「정보시스템연구」 제6권 제1호
한 국 정 보 시 스 템 학 회
1997. 6, pp. 159~180

공유 데이타베이스 환경에서 고성능 트랜잭션 처리를 위한 버퍼 무효화 기법

김 신 희* · 배 정 미** · 강 병 욱***

< 목 차 >

Ⅰ. 서 론

1.1. 연구 배경

일반적으로 은행 업무, 공장제어, 비행기 예약업무와 같은 온라인 트랜잭션 처리를 요하는 많은 응용 프로그램들은 고성능의 트랜잭션 처리 시스템을 필요로 한다. 그러한 시스템에서는 다음과 같은 사항을 충실히 제공할 수 있어야 한다.

* 대경전문대학 컴퓨터정보과 겸임교수
** 대경전문대학 컴퓨터정보과 조교수
*** 영남대학교 컴퓨터공학과 교수

첫째, 높은 트랜잭션 처리율(transaction rate)을 보장할 수 있어야 한다. 실례를 들어, 은행 업무 처리에 있어서 현재의 트랜잭션 처리 시스템은 초당 200개의 트랜잭션(200TPS)까지 처리 가능하고(Anon, 1985), 십만개 이상의 터미날과 1000개 이상의 디스크를 가진 대형 트랜잭션 처리 시스템의 경우 초당 수천개의 트랜잭션 처리가 가능하다(Gray, 1985).

둘째, 높은 가용성(availability)을 제공하여야 한다. 일반적으로 정전으로 인한 기계의 동작 중지는 일년에 5분미만일 것이 요구되는데, 이를 위해서는 하드웨어와 소프트웨어 자원 특히 처리 노드가 중복적으로 제공되어야 한다. 또한 시스템 내 각 요소의 고장이 사용자에게 영향을 미치지 않아야 하고, 시스템 구성상의 변화는 온라인으로 바로 적용되어야 하며, 공유되는 데이타베이스는 어떤 시점에서도 일관되고 최신의 상태를 반영하고 있어야 한다.

세째, 시스템의 확장이 용이하여야 한다. 트랜잭션 처리율은 새로운 처리 노드가 추가됨에 따라 증가한다. 따라서 처리 노드를 추가하여도 온라인 트랜잭션의 응답시간(response time)은 영향을 받지 않아야 한다.

마지막으로, 유지관리가 쉬워야 한다. 시스템은 단말 사용자 또는 프로그래머에게 고수준의 인터페이스를 제공함으로써 다수의 처리 노드로 이루어진 복잡한 구성에 대해 신뢰성을 제공할 수 있어야 한다.

이러한 요구조건은 하나의 공유 메모리(shared memory)와 운영체제 그리고 DBMS를 갖는 밀결합 시스템(tightly coupled system)에서는 제공되기 어렵다. 특히 하나의 공유 메모리는 시스템 파손이나 경합(contention)의 주된 원인이기 때문에 충분한 신뢰성과 확장성을 제공하지 못한다. 이러한 이유로 해서 소결합 시스템(loosely coupled system)이 많이 이용된다. 소결합 시스템에서 각 처리 노드는 별도의 메모리와 운영체제 그리고 DBMS를 가지며, 메시지를 이용하여 처리 노드 간에 통신을 한다. 따라서 처리 노드 간의 독립성을 최대한 보장함으로써 서로의 간섭을 줄이고, 최대의 가용성을 제공할 수 있다. 그러나 처리 노드 간의 통신 오버헤드가 시스템의 성능에 상당한 영향을 미친다.

분산 트랜잭션 처리 시스템(distributed transaction processing system)은 크게 이질형(heterogeneous)과 동질형(homogenous) 구조로 나누어진다.

이질형 시스템은 기존에 독자적으로 개발되어 사용되고 있는 서로 다른 지역 데이타베이스 시스템이 통신망을 통하여 논리적으로 통합되어 있다. 이 시스템은 하나이상의 데이타베이스를 액세스하는 응용 프로그램에 대해서도 처리가능하지만, 각 처리 노드가 상이한 사용자 인터페이스, 데이타 모델, 트랜잭션 관리 기법을 사용하므로 처리 노드의 자치성(autonomy)을 보장하기 위한 부담이 따른다. 클라이언트/서버(client/server) 시스템이 이질형으로 분류된다(Breibart, 1992, Du, 1989).

동질형 시스템에서 모든 처리 노드는 동일한 트랜잭션 처리 기능을 가진다. 동질형에 해당하는 부류에는 메모리 공유 시스템(shared memory system)과 데이타베이스 분할 시스템(database

partitioning system) 그리고 데이타베이스 공유 시스템(DataBase Sharing System: DBSS) 등이 있다. 메모리 공유 시스템은 밀결합된 다중 처리 노드 상에서 수행되는 트랜잭션 처리 시스템으로서, 주기억장치와 주변장치를 공유하며 소프트웨어의 변화가 거의 없는 시스템으로서, 중앙집중식 트랜잭션 처리에 적합하다. 데이타베이스 분할 시스템은 소결합된 자치적인 처리 노드들로 구성되는데, 각 처리 노드는 자신의 디스크를 가지며 데이타베이스는 분할되어 처리 노드의 디스크에 분산되어 저장된다. 지역 데이타(local data)에 대한 요구는 처리 노드 내에서 바로 처리되며, 전역 데이타(remote data)는 해당 데이타를 가진 처리 노드로 보내져서 처리된다. DBPS의 상용 제품으로는 IBM사의 CICS(IBM Manual, 1987), Tandem사의 Encompass(Borr, 1981) 등이 있다.

DBSS에서 처리 노드들은 하나의 데이타베이스를 공유하므로 전체 데이타베이스를 바로 접근할 수 있다. 본 연구에서는 소결합된 DBSS에 대해 고찰한다.

1.2 연구 동기

DBSS은 고성능의 온라인 트랜잭션 처리가 필요한 응용분야를 효율적으로 지원하기 위하여 제안된 시스템이다. 별도의 메모리와 운영체제 그리고 DBMS을 갖는 여러개의 처리 노드로 구성되며 각각의 처리 노드는 디스크 레벨에서 하나의 데이타베이스를 공유한다. 뿐만 아니라 처리 노드들은 물리적으로 인접한 위치에 존재하며, 고속의 통신 시스템을 이용하여 메시지 교환을 통해 교신한다. DBSS의 상용 제품으로 IBM사의 IMS/VS 데이타 공유 버전(Strickland, 1982), AMOEBA project(Shoens, 1985), DEC사의 VAX DBMS(Kronenberg, 1986) 등이 있다.

DBSS의 장점은 높은 융통성을 가진다는 데 있다. 즉, 각 처리 노드의 DBMS은 전체 데이타베이스를 액세스할 수 있고, 다른 처리 노드의 DBMS들과 상호 독립적으로 동작하기 때문에 현재 시스템 상태에 따라 동적으로 처리 노드에 트랜잭션을 분배함으로써 부하 균형(load balancing)이 가능하고, 트랜잭션 프로그램이나 데이타베이스 스키마를 변경하지 않고도 새로운 처리 노드와 DBMS를 추가시킬 수 있다. 또한 하나의 처리노드의 고장이 시스템 내 다른 처리 노드의 DBMS가 데이타베이스를 액세스하는 데 영향을 주지 않고, 고장난 처리 노드에서 수행중이던 트랜잭션은 철회(roll-back)되어 자동적으로 다른 이용가능한 처리 노드에 재분배된다.

그러나 DBSS은 다음과 같은 문제점을 가진다.

첫째, 동기화(synchronization)의 문제가 있다. DBSS에서는 공유 메모리를 갖지 않으므로 메시지를 통해 동기화 문제를 해결하는데, 이로 인해 중앙집중형 DBMS에 비해 로크 요청을 처리하기 위한 시간이 더 필요하다. 즉 트랜잭션의 응답시간이 더 길어진다. 이러한 상황에서 처리율을 일정하게 유지하기 위해서는 트랜잭션당 평균 동기화 메시지 수를 최소화할 수 있는 동기화 알고리즘이 필요하다.

