주사본 권한을 이용한 동적 트랜잭션 분배 알고리즘

김 기 형[†] · 조 행 래[††] · 남 영 환[†††]

요 약

데이터베이스 공유 시스템(Database Sharing System : DSS)은 고성능 트랜잭션 처리를 위해 제안된 시스템이다. DSS에서 고속의 통신망으로 연결된 노드들은 별도의 메모리와 운영체제를 가지며, 데이터베이스를 저장하고 있는 디스크는 모든 노드에 의해 공유된다. 그리고 빈번한 디스크 액세스를 피하기 위해 각 노드는 자신의 메모리 버퍼에 최근에 액세스한 페이지들을 캐싱한다. 본 논문에서는 DSS를 구성하고 있는 각 노드의 부하를 효과적으로 분산할 수 있는 동적 트랜잭션 분배 알고리즘을 제안한다. 제안한 알고리즘은 각 노드에 할당된 주사본 권한을 이용함으로써 노드별 참조 지역성을 지원하고, 그 결과 캐쉬 이용률을 증가하여 디스크 액세스 수를 최소화한다. 뿐만 아니라, 노드의 현재 부하를 고려하여 트랜잭션 분배 정책을 결정함으로써 특정 노드에 트랜잭션이 집중되는 것을 피한다. 제안된 알고리즘의 성능평가를 위해 시뮬레이션 실험을 수행하였으며, 실험결과 제안된 알고리즘이 기존 알고리즘들보다 트랜잭션 처리율에서 높은 성능을 보였다. 특히 트랜잭션 부하량이 높은 경우와 편중된 데이터 참조를 보이는 경우에 좋은 성능을 보였다.

A Dynamic Transaction Routing Algorithm with Primary Copy Authority

Kihyung Kim[†] · Hangrae Cho[††] · YoungHwan Nam[†††]

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 and a separate copy of operating system. To reduce the number of disk accesses, the node caches database pages in its local memory buffer. In this paper, we propose a dynamic transaction routing algorithm to balance the load of each node in the DSS. The proposed algorithm is novel in the sense that it can support node-specific locality of reference by utilizing the primary copy authority assigned to each node ; hence, it can achieve better cache hit ratios and thus fewer disk I/Os. Furthermore, the proposed algorithm avoids a specific node being overloaded by considering the current workload of each node. To evaluate the performance of the proposed algorithm, we develop a simulation model of the DSS, and then analyze the simulation results. The results show that the proposed algorithm outperforms the existing algorithms in the transaction processing rate. Especially the proposed algorithm shows better performance when the number of concurrently executed transactions is high and the data page access patterns of the transactions are not equally distributed.

키워드 : 데이타베이스(Database), 데이타베이스 공유시스템(Database Sharing System), 트랜잭션 라우팅(Transaction Routing), Primary Copy Authority, 친화도(Affinity), 동적 트랜잭션 라우팅(Dynamic Transaction Routing)

1. 서 론

저렴한 처리 노드들을 연동하여 고성능 트랜잭션 처리 및 복잡한 질의에 대한 빠른 응답 시간을 지원하기 위한 구조로 데이터베이스 분할 시스템(Database Partition System : DPS)과 데이터베이스 공유 시스템(Database Sharing System : DSS)이 제안되었다[1, 5]. DPS는 전체 데이터베이스를 분할하여 각 노드가 데이터베이스의 일정 분할을 관리하는 구조이며, DSS는 모든 노드들이 전체 데이터베이스를 공유하는 구조를 갖는다. DSS는 모든 노드들이 전체 데이터베이스를 공유하므로 하나의 노드가 고장이 나도 다른 노드들은 기존 작업을 계속 수행할 수 있다. 물론 새로운 노드의 추가도 용이하며, 모든 노드들이 동일한 데이터를 액세스할 수 있으므로 노드들간의 부하 분산도 용이하다[10].

