실시간 능동 데이터베이스에서 triggering 관계를 고려한 동시성 제어 기법

홍 석 희†

요 약

실시간 능동 데이터베이스 시스템은 외부의 환경의 변화에 반응하여 트랜잭션들을 주어진 시간 내에 처리할 수 있어야 한다. 본 논문에서는 실시간 능동 트랜잭션들을 위한 동시성 제어 기법인 다중버전 실시간 능동 동시성 제어 기법을 제안한다. 능동 규칙에 의해서 트랜잭션들 사이에는 triggering 관계가 성립하며 공유되는 데이터에 대한 충돌을 해결하기 위해 사용되는 중요한 개념이다. 제안하는 동시성 제어 기법은 데이터 충돌에 포함된 트랜잭션들 사이의 triggering 관계와 우선순위, 선행관계 등을 참조하여 데이터 충돌을 해결한다. 우선순위가 높은 트랜잭션은 낮은 우선순위의 트랜잭션 보다 유리한 서비스를 받는다. 그러나, 낮은 우선순위의 트랜잭션이라도 종료시점에 근접했다면 철회됨으로서 낭비될 시스템 자원을 보존하기 위해서 유리한 서비스를 받을 수 있다. 또한, 트랜잭션들 사이의 triggering 관계를 기반으로 intra triggering 충돌과 inter triggering 충돌로 분류해 각각 다른 방식으로 데이터 충돌을 해결한다. 본 논문에서는 모의실험을 통해서 제안한 동시성 제어기법의 성능을 비교 평가하였다.

키워드 : 실시간 데이터베이스, 동시성제어기법, 능동 데이터베이스

Concurrency Control Based on Triggering Relationship for Real-Time Active Databases

Seok-Hee Hong†

ABSTRACT

Transactions in real-time active databases have the notion of activeness where transactions are generated by external effects and another transaction. In this paper, we propose the multiversion concurrency control algorithm for real-time active transactions. A real-time active transaction has a timing constraint in the form of a deadline until which the user wants to complete the transaction, and is characterized by triggering relationships which mean that association between a transaction that triggers the execution of another transaction and the triggered transaction. The triggering relationship is an important factor to resolve data conflicts among real-time active transactions. The proposed concurrency control mechanism resolves data conflicts by considering triggering relationships between conflicting transactions as well as priorities and precedence relationships. The conflict resolution mechanism considers association types of the triggering relationship such as abort and commit dependency, and then resolves data conflicts in favor of higher priority transactions. *We also present the experimental results of our algorithm comparing* other real-time active concurrency control algorithms.

Key word : real-time database, concurrency control, active database

1. 서 론

1.1 연구 배경

실시간 시스템은 외부 환경으로부터 다양한 변화들을 감지하면서 여러 가지 하부 조절 시스템을 관리하는 기능을 한다. 실시간으로 대량의 데이터를 처리해야 하기 때문에 데이터베이스 시스템과의 연동을 필수적으로 요구한다. 데이터베이스 시스템은 다중 사용자에게 효율적이고 안정적으로 데이터 요청을 처리하는 복잡한 시스템으로 실시간 시스템의 중요한 하부 시스템 중의 하나이다. 실시간 시스템을 위한 실시간 데이터베이스 시스템의 트랜잭션들은 주어진 시간제약(timing constraints)을 만족해야 한다. 이와 같은 시간제약을 종료시한(deadline)이라고 하며 각 트랜잭션은 이 종료시한 내에 수행을 완료해야 한다[3-5, 7]. 실시간 데이터베이스 시스템의 트랜잭션들이 일반 데이터베이스 시스템의 트랜잭션과 다른 점은 이와 같은 종료시한이라는 시간 개념을 가진다는 것이다. 실시간 트랜잭션들이 데이터

※ 이 논문은 1998년 한국학술진흥재단의 학술연구비에 의하여 지원되었음
† 정 회 원 : 경성대학교 정보과학부 조교수
논문접수 : 2000년 8월 28일, 심사완료 : 2001년 2월 1일

일관성을 만족하면서 효율적으로 처리되기 위해서는 종료시한을 고려하여 트랜잭션들을 수행시켜야 한다. 최근까지 데이터베이스 분야에서 트랜잭션 처리에 관한 연구는 많은 성과를 가지고 있다. 그러나, 일반 데이터베이스 시스템을 위해서 연구되어진 트랜잭션 처리 기법은 실시간 데이터베이스에 직접 적용되기 힘들다. 따라서, 실시간 데이터베이스 시스템을 위한 트랜잭션 처리 기법의 연구는 중요한 분야라고 할 수 있다.

최근에 실시간 데이터베이스 시스템의 응용 분야는 컴퓨터에 의한 제조, 항공관제 시스템, 교통 통제 시스템, 군사 제어 시스템, 공장 자동화 시스템과 같이 점점 다양하게 확대되고 있다. 이와 같은 실시간 데이터베이스 시스템은 외부 환경에 대해서 다양하고 신속한 반응을 해야 하기 때문에 능동적인 요소를 지원해야 한다. 이와 같은 능동적인 요소를 실시간 데이터베이스 시스템에 적용한 시스템을 실시간 능동 데이터베이스 시스템이라 한다. 실시간 능동 데이터베이스는 외부의 사건이나 다른 트랜잭션에 의해서 트랜잭션들이 수행되는 실시간 데이터베이스 시스템이다. 실시간 능동 데이터베이스에서 트랜잭션은 각 트랜잭션이 만족해야 하는 종료시한 외에 능동 규칙에 의해서 다른 트랜잭션을 수행시키는 능동 관계를 만족해야 한다.

1.2 연구 내용

일반 데이터베이스 분야에서 연구되어진 많은 기법들을 실시간 데이터베이스 시스템에 직접 적용시키기는 쉽지 않다. 일반 데이터베이스 시스템에서 성능평가의 기준이 되는 것은 트랜잭션들의 평균응답 시간이다. 이에 반해 실시간 데이터베이스 시스템은 종료시한 내에 수행을 완료한 트랜잭션들의 수로서 성능을 평가할 수 있다. 일반 데이터베이스 시스템에서는 모든 트랜잭션이 동일한 조건하에서 수행되지만 실시간 데이터베이스의 트랜잭션들은 각각 만족해야 하는 종료시한이 다르기 때문에 동일한 조건으로 모든 트랜잭션을 수행시킬 수 없다. 예를 들어, 두 트랜잭션 T_1과 T_2가 각각 만족해야 하는 종료시한이 T_1보다 T_2가 앞선다면 실시간 데이터베이스 환경에서는 T_1보다는 T_2를 더 유리한 조건으로 수행시킴으로써 트랜잭션 T_2가 T_1보다 먼저 수행을 완료할 기회를 가질 수 있게 한다. 일반 데이터베이스 시스템의 스케줄링 기법들은 모든 트랜잭션을 동일한 조건으로 취급하기 때문에 각 트랜잭션의 종료시한을 고려해야 하는 실시간 데이터베이스 시스템에는 적합하지 않다.

능동 데이터베이스에 대한 연구는 활발한 편이나 실시간 개념을 고려한 실시간 능동 데이터베이스에 대한 연구는 아직 초보적인 수준에 머무르고 있다. 특히, 실시간 능동 데이터베이스 환경에서 트랜잭션들에 대한 스케줄링과 동시성 제어 기법에 대한 응용 분야는 많은데 비해 이들 분야에 대한 연구는 미비한 상태이다. 실시간 능동 트랜잭션들을 수행하는 경우 공유된 데이터에 대한 충돌을 해결하기 위해서는 데이터 일관성, 시간제약, 능동성 등을 고려해야 한다. 실시간 능동 데이터베이스 환경에서 트랜잭션은 사용자에 의해서 실행되기보다는 외부 환경변화에 반응한 다른 트랜잭션에 의해서 실행된다. 외부 환경변화에 의해서 실행된 트랜잭션은 외부 환경변화를 감지한 트랜잭션이 실행의 주체가 되기 때문에 이들 트랜잭션은 서로 직렬화가능 순서(serialization order)를 가진다. 실시간 능동 트랜잭션들 사이의 직렬화가능 순서는 능동성에 의해서 직접적으로 결정된다. 따라서, 능동성에 기반 한 직렬화가능 순서를 기반으로 트랜잭션들 사이의 데이터 충돌을 해결해야 한다. 실시간 능동 트랜잭션들 사이의 데이터 충돌을 해결하기 위해서는 시간제약 뿐 아니라 능동성에 의해서 생성된 직렬화가능 순서까지도 고려해야 한다. 본 논문은 실시간 능동 데이터베이스 시스템을 위한 트랜잭션 모델을 제시하고 실시간 능동 트랜잭션들 사이의 데이터 충돌을 해결하기 위한 다중버전 동시성 제어 기법을 제안한다. 본 논문에서 제안한 동시성 제어 기법은 기존에 제안된 실시간 다중버전 동시성 제어 기법인 MVPR [7, 16]을 기반으로 능동성 개념을 적용시켰다.

본 논문의 구성은 2장에서 기존에 연구되었던 실시간 능동 트랜잭션 처리 기법에 대해서 알아보고, 3장에서는 실시간 능동 데이터베이스에서 트랜잭션 처리 모델에 대해서 기술하고, 4장에서는 실시간 능동 트랜잭션을 위한 동시성 제어 기법을 제안한다. 5장에서는 제안한 동시성 제어 기법을 다른 기법과 비교함으로서 성능평가를 한다. 마지막으로 6장에서 결론과 앞으로의 연구방향에 대해서 알아본다.

2. 관련 연구

본 장에서는 기존에 제안되었던 실시간 동시성 제어 기법의 개념을 간략하게 소개하고 실시간 능동 데이터베이스에서 트랜잭션 관리 기법에 대해서 논의한다.

2.1 실시간 동시성 제어 기법

실시간 트랜잭션들에 대한 동시성 제어는 일반 데이터베이스 시스템에서 많이 연구되어 왔던 2단계 잠금 기법이나 낙관적인(optimistic) 방법 등을 기반으로 많은 연구가 진행되었다. 공유되는 데이터를 접근하는 트랜잭션들 간의 데이터 충돌을 해결하기 위하여 트랜잭션의 수행을 관리하는 트랜잭션 관리자는 데이터 충돌에 포함된 트랜잭션들의 수행을 제어함으로써 데이터 충돌을 해결한다. 일반적으로 데이터 충돌을 해결하는 방법은 트랜잭션의 블로킹과 철회 등의 두 가지로 분류할 수 있다. 먼저, 트랜잭션의 블로킹은 데이터 충돌을 유발시킨 연산을 요청한 트랜잭션을 기다리게 하는 방법이다. 또한, 트랜잭션을 철회시키는 방법

은 데이터 충돌에 포함된 임의의 트랜잭션을 철회시키는 것이다.

