구분하여 스트리밍시키는 빈도에 차이를 둠으로써 질의들의 평균 응답 시간을 줄일 수 있다. 단, cold 데이터에 대한 질의 처리 시간이 상대적으로 증가하는 정도에 상한을 설정할 필요가 있는데 이러한 제약 하에서 질의들의 평균 응답 시간을 단축하기 위한 기법에 대한 연구가 필요하다.

참 고 문 헌

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

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

[3] S. Bose, L. Fegaras, "XFrag: A Query Processing Framework for Fragmented XML Data," Proc. WebDB, pp.97-102, 2005.

[4] H. Huo, G. Wang, X. Hui, R. Zhou, B. Ning, C. Xiao, "Efficient Query Processing for Streamed XML Fragments," Lecture Notes in Computer Science(LNCS) 3882, DASFAA, pp.468-482, 2006.

[5] 이상욱, 김진, 강현철, "XML 레이블링을 이용한 XML 조각 스트림에 대한 질의 처리 기법," 정보과학회논문지 : 데이타베이스, 제35권, 제1호, pp.67-83, 2008년 2월.

[6] 이상욱, 김진, 강현철, "동적 XML 조각 스트림에 대한 메모리 효율적 질의 처리," 정보처리학회논문지D, 제15-D권, 제1호, pp.1-14, 2008년 2월.

[7] J.M. Bremer, M. Gertz, "On Distributing XML Repositories," Proc. WebDB, pp.73-38, 2003.

[8] H. Ma, K. D. Schewe, "Fragmentation of XML Documents," Proc. SBBD, pp.200-214, 2003.

[9] A. Andrade, G. Ruberg, F. Baiao, V. P. Braganholo, M. Mattoso, "Efficiently Processing XML Queries over Fragmented Repositories with PartiX," Proc. EDBT, pp.150-163, 2006.

[10] A. Bonifati, A. Cuzzocrea, "Efficient Fragmentation of

Large XML Documents," Proc. DEXA, pp.539-550, 2007.
[11] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda, "Lazy Query Evaluation for Active XML," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2004.S. Hartmann, H. Ma, K. D. Schewe, "Cost-Based Vertical Fragmentation for XML," Proc. APWeb/WAIM, pp.12-14, 2007.
[12] C. Li, T. W. Ling, "QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates," Proc. CIKM, pp.501-508, 2005.
[13] P. O'Neil, E. Oneil, S. Pal, I. Cseri, G. Schaller, N. Westbury, "ORDPATHs: Insert-Friendly XML Node Labels," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp. 903-908, 2004.
[14] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2002, pp.204-215.E. Y. C. Wong, A. T. S. Chan, H. V. Leong, "Efficient management of XML contents over wireless environment by Xstream," Proc. SAC, pp.1122-1127, 2004.
[15] E. Y. C. Wong, A. T. S. Chan, H. V. Leong, "Efficient management of XML contents over wireless environment by Xstream," Proc. SAC, pp.1122-1127, 2004.
[16] H. Huo, G. Wang, X. Hui, C. Xiao, R. Zhou, "Document Fragmentation for XML Streams Based on Query Statistics," Lecture Notes in Computer Science(LNCS) 4255, WISE, pp.350-356, 2006.
[17] L. Mignet, D. Barbosa, P. Veltri, "The XML Web: a First Study," WWW 2003, pp.500-510.
[18] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, R. Busse, "XMark: A Benchmark for XML Data Management," Proc. Int'l Conf. on VLDB, pp.974-985, 2002.

김 진

e-mail : jkim@dblab.cse.cau.ac.kr
2006년 중앙대학교 컴퓨터공학과(학사)
2008년 중앙대학교 컴퓨터공학과(공학석사)
관심분야 : XML 스트림 데이터 처리, 웹 데이터베이스 등

강 현 철

e-mail : hckang@cau.ac.kr
1983년 서울대학교 컴퓨터공학과(공학사)
1985년 U. of Maryland at College Park, Computer Science(M.S.)
1987년 U. of Maryland at College Park, Computer Science(Ph.D.)
1988년~현 재 중앙대학교 컴퓨터공학부 교수
관심분야 : XML 및 웹 데이터베이스, Stream 데이터 관리, 센서네트워크 데이터베이스 등