DSS에서 많이 사용되는 동시성 제어 기법은 2단계 로킹 기법이며, 로킹을 구현하는 기법으로는 중앙집중형 기법과 분산형 기법이 각각 존재한다. 중앙집중형 기법은 공유 데이터에 대한 모든 로킹 정보를 관리하는 하나의 전역 로크 관리자(Global Lock Manager : GLM)가 존재한다[2, 6]. 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 존

† 종신회원 : 영남대학교 전자정보공학부 교수
†† 정 회 원 : 영남대학교 전자정보공학부 교수
††† 준 회 원 : 쌍용정보통신 IT 솔루션센터 시스템통합 컨설턴트
논문접수 : 2003년 1월 13일, 심사완료 : 2003년 7월 14일

재한다.

분산형 기법에서는 데이터베이스가 논리적으로 분할되어 각 분할에 대한 로킹 정보는 DSS를 구성하는 노드들에 나누어 저장된다[8-10]. 각 노드에 할당된 데이터베이스 분할에 관한 권한을 주사본 권한(Primary Copy Authority : PCA)이라 부른다. 이때 노드가 액세스하는 데이터는 자신이 PCA를 가지고 있는 지역 데이터와 다른 노드에게 PCA가 할당된 원격 데이터로 나눌 수 있다. 중앙집중형 기법에 비해 분산 기법은 지역 데이터를 액세스하는 경우 로킹에 관련된 통신이 필요 없으므로 성능이 향상되고, 로킹 정보가 여러 노드에 나누어 저장되므로 신뢰성을 향상시킬 수 있다.

본 논문에서는 PCA 방식의 분산 로킹 기법을 지원하는 DSS에서 효율적인 트랜잭션 분배 정책을 제안한다. 트랜잭션 분배란 DSS를 구성하는 여러 노드들 중에서 사용자 트랜잭션을 처리할 노드를 결정하여 트랜잭션을 할당하는 것을 의미한다[11]. 이때 트랜잭션이 할당된 노드들을 트랜잭션 처리 노드라 하며, 트랜잭션 처리 노드의 부하량은 해당 노드에 할당된 트랜잭션의 종류와 개수에 따라 결정된다. DSS의 성능, 즉 단위 시간당 트랜잭션 처리량을 향상시키기 위해서는 특정 노드에 트랜잭션이 집중되지 않고 트랜잭션에 대한 효과적인 분배를 통해 각 노드의 부하가 평준화되어야 한다.

DSS에서 기존에 제안된 트랜잭션 분배 기법은 랜덤 분배, 최소부하노드 기반 분배, 그리고 PCA를 이용한 분배의 세 가지 기법이 존재한다[2, 3, 9]. 먼저 랜덤 분배 기법에서는 사용자가 요청한 트랜잭션을 임의의 노드에게 할당한다. 구현이 단순하며 트랜잭션 분배를 위한 추가적인 정보가 필요 없다는 장점이 있지만, 트랜잭션의 참조 지역성을 이용한 캐쉬 효과를 기대할 수 없으므로 디스크 액세스나 노드들간의 페이지 전송이 빈번히 발생하게 되고, 그 결과 전체 시스템의 트랜잭션 처리 효율이 저하될 수 있다.

최소부하노드 기반 분배 기법[11]은 현재 부하가 가장 작은 노드에게 트랜잭션을 할당하는 정책으로, 랜덤 분배 기법에 비해 노드들 간의 부하를 어느 정도 분산할 수 있다는 장점을 갖는다. 그러나 단순히 각 노드의 부하만을 고려하여 트랜잭션을 할당함으로써 동일한 데이터를 액세스하는 트랜잭션들이 여러 노드에 분산될 수 있다. 그 결과, 랜덤 분배 기법과 마찬가지로 노드별 참조 지역성을 지원할 수 없다는 단점을 갖는다.