MVPR(Multi-Version concurrency control with Precedence Relationship)은 실시간 데이터베이스 환경을 위한 다중 버전 동시성 제어 기법이다[7, 16]. 실시간 트랜잭션의 종료시한, 중요도, 실행의 시급 정도 등을 우선 순위(priority)로 표현하여 트랜잭션을 수행하고 데이터 충돌을 해결할 때 사용되는 중요한 요소이다. 실시간 데이터베이스 환경에서는 우선 순위가 높은 트랜잭션을 선호하여 두 트랜잭션들 사이의 데이터 충돌을 해결한다. MVPR은 전통적인 동시성 제어 기법중 하나인 다중버전 2단계 잠금 기법(MV2PL : Multi- Version 2 Phase Locking)[15]에 기반한다. MV2PL은 공유되는 데이터를 접근하기 위해서 잠금 기법을 사용하며 쓰기 연산의 동시성을 향상시키기 위해서 다중 버전으로 데이터를 유지한다. 그러나, MV2PL은 실시간 트랜잭션의 특성인 시간 개념을 고려하지 않기 때문에 실시간 데이터베이스 환경에 적합하지 않다. 트랜잭션 T_1이 쓰고 있는 데이터를 트랜잭션 T_2가 읽기 연산을 요청하는 경우 MV2PL은 트랜잭션 T_2를 트랜잭션 T_1이 쓰기 연산을 종료할 때까지 대기시킨다. 이와 같은 상황은 실시간 데이터베이스 환경에서 중대한 문제를 발생시킬 수 있다. 만일, 트랜잭션 T_2가 T_1 보다 앞선 종료시한을 가진다면 트랜잭션 T_2는 T_1의 쓰기 연산을 기다리는 동안 종료시한을 넘길 수도 있다. 실시간 데이터베이스 환경에서 이와 같은 상황을 우선순위 반전(priority inversion)이라 한다.

동시에 수행되는 트랜잭션의 수가 많지 않을 때 성능이 우수하다고 알려진 낙관적 동시성 제어기법을 실시간 데이터베이스 환경에 적용시킨 기법으로 OCCL[16]이 있다. OCCL에서는 R-Lock과 V-Lock의 두 가지 잠금을 제공하여 R-Lock은 검증 단계(validation phase) 이전에 사용하는 데이터를 위한 잠금으로, V-Lock은 검증 단계에서 기록하는 데이터를 위한 잠금으로 사용한다. 그러나, OCCL에서는 트랜잭션들 사이의 일관성을 보장하기 위해서 검증 단계에서 각 트랜잭션은 임계영역(critical section) 내에서 장기간 수행을 한다. 결과적으로 트랜잭션들 간의 동시성을 감소시키기 때문에 실시간 데이터베이스 시스템의 성능을 저하시키게 된다.

2.2 실시간 능동 트랜잭션 관리 기법

최근에 실시간 능동 데이터베이스 분야에 많은 연구가 진행되고 있다. 실시간 데이터베이스에 대한 대부분의 연구 분야는 트랜잭션 스케줄링, 동시성 제어, 트랜잭션 모델링 등에 집중되어 있다. 또한, 능동 데이터베이스 연구 분야의 경우 능동 규칙 선택, 평가, 능동 규칙에 근거한 확장된 트랜잭션 모델링 등을 위주로 연구가 진행되어왔다[9-13]. 그러나, 실시간 능동 데이터베이스 시스템의 효용성에 비해서 실시간 능동 트랜잭션 관리 기법에 대한 연구는 많지 않았다.

기존의 실시간 트랜잭션을 위한 동시성 제어 기법[3, 8]을 기반으로 실시간 능동 트랜잭션을 위한 동시성 제어 기법을 제안한 연구가 있었다[1]. [1]의 연구에서는 기존의 실시간 낙관적 동시성제어기법인 OCCL[16]을 기반으로 한 OCC-APFS(OCC Adaptative Priority Fan-out Sum)이라는 동시성 제어 기법을 고안하였다. OCC-APFS는 데이터 충돌을 해결하기 위해서 트랜잭션들 사이의 능동성 관계와 우선순위를 고려하였다. 트랜잭션의 철회 가능성을 CPI (Concurrency Priority Index)라는 척도로 표현하여 데이터 충돌을 해결하기 위한 중요한 요소로 활용한다. CPI는 능동성 관계에 포함된 트랜잭션들의 수와 우선순위 등을 포함하기 때문에 트랜잭션들이 불필요하게 철회되거나 시스템 자원을 낭비하지 않도록 능동성 관계를 고려하여 데이터 충돌을 해결하도록 한다. OCC-APFS는 데이터 충돌을 해결하기 위해서 서로 능동성 관계를 가지고 있지 않은 트랜잭션들 사이의 CPI를 사용하였다. 그러나, 능동성 관계를 가지고 있는 트랜잭션들 사이의 데이터 충돌을 해결하는 경우 낮은 우선순위의 트랜잭션을 불필요하게 철회시킬 필요가 없을 경우도 발생한다.

3. 실시간 능동 트랜잭션 모델

본 장은 실시간 능동 트랜잭션 관리를 위해서 필요한 트랜잭션 모델에 대해서 기술한다. 실시간 능동 트랜잭션들은 외부 환경변화에 의해서 다른 트랜잭션들을 실행시키며 이 트랜잭션들 사이에는 여러 가지 능동성 관계가 성립한다.

3.1 능동 규칙

능동 데이터베이스 시스템에서 능동 또는 ECA(Even-Condition-Action) 규칙은 중요한 개념이다. ECA 규칙은 사건(Event), 조건(Condition), 행위(Action) 등의 세 부분으로 구성되며 주어진 사건이 발생한 경우 조건을 판단하여 조건이 만족되는 경우에만 행위를 실행시킨다. 예를 들어, 외부 온도를 감지하여 적절한 행동을 하는 용광로 관리 시스템을 가정하자. 외부 온도를 30초마다 감지하여 시스템의 데이터베이스에 삽입하는 사건이 발생하는 경우 관리 시스템은 감지된 온도가 주어진 조건을 만족하는지를 확인한다. 만일, 정해진 온도 이하 또는 이상의 값을 가진다면 적절한 행동을 하기 위한 작업을 실행시켜야 한다. 이와 같은 상황에서 외부 온도의 범위를 확인하여 적절한 작업을 실행시키는 트랜잭션을 triggering 트랜잭션이라 하며, triggering 트랜잭션이 실행시킨 트랜잭션을 triggered 트랜잭션이라 한다.

3.2 트랜잭션들 사이의 종속관계

능동 데이터베이스에서 ECA 규칙에 의해서 생성된 트랜

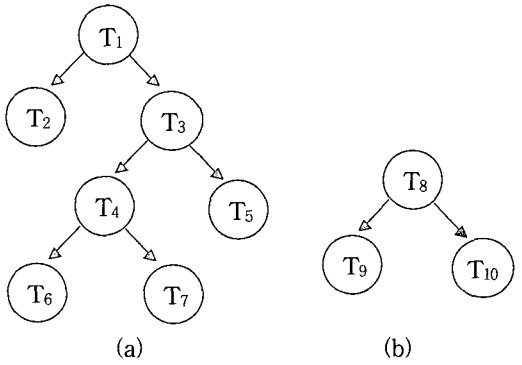
잭션들 사이에는 triggering 관계가 성립하며 능동 데이터베이스의 데이터 일관성(consistency)을 만족시키기 위한 중요한 개념이 된다[1]. triggering 트랜잭션은 주어진 조건을 판단하여 참(true)으로 판명되면 triggered 트랜잭션을 실행시킨다. 즉, triggered 트랜잭션과 triggering 트랜잭션 사이에는 일정한 종속 관계가 발생하게 된다. 트랜잭션 T_i가 트랜잭션 T_j를 ECA 규칙에 의해서 실행시켰다고 가정하자. 즉, T_i는 triggering 트랜잭션으로, T_j는 triggered 트랜잭션으로 볼 수 있다. T_i와 T_j가 병렬로 수행된다면 두 트랜잭션들 중에서 한 트랜잭션이 철회하는 경우 다른 트랜잭션의 수행에 영향을 줄 수도 있다. triggering 트랜잭션인 T_i가 철회됐다고 가정하자. T_j는 T_i가 실행시켰기 때문에 생성된 트랜잭션이다. 따라서, triggering 트랜잭션인 T_i가 철회됨으로서 더 이상 T_j는 수행을 계속할 정당성이 없게된다. 결과적으로, T_i의 철회는 T_j의 철회를 촉발시키게 된다. 이와 같은 종속 관계를 철회 종속성(abort dependency)라고 한다. 또한, 철회 종속성에 의해서 불필요한 트랜잭션 철회를 방지하기 위해서 triggered 트랜잭션은 triggering 트랜잭션이 종료된 후에 종료되어야 하는 종속 관계를 종료 종속성(commit dependency)라고 한다. 능동 데이터베이스에서 트랜잭션들 사이의 동시성 제어 기법을 고안하기 위해서는 이와 같은 triggering 관계를 기반으로 철회/종료 종속성을 고려하여야 한다.

4. 실시간 능동 동시성 제어 기법

본 장에서는 기존에 제안된 실시간 다중버전 동시성 제어 기법인 MVPR에 대하여 살펴보고 실시간 능동 트랜잭션을 위한 다중버전 동시성 제어 기법을 제안한다.

4.1 triggering 관계와 선행관계

ECA 규칙에 의해서 트랜잭션들 사이에는 triggering 관계가 존재하며 철회 또는 종료 종속성이 성립한다. 본 논문에서 제안한 동시성 제어 기법은 실시간 다중버전 동시성 제어 기법인 MVPR[7, 16]을 기반으로 한다. 트랜잭션들 사이의 triggering 관계는 트리 구조로 표현될 수 있으며 이를 triggering 트리라고 한다. triggering 트리에서 각 노드는 트랜잭션을 나타내며 자식 노드는 부모 노드에 의해서 실행된 triggered 트랜잭션을 나타낸다. (그림 1)은 임의의 triggering 트리 구조를 나타낸다. 트랜잭션 T_1이 트랜잭션들 T_2와 T_3를 실행시킨 것을 알 수 있으며, 또한 트랜잭션 T_3는 T_4와 T_5 등의 트랜잭션을 실행시켰음을 알 수 있다. 이와 같은 triggering 트리 구조에서 만일 트랜잭션 T_1이 철회된다면 철회 종속성에 의해서 T_1의 하위 트리의 트랜잭션들인 T_2, T_3, T_4, T_5, T_6, T_7 등도 함께 철회되어야 한다.



