공유 데이타베이스 환경에서 고성능 트랜잭션 처리

김신희
대경전문대학 컴퓨터정보과
배정미
대경전문대학 컴퓨터정보과

요약

데이타베이스 공유 시스템(database sharing system)은 고성능의 트랜잭션 처리를 위한 시스템으로서, 별도의 메모리와 운영체제 그리고 데이타베이스 관리 시스템(DBMS)을 가진 처리 노드와 고속의 통신 시스템으로 구성된다. 공유 데이타는 다수의 DBMS에 의해 동시에 캐싱될 수 있으므로 처리노드 간의 일관성 유지를 위해 동시성 제어(concurrency control)와 일관성 제어(coherency control) 기법이 필요하다.

본 연구에서는 동적 PCA 관리 방식과 이를 위한 버퍼 무효화 기법을 제안한다. 제안된 동적 PCA 관리 방식은 시스템 환경의 변화에 대한 적응성을 제공한다. 또한 공유 데이타의 일관성 유지를 위한 세가지의 버퍼 무효화 기법은 디스크 I/O 오버헤드와 메시지 전송량을 감소함으로써 동적 PCA 관리를 효율적으로 지원할 수 있다.

1. 서론

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.

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 처리가능하고(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 처리기가 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 처리기가 추가됨에 따라 증가한다. 따라서 처리기를 추가하여도 온라인 트랜잭션의 응답시간(response time)은 영향을 받지 않아야 한다.

마지막으로, 유지관리가 쉬워야 한다. 시

스템은 단말 사용자 또는 프로그래머에게 고수준의 인터페이스를 제공함으로써 다수의 처리기로 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 처리기는 별도의 메모리와 운영체제 그리고 DBMS를 가지며, 메시지를 이용하여 처리기 간에 통신을 한다. 따라서 처리기 간의 독립성을 최대한 보장함으로써 서로의 간섭을 줄이고, 최대의 가용성을 제공할 수 있다. 그러나 처리기 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 하나의 메모리를 공유하는 시스템(Shared Memory System: SMS)과 데이타베이스 분할 시스템(DataBase Partitioning System: DBPS) 그리고 데이타베이스 공유 시스템(DataBase Sharing System: DBSS) 등이 있다. SMS는 밀결합된 다중 처리노드 상에서 수행되는 트랜잭션 처리 시스템으로서, 주기억장치와 주변장치를 공유하며 소프트웨어의 변화가 거의 없는 중앙집중형 트랜잭션 처리 시스템에 적합하다. DBPS은 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 처리기들은 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. 뿐만아니라 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 처리노드의 DBMS가 데이타베이

스를 액세스하는 데 영향을 주지 않고, 고장난 처리기에서 수행중이던 트랜잭션은 철회(roll-back)되어 자동적으로 다른 이용가능한 처리기에 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 처리기는 전체 데이타베이스를 액세스할 수 있고, 자신의 지역 데이타베이스 버퍼를 가지기 때문에 임의의 처리기에서 어떤 데이타를 갱신하게 되면 해당 데이타를 캐싱하고 있는 다른 모든 처리기에도 갱신이 반영되어야만 일관성이 유지된다.

세째, 이용가능한 처리기들에 현재의 작업부하(workload)를 효과적으로 분배하기 위한 부하 제어(load control)기법이 필요하다. 각 처리기의 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 처리기에 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 처리기가 고장나더라도 트랜잭션 처리를 계속하기 위해 전역 저널을 이용하여 고장난 처리기를 복구하여 재통합할 수 있어야 한다. 고장난 처리기의 완료되지 못한 트랜잭션은 철회되어 다른 처리기에서 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 PCA를 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 적용하여

도 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).

지금까지 분산 트랜잭션 처리 시스템(distributed transaction processing system)에서 트랜잭션 할당에 대한 연구는 대부분 균형적 부하 분배에 집중되었다. 그러나 실제로 트랜잭션의 응답시간은 CPU 대기시간이나 페이지 교체로 인한 지연시간과 같은 시스템에 관련된 요소보다는 I/O 지연, 로크 대기, 원격 데이타 요청과 같은 DB 관련 요소에 의해 결정된다. 따라서 고성능의 트랜잭션 처리는 단순히 시스템 관련 요소만을 고려한 균형적 부하 분배 기법보다는 유사성에 근거한 트랜잭션 경로 배정(Affinity-based Transaction Routing: ATR) 기법에 영향을 받는다고 볼 수 있다(Rahm, 1992). ATR은 유사성을 갖는 트랜잭션들에 대해 같은 처리 노드에서 같은 데이타베이스 부분을 참조할 수 있도록 처리 노드에 배정한다. 이것은 처리 노드 간의 통신 횟수를 줄임으로써 트랜잭션의 처리율과 응답시간을 개선할 수 있게 한다. 그러나 유사성을 갖는 트랜잭션들이 대량으로 발생할 경우 특정 처리 노드에 부하가 집중될 수 있다. 따라서 ATR 외에 부하 균형이 부수적으로 고려되어야 한다.

