유효한 XML 환경에서 유효성과 병행수행의 결합을 위한 낙관적 기법

윤일국 고한영O 박 석
서강대학교 컴퓨터공학과
bingo619@sogang.ac.kr, koonezr@naver.comO, spark@sogang.ac.kr

An Optimistic Mechanism for Combining Concurrency Control and Validation in Valid XML

Ilkook Yun Hanyoung KoO Seog Park
Department of Computer Science and Engineering, Sogang University

요 약

데이터베이스가 DTD를 가지는 valid XML을 관리하기 위해서는 XML 문서를 변경하는 트랜잭션들에 대한변경 유효성을 검사할 수 있는 메커니즘이 필요하게 된다. 그리고 이러한 유효성의 검증 범위는 유효성을 검증하기 위해 필요한 정보를 담고 있는 노드들을 나타낸다고 할 수 있는데 이것은 유효성 검증이 올바르게 수행되기 위해서는 검증 범위에 속하는 데이터 아이템들이 다른 트랜잭션들에 의해서 변경되지 않도록 보장하는 병행수행 제어 기법이 필요하다는 것을 의미하며 이를 위해 유효성과 병행수행에 대한 낙관적 처리 기법이 필요하게 된다. 본 논문에서는 효율적인 충돌 탐지와 같은 검증 범위에서의 유효성 검사를 통해 변경 연산의 트랜잭션들의 병행수행 성능을 향상시키는 기법을 제안하고 기존연구의 유효성 검증과 충돌 탐지 기법을 비교, 분석한다.

1. 서 론

웹 문서를 위한 표준 언어로써 뿐만 아니라 기종이 다른 시스템 간의 데이터 교환 포맷이나 데이터 표현을 위한 포맷으로써 XML이 널리 사용됨에 따라 XML과 데이터베이스의 상호작용에 대한 연구의 필요성은 갈수록 증가하고 있다. 최근에 데이터베이스에 저장된 XML에 대한 질의뿐만 아니라 XML 문서의 일부분을 변경할 수 있는 변경 언어에 대한 연구들이 있었다[1][2]. 이러한 XML 변경은 XML 데이터베이스 시스템에 복잡한 문제들을 야기할 수 있다. 따라서 이러한 연구에는 잘 정형화된(well-formed) XML에 대한 갱신 연산 및 효율적인 처리 기술에 대한 연구와 유효한 XML에서 갱신 연산 후 변경된 문서와 스키마 사이의 검증 작업이 필요하다.

변경 연산에 의해 발생하는 XML 데이터베이스의 문제 중에서 XML 유효성(validity)은 기존의 데이터베이스 시스템에서의 데이터 무결성의 관점에서 바라볼 수 있다. 즉 DTD에 의해 문서 형식이 정의된 XML 문서의 경우 XML에 대한 사용자의 변경이 DTD에 유효하지 않은 결과 XML을 만들어 낼 수 있다면, 이것은 데이터 무결성 측면에서 데이터베이스 시스템에 심각한 문제가 될 수 있다. 따라서 XML 데이터베이스가 DTD를 가지는 valid XML을 관리하기 위해서는 XML 문서를 변경하는 트랜잭션들에 대한 변경 유효성을 검사할 수 있는 메커니즘이 필요하게 된다. 또한 유효성의 검증 범위는 유효성을 검증하기 위해 필요한 정보를 담고 있는 노드들을 나타낸다고 할 수 있다. 이것은 유효성 검증이 올바르게 수행되기 위해서는 검증 범위에 속하는 데이터 아이템들이 다른 트랜잭션들에 의해서 변경되지 않도록 보장하는 병행수행 제어 기법이 필요하다는 것을 의미한다.

현재 대부분의 XML 데이터베이스 시스템들은 트랜잭션에 의한 XML의 변경이 유효한지 검사하기 위해서 변경된 XML 문서를 유효성 검증 파서(validating parser)를 사용하여 파싱함으로써 변경의 유효성을 검증하고 있다. 이것은 항상 문서 전체에 대한 유효성을 검증하기 때문에 잦은 변경이 수행되는 경우 유효성 검증의 오버헤드가 커진다는 단점을 가지고 있다. 이러한 전체 검증의 문제를 해결하기 위한 부분 검증 기법에 대한 연구들이 수행된 바 있지만, 기존의 유효성 검증 기법들은 XML에 대한 사용자의 변경이 DTD에 유효하지 않은 결과를 생성하는 문제점을 안고 있다. 또한 XML 문서에서 여러 트랜잭션들의 병행수행 처리를 위한 충돌 탐지는 트랜잭션들의 변경 연산들이 XML 문서의 특징인 트리 구조의 특징에 따라 각각의 연산들의 실제적인 전이(traverse)를 통해 이루어진다. 따라서 하나의 연산이 소유하고 있는 정보는 다른 연산에 의해 변경될 수 없게 만들어야 한다. 이것은 불필요한 범위의 정보까지도 포함시켜 소유하게 되므로 병행수행 될 수 있는 다른 연산의 수행도 막게 되는 것을 야기한다. 또한 각각의 연산마다 실제적인 트리 구조의 전이(traverse)를 통해 이루어지므로 필요한 정보를 가지는 노드까지의 모든 노드들을 살펴봐야 하므로 같은 XML 문서에서 많은 연산들에 대한 충돌 탐지는 전이(traverse)에 대한 많은 오버헤드를 가지게 된다.