(그림 1) triggering 트리의 예

트랜잭션들 사이의 triggering 관계를 유지하기 위해서 *Triggering_Set* [T_i]와 *Triggered_Set* [T_i] 등의 자료구조를 사용한다. *Triggering_Set* [T_i]와 *Triggered_ Set* [T_i]는 각각 T_i를 실행시킨 트랜잭션들과 T_i가 실행시킨 트랜잭션들을 나타낸다. triggering 관계가 트리 구조로 표현되기 때문에 *Triggering_Set* [T_i]는 오직 한 개의 트랜잭션만 포함할 수 있다. 또한, *Triggering_ Set**와 *Triggered_Set**는 각각 *Triggering_Set*와 *Triggered_ Set*에 대한 transitive closure에 해당한다. 예를 들어, *Triggering_Set* [T_2] = {T_1}와 *Trigg ered _Set* [T_1] = {T_2, T_3}가 성립한다. 또한, *Triggering_Set**[T_6] = {T_1, T_3, T_4}와 *Triggered_Set**[T_1] = {T_2, T_3, T_4, T_5, T_6, T_7} 등도 성립함을 알 수 있다. *Triggered_Set**[T_1]은 트랜잭션 T_1이 직접 또는 간접적으로 실행시킨 모든 triggered 트랜잭션들을 나타내며, *Triggering_Set**[T_6]는 트랜잭션 T_6를 실행시키는데 직접 또는 간접적으로 실행시킨 모든 트랜잭션들을 나타낸다. 하나의 triggering 트리를 구성하는 노드에 해당하는 트랜잭션들 사이에는 triggering 관계가 성립한다. 종료 종속성으로 인해서 triggering 관계는 트랜잭션들 사이의 선행관계를 강요한다. 예를 들어, T_1이 T_2을 ECA 규칙에 의해서 수행시켰다면 종료 종속성에 의해서 T_2는 T_1이 종료한 후에야 실행을 완료할 수 있다. 즉, 직렬성 순서에서 T_1은 T_2를 앞서게 된다. triggering 관계는 선행관계와 유사하게 비대칭(anti-symmetric) 그리고 이행(transitive) 관계의 특성을 가진다. 트랜잭션 T_i가 루트인 triggering 트리에서 트랜잭션 T_j가 하위 트리의 한 노드라고 했을 때 두 트랜잭션들 사이의 관계를 $T_i \ll T_j$로서 표현한다. 즉, $T_j \in$ *Triggered_Set** [T_i]가 성립한다. (그림 1)에서 $T_1 \ll T_3$, $T_1 \ll T_6$, $T_1 \ll T_4$ 등의 관계가 성립한다.

4.2 실시간 다중버전 동시성 제어 기법(MVPR)

MVPR은 일반적인 동시성 제어 기법인 다중버전 2단계 잠금 기법(MV2PL)[15]을 실시간 트랜잭션 처리에 적합하게 적용한 동시성 제어 기법이다. MV2PL은 쓰기 연산의 동시성을 증가시키기 위해서 각 데이터에 대해서 다중버전을 생성하며 직렬화가능 스케줄을 위해서 2단계 잠금 기법

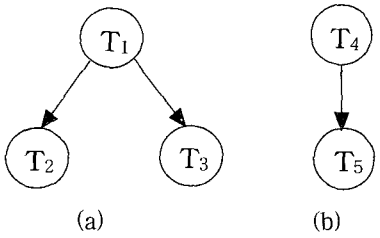
을 사용한다. MVPR은 다중버전을 사용하여 트랜잭션의 동시성을 증가시키는 것 외에 트랜잭션들 사이의 수행순서를 동적으로 유지하여 불필요한 트랜잭션 철회를 감소시켰다. 또한, 데이터 충돌에 포함된 트랜잭션들의 우선순위를 고려하여 높은 우선순위의 트랜잭션을 선호하여 데이터 충돌을 해결한다.

트랜잭션들 사이의 수행순서는 선행관계(precedence relationship)로 표현될 수 있으며 처음으로 데이터 충돌이 발생한 시점에 생성된다. 선행관계는 *Before_Set* [T_i]과 *After_Set* [T_i]이라는 명칭으로 표현되며 각각 수행순서에서 트랜잭션 T_i를 직접적으로 앞서는 트랜잭션과 뒤에 오는 트랜잭션을 의미한다. 또한, *Before_ Set**[T_i]와 *After_Set**[T_i]는 각각 트랜잭션 T_i 보다 직렬화가능 순서가 앞서는 트랜잭션들과 뒤에 오는 트랜잭션들을 의미한다. 예를 들어, 직렬화가능 순서에서 T_i가 T_j 보다 앞서는 것을 $T_1 < T_2$로 표현했을 때 $T_i \in$ *Before_Set**[T_j]와 $T_j \in$ *After_Set**[T_i]가 동시에 성립하게 된다. 또한, 종료 종속성으로 인해서 *Triggered_Set**[T_i]가 T_j를 포함한다면 $T_j \in$ *After_Set**[T_i]가 성립해야 함을 알 수 있다. 역으로, *Triggering_Set**[T_i]가 T_j를 포함한다면 $T_j \in$ *Before_Set**[T_i]가 성립해야 한다. 두 트랜잭션 T_i와 T_j 사이에 선행관계가 성립하지 않은 상태에서 서로 호환하지 않는 연산을 수행하는 순간 T_i와 T_j 사이에 선행관계가 생성된다. 그 후에 두 트랜잭션들의 우선순위를 사용하여 <표 2>와 <표 3>에서 제시한 호환성 표와 유사한 방식으로 두 트랜잭션들 사이의 데이터 충돌을 해결하게 된다. 우선순위를 고려하여 데이터 충돌을 해결하기 때문에 대부분의 데이터 충돌 상황에서 높은 우선순위의 트랜잭션이 낮은 우선순위의 트랜잭션 보다 선호되어 스케줄 된다. 그러나, 낮은 우선순위의 트랜잭션이 철회되어야 하는 상황에서 만일 낮은 우선순위의 트랜잭션이 종료시점에 근접하였다면 높은 우선순위의 트랜잭션을 대기시켜 낮은 우선순위의 트랜잭션이 계속 수행을 하도록 허용한다. 물론, 이와 같은 상황에서 실시간 환경에서 중요한 문제인 우선순위 반전(priority inversion)이 발생할 수 있다. 그러나, 이 경우 높은 우선순위의 트랜잭션 대기 시간이 길지 않기 때문에 우선순위 반전이 큰 문제가 되지는 않는다.

4.3 실시간 능동 데이터베이스 환경에서 MVPR의 문제점

MVPR은 실시간 데이터베이스 환경에 적합하도록 고안되었기 때문에 트랜잭션들 사이에 선행관계 외에 능동성에 근거한 triggering 관계가 성립하는 실시간 능동 데이터베이스 환경에는 적합하지 않다는 문제점이 있다. MVPR에서는 데이터 충돌이 발생해야만 선행관계가 성립하게 되는데 종료 종속성에 의해서 미리 결정된 선행관계를 무시한다면 트랜잭션들 사이의 데이터 충돌을 효율적으로 해결할 수는 없을 것이다. 또한, 낮은 우선순위의 트랜잭션을 철회하는 상황에서 해당 트랜잭션만의 정보를 이용해서 데이터 충돌을 해결한다면 철회 종속성으로 인해서 필요 이상의 트랜잭션들이 연쇄 철회(cascading abort)될 것이다. 본 절에서는 MVPR이 실시간 능동 데이터베이스 환경에 직접 적용되는 경우에 발생하는 문제점을 지적하여 실시간 능동 트랜잭션을 위한 동시성 제어기법을 제안하고자 한다.

triggering 관계에 의해서 트랜잭션들 사이의 데이터 충돌은 크게 intra와 inter triggering 충돌 등으로 분류된다. 두 트랜잭션 T_i와 T_j가 서로 데이터 충돌이 발생했을 때 intra triggering 충돌은 Ti와 Tj가 같은 triggering 트리에 포함되는 경우이다. 즉, $T_i \in$ *Triggering_Set**[T_j]이거나 $T_j \in$ *Triggered_Set**[T_i]가 성립한다. 또한 inter triggering 충돌은 T_i와 T_j가 서로 다른 triggering 트리에 포함되는 경우에 해당한다. 따라서 두 트랜잭션들 사이에는 어떤 triggering 관계도 성립하지 않는다. 제안하는 동시성 제어 기법은 트랜잭션들 사이의 충돌의 종류에 따라서 서로 다른 해결 기법을 사용한다.



(그림 2) triggering 트리의 예

(그림 2)는 MVPR의 실시간 능동 데이터베이스 환경에서의 문제점을 기술하기 위한 다음 예에서 사용할 triggering 트리의 예이다. 다음 예들은 MVPR에 의해서 데이터 충돌이 해결되며, $O_i[a]$는 O는 읽기(R), 쓰기(W), 검증(C) 등의 연산을 트랜잭션 i가 데이터 항목 a에 대해서 수행함을 의미한다. 각 세 가지 연산을 실행하기 위해서는 각 연산에 대응하는 잠금(lock)인 읽기, 쓰기, 검증 등을 요청해야 한다.

[예 1] (그림 2) (a)의 triggering 트리에서 두 트랜잭션 T_1와 T_2 사이에 트랜잭션 T_2가 T_1 보다 높은 우선순위를 가진다고 하자. 그리고, 다음과 같이 각 트랜잭션들이 실행된다고 하자.

$T_1 = R_1[a]; W_1[a]; C_1[a]$
$T_2 = R_2[a]; W_2[a]; C_2[a]$
$H = R_2[a]; R_1[a]; W_1[a]; W_2[a]; C_1[a]; C_2[a]$

(그림 2) (a)의 triggering 트리에 의해서 트랜잭션 T_1이 T_2를 실행시켰음을 알 수 있다. 수행 스케줄 H에서 트랜잭션 T_1이 $W_1[a]$을 요청했을 때 T_2가 이미 데이터 a에 대해서 읽기 연산을 했으므로 T_1의 쓰기 연산은 T_2의 읽기 연

산보다 직렬화가능 순서에서 뒤에오게 된다. 따라서, MVPR 스케줄러는 $T_2 < T_1$의 직렬화가능 순서를 생성한 후 T_1에게 $W_1[a]$을 허용한다. T_2가 $W_2[a]$를 요청하는 경우 T_1이 이미 같은 데이터를 읽었기 때문에 $T_1 < T_2$의 순환적인 직렬화가능 순서를 생성한다. 결과적으로, MVPR 스케줄러는 데이터 일관성을 유지하기 위하여 낮은 우선순위의 트랜잭션인 T_1을 철회시켜야 한다. 그러나, T_1의 철회는 철회 종속성에 의해서 T_1이 실행시킨 T_2도 철회되어야 한다. □

[예 1]에서 트랜잭션 T_1과 T_2 사이의 데이터 충돌은 intra triggering 충돌에 해당한다. T_1과 T_2 사이에는 $T_1 \ll T_2$라는 triggering 관계를 가지기 때문에 T_1과 T_2 사이에는 종료 종속성에 의해서 직렬화가능 순서인 $T_1 < T_2$ 역시 성립한다. 따라서, 같은 triggering 트리 내의 트랜잭션들 사이의 직렬화가능 순서는 triggering 관계에 의해서 생성되며 intra triggering 충돌을 해결하는 경우 고려되어야 할 요인이다. [예 1]에서처럼 T_1과 T_2 사이의 직렬화가능 순서와 우선순위만을 고려하여 데이터 충돌을 해결한다면 T_1의 철회는 철회 종속성에 의해서 $Triggered_Set^*[T_1]$에 포함된 모든 트랜잭션들도 함께 철회시켜야 한다. 따라서, T_1이 우선순위가 낮다고 하더라도 T_1이 실행시킨 트랜잭션에 의해서 철회되지 못하도록 해야 한다.

