분산 실시간 트랜잭션 처리 시스템의 동시실행 제어와 원자적 종료를 위한 시간 구동형 스케쥴링 기법 연구

김 진 환†

요 약

분산 실시간 트랜잭션 처리 시스템에서 데이타의 복제는 유용성의 향상 및 여러 노드에서 발생한 트랜잭션들을 병행적으로 처리할 수 있기 때문에 성능을 향상시킬 수 있다. 데이타의 일관성과 실시간적 제약성을 충족하기 위해서 동시실행 제어 및 원자적 종료 프로토콜이 시간 구동형 스케쥴링 과정에 통합될 필요가 있다. 기존의 동시실행 제어 프로토콜들에게 발생하는 중단 현상은 트랜잭션들의 종료시한을 만족할 수 있는 스케쥴링이 어렵기 때문에 시간 구동형 스케쥴링에 적합하지 않다. 복제된 데이타의 일관성 유지 그리고 스케쥴링 가능성 및 예측가능성을 향상시키기 위하여 제시된 시간 구동형 스케쥴링 기법은 중단 시간을 최소화하고 우선순위에 따른 직렬가능성 순서를 생성하는 낙관적 동시실행 제어 프로토콜을 통합하였다. 분산 환경에서 트랜잭션의 성공적인 종료를 보장하는 원자성도 유지된다. 시간 구동형 스케쥴링 기법의 성능 분석 결과 및 구체적인 과정들이 기술된다.

A Study for Time-Driven Scheduling for Concurrency Control and Atomic Commitment of Distributed Real-Time Transaction Processing Systems

Jinhwan Kim†

ABSTRACT

In addition to improved availability, replication of data can enhance performance of distributed real-time transaction processing system by allowing transactions initiated at multiple nodes to be proecessed concurrently. To satisfy both the consistency and real-time constraints, it is necessary to integrate concurrency control and atomic commitment protocols with time-driven scheduling methods. Blocking caused by existing concurrency control protocols is incompatible with time-driven scheduling because they cannot schedule transactions to meet given deadlines. To maintain consistency of replicated data and to provide a high degree of schedulability and predictability, the proposed time-driven scheduling methods integrate optimistic concurrency control protocols that minimize the duration of blocking and produce the serialization by reflecting the priority of transactions. The atomicity of transactions is maintained to ensure successful commitment in distributed environment. Specific time-driven scheduling techniques are discussed, together with an analysis of the performance of this scheduling.

† 정 회 원:한성대학교 정보전산학부 전임강사
논문접수:1996년 2월 9일, 심사완료:1996년 3월 27일

1. 서 론

최근 실시간성이 강조됨에 따라 비행 추적과 방어 시스템, 공장 설비의 감시와 제어, 핵 발전제어 등의 다양한 응용 분야에서 분산 환경의 실시간 트랜잭션 시스템이 적용된다[1, 2]. 각 트랜잭션마다 종료시한(deadine)이라는 시간적 제약성이 설정되는 실시간 트랜잭션 처리 시스템의 중요한 목표는 종료시한을 보장하면서 가능한 많은 트랜잭션들을 종료시키는 것이다[3]. 따라서 실시간 트랜잭션 처리 시스템의 스케쥴링 기법은 트랜잭션들의 시간적 제약성을 최대한 준수해야 하며 데이타에 대한 일관성을 보장하기 위하여 트랜잭션들의 실행 순서를 정확히 제어하여야 한다[4]. 즉 실시간성과 데이타의 일관성을 동시에 보장하기 위하여 실시간 트랜잭션 처리 시스템에서는 실시간 스케쥴링 프로토콜과 동시실행 제어 프로토콜이 통합될 필요가 있다[5].

그러나 서로 다른 두 프로토콜들을 실시간 트랜잭션 처리 시스템에서 통합하는 것은 현실적으로 어려움이 많다. 실시간 스케쥴링 프로토콜에서는 주어진 태스크(task)들에 대하여 최적한 스케쥴링이 수행되기 위해서는 CPU, 디스크등의 I/O 장치, 버퍼 등의 물리적 자원에 대한 특별한 알고리즘이 각기 사용되며 대부분 NP-hard 문제들로 귀결된다[6, 7]. 특히 통신망을 기반으로 하는 분산 실시간 환경에서는 통신 지연시간에 대한 스케쥴링 기법도 통합되어야 한다[8, 9]. 우선순위가 사용되는 기존의 실시간 스케쥴링 프로토콜이 동시실행 제어 프로토콜과 연관된 경우 우선순위가 높은 트랜잭션이 우선순위가 낮은 트랜잭션에 의하여 지연되는 우선순위 전도(inversion) 현상이 발생할 수 있다[2]. 또한 우선순위가 높은 트랜잭션이 우선순위가 낮은 트랜잭션으로부터 CPU를 선점(preemption)하더라도 보유하던 나머지 자원에 대한 잠금 장치를 해결할 수 없는 경우 지연 현상이 발생하게 된다[10]. 따라서 실시간 태스크 스케쥴링 프로토콜의 우선순위에 관련된 선점 기법 등이 실시간 트랜잭션 처리 시스템에 적용될 경우 우선순위 전도 현상등이 빈번히 발생하게 되며 트랜잭션의 종료시한 보장이 어렵게 된다. CPU등의 물리적 자원에 대한 스케쥴링 알고리즘이외에 논리적 자원 즉 데이타 객체에 대한 충돌 현상을 해결하는 기법이 실시간 트랜잭션 처리 시스템의 스케쥴링 기법에 필요하다. 즉 데이타에 대한 충돌시 중단(blocking) 또는 지연(delay) 현상의 발생은 트랜잭션의 실행을 지연시킴으로써 종료시한내에 종료 여부를 판단하는 예측가능성(predictability) 및 분산 실시간 트랜잭션 처리 시스템의 전체 스케쥴링 과정을 사실상 어렵게 만든다.

실시간 트랜잭션 처리 시스템에서 공유 데이타에 대한 트랜잭션들의 충돌 현상을 해결하기 위하여 2단계 잠금(two-phase locking) 방법과 낙관적(optimistic) 방법이 동시실행 제어 기법에 사용되고 있다. 2단계 잠금 방법에 기초한 동시실행 제어 프로토콜들은[11, 12] 우선순위가 다른 트랜잭션들 간에 중단 현상이 자주 발생함으로써 낙관적 방법에 기초한 동시실행 제어 프로토콜보다 실시간적 성능이 저하되는 것으로 분석된 바 있다[13]. 특히 2단계 잠금 방법에서 발생하는 잦은 중단 현상은 통신 지연시간과 연관되어 분산 환경에서의 실시간 트랜잭션 처리 시스템의 정확한 스케쥴링을 더욱 어렵게 함으로써 예측가능성 및 전체 시스템의 성능을 저하시키게 된다[5, 14]. 현재 2단계 잠금 방법에 기초한 분산 실시간 트랜잭션 처리 시스템의 동시실행 제어 방법인 우선순위 상한(priority ceiling) 프로토콜[15]은 우선순위 전도 현상을 최대한 방지하여 시스템 성능을 향상시켰으나 트랜잭션이 액세스할 데이타들이 집합이 사전에 파악되는 것을 가정하기 때문에 비실용적인 것으로 평가된다[5].

한편 낙관적 방법에 기초한 실시간 동시실행 제어 프로토콜들에서도[16, 17] 검증 단계에서 지연 현상이 발생하며 이는 분산 환경에서 통신 지연 시간과 연관되어 종료시한의 보장을 어렵게 하며 교착상태(deadlock)를 야기할 수 있다. 또한 지연 시간동안 새로운 충돌 현상으로 인하여 취소(abort)되는 트랜잭션의 수가 증가하기 때문에 분산 실시간 트랜잭션 처리 시스템의 성능 향상을 위하여 개선될 필요가 있다. 특히 데이타가 여러 노드에서 복제된(replicated) 분산 데이타베이스 시스템에서는 중앙 시스템과는 달리 검증 단계에 대한 임계영역(critical section) 설정이 어려우며 원자성을 보장하는 과정이 복잡하게 된다[18]. 따라서 분산 실시간 환경에 적절한 검증 단계가 낙관적 동시실행 제어 프로토콜내에 구성되어야 한다. 즉 공유 데이타에 대한 실시간 트랜잭션들의 충돌 현상을 설정된 종료시한 이내에 정확히 제어하는 한편 각 트랜

잭션의 원자적 종료를 보장할 수 있는 검증 단계가 분산 실시간 트랜잭션 처리 시스템의 낙관적 동시실행 제어 기법에 필요하다. 본 논문에서는 데이타가 복제된 분산 환경에서 중단 현상이 방지된 낙관적 동시실행 제어 기법을 분산 실시간 트랜잭션 처리 시스템의 시간 구동형 스케쥴링 기법으로 구성하였다.