본 논문은 유효성 검증 기법과 유효성 검증 기법의 정확성을 보장하기 위한 병행 수행 제어 기법에 필요한 특징을 살펴보고 병행수행 제어 기법과 유효성 검증 기법 사이의 관련성을 통해 그 효율성을 높이면서 유효한 XML 환경에서의 인스턴스정보를 메타 데이터로 가지는

* 본 연구는 한국과학재단 특정기초연구(R-01-2006-000-10609-0) 지원으로 수행되었음.

효율적인 충돌 탐지 기법을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 배경이 되는 XML에 대한 유효성 검증 기법과 XML 트랜잭션의 병행수행 제어 기법에 대한 관련 연구들을 살펴본다. 3장에서는 본 연구에 대한 자세한 환경과 제안하는 방법을 유효성 검증과 병행수행처리 과정으로 나누어 자세히 설명한다. 4장에서는 실험 및 분석을 통해 제안하는 방법의 우수성을 보이고 5장에서는 본 연구에 대한 결론을 기술한다.

2. 유효성 검증과 병행수행 제어

[3]에서의 유효성 검증은 즉시 검증과 부분 검증으로 나누어 설명할 수 있는데 즉시 검정은 변경이 수행되기 전에 변경 연산의 유효성 여부를 검증하여 유효하지 않은 연산들은 수행되지 않도록 하여 롤백(rollback) 오버헤드를 줄이게 되며 부분 검증은 DTD가 정의하는 엘리먼트들의 순서는 형제 관계에 있는 엘리먼트들간의 지역적인 순서라는 사실에 기초하여, 변경되는 엘리먼트의 형제 엘리먼트 순서만을 검증하는 기법이다.

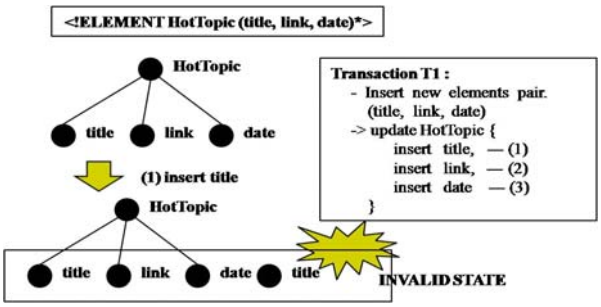

그림 3. 유효하지 않은 연산을 수행하는 유효한 트랜잭션

트랜잭션의 수행 도중 트랜잭션을 구성하는 일련의 연산들에 의해 데이터베이스의 일관성이 깨어지는 상황이 발생하더라도, 트랜잭션의 모든 연산이 다 수행된 후에 데이터베이스의 일관성이 유지되고 있다면 그 트랜잭션은 일관성 속성을 가지고 있다고 보는 것이다. 따라서 [그림 1]의 T1은 의미적으로 유효하고 데이터 베이스의 관점에서도 일관성을 가지고 있는 트랜잭션이기 때문에 수행될 수 있어야 하지만 즉시 검증 기법은 트랜잭션의 대한 유효성 검증이 아닌 변경 연산에 대한 유효성 검증을 수행하기 때문에 트랜잭션의 수행을 허용하지 못하는 문제점을 나타내고 있다. 이러한 문제는 질의처리의 정확성 및 유효성 검증의 정확성에 대한 문제를 가진다고 할 수 있다.

[4]의 연구에서는 데이터 아이템에 대한 read / write 연산들의 충돌을 제어하기 위한 락킹의 단위(granularity)에 따라 4가지 락킹 프로토콜(locking protocol)-Doc2PL, Node2PL, NO2PL, OO2PL-을 제안하고 있다. Node2PL, No2PL, OO2PL은 [그림 2]와 같은 XML 문서 모델을 가정하고 있다. 이 문서 모델은 XML 문서의 트리를 데이터 아이템인 노드와 부모 및 형제 관계의 노드들 간에 연결된 포인터로 구성된 구조로 정의한다. 그리고 트리에 대한 전이(traverse) 연산은 이 포인터를 따라 트리 구조를 순회하고 변경(modify) 연산은 노드의 삭제나 삽입을 위해 이 포인터를 수정하는 연산이라고 정의하고 있다. 이러한 기법은 락의 호환 행렬을 기반으로 트리 구조를 경유하여 들어오는 락을 살펴보면서 탐지하므로 XML 인스턴스에 많은 락이 존재하며, 트리 구조의 실제적 전이(traverse)를 통해 실제 노드에서의 충돌을 탐지하므로 탐지 시간 증가하게 된다.

