구조적 중복을 사용한 XML 문서의 릴레이션으로의 분할저장

(Shredding XML Documents into Relations using Structural Redundancy)

김 재 훈 † 박 석 ††

(Jaehoon Kim) (Seog Park)

요 약 본 논문에서는 XML 데이타를 릴레이션으로 분할 저장할 경우, 분할된 XML 데이타로부터 질의 결과 XML 문서를 재구성하는데 소모되는 질의 처리 비용을 줄이기 위한 구조적 중복 방법을 소개한다. 기본 아이디어는 주어진 질의 패턴을 분석하여, 적절한 데이타들을 중복시킴으로서 질의 처리 성능을 향상시키는 것이다. 이러한 구조적 중복 방법으로 실질적으로 유효할 수 있는 ID, VALUE, SUBTREE의 세 가지 유형의 특성을 분석하였다. 본 논문에서는 추가적으로 주어진 XML 데이타와 질의들이 매우 크고 복잡할 경우 최적의 중복 집합을 찾는 것이 매우 어려운 작업이 될 수 있으므로, 이를 위한 경험적 탐색 방법을 소개한다. 마지막으로 몇 가지 실험을 통하여, 중복 데이타를 사용함으로 발생하는 XML 질의 처리 비용과 제안된 탐색 방법의 효율성을 분석한다. 중복 데이타를 사용함으로 XML 판독 질의는 빨라지지만, XML 갱신 질의는 중복 데이타의 갱신 일관성 비용 때문에 느려지는 것은 당연하다. 하지만 실험 결과는 매우 과도한 갱신 비용의 경우에도 in-place ID 중복은 효율적이며, 갱신 비용이 매우 과도하지만 않다면 multiple-place SUBTREE 중복은 판독 질의 처리 성능을 크게 향상시킬 수 있음을 보여주었다.

키워드 : XML, 릴레이션, 구조적 중복, 탐색

Abstract In this paper, we introduce a structural redundancy method. It reduces the query processing cost incurred when reconfiguring an XML document from divided XML data in shredding XML documents into relations. The fundamental idea is that query performance can be enhanced by analyzing query patterns and replicating data essential for the query performance. For the practical and effective structural redundancy, we analyzed three types of ID, VALUE, and SUBTREE replication. In addition, if given XML data and queries are very large and complex, it can be very difficult to search optimal redundancy set. Therefore, a heuristic search method is introduced in this paper. Finally, XML query processing cost arising by employing the structural redundancy, and the efficiency of proposed search method are analyzed experimentally. It is manifest that XML read query is performed more quickly but XML update query is performed more slowly due to the additional update consistency cost for replicas. However, experimental results showed that in-place ID replication is useful even in having excessive update cost. It was also observed that multiple-place SUBTREE replication can enhance read query performance remarkably if only update cost is not excessive.

Key words : XML, relation, structural redundancy, search

1. 서 론

최근 대량의 XML 데이타를 처리하려는 노력이 크게 두 가지 방향으로 진행되었다. 하나는 새로운 XML 전용 DBMS를 만드는 것이고, 다른 하나는 기존의 RDBMS나 OODBMS를 활용하는 것이다. 스탠포드 대학의 LORE, Natix[1], ObjectDesign의 eXcelon, Software AG의 Tamino등은 전자에 해당하고, STORED

· 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10395-0) 지원으로 수행되었음

† 비 회 원 : 서강대학교 컴퓨터학과
chris3@dblab.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
spark@dblab.sogang.ac.kr

논문접수 : 2004년 6월 12일
심사완료 : 2004년 12월 13일

[2], 참고문헌 [3], XPERANTO, SilkRoute [4]등은 후자에 해당한다. 하지만 사용자들은 기존의 RDBMS를 손쉽게 활용하고자 하는 경제적인 이유와, 새로운 안정적인 DBMS를 만들기까지는 많은 시간을 요하게 되는 기술적 이유 때문에 당분간 후자에 더 많은 관심을 가질 것이라고 생각된다.

XML 데이타를 RDBMS에 저장하여 활용하는 방법은 질의 결과의 대상이 무엇이냐에 따라 크게 두 가지로 나눌 수 있다. 하나는 질의 대상이 문서 전체인 경우로 XML 문서 자체를 CLOB 컬럼과 같은 곳에 저장하고, 인덱싱 방법을 사용하는 것이다[5]. 다른 하나는 질의 대상이 전체문서가 아닌 문서의 부분들에 관한 셀렉션, 프로젝션, 조인, 갱신을 위한 것으로, XML 데이타를 테이블로 분할 맵핑하는 것이다[2,3,6]. 이러한 분할 저장 방법들은 질의 처리를 위한 대량의 XML 문서들의 DOM 파싱 비용을 줄일 수 있기 때문에 질의 대상이 문서의 부분이라면 더욱 바람직하다. 본 연구의 관심도 후자의 방법에 있다. 하지만 XML을 릴레이션으로 맵핑하는 것의 한 가지 문제점은 주어진 XML 질의를 위해 재 작성된 SQL 문장이 분할된 많은 테이블들로 인해 과도한 조인과 유니온 연산 비용을 초래할 수 있다는 것이다[3,7,8]. 또한 릴레이션 형태의 결과 데이타를 XML 형식으로 변형하기 위한 구조화와 태깅(tagging) 비용이 요구된다.

이러한 기존의 RDBMS를 수정하지 않고, 그 위에 새로운 XML DBMS 상위 층을 만드는 것은, 비록 기존의 RDBMS 응용 프로그램을 제작하는 것과 유사하겠지만, 재미있는 생각을 준다. 그것은 RDBMS를 일종의 물리적 계층으로 간주하여 주어진 XML 논리 스키마에는 영향을 주지 않으면서, 질의 성능을 높이기 위하여 필요시마다 관계형 스키마를 자유롭게 변경하는 것이다. 이것은 XML 데이타가 릴레이션으로 분할 저장될 경우, DTD[9](혹은 XML schema[10])와 관계형 스키마가 어떤 맵핑 정보(예로, IBM DB2의 DAD와 같은)에 의하여 분리될 수 있기 때문이다.

관계형 스키마가 자유롭게 변경될 수 있을 때, 질의 성능을 높이기 위한 실제적인 방법의 하나로 중복 방법이 고려될 수 있다. 즉, 종전의 관계형 스키마 설계시 조인되는 테이블의 수를 줄이기 위하여 조인되는 테이블의 어느 한쪽 필드 값을 다른 테이블로 중복시키는 것과 같은 개념이다. 그러한 가능한 중복은 E/R 모델을 통하여 파악될 수 있다. 이와 비슷하게, XML을 릴레이션으로 맵핑하는 경우에도, XML의 구조적 특성(즉, DTD의 +, *, ?, | 등의 기호와 XML의 트리 구조)에 기인한 몇 가지 중복 방법을 고려할 수 있으며, 본 논문에서 이를 소개하고자 한다.

본 논문은 다음 주요 내용을 고찰한다.:

- XML 데이타의 릴레이션으로의 맵핑시, 구조적 중복 방법의 분류 및 소개
- 그러한 중복 데이타의 갱신 일관성 유지 문제와 그에 따른 전반적 질의 비용의 분석
- 대량의 복잡한 XML 데이타와 질의들이 주어질 경우, 본 논문에서 소개된 중복 방법들의 최적의 설정 해를 얻기 위한 경험적 탐색 방법의 소개

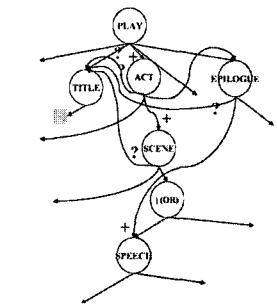
본 논문은 다음과 같이 구성된다. 우선 2장에서는 XML 데이타를 릴레이션으로 맵핑하고 XML 질의어를 SQL로 질의 재 작성 하는 것에 관하여 간단히 소개한다. 3장에서는 본 연구와 관련 연구를 비교하며, 4장에서는 세 가지의 구조적 중복 방법을 소개한다. 5장에서는 효과적 중복 집합을 찾기 위한 경험적 탐색 방법을 제안한다. 6장에서는 중복 방법의 사용에 따른 전반적 질의 비용의 분석과 탐색 알고리즘에 대한 성능 분석을 수행하며, 7장에서는 결론을 맺는다.

2. XML 데이타의 릴레이션으로의 맵핑

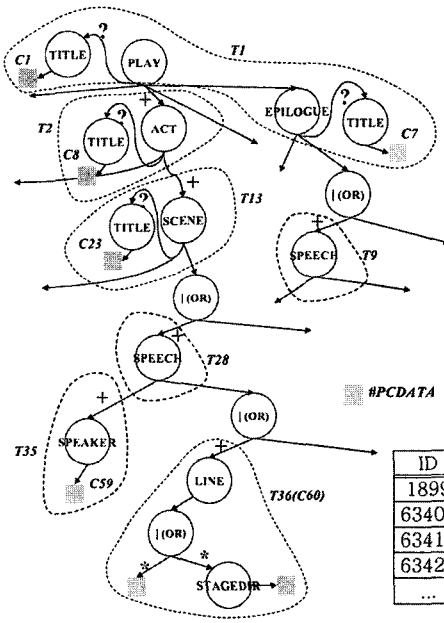
본 장에서는 구조적 중복 방법의 소개에 앞서, 본 연구에서 고려한 XML 데이타의 릴레이션으로의 맵핑 방법 및 XML 질의를 SQL 질의로 재작성하는 방법에 대하여 간단히 설명한다.

먼저 맵핑 방법은 쉐어드 인라이닝 방법[3]과 STORED의 컬렉션 어트리뷰트 처리[2]처럼, DTD에서의 +, * 기호에 의해 일대다의 관계를 갖는 부모 노드와 자식 노드를 각기 다른 테이블로 맵핑하는 방법을 사용하였다. 단, 다른 XML 경로를 갖는 동명의 XML 노드를 동일 테이블로 맵핑시키는 쉐어드 인라이닝 방법과는 달리, 본 연구에서는 다른 테이블로 맵핑시켰다(예로, '/play/act/scene/speech'와 '/play/epilogue/speech'의 다른 XML 경로를 갖는 동명의 SPEECH 노드에 대하여 T28과 T9의 각기 다른 테이블로 맵핑시켰음). 사실 이러한 부분의 선택은 주어진 질의 분석을 통한 비용 기반의 관점[11]에서 결정될 문제라고 본다. 즉, XPath '/play//speech'와 같은 질의의 빈도가 높다면, 동일 테이블로 맵핑하는 것이 이득이며, 반대로 XPath '/play/act/scene/speech'와 '/play/epilogue/speech'의 질의의 빈도가 높다면, 각기 다른 테이블로 맵핑하는 것이 이득일 것이다. 또한 그림 1(b)에서의 LINE 노드처럼 태그와 #PCDATA가 혼합된 경우, 참고문헌 [3]에서처럼 원 XML 형태를 하나의 컬럼에 그대로 저장한다.

