트랜잭션의 낭비적 재시작 문제를 개선한 대체버전 병행제어 알고리즘

임종우°, 서창석, 이병욱
경원대학교 전자계산학과

Alternative Version Concurrency Control Algorithm by improving the wasted restart transaction problem

Jongwoo Lim°, Changsuk Seo, Byungwook Lee

요 약

기존의 병행제어 알고리즘들은 특정 종류의 응용에서만 우수한 성능을 보여왔다. 그러나, 실시간 데이터베이스 응용에서는 여러 종류, 비 실시간, 소프트 종료시한, 펌 종료시한, 하드 종료시한 트랜잭션들이 혼합된 시스템이 대부분이다. 따라서, 여기에 합당한 병행제어 알고리즘이 필요하다. 대체버전 병행제어 기법은 2단계 로킹-높은 우선순위(2PL-HP)의 문제점인 낭비적 재시작과 낭비적 수행 문제를 해결하기 위해 제안되었다. 이 방식은 충돌이 자주 발생하는 응용 즉, 부하가 많은 응용에서는 불필요한 트랜잭션의 재시작과 블록킹이 많이 발생하고, 종료시한을 만족하기 힘들었다. 이는 트랜잭션 충돌시 재시작 버전 트랜잭션의 생성 수를 제한하지 않고 충돌이 발생할 때 마다 충돌 해결 정책을 사용하였기 때문이다. 본 논문은 실시간 데이터베이스 시스템이 단위 시간에 처리할 수 있는 최대 트랜잭션의 수를 고려하여 낭비적 재시작 트랜잭션의 수를 제한하기 때문에 충돌 연산과 관련이 없는 다른 트랜잭션에 보다 많은 실행의 기회를 제공함으로써, 처리율을 향상한다. 또한 중단된 트랜잭션의 수가 자원을 낭비하는 것을 방지하여 자원 활용면에서도 효율이 증가한다.

Ⅰ. 서론

실시간 데이터베이스 시스템은 항공기 관리와 무기체계, 공장자동화, 로보틱스, 원자력 발전소, 교통제어 시스템과 같은 다양한 응용분야에서 점차 중요시되고 있다. 최근에는 전사적 자원 관리(Enterprise Resource Planning), 데이터웨어 하우징의 지원 도구인 OLAP 등의 분야에서도 그 필요성이 요구된다.

최근에 이 분야의 연구자들은 기존의 데이터 모델과 트랜잭션 모델이 시간적 제약을 가지는 응용분야에 적합하지 않기 때문에, 공유 데이터를 수집하고, 갱신하고, 검색하는데 있어서 시간적인 제약조건들을 만족시킬 수 있는 데이터베이스 시스템에 대해 많은 연구들을 진행하여 왔다[8]. 실시간 데이터베이스 시스템의 요구 조건과 설계 목표는 기존의 데이터베이스 시스템과는 많이 다르며, 새로운 기술들이 데이터베이스의 일관성을 유지하기 위해서 필요하다. 실시간 데이터베이스 시스템은 시간에 기반한 스케줄링과 양립할 수 있어야 하며, 시스템의 응답 시간 요구 조건들과 시간적 일관성 요구 조건들을 두루 만족해야 한다. 이때 중요한 것은 어떻게 기존의 데이터베이스 시스템을 수정해서 그 성능과 예측 가능성이 실시간 응용분야에 적합하도록 설계하는데 있다. 데이터베이스 시스템에서의 병행제어 이론과 실시간 태스크 스케줄링 분야에 많은 진보가 있어온 반면에, 병행제어 프로토콜과 실시간 스케줄링 알고리즘 사이의 상호작용에 대해서는 거의 연구가 이루어지지 않았다. 실시간 데이터베이스 시스템에서 해결되어야 하는 문제들 중의 하나는 세 가지의 제약조건, 즉 데이터의 일관성, 트랜잭션의 정확성, 트랜잭션 마감 시간을 만족시키면서 동시성과 자원에 대

한 활용도를 최대화하는 실시간 스케줄링과 동시성 프로토콜에 대한 통합 이론을 만드는 것이다. 최근에 여러 개의 프로젝트들이 실시간 데이터베이스 시스템에서 시간적 제약조건을 효율적이고 정확하게 관리할 수 있도록 데이터베이스 기술과 실시간 요구조건들을 통합하고 있다. 본 연구에서는 실시간 데이터베이스 병행제어 알고리즘 중 대체 버전 병행제어 알고리즘에서 생기는 문제점인 낭비적 재시작 트랜잭션의 수의 기하급수적 증가를 제한하여 기존 대체 버전 병행제어 알고리즘을 개선하고자 한다.

