전자 도서관에서 문서의 메타데이타 관리를 위한 2 버전 래치 기법

(Two Version Latch Technique for Metadata Management of Documents in Digital Library)

좌 은 희 † 박 석 ††
(Eun-Hee Jwa) (Seog Park)

요 약 최근 메타데이타의 주요 논쟁점으로 메타데이타의 표준화 문제가 등장하고 있다. 새로운 표준화 방향으로 인한 메타데이타의 확장성은 기존 메타데이타 관리 기법의 변화를 요구하게 되었다. 즉, 동적인 자료의 일관성 있는 저장과 유지방안이 필요하게 되었다. 이에, 본 논문에서는 새로운 표준화 형태의 메타데이타 특징들을 정의하고, 이러한 특징들을 만족하는 병행수행 제어 기법인 2 버전 래치 기법(Two Version Latch : 2VL)을 제안한다. 2VL은 래치를 사용하여 2 버전을 유지한다. 이러한 기법은 판독과 기록 연산간의 충돌을 최소화하고, 불필요한 로크의 소유를 제거함으로써 리프레쉬 지연을 최소화한다. 따라서, 기존 메타데이타 관리 기법에 비해서 판독 연산에 있어서의 빠른 응답시간과 높은 최근성 반영율을 제공한다. 성능 평가를 통해, 2VL 알고리즘이 메타데이타 관리에 있어서 기존의 알고리즘에 비해 좋은 성능을 가짐을 보인다.

키워드 : 전자 도서관, 메타데이타 관리 기법

Abstract Recently, a major issue in the research of metadata is the standardization of metadata format. The new extension capability of metadata in the standardization requires some changes - storing and managing dynamic data consistently. In this paper, we define the characteristics of new metadata and propose a concurrency control algorithm called Two Version Latch (2VL). 2VL uses a latch and maintains two versions. Maintaining two versions using latch minimizes conflicts between read operation and write operation. The removal of unnecessary lock holding minimizes refresh latency. Therefore, this algorithm presents fast response time and recent data retrieval in read operation execution. As a result of the performance evaluation, the 2VL algorithm is shown to be better than other algorithms in metadata management system.

Key workds : Digital Library, Metadata Management

1. 서 론

최근 전자 도서관의 메타데이타에서 주요한 논쟁점으로는 메타데이타 표준화 문제를 들 수 있다. 다양한 형태로 존재하는 메타데이타를 하나의 프레임워크로 통합함으로써 상호운영성을 증대시키고, 웹 환경을 기반으로 한 전자상거래의 개념을 어떻게 메타데이타에 도입하는가가 그 초점이 되고 있다. 이에 대한 대표적인 프로젝트로 INDECS(INteroperability of Data in E-Commerce Systems)[1][2][3]와 DC[4]에서 메타데이타 표준화 방안을 마련하고 있다. 표준화 내용의 특징으로는 기존의 메타데이타가 단순히 서지 정보의 정적인 기록만을 의미했던 데 반해 새로운 표준화 내용은 관리적(지적 재산권 내용 포함), 상업적 내용을 포함하는 동적인 형태의 정보로 확장되고 있다는 점이다. 가령, INDECS 메타데이타 집합에서는 어떤 사용자가 어떤 문서를 어느 시각, 어느 위치에서 지불을 마치고 다운 받았는지에 대한 상세한 정보까지 표현할 수 있게 되었다. 이러한 정보의 표현은 시각에 따라 수시로 기록되야 하는

· 본 연구는 한국과학재단 목적기초연구(과제번호 : R01-2000-00272) 지원으로 수행되었음.
† 비 회 원 : (주)퓨처시스템 정보통신 연구소 전임연구원
ehjwa@future.co.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
spark@dblab.sogang.ac.kr
논문접수 : 2000년 11월 17일
심사완료 : 2002년 3월 22일

정보로써 Dublin 메타데이타 집합[4]에 비해 동적인 형태를 갖는다는 특징이 있다. 즉, Dublin 메타데이타에서는 갱신 연산이 거의 없었던데 반해 INDECS 메타데이타는 빈번한 갱신 연산을 갖게 된다. 참고문헌 [5]에서는 INDECS 프로젝트에서 제시한 모델의 개념을 기반으로 하여 정의 및 구현한 메타데이타 집합을 소개한다.

표준화 방향으로 인한 메타데이타의 확장성은 기존 메타데이타 관리 기법의 변화를 요구하게 되었다. 즉, 사용자의 질의에 대한 신속한 정보 제공과 함께 동적인 자료의 일관성 있는 저장과 유지방안을 필요로 하게 되었다. 이러한 메타데이타 관리는 기존 관리 방안[6][7]에서 수행하였던 데이타에 대한 삽입과 삭제 연산만으로 처리하기가 어렵다. 이는 처음 입력된 데이타에 대해 갱신 연산이 거의 없는 환경을 가정하였기 때문에 기록 연산이 빈번해질 경우 판독 연산에 대한 응답시간의 지연과 웹 환경을 기반으로 하였을 때 단일 버전 훼손시 발생될 수 있는 문제점들을 갖게 된다[8][9]. 가령, 단일 버전만을 유지하여 온라인 상에서 정보 갱신을 허용하였을 경우 데이타에 대한 로크를 소유한 채로 사용자의 접속이 중단되는 상황이 발생할 수 있다. 또한, 단일 버전의 사용은 정보의 덮어쓰기(overwrite)와 같은 요구되지 않는 결과를 발생시킬 수도 있다. 반면 다중 버전 알고리즘[10][11][12]을 고려해 볼 수 있는데 이는 저장되는 데이타의 방대함으로 부적합하다. 이에, 최소 수의 버전인 두 개의 버전을 가지고 판독 연산과 기록 연산 간의 충돌을 해결하고자 제안된 알고리즘인 2VQL[13]을 고려해 볼 수 있다. 그러나 2VQL은 메타데이타 상호간에 의존성이 존재하지 않는 서로 다른 데이타 항목들 간에 불필요한 판독 로크를 소유하도록 함으로써 갱신 트랜잭션의 리프레쉬를 상당시간 지연시킨다. 이러한 문제는 [그림 1]의 스케쥴에서 살펴볼 수 있다. [그림 1]에서 트랜잭션 T3와 T4는 4번 시점과 7번 시점에서 트랜잭션 Q2와 Q1이 각각 X의 기본 버전인 X_0와 Z의 기본 버전인 Z_0에 대한 판독 로크를 소유하고 있기 때문에 리프레쉬를 할 수가 없다. 트랜잭션 Q2가 8번 시점에서 완료되었지만 트랜잭션 T3는 T4와 MRU(Merged Refresh Unit)을 생성하였기 때문에 Q1이 완료되는 시점까지 리프레쉬는 지연된다. 따라서 Q5는 9번 시점에서 트랜잭션 T3의 기록 결과를 판독하지 못함으로써 최근성 반영율을 낮게 한다.

