웹에서의 XML 질의 캐쉬 기법

Techniques of XML Query Caching on the Web

박대성(Daesung Park)*, 강현철(Hyunchul Kang)**

초 록

e-Commerce와 같은 응용 등에 의해 웹에서 XML 데이터의 양이 많아짐에 따라 XML 질의 처리를 신속하게 해주는 기술이 필요하게 되었다. 이를 가능하게 해주는 것이 XML 질의 캐쉬이다. 자주 제기되는 질의에 대하여 질의 결과를 캐쉬한 후, 동일 질의에 재사용함으로써 빠른 응답 시간을 보장할 수 있다. 본 논문에서는 XML 질의 결과의 포맷으로 가장 보편적으로 사용되고 있는 노드 식별자 집합(NIS)을 캐쉬하여 XML 질의 성능을 향상시키는 기법을 제안한다. 캐쉬되는 NIS는 질의 결과를 구성하는 소스 XML 엘리먼트들의 식별자 집합이다. 따라서 NIS는 질의 결과의 재구성, 변형, 다른 데이터와의 통합 등 웹 응용의 데이터 검색 요건을 충족시키기에 적절하고, XML 소스의 변경에 따른 점진적 갱신에 효율적이다. 그러나 XML 문서 형태로 질의 결과를 반환해야 할 경우에는 소스 XML 엘리먼트를 검색하는 실체화 과정을 필요로 한다. 본 논문에서는 세가지의 서로 다른 NIS의 구성을 고려하여 이들의 생성, 실체화, 점진적 갱신 알고리즘을 제안하고 RDBMS를 이용하여 구현하였다. 다양한 실험을 통한 성능 평가 결과 본 논문에서 제시하는 XML 질의 캐쉬 기법의 효율성을 확인하였다.

ABSTRACT

As data on the Web is more and more in XML due to proliferation of Web applications such as e-Commerce, it is strongly required to rapidly process XML queries. One of such techniques is XML query caching. For frequently submitted queries, their results could be cached in order to guarantee fast response for the same queries. In this paper, we propose techniques for XML query performance improvement whereby the set of node identifiers(NIS) for an XML query is cached. NIS is most commonly employed as a format of XML query result, consisting of the identifiers of the XML elements that comprise the query result. With NIS, it is suitable to meet the Web applications data retrieval requirements because reconstruction and/or modification of query results and integration of multiple query results can be efficiently done. Incremental refresh of NIS against its source updates can also be efficiently done. When the query result is requested in XML, however, materialization of NIS is needed by retrieving the source XML elements through their identifiers. In this paper, we consider three different types of NISs, proposing the algorithms of their creation, materialization, and incremental refresh. All of them were implemented using an RDBMS. Through a detailed set of performance experiments, we showed the efficiency of the proposed XML query caching techniques.

키워드 : XML, XML 질의처리, XML 질의 캐쉬, 노드 식별자 집합, 실체뷰, 점진적 캐쉬 갱신
XML, XML query processing, XML query cache, node identifier set,
materialized view, incremental cache refresh

이 논문은 2005년도 중앙대학교 학술연구비(일반연구비) 지원에 의한 것임

* 중앙대학교 대학원 컴퓨터공학과
** 중앙대학교 컴퓨터공학부 교수

1. 서 론

웹에서 데이터 표현 및 교환의 표준으로 XML이 등장한 이래 e-Commerce, 시맨틱 웹, 웹 검색과 같은 차세대 응용에서 질의의 대상이 되는 데이터가 점차 XML로 많이 표현되고 있다. 이들 응용은 웹에 산재해있는 XML 소스를 검색하여 질의 결과를 구한다. 질의 대상이 되는 XML 소스의 양이 웹의 특성 상 아주 방대할 수 있으므로 동일한 질의가 제기될 때마다 매번 XML 소스로부터 결과를 검색하는 것은 매우 비효율적이다. 따라서 자주 제기되는 XML 질의에 대해서는 그 결과를 캐쉬해 후속 질의 처리에 재사용하는 XML 질의 캐쉬 기법이 요구된다.

질의 캐쉬는 기존의 RDBMS 환경에서 많이 연구되어 실체뷰(materialized view) 기법이 제시되었다 [1]. 관계 실체뷰란 SQL문의 결과 집합(SQL result set)을 캐쉬한 것을 의미한다. 따라서 동일한 SQL문이 다시 제기되면 이미 그 결과를 캐쉬하고 있으므로 소스 테이블들의 검색을 생략할 수 있다. 그러나 소스 테이블에 변경이 발생하면 관련 실체뷰의 일관성 유지가 필요한데 변경 내용 중 실체뷰와 연관성이 있는 것만 점진적으로(incrementally) 반영하는 것이 효율적이다. 이와 같은 관계 실체뷰 기법은 90년대에 많이 연구되어 현재 대표적인 상용 RDBMS들이 제공하고 있으며, 데이터웨어하우스 응용에 중요하게 사용되어 오고 있다.

XML 질의에 대해서도 질의 캐쉬를 생각할 때 그 기본 개념 면에서는 RDBMS의 그것과 거의 유사하다 하겠다. 그러나 XML 질의 캐쉬 기법을 설계할 때는 관계 실체뷰 기법과 비교하여 캐쉬의 내용 면에서의 중요한 차이점을 고려해야 한다. 관계 실체뷰의 경우 캐쉬되는 내용은 해당 질의 SQL문의 결과 집합 그 자체로 단순하다. 그러나 XML 질의의 경우는 그 처리 결과로 캐쉬되는 내용이 무엇이어야 하는가가 관계 실체뷰의 경우처럼 단순하지 않다. 그 이유는 XML 질의 결과의 형태가 명확히 규정되어 있지 않기 때문이다. 현재 W3C에서 명시한 XML 질의어 표준의 대표적 예인 XPath와 XQuery의 두 경우 모두 그 사양에서 질의 결과의 형태를 명확히 규정하고 있지 않다. 이는 XML 질의 또는 더 일반적으로 웹 질의의 결과는 검색된 소스 데이터 항목으로부터 응용이 요구하는 형태로 다양하게 변형, 재구성, 데이터 항목들 간에 통합될 필요에 따라 유연성을 가져야 하기 때문이다. 이는 관계 질의의 결과가 테이블로 미리 정의되어 있는 것과 대조적이다.

Tatarinov et al. [2] 등 대부분의 XML 질의 처리 연구에서는 XML 질의 결과의 포맷으로 질의 조건을 만족하는 노드 식별자 집합(node identifier set, 이하 NIS)을 고려하고 있다. 여기서 노드란, XML 데이터가 보통 노드 레이블된 순서가 있는 트리(node-labelled ordered tree)로 모델링되어 각 XML 엘리먼트가 트리의 노드로 표현되므로 결국 질의의 조건을 만족하는 XML 엘리먼트를 의미한다. XML 질의를 제기한 응용에게 NIS를 그 결과로 넘겨주면 응용은 그것으로부터 융통성 있게 필요한 후속 처리를 수행할 수 있다. 예를 들어, 서로 관련된 여러 질의들의 결과 NIS들 간에 교집합을 구한 후 그 결과 식별

자들을 통해 해당 XML 문서 조각(fragment)을 검색, 적절히 통합한 후 구조화한 XML 문서를 생성하는 경우를 생각할 수 있다.

NIS를 XML 질의 결과로 캐쉬할 경우, 질의 조건을 만족하는 XML 엘리먼트를 XML 소스로부터 검색하여 XML로 표현하는 과정을 관계 실체뷰란 용어에 따라 실체화(materialization)라 부르고 그 결과를 XML 실체뷰라고 부르자. 만약 XML 질의의 최종 결과로서 단순히 XML 실체뷰가 요구되는 경우라면, 질의 캐싱을 통해 응답 시간을 줄이는 면에서는 NIS 보다는 XML 실체뷰를 캐쉬하는 것이 더 효율적일 것이다. 그러나 XML 실체뷰는 XML 소스의 변경에 따른 캐쉬의 점진적 갱신 면에서 NIS에 비해 비효율적이다. 실체뷰로 캐쉬될 경우 그것의 점진적 갱신을 위해 복잡하고 많은 양의 실체뷰와 소스 간 사상(mapping) 정보를 유지해야 하기 때문이다 Quan et al. [3], Chen and Rundensteiner [4]. 또한 NIS는 XML 실체뷰에 비해 그 용량이 현저히 작다. 이는 웹 응용과 같이 일반적으로 불특정 다수의 클라이언트에 의해 상당히 많은 수의 질의 처리가 요구되는 환경에서 실체뷰의 경우에 비해 훨씬 많은 수의 질의 캐쉬를 유지하는 것이 가능하다는 것을 의미하므로 질의 처리 성능 향상 및 그에 따른 웹 응용의 확장성(scalability) 면에서 결정적으로 유리하다.