II. 관련 연구

2.1 실시간 데이터베이스 시스템

실시간 데이터베이스 시스템은 기존의 데이터베이스 시스템의 조건과 트랜잭션이 종료시한(deadline)과 같은 시간적 제약 조건을 가지는 데이터베이스 시스템이다. 다음 그림 2-1은 트랜잭션의 흐름에 따른 실시간 데이터베이스 모델을 보여준다. 이 모델은 [17]을 참조하여 확장한 모델이다. 정해진 사용자들이 트랜잭션을 요청하는 경우를 모델화한 것이고, 이 모델은 사용자의 수의 제한이 없다.

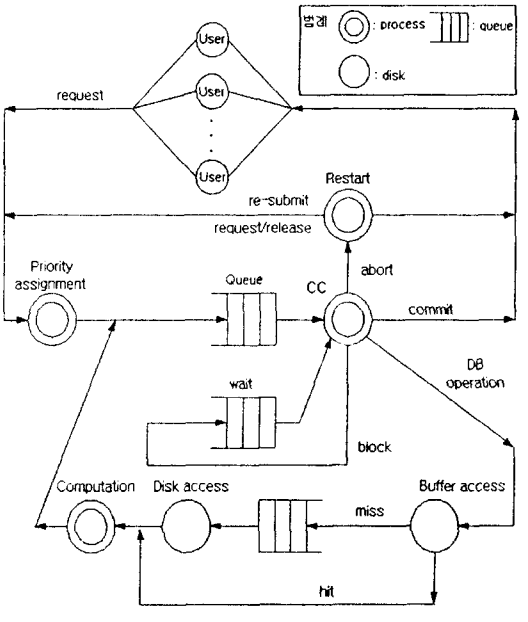

그림 2-1 실시간 데이터베이스 모델

새로 생성되는 트랜잭션이나 다시 실행되는 트랜잭션들은 병행적으로 실행되는 다른 트랜잭션들과의 순서화을 위한 우선권을 할당받는다. 트랜잭션은 데이터에 대해 연산을 하기 전에, 객체에 대한 로크(lock)을 얻기 위해 병행제어 콤포넌트를 통과해야 한다. 요청이 거절되면, 트랜잭션은 대기 큐에 있게 된다. 대기하고 있는 큐는 요청받은 로크가 풀리면(released) 깨어난다. 요청이 받아지면, 트랜잭션은 global buffer access, disk access 그리고 computation으로 구성된 것을 작동한다. 트랜잭션은 완료될 때까지 "요청-동작 순환(request-operation cycle)"을 계속해서 한다. 트랜잭션이 완료(commit) 단계에 이르면, 트랜잭션은 자신이 가지고 있는 모든 로크를 해제한다. 병행제어 알고리즘은 몇 가지 이유로 트랜잭션을 취소한다. 이런 경우에 'restart' 콤포넌트는 병행 정책에 의해 취소된 트랜잭션이 다시 '재요청(re-submitted)'또는 '종료(terminated)'를 결정한다. 이 모델은 단지 트랜잭션 처리를 포함한 논리적인 연산을 반영한 것이고 물리적인 자원을 처리하는 콤포넌트와의 상호 작용과 CPU 스케줄링 알고리즘은 보여주지 않는다.

2.2 실시간 스케줄링

실시간 스케줄링 방법은 스케줄링 시점이 결정되는 방식에 따라 크게 두 가지로 분류할 수 있는데 하나는 시간 구동(time-Driven) 방식이고 또 다른 하나는 이벤트 구동(event-Driven) 방식이 있다.

