트랜잭션 캐쉬 일관성을 유지하기 위한 지연 로킹 기법의 성능 평가

권 혁 민†

요 약

데이타전송(data-shipping) 모델에 근간을 둔 클라이언트-서버(client-server) DBMS는 트랜잭션간 캐싱(inter-transaction caching)을 허용함에 의해 클라이언트의 자원을 효율적으로 이용할 수 있다. 그러나 트랜잭션간 캐싱을 허용하면 각 클라이언트는 데이타베이스의 일부분을 동적으로 캐싱할 수 있기 때문에 트랜잭션 캐쉬 일관성 유지(transactional cache consistency maintenance : TCCM) 기법의 필요성을 야기한다. 지연 로킹(deferred locking : DL) 기법은 주사본 로킹 기법에 근간을 둔 새로운 검사기반의 TCCM 기법이다. DL에서는 클라이언트에 캐싱된 데이타의 유효성 검사를 위한 메시지 부담을 줄이기 위하여 다수의 로크 요청과 데이타전송 요청을 단일 메시지로 구성하였다. 모의실험을 통하여, DL 기법과 검사기반 기법중 가장 대표적인 적응적 낙관적 동시성 제어 기법과 캐싱 두단계 로킹 기법과의 성능을 비교하였다. DL 기법은 다른 검사기반 기법과 비교하여 적당한 수준의 트랜잭션 철회율을 보이며 성능도 우수하다.

Performance Evaluation of Deferred Locking for Maintaining Transactional Cache Consistency

Hyeok-Min Kwon†

ABSTRACT

Client-server DBMS based on a data-shipping model can exploit client resources effectively by allowing inter-transaction caching. However, inter-transaction caching raises the need of transactional cache consistency maintenance(TCCM) protocol, since each client is able to cache a portion of the database dynamically. Deferred locking(DL) is a new detection-based TCCM scheme designed on the basis of a primary-copy locking algorithm. In DL, a number of lock requests and a data shipping request are combined into a single message packet to minimize the communication overhead required for consistency checking. Using a simulation model, the performance of the proposed scheme is compared with those of two representative detection-based schemes, the adaptive optimistic concurrency control and the caching two-phase locking. The performance results indicate that DL improves the overall system throughput with a reasonable transaction abort ratio over other detection-based schemes.

1. 서 론

고성능 워크스테이션의 등장과 네트워크 기술의 발전으로 인하여 클라이언트-서버(client-server) DBMS는 분산 계산환경에서 중요한 요소가 되었다. 클라이언트-서버 DBMS는 정보교환 단위에 따라 데이타전송(data-shipping) 방식과 질의전송(query-shipping) 방식으로 분류한다[3, 7, 8]. 질의전송 방식에서 서버는 질의를 처리하고 그 결과만을 클라이언트로 전송하는데, 대부분의 상용 관계형 DBMS들은 이 방식을 채택하고 있다[8, 16]. 반면, 데이타전송 방식에서 서버는 클라이언트가 질의를 처리할 수 있도록 필요한 데이타를 클

† 정 회 원 : 세명대학교 소프트웨어학과 교수
논문접수 : 2000년 2월 28일, 심사완료 : 2000년 8월 4일

라이언트로 전송하는데, 대부분의 분산 객체지향(object-oriented) DBMS들은[5, 10, 11] 이 방식을 사용하고 있는 실정이다. 이 방식은 대부분의 DBMS 기능들을 클라이언트로 이관하기 때문에 비교적 풍부하고 값싼 클라이언트의 자원들을 효율적으로 이용한다. 따라서 공유되는 서버의 의존성을 줄일 수 있는 이점이 있다. 그러나 클라이언트가 다량의 데이타전송을 요청하면, 데이타전송 시스템은 네트워크의 사용집중에 따른 병목현상으로 인하여 심각한 성능 장애를 야기할 수 있다.

데이타전송 시스템에서, 트랜잭션의 완료후에도 클라이언트에 캐쉬된 데이타를 계속 유지 관리하여 다른 트랜잭션들이 이를 사용하도록 허용하는 트랜잭션간 캐쉬(inter-transaction caching)은 네트워크의 사용집중을 해결하기 위한 효과적인 방법이다. 트랜잭션간 캐쉬은 메모리와 CPU와 같은 클라이언트 자원들이 효율적으로 이용되도록 해주며, 데이타의 전송부담 및 클라이언트와 서버에서 메시지를 처리하기 위하여 필요한 CPU의 부담을 상당히 완화시켜 준다. 그러나 트랜잭션간 캐쉬을 허용하면 각 클라이언트는 데이타베이스의 일부분을 동적으로 캐쉬하게 되므로 공유되는 데이타의 중복사본(replica)이 여러 클라이언트에 존재하게 된다. 그러므로 트랜잭션간 캐쉬은 클라이언트 트랜잭션들이 일관성 있는 데이타베이스를 액세스한다는 것을 보장하기 위하여, 기본적으로 동시성 제어(concurrency control)와 중복사본 관리(replica management)의 양면성을 지닌 트랜잭션 캐쉬 일관성 유지(transactional cache consistency maintenance : TCCM) 기법의 필요성을 야기한다. 이 논문에서 클라이언트에 캐쉬된 데이타를 중복사본으로 취급한다. 중복사본들의 일관성을 유지하기 위해서 클라이언트와 서버사이에 정보교환이 필요하므로, TCCM 기법은 구현하기가 복잡하고 실행시키는데도 부담이 뒤따른다. TCCM 기법의 잠재적인 이점과 이에 연관된 실행 부담사이의 절충(trade-off)은 부하(workload)의 특징에 따라서 다르게 나타나기 때문에, DBMS의 성능은 채택된 TCCM 기법과 사용환경에 따라 상당한 차이를 보인다.

따라서 다수의 TCCM 기법들이 제안되고 그들의 성능이 비교되었다[1, 3, 79, 14, 17, 18]. 이들은 유효하지 않은 데이타를 액세스한 어떤 트랜잭션도 완료할 수 없다는 것을 보장하는데, 이를 보장하는 방법의 차이에 따라 검사기반(detection-based) 기법과 회피기반(avoidance-based) 기법으로 분류된다[8]. 회피기반 기법은 비최신의 데이타가 클라이언트 버퍼에 캐쉬되는 것을 방지하여 유효하지 않은 데이타를 액세스하는 것을 불가능하게 한다. 회피기반 기법은 중복사본 관리를 위하여 보통 읽기 연산에 대해서는 자신의 중복사본만을 로크하고 쓰기 연산은 모든 중복사본들을 로크하는 ROWA(read-one and write-all) 방식을[2] 사용한다. ROWA 방식에서 읽기 연산은 매우 효율적으로 실행된다. 그러나 갱신 트랜잭션이 완료(commit)를 시도하거나 또는 쓰기 연산을 실행하려면, 각 클라이언트에 캐쉬된 중복사본들의 일관성을 유지시키기 위하여 상당한 통신부담을 야기한다. 그리고 이 일관성을 유지시키기 위하여 필요한 실행에는 그 트랜잭션을 제기한 클라이언트와 서버, 그리고 트랜잭션이 갱신하려는 데이타를 캐쉬하고 있는 모든 클라이언트들이 관여한다. 만일 이들중 어느 하나라도 부분적인 네트워크 단절이나 고장으로 인하여 동작이 불가능하면, 그 쓰기 연산은 실행될 수 없기 때문에 쓰기 연산의 가용성(availability)을 상당히 감소시킬 수 있다.

검사기반 기법은 비최신의 데이타를 일시적으로 클라이언트 버퍼에 캐쉬할 수 있다. 그러므로 각 트랜잭션은 최소한 완료하기 전까지는 자신이 액세스한 모든 데이타에 대한 유효성을 보장받아야 한다. 검사기반의 대표적인 기법으로 AOCC(adaptive optimistic concurrency control)와[1, 9] C2PL(caching two-phase locking) 기법이[3, 7] 있다. AOCC 기법은 어떤 제한도 가하지 않고 중복사본의 액세스를 허용하므로 유효성을 검사하기 위한 메시지 부담을 상당히 줄일 수 있다. 그러나 트랜잭션이 비최신의 데이타를 액세스할 가능성이 있고, 액세스한 데이타의 유효성 보장이 완료단계에 도달해서야 이루어지므로 트랜잭션의 철회(abort)가 빈번할 수 있다. 반면, C2PL 기법은 중복사본을 액세스하기 전에 서버에서 유효성을 검사받고 서버의 로크표에 적절한 로크를 설정한다. 그러므로 C2PL은 낮은 트랜잭션 철회율을 보이나, 중복사본의 유효성을 검사받기 위하여 서버와 왕복 메시지를 주고 받아야 하므로 많은 통신부담을 야기한다.

검사기반 기법에서 트랜잭션의 일관성을 유지시키기 위한 실행에는 그 트랜잭션을 제기한 클라이언트와 서버만이 관여한다. 그러므로 검사기반 기법은 회피기반 기법에 비해서 구현이 용이하며 고장에 매우 견고하다. 후자의 장점은 비교적 고장에 견고하지 못한 클라

이언트를 고려할 때 매우 바람직한 특징이라 할 수 있다. 또한, 점점 보편화되고 있는 이동 응용(mobile application)에서 이동 클라이언트의 간헐적인 통신장애를 고려한다면 이 장점은 점점 부각될 것이다. 그렇지만 검사기반 기법은 중복사본의 유효성 검사 시기 및 방법에 따라서 높은 통신부담으로 인하여 낮은 성능을 보이거나, 또는 지연된 유효성 검사로 인하여 높은 트랜잭션 철회율을 보이는 단점이 있다[1, 3, 8, 9]. 이와 같은 단점에도 불구하고 검사기반 기법이 지니는 잠재적인 장점의 중요성을 인식한다면 이에 대한 더욱 심도있는 연구가 필요하다.