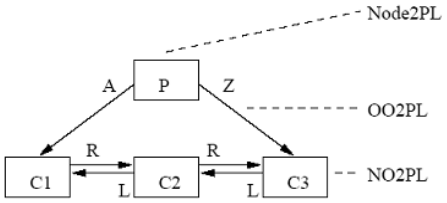

그림 4 Locking Protocol 의 문서 모델

3. 유효성과 병행수행의 결합을 위한 낙관적 기법

신문사나 잡지사 혹은 블로그(Blog)와 같은 웹 사이트에서는 컨텐츠들의 정보가 요약된 RSS[5]라는 XML 형태의 파일을 제공한다. 이러한 RSS는 사용자가 RSS 구독기를 이용하여 웹 사이트로부터 RSS 파일을 풀링(pulling)함으로써 RSS 파일이 서버에서 갱신되었는지 주기적으로 확인하게 되는데 이는 통합 RSS에 대한 각 엘리먼트들에 대한 질의 및 변경을 수행하는 트랜잭션들의 연산에 의해 이루어진다. 따라서 이러한 통합 RSS를 관리하기 위한 데이터베이스 시스템에서는 트랜잭션들이 획득하고자 하는 정보가 다른 트랜잭션들의 변경연산에 의해 내용이 수정될 수 있으므로 이에 대한 유효성 검증 및 병행수행 제어 기법이 필요하다고 할 수 있다.

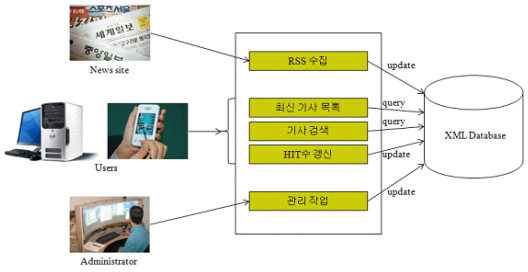

그림 5 Site Syndicate Aggregator의 서비스

예제 환경에서의 통합 RSS는 DTD에 의해 문서 형식이 정의된 valid XML 파일이다. 각 사이트에서 제공하는 컨텐츠가 증가함에 따라 item 엘리먼트들의 수도 증가하므로 통합 RSS는 시간에 따라 문서 트리의 크기가 횡적으로 증가하는 문서라고 할 수 있다. 이러한 환경에서는 부분 검증 기법 및 형제 노드 락킹 기법이 검증 범위가 변경 노드의 모든 형제 노드를 포함하기 때문에 변경되는 노드의 형제의 노드수가 비교적 많다. 따라서 유효성 검증의 오버헤드가 증가하기 될 뿐만 아니라 병행수행 능력도 락킹되는 노드의 수가 많아지기 때문에 전체적으로 낮은 성능을 가지게 된다. 따라서 락이 아닌 다른 방법의 충돌 탐지의 방법과 이때 같은 검증 범위안에서의 유효성 검사를 통해 오버헤드를 줄여서 데이터베이스 트랜잭션 병행수행 성능을 향상 시킬 수 있는 기법을 제안하고자 한다.

3.1 시스템 구조

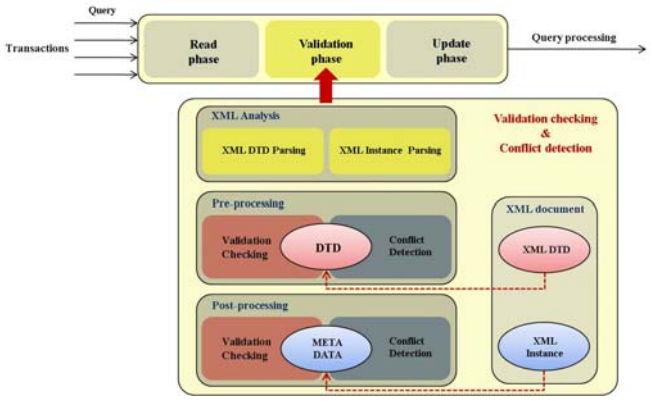

그림 6 시스템 구조

[그림 4]는 본 연구에서 제안하는 시스템의 구조를 보여주고 있다. XML 데이터베이스로 다양한 트랜잭션들로부터 다양한 쿼리가 입력되게 되는데 이때 Read Phase는 이러한 쿼리들을 입력 받는다. 입력된 쿼리들은 Validation Phase를 통해서 입력된 쿼리들이 유효한지 그리고 병행수행 처리가 가능한지를 판단하게 되는데 이때 각 쿼리들은 실행되던지 아니면 취소나 롤백(rollback)되거나 대기상태로 되게 된다. Validation Phase 를 통해 걸러진 쿼리들, 즉 병행수행 처리를 통해 실행 가능한 쿼리들만이 Update Phase로 입력되어 지고 입력된 쿼리들은 실행(processing) 되게 된다.