둘째, 버퍼 무효화(buffer invalidation)를 해결하기 위한 버퍼 제어(buffer control)기법이 필요하다. 각각의 처리 노드는 전체 데이타베이스를 액세스할 수 있고, 자신의 지역 데이타베이스 버퍼를 가지기 때문에 임의의 처리 노드에서 어떤 데이타를 갱신하게 되면 해당 데이타를 캐싱하고 있는 다른 모든 처리 노드에도 갱신이 반영되어야만 일관성이 유지된다.

세째, 이용가능한 처리 노드들에 현재의 작업부하(workload)를 효과적으로 분배하기 위한 부하 제어(load control)기법이 필요하다. 각 처리 노드의 부하는 최소의 상태이어야 하며, 최소의 통신비용으로 동기화가 가능하도록 라우팅(routing)기법이 지원되어야 한다. 부하제어가 원만하게 이루어지기 위해서는 작업부하 또는 시스템 구성상의 변화에 대해 동적으로 대처할 수 있어야 한다. 일반적으로 트랜잭션 라우팅은 트랜잭션의 데이타 참조 유형을 분석함으로써 가능하다. 이에 대해서는 (Rahm, 1986, Reuter, 1986)에서 상세하게 언급하고 있다.

마지막으로, 시스템 파손에 대한 복구(recovery)의 문제가 있다. 각 처리 노드에 지역 로그 정보를 기록한 지역 저널(local journal)을 두고 이를 병합(merge)함으로써 시스템 전체에 대한 전역 저널(global journal)을 구성할 수 있다. 복구기법은 한 처리 노드가 고장나더라도 트랜잭션 처리를 계속하기 위해 전역 저널을 이용하여 고장난 처리 노드를 복구하여 재통합할 수 있어야 한다. 고장난 처리 노드의 완료되지 못한 트랜잭션은 철회되어 다른 처리 노드에서 다시 시작되고, 사용자는 이러한 사실을 알 필요가 없다(transparency).

DBSS에서 공유 데이타의 일관성을 유지하기 위하여 동시성 제어 기법이 필요하다.

동시성 제어 기법으로 로킹(locking)이 많이 쓰인다. DBSS에서 로킹을 구현하는 기법으로는 중앙집중형과 분산형이 있다(Rahm, 1993).

중앙집중형 기법은 공유 데이타에 대한 모든 로킹 정보를 관리하는 하나의 전역 로크 관리자(Global Lock Manager: GLM)가 존재한다. 각 처리 노드에서 로크를 요청하거나 해제하기 위해서는 반드시 GLM을 통해야만 하므로 GLM으로의 통신 집중에 따른 성능저하를 초래하고, GLM 처리 노드의 고장시 전체 시스템이 마비되므로 신뢰성이 떨어진다.

분산 기법에서 공유 데이타에 대한 로킹 정보는 DBSS를 구성하는 처리 노드들에 나누어 저장된다. 각 처리 노드들이 액세스하는 데이타는 자신이 로킹 정보를 관리하는 지역 데이타와 다른 처리 노드들에 의해 로킹 정보가 관리되는 원격 데이타로 나누어진다. 중앙집중형 기법에 비해 지역 데이타를 액세스하는 경우에는 로킹에 관련된 통신이 필요없으므로 성능을 향상시킬 수 있고, 로킹 정보가 여러 처리 노드에 나누어 저장되므로 신뢰성을 향상시킬 수 있다. 이러한 지역 데이타 지정 방식을 주사본 권한 (Primary Copy Authority: PCA)이라 한다(Reuter, 1984).

PCA의 기본 개념은 각 처리 노드의 부하를 균등하게 유지하면서 동일한 데이타를 액세스하는 트랜잭션들은 해당 데이타에 대한 PCA를 가지고 있는 처리 노드에서 실행하도록 함으로써 트랜잭션당 요구되는 전역 로크 요청 횟수를 줄인다는 것이다. 따라서 성능향상을 위해서는 각 처리 노드의 현재 부하와 입력되는 트랜잭션들의 참조 패턴(reference pattern)을 고려하여 PCA

를 적절하게 할당하여야 한다. 실제로 은행 업무에서의 트랜잭션(debit-credit transaction)과 같은 아주 이상적인 경우(Gray, 1991)를 제외한 대부분의 경우, 트랜잭션 경로 배정을 위한 정책을 적용하여도 트랜잭션에 의해 발생하는 로크 요청의 약 30%만이 지역적으로 처리된다는 사실을 감안할 때 시스템의 성능은 PCA를 관리하는 방식에 상당히 의존한다고 볼 수 있다(Reuter, 1986).

본 논문에서는 PCA를 동적으로 관리함으로써 보다 효율적으로 공유 데이타의 일관성을 유지할 수 있는 버퍼 무효화 기법(buffer invalidation)을 제안한다. 제안된 기법은 각 처리 노드에서의 디스크 I/O 오버헤드를 줄임으로써 성능을 향상시킬 수 있다. 또한 시스템 환경의 변화에 대한 적응성을 제공하고, 부하 불균형 문제를 해결하므로 고성능의 트랜잭션 처리가 가능하다.

논문의 구성은 다음과 같다. 2장에서 관련 연구로서 주사본 로킹 기법과 로크 보유 기법에 대해 기술한다. 3장에서 기존의 정적 PCA 관리 방식과 본 논문에서 제안한 동적 PCA 관리 방식에 대해 비교분석하고, 4장에서는 제안된 버퍼 무효화 기법에 대해 기술한다. 5장에서 성능 평가를 위해 구성한 시뮬레이션 모델을 설명하고, 6장에서 시뮬레이션 결과를 비교 분석한다. 마지막으로 7장에서 결론 및 앞으로의 연구 방향을 기술한다.

II. 관련연구

본 장에서는 본 연구의 근거가 되는 동시성 제어 기법인 주사본 로킹과 성능 개선을 위해 도입하는 로크 보유 기법에 대해 기술한다.

2.1 주사본 로킹

DBSS에서 PCA를 이용한 트랜잭션 처리 방식으로 주사본 로킹(Primary Copy Locking: PCL)이 있다(Ceri, 1984, Rahm, 1986).

PCL은 처리 노드의 갯수만큼 데이타베이스를 논리적으로 분할하여 처리 노드에게 각 분할(partition)에 대한 PCA를 할당함으로써 로크 관리의 책임을 전체 시스템에 분산시키는 방식이다. PCA를 가진 처리 노드는 해당 분할에 대한 로크 관리를 전적으로 담당한다. 임의의 처리 노드에서 지역 데이타에 대한 로크 요청이 발생하면 통신 오버헤드나 지연없이 해당 노드에서 바로 처리되며, 그렇지 않으면 PCA를 가진 처리 노드에 로크 요청 메시지를 보내야 한다.

PCL은 다음과 같은 문제점을 가진다.

첫째, 균형적인 부하 분배(load balancing)가 이루어지지 않는다는 것이다. 즉, PCA는 각 처

리 노드에 정적으로 할당되므로 해당 분할에 대한 모든 로크 요청은 처리 노드의 현재 부하상태와 상관없이 PCA 노드로 집중된다. 따라서 시스템의 성능 향상을 위해서는 효율적인 부하 균형 기법과 PCA 할당에 관련된 공유 데이타베이스 분할 기법이 요구되는데, 일반적으로 PCL에서는 이들 기법이 잘 수행되는 경우를 기본적으로 가정하고 있다.

둘째, 캐쉬 일관성(cache coherency)을 유지하기 위한 버퍼 무효화의 문제가 있다. 데이타 공유 환경 하에서는 동시에 다수의 처리 노드가 동일한 페이지 사본을 버퍼에 캐싱할 수도 있는데, 처리 노드 간의 일관성 유지를 위해 FORCE 정책과 NOFORCE 정책을 수행한다. FORCE 정책은 최신 페이지가 항상 디스크에 유지되도록 하기 때문에 과도한 디스크 I/O 오버헤드를 갖는다. FORCE에 비해 효율적인 NOFORCE 정책의 경우, PCA 처리 노드가 항상 최신 페이지 또는 최신 페이지를 캐싱하고 있는 처리 노드에 대한 정보를 가지도록 하는데, 이를 위한 메시지 전송량이 많다는 것이 단점이다.