본 논문에서 제안된 기법은 중복사본의 관리 차원에서는 통신부담을 줄이기 위하여 비동기적으로 유효성을 검사하며, 동시성 제어 측면에서는 트랜잭션의 철회율을 줄이기 위하여 로킹 기법을 채택한다. 이 기법은 주사본 로킹 기법에 근간을 두고 있지만, 클라이언트에 캐쉬된 중복사본의 유효성 검사 및 로크 설정의 요청은 캐쉬미스(cache miss)로 인하여 클라이언트에서 서버로 메시지를 보낼 필요가 있을 때까지 지연된다. 그러므로 이 기법을 지연 로킹(deferred locking : DL)이라 명명한다. DL은 클라이언트가 서버와 통신하지 않고 중복사본을 액세스한다는 점에서 AOCC와 비슷하나 동시성 제어에 있어서 로킹 기법을 사용한다는 점이 다르다. DL과 C2PL은 모두 주사본 로킹 기법에 근간을 두고 있지만, DL은 로크 설정을 위한 통신부담이 매 액세스마다 필요한 것이 아니라 매 캐쉬미스마다 필요하다는 점에서 C2PL과 다르다. 이 논문의 구성은 다음과 같다. 2절에서 기존에 제안된 기법들을 살펴보고, 3절에서는 본 논문에서 제안한 기법을 설명한다. 그리고 4절에서는 제안된 기법의 성능을 파악하기 위한 모의실험 모델을 설계하고, 5절에서는 실험 결과를 제시하고 분석한다. 마지막으로, 6절에서 본 논문의 결론을 맺는다.

2. 관련 연구

이 절에서는 본 논문의 성능평가에서 사용된 기법들을 중심으로 기존의 기법들을 살펴본다. 클라이언트-서버 시스템에서 정보교환의 단위가 객체(object), 또는 페이지인가에 따라, 서버는 객체 서버 또는 페이지 서버로 구성할 수 있다. 페이지 서버는 객체 서버에 비해 구현이 간단하고 보통 성능이 우수하므로[6, 7], 페이지 서버 관점에서 각 기법을 설명한다. 즉 이 논문에서 동시성 제어의 단위와 데이타전송의 단위가 데이타 페이지라고 가정한다. 이 가정은 모의실험에서도 그대로 적용된다.

2.1 검사기반 기법

검사기반 기법에서 클라이언트는 비최신의 데이타를 일시적으로 캐쉬할 수 있기 때문에 최소한 트랜잭션이 완료하기 전까지는 액세스한 모든 데이타의 유효성 여부를 검사받아야 한다. 검사기반 기법은 중복사본의 유효성을 검사하는 시점에 따라 동기적 그리고 비동기적 기법으로 분류할 수 있다. 동기적 기법의 알고리즘으로 C2PL이[3, 7] 있는데, 이 기법에서 트랜잭션은 데이타를 액세스하기 전에 서버의 중앙 로크표(central lock table)에 적절한 로크를 설정한다. 그러므로 클라이언트가 자신의 중복사본을 읽기 위해서는 그 데이타에 대한 읽기 로크(Read Lock)를 서버로 요청하는데, 이때 그 중복사본의 LSN(log sequence number)을 같이 전송한다. 이 메시지를 받으면, 서버는 읽기 로크를 설정한 후, 해당 페이지에 대한 클라이언트의 LSN과 자신의 LSN을 비교하여 클라이언트 중복사본의 유효성을 검사한다. 만일 중복사본이 유효하지 않으면 서버는 로크응답 메시지에 그 데이타의 최신 버전을 첨부하여 전송한다. C2PL은 중복사본의 유효성을 검사하고 적절한 로크를 설정하기 위하여 매 액세스마다 서버와 왕복 메시지를 주고 받아야 하므로 심각한 통신부담을 야기한다.

비동기적으로 중복사본의 유효성을 검사하는 기법으로 AOCC가[1, 9] 있는데, 이 기법은 트랜잭션의 실행단계에서는 어떤 제한도 가하지 않고 데이타의 액세스를 허용한다. 대신 트랜잭션의 완료단계에서 액세스한 데이타의 유효성을 보장하는데, 이를 위해 각 트랜잭션이 읽고 갱신한 데이타에 대한 정보를 각각 읽기집합(read-set), 쓰기집합(write set)에 유지 관리한다. 각 클라이언트는 트랜잭션의 완료단계에서 이 두 집합과 자신의 버퍼에서 갱신된 새로운 데이타를 완료 요청과 함께 서버로 전송된다. 서버는 그 트랜잭션이 읽은 데이타중에서 무효화된 데이타가 있는지를 조사하여 트랜잭션의 완료 또는 철회를 결정한다. 이 기법은 데이타 충돌의 해결을 전적으로 트랜잭션 철회에 의존하므로 철회율이 높은 단점이 있다. 그러나 이 기법은 캐쉬된 데이타의 유효성을 검사하기 위한 별도의 통신부

담을 야기하지 않기 때문에 메시지 부담이 상당히 적은 것으로 알려져 있다.

2.2 회피기반 기법

회피기반 기법은 한 트랜잭션의 갱신결과를 모든 클라이언트 버퍼에 원자적으로 반영하므로 비최신의 데이타를 캐싱하는 것이 불가능하다. 회피기반의 알고리즘으로 O2PL(optimistic two-phase locking)과[3, 4, 7, 8] CBL(callback locking)[3, 7, 8, 17] 기법이 있는데, 이들의 차이점은 트랜잭션의 갱신결과를 언제 클라이언트 버퍼에 파급시키는가에 있다. O2PL 기법은 트랜잭션의 완료단계에서 갱신결과를 파급시키는 반면, CBL은 실제 갱신 연산이 제기된 순간에 파급시킨다.

O2PL은 ROWA 방식을 채택한 분산 낙관적 로킹(optimistic locking)[4] 기법에 기초를 두고 있다. 클라이언트는 트랜잭션의 실행단계에서 읽기와 쓰기 로크를 자신의 로크표에 설정하고 중복사본을 이용하여 연산을 진행한다. 각 트랜잭션은 자신의 지역버퍼에서 갱신 연산을 실행하고, 이 갱신결과는 완료단계에서 완료요청 메시지와 함께 서버로 전송된다. 이 메시지를 받으면 서버는 중복사본의 일관성을 유지시키기 위해 변경된 페이지를 캐싱하고 있는 모든 클라이언트와 협조하여 갱신결과를 각 클라이언트 버퍼에 원자적으로 반영한다. 갱신결과를 반영하는 방법의 차이에 따라 [3, 7, 8]은 O2PL-I, O2PL-P, 그리고 O2PL-D 등 세 가지의 O2PL 기법을 제시하였다. O2PL-I는 변경된 데이타를 버퍼에서 제거하여 무효화시키는 반면, O2PL-P는 변경된 데이타를 각 클라이언트로 전파한다. O2PL-D는 무효화(invalidation)와 전파(propagation) 정책을 동적으로 선택한다. O2PL은 각 클라이언트에 설정된 로크들 사이의 충돌(conflict)을 트랜잭션의 완료단계에서 검사한다. 그리고 갱신 트랜잭션이 완료를 시도할 때마다 캐쉬 일관성을 유지시키기 위한 실행에 여러 클라이언트들이 관여한다. 이 요인이 성능에 얼마나 큰 영향을 미치는가를 파악하기 위하여 다음 예를 살펴보자.

예 1(O2PL에서 완료시의 부담으로 인한 성능 제한) : 클라이언트 C_1에서 데이타 x를 갱신한 트랜잭션 T_1이 완료단계에 도달했다고 가정하자. 그리고 C_2와 C_3는 x를 캐싱하고 있고, C_3에서 실행중인 T_3는 x에 대하여 이미 읽기 연산을 수행했다고 가정하자. 서버는 T_1의 완료요청 메시지를 받으면, x를 캐싱하고 있는 C_2와 C_3로 완료준비 메시지를 전송한다. C_2는 x에 로크를 설정하지 않았기 때문에 즉시 승인 메시지를 보낸다. T_1은 C_3에서 승인 메시지를 받아야 완료할 수 있는데, C_3는 T_3가 x의 읽기 로크를 해제하기 전에는 승인 메시지를 전송할 수 없다. 만일 T_3가 방금 전에 실행을 시작한 트랜잭션이라면 이 지연은 매우 길어질 수 있다. 불행히도 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 선정되어 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. O2PL-P는 갱신결과를 다수의 클라이언트에 원자적으로 반영하기 위하여 반드시 두단계 완료 규약을 채택해야 한다.■

CBL 기법도 O2PL과 마찬가지로 한 트랜잭션의 갱신결과를 모든 클라이언트 버퍼에 원자적으로 반영한다. 그러나 CBL은 갱신결과의 반영을 갱신 연산이 제기된 시점에 시도한다는 측면에서 O2PL과는 다르다. CBL에서 클라이언트 C_1의 트랜잭션 T_1이 데이타 x를 갱신하려면 연산을 실행하기 전에 그 데이타에 대한 전역적인 갱신 권한을 소유해야 한다. 이를 위해서 C_1은 서버를 통하여 x를 캐싱하고 있는 모든 클라이언트로 로크회수(callback) 메시지를 전송한다. 이 메시지를 받은 클라이언트는 자신의 버퍼에서 x를 제거하고 회수승인 메시지를 서버로 전송하는데, 만일 트랜잭션이 x를 사용하고 있으면 그 트랜잭션이 완료 또는 철회될 때까지는 회수승인 메시지를 보낼 수 없다. 모든 회수요청 메시지가 승인되면, C_1은 x에 대한 갱신 권한을 소유하게 된다. CBL은 갱신 연산에 대한 권한을 트랜잭션 실행중에 전역적으로 승인받기 때문에 완료단계에서는 오직 그 트랜잭션을 제기한 클라이언트와 서버만이 관여한다. 그러므로 CBL 기법은 완료 연산을 실행하는데 필요한 부담은 적은 반면, 트랜잭션의 실행단계에서 전역적인 갱신 권한을 획득할 때 예1과 비슷한 문제점이 발생된다. 회피기반 기법은 클라이언트의 캐쉬 일관성을 항상 유지시켜 읽기 연산은 매우 효율

적으로 실행된다. 그러나 최소한 트랜잭션이 완료하기 전에는 그 트랜잭션의 갱신결과를 다수의 클라이언트 버퍼에 원자적으로 반영해야 하므로 쓰기 연산을 실행하기 위한 부담은 상당히 큰 편이다.

