실시간 보안 데이타베이스 시스템을 위한 낙관적 동시성 제어 기법

(Optimistic Concurrency Control for Secure Real-Time Database Systems)

김 대 호 † 정 병 수 †† 이 승 룡 ††

(Daeho Kim) (Byeong Soo Jeong) (Sungyoung Lee)

요 약 서로 다른 보안등급을 가진 사용자들에 의해 공유되는 정보를 관리해야 하는 데이타베이스 시스템에서는 보안은 중요한 요구사항중 하나이다. 실시간 보안 데이타베이스 시스템은 데이타의 논리적인 정확성뿐만 아니라, 트랜잭션을 마감시간 내에 처리해야 하는 시간적인 제약조건과 데이타와 트랜잭션의 등급에 따라 데이타의 사용을 통제하고 상위 등급의 정보가 하위 등급으로 유출되는 것을 방지하여야 하는 보안 제약조건을 추가적으로 갖는다. 본 논문에서는 이러한 실시간 보안 데이타베이스 시스템의 요구사항을 만족시키는 동시성 제어 기법을 제안한다. 제안하는 프로토콜은 낙관적 기법에 기반을 두면서 상충되는 실시간 제약 조건과 보안 제약 조건을 모두 만족시키기 위해 다중 버전과 Mark기법을 사용한다. 제안된 프로토콜은 직렬성을 만족하며 로킹에 기반을 둔 프로토콜과의 모의실험 비교 결과 데이타 충돌이 많은 환경에서 우수한 성능을 보이고 있다.

Abstract In many real-time applications that the system maintains sensitive information to be shared by multiple users with different security levels, security is another important requirement. A secure real-time database system must satisfy not only logical data consistency but also timing constraints and security requirements associated with transactions. Even though an optimistic concurrency control method outperforms locking based method in firm real-time database systems, where late transactions are immediately discarded, most existing secure real-time concurrency control methods are based on locking. In this paper, we propose a new optimistic concurrency control protocol for secure real-time database systems, and compare the performance characteristics of our protocol with locking based method while varying workloads. The result shows that our proposed O.C.C protocol has good performance in case of many data conflict.

1. 서 론

데이타베이스 시스템은 이용 환경에 따라 데이타의 논리적인 정확성(Logical Data Consistency) 뿐만 아니라 부가적인 제약조건들을 요구하게 된다. 제어 시스템이나 주식 정보 시스템과 같은 실시간 이용 환경에서의 데이타베이스는 트랜잭션들을 정해진 마감시간(Deadline)내에 처리하여야 하는 시간적인 제약조건을 가지며[1], 데이타의 보안을 필요로 하는 경우에 있어서는 데이타와 트랜잭션의 등급에 따라 데이타의 사용을 통제하고 상위 등급의 정보가 하위 등급으로 유출되는 것을 방지하여야 하는 제약조건을 요구한다[2]. 따라서, 종래의 데이타베이스 시스템에서 사용하는 동시성 제어 기법들은 위와 같은 이용 환경에서 그대로 사용하기에는 많은 문제점들을 가지고 있다.

실시간 데이타베이스에서는 트랜잭션들이 마감시간에 임박한 정도에 따라 우선 순위(Priority)를 부여하여 상위 우선 순위의 트랜잭션에 시스템 자원을 우선적으로 할당함으로써 마감시간이 임박한 트랜잭션들을 우선적으로 처리하고 있는 데, 기존의 동시성 제어 기법을 이용할 경우 데이타 충돌에 따라 하위 우선 순위 트랜잭

· 본 논문은 1998년 정보통신부 국제공동과제(과제번호 IJRP-9803-6)의 지원을 받았음

† 학생회원 : 경희대학교 전자계산공학과
tonkey@jupiter.kyunghee.ac.kr
†† 종신회원 : 경희대학교 전자정보학부 교수
jeong@nms.kyunghee.ac.kr
sylee@nms.kyunghee.ac.kr

논문접수 : 1999년 7월 9일
심사완료 : 2000년 1월 8일

션에 의해 상위 우선 순위 트랜잭션이 지연되거나 취소(Abort)되는 우선 순위 역전 현상(Priority Inversion)[1]이 발생하는 문제점을 가지고 있다. 또한 보안 데이타베이스에서는 데이타의 충돌로 인하여 하위 등급의 트랜잭션이 상위 등급의 트랜잭션에 의해 지연되거나 취소될 수 있는 점을 이용하여 의도적인 충돌에 의해 상위 등급의 정보를 하위 등급으로 유출시킬 수 있는 비밀채널(Covert Channel)[3]을 생성할 수 있는 문제점을 가지고 있다.

이러한 문제점들을 해결하기 위하여 종래의 로킹(Locking) 기법이나 타임스탬프를 이용한 낙관적인(Optimistic) 방법을 기반으로 하여 실시간 데이타베이스를 위한 여러 동시성 제어 기법들이 연구, 제안되어 왔다[4, 5]. 이들은 주로 데이타 충돌 발생 시 논리적인 데이타의 일관성을 유지함과 동시에 우선 순위 역전 현상이 발생하지 않도록 하여, 실시간 데이타베이스의 성능 척도가 될 수 있는 마감시간을 만족하는 트랜잭션들의 성공적 수행율을 높일 수 있도록 하고 있다. 보안을 위한 데이타베이스에 있어서도 다중 버전을 사용하여 상위 등급의 트랜잭션들과 하위 등급의 트랜잭션들간의 충돌 현상을 미연에 방지함으로써 비밀채널의 발생 가능성을 제거하는 프로토콜들이 제안되고 있다[6, 7, 8].

최근에는, 국방 통제 시스템과 같이 데이타베이스의 실시간적 요구사항과 더불어 데이타의 보안을 동시에 만족하여야 하는 응용 환경이 대두됨에 따라 실시간적 제약조건과 보안을 위한 제약조건들을 동시에 만족시킬 수 있는 동시성 제어 기법에 대한 연구가 진행되고 있다[9, 10]. 그러나, 지금까지 이러한 두 가지 제약조건을 동시에 만족하도록 제안된 동시성 기법들은 모두 로킹을 사용하는 방법을 기반으로 하고 있고, 낙관적 기법을 사용하는 프로토콜에 대한 연구는 아직 발표되지 않고 있다. 더욱이, 펌 실시간 데이타베이스 시스템(Firm Real Time Database System)에서와 같이 마감시간이 경과한 트랜잭션들이 즉시 수행을 중단하게 되는 경우에 있어서는 트랜잭션의 검증 단계(Validation Phase)에서 데이타의 충돌에 따른 동시성 제어를 하는 낙관적 기법이 로킹을 사용하는 기법에 비하여 효율적일 수가 있으므로[11] 낙관적 기법을 이용하는 동시성 제어 기법의 연구도 중요하다 할 것이다.

마감 시간이 경과한 트랜잭션들의 수행을 즉시 중단하는 펌 실시간 데이타베이스 시스템 환경에서 낙관적 기법이 로킹 기법에 비해 효과적일 수 있는 경우는 아래의 [그림 1]과 같이 마감 시간을 만족하지 못하게 될 트랜잭션 T_1에 의해 트랜잭션 T_2가 블럭되어 질 경우에 나타날 수 있다. 블록 되어진 T_2는 T_1이 마감 시간을 어겨 취소되는 시점에 필요한 로크를 얻어 수행을 재개하지만 T_2 역시 마감 시간을 만족하지 못하고 중단될 수가 있다(T_2의 작업소요 단위를 5, 마감 시간은 시점 6이라 가정한다). 그러나, 검증 단계에서 데이타 충돌을 검사하는 낙관적 기법인 경우에 있어서는 T_2의 검증 시점에 T_1은 이미 중단되어 T_2와의 데이타 충돌을 야기하지 않으므로 T_2는 시점 5에서 성공적으로 종료될 수가 있다.