마지막으로 PCA를 이용한 기법[3, 12]에서는 트랜잭션이 액세스하는 데이터에 대한 대부분의 PCA를 가지는 노드에게 해당 트랜잭션을 할당한다. 이 방법은 트랜잭션의 친화도(affinity)를 이용한 것이다. 즉 동일한 데이터를 참조하는 트랜잭션들을 동일한 노드(PCA 노드)에서 실행함으로써 노드별 참조 지역성을 높여 캐쉬 효율성을 극대화하고, 로킹에 소요되는 메시지 오버헤드를 최소화할 수 있는 장점을 가진다. 그러나 특정 데이터를 참조하는 트랜잭션이 폭주할 경우에는 해당 데이터의 PCA 노드에 과부하가 발생하므로 전체 시스템의 성능이 저하될 수 있다.

본 논문에서는 PCA와 노드의 현재 부하 상태를 모두 고려한 동적 트랜잭션 분배 알고리즘을 제안한다. 제안한 알고리즘은 평상시에는 PCA에 기반하여 트랜잭션을 분배하며, 특정 노드에 대한 트랜잭션 부하가 폭주하게 되면 트랜잭션 라우터는 이를 감지하고 다른 노드로 트랜잭션을 분산시킨다. 이때 분산된 트랜잭션을 처리할 노드는 캐쉬 효율을 최대화할 수 있도록 토큰(token)이라는 데이터 분할의 개념을 이용하여 선정된다. 즉, 과부하가 발생한 PCA 노드의 토큰을 다른 노드 N_t에게 일시적으로 이전한 후, 해당 토큰을 액세스하는 트랜잭션들을 N_t로 할당함으로써 N_t의 메모리에 대한 캐쉬 효과를 증가시키도록 한다. PCA 노드의 부하가 정상 상태로 환원되면, N_t에 할당된 토큰을 회수하여 PCA에 기반한 트랜잭션 분배 정책을 다시 수행한다.

제안한 알고리즘의 성능을 평가하기 위해 이 기종 환경의 DSS에서 트랜잭션 분배와 처리를 위한 모델을 구성하고 시뮬레이션을 수행한다. 시뮬레이션 모델 구성을 위해 웹 기반 환경에서의 시뮬레이션 기능을 제공하는 SimJava 시뮬레이션 라이브러리[4]를 사용한다. 이를 통해 트랜잭션 생성기, 트랜잭션 라우터, 트랜잭션 처리 노드 등을 모델링하여 시뮬레이션을 수행한다. 이때 트랜잭션 라우터는 기존의 트랜잭션 분배 기법과 제안한 알고리즘을 각각 구현한다.

본 논문의 구성은 다음과 같다. 먼저 2절에서는 본 논문의 기반이 되는 PCA 기반의 분산트랜잭션 시스템과 트랜잭션 분배 기법에 대해 살펴본다. 3절에서는 기존의 트랜잭션 분배 기법의 문제점을 개선한 효과적인 동적 트랜잭션 분배 알고리즘을 제안한다. 4절에서는 시뮬레이션 모델과 환경에 대해 설명하고, 시뮬레이션 모델을 이용하여 다양한 시스템 구성과 데이터베이스 부하 조건에서 실험한 결과를 분석한다. 끝으로 5절에서 본 논문의 결론을 맺는다.

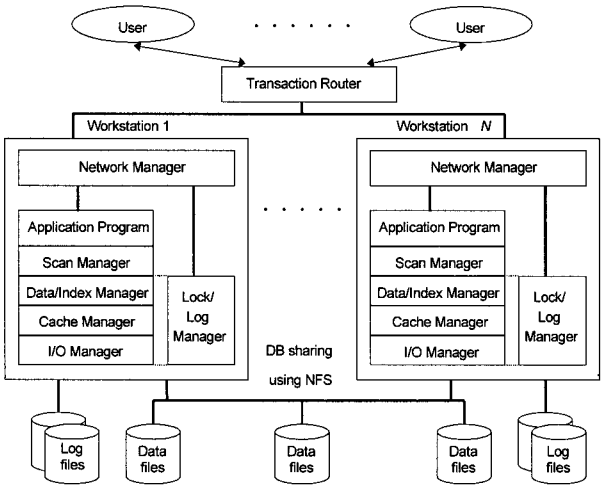
2. 기반 연구

본 절에서는 본 논문의 기반이 되는 PCA 기반의 분산 트랜잭션 시스템을 살펴본다. 또한 기존 PCA 기반 분산 트랜잭션 시스템에서의 트랜잭션 라우팅 기법중 하나인 친화 기반 라우팅 기법을 설명한다.

