유비쿼터스 환경에서 실시간 센서 데이터를 위한 XML 질의언어 처리 엔진

An XQuery Processing Engine for Real-Time Sensor Data in Ubiquitous Environments

임형준(Hyung-Jun Yim)*, 김재훈(Jae-Hoon Kim)**, 이규철(Kyu-Chul Lee)***

초 록

최근에 유비쿼터스나 전자상거래와 같은 환경에서 발생하는 실시간 데이터를 처리해야 하는 요구가 늘어나고 있다. 유비쿼터스 환경에서 사용되는 센서 데이터는 그 크기가 작고 XML 문서로 표현 되어 있으며 대량으로 발생하는 특징이 있다. 이러한 대량의 센서 데이터를 처리하기 위한 효율적인 방법이 요구되고 있다. 센서 데이터에 대한 XML 질의언어(XQuery)는 주로 센서 데이터가 발생한 센서의 식별자나 표현하고자 하는 정보를 얻어오는 것과 사용자의 편의를 위한 결과 재구성으로 분류된다. 기존의 XML 질의언어 엔진들은 센서 데이터를 일괄적으로 처리하기 위한 효율적인 방법을 가지고 있지 않다.

본 논문에서는 대량의 센서 데이터들을 실시간으로 처리하기 위하여 역 경로 요약을 이용한 가지 질의(Twig Query) 처리 기법을 적용하였다. 또한, 재구성(Restructuring) 일괄 처리 기법을 개발하여 적용하였다. XMark와 RFID EPC 데이터를 이용한 성능 측정을 하고 MonetDB/XQuery와 Oracle Berkeley DB XML과의 비교 분석을 수행하였다.

ABSTRACT

Recently, it is necessary to process real time sensor data, which is generated from ubiquitous environments. Data, which are written by XML, are small, but, large volumes of data. Therefore, weneed to use an efficient method for processing a large amount of it. An XQuery has two types for sensor data: one is to get sensor identification and value from sensor data; the other is restructuring for user's convenience. Existing XQuery engines don't have efficient method for batch processing of sensor data.

This paper proposed the twig query processing over reverse path summary, and we developed and applied restructuring batch processing method for real time processing of a large amount of sensor data. Finally, we do performance evaluation using XMark and RFID EPC data, and comparison analysis with MonetDB/XQuery and Berkeley DB XML.

이 논문은 2008년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2008-521-D00373).

* 충남대학교 컴퓨터공학과

** 한국과학기술정보연구원

*** 교신저자, 충남대학교 컴퓨터공학과

2010년 09월 27일 접수, 2010년 10월 17일 심사완료 후 2010년 11월 05일 게재확정.

키워드 : XML 질의언어, XML DBMS, 역경로 요약, 가지 질의, 재구성, 센서 데이터
XQuery, XML DBMS, Reverse Path Summary, Twig Query, Reconstruction, Sensor Data

1. 서 론

최근 들어 유비쿼터스 환경에서는 RFID, ZigBee 등의 실시간 센서 데이터가 많이 사용됨에 따라 센서 데이터 처리에 대한 요구가 증가하고 있다. 또한, 이미 많은 전자상거래 시스템에서 메시지 송수신을 센서 데이터와 유사한 특징을 지닌 XML 문서를 이용하고 있다. 예를 들어, 군사 영역 실시간 모니터링이나 기상 정보를 수집하기 위해 넓은 영역에 뿌려 놓은 RFID 센서 데이터들의 정보를 확인하기 위해 XML로 표현된 EPC(Electronic Product Code) 데이터를 전송하도록 되어 있다. EPC 데이터는 <표 1>과 같이 PML Core (Physical Markup Language)[3]를 따른다. <표 1>은 온도 센서에서 생성되는 데이터의 예이다. 이러한 센서 데이터의 특징은 <표 2>와 같이 크기가 작고 많은 수가 생성되기 때문에 평균 초당 12,000건 처리가 필요하며, 저장된 데이터에 대한 질의를 실시간으로 처리하거나 분석해야 한다[1].

센서 데이터의 특징은 하나의 센서에서 들어오는 실시간 센서 데이터를 처리하는 것보다는 수많은 센서 데이터들로부터 수집된 데이터들을 빠르게 분석하는 것이 중요하다. 또한, 센서에서 발생하는 센서 데이터는 크기가 작고 동일한 스키마를 가지고 있다. 이는 센서의 데이터가 동일한 구조로 표현되어 있기 때문에 들어오는 데이터의 구조를 파악하기보다 동일한 스키마의 데이터 내용을 처리해야 한다.

따라서, 본 논문에서는 수많은 센서에서 들어오는 데이터를 분석하고 처리하기 위해 Main-Memory DBMS를 사용하고 XML 질의언어(XQuery) 처리 엔진의 개발을 통해 타 시스템과 비교한 내용을 다룬다.

〈표 1〉 온도 센서 데이터의 예
'sensorData.xml'

```
<Sensor>
 <ID>urn : epc : 1 : 124.162.37</ID>
 <Observation>
   <DateTime>2002-11-06T13 : 04 : 34-06 : 00
</DateTime>
   <Data>
    <XML>
      <TemperatureReading
xmlns =“http://sensor.example.org/”>
       <Unit>Celsius</Unit>
       <Value>17.5</Value>
      </ TemperatureReading >
   </XML>
  </Data>
 </Observation>
</Sensor>
```

〈표 2〉 RFID 태그의 종류와 XML 데이터의 크기

형태	라벨형	원반형	카드형	박스형
크기	60*10*t0.25	8*t5	54*86*t0.76	63*98*t12
데이터크기	~606B	~254B	~1KB	~0.5KB

이러한 과정을 센서 데이터의 XML 스트림 처리의 일종으로 센서 데이터의 스트림 처리를 위해서는 인바운드(Inbound) 처리 방식과 데이터를 DBMS에 저장한 후 처리하는 아웃

바운드(Outbound) 방식으로 나눌 수 있다[6]. 인바운드 처리 방식은 스트림으로 들어오는 데이터에 대한 질의를 한 후 저장하는 것과 달리 아웃바운드 방식은 들어오는 스트림 데이터를 저장한 후 처리 및 질의를 한다. [6]에서 언급한 바와 같이 보통의 스트림 데이터 처리는 인바운드의 스트림 엔진에서 처리하는 것이 효과적으로 나타났지만, 본 논문에서는 실시간 센서 데이터의 분석적 관점에서 접근하여 DBMS의 기능에 초점을 맞추고 있으며 XML로 표현된 데이터에 대한 저장 구조와 질의 최적화 방법 등을 다루기 때문에 현재의 XML 질의언어 처리 엔진들과 비교 평가를 하고 추후 인바운드 방식과 비교가 필요할 것이다. 또한, 분석적 관점 이외에도 실시간으로 데이터를 처리하기 위해 병목현상을 줄이기 위한 방법으로 Main-Memory DBMS를 사용하여 대응할 수 있도록 한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 본 논문과 관련된 연구에 대해 살펴보고, 본 논문에서 제안하는 UbiDB/XQuery 엔진의 설계에 대한 내용을 제 3장에서 다룬다. 제 4장에서는 설계를 바탕으로 구현한 내용을 서술하고, 제 5장에서는 설계와 구현이 제대로 되었는지 확인하기 위해 성능측정을 하고 그에 대한 분석을 한다. 마지막으로 제6장에서는 본 논문의 결론과 향후 연구 계획에 대해 논한다.

2. 관련 연구

본 장에서는 XML 질의언어의 예를 살펴보고, 이를 처리하는 과정을 통해 본 시스템과의 비교대상을 선정하는 내용을 다룬다.

2.1 XML 질의언어(XQuery)

XML 질의언어는 XML 질의를 위한 새로운 W3C 표준으로 정의되었고, 최초의 XML 질의어인 Quilt에서 유래 되었으며, 이미 W3C 표준인 XSLT, XPath, XML 등이 가지고 있는 몇 가지 유용한 기능들이 추가된 것이다. XML 질의언어의 특징은 XML 데이터의 저장 형식에 무관하게 질의가 가능하며, 간결하고, 쉽게 이해되도록 설계되었다. 그리고 W3C의 다른 표준안들과 호환성을 유지한다.

