□특 집□

실시간 데이타베이스에서 트랜잭션 스케쥴링

서강대학교 전자계산학과 박 석*
Department of Computer Science, University of Virginia 손 상 혁**

목 차

I. 서 론

실시간 데이타베이스 시스템(Real-Time Database System: RTDBS)은 트랜잭션이 종료시한(deadline)과 같은 시간적 제약조건을 갖는 시스템이다. 시스템의 정확성(correctness)은 논리적 결과 뿐만아니라 결과가 생성되는 시간에 의존한다. RTDBS에서 트랜잭션은 종료시한을 넘기기 전에 완전하게 실행을 끝내도록 스케쥴되어야 한다. 예를 들어 미사일에 대한 데이타를 추적하는 질의 및 갱신은 주어진 종료시한 내에 처리되어야 한다.

RTDBS는 넓은 영역에 걸쳐서 매우 중요하게 응용되고 있다. 즉, 우주항공 산업, 컴퓨터 통합제작(Computer Integrated Manufacturing), 로보틱스, 원자력 플랜트, 교통제어 시스템 등을 들 수 있다. 최근 실시간 컴퓨팅 학술회의[10]에서 연구자들은 전통적인 데이타 모델과 데이타베이스가 시간적 제약을 갖는 응용에는 적합하지 못하므로 공유데이타를 수집, 갱신, 검색함에 있어 시간적 제약조건을 만족시키는 방법에 대한 기초연구가 필요함을 지적했다. 매우 적은 수의 데이타베이스 시스템만이 사용자가 시간적 제약조건을 명시하고 확신할 수 있다. 최근 이러한 사항에 대한 여러 연구가 진행중에 있다[1, 2, 6, 11, 13, 15, 16. 17].

실시간 데이타베이스 시스템의 트랜잭션을 하드(hard)와 소프트(soft) 트랜잭션으로 구분하는 것이 유용하다[17]. 하드 실시간 트랜잭션은 시간적 제약조건이 반드시 만족되도록 보장되어야 하는 트랜잭션을 의미한다. 이러한 트랜잭션이 종료시한을 넘기면 커다란 재앙을 초래한다. 대조적으로 소프트 실시간 트랜잭션도 시간적 제약조건을 가지지만 종료시한 이후에 완료되어도 어느 정도 합당한 이득이 있는 것을 말한다. 소프트 트랜잭션은 시간적 제약조건을 고려하여 스케쥴되지만 종료시한 내에 끝나도록 보장하지는 않는다. 두 가지 종류의 트랜잭션을 지원해야할 필요가 있는 많은 실시간 시스템이 있다. 예측 불가능한 데이타 요구사항에 따라 하드 실시간 종료시한을 보장하는 것은 매우 어렵기 때문에 본 논문에서는 소프트 실시간 데이타베이스 시스템만을 설명한다.

일반적인 데이타베이스 시스템은 두 가지 불충분성 즉, 성능저하와 예측성 부족 때문에 실시간 응용에

* 종신회원
** 정회원

서는 전형적으로 사용되지 않는다. 일반적인 데이타베이스 시스템에서 트랜잭션 처리는 보조 기억장치에 저장되어 있는 데이타베이스의 접근을 요구한다. 따라서 트랜잭션 응답시간은 ms단위의 디스크 접근(access) 지연에 의해서 제한된다. 이러한 데이타베이스는 아직 수초의 반응시간이 사용자들에게 받아들여 질 수 있는 보통의 응용들에는 충분히 적절하다. 그러나 이러한 시스템이 고성능 실시간 응용에는 충분히 빠른 반응을 제공할 수 없다. 고성능을 성취하는 한 가지 방법은 느린 속도의 장치(예, 디스크)를 빠른 속도의 장치(예, RAM)로 대체하는 것이다. 또 다른 사용의 방법은 병행수행 정도(degree)를 높이기 위하여 응용에 특정한 지식을 사용하는 것이다. 예를 들어 트랜잭션 및 데이타와 연관된 의미적 정보를 탐구함으로써 직렬성(serializability)과는 다른 정확성 개념을 사용할 수 있다. 시간적 제약조건을 갖는 데이타베이스 시스템에서 병행수행 제어를 위한 스킴으로는 병행수행의 제약 때문에 직렬성은 너무 강한 정확성 기준일 수 있다[5]. 필요하다면 데이타 일관성은 시간적 제약조건을 만족시키기 위해서 완화될 수 있다.

예측성 관점에서 보면 현재 데이타베이스 시스템은 응답시간 요구조건을 만족시키도록 트랜잭션을 스케쥴하지 않으며, 데이타 일관성을 보장시키도록 보통 데이타를 로크(lock)한다. 로크와 시간제약 스케쥴링은 기본적으로 서로 적합하지 않는다. 우선순위가 낮은 트랜잭션이 우선순위가 높은 트랜잭션을 블럭할 수 있으며 따라서 시간적 제약조건을 만족시키지 못하게 될 수 있다. 결과적으로 실시간 데이타베이스 시스템에 대한 요구사항과 설계는 전통적인 데이타베이스 시스템과는 크게 다르다. 데이타 일관성을 관리하기 위한 새로운 기법이 필요하며, 이 기법은 시간제약 스케쥴링과 적합해야 한다. 따라서 시스템의 응답시간과 시간적 일관성 요구사항을 만족시켜야 한다. 일반적인 데이타베이스 시스템이 성능과 예측성 측면에서 실시간 응용에 적합하도록 어떻게 수정되는가를 고려함이 자연스럽다.

종료시한과 같은 트랜잭션의 시간적 제약조건에 부가하여 트랜잭션의 중요도를 표현하는 긴급성(critical)도 트랜잭션 순위를 계산하는데 고려되어야 한다. 그러므로 실시간 트랜잭션 스케쥴링에서 우선순위와 충돌 해소(conflict resolution)의 특별한 관리는 실시간 데이타베이스 시스템의 예측성과 반응성을 위해서 필수적이다.

데이타베이스 시스템의 병행수행 제어 이론과 실시간 태스크 스케쥴링이 많은 발전을 해왔지만, 병행수행 제어 프로토콜과 실시간 스케쥴링 알고리즘 사이의 상호작용에는 별 주의가 없었다. 일반적인 데이타베이스 병행수행 제어에서 종료시간을 만족시키는 것은 고려되지 않았다. 일반적인 병행수행의 목적은 병행수행 정도를 높이려는 것과 데이타 일관성을 위반하지 않으면서 평균 응답시간을 빠르게 하는 것이다. 반대로 실시간 스케쥴링에서는 디스크들은 서로 독립적이며 공유 데이타에 대한 접근을 동기화하는데 소비한 시간은 실행시간과 비교할 때 무시할만하다고 가정하는 것이 보통이다. 여기서의 목적은 시간적 제약조건을 만족시키기 위하여 CPU 이용도와 같은 자원 이용도를 최대화하는 것이다. 직렬성을 보장시키기 위한 데이타 일관성은 실시간 스케쥴링에서는 고려되지 않는다. 따라서 공유 데이타의 일관성을 보장시키는 문제는 무시한다. 추가적으로 전통적인 실시간 시스템은 디스크의 자원요구 사항에 대한 사전지식을 가정한다.

실시간 데이타베이스 시스템에서 도전하는 문제는 세 가지 제약조건 즉 데이타 일관성, 트랜잭션 정확성(correctness)과 트랜잭션 종료시한에 관련된 병행수행과 자원 이용도를 최대화하는 실시간 스케쥴링과 병행수행 제어 프로토콜에 대한 이론을 창출하는 것이다. 최근의 여러 프로젝트에서 시간적 제약조건의 효율적이며 정확한 관리를 가능하게 하기 위하여 시간적 제약조건을 데이타베이스 시스템에 추가하는 문제를 조사연구 하였다[6, 17]. 이러한 목표를 성취하는 데는 여러가지 어려움이 있다. 예를 들어, 데이타베이스 접근 연산은 수행중에 디스크 I/O, 로깅(logging), 버퍼링(buffering) 등이 요구되는 것에 따라서 수행시간이 크게 다르다. 더우기 병행수행 제어는 수행중인 트랜잭션의 취소(abort) 또는 연기(delay)를 초래한다.

