유효한 XML 문서에 대한 경계 로킹에 기반한 시퀀스 그룹 검증 기법

(Sequence Group Validation based on Boundary Locking for Valid XML Documents)

최 윤 상 † 박 석 ††

(YoonSang Choi) (Seog Park)

요 약 많은 웹 응용 영역에 XML이 적용되면서 트랜잭션의 변경과 접근에 대한 고립성을 만족시키는 XML 문서에 대한 병행수행은 중요한 이슈가 되고 있다.

DTD(혹은 XML 스키마)의 규칙을 잘 지키는 문서를 유효한 XML 문서라 하는데, 유효한 XML 문서에 대한 갱신 연산은 연산 후의 XML 문서가 원래 DTD의 규칙을 그대로 유지해야하는 유효성 문제를 앉고 있다. 일반적인 유효성 검증 방법은 갱신 후의 XML 문서 전체에 대해 유효성을 검증하는 방법이다.

그러나, 위에서 언급한 유효성 검증 방법은 낮은 병행수행의 결과를 낳는다. 따라서, XML 문서의 유효성 검증 범위를 최소화하면서 높은 병행수행 정도를 보이는 새로운 유형성 검증 방법과 로킹 방법이 요구된다.

본 논문은 유효성 검증의 검증 범위를 최소화 시켜 유효성 검증이 효율적으로 수행될 수 있는 시퀀스 그룹 검증 기법을 제안한다. 또한 이 검증 기법의 정확성을 보장하면서 로킹되는 데이타 아이템의 수를 최소화 할 수 있는 경계 로킹 기법을 제안한다. 마지막으로 제안된 유효성 검증 기법과 경계 로킹 기법이 기존의 방법에 비해 트랜잭션의 병행수행 성능을 향상시키고 있음을 실험을 통해 보인다.

키워드 : XML, DTD, 유효성 검증, 병행수행 제어

Abstract The XML is well accepted in several different Web application areas. As soon as many users and applications work concurrently on the same collection of XML documents, isolating accesses and modifications of different transactions becomes an important issue.

When an XML document correctly corresponds to the rules laid out in a DTD or XML schema, it is also said to be valid. The valid XML document's validity should be guaranteed after the document is updated.

The validation method mentioned above, however, results in lower degree of concurrency. For getting higher degree of concurrency and minimizing the range of the XML document validity, a new validation method based on a specific locking method is required.

In this paper we propose the sequence group validation method for minimizing the range of the XML document validity. We also propose the boundary locking method for isolating accesses and modifications of different transactions while supporting the valid XML document's validity. Finally, the results of some experiments show the validation and locking methods increase the degree of transaction concurrency.

Key words : XML, DTD, validation, Concurrency Control

· 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2004-041-D00569)

† 비 회 원 : 서강대학교 컴퓨터학과
heiyo@dblab.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과
spark@dblab.sogang.ac.kr
(Corresponding author임)

논문접수 : 2004년 8월 13일
심사완료 : 2005년 8월 29일

1. 서 론

현재 XML은 웹 문서를 위한 표준 언어라는 개발 당시의 목적을 넘어 데이타 표현 및 교환 언어로써 널리 사용되고 있다. 이러한 XML 사용빈도의 증가는 XML로 표현된 데이타를 관리할 수 있는 데이타베이스 시스템을 요구하게 되었고, 데이타베이스에 저장된 XML 데

이타에 대해 접근할 수 있는 XML 질의 언어를 필요로 하게 되었다. 그래서 W3C는 XML 문서 집합에 대해 데이타베이스에 접근하는 것처럼 질의할 수 있게 하는 XQuery라는 질의 언어를 개발하였다[1,2]. 하지만 XML 형태로 생산되는 데이타의 증가는 XML 데이타에 대한 질의뿐만 아니라 변경 연산을 필요로 하게 됨으로써 질의 구문만을 정의한 XQuery 1.0 스펙은 한계를 드러내게 되었다. 이에 따라 최근에 XQuery 문법에 변경을 위한 구문을 추가함으로써 XML 변경을 지원하는 XML 조작언어에 대한 연구들이 수행되었다[3,4].

XML 데이타가 가지고 있는 특징들은 변경 연산과 관련하여 복잡한 문제를 야기할 수 있다. XML 문서의 각 노드들은 계층적 구조 하에서 부모-자식 관계라는 수직적 관계로 연결되어 있다. 그리고 XML 트랜잭션에서 노드에 접근하기 위해서는 XPath와 같은 경로 표현을 사용하게 된다. XML의 노드는 하위 노드를 구조적으로 포함하는 데이타 아이템이면서 그 자체로도 연산의 대상이 되는 데이타 아이템이기 때문에, 계층적 구조와 경로 표현을 통한 접근이라는 XML 데이타의 특징은 트랜잭션들이 병행수행되는 경우 기존의 방법으로 제어할 수 없는 문제를 안고 있다. 이러한 XML 트랜잭션에 대한 병행수행 제어의 문제를 해결하기 위해 [5-9]와 같은 연구들이 수행되었다.

그러나 XML에서 데이타 아이템들은 트리 구조 하에서 수직적 관계를 갖는 노드의 특징만을 갖는 것은 아니다. XML에서의 노드들은 형제 관계에 있는 노드들과의 순서 관계가 존재한다. 특히 DTD에 의해 문서 형식이 정의된 유효(valid) XML의 경우에는 엘리먼트의 유효한 순서(valid order)가 정의되어 있기 때문에 노드들 간의 수평적 관계가 중요하다.

유효 XML에서 데이타 아이템 간의 수평적 관계는 트랜잭션들의 엘리먼트 삽입/삭제 연산들의 순서 충돌 외에도 변경 유효성 검증과 관련된 트랜잭션 병행수행 제어에도 영향을 미친다. 하지만 기존의 병행수행 제어에 대한 연구들은 노드들의 수직적 관계에 의해서 발생하는 의미적인 충돌에만 초점을 맞췄기 때문에 노드들 간의 수평적 관계에 의해서 발생하는 연산 충돌에 대해서는 고려하지 않았다. 그리고 노드들 간의 수평적 관계에 의해서 발생하는 문제들은 변경 유효성 검증의 측면에서 연구된 바 있으나[10], 검증 효율성만 고려하였을 뿐 유효성 검증 기법이 영향을 미칠 수 있는 트랜잭션의 병행수행 성능에 대해서는 고려하지 않았다.

본 연구에서는 유효 XML 문서에 대해 트랜잭션들이 병행수행 되는 환경에서 노드들의 수평적 관계에 의해서 발생하는 변경 유효성의 문제에 대해 살펴보고, 유효성 검증 기법과 병행수행 제어 기법의 관계에 대해 설명한다. 그리고 대용량 XML 문서에 대한 효율적인 유효성 검증을 수행할 수 있는 시퀀스 그룹 검증 기법을 제안하고, 노드들 간의 수평적 관계를 고려함으로써 시퀀스 그룹 검증의 정확성을 보장하며 더 나은 병행수행 성능을 제공할 수 있는 경계 로킹 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 XML 문서 변경에 대한 기존의 병행수행 제어 기법들과 변경 유효성 검증 기법들을 살펴보고, 기존 연구들의 문제점을 지적한다. 그리고 3장에서 시퀀스 그룹 검증 기법과 경계 로킹 기법에 대해 기술한 후 4장에서 제안하는 방법의 성능에 대해 평가한다. 그리고 5장에서 결론 및 추후 연구에 대해 설명하겠다.

2. 관련 연구 및 문제점

2.1 트랜잭션의 연산

표 1은 [1,2]에서 제안된 XQuery의 구문과 [3,4]에서 정의된 변경 구문에 기초한 유효 XML에 대한 트랜잭션 연산들의 정의를 보여 주고 있다. [3,4]에서 제안된 변경 연산 중 REPLACE 연산은 DELETE 및 INSERT 연산의 쌍으로 치환될 수 있기 때문에 제외하였으며, RENAME 연산은 유효 XML 문서에 대한 변경 연산으로는 큰 의미가 없기 때문에 고려하지 않았다.

여기서 선택 노드들에 대한 read 연산은 선택 노드들의 하위 노드들을 모두 읽어 들인다는 점에서 XPath에 의한 해당 노드를 선택하는 select 연산과 다르다. 즉, select 연산은 노드 선택을 위해서 XML 문서의 트리 구조를 탐색하는 연산이고, read 연산은 선택 노드의 서브 트리를 모두 읽는 연산이라고 할 수 있다.

표 2는 XQuery 구문에 기초한 질의 및 변경 연산들이 어떻게 표 1의 연산들로 표현될 수 있는지를 보여주고 있다.

2.2 병행수행 제어 기법

XML은 데이타 아이템들이 트리 형태로 계층적으로 구성되어 있다. 이러한 XML의 계층 구조는 경로 표현을 사용하는 접근이라는 접근 방식의 특징과 관련하여 병행수행되는 트랜잭션의 충돌을 야기할 수 있다. 즉, 한 트랜잭션이 어떤 데이타 아이템에 접근하기 위해서는 계층적으로 그 데이타 아이템의 상위에 있는 데이타 아이템들이 먼저 접근되어야 한다. 이러한 데이타의 계층적 구조에 따른 접근 특성은 여러 트랜잭션들이 병행수행되는 경우 여러 가지 의미적인 충돌을 유발할 수 있다. [5-9]는 XML 트랜잭션들 사이에 발생하는 이러한 문제를 해결하기 위한 병행수행 제어 기법들을 제안하고 있다.