[예 2] 이제는 MVPR이 triggering 관계를 고려하여 데이터 충돌을 해결한다고 하자. [예 1]의 상황에서 T_1과 T_2 사이의 수행 스케줄이 다음과 같다고 하자.

$$H = R_1[a];W_1[a];R_2[a];C_1[a];W_2[a];C_2[a]$$

이제 MVPR 스케줄러는 triggering 관계를 고려하여 트랜잭션 T_1과 T_2의 직렬성 순서를 생성한다. T_2가 $R_2[a]$를 요청할 때 $T_1 < T_2$의 직렬화가능 순서에 의해서 T_2의 실행을 T_1이 데이터 항목 a에 대한 잠금을 해제할 때까지 중지시킨다. □

[예 2]에서 트랜잭션 T_2는 T_1이 직렬화가능 순서에서 앞서기 때문에 T_1이 데이터 항목 a에 대한 검증연산(certify)을 완료하여 새로운 데이터를 생성할 때까지는 수행이 중지된다. 그러나, T_2가 $R_2[a]$를 요청할 때 T_1이 검증연산을 하자마자 읽기 연산을 허용하여 T_1이 생성한 데이터를 읽게 해도 데이터 일관성을 유지할 수 있다. 종료 종속성에 의해서 T_2는 T_1이 종료를 해야 종료 할 수 있기 때문에 T_2는 T_1이 종료연산을 실행하기 전에는 종료할 수 없다. 따라서, T_2가 데이터 항목 a에 대해서 읽기 연산을 실행하더라도 검증단계에서 T_1의 종료를 기다려야 한다. 또한, 철회 종속성에 의해서 T_2가 데이터 항목 a를 읽은 후 T_1이 철회된다면 T_2 역시 철회되어야 한다. 결과적으로 T_1과 T_2 사이의 직렬화가능 순서가 $T_1 < T_2$가 성립할지라도 T_2의 $R_2[a]$는 T_1의 검증연산을 대기할 필요없이 바로 실행될 수 있다. 제안하는 동시성 제어 기법에서 intra triggering 충돌이 발생하는 경우 직렬화가능 순서에 의한 수행 대기 상황을 최소화할 수 있으므로 트랜잭션의 동시성을 향상시킬 수 있다.

[예 3] MVPR 스케줄러가 triggering 관계를 고려하지 않는다고 하자. (그림 1)와 같은 triggering 트리에 의한 triggering 관계를 가지는 세 트랜잭션 T_1, T_4, T_5가 있다. T_1과 T_4가 각각 가장 낮은 우선순위와 가장 높은 우선순위를 가진다고 가정할 때 다음과 같은 수행 스케줄 H의 일부분을 보자.

$$T_1 = R_1[a];R_1[b];W_1[b];C_1[b]$$
$$T_4 = R_4[a];W_4[a];C_4[a]$$
$$T_5 = R_5[b]$$
$$H = R_1[a];R_4[a];R_1[b];W_4[a];W_1[b];R_5[b]$$

MVPR에서 데이터에 쓰기 연산은 실제 데이터베이스에 반영하는 것이 아니라 지역 공간에 임시로 저장함을 의미한다. 따라서, T_4가 $W_4[a]$를 요청 시에 T_1이 이미 T_4가 데이터 항목 a에 대한 검증을 하지 않은 데이터를 읽었기 때문에 $T_1 < T_4$의 직렬화가능 순서가 성립한다. 또한, T_5가 $R_5[b]$를 요청 시에 $T_5 < T_1$의 직렬화가능 순서가 성립한다. 그러나, triggering 트리에 의하면 T_4가 T_5를 실행시켰기 때문에 종료 종속성에 의해서 $T_4 < T_5$의 직렬화가능 순서가 성립한다. 결과적으로 $T_1 < T_4 < T_5 < T_1$의 직렬화가능 순서가 유추될 수 있으며 이는 교착상태(deadlock)를 유발시킨다. 제안하는 동시성 제어 기법에서는 이와 같은 inter triggering 충돌을 해결하기 위해서 $R_5[b]$를 허용하지 않는다. 즉, 교착상태를 미연에 방지함으로서 불필요한 트랜잭션 철회를 방지할 수 있다. □

[예 3]에서 MVPR 스케줄러가 T_5의 $R_5[b]$를 허용한다면 $T_1 < T_4 < T_5 < T_1$의 직렬화가능 순서를 생성할 수 있기 때문에 데이터 충돌을 발생시킨 낮은 우선순위의 트랜잭션인 T_1을 철회시킨다. 결과적으로 MVPR 스케줄러는 데이터 일관성을 유지하도록 직렬화가능 순서를 유지시킬 수 있다. 그러나, 이와 같은 상황에서 트랜잭션 T_1의 철회는 철회 종속성에 의해서 T_2와 T_3의 철회를 유발시키며 이는 시스템 자원을 낭비하는 결과가 된다. 제안하는 동시성 제어 기법에서는 inter triggering 충돌의 경우 트랜잭션들 사이의 triggering 트리 구조를 고려하여 만일 철회될 트랜잭션이 우선순위가 낮더라도 철회 종속성에 의해서 철회될 트랜잭션의 수가 많다면 낮은 우선순위의 트랜잭션이라 하더라도 일방적으로 철회되도록 하지는 않는다.

4.4 잠금과 종료 방법

제안하는 동시성 제어 기법은 MVPR을 기반으로 능동성 개념을 고려하여 데이터 충돌을 해결하도록 했다. 따라서, 쓰기 연산을 하더라도 다른 트랜잭션은 해당 데이터를 사

용할 수 없다. 쓰기 연산 후 검증연산을 수행한 후 최종적으로 디스크에 반영되어야 다른 트랜잭션이 데이터를 사용할 수 있다. RTA-MVPR[1]은 트랜잭션이 데이터에 대한 연산을 요청했을 때 해당 데이터에 대해서 기존의 트랜잭션과 데이터 충돌이 발생하는지를 판단한다. 다음은 임의의 트랜잭션 t가 데이터 x에 대해서 잠금 l을 요청한 경우에 수행되는 프로시주어인 LOCK을 기술한다.

LOCK(t, l, x)
1. 만일 다른 트랜잭션 t'가 x와 충돌하는 잠금 을 소유한다면,
   1.1 필요한 경우 t와 t'사이의 선행관계를 갱신한다.
   1.2 데이터 충돌의 종류에 따라서 적당한 잠금 호환성 표를 참조하여 t와 t' 사이의 데이터 충돌을 해결한다.
2. 1.의 조건이 만족되지 않은 경우 t에게 l를 허용한다.

RTA-MVPR에서는 데이터 충돌에 포함된 트랜잭션들 사이의 triggering 관계를 참조하여 데이터 충돌의 종류에 따라서 데이터 충돌을 해결한다. LOCK 프로시주어의 1.2 단계에서 intra triggering 충돌인지 아니면 inter triggering 충돌인지에 따라서 적당한 잠금 호환성 표를 참조한다. 다음절에서 intra triggering 충돌과 inter triggering 충돌의 해결을 위한 잠금 호환성 표를 제안한다.

MVPR에서처럼 제안하는 동시성 제어 기법은 각 트랜잭션이 종료하는 시점에 선행관계 및 triggering 관계를 사용하여 대기 상태에 들어갈 필요가 있다. 다음 프로시주어 COMMIT는 임의의 트랜잭션 T가 종료하는 시점에 수행하는 과정이다.

COMMIT(T)
1. 쓰기 잠금을 모두 검증 잠금으로 변환한다.
2. Triggering_Set*[T]와 Before_Set*[T]가 공집합일 때까지 대기한다.

임계 영역의 시작
3. 새로 생성된 모든 데이터를 검증한다.
4. 모든 읽기 잠금과 검증 잠금을 해제한다.
5. Triggering_Set[k] : = Triggering_Set[k] - {T}, for all k ∈ Triggered_Set*[T]
6. Before_Set[k] : = Before_Set[k] - {T}, for all k ∈ After_Set*[T]

임계 영역의 끝

2단계와 5단계에 의해서 COMMIT 프로시주어는 triggered 트랜잭션은 triggering 트랜잭션 이전에 종료할 수 없음을 보장할 수 있다. RTA-MVPR은 데이터 충돌이 발생한 경우 intra triggering 충돌인지 inter triggering 충돌인지를 판단하고 각 데이터 충돌에 따라서 적절한 잠금 호환성 표를 참조하여 데이터 충돌을 해결한다. 각 트랜잭션의 종료 시점에 COMMIT 프로시주어에 의해서 triggering 관계를 만족하도록 트랜잭션을 종료시킨다. 다음절에서 데이터 충돌에 따른 해결 방법을 알아보기로 한다.

4.5 데이터 충돌 해결 기법

본 절에서 intra triggering 충돌과 inter triggering 충돌을 해결하기 위한 잠금 호환성 표를 알아본다. 편의상, intra triggering 충돌과 inter triggering 충돌을 해결하기 위한 잠금 호환성 표를 각각 intra 잠금 호환성 표와 inter 잠금 호환성 표로 언급하기로 한다.