마지막으로, PCL에서는 PCA를 재할당하는 작업이 용이하지 않기 때문에 시스템 초기에 일단 PCA를 할당하게 되면 새로운 처리 노드가 추가되거나 기존의 처리 노드가 고장난 경우를 제외하고는 재할당을 고려하지 않는다. 즉 PCA 할당은 공유 데이타베이스가 분할될 때 처리 노드의 갯수에 따라 정적으로 결정되므로 도중에 새로운 처리 노드가 추가되면 공유 데이타베이스 분할은 물론 PCA 할당 과정을 다시 거쳐야 한다. 이 경우의 PCA 재할당은 트랜잭션이 액세스할 데이타가 원격 데이타일 가능성을 높게 하고, 새로운 데이타베이스 분할과 트랜잭션 경로 배정 정책이 재할당 이전과 같은 참조 국부성(locality of reference)을 보장하지 못한다면 전역 로크 요청 메시지의 증가로 인해 트랜잭션의 처리율은 떨어지게 된다.

2.2 로크 보유 기법

PCA를 동적으로 할당할 수 있는 방법 중의 하나가 로크 보유 기법(Dan, 1992, Franklin, 1992)이다. 로크 보유 기법(lock retention)의 기본 개념은 트랜잭션 실행 중에 획득한 로크를 트랜잭션이 완료된 후에도 해제하지 않고 계속 유지하는 것이다. 이후 동일한 처리 노드에서 실행되는 다른 트랜잭션이 그 로크를 요청할 때 로크 요청 메시지를 전송할 필요없이 바로 로크를 부여받을 수 있고, 트랜잭션 완료시에도 로크 해제 메시지를 전송할 필요가 없다. 로크 보유 기법에는 보유되는 로크 형태에 따라 읽기 로크 보유와 쓰기 로크 보유가 있다. 읽기 로크 보유 기법은 트랜잭션 간에 읽기 로크만 보유하므로 쓰기 로크를 가진 트랜잭션은 완료시에 바로 로크를 해제한다. 쓰기 로크 보유 기법은 읽기 로크와 쓰기 로크 모두를 보유할 수 있기 때문에 쓰기 로크를 가진 트랜잭션이 완료된 후에도 해당 처리 노드는 쓰기 로크를 계속 보유하며 갱신된 페이지는 자신만 캐싱한다. 읽기 로크 보유 기법은 쓰기 로크 보유 기법에 비해 로크 보유율이 높아 메시지 전송량을 줄일 수 있는 반면 시스템 고장시 복구 작업이 그만큼 복잡하게 된다.

로크 보유 기법은 메시지 전송량을 감소시킴으로써 성능을 향상시킬 수 있다. 뿐만 아니라 각 처리 노드에서 자주 액세스되는 데이타에 대한 로크는 그 처리 노드에서 유지될 가능성이 높기 때문에 PCA가 동적으로 할당되는 효과를 갖는다.

그러나 다음과 같은 단점을 가진다.

첫째, 자주 액세스되지 않는 데이타라 할지라도 데이타를 액세스할 때마다 그 데이타에 대한 로크가 유지되기 때문에 로크 관리에 대한 많은 부담이 따른다. 즉 데이타에 대한 로크는 그 데이타를 액세스한 여러 개의 처리 노드에 의해 동시에 보유될 수 있으므로 GLM으로 새로운 전역 로크 요청이 들어왔을 때 그 로크를 보유한 처리 노드가 많을수록 처리 과정이 복잡해진다.

둘째, 비교적 작은 단위(페이지)에 대해 로크 보유 기법이 적용되므로 트랜잭션에 의해 평균적으로 많은 전역 로크 요청이 발생한다. 특히 대량의 데이타를 액세스하는 배치 트랜잭션의 경우 데이타 처리시간에 비해 로크 획득에 걸리는 시간에 대한 부담이 상당히 크다.

Ⅲ. PCA 관리 방식

본 장에서는 PCA를 정적으로 관리하는 기존의 방식과 본 연구에서 제안하는 동적 PCA 관리 방식의 효율성에 대해 비교분석한다.

3.1 정적 PCA 관리

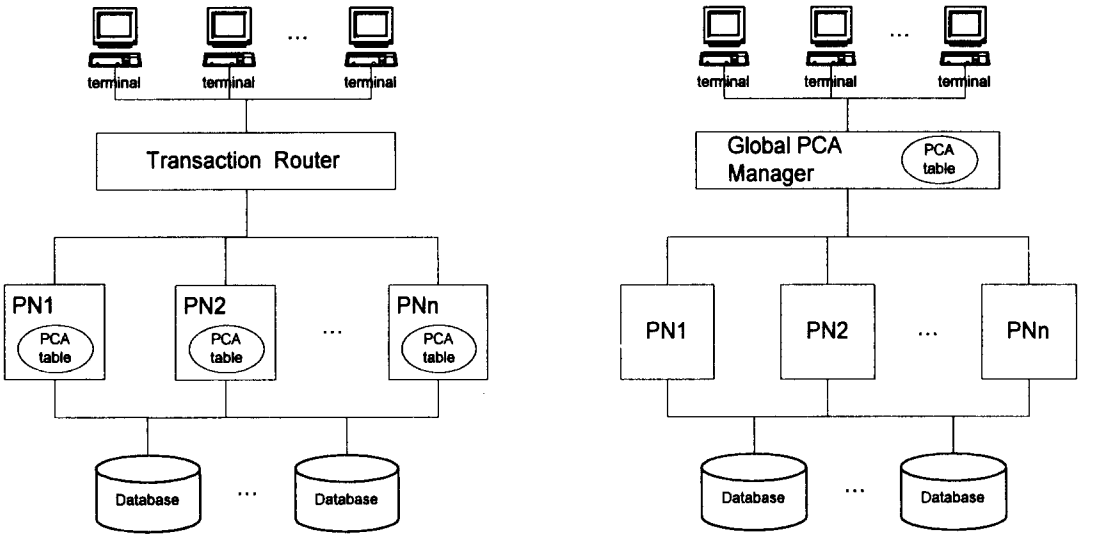
정적 PCA 관리(Static PCA Management: SPM)는 시스템 전체의 PCA 할당에 관한 정보를 카탈로그(catalog) 형식으로 구성하여 모든 처리 노드가 동일한 사본을 가지도록 한 방식이다. 시스템 초기에 카탈로그가 한번 만들어지면 시스템 구성에 대한 변동사항이 발생하지 않는 한 카탈로그 정보를 갱신하지 않는다. 왜냐하면 각 처리 노드가 완전한 카탈로그를 가지므로 임의의 노드에서 카탈로그가 갱신될 경우 시스템 전체가 영향을 받기 때문이다. 이러한 부담때문에 PCA는 일단 처리 노드에 할당되고 나면 가급적 재할당되지 않는다. 따라서 SPM은 트랜잭션 경로 배정 정책에 많은 영향을 받는다.

각 처리 노드는 트랜잭션의 로크 요청이 발생하면 카탈로그를 참조하여 해당 요청이 지역적이면 별도의 통신 오버헤드나 지연없이 바로 처리하고, 그렇지 않으면 자신의 카탈로그를 참조하여 PCA 처리 노드로 전역 로크 요청 메시지를 전송하여 처리한다.

이 방식은 기존의 PCL에서 사용된다.

3.2 동적 PCA 관리

SPM에서, PCA는 항상 일정한 노드에 의해 유지되므로 시스템이 운영되는 도중에 임의의 노드에 과다한 작업 부하(workload)가 걸리더라도 적응성있게 대처하지 못하여 성능저하를 초래한다. 또한 새로운 처리 노드가 추가되는 경우 모든 처리 노드의 카탈로그는 재조정되어야 한다. 물리적인 시스템 구성의 변동이나 처리 노드의 부하량의 변화시에 보다 융통성있고 적응성있게 대처하기 위해 PCA를 동적으로 할당하는 방식을 동적 PCA 관리(Dynamic PCA Management: DPM)라 한다. 제안된 DPM에서 카탈로그는 PCA가 재할당되거나 시스템 구성상의 변화시에 갱신된다. 그러나 PCA 할당에 관한 완전한 카탈로그는 하나의 전위 노드(front-end node)에만 존재하므로 카탈로그 갱신에 따른 부담을 줄일 수 있다. 본 논문에서는 PCA 처리 노드의 지역 버퍼에서 페이지 교체가 발생하였을 때 PCA를 동적으로 재할당한다. 또한 부분적으로 읽기 로크 보유 기법을 도입함으로써 PCA 재할당의 효과를 증대시켜 로크 관리의 부담을 줄이고, PCA를 가진 처리 노드에서 해당 페이지에 대한 버퍼 정보를 쉽게 유지관리할 수 있게 한다.