2.2.1 시간 구동 스케줄링 기법

스케줄링 시점이 주기적인 클럭 인터럽트에 의해서 결정되거나, 또는 예약된 시점에 걸리는 비주기적인 클럭 인터럽트에 의해서 결정된다. 각각의 스케줄링 시점에서 스케줄러는 다음 시간 구간 동안 실행될 태스크를 결정하는데, 일반적으로 구해진 스케줄을 케이블에 저장해 두고 이를 참조하여 결정한다. 스케줄이 정적으로 결정되기 때문에 이러한 스케줄링 방법을 오프라인(off-Line) 스케줄링이라고 한다. 시간 구동형 스케줄링 방식의 전형적인 예로 순환 실행 구조(cyclic executive)방법을 들 수 있다 [5]. 순환 실행 구조 스케줄링 방법에서는 정적으로 정해진 실행 순서에 따라서 태스크들을 차례로 실행한다. 시간 구동형 스케줄링 방식은 스케줄링 형태가 미리 결정되어 있기 때문에 시스템에 대한 예측 가능성이 매우 높다는 장점이 있다. 그러나, 같은 이유로 유연성이 부족하기 때문에 태스크 집합이 유동적인 시스템에는 적응하기 어렵다.

2.2.2 이벤트 구동 스케줄링 기법

이벤트 구동형 스케줄링 방식에서는 스케줄링 시점이 각각의 이벤트가 발생하는 시점과 일치한다. 여기서 말하는 이벤트란 태스크가 실행되기 위해 해제되거나 또는 실행을 완료하는 두 가지 상황을 의미한다. 그리고, 각각의 이벤트가 발생한 시점에서 다음에 실행될 태스크를 결정하는데, 이 경우 우선순위(priority)가 가장 높은 태스크가 선택된다. 이런 이유로 이벤트 구동형 스케줄링 방식은 우선 순위 기반 스케줄링 방식으로 더 많이 불리운다. 우선 순위 기반 스케줄링 방식은 태스크의 우선 순위가 고정되어 있는 고정 우선 순위 기반 스케줄링 방식과 태스크 인스턴스마다 서로 다른 우선 순위가 동적으로 부여될 수 있는 동적 우선 순위 기반 스케줄링 방식으로 구분된다. 한편, 실시간 스케줄링 방식은 범용 스케줄링 방식에서와 마찬가지로 선점(preemption)가능한 방식과 불가능한 방식으로 구분되는데, 대부분의 우선 순위 기반 스케줄링 방법은 선점형 방식을 따른다. 우선 순위 기반 스케줄링 방법에 의한 스케줄링은 결정적인(deterministic)성질이 순환 실행 구조 스케줄링 방식보다는 떨어지지만 유연성 측면에서 장점을 가진다.

2.3 대체버전 병행제어

대체 버전 병행 제어 기법은 로크 기반 알고리즘인 2PL-HP을 바탕으로 낙관적 병행제어의 지연 재시작 기법을 적용한 것이다[15,16].

2.3.1 대체버전 병행제어 개념

로크 기반 알고리즘인 2PL-HP을 바탕으로 낙관적 병행제어의 지연 재시작 기법을 적용한 개념을 그림 2-6에서 보여준다.

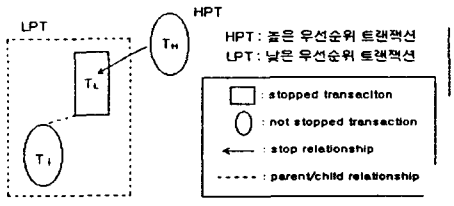

그림 2-6 대체 버전 병행제어 기법 개념도

HPT T_H(로크를 요청하는 높은 우선순위 트랜잭션)이 LPT T_L(로크를 가지고 있는 낮은 우선순위 트랜잭션)과 충돌하면 T_L은 멈추고, T_L의 대체로써 즉시 재시작될 T_i가 초기화된다. 따라서, 멈춘 버전(stopped version)과 재시작된 버전(restarted version) 모두를 동시에 가지게 된다. T_L은 낙관적 병행제어의 지연/재시작 기법에 사용되는 트랜잭션이고, T_i는 2PL-HP의 즉시 재시작 기법에 적용되는 트랜잭션으로 두 가지 기법의 장점을 모두 활용할 수 있음을 보여준다. 따라서, 대체 버전 병행제어 기법은 각 트랜잭션마다 그 트랜잭션의 대체 버전을 유지하여 2PL-HP의 장점과 낙관적 병행제어의 장점 모두를 수행할 수 있다. 2PL-HP는 폼 종료 시한 트랜잭션에 대해서 낭비적 재시작과 낭비적 기다림의 문제를 발생시키고, 낙관적 병행제어에서는 비실시간 트랜잭션이나 소프트 종료시한 트랜잭션에 대해서 낭비적 수행의 문제를 발생시킨다. 관련된 문제를 간단히 정의하면 다음과 같다.