3.2 XML 분석

XML 분석 단계는 크게 두 부분으로 나뉘어질 수 있는데 첫번째로는 현재 문서들의 DTD를 분석하는 단계이고 두번째로는 분석된 DTD를 통해 인스턴스 레벨로 내려가서 분석된 정보를 가져오는 단계이다.

첫 번째 DTD 파싱 단계는 XML DTD를 통해 각 엘리먼트에 어떤 심볼(symbol), 즉 어떤 카디널리티가 연결되었는지를 파악하고 엘리먼트들의 반복(recursive)에 대한 정보와 데이터를 필요로 하는 엘리먼트들의 종류를 파악할 수 있게 된다. [그림 5][그림 6]은 예제 환경에서의 통합 RSS의 DTD를 분석한 것을 보여주고 있다.

```
<!ELEMENT rss (channel)*>
<!ELEMENT channel (title, link, item*, description, lastmodified, (author | editor), hit, rank? )>
<!ELEMENT item (title, link, description, pubdate, author, item*)>
<!ELEMENT title #PCDATA>
<!ELEMENT link #PCDATA>
<!ELEMENT description #PCDATA>
<!ELEMENT author #PCDATA>
<!ELEMENT editor #PCDATA>
<!ELEMENT lastmodified #PCDATA>
<!ELEMENT pubdate #PCDATA>
<!ELEMENT hit #PCDATA>
<!ELEMENT rank #PCDATA>
```

그림 7 통합 RSS DTD

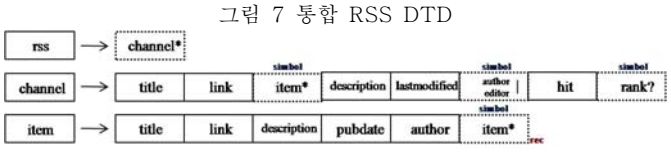

그림 8 통합 RSS DTD 분석

우선 각 노드를 순차적으로 탐색하게 된다. 처음의 rss 노드를 탐색후 그 하위노드인 channel 노드를 탐색한다 그리고 다시 그 하위 노드들인 title, link, item, description, lastmodified, author, editor, hit, rank들을 차례대로 탐색하면서 반복적으로 발생되는지 아니면 심볼이 붙어있는지를 체크한다. 여기서는 item에 *심볼이 있으므로 item에 대한 부모노드와 심볼을 저장하고, author와 editor에 |심볼이 있으므로 author과 editor에 대한 부모노드와 심볼을 저장한다. 역시 마찬가지로 rank에 대한 부모노드와 ?심볼을 저장한다. 다음으로 item에 대한 하위노드를 탐색한다. 여기서는 item에 대한 * 심볼이 나오지만 item은 반복적으로 나타나기 때문에 이에 대한 정보도 필요하다고 체크하게 된다. 또한 데이터값이 필요한 노드에 대한 정보도 체크한다. [그림 7]은 분석 후 필요한 정보들의 테이블을 보여주고 있다.

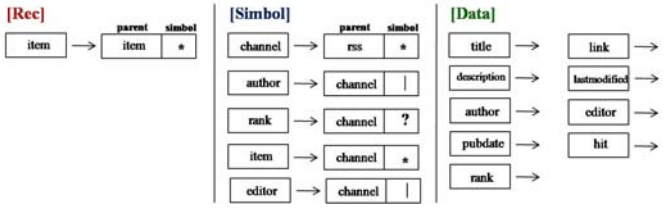

그림 9 DTD 분석 후 필요한 정보

두 번째 인스턴스 파싱 단계로서 직접적으로 현재 XML 문서(document) 레벨로 내려가게 된다. 이때의 XML 인스턴스에 포인트로 연결된 Dewey Order Numbering 스킴[6]을 사용하게 된다. 루트 노드부터 차례대로 인덱싱이 되어지는데 이때 이전 DTD 레벨에서 가져온 정보의 빈 칸에 차례대로 해당되는 정보를 저장해 주면 된다. 그리고, 심볼이 포함된 인스턴스는 동적인 업데이트를 위해서 각각의 노드의 CDBS(Compact Dynamic Binary String)기법[7]을 사용한 인덱싱도 함께 저장된다.

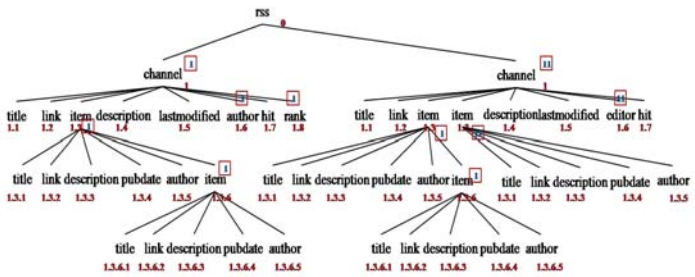

그림 10 XML 인스턴스 정보

[그림 8]은 현재 XML 문서에 대한 인스턴스 정보를 나타내주고 있다. [그림 9]은 [그림 7]에서 XML DTD 분석 후 필요한 정보들을 XML 인스턴스 분석을 통해 저장되어진 rss를 부모로 가지는 *심볼을 가진 channel 노드와 channel을 부모로 가지는 *심볼을 가진 item노드, channel을 부모로 가지는 ?심볼을 가진 rank노드, channel을 부모로 가지고 |심볼을 가진 author와 editor노드에 대한 정보와 item을 부모로 가지고 *심볼을 가지는 item노드에 대한 반복(recursive)에 대한 정보를 보여주고 이러한 데이터를 XML 인스턴스에서 분

석되어진 현재 문서의 메타데이터(Metadata)라고 할 수 있다.

