濟州大學校 白鹿論叢 第3集(2001. 5. 15)

가상대학을 지원하기 위한 로킹에 기반을 둔 트랜잭션 관리 기법

박 찬 정*
김 한 일**

目 次

Ⅰ. 서 론

최근, 인터넷의 폭발적인 보급으로 인해 인터넷은 모든 분야에서 정보전달의 수단뿐만 아니라 불특정 다수에 대한 통신 수단으로 넓게 이용되고 있다. 또한, 하이퍼텍스트(hypertext)를 기반으로 하는 웹(web)의 출현은 사회의 전 분야에 걸쳐 많은 변화를 촉진시키고 있다. 웹의 발전은 정보화로 인한 사회의 변화와 맞물려 전자상거래, 가상대학, 원격 진료 등 새로운 응용을 창출하고 있다(박찬정:2000). 특히, 교육 분야에서 사람들은 자신들이 원하는 시간에 학습을 하며 모든 과정에 참여하기보다는 자신에게 부족한 부분을 충족시킬 수 있는 부분에만 참여할 수 있는 수요자 중심의 학습 환경을 필요로 하게 되면서, 가상대학을 요구하게 되었다.

가상대학은 매우 체계적으로 설계되어야 하며, 많은 하드웨어 자원 및 네트워크 자원, 교수학습 자원들을 요구한다. 현재, 국내에서도 가상대학이라는 이름으로 서비스를 제공하는 사이트들이 증가하고 있는 추세이다(김성일:1998, 황대준:1998). 가상대학은 일부대학에서 일부과목에 대해서 시험적으로 운영되는 형태에서 벗어나 점차 정규과목으로 확대 운영되고 있다. 또한, 현재 국내외에서 약 230 여개의

* 컴퓨터교육과 전임강사
** 컴퓨터교육과 조교수

가상대학이 운영되고 있으며, 재교육 등 다양한 수요계층으로 말미암아 앞으로 가상대학에 대한 관심과 현재 제한되어 있는 교과목에 대한 개설도 크게 늘어나리라 전망된다(황대준:1998).

가상대학을 구현하기 위해서는 다양한 기술적 기반과 교육 방법 등이 제공되어져야 하며 또한 통합되어야 한다. 특히, 가상대학을 위한 기술적 기반으로서는 4계층이 필요하다(황대준:1998). 제일 하부 계층인 서비스 접근 계층은 학습자들이 교육 컨텐츠(content)와 같은 상위 계층에 접근할 수 있는 물리적인 환경을 제공한다. 즉, 전송 프로토콜과 이에 관련된 네트워크 기술 등이 이 계층에 포함된다. 서비스 생성 계층에서는 개별/그룹학습, 동기/비동기 교육 등 다양한 교육 방법과 교수-학생간의 상호작용을 지원한다. 콘텐츠 생성 계층에서는 다양한 저작도구를 관리하고 컨텐츠 데이터베이스 서버 관리 및 운영을 담당한다. 마지막으로, 학사 관리 계층에서는 입학, 등록, 수강에 관한 서비스를 담당한다(황대준:1998). 4 계층 중에서 가상대학이 기존의 면대면 형태의 수업을 대신하기 위해서는 무엇보다 다양한 컨텐츠가 요구될 것이고, 방대한 양의 컨텐츠를 효율적으로 관리하기 위한 데이터베이스 시스템의 도입이 필수적이라 할 수 있다.

일반적으로, 데이터베이스에 대한 논리적인 작업은 여러 개의 연산으로 구성된다. 이와 같은 여러 개의 연산들의 모임을 트랜잭션(transaction)이라고 하며, 데이터베이스 시스템의 중요한 기능 중에 하나가 바로 트랜잭션의 관리라 할 수 있다. 지금까지, 트랜잭션의 관리를 위한 연구들은 활발히 진행되고 있지만, 가상대학을 구현하기 위해서는 여러 가지 특성들, 즉, 온라인성(on-line), 실시간성(real-time), 상호작용성(interactivity), 원거리 접근성(remote accessibility) 등이 고려되어야 하며, 이러한 특성들이 고려된 새로운 트랜잭션 관리 기법의 개발이 요구되고 있다(전우천:1998). 특히, 많은 양의 데이터에 대해 판독만을 수행하는 긴 판독-전용(long read-only) 트랜잭션들이 많이 발생한다. 하지만, 일반적인 트랜잭션 기법에서는 판독-전용 트랜잭션을 갱신 트랜잭션(update transaction)과 구별하여 처리하고 있지 않아서 성능면에서 문제점들이 야기되어지며 이를 위한 기법이 요구되고 있다.

본 논문에서는 가상대학을 지원하는 트랜잭션 관리 기법 중에서 로킹(locking)에 기반을 둔 새로운 프로토콜을 제안한다. 제안하는 프로토콜은 판독-전용 트랜잭션을 갱신 트랜잭션과 구별하며, 트랜잭션의 지연시간을 최소화하기 위해 각 트랜잭션의 고유 버전 선택 시점을 정의한 후, 이를 이용하여 트랜잭션들이 데이터 접근 시 충돌로 인해 지연될 수 있는 요소를 제거한 마크라는 방법을 제안한다. 그리고 새로운 버전 선택의 규칙과 다중 버전을 관리하기 위한 버전 관리 기법을

제안한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 가상대학을 구현하기 위해 일반적으로 요구되는 트랜잭션의 요구 사항들을 제시하고, 제 3 장에서는 이 요구사항들을 기반으로 트랜잭션의 고유한 버전 선택 시점을 정의한 후 제안하고자 하는 프로토콜의 특성과 규칙들을 제안한다. 제 4 장에서 제안한 기법의 정확성을 증명한 후, 제 5 장에서는 본 논문의 결론과 앞으로의 연구 과제를 제시한다.

