주사본 권한을 지원하는 공유 데이타베이스 시스템을 위한 캐쉬 일관성 기법

김 신 희† · 조 행 래†† · 강 병 욱†††

요 약

데이타베이스 공유 시스템(Database Sharing System: DSS)은 고성능 트랜잭션 처리를 위해 제안된 시스템이다. DSS에서 고속의 통신망으로 연결된 노드들은 별도의 메모리와 운영체제, 그리고 DBMS를 가지며, 데이타베이스를 저장하고 있는 디스크는 모든 노드에 의해 공유된다. 빈번한 디스크 액세스를 피하기 위해 각 노드는 자신의 메모리 버퍼에 최근에 액세스한 페이지들을 캐싱한다. 그러나 동일한 페이지가 여러 노드에 의해 동시에 캐싱될 수 있으므로, 각 노드가 최신의 내용을 항상 액세스하기 위해서는 캐싱된 데이타의 일관성이 유지되어야 한다. 본 논문에서는 로킹 오버헤드를 줄이기 위해 주사본 권한을 이용하여 데이타베이스를 논리적으로 분할한 DSS 구조에서 필요한 캐쉬 일관성 기법들을 제안한다. 제안된 기법들은 캐쉬 일관성을 위해 소요되는 메시지 전송량과 디스크 입출력 오버헤드를 줄임으로써 성능을 향상시키며, 데이타베이스 부하가 동적으로 변하는 경우에도 효율적으로 동작한다는 장점을 갖는다.

Cache Coherency Schemes for Database Sharing Systems with Primary Copy Authority

Shinhee Kim† · Haengrae Cho†† · Byeonguk Kim†††

ABSTRACT

Database sharing system (DSS) refers to a system for high performance transaction processing. In DSS, 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 cache consistency schemes in DSS, 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.

※ 본 논문은 정보통신부 국책기술개발사업의 부분적인 지원으로 수행되었음

† 정 회 원 : 영남대학교 컴퓨터공학과 박사과정
†† 정 회 원 : 영남대학교 컴퓨터공학과 조교수
††† 종신회원 : 영남대학교 컴퓨터공학과 교수

논문접수 : 1997년 10월 18일, 심사완료 : 1998년 3월 31일

1. 서 론

데이타베이스 공유 시스템(Database Sharing System: DSS)은 고성능의 온라인 트랜잭션 처리가 필요한 응용분야를 효율적으로 지원하기 위하여 제안된 시스템이다[12,19]. DSS는 별도의 메모리와 운영체제, 그리고 DBMS를 포함하는 노드들로 구성되며, 데이타베이스를 저장하고 있는 디스크는 모든 노드에 의해 공유된다. 뿐만 아니라, 노드들은 물리적으로 인접한 위치에 존재하며, 고속의 통신 시스템을 이용하여 메시지 교환을 통해 교신한다. DSS의 장점은 DBMS들이 별개의 노드에서 상호 독립적으로 동작하기 때문에 트랜잭션 실행시 부하 균형이 가능하고, 새로운 노드의 추가가 용이하며, 기존 노드의 고장이 전체 시스템에 영향을 주지 않는다는 점 등을 들 수 있다. DSS의 상용 제품으로 IBM사의 IMS/VS 데이타 공유 버전[17], AMOEBA project[16], DEC사의 VAX DBMS[7], 그리고 Oracle사의 parallel server[10] 등이 있다.

DSS에서 각각의 노드는 자신의 버퍼에 최근에 액세스한 페이지들을 캐싱한다. 노드가 항상 최신의 데이타를 사용할 수 있기 위해서는 버퍼에 캐싱된 데이타의 일관성이 유지되어야 한다[2,3,8,12]. 이를 위해 버퍼 관리자는 캐쉬 일관성 기법을 지원하여야 하며, 이러한 캐쉬 일관성 기법은 동시성 제어 기법과 밀접한 관계를 갖는다.

DSS에서 많이 사용되는 동시성 제어 기법은 2 단계 로킹 기법이며, 로킹을 구현하는 기법으로는 중앙집중형 기법과 분산형 기법이 각각 존재한다. 중앙집중형 기법은 공유 데이타에 대한 모든 로킹 정보를 관리하는 하나의 전역 로크 관리자(Global Lock Manager: GLM)가 존재한다[2, 8]. 중앙 집중형 기법을 사용할 경우 캐쉬 일관성 기법은 비교적 단순해지는데, 그 이유는 GLM이 데이타베이스 페이지의 최신 버전을 캐싱하고 있는 노드들에 대한 정보를 관리할 수 있기 때문이다. 그러나, 이 기법에서는 모든 로크 요청 및 해제 메시지가 GLM으로 전송되므로 GLM으로의 통신 집중에 따른 성능 저하가 발생할 수 있다. 뿐만 아니라, GLM 노드의 고장시 전체 시스템의 동작이 중단되므로 신뢰성이 떨어진다는 단점도 존재한다.

분산 기법에서는 데이타베이스가 논리적으로 분할되어 각 분할에 대한 로킹 정보는 DSS를 구성하는 노드들에 나누어 저장된다[12,13]. 각 노드에 할당된 데이타베이스 분할에 관한 권한을 주사본 권한 (Primary Copy Authority: PCA)이라 부른다[14]. 이때 노드가 액세스하는 데이타는 자신이 PCA를 가지고 있는 지역 데이타와 다른 노드에게 PCA가 할당된 원격 데이타로 나눌 수 있다. 중앙집중형 기법에 비해 분산 기법은 지역 데이타를 액세스하는 경우 로킹에 관련된 통신이 필요없으므로 성능이 향상되고, 로킹 정보가 여러 노드에 나누어 저장되므로 신뢰성을 향상시킬 수 있다.

분산 기법이 최적의 성능을 나타내기 위해서는 각 데이타의 최신 버전이 그 데이타에 대한 PCA를 가지고 있는 노드의 버퍼에 캐싱되어 있을 확률이 커야 한다. 예를 들면, 데이타 P_1의 PCA는 노드 N_1이 가지고 있지만, P_1의 최신 버전은 노드 N_2에 캐싱되어 있다고 가정하자.[1] 이때 노드 N_3에서 실행되는 트랜잭션이 P_1의 로크 및 최신 데이타를 N_1에게 요청할 때, 로크가 허용되면 N_1은 P_1의 최신 내용을 N_3에게 전송하도록 하는 메시지를 N_2에게 보내야 한다. 즉, 추가적인 페이지 전송 요청 메시지가 N_1에서 N_2로 전송되어야 한다. 데이타베이스 참조 형식이 변하거나[18] PCA 할당이 잘못 설정된 경우, PCA를 가진 노드에서 지역 데이타를 액세스할 가능성이 줄어들게 되고, 따라서 빈번한 페이지 전송 요청 메시지로 인해 시스템 성능이 크게 저하될 수 있다.

이러한 문제점에 대해 Rahm은 다음과 같은 2가지 해결책을 제시하였다[13]. 먼저, N_2에서 P_1을 갱신한 트랜잭션이 완료할 때 P_1의 최신 내용을 공유 디스크에 저장하는 방법이다. 이후 N_3로부터 전송된 로크 요청에 대해 N_1은 로크 허용 메시지에 P_1의 최신 내용이 공유 디스크에 있음을 알리는 내용을 포함한다. 그러나, 이 방법은 빈번한 공유 디스크 입출력으로 인해 성능이 더욱 저하될 수 있다. 실제로, [8]에서는 네트워크를 통해 페이지를 전송하는 것이 공유 디스크를 이용하는 것보다 좋은 성능을 보인다고 주장하고 있으며, 이러한 관점은 의뢰자-서버 DBMS 분야에서도 제안된 바 있다[4].

1) N_2에서 실행되는 트랜잭션이 P_1을 갱신한 후, 완료했을 때 이러한 현상이 발생할 수 있다.

