韓國컴퓨터情報學會 論文誌
第14卷 第3號, 2009. 3.

2009-14-3-1-1

분산 환경에 질의 최적화를 위한 XQuery 질의 재작성

박 종 현*, 강 지 훈**

XQuery Query Rewriting for Query Optimization in Distributed Environments

Jong-Hyun Park *, Ji-Hoon Kang **

요 약

XQuery가 XML 데이터를 위한 표준 질의어로 제안되면서, XQuery를 효율적으로 처리하기 위한 연구는 새로운 연구의 주제가 되었고, 몇몇 연구자들은 XQuery 질의를 최적화하기 위한 방법을 제안하고 있다. 그러나 앞선 대부분의 연구들은 XML 데이터 관리 시스템에 특화된 최적화 규칙만을 정의하고 있을 뿐, 어떠한 시스템에서도 일반적으로 사용할 수 있는 최적화 방법과는 거리가 멀다. 또한 앞선 몇몇 연구에서는 XML 스키마 또는 DTD와 같은 미리 정의된 XML 데이터의 구조정보를 이용하여 최적화하는 방법을 제안하고 있다. 그러나 현재 모든 응용이 XML 데이터를 위한 구조정보를 포함하고 있지는 않은 것이 현실이다. 그러므로 본 논문에서는 XQuery 질의의 특성을 파악하고 XQuery 질의 자체만을 이용한 최적화 방법들을 제안한다.

본 논문에서는 XQuery 질의의 특성들을 고려한 세 가지 XQuery 질의를 최적화 방법을 제안한다. 첫 번째 방법은 XQuery 질의에 존재하는 불필요한 표현을 제거하는 것이고, 두 번째 방법은 질의 재배치를 이용한 최적화 방법이다. 마지막으로 세 번째 방법은 XQuery가 For절에 의해서 중첩된다는 점을 고려하여 For절에 의해서 발생하는 불필요한 반복을 최소화하는 방법이다. 성능 평가를 통해 논문에서 제안한 방법들에 의해 재작성된 질의의 처리 시간은 원본 질의의 처리 시간보다 뛰어나다는 것을 알 수 있다. 또한 각 방법들은 독립적으로 수행될 수 있으므로 XQuery 엔진의 필요에 따라 개별적으로 사용이 가능하다.

Abstract

XQuery query proposed by W3C is one of the standard query languages for XML data and is widely accepted by many applications. Therefore the studies for efficient Processing of XQuery query have become a topic of critical importance recently and the optimization of XQuery query is one of new issues in these studies. However, previous researches just focus on the optimization techniques for a specific XML data management system and these optimization techniques can not be used under the any XML data management systems. Also, some previous researches use predefined XML data structure information such as XML schema or DTD for the optimization. In the real situation, however, applications do not all refer to the structure information for XML data. Therefore, this paper analyzes only a XQuery query and optimize by using itself of the XQuery query.

• 제1저자 : 박종현 교신저자 : 강지훈
• 투고일 : 2008. 9. 4, 심사일 : 2008. 9. 30, 게재확정일 : 2009. 2. 10.
* 거제대학 조선정보계열 초빙교수 ** 충남대학교 전기정보통신공학부 교수
※ 본 연구는 지식경제부 및 정보통신연구진흥원의 지원을 받아 수행되었음(B1120-0901-0002, IT특화연구소설립: "유비쿼터스 신기술 연구센터"설립 및 운영)

In this paper, we propose 3 kinds of optimization method that considers the characteristic of XQuery query. First method removes the redundant expressions described in XQuery query, second method replaces the processing order of operation and clause in XQuery query and third method rewrites the XQuery query based on FOR clause. In case of third method, we consider FOR clause because generally FOR clause generates a loop in XQuery query and the loop often rises to execution frequency of redundant operation. Through a performance evaluation, we show that the processing time for rewritten queries is less than for original queries. also each method in our XQuery query optimizer can be used separately because the each method is independent.

▸ Keyword : 엑스쿼리(XQuery), 질의 최적화(Query Optimization), 질의 재작성(Query Rewriting), 질의 정재(Query Refinement)

I. 서 론

W3C는 XML 데이터 검색을 위한 여러 가지 질의어들의 특성을 취합하여 XQuery를 XML 데이터를 검색하고 통합하기 위한 표준으로 제안하였으며 현재 Version 1.0이 완성되어있다[1]. XQuery에 대한 관심이 증가함에 따라 기존의 다른 질의어들과 마찬가지로 여러 응용에서 이를 효율적으로 처리하고자 하는 연구가 진행되고 있으며, 그 중 대표적인 한 분야가 XQuery 질의 최적화이다. 현재 XQuery 질의를 최적화하기 위한 연구는 초기단계에 있으며, 많은 연구가 기존에 존재하는 질의 최적화 방법을 XQuery 질의를 최적화하기 위한 방법에 적용하기 위한 시도 중에 있다. 특히 데이터베이스 표준 질의어인 SQL 질의를 최적화하기 위한 방법을 XQuery 질의를 최적화하기 위한 방법으로 사용하기 위한 연구는 매우 활발하게 진행 중에 있다[2, 3, 4]. 이러한 접근법이 가능한 이유는 XQuery의 문법이 SQL과 유사하기 때문이다. 그러나 이들 대부분 방법은 단순히 SQL 질의의 최적화 방법을 XQuery 질의를 위하여 적용할 수 있다는 것만을 기술하고 있을 뿐, 구체적인 언급은 없는 실정이다. 또한, 이들 대부분 방법들은 XML 데이터 관리 시스템의 일부로 사용되기 위한 방법이므로 응용의 고유한 저장 방법이나 인덱스 구조에 의존적인 방법들이다. 본 논문에서 고려하는 환경은 각 응용들이 어떠한 종류의 데이터를 관리하며 어떤 저장 구조를 갖는지 등과 무관하게 단순히 해당 응용이 XQuery를 지원한다는 가정을 기반으로 응용에 질의하기 위한 XQuery 질의를 최적화하는 것을 그 목표로 한다.

논문에서는 3가지 방법으로 XQuery 질의를 최적화하기 위한 방법을 제안한다. 첫째, XQuery 질의에 불필요한 표현을 제거하여 최적화를 수행하는 방법과, 둘째 질의 처리의 순서 교환을 통한 최적화 방법, 셋째 XQuery 질의에서 사용되는 FOR절을 기반으로 질의를 재작성하는 것에 의한 최적화 방법이다. 본 논문에서 제안하고 있는 최적화 방법들은 개별적으로 구현되었으므로 응용의 필요에 따라 모듈 별로 사용이 가능하다. 논문에서 제안하고 있는 몇몇 방법들은 컴파일러 연구 분야에서 질의의 최적화를 위해서 이미 연구되고 검증된 방법[5]이지만 특별히 XQuery 질의의 처리를 위해서 적용할 수 있다는 것을 실험을 통하여 증명한다. 제안한 최적화 방법들은 성능 평가를 위하여, Oracle XQuery 엔진을 이용하여 원본질의와 최적화 된 질의 사이의 수행 시간 측정을 통한 성능을 평가 한다.

