분산 환경에서 통합 XQuery 질의 처리를 위한 조인과 통신비용에 대한 연구

최성일[0], 박종현, 강지훈
충남대학교 전기정보통신공학부 컴퓨터전공
{sichoi[0], jonghyunpark, jhkang}@cnu.ac.kr

A Study of Join and Communication Cost for processing Integrated XQuery queries over Distributed Environment

Seong-Il Choi[0], Jong-Hyun Park, Ji-Hoon Kang
Dept. of Computer Science & Engineering, ChungNam National University

요 약

XML은 웹 상에서 정보교환의 표준이며, 이종의 데이터를 보유하고 있는 지역 시스템들은 XML View를 이용하여 정보를 공개한다. 사용자는 XML을 위한 표준 질의어인 XQuery를 사용하여 분산된 XML View들을 대상으로 통합 XQuery질의를 생성할 수 있다. 이렇게 생성된 통합 XQuery질의는 자연스럽게 다른 지역시스템들 사이의 조인을 포함 할 수 있다. 조인은 비용이 많이 드는 연산자이므로 조인을 효율적으로 처리하는 것은 전체 질의의 성능에 큰 영향을 준다. 그러므로 조인을 효율적으로 처리하기 위한 비용을 결정하는 연구가 중요하다고 할 수 있다. SQL에서는 이와 같은 연구들이 많이 존재하며, 분산 환경에서의 조인을 효율적으로 처리하기 위해 크게 두 가지 비용을 고려한다. 그 중 하나는 지역시스템 내에서 조인을 처리하는 조인비용이며, 나머지 하나는 조인을 수행하기 위하여 다른 지역시스템으로 데이터를 전송하는 통신비용이다. 이 두 비용은 분산 조인을 처리하기 위한 중요한 요소이므로[6,7,8] 이를 측정하는 것은 통합 질의 처리를 위해서 필요한 연구라 할 수 있다. 그러나 테이블 구조의 데이터를 검색하기 위한 SQL 의 방법들을 구조적인 XML 데이터를 검색하기 위한 XQuery질의를 위해서 그대로 사용하는 것은 어려운 일이다. 본 논문에서는 분산 환경에서 통합 XQuery질의의 조인을 효율적으로 처리하기 위해 조인비용과 통신비용을 측정하는 방법을 제안한다. 본 논문의 기여는 앞서 기술한 문제점을 해결하여, XQuery 질의의 조인을 효율적으로 처리하기 위한 비용을 측정할 수 있도록 한다.

1. 서 론

XML은 인터넷 상에서 데이터 교환을 위한 표준이다. 이종의 데이터를 보유하고 있는 응용들은 이종 데이터들을 XML 데이터처럼 바라볼 수 있도록 하기 위하여 XML View를 사용한다. XML View는 DTD[1], XML Schema[2], XQuery[3] 등을 이용하여 기술할 수 있으며[4][5], 본 논문에서 사용하는 XML View는 XQuery로 기술되어 있다고 가정한다[4]. 사용자는 분산된 XML View들을 대상으로 질의할 수 있으며, 이 때 검색을 위하여 XQuery 질의어를 사용하는 것은 상호운용성을 위한 자연스러운 선택이다. 본 논문에서는 분산된 다수의 XML View들을 대상으로 작성된 XQuery 질의를 통합 XQuery 질의라 부르며, 이 질의는 여러 지역 시스템들 사이에 조인 연산을 포함할 수 있다. 조인 연산은 질의 처리를 위하여 비용이 많이 드는 연산자들 가운데 하나이므로 효율적인 질의의 처리를 위하여 중요하게 고려되는 요소들 가운데 하나이다. 그러므로 통합질의의 조인을 효율적으로 처리하기 위한 비용들을 연구하는 것 역시 중요하다.

