클라이언트-서버 DBMS에서 그림자 트랜잭션을 이용한 트랜잭션 캐쉬 일관성 유지 기법

(Transactional Cache Consistency Maintenance Scheme with Shadow Transaction in Client-Server DBMSs)

권 혁 민 †

(Hyeok Min Kwon)

요 약 트랜잭션간 캐싱을 허용하는 데이타전송 시스템은 각 클라이언트가 데이타베이스의 일부분을 동적으로 캐싱하므로 트랜잭션 캐쉬 일관성 유지 기법의 필요성을 야기한다. 지연 로킹 기법은 클라이언트가 액세스한 데이타에 대한 로크 설정 및 유효성 검사가 비동기적으로 이루어지는 검사기반 기법이다. 지연 로킹 기법은 매우 낮은 통신부담을 보이므로 높은 성능을 발휘할 수 있지만, 트랜잭션 철회율이 높은 단점이 있다. 이 단점에 대처하기 위하여 본 논문에서는 철회되는 트랜잭션 대신에 실행되기 위하여 관리되는 백업 목적의 트랜잭션인 그림자 트랜잭션의 개념을 제안한다. 그리고 이 개념과 지연 로킹 기법을 통합하여 그림자 트랜잭션을 이용한 새로운 트랜잭션 캐쉬 일관성 유지 기법을 제안한다. 그리고 모의실험을 통하여 제안된 기법의 성능과 검사기반 기법 중 가장 대표적인 적응적 낙관적 동시성 제어 기법과 캐싱 두단계 로킹 기법과의 성능을 비교한다.

Abstract A data-shipping system that allows inter-transaction caching raises the need of transactional cache consistency maintenance protocol, since each client is able to cache a portion of the database dynamically. Deferred locking(DL) is a detection-based scheme, in which locks and validity check of accessed cached copies are requested asynchronously. Due to its low communication overhead, it could show a superior performance, but it tends to show a high ratio of transaction abort. For coping with this drawback, this paper develops a new notion of shadow transaction, which is a backup-purpose one that is kept ready to replace an aborted transaction. This notion and the locking mechanism of DL have been incorporated into deferred locking with shadow transaction. Using a simulation model, the performances of the proposed schemes are compared with those of two representative detection-based schemes, the adaptive optimistic concurrency control and the caching two-phase locking.

1. 서 론

클라이언트-서버(client-server) DBMS에서 대부분의 객체지향 DBMS들은[1-3] 데이타전송(data-shipping) 방식을 채택하고 있다. 데이타전송 시스템은 질의가 클라이언트에서 처리될 수 있도록 필요한 데이타를 클라이언트로 전송한다. 이 방식은 DBMS 기능의 대부분을 클라이언트로 이관하므로 클라이언트의 자원들을 효율적으로 이용할 수 있어 공유되는 서버의 의존성을 줄일 수 있는 이점이 있다. 그러나 클라이언트가 다량의 데이타전송을 요청하면, 데이타전송 시스템은 네트워크의 사용집중에 따른 병목현상으로 인하여 심각한 성능장애를 야기할 수 있다. 트랜잭션의 완료후에도 클라이언트에 캐싱된 데이타를 계속 유지 관리하여 다른 트랜잭션들이 이를 사용하도록 허용하는 트랜잭션간 캐싱(inter-transaction caching)은 네트워크의 사용집중을 해결하기 위한 매우 효과적인 기술이다. 그러나 트랜잭션간 캐싱을 허용하면 각 클라이언트는 데이타베이스의 일부분을 동적으로 캐싱하므로 공유되는 데이타의 중복사본이 여러 클라이언트에 존재하게 된다. 그러므로 트랜잭션간 캐싱은 기본적으로 동시성 제어(concurrency control)와

† 정 회 원 : 세명대학교 소프트웨어학과 교수
hmkwon@venus.semyung.ac.kr
논문접수 : 2000년 4월 26일
심사완료 : 2000년 11월 1일

중복사본 관리(replica management)의 양면성을 지닌 트랜잭션 캐쉬 일관성 유지(transactional cache consistency maintenance: TCCM) 기법의 필요성을 야기한다. TCCM 기법은 트랜잭션들이 일관성 있게 중복사본을 액세스한다는 것을 보장해야 하므로 구현하기가 복잡하고 실행시키는데도 부담이 뒤따른다. TCCM 기법의 잠재적인 이점 및 이에 연관된 실행부담사이의 절충(trade-off)은 부하(workload)의 특징에 따라서 다르게 나타나기 때문에, DBMS의 성능은 채택된 TCCM 기법과 사용 환경에 따라 상당한 차이를 보일 수 있다.

따라서 다수의 TCCM 기법들이 제안되고 그들의 성능이 비교되었다[4-11]. 이들은 트랜잭션 실행결과의 정확성을 보장하는 방법의 차이에 따라 회피기반(avoidance-based)과 검사기반(detection-based) 기법으로 분류된다[7]. 회피기반 기법은 비최신의 데이타가 클라이언트 버퍼에 캐싱되는 것을 방지하여 트랜잭션이 유효하지 않은 데이타를 액세스하는 것을 원천적으로 봉쇄하는 반면, 검사기반 기법은 비최신의 데이타가 일시적으로 클라이언트 버퍼에 캐싱되는 것을 허용한다. 회피기반 기법은 일반적으로 ROWA(read-one write-all)[12] 방식을 채택한 분산 로킹 기법에 기초하고 있기 때문에 읽기 연산은 매우 효율적으로 진행되나, 최소한 트랜잭션이 완료하기 전에는 그의 갱신결과를 다수의 클라이언트 버퍼에 원자적으로 반영해야 하는 부담이 있다. 검사기반 기법은 이와 같은 부담을 야기하지 않지만 트랜잭션이 비최신의 데이타를 액세스할 가능성이 있다. 따라서 트랜잭션이 완료하기 전까지는 자신이 액세스한 모든 데이타에 대한 유효성을 보장받아야 한다.

회피기반 기법에서 한 트랜잭션의 일관성 유지를 위한 실행에는 다수의 클라이언트와 서버가 관여하는 반면, 검사기반 기법에서는 그 트랜잭션을 제기한 단일 클라이언트와 서버만이 관여한다. 그러므로 검사기반 기법은 회피기반 기법에 비해서 구현이 용이하며 고장에 매우 견고하다고 할 수 있다. 후자의 장점은 비교적 고장에 견고하지 못한 클라이언트를 고려할 때 매우 바람직한 특징이라 할 수 있다. 그렇지만 검사기반 기법은 중복사본의 유효성 검증 시기 및 방법에 따라 높은 통신부담으로 인하여 낮은 성능을 보이거나, 또는 지연된 유효성 검사로 인하여 높은 트랜잭션 철회율을 보이는 단점이 있다[4-8]. 본 논문은 이와 같은 단점을 극복하기 위하여 그림자 트랜잭션의 개념을 제안하고, 이를 비동기적으로 중복사본의 유효성을 검사하는 기법에 적용하여 낮은 철회율을 보이면서 우수한 성능을 발휘하는 검사기반 기법을 제안한다.

지연 로킹(deferred locking: DL) 기법은 중복사본의 관리 차원에서는 비동기적으로 유효성 검사를 실시하며, 동시성 제어 측면에서는 로킹 방법을 채택한다. 이 기법은 주사본 로킹 기법에 기초하고 있지만, 클라이언트에서 액세스된 중복사본에 대한 유효성 검사 및 로크 설정의 요청은 캐쉬미스(cache miss)로 인하여 클라이언트에서 서버로 메시지를 보낼 필요가 있을 때까지 지연된다. DL은 비동기적으로 중복사본의 유효성 검사 및 로크 설정을 서버에게 요청하기 때문에 낮은 통신부담을 보이지만 트랜잭션 철회율이 높은 단점이 있다. 이 단점은 트랜잭션이 전적으로 프로그램 제어하에 실행되거나, 또는 사용자에 의해 제기된 트랜잭션이 철회시에도 더 이상의 입력이 없이 다시 실행될 수 있는 응용환경에서는, 트랜잭션 철회로 인하여 시스템의 성능이 저하되지 않는 한 큰 문제가 되지 않는다. 그러나 트랜잭션이 사용자와 상호 협력적으로 진행되는 사용자 대화식의 응용분야에서 트랜잭션 철회는 사용자 작업시간의 심각한 낭비를 초래하므로 바람직하지 않다. 이와 같은 응용분야에서는 설사 시스템의 부담을 가중시키더라도 트랜잭션 철회의 부정적인 영향을 줄이기 위한 방법이 필요하다. 이와 같은 관점에서 본 논문은 철회되는 트랜잭션 대신 실행되기 위하여 관리되는 백업 목적의 트랜잭션인 그림자 트랜잭션의 개념을 제안한다. 그리고 이 개념과 DL의 로킹 기법을 통합하여 DL-ST(deferred locking with shadow transaction)라는 새로운 기법을 제안한다. 이 논문의 구성은 다음과 같다. 2장에서 기존에 제안된 TCCM 기법들을 살펴보고, 3장에서 본 논문에서 제안된 기법에 대하여 설명한다. 그리고 4장에서 제안된 기법의 성능 평가를 위한 모의실험 모형을 설계하고, 5장에서 실험 결과를 제시하고 분석한다. 마지막으로, 6장에서 본 논문의 결론을 맺는다.

2. 관련 연구

이 장에서는 본 논문의 성능 평가시 선정된 기법들을 중심으로 기존에 제안된 기법들을 살펴본다. 본 논문은 페이지 서버의[6, 13] 관점에서 각 기법을 설명한다. 즉, 동시성 제어와 데이타전송의 단위가 데이타 페이지라고 가정한다.

검사기반 기법은 중복사본의 유효성을 검사하는 시점에 따라 동기적 그리고 비동기적 기법으로 분류된다. 동기적 기법의 가장 대표적인 알고리즘인 C2PL(caching two-phase locking)은[5, 6] 클라이언트의 중복사본을 액세스하기 전에 그의 유효성을 검사하며 서버의 중앙 로크표에 적절한 로크를 동기적으로 설정한다. C2PL은

중복사본의 로그 일련 번호(log sequence number: LSN)와 서버의 LSN을 비교하여 중복사본의 유효성 여부를 검사한다. 이 기법은 매 액세스마다 중복사본의 유효성을 검사하며 적절한 로크를 설정하기 때문에 트랜잭션 철회율이 낮으나 심각한 통신부담을 야기하는 단점이 있다. 비동기적으로 유효성을 검사하는 기법으로는 AOCC(adaptive optimistic concurrency control)[4, 8]가 있는데, 이 기법은 트랜잭션의 실행단계에서는 어떤 제한도 가하지 않고 중복사본의 액세스를 허용한다. 대신 트랜잭션의 완료단계에서 액세스한 데이타의 유효성을 검증하는데, 이를 위해 각 트랜잭션이 읽고 갱신한 데이타에 대한 정보를 유지 관리한다. 서버는 트랜잭션의 완료단계에서 그 트랜잭션이 읽은 데이타 중에서 무효화된 데이타가 있는지를 조사하여 그의 완료 및 철회를 결정한다. AOCC는 데이타 충돌의 해결을 전적으로 트랜잭션 철회에 의존하므로 철회율이 높은 단점이 있지만, 중복사본의 유효성 검사를 위한 통신부담을 야기하지 않기 때문에 우수한 성능을 발휘하는 것으로 알려져 있다.