[5,6]은 XML의 트리 구조를 순회하는 연산과 트리 구조에 대한 변경을 수행하는 노드 삽입/삭제 연산들의

표 1 XML 트랜잭션의 연산 정의

Operation	Return Value	Description
select(path)	NodeSet (Path에 의해 선택된 노드들의 집합)	- Path에 의해 선택되는 노드들의 집합을 리턴하 는 연산 - XQuery의 FOR, LET 문에 해당하는 연산
read(NodeSet)	XML fragment	- NodeSet의 노드들의 내용을 리턴하는 연산, 즉 노드의 subtree를 모두 읽음 - XQuery의 Return 문에 해당하는 연산
insert(NodeSet, child) insertBefore(NodeSet, ref, child) insertAfter(NodeSet, ref, child)	NodeSet (삽입된 새로운 자식 노드들의 집합)	- NodeSet의 모든 노드들에 대해 새로운 자식 노드인 child를 삽입한다 - NodeSet의 모든 노드들에 대해 ref라는 자식 노드의 앞(뒤)에 새로운 자식 노드 child를 삽입한다.
delete(NodeSet, child)	boolean (true | false)	- NodeSet의 모든 노드들에 대해 자식 노드인 child 노드를 삭제한다. - child라는 자식 노드가 존재하지 않으면 false

표 2 XQuery 표현과 표 1의 연산 표현

XQuery 표현	표 1의 연산에 의한 표현
FOR $c IN /rss/channel LET $i := $c/item[0] RETURN $i	$c = select('/rss/channel') $i = select('$c/item[0]') result = read($i)
FOR $c IN /rss/channel[11] UPDATE $c { insert <lastmodified> 20040614 11:29 </lastmodified> after $c/lastmodified[0] delete $c/lastmodifed[0] insert <item> ... </item> before item[0]	$c = select(/rss/channel[11]) $1 = insertAfter($c, 'lastmodified[0]', 'lastmodified') $2 = insert($1, '20040614 11:29') delete($c, 'lastmodified[0]') $3 = insertBefore($c, 'item[0]', 'item') $4 = insert($3, ' ... ')

수행 순서에 따른 충돌 문제를 지적하고 이것을 해결하기 위한 4가지 로킹 프로토콜을 제안하고 있다. 4가지 로킹 프로토콜은 각각 다른 로킹 단위(locking granularity)를 가지고 있어서 다른 수준의 복잡도와 병행수행 성능을 제공한다. [7]에서 제안된 로킹 프로토콜은 노드들의 수직적 관계뿐 만 아니라 노드의 전후 순서와 같은 수평적 관계를 고려하고 있기 때문에 다음 절에 설명할 유효성 검증 기법들과 적절히 결합되어 사용될 수 있다. 하지만 기본적으로 [5,6]의 기법들은 유효 XML을 가정하고 있지 않기 때문에 유효성 검증 기법과 결합될 경우 상대적으로 낮은 병행수행 성능을 제공하는 Doc2PL, Node2PL을 사용할 수 밖에 없다.

[7]의 DGLOCK은 노드 테스트(node test)를 포함하는 XPath 표현에 대해서 노드 테스트 부분을 프레디킷 로크(predicate lock)로 사용함으로써 XML 트랜잭션의 병행수행 제어를 스키마 레벨에서 수행할 수 있도록 하는 기법이라고 할 수 있다. DGLOCK은 XML의 데이타 아이템에 접근하기 위한 XPath 경로 표현 부분에 대해서는 인텐션 로크(intension lock)를 사용하고 노드 테스트에 대해서는 프레디킷 로크를 사용함으로써 같은 경로에 대한 접근이라도 서로 다른 노드에 대해 접근하는 트랜잭션들 간에는 충돌이 발생하지 않도록 한다. 따라서 DGLOCK은 스키마 레벨에서 병행수행 제어를 함으로써 로킹의 오버헤드를 줄이면서 병행수행 성능을 높일 수 있도록 한다.

[8,9]의 경로 로킹(path locking)은 질의를 수행하는 트랜잭션에 대한 일관성 있는 결과를 보장하기 위해 어떤 질의 연산에 대해 그 연산의 XPath 표현과 일치하는 노드를 삽입/삭제하려는 다른 트랜잭션의 연산을 충돌하도록 하는 경로 로크(path lock)를 정의한다. 경로 로킹은 데이타 아이템에 대한 접근이라고 할 수 있는 XPath 표현을 로킹 정보로써 직접 사용함으로써 같은 경로에 대한 접근 간의 충돌을 감지하는 기법이다. 하지만 [8,9]의 연구는 XML에 대해서 노드 순서가 없는 데이타 모델을 사용하고 있기 때문에 path locking은 유효 XML과 같이 노드 간의 수평적 관계가 중요한 환경에서 트랜잭션의 병행수행 제어를 위해서는 한계를 가지고 있다.

기존의 XML 트랜잭션에 대한 병행수행 제어 기법들은 주로 데이타의 계층적 구조에 의한 트랜잭션 간의 충돌을 해결하기 위한 기법이라고 할 수 있다. 물론 [5,6]의 로킹 기법들은 노드들의 순서에 대해서 고려하고 있지만 단지 인접 노드와의 순서 관계만을 고려하고 있어 유효 XML과 같이 형제 엘리먼트 간의 유효 순서가 존재하는 환경에서는 역시 문제를 갖고 있다.

2.3 유효성 검증 기법

유효 XML은 각 엘리먼트나 애트리뷰트의 내용이 DTD(혹은 XML Schema, 본 논문은 DTD를 기준으로

함)에 의해서 정의된다. 따라서 유효 XML에 대한 변경은 DTD에 대한 유효성이 지켜질 때만 허용될 수 있어야 한다. 그래서 유효 XML을 변경하는 트랜잭션에 대해서 그 변경이 유효한 변경인지 검사함으로써 그 결과에 따라 트랜잭션을 승인하거나 취소할 수 있는 방법이 필요하다. 그런데 유효 XML에 대한 변경 유효성에 대한 검증은 엘리먼트 간의 순서 유효성에 대한 검증 때문에 연산 단위의 검증 자체가 불가능하다. 그것은 변경된 노드뿐 만아니라 그 주변의 다른 노드들과의 순서 정보가 필요하고 트랜잭션은 여러 개의 변경 연산들로 구성될 수 있기 때문이다.

변경 유효성을 검증하는 가장 간단한 방법은 트랜잭션에 의해 변경된 XML을 유효성 검증 파서(validation parser)를 사용하여 파싱(parsing)하는 것이다. 이러한 검증 기법은 변경의 부분에 대한 고려는 하지 않고 항상 문서 전체에 대한 검증을 수행하는 방법이기 때문에 낮은 검증 효율성을 가진다는 단점을 가지고 있다. 본 연구에서 다음에 설명할 부분 검증 기법과 반대되는 의미로 이 검증 기법에 대해 '전체 검증 기법'이라는 용어를 사용한다.

문서 전체에 대한 유효성 검증을 수행하는 기법이 가지는 비효율성을 개선하기 위해서 [10]에서는 즉시 부분 검증 기법을 제안하였다. 즉시 부분 검증 기법에서 즉시 검증은 트랜잭션이 하나의 연산으로 정의됐다는 가정을 기초로 하기 때문에, 본 연구에서는 부분 검증 기법만을 고려한다. 부분 검증 기법은 XML에서 노드들 간의 순서 관계는 형제 관계에 있는 노드들 사이에만 존재하는 지역적인 관계라는 사실에 기반으로 하여 변경된 노드와 형제 관계에 있는 노드들만 읽어 들여 순서 유효성을 검증하는 기법이다.

하지만 [10]의 부분 검증 기법은 pan-out 값이 큰 XML에 대해서는 효율성이 떨어질 수 있다는 문제를 가지고 있다. DTD의 엘리먼트 선언이 순환적인 정의를 포함하지 않는다면 XML 문서 트리는 DTD에 의해 고정된 깊이(depth)를 가지게 되고, 엘리먼트 선언의 카디널리티에 의해서 pan-out 값이 증가되는 크기 증가만이 가능하게 된다. 이렇게 수평 방향으로의 크기 증가만이 가능한 XML 문서가 데이타베이스에 저장되어 있고, 변경 유효성 검증을 위해 부분 검증 기법을 사용한다면 문서 크기 증가에 따른 검증 오버헤드는 갈수록 증가하게 된다. 특히 형제 노드들을 읽는 작업은 비교적 큰 비용(cost)이 드는 I/O 작업을 요구할 수 있으므로 이 경우 유효성 검증에 상당한 시간이 소요될 수 있어 시스템의 응답시간을 증가시키는 문제가 발생할 수 있다.