실시간 데이타베이스 시스템은 많은 새로운 문제를 제기한다. 실시간 트랜잭션과 데이타에 대한 적절한 모델은 무엇인가? 실시간 제약조건을 기술하기 위해 적절한 언어 구조(construct)는 무엇인가? 트리거링(triggering)을 기술하고 계산하는데 있어 적절한 메카니즘은 무엇인가? 트랜잭션들을 어떻게 스케쥴하는가? 본 논문에서는 마지막 두 문제 즉, 트랜잭션 스케쥴링과 병행수행 제어와 관련된 문제를 연구하여 병행수행 제어를 위한 낙관적 방법을 사용한 통합 스케쥴러를 제안한다. 본 논문의 구성은 다음과 같다.

2절에서 실시간 데이타베이스 시스템의 특징과 몇 가지 주요연구 과제를 간단히 설명한다. 3절에서 통합 실시간 로킹 프로토콜을 설명한다.

II. 스케쥴링과 병행수행 제어

태스크 각각의 시간적 제약조건을 고려하는 일반적인 실시간 시스템은 데이타 일관성 문제를 대부분 무시한다. 또한 전형적으로 예측가능한 데이타 요구들을 가지는 간단한 태스크들만 다룬다. 실시간 태스크 스케쥴링에서는 모든 태스크들이 선점 가능하다고(preemptable) 가정한다. 그러나 독점적 기록 모드에 있는 화일 자원을 사용하는 태스크를 선점하면 관련된 태스크들이 불일치한 정보를 읽게 만드는 결과를 가져온다.

실시간 시스템과는 달리 전통적인 데이타베이스 시스템은 충돌하는 트랜잭션 사이의 직렬성 순서를 고려하며 트랜잭션의 평균 응답시간을 줄이는데 촛점을 두었을 뿐, 트랜잭션의 시간적 제약조건이나 종료시한을 중요시하지 않았다. 예를 들어 이단계 로킹 (two phase locking: 2PL) 프로토콜을 블럭킹(blocking)과 복귀(rollback)로써 트랜잭션들의 병행적 데이타 접근을 동기화시키는데 이것은 트랜잭션의 시간적 제약조건은 위반할 수도 있다.

실시간 데이타베이스 시스템(RTDBS) 스케쥴링의 목적은 데이타 일관성 제약조건 뿐만아니라 시간적 제약조건도 만족시키는 데 있다. 실시간 태스크 스케쥴링 방법은 실시간 트랜잭션 스케쥴링을 위해 확장될 수 있지만, 연산 스케쥴링의 데이타 일관성을 유지시키기 위해 병행수행 제어 프로토콜은 여전히 필요하다. 일반적인 접근방법은 2PL과 같은 기존의 동시성 제어 프로토콜을 사용하면서 시급한 트랜잭션들을 우선적으로 스케쥴하는 시간제약(time-critical) 트랜잭션 스케쥴링 방법을 적용한다. 기존의 모든 병행수행 제어 방법이 블럭킹과 복귀에 의해 트랜잭션의 병행적 데이타 접근이 동기화되므로 그러한 접근방법은 병행수행 제어 프로토콜에 의해 제한되는 단점을 가진다. 실시간 시스템, 데이타베이스 시스템, 실시간 데이타베이스 시스템의 특성이 간단히 〈표 1〉에 나타나 있다.

병행수행 제어 프로토콜은 충돌하는 트랜잭션들 사이에 직렬화 순서를 만들어낸다. 비실시간(non real-time) 병행수행 제어 프로토콜에서는 이런 순서를 만들어내는 데 시간적 제약조건은 중요한 요소가 아

〈표 1〉 실시간 시스템, 데이타베이스 시스템, 실시간 데이타베이스 시스템의 특성

기준 / 시스템	정확성 제약조건	자원의 예측성	성능 목표	스케쥴링
실시간 시스템	시간 (종료시한)	예측가능한 데이타 요구사항과 실행시간	시간과 연관된 성능 척도를 최대화(종료시한 만족)	CPU 위주
데이타베이스 시스템	데이타 일관성	예측 불가능한 데이타 요구사항	평균 응답시간을 최소화	I/O 위주
실시간 데이타베이스 시스템	시간과 데이타 일관성	예측가능한 CPU요구와 예측 불가능한 데이타 요구사항	긴급성의 잇점과 데이타 일관성을 가지는 종료시한의 보상	CPU와 I/O 연산의 혼합

니므로 RTDBS에서는 명백한 결점이 된다. 예를 들어, 이단계 로킹 프로토콜에서 직렬화 순서는 동적으로 만들어지고 충돌하는 트랜잭션 사이의 순서에 따라 공유 데이타에 접근한다. 즉, 직렬화 순서는 전혀 융통성(flexibility)을 갖지 못하는 수행기록(history)으로 제한된다. 우선순위가 높은 트랜잭션 T_H가 우선순위가 낮은 트랜잭션 T_L이 소유한 독점 로크(exclusive lock)를 요청하면 T_L을 취소시키거나, T_L의 수행이 끝나기를 기다려야 한다. 그러나 이들 중 어떤 선택도 불충분하다. 보수적인 2PL은 블럭킹을 사용하는데 RTDBS에서는 이런 블럭킹이 우선순위 반전(priority inversion)을 일으킬 수 있다. 우선순위 반전은 우선순위가 높은 트랜잭션이 우선순위가 낮은 트랜잭션에 의해 블럭되는 현상을 말한다. 또 다른 방법은 우선순위 반전이 일어났을 때 우선순위가 낮은 트랜잭션을 취소시키는 것인데 이 방법은 취소된 트랜잭션으로 인한 작업이 낭비되고 시간 위주 스케쥴링의 성능을 떨어뜨릴 수도 있다. 실시간 트랜잭션을 위한 로크에 기초한 병행수행 제어 메카니즘들을 가지는 다양한 스케쥴링 방법이 연구되어 왔다[1, 2]. 또한, 타임스탬프에 기초한 병행수행 제어 메카니즘이 우선순위 반전을 피하기 위해 연구되었다[4].

초기에 실시간 운영체제(real-time operating system)에서 태스크 스케쥴링 프로토콜로 개발된 우선순위 상한 프로토콜(priority ceiling protocol)이 RTDBS에 확장되었다[15]. 이 프로토콜은 2PL에 기초한 보수적인 접근방법으로 충돌을 해결하기 위해 블럭킹만을 사용하고 복귀는 사용하지 않는다. 일반적인 데이타베이스 시스템에서는 블럭킹과 복귀를 포함하는 것이 최적의 성능을 가지는 것으로 알려졌다[19]. RTDBS에서도 비슷한 결과를 기대할 수 있다. 몇개의 우선순위가 낮은 트랜잭션들을 취소시키고 후에 재시작(restart) 시킴으로써 우선순위가 높은 트랜잭션들이 종료시한을 만족시킬 수 있고, 결과적으로 시

스템 성능을 개선시킬 수 있다. 우선순위 상한 프로토콜의 단점은 트랜잭션의 수행 이전에 모든 트랜잭션에 대한 지식을 알아야 한다는 것이다. 이것은 대부분의 데이타베이스 시스템에서는 만족시키기 어려운 조건이다.