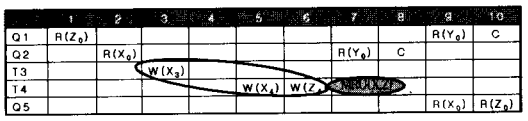

그림 1 2VQL이 생성하는 스케쥴

이에 본 논문에서는 메타데이타 관리시 요구되는 속성들을 정의하고 이를 고려한 2VL(2 Version Latch) 알고리즘을 제안한다. 2VL은 일관된 데이타의 판독을 보장하고 최근성 반영율을 높이며 짧은 응답 시간을 갖도록 하여, 기존 메타데이타 관리 기법이 동적인 데이타를 관리할 때 발생될 수 있는 문제를 해결한다.

2. 메타데이타 관리 요구사항

메타데이타는 그 응용분야에 따라 다른 성질을 가지게 된다. 전자 도서관에서 필요로 하는 메타데이타는 원문에 대한 정보 제공을 위한 데이타이다. 이는 기존의 일반적인 데이타와는 다른 성질을 갖는다. 이에 우선 본 장에서는 INDECS 메타데이타를 소개하며 메타데이타가 갖는 특성을 정의하고 메타데이타에 대한 연산을 수행하는 트랜잭션을 정의한다.

2.1 INDECS 메타데이타 요소

전자상거래 기반의 환경이 구축됨에 따라 그에 상응하는 다양한 형태의 정보를 표현할 수 있는 메타데이타 집합에 관한 연구가 필요하게 되었다. 이러한 메타데이타 표준에 대한 연구의 일환으로 나타난 것이 INDECS 메타데이타 모델이다. INDECS 메타데이타 모델은 다양한 형태의 정보를 표현할 수 있는 추상적 시각(abstract view), 창조적 시각(creative view), 상거래 시각(commerce view), 법률적 시각(legal view)의 네 가지 관점을 제시하였다. 또한 이 모델은 여러 네트웍 상의 문서 검색에 대한 표준안을 제안하며 전자 도서관뿐만 아니라 웹 상의 많은 다양한 정보를 표현할 수 있는 형태의 모델을 제안하였다. 본 논문에서의 메타데이타 관리 기법은 INDECS 메타데이타 모델을 그 기반으로 삼는다.

INDECS 메타데이타 집합은 상위 요소로 크게 Label, Class, Relation, Extent의 4가지로 구성된다. 4가지 요소는 그 서브타입에서 상세한 내용을 기록할 수 있으며 Label을 제외한 나머지 요소들은 사용자의 편의에 따라 생략 가능하다. Label은 엔터티를 식별하기 위한 내용을 포함한다. Class는 엔터티의 내용에 관한 상세한 기술 혹은 범주 등을 기술한다. Class의 서브타입으로 Primitive, IP(Intellectual Property), Creation, Transaction의 요소를 갖는다. Primitive는 INDECS 메타데이타 모델의 추상적 시각을 표현하고, IP는 법률적 시각, Creation은 창조적 시각, Transaction은 상거래 시각의 관점을 각각 기술한다. 상위 세 번째 요소인 Relation은 하나의 엔터티와 또 다른 엔터티 사이의 일대일 관계를

주로 기술한다. 마지막으로 Extent는 엔터티의 측정 가능한 요소들을 기술한다.

INDECS 모델의 각 요소들로부터 알 수 있듯이 INDECS 메타데이타는 기존의 메타데이타에 비해 동적인 정보를 포함한다. 가령, 저작권자의 위임이라든가(IP 요소로 표현), 서지 거래 내역(Transaction 요소로 표현)등과 같이 서지와 관련되어 발생되는 사건(event)들을 메타데이타에 표현하도록 하고 있다. 이러한 요소들은 기존의 Dublin 메타데이타 집합에서는 포함하지 않던 요소들로 Dublin 집합의 정적인 속성과 INDECS 집합의 동적인 속성을 구별 짓는 한 예이기도 하다.

2.2 메타데이타의 특성 정의

전자 도서관의 메타데이타는 다른 전통적 응용 환경의 일반 데이타와는 다른 특성을 갖는다. 이에 본 논문에서는 다음과 같이 메타데이타의 3 가지 특성을 정의한다.

[성질 1]

메타데이타 상호간에는 의존성(dependency)이 존재하지 않는다. 따라서, 어떤 메타데이타에 대해 완료된 연산 결과가 다른 메타데이이타에 영향을 미치지 않는다.

[성질 2]

판독 연산은 일정 시간동안 판독한 데이타에 대해 재 판독하지 않는다.

[성질 3]

하나의 메타데이타 문서 안에서 요소들간의 의존성은 존재한다. 즉, 다중 버전인 경우 버전 간의 의존성이 존재한다.

