스트리밍 XML 데이타에서 영역 윈도우를 사용한 조인 질의의 범위 최소화 기법

(Scope Minimization of Join Queries using a Range Window on Streaming XML Data)

박 석 † 김 미 선 ††
(Seog Park) (Misun Kim)

요 약 XML이 인터넷 상에서 데이타 교환의 표준으로 자리매김하면서 스트리밍 환경의 XML 데이타에 대한 효과적인 조인 질의 처리도 증가하고 있다. 튜플 단위로 처리하는 기존의 데이타베이스 기법을 스트리밍 XML 데이타에 적용했을 때 제한된 메모리 사용에 따른 메모리 한계를 초과하는 문제가 발생한다. 또한 구조적인 특징을 가지는 XML 데이타에 대한 질의 경로 탐색 및 특정 부분 데이타에 대한 접근에 소모되는 처리 비용이 급격히 증가하는 문제가 발생하게 된다. 근본적으로 전체 데이타가 아닌 부분 데이타를 저장하고 질의 처리해야 하는 스트리밍 환경에 적용하기에는 부적절하다. 따라서 스트리밍 XML 데이타에 맞는 저장 기법으로 적은 메모리의 사용을 통해 빠르게 조인 프레디킷을 만족하는 부분 스트리밍 데이타를 검색할 수 있는 새로운 기법이 요구된다. 본 논문에서는 적은 메모리 사용을 위한 저장 기법을 위해 PCDATA와 CDATA에 해당되는 부분만을 추출하여 저장한다. 그리고 빠른 조인 프레디킷(Predicate) 비교를 위해 DTD의 구조정보 중 지시자(Cardinality) "*" 와 "+"를 기초하여 영역 윈도우(Range Window)를 설정하여 질의에 만족하는 윈도우만을 선택적으로 조인하는 기법을 제안하여 문제를 해결한다.

키워드 : 스트리밍 XML 데이타, 영역 윈도우 조인, DTD 기반 넘버링 스킴

Abstract As XML became the standard of data exchange in the internet, the needs for effective query processing for XML data in streaming environment is increasing. Applying the existing database technique which processes data with the unit of tuple to the streaming XML data causes the out-of-memory problem due to limited memory volume. Likewise the cost for searching query path and accessing specific data may be remarkably increased because of special structure of XML. In a word it is unreasonable to apply the existing database system to the streaming environment that processes query for partial data, not the whole one.Thus, it should be able to search partial streaming data that rapidly satisfies join predicate through using low-capacity memory, based on a store technique suitable to streaming XML data. In this thesis, in order to study the store technique for low-capacity memory, the PCDATA and the CDATA-related parts, which can be used as predicate on join query, were fetched and saved. In addition, in an attempt to compare rapid join predicates, the range window of streaming XML data was set with the object of selectively joining windows that satisfies the query, based on Cardinality * and + among the structure information of DTD.

Key words : Streaming XML data, Range Window Join, Numbering Scheme based DTD

1. 서 론

최근 인터넷의 급격한 발전과 유비쿼터스 컴퓨팅 환경(ubiquitous computing environment) 그리고 센서 네트워크(sensor network)와 같은 많은 정보들의 교환이 이루어지는 환경에서 무한의 연속적으로 전송되는 데이타에 대한 처리가 요구되고 있다. 이러한 무한의 연속 데이타를 스트리밍 데이타(streaming data)라고 한

· 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10395-0) 지원으로 수행되었음
† 종신회원 : 서강대학교 컴퓨터학과 교수
spark@dblab.sogang.ac.kr
†† 정 회 원 : LG전자 단말연구소 연구원
vmisunv@hotmail.com
논문접수 : 2005년 3월 11일
심사완료 : 2006년 1월 26일

다. 스트리밍 데이타의 특징은 많은 데이타가 끊임없이 들어오며, 데이타의 질의 또한 다수의 연속 질의처리가 요구된다. 이러한 연속 질의에 대해 빠른 응답을 요구하는 환경을 만족하기 위해서는 모든 프로세싱(processing)은 메인 메모리(main memory)에서 이루어져야 한다. 왜냐하면 디스크 입출력(I/O)은 스트리밍 환경의 연속 질의에 대한 빠른 응답이 요구되는 특징에 만족하지 못하기 때문이다[1,2].

특히 무한의 스트리밍 데이타 상에서의 조인 연산은 비용과 시간이 많이 소모되는 연산 중에 하나이다. 왜냐하면 조인을 위해서는 데이타의 모든 튜플(tuple)을 스캔(scan)하여 비교해야 하기 때문이다. 하지만 스트리밍 데이타의 특성상 모든 튜플을 스캔 한다는 것은 불가능하다. 이를 해결하기 위한 방법이 연속 질의를 실행할 때 각 스트리밍 데이타에 대해 시간 간격(time interval)을 이용하여 무한의 스트리밍 데이타 조인 영역을 윈도우(window)라는 작은 영역으로 나눠 제한하는 기법이 기존 연구들에서 보이고 있다[3-5].

스트리밍 데이타가 이전의 튜플과 같이 단순한 속성(attribute)과 값(value)의 쌍으로 이루어지지 않고, 구조정보를 가지는 스트리밍 XML 데이타라면, 윈도우로 작은 영역을 나눠 조인하는 스트리밍 데이타에 대한 조인 연산을 스트리밍 XML 데이타에는 적용할 수 없다. 왜냐하면 스트리밍 데이타에서의 윈도우는 시간 간격을 기준으로 나눠질 수 있지만, 스트리밍 XML 데이타 자체를 단순하게 시간 간격으로 나눠질 수 없기 때문이다. 스트리밍 데이타의 윈도우에 속하는 데이타는 조인을 위해 모두 고려해야 하지만, 시간 간격으로 나눠진 스트리밍 XML 데이타의 윈도우에 속하는 데이타는 조인 질의에서 원하지 않는 경로 즉, 원하지 않는 구조를 가지는 데이타가 존재할 수 있기 때문에 무조건 처리할 수 없다. 따라서 스트리밍 XML 데이타의 윈도우는 시간이 아닌 구조정보를 기준으로 나눠져야 한다. 스트리밍 데이타의 윈도우는 질의에 따라 다양한 시간간격을 가지는 윈도우가 존재할 수 있지만, 구조정보를 기준으로 나눠진 스트리밍 XML 데이타의 윈도우는 다수의 조인 질의를 위해 각각 나눌 필요가 없다. 왜냐하면 구조가 변하지 않는 한 초기 윈도우 설정이 다수의 조인 질의에 동일하게 적용될 수 있기 때문이다.

제한된 메모리를 최대한 활용할 수 있는 스트리밍 XML 데이타에 맞는 저장 기법을 통해 빠르게 조인 프레디킷을 만족하는 부분 스트리밍 데이타를 검색할 수 있어야 한다. 이 논문에서 제안하고자 하는 기법은 적은 메모리 사용을 위한 저장 기법을 위해 전체 스트리밍 XML 데이타를 저장하는 것이 아니라 텍스트(text) 즉, 조인 질의에서 프레디킷으로 사용될 수 있는 PCDATA 그리고 CDATA를 가지는 엘리먼트(element) 그리고 엘리먼트의 속성(attribute)에 해당되는 부분만을 추출하여 저장한다. 기존의 XML 문서의 저장 기법들을 스트리밍 환경에 적용했을 때, 텍스트 파일로 저장하는 기법은 적은 메모리 사용을 보장하지만 질의처리에서는 언제나 처음부터 경로 검색을 해야 하기 때문에 낮은 성능을 보이고 있으며, 트리(tree)로 변환하여 저장하는 기법은 구조정보를 가장 정확하고 쉽게 표현할 수 있지만, 마찬가지로 질의처리를 위해서는 트리의 루트부터 검색해야 하는 단점을 가지고 있다. RDBMS 스키마(schema)로 변환하여 저장하는 기법은 XML 문서의 중복 데이타의 저장을 피할 수 있는 작은 단위로 쪼개 테이블에 저장하기 때문에 효율적인 저장 공간의 활용이 이루어지지만, 질의를 위해서는 테이블 간 내부 조인이 반드시 필요하므로 질의처리 과정 중 테이블 간 조인을 위한 오버헤드가 상당하다[6].

모든 데이타를 저장하여 질의처리 하는 기존 기법들과 비교한다면 제안하는 저장 기법은 적은 메모리 사용을 보장한다. 또한 질의처리 비용을 줄이기 위한 제안 기법으로 빠른 조인 프레디킷 비교를 위해 DTD의 구조정보 중 지시자 "*"와 "+"를 기초하여 스트리밍 XML 데이타의 영역 윈도우(Range Window) 즉, 위에서 언급했던 구조정보를 기초로 한 윈도우를 설정한다. 그리고 질의처리는 XQuery의 조인 질의에 등장하는 XPath를 크게 선택, 조인, 반환 부분 경로로 나눌 수 있다. 가장 먼저 선택 부분의 프레디킷을 실행으로 영역 윈도우를 선택한다. 선택된 영역 윈도우를 대상으로 조인 부분 프레디킷을 실행하여 다시 걸러낸 후 최종적으로 남는 윈도우 영역에서 반환 부분 경로에 해당되는 결과를 낸다. 이와 같이 끊임없이 들어오는 스트리밍 XML 데이타를 작은 영역 윈도우로 쪼개 실질적으로 질의에서 필요로 하는 영역 윈도우만을 선택적으로 조인할 수 있게 된다. 따라서 조인 질의 프로세싱에 참여하는 스트리밍 XML 데이타의 범위를 윈도우 단위로 줄일 수 있기 때문에 기존 조인 기법에 비해 빠르게 처리할 수 있다.