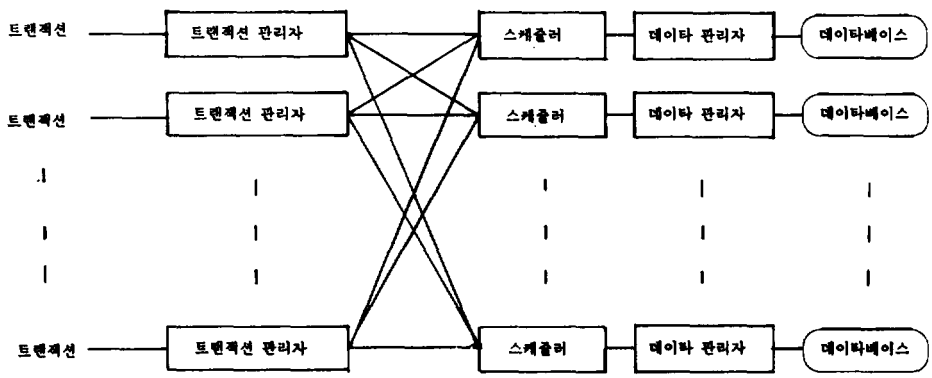
제시된 시간 구동형 스케쥴링 기법에서는 불필요한 중단 현상을 배제함으로써 분산 환경에서 복제된 공유 데이타들을 액세스하는 트랜잭션들의 종료 여부가 설정된 종료시한 이내에 정확히 결정된다. 즉 분산 실시간 트랜잭션 처리 시스템의 예측가능성 및 스케쥴링 가능성을 향상시켰다. 또한 제시된 기법은 트랜잭션들의 액세스 과정을 우선순위에 따라 동기화함으로써 실행 순서에 대한 최종 결과의 정확성 및 전체 데이타의 일관성을 보장한다. 분산 환경의 검증 단계에서 중단 현상으로 인한 교착상태를 방지하며 2단계 종료과정을 포함하여 트랜잭션의 원자적 종료 보장 및 전체 시스템의 신뢰성을 향상시킨다. 제시된 시간 구동형 스케쥴링 기법은 2단계 잠금 방법에 비하여 트랜잭션들의 종료시한이 경과되는 비율이 감소됨으로써 실시간적 성능이 향상된 결과가 실험을 통하여 분석되었다. 복제된 데이타베이스 환경에서 낙관적 동시실행 제어 프로토콜을 포함하는 시간 구동형 스케쥴링 기법은 분산 실시간 트랜잭션 처리 시스템의 시간적 제약성을 최대한 충족시키며 데이타에 대한 일관성을 보장한다. 제시된 시간 구동형 스케쥴링 기법은 향후 물리적 자원을 대상으로 하는 실시간 스케쥴링 프로토콜과 통합되어 효과적인 분산 실시간 트랜잭션 처리 시스템을 구축할 것으로 기대된다.

2. 분산 실시간 트랜잭션 처리 시스템

2.1 분산 모델의 낙관적 기법

분산 실시간 트랜잭션 처리 시스템의 구조적 모델은 그림 1과 같다. 각 노드에서 트랜잭션 관리자, 데이타 관리자, 스케쥴러 등의 소프트웨어 모듈이 구성될 수 있다. 제시된 모델에서 트랜잭션은 시작, 종료, 판독, 기록 등의 연산 수행시 트랜잭션 관리자와 통신한다. 트랜잭션이 자신의 시작 연산을 요청하면 해당 트랜잭션 관리자는 이 트랜잭션을 위한 국지 작업 영역을 생성한다. 단 이 국지 작업 영역의 위치와 구성은 질의어 최적화 기법과 관련되므로 본 논문에서는 구체적인 설명을 생략한다[19]. 이후 트랜잭션 관리자는 스케쥴러와 통신한다. 스케쥴러는 트랜잭션에 대한 종료시한의 보장 여부와 충돌 문제 등을 해결하며 스케쥴러들 간에는 상호 통신이 수행된다. 또한 스케쥴러는 데이타를 직접 조작하는 데이타 관리자와 통신한다. 데이타베이스는 화일, 레코드, 페이지 등의 데이타 객체들로 구성되며 한 데이타 객체는 여러 노드에 복제된 상황을 가정한다.

제시된 모델에서 각 스케쥴러는 동시실행 제어 프로토콜로서 낙관적 기법을 사용한다. 전형적인 낙관적 동시실행 제어 프로토콜에서 각 트랜잭션의 실행



(그림 1) 분산 실시간 트랜잭션 처리 시스템의 구조
(Fig. 1) Structure of distributed real-time transaction processing system

단계는 판독 단계(read phase), 검증단계(validation phase), 기록 단계(write phase) 등의 3단계로 구분된다[20]. 제시된 모델의 각 단계에서 다음과 같은 과정들이 수행된다.

판독 단계:트랜잭션 T의 실행 단계중 대부분을 차지하는 단계이다. 트랜잭션이 임의의 데이타 x에 대한 판독 연산(예, Read(x))과 기록 연산(예, Write(x, new-value))을 요청한 경우 트랜잭션 관리자와 통신이 수행되며 데이타 x에 연관된 계산 작업이 수행된다. Read(x)의 경우 트랜잭션 관리자는 트랜잭션의 국지 작업 영역에서 복제된 데이타 x의 존재 여부를 확인한다. 데이타 x가 존재하면 트랜잭션이 즉시 사용할 수 있으며 그렇지 않은 경우에는 트랜잭션 관리자가 질의어 최적화 과정을 이용하여 복제된 데이타 x가 위치한 노드들 중 한 데이타 관리자를 선정하여 Read(x)를 전송한다. 그리고 트랜잭션의 Write(x, new-value) 연산은 트랜잭션 관리자가 속한 노드내의 작업 영역에서 수행되므로 다른 트랜잭션들은 변경된 x를 참조할 수 없다.

검증 단계:임의의 트랜잭션 T는 검증 단계에서 다른 트랜잭션들과의 충돌 관계가 파악된다. 즉 해당 트랜잭션 관리자는 트랜잭션 T의 판독 연산과 기록 연산이 수행된 데이타들의 집합(RS(T)과 WS(T)로 표시)을 다른 트랜잭션들과 비교함으로써 충돌 관계를 파악한다. 이는 트랜잭션 관리자가 소속된 노드에서 검증이 수행되므로 지역 검증이라 한다. 그리고 트랜잭션 관리자는 동일한 데이타가 여러 노드에 복제되어 있는 경우에도 해당 노드의 스케쥴러들에게 RS(T)와 WS(T)를 전송한다. 이는 원격 검증이라 하며 각 스케쥴러는 다른 트랜잭션들과의 충돌 관계를 파악한 후 트랜잭션의 종료 가능성 여부를 결정하게 된다. 지역 검증과 원격 검증 단계에서 모두 성공하면 해당 트랜잭션 T의 종료가 보장되며 그렇지 않으면 이 단계에서 취소된다.

기록 단계:관련된 스케쥴러들로 부터 트랜잭션 T의 성공적인 종료가 보장되면 트랜잭션 관리자는 이 사실을 해당 스케쥴러들에게 통보한다. 임의의 데이타 x에 대하여 기록 연산이 수행된 경우 x가 복제된 모든 노드들의 스케쥴러들은 WRITE(x, new-value) 연산을 해당 데이타 관리자들에게 전송하게 되며 각 데이타 관리자는 데이타베이스에 새로운 x를 저장하게 된다. 즉 동일 데이타가 여러 노드에 복제된 경우 갱신 작업이 동시에 수행된다. 이 기록 연산은 트랜잭션 관리자의 국지 작업 영역에서 수행되는 것이 아니므로 다른 트랜잭션들은 데이타베이스에 저장된 새로운 x를 참조할 수 있게 된다. 데이타베이스에 대한 모든 기록 연산이 수행된 이후 트랜잭션 관리자는 T를 위한 국지 작업 영역을 제거한다.

2.2 낙관적 기법의 실시간적 특성

낙관적 기법의 검증 단계는 이미 종료된 트랜잭션들을 대상으로 하는 향후(backward) 검증 방법과 아직 종료되지 않은 실행 중인 트랜잭션들과의 충돌 관계를 파악하는 전향(forward) 검증방법으로 구분된다[20]. 실시간 트랜잭션 처리 시스템에서는 2단계 잠금 기법보다 낙관적 기법에 기초한 동시실행 제어 프로토콜의 성능이 향상된 것으로 분석되고 있으며 대부분 전향 검증 방법이 사용되고 있다. 전향 검증 단계에서 실시간 트랜잭션들의 우선순위를 고려함으로써 충돌문제를 해결하는 낙관적 기법들은 다음과 같다[16].

2.2.1 낙관적 취소(abort)

충돌이 일어난 판독 단계의 트랜잭션들이 검증 단계의 트랜잭션보다 우선순위가 모두 높다면 검증 단계의 트랜잭션이 즉시 취소되며 그렇지 않은 경우 판독 단계의 트랜잭션들이 모두 취소되는 두가지 경우가 발생한다.

2.2.2 낙관적 희생(sacrifice)

충돌이 일어난 판독 단계의 트랜잭션들 중 하나라도 검증 단계의 트랜잭션보다 우선순위가 높다면 검증 단계의 트랜잭션이 취소된다. 그렇지 않은 경우에는 판독 단계의 트랜잭션들이 모두 취소되므로 낭비적인 취소가 많이 발생할 수 있다.

2.2.3 낙관적 지연(wait)

충돌이 일어난 판독 단계의 트랜잭션들의 우선순위가 더 높은 경우 검증 단계의 트랜잭션은 우선순위가 높은 트랜잭션들이 먼저 종료될 수 있도록 기회를

부여하기 위해 지연된다. 이 방법은 우선순위가 높은 트랜잭션의 처리를 선호하는 실시간 시스템에 적합한 반면 지연되는 트랜잭션은 결국 우선순위가 더 높은 트랜잭션들에 의해 나중에 취소될 가능성이 많다.