II. 관련 연구

XQuery 질의를 최적화하기 위해서 제안된 방법들 가운데 가장 많은 접근 방법은 SQL 질의를 위한 최적화 방법을 XQuery 질의의 최적화를 위하여 사용하기 위한 연구이다[2, 3, 4]. SQL 질의에 대한 최적화는 이미 많은 시간 동안 연구되었으며, 그 방법 또한 다양하므로 SQL 질의를 위한 최적화 방법을 XQuery 질의를 위해서 사용하기 위한 연구는 매우 바람직해 보인다. 또한, XQuery의 문법은 많은 부분이 SQL 질의와 유사하다. 그러므로 SQL 질의를 위한 최적화 방법들 중 상당부분은 XQuery 질의를 위해서 사용될 수 있을 것으로 사료된다. 그러나 이들 방법은 아직까지 초기 연구 단계이므로 SQL 질의의 최적화 방법 중 어떤 방법이 XQuery 질의의 최적화를 위해서 사용 가능하다고 기술하고 있을 뿐, 구체적인 언급은 없는 실정이다. [2]와 같은 경우, XQuery 질의의 최적화를 위하여 SQL과 OQL질의의 최적화를 위한 방법으로부터 얻어진 열네 개의 XQuery 질의 최적화 방법을 제안하고 있다. 그러나 앞서 언급한 것처럼 열네 개의 최적화 방법 각각은 XQuery 질의를 최적화하기 위해서 모두 사용이 가능하지만 각 방법이 어떤 형태의 XQuery 질의를 최적화하

기 위해서 사용이 가능한지는 구체적으로 기술하고 있지 않다. 그 이유는 관계형 데이터의 저장 구조는 이미 정해져 있지만 트리 구조의 XML 데이터를 저장하기 위한 방법은 아직까지 결정되지 않는 상태이므로 저장 방법에 따라 어떤 방법은 효율적일 수 있지만 어떤 방법은 적용조차 할 수 없기 때문이다. 그러므로 본 논문에서는 구체적으로 어떤 종류의 XQuery 질의를 최적화할 수 있는지 정의한다. 또한 논문에서는 컴파일러 분야에서 이미 검증된 질의 최적화 방법들[5] 가운데 XQuery 질의에 적용할 수 있는 개념들을 취하여 해당 방법들이 XQuery 질의의 최적화를 위해서 어떻게 작용하는지 실험을 통해 보인다.

[3, 4, 6]에서는 XQuery 질의의 최적화를 위하여 XAT라는 XQuery 질의 모델을 정의하고 XQuery 질의 계획을 생성한 후 이를 최적화하기 위한 방법을 제안하고 있다. 이 방법은 입력 질의의 대상이 되는 XQuery 뷰를 기반으로 XQuery 질의 계획 가운데 불필요한 연산을 제거하여 질의를 최적화 하는 것을 그 목적으로 한다. [7, 8, 9, 10, 11, 12] 역시 XQuery 형태의 뷰는 아니지만 미리 정의된 XML 데이터의 정보를 이용하여 XQuery 또는 XPath 질의를 최적화하기 위한 방법을 제안하고 있다. 그러나 논문에서 고려하는 환경에서는 질의 최적화기가 항상 질의 대상의 저장 뷰 또는 XML Schema, DTD와 같은 정보를 참조할 수 없다는 점을 고려하여 XQuery 질의 자체만을 이용하여 최적화 하는 방법을 제안한다. 이는 어떠한 형태의 XQuery 질의를 위해서 사용이 가능할 뿐만 아니라 앞서 언급한 연구들에서 제안한 방법들과 함께 사용될 수 있으므로 최적화 능력을 극대화할 수 있을 것으로 기대된다.

III. 통합 XQuery 질의 분할의 고려사항

1. 불필요한 표현의 제거를 이용한 최적화

XQuery 질의를 최적화하기 위한 첫 번째 방법은 XQuery 질의 내부에 불필요한 표현을 제거하는 것으로 다음과 같은 세 가지 표현을 제거한다.

- 사용하지 않는 LET절의 제거
- 중복된 XPath 표현의 제거
- WHERE절의 조건 중 동치 또는 포함관계의 연산 중 의미적으로 중복된 부분을 제거

표 1. 잉여의 LET절 제거
Table 1. Removing Redundant LET clauses.

Original XQuery Query	〈Results〉{ FOR $a in document("a.xml")/a FOR $z in document("x.xml")/x/y/z **LET $c := $a/b/c** WHERE $a/text()=$z/text() RETURN 〈Result〉{$a, $z}〈/Result〉 }〈/Results〉
Simplified XQuery Query	E1(FOR a in E2 FOR z in E3 **LET c := E4(a)** WHERE C1(a, z) RETURN E5(a, z))
Optimized XQuery Query	E1(FOR a in E2 FOR z in E3 WHERE C1(a, z) RETURN E5(a, z))

〈표 1〉은 LET절이 생략 가능한 경우의 예이다. 표의 최상단 질의는 원본 XQuery 질의이고, 중앙에 기술된 질의는 원본 XQuery 질의의 간략화된 표현이다. 본 논문에서는 앞으로 XQuery 질의를 간략하게 표현하기 위하여 간략화된 XQuery 질의 표현 방법을 사용한다. 그러나 최적화 알고리즘을 설명하기 위하여 반드시 필요한 부분은 원본 XQuery 질의 표현 방법을 사용한다. 간략화된 XQuery 질의 표현 중 E는 XPath 표현(Expression)이고, C는 조건(Condition)을 의미하며, 괄호에 포함된 문자는 E또는 C를 수행하기 위해서 사용해야 할 변수를 의미한다. 〈표 1〉의 원본 질의의 경우, 중간에 선언된 LET절은 그 하위에서 더 이상 사용되지 않으므로 불필요한 LET절의 수행을 "첫 번째 FOR절의 횟수 × 두 번째 선언된 FOR절의 횟수"만큼 반복한다. 그러므로 위와 같은 질의의 경우 LET절을 생략해도 질의의 결과에는 아무런 영향을 미치지 못하므로, 〈표 1〉의 최하단 XQuery 질의와 같이 LET절을 생략하여 최적화를 수행한다.

표 2.잉여의 XPath 표현의 제거
Table 2.Removing Redundant XPath Expressions.

Original XQuery Query	Optimized XQuery Query
FOR a in E1 FOR b in E2 **LET c := E3** WHERE **E3**/text() = "C1" RETURN E4(a, b, **E3**)	FOR a in E1 FOR b in E2 LET c := E3 WHERE **c**/text() = "C1" RETURN E4(a, b, **c**)

〈표 2〉는 원본 XQuery 질의와 중복된 XPath 표현을 제거한 XQuery 질의 이다. 원본질의의 경우, E3의 결과는

LET절에 의해서 이미 변수 c에 할당 되어 있다. 그러나 WHERE절과 RETURN절에서 동일한 XPath 표현을 선언하여 사용하고 있으므로 중복된 XPath 표현을 처리해야만 한다. 그러므로 이를 제거하기 위하여 XPath 표현 E3대신 변수 c를 이용하여 중복된 XPath 표현을 제거한다. 위와 같은 경우 선 처리되는 XPath 표현이 LET절이 아닌 FOR절이라 하더라도 동일한 최적화 방법을 적용할 수 있다.