회피기반 기법은 한 트랜잭션의 갱신결과를 다수의 클라이언트 버퍼에 원자적으로 반영한다. 회피기반의 대표적인 알고리즘으로 O2PL(optimistic two-phase locking)[5-7, 14]과 CBL(callback locking)[5-7, 10] 기법이 있다. O2PL은 트랜잭션의 완료단계에서 갱신결과를 파급시키는 반면, CBL은 실제 갱신 연산이 제기된 순간에 그 결과를 파급시킨다. O2PL은 ROWA 방식을 채택한 분산 낙관적 로킹[14] 기법에 기초하고 있다. 각 클라이언트는 트랜잭션의 실행단계에서는 지역 로크표에 적절한 로크를 설정하고 중복사본을 액세스하면서 연산을 실행한다. 트랜잭션이 완료단계에 도달하면 서버는 그 트랜잭션이 갱신한 데이타를 캐쉬하고 있는 모든 클라이언트와 협조하여 그 갱신결과를 각 클라이언트 버퍼에 원자적으로 반영한다. [5-7]에서 제안된 O2PL 기법중에서 변경될 데이타를 클라이언트 버퍼에서 제거하는 무효화 정책을 채택한 O2PL-invalidation (O2PL-I)은 다양한 부하환경에서 우수한 성능을 발휘한다. O2PL에서 각 클라이언트에 설정된 로크들 사이의 충돌은 트랜잭션의 완료단계에 가서야 감지되며, 갱신 트랜잭션이 완료를 시도할 때마다 캐쉬 일관성 유지를 위한 실행이 여러 클라이언트에서 실행된다. 이 요인이 성능에 얼마나 큰 영향을 미치는가를 파악하기 위해 다음 예를 살펴보자.

예 1(O2PL에서 완료단계의 부담): 클라이언트 C_1에서 x를 갱신한 트랜잭션 T_1이 완료단계에 도달했다고 가정하자. 그리고 C_2와 C_3은 x를 캐쉬하고 있고, C_3에서 실행중인 T_3은 x에 대하여 이미 읽기 연산을 수행했다고 가정하자. 서버는 T_1의 완료요청 메시지를 받으면, x를 캐쉬하고 있는 C_2와 C_3으로 완료준비 메시지를 전송한다. C_2는 x에 로크를 설정하지 않았기 때문에 즉시 승인 메시지를 보낸다. T_1은 C_3에서 승인 메시지를 받아야 최종적으로 완료하는데, C_3은 T_3이 x의 읽기 로크를 해제하기 전에는 승인 메시지를 전송할 수 없다. 만일 T_3이 방금 전에 실행을 시작한 트랜잭션이라면 이 지연은 매우 길어질 수도 있는데, 이 지연은 완료를 시도하는 T_1에 그대로 반영된다. O2PL-I는 불필요한 무효화를 야기할 가능성도 있다. 위의 실행에서 C_2는 승인 메시지를 보내기 전에 자신의 버퍼에서 x를 제거한다. 그런데, 만일 나중에 T_1이 C_3에서 대기로 인한 교착상태가 발생하여 희생자로 선택되어 철회된다면, C_2의 버퍼에서 x를 제거한 것은 불필요한 무효화가 될 것이다. 무효화를 구현하기 위해 두단계 완료(two-phase commit) 규약을 사용하면 이와 같은 잘못된 무효화를 방지할 수 있지만, 이는 더 많은 메시지 부담을 야기할 것이다.■

CBL 기법은 O2PL과는 다르게 갱신 연산에 대한 권한을 트랜잭션 실행중에 전역적으로 승인 받는다. 따라서 완료단계에서는 오직 그 트랜잭션을 제기한 클라이언트와 서버만이 관여하므로 완료부담은 적은 편이다. 그러나 트랜잭션의 실행단계에서 전역적인 갱신 권한을 획득하는 과정에서 예 1과 같은 통신부담 및 불필요한 무효화가 발생된다. 회피기반 기법은 클라이언트의 캐쉬 일관성을 항상 유지시켜 읽기 연산은 매우 효율적으로 실행되며 철회율이 낮은 장점이 있다. 그러나 어떤 트랜잭션이 완료하기 전에는 그 트랜잭션의 갱신결과를 다수의 클라이언트 버퍼에 원자적으로 반영해야 하므로 쓰기 연산을 위한 부담이 상당히 큰 단점이 있다.

3. 캐쉬 일관성 유지 기법: DL과 DL-ST

본 논문은 중복사본의 유효성 검사를 위하여 데이타베이스의 각 페이지에는 그 상태를 파악하기 위한 로그 일련 번호(log sequence number: LSN)가 태그되어 있다고 가정한다. 실제로 회복을 위해 로그 우선 기록을 사용하는 시스템에서 데이타베이스의 각 페이지는 LSN이 태그되어 있다. DL 기법은 이 LSN을 이용하여 중복사본의 유효성을 검사한다. 그리고 각 트랜잭션은 교착상태 검출시에 희생자를 선정하기 위하여 타임스탬프를 가지고 있다고 가정한다.

3.1 DL의 기본 개념 및 로킹 기법

DL 기법은 주사본 로킹 기법에 기초하고 있기 때문에 각 트랜잭션은 서버의 중앙 로크표에 적절한 로크를 설정해야 하지만, 로크의 설정이 반드시 데이타를 액세스하기 전에 실행되는 것은 아니다. DL은 클라이언트에 캐싱된 중복사본의 액세스를 일단 허용하고, 그에 대한 로크 설정 및 유효성 검사는 추후에 클라이언트와 서버 사이에 정보교환이 필요한 시점에서 요청되어 비동기적으로 이루어진다. 캐쉬미스가 발생하면 클라이언트는 서버로 데이타전송 요청 메시지를 보내는데, 이때 클라이언트는 전 캐쉬미스 이후 현재까지 액세스한 페이지 각각에 태그되어 있는 LSN과 그 페이지에 대한 로크 요청을 첨부하여 서버로 전달한다. 서버는 첨부된 로크들을 설정하기 전에 해당 페이지에 대한 클라이언트의 LSN과 서버의 LSN을 비교하여 클라이언트가 유효한 데이타를 액세스했는지 여부를 검사한다. 만일 트랜잭션이 유효하지 않은 데이타를 액세스했다면 철회된다. 중복사본의 유효성이 검증되고 요청된 로크들이 전부 설정되면, 서버는 목표 데이타를 찾아 클라이언트로 전송한다. DL에서 서버는 교착상태 검출을 위한 대기그래프(wait-for-graph)를 유지 관리한다. 교착상태는 여기에 관여된 가장 늦게 시작된 트랜잭션을 철회하여 해결한다. 공정성을 위해, 로크 요청은 뒤에서 설명할 규칙 1 이외에는 선입선출 원리에 의해 먼저 요청된 다른 로크와 호환성이 성립해야 인증된다. 이를 위해 DL에서는 보통 로크 요청의 순서대로 로크표를 유지 관리한다. DL은 세 가지의 로크 유형을 사용한다: 읽기 로크(read lock: R-lock), 쓰기 로크(write lock: W-lock), 그리고 완료 로크(commit lock: C-lock). 이들 사이의 호환성 표는 표 1에 있다. 본 논문은 로크를 인증(granting) 모드로 설정한 트랜잭션뿐만 아니라, 대기(waiting) 모드로 설정한 트랜잭션도 해당 데이타에 대한 로크 소유자로 표현한다.

클라이언트는 자신의 버퍼에서 갱신 연산을 실행하고, 이 연산과 연관된 쓰기 로크는 서버로 전송될 다음 메시지에 첨부되어 요청된다. 서버는 W-lock을 항상 대기 모드로 설정하는데 이는 나중에 요청되는 R-lock이 W-lock보다 먼저 인증될 수도 있기 때문이다(규칙 1 참고). W-lock을 대기 모드로 설정하지만, 해당 트랜잭션은 대기하지 않고 계속 실행한다. 이 대기는 트랜잭션이 완료단계에 도달할 때까지 지연된다. 클라이언트 트랜잭션이 완료단계에 도달하면, 자신이 갱신한 새로운 데이타를 완료요청 메시지에 첨부하여 서버로 전송한다. 서버는 트랜잭션의 완료단계에서 그 트랜잭션의 모든 W-lock을 C-lock으로 변경하면서 인증 여부를 검사한다. 각 C-lock의 인증 여부는 로크표의 상태에 따라 결정된다; DL에서 로크표를 로크 요청의 순서대로 유지하므로 C-lock 앞에 어떤 로크도 존재하지 않으면 그 로크는 인증 가능하다. 완료를 원하는 트랜잭션은 자신의 모든 C-lock이 인증될 때까지 대기한다. 모든 C-lock이 인증되면, 서버는 클라이언트에서 갱신되어 전송된 페이지의 LSN을 갱신하고 그 페이지를 데이타베이스에 반영한 다음 로크를 해제한다. 그리고 나서 서버는 완료승인 메시지에 갱신된 페이지의 LSN을 첨부하여 해당 클라이언트로 전송한다. 클라이언트는 이 LSN을 자신의 버퍼에서 갱신된 페이지에 반영하여 중복사본의 LSN과 서버의 LSN을 동기화시킨다.

표 1 로크 호환성 표

소유자 \ 요청자	Rlock-I	Rlock-E	W-lock
R-lock	C	C	I, NB
W-lock	I, B	규칙 1	I, NB
C-lock	I, B	I, AR	I, NB

표시-
C: 호환; I: 비호환;
B: 요청자 대기; NB: 요청자 비대기 상태;
AR: 요청자 철회; 규칙 1: 본문 내용 참고.

클라이언트는 데이타전송 메시지를 통하여 이미 액세스한 데이타에 대한 R-lock과 현재 요청되는 데이타에 대한 R-lock의 설정을 서버에게 요청한다. 이 두 유형의 R-lock은 전자는 이미 그 연산을 실행했고, 후자는 아직 연산을 실행하지 않았다는 점에서 다르기 때문에 서로 구분되어야 한다. 본 논문은 전자의 R-lock을 Rlock-E(Rlock-Explicit)라 명명하고 후자를 Rlock-I (Rlock-Implicit)라 명명한다. 이 R-lock들은 일단 서버의 로크표에 설정되면 동일한 R-lock으로 취급된다. Rlock-I는 기존의 읽기 로크와 성격이 동일하므로 일반의 두단계 로킹(two phase locking: 2PL) 기법과 동일하게 스케쥴링한다. Rlock-E는 그와 연관된 읽기 연산이 이미 실행되었기 때문에 이와 충돌되는 C-lock이 존재하면 해당 트랜잭션은 철회된다. 만일 충돌되는 W-lock이 존재하면, Rlock-E는 다음 규칙에 의하여 스케쥴링된다.

규칙 1(W-lock에 대한 Rlock-E 스케쥴링): Rlock-E를 요청한 트랜잭션의 타임스탬프가 이 로크와 충돌하는 W-lock을 설정한 모든 트랜잭션들의 타임스탬프보다 작다면 그 Rlock-E는 인증된다. 그렇지 않으면 Rlock-E를 요청한 트랜잭션은 철회된다.

