다단계 보안 환경에서 동적 다중 버전 제어

정 현 철[†] · 황 부 현[††]

요 약

데이타들의 일관성과 보안성의 유지는 데이타베이스 보안에서 해결되어야 할 중요한 문제이다. 이 목적을 달성하기 위하여 트랜잭션의 직렬성이 보장되어야 하며 상하위레벨(혹은 상하위 보안등급) 트랜잭션들 사이에 비밀경로가 발생하지 않아야 한다. 본 논문에서는 빈번한 갱신이 발생할 때 다중 버전 유지에 따른 디스크 공간 부담과 이중 버전 유지에 의한 오래된 버전을 판독하게 되는 문제를 해결할 수 있는 동적 버전 제어 방법을 제안한다. 디스크 공간 문제는 적절한 버전 생성과 동적인 버전 갯수를 유지시키므로써 해결될 수 있고 오래된 버전 판독 문제는 가능한 한 최근의 버전을 판독하게 하므로써 해결될 수 있다.

Dynamic Multiversion Control in Multilevel Security Environments

Hyun Cheol Jeong[†] · Bu Hyun Hwang[††]

ABSTRACT

Security as well as consistency of data is very important issue in database security. Thus, the serializability of transactions must be maintained and particularly covert channel not caused between a high-level transaction and a low-level one. In this paper, we propose a secure transaction management algorithm using dynamic version control method that can slove disk space overhead to maintain multiversion and the problem that transactions read too old versions when two versions are maintained. Disk space overhead can be sloved by properly creating versions and dynamically maintaining the number of versions and the problem for reading too old version can be solved by having transactions read versions as recent as possible.

1. 서 론

데이타베이스 관리 시스템은 데이타베이스에 저장되어 있는 데이타에 대한 불법적인 접근, 고의적인 파괴, 변경, 그리고 비일관성을 발생시키는 접근으로부터 데이타를 보호하기 위하여 보안정책을 수행하여야 한다. 다단계 보안 환경에서 동시성 제어는 트랜잭션들의 직렬성을 보장해야 하며 트랜잭션들 사이의 공모로 인하여 데이타의 충돌시 형성될 수 있는 비밀경로(covert channel)[1]를 방지해야 한다. 또한, 상위레벨 트랜잭션 T^H이 하위레벨 트랜잭션 T^L의 수행에 영향을 주지 않아야 하는 불간섭 성질[2][3] 또한 보안성을 위반하지 않아야 한다. 일반적으로 다단계 보안 데이타베이스 시스템에서는 BLP (Bell-LaPadula) 모델[4][5][6]에 따른 접근 제어 메커니즘을 사용한다. 이 모델에서 객체는 데이타 파일, 하나의 레코드 혹은 한 레코드내의 일부분이 될 수 있고 주체는 객체에 접근하여 연산을 수행하려는 프로세스가 된다. 객

※이 논문은 '96년도 한국학술진흥재단의 공모과제 연구비 지원에 의해 연구되었음.
† 정 회 원:전남대학교 대학원 전산통계학과 박사과정 수료
†† 정 회 원:전남대학교 전산학과 교수
논문접수:1997년 1월 6일, 심사완료:1997년 3월 15일

체와 주체는 보안등급을 할당받으며 이들의 보안등급(혹은 레벨)은 부분적인 순서를 갖는다. 객체를 접근할 때 BLP 모델에서는 상위레벨의 객체에서 하위레벨의 주체로 정보가 흐르는 것을 방지하기 위해 단순 성질과 *-성질을 갖는다. 특히, 데이타의 무결성을 보장하기 위해 주체(혹은 트랜잭션)의 레벨이 객체(혹은 데이타)의 레벨과 같을 때만 객체에게 기록연산이 허용되도록 *-성질을 제한할 수 있다. BLP 모델은 상위레벨에서 하위레벨로의 직접적인 정보 흐름을 방지할 수 있으나 상위레벨의 주체가 하위레벨의 주체에게 간접적으로 정보 전달을 허용할 수 있는 비밀경로를 방지할 수는 없다. 비밀경로는 저장 비밀경로(covert storage channel)[7]와 시간 비밀경로(covert timing channel)[7]로 분류된다. 동시성 제어 메커니즘에 관련되는 시간 비밀경로는 연속적으로 제출되는 트랜잭션들 사이에서 상대적인 시간 변화로 인하여 발생하는데 T^H와 T^L이 서로 공모하여 T^L이 접근할 수 없는 중요한 정보를 T^H가 T^L에게 노출시키므로써 이 정보가 불법적으로 이용될 수 있다. 기존의 다단계 보안 동시성 제어 기법은 T^H와 T^L 사이에서 발생할 수 있는 비밀경로를 피하기 위하여 제안되어 왔다. 기존에 연구된 방법은 단일버전 방법과 다중 버전 방법으로 분류될 수 있다. 단일버전 방법에서는 비밀경로를 피하기 위하여 T^H와 T^L 사이에 충돌이 발생할 때 T^H를 철회하거나 지연시킨다[8][9]. 그래서 T^H의 기근 문제(starvation problem)[10]가 발생할 수 있다. 또한, 두 트랜잭션들 사이의 불간섭 성질을 만족시키기 위하여 T^H를 철회시키는 것은 공평하지 못하다. 만약 T^H가 많은 데이타를 접근하고 오랜 시간 동안 수행되며 판독연산만을 갖는 장기 트랜잭션(long transaction)[11]이라면 이 판독전용 트랜잭션(ROT; Read-Only Transaction)은 완료되지 못할 가능성이 높다. 다중 버전 방법에서는 T^H와 T^L 사이의 불간섭 성질을 만족시키기 위하여 T^H가 옛 버전을 읽게 하여 T^H와 T^L 사이의 충돌관계를 피할 수 있지만 한 객체에 대하여 다수의 버전을 유지해야 하기 때문에 디스크 공간에 대한 부담을 갖게 된다[12][13][14]. 다중 버전 방법에서 트랜잭션이 가능한 한 최근의 버전을 읽을 수 있도록 하는 것은 중요한 문제이다. 대부분의 다중 버전 방법은 트랜잭션 T가 시스템에 처음 도착했을 때, 지금 수행중인 트랜잭션들의 타임스템프에 근거하여 T의 타임스템프가 결정되고, T의 타임스템프는 T의 수행중에는 변화되지 않는다. T가 타임스템프를 할당받을 때, 현재 수행중인 T^L의 타임스템프 보다 더 작게 부여된다면 옛 버전을 읽는다. T의 타임스템프가 현재 수행중인 T^L의 타임스템프 보다 더 크면 현재 수행중인 모든 트랜잭션이 완료되도록 기다려야만 한다. T의 타임스템프가 현재 수행중인 트랜잭션들의 타임스템프중 가장 작은 것과 가장 큰 것 사이에 있다면 이에 따라 T가 판독할 버전이 결정된다.