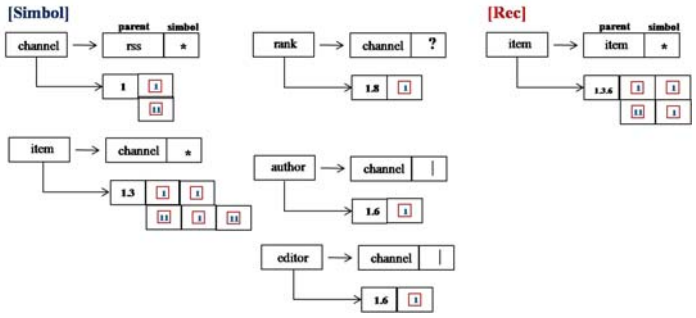

그림 11 메타데이터

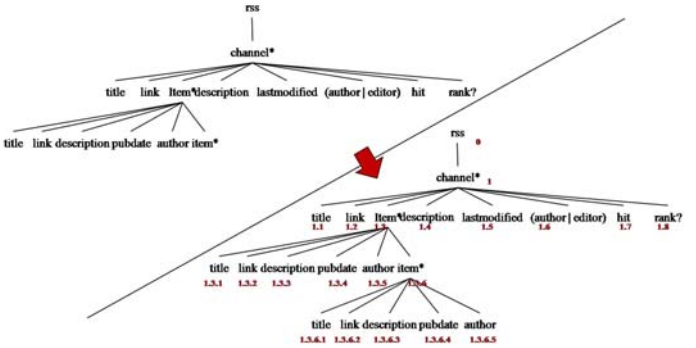

그림 12 DTD 트리 변환

[그림 10]에서의 좌측의 트리는 통합 RSS의 예제 환경의 기본 DTD에서의 트리를 나타내주고 있다. 하지만 여기서는 item 밑의 item 노드에 대한 반복구조에 대한 정확한 구조를 알 수 없기 때문에 이에 대한 노드들의 구조를 정확히 판단하기 어려워서 반복구조에 대한 충돌 탐지가 불가능하다. 하지만 XML 인스턴스 정보에서 가져온 정보를 통해 우측의 트리처럼 확장이 가능하며 이를 통해 현재 문서에서의 item에 대한 반복되는 구조에 대한 정확한 정보를 알 수 있게 된다. 지금까지의 단계를 통해 기본 과정인 XML 분석과정이 모두 마치게 되고 이를 통해 다음절에서 이루어질 유효성에 대한 검증과 병행수행을 위한 충돌 탐지가 이루어지게 된다.

3.3 유효성 검증

자식 엘리먼트들의 순서 및 개수는 엘리먼트 선언에서 순차 내용 모델(sequence content model), 선택 내용 모델(choice content model), 카디널리티(cardinality)를 사용하여 정의된다. 순차 내용 모델은 ‘,’ 로 표현되며 자식 엘리먼트들의 기본적인 순서를 정의한다. 선택 내용 모델은 ‘|’ 로 표현되며 XML 문서에서 자식 엘리먼트들이 나타날 수 있는 상대적인 조건을 정의한다. 그 중에서 카디널리티는 [그림 11]과 같이 다음의 ‘*’, ‘+’, ‘?’, ‘None’의 4가지로 구분하여 표현될 수 있으며 이를 통해 자식 엘리먼트들의 개수를 정의하게 된다. 순차 및 선택 내용 모델 내용 그리고 카디널리티는 서로 혼합되어 사용될 수 있으며, 그것들에 의해 자식 엘리먼트들에 대한 복잡한 순서를 정의할 수 있다. 이러한 DTD 제약사항을 통해서 선처리 과정과 후처리 과정으로 나뉘어서 유효성 검증이 일어날 수 있다.