<표 3>은 실제로 사용되는 온도 센서데이터인 <표 1>의 'sensorData.xml'에서 온도의 값(Value)가 17도 보다 높은 경우의 ID와 발생시간, 온도결과를 재구성하는 형태의 XML 질의언어 예이다.

〈표 3〉 XML 질의언어의 예

XQuery

```
let $sensor := doc('sensorData.xml') return
for $b in $sensor/Sensor
where $b//Data/XML/Read/Value/text() > 17
return <Alert>
<ID> $b/ID/text() </ID>
<AlertTime>$b/Observation/DateTime/text()</AlertTime>
<Celsius>$b/Observation/Data/XML/Read/Value/text() </Celsius>
 </Alert>
```

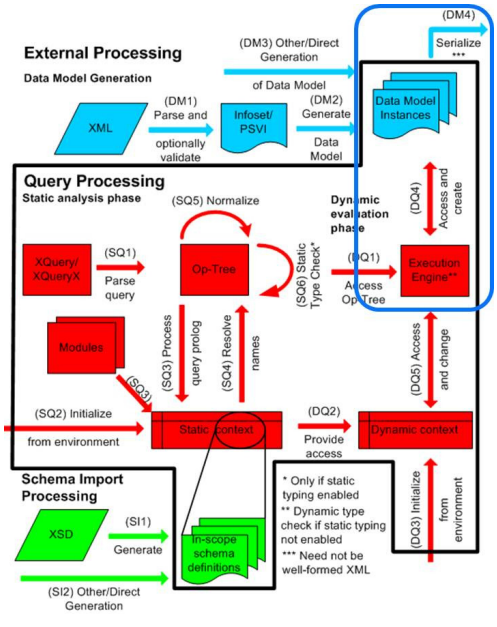
Result

```
<Alert>
   <ID> urn:epc:1:124.162.37</ID>
   <AlertTime> 2002-11-06T13:04:34-06:00 </AlertTime>
   <Celsius> 17.5 </Celsius>
</Alert>
```

2.2 XML 질의언어 처리과정

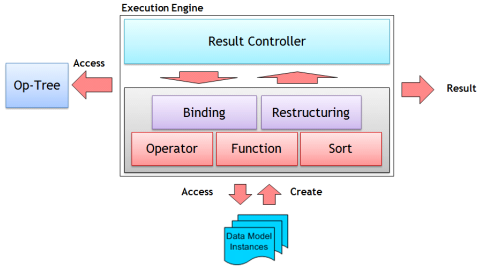
XML 질의언어 처리 과정은 <그림 1>과 같이 3가지로 구성된다. XML 문서를 데이터 모

델로 생성하는 과정(외부 처리-External Processing의 XML, Infoset/PSVI와 Data Model Instances 부분), 스키마 문서에서 정의를 추출하여 생성하는 과정(XML 스키마 처리-Schema Import Processing의 XSD와 In-scope Schema Definitions 부분), XML 질의언어를 파싱(parsing)하여 질의 계획 트리(Op-Tree)를 생성하고 이를 기준으로 데이터 모델에 접근하여 결과를 생성하는 과정(질의 처리-Query Processing의 XQuery/XQueryX, Modules, Op-Tree, Static Context, Execution Engine과 Dynamic Context 부분)이다 [8].



〈그림 1〉 XML 질의언어 처리 모델 개요[8]

본 논문에서는 <그림 1>의 오른쪽 위에 네모 박스로 표시된 부분인 질의 계획 트리를 통해 얻은 정보를 이용하여 XML 데이터 모델 인스턴스(XML Data Model Instance)에 접근하고 원하는 결과를 생성하는 실행 엔진(Execution Engine) 부분을 향상시켰다. 이 부분의 향상은 수많은 센서에서 들어오는 XML 스트림 데이터를 실시간으로 처리하고 분석에 효과적인 영향을 준다. <그림 2>는 향상시킨 부분을 도식화한 것으로 XML 질의언어 실행 엔진을 나타낸다.



〈그림 2〉 XML 질의언어 실행 엔진

XML 질의언어 실행 엔진은 본 논문에서 중요한 부분으로 연결(Binding), 재구성(Restructuring), 연산자(Operator), 함수(Function), 정렬(Sorting)과 같은 XML 질의언어 처리를 담당한다. 질의 계획 트리에서 필요한 정보를 얻어 와서 데이터 모델 인스턴스(본 논문에서는 DB 테이블)에 접근하여 원하는 결과를 생성하는데 이는 각각의 처리 모듈에서 수행하여 생성된 결과를 구성하여 사용자에게 보여주게 된다. 이에 대한 내용은 본론에서 자세히 다루도록 한다.

2.3 XML 질의언어 엔진의 분류

기존의 XML DBMS는 XML의 특성을 지원하여 저장 관리하는 Native XML DBMS

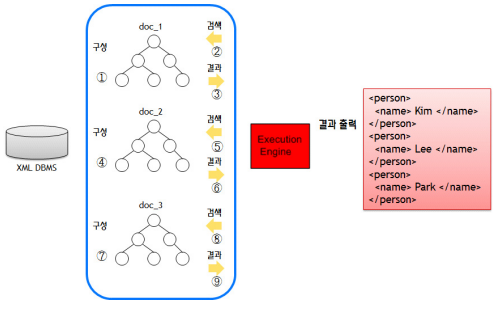
와 XML 문서를 관계형 데이터베이스에 매핑(Mapping)하여 저장 관리하는 XML-Enabled DBMS로 크게 나눌 수 있다[7].

Native XML DBMS는 XML의 특성에 맞는 처리 및 저장구조를 사용하기는 하나, 디스크(Disk) 기반의 DBMS이기 때문에 실시간 처리에는 한계를 가지게 된다. XML-Enabled DBMS는 XML 문서를 관계형 데이터베이스로 매핑하는 과정에서 XML 문서의 구조 정보를 손실 시킬 수 있어 효율성의 문제가 발생한다. 이러한 한계점을 극복하려면 XML 처리를 실시간으로 지원할 수 있는 메커니즘을 가진 Main-Memory Native XML DBMS의 개발이 필요하다. 하지만, 유비쿼터스 환경의 동일한 스키마에서 생성되는 수많은 센서 데이터를 처리하기 위해서는 Native XML DBMS의 XML 문서를 모두 DOM으로 변환해야 하는 문제가 있기 때문에 이 환경에서는 XML-Enabled DBMS가 적합할 것으로 보고 본 논문의 성능평가를 통해 확인하였다. 또한 대표적인 XML 질의언어의 벤치마크(Benchmark)인 XMark(A Benchmark for XML Data Management)[2]를 수행하여 구조적 문제가 발생하지 않음을 확인한다.

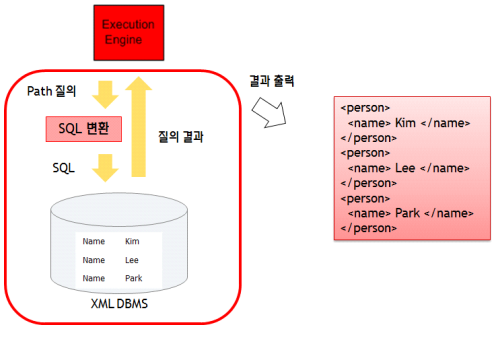
앞서 살펴 본 XML 질의언어 처리 과정 중 본 논문에서 향상 시킨 실행 엔진은 XML 데이터 모델에 접근하여 결과를 생성하는 과정이다. 다시 말하면, 저장된 XML 문서를 XML 데이터 모델로 변환하면 XML 질의언어를 통해 결과를 얻어 오는 부분이다. 이는 XML 문서가 저장된 DBMS에서 질의를 하기 위해 XML 문서를 구성하게 되는데 XML 문서를 저장하는 방법에 따라 구분된다.