병행수행 제어 프로토콜이 트랜잭션의 시간적절성(timeliness)을 충족시키기 위해, 병행수행 제어 프로토콜이 생성한 직렬화 순서는 트랜잭션의 우선순위를 반영해야 한다. 낙관적(optimistic) 방법[12]은 이런 목적을 성취하는 가능한 한 가지 방법이다. 검증 단계(validation phase)에서 충돌을 해결하기 때문에, 실제로 버려진(discarded) 트랜잭션이 다른 트랜잭션들을 취소시키지 않고 트랜잭션의 우선순위가 고려됨을 보장할 수 있다. 낙관적 방법에 기초한 몇개의 병행수행 제어 프로토콜이 제안되었다[8, 9, 16]. 낙관적 병행수행 제어 프로토콜의 중요한 요소는 트랜잭션들의 취소 혹은 완료가 결정되는 검증단계이다. 이들 중 몇 가지(예, [9]에서 OPT-WAIT 프로토콜)는 충돌이 발생하는 우선순위가 높은 트랜잭션이 완료될 때까지 우선순위가 낮은 트랜잭션이 기다리도록 하는 우선순위 대기(priority wait)와 같은 우선순위에 기초한 충돌 해결 메카니즘과 함께 사용된다. 그러나 트랜잭션은 검증단계에서만 취소되기 때문에 어떤 트랜잭션이 재시작해서 종료시한 내에 끝내기는 너무 늦게 되므로 검증단계에서 충돌을 찾는 이 방법은 시스템의 예측가능성을 감소시킨다. 로크에 기초한 병행수행 제어 프로토콜이 수행중인(active) 트랜잭션들의 직렬화 순서를 동적으로 조정하는 메카니즘을 지원한다면 통합 스케쥴러(integrated scheduler)는 이러한 문제를 해결할 수 있다. 통합 스케쥴러가 다음 절에서 제시되는데, 이것은 우선순위에 기초한 로킹과 낙관적 방법을 통합한 방법이다.

앞에서 연구되어야 할 또 다른 중요한 문제는 트랜잭션 처리에 있어서 올바른 수행에 대한 다른 개념이다. RTDBS에서 시간적 제약조건이 데이타 일관성보다 더 중요하다는 근거에 기초한다면, 시간적 제약조건을 만족시키기 위해 어느 정도까지는 데이타베이스 일관성을 일시적으로 희생시키는 시도가 이루어져야 한다[13]. 이것은 실시간 데이타베이스의 새로운 일관성 모형에 기초하는데, 실시간 데이타베이스에서는 외부적인 데이타 일관성(external data consistency: 데이타 객체의 값들이 데이타베이스 외부세계의 옳은 값들을 나타낸다)을 유지하는 것이 내부적인 데이타 일관성(internal data consistency: 어떤 데이타도 일관성 제약사항을 위반하지 않는다)을 유지하는 것보다 중요하다. 몇몇 응용에서는 약화된 일관성(weaker consistency)이 허용될지라도[7], 직렬성보다 덜 엄격한 범용의 일관성 기준은 아직 제안되지 않았다. 문제는 일시적인 비일관성 상태가 수행중인 트랜잭션에 영향을 끼쳐서 이 트랜잭션들의 완료는 비일관성 상태가 제거될 때까지 지연될 필요가 있다는 것이다. 그렇지 않다면, 완료된 트랜잭션 조차도 복귀될 필요가 있을 수 있다. 그러나, 실시간 시스템에서 몇몇 작업(action)은 원래로 되돌릴 수 없다.

트랜잭션 스케쥴링에서 의미상의 정보(semantic information)와 다중버젼 데이타의 사용이 RTDBS 응용에 흔히 제안된다[14, 18]. 다중버젼은 시간에 따라 변하는 값으로서의 데이타를 감독하는데 유용하다. 그러한 상황에서, 데이타의 값에 의해 나타나는 경향은 적당한 작업을 유발(trigger)하는데 사용된다[11]. 다중버젼을 사용하는 또 다른 목적은 동시성의 정도(the degree of concurrency)를 증가시키고 데이타 뷰의 계승을 제공함으로써 트랜잭션 거절(rejection)의 가능성을 줄이는 것이다.

다중버젼을 효과적으로 사용하기 위해 해결되어야 하는 많은 문제점들이 있다. 예를 들어, 트랜잭션의 구버젼 선택은 트랜잭션이 볼 수 있는 상태에서 요구된 일관성을 보장해야 하는 것이다. 이외에도, 구버젼을 저장하는데 필요한 저장공간(storage) 관리 문제가 있다.

III. 통합 스케쥴러

실시간 데이타베이스에서 실시간 스케쥴링은 두 개의 스케쥴링 메카니즘(트랜잭션 스케쥴링과 연산 스케쥴링)으로 구성된다. 새로운 병행수행 제어 방법을 찾기 위해서 연산 스케쥴링에 촛점을 두어야 한다.

트랜잭션의 기록 연산을 연기함으로써 직렬화 순서에 기반을 둔 과정의 트랜잭션의 실행순서가 완화될 수 있다. 이것은 시간 제약과 트랜잭션의 중요도에 따라 트랜잭션들 사이의 직렬화 순서를 동적으로 조절할 수 있게 한다. 동작(active) 트랜잭션의 직렬화 순서를 동적으로 조정하기 위하여 우선순위 종속적(priority-dependent)인 로킹 프로토콜을 제안한다. 이 프로토콜은 높은 우선순위를 가지는 트랜잭션이 아직 완료되지 않은 낮은 우선순위를 가지는 트랜잭션들에 의해 블럭되지 않게 하면서 우선순위가 높은 트랜잭

션이 먼저 실행되도록 한다. 그리고 충돌 연산의 경우라도 낮은 우선순위의 트랜잭션이 취소되지 않을 수도 있다.

예를 들어, 우선순위가 낮은 T1과 우선순위가 높은 T2라는 트랜잭션이 있다. T2가 데이타 객체 x를 읽기 전에 T1이 쓴다. 만약 2PL에서 x에 충돌이 발생하면 두 트랜잭션 사이에 어떠한 다른 충돌 연산이 없음에도 불구하고 T2는 T1을 취소시키거나 T1이 기록 로크를 해제할 때까지 블럭되어야 한다. 이것은 이미 T1→T2의 직렬화 순서가 결정되었기 때문에 T2는 T1을 결코 선행할 수 없다. 그런 충돌이 일어나면 우선순위 종속적인 로킹 프로토콜에 의해 두 트랜잭션의 직렬화 순서는 T2를 고려하여 조정될 것이다. 즉, T2→T1으로 조정되면 T2는 블럭되지 않고 T1도 취소되지 않는다.

3.1 우선순위 종속적 로킹 프로토콜

임의로 도착하는 트랜잭션을 가지는 하나의 프로세서를 가정해 보자. 각 트랜잭션은 시스템에 들어올 때 초기 우선순위(initial priority)와 시작-타임스탬프(start-timestamp)를 할당받는다. 초기 우선순위는 종료시한과 다른 트랜잭션과 비교하여 상대적인 중요성을 나타내는 트랜잭션의 긴급성(criticality)를 기준으로 삼는다. 초기 우선순위에 시작-타임스탬프가 덧붙여져 스케줄링에 사용되는 실제 우선순위(actual priority)가 만들어진다. 트랜잭션의 우선순위를 언급할 때는 항상 타임스탬프가 덧붙여진 실제 우선순위를 의미한다. 타임스탬프는 유일하기 때문에 각 트랜잭션의 우선순위로 사용할 수 있다. 트랜잭션들이 초기 우선순위가 같으면 시작 타임스탬프에 의해 구별되고 취소된 트랜잭션은 재시작(restart)시에만 새로운 우선순위를 할당받는다.

스케줄된 모든 트랜잭션은 준비 큐(ready queue) 안에 대기하고 있다. 단지 준비 큐안에 있는 트랜잭션들만이 실행할 수 있도록 스케줄된다. 어떤 트랜잭션이 블럭되면 그 트랜잭션은 준비 큐에서 제거된다. 블럭이 해체되면 다시 준비 큐안에 첨가되지만 여전히 CPU에 할당되기 위해 기다려야 할지도 모른다. 어떤 트랜잭션이 준비 큐안에 있지만 실행하지 않으면 '연기된다(suspended)'라고 한다. 한 트랜잭션이 I/O 연산 중일 때 블럭되고 그 연산이 끝나자마자 블럭이 해제된다. 그리고 로크 요청이 허가되기를 기다릴 때 역시 블럭된다.