XML 데이타의 릴레이션으로의 다른 맵핑 방법으로, Florescu와 Kossmann이 제안한 에지 테이블 방법[6]이 있다. 이 방법의 장점은 스키마가 존재하지 않는 경우도 효율적으로 저장할 수 있다는 것이다. 하지만 본 연구에



(a) DTD 방향성그래프



(b) 확장된 X2RMap 트리

ID	C0 (document_name)	...	C1 (title)
1	a_and_c.xml	...	The Tragedy of Antony and Cleopatra
2	hamlet.xml	...	The Tragedy of Hamlet, Prince of Denmark
...	...	...	...

T1

ID	C8 (title)	...	PID
13	ACT II	...	1
179	ACT II	...	2
...	...	...	...

T2

ID	C23 (title)	PID
58	SCENE I. Messina. POMPEY's house	13
786	SCENE II A room in the castle	179
...	...	...

T13

ID	PID
546	58
14975	786
14979	786
14981	786
...	...

T28

T35

ID	C59 (speaker)	PID
1899	POMPEY	546
63404	ROSENCRANTZ	14975
63416	GUILDENSTERN	14979
63421	HAMLET	14981
...	...	...

T36

ID	C60	PID
1901	...	546
1903	...	546
63406	...	14975
63418	...	14979
63423	...	14981
63425	...	14981
...	...	...

(c) 맵핑 릴레이션

그림 1 셰익스피어 연극 XML 데이타의 릴레이션으로의 저장

선 스키마가 존재하는 대량의 XML 데이타를 가정하며, 이러한 경우 그 커다란 단일 에지 테이블을 여러 번 셀프 조인하는 것은 비용이 매우 크다고 보기 때문에, 본 연구에서는 고려하지 않았다. 하지만 지면 관계상 생략되었지만, 본 연구의 중복 방법들은 그러한 저장 방법에 또한 적용 가능한 개념임을 밝힌다.

그림 1(a)는 John Bosak의 셰익스피어 연극 XML 데이타[12]의 DTD를 방향성 그래프로 표현한 것이다. 다음으로 그림 1(b) 트리는 참고문헌 [3]과 같이 그러한 그래프를 확장하여 얻어진 것이며, 마지막으로 *, + 심볼에 의하여 각 그래프의 노드가 어떤 테이블의 컬럼으로 맵핑된다. 이때 각 노드(즉, XML 엘리먼트 혹은 어트리뷰트)와 테이블의 컬럼 사이의 맵핑 정보가 그림 1(b)의 트리 구조에 정보화되며, 이것을 X2RMap이라고 명명한다. 이러한 맵핑 정보에 따라 그림 1(c)와 같이 XML 데이타는 분할 저장된다.

다음으로, XML 질의가 주어지면, 이의 수행을 위해 XML 질의는 SQL로 재작성되어 수행되어야 한다. 본 연구에서는 이러한 질의 재작성을 위하여, 판독 질의의 경우 참고문헌 [8]에서 제안된 sorted outer union 방법을 이용하며, 갱신 질의의 경우 참고문헌 [7]에서 제안된 trigger-based deletion과 tuple-based insertion을 이용한다.

Sorted outer union 방법의 기본 전략을 간단히 소개

하면 다음과 같다. 즉 테이블의 조인 연산이 한번만 적용되도록 하는 것이다. 또한 주어진 XQuery 질의를 수행하기 위한 SQL 문이 가능한 하나의 문장으로 작성되도록 하는 것이다. 일반적으로 여러 개의 SQL 문장보다는 동등한 결과를 가져올 수 있는 하나의 문장 작성이 바람직하다. 예로, 다음과 같은 XQuery 질의어가 주어지면, 하향식으로 부모 자식노드들 사이에서는 조인(join) 연산을 적용하고, 자매 노드들 사이에서는 유니온(*union*) 연산을 적용한다. 그러므로 각 테이블은 한번의 조인 연산에 적용되며, 조인 연산에 의한 중간 결과는 재사용된다.

```
FOR $S IN document("hamlet.xml")/play/act/scene
WHERE contains($S/title, "castle")
RETURN $S/speech
```

$R1 \rightarrow (\sigma_{c0 = 'hamlet.xml'} T1) \bowtie T2 \bowtie (\sigma_{c23 \text{ like } '\%castle\%'} T13) \bowtie T28$

$R2 \rightarrow R1 \bowtie T35$

$R3 \rightarrow R1 \bowtie T36$

$R4 \rightarrow R2 \cup R3$

...

이러한 sorted outer union의 또 하나의 장점은 기존 RDBMS의 효율적 정열 알고리즘을 통하여 그 실행 결과를 정렬하고, 정렬된 결과 릴레이션을 한번만 스캔함으로서 결과 XML 문서로의 구조화 및 XML 태깅(tagging) 작업을 빠르게 수행하는 것이다.

3. 관련 연구

데이타 분포 및 질의 분석에 근거하여 적절한 중복 데이타를 사용하는 본 연구는 XML을 저장하기 위한 비용기반의 방법이다. 이러한 비용기반의 접근은 STORED[2], LegoDB[11]등에서 찾아 볼 수 있다. STORED는 릴레이션으로 맵핑될 빈도수가 높은 트리 패턴을 찾기 위하여 데이타 마이닝 기법인 apriori 알고리즘을 사용한다. LegoDB 시스템은 비정규화, 수직 수평 분할 등의 스키마 변형에 대해 욕심쟁이 알고리즘을 이용하여 최적의 맵핑 스키마를 찾아낸다. 하지만 두 시스템은 중복을 사용하는 본 연구와 구별된다. 본 연구는 XML 판독 및 갱신 질의를 함께 고려하며, 갱신 질의시의 중복 데이타의 갱신 일관성 비용을 고려한다. 만약 갱신 질의의 비용이 크기 않다면, 본 중복 방법은 스키마 변형보다 더욱 큰 성능 향상을 가져 올 수 있다.

MARS 시스템[13]은 이러한 중복 개념을 연구한 또 하나의 비용 기반 연구이다. 그것은 RDBMS를 활용하는 것뿐만 아니라, 새로운 XML 전용 데이타베이스를 함께 저장소로 사용하는 환경에서 XML 데이타가 중복되었을 때, 미들웨어의 XQuery 질의 최적화기에 대한 연구를 수행하였다. 중복된 데이타를 활용하는 질의 재작성 방법은 참고문헌 [14]의 chase & backchase의 방법을 사용한다. 먼저 모든 RDBMS, XML 전용 데이타베이스에 중복된 데이타는 1차 술어 논리 문장으로 표현되어 있다. 어떤 XQuery가 주어지면 그 질의 또한 1차 술어 논리 문장으로 재 작성되고, 첫 번째 chase 단계에서 그 XQuery와 관련되는 중복 데이타들(1차 술어 논리 문장으로 표현된)을 추론(reasoning)을 통하여 그 XQuery의 1차 술어 논리 문장 몸체에 덧붙인다. 다음으로 backchase 단계를 통하여 가장 비용이 적게 드는 최소의 질의 계획을 몸체로부터 계산한다. 이 과정에서 일반적인 관계형 최적화기의 동적 프로그래밍(dynamic programming) 방법[15]을 이용한다. 비록 MARS 시스템이 중복을 통한 XML 질의 성능 향상을 다루고 있지만, 그것은 미들웨어에서의 질의 재작성 알고리즘에 초점이 맞추어져 있고, 본 연구는 RDBMS 활용시의 실질적 중복 방법의 연구와 최적의 중복 탐색 문제에 초점이 맞추어져 있다.

사실 과거 OODBMS에서도 객체 참조의 깊이가 깊어지게 될 경우 과도한 조인비용을 해결하기 위한 다양한 방법의 연구가 수행되었다. 예로, ASR(Access Support Relation)[16]과 참고 문헌 [17]은 그러한 연구일 것이다. 하지만 본 연구에서는 이러한 개념을 XML 데이타를 RDBMS로 맵핑하는 상황에서 새롭게 고찰하였으며, 실제 구현방법 및 새로운 특성 등을 관찰할 수 있었다. 또한 XML 갱신 연산자에 의한 중복 데이타의 갱신 일관성 (update consistency)의 비용을 분석하였다.

4. 구조적 중복 방법의 소개

본 연구에서는 XML 데이타의 릴레이션 맵핑시 사용할 수 있는 효과적 중복 방법을 ID 컬럼 중복, VALUE 컬럼 중복, SUBTREE 조각 중복의 세 가지 유형으로 분류하였다.

4.1 ID 컬럼 중복

우선 ID 컬럼의 정의는 다음과 같다. 그림 1(c)의 ID 컬럼과 같이 만약 어떤 컬럼이 기본키(primary key)로 사용된다면, 그 컬럼을 ID 컬럼으로 정의한다. 단, 이러한 컬럼은 어떠한 XML 갱신 연산자에 의해서도 한번 할당된 값이 바뀌어서는 않되고, 하지만 할당된 값의 삭제(즉, 튜플의 삭제)와 새로운 값의 할당(즉, 새로운 튜플의 삽입)은 허용된다. 본 연구의 경우 그림 1(c)의 ID 컬럼은 시스템에 의해 자동으로 생성되는 유일 값을 가정한다. 이와 반대로 4.2절에서 설명될 VALUE 컬럼은 값을 수정하는 갱신 연산자의 적용이 가능한 컬럼으로, 결국 ID와 PID 컬럼을 제외한 모든 컬럼이 이에 해당한다. 그림 1(c)에서 ID 컬럼을 참조하는 PID 컬럼은

XML 문서의 부모-자식 노드간의 관계성을 유지한다.

만약 어떤 XPath 'document("hamlet.xml")/play/ act/ scene/speech/speaker'가 그림 1(a)에 대하여 주어지면, 재작성된 SQL 문장은 'T1 ⋈ T2 ⋈ T13 ⋈ T28 ⋈ T35'가 되어야 한다. 하지만 만약 T1(ID)가 T35 테이블로 미리 중복되어 있다면, 'T1 ⋈ T35'의 조인만으로 수행될 수 있다. 즉, 이처럼 어떤 주어진 XML 경로를 평가하기 위해 요구되어지는 조인 수를 줄이기 위해, ID 컬럼을 미리 중복시키는 경우 이를 ID 컬럼 중복이라고 정의한다. ID 컬럼 중복은 다음과 같이 두 가지 형태를 갖는다.