XML 문서 단위로 DOM(Document Object Model)을 구성하는 방법으로 보통 Native XML Database에서 사용하고 XML 문서를 태그 또는 임의 단위로 분할하여 DBMS에 저장하는 방법(Shredded XML in DBMS)은 XML-Enabled Database에서 사용하는 것으로 나뉜다.

XML 문서 단위로 DOM을 구성하는 방법은 XML 질의언어를 처리할 경우 해당되는 모든 XML 문서를 DOM으로 변환하여 결과를 얻는 것으로 Oracle Berkeley DB XML, X-Hive/DB, Qizx/DB 등이 있으며 <그림 3>으로 도식화 할 수 있다.



〈그림 3〉 XML 문서 단위로 DOM을 구성하는 방법



〈그림 4〉 Shredded XML in DBMS

XML 문서를 태그 또는 임의 단위로 분할하여 DBMS에 저장하는 방법을 사용하면 XML

질의언어를 처리할 경우 XML 문서 단위의 데이터 모델을 생성하지 않고 테이블을 검색하여 결과를 일괄적으로 얻어올 수 있다. 일괄적으로 결과를 얻어올 수 있지만, 이를 위해 테이블에 질의를 해야 하기 때문에 XML 질의언어를 SQL로 변환하는 추가 작업이 필요하다. 이러한 방법을 따르는 XML 질의언어 엔진은 <그림 4>와 같이 도식화 할 수 있으며 대표적인 제품으로는 MonetDB/XQuery가 있고, 본 논문의 엔진인 UbiDB/XQuery가 해당된다.

본 논문에서 사용하고 있는 XML 문서를 태그 또는 임의 단위로 분할하여 DBMS에 저장하여 일괄적으로 XML 질의언어를 처리하기 때문에 XML 문서 단위로 DOM을 구성하여 질의하는 방법을 사용하는 Oracle Berkeley DB XML을 비교대상으로 선정하였다. 이는 본 논문에서 사용하는 방법은 XML 질의언어를 SQL로 변환하여 결과를 얻어 오는 방법이 XML 문서마다 DOM으로 구성하여 질의를 하는 방법과 성능의 영향을 확인하기 위함이다. 또한, 본 논문과 동일한 방법을 사용하는 MonetDB/XQuery도 비교대상에 포함하여 동일한 스키마를 구성하는 수많은 XML 문서에 대해 실시간 처리의 우월함을 확인하는 목적을 가진다.

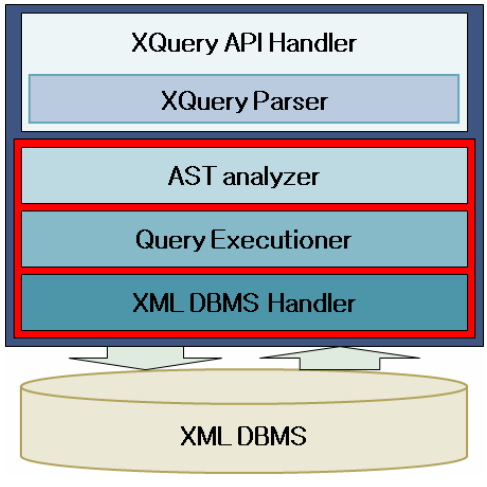
3. UbiDB/XQuery 설계

본 장에서는 XML 문서를 태그 또는 임의 단위로 분할하여 DBMS에 저장하는 방법에 대해 서술한다. 이를 위해 XML 질의언어를 처리하기 위한 역 경로 요약(Reverse Path Summary)[4]를 적용하고 가지 질의(Twig Query)와 재구성(Restructuring) 일괄처리 기법을 통해 설계한 내용을 다룬다.

3.1 UbiDB/XQuery 엔진 구조

본 연구에서 개발한 UbiDB/XQuery의 구조는 <그림 5>와 같다.

기본적인 구조는 XQilla[11]를 수정하여 XQuery 엔진을 구현하였다. 기존 XQilla는 파일을 입력 받아 XML 문서단위로 DOM을 구성하기 때문에 본 연구에서는 XQilla의 XML 질의언어 파싱 부분만을 사용하고 XML 인스턴스는 XML DBMS에서 얻어오게 된다. XML DBMS는 본 연구실에서 개발한 FastDB[5] 기반의 XML DBMS를 사용한다.



〈그림 5〉 UbiDB/XQuery 구조

XML 질의언어의 파서(Parser)를 포함한 XQuery API는 XQilla를 사용하고 있으며 AST 분석기(Abstract Syntax Tree analyzer), 질의 실행기(Query Executioner), XML

DBMS 핸들러(Handler)를 추가하여 대량의 데이터에 대해 효율적인 XML 질의언어의 처리를 가능하게 하였다.

이와 같이 개발한 부분을 세 부분으로 분리함으로써 XML DBMS와 XML 질의언어 엔진의 독립성을 유지하였고 수정이 용이하게 하였다.

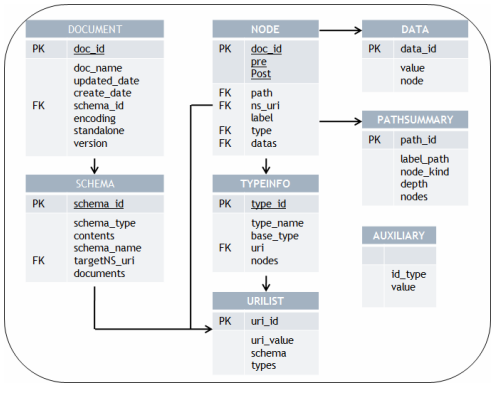
AST 분석기는 XML 질의언어 파서를 거친 XML 질의언어가 AST로 변환이 되는데 이를 분석하여 필요한 정보를 얻어오는 모듈이다. AST는 XML 질의언어 처리 과정에서 질의 계획 트리와 같은데 XQilla에서는 AST 클래스(Class)로 유지 한다.

질의 실행기는 AST 분석기를 통해 얻어온 정보를 가지고 경로 질의를 역 경로 요약으로 표현하여 SQL에 조건으로 삽입하는 방법으로 변환하고, SQL을 XML DBMS 핸들러에 전달한다. 또한 SQL 질의의 결과 값을 저장하고 이를 정제하는 일을 한다. 질의 실행기의 가장 핵심적인 기법은 재구성 일괄처리로써 결과를 저장하는 결과리스트(Result List) 구조체를 이용하여 구현하였다. 결과리스트 구조체는 NODE 테이블의 정보(doc_id, pre, post, label) 또는 DATA 테이블의 정보(value)를 가지고 있다.

XML DBMS 핸들러는 XML DBMS에 SQL 질의를 하고 결과 값을 결과리스트에 저장하여 질의 실행기에 넘겨주는 역할을 한다.

3.2 XML DBMS 스키마

XML DBMS의 스키마와 사용된 방식은 대량의 문서에 대한 SQL 질의 처리 시 효율적일 수 있도록 고안되었다. <그림 6>은 XML DBMS의 스키마이다.



〈그림 6〉 XML DBMS 스키마

각 노드의 경로는 PATHSUMMARY 테이블에 저장이 되는데 여기서 사용되는 방법이 역 경로 요약이다. 역 경로 요약은 경로 질의 표현식을 가지고 효율적인 질의 처리를 위해 경로 질의를 효과적으로 요약하여 나타내는 표현법이다. 역 경로 요약이라고 말하는 것은 경로 질의에서 각 단계의 경로를 처리할 때 앞에서부터 처리하는 순 경로 요약이 아니라 뒤에서부터 각 단계의 경로를 처리하기에 때문이다. 이를 쓰는 이유는 경로정보를 역 경로 요약 표현식으로 변환하여 표현함으로써 질의 대상 후보군을 줄이고 질의 효율을 높이기 때문이다.

<표 4>와 <표 5>는 <표 1>의 'sensor Data.xml'을 순 경로 요약과 역 경로 요약의 예를 보여주고 있다.