본 논문에서는 ATR과 부하균형을 동시에 지원할 수 있는 동적 PCA 관리 기법을 제안한다. 또한 동적 PCA 관리 방식에서 보다 효율적으로 공유 데이타의 일관성을 유지할 수 있는 버퍼 무효화 기법(buffer invalidation)을 제안한다. 제안된 기법은 시스템 환경의 변화에 대한 적응성을 제공하고, 부하 불균형 문제를 해결함으로써 고성능의 트랜잭션 처리를 가능하게 한다.

논문의 구성은 다음과 같다. 2장에서 관련 연구로서 주사본 로킹 기법과 로크 보유 기법에 대해 기술한다. 3장에서 기존의 PCA 관리 방식과 본 논문에서 제안한 동적 PCA 관리 방식에 대해 비교분석하고, 4장에서는 제안된 캐쉬 일관성 기법에 대해 기술한다. 마지막으로 5장에서 결론 및 앞으로의 연구 방향을 기술한다.

2. 관련연구

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 처리 노드에 정적으로 할당되므로 해당 분할에 대한 모든 로크 요청은 처리 노드의 현재 부하 상태와 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 PCA의 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.

3. PCA 관리 방식

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 ATR의 효율에 많은 영향을 받는다.

각 처리 노드는 트랜잭션의 로크 요청이 발생하면 카탈로그를 참조하여 해당 요청이 지

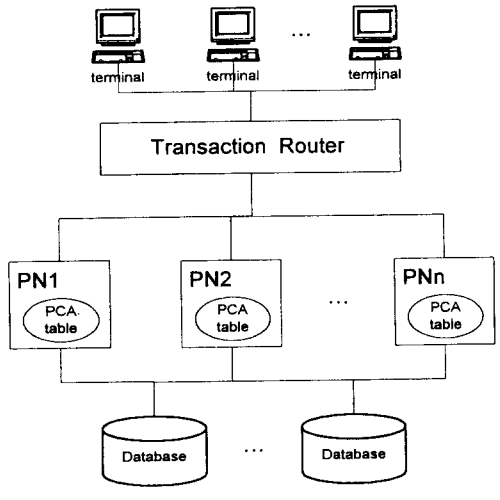
역적이면 별도의 통신 오버헤드나 지연없이 바로 처리하고, 그렇지 않으면 자신의 카탈로그를 참조하여 PCA 처리 노드로 전역 로크 요청 메시지를 전송하여 처리한다.

이 방식은 기존의 PCL에서 사용된다.

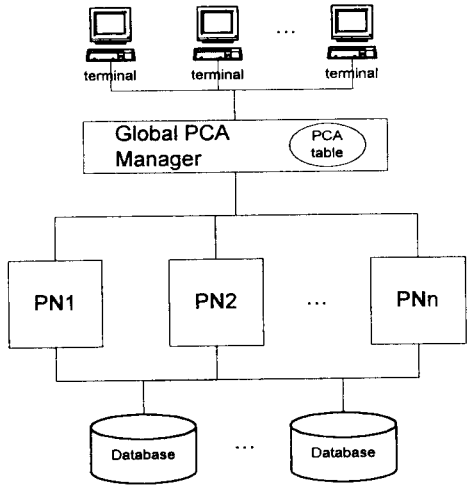
3.2. 동적 PCA 관리

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 노드로 갱신 요청이 발생하였거나 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, PCA 처리 노드에서 해당 페이지에 대한 버퍼 정보를 쉽게 유지관리할 수 있게 한다.