Ⅰ. 관련 연구

1. 가상대학에서의 트랜잭션 요구사항

궁극적으로, 가상대학의 구현에 요구되는 사항은 실시간 교육이라 할 수 있다(김성일:1998, 황대준:1998). 즉, 인터넷 등 통신망을 이용해서 양방향으로 학습자가 교수나 교육 콘텐츠와의 실시간 접속을 필요로 한다. 또한, 가상대학의 구현은 상호작용성(interactivity) 을 필요로 한다. 즉, 멀티미디어와 학습자간의 상호작용, 학습자와 학습자 사이, 학습자와 교수 사이의 상호작용을 필요로 한다. 특히, 멀티미디어와의 상호작용은 단순히 정보를 전달 받는 차원을 넘어서 피드백(feed-back)을 이용해 학습 목표를 향해 갈 수 있기 때문에 매우 중요하다(김성일:1998). 가상대학은 원격교육에 기초를 두었기 때문에 필연적으로 원격 데이터베이스 접근(remote database access)을 필요로 한다. 원격 데이터베이스 접근은 통신망을 이용하여 구현될 수 있고 이미 표준화된 서비스가 제공되고 있다(박정선:1996).

한편, 가상대학을 지원하는 데이터베이스에서는 사용자가 데이터에 접근할 때 데이터간의 관계를 이용하는 항해(navigation)를 이용하게 되고 멀티미디어 데이터의 용량의 크기로 인해서, 데이터베이스에 접근하는 트랜잭션의 처리 시간이 길어진다(Kim:1990). 가상대학 구현을 위한 트랜잭션이 만족해하는 가장 큰 조건은 트랜잭션의 길이가 유연해야 한다. 즉, 가상대학의 트랜잭션들은 크기가 큰 멀티미디어 정보를 이용하고 또한 항해를 이용하기 때문에 길이가 길어진다(전우천:1998).

또한, 가상대학의 트랜잭션은 실시간 (real-time)으로 학습자와 학습자. 또는 학습자와 교수 사이에 공동작업을 지원해야 한다. 즉, 공유한 데이터를 서로 주고받

으면서 작업을 완성할 수 있고 또한 공동의 작업을 하는 도중 작업을 서로 나누어 각자 해야 하는 등 하나의 큰 작업을 완성하기 위한 협력 작업을 지원해야한다(전우천:1998).

2. 정확성 기준

전통적인 데이터베이스 시스템을 위한 논리적 일관성 기준은 직렬성(serializability)이다. 임의의 두 트랜잭션 T_i와 T_j의 연산으로 이루어진 수행기록 H에서 T_i의 연산이 T_j의 연산을 모두 선행하거나 모두 후행하면 그 수행기록은 '직렬(serial)이다'라고 한다. 또한, T_i와 T_j가 병행수행하여 생성된 수행기록 H'의 결과가 두 트랜잭션이 직렬로 수행된 결과들 - 예를 들면, T_iT_j 혹은 T_jT_i - 중에 하나와 일치일 때, H'는 직렬적(serializable)이라고 한다(Bernstein:1987).

한편, 다중버전을 위한 정확성 기준은 단일-복사본(one-copy) 직렬성으로, 다중버전 수행기록인 H와 임의의 두 트랜잭션 T_i, T_j에 대해서, 연산 $r_i[x_j]$가 H에 존재할 때 - 즉, T_j가 기록한 x의 버전 x_j를 T_i가 판독하였을 때 - 항상 T_j가 T_i이거나 혹은 가장 최근에 x를 생성한 트랜잭션이라면 H는 1-직렬이라고 하고, 다중버전 수행기록 H가 단일-복사본 직렬적(one-copy serializable:1SR)이라는 것은 H가 특정한 1-직렬인 다중버전 수행기록 H'과 동치[1]일 때를 의미한다(Bernstein: 1987).

3. 로킹에 기반을 둔 다중버전 병행수행 제어

다중버전에 기초를 둔 프로토콜들 중에서 다중버전 2 단계 로킹(multiversion two phase locking:MV2PL) 프로토콜이 있다(Bernstein:1983). 다중버전 2 단계 로킹 프로토콜은 판독-기록 연산간의 충돌과 기록-기록 연산간의 충돌을 제거하는 대신, 공인(certifying) 로크라는 새로운 로크를 정의하였다. 따라서, 각 데이터 항목은 여러 개의 비공인(uncertified) 버전을 갖게 되지만, 단일-복사본 직렬성을 만족시키기 위해서 트랜잭션들은 항상 가장 최근에 공인된 데이터 항목들만을 판독한다. 트랜잭션들이 가장 최근에 공인된 데이터 항목들만을 판독하는 다른 이유는 버전의 수를 최소화시키기 위한 것이다.

---

1) 임의의 다중버전 수행기록 H,H'에 대해서 두 수행기록이 모두 같은 연산들로 구성되어 있고 같은 기록-관계를 가질 때, H와 H'은 동치(equivalent)이다.

T_R \ T_H	R	W	C
R	Y	Y	N
W	Y	Y	Y
C	N	Y	N

T_H : 로크 소유 트랜잭션
T_R : 로크 요그 트랜잭션
R : 판독연산
W : 기록연산
C : 공인연산
Y : 공유 가능
N : 공유 불허

표 1 MV2PL의 호환성 표

표 1과 같이 어떤 트랜잭션이 데이터를 판독하고 있는 경우, 다른 트랜잭션들은 같은 데이터 항목에 대해서 공인 연산을 수행할 수 없다. 하지만, 만일 각 트랜잭션에 대해서 적절한 버전 선택 시점을 설정하여 준다면, 판독-공인 연산간의 충돌은 제거할 수 있다. 다음 장에서는 충돌을 제거하는 방법을 제안한다. 질의가 많은 트랜잭션 처리 시스템의 경우에는 기록 트랜잭션의 수행에 영향을 적게 받으면서 지연되지 않고 질의를 처리할 수 있게 되어 성능을 향상시킬 수 있다.

Ⅲ. 로킹에 기반을 둔 프로토콜

이 장에서는 제안하는 프로토콜의 특성과 규칙을 기술하고, 제안하는 프로토콜을 병행수행 제어 알고리즘으로 사용하는 경우에 데이터 관리자가 버전을 효율적으로 관리할 수 있는 기법에 대해서 기술한다.