Read#&XML#&Data#&Observation#&Sensor에 대해 질의를 하게 되면 순 경로 요약은 모든 행에 대해 끝까지 경로를 비교하지만 역 경로 요약은 첫 노드 경로만을 비교하여

Read가 먼저 나오는 다섯 번째 행을 효율적으로 찾게 된다. 이는 데이터베이스 질의를 통해 결과 값을 얻어오는 시간을 줄일 수 있게 된다. 여기에 사용된 “#, &” 기호는 SQL에서 사용하지 않는 기호로써 노드를 구분하기 위하여 쓴다.

NODE 테이블에 노드 정보를 저장 할 때 pre, post라는 열이 있는데 이는 노드들 간의 관계를 정의하기 위한 넘버링 기법으로 Pre and Post Guard[10]을 사용하고 있기 때문이다. Pre and Post Guard 넘버링은 결과들 간의 구조적 조인(Structural Join)이 가능하게 해준다.

〈표 4〉 순 경로 요약의 예

Sensor&
Sensor&#Observation&
Sensor&#Observation&#Data&
Sensor&#Observation&#Data&#XML&
Sensor&#Observation&#Data&#XML&#Read&

〈표 5〉 역 경로 요약의 예

Sensor
Observation#&Sensor
Data#&Observation#&Sensor
XML#&Data#&Observation#&Sensor
Read#&XML#&Data#&Observation#&Sensor

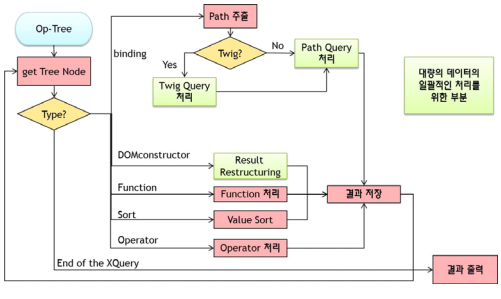
4. UbiDB/XQuery 구현

본 장에서는 UbiDB/XQuery의 동작과정과 제 3장에서 설계한 내용에 대한 구현 내용을 다룬다.

4.1 UbiDB/XQuery 동작과정

UbiDB/XQuery의 동작 과정은 <그림 7>과 같다. 모든 동작은 질의 계획 트리에서 동작 타입을 얻어오는 것에서 시작된다. 각각의 동작은 크게 Binding, DOMConstructor, Function, Sort, Operator로 나뉜다. Binding을 통해 변수에 결과 값을 대입하고 DOMConstructor, Function, Sort, Operator에서 그 변수를 정제하여 원하는 결과값을 추출하게 된다. 본 연구에서 대량의 데이터를 일괄적으로 처리하기 위해 노력한 부분은 <그림 7>에 표시된 가지 질의[12], 경로 질의(Path Query)[9]와 결과 재구성(Result Restructuring) 부분이다.

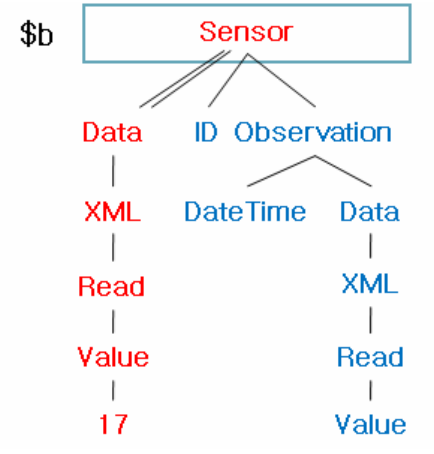
<표 3>의 XML 질의언어의 예에서 <그림 8>과 같이 경로를 추출한다. 질의 계획 트리에서 <Step>태그를 통해 추출하며 <Variable> 태그를 통해 변수를 대입한다. 경로를 추출한 후에는 가지 질의 여부를 확인 한 뒤 <그림 9>와 같이 SQL 질의로 변환한다.



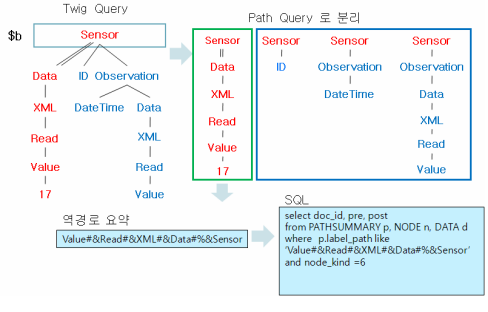
〈그림 7〉 UbiDB/XQuery 동작과정

가지 질의는 하나 이상의 분기를 가지고 있는 질의를 나타낼 때 사용되는 표현법이다. <그림 10>과 같은 경우가 가지 질의인데 이는 경로 질의로 분리를 한다. 각각의 경로 질

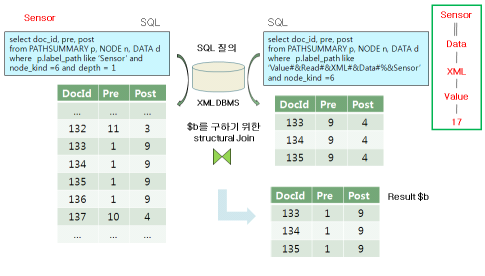
의는 역 경로 요약으로 변환하고 이는 SQL 생성에서 PATHSUMMARY 테이블의 조건으로 사용된다.



〈그림 8〉 XML 질의언어의 경로 추출



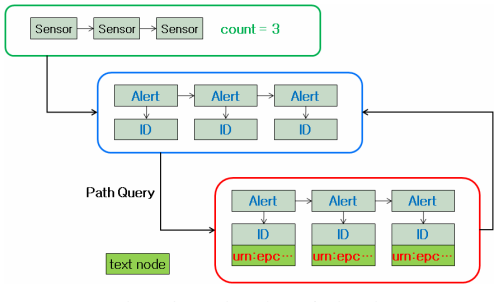
〈그림 9〉 가지 질의의 SQL 변환



〈그림 10〉 SQL 질의 결과 처리

<표 3>에서 보듯이 실제 변수 b에 해당하는 값은 'Value#&Read#&XML#&Data#%&Sensor'가 아니라 'Sensor'이다. 이 'Sensor' 값을 구하기 위해' Value#&Read#&XML#&Data#%&Sensor'와 'Sensor'의 pre, post 값을 이용한 구조적 조인을 수행하여 조건에 맞는 'Sensor' 값을 구하게 된다. 여기서 나온 결과 값은 변수 b에 대입되게 된다. 여기서 대입된 변수 b는 나머지 경로 질의 에서도 마찬가지로 Structural Join의 값으로 쓰여 원하는 결과 값을 얻을 수 있다. 이는 <그림 10>에 표현되어 있다.

<표 3>의 Binding을 수행 한 후에는 Return 절을 처리 하게 되는데 이는 재구성을 수행하기 때문이다. 기존에는 재구성 중에 경로 질의가 나타나면 질의 처리를 위해서 변수에 대입된 노드 수만큼 같은 질의를 반복해야 하는 문제가 있었다. 하지만 UbiDB/XQuery에서는 일괄적인 재구성을 위해 <그림 11>과 같은 방식을 고안하여 적용하였다.



〈그림 11〉 재구성의 예

먼저 바인딩된 변수의 결과 노드에 대한 수를 세게 되는데 이는 이 개수만큼 결과값이 생성될 것이다. 경로 질의가 아닌 XML 질의언어에서는 재구성을 위해 명시한 노드

가 나오면 앞서 세 놓았던 개수만큼 생성을 하여 결과 리스트에 일괄적으로 삽입한다. 만약 경로 질의가 나오면 해당 경로 질의를 수행하고 나온 결과 노드들을 결과 리스트에 일괄적으로 삽입하는 과정을 반복하게 된다.

기존에는 재구성을 처리하기 위해서는 '변수의 결과 노드 개수*경로 질의의 개수'만큼의 데이터베이스 질의가 필요 했지만 본 연구에서 고안한 일괄적 처리 기법을 사용하면 '경로 질의의 개수' 만큼의 데이터베이스 질의만이 필요하기 때문에 빠른 재구성이 가능하다.