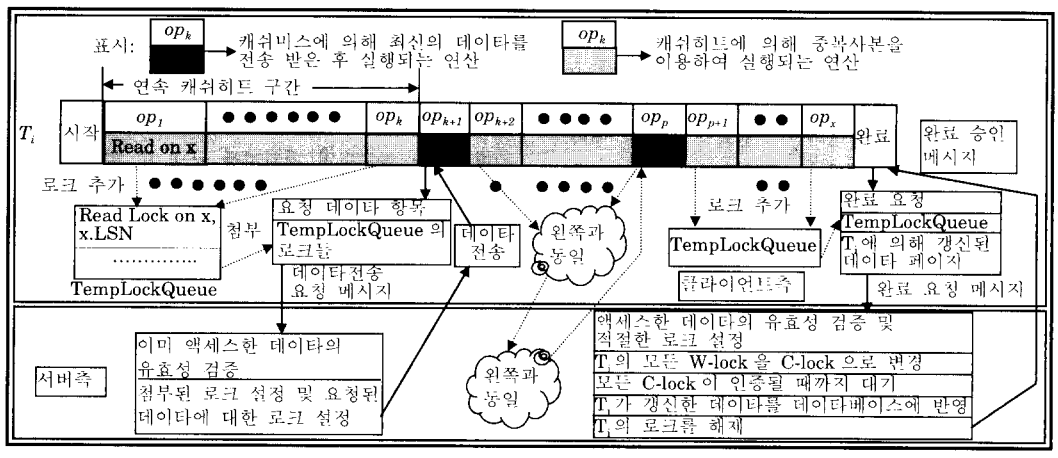
3. 지연 로킹 기법

본 논문에서 데이타베이스의 각 페이지는 그 상태를 파악하기 위하여 로그 일련 번호(log sequence number : LSN)라는 버전 번호를 가지고 있다고 가정한다. DL 기법은 C2PL과 마찬가지로 LSN을 이용하여 중복사본의 유효성을 검사한다. 그리고 교착상태 검출시에 희생자를 선정하기 위하여 각 트랜잭션은 시작시에 부여받은 타임스탬프를 가지고 있다고 가정한다. 이를 위해 각 시스템은 네트워크 타임 프로토콜(network time protocol)[13]로 구현된 느슨히 동기화된 클럭을 가지고 있고 이 클럭으로부터 타임스탬프를 부여받는다고 가정한다.

3.1 기본 개념

DL 기법은 주사본 로킹 기법에 기초를 두고 있지만, 로크의 설정이 반드시 데이타를 액세스하기 전에 이루어지는 것은 아니다. (그림 1)과 같이, 클라이언트는 자신의 중복사본을 이용하여 연산을 실행하고, 그에 대한 로크 설정 및 유효성 검사는 추후에 클라이언트와 서버사이에 정보교환이 필요한 시점에서 요청되어 비동기적으로 실행된다. DL에서 클라이언트는 중복사본을 이용하여 읽기 연산을 실행할 때마다 TempLockQueue에 해당 데이타에 대한 읽기 로크(read lock : R-lock) 요청과 그 데이타의 LSN을 추가한다. 그리고 쓰기 연산을 실행하면 적절한 쓰기 로크(write lock : W-lock)를 TempLockQueue에 추가하고 자신의 지역버퍼에 새로운 값을 저장한다. 캐쉬미스가 발생하면, 클라이언트는 TempLockQueue의 내용을 데이타전송 요청 메시지에 첨부하여 서버로 전송하고 TempLockQueue를 소거한다. 서버는 첨부된 읽기 로크들을 설정하기 전에 해당 페이지에 대한 클라이언트의 LSN과 자신의 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.

클라이언트 트랜잭션이 완료단계에 도달하면, 자신의 지역버퍼에서 갱신한 새로운 데이타와 TempLockQueue의 내용을 완료요청 메시지에 첨부하여 서버로 전송한다. 완료요청 메시지에 로크가 첨부되어 있으면 서버는 데이타전송 요청을 처리할 때와 동일한 방법으로 데이타의 유효성을 검사하고 적절한 로크를 설정한다. 그리고 나서 서버는 완료를 시도하는 트랜잭션의 모든 W-lock을 완료 로크(commit lock : C-lock)로 변경한다. 이 트랜잭션은 자신의 C-lock이 전부 인증될 때까지 대기하여야 한다. 모든 C-lock이 인증되면, 서버는 그 트랜잭션이 갱신한 데이타를 데이타베이스에 반영하고 로크를 해제한 다음 클라이언트로 완료승인



(그림 1) DL 기법의 실행 시나리오

메시지를 전송한다.

3.2 DL의 로킹 기법

DL 기법에서 서버는 클라이언트에서 요청되는 로크를 설정하기 위하여 중앙 로크표(central lock table)와 교착상태를 검출하기 위하여 대기그래프(wait for-graph)를 유지 관리한다. 비호환 로크를 스케쥴링할 때는 로크 인증의 순서를 고려하여 대기그래프에 적절한 선분을 추가하고 교착상태를 검출한다. 교착상태가 발생하면, 여기에 관여된 가장 늦게 시작된 트랜잭션을 철회하여 해결한다. 공정성을 위해, 요청된 로크들은 특별한 언급이 없는 한 선입선출(FIFO) 원리에 의해 인증된다 ; 요청된 로크들은 뒤에서 설명되는 규칙 1 이외에는 먼저 스케쥴링된 다른 로크와 호환성(compatibility)이 성립해야 인증된다. 이를 위해 DL은 보통 로크 요청의 순서대로 로크표를 유지 관리한다. DL은 3.1절에서 기술한 것처럼 세가지의 로크 유형을 사용하는데, 이들 사이의 호환성 표(compatibility matrix)와 기본적인 알고리즘이 <표 1>과 (그림 2)에 있다. 본 논문에서는 로크를 인증(granted) 모드로 설정한 트랜잭션 뿐만 아니라, 대기(waiting) 모드로 설정한 트랜잭션도 해당 데이타의 로크 소유자로 표현한다.

〈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 ;
AR : 요청자 철회 ; NB : 요청자 비대기 상태 ;
B : 요청자 대기 ; 규칙 1 : 본문 내용 참고.

```
/* 트랜잭션 Ti의 데이타 x에 대한 로크 요청 처리 */
if requested lock type == Rlock-I then {
   if no conflicting lock exists then grant the lock;
   else wait for the lock; /* set the lock in waiting mode */}
else if requested lock type == Rlock-E then {
   if client's LSN on x != server's LSN on x then Abort(Ti);
   else if ((any conflicting C-lock exists) || (any conflicting
        W-lock, which is set by elder transaction than Ti, exists))
              then Abort(Ti);
   else grant the lock; /* set the lock in granted mode */}
else  { /* W-lock request */
   set the lock in waiting mode; }
```

(그림 2) DL 기법의 로킹 알고리즘

클라이언트는 데이타전송 요청 메시지를 통하여 TempLockQueue에 있던 R-lock과 현재 요청되는 데이타에 대한 R-lock의 설정을 서버에게 요청한다. 이 두 유형의 R-lock은 전자는 이미 연산을 실행했고, 후자는 아직 연산을 실행하지 않았다는 점에서 다르기 때문에 서로 구분되어야 한다. 본 논문에서 전자의 R-lock을 Rlock-E(Rlock Explicit)라 명명하고, 후자를 Rlock-I(Rlock-Implicit)라 명명한다. 이 R-lock들은 일단 서버의 로크표에 설정되고 나면, 동일한 읽기 로크로 취급된다. Rlock-I는 기존의 읽기 로크와 성격이 동일하므로 일반의 2PL 기법과 동일하게 스케쥴링한다. 즉, 비호환 로크가 존재하면 해당 트랜잭션은 로크가 인증될 때까지 대기한다. Rlock-E는 그와 연관된 읽기 연산이 이미 실행되었기 때문에 비호환 로크로 인하여 인증될 수 없다면 해당 트랜잭션은 대기하지 않고 철회된다. 만일 비호환 로크의 소유자가 성공적으로 완료한다면, Rlock-E를 요청한 트랜잭션이 액세스한 데이타는 갱신되어 더 이상 유효하지 않을 것임에 유의해야 한다. Rlock-E를 스케쥴링할 때, 만일 이와 충돌하는 W-lock이 존재하면 다음 규칙에 의하여 로크 인증을 요청의 순서와는 다르게 할 수 있다.

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

규칙 1은 DL의 로크 승인에 있어 FIFO 원칙을 위배하는 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. 충돌하는 W-lock에 앞서 Rlock-E가 먼저 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 읽기 로크로 인하여 C-lock의 인증이 지연될 수는 있다. 충돌하는 W-lock에 대하여 Rlock-E는 교착상태가 발생하지 않으면 항상 인증 가능한데, 규칙 1만을 적용한 이유는 C-lock 인증의 대기와 깊은 관련이 있다. 만일 충돌하는 C-lock이 존재하면 Rlock-E를 요청한 트랜잭션은

철회된다. 이는 C-lock을 설정한 트랜잭션은 이미 완료단계에 도달했다는 점을 고려하기 위하여 그 트랜잭션에게 우선권을 주기 위함이다.

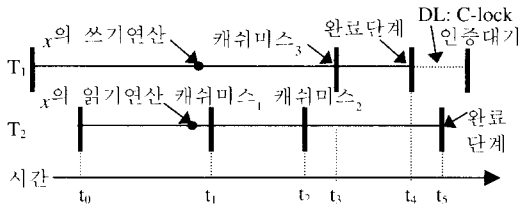
서버는 W-lock을 항상 대기 모드로 로크표에 설정하는데, 이는 나중에 요청된 Rlock-E가 W-lock에 앞서 인증될 수도 있기 때문이다. 물론, W-lock을 스케쥴링할 때, 비호환 로크가 존재하면 대기 그래프에 적절한 선분을 추가하고 교착상태를 검사한다. FIFO 원칙 때문에, W-lock을 대기 모드로 설정했다고 해서 나중에 요청되는 비호환 로크들이 W-lock에 앞서 인증되는 것은 아니다(규칙 1의 경우는 제외). W-lock을 대기 모드로 설정하지만 해당 트랜잭션은 대기하지 않고 계속 실행한다. 이 대기는 그 트랜잭션이 완료단계에 도달할 때까지 지연된다. 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 대기하여야 한다.

본 논문에서 C-lock을 도입한 이유는 Rlock-E를 스케쥴링할 때 W-lock과 구분하기 위함이다. 그리고 이미 W-lock을 설정할 당시에 그 W-lock과 연관된 직렬화 순서가 결정되기 때문에 C-lock으로 변경시에는 단순히 W-lock을 C-lock으로 변경하기만 하면 된다. O2PL에서 전역적인 갱신 권한을 완료단계에서 획득하는 것처럼, DL 기법도 트랜잭션의 실행단계에서 W-lock을 전혀 설정하지 않고 완료단계에서 C-lock만을 설정함에 의해 스케쥴링할 수 있다. 그러나 이 경우에 C-lock에 앞서 많은 R-lock들이 인증될 수 있기 때문에 완료단계에서 트랜잭션의 대기시간이 매우 길어질 수 있다. DL 기법에서 W-lock을 트랜잭션의 실행단계에서 설정하는 주된 이유는 이를 방지하기 위함이며, 또한 공정성을 위해서 로크 인증을 그 로크와 연관된 연산이 클라이언트에서 실행된 순서와 비슷하게 스케쥴링하기 위함이다. W-lock의 또 다른 이점은 로크 충돌로 인하여 야기되는 교착상태를 일찍 발견하는 것을 가능하게 해 준다는 점이다.