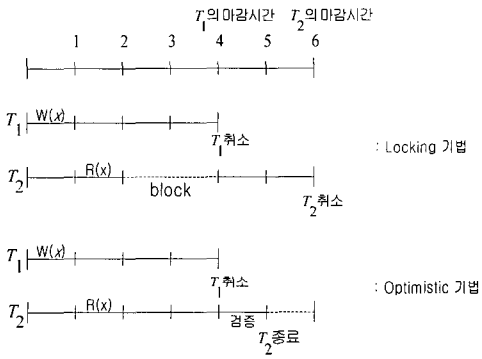


그림 1 Optimistic 기법의 waste abort의 해결 예

본 논문에서는, 낙관적 기법을 기반으로 하는 다단계 보안 실시간 데이타베이스의 동시성 제어 기법을 제안하고, 제안된 기법의 성능적 특징에 대하여 로킹을 기반으로 한 기법과의 비교 실험을 통하여 알아본다. 본 논문의 구성은 2절에서 실시간 데이타베이스 및 보안 데이타베이스를 위하여 지금까지 제안된 동시성 제어 기법들의 내용과 성능적 특징들을 간략히 요약하고, 3절에서는 제안된 프로토콜의 내용을 기술한다. 4절에서는 제안된 프로토콜의 논리적 정확성을 증명하고 5절에서는 성능 실험을 위한 시뮬레이션 모델과 실험 결과를 요약하며 6절에서 결론을 맺는다.

2. 관련 연구

실시간 데이타베이스 및 보안 데이타베이스의 동시성 제어 기법에 대한 연구는 초기에는 각각 별도로, 즉 실시간 데이타베이스의 제약조건만을 고려한 연구와 보안의 특성만을 고려한 연구로 독립적으로 진행되어 왔다. [1]에서는 실시간 데이타베이스 시스템에서의 우선 순위 역전 현상을 제거하기 위하여 데이타 충돌의 발생 시 상위 우선 순위의 트랜잭션들이 우선적으로 스케줄링 될 수 있는 여러 방법으로 Wait Promote, High Priority, Conditional Restart의 기법을 제안하고 이들

의 성능적 특징을 비교 실험하였다. [5]에서는 2PL-HP(Two Phase Locking-High Priority)기법을 펌 실시간 시스템에 적용할 경우 마감 시간을 어겨 중단될 상위 우선 순위 트랜잭션에 의하여 불필요하게 하위 트랜잭션들의 수행이 지연되거나 취소될 수 있는 문제점을 지적하고 이러한 불필요한 지연(Wasted Delay)이나 재시작(Restart)을 최소화할 수 있는 방법들(OPT-SACRIFICE, OPT-WAIT, WAIT-50)을 낙관적 기법에 기초하여 제안하고 있다.

보안 데이타베이스에 관하여는 Bell-LaPadula 모델[2]을 보안 모델로 하는 다단계 보안 데이타베이스(Multilevel Secure Database) 환경에서 트랜잭션의 병행 수행시 등급이 다른 트랜잭션들간의 데이타 충돌이 일어난 경우에 발생할 수 있는 비밀채널을 제거하기 위한 보안 프로토콜들이 제안되고 있다[7,12]. [7]에서는 다중 버전을 사용함으로써 등급이 다른 트랜잭션들간의 데이타 충돌을 없애고 직렬화 순서를 유지하면서 다중 버전을 유지 관리하는 방법을 소개하고 있고, [12]에서는 오렌지 로크(Orange Lock)라는 새로운 로크를 정의하여 단일 버전을 사용하면서 하위 등급 트랜잭션과 충돌할 경우 상위 등급 트랜잭션을 취소시킴으로써 하위 등급으로의 정보 흐름을 차단하고 있다. 유사한 방법으로 [10]에서는 가상 로크(Virtual Lock)를 사용하여 데이타 충돌이 일어날 경우 하위 등급 트랜잭션은 지연되거나 취소되지 않고 자신의 지역 공간에서 쓰기 작업을 계속하다가 해당 데이타 객체에 대하여 로크가 풀리면 가상 로크는 실제 로크(Real Lock)로 바뀌어 실제로 데이타베이스에 쓰기 작업이 수행되는 기법을 제안하고 있다. [12]방법의 문제점으로는 상위 등급의 트랜잭션이 지속적으로 취소되는 기아 현상(Starvation)의 문제점을 가지고 있고 [10] 방법은 데드락(Deadlock)의 가능성을 해결하여야 하는 문제점이 있다.

최근의 연구들은 실시간적 요건과 보안 요구사항을 동시에 만족할 수 있는 프로토콜들을 제안하고 있는 데, Son 등이 제안한 SRT-2PL[9]에서는 2단계 로킹 기법을 사용하는 기존의 실시간 프로토콜을 기반으로 비밀채널을 제거하기 위해 각 데이타 객체들에 대한 원본과 함께 별도의 복사본을 유지하여 같은 데이타 객체에 대한 상 하위 등급 트랜잭션의 읽기/쓰기 작업을 분리함으로써 충돌 발생을 없애고 있다. 아울러 직렬화 순서를 유지하기 위하여 복사본의 유지 관리를 갱신 큐(Update Queue) 구조를 통하여 효과적으로 수행하는 방법을 제시하고 있다. [8]에서도 다중 버전을 이용하고 로킹 기법에 기반한 실시간 보안 프로토콜을 제시하고 있는 데, 주목할 것은 직렬화의 기준으로 1-Copy Serializability 보다 완화된 FR(First Read)-Serializability를 도입하여 읽기 작업이 많은 트랜잭션 처리 환경에서 병행성을 높일 수 있는 이론적 근거를 제시하고 있다. 그러나 위의 방법들은 모두 로킹 기법을 기반으로 하고 있기 때문에 펌 실시간 데이타베이스 시스템에서 보안을 유지할 필요가 있는 환경에서는 성능적 취약점을 보일 수 있다. 본 논문에서는 다중 버전을 이용하여 비밀 채널을 제거하고 실시간적 제약조건을 충족하기 위하여 낙관적 기법을 사용하는 새로운 실시간 보안 프로토콜을 제안한다. 제안한 프로토콜은 펌 실시간 데이타베이스 시스템에서 불필요한 지연이나 재시작의 가능성을 줄임으로써 마감시간을 만족하는 트랜잭션의 성공률을 높일 수가 있다.

실시간 보안 프로토콜에 대한 그 밖의 연구로는 [13]에서 실시간적 요구사항과 보안의 요구사항이 상충하는 환경에서 시스템의 작업 부하에 따라 동적으로 실시간 혹은 보안 프로토콜을 선택적으로 사용하는 방법론을 제시하고 있고, [14]에서는 낙관적 기법을 사용하는 실시간 데이타베이스 시스템에서 직렬화 순서를 동적으로 재조정하여 불필요한 재시작을 줄일 수 있는 방법을 제안하고 있다. 그러나, [14]에서는 보안의 요구사항을 고려하지 않고 있고, [13]의 방법들은 데이타 보안의 요구가 절대적인 환경에서는 이용할 수 없는 문제점을 가지고 있다.