<표 3>에서 등장하지 않은 Function, Operator는 XQuery 1.0 and XPath 2.0 Functions and Operator에 따라 중간 결과를 함수와 연산자에 따라 처리하고 정렬도 결과 값들을 결과리스트에 맞게 구현한 함수로 처리 한다.

본 논문에서 개발한 UbiDB/XQuery는 역경로 요약에 기반한 가지 질의를 사용하고 새로 개발한 재구성 일괄 처리 방식을 이용하여 기존의 XML DBMS/XQuery와의 차별성을 두었다.

5. 성능 평가

XML 질의언어 엔진에 대한 성능 평가는 질의 수행 시 필요한 소요 시간을 측정한다.

여기에서는 실제 유비쿼터스 환경에서 실시간 센서 데이터에 대해 효과적인 질의 처리가 가능한지를 알아보기 위해 대량의 RFID EPC데이터에 대해서 질의 수행 시간을 측정한다. 또한, XML 문서의 구조적인 문제가 없는지 확인하기 위해 XMark에서 제공하는 XML 문서와 XML 질의언어 20개에 대해서 질의 수행 시간을 측정한다.

본 연구에서 개발한 UbiDB/XQuery엔진과 성능 비교할 기존의 XQuery엔진은 다음과 같다.

- MonetDB/XQuery(v 0.26.0)
- Oracle Berkeley DB/XML(v 2.4.16)

5.1 성능측정 환경

본 논문에서는 UbiDB/XQuery를 비롯해서 이와 비교할 기존의 XML 질의언어 엔진을 동일한 시스템 환경에 설치한 후, 질의 처리 속도를 측정 하였다.

질의 처리 속도를 측정한 시스템 환경은 아래 <표 6>과 같다.

〈표 6〉 성능측정 환경

CPU	Memory	OS
Single Core Intel Pentium4 3.00GHz, 1MB L2 Cache	3GB	Fedora Core4 (Linux 2.6.11 Kernel)

5.2 성능측정에 사용된 데이터와 질의

성능 측정은 크게 두 가지로 나누어 진행하였다.

첫 번째는 PML Core Specification에서 예시로 작성된 온도 센서 데이터와 3개의 질의를 이용한 방법이다. 이는 실제로 사용되는 센서 데이터에 대해 성능을 측정함으로써 실시간으로 처리 할 수 있는지 여부를 확인한다.

두 번째 성능 측정 방법은 XMark에서 제공하는 테스트 데이터와 정의된 20개의 질의를 이용하는 것이다. 이는 XML 질의언어의 주요기능들에 대해 동작하는지 알아보고 특정 질의 패턴에 대해 성능 분석을 하기 위함과 동시에 문서의 크기에 따른 성능 차이도 확인하기 위하여 성능을 측정한다.

5.2.1 RFID EPC 데이터

유비쿼터스 환경에서 발생하는 데이터에 대해서 실시간 질의 처리 성능을 평가하기 위해 RFID 센서에서 사용되는 XML로 표현되어 있는 EPC 데이터를 가지고 성능 측정을 한다. 아래 <표 1>은 RFID온도센서에서 발생하는 XML데이터를 나타낸다.

〈표 7〉 RFID EPC 데이터에 대한 질의

Q1	let $sensor := doc('sensorData.xml') for $b in $sensor/Sensor where $b//TemperatureReading/Value/text() > 17 return $b/ID/text()
Q2	let $sensor := doc('sensorData.xml') for $b in $sensor/Sensor where $b/Observation/Data/XML/ TemperatureReading/Value/text() > 17 return $b/ID/text()
Q3	let $sensor := doc('sensorData.xml') return for $b in $sensor/Sensor where $b//XML/TemperatureReading/ Value/text()>17 return <Alert> <ID> { $b/ID/text() } </ID> <AlertTime>{$b/Observation/DateTime/te xt()}</AlertTime> <Celsius>{$b/Observation/Data/XML/Tem peratureReading/Value/text()}</Celsius> </Alert>

ID 태그는 센서들의 ID를 나타내고 Date Time 태그는 온도를 측정한 시간을 나타낸다. 또한 Value 태그는 측정한 온도를 나타낸다.

<표 7>의 RFID EPC 데이터에 대한 질의는 RFID/USN 환경에서 일반적으로 데이터를 검색하는 질의 유형들이다.

Q1, 2는 온도 센서 데이터에서 온도 이상이 발생한 태그의 ID를 얻어오는 질의로써 각각 경로 표현을 Child Axis만으로 구성한 것과 Descendant Axis도 포함한 질의로 구분한 것이다. Q3은 경로 표현을 Child Axis와 Descendant Axis를 복합적으로 사용했으며 주된 평가 요소는 재구성 수행에 있다.

5.2.2 XMark 데이터

XMark에서는 XML 질의언어 엔진에 대한 구조적 문제를 확인하고 성능 평가를 위해 'auction.xml'이라는 XML 문서와 이 문서에 대한 질의 20개를 제공한다.

특히, XMark는 ScaleFactor를 통해 크기가 다른 XML 문서에 대해서 자동 생성기능을 제공한다. 본 논문에서 측정하고자 하는 XML문서 크기는 <표 8>과 같다.

〈표 8〉 Scale Factor에 따른 문서 크기

Factor	0.00001	0.0001	0.001	0.01
크기	27KB	34KB	116KB	1.1MB

XMark에서 제공하는 20개의 질의에 대해서 각각의 질의가 가지는 중점 평가 내용은 <표 9>와 같다. 여기서 질의 18번은 사용자 정의 함수에 대한 질의인데 UbiDB/XQuery는 사용자 정의 함수 기능을 지원하지 않아 성능평가에 포함하지 않았다.

〈표 9〉 XMark 질의의 중점 평가 내용

XMark 질의	중점 평가 내용
Q1, Q2, Q3	Predicate
Q4	Node-order
Q5	함수 처리
Q6, Q7	Descendant 경로
Q8, Q9	중첩 Binding과 다양한 경로
Q10	복잡한 재구성
Q11, Q12, Q17	Cardinality와 연산자
Q13	노드를 이용한 재구성
Q14	Full-Text 질의
Q15, Q16	Depth가 깊은 경로
Q18	사용자 정의
Q19	결과 정렬
Q20	FLWOR 절이 포함된 복잡한 재구성

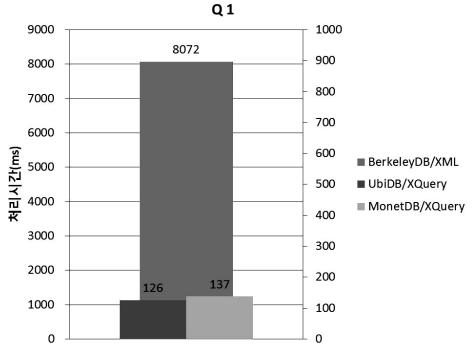
5.3 성능측정 및 분석

5.3.1 RFID EPC 데이터 성능측정 및 분석

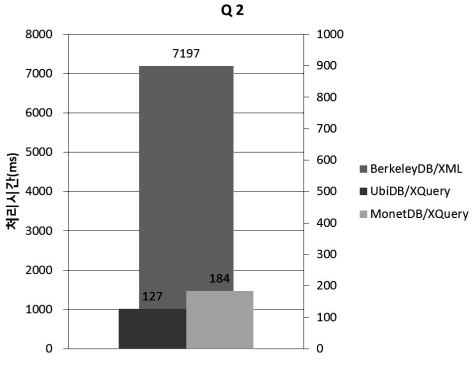
RFID EPC 데이터에 대한 질의 성능 평가는 <표 1>에 나와 있는 XML 문서 12000개를 UbiDB/XQuery 엔진을 비롯한 타 XML 질의 언어 엔진에 저장시킨 후 <표 7>의 세 가지 질의를 수행시켜 RFID/USN 환경에서 실시간 데이터 처리가 가능한지를 알아보았다.

비교적 수행 속도가 빠른 UbiDB/XQuery와 MonetDB/XQuery는 100ms 단위의 오른쪽 시간 축을 기준으로 그래프를 표현 하였고 Oracle Berkeley DB XML은 1000ms 단위의 왼쪽 시간 축을 기준으로 표현 하였다.