또 다른 방법은 N_2에서 P_1을 갱신한 트랜잭션이 완료될 때 P_1의 최신 내용을 N_1에게 전송한 후, N_1이 자신의 버퍼에 이 내용을 캐싱하는 것이다. 이후 N_3에서 발생한 P_1에 대한 최신 버전의 요청은 N_1에서 직접 처리할 수 있다. 그러나, 이 방법은 다음과 같은 문제점들을 가진다. 첫째로, N_1이 아닌 다른 노드에서 P_1이 자주 갱신될 때, P_1의 내용이 여러번 N_1으로 전송되어야 하므로 메시지 전송량이 많아질 수 있다. 둘째로, N_1이 P_1을 캐싱하기 위해서는 기존에 N_1의 버퍼에 존재하던 페이지를 교체하여 P_1을 저장할 버퍼 공간을 할당하여야 하는데, 이후 교체된 페이지를 다시 액세스하고자 할 경우 페이지 부재가 발생하여 공유 디스크에 대한 입출력이 필요하다. 특히 노드의 버퍼 용량이 적거나 빈번한 데이타 갱신이 발생할 경우, N_1은 자신이 액세스하는 페이지와 다른 노드에서 전송된 페이지들을 동시에 캐싱할 수 없게 된다. 따라서, 과도한 페이지 부재로 인한 thrashing 현상이 발생할 수 있다.

본 논문에서는 PCA를 이용하여 분산 동시성 제어 기법을 지원하는 DSS를 위한 캐쉬 일관성 기법들을 제안한다. 제안된 기법들은 PCA를 동적으로 할당함으로써 페이지 부재율을 줄이고 노드들간의 메시지 전송 오버헤드를 줄인다. 뿐만 아니라, 새로운 노드가 DSS에 포함되거나 기존 노드의 고장 발생시, 혹은 데이타베이스 참조 형식이 변할 때와 같은 시스템 환경의 변화에 대한 적응성을 제공하며, 노드들간에 부하 불균형 문제를 해결함으로써 고성능의 트랜잭션 처리를 가능하게 한다.

본 논문의 구성은 다음과 같다. 2 절에서는 다양한 PCA 할당 방식을 설명하고, 각 방식들의 장단점 및 구현시 고려 사항에 대해 살펴본다. 3 절에서는 본 논문에서 제안한 동적 PCA 할당 방식을 이용한 캐쉬 일관성 기법을 설명한다. 제안된 캐쉬 일관성 기법의 성능을 분석하기 위하여 본 연구에서 개발한 모의 실험 모형을 4 절에서 설명한 후, 5 절에서 실험 결과를 분석하도록 한다. 마지막으로, 6 절에서 본 논문의 결론을 맺는다.

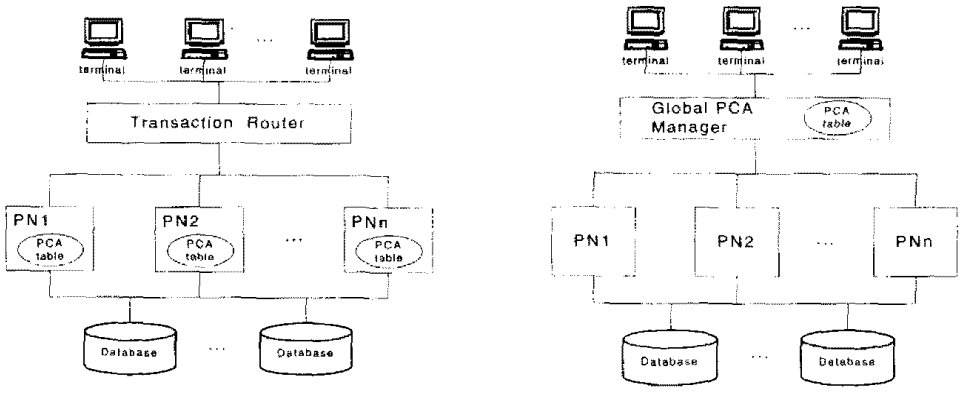
2. PCA 관리

PCA 관리 방식은 PCA의 변화를 쉽게 지원하는가 혹은 그렇지 않은가에 따라 정적 PCA 관리와 동적 PCA 관리로 나누어질 수 있다.

정적 PCA 관리는 시스템 전체의 PCA 할당에 관한 정보를 카탈로그(catalog) 형식으로 구성하여 모든 노드에 중복시키는 방식이다[11]. 이 방식에서는 일단 카탈로그가 만들어지면 시스템 구성에 대한 변동사항이 발생하지 않는 한 카탈로그 정보를 갱신하지 않는다. 각 노드는 트랜잭션의 로크 요청이 발생하면 카탈로그를 참조하여 지역 데이타에 대한 요청이면 바로 처리하고, 그렇지 않으면 해당 PCA 노드로 로크 요청 메시지를 전송하여 처리한다. 정적 PCA 관리의 장점은 원격 데이타에 대한 PCA 노드를 추가적인 메시지 오버헤드 없이 빨리 찾을 수 있다는 점이다. 이 방식은 기존의 주사본 로킹 기법[12,13]에서 사용되었다.

정적으로 PCA를 관리할 경우 PCA는 항상 일정한 노드에 고정되어 있다. 따라서 시스템이 운영되는 도중 특정 노드에 과다한 부하가 걸릴 경우 적응성있게 대처하지 못하여 성능저하를 초래할 수 있다. 또한 새로운 노드가 추가되는 경우 모든 노드의 카탈로그는 재조정되어야 한다. 물리적인 시스템 구성의 변동이나 노드의 부하량의 변화시에 보다 융통성있고 적응성있게 대처하기 위해 PCA를 동적으로 할당하는 방식을 동적 PCA 관리라 한다[11].

PCA를 동적으로 할당하는 방법 중의 하나가 로크 보유 기법이다[1,3]. 로크 보유(lock retention)의 기본 개념은 트랜잭션 실행 중에 획득한 로크를 트랜잭션이 완료된 후에도 해제하지 않고 계속 유지하는 것이다. 이후 동일한 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 없으므로 부하가 분산되는 효과를 가진다. 뿐만 아니라 로크 보유 기법을 이용함으로써 각 노드에서 자주 액세스되는 데이타에 대한 로크는 그 노드에서 유지될 가능성이 높고, 따라서 PCA가 동적으로 할당된다고 볼 수 있다. 그러나 자주 액세스되지 않는 데이타라 할지라도 데이타를 액세스할 때마다 그 데이타에 대한 로크가 유지되기 때문에 로크 관리에 대한 많은 부담이 따른다. 뿐만 아니라, 기존에 유지하고 있는 로크와 상충된 새로운 로크



(그림 11) PCA 관리 방식에 따른 시스템 구성
(Fig. 1) System Organization on PCA Management Strategies

가 요청될 경우 여러 노드들에 대한 해제 작업이 이루어져야 한다.

그림 1은 PCA 관리 방식에 따라 구성된 시스템 예이다. (a)는 정적 PCA 관리 방식으로 분산 동시성 제어 기법을 구현할 경우의 시스템 구성도로서, 전체 시스템의 PCA 정보를 저장한 PCA 테이블이 모든 노드(processing node: PN)에 의해 중복된다. (b)는 동적 PCA 관리를 위해 구성한 시스템으로서, 정적 PCA 관리 방식과 달리 별도의 전역 PCA 관리자(Global PCA Manager: GPM) 노드를 두고 있다. GPM 노드는 트랜잭션 경로 배정외에 PCA 테이블을 관리하는 역할을 담당한다. PCA 테이블은 GPM 노드에만 유일하게 존재하며, 각 노드는 자신에게 할당된 PCA 정보만을 유지한다. GPM 노드는 PCA 테이블을 참조하여 각 노드로부터의 전역 로크 요청 메시지를 해당 PCA 노드로 전송한다.[2]

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

3. 캐쉬 일관성 기법

DSS에서는 다수의 노드가 동시에 동일한 페이지 사본을 버퍼에 캐싱할 수 있다. 캐쉬 일관성 기법은 임의의 노드에서 발생한 액세스 요청에 대해 항상 최신 페이지를 제공하기 위해 필요하다. 본 절에서는 동적으로 PCA를 관리하는 환경에서 디스크 입출력 오버헤드와 노드 간의 메시지 전송량을 줄일 수 있는 기법들을 제안한다. 이를 위해 먼저 제안된 기법들이 가정하고 있는 로크 관리 방식에 대해 3.1절에서 설명하고, 3.2절에서 제안된 캐쉬 일관성 기법들에 대해 설명한다.