2.2.4 WAIT-50

낙관적 지연 기법의 문제점을 개선하고자 제안된 것이 WAIT-50 기법이다. 충돌이 일어난 트랜잭션들 중 50% 이상이 검증 단계에 있는 트랜잭션보다 우선순위가 높으면 검증 단계의 트랜잭션이 지연된다. 그렇지 않은 경우 즉 우선순위가 높은 비율이 50% 미만이면 검증 단계의 트랜잭션이 자신의 기록 단계로 전환되며 현재 충돌 중인 트랜잭션들을 모두 취소시킨다. WAIT-50 기법은 낙관적 지연 기법에 비하여 실시간 트랜잭션 처리 시스템의 성능을 향상시킨 것으로 분석된 바 있다[16].

낙관적 취소와 회생 방법에서는 충돌이 일어난 트랜잭션들의 낭비적인 취소가 자주 발생함으로써 실시간 트랜잭션 처리 시스템의 성능을 저하시키게 된다. 그리고 낙관적 취소와 회생 방법에 비하여 성능이 향상된 낙관적 지연과 WAIT-50 방법에서는 검증 단계의 트랜잭션이 지연되는 동안 새로운 충돌이 발생할 수 있다. 이러한 현상은 동시에 실행되는 트랜잭션들의 수가 증가하는 시스템일수록 충돌로 인하여 취소되는 트랜잭션들의 수가 역시 증가하게 될 것이므로 결국 실시간 트랜잭션 처리 시스템의 성능을 저하시키는 요인으로 분석된다. 검증 단계에서 지연 현상이 발생하는 낙관적 동시실행 제어 프로토콜들은 실시간 트랜잭션 처리 시스템의 성능 향상 및 분산 환경에의 적용을 위해 여러가지 개선 방안이 필요하다. 분산 실시간 트랜잭션 처리 시스템을 위한 낙관적 기법에서 고려될 사항들은 다음과 같이 기술된다.

첫째, 검증 단계에서의 지연 현상 배제

검증 단계에서 발생하는 지연 현상은 분산 환경에서 통신 지연 시간과 연관되며 실시간 트랜잭션의 종료시한 준수를 위한 예측가능성을 향상시키지 못한다. 따라서 검증 단계에서 소요되는 시간을 단축하는 것이 분산 실시간 환경에 바람직하다. 또한 검증 단계에서 발생하는 지연 현상은 트랜잭션들 간에 교착상태를 유발할 가능성이 높다[18].

둘째, 전역적인 직렬가능성(serializability) 보장

집중화된 시스템처럼 검증 단계를 특정 노드에서만 임계 영역으로 설정하여 처리하게 되면 분산 시스템의 효율성과 신뢰성이 저하된다[21]. 그리고 분산 트랜잭션의 검증을 각 노드에서 임계 영역으로 설정하여 순차적으로 처리하게 되는 경우 데이타에 대한 잠금(lock)이 설정되며 해당 트랜잭션이 검증 단계를 마칠때 잠금이 해제되므로 결국 교착상태를 유발하게 된다[22]. 따라서 각 노드에서 직렬화된 순서가 전체적인 직렬화 순서와 일치되도록 검증 단계에 있는 모든 노드에서 동일한 순서로 검증이 수행될 수 있는 방법이 설정되어야 한다.

분산 트랜잭션 처리 시스템에서는 트랜잭션의 원자성 보장을 위하여 통신망을 통한 2단계 종료 과정이 포함되어야 한다[23]. 2단계 종료 과정에서 발생하는 중단 현상은 분산 시스템의 신뢰성과 동시실행 제어에 직접적인 영향을 미친다. 따라서 분산 실시간 트랜잭션 처리 시스템의 낙관적 동시실행 제어 기법의 검증 단계와 기록 단계에서도 각 노드에서의 검증 결과를 수합하여 한 트랜잭션의 완전한 종료 또는 취소 여부를 결정하는 2단계 종료 과정이 포함되어야만 한다.

3. 분산 실시간 트랜잭션 처리 시스템의 시간 구동형 스케쥴링 기법

3.1 시간 구동형 스케쥴링 기법

3.1.1 종료시한 설정

트랜잭션이 시작될 때 시스템에 의하여 종료시한(deadline)은 다음과 같은 수식에 의하여 설정된다.

$$deadline = interarrival\_time + SF * service\text{-}time \quad (1)$$

$$service\text{-}time = trans\_size * (CPU\_time + I/O\_time) + trans\_size * (1 - RepNo/N) * 2 * Comm\_time \quad (2)$$

트랜잭션들의 평균 도착간격 시간을 의미하는 *interarrival_time* 변수에 의하여 트랜잭션 T의 시작 시간이 Start(T)로 설정된다. 변수 *SF*는 종료시한을 보장하기 위한 시간적 여유 정도를 의미하며 *service-time* 변수는 한 트랜잭션이 데이타들을 액세스하고 처리하는데 필요한 총 소요시간을 의미한다. 데이타베이스에 저장된 특정 데이타를 판독하고자 하는 경우 디

스크에 대한 입출력 처리 시간(*I/O_time*)과 중앙 장치 이용 시간(*CPU_time*)이 필요하게 된다. 그리고 특정 데이타가 *N*개의 노드 중 *RepNo*개의 노드에 복제된 경우 트랜잭션이 액세스할 데이타가 트랜잭션 관리자와 다른 노드에 존재할 확률은 1-*RepNo*/*N*이 되며 통신망을 통한 왕복 지연 시간(2∗*Comm_time*)이 요구된다. 본 논문에서는 모든 데이타가 동일한 갯수의 노드에 복제된 상황을 가정하였다. 특정 데이타에 대한 처리 시간은 한 트랜잭션이 액세스할 데이타의 갯수인 *trans_size* 변수와 곱해져 *service-time*을 결정하게 된다. 종료시한 설정에 대한 변수의 구체적인 값은 4절의 성능 분석에서 기술된다.

3.1.2 스케쥴링 기법

수식 1에서 결정된 트랜잭션 T의 시작 시간과 종료시한은 Start(T)와 Deadline(T)로 각각 표기한다. 트랜잭션의 시간적 제약성은 트랜잭션이 종료되는 시간 즉 종료 타임스탬프 START(T)+C가 Start(T)〈Start(T)+C≤Deadline(T) 일때 충족된다. 이때 C는 종료 과정까지 소요된 시간을 의미한다. 트랜잭션 관리자는 트랜잭션이 시작된 이후 종료시한 이내에 해당 트랜잭션이 종료시한을 준수할 수 있는 지의 여부를 확인하게 된다. 트랜잭션 관리자는 판독 단계의 판독 연산을 데이타가 존재하는 노드의 스케쥴러에게 전송한 후 해당 데이타 관리자가 실제 판독 연산을 수행한 시간을 판독 타임스탬프 Start(T)+R로 설정한다. 이때 R은 판독 연산에 대한 소요 시간이 되며 *I/O_time*, *CPU_time*, 2∗*Comm_time* 등을 포함한다. 액세스할 데이타들을 x, y,…,z으로 가정할때 모든 판독 연산들에 대한 판독 타임스탬프가 Start(T)〈Start(T)+R_i≤Deadline(T)-($Val_{local}+Val_{remote}+2*Comm\_time$) (i=x, y,…,z)이면 트랜잭션 관리자는 트랜잭션의 검증 단계를 시작할 수 있다. T의 검증 단계가 시작된 시간은 Finish(T)로 표기한다.

검증 단계는 트랜잭션 관리자가 속한 노드에 대한 지역 검증과 다른 노드들에 대한 원격 검증으로 구분된다. 따라서 지역 검증 과정에 소요되는 시간(Val_{local})과 원격 검증 과정에 소요되는 시간(Val_{remote}) 그리고 원격 검증에 필요한 통신망에 대한 왕복 지연 시간(2∗*Comm_time*)이 필요하므로 종료시한 Deadline(T)에서 이 시간 만큼의 여유가 있을때 검증 단계가 시작될 수 있다. 실제로 Val_{local}과 Val_{remote}은 중앙처리장치와 주 기억장치만이 이용되므로 통신 지연 시간(*Comm_time*)에 비하여 작은 값이 된다. 본 논문에서는 브로드캐스트 프로토콜을 이용한 원격검증 과정을 가정한다. 원격 검증 과정이 시작된 이후 데이타 x가 존재한 노드의 스케쥴러로 부터 검증 과정의 결과를 통보받은 시간 Start(T)+Val_x를 검증 완료 타임스탬프로 설정한다. 트랜잭션 관리자는 해당 스케쥴러들로부터 통보받은 모든 검증 결과가 긍정적이고 검증 완료 타임스 탬프가 Start(T)+Val_i≤Deadline(T) (i=x, y,…,z)일때 해당 트랜잭션이 종료시한이내에 성공적으로 종료된 것을 보장한다. 그리고 이 시간을 종료 타임스탬프 Start(T)+C로 설정하며 Ts(T)로 표기한다. 이때 트랜잭션 관리자는 관련된 스케쥴러들에게 T의 종료사실과 Ts(T)를 다시 통보한다. 만일 검증 완료 타임스탬프가 종료시한을 넘긴 경우에는 해당 트랜잭션을 즉시 취소하며 취소된 사실을 관련된 스케쥴러들에 통보하게 된다.