(a) 정적 PCA 관리 (b) 동적 PCA 관리

<그림 1> PCA 관리 방식에 따른 시스템 구성도

<그림 1>은 PCA 관리 방식에 따라 구성된 시스템 예이다. (a)는 SPM 방식의 PCL에서의 시스템 구성도로서, 모든 처리 노드가 PCA 카탈로그를 가지고 있다. 이미 언급하였듯이, PCL은 자체적으로 트랜잭션 경로 배정 기능을 지원하지 못하기 때문에 이를 위한 안정된 정책이 제공

된다고 기본적으로 가정하고 있다. <그림 1>의 (a)에서 트랜잭션 경로 배정기(transaction router)가 전위 노드에 제공되어 있다. (b)는 본 논문에서 동적 PCA 관리를 위해 구성한 시스템으로서, SPM에서와 달리 별도의 GPM(Global PCA Manager) 처리 노드를 두고 있다. GPM 처리 노드는 트랜잭션 경로 배정기와 PCA 테이블 관리자(PCA Table Manager: PTM)를 가진다. PTM은 시스템 내에서 유일한 PCA 카탈로그를 전적으로 유지관리하고, 각 처리 노드로부터 전역 로크 요청 메시지가 들어 오면 PCA 테이블을 참조하여 해당 PCA 처리 노드로 메시지를 전송한다.

동적으로 PCA를 관리하는 방식에서 별도의 GPM 처리 노드를 둠으로써 시스템의 신뢰성에 미치는 영향을 고려해 보면 다음과 같다. 우선 새로운 처리 노드가 추가되거나 기존의 처리 노드가 고장난 경우, GPM 처리 노드 내의 카탈로그를 참조하여 PCA를 재분배하거나 고장난 처리 노드의 PCA를 다른 처리 노드에게 재할당할 수 있으므로 정적으로 PCA를 관리하는 경우에 비해 구현이 용이하다는 장점을 갖는다. 그러나 GPM 처리 노드가 고장난 경우에는 시스템 내의 모든 처리 노드의 로킹 정보를 통합하여 PCA 테이블을 재구성하여야 하므로 정적 PCA 관리 방식에 비해 신뢰성이 다소 떨어지는데, 중앙집중형의 경우처럼 시스템이 완전히 마비되지는 않는다. 왜냐하면 각 처리 노드에서의 지역 로크 요청은 지역의 로크 테이블을 이용하여 계속 처리할 수 있기 때문이다.

Ⅳ. 버퍼 무효화 기법

데이타베이스를 공유하는 환경에서는 다수의 처리 노드가 동시에 동일한 페이지 사본을 버퍼에 캐싱할 수도 있다. 버퍼 무효화 기법은 임의의 처리 노드에서 발생한 액세스 요청에 대해 항상 최신 페이지를 제공하기 위해 필요하며, 일반적으로 FORCE와 NOFORCE 정책을 통해 실현된다(Wang, 1991).

FORCE 정책에서 최신 페이지는 항상 디스크에 유지되므로 갱신 트랜잭션이 완료될 때마다 반드시 디스크에 갱신된 페이지를 기록해야 하며, 모든 페이지 액세스도 디스크를 통해야만 가능하다. 이 정책은 단순하여 상용 DBSS에서 사용되기는 하지만 빈번한 디스크 I/O로 인해 비효율적이므로 본 연구에서는 고려하지 않는다.

NOFORCE 정책은 최신 페이지를 디스크 또는 임의의 처리 노드에 유지함으로써 FORCE에서와 같은 과도한 디스크 I/O 오버헤드를 대폭 줄일 수 있다. 다수의 처리 노드가 동시에 캐싱하고 있는 페이지에 대한 액세스가 발생하였을 때 항상 최신 페이지를 제공하기 위해서는 버퍼 무효화 기법이 보다 신중하게 고려되어야 한다.

본 장에서는 동적으로 PCA가 관리되는 환경에서 캐쉬 일관성 유지를 위해 제안한 버퍼 무효화 기법에 대해 기술한다. 이를 위해 우선 4.1에서 정적 PCA 관리 방식에서의 버퍼 무효화 기법(Rahm, 1986)에 대해 설명한다. 4.2에서 각 처리 노드의 디스크 I/O 오버헤드를 줄이기 위해 제안한 버퍼 무효화 기법에 대해 설명한다. 실제로 트랜잭션 처리 시간의 대부분을 디스크 입출력에 소요하므로 이에 따른 오버헤드를 줄이는 것은 성능향상에 많은 영향을 미친다고 볼 수 있다(Franklin et al., 1986, Mohan & Narang, 1991).

아래의 기법들은 <그림 2>와 같은 로크 테이블 구조를 사용한다. 처리 노드 내의 트랜잭션 스케쥴러는 각각 전역 로크 테이블(Global Lock Table: GLT)과 지역 로크 테이블(Local Lock Table: LLT)을 관리한다. GLT는 전역 로크 요청을 처리하기 위해 자신이 PCA를 가진 페이지에 대한 시스템 전체의 로킹 정보를 포함하고, LLT는 지역 로크 요청을 처리하기 위해 필요한 로킹 정보를 포함한다. 로킹은 페이지 단위로 이루어진다.

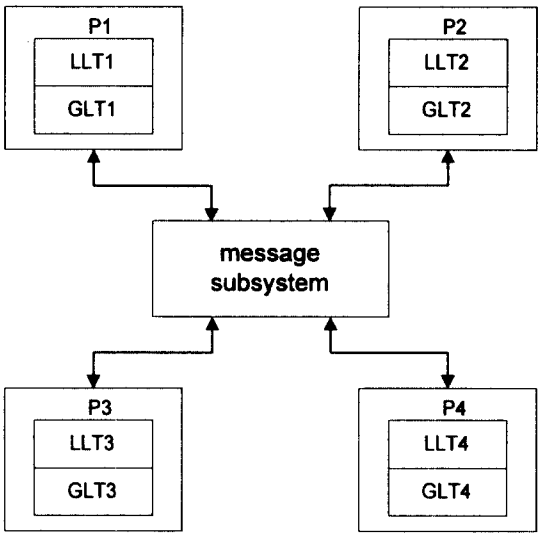
4.1 PCA 처리 노드에서 최신 페이지를 유지하는 기법

임의의 처리 노드에서 발생한 로크 요청이 지역적으로 처리될 수 없으면 PCA 처리 노드에 의해 처리되어야 한다. 이때 PCA 처리 노드는 로크를 허용가능하다면 로크 허용 메시지와 함께 로크를 요청한 처리 노드가 최신 페이지를 자신의 버퍼에 캐싱하고 있지 않으면 최신 페이지를 제공하여야 한다. PCA 처리 노드는 가장 최근에 페이지를 갱신한 처리 노드에 대한 정보를 유지함으로써 최신 페이지를 제공할 수는 있으나 여러 단계에 걸친 메시지 전송으로 인해 오버헤드가 따르고, 최근에 갱신한 처리 노드의 버퍼에 해당 페이지가 없다면 디스크로부터 읽어 들여야 하므로 디스크 액세스로 인한 오버헤드도 적지 않다.

```
struct TR_List {
 int tid;                          /* transaction no. */
 int mode;                         /* lock mode: NONE, S, X */
 int proc_id;                      /* processing node no. */
}

struct GLT {
 int pid;                          /* page no. */
 int mode;                         /* lock mode: NONE, S, X */
 int buf_inv[MAXDBMS];             /* buffer invalidation vector: 1→invalid
 struct TR_List *gbl_wait_list;                                   0→valid*/
}

struct LLT {
 int pid;
 struct TR_List *granted_list;
 struct TR_List *waited_list;
}
```



<그림 2> 로크 테이블 구조

따라서 PCA 처리 노드에서 최신 페이지를 유지하는 것이 바람직하다. 이를 위해 (Rahm, 1986)에서는 임의의 처리 노드에서 갱신 트랜잭션이 완료될 때마다 PCA 처리 노드로 갱신된 페이지를 전송한다. 그러나 로크 해제 메시지에 최신 페이지를 포함해야 하기 때문에 통신 오버헤드가 증가한다. 또한 버퍼 관리의 부담이 여전히 남아 있다. 즉, 트랜잭션 완료시에 최신 페이지를 PCA 처리 노드로 전송하기 위해서는 트랜잭션이 완료될 때까지 버퍼 내에 유지해야 하고, PCA 처리 노드로 전송된 페이지를 위해서는 버퍼 내 공간을 할당해야 하므로 더 많은 페이지 교체가 발생한다. 그리고 PCA 수도 있다. 스래싱으로 인한 높은 페이지 교체율을 줄이기 위해서는 큰 버퍼 크기가 요구된다.