표 3.의미적으로 중복된 부분을 제거
Table 3. Removing Redundant Conditions.

Original XQuery Query	Optimized XQuery Query
FOR a in E1 WHERE a/@id = "1" and a/@id < "3" RETURN E2(a)	FOR a in E1 WHERE a/@id = "1" RETURN E2(a)

동치 또는 포함관계의 연산 중 의미적으로 중복된 부분을 제거 하는 예는 〈표 3〉과 같다. 원본 질의의 WHERE절에 선언된 조건은 우측 조건과 좌측 조건이 'AND'연산으로 묶여있고 좌측 조건이 우측 조건보다 그 범위가 더 작으므로 〈표 3〉의 우측 XQuery 질의와 같이 최적화할 수 있다. 현재 본 논문에서는 위와 같이 연산의 의미적인 동치 혹은 중복을 제거하기 위하여 XQuery의 연산자들 중 값 비교 연산자(Value Comparisons)과 일반 비교 연산자(General Comparisons)만을 그 대상으로 하고 있다.

2. 연산 순서의 재배치를 이용한 최적화

연산 순서의 재배치를 이용한 최적화는 크게 연산자의 위치를 이동하는 방법과 LET절의 위치를 이동하는 방법으로 구분된다. 연산자의 위치를 이동하는 방법은 XQuery 질의의 WHERE절에 선언된 연산자들 중 상수 값(Constant Value)을 얻을 수 있는 연산자는 상수 값으로 대체하고, 상수 값을 피 연산자로 갖는 연산을 연산자들 중 가장 앞쪽으로 배치한다. 물론 XQuery 처리기가 XQuery 질의를 처리할 때, 연산자들 중 가장 앞쪽에 위치한 연산자를 가장 먼저 처리하지 않을 수도 있다. 그러나 대부분 XQuery 질의 처리기가 XQuery 질의를 처리하는 순서는 XQuery 구문 분석 트리로부터 bottom-up, left-right로 처리하므로 우리는 상수 값을 갖는 연산자의 순서를 앞쪽으로 재배치하여 최적화를 수행한다.

표 4. 연산자의 재배치
Table 4. Replacing Conditions.

Original XQuery Query	Optimized XQuery Query
FOR a in E1 FOR b in E2 FOR c in E3 WHERE a/text()=b/x/text() and b/@id<c/@id and b/@id="1" RETURN E4(b, c)	FOR a in E1 FOR b in E2 FOR c in E3 WHERE b/@id="1" and "1"<c/@id and a/text()=b/x/text() RETURN E4(b, c)

〈표 4〉에서 알 수 있는 것처럼, XPath 표현 'b/@id'의 값은 "1"이다. 그러므로 XQuery 질의 내에 동일한 XPath 표현들은 "1"로 바꾸어 주어도 무방하다. 또한 세 조건들 중 상수 값을 피 연산자로 하는 연산의 위치를 앞으로 재배치함으로 뒤쪽에 오는 연산의 대상이 되는 데이터의 크기를 줄이고자한다. XQuery 질의의 연산 순서는 OR의 위치에 무관하게 AND가 항상 먼저 수행된다. 그러므로 만약 연산자들 사이에 OR 연산이 존재한다면 OR 연산을 기준으로 좌.우측의 연산자들 사이의 재배치만이 가능하다.

연산 순서를 재배치하기 위한 다른 한 가지 방법은 LET절의 위치를 이동하는 것이다. 〈표 5〉는 LET절의 위치를 이동하여 최적화를 수행하는 예제 질의이다. 원본 XQuery 질의에 선언된 LET절을 수행하기 위해서는 반드시 변수 a의 값을 얻어야만 하고 변수 a의 값은 가장 첫 번째 선언된 FOR절을 수행하여 얻을 수 있다. 즉, 원본 질의의 두 번째 선언된 FOR절은 LET절을 수행하기 위해서 어떤 역할도 하지 못한다. 그러므로 LET절의 위치를 LET절을 수행하기 위해서 필요한 정보를 가장 먼저 얻을 수 있는 위치로 이동 시킨다. 〈표 5〉의 원본 질의의 LET절은 첫 번째 FOR절과 두 번째 FOR절을 곱한 수만큼 수행하지만, 최적화된 질의에서 LET절은 첫 번째 FOR절만큼의 수행으로 동일한 결과를 얻을 수 있다.

표 5. LET절의 재배치
Table 5. Replacing LET Clauses.

Original XQuery Query	Optimized XQuery Query
FOR a in E1 FOR b in E2 LET c := E3(a) WHERE b/@id=c/@id and c/text()="Bali" RETURN E4(a, b)	FOR a in E1 LET c := E3(a) FOR b in E2 WHERE b/@id=c/@id and c/text()="Bali" RETURN E4(a, b)

2. FOR절 기반 질의 재작성을 이용한 최적화

For절 기반의 질의 재작성을 이용한 XQuery 질의 최적화의 목적은 질의 재작성에 의해서 불필요한 연산의 수행을 줄이는 것이다. 많은 경우 XQuery 질의의 반복은 FOR절로부

터 야기되고 이러한 반복은 불필요한 중복 연산을 발생시킨다. 그러므로 본 논문에서는 어떤 형태의 FOR절이 불필요한 연산을 발생하는지 정의하고, 이를 줄이기 위한 질의 재작성 방법을 제안한다. FOR절 기반의 질의 재작성을 위해서 본 논문에서는 먼저 XQuery 질의에서 사용되는 FOR절을 위치에 따라 다음과 같이 크게 네 가지로 구분한다.

1. FOR절의 변수가 WHERE절과 RETURN절에 모두 사용되는 FOR절.
2. FOR절의 변수가 WHERE절에는 사용되나 RETURN절에는 사용되지 않는 FOR절.
3. FOR절의 변수가 WHERE절에는 사용되지 않으나 RETURN절에는 사용되는 FOR절.
4. FOR절의 변수가 WHERE절과 RETURN절에 모두 사용되지 않는 FOR절.

위 네 가지 경우는 XQuery 질의가 내포 질의를 포함하는지 여부를 고려하지 않은 구분이다. 그러나 내포 질의를 포함한다 하더라도 내포 질의 역시 위의 4가지로 구분이 가능하다. FOR절의 네 가지 분류 중 1번과 2번의 두 가지 경우에는 WHERE절의 조건을 만족하는 FOR절이 RETURN절에 의해서 생성되는 결과에 영향을 미치는 경우이다. 즉, FOR절의 변수가 WHERE절에 다시 사용되므로 RETURN절은 WHERE절의 조건을 수행한 후의 결과의 횟수만큼 반복된다. 나머지 두 가지(3,4)의 경우의 FOR절은 FOR절의 변수가 WHERE절에 사용되지 않으므로 단순히 RETURN절의 반복의 의미를 갖는다. 즉, 이때의 FOR절은 XQuery 질의의 WHERE절에 대해서 조건들을 반복 수행시킬 뿐 어떠한 의미도 부여하지 못한다. 그러나 반면 XQuery 질의의 RETURN절에 대해서는 동일한 결과를 반복하여 생성하도록 한다.