정리하면 부분 검증 기법은 전체 검증 기법과 비교하여 문서 용량이 큰 XML 문서에 대한 유효성 검증의 오버헤드를 줄임으로써 검증 효율성을 향상 시켰지만, 큰 pan-out 값을 가지는-수평 방향의 크기가 큰-XML에 대해서는 여전히 큰 검증 오버헤드를 가진다는 단점을 가지고 있다.

2.4 유효성 검증 기법과 병행수행 제어 기법

기존의 유효성 검증 기법의 또 다른 문제점 중의 하나는 유효성 검증이 트랜잭션의 병행수행 성능을 저하시킬 수 있다는 점이다. 이것은 기존의 병행수행 제어 기법들은 유효 XML에서의 순서 유효성과 같은 데이타 아이템들 사이의 수평적 관계에 대한 고려를 하지 않아 유효성 검증 기법과 함께 사용될 경우 원래의 병행수행 성능을 제공할 수 없기 때문이다.

정의 1. 순서 검증 정보 엘리먼트 : 변경을 수행한 트랜잭션의 변경 유효성을 검증하기 위해서는 그 트랜잭션에 의해 접근되지 않았더라도 순서 유효성 검증을 위해서 필요한 엘리먼트들이 있다. 이러한 엘리먼트를 **순서 검증 정보 엘리먼트**라 한다.

순서 검증 정보 엘리먼트는 유효성 검증을 위해 필요한 엘리먼트들을 의미한다. 이 순서 검증 정보 엘리먼트는 실제로 트랜잭션에 의해서 접근(access)된 데이타 아이템이 아니기 때문에 일반적으로 병행수행 제어에 의해서 로킹되지 않은 데이타 아이템들이다. 그러나 순서 검증 정보 엘리먼트는 변경 유효성 검증에 필요한 정보를 가지고 있는 데이타 아이템들이므로 유효성 검증이 정확하게 수행되기 위해서는 이 엘리먼트들이 다른 트랜잭션에 의해 삭제되거나 순서가 변경되지 않도록 보장하는 것이 필요하다. 즉, 유효성 검증 기법은 검증 정확성을 위해서 순서 검증 정보 엘리먼트들의 변경을 막을 수 있는 병행수행 제어 기법을 필요로 한다.

전체 검증 기법에서 순서 검증 정보 엘리먼트는 연산에 상관없이 XML 문서의 모든 엘리먼트라고 할 수 있다. 전체 검증 기법은 항상 문서 전체를 파싱해야 함으로 항상 문서의 모든 엘리먼트들을 필요로 한다. 따라서 전체 검증 기법은 검증 정확성을 위해서 문서 전체에 대한 로킹을 필요로 한다. 이러한 검증을 위한 병행수행 제어 기법으로는 낮은 병행수행 성능을 제공하는 [5,6]의 Doc2PL과 같은 병행수행 제어 기법을 사용할 수밖에 없다.

부분 검증 기법의 경우 순서 검증 정보 엘리먼트는 삽입/삭제된 엘리먼트의 모든 형제 엘리먼트들이 된다. 그래서 부분 검증 기법은 [5,6]의 Node2PL과 같이 변경되는 노드의 부모 노드에 로킹을 하는 기법에 의해서 검증 정확성을 보장받을 수 있다. 그러나 pan-out이 큰 XML의 경우 부모 노드 로킹은 병행수행 성능을 저하시킬 수 있다는 문제를 가지고 있으므로 부분 검증 기법과 마찬가지로 수평 방향으로 증가하는 XML에 대해

서 비효율적이다.

2.5 예제를 통해 살펴 본 문제점 분석

앞 절에서 설명한 바와 같이 순서 검증 정보 엘리먼트의 수는 유효성 검증의 효율성뿐 만 아니라 트랜잭션의 병행수행 성능에 큰 영향을 미치게 된다. 그리고 순서 검증 정보 엘리먼트는 변경 엘리먼트와 수평적 관계에 있는 데이타 아이템들이기 때문에 [7-9]와 같은 로킹 기법을 사용하는 것은 유효성 검증 기법의 정확성을 보장할 수 없게 된다. 다음과 같은 예제 환경을 생각해 보자.

예제 환경 : Site Syndicate Aggregator

여러 사이트에서 RSS를 수집하여 사이트들의 컨텐츠(contents)들에 대한 통합 서비스를 제공하는 SSA(Site Syndicate Aggregator)라는 웹 서비스를 가정하자. SSA 서비스는 등록된 사이트들의 컨텐츠에 대해서 사용자들에게 최근 업데이트된 컨텐츠 목록을 제공하거나 컨텐츠에 대한 검색 서비스를 지원한다. 이때 RSS는 XML 형식의 문서이고 SSA가 웹 서비스의 형태로 제공되기 때문에 SSA의 데이타 관리를 위해 XML 데이타베이스를 사용하는 것은 적절한 선택이 될 수 있다.

일반적으로 RSS는 최근 업데이트된 컨텐츠 정보만을 제공하기 때문에 SSA 서비스는 검색 서비스 지원을 위해서 과거 컨텐츠 정보를 유지하고 있어야 한다. 따라서 SSA 서비스를 위한 XML 데이타는 각 사이트들에서 수집한 RSS와 쉽게 병합(merge)될 수 있도록 그림 1과 같이 RSS와 유사한 형태로 정의할 수 있다. 여기에서 hit와 rank 엘리먼트는 SSA에서 각 사이트에 대한 사용자 접근 빈도 정보를 유지하기 위해 정의된 엘리먼트들이다.

그리고 SSA 서비스는 그림 2와 같은 작업들로 구성될 수 있으며, 이러한 작업들은 통합 RSS에 대한 질의 및 변경 트랜잭션을 수행함으로써 XML 데이타베이스에 접근하게 된다. 그림 3은 통합 RSS에 대해 접근하는 트랜잭션의 예를 보이고 있다. 그림 3의 T1, T2, T3

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

그림 1 통합 RSS를 위한 DTD

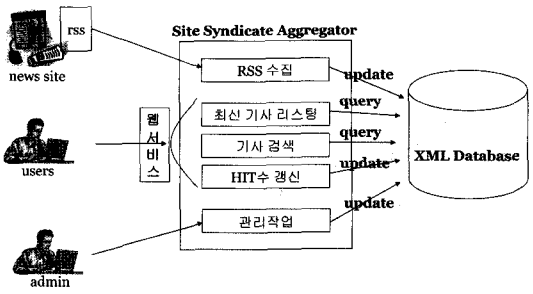

그림 2 SSA 서비스 구성

```
T1 :
  FOR $i IN /rss/channel/item[0]
  RETURN $i
T2 :
  LET $c := /rss/channel[0]
  UPDATE $c {
    REPLACE $c/lastmodified
      WITH <lastmodified> 20040614 11:29 </lastmodified>
    INSERT <item> ... </item> BEFORE $c/item[0]
  }
T3 :
  LET $c := /rss/channel[0]
  UPDATE $c {
    REPLACE $c/hit WITH <hit> 321340 </hit>
  }
```

그림 3 SSA 서비스에서 XML 트랜잭션의 예

는 각각 최신 기사 리스팅, RSS 수집 및 병합, Hit 수 갱신 작업에 대응된다.

각 사이트들에서 제공하는 컨텐츠가 증가함에 따라 통합 RSS의 channel 엘리먼트 밑의 item 엘리먼트의 수가 증가하게 된다. 이것은 통합 RSS 파일은 시간이 지남에 따라 channel 엘리먼트들의 pan-out 값이 증가한다는 것을 의미하고 수평 방향으로 크기가 증가하는 XML 문서라는 것을 의미한다. 또한 통합 RSS 파일에 접근하는 트랜잭션이 올바르게 수행되기 위해서는 통합 RSS의 DTD 유효성이 항상 보장될 수 있어야 하기 때문에 통합 RSS는 변경 트랜잭션들에 대한 유효성 검증이 필요한 유효 XML 문서라고 할 수 있다.

이러한 환경에서 그림 3과 같이 정의된 트랜잭션들이 병행수행될 때 전체 검증 기법 및 Doc2PL 부분 검증 기법 및 Node2PL 방법이 어떤 문제를 가지고 있는지 살펴보자.

전체 검증 기법 및 Doc2PL : 전체 검증 기법은 T2, T3 트랜잭션과 같은 변경 트랜잭션의 유효성을 검증하기 위해서 문서 전체에 대한 파싱을 해야 한다. 그리고 유효성 검증이 병행수행되는 트랜잭션들에 의해 영향을 받지 않도록 하기 위해 Doc2PL과 같은 문서 전체에 대한 로킹을 하는 병행수행 제어 기법을 필요로 한다. 따

라서 이 경우 T2, T3 트랜잭션은 다른 트랜잭션과 병행수행될 수 없게 된다.

부분 검증 기법 및 Node2PL : 부분 검증 기법은 변경된 엘리먼트의 형제 엘리먼트들만 읽기 때문에 형제 엘리먼트들에 대한 로킹을 필요로 한다. XML의 계층적 구조를 이용하여 변경된 노드의 부모 노드에 로킹을 하는 Node2PL를 사용하여 부분 검증 기법의 정확성을 보장할 수 있다. 그러나 T1, T2, T3가 모두 첫 번째 channel 엘리먼트의 자식 엘리먼트에 접근하는 트랜잭션들이기 때문에 Node2PL에 의해 세 트랜잭션들 모두 첫 번째 channel 엘리먼트에 대한 로크를 요구하게 되어 병행수행이 불가능해진다.