4.2 PCA 처리 노드에서 페이지 교체시마다 PCA를 재할당하는 기법

이미 살펴 보았듯이, 4.1에서와 같이 PCA 처리 노드에서 최신 페이지를 유지하는 경우에는 PCA 처리 노드가 고정됨으로 인해 불필요한 메시지 전송과 디스크 입출력이 발생한다.

이 기법은 PCA 처리 노드에서 페이지 교체시에 해당 페이지에 대한 PCA를 재할당함으로써 페이지 교체로 인해 발생하는 디스크 액세스를 줄이기 위해 제안되었다. 즉, 페이지를 교체할 때 최신 페이지를 캐싱하고 있는 다른 처리 노드로 PCA를 재할당하면 교체전에 굳이 디스크에 기록할 필요가 없고, PCA는 항상 최신 페이지를 가진 처리 노드에 할당되므로 이후의 페이지 참조시에도 디스크로부터 읽어들일 필요가 없다. 물론 PCA 재할당을 위한 메시지 오버헤드가 있

지만 디스크 액세스로 인한 오버헤드보다는 크지 않다. 따라서 PCA 처리 노드에서 페이지가 교체됨으로 인한 디스크 액세스는 PCA를 재할당할 노드가 존재하지 않는 경우에만 일어나며, 이 때 PCA 재할당은 고려되지 않는다. 페이지 교체시에 PCA를 재할당하기 위해서는 PCA 처리 노드 외에 현재 어느 처리 노드가 최신 페이지를 캐싱하고 있는가에 대한 버퍼 정보를 알아야 하는데, 이것은 로크 보유 개념을 이용한다. 로크 보유는 읽기 로크에 대해 적용되므로 다수의 처리 노드에서 로크를 보유할 수 있고, 만약 로크를 보유한 처리 노드의 버퍼에 해당 페이지가 캐싱되어 있다면 최신 페이지일 것이므로 디스크 액세스나 페이지 전송없이도 PCA를 재할당할 수 있다. 로크를 보유한 처리 노드는 페이지 교체가 발생할 때 로크 보유된 페이지이면 PCA 처리 노드에 페이지가 교체되었음을 알림으로써, PCA 처리 노드로 하여금 최신 페이지를 캐싱하고 있는 다른 처리 노드로 올바르게 PCA를 재할당할 수 있도록 한다.

V. 시뮬레이션 모델

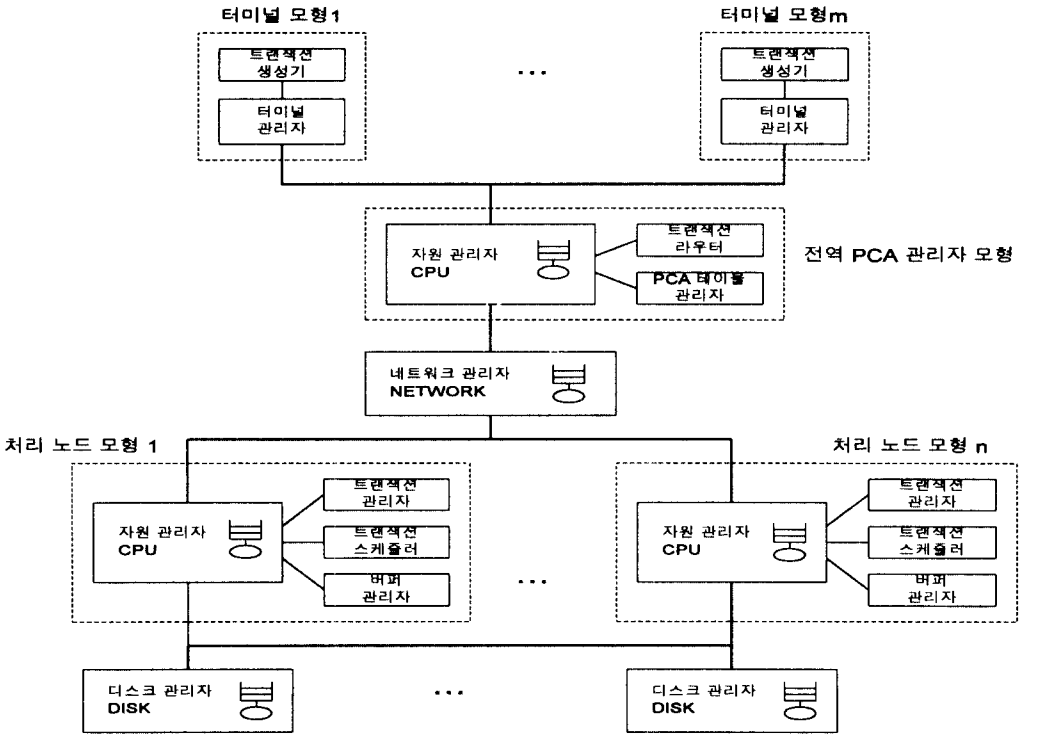
본 장에서는 제안된 기법들의 성능평가를 위한 시뮬레이션 모델에 대해 기술한다. 시뮬레이션은 미국의 MCC에서 개발한 CSIM 언어(Schweman, 1992)를 이용하여 수행되었다. 시뮬레이션을 위한 공유 데이타베이스 환경이 <그림 3>에 나타나 있다.

<그림 3>에서 전역 PCA 관리자(GPM)는 네트워크를 통해 처리 노드와 연결되어 있다. 각 처리 노드는 모든 디스크를 공유한다. 터미널의 트랜잭션 생성기는 일정 간격으로 트랜잭션을 생성하고, 터미널 관리자는 생성된 트랜잭션을 GPM으로 전송한다.

GPM의 트랜잭션 라우터는 터미널로부터 전송된 트랜잭션을 적절한 처리 노드로 배정하는데, 배정 기준은 트랜잭션이 액세스하는 페이지에 대해 가장 많은 PCA를 가진 처리 노드를 선택하는 것이다. PCA 테이블 관리자는 PCA 테이블을 토대로 처리 노드로부터 들어온 전역 로크 요청을 처리하고, PCA 재할당시에 PCA 테이블을 조정한다.

각 처리 노드는 별도의 지역 버퍼를 가지며, 버퍼 관리자는 LRU 정책을 바탕으로 자신의 버퍼를 관리한다. 또한 자원 관리자는 자신에게 할당된 CPU 작업을 모델링하며 네트워크를 통한 GPM과의 통신 및 처리 노드 간의 페이지 전송과정을 수행한다. 트랜잭션 관리자는 로크 요청 및 페이지 액세스 요청 등 트랜잭션을 실질적으로 실행한다. 트랜잭션 스케줄러는 페이지 단위의 2 단계 로킹 기법을 지원하며, 교착 상태를 해결하기 위해 대기 그래프에 바탕을 둔 교착 상태 탐지 및 해결 기법을 지원한다.

본 논문에서 사용한 입력 매개 변수는 <표 1>과 같다. 각 매개 변수의 구체적인 값은 (Carey et al., 1994)에서 많이 참조하였다.