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

그림 13 DTD의 지시자 종류

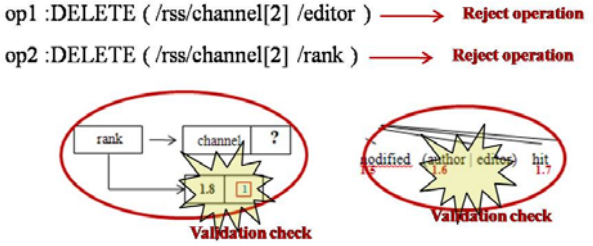

그림 14 유효성 검증 예제

[그림 12]에서의 예제를 통해 살펴보면, 먼저 op1의 연산은 rss 노드 하위의 channel 노드의 editor 노드에 대한 삭제 연산이다. 하지만 DTD를 통해서 살펴보면 editor노드는 author노드와의 선택모델로서 반드시 하나는 존재 해야 한다. 따라서 editor노드에 대한 삭제 연산 op1은 DTD에 의한 선처리 과정에 의해 유효성에 위배가 되어서 거절 질의로 판명된다. 그리고 op2는 rss 노드 하위의 channel 노드의 rank 노드에 대한 삭제 연산이다. 하지만 rank 노드는 ? 심볼을 가지고 있기 때문에 이에 대한 정확한 정보를 알지 못한 상황에서는 삭제 연산을 수행할 수 없게 된다. 따라서 XML 인스턴스의 메타데이터를 가지고 유효성을 판단하는 후처리 과정을 통해 channel[2]에 대한 rank 노드는 존재하지 않으므로 이에 대한 유효성 검증 역시 위배되어서 거절 질의가 된다.

3.4 충돌 탐지

충돌 탐지에 앞서 목적노드를 정의한다.

- **목적노드** : 주어진 XPath 경로 표현식에서 프레디키트를 제외한 마지막 노드를 목적노드라 한다.

해당 연산들의 DTD를 가지고 이에 대한 목적노드를 찾고 목적노드들의 인덱스의 범위를 확인하여 ancestor, descendant, self 관계를 파악해서 충돌을 탐지한다. 그리고 추가적으로 XML 분석과정을 통해서 얻은 XML 메타데이터인 [그림 9]의 정보를 통해 프레딧킷에 대한 부정확한 충돌을 정확하게 탐지하는 방법을 사용한다.

연산 Oi와 Oj에 대한 관계는 다음과 같이 분할되어진다.

- Oj의 인덱싱의 숫자가 Oi의 인덱싱의 숫자보다 클때 Oj는 Oi에 대해 FOLLOWING 관계라 정의한다.
- Oj의 인덱싱의 숫자가 Oi의 인덱싱의 숫자보다 작을때 Oj는 Oi에 대해 PRECEDING 관계라 정의한다.

- Oj의 포인트 인덱싱의 숫자가 Oi의 인덱싱의 숫자보다 앞에 나타날 때 Oj는 Oi에 대해 ANCESTOR 관계라 정의한다.
- Oj의 인덱싱의 숫자가 Oi의 인덱싱의 숫자보다 뒤에 나타날 때 Oj는 Oi에 대해 DESCENDANT 관계라 정의한다.
- Oj의 인덱싱의 숫자가 Oi의 인덱싱의 숫자와 같을 때 Oj는 Oi에 대해 SELF 관계라 정의한다.

이상의 다섯 가지 분할에 의해 각 연산에 대한 충돌 탐지가 이루어지는데 먼저 READ 연산과 UPDATE 연산에 대한 충돌은 두 연산의 관계가 DESCENDANT 관계이거나 ANCESTOR 관계, 그리고 SELF 관계 중에 하나라도 만족하게 되면 두 연산은 충돌이라고 할 수 있다. 다음으로 UPDATE 연산과 UPDATE 연산에 대한 충돌을 살펴보자. 이때 두 연산에서 연산들 사이에 관계가 DESCENDANT 이거나 ANCESTOR 관계 또는 SELF 관계 중에 하나라도 만족되면 두 연산은 충돌이라고 할 수 있다.

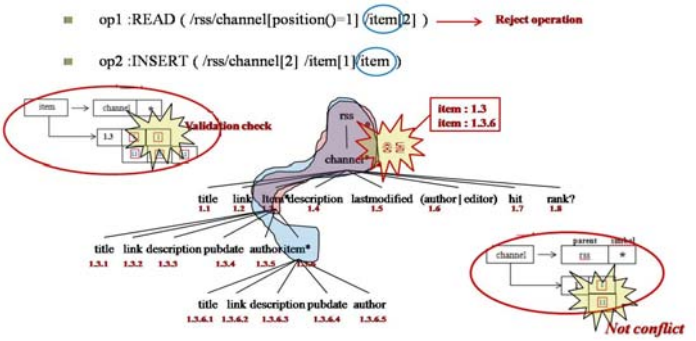

그림 15 충돌 탐지 예제

[그림 13]에서의 예제에서의 두연산 op1과 op2의 충돌을 살펴보면 먼저 op1의 목적노드인 item(1.3)과 op2의 목적노드인 item(1.3.6)은 서로 ancestor 관계를 통해 충돌임을 알 수 있다. 하지만 목적노드들인 item의 상위 노드channel노드를 메타데이터로 살펴보면 rss하위의 channel 노드는 1번과 2번 위치를 가지는 것으로 구분할 수 있다. 따라서 메타데이터를 통해서 충돌이 일어나지 않고 병행수행이 가능한 연산이라고 판명되어진다. 하지만 여기서 우리가 유효한 연산이라고 생각하고 탐지했던 op1은 메타데이터를 통해서 살펴보면 실제적으로 rss 노드 하위의 포지션 값을 1로 가지는 channel 노드에는 item이 하나 존재 하기 때문에 item[2]에 대한 연산은 실제로는 유효성에 위해 되어 거절 질의 이다. 따라서 같은 검증 범위에서의 유효성을 통해서 우리는 충돌 탐지에 대한 이중적 처리를 사전에 막을 수 있기 때문에 이에 대한 효과를 얻을 수 있게 된다. 처리 과정에 대한 Pseudo Code로 나타낸 것이 [그림 14]와 같다.