이상의 이유로 본 논문에서는 XML 질의 캐쉬의 내용으로 NIS를 고려한다. 본 논문에서는 XML 질의 캐쉬 기법의 요소 기술로서, 캐쉬 대상 XML 질의의 NIS를 생성하는 알고리즘, NIS의 실체화 알고리즘, XML 소스의 변경과 NIS 간의 연관성 검사 알고리즘, 그리고 NIS의 점진적 갱신 알고리즘을 제안한다. 제안한 알고리즘들은 XML 소스 및 NIS의 저장소로 RDBMS를 사용하여 구현하고 다양한 실험을 통해 그 성능을 평가하여 그 효율성을 검증하였다.

본 논문의 구성은 다음과 같다. 2절에서는 본 논문과 관련된 기존 연구들을 살펴본다. 3절에서는 세가지의 서로 다른 NIS 구성을 제시하고, 그들의 생성 및 실체화 알고리즘을 기술한다. 4절에서는 하부 XML 소스의 변경에 따른 NIS의 점진적 갱신 알고리즘을 기술한다. 5절에서는 제시한 XML 질의 캐쉬 기법의 구현 및 성능 평가 결과를 기술한다. 마지막으로 6절에서 결론을 맺고 향후 연구 과제를 기술한다.

2. 관련 연구

Quan et al. [3], Vidal et al. [5], Dimitrova et al. [6]에서는 XML 실체뷰 기법을 연구하였다. Quan et al. [3]에서는 Argos라는 XQL 질의에 대한 캐싱 시스템을 제시하였다. Argos에서는 APIX(Aggregate Path IndeX)[4]라는 인덱스 구조를 이용하여 실체뷰를 점진적으로 갱신한다. APIX는 질의의 구조 패턴과 그 패턴을 만족하는 객체의 모음이다. Argos는 PDOM(Persistent DOM)[7] 형태로 저장된 XML 소스를 대상으로 질의를 처리한다. 원격 PDOM들을 지역 시스템으로 가져와서 XQL 질의 엔진을 사용하여 질의를 처리하여 실체뷰를 구성한다. 이때 실체뷰의

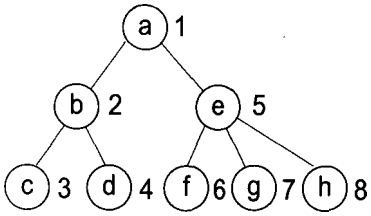
유지 관리를 위하여 APIX라는 보조 인덱스 구조를 생성한다. APIX를 이용하여 소스에 일어난 변경 중 (1) 수정된 값이 해당 조건 만족 여부에 영향을 주지 않는 경우, (2) 질의와 상관없는 객체에 대한 수정인 경우, (3) 삭제된 객체 또는 그 부모 객체가 APIX에 없는 경우, (4) 값이 수정된 객체의 부모 객체가 APIX에 없는 경우 등을 뷰에 영향을 주지 않는 변경 연산으로 자가 진단할 수 있다.

Vidal et al. [5]에서는 관계형 데이터에 대해 정의된 XML 뷰를 실체화하여 점진적 갱신 기법에 의해 유지하는 문제에 대해 연구하였다. XML 실체뷰를 점진적으로 갱신하기 위하여 관계형 데이터 소스에 event-condition-rules를 지정한다. 규칙(rule)의 생성을 위하여 우선 하부 데이터 소스의 스키마를 XML 스키마로 변환한다. 변환된 하부 데이터 소스의 XML 스키마와 뷰 스키마의 매칭에 의해 대응 가정(correspondence assertions)을 생성한다. 대응 가정은 하부 데이터 소스의 변경 연산이 유지해야 하는 특별한 타입의 제약 조건으로 다루어진다. 생성된 대응 가정에 점진적 갱신 규칙을 정의함으로써 변경 연산에 대해 XML 실체뷰의 일관성을 유지하도록 한다.

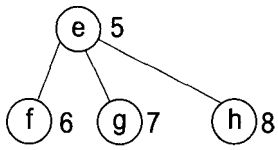
Dimitrova et al. [6]에서는 VOX(View Maintenance for Ordered XML)라고 하는 XML 순서가 유지되는 XQuery 실체뷰의 점진적 갱신을 위한 대수 접근 방법(algebraic approach)을 연구하여 XML 순서를 고려한 XML 실체뷰 유지 기법을 제시하였다. VOX는 Rainbow[8]라는 XML 데이터 관리 시스템 상에 구현되었다. VOX에서는 XAT[9]라고 하는 XML Algebra를 이용하였다. VOX에서는 실체뷰에서의 순서 유지를 위해 FLEX Keys[10]를 XML 노드 식별자로 이용하는 하부 XML 소스를 가정하였다. VOX에서는 XAT의 각 연산자와 변경 연산 타입에 대하여 변경 전파 규칙(update propagation rules)을 정의함으로써 XML 실체뷰를 점진적으로 갱신한다.

이들 XML 실체뷰 기법들은 모두 관계 실체뷰 기법의 개념을 바탕으로 XML 실체뷰 유지 문제에 접근한 것들로서 초보적인 결과들을 제시하고 있다. 이들 기법의 문제점은 다음과 같다. 첫째, XML 질의 나아가 웹 질의 처리에서는 일반적으로 질의 결과 구성에 유연성을 가져야 하는데 그런 점에서 스냅샷(snapshot)으로 제공되는 실체뷰가 아닌 경우에는 캐쉬된 질의 결과의 활용성이 저하되는 문제점을 갖고 있다. 둘째, 하부 소스의 변경을 점진적으로 반영하는 데 사용하고자 유지해야 하는 보조 정보의 양이 너무 크다. 이들 보조 정보의 양은 캐쉬된 실체뷰 각각마다 소스의 데이터 양에 비례하므로 웹 상의 방대한 소스에 대한 실체뷰를 유지하는 부담이 너무 커진다. 셋째, 실체뷰 및 하부 소스의 저장 구조 문제를 구체적으로 다루지 않고 소스의 변경을 실체뷰로 반영하는 기법에만 중점을 두고 있다. XML 실체뷰도 XML 데이터이므로 그 소스와 더불어 어떤 저장 구조를 갖느냐 하는 문제는 실체뷰 유지 기법의 성능에 밀접한 영향을 미친다.

본 논문에서는 관계 실체뷰에 단순 대응되는 XML 실체뷰 기법 대신 NIS를 캐쉬하는 기법을 제시한다. 또한 XML 소스와 NIS의 저장에 RDBMS를 이용한다. NIS는 XML 질



〈그림 1〉 XML 트리



〈그림 2〉 XML 질의 Q=/a/e 의 결과 트리

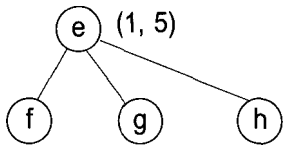
의 결과의 포맷으로 현재 가장 적절한 것으로 인식되고 있고, 해당 실체뷰에 비해 용량이 훨씬 작으므로 캐쉬할 수 있는 질의의 수를 현저히 늘려 웹 응용의 확장성(scalability)을 제고할 수 있다. 한편, RDBMS 기반의 XML 저장 체계는 그 실용성 면에서 현재 가장 널리 사용되는 기법이므로 본 논문의 XML 질의 캐쉬 기법의 실용성을 담보할 수 있다.

3. NIS 캐싱

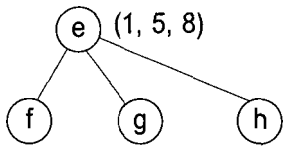
본 절에서는 NIS를 RDBMS에 유지할 수 있는 세 가지 NIS 구성 기법을 제안하고 그들의 생성, 실체화 방법을 기술한다. 본 논문에서는 XML 소스를 저장하는 하부 시스템으로 RDBMS를 사용한다고 가정하고, XML 질의로는 XQuery와 XPath 등 XML 질의어의 핵심 기능인 경로 표현식을 대상으로 한다. 본 논문에서 고려한 경로 표현식은 W3C의 XPath 표준에서 명시한 axis 중 실용적인 질의 표현에 가장 많이 사용되는 양대 axis 인 자식 및 후손(descendant-or-self) axis와 필터 연산(predicate)으로 구성된다.