3.3 DL 기법과 AOCC 기법의 비교

DL 기법은 트랜잭션이 액세스하려는 데이타가 캐싱되어 있으면 서버와 통신하지 않고 실행이 가능하며, 트랜잭션의 완료단계를 위한 연산에는 단지 그 트랜잭션을 제기한 클라이언트와 서버만이 관여한다는 점에서 AOCC와 비슷한 측면이 있다. 그러나 DL 기법은 캐쉬미스가 발생하면 액세스한 데이타에 대하여 적절한 로크를 설정함에 의해, 추후에 다른 트랜잭션들이 그의 실행 결과를 무효화시키는 것을 방지한다는 점에서 AOCC와는 다르다.

예 2(DL과 AOCC 기법의 비교) : 다음과 같이 클라이언트 C_1, C_2에서 각각 T_1, T_2가 실행된다고 가정하자.



(그림 3) 트랜잭션의 실행 스케쥴

위 그림에서 두 기법 모두 캐쉬미스1을 위한 실행을 성공적으로 마치고 나면 그때까지는 최신의 정보만을 액세스했다는 것이 보장된다. DL 기법은 서버에서 데이타전송 요청을 처리할 때 T_2가 t_0~t_1 구간에서 액세스한 데이타의 유효성을 검사하고 적절한 로크를 설정함에 의해 이를 보장한다. 반면, AOCC는 캐쉬미스1의 데이타전송 요청의 응답 메시지에 다른 트랜잭션들에 의해 갱신되어 무효화된 중복사본에 대한 정보를 함께 전송한다. AOCC는 이 정보를 바탕으로 T_2가 무효화된 데이타를 액세스했는지 여부를 파악함에 의해 검증을 실시한다. 캐쉬미스2에서 데이타전송 요청 및 응답을 처리하는 실행을 살펴보면, DL은 T_2가 t_1에서 t_2까지 액세스한 데이타에 대한 유효성을 검사하며 적절한 로크를 설정한다. 반면, AOCC는 t_1에서 t_2까지 액세스한 데이타뿐만 아니라, 캐쉬미스1에서 검증된 데이타에 대해서도 다시 유효성 검사를 실시해야 한다 ; 만일 t_1에서 t_2사이에 다른 트랜잭션 T_3가 x를 갱신하고 성공적으로 완료한다면, T_2가 읽은 x는 더 이상 유효하지 않음에 유의해야 한다. 그러나 이와 같은 실행 스케쥴이 DL에서 발생하면, T_2가 t_1에서 x에 대하여 읽기 로크를 설정해 놓았기 때문에 T_3의 x에 대한 C-lock의 인증은 T_2가 로크를 해제할 때까지 지연될 것이다.

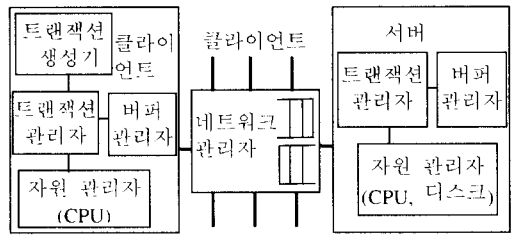
T_1의 캐쉬미스3을 위한 실행을 살펴보면, DL은 x에

대한 W-lock을 서버의 로크표에 대기 모드로 설정한다. 그러나 T_1은 대기상태에 빠지지 않고 계속 실행한다. T_1은 완료단계에서 x에 대한 W-lock을 C-lock으로 변경하는데, 이 C-lock의 인증은 (그림 3)과 같이 T_2가 읽기 로크를 해제할 때까지 지연된다. T_2가 t_5에서 x의 읽기 로크를 해제하고 완료하면, T_1도 x의 C-lock이 인증되어 완료 가능하다. 그러나 이를 AOCC에서 살펴보면, T_1은 t_4에서 성공적으로 완료한다. T_2가 완료단계에 도달하면 서버에서 최종적으로 자신이 액세스한 데이타의 유효성을 검증받는다. 그런데 T_2가 액세스한 데이타 x가 T_1에 의해 갱신되었기 때문에 T_2는 철회된다. 이상과 같이 두 기법은 캐쉬된 데이타를 사용한 후 검증을 받는다는 측면에서 비슷하다. 그러나 DL에서는 일단 검증된 데이타의 유효성이 로크로 인하여 완료단계까지 보장되는데 비해, AOCC에서는 캐쉬미스가 발생할 때 유효성이 검증된 데이타도 추후에 다른 트랜잭션에 의해 아무런 제한이 없이 갱신될 수 있기 때문에 그렇지 않다. 따라서 AOCC 기법은 DL에 비해 높은 트랜잭션 철회율을 보일 것이다.■

DL과 AOCC 기법은 검사기반 기법이므로 클라이언트가 유효하지 않은 데이타를 캐쉬할 수 있다. 이는 높은 트랜잭션 철회율을 야기할 가능성이 있기 때문에, 이 기법들은 중복사본의 유효성을 효율적으로 관리하기 위하여 전파 방법과 무효화 방법을 조합하여 사용한다. 만일 트랜잭션이 비최신의 데이타를 액세스했기 때문에 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 클라이언트의 중복사본에만 반영된다. 따라서 그 갱신결과와 관련이 있는 다른 클라이언트의 중복사본은 더 이상 유효하지 않다. 이와 같은 사항을 알리기 위하여 서버는 갱신된 데이타를 캐쉬하고 있는 클라이언트로 해당 데이타의 무효화 메시지를 전송한다. 이 메시지를 받은 클라이언트는 즉시 해당 데이타를 버퍼에서 제거하여 나중에 트랜잭션들이 그 데이타를 액세스하는 것을 방지한다. 무효화 메시지는 서버에서 클라이언트로 전송되는 다른 메시지에 첨부되어 전달되므로 메시지의 전송 횟수에는 영향을 미치지 않는다. 무효화 메시지를 전송하기 위해 서버는 각 데이타마다 이를 캐쉬하고 있는 클라이언트의 정보를 갱신하기 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.

4. 모의실험 모델

이 절에서는 DL 기법의 성능을 평가하기 위한 모의실험(simulation) 모델을 제시한다. 비교 대상으로 검사기반 기법중 C2PL과 AOCC를 선정하였다. C2PL은 DL과 같이 주사본 로킹 기법을 채택하고 있기 때문에 선정되었고, AOOC는 검사기반 기법중 성능이 가장 우수한 것으로 알려져 있기 때문에[1, 9] 선정되었다. 그리고 회피기반 기법중 우수한 성능을 발휘하는 것으로 알려진 O2PL-I의 성능도 같이 비교한다. 모의실험 모델은 [3, 8]에서 연구된 폐쇄 큐잉(closed queuing) 모델을 참고로 하고 있고, MCC에서 개발한 CSIM[15] 언어를 이용하여 구현하였다.



(그림 4) 모의실험 모델

모의실험 모델은 크게 네트워크 관리자, 클라이언트 모듈, 그리고 서버 모듈로 구성된다. 네트워크 관리자는 서버와 클라이언트의 정보교환을 관리하며 FIFO 큐 형태로 구현되었다. 트랜잭션 생성기는 트랜잭션을 생성하여 시스템에 제기하는데, 한 트랜잭션이 성공적으로 종료하면, 즉시 다음 트랜잭션을 생성한다. 이 논문에서 클라이언트만이 트랜잭션을 제기한다고 가정한다. 클라이언트의 트랜잭션 관리자는 트랜잭션의 실행을 관리하며 각 TCCM 기법에 따라 동시성 제어를 책임진다. 서버의 트랜잭션 관리자는 클라이언트에서 요청되는 요구에 적절한 응답을 하며, 적용된 각 TCCM 기법에 알맞은 동시성 제어 및 중복사본을 관리하기 위한 실행을 책임진다. 클라이언트와 서버에 있는 자원 관리자는 디스크와 CPU와 같은 자원의 할당을 관리하는데, 자원의 할당은 FIFO 원리에 의하여 처리된

다. 본 논문에서 서버와 클라이언트는 각각 하나의 CPU를 소유하고 있는데 그들의 성능은 서로 다르며, 클라이언트에는 디스크가 없다고 가정한다. 버퍼 관리자는 LRU 정책을 바탕으로 자신의 버퍼를 관리한다.

〈표 2〉 트랜잭션 및 시스템 자원 변수

입력변수	의 미	설 정
DbSize	데이타베이스 크기	1000 페이지
PageSize	페이지 크기	4096 바이트
MeanTranSize	평균 페이지액세스 수	20 페이지
UpdateProbability	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	페이지수마다 추가되는 명령수	10,000
LockInst	로크 설정/해제를 위한 명령수	300
LSNManInst	LSN을 비교하기 위한 명령수	300
PageValInst	페이지 검증에 필요한 명령수	10~300
RegisterInst	캐쉬 사이트 조사/캐쉬 사이트 추가 및 삭제를 위한 명령수	300
DiskOvhdInst	디스크 I/O를 위한 명령수	5000
MinDiskAcc	최소 디스크 액세스 시간	10 millisec.
MaxDiskAcc	최대 디스크 액세스 시간	30 millisec.

트랜잭션 및 시스템 자원, 그리고 실행부담에 관한 입력변수들은 각각 <표 2>, <표 3>에 있다. 본 실험에서 데이타베이스는 DbSize의 페이지로 구성된다. MeanTranSize는 한 트랜잭션에 의해 액세스되는 평균 데이타 페이지 수를 정의하는데, 트랜잭션들이 액세스하는 페이지 수는 MeanTranSize의 ±20% 편차 구간에서 균등 분포(uniform distribution)를 이룬다. 트랜잭션들은 평균적으로 20개의 서로 다른 페이지를 읽는데, 별도의 언급이 없으면 이 중에서 평균 20%의 페이지가 갱신된다. 모의실험에서 트랜잭션의 재실행 정책에는 기본적으로 진짜 재실행(real restart : RR)과 위조 재실행(fake restart : FR)의 두 방법이 있다. RR 정책에서 트랜잭션이 철회되면 그 트랜잭션이 처음부터 다시 실행된다. 반면, FR 정책에서는 원래의 트랜잭션이 다른 트랜잭션으로 대치되어 실행된다. RR 정책에서 트랜잭션의 재실행은 이미 이전의 실행에서 자신이 필요로 하는 데이타의 상당량을 버퍼에 가져다 놓았을 것이므로 매우 빠르게 실행될 수 있다. 이는 RR 정책은 높은 철회율을 보이는 기법에 상대적인 이점을 준다는 것을 의미한다. 그리고 만일 트랜잭션이 철회되면 사용자는 원래의 트랜잭션을 다시 실행하기보다는 다른 트랜잭션의 실행을 원할 수도 있다. 이와 같은 점을 고려하여 본 논문은 FakeRestartPct로 명기된 위조 재실행 확률을 정의한다 ; 트랜잭션이 철회되면, 그 트랜잭션은 FakeRestartPct의 확률로 다른 트랜잭션으로 대치되어 재실행된다. 트랜잭션이 철회되어 재실행되면 새로운 타임스탬프를 부여받는다고 가정한다.