본 논문의 나머지는 다음과 같이 구성되어 있다. 스트리밍 XML 데이타 처리와 관련된 기존 연구들을 살펴보고, 다음은 연구 동기와 제안하는 구조정보를 기초로 한 영역 윈도우를 스트리밍 XML 데이타의 조인 질의 기법을 살펴보고, 그리고 제안하는 기법을 이용한 실험을 통해서 비교 평가한다. 마지막으로 결론 및 추후 연구를 기술한다.

2. 관련 연구

2.1 DTD와 지시자(Cardinality)

XML의 특징과 데이타 교환에 사용하기 위해 공개되는 DTD는 XML을 위한 모델 언어 또는 스키마(schema)로서 XML 문서 내에 등장하는 모든 엘리먼트(element), 속성(attribute) 그리고 문서에 나타날 수 있는 반복 정도를 표현하기 위해 "None", "?", "*", "+"의 4가지 엘리먼트 지시자(cardinality)를 표 1과 같이 정의한다[7].

[8]의 예제로 그림 1의 상품에 대한 commodity.xml과 환율에 대한 currency.xml이 XML 문서 형태를 가지는 스트리밍 데이타가 들어올 때, 그림 2의 모델명이 Palm-Pilot인 상품의 달러 가격을 파운드로 변환된 가격으로 반환하는 조인 질의를 처리하기 위해서는 메인 메모리에 잠시 저장을 해야 한다. 스트리밍 데이타의 특징은 많은 데이타가 끊임없이 들어오며, 데이타의 질의 또한 다수의 연속 질의처리가 요구되고 빠른 응답을 만족하기 위해서는 모든 프로세싱은 메인 메모리에서 이루어져야 한다. 왜냐하면 디스크 입출력은 스트리밍 환경의 연속 질의에 대한 빠른 응답이 요구되는 특징에 만족하지 못하기 때문이다[1,2].

표 1 DTD의 지시자 종류

지시자	설 명
None	해당 엘리먼트는 반드시 한번만 나타나야 한다는 것을 지시 (Default)
?	해당 엘리먼트는 한번 나타나거나 아니면 나타나지 않음을 지시
+	해당 엘리먼트는 한번 나타나거나 여러 번 나타날 수 있음을 지시
*	해당 엘리먼트는 나타나지 않거나 여러 번 나타날 수 있음을 지시

```
<?XML version="1.0"?>
<!DOCTYPE recipe
SYSTEM "commodities.dtd">
<!--Example for XML document-->
<commodities>

<vendor>

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

```
<?XML version="1.0"?>
<!DOCTYPE recipe
SYSTEM "currencies.dtd">
<!--Example for XML document-->
<currencies>
  <currency>
    <code> USD </code>
    <name> US Dollars </name>
    <rates>
      <currency>
        <code> GBP </code>
        <buying> 1.34 </buying>
        <selling> 1.32 </selling>
      </currency>... </rate>
  </currency>...
</currencies>
```

그림 1 XML 예제 문서 commodity.xml(좌)과 currency.xml(우)