동시실행 제어 프로토콜로 2단계 잠금 방법이 사용된다면 판독 연산에 대한 수행 시간은 충돌시 다른 트랜잭션들로 인하여 정확한 예측이 어렵게 된다. 또한 낙관적 기법의 검증 단계에서도 지연 현상이 발생하게 되는 경우 트랜잭션의 종료시한내의 종료 여부를 정확히 예측하기 어렵게 된다. 따라서 본 논문에서는 분산 실시간 트랜잭션 처리 시스템의 예측가능성을 향상시키면서 정확한 스케쥴링을 하고자 판독 연산과 검증 단계에 따른 지연 시간을 방지하는 낙관적 동시실행제어 프로토콜을 사용한다.

3.2 복제된 데이타를 위한 낙관적 동시실행 제어 프로토콜

3.2.1 검증 단계의 충돌 해결 기법

트랜잭션의 판독 단계 이후 검증 단계에서는 다른 트랜잭션들과의 충돌 관계가 파악된다. 각 스케쥴러에서 수행되는 동시실행 제어 프로토콜의 정확성은 직렬가능성으로 증명된다[24]. 예를들면, 서로 다른 트랜잭션 T_1, T_2,…, T_n에 대하여 이들에 의하여 수행된 최종적인 결과가 특정 순서에 따른 T_1→T_2→…→T_n 실행 결과가 된다면 직렬가능성이 증명된다. 두 트랜잭션에 대하여 T_i→T_j 결과는 T_i가 종료된 이후 T_j가 시작된 것과 동일함을 의미한다. 본 논문에서는

우선순위 순서를 직렬가능성 순서로 설정하여 우선순위가 서로 다른 트랜잭션들의 충돌 문제를 해결하였다.

노드 k에서 발생된 트랜잭션 T_i를 $T_{i,k}$로 노드 l에서 발생된 트랜잭션 T_j를 $T_{j,l}$로 각각 표기한다. 데이타가 복제된 분산 실시간 환경에서 트랜잭션 $T_{j,l}$이 검증 단계에 진입한 경우 다른 트랜잭션 $T_{i,k}$와 충돌이 발생할 수 있는 상황은 다음과 같다.

첫째, 순차적 실행(그림 2의 (a) 참조)

트랜잭션 $T_{i,l}$이 노드 j에서 시작되기 이전에 트랜잭션 $T_{j,k}$가 노드 k에서 이미 종료되었기 때문에 이미 두 트랜잭션들은 순차적으로 실행된 것이다. 따라서 이 경우에는 두 트랜잭션들의 충돌관계가 없다.

둘째, 순차적 검증(그림 2의 (b) 참조)

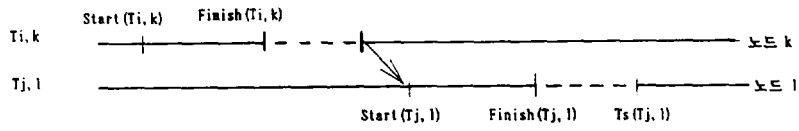
$T_{j,l}$이 시작된 이후이긴 하나 아직 검증 단계에 진입하기 전에 $T_{i,k}$가 이미 종료된 상태이다. 즉 두 트랜잭션들의 판독 단계가 동시에 실행된 것을 의미한다.

세째, 병행적 검증(그림 2의 (c) 참조)

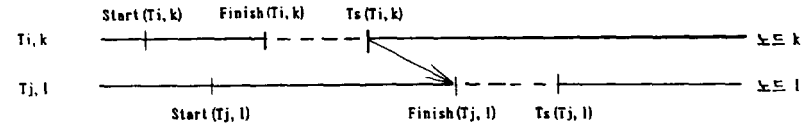
$T_{j,l}$이 검증 단계에 진입하기 이전에 이미 $T_{i,k}$가 검증 단계에 진입한 상황이므로 검증 단계가 병행적으로 실행되는 것을 의미한다.

네째, 동시적 검증(그림 2의 (d) 참조)

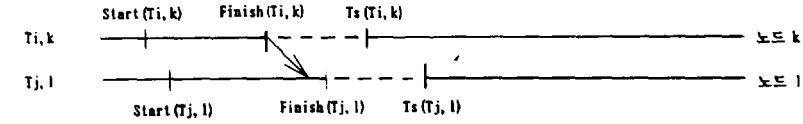
$T_{i,k}$가 검증 단계에 진입한 시점에서 $T_{j,l}$의 검증 단계 진입 사실이 노드 k에 아직 통보되지 않았고 또한 $T_{j,l}$이 검증 단계에 진입한 시점에서도 $T_{i,k}$의 검증 단계 진입 사실이 노드 l에 아직 통보되지 않은 상황이다.



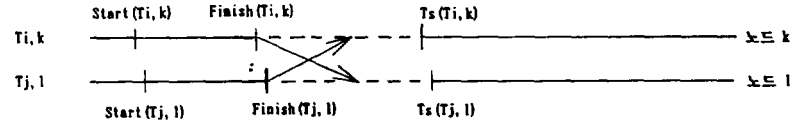
(a) $T_{i,k}$ 에 대한 $T_{j,l}$의 순차적 실행



(b) $T_{i,k}$ 에 대한 $T_{j,l}$의 순차적 검증



(c) $T_{i,k}$ 에 대한 $T_{j,l}$의 병행적 검증



(d) $T_{i,k}$ 에 대한 $T_{j,l}$의 동시적 검증

(그림 2) $T_{j,l}$의 검증 과정
(Fig. 2) Validation phase of $T_{j,l}$

순차적 실행인 경우에는 두 트랜잭션들이 이미 직렬화되어 있기 때문에 충돌 관계를 파악할 필요가 없으나 순차적 검증, 병행적 검증, 동시적 검증에서는 두 트랜잭션들의 직렬화 순서 보장을 위하여 트랜잭션들의 우선순위와 검증 단계에 진입한 순서를 고려한다. 트랜잭션 $T_{j,l}$에 대하여 기록 연산이 수행된 데이타들의 집합은 $WS(T_{j,l})$, 판독 연산이 수행된 데이타들의 집합은 $RS(T_{j,l})$, 우선순위는 $prior(T_{j,l})$로 각각 표기된다. 본 논문에서는 임의의 데이타에 대한 판독 연산이 수행된 이후에만 기록 연산이 수행되는 것을 가정한다. $T_{j,l}$의 검증 단계에 대한 조건들은 다음과 같다.

조건 1. $T_{j,l}$이 검증 단계를 시작하기 이전에 이미 종료되어 순차적 검증 단계에 있는 임의의 $T_{i,k}$에 대한 검증 조건은 $WS(T_{i,k}) \cap RS(T_{j,l}) = \phi$이다(그림 2의 (b) 참조). 이때 두 트랜잭션들의 실행 순서 관계는 $T_{i,k} \rightarrow T_{j,l}$이 되며 $T_{i,k}$가 종료된 이후 $T_{j,l}$이 종료된 것과 동일한 결과가 된다. 그러나 충돌이 발생한 $WS(T_{i,k}) \cap RS(T_{j,l}) \neq \phi$인 경우에는 두 트랜잭션들의 우선순위 비교 결과에 의하여 $T_{j,l}$의 취소 여부가 결정된다. 이에 관한 알고리즘은 그림 3에 기술되어 있다.

과정 1-a. $prior(T_{i,k}) > prior(T_{j,l})$

$T_{j,l}$과 $T_{i,k}$ 간에 충돌이 발생한 임의의 데이타 x에 대하여 $T_{i,k}$가 갱신한 시간(update_ts($T_{i,k}$, x))과 $T_{j,l}$이 판독한 시간(read_ts($T_{j,l}$, x))을 비교한다. update_ts($T_{i,k}$, x) < read_ts($T_{j,l}$, x)인 경우 $T_{i,k}$가 생성한 데이타 x를 $T_{j,l}$이 판독한 것이기 때문에 $T_{i,k} \rightarrow T_{j,l}$ 판독 관계가 성립되며 이는 우선순위 순서($prior(T_{i,k}) > prior(T_{j,l})$)와 일치하게 된다. 즉 우선순위가 높은 $T_{i,k}$가 생성한 데이타들을 $T_{j,l}$이 판독한 경우에 한해 $T_{i,k}$가 우선순위가 낮은 $T_{j,l}$보다 먼저 실행된 순서와 일치하는 $T_{i,k} \rightarrow T_{j,l}$의 실행 순서 관계가 성립된다. 따라서 우선순위가 낮은 $T_{j,l}$은 취소될 필요가 없다. 그러나 $T_{i,k}$가 갱신하기 이전의 데이타를 하나라도 $T_{j,l}$이 판독한 경우에는 $T_{j,l} \rightarrow T_{i,k}$ 실행 순서가 되며 우선순위 순서와 일치하지 않으므로 $T_{j,l}$은 검증 단계에서 즉시 취소된다. 이에 관한 알고리즘은 그림 3에서 기술된다.

```
/* 조건 1 : 순차적 검증 */
for T_{i,k} in Start(T_{j,l}) < Ts(T_{i,k}) < Finish(T_{j,l}) do
 do
    if WS(T_{i,k}) ∩ RS(T_{j,l}) ≠ φ
       if prior(T_{i,k}) > prior(T_{j,l}) then  /* 과정 1-a */
          for x in WS(T_{i,k}) ∩ RS(T_{j,l}) do
             if update_ts(T_{i,k}, x) > read_ts(T_{j,l}, x) then
                restart T_{j,l};
       else    /* prior(T_{i,k}) < prior(T_{j,l}) : 과정 1-b */
          begin
             if WS(T_{i,k}) ∩ WS(T_{j,l}) ≠ φ   /* 단계 1 */
                restart T_{j,l};
             else /* WS(T_{i,k}) ∩ WS(T_{j,l}) = φ */
                for x in WS(T_{i,k}) ∩ RS(T_{j,l}) do  /* 단계 2 */
                   if update_ts(T_{i,k}, x) < read_ts(T_{j,l}, x) then
                      restart T_{j,l};
          end;
 end;
```

(그림 3) 조건 1(순차적 검증)에 대한 알고리즘
(Fig. 3) Algorithm for condition 1(sequential validation)

과정 1-b. $prior(T_{i,k}) < prior(T_{j,l})$

단계 1.