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 CPU 부담은 각 메시지마다 반드시 필요한 고정된 명령수(FixedMsgInst)와 메시지에 포함된 각 페이지마다 필요한 PageMsgInst로 모델링된다. 이 명령수를 실행하는 부담은 메시지를 전송하는 사이트와 회신하는 사이트에서 모두 필요하다. 로크 및 데이타전송 요청과 같은 제어 메시지의 크기는 CtrlMsgSize로 정의되는데, DL과 AOCC에서는 제어 메시지에 다른 정보들이 첨부되므로 이 크기를 두 배로 설정한다.

디스크의 수는 4로 설정하며, 디스크 액세스 시간은 MinDiskAcc에서 MaxDiskAcc까지 균등 분포를 이룬다. 트랜잭션이 성공적으로 완료하면, 서버는 그 트랜잭션이 갱신한 페이지를 자신의 버퍼에 반영하고 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 데이타를 로킹하기 위해 LockInst의 CPU 처리부담을 필요로 한다. DL은 클라이언트가 TempLockQueue에 로크를 추가할 때마다 LockInst의 실행부담을 부과한다. 그리고 DL에서 특정 데이타에 대한 클라이언트의 LSN과 서버의 LSN을 비교하기 위한 부담과 트랜잭션의 완료로 특정 데이타의 LSN을 갱신하기 위한 부담을 LSNManInst으로 정의한다. AOCC는 트랜잭션의 읽기 집

합에 속한 각 데이타가 무효화 정보에 포함되어 있는지를 조사하는 검증(validation) 방법을 채택하고 있다. 각 데이타의 검증을 위해서, 무효화 페이지가 적으면 페이지당 10 명령수, 무효화 페이지가 30개 이상이면 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.

본 논문에서 사용된 주요 성능지수는 초당 처리되는 트랜잭션 처리율과 트랜잭션 응답 시간인데, 폐쇄 큐잉 모델에서 두 성능중 하나의 결과를 알면 다른 값은 계산을 통하여 파악할 수 있기 때문에 트랜잭션 처리율만을 제시한다. 트랜잭션 철회율과 트랜잭션이 완료할 때까지 필요한 메시지 수 및 버퍼의 히트율과 같은 보조 성능지수들이 성능 결과를 분석하기 위하여 조사되었다. 트랜잭션 완료당 필요한 메시지 수에는 철회로 인하여 낭비되는 메시지도 포함된다. 실험은 복사 방법(replication approach)을[12] 사용하여 실행되었는데, 실험 시작시의 초기 편향(initial bias)을 제거하기 위하여 초기 800개의 트랜잭션 완료의 결과는 무시하였다. 본 논문에서 제시된 결과값은 6개의 다른 임의수를 사용하여 실시된 모의실험 결과의 평균값으로, 각 복사(replication)는 5000개의 트랜잭션이 완료할 때까지 실시되었다. 이렇게 산출된 결과는 95%의 신뢰수준을 만족한다.

5. 실험 결과 및 분석

본 모의실험은 다양한 데이타 공유형태를 반영하기 위하여 UNIFORM, HIGHCON, 그리고 HOTCOLD라고 명명된[3, 7, 8] 세 부하(workload)에서 실시되었는데 각 부하의 특징은 해당 절에서 설명한다. 본 논문에서는 원래의 C2PL과 O2PL-I의 약점을 보완하고 실험을 실시하였다. O2PL-I는 다음의 원인으로 인해 성능이 저하되는 경향이 있다. 첫째, 이 기법은 트랜잭션의 완료 단계에서 많은 메시지 부담을 야기한다. 이 부담을 완화하기 위하여 본 논문에서는 완료준비 메시지를 브로드캐스팅(broadcasting) 방식으로 보낸다. 이 메시지를 보낼 때 필요한 부담은 하나의 제어 메시지를 보내는 것과 동일하게 설정하였다. 둘째, O2PL-I는 전역 교착상태를 주기적으로 검사하는데, [3, 7]에서 이 주기를 1초로 설정하였다. 그러나 이는 어떤 부하에서는 적합하지 않다. 본 논문에서는 본격적인 실험에 앞서 각 부하에서 전역 교착상태의 주기를 변경하면서 성능을 미리 파악해 보았다. 그리고 각 부하에서 보편적으로 가장 우수한 트랜잭션 처리율을 보이는 전역 교착상태 주기를 선택하여 O2PL-I에 적용하고 이를 O2PL-Ix라 명명한다 ; O2PL-Ix 기법은 x초마다 전역 교착상태를 검사한다. C2PL은 클라이언트 버퍼의 상당량을 무효화된 페이지를 캐싱하느라 낭비한다. 이를 개선하기 위하여 C2PL-I(C2PL-Invalidation)라 명명된 기법은 DL과 동일한 방법으로 무효화 정보를 클라이언트로 전파해 무효화된 중복사본을 버퍼에서 일찍 제거한다. 이 논문에서 C2PL과 O2PL-I의 성능이 통계적으로 의미가 없거나, 그들의 변형에 비해 매우 낮으면 결과를 생략한다.

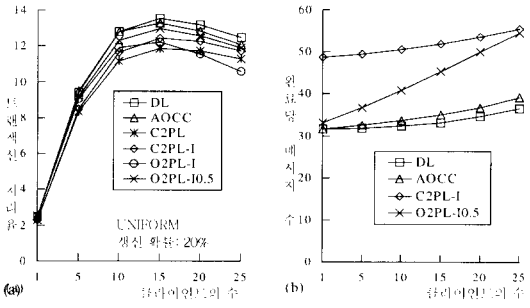
5.1 실험 1-UNIFORM 부하

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.

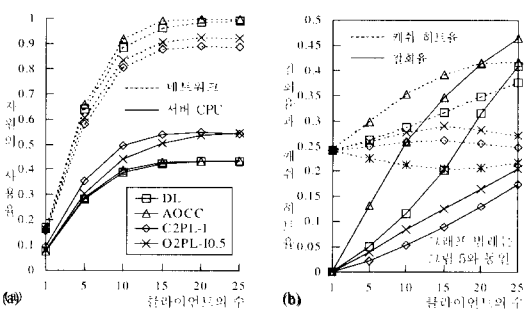
5.1.1 클라이언트 수에 따른 성능

트랜잭션 처리율과 트랜잭션이 완료할 때까지 필요한 평균 메시지 수가 (그림 5)에 있다. 이 실험에서 DL과 AOCC는 다른 기법에 비하여 우수한 성능을 발휘한다. (그림 5(a))를 살펴보면, NumClients가 증가하면 각 기법의 성능도 증가한다. 그러나 NumClients가 15 이상이 되면 성능은 저하되기 시작하는데, 이는 기법에 따라 약간 다른 원인에서 기인된다. C2PL은 15-클라이언트에서 29%의 트랜잭션 대기율을 보인다. 트랜잭션 대기율이란 시스템에 제기되어 있는 전체 트랜잭션 대비 로크 인증을 기다리며 대기 상태에 빠진 트랜잭션 수의 비율을 의미한다. 반면, 20-클라이언트에서는 41%의 대기율을 보인다. 트랜잭션의 수가 5만큼 증가되었지만, 실제 활성 트랜잭션의 수는 단지 1.15(20 x 0.59 - 15 x 0.71)만큼만 증가된다. 이는 15-클라이언트 이상에서 NumClients의 증가는 동시성을 높이는데 기여하기보다는 데이타 충돌만을 가중시켜 이로 인한 쓰래싱(thrashing) 현상을 야기한다는 것을 의미한다. 이런 관찰로부터 C2PL의 성능은 데이타 사용경쟁으로 인하여 제한된다고 할 수 있다. 이런 주장은 (그림 6)에서 보는 바와 같이 15-클라이언트에서 서버 CPU와

네트워크 대역폭에 병목현상이 발생되지 않는다는 사실이 뒷받침을 해 준다. O2PL-I의 성능추이도 이런 맥락에서 해석된다.



(그림 5) (a) 트랜잭션 처리율 (b) 메시지 수



(그림 6) (a) 자원 사용율 (b) 철회율과 캐쉬히트율

AOCC는 기본적으로 트랜잭션 대기를 야기하지 않고, DL은 비교적 낮은, 20-클라이언트에서 20% 정도의, 트랜잭션 대기율을 보이므로 클라이언트와 서버의 자원이 효율적으로 사용된다. 그렇지만 이 기법들은 NumClients가 15 이상이 되면 네트워크 사용집중으로 인한 성능 병목현상이 발생한다. 이것이 DL과 AOCC의 성능을 제한하는 가장 큰 요인이다. 15-클라이언트에서 DL은 C2PL에 비해 15% 이상의 우수한 트랜잭션 처리율을 보인다. 반면 C2PL-I는 C2PL에 비해 5% 정도의 트랜잭션 처리율 향상을 가져 온다. 이는 DL의 우수한 성능은 대부분 지연 로킹에 의한 메시지 부담 감소 및 트랜잭션 철회와 대기의 균형에서 비롯된다는 것을 의미한다.