그림 1은 PCA 관리 방식에 따라 구성된 시스템 예이다. (a)는 SPM을 따르는 PCL에서의 시스템 구성도로서, 모든 처리 노드(processing node: PN)가 PCA 카탈로그를 가지고 있다. 이미 언급하였듯이, PCL은 자체적으로 트랜잭션 경로 배정 기능을 지원하지 못하므로 이를 위한 안정된 정책이 제공된다고 기본적으로 가정한다. 그림 1의 (a)에서 트랜잭션 경로 배정기(Transaction Router: TR)가 전위 노드에 제공되어 있다. (b)는 본 논문에서 동적 PCA 관리를 위해 구성한 시스템으로서, PCL에서와 달리 별도의 GPM(Global PCA Manager) 처리 노드를 두고 있다. GPM 처리 노드는 TR와 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.



(a) 정적 PCA 관리



(b) 동적 PCA 관리

그림 1. PCA 관리 방식에 따른 시스템 구성도

PCL에 비해 별도의 GPM 처리 노드를 둠으로써 시스템의 신뢰성에 미치는 영향을 고려해 보면, 우선 GPM 처리 노드가 고장난 경우에는 시스템 내의 모든 처리 노드의 지역 로크 테이블을 통합하여 카탈로그를 재구성하여야 하므로 신뢰성이 다소 떨어지지만, 중앙집중형의 경우처럼 시스템이 완전히 마비되지는 않는다. 왜냐하면 각 처리 노드에서의 지역 로크 요청은 지역의 로크 테이블을 이용하여 계속 처리할 수 있기 때문이다. 나머지 처리 노드가 추가되거나 고장난 경우에는 GPM 처리 노드 내의 카탈로그를 참조하여 고장난 처리 노드의 PCA를 재할당할 수 있으므로 PCL에 비해 신뢰성이 우수하다고 볼 수 있다.

4. 캐쉬 일관성 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

```
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;
}
```

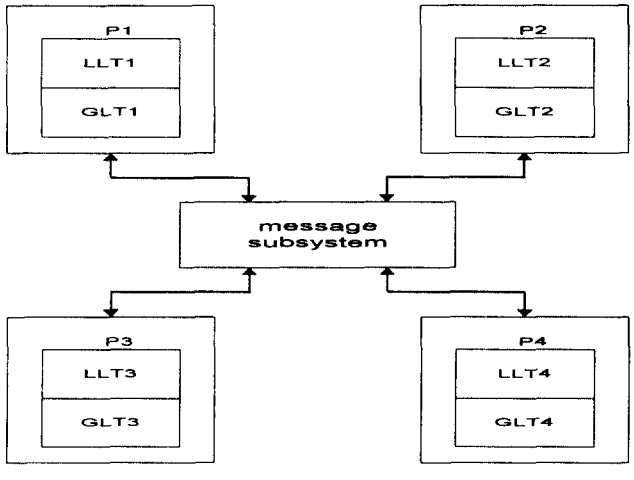


그림 2. 로크 테이블 구조

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 환경 하에서 버퍼 무효화를 통한 캐쉬 일관성 유지를 위해 제안한 세가지 기법에 대해 기술한다. 각 기법들은 디스크 I/O 오버헤드와 처리 노드 간의 메시지 전송량을 줄이기 위해 제안되었다. 실제로 디스크 I/O 시간과 메시지 전송 시간이 트랜잭션 처리 시간의 대부분을 차지하기 때문에 각 기법의 성능비교를 위한 기준으로 사용하였다.

아래의 기법들은 그림 2와 같은 로크 테이블 구조를 사용한다. 전역 로크 테이블(Global Lock Table: GLT)은 PCA 처리 노드에서 전역 로크 처리를 위해 필요하고, 지역 로크 테이블(Local Lock Table: LLT)은 지역 로크 처리를 위해 필요하다.

4.1. PCA 처리 노드에서 최신 페이지를 유지하는 기법

임의의 처리 노드에서 로크 요청이 발생하면 일단 PCA 처리 노드로 로크 요청 메시지를 보내야 한다. PCA 처리 노드는 로크 요청이 들어 오면 로크를 바로 부여할 수 있는지 판단하여 가능하다면 해당 처리 노드로 최신 페이지를 전송해야 한다. 이때 PCA 처리 노드의 버퍼에 최신 페이지가 없다면 최신 페이지를 가진 다른 처리 노드를 찾아보고, 어떤 처리 노드에도 해당 페이지를 캐싱하고 있지 않으면 디스크로부터 읽어 들이게 한다. 따라서 최신 페이지를 가진 처리 노드를 찾는데 필요한 메시지를 줄이기 위해서는 PCA 처리 노드가 항상 최신 페이지를 캐싱하고 있도록 하면 된다.