2.1 PCA 로킹 기법을 이용한 분산 트랜잭션 처리 시스템

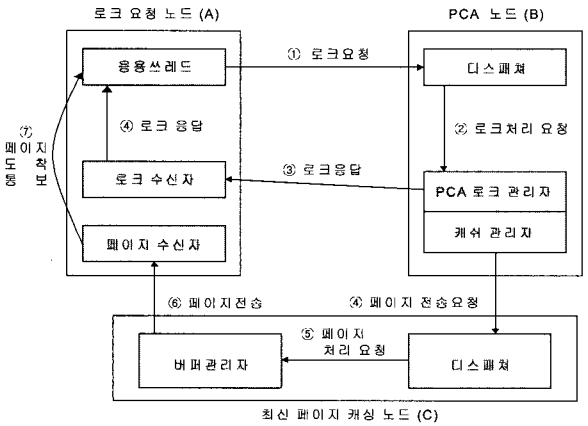
중앙 집중형 동시성 제어 방법에서는 전역 로크 관리자(GLM)가 모든 공유 데이터에 대한 로킹 정보를 관리하는 반면 분산형 동시성 제어 방법에서는 각 노드(워크스테이션)에 할당된 데이터베이스 분할(partition)에 대한 권한, 즉 주사본 권한(PCA)을 각 노드가 가지게 된다. PCA 로킹 기법을 이용한 분산 트랜잭션 처리 시스템은(그림 1)과 같은 구조를 가진다[7]. 각 노드는 공유 데이터베이스에서 데이터를

검색하거나 저장하기 위한 단위 트랜잭션 처리 시스템으로서, 데이터베이스 하부구조에 저장 관리자, 로크/로그 관리자, 그리고 통신 관리자로 구성된다. 사용자로부터 요청된 트랜잭션은 트랜잭션 라우터를 통해 특정 노드로 할당된다.



(그림 1) 분산 트랜잭션 시스템의 전체구조

PCA는 각 디스크 볼륨(또는 페이지)마다 할당되고 특정한 노드가 각 PCA의 로크 정보를 유지하는 책임을 가지게 된다. 한 노드에서 그 노드에 할당된 데이터 페이지에 대한 로크 요청은 통신 오버헤드나 지연이 없이 실행되는 반면, 다른 노드들이 PCA를 가지는 데이터 페이지에 대한 로크 요청은 해당 PCA 노드에게 전달되어 수행되어 진다.



(그림 2) PCA 기반의 분산 로킹 처리 과정

(그림 2)는 일반적인 경우의 분산 로킹 처리 과정을 보여준다. 트랜잭션 처리를 위해 요구되는 데이터는 자신이나 다른 노드에 존재할 수 있는데, 본 예제에서는 트랜잭션이 참조하는 데이터 페이지의 PCA는 노드 B가 가지고 있지만, 노드 C에 페이지의 최신 버전이 캐싱되어 있는 경우를 나타낸다. 노드 A에서 실행되는 트랜잭션은 페이지의 로크 및 최신 데이터를 PCA 노드 B에 요청한다①. PCA 노드 B는 내부적인 로크 처리요청과정②을 통해 로크가 허용되면, 로크 요청에 대한 응답으로 로크요청 노드 A의 로크 수신자에게 로크 응답메시지를 보내는③ 동시에, 최신 페이지의 캐싱 노드 C에게 최신 페이지를 노드 A에게 전송하도록 요청하는 페이지 전송 요청 메시지를 보낸다④. 노드 C는 내부 처리⑤를 거쳐서 이에 대한 응답으로서 캐쉬 데이터를 노드 A에게 전송하게 된다⑥. 노드 A의 페이지 수신자는 캐쉬 데이터를 받은 후 페이지 도착을 처음 로크 요청을 했던 응용 쓰레드에게 페이지 도착을 통보하게 된다⑦.