FOR절의 변수의 사용에 따라 위와 같이 네 가지로 구분된다 할지라도, 해당 FOR절은 XQuery 질의 안에 선언된 다른 FOR절들 사이의 위치에 따라 다음과 같이 다시 세 가지로 구분된다.

A. 해당 FOR절이 XQuery 질의 내의 다른 FOR절들 보다 가장 상위에 선언된 경우
B. 해당 FOR절이 XQuery 질의 내의 다른 FOR절들 사이에 선언된 경우
C. 해당 FOR절이 XQuery 질의 내의 다른 FOR절들 보다 가장 하위에 선언된 경우

그러므로 XQuery 질의에 선언된 FOR절은 총 열두 가지의 종류로 구분되며, 〈표 6〉은 XQuery 질의 내에서 FOR절이 사용되는 열두 가지 경우의 XQuery 예제이다. 〈표 6〉의 각 예제 질의에 이탤릭체로 쓰인 질의 번호는 위에서 설명한 FOR절의 열두 경우를 표현한다. 예를 들어, query 2-B는 FOR절의 변수가 WHERE절에는 사용되나 RETURN절에는 사용되지 않으며, FOR절이 다른 FOR절들 사이에 선언된 경우의 질의이다. 또한 굵게 기술된 FOR절은 앞서 언급한 12가지의 FOR절의 조건을 만족하는 해당 FOR절이다. FOR절을 기반으로 불필요한 연산의 수행을 줄이기 위한 기본 아이디어는 컴파일러의 Loop Invariant를 이용하여 FOR절에 의해서 발생되는 반복 횟수를 줄이는 것이다. 이를 위하여 본 논문에서는 아래와 같은 세 가지 질의의 재작성 방법을 이용한 XQuery 질의의 최적화 방법을 제안한다.

표 6. XQuery 질의에서 사용되는 FOR절의 12가지 분류를 위한 예제 질의들
Table 6. 12 cases of FOR clause in XQuery queries.

▪ *query1-A* ***For x in E1*** For a in E2 Where C1(***x***) and C2(a) Return ***x***	▪ *query 1-B* For a in E1 ***For x in E2*** For b in E3(a) Where C1(***x***) and C2(b, ***x***) Return a, ***x***	▪ *query 1-C* For a in E1 ***For x in E2*** Where C1(a) and C2(***x***) Return ***x***
▪ *query 2-A* ***For x in E1*** For a in E2 Where C1(a) and C2(***x***) Return a	▪ *query 2-B* For a in E1 ***For x in E2*** For b in E3(a) Where C1(***x***) and C2(b) Return a	▪ *query 2-C* For a in E1 ***For x in E2*** Where C1(a) and C2(***x***) Return a
▪ *query 3-A* ***For x in E1*** For a in E2 Where C1(a) Return a, ***x***	▪ *query 3-B* For a in E1 ***For x in E2*** For b in E3(a) Where C1(b) Return a, ***x***	▪ *query 3-C* For a in E1 ***For x in E2*** Where C1(a) Return a, ***x***
▪ *query 4-A* ***For x in E1*** For a in E2 Where C1(a) Return a	▪ *query 4-B* For a in E1 ***For x in E2*** For b in E3(a) Where C1(b) Return a	▪ *query 4-C* For a in E1 ***For x in E2*** Where C1(a) Return a

■ XQuery 질의의 Subset을 FOR절의 내포 질의로 재작성하는 방법.
■ XQuery 질의의 Subset을 LET절의 내포 질의로 재작성하는 방법.
■ XQuery 질의의 Subset을 RETURN절의 내포 질의로 재작성하는 방법.

표 7. FOR절의 내포 질의로 질의를 재작성하여 최적화한 예제 XQuery 질의들
Table 7. Rewriting a part in original XQuery query into an inner query inside a FOR clause.

▪optimized query 1-A ***For x' in*** ***For x in E1*** ***Where C1(x)*** ***Return x*** For a in E2 Where C2(a) Return ***x'***	▪optimized query 1-B For a in E1 ***For x' in*** ***For x in E2*** ***Where C1(x)*** ***Return x*** For b in E3(a) Where C2(b, ***x'***) Return a, ***x'***	▪optimized query 1-C For a in E1 ***For x' in*** ***For x in E2*** ***Where C2(x)*** ***Return x*** Where C1(a) Return ***x'***
▪optimized query2-A ***For x' in*** ***For x in E1*** ***Where C2(x)*** ***Return x*** For a in E2 Where C1(a) Return a	▪optimized query 2-B For a in E1 ***For x' in*** ***For x in E2*** ***Where C1(x)*** ***Return x*** For b in E3(a) Where C2(b) Return a	▪optimized query 2-C For a in E1 ***For x' in*** ***For x in E2*** ***Where C2(x)*** ***Return x*** Where C1(a) Return a

첫 번째 방법인 FOR절 안에 내포 질의를 생성하여 최적화하는 방법은 앞서 언급된 FOR절의 구분 중 1번과 2번 질의를 최적화하기 위해서 사용된다. 즉, 〈표 6〉에 기술된 1-A, 1-B, 1-C, 2-A, 2-B그리고 2-C의 질의들이 이에 속하며 〈표 7〉과 같이 최적화 된다. 〈표 7〉에서 볼 수 있는 것처럼, 조건에 포함된 변수를 선언한 FOR절은 조건과 함께 내포 질의로 재구성된다. 이렇게 재작성된 질의는 원본 질의에 비해서 적은 연산의 수행을 필요로 한다. 예를 들어 〈표 6〉에 기술된 원본 질의 1-A, 1-C, 2-A 그리고 2-C에서, 질의의 첫 번째 FOR절에 의해 반복되는 x가 100개이고, 두 번째 FOR절에 의해 반복되는 a가 100개라면, WHERE절에 선언된 조건 C1과 C2는 각각 10,000번씩 수행된다. 그러나 최적화된 질의들의 경우, 각각 조건 C1의 수행 횟수는 100번, 조건 C2의 수행횟수는 "C1에 의해서 선 처리된 x의 개수 × 100번" 이다.

LET절 안에 내포 질의를 생성하여 XQuery 질의를 최적화하는 방법은 〈표 6〉에 기술된 질의들 중 3-A와 4-A와 같은 경우의 FOR절이 존재할 경우 사용이 가능하다. 〈표 8〉은 〈표 6〉에 기술된 질의들 중 3-A와 4-A를 최적화한 예제 질의이다.