PCA 처리 노드에서 최신 페이지를 유지하기 위해서는 갱신 트랜잭션이 완료될 때마다 PCA 처리 노드로 갱신된 페이지를 전송하여야 한다. 또한 PCA 처리 노드의 버퍼가 가득찼을 경우 갱신 트랜잭션이 완료될 때마다 전송되는 페이지를 유지하기 위해 페이지 교체가 발생하며, 스래싱(thrashing)으로 인한 성능저하를 초래할 수도 있다.

디스크 I/O는 PCA 처리 노드의 버퍼에 캐싱되어 있지 않은 페이지에 대한 로크 요청이 들어 왔을 때 발생한다.

임의의 처리 노드가 PCA 처리 노드로 로크 요청을 할 때 자신의 page_LSN 정보를 로크 요청 메시지에 포함시킨다. PCA 처리 노드로부터의 로크 응답 메시지에는 로크 요청한 처리 노드가 최신 페이지를 가지는지, PCA 처리 노드로부터 최신 페이지가 전송되는지, 디스크로부터 읽어 들여야 하는지에 대한 정보가 들어있다. 이를 통해 페이지 전송에 따른 메시지 길이를 줄일 수 있다.

처리 과정은 다음과 같다.

1. 처리 노드 PN_i에서 트랜잭션 T가 로크를 요청하면 로크 관리자 LM_i는 자신의 GLT와 LLT에 해당 페이지 블록이 있는지를 조사한다. 있으면 x-possible이 0인 경우 지역적으로 처리되고, 그렇지 않으면 GPM 처리 노드로 전역 로크 요청 메시지를 전송한다.

   GLT는 자신이 PCA를 가진 페이지 블록들로 구성되고, LLT는 로크를 보유하면서 지역 버퍼에 캐싱하고 있는 페이지에 대한 블록들로 구성된다.
2. GPM 처리 노드는 PCA 테이블을 참조하여 해당 페이지에 대한 PCA 처리 노드 PN_j로 메시지를 전송한다.
3. PN_j의 로크 관리자 LM_j는 읽기 로크 요청에 대해서는 요청한 로크가 바로 부여가능한 상태이면 자신의 버퍼에 있는 최신 페이지를 응답 메시지와 함께 PN_i로 전송한다.

   쓰기 로크 요청의 경우, GLT의 해당 페이지 블록 내의 버퍼 무효화 벡터를 조사하여 유효한 페이지를 가진 처리 노드로 로크 해제 요청 메시지를 보내고 응답 메시지가 올 때까지 대기한다. 로크를 부여가

능한 상태이면 자신의 버퍼에 있는 최신 페이지를 응답 메시지와 함께 PN_i로 전송한다.

만약 해당 페이지가 지역 버퍼 내에 캐싱되어 있지 않다면 디스크로부터 읽어 들이라는 응답 메시지를 전송한다.

이때 PN_j로부터 로크 해제 요청 메시지를 받은 모든 처리 노드는 해당 페이지에 대해 수행중이거나 대기중인 트랜잭션이 없으면 해당 블록을 LLT로부터 삭제한 뒤 PN_j로 로크 응답 메시지를 보낸다. 그렇지 않으면 x-possible을 1로 세트한다.

4. PN_i의 T가 갱신 트랜잭션이었다면 실행 완료시에 우선 자신의 LLT 내의 해당 페이지 블록을 삭제하여 로크를 해제한 한 뒤 PCA 처리 노드인 PN_j로 갱신된 페이지와 함께 로크 해제 메시지를 전송해야 한다. 읽기 로크의 경우 나중에 PCA 처리 노드로부터 로크 해제 요청 메시지가 올 때까지 로크는 그대로 보유된다.

4.2. PCA 처리 노드에서 버퍼 정보 관리를 통한 기법

이미 살펴본 바와 같이, PCA 처리 노드에 최신 페이지를 유지할 경우 갱신 트랜잭션이 완료될 때마다 페이지 전송을 위한 긴 메시지가 필요하고, PCA 처리 노드에서 빈번한 페이지 교체로 인한 부담이 따른다.