규칙 1은 DL의 로크 승인에 있어 FIFO 원칙을 위배

하는 유일한 경우로서, Rlock-E와 W-lock의 충돌을 해결할 때 FIFO 원칙을 반드시 준수하기 위하여 Rlock-E를 요청한 트랜잭션을 무조건 철회하기보다는 더 오래된 트랜잭션에게 우선권을 주기 위해 사용되었다. 충돌되는 W-lock에 대해서 Rlock-E가 인증된다는 것이 W-lock을 설정한 트랜잭션의 철회를 의미하지는 않는다. 그러나 W-lock을 설정한 트랜잭션은 완료단계에 도달하면 자신의 W-lock을 C-lock으로 변경한 후 인증 여부를 검사하는데, 이때 그 읽기 로크로 인하여 C-lock의 인증이 지연될 수는 있다.

DL 기법은 트랜잭션 실행결과의 정확성에 대한 판단 기준으로 DBMS에서 가장 보편적으로 받아들여지는 직렬화가능성(serializability)을[12] 채택하고 있다. DL은 완료단계에서 트랜잭션의 갱신결과를 반영한 후에 그 트랜잭션의 로크를 일괄적으로 해제하므로 두단계 로킹 규약을 준수한다. DL은 C-lock을 인증받은 후에 트랜잭션의 갱신결과를 실제 데이타베이스에 반영하므로 알고리즘의 정확성 관점에서 C-lock은 2PL에서의 W-lock과 동일한 역할을 한다. DL에서 W-lock은 성능을 위한 것이지 알고리즘의 정확성과는 관련이 없고 Rlock-I는 2PL의 R-lock과 성격이 동일하다. 그러므로 트랜잭션 실행결과의 직렬화가능성 관점에서 DL 기법이 기존의 2PL과 다른 점은 Rlock-E를 도입했다는 것이다. Rlock-E에 대한 연산의 실행 순서를 살펴보면 '읽기연산 실행→R-lock 설정→R-lock 해제'의 순서로 실행된다. DL 기법은 Rlock-E 요청을 인증하기 전에 중복사본의 LSN과 서버의 LSN을 비교하여 중복사본이 유효하지 않다면 Rlock-E를 요청한 트랜잭션을 철회한다. 만일 트랜잭션이 철회되지 않았다면 액세스한 중복사본이 유효하다는 것을 의미한다. 그러므로 읽기 연산을 실행하고 나서 R-lock을 설정하는 순서로 실행된 결과는 R-lock을 설정하고 나서 읽기 연산을 실행했더라도 그 결과가 동일하다고 볼 수 있다. 그러므로 DL 기법은 성공적으로 완료한 트랜잭션 실행결과의 직렬화가능성을 보장한다.

3.2 그림자 트랜잭션의 기본 개념

DL 기법의 가장 큰 단점은 트랜잭션 철회율이 높다는 것이다. 그러나 이 요인도 트랜잭션 대기와 철회사이에 균형을 유지시켜 성능을 향상시키는데 일조를 할 수 있다. 특히 높은 데이타 충돌 환경에서 어느 정도의 철회는 빈번한 로크 충돌로 인하여 과다한 수의 트랜잭션들이 대기상태에 빠지는 현상을 완화시켜 주기 때문에 성능에 크게 기여한다고 알려져 있다[5, 15]. 그러나 모든 환경에서 성능만이 유일하게 중요한 요소는 아니다. 예를 들어, 사용자와 상호 협조적으로 진행되는 트랜잭션이 많이 제기되는 환경에서는 사용자의 작업시간의 양도 시스템 성능만큼 중요하다. 이와 같은 관점에서 설사 시스템의 부담을 가중시키더라도 트랜잭션 철회의 부정적인 영향을 줄이기 위한 방법이 필요하다. 그림자 트랜잭션의 기본 개념은 트랜잭션의 철회에 대비하여 적절한 시기에 그 트랜잭션과 동일한 결과와 상태를 갖는 백업 목적의 트랜잭션인 그림자 트랜잭션을 준비하고, 만일 원래의 트랜잭션이 철회되는 경우에는 그 트랜잭션을 처음부터 다시 실행시키는 대신에 그림자 트랜잭션을 실행시켜 낭비되는 작업량을 줄이자는 것이다. 본 논문에서는 트랜잭션이 직렬화가능성을 위반할 가능성이 있는 경우에 그림자 트랜잭션을 생성한다.

3.3 그림자 트랜잭션을 이용한 지연 로킹 기법

그림자 트랜잭션의 개념과 DL 기법을 통합한 DL-ST(deferred locking with shadow transaction)는 클라이언트의 중복사본을 액세스하는 것을 직렬화가능성을 위반할 잠재적 위협으로 간주한다. 그러므로 중복사본을 액세스하기 전에 현재 실행되고 있는 트랜잭션을 위해 그림자 트랜잭션을 생성한다. 그림자 트랜잭션은 원래의 트랜잭션이 액세스한 중복사본에 대한 유효성 여부가 파악될 때까지 대기상태에 둔다. 원래의 트랜잭션은 클라이언트의 중복사본은 항상 유효하다고 가정하고 이를 액세스하면서 자신의 연산을 실행한다. 그러므로 이 트랜잭션은 낙관적 트랜잭션으로 간주되는 반면, 그림자 트랜잭션은 중복사본이 유효하지 않다고 가정하고 대기상태에 있기 때문에 비관적 트랜잭션으로 간주된다. 추후, 액세스한 중복사본이 유효하지 않다고 판정되면, 원래의 트랜잭션이 철회되어야 한다. 이때 그림자 트랜잭션들중 하나가 새로운 낙관적 트랜잭션으로 선택되어 원래의 트랜잭션을 대신하여 실행된다. 이 과정을 더 명확히 설명하기 위하여 다음의 실행 스케줄을 살펴보자.

예 2(그림자 트랜잭션 개념의 적용): 트랜잭션 T_i가 그림 1의 시간 t_2까지 성공적으로 실행되었다고 가정하자.

그림 1에서 op_{k-1} 형태의 연산은 캐쉬미스로 인하여 서버의 로크표에 적절한 로크를 설정하고 데이타를 전송받은 후에 실행되는 연산을 의미하며, op_k 형태의 연산은 캐쉬히트가 발생하여 중복사본을 액세스하면서 실행되는 연산을 의미한다. 그리고 $T_{i(k)}$는 T_i가 k번째 연산을 실행한 직후에 생성한 T_i의 그림자 트랜잭션임을 의미한다. 그러므로 $T_{i(k)}$는 T_i가 k번째 연산을 실행한직후까지 산출한 동일한 결과와 상태를 가지고 있다. op_k 연산은 중복사본을 이용하여 실행되므로 T_i는 이 연산

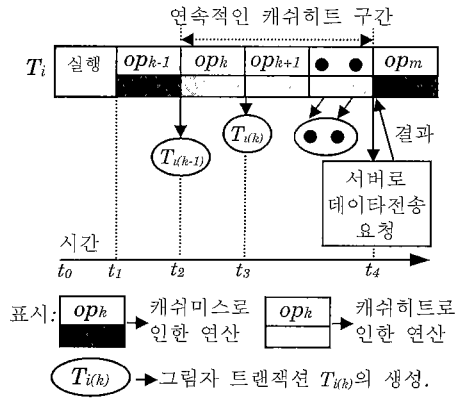

그림 1 그림자 트랜잭션의 생성

의 실행 직전에 $T_{i(k-1)}$을 생성한다. 이와 같이 T_i는 캐쉬히트를 만날 때마다 새로운 그림자 트랜잭션을 생성한다.

T_i가 t_4에서 캐쉬미스를 겪는다고 가정해 보자. 이 경우 T_i는 op_m 연산을 수행하기 위하여 서버로 데이타 전송 요청을 한다. 이 요청에는 t_2에서 t_4까지 액세스한 중복사본에 대한 로크 요청 및 그의 LSN이 첨부되어 전송된다. 서버는 클라이언트의 LSN에 기초하여 T_i와 그의 그림자 트랜잭션들의 운명을 결정하여 그 정보를 클라이언트로 전송한다. 예를 들어, T_i가 연산 op_{k+1}의 실행시에 비최신의 데이타를 액세스했다면, T_i는 철회되어야 하며, $T_{i(k)}$가 대기상태에서 깨어나 T_i를 대신하여 실행되어야 한다는 정보를 클라이언트로 전송한다. 이 정보를 보낼 때, 서버는 op_{k+1} 연산을 위한 최신의 페이지를 첨부하여 클라이언트로 전송한다. 이 경우에 T_i의 다른 그림자 트랜잭션들은 $T_{i(k)}$보다 덜 진행되었거나, 또는 그들도 op_{k+1} 연산의 실행시에 비최신의 데이타를 액세스했기 때문에 제거된다. 만일 T_i가 비최신의 데이타를 액세스하지 않았다면 T_i의 모든 그림자 트랜잭션들은 제거된다.■

DL-ST의 로킹 방법은 DL과 거의 비슷하다. 그들의 로킹 기법에서 유일한 차이는 Rlock-E의 스케쥴링 방법에 있다; Rlock-E의 요청이 유효하지 않은 중복사본의 액세스로 인하여 인증될 수 없다면, DL에서는 그 트랜잭션이 철회되어 처음부터 다시 실행되는 반면, DL-ST에서는 그의 그림자 트랜잭션들중의 하나가 대기상태에서 깨어나 실행된다. 마찬가지로 Rlock-E를 요청한 트랜잭션이 충돌되는 C-lock이나 W-lock으로 인하여 철회될 경우에도 그림자 트랜잭션들중 하나가 원 트랜잭션을 대신하여 실행된다. 그림자 트랜잭션의 개념은 교착상태로 인한 트랜잭션 철회시에도 적용될 수 있다. 예를 들어 그림 1에서 T_i가 op_k 연산의 실행으로 인하여 교착상태가 발생하였다면 $T_{i(k-1)}$이 T_i 대신 실행할 수 있을 것이다. 물론 이 경우에는 op_k 연산을 실행하기에 앞서 서버로부터 로크를 인증받고, 필요시에는 새로운 데이타를 전송받아야 한다. 그렇지 않으면 다시 교착상태에 빠질 수 있다. 그렇지만 t_1 시점 이전에 실행된 연산의 결과로 교착상태가 발생되었다면 T_i는 처음부터 다시 실행되어야 한다. 만일 생성된 그림자 트랜잭션을 계속 유지하면 모든 교착상태에 대하여 그림자 트랜잭션의 개념을 적용할 수는 있지만, 이는 그림자 트랜잭션의 결과와 상태를 저장하기 위한 심각한 공간부담을 야기할 것이다. 따라서 본 논문에서는 교착상태의 경우까지 그림자 트랜잭션의 개념을 확장하지 않는다. 로킹 기법에서 충돌되는 로크들의 인증 순서는 그 로크를 요청한 트랜잭션간의 직렬화 순서(serial order)를 결정하므로 매우 중요하다. 그리고 이미 실행된 연산의 정확성은 로크에 의해 보장된다. 그러므로 그림자 트랜잭션의 결과는 그 트랜잭션이 생성될 때까지 원래의 트랜잭션을 위해 설정된 또는 설정될 것이라고 예상되는 로크에 의해 정확성이 보장되는 것이다. 그러므로 트랜잭션 T_i가 철회되어 그의 그림자 트랜잭션들중의 하나로 대치될 때, 새로이 낙관적 트랜잭션으로 선택된 트랜잭션은 T_i의 로크를 계승받는다.