2.2 PCA 기반 트랜잭션 라우팅 기법

PCA 기반 분산 트랜잭션 시스템에서 트랜잭션 라우터는 사용자가 요청한 트랜잭션을 특정노드에게 할당하는 역할을 하는데, 이를 위해 다양한 라우팅 기법들이 연구되었다. 대표적인 라우팅 기법으로서 본 논문의 기반이 되는 친화도 기반 라우팅 기법은 트랜잭션 타입의 참조행위를 분석하여 라우팅에 이용한다. 즉 같은 데이터베이스 영역을 접근하는 트랜잭션들을 같은 노드에 할당하는 것이다. 결과적으로 다른 노드에서 실행되는 트랜잭션들은 서로 다른 데이터베이스 영역을 접근하게 된다. 이를 위해 트랜잭션 라우터는 PCA 정보를 이용하게 된다. 즉 사용자가 트랜잭션 실행을 요청하게 되면 트랜잭션 라우터는 그 트랜잭션의 참조행위를 분석하고 그 트랜잭션이 필요로 하는 대다수의 PCA를 가진 노드를 찾아서 그 노드에게 트랜잭션을 할당하는 것이다. 이때 대다수의 PCA를 가진 노드를 메이저 PCA 노드라고 부른다.

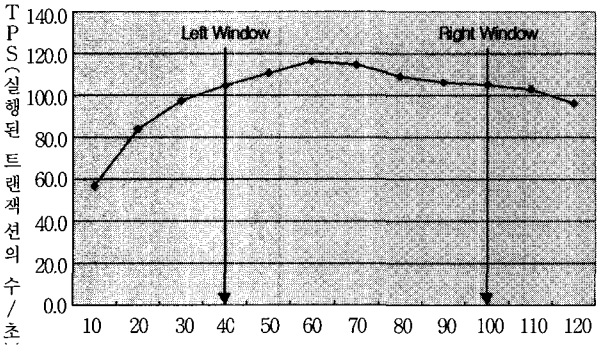
각각의 트랜잭션만을 놓고 볼 때는 이 기법이 로킹을 위한 노드간 통신지연을 최소로 하는 기법이 된다. 그러나 만일 동일한 메이저 PCA 노드를 갖는 트랜잭션들이 집중적으로 요청된다면 이들 트랜잭션들은 특정 노드에 집중될 것이고 전체 시스템 성능을 저하시킬 것이다. 이러한 부하편중 문제를 완화하기 위해서는 시스템을 초기화할 때 각 트랜잭션 클래스의 평균 부하량(단위 시간당 입력되는 트랜잭션 수)과 그 클래스에 속하는 트랜잭션들이 액세스하는 데이터 집합들을 조사하여야 한다. 이를 바탕으로 부하량이 상대적으로 큰 클래스의 트랜잭션들을 실행할 노드는 그 클래스의 PCA만 가지며, 부하량이 적은 클래스를 실행할 노드는 여러 클래스의 PCA를 갖도록 PCA를 할당함으로써 각 노드의 부하 차이가 크지 않도록 설정한다. 그러나 [13]에서 지적한 바와 같이 특정 트랜잭션 클래스가 비정상적으로 폭주하거나 트랜잭션 클래스의 부하량이 시간대별로 변할 경우, 혹은 노드의 고장으로 인해 그 노드에 할당된 트랜잭션 클래스를 다른 노드로 이전되어야 할 경우 등의 여러 가지 요인으로 인해 초기에 가정한 트랜잭션 클래스별 평균 부하량과 실제 노드에 할당되는 트랜잭션 부하량과는 차이가 발생할 수 있다. 그 결과, 노드의 동적 부하량을 이용하는 트랜잭션 라우팅 기법이 필요로 하게 된다.