IN-PLACE 방법: 이것은 조상 테이블의 ID 컬럼이 후손 테이블 내로 중복되어 단지 두 테이블 사이의 하나의 조인 연산이 이루어지는 것이다(예로, 'T1 ⋈ T35').

이러한 중복 형태는 XML 갱신 연산시의 갱신 일관성(update consistency)을 위하여 다음과 같은 특성이 관찰될 수 있다. 조상 테이블의 튜플들(예로, ACT 노드)이 지워질 때 후손 테이블의 튜플들(예로, SPEAKER 노드) 또한 trigger-based deletion[7]에 의해 자동 삭제되므로, 중복된 ID 값을 삭제하기 위한 별도의 SQL 문장이 요구되지 않는다. 또한 앞서 설명하였듯이 ID 값은 절대 변경되지 않는 것이므로, ID 값의 변경에 의한 후손 테이블에서의 갱신 일관성 비용이 들지 않는 것은 당연하다. XML 삽입 연산자에 관하여, 어떤 노드 x에 y 노드를 루트로 하는 XML 일부 조각이 연결되는 경우(예로 어떤 SCENE 노드에 어떤 SPEECH가 삽입), 만약 x를 포함한 x의 조상 노드들 중 y를 포함한 y의 하위 노드들 사이에 ID 중복 관계가 있는 경우(예로, T1(ID)가 T35 테이블로 중복되어 있는 경우), 조상 테이블의 ID 값을 중복시키기 위한 추가적 SQL 문장들('T1⋈T2⋈T13')이 요구된다. 하지만, 새로운 XML 문서를 삽입하는 'insert document("Macbeth.xml")'은 ID 값을 중복시키기 위한 별도의 SQL 문장을 요구하지 않는다. 왜냐하면 중복된 컬럼을 위한 값을 포함하는 INSERT 문들이 이미 XML 문서의 분할 저장 시에 작성되기 때문이다. 그래서 IN-PLACE ID 중복은 매우 낮은 갱신 일관성 비용을 가지게 된다. 이러한 사실은 6.1절의 실험에서 살펴 볼 수 있다.

SEPARATE 방법: 전 방법과는 달리 동일 경로상의 중복될 ID 컬럼들이 그 경로를 위한 별도의 새로운 테이블로 함께 중복된다. 예로, T1-T2-T13-T28-T35의 동일 경로상의 T1(ID)와 T35(ID)는 그림 2(b)에 보이는 것처럼 그 경로를 위한 별도의 ST1 테이블로 중복된다. 이 경우 어떤 주어진 XML 질의의 경로를 평가하기 위하여 별도 테이블과의 조인이 요구된다(예로, 'T1 ⋈ ST1 ⋈ T35'). 이러한 방법이 IN-PLACE 방법과 구별되는 목적은 갱신 비용과 관련된다. 예로, T1-T2-T13-T28-T35의 경로에 대하여, 만약 테이블 T1의 ID 컬럼이 T13, T28, T35로 중복되었다면, T1(ID)의 갱신은 모든 테이블로의 갱신을 유발할 것이다. 하지만 SEPARATE 경우는 별도 테이블로 중복된 T1(ID) 값이 한번만 갱신되면 된다. 그럼에도 불구하고, 이것은 ID 중복에서 만큼은 잘못된 관찰이다. 왜냐하면 앞서 언급한 조상 테이블의 튜플 삭제시의 후손 테이블의 자동 튜플 삭제와, ID 컬럼은 값의 변경 개념이 존재하지 않는다는 특성 때문이다. 오히려, XML 삽입 삭제 연산시의 갱신 일관성을 위한 별도 테이블에 대한 추가적 SQL 문장 때문에(예로, 위에서 예를 든 ACT 노드의 삭제와 'insert document("Macbeth.xml")'의 경우 별도 테이블 ST1에 대한 추가적 삽입 삭제 SQL 문), SEPARATE는 IN-PLACE보다 나쁜 성능을 가질 것이다. 이러한 사실 또한 6.1절의 실험을 통하여 살펴 볼 수 있다.

4.2 VALUE 컬럼 중복

VALUE 컬럼 중복은 값을 가지는 XML 엘리먼트 혹은 어트리뷰트 맵핑 컬럼을 후손 테이블로 중복시키는 것이다. 이러한 중복의 이점은 질의의 결과가 적은 수의 테이블들로부터 얻어질 수 있는 장점을 갖는다. 예로, 만약 모든 TITLE 노드들이 T35 테이블로 미리 중복되어 있다면, 다음 질의의 결과는 단지 T35 테이블로부터 얻어 질 수 있다.

ID	C59	R1	PID
1899	POMPEY	1	546
63404	ROSENCRANTZ	2	14975
63416	GUILDENSTERN	2	14979
63421	HAMLET	2	14981
...	...	...	...

T35 (R1: T1(ID))

(a) IN-PLACE 방법

ID	R1	R2
1000	1	1899
1001	2	63404
1002	2	63416
1003	2	63421
...	...	...

ST1 (별도의 테이블, R1: T1(ID), R2: T35(ID))

(b) SEPARATE 방법

그림 2 ID 컬럼 중복의 두 가지 형태

FOR $S in /play /* XQ3 */
RETURN $S/title, $S/act/title, $S/act/scene/title, $S/act/scene/speech/speaker

VALUE 컬럼 중복 또한 다음과 같이 IN-PLACE, SEPARATE의 두 가지 형태를 갖는다.

IN-PLACE 방법: 하나의 XML 노드 삭제는 후손 테이블의 튜플을 삭제하거나(예로 그림 1에서, C8 컬럼이 T35 테이블로 중복되어 있고, ACT 노드가 삭제될 경우), 중복된 컬럼의 값을 널(null) 값으로 대치하여야 한다(예로, C8 컬럼이 T35 테이블로 중복되어 있고, TITLE 노드가 삭제될 경우). 후손 테이블의 튜플 삭제의 경우는 4.1절에서 설명한 trigger-based deletion에 의한 자동적인 중복 데이타 삭제 때문에 갱신 일관성을 고려할 필요가 없다. 하지만 널 값 할당의 경우는 이를 위한 추가적 UPDATE SQL 문장이 요구된다. XML 삽입 연산의 경우는 IN-PLACE ID 중복과 비슷하다. 하지만 값을 바꾸는 XML 갱신 연산의 경우는 ID 컬럼 중복과 달리 후손 테이블의 VALUE 중복 값을 수정하기위한 추가적 비용이 요구된다.

SEPARATE 방법: 동일 경로상의 몇 개의 VALUE 컬럼들이 그 경로를 위한 새로운 별개의 테이블로 중복되고, 그림 3(b)에 보이는 것처럼 중복된 VALUE 값의 갱신 일관성 유지를 위하여 반드시 ID 중복이 함께 수반되어야 한다. SEPARTE ID 중복과 달리, 값을 바꾸는 XML 갱신 연산자가 적용될 수 있기 때문에 이러한 방법은 효율적일 수 있다. XML 삭제 혹은 삽입 연산의 경우, 별도 테이블의 중복된 ID 컬럼 혹은 VALUE 컬럼에 널 값을 할당하는 추가적 SQL 문장이 필요하거나, 널 컬럼에 새로운 값을 할당하는 추가적 SQL 문장이 필요할 수 있다. 또한 새로운 튜플의 삭제 혹은 삽입을 위한 추가적 SQL 문장이 필요할 수 있다.

4.3 SUBTREE 조각 중복

다음 질의를 고려하자.

FOR $S in /play/act/scene /* XQ4 */
WHERE contains($S/title/text(), "castle") RETURN $

만약 SCENE 노드 이하의 원래의 XML 문서 조각이 그림 4와 같이 미리 중복되어 있다면, 일련의 조인 유니온 연산과 XML 결과 문서를 구성하기 위한 구조화 및 태깅 작업은 불필요 할 것이다. 이러한 SUBTREE 조각 중복은 크기가 작을 경우 VARCHAR 컬럼, 크기가 클 경우 CLOB 컬럼으로 저장될 수 있다. 저장 테이블은 그림 4의 경우처럼 SUBTREE의 루트 노드 (SCENE 노드)가 속하는 테이블이다.

SUBTREE 조각 중복은 어떤 하위 트리가 반드시 한 번 중복되어야 하는 ONE-PLACE 방법과 어떤 하위 트리가 여러 번 중복될 수 있는 MULTIPLE-PLACE 방법으로 구분된다. 즉 그림 4에 보이는 것처럼, ONE-PLACE는 하나의 SUBTREE 조각 중복이 이미 설정되어 있었다면, 다시 그 하위의 트리가 SUBTREE 조각 중복이 필요한 경우 분할된다. MULTIPLE-PLACE는 존재하는 SUBTREE 조각 중복에 상관없이 수행된다. ONE-PLACE는 판독 질의 처리시 모든 관련 SUBTREE 조각들이 연결되어야 하기 때문에 판독 질의 비용이 MULTIPLE-PLACE보다 크다. 하지만 MULTI-

ID	R1	R2	R3	R4	R5	R6	R7	R8
1000	1	..Antony..	13	ACT II	58	SCENE I..	1899	POMPEY
1001	2	..Hamlet..	179	ACT II	786	SCENE II..	63404	ROSENCRANTZ
1002	2	..Hamlet..	179	ACT II	786	SCENE II..	63416	GUILDENSTERN
1003	2	..Hamlet..	179	ACT II	786	SCENE II..	63421	HAMLET
...	...	...	...	...	...	...	...	...

T35 (R1: T1(C1), R2: T2(C8), R3: T13(C23))

(a) IN-PLACE 방법

ID	C59	R1	R2	R3	PID
1899	POMPEY	..Antony..	ACT II	SCENE I..	546
63404	ROSENCRANTZ	..Hamlet..	ACT II	SCENE II..	14975
63416	GUILDENSTERN	..Hamlet..	ACT II	SCENE II..	14979
63421	HAMLET	..Hamlet..	ACT II	SCENE II..	14981
...	...	...	...	...	...