3.1 NIS의 구성 및 생성

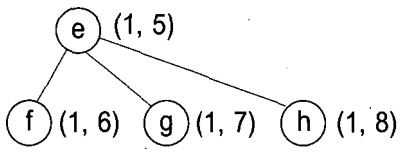
본 절에서는 세 가지 형태의 NIS 구성 기법을 제안한다. 〈그림 1〉은 XML 문서를 트리 구조로 나타낸 것으로서 문서 식별자(Document ID, 이하 DID)값은 1이며, 각 노드 옆에 명시된 노드 식별자 즉, XML 엘리먼트의 식별자(Element ID, 이하 EID)는 XML 트리의 preorder 탐색 순서로 할당되었다. 〈그림 2〉는 XML 질의 Q=/a/e의 결과 XML 트리이다.



〈그림 3〉 TE/DE의 NIS 구성



〈그림 4〉 TE/DER의 NIS 구성



〈그림 5〉 AE/DE의 NIS 구성

3.1.1 TE/DE(Target Element's DID&EID) 기법

기법 TE/DE의 NIS 구성은 질의의 목적(target) 엘리먼트 (즉, Q의 경우 그림 1의 e 노드)의 DID와 EID 쌍의 집합으로 이루어진다. 즉, 캐쉬되는 NIS는 {(x,y)|x = DID, y = Target Element's EID}이다. 따라서 예를 들어 〈그림 1〉과 같은 XML 문서에 대하여 질의 Q가 제기될 경우, 캐쉬되는 NIS는 Q의 목적 엘리먼트인 e 엘리먼트의 DID와 EID 쌍인 (1,5)로 구성된 집합 {(1,5)}이다. 〈그림 3〉은 이를 도식화한 것으로 e 노드 옆에 표시된 (1,5)만 NIS에 포함되고 e의 하위 엘리먼트 노드인 f, g, h에 대해서는 NIS에 저장하지 않는다 (이들 노드를 점선으로 표시).

3.1.2 TE/DER(Target Element's DID&EID&RmdEID) 기법

기법 TE/DER의 NIS 구성은 질의의 목적 엘리먼트의 DID와 EID 외에 그것의 가장 오른쪽 후손 엘리먼트의 식별자(Rightmost Descendant Element ID, 이하 RmdEID)의 세 값으로 된 3-tuple의 집합으로 이루어진다. 즉, 캐쉬되는 NIS는 {(x,y,z)|x = DID, y = Target Element's EID, z = Target Element's RmdEID}이다. 〈그림 1〉과 같은 XML 문서에 대하여 질의 Q가 제기될 경우, e의 RmdEID는 8이다. 왜냐하면 e의 가장 오른쪽 후손 엘리먼트는 h이고 h의 EID가 8이기 때문이다. 따라서, 기법 TE/DER에서 캐쉬되는 NIS는 집합 {(1,5,8)}이다. 〈그림 4〉는 이를 도식화한 것이다.

3.1.3 AE/DE(All Element's DID&EID) 기법

기법 AE/DE의 NIS 구성은 질의의 목적 엘리먼트 및 그것의 모든 후손 엘리먼트들의 DID와 EID 값 쌍의 집합으로 이루어진다. 즉, 즉, 캐쉬되는 NIS는 {(x,y)|x = DID, y = Target or its Descendant Element's EID} 이다. 〈그림 1〉과 같은 XML 문서에 대하여 질의 Q가 제기될 경우, 기법 AE/DE에서 캐쉬되는 NIS는 집합 {(1,5), (1,6), (1,7), (1,8)} 이다. 〈그림 5〉는 이를 도식화한 것으로 〈그림 3〉 및 〈그림 4〉와는 달리 노드 e의 후손 엘리먼트 노드들이 모두 실선으로 표시되어 있다. 〈표 1〉은 이상의 세 기법을 〈그림 1〉의 XML 트리와 질의 Q = /a/e에 대하여 비교한 것이다.

3.2 NIS의 생성

본 절에서는 전절에서 제안한 각 기법에서 NIS의 생성 방법을 기술한다. 〈그림 1〉과 같은 소스 XML 트리는 노드 단위로 관계 튜플로 사상되어 관계 테이블에 저장된다. 이때 구조 조인(structural join)의 지원 및 문서 내 엘리먼트 간 순서 표현을 위하여 XML 번호 체계(numbering scheme)가 적용되는데 본 논문에서는 Zhang et al. [11] 등의 연구를 비롯하여 가장 널리 채택되고 있는 범위 번호 체계(range numbering scheme)를 사용한다. 따라서 XML 트리의 각 노드에 할당되는 번호는 (DID, EID, RmdEID, Level)의 네가지이며 이들은 해당 노드를 표현하는 관계 튜플의 컬럼 값으로 저장된다. 여기서 Level은 트리의 레벨을 나타낸다.

각 기법 별로 NIS를 생성하여 관계 테이블에 저장하는 과정은 다음과 같다. TE/DE 기법에서는 XML 질의 처리 엔진이 주어진 질의의 목적 엘리먼트에 대응되는 관계 튜플들을 검색해 내면 각 튜플 별로 DID 및 EID 컬럼 값을 취한다. TE/DER 기법에서는 앞의 두 컬럼 외에 RmdEID 컬럼 값까지 취한다. AE/DE 기법에서는 먼저 TE/DE 기법에서와 같이 목적 엘리먼트의 DID 및 EID 컬럼 값을 취하여 그것으로 NIS를 초기화한 후 다음을 수행한다. 각 목적 엘리먼트 p에 대해 3-tuple (DID(p), EID(p), RmdEID(p))을 구하여 그로부터 p의 모든 후손 엘리먼트 q를 표현한 튜플들을 구한다. 이때 q의 튜플이 만족해야 하는 조건은 DID(q) = DID(p) 와 EID(p) 〈= EID(q) 〈= RmdEID(q) 〈= RmdEID(p) 이다 [11]. q의 튜플들로부터 DID 및 EID 컬럼 값을 취하여 NIS에 추가한다. 각 기법 별로 NIS는 하나의 관계 테이

〈표 1〉 NIS 구성의 예(그림 1의 XML 트리와 질의 Q=/a/e)

NIS 구성 기법	NIS
TE/DE	{(1, 5)}
TE/DER	{(1, 5, 8)}
AE/DE	{(1, 5), (1, 6), (1, 7), (1, 8)}

블에 저장된다.

3.3 NIS의 실체화

본 절에서는 3.1절에서 기술한 세 가지 NIS 구성 기법 별로 〈그림 2〉의 질의 Q를 예로 들어 NIS의 실체화 방법을 설명한다.

- *TE/DE 기법*
  기법 TE/DE의 NIS는 목적 엘리먼트의 (DID,EID) 쌍의 집합으로 이루어진다. 실체화를 위해 우선, XML 소스에서 목적 엘리먼트의 DID와 EID가 같은 엘리먼트 (즉, Q의 경우 그림 1의 e)를 검색한다. 둘째로 찾아낸 엘리먼트의 RmdEID (즉, 8)를 알아낸다. 끝으로 XML 소스에서 목적 엘리먼트의 DID (즉, 1)와 같으며, EID가 목적 엘리먼트의 EID (즉, 5)와 RmdEID (즉, 8) 그리고 그 사이에 존재하는 엘리먼트 (즉, e,f,g,h)들을 검색한다.

- *TE/DER 기법*
  기법 TE/DER의 NIS는 목적 엘리먼트의 (DID,EID,RmdEID) 세 값의 집합으로 이루어진다. 기법 TE/DER는 기법 TE/DE와는 달리 RmdEID 정보를 저장해두기 때문에 실체화 과정에서 RmdEID를 검색하는 비용이 들지 않는다. 따라서 실체화를 위해서는 XML 소스에서 목적 엘리먼트의 DID (즉, 1)와 같으며, EID가 목적 엘리먼트의 EID (즉, 5)와 RmdEID (즉, 8) 그리고 그 사이에 존재하는 엘리먼트 (즉, e,f,g,h)들을 검색한다.

- *AE/DE 기법*
  기법 AE/DE의 NIS는 목적 엘리먼트 및 그것의 모든 후손 엘리먼트들의 (DID,EID) 값 쌍의 집합으로 이루어진다. 따라서 실체화를 위해서는 XML 소스로부터 DID (즉, 1)와 EID (즉, 5,6,7,8) 값 쌍이 NIS에 존재하는 모든 엘리먼트 (즉, e,f,g,h)들을 검색한다.

XML 소스가 RDBMS에 저장되어 있으므로 상기 기법 별로 검색된 결과는 관계 튜플들이다. 따라서 이들을 DID, EID 값으로 정렬한 후 스택을 이용하여 XML로 복원한다.