하나의 메타데이타는 어떤 하나의 원문에 대한 정보만을 포함한다. 따라서, [성질 1]은 메타데이타는 원문으로부터 유도된 데이타이며 각 원문 상호간에는 의존성이 없으므로 메타데이타 상호간의 관계 역시 의존성이 없음을 의미한다. 그리고 [성질 2]는 메타데이타가 원문 검색을 위해 제공되는 정보이기 때문에 일정 시간동안 판독된 데이타는 재 판독되지 않는다는 내용이다. [성질 3]은 하나의 문서에 대한 메타데이타 원소들간에는 서로 의존성이 존재하기 때문에 다중 버전을 생성할 경우 각 버전간의 의존성이 존재함을 의미한다.

2.3 트랜잭션의 분류 및 특성

메타데이타 관리에 요구되는 작업들을 크게 두 가지 형태인 갱신 트랜잭션과 판독전용 트랜잭션으로 분류한다. 다음 [표 1]은 트랜잭션 분류에 따른 세부 내용이다.

판독전용 트랜잭션은 판독 연산만을 포함하는 트랜잭션을 말하며 판독 로크만을 사용한다. 갱신 트랜잭션은 기록 연산과 판독 연산간의 충돌을 최소화하고자 각 연산내용에 따라 세분화시켰다. 다음은 갱신 트랜잭션 연산을 분류하고 수행되는 내용을 소개한다.

표 1 트랜잭션의 분류

성격 / 분류	갱신 트랜잭션	판독 전용 트랜잭션
상대적 수행시간	장기간	단기간
접근하는 데이타량	소량	다양한 범위
주요 연산 종류	판독과 기록	판독
판독 데이타 최근성 요구 정도	절대적	절대적이지 않음
동시 수행되는 트랜잭션 수	적음	많음
연산 내용	메타데이타 추가, 갱신, 삽입, 삭제 처리	질의 처리

- 갱신, 추가 연산
  일반 기록 로크로 수행한다.
- 삽입 연산
  새로운 메타데이타의 삽입은 그 시점에서 수행되는 연산과의 충돌이 전혀 없다. 따라서, 삽입 연산은 별도의 로크 획득 없이 수행한다. 삽입 연산은 갱신 트랜잭션에서 수행되나 로크 획득이 필요 없으므로 INS(data)로써 표현한다.
- 삭제 연산
  삭제 로크는 DEL(data)로써 표현한다. 삭제 연산 수행은 연산 발생시 메타데이타에 삭제된 데이타임을 표시만하고 실제 연산 수행은 시스템 로드가 적은 시점에 원문과 함께 배치 처리한다.

메타데이타 관리에 있어 중요 평가 사항 중 하나는 어느 정도의 최근성을 반영할 수 있는가하는 점이다. 최근성 반영율은 데이타베이스가 유지해야 하는 일관성 기준의 완화로써 획득될 수 있다. 갱신 트랜잭션과는 달리 판독전용 트랜잭션은 데이타베이스 상태에는 아무런 영향을 미치지 않는다. 따라서 판독전용 트랜잭션이 유지해야 하는 논리적 일관성은 완화될 수 있다. 일반적으로 판독전용 트랜잭션에 대해 적용될 수 있는 일관성 기준은 [표 2]와 같다[1].

[표 2]에 제시된 일관성 기준 중 메타데이타베이스에서의 판독전용 트랜잭션은 갱신 일관성 기준을 만족한다. 메타데이타베이스가 갱신 일관성 기준을 만족함은 [정리 1]에서 증명하였다.

[정리 1] 2VL의 판독 전용 트랜잭션은 갱신 일관성 기준을 만족한다.

[증명]

·[성질 1]에 의해 판독 연산은 판독에 영향을 미치

표 2 판독전용 트랜잭션에 대한 일관성 기준의 형태

형태	설명	SG(Serialization Graph)에서의 제한
엄격한 일관성 (strict consistency)	완료 순서와 일치된 갱신 트랜잭션들의 직렬 순서를 본다.	갱신 트랜잭션들의 순서가 완료순서와 일치하며 사이클이 금지된다.
강성 일관성 (strong consistency)	갱신 트랜잭션들의 직렬 순서를 본다.	사이클이 금지된다.
약성 일관성 (weak consistency)	독립적으로 갱신 트랜잭션들의 직렬 순서를 본다.	단일 질의 사이클이 금지된다.
갱신 일관성 (update consistency)	판독전용 트랜잭션이 보는 값을 생성하는 갱신 트랜잭션들의 집합과 관련하여 판독전용 트랜잭션이 직렬성을 만족한다.	두개의 갱신 트랜잭션간의 r-w 에지를 제거함으로써 사이클이 제거된다면 단일 질의 사이클을 허용한다.

는 갱신 트랜잭션 연산만을 고려한다.

· [성질 2]에 의해 판독 전용 트랜잭션은 판독한 데이타에 대해 재 판독하지 않는다.

· 기록 충돌 연산간에는 배타적 래치의 적용으로 각 연산간 직렬 순서를 보장한다.

메타데이타베이스가 갱신 일관성 기준을 만족함으로써 제안하는 알고리즘 2VL이 생성하는 스케쥴 [그림 2]는 최근 데이타를 판독할 수 있다. 트랜잭션 T1은 1번 시점에서 판독 연산을 수행하지만 제안하는 알고리즘인 2VL 알고리즘에 따라 판독 연산이 완료됨과 동시에 로크를 해제한다. 따라서, 트랜잭션 T2는 7번 시점에서 리프레쉬를 수행하고 트랜잭션을 완료하여 트랜잭션 Q3이 트랜잭션 T2의 결과 값을 판독할 수 있게 된다. 즉, 메타데이타 관리 요구 사항인 최근성을 반영할 수 있다. 트랜잭션 Q3이 판독하는 데이타의 일관성은 갱신 일관성 기준을 만족한다.