2. 문제 정의

두 트랜잭션들의 충돌관계로 형성되는 비밀경로를 통하여 T^H는 T^L에게 간접적으로 데이타를 보낼 수 있다. 이를 해결하기 위하여 단일버전 방법에서는 T^H와 T^L의 충돌관계를 방지하기 위하여 T^H를 철회시킨다. 다중 버전 방법에서는 각 데이타들에 대해 다수의 버전들이 유지되므로 T^H와 T^L 사이에 충돌관계가 형성되지 않도록 T^H에 적당한 버전을 제공할 수 있으므로 비밀경로를 방지할 수 있다. T^H의 판독연산($R^H(x)$)이 T^L의 기록연산($W^L(x)$)과 충돌관계일때, 직렬성의 위반없이 $R^H(x)$은 $W^L(x)$가 생성한 것 보다 옛 버전을 읽도록 허용하고, $R^H(x)$가 먼저 수행되었다면 $W^L(x)$가 새로운 버전을 생성하게 하므로써 충돌관계를 해결할 수 있다. 대부분 다단계 보안 다중 버전 동시성 제어 방법은 트랜잭션이 시스템에 도착했을 때 현재 수행중인 트랜잭션에 관련된 트랜잭션이 판독 할 수 있는 버전의 최근성이 결정된다. 수행중인 트랜잭션의 타임스템프에 따라서 트랜잭션이 시스템에 제출될 때 현재 수행중인 트랜잭션들의 타임스템프 중 가장 작은 타임스템프 보다 더 작은 타임스템프를 할당하여 수행중인 트랜잭션이 생성한 버전 보다 옛 버전을 판독하게 하는 방법[14]이 있다. 상위레벨로 제출되어 수행되는 트랜잭션의 판독연산이 트랜잭션의 레벨 보다 낮아 엄격히 지배되는 레벨에 있는 데이타의 버전을 판독하기 위해 접근하는 연산을 하향판독연산(RD-op; Read Down-operation)이라고 하는데, 이 방법에서는 RD-op를 연기하지 않지만 다른 방법에 비해 더 옛 버전을 읽는다. 또 수행중인 트랜잭션들의 어떤 타임스템프 보다도 더 큰 타임

스템프를 할당하여 수행중인 가장 큰 타임스템프를 갖는 트랜잭션이 생성한 버전을 판독하게 하는 방법[13]에서는 제출된 트랜잭션 자신보다 더 작은 타임스템프를 갖으면서 충돌관계인 모든 트랜잭션들이 완료될 때까지 그 트랜잭션의 RD-op를 연기시킨다. 그러나, T^H는 충돌관계이면서 자신 보다 더 작은 타임스템프를 갖는 T^L의 존재를 알 수 없으므로 T^L이 새로운 버전을 생성한다면 T^H는 그 버전을 다시 판독해야 한다. T^H의 재수행 트랜잭션은 원래의 T^H와 충돌관계가 형성되고 T^H 보다 큰 타임스템프를 갖는 다른 모든 트랜잭션들의 재수행을 일으킬 수 있다. 그리고 수행중인 트랜잭션들의 가장 작은 타임스템프와 가장 큰 타임스템프 사이의 타임스템프를 할당하여 판독할 버전의 최근성을 결정하는 방법[12]은 제출된 모든 트랜잭션들이 가장 큰 타임스템프 보다 더 작은 타임스템프를 원할 경우 가장 큰 타임스템프를 갖는 트랜잭션은 계속 철회될 수 있으며 재수행 문제가 생긴다. 다중 버전 방법에서의 어려운 점은 버전 수를 고정시킬 수 없다는 것이다. 가능한 한 버전 수가 적고 최근의 버전이 제공될 수 있는 방법이 필요하다. 장기 트랜잭션을 지원하는 잠금 프로토콜[17]은 이중 버전을 사용하였으나 장기 트랜잭션과 함께 수행되는 트랜잭션들이 판독하는 최근의 버전은 장기 트랜잭션의 수행시간에 종속된다. 현재 버전을 옛 버전으로 추가하는 버전간격(VI; Version Interval)의 결정에 있어서 VI가 짧을수록 장기트랜잭션의 철회가 증가하고 길수록 장기트랜잭션과 함께 수행되는 단기 트랜잭션들이 더 옛 버전을 판독하게 된다. 본 논문에서는 트랜잭션이 각 레벨에서 서로 다른 VI를 갖고 가능한 한 최근의 버전을 판독하면서 버전의 수를 동적으로 유지할 수 있는 새로운 방법을 제안한다. 어떤 레벨에서 트랜잭션의 최초 연산이 수행될 때 그 레벨의 버전들에 대한 하향 판독 타임스템프(rdts; read-down time stamp)가 결정된다. 이 방법을 사용할 경우 트랜잭션들은 가능한 한 최근의 버전을 판독할 수 있고 빈번한 갱신에 따른 다중 버전의 유지는 많은 디스크 공간을 차지할 수 있으므로 적절한 시점에서 버전을 생성시키고 더 이상 판독될 필요가 없는 버전들을 제거하여 버전의 갯수를 동적으로 유지하므로써 디스크 공간을 효율적으로 사용할 수 있다. 또한, 제안하는 방법에서는 가능한 한 최근의 갱신을 버전으로 반영할 수 있기 때문에 장기 트랜잭션과 함께 수행되는 단기 트랜잭션에게 가능한 한 최근의 버전을 제공할 수 있고 장기 판독 전용 트랜잭션(LROT; Long ROT)는 VI에 상관없이 성공적으로 완료할 수 있다. 비록 사용되는 버전의 수가 많을수록 가능한 한 최근의 버전을 판독할 수 있고 버전의 수가 적을수록 옛 버전을 판독하는 상쇄관계가 존재하지만 LROT의 경우 VI가 변하여도 철회됨이 없이 성공적으로 수행을 완료할 수 있기 때문에 이러한 상쇄관계는 허용될 수 있다.

3. 관련연구

본 절에서는 다단계 보안 데이타베이스 시스템 환경에서 다중 버전을 사용하여 비밀 경로를 방지하면서 동시성을 제어하는 기법과 이중 버전을 이용한 장기 트랜잭션의 경우에 대해 서술하며 그 문제점을 살펴본다.

[14]에서는 도착되는 트랜잭션들에게 순서스템프를 할당하는 기법을 제안하였다. 트랜잭션들의 레벨이 $L(P) < L(T)$일 때 트랜잭션 T의 순서스템프는 수행중인 트랜잭션 P의 순서스템프 보다 작게, 트랜잭션들의 레벨이 $L(Q) > L(T)$일 때 트랜잭션 T의 순서스템프는 수행중인 트랜잭션 Q 보다 크게 결정된다. 이 기법에서는 하위레벨에서 수행중인 모든 트랜잭션들에게 할당된 것보다 더 작은 순서스템프를 T^H들이 할당받아 수행된다. T^H는 자신 보다 큰 순서스템프를 갖는 T^L이 생성한 것 보다 옛 버전을 읽게 되고 T^H가 읽고자 하는 최근 버전은 하위레벨에서 수행중인 트랜잭션의 결과에 달려 있다. 특히, 하위레벨에서 수행중인 트랜잭션이 장기 트랜잭션이라면 T^H는 매우 옛 버전을 읽게 된다.