3. 낙관적 기법을 이용한 실시간 보안 프로토콜

3.1 다단계 실시간 데이타베이스 모델

본 절에서는 제안한 프로토콜의 이해를 위하여 본 논문에서 사용하는 보안 데이타베이스 모델과 낙관적 기법을 사용하는 실시간 프로토콜에 대하여 간략히 소개한다. 일반적으로 보안 데이타베이스는 Bell-LaPadular 보안 모델[2]에 근거한 다단계 보안 데이타베이스 모델을 사용하는 데 제안한 프로토콜도 이 모델을 사용한다. Bell-LaPadular 모델은 데이타 객체와 트랜잭션들을 다단계의 보안 등급(예: Top-Secret, Secret, Confidential,...)으로 구분하고 데이타의 사용은 아래와 같은 규칙에 따라 이루어지도록 함으로써 데이타 보안을 제공하는 강제적 접근 제어(Mandatory Access Control) 모델의 일종이다.

(1) 단순 보안 특성(Simple Security Property) : 트랜잭션의 읽기 연산은 트랜잭션의 보안 등급이 접근하는 데이타 객체와 동일 등급이거나 상위 등급일 경우에만 허용한다.

(2) 제한된 * 특성(Restricted *-Property) : 트랜잭션의 쓰기 연산은 트랜잭션의 보안 등급이 접근하는 데이타 객체와 동일 등급일 경우에만 허용한다. (*-특성인 경우는 상위 등급 데이타 객체에 대한 쓰기 연산도 함께 허용한다.)

낙관적 병행수행 기법은 트랜잭션을 읽기 단계(Read Phase), 검증 단계(Validation Phase), 쓰기 단계(Write Phase)로 나누어 수행하는 데, 읽기 단계에서는 트랜잭션의 지역공간을 이용하여 데이타에 대한 읽기/쓰기 연산이 이루어지며, 검증 단계에서 직렬화 순서의 유지를 위하여 데이타 충돌에 따른 조정 작업을 한 후, 검증이 성공적이면 실제 데이타베이스에 쓰기 작업을 하는 쓰기 단계를 거친다. 낙관적 실시간 프로토콜은 검증 단계에서 데이타 충돌시 트랜잭션의 우선 순위를 고려하여 하위 우선 순위 트랜잭션들을 재시작시킴으로써 마감시간을 만족하는 트랜잭션의 성공률을 높이고 있다. 본 논문에서 제안하는 프로토콜은 Bell-LaPudala의 보안 모델을 기반으로 낙관적 기법을 이용하여 실시간 요건을 만족할 수 있도록 설계되었다. 본 논문에서 사용하는 데이타베이스 시스템 모델에서는 수행되는 트랜잭션들은 보안 요건에 따른 등급과 마감시간을 고려한 우선 순위를 가지며 데이타 객체들도 보안 등급이 명시된다. 아래 절부터는 제안한 프로토콜에 대하여 기술한다.

3.2 실시간 보안 프로토콜

낙관적 병행수행 제어는 각 트랜잭션들이 읽기 단계에서는 서로 아무런 간섭 없이 병행 수행되다가 완료 시점에 이르면 검증 단계를 통하여 트랜잭션의 직렬화 순서를 검사하여 직렬화 순서에 위배되지 않으면 검증 단계의 트랜잭션은 완료(Commit)되고 위배될 경우는 충돌 해결(Conflict Resolution) 작업을 통하여 데이타의 일관성을 유지시키는 간단한 형태의 동시성 제어 기법이다. 검증 단계의 작업은 데이타 충돌을 검사하는 데 있어 완료된 트랜잭션들이 쓰기 작업을 한 데이타를 고려하는 역방향 검증(Backward Validation)이 있고 현재 활성중인(Active) 쓰기 단계의 트랜잭션들이 읽기 작업을 한 데이타와의 충돌을 고려하는 순방향 검증(Forward Validation) 방법이 있는 데 충돌 해결 과정에서 활성중인 트랜잭션들과의 우선 순위를 고려하여야 하는 실시간 환경에서는 순방향 검증을 사용하여야 한다[14]. 순방향 검증 작업은 아래와 같은 절차로 이루어진다.

validate(T_v)

```
{
    valid := true;
    foreach Ta(a = 1, 2, ... ,n) {
      if RS(Ta) ∩ WS(Tv) ≠ empty then valid := false;
    }
    if valid then commit WS(Tv) to database
    else conflict-resolution(Tv);
}
// Ta : 활성(읽기 단계)중인 트랜잭션,
   Tv : 검증 단계의 트랜잭션
// RS(Ta) : Ta가 읽은 데이타 집합,
   WS(Tv) : Tv가 기록한 데이타 집합
```

검증 과정에서 데이타 충돌이 있을 경우 T_a 혹은 T_v 중 어느 하나를 재시작 시킴으로써 트랜잭션의 직렬화 순서를 유지하게 되는 데 실시간 프로토콜에서는 우선 순위를 고려하여 하위 우선 순위의 트랜잭션을 재시작시켜 마감시간이 임박한 상위 우선 순위 트랜잭션을 우선적으로 완료시키게 된다.

표 1 실시간 보안 환경에서 가능한 공유데이타 충돌

Trans. / cases	T_v Write		T_a Read	
	priority	security level	priority	security level
case 1	*High*	*Equal*	*Low*	*Equal*
case 2	*High*	*Low*	*Low*	*High*
case 3	*Low*	*Equal*	*High*	*Equal*
case 4	*Low*	*Low*	*High*	*High*

실시간 보안 프로토콜에서는 검증 단계에서 트랜잭션의 우선 순위(본 논문에서는 각 트랜잭션의 우선 순위는 유일하다고 가정한다)뿐만 아니라 보안 등급도 같이 고려하게 되는 데 순방향 검증에서의 고려하여야 할 공유 데이타 충돌은 검증 단계 트랜잭션이 기록 연산을 수행한 데이타 집합과 활성중인 트랜잭션이 읽기 연산을 수행한 데이타 집합에 한하고 Bell-LaPadula 모델에서는 하향 기록 작업을 허용하지 않으므로 기록 연산을 한 트랜잭션이 상위 보안 등급일 경우는 발생하지 않게 된다. 따라서 공유 데이타의 충돌시 발생 가능한 트랜잭션들의 우선 순위와 보안 등급들은 [표 1]의 네 가지 경우에 한정한다.

위의 경우 중 첫 번째와 세 번째의 경우는 같은 보안 등급간의 트랜잭션 사이에 발생하는 데이타 충돌이므로 보안의 위배 사항(비밀채널)은 발생하지 않으며 기존의 실시간 프로토콜에 따라 낮은 우선 순위의 트랜잭션을

취소 혹은 재시작 시킴으로써 데이타의 일관성과 함께 시간적 제약조건도 만족시킬 수가 있다. 두 번째 경우도 우선 순위가 낮고 보안 등급이 높은 활성중인 트랜잭션을 취소 혹은 재시작 시키게 되면 보안 및 실시간 요건을 만족하게 된다. 문제가 되는 경우는 네 번째의 경우로서 검증 단계의 트랜잭션(T_v)을 취소시키면 비밀채널 발생의 문제점이 있고, 활성중인 트랜잭션(T_a)을 취소시키면 실시간 요건을 위배하여 우선 순위 역전 현상을 야기하는 문제점이 있게된다. 이 경우는 상위 우선 순위 트랜잭션이 하향 판독(Read Down)한 데이타에 대하여 하위 우선 순위 트랜잭션이 쓰기 작업을 하고 완료를 위한 검증 작업을 하는 경우에 해당된다. 제안하는 프로토콜은 실시간 요건과 보안의 요건이 서로 상충하게 되는 이러한 네 번째 경우에 대한 해결 방안을 제시한다.