만일 트랜잭션이 n개의 데이타를 액세스한다면, 그 트랜잭션을 위해서 최대 n개의 그림자 트랜잭션이 존재할 수 있다. 이는 그림자 트랜잭션의 생성 및 그 결과를 저장하기 위하여 심각한 실행부담과 버퍼 공간부담을 야기할 것이다. 그러므로 시스템의 성능이 심각하게 저하되지 않도록 그림자 트랜잭션의 수를 제한하는 것이 바람직하다. 이와 같은 관점에서 본 논문은 하나의 트랜잭션을 위하여 한 순간에는 최대 k개까지 그림자 트랜잭션을 허용하고, 이 기법을 DL-ST/k라 명명한다. DL-ST/k에서 어느 경우에 그림자 트랜잭션을 생성할 것인가는 매우 흥미있는 연구가 될 것이나, 본 논문은 DL-ST/k의 간략화를 위해 그림자 트랜잭션의 수가 k에 도달할 때까지 매 캐쉬히트마다 그림자 트랜잭션을 생성하는 정책을 채택한다. 데이타 충돌이 심한 환경에서 k 값을 크게 설정하는 것이 바람직하다. 그러나 낮은 충돌환경에서 큰 k 값은 별다른 이점도 주지 못하면서 단지 그림자 트랜잭션과 연관된 부담만을 야기한다. 이와 같은 점을 고려하여 k 값은 데이타 충돌의 정도를 반영하여 한 시스템에서 동적으로 변경될 수도 있다. 그리고 k 값은 모든 트랜잭션에게 동일한 값으로 적용될

필요도 없다. 각 트랜잭션은 자신의 데이타 충돌 정도를 반영하여 자신만의 k 값을 가질 수도 있다. 본 논문에서 이와 같은 연구는 미래의 과제로 남겨둔다.

3.4 관련 연구들의 비교

검사기반 기법은 클라이언트가 비최신의 데이타를 캐싱할 수 있기 때문에, 중복사본의 유효성을 효율적으로 관리하기 위한 기술이 필요하다. 이를 위해, DL 기법과 AOCC는 전파(propagation) 방법과 무효화(invalidation) 방법을 조합하여 사용한다. 만일 트랜잭션이 비최신의 데이타를 액세스하여 서버에서 철회가 결정되면, 서버는 그 데이타의 최신 버전을 철회 메시지와 함께 해당 클라이언트로 전송한다. 만일 갱신 트랜잭션이 성공적으로 완료하면, 그 트랜잭션의 갱신결과는 서버의 데이타베이스와 그 트랜잭션을 제기한 클라이언트에만 반영된다. 따라서 그 갱신결과와 관련이 있는 다른 클라이언트의 중복사본들은 더 이상 유효하지 않다. 이와 같은 사항을 알리기 위하여 서버는 관련된 클라이언트로 해당 데이타의 무효화 메시지를 전송한다. 이 메시지를 받은 클라이언트는 즉시 해당 데이타를 버퍼에서 제거한다. 무효화 메시지는 서버에서 클라이언트로 전송되는 다른 메시지에 첨부되어 전달되므로 메시지의 전송 횟수에는 영향을 미치지 않는다. 무효화 메시지를 전송하기 위해 서버는 각 데이타마다 이를 캐싱하고 있는 클라이언트의 정보를 간직하기 위한 자료구조를 유지 관리하는데, 이는 회피기반 기법에서도 갱신결과를 어떤 클라이언트로 파급시킬 것인가를 파악하기 위해 필요하다. DL 기법은 이 자료구조에 각 데이타에 대한 서버의 LSN도 같이 저장하여 관리한다.

DL 기법은 트랜잭션의 실행중에는 중복사본의 유효성 검사를 위하여 별도의 통신부담을 야기하지 않고 한 트랜잭션의 완료를 위한 연산에는 단일 클라이언트와 서버만이 관여한다는 점에서 AOCC와 유사한 측면이 있다. 그러나 DL 기법과 AOCC는 동시성 제어 측면의 차이로 인하여 그림자 트랜잭션의 효과에 있어서 큰 차이가 있다. DL 기법에서는 캐쉬미스시에 어떤 트랜잭션 중간결과의 유효성이 성공적으로 검증되고 나면, 로크로 인하여 그 중간결과의 유효성은 이후에도 계속 보장된다. 그러므로 DL 기법에서 적절한 시기에 그림자 트랜잭션을 생성해 놓으면, 추후에 트랜잭션이 비최신의 데이타를 액세스하여 철회되면 그림자 트랜잭션이 원래의 트랜잭션을 대신하여 실행될 수 있다. 그러나 AOCC는 그림자 트랜잭션을 생성해 놓더라도 다른 트랜잭션이 그의 결과를 아무런 제한없이 무효화시킬 수 있기 때문에 그림자 트랜잭션의 효과가 그다지 크지 않다. OCC 기반 기법에서 철회율을 줄이기 위해서는 그림자 트랜잭션을 적용하기보다는 검증 알고리즘을 개선하여 더 많은 트랜잭션들을 완료할 수 있게 하는 것이 더 바람직할 것이다. 예를 들어, 트랜잭션들의 직렬화 순서를 타임스탬프와는 다르게 재순서화함에 의해 트랜잭션의 완료 가능성을 대폭 향상시킨 TSH[16] 또는 TSH-WN(TSH with write notification)[17]기법의 검증 알고리즘을 TCCM 기법에 적용할 수 있을 것이다.

DL은 로킹 기법을 사용하고 각 페이지의 LSN에 기초하여 중복사본의 유효성을 검증한다는 점에서 C2PL과 유사한 측면이 있다. 그러나 C2PL은 중복사본을 액세스하기 전에 서버에서 그의 유효성을 동기적으로 검사하는 반면, DL은 서버와 통신이 필요한 시점에서 중복사본의 유효성을 비동기적으로 검사한다는 점에서 다르다. NWL(no-wait locking)[10]에서 트랜잭션은 매 액세스마다 서버로 로크 설정을 요청하는데, 로크 응답을 기다리지 않고 로크가 정상적으로 인증된다고 가정하고 계속 실행한다. 만일 이 로크가 인증되지 않으면 그 트랜잭션은 철회된다. NWL은 데이타를 액세스한 후에 이와 연관된 로크 설정이 비동기적으로 이루어진다는 점에서 DL과 유사하지만, 중복사본의 매 액세스마다 서버로 로크 설정을 요청한다는 점이 다르다. NWL에서 트랜잭션이 비최신의 데이타를 액세스할 수 있기 때문에, [10]은 NWL을 확장하여 갱신 트랜잭션이 성공적으로 완료할 때마다 그의 결과를 각 클라이언트로 전파하는 NWL-WN(no-wait locking with notification)을 제안하였다. 그렇지만 NWL-WN은 전파를 위한 부담 때문에 평균적으로 NWL보다는 낮은 성능을 나타낸다. 무효화 방법을 사용하면 전파를 위한 부담을 상당히 줄일 수 있는데, [10]에서는 이에 대한 연구는 없었다.

NL(Notify Lock)[11]은 일종의 회피기반 기법으로 한 트랜잭션의 갱신결과를 완료단계에서 다수의 클라이언트로 전파시킨다는 점에서 O2PL-P(O2PL-Propagation)와 유사하다. 그러나 이 기법은 O2PL-P가 트랜잭션의 갱신결과를 반영할 때 다른 트랜잭션 T_1이 이미 해당 데이타를 로킹했다면 대기하는데 비해, NL은 선점유 방식을 사용하여 T_1을 철회하고 갱신결과를 전파한 점이 다르다. 그러므로 NL은 한 트랜잭션의 갱신결과가 다른 클라이언트에 원자적으로 반영되기 전에 그 트랜잭션은 완료 가능하다. 그러나 전체적인 메시지 부담에 있어서는 O2PL-P와 그다지 다르지 않다.

4. 모의실험 모델

이 장에서는 DL과 DL-ST 기법의 성능 평가를 위한

모의실험(simulation) 모델을 제시한다. 비교 대상으로 C2PL과 AOCC를 선정하였는데, C2PL은 DL과 같이 주사본 로킹 기법을 채택하고 있기 때문에 선정되었고, AOOC는 검사기반 기법중 성능이 우수한 것으로 알려져 있기 때문에 선정되었다. DL-ST의 대표격으로 DL-ST/1을 선정하였는데, 그 이유는 비록 하나의 그림자 트랜잭션을 허용하더라도 그림자 트랜잭션으로 인한 성능 절충 현상을 충분히 파악할 수 있기 때문이다. 회피기반 기법중 우수한 성능을 발휘하는 것으로 알려진 O2PL-I의 성능도 제시한다. 모의실험 모델은 폐쇄 큐잉(closed queuing) 모델을[5, 6] 참고로 하여, MCC에서 개발한 CSIM[18] 언어를 이용하여 구현하였다.

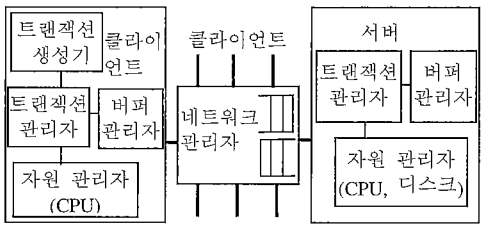

그림 2 모의실험 모델

모의실험 모델은 크게 네트워크 관리자, 클라이언트 모듈, 그리고 서버 모듈로 구성된다. 네트워크 관리자는 서버와 클라이언트의 정보교환을 관리하기 위하여 FIFO 큐 형태로 구현되었다. 트랜잭션 생성기는 한 트랜잭션이 성공적으로 종료하면 즉시 다음 트랜잭션을 생성하여 시스템에 제기한다. 본 논문은 클라이언트만이 트랜잭션을 제기한다고 가정한다. 클라이언트와 서버의 트랜잭션 관리자는 서로 협조하여 트랜잭션의 실행을 관리하며, 적용된 각 TCCM 기법에 알맞게 동시성 제어 및 중복사본 관리의 실행을 책임진다. 자원 관리자는 FIFO 원리로 디스크와 CPU와 같은 자원의 할당을 관리한다. 본 논문은 클라이언트에는 디스크가 없고, 서버와 각 클라이언트는 각각 하나의 CPU를 소유하고 있다고 가정한다. 버퍼 관리자는 LRU 정책을 채택하여 구현되었다.

트랜잭션 및 시스템 자원, 그리고 실행부담에 관한 입력 변수들은 각각 표2, 3에 있다. 본 실험에서 데이타베이스는 DbSize의 페이지로 구성된다. MeanTranSize는 한 트랜잭션에 의해 액세스되는 평균 데이타 페이지 수를 정의하고, UpdateProb는 액세스된 페이지가 갱신될 확률을 정의한다. 트랜잭션들이 실제 액세스하는 페이지 수는 MeanTranSize의 20% 편차 구간에서 균등 분포

표 2 트랜잭션 및 시스템 자원 변수

입력변수	의 미	설 정
DbSize	데이타베이스 크기	1000 페이지
PageSize	페이지 크기	4096 바이트
MeanTranSize	평균 페이지 액세스 수	20 페이지
UpdateProb	갱신확률	20%
FakeRestartPct	위조 재실행 확률	20%
NumClients	클라이언트의 수	1-25 클라이언트
NumDisks	디스크의 수	4 디스크
ClientCPU	클라이언트 CPU 속도	15 MIPS
ServerCPU	서버 CPU 속도	30 MIPS
NetBandwidth	네트워크 대역폭	10 Mbits/sec
ClientBufSize	클라이언트 버퍼 크기	25% of DbSize
ServerBufSize	서버 버퍼 크기	50% of DbSize

표 3 시스템 부담 변수

입력변수	의 미	설 정
CtrlMsgSize	제어 메시지 크기	256 바이트
PageInst	페이지 읽기연산을 위한 명령수	30,000
FixedMsgInst	메시지 처리를 위한 고정명령수	20,000
PageMsgInst	페이지마다 추가되는 명령수	10,000
LockInst	로크 설정/해제를 위한 명령수	300
PageValInst	페이지 검증에 필요한 명령수	10~300
RegisterInst	캐쉬 사이트 조사/캐쉬 사이트 추가 및 삭제를 위한 명령수	300
ForkShadowInst	그림자 트랜잭션 생성 명령수	100,000
SpaceForShadow	그림자 트랜잭션의 공간부담	10 페이지
DiskOvhdInst	디스크 I/O를 위한 명령수	5000
MinDiskAcc	최소 디스크 액세스 시간	10 millisec.
MaxDiskAcc	최대 디스크 액세스 시간	30 millisec.