ST1 (별도의 테이블, R1: T1(ID), R2: T1(C1), R3: T2(ID), R4: T2(C8), R5: T13(ID), R6:

(B) SEPARATE 방법

그림 3 VALUE 컬럼 중복의 두 가지 형태

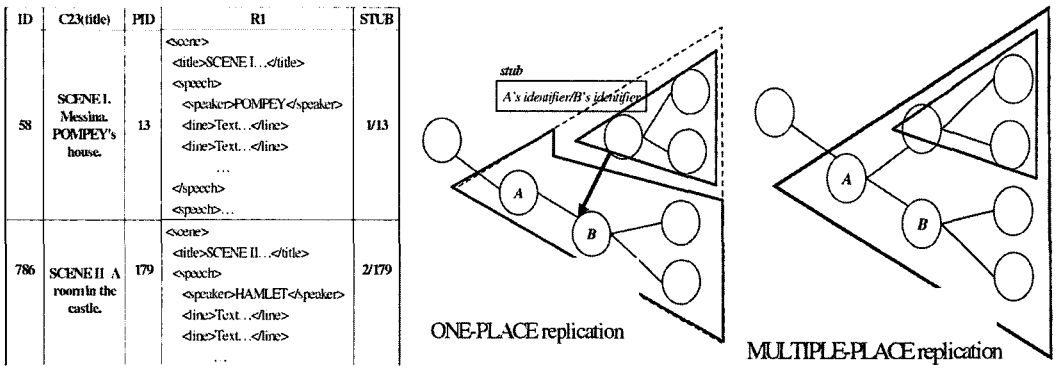


그림 4 SUBTREE 조각 중복의 두 가지 형태

ID	C0 (document name)	...	C1 (title)	SUBTREE_FRAG
1	a_and_c.xml	...	The Tragedy of Antony and Cleopatra	<play stub="1">... <act stub="13"> <title>ACT II</title>...</act> <act stub="14"> <title>ACT III</title>...</act> </play>
2	hamlet.xml	...	The Tragedy of Hamlet, Prince of Denmark	<play stub="2">... <act stub="179"> <title>ACT II</title>... </act>...</play>
...	...	...	...	...

그림 5 연결 스터브 정보의 예

PLE-PLACE는 단지 관련 SUBTREE 조각 중복을 반환하면 되기 때문에 빠르다. 그러나 중복 데이타의 양과 갱신 일관성 유지의 비용이 매우 클 가능성이 있다.

XML 갱신 질의 시에, 중복된 SUBTREE 조각들의 내부 노드의 갱신 일관성이 유지되어야 한다. 이러한 처리를 위하여, CLOB 컬럼 혹은 VARCHAR 컬럼의 중복된 SUBTREE 조각들을 DOM 파싱한 후, 관련 노드들을 갱신한 후 다시 저장하는 방법과, 갱신 연산이 적용된 테이블에 대하여 sorted outer union 방법을 이용하여 SUBTREE 조각 중복을 새로이 만들어 이전 것을 덮어 쓰는 방법이 있을 수 있다.

6.2절의 실험을 통하여, 중복된 SUBTREE 조각에 대한 심지어 작은 갱신에도 갱신 일관성의 비용이 매우 높아지는 것을 확인할 수 있었다. 따라서 만약 그러한 원인이 되는 내부 노드들이 있다면, 그러한 노드들을 제외한 SUBTREE 조각 중복만을 고려하는 방법이 있다. 예로, 그림 1에서, 만약 어떤 판독 질의를 위하여 '/play/act/scene'에 대한 SUBTREE 조각 중복이 요구되고, 하지만 '/play/act/scene/speech/line'에 많은 갱신이 있다면, LINE 노드 이하를 제외한 SUBTREE 조각 중복을 고려할 수 있다. 만약 '/play/act/scene/'의 XPath 질의가 주어지면, 그 SUBTREE 조각 중복에 단지 LINE 노드를 연결하면 된다. 하지만 판독 질의의 빈도 율이 매우 높다면, LINE 노드의 연결 비용 때문에, 그러한 갱신 일관성 비용을 무시하고 완전한 SUBTREE 조각 중복을 설정하는 것이 유리할 수 있다. 따라서 이러한 최적의 중복 설정은 전반적 질의 비용 분석을 토대로 이루어 져야 한다.

One-place의 경우 분할된 SUBTREE 조각들을 연결하기 위하여, 또한 위의 경우처럼 제외된 노드들을 SUBTREE 조각에 연결하기 위하여, 그림 4와 같은 어떤 연결 스터브(stub) 정보를 사용하여야 한다. 본 연구에서는 이러한 정보를 위하여 ID 컬럼 값을 이용하였다. 예로, 그림 5는 PLAY 노드 이하를 one-place SUBTREE 중복시킨 경우를 보여주는데, 그림 4의 '1/13'의 연결 스터브 값을 갖는 하위 SUBTREE 조각이 그림 5의 상위 SUBTREE 조각을 파싱한 DOM 트리상의 PLAY 노드의 STUB 어트리뷰트가 1이고 ACT 노드의 STUB 어트리뷰트가 13인 ACT 노드에 연결될 수 있다. 이러한 원래의 XML 데이타에는 존재하지 않는 STUB 어트리뷰트는 SUBTREE 조각의 연결 시에 제거된다.

5. 경험적 욕심쟁이 알고리즘을 이용한 효과적 중복 탐색

보통 데이타베이스의 물리적 설계단계에서 의미적 중복데이타를 설정하는 작업은 데이타 통계와 질의 분석을 통하여 DBA가 수작업으로 계산한다. 하지만 XML을 릴레이션으로 맵핑하는 응용의 경우 최적의 중복 집합을 찾아내는 것은 매우 어려운 작업이 될 수 있다. 왜냐하면, 분할된 많은 테이블과, 매우 복잡한 SQL문장으로 재 작성될 수 있는 많은 복잡한 XML 질의 문장들, 또한 고려 대상의 가능한 중복 방법들이 많을 수 있기 때문이다. 따라서 갱신 일관성을 고려한 전반적 질의 비용 측면에서 효과적 중복 데이타를 자동으로 탐색하는 도구가 필요하다.

하지만 이러한 탐색문제는 가능한 중복 데이타의 수에 있어 NP-hard 이다. 참고문헌 [18]은 데이타웨어하우스에서 판독, 갱신 질의가 주어졌을 경우 실체화 (materialize) 될 뷰들(views)의 최적의 집합을 탐색하는 문제를 다루었는데, 그러한 문제가 NP-hard이며, 탐색 방법으로 욕심쟁이 알고리즘(greedy algorithm)이 매우 효율적이라는 것을 밝힌바 있다. 이 문제는 본 연구의 최적의 구조적 중복 데이타 탐색 문제로 쉽게 변형(reduce)될 수 있음을 알 수 있는데, 그 이유는 개략적으로 다음과 같다. (1) 참고문헌 [18]에서의 뷰를 필요로 하는 판독 질의는 어떤 XQuery 질의를 위해 재 작성된 SQL 질의로 간주될 수 있고, (2) 실체화될 뷰는 본 연구에서의 separate ID와 VALUE 중복으로 간주될 수 있고, (3) 갱신 질의 시의 뷰의 갱신 일관성 비용은 역시 XML 갱신 질의 시의 구조적 중복 데이타의 갱신 일관성 비용으로 간주될 수 있기 때문이다. 따라서 본 연구에서도 탐색을 위한 하나의 경험적 (heuristic) 방법으로 욕심쟁이 알고리즘을 사용하였으며 이를 소개한다.

5.1 가능한 중복 데이타의 열거

본 연구에서의 욕심쟁이 알고리즘 적용의 첫 번째 단계는 모든 가능한 중복 데이타를 열거하는 것이다. 즉, 주어진 DTD에서 고려될 수 있는 모든 가능한 중복 데이타들을 계산해야 한다. 하지만 그러한 가능한 중복 데이타의 수는 매우 크므로, 하나의 경험적 방법으로 주어진 각각의 질의에 큰 영향을 줄 수 있는 중복 데이타들만으로 한정하는 방법을 사용하였다. 가능한 중복 데이타의 계산은 SUBTREE 조각 중복, VALUE 컬럼 중복, ID 컬럼 중복의 순서를 따른다.

• SUBTREE 조각 중복의 열거

(1) 모든 주어진 XML 질의들에 대하여 모든 가능한 MULTIPLE-PLACE SUBTREE 조각 중복들이 열거된다 (여기서 ONE-PLACE는 일단 고려되지 않음).

(2) (1)에서 구해진 SUBTREE 조각 중복들에 대하여 과도한 갱신을 일으키는 노드들을 뺀 경우가 다시 한번 열거된다. SUBTREE 조각 중복의 과도한 갱신 일관성 비용을 일으키는 노드들의 식별을 위하여, 본 연구에서는 각 갱신 XML 질의가 영향을 미치는 SUBTREE 조각 중복의 평균 크기, 영향 받는 튜플 수 (sf_u), 그 질의의 발생 빈도수(w) 등을 따져(6.2절의 실험 참조) 그 비용이 특정 임계값(threshold)을 넘어설 경우 이를 식별하는 방법을 사용하였다.

예로, 그림 6의 DTD에 대한 XML 질의들을 고려할 경우, 다음과 같은 SUBTREE 조각 중복들이 열거된다 (Q5는 과도한 갱신 비용을 갖는다고 가정하며, 따라서 j 노드는 과도한 갱신 일관성 비용을 일으키는 노드이다): 'multiple-place /a/b/d', 'multiple-place /a/b/d - {j}', 'multiple-place /a/c', 'multiple-place /a/b/d/c', 'multiple- place /a/b/d/g', 'multiple-place /a/b/d/g - {j}', 'multiple-place /a', 'multiple-place /a - {j}'.

고려되지 않은 ONE-PLACE SUBTREE 조각 중복에 대한 평가는 5.2절에서 설명될 GetReplicasCost 함수의 실행 시에 이루어진다. 즉 'multiple-place /a/b/d/g'가 먼저 optimalReplicas에 포함된 상황이라면, 그림 6의 점선 삼각형처럼 'multiple-place /a/b/d'를 평가할 경우, 'one-place /a/b/d'도 자동으로 평가된다. MULTIPLE-PLACE와 ONE-PLACE의 의미적 충돌이 일어날 경우 optimalReplicas에 먼저 포함된 것을 우선시 하는데, 즉 이미 'one-place /a/b/d'가 선택된 상황이라면, 후의 'multiple-place /a/b/d/c'는 고려될 수 없다.