1. 프로토콜의 특성 및 규칙

다음 그림 1과 같은 다중버전 2 단계 로킹 알고리즘에 의해 수행된 수행기록을 살펴보자. 시점 3에서 트랜잭션 T_2가 공인연산을 수행하려고 할 때, T_1에 의해 지연된다. 즉, T_2는 T_1이 종료된 후, 시점 6에서 완료된다. 하지만, 이와 같은 T_2의 지연은 불필요한 것이다. 즉, T_2가 시점 3에서 종료를 하여도 직렬성을 위배하지 않고 T_1, T_2의 순서로 트랜잭션을 수행시킨 결과와 동치이다.

트랜잭션 \ 시간	1	2	3	4	5	6
T_1	$r_1[x_0]$			$r_1[y_0]$	c_1	
T_2		$w_2[x_2]$	c_2 blocked	blocked	blocked	c_2

그림 1 수행기록의 예제

본 논문에서는 위와 같은 불필요한 지연을 없애고 트랜잭션간의 판독-기록 충돌을 제거하기 위해서 각 트랜잭션 T의 버전 선택 시점을 정의하고 이를 이용하여 병행수행 제어를 하고자 한다.

【정의 1】 임의의 트랜잭션 T가 한 데이터 항목 x를 판독하고 있을 때, 다른 트랜잭션 T′에 의해 공인 로크를 요청받아 트랜잭션 관리자가 공인 로크를 허용하거나 또는 T가 x를 판독하려고 할 때 이미 다른 트랜잭션 T′에 의해 공인 로크가 설정되어 있지만 트랜잭션 관리자가 판독 로크를 허용할 때 T는 마크(marked)된다고 한다.

【정의 2】 각 트랜잭션 T에 대해서, T가 판독할 데이터 항목들의 버전선택 시점(version selection point)을 VSP(T)라고 정의하고 이 때, VSP(T)는 $+\infty$를 초기값으로 갖는다. T가 마크될 때, VSP(T)는 $+\infty$가 아닌 양의 정수 값을 갖는다.

각 트랜잭션 T에 대해서, VSP(T)는 다음과 같은 성질들을 갖는다.

【성질 1】 T가 한 데이터 항목 x에 대해서 판독 로크를 획득하려 할 때, 다른 활성(active) 트랜잭션 T′이 x에 대한 공인 로크를 소유하여 T가 마크되고, 이 충돌이 T의 첫 번째 충돌이라면, 판독 로크를 획득하려 한 시점을 VSP(T)의 값으로 할당한다. 만일, 첫 번째 충돌이 아니거나 충돌이 발생하지 않는다면, VSP(T)는 변경되지 않는다.

【성질 2】 T가 임의의 데이터 항목 x에 대해서 공인 로크를 획득하려 할 때, 다른 활성 트랜잭션들 T′들이 이미 판독 로크를 획득하고 있어 T′이 마크되고 T′에 대해 이 충돌이 첫 번째 충돌이라면, T가 공인 로크를 획득하려 한 시점을 VSP(T′)들의 값으로 할당한다. 충돌이 발생하지 않는 경우, VSP(T′)들은 변경되지 않는다.

【성질 3】 T는 여러 번 마크될 수 있으나 VSP(T)는 마크될 때마다 갱신되는 것이 아니고, 현재 VSP(T)값과 현재 마크된 시점을 비교하여 가장 작은 값을 갖는다. 즉, VSP(T) = min(VSP(T), 현재 마크된 시점)이 된다. VSP(T)는 T가 마크되지 않으면 항상 $+\infty$ 값을 유지하며, 여러 번 마크되는 경우에는 첫 번째 마크된 시점을 가지고 있다.

예를 들면, 다음 그림 2와 같은 수행기록에서 T_1은 T_2에 의해서 마크되고, $VSP(T_1) = 4$가 되고 $VSP(T_2) = +\infty$가 된다.

트랜잭션 \ 시간	1	2	3	4	5	6
T_1	$r_1[x_0]$				$r_1[y_0]$	c_1
T_2		$r_2[x_0]$	$w_2[x_2]$	c_2		

그림 2 VSP(T)에 대한 예제 수행기록

제안하는 프로토콜을 위해서 각 트랜잭션 T는 자신이 판독할 데이터 항목들의 집합인 판독집합 R(T)와 자신이 기록할 데이터 항목들의 집합인 기록집합 W(T)를 가진다고 가정한다. 그러면, 임의의 두 트랜잭션 T_i와 T_j의 관계를 그들의 판독집합과 기록집합에 따라서 다음과 같이 구분할 수 있다.

【경우 1】 $R(T_i) \cap R(T_j) \neq \emptyset$

	1	2	3	4	5	6
T_i	$r_i[x_k]$		$w_i[z_i]$		c_i	
T_j		$w_j[x_n]$		$w_j[y_j]$		c_j

그림 3 $R(T_i) \cap R(T_j) \neq \emptyset$인 예제 수행기록

판독집합만 공통 데이터 항목을 갖는 경우는 충돌을 일으키지 않기 때문이 아무 문제가 없다. 즉, 위의 그림 3과 같이 트랜잭션들은 자신의 VSP보다 이전에 생성된 버전중 가장 최근에 생성된 데이터 항목의 버전을 판독하고 기록 연산을 수행하면 된다. 만일, VSP 값이 $+\infty$라면, 이는 현재 시간을 나타낸다. 따라서, 가장 최근에 생성된 버전으로 선택을 하면 된다.

【경우 2】 $R(T_i) \cap W(T_j) \neq \emptyset$ (혹은, $W(T_i) \cap R(T_j) \neq \emptyset$)

	1	2	3	4	5	6
T_i	$r_i[x_k]$		$w_i[z_i]$			c_i
T_j		$w_j[x_j]$		$w_j[y_j]$	c_j	

그림 4 $R(T_i) \cap W(T_j) \neq \emptyset$인 예제 수행기록

그림 4에서 만일, T_i가 T_j보다 먼저 완료한다면, 충돌이 발생하지 않아서 문제가 없다. 하지만, T_j가 먼저 완료하는 경우에는 T_i는 T_j에 의해 마크된다. 그러면, T_i의 $VSP(T_i)$는 T_j의 공인 요청시간인 5로 설정된다. 그리고 T_i가 $VSP(T_i)$ 이전에 생성된 데이터 항목의 버전들만을 판독하면 직렬성을 위배하지 않고 병행수행이 가능하다. 기록 연산끼리는 문제를 발생시키지 않는다.