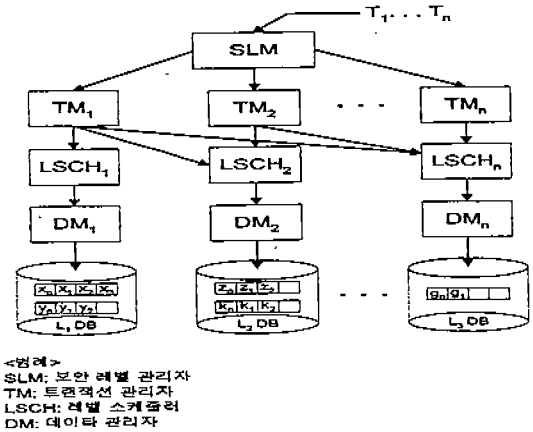
트랜잭션이 시스템에 도착하는 순서대로 타임스템프를 할당받아서 그 타임스템프(ts; timestamp)대로 트랜잭션을 수행하는 기법이 [13]에서 제안되었다. $L(T) > L(P)$ 이고 $ts(T) > ts(P)$일 경우에 하위레벨 트랜잭션 P가 완료될 때까지 상위레벨 트랜잭션 T의 완료는 연기된다. 만일, $L(T) > L(Q)$이고 $ts(T) > ts(Q)$일 때, T가 객체 x의 버전 x_i를 읽은 후에 Q가 x의 새로운 버전 x_j를 트랜잭션 생성한다면, 트랜잭션들이 타임스템프 순서대로 수행되기 때문에 T의 판독연산

$R(x_i)$는 Q 보다 큰 타임스템프를 갖는 $R(x_j)$로 재수행되어야 한다. 하위레벨에 장기 트랜잭션 T^L이 있다면 $ts(T^H) > ts(T^H)$인 T^H의 완료는 오랜 시간 동안 연기될 수 있다. 트랜잭션들이 각각 다른 최근성을 제공받는 기법[12]에서는 트랜잭션들이 원하는 최근의 데이타를 읽을 수 있도록 하였다. 그렇지만 이 기법의 문제는 다른 레벨의 데이타를 접근하는 트랜잭션들이 연속적으로 재수행될 수 있고 몇몇 트랜잭션들이 잠재적으로 기근문제를 갖는다는 것이다. 제출되는 트랜잭션이 수행중인 트랜잭션 보다 더 오래된 버전을 요구할 경우 수행중인 트랜잭션은 기근문제를 갖을 수 있다. 어떤 레벨로 제출된 갱신 트랜잭션 T_1이 하위레벨 객체에 대해 하나의 최근 버전을 요구하고 새로운 트랜잭션 T_2가 동일레벨에서 수행중인 트랜잭션 T_1보다 작은 최근성을 필요로 한다면 두 트랜잭션들 중 하나가 철회된다. 한 데이타에 대해서 다수의 버전을 유지하는 것은 기억공간에 상당한 부담이 된다. 그래서 [15]에서는 디스크 공간 문제, 상위레벨 데이타베이스에 있는 중복 사본으로의 갱신된 값을 전파할 때 발생할 수 있는 보안 문제[16]와 무한한 옛 버전 유지 문제[14]를 개선하기 위해 동일한 VI내에서 두 개의 스냅샷을 사용하는 알고리즘을 제안하였다. 그러나 이 알고리즘에서는 트랜잭션들 사이의 직렬성을 유지하기 위하여 상위레벨에 있는 RD-op가 하위레벨에 있는 옛 스냅샷을 접근하는 경우, 상위레벨에서는 이 RD-op의 수행이 끝날때까지 새로운 스냅샷의 생성이 연기되어진다. 따라서 장기 트랜잭션에는 적절치 못하다.

장기 트랜잭션을 지원하기 위하여 다단계 보안 데이타베이스에 대한 잠금 프로토콜[17]이 제안되었다. 이 프로토콜은 각 객체들에 대해서 두 개의 버전 즉, 옛 버전과 현재 버전을 유지한다. 객체와 동일레벨에 있는 트랜잭션은 현재 버전을 접근하며 하위레벨에서는 옛 버전을 읽는다. VI가 변화된 후에 새로운 버전이 생성되고 현재 버전은 옛 버전으로 변경된다. 이 프로토콜은 한 트랜잭션의 모든 RD-op들은 동일한 VI에서 수행되지만 동일 레벨의 장기 트랜잭션은 VI가 바뀌어도 옛 버전을 판독할 수 있다. 따라서 장기 트랜잭션과 같이 수행되고 있는 트랜잭션들은 장기 트랜잭션이 판독하는 동일한 버전들을 읽어야 한다. 이것은 다른 다중 버전 프로토콜이 갖는 디스크 공간 문제를 두 버전의 사용으로 해결하지만 VI가 길수록 RD-op들은 더 옛 버전을 읽게 되고 VI가 변화할 때 갱신 트랜잭션과 RD-op의 숫자가 증가하면 철회되는 트랜잭션들이 많이 발생한다. 가장 긴 트랜잭션의 길이, 장기 트랜잭션들이 제출되는 정도, 트랜잭션들 사이의 충돌을 반영하여 VI를 효율적으로 정하는 것은 쉽지 않다.

4. 트랜잭션 처리 모델

다단계 보안 트랜잭션 처리를 위한 참조모델 (그림 1)에는 하나의 보안 레벨 관리자(SLM;Security Level Manager)와 각 레벨마다 트랜잭션 관리자(TM;Transaction Manager), 레벨 스케줄러(LSCH;Level SCHeduler), 그리고 데이타 관리자(DM;Data Manager)가 있다. 모든 트랜잭션들은 SLM으로 제출되고 SLM은 트랜잭션의 보안 등급에 따라 해당 등급의 TM에 트랜잭션을 제출한다. 이 때 SLM에서는 제출되는 트랜잭션에게 유일한 타임스템프를 할당한다. 또한, 제출되는 트랜잭션들은 자신이 판독전용인 LROT인지 혹은 갱신 트랜잭션(UPD;UPDater)인지를 SLM에게 알려준다. TM은 하위레벨의 최근 버전을 판독하기 위한 RD-op를 각 하위레벨의 LSCH에게 제출한다. 이 때 TM은 레벨이 동등하거나 하위레벨의 LSCH에게만 연산을 보낼 수 있다. 각 SLM들은 트랜잭션의 타임스템프를 (유일한 보안 등급 분류 첨자)+(보안 등급

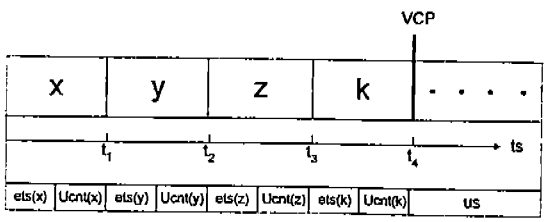


(그림 1) 트랜잭션 처리 모델
(Fig. 1) Transaction Processing Model

갯수)×(임의의 0≤I인 정수)으로 결정한다. 하위레벨의 LSCH는 어떤 T^H의 최초 연산이 수행될 때 가능한 한 최근의 버전을 판독할 수 있도록 rdts를 결정한다. 각 LSCH는 동일레벨이나 상위레벨에 있는 트랜잭션들이 제출한 연산들을 타임스템프 순서 프로토콜에 따라 독자적으로 스케쥴링한다.