- 낭비적 재시작(wasted restart) : 낭비적 재시작은 높은 우선순위 트랜잭션(이하:HPT)가 낮은 우선순위 트랜잭션(이하:LPT)를 취소시키고 나서, HPT가 종료시한 문제로 취소되는 경우에 발생한다.
- 낭비적 기다림(wasted wait) : 낭비적 기다림은 LPT가 HPT의 완료를 기다리는데 나중에 HPT가 종료시한 문제로 취소되는 경우에 발생한다.
- 낭비적 수행(wasted execution) : 낭비적 수행은 검증 단계에 있는 LPT가 아직 완료하지 않은 HPT와 충돌하기 때문에 재시작 되는 경우에 발생한다.

대체 버전 병행제어 기법은 각 트랜잭션 마다 여러 개의 버전을 유지함으로써 이러한 문제를 해결하고 있다.

III. 재시작 트랜재션의 수 증가를 제한한 대체 버전 병행제어

3.1 제안 알고리즘 개념

대체 버전 병행제어 알고리즘의 문제점인 재시작 버전 트랜잭션 수가 시스템이 허용하는 범위안에서 무제한으로 생성되는 것을 제한하는 개념도는 그림 3-1과 같다.

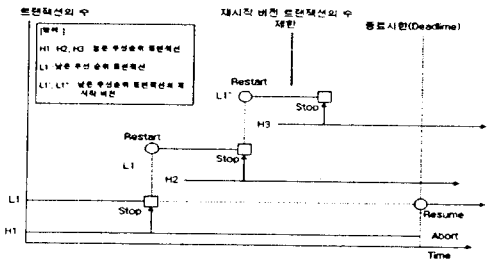

그림 3-1 재시작 트랜잭션의 수 제한 개념도

기존의 대체 버전 병행제어 알고리즘은 충돌시, 낮은 트랜잭션은 멈추고, 그 트랜재션의 초기화 트랜잭션을 생성하고 높은 트랜잭션은 실행된다. 즉, 충돌 시 초기화하여 생성되는 재시작 버전 트랜잭션의 수는 종료 시한내에서 시스템이 허용하는 범위에서 제한없이 증가할 수 있다. 이는 중단된 버전이 자원을 계속 점유하고 있기 때문에, 병행성이 떨어질 수 있고 종료시한을 어기는 경우도 많아지게 되는 문제가 발생한다. 낭비적 재시작 트랜잭션의 수가 증가되어 자원 제한적이고 부하가 많은 응용에서는 시스템의 오버헤드를 발생시킬 수 있는 문제가 발생한다. 또한 충돌 연산과 관계 없는 다른 트랜잭션들이 실행할 수 있는 기회를 상실함으로써 시스템의 처리율이 낮아지게 된다. 그림 3-1에서 제안 알고리즘은 종료시한내에 무제한으로 발생할 수 있는 낭비적 재시작 트랜잭션 수의 증가를 전역 공유 로크 테이블에서 재시작 트랜잭션의 수를 참조하여 일정 수 이상의 생성을 제한함으로써 시스템의 오버헤드 발생 요인을 감소시키고, 자원을 계속 점유하고 있는 중단된 버전 트랜잭션의 수도 줄어들어 자원의 낭비를 줄임으로써 충돌 연산과 관계가 없는 다른 트랜잭션들에게 실행 기회를 주어 종료 시한에 처리할 수 있는 트랜재션의 수를 증가시킬 수 있다.

3.2 제안 알고리즘 절차

다음은 '로크 요청(lock_acquire), 완료(commit), 취소(discard)' 등 기존 대체 버전 병행제어 알고리즘을 개선한 제안 알고리즘의 3가지 주요 절차를 보여준다.

(1) 로크 요청(lock_acquire)

트랜잭션 충돌시 높은 우선순위 트랜잭션이 록을 요청하는 절차는 다음과 같다.