(uniform distribution)를 이룬다. 모의실험에서 트랜잭션의 재실행 정책에는 진짜 재실행(real restart: RR)과 위조 재실행(fake restart: FR) 방법이 있다. RR 정책에서 트랜잭션이 철회되면 그 트랜잭션이 처음부터 다시 실행된다. 반면, FR 정책에서는 원래의 트랜잭션이 다른 트랜잭션으로 대치되어 실행된다. 본 논문에서는 트랜잭션이 철회되면 사용자는 원래의 트랜잭션을 다시 실행하기보다 다른 트랜잭션의 실행을 원할 수도 있다는 점을 고려하여 FakeRestartPct로 명기된 위조 재실행 확률을 정의한다. 즉 트랜잭션이 철회되면, 그 트랜잭션은 FakeRestartPct의 확률로 다른 트랜잭션으로 대치되어 재실행된다. 클라이언트 CPU는 트랜잭션이 한 페이지를 읽을 때마다 PageInst의 명령수를 실행하며, 쓰기 연산은 이 두 배의 명령수 처리시간이 필요하다. 네트워크 전송속도는 10Mbps를 기본값으로 설정하였는데, 네트워크 속도 변화에 따른 성능추이도 살펴보았다. 메시지를 처리하기 위한 CPU의 부담은 각 메시지마다 반드시 필요한 FixedMsgInst와 메시지에 포함된 각 페

이지마다 필요한 PageMsgInst로 모델링된다. 이 명령수를 실행하는 부담은 메시지를 전송하는 사이트와 회신하는 사이트에서 모두 필요하다. 로크 및 데이타전송 요청과 같은 제어 메시지는 CtrlMsgSize로 정의되는데, DL기반 기법과 AOCC는 각 제어 메시지에 다른 정보들이 첨부되므로 이 크기를 두 배로 설정한다. 디스크 수는 4로 설정하며, 디스크 액세스 시간은 MinDiskAcc에서 Max DiskAcc까지 균등 분포를 이룬다. 트랜잭션이 성공적으로 완료하면, 서버는 그 트랜잭션이 갱신한 페이지를 자신의 버퍼에 반영하고 그 페이지에 변경 표지를 남겨 놓는다. 변경 표지가 설정된 페이지는 버퍼 교체 알고리즘에 의해 희생자로 선택될 때 디스크에 기록된다. 단일 디스크 입출력을 위해서는 DiskOhInst의 명령수 처리시간이 필요하다. 한 데이타의 로크를 위해 LockInst 수의 CPU 처리부담을 필요로 한다. DL기반 기법은 다수의 로크를 모아 다음에 전송될 메시지에 첨부하는데, 로크가 모아질 때마다 LockInst 수의 명령수 처리시간이 필요하다. AOCC는 트랜잭션의 읽기 집합에 속한 각 데이타가 무효화 정보에 포함되어 있는지를 조사하는 검증 방법을 채택하고 있다. 각 데이타의 검증을 위해서, 무효화 페이지가 적을 경우에는 페이지당 10 명령수, 무효화 페이지가 30이상이면 고정적으로 300 명령수 처리부담이 필요하다. 특정 데이타를 캐싱하고 있는 사이트를 조사하는 시간, 특정 데이타에 대한 새로운 캐싱 사이트의 추가 및 제거를 위한 처리부담은 RegisterInst로 정의한다.

그림자 트랜잭션은 기본적으로 어떤 트랜잭션의 복사본이므로 프로세스 포크(process fork)나 쓰레드 포크(thread fork)를 이용하여 구현할 수 있다. CVAX 단일 프로세서 시스템에서 Ultrix 널 프로세스를 생성하는데 11,300μs의 실행시간이 필요하고, 쓰레드를 포크하기 위해서는 948μs의 실행시간이 필요하다고 보고 되었다[19]. 본 논문은 프로세스 포크를 이용하여 그림자 트랜잭션을 생성한다고 가정하고, 그림자 트랜잭션을 생성할 때마다 ForkShadowInst의 명령수 실행부담을 클라이언트 CPU에 부과한다. ForkShadowInst의 값은 100,000으로 설정했는데, 이는 프로세스를 포크하고 20개의 데이타 페이지가 다 복사될 수 있도록 충분히 큰 값이다. 한 그림자 트랜잭션의 결과와 상태를 저장하기 위한 공간부담은 SpaceForShadow로 정의한다.

본 논문에서 사용된 주요 성능지수는 초당 처리되는 트랜잭션 처리율과 트랜잭션 응답시간인데, 폐쇄 큐잉 모델에서 두 성능 중 하나의 결과를 알면 다른 값은 계산을 통하여 파악할 수 있기 때문에 트랜잭션 처리율만을 제시한다. 트랜잭션 철회율과 완료되는 트랜잭션당 필요한 메시지 수 및 버퍼의 히트율과 같은 보조 성능지수들이 성능 결과를 분석하기 위하여 조사되었다. 그리고 트랜잭션 대기율이 조사되었는데, 이는 시스템에 제기되어 있는 전체 트랜잭션 대비 로크 인증을 기다리며 대기 상태에 빠진 트랜잭션 수의 비율을 의미한다. 본 모의실험은 복사(replication) 방법을[20] 사용하여 실행되었는데, 실험 시작시의 초기 편향(initial bias)을 제거하기 위하여 초기 800개의 트랜잭션 완료의 결과는 무시하였다. 본 논문의 결과값은 6개의 다른 임의 수를 사용하여 실시된 모의실험 결과의 평균값으로, 각 복사는 5000개의 트랜잭션이 완료할 때까지 실시되었다. 이렇게 산출된 결과는 95%의 신뢰 수준을 만족한다.

5. 실험 결과 및 분석

모의실험은 다양한 데이타 공유형태를 반영하기 위하여 UNIFORM, HIGHCON, 그리고 HOTCOLD라고 명명된 세 부하에서 실시되었는데 각 부하의 특징은 해당 절에서 설명한다. 본 논문은 O2PL-I의 전역 교착상태 검사 주기를 [5, 6]에서와 같이 1초로 설정한다. 그리고 한 트랜잭션의 갱신결과를 다수의 클라이언트에 브로드캐스팅(broadcasting) 방식으로 파급시키는데, 이 메시지를 보낼 때 필요한 부담은 하나의 제어 메시지를 보내는 것과 동일하게 설정한다. AOCC 기법에 그림자 트랜잭션을 적용한 AOCC-ST/1은 액세스하는 데이타 수의 관점에서 트랜잭션 평균 길이의 중간에서 그림자 트랜잭션을 생성한다.

5.1 실험 1- UNIFORM 부하

이 부하에서 트랜잭션은 데이타베이스에서 임의의 페이지를 선택하여 액세스한다. 그러므로 모든 페이지는 동일한 확률로 액세스된다. 이 부하는 낮은 접근 국부성(reference locality)을 보이며, 데이타 충돌의 정도는 HIGHCON과 HOTCOLD 부하의 중간 정도이다.

5.1.1 클라이언트 수에 따른 성능

트랜잭션 처리율과 트랜잭션 완료당 필요한 평균 메시지 수가 그림 3에 있다. 이 실험에서 대체로 DL-ST/1, DL, 그리고 AOCC 기법 순으로 우수한 성능을 발휘한다. AOCC-ST/1 기법은 AOCC와 거의 유사한 성능을 보인다. NumClients의 증가에 따라 각 기법의 성능은 향상되지만, 15-클라이언트 이상이 되면 성능이 저하되기 시작한다. C2PL은 15-클라이언트에서 로크 충돌로 인하여 29%의 트랜잭션 대기율을 보이는데, 20-클라이언트에서는 41%의 대기율을 보인다. 이는 클라이언트의 수가 5만큼 증가되지만, 실제 활성 트랜잭션의

수는 단지 1.15(20x0.59 - 15x0.71)개 증가된다. 이는 15-클라이언트 이상에서 NumClients의 증가는 동시성을 높이는데 기여하기보다는 데이타 충돌만을 가중시켜 이로 인한 성능저하를 야기한다는 것을 의미한다. 이런 관찰로부터 C2PL의 성능은 데이타 사용경쟁으로 인하여 제한된다고 할 수 있다. 이런 주장은 그림 4(a)에서 보는 바와 같이 15-클라이언트에서 서버 CPU와 네트워크 대역폭에 병목현상이 발생되지 않는다는 사실이 뒷받침을 해 준다. O2PL-I의 성능추이도 이런 맥락에서 해석된다. AOCC는 기본적으로 데이타 충돌로 인한 트랜잭션 대기를 야기하지 않고, DL과 DL-ST/1은 비교적 낮은 트랜잭션 대기율을 보이므로 시스템의 자원을 매우 효율적으로 사용할 수 있다. 그렇지만 이 기법들은 15-클라이언트 이상이 되면 네트워크 사용집중으로 인한 성능 병목현상이 발생하여 성능이 더 이상 증가하지 않는다.

그림 3(b)의 메시지 수의 변화추이를 살펴보면 회피기반 기법과 검사기반 기법은 뚜렷한 차이를 보인다. 회피기반 기법인 O2PL-I에서 NumClients가 증가함에 따라 메시지 부담이 꾸준히 증가한다. 이의 근본적인 원인은 트랜잭션의 완료단계에서 그의 갱신결과를 파급시켜야 할 클라이언트의 수가 증가하기 때문이다. 이 완료단계의 메시지 부담을 살펴보기 위하여 NumClients를 n이라 가정하자. 이 실험에서 각 트랜잭션은 평균 4 페이지를 갱신한다. 클라이언트의 버퍼 크기를 DbSize의 25%로 설정했기 때문에, 어떤 클라이언트가 갱신되는 4 페이지를 전혀 캐싱하고 있지 않을 확률은 $(1-0.25)^4$, 즉 0.32가 될 것이다. 그러므로 서버는 한 트랜잭션의 완료시에 하나의 무효화 메시지를 전송하고 (n-1)x(0.68) 수의 클라이언트로부터 응답 메시지를 받아야 한다. 그러므로 O2PL-I는 완료단계의 부담으로 인한 메시지 수가 NumClients 대비 약 0.68의 경사를 갖고 증가된다.

