□특 집□

실시간 분산 데이타베이스 시스템의 설계 고려사항

건국대학교 전자계산학과 한 기 준*

목 차

Ⅰ. 서 론

실시간 응용들은 빠른 응답 시간과 파국적인 고장에도 불구한 계속적인 연산과 같은 엄격한 성능조건을 요구하기 때문에, 분산 시스템은 실시간 응용(예를들면, 군사 훈련과 무기 시스템)에서 아주 중요한 위치를 차지하고 있다[7,11,25]. 대부분의 실시간 응용들은 원래 분산되어 있기 때문에 자연스럽게 분산 시스템으로 발전하여 왔다. 실시간 분산 시스템은 통신하며 협동하는 컴퓨터들의 집합으로 구성되는데, 이들은 공통의 목적을 위해 작업을 수행한다.

항공 우주, 방어 시스템과 같은 다양한 응용에서 실시간 분산 시스템의 중요도는 점차 증가하고 있다. 실시간 분산 시스템의 주된 특징은 다음과 같다[19]. 첫째, 시간이 관리되어야 하는 가장 중요한 자원이다. 타스크들이 제한시간 이전에 완료될 수 있도록 할당되고 스케쥴되어야 한다. 그러므로 메시지들이 상호작용하는 실시간 타스크들간에 적시에 보내지고 받아져야 한다. 둘째, 실시간 시스템의 고장은 경제적인 재난 또는 생명의 손실을 야기할 수 있기 때문에 신뢰성이 매우 중요하다. 셋째, 한 컴퓨터가 운영되는 환경은 임의의 실시간 시스템의 활동적인(active) 한 구성요소이다. 즉, 비행기가 자동 항법을 따르는 경우에서 비행기 자체를 고려하지 않고 내장된 컴퓨터를 고려하는 것은 의미가 없다. 일반적으로는 실시간 분산 시스템은 고도의 성능과 고도의 신뢰성을 제공할 수 있다.

실시간 데이타베이스는 항공기 추적과 제조 설비의 제어와 같은 광범위한 응용에서 점차 중요해지고 있다. 실시간 데이타베이스 상황에서의 동시성 제어 프로토콜들은 데이타베이스의 일관성 제한조건을 유지해야 할 뿐 아니라 데이타베이스를 액세스하는 트랜잭션들의 시간 제한조건들도 만족시켜야 한다[7,19,24]. 즉, 추적과 같은 실시간 응용에서는 일관성 역시 중요하나, 데이타 객체들의 값들이 가능하면 적시에 맞게 하는 것도 중요하다. 추적되는 객체가 빠르게 움직인다면, 매우 절박한 시간 제한조건이 객체의 위치를 변경하는 트랜잭션들에 부여될 것이다. 이러한 시간 제한조건들을 충족시키지 못하면 데이타가 시간에 뒤지기 때문에 추적 과정은 실패하게 된다.

데이타가 중복된 분산 데이타베이스 시스템은 향상된 신뢰성, 향상된 응답성, 디렉토리 비관리, 부담 분배의 용이성 등과 같은 바람직한 특성을 갖고 있기

* 종신회원

때문에 실시간 응용에서 아주 중요하다. 한 컴퓨터가 붕괴(crash)되거나 또는 네트워크가 분할되더라도 데이타 객체들을 계속 액세스할 수 있고, 또한 이러한 고장하에서도 트랜잭션 처리가 중단되지 않기 때문에 향상된 신뢰성을 얻을 수 있다. 향상된 신뢰성 때문에, 중복된 분산 데이타베이스 시스템은 컴퓨터 붕괴 또는 네트워크의 분할이 불가피하고 이러한 고장하에서도 계속적인 연산이 매우 중요한 실시간 환경(예를들면 무기 시스템)에서 매우 바람직하다[22]. 또는 중복된 분산 데이타베이스 시스템은 모든 데이타 객체들이 모든 장소에서 지역적으로 사용가능하기 때문에 질의에 대해서 매우 빠른 응답을 제공할 수 있다. 왜냐하면 긴 통신 지연이 필요한 원격 데이타의 액세스가 필요가 없기 때문이다.

그러나, 데이타의 중복은 또한 댓가가 필요하다. 즉, 부가적인 기억장소 비용 뿐만 아니라 다중 복사 갱신의 동기화는 복잡하고 많은 통신을 요구하므로 중복 데이타베이스에 대한 갱신은 비용이 비싸게 된다. 이러한 제경비에도 불구하고 중복된 분산 데이타베이스 시스템은 여러가지 바람직한 특성을 갖고 있으므로 실시간 분산 시스템에 매우 적합하다[12,23,26].

본 논문에서는 실시간 분산 데이타베이스 시스템의 설계에 있어서 고려해야 할 사항들에 대해서 소개하고자 한다. 즉, 실시간 분산 데이타베이스 시스템의 설계에 필요한 일반 요구조건에 대해서 언급하고, 고장 허용과 신뢰성 고려사항인 장소 회복과 네트워크 분할에 대해서 설명하겠다. 또한, 동시성 제어 고려사항에 대해서 언급하고, 서너가지 동시성 제어 프로토콜들에 대해서 자세하게 소개하겠다.

II. 시스템의 설계 고려사항

2.1 일반 요구조건

실시간 분산 데이타베이스 시스템은 협동하는 자치권있는 단일-장소 실시간 데이타베이스 시스템들의 집합이기 때문에 실시간 분산 데이타베이스 시스템의 설계에 있어서 주된 문제점은 어떻게 서너개의 자치권이 있는 데이타베이스 시스템들을 공동으로 수행하여 상위 수준의 성능을 얻느냐 하는 것이다[22]. 즉, 각각의 컴퓨터에 있는 데이타베이스가 실시간 응답을 제공할 수 있을 때, 문제는 동시성 제어, 질의 최적화, 데드록 발견/처리 방법 등을 어떻게 설계하여 바람직한 수준의 성능을 실시간 분산 데이타베이스 시스템에서 얻느냐는 것이다. 한 트랜잭션의 수행이 통신 네트워크 상에서 정보 교환과 제어를 위해 메세지 교환을 요구할 때, 디스크 속도가 단일-장소 데이타베이스 시스템의 성능을 제한하듯이 통신 네트워크 속도는 실시간 분산 데이타베이스 시스템의 성능을 제한할 것이다. 그러므로, 실시간 분산 데이타베이스 시스템의 설계에서의 주된 문제점은 컴퓨터간의 통신 요구도를 가능한 줄이는데 있다.

실시간 응용을 위해 분산 데이타베이스 시스템을 사용하기 위해서는 기존의 분산 데이타베이스 시스템의 주된 문제점들의 해결이 필요하다. 분산 데이타베이스 시스템은 데이타베이스 갱신에 대해 병행 수행되는 질의에 대하여 최대 응답시간을 보장해야 한다. 그러므로 질의에 적절히 응답하기 위해서는 실시간에 원격 데이타를 액세스하는 것은 일반적으로 불가능하다. 즉, 데이타가 writer에 의해서 동시에 갱신될 때 실시간 reader가 writer들의 commit를 기다리는 것은 불가능하다. 그러므로, 고전적인 2-단계 록킹은 실시간 데이타베이스 동시성 제어를 위해서는 적합하지 못하다[11]. 또 하나의 문제는 시스템이 매우 빠른 속도로 관련된 데이타를 기록하기 위하여 신속한 기록 갱신에 대한 요구이다.