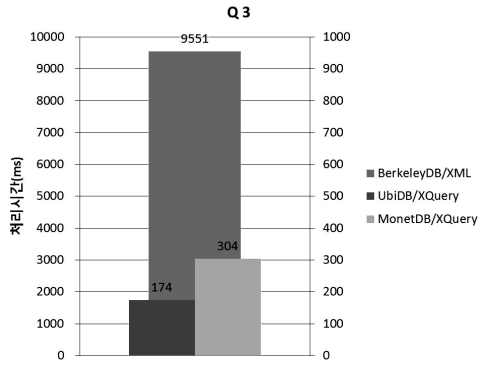
XML 문서를 태그 또는 임의 단위로 분할하여 DBMS에 저장하는 방법을 사용하는 UbiDB/XQuery와 MonetDB/XQuery가 XML 문서 단위로 DOM을 구성하는 방법을 사용하는 Oracle Berkeley XML DB보다 훨씬 빠른 질의 처리 속도를 보여주고 있다. 질의 Q1, 2에 대해서는 UbiDB/XQuery와 MonetDB/ XQuery가 비슷한 질의 처리 속도를 보여주고 있지만 재구성이 포함되어 있는 Q3에서는 MonetDB/XQuery보다 2배정도 빠른 것을 볼 수 있다. 이는 본 논문에서 개발한 재구성 일괄처리 방식을 이용했기 때문으로 분석할 수 있다. 측정 결과와 같이 12,000개의 EPC 데이터를 1초 안에 처리함으로써 UbiDB/XQuery가 대량의 센서 데이터에 대해서 실시간 처리가 가능함을 나타내고 있다.



〈그림 12〉 Q1 질의 측정 결과



〈그림 13〉 Q2 질의 측정 결과



〈그림 14〉 Q3 질의 측정 결과

5.3.2 XMark 성능측정 및 분석

XMark 데이터에 대한 성능 측정은 <표 8>에 나와 있는 4개의 문서에 대해 <표 9>에 나와 있는 XMark 19개의 질의에 대해서 총 5번을 수행한 후 수행 시간의 평균을 최종 질의 수행 시간으로 측정하였다. 측정 단위는 밀리세컨드이고 그래프에서 가로축은 XMark 질의를 나타내고 세로축은 질의 수행 시간을 나타낸다. 또한 각각의 질의 밑에 나와 있는 숫자는 질의 수행 시간을 가리킨다. 각 그래프 마다 XML 문서 크기 변화에 따른 성능 분석을 하고 마지막에는 질의 유형에 따른 각 XML 질의언어 엔진에 따른 분석을 한다.

UbiDB/XQuery가 모든 질의 유형에서 속도가 빠름을 확인할 수 있다. Q10과 Q20에서 속도가 가장 느린데 이는 재구성이 포함된 질의이다. 하지만 다른 XQuery 엔진에 비해서는 가장 빠른 것을 확인할 수 있다.

MonetDB/XQuery는 작은 문서에서 UbiDB/XQuery나 Oracle Berkeley DB XML보다 속도가 느린 것을 확인 할 수 있다. 이는 내부적인 전처리 과정에 시간이 많이 소모되기 때문으로 추측할 수 있다. Q3, Q9, Q10, Q20에서 속도가 느린 편인데 이는 연산자, 중첩 바인딩, 재구성이 포함된 질의이다. Oracle Berkeley DB XML은 UbiDB/ XQuery보다 2배 정도 속도가 느린 것을 확인할 수 있다. Q10, Q15, Q16, Q20에서 속도가 느린 편인데 이는 재구성, depth가 깊은 경로 처리가 포함된 질의이다.

모든 XQuery 엔진이 27KB XML 문서 측정 결과보다 일부 질의에서 속도가 소폭으로 느려진 것을 볼 수 있다. 이는 문서의 크기 증가에 따라 처리해야 할 노드가 많아지기 때문으로 추측할 수 있다.

116KB XML 문서 측정 결과를 보면 UbiDB/XQuery와 MonetDB/XQuery는 속도가 소폭 느려졌으나 Oracle Berkeley DB XML은 2~3배 정도 속도가 느려진 것을 볼 수 있다. 이는 Oracle Berkeley DB XML이 다른 두 XML 질의언어 엔진에 비해 문서의 크기에 영향을 많이 받음을 알 수 있다.

UbiDB/XQuery는 다른 질의유형에 비해 Q15, Q16에서 질의 처리 속도가 가장 빠름을 확인 할 수 있는데 이는 역 경로 요약과 가지 질의로 인한 성능 향상으로 추측할 수 있다. 하지만 Q3의 속도가 많이 느려진 것을 확인할 수 있다. Q3은 연산자와 Predicate이 복합적으로 나온 질의 유형이다.

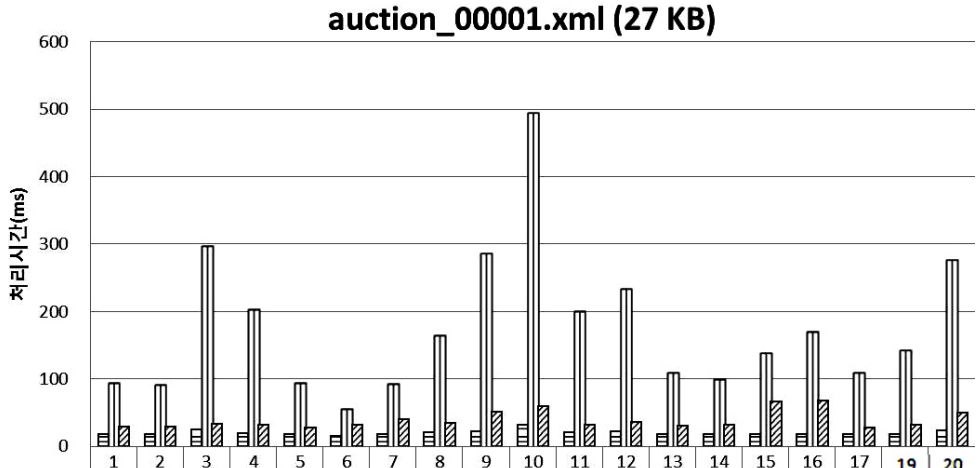
MonetDB/XQuery는 문서 크기 증가에 따라 꾸준히 증가함을 알 수 있다.

Oracle Berkeley DB XML은 1.1MB XML 문서를 처리 하면서 Q8, Q9, Q11, Q12에서 비약적으로 속도가 느려짐을 확인 할 수 있다.

전체적인 측정 결과를 보면 알 수 있듯이

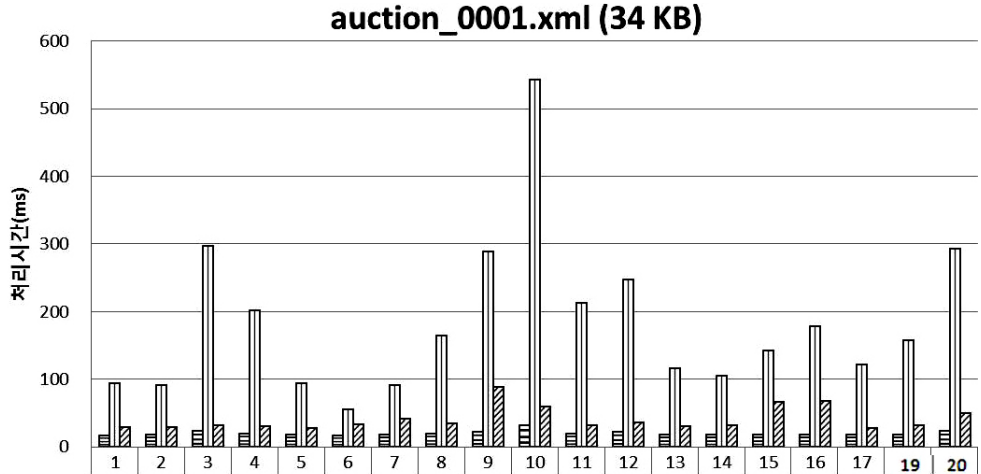
UbiDB/XQuery가 모든 문서에 대해 가장 빠른 질의 처리 속도를 보이고 있고 그 다음으로 MonetDB/XQuery, Berkeley DB XML 순으로 질의 처리 속도가 빠르다.