	1	2	3	4	5	6	7	8
T1	$R(W_0)$		$R(Z_0)$	$W(Z_1)$	C			
T2					$R(W_0)$	$W(W_1)$	C	
Q3		$R(Z_0)$					$R(W_1)$	C

그림 2 2VL이 생성하는 스케쥴

3. 메타데이타 관리를 위한 2 버전 병행수행 제어 기법

기록 연산이 빈번한 환경에서 2VQL과 기존 메타데이타 관리 기법은 메타데이타의 속성을 고려할 때 효율적이지 못하다. 이에 제안하는 알고리즘인 2VL에서는 메타데이타 속성을 고려하여 래치를 사용함으로써 기존 기법에서 발생 가능한 문제점을 해결한다. 래치는 2PL (Two-phase Locking)과 같은 로킹 기법에 비해 비용 및 구현 시 실제 오버헤드가 적다는 장점을 갖는다[13]. 또한 2 버전을 유지함으로써 판독과 기록 연산간의 충돌로 인한 지연을 제거함으로써 판독 연산에 있어서의 빠른 응답시간 뿐만 아니라 기록 연산에 있어서도 지연없는 수행을 함으로써 이후 판독 연산에 한해서는 최근성 반영율이 높게 된다. 2VL의 기본 내용은 [그림 3]과 같다.

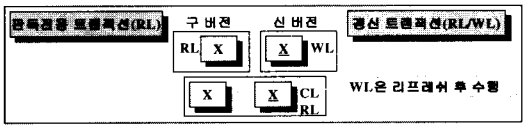

그림 3 2VL 개요

2PL에서는 트랜잭션이 완료될 때 스케쥴러가 트랜잭션이 소유하는 로크를 한꺼번에 해제한다. 따라서 이미 판독된 데이터에 대해 불필요한 로크를 소유하고 있게 된다. 메타데이타 관리에 있어 판독 로크의 실제 역할은 다른 연산과의 충돌을 일으키지 않도록 데이터를 판독하였음을 알리는 플래그 설정의 기능을 한다. 따라서 판독하고 있는 동안에만 데이터에 대한 로크를 설정함으로써 불필요한 로크의 소유를 방지하고 다른 연산과의 충돌을 피하도록 한다.

2VL의 알고리즘에서는 판독 연산에 대한 로크로써 RL(Read Latch), 기록 연산에 대한 로크로써 WL (Write Latch)과 리프레쉬 연산의 로크로써 CL (Certificate Latch)을 사용한다. 그리고 2.1절에서 소개한 삽입 로크 INS와 삭제 로크 DEL을 사용한다. 2VL에서는 2 버전 데이터의 세 가지 상태인 [그림 4]의 경우를 허용한다. 본 논문에서는 데이터 버전 역할의 의미상 먼저 생성되어 사용되고 있는 버전을 구 버전(Old Version)이라 하고 기록 연산에 의해 새로 생성된 버전을 신 버전(New Version)이라 한다.

[그림 3]의 2VL은 두 개의 버전을 유지한다. 판독 연산은 판독 로크를 획득한 후 [그림 4]의 판독 데이타 테이블에 따라 판독 버전을 결정한다. 기록 연산은 기록 로크를 획득한 후 신 버전을 생성하고, 리프레쉬 로크를 획득하여 리프레쉬 한다. 리프레쉬 시점 이후에 수행되는 판독 연산은 판독 데이타 테이블에 따라 신 버전을 판독하도록 한다. 리프레쉬 수행시 발생되는 기록 연산은 리프레쉬가 완료될 때까지 지연되었다가 수행된다.

	판독 전용 트랜잭션	갱신 트랜잭션
상태 1	구 버전	구 버전
상태 2	구 버전	구 버전
상태 3	신 버전	신 버전

그림 4 2VL 판독 데이타 테이블

그러나 구 버전에 대한 판독 연산의 로크 기간은 래치 사용으로 단시간이며 신 버전이 생성된 이후 발생되는 판독 연산은 신 버전을 판독하므로 리프레쉬 수행 시간은 짧게 된다. 따라서 대기하는 기록 연산의 시간도 짧다고 할 수 있다.

3.1 로크 호환 테이블

[그림 4]는 각 상태에서 판독전용 트랜잭션과 갱신 트랜잭션이 판독하게 되는 각 데이타 버전을 상태별로 구분하여 표현한 것이다. 2VL에서는 2 버전 데이타의 세 가지 상태를 허용한다. 그러나 판독 로크에 대한 리프레쉬 지연의 감소로 상태 3은 즉시 상태 1로 변환된다. [그림 4]에서와 같이 어떤 경우에도 동일 시점에서 한 트랜잭션이 두 가지 버전을 동시에 판독할 수 없고 항상 데이타의 상태에 따라 결정된다. 따라서, 두 개의 버전을 판독함으로써 발생될 수 있는 사이클을 방지하기 위한 종속 그래프의 추가 비용이 필요치 않다. 또한 신 버전이 생성되었을 때 이후 시점의 판독 전용 트랜잭션은 신 버전을 판독하도록 함으로써 구 버전에 대한 판독 로크로 발생될 수 있는 리프레쉬의 지연을 방지하며 판독전용 트랜잭션의 결과가 항상 최근성을 반영할 수 있도록 한다.

표 3 2VL 로크 호환 테이블

		Lock holder				
		INS	DEL	RL	WL	CL
Lock requester	INS	○	○	○	○	○
	DEL	○	×	○	×	×
	RL	○	○	○	○	○
	WL	○	×	○	×	×
	CL	○	×	○	×	

[표 3]은 2VL의 로크 호환 테이블이다. RL은 단기간 공유 래치(short-term shared latch)를 적용하고, WL과 DEL은 단기간 배타 래치(short-term exclusive latch)를 적용한다. 그리고, CL은 RL과 같이 단기간 공유 래치를 적용한다. INS는 다른 연산과의 충돌이 없으므로 별도의 로크를 고려하지 않는다.