3. PCA 기반 동적 트랜잭션 분배 알고리즘

본 절에서는 DSS를 구성하는 각 노드의 부하 상태를 고려한 PCA 기반 동적 트랜잭션 분배 알고리즘을 제안한다. 전체 시스템의 트랜잭션 처리율을 극대화 하려면 각 노드의 처리율을 극대화 하여야 한다. 이를 위해 먼저 단일 트랜잭션 처리시스템의 성능을 분석하고 이를 토대로 전체 시스템의 처리율을 극대화 할 수 있는 분배 알고리즘을 제시한다.

3.1 단일 트랜잭션 처리 시스템의 성능 분석

단일 트랜잭션 처리 시스템의 성능을 분석하기 위해 동시에 요청되는 트랜잭션의 수를 변화시키면서 트랜잭션 처리율의 변화를 측정하였다. 이를 위하여 분산 트랜잭션 처리 시스템인 PHLOX[7]를 하나의 노드에서 실행하도록 구성하였다. PHLOX는 DSS 구조의 분산 트랜잭션 시스템이며 동시성 제어를 위해 PCA 기반의 분산 로킹을 지원한다.



(그림 3) 단일 트랜잭션 시스템의 TPC-B 벤치마크 성능 분석 결과

(그림 3)은 그 결과를 보여준다. 할당된 각 트랜잭션마다 독립적인 쓰레드가 할당되기 때문에 일정 범위 이상의 트랜잭션 수에 대해서는 쓰레드들간의 문맥 교환 및 디스크 I/O시간의 증가로 인해 성능의 저하 현상이 나타나게 된다. 그림에 표시된 Left Window(LWin)와 Right Window(RWin)는 각각 일정 수준 이상의 트랜잭션 처리율을 보이는 최소와 최대의 부하량을 나타낸다. 단일 시스템이 최대 성능을 내려면 동시에 요청되는 트랜잭션의 수가 LWin과 RWin 사이가 되도록 해 주어야 함을 알 수 있다.

3.2 동적 트랜잭션 분배 알고리즘

전체 시스템의 성능(트랜잭션 처리율)을 극대화하기 위해서는 각 노드의 성능을 극대화 하여야 하며 이를 위해서는 각 노드에 할당되는 부하를 평준화 하여야 한다. 본 절에서는 분산 트랜잭션 처리 시스템에서의 효과적인 부하 평준화를 위해 캐쉬 효과와 부하 상태를 고려한 트랜잭션 분배 기법을 제안한다. 이를 위하여 본 논문에서 가정하고 있는 동시성 제어 기법과 캐쉬 관리 정책을 설명한 후, 제안한 트랜잭션 분배 알고리즘을 자세히 설명하도록 한다.

3.2.1 동시성 제어 기법과 캐쉬 관리 정책

본 논문에서는 동시성 제어 기법으로 주사본 로킹 기법을 가정하며, PCA 기반 로킹의 자세한 구현 방식은 [7]를 참조하였다. 즉, 레코드와 같은 미세 단위 로킹을 지원하고, PCA는 데이터 볼륨별로 할당하는데 PCA 정보를 관리하기 위해 **[볼륨 식별자, 볼륨에 대한 PCA 노드 식별자]**로 구성된 PCA 테이블을 모든 노드들이 중복하여 저장한다. 이때 각 노드는 자신이 PCA를 가지고 있는 데이터들에 대한 로킹 테이블을 저장한다. 로킹 과정을 설명하기 위하여 노드 N_1에서 실행되는 트랜잭션 T_1이 레코드 R_1을 액세스하는 경우를 가정하자. 만약 R_1이 포함된 데이터 볼륨의 PCA 노드가 N_1일 경우, T_1의 R_1에 대한 로크 요청은 N_1에서 지역적으로 처리할 수 있다. 그러나 R_1의 PCA 노드가 다른 노드(N_2)일 경우에는 해당 로킹 정보를 N_2에서 관리하므로 T_1의 R_1에 대한 로크 요청을 N_2로 전송하여야 한다. 로크 정보가 여러 노드에 분산되므로 분산 교착상태가 발생할 수 있는데, 이는 타임아웃이나 분산 대기 그래프를 유지함으로써 해결할 수 있다. 지역 교착상태의 경우는 지역 대기 그래프를 이용한 교착상태 탐색 방법으로 간단히 해결할 수 있다. 그리고 [10]과 [14]에서는 트랜잭션 T_1이 실행 중에 획득한 로크를 T_1의 완료 후에도 그 트랜잭션을 실행한 노드(N_1)에서 보유하는 방식을 제안하였는데, 본 논문에서는 이를 가정하지는 않는다. 즉, 모든 연산에 대해 로크를 획득하기 위해 PCA 노드와의 통신이 필요하다고 가정한다.