① 트랜잭션이 데이터에 로크를 요청할 때, 로크 호환성과 부모/작식 관계를 체크해야 한다.

② 보다 높은 우선 순위 트랜잭션을 요청하는 로크와 낮은 우선순위 트랜잭션이 충돌하면, 높은 우선순위 트랜잭션은 낮은 우선순위의 로크를 갖고 계속 실행하게 되고 낮은 우선순위 트랜잭션은 중단된다.

③ 전역공유 로크 테이블에서 관련된 재시작 버전 트랜잭션의 수(T_i.stop_cnt의 수)가 N 개 미만인지 체크한다.

④ N개 미만이면 낮은 우선순위 트랜잭션의 재시작 버전이 생성되어 초기화 된고 높은 우선순위 트랜잭션은 록을 소유하게 된다. N개 이상이면, 재시작 버전은 생성되지 않고 중단된 상태로 있고, 높은 우선순위 트랜잭션은 록을 소유하게 된다. (N < 시스템에서 동시에 처리할 수 있는 트랜잭션의 수)

(2) 완료(commit)

록을 요청한 트랜잭션의 완료(commit) 절차는 다음과 같다.

① 지역 트랜잭션의 정보를 global shared lock 테이블에 복사한다.

② 높은 트랜잭션이 완료하면, stop_list에 있는 중단 버전들과 그 상위 버전을 제거한다.

③ 중단된 트랜잭션의 재시작 버전 트랜잭션은 계속 실행된다.

여기서는 지역은 전역을 복사하고 멈춤 버전을 제거한다. 그림 3-4에서 보여주듯이 트랜잭션 T_l이 완료될 때, 트랜잭션 T_l은 stop_list를 사용하여 T_l에 의해 멈춘 트랜잭션과 T_l의 조상 트랜잭션을 제거한다. 그러므로 T_l이 완료될 때, T_l의 stop_list에 있는 T_h와 T_h의 조상 트랜잭션 T_a는 제거된다.

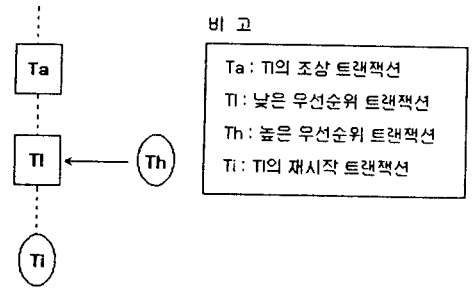

그림 3-4 트랜잭션 완료

(3) 취소(abort)

록을 요청한 트랜잭션이 취소될 때의 프로시저는 다음과 같다.

① 취소된 트랜잭션에 의해 중단된 트랜잭션을 취소시키고 재시작 버전 트랜잭션은 계속 실행된다.

② 중단된 트랜잭션과 자식관계가 있는 트랜잭션이 또 다른 트랜잭션에 의해 중단되었는지를 stop_list에서 조사한다.

③ 자식 관계가 있는 트랜잭션이 다른 트랜잭션에 의해 중단되어 있으면, 그 트랜잭션은 실행하고 자식 관계 트랜잭션은 제거된다.

위의 3가지 절차 '록 요청(lock_acquire)', '완료(commit)', '취소(discard)'에 의해서 즉시/재시작 버전 트랜잭션이 종료시한내에서 제한없이 생성되는 것을 제한할 수 있다.

IV. 모의 실험 및 결과

제안된 병행제어 알고리즘인 즉시/재시작 버전 트랜잭션 수를 제한한 대체 버전 병행제어 기법을 실험 평가하기 위하여 기존의 대체버전 병행제어기법과 비교 평가를 하였다. 실시간 데이터베이스 모델은 2장에서 제시한 실시간 데이터베이스 시스템 모델을 기초로 한다. 모의실험 평가 프로그램은 모의실험(simulation) 도구인 SLAM II를 사용하여 수행한다.