표 8. LET절의 내포 질의로 질의를 재작성하여 최적화한 예제 XQuery 질의들
Table 8. Rewriting a part in original XQuery query into an inner query inside a LET clause.

▪ optimized query 3-A ***Let a' :=*** ***For a in E2*** ***Where C1(a)*** ***Return a*** For x in E1 ***For a" in a'*** Return ***a"***, x	▪ optimized query 4-A ***Let a' :=*** ***For a in E2*** ***Where C1(a)*** ***Return a*** For x in E1 Return ***a'***

질의의 일부를 LET절의 내포 질의로 재작성하여 최적화하는 경우, 선택된 FOR절에 의해 포함된 하위 질의는 LET절의 내포 질의로 재작성된다. 〈표 6〉의 질의 3-A의 첫 번째 FOR절은 RETURN절에만 사용되는 FOR절이므로, 해당 FOR절의 하위에 선언된 질의는 LET절의 내포 질의로 재작성된다. 〈표 6〉의 4-A의 경우, 첫 번째와 두 번째 선언된 FOR절이 각각 100번씩 수행한다면, 조건 C1(a)은 10,000번 수행해야 하지만, 〈표 8〉의 최적화된 질의 4-A의 경우, C1(a)는 100번만 수행하면 된다. 또한 두 번째 FOR절의 경우 첫 번째 FOR절의 영향을 받지 않고 100번만 수행하므로 FOR절의 불필요한 중복 수행을 제거할 수 있다.

RETURN절 안에 내포 질의를 생성하여 최적화 하는 방법은 〈표 6〉에 기술된 질의들 중 3-C와 4-C의 경우에 적용된다. 〈표 9〉는 〈표 6〉에 기술된 질의들 가운데 질의 3-C와 4-C를 최적화한 예이다. 위 예제질의의 경우, 재작성된 FOR절들은 모두 RETURN절에만 영향을 미치는 경우이다. 즉, 선택된 FOR절에 포함된 WHERE절은 반복 연산만 수행할 뿐 FOR절은 WHERE절의 결과에 아무런 영향도 미치지 못한다. 그러므로 FOR절이 RETURN절에 만 영향을 미치도록 하기 위하여 원본질의의 RETURN절에 내포 질의를 생성한다. 이러한 최적화 방법은 FOR절에 의해 반복되는 조건들의 불필요한 수행을 제거할 수 있다.

표 9. RETURN절의 내포 질의로 질의를 재작성하여 최적화한 예제 XQuery 질의들
Table 9. Rewriting a part in original XQuery query into an inner query inside a RETURN clause.

- optimized query 3-C For a in E1 Where C1(a) Return ***For x in E2*** Return a, x	- optimized query 4-C For a in E1 Where C1(a) Return ***For x in E2*** Return a

<표 6>의 3-B와 4-B의 경우, 질의 재작성을 이용한 최적화는 매우 어렵다. 질의 3-B과 4-B에 굵게 기술된 FOR절은 원본질의에서 결과의 순서에 영향을 미치기 때문에 재작성될 수 없다. 다시 말하면, 재작성된 질의와 원본 질의가 비록 동일한 조건을 수행하고 동일한 결과의 집합을 얻는다 할지라도, FOR절에 의해서 두 질의의 결과를 구성하는 순서가 달라지기 때문에 이들을 재작성하는 것은 어려운 일이다.

IV. 결 론

<그림 1>은 XQuery 질의 최적화 시스템의 구성을 보인다.

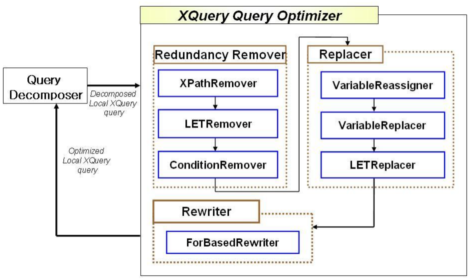

그림 1. XQuery 질의 최적화기의 구조
Fig. 1. An Architecture of XQuery query Optimizer

XQuery 질의 최적기는 RedundnatRemover, Replacer, 그리고 Rewriter의 세 가지 모듈로 구성되어 있다. 각 모듈들의 물리적인 입력 형태는 모두 XQuery 구문 분석 트리이고, 출력 형태 역시 최적화된 XQuery 구문 분석 트리이다. 또한 모듈들은 독립적으로 구성되어 있으므로 응용의 필요에 따라서 부분적 수행이 가능할 뿐만 아니라 향후 최적화를 위한 과정이 추가된다면 손쉽게 모듈을 추가할 수 있다. <표 10>은 Rewriter가 사용하는 알고리즘이며 <표 11>은 사용자가 입력한 원본 XQuery 질의와, 질의 재작성 알고리즘을 통해 얻어진 최적화된 XQuery 질의를 보인다. 최적화된 질의는 앞서 언급한 세 가지 최적화를 위한 재작성 방법을 모두 적용한 예제이다. 원본질의의 첫 번째 선언된 FOR절(원본 질의의 line 2)은 하위의 어느 곳에서도 사용되지 않으므로 그 하위에 포함된 질의는 LET절의 내포 질의로 재작성 된다(최적화된 질의의 Line 2~17). LET절의 내포 질의는 다시 원본 질의의 5 번째에 선언된 FOR절(원본 질의의 Line 6)에 선언된 변수가 WHERE절과 RETURN절에 사용되었으므로 WHERE절의 조건과 함께 새로운 FOR절의 내포 질의로 재작성된다(최적화된 질의의 Line 6~9). 또한 가장 마지막에 선언된 FOR절(원본 질의의 Line 9)은 그 변수가 RETURN절에만 사용되므로 RETURN절의 내포 질의로 재작성될 수 있다(최적화된 질의의 Line 15~17). 마지막으로 Line 9에 선언된 FOR절이 RETURN절의 내포 질의로 생성되면서 원본질의의 Line 8에 선언된 FOR절이 가장 하위에 선언된 FOR절이 되고, 또한 그 변수가 WHERE절의 조건에 사용되었으므로 이들을 내포 질의로 하는 새로운 FOR절의 내포 질의로 재작성된다(최적화된 질의의 Line 11~14).