4.5.1 intra triggering 충돌 해결 기법

MVPR에서는 모든 트랜잭션들은 반드시 검증된 데이터만을 읽도록 보장한다. 따라서, 아직 검증되지 않은 데이터를 읽기 위해서는 연관된 트랜잭션이 데이터 항목에 대한 검증 연산을 실행하고 검증 잠금을 해제할 때까지 다른 트랜잭션들의 읽기 연산은 보류되어야 한다. 또한, 트랜잭션 스케줄링의 엄정성(strictness)을 만족시키기 위해서 직렬화 가능 순서에서 앞서는 트랜잭션이 검증한 데이터는 해당 트랜잭션이 종료해야 다른 트랜잭션이 읽을 수 있도록 하였다. 결과적으로, MVPR은 데이터를 생성한 트랜잭션이 철회됨으로서 다른 트랜잭션도 연쇄적으로 철회되는 현상을 방지하였다. 또한, MVPR에서 대부분의 경우에 우선순위를 고려하여 높은 우선순위의 트랜잭션은 낮은 우선순위의 트랜잭션을 기다리지 않도록 고안되었다. RTA-MVPR에서 triggering 관계를 가지고 있는 트랜잭션들 사이의 데이터 충돌을 해결하는 방법은 엄정성에 대해서 MVPR과 다르다. MVPR에서는 엄정성을 만족시켜야 하기 때문에 어떤 트랜잭션도 검증되지 않은 데이터를 사용할 수 없다. [예 2]에서와 같이 triggering 트랜잭션이 쓰기 잠금을 소유한 상태에서 triggered 트랜잭션이 동일한 데이터를 읽으려는 경우 MVPR에서는 우선순위에 따라서 triggered 트랜잭션이 철회되거나 triggered 트랜잭션이 수행을 대기해야 한다. 즉, triggered 트랜잭션은 해당 데이터가 triggering 트랜잭션이 검증한 후에 읽도록 하여 엄정한 스케줄링을 보장한다. 그러나, RTA-MVPR에서 이와 같은 상황에서 triggered 트랜잭션은 우선순위와 관계없이 철회되거나 수행이 중지될 필요가 없다. 이것은 종료 또는 철회 종속성에 의해서 triggered 트랜잭션의 운명은 triggering 트랜잭션에 의해서 좌우되기 때문에 가능하다.

먼저, triggering 트랜잭션이 철회된 경우를 보자. triggering 트랜잭션이 생성한 데이터를 triggered 트랜잭션이 읽도록 허용된 경우 triggered 트랜잭션은 수행을 계속할 수 있다. 만일, triggering 트랜잭션이 철회된다면 trig-

1) 본 절에서 제안하는 실시간 능동 동시성 제어 기법을 RTA-MVPR(Real-Time Active MVPR)로써 언급하기로 한다.

gering 트랜잭션이 생성한 데이터를 읽은 triggered 트랜잭션도 철회되어야 한다. 이와 같은 상황은 엄정한 스케줄링을 하지 않았기 때문이다. 그러나, 실시간 능동 데이터베이스 환경에서 종료 종속성에 의해서 triggered 트랜잭션은 triggering 트랜잭션의 종료 이전에 종료할 수 없다. 즉, COMMIT 프로시주어의 2단계와 5단계에 의해서 triggered 트랜잭션은 triggering 트랜잭션이 종료할 때까지 종료단계에서 대기해야 한다. 또한, 철회 종속성에 의해서 triggered 트랜잭션은 triggering 트랜잭션이 철회되면 함께 철회되어야 한다. 결과적으로, triggered 트랜잭션이 triggering 트랜잭션이 생성한 데이터를 검증되기 전에 읽더라도 데이터 일관성에는 아무 영향도 주지 않는다. 이번에는 triggering 트랜잭션이 종료된 경우를 보자. 종료 종속성에 의해서 triggering 트랜잭션이 종료된 후 triggered 트랜잭션이 종료될 수 있으므로 역시 데이터 일관성을 만족시키게 된다. RTA-MVPR은 이와 같이 MVPR에서는 철회나 수행 대기를 발생시킬 수 있는 데이터 충돌을 트랜잭션들의 수행을 방해하지 않고 계속 진행시킬 수 있다. 결과적으로, 동시에 수행할 수 있는 트랜잭션들의 수를 증가시킴으로서 종료시한을 만족시킬 수 있는 트랜잭션들의 수를 증가시킬 수 있다.

<표 1>는 데이터를 요청하는 트랜잭션 T_R과 데이터를 소유한 트랜잭션 T_O 사이의 잠금 호환성 표를 나타내며, T_1이 T_2를 실행시키는 triggering 관계는 $T_1 \ll T_2$로 표시한다. intra 잠금 호환성 표는 잠금을 요청한 트랜잭션과 잠금을 소유하고 있는 트랜잭션들 사이의 triggering 관계에 의해서 크게 두 부분으로 구성된다. <표 1>에서 $T_O \ll T_R$의 triggering 관계일 때 잠금을 요청하는 트랜잭션 T_R은 triggering 트랜잭션에 해당하는 T_O가 이미 잠금을 소유 중이므로 대부분의 경우 그대로 잠금이 허용되거나 수행을 대기하게 됨을 알 수 있다. 이와는 반대로 $T_R \ll T_O$의 경우 잠금을 요청하는 트랜잭션 T_R가 잠금을 소유한 트랜잭션 T_O를 실행시켰기 때문에 직렬화가능 순서에서 $T_R < T_O$의 관계가 발생한다. 결과적으로 직렬화가능 순서의 역순으로 T_R가 잠금을 요청하기 때문에 직렬화가능 순서를 유지시키기 위해서 T_R는 철회될 확률이 높게된다. intra triggering 충돌을 해결하는 경우 트랜잭션의 우선순위는 고려할 필요가 없다. triggering 관계에 의해서 트랜잭션 사이의 선행관계가 생성되고 철회 종속성 또는 종료 종속성이 충돌 해결 정책에 밀접한 영향을 주게된다.

〈표 1〉 intra triggering 잠금 호환성 표

요청자(T_R) / (T_O)소유자	$T_O \ll T_R$			$T_R \ll T_O$		
	읽기	쓰기	검증	읽기	쓰기	검증
읽기 (Read)	Y	Y	T_R 대기	Y	T_O 철회	T_O 철회
쓰기 (Write)	T_R 대기	Y	T_R 대기	Y	Y	Y
검증 (Certify)	Y	Y	T_R 대기	T_O 철회	T_O 철회	T_O 철회

4.5.2 inter triggering 충돌 해결 기법

inter triggering 충돌은 트랜잭션들 T_i와 T_j 사이에 어떤 triggering 관계도 성립하지 않는 경우에 해당한다. 즉, $T_i \ll T_j$ 나 $T_j \ll T_i$ 어떤 관계도 성립하지 않음을 의미한다. inter triggering 충돌을 해결하기 위한 잠금 호환성 표는 사실상 MVPR의 잠금 호환성 표와 유사하다. 따라서, 데이터 충돌을 해결하는 경우 트랜잭션들 사이의 우선순위와 선행관계가 고려되어야 한다. 그러나, RTA-MVPR에서는 MVPR의 데이터 해결 방법에 triggering 관계도 고려해야 하는 차이점을 가지고 있다. (그림 1)의 triggering 트리 구조에서 트랜잭션 T_3가 호환하지 않는 연산을 요청하여 T_9와 데이터 충돌을 발생시켰다고 하자. T_3와 T_9는 각각 서로 다른 triggering 트리에 포함되어 있으므로 T_3와 T_9 사이에는 inter triggering 충돌이 발생한 것이다. MVPR에서 이 데이터 충돌을 해결하는 경우 우선순위와 선행관계에 의해서 T_3가 철회되어야 한다면 철회 종속성에 의해서 트랜잭션 T_3가 실행시킨 모든 triggered 트랜잭션들인 T_4, T_5, T_6, T_7까지도 함께 철회되어야 한다. 이와 같이 데이터 충돌을 해결하는 경우 트랜잭션 T_3의 철회는 연쇄적인 철회를 발생시키기 때문에 많은 시스템 자원을 낭비하는 문제점을 가지고 있다. 따라서, T_3와 T_9 사이의 데이터 충돌을 해결하는 경우 T_3와 T_9 만의 정보뿐 아니라 각 트랜잭션이 실행시킨 triggered 트랜잭션들에 대한 정보까지도 고려해야 한다.

RTA-MVPR에서는 triggering 트리에 대해서 그룹 우선순위인 GP(Group Priority)라는 개념을 사용하여 triggering 트리 내의 트랜잭션들의 우선순위를 고려한다. 트랜잭션 T의 GP는 T의 하부 트리에 포함된 모든 트랜잭션의 우선순위의 합을 의미한다. 예를 들어, (그림 1)에서 T_3, T_4, T_5, T_6, T_7의 우선순위를 각각 20, 30, 40, 50, 60이라 했을 때 GP(T_3)는 이들 트랜잭션의 우선순위 합인 300이 된다. Inter triggering 충돌을 해결하기 위해서는 그룹 우선순위를 고려해야 한다. RTA- MVPR은 트랜잭션들 사이의 우선순위를 triggering 관계와 연계하여 고려하게 된다. 그룹 우선순위의 개념에 의해서 두 트랜잭션이 충돌을 하는 경우 각 트랜잭션만의 우선순위가 아닌 각 트랜잭션이 실행시킨 트랜잭션의 우선순위까지도 고려한 GP 값을 사용하여 데이터 충돌을 해결한다. RTA-MVPR에서는 MVPR에서와 같이 데이터 충돌에 포함된 트랜잭션들만의 수행 상태를 참조할 수 없다. 데이터 충돌에 포함된 트랜잭션이 실행시킨 triggered 트랜잭션들의 수행 상태까지도 고려해야 트랜잭션 철회로 인한 시스템 자원의 낭비 문제를 해결할 수 있다.

(그림 1)에서 트랜잭션 T_3가 데이터 항목 a에 대해서 쓰기 잠금을 소유하고 있고 트랜잭션 T_9이 같은 데이터에 대해 읽기 잠금을 요청한다고 하자. 두 트랜잭션 사이의 직렬화가능 순서는 $T_3 < T_9$이고 T_3가 T_9 보다 낮은 우선순위를

가진다. MVPR 스케줄러는 T_3가 종료단계에 있다면 T_9은 T_3가 종료할 때까지 대lbs기해야 한다. 그러나, T_3가 종료단계에 있지 않다고 한다면 T_3는 철회되어 높은 우선순위의 트랜잭션인 T_{9a}이 읽기 잠금을 허용받을 수 있게 된다. 그러나, RTA- MVPR은 종료단계에 있지 않은 낮은 우선순위의 트랜잭션인 T_3를 무조건 철회시킬 수 없다. 만일, triggering 트리에서 T_3의 하위 트리에 포함된 트랜잭션들 중 많은 수가 이미 종료단계에 있다면 T_3의 철회로 인하여 T_3가 실행시킨 모든 트랜잭션도 철회되어야 한다. 따라서, 이와 같은 경우에 T_3와 T_3가 실행시킨 triggered 트랜잭션의 수행상태도 고려하여 T_3를 철회시키지 않는 게 좋을 것이다.