제안하는 해결 방안은 데이타 객체에 대한 다중 버전을 이용하여 위와 같은 네 번째 경우의 데이타 충돌의 발생시 검증 단계의 트랜잭션도 완료시키고 활성중인 트랜잭션도 데이타 일관성을 위한 올바른 버전을 사용하게 하여 수행을 지속하게 하는 것이다. 트랜잭션의 검증 단계에서 위의 네 번째 경우가 발생하면 해당 활성중인 트랜잭션은 *Mark*되고 검증 시점의 타임스탬프(Marked TimeStamp) 값(본 논문에서는 트랜잭션의 검증 작업은 순서적(Sequential)으로 이루어지며 이때 고유한 타임스탬프 값을 부여받는 것을 가정한다)과 검증 단계 트랜잭션의 고유 번호(Identifier)를 가지고 MTL(Marked Transaction List)에 삽입된다. 이와 같이 *Mark*되어진 트랜잭션들은 *Mark*되어진 이후 읽기 연산을 수행할 때는 *Mark* 되어질 때 부여받은 타임스탬프 값과 데이타 버전들의 타임스탬프와 비교하여 올바른 버전을 선택하게 된다. Marked Transaction List 및 데이타 버전을 관리하기 위한 자료 구조와 규칙들은 다음절에서 자세히 기술한다.

3.3 데이타 버전의 선택 및 관리

제안한 프로토콜에서는 트랜잭션의 읽기 및 쓰기 연산은 아래의 규칙에 따라 이루어진다.

[규칙 1] 트랜잭션의 검증 작업은 순차적(Sequential)으로 이루어지며 이때 유일한 타임스탬프 값을 부여한다. 타임스탬프 값은 실제 시간과는 무관한 논리적인 순서를 의미하는 값으로 Counter에 의하여 트랜잭션의 검증 작업이 완료될 때마다 하나씩 증가된다.

[규칙 2] 모든 트랜잭션의 쓰기 동작은 트랜잭션의 지역 공간에서 이루어지며 검증 단계의 작업을 성공적으로 거치게되면 데이타베이스에 쓰기 작업을 한다. 이때 데이타에 대한 새로운 버전이 생성되며 데이타의 WTS(Write TimeStamp)는 검증 시점의 타임스탬프 값으로 기록된다.

[규칙 3] 트랜잭션의 검증 단계에서 [표 1]의 네 번째 경우가 발생하면 활성중인 트랜잭션들은 Mark되고 검증 시점의 타임스탬프 값을 MTS(Marked TimeStamp) 값으로 하여 트랜잭션 ID 정보와 함께 MTL(Marked Transaction List)에 삽입된다.

[규칙 4] Mark되어진 트랜잭션의 읽기 연산은 $WTS(x) < MTS(T)$ 인 데이타 버전중 가장 최근의 버전을 사용한다.($WTS(x)$는 데이타 x의 write timestamp값이고, $MTS(T)$는 트랜잭션 T가 Mark되어진 시점의 Timestamp값이다.)

[규칙 5] Mark되지 않은 트랜잭션의 읽기 연산은 항상 가장 최근의 데이타 버전을 사용한다.

[규칙 6] 모든 트랜잭션의 하향 판독(Read Down) 작업 시는 해당 데이타 버전의 하향 판독 트랜잭션 리스트에 트랜잭션 ID를 삽입한다.

다음의 규칙들은 데이타 버전 및 MTL의 관리에 관한 규칙들이다.

[규칙 7] Mark된 트랜잭션들이 완료되거나 재시작 되어질 경우는 MTL에서 해당 노드를 삭제한다.

[규칙 8] 모든 트랜잭션이 완료되거나 재시작 되어질 경우는 해당 트랜잭션이 하향 판독한 데이타 버전의 하향 판독 트랜잭션 리스트에서 해당 트랜잭션 ID를 삭제한다.

[규칙 9] MTL의 변경이 있을 경우(Mark 된 트랜잭션이 완료되거나 재시작 될 경우) MTL의 최소 MTS 값을 기준으로 데이타 버전들 중에서 최소 MTS 값보다 작은 WTS를 갖는 가장 최근의 한 데이타 버전을 제외한 이전 버전들은 삭제한다.

[규칙 10] 데이타 버전들 중에서 하향 판독한 트랜잭션이 없고 [규칙 9]를 위배하지 않을 경우 해당 데이타 버전을 삭제한다.

제안된 프로토콜에서, 트랜잭션의 수행은 읽기, 검증, 그리고 쓰기의 세 단계로 이루어진다. 읽기 단계의 과정은 다음에 기술하는 알고리즘과 같다.

```
read_phase(Ta)
{
    foreach Di (i = 1, 2, ..., m) in RS(Ta) {
      if (Ta is marked)
          select the version of Di where WTS(Di) <
          MTS(Ta)
          read that version
```

```
    else  read the most recent version of Di
    }
    foreach Dj (j = 1, 2, ..., n) in WS(Ta)
    write Dj in local workspace
}
```

읽기 단계가 끝난 후, 트랜잭션은 3.1절에서 기술한 것과 같이 순방향 검증기법에 따라 검증을 수행한다. 만일 검증을 수행하는 동안 데이타 충돌이 발생하면, 충돌을 해결해야 하는데 충돌 해결 알고리즘은 다음과 같다.

```
conflcit_resoution(Tv)
{   // P(T): Priority of Transaction, T     L(T): Security
      Level of Transaction, T
    case 1: P(Tv) > P(Ta) and L(Tv) = L(Ta)
            abort(Ta);     commit(Tv);
    case 2: P(Tv) > P(Ta) and L(Tv) < L(Ta)
            abort(Ta);     commit(Tv);
    case 3: P(Tv) < P(Ta) and L(Tv) = L(Ta)
            abort(Tv);
    case 4: P(Tv) < P(Ta) and L(Tv) < L(Ta)
            mark(Ta);     commit(Tv);
}
```

일단, 검증이 성공적으로 수행되면, 쓰기 단계에서 트랜잭션의 지역 공간에서 이루어진 데이타의 갱신을 데이타베이스에 반영한다. 이 때, 데이타의 새로운 버전은 검증 시점의 타임스탬프 값인 WTS를 타임스탬프 값으로 갖는다.