3.1 로크 관리

트랜잭션은 액세스하는 페이지와 액세스 유형으로 구성된 연산 리스트 형태로 터미널에서 발생하며, GPM 내의 트랜잭션 경로 배정기를 거쳐 적절한 노드로 배정된다. 노드 내의 트랜잭션 스케쥴러는 전역 로크 테이블과 지역 로크 테이블을 관리한다. 전역 로크 테이블은 전역 로크 요청을 처리하기 위해 자신이 PCA를 가

2) GPM과의 메시지 전송을 줄이기 위해 각 처리 노드에서 최근에 액세스한 PCA 테이블의 일부를 캐싱하는 것이 가능하다. 이러한 방법은 분할된 카탈로그를 사용하는 이질형 분산 데이타베이스 시스템에서 성능을 개선하기 위해 제안된 바 있다[6].

각 페이지에 대한 시스템 전체의 로킹 정보를 포함하고, 지역 로크 테이블은 지역 로크 요청을 처리하기 위해 필요한 로킹 정보를 포함한다. 로킹은 페이지 단위로 이루어진다고 가정한다. 트랜잭션은 연산 리스트의 순서로 실행되는데 만약 액세스할 페이지에 대해 자신이 PCA 노드이면 지역적으로 바로 처리하고, 그렇지 않으면 전역 로크 요청 메시지를 GPM으로 전송한다. GPM 내의 PCA 테이블 관리자는 전역 로크 요청 메시지에 대한 PCA 노드를 식별하여 해당 노드로 메시지를 전송하여 로크 요청을 처리할 수 있게 한다.

전역 로크 요청 메시지는 〈노드 식별자, 트랜잭션 식별자, 페이지 식별자, 로크 모드, 지역 버퍼내에 캐싱하고 있는 페이지의 page_LSN〉으로 구성된다. 이때 로크 모드는 판독 연산일 경우 "S"로 설정되며, 갱신 연산일 경우 "X"로 설정된다. page_LSN은 그 페이지를 최종적으로 갱신한 연산에 대한 로그의 일련 번호를 저장하는 필드이다[9].

본 논문에서는 S 로크에 대한 로크 보유 기법을 지원한다. 즉, 각 노드는 트랜잭션 완료시에 S 로크에 대해서는 바로 해제하지 않고 PCA 노드로부터 로크 해제 요청 메시지가 들어 올 때까지 계속 유지한다. 일반적으로 판독 연산이 갱신 연산에 비해 자주 실행되며 S 로크는 공유될 수 있으므로, S 로크만 보유함으로써 로크 보유에 따른 오버헤드를 최소화할 수 있으며 성능을 최적화할 수 있다. 동일한 주장이 의뢰자-서버 DBMS 분야에서도 제안된바 있다[5].

3.2 캐쉬 일관성 기법

전역 로크 요청 메시지에 포함된 page_LSN은 캐쉬 일관성을 유지하기 위해 사용된다. 전역 로크를 요청하는 노드는 자신의 버퍼에 캐싱된 페이지의 page_LSN 값을 메시지에 포함시킨다. 이때 버퍼에 해당 페이지가 캐싱되어 있지 않다면 page_LSN 값을 "-1"로 둔다. PCA 노드는 로크가 허용될 경우, 전송된 메시지에 포함된 page_LSN을 이용하여 로크를 요청한 노드에 캐싱된 페이지가 최신 페이지인지를 파악한다. 이를 위해 PCA 노드는 최신 페이지에 대한 정보를 유지하여야 하며, 전송된 page_LSN과 유지하고 있는 page_LSN을 비교한다. 전송된 page_LSN과 유지하고 있는 page_LSN 값이 같을 경우, 로크를 요청한 노드에 캐싱된 페이지는 최신 페이지이다. 그러나, 유지하고 있는 page_LSN 값이 클 경우, 로크를 요청한 노드는 최신 페이지를 캐싱하기 위해 공유 디스크를 액세스하거나 다른 노드로부터 최신 페이지를 전송받아야 한다.[3]

PCA 노드가 최신 페이지 정보를 유지하는 방법은 크게 두가지로 나눌 수 있다. 첫번째 방법은 PCA 노드가 각 페이지에 대해 최신 버전의 page_LSN과 최신 버전을 캐싱하고 있는 노드에 대한 정보를 유지하는 것이다[8]. 페이지 전송이 필요할 경우, 최신 버전을 캐싱하고 있는 노드중 하나를 선택하여 페이지 전송 요청 메시지를 해당 노드에 전송한다. 이 방식의 문제점은 모든 노드의 버퍼 상태를 PCA 노드들이 정확하게 추적해야 된다는 것이다. 즉, 페이지가 갱신되어 page_LSN이 변하거나, 캐싱된 페이지가 버퍼에서 교체되어 공유 디스크에 기록될 때 이러한 사실이 PCA 노드로 전송되어야 한다. 이러한 과정은 많은 수의 메시지 전송이 필요하며 최신 버전에 대한 정보를 저장하기 위한 추가적인 메모리 구조가 요구된다. 뿐만 아니라, 메시지들간의 전송 시차로 인하여 버퍼에서 이미 교체되어 디스크에 저장된 페이지에 대한 전송 요청이 PCA 노드로부터 발생할 수도 있다. 보다 현실적인 방식은 [13]에서 제안된 바와 같이 PCA 노드가 항상 최신 페이지를 캐싱하도록 한다는 것이다. 즉, 페이지 전송이 필요한 경우, 다른 노드들과의 메시지 전송없이 PCA 노드로부터 원하는 페이지가 해당 노드로 바로 전송될 수 있다.

본 절에서는 PCA 노드가 최신 페이지를 캐싱하도록 하기 위한 새로운 캐쉬 일관성 기법들을 제안하고 각 기법의 장단점을 살펴보도록 한다. 제안된 기법들의 특징을 이해하기 위하여, 먼저 PCA를 정적으로 유지하는 기존 기법을 살펴보도록 한다.

3.2.1 정적 캐쉬 일관성 기법(SPCA)

SPCA는 PCA가 정적으로 유지될 때 사용되는 기법

3) Oracle Parallel Server[10]의 경우, 노드간의 페이지 전송은 공유 디스크를 통해 이루어진다. 즉, 페이지를 전송할 노드가 해당 페이지를 디스크에 기록한 후, 전송받는 노드가 디스크에서 그 페이지를 판독함으로써 노드간의 페이지 전송이 이루어진다. 그러나, [8]에서 지적되었듯이 이러한 방식은 빈번한 디스크 I/O로 인하여 성능이 매우 저하될 수 있다. 그러므로, 본 논문에서는 노드의 메모리상에서 네트워크를 통해 페이지가 직접 전송된다고 가정한다.