【경우 3】 $W(T_i) \cap W(T_j) \neq \emptyset$

	1	2	3	4	5	6
T_i	$r_i[x_k]$		$w_i[z_i]$		c_i	
T_j		$w_j[y_n]$		$w_j[z_j]$		c_j

그림 5 $W(T_i) \cap W(T_j) \neq \emptyset$인 예제 수행기록

위의 그림 5와 같이 다중버전의 경우, 기록과 기록간의 충돌은 존재하지 않는다. 따라서, 아무 문제도 발생시키지 않는다.

【경우 4】 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $R(T_i) \cap W(T_j) \neq \emptyset$

	1	2	3	4	5	6	7
T_i	$r_i[x_k]$					$r_i[z_?]$	c_i
T_j		$r_j[x_n]$	$r_j[z_0]$	$w_j[z_j]$	c_j		

그림 6 $R(Ti) \cap R(Tj) \neq \emptyset$ 이고 $R(Ti) \cap W(Tj) \neq \emptyset$인 예제 수행기록

그림 6에서 x에 대해서 T_i와 T_j가 같은 버전을 판독할 수도 있고 그렇지 않을 수도 있다. 또한, 이 경우에는 T_j가 T_i를 마크시킬 수 있다. T_i는 자신이 마크되었

는지의 여부에 따라서 자신의 VSP(T_i)를 설정하고, 판독 시에는 VSP(T_i) 이전에 생성된 버전들만을 판독하면 직렬성을 보장할 수 있다.

【경우 5】 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$

	1	2	3	4	5	6	7
T_i	$r_i[x_t]$					$w_i[y_?]$	c_i
T_j				$r_j[x_k]$	$w_j[y_j]$		
T_k		$r_k[x_k]$	c_k				

그림 7 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$인 예제 수행기록

그림 7에서 T_i와 T_j가 같은 데이터 버전을 판독한 경우에는 문제가 발생하지 않지만, 위의 그림에서와 같이 제 3의 트랜잭션에 의해 간접 싸이클이 발생될 수 있다. 즉, T_i가 마크되었는지의 여부에 따라서 T_j가 선택하는 버전이 달라진다. 만일, T_i가 마크되지 않았으면, T_j의 VSP(T_j)는 VSP(T_i)에 영향 받지 않으나, T_i가 마크되었다면, VSP(T_j)는 VSP(T_i)의 값을 상속받는다. 위의 경우에서는 T_j는 T_i가 판독한 버전을 선택한다. 즉, k = t이다.

【경우 6】 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap R(T_j) \neq \emptyset$

	1	2	3	4	5	6
T_i	$w_i[y_i]$				$r_i[x_j]$	c_i
T_j		$r_j[y_0]$	$w_j[x_j]$	c_j		

(가) 데드록이 발생하지 않은 경우

	1	2	3	4	5	6
T_i	$r_i[x_0]$			$w_i[y_i]$		c_i (취소)
T_j		$r_j[y_0]$	$w_j[x_j]$		c_j	

(나) 데드록이 발생한 경우

그림 8 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap R(T_j) \neq \emptyset$인 예제 수행기록

이 경우는 MV2PL 규칙을 적용했을 때 데드록이 발생할 수 있는 경우이다. 그림 8 (가)의 경우에는 T_i가 T_j에 의해 마크되지 않고 수행된다. 따라서, T_j가 완료한 후, T_i는 Tj가 기록한 x의 버전을 판독한다. 반면, 그림 8 (나)의 경우는 T_i가 T_j에 의해서 마크되고, MV2PL 프로토콜 규칙을 적용시키면 데드록이 발생하는 경우이다. 즉, T_i 혹은 T_j를 취소시키지 않으면 직렬성을 보장할 수 없게 된다. 따라서, 이 경우에는 T_j를 완료시키고, T_j를 완료시키는 시점에서 T_i를 취소시킴으로써 데드록을 제거하고 직렬성도 보장한다.

【경우 7】 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$

이 경우에도 T_i가 T_j에 의해 마크되지 않는다면 문제가 없으나 그림 9와 같이 시점 4에서 T_i가 T_j에 의해서 마크되는 경우에는 T_i의 기록 연산들 중에서 T_j와 공통으로 기록하는 데이터 항목에 대한 연산은 T_i T_j 순서로 수행시켰을 때 덮어쓰기를 당하기 때문에 무시할 수 있다(블라인드 기록).

	1	2	3	4	5	6
T_i	$r_i[x_0]$				$w_i[y_i]$ (무시)	c_i
T_j		$w_j[x_j]$	$w_j[y_j]$	c_j		

그림 9 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$인 예제 수행기록

【경우 8】 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$

	1	2	3	4	5	6	7	8
T_i	$r_i[x_0]$	$r_i[y_k]$					$w_i[y_i]$ (무시)	c_i
T_j			$r_j[y_k]$	$w_j[x_j]$	$w_j[y_j]$	c_j		

그림 10 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$인 예제 수행기록

이 경우는 T_i가 기록하는 데이터 항목을 T_j는 판독하지 않는다. 이는 【경우 5】에서처럼 T_i가 마크된 트랜잭션인가의 여부에 따라 선택이 달라진다. 즉, T_i와 T_j가 공통으로 판독하는 데이터 항목에 대해서 만일, T_i가 마크된 트랜잭션이라면

VSP(T_j)는 VSP(T_i)를 상속받는다. 그리고 【경우 7】과 같이 만일, T_i가 T_j에 의해 마크된다면, T_i와 T_j가 공통으로 기록하는 데이터 항목에 대한 T_i의 기록연산은 무시된다. (그림 10 참조)

【경우 9】 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap R(T_j) \neq \emptyset$

	1	2	3	4	5	6	7
T_i	$r_i[y_k]$					$w_i[z_i]$ (취소)	c_i (취소)
T_j		$r_j[y_k]$	$w_j[y_j]$	$w_j[z_n]$	c_j		