상위레벨의 $LSCH_i$가 하위레벨의 $LSCH_j$에게 타임스템프 값을 노출하게 되면 신호 경로가 형성될 수 있기 때문에 신호 경로 방지를 위하여 LSCH들은 서로 직접 통신하여 필요한 정보를 교환할 수 없다.

각 DM에는 트랜잭션에 할당된 타임스템프, 트랜잭션들의 식별자 그리고 보안레벨과 함께 갱신된 값을 유지하는 갱신 리스트(UL; Update List) (그림 2)가 있고 UL에는 모든 트랜잭션들에 대한 갱신 연산을 유지한다. UL은 갱신되어 완료된 데이타 항목과 그 항목의 타임스템프, 그리고 갱신된 횟수(Ucnt; Update counter)를 나타내는 부분으로 구성되고 버전화 검사점(VCP; Versionization CheckPoint)마다 초기화된다. VCP는 드물게 일어나는 갱신을 버전으로 반영하기 위한 정기적인 버전 생성점으로 시스템이나 응용에서 수행되는 데이타 항목들의 갱신 패턴에 따라 적절히 조정할 수 있다. VCP에서는 Ucnt가 0이 아닌 모든 갱신 항목에 대해서 새로운 버전이 생성된다. UL내의 Ucnt가 시스템에서 정한 UCNT를 만족하는 데이타 항목은 새로운 버전으로 생성된다. DM은 VCP까지의 각 데이타에 대한 갱신 횟수를 검사하여 이를 각 데이타의 버전 생성 간격 VI로 정하고 버전화 트랜잭션(VT; Versionized Transaction)을 제출한다. 이때 ts(VT)는 UL의 갱신된 값이 갖는 타임스템프와 동일하다. DM은 VT를 제출하여 새로운 버전을 생성하게 된다.



VCP:버전화 검사점
Ucnt: 갱신횟수
ts:타임스템프
ets:가장 최종 ts
us:갱신상태

(그림 2) 갱신 리스트
(Fig. 2) Update List

UL에서 갱신되어 완료된 값은 VT에 의해서 새로운 버전으로 생성되고 이 버전은 버전화될 때 VT의 타임스템프인 버전화 타임스템프(vts; versionization timestamp)를 갖으며 RD-op를 위하여 rdts를 갖는데 처음 버전이 생성될 때 rdts와 vts는 동일한 값을 갖는다. T^H가 하향판독할 버전의 rdts는 T^H가 가능한 한 최근의 옛 버전을 읽어야 하기 때문에 하위레벨에서 아직 버전화되지 않는 가장 빠른 갱신의 타임스템프와 T^H의 타임스템프중 작은 값이면서 옛 버전들 중 가장 큰 vts를 갖는 버전을 선택한다. 각 하위레벨에서 버전을 읽는데 사용되는 rdts는 그 트랜잭션이 속하는 LSCH에 의해서 결정되기 때문에 각 LSCH는 트랜잭션의 레벨을 구분하지 않고 타임스템프 순서 프로토콜에 따라서 스케쥴링한다. 각 LSCH는 TM에서 제출되는 트랜잭션의 단순 연산(r(x))을 버전 연산($r(x_i)$)으로 번역하며 이 버전 연산은 DM에게 제출되고 DM은 각 데이타에 대한 버전 리스트를 유지하여 버전 연산에 해당되는 버전으로 접근할 수 있도록 한다.

5. 알고리즘

본 절에서는 다단계 보안 동적 다중 버전 제어에 대한 알고리즘을 기술한다.

먼저, 다단계 보안 환경에서 동적 다중 버전 제어 알고리즘을 제시하기 위해 다음의 가정이 필요하다. SLM은 제출된 트랜잭션들을 해당 레벨의 TM으로 보내고, 각 트랜잭션들이 SLM으로 제출될 때 LROT인지 혹은 UPD인지를 알리기 때문에 SLM은 트랜잭션을 구분할 수 있다. SLM은 제출되는 트랜잭션들에 유일한 타임스템프를 할당하고 DM에는 UL이 유지된다. 빈번한 갱신으로 인한 버전 수의 증가 그리고 갱신의 철회로 인한 LROT들의 연속적인 철회를 방지하고 가능한 한 최근에 완료된 값을 판독할 수 있도록, 일단 갱신이 발생하면 UL에 유지시킨 후 UPD들의 수행이 성공적으로 완료되면 갱신된 값을 각 레벨에 있는 데이타베이스로 버전화시킨다. 정보의 부적절한 흐름이 발생하지 않도록 트랜잭션을 처리하기 위하여 상위레벨의 장기 트랜잭션은 ROT들이며 데이타와 동일레벨인 UPD들만 새로운 버전을 생성

할 수 있다.

알고리즘에서의 주요한 고려 사항은 제출된 트랜잭션들에 대한 유일한 타임스템프 할당, 하향판독을 위한 rdts의 결정 그리고 버전 생성과 제거이다. 각 데이타 d, 트랜잭션 T의 레벨에 대하여, $L(d) < L(T)$일 때 T^H이 사용할 수 있도록 다수의 버전을 유지하며 T^H들이 사용하는 버전의 집합을 $hvs = \{x_i, y_j, z_k, ...\}$이라 한다. 또, $L(d) = L(T)$일 때 동일레벨의 트랜잭션이 사용하기 위한 하나의 버전이 유지되며 동일레벨 트랜잭션들이 사용하는 버전을 $cv = \{x_i + 1, y_{j+1}, z_{k+1}, ...\}$이라 한다. cv에 있는 버전은 새로운 버전이 생성될 때 마다 hvs에 추가된다. 여기서, ets(i)는 UL에서 레벨 i로 아직 버전화되지 않는 트랜잭션들의 타임스템프들중 가장 작은 타임스템프 즉, ets(i) = min{ts(unversionized T's from UL at level i}를 의미한다. 다단계 보안 동적 다중 버전들에 대한 동시성 제어(MLS/DMVC;Multi Level Secure Dynamic MultiVersion Concurrency control) 알고리즘의 전체적인 개요를 보면 트랜잭션이 제출될 때 마다 SLM, TM, LSCH, DM에 대한 알고리즘이 호출된다. DM은 새로운 버전이 생성된 후에 ets를 조정해주고 사용되지 않는 버전을 제거한다. 각각에 대한 알고리즘을 살펴보면 다음과 같다.