$WS(T_{i,k}) \cap W(T_{j,l}) \neq \phi$의 관계가 먼저 확인된다. 즉 $T_{i,k}$가 갱신한 데이타를 $T_{j,l}$이 다시 갱신하게 되는 경우에는 겹쳐쓰기(overwriting) 현상으로 인하여 데이타의 일관성이 보장되지 않는다. 따라서 기록 연산들 간의 충돌이 발생한 경우에는 우선순위가 높더라도 $T_{j,l}$이 즉시 취소된다.

단계 2.

$WS(T_{i,k}) \cap WS(T_{j,l}) = \phi$이고 $WS(T_{i,k}) \cap RS(T_{j,l}) \neq \phi$ 경우에는 충돌이 일어난 임의의 데이타 x에 대하여 update_ts($T_{i,k}$, x)와 read_ts($T_{j,l}$, x)가 비교된다. update_ts($T_{i,k}$, x) < read_ts($T_{j,l}$, x)인 경우 우선순위가 높은 $T_{j,l}$이 우선순위가 낮은 $T_{i,k}$가 생성한 데이타 x를 판독한 것이기 때문에 $T_{i,k} \rightarrow T_{j,l}$ 판독 관계는 우선순위 순서와 일치하지 않게 된다. 따라서 우선순위가 낮은 $T_{i,k}$가 생성한 데이타를 하나라도 우선순위가 높은 $T_{j,l}$이 판독한 경우에는 검증 단계에서 즉시 취소된다.

그림 3의 알고리즘에서 단계 1과 단계 2가 기술된다. 이 두 단계에서 취소되지 않은 $T_{j,l}$은 우선순위가 낮은 $T_{i,k}$가 갱신하기 이전의 데이타들을 먼저 판독한 경우이므로 $T_{j,l} \rightarrow T_{i,k}$ 실행 순서가 성립되며 이는 우

선순위 순서와 일치하게 된다. 따라서 이 경우에 한해 우선순위가 높은 $T_{j,l}$은 검증 단계에서 취소될 필요가 없다.

조건 2. $T_{j,l}$이 검증 단계에 진입하기 이전에 $T_{i,k}$가 검증 단계에 먼저 진입한 경우의 병행적 검증 단계에서의 검증 조건은 $WS(T_{i,k}) \cap RS(T_{j,l}) = \phi$이다(그림 2의 (c) 참조). 그러나 충돌이 발생한 $WS(T_{i,k}) \cap RS(T_{j,l}) \neq \phi$인 경우에는 우선순위의 비교 결과와 겹쳐쓰기 현상을 고려하여 $T_{j,l}$의 취소 여부가 결정되며 그림 4에서 이에 관한 알고리즘이 기술된다.

```
/* 조건 2 : 병행적 검증 */
for Ti,k in Finish(Ti,k) < Finish(Tj,l)
 do
   if WS(Ti,k) ∩ RS(Tj,l) ≠ φ
     if prior(Ti,k) > prior(Tj,l) then  /* 과정 2-a */
         restart Tj,l;
     else    /* prior(Ti,k) < prior(Tj,l) : 과정 2-b */
        begin
          if WS(Ti,k) ∩ WS(Tj,l) ≠ φ   /* 단계 1 */
              restart Tj,l;
        end;
 end;
```

(그림 4) 조건 2(병행적 검증)에 대한 알고리즘
(Fig. 4) Algorithm for conditon 2(parallel validation)

과정 2-a. $prior(T_{i,k}) > prior(T_{j,l})$

우선순위가 낮은 $T_{j,l}$이 취소되지 않는다면 두 트랜잭션들 간의 실행 순서는 $T_{j,l} \rightarrow T_{i,k}$가 되며 우선순위 순서와 일치하지 않게 된다. 따라서 우선순위에 따른 정확한 직렬화 순서를 위하여 $T_{j,l}$은 즉시 취소된다.

과정 2-b. $prior(T_{i,k}) < prior(T_{j,l})$

$WS(T_{i,k}) \cap WS(T_{j,l}) \neq \phi$ 관계가 먼저 파악된다. 즉 기록 연산들 간의 충돌이 발생한 경우에는 동일 데이타에 대한 겹쳐쓰기 현상이 발생하므로 데이타의 일관성 보장을 위하여 우선순위가 높은 $T_{j,l}$이 즉시 취소된다. 그러나 $WS(T_{i,k}) \cap WS(T_{j,l}) = \phi$인 경우에는 $T_{j,l}$의 판독 연산이 $T_{i,k}$의 기록 연산보다 먼저 수행된 것과 동일한 결과가 되며 $T_{j,l} \rightarrow T_{i,k}$ 실행 순서가 성립된다. 이는 우선순위 순서와 일치하므로 $T_{j,l}$이 취소될 필요가 없다.

이와 같이 분산 실시간 트랜잭션 처리 시스템에서 조건 1 (순차적 검증)과 조건 2(병행적 검증)는 $T_{j,l}$이 발생된 노드 1에서 이미 종료가 되었거나 검증 단계가 시작된 트랜잭션들에 대하여 지역적으로 검증 과정이 수행된다. 반면 조건 3(동시적 검증)은 $T_{j,l}$이 액세스한 데이타가 위치한 노드에 대하여 원격 검증이 수행된다. 이때 복제된 데이타가 존재하는 노드에 대하여도 검증 과정이 수행되며 원격 검증이 수행될 노드들의 집합은 $RepS(T_{j,l})$로 그림 5에서 표기된다. 조건 3에서는 통신망을 통하여 다른 노드들로부터 $T_{j,l}$에 대한 검증 결과(노드 k의 검증 결과는 그림 5의 알고리즘에서 $reply_k$로 표기됨)가 모두 긍정적일때 트랜잭션 $T_{j,l}$의 종료가 인정되며 원자성이 보장된다. 트랜잭션 관리자는 $T_{j,l}$의 성공적인 종료 사실을 복제된 데이타를 포함한 관련 노드의 스케쥴러들에 통보하며 해당 데이타 관리자는 갱신된 새로운 데이타를 영구적으로 기록한다. 그러나 $T_{j,l}$에 대한 검증 결과가 부정적인 경우에는 트랜잭션 관리자가 $T_{j,l}$을 즉시 취소하며 이 사실을 해당 노드의 스케쥴러에 통보한다.

```
valid=TRUE;
for node k in RepS(Tj,l)   /* 조건 3 */
     send validation(RS(Tj,l), WS(Tj,l)) to node k;

for node k in RepS(Tj,l)
   do
       receive replyk from k;
       if replyk is NO then
            valid = FALSE;
   end;

for node k in RepS(Tj,l)
   do
       if valid is TRUE then
            send commit of Tj,l to k;
       else /* valid is FALSE */
            send abort of Tj,l to k;
   end;
```

(그림 5) 원격 검증 단계에서 원자성 보장을 위한 알고리즘
(Fig. 5) Algorithm for atomicity in remote validation phase

조건 3. $T_{i,k}$와 $T_{j,l}$이 서로 다른 노드 k와 l에서 동시에 검증 단계에 진입한 경우(그림 2의 (d) 참조) 우선순위 비교 결과에 따라 취소 여부가 결정된다. $T_{j,l}$이 액

세스한 데이타가 위치하는 노드 k에서 수행되는 $T_{j,l}$에 대한 검증 과정은 다음과 같으며 그림 6의 알고리즘에서 기술된다.

```
/* 조건 3 : 노드 k에서의 동시적 검증 */
for T_j,l in Finish(T_j,l) ~ Finish(T_i,k)
 do
   if prior(T_i,k) > prior(T_j,l)  /* 과정 3-a */
       if WS(T_i,k) ∩ RS(T_j,l) ≠ φ
           restart T_j,l and send NO to node l;
   else   /* prior(T_i,k) < prior(T_j,l)  /* 과정 3-b */
       begin
           if WS(T_i,k) ∩ WS(T_j,l) ≠ φ
               restart T_j,l and send NO to node l;
       end;
   send YES to node l:  /* 과정 3-a와 3-b에서 취소되지 않은 경우 */
 end;
```

(그림 6) 조건 3(동시적 검증)에 대한 알고리즘
(Fig. 6) Algorithm for condition 3(simultaneous Validation)

과정 3-a. $prior(T_{i,k}) > prior(T_{j,l})$

$WS(T_{i,k}) \cap RS(T_{j,l}) \neq \phi$ 경우 우선순위 순서와 일치하지 않는 $T_{j,l} \rightarrow T_{i,k}$ 순서 관계가 성립되므로 우선순위가 낮은 $T_{j,l}$은 취소된다.