본 논문에서는 통합 질의를 효율적으로 처리하기 위한 다음의 두 비용을 고려한다. 첫째, 지역시스템 내에서 조인을 처리하기 위한 조인비용을 고려한다. 이 비용은 최소의 조인비용을 갖는 실행계획 생성 시 중요한 요소가 된다. 둘째, 지역 시스템들 사이의 데이터 전송량을 위한 통신비용을 고려한다. 분산 환경에서 조인을 처리하기 위해서는 지역 시스템들 사이의 데이터 전송이 불가피하므로, 이 비용은 분산 환경에서의 통합 질의 처리를 위한 중요한 요소라 할 수 있다. SQL에서는 위의 두 비용을 결정하는 연구가 이미 많이 수행되어 있다. 하지만 XQuery를 위하여 SQL의 방법을 적용하기 위해서는 고려해야 할 사항이 존재한다. 그 이유는 SQL은 그 질의 대상이 데이터베이스의 평평한 구조인 테이블이고, XQuery는 그 질의 대상이 반 구조적인 XML로 다르기 때문이다.

본 논문에서는 분산 환경에서 통합 XQuery질의를 처리하기 위한 조인비용과 통신비용을 측정하는 방법을 제안한다. 본 논문의 기여는 통합 XQuery 질의를 위하여 위의 두 비용을 적용할 수 있도록 하는 것이다.

본 논문의 구성은 다음과 같다. 2절에서는 관련연구들

을 소개하고, 3절에서는 조인비용과 통신비용을 설명하기 위한 배경지식을 기술한다. 4절에서는 XQuery를 위한 조인비용을 측정하는 방법을 기술하고, 5절에서는 XML의 특성을 고려한 통신비용을 측정하는 방법을 설명한다. 마지막으로 6절에서는 결론과 향후 연구를 기술한다.

2. 관련연구

SQL의 경우, 분산 환경에서 통합질의를 효율적으로 처리하기 위한 연구들이 많이 존재한다[6][7][8]. [6],[7]에서는 최소의 비용을 갖는 질의 실행계획을 선택하기 위해 조인비용과 통신비용을 고려한다. [8]에서는 분산 환경에서 최적의 조인 순서를 결정하기 위해서 통신비용을 고려한다. [6],[7]의 조인비용은 조인을 처리하기 위해 입력되는 데이터의 크기와 출력되는 데이터의 크기로 계산되며, 데이터의 크기는 튜플의 수와 튜플의 평균크기의 곱으로 결정한다. 하지만 우리는 노드의 수만을 안다고 가정하므로 입출력 비용 대신에 조인 연산을 수행하는 CPU의 비용으로 결정한다. 본 논문에서의 XQuery를 위한 조인비용은 노드들의 수를 고려하여 결정한다. [6],[7],[8]의 통신비용은 전송되는 데이터들의 크기로 계산되고 우리는 전송되는 노드의 수로 계산한다. [6][7][8]의 방법은 그 질의 대상이 평평한 구조의 테이블이고 우리가 사용하는 XQuery는 반 구조적인 XML을 질의 대상으로 하기 때문에, 구조적인 차이가 발생한다. 데이터베이스의 테이블은 칼럼들이 서로 일대 일 관계에 있지만, XML은 각 노드들이 서로 다대 다, 다대 일, 일대 다 등 다양하게 나타날 수 있다. 그러므로 전송될 데이터의 수를 결정할 때, 조인 조건을 만족하는 다른 노드들의 수를 계산해야 하므로, 이와 같은 구조적인 관계를 고려해야 한다. 우리의 통신비용 계산방법은 XML View를 이용하여 구조적인 관계를 찾아내고, 적용한다.

3. 배경지식

3.1 XML View의 특성

XML View는 XQuery로 기술되어 있으므로, FLWOR표현으로 구성된다[3]. 이 표현으로 크게 두 가지를 추정할 수 있다. 첫째, View의 각 노드들의 구조를 알 수 있고, 둘째, 각 노드와의 관계를 알 수 있다. View의 구조는 Return절의 구조를 통해 알 수 있고, 각 노드들의 관계는 For절, Let절, Nested FLWOR절을 통해서 알 수 있다. 각 관계는 Optional character로 표현할 수 있고 다음과 같다.