그림 11 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap R(T_j) \neq \emptyset$인 예제 수행기록

【경우 4】와 【경우 6】을 함께 고려한 경우로써 MV2PL 규칙을 적용시키면, 경우 6과 마찬가지로 데드록을 발생시킨다. 두 개의 트랜잭션을 모두 수행시키면 직렬성을 보장할 수 없기 때문에 만일, 한 트랜잭션이 다른 트랜잭션을 마크시키는 경우에는 먼저 완료하는 트랜잭션을 제외한 나머지 트랜잭션을 취소시킨다. (그림 11 참조)

【경우 10】 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap R(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$

	1	2	3	4	5	6	7	8
T_i	$r_i[x_k]$					$w_i[x_i]$	$w_i[y_i]$	c_i
T_j		$r_j[y_s]$	$w_j[y_j]$	$w_j[x_n]$	c_j			

그림 12 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap R(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$인 예제 수행기록

Ti와 Tj가 상대방에 의해 마크되지 않는다면, 문제가 없지만 그렇지 않다면, 【경우 6】과 마찬가지로 데드록으로 처리하여 먼저 완료하는 트랜잭션의 완료시점에서 나머지 트랜잭션을 취소시킴으로써 직렬성을 보장한다. (그림 12 참조)

【경우 11】 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고 $W(T_i) \cap R(T_j) \neq \emptyset$

이고 $W(T_i) \cap W(T_j) \neq \emptyset$

	1	2	3	4	5	6	7	8	9	10
T_i	$r_i[x_k]$	$r_i[y_n]$						$w_i[x_i]$	$w_i[y_i]$	c_i
T_j			$r_j[x_m]$	$r_j[y_s]$	$w_j[y_j]$	$w_j[x_t]$	c_j			

그림 13 $R(T_i) \cap R(T_j) \neq \emptyset$ 이고 $R(T_i) \cap W(T_j) \neq \emptyset$ 이고
$W(T_i) \cap R(T_j) \neq \emptyset$ 이고 $W(T_i) \cap W(T_j) \neq \emptyset$인 예제 수행기록

【경우 5】에서처럼, T_i가 마크되었는지의 여부에 따라서 T_j가 선택하는 버전이 달라진다. 만일, T_i가 마크되지 않았으면, T_j의 $VSP(T_j)$는 $VSP(T_i)$에 영향받지 않으나, T_i가 마크되었다면, $VSP(T_j)$는 $VSP(T_i)$의 값을 상속받는다. 위의 경우도 【경우 10】과 마찬가지로 그림과 같이 T_i의 늦은 기록연산으로 인해 데드록이 발생되면 Ti를 취소시킴으로써 직렬성을 보장한다.

모든 경우를 통합하여 프로토콜을 요약하면, 다음과 같이 기술되어지고 알고리즘은 그림 14에 기술되어 있다.

(1) 트랜잭션 T는 수행의 시작 시, 판독집합인 R(T)와 기록집합인 W(T)를 부여받는다.

(2) T의 버전 선택 시점인 VSP(T)는 $+\infty$를 초기값으로 갖는다.

(3) 마크된 트랜잭션 목록(marked transaction list)인 MTL은 공집합으로 초기화되며, 트랜잭션이 마크되면 트랜잭션의 식별자를 원소로 갖는다.

(4) 모든 연산은 먼저 로크를 획득한 후 연산을 수행하고, 한번 로크를 해제하면 다시 로크를 획득할 수 없다.

(5) 판독과 다른 연산(판독, 기록, 공인)간, 기록-기록, 기록-공인 연산간에는 데이터 공유로 인한 충돌이 발생하지 않는다. 단, 공인-공인 연산간에는 충돌이 존재한다.

(6) 트랜잭션 T가 임의의 데이터 항목에 대해 판독연산을 수행할 때, 항상 VSP(T) 이전에 생성된 버전들 중에서 가장 최근에 생성된 버전을 판독한다. VSP(T)가 $+\infty$일 때는 현재 시점을 기준으로 가장 최근에 생성된 버전을 판독한다.

(7) 트랜잭션 T가 임의의 데이터 항목에 대해 기록연산을 수행할 때, 지연되지 않고 자신의 지역영역(local space)에 새로운 버전을 기록한다. 완료되기 전에 T가 기록한 데이터 항목들에 대해서는 공인 연산을 수행한다.

(8) 트랜잭션 T가 다른 트랜잭션 T′에 의해 마크를 당하면, VSP(T)값을 정의에 따라서 갱신시키고, 다음과 같은 사항을 점검한다.

(i) $R(T) \cap W(T') \neq \emptyset$이고 $W(T) \cap R(T') \neq \emptyset$: T의 판독집합과 T를 마크한 T′의 기록집합간에 공집합이 존재하고 T의 기록집합과 T′의 판독집합간에 공집합이 존재하면, MV2PL 규칙을 적용시킨 경우, 데드록이 발생한 상태이며, 두 개의 트랜잭션을 모두 완료시키면 직렬성을 위반하게 된다. 따라서, 먼저 공인연산을 수행한 트랜잭션은 완료시키고, 나머지 활성중인 트랜잭션은 취소시킨다. 즉, 이 프로토콜은 데드록 문제를 해결한다.

(ii) $R(T) \cap W(T') \neq \emptyset$이고 $W(T) \cap W(T') \neq \emptyset$: T의 판독집합과 T′의 기록집합간에 공집합이 존재하고 두 트랜잭션이 특정 데이터 항목에 대해 공통으로 기록연산을 갖는다면, T→T′의 순서로 트랜잭션을 수행시킨 결과를 갖도록 처리한다. 즉, T→T′으로 수행이 된다면, T와 T′이 공통으로 기록하는 데이터 항목에 대해서는 T′의 기록연산으로 인해 T의 기록연산은 덮어 쓰기를 당한다. T가 T′에 의해 마크되었기 때문에, T의 기록연산은 시간적으로 늦은(late) 기록연산이 되고 이는 무시될 수 있다. 따라서, 마크된 트랜잭션 T와 T′이 공통으로 기록하는 데이터 항목에 대해서, T의 기록을 무시한다. 그리고 T를 MTL에 삽입한다.