RL과 WL의 로크 호환은 각 상태별로 약간의 의미상 차이가 있다. [그림 4]의 상태 3에서 RL을 획득한 경우의 WL의 요구는 WL이 잠시 지연됨을 의미하며, 상태 1과 2에서 WL과 RL에 대한 요구는 호환이 된다. 그리고, CL은 리프레쉬를 하는 단계로서 판독 연산으로 하여금 신버전이 판독 가능 상태임을 암시한다. 따라서 RL은 신버전을 판독하게 되어 CL과 호환되며 즉시 리프레쉬를 할 수 있다.

2VL은 [표 1]의 트랜잭션 분류에 따라 판독 전용 트랜잭션과 갱신 트랜잭션의 수행으로 나누어 소개한다.

3.2 판독 전용 트랜잭션

2VL에서는 판독 연산 결과에 대한 최근성 반영율을 높이고 지연 없는 판독 연산을 수행하기 위해 다음과 같은 판독 규칙을 정의한다.

[판독 규칙 1]

모든 판독 연산은 판독 데이타 테이블에 따라 판독 가능한 데이타 버전을 선택한다.

[판독 규칙 2]

판독 연산에 대한 로크는 단기간 공유 래치를 사용한다

[판독 규칙 1]은 판독 연산이 항상 최근 버전만을 판독하도록 함으로써 최근성 반영율을 높인다. 또한, 두 개의 버전을 동시에 판독하지 않음으로써 두 개의 버전을 관리하는데 요구되는 비용을 최소화한다. [판독 규칙 2]는 판독 연산에 단기간 공유 래치를 적용함으로써 트랜잭션 수행이 완료되는 시점까지 로크를 소유하는 비용을 최소화한다. 다시 말해, 하나의 데이타에 대해 판독 연산이 완료되면 즉시 로크를 해제함으로써 리프레쉬 연산과 발생될 수 있는 충돌을 최소화한다.

3.3 갱신 트랜잭션

3.3.1 갱신 트랜잭션의 수행

갱신 트랜잭션의 연산으로는 판독 연산과 기록 연산으로 나누어 수행한다. 판독 연산에 대한 래치는 판독 전용 트랜잭션과 마찬가지로 단기간 공유 래치를 사용한다. 기록 연산은 단기간 배타 래치를 적용함으로써 기록 연산간의 직렬성을 유지한다. 즉, 판독 연산은 동시에 여러 개가 수행될 수 있으나, 기록 연산은 하나의 연산만이 수행된다. [그림 5]는 갱신 트랜잭션에 의한 버전 생성단계를 보여 준다.

[그림 5]에서 OV(One Version)는 데이타의 상태가 1 버전임을 나타내는 마크로써 OV가 SET될 경우에만 기록 래치가 가능하다. [단계 1]에서 갱신 트랜잭션은 구 버전을 판독한다. [단계 2]에서 기록을 위해 WL을 획득하고 OV를 RESET 한다. [단계 2]인 동안 새로운 WL에 대한 요구는 래치 호환 테이블에 의해 획득되지

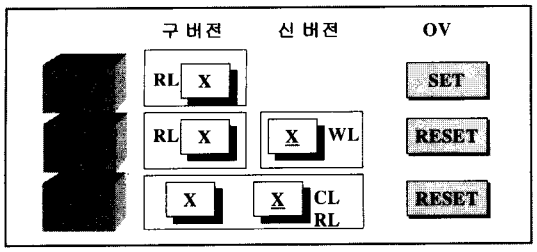

그림 5 갱신 트랜잭션의 수행

못한다. [단계 3]에서는 두 개의 버전이 공존하게 되며 WL을 CL로 변환하여 데이타가 판독 가능 상태임을 알린다. 판독 가능 테이블에 따라 CL로의 래치 변환 시점 이후부터 발생하는 판독 연산들은 신 버전을 판독하도록 한다. 마지막으로 3.4절의 래치 관리 테이블을 참조하여 구 버전에 대한 이전 판독 연산들이 완료될 때 리프레쉬하여 [단계 1]로 돌아간다. 그러나, 구 버전에 대한 판독 연산은 짧은 시간에 완성되며 신 버전 완료 시점 이후부터의 판독 연산은 신 버전을 판독하므로 [단계 3]의 유지 기간은 짧다고 할 수 있다. 이러한 흐름은 다음의 기록 규칙으로 정의한다.

[기록 규칙 1]

기록 연산의 로크는 배타적 래치를 사용한다.

[기록 규칙 2]

기록 로크와 다른 로크와의 충돌 관계는 로크 호환 테이블에 따르며 충돌하는 연산간에는 직렬성을 유지하기 위해 이전 연산의 리프레쉬 완료 후 다음 연산을 수행한다.

[기록 규칙 1]은 기록 연산에 배타적 래치를 사용함으로써 기록 연산간의 직렬성을 유지하고자 함이다. [기록 규칙 2]는 기록한 트랜잭션이 완료 후 리프레쉬 되기 전에, 또 다른 트랜잭션이 동일한 데이타에 기록 래치를 요구한 경우, 2 버전 알고리즘이 가질 수밖에 없는 한계 상황에 대한 내용이다. 그러나, 앞서 밝혔듯이 [단계 3]의 수행 기간은 짧으며 갱신 트랜잭션이 판독전용 트랜잭션에 비해 장기간 수행 시간을 갖으므로 갱신 트랜잭션의 기록 연산이 단시간의 지연을 갖도록 하는 것이 최소 비용을 요구한다.

3.3.2 갱신 트랜잭션의 리프레쉬

갱신 트랜잭션이 리프레쉬 하기 위해서는 다음의 리프레쉬 수행 조건을 만족해야 한다.

[리프레쉬 수행 조건]

새로 생성된 데이타 버전이 리프레쉬 하기 위해서는 먼저 래치 관리 테이블에서 구 버전에 대한 판독 로크가 없어야 한다. 그리고, 직렬 순서상 충돌하는 데이타에 대한 이전 리프레쉬 연산들이 완료된 후여야 한다.