실시간 분산 데이타베이스 시스템은 기존의 분산 데이타베이스 시스템과 비교하여 구별되는 특징들을 갖고 있다. 즉, 트랜잭션들과 관련된 시간 제한조건을 갖고 있다. 그러므로, 실시간 분산 데이타베이스 시스템에서의 트랜잭션들은 데이타 일관성과 시간 제한조건 모두를 고려하면서 스케쥴되어야 한다[16]. 또한, 실시간 분산 데이타베이스 시스템에서는 운영 환경에서의 변화를 고려하고 중요한 트랜잭션의 완료도 보장해야 한다.

분산 데이타베이스 시스템에서 데이타의 중복은 데이타의 가용성의 향상을 위한 주된 조건이다. 중복된 데이타가 여러 장소에 저장되어 있으면, 장소 고장에 기인하여 서너개의 복사(copy)들이 사용 불가능할 때도 사용자는 사용가능한 복사를 액세스할 수 있다[23]. 중복을 허용함으로서 생기는 주된 단점은 중복된 복사들이 단일 복사처럼 행동해야 한다는 것이다. 즉, 내부 일관성 뿐만 아니라 상호 일관성도 유지되어야 한다.

2.2 고장 허용과 신뢰성 고려사항

실시간 응용에서는 파국적인 고장하에서도 계속적

인 운영이 필요하고, 또한 고장으로부터의 신속한 회복이 매우 중요하므로, 고장 허용과 신뢰성은 많은 실시간 응용에서 꼭 필요한 조건이다. 고장 허용과 고도의 신뢰성은 여분의 처리과정 수행과 여분의 메세지 교환을 요구하므로, 종종 "고장 허용", "고도의 신뢰성", "신속한 응답"의 요구 조건들은 서로 모순된다. 여기서는 장소 회복과 네트워크 분할을 위한 회복 기법에 포함된 고려 사항들에 대해서 설명하겠다[21].

2.2.1 장소 회복

중복된 분산 데이타베이스 시스템에서는 한 장소가 붕괴로부터 회복을 수행할 때 데이타베이스를 최근의 상태로 재저장하여야 하고, 또한 다른 장소들과 일관성 있게 합병되어야 한다. 대부분의 회복 기법들은 데이타베이스와 트랜잭션 수행의 이력에 대한 여분의 정보를 유지 관리하고, 이를 이용하여 데이타베이스를 이전의 일관성 상태로 재저장함으로써 한 장소를 회복한다. 여분의 정보의 예는 감사 트레일(audit trail) 또는 로그(log), 검사-포인트, 차별(differntial) 화일 등이 있다. SDD-1에서의 회복은 spoolers의 개념에 근거를 두고 있는데, 여기서는 장소 붕괴가 발생할 경우에 붕괴된 장소로 향하는 모든 메세지들은 spoolers 장소에 임시 저장된다. 한 장소가 회복될 때 spooler 장소들에 저장된 모든 메세지들을 unspool 한다[9]. 실시간 응용에서 붕괴된 장소는 신속하게 회복되어야 하고 회복이 발생될 때 그 장소는(부분적으로) 기능을 수행할 필요가 있다. DDM은 어느 정도까지 이러한 기능들을 갖고 있는 시스템인데, incremental회복을 사용하여 데이타 가용성을 향상시켰고 고장 발생시에 로그에 근거한 회복 기법을 사용하여 장소들간의 데이타 전송도 감소시켰다[3].

2.2.2 네트워크 분할

고도의 가용성의 요구 조건때문에 실시간 응용은 네트워크 분할에도 불구하고 다른 분할들은 트랜잭션 처리를 계속하기를 요구한다. 그러나, 만약 네트워크 분할이 발생할 경우에 트랜잭션 처리를 허용한다면 다른 분할들내에 있는 데이타베이스의 상태는 시간이 지남에 따라 차이가 생겨서 데이타베이스의 일관성을 유지할 수 없게 된다. 그러므로, 분산 데이타베이스 시스템이 분할로부터 회복할 때 다른 분할들내에 있는 데이타베이스 상태는 공통의 값으로 조정되어야 한다. 이를 위해 모든 장소가 로그나 감사 트레일을 유지한다. 또한 데이타베이스가 분할로부터 회복될 때 다른 분할들의 저널(journal)들을 합병함으로써 공통의 저널을 구하고, 다른 분할들내의 데이타베이스에 대한 비일관성을 조정하기 위해서 합병되는 분할들의 모든 장소들에 공통의 저널을 적용한다. 실시간 응용에서는 이러한 타스크를 신속히 처리할 수 있고 회복 중에도 부분적인 기능 수행을 허용하는 기법이 요구된다. 중복된 분산 데이타베이스 시스템의 분할들을 일관성있게 통합하기 위한 서너개의 기법들이 [4]에 소개되어 있다.

2.3 동시성 제어 고려사항

분산 데이타베이스 시스템에서 중복된 데이타의 일관성을 유지하기 위해 많은 동시성 제어 프로토콜들이 제시되었다. 일관성과 가용성간의 상호 중요도를 고려하여 이러한 프로토콜들은 비관적인 방법과 낙관적인 방법으로 구분될 수 있다. 비관적인 방법의 예로는 기본 복사[1], 토큰[13], 투표[8] 방법 등이 있다. 기본 복사 방법은 한 아이템의 복사 하나를 현재 값을 갖고 있는 기본 복사로 지정한다. 토큰 방법은 각각의 아이템에 토큰을 관련시킨다. 오직 토큰을 갖는 장소만이 그 아이템을 액세스할 수 있다. 투표 방법에서는 데이타의 각각의 복사에 얼마간의 투표들이 할당된다. 만약에 한 트랜잭션이 다수의 투표들을 수집하면 그 데이타를 액세스할 수 있다. 비관적인 방법은 항상 최악의 경우를 가정하기 때문에 비일관성을 야기하는 연산은 허용되지 않는다. 그러므로 가용성은 희생되고 일관성은 유지된다.

반면에 액세스가 요구될 때 낙관적인 방법에서는 액세스가 봉쇄되지 않는다. 데이타 비일관성은 발견된 후에만 해결되며, 비일관성이 발견될 확률은 적다고 가정된다. 이러한 방법의 예로는 버전 벡타[14], 낙관적 프로토콜[4], 데이타-조각(patch)[6] 등이 있다. 버전 벡타 방법에서는 한 아이템의 각각의 복사가 그 복사에 대한 갱신의 갯수를 기록한 벡타와 관련된다. 그러므로, 충돌은 같은 아이템에 대한 버젼 벡타들을 다른 분할들과 비교함으로써 발견된다. 낙관적 프로토콜은 충돌을 발견하기 위해서 데이타 엑세스의 우선(precedence) 그래프를 작성한다. 데이타-조각 방법은 DBA가 차이지는 데이타베이스들을 통합하기 위한 규칙들을 지정할 수 있게 허용한다.

일반적으로 직렬가능성(serializability)이 병행 수행하는 트랜잭션 수행들의 정확성을 보장하기 위한 수

단으로 사용된다. 또한 일관성 단계를 강한(strong) 일관성과 약한(weak) 일관성으로 구분할 수 있다[6]. 만약에 한 트랜잭션의 수행이 다른 트랜잭션들과 직렬가능하다면 강한 일관성으로 수행되는 것이다. 판독만 하는 트랜잭션들인 경우에는 이들의 수행이 모든 갱신 트랜잭션들과 직렬가능일 필요는 없다. 즉, 오직 약한 일관성만 요구된다. 두 단계의 일관성을 갖는다면 보다 나은 처리율을 분산된 데이타베이스에서 얻을 수 있다.