각 트랜잭션의 실행은 판독 단계(read phase), 대기 단계(wait phase), 기록 단계(write phase)로 나누어진다. 이것은 낙관적인(optimistic) 방법과 유사하다. 판독 단계 동안에는 하나의 트랜잭션이 데이타베이스로부터 읽어서 자신의 지역 작업영역에 쓴다. 그 작업이 끝나면, 트랜잭션은 대기 단계에서 완료(commit)를 기다린다. 이 세 가지 단계 중 어느 하나의 단계에 있는 트랜잭션을 동작(active) 트랜잭션이라 한다. 만약 동작 트랜잭션이 기록 단계에 있으면 곧 완료되고 데이타베이스에 기록한다. 이런 접근방법은 판독-기록 충돌을 위한 2PL과 기록-기록 충돌을 위한 TWR(Tomas' Write Rule)을 사용하는 통합 스케줄러를 기반으로 한다. TWR은 너무 늦게 도착한 기록 요청을 거절하는 대신에 무시해 버리는 규칙이다.

트랜잭션은 다음과 같은 순서로 진행된다.

```
트랜잭션={시작 ( );
          판독단계;
          대기단계;
          기록단계;
          }
```

이러한 접근방법에서 일관된 방법으로 판독과 갱신을 행해야 할 다양한 데이타구조가 있다. 그러므로 어느 시점에서 단지 하나의 프로세서에게 데이타 구조를 갱신할 수 있도록 할 임계구역(critical section)이 있다고 가정해야 한다. 그리고 임계구역은 최대한으로 동시수행할 수 있는 데이타 구조로 묶여 있다고 가정하다.

3.2 판독단계(Read Phase)

판독 단계에서는 트랜잭션 각각의 지역 작업영역(local workspace)에 있는 데이타 사본에 기록 연산을 수행하는 것 이외에는 일반적인 트랜잭션의 수행이다. 그런 기록 연산을 예비기록(prewrite)라고 하는데, 예비기록 연산의 장점은 어떤 트랜잭션이 취소되었을 때 데이타베이스에 있는 내용을 변화시키지 않고 그 트랜잭션의 지역 작업영역의 데이타를 없애므로서 간단히 회복시킬 수 있다는 점이다. 여기에서는 $r_T[x]$, $w_T[x]$, $pw_T[w]$로 트랜잭션 T가 데이타 객체 x를 판독, 기록, 예비기록 한다고 표시한다.

우선순위에 기초한 로킹 프로토콜을 사용하여 판독 단계에서 동작 트랜잭션들 사이에 일어나는 판독-예비기록 충돌이나 예비기록-판독 충돌을 동기화 시킨

다.

어떤 트랜잭션이 한 데이타 객체에 판독(예비기록) 연산을 하기 전에 먼저 판독(쓰기) 로크를 얻어야 한다. 만약 트랜잭션이 자신에 의해 쓰여진 데이타를 읽으려면 즉시 자신의 작업영역에서 사본을 얻을 수 있고 판독 로크도 필요하지 않다.

각 로크는 그 로크를 소유하는 트랜잭션의 우선순위를 포함하고 있다. 로킹 프로토콜은 우선순위가 낮은 트랜잭션이 끝나기 전에 우선순위가 높은 트랜잭션이 끝나야 한다. 이것은 두 트랜잭션 사이에 충돌이 일어나면 직렬화 순서상에서 우선순위가 높은 트랜잭션은 우선순위가 낮은 트랜잭션보다 선행되어야 함을 의미한다. 트랜잭션의 우선순위를 고려하는 CPU 스케쥴링 방법과 함께 우선순위가 낮은 트랜잭션이 이미 완료되었거나 기록 단계에 있는 경우를 제외하고는 우선순위가 높은 트랜잭션이 먼저 완료되도록 스케쥴해야 한다. 만약 우선순위가 낮은 트랜잭션이 우선순위가 높은 트랜잭션보다 먼저 끝나려고 할 때는 그 트랜잭션의 완료가 우선순위가 높은 트랜잭션의 취소를 유발하지 않는다는 것을 확인할 때까지 기다려야 한다. 트랜잭션이 판독 단계 중에는 데이타베이스에 기록하지 않으므로 기록-기록 충돌은 고려하지 않는다.

동작 트랜잭션 T1은 T2보다 우선순위가 낮다고 가정하자. 다음은 충돌의 4가지 경우와 직렬화 순서상에서 트랜잭션들의 종속관계를 나타내고 있다.

(1) $r_{T2}[x]$ $Pw_{T1}[x]$ ⟶ T2→T1

(2) $pw_{T2}[y]$ $r_{T1}[y]$ ⟶ T1→T1(지연된 판독)

or

T1→T2 (즉시 판독)

(3) $r_{T1}[x]$ $pw_{T2}[x]$ ⟶ T1→T1

(4) $pw_{T2}[y]$ $r_{T1}[y]$ ⟶ T2→T1 (즉시 판독)

or

T1→T2 (지연된 판독)

(1)의 경우는 우선순위가 낮은 트랜잭션 이전에 우선순위가 높은 트랜잭션이 끝나는 원칙에 맞는다. (2)의 경우는 지연된 판독을 선택해야 한다. 즉, T1은 T2가 완료되고 x를 데이타베이스에 쓸 때까지 x를 읽지 못한다. 그러므로 T1→T2(즉시 판독)은 제외한다. (3)은 제시된 원칙을 위반한다. 만약 T1이 이미 완료되지 않았다면 T1은 보통 취소된다. T1을 취소시키지 않는다면 T1은 T2 이전에 완료되어야 하고 T2는 블럭될 것이기 때문이다. 그러나 이미 T1의 작업이 끝났다면 즉, 대기단계에 있으면 T1을 취소시키는 것을 피한다. 왜냐하면 작업이 끝난 트랜잭션의 취소는 시스템 성능을 상당히 저하시키기 때문이다. 그 동안에 T2가 T1에 의해 블럭되는 것을 원하지도 않는다. 그러므로 그런 충돌이 발생하고 T1이 대기단계에 있다면 T2가 완료하기 전에 T1이 완료할 기회를 줄 수 있도록 T2가 완료하기 이전까지는 T1을 취소시키지 않는다. (4)의 경우에서 만약 T1이 이미 완료되고 기록 단계에 있다면 T1이 x를 쓴 후에 T2가 x를 읽도록 T2를 연기시킨다. 이 블럭킹은 T2에게 있어 그리 심각한 문제는 아니다. 왜냐하면, T1이 이미 기록단계에 있고 x에 기록을 곧 끝낼 수 있다고 기대할 수 있기 때문이다. T2는 T1이 꼭 전체 기록 단계를 끝낸 후가 아니더라도 데이타베이스에 x기록을 끝내자마자 x를 읽을 수 있다. 그러므로 T2는 오랜기간 동안 블럭되지 않는다. 그렇지않고 T1이 아직 완료되지 않았다면 즉, T1이 판독 단계나 대기단계에 있다면 T2는 데이타베이스로부터 즉시 x를 읽어야 한다. 왜냐하면 우선순위가 높은 트랜잭션이 우선순위가 낮은 트랜잭션보다 먼저 수행해야 한다는 원칙에 입각해서이다.

트랜잭션들이 수행되고 충돌 연산이 일어남에 따라 직렬화 순서상의 종속관계에 대한 정보가 유지되어야 한다. 이런 정보를 유지하기 위하여 각 트랜잭션에 before-trans-set, after-trans-set, before-count를 결부시킨다. before-trans-set(after-trans-set)은 직렬화 순서에서 이 트랜잭션을 선행해야 하는 (뒤따라야 하는) 우선순위가 낮은 모든 동작(active) 트랜잭션을 포함한다. before-count는 직렬화 순서에서 이 트랜잭션을 선행해야 하는 우선순위가 높은 모든 트랜잭션의 수를 포함한다. 두 트랜잭션 사이에 충돌이 발생하면 종속관계가 결정되고 before-trans-set, after-trans-set, before-count의 값이 적절하게 변한다. 위에서 논의된 것을 간단히 요약하여 우선순위 종속적 로킹 프로토콜을 아래와 같이 정의한다.