(OldVer_ReadNum = 0) ∧(SET_Prev_Refresh(n) > SET_Cur_Refresh(n))

- OldVer_ReadNum : 래치 관리 테이블에서 충돌하는 데이타의 구 버전 판독 수
- SET_Prev_Refresh(n) = SET_Prev_Refresh(a) ∪ SET_Prev_Refresh(b) ∪...∪ SET_Prev_Refresh(m) : 충돌하는 데이타간의 직렬 순서상에서 리프레쉬 수행 순서 n 이전에 위치하는 리프레쉬 집합들
- SET_Cur_Refresh(n) : 현재 리프레쉬 수행 순서 n의 리프레쉬 집합

이러한 조건이 만족되면 다음의 리프레쉬 단계로 들어간다.

[리프레쉬 수행 단계]

① 구 버전 판독 리스트를 검사한다.

② 다른 트랜잭션이 구 버전에 대해 연산을 수행하지 않도록 구 버전에 배타적 래치를 획득한다. 그러나 신 버전에 한해서는 공유 래치를 획득한다.

③ 구 버전에 대한 관리 정보를 신 버전으로 이동시킨다(version switching).

④ 데이타에 대한 OV(One Version)를 설정(SET)한다.

⑤ 구 버전에 대한 단기간 배타 래치와 신 버전의 단기간 공유 래치를 해제한다.

⑥ 리프레쉬 전의 구 버전을 위해 할당된 메모리를 반환한다

리프레쉬 수행시 OV(One Version)를 설정(SET)함으로써 신 버전이 현재 구 버전이 되었음을 암시하고 래치 관리 테이블의 대기 리스트에 있는 기록 연산이 가능하도록 한다.

3.4 래치 관리 테이블

래치는 연산이 수행되는 동안에만 페이지에 걸리는 단기 로크로써, 판독 연산과 기록 연산간의 충돌을 방지하기 위한 별도의 관리 방안이 필요하다. 이에 대해 [그림 6]의 래치 관리 테이블을 유지함으로써 연산 수행 순서의 직렬성을 유지한다. 래치 관리 테이블은 [성질 1]

데이타	OV	구 버전 판독 리스트	신 버전 판독 리스트	기록 리스트	리프레쉬 리스트	대기 리스트
X	RESET		Q3		T2	T5
Y	RESET	Q6		T7		T8
Z	SET	Q9, Q10				

그림 6 래치 관리 테이블

을 근거로 하여 각 데이타 단위별로 현재 로크를 획득한 트랜잭션 아이디들의 리스트를 유지한다. 이러한 테이블의 구현은 각 트랜잭션 아이디 리스트를 비트맵으로 표현함으로써 비용을 최소화한다.

4. 성능 평가

4.1 평가 모델

제안하는 알고리즘의 성능 평가는 [그림 7]의 시스템 모델[11]과 시뮬레이션 툴 AweSim[15]을 사용하였다. [그림 7]은 시뮬레이션에 사용된 모델이며 [표 4]는 평가에서 사용된 각 파라미터(parameter)의 초기 값들이다. 각 파라미터 값은 전자 도서관에서 메타데이타 검색을 위한 판독 전용 트랜잭션이 갖는 연산수의 다양성과 접근 데이타의 방대함을 고려하여 설정하였다.

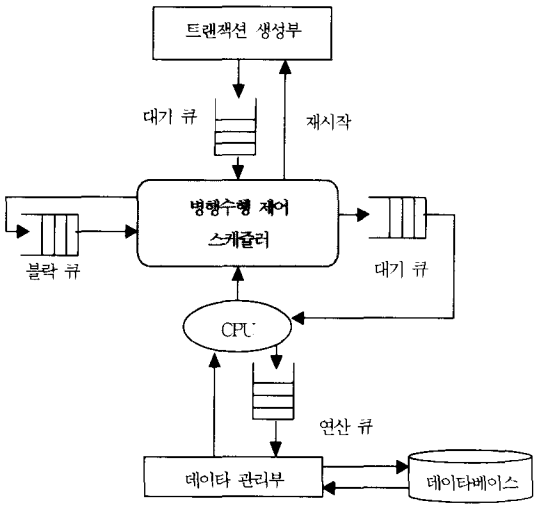

그림 7 시뮬레이션 모델

표 4 시뮬레이션 파라미터

파라미터	값
데이타베이스 크기	100
트랜잭션 수	50
갱신 트랜잭션이 갖는 연산의 최대/최소 수	20/5
판독 전용 트랜잭션이 갖는 연산의 최대/최소 수	40/5
디스크 접근 시간	20 ms
CPU 처리 시간	10 ms
블록 오버헤드	5 ms

4.2 평가 분석

평가 대상으로는 2VQL과 기존 메타데이타 관리 기법, 제안하는 알고리즘인 2VL로 세 가지 알고리즘을 평가한다. 평가 기준으로는 다음의 세 가지 사항을 평가한다. 첫째, 갱신 트랜잭션의 수행시간을 비교한다. 메타데이타 관리에서는 상대적으로 많은 수의 판독전용 트랜잭션이 수행되게 되는데 판독 전용 트랜잭션의 비율에 따라 병행 수행되는 갱신 트랜잭션의 수행시간을 측정하여 각 알고리즘과 비교한다. 둘째, 리프레쉬 수행시간을 비교한다. 2VL에서는 불필요한 로크의 소유로 인한 리프레쉬의 지연을 해결하였음을 각 알고리즘의 리프레쉬 수행시간을 비교하여 평가한다. 셋째, 구 버전 판독 비율을 측정한다. 2VL에서는 구 버전 판독의 문제를 해결함으로써 메타데이타 관리에 있어 효율적인 기법임을 성능평가를 통해 밝힌다.