트랜잭션의 수행 상태를 고려하기 위해서 RTA-MVPR에서는 triggering 트리 구조에 대해서 각 트랜잭션의 종료시점에 근접한 정도를 나타내는 CET(Commitment Extent of Triggering tree)라는 평가 척도를las 사용한다. 트랜잭션 T에 대한 CET는 다음과 같이 정의된다.

$$CET(T) = \frac{|T_{Triggered}| - T_{Uncomm}|}{|T_{Triggered}|} \times 100(\%)$$

위 식에서 $|T_{Triggered}|$는 트랜잭션 T의 하위 triggering 트리의 모든 트랜잭션들의 수를 의미한다. $|T_{Uncomm}|$은 트랜잭션 T의 하위 triggering 트리의 모든 트랜잭션들 중에서 아직 종료단계에 있지 않은 트랜잭션들의 수를 의미한다. 결국, CET(T)는 트랜잭션 T가 실행시킨 트랜잭션들 중에서 종료단계에 있는 트랜잭션의 비율을 의미한다. RTA-MVPR에서 CET (T)는 트랜잭션 T가 종료단계에 있는 얼마나 많은 트랜잭션을 실행시켰는가를 평가한다. 데이터 충돌에 포함된 트랜잭션 T의 CET 값이 70이라면 실행시킨 트랜잭션들 중 70%가 종료단계에 있음을 알 수 있다. RTA-MVPR은 CET의 값에 대한 임계값인 $Rate_{dc}$ 값을 사용하여 $Rate_{dc}$ 보다 CET(T)가 크다면 트랜잭션 T가 종료단계에 근접한 것으로 여기며 이와 같은 상황을 트랜잭션 T가 DC(Dominant Commit) 상태에 있다고 한다. $Rate_{dc}$는 시스템에서 적절하게 변경할 수 있는 값으로 RTA-MVPR 스케줄러의 전체 성능에 영향을 줄 수 있다. $Rate_{dc}$ 값이 높을수록 보다 많은 수의 트랜잭션들이 철회될 것이다.

높은 그룹 우선순위를 가지는 트랜잭션을 T_{Hgp}라 하고 낮은 그룹 우선순위를 가지는 트랜잭션을 T_{Lgp}라 하자. <표 2>은 T_{Lgp}가 소유한 잠금에 대해서 T_{Hgp}가 잠금을 요청한 경우 inter triggering 충돌을 해결하기 위한 잠금 호환성 표이다. 또한, <표 3>는 T_{Lgp}가 소유한 잠금에 대해서 T_{Hgp}가 잠금을 요청한 경우 inter triggering 충돌을 해결하기 위한 잠금 호환성 표이다. inter triggering 잠금 호환성 표는 MVPR의 잠금 호환성 표와 거의 동일하다. MVPR의 잠금 호환성 표와 다른 점은 각 트랜잭션의 우선순위만을 고려하는 것이 아니라 트랜잭션의 그룹 우선순위를 사용하여 데이터 충돌을 해결한다. 또한, 시스템 자원을 낭비하지 않기 위해서 잠금을 소유하고 있는 트랜잭션이 실행시킨 triggered 트랜잭션들의 수행상태를 데이터 충돌 해결에 반영한다.

〈표 2〉 T_{Hgp}가 T_{Lgp}가 소유한 잠금을 요청했을 때 inter triggering 잠금 호환성 표

(T_{Hgp})소유자 \ 요청자(T_{Lgp})		$T_{Lgp} \ll T_{Hgp}$			$T_{Hgp} \ll T_{Lgp}$		
		읽기	쓰기	검증	읽기	쓰기	검증
읽기 (Read)	NOT DC	Y	Y	소유자 철회	Y	소유자 철회	소유자 철회
	DC	Y	Y	요청자 대기	Y	소유자 철회	소유자 철회
쓰기 (Write)	NOT DC	소유자 철회	Y	소유자 철회	Y	Y	Y
	DC	요청자 대기	Y	요청자 대기	Y	Y	Y
검증 (Certify)	NOT DC	소유자 철회	Y	소유자 철회	Y	Y	Y
	DC	요청자 대기	Y	요청자 대기	Y	Y	Y

〈표 3〉 T_{Lgp}가 T_{Hgp}가 소유한 잠금을 요청했을 때 inter triggering 잠금 호환성 표

(T_{Hgp})소유자 \ 요청자(T_{Lgp})		$T_{Lgp} \ll T_{Hgp}$			$T_{Hgp} \ll T_{Lgp}$		
		읽기	쓰기	검증	읽기	쓰기	검증
읽기 (Read)	NOT DC	Y	요청자 철회	요청자 철회	Y	Y	요청자 대기
	DC	Y	요청자 철회	요청자 철회	Y	Y	요청자 대기
쓰기 (Write)	NOT DC	Y	Y	Y	요청자 대기	Y	요청자 대기
	DC	Y	Y	Y	요청자 대기	Y	요청자 대기
검증 (Certify)	NOT DC	Y	요청자 철회	Y	요청자 대기	Y	요청자 대기
	DC	Y	요청자 철회	Y	요청자 대기	Y	요청자 대기

4.5.3 RTA-MVPR의 정확성

RTA-MVPR은 기존에 제안된 MVPR을 기반으로 하는 알고리즘이기 때문에 선행관계에 의한 트랜잭션들 사이의 직렬화가능 스케줄의 증명은 피하기로 한다. MVPR에 대한 보다 자세한 증명은 [7, 16]에서 참조할 수 있다. RTA-MVPR에서 triggering 관계는 선행관계와 밀접한 연관성을 가진다. 예를 들어, 두 트랜잭션 T_1과 T_2가 $T_1 \ll T_2$의 triggering 관계를 가진다면 T_1과 T_2 사이에는 $T_1 < T_2$의 선행관계가 항상 성립한다. triggering 관계가 트리 구조로 표현되기 때문에 트랜잭션들 사이의 triggering 관계는 순환적인 형태를 가질 수 없다. 즉, triggering 관계를 기반으로 하는 $T_1 < T_2$와 $T_2 < T_1$이 동시에 만족되는 직렬화가능 스케줄을 위반하는 경우는 발생하지 않게 된다. 따라서, triggering 관계를 가지고 있는 트랜잭션들 사이에는 직렬화가능 스케줄을 항상 만족시킬 수 있다. triggering 관계가 없는 inter triggering 충돌에 포함되는 트랜잭션들 사이에는 선행관계가 동적으로 생성된다. 이 경우 이 트랜잭션들 사이의 수행 순서는 선행관계로써 정의되며 MVPR에서 선행관계는 순환적인 형태가 형성되지 않음이 [7, 16]에서 이미 증명되었다. 결국, RTA-MVPR 스케줄러는 intra triggering 충돌과 inter triggering 충돌에 의해서 형성된 직렬화가능 스케줄을 만족함이 증명된다.

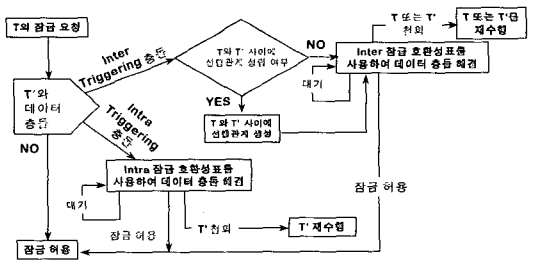
5. 성능 평가

본 장에서는 제안하는 동시성 제어기법과 다른 실시간 동시성 제어 알고리즘을 비교함으로서 성능 평가 결과를 기술하고자 한다. 성능 평가 프로그램은 쓰레드(thread)를 기반으로 하는 모의실험(simulation) 도구인 C++SIM[18]을 사용하여 수행된다.

5.1 RTA-MVPR의 구현

RTA-MVPR은 Newcastle 대학에서 만든 모의실험 도구인 C++SIM을 사용하여 구현되었다. C++SIM은 쓰레드 기반의 다중 프로세스 모델의 알고리즘을 구현하기 용이하게 설계된 C++ 기반의 클래스 라이브러리이다. Solaris 2.7 운영체제의 SUN Sparc 시스템에서 lwp(light-weight process) 기반으로 컴파일된 C++SIM의 프로세스 클래스를 링크하여 RTA-MVPR을 구현하였다.

RTA-MVPR은 4.4절의 LOCK과 COMMIT 등의 크게 두 프로시주어로 구분된다. 트랜잭션은 읽기, 쓰기, 검증 등의 세 가지 연산을 실행하여 데이터베이스를 사용하며 각 연산은 연관된 잠금을 요청하여 획득해야 한다. 잠금을 요청할 때 수행하는 프로시주어는 LOCK이며 트랜잭션 ID, 잠금의 종류, 대상 데이터 등의 인자를 가진다. LOCK의 수행으로 해당 트랜잭션은 대기, 철회, 잠금 허용 등의 상태 중 하나의 상태에 들어가게 된다. 만일, 잠금이 허용되었다면 해당 데이터를 사용할 수 있게 된다. (그림 3)는 LOCK 프로시주어의 알고리즘을 흐름도의 형태로 보여준다. MVPR에서는 엄정 스케줄을 지원하기 때문에 획득된 모든 잠금은 트랜잭션의 종료 시점에서 한번에 해제되므로 각 잠금에 대한 별도의 해제 프로시주어는 필요하지 않다. 종료 시점에서 4.4절의 COMMIT 프로시주어를 수행하며 해당 트랜잭션은 종료 종속성과 선행관계에서 앞서는 모든 트랜잭션이 종료될 때까지 대기하기 때문에 임의의 직렬화가능 순서에 일치하는 트랜잭션 수행을 할 수 있다.



(그림 3) LOCK 프로시주어의 흐름

5.2 모의실험 모델

본 절에서는 성능 평가 프로그램의 기반이 되는 모의실험 모델에 대해서 기술하고자 한다. 성능 평가에 사용되는 실시간 능동 데이터베이스 시스템은 데이터베이스가 디스크 상에 저장되어있다고 가정하며 하나의 프로세서를 기반으로 운영된다. 시스템에 도착하는 트랜잭션들 간의 시간은 지수(exponential) 분포를 따르며, 각 트랜잭션은 트랜잭션의 크기, 종료시한, 우선순위, 도착시간 등의 속성을 가지고 수행된다. 임의의 트랜잭션은 일련의 읽기 또는 쓰기 연산으로 구성되며 성능 평가를 단순화시키기 위해서 버퍼 링은 고려하지 않는다. <표 4>은 모의실험에 필요한 기본적인 매개변수와 의미를 나타낸다. 모의실험 프로그램은 NumTrans개의 트랜잭션이 시스템에서 생성되어 수행 종료될 때까지 계속 수행된다. 각 트랜잭션은 DBsize개의 페이지를 가지는 데이터베이스에서 TranSize개의 임의의 페이지를 접근한다. 트랜잭션이 접근하는 디스크 페이지 중에서 WriteProb의 확률로 쓰기 연산이 실행된다. 트랜잭션 i의 종료시한은 다음과 같이 계산된다.