4. NIS의 효율적인 점진적 갱신

본 절에서는 XML 소스가 변경될 때 변경된 내용을 NIS에 점진적으로 반영하여 NIS를 갱신하는 기법을 다룬다. NIS에 소스의 변경 내용을 반영할 때는 변경 내용 중 질의와 관련 있는 것만 대상으로 하는 것이 효율적이다. 본 절의 구성은 다음과 같다. 4.1절에서 XML 소스의 변경 내용을 표현하는 변경 정보에 대해 살펴본 후 4.2절에서 NIS에 영향을 미치는 XML 소스의 변경을 판별해내는 연관성 검사 알고리즘을 기술한다. 끝으로 4.3절에서 연관된 변경을 NIS에 반영하는 방법을 기술한다.

〈표 2〉 변경 정보 구조

변경정보				
LSN	Type	TargetPath	Value	ChangedElement

ChangedElement			
DID	EID	RmdEid	ParentEid

4.1 변경 정보

본 논문에서는 XML 소스의 변경으로 단말(leaf) XML 엘리먼트의 삽입, 삭제, 수정을 고려한다. 이 세가지 변경으로는 임의의 복잡한 XML 변경을 모두 표현할 수 있다 [12]. 소스 XML 데이터에 변경이 발생하면 캐쉬된 NIS의 일관성 유지를 위해 하부 XML 저장 시스템은 어떤 변경 연산이 일어났었는지에 대한 정보를 남긴다. 변경 정보의 구조는 〈표 2〉와 같다.

LSN(Log Sequence Number)은 각 변경 정보의 식별자이다. 소스의 변경이 캐쉬된 NIS에 즉각 반영되지 않고 변경 로그에 기록되었다가 특정 NIS의 갱신이 요구될 때 일괄 반영될 경우 〈표 2〉의 변경 정보는 변경 로그 레코드로서 변경 로그에 시간 순으로 기록된다. 이때 LSN은 변경 로그 관리를 위해 필요하며 시스템 타임스탬프 값을 부여함으로써 변경 로그 레코드의 생성 순서를 표현할 수 있다. Type은 변경의 종류를 나타낸다. Type의 종류로 INSERT, DELETE, MODIFY, CHANGE_RmdEid가 있다. INSERT, DELETE, MODIFY는 단말 엘리먼트가 삽입, 삭제, 변경된 경우를, CHANGE_RmdEid는 단말 엘리먼트의 삽입으로 인하여 그것의 조상 엘리먼트의 RmdEid가 변경되는 경우를 나타낸다. TargetPath는 변경이 일어난 엘리먼트의 경로에서 필터 연산을 제외한 경로를 의미한다. 만약 경로표현식 '/a/b/c[d = "x"]'의 목적 엘리먼트가 변경되었다면 TargetPath는 '/a/b/c'이다. Value는 단말 엘리먼트의 텍스트 (즉, PCDATA)을 변경할 때 변경된 내용을 나타낸다. ChangedElement는 변경 연산에 영향을 받는 엘리먼트로서 그것의 해당 DID, EID, RmdEid, ParentEid 값으로 이루어진다.

- *INSERT Type의 예*

  경로 '/a/b/c'의 하위에 'x'라는 텍스트를 가지는 'd' 엘리먼트(〈d〉x〈/d〉)를 추가할 경우 〈표 3〉과 같은 변경 정보가 생성된다. 이 경우 변경에 영향을 받은 엘리먼트의 TargetPath는 '/a/b/c' 경로에 'd'가 합쳐진 '/a/b/c/d'가 된다. ChangedElement에서는 DID가 1이고 EID와 RmdEid가 5이며 ParentEid는 4라고 가정하였다

〈표 3〉 변경 정보 예 1

LSN	Type	TargetPath	Value	ChangedElement
1	INSERT	/a/b/c/d	x	(1, 5, 5, 4)

〈표 4〉 변경 정보 예 2

LSN	Type	TargetPath	Value	ChangedElement
1	DELETE	/a/b/c	null	(1, 5, 5, 4)

〈표 5〉 변경 정보 예 3

LSN	Type	TargetPath	Value	ChangedElement
1	MODIFY	/a/b/c	y	(1, 5, 5, 4)

〈표 6〉 변경 정보 예 4

LSN	Type	TargetPath	Value	ChangedElement
1	CHANGE_RmdEid	/a/b/c	null	(1, 5, 6, 4)

• *DELETE Type의 예*

'/a/b/c[d="x"]'의 경로에 있는 엘리먼트를 삭제할 경우 〈표 4〉와 같은 변경 정보가 생성된다. ChangedElement에서는 DID가 1이고 EID와 RmdEid가 5이며 ParentEid는 4라고 가정하였다.

• *MODIFY Type의 예*

'/a/b/c[d="x"]'의 경로에 있는 엘리먼트의 내용을 "y"로 변경할 경우 〈표 5〉와 같은 변경 정보가 생성된다. ChangedElement에서는 DID가 1이고 EID와 RmdEid가 5이며 ParentEid는 4라고 가정하였다.

• *CHANGE_RmdEid Type의 예*

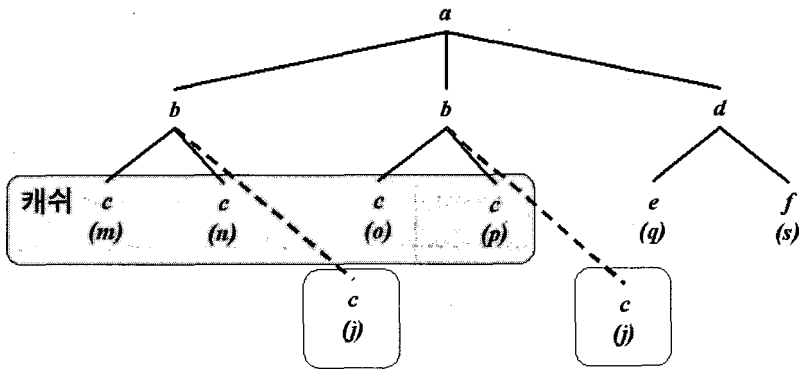
경로 '/a/b/c'에 있는 엘리먼트의 RmdEid 값이 5에서 6으로 변경되었을 경우 〈표 6〉과 같은 변경 정보가 생성된다. ChangedElement에서는 DID가 1이고 EID와 5이며 ParentEid는 4라고 가정하였다.

4.2 연관성 검사

XML 소스의 변경으로 인해 캐쉬된 NIS의

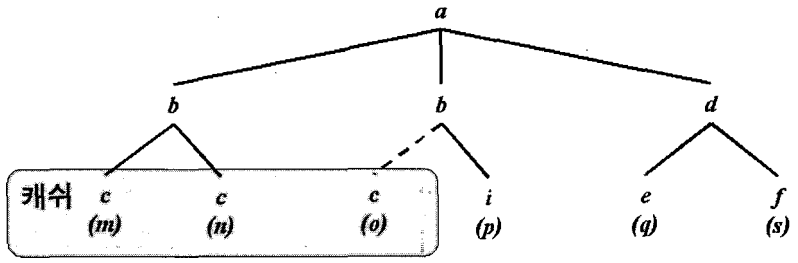
〈표 7〉 TE/DE 기법의 연관성 유형

유형	설명
INSERT_T	질의의 목적 엘리먼트가 삽입되었을 경우
DELETE_T	질의의 목적 엘리먼트가 삭제되었을 경우



• 캐쉬 : /a/b/c
• 삽입 변경 : /a/b 아래에 〈c〉j〈/c〉 삽입

〈그림 6〉 INSERT_T 유형



• 캐쉬 : /a/b/c
• 삭제 변경 : /a/b[i = "p"]/c 삭제

〈그림 7〉 DELETE_T 유형

일관성이 파괴될 경우 NIS의 갱신도 필요하다. 그러나 XML 소스의 변경 내용이 항상 캐쉬된 관련 NIS에 영향을 미치는 것은 아니다. XML 소스의 변경이 NIS에 어떤 영향을 미치는지 여부를 판단하는 것이 연관성 검사이다. 이 검사를 통해 연관성이 있는 변경만 NIS에 반영하는 것이 효율적이다.

4.2.1 연관성 유형

본 절에서는 세 가지 NIS 구성 기법 별로 소스의 변경과 NIS 간의 연관성 유형을 살펴본다.

- *TE/DE 기법*

  〈표 7〉은 TE/DE기법의 NIS에 영향을 미치는 XML 소스의 변경에 대한 연관성 유형을 정리한 것이다. XML 소스의 변경 유형 중 Modify 유형이 NIS에 영향을 미치지 않는 이유는 TE/DE기법의 NIS에는 엘리먼트의 텍스트 (즉, PCDATA)를 유지하지 않기 때문에 텍스트에 대한 변경은 NIS에 아무런 영향을 미치지 않기 때문이다.