(그림 5(b))를 살펴보면 회피기반 기법과 검사기반 기법은 뚜렷한 차이를 보인다. 회피기반 기법인 O2PL-I에서 NumClients의 증가에 따라 메시지 부담이 꾸준히 증가하는데, 다른 검사기반 기법은 그 증가율이 그다지 크지 않다. 이의 근본적인 원인은 중복사본 관리를 위해서 회피기반 기법은 한 트랜잭션의 갱신결과를 동기적으로 다른 클라이언트에 파급시키는 반면에, 검사기반 기법은 다른 메시지에 첨부하여 비동기적으로 파급시키기 때문이다. O2PL-I는 트랜잭션이 완료할 때마다 그 트랜잭션이 갱신한 데이타를 캐싱하고 있는 모든 클라이언트로 무효화를 위한 완료준비 메시지를 전송하고 응답을 받는다. 이 메시지 부담을 살펴보기 위하여 NumClients를 n이라 가정하자. 이 실험에서 트랜잭션은 평균 4개의 페이지를 갱신한다. 클라이언트의 버퍼 크기를 DbSize의 25%로 설정했기 때문에, 어떤 클라이언트가 갱신되는 4페이지를 전혀 캐싱하고 있지 않을 확률은 $(1-0.25)^4$, 즉 0.32가 될 것이다. 그러므로 서버는 트랜잭션의 완료시에 하나의 완료준비 메시지를 전송하고 $(n-1)\cdot(0.68)$ 수의 클라이언트로부터 응답 메시지를 받는다. 그러므로 O2PL-I에서 트랜잭션의 완료단계에서 갱신결과를 다른 클라이언트로 파급시키는데 필요한 메시지 수의 부담은 약 0.68의 경사를 갖고 증가된다. (그림 5(b))에서 메시지 수가 이보다 약간 높은 비율로 증가하는데, 이는 NumClients의 증가에 따라 전역 교착상태를 검출하기 위한 메시지 수와 트랜잭션 철회로 인하여 낭비되는 메시지 수도 약간 증가하기 때문이다.

검사기반 기법에서 한 트랜잭션의 완료 연산에는 단일 클라이언트와 서버만이 관여하므로, NumClients가 증가해도 완료단계의 메시지 부담이 증가하지는 않는다. 그러나, C2PL-I는 트랜잭션의 매 액세스마다 데이타 유효성 검사 및 적절한 로크 설정을 위하여 서버와 통신해야 한다. 그러므로 완료단계의 메시지 부담이 적음에도 불구하고 많은 수의 메시지 교환이 필요하다. NumClients가 증가하면 데이타 충돌이 심화되므로 트랜잭션 철회율이 증가한다. C2PL-I와 O2PL-I에서 트랜잭션 철회는 오직 로크 충돌에 의한 교착상태로 인하여 발생되므로, 증가율이 그다지 크지 않다. 그러나 DL과 AOCC는 NumClients가 증가함에 따라 비최신의 데이타를 읽을 가능성이 점점 높아지므로 철회율이 급격하게 증가한다. 두 기법에서 트랜잭션은 캐쉬 미스마다 자신이 액세스한 데이타의 유효성 검사를 실시하는데, DL은 일단 검증된 데이타의 유효성을 완료단계까지 보장하는데 비해, AOCC는 그렇지 않다. 그러므로 AOCC는 DL보다 높은 철회율을 보인다. 비록 각 기법이 다른 철회율을 보이지만, 이것이 그들의 성

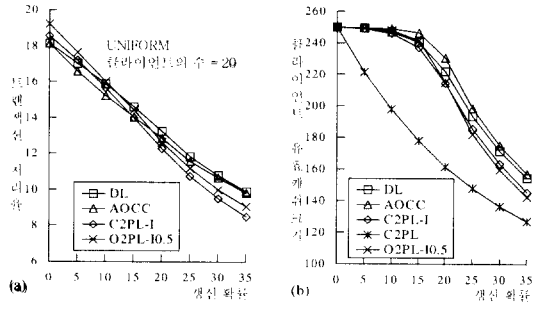
능과 메시지 부담에 심각한 영향을 미치지는 않는다. 왜냐하면 재실행되는 트랜잭션들은 그들이 필요로 하는 데이타를 이전 실행시에 자신의 버퍼로 옮겨 놓았을 가능성이 높기 때문이다. 캐쉬히트가 발생하면 트랜잭션 실행시에 클라이언트 CPU만이 사용되는데, 클라이언트에는 한 순간에 하나의 트랜잭션이 실행되므로 CPU 사용으로 인한 부담은 성능에 큰 영향을 미치지 않는다. 다만 C2PL기반 기법은 트랜잭션의 재실행시에도 매 액세스마다 서버와 통신하므로 철회는 메시지 수에 큰 영향을 미치는데, 이 기법들은 낮은 철회율을 보이므로 메시지 수는 크게 증가하지 않는다.

(그림 6(a))에서 주목할 점은 AOCC와 DL은 낮은 메시지 부담을 보임에도 불구하고 네트워크 사용집중으로 인한 성능저하가 발생한다는 것이다. 이는 AOCC와 DL은 낮은 트랜잭션 대기율을 보이므로 상대적으로 많은 트랜잭션들이 네트워크를 사용하기 때문이다. 또한, 이 기법들에서 제어 메시지의 크기가 첨부되는 별도의 정보때문에 다른 기법에 비해 크다는 사실도 여기에 일조를 한다. 메시지 교환의 수가 네트워크 대역폭의 사용에는 그다지 큰 영향을 미치지 않지만 서버 CPU의 사용율에는 큰 영향을 미친다. 따라서 C2PL-I와 O2PL-I는 DL과 AOCC보다 높은 CPU 사용율을 보인다.

NumClients의 증가는 캐쉬히트율에 서로 상반되는 두가지 효과를 가져온다. 첫째, NumClients가 증가하면 데이타 충돌을 심화되므로 트랜잭션 철회율이 높아진다. 이는 트랜잭션 재실행시의 높은 캐쉬히트로 인하여 전체 캐쉬히트율에 긍정적인 영향을 미친다. 반면, NumClients의 증가는 클라이언트 캐쉬를 무효화시키는 속도를 가속시킨다. 이는 결과적으로 유효캐쉬크기(effective cache size)의 감소로 이어져 히트율에 부정적인 영향을 미친다. 유효캐쉬크기는 유효한 데이타 페이지를 캐싱하고 있는 버퍼 슬롯(slot)의 수를 의미한다. (그림 6(b))에서 보는 바와 같이 15-클라이언트 지점까지 C2PL-I와 O2PL-I의 캐쉬히트율은 긍정적인 효과로 인하여 약간 증가하나, 이 이상에서는 부정적인 효과가 더 크게 작용하므로 감소한다. 반면, DL과 AOCC에서는 높은 철회율로 인하여 긍정적인 효과가 부정적인 효과를 항상 압도한다. 따라서 NumClients가 증가하면 캐쉬히트율도 증가한다.

5.1.2 갱신확률의 변화에 따른 성능

이 실험은 NumClients를 20으로 고정하고 갱신확률을 변화시키면서 실시되었다. O2PL-I의 가장 큰 단점은 트랜잭션의 완료단계에서 심각한 부담을 야기한다는 점이다. 하지만 이 부담은 갱신확률이 낮으면 그다지 크지 않다. 극단적으로 갱신 연산이 없으면 O2PL-I는 검사기반 기법과 동일한 완료 부담만을 필요로 한다. 그러므로 갱신확률이 0%~10% 구간에서는 O2PL-I가 가장 우수한 성능을 보인다. 이 구간에서 트랜잭션 대기율은 매우 낮기 때문에 네트워크 사용율은 각 기법에서 95% 이상에 육박한다. 그러므로 갱신확률이 낮으면 데이타 충돌보다는 네트워크 대역폭의 한계가 성능에 더 큰 영향을 미친다. 낮은 충돌환경에서 DL과 AOCC는 상대적으로 낮은 성능을 보이는데, 이는 제어 메시지의 크기가 다른 기법보다 크기 때문이다. 갱신확률의 증가에 따라 트랜잭션 완료당 필요한 메시지 수는 증가한다. 특히 C2PL-I와 O2PL-I에서 급격히 증가한다. C2PL-I에서는 재실행 부담으로 인하여 증가하며, O2PL-I에서는 한 트랜잭션의 완료에 더 많은 클라이언트들이 관여하므로 증가한다. DL과 AOCC에서는 철회율이 높아지더라도 재실행시의 높은 캐쉬히트로 인하여 메시지 부담이 크게 증가하지는 않는다.



(그림 7) (a) 트랜잭션 처리율 (b) 유효캐쉬크기

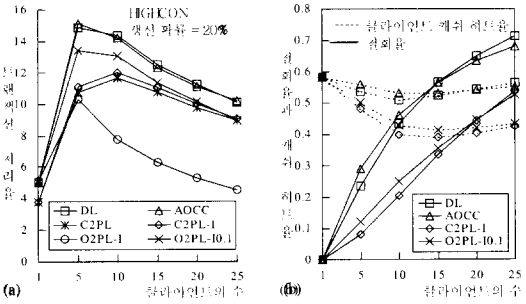
갱신확률의 증가는 5.1.1절에서 논의한 것처럼 캐쉬히트율에 서로 상반되는 효과, 즉 철회의 증가에 따른 재실행시의 높은 히트율과 버퍼의 빈번한 무효화로 인한 유효캐쉬크기의 감소를 야기한다. (그림 7(b))에서 갱신확률의 증가에 따라 유효캐쉬크기는 꾸준히 감소한다. 이 원인을 분석하기 위해 클라이언트 C_1에서 트랜잭션 T_1이 실행된다고 가정하고, 참고로 C2PL-I의 유효캐쉬크기를 살펴보자. C2PL-I는 20%의 갱신확률에서 약 215페이지의 유효캐쉬크기를 보인다. 트랜잭션 철회를 무시한다면, 0.215의 캐쉬히트율을 보일 것이다. 그러므로 T_1의 실행중에 15.7(20 - 20 x 0.215)개의

새로운 페이지가 C_1의 버퍼에 캐싱될 것이다. T_1이 제기되어 완료되는 사이에 다른 클라이언트들에서 평균 19개의 트랜잭션들이 완료할 것이다. 만일 이 트랜잭션들이 갱신하는 데이타 페이지가 서로 겹치지 않는다면, 이들은 C_1의 버퍼에 있는 페이지중 평균 16.3(19 x 4 x 0.215)개의 페이지를 무효화시킬 것이다. 물론 갱신되는 페이지가 서로 겹칠 수 있기 때문에 이보다는 약간 적은 수의 페이지가 무효화될 것이다. 이와 같이 새로이 캐싱되는 페이지 수와 무효화되는 페이지 수가 비슷하므로 버퍼 용량은 더 크지만 유효캐쉬크기는 더 증가되지 않는다.