[표 2]의 수행 기록에 대하여 위의 규칙들에 따라 트랜잭션의 동작을 기술하면 아래와 같다. time 16(시스템에서 사용하는 타임스탬프 값은 검증 시점에서 유일하게 순서적으로 부여된다.)에서 T_2의 검증 작업이 수행되기 전까지는 트랜잭션들간에 아무런 간섭이 없이 읽기/쓰기 작업이 수행된다. T_1의 $r(x)$, $r(y)$ 및 T_2의 $r(y)$는 각각 최근의 버전인 $x=30$,

표 2 수행 기록의 예

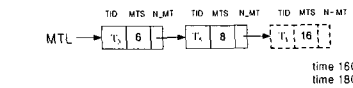
time / Trans.	···	10	11	12	13	14	15	16	17	18	···
T_1		r(x)			w(x)		r(y)		r(z)	v	···
T_2			r(y)	w(y		w(z)		v	···		

$P(T_1) > P(T_2)$, $L(T_1) = L(x) > L(T_2) = L(y) = L(z)$

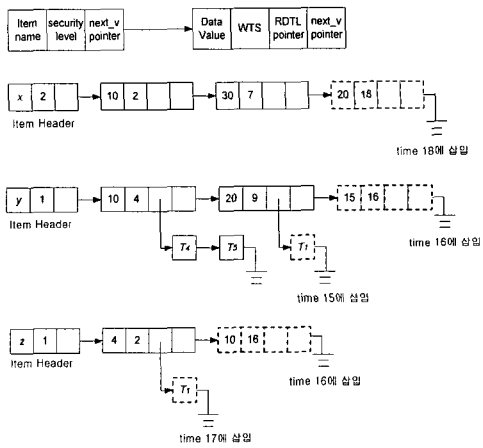
$P(T_i)$: T_i의 우선 순위 $L(T_i)$: T_i의 보안 등급

$L(x)$: 데이타 x의 보안 등급 v : validation phase

$y=20$을 각각 읽고 T_1과 T_2의 $w(x)$, $w(y)$, $w(z)$는 T_1, T_2의 지역 공간에서 쓰기 연산을 각각 수행한다. 이 때 T_1의 $r(y)$은 하향 판독이므로 해당 데이타 버전 y의 하향 판독 트랜잭션 리스트에 트랜잭션 ID를 삽입한다. time 16에서의 T_2의 검증 작업은 데이타 y로 인하여 [표 1]의 네 번째 경우에 해당하는 충돌이 발생하고 위의 규칙에 따라 [그림 2]의 (a)와 같이 T_1을 Mark하고 검증을 완료한 후 데이타베이스에 쓰기 작업을 한다. 데이타베이스에 쓰기 작업을 하게되면 [그림 2]의 (b)와 같이 y, z에 대한 새로운 버전이 생성된다. time 17에서의 T_1의 $r(z)$은 T_1이 Mark되어 있으므로 T_1의 MTS(=16) 값보다 작은 WTS 값을 갖는 버전 중 가장 최근인 $z=4$를 읽는다. 그 이후 T_1이 검증되어 수행을 완료하게 되면 x에 대한 새로운 버전을 생성하고 MTL에서 T_1의 노드를 삭제한다. MTL의 변경에 따라 MTL의 최소 MTS 값이 변경되면, 즉 [그림 2]의 (a)에서와 같이 T_3의 수행이 완료되거나 취소되어 MTL의 최소값이 6에서 8로 변경이 되면 버전 삭제 작업이 수행되어 [그림 2](b)의 데이타 버전들 중 데이타 x의 첫 번째 버전(value=10, WTS=2)은 삭제된다.



(a) MTL의 데이타구조



(b) 데이타구조 버전의 자료 구조

그림 2 제안한 프로토콜의 데이타 구조

4. 프로토콜의 정확성

본 절에서는 위에서 기술한 동시성 제어 기법이 실시간 및 보안의 요구사항과 더불어 데이타의 일관성을 위한 트랜잭션의 직렬화 수행을 만족하는 지에 대하여 직

렬화 그래프(Serialization Graph)와 서술적인 증명을 통하여 알아본다. 제안한 프로토콜은 아래와 같은 성질을 만족한다. [Lemma 1]은 제안한 프로토콜이 보안의 요구사항을 만족함을 설명하고 [Lemma 2]는 실시간 데이타베이스의 요구사항을 만족할 수 있음을 설명한다.

[Lemma 3]과 [Lemma 4]는 제안한 프로토콜에 따른 트랜잭션의 수행은 데이타 일관성을 지키기 위한 트랜잭션의 직렬화 순서를 유지할 수 있음을 설명한다.

[*Lemma 1*] 하위 보안 등급의 트랜잭션은 상위 등급 트랜잭션과의 데이타 충돌로 인하여 지연되거나(Delay Security) 취소되지 않는다(Recovery Security). 또한 상위 등급의 기록 작업은 하위 등급 데이타에 영향을 미치지 않는다(Value Security).

proof : 제안한 프로토콜은 낙관적 기법을 이용하므로 로킹 프로토콜의 경우와 같은 지연(Blocking)은 발생하지 않으며 트랜잭션의 취소(혹은 재시작)도 상위 등급 트랜잭션과의 데이타 충돌로 인하여는 발생하지 않는다. [규칙 3]과 [규칙 4]에서와 같이 상·하위 등급간의 데이타 충돌이 발생하였을 때 실시간 요건을 지키기 위하여 상위 등급 트랜잭션을 취소(혹은 재시작)하지 못할 경우는 하위 등급 트랜잭션은 Marking이 되고 계속 작업을 수행하게 된다. 또한 Bell-LaPadula 보안 모델에 따라 상위 등급 트랜잭션의 하향 기록은 발생하지 않는다. ■

[*Lemma 2*] 상위 우선 순위 트랜잭션은 하위 우선 순위 트랜잭션에 의하여 지연되거나 취소(혹은 재시작)되지 않는다.

proof : 제안한 프로토콜은 실시간 낙관적 동시성 제어 기법을 기반으로 하기 때문에 트랜잭션의 검증 단계에서 상위 우선 순위 트랜잭션이 취소(혹은 재시작)되는 경우는 발생하지 않게 된다. ■

[*Lemma 3*] 같은 보안 등급의 트랜잭션 수행은 모두 직렬적이다.

proof : 같은 보안 등급의 트랜잭션의 수행에 있어서는 [표 1]에서의 case 2, 4와 같은 경우가 발생하지 않기 때문에 제안한 프로토콜은 [5]에서와 같은 실시간 낙관적 동시성 제어 기법과 같이 동작하게 된다. 따라서 [5]에서와 같이 완료된 트랜잭션들은 모두 직렬적임을 알 수 있다. 같은 보안 등급의 트랜잭션들에 대한 직렬화 순서는 트랜잭션의 검증 순서와 일치하는 완전 순서(Total Ordering)의 형태로 주어질 수 있다. ■

[*Lemma 4*] 완료된 트랜잭션들은 모두 직렬적이다.

proof : [성질 3]에서 같은 보안 등급의 트랜잭션들의 수행은 모두 직렬적임을 증명하였다. 따라서 보안 등급이 다른 트랜잭션들의 수행도 모두 직렬적임을 증명하면 될 것이다. 증명은 만약 트랜잭션들의 수행이 직렬화에 위배되면 아래와 같은 경우의 순환적인 직렬화 그래프가 존재하게 될 것이므로 제안한 프로토콜에 따르면 이러한 경우가 모순됨을 보인다.