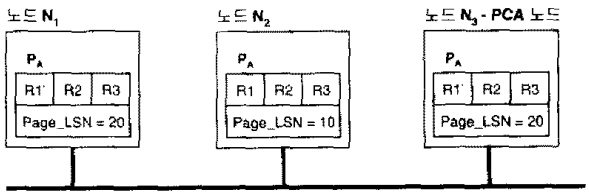
으로, [13]에서 제안되었다. 기본 개념은 임의의 노드에서 갱신 트랜잭션이 완료될 때마다 PCA 노드로 갱신된 페이지를 전송한다는 것이다. PCA 노드는 전송받은 페이지를 자신의 버퍼에 캐싱하고, 이 페이지에 대한 이후 액세스 요청은 PCA 노드에서 바로 처리할 수 있다. 그러나, SPCA는 다음과 같은 몇가지 문제점을 가지고 있다. 먼저 PCA 노드로 전송되는 로크 해제 메시지에 갱신된 페이지를 포함해야 하기 때문에 통신 오버헤드가 증가한다. 뿐만 아니라, 로크 해제 메시지를 전송하는 노드에서 추가적인 디스크 입출력이 발생할 수도 있다. 즉, 갱신된 페이지가 트랜잭션 실행중에 버퍼에서 교체되었을 경우, 트랜잭션 완료시 페이지를 PCA 노드로 전송하기 위해서는 해당 노드가 그 페이지를 디스크에서 다시 판독하여야 한다. 마지막으로, 빈번한 갱신이 발생할 경우 PCA 노드는 자신이 액세스하는 페이지와 다른 노드에서 전송된 모든 페이지를 동시에 캐싱할 수 없게 된다. 따라서, 과도한 페이지 부재로 인한 thrashing 현상이 발생할 수 있고, 디스크 입출력의 빈도수가 급격히 증가할 수 있다.

이러한 SPCA의 문제점을 해결하기 위하여 본 절에서는 PCA의 동적 할당을 지원하는 캐쉬 일관성 기법들을 제안한다. 제안된 캐쉬 일관성 기법들은 PCA를 변경하는 조건에 따라 나누어진다. 첫 번째 기법은 PCA 노드의 버퍼에서 페이지가 교체될 때 PCA를 변경하며, 두 번째 기법은 페이지가 갱신될 때 PCA를 변경하는 것이다.

3.2.2 페이지 교체시마다 PCA를 재할당(DPCA_P)

DPCA_P는 SPCA에서 발생할 수 있는 thrashing 현상을 피하기 위해 제안된 기법이다. DPCA_P의 기본 개념은 PCA 노드에서 페이지 교체시에 해당 페이지에 대한 PCA를 재할당함으로써 디스크 액세스를 줄인다는 것이다. 즉, 페이지를 교체할 때 최신 페이지를 캐싱하고 있는 다른 노드로 PCA를 재할당하면 교체선에 굳이 디스크에 기록할 필요가 없고, PCA는 항상 최신 페이지를 가진 노드에 할당되므로 이후의 페이지 참조시에도 디스크로부터 읽어들일 필요가 없다. 물론 PCA 재할당을 위한 메시지 오버헤드가 있지만 디스크 액세스로 인한 오버헤드보다는 크지 않다. PCA 노드에서 페이지가 교체됨으로 인한 디스크 액세스는 PCA를 재할당할 노드가 존재하지 않는 경우에만 일어나며, 이에 PCA 재할당은 고려되지 않는다.

예를 들어 아래 그림 2를 살펴보자. 그림 2는 노드 N_1에서 레코드 R_1을 R_1'로 갱신한 후, 갱신된 페이지 P_A를 PCA 노드인 N_3로 전송한 후의 결과를 나타낸다. 이후 N_3의 버퍼에서 P_A가 교체된 경우, SPCA에서는 P_A의 내용이 디스크에 기록되고 P_A의 PCA 노드는 계속 N_3로 고정된다. 그러나, P_A의 최신 버전이 N_1에 존재하므로, DPCA_P에서는 P_A의 PCA를 N_1으로 변경한다. 이때 P_A를 디스크에 기록하는 책임이 N_1으로 변경되므로, N_3가 P_A의 내용을 디스크에 기록할 필요가 없어진다.



(그림 12) 세 개의 노드로 구성된 DSS
(Fig. 2) A DSS with 3 nodes

PCA를 재할당하기 위해서는 현재 어느 노드가 최신 페이지를 캐싱하고 있는가에 대한 정보를 알아야 하는데, 이것은 로크 보유 개념을 이용한다. 로크 보유는 S 로크에 대해 적용되므로 다수의 노드에서 로크를 공유할 수 있고, 만약 로크를 보유한 노드의 버퍼에 해당 페이지가 캐싱되어 있다면 최신 페이지일 것이므로 디스크 액세스나 페이지 전송없이도 PCA를 재할당할 수 있다. 로크를 보유한 노드는 해당 페이지가 교체되거나 다른 노드에서 그 페이지를 갱신하고자 할 때, 보유 로크를 해제하고 이를 PCA 노드에게 알린다. 따라서, PCA 노드는 로크를 보유하고 있는 모든 노드가 항상 최신 페이지를 캐싱하고 있음을 보장할 수 있다. 단, SPCA와 동일하게 DPCA_P의 경우 갱신 트랜잭션이 갱신한 페이지는 트랜잭션 완료시에 PCA 노드로 전송되어야 한다.

3.2.3 갱신 요청시마다 PCA를 재할당(DPCA_U)

DPCA_U의 기본 개념은 갱신 트랜잭션의 완료시에 갱신된 페이지의 전송을 하지 않도록 하는 것이다. 이를 위해 PCA 노드로 X 로크 요청이 들어오면 요청한 노드에 로크 허용과 동시에 PCA를 재

한다. 따라서 SPCA나 DPCA_P와는 달리 갱신 트랜잭션이 완료될 때 페이지를 전송할 필요가 없으므로 메시지 오버헤드가 적고, PCA 노드에서의 페이지 교체율도 줄일 수 있다.

예를 들어, 그림 2에서 노드 N_2가 P_A를 갱신할 경우를 생각해보자. N_2는 P_A의 소유자 노드인 N_3에게 P_A에 대한 X 로크를 요청하게 되고, 로크가 요청될 경우 N_3는 N_2에게 P_A의 최신 버전을 전송할 것이다. 이 때 SPCA와 DPCA_P에서는 N_3가 P_A의 PCA를 계속 가지게되며, N_2에서 P_A를 갱신한 트랜잭션이 완료할 경우 P_A의 최신 버전이 N_3로 다시 전송되게 된다. 그러나, DPCA_U에서는 P_A에 대한 X 로크가 허용될 때 P_A의 PCA가 N_2로 변경되므로, 이후 P_A의 최신 버전이 N_3로 전송될 필요가 없다.

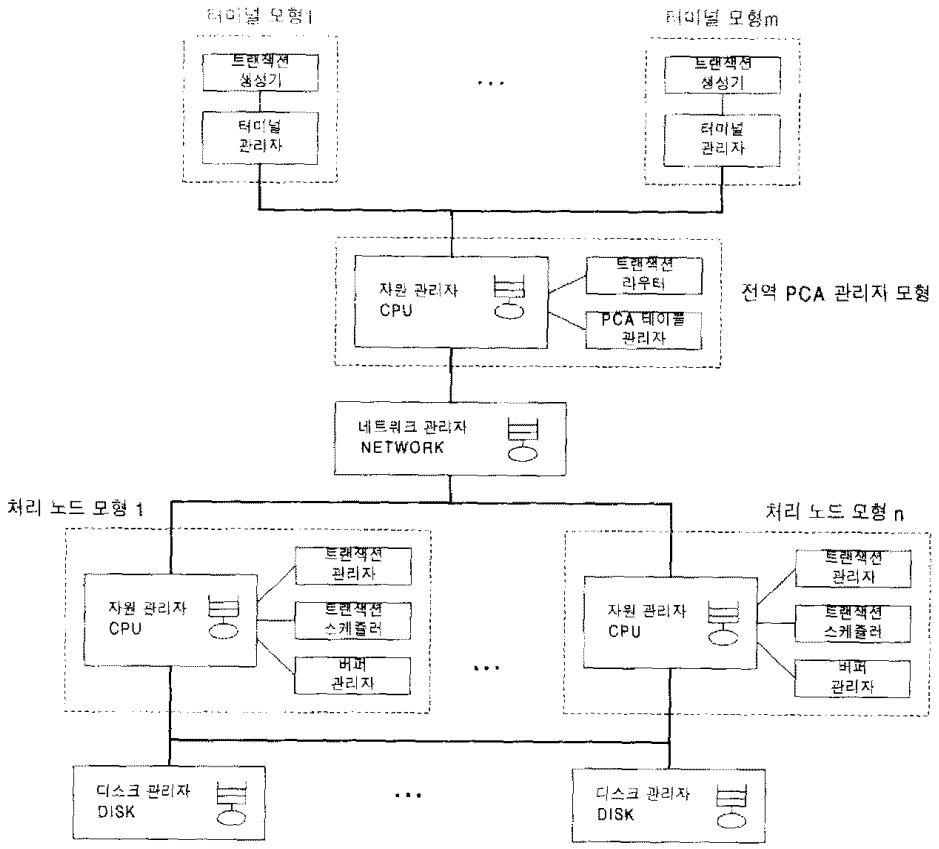
DPCA_U의 또다른 장점은 PCA 변경 오버헤드가 DPCA_P에 비해 작다는 것이다. 그 이유는 DPCA_P에서 페이지 P_A에 대한 PCA를 변경할 경우, 여러 노드가 P_A에 대한 S 로크를 보유할 수 있으므로 기존의 로크 정보 및 캐싱 정보를 새로운 PCA 노드에게 전송하여야 한다. 그러나, DPCA_U는 P_A에 대한 X 로크가 허용된 후 PCA를 재할당하며 X 로크는 공유될 수 없으므로, 로크 정보가 새로운 PCA 노드로 전송될 필요가 없다. 즉, P_A에 대해서는 새로운 PCA 노드외에 로크를 보유한 다른 노드가 존재할 수 없다. 뿐만 아니라, P_A의 최신 버전은 새로운 PCA 노드만 캐싱하고 있으므로, 캐싱 정보도 전송할 필요가 없다.