LP1. 트랜잭션 T가 데이타 객체 x에 대해 판독 로크를 요청한다.

for each transaction t holding a write lock on x(i.e. the holder) *do*

if the priorty of the holder(t) is > the requester(T) or the holder in write phase,

then deny the lock and exit;

endif

```
enddo
for each transaction t holding a write lock on x do
  if the holder is a member of before-trans-set
     of the request
  then    abort the holder
  else
        if the holder is not a member of after-
           trans-set of the request
        then add the holder to after-trans-set of
             the request and increase before-
             count of the holder by one
        endif
  endif
enddo
grant the read lock
```

LP2. 트랜잭션 T가 데이타 객체 x에 대해 기록 로크를 요청한다

```
for each transaction t holding a read lock on x
do
  if the priority of the holder(t) is>the requester(T)
  then
       if the requester is not a member of after-
          trans-set of the holder
       then add the requester to after-trans-set of
            the holder and increase before-count of
            the requester by one
       endif
  else
       if the holder is in wait phase
       then
            if the holder is a member of after-
               trans-set of the request
            then abort the holder
            else add the holder to after-trans-set of
                 the requester
            endif
       else
            if the holder is in read phase
            then abort the holder
            endif
       endif
  endif
enddo
grant the write lock
```

LP1과 LP2는 로크가 요청되었을 때 로크 관리자가

로크소유자 / 로크요청자	판독 단계		대기 단계		기록 단계
	판독	기록	판독	기록	기록
판 독					
기 록					

(가) 로크 요청자의 우선 순위가 낮을때

로크소유자 / 로크요청	판독 단계		대기 단계				기록 단계
			판 독		기 록		
	판독	기록		*	**		기록
판 독							
기 록							

(나) 로크 요청자의 우선 순위가 높을때

로크가 허가됨 로크 소유자가 취소됨 로크 요청자가 블럭됨

* 로크 소유자가 로크 요청자의 after-trans-set의 멤버(member)이다
** 로크 소유자가 로크 요청자의 before-trans-set의 멤버(member)이다

(그림 1.A) 로크 호환성 테이블

로크 요청자 / 로크 소유자	판 독	기 록
판 독		
기 록		

(가) 로크 요청자의 우선 순위가 낮을때

로크 요청자 / 로크 소유자	판 독	기 록
판 독		
기 록		

(나) 로크 요청자의 우선 순위가 높을때

로크가 허가됨 로크 소유자가 취소됨 로크 요청자가 블럭됨

(그림 1.B) 높은 우선순위 스킴을 동반한 2PL의 로크 호환성 테이블

실행하는 프로시쥬어이다. 충돌 때문에 로크가 거부되면 충돌 로크가 해제될 때까지 요청을 연기시킨다. 그리고 로크가 당장 허가될 수 있는지 결정하기 위하여 처음부터 다시 로킹 프로토콜이 적용된다. (그림 1)은 충돌이 발생했을 때 취할 수 있는 행동이 나타난 로크 호환성 테이블(lock compatibility table)이다. 이 호환성은 로크를 소유하고 있는 트랜잭션의 우선순위와 로크를 요청하는 트랜잭션의 우선순위, 로크의 타입, 로크 소유자의 현재 단계에 의존한다. 같은 로크 타입임에도 불구하고 로크 소유자와 로크 요구자의 우선순위에 따라 다른 행동을 취할 수 있다. 그러므로 테이블의 각 엔트리(entry)는 서로 다른 행동을 반영하는 하나 이상의 블럭(block)을 포함할 수 있다.

지금부터 로킹 프로토콜을 가지고 동시에 몇개의 트랜잭션들에 의해 판독 로크와 기록 로크된 데이타 객체에 대하여 살펴보자. 2PL과는 달리 공유 로크와 독점 로크로 간단히 구분되지는 않는다. (그림 1.B)에서는 높은 우선순위 스킴을 동반한 2PL 기법의 호환성을 나타내는데 우선순위가 높은 트랜잭션은 결코 우선순위가 낮은 트랜잭션에 의해 블럭되지 않음을 볼 수 있다. (그림 1.A)와 (그림 1.B)를 비교해 보면 제안된 로킹 프로토콜이 많은 융통성(flexible)을 가지므로 블럭킹과 취소가 더 적게 일어난다. (그림 1)을 살펴보면 우선순위가 낮은 트랜잭션이 대기 단계에 있을 때도 취소된다. 제안된 로킹 프로토콜에서는 우선순위가 높은 트랜잭션이 결코 완료되지 않은 우선순위가 낮은 트랜잭션과의 충돌 때문에 블럭되거나 취소되는 일이 없도록 한다. 같은 조건하에서 2PL보다는 우선순위가 낮은 트랜잭션을 취소시키는 경우도 적다.

트랜잭션이 도착하자마자 실행할 수 있도록 한다. 먼저 트랜잭션이 판독 단계에서 데이타 객체를 읽거나 예비기록을 하려고 하면 로크를 요청한다. 그 로크는 허가되거나 아니면 로킹 프로토콜에 맞지 않아 트랜잭션이 취소될 수도 있다. 트랜잭션을 취소하려면 아래의 프로시쥬어를 실행시킨다.

```
abort=(
        release all locks;
        for each transaction t of after-trans-set of
            T do
              decrease before-count of t by one;
              if before-count of t is equal to zero
                 and is in the wait phase,
              then unlock it;
              endif
        enddo
        delete an aborted transaction(T) from the
        phase it belongs to;
        remove reference to T from before-trans-set
        and after-trans-set of others;
        ).
```

3.3 대기 단계(Wait Phase)

대기 단계는 트랜잭션이 완료될 때까지 기다리는 단계이다. 단지 직렬화 순서상에서 그 트랜잭션을 선행하는 우선순위가 높은 모든 트랜잭션이 완료됐거나 취소됐을 때만 트랜잭션 T가 완료할 수 있다. T의 before-count는 그런 우선순위가 높은 트랜잭션의 갯수이므로 before-count가 0이어야만 T가 완료할 수 있다. 대기단계에 있는 트랜잭션이 완료할 수 있을 때, 즉, before-count가 0일 때, 그 트랜잭션은 기록 단계로 넘어가고 그 트랜잭션의 모든 판독 로크를 해제한다. 최종 타임스탬프가 트랜잭션에 지정되면 절대적인 직렬화 순서가 된다. 프로시쥬어는 다음과 같다.

```
wait=(
   waiting:=TRUE;
   while(waiting) do
     if T can be committed (before-count=0),
     then switch T into write phase and attach a final
          time-stamp to T using the system time-
          stamp;
          for each transaction of before-trans-set of
              T do
            if t is in the read or wait phase,
            then abort t;
            endif
          enddo
          waiting:=FALSE;
     else let T be blocked;
     endif
   enddo
release all read locks;
   for each transaction t of after-trans-set of T do
     if t is in the read or wait phase,
     then decrease before-count by one
          if before-count is equal to zero and is
             in the wait phase,
```

then unblock t;
endif
endif
enddo
).

대기 단계에 있는 트랜잭션은 다음 두 가지 이유로 취소된다. 첫째는, T가 계속해서 모든 로크를 소유하면서 완료되지 않았을 경우에 [예 1]에서 처럼 로킹 프로토콜에 의해 높은 우선순위의 트랜잭션이 요구하는 로크와 충돌하므로 T는 취소된다. 둘째는, 직렬화 순서상 T를 뒤따르는 높은 우선순위의 트랜잭션이 완료된다고 하면, 그런 트랜잭션이 완료될 때 T가 아직도 그 트랜잭션의 before-trans-set에 있으면 T는 취소된다.

[예 1] 동작 트랜잭션 T1은 T2보다 우선순위가 낮다고 가정하자. 트랜잭션은 도착하자 마자 실행시킨다. T1은 $pw_{T1}[x]$ $r_{T1}[x]$, T2는 $r_{T2}[x]$ $pw_{T2}[x]$라고 하자. 다음 두 실행 순서를 고려해 보자.

가) $H1 = pw_{T1}[y]$ $r_{T2}[y]$ $r_{T1}[x]$ $pw_{T2}[x]$

첫번째 기록-판독 충돌은 (4)의 경우이다. 그러므로 그들 사이의 트랜잭션 종속관계는 T2→T1(즉시 판독)로 놓을 수 있다. 그러면 T1는 T2의 after-trans-set에 포함되고 T1의 before-count는 1로 지정된다. 그후 T2에서 y에 대한 판독 로크가 허락된다. 두번째 판독-기록 충돌은 (3)경우에 속한다. $r_{T1}[x]$후에 T1은 대기 단계에 있고 현재 트랜잭션의 종속관계는 T1→T2이다. 이 직렬화 순서는 앞의 T2→T1에 위배된다. T1은 T2의 after_trans_set에 속해있으므로 T1은 취소된다.