O2PL-I는 비최신의 데이타를 클라이언트 버퍼에 절대 캐싱하지 않으므로 유효캐쉬크기에 있어서 가장 우수한 성능을 발휘할 것 같지만 실제는 그렇지 않다. (그림 7(b))를 살펴보면, O2PL-I는 C2PL-I 수준의 유효캐쉬크기를 보인다. 이는 예 1에서 설명한 것처럼 O2PL-I는 잘못된 무효화를 야기할 수 있기 때문이다. C2PL을 제외한 다른 검사기반 기법들은 클라이언트 버퍼의 효율적인 사용을 위해 동일한 비동기적 무효화 정책을 사용하는데, DL과 AOCC는 C2PL-I보다 큰 유효캐쉬크기를 보인다. 이는 DL과 AOCC는 비최신의 데이타를 액세스하여 서버에서 트랜잭션의 철회가 결정되면 최신의 데이타를 전송하기 때문이다. AOCC는 DL보다 높은 트랜잭션 철회율을 보이며, 트랜잭션 대기가 발생하지 않아 더 자주 새로운 데이타를 클라이언트로 전송하므로 DL보다 큰 캐쉬크기를 보인다. (그림 7(b))에서 C2PL은 다수의 버퍼 슬롯을 비최신의 데이타를 캐싱하는데 낭비하고 있음을 파악할 수 있다. 이 비효율적인 버퍼의 사용이 논문 전체에 나타나는 C2PL의 낮은 캐쉬히트율에 대한 근본적인 원인이다.

5.2 실험 2-HIGHCON 부하

이 부하는 모든 트랜잭션이 HIGHCON 영역이라는 데이타베이스의 특정 부분을 매우 높은 확률로 액세스하는 환경을 모델링한다. 데이타베이스에는 250-페이지의 HIGHCON 영역이 있고, 트랜잭션이 실행하는 연산의 80%는 이 영역을 액세스한다고 가정한다. 그러므로 이 부하는 심한 데이타 충돌과 높은 참조 국부성의 특징을 지닌다. (그림 8(a))에서 보는 것과 같이 DL과 AOCC는 다른 기법에 비해 매우 우수한 성능을 발휘한다. 이는 UNIFORM 부하에서 설명한 것처럼, 이들은 낮은 트랜잭션 대기율을 보이며 메시지 부담이 적기 때문이다. 이전 연구에서는[3, 7] 이 실험에서 C2PL은 O2PL보다 우수한 성능을 발휘하는데, 이는 O2PL의 전역 교착상태 검사주기를 1초로 설정하여 교착상태 발견까지의 긴 시간지연으로 인한 것으로 해석된다. 교착상태 검사주기를 0.1초로 설정함으로써 O2PL-I의 성능을 상당히 향상시킬 수 있다는 사실이 이 해석을 뒷받침한다.



(그림 8) (a) 트랜잭션 처리율 (b) 철회율과 캐쉬히트율

이 실험에서 대부분의 데이타가 HIGHCON 영역에서 액세스되고 트랜잭션 철회율이 높기 때문에, 각 기법은 UNIFORM 부하보다 이 부하에서 높은 캐쉬히트율을 보인다. NumClients가 증가함에 따라 트랜잭션 철회율이 급격하게 증가하는데, 트랜잭션 재실행시의 캐쉬히트가 UNIFORM의 경우만큼 그렇게 높지 않다. 이는 전 실행에서 클라이언트 버퍼로 가져다 놓은 데이타들이 곧 다시 무효화될 가능성이 높기 때문이다. 그러므로 NumClients의 증가에 따라 메시지 부담은 UNIFORM 부하보다 훨씬 급격하게 증가한다. NumClients에 따른 메시지 수의 변화추이 및 요구되는 메시지 수의 상대적인 순서는 UNIFORM 부하와 거의 동일하다.

(그림 8(b))에서 주목할 점은 트랜잭션 철회율에 있어서 DL과 AOCC기법간에 역전이 발생한다는 것이다. 이는 (그림 6(b))의 철회율 변화추이에서 어느 정도 예상된다. (그림 6(b))에서 데이타 충돌이 심화됨에 따라 DL과 AOCC 기법간의 철회율의 차이는 점점 커지는데, 데이타 충돌이 어느 이상으로 극심해지면 오히려 차이는 줄어들기 시작한다. 그러므로 마침내 HIGHCON과 같은 극심한 데이타 충돌환경에서 역전 현상이 발생하는 것이다. 낮은 데이타 충돌환경에서 로크를 인증받기 위하여 대기하고 있는 트랜잭션은 추후에 로크를 인증받아 실행을 계속하게 될 가능성이 높기 때문에 철회율을 줄이는데 큰 공헌을 한다. 그러나 데이타 충돌이 심화됨에 따라 트랜잭션 대기는 대부분 교착상태로 이어

져 철회에 대한 로크의 효과가 급격하게 줄어든다. AOCC는 데이타 충돌로 인한 대기를 야기하지 않기 때문에 트랜잭션이 많이 진행된 후에 철회되는 경향이 있는데, 이도 극심한 충돌환경에서 긍정적으로 작용한다.

이에 대한 이해를 위해, 끝까지 실행된 트랜잭션 T_1이 철회되어 처음부터 다시 재실행된다고 가정하자. 서버가 T_1의 철회를 결정하면, 서버는 T_1이 액세스한 데이타 중 무효화된 데이타의 최신 버전을 T_1의 철회 메시지에 첨부하여 클라이언트로 전송한다. T_1의 재실행시에는 캐쉬미스가 발생하지 않으므로, T_1은 클라이언트에서만 실행되면서 완료단계에 도달한다. T_1은 매우 빠르게 진행되었기 때문에, T_1의 실행중에 상대적으로 적은 수의 트랜잭션이 완료에 성공했을 것이다. 따라서 적은 수의 페이지가 무효화될 것이다. 그러므로 AOCC에서 T_1의 완료성공 가능성은 매우 높아진다. 그러나 DL에서는 T_1이 비록 유효한 데이타를 액세스했더라도 충돌하는 W-lock 또는 C-lock으로 인해 다시 철회될 가능성이 있고, 상충하는 R-lock으로 인하여 대기할 수도 있다. 그러므로 트랜잭션의 재실행시에는 AOCC가 DL보다 낮은 철회율을 보인다. 실제 실험 결과도 이 주장을 뒷받침한다 ; 15-클라이언트에서 두 기법은 모두 56% 정도의 트랜잭션 철회율을 보이는데, 트랜잭션이 처음 실행에서 철회되는 확률은 DL과 AOCC에서 각각 59%, 67%이다. 이는 반대로 재실행되는 트랜잭션의 철회율은 AOCC가 DL보다 낮다는 것을 의미한다. C2PL-I와 O2PL-I0.1간의 트랜잭션 철회율의 관계도 이런 맥락에서 설명된다.

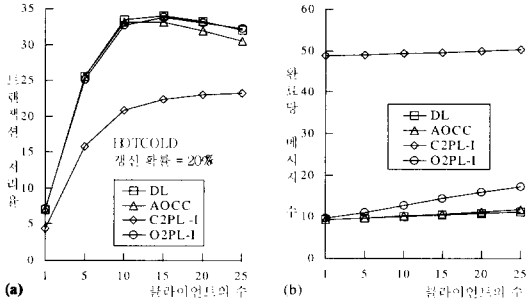
이 실험에서 HIGHCON 유효캐쉬크기(effective HIGHCON cache size)를 살펴보았다. HIGHCON 유효캐쉬크기란 용어는 전체 버퍼에서 HIGHCON 영역의 최신 데이타 페이지를 캐싱하고 있는 버퍼 슬롯의 수를 의미한다. NumClients의 증가에 따라 트랜잭션 철회율이 급격히 증가하므로 캐쉬히트율도 증가할 것 같지만, 10-클라이언트 지점까지의 결과는 이와 상반된다. 이는 NumClients가 증가하면 HIGHCON 유효캐쉬크기가 심하게 감소하기 때문이다. HIGHCON 영역의 데이타는 캐싱될 가능성이 매우 높으나, 그들은 또한 무효화될 가능성도 높다. NumClients가 1의 경우처럼 버퍼 무효화가 전혀 발생하지 않으면, 각 기법은 전체 유효 버퍼 슬롯의 70% 이상을 HIGHCON 데이타를 캐싱하는데 사용한다. 그러나 10-클라이언트에 도달하면 그 사용율은 40% 정도로 급격히 감소한다. 전체 유효캐쉬크기는 HIGHCON 유효캐쉬크기에 비해서 그다지 심각하게 감소하지 않는다. 25-클라이언트에서도 C2PL을 제외한 다른 기법들은 대체로 200-페이지 내외의 유효캐쉬크기를 보인다. 이는 non-HIGHCON 데이타는 일단 캐싱되면 오랫동안 버퍼에 간직될 수 있기 때문이다. HIGHCON 데이타의 높은 무효화율로 인하여 빈 버퍼 슬롯이 존재할 가능성이 높기 때문에 non-HIGHCON 데이타가 LRU 버퍼 교체 알고리즘에 의해 희생자로 선택될 가능성이 낮음에 유의해야 한다.

DL은 데이타 충돌로 인한 트랜잭션 대기를 야기하며, AOCC와 비슷한 수준의 철회율을 보인다. 그런데도 불구하고 두 기법이 비슷한 성능을 보이는 이유는 트랜잭션 철회당 낭비되는 시간과 밀접한 관련이 있다. 15-클라이언트에서 DL은 AOCC에 비하여 70% 정도의 트랜잭션 철회당 낭비되는 시간을 보였다. 이 시간에는 로크 충돌로 인한 대기시간도 포함되어 있다. 이와 같은 결과를 보인다는 것은 AOCC의 경우 트랜잭션이 훨씬 많이 진행된 상태에서 철회된다는 것을 의미한다. 이는 물론 트랜잭션 재실행시의 철회율 관점에서는 바람직하지만 자원의 사용 관점에서는 바람직하지 않다.

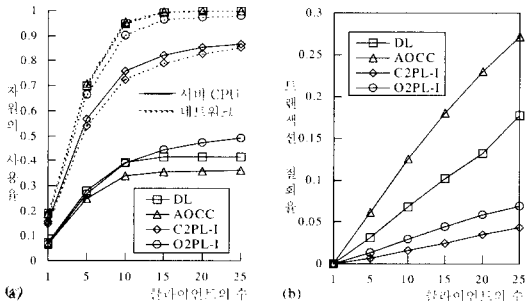
5.3 실험 3-HOTCOLD 부하

이 부하에서 각 클라이언트는 데이타베이스에서 자신만이 자주 액세스하는 40-페이지의 HOT 영역을 가지고 있다고 가정한다. 한 클라이언트의 HOT 영역은 다른 클라이언트의 HOT 영역과는 완전히 분리되어 있고, 트랜잭션은 필요로 하는 데이타의 80%를 자신의 HOT 영역에서 액세스한다고 가정한다. 따라서, 이 부하는 높은 참조 국부성과 낮은 데이타 충돌의 특징을 보인다. 이 실험은 버퍼 크기가 작더라도 높은 캐쉬히트율을 보이므로 ClientBufSize를 DbSize의 10%로 설정하였다.