(case 1) $T_i \rightarrow t_j \rightarrow t_k \rightarrow T_i$ (T_i : 상위 보안 등급, t_j, t_k : 하위 보안 등급)

제안한 프로토콜에서는 보안 모델로 하향 판독만을 고려하므로(즉 상위 등급 T_i가 쓰기 연산을 한 데이타를 하위 등급 t_j가 읽기 연산을 하는 W -> R 경우는 발생하지 않는다.) T_i -> t_j 는 상위 등급 Ti가 읽기 연산(하향 판독)을 한 하위 등급 데이타 x에 대하여 t_j가 나중에 쓰기 연산을 한 경우이다. 이러한 경우는 T_i가 t_j보다 먼저 검증을 거쳐 완료를 한 경우(I)와 t_j의 완료 후에 T_i가 완료되는(II) 두 가지 경우를 생각할 수 있다. II의 경우에는 t_j의 검증 단계에서 T_i가 Mark되어진다. 그리고 t_k -> T_i는 t_k가 쓰기 연산을 한 데이타 y를 나중에 T_i가 읽기 연산을 한 경우이다. I의 경우에는 T_i의 타임스탬프가 t_k의 타임스탬프보다 작은 값을 갖게 되므로 t_k -> T_i 가 발생할 수 없고 II의 경우에도 T_i의 읽기 연산은 항상 T_i의 MTS 값보다 적은 WTS 값을 가진 데이타 버전을 읽게 되고 t_k가 쓰기 연산을 한 데이타 버전의 WTS는 항상 T_i의 MTS 값보다 크게 된다([성질 3]에서 t_k의 타임스탬프 값은 항상 t_j의 타임스탬프 값보다 크고 T_i의 MTS 값은 t_j의 타임스탬프 값과 같기 때문이다). 따라서 II의 경우에도 t_k -> T_i는 발생할 수 없고 $T_i \rightarrow t_j \rightarrow t_k \rightarrow T_i$ 의 순환적 직렬화 그래프는 제안한 프로토콜에서는 발생하지 않는다.

(case 2) $t_i \rightarrow T_j \rightarrow T_k \rightarrow t_i$ (T_j, T_k : 상위 보안 등급, t_i : 하위 보안 등급)

이 경우도 (경우 1)에서와 같은 방법으로 모순을 증명할 수 있는 데, 제안한 프로토콜에서 t_i -> T_j가 의미하는 것은 (경우 1)에서의 설명처럼 하위 등급 t_i가 기록 연산을 한 하위 등급 데이타 x를 T_j가 읽기 연산(하향 판독)을 수행한 경우이며 이것은 t_i가 T_j보다 먼저 수행을 완료하였을 경우에 발생한다. T_k -> t_i가 의미하는 것은 상위 등급 T_k가 하향 판독한 하위 등급 데이타 y를 나중에 t_i가 쓰기 연산을 한 경우이다[그림 3]. T_j -> T_k의 의미는 T_j가 쓰기 연산을 한 데이타 z

를 T_k가 나중에 읽기 연산을 한 경우인데 [그림 3]에서처럼 트랜잭션 T_k는 t_i의 검증 과정에서 Mark 되어지고 그 이후의 읽기 동작은 항상 MTS 값 이전의 데이타 버전에 대해서 이루어진다. 따라서 T_j -> T_k의 경우는 발생하지 않게 되고 (경우 2)의 순환적 직렬화 그래프는 제안한 프로토콜에서는 발생하지 않는다. ■

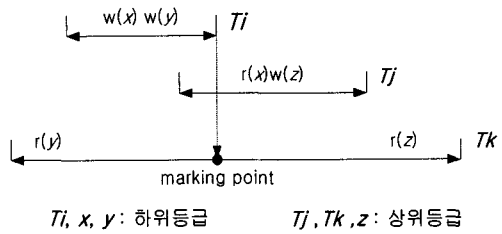


그림 3 (경우 2)의 트랜잭션 수행 과정

5. 성능 평가

이 장에서는 제안된 프로토콜의 성능 평가에 사용된 시뮬레이션 모델과 실험환경에 대한 내용을 설명한다. 또한 로킹 기반 프로토콜과의 성능적 특징을 비교 분석하기 위해 모의 실험한 결과를 기술한다.

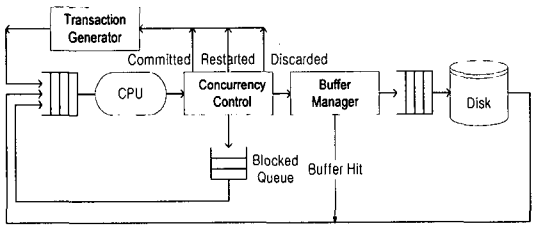


그림 4 시뮬레이션 큐잉 모델

5.1 시뮬레이션 모델

[그림 4]는 실험을 위해 사용된 시뮬레이션 큐잉 모델이다. 각 트랜잭션은 트랜잭션 생성기에서 포아송 분포에 따르는 도착 간격을 가지고 생성되며, 생성 후 CPU 큐에서 수행 시작을 기다린다. 시스템의 다양한 작업부하(workloads)에 따른 성능 실험을 위해 트랜잭션의 평균 도착율을 변화하면서 실험하였다. CPU 큐의 트랜잭션은 마감시간으로부터 계산되는 우선 순위에 의해 작업순서가 정해진다. 작업순서가 정해지면, 트랜잭션은 주어진 프로토콜에 의해 데이타에 대한 사용가능 여부를 확인해야 한다. 일단, 사용이 허락되면, 트랜잭션은 디스크 접근과 CPU 사용 등의 데이타 연산을 수행한다. 사용이 허락되지 않으면, 트랜잭션은 blocked 큐에 들어가 프로토콜에 의해 재시작 되거나 마감시간을 만족할 수 없으면 큐로부터 제거된다. 이미 마감시간을 지난 트랜잭션이 있으면 즉시 CPU 큐로부터 제거된다.

슬랙 값은 실시간 시스템의 부하 특성을 나타내는 값으로 모의 실험에서는 슬랙 값의 변화를 통하여 다양한 실시간 환경에서의 프로토콜 성능을 측정하였다.

시뮬레이션에 사용된 인수들은 [표 3]과 같다. 데이타베이스는 디스크에 있는 데이타 페이지의 집합을 나타내고, 슬랙 값은 다음 식에서와 같이 트랜잭션의 마감시간을 계산하는데 사용된다.

$$deadline = arrival\_time + slack * transaction\_size * execution\_time$$

트랜잭션의 마감시간은 각 트랜잭션의 우선 순위 할당에 사용되며, 각 트랜잭션의 우선 순위 할당은 EDF(Earliest Deadline First)알고리즘을 사용하였다. 트랜잭션의 보안등급은 생성시 임의로 주어지고 데이타 객체의 보안등급은 각기 다른 등급으로 초기에 설정된다. 트랜잭션 크기는 하나의 트랜잭션이 가진 평균 연산(읽기 혹은 쓰기)의 수를 나타내고, 크기를 변화시킴으로서 다양한 시스템 부하를 가진 환경을 실험하였다. 트랜잭션이 데이타 페이지를 읽고자 할 때, 시스템은 페이지가 메모리에 있는지 여부를 확률(buffer hit)을 사용하여 결정한다. 마감시간 결정을 위해 사용되는 수행시간은 다음과 같이 계산된다.