4. 제안하는 기법의 분석 및 비교

실험에 앞서 실험에 대한 가정으로 유효하지 않은 트랜잭션의 수행이나 데드락(deadlock)현상에 의해서 트랜잭션이 취소되는 현상이 발생하지 않도록 트랜잭션들

```
Input : Operations query of transactions, hashmap of Meta data
Output : Validation value and Conflict detection value


while (input_op) {
    if checking validation rule of input_op with DTD and hashmap of meta data then
        if checking the transaction is over then
            establish reject query operation
        else
            waiting
        end if
    end if
    if checking input_op has predicates then
        if input_op is dynamic then
            conflict detection of input_op
            with hashmap of meta data and DTD and CDBS

            update meta data and CDBS
        else
            conflict detection of input_op
            with hashmap of meta data and DTD

            update meta data
        end if
    else
        if input_op is dynamic then
            conflict detection of input_op with DTD and CDBS
            update meta data and CDBS
        else
            conflict detection of input_op with DTD
            update meta data
        end if
    end if
}
return to validation and Conflict_detection value
```

그림 16 알고리즘

을 정의하고 루트 노드에서 접근되는 노드까지의 경로에 있는 노드들의 수와 깊이만을 고려 연산의 수행중에 발해되는 다른 수행 시간은 고려되지 않는다.

첫 번째 유효성 검증에 대한 실험으로 실험에 사용된 변수는 우선 maxTransaction을 1로 함으로써 트랜잭션들간의 변경 없이 하나의 트랜잭션으로 연산이 수행되도록 하였다. 이때 channel당 item 엘리먼트의 수를 10에서 100까지 증가 시키면서 pan-out 증가에 따른 검증 소요 시간의 변화를 측정하였다. [그림 15]는 이 실험에 대한 결과를 보여주고 있다. 전체 검증의 경우 문서 크기의 증가에 따라 검증 소요 시간 큰 폭 증가하며 부분 검증의 경우 문서 크기의 증가에 따라 시간이 증가하지만 변경 트랜잭션이 1이므로 소폭으로 증가하게 된다.

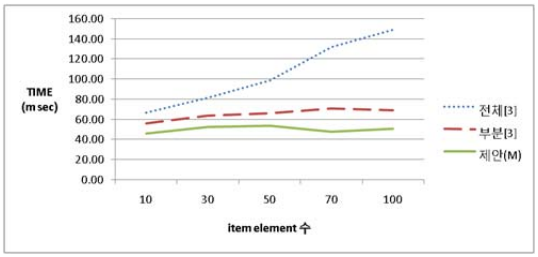

그림 17 XML item 증가에 따른 유효성 검증

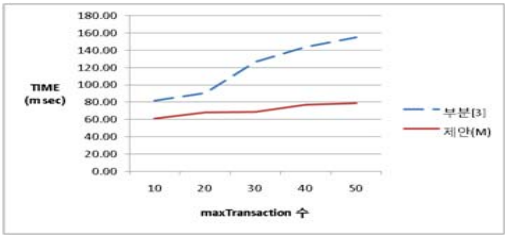

그림 18 XML transaction 증가 따른 유효성 검증

두 번째 실험에 사용된 변수는 우선 item 엘리먼트의 수를 100으로 고정하고 maxTransaction의 수를 10에서 50까지 증가 시키면서 트랜잭션들의 연산의 증가에 따른 검증 소요 시간의 변화를 측정하였다. [그림 16]은 이 실험에 대한 결과를 보여주고 있다. 첫 번째 실험에

서 하나의 트랜잭션에서의 item 엘리먼트의 증가에 많은 영향을 받지 않았던 부분 검증의 방법이지만 트랜잭션의 수를 증가시킴으로 인해 각각의 연산들의 모든 형제 노드들의 순서를 검증해야 하므로 큰 폭으로 시간이 증가함을 볼 수 있다. 하지만 제안하는 기법은 DTD와 메타 데이터를 통해 유효성을 바로 검증이 가능하여 문서 크기의 증가나 아무리 많은 트랜잭션들로부터 많은 연산들이 들어 온다 하더라도 영향을 받지 않는 것을 알 수 있다.