Q1. *return* /a/b/d
Q2. *return* /a//c
Q3. *return* /a/b/d/g
Q4. *return* /a
Q5. *delete* j

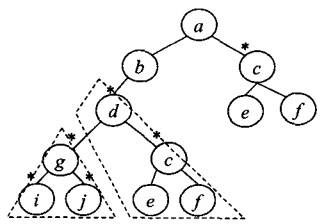


그림 6 SUBTREE 조각 중복의 열거

• VALUE 컬럼 중복의 열거

(1) 어떤 XML 질의에 관련된, 동일 경로상의 VALUE 컬럼들 모두가 최하위 맵핑 테이블로 중복되는 경우만이 열거된다. 예로 4.2절의 XQ3의 경우, 'T1-T2-T13-T28-T35'의 동일 경로상의 C1, C8, C23 VALUE 컬럼들이 C59를 포함하는 최하위 맵핑 테이블 T35로 중복되는 경우만이 열거된다:

$S as /play, $T as $S/act, $V as $S/act/scene/title,

inplace_value $S/title, $T/title, $V/title into T35.

사실, 어떤 XML 질의에 관하여 영향을 줄 수 있는 여러 부분적인 VALUE 컬럼 중복들이 고려될 수 있지만, 예로 XQ3에 대한 다음과 같은 부분적 VALUE 컬럼 중복들을 고려할 수 있다:

inplace_value $S/title into T2, inplace_value $S/title into T13, ...

inplace_value $T/title into T13, inplace_value $T/title into T35, ...

inplace_value $S/title, $T/title into T13, inplace_value $S/title, $T/title into T35 ... .

하지만, 4.2절에서 설명하였듯이, ID 컬럼 중복에 비해 VALUE 컬럼 중복이 가질 수 있는 장점이 적은 수의 테이블로부터의 질의 결과를 얻는 것이므로, 본 연구에서는 동일 경로상의 모든 VALUE 컬럼들이 하나의 최하위 테이블로 중복되는 경우만을 고려하였다.

(2) 주어진 각 질의에 대하여 위의 과정이 반복되며, 만약 SEPARATE VALUE 컬럼 중복이 가능하면 열거된다. 예로, XQ3와 XQ14를 위하여 다음이 열거된다.

FOR $P in /play /* XQ14 */
RETURN $P/title, $P/act/title, $P/act/scene/title
inplace_value $S/title, $T/title, $V/title into T35
inplace_value $S/title, $T/title into T13
separate_value $S/title, $T/title, $V/title, $V/speech/speaker into ST1
separate_id T1(ID), T2(ID), T13(ID), T28(ID), T35(ID) into ST1
/* ST1은 SEPARATE VALUE 컬럼 중복을 위한 별도 테이블임 */

• ID 컬럼 중복의 열거

(1) 주어진 각각의 질의에 관하여, 질의 노드들 사이의 경로 평가를 위해 수행되는 조인 수를 줄이기 위한 β 이내의 링크에 의하여 연결되는 모든 ID 컬럼 중복들이 열거된다. 예로, 그림 7의 DTD 그래프에 대한 질의에 관하여, 질의 노드 A-E, E-O, E-P 사이의 조인 수를 줄이기 위한 점선의 ID 컬럼 중복들이 열거된다 (여기서 각 노드가 맵핑된 테이블을 $T_{(노드이름)}$이라고 하자). β의 값은 2로 설정된 경우인데, 따라서 한번 혹은 두 번 만에 연결되는 경우가 고려되었다. 역시 본 연구에서도, β의 값으로 2를 선택하였고, 이는 실험 과정 중 만족스러웠다. 하지만 만약 그 인자 값이 너무 적다면, 어떤 유용한 ID 컬럼 중복을 놓칠 가능성이 있으며, 만약 크다면 가능한 ID 컬럼 중복들의 수가 커지므로 탐색 공간이 커질 것이다.

본 연구에서는 4.1절의 관찰에 의하여 비효율적인 SEPARATE ID 컬럼 중복은 고려하지 않았다.

5.2 욕심쟁이 알고리즘

이 방법은 알고리즘 5.1에서 보이는 바와 같이 한번에 하나씩의 중복 데이타를 고려해 나간다. 우선 GetNoReplicasCost 함수는 어떤 중복 데이타도 고려되지 않은 경우의 전체 질의 비용 minCost (xQuerySet의 각 판독 및 갱신 질의 비용의 총합)를 데이타 통계(xStats)와 질의 빈도(xWkld)를 이용하여 계산한다. 다음으로, GetAllPossibleReplicas 함수에서, 앞서 설명된 주어진 XML 질의들(xQuerySet)에 대한 모든 가능한 중복 데이타들(replicas)을 계산하여 열거한다. 그 함수는 또한 어떤 중복 데이타도 고려되지 않은 경우에서의 각 가능한 중복 데이타마다의 각 전체 질의 비용을 데이타 통계(xStats)와 질의 빈도(xWkld)를 이용하여 계산한다. 그래서 모든 가능한 중복 데이타들(replicas) 중 가장 적은 전체 질의 비용을 가져오는 중복 데이타에 대한 참조(index)를 반환한다. 다음으로 그 replicas

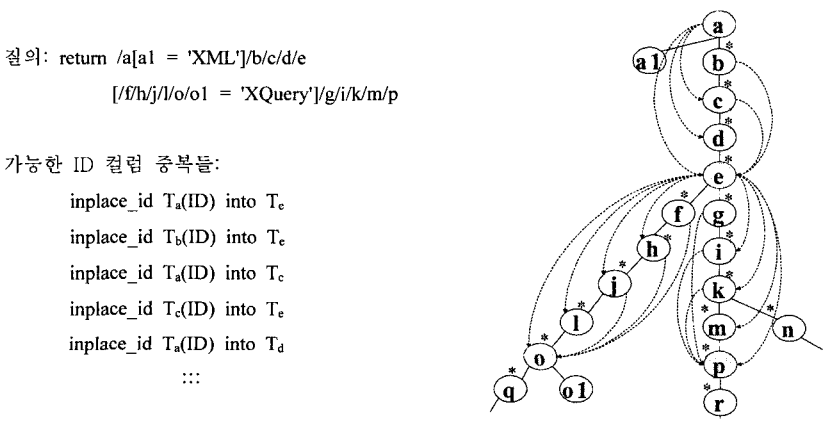


그림 7 ID 컬럼 중복의 열거

[index].cost가 minCost 보다 적다면 while 루프로 진입하고, 그 중복 데이타는 replicas로부터 삭제되며, 처음으로 최적의 중복 집합(optimalReplicas)에 포함된다. minCost의 값은 replicas[index].cost로 대체된다.

GetReplicasCost 함수에서는 현재 optimalReplicas 하의 모든 가능한 각 중복 데이타들(replicas)의 전체 질의 비용이 다시 계산된다. 그리고 그 함수는 가장 작은 질의 비용을 가져오는 중복 데이타의 참조(index)를 반환한다. 만약 선택된 중복 데이타에 의한 전체 질의 비용이 minCost보다 작다면, minCost의 값은 그 값으로 대체되고, 그 중복 방법은 최적의 중복 집합에 포함된다. 이러한 처리는 전체 질의 비용이 더 이상 좋아지지 않을 때까지 반복된다.

알고리즘 5.1은 모든 가능한 중복 데이타들이 모두 최적의 중복 집합에 포함됨을 가정한다면(이럴 가능성은 희박하지만), 최악의 경우 시간 복잡도는 $O(nm^2)$을 갖는다(n은 주어진 질의 수, m은 가능한 중복 데이타의 수). 즉, *while* 루프에서 매번 replicas의 중복 데이타들이 n개의 질의들에 대하여 재평가되므로, $(m-1)*n+(m-2)*n+(m-3)*n+ \ldots +(1)*n$ 번 반복되고, GetAllPossibleReplicas 함수에서의 $(m)*n$을 추가적으로 더하면, $n\sum_{i=1}^{m} i$ 이다.

알고리즘 5.1 경험적 욕심쟁이 알고리즘

```
procedure GreedySearch
input:  xQuerySet: 주어진 XML 판독 및 갱신 질의들
        xWkld: 각 XML 질의어에 대한 빈도 정보
        xStats: 히스토그램으로 요약된 XML 데이타 통계 정보
        replicas: 주어진 XML 질의들에 대한 가능한 중복 방법들
        X2RMap: XML 데이타의 릴레이션으로의 맵핑 정보
output: optimalReplicas: 주어진 XML 질의들에 대한 최적의 중복 집합 해
begin
        minCost=GetNoReplicasCost(xQuerySet, xWkld, xStats, X2RMap)
        index=GetAllPossibleReplicas(replicas, xQuerySet, xWkld, xStats, X2RMap)
        while(replicas[index].cost < minCost) do
            AddtoList(optimalReplicas, replicas[index])
            minCost =replicas[index].cost
            index=GetReplicasCost(replicas, optimalReplicas, xQuerySet, xWkld, xStats, X2RMap)
        endwhile
        return optimalReplicas
end.
```

6. 실험 및 분석

본 장에서는 각 중복 방법과 제안된 탐색 방법의 효율성을 분석하기 위해 수행된 몇 가지 실험을 소개한다. 비록 XML 벤치마크를 위한 XMark[19]와 같은 실험 데이타들이 존재하지만, 특별히 본 연구의 실험 의도에 적합한 실험 데이타를 위하여 인위적으로 만든 실험 데이타를 사용하였다. 실험은 1 GHz 펜티엄 III 듀얼 CPU, 2GB 메모리를 가진 윈도우 2000 서버와 오라클 8i 서버, 프로그래밍 언어는 자바 언어를 사용하였다.

6.1 ID 컬럼 중복의 실험

그림 8의 실험 모델을 고려하자. 그림 8(b)는 그림 8(a)의 XML 데이타를 맵핑하는 관계형 스키마이다. IN-PLACE ID 컬럼 중복을 위하여 T1(ID)가 T3, T5, T7 테이블들로 중복되고, T3(ID)는 T5, T7으로, T5(ID)는 T7으로 중복된다. SEPARATE ID 컬럼 중복을 위하여 T1(ID), T3(ID), T5(ID), T7(ID)가 별도의 테이블로 함께 중복된다. 두 경우에 대하여, 어떤 컬럼을 조건으로 하는(f_r = 조건에 해당하는 한 테이블에서의 튜플 수 / 그 테이블의 모든 튜플 수) 몇 개의 XML 판독 질의가 그림 8(c)와 같이 수행된다. 또한 XML 갱신 질의가 T1, T5, T7에 대하여 수행된다 (f_u 역시 f_r과 같음). 그림 8(c)의 질의에 관하여, Q1은 A1 노드를 조건으로 하여 G2 노드를 검색한다. Q5는 E1 노드를 조건으로 하여 E 노드를 삭제하며, Q8은 F1 노드의 조건에 부합하는 모든 F 노드에 하나의 G 노드를 연결한다. 여기서 w은 각 질의의 발생 빈도수를 의미한다.