```
<result>{
 For   $c in stream(commodity.xml)//vendor/items/item,
       $u in stream(currency.xml)//rates/currency
 Where $c/model/text() = "Palm-Pilot"
and $u/code/text() = "GBP"
 and $u/../../code/text()=$c/price/@currency
 Return <price> $u/buying/value()*$c/price/value() </price>
}</result>
```

그림 2 예제 XML 문서의 XQuery 조인 질의

2.2 기존 XML 문서의 저장 기법

기존 XML 데이타의 유지, 관리, 저장의 필요성에 따라 Flat Text File, Tree 표현, RDBMS 스킴으로 사상(Mapping)등 다양한 저장기법이 제안되었다[9]. 이러한 XML 데이타 저장 기법들을 스트리밍 환경에 적용했을 때 제한된 메모리를 사용해야 하는 시스템 측면의 제약사항과 질의처리를 위한 경로 탐색을 위한 접근기법으로 인해 문제점이 발생한다.

Flat Text File은 질의 처리를 위해 텍스트 파일은 처음부터 끝까지 차례대로 접근하기 때문에 전체가 아닌 부분만을 취해야 하는 질의에는 부적절한 저장 기법이라고 할 수 있다.

Tree로 표현할 때 경로 검색에 대한 다양한 연산을 지원할 수 있어 질의 경로 탐색에 효율적이나 데이타를 표현하기에 가장 많은 노력과 공간이 필요하다.

RDBMS 스킴으로 사상하는 기법 중 그림 3과 같이 구조정보를 하나의 Edge 테이블에 한꺼번에 저장하는 Edge Approach[9]는 경로 검색을 위해 Edge 테이블을 Self-Join해야 한다.

DTD의 구조정보를 바탕으로 미리 테이블 스키마를 구성하는 Shared Inlining[10]의 경우 중복 데이타를 없앨 수 있는 반면에 DTD의 지시자를 기준으로 데이타를 다수의 테이블로 분할 저장되어 경로 검색을 위해서는 분할된 테이블을 조인하는 내부 조인 과정이 반드시 필요하다. DTD의 지시자가 많은 경우 분할되는 테이블의 수는 급격하게 증가하게 되며, 이에 따라 내부 조인에 소모되는 비용 또한 급격하게 증가하게 되는 문제점이 있다. 그림 4는 commodity.xml의 DTD 그래프를 표현한 것이며, 그림 5는 DTD 그래프를 RDBMS 스키마로 변환한 것이다.

따라서 기존 저장기법은 XML의 전체 데이타를 모두 저장하는 방법으로 스트리밍 환경의 제한적 메모리 사용을 충분히 만족하지 못한다. 그러므로 질의처리를 위한 임의 접근, 점프 등이 가능한 부분 데이타 저장 기법이 필요하다.

저장된 XML 데이타를 보다 효과적으로 질의처리하기 위해서는 위의 저장기법뿐만 아니라 인덱스를 이용하는 기법을 병행하여 사용하고 있다. 넘버링 스킴을 이용한 XML 인덱스 기법은 데이타의 계층적인 구조정보를 유지함으로써 XML 데이타의 저장과 질의처리에 효과적인 성능을 보이고 있다[9].

2.3 기존 XML 문서의 인덱스 기법

인덱스 기법에는 XML 문서의 트리 구조를 루트부터

Edge Table

source	Ordinal	Tag	Flag	Target
&0	1	Commodities	ref	&1
&1	1	Vendor	ref	&2
&2	1	Name	string	&3
&2	2	items	ref	&4
&4	1	item	ref	&5
&5	1	name	string	&6

Value Table

Node	Value
&3	WallMart
&6	PDA

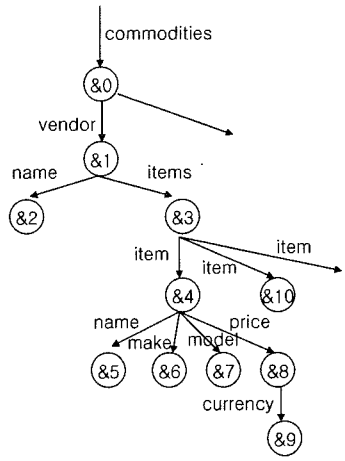

그림 3 Edge Approach

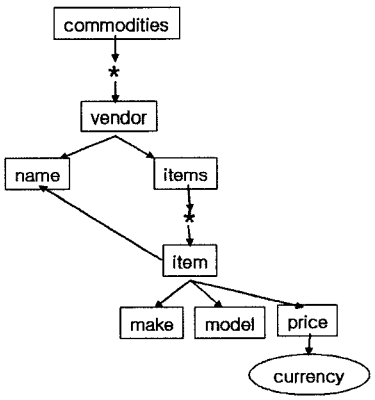

그림 4 Shared Inlining의 DTD 그래프

Commodities (commoditiesID:integer)
Vendor (vendorID:integer, vendor.itemsID:integer)
Name (nameID:integer, name.parentID:integer, name.parentCODE:integer)
Item (itemID:integer, item.make:String, item.mode:String, item.price:String, item.price.currency:String)

그림 5 Shared Inlining의 DTD 그래프를 RDBMS 스키마로 변환

시작하여 깊이 우선 탐색 방법을 사용하여 차례대로 숫자를 정의한다. 순차적인 숫자를 사용하는 Dietz의 넘버링 스킴[11]과 Global Order 넘버링 스킴, Local Order 넘버링 스킴, Dewey Order 넘버링 스킴[12] 등이 있고 순차적인 숫자를 사용하지만 숫자 사이의 일정한 간격을 두는 XISS의 넘버링 스킴[13]이 있다.

2.3.1 Dietz의 넘버링 스킴

Dietz의 넘버링 스킴은 XML의 모든 엘리먼트에 <preorder, postorder>의 레이블(label)을 정의한다. 만약 노드 x와 y가 있고, x가 y의 조상노드 라면, x.preorder

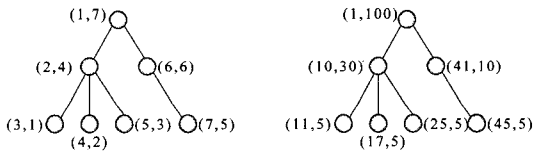

그림 6 Dietz의 넘버링 스킴(좌)과 XISS의 넘버링 스킴(우)

< y.preorder & x.postorder > y.postorder의 조건을 만족한다. 그림 6의 예를 들어 보면, <1, 7>의 레이블을 갖는 노드는 <4, 2> 레이블을 갖는 노드의 조상노드다. 왜냐하면, <1, 7> 노드는 <4, 2> 노드의 Preorder (1<4)이고, Postorder (7>2)이기 때문이다. 장점은 조상-자손(ancestor-descendant) 관계를 트리 노드의 pre-order와 postorder의 간단한 비교만으로 결정할 수 있다. 단점은 새로운 노드가 추가될 때 많은 트리 노드의 레이블이 재구성되어야 한다는 것이다.

2.3.2 XISS의 넘버링 스킴

XISS의 넘버링 스킴은 Dietz의 넘버링 스킴 <pre-order, postorder>에서 <order, size>로 확장한 것이다. Dietz의 기법의 단점을 보완하는 XISS의 넘버링 스킴은 다음의 두 가지 조건을 만족할 수 있도록 구성한다.

(1) 노드 y와 그 노드의 부모노드가 x라고 했을 때, order(x) < order(y) 그리고 order(y) + size(y) <= order (x) + size(x)를 만족한다.

(2) 노드 x와 노드 y가 형제노드고, x가 y보다 앞에 나타난다면, order(x) + size(x) < order(y)를 만족한다.

이 기법의 장점은 Dietz의 스킴보다 더 유연성을 가지며, XML 데이타의 동적인 갱신에 보다 효과적으로 처리할 수 있다. 단점은 노드 사이에 미리 정의된 size를 초과하는 많은 수의 노드가 추가된다면 이 스킴 또한 트리 노드의 많은 레이블을 재구성해야 한다는 점이다.