3.2.4 비교

본 논문에서 제안한 DPCA_P와 DPCA_U는 PCA를 동적으로 할당함으로써 기존의 SPCA에 비해 메시지 전송 오버헤드 및 디스크 액세스 빈도수를 줄일 수 있다. PCA의 변경 사항은 GPM 노드에게 즉시 반영되므로, 특정 페이지에 대한 PCA 노드의 검색은 GPM 노드를 이용하여 이루어질 수 있다. 뿐만 아니라, DPCA_P에서는 PCA가 변경될 때 새로운 PCA 노드에게 로크 정보 및 캐싱 정보가 전송된다. 앞 절에서 설명했듯이 DPCA_U에서는 새로운 PCA 노드에게 로킹이나 캐싱에 관련된 정보가 전송될 필요가 없다. 따라서, DPCA_P와 DPCA_U의 경우 모두 로킹이나 캐싱에 관해 손실되는 정보없이 캐쉬 일관성이 정확히 이루어질 수 있다.

PCA를 정적으로 유지하는 경우에 비해 PCA를 재할당하게 되면 두가지의 부담이 따른다. 첫째, PCA 테이블이 GPM 노드에만 존재하므로 전역 로크 요청 및 해제를 위해 GPM 노드를 거쳐야 한다는 것이다. 그러나 이러한 부담이 시스템의 성능에 많은 영향을 미치지는 않는다. 즉, 전역 로크 요청에 대한 응답 메시지에 포함되어 있는 PCA 노드 정보를 캐싱함으로써 GPM 노드를 거치지 않고 바로 PCA 노드로 로크 해제 메시지를 전송할 수 있다. 뿐만 아니라, 읽기 로크가 보유되므로 트랜잭션이 완료되어도 반드시 로크를 해제할 필요가 없다. 특히 DPCA_U의 경우, PCA 노드에 의해서만 갱신 로크가 처리되므로 로크 해제 메시지를 보낼 필요가 없다.

둘째, PCA를 재할당하게 되면 PCA 노드의 전역 로크 테이블에 저장된 해당 페이지의 전역 로킹 정보를 PCA 재할당 메시지에 포함하여 새로운 PCA 노드로 전송해야 한다. 또한 새로운 PCA 노드는 전송된 전역 로킹 정보를 자신의 전역 로크 테이블에 추가해야 한다. 버퍼 크기가 작을수록 재할당 횟수가 많아지기 때문에 PCA 재할당에 따른 부담은 커진다. 그러나 DPCA_P의 경우, 재할당 횟수가 많을수록 SPCA에 비해 최신 페이지를 버퍼에 캐싱할 확률은 더 커지므로 재할당에 따른 부담을 상쇄하기에 충분하다. 그 이유는 재할당 메시지 전송에 따른 오버헤드와 로크 테이블 관리와 관련된 CPU 연산은 디스크 액세스에 따른 오버헤드에 비해 상대적으로 적기 때문이다. 앞에서 설명했듯이 DPCA_U는 전역 갱신 로크 요청이 발생할 때 로크 허용과 함께 PCA를 재할당하므로 PCA 재할당에 따른 메시지 오버헤드는 DPCA_P에 비해 적다. 그러나 DPCA_U는 DPCA_P에 비해 PCA 재할당 빈도수가 증가할 수 있다는 단점을 갖는다. 특히 자주 갱신되는 데이타 (hot-spot 데이타)가 존재할 경우, 빈번한 PCA 재할당이 예상된다. 정적으로 PCA를 유지하는 SPCA와 관련시켜 고려해 볼 때, PCA 재할당 횟수가 성능에 미치는 영향은 갱신 트랜잭션 완료시에 PCA 노드로의 페이지 전송 오버헤드, 그로 인한 PCA 노드에서의 페이지 교체, PCA 노드에서의 최신 페이지 캐싱 확률 등 여러가지 측면에서 검토해야 한다.



(그림 13) 모의 실험 모형
(Fig. 3) Simulation Model

4. 모의 실험 모형

본 절에서는 제안된 캐쉬 일관성 기법들의 성능을 평가하기 위해 개발된 모의 실험 모형에 대해 기술한다. 모의 실험은 미국의 MCC에서 개발한 CSIM 언어[15]를 이용하여 수행되었다. 모의 실험을 위한 공유 데이타베이스 환경이 그림 3에 나타난다.

그림 3에서 전역 PCA 관리자(GPM)는 네트워크를 통해 노드와 연결되어 있다. 각 노드는 모든 디스크를 공유한다. 터미널의 트랜잭션 생성기는 일정 간격으로 트랜잭션을 생성하고, 터미널 관리자는 생성된 트랜잭션을 GPM으로 전송한다. GPM의 트랜잭션 라우터는 터미널로부터 전송된 트랜잭션을 적절한 노드로 배정하는데, 배정 기준은 트랜잭션이 액세스하는 페이지에 대해 가장 많은 PCA를 가진 노드를 선택하는 것이다. PCA 테이블 관리자는 PCA 테이블을 토대로 노드로부터 들어온 전역 로크 요청을 처리하고, PCA 재할당시 PCA 테이블을 조정한다. 각 노드는 별도의 버퍼를 가지며, 버퍼 관리자는 LRU 정책을 바탕으로 자신의 버퍼를 관리한다. 또한 자원 관리자는 자신에게 할당된 CPU 작업을 모델링하며 네트워크를 통한 GPM과의 통신 및 노드 간의 페이지 전송과정을 수행한다. 트랜잭션 관리자는 로크 요청 및 페이지 액세스 요청 등 트랜잭션을 실제로 실행하는 역할을 담당한다. 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 논문에서는 SPCA와 DPCA_P, 그리고 DPCA_U의 성능을 평가하도록 한다. DPCA_P와 DPCA_U의 경우는 그림 3의 모형에서 실험을 수행하였으며, SPCA에서는 PCA 테이블이 모든 처리 노드들과 GPM에 중복되어 저장된다.

본 논문에서 사용한 입력 매개 변수는 표 1과 같다. 각 매개 변수의 구체적인 값은 [1]에서 많이 참조하였다.