표 10. 질의 재작성 알고리즘
Table 10.An Algorithm for XQuery query Rewriting.

```
- FORclause[] : 현재 질의에 포함된 모든 FOR절;
- WHERECondition[] : WHERE절에 기술된 조건들;
- VariableForRETURN : 새로 생성된 내포 질의의 반환을 위한 변수;

Rewrite(XQuery Query){
 For(i=0 to The number of FOR clauses){
  v = the variable of current FOR clause;
  FORExpression = The Expression of current FOR clause;
  /** 질의 구분을 위한 부분 **/
  if(FORExpression = = inner query) Rewrite(inner query);
  for(j=0 to The number of conditions in WHERE clauses) {
   if(v is used in WHERECondition[j]){
    NewCondition[] add WHERECondition[j];
    if(v is used in RETURN clause){
     VariableForRETURN= v; CurrentCase= CASE1;}
    else {CurrentCase = CASE2;}
   else if(j == The number of conditions in WHERE clauses){
    if(v is used in RETURN clause) {
     VariableForRETURN = v; CurrentCase= CASE3;}
    else {CurrentCase = CASE4;} }
  if(I = =0) {CurrentCase = CurrentCase + "A";}
  else if(I = = The number of FOR clauses){
   CurrentCase = CurrentCase + "C";}
  else CurrentCase = CurrentCase + "B";

  /** 질의 재작성을 위한 부분 **/
  if(CurrentCase is CASE1-A, CASE1-B, CASE1-C, CASE2-A,
CASE2-B or CASE2-C){
   create new FORclause including FORExpression,
    NewCondition[] and VariableForRETURN;
   replace FORclause[i] with created new FORclause;}
  else if(CurrentCase is CASE3-A or CASE4-A){
   create new LETclause including all clauses in
    current query except FORclause[i]; }
   append FORclause[i];
   if(CurrentCase is CASE3-A) {
    create new FORclause which returns the variable
     of new
    created LETclause;
    create new RETURNclause which return all
     variables of FORclauses;
 else if(CurrentCase is CASE3-C or CASE4-C){
  create new RETURNclause consist of FORclause[i]
   and RETURNclause in current query
  replace RETURNclause with created new
   RETURNclause; } } }
```

표 11. FOR절 기반의 질의 재작성을 이용한 XQuery 질의 최적화
Table 11. Optimized XQuery query by query Rewriting.

Original XQuery query

```
1. 〈Results〉{
2.  for $Person in doc("people.xml")/people/person
3.  for $Open_Auction in doc("open_auctions.xml")/
      open_auctions/open_auction
4.  for $ItemAsia in
      doc("ItemsOfAsia.xml")/asia/item
5.  for $Categories in doc("catgraph.xml")/catgraph
6.  for $Open_Auction_Reserve in
      $Open_Auction/reserve
7.  for $Open_Auction_Itemref in
      $Open_Auction/itemref/@item
8.  for $ItemAsiaID in $ItemAsia/@id
9.  for $Cat_edge in $Categories/edge
10.  where $Open_Auction_Reserve 〈 100 and
          $Open_Auction_Itemref = $ItemAsiaID
11.  return 〈Result〉{ $Open_Auction_Itemref,
          $Open_Auction_Reserve, $Cat_edge }
〈/Result〉
12. }〈/Results〉
```

Optimized XQuery query

```
1. 〈Results〉{
2.  let $MakedLet0 :=
3.   for $Open_Auction in oc("open_auctions.xml")
       /open_auctions/open_auction
4.   for $ItemAsia in
       doc("ItemsOfAsia.xml")/asia/item
5.   for $Categories in
       doc("catgraph.xml")/catgraph
6.   for $MakedFor0 in
7.       for $Open_Auction_Reserve in
           $Open_Auction/reserve
8.       where $Open_Auction_Reserve 〈 100
9.       return $Open_Auction_Reserve
10.  for $Open_Auction_Itemref in
       $Open_Auction/itemref/@item
11.  for $MakedFor1 in
12.      for $ItemAsiaID in $ItemAsia/@id
13.      where $Open_Auction_Itemref =
            $ItemAsiaID
14.      return $ItemAsiaID
15.  return
16.      for $Cat_edge in $Categories/edge
17.      return 〈Result〉{ $Open_Auction_Itemref,
           $MakedFor0, $Cat_edge }〈/Result〉
18. for $Person in doc("people.xml")/people/person
19. return $MakedLet0
12.}〈/Results〉
```

V. 성능 평가

본 논문에서 실험 환경으로 운영체제는 Windows Server 2003를 사용하였으며, 실험에 사용된 시스템 사양은 Intel(R) Pentium(R) 4 CPU 3.0GHz와 2.0GB RAM을 사용하였다.

표 12. 불필요한 표현 제거를 통한 최적화의 성능평가를 위한 sample 질의들
Table 12.Sample XQuery queries for evaluating optimization methods by removing redundant expressions.

XQ1	〈results〉{ for $closedAuction in doc("closed_auctions.xml")/closed_auctions/closed_auction for $itemAsia in doc("ItemsOfAsia.xml")/asia/item for $itemEurope in doc("ItemsOfEurope.xml")/europe/item let $itemClosed := $closedAuction/itemref let $mail := $itemAsia/mailbox/mail where $itemAsia/quantity 〈 3 and $itemAsia/quantity = 1 and $closedAuction/itemref/@item = $itemAsia/@id and $itemAsia/quantity = $itemEurope/quantity eturn〈result〉〈asia〉{$itemAsia/name} 〈/asia〉〈/result〉 }〈/results〉
XQ2	where $itemAsia/quantity 〈 3 and $itemAsia/quantity = 1 and $closedAuction/annotation/happiness 〉 5 and count($itemAsia/incategory) 〉 3 and $closedAuction/price 〈 100 and $closedAuction/itemref/@item = $itemAsia/@id and $itemAsia/quantity = $itemEurope/quantity

원본질의와 최적화된 질의 사이의 성능 평가를 위한 XQuery 엔진은 Oracle XQuery 엔진을 사용한다. 물론 Oracle XQuery 엔진 이외에 성능 평가를 위한 XQuery 처리기는 다수 존재한다. 그러나 Oracle XQuery 엔진을 선택한 이유는 Oracle 엔진이 현재 XQuery Version을 대부분 지원하고 있으며, 자유롭게 다운로드가 가능하기 때문이다. 또한 성능 평가를 위해서 찾아본 몇몇의 XQuery 엔진(SAXON, XMLSpay, BIZQuery)들은 대부분 자체적으로 최적화를 수행한다. 엔진의 자체적인 최적화는 성능 평가에 영향을 미칠 수 있으므로 우리는 Oracle XQuery 엔진을 선택하였다. 성능평가를 위한 견본 XML 데이터는 다수의 문서를 생성하기 위하여 XMark의 Auction 데이터 중 차 상위 노드인 People, open_auctions, asia, and catgraph element 노드를 최상위 노드로 하는 4개의 문서를 생성하여 사용했으며, 원본 질의는 〈표 12〉와 같다. 〈표 12〉의 XQ2는 XQ1과 FOR와 RETURN절의 구조는 같지만 WHERE절에 보다 많은 조건을 포함하는 질의이다.