분산 데이타베이스 시스템을 위한 동시성 제어 프로토콜은 이와 같이 현재 많이 제안되어 있는 형편이나, 이러한 기존의 동시성 제어 프로토콜을 실시간 분산 데이타베이스 시스템을 위해 직접 사용하는 데는 적합하지 않다. 또한, 실시간 분산 데이타베이스 시스템을 위해 제안된 동시성 제어 프로토콜은 현재 거의 없는 형편이다.

III. 동시성 제어 프로토콜

본 장에서는 실시간 분산 데이타베이스 시스템의 설계 고려사항 중에서 제일 중요한 동시성 제어 프로토콜을 서너개 소개하려 한다.

3.1 완전-분산 프로토콜

3.1.1 서 론

중복된 데이타베이스들을 위한 기존의 동시성 제어 프로토콜들에서는 충돌에 기인한 봉쇄지연이 통신지연과 갱신들간의 충돌 확률에 민감하기 때문에 성능이 저하된다. 이러한 프로토콜들에서는 갱신의 실행이 조화를 이루지 못하기 때문에 봉쇄지연이 이러한 파라메타들에 민감하게 된다. 사용자가 한 장소에 대해 갱신을 시작하면, 그 장소는 동기화를 수행하고, 갱신을 실행하고, 계산된 값들을 데이타베이스의 복사에 기록하고, 계산된 값들을 다른 모든 장소들에 퍼뜨린다. 다른 장소들은 이러한 값들을 받아서 데이타베이스의 그들의 복사에 기록한다. 이와같이 오직 한 장소만이 갱신을 완전히 수행하고 다른 장소들은 오직 갱신의 기록만 수행하기 때문에 부조화가 발생한다. 이러한 프로토콜들은 반-분산 프로토콜이라 불린다[20].

한 장소가 다른 장소들로부터 계산된 값을 받을 때까지 갱신이 봉쇄 상태로 존재할 수 있기 때문에 이러한 프로토콜에서는 봉쇄지연이 통신지연에 민감하다. 즉, 원격 장소들로부터 값들이 통신 네트워크를 통하여 오기 때문에 갱신이 봉쇄 상태로 있는 기간은 통신 네트워크 지연에 의존한다. 충돌이 증가하면 보다 긴 봉쇄지연을 야기하는 갱신 대기가 보다 빈번해 지기 때문에 봉쇄지연은 충돌 확률에 민감하게 된다.

본 절에서는 충돌에 기인한 봉쇄지연이 통신 네트워크의 속도가 아니라 디스크(I/O 장치)와 CPU 속도에 의존하는 중복된 분산 데이타베이스 시스템에서의 동시성 제어를 위한 새로운 프로토콜을 소개하겠다[21]. 디스크와 CPU가 일반적으로 통신 네트워크보다 빠르기 때문에, 이 프로토콜은 성능면에 많은 향상을 얻을 수 있다. 또한 실시간 분산 데이타베이스 시스템에서는 데이타베이스가 주로 주기억 장소에 상주하고 있기때문에 I/O 지연은 근본적으로 감소된다. 그러므로 이 프로토콜은 실시간 분산 데이타베이스 시스템에서 보다 더 적합하다.

3.1.2 완전-분산 프로토콜

데이타베이스가 1부터 M까지 번호가 할당된 M개의 데이타 객체들로 구성되었다고 가정하자. 사용자는 갱신 트랜잭션들을 실행함으로써 데이타베이스를 갱신한다. 갱신 U는 3-튜플 U=(rs, rw, f)로 정의되는데 (a) rs⊆{1, 2, ⋯, M}은 U에 의해서 판독되는 데이타 객체들을 의미하는데, U의 판독집합으로 불린다. (b) ws⊆{1, 2. ⋯, M}은 U에 의해서 기록되는 데이타 객체들을 의미하는데, U의 기록 집합이라 불린다. 또한 (c) f는 U에 의해서 수행된 계산을 모델하는 함수이다. 함수 f는 데이타베이스의 내부 일관성을 유지한다고 가정하자. 갱신 U의 판독집합과 기록집합을 각각 RS(U)과 WS(U)로 표시한다.

임의의 두개의 갱신 U^1과 U^2에 대해서 만약에 $RS(U^1) \cap WS(U^2) \neq \phi$, $WS(U^1) \cap RS(U^2) \neq \phi$ 또는 $WS(U^1) \cap WS(U^2) \neq \phi$이면, U^1이 U^2에 대해서 각각 판독-기록, 기록-판독, 또는 기록-기록 충돌이라고 한다. 또한 이러한 충돌중에서 적어도 하나가 U^1과 U^2 사이에 존재하면 갱신 U^1과 갱신 U^2는 충돌이라고 한다. 장소 S_i가 갱신 U^1를 다음과 같이 실행한다.

(a) S_i가 데이타 객체 $RS(U^1)$를 판독한다.
(b) S_i가 $WS(U^1)$내에 있는 데이타 객체들에 대한 값들을 계산한다.
(c) S_i가 계산된 값들을 데이타 복사인 $WS(U^1)$내에 있는 데이타 객체들에 기록한다.

갱신의 판독집합과 기록집합은 시스템내에서 처음부터 알 수 있다고 가정하자. 시간 스탬프(timestamp)는 Lamport의 방법[10]에 따라 생성되며, 갱신 U의 시간 스탬프는 TS(U)로 표시한다.

중복된 분산 데이타베이스 시스템에서는 갱신 실행을 위해 다음과 같은 방식을 사용한다. 한 장소가 사용자로부터 갱신을 받으면, 그 갱신에 대해서 시간 스탬프를 할당하고 갱신과 시간 스탬프를 모든 다른 장소들에 보낸다. 그리고 나면 모든 장소들은 데이타 객체들의 계산된 값들을 교환하지 않고 갱신을 완전하게 실행한다. 즉, 모든 장소들은 동기화를 수행하고 모든 갱신의 판독, 계산, 기록 작업을 실행한다. 그러므로 각각의 장소는 모든 갱신을 실행하는데 있어서 동등하게 포함되기 때문에 갱신 실행에 대해서 균형있는 또는 완전-분산 방법이 된다.

갱신 실행에 대한 완전-분산 방법에서는 동기화를 위해 낙관적 또는 비관적 기법을 사용할 수 있다. 여기서는 갱신 실행을 위해 [20]에서 제시한 프로토콜의 동기화 기법을 완전-분산 방법에 접목시킨 완전-분산 프로토콜을 소개한다.

장소 S_i가 갱신 U=(rs, rw, f)를 사용자로부터 받으면, 다음과 같은 일련의 일을 수행한다.

(a) 자신의 클럭(clock)을 갱신하고 시간 스탬프(예를 들면 ts)를 U에 할당한다.

(b) UPDATE (rs, ws, f, ts) 메세지를 모든 다른 장소들에 보낸다.

(c) 갱신을 할당된 시간 스탬프와 함께 저장한다.

장소 S_j는 다음과 같은 방법으로 UPDATE(rs, ws, f, ts) 메세지에 응답한다.

(a) 자신의 클럭을 갱신한다.

(b) S_i에게 REPLY(ts1) 메세지를 반송한다(여기서 ts1은 S_j에서 갱신된 시간 스탬프이다).

(c) (rs, ws, f, ts)를 저장한다.