(9) 트랜잭션 T가 시작할 때, MTL에 존재하는 트랜잭션인 T′들과 다음 조건을 확인한다. 만일, T와 T′이 특정 데이터 항목을 공통으로 판독하고 또 다른 - 같을 수도 있음- 데이터 항목들을 공통으로 기록한다면, 가장 작은 값을 갖는 VSP(T′)이 VSP(T)로 상속된다. 그리고, T 자신도 MTL에 삽입된다.

(10) 트랜잭션 T가 완료할 때, 만일 T가 마크된 트랜잭션이라면, MTL에서 자신의 식별자를 제거한다.

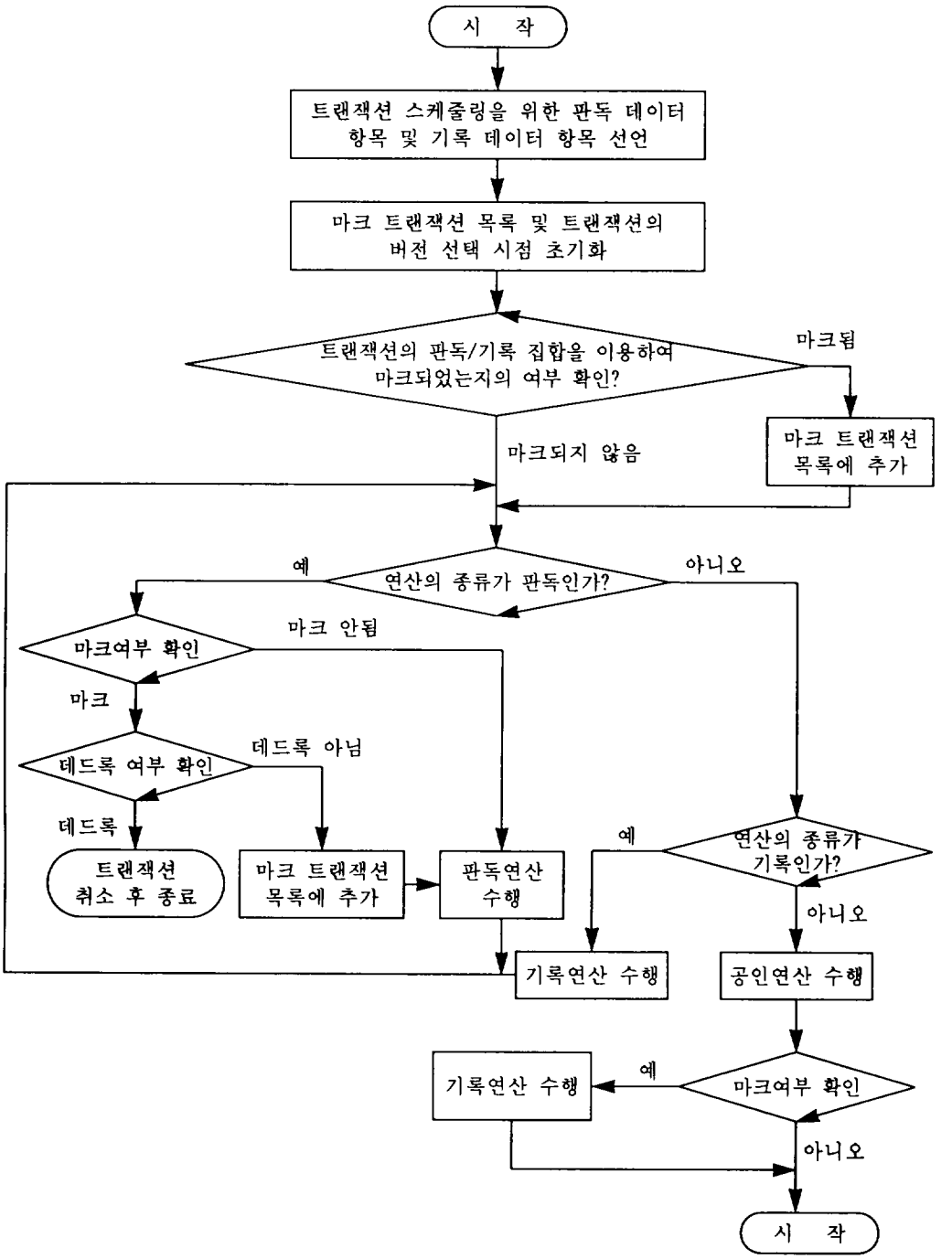

시 작
트랜잭션 스케줄링을 위한 판독 데이터 항목 및 기록 데이터 항목 선언
마크 트랜잭션 목록 및 트랜잭션의 버전 선택 시점 초기화
트랜잭션의 판독/기록 집합을 이용하여 마크되었는지의 여부 확인?
마크됨
마크 트랜잭션 목록에 추가
마크되지 않음
예
연산의 종류가 판독인가?
아니오
마크여부 확인
마크 안됨
마크
데드록 여부 확인
데드록 아님
데드록
트랜잭션 취소 후 종료
마크 트랜잭션 목록에 추가
판독연산 수행
예
연산의 종류가 기록인가?
아니오
기록연산 수행
공인연산 수행
기록연산 수행
예
마크여부 확인
아니오
시 작

그림 14 알고리즘

2. 버전 관리 기법

이 장에서는 이 전장에서 기술한 프로토콜을 병행수행 제어 알고리즘으로 채택하였을 때, 생성되는 데이터 버전을 효율적으로 관리하기 위한 기법에 대해서 기술한다. 임의의 데이터 항목 x의 버전을 관리하기 위하여 다음 그림 15와 같은 자료구조를 정의한다. 즉, 버전과 버전간은 포인터로 연결되고 value는 데이터 값을 저장하기 위한 필드이며, count는 현재 몇 개의 활성 트랜잭션들이 판독하고 있는가를 나타낸다. Timestamp는 버전이 생성된 시간을 나타내고 tlist는 해당 버전을 판독하고 있는 트랜잭션들의 목록으로 트랜잭션들의 식별자를 갖는다.