나) $H2 = r_{T1}[x]$ $pw_{T2}[x]$ $pw_{T1}[y]$ $r_{T2}[y]$

첫번째 x에 대한 판독-기록 충돌은 (3)의 경우에 속한다. 그러므로 트랜잭션의 종속관계는 T1→T2이다. T1이 판독 단계에 있으므로 T1은 취소된다. 만약 기록 연산의 순서를 $pw_{T2}[x]$ $pw_{T1}[y]$에서 $pw_{T1}[y]$ $pw_{T2}[x]$로 수정해 준다면 T1은 대기 단계에 있게 된다. 그러면 T1은 T2의 before_trans_set안에 포함되고 T2에게 x에 대한 로크가 허가된다. 그러나 T1이 T2의 before_trans_set의 원소이므로 T1은 취소된다.

그러므로, 두개의 동작 트랜잭션의 순서가 H1 또는 H2 중 어떤 것을 포함하던지 간에 우선순위가 높은 트랜잭션은 실행되고 우선순위가 낮은 트랜잭션은 취소된다.

[예 2] 세개의 동작 트랜잭션 T1, T2, T3를 가정하자. T3는 우선순위가 가장 높고 T1은 우선순위가 가장 낮다. 또한 이 트랜잭션들은 도착하자마자 실행시킨다. T1은 $r_{T1}[x]$, T2는 $pw_{T2}[x]$ $r_{T2}[y]$, T3는 $pw_{T3}[y]$라고 하자. 다음 실행 순서를 고려해 보자. T1, T2는 $r_{T1}[x]$, $r_{T2}[y]$ 수행 후에 대기 단계에 있다고 하고 T3를 진행시키려고 한다.

$H3 = r_{T1}[x]$ $pw_{T2}[x]$ $r_{T2}[y]$ $pw_{T3}[y]$

H3에서는 (3) 경우에 속하는 기록-판독 충돌이 두번 일어난다. 첫번째 충돌에서는 트랜잭션의 종속관계가 T1→T2이다. $r_{T1}[x]$의 수행 후에 T1은 대기 단계에 있고 T2의 before-trans-set에 첨가된다. T2에게 x에 대한 기록 로크를 허가한다. 두번째 충돌에서는 종속관계가 T2→T3이고 첫번째 충돌의 경우와 같은 방법으로 다룬다. $r_{T2}[y]$ 수행 후에 T2는 대기 단계에 있고 T3의 before-trans-set에 첨가된다. 그리고 T3에게 기록 로크를 허가한다.

T3가 $pw_{T3}[y]$ 이후에 완료된다면 이미 [예 1]에서 보다시피 T2가 T3의 before-trans-set에 포함됐기 때문에 T2는 취소된다. 다음 T1이 완료될 수 있다. 만약 T3가 취소되면 T2는 완료되고 T1은 취소된다.

그러므로 T3의 완료 여부에 따라 T1, T2는 완료될 수 있다. 그 이유는, 이미 작업이 종료된 우선순위가 낮은 트랜잭션으로 우선순위가 높은 트랜잭션과 충돌을 일으킬 때에도 완료하기 위해 대기하기 때문이다. 일반적으로, 일련의 트랜잭션의 실행에 H3와 같이 연속적인 판독-기록 충돌이 발생하면, 마지막 트랜잭션 이전의 트랜잭션들 (예, H3에서 T3) 반정도가 완료될 것이다. 이것은 전통적인 로킹 프로토콜과 더불어 통합된 우선순위에 기초한 로킹 프로토콜(integrated priority-based locking protocol)에 의해 성능 향상에 중요한 요인이 된다.

이 에에 2PL 프로토콜을 적용한다고 가정하자. 우선순위가 높은 트랜잭션이 우선순위가 낮은 트랜잭션이 확보하고 있는 데이타에 로크를 요청하면, 일반적으로 우선순위가 높은 트랜잭션을 대기시키거나 우선순위가 낮은 트랜잭션을 취소한다. 첫번째 방법을 선택했을 경우에는 T1이 x에 판독 로크를 유지하기 때문에 T2는 T1에 의해 블럭된다. 두번째 방법을 선택했을 경우에는 T1은 T2가 x를 예비기록 했을 때 T2에 의해 취소되며, T2는 T3가 y를 예비기록 했을

때 T3에 의해 취소될 것이다. 이 예는 통합된 우선순위에 기초한 로킹 프로토콜을 사용함으로써 얻어지는 잇점을 명확히 보여주고 있다.

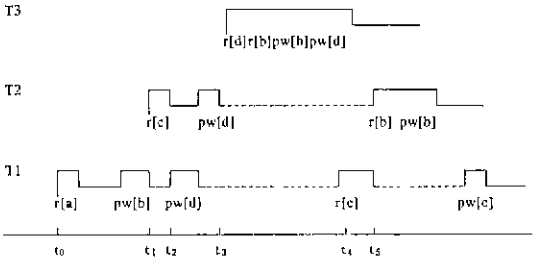
3.4 기록 단계 (Write Phase)

트랜잭션이 일단 기록 단계에 있으면, 완료(commit)한 것으로 생각한다. 모든 완료 트랜잭션들은 최종 타임스탬프(final-timestamp)에 의해 직렬화될 수 있다. 기록 단계에서, 트랜잭션이 하는 유일한 작업은 데이타베이스에 영구적인 갱신을 하는 것이다. 데이타 항목은 트랜잭션의 지역적 작업영역(local workspace)에서 데이타베이스로 복사된다. 각각의 기록 연산 후에, 해당 기록 로크는 해제된다. Thomas' Write Rule(TWR)이 여기에 적용된다. 각 트랜잭션의 기록 요구는 데이타 관리기(data manager)에 보내지고, 데이타 관리기는 데이타베이스에 기록 연산을 수행한다. 트랜잭션은 최종 타임스탬프를 가지고 기록 요구를 한다.

각각의 데이타 객체에 대해서, 기록 요구는 단지 오름차순의 타임스탬프 순서로 데이타 관리기에 보내진다. 타임스탬프 n을 가진 데이타 객체 x에 대한 기록 요구가 데이타 관리기에 보내진 후에, n보다 작은 타임스탬프를 가진 x에 대한 기록 요구 중 어떤것도 데이타 관리기에 보내지지 않을 것이다. 기록 요구는 데이타 관리기에 의해 버퍼에 저장된다. 데이타 관리기는 선입선출(FCFS) 또는 가장 우선순위가 높은 선택에 의해 기록 요구를 처리할 수 있다. 새로운 요구가 도착했을 때, 같은 데이타 객체에 대한 또 다른 기록 요구가 버퍼에 존재한다면 타임스탬프가 적은 요구는 무시된다. 따라서 각각의 데이타 객체에 대해서 버퍼에는 단지 하나의 기록 요구만이 존재할 수 있으며, TWR을 보장한다.