문서 크기에 따라 모든 XQuery 엔진이 속도가 증가하였으나 Oracle Berkeley DB



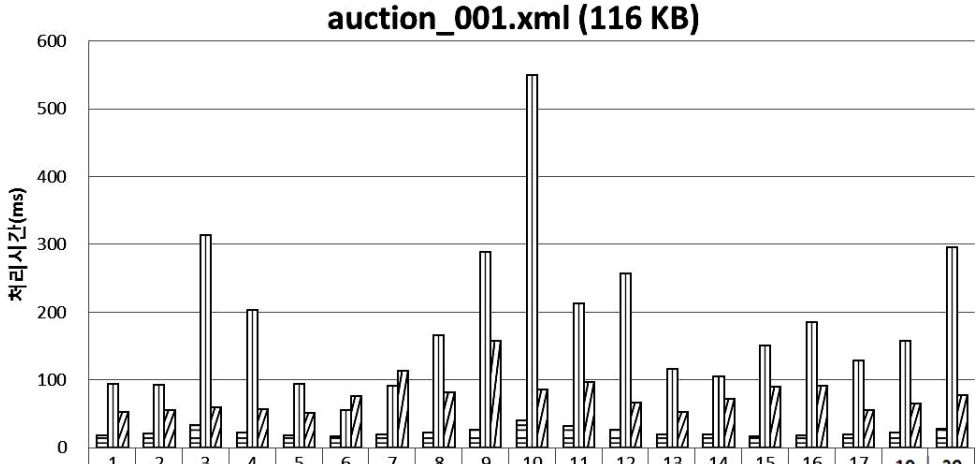
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20
UbiDB/XQuery	18	19	25	20	18	16	18	21	23	32	21	22	18	18	18	19	18	19	24
MonetDB/XQuery	94	92	297	202	95	55	92	165	286	495	200	233	109	99	138	170	109	143	275
BerkeleyDB/XML	29	29	33	32	28	33	40	35	51	60	32	36	30	32	67	68	28	32	51

〈그림 15〉 27KB XML 문서 측정 결과



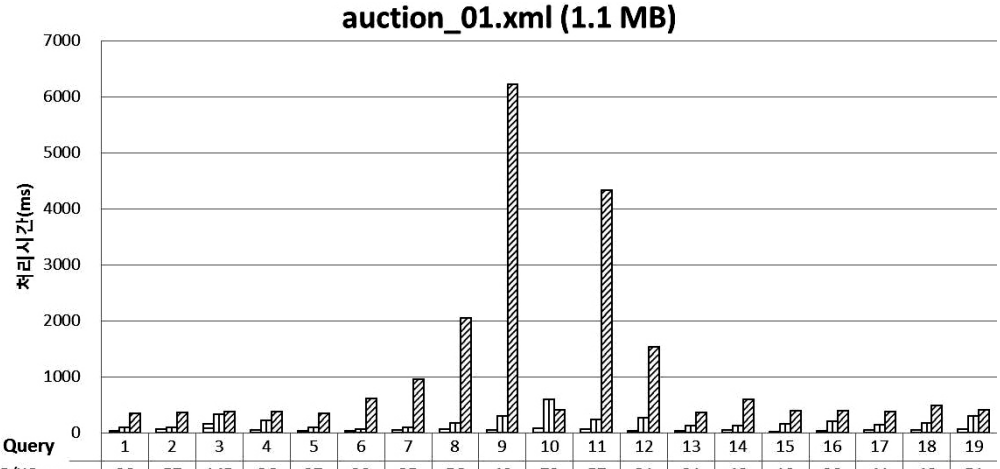
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20
UbiDB/XQuery	17	18	24	20	18	17	18	20	23	32	20	22	18	18	18	19	18	19	24
MonetDB/XQuery	94	92	297	202	95	56	92	165	288	543	213	247	116	105	143	178	122	157	294
BerkeleyDB/XML	29	29	32	31	29	34	42	35	88	60	32	36	30	32	67	68	28	32	50

〈그림 16〉 34KB XML 문서 측정 결과



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20
UbiDB/XQuery	19	21	33	23	18	17	20	23	27	40	32	26	20	20	17	19	20	22	28
MonetDB/XQuery	94	93	313	203	94	55	92	166	289	550	213	257	116	105	151	185	130	158	295
BerkeleyDB/XML	53	55	60	57	52	76	114	82	158	86	97	67	53	73	91	92	55	65	78

〈그림 17〉 116KB XML 문서 측정 결과



XMark Query	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
UbiDB/XQuery	28	57	145	36	27	20	35	56	49	72	57	31	21	48	19	20	41	48	51
MonetDB/XQuery	96	97	318	217	97	58	94	175	297	583	223	262	126	116	157	196	133	166	298
BerkeleyDB/XML	346	361	375	372	341	605	954	2039	6215	395	4326	1527	354	587	381	384	369	480	403

〈그림 18〉 1.1MB XML 문서 측정 결과

XML은 <그림 18>에서 처럼 질의 처리 속도가 문서의 크기에 영향을 많이 받는 것을 확인 할 수 있다.

XMark 질의 유형별로 살펴보면 다음과 같다. UbiDB/XQuery는 Q15, Q16과 같이 depth가 깊은 경로가 포함된 질의에서 가장 빠르

다. 이는 센서 데이터의 특정 노드에 접근하여 값을 가져오는 과정을 역 경로 요약과 가지 질의 방식을 이용하여 처리 속도를 향상 시켰기 때문이다. 또한, Q10, Q20과 같이 재구성이 포함된 질의에서 가장 느리지만 이는 다른 XML 질의언어 엔진에 비해서는 빠른 것이다. 본 논문에서 개발한 재구성 일괄 처리 방식을 이용하여 처리 속도를 향상 시켰기 때문이다. 대량의 센서 데이터를 처리하기 위한 방법으로 역경로 요약, 가지 질의, 재구성 일괄 처리 방식을 이용하여 실시간 처리가 가능하게 되었다.

MonetDB/XQuery는 Q15, Q16과 같이 depth가 깊은 경로가 포함된 질의에서 다른 질의 유형에 비해 중간 정도의 질의 처리 속도를 보이고 있다. 이는 전처리 과정에서의 시간 소모가 있기 때문으로 파악할 수 있다. Q10, Q20과 같이 재구성이 포함된 질의에서는 속도가 매우 느리다. MonetDB/XQuery에서는 재구성 처리에 신경쓰지 않는 것으로 추측할 수 있다.

Oracle Berkeley DB XML은 Q15, Q16과 같이 depth가 깊은 경로가 포함된 질의와 Q10, Q20과 같이 재구성이 포함된 질의에서 다른 질의 유형과 비슷한 질의 처리 속도를 보이고 있다.

UbiDB/XQuery 성능 평가 중 <그림 18>에서 보는 바와 같이 Q3, Q11, Q20의 속도가 느린 것을 확인 할 수 있는데 이는 Operator 처리가 미흡함을 나타내며 그 이유는 경로 질의를 통해 얻어온 결과 값들을 순차적으로 Operator 처리를 해야 하기 때문이다. 또한, Q2, Q3의 속도가 느린 것도 확인 할 수가 있는데 이는 Predicate 처리가 미흡함을 나타내며 그 이유는 '$b/bidder[1]/increase/text()'라는 Predicate 조건의 경로의 경우 세 번의 데이터베이스 질의와 두 번의 구조적 조인, 그리고 한 번의 Predicate 처리 과정이 필요하기 때문이다. 첫째로 '$b/bidder'의 결과 값을 데이터베이스에서 얻어 온 후 첫 번째 결과를 저장한다. 둘째로 '$b/bidder/increase'의 결과 값을 구해 바로 전에 저장한 결과와 구조적 조인을 실행하여 저장한다. 마지막으로 '$b'의 결과 값을 구해 바로 전에 저장한 결과와 구조적 조인을 실행하여 '$b'의 값을 구해야 하기 때문이다.