4.1 모의 실험 환경

가정하는 시스템 환경은 한 개의 CPU와 한 개의 디스크를 가진다. 모의 실험 운영체제는 Unix Sun Solaris 1.x이고 사용 Tool은 SLAM II를 사용하였다. 가정하는 트랜잭션의 수는 1,000개로 정하고 트랜잭션 발생기에서는 지수 함수 분포(exponential distribution)를 갖는 임의의 변수 즉, 평균 도착간격 시간(mean interarrival time)에 따라 트랜잭션을 발생시킨다. 즉, 평균 도착간격 시간이 작을수록 초(sec)당 시스템에 유입되는 전체 트랜잭션들의 수가 증가되는 것이다. 트랜잭션 발생 간격을 작게하는 것은 데이터베이스 시스템의 활용 빈도가 높음을 의미한다. 제안 알고리즘에 대한 모의 실험은 평균 도착 간격 시간은 1msec으로 고정시키고 실험하였다. 충돌율은 트랜잭션이 발생하여 종료시한내에 수행중 충돌이 일어날 확률로, 모의 실험을 단순화하기 위해 각각 발생한 트랜잭션에 대해 충돌이 일어날 확률을 10% - 80%까지 10%씩 가변 설정하여 충돌율에 따른 트랜잭션의 완료율을 실험하였다. 충돌이 일어난 트랜잭션중 우선 순위가 낮은 트랜잭션은 우선 순위가 높은 트랜잭션이 완료 또는 종료시한에 의해 취소가 될 때까지 충돌이 계속 일어난다. 즉, 낮은 트랜잭션은 그와 동일한 트랜잭션을 재시작 트랜잭션과 멈춤 트랜잭션으로 나누어져 계속 생성한다. 높은 트랜잭션이 완료한 경우는 대기 큐에서 LIFO로 재시작 버전 트랜잭션 중 최근에 생성된 트랜잭션을 선택하여 실행시키고, 높은 트랜잭션이 취소한 경우는 대기 큐에서 FIFO로 멈춤 트랜잭션 중 가장 오래된 것을 실행시킨다. 모의 실험을 단순화하기 위해 버퍼링은 고려하지 않는다. 각 트랜잭션은 다음과 같은 식을 이용하여 종료 시한과 우선순위를 결정하였다.

종료시한 = 평균도착 간격시간 + 트랜잭션 연산 수 * OP_TIME*SLACK

우선순위 = 종료시한/1000.0(earlist-deadline-first policy)

SLACK : 여유 지연 시간인 SLACK은 1-2사이에 할당하고, 이를 수행하기 위해서 소요되는 시간인 OP_TIME은 연산에 대한 CPU 서비스 시간을 의미한다. 다음은 실험 평가를 위해 가정한 매개 변수들이다.

표 4-1 매개변수 목록

매 개 변 수	값	비 고
트랜잭션의 수	1000	지수 분포
평균도착 간격시간	1 msec	지수 분표
재시작버전 트랜잭션 생성 시간	1 msec	지수 분포
충돌율	0.1 - 0.8	
종료 시한 slack	1 -2	
트랜잭션 수행시간	9 - 10msec	

4.2 모의 실험 모델

펌 실시간 트랜잭션 시스템에서 충돌 연산이 많은 응용을 모델로하여 모의 실험을 한다. 이에 요구되는 시스템 관리기 구성 다음과 같다.

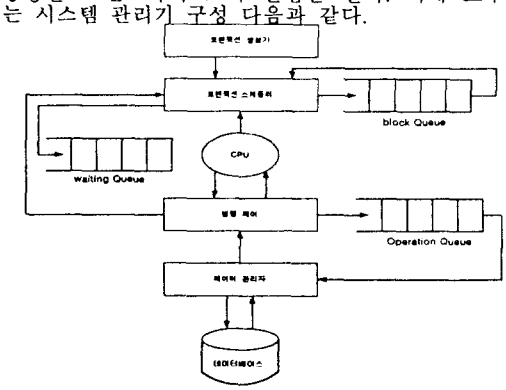

그림 4-1 모의실험 모델의 시스템 구조도

그림 4-1은 실험 평가 모델의 시스템 구조도로써, 트랜잭션 스케줄러(Transaction Scheduler), 병행제어(Concurrency Control), 데이터 관리자(Data Management), 트랜잭션 생성기(Transaction Generator)등의 흐름을 보여준다. 각각에 대한 역할은 다음과 같다.