이러한 문제는 PCA 처리 노드에 최신 페이지 대신 해당 페이지에 대한 시스템 전체의 버퍼 관리 정보를 두어 각 처리 노드 내의 페이지가 유효한지를 알 수 있게 함으로써 해결가능하다. 이를 위해 갱신 트랜잭션이 완료될 때 갱신된 최신 페이지 대신에 해당 페이지가 갱신되었다는 정보만 PCA 처리 노드로 전송하면 된다. 따라서 PCA 처리 노드의 버퍼에는 최신 페이지가 있을 수도 있고 없을 수도 있다. 그런데 PCA 처리 노드의 버퍼에 최신 페이지가 캐싱되어 있지 않다고 해도 해당 페이지에 대한 모든 로크 요청은 일단 PCA 처리 노드를 거쳐 최신 페이지를 캐싱하고 있는 처리 노드로부터 페이지를 획득해야 하므로 처리 노드 간의 빈번한 메시지 전송이 발생한다.

디스크 I/O는 PCA 처리 노드로부터 페이지 액세스 요청을 받은 처리 노드의 버퍼에 최신 페이지가 캐싱되어 있지 않을 때 발생한다.

처리 과정은 다음과 같다.

1. 처리 노드 PN_i의 트랜잭션 T로부터 발생한 전역 로크 요청 메시지가 GPM 처리 노드를 거쳐 PCA 처리 노드인 PN_j로 들어오면 로크 관리자 LM_j는 로크를 부여할 수 있는 상황인지를 판단한다. 로크를 바로 부여할 수 있으면 GLT 내의 해당 페이지 블록에 있는 버퍼 무효화 벡터 **buf_inv[PN_i]**를 조사한다.
2. **buf_inv[PN_i]**가 0이면 PN_i 내의 해당 페이지는 유효하므로 응답 메시지를 보내고, 1이면 버퍼 무효화 벡터를 조사하여 최신 페이지를 가진 처리 노드 PN_k로 메시지를 전송한다.
3. PN_k는 자신의 버퍼에 최신 페이지를 캐싱하고 있다면 PN_i로 전송하고, 그렇지 않으면 디스크로부터 읽어 들이라는 메시지를 전송한다.
4. PN_i의 T가 갱신 트랜잭션이었다면 실행 완료시에 PCA 처리 노드로 페이지 정보가 갱신되었음을 전달해야 한다.
5. PCA 처리 노드 PN_j가 PN_i로부터 페이지가 갱신되었다는 메시지를 받으면 **buf_inv[PN_i]**를 0으로 하고, 나머지는 1로 세트함으로써 현재 PN_i만이 최신 페이지를 캐싱하고 있음을 알 수 있다.

4.3. PCA 재할당을 통한 기법

이미 살펴 보았듯이, **4.1**과 **4.2**의 경우 PCA 처리 노드가 고정됨으로 인해 불필요한 메시지 전송과 디스크 I/O가 발생하였다.

이 절에서는 PCA를 동적으로 재할당함으로써 이러한 통신 오버헤드를 줄이고자 한다. PCA를 재할당하는 시기에 따라 두가지 기법이 제안되었다.

4.3.1. PCA 처리 노드에서의 페이지 교체시마다 PCA 재할당

PCA 처리 노드에서 페이지 교체시에 해당 페이지에 대한 PCA를 재할당하여 페이지 교체로 인해 발생하는 디스크 I/O를 없앰으로써 **4.1**과 **4.2**의 기법을 개선시킬 수 있다. 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. 따라서 디스크 I/O는 PCA 처리 노드에서 페이지가 교체되는 시점에서 시스템 내의 어떤 처리 노드에도 해당 페이지가 캐싱되어 있지 않을 때만 일어나며, 이때 PCA 재할당은 고려되지 않는다. 페이지 교체시에 PCA를 재할당하기 위해서는 PCA 처리 노드 외에 현재 어느 처리 노드가 최신 페이지를 캐싱하고 있는가를 알아야 하는데, 이것은 버퍼 관리 정보를 유지함으로써 가능하다.