- 1 : one
- ? : zero or one
- * : zero or one more ('+'포함)

이 Optional character는 XQuery의 FLWOR절의 특성으로 구할 수 있다. XQuery의 For절과 Let절의 목적은 tuple stream을 생성하는 것이다. For절과 Let절은 하나의 변수와 Expression으로 구성되며, Expression에 Nested FLWOR절이 나올 수 있다. For절은 [표 1]과 같이 Expression에 포함된 sequence만큼 반복하여 tuple stream을 생성한다. Let절은 [표 2]와 같이 Expression에 포함된 sequence를 반복 없이 For절에 의해 생성된 tuple stream에 추가한다. 이 특성이 For절과 Let절의 큰 특성이며, 우리의 알고리즘에 핵심 역할을 한다. Where절과 Return절은 tuple stream안의 각 tuple에 대해 한번 수행된다[3].

[표 1] For절의 예제 XQuery질의와 결과

```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

```
<out> <one/> </out>
<out> <two/> </out>
<out> <three/> </out>
```

[표 2] For절의 예제 XQuery질의와 결과

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

```
<out>
      <one/> <two/> <three/>
</out>
```

Optional character는 Return절에 있는 변수가 For절로 선언되어 있으면 '1', Let절로 선언되어 있으면 '?', Nested FLWOR절이 있으면 *'로 결정할 수 있다.

3.2 XML View의 예

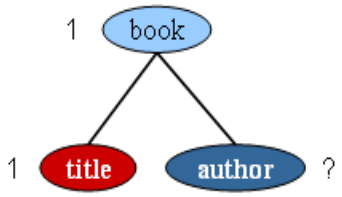
[표 3]은 XML View의 예를 보여주고, 그림. 3은 XML View의 구조를 보여준다.

[표 3] XML View의 예