표 3 시뮬레이션 인수

Parameters	Value
Database Size	*100*
Slack Value	*1.5~4.5*
Security Level	*3*
Transaction Size	*5~20*
CPU Time	*3.0 ms*
Disk Time	*25 ms*
Buffer Hit	*0.7*
Transaction Arrival Rate	*5~30 tr./sec*

$$execution\_time = CPU\_time + (1 - Buffer\_Hit) * Disk\_time$$

5.2 성능 분석

제안한 프로토콜의 성능적 특징을 로킹을 기반으로 하는 기법과 비교 분석하기 위하여 제안한 프로토콜과 더불어 로킹을 기반으로 하는 PSMVL[15]을 함께 구현하여 비교하였다. PSMVL은 제안한 프로토콜과 마찬가지로 실시간 요건과 보안의 요구사항을 모두 만족할 수 있도록 설계된 것으로, 제안한 프로토콜과의 차이점은 로킹 기법을 기반으로 하고 있다는 것이다. 실시간 요건과 보안의 요건이 상충되는 상황을 해결하는 방법에 있어서는 데이타의 다중 버전을 활용하는 측면이 제안한 프로토콜과 유사하다. [14]에서 언급한 것과 같이 로킹 기법과 낙관적 기법간의 프로토콜 성능 비교는 두 기법의 구현에 따르는 프로토콜 오버헤드의 차이로 인하여 정확한 성능 비교에는 많은 어려움이 따른다. 본 모의 실험에서는 두 기법간의 프로토콜 오버헤드는 같다는 가정하에 두 프로토콜에서 데이타 충돌이 성능에 미치는 영향을 주로 고려하였다. 본 실험에 사용한 시뮬레이션은 Sun UltraSparc에서 CSIM 라이브러리 함수를 이용하였다.

성능 비교에 사용되는 주요 척도로는 트랜잭션의 마감 시간을 만족하지 못하는 비율을 나타내는 트랜잭션 Miss Ratio와 트랜잭션 취소에 따른 트랜잭션의 평균 재시작 비율인 Restart Ratio를 사용하였다. 트랜잭션의 마감 시간 만족도를 측정하기 위한 트랜잭션 Miss Ratio는 다음 공식으로 계산한다.

Miss Ratio = 100 * (*# of discarded transactions / # of transactions arrived*)

모의 실험에서는 이러한 트랜잭션 Miss Ratio와 Restart Ratio를 시스템 부하, 트랜잭션의 크기, 그리고 슬랙 값을 변화해 가면서 측정하였다. 추가적으로, 트랜잭션의 평균 대기(blocking)시간과 조만간 제거될 트랜잭션에 의한 불필요한 재시작(restart)과 대기(blocking) 시간을 측정하였다.

[그림 5]와 [그림6](그래프에서 SRT_OPT는 제안한 프로토콜을 나타낸다.)은 평균 트랜잭션 도착율에 따른 트랜잭션의 마감시간 만족도를 보여준다. 모의 실험에서 트랜잭션 수행에 따른 데이타 충돌의 빈도를 조절하기 위하여 트랜잭션 쓰기 확률을 각각 0.1과 0.5로 설정하여 수행하였는데, [그림 5]에서와 같이 쓰기 확률 값이 작을 때는 데이타 충돌이 거의 발생하지 않기 때문에 두 프로토콜이 거의 같은 성능을 보인다. 하지만, [그림 6]과 같이 데이타 충돌이 많아지고 시스템의 부하가 커질수록 제안한 프로토콜과의 마감시간 만족도 차이가

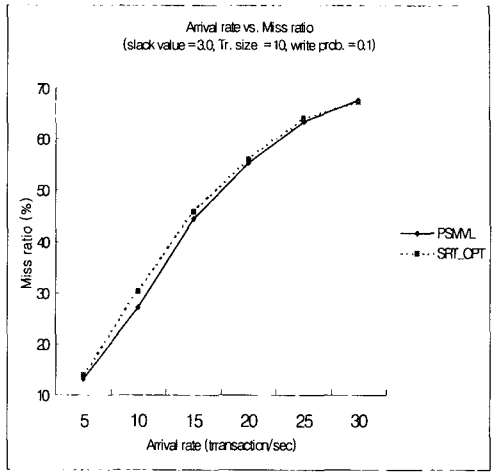


그림 5 평균 도착율에 따른 Miss Ratio(쓰기 확률 = 0.1)

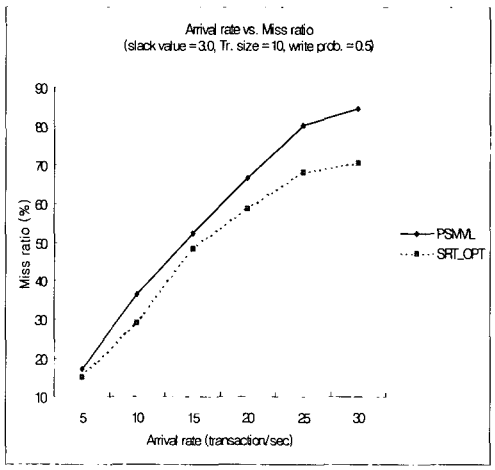


그림 6 평균 도착율에 따른 Miss Ratio(쓰기 확률 = 0.5)

커진다. 이는, 로킹 기법이 곧 제거될 트랜잭션에 의한 불필요한 재시작이 많기 때문이다. [그림 7]도 비슷한 이유로 설명할 수 있다. 트랜잭션 크기가 커질수록 데이타 충돌이 더 빈번하게 일어나기 때문에 마감시간 만족도의 차이가 크기가 커질수록 커진다. [그림 8]은 충분한 슬랙 값을 가지면, 두 프로토콜 모두 충돌하는 트랜잭션을 재시작 시킴으로서 마감시간을 만족하는 트랜잭

션이 증가함을 보여준다.

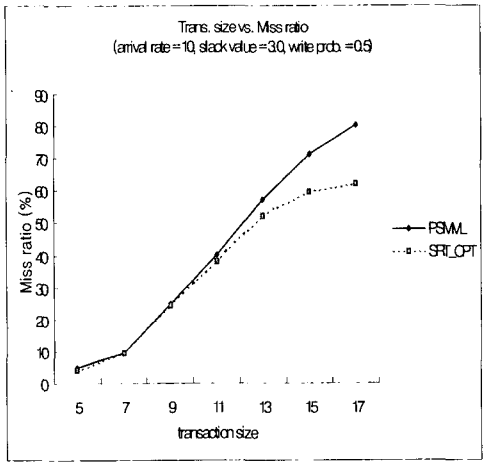


그림 7 트랜잭션 크기에 따른 Miss Ratio

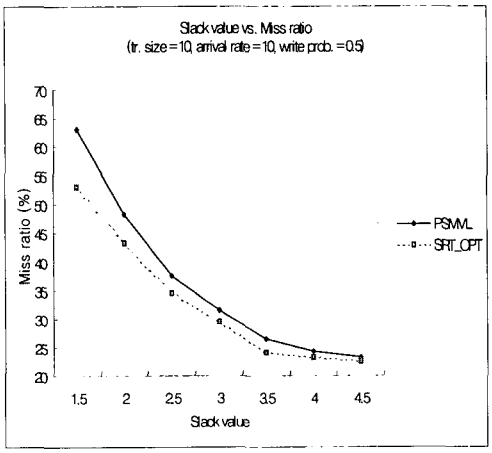


그림 8 슬랙 값에 따른 Miss Ratio

[그림 9]와 [그림 10]은 트랜잭션의 평균 도착율과 슬랙 값에 다른 재시작 비율을 보여준다. 재시작 비율은 시뮬레이션 동안의 트랜잭션의 평균 재시작 비율을 의미한다. 사실, 비교하는 두 프로토콜은 데이타 충돌이 발생할 때 충돌에 대한 해결방법이 다르기 때문에 재시작 비율이 의미가 큰 것은 아니다. 낙관적 접근법에서는 데이타 충돌을 트랜잭션의 재시작으로 해결하고, 로킹 기반의 방법에서는 트랜잭션을 재시작 시키거나 대기(blocking)시킴으로서 해결한다. [그림 9]에서와 같이 시스템의 부하가 많은 상황에서는 마감시간을 만족시키지 못하는 트랜잭션이 많아지기 때문에 재시작 비율이 작아진다. 비슷한 이유로 [그림 10]에서와 같이 충분한 슬랙 값을 가지면 재시작에 의해 마감시간을 만족시키는 트랜잭션이 많아지기 때문에 재시작 비율이 슬랙 값이 커질수록 높아진다.