SLM에서는 도착한 트랜잭션들을 전체적 리스트인 Tr_List에 유지하며 트랜잭션들을 보안 등급별로 분류하여 해당되는 레벨의 TM에게 제출한다. 또한 이들 트랜잭션들에게 타임스템프 생성 규칙에 따라서 유일한 타임스템프를 부여한다.

```
Algorithm for SLM (LROTi's, UPDi's):
for arrived transaction,
1. Assign timestamp according to timestamp generation rule.
2. Insert transactions into Tr_List.
3. Classify each transaction into each level.
4. Submit transactions into each TM with transaction's level.
```

(알고리즘 1) SLM 알고리즘
(Algorithm 1) SLM Algorithm

TM에서는 트랜잭션들을 연산으로 분해하여 해당 레벨의 LSCH에게 제출한다.

LSCH에서는 TM에서 받은 각 연산에 대해서 RD-op의 경우 T_j의 최초 연산이면 $rdts(T_j)$를 결정하며 타임스템프 순서 규칙에 따라 연산들의 레벨을 구분하지 않고 스케줄링하며 단순 연산(r(x))을 버전 연산($r(x_j)$)로 번역하여 DM으로 제출한다.

```
Algorithm for TM ((LROTi's, UPDi's) ∈ level):
1. Decompose transaction into read-op.'s or write-op.'s.
2. IF (L(op.'s)=L(x)) THEN Submit op.'s into current level LSCH.
3. ELSE Submit RD-op.'s into low level LSCH.
```

(알고리즘 2) TM 알고리즘
(Algorithm 2) TM Algorithm

```
Algorithm for LSCHi (RD-ops, UPDs ∈ LSCHi, etsj(i)):
for Rj(x) operation,
IF (Rj(x) is a high level operation) THEN
   IF (Rj(x) is first operation of Tj) THEN
      detremine rdts_old(Tj) = min{for L(i) < L(Tj), ts(Tj), ets(i)}
                               AND max{all vts_old_vers(xi)}
   ELSE  /*Rj(x) is not the first operation of Tj */
      rdts_old(Tj)={rdts to be already selected by first operation}
ELSE IF (Rj(x) is the same level with data) THEN
      schedule Rj(x) according to timestamp rule
      submit to DM
for Wj(x) operation,
submit to DM
```

(알고리즘 3) LSCH 알고리즘
(Algorithm 3) LSCH Algorithm

DM은 LSCH에서 받은 연산들을 수행하고 각 데이타의 Ucnt에 따라 버전을 생성할 것인지의 여부를 결정한다. 알고리즘에서 Ucnt(x)는 데이타 x가 갱신되어 완료된 횟수이고 UCNT는 버전 생성을 위하여 시스템에서 정한 갱신 횟수이다. 새로운 버전은 Ucnt가 UCNT를 만족하거나 VCP 시점에서 생성된다. 갱신 항목들의 Ucnt와 타임스템프는 UL에 유지되고 버전화된 항목들은 타임스템프 순서대로 유지된다. 새로운 버전이 생성되면 사용되는 변수들은 초기화되고 hvs에는 버전이 추가된다.

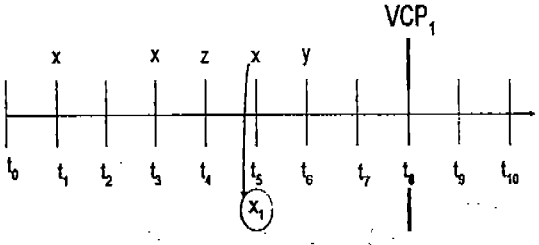
```
Algorithm for DMi (RD-ops, UPDs ∈ LSCHi, etsj(i)):
IF (Rj(x))
 THEN read a version with timestamp determined at LSCH
IF (Wj(x)) THEN increment Ucnt(x)
 IF ((Ucnt(x)==UCNT)) THEN  issue  a VT
 ELSE update UL  /*not generating a new version*/
```

```
IF (current_time==VCP) THEN
 for any data y with its Ucnt not zero in UL,
 issue a VT
IF (VT is issued) THEN
 assign ts(VT) = vts = rdts into new version
 hvs=hvs+{x's version immediately before generated new version}
 cv=cv-{x's version immediately before generated new version}
 delete Tj in UL
IF (Tk is the first operation in UL)
 THEN invokes Algorithm for etsi(j) Adjustment;
IF (there are unnecessary versions)
 THEN invokes Algorithm for Deletion of Unnecessary Version
```

(알고리즘 4) DM 알고리즘
(Algorithm 4) DM Algorithm

앞서 기술한 알고리즘에 대해서 버전 생성의 예제를 살펴보면 다음과 같다.

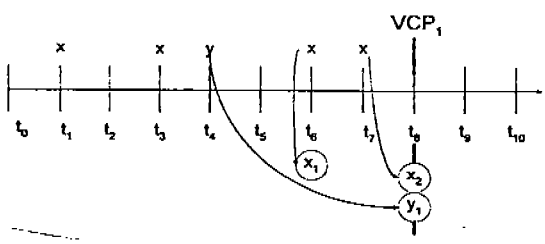
【예제1】 Ucnt(x)=UCNT인 경우



여기서 UCNT=3, VCP=8이라 하자. 각 시점에서의 각 변수의 상태를 보면 다음과 같다. t_1에서 Ucnt(x)=1, t_3에서 Ucnt(x)=2, t_5에서 Ucnt(x)=3이고 Ucnt(x)가 UCNT를 만족하므로 x_1을 생성하고 Ucnt(x)=0으로 초기화된다. □

다음은 한 VCP시점에서 UCNT를 초과했거나 드물게 발생하는 갱신에 대해서 버전을 생성하는 경우이다.

【예제2】 Ucnt(x)>UCNT 혹은 드문 갱신인 경우



여기서 UCNT=3, VCP=8이라 하자. 이 때, 각 시점에서의 데이타 D(x)에 대한 상태를 보면 다음과 같다. t_1에서 Ucnt(x)=1, t_3에서 Ucnt(x)=2, t_4에서 Ucnt(y)=1, t_6에서 Ucnt(x)=3이므로 x_1을 생성, Ucnt(x)=0으로 초기화 한다. t_7에서 Ucnt(x)=1, t_8에서 VCP_1이므로 x_2와 y_1을 생성, Ucnt(x)=Ucnt(y)=0으로 초기화한다. □

DM이 VT를 제출하여 VCP까지에서 버전을 생성시켰으면 해당 레벨의 LSCH의 ets를 조정하기 위하여 이미 버전화된 갱신 값의 ets는 버리고 min{ts(D(x))}를 선택해서 다음의 ets로 정하여 LSCH에게 알린다.

```
Algorithm for etsi(j) Adjustment (after issuing VT):
IF ((Ucnt(x) == UCNT) OR (at a VCPi-1))
  THEN      /* version creation for all D(x) */
   discard etsi-1(j) at VCPi-1
   select min{ts(D(x))} within a VCPi
   submit it as a new etsi(j) to LSCH.
ELSE        /* not create a new version */
   not adjust etsi-1(j)
```