〈그림 6〉은 질의 '/a/b/c'의 NIS가 캐쉬되어 있을 때, 경로 '/a/b' 아래 'j'라는 텍스트를 가진 엘리먼트 c(〈c〉j〈/c〉)를 삽입하는 경우를 나타낸 것이다. 이는 질의의 목적 엘리먼트가 삽입되는 경우이므로 NIS에도 새롭게 삽입된 엘리먼트의 내용을 추가해주어야 한다. 이러한 연관성 유형이 INSERT_T 유형이다.

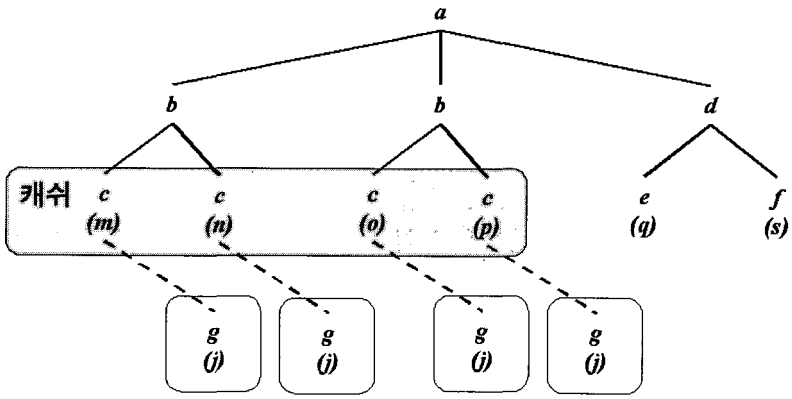
〈그림 7〉은 질의 '/a/b/c'의 NIS가 캐쉬되어 있을 때, '/a/b[i="p"]/c'를 만족하는 엘리먼트를 삭제하는 경우를 나타낸 것이다. 이는 질의의 목적 엘리먼트가 삭제되는 경우이므로 NIS에 존재하던 삭제된 엘리먼트의 정보 역시 삭제해야 한다. 이러한 연관성 유형이 DELETE_T 유형이다.

- *TE/DER 기법*

  TE/DER 기법에서는 〈표 8〉과 같이 세 가지 연관성 유형이 존재한다. 이 중 INSERT_T 유형과 DELETE_T 유형은 TE/DE 기법의 그들과 동일하다. 그러나 TE/DE 기법과는 달리 CHANGE_RmdEid 유형이 추가로 존재한다. TE/DER 기법은 NIS에 목적 엘리먼트의 RmdEid를 가지고 있기 때문에 이 값이 XML 소스에서 변경되면 NIS에서도 변경시켜야하기 때문이다. 이러한 현상이 일어나는 경우는 목적 엘리먼트의 하위에서 새로운 엘리먼트가 삽입될 때이다. TE/DER 기법 역시 TE/DE 기법과 마찬가지로 NIS에 엘리먼트의 텍스트 (즉, PCDATA)를 유지하지 않기 때문에 엘리먼트의 텍스트 내용을 변경하는 Modify 경우는 고려하지 않아도 된다.

〈표 8〉 TE/DER 기법의 연관성 유형

유형	설명
INSERT_T	질의의 목적 엘리먼트가 삽입되었을 경우
DELETE_T	질의의 목적 엘리먼트가 삭제되었을 경우
CHANGE_RMDEid	새로운 엘리먼트가 삽입되어 NIS에 포함된 엘리먼트의 RmdEid 값이 바뀌었을 경우



• 캐쉬 : /a/b/c
• 삽입 변경 : /a/b/c 아래에 〈g〉j〈/c〉 삽입

〈그림 8〉 CHANGE_RmdEid 유형

〈그림 8〉은 질의 '/a/b/c'의 NIS가 캐쉬되어 있을 때, 경로 '/a/b/c' 아래 'j'라는 텍스트를 가진 엘리먼트 g(〈g〉j〈/g〉)를 삽입하는 경우를 나타낸 것이다. 이는 질의의 목적 엘리먼트가 삽입되는 경우가 아니라 목적 엘리먼트의 하부에 새로운 엘리먼트가 삽입되어 XML 소스 상의 'c' 엘리먼트의 RmdEid 값이 바뀌어 NIS의 RmdEid 값도 변경해야 하는 경우다. 이러한 연관성 유형이 CHANGE_RmdEid 유형이다.

• *AE/DE 기법*

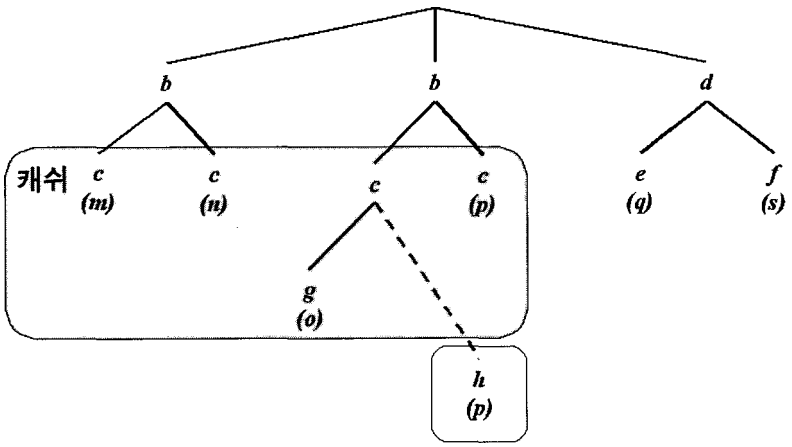
AE/DE 기법에서는 〈표 9〉와 같이 네 가지의 연관성 유형이 존재한다. TE/DE 기법 및 TE/DER 기법과는 다르게 질의의 목적 엘리먼트의 하위에 새로운 엘리먼트가 삽입/삭제되는 경우에도 NIS를 변경시켜야 한다. 〈그림 9〉와 〈그림 10〉에서 뷰 정의를 만족하는 목적 엘리먼트의 하위에 존재하는 엘리먼트가 삽입/삭제되어 NIS에 영향을 미치는 경우를 예를 들어 설명하겠다.

〈그림 9〉는 질의 '/a/b/c'의 NIS가 캐쉬되어 있을 때, 경로 '/a/b/c[g="o"]' 아래 'p'라는 텍스트를 가진 엘리먼트 h(〈h〉p〈/h〉)가 삽입되는 경우를 나타낸 것이다. AE/DE 기법의 경우 질의의 목적 엘리먼트와 그 하위에 존재하는 엘리먼트 모두의 식별자를 NIS에 유지하므로 〈그림 9〉에서와 같이 목적 엘리먼트 하위에 새로운 엘리먼트가 삽입되면 그 내용을 NIS에도 추가시켜주어야 한다. 이러한 연관성 유형이 INSERT_E 유형이다.

〈그림 10〉은 질의 '/a/b/c'의 NIS가 캐쉬되어 있을 때, 경로 '/a/b/c[g="o"]/h'에 해당하는 엘리먼트 h(〈h〉p〈/h〉)를 삭제하는 경우를 나타낸 것이다. AE/DE 기법의 경우 질의의 목적 엘리먼트와 그 하위에 존재하는 엘리먼트 모두의 식별자를 NIS에 유지하므로 〈그

〈표 9〉 AE/DE 기법의 연관성 유형

유형	설명
INSERT_T	질의의 목적 엘리먼트가 삽입되었을 경우
DELETE_T	질의의 목적 엘리먼트가 삭제되었을 경우
INSERT_E	질의의 목적 엘리먼트 하위에 새로운 엘리먼트가 삽입된 경우
DELETE_E	질의의 목적 엘리먼트 하위에 존재하던 엘리먼트가 삭제된 경우



• 캐쉬 : /a/b/c
• 삽입 캐쉬 : /a/b/c[g = "o"]아래에 〈h〉p〈/h〉 삽입

〈그림 9〉 INSERT_E 유형

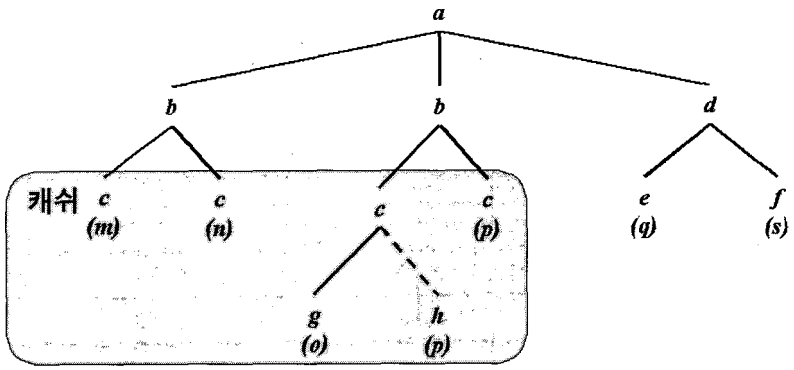
림 10〉에서와 같이 목적 엘리먼트 하위에서 어떤 엘리먼트가 삭제되면 그 내용을 NIS에도 반영시켜주어야 한다. 이러한 연관성 유형이 DELETE_E 유형이다.