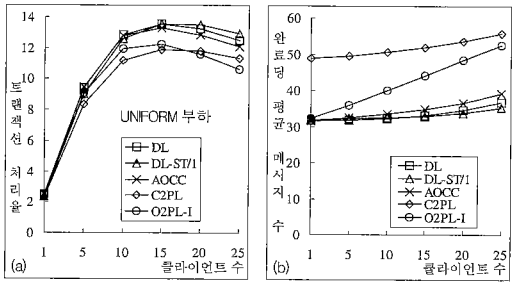

그림 3 (a) 트랜잭션 처리율 (b) 메시지 수

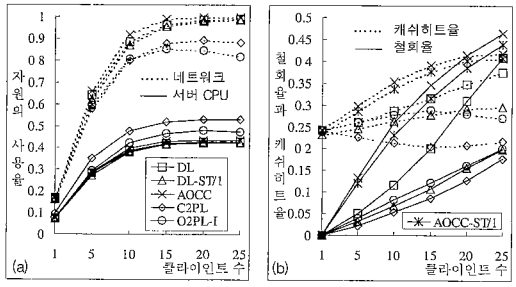

그림 4 (a) 자원 사용율 (b) 철회율과 캐쉬히트율

검사기반 기법에서 한 트랜잭션의 완료 연산에는 단일 클라이언트와 서버만이 관여하므로, NumClients가 증가해도 완료단계의 메시지 부담이 증가하지 않는다. 그렇지만, C2PL은 트랜잭션의 매 액세스마다 유효성 검사 및 적절한 로크 설정을 위하여 서버와 메시지 교환을 필요로 하므로 평균 메시지의 수가 크다.

Num Clients가 증가하면 데이타 충돌이 심화되므로 트랜잭션 철회율이 증가한다. C2PL과 O2PL-I에서 철회는 오직 로크 충돌에 의한 교착상태로 인하여 발생되므로, 그 증가율이 그다지 크지 않다. 그러나 DL과 AOCC에서는 NumClients의 증가에 따라 무효화된 데이타를 액세스할 가능성이 점점 높아지므로 철회율이 급격히 증가한다. 두 기법에서 트랜잭션은 캐쉬미스마다 자신이 액세스한 데이타의 유효성 검사를 실시하는데, DL에서는 일단 검증된 데이타의 유효성이 완료단계까지 보장되는데 반해, AOCC에서는 그렇지 않다. 그러므로 AOCC는 DL보다 더 높은 철회율을 보인다. 그림자 트랜잭션을 적용한 기법을 비교하면, DL-ST/1은 DL에 비해 50% 정도의 철회율을 보이는데, AOCC-ST/1과 AOCC는 철회율의 차이가 크지 않다. 이의 원인은 DL-ST/1에서 그림자 트랜잭션의 결과가 로크로 인해 유효성이 보장되는데 비해, AOCC-ST/1에서는 그 결과가 다른 트랜잭션에 의해 언제라도 무효화될 수 있기 때문이다. 비록 각 기법이 다른 철회율을 보이지만, 이것이 성능 및 메시지 부담에 큰 영향을 미치지 않는다. 왜냐하면 재실행되는 트랜잭션들은 그들이 필요로 하는 데이타를 이전 실행시에 자신의 버퍼로 옮겨 놓았을 가능성이 높기 때문이다. 캐쉬히트가 발생하면 트랜잭션 실행시에 클라이언트 CPU만이 사용되는데, 각 클라이언트에는 한 순간에 하나의 트랜잭션이 실행되므로 클라이언트 CPU 사용으로 인한 부담은 성능에 큰 영향을 미치지 않는다. 다만 C2PL에서는 매 액세스마다 서버와 통신이 필요하므로 트랜잭션 철회는 메시지 수에 큰 영

향을 미치는데, 이 기법은 기본적으로 낮은 철회율을 보이므로 메시지 수는 크게 증가하지 않는다.

클라이언트 수의 증가는 캐쉬히트율에 서로 상반되는 두 효과를 가져온다. 첫째, NumClients가 증가하면, 데이타 충돌이 심화되므로 철회율이 높아진다. 이는 트랜잭션 재실행시의 높은 캐쉬히트로 인하여 전체 캐쉬히트율에 긍정적인 영향을 미친다. 반면, NumClients의 증가는 클라이언트 버퍼를 무효화시키는 속도를 가속시킨다. 이는 결과적으로 유효캐쉬크기(effective cache size)의 감소로 이어져 히트율에 부정적으로 작용한다. 여기서 유효캐쉬크기란 용어는 유효한 데이타 페이지를 캐싱하고 있는 버퍼 슬롯의 수를 의미한다. 그림 4(b)에서 보는 바와 같이 15-클라이언트 지점까지 O2PL-I의 캐쉬히트율은 긍정적인 효과로 인하여 약간 증가하나, 15-클라이언트 이상에서는 부정적인 효과가 더 크게 작용하므로 감소하기 시작한다. 반면, DL과 AOCC에서는 높은 철회율로 인하여 긍정적인 효과가 부정적인 효과를 항상 압도한다. DL-ST/1은 그림자 트랜잭션을 위한 공간부담 때문에 작은 NumClients에서 다른 기법에 비해 낮은 캐쉬히트율을 보인다. 그러나 NumClients의 증가에 따라, 트랜잭션이 철회되어 그림자 트랜잭션으로 대치될 확률이 증가한다. 그림자 트랜잭션은 원 트랜잭션이 철회된 지점까지 부분적으로 재실행되는데, 이때는 높은 캐쉬히트율을 보일 수 있다. 따라서 DL-ST/1은 NumClients의 증가에 따라 캐쉬히트율도 꾸준히 증가한다.

O2PL-I는 비최신의 데이타를 클라이언트 버퍼에 캐싱하지 않기 때문에 유효캐쉬크기에 있어서 다른 기법에 비해 우수한 성능을 발휘할 것 같지만 실제는 그렇지 않다. 비동기적으로 클라이언트 버퍼를 무효화시키는 DL과 AOCC는 오히려 O2PL-I보다 큰 유효캐쉬크기를 보인다. 이는 O2PL-I는 예 1에서 설명한 것과 같이 불필요한 무효화를 야기하기 때문이다. 참고로 25-클라이언트에서 AOCC, DL, DL-ST/1, O2PL-I, C2PL은 각각 197, 193, 185, 180, 147 페이지의 유효캐쉬크기를 보인다. 각 기법이 자신의 버퍼 용량 240~250 페이지를 최신의 데이타를 캐싱하는데 전부 사용하지 못하는 이유는 새로이 캐싱되는 페이지보다 무효화되는 페이지의 수가 더 많기 때문이다. C2PL이 유난히 작은 유효캐쉬크기를 보이는 이유는 이 기법은 클라이언트 버퍼의 무효화를 위한 어떤 조치도 취하지 않고 있기 때문에 상당량의 버퍼 슬롯이 무효화된 데이타를 캐싱하느라 낭비되고 있기 때문이다.

5.1.2 사용자 대화식 환경에서의 성능

본 실험은 사용자 대화식(user interactive) 응용에서 각 기법의 성능 비교를 위하여 대화식 트랜잭션을 제기하면서 실시하였다. 대화식 트랜잭션은 다수의 읽기 연산을 진행하고, 사용자가 약간의 생각을 한 후에 쓰기 연산을 하는 형태로 모델링되었다. 따라서 사용자의 작업시간은 쓰기 연산마다 필요하다고 가정한다. 일반적으로 대화식 트랜잭션을 위한 성능평가 실험은 사용자 작업시간을 1-10초로 변화하면서 실시한다[10, 15]. 본 실험도 사용자 작업시간에 변화를 주면서 실시하였는데, 사용자 작업시간을 변경시키더라도 각 기법의 상대적인 성능결과 및 변화추이에 있어서는 큰 차이가 없었다. 그러므로 본 논문은 매 쓰기 연산마다 3초의 사용자 작업시간을 요구하는 경우의 결과만을 제시한다.

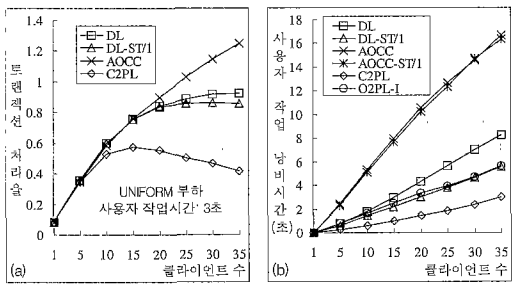

그림 5 (a) 트랜잭션 처리율 (b) 작업 낭비시간

트랜잭션 처리율과 트랜잭션 완료당 낭비되는 사용자 작업시간이 그림 5에 있다. 트랜잭션 철회에 대한 변화추이 및 상대적인 결과는 5.1.1절의 그림 4(b)와 유사하다. 대화식 트랜잭션이 제기되는 시스템에서 사용자는 트랜잭션이 실제 실행되는 시간에 비해서 상당히 긴 시간을 소모한다. 그러므로 자원의 사용율은 매우 낮게 마련인데, 자원을 가장 효율적으로 사용하는 AOCC도 CPU 사용율 및 네트워크 사용율이 15%를 초과하지 않는다. 이와 같이 자원의 사용율이 낮을 경우에는 각 TCCM 기법의 성능은 데이타 충돌을 해결하는 정책에 따라 결정되기 마련인데, 가급적 철회에 의존하여 데이타 충돌을 해결하는 기법이 자원을 효율적으로 사용하므로 우수한 성능을 발휘한다. 따라서 AOCC는 다른 기법에 비해 매우 우수한 성능을 발휘한다. 이 실험에서 O2PL-I는 DL과 거의 유사한 성능을 보이며 AOCC-ST/1은 AOCC와 비슷한 성능을 보인다.

만일 트랜잭션이 처음 실행에서 성공적으로 완료하면, 트랜잭션은 평균 4개의 페이지를 갱신하므로 사용자는 12초의 작업시간이 필요할 것이다. 그림 5(b)에서 25-클

라이언트 지점의 결과를 살펴보면 AOCC는 12초 정도의 낭비시간을 보인다. 이는 사용자는 평균적으로 자신의 작업을 두번 해야 함을 의미한다. 반면 DL-ST/1과 O2PL-I에서는 1.3배 정도의 작업을 해야 한다. 이와 같이 AOCC는 다른 기법에 비해 매우 높은 낭비시간을 보이는데, 이는 AOCC가 높은 철회율을 보인다는 점과 밀접한 관련이 있다. 또한 다른 기법에 비해서 트랜잭션이 많이 진행된 상태에서 철회된다는 점도 여기에 일조를 한다. AOCC-ST/1은 AOCC와 비교하여 사용자 작업시간에 있어서 별다른 이점을 주지 못하지만, DL-ST/1은 사용자 작업시간의 낭비를 O2PL-I 수준까지 낮출 수 있다. 사용자 대화식 환경에서는 트랜잭션이 완료하기까지 사용자 작업시간의 양이 시스템 성능만큼 중요하다. 그러므로 이 환경에서는 사용자 작업시간에 있어서 적은 낭비시간을 보이며 성능면에서도 비교적 우수한 DL-ST/1과 O2PL-I 기법을 채택하는 것이 바람직할 것이다.