전체 검증 기법과 부분 검증 기법의 정확성을 보장하기 위한 Doc2PL, Node2PL은 T1, T2, T3 트랜잭션을 병행수행 시킬 수 없음을 보였다. 세 트랜잭션은 모두 다른 데이타 아이템에 대해서 접근하는 트랜잭션들이며 DTD의 엘리먼트 선언을 고려하더라도 직접적인 순서 관계에 있는 엘리먼트에 접근하지 않는다. 하지만 이 예제 환경에서는 유효성 검증 기법의 정확성 보장 때문에 의미적으로 불필요한 충돌이 발생하게 되는 병행수행 제어기법을 사용할 수밖에 없다.

따라서 기존의 기법보다 더 적은 수의 순서 검증 정보 엘리먼트를 가지고 순서 유효성을 검증할 수 있는 유효성 검증 기법이 필요하다. 보다 적은 수의 순서 검증 정보 엘리먼트만으로 유효성 검증이 가능하다면 유효성 검증의 효율성을 높일 수 있을 뿐만 아니라, 보다 나은 성능을 제공할 수 있는 병행수행 제어 기법을 사용할 수 있을 것이다.

3. 시퀀스 그룹 검증 기법 및 경계 로킹

3.1 엘리먼트 시퀀스 그룹

부분 검증 기법은 순서 검증 정보 엘리먼트들을 엘리먼트들의 형제 관계를 이용하여 정의하였다. 이것보다 적은 수의 순서 검증 정보 엘리먼트만으로 유효성 검증을 수행하기 위해서는 형제 관계보다 더 작은 범위의 관계를 이용하여 엘리먼트들을 그룹화시킬 필요가 있다. 그래서 본 연구에서는 엘리먼트 시퀀스 그룹을 정의한다.

정의 2. 엘리먼트 시퀀스 그룹 : DTD의 엘리먼트 선언에서 순차 내용 모델(sequence content model)만으로 묶여 있는 엘리먼트들을 엘리먼트 시퀀스 그룹, 혹은 시퀀스 그룹이라고 한다.

DTD의 엘리먼트 선언은 순차 내용 모델, 선택 내용 모델(choice content model), 카디널리티(cardinality)를 사용하여 정의된다. 여기에서 순차 내용 모델은 순수하게 엘리먼트들의 순서만을 정의한다. 반면 선택 내용 모델과 카디널리티는 조건과 갯수를 정의하게 된다. 따라서 순차 내용 모델에 의해서만 정의된 엘리먼트 시퀀스 그룹은 가장 작은 단위의 엘리먼트 순서 정보를 포함하고 있다고 할 수 있다. 그림 4는 channel 엘리먼트의 자식 엘리먼트들을 DTD의 엘리먼트 선언에 따라 엘리먼트 시퀀스 그룹으로 그룹화하는 예를 보이고 있다. DTD의 엘리먼트 선언에서 순차 내용 모델로만 정의된 엘리먼트들의 그룹은 [title, link, description, lastmodified], [author], [editor], [hit, rank], [item] 들이다. 이것들은 channel 엘리먼트가 자식으로 가질 수 있는 엘리먼트 시퀀스 그룹들의 종류들이다.

XML 문서에서 형제 관계에 있는 엘리먼트들을 DTD의 엘리먼트 시퀀스 그룹으로 맵핑시키면 엘리먼트 순서에 대한 유효성 검증은 엘리먼트 시퀀스 그룹들의 선택(choice) 및 개수(cardinality) 유효성 문제로 환원된다. 따라서, 엘리먼트들이 유효한 순서를 가지고 있는지 검사하는 것은 각 엘리먼트 시퀀스 그룹의 엘리먼트 순서에 대한 유효성과 엘리먼트 시퀀스 그룹들의 순서에 대한 유효성을 검사하는 두 부분으로 나누어질 수 있다.

엘리먼트 시퀀스 그룹의 정의에 의해 엘리먼트 시퀀스 그룹 내의 엘리먼트 순서는 오직 순차 내용 모델만으로 정의되어 있다. 따라서, 엘리먼트들의 순서를 DTD의 엘리먼트 시퀀스 그룹과 단순히 비교함으로써 엘리먼트들의 순서 유효성은 쉽게 검증될 수 있다. 또한 이러한 비교는 모든 형제 엘리먼트 순서를 다 검사해야 하는 부분 검증의 방법보다는 더 적은 수의 엘리먼트들

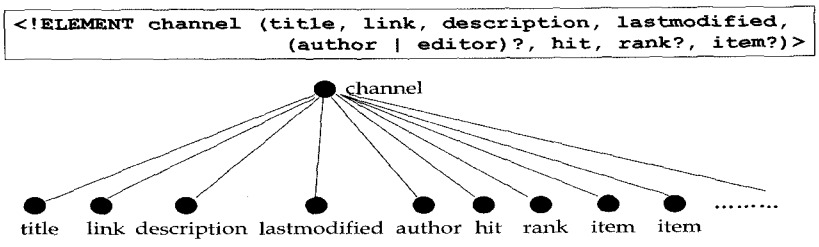

그림 4 엘리먼트 시퀀스 그룹

의 순서만을 검사할 수 있으므로 오버헤드가 적다. 예를 들어 [title, link, description, lastmodified]을 엘리먼트 시퀀스 그룹 Group1이라 하고 이 그룹의 엘리먼트를 변경하는 트랜잭션이 다음과 같다고 하자.

```
Transaction T1 :
        update channel {
                Replace title With ... ,
                Replace title With ... ,
        }
```

이 예에서 T1은 승인(commit)되는 시점에 엘리먼트 시퀀스 그룹 Group1의 엘리먼트 순서에 대해서만 검증을 수행함으로써 트랜잭션 유효성을 검증할 수 있게 된다.

트랜잭션에 의한 XML의 변경은 엘리먼트 시퀀스 그룹의 삭제나 삽입을 포함할 수 있다. 이러한 경우 엘리먼트 시퀀스 그룹의 엘리먼트 순서뿐만 아니라 엘리먼트 시퀀스 그룹들 간의 순서도 변할 수 있게 된다. 따라서, 엘리먼트 시퀀스 그룹 간의 순서 유효성 역시 검증될 필요가 있다. 엘리먼트 시퀀스 그룹들 간의 순서는 오직 선택 내용 모델과 카디널리티에 의해서만 정의된다. 이것은 엘리먼트 시퀀스 그룹들 간의 순서 유효성이 인접한 엘리먼트 시퀀스 그룹만을 살펴봄으로써 검증될 수 있다는 것을 의미한다. 선택 내용 모델로 정의된 엘리먼트 시퀀스 그룹들은 다른 어떤 것들도 함께 XML 문서에 나타날 수 없고, 그 중에서 오직 하나의 엘리먼트 시퀀스 그룹만이 나타날 수 있다. 따라서, 인접한 두 엘리먼트 시퀀스 그룹이 DTD에서의 카디널리티는 Zero, One, Many만을 표현할 수 있기 때문에 인접한 두 엘리먼트 시퀀스 그룹을 포함한 연속적인 3개의 엘리먼트 시퀀스 그룹만을 살펴봄으로써 엘리먼트 시퀀스 그룹의 카디널리티 유효성을 검증해 낼 수 있다. 그러므로 엘리먼트 시퀀스 그룹 단위로 트랜잭션의 변경 유효성을 검증하는 것은 변경된 엘리먼트 시퀀스 그룹의 엘리먼트 순서가 유효한지 검증하고, 인접한 엘리먼트 시퀀스 그룹이 유효한 인접 엘리먼트 시퀀스 그룹인지만 검증하면 되기 때문에 변경된 엘리먼트의 모든 형제 엘리먼트 순서를 검증해야 하는 부분 검증 기법보다 더 적은 범위의 유효성 검증 범위를 가질 수 있게 된다. 그리고 하나 이상의 엘리먼트로 구성되는 엘리먼트 시퀀스 그룹들은 하나의 변경 연산(insert or delete)만으로는 유효한 엘리먼트 순서를 가질 수 없기 때문에 같은 트랜잭션에 의해 추가적인 변경 연산이 수행될 것이라고 기대할 수 있다. 이것은 유효 XML 환경에서는 트랜잭션의 변경이 엘리먼트 시퀀스 그룹 단위로 이루어지도록 하는 것이 예상되는 트랜잭션들의 충돌을 막을 수 있음을 의미한다. 따라서, 엘리먼트 시퀀스 그룹을 병행수행 제어의 단위로 사용하는 것은 모든 형제 노드를 로킹하는 것보다 병행수행 성능을 향상시킬 수 있을 뿐만 아니라 의미적으로도 유효 XML에 대해 효과적인 변경이 가능하도록 한다.