4.2 모의 실험 결과 및 분석

기존 알고리즘과 제안한 알고리즘을 충돌율(%)에 따라 트랜잭션이 완료된 수를 트랜잭션 생성 수로 나눈 비율이다. 트랜잭션 발생기에서 트랜잭션의 생성은 지수분포(exponential)를 따라 생성하기 때문에 각각에 따라 트랜잭션의 생성수가 다르다. 따라서 완료된 트랜잭션의 수를 비교하는 것보다는 완료된 트랜잭션의 수 비율로 비교하는 것이 보다 객관적이다.

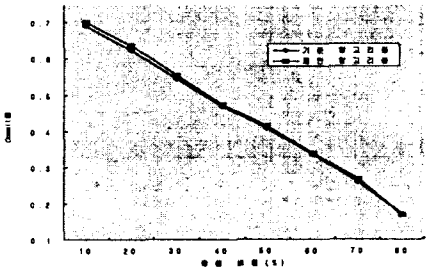

그림 4-2 충돌율(%)변화에 따른
완료(Commit)된 트랜잭션의 수 비율

그림 4-2에서는 기존 알고리즘과 제안 알고리즘이 충돌율 변화에 따른 트랜잭션의 완료율을 비교하여 보여준다. 충돌율 전반에 걸쳐 제안 알고리즘의 완료율이 우수하다는 것을 볼 수 있다. 충돌비율(%)이 높아질수록 제안 알고리즘과 기존 알고리즘의 완료율 차이가 점점 줄어 드는 것은 트랜잭션당간에 충돌수가 증가할수록 충돌로 인해 종료시한을 어긴 트랜잭션의 수가 증가하기 때문이다. 대체버전 병행제어 알고리즘에서는 충돌이 일어나면 조건 없이 그와 동일한 수행을 하는 트랜잭션이 버전화되어 생성하여 재시작하게 된다. 동일한 수행을 하는 트랜잭션이 많이 생성된다는 것은 충돌과 관계없는 트랜잭션이 실행할 수 있는 기회를 상실하게 되어 전반적으로 처리율이 저하된다. 제안 알고리즘은 동일한 수행을 하는 트랜잭션의 생성 수를 제한함으로써 충돌과 관계없는 트랜잭션의 실행기회를 부여함으로써 시스템 전체 처리율을 향상 시킬 수 있었다.

V. 결 론

대체 버전 병행제어 기법은 2PL-HP의 문제점인 낭비적 재시작과 낭비적 수행 문제를 해결하기 위해 제안되었다. 이 방식은 충돌이 자주 발생하는 응용 즉, 부하가 많은 응용에서는 불필요한 재시작 버전 트랜잭션의 생성과 블록킹이 많이 발생하고, 종료시한을 만족하기 힘들었다. 이는 트랜잭션 충돌시 재시작 버전 트랜잭션의 생성 수를 제한하지 않고 충돌이 발생할 때 마다 충돌 해결 정책을 사용하였기 때문이다. 본 논문은 실시간 데이터베이스 시스템이 단위 시간에 처리할 수 있는 최대 트랜잭션의 수를 고려하여 낭비적 재시작 트랜잭션의 수를 제한하기 때문에 충돌 연산과 관련이 없는 다른 트랜잭션에 보다 많은 실행의 기회를 줌으로써, 처리율을 향상한다. 또한 중단된 트랜잭션의 수가 자원을 낭비하는 것을 방지하여 자원 활용면에서도 효율이 증가한다.

참 고 문 헌

[1] G. Ozsoyoglu and R. Snodgrass, "Temporal and Real-Time Databases: A Survey," IEEE Transactions on Knowledge and Data Engineering, 7(4): pp.513-532, August 1995.

[2] R. Agrawal, M.J. Cargey and M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications," ACM Transaction on Database Systems, Vol.12, No.4, Dember 1987.

[3] T. P. Baker and A. Shaw, "The Cyclic Executive Model and Ada," In Proceedings of the IEEE Real-Time Systems Symposium, pp. 120-129, 1988.

[4] D. Hong, S. Chakravarthy, and T. Johnson, "Alternative Version Concurrency Control(AVCC) for firm real-time database system," Tech. Report UF-CIS-TR-95-031, Florida Univ., Oct. 1995.

[5] D. Hong, S. Chakravarthy, and T. Johnson, "Locking Based Concurrency Control for Intergrated Real-Time Database System," RTDB 96 Conference, California, Mar., 1996, Univ. of Florida, Dept of CISE, 1996