[예 3][1]에는 세개의 트랜잭션이 있다. T3는 가장 높은 우선순위를 가지고 있고, T1은 가장 낮은 우선순위를 가지고 있다. T1은 t_0의 시간에 도착해서 데이타 객체 a를 읽는다. 이것은 페이지 폴트(page fault)를 유발한다. I/O 연산후에, b에 대해 예비 기록을 수행한다. 그리고 나서 t_1의 시간에 T2가 도착하고 T1을 선점(premption)한다. t_2의 시간에 T2는 c를 읽는데 이것 역시 또 다른 페이지 폴트이다. 그래서 T2는 I/O 연산을 위해 블럭되고 T1은 실행을 계속한다. T1이 d에 대해 예비 기록을 실행한 후에 T2는 I/O 연산을 끝마치고 다시 T1을 선점한다. T1이 단지 기록 로크를 소유한 d에 T2는 예비 기록을 실행한다. t_3의 시간에 T3가 도착해서 T2를 선점한다. T3는 우선 T1과 T2에 의해 기록 로크되어 있는 d를 읽는다. 그러므로 T3의 after-trans-set은 {T2, T1}이 되고, T2와 T1의 before-count는 모두 1이 된다. 그런 후 T3는 b를 읽는데, 이것은 T1이 기록 로크를 보유한 것이다. T1이 이미 T3의 before-trans-set에 포함되어 있기 때문에 변화된 것은 없다. T3은 b에 예비 기록을 수행하고, d에 예비 기록을 실행한다. 이 두 데이타 객체에 대해서 다른 트랜잭션이 판독 로크를 소유하고 있지 않기 때문에, T3에 기록 로크가 곧바로 허용된다. t_4의 시간에 T3는 기록 단계로 변환된다. T2와 T1의 before-count 모두 0이 된다. 이제 T2가 수행되어야 하지만, T3가 기록 로크를 가진 b를 읽어야 하므로 T2 대신에 T1이 실행된다. T1은 T2가 판독 로크를 소유한 c를 읽는다. t_5의 시간에 T3는 b에 대한 기록을 끝내고 로크를 해제하면, T2는 T1을 선점하고 실행을 계속한다. T2는 T1이 기록 로크를 소유한 b를 읽는다. 이제 T2의 before-trans-set은 {T1}이 되고 T1의 before-count는 1이 된다. T2는 b에 예비 기록을 한 후, 기록 단계로 변환되면 T1의 before-count는 다시 0이 된다. T1은 다시 실행되어 c에 예비 기록을 실행한 후 기록 단계로 변환된다.

[예 3]



[1] 낮은 단계의 실선은 해당 트랜잭션이 페이지 폴트(page fault)로 인해 I/O 연산을 수행 중이거나 기록 단계에 있음을 나타낸다. 낮은 단계의 점선은 해당 트랜잭션이 일시 중지(suspension)되어 있거나 블럭(block)되어 있거나, 어떠한 I/O 연산도 수행중이지 않은 상태를 나타낸다. 높은 단계의 선은 트랜잭션이 수행중임을 나타낸다. 선이 존재하지 않는 부분은 트랜잭션이 아직 도착하지 않았거나 이미 완료했음을 나타낸다

이 예에서, 가장 시급한(즉, 우선순위가 높은) 트랜잭션으로 가정된 T3는 가장 나중에 도착했어도 가장 먼저 끝마쳤다. 가장 덜 시급한 트랜잭션으로

가정된 T1은 가장 마지막에 완료했다. 새 트랜잭션 중 어느 것도 취소되지 않았다. 위의 예에서 2PL을 사용한다고 가정하자. 충돌이 발생했을 때, 우선순위가 높은 트랜잭션이 기다리게 한다고 가정한다면, T2와 T3은 T1이 d에 기록 로크를 소유하고 있기 때문에 블럭된다. 충돌시에 우선순위가 낮은 트랜잭션을 취소시킨다면, T1은 T2가 d에 예비 기록을 할 때 T2에 의해 취소되고, T2는 T3가 d를 읽을 때 T3에 의해 취소된다. 이 예는 2PL에 대한 프로토콜의 장점을 설명한다.

RTDBS에 대해서, 낙관적인 방법이 로크에 기초한 비관적 방법에 비해 광범위한 시스템 부하(load) 측면과 자원 가용성(resource availability) 측면에서 능가한다는 것은 놀라운 일이다. 왜냐하면 일반 데이타베이스 시스템, ([예 3])의 병행수행 제어 프로토콜에 대한 성능 연구에서 자원이 제한되었을 때 로킹 프로토콜이 낙관적 방법보다 좋은 성능을 보인다고 결론지었기 때문이다[9]. 단정적 결론을 내리기 전에 이 분야에 대한 좀 더 많은 실험적 연구뿐만 아니라 이론적인 연구가 필요하다.

여기 제시된 우선순위에 기초한 로킹 프로토콜이 기록 연산을 지연시키고 충돌 트랜잭션들에 직렬화 순서를 동적으로 조정함으로써 우선순위에 기초한 로킹과 낙관적 방법을 통합한 반면, RTDBS를 위한 낙관적 병행수행 제어를 사용한 다른 방법들이 있다. 예를들면, Wait-50 프로토콜[9]은 종료시한에 기초를 둔 우선순위 정보를 실시간 데이타베이스 시스템에 사용하기 위한 낙관적 병행수행 제어 알고리즘에 적용한다. 이 방법은 트랜잭션의 충돌 상태를 감독하고 우선순위가 낮은 트랜잭션이 기다려야 하는 때와 기다려야 하는 기간을 동적으로 결정하는 대기 제어 메카니즘(wait control mechanism)에 특징이 있다. 로크에 기초한 프로토콜은 트랜잭션들 사이의 판독-기록 충돌을 분석하여, 우선순위가 높은 트랜잭션에 우선권을 주는 방법으로 충돌을 해결한다. 반면, Wait-50에서 충돌 해결은 단지 점증 단계에서만 행해진다. 충돌이 발견되고, 그 트랜잭션의 전체 충돌 크기에 대한 우선순위가 높은 트랜잭션의 비율이 50보다 클 때에만 트랜잭션이 기다리게 된다. 실제로 Wait-50은 충돌이 발생한 우선순위가 높은 트랜잭션을 항상 기다리는 OPT-WAIT의 확장이다. 충돌을 늦게 발견하기 때문에, Wait-50은 늦게 재시작을 해서 충돌 트랜잭션의 수를 증가시키고 결과적으로 좀 더 많은 트랜잭션이 재시작하게 된다.

우선순위에 기초한 로킹 프로토콜에서, 두 트랜잭션 간에 판독-기록 충돌이 발생했을 때, 직렬화 순서는 우선순위가 높은 트랜잭션에 우선권을 주는 순서를 따른다. 같은 두 트랜잭션에 또 다른 충돌이 발생했다면, 새로운 직렬화 순서가 받아들여질 것이다. 새로운 직렬화 순서가 이전의 직렬화 순서와 일치하지 않는다면, 두 트랜잭션이 함께 완료할 수 없기 때문에 우선순위가 낮은 트랜잭션은 어떤 직렬화 순서를 취하든 취소되어야 한다. 이것은 [예 1]에 나타나 있다. 만약 직렬(완료) 순서를 변화시켜서 두 트랜잭션 모두 완료할 수 있다면, [예 4]에서처럼 우선순위가 낮은 트랜잭션을 블럭시킨다.

그러므로 우선순위 종속적 로킹 프로토콜은 우선순위가 높은 트랜잭션에 우선권을 줌으로써 판독-기록 충돌을 효과적으로 다룬다. 게다가, 기록-기록 충돌은 TWR에 의해 해결된다. 이 두 가지를 고려하여, 스케쥴러는 직렬화 순서를 동적으로 조정할 수 있다. 그러면, 트랜잭션이 나중에 무시(discard)될 다른 트랜잭션에 의해서 희생되는 낭비된 희생(waisted sacrifice)의 문제가 남는다.

[예 4] 세개의 수행중인 트랜잭션 T1, T2, T3에 대해 T3가 가장 높은 우선순위를 가지고 있고, T1이 가장 낮은 우선순위를 가지고 있다고 가정하자. 트랜잭션은 도착하는대로 실행시킨다. T1은 $pw_{T1}[x]$, T2는 $r_{T2}[x]$ $pw_{T2}[y]$, T3는 $r_{T3}[y]$라고 하고 다음의 실행 순서를 생각하자. T1과 T2는 T3가 완료하거나 취소되기 전에 대기 단계(wait phase)에 있다고 가정한다.

$$H4 = pw_{T1}[x]\ r_{T2}[x]\ pw_{T2}[y]\ r_{T2}[y]$$

H4에서, (4)의 경우에 속하는 두개의 기록-판독 충돌이 있다. 첫번째 충돌후에, 트랜잭션의 종속관계는 T2→T1이라고 가정된다. 그리고 T2의 after-trans-set은 T1이 되고, T1의 before-count는 1만큼 증가된다. 그리고 z에 대한 판독 로크가 T2에 허용된다. 비슷하게 두번째 충돌에서 T3→T2가 가정되고 T3의 after-trans-set은 T2가 되고 T2의 before-count는 1만큼 증가한다. 이제 T3가 완료하거나 취소된다면, T2와 T1의 순서로 완료할 수 있다. 그러므로 T1과 T2의 취소는 OPT-WAIT에서와 유사하게 발생하지 않는다. 이 예는 우선순위가 높은 트랜잭션의 완료가 반드시 대기중인 트랜잭션을 재시작하게 하지는 않는다는 것을 설명한다. 그러나 데이타 충돌은 모든 로크를 보유한 채 대기 단계에 있는 T1과 T2 때문에

증가될 것이다.