이 기법은 PCA 처리 노드에서 최신 페이지를 유지하기 때문에 전역 로크 요청을 처리하는 과정은 **4.1**과 같다. **4.1**과 다른 점은 PCA 처리 노드에서 페이지 부재로 인한 디스크 I/O가 발생하지 않는다는 것이다. 즉, PCA 처리 노드에서 최신 페이지에 대한 교체가 발생하면 GLT 내의 해당 페이지 블록의 버퍼 무효화 벡터 **buf_inv[PN$_i$]**를 조사하여 최신 페이지를 캐싱하고 있는 처리 노드에 PCA를 재할당하므로 페이지 교체와 상관없이 PCA 처리 노드는 최신 페이지를 유지할 수 있다. 버퍼 관리 정보를 나타내는 **buf_inv[PN$_i$]**의 값이 0이면 PN$_i$는 최신 페이지를 캐싱하고 있으면서 로크를 보유하고 있음을 의미한다. 각 처리 노드는 로크를 보유한 페이지를 교체할 경우 PCA 처리노드로 로크 해제 메시지를 보내기 때문에 PCA 처리 노드는 페이지에 대한 시스템 전체의 버퍼 캐싱 여부를 정확히 파악할 수 있다.

4.3.2. 갱신 요청시마다 PCA 재할당

이 기법은 PCA로 갱신 요청이 들어오면 요청한 처리 노드에 PCA를 재할당함으로써 갱신 트랜잭션이 완료될 때 별도의 페이지 전송없이 PCA 처리 노드에서 항상 최신 페이지를 유지할 수 있게 한다.

디스크 I/O는 요청된 페이지가 PCA 처리 노드의 버퍼에 캐싱되어 있지 않을 때 발생한다.

처리 과정은 다음과 같다.

1. PN$_i$의 트랜잭션 T로부터 PCA 처리 노드 PN$_j$로 갱신 로크 요청이 들어오면 로크 관리자 LM$_i$는 로크 부여가 가능한 상황에서 GPM 처리 노드에 PCA 재할당 메시지를 보낸다.
2. PN$_j$는 GPM 처리 노드로부터 응답 메시지를 받고 나서 PN$_i$가 새로운 PCA 처리 노드임을 알리는 정보와 함께 응답 메시지를 PN$_i$로 전송한다.
3. PN$_i$는 자신의 GLT에 해당 페이지 블록을 추가하고, T를 실행한다. 이후 해당 페이지에 대한 모든 로크 요청은 GPM 처리 노드를 거쳐 PN$_i$로 전송될 것이다.

4.4. 제안된 기법들의 비교

이 절에서는 예 1을 통해 네가지 기법의 효율성을 비교분석한다. 이를 위해 각 기법에서 페이지 액세스를 위해 필요로 하는 메시지 전송량과 디스크 I/O 오버헤드를 중심으로 고찰한다. 일반적으로 처리 노드 간의 메시지 전송 시간은 디스크 I/O 시간에 비해 상대적으로 짧다.

[예 1] n개의 처리 노드(processing node: PN)를 가진 시스템에서, 페이지 P1에 대해 PCA 처리 노드인 PN1에서 P1에 대한 교체가 발생하여 최신 페이지를 유지하지 못하고, 대신에 PNn의 갱신 트랜잭션 T가 가장 최근에 완료되어 PNn에서 최신 페이지를 유지하고 있다고 가정하자. 그림 3은 PN2가 P1에 대해 페이지 액세스를 요청하였을 때 각 기법에서의 페이지 획득 과정을 보여주고 있다.

4.1의 경우, PN1이 여전히 PCA 처리 노드이므로 일단 PN1에게 로크 요청 메시지가 전송된다. PN1은 자신의 버퍼에 최신 페이지가 없음을 확인하고 PN2에게 디스크로부터 읽어들이라는 메시지를 전송한다. 즉, PN1에 최신 페이지가 있음에도 불구하고 PN2는 디스크로

부터 최신 페이지를 읽어 들인다. 따라서 부가적인 디스크 I/O가 필요하다.

4.2의 경우도 마찬가지로 PN1으로 로크 요청 메시지가 전송된다. 그러나 **4.1**과 다른 점은 자신의 버퍼에 최신 페이지가 없음을 확인하고, PNn으로 페이지 액세스 요청 메시지를 전송한다. PNn은 PN2로 페이지를 전송한다. 따라서 부가적인 메시지 전송이 필요하다.

반면에 **4.3**의 경우, PN1에서 페이지 교체가 발생하면 PNn이 새로운 PCA 처리 노드가 된다. GPM 처리 노드가 PN2로부터의 로크 요청 메시지를 PNn에게 전송하면 PNn은 최신 페이지 P1를 PN2에게 바로 제공할 수 있다. 갱신 요청시에 PCA를 재할당하는 경우에도 PNn이 새로운 PCA 처리 노드가 될 것이므로 처리 과정은 같다. 물론 PCA 재할당을 위해 GPM 처리 노드로 메시지를 전송해야 하지만, 페이지 전송시의 메시지 길이나 디스크 I/O로 인한 오버헤드와 비교해 볼 때 그다지 심각하지 않다. 따라서 **4.1**과 **4.2**에 비해 PN2의 트랜잭션이 최신 페이지를 획득하기 위해 필요로 하는 메시지 수와 디스크 I/O로 인한 오버헤드는 적음을 알 수 있다.