2.3.3 Global / Local / Dewey Order 넘버링 스킴

XML 문서의 구조에 대한 질의를 가능하게 하기 위해 구조적 특성을 고려하여 순서정보를 유지하기 위한 넘버링 스킴에는 크게 3 가지로 Global Order 넘버링 스킴, Local Order 넘버링 스킴, Dewey Order 넘버링 스킴이 있다. Global Order 넘버링 스킴은 문서의 엘리먼트를 Preorder에 따라 번호를 부여한 것이며, Local Order 넘버링 스킴은 동일한 부모노드를 가지는 자식노드 사이의 순서에 따라 번호를 부여한 것이며, Dewey Order 넘버링 스킴은 Global Order 넘버링 스킴과 Local Order 넘버링 스킴을 혼합한 기법으로 그림 7과 같다.

Global Order 넘버링 스킴의 장점은 질의처리에 효과적이지만, 삽입이 발생했을 때 가장 많은 노드에 넘버를 다시 부여해야 한다는 단점이 있다. Local Order 넘버링 스킴은 삽입이 발생했을 때 가장 적은 노드에 대해서만 넘버를 다시 부여한다는 장점이 있지만, 동일한 넘버를 가지는 노드가 다수 존재하기 때문에 질의처리에서 성능이 떨어지게 된다. Global Order 넘버링 스킴과 Local Order 넘버링 스킴의 장점만을 혼합한 Dewey Order 넘버링 스킴의 경우 질의처리 시에 Global Order 넘버링 스킴과 비슷한 성능을 내며, 삽입이 발생했을 때 Local Order 넘버링 스킴보다는 많지만, Global Order 넘버링 스킴보다는 적은 수의 노드에 대한 넘버링 다시 부여한다.

스트리밍 환경에서는 XML 인스턴스에 대한 삽입 또는 삭제에 대해 고려하지 않는다. 따라서 기존에 연구된 XML 인스턴스 기반의 넘버링 스킴의 노드 삽입과 삭제를 고려한 기능은 불필요한 것이다. 그리고 무한의 스트리밍 데이타에 대해 순차적인 숫자를 지정하는 것 또한 무한의 숫자를 필요로 하게 된다. 그리고 스트리밍 XML 데이타의 경우 동일한 구조 또는 경로를 가지는 데이타의 반복으로 구성되어 있기 때문에 경로는 동일하나 인덱스가 다른 경우가 발생할 수 있다.

따라서 인스턴스의 업데이트를 고려하지 않는 Append-Only 특성을 가지는 스트리밍 환경에 적합한 새로운 넘버링 기법을 위해 고정된 DTD를 기반으로 한 넘버링 스킴이 필요하다.

3. 영역 윈도우 기반의 스트리밍 XML 데이타 조인

DTD 기반의 넘버링 스킴을 이용한 영역 윈도우를 제안하여 빠른 결과를 반환해야하는 스트리밍 XML 데이타의 조인에 적용하여 효과적인 성능을 내고자 한다.

3.1 연구동기

스트리밍 XML 데이타의 조인 질의에 대해 효과적인 성능 향상을 위해서는 일반적인 스트리밍 데이타와는 달리 아래에 나열된 여섯 가지 요구사항들을 만족해야 한다.

첫번째, 스트리밍 XML 데이타의 공간적 제약 사항

스트리밍 XML 데이타에 대한 공간적 제약을 극복하기 위해 제한된 메모리에 XML 데이타 구조정보의 손실이 없는 한도 내에서 적절한 부분 스트리밍 데이타를 저장하는 기법이 요구된다.

두번째, 스트리밍 XML 데이타의 조인 영역을 제한하는 기법

무한한 크기를 가지는 스트리밍 데이타의 조인 질의는 일정한 시간 간격의 윈도우로 나눠 제한하는 기법이 연구되었다. 하지만 스트리밍 XML 데이타는 XML 데이타의 구조 정보를 유지하기 위해 시간 간격을 이용한 윈도우를 적용할 수 없다.

세번째, 부분 데이타 또는 형제노드의 직접 접근

XQuery의 For문의 참조 변수에 정의된 반복경로에 대한 접근이 빈번하게 발생한다. 따라서 접근 빈도가 높은 반복경로 즉, 형제노드에 대해 직접 접근 기법이 요구된다.

네번째, 조인 프레디킷의 빠른 비교 연산

조인은 프레디킷의 값의 비교로 조인 여부가 결정된다. 따라서 루트부터의 경로 탐색 후 값의 비교보다 단말노드 값의 직접 비교가 가능하다면 보다 빨리 조인 여부가 결정될 수 있다.

다섯번째, 단말노드에 도달하는 경로의 빠른 검색

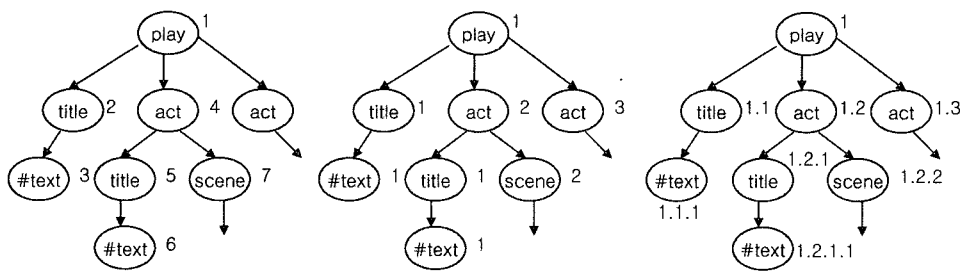

그림 7 Global, Local, Dewey Order 넘버링 스킴

조인 질의는 프레디킷 비교를 위해 단말노드까지 탐색해야 하는 경로가 빈번하기 때문에 경로 탐색 비용을 최대한 줄이기 위해 단말노드까지 도달하는 경로의 빠른 검색 기법이 요구된다.

여섯번째, 불필요한 데이타의 배제 기법

질의에서 고려하지 않는 부분 데이타를 프로세싱 과정에서 빠르게 점프하거나 배제시킬 수 있는 기법이 요구된다.

3.2 가정 및 처리 과정

• 가정

- DTD 기반의 넘버링 스킴을 이용한 정확한 데이타 교환을 위해서 스트리밍 XML 데이타의 DTD 즉 스키마는 항상 공개된다.
- DTD 종속적인 방법으로 한번의 DTD 기반의 넘버링을 통해 정의된 인덱스 값을 사용하기 때문에 DTD의 변경으로 인한 업데이트는 발생하지 않는다.
- 태그 순서와 종류에 따른 텍스트 정보 테이블과 윈도우 정보 테이블을 구성하기 위해서는 스트리밍 XML 데이타는 항상 Document Order를 지키면서 들어온다.

• 제안기법 처리과정

제안하는 기법의 질의처리 과정은 그림 8에서 볼 수 있듯이 총 4 단계로 이루어져 있다. 가장 먼저 질의처리의 바탕이 되는 DTD 구조정보 테이블과 구축된 구조정보 테이블을 사용하여 질의에서 사용되는 경로의 분석이 이루어져야 한다. 그리고 연속적으로 들어오는 스트리밍 XML 데이타에서 단말노드를 추출하여 텍스트 정보 테이블에 저장하는 단계와 마지막으로 실질적인 질의처리 하는 단계로 이루어져 있다.

단계 1. DTD 기반 구조정보 테이블 구축

DTD를 기반으로 루트부터 시작하는 모든 발생 가능한 경로에 대한 유일한 TagID를 정의하여 구조정보 테이블에 저장한 후 들어오는 스트리밍 데이타에 대한 해당 TagID 검색 및 질의 분석에 사용한다. 이때 사용되는 TagID는 <*Level, preorder*>로 이루어진다. Level은 DTD에서 동일한 부모를 가지는 동일한 이름의 엘리먼트는 같은 레벨에 둘 이상 정의하지 않는다는 규칙에 따라, 스트리밍 데이타의 추출 및 저장 단계에서 해당 TagID를 빠르게 검색하기 위해 사용되고, 질의 결과를 재구성할 때 사용된다. 또한 DTD의 지시자 "*", "+"에 대한 정보를 유지하여, 스트리밍 XML 데이타에서 동일한 경로를 가지며 반복적으로 나타나는 엘리먼트를 구분할 수 있는 기준으로 영역 윈도우를 설정한다.

단계 2. 단말노드 추출 및 텍스트 정보 테이블 저장