장소 S_k, k=1, 2, …, N은 다음과 같은 일련의 일을 수행한 후 U의 판독과 계산 작업을 실행한다.

(a) 모든 다른 장소들로부터 'ts'보다 큰 시간 스탬프를 갖는 메세지를 받는다.

(b) 'ts'보다 작은 시간 스탬프를 갖고 'rs'내의 데이타 객체들을 변경하려는 모든 갱신들의 기록 작업을 장소 S_k에서 실행한다.

장소는 다음과 같은 일련의 일을 실행한 후 U에 대한 계산된 값들을 데이타베이스의 복사인 'ws'내에 있는 데이타 객체들에 기록한다.

(c) 'ts'보다 작은 시간 스탬프를 갖고 'ws'내의 데이타 객체들을 판독하려는 모든 갱신들의 판독 작업을 실행한다.

(d) 'ts'보다 작은 시간 스탬프를 갖고 'ws'내의 데이타 객체들을 변경하려는 모든 갱신들의 판독 작업을 그 장소에서 실행한다.

완전-분산 프로토콜에서는 모든 장소가 갱신에 의해 기록될 값들을 계산하기 때문에, 이 방법의 중요한 점은 한 장소가 다른 장소들로부터 계산된 값들을 기다릴 필요가 없다는 것이다. 계산된 값들에 대한 대기가 필요없기 때문에 장소들 간의 분리 효과가 존재한다. 이러한 분리는 봉쇄지연이 통신지연과 충돌확률에 영향을 받지않게 한다. 또한, 이것은 갱신 응답시간에도 상당한 영향을 준다. 즉, 이 프로토콜은 봉쇄지연들이 통신 네트워크의 속도에 의존하지 않고 CPU와 디스크의 속도에 의존하기 때문에 다른 프로토콜보다 빠른 응답시간과 고도의 처리률을 제공할 수 있다.

3.2 시멘틱 정족수 프로토콜

3.2.1 서 론

hard 실시간 시스템에서는 트랜잭션이 제한시간 이전에 완료되어야 한다. 그렇지 않은 경우는 수행이 실패되었다고 간주된다. 이러한 환경에서는 데이타 일관성이 유지되기를 원할 뿐만아니라 데이타베이스가 고도의 가용성도 갖기를 요구한다[16]. 사실상 데이타베이스 가용성은 여러 응용에서 데이타 일관성보다 더 중요하게 고려된다. 이것은 데이타 비일관성은 발견되자 마자 제거될 수 있는데 반해서 잃어버린 시간은 보상 받을 수 없기 때문이다. 그러므로 실시간 분산 데이타베이스 시스템에서는 엑세스가 불가능한 데이타의 존재 여부에 무관하게 분할들내에서 연산을 계속하는 것이 바람직할 뿐 아니라 아주 필요하다.

실시간 분산 데이타베이스 시스템에서 주된 문제점들 중의 하나는 고도의 가용성을 위해 데이타가 중복되어 있을 때 데이타 일관성을 유지하는 것이다. 특히 실시간 분산 데이타베이스 시스템 환경에서는 결과가 제한시간 이전에 생성되어야 하기 때문에 일관성보다 가용성이 더 중요할 수 있다. 본 절에서는 실시간 분산 데이타베이스 시스템에서 가용성을 증가시키기 위한 동시성 제어 프로토콜을 소개한다[11]. 즉, 네트워크가 분할되었을 때 데이타베이스 가용성을 증가시키기 위한 두개의 정족수(quorum)프로토콜을

소개한다. 이 프로토콜들은 대부분의 실시간 데이타베이스 시스템들이 정적이고 잘 정의된다는 사실에 가정을 둔다. 더우기 가끔 데이타 일관성이 나중에 회복될 수 있다면 잠시 무시된다고 가정한다. 또한 이 프로토콜은 가장 최근 정보를 사용할 수 없고 직렬가능한 연산이 보장될 수 없을 때 트랜잭션이 수행될 수 있게 한다.

3.2.2 시멘틱 정족수 프로토콜

네트워크에서 분할이 발생될 때 만약에 두개의 장소가 같은 분할내에 있으면 서로가 통신할 수 있고, 그렇지 않다면 통신할 수 없다. 여기서 장소 고장은 네트워크 고장과 구분 가능하고 각각의 장소는 네트워크 고장을 즉시 발견할 수 있다고 가정한다. 분할이 발생된 후에 각각의 장소는 현재 연결되어 있는 모든 장소들을 찾기 위해서 재구성(reconfiguration) 메세지를 내보낸다.

최고의 가용성을 달성하기 위해서 세 종류의 정족수가 각각의 데이타 아이템에 대해서 정의된다. 즉, 데이타 정족수, 사용자 정족수, 무결성 정족수이다. 데이타 정족수는 데이타에 대한 연산들이 직렬가능하다는 것을 보장하기 위하여 중복된 데이타 복사들에 의해서 사용된다. 판독 정족수와 기록 정족수는 [8]에서 처럼 성능을 향상시키기 위해서 사용된다. 또한, 데이타를 공용하는 프로세스들을 위한 사용자 정족수와 무결성 제한조건내에 포함된 데이타를 위한 무결성 정족수가 사용된다.

(1) 사용자 정족수

일반적으로 분산 데이타베이스 시스템에서는 언제 어디서 트랜잭션이 시작되었는지에 대한 정보를 갖고 있지 않다. 네트워크가 분할되는 동안 데이타베이스가 사용할 수 있는 유일한 정보는 데이타 복사들의 갯수이다. 일반적으로 복사들의 정족수를 수집할 수 있는 트랜잭션은 그 데이타를 사용하는 것이 허용된다. 그러나 실시간 시스템에서는 데이타 중복의 목적이 편의를 위해 간단할 수 있기 때문에 이것은 좋은 생각이 아니다. 즉, 데이타 복사들을 다수 갖고 있는 분할이 그 데이타를 가장 빈번히 또는 가장 중요하게 필요로 하지 않을 수 있다. 사용자 정족수 프로토콜에서는 한 데이타 아이템에 대해 다수 사용자들을 갖는 분할이 그 데이타에 대한 액세스(갱신) 권한을 갖게 한다. 다른 말로 말하면, writer를 소수 또는 갖지 않는 분할은 얼마나 많은 데이타 복사들을 가졌냐에 무관하게 그 데이타에 대한 판독만 가능하게 한다.

한 데이타 아이템에 대한 사용자 정족수내의 투표들은 그 아이템을 갱신하는 프로세스들에 할당된다. 이러한 투표들은 오직 분할이 발생될 때 수집된다. 데이타베이스가 완전하게 연결되면 동시성 제어를 위해서 오직 데이타 정족수만 사용된다. 한 프로세스가 분할을 발견할 때, 그때 시간이 기록되고 시스템은 "분할" 모드로 변환된다. 분할내에 있는 기록 프로세스들의 전체 갯수는 사용자 정족수와 비교된다. 기록 프로세스들의 갯수가 이전의 사용자 정족수보다 크면, 데이타 정족수는 그 분할내에 있는 기록들을 수용하기 위해서 갱신된다. 그렇지 않으면, 데이타 정족수는 어떠한 갱신도 허용하지 않기 위해서 변경된다. 그러므로 각각의 분할은 분할내에 있는 프로세스들의 실제 요구에 따라서 데이타를 활용하기 위한 권한을 갖는다. 더우기 동적 정족수 할당으로 어떠한 두개의 프로세스도 같은 또는 다른 분할 내의 데이타 아이템을 비일관성있게 갱신할 수 없다.