과정 3-b. $prior(T_{i,k}) < prior(T_{j,l})$

$WS(T_{i,k}) \cap WS(T_{j,l}) \neq \phi$ 경우 동일 데이타에 대한 겹쳐쓰기 현상이 발생하므로 우선순위가 높은 $T_{j,l}$은 취소된다. 그러나 $WS(T_{i,k}) \cap WS(T_{j,l}) = \phi$이고 $WS(T_{i,k}) \cap RS(T_{j,l}) \neq \phi$ 경우에는 우선순위가 높은 $T_{j,l}$의 판독 연산이 $T_{i,k}$의 기록 연산보다 먼저 수행되어 우선순위 순서와 일치하는 $T_{j,l} \rightarrow T_{i,k}$ 실행 순서가 성립된다. 따라서 우선순위가 높은 $T_{j,l}$은 취소될 필요가 없다.

3.2.2 제시된 기법의 정확성

제시된 낙관적 기법의 검증 단계에서는 지연 현상을 방지함으로써 서로 다른 트랜잭션들이 검증 결과를 기다리게 되는 교착상태를 배제하였다. 예를 들어 조건 3에서 $T_{i,k}$는 노드 k에서 $T_{j,l}$의 검증 결과를 기다리고 $T_{j,l}$은 노드 l에서 $T_{i,k}$의 검증 결과를 서로 기다린다면 교착상태가 발생하게 된다. 따라서 적절한 타임아웃을 설정함으로써 교착상태를 예방할 수 있는 별도의 기법이 마련되어야 한다. 또한 두 트랜잭션들의 실행 순서도 노드 k에서는 $T_{j,l} \rightarrow T_{i,k}$가 되고 노드 l에서는 $T_{j,l} \rightarrow T_{i,k}$가 될 수 있다. 즉 두 트랜잭션들의 실행 순서에서 선후 관계가 불분명하게 되는 주기(cycle)가 형성된다. 트랜잭션들의 교착상태 방지와 최종 실행 결과의 정확성을 보장하기 위해서는 낙관적 기법의 검증 단계에서 불필요한 지연 현상을 방지하는 것이 바람직하다. 시간 구동형 스케쥴링 기법을 위한 낙관적 동시실행 제어 프로토콜에서는 검증 단계에서 서로 다른 트랜잭션들 간의 우선순위를 비교함으로써 지연 현상없이 트랜잭션의 취소 여부를 결정하게 된다.

서로 다른 트랜잭션 $T_{i,k}$와 $T_{j,l}$간의 충돌로 인한 실행 순서가 $T_{j,l} \rightarrow T_{i,k}$ 관계인 경우 우선순위 순서($prior(T_{j,l}) > prior(T_{i,k})$)와 일치하면 두 트랜잭션들은 모두 종료될 수 있다. 그러나 우선순위 순서와 일치하지 않는 경우에는 검증 단계의 조건에 따라 둘 중 한 트랜잭션이 반드시 취소된다. $T_{j,l}$의 검증 단계에서 $T_{i,k}$와 충돌이 발생한 경우($WS(T_{i,k}) \cap RS(T_{j,l}) \neq \phi$) 우선순위 비교 결과에 따라 다음과 같이 해결된다.

(i) $prior(T_{i,k}) > prior(T_{j,l})$

$T_{j,l}$의 판독 연산이 $T_{i,k}$의 기록 연산보다 먼저 수행된 것이므로 우선순위 순서와 일치하지 않는 $T_{j,l} \rightarrow T_{i,k}$ 실행 순서가 성립된다. 따라서 $T_{j,l}$은 즉시 취소된다.

(ii) $prior(T_{i,k}) < prior(T_{j,l})$

이때 $WS(T_{i,k}) \cap WS(T_{j,l}) \neq \phi$인 경우 우선순위에 상관없이 데이타의 일관성 보장을 위하여 $T_{j,l}$은 즉시 취소된다. 그러나 $WS(T_{i,k}) \cap WS(T_{j,l}) = \phi$인 경우 $T_{j,l}$의 판독 연산과 $T_{i,k}$의 기록 연산만이 충돌한 경우에는 $T_{j,l} \rightarrow T_{i,k}$ 실행 순서가 성립되며 이는 우선순위 순서와 일치하므로 $T_{j,l}$과 $T_{i,k}$는 모두 종료될 수 있다.

결과적으로 검증 단계에서 충돌이 발생한 트랜잭션들 T_1, T_2,…, T_{n-1}, T_n에 대하여 주기가 형성되는 $T_1 \rightarrow T_2 \rightarrow \cdots T_{n-1} \rightarrow T_n \rightarrow T_1$ 실행 순서는 우선순위 순서 관계가 $prior(T_1) > prior(T_2) > \cdots > prior(T_{n-1}) > prior(T_n) > prior(T_1)$일때 성립될 수 있다. 그러나 본 논문의 제시된 낙관적 기법에서는 이러한 우선순위 순서 관계가 발생하지 않기 때문에 동시실행된 트랜잭션들이 직렬적으로 실행된 것과 동일한 결과를 생성함으로써 우선순위 순서에 따른 직렬가능성이 보장된다[24]. 즉 트랜잭션들의 최종 결과에 대한 정확성이 보장됨으로써 전체 데이타에 대한 일관성이 유

지된다.

3.2.3 분산 실시간 환경에서의 신뢰성

데이타가 복제된 분산 실시간 트랜잭션 처리 시스템에서는 데이타에 대한 유용성(availability)을 향상시킬 수 있는 장점이 있다. 그러나 분산 시스템에 이상(failure) 현상이 발생한 경우 데이타 액세스에 대한 계속적인 서비스가 어렵게 되며 실시간 스케쥴링 과정에서도 종료시한을 준수할 수 있도록 이에 대한 방안이 강구되어야 한다. 제시된 기법의 원격 검증 과정에서도 시스템의 이상 현상으로 인하여 관련된 모든 노드들로부터 응답이 수합되지 못한 경우 트랜잭션들의 동시실행 제어와 원자성에 영향을 미치게 된다. 따라서 본 논문에서는 원격 검증 과정을 시작한 트랜잭션 관리자는 종료시한내에 적절한 타임아웃(통신망에 대한 왕복 지연 시간이 포함됨)을 설정한다. 즉 타임아웃내에 특정 노드의 응답 신호가 도착하지 않으면 특정 노드에 이상 현상이 발생한 것으로 간주한다. 데이타가 복제된 노드들 중 과반수 이상으로부터 긍정적인 응답 신호가 수합되면 해당 트랜잭션 관리자는 검증 단계의 트랜잭션이 성공적으로 종료된 것을 가정한다. 즉 다수결 원칙(majority rule)에 따라 이상 현상이 발생하지 않은 노드들에게 트랜잭션의 종료 사실을 통보하며 데이타의 새로운 갱신이 각 노드에서 수행되도록 한다. 결과적으로 시스템의 이상 현상 발생시에도 한 트랜잭션의 종료를 종료시한내에 보장하며 복제된 데이타들에 대한 유용성을 향상시킬 수 있다.

이상 현상이 발생했던 노드들이 나중에 복구되면 복제된 데이타들이 존재하는 노드들에 대하여 데이타의 갱신 여부를 문의하게 된다. 갱신 여부를 문의받은 노드들은 갱신된 데이타를 복구된 노드들에 전송함으로써 데이타 전체에 대한 일관성과 유용성을 유지하게 된다. 본 논문에서는 분산 실시간 트랜잭션 처리 시스템의 복구 시간과 구체적인 과정들에 대한 언급을 생략한다.

4. 성능 분석 및 평가

4.1 시뮬레이션 환경

분산 실시간 트랜잭션 처리 시스템에 대한 시뮬레이션은 호스트가 10(=N)개로 구성된 분산 시스템 환경에서 각 호스트마다 100개의 데이타 객체를 할당하여 총 1,000개의 데이타(트랜잭션이 실제 액세스 단위는 페이지로 고려함)들로 데이타베이스를 구성하였다. 그리고 각 데이타는 3(=RepNo)개의 노드에 복제된 상황을 가정하였기 때문에 각 노드의 데이타베이스는 실질적으로 300개의 데이타들로 구성된다. 종료시한 설정(수식 1, 2 참조)과 시뮬레이션을 위한 중요변수들은 표 1과 같다.