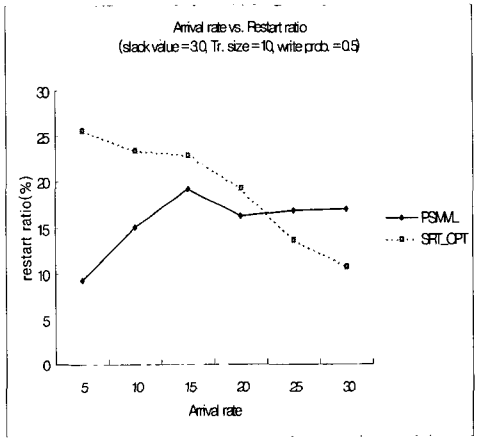


그림 9 평균 도착율에 다른 재시작 비율

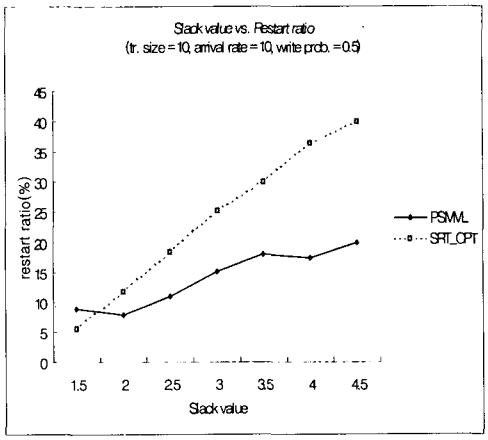


그림 10 슬랙 값에 따른 재시작 비율

6. 결 론

본 논문에서는 낙관적 기법에 근거한 새로운 보안 실시간 동시성 제어 기법을 제안하였다. 제안한 프로토콜은 다중 버전을 관리하고 Mark기법을 사용함으로써, 실시간 제약 조건과 보안 제약 조건이 상충하는 경우에

트랜잭션을 취소시키는 대신 적절한 데이타 버전을 사용하여 트랜잭션을 계속 진행시킴으로써 트랜잭션들의 직렬성과 더불어 실시간 제약사항과 보안요구사항을 동시에 만족시킨다. 성능평가를 위해 로킹을 기반으로한 방법인 PSMVL과 제안한 기법에 대하여 트랜잭션 크기, 슬랙 값, 트랜잭션의 평균 도착율 등을 변화하면서 다양한 환경에서의 데이타 충돌의 영향을 비교, 실험하였다. 실험결과는 순방향 검증 기법을 사용하는 낙관적 기법이 로킹 기법에 비해 불필요한 재시작을 감소시키기 때문에 데이타 충돌이 많은 환경에서 더 좋은 성능을 나타냄을 보여주었다.

참 고 문 헌

[1] Abbott. R. K. and Garcia-Molin H, "Scheduling Real-Time Transactions : A Performance Evaluation," ACM Transactions on Database Systems, 17, pp513-560, 1992.

[2] Bell. D. E and LaPadula. L. J., "Secure Computer Systems:Unified Exposition and Mulitcs Interpretation," The Mitre Corp., 1976.

[3] Ira S. Moskowitz and Myong H. Kang, "covert channels - Here to Stay?," Proc. COMPASS 94, pp235-243, 1994.

[4] Sha L., R. Rajkumar, S.H. Son, and C. Chang, "A Real-Time Locking Protocol," IEEE Trans. on Computer, pp782-800, 1991.

[5] Haritsa, J. R., M. J. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control," 11th IEEE Real-Time Systems Symposium, 1990.

[6] S. H. Son and R. David, "Design and Analysis of a Secure Two-Phase Locking Protocol," COMPSAC'94, pp374-379, 1994.

[7] Thomas F. Keefe, W. T. Tsai, Jaideep Srivastava, "Database Concurrency Control in Multilevel Secure Database Management Systems," IEEE Trans. on knowledge and Data Engineering vol5, no. 6, pp1039-1055, 1993.

[8] 박찬정, 박석, "다단계 보안 데이타베이스 시스템에서 얼림 기법에 기초를 둔 다중 버전 병행수행 제어 프로토콜", 정보과학회논문지(B), 제 25권, 제8호, 1998.

[9] R. Mukkamala and S. H. Son, "A Secure Concurrency Control Protocol for Real-Time Database," IFIP WG 11.3 Conference of Database Security, pp235-253, 1995.

[10] R. David, S. H. Son and R. Mukkamala, "Supporting Security Requirements in Multilevel Real-Time Databases," IEEE Symposium on Security and Privacy, pp199-210, 1995.

[11] George, B. and J. Haritsa, "Secure Transaction Processing in Firm Real-Time Database System," Proceedings SIGMOD, pp462-473, 1997.

[12] J. Mcdermott and S. Jajodia, "Orange Locking Channel-Free Database Concurrency Control via Locking," IFIP WG 11.3 Workshop in Database Security, pp267-284, 1992.

[13] S. H. Son, R. David, and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems," Annual IFIP WG 11.3 Conference of Database Security, pp223-233, 1995.

[14] J. Lee and S. H. Son, "Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems," 14th IEEE Real-Time Systems Symposium, pp66-75, 1993.

[15] Chanjung Park, Seog Park, and Sang H. Son, "Priority-driven Secure Multiversion Locking Protocol for Real-Time Secure Database Systems," Proceedings of IFIP 11th Working Conference on Database Security, 1997.8

김 대 호

1997년 경희대학교 전자계산공학과 학사. 1999년 경희대학교 전자계산공학과 석사. 1999년 ~ 현재 경희대학교 전자계산공학과 박사과정. 관심분야는 실시간 데이타베이스, 멀티미디어 데이타베이스

정 병 수

1983년 서울대학교 전자계산기공학과 학사. 1985년 한국과학기술원 석사. 1995년 Georgia Institute of Technology, College of Computing 박사. 1985년 ~ 1989년 8월 한국데이타통신(주) 정보통신연구소 선임연구원. 1996년 ~ 현재 경희대학교 전자정보학부(전자계산공학 전공) 조교수. 관심분야는 병렬 데이타베이스, 실시간 데이타베이스, 멀티미디어 데이타베이스 등

이 승 룡

1978년 고려대학교 재료공학과 학사. 1987년 12월 Illinois Institute of Technology 전산학과 석사. 1991년 12월 Illinois Institute of Technology 전산학 박사. 1992년 ~ 1993년 Governors State University 조교수. 1993년 ~ 현재 경희대학교 전자정보학부(전자계산공학전공) 부교수. 관심분야는 실시간 시스템, 실시간 고장허용시스템, 멀티미디어 시스템