두 번째 병행수행에 대한 실험으로 실험 변수로 maxTransaction의 값을 100으로 설정하여 최대 100개의 트랜잭션들의 연산이 병행수행이 되도록 하였으며, channel 값은 5로 설정하여 병행 수행되는 트랜잭션들 중 적어도 20% 정도는 같은 channel 엘리먼트에 접근하도록 하였다. 각 실행에서 트랜잭션들의 연산은 모두 100개씩이 수행되었다. 그리고 질의/변경 트랜잭션의 연산의 비율이 0 ~ 90%로 변화시켜가면서 기존 연구에서의 문서 락킹 기법인 NL방법과 형제 노드의 부모락킹 기법인 SL방법과 본 연구에서 제안하는 메타 데이터를 이용한 M방법을 측정하였다.

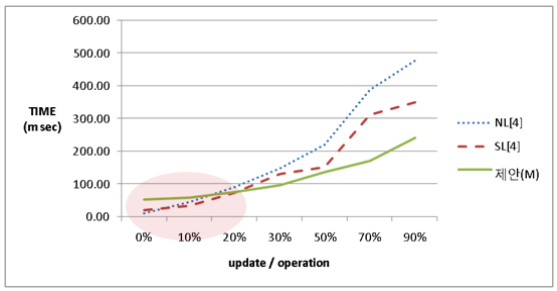

그림 19 질의/변경 연산비율에 따른 병행수행성능 검증

세 기법 모두 변경 트랜잭션의 비율이 증가함에 따라 수행시간이 증가하는 것을 확인할 수 있다. 특히 질의 트랜잭션만이 수행되는 경우 락에 의해 블록되는 트랜잭션이 없기 때문에 락킹에 대한 오버헤드가 가장 적은 문서 락킹기법인 NL이 가장 좋은 성능을 보이고 있다. 하지만 문서락킹과 부모노드락킹을 사용한 경우는 변경 트랜잭션의 비율이 증가함에 따라 병행수행 능력이 하락한 반면 메타 데이터를 이용한 기법은 변화의 폭이 작다는 점이다. 이것은 어떤 애플리케이션 도메인에서 트랜잭션의 패턴이 다를 수 있는데 모든 상황에서 안정적인 성능을 보장해 줄 수 있다는 것을 의미한다.

5. 결론 및 향후 연구

본 연구를 통해 유효한 XML 환경에서의 효율적이고 효과적인 연산 간의 충돌을 탐지 하고 유효성을 검증할 수 있었다. 하지만 본 연구에서의 예제 환경은 많은 pan-out의 경우를 설정하고 있는데 이와 반대로 레벨이 깊어지는 환경에서는 형제 노드들의 부모 노드 락킹에 의한 병행수행처리는 락의 수가 적어지고 유효성 검증에 있어서 부분검증 역시 형제노드가 적어지게 된다. 이러한 경우에는 검증에 앞서 DTD에 대한 파싱 과정을 가지는 본 연구의 기법이 제약사항을 가질 수 있을 것이라 여겨진다. 하지만 본 연구는 현재 문서의 트리를 전이하는 것이 아니고 DTD를 통해 처리되는 시스템이므로 이러한 시스템은 유효한 XML 데이터베이스와 독립성을 추구하기 때문에 어떤 XML DBMS에서도 탑재가 가능하다는 장점도 가지고 있다. 지금은 연속적으로 들어오는 동적인 트랜잭션들의 변경 연산에서 포지션 프리디킷만을 고려하였지만 포지션 뿐 아닌 모든 인스턴스에 대한 동적 변경 연산에 대한 확장 연구를 추후 연구 과제로 남기고자 한다.

참고문헌

[1] sdfIgor Tatarinov, Zachary G. Ives, Alon Y. Halevy, Daniel S. Weld, "*Updating XML*", ACM SIGMOD, pp413-424, Santa Barbara, California, USA, May 2001

[2] Jonathan Robie, Ratrick Lehti, "*Updates in Xquery*", In Proceeding of XML Conference, 2001

[3] Sang-Kyun Kim, Myungcheol Lee, Kyu-Chul Lee, "*Immediate and Partial Validation Mechanism for the Conflict Resolution of Update Operations in XML Databases*", WAIM 2002, Vol.2419, pp.387-396, Aug 2002

[4] Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, "*Evaluating Lock-based Protocols for Cooperation on XML Document*", SIGMOD Record, Vol 33, No.1, March 2004

[5] D. Winer. RSS 2.0 Specification. Available at http://blogs.law.harvard.edu/tech/rss, 2005

[6] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita and Chun Zhang, "*Storing and Querying Ordered XML Using a Relational Database System*", In Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2002, pp. 204-215

[7] Changqing Li, Tok Wang Ling, Min Hu "*Efficient Processing of Updates in Dynamic XML Data*" In Proc. Of ICDE 2006

[8] Torsten Grabs, Klemens Bohm, Hans-Jorg Schek, "*XMLTM: Efficient Transaction Management for XML Documents*", CIKM' 02, McLean, Virginia, USA, November 2002

[9] Stijn Dekeyser, Jan Hidders, Jan Paredaens, "*Transaction Model for XML Databases*"World Wide Web Journal, Kluwer, 2003