6. 결 론

최근 빠르게 확대되어 가고 있는 유비쿼터스 환경에서 실시간으로 생성되고 처리 되는 센서 데이터의 수가 엄청나게 늘어나고 있다. 센서 데이터는 XML로 작성되어 있으며 그 크기가 1KB 미만으로 아주 작은 데이터이다. 이렇게 발생되는 수많은 센서 데이터를 실시간으로 처리하기 위한 XML DBMS/XQuery의 요구가 늘어나고 있다. 같은 종류의 센서에서 발생하는 센서 데이터는 크기가 작고 동일한 스키마를 가지고 있다. 이는 센서가 표현하고자 하는 데이터가 동일한 경로에 존재하고 있다는 뜻이다. 이런 특징을 가지고 있는 센서 데이터들을 실시간으로 처리하기 위해서는 그에 특화된 XML 질의언어 처리 기법이 필요하다. 이러한 요구에 따라 본 논문에서는 작은 크기의 대량의 XML 데이터에 대한 실시간 검색을 처리 할 수 있는 XML 질의언어 엔진을 설계 및 구현하였다.

XML 문서 단위로 DOM을 구성하는 방법으로 보통 Native XML Database에서 사용하고 XML 문서를 태그 또는 임의 단위로 분할하여 DBMS에 저장하는 방법은 XML-Enabled Database에서 사용하는 것으로 나뉜다.

XML 문서 단위로 DOM을 구성하는 방법은 대량의 문서를 처리 할 때 순차적으로 한 번에 한 문서씩 구성하게 되는데 DOM 구조의 특성으로 인해 질의 처리 시에 별도의 노드 재조립 과정 없이 링크 검색만으로도 해당 노드를 찾을 수 있다는 장점이 있지만, Descendant Axis와 같은 질의가 나타날 경우 모든 노드에 대해 검색을 해야 한다는 단점이 있다.

XML 문서를 태그 또는 임의 단위로 분할하여 DBMS에 저장하는 방법은 대량의 문서를 처리할 때 데이터베이스에 일괄적인 SQL 질의를 통해 원하는 노드 정보들을 가져와서 일괄처리 하는 방식이다. 질의 처리 시 문서의 개수에 영향을 받지 않는다는 장점이 있지만, 매 경로 질의 시마다 데이터베이스에 질의를 해야 하고 XML 질의언어를 SQL로 변환해야 한다는 단점이 있다.

본 논문에서 개발한 UbiDB/XQuery는 문서 일괄 처리 방식을 사용하기 위한 역 경로 요약과 가지 질의 방식을 이용하였고 효율적인 재구성 처리 방법을 개발하여 실시간 센서 데이터 처리를 가능하게 하였다.

본 논문은 위와 같은 기술을 이용하여 유비쿼터스 환경에서 발생하는 대량의 센서 데이터들을 실시간으로 처리할 수 있음을 검증하였다.

제 5.3.2절 마지막에 언급한 문제를 해결하기 위해서는 향후 DATA 테이블의 value 속성을 Integer Type과 String Type으로 분리하여 연산자 처리 시 DB 질의 처리로만 가능하도록 해야 할 것이다. 또한 향후 sibling도 구분 할 수 있는 넘버링(numbering) 기법을 개발하여 순서를 조건으로 하는 Predicate 처리를 단순화 하는 방법으로 해결해야 할 것이다.

USN(Ubiquitous Sensor Network)과 같은 유비쿼터스 기술은 앞으로도 계속 확산될 전망이며, 실시간으로 저장/처리해야 할 XML 데이터의 양은 급속도로 증가 될 것이다. 또한, 전자상거래에서도 많이 사용하는 XML 문서를 처리해야 하는 요구도 계속 늘어날 것으로 본다. 따라서 다양하고 대량의 XML 데이터를 신속하게 저장하고 처리할 새로운 형태의 XML DBMS가 필요하게 될 것이며, 이와 같은 DBMS는 차세대 컴퓨팅 환경에서 필수 요소가 될 전망이다.

참 고 문 헌

[1] 유승화, “표준화논단 : RFID/USN 표준화 추진방향”, TTA저널, 제94호, 2004, pp. 12-18.

[2] Albercht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, loana Monolescu and Ralph Busse, “XMark : A Benchmark for XML Data Management,” Proceedings of the 28th international conference on Very Large Data Bases, 2002, pp. 974-985.

[3] Auto-ID Center, "PML Core Specification 1.0," Auto-ID Center Recommendation, 2003.

[4] Daniela Florescu and Donald Kossmann, "Storing and Querying XML Data using an RDBMS," Bulleting of the IEEE Computer Society Technical Committee on Data Engineering, 1999, pp. 27-34.

[5] FastDB 2.41, http://www.ispras.ru/~knizhnik/fastdb.html

[6] Michael Stonebraker and Uğur Çetintemel. "One Size Fits All : An Idea Whose Time has Come and Gone," In Proceedings of the International Conference on Data Engineering(ICDE), 2005, pp. 2-11.

[7] Ronald Bourret, "XML Database Products : Native XML Databases," http://www.rpbourret.com/xml/ProdsNative.htm, 2005.

[8] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie and Jérôme Siméon, "XQuery 1.0 : An XML Query Language," Recommendation, W3C, http://www.w3.org/TR/xquery, 2007.

[9] Su-Cheng Haw and G. S. V. Radha Krishna Rao, "Path Query Processing in Large Scale XML Databases," Asian Network for Scientific Information, 2007, pp. 2736-2743.

[10] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita and Chun Zhang, "Storing and Querying Ordered XML Using a Relational Database System," In Proceedings of ACM SIGMOD, 2002, pp. 204-215.

[11] XQilla, http://xqilla.sourceforge.net/HomePage

[12] Zografoula Vagena, Mirella M. Moro and Vassilis J. Tsotras, "Twig Query Processing over Graph-Structured XML Data," Proceedings of the 7th International Workshop on the Web and Databases, 2004, pp. 43-48.

저 자 소 개

임형준	(E-mail : hyungjun25@cnu.ac.kr)
2003년~2007년	충남대학교 전기정보통신공학부 컴퓨터전공 (학사)
2007년~2009년	충남대학교 대학원 컴퓨터공학 (석사)
2009년~현재	충남대학교 대학원 컴퓨터공학 (박사과정)
관심분야	DDS, 웹서비스, 유비쿼터스 웹서비스(UWS)

김재훈	(E-mail : kensyu@cnu.ac.kr)
2000년~2007년	충남대학교 전기정보통신공학부 컴퓨터전공 (학사)
2007년~2009년	충남대학교 대학원 컴퓨터공학 (석사)
2009년~현재	한국과학기술정보연구원
관심분야	XML, 데이타베이스, 클라우드 컴퓨팅

이규철	(E-mail : kclee@cnu.ac.kr)
1984년	서울대학교 컴퓨터공학 (학사)
1986년	서울대학교 컴퓨터공학 (석사)
1990년	서울대학교 컴퓨터공학 (박사)
1989년~현재	충남대학교 컴퓨터공학과 교수
1989년~1994년	미국 IBM Almaden Research Center 초빙연구원
1995년~1996년	미국 Syracuse University, CASE Center 초빙 교수
1997년~1998년	교육부 학술진흥재단 부설 첨단학술센터 파견 교수
1997년~현재	한국정보과학회 논문편집위원
2000년~2004년	산업자원부 한국 ebXML 전문위원회 위원장
2003년~현재	한국전자거래학회 이사
2003년~현재	조달청 목록자문위원
2003년~현재	웹 코리아포럼 위원장
2005년~현재	한국기록관리학회 이사
2006년~현재	충남대학교 소프트웨어연구소 소장
2006년~현재	대통령자문 정부혁신 지방분권위원회 위원
관심분야	데이타베이스, XML, 정보 통합, 멀티미디어 시스템, e-비즈니스 시스템, 유비쿼터스 웹서비스(UWS)