(2) 무결성 정족수

실시간 시스템에서는 모든 병행 수행되는 연산들을 직렬화시키는 것이 불가능할 수 있다. 그러므로 시스템 정확성을 위한 다른 기준(criteria)이 필요하다. 데이타베이스의 정확성은 무결성 제한조건에 의해서 실제로 결정된다. 그러나 데이타베이스내에 포함된 제한조건들의 전체 갯수가 너무 커서 모든 제한조건들을 검사하는 것은 비실용적이기 때문에 대부분의 데이타베이스 시스템들은 병행 수행되는 트랜잭션들의 정확성 척도로써 직렬가능성을 사용한다. 그러나 실시간 시스템들은 보통 고정된 갯수의 프로세스들을 갖고 데이타베이스들은 정적으로 구성되기 때문에, 시스템의 정확성에 대해서 매우 중요한 작은 갯수의 무결성 제한조건들을 지정하는 것이 바람직할 수 있다. 제한조건내의 데이타 아이템이 갱신될 때마다 트랜잭션은 다른 트랜잭션들이 제한조건을 위반하면서 제한조건내의 그 데이타 아이템을 갱신하지 않는다는 것을 확인해야 한다.

실시간 데이타베이스가 분할될 때 만약에 제한조건내에 포함된 모든 데이타 아이템들이 한 분할내에서 기록 가능하면 이들은 그 분할내에서 정상적으로 갱신될 수 있다. 만약에 제한조건내의 데이타의 부분만이 분할내에서 기록 가능하면, 기록 불가능한 데이타는 다른 분할들내에서 비일관적으로 갱신될 수 있다. 이에 대한 해결책은 분할들 중에서 오직 한 군데내에서만 프로세스들이 데이타를 갱신하고 제한조건을 유지하게 허용하는 것이다. 다수의 기록 가

능한 아이템들을 갖는 분할이 그 특권을 갖게 선택된다. 즉, 일반적으로 더 많은 트랜잭션들이 수행될 수 있게 하기 위해서 제한조건내의 다수 데이타 아이템들을 갖는 분할이 선택된다.

(3) 일관성 단계

사용자 정족수와 무결성 정족수를 사용하면 데이타 아이템들의 일부분만도 각각의 분할 내에서 기록 가능할 수 있다. 만약에, 한 트랜잭션에 의해서 사용되는 모든 데이타가 분할내에서 기록 가능하면, 그 트랜잭션은 일반적으로 수행될 수 있다. 새로운 정족수들의 집합이 할당될 수 있으나 그 트랜잭션은 중단되지 않는다. 만약에 한 트랜잭션에 의해서 요구된 모든 데이타가 무결성 정족수를 사용하여 분할내에서 기록 가능하지 않다면, 그 트랜잭션은 판독만 가능한 아이템들이 다른 분할들 내에서 갱신될 수 없다고 결론 내릴 수 있다. 두 경우 모두에서 데이타베이스의 일관성은 손상되지 않는다.

한 트랜잭션이 제한시간 이전에 종료되어야 하나 약간의 데이타를 이용할 수 없는 경우에 트랜잭션은 최근 것이 아닐 수 있는 데이타 값들을 갖고 진행되기를 원할 수 있다. 어떤 트랜잭션들에서는 사용된 데이타가 일시적인 일관성을 갖고 있는 한 여전히 엑세스 가능하다. 판독만 하는 많은 트랜잭션들에서는 이러한 일시적인 일관성이 요구되는 전부이다. 결과가 요구되지만 최근 데이타나 일시적인 일관성을 갖는 데이타를 사용할 수 없는 가장 극단적인 경우에도 트랜잭션은 여전히 진행될 수 있다. 이러한 결과는 임시적인 응답을 제공하기 위해 일반적으로 요구된다. 보다 더 일관성 있는 데이타가 사용가능한 후에 이 결과는 정확해 질 수 있다.

(4) 알고리즘

데이타베이스내의 모든 데이타 아이템에 대해서 정족수 벡타(t, c, rq, wq, uq)가 할당된다. 이때, t는 데이타가 갱신된 마지막 시간이고, c는 사용 가능한 투표들의 갯수이고, rq, wq, uq는 각각 데이타 판독 정족수, 데이타 기록 정족수, 사용자 정족수이다. 시스템이 정상적으로 운영될 때 동시성 제어를 위해서 데이타 정족수들을 사용한다. 여기서 데이타 복사가 한개 이상의 투표들을 갖는 것이 허용된다. 각각의 아이템에 대해서 트랜잭션은 판독하기 위해서 적어도 rq 투표들을 가져야 하고, 기록하기 위해서 wq 투표들은 가져야 한다. 데이타 정족수들은 다음과 같은 제한 조건들에 따라 할당된다.

(1) $rq+wq>c$

(2) $2wq>c$

첫번째 제한조건은 판독-기록 충돌을 피하기 위한 것이고, 두번째 제한조건은 기록-기록 충돌을 피하기 위한 것이다.

하나의 트랜잭션이 시작되었을 때 프로세스는 요구되는 일관성 수준을 지정해야 한다. 각각의 트랜잭션은 미리 정의된 무결성 제한 조건들의 집합을 역시 갖고 있다. 트랜잭션이 수행되기 위해서는 그 분할이 충분한 무결성 투표들을 갖고 있는지를 알아보기 위해서 각각의 제한조건에 포함된 데이타를 검사한다. 만약에 그렇지 않다면 그 트랜잭션은 즉각 취소된다. 통상적으로 트랜잭션은 세가지 일관성 수준을 지정할 수 있다. 즉, 강한(strong) 일관성, 약한(weak) 일관성, 그리고 낙관적(optimistic) 일관성이다. 이러한 세가지 일관성에 대한 자세한 설명은 [11]을 참조하라.

네트워크 분할이 발생하면 각각의 분할내의 사용가능한 데이타 투표들, c', 사용자 숫자, u가 새로운 정족수 벡타를 생성하기 위해서 수집된다. 정족수 벡타는 강한 일관성을 위해서 정의된다. 낙관적 일관성을 갖고 수행하기를 원하는 트랜잭션들은 정족수를 따를 필요가 없다. 정족수들을 재할당하기 위해서 분할내에 있는 사용자 프로세스들의 갯수가 이전의 uq와 비교된다. 이때 다음과 같은 두가지 경우가 가능하다.

(1) $u \geq uq/2$

이것은 사용자 프로세스들의 다수가 분할내에 존재하는 것을 의미한다. 새로운 판독 정족수와 기록 정족수가 분할내에서 사용 가능한 투표들의 전체 갯수에 근거하여 재할당된다. 가능한 정족수의 벡타는 (t, c', $\lceil (c'/2) \rceil$, $\lceil (c'+1)/2 \rceil$, uq)이다.

(2) $u<uq/2$

이 경우는 정족수의 벡타가 (t, c', ∞, ∞, ∞)로 할당된다. 어떠한 갱신도 허용되지 않기 때문에 모든 정족수들이 ∞로 할당된다.

분할들이 재연결될 때 모든 분할들내에 있는 모든 갱신 이력(history)들이 조사된다. 만약에 어떠한 트랜잭션도 낙관적 일관성을 사용하지 않았다면, 모든 데이타는 시간 스탬프 순서에 따라 합병된다. 만약 어떤 트랜잭션이 낙관적 일관성을 사용하였다면, 대

이타베이스는 취소되거나 약간의 연산들을 보충해야 한다. 낙관적 일관성하에서의 갱신은 강한 일관성을 사용하는 갱신 또는 낙관적 일관성을 사용하는 다른 갱신과 충돌되어서는 안된다.