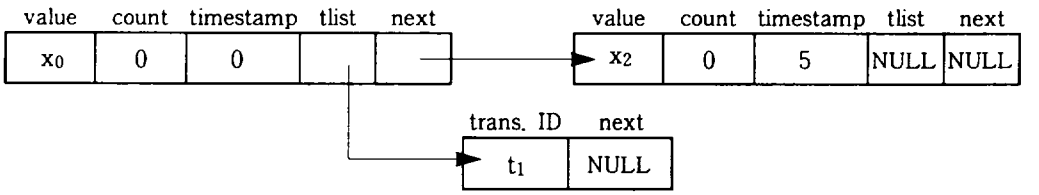
	value	count	timestamp	tlist	next
데이터 항목 x	x_0	0	0	NULL	NULL

그림 15 버전을 관리하기 위한 자료구조: 데이터 항목 x에 대한 초기 버전

만일, 트랜잭션들끼리 마크시키지 않는다면, 저장 공간에서는 항상 하나의 버전만을 유지할 것이다. 하지만, 마크된 트랜잭션이 발생하면, 두 개 이상의 버전들이 생성된다. 예를 들면, 다음 그림 16의 (가)와 같은 수행기록이 있다면, 시점 5에서 x에 대한 버전은 그림 16의 (나)와 같다.

	1	2	3	4	5	6
T_1	$r_1[x_0]$		$w_1[z_1]$			c_1
T_2		$w_2[x_2]$		$w_2[y_2]$	c_2	

(가) 예제 수행기록



(나) 데이터 항목 x에 대한 버전

그림 16 버전 생성의 예제

시점 5에서 이전 버전인 x0를 판독하는 트랜잭션들의 식별자는 MTL에 삽입되어진다. tlist 필드를 구현하는 방법은 위 그림 16과 같이 포인터를 사용할 수도 있고 비트 벡터(bit vector)를 사용할 수도 있다. 데이터 관리자는 다음과 같은 절차로 데이터 버전을 관리한다.

(1) 트랜잭션 T가 임의의 데이터 항목 x에 대해 판독연산을 요청한 경우, 데이터 관리자는 T의 VSP(T)와 x의 timestamp 필드 값을 이용하여 timestamp 〈 VSP(T)인 버전들 중에서 가장 큰 timestamp를 갖는 버전을 선택한다.

(2) 기록연산에 의해 트랜잭션 자신의 지역적 공간에 임시적으로 데이터를 기록하고 공인 연산의 수행 후에 다른 트랜잭션들이 판독할 수 있도록 저장공간에 저장한다.

(3) 트랜잭션 T가 자신이 기록했던 데이터 항목 x에 대한 공인연산을 요청한 경우, 버전 중에서 count 필드의 값이 0인 버전이 존재하는지를 확인한다. 존재하면, 그 버전은 제거되고, 그 버전의 저장 공간에 새 버전의 내용으로 갱신한다. 만일, count가 0인 버전이 없다면, 다른 트랜잭션들에 의해 버전들이 판독되고 있는 상태이므로, 새로운 공간을 할당받아 새 버전을 저장한 후 기존 버전들의 가장 마지막 부분으로 추가시킨다.

(4) 한 트랜잭션들이 완료되면, 그 트랜잭션이 판독했던 데이터 항목들의 버전에 대해서 count 필드 값을 하나 감소시킨다.

위의 연산 (3)에서 트랜잭션 T가 데이터 항목 x의 새로운 버전을 생성하였을 때, count가 0인 버전 x′들이 없었다면, x′을 판독하고 있는 트랜잭션 T′들은 T에 의해 마크를 당한다. 따라서 T′들은 MTL에 삽입된다.

Ⅳ. 정확성의 증명

이 장에서는 제안한 프로토콜의 정확성을 증명한다. 제안하는 프로토콜은 다중버전 2 단계 로킹 프로토콜에 기반을 두고 있기 때문에 정확성을 증명하는 경우 2장에서 기술하였던 단일-복사본 직렬성을 만족함을 보이면 된다.

【정리 1】 임의의 다중버전 수행기록 H가 있을 때, 임의의 한 버전순서 ≪가 존재하여, H에 대한 MVSG(H,≪)[2]가 비순환적이라면, H는 단일-복사본 직렬적이다(Bernstein:1987). □

【정리 2】 제안된 프로토콜에 의해 생성된 모든 수행기록들은 단일-복사본 직렬적이다.

증명: T = { T_1, T_2, …, T_n }을 트랜잭션들의 집합이라고 할 때, T에서 정의된 트랜잭션들이 제안된 프로토콜에 의해 수행되어 생성된 수행기록을 H라고 하고 트랜잭션 T_i의 공인연산을 cr_i라고 나타내자. 그리고 버전순서 $x_i \ll x_j$는 $cr_i < cr_j$ 연산을 함축한다고 정의하자. 우선, T_i가 선택하는 버전들을 살펴보자. 만일 T_i가 마크되지 않았다면, 버전을 선택하는 규칙은 다중버전 2 단계 로킹 프로토콜과 같이 가장 최근에 완료된 버전을 선택한다. 그렇지 않다면, T_i가 판독하는 버전은 VSP(T) 이전에 생성된 것들이다.

둘째, MVSG(H, ≪)에 속하는 모든 에지 $T_i \rightarrow T_j$에 대해서, 그들이 공인 순서로 되어 있음을 보임으로써 MVSG(H, ≪)가 비순환적임을 증명하면 된다. 즉, $T_i \rightarrow T_j$이면, $cr_i < cr_j$임을 보인다. 우선, $T_i \rightarrow T_j$가 직렬화그래프 SG(H)[3]에 속하는 에지라고 가정하자. 이 에지는 판독관계를 나타낸다. 그러면, 모든 트랜잭션들은 항상 공인된 데이터 버전을 판독하기 때문에 $cr_i < cr_j$이다.