위의 실험 모델에서, T1, T2, T3, T4, T5, T6, T7의 튜플 수는 각각 1000, 3,000, 9,000, 27,000, 81,000, 243,000, 729,000이다. 따라서 부모 테이블의 한 튜플 당 자식 테이블의 평균 튜플 수는 3이다. 사용된 데이타의 전체 크기는 약 100 MB이었다.

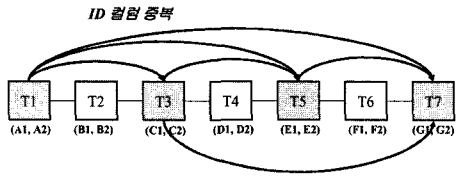
그림 9(a)의 그래프는 중복을 사용하지 않은 경우, IN-PLACE, SEPARATE의 경우에 각 질의의 측정된 실행 시간을 보여준다. 판독 질의의 경우 중복 데이타를 사용하여 그 질의 처리 비용이 줄어듦을 알 수 있다. 그리고 중복을 사용하지 않은 경우와 IN-PLACE는 갱신 질의에 있어 질의 처리 비용이 거의 차이가 없는데, 이것은 다음과 같이 설명될 수 있다.

- 4.1절에서 설명된 것처럼, IN-PLACE ID 중복은 갱신 연산 시 중복 데이타의 갱신 일관성을 유지하기 위한 비용이 거의 들지 않는다.
- Q6와 Q9에 관하여, ID 컬럼 중복 데이타가 그 조건을 만족하는 E와 D 노드들을 검색하는 것에 사용될 수 있다. 다음은 Q6과 Q9를 위해 재작성된 SQL 문장이다.

```
<!DOCTYPE A [
        <!ELEMENT A  (B+, A1?, A2)>
        <!ELEMENT B  (C+, B1?, B2)>
        <!ELEMENT C  (D+, C1?, C2)>
        <!ELEMENT D  (E+, D1?, D2)>
        <!ELEMENT E  (F+, E1?, E2)>
        <!ELEMENT F  (G+, F1?, F2)>
        <!ELEMENT G  (G1?, G2)>
        <!ELEMENT A1 (#PCDATA)>
        <!ELEMENT A2 (#PCDATA)>
        <!ELEMENT B1 (#PCDATA)>
              : : :
]>
```

(a) 실험 DTD

+ 심볼의 경우 한 부모 노드당 자식 노드의 평균 진출 차수: 3

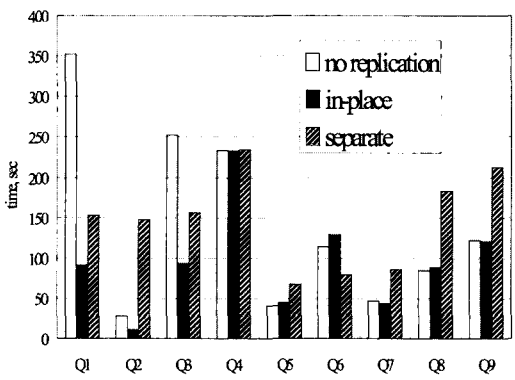


(b) 맵핑 릴레이션

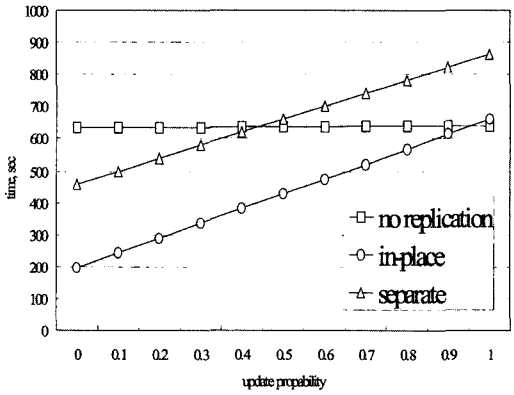
Q		w	f_r 혹은 f_u
판독질의	Q1. *return /A[A1=string1]/B/C/D/E/F/G/G2*	8	0.05
	Q2. *return /A[A1=string1]/B/C/D/E/E2*	8	0.05
	Q3. *return /A/B/C[C1=string1]/D/E/F/G/G2*	8	0.05
갱신질의	Q4. *delete /A[A1=string1]* *(이러한 질의는 XML 문서 전체를 지우는 것을 의미한다.)*	2	0.05
	Q5. *delete /A/B/C/D/E[E1=string1]*	1	0.05
	Q6. *delete /A[A1=string1]/B/C/D/E*	1	0.05
	Q7. *insert A* *(이러한 질의는 하나의 XML 문서 전체를 삽입하는 것을 의미한다.)*	5	
	Q8. *insert G into /A/B/C/D/E/F[F1=string1]*	1	0.05
	Q9. *insert E into /A[A1=string1]/B/C/D*	1	0.05

(c) 실험 질의

그림 8 ID 컬럼 중복의 실험



(a) 각 질의 비용 (= 단일 질의 비용 * w)



(b) P_u에 따른 전체 질의 비용

그림 9 ID 컬럼 중복 실험의 결과 그래프

```
Q6: delete from T5 // T1_ID는 T1(ID)가 중복된 컬럼이다.
    where exists (select * from T1 where T1.A1=
    'PRINCE HENRY' and T1.ID=T5.T1_ID);
Q9: ResultSet rs=st.executeQuery("select T1.ID, T3.ID,
T4.ID from T1, T3, T4
        where T1.A1='PRINCE HENRY' and T1.ID=T3.T1_
        ID and T3.ID=T4.PID");
    while(rs.next()){
     i1=rs.getLong(1); i2=rs.getLong(2); i3=rs.getLong(3);
        T5, T6, T7 테이블로의 SQL insert 문장들;
        // 이러한 삽입 문장은 중복 컬럼을 위한 값을 포함한다.
    }
```

- Q7에 대하여, 중복 데이타를 위한 값을 포함하는 INSERT 문장들이 데이타 입력시의 분할 과정에 이미 작성되므로, ID 컬럼을 중복하기 위한 별도의 SQL 문장을 요구하지 않는다.
- 비록 Q8은 T1(ID), T3(ID), T5(ID)를 T7으로 중복

하기 위하여, 'T1 ⋈ T2 ⋈ T3 ⋈ T4 ⋈ T5 ⋈ T6'의 추가적인 조인을 요구하지만, T5는 이미 중복된 T1(ID)와 T3(ID)를 포함하기 때문에 단지 'T5 ⋈ T6'의 조인을 요구한다.

```
Q8: ResultSet rs=st.executeQuery("select T5.T1_ID, T5.T3_ID, T5.ID, T6.ID from T5, T6
        where T6.F1='PRINCE HENRY' and T5.ID=T6.PID");
  while(rs.next()){
    i1=rs.getLong(1); i2=rs.getLong(2); i3=rs.getLong(3);
    i4=rs.getLong(4);
    insert into T7 ...; // 이러한 삽입 문장은 중복 컬럼을 위한 값을 포함한다.
  }
```

전반적으로 SEPARATE가 IN-PLACE보다 더 나쁜 성능을 보였다. 이는 SEPARATE의 경우 그 동일 경로를 위한 별도의 테이블과의 추가적 조인과 또한 그 테이블에 대한 별도의 갱신 일관성이 요구되기 때문이다. Q2의 SEPARATE의 경우, 판독 질의 비용이 다소 높은데, 이것은 그 별도의 테이블에서 T7(ID)에 의해 T1(ID)와 T5(ID)의 수가 증가하기 때문이다.

전반적 질의 비용의 관점에서 중복 데이타의 효율성을 보기 위하여, 다음 수식이 각 질의의 측정된 시간을 사용하여 계산될 수 있다.

$$C_t = C_r * (1 - P_u) + C_u \times P_u \quad (0 \le P_u \le 1) \qquad (6\text{-}1)$$

여기서 C_r은 Q1, Q2, Q3의 측정된 시간의 합이며 C_u는 갱신 질의들의 측정된 시간의 합이다. P_u는 전체 질의 중 갱신 질의들이 차지하는 비율을 의미하고, 따라서 갱신 질의의 증가에 따른 전반적 질의 비용의 측면에서 중복 방법의 효율성을 살펴 볼 수 있다.

그림 9(b)의 그래프는 만약 갱신 질의가 과도 하지 않으면, 당연히 전반적 질의 비용이 낮추어 질 수 있음을 보여준다. 재미있는 사실은, 만약 IN-PLACE의 갱신 일관성 비용을 증가시키는 질의(즉, Q8과 Q9과 같이 중복 데이타를 위한 추가적인 조인을 필요로 하는 질의)가 흔치 않다면, IN-PLACE ID 중복은 많은 갱신 질의가 존재하는 상황에서도 매우 유용하다는 것이다.

중복을 사용하지 않는 경우의 전체 질의 비용의 그래프가 수평인데, 이것은 실험에 사용된 판독질의들의 비용의 합과 갱신 질의들의 비용의 합이 같도록 의도된 것이며, 하지만 만약 판독 질의들의 비용이 더 크다면, 낮은 P_u에서의 그 차이는 더욱 클 것이다. 또한 그림 8의 실험 모델은 다소 단순한 경우이나, 만약 조인 경로상의 테이블의 수가 증가하거나, 한 XML 노드 당 평균 진출 차수가 증가하거나, f_r이 커진다면, XML 판독 질의 처리 비용의 감소 효과는 더욱 클 것이다.

지면 관계상 본 논문에서는 VALUE 컬럼 중복에 관한 실험 내용을 생략한다. 하지만, 그 효과는 ID 컬럼 중복의 실험과 비슷한 결과를 보여주었으며, 단지 4.2절에서 설명된 것처럼, ID 컬럼 중복과는 달리 SEPARATE가 유용한 경우가 존재함을 확인하였다.