우선순위가 낮은 트랜잭션이 우선순위가 높은 트랜잭션의 실행을 지연시키는 우선순위 반전은 시기적절한 수행을 할 수 없게 하고 RTDBS에서 스케줄링을 어렵게 한다. 우선순위에 기초한 로킹 프로토콜은 우선순위 반전을 감소시키도록 하고, 종료시한 이전에 수행된 트랜잭션의 수를 증가시키도록 한다. 우선순위 반전을 방지하는 새로운 프로토콜로 우선순위 반전의 문제가 자원에 대해서 요구자-소유자(requester-holder)관계를 나타내는 대기 관계(wait-for relation)와 트랜잭션 사이의 우선순위 관계를 사용하여 형식화되었다[4].

IV. 결 론

본 논문에서는 실시간 데이타베이스 시스템의 특징인 정확성 제약조건, 예측성, 성능 목표 등을 설명하였다. 그리고 실시간 데이타베이스 시스템을 위한 트랜잭션 스케줄링과 병행수행 제어와 관련된 문제를 연구하였다. 시간적으로 긴급한 스케줄링에서, 연산 스케줄링 측면에 촛점을 맞추었으며 충돌해소를 위해서 우선순위를 기초로 하는 로킹과 낙관적 방법을 통합한 스케줄러를 제안하였다. 이 방법은 우선순위에 기초한 병행수행 제어 메카니즘이다.

통합 스케줄러에서 트랜잭션의 수행은 낙관적 병행수행 메카니즘과 유사하게 트랜잭션의 기록연산을 연기함으로써 직렬화 순서에서 과거의 실행순서에 의해 부과된 제약이 완화될 수 있다. 동작 트랜잭션들의 직렬화 순서를 동적으로 조정하기 위해서 우선순위에 종속적인 로킹 프로토콜을 제안했다. 트랜잭션의 우선순위가 종료시한과 긴급성과 같은 시간적 제약조건들을 반영한다고 가정한다. 판독 기록(read-write) 충돌을 위해서 2PL 프로토콜을 사용하고, 기록-기록(write-write) 충돌을 위해서 토마스 기록규칙(TWR)을 사용한다. 이 프로토콜은 우선순위가 높은 트랜잭션이 결코 우선순위가 완료되지 않은 낮은 트랜잭션에 의해서 블럭되지(blocked) 않으면서 우선순위가 높은 트랜잭션을 먼저 실행되도록 하는 방법이다. 즉 우선순위가 낮은 트랜잭션이 이미 완료되었고 기록단계에 있는 경우를 제외하고는 우선순위가 낮은 트랜잭션 전에 우선순위가 높은 트랜잭션이 완료되도록 스케줄한다.

이 통합 스케줄러는 우선순위가 높은 스킴을 동반한 2PL 프로토콜[1]보다도 늦은 재시작(late restart) 수와 충돌 트랜잭션 수를 감소시킨다. 왜냐하면 이 방법은 가능하면 트랜잭션 사이의 판독-기록 충돌을 일찍 분석하며 불필요한 충돌해소 연기를 피하면서 우선순위가 높은 트랜잭션을 선호하여 해결한다. 이 방법은 시스템의 병행수행 수준을 감소시킬 필요없이 가능한 트랜잭션이 시간적 제약조건 내에 끝내도록 허용하는 기능을 갖고 있다. 이 방법은 예측불가능한 데이타를 요구하는 응용에 적합하다.

참 고 문 헌

1. R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: a Performance Evaluation", Proc. of the 14th VLDB Conf., Sep. 1988, pp. 1~12.
2. R Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data", Proc. of the 15th VLDB Conf. Aug. 1989, pp. 385~396.
3. R. Agrawal *et al.*, "Concurrency Control Performance Modeling: Alternatives and Implications", ACM Trans. on Database Systems, Vol. 12, No. 4, Dec. 1987.
4. O. Babaoglu, K. Marzullo, and F. Schneider, "Priority Inversion and its Prevention", Dept. of Computer Science, Cornell Univ., Feb. 1990.
5. P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley Publishing Company, 1987.
6. A. P. Buchmann, D. R. McCarthy, H. Hsu, and U. Dayal, "Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control", 5th Data Engineering Conf. Feb. 1989. pp. 470~480.
7. H. Garcia-Molina, "Using Semantic Knowledge for Transaction Processing in a Distributed Database", ACM Trans. on Database Systems, Vol. 8, No. 2, Jun. 1983, pp. 186~213.
8. J. R. Haritsa, M. J. Carey, and M. Livny. "On Being Optimistic about Real-Time Constraints", ACM PODS Symposium, Apr. 1990, pp. 331~343.
9. J. R. Haritsa, M. J. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control", 11th IEEE Real-Time Systems Symposium, Dec. 1990, pp. 94~103.
10. 12th IEEE Real-Time System Symposium. San

Antonio, Texas, Dec. 1991.
11. H. Horth, "Triggered Real-Time Databases with Consistency Constraints", Proc. of the 16th VLDB Conf., Aug. 1990.
12. H. Kung and J. Robinson, "On Optimistic Methods for Concurrency Control", ACM Trans. on Database Systems, Vol. 6, No. 2, Jun. 1981, pp. 213~226.
13. K. Lin, "Consistency Issues in Real-Time Database Systems", Proc. 22nd Hawaii Intl. Conf. System Sciences, Jan. 1989.
14. J. W. S Liu, K. J. Lin, and X. Song, "Scheduling Hard Real-Time Transactions", 5th IEEE Workshop on Real-Time Operating Systems and Software, May 1988, pp. 112~260.
15. L. Sha, R. Rajkumar, S. H. Son and C. Chang, "A Real-Time Locking Protocol", IEEE Trans, on Computers, Vol. 40, No.7, Jul. 1991, pp. 793~800.
16. S. H. Son, S. Park and Y. Lin, "An Integrated Real-Time Locking Protocol," 8th IEEE Conf. on Data Engineering, Feb. 1992, pp. 527-535.
17. S. H. Son, editor. ACM SIGMOD Record, Vol. 17, No. 1, Special Issue on Real-Time Database Systems, Mar. 1988.
18. X., Song and J. Liu, "Peformance of Multiversion Concurrency Control Algorithms in Maintaining Temporal Consistency", COMPSAC '90, Oct. 1990.
19. P. Yu and D. Dias, "Concurency Control using Locking with Deferred Blocking", 6th Int. Conf. Data Engineering. Feb. 1990, pp. 30~36.
20. S. Vrbsky and K. J. Lin, "Recovering Imprecise Transactions with Real-Time Constraints", Symp. Reliable Distributed Systems, Oct. 1988, pp. 185~193.

박 석

1978 서울대학교 계산통계학과(학사)
1980 한국과학기술원 전산학과(석사)
1983 한국과학기술원 전산학과(박사)
1989 ~1990 미국 버지니아 대학 (Visiting Associate Professor)
1983 ~현재 서강대학교 전자계산학과 교수
관심 분야 : 데이타베이스 시스템, 실시간 시스템, 시스템 프로그래밍

손 상 혁

1976 서울대학교 전자공학과(학사)
1978 한국과학기술원 전기 및 전자공학과(석사)
1984 미국 메릴랜드 대학(석사)
1986 미국 메릴랜드 대학(박사)
관심 분야 : 실시간 시스템, 데이타베이스 시스템, 운영체제