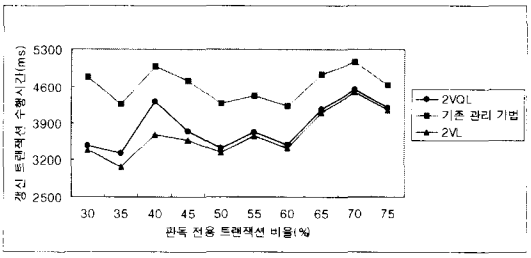

그림 8 판독 전용 트랜잭션 비율에 따른 갱신 트랜잭션 수행시간

[그림 8]은 판독전용 트랜잭션의 비율이 증가함에 따라 갱신 트랜잭션의 수행시간이 어떻게 변화하는지를 보여 준다. 갱신 트랜잭션 수행시간 측정 식은 다음과 같다.

갱신 트랜잭션 수행시간 =

$$\frac{\sum_{i=1}^{N} \text{종료시간}(Ti) - \text{시작시간}(Ti)}{N},$$

N = 트랜잭션 수

판독 전용 트랜잭션 비율이 적은 경우의 기존 메타데이타 관리 기법은 갱신 트랜잭션 수행시간에 있어 2VL, 2VQL보다 높은 수행시간을 갖는다. 이는 기존 관리 기법은 단일 버전을 사용하였기 때문에 구 버전에 대한 판독 로크 소유로 전체적인 갱신 트랜잭션 수행에 있어 지연을 갖기 때문이다. 2VL이 2VQL보다 갱신 트랜잭션 수행시간에서 좀더 나은 성능을 보이는 것은 2VL에서는 리프레쉬의 지연을 최소화함으로써 갱신 트랜잭션

의 수행시간을 단축시켰기 때문이다. 그리고, [그림 8]에서 판독 전용 트랜잭션의 비율 증가에 반해 일부 구간에서 하향 곡선이 보이는 것은 시뮬레이션 변수 설정 시 판독과 기록 연산의 수는 임의의 수를 설정하도록 하였기 때문이다. 이는 전자 도서관 환경의 다양한 연산 수를 고려하고자 함이었다. 따라서, 트랜잭션 비율에 따라 실제 수행되는 연산의 수가 반드시 많은 것은 아니므로 일부의 하향 구간 그래프가 생긴 것으로 고려된다.

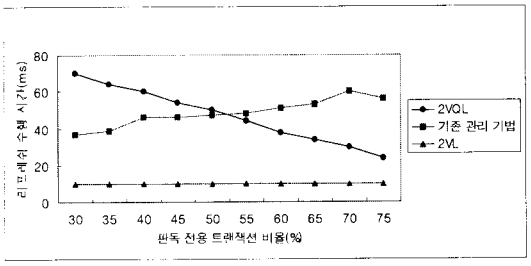

그림 9 판독 전용 트랜잭션 비율에 따른 리프레쉬 수행 시간

[그림 9]는 판독전용 트랜잭션의 비율에 따른 기록 연산의 리프레쉬 수행시간을 나타낸다. 리프레쉬 수행 시간의 측정 공식은 다음과 같다.

리프레쉬 수행 시간 =

$$\frac{\sum_{i=1}^{N} \text{종료시간}(OPi) - \text{시작시간}(OPi)}{N},$$

N = 수행된 전체 리프레쉬 수,
OPi = i번째 리프레쉬 연산

2VL은 판독전용 트랜잭션의 비율에 관계없이 거의 일정한 수행 시간을 유지하지만 2VQL은 판독전용 트랜잭션의 비율에 따른 하향 곡선을 그리고 있다. 즉, 2VQL은 갱신 트랜잭션이 많은 시점에서의 MRU (Merged Refresh Unit) 생성으로 리프레쉬 지연을 가져오기 때문에 리프레쉬의 수행시간이 증가한다. 기존 관리 기법에서의 리프레쉬 수행 시간은 새로운 데이타가 삽입되고 이전 데이타가 삭제되어 사용자가 판독 가능한 시점까지의 시간을 의미한다. 기존 관리 기법에서는 삭제될 버전에 대한 판독 연산이 없는 시점에서만 삽입과 삭제 연산이 동시에 이루어져야 하므로 판독전용 트랜잭션의 비율에 따른 그래프 증가율을 보인다.

[그림 10]은 판독 전용 트랜잭션의 비율에 따른 구 버전 판독 수를 나타낸다. 구 버전 판독 수는 신 버전이 생성된 이후 시점에 구 버전을 판독하는 트랜잭션

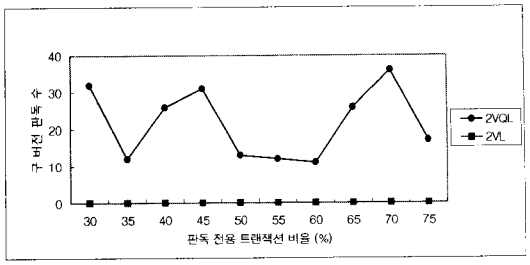

그림 10 트랜잭션의 구 버전 판독 수

수를 의미한다. 2VL은 신 버전이 생성된 후 판독 데이타 테이블에 따라 모든 트랜잭션이 항상 신 버전을 판독하므로 구 버전 판독 수는 거의 없다. 그러나, 2VQL에서는 알고리즘에 따른 구 버전 판독과 리프레쉬의 지연으로 인한 신 버전의 장시간 유지로 구 버전 판독 수가 상대적으로 높다. 그리고, 2VQL 그래프에서의 일정 간격의 하향 곡선은 리프레쉬로 인한 영향으로 고려된다. 기존 메타데이타 관리 기법은 1 버전을 유지하므로 구 버전 판독 비율의 평가 대상에서 제외하였다.