2.3 통합된 프로토콜

2.3.1 서 론

전형적으로 동시성 제어 프로토콜 설계자는 트랜잭션들의 시간 제한조건들을 고려하지 않는다. 그보다는 데이타베이스의 일관성을 유지하면서 동시성을 최대화하는 프로토콜을 설계한다. 광범위하게 사용되고 있는 동시성 제어 프로토콜은 2-단계(two phase) 록(lock) 프로토콜이다[5]. 각각의 동시성 제어 프로토콜은 관련된 스케쥴들(프로토콜에 의해 허용되는 트랜잭션 단계들의 interleavings) 집합을 갖는다. 반면에, 실시간 스케쥴링 프로토콜 설계자는 전형적으로 데이타 일관성 문제에 대해서 고려하지 않는다. 그보다는 스케쥴 가능성을 최대화하는 식으로 타스크들의 수행을 우선화하는 방법을 찾는다.

데이타베이스 일관성을 유지하고 상위 수준의 스케쥴을 제공하기 위해서는 최악의 봉쇄 기간을 최소화하는 동시성 제어 프로토콜을 사용하여야 한다. 또한 봉쇄가 발생하면 타스크 우선순위를 적절히 관리해야 한다. 본 절에서 소개되는 방법은 두 가지 중요한 측면을 갖는다. 첫째, 중단성은 모듈러(modular) 동시성 제어의 이론[17]을 사용하여 데이타베이스와 트랜잭션들을 분해함으로써 증진된다. 둘째, 봉쇄동안 우선순위는 우선순위 한계 방법을 사용하여 관리된다. 즉, 동시성 제어를 위한 setwise 2-단계 록킹 프로토콜과 실시간 스케쥴링을 위한 우선순위 한계 프로토콜을 결합한 통합된 프로토콜을 소개한다[15, 18].

2.3.2 setwise 2-단계 록킹 프로토콜

직렬가능성(serializability) 이론[2]에 의해 제공되는 동시성은 제한적이며, 또한 이것은 비직렬 가능한 동시성 제어 방법에 대한 연구의 동기가 되었다. 실시간 스케쥴링 관점에서, [17]에서 사용된 분해 방법은 동시성 제어에서 야기된 봉쇄 기간을 단축시킴으로써 중단성을 증진시킨다. 이러한 방법은 다음과 같이 3가지 단면을 갖는다.

(1) 데이타베이스를 원자 데이타 세트(ADS)들로 분해

각각의 원자 데이타 세트의 일관성은 다른 원자 데이타 세트들과 무관하게 유지 관리될 수 있다. 또한, 모든 ADS 일관성 제한 조건들의 결합은 데이타베이스의 일관성 제한 조건들과 동등하며, 원자 데이타 세트들의 합집합도 역시 원자 데이타 세트이다.

(2) 복합 트랜잭션을 기본 트랜잭션들의 부분 순서 집합으로 분해

각각의 기본 트랜잭션은 데이타베이스의 일관성을 유지하며, 홀로 수행될 때 자신의 후(post)-조건들을 만족한다. 부가적으로, 임의의 수행 경로상에 있는 복합 트랜잭션의 기본 트랜잭션들의 후-조건들의 결합은 그 복합 트랜잭션의 후-조건들과 동등하다.

(3) 기본 트랜잭션들이 원자 데이타 세트에 대해서 직렬가능으로 수행된다는 것을 보장하기 위해서 록킹 프로토콜의 사용

트랜잭션들의 한 세트가 setwise하게 직렬적으로 수행된다면, 대응되는 스케쥴은 setwise 직렬가능 스케쥴이라 불린다. 복합 트랜잭션들이 기본 트랜잭션들로 분해되고 오직 기본 트랜잭션들이 setwise하게 직렬적으로 스케쥴된다면, 대응되는 스케쥴은 일반화된 setwise 직렬가능 스케쥴이라 불린다. 더우기 일반화된 setwise 직렬가능 스케쥴들은 일관성, 정확성, 모듈성의 특성들을 갖는다[17].

실시간 데이타베이스는 데이타베이스 객체들의 중복이 없는 원자 데이타 세트들로 종종 분해될 수 있다. 예를 들면, 비행기가 두개의 레이더 기지에 의해 추적되고 있고, 데이타 객체 O_1과 O_2의 집합은 이러한 두 기지들의 지역(local) 관점들을 표현한다고 가정하자. 이러한 데이타 객체들은 특별한 기지에서 보여지는 현재 위치, 속도, 비행기의 신원 확인 등을 포함할 수 있다. 이러한 데이타 객체들의 각각은 하나의 원자 데이타 세트를 형성한다. 이것은 각각의 추적과 관련된 일관성 제한조건은 지역적으로 검사되고 검증될 수 있기 때문이다. 새로운 스캔(scan)은 데이타 객체의 새로운 버젼을 생성하고, 시간의 경과 중에 O_1과 O_2의 값들은 두개의 서로 연관된 다변화(multivariate)시간 연속(series)들을 형성한다. 이러한 상호 연관은 전역(global) 데이타 객체 O_3를 생성하는데 사용될 수 있다. 합집합 $G=\{O_1, O_2 O_3\}$은 감시하에 있는 비행기의 전역과 지역 관점들을 표현한다. 즉, 전역 추적 O_3를 루트로 하고 두개의 지역 추적인 O_1과 O_2를 잎으로 하는 트리 형태의 전역 원자 데이타 세트 G를 얻는다.

복합 트랜잭션은 데이타베이스 연산과 비-데이타

베이스 연산 모두를 갖는 일반화된 타스크를 모델링 한다. 동시성 제어 관점에서는 비-데이타베이스 연산에 대해서 어떠한 특별한 취급도 하지 않는다. 그러나, 실시간 스케쥴링 관점에서 데이타베이스 연산은 봉쇄를 야기하는 반면에 비-데이타베이스 연산은 봉쇄를 야기할 수 없기 때문에 이들을 구분하는 것이 중요하다. 이제부터 비-데이타베이스 연산인 기본 트랜잭션은 "기본 타스크"라고 언급하겠고, "기본 트랜잭션"이라는 말은 데이타베이스에 연산하는 기본 트랜잭션들을 위해 사용하겠다. 마지막으로 "타스크"와 "복합 트랜잭션"이란 용어는 같은 의미로 사용하겠다.

기본 트랜잭션들을 위한 간단한 록킹 프로토콜은 setwise 2-단계 록킹 프로토콜이다. 기본 트랜잭션은 원자 데이타 세트에 대한 모든 록을 얻을 때까지 원자 데이타 세트에 대해 어떠한 록도 해제(release)할 수 없다. 일단 원자 데이타 세트에 대해서 하나의 록을 해제하면, 그 원자 데이타 세트에 대해 새로운 록을 얻을 수 없다. 그러나, 다른 원자 데이타 세트들에 대해서는 새로운 록을 할 수 있다. 복합 트랜잭션의 모든 기본 트랜잭션들에 의해 공용되는 복합 트랜잭션들의 지역 변수인 원자 변수들을 사용함으로써 기본 트랜잭션들은 그들의 계산 결과를 전달한다.