4.2.2 연관성 검사 알고리즘

TE/DE 기법, TE/DER 기법, AE/DE 기법은 각기 다른 NIS 구성을 가지기 때문에 연관성 검사 알고리즘도 조금씩 다른 구조로 작동한다. 각 기법은 변경 연산 시 생성되는 변경 정보와 질의의 정의를 비교하여 연관성 검사를 수행한다. 본 절에서는 각 기법 별로 변경 정보의 분석을 통한 연관성 검사 알고리즘을 기술한다.

• *TE/DE 기법*

1. 변경 정보의 TargetPath가 질의 정의와 일치하는지 검사한다. 일치하지 않는다면 NIS에 변경 내용을 반영할 필요가 없으므로 변경 정보를 무시한다.
2. 변경 정보의 TargetPath가 질의 정의와 일치한다면 변경 정보의 Type값을 본다.



• 캐쉬 : /a/b/c
• 삭제 변경 : /a/b/c[g = "o"]/h 삭제

〈그림 10〉 DELETE_E 유형

Type이 INSERT라면 연관성 유형은 INSERT_T가 되며 Type이 DELETE라면 연관성 유형은 DELETE_T가 된다. 이외의 경우는 NIS와 상관없는 변경 정보이므로 무시한다.

• *TE/DER 기법*

1. 변경 정보의 TargetPath가 질의 정의와 일치하는지 검사한다. 일치하지 않는다면 NIS에 변경 내용을 반영할 필요가 없으므로 변경 정보를 무시한다.
2. 변경 정보의 TargetPath가 질의 정의와 일치한다면 변경 정보의 Type값을 본다. Type이 INSERT라면 연관성 유형은 INSERT_T가 되며 Type이 DELETE라면 연관성 유형은 DELETE_T가 된다. 만약 Type이 CHANGE_RmdEid 라면 연관성 유형 또한 CHANGE_RmdEid가 된다. 이외의 경우는 NIS와 상관없는 변경 정보이므로 무시한다.

• *AE/DE 기법*

1. 변경 정보의 TargetPath가 질의 정의와 일치하거나 질의 정의에 포함되는지 검사한다. 즉, 질의 정의 상의 목적 엘리먼트의 변경이거나 목적 엘리먼트의 하위에 존재하는 엘리먼트의 변경인지 검사한다. 일치하지 않거나 포함되지 않으면 NIS에 변경 내용을 반영할 필요가 없으므로 변경 정보를 무시한다.
2. 변경 정보의 TargetPath가 질의 정의와 일치한다면 변경 정보의 Type값을 본다. Type이 INSERT라면 연관성 유형은 INSERT_T가 되며 Type이 DELETE라면 연관성 유형은 DELETE_T가 된다.
3. 변경 정보의 TargetPath가 질의 정의 상의 목적 엘리먼트의 하위에 이르는 것이면 변경 정보의 Type을 본다. Type이

INSERT라면 연관성 유형은 INSERT_E가 되며 DELETE라면 연관성 유형은 DELETE_E가 된다. 이외의 경우는 NIS와 상관없는 변경 정보이므로 무시한다.

4.3 NIS로의 변경 반영

XML 소스의 변경 중 캐쉬된 NIS와 연관성이 있는 것을 NIS에 반영하는 방법은 간단하다. 각 연관성 유형별로 그 방법의 개요는 아래와 같다.

- INSERT_T 유형 : 삽입된 엘리먼트에 대한 식별자 값을 변경 정보로부터 구하여 NIS에 추가한다. 예를 들어, 기법 TE/DE의 경우, 변경 정보로부터 (DID, EID) 값의 쌍을 구하여 NIS에 추가한다.
- DELETE_T 유형 : NIS에서 해당 엘리먼트 관련 모든 정보를 삭제한다.
- CHANGE_RmdEid 유형 : NIS에서 해당 엘리먼트를 찾아 RmdEid 값을 변경 정보의 것으로 수정한다.
- NSERT_E 유형 : 삽입된 엘리먼트에 해당되는 정보를 NIS에 추가한다.
- DELETE_E 유형 : 해당 엘리먼트 관련 모든 정보를 NIS에서 삭제한다.

5. 구현 및 성능 평가

본 절에서는 본 논문에서 제안한 NIS 캐쉬 기법들을 구현하여 성능을 평가한 결과를 기술한다. 5.1절에서는 실험 환경 및 실험에 사용된 XML 소스 데이터에 대해 기술하고, 5.2절에서는 성능 척도와 실험에 사용된 XML 질의에 대해 기술하며, 5.3절에서는 실험 결과를 기술한다.

5.1 실험 환경 및 데이터

실험의 플랫폼은 Windows 2000 Server이며 XML 소스 및 NIS의 저장 시스템으로 사용된 RDBMS는 Oracle 9i이다. 실험에 사용된 서버의 CPU는 Pentium IV 1.3GHz 듀얼 CPU이며 메모리는 2,304MB이다. 실험에 사용된 XML 소스 데이터는 세익스피어 희곡 XML 문서다. 〈표 10〉은 이상의 내용을 정리한 표이다.

실험에 사용한 세익스피어 희곡 XML 문서는 총 37개의 문서로 이루어진 것으로서 총

〈표 10〉 실험 환경 및 데이터

항목		내용
OS		Windows 2000 Server
DBMS		Oracle 9i
시스템	CPU	Pentium IV 1.3Ghz * 2
	RAM	2304MB
실험데이터		세익스피어 희곡 XML 문서 (표 11 참조)

〈표 11〉 실험 데이터 (세익스피어 희곡 XML 문서) 상세정보

Number of documents	37
Total size	7.65
Average size/document (KB)	206.71
Number of element node	179,689
Number of attribute node	0
Number of text node	147,442
Number of simple node	57

〈표 12〉 실험에 사용된 질의 정의

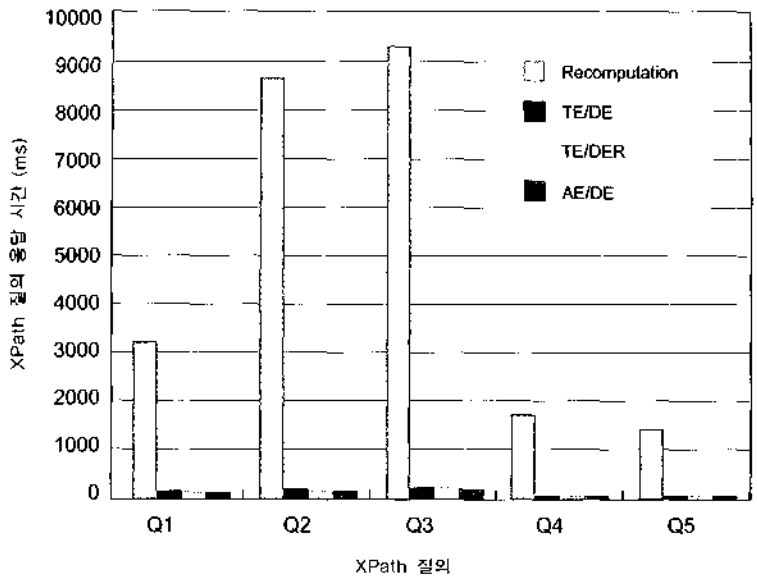
질의 정의
Q1 /PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR
Q2 //SCENE/TITLE
Q3 //ACT//TITLE
Q4 /PLAY/ACT/SCENE/SPEECH[SPEAKER = 'CURIO']
Q5 /PLAY/ACT/SCENE/[//SPEAKER = 'Steward']/TITLE

크기는 7.65MB이다. 한 문서 당 평균 크기는 206.71KB이다. 엘리먼트의 수는 179,689이며 속성은 존재하지 않는다. 텍스트를 갖는 단말 엘리먼트의 수는 147,442개이며 존재하는 모든 가능한 경로는 총 57가지이다. 〈표 11〉은 이상의 내용을 정리한 것이다.