〈표 1〉 입력 매개 변수
〈Table 1〉 Input Parameters

시스템 구성 변수		
LCPUSpeed	처리 노드의 CPU 속도	10 MIPS
GPMCPUSpeed	GPM의 CPU 속도	50 MIPS
NetBandWidth	네트워크의 데이타 전송 속도	10Mbps
NumDBMS	처리 노드의 수	3-10
NumTerms	시스템 전체의 터미널 수	100
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 갯수	5000
BufSize	처리 노드의 버퍼 크기	20 % of DB size
오버헤드 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
트랜잭션 변수		
CreationDelay	트랜잭션 생성시 평균 대기 시간	1 second
TranSize	트랜잭션당 평균 페이지 액세스 수	10 pages
TRSizeDev	트랜잭션 길이의 편차	0.1
WriteOpPct	갱신연산의 비율	0.2

전체 시스템의 병목 현상을 방지하기 위해 GPM이 사용하는 CPU는 다른 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 입출력 요청 및 로크 요청 등을 들어온 순서대로 처리한다. 네트워크 관리자는 10Mbps의 대역폭을 갖는 FIFIO 서버로 구현되었다. 네트워크를 통해 메시지를 전송하는 과정을 표현하기 위해 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.

모의 실험에서 사용되는 주요 성능 지수는 단위 시간당 트랜잭션 처리율(throughput)과 트랜잭션당 평균 응답 시간(response time)이다. 트랜잭션 처리율은 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 모형은 폐쇄 큐잉 시스템이므로 트랜잭션 처리율과 응답 시간은 동일한 결과를 나타낸다. 즉, 트랜잭션 처리율이 높은 캐쉬 일관성 기법은 응답 시간이 빠르다. 그러므로 본 논문에서는 성능 지수로서 트랜잭션 처리율만을 표현한다. 본 논문에서 사용된 보조 성능 지수로 트랜잭션당 평균 디스크 액세스 횟수를 사용하였다. 그 이유는 트랜잭션의 응답 시간이 대략적으로 로크 요청을 처리함에 따른 지연시간, 페이지 전송에 따른 부담시간, 그리고 디스크 액세스 시간으로 구성된다고 보았을 때, [8]에서 지적한 바와 같이 디스크 액세스 시간이 트랜잭션 처리율에 미치는 영향이 가장 크기 때문이다.

신뢰성있는 모의 실험 결과를 얻기 위해 배치 평균 기법(batch mean method)을 이용하였다. 본 논문에

서 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.

5. 결과 분석 및 검토

본 절에서는 모의 실험 모형을 이용하여 캐쉬 일관성 기법들에 대해 실험 결과를 분석하고 성능을 평가한다. 다양한 작업 환경에서 캐쉬 일관성 기법들의 성능을 분석하기 위하여, 데이타베이스 액세스 유형을 분할 액세스 환경과 높은 충돌 환경으로 나누어 실험을 하였다.

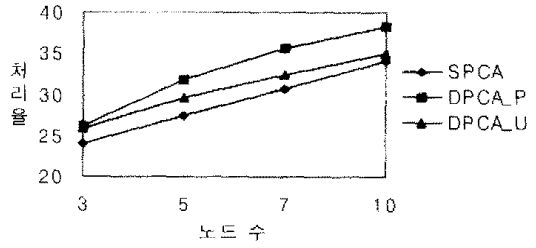
5.1 분할 액세스 환경

분할 액세스 환경은 트랜잭션들이 액세스 유형에 따라 몇 개의 그룹으로 나누어지며, 각 그룹에 속한 트랜잭션들은 특정 데이타를 빈번하게 수행하는 경우이다. 노드들의 PCA를 트랜잭션 그룹에 관련하여 할당할 경우, 트랜잭션의 지역 처리 가능성이 증가할 수 있으므로 DSS의 가장 바람직한 환경중의 하나라고 할 수 있다. 본 논문에서는 분할 액세스 환경을 각 그룹에 할당된 데이타를 액세스하는 비율에 따라 낮은 참조 지역성 환경과 높은 참조 지역성 환경으로 다시 나누어 실험을 하였다.

5.1.1 낮은 참조 지역성

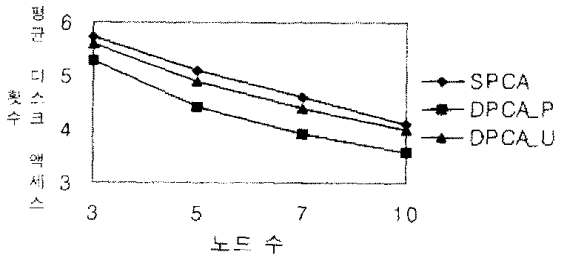
이 환경에서는 트랜잭션이 실행하는 연산의 20%만 지역적으로 처리되고, 나머지 80%는 전역적으로 처리되도록 한 환경으로서, 트랜잭션이 공유 데이타베이스를 무작위로 액세스하는 경우에 해당한다. 따라서 전역 로크 처리가 많아지게 되고 PCA 노드로의 페이지 액세스 요청이 빈번히 발생한다. 그 결과로, 각 노드의 지역 버퍼의 페이지 교체율도 증가한다. 그림 4는 낮은 참조 지역성 환경에서의 트랜잭션 처리율을 나타내며, 이때의 디스크 액세스 횟수가 그림 5에 나타난다.

노드의 수가 증가할수록 트랜잭션 처리율은 증가하였다. 그 이유는 시스템 전체의 관점에서 보았을 때 노드의 수가 증가할수록 사용가능한 버퍼 크기가 증가하므로 액세스하고자 하는 페이지를 메모리 버퍼에서 발견할 가능성이 커지기 때문이다. 뿐만 아니라, 노드 수만큼 공유 데이타베이스를 논리적으로 분할하여 노드에게 페이지에 대한 PCA를 할당하므로 노드 수가 많을수록 각 노드에게 할당되는 PCA 수가 적다. 따라서 PCA 노드의 버퍼에서 최신 페이지를 캐싱하고 있을 확률이 높기 때문에 디스크 액세스를 줄일 수 있다.



(그림 14) 낮은 참조 지역성 환경에서 트랜잭션 처리율

(Fig. 4) Transaction Throughput at Low Locality of Reference



(그림 15) 낮은 참조 지역성 환경에서 평균 디스크 액세스 횟수

(Fig. 5) Average Disk Access Numbers at Low Locality of Reference

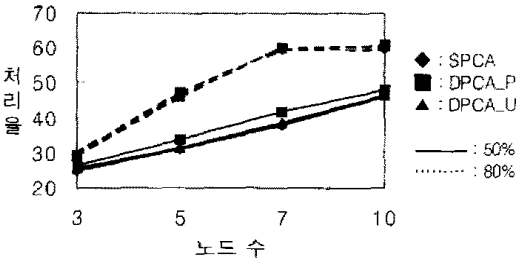
PCA를 정적으로 유지하는 SPCA에 비해 DPCA_P가 약 20%, DPCA_U가 약 10% 정도 성능이 향상되었다. SPCA에서 각 노드는 별도의 PCA 할당 테이블을 가지므로 전역 로크 요청 및 해제시에 GPM 노드를 거칠 필요없이 바로 해당 PCA 노드로 메시지를 전송 가능하므로 로크 요청 및 해제와 관련된 통신 오버헤드는 DPCA_P와 DPCA_U에 비해 적다. 그러나 그림 5에 나타나듯이 SPCA에 비해 DPCA_P와 DPCA_U의 평균 디스크 액세스 횟수가 작다. 디스크 액세스에 대한 오버헤드가 메시지 전송 오버헤드보다 훨씬 크기 때문에, 디스크 액세스를 많이 하는 SPCA의 성능이 다른 기법들에 비해 낮게 나타났다.