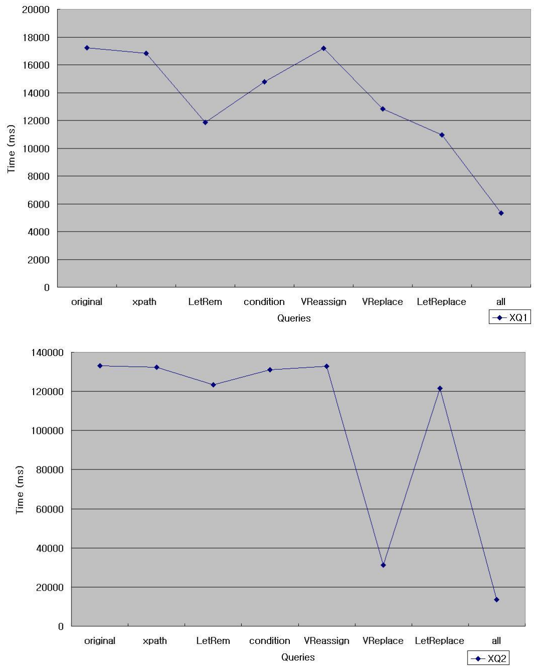

그림2.〈표 12〉의 XQ1과 XQ2의 수행 시간과 각 최적화 방법에 의해 최적화된 질의들의 수행시간
Fig.2.Execution time of XQ1 and XQ2 in Table 12 and optimized queries

〈그림 2〉는 약 1MB의 견본 XML 데이터를 대상으로 〈표 12〉에 기술된 질의들과 이들을 최적화한 질의들의 성능을 측정한 결과이다. 그래프의 X축에 항목들은 각각 "원본질의", "불필요한 XPath 표현을 제거한 질의", "불필요한 LET절을 제거한 질의", "의미적으로 중복된 조건을 제거한 질의', "조건의 피 연산자를 상수 값으로 대체한 질의", "WHERE절의 조건들 중 상수 값을 갖는 조건을 앞쪽으로 재배치시킨 질의", "LET절의 위치를 최소 정보를 얻을 수 있는 곳으로 재배치시킨 질의", 그리고 "앞선 모든 최적화 방법을 적용한 질의"이다. 성능 평가의 결과, XQ1의 경우 "불필요한 LET절을 생략한 질의"와 "LET절의 위치를 이동한 질의"가 가장 높은 성능의 개선을 보였다. 이는 불필요한 LET절을 생략 하거나 원본 질의에서 보다 미리 수행하므로 앞서 선언된 FOR절들에 의해서 LET절을 반복하여 수행하는 횟수가 그만큼 감소하였기 때문이다. XQ2의 경우도 XQ1과 마찬가지로 앞선 두 가지 방법에 의해서 최적화된 질의가 좋은 성능 개선을 보였으나, 가장 높은 성능의 개선을 보인 최적화 방법은 상수 값을 갖는 질의를 다른 질의 보다 먼저 수행할 수 있도록 위치를 옮기는 방법이다. 이는 XQ2의 경우 WHERE절에 조건의 수가 많아 이를 처리하기 위한 시간이 많이 필요하므로 조건을 처리하기 위한 시간을 줄일 때 많은 성능의 개선을 가져왔다고 보인다. 그러나 "불필요한 XPath 표현을 제거한 질의"와 "WHERE 조건에 피 연산자를 상수 값으로 대체한 경우의 질의"는 원본 질의를 처리하기 위한 시간과 유사한 수행 시간을 보인다. 이러한 이유는 Oracle XQuery 엔진 자체에서 이와 관련된 최적화를 이미 수행하고 있기 때문이다.

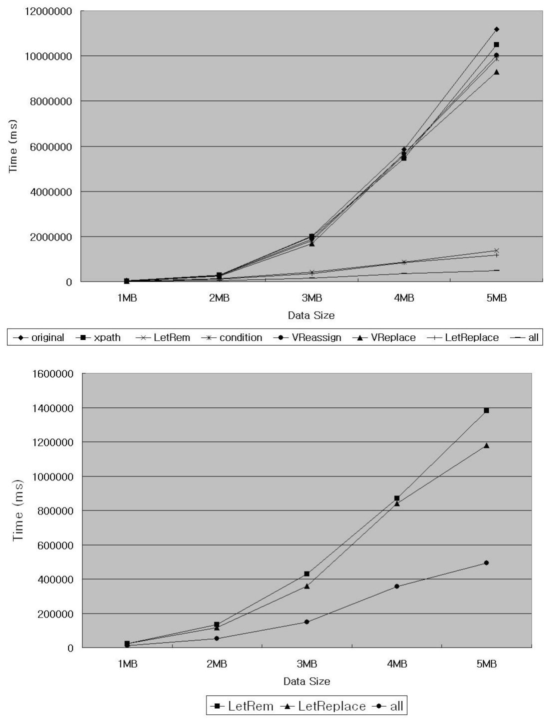

그림3. 데이터 크기에 따른 〈표 12〉의 XQ1과 최적화 질의들의 수행시간
Fig. 3. Execution time of XQ1 in Table 12 and optimized queries according to Data size.

〈그림 3〉은 〈표 12〉의 XQ1 질의와 이를 최적화한 질의를 데이터 크기에 따라 성능을 측정한 결과이다. 데이터 크기에 따른 성능 측정 방법은 데이터의 크기를 1MB에서 5MB까지 1MB씩 증가시키면서 질의들의 수행 시간을 측정하였다. 〈그림 3〉의 그래프 중 위쪽 그래프는 여덟 가지 종류의 질의를 수행한 결과를 나타내고 있으며, 아래쪽 그래프는 "불필요한 LET절을 제거한 질의", "LET절의 위치를 이동한 질의", "모든 최적화 방법을 적용한 질의"의 세 가지 경우만을 보다 자세히 보인다. 위 그림에서 알 수 있는 것처럼, 최적화를 위한 모든 방법은 데이터 크기와 비례하여 증가하며, LET절과 연관

된 최적화 방법들은 데이터 크기가 증가할수록 원본 질의를 처리하기 위한 수행시간에 비해 월등한 성능 개선을 보인다.

표 13. 질의 재작성 기반의 최적화 방법의 성능평가를 위한 견본 질의들
Table 13. Sample XQuery queries for evaluating optimization methods by query rewriting.

XQ1	Origianl Xquery query in Table 11
XQ2	where $Open_Auction_Reserve < 100 and $Open_Auction_Itemref = $ItemAsiaID and $ItemAsia/quantity = Open_Auction/quantity and $Open_Auction/annotation/happiness >5
XQ3	return <Result>{<item id="$ItemAsiaID"> {$ItemAsia/name, ItemAsia/location} </item>, $Open_Auction, <category>{$Cat_edge}</category> }</Result>

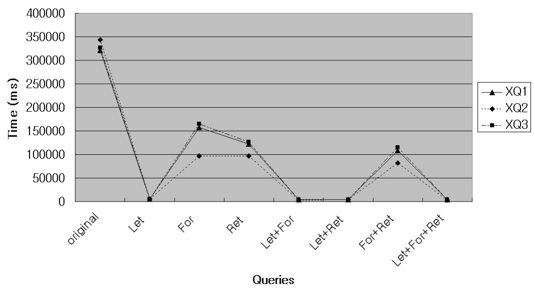

그림 4. 〈표 13〉의 XQ1, XQ2, XQ3과 질의 재작성에의해 최적화된 질의들의 수행 시간
Fig. 4. Execution time of XQ1, XQ2, and XQ3 in Table 13 and optimized queries by query rewriting.