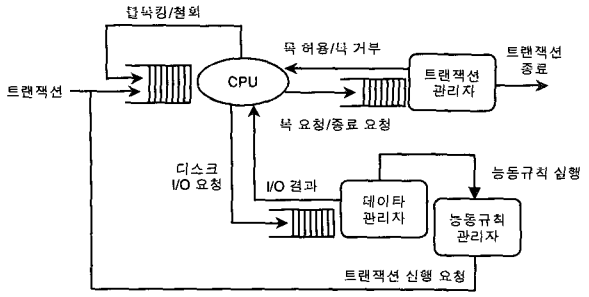
$$Deadline_i = ArrTime_i + (SlackFactor * TranSize_i * (DiskI/O + CPU))$$

$ArrTime_i$는 트랜잭션 i가 시스템에 도착한 시간을 나타낸다. 종료시한은 트랜잭션이 도착한 시간에 트랜잭션의 예상 수행시간을 더한 값이 되며 이 종료시한의 역수를 우선순위로 사용한다. 결과적으로, 본 모의실험에서는 트랜잭션의 우선순위는 종료시한이 가까울수록 높은 우선순위를 갖도록 하는 EDF(Earliest Deadline First) 방식을 채택한다. TriggerProb는 사건에 의해서 조건이 만족하여 능동규칙의 대응하는 처리가 실행될 확률을 의미한다. TriggerProb 값이 클수록 동시에 실행되는 실시간 능동 트랜잭션의 수가 증가하게 된다.

<표 4> 매개 변수와 의미

매개 변수	의 미
DBsize	데이터베이스의 크기 (페이지의 수)
NumTrans	수행될 총 트랜잭션의 수
WriteProb	쓰기 연산의 비율 (%)
DiskI/O	디스크 I/O 시간 (ms)
CPU	하나의 데이터 페이지에 대한 CPU 수행시간 (ms)
TranSize	트랜잭션의 크기
ArrRate	트랜잭션 도착 비율 (trans/sec)
SlackFactor	종료시한의 여유 정도
TriggerProb	능동규칙의 실행 비율 (%)

(그림 4)은 모의실험을 위한 실시간 능동 데이터베이스 시스템의 모델을 나타낸다. 트랜잭션이 종료시한을 초과하더라도 계속해서 수행을 시키는 soft 실시간 데이터베이스 환경을 가정한다. 따라서, 트랜잭션이 수행 중에 철회되더라도 완료될 때까지 계속해서 수행된다. 디스크에 저장된 데이터를 읽거나 쓰는 경우 각 트랜잭션은 동시성 제어 관리자에게 해당하는 잠금을 요청하며, 동시성 제어 관리자는 데이터 충돌 발생 여부를 확인하여 잠금을 허용하거나, 잠금을 요청한 트랜잭션을 블로킹 또는 철회시킨다.



(그림 4) 실시간 능동 데이터베이스 모델

5.3 매개 변수 설정

본 절에서는 모의실험을 위해서 설정되는 매개 변수의 값을 기술하고 성능 평가의 결과로 사용할 변수를 정의한다. <표 5>는 모의실험 중에 고정되는 매개 변수의 값을 나타낸다. <표 4>에서 나타난 나머지 매개 변수들의 값은 모의실험의 종류에 따라서 변경되기 때문에 <표 5>에서 나타내지 않았다. 일반 데이터베이스에서 트랜잭션 관리자의 성능은 트랜잭션의 평균 응답시간이나 단위시간 동안에 처리된 트랜잭션의 수로 평가할 수 있다. 그러나, 실시간 능동 데이터베이스 환경에서는 종료시한을 초과한 트랜잭션들의 수로 그 성능을 평가할 수 있다.

〈표 5〉 매개 변수의 설정

매개 변수	값
DBsize	2,000개 페이지
WriteProb	20%
DiskI/O	25ms
CPU	8ms

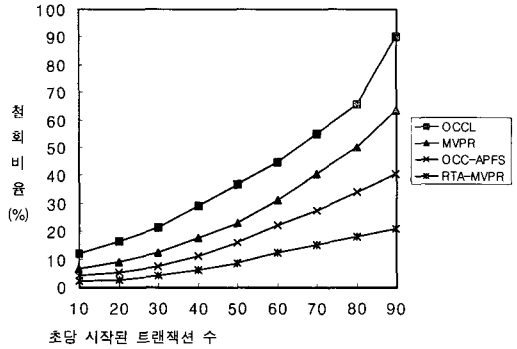
본 모의실험에서 제안하는 실시간 동시성 제어 알고리즘과 성능을 비교하기 위한 알고리즘으로 OCCL[16], MVPR [7, 16], OCC-APFS[1] 등을 채택하였다. OCCL과 OCC-APFS는 낙관적 동시성 제어기법에 잠금을 적용하여 데이터 일관성을 유지한다. OCCL에서는 실시간 동시성 제어기법이기 때문에 triggering 관계를 고려하지 않고 데이터 충돌을 해결하므로 실시간 능동 데이터베이스 환경에서는 좋지 않은 성능을 보여줄 것이다. OCCL의 기본 개념을 기반으로 하는 OCC-APFS는 트랜잭션들 사이의 직렬화가능 순서를 유지하기 위해서 타임 스탬프 개념을 적용하였다. 트랜잭션들 사이의 triggering 관계를 고려하기 위해서 해당 트랜잭션의 triggering 트리 상의 자식 트랜잭션의 수와 우선순위가 데이터 충돌의 해결에 중요한 요인이 된다. MVPR 역시 OCCL와 마찬가지로 실시간 동시성 제어기법이기 때문에 실시간 능동 데이터베이스 환경에서는 좋지 못한 성능을 보여준다. 그러나, MVPR과 OCCL의 성능평가 결과에서 MVPR이 우위를 차지했기 때문에 실시간 능동 데이터베이스 환경에서도 MVPR이 OCCL 보다 우수한 성능을 보여줄 것이다[7, 16].

OCCL과 MVPR의 가장 큰 차이점은 OCCL은 낙관적 기법이기 때문에 트랜잭션의 검증 단계에서 데이터 충돌을 해결한다. 따라서, 결국에는 철회될 트랜잭션이 종료시점에 근접한 검증 단계에서 철회되기 때문에 불필요하게 시스템 자원을 낭비하게 된다. 이와 같은 상황은 트랜잭션의 수가 증가할수록 심해지게 된다. OCC-APFS 역시 검증 단계에서 데이터 충돌을 해결하기 위한 대부분의 알고리즘이 수행되므로 OCCL과 유사한 문제점이 있다. 그러나, MVPR은 검증 단계 이전부터 직렬화가능 순서와 우선순위를 기반으로 데이터 충돌을 해결하므로 철회되어야 할 트랜잭션은 검증 단계까지 수행되기 힘들게 된다.

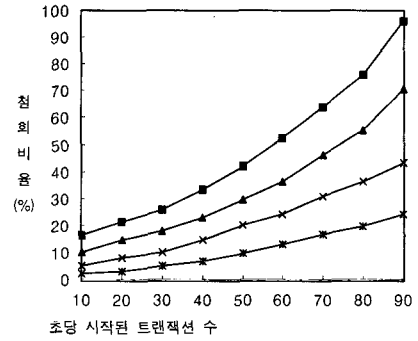
5.4 성능 평가 결과

본 절에서는 모의실험을 토대로 OCCL, MVPR, OCC-APFS과의 성능 평가 결과를 제시하고 분석을 하기로 한다. (그림 5)는 TransSize, SlackFactor가 각각 50과 2.022인 트랜잭션의 철회 비율을 보여준다. (그림 6)에서 (a)와 (b)는 각각 TriggerProb가 각각 50%와 70%일 때를 나타낸다. (그림 5)에서 나타난바와 같이 제안하는 알고리즘인 RTA-MVPR이 다른 알고리즘에 비해서 철회 비율이 현저하게 낮은 걸 알 수 있다.

OCCL의 경우 초당 트랜잭션의 수가 50인 경우 Trigger-Prob가 각각 50%와 70%인 경우 각각 36.7%와 42%의 트랜잭션 철회 비율을 보인다. 그러나, 트랜잭션의 수가 90에 근접하는 경우 거의 100%의 철회 비율에 가까워짐을 알 수 있다. 이와 같은 상황은 OCCL이 낙관적인 기법을 채택하고 있기 때문에 시스템 내에서 동시에 수행하는 트랜잭션의 수가 증가할수록 검증 단계에 있을 확률이 높아지고 결과적으로 데이터 충돌에 포함된 트랜잭션들을 연쇄적으로 철회시키기 때문이다. RTA-MVPR과 OCC-APFS는 트랜잭션의 수가 50 이하인 경우 비슷한 철회 비율을 보이지만 OCCL 기법을 기반으로 하는 OCC-APFS는 트랜잭션의 수가 증가할수록 OCCL의 경우에서처럼 철회 비율이 점점 증가하는 것을 볼 수 있다. 그에 반해 RTA- MVPR은 선행관계와 triggering 관계를 동시에 고려하고 검증 단계 이전부터 데이터 충돌을 해결하려고 하기 때문에 트랜잭션의 수가 증가하더라도 철회 비율이 급격하게 증가하지는 않고 있다. 또한, OCC-APFS가 트랜잭션이 실행시킨 triggered 트랜잭션들의 수를 의미하는 *FS(Fan-out Sum)* 값을 사용하지만 같은 trigger 트리 내의 트랜잭션을 고려하고 있지 않기 때문에 불필요하게 triggered 트랜잭션을 철회시킬 수 있다. 결과적으로 (그림 5)에서 나타난바와 같이 RTA-MVPR은 OCC-APFS에 비해서 트랜잭션 철회 비율이 낮음을 볼 수 있다.