5. 결론 및 앞으로의 연구 방향

데이타베이스를 공유하는 환경에서 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.

본 논문에서는 PCA를 동적으로 관리하기 위한 기법에 대해 연구하였다. 즉, PCA 처리 노드 내의 버퍼에서 페이지가 교체될 때 그리고 페이지에 대한 갱신 요구가 들어 왔을 때 PCA를 재할당하여 각 처리 노드의 작업 부하량을 일정하게 유지함으로써 트랜잭션의 처리율과 응답시간을 개선할 수 있는 알고리즘을 제안하였다. 제안된 동적 PCA 관리와 버퍼 무효화를 위한 기법들은 시스템 환경의 변화시에 적응성있는 ATR과 균형적 부하 분배를 제공함으로써 고성능의 트랜잭션 처리가 가능할 것으로 기대된다.

앞으로 시스템 시뮬레이션을 통해 각 기법의 효율에 대해 제고하고, 성능을 평가하고자 한다. 시뮬레이션 모델은 CSIM 언어

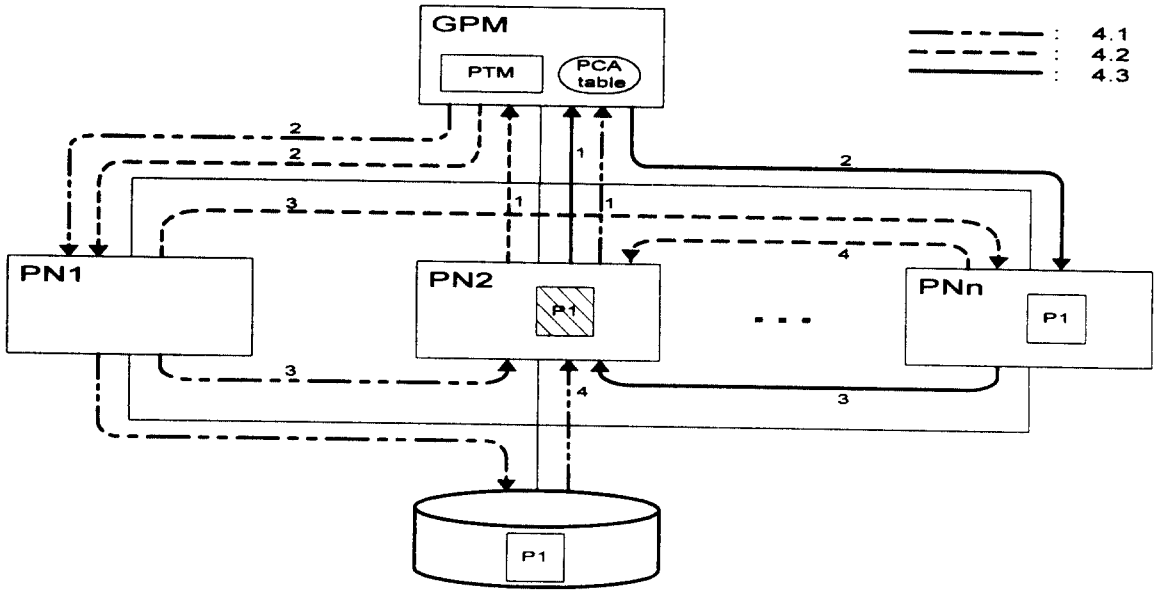


그림 3. 캐쉬 일관성 기법에서의 페이지 획득 과정

(Schwetman, 1992)를 이용하여 구현할 예정이다.

또, 한가지 대안을 추가로 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 크다. 따라서 SPM이 제공하는 PCA 관리의 안정성과 DPM이 제공하는 PCA 관리의 융통성을 혼합한 방식을 제안한다. 이 방식을 혼성 PCA 관리(Hybrid PCA Management: HPM)라 한다.

그림 3에서 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.