스트리밍 XML 데이타에서 단말노드 즉, PCDATA와 CDATA 값을 가지는 엘리먼트 만을 추출하여 DTD 구조정보 테이블에서 TagID와 TagID가 포함될 수 있는 영역 윈도우의 Window_TagID를 검색하여 <*TagID, PCDATA or CDATA, Window_TagID*>의 구조로 텍스트 정보 테이블에 저장한다. 이때 계층적인 구조를 가지는 영역 윈도우 정보는 <*Window_TagID, Parent_Window_TagID*>의 구조로 윈도우 정보 테이블에 저장한다.

단계 3. 질의분석

조인 질의에 등장하는 모든 경로를 분석하여 DTD 구조정보 테이블에서 관련 있는 TagID를 검색하여 이후 진행하는 경로 처리는 TagID로 대신한다. 왜냐하면, TagID에는 절대 경로 정보를 포함하고 있기 때문에 이후 별도의 경로 검색이 필요하지 않는다. 그리고 텍스트 정보도 TagID로 저장했기 때문에 검색을 위해서는 TagID를 사용해야 한다.

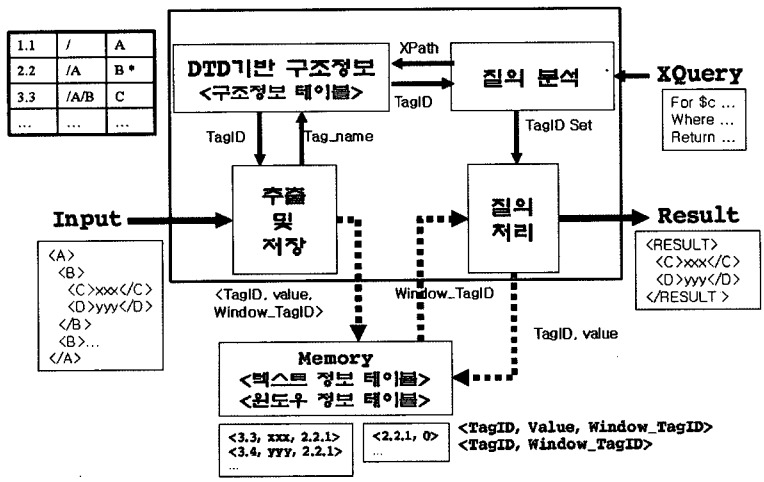

그림 8 스트리밍 XML 데이타에 대한 질의처리 과정

단계 4. 질의처리

XQuery의 모든 경로를 선택, 조인, 반환 세가지 부분으로 나눠 부분 별 분할 처리한다. 선택 부분에 속하는 경로의 TagID를 선택하여 선택 프레디킷을 만족하는 영역 윈도우의 Window_TagID를 검색하고, 조인 부분에 속하는 경로의 TagID는 선택 부분에서 선택된 Window_TagID를 바탕으로 조인 프레디킷을 만족하는 Window_TagID를 선택하고, 조인 부분에서 선택된 Window_TagID를 바탕으로 반환 부분에 해당하는 데이타를 추출하여 재구성 후 결과를 반환한다.

3.3 DTD 기반 구조정보 구축 단계

3.3.1 DTD 기반의 넘버링 스킴과 영역 윈도우

스트리밍 XML 데이타는 인스턴스의 끝을 알 수 없기 때문에 기존의 인스턴스 기반의 넘버링 스킴이 아닌 고정된 DTD 기반의 넘버링 스킴을 정의하고 무한으로 들어오는 XML 인스턴스에 적용한다. 이 기법은 각 엘리먼트의 구조정보를 유일하게 구분할 수 있어 부분 데이타 또는 형제노드로의 직접 접근이 가능하고 단말노드에 도달하는 경로를 탐색하기 위한 경비를 줄일 수 있다. 그림 9의 (a)는 Global Order 넘버링 스킴을 변형한 형태로 노드의 Preorder 순서와 레벨(level)을 혼합하여 DTD의 모든 엘리먼트에 부여한 것으로 <Level, Preorder>의 구조를 가진다. (b)는 XML 인스턴스에서 (a)의 DTD에서 명시한대로 지시자를 가지는 엘리먼트의 XML 인스턴스내의 반복 횟수를 순차적으로 넘버링한 것으로 B 노드의 경우 두번 나타났기 때문에 첫번째 B 노드는 1을 두번째 B 노드는 2를 부여받는다. (c)는 (a)의 DTD 인덱스과 (b)의 지시자 인덱스을 혼합하여 XML 인스턴스의 노드 하나하나에 최종적으로 부여받게 되는 인덱스를 나타낸 것이다. 이때 첫번째 B 노드의 경우 (a)의 2.2와 (b)의 1을 통합하여 2.2.1이라는 인덱스를 부여받은 것이며, Level 3의 C와 D가 동일한 인덱스를 부여받았지만, 제안하는 영역 윈도우의 개념을 추가한다면, 부모 노드의 인덱스가 다르기 때문에 박스로 표시한 것과 같이 첫번째 B 노드의 자식 노드인 C와 D는 두번째 B 노드의 자식 노드인 C와 D와는 구분된다.

정의 1. DTD 기반의 구조정보 넘버링 스킴

<Level, Preorder> 형태의 TagID를 가지는 노드는 다음의 규칙을 가진다.

<Lx, Px>의 TagID를 가지는 노드 X가 <Ly, Py>의 TagID를 가지는 노드 Y의 부모 또는 조상노드일 때,

- Lx < Ly : 부모 또는 조상노드는 자식노드의 상위 레벨에 존재한다.
- Px < Py : 부모 또는 조상노드는 자식노드 이전에 나타난다.

<Lx, Px>의 TagID를 가지는 왼쪽 형제노드 X와 <Ly, Py>의 TagID를 가지는 오른쪽 형제노드 Y가 있을 때,

- Lx = Ly : 형제노드는 동일한 레벨에 존재한다.
- Px < Py : 왼쪽 형제노드는 오른쪽 형제노드 이전에 나타난다.

3.3.2 영역 윈도우 레이블(Range Window Label)

이 논문에서 사용되는 넘버링 스킴은 기존의 넘버링 스킴의 목적인 XML 문서의 업데이트 발생 시 적은 비용으로 인덱스를 재부여(Reordering)하는 것이 아니라, XPath를 단순화시키고, 영역 윈도우를 설정하기 위한 레이블로 사용된다.

조인 프레디킷의 빠른 비교 연산을 위한 단말노드 값의 직접 비교가 가능하기 위해서는 다른 경로의 동일한 이름을 가지는 단말노드뿐만 아니라 동일한 경로의 단말노드에 대해서도 구분할 수 있어야 한다. 이를 해결하기 위한 제안 기법이 바로 영역 윈도우(Range Window)이다. 그림 9의 (c)를 보면 동일한 경로를 가지는 단말노드 C와 D를 구분하기 위해 지시자를 가지는 상위 노드의 TagID를 영역 윈도우의 레이블로 사용한다. 따라서 첫 번째 단말노드 C와 D는 영역윈도우 2.2.1에 속하고, 두 번째 단말노드 C와 D는 영역윈도우 2.2.2에 속하게 되어 구분이 가능하다.

정의 2. 영역 윈도우(Range Window)

<Level, Preorder, Count> 형태의 레이블을 가지는 영역 윈도우는 다음의 규칙을 가진다.

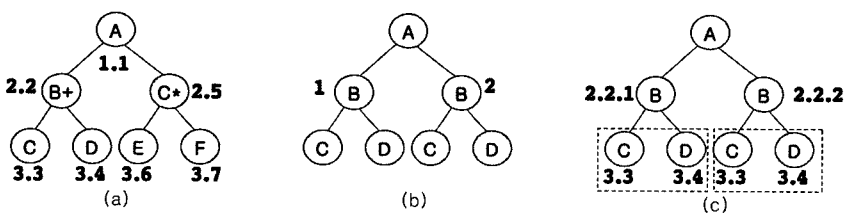

그림 9 기존 스킴을 수정한 DTD 기반의 넘버링 스킴

<Lx, Px, Cx>의 레이블을 가지는 영역 윈도우 X가 <Ly, Py, Cy>의 레이블을 가지는 영역 윈도우 Y를 포함할 때,

- Lx < Ly : X가 Y를 포함하기 위해서는 X는 부모 또는 조상노드이기 때문에 상위 레벨에 위치한다.
- Px < Py : X가 Y의 부모 또는 조상노드이므로, Preorder에서 X가 Y보다 먼저 탐색되어 숫자를 부여된다.
- Cx <= Cy : 하나의 상위 영역 윈도우는 하나 이상의 하위 영역 윈도우를 포함할 수 있기 때문에 항상 Cx는 Cy보다 작거나 같다.

3.3.3 DTD 기반의 구조정보 테이블 구축

DTD 기반 구조정보 테이블은 TagID, Path, Name, Cardinality 속성으로 구성된다.

TagID 속성 - 루트부터 시작하는 가능한 모든 경로에 대해 *<Level, Preorder>*의 형태를 가지는 유일한 숫자를 부여한다. 이후 지시자를 가지는 엘리먼트의 영역 윈도우 레이블은 *<Level, Preorder, Count>*의 형태로 사용된다.

Path 속성 - 루트부터 부모노드까지의 절대 경로를 나타낸다.

Name 속성 - 엘리먼트 이름 또는 엘리먼트의 속성 이름을 나타낸다.

Cardinality 속성 - 지시자 "*"와 "+"를 가지는 엘리먼트 정보를 나타낸다.

3.4 스트리밍 XML 데이타의 추출 및 저장 단계