분산 데이타베이스 환경에서 원자 데이타 세트는 분산을 위한 데이타 객체들의 논리적 단위이다. ADS에 대해 연산하는 기본 트랜잭션은 동시성 수행을 위한 연산의 원자 단위를 나타낸다. 예를들면, 전역 ADS G는 많은 지역 추적 ADS들과 전역 추적 ADS의 합집합이 될 수 있다. 합집합내의 각각의 ADS는 다른 컴퓨터에 놓여질 수 있다. 신뢰성을 향상시키기 위해서 한 원자 데이타 세트를 다른 프로세서에 중복시킬 수 있다. 중복된 ADS와 원래의 ADS는 단일 원자 데이타 세트의 부분인 것처럼 보일 수 있다. 그러나 만약에 하나가 다른 것의 이력(historical) 버전으로 허용된다면 이들은 두개의 상이한 원자 데이타 세트들이 된다. 예를 들면, 네트워크내에 존재하는 두개의 노드상에 두개의 ADS가 있다고 가정하자. 즉, ADS $A_1=\{O_1\}$과 이의 복사 $A_2=\{O_2\}$를 갖고 있다고 가정하자. 만약에 A_1과 A_2가 모든 참조들에 관해서 동일하다고 한다면 (즉 $O_1=O_2$), 이러한 데이타 객체들은 단일 ADS의 부분이 될 것이며, 또한 이들에 대한 갱신은 동시적인 사건으로 보여져야 한다. 이것은 O_1과 O_2에 대한 동시적인 갱신을 수행하기 위해서 sert-wise 2-단계 록킹을 사용함으로써 해결될 수 있다. 그러나, 하나가 다른 것의 이력 복사라는 요구조건을 만족하기 위해서는 A_1과 A_2는 두개의 ADS로 모델될 수 있고 비동기적으로 갱신될 수 있다.

원자 데이타 세트들은 논리적으로 일관성있는 관점을 제공하지만, 이들의 복사는 네트워크상의 지연 때문에 일시적인 불일치를 야기할 수 있다. 즉, 관점들의 어떤 것들은 최근의 것이 아닐 수 있다. 데이타 객체의 이력 버전으로써 복사를 허용하는 것은 성능을 저해하는 요인으로 보일 수 있다. 그러나, 그 반대가 항상 사실이다. 통신지연 때문에, 원격 복사들을 지역 복사와 같이 최근의 것으로 만드는 것은 불가능하다.

물론 통신 지연때문에 지역과 원격 프로세서들에 대해 동일한 제한 시간을 지키는 것은 어렵다. 전형적으로 원격 장소들에 있는 데이타 객체들의 버전들은 지역 장소보다 뒤떨어질 것이다. 그러므로, 네트워크 전반에 걸친 일관성 관점을 보장하기 위해서는 동시성 제어 문제는 분산된 버전들내에서 시간 지연을 제한하는 네트워크-단계 실시간 스케쥴링 문제가 된다. 일단 지연이 제한되면 분산 타스크들은 분산 데이타 객체들의 적절한 버전들을 읽을 수 있고, 또한 그들의 결정이 일관성있는 데이타에 근거하였다는 것을 보장할 수 있다.

2.3.3 우선순위 한계 프로토콜

시그날 처리와 상호 연관과 같은 비-데이타베이스의 연산들의 지속 시간은 종종 데이타베이스를 읽고 쓰는데 걸리는 시간보다 훨씬 길 수 있다. 그러므로 실시간 동시성 제어 프로토콜의 연구에 있어서 비-데이타베이스 연산들의 스케쥴링을 빠뜨릴 수 없다. 실시간 스케쥴링 알고리즘에 따라 타스크(복합 트랜잭션)들에 우선순위를 할당하고, 이들의 기본 타스크들과 기본 트랜잭션들은 할당된 우선순위를 사용하게 된다. 그러나, 이것만으로는 충분하지 않다. 즉, 데이타베이스 원소들의 공용에 기인하여 타스크들이 서로가 독립적이지 않을 때, 우선순위 역(inversion)이 존재할 수 있다. 즉, 상위 우선순위 타스크가 하위 우선순위 타스크들에 의해서 봉쇄되었을 때 우선순위 역이 발생되었다고 말한다. 우선 순위 역은 트랜잭션 시스템들에서는 불가피한 것이다. 그러나 실시간 응용에서 상위수준의 스케쥴을 달성하기 위해서 우선순위 역이 무한정한 시간동안 발생하는 제어할 수 없는 우선순위 역 문제를 해결해야 한다.

그러므로 주어진 동시성 제어 프로토콜을 위한 적절한 우선순위 관리 프로토콜을 설계하여 데드록이 피해질 수 있고 봉쇄 시간이 아주 제한되게 하는 것이

필요하다. 우선순위 계승의 사용은 록킹 프로토콜에 의해 야기되는 임의의 지연을 제한하기 위한 한 방법이다. 우선순위 계승의 기본 개념은 타스크 τ의 트랜잭션 T가 보다 상위 우선순위 타스크들을 봉쇄할 때, T에 의해 봉쇄된 모든 트랜잭션들의 우선순위 중에서 최고 우선순위를 갖고 기본 트랜잭션 T가 수행된다는 것이다. 이러한 간단한 우선순위 계승 개념은 상위 우선순위 타스크의 봉쇄 시간을 하위 우선순위 타스크들의 전체 수행시간에서부터 오직 하위 우선순위 타스크들의 기본 트랜잭션들의 지속 시간으로 감소시킨다. 그러나, 이러한 간단한 생각은 다음의 두가지 이유에서 충분히 적합하지 않다. 첫째, 데드록의 문제가 해결되지 않는다. 둘째, 봉쇄된 것들의 체인이 형성될 수 있으므로, 한 타스크의 봉쇄 기간이 비록 제한되지만 여전히 길 수 있다.

위에서 언급한 두가지 이유때문에 보다 나은 "우선순위 한계 프로토콜"이 제시되었다[18]. 이 프로토콜은 타스크 τ의 봉쇄 시간을 기껏해야 하나의 기본 트랜잭션으로 최소화할 뿐만 아니라 또한 데드록을 방지해 준다. 이 프로토콜의 기본 개념은 한 트랜잭션이 다른 트랜잭션을 봉쇄할 때 새로운 트랜잭션이 수행되는 우선순위는 우선순위 계층 프로토콜을 고려하여 봉쇄된 모든 트랜잭션들의 우선순위들 보다 엄밀히 높다는 것을 보장한다. 만약 이 조건이 만족될 수 없다면, 트래잭션 T는 일시 정지되고 T를 봉쇄한 트랜잭션이 T의 우선순위를 계승한다.

IV. 결 론

실시간 분산 데이타베이스 시스템은 신속한 응답 시간과 파국적인 고장에도 불구한 계속적인 연산과 같은 엄격한 성능 제한조건을 갖는 응용을 지원한다. 실시간 분산 데이타베이스 시스템에서는 데이타가 여러 장소에 중복되어 저장될 수 있고, 또한 한 장소에서 수행중인 트랜잭션이 다른 장소에 저장된 데이타를 액세스할 필요가 있을 수 있다. 이러한 상황에서 트랜잭션이 완료되기 위해서는 서너 장소 모두가 같이 운영되어야 한다. 신뢰성 없는 장소를 갖는 실시간 분산 데이타베이스 시스템에서는 트랜잭션에 대한 데이타의 가용성을 증가시키기 위하여 데이타를 중복시켜서 여러 장소에 저장한다. 그러면 한 장소가 고장 나더라도 중복된 데이타를 갖는 다른 장소를 액세스할 수 있다. 데이타의 중복은 가용성의 향상 뿐만아니라 데이타를 요구하는 프로세스에 가깝게 저장함으로써 성능도 향상시킬 수 있다. 즉, 필요한 데이타가 저장된 장소에서 질의가 발생되면 통신 지연없이 지역적으로 처리될 수 있고, 또한 작업 부하가 전체 장소들에 균등히 배분될 수 있다. 그러나 데이타 중복은 중복된 데이타의 동기화를 위한 부가적인 작업을 요구한다.