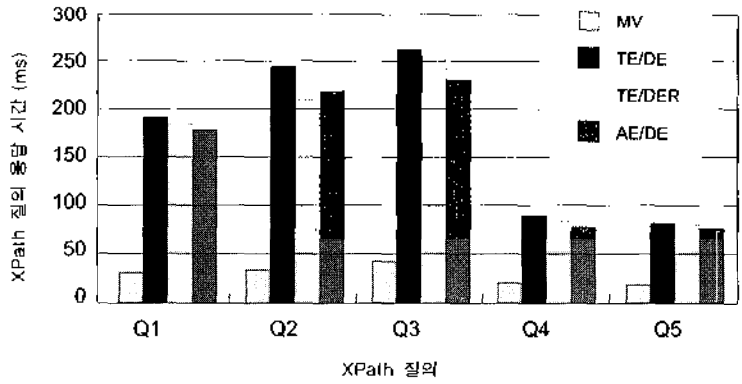
5.2 성능 척도 및 XML 질의

본 논문이 제안하는 XML 질의 캐쉬 기법의 효율성은 NIS 캐싱을 사용하지 않는 질의 재수행(recomputation) 기법과 XML 질의 응답 시간을 비교해봄으로써 확인할 수 있다. 질의 재수행이란 캐쉬되지 않은 질의의 경우 해당 질의가 제기될 때마다 매번 XML 소스를 대상으로 동일한 질의 처리를 반복하는 것을 의미한다. 한편 본 논문에서 제시된 질의 캐쉬 기법의 부담으로는 NIS의 실체화, NIS의 생성, 그리고 NIS의 갱신을 들 수 있다.

〈표 12〉는 실험에 사용된 XML 질의의 정의들이다. Q1의 경우 '/' 연산자로만 이루어진 질의이다. Q2의 경우 '//' 연산자와 '/' 연산자가 함께 사용된 질의 정의이다. Q3의 경우 다수의 '//' 연산자로 이루어진 질의 정의이다. Q4의 경우 '/' 연산자에 필터 연산이 함께 사용된 경우이다. 끝으로 Q5의 경우 '/' 및 '//' 연산자와 필터 연산이 함께 사용된 경우이다.



〈그림 11〉 XML 질의 응답 시간



〈그림 12〉 XML질의 응답 시간

5.3 실험결과

본 절에서는 실험 결과를 기술한다. 모든 실험 결과는 10회 실험의 평균을 구한 것이다.

5.3.1 XML 질의 응답시간

〈그림 11〉은 질의 재수행과 본 논문의 세 가지 NIS 캐싱 기법으로 다양한 질의를 처리할 때의 응답시간을 측정한 결과이다. 본 논문에서 제시된 기법의 경우에는 NIS를 실체

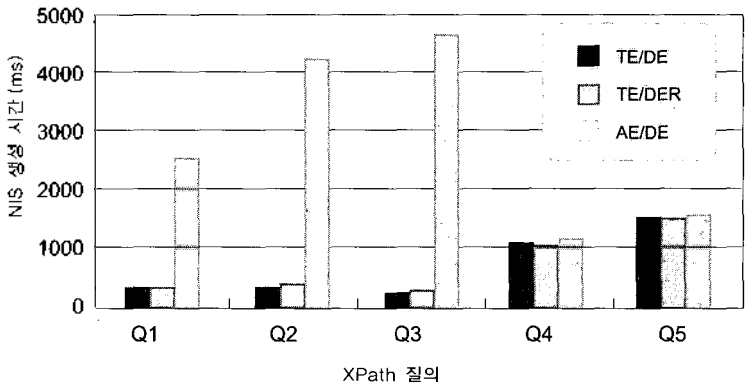
화해야 하는 경우만 평가 대상으로 하였다. 왜냐하면 실체화는 필요 없이 NIS만 반환하는 경우의 질의 처리 성능은 본 논문의 기법의 경우 최적이기 때문이다. 실험 결과 본 논문의 XML 질의 캐싱이 모든 질의에 대해 질의 재수행에 비해 10배 이상의 월등한 성능을 나타냄을 알 수 있었다. 특히, 질의 Q2와 Q3처럼 '//' 연산자가 사용될 때의 질의 처리에 오랜 시간이 걸리는 경우에 캐싱을 통해 현저히 질의 성능을 향상시킬 수 있음을 알 수 있었다.

반면에 NIS를 실체화해야 하는 경우에 본 논문의 기법은 실체뷰 기법과 비교하여 어떤 성능을 나타낼까를 검토해보는 것도 의미가 있다. 〈그림 12〉는 XML 실체뷰 기법과 본 논문의 세 가지 NIS 캐싱 기법의 XML 질의 응답 시간을 비교한 것이다. 〈그림 12〉의 MV는 XML 실체뷰 기법의 성능을 표기한 것이다. 실험 결과 XML 실체뷰 기법이 모든 질의에 대해 본 논문의 XML 질의 캐싱 기법보다 빠른 응답시간을 보였다. 그러나 그 차이는 본 논문 기법의 성능이 질의 재수행의 성능을 압도하는 것에 비해 극히 미미하였다. 이는 본 논문의 NIS 캐싱이 실체뷰 기법에 비해 갖는 여러 가지 장점을 견지하면서 실체화의 부담은 지불하는 것이 효율적이라는 것을 의미한다.

한편, 〈그림 12〉를 통해 본 논문의 세 가지 NIS 캐싱 기법 간의 성능 비교를 해 볼 수 있는데, 질의 응답 시간이 전반적으로 비슷하지만 일부 경우를 제외하면 NIS에 가장 많은 정보를 유지하고 있는 AE/DE 기법이 가장 빠른 응답 시간을 보이며, 다음으로 TE/DER 기법이 빠르고, TE/DE 기법은 그 다음으로 응답 시간이 근소하게 더 걸리는 것으로 나타났다.

5.3.2 NIS 생성 시간

〈그림 13〉은 세 가지 NIS 캐싱 기법의 NIS 생성 시간을 나타낸 것이다. AE/DE 기법의 경우 질의가 많은 목적 엘리먼트를 가질수록 NIS 생성 시간은 증가하였다. 그 이유는 목적 엘리먼트 수가 증가할수록 목적 엘리먼트의 하위에 존재하는 모든 후손 엘리먼트들을 검색하는 횟수가 증가하기 때문이다. 또한 TE/DE 기법과 TE/DER 기법의 경우 질의



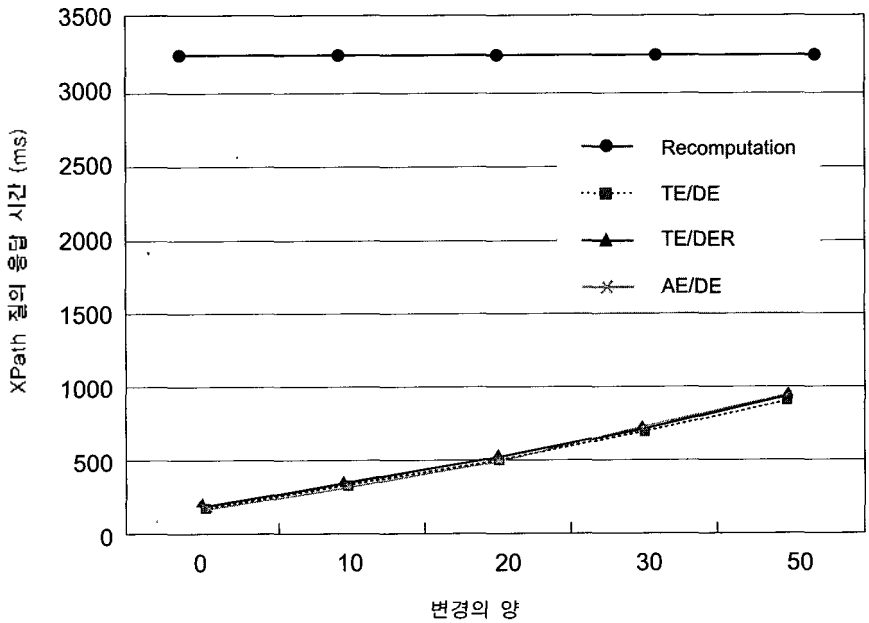
〈그림 13〉 NIS 생성 시간

의 목적 엘리먼트의 수 보다는 질의 정의의 복잡성에 비례해서 NIS 생성시간이 결정되는 것으로 나타났다. NIS 생성 비용은 질의 캐싱 때 한번만 소요되므로 후속 질의 처리 성능과는 독립적이다.