```
<Results>{
for $book in doc("bib.xml")/bib/book
for $title in $book/title
let $author := $book/author
return
<book>
    <title>{$title/text()}</title>
    <author>{$author/text()}</author>
</book>
}</Results>
```

[표 3]의 XML View의 Return절을 통해 [그림 1]과 같은 모습으로 구조를 예측할 수 있다. book 노드의 자식은 title노드와 author노드임을 알 수 있다. 그리고 book노드와 title노드는 For절로 선언되어 있고, author 노드는 Let절로 선언되어 있으므로, book과 title의 Optional character는 '1', author는 '?' 관계가 된

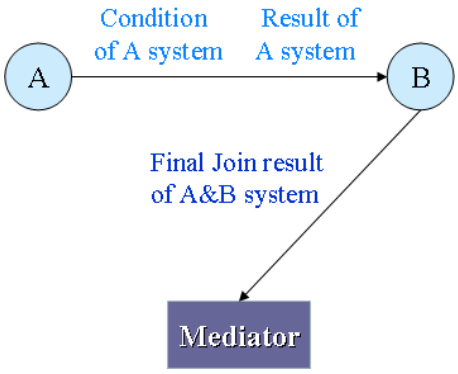
다. 즉, 두 개의 For절과 한 개의 Let절로 구성된 Tuple stream의 반복 마다 return절이 한 번씩 수행 될 때, 위의 구조가 반복되어 나타난다. 그 반복마다 book과 title은 항상 한 번씩 생성되며, author는 생성되지 않거나 한 번 생성된다.



[그림 1] XML View의 구조

3.2 분산 환경에서의 조인수행방법

분산 환경에서 조인을 수행하기 위해서는 어느 한쪽의 데이터를 전송하는 것이 필수적이다. 그 이유는 A 시스템과 B 시스템과의 조인이 있을 때, 이 조인을 수행하려면 A 시스템의 데이터와 B 시스템의 데이터가 같은 시스템에 있어야 가능하기 때문이다.



[그림 2] 분산조인 수행방법

[그림 2]는 분산 환경에서 조인을 처리하는 방법을 보여준다. A와 B는 지역 시스템이며, Mediator는 사용자의 통합질의를 입력받고 최종 결과를 출력하는 시스템이다. [그림 2]는 A와 B사이에 조인이 존재할 때, A의 데이터를 B로 이동시킨 후, B에서 조인을 처리하고 최종결과를 Mediator로 전송하는 모습을 보여준다. 이때, 전송되는 데이터는 조인을 수행하는 데이터와 조인을 만족하였을 때 최종결과에 반환되어야 하는 데이터를 포함한다. 이때, 조인비용은 B 시스템에서 A의 데이터와 B의 데이터를 조인하는 비용이며, 통신비용은 A의 조인을 수행하는 데이터와 반환해야하는 데이터를 전송하는 비용이며, 조인 수행 후에 Mediator로 최종결과를 전송하는 비용이다.

4장과 5장에서는 조인비용과 통신비용을 계산하는 방법을 기술한다. 우리는 이를 계산하기 위해서 XML View가 나타내고 있는 모든 노드들의 수를 알 수 있다고 가정한다.

4. 조인비용

4.1 조인비용 계산

통합질의를 효율적으로 처리하기 위한 첫 번째 비용은 조인비용이다. 이 절에서는 기존의 SQL에서의 조인비용 계산하는 방법을 XQuery에 적용하고, 구조적인 차이에 의해 발생되는 문제점들을 해결한다.

XQuery 질의의 조인비용은 조인연산의 비교횟수로 결정한다. 예를 들어 $A⋈$B의 조인이 있을 때, 조인비용은 $A의 노드 수와 $B의 노드 수를 곱하여 계산한다. 그 이유는 3.1의 XML View의 특성에서 보는 바와 같이 XQuery의 Where절은 For절에 의한 반복에 영향을 받기 때문이다.

$A⋈$B의 조인비용 : |$A| * |$B|

다음의 예제를 통해 조인비용을 측정하는 방법을 적용하는 예를 보인다. [표 4]의 예제는 3개의 A, B, C 지역 시스템의 XML View를 기반으로 작성된 통합 질의이다. 각 A, B, C 시스템에 대하여 각각 $A, $B, $C를 For절로 선언하였다.

[표 4] 통합 XQuery 질의의 예

For $A in *Path Expression of $A* For $B in *Path Expression of $B* For $C in *Path Expression of $C* Where $A/@id = $B and $B = $C/@id Return <result>{$A/C, $C/D}</result>
|$A/@id\| = 10, \|$B| = 5, |$C/@id\| = 30 \|$A/C| = 10, |$C/D| = 60 *|X| = X의 노드의 수*

위와 같이 예제 XQuery질의와 각 변수들의 노드 수가 주어졌을 때, 조인비용을 결정하는 방법은 다음과 같다. 예제 질의의 Where절에 '$A/@id = $B and $B = $C/@id' 와 같이 2개의 조인이 있다. 이 중 첫 번째 조인인 '$A/@id = $B' 의 조인비용을 계산한다.

조인비용 = |$A/@id| * |$B| = 10 * 20 = 200

$A/@id의 노드 수와 $B의 노드수를 곱하여 위와 같이 조인비용을 결정할 수 있다.

4.2 조인 결과의 크기

조인비용을 결정한 후, 그 다음은 조인을 만족하는 결