연속적으로 들어오는 스트리밍 XML 데이타를 제한된 메모리에 모든 데이타를 저장하기 보다는 실질적으로 XQuery에서 프레디킷으로 사용될 수 있는 단말노드의 텍스트 값만을 추출하여 저장함으로써 스트리밍 XML 데이타의 공간적 제약 사항을 해결하고자 한다. DTD를 통해 알 수 있는 계층 구조의 깊이만큼의 크기를 가지는 경로와 윈도우 스택(stack) 2개를 이용해 DTD 구조정보 테이블을 참조하여 경로정보와 영역 윈도우 정보를 임시 저장하고, 스트리밍 XML 데이타의 태그 종류에 따라 다른 실행 방법을 통해 단말노드를 TagID로 변환하여 저장한다. 그림 10은 태그 종류에 따라 별도의 실행 과정을 거쳐 텍스트를 추출 저장하는 과정을 보이고 있다.

시작 태그

태그 이름에 해당하는 TagID를 검색하여, 만약 지시자를 가진 태그라면 TagID에 반복 횟수를 카운트하여 추가하고, 윈도우 정보 테이블에 TagID와 윈도우 스택의 TOP에 저장된 Window_TagID를 *<TagID, Window_TagID>* 구조로 저장한 후 윈도우 스택에 TagID를 PUSH한다. 지시자를 가지지 않은 태그라면 TagID를 경로 스택에 저장한다. 예를 들어, 그림 10의 첫번째 단계인 <commodities> 태그가 도착했을 때, 이 태그는 시작 태그이므로 경로 스택에 저장한다. 하지만 두번째 단계인 <vendor> 태그의 경우 * 지시자를 가지기 때문

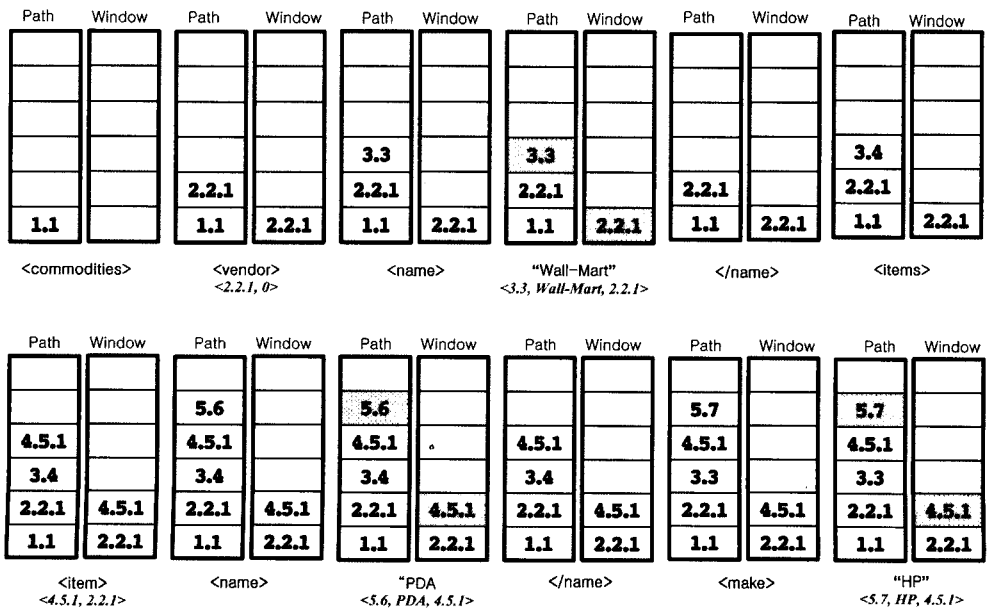

그림 10 스택을 이용한 스트리밍 XML 데이타의 TagID 변환 및 저장 과정

에 경로 스택에 TagID를 저장할 때, 반복 횟수를 카운트하여 추가하고, 윈도우 스택과 경로 스택에 동시에 저장하고, 2.2.1에 대한 Window_TagID가 존재하지 않기 때문에 0으로 설정하여 <2.2.1, 0> 구조를 윈도우 정보 테이블에 저장한다.

PCDATA or CDATA

경로 스택의 TOP에 저장되어 있는 TagID와 윈도우 스택의 TOP에 저장되어 있는 Window_TagID를 PCDATA와 함께 *<TagID, PCDATA, Window_TagID>* 구조로 텍스트 정보 테이블에 저장한다. 예를 들어, 그림 10에서 네번째 단계인 "Wall-Mart"가 도착했을 때, 경로스택의 Top에 저장되어 있는 TagID 3.3과 윈도우 스택에 저장되어 있는 Window_TagID 2.2.1을 읽어 <3.3, "Wall-Mart", 2.2.1> 구조로 구성하여 텍스트 정보 테이블에 저장한다.

종료 태그

경로 스택의 TOP에 저장되어 있는 TagID를 POP하고, 삭제된 TagID가 지시자를 가졌다면, 윈도우 스택의 Top에 저장되어 있는 Window_TagID도 POP한다. 예를 들어, 그림 10의 다섯번째 단계인 </name>이 도착하면 경로 스택의 Top에 있는 TagID 3.3을 POP한다.

위의 태그 종류에 따라 스택에 TagID 또는 Window_TagID를 PUSH/POP하여 정보를 저장하고 PCDATA/CDATA가 도착하면 스택의 정보를 읽어 텍스트 정보 테이블과 윈도우 정보 테이블에 저장하는 과정을 반복하여 구성한다.

그림 11은 텍스트 정보 테이블과 윈도우 정보 테이블의 저장소에 대해 나타내고 있다. 왼쪽 XML 인스턴스에서 추출한 <TagID, Value, Window_TagID>는 XML 인스턴스의 Depth를 기준으로 Hash Table을 이용해 텍스트 정보 테이블에 저장하고, <TagID, Window_TagID>는 TagID에서 반복횟수를 나타내는 Count부분을 제외한 <Depth, Preorder> 인덱스를 기준으로 Hash Talble을 이용해 윈도우 정보 테이블을 구성한다.

3.5 조인 질의 분석 및 질의처리

조인 질의의 모든 경로를 절대 경로 파악이 가능한 TagID로 변환하여 이후 사용되는 모든 경로에 대해 추가 과정 필요 없이 TagID를 기준으로 경로 검색이 가능하도록 한다.

XQuery의 조인 질의에는 다양한 목적을 가지는 경로가 존재한다. 데이타의 필터링을 위한 경로와 조인 프레디킷을 표현하는 경로, 그리고 최종 결과를 반환하기 위한 경로이다. 목적에 따라 *선택 부분, 조인 부분, 반환 부분*으로 경로를 나눠 실행한다.

3.5.1 선택 부분 질의처리

선택 부분로 분류된 경로의 TagID를 이용해 텍스트 정보 테이블에서 선택 부분에 속하는 경로의 조건을 만족하는 Window_TagID를 검색한다. 선택 부분에 속하는 경로가 다수일 경우 각각의 Window_TagID를 구하고, Window_TagID들의 교집합을 구한다. 예를 들어 그림 12와 같은 스트리밍 XML 데이타가 있을 때 다음과 같은 선택, 반환 부분의 경로에 대한 질의처리는 선택 부분의 경로에 해당하는 TagID로 대체한 후 해당 TagID를 가지는 단말노드의 선택 프레디킷을 만족하는 영역 윈도우의 Window_TagID를 추출한다. 다수의 선택 부분의 TagID에 대한 Window_TagID는 교집합을 통해 최종 선택 부분을 모두 만족하는 영역 윈도우를 선택한다.

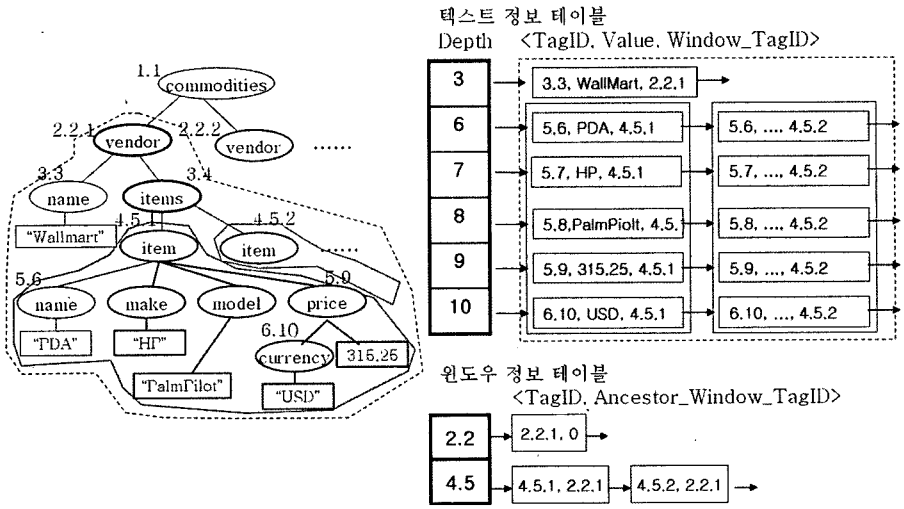

그림 11 윈도우 정보 테이블과 텍스트 정보 테이블 구성

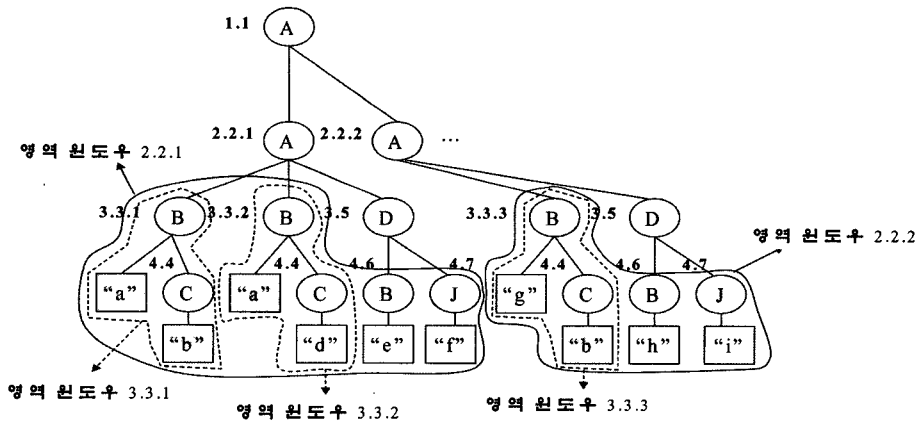

그림 12 영역 윈도우 질의 처리 예제

선택 : //A//C = "b" → 4.4 = {3.3.1, 3.3.3, ...}
//A/B = "a" → 3.3 = {3.3.1, 3.3.2, ...}
4.4 ∩ 3.3 = {3.3.1, ...}

최종 선택 부분을 만족하는 영역 윈도우를 바탕으로 반환 부분에 속하는 TagID에 대한 영역 윈도우를 설정할 수 있다. 예를 들어 TagID 4.7은 영역 윈도우 3.3.1을 포함하는 상위 영역 윈도우 2.2.1에 속하는 엘리먼트의 값을 반환한다.

반환 : //A//B/text() → 3.3 = {3.3.1, ...}
//A//B/text() → 4.6 = {2.2.1, ...}

3.5.2 조인 부분 질의처리

선택 부분에서 구한 Window_TagID를 기준으로 텍스트 정보 테이블에서 조인 부분 경로의 TagID에 해당하는 데이타를 검색하여 다른 스트리밍 데이타의 조인 경로와 비교한 후 조인을 만족하는 Window_TagID를 반환한다.

3.6.3 반환 부분 질의처리

조인 부분에서 반환된 Window_TagID와 반환 부분 경로의 TagID를 이용해 텍스트 정보 테이블에서 검색하여 결과를 반환한다. 반환된 Window_TagID를 포함하는 영역 윈도우의 반환 부분 TagID에 해당하는 데이타를 재구성하여 결과를 낸다. 다음 알고리즘은 기존의 Nested Loop 조인에 영역 윈도우를 추가하여 수정된 것이다. 프레디킷 비교 전에 값의 영역 윈도우가 선택 부분에서 선택된 영역윈도우에 속하는지 검색하는 단계가 추가된다.

Algorithm : Range-Window 기반 Nested Loop 조인