FOR절 기반의 질의 재작성을 위한 성능 평가를 위하여, 논문에서는 앞서 기술한 〈표 11〉의 XQ1과 함께 2가지 형태의 질의를 더 추가하여 견본XQuery 질의로 사용한다. 〈표 13〉의 XQ2는 XQ1과 동일한 FOR와 RETURN절의 구조를 갖지만 WHERE절에 보다 많은 조건을 포함하는 질의이고, XQ3는 RETURN절에 보다 많은 반환 결과를 요구하는 질의이다. 〈그림 4〉는 〈표 13〉의 원본 XQuery 질의들과 이를 FOR절 기반으로 재작성한 XQuery 질의들을 약 1MB의 견본 XML 데이터를 기반으로 수행 시간을 측정한 그래프이다. 그래프의 X축의 항목들 중 LET, FOR, 그리고 RET는 각각 "LET절 안에 내포 질의를 생성하여 재작성된 질의", "FOR절 안에 내포 질의를 생성하여 재작성된 질의", 그리고 "RETURN절 안에 내포 질의를 생성하여 재작성된 질의"를 나타낸다. 또한 LET+FOR+RET는 앞선 세 가지 최적화 방법을 모두 적용한 질의로 결국 〈표 11〉의 최적화된 XQuery 질의이다. 성능 평가의 결과 LET절의 내포 질의로 최적화된 질의들은 많은 성능의 개선을 보인다. 그 이유는 원본 XQuery 질의의 가장 바깥쪽에 선언되어있던 FOR절이 최적화에 의해서 생성된 LET절을 수행한 후 수행되므로, LET절에 포함된 내포 질의의 수행 횟수를 FOR절에 의해서 발생되는 횟수만큼 줄일 수 있기 때문이다. 또한 XQ2가 다른 질의들 보다 최적화된 질의의 수행 시간이 대체적으로 적은 이유는 WHERE 절에 조건이 많기 때문에 더 많은 최적화를 수행할 수 있기 때문이다. XQ3의 경우를 통해 XQuery 질의에 의해 출력되는 결과의 크기는 질의의 전체적인 성능에는 영향을 미치지만 최적화와는 무관한 것을 알 수 있다. 〈그림 4〉의 성능 평가 역시 데이터 크기를 점진적으로 증가하여 평가했을 때, 그 수행 속도가 문서 안의 노드의 수의 증가와 비례하는 것을 볼 수 있었으며. 데이터의 크기가 커지면 최적화 방법을 적용한 XQuery 질의의 수행 시간이 좋아지는 것을 알 수 있다. 특별히 LET절의 내포 질의로 재작성한 질의의 경우 월등한 성능의 개선을 보였다.

Ⅵ. 결론 및 향후 연구

본 논문이 XQuery 질의 최적화를 위한 기여는 다음과 같다. 첫째, XQuery 질의를 최적화하기 위한 방법을 구체적으로 어떤 형태의 질의에 어떤 방법을 적용할 수 있는지 제안하였다. 둘째, XQuery 질의에서 FOR절이 어떻게 사용되는지 구분하고, 각 경우에 적절한 최적화 방법을 제안하였다. 셋째, XQuery 질의 자체만을 이용한 최적화 방법을 제안함으로써 기존에 제안된 다른 XQuery 질의 최적화 방법들과 병행하여 사용이 가능하다. 넷째, 제안한 방법들을 기반으로 프로토타입을 설계하고 구현한 후, XQuery 질의 처리기와 연동하여 성능을 평가하였다.

참고문헌

[1] 박종현, 강지훈, "디지털 방송을 위한 Set-Top Boxr1반 TV-Anytime 메타데이터 관리 시스템," 한국컴퓨터정보학회논문지, 제 13권, 제 4호, 71-78쪽, 2008년 7월.

[2] F. Frasincar, G. J. Houben & C. Pau "XAL: An Algebra For XML Query Optimization Proc. Australasian Database Conference 2002, Melbourne, Australia, Feb. 2002.

[3] X. Zhang, B. Pielech & E. A. Rundensteiner "*XML Algebra Optimization*" Technical Report, WPI-CS-TR-02-25, Worcester Polytechnic

Institute, Oct. 2002.
〔4〕 X. Zhang, B. Pielech & E. A. Rundensteiner, "Honey, I shrunk the XQuery!: an XML algebra optimization approach," Proc. WIDM 2002, McLean, Virginia, USA, Nov. 2002.
〔5〕 Alfred V. Aho, Monica S. Lam, Ravi Sethi & Jeffrey D. Ullman, *"Compilers: Principles, Techniques, and Tools (2nd Edition)"* Addison Wesley, 2006.
〔6〕 X. Zhang & E. A. Rundensteiner *"XAT: XML Algebra for the Rainbow System"* Technical Report WPI-CS-TR-02-24, Worcester Polytechnic Institute, July 2002.
〔7〕 M. Grinev & D. Lizorkin "XQuery Function Inlining for Optimizing XQuery Queries," Proc. ADBIS 2004, Budapest, Hungary, Sept., 2004.
〔8〕 M. Grinev, & P. Pleshachkov, "Rewriting-Based Optimization for XQuery Transformational Queries," IDEAS 2005, Montreal, Canada, July 2005.
〔9〕 H. Su, E. A. Rundensteiner & Murali "Semantic Query Optimization in an Automata Algebra Combined XQuery Engine over XML Streams," Proc. VLDB 2004, Toronto, Canada, Sept. 2004.
〔10〕 G. Wang, M. Liu, J. X. Yu, B. Sun, G. Yu, J. Lv & H. Lu "Effective Schema-Based XML Query Optimization Techniques," Proc. IDEAS 2003, Hong Kong, China, July 2003.
〔11〕 S. Groppe, & S. Bottcher, "Schema-based Query Optimization for XQuery Queries", Proc. ADBIS 2005, Tallinn, Estonia, Sept. 2005.
〔12〕 C. Koch, S. Scherzinger, N. Schweikardt & B. Stegmaier "FluXQuery: An Optimizing XQuery Processor for Streaming XML Data," Proc. VLDB 2004, Toronto, Canada, Sept. 2004.

저 자 소 개

박 종 현

2002년 충남대학교 컴퓨터과학과 석사 졸업.
2007년 충남대학교 컴퓨터과학과 박사 졸업.
2007년~2008년 충남대학교 소프트웨어 연구소 전임연구원.
2009~현재 거제대학 조선정보계열 초빙교수
주관심분야: XML, XQuery, Ontology, 분산 데이터베이스, 유비쿼터스 컴퓨팅, 웹정보시스템, 추론

강 지 훈

1981년 한국과학기술원 전산학과 석사졸업.
1996년 한국과학기술원 전산학과 박사졸업.
2000년~2002년 충남대학교 정보통신원장.
1985년~현재 충남대학교 전기정보통신공학부 교수.
주관심분야: 시멘틱웹, 추론, XML, XQuery, 데이터베이스 시스템, 웹정보시스템.