3.2 엘리먼트 그룹화 기법

트리 형태의 자료구조는 부모-자식 관계와 같은 노드들의 종적 관계가 주로 고려되기 때문에 엘리먼트 시퀀스 그룹과 같은 독특한 횡적 관계에 있는 노드들을 관리하고 구분하기 힘들다는 문제를 가지고 있다. 따라서, XML 문서에서 형제 관계에 있는 엘리먼트들을 엘리먼트 시퀀스 그룹으로 그룹화하기 위해서는 특별한 방법이 필요하다.

엘리먼트 시퀀스 그룹들은 DTD의 엘리먼트 선언에 의해 정의된다. 따라서, DTD의 엘리먼트 선언을 사용하여 엘리먼트들을 시퀀스 그룹으로 그룹화한다. [10]에서는 엘리먼트의 선언에 대한 DFA(deterministic Finite Automata)를 구성하고 있는데 유효성 검증의 용도로만 사용하고 있다. 본 논문은 이를 확장하여 엘리먼트들을 엘리먼트 시퀀스 그룹으로 그룹화할 수 있도록 하기 위해 그룹화된 상태들로 정의된 SG-DFA를 제안한다.

정의 3. SG-DFA(State Group DFA) : 일반적인 DFA와 같이 alphabet, transition function, state set, initial state, final state로 구성된다. 하지만 state가 (group-id, sequence-id) 쌍으로 정의된다.

DTD의 각 엘리먼트 선언에 대해서 SG-DFA를 구성하고 각 엘리먼트들에 의한 SG-DFA의 전이 특성에 따라 엘리먼트들을 그룹화하게 된다. 그림 5는 channel 엘리먼트에 대한 SG-DFA의 예를 보이고 있다. 여기서 각 상태들의 라벨 중 첫 번째 숫자는 Group ID를 나타내며 두 번째 숫자는 Sequence ID를 나타낸다. 초기 상태(0,0)에서 channel 엘리먼트 선언에서 처음에 오는 엘리먼트는 title이라 이를 SG-DFA의 에지 레이블로 놓고 상태(1,1)로 전이한다. 그 다음에 오는 link, description, lastmodified 엘리먼트는 시퀀스 그룹을 이루기 때문에 같은 Group ID를 갖고 Sequence ID만 하나씩 증가하게 되어 상태 (1,4)까지 전이한다. 다음에 오는 엘리먼트는 (author|editor)?이기 때문에 세 갈래로 분기한다(?의 의미는 없거나 하나 존재할 수 있는 카디널리티). 따라서, 없을 때는 바로 hit 엘리먼트가 나올 수 있고, 하나 존재하는 경우는 author가 존재하거나 editor가 존재할 수 있기 때문에 이들은 각각 다른 Group ID를 갖게 된다. 그 다음에 오는 엘리먼트가 (rank)?이기 때문에 마찬가지로 없거나 하나 존재하는 카디널리티에 의해 두 갈래로 분기하고 각각 다른 Group ID를 갖게 된다. 마지막으로 item*에 대한 DFA는 재귀적인 순환이 나온다.

이제 SG-DFA를 사용하여 엘리먼트들을 엘리먼트

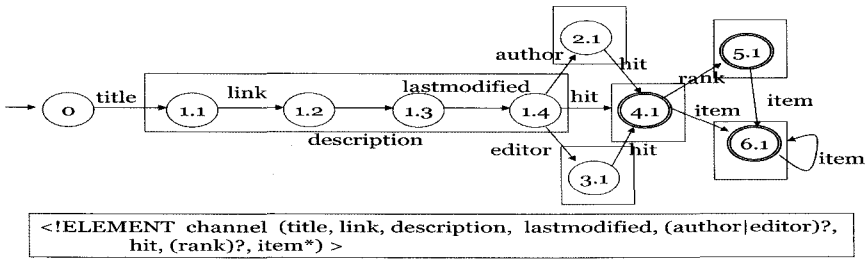

그림 5 channel 엘리먼트에 대한 SG-DFA의 예

시퀀스 그룹으로 분류하는 방법에 대해서 설명한다. SG-DFA에서 엘리먼트 스트링에 대해서 상태 전이 시 상태의 Group ID가 변경되었다면 그 전이에 대한 입력 이었던 엘리먼트부터 새로운 엘리먼트 시퀀스 그룹이 시작된다는 것을 의미한다. 하지만 카디널리티를 가지는 순차 내용 모델의 경우에는 새로운 엘리먼트 시퀀스 그룹이 시작되더라도 DTD 상에서는 같은 그룹이기 때문에 Group ID가 변하지 않을 수 있다. 이때는 Sequence ID를 사용하여 새로운 엘리먼트 시퀀스 그룹의 시작을 감지할 수 있다.

그림 6은 SG-DFA를 사용하여 엘리먼트들을 시퀀스 그룹으로 그룹화 하는 예를 보이고 있다. editor 엘리먼트가 입력되면 상태는 1.4에서 3.1의 상태로 전이된다. 이때 Group ID가 1에서 3으로 변하였기 때문에 editor는 앞쪽 엘리먼트들과 다른 엘리먼트 시퀀스 그룹에 속하게 된다. 그리고, 첫 번째 item 엘리먼트가 읽혔을 때도 상태 4.1에서 6.1로 전이되어 상태의 Group ID가 변했기 때문에 새로운 시퀀스 그룹의 시작을 알 수 있다. 반면 두 번째 item을 읽는 경우 상태가 6.1에서 6.1로 변하지 않으므로 동일 상태 그룹에서의 전이라 판단될 수 있다. 하지만 이때 Sequence ID가 증가하지 않고 1로 그대로 유지되기 때문에 카디널리티에 의한 시퀀스 그룹의 시작을 감지해 낼 수 있게 된다.

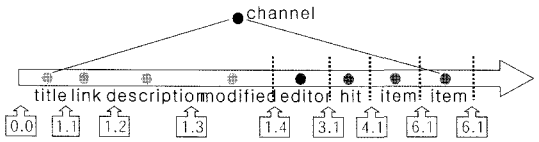

그림 6 SG-DFA를 이용한 엘리먼트의 시퀀스 그룹화

3.3 시퀀스 그룹 검증

시퀀스 그룹 단위의 검증을 위해서는 다음과 같은 두 가지 문제가 선결되어야 한다. 첫 번째는 시퀀스 그룹을 삭제하거나 새로운 시퀀스 그룹을 삽입하는 경우 그 시퀀스 그룹에 대한 순서 유효성 검증이 쉽지 않고, 또한 트랜잭션들 간의 시퀀스 그룹 삽입/삭제의 순서에 따른 충돌이 발생할 수 있다는 점이다. 두 번째 문제는 시퀀스 그룹이 트리 구조에 의해서 정의된 노드 그룹이 아니라 DTD에 의해서 정의된 그룹이기 때문에 SG-DFA를 이용한 그룹화 기법이 필요하게 되는데 이러한 오버헤드를 최소화 할 수 있어야 한다는 점이다.

첫 번째 문제는 정리 1을 사용하여 시퀀스 그룹 검증을 정의함으로써 해결될 수 있다.

정리 1. 스트링 $I=(e_1e_2...e_x\ \sigma\ e_ye_{y+1}...e_n)$가 어떤 $DFA\ M$에 의해 승인될 때 상황 $[Q_x,\ \sigma\ e_y...e_n]$과 $[Q_y,\ e_{y+1}...e_n]$이 발생한 후 승인상태에 도달한다고 하자. 그리고 스트링 $I'=(e_1e_2...e_x\ \sigma'\ e_y...e_n)$에 대해 I'의 서브스트링 $(\sigma'\ e_y)$가 M에 대해 상태 Q_x를 Q_y로 전이시킬 수 있다면 스트링 I'도 M에 의해 승인되는 스트링이다.

증명. DFA에서는 모든 전이함수는 오직 하나의 상태만을 결과로 갖기 때문에 M은 입력 스트링 I'에 대해서도 상황 $[Q_x,\ \sigma'\ e_y...e_n]$이 발생할 것이 분명하다. 마찬가지로 I'에 의해 M의 상황 $[Q_y,\ e_{y+1}...e_n]$이 발생한다면 I'는 M에 의한 승인상태에 도달할 수 있다. 따라서 I'가 M을 $[Q_x,\ \sigma'\ e_y...e_n]$에서 $[Q_y,\ e_{y+1}e_n]$으로 이동시킬 수 있다면 I'는 M에 의해 승인될 수 있다.

정리 1은 변경되는 시퀀스 그룹과 인접한 엘리먼트들과 그 엘리먼트들에 의한 SG-DFA의 전이 상태를 사용하여 변경되는 부분에 대한 유효성 검증이 가능하다는 것을 보이고 있다. 새로운 시퀀스 그룹의 삽입은 σ이 빈 스트링(empty string)일 경우를 의미하고 시퀀스 그룹의 삭제는 σ'가 빈 스트링일 경우를 의미한다. 따라서 정리 1을 이용하면 시퀀스 그룹의 삽입/삭제뿐만 아니라 어떤 형태의 변경에 대해서도 적은 순서 검증 정보 엘리먼트만으로 순서 유효성을 검증할 수 있게 된다. 또한 시퀀스 그룹 검증 정의 4와 같이 정의된다.