과의 크기를 측정해야 한다. 그 이유는 다음 조인을 수행하기 위해서 현재 조인을 만족하는 노드와 그와 같이 반환해야하는 노드들을 다음 시스템으로 전송해야 하기 때문이다. 조인 결과의 크기를 측정하기 위해 조인 선택치를 사용하는데, 이는 SQL에서의 방법을 적용한 것이며, [표 5]에 기술되어 있다[9].

[표 5] 조인 선택치의 3가지 계산방법

1. 한쪽 노드가 ID(Key)일 때.
   - $A/text() = $B/@id
   - Join Selectivity (js) = 1/|id| = 1/|$B/@id|
2. A,B가 모두 ID일 때.
   - $A/@id = $B/@id
   - Join Selectivity (js) = 1/max(|$A/@id|,|$B/@id|)
3. A,B가 모두 ID가 아닐 때.
   - $A/text() = $B/text()
   - Join Selectivity (js) = 1/max(|$A|,|$B|)

$A⋈$B의 조인 선택치는 표 5의 세 가지 경우를 고려하여 결정한다. |$A|와 |$B|의 조인 선택치를 JS_{ab}로 표현한다. $A⋈$B의 조인 결과의 크기는 다음과 같이 계산한다.

$$|\$A⋈\$B| = JS_{ab} * |\$A| * |\$B|$$

다음의 예제를 통해 조인결과의 크기를 결정하는 방법을 적용하는 예를 보인다. [표 4]의 예제 질의에서 '$A/@id = $B'의 조인 결과의 크기를 구하는 방법을 설명한다. $A와 $B의 JS_{ab}는 [표 5]의 첫 번째 경우에 속하므로 1/|$A/@id|의 값으로 결정된다.

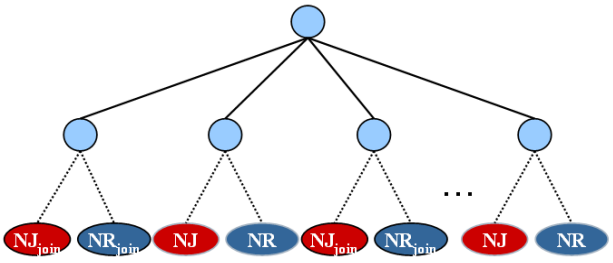
$$|\$A/@id⋈\$B| = JS_{ab} * |\$A/@id| * |\$B|$$
$$= 1/10 * 10 * 5 = 5$$

5. 통신비용

통합질의를 효율적으로 처리하기 위한 두 번째 비용은 통신비용이다. 통신비용은 조인을 위해 전송되는 데이터의 크기로 결정된다. 우리는 XML View에서 공개하는 노드들의 수를 안다고 가정하기 때문에, 통신비용은 노드의 수로 결정한다. 통신비용을 결정하는 방법은 다음과 같다.

통신비용 = A + X
X = $|NJ_{join}|$ + $|NR_{join}|$
NJ = 조인을 수행하는 노드
NR = 반환해야하는 노드
NJ_{join} = NJ중에서 조인 조건을 만족하는 노드
NR_{join} = NR중에서 조인 조건을 만족하는 노드

A는 통신을 연결하기 위한 초기 시간이며, X는 전송되는 노드의 수이다. 하지만 A의 값은 극히 적은 시간이므로 A는 0으로 가정한다. X는 NJ_{join}의 수와 NR_{join}의 수의 합으로 결정한다. [그림 3]은 NJ와 NR과의 관계를 보여주며, 이 관계는 XML View를 이용하여 알 수 있다. NJ는 조인을 수행하는 노드이며, NR은 통합질의의 Return절에 속한 변수가 지정하는 노드이다. 이 두 노드는 통합질의의 조인을 수행하고 최종 결과를 생성할 때, 필요한 노드들이므로 전송이 필수적인 노드들이다. 첫 번째 조인을 하기위해 전송되는 X의 값을 결정할 때, NJ와 NR의 합으로 결정한다. 그리고 첫 번째 조인을 수행한 다음은 NJ와 NR중에서 조인 조건을 만족하는 값을 X로 결정한다. 그 값은 NJ_{join}과 NR_{join}으로 표현하며, [그림 3]과 같이 표현된다. 다음은 이 값들을 결정하는 방법을 기술한다.



[그림 3] NJ와 NR의 구조

NJ_{join}은 4.2절에 기술된 조인 결과의 크기로 구하고 NR_{join}은 NJ_{join}의 값을 이용하여 구한다. NR_{join}을 구하는 방법은 다음과 같다. 각 NJ와 NR은 XML View에서 For절이나 Let절로 표현될 수 있으므로, 그에 따라 각각 Optional character를 구할 수 있게 된다. [표 6]과 같이 NJ와 NR의 Optional character에 따라 네 가지 경우로 나눌 수 있다.

[표 6] NR 측정 방법의 4가지 경우

- ❍ Case 1 - 1 : 1
  - NJ : For, NR : For
  - $NR_{join} = NJ_{join}$
- ❍ Case 2 - 1 : ?
  - NJ : For, NR : Let
  - $|NR_{join}| = |NR| * |NJ_{join}|/|NJ|$
- ❍ Case 3 - ? : 1
  - NJ : Let, NR : For
  - $NR_{join} = NJ_{join}$
- ❍ Case 4 - ? : ?
  - NJ : Let, NR : Let
  - Min : $|NR_{join}| = 0$
  - Max : $|NR_{join}| = |NR| * |NJ_{join}|/|NJ|$
  - NR_{join} = (Min + Max) / 2