6.2 SUBTREE 조각 중복의 실험

SUBTREE 조각 중복의 분석을 위하여, 그림 10(b)에 보이는 것처럼 A와 D 노드 이하가 ONE-PLACE와 MULTIPLE-PLACE로 실험되었다. A1 노드를 조건으로 A 노드의 XML 하위 트리, D1 노드를 조건으로 D 노드의 XML 하위 트리를 검색하는 XML 판독 질의가 수행된다. 또한 F2 노드에 대한 갱신 질의가 처리되며, 실험 데이타의 크기는 대략 150 MB이었다. 그림 10(c)의 sf_u 컬럼은, 갱신 질의에 의한, 갱신 일관성 유지를 위하여 영향 받는 조각 중복 율을 의미한다. 예로 Q4 질의에 의해 F2 노드가 갱신될 때, 만약 ONE_PLACE가 이미 설정되어 있었다면, T1의 갱신되는 SUBTREE 조각 중복의 수는 전체 튜플의 5%가 되게 하였다. 만약 MULTIPLE_PLACE가 설정 되어있었다면, T1과 T4의 갱신되는 조각 중복의 수는 각각 전체 튜플의 5%가 되게 하였다('O'는 ONE-PLACE를 'M'은 MULTIPLE-PLACE를 의미).

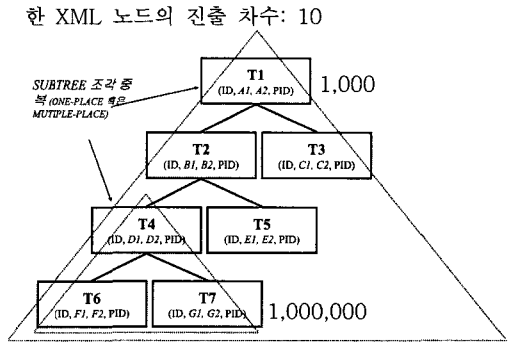
그림 11(b)는 적절한 SUBTREE 조각 중복을 사용하는 것이 전체 질의 처리 성능을 매우 향상 시킬 수 있음을 보여준다. 낮은 갱신 율에서는 multiple-place ('multiple-place 1')가 one-place보다 더욱 효율적이고, 높은 갱신 율에서는 예상과는 달리 one-place와 multiple-place의 차이가 그리 크지 않다. 이러한 이유는 one-place 경우 그림 11(a)의 Q1 질의 시의 두 SUBTREE 조각을 연결하는 비용이 매우 크기 때문인데, 따라서 one-place의 경우 SUBTREE 조각의 연결 비용이 높기 때문에, 실제로 multiple-place가 더욱 유리한 경우가 많을 것이라고 사료된다. 4.3절의 후반부에서 설명된 것처럼, 'multiple-place 2'는 SUBTREE 조각 중복 데이타로부터 과도한 갱신 비용을 갖는 F2 엘리먼트를 제외한 경우이다. 이 경우 판독 질의 처리 비용의 증가는 F2 엘리먼트를 그 조각 중복 데이타와 연결하는 비용 때문이고, 갱신 질의 처리 비용의 감소는 쉐어드 인라이닝 방법의 Q4 질의 처리 비용과 같아지기 때문이다.

그림 11(a)의 Q4 질의의 비용은 중복을 사용할 때 매우 높고 이것은 중복 방법의 전체 질의 처리 비용을 극대화 시키는 주요 요인이 된다. 그 이유는 CLOB 컬럼 혹은 VARCHAR 컬럼의 조각 중복을 DOM-파싱하고, 그 파싱된 트리로부터 몇 개의 엘리먼트의 값을 수정하고, 다시 그 트리를 CLOB 혹은 VARCHAR 컬럼으로 저장하는 비용이 매우 크기 때문이다.

```
Q4: ResultSet rs1=s1.executeQuery("select distinct T4.ID from T4, T6
```

```
<!DOCTYPE A [
        <!ELEMENT A  (B+, C+, A1, A2)>
        <!ELEMENT B  (D+, E+, B1, B2)>
        <!ELEMENT D  (F+, G+, D1, D2)>
        <!ELEMENT C  (C1, C2)>
        <!ELEMENT D  (D1, D2)>
        <!ELEMENT E  (E1, E2)>
        <!ELEMENT F  (F1, F2)>
        <!ELEMENT G  (G1, G2)>
        <!ELEMENT A1 (#PCDATA)>
        <!ELEMENT A2 (#PCDATA)>
        <!ELEMENT B1 (#PCDATA)>
        <!ELEMENT B2 (#PCDATA)>
                : : :
]>
```

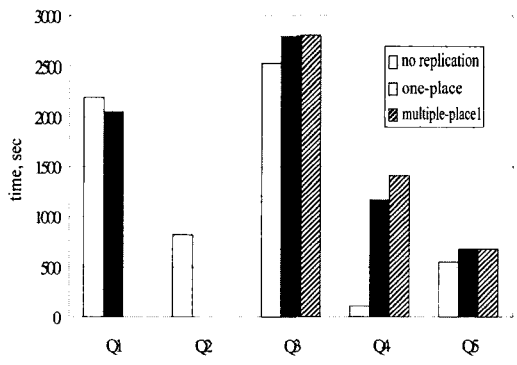
(a) 실험 DTD



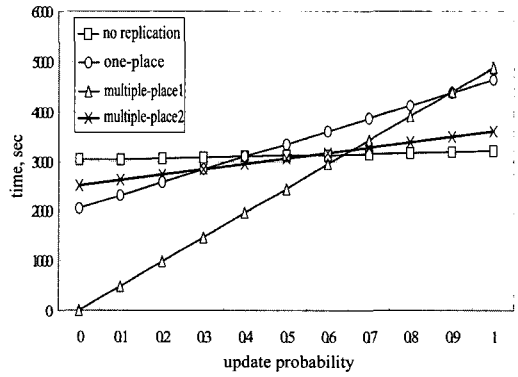
(b) 맵핑 릴레이션

Q		w	f_r 혹은 f_u	sf_u
판독질의	Q1. *return /A[A1=string1]*	12	0.05	.
	Q2. *return /A/B/D[D1=string1]*	12	0.05	.
갱신질의	Q3. *delete /A[A1=string1]* *(이러한 질의는 XML 문서 전체를 지우는 것을 의미한다.)*	5	0.05	.
	Q4. *update /A/B/D/F[F1=string1]/F2 set string2*	5	0.05	0.05($T1_O$), 0.05($T1_M$), 0.05($T4_M$)
	Q5. *Insert A* *(이러한 질의는 하나의 XML 문서 전체를 삽입하는 것을 의미한다.)*	5	.	.

그림 10 SUBTREE 조각 중복의 실험



(a) 각 질의 비용 (= 단일 질의 비용 * w)



(b) P_u에 따른 전체 질의 비용

그림 11 SUBTREE 조각 중복 실험의 결과 그래프

```
where T6.F1='PRINCE HENRY' and T4.ID=T6.PID");
 while(rs1.next()){
    i1=rs1.getLong(1);
    ResultSet rs2=st2.executeQuery("select T4.FRAG1
    from T4
                        where id="+i1+" for update");
    rs2.next(); clob=rs2.getCLOB(1);
    DOM-parsing을 하고, 갱신 후, 변경된 트리를 다시
    CLOB 컬럼으로 저장하는 문장 ;
 }
ResultSet rs1=s1.executeQuery("select distinct T1.ID
from T1, T2, T4, T6 where T6.F1='PRINCE HEN-
```

```
RY' and T1.ID=T2.PID and T2.ID=T4.PID and
T4.ID=T6.PID"); while(rs1.next()){
   i1=rs1.getLong(1);
   ResultSet rs2=st2.executeQuery("select T1.FRAG1
   from T1
                        where id="+i1+" for update");
   rs2.next(); clob=rs2.getCLOB(1);
   DOM-parsing을 하고, 갱신 후, 변경된 트리를 다시
   CLOB 컬럼으로 저장하는 문장 ;
}
update T6 set T6.F2= 'DOMITIUS ENOBARBUS'
where T6.F1='PRINCE HENRY'
```

이러한 비용은 갱신되는 SUBTREE 조각 중복의 수(*sfu*)와 그 조각의 평균 크기에 영향을 받는다.

6.3 욕심쟁이 알고리즘을 이용하는 탐색 방법의 성능 분석

본 절에서는 5장에서 제시된 탐색 방법을 실험하였으며, 실험을 위해 그림 1의 셰익스피어 연극 XML 데이타[12]를 사용하였다. 비록 그 데이타의 크기가 7 MB이지만, 같은 파일을 여러 번 저장하거나, 값을 갖지 않는 INDUCT, PROLOGUE, EPILOGUE 노드에 인위적인 값을 저장함으로써, 그 크기를 200 MB로 증가 시켰다. 실험 질의를 위하여, 그림 12(a)와 같은 몇 개의 질의를 정의하였으며(15 개의 판독질의와 10 개의 갱신 질의),

	Q	*w*
판독질의	XQ1. *Return document("a_and_c.xml")/play/act/scene/speech/line*	100
	XQ2. *Return document("hamlet.xml")//speech[speaker="HAMLET"]*	100
	XQ3. (4.2절에 있음)	20
	XQ4. (4.3절에 있음)	50
	XQ5. *FOR $S in /play, $T in $S/prologue//speaker, $U in $S/act//speaker, $V in $S/epilogue//speaker WHERE $T/text()=$U/text() and $U/text()=$V/text() RETURN $S/title, $T*	20
	XQ6. *Return document("hamlet.xml")*	100
	...	...
갱신질의	XQ16. *Delete document("a_and_c.xml")*	10
	XQ17. *Delete document("hamlet.xml")//speech[speaker="CORNELIUS"]*	20
	XQ18. *FOR $S in /play/act/scene/speech[contains(./line/text(), "love")] UPDATE $S { INSERT <line>Marking</line> }*	80
	XQ19. *FOR $S in document("othello.xml")/play/act[title="ACT II"]/scene[contains(./title/text(), "SCENE III")] UPDATE $S { INSERT <speech><speaker>IAGO</speaker> ... </line> </speech> }*	50
	...	...

(a) 실험 질의

$S as /PLAY, $T as $S/ACT, $V as $T/SCENE

[ID replication]

XQ1: *r1. inplace_id T1(ID) into T36*
r2. inplace_id T2(ID) into T36
r3. inplace_id T2(ID) into T28
r4. inplace_id T13(ID) into T36
r5. inplace_id T1(ID) into T13
XQ2: *r6. inplace_id T1(ID) into T24*
r7. inplace_id T1(ID) into T28
...

[SUBTREE replication]

XQ4: *r20. oneplace_subtree $V*
r21. oneplace_subtree $V - {$V/SPEECH/LINE}
XQ6: *r22. oneplace_subtree /*
r23. oneplace_subtree / - {$V/SPEECH/LINE}
...