정의 4. 시퀀스 그룹 검증 : 어떤 엘리먼트 Ep의 자식인 엘리먼트 시퀀스 그룹 SGx가 두 엘리먼트 시퀀스 그룹 SGy, SGz와 SGy-SGx-SGz의 순서로 인접해 있고 트랜잭션 Tx에 의해 SGx'로 변경되었다고 할 때 다음과 같이 Tx의 SGx'에 대한 변경 유효성을 검증하는 기법을 시퀀스 그룹 검증 기법이라 한다. (Ep에 대한

*DFA*는 *M*이라 하자)

*i) SGx*가 *Tx*에 의해 변경되기 전에 *SGy*의 마지막 엘리먼트 *ey*에 의한 *M*의 전이 상태 *Qy*와 *SGz*의 첫 엘리먼트 *ez*에 의한 *M*의 전이 상태 *Qz*를 저장한다.

ii) *Tx*가 모든 연산을 다 수행한 후 승인(commit)될 때 *SGx'*의 모든 엘리먼트와 *ez*에 의해 *M*이 상태 *Qy*에서 *Qz*로 전이될 수 있는 지 검사한다. 전이할 수 있다면 *Tx*가 수행한 변경 $SGx \mapsto SGx'$은 유효한 변경이다.

시퀀스 그룹 검증 기법은 정리 1에 의해 정확한 결과를 보장하는 유효성 검증을 수행할 수 있다. 만약 트랜잭션 T_x가 엘리먼트 시퀀스 그룹 SG_x의 엘리먼트들을 모두 삭제한 경우 e_z라는 엘리먼트에 대해 상태 Q_y에서 Q_z로의 전이가 정의되었는지 검사함으로써 T_x의 변경 유효성을 검증할 수 있다. 또한 T_x가 새로운 엘리먼트 시퀀스 그룹들을 삽입한 경우라면 SG_x를 빈 엘리먼트 시퀀스 그룹으로 보고 SG_x'와 e_z에 의한 상태 전이를 검사함으로써 유효성 검증을 수행할 수 있다. 즉, 시퀀스 그룹 검증 기법은 엘리먼트 시퀀스 그룹의 삽입이나 삭제와 같이 엘리먼트 시퀀스 그룹 간의 순서가 변경되는 경우에도 정확한 엘림너트 순서 유효성 검증을 수행할 수 있다. 그림 7은 시퀀스 그룹 검증 알고리즘을 표현하고 있다.

시퀀스 그룹 검증 기법의 검증 범위는 변경이 수행된 엘리먼트 시퀀스 그룹이라고 할 수 있다. 엘리먼트 선언들은 일반적으로 다수의 엘리먼트 시퀀스 그룹들로 구성되기 때문에 대부분의 경우 시퀀스 그룹 검증 기법의 검증 범위는 부분 검증 기법의 검증 범위보다 더 작은 범위를 가진다고 할 수 있다. 이것은 유효성 검증에 대한 정확성을 위해 로킹되는 노드의 수를 줄여 트랜잭션들의 병행수행 성능을 향상시킬 수 있음을 의미한다.

앞에서 지적한 두 번째 문제는 경계 로킹 기법에 의해서 해결된다. 이것은 노드와 노드에 대한 SG-DFA의 상태 정보가 트랜잭션에 의해서 접근될 때 획득되고 이 정보를 로킹 정보에 포함시킴으로써 반복적인 시퀀스 그룹 판별 작업을 최소화 시킨다. 그리고 경계 로킹 기법은 트리 구조와 적합하지 않은 시퀀스 그룹을 로킹의 단위로 사용함으로써 시퀀스 그룹 검증의 검증 유효성을 보장하면서도 높은 병행수행 성능을 보장할 수 있다.

```
Input : 검증할 시퀀스 그룹에 대한 경계 로크가 걸린 두 노드 SBNode, EBNode
        두 노드에 의한 전이 상태 SBState, EBState
Output : 유효하면 "yes", 아니면 "No"를 리턴
Steps :
CurrentState := SBState;
for Node each All Nodes between SBNode and EBNode
    CurrentState := NextState(CurrentState, Node);
    if CurrentState == UndefinedState then return "no";
    end if
end
CurrentState := NextState(CurrentState, EBNode);
if CurrentState == EBState then return "yes";
else return "no";
```

그림 7 시퀀스 그룹 검증 알고리즘

3.4 경계 로킹 기법

경계 로킹은 시퀀스 그룹 검증 기법과 밀접한 관련을 가지고 있다. 경계 로킹은 시퀀스 그룹 검증의 정확성을 보장할 뿐만 아니라 유효성 검증에 필요한 정보를 제공하는 역할을 한다.

정의 5. 경계 엘리먼트 : 어떤 엘리먼트 시퀀스 그룹 SGx에 대해서 SGx에 인접하는 엘리먼트를 **경계 엘리먼트**라 한다. 그리고 인접하는 위치에 따라 SGx의 왼쪽에 인접하는 엘리먼트를 **시작 경계 엘리먼트**, 오른쪽에 인접하는 엘리먼트를 **끝 경계 엘리먼트**라 한다.

정의 6. 경계 로킹 기법 : 경계 로크는 변경 연산 insert와 delete에 의해 생성되며 연산이 수행되기 전에 변경될 엘리먼트에 대해 다음과 같은 절차를 통해 경계 로크와 배타 모드 로크를 설정한다. 다음의 절차는 하나의 로크를 설정하는 것처럼 원자적(atomic)으로 수행되도록 보장되어야 한다.

i) SG-DFA를 이용하여 변경되는 엘리먼트 시퀀스 그룹의 경계를 알아낸다.

ii) 변경되는 엘리먼트 시퀀스 그룹에 인접하는 두 엘리먼트에 시작 경계 로크와 끝 경계 로크를 설정한다. 시작/끝 경계 로크는 로킹의 대상 엘리먼트에 의한 DFA의 전이 상태 정보를 가진다.

iii) 두 경계 로크 사이에 있는 모든 엘리먼트들-즉 변경되는 엘리먼트 시퀀스 그룹의 엘리먼트들-에 배타 모드 로크를 건다.

경계 로킹의 절차 중 iii)은 시퀀스 그룹 검증의 i)에 해당하는 역할을 수행함으로써 검증에 필요한 정보를 미리 획득한다. 경계 로킹의 영역이 교차되거나 중첩되는 것을 막고, 경계 엘리먼트가 다른 트랜잭션들에 의해 변경되는 것을 막기 위해서 경계 로크는 다음과 같은 로크 충돌 규칙을 갖는다.

1. 다른 트랜잭션의 배타 모드 로크와 경계 로크는 충돌한다.
2. 다른 트랜잭션의 경계 로크와 같은 타입의 경계 로크는 충돌한다.

위의 1, 2 상황을 제외한 다른 상황에서 경계 로크는 충돌을 일으키지 않는다. 이것은 경계 로크가 설정된 엘리먼트에 대해서 다른 트랜잭션이 공유모드의 로크를 설정하는 것이 가능하다는 것을 의미한다. 그리고 배타 모드 로크와의 충돌은 변경되는 엘리먼트 시퀀스 그룹과 인접한 엘리먼트 시퀀스 그룹에 대한 다른 트랜잭션의 변경을 허용하지 않는다는 것을 의미한다. 그림 8은

```
Input : 변경을 수행하는 트랜잭션 UpdateTransaction,
        변경될 시퀀스 그룹과 인접한 시작 경계 노드 SBNode,
        변경될 시퀀스 그룹과 인접한 끝 경계 노드 EBNode
Output : 경계 로킹이 가능하면 로크를 설정하고 "yes"를 리턴, 그렇지 않으면 "no" 리
         턴
Steps:
/* 경계 로킹이 가능한지 검사 */
if other transaction has x-lock for SBNode, EBNode
      then return "no";
elsief other transaction has SB-lock for SBNode
      then return "no";
elsief other transaction has EB-lock for EBNode
      then return "no";
end if
/* 경계 로크가 확장되어야 하는지 검사 */
if UpdateTransaction has EB-lock or X-lock for SBNode
      then set X-lock for SBNode
else
      set SB-lock for SBNode
end if
/* SBNode와 EBNode 사이의 노드들에 X-lock */
for Node each All Nodes between SBNode and EBNode
      set X-lock for Node
end
return "yes"
```

그림 8 경계 로킹 알고리즘

경계 로킹 알고리즘을 설명하고 있다.

3.5 시퀀스 그룹 검증과 경계 로킹의 사용 예

앞 절에서 설명한 바와 같이 시퀀스 그룹 검증과 경계 로킹은 서로 긴밀하게 연결되어 있다. 시퀀스 그룹 검증과 경계 로킹이 그림 3에서 정의된 트랜잭션들의 병행수행과 유효성 검증을 어떻게 수행하는 지 살펴보자. 그림 9는 그것에 대한 예를 보이고 있다.