〈표 1〉 시뮬레이션에서 사용된 주요 변수
〈Table 1〉 Primary variables used in the simulation

변수	값
interarrival_time	20−90 msec
SF	1.33−4.0(2.0으로 고정)
trans_size	8−24
CPU_time	5−15 msec
I/O_time	15−25 msec
Comm_time	1 msec(완전 연결된 경우) 1−9(완전 연결이 아닌 경우)
P_W	0.1−0.9

interarrival_time 변수는 20 msec에서 90 msec까지 10 msec씩 증가시키며 실험하였다. 평균 도착간격 시간이 작을수록 초(sec)당 시스템에 유입되는 전체 트랜잭션들의 수가 증가된다. 변수 *SF*는 1.33에서 4.0까지 가변적으로 설정할 수 있으나 본 논문에서는 2.0으로 고정하였다. *trans_size* 변수는 8과 24 사이의 임의의 값이 가변적으로 설정되도록 균등한 분포(uniform distribution)를 고려하였다. 데이타베이스에 저장된 특정 데이타를 판독하고자 하는 경우 디스크에 대한 입출력 처리 시간(*I/O_time*)과 중앙 처리 장치 이용 시간(*CPU_time*)이 필요하게 된다. *CPU_time*과 *I/O_time*의 평균값은 각각 10 msec와 20 msec로 설정하였으며 삼각형 분포(triangular distribution)를 고려하여 최대값과 최소값들은 평균값에서 5 msec를 가감하여 설정하였다. *Comm_time* 변수는 노드들이 완전 연결된 경우 통신 지연 시간이 거의 없으므로 값이 1 msec로 설정되며 통신 지연 시간이 실제 존재하는 통신망의 경우에 평균값을 5 msec로 설정하여 두가지 경우에 대한 시뮬레이션을 각각 수행하였다. 그리고

검증 단계에 대한 소요 시간 Val_{local}(지역 검증)과 Val_{remote}(원격 검증)는 각각 1 msec로 설정하였다.

트랜잭션이 특정 데이타를 판독한 후 기록할 확률 P_w는 0.1에서 0.9까지 0.1씩 증가시키며 실험하였다. 기록 확률 P_w가 높을수록 실제 트랜잭션들 간의 충돌 비율이 높아지게 된다. 기록 확률 P_w가 0 혹은 1.0이 되는 비현실적인 경우를 제외하였으며 모든 데이타에 대한 기록 확률은 동일한 것으로 가정하였다.

본 논문에서는 물리적 자원(중앙처리 장치, 입출력 장치, 통신망)에 대한 별도의 실시간적 스케쥴링 방법을 고려하지 않았다. 그리고 개방형 큐잉(open queuing) 모델을 이용하여 각 호스트는 8개의 중앙 처리 장치와 16개의 하드디스크를 위한 입출력 장치로 구성된 다중프로세서를 가정하였다[16].

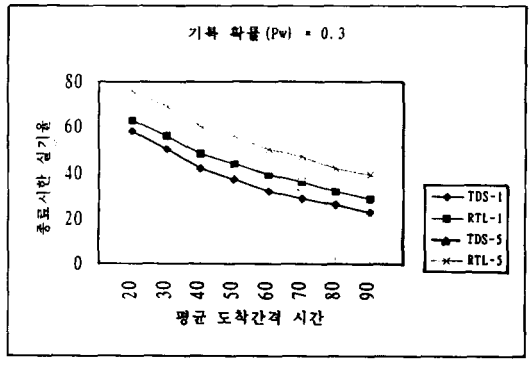
4.2 성능 평가

우선순위 설정 방법은 최초 종료시한(earliest deadline) 방법에 따라 종료시한이 이를수록 트랜잭션의 우선순위가 상대적으로 높아지는 방법을 적용하였다[13]. 우선순위 설정 방법은 제시된 낙관적 동시실행 제어 프로토콜은 물론 비교 대상인 RTL(real-time locking) 프로토콜[12]에 동일하게 적용된다. 2단계 잠금 방법에 기초한 RTL 프로토콜에서는 우선순위가 높은 트랜잭션의 판독 연산이 항상 지연된다. 그리고 한 트랜잭션이 검증 단계와 유사한 대기(wait) 단계에 도달할때 이보다 우선순위가 더 높은 트랜잭션이 판독 혹은 대기 단계에서 존재하는 경우 먼저 종료될 때까지 역시 지연되는 특성이 있다. 두가지 동시실행 제어 프로토콜들을 분산 실시간 트랜잭션 처리 시스템 모델에 적용하여 시간 구동형 스케쥴링 기법의 시뮬레이션 결과를 분석하였다.

본 논문에서는 종료시한이 경과되면 트랜잭션의 의미가 급격히 상실되는 준경성 실시간(firm real-time) 시스템 환경을 고려하였다[16]. 따라서 종료시한내에 성공적으로 종료되지 못한 트랜잭션들의 백분율을 실기율(miss ratio)로 정의하며 가장 중요한 성능 평가 요소로 고려하였다. 실기율이 낮을수록 분산 실시간 트랜잭션 처리 시스템의 성능이 향상되는 것을 의미한다.

4.2.1 평균 도착간격 시간에 따른 실기율(miss ratio)

데이타에 대한 기록 확률(P_w)을 0.3으로 고정한 상태에서 평균 도착간격 시간을 20 msec에서 90 msec까지 10 msec씩 증가시키며 실험한 결과는 그림 7에서 상호 비교되고 있다. 제시된 시간 구동형 스케쥴링 기법은 그림 7의 그래프에서 TDS(Time-Driven Scheduling)로 표기되며 통신 지연 시간 *Comm_time*이 1 msec와 평균값 5 msec인 경우에 대한 결과들이(TDS-1과 TDS-5로 표기) RTL 기법과의 시뮬레이션 결과들(RTL-1과 RTL-5로 표기)과 비교되고 있다. 두가지 기법에서 평균 도착간격 시간이 커질수록 시스템에서 수행되는 트랜잭션들의 수가 작아지므로 데이타의 충돌 빈도수가 함께 지연되거나 취소되는 트랜잭션들의 수가 점차 감소되는 결과가 나타났다. 따라서 평균 도착간격 시간이 커질수록 두가지 기법이 실기율은 모두 낮아지며 동일한 평균도착간격 시간에 대해서는 RTL-1과 RTL-5 기법의 실기율이 TDS-1 과 TDS-5 기법의 실기율보다 각각 더 높은 결과가 나타났다.

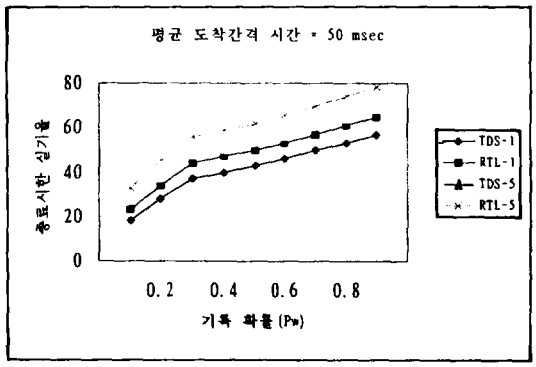


(그림 7) 평균도착간격 시간에 대한 종료시한 실기율
(Fig. 7) Percentage of missed deadlines for mean interarrival time

이때 RTL 기법에서 데이타에 대한 충돌시 지연 시간으로 인하여 종료시한을 준수하지 못하는 트랜잭션들의 수가 TDS 기법보다 증가하기 때문인 것으로 분석되었다. 그리고 이러한 지연 시간으로 인한 현상은 실제 통신 지연 시간을 고려한 시뮬레이션에서도 RTL-5의 실기율이 TDS-5 기법의 실기율보다 높게 나타나는 결과가 확인되었다.

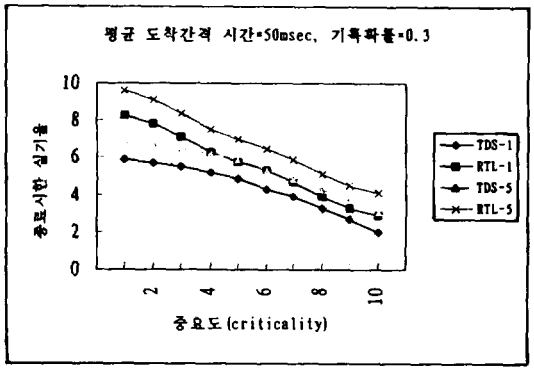
4.2.2 기록 확률에 따른 실기율(miss ratio)

시스템의 적정한 부하 상태를 가정하여 평균 도착 간격 시간을 50 msec로 고정한 상태에서 기록 확률(P_w)만 0.1부터 0.9까지 0.1씩 증가시키며 실험한 결과들이 그림 8에서 상호 비교되고 있다. 기록 확률의 증가는 빈번한 충돌 현상을 야기함으로써 트랜잭션들의 실기율을 증가시키게 된다. 제시된 TDS 기법의 실기율은 RTL 기법의 실기율보다 항상 낮게 나타났다. 즉 통신 지연 시간을 거의 고려하지 않은 실험에서 RTS-1의 실기율이 TDS-1의 실기율보다 높은 결과가 나타났으며 또한 통신 지연 시간을 고려한 경우에서도 RTS-5의 실기율이 TDS-5의 실기율보다 더 높은 결과가 나타났다. 통신 지연 시간이 고려되는 분산 처리 환경에서 제시된 TDS 기법은 RTL 기법보다 실시간적 성능을 향상시킬 수 있다.