본 논문에서는 실시간 분산 데이타베이스 시스템의 설계를 위한 고려사항들에 대해서 언급하였다. 즉, 실시간 분산 데이타베이스 시스템의 설계에 필요한 일반 요구조건들에 대해서 설명하였다. 그리고 고장 허용과 신뢰성 고려사항에 대해서 언급하였고, 동시성 제어 고려사항에 대해서도 설명하였다. 특히 실시간 분산 데이타베이스 시스템의 설계에서 제일 중요한 동시성 제어 프로토콜을 서너개 소개하였다. 즉, 오랜 통신 지연이 실시간 분산 데이타베이스 시스템에서 성능을 제한하는 주된 요인이기 때문에, 성능이 통신 네트워크의 속도에 의존하지 않고 CPU와 I/O 장치의 속도에 의존하는 완전-분산 프로토콜을 소개하였다. 또한, 실시간 제한 조건들을 갖는 트랜잭션들은 분할에도 불구하고 계속 진행되기를 원할 수 있다. 네트워크 분할 동안 실시간 분산 데이타베이스 시스템의 가용성을 향상시킬 수 있는 시멘틱 정족수 프로토콜을 소개하였다. 마지막으로 모듈러 동시성 제어 이론과 실시간 스케쥴링 프로토콜을 결합하여 우선 순위 한계를 갖는 setwise 2-단계 록킹 방법인 통합된 프로토콜을 소개하였다.

참 고 문 헌

1. Alsberg, P. A., Day. J. D., "A Principle for Resilient Sharing of Distributed Resources," Proc. Int. Conf. Soft. Eng., 1976, pp. 627~644.
2. Bernstein, P. A. and Goodman, N., "Multiversion Concurrency Control-Thery and Algorithms," ACM Trans. on Database Systems, Vol. 8, No. 4, 1983, pp. 465~483.
3. Chan, A., Dayal, U., Fox, S., Goodman, N., Skeen, D. and Ries, D., "DDM: An Ada Compatible Distributed Database Manager," IEEE COMPCON Digests of Papers, 1983.
4. Davidson, S B., Garcia-Molina, H and Skeen, D., "Consistency in Partitioned Network," ACM Computing Surveys, Vol. 17, No. 3. 1985, pp. 341~370.
5. Eswaran, K. P., Gray, J. N., Lorie, R. A. and

Traiger, I. L., "The Notion of Consistency and Predicate Lock in a Database System," Communication of ACM, Vol. 19, No. 11, 1976, pp. 624~633.
6. Garcia-Molina, H., *et al.*, "Data-Patch: Integrating Inconsistent Copies of a Database after a Partition," Proc. Symp. Reliability in Distributed Soft. and Database Systems, 1983, pp. 38~48.
7. Gardarin, G., Simon, E. and Verlaine, L., "Temporal View: A Tool for Real Time Distributed Data Bases," Distributed Data Sharing Systems, Schreiber and Litwin (eds.), North-Holland, 1985, pp. 195~202.
8. Gifford, D. K., "Weighted Voting for Replicated Data," Proc. Symp. Operating Systems Principles, 1979, pp. 150~162.
9. Hammer, M. and Shipman, D., "Reliability Mechanism for SDD-1: A System for Distributed Databases," ACM Trans. on Database Systems, Vol. 5, No. 4, 1980, pp. 431~466.
10. Lamport, L. "Time, Clocks and Ordering of Events in Distributed Systems," Communications of ACM, Vol. 21, No. 7, 1978, pp. 558~565.
11. Lin, K. and Lin, M., "Enhancing Availability in Distributed Real-Time Databases," ACM SIGMOD RECORD, Vol. 17, No. 1, 1988, pp. 34~43.
12. Liu, L. Y. and Shyamasundar, R. K., "Static Analysis of Real-Time Distributed Systems," IEEE Trans. on Soft. Eng., Vol. 16, No. 4, 1990, pp. 373-388.
13. Minoura, T. and Wiederhold, G., "Resilient Extended True-Copy Token Scheme for a Distributed Database System," IEEE Trans. on Soft. Eng., Vol. 8, No. 3, 1982, pp. 173~189.
14. Parker, D. S. *et al.*, "Detection of Mutual Inconsistency in Distributed Systems," IEEE Trans. on Soft. Eng., Vol. 9, No. 3, 1983, pp. 240~247.
15. Rajkumar, R., *Synchronization in Real-Time Systems, A Priority Inheritance Approach,* Kluwer Academic Pub. 1991.
16. Ramamritham, K. and Stankovic, J., "Dynamic Task Scheduling in Distributed Hard Real-Time Systems," IEEE Software, Vol. 1, No. 3. 1984.
17. Sha, L., Lehoczky, J. P., and Jensen, E. D., "Modula Concurency Control and Failure Recovery." IEEE Trans. on Computers, Vol. 37, No. 2, 1988, pp. 146~159.
18. Sha, L., Rajkumar, R. and Lehoczky, J. P., "Concurrency Control for Distributed Real-Time System Databases," ACM SIGMOD RECORD, Vol. 17, No. 1, 1988, pp. 82~98.
19. Shin. K. G., "HARTS: A Distributed Real-Time Architecture," IEEE Computer, Vol. 24, No. 5, 1991, pp. 25~35.
20. Singhal, M. and Agrawala, A. K., "A Concurrency Control Algorithm and its Performance for Replicated Database Systems," Proc. Int. Conf. on Distributed Computing Systems," May 1986.
21. Singhal, M., "Issued and Approaches to Design of Real-Time Database Systems," ACM SIGMOD RECORD, Vol. 17, No. 1, 1988, pp. 19~33.
22. Son, S. H., "Using Replication for High Performance Database Support in Distibuted Real-Time Systems," Proc. Symp on Real-Time Systems, 1987, pp. 79~86.
23. Son, S. H., "Replicated Data Management in Distributed Database Systems." SIGMOD RECORD, Vol. 17, No. 4, 1988, pp. 62~69.
24. Son, S. H. and Chang, C. H., "Distributed Real-Time Database Systems: Prototyping and Performance Evaluation," Int. Symp. on Database Systems for Advanced Applications, 1989, pp. 251~259.
25. Son, S. H., "Real-Time Database Systems: A New Challenge," IEEE Data Engineering, Vol. 13. No. 4. 1990, pp. 233~241.
26. Son, S. H., Poris, M. S. and Iannacone, C. C., "Implementating a Distributed Real-Time Database Manager," Int. Symp. on Database Systems for Advanced Applications, 1991, pp. 51~60.

한 기 준

1979 서울대학교 수학교육학과(이학사)
1981 한국과학기술원 전산학과(공학석사)
1985 한국과학기술원 전산학과(공학박사)
1990 Stanford대학 전산학과 visiting scholar
1985 ~현재 건국대학교 전기계산학과 부교수
관심 분야 : 객체 지향 데이타베이스, 멀티미디어 데이타베이스, 실시간 데이타베이스, 연역 데이타베이스