(a)

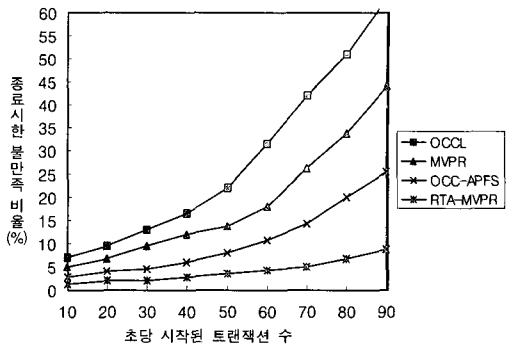


(b)

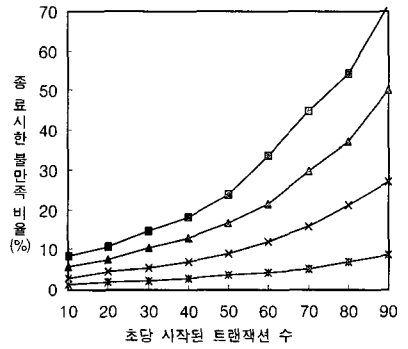
(그림 5) TriggerProb에 따른 트랜잭션 철회 비율

(그림 6)에서 (a)와 (b)는 TriggerProb가 각각 50%와 70%인 경우 종료시한을 만족하지 못한 트랜잭션의 비율을 나타낸다. 동시 수행되는 트랜잭션의 수가 증가할수록 종료시한을 만족하지 못하는 트랜잭션의 수가 증가함을 알 수 있다. 그러나, (그림 6)에서 나타난바와 같이 RTA-MVPR의 경우 트랜잭션의 수가 증가하여도 종료시한을 만족하지 않는 트랜잭션 수가 다른 알고리즘들에 비해서 많지 않다. soft 실시간 데이터베이스 환경을 가정했기 때문에 종료시한을 만족하지 못하는 트랜잭션의 수는 (그림 5)의 트랜잭션 철회 비율과 밀접한 관계를 가진다. 본 모의실험에서는 트랜잭션이 수행도중 철회되는 경우 재 수행을 하므로 철회 비율이 높아질수록 종료시한을 만족할 가능성은 낮아질 수밖에 없다. 따라서, 철회 비율이 높을수록 종료시한을 만족하지 못하는 트랜잭션의 수는 증가하게 된다. OCC-APFS의 경우 낮은 우선순위의 트랜잭션을 철회할 때 이 트랜잭션에 의해서 실행된 모든 triggered 트랜잭션의 수행 상태를 고려하지 않기 때문에 RTA-MVPR에 비해서 더 많은 트랜잭션 철회가 발생한다.

마지막으로, (그림 7)의 (a)와 (b)는 TriggerProb가 각각 50%와 70%일 경우 유용한 재 수행 비율을 나타낸다. 유용한 재 수행 비율이란 철회된 트랜잭션들이 종료시한을 만족하는 비율을 의미한다. 만일, 데이터 충돌을 해결하기 위해서 트랜잭션을 철회한 경우 결과적으로 종료시한을 만족하지 못한다면 무의미하게 트랜잭션을 철회한 셈이 된다.

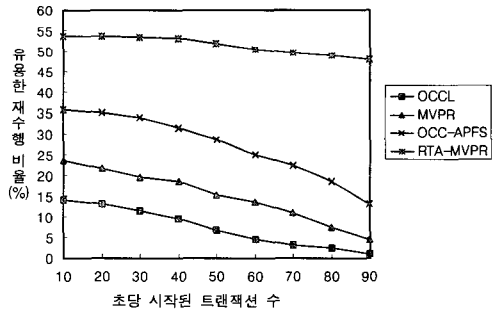


(a)

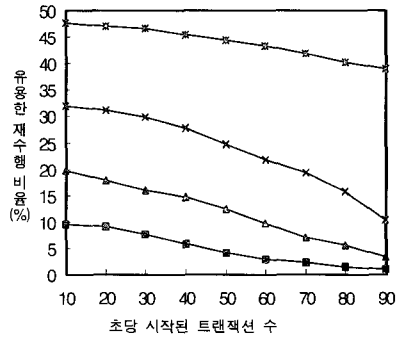


(b)

(그림 6) TriggerProb에 따른 종료시한 불만족 비율



(a)



(b)

(그림 7) TriggerProb에 따른 유용한 재 수행 비율

(그림 7)의 (a)에서 초당 시작된 트랜잭션의 수가 40인 경우 유용한 재 수행 비율은 OCCL, MVPR, OCC-APFS, MVPR이 각각 9.6, 18.5, 31.4, 53.2 등으로 나타난다. OCC-A

PFS의 유용한 재 수행 비율은 RTA-MVPR에 비해서 21.8% 정도가 낮게 나온다. 이와 같은 결과는 OCC-APFS는 트랜잭션을 철회시킬 경우 triggered 트랜잭션들의 수행 상태를 고려하지 않기 때문에 하위 트리의 모든 triggered 트랜잭션들도 모두 철회된다. 만일, triggered 트랜잭션들 중 많은 수가 종료시점에 근접했다면 자원의 낭비 뿐 아니라 재 수행되는 경우 종료시한을 만족할 확률이 감소하게 되므로 결과적으로 유용한 재 수행 비율이 감소하게 된다. 그러나, RTA-MVPR은 트랜잭션의 철회 시 해당 triggered 트랜잭션들의 수행 상태를 고려하기 때문에 종료시점에 근접한 트랜잭션들이 많다면 철회되지 않도록 데이터 충돌을 해결한다.

6. 결론 및 향후 연구 방향

본 논문에서는 최근 각광 받고 있는 연구 분야인 실시간 능동 데이터베이스 환경을 위한 동시성 제어 기법을 제안하였다. 제안한 동시성 제어 기법은 능동 데이터베이스에서 중요한 개념인 triggering 관계를 고려하여 실시간 능동 데이터베이스 시스템에서도 적절하게 적용될 수 있음을 보였다. 이와 같은 triggering 관계와 선행 관계를 함께 고려함으로서 보다 효율적으로 트랜잭션들 사이의 충돌을 해결하고자 시도하였다. triggering 관계를 기반으로 트랜잭션들 사이의 종속관계를 정의하였으며 이와 같은 종속관계에 의해서 실시간 능동 트랜잭션 모델을 제안하였다. 트랜잭션들 사이의 종속관계에 의해서 데이터 충돌을 inter triggering 충돌과 intra triggering 충돌 등의 두 가지로 분류하였으며 각각을 위해서 충돌 해결 기법을 제안하였다. 이와 같은 충돌 해결 기법에 의해서 트랜잭션들에 대한 불필요한 대기와 철회와 같은 문제들을 해결하였으며, triggering 관계를 기반으로 종료 시한을 만족하는 트랜잭션의 수를 증가시켰다. 또한, 모의실험을 통해서 OCCL, MVPR, OCC-APFS 등과 같은 알고리즘과 성능 평가를 하였다. 성능평가 결과에서 제안한 알고리즘인 RTA-MVPR이 triggering 관계와 우선순위를 고려하여 낮은 트랜잭션 철회비율과 종료시한을 만족하는 트랜잭션의 비율이 높음을 보였다. 또한, 데이터 충돌을 해결하기 위해서 트랜잭션을 철회하는 경우 재 수행되어 종료시한을 만족하는 트랜잭션의 비율이 높다는 결과도 보여주었다.

향후 연구 방향은 inter triggering 충돌을 해결하기 위해서 사용한 CET 척도의 정확성을 향상시키는 것이다. 단순히 triggered 트랜잭션들 중에서 종료 단계에 있는 트랜잭션들의 비율만 고려하는 것이 아니라 triggering 트리의 구조나 triggered 트랜잭션들의 수까지도 고려하도록 한다.

참 고 문 헌

[1] A. Bajaj A. Datta and R. Veloo, "A study of concurrency control in real-time active database systems," Technical Report RTRG-TR-95-06, Dept. of MIS, Univ. of Arizona, Tucson, AZ 85721, August 1995.

[2] J. A. Stankovic R. M. Sivasankaran, K. Ramamritham and D. Towsley, "Data placement, logging and recovery in real-time active databases," In International Workshop on Active and Real-Time Database Systems. ACT-NET, June 1995.

[3] J. Lee and S. H. Son, "Using dynamic adjustment of serialization order for real-time database systems," In Proceedings of the 11th Real-Time Systems Symposium, pp. 66-75, December 1993.

[4] J. R. Haritsa, M. J. Carey, and M. Linvy, "On being optimistic about real-time constraints," In Proceedings of the 1990 ACM PODS Symposium, pp.331-343, April 1990.

[5] R. Abbott, "Scheduling real-time transactions : A performance eval uation. ACM Trans. on Database Systems," Vol.17, No.3, pp.513-560, September 1992.

[6] S. Chakravarthy and E. Anwar, "Exploiting active database parad igm for supporting flexible transaction model," Technical Report UF-CIS-TR-95-026, University of Florida, Computer and Information Science and Engineering Department, Gaineville, Florida, April 1995.

[7] S. H. Hong and M. H. Kim, "Resolving Data Conflicts with Multiple Versions and Precedence Relationships in Real-Time Data bases," Information Processing Letters(IPL), Vol.61, No.3, pp.149-156, 1997.

[8] Y. Lin and S. H. Son, "Concurrency control in real-time database by dynamic adjustment of serialization order," In Proceedings of the 11th Real-Time Systems Symposium, pp.104-112, Dec. 1990.

[9] J. A. Stankovic, R. M. Sivasankaran, B. Purimetla and K. Ramamritham, "Network services database-a distributed active real-time database(dartdb) application," In IEEE Workshop on Real-Time Applications, May 1993.

[10] R M. Sivarankaran, B. Purimetla and J. A. Stankovic, "A study of distributed real-time active database applications," In IEEE Workshop on Parallel and Distributed Real-Time Systems, April 1993.

[11] S. Chakravarthy and E. Anwar, "Exploiting active database paradigm for supporting flexible transaction model," Technical Report UF-CIS-TR-95-026, University of Florida, Computer and Information Science and Engineering Department, Gaineville, Florida, April 1995.

[12] R. Jauhari, M. J. Carey and M. Livny, "On trans action boundaries in active databases : A performance perspective," IEEE Transaction on Knowledge and Data Engineering, Vol.3, No.3, pp.320-336, September 1991.

[13] A. P. Sistla and O. Wolfson, "Temporal triggers in active databases. IEEE Trans. on Knowledge and Data Engineering," Vol.7, No.3, pp.471-486, June 1995.

[14] U. Dayal, M. Hsu aand R. Ladin, "Organizing long running activities with triggers and transactions," ACM SIGMOD,

17, 1, March 1990.
[15] A. Bernstein, A. Philip, V. Hadzilacos and N. Goodman, "Concurrency control and Recovery in database systems," Addison-Wesley, 1987.
[16] J. Huang, J. A. Stankovic, and D. Towslwy K. Ramamritham, "Experimental evaluation of real-time optimistic concurrency control schemes," In Prooceedings of the 17th International Conference on Very Large Data Bases, pp.35-46, Barcelona, Spain, September 1991.
[17] 김명호, 홍석희, "선행관계를 고려한 다중버전 기반 실시간 동시성 제어 기법", 한국정보과학회지, 제24권 제11호, pp. 1123-1133, 1997.
[18] Univ. of Newcastle upon Tyne, "C++SIM User's Guide," 1996.

홍 석 희

e-mail : shhong@csd.kyungsung.ac.kr
1989년 홍익대학교 공과대학 전자 계산학과 졸업(학사)
1991년 한국과학기술원 전산학과 졸업 (석사)
1997년 한국과학기술원 전산학과 졸업 (박사)
1997년 한국전자통신연구원 데이터베이스 연구실 Post Doc.
1997년~현재 경성대학교 정보과학부 조교수
관심분야 : 실시간 데이터베이스, 능동 데이터베이스, 동시성 제어, 트랜잭션 처리, 저장 시스템