```
WTI      : Window_TagID
TI       : TagID
value    : 조인 Key Predicate
N        : 입력 스트림의 채널 수
SWTI     : 선택 부분를 만족하는 WTI
JWTI     : 조인 부분를 만족하는 WTI
joinpart : 질의의 조인 부분에 해당하는 경로의 TagID
Input    : 입력 스트림 Si의 새로운
           window_TagID(wti)를 가지는 k
Output   : 조인 조건이 만족하는 <S1.window_TagID, ...
           , Sn.window_TagID>

JWI[n];
Si.SWI ← k.wti;
JWI[i] ← k.wti;
All u.wti ∈ S1.SWI and u.ti ∈ S1.joinpart
  If k.value = u.value {
    JWI[1] ← u.wti;
    ...
    All v.wti ∈ Si-1.SWI and v.ti ∈ Si-1.joinpart
      If k.value = v.value {
        JWI [i-1] ← v.wti;
        All w.wti ∈ Si+1.SWI and w.ti ∈
        Si+1.joinpart
          If k.value = w.value {
            JWI [i+1] ← w.wti;
            ...
              All x.wti ∈ Sn.SWI and x.ti ∈
              Sn.joinpart
                If k.value = x.value {
                  JWI [n] ← u.wti;
                  Return JWI;
        }...}}...}
```

4. 비교 및 성능 평가

과거 스트리밍 데이타의 질의처리 과정은 제한적인 메모리 사용을 해결하기 위해 윈도우라는 개념을 사용하여 정확한 결과보다는 근사치 결과를 반환해 왔다. 그

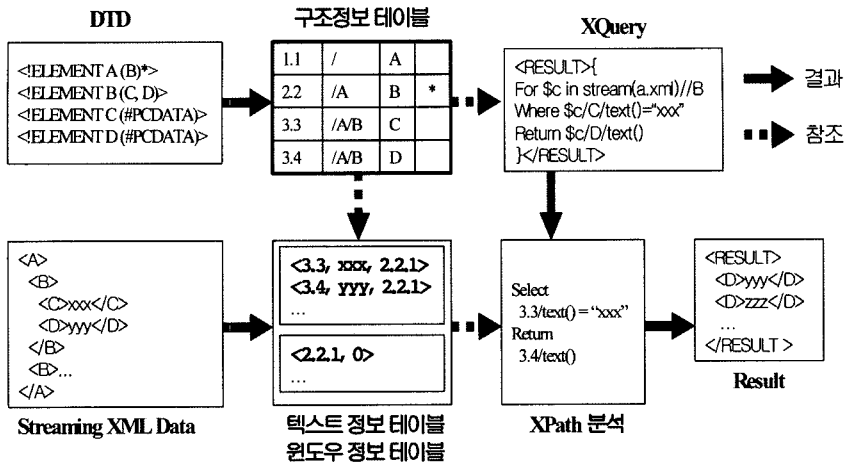

그림 13 영역-윈도우 기반의 중첩 루프 조인 알고리즘 및 전체 처리 진행과정 예

러나 스트리밍 XML 데이타의 경우는 근사치가 아닌 질의에서 요구하는 구조를 정확하게 만족하는 정확한 결과의 반환을 요구한다. 따라서 기존의 전체 데이타를 윈도우라는 시간 간격으로 나눠 처리하는 질의 과정을 대신 XML 데이타의 구조적인 성질을 영역 윈도우로 유지하면서 텍스트 값을 가지는 엘리먼트 또는 엘리먼트 속성을 저장하여 처리하는 질의 과정을 통해 이러한 문제를 해결하고자 한다.

4.1 평가 기준

기존의 Shared Inlining 기법과 제안하는 영역 윈도우 기법의 저장 및 질의처리 과정은 많은 부분이 비슷하게 진행된다. 공개된 DTD를 이용하여 데이타를 분할 저장한다는 것과 XQuery를 두 기법에서 제시하는 질의어 형태에 맞게 변형하여 사용한다는 것이다. 또한 Shared Inlining에서는 명확한 넘버링 스킴을 제안하고 있지는 않지만, 각 테이블의 튜플마다 기본키(primary key)를 설정해야 하기 때문에 기존 형제노드 간의 구분을 위한 Local Order 넘버링 스킴과 유사하다고 볼 수 있고, 영역 윈도우 기법은 기존의 Global Order 넘버링 스킴과 Local Order 넘버링 스킴을 혼합, 변형하여 영역 윈도우의 레이블로 사용하고 있다. 따라서 논리적으로 분할 저장하는 영역 윈도우의 성능 비교를 위해 물리적으로 분할 저장 하는 Shared Inlining 기법을 선택한다. 단 두 기법의 다른 점은 Shared Inlining은 저장된 XML 데이타에 대한 질의를 대상으로 하고, 영역 윈도우 기법은 스트리밍 XML 데이타에 대한 질의를 대상으로 한다는 점이다. 그러나 영역 윈도우의 스트리밍 XML 데이타 질의처리는 먼저 들어오는 스트리밍 데이타에 대해 단말노드만을 추출하고 해당 구조정보를 추가하여 메모리에 저장하고, 저장된 데이타에 질의처리가 수행된다. 따라서 스트리밍 데이타도 메모리에 저장된 XML 데이타에 대한 질의처리로 간주할 수 있다.

실험을 통한 비교를 위해 메모리에 저장된 XML 데이타를 처리한다는 동일한 환경을 조성하고 Shared Inlining 기법의 테이블은 메모리의 Hash Table로 구성하고, 영역 윈도우를 이용한 기법의 텍스트 정보 테이블과 윈도우 정보 테이블도 메모리의 Hash Table로 구성하여 조인 질의처리를 위해 필요한 시간을 측정하고, 조인 프레디킷 비교에 고려되는 엘리먼트 수를 측정하여 실질적으로 질의에서 고려하는 조인 영역의 범위가 질의처리 시간에 영향을 준다는 것을 보이고자 한다.

스트리밍 XML 데이타를 크게 네트워크를 통해 센서

표 2 Shared Inlining과 영역 윈도우 기법의 비교

Shared Inlining	영역 윈도우 기법
DTD 지시자 기반의 테이블 스킴 설정 물리적 분할 저장	DTD 지시자 기반의 윈도우 설정 논리적 분할 저장
테이블의 튜플마다 기본키 부여 Local Order 넘버링 스킴과 비슷	DTD 기반 전체 Path에 TagID 부여 Global+Local Order 넘버링 스킴 혼합
텍스트 값을 RDB 테이블에 저장	텍스트 값을 Hash Table에 저장
XQuery를 SQL로 변환	XQuery의 XPath를 해당 TagID로 변환
Stored XML data에 대한 질의	Streaming XML data에 대한 질의

로부터 전송되어 오는 스트리밍 데이타와 디스크에 저장된 XML 데이타를 메모리에 로딩할 때 들어오는 스트리밍 데이타로 나눌 수 있다. 실험에서 두 가지 상황을 고려하기 위해 작은 사이즈의 깊이가 크지 않은 XML 데이타에서 큰 사이즈의 깊이가 큰 XML 데이타로, Fan-out이 작은 XML 데이타에서 Fan-out이 큰 데이타로 증가했을 때의 성능을 비교한다. 또한 Shared Inlining과 영역 윈도우 기법이 DTD의 지시자를 이용하여 데이타를 분할하는 방법을 사용하므로 지시자의 개수에 따른 두 기법의 성능을 비교하고자 한다. 그리고 마지막으로 조인 프레디킷의 비교 과정에서 고려하는 엘리먼트의 수를 비교하여 영역 윈도우 기법이 조인 질의 적용 영역을 최소화할 수 있다는 것을 보이고자 한다.

• Depth와 지시자 변화에 따른 질의처리 시간 비교

Depth의 증가에 따라 Shared Inlining의 경우 테이블 스키마에 구조정보를 포함하는 속성을 추가해야 한다. 단, 지시자를 가지지 않는다는 가정 하에서만 발생한다. 만약 Depth가 증가하면서 지시자도 증가하면, 테이블의 개수 또한 증가하게 된다. 하지만 영역 윈도우는 Depth의 증가와 상관없이 DTD 기반 TagID만을 유지하면 구조정보를 위한 추가 비용이 필요하지 않는다. 따라서 Depth와 지시자의 정도의 변화가 질의처리 시간에 영향을 준다는 것을 보일 수 있다.

• Fan-out과 지시자 변화에 따른 질의처리 시간 비교

Fan-out의 증가에 따라 Shared Inlining의 경우 하나의 테이블에 속하게 되는 엘리먼트의 수가 증가하게 된다. 단, 지시자를 가지지 않는다는 가정 하에서만 발생한다. 만약 자식노드의 수가 증가하면서 지시자 또한 증가하게 된다면, 하나의 테이블이 아닌 여러 개의 테이블로 자식노드들을 분할 저장 해야 한다. 하지만 영역 윈도우는 Fan-out의 증가와 상관없이 DTD 기반 TagID만을 유지하면 구조정보를 위한 추가 비용이 필요하지 않는다. 따라서 Fan-out과 지시자의 정도의 변화가 질의처리 시간에 영향을 준다는 것을 보일 수 있다.

• 평균 프레디킷 비교 영역 크기(엘리먼트 수) 비교

하나의 조인 질의에 대한 결과를 내기까지 비교해야 하는 엘리먼트의 수를 측정하여 단말노드를 영역 윈도우 단위로 처리하는 영역 윈도우 기법에서 실질적으로 질의처리 과정에 참여하게 되는 단말노드의 수와 테이블로 분할 저장하여 튜플 단위로 처리하는 Shared Inlining의 질의 범위를 비교한다. 질의의 단말노드까지의 접근을 위해 Shared Inlining의 경우 테이블 간 내부 조인을 통해 접근할 수 있다. 따라서 지시자가 증가할수록 테이블의 수가 증가하게 되어 내부 조인도 늘어나게 된다. 따라서 지시자의 정도에 종속적인 테이블 분할 저장과 영역 윈도우를 이용하는 것이 질의처리 시 접근하는 엘리먼트의 수에 영향을 준다는 것을 알 수 있다.

4.2 실험 환경

실험 환경은 펜티엄4의 CPU가 2.8GHz이며 RAM이 1GB인 윈도우 운영체제에서 Java SDK 1.4를 이용해 구현한다. 실험 데이타는 Depth가 4, Fan-out이 3인 Bose의 논문[8]에서 사용하는 XML 파일과 Depth가 8, Fan-out이 8인 SIGMOD Record의 XML 파일을 사용한다.

실험을 위한 가정으로 첫 번째는 DTD 종속적이므로 정확한 정보 교환을 위해 DTD는 공개되고, 구조정보 업데이트는 고려하지 않기 때문에 DTD는 수정되지 않는다. 두 번째는 동일한 실험 환경을 조성하기 위해 메모리에 저장된 데이타에 대한 질의처리를 기본으로 한다. 세 번째는 정확한 실험 측정을 위해 질의처리는 저장이 완료된 상태에서 실시한다. 네 번째는 질의처리에 필요한 정확한 시간 측정을 위해 데이타 및 질의 변환에 대한 비용은 질의처리 시간에서 제외하고 측정한다. 다섯 번째 질의의 필터링을 목적으로 하는 선택 부분에 속하는 경로는 최소한 2개 이상인 것으로 한다.

4.3 성능 측정

• 데이타 변환 처리 시간 비교

그림 14의 좌측 그래프는 XML 데이타를 DTD 기반으로 각각의 환경에 맞게 변환하여 저장하는 단계에 필요한 비용을 표현한 것이다. Shared Inlining은 증가한 테이블 수에 따른 별도의 저장이 필요하기 때문에 지시자의 포함 정도가 늘어날수록 급격한 증가를 보이고, Range Window는 XML 데이타에 순차적으로 TagID와 해당 영역 윈도우의 Window_TagID 정보만 추가 저장하기 때문에 별도의 지연시간이 없어 완만한 증가를 보인다. 따라서 지시자의 포함 정도가 큰 데이타를 변환해야 할 경우에는 영역 윈도우 기법이 더 효과적이다.

• 지시자 포함 정도의 변화에 따른 조인 프레디킷 비교 영역 크기 변화

그림 14의 우측 그래프는 영역 윈도우로 그룹화된 엘리먼트들에 의해 질의처리 과정에서 실행되는 프레디킷을 비교하기 위해 고려하는 엘리먼트 수가 감소하는 것을 보이고 있다. 그 이유는 질의에서 프레디킷 비교가 필요한 엘리먼트의 이름이 주어지고, 영역 윈도우 기법에서는 그 엘리먼트의 이름으로 TagID를 검색하여 TagID를 포함하는 해당 영역 윈도우를 한번에 골라낼 수 있다. 또한 프레디킷 비교 결과가 맞지 않는다면, 그 룹의 모든 엘리먼트는 비교 대상에서 한꺼번에 제외될 수 있다. Shared Inlining의 경우는 지시자가 0%인 경우 하나의 테이블에 모든 데이타를 저장할 수 있어 조건에 만족하는 엘리먼트를 내부 조인 없이 한번에 찾아

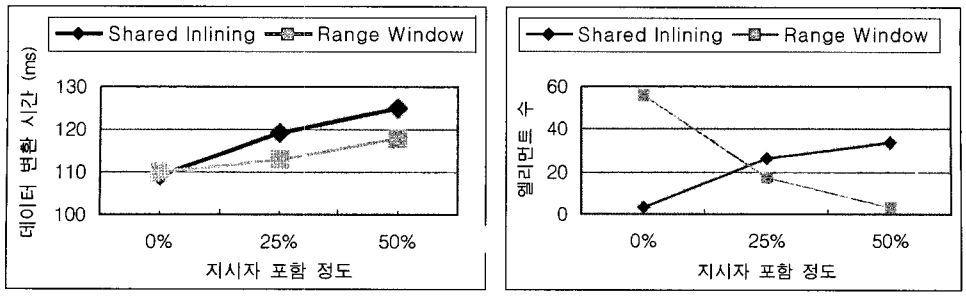

그림 14 데이타 변환 처리 시간(좌)과 평균 프레디킷 비교 영역(우) 비교

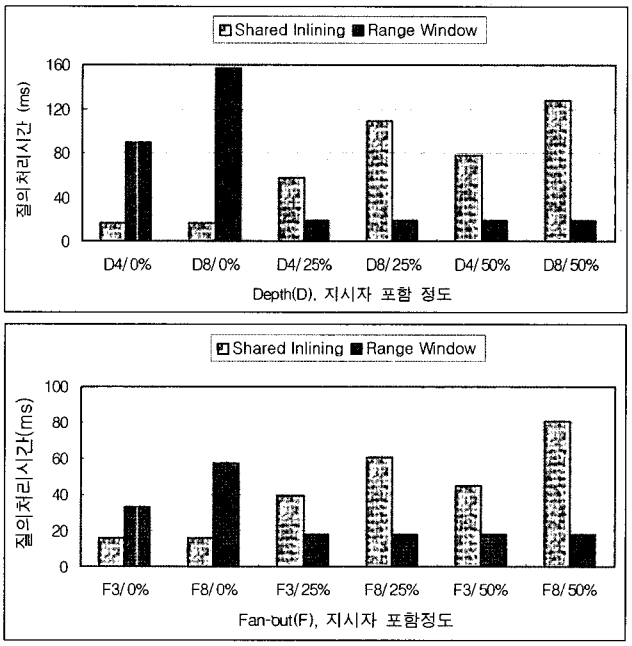