참고 문헌

Anon et al., A measure of transaction processing power, Datamation, pp.112-118, April, 1985

A. Borr, Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing, In Proc. 7th Intl. Conf. VLDB, pp.155-165, 1981

Y. Breibart, H. Garcia-Molina and A. Silberschatz, Overview of Multidatabase Transaction Management, VLDB, Vol.1, No.2, 1992

S. Ceri et al., Distributed Databases: Principles and Systems, McGraw-Hill, 1984

Customer Information Control system(CICS), Gerneral Information, IBM Manual GC33-1055-3, 1987

A. Dan and P. Yu, Performance Analysis of Coherency Control Policies through Lock Retention, Proc. ACM SIGMOD, pp.114-123, 1992

W. Du, A. Elmagarmid, Y. Leu and S. Osterman, Effects of Autonomy on maintaining Global Serializability in Heterogeneous Distributed Database

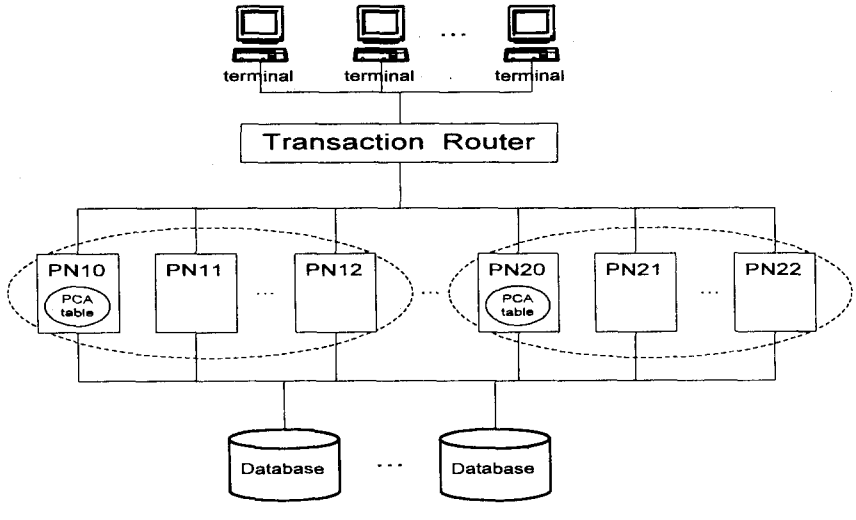


그림 4. 혼성 PCA 관리

Systems, Proc. Data Knowledge System for Manufacturing and Engineering, 1989

M. Franklin and M. Carey, Client-Server Caching Revisited, Computer Science Tech. Rep. #1089 Univ. Wisconsin-Madison, May, 1992

J. Gray et al., One Thousand Transactions Per Second, In proc. IEEE Spring CompCon, San Francisco, pp.96-101, 1985

J. Gray, The Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufmann, 1991

J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann Pub., pp.406-419, 1993

H. Korth and A. Silberschatz, Database System Concepts, 2nd Ed., McGraw-Hill, pp.353-357, 1991

N. Kronenberg, M. Levy, and D. Strecker, VAX clusters: A Closely Coupled Distributed Syetem, ACM Trans. on Computer Systems, Vol.4, No.2, pp.130-146, 1986

E. Rahm, A Framework for Workload Allocation in Distributed Transaction Systems, J. System Software, Vol.18, No.3, pp.171-190, 1992

E. Rahm, Algorithms for Efficient Load Control in Multiprocessor Database Systems, Angewandte Informatik 28(4), 161-169, 1986

E. Rahm, Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems, ACM Trans. on Database Systems, Vol.18, No.2, pp.333-337, 1993

E. Rahm, Primary Copy Synchronization for DB-Sharing, Information Systems, Vol.11, No.4, pp.275-286, 1986

A. Reuter, Load Control and Load Balancing in a Shared Database Management System, Proc. 2nd IEEE International Conf. on Data Eng., pp.188-197, 1986

A. Reuter and K. Shoens, Synchronization in a Data Sharing Environment, IBM San Jose Research Lab., Tech. Rep.(preliminary version), 1984

H. Schwetman, CSIM Users Guide for use with CSIM Revision 16, MCC, 1992

K. Shoens et al., The AMOEBA Project, Proc. IEEE CompCon, pp.102-105, 1985

P. Strickland et al., IMS/VS: An Evolving System, IBM Systems Journal, Vol.21, No.4, pp.490-510, 1982

Y. Wang and L. Rowe, Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture, Proc. ACM SIGMOD Conf., pp.367-376, 1991