<그림 3> 시뮬레이션 모델 구성도

<표 1> 입력 매개 변수

LCPUSpeed	처리 노드의 CPU 속도	10 MIPS
GPMCPUSpeed	GPM의 CPU 속도	50 MIPS
NetBandWidth	네크워크의 데이터 전송 속도	10Mbps
NumDBMS	처리 노드의 수	3-50
NumDisk	공유 디스크의 수	4 disks
MinDiskTime	최소 디스크 액세스 시간	10 milliseconds
MaxDiskTime	최대 디스크 액세스 시간	30 milliseconds
PageSize	각 페이지의 크기	4096 bytes
DatabaseSize	DB에 저장된 페이지의 개수	5000
BufSize	처리 노드의 버퍼 크기	20 % of DB size

(a) 시스템 구성 변수

FixedMsgInst	메시지 처리를 위한 고정 명령수	20,000
PerByteMsgInst	메시지 길이당 추가되는 명령수	10,000 per page
ControlMsgSize	제어 메시지의 길이	256 bytes
LockInst	로크 등록/해제를 위한 명령수	300
PerIOInst	디스크 I/O를 위한 명령수	5000

(b) 오버헤드 변수

CreationDelay	트랜잭션 생성시 평균 대기시간	1 second
TranSize	트랜잭션당 평균 페이지 액세스 수	50 records
TRSizeDev	트랜잭션 길이의 편차	0.1
WriteOpPct	갱신연산의 비율	0.05 - 0.5

(c) 트랜잭션 변수

전체 시스템의 병목 현상을 방지하기 위해 GPM이 사용하는 CPU는 다른 처리 노드에서 사용하는 CPU보다 성능이 더 우수하다고 가정하였다. 디스크 수는 4로 가정하였으며, 디스크 액세스 시간은 0.01초에서 0.03초까지의 일양 분포(uniform distribution)를 따른다. 디스크와 CPU는 FIFO 큐를 이용하여 I/O 요청 및 로크 요청 등을 들어온 순서대로 처리한다.

네트워크 관리자는 10Mbps의 대역폭을 갖는 FIFIO 서버로 구현되었다. 네트워크를 통해 메시지를 전송하는 과정을 표현하기 위해 처리 노드의 CPU 및 GPM의 CPU는 메시지마다 FixedMsgInst만큼의 고정된 명령수와 PerByteMsgInst + ControlMsgSize만큼의 추가적인 명령수를 실행한다.

트랜잭션 생성시간은 평균값을 CreationDelay로 하는 지수 분포를 따른다. 각 트랜잭션이 액세스하는 페이지 수는 TranSize ± TranSize × TRSizeDev 사이의 일양 분포를 따른다. 액세스하는 페이지들에 대해 갱신할 확률은 WriteOpPct이다.

시뮬레이션에서 사용되는 주요 성능 지수는 단위 시간당 트랜잭션 처리량(throughput)과 트랜잭션당 평균 응답 시간(response time)이다. 트랜잭션 처리량은 초당 완료되는 트랜잭션 수를 의미하며, 트랜잭션의 응답 시간은 트랜잭션이 생성된 후 완료될 때까지의 시간을 의미한다. 응답 시간에는 큐에서의 대기 시간 및 트랜잭션이 철회되어 재실행되는 시간도 모두 포함된다.

본 논문에서 구현된 실험 모델은 폐쇄 큐잉 시스템이므로 트랜잭션 처리량과 응답 시간은 동일한 결과를 나타낸다. 즉, 트랜잭션 처리량이 높은 버퍼 무효화 기법은 응답 시간이 빠르다. 그러므로 본 논문에서는 성능 지수로서 응답 시간만을 표현한다.

신뢰성있는 시뮬레이션 결과를 얻기 위해 배치 평균 기법(batch mean method)을 이용하였다. 본 논문에서 나타난 실험 결과들은 30개의 다른 seed를 이용하여 산출된 결과들의 평균값이다. 각 기법들에 대한 실험은 완료된 트랜잭션 수가 2,000개가 될 때까지 수행하며, 초기 200개가 완료될 때까지의 결과들은 무시하였다. 이러한 기법을 이용하여 산출된 결과들은 90 퍼센트의 신뢰 수준을 만족하였다.

VI. 결과분석 및 검토

본 절에서는 5절에서 구성한 시뮬레이션 모델을 이용하여 버퍼 무효화 기법들에 대해 실험 결과를 분석하고 성능을 평가한다. 구현된 기법들은 PCA 처리 노드에서 최신 페이지를 유지하는 기법(SD1)과 PCA 처리 노드에서 페이지 교체시마다 PCA를 재할당하는 기법(SD2)이다. 제안된 버퍼 무효화 기법은 PCA 처리 노드에서의 지역 로크 처리율을 다양하게 주면서 실험하였다.

각 기법에서 최신 페이지는 PCA 처리 노드의 지역 버퍼 내에 캐싱되어 있거나 디스크에 저장되어 있다. PCA 처리 노드는 요구된 페이지가 버퍼 내에 없을 경우 디스크로부터 읽어 들이라는 메시지를 전송한다.

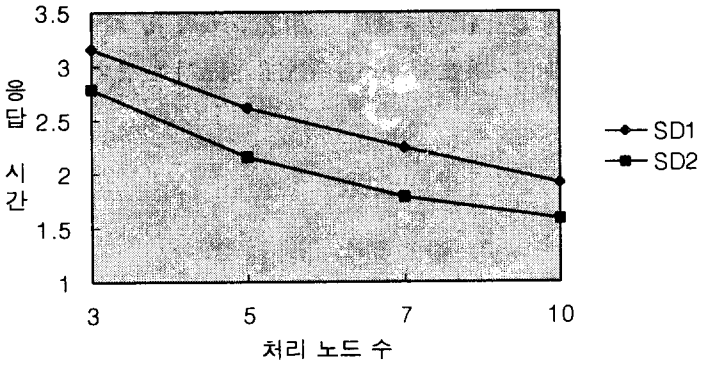
응답 시간이 대략적으로 로크 요청을 처리함에 따른 지연시간인 block_time, 페이지 전송에 따른 부담시간인 comm_time, 그리고 디스크 입출력에 따른 disk_time으로 구성된다고 보았을 때, disk_time이 응답 시간에 미치는 영향이 가장 크므로 디스크 액세스 횟수를 중심으로 비교한다. 그림 4와 그림 5에서 보듯이 지역 로크 처리율에 따라 버퍼 무효화 기법들의 응답 시간을 비교한 그래프는 디스크 액세스 횟수를 비교한 그래프와 거의 일치하였다. 디스크 액세스는 요청된 페이지를 캐싱하기 위해 버퍼 내의 기존 페이지를 교체할 때와 PCA 처리 노드에서 최신 페이지를 캐싱하고 있지 않은 상황에서 액세스 요청이 들어 왔을 때 발생한다. 응답 시간의 단위는 초(second)이다.

그래프들은 터미널 수를 100, 디스크 수를 4, 갱신 연산 비율을 20%로 하였을 때의 결과들이다.

6.1 낮은 지역 로크 처리율

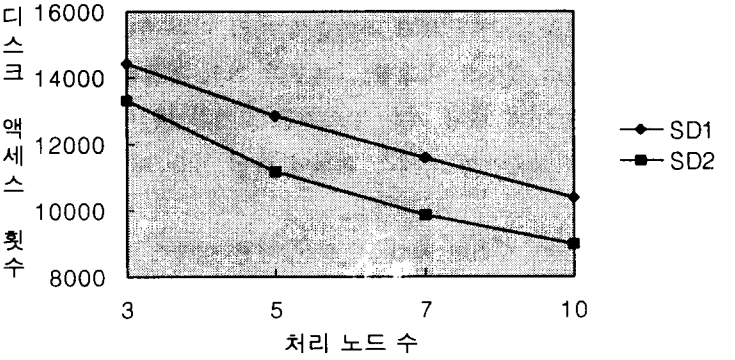
이 환경에서는 트랜잭션이 실행하는 연산의 20%만 지역적으로 처리되고, 나머지 80%는 전역적으로 처리되도록 한 환경으로서, 트랜잭션이 공유 데이타베이스를 무작위로 액세스하는 경우에 해당한다. 따라서 전역 로크 처리가 많음에 따라 PCA 처리 노드로의 페이지 액세스 요청

이 빈번히 발생하고, 각 처리 노드의 지역 버퍼의 페이지 교체율도 증가한다.