DPCA_P에 비해 DPCA_U에서 디스크 액세스 횟수가 많은데 그 이유는 다음과 같다. 첫째, DPCA_U의 경우 PCA 노드에는 최신 페이지가 없지만 다른 노드에는 최신 페이지가 존재할 때에도 디스크 액세스를 수행한다. 그러나 DPCA_P에서는 이러한 경우가 존재할 수 없다. 둘째, SPCA와 DPCA_P의 경우 갱신 트랜잭션이 완료될 때마다 갱신된 페이지를 PCA 노드로 전송해야 하는데, 이로 인해 최신 페이지 전송에 따른 메시지 오버헤드가 있고 또 PCA 노드에서의 잦은 페이지 교체가 따른다. 그러나 SPCA와는 달리 DPCA_P에서는 교체될 페이지가 최신 페이지라면 디스크에 기록하지 않고 해당 시점에서 그 페이지를 캐싱하고 있는 다른 노드로 PCA를 재할당하므로, 페이지 교체가 빈번히 발생한다고 하더라도 그로 인한 디스크 액세스는 많지 않다. 마지막으로, DPCA_P에서 PCA 노드로 전송되는 갱신된 페이지는 PCA 노드의 LRU 버퍼의 최상위에 저장되게 된다. 따라서 이 페이지가 이후 교체될 가능성은 매우 낮다. 이에 대해 DPCA_U에서 갱신된 페이지는 트랜잭션 완료시 재할당된 PCA 노드의 LRU 버퍼에서 위치가 바뀌지 않는다. 따라서 DPCA_P에 비해 갱신된 페이지가 PCA 노드에서 빨리 교체되므로 캐쉬 미스(cache miss)가 발생할 확률이 상대적으로 크다.

그림 4에서 보면 SPCA와 DPCA_P에 비해 DPCA_U는 노드 수가 증가함에 따라 성능 향상 정도가 비교적 완만하다. 그 이유는 노드 수가 많아질수록 다양한 노드에서 갱신 연산이 발생할 수 있기 때문이다. 즉, DPCA_U는 PCA 노드가 아닌 노드에서 갱신 연산이 발생할 경우 PCA를 재할당하므로 노드 수가 많아질수록 PCA 재할당이 빈번하게 발생한다. 따라서 노드 수의 증가로 얻은 장점이 어느 정도 상쇄된다. 뿐만 아니라, 노드 수가 증가함에 따라 시스템 전체 관점에서 사용가능한 버퍼의 크기가 증가하고 각 노드에 할당된 PCA 수는 감소하는데, 이러한 점이 DPCA_U에 비해 SPCA와 DPCA_P에 보다 좋은 영향을 미치기 때문이다. 즉, SPCA와 DPCA_P는 PCA 노드가 아닌 노드에서 갱신된 페이지는 PCA 노드로 전송되어야 하는데, 각 노드에 할당된 PCA 수는 감소하였으므로 PCA 노드가 관리해야될 페이지 수가 줄어든다. 따라서 갱신된 페이지의 전송으로 인한 페이지 교체의 발생 가능성이 낮아진다.

5.1.2 높은 참조 지역성

이 환경은 트랜잭션의 액세스 유형이 높은 참조 지역성을 가짐에 따라 하나의 PCA 노드에서 지역적으로 처리될 가능성이 높은 경우이다. 이를 위해 트랜잭션이 50%의 참조 지역성을 갖는 경우(트랜잭션이 액세스하는 페이지의 50%가 한 노드에서 지역적으로 처리되고 나머지 50%에 대해서는 다른 노드에 의해 전역적으로 처리되도록 하는 경우)와 80%의 참조 지역성을 갖는 경우에 대해 실험을 각각 수행하였다. 그림 6은 각 경우에서 트랜잭션 처리율을 나타내는데, 실선은 참조 지역성이 50%일 경우이며 점선은 참조 지역성이 80%일 경우이다.



(그림 6) 높은 참조 지역성 환경에서 트랜잭션 처리율
(Fig. 6) Transaction Throughput at High Locality of Reference

참조 지역성이 낮을 때에 비해 모든 캐쉬 일관성 기법들의 성능이 향상되었다. 그 이유는 참조 지역성이 높아지면 대부분의 로크가 PCA 노드에 의해 지역적으로 처리되므로, 로크 요청을 처리하기 위한 시간과 페이지 전송에 따른 메시지 오버헤드가 감소하기 때문이다. 뿐만 아니라, 각 노드에서 실행되는 트랜잭션들이 공통된 페이지를 액세스할 가능성이 높아지기 때문에, 각 노드에서 페이지 교체가 일어날 확률이 적고 최신 페이지가 PCA 노드의 버퍼에 캐싱되어 있을 확률도 높아지게 된다.

참조 지역성이 커질수록 캐쉬 일관성 기법들간의 차이가 줄어들었으며, 참조 지역성이 80%인 경우 캐쉬 일관성 기법들의 성능이 거의 동일하게 나타났다. SPCA와 본 논문에서 제안한 기법들간의 차이는 버퍼 공간의 부족으로 인해 페이지 교체가 빈번하게 발생할 경우(DPCA_P), 혹은 PCA 노드가 아닌 노드에서 갱신이 발생할 경우(DPCA_U)에서 나타난다. 그러나 참

소 지역성이 높아지면, 충각의 경우능이 발생할 가능성이 낮아지므로 그만큼 캐쉬 일관성 기법들간의 차이도 줄어들게 된다.

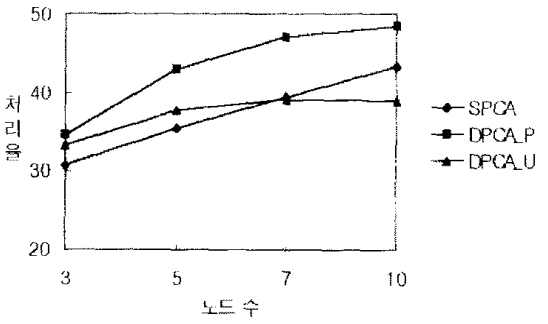
참조 지역성이 50%인 경우, 노드 수가 3일 때와 10일 때 캐쉬 일관성 기법들간의 차이는 비교적 작게 나타났다. 그 이유는 시스템에서 사용 가능한 버퍼 용량과 각 노드에 할당된 PCA 수에 기인한 것이다. 즉, 노드 수가 아주 작을 경우 노드에 할당된 PCA 수는 매우 커진다. 각 노드는 자신이 PCA를 가지고 있는 모든 페이지들을 버퍼에 캐싱할 수 없고, PCA 노드가 아닌 다른 노드에서 최신 페이지를 캐싱할 확률도 상대적으로 낮아진다. 뿐만 아니라, 현재 캐싱된 페이지가 교체되기 전에 다시 액세스될 가능성도 매우 낮으므로, DPCA_P나 DPCA_U가 SPCA에 비해 좋은 성능을 갖기가 힘들어진다. 반대로, 노드 수가 매우 클 경우는 각 노드에 할당된 PCA 수가 현격히 작아지므로, PCA를 가지고 있는 대부분의 페이지들이 지역 버퍼에 상주할 수 있다. 따라서 SPCA에서 디스크를 액세스할 확률이 줄어들게 되고, 각 기법들간의 성능도 비슷해진다.

5.2 높은 충돌 환경

높은 충돌 환경은 각 노드에서 실행되는 트랜잭션들이 데이타베이스의 특정 부분을 높은 확률로 액세스하는 경우를 모델링한 것이다. 이 환경에서 각 트랜잭션이 실행하는 연산의 60%는 전체 데이타베이스의 20%에 해당하는 특정부분을 액세스하며, 연산의 40%는 나머지 데이타베이스를 무작위로 액세스한다.

그림 7은 높은 충돌 환경에서 노드 수의 변화에 따른 트랜잭션 처리율을 나타내고 있다. 분할 액세스 환경에서 참조 지역성이 20%와 50%일 때에 비해 처리율은 더 높게 나타났다. 특히 노드 수가 3일 때의 트랜잭션 처리율은 높은 참조 지역성(80%)의 경우보다도 성능이 우수하였다. 그 이유는 데이타베이스의 특정 부분이 집중적으로 액세스되므로 PCA 노드의 버퍼에 캐싱되어 있을 확률이 높기 때문이다. 또한 시스템 전체의 사용가능한 버퍼 크기는 노드 수에 비례하는데, 높은 충돌 환경의 경우 노드 수가 적을 때에도 빈번히 액세스되는 페이지가 PCA 노드의 버퍼에 캐싱되어 있을 확률이 높다.