(알고리즘 5) ets 조정 알고리즘
(Algorithm 5) ets Adjustment Algorithm

동일레벨의 트랜잭션들이 사용하는 cv는 VT가 제출될 때마다 상위레벨의 RD-op들을 위하여 주기적으로 hvs으로 추가되기 때문에 많은 옛 버전들이 존재할 수 있다. 따라서, RD-op가 더 이상 접근하지 않는 hvs에 있는 옛 버전들이 제거되면 디스크 공간을 절약할 수 있다. 이를 위하여 세 단계의 불필요한 버전 제거(DUV; Deletion of Unnecessary Version) 알고리즘이 요구되며 DUV를 위하여 윈도우(WDUV; Window for DUV)를 사용한다. WDUV에서 각 버전은 $BP(x_i)$와 $EP(x_i)$를 갖는다. 첫째 단계에서 제거될 버전은 min{ts(RD-op_j) in each level} 보다 더 작고 hvs에 있는 $vts(x_i)$ 중 가장 작은 버전이 선택된다. 이 때 vts는 그 버전을 생성한 트랜잭션의 타임스템프이다. 둘째 단계에서는 선택된 버전에 대한 확인 요청을 SLM에게 의뢰하고 셋째 단계에서는 선택된 버전의 제거 및 유지를 결정하기 위하여 SLM이 Tr_List에서 그 vts와 동일한 타임스템프를 갖는 트랜잭션이

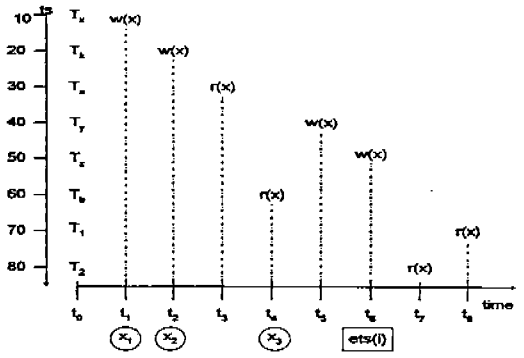
존재하는지의 여부를 확인한다. 수행이 끝나서 Tr_List에 존재하지 않는 트랜잭션이라면 그 버전 x_i는 제거되고 WDUV의 $EP(x_i)$는 $BP(x_j)$로 대치된다.

```
Algorithm for Deletion of Unnecessary Version(ts(RD-op_j)):
 for D(x) ∈ WS in WDUV,
  IF (version order is x_i < x_j)
    THEN BP(x_i) = 1, EP(x_i) = BP(x_j) = i + 1
  IF (rdts < ts(RD-ops))
    THEN Replace rdts with ts(RD-ops)
  ELSE not change
 for a version to be deleted,
 IF (min{vts(x_i)∈hvs}<min{ts(RD-op_j) in each level}) /*1st phase*/
  THEN Send selected vts(x_i) to LSCH_i          /*2nd phase*/
   LSCH_i Hold the vts(x_i) and send SLM
   SLM Retrieve the vts(x_i) in Tr_List          /*3rd phase*/
   Return whether on not deletion for the version
 IF (deletion for the version)      /*vts(x_i) is not in Tr_List*/
   THEN Delete x_i and Replace EP(x_i) with BP(x_j)
 ELSE not Delete x_i and Keep BP(x_i) < EP(x_i)
```

(알고리즘 6) 불필요 버전 제거 알고리즘
(Algorithm 6) Deletion of Unnecessary Version Algorithm

다음은 불필요한 버전을 제거하는 예제이다. 먼저, min{모든 $vts(x_i) \in hvs$} < min{ts(RD-ops) in each level}인 버전을 찾는다. 이 때, vts는 버전을 생성한 트랜잭션의 타임스탬프이므로 그 트랜잭션이 Tr_List에 있는지 여부를 확인하여 제거 및 유지를 결정한다.

【예제4】 불필요한 버전 제거



여기에서 초기의 $rdts(x_1) = vts(x_1) = 10$, $rdts(x_2) = vts(x_2) = 20$이고 $ts(T_a) = 30$, $ts(T_b) = 60$이면, $rdts(T_a)$를 rdts 결정 규칙에 따라서 20으로 결정된다.

$min(ts(RD\text{-}op)\} = 30$이고 $min\{vts(x_i)\} = 10$이므로 x_1을 선택한다. 이 버전의 vts를 SLM에게 전송하고 vts와 같은 타임스탬프의 트랜잭션이 Tr_List에서 없으면 제거한다. □

6. 알고리즘의 정확도

본 절에서는 다단계 보안 환경에서 동적인 다중 버전 제어 알고리즘이 직렬성과 보안성을 보장한다는 것을 증명한다. 먼저, MLS/DMVC의 알고리즘이 트랜잭션의 직렬성을 보장함을 보임에 있어 T^H인 $LROT_i$들은 타임스탬프 순으로 가장 최근의 버전을 판독하기 때문에 직렬가능하다.

【보조정리 1】 직렬화 순서에서 T^H인 LROT은 T^L을 선행한다.

증명) 트랜잭션들이 SLM으로 제출되면 그 트랜잭션은 타임스탬프와 레벨 i를 할당받고 레벨 i 보다 낮은 각 레벨 j에서 트랜잭션의 판독 타임스탬프 $rdts\_old(T_j) = min\{for\ L(i) < L(T_j),\ ts(T_j),\ ets(i)\}$ AND $max\{vts_old_vers(x_i)\}$에 의해서 결정된다. 판독연산은 레벨 i에서 수행중인 트랜잭션들의 타임스탬프와 레벨 i에 제출되어 아직 버전화되지 않은 트랜잭션의 타임스탬프중 가장 작은 타임스탬프 보다 더 작은 모든 버전들 중 가장 큰 vts를 갖기 때문에 레벨 i에서 레벨 i 보다 상위레벨을 갖는 트랜잭션의 연산들은 판독연산뿐이다. 레벨 i에서 레벨 i 보다 상위레벨을 갖는 데이타는 존재하지 않음으로 직렬화 순서에 있어서 상위레벨에서 수행중인 트랜잭션들은 하위레벨에서 수행중인 트랜잭션들을 선행한다. □

MLS/DMVC의 직렬성은 각 버전들이 유일한 rdts를 갖어야함을 이용하여 스케쥴링에 사이클이 발생하면 모순임을 제시하므로써 직렬성이 보장됨을 증명한다.

【정리 1】 MLS/DMVC는 트랜잭션의 직렬성을 보장한다.