<그림 4> 지역 로크 처리율리 20%일 때의 응답 시간

<그림 4>에서 보면 처리 노드의 수가 증가할수록 응답 시간은 감소한다. 이유는 두가지로 생각할 수 있다. 첫째, 시스템 전체의 버퍼 크기가 증가하므로 페이지 교체율이 적고, 액세스하고자 하는 페이지를 캐싱하고 있을 확률이 높다. 둘째, 처리 노드 수만큼 공유 데이타베이스를 논리적으로 분할하여 처리 노드에게 페이지에 대한 PCA를 할당하므로 처리 노드 수가 많을수록 각 처리 노드에게 할당되는 PCA 수가 적다. 따라서 PCA 처리 노드의 버퍼에서 최신 페이지를 캐싱하고 있을 확률이 높기 때문에 디스크 액세스를 줄일 수 있다.



<그림 5> 지역 로크 처리율리 20%일 때의 디스크 액세스 횟수

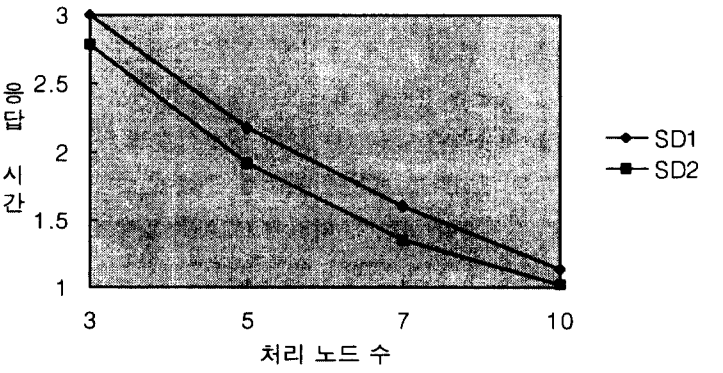
PCA를 정적으로 유지하는 SD1에 비해 SD2가 약 17% 정도 성능이 우수하다. SD1과 SD2의 경우 PCA 처리 노드에서 최신 페이지를 유지하기 위해 갱신 트랜잭션이 완료될 때마다 갱신된 페이지를 PCA 처리 노드로 전송해야 하는데, 이로 인해 최신 페이지 전송에 따른 메시지 오버헤드가 있고 또 PCA 처리 노드에서의 잦은 페이지 교체가 따른다.

그러나 SD2에서는 교체될 페이지가 자신이 PCA를 가진 최신 페이지라면 디스크에 기록하지 않고 해당 시점에서 교체될 페이지를 캐싱하고 있는 다른 처리 노드로 PCA를 재할당하므로 페이지 교체가 빈번히 발생한다고 하더라도 그로 인한 디스크 액세스는 많지 않다. 또한 자신이 PCA를 가진 페이지를 버퍼 내에 캐싱하고 있을 확률이 높다. PCA 처리 노드의 버퍼에 캐싱하고 있지 않은 페이지에 대한 액세스 요청이 들어오면 디스크로부터 읽으라는 메시지만 전송하므로 캐싱하고 있는 경우에 비해 디스크 액세스는 빈번히 발생한다. 따라서 정적으로 PCA를 유지하는 SD1에 비해 페이지 교체로 인한 디스크 액세스 횟수가 적으므로 평균 응답 시간이 빠르다.

6.2 높은 지역 로크 처리율

이 환경은 트랜잭션이 액세스하는 대부분의 페이지가 하나의 처리 노드에서 PCA를 가지는 경우, 즉 참조 지역성이 존재하는 경우이다. 트랜잭션이 실행하는 연산의 60%가 한 처리 노드에서 지역적으로 처리되고, 나머지 40%에 대해서는 다른 처리 노드에 의해 전역적으로 처리되도록 하였다.

지역 로크 처리율이 높아지면 로크 요청을 처리하기 위한 시간과 페이지 전송에 따른 메시지 오버헤드가 감소한다. 또한 참조 지역성으로 인해 페이지 교체가 일어날 확률이 적고, 최신 페이지가 PCA 처리 노드의 버퍼에 캐싱되어 있을 확률이 높아진다. <그림 6>에서 보듯이 지역 로크 처리율이 낮을 때에 비해 버퍼 무효화 기법들의 성능이 모두 향상되었다.

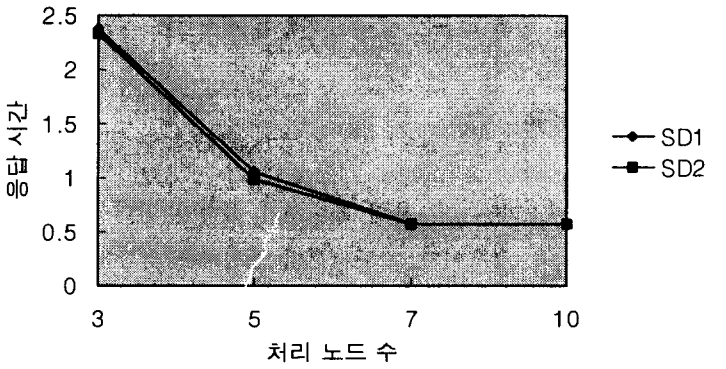


<그림 6> 지역 로크 처리율리 60%일 때의 응답 시간

그런데, SD1에 비해 SD2의 성능은 처리 노드 수가 많아짐에 따라 비교적 완만하게 증가한다. 그 이유는 SD2가 페이지 교체로 인한 영향을 적게 받으면서 PCA 처리 노드의 버퍼에 최신 페이지를 캐싱하고 있을 확률을 높이기 위한 기법이기 때문에 페이지 교체율이 클수록 효과가 크다고 볼 수 있는데, 지역 로크 처리율이 높아지면 각 처리 노드에서 액세스하는 데이타베이스

의 범위가 좁아져 페이지 교체율이 적다.

<그림 7>은 참조 지역성이 아주 높은 경우의 실험 결과이다. 즉, 트랜잭션이 실행하는 연산의 80%가 지역적으로 처리되고, 나머지 20%만 전역적으로 처리된다. 대부분의 연산이 지역적으로 처리되므로 처리 노드 간의 통신 오버헤드는 물론 페이지 교체율이 상당히 낮기 때문에 두 기법의 성능 차이는 거의 없다.



<그림 7> 지역 로크 처리율이 80%일 때의 응답 시간

Ⅶ. 결론 및 앞으로의 연구 방향

DBSS에서 데이타는 다수의 DBMS에 의해 동시에 캐싱되어질 수 있다. 데이타베이스 공유 시스템의 성능은 공유 데이타의 일관성 유지를 위한 동시성 제어 기법과 일관성 제어 기법에 의해 많은 영향을 받는다. PCA를 이용하여 공유 데이타의 일관성과 동시성 제어를 수행하는 기존의 PCL 방식은 PCA를 정적으로 관리하기 때문에 시스템 구성이 변화될 때 부담이 크고, 작업 부하량의 변동시에 능동적으로 대처하지 못한다. 따라서 PCA를 동적으로 관리하여 시스템 환경의 변화에 대해 적응성있게 대처할 수 있는 기법이 필요하다.

본 논문에서는 PCA를 동적으로 관리하는 버퍼 무효화 기법(SD2)을 제안하고, 시뮬레이션을 통해 정적으로 PCA를 관리하는 SD1과 성능 비교를 하였다. SD1은 PCA 처리 노드가 최신 페이지를 제공하기 위해 갱신 트랜잭션이 완료될 때마다 갱신된 페이지를 PCA 처리 노드로 전송하는데, 이로 인해 디스크 액세스가 요구되는 페이지 교체가 발생하고, 페이지 전송에 따른 오버헤드가 따른다. SD2는 PCA 처리 노드에서 페이지 교체가 발생할 때 교체될 페이지가 최신의 것이라면 해당 페이지를 캐싱하고 있는 다른 처리 노드로 PCA를 재할당하여 최신 페이지가 가급적 PCA 처리 노드의 버퍼에 캐싱되도록 함으로써 디스크 액세스 빈도를 줄일 수 있다.