이상의 평가 분석에서 2VL은 다른 두 알고리즘에 비해 트랜잭션 수행시간과 최근성 반영율 면에서 좋은 결과를 보였다. 제안하는 알고리즘은 갱신 트랜잭션의 비율이 증가되는 경우 기존 메타데이타 관리 기법에 비해 수행시간에 있어서 좋은 성능을 보였으며 리프레쉬 수행시간이나 최근성 반영율에 있어서도 2VQL보다 좋은 결과를 보였다.

5. 결 론

메타데이타베이스는 동적인 정보의 일관성 있는 관리와 사용자의 질의에 대한 빠른 응답 시간과 함께 최근성을 제공할 수 있어야만 한다. 그러나 현재의 메타데이타 관리 기법에서는 이러한 요구사항을 충족시키지 못하고 있다. 기존의 메타데이타는 정적인 형태의 정보만을 포함함으로써 최근성 반영 여부나 기록 연산에 대한 별다른 고려를 하지 않았다. 그러나 전자상거래의 개념을 표현하고자 하는 메타데이타 표준화 방향에 의해 판독 연산의 최근성 반영 여부와 기록 연산의 고려가 요구된다.

이에, 본 논문에서는 메타데이타의 속성을 정의하고 트랜잭션을 분류함으로써 판독 전용 트랜잭션의 일관성 기준을 약화시켜 판독 연산에 있어 빠른 응답시간과 최근 정보를 판독할 수 있는 방안을 제시하였다. 제안하는 알고리즘인 2VL은 2 버전을 래치를 사용하여 유지함으

로써 불필요한 로크의 소유를 제거하며 리프레쉬의 지연을 최소화하였다. 즉, 판독 전용 트랜잭션과 갱신 트랜잭션의 충돌을 최소화하여 질의에 대한 빠른 응답시간과 최근성 있는 정보를 제공할 수 있다. 따라서 기존의 Dublin 메타데이타의 관리뿐 만 아니라 현재 진행되는 새로운 표준화 형태의 메타데이타 관리에 있어서도 효율적인 성능을 낼 수 있다. 또한 전자 도서관의 메타데이타베이스의 속성을 갖는 환경의 데이타 관리에도 효율적인 성능을 나타낼 수 있을 것으로 기대된다.

한편, 제안하는 알고리즘은 메타데이타내에서 원소들 간의 의존성 때문에 페이지 단위의 로킹을 사용하였다. 그러나, 추후에 로킹 단위를 최소화하는 연구를 통해 실제 환경에 적용했을 때 더 효율적인 성능을 낼 수 있으리라 기대된다. 그리고, 전자 도서관의 병행수행 제어 기법의 적용은 각 사용자 계층에 따른 보안 사항도 함께 고려되어야 한다. 관리자 계층과 출판물 저작권 소유자 계층, 일반 사용자 계층의 분류에 따른 보안환경의 적용은 온라인상에서의 메타데이타 관리를 좀 더 용이하게 만들기 때문이다. 따라서, 메타데이타 관리의 병행수행 제어 기법의 적용과 더불어 이에 대한 보안 환경의 연구도 앞으로 요구된다.

참 고 문 헌

[1] Rust, Godfrey & Bide, Mark, "The <indecs> metadata model," <indecs> London conference, 1999.7.5. URL:http://www.indecs.org

[2] Rust Godfrey," INDECS Connections," 1999.2.16. URL:http://www.indecs.org

[3] Rust, Godfrey & Bide, Mark, "Introduction to the INDECS metadata schema," 1999.2.16. URL:http://www.indecs.org

[4] Renato iannella, "DC architecture WG proposal," 1999. URL:http://archive.dstc.edu.au/RDU/DCAC/arch-wg.html

[5] 좌은희, 박석, "디지털 정보에 대한 식별자 부여 및 전자 상거래용 메타데이타 모델에 관한 연구", KERIS 연구보고 RR 1999-2, 1999.

[6] 이해민, 박석, "디지털 도서관에서 래치에 기초한 메타데이타 관리 기법", 정보과학회논문지 제27권 제1호, pp.22-32, 2000.

[7] Mohan Kamath and Krithi Ramamritham, "Efficient transaction management & query processing in massive digital databases." Technical report 95-93 Dept. of C.S., University of Massachusetts, 1995.

[8] William Rosener, "Web Interfaces: The Benefits and Challenges Encountered in Developing Web-based Interfaces," AusWeb96, july 7-9, 1996.

[9] Alexa T. McCray, Marie E. Gallagher, Michael A. Flannick, "Extending the Role of Metadata in a Digital Library System," 1999.

[10] B. Claybrook. OLTP online transaction processing systems. Reading, MA: John Wiley & Sons, 1992.

[11] P. A. Bernstein, V. Hadzilacos and N. Goodman. Concurrency Control and Recovery in Database Systems. Reading, Addison-Wesley, 1987.

[12] Paul M. Bober and M. J. Carey, "Multiversion Query Locking," in Proc. of the VLDB, pp. 497-510, Aug. 1992.

[13] Hoewon Kim and Seog Park, "Two Version Concurrency control Algorithm with Query Locking for Decision Support," Proceedings of the International Workshops on Data Warehousing & Data Mining etc. (ER'98),pp153-164, Singapore, 1998.11.

[14] Vibby Gottemukkala, Tobin J. Lehman, "Locking and Latching in a Memory-Resident Database System," Proceeding of the 18th VLDB Conference Vancouver, British Columbia, Canada 1992.

[15] A. Alan B. Pritsker, Jean J. O'Reilly and David K. LaVal, "Simulation with Visual SLAM and AweSim," Systems Publishing Corporation, Indiana, 1997.

좌 은 희

1998년 동덕여대 전자계산학과(학사). 2000년 서강대 컴퓨터학과(공학 석사). 2001년 (주)데이콤 종합연구소 연구원. 현재 (주)퓨처시스템 정보통신 연구소 전임연구원. 관심분야는 디지털 도서관, 메타데이타, 트랜잭션 관리

박 석

정보과학회논문지 : 데이타베이스
제 29 권 제 1 호 참조