[VALUE replication]

XQ3: *r14. inplace_value $S/TITLE, $T/TITLE, $V/TITLE into T35*
XQ19: *r15. inplace_value $T/TITLE into T13*
...

(b) 주어진 질의에 관하여 얻어진 모든 가능한 중복 방법

그림 12 욕심쟁이 알고리즘을 이용하는 경험적 탐색 방법의 실험

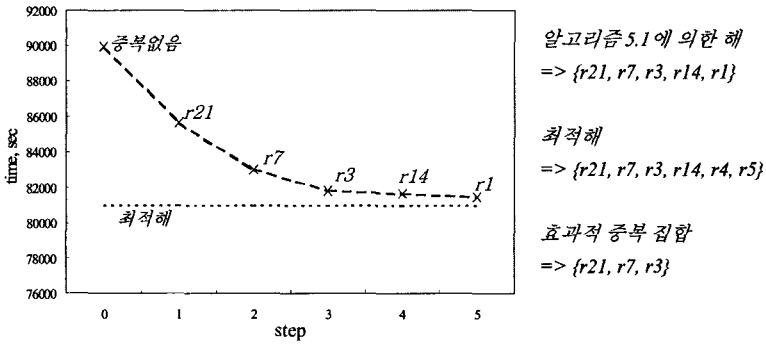


그림 13 탐색 방법의 결과 그래프

각 질의에 대한 모든 가능한 중복 방법들을 그림 12(b)와 같이 얻을 수 있었다(38개).

그림 13의 그래프는 경험적 욕심쟁이 알고리즘에 의한 매 while 루프별 전체 질의 비용의 변화를 보여준다. 제로 단계의 비용(즉, 어떤 중복 데이타도 사용하지 않은 경우)은 89,966초이고 알고리즘 5.1에 의한 해의 최종 비용은 81,492초이다. 비록 알고리즘이 최적의 해(80,994 초)를 발견하지 못하였지만, 그 차이는 매우 작음을 알 수 있다(498초). while 루프의 반복 횟수는 다섯 번으로 매우 적음을 알 수 있다.

재미있는 관찰은 연속적인 두 단계 사이의 매우 큰 차이를 가져오는 중복 데이타들이 있다는 것이다(예로, r21, r7, r3). 이러한 중복 데이타들은 아마도 빠른 질의 처리를 위하여 반드시 고려되어야 하는 것들일 것이다. 비록 r14와 r1이 어는 정도의 성능 향상을 가져오지만, 이러한 질의 비용의 계산이 히스토그램과 같은 데이타 통계에 의한 근사 비용 계산[15, 20]이라는 점을 감안하면, 그들은 실질적인 효과가 있는 중복 데이타가 아닐 수도 있다. 따라서 효과적 중복 데이타에 대한 다음을 정의한다.

정의 6.1 먼저 O를 알고리즘 5.1의 실행 완료에 의한 최종 중복 집합 해라고 하고, 반면 O'는 그 알고리즘의 실행 중간의 어떤 가능한 중복 데이타가 고려되기 전까지 결정된 불완전 중복 집합 해라고 하자. 그리고 효과적 중복 집합(effective replication set) E의 초기 상태는 공집합이라고 하자. 만약 O'에 의한 알고리즘 5.1의 minCost의 감소가 각 while 루프시 어떤 임계값 α 이상이라면, 그때만 그러한 E를 O'로 대치한다. 결국 while 루프 종료시의 최종적인 E가 효과적 중복 집합으로 정의 된다. 예로, 만약 α를 1000 초로 설정하면, {r21, r7, r3}은 본 실험의 효과적 중복 집합이 된다. 이러한 효과적 중복 집합은 질의 성능 향상을 위하여 반드시 고려되어야 한다.

7. 결 론

본 논문에서는 XML 데이타의 릴레이션으로의 맵핑 시 구조적 중복 데이타를 사용한 질의 처리 성능 향상과, 대량의 XML 데이타와 많은 복잡한 XML 질의어가 주어질 경우 최적의 중복 집합을 찾기 위한 경험적 탐색 방법을 소개하였다. 몇 가지 실험은, 만약 매우 과도한 갱신 비용을 갖는 XML 갱신 질의들이 XML 판독 질의들에 비하여 상대적으로 적다면, 이러한 중복 전략이 매우 효율적일 수 있음을 보여주었다. 특별히 in-place ID 컬럼 중복은 과도한 갱신 질의들이 존재하는 XML 경로에 대해서도 그 경로 평가를 위한 테이블의 조인 비용을 줄이기 위한 매우 효율적인 중복 방법으로 사용될 수 있음을 확인할 수 있었다. 또한 과도한 갱신 비용을 갖는 갱신 질의가 적게 적용되는 XML 문서의 부분에 관해서는, multiple-place SUBTREE 조각 중복이 조인, 유니온, 및 XML 질의 형태로의 구조화 및 태깅 비용을 줄이기 위한 매우 효율적인 방법임을 확인할 수 있었다.

탐색 방법에 관한 실험에서는 욕심쟁이 알고리즘을 이용하는 경험적 탐색 방법이 효율적임을 보여주었으며, 앞서의 중복 데이타 사용의 문제점이 XML 갱신 질의 시의 중복 데이타의 갱신 일관성 비용으로 인하여 전체 질의 비용이 커질 수 있는 것인데, 제안된 탐색 방법을 통하여 과도한 중복 데이타의 갱신 일관성 비용을 가져오는 중복 데이타들은 선택되지 않을 것이다.

향후 본 연구에서는 지금까지는 배제된 적절한 인덱스와 관련 연구에서 언급한 LegoDB의 스키마 변형 방법, 그리고 구조적 중복 방법을 함께 고려한 최적의 맵핑 스키마 탐색 문제를 살펴보고자 한다. XML 데이타에 대한 인덱싱 방법으로는 최근 고전적 B 트리와는 다른 새로운 방법들[21,22]이 다수 제안된 바 있으며, 이러한 인덱싱 방법과 구조적 중복 방법을 함께 고려하는 방법에 관하여 살펴보고자 한다.

참 고 문 헌

[1] C.C Kanne, G. Moerkotte, "Efficient Storage of XML Data," Proceedings of International Conference on DATA ENGINEERING, California, USA, p. 198, 2000.

[2] A. Deutsch, M. Fernandez, D. Suciu, "Storing semistructured data with STORED," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 431-442, 1999.

[3] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. De-Witt, J. Naughton, "Relational databases for querying XML documents: limitations and opportunities," Proceedings of VLDB, Edinburgh, UK, pp. 302-314, 1999.

[4] M. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, W. Tan, "SilkRoute: A framework for publishing relational data in XML," ACM TODS, 27(4): pp. 438-493, 2002.

[5] H. I. Kang, B. Y. Lee, J. S. Yoo, "Design and Implementation of a XML Repository System Using DBMS and IRS," The Seventh Annual Conference for XML, SGML and markup technologies, XML Asia Pacific 2000, Sydney, 2000.

[6] D. Florescu, D. Kossmann, "Storing and Querying XML document using an RDBMS," IEEE Data Engineering Bulletin, 22(3), 1999.

[7] I. Tatarinov, Z. G. Ives, A. Y. Halevy, D. S. Weld, "Updating XML," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 413-424, 2001.

[8] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, B. Reinwald, "Efficiently publishing relational data as XML documents," Proceedings of VLDB, Cairo, Egypt, pp. 65-76, 2000.

[9] Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml#dt-doctype, October 2000.

[10] XML Schema Part 0: Primer, http://www.w3.org/TR/xmlschema-0/, May 2001.

[11] P. Bohannon, J Freire, P. Roy, J. Simeon, "From XML Schema to Relations: A Cost-Based Approach to XML Storage," Proceedings of International Conference on DATA ENGINEERING, San Jose, California, pp. 64-75, 2002.

[12] J. Bosak, 셰익스피어 연극 XML 데이타, http://www.oasis-open.org/cover/ bosakShakespeare200.html

[13] A. Deutsch, V. Tannen, "MARS: A system for publishing XML from mixed and redundant storage," Proceedings of VLDB, Berlin, Germany, pp. 201-212, 2003.

[14] L. Popa, "Object/Relational Query Optimization with Chase and Backchase," PhD thesis, Univ. of Pennsylvania, 2000.

[15] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, T. Price, "Access Path Selection in a Relational Database Management System," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 23-34, 1979.

[16] A. Kemper, G. Moerkotte, "Access support in object bases," Proceedings of the ACM SIGMOD conference, pp. 364-374, 1990.

[17] E. Shekita, M. Carey, "Performance Enhancement through Replication in an Object-Oriented DBMS," Proceedings of the ACM SIGMOD conference, pp. 325-336, 1989.

[18] H. Gupta, "Selection of Views to Materialize in a Data Warehouse," ICDT 1997, pp. 98-112, 1997.

[19] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, R. Busse, "XMark: A Benchmark for XML Data Management," Proceedings of VLDB, Hong Kong, China, pp. 974-985, 2002.

[20] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, J. Siméon, "StatiX: Making XML count," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 181-191, 2002.

[21] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, M. Shadmon, "A Fast Index for Semistructured Data," Proceedings of VLDB, pp. 341-350, 2001.

[22] R. Kaushik, P. Bohannon, J. F Naughton, H. F Korth, "Covering Indexes for Branching Path Queries," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 133-144, 2002.

김 재 훈

1997년 건국대학교 전자계산학과 공학사 1999년 건국대학교 컴퓨터·정보통신공학과 공학석사. 2005년 서강대학교 컴퓨터학과 공학박사. 관심분야는 웹 데이타베이스, 시맨틱 웹, 데이타 웨어하우징 & 데이타 마이닝

박 석

1978년 서울대학교 계산통계학과(이학사). 1980년 한국과학기술원 전산학과(공학석사). 1983년 한국과학기술원 전산학과(공학박사). 1983년 9월~현재 서강대학교 컴퓨터학과 교수. 1989년~1991년 University of Virginia 방문교수. 1996년~1998년 한국정보과학회 데이타베이스 연구회 운영위원장. 1997년 2월~현재 한국통신정보보호학회 이사. 2005년 1월~현재 한국정보과학회 부회장. 2000년 4월~현재 DASFAA Steering Committee Member. 관심분야는 데이타베이스 보안 XML, 트랜잭션 관리, 데이타웨어하우스, 웹과 데이타베이스, 워크플로우