(그림 8) 기록 확률(P_W)에 대한 종료시한 실기율
(Fig. 8) Percentage of missed deadlines for write probability(P_W)

4.2.3 중요도에 따른 실기율



(그림 9) 중요도에 대한 종료시한 실기율
(Fig. 9) Percentage of missed deadlines for criticality level

본 논문에서는 종료시한내에 종료된 트랜잭션들의 중요도를 분석하였다. 동일한 종료시한 충족율에 대하여 1부터 5까지의 작은 중요도를 갖는 트랜잭션들이 종료되는 것보다는 6부터 10까지 중요도가 큰 트랜잭션들이 종료되는 것이 더욱 바람직하다. 이 실험에서는 *trans-size*가 16이며 평균 도착간격 시간은 50msec 그리고 기록 확률은 0.3으로 각각 설정되었다. 그리고 각 트랜잭션의 중요도는 종료시한과는 무관하게 설정된다. 그림 9의 그래프는 설정된 중요도를 우선순위 수준으로 고려한 실험 결과이다. 두 기법에 대한 종료된 트랜잭션들에 대한 중요도 분석이 통신 지연 시간을 고려한 경우와 고려하지 않은 경우에 대하여 나타나고 있다. 각 기법에서 중요도가 클수록 실기율이 낮아지는 것을 알 수 있다. 동일한 중요도에서 RTL 기법의 실기율(RTS-1과 RTS-5)은 TDS 기법의 실기율(TDS-1과 TDS-5)보다 높게 나타났으며 두 기법 모두 우선순위가 높은 트랜잭션들이 우선순위가 낮은 트랜잭션들보다 종료시한 내에 종료될 수 있는 기회가 많은 것으로 분석되었다.

5. 결 론

본 논문에서는 낙관적 동시실행 제어 프로토콜을 데이타가 복제된 분산 실시간 트랜잭션 처리시스템에 적용하는 시간 구동형 스케쥴링 기법을 제시하였다. 데이타 충돌시 2단계 잠금 방법에서 발생하는 중단 현상과 기존의 낙관적 방법에서 발생하는 검증 단계에서의 지연 현상은 트랜잭션의 종료시한 준수 및 실시간 스케쥴링의 예측가능성을 어렵게 하는 요인이 된다. 따라서 분산환경에서 발생하는 불필요한 중단 현상과 지연 현상을 배제함으로써 트랜잭션의 종료시한을 정확히 준수할 수 있는 낙관적 동시실행 제어 프로토콜이 시간 구동형 스케쥴링 기법에 적용되었다. 제시된 낙관적 기법에서는 데이타가 복제된 환경에서 발생하는 서로다른 우선순위의 트랜잭션들을 정확히 제어하며 가능하면 우선순위가 높은 트랜잭션의 취소를 방지하고 있다. 검증 단계는 지역 검증 단계와 원격 검증 단계로 구성됨으로써 데이타가 복제된 환경에서도 동일 데이타에 대한 기록 연산이 동기화되며 실시간 트랜잭션의 원자성이 보장된다. 낙관적 기법의 정확성은 트랜잭션들의 최종 실행 결과

가 우선순위 순서에 따른 직렬가능성을 보장함으로써 증명되며 전체 데이타에 대한 일관성도 유지된다. 낙관적 동시실행 제어 프로토콜을 적용하는 본 논문의 시간 구동형 스케쥴링 기법은 실시간 트랜잭션들에 대한 충돌 과정, 원자성, 종료시한 준수, 데이타의 일관성 등을 효율적으로 보장함으로써 전체 시스템의 성능을 향상시킨다.

기록 확률과 평균 도착간격시간을 토대로한 전반적인 실험 영역에서 종료시한을 준수하지 못하는 비율 즉 실기율을 분산 실시간 트랜잭션 처리 시스템의 중요한 성능 평가 요소로 고려하였다. 제시된 시간 구동형 스케쥴링 기법의 실기율은 2단계 잠금에 기초한 방법의 실기율보다 낮은 것으로 분석되었다. 즉 2단계 잠금 방법에서 트랜잭션의 수행시 발생되는 지연 현상은 실시간 스케쥴링의 예측가능성을 어렵게 만들 뿐아니라 분산 환경에서 실시간 트랜잭션 처리 시스템의 성능을 저하시키게 된다. 제시된 시간 구동형 스케쥴링 기법은 데이타에 대한 일관성을 보장하는 한편 동시에 실행되는 트랜잭션들의 수가 증가되어도 종료시한을 보장할 수 있는 비율을 증가시킬 수 있다 결과적으로 제시된 기법은 통신망을 기반으로 하는 분산 실시간 트랜잭션 처리시스템의 성능을 향상 시키게 되며 다른 구성요소들에 대한 실시간 스케쥴링 기법과 통합되어 보다 실질적인 시스템을 구성할 것으로 기대된다.

참 고 문 헌

[1] B. Kao and H. Garcia-Molina, "An overview of real-time database systems," *Proc. of the NATO ASI on Real Time Computing, vol. 127*, pp. 261-282, Oct. 1992.

[2] L. Sha, R. Rajkumar and J. P. Lehoczky, "Concurrency control for distributed real-time databases," *ACM SIGMOD Record, vol. 17, no. 1*, pp. 82-88, Mar. 1988.

[3] S. H. Son, "Real-time database systems: issues and approaches," *ACM SIGMOD Record, vol. 17, no. 1*, Mar. 1988.

[4] R. Abbott and H. Garcia-Molina, "Scheduling real-time transactions: a performance evaluation," *Proc. 14th VLDB Conf.*, pp 1-12, 1988.

[5] S. H. Son and J. Lee, "Scheduling real-time transactions in distributed database systems," *7th IEEE Workshop on Real-Time Operating Systems and Software*, pp. 39-43, 1990.

[6] J. Stankovic, "Real-time computing systems: the next generation," *Tutorial: Hard Real-Time Syst.*, pp. 14-37, May 1988.

[7] S. R. Thuel and J. P. Lehoczky, "Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing," *Proc. 15th Real-Time Syst. Symp.*, pp. 22-33, Dec. 1994.

[8] M. D. Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real-time systems," *Proc. 15th Real-Time Syst. Symp.*, pp. 216-227, Dec. 1994.

[9] K. W. Tindel, H. Hanson and A. J. Wellings, "Analysing real-time communications: controller area network(CAN)," *Proc. 15th Real-Time Syst. Symp.*, pp. 259-263, Dec. 1994.

[10] M. J. Carey, R. Jauhari and M. Livny, "Priority in DBMS resource scheduling," *Proc. 15th VLDB Conf.*, pp. 397-410, 1989.

[11] J. Huang, et al., "On using priority inheritance in real-time databases," *Proc. 12th Real-Time Syst. Symp.*, IEEE, pp. 210-221, 1991.

[12] T. Lin and S. H. Son, "Concurrency control in real-time databases by dynamic adjustment of serialization order," *Proc. 11th Real-Time Syst. Symp.*, pp. 104-112, Dec. 1990.

[13] J. R. Harista, M. Livny and M. J. Carey, "Earliest deadline scheduling for real-time database systems," *Proc. 12th Real-Time Syst. Symp.*, pp. 232-242, Dec. 1991.

[14] M. Singahl, "Issues and approaches to design of real-time database systems," *ACM SIGMOD Record, Vol. 17, No. 1*, pp. 19-33, Mar. 1988.

[15] S. H. Son and C. Chang, "Performance evaluation of real-time locking protocols using a distributed software prototyping environment," *Proc. 10th Int'l. Conf. on Dist. Comp. Syst.*, pp. 124-131,

Jun. 1990.

[16] J. R. Harista, et al., "Dynamic real-time optimistic concurrency control," *Proc. 11th Real-Time Syst. Symp.*, pp. 94-103, 1990.

[17] J. Huang, et al., "Experimental evaluation of real-time optimistic concurrency control schemes," *Proc. 17th VLDB Conference*, pp. 35-46, 1991.

[18] X. Jia, K, Shimizu and M. Maekawa, "Atomic accesses to a single replicated file in distributed systems," *Proc. 7th Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 693-699, 1994.

[19] N. Goodman, et al., "Query processing in SDD-1," Tecnical Report, 79-06, Computer, Corp. of Am., Oct 1979.

[20] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Trans. Database Syst., vol. 6, no. 2*, pp. 213-226, Jun. 1981.

[21] S. J. Mullender and A. S. Tanenbaum, "A distributed file service based on optimistic concurrency control," *Proc. of the 10th Symp. on Operating Systems Principles*, 1985.

[22] A. Thomasian and E. Rahm, "A new distributed optimistic concurrency control method and a comparison of its performance with two-phase locking," *Proc. of Int'l Conf. on Distributed Computing Systems*, pp. 294-301, 1990.

[23] P. A. Bernstein and N. Goodman, "Timestamp-based algorithms for concurrency control in distributed database systems," *Proc. 6th VLDB Conf.*, pp. 285-300, 1980.

[24] M. H. Graham, "How to get serializability for real-time transactions without having to pay for it," *Proc. 14th Real-Time Symp.*, pp. 56-64, 1993.

김 진 환

1986년 서울대학교 공과대학 컴퓨터공학과 졸업(학사)
1988년 서울대학교 대학원 컴퓨터공학과 졸업(석사)
1994년 서울대학교 대학원 컴퓨터공학과 졸업(박사)
1994년~현재 서울대학교 컴퓨터신기술공동연구소 특별연구원
1995년~현재 한성대학교 정보전산학부 전임강사
관심분야:분산 및 실시간 시스템, 트랜잭션 처리 시스템