시퀀스 그룹 검증 : 트랜잭션 T2는 두 시퀀스 그룹에 대한 변경을 수행하는 트랜잭션이다. 그리고, 두 시퀀스 그룹은 인접해 있지 않기 때문에 T2는 두 개의 경계 로크 쌍을 가진다. T2에 대한 유효성 검증은 이 두 경계 로크의 사이에 있는 엘리먼트들에 대해서만 수행된다. 즉, (title~author) 엘리먼트들에 의해서 초기 상태 (0,0)에서 상태 (2,1)로 전이되는 지를 검증하고, (item, item)에 의해서 상태 (5,1)에서 상태 (6,1)로 전이되는 지를 검증한다. 만약 item 엘리먼트의 수가 많다면 시퀀스 그룹 검증 기법에 비교해서 검증 오버헤드를 상당히 줄일 수 있을 것이다.

경계 로킹 기법 : 전체 검증 기법과 부분 검증 기법을 위한 Doc2PL과 Node2PL 하에서는 세 트랜잭션들이 병행수행 되지 못했지만, 경계 로킹 기법이 사용되는 경우 T1, T2, T3에 의한 로크들은 서로 충돌을 일으키지 않기 때문에 병행수행이 가능하다. 또한, T2와 T3는 서로 다른 시퀀스 그룹을 변경하고, T1에 의한 S-lock과 T2에 의한 B-lock이 서로 호환되기 때문에 T1, T2 그리고 T3는 동시에 수행될 수 있다. 따라서, 경계 로킹 기법은 기존의 유효성 검증 기법과 관련된 로킹 기법과 비교해서 더 나은 병행수행 성능을 가지게 됨을 알 수 있다.

예제 환경의 통합 RSS와 같이 문서의 크기가 횡적으로 증가하고 비교적 큰 pan-out 값을 가지며 동일한 엘리먼트 레벨에 대한 접근이 많은 환경에서는 시퀀스 그룹 검증 및 경계 로킹 기법은 부분 검증 기법 및 형제 노드 로킹에 비해 더 나은 검증 효율성과 병행수행 성능을 보장할 수 있게 되어 각 트랜잭션은 짧은 응답 시간을 가지게 될 것이다.

4. 성능 평가

4.1 시간 및 공간 복잡도

전체 검증 기법/Doc2PL, 부분 검증 기법/Node2PL, 그리고 시퀀스 그룹 검증/경계 로킹의 성능을 비교하기 위해 시간 복잡도 및 공간 복잡도를 계산해 본다. 하나의 변경 연산이 수행되는 경우의 복잡도를 계산하며 그에 대한 변수는 다음과 같이 정의한다.

N: XML 문서의 전체 노드 수

depth: 변경되는 노드의 XML 문서 트리에서의 깊이

sibling: 변경되는 노드의 형제 노드들의 수

sg: XML 문서에서 엘리먼트가 자식으로 가지는 평균 엘리먼트 시퀀스 그룹의 수

전체 검증 기법 / Doc2PL : 전체 검증 기법은 문서 전체의 노드들을 모두 읽어야 하기 때문에 O(N)의 시간 복잡도를 갖는다. 그리고 Doc2PL은 문서 단위의 로

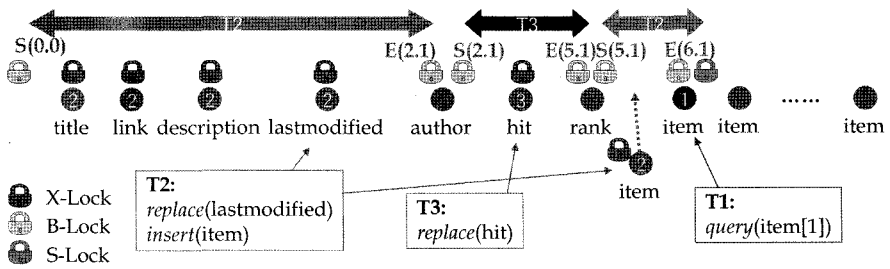

그림 9 시퀀스 그룹 검증과 경계 로킹

킹이므로 상수 시간 내에 로크를 설정할 수 있으며, 추가적인 공간을 요구하지도 않는다. 따라서 Doc2PL의 시간 및 공간 복잡도는 O(1)이다.

부분 검증 기법 / Node2PL : 부분 검증 기법은 변경되는 노드의 모든 형제 노드를 읽어야 하므로 O(sibling)의 시간 복잡도를 갖는다. 그리고 Node2PL은 부모 노드에 대한 로킹을 하므로 O(depth)의 시간 및 공간 복잡도를 갖게 된다.

시퀀스 그룹 검증 / 경계 로킹 : 시퀀스 그룹 검증은 변경되는 엘리먼트 시퀀스 그룹들에 대해서만 순서 유효성을 검사함으로 평균 sibling/sg 개의 엘리먼트들을 읽게 된다. 따라서 검증의 시간 복잡도는 O(sibling/sg)가 된다. 경계 로킹은 로크를 설정하기 위해서 변경 노드까지의 경로 외에도 엘리먼트 시퀀스 그룹의 경계를 계산하기 위해 최악의 경우 모든 형제 노드들을 읽어야 하므로 O(d+sibling)의 시간 복잡도를 가진다. 그리고 경로에 있는 엘리먼트와 두 경계 엘리먼트, 그리고 엘리먼트 시퀀스 그룹들에 대해서 로크를 설정하게 되므로 경계 로킹의 공간 복잡도는 O(d+sibling/sg)이 된다.

시간 및 공간 복잡도 비교에서 시퀀스 그룹 검증 기법은 가장 좋은 성능을 가진다는 것을 확인할 수 있었다. 하지만 경계 로킹의 경우 트리 구조에서의 엘리먼트 시퀀스 그룹을 설정하고 시퀀스 그룹 검증을 위한 상태 정보 획득과 같은 다소 복잡한 작업을 수행하기 때문에 다른 로킹 기법들보다 높은 시간/공간 복잡도를 보이고 있다. 하지만 경계 로킹은 로킹되는 노드의 수를 줄일 수 있기 때문에 더 나은 병행수행 성능을 보장할 수 있다. 다음 절의 실험은 경계 로킹에 의한 병행수행 성능 향상을 보이고 있다.

4.2 성능 실험

시간 및 공간 복잡도 비교에서 시퀀스 그룹 검증 기법은 가장 좋은 성능을 가진다는 것을 확인할 수 있었다. 하지만 경계 로킹의 경우 트리 구조에서의 엘리먼트 시퀀스 그룹을 설정하고 시퀀스 그룹 검증을 위한 상태 정보 획득과 같은 다소 복잡한 작업을 수행하기 때문에 다른 로킹 기법들보다 높은 시간/공간 복잡도를 보이고 있다. 하지만 경계 로킹은 로킹되는 노드의 수를 줄일 수 있기 때문에 더 나은 병행수행 성능을 보장할 수 있다. 다음 절의 실험은 경계 로킹에 의한 병행수행 성능 향상을 보이고 있다.

제안하는 시퀀스 그룹 검증 기법 및 경계 로킹 기법과 기존의 방법들의 성능을 측정하기 위해서 다음과 같은 환경에서 실험을 수행하였다.

XML Document : 그림 1의 DTD로 정의되는 통합 RSS

XML Transaction : 그림 3에 정의된 질의 및 변경 트랜잭션

Operation System : Microsoft Windows 2000

CPU : 펜티엄 4 2.5GHz, RAM : 512M, Programming Language : Java (jre 1.4.2)

가) 실험을 위한 가정

- **트랜잭션의 롤백은 발생하지 않는다:** 유효하지 않는 트랜잭션의 수행이나 deadlock 현상에 의해 트랜잭션의 취소되는 현상이 발생하지 않도록 트랜잭션들을 정의한다.
- **read나 insert/delete 연산에 의한 I/O time은 오직 실제 접근되는 노드의 수의 영향만을 받는다:** 이것은 루트 노드에서 접근되는 노드까지의 경로에 있는 노드들의 수는 고려하지 않는다는 것을 의미한다. 실험에 사용되는 XML 문서의 depth가 4로 고정되어 있고, 모든 트랜잭션들이 channel 엘리먼트의 자식 엘리먼트들에만 접근하기 때문에 이 가정은 실험 결과의 정확성에 영향을 미치지 않을 것이다.
- **유효성 검증을 위해서 노드에 접근하는 경우 I/O time을 고려하지 않는다:** 유효성 검증에 필요한 노드들의 수와 유효성 검증을 위한 I/O time은 비례한다. 그리고 각 검증 기법들은 검증에 필요한 노드들의 수가 각각 다르기 때문에 특별히 I/O time을 고려하지 않더라도 유효성 검증의 효율성을 측정하는데 문제가 없다.