증명) MLS/DMVC가 MVSG(S)에서 사이클을 갖게

되면 1-사본 직렬가능한 스케줄을 생성하지 않게 된다. 그러므로, MVSG(S)에 사이클이 존재한다고 가정하자. 각 레벨에서 스케줄은 직렬가능하기 때문에 직렬화 순서 그래프에서 사이클을 생성하지 않는다.

MVSG(S)에 사이클이 생성되려면 둘 이상의 트랜잭션들이 둘 이상의 레벨 스케줄에 존재해야 한다. 즉, $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \ldots T_n \rightarrow T_1$이다. 선분 →은 직렬화 순서와 하향판독하기 위하여 T^H들이 선택한 버전들의 타임스탬프인 rdts_old 순서를 나타낸다. 레벨 $i < L(T_1)$, $L(T_2)$에 대해서 $rdts_old(T_1) < rdts_old(T_2)$이면 선분 $T_1 \rightarrow T_2$가 생긴다. 다시 말해서, T_1는 T_2가 읽은 버전 보다 먼저 생성되어진 옛 버전을 읽는다는 것이다. T_i가 기록한 x의 버전을 x_i라고 할때, $x_i < x_j$는 버전 순서에 있어서 x_i가 x_j를 선행한다는 것이고 $T_i \rightarrow T_j$이다. $x_i < x_j < x_k$인 관계가 존재하고 $w_i[x_i]$ $r_x[x_i]w_k[x_k]$이면 $T_i \rightarrow T_x \rightarrow T_j \rightarrow T_k$이다. 다른 연산들 사이에 충돌이 없다면 동일한 버전을 판독하는 트랜잭션들 사이에 충돌은 없다. 직렬가능한 스케줄에서 hvs에 있는 데이타를 판독하는 연산은 각 레벨에서 cv를 판독하거나 기록하는 연산을 선행한다. $L(T_i) > L(T_j)$이고 T_i와 T_j가 동시에 수행되는 트랜잭션들일 경우 보조정리 1에 의해서 T_i는 T_j를 선행한다.

MVSG(S)에서 사이클이 생성되는 경우를 보면 다음과 같다.

경우1: 레벨 i에서 $T_1 \rightarrow T_2$이고 레벨 j에서 $T_2 \rightarrow T_1$.

레벨 i > 레벨 j, $L(T_1) > L(T_2)$, $ts(T_1) < ts(T_2)$라고 하자. 레벨 i에서 $T_1 \rightarrow T_2$은 레벨 i에서 하향판독하는 T_1과 T_2의 레벨에 상관없이 최대의 갱신 카운터 값을 갖는 VI가 갱신 카운터 보다 커서 변화가 없으면 $rdts_old(T_1) = rdts_old(T_2)$이고 VI가 갱신 카운터 보다 작거나 같아서 변화가 있다면 $rdts_old(T_1) < rdts_old(T_2)$이다. 그래서 $rdts_old(T_1) \leq rdts_old(T_2)$이다. 레벨 j 보다 상위레벨 레벨 i에서 T_1과 T_2의 연산이 존재함으로 T_1과 T_2의 연산은 옛 버전을 읽는 판독연산들이다. 따라서, 레벨 j에서 $T_2 \rightarrow T_1$은 $rdts_old(T_2) < rdts_old(T_1)$이다. 따라서, $rdts\_old(T_1) \leq rdts\_old(T_2) < rdts\_old(T_1)$. 이것은 각 레벨에서 각 트랜잭션들이 유일하게 rdts_old를 갖는다는 사실에 모순이다. 그러므로, MVSG(S)에서는 어떠한 사이클도 생성하지 않는다.

경우2: 레벨 i에서 $T_1 \rightarrow T_2$, 레벨 j에서 $T_2 \rightarrow T_3, \ldots$, 레벨 k에서 $T_m \rightarrow T_n$, 레벨 l에서 $T_n \rightarrow T_1$ 같은 셋이상의 관계가 존재한다. 레벨 i > 레벨 j > ... > 레벨 k > 레벨 l이라고 하자. 레벨 i에서 $T_1 \rightarrow T_2$로부터 T_1과 T_2의 레벨에 상관없이 $rdts_old(T_1) \leq rdts_old(T_2)$를 알 수 있고 레벨 j에서 $T_2 \rightarrow T_3$로부터 T_2과 T_3의 레벨에 상관없이 $rdts_old(T_2) \leq rdts_old(T_3)$를 알 수 있으며 레벨 k에서 $T_m \rightarrow T_n$로부터 T_m과 T_n의 레벨에 상관없이 $rdts_old(Tm) \leq rdts_old(T_n)$를 알 수 있다. 레벨 l에서는 더 높은 레벨에서의 T_n과 T_1의 연산이 존재함으로 이들은 모두 판독연산들이다. T_n과 T_1이 레벨 l에서 옛 버전을 읽고 T_1이 T_n과 충돌이므로 $rdts_old(T_n) < rdts_old(T_1)$이다. 그래서 $rdts_old(T_1) \leq rdts_old(T_2) \leq rdts_old(T_3) \leq \ldots \leq rdts_old(T_m) \leq rdts_old(T_n) < rdts_old(T_1)$이다. 이것은 모든 레벨에서 각 트랜잭션들이 유일한 rdts_old를 갖는다는 사실에 위배된다. 그러므로, MVSG(S)에는 어떠한 사이클도 발생하지 않는다. 따라서, MLS/DMVC는 직렬성을 보장한다. □

MLS/DMVC는 직렬성 뿐만아니라 불법적인 정보 유출, 무결성 위배 등의 문제를 제공할 수 있는 비밀 경로를 방지해야 함으로 보안성이 보장되어야 한다.

【정리 2】 MLS/DMVC는 비밀경로가 발생하지 않는다.

증명) ets(i)와 rdts_old(T)는 각각의 해당 레벨 스케줄러에서만 유지되고 자신의 레벨 스케줄러로 제출되는 트랜잭션들을 동기화하기 위하여 사용된다. 각기 다른 레벨의 스케줄러들은 서로 정보를 교환할 수 없기 때문에 서로 다른 레벨 스케줄러로 제출되는 트랜잭션들이 이러한 자료구조를 공유하므로써 발생하는 비밀경로는 형성될 수 없다. 또한, MLS/DMVC에서 상위레벨 트랜잭션과 하위레벨 트랜잭션들이 서로를 간섭하지 않고 수행되기 때문에 두 트랜잭션들간의 데이타 충돌로 인한 비밀경로가 발생하지 않으며 보조정리 1에 의해서 이로 인한 직렬성은 위배되지 않는다. 그러므로, MLS/DMVC는 보안정책을 만족시킨다. □