5.2 실험 2- HIGHCON 부하

이 부하는 모든 트랜잭션이 HIGHCON이라는 데이타베이스의 특정 부분을 매우 높은 확률로 액세스하는 환경을 모델링한다. 데이타베이스에는 250 페이지의 HIGHCON 영역이 있고, 각 트랜잭션이 실행하는 연산의 80%는 이 영역을 액세스한다고 가정한다. 그러므로 이 부하는 심한 데이타 충돌과 높은 참조 국부성의 특징을 지닌다. 그림 6(a)에서 보는 것과 같이 DL, DL-ST/1, 그리고 AOCC는 다른 기법에 비해 우수한 성능을 발휘한다. 이는 이 기법들은 상대적으로 낮은 트랜잭션 대기율을 보이며 트랜잭션 완료당 필요한 메시지 부담이 적기 때문이다. 참고로 DL과 DL-ST/1은 15-클라이언트에서 각각 24%, 28%의 트랜잭션 대기율을 보이며, C2PL과 O2PL-I는 각각 53%, 57%의 대기율을 보인다. O2PL은 전역 교착상태를 발견할 때까지의 긴 시간지연 때문에 C2PL보다 높은 대기율을 보인다.

DL-ST/1은 낮은 NumClients에서 그림자 트랜잭션으로 인한 부담 때문에 DL과 AOCC에 비해서 낮은 성능을 보인다. 그러나 NumClients가 증가하면 그림자 트랜잭션의 효과가 증대되어, 10-클라이언트에 도달하면 이들보다 우수한 성능을 발휘하기 시작한다. AOCC는 로크로 인한 트랜잭션 대기가 없고 DL과 비슷한 수준의 철회율을 보인다. 그럼에도 불구하고 이들은 비슷한 성능을 보인다. 이는 그림 6(b)의 트랜잭션 철회당 낭비되는 시간과 밀접한 관련이 있다. 이 낭비시간은 트랜잭션이 제기되어 철회될 때까지의 평균시간을 의미한다. AOCC는 DL기반 기법보다 더 긴 낭비시간을 보인다. 이는 AOCC에서는 트랜잭션이 많이 진행된 후에 철회되며, 이로 인하여 자원의 낭비가 심하다는 것을 의미한다. 참고로 AOCC는 15-클라이언트에서 53% 정도의 네트워크 낭비율을 보인다. 네트워크 낭비율이란 전체 네트워크 사용율 대비 철회되는 트랜잭션에 의해 사용되는 비율을 의미한다. DL은 네트워크 낭비율이 40% 정도이며, DL-ST/1은 그 낭비율이 25% 정도에 지나지 않는다. 이 실험에서 대부분의 데이타가 HIGHCON 영역에서 액세스되므로 각 기법은 UNIFORM보다 이 부하에서 높은 캐쉬히트율을 보인다. 이 부하에서는 Num Clients가 증가하면 철회율이 급격히 증가하는데, 트랜잭션 재실행시의 캐쉬히트가 UNIFORM의 경우만큼 그렇게 높게 예상되지는 않는다. 그 이유는 전 실행에서 클라이언트 버퍼로 가져다 놓은 데이타들이 곧 다시 무효화될 가능성이 높기 때문이다. 따라서 NumClients의 증가에 따라 메시지 부담은 UNIFORM의 경우보다 훨씬 급격히 증가한다. NumClients에 따른 메시지 수의 변화추이 및 요구되는 메시지 수의 상대적인 순서는 UNIFORM의 경우와 거의 동일하다.

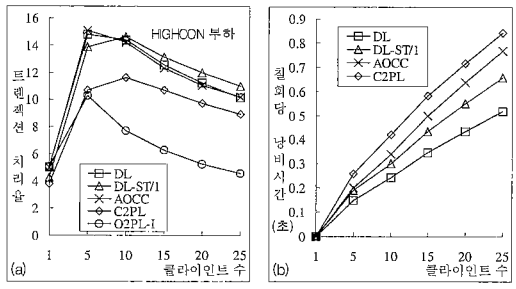

그림 6 (a) 트랜잭션 처리율 (b) 철회당 낭비시간

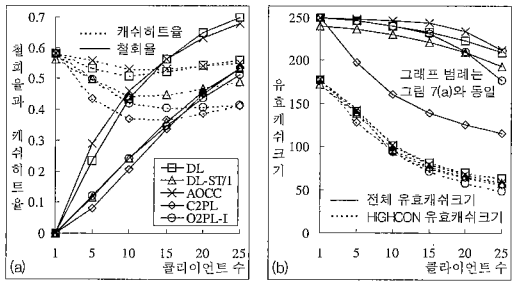

그림 7 (a) 철회율과 캐쉬히트율 (b) 유효캐쉬크기

그림 7(a)에서 주목할 점은 철회율에 있어서 DL과 AOCC기법간에 역전이 발생한다는 것이다. 이 현상은 그림 4(b)의 철회율 변화추이에서 어느 정도 예상된다.

그림 4(b)에서 데이타 충돌이 심화됨에 따라 두 기법간의 철회율 차이는 점점 커지는데, 충돌이 어느 이상으로 극심해지면 오히려 그 차이는 줄어들기 시작하여 마침내 HIGHCON과 같은 극심한 데이타 충돌 환경에서는 역전 현상이 발생하는 것이다. 낮은 데이타 충돌 환경에서 로크 인증을 위해 대기하고 있는 트랜잭션은 추후에 로크를 인증받아 실행을 계속하게 될 가능성이 높기 때문에 로크는 철회율을 줄이는데 큰 공헌을 한다. 그러나, 데이타 충돌이 극심해짐에 따라 트랜잭션 대기는 대부분 교착상태로 이어져 철회에 대한 로크의 효과가 급격히 줄어든다. 또한 AOCC에서는 트랜잭션이 많이 진행된 후에 철회되는 경향이 있는데, 이도 재실행되는 트랜잭션의 철회율 관점에서는 긍정적으로 작용한다. 트랜잭션이 많이 진행된 후에 철회되어 처음부터 다시 실행되면, 그 트랜잭션의 실행은 캐쉬히트율이 높기 때문에 매우 빠르게 진행된다. AOCC에서 어떤 트랜잭션이 빠르게 실행되면, 그 트랜잭션의 실행동안에는 상대적으로 적은 수의 페이지가 무효화되므로 그의 완료성공 가능성은 매우 높아진다. 그러나, DL에서는 트랜잭션이 빠르게 실행되더라도 결국에는 자신이 액세스한 데이타에 대하여 로크 승인을 받아야 한다. 이 과정에서 충돌되는 로크로 인하여 트랜잭션이 대기할 수도 있고 교착상태로 인하여 다시 철회될 수도 있다. 그러므로 트랜잭션 재실행시는 AOCC가 DL보다 낮은 트랜잭션 철회율을 보인다. 따라서 이 부하에서는 AOCC가 DL보다 낮은 철회율을 보일 수도 있는 것이다.

그림 7(b)는 전체 유효캐쉬크기 및 HIGHCON 유효캐쉬크기를 나타낸다. 여기서 HIGHCON 유효캐쉬크기란 용어는 전체 버퍼에서 HIGHCON 영역의 최신 데이타 페이지를 캐싱하고 있는 버퍼 슬롯의 수를 의미한다. NumClients의 증가에 따라 철회율이 급격히 증가하므로 각 기법의 캐쉬히트율도 증가할 것 같지만, 그림 7(a)에서 10-클라이언트까지의 결과는 이와 상반된다. 이는 그림 7(b)에서 보는 것과 같이 NumClients의 증가에 따라 HIGHCON 유효캐쉬크기가 크게 감소하기 때문이다. HIGHCON 영역의 데이타는 캐싱될 가능성이 매우 높으나, 그들은 또한 무효화될 가능성도 높다. 버퍼 무효화가 전혀 발생하지 않는 1-클라이언트에서 각 기법은 전체 버퍼 슬롯의 70% 이상을 HIGHCON 데이타를 캐싱하는데 사용한다. 그러나 10-클라이언트 지점에 도달하면 그 사용율은 40% 정도로 급격히 감소한다. 비록 각 기법이 중복사본을 관리하기 위하여 다른 기술을 사용하지만 HIGHCON 데이타의 무효화율이 매우 높기 때문에 HIGHCON 캐쉬크기에 있어서는 큰 차이를 가져오지 않는다. O2PL-I는 불필요한 무효화를 야기하므로 C2PL보다도 작은 HIGHCON 캐쉬크기를 보인다. 전체 유효캐쉬크기는 그다지 심각하게 감소하지는 않는다. 이는 HIGHCON 데이타의 높은 무효화율로 인하여 빈 버퍼 슬롯이 존재할 가능성이 높으므로 non-HIGHCON 데이타는 일단 캐싱되면 오랫동안 버퍼에 간직될 수 있기 때문이다.

5.3 실험 3- HOTCOLD 부하

이 부하에서 각 클라이언트는 데이타베이스에 자신만이 자주 액세스하는 40 페이지의 HOT 영역을 가지고 있다고 가정한다. 한 클라이언트의 HOT 영역은 다른 클라이언트의 HOT 영역과는 완전히 분리되며, 각 트랜잭션이 실행하는 연산의 80%는 자신의 HOT 영역을 액세스한다고 가정한다. 따라서, 이 부하는 높은 참조국부성과 낮은 데이타 충돌의 특징을 보인다. 이 실험은 버퍼 크기가 작더라도 높은 캐쉬히트율을 보이므로 ClientBufSize를 DbSize의 10%로 설정한다. 클라이언트 수에 따른 트랜잭션 처리율 및 철회율이 그림 8에 있다. 이 실험에서 C2PL은 다른 기법에 비해서 높은 메시지 부담을 보이므로 상대적으로 낮은 성능을 보인다. 이 부하에서 각 클라이언트는 자신만의 HOT 영역을 높은 확률로 액세스하므로 각 클라이언트에 캐싱된 데이타가 서로 겹칠 가능성이 매우 낮다. 이는 O2PL-I에서 완료부담의 감소로 이어지고, DL, DL-ST/1, 그리고 AOCC에서는 무효화된 데이타를 읽을 가능성의 감소로 나타난다. 그리고 이 기법들은 중복사본을 통신부담없이 액세스할 수 있기 때문에 트랜잭션 실행단계의 부담이 거의 비슷할 것이다. 그러므로 이들은 비슷한 성능을 보인다.