나) 실험 변수

각 유효성 검증 기법과 로킹 기법의 성능 측정을 위해 표 3의 실험 변수들을 사용하여 실험을 수행하였다. 표 3에서 channel과 item은 XML 문서의 크기를 결정한다. 따라서, 실험에서는 channel과 item의 값을 조정함으로써 각 기법들의 문서 크기에 대한 확장성을 측정할 수 있다. query와 update에 의해서 응용 도메인에 따른 질의 패턴이 결정된다. 이것은 변경 트랜잭션의 빈도 수 증가에 따른 각 깁버들의 성능을 측정하기 위해서 사용된다. 실험에서는 100개의 트랜잭션들의 질의 및 변경 비율을 변경해 가며 실험을 수행하였다. max-

표 3 실험 변수들

실험 변수	설명
channel	실험에 사용되는 통합 RSS 파일의 channel element의 수
item	실험에 사용되는 통합 RSS 파일의 channel element 당 item element의 수
query	XML 문서를 변경하지 않고 질의만을 수행하는 트랜잭션의 수
update	XML 문서에 대한 변경을 수행하는 트랜잭션의 수
maxTransaction	동시에 수행될 수 있는 최대 트랜잭션의 수

Transaction은 병행수행이 가능한 최대 트랜잭션의 수를 의미한다.

다) 유효성 검증의 성능 실험

변수 설정 :

channel 당 item 엘리먼트의 수를 1000에서 10000으로 증가

update = 100, query = 0, maxTransaction = 1

유효성 검증 기법들의 성능을 실험하기 위하여 통합 RSS 파일의 channel 엘리먼트 당 item 엘리먼트의 수를 증가시켜가면서 부분 검증과 시퀀스 그룹 검증에 소요되는 시간을 측정하였다. 이렇게 XML 문서의 크기가 횡적으로 증가하는 경우 그림 10은 부분 검증 기법의 검증 시간이 증가하는 것을 보이고 있다. 이것은 변경 노드의 형제 노드 수가 증가하기 때문이다. 반면 시퀀스 그룹 검증은 변경되는 시퀀스 그룹에 대한 순서 유효성만을 검증하기 때문에 문서 크기 증가에 따른 영향을 받지 않는다. 이 실험에서는 데이타베이스의 I/O 시간을 고려하지 않았기 때문에 검증 시간이 수 밀리세컨드 정도 밖에 되지 않지만 비교적 큰 비용(cost)이 소요되는 I/O 시간을 고려하면, 문서 크기 증가에 따른 부분 검증의 소요 시간 증가는 트랜잭션의 응답시간을 크게 증가시킬 수 있게 된다는 것을 의미한다.

라) 병행수행 성능 실험

변수 설정 :

channel = 5, item = 200, maxTransaction = 30

update/query = 100/0, 90/10, 80/20, 70/30, 60/40, 50/50, 40/60, 30/70, 20/80, 10/90, 0/100

변경을 수행하는 트랜잭션은 데이타 아이템에 대한 배타 모드 로크를 요구하게 되므로 전체 트랜잭션에서 변경 트랜잭션이 차지하는 비율이 병행수행 성능에 큰 영향을 미칠 수 있다. 각 시행에서 트랜잭션들은 모두 100개씩 수행되었다. 그리고 질의/변경 트랜잭션의 비율을 10:0, 9:1, ..., 0:10으로 변화시켜가면서 각 로킹 기법에 의한 병행수행 성능의 척도라고 할 수 있는 초당 트랜잭션 수를 측정하였다.

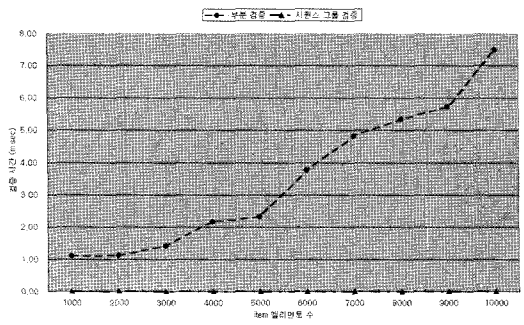

그림 10 부분 검증 기법과 시퀀스 그룹 검증과의 비교

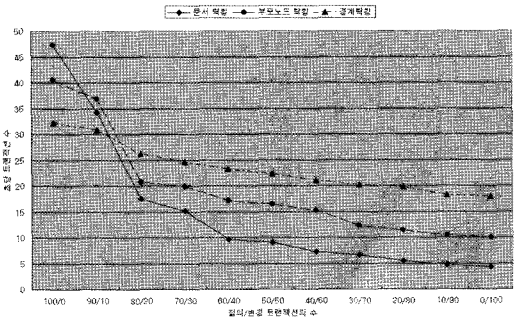

그림 11 질의/변경 트랜잭션 비율에 따른 병행수행 성능 측정

그림 11의 실험 결과를 보면 질의 트랜잭션만 수행되는 환경에서는 로킹의 오버헤드가 적은 순서인 Doc2PL>Node2PL>경계 로킹 순서로 성능 차이를 보이고 있다. 하지만 변경 트랜잭션의 비율이 증가함에 따라 Doc2PL과 Node2PL의 throughput은 급격히 하락하게 된다. 하지만 경계 로킹은 변경 트랜잭션 비율의 증가에 따른 성능 하락 폭이 가장 적어서 변경 트랜잭션의 비율이 20%를 넘을 때부터 가장 나은 성능을 가지게 되며 전반적으로 안정적인 병행수행 성능을 제공하고 있음을 확인할 수 있다. 이것은 경계 로킹이 로킹의 오버헤드가 증가하는 대신 트랜잭션들 간의 충돌 발생 가능성을 줄일 수 있기 때문이다.

5. 결론 및 추후 연구

유효 XML의 변경에 대한 기존의 유효성 검증 기법들은 검증 정확성 보장을 위해 필요한 병행수행 제어를 고려를 하지 않았기 때문에 비교적 낮은 병행수행 성능을 가지는 로킹 기법을 사용해야만 했다. 또한 큰 XML 문서에 대해서 유효성 검증을 수행할 때 검증 효율성의 문제를 가지고 있었다.

제안한 시퀀스 그룹 검증 기법은 pan-out이 큰 XML 문서에 대해서도 빠른 유효성 검증을 수행할 수 있으며, 적은 수의 순서 검증 정보 엘리먼트를 필요로 함으로써 유효성 검증의 효율성을 향상시킬 수 있었다. 또한 시퀀스 그룹 검증 기법의 정확성을 보장하기 위한 경계 로킹 기법은 엘리먼트 시퀀스 그룹 단위의 로킹을 가능하게 함으로써 기존 방법들에 비해 높은 병행수행 성능을 제공할 수 있다. 그리고 경계 로킹은 변경 트랜잭션의 비율 증가에 따른 성능 변화의 폭이 적어 어떤 환경에서도 안정적인 트랜잭션 처리 성능을 제공해 줄 수 있다.

하지만 경계 로킹은 유효성 검증과 같이 노드들의 수평적 관계에 의한 병행수행의 문제를 해결하기 위한 기

법이기 때문에 기존의 병행수행 제어 기법들이 지적하고 있는 데이타 아이템의 계층적 구조에 따른 충돌 제어에 대해서는 문제점을 가지고 있다. 따라서 XML에서 데이타 아이템들의 수직/수평 관계를 모두 고려하는 병행수행 제어 기법에 대한 연구가 필요하다.

참 고 문 헌

[1] James Clark, Steve DeRose, "XML Path Language (XPath) Version 1.0," http://www.w3.org/TR/xpath, W3C Recommendation, November 1999.

[2] Soctt Boag, et. al., "XQuery 1.0:An XML query language," http://www.w3c.org/TR/XQuery/, W3C working draft, November 2002.

[3] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, Daniel S. Weld, "Updating XML," ACM SIGMOD, pp.413-424, Santa Barbara, California, USA, May 2001.

[4] Jonathan Robie, Ratrick Lehti, "Updates in XQuery," In Proceeding of XML Conference, 2001.

[5] Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, "Isolation in XML Bases," Technical report, University of Mannheim, Germany, 2001.

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

[7] Torsten Grabs, Klemens Böhm, Hans-Jörg Schek, "XMLTM: Efficient Transaction Management for XML Documents," CIKM'02, McLean, Virginia, USA, November 2002.

[8] Stijn Dekeyser, Jan Hidders, "Path Locks for XML Document Collaboration," WISE, pp.105-114, 2002.

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

[10] 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, Lecture Notes in Computer Science, Vol.2419, pp.387-396, Aug 2002.

최 윤 상

2002년 서강대학교 컴퓨터학과(공학사) 2004년 서강대학교 컴퓨터학과(공학석사). 2005년 1월~현재 삼성SDS 삼성정보전략실 통합PJT팀원. 관심분야는 XML 데이타 처리 및 데이타베이스, 트랜잭션 관리, 웹 어플리케이션 임

박 석

1978년 서울대학교 계산통계학과(이학사). 1980년 한국과학기술원 전산학과(공학석사). 1983년 한국과학기술원 전산학과(공학박사). 1983년 9월~현재 서강대학교 컴퓨터학과 교수. 1989년~1991년/2002년~2003년 University of Virginia 방문교수. 1997년 2월~현재 한국정보보호학회 이사. 2005년 1월~현재 한국정보과학회 부회장, 2004년 1월~현재 한국정보과학회지 편집위원장 1999년~현재 DASFAA Steering Committee 멤버. 관심분야는 데이타베이스 보안, 실시간 시스템, 트랜잭션 관리, 데이타웨어하우스, 웹과 데이타베이스, XML, 역할기반 접근제어 임