클라이언트 수에 따른 트랜잭션 처리율 및 메시지 부담이 (그림 9)에 있다. C2PL-I는 다른 기법에 비해 높은 메시지 부담을 보이므로 상대적으로 낮은 트랜잭션 처리율을 보인다. 이 부하에서 클라이언트는 자신만의 HOT 영역을 높은 확률로 액세스하므로 각 클라이언트에 캐싱된 데이타가 서로 겹칠 가능성이 다른 부하에 비해 매우 낮다. 이는 O2PL-I에서 완료 부담의 감소로 이어지고, DL과 AOCC에서는 비최신의 데이타를 액세스할 가능성의 감소로 이어진다. 그리고 이 세 기법들은 서버와 통신하지 않고 중복사본을 액세스하므로 트랜잭션 실행단계의 부담이 거의 비슷할 것이다. 그러므로 이 세 기법은 이 부하에서 비슷한 성능을 보인다.



(그림 9) (a) 트랜잭션 처리율 (b) 메시지 수



(그림 10) (a) 자원 사용율 (b) 트랜잭션 철회율

(그림 10)은 자원의 사용율과 트랜잭션 철회율을 나타낸다. 서버 CPU 사용율의 차이는 대부분 클라이언트와 서버사이에 교환되는 메시지 수의 차이에서 비롯된다. 그러므로 C2PL-I는 상대적으로 높은 CPU 사용율을 보인다. C2PL-I는 클라이언트와 서버의 CPU에서 메시지를 처리하기 위한 지연때문에 네트워크 사용율이 낮은 편이다. (그림 10(a))에서 주목할 것은 DL, AOCC, 그리고 O2PL-I는 메시지 교환이 적음에도 불구하고 그들의 성능은 네트워크 사용집중으로 인한 병목현상이 발생한다는 점이다. 이를 이해하기 위해서는 단위시간에 전달되는 메시지의 양을 조사해야 한다. 이 실험에서 한 트랜잭션의 성공적인 종료를 위해 평균 8개의 데이타 페이지가 네트워크를 통해 전송되어야 한다. 이중 4개의 페이지는 캐쉬미스로 인한 데이타전송이고(이 실험에서 평균 80%의 캐쉬히트율을 보임.) 나머지 4 페이지는 완료시 클라이언트에서 갱신된 내용을 서버로 전송하기 위함이다. 이 실험에서 이 세 기법은 C2PL-I에 비해서 대략 1.5배 정도의 트랜잭션 처리율을 보인다. 이는 C2PL-I에서 하나의 트랜잭션이 제거되어 완료하는 동안에 이들은 C2PL-I보다 4개의 페이지를 더 전송함을 의미한다. 이 메시지의 양을 네트워크 대역폭 관점에서 살펴보면 64개의 제어 메시지에 해당한다. 따라서 이 세 기법에서 네트워크의 사용집중으로 인한 병목현상이 더 심각한 것은 당연하다. C2PL에서 교환되는 메시지 수는 상당히 많지만 높은 캐쉬히트로 인하여 대부분의 메시지는 제어 메시지일 것이라는 사실을 유념해야 한다. 이 실험에서 모든 기법들은 거의 비슷한, 대략 80% 정도의 캐쉬히트율을 보인다.

HOTCOLD 부하에서 DL과 AOCC의 성능은 네트워크 대역폭의 한계로 인하여 제한된다. 이 대역폭의 한계를 완화하기 위하여 NetBandwidth를 50Mbps까지 증가시키면서 성능추이를 살펴보았다. NetBandwidth가 증가되면 CPU와 디스크와 같은 다른 자원의 사용율이 증가하여 병목현상이 발생할 것이라는 점을 감안하여, 디스크의 속도는 기본속도의 3배로, 클라이언트와 서버 CPU의 속도는 기본값의 2배로 설정하고 실험을 실시하였다. NumClients는 시스템의 향상된 자원을 사용할 충분한 트랜잭션을 제공하기 위하여 25로 고정하였다. 이 실험에서 C2PL-I를 제외한 다른 기법의 성능은 NetBandwidth가 30Mbps가 될 때까지 거의 선형적으로 증가한다. C2PL-I는 많은 수의 메시지 교환을 필요로 하므로 NetBandwidth가 20Mbps에 도달하면 서버 CPU의 사용율은 90% 정도에 육박한다. NetBandwidth가 30Mbps에 도달하면 CPU의 과다사용으로 병목현상이 발생해 성능이 아주 미약하게 증가한다. 다른 기법들의 성능도 NetBandwidth가 40Mbps가 되면 디스크의 과다사용으로 병목현상이 발생하기 시작한다. AOCC는 향상된 시스템 자원을 효율적으로 사용하지만 DL보다 높은 철회율을 보인다. 트랜잭션 재실행시의 높은 캐쉬히트의 효과가 전체 캐쉬히트에 미치는 영향이 이 부하에서는 그다지 크지 않다. 클라이언트는 자신만의 HOT 영역에서 대부분의 데이타를 액세스하므로 트랜잭션은 처음 실행시에도 높은 캐쉬히트율을 보이기 때문이다. 그리고 DL에서 로크 충돌로 인한 트랜잭션 대기의 대부분은 추후에 로크가 인증되어 다시 실행이 가능할 것이다. 이와 같은 이유로 인해서 NetBandwidth가 50Mbps까지 증가되는 동안 DL은 AOCC보다 미약하기는 하지만 항상 우수한 성능을 보인다.

6. 결 론

본 논문에서는 주사본 로킹 기법에 근간을 둔 새로운 검사기반의 TCCM 기법인 DL 기법을 제안하고, 그 성능을 모의실험을 통하여 분석하였다. DL에서 클라이언트는 자신의 버퍼에 캐쉬되어 있는 데이타를 서버와 통신하지 않고 액세스할 수 있다. 액세스된 중복사본의 유효성 검사 및 로크 설정은 캐쉬미스로 인하여 클라이언트와 서버사이에 통신이 반드시 필요할 때 요청되어 비동기적으로 이루어진다. 그러므로 DL은 주

사본 로킹에 기초한 기법의 가장 큰 단점으로 알려진 통신부담을 상당히 줄일 수 있다. 그리고 DL 기법은 클라이언트 트랜잭션의 갱신결과를 다른 클라이언트로 파급시키기 위하여 비동기적 무효화 정책을 사용하므로, 회피기반 기법에서 갱신 연산으로 인하여 발생하는 부담을 상당히 경감시킬 수 있다.

본 논문에서는 DL 기법과 검사기반 기법중 가장 대표적인 AOCC와 C2PL과의 성능을 비교하였다. DL 기법은 C2PL에 비하여 훨씬 적은 수의 메시지 부담을 야기하므로 다양한 부하환경에서 월등한 성능을 발휘한다. DL 기법과 AOCC는 클라이언트에 캐쉬된 데이타를 액세스한 후 그의 유효성을 검증받기 때문에 비슷한 수준의 통신부담을 야기한다. 그러므로 이 두 기법은 매우 비슷한 성능을 발휘한다. 그러나 이들은 동시성 제어 측면의 차이로 인해, DL은 일단 검증된 데이타의 유효성을 완료단계까지 보장하는데 비해, AOCC는 그렇지 않다. 그러므로 DL은 매우 극심한 데이타 충돌환경이외에는 AOCC에 비해 훨씬 낮은 트랜잭션 철회율을 보인다. DL 기법은 회피기반 기법중 가장 성능이 우수한 것으로 알려진 O2PL 1보다도 우수한 성능을 발휘한다. 비록 DL 기법이 C2PL에 비해서 높은 트랜잭션 철회율을 보이지만, 이 요인도 높은 충돌환경에서는 성능을 높이는데 어느 정도 공헌을 한다. 그 이유는 DL 기법은 데이타 사용경쟁의 해결을 단순히 트랜잭션 대기에만 의존하는 것이 아니라, 조건에 따라 철회 및 대기를 결정하기 때문에 트랜잭션 대기와 철회 사이에 어느 정도 균형을 맞추어 동시성의 정도를 향상시킬 수 있기 때문이다. AOCC 기법은 데이타 충돌의 해결을 전적으로 트랜잭션의 철회에만 의존하므로 자원의 낭비가 너무 심한 경향이 있다.

DL 기법에서 한 트랜잭션의 실행 및 완료에는 그 트랜잭션을 제기한 클라이언트와 서버만이 관여하므로, 회피기반 기법에 비하여 클라이언트의 고장에 훨씬 견고하다고 볼 수 있다. 이 장점은 점점 보편화되고 있는 이동 응용에서 이동 클라이언트의 간헐적인 통신장애를 고려한다면 점점 부각될 것이다. 그러나 DL 기법은 회피기반 기법에 비해서 높은 철회율을 보인다. 미래 연구 과제로서 이 단점을 극복하기 위한 연구가 진행될 것이다. 그리고 데이타전송과 질의전송이 조합된 시스템을 위한 TCCM 기법에 대한 연구가 진행될 것이다.

참 고 문 헌

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

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

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

[4] 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.

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

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

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

[8] 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.

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

[10] 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.

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

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

[13] D. L. Mills, "Network Time Protocol : Specification and Implementation," DARPA-Internet Report RFC 1059, DARPA, 1988.

[14] M. T. Özsu, K. Voruganti, and R. C. Unrau, "An Asynchronous Avoidance-Based Cache Consistency Algorithm for Client Caching DBMSs," Proc.

Of Int. Conf. On VLDB, pp.440-451, 1998.
[15] H. Schwetman, 'CSIM Users' Guide for Use with CSIM Revision 16', Microelectronics and Computer Technology Corporation, 1992.
[16] M. Stonebraker, L. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, P. Bernstein, and D. Beech, "Third-Generation Data Base System Manifesto," ACM SIGMOD Record, Vol.19, No.3, pp.31-44, 1990.
[17] Y. Wang and L. A. Rowe, "Cache Consistency and Concurrency Control in A Client/Server DBMS Architecture," Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp.367-376, 1991.
[18] K. Wilkinson and M. Neimat. "Maintaining Consistency of Client Cached Data." Proc. Of Int. Conf. On VLDB, pp.122-133, 1990.

권 혁 민

e-mail : hmkwon@venus.semyung.ac.kr

1984년 서울대학교 제어계측공학과(학사)

1994년 한국과학기술원 정보 및 통신공학과(공학석사)

1998년 한국과학기술원 정보 및 통신공학과(공학박사)

1984년~1991년 대우전자 중앙연구소 컴퓨터개발부 선임연구원

1999년~현재 세명대학교 소프트웨어학과 전임강사

관심분야 : 트랜잭션 처리, 분산 데이타베이스, 멀티미디어 데이타베이스