따라서, 정리 1과 정리 2에 의해서 MLS/DMVC는 직렬성과 보안성을 보장한다. □

7. 논 의

MLS/DMVC는 각 레벨에서 하향판독할 버전의 타임스템프가 결정되며 더 이상 하향판독될 필요성이 없는 버전들은 제거된다. 트랜잭션이 수행되는 동안에 각 레벨의 rdts_old가 결정되므로 가장 적절한 버전을 트랜잭션들에게 제공할 수 있다. 대부분의 다른 방법[12][13][14]은 모든 레벨에서 동일 시점을 트랜잭션에게 제공하며 불필요한 버전들이 유지된다. [12][13]은 트랜잭션 T 보다 작은 타임스템프를 갖는 트랜잭션에 앞서 트랜잭션 T의 하향판독 연산을 즉시 수행한다면 T가 원하는 최근의 버전을 재판독해야 하므로 재수행 문제가 있다. MLS/DMVC에서는 한 트랜잭션이 LSCH로 제출될 때 하위레벨에서 수행중인 모든 트랜잭션들은 그 트랜잭션이 완료될 때까지 지연되므로 그 트랜잭션은 [14]에 의해서 주어진 것과 같은 최근의 버전을 접근한다. 각 레벨에서 트랜잭션 T의 rdts_old가 결정될 때마다 T 보다 작은 타임스템프를 갖는 모든 트랜잭션들이 완료된다면 T는 적어도 [13]에 의해서 주어진 것과 같은 최근의 버전을 접근한다. T 보다 큰 타임스템프를 갖는 트랜잭션들이 어떤 레벨에서 완료된다면 T는 [13]에 의해서 읽혀진 것 보다 더 최근의 버전을 읽을 수 있다. 그러므로, MLS/DMVC에 따라 각 레벨에서 선택된 최근의 버전은 트랜잭션의 최초 연산이 수행될 때 그 레벨에서 수행중인 트랜잭션들의 완료에 달려 있다.

8. 결 론

다단계 보안 환경에서 트랜잭션 관리 기법은 비밀 경로를 방지하므로써 데이타베이스의 보안성과 트랜잭션들의 직렬성을 보장할 수 있다. 그러나, 비밀 경로를 방지하기 위하여 상위레벨 트랜잭션이 하위레벨의 트랜잭션으로 인하여 철회되거나 지연된다면 정당성이 위배되는 문제점을 갖게 된다. 한 데이타에 대해서 다중 버전이 유지된다면 충돌이 발생된 두 트랜잭션에 적당한 버전을 제공하므로써 상위레벨 트랜잭션의 철회나 지연을 방지할 수 있다. 그러나 두 개 이상의 버전을 유지하는 방법들은 너무 오래된 버전을 판독하거나 버전 유지를 위해 많은 공간을 필요로 한다는 단점을 갖는다. 따라서, 본 논문에서는 이러한 문제들을 해결하고자 MLS/DMVC 알고리즘을 제안하였다. 제안한 방법에서는 데이타 항목의 갱신 횟수 혹은 일정한 시간 간격마다 새로운 버전을 생성하므로써 수행되는 트랜잭션에게 가능한 한 가장 최근의 버전을 제공할 수 있도록 하였고, 불필요한 버전들을 제거해 주므로써 디스크를 효율적으로 사용할 수 있다. 수행되는 트랜잭션들의 직렬성을 보장하기 위하여 T^H의 스케쥴이 직렬가능하지 않는 경우에만 T^H가 연기되므로 트랜잭션을 수행하는데 있어서 기존의 연구 방법에 비해 효율적이다. 그러나, 이중 버전 방법에 비해서 많은 버전이 존재하지만 다중 버전 방법 보다는 부담이 적으면서 장기 판독 전용 트랜잭션들이 효율적으로 수행될 수 있다.

다단계 보안 데이타베이스 시스템 환경에서도 발생할 수 있는 고장이 발생할 수 있는데 이에 대해서 회복이 가능하도록 동적 다중 버전에 대한 회복 관리는 앞으로 더 고려되어야 할 중요한 문제로 남아 있다.

참 고 문 헌

[1] D.E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.

[2] A. Goguen and J. Meseguer, "Security Policy and Security Models", Proc. 1982 Symp. Security and Privacy, Oakland, Calif., Apr., 1982.

[3] K. P. Smith, B. T. Blaustein, "Correctness Criteria for Multilevel Secure Transactions", IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No. 1, Feb., 1996.

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

[5] C. P. Pfleeger, *Security in Computing*, Prentice Hall, Reading, MA, 1989.

[6] S. Castano, *Database Security*, Addison-Wesley, Reading, MA, 1994.

[7] K. Smith, "Execution Ordering for Multilevel Secure Rules", Proceedings, 4th International

Workshop on Research Issues in Data Engineering, Active Database Systems, IEEE Computer Society Press, 1994.

[8] P., Ammann and S. Jajodia, "A Timestamp Ordering Algorithm for Secure Single-Version Multilevel Databases", Proceedings of the 5th IFIP Working Conference in Database Security, 1991.

[9] R. David and S. H. Son, "Design and Analysis of A Secure Two-Phase Locking Protocol", 18th International Computer Software and Applications Conference (COMPSAC'94), Taipei, Taiwan, Nov., 1994.

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

[11] J. N. Gray, The Transaction Concept: Virtues and Limitations, In Proceedings of the 7th International Conference on Very Large Data Bases, Cannes, France, Aug., 1981.

[12] V. Atluri, E. Bertino, and Sushil Jajodia, "Providing different degrees of recency options to transactions in multilevel secure databases", Proc. of the IFIP WG 11.3 Ninth Anual Working Conference on Database Security, Aug., 1995.

[13] S. Jajodia and V. Atluri, "Alternative Correctness Criteria for Concurrent Execution of Transactions in Multilevel Secure Databases", Proc. IEEE Symposium on Security and Privacy, May, 1992.

[14] T. F. Keefe "Multiversion Concurrency Control for Multilevel Secure Database Systems", Proc. IEEE Symposium on Security and Privacy, May, 1990.

[15] P. Ammann, F. Jaeckle, S. Jajodia, "A Two Snapshot Algorithm For Concurrency Control In Mlti-Level Secure Data-bases", Proc. IEEE Symposium on Security and Privacy, May, 1992.

[16] S. Jajodia, B. Kogan, "Data Replication Multilevel Secure Transaction Processing", Proc. IEEE Symposium on Security and Privacy, May, 1990.

[17] S. Pal, "A Locking Protocol for Multilevel Secure Databases Providing Support for Long Transactions", Proc. of the IFIP WG 11.3 Ninth Annual Working Conference on Database Security, 1995.

정 현 철

1987년 조선대학교 계산통계학과졸업(이학사)
1990년 중앙대학교 대학원 전자계산학과(공학석사)
1997년 현재 전남대학교 대학원 전산통계학과 박사과정 수료

관심분야: 데이타베이스 보안, 이동 컴퓨팅, 트랜잭션 관리, 분산 처리 시스템 등

황 부 현

1978년 숭실대학교 전산학과졸업(학사)
1980년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)
1981년~현재 전남대학교 전산학과 교수

관심분야: 데이타베이스 보안, 이동 컴퓨팅, 트랜잭션 관리, 분산 처리 시스템 등