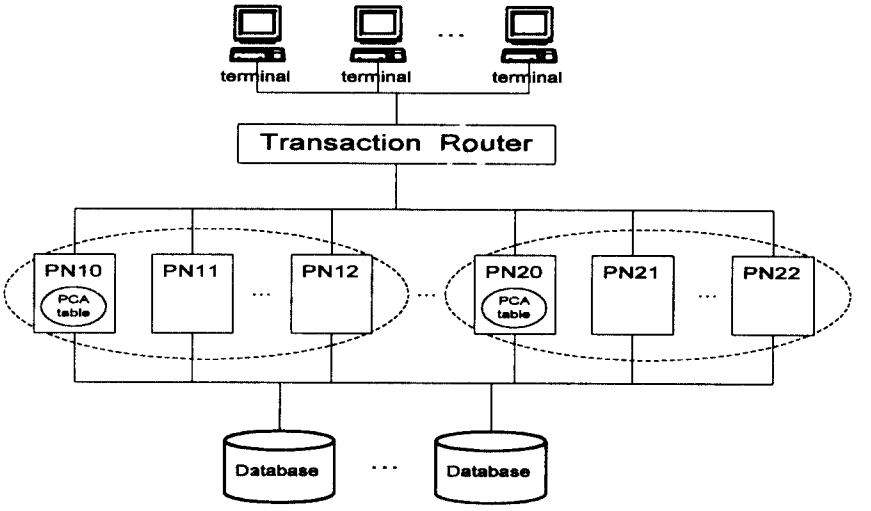
제안된 기법은 DSS 환경이 지원되는 시뮬레이션 모델을 구성하여 지역 로크 처리율을 변화시켜 가면서 성능 평가를 하였다. 그 결과로 지역 로크 처리율이 낮은 경우 SD1에 비해 SD2가 17%의 성능이 향상되었음을 알 수 있었다. 지역 로크 처리율이 높아질수록 SD1의 성능은 큰 폭으로 증가하여 지역 로크 처리율이 80%일 때 두가지 기법의 성능은 거의 비슷하였다. 그러나 (Reuter, 1986)에 의해 제기된 바와 같이, 일반적으로 트랜잭션 경로 배정 정책이 수행된다고 하더라도 아주 이상적인 경우를 제외하고는 트랜잭션의 지역 로크 처리율은 30% 정도이므로 낮은 지역 로크 처리율에서의 성능이 더 큰 의의를 가진다고 볼 수 있다.

향후 혼성 PCA 관리 방식에 대해 연구하고자 한다.

본 논문에서 고려하고 있는 시스템은 동적 PCA 관리를 전담하는 GPM이라는 별도의 처리 노드를 두고 있다. 이미 언급하였듯이, DPM은 GPM 처리 노드의 고장에 따르는 위험 부담이 따르기 때문에 SPM이 제공하는 PCA 관리의 안정성과 DPM이 제공하는 PCA 관리의 융통성을 혼합한 혼성 PCA 관리(Hybrid PCA Management: HPM)을 제안한다.

그림 8에서 보듯이 HPM에서 DBSS는 다수의 처리 노드를 가진 서브시스템들로 구성된다. 각 서브시스템에는 PCA 카탈로그를 가진 하나의 대표 처리 노드와 하나이상의 종속 처리 노드가 존재한다. PCA 카탈로그는 대표 처리 노드의 수만큼 논리적으로 분할되어 각 대표 처리 노드에게 할당된다.

HPM은 PCA 관리를 위해 서브시스템 내에서는 DPM을 따르고, 서브시스템 간에는 SPM을 따른다. 따라서 DPM에 비해 PCA 테이블 관리의 책임을 분산시킬 수 있고, SPM에 비해 시스템 환경 변화에 보다 적응성있게 대처할 수 있다.



<그림 8> 혼성 PCA 관리

참 고 문 헌

1. Anon et al., "A measure of transaction processing power", *Datamation*, April 1985, pp. 112-118.
2. A. Borr, "Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing", In *Proc. 7th Intl. Conf. VLDB*, 1981, pp. 155-165.
3. Y. Breibart, H. Garcia-Molina and A. Silberschatz, "Overview of Multidatabase Transaction Management", *VLDB*, Vol. 1, No. 2, 1992.
4. M. Carey, M. Franklin, and M. Zaharioudakis, "Fine-Grained Sharing in a Page Server DBMS", *Proc. ACM SIGMOD*, 1994. pp. 359-370.
5. S. Ceri et al., *Distributed Databases: Principles and Systems*, McGraw-Hill, 1984.
6. Customer Information Control System(CICS), "Gerneral Information", *IBM Manual GC33-1055-3*, 1987.
7. A. Dan and P. Yu, "Performance Analysis of Coherency Control Policies through Lock Retention", *Proc. ACM SIGMOD*, 1992, pp. 114-123.
8. W. Du, A. Elmagarmid, Y. Leu and S. Osterman, "Effects of Autonomy on maintaining Global Serializability in Heterogeneous Distributed Database Systems", *Proc. Data Knowledge System for Manufacturing and Engineering*, 1989.
9. M. Franklin and M. Carey, "Client-Server Caching Revisited", *Computer Science Tech. Rep. #1089,* Univ. Wisconsin-Madison, May 1992.
10. J. Gray et al., "One Thousand Transactions Per Second", In *proc. IEEE Spring CompCon*, San Francisco, 1985, pp. 96-101.
11. J. Gray, *The Benchmark Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann, 1991.
12. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Pub., 1993, pp. 406-419.
13. H. Korth and A. Silberschatz, *Database System Concepts*, 2nd Ed., McGraw-Hill, 1991, pp. 353-357.
14. N. Kronenberg, M. Levy, and D. Strecker, "VAX clusters: A Closely Coupled Distributed Syetem", *ACM Trans. on Computer Systems*, Vol. 4, No. 2, 1986, pp. 130-146.
15. E. Rahm, "A Framework for Workload Allocation in Distributed Transaction Systems", *J. System Software*, Vol. 18, No. 3, 1992, pp. 171-190.

16. E. Rahm, "Algorithms for Efficient Load Control in Multiprocessor Database Systems", *Angewandte Informatik* 28(4), 1986, 161-169.
17. E. Rahm, "Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems", *ACM Trans. on Database Systems*, Vol. 18, No. 2, 1993, pp. 333-337.
18. E. Rahm, "Primary Copy Synchronization for DB-Sharing", *Information Systems*, Vol. 11, No. 4, 1986, pp. 275-286.
19. A. Reuter, "Load Control and Load Balancing in a Shared Database Management System", *Proc. 2nd IEEE International Conf. on Data Eng.*, 1986, pp. 188-197.
20. A. Reuter and K. Shoens, "Synchronization in a Data Sharing Environment", *IBM San Jose Research Lab., Tech. Rep.(preliminary version)*, 1984.
21. H. Schwetman, *CSIM Users Guide for use with CSIM Revision 16*, MCC, 1992.
22. K. Shoens et al., "The AMOEBA Project", *Proc. IEEE CompCon*, 1985, pp. 102-105.
23. P. Strickland et al., "IMS/VS: An Evolving System", *IBM Systems Journal*, Vol. 21, No. 4, 1982, pp. 490-510.
24. Y. Wang and L. Rowe, "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture", *Proc. ACM SIGMOD Conf.*, 1991, pp. 367-376.

<Abstract>

Buffer Invalidation Schemes for High Performance Transaction Processing in Shared Database Environment

Shin Hee Kim · Jeong Mi Bae · Byeong Ug Kang

Database sharing system(DBSS) refers to a system for high performance transaction processing. In DBSS, the processing nodes are locally coupled via a high speed network and share a common database at the disk level. Each node has a local memory, a separate copy of operating system, and a DBMS. To reduce the number of disk accesses, the node caches database pages in its local memory buffer. However, since multiple nodes may be simultaneously cached a page, cache consistency must be ensured so that every node can always access the latest version of pages.

In this paper, we propose efficient buffer invalidation schemes in DBSS, where the database is logically partitioned using primary copy authority to reduce locking overhead. The proposed schemes can improve performance by reducing the disk access overhead and the message overhead due to maintaining cache consistency. Furthermore, they can show good performance when database workloads are varied dynamically.