그림 15 Depth(위), Fan-out(아래)과 지시자의 포함 정도에 따른 처리시간 비교

낼 수 있다. 하지만 지시자의 포함 정도가 증가할수록 테이블 수도 증가하기 때문에 자식노드가 속한 테이블을 찾기 위해서는 부모노드의 테이블을 반드시 참조해야 한다. 따라서 지시자의 개수가 증가할수록 제안하는 기법이 질의에서 고려하는 엘리먼트의 수를 감소시킬 수 있으므로 효과적이다.

• Depth와 지시자의 포함 정도의 변화에 따른 처리 시간 비교

그림 15의 좌측 그래프는 Depth가 4, 8이고, 지시자 포함 정도가 0%, 25%, 50%로 증가할 때 질의처리 시간을 비교한 것이다. Shared Inlining의 경우 지시자의 포함 정도가 증가할수록 처리 시간도 증가한다. 그 이유는 Depth의 증가와 지시자에 따라 테이블의 증가로 인한 내부 조인 시간에 대한 오버헤드가 증가하기 때문이다. 그에 비해 영역 윈도우의 경우 지시자의 포함 정도가 0%인 경우 Depth의 증가에 따라 처리 시간이 증가하는 것을 볼 수 있다. 그 이유는 영역 윈도우 단위로 처리해야 하는데 그 영역이 존재하지 않기 때문에 Depth의 증가로 인한 질의에서 고려해야 하는 엘리먼트의 수 또한 늘어나기 때문이다. 따라서 일정한 수준의 지시자를 가지는 큰 사이즈의 XML 데이타의 조인 질의처리에서는 영역 윈도우를 이용하는 것이 효과적이다.

• Fan-out과 지시자의 포함 정도의 변화에 따른 처리 시간 비교

그림 15의 우측 그래프는 Fan-out이 3, 8이고, 지시자의 포함 정도가 0%, 25%, 50%로 증가할 때 질의처리 시간을 비교한 것이다. Shared Inlining의 경우 Fan-out의 증가와 지시자의 포함 정도에 비례하여 테이블 수가 증가하고 하나의 테이블에 속성으로 포함되는 엘리먼트의 증가로 인한 내부 조인 및 비교 시간에

대한 오버헤드가 발생하여 처리 시간이 증가한다. 그에 비해 영역 윈도우의 경우 지시자의 포함 정도가 0%일 때 Fan-out의 증가에 따라 처리 시간이 증가하는 것을 볼 수 있다. 그 이유는 앞의 실험에서와 마찬가지로 영역 윈도우가 존재하지 않아 영역 윈도우 단위의 처리가 불가능하기 때문에 Fan-out의 증가로 인해 질의에서 고려해야 하는 엘리먼트의 수 또한 증가하게 되어 질의 처리 시간이 증가한다.

위 실험을 통해 알 수 있듯이 Depth, Fan-out의 증가와 더불어 지시자 포함 정도가 큰 경우에는 Shared Inlining 보다 제안하는 Range Window 기법이 질의에서 고려해야 하는 엘리먼트의 수를 영역 윈도우 단위로 제외시킬 수 있어 조인 질의처리를 위한 비용을 감소에 효과적인 성능을 보인다는 것을 알 수 있다.

5. 결론 및 향후 연구

무한의 연속적인 스트리밍 데이타의 교환이 이루어지고 있는 환경에서 연산 비용이 비싼 조인 연산을 처리하기 위해 윈도우 개념의 시간 간격(time interval)을 이용하여 스트리밍 데이타 조인 영역을 윈도우(window)라는 작은 영역으로 나눠 제한하는 기법이 기존에 연구되어 왔다.

구조정보를 가지며, 구조 질의처리가 가능해야 하는 스트리밍 XML 데이타를 작은 영역으로 나눠 조인하기 위해서는 시간 간격이 아닌 구조정보를 기준으로 윈도우를 설정해야 한다. 스트리밍 데이타는 질의에 따라 다양한 시간 간격을 가지는 윈도우가 존재하기 때문에 다양한 윈도우 재평가가 필요하지만, 구조정보를 기준으로 나눠진 스트리밍 XML 데이타는 정적인 윈도우를 가지기 때문에 초기 설정된 윈도우가 다수의 조인 질의에 동일하게 적용될 수 있어 윈도우 재평가를 위한 과정이 필요하지 않다.

메모리 제약 사항을 최대한 활용할 수 있는 스트리밍 XML 데이타에 맞는 저장 기법을 통해 빠르게 조인 프레디킷을 만족하는 부분 스트리밍 데이타를 검색할 수 있어야 한다. 이 논문에서 제안하고자 하는 기법은 적은 메모리 사용을 위한 저장 기법을 위해 전체 스트리밍 XML 데이타를 저장하는 것이 아니라 텍스트(text) 즉, 조인 질의에서 프레디킷으로 사용될 수 있는 PCDATA 그리고 CDATA에 해당되는 부분만을 추출하여 저장한다. 기존의 XML 문서의 저장 기법들을 전체 데이타를 하나 또는 다수로 분할 저장 하는 기법들이 연구되었다. 하지만 스트리밍 환경에서의 전체 데이타를 저장한다는 것은 무한의 저장 공간을 필요로 하기 때문에 전체가 아닌 스트리밍 데이타의 일부분만을 추출하여 저장하는 제안 기법으로 저장 공간을 절약할 수 있다. 저장 측면뿐만 아니라 질의처리 비용을 고려해 본다면, 기존 기법의 경우 전체 또는 일부 경로에 대한 엘리먼트 각각을 검색하는 과정을 거쳐야 한다. 하지만 질의에서 실질적으로 요구되는 엘리먼트는 일부분에 지나지 않기 때문에 모든 엘리먼트를 순차적으로 검사한다는 것은 비효율적이다. 따라서 질의처리 비용을 줄이기 위한 방법으로 빠른 조인 프레디킷 비교를 위해 DTD의 구조정보 중 지시자 "*" 와 "+" 를 기초하여 스트리밍 XML 데이타의 영역 윈도우(Range Window)을 설정한 후 모든 질의처리 과정은 영역 윈도우 단위로 진행한다. 이와 같이 끊임없이 들어오는 스트리밍 XML 데이타를 작은 영역 윈도우로 쪼개 실질적으로 질의에서 필요로 하는 영역 윈도우를 대상으로 선택적으로 조인할 수 있게 된다. 따라서 조인 질의 프로세싱에 참여하는 스트리밍 XML 데이타의 영역을 윈도우 단위로 한번에 줄일 수 있기 때문에 기존 기법에 조인 비교 영역이 줄어들게 되어 빠르게 처리할 수 있다. 특히 Depth, Fan-out이 크고, 지시자를 다수 포함하는 대용량의 XML 데이타에서 효과적이다.

제안 기법의 가정으로 DTD의 업데이트는 고려하지 않고 있다. 만약 업데이트가 드물게 또는 빈번하게 업데이트되는 환경에서 DTD 구조정보의 변경이 정의된 TagID와 Window_TagID에 변화를 일으킨다. 따라서 TagID와 Window_TagID를 저장하는 텍스트 정보 테이블과 윈도우 정보 테이블의 업데이트에 소모되는 비용을 최소할 수 있는 기법 연구가 요구된다. 또한 현재 영역 윈도우 기법은 그림 2의 예제 질의에서 볼 수 있듯이 스트리밍 XML 데이타를 XML의 인스턴스 단위로 메모리에 저장하고 있다. 인스턴스의 크기에 능동적으로 적용할 수 있는 확장된 저장 기법의 연구가 필요하다.

참 고 문 헌

[1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani and Jennifer Widom, "Model and Issues in Data Stream Systems," In Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS), pp. 1-16, 2002.

[2] Lukasz Golab and M. Tamer Ozsu, "Processing Sliding Window Multi-Joins in Countinuous Queries over Data Streams," In Proceedings of the 29th VLDB Conference, pp. 500-511, 2003.

[3] Lukasz Golab and M.Tamer Özsu, "Data Stream Management Issues - A Survey," Technical Report, 2003. http://db.uwaterloo.ca/~ddbms/publications/stream/streamsurvey.pdf

[4] Lukasz Golab, Shaveen Garg and M. Tamer Özsu, "On Indexing Sliding Windows over On-line Data

Streams," In Proceedings of the International Conference on Extending DataBase Technology (EDBT), pp. 712-729, 2004.
[5] Jaewoo Kang, Jeffrey F. Naughton and Stratis D. Viglas, "Evaluating Window Joins over Unbounded Streams," In Proceedings of the IEEE International Conference on Data Engineering (ICDE), pp. 341-352, 2003.
[6] Richard Atterer, "Efficient Storage of XML Data Streams," http://atterer.net/uni/thesis/efficient-xml-storage.pdf
[7] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, "Extensible Markup Language (XML) 1.0," http://www.w3.org/TR/REC-xml/, World Wide Web Consortium (W3C), February 2004.
[8] Sujoe Bose, Leonidas Fegaras, David Levine and Vamsi Chaluvadi, "A Query Algebra for Fragmented XML Stream Data," In Proceedings of the 9th International Workshop on Data Base Programming Languages (DBPL), pp. 195-215, 2003.
[9] D. Florescu, D. Kossman, "Storing and Querying XML Data using an RDMBS," IEEE Data Engineering Bulletin 22(3), pp. 27-34, 1999.
[10] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David De Witt and Jeffrey Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," In Proceedings of the 25th VLDB Conference, pp. 302-314, 1999.
[11] Paul F. Dietz, "Maintaining order in a linked list," In Proceedings of the 14th Annual ACM Symposium on Theory of Computing, pp. 62-69, 1982.
[12] 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 the 2002 ACM SIGMOD international conference on Management of data, pp. 204-215, 2002.
[13] Quanxhong Li and Bongki Moon, "Indexing and Querying XML Data for Regular Path Expressions," In Proceedings of the 27th VLDB Conference, pp. 361-370, 2001.

박 석
정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조

김 미 선
2002년 숙명여자대학교 컴퓨터과학과(공학사). 2004년 서강대학교 컴퓨터학과(공학석사) 2005년 1월~ 현재 LG전자 단말연구소 연구원. 관심분야는 XML 데이타 처리 및 데이타베이스, 실시간 스트리밍 데이타 처리 임