노드 수가 증가함에 따라 SPCA와 DPCA_P의 성능은 증가하는데 비해 DPCA_U의 성능은 거의 일정하다. DPCA_U에서는 지역 갱신 로크 요청에 대해 로크 허용과 동시에 PCA를 재할당하므로 다른 기법에 비해 PCA 재할당 빈도수가 많다. 이로 인해 PCA 재할당에 따른 별도의 오버헤드가 부가되고, 또 갱신 로크는 공유할 수 없기 때문에 로크 요청시 로크를 보유하고 있는 다른 노드로 로크 해제 요청을 해야 하므로 로크 처리와 페이지 전송을 위한 오버헤드가 크다. 따라서 노드 수가 증가할수록 자주 액세스되는 페이지에 대한 로크 처리 및 페이지 전송을 위한 오버헤드는 증가하므로 최신 페이지가 PCA 노드의 버퍼에 캐싱되어 있을 확률이 높다하더라도 성능은 별 차이가 없게 된다. 높은 충돌 환경에서도 DPCA_P는 SPCA에 비해 최신 페이지를 캐싱할 확률이 높으므로 성능이 우수하다.



(그림 7) 높은 충돌 환경에서 트랜잭션 처리율
(Fig. 7) Transaction Throughput at High Contention Environment

6. 결 론

본 논문에서는 DSS 환경에서 PCA 동적 할당 개념을 이용하여 디스크 액세스를 줄일 수 있는 새로운 캐쉬 일관성 기법인 DPCA_P와 DPCA_U를 제안하였다. PCA를 정적으로 유지하는 환경에서 제안된 기존의 캐쉬 일관성 기법(SPCA)은 PCA 노드에서 최신 페이지를 제공할 수 있도록 하기 위해 갱신 트랜잭션이 완료될 때마다 갱신된 페이지를 PCA 노드로 전송하는데, 이로 인해 디스크 액세스가 요구되는 페이지 교체가 발생하고 페이지 전송에 따른 오버헤드도 커진다. 이에 대해 DPCA_P는 PCA 노드에서 페이지 교체가 발생할 경우, 교체될 페이지의 최신 버전을 캐싱하고 있는 노드로 PCA를 재할당한다. 따라서 최신 페이지를 PCA 노드의 버퍼에 캐싱할 확률을 높임으로써 디스크 액세

스 빈도를 줄일 수 있다. DPCA_U는 갱신 로크 요청이 들어 왔을 때 해당 페이지를 요청한 노드로 PCA를 재할당하므로, 갱신 트랜잭션이 완료될 때 갱신된 페이지를 전송함에 따른 페이지 교체를 고려할 필요가 없고 페이지 전송에 따른 메시지 오버헤드도 줄일 수 있다.

제안된 기법들의 성능을 분석하기 위하여 DSS 환경을 위한 모의 실험 모형을 개발하였고, 다양한 데이타 액세스 환경에서 실험을 하였다. 트랜잭션의 데이타 액세스 유형이 낮은 참조 지역성을 가질 경우, SPCA에 비해 DPCA_P가 약 20%, DPCA_U가 약 10% 정도 성능이 향상되었다. 그러나 참조 지역성이 증가할수록 캐쉬 일관성 기법들간의 성능이 점차 줄어들었다. 대부분의 DSS 응용 분야에서 트랜잭션의 참조 지역성은 30% 내외이며[14], 노드 수가 증가할수록 참조 지역성은 더욱 저하되므로, 낮은 참조 지역성 환경에서 좋은 성능을 보이는 것은 매우 중요하다. 높은 충돌 환경에서는 분할 액세스 환경에서 참조 지역성이 20%일 때와 50%일 때보다 최신 페이지를 캐싱할 확률이 높아 성능이 우수하였다. 특히 노드 수가 적을 때 높은 충돌 환경에서의 성능은 분할 액세스 환경에서보다 우수하였다. SPCA와 DPCA_P는 노드 수가 증가함에 따라 성능이 향상되었는데 비해, DPCA_U는 잦은 PCA 재할당에 따른 오버헤드로 인해 성능 향상이 상당히 완만하였다.

참 고 문 헌

[1] M. Carey, M. Franklin, and M. Zaharioudakis, "Fine-Grained Sharing in a Page Server DBMS," Proc. ACM SIGMOD, pp.359-370, 1994.

[2] A. Dan and P. Yu, "Performance Analysis of Buffer Coherency Policies in a Multisystem Data Sharing Environments," IEEE Trans. on Parallel and Distributed Syst., Vol.4, No.3, pp.289-305, 1993.

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

[4] M. Franklin, M. Carey, and M. Livny, "Global Memory Management in Client-Server DBMS Architectures," Proc. 18th VLDB Conf., pp. 641-654, 1992.

[5] M. Franklin, M. 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.

[6] E. Hong, "Performance of Catalog Management Schemes for Running Access Modules in a Locally Distributed Database System," Proc. 19th VLDB Conf., pp.194-205, 1993.

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

[8] C. Mohan and I. Narang, "Recovery and Coherency Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," Proc. 17th VLDB Conf., pp. 193-207, 1991.

[9] C. Mohan et al., "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM Trans. on Database Syst., Vol.17, No.1, pp.94-162, 1992.

[10] Oracle 7 Parallel Server Concepts and Administration, Oracle Corp. part A42522-1, 1996

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

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

[13] E. Rahm, "Primary Copy Synchronization for DB-Sharing," Information Syst., Vol.11, No. 4, pp.275-286, 1986.

[14] A. Reuter, "Load Control and Load Balancing

in a Shared Database Management System," Proc. 2nd Int. Conf. on Data Eng., pp.188-197, 1986.

[15] H. Schwetman, 'CSIM Users Guide for use with CSIM Revision 16,' MCC, 1992.

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

[17] P. Strickland et al., "IMS/VS: An Evolving System," IBM Syst. J., Vol.21, No.4, pp.490-510, 1982.

[18] P. Yu and A. Dan, "Performance Evaluation of Transaction Processing Coupling Architectures for Handling System Dynamics," IEEE Trans. on Parallel and Distributed Syst., Vol.5, No.2, pp.139-153, 1994.

[19] P. Yu et al., "On Coupling Multi-Systems through Data Sharing," Proc. IEEE, Vol.75, No.5, pp.573-587, 1987.

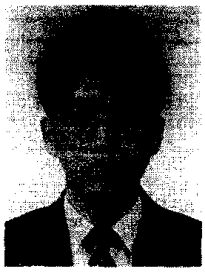
김 신 희

1989년 영남대학교 전산공학과 (학사)

1991년 영남대학교 전산공학과 (공학석사)

1993년~현재 영남대학교 컴퓨터공학과 박사과정 재학중

관심분야 : 트랜잭션 처리, 분산 데이타베이스, 연역 데이타베이스



조 행 래

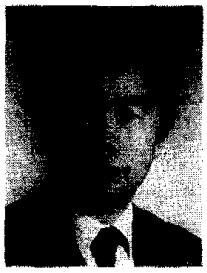
1988년 서울대학교 컴퓨터공학과 (학사)

1990년 한국과학기술원 전산학과 (공학석사)

1995년 한국과학기술원 전산학과 (공학박사)

1995년~현재 영남대학교 컴퓨터공학과 조교수

관심분야 : 트랜잭션 처리, 분산 데이타베이스, DBMS 개발 등



강 병 욱

1970년 영남대학교 전기공학과 (공학사)

1977년 영남대학교 전자공학과 (공학석사)

1994년 경북대학교 전자공학과 (공학박사)

1979년~현재 영남대학교 컴퓨터공학과 교수

관심분야 : 소프트웨어 공학, 프로그래밍 언어, 데이터 압축