다음은 서로 상이한 i, j, k에 대해서 H의 연산 $w_i[x_i]$와 $w_j[x_j]$, $rk[x_j]$를 고려해 보자. 그러면, 두가지 경우가 가능하다: (i) $x_i \ll x_j$와 (ii) $x_j \ll x_i$이다. 첫 번째 경우는 MVSG(H, ≪)에 $T_i \rightarrow T_j$버전순서 에지를 추가시킨다. 그러므로 버전순서 ≪의 정의에 의해서 $cr_i < cr_j$이다. 두 번째 경우는 MVSG(H, ≪)에 $T_k \rightarrow T_i$로의 버전순서 에지를 추가시킨다. 그러면, 【성질 4】에 의해서 $cr_i < cr_j$ 이거나 $cr_k < cr_i$이다. 첫 번째 경우는 버전순서 ≪의 정의에 의해서 $cr_j < cr_i$를 의미하기 때문에 불가능하다. 그러므로 $cr_k < cr_i$이다. MVSG(H, ≪)에 속하는 모든 에지들이 공인연산 순서로 되어 있기 때문에 MVSG(H, ≪)는 비순환적이다. 따라서, H는 FR-직렬적이고 정리 1에 의해서 H는 단일-복사본 직렬적이다. □

---

2) 임의의 다중버전 수행기록 H와 버전 순서 ≪가 주어졌을 때, 다중버전 직렬화 그래프 MVSG(H, ≪)는 H에 속하는 완료 트랜잭션들을 노드로 갖고 충돌 에지와 버전 순서 에지라는 두가지 종류의 에지(edge)를 갖는 그래프이다. 트랜잭션 T_i에서 T_j로의 충돌 에지는 임의의 데이타 항목 x에 대해서 H가 $w_i[x_i]$와 $r_j[x_i]$ 연산을 포함하고 있을 때 발생한다. 서로 상이한 i, j, k에 대해서 임의의 두 연산 $rk[x_j]$와 $w_i[x_i]$가 H에 포함되어 있을 때 T_i에서 T_j로의 버전순서 에지는 $x_i \ll x_j$라면, 발생한다. 그렇지 않다면, T_k에서 T_i로의 버전순서 에지가 발생한다.

3) 수행기록 H의 직렬화 그래프 SG(H)는 방향성 그래프로서 트랜잭션들을 노드로 갖고 두 트랜잭션이 충돌관계를 가지면 충돌관계를 에지로 갖는다.

Ⅴ. 결론 및 연구과제

본 논문에서는 가상대학의 구현을 위한 학습 데이터베이스를 접근하여 트랜잭션을 처리를 위한 관점에서 필요한 일반적인 조건들과 특성들을 논의하고 이를 기반으로 가상대학을 지원을 위한 트랜잭션의 요구사항을 제시하였다. 또한, 이 요구사항들을 만족시킬 수 있는 트랜잭션 스케줄링 방법을 제안하였다. 제안한 프로토콜에서 각 트랜잭션들은 마킹이라는 기법을 통해서 트랜잭션들의 판독-기록간 충돌을 제거한다.

본 논문의 효과는 다음과 같다. 첫째, 기존의 상용 DBMS들은 로킹에 기반을 두고 있어 판독연산만을 수행하는 질의 트랜잭션들도 갱신연산을 함께 수행하는 트랜잭션들에 의해 불필요하게 지연될 수 있다. 하지만, 본 논문을 질의가 주를 이루는 가상학교나 의사결정 시스템 등에 도입하는 경우, 질의 트랜잭션들은 절대로 지연되지 않기 때문에 빠른 응답시간을 제공할 수 있어 성능의 향상을 가져오게 된다. 둘째, 본 논문에서 제안하는 마킹 기법은 향 후, 강제적 접근 제어 기법을 취하는 다단계 보안 데이터베이스 시스템을 위한 트랜잭션 기법에서도 사용될 수 있다는 확장성을 갖는다.

향 후, 본격적으로 운영될 가상대학을 위해 다음과 같은 연구가 필요하다. 우선, 현재의 연구에서는 상호작용성과 병렬성이 제한된 범위에서 이루어지고 있으나, 앞으로는 구체적인 학습요소의 필요에 따라 더 확대될 필요성이 있다(전우천:1998). 또한, 새로운 정확성 기준이 제공되어 현재의 정확성 기준 때문에 발생하게 되는 제한적인 면들을 제거함으로써 보다 효율적인 프로토콜에 대한 연구가 진행되어야 한다. 또한, 웹의 특성을 살리면서 실시간성이나 멀티미디어 데이터를 속성 등을 고려한 방법들이 제공되어야 하며, 데이터 보안성을 위한 고려도 함께 이루어져야 할 것이다.

참 고 문 헌

김성일, "가상대학의 당면과제와 운영방안", 한국정보과학회지, 제16권 제10호, 1998.

박정선, "원격 데이터베이스 접근", 데이터베이스 월드, 1996.

박찬정, 웹-기반 가상대학에서 사용자 인증을 위한 인증시스템 구축 방안 연구 인증, 백록논총, 제주대학교, 2000.

전우천, 홍석기, "가상대학을 지원하기 위한 트랜젝션 모형", 한국정보교육학회논문지, 제2권 제2호, 1998.

황대준, "가상대학의 현황과 발전방향", 한국정보과학회지, 제16권 제10호, 1998.

Bernstein, P. A. and N. Goodman, "Multiversion Concurrency Control - Theory and Algorithms," ACM Transactions on Database Systems, Vol. 8 No. 4, December 1983.

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

Kim, W., Introduction to Object-Oriented Databases, The MIT Press, 1990.

Abstract

A Locking-based Transaction Management Scheme for Supporting Virtual Universities

Chan-Jung Park
Han-Il Kim

With the advance of computer and communication technologies, a lot of organizations operate virtual universities that provide learning environments out of temporal and spatial constraints and many research works on the web-based virtual universities have been proposed. Various techniques are required to build the virtual universities efficiently. One of them is the scheme for transaction management. However, existing transaction management schemes have inappropriate features for supporting virtual universities. In this paper, after we present the requirements for transaction management in virtual universities, we propose a new locking-based transaction processing protocol that satisfies the requirements. In the protocol, a new method, called mark, is adopted to satisfy the requirements. The proposed protocol also eliminates unnecessary delays or abortions and thus, the proposed protocol can improve the performance of database systems.