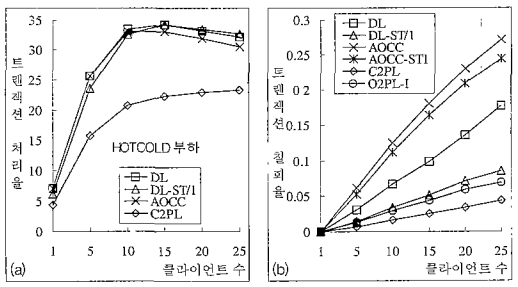

그림 8 (a) 트랜잭션 처리율 (b) 철회율

C2PL은 15-클라이언트에서 서버 CPU의 사용율이 80% 정도에 이른다. 그러나 다른 기법들은 25-클라이언트에 도달하더라도 그 사용율이 50%를 초과하지 않

는다. 이 차이는 대부분 클라이언트와 서버간에 교환되는 메시지 수의 차이에서 비롯된다. 참고로 이 실험에서 C2PL은 트랜잭션 완료당 필요한 메시지 수가 대략 50개이며, DL기반 기법과 AOCC는 약 10개 정도이다. O2PL-I는 클라이언트 수의 증가에 따라 완료단계의 메시지 수가 약간 증가하여 25-클라이언트에서 17개의 메시지 부담을 보인다. C2PL은 메시지를 처리하기 위한 CPU 실행시간 때문에 다른 기법에 비해 네트워크 사용율이 낮은 편인데, 15-클라이언트에서 약 80%의 사용율을 보인다. 반면, 다른 기법들은 NumClients가 10에 도달하더라도 그 사용율이 90% 이상이 된다. C2PL 이외의 기법들은 메시지 교환이 매우 적음에도 불구하고 그들의 성능은 네트워크 사용집중으로 인한 병목현상이 발생한다. 이를 이해하기 위해서는 단위시간에 전달되는 메시지의 양을 조사해야 한다. 이 실험에서 한 트랜잭션의 성공적인 종료를 위해 평균 8개의 데이타 페이지가 네트워크를 통해 전송된다. 여기서 4개의 페이지는 캐쉬 미스로 인한 데이타전송이고(평균 80%의 캐쉬히트율을 보임.) 나머지 4 페이지는 트랜잭션 완료시 클라이언트에서 갱신된 내용을 서버로 전송하기 위함이다. 이 실험에서 DL기반 기법과 AOCC는 C2PL에 비해서 대략 1.5배의 트랜잭션 처리율을 보인다. 이는 C2PL에서 한 트랜잭션이 제기되어 완료하는 시간동안에 이 기법들은 C2PL보다 4개의 페이지를 더 전송함을 의미한다. 이 메시지의 양을 네트워크 대역폭 관점에서 살펴보면 64개의 제어 메시지에 해당된다. 따라서 DL기반 기법과 AOCC에서 네트워크의 사용집중으로 인한 병목현상이 더 심한 것이다. C2PL에서 교환되는 메시지 수는 매우 크지만 높은 캐쉬히트로 인하여 대부분의 메시지는 제어 메시지일 것이라는 사실을 유념해야 한다. 이 실험에서 모든 기법들은 거의 비슷한, 약 80%의 캐쉬히트율을 보인다.

HOTCOLD 부하에서 DL기반 기법과 AOCC는 네트워크 대역폭의 한계로 인하여 성능이 제한된다. 이 대역폭의 한계를 완화하기 위하여 네트워크 속도를 50Mbps까지 증가하면서 성능추이를 살펴보았다. 네트워크 속도가 증가되면 CPU와 디스크 사용율의 증가로 인한 병목현상이 발생할 것이다. 이 점을 감안하여 디스크 속도는 기본 속도의 3배로, 클라이언트와 서버 CPU 속도는 기본값의 2배로 설정하고 실험을 실시하였다. NumClients는 시스템의 향상된 자원을 사용할 충분한 트랜잭션을 제공하기 위하여 25로 설정하였다. 이 실험에서 DL기반 기법과 AOCC의 성능은 네트워크 속도가 30Mbps가 될 때까지 거의 선형적으로 증가한다. 그러나 C2PL은 많은 수의 메시지 교환을 필요로 하므로 네트워크 속도가 20Mbps에 도달하면 서버 CPU의 사용율이 90% 정도에 이르고, 네트워크 속도가 30Mbps에 도달하면 CPU의 과다사용으로 인해 성능이 거의 증가하지 않는다. DL기반 기법과 AOCC의 성능은 네트워크 속도가 40Mbps가 되면 디스크의 과다사용으로 인한 성능 병목현상이 발생하기 시작한다. AOCC가 향상된 시스템 자원을 효율적으로 사용할 수 있지만 DL기반 기법보다 더 높은 철회율을 보인다. 이 부하에서 각 트랜잭션은 처음 실행시에도 높은 캐쉬히트율을 보이므로 트랜잭션 재실행시의 높은 캐쉬히트의 효과가 전체 캐쉬히트율에 미치는 영향이 그다지 크지 않다. 그리고 DL기반 기법에서 로크 충돌로 인한 트랜잭션 대기의 대부분은 추후에 로크가 인증되어 다시 실행이 가능할 것이다. 이와 같은 이유로 인해서 네트워크 속도가 50Mbps로 증가할 때까지 DL기반 기법은 AOCC보다 미약하기는 하지만 항상 우수한 성능을 보인다.

6. 결 론

본 논문에서는 주사본 로킹 기법에 기초한 검사기반의 TCCM 기법인 DL과 DL-ST 기법을 제안하였다. 일반적으로 주사본 로킹에 기초한 기법은 통신부담이 높아 낮은 성능을 보이는 경향이 있기 때문에, DL 기법의 성능을 최적화하는데 있어서 본 논문의 주요 목표는 중복사본의 일관성 검사를 위하여 필요한 메시지 부담의 경감화에 집중되었다. 이 목표는 다수의 로크 요청 및 데이타전송 요청을 하나의 단일 메시지로 구성하여, 액세스한 중복사본에 대한 로크 설정 및 유효성 검사를 비동기적으로 실행함에 의해 실현되었다. 이와 같은 비동기적 검사기반 기법은 높은 트랜잭션 철회율을 야기할 수 있는데, 본 논문에서는 이 단점을 보완하기 위하여 그림자 트랜잭션의 개념을 도입하여 이 개념과 DL의 로킹 기법을 DL-ST 기법에 통합하였다.

본 논문에서는 모의실험을 통하여 DL, DL-ST, AOCC, 그리고 C2PL 기법의 성능을 비교하였다. 검사기반 기법은 클라이언트 중복사본의 유효성 검사를 동기적 또는 비동기적으로 실행하는가에 따라 메시지 교환의 수와 트랜잭션 철회율 사이에 성능 절충 현상을 보인다. 비동기적 기법인 DL과 AOCC 기법은 트랜잭션 완료당 필요한 메시지 부담이 매우 낮기 때문에 동기적 기법인 C2PL에 비해 다양한 부하환경에서 매우 우수한 성능을 발휘한다. 그러나 이 기법들은 지연된 유효성 검증으로 인하여 트랜잭션 철회율이 높은 단점이 있다. 특히 AOCC는 데이타 충돌의 해결을 전적으로 철회에 의

존하기 때문에 철회율이 매우 높아 자원의 낭비가 심한 경향이 있다. 비동기적 검사기반 기법에 그림자 트랜잭션의 개념을 적용시키면 철회율을 크게 감소시킬 수 있는데, DL-ST 기법은 UNIFORM과 HOTCOLD 부하에서 DL 기법의 철회율을 50% 이상 감소시킨다. DL-ST는 회피기반 기법중 가장 성능이 우수한 것으로 알려진 O2PL-I보다도 우수한 성능을 발휘하면서 비슷한 수준의 철회율을 보인다.

제안된 기법들을 비교하면, DL-ST 기법은 낮은 데이타 충돌 환경에서 그림자 트랜잭션과 연관된 부담 때문에 DL보다 낮은 성능을 보인다. 그러나 DL-ST는 데이타 충돌이 심화되어 트랜잭션 철회가 빈번한 환경에서는 그림자 트랜잭션으로 인한 효과가 증대되어 DL보다 우수한 성능을 보인다. 그리고 DL-ST는 사용자 대화식 환경에서 트랜잭션 철회로 인한 사용자 작업시간의 낭비를 크게 줄일 수 장점이 있다. AOCC-ST에서 그림자 트랜잭션의 결과는 추후에 다른 트랜잭션에 의해 아무런 제약없이 무효화될 수 있기 때문에 그림자 트랜잭션의 효과가 트랜잭션 처리율 및 철회율에 큰 영향을 미치지 않는다. 본 연구의 미래 연구 과제로서 비동기적으로 클라이언트 버퍼를 무효화시키는 회피기반 기법에 관한 연구가 진행될 것이다. 그리고 데이타전송과 질의전송(query-shipping)이 조합된 시스템을 위한 TCCM 기법에 대한 연구가 진행될 것이다.

참 고 문 헌

[1] O. Deux et al, "The O2 System," Comm. ACM, Vol. 34, No. 10, pp. 34-48, 1991.

[2] W. Kim, J. F. Garza, N. Ballou, and D. Woelk, "The Architecture of the ORION Next-Generation Database System," IEEE Trans. on Knowledge and Data Eng., Vol. 2, No. 1, pp. 109-124, 1990.

[3] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb, "The ObjectStore Database System," Comm. ACM, Vol. 34, No. 10, pp. 50-63, 1991.

[4] A. Adya, R. Gruber, B. Liskov, and U. Maheshwari, "Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks," ACM SIGMOD Int. Conf. on Management of Data, pp. 23-34, 1995.

[5] M. J. Carey, M. J. Franklin, M. Linvy, and E. J. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures," ACM SIGMOD Int. Conf. on Management of Data, pp. 357-366, 1991.

[6] M. J. Franklin and M. J. Carey, "Client-Server Caching Revisited," Technical Report #1089, Computer Sciences Dept., Univ. of Wisconsin-Madison, 1992.

[7] M. J. Franklin, M. J. Carey, and M. Livny, "Transactional Client-Server Cache Consistency: Alternatives and Performance," ACM Trans. on Database Syst., Vol. 22, No. 3, pp. 315-363, 1997.

[8] R. Gruber, "Optimism vs. Locking: A Study of Concurrency Control for Client-Server Object-Oriented Databases," PhD Thesis, MIT, 1997.

[9] M. T. Özsu, K. Voruganti, and R. C. Unrau, "An Asynchronous Avoidance-Based Cache Consistency Algorithm for Client Caching DBMSs," Int. Conf. On VLDB, pp. 440-451, 1998.

[10] Y. Wang and L. A. Rowe, "Cache Consistency and Concurrency Control in A Client/Server DBMS Architecture," ACM SIGMOD Int. Conf. on Management of Data, pp. 367-376, 1991.

[11] K. Wilkinson and M. Neimat, "Maintaining Consistency of Client Cached Data," Int. Conf. On VLDB, pp. 122-133, 1990.

[12] P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.

[13] D. J. DeWitt, P. Futtersack, D. Maier, and F. Velez, "A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems," Int. Conf. On VLDB, pp. 107-121, 1990.

[14] M. J. Carey and M. Livny. "Conflict Detection Tradeoffs for Replicated Data," ACM Trans. on Database Syst., Vol. 16, No. 4, pp. 703-746, 1991.

[15] R. Agrawal, M. J. Carey, and M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications," ACM Trans. on Database Syst., Vol. 12, No. 4, pp. 609-654, 1987.

[16] P. S. Yu, H. Heiss, and D. M. Dias, "Modeling and Analysis of a Time-Stamp History Based Certification Protocol for Concurrency Control," IEEE Trans. on Knowledge and Data Eng., Vol. 3, No. 4, pp. 525-537, 1991.

[17] S. H. Cho, K. Y. Bae, and C. S. Hwang, "Write Notification for Certification Protocol Based on Time-Stamp History," Int. Workshop on Database and Expert Syst. App., pp. 919-924, 1998.

[18] H. Schwetman, CSIM Users' Guide for Use with CSIM Revision 16, Microelectronics and Computer Technology Corp., 1992.

[19] T. E. Anderson, B. N. Bershad, E. D. Lazowska, and H. M. Levy, "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism," ACM Trans. on Computer Syst., Vol. 10, No. 1, pp. 53-79, 1992.

[20] A. M. Law and W. D. Kelton, Simulation Modeling & Analysis, McGraw-Hill, 1991.

권 혁 민

1984년 서울대학교 제어계측공학과(학사). 1994년 한국과*학기술원 정보및통신공학과(석사). 1998년 한국과학기술원 정보및통신공학과(박사). 1984년 ~ 1991년 대우전자 중앙연구소 컴퓨터개발부 선임연구원. 1999년 ~ 현재 세명대학교 소프트웨어학과 전임강사. 관심분야는 트랜잭션 처리, 분산 데이타베이스, 멀티미디어 데이타베이스