Case 1은 NJ와 NR가 모두 For절로 선언된 경우이다. 이 경우는 tuple stream의 반복마다 NJ와 NR가 모두 한 번 생성되므로, NR_{join}은 NJ_{join}의 수와 같다. Case 2는NJ는 For절, NR는 Let절로 선언된 경우이므로, tuple stream의 반복마다 NJ는 한번, NR는 존재하지 않거나 한 번 생성된다. 이 경우는 NJ에 대한 NR의 비율을 명확하게 알 수 없기 때문에, NJ가 조건을 만족하는 확률($|NJ_{join}|/|NJ|$)을 NR에 적용하여 $|NR_{join}|$를 구한다. Case 3은 NJ가 Let절, NR이 For절로 선언된 경우이므로, tuple stream의 반복마다 NJ는 존재하지 않거나 한번 생성되고, NR는 한번 생성된다. 이 경우는 NJ가 조건을 만족하는 경우 NR는 항상 있으므로, NJ_{join}의 수만큼 NR_{join}도 결정된다. 마지막으로, Case 4는 NJ와 NR가 모두 Let절인 경우이므로, tuple stream의 반복마다 NJ와 NR가 각각 존재하지 않거나 한번 나타난다. 이 경우는 NJ와 NR가 모두 연관되지 않을 수도 있고 모두 연관될 수도 있다. 따라서 우리는 이 두 경우의 평균값으로 NR_{join}값을 추정한다.

6. 결론

우리는 통합 XQuery질의를 효율적으로 처리하기 위해 중요하게 고려되는 조인비용과 통신비용을 측정하는 방법을 제안했다. 기존의 SQL의 방법을 XQuery에 적용하고, 테이블과 XML과의 구조적인 차이로부터 발생하는 문제점을 해결하였다. 본 논문의 기여는 XQuery에 대하여 조인비용과 통신비용을 적용할 수 있도록 하였다. 우리의 향후연구는 통합질의의 실행계획을 생성할 때 위의 두 비용을 이용하여 결정하는 것이다.

7.Acknowledgement

본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스컴퓨팅및네트워크원천기반기술개발사업의 지원에 의한 것임.

8.참고문헌

[1] W3C, Extensible Markup Language (XML) 1.1 (Second Edition) W3C Recommendation 16 August 2006, (http://www.w3.org/TR/2006/REC-xml11-20060816).

[2] W3C, XML Schema Part 0: Primer Second Edition W3C Recommendation 28 October 2004, (http://www.w3.org/TR/2004/REC-xmlschema-0-20041028)

[3] W3C, XQuery 1.0: An XML Query Language, W3C Recommendation 23 January 2007,(http://www.w3.org/TR/2007/REC-xquery-20070123).

[4] S. Groppe, & S. Bottcher, "Schema-based Query Optimization for XQuery Queries", Proc. ADBIS 2005, Tallinn, Estonia, September, 2005.

[5] I. Manolescu, D. Florescu & D Kossmann, "Answering XML Queries over Heterogeneous Data Sources," Proc. 27th International Conference on Very Large Data Bases, Roma, Italy, pp. 241-250, September 11-14, 2001.

[6] M. J. Yu & P. C.-Y. Sheu,"Adaptive Join Algorithms in Dynamic Distributed Databases", Distrib. Parallel Databases, Vol. 5, No. 1, pp. 5-30, January, 1997.

[7] L. Liu, C. Pu & K. Richine, "Distributed Query Scheduling Service: An Architecture and Its Implementation," IJCIS, Vol. 7, No. 2-3, pp. 123-166, 1998.

[8] I. Eldosouky, H. Arafat, & A. A. Eldin, "New Heuristic Approaches for Improving Distributed Query Processing based on The Enhancement of Semi-Join Strategies", Proc. the International Conference on Statistics, Computer Science, and Operational Research, Egypt Dec, 2001.

[9] S. B. Navathe & R. Elmasri, "Fundamentals of Database Systems," fourth edition, Addison Wesley, 2003.