5.3.3 NIS의 점진적 갱신 부담

XML 소스에 발생한 변경을 연관성이 있는 NIS에 즉각 반영하는 시간은 극히 짧지만 캐쉬된 NIS의 수가 웹 응용의 경우처럼 아주 많을 때에는 비효율적인 방법이다. 따라서 본 논문에서는 변경 정보를 변경 로그에 기록해 두었다가 특정 NIS의 갱신이 요구될 때 그때까지 미반영된 변경 로그의 변경 로그 레코드들을 일괄 점검하여 연관성이 있는 것들만 반영한다고 가정한다. 〈그림 14〉는 질의 Q1에 대하여 변경 로그에 누적된 변경의 양이 증가함에 따라 질의 재수행과 본 논문의 기법들 간에 XML 질의 응답시간이 어떻게 변화하는지를 나타낸 것이다. 여기서 누적된 변경은 단말 엘리먼트의 삽입, 삭제, 수정 연산이 균일하게 분포된 것이다.

질의 재수행에서 질의 응답시간이 일정한 이유는 질의 처리를 할 때는 이미 XML 소스의 변경 처리가 끝난 상태이기 때문이다. 세 가지 NIS 캐싱 기법 모두 누적되는 변경의 양이 많이 질수록 NIS의 갱신 부담이 증가하기 때문에 질의 응답 시간이 점점 늦어졌다. 하지만 변경이 상당량 누적되더라도 질의 재수행 기법보다는 훨씬 빠른 응답 시간을 나타냈다. 이는 웹 응용에서 XML 소스 변경의 양이 적정하다면 본 논문의 XML 질의 캐싱이 질의 재수행보다 더 효율적임을 의미한다.



〈그림 14〉 변경 양의 변화에 따른 질의 응답시간

6. 결론 및 향후 연구

XML이 웹 상에서의 데이터 교환의 표준으로 부각되어 웹에서 XML 문서가 차지하는 비중이 점점 커지고 있다. 이에 따라서 웹 상의 방대한 XML 소스에 대하여 신속한 질의 처리를 해줄 수 있는 기술에 대한 연구가 활발히 진행되고 있다. 그 대표적 기술로 XML 질의 캐싱을 들 수 있다.

본 논문에서는 XML 질의 결과 포맷으로 가장 보편적인 노드 식별자 집합(NIS)을 그것의 소스와 더불어 RDBMS에서 유지할 수 있는 세 가지 XML 질의 캐쉬 기법인 TE/DE, TE/DER, AE/DE을 제안하고 구현하였다. 본 논문은 제안한 각 기법 별로 NIS의 생성 및 실체화 알고리즘, 하부 XML 소스에 대한 변경을 점진적으로 NIS에 반영하는 알고리즘을 제안하였다.

성능 평가 실험 결과 XML 질의 처리에 있어서 본 논문의 질의 캐쉬 기법을 사용하는 경우가 그렇지 않은 경우에 비해 훨씬 좋은 성능을 보였다. 특히, XML 질의가 복잡하고 다양한 연산을 필요로 하는 경우에 그 차이가 더욱 커졌다. XML 소스의 변경을 NIS에 반영하는 점진적 갱신의 부담을 평가하는 실험도 이루어졌다. 그 결과 XML 소스의 변경이 지속적이지 않고 그 양이 적정한 웹 응용의 경우 XML 질의 캐싱이 아주 효율적인 것으로 평가되었다. 이러한 실험 및 분석 결과를 통해 XML 질의 캐쉬 기법의 장단점에 따른 질의 최적화의 가이드라인을 제시할 수 있다. 이는 크게 다음 두 가지 사항으로 요약할 수 있다. 첫째, 질의의 복잡도, 빈도, 해당 소스의 변경 빈도에 따른 캐싱 여부 결정. 둘째, 변경 로그에 기록된 XML 소스의 변경 양에 따른 캐쉬의 점진적 갱신과 질의 재수행 간의 선택이다. 질의 최적화 가이드라인은 질의의 복잡도, 빈도가 높을수록, 그리고 소스의 변경 빈도가 낮을수록 캐싱을 사용하는 것이 효율적이고, 누적된 소스의 변경 양이 많지 않을 때 캐쉬의 점진적 갱신을 선택하는 것이다.

본 논문의 향후 과제로는 다음을 들 수 있다. 첫째, 본 논문에서 제시한 NIS 캐쉬 기법에 기반을 둔 차세대 XML 실체뷰 기법의 개발이다. 기존의 XML 실체뷰 기법은 관계 실체뷰 개념을 XML의 특성에 맞게 확장하는 방향으로 연구되어 초보적인 성과를 거두었지만, 확장성(scalability)과 실용성이 떨어지는 한계점을 갖고 있다. 이를 극복하기 위한 차세대 XML 실체뷰 기법의 골격은, NIS의 캐쉬와 더불어 NIS를 완전히 실체화한 종래의 XML 실체뷰가 아닌 웹 응용이 스냅샷으로 요구하는 부분만 실체화하고 나머지는 노드 식별자로 유지하여 관리하는 것으로 구성된다. 둘째, 본 논문에서는 XML 질의 결과로 노드 셋만 고려하였는데 현재 W3C의 XPath 및 XQuery 표준에서 제시한 질의 결과로는 스트링, 수(number), Boolean 등 다양한 데이터 타입이 가능하다. 특히, 집산(aggregate) 연산 등 XML 소스에 나타나지 않는 데이터를 결과로 구하는 경우도 있다. 이들에 대한 캐쉬도 지원하도록 확장하는 연구가 필요하다.

참 고 문 헌

[1] A. Gupta and I. Mumick, "Materialized Views : Techniques, Implementations, and Applications," MIT Press, 1999.

[2] I. Tatarinov et al., "Storing and Querying Ordered XML Using a Relational Database System," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2002, pp. 204-215.

[3] L. Quan et al., "Argos : Efficient Refresh in an XQL-Based Web Caching System," Proc. Workshop on the Web and Databases, 2000, pp. 23-28.

[4] L. Chen and E. Rundensteiner, "Aggregate Path Index for Incremental Web View Maintenance," Proc. 2nd Int'l Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, 2000.

[5] V. Vidal, M. Casanova and V. Araujo, "Generating Rules for Incremental Maintenance of XML View of Relational Data," Proc. ACM Int'l Workshop on Web Information and Data Management (WIDM), 2003, pp. 139-146.

[6] K. Dimitrova, M. El-Sayed and E. A. Rundensteiner, "Order-Sensitive View Maintenance of Materialized XQuery Views," Int'l Conf. on Conceptual Modeling(ER), 2003, pp. 144-157.

[7] G. Huck, I. Macherius and P. Frankhauser, "PDOM : Lightweight Persistency Support for the Document Object Model," Proc. OOPSLA Workshop " Java and Databases : Persistence Options", 1999.

[8] X. Zhang, K. Dimitrova, L. Wang, M. El-Sayed, B. Murphy, B. Pielech, M. Mulchandani, L. Ding and E. A. Rundensteiner, "Rainbow : Multi-XQuery Optimization Using Materialized XML Views," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2003, pp. 671-671.

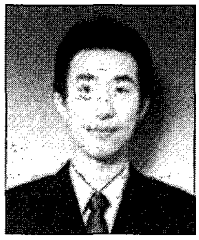
[9] X. Zhang, B. Pielech and E. A. Rundensteiner, "Honey, I Shrunk the XQuery! : An XML Algebra Optimization Approach," Proc. Int'l Workshop on Web Information and Data Management (WIDM), 2002, pp. 15-22.

[10] K. Deschler and E. A. Rundensteiner, "MASS : A Multi-Axis Storage Structure for Large XML Documents," Proc. Int'l Conf. on Information and Knowledge Management, 2003, pp. 520-523.

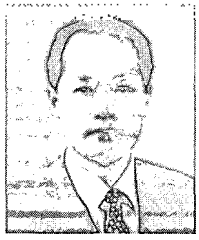
[11] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 2001, pp. 425-436.

[12] E. Cohen, H. Kaplan, and T. Milo, "Labeling Dynamic XML Trees," Proc. PODS02, 2002, pp. 271-281.

저 자 소 개



박대성 (E-mail : dspark@telcoware.com)
2003. 중앙대학교 컴퓨터공학과 졸업(학사)
2005. 중앙대학교 대학원 컴퓨터공학과 졸업(석사)
관심분야 XML 데이터베이스, 음성핵심망



강현철 (E-mail : hckang@cau.ac.kr)
1983. 서울대학교 컴퓨터공학과 졸업(학사)
1985. U. of Maryland at College Park, Computer Science(M.S.)
1987. U. of Maryland at College Park, Computer Science(Ph.D.)
1988 ~ 현재 중앙대학교 컴퓨터학부 교수
관심분야 XML 데이터베이스, 웹 데이터베이스
스트림 데이터 관리, 이동 데이타베이스