여러 노드의 버퍼에 캐싱된 데이터 페이지의 일관성을 유지하기 위하여 DSS에서 가장 대표적인 캐쉬 관리 정책인 ARIES/SD[6] 알고리즘을 PCA 환경으로 확장하였다. 먼저, [6]에서 제안한 "NoForce 정책"을 지원하여 트랜잭션이 완료될 때 그 트랜잭션이 갱신한 데이터를 디스크에 반드시 저장할 필요는 없도록 한다. 이를 위해 본 논문에서는 각 페이지에 저장된 PageLSN의 개념을 이용한다. PageLSN은 그 페이지를 최종적으로 갱신한 연산에 대한 로그의 일련번호를 저장하는 필드이다. 예를 들면, 노드 N_1의 트랜잭션 T_1이 페이지 A의 한 레코드를 갱신하면, 그 결과로 A의 PageLSN 필드 내용이 변경된다. T_1이 완료할 때 T_1이 보유하고 있던 로크에 대한 해제 요청과 함께 A의 현재 PageLSN 필드 내용이 PCA 노드에게 전송된다. PCA 노드는 T_1의 로크를 해제하기 전에 A의 PageLSN을 캐쉬 정보 테이블에 등록한다. 이후 A를 캐싱하고 있던 노드 N_2의 트랜잭션 T_2가 A에 있는 레코드의 로크를 PCA 노드에게 요청할 때, A에 대한 PageLSN도 같이 요청한다. 그 결과로 A의 현재 PageLSN이 N_2로 다시 전송되게 되고, N_2는 자신이 캐싱하고 있는 A의 PageLSN과 전송받은 PageLSN을 비교한다. 만약 A의 PageLSN이 전송받은 PageLSN보다 작다면 A의 내용이 변경되었다는 것을 의미하므로, N_2는 A의 소유자인 N_1으로부터 A의 최신 버전을 전송받아야 한다.

PCA의 정의 및 여러 노드로 할당하는 방식은 트랜잭션

액세스 패턴과 밀접한 관계가 있다. [12]의 경우, 액세스하는 데이터들의 집합에 따라 트랜잭션들을 몇 개의 클래스로 분리한 후 각 트랜잭션 클래스마다 하나 혹은 여러 개의 노드를 할당하는 방식을 제안하고 있는데, 이러한 방식은 PCA를 정의할 경우에도 활용될 수 있다. 즉, 특정 트랜잭션 클래스가 액세스하는 데이터들의 집합을 하나의 PCA로 정의하며, 각 PCA를 별개의 PCA 노드에게 할당한다. 그리고 각 노드마다 그 노드가 PCA를 갖는 데이터를 액세스하는 트랜잭션 클래스를 라우팅 함으로써 로킹 오버헤드 및 캐쉬 관리 오버헤드를 최소화할 수 있다. 그러나 이러한 장점은 각 트랜잭션 클래스의 부하가 동일하다는 가정 하에서만 성립하므로, 트랜잭션 클래스사이의 부하 편차가 심할 경우에는 동적으로 트랜잭션을 분배할 수 있는 알고리즘이 필요하다.

3.2.2 트랜잭션 분배 알고리즘

제안된 트랜잭션 분배 알고리즘은 데이터 참조 경향을 고려하는 경우, 데이터베이스 공유시스템의 성능(트랜잭션 처리율)을 극대화하기 위해 다음과 같은 트랜잭션 라우터의 일반적인 요구조건을 이용한다.

① 균등한 데이터 참조 경향을 가진 트랜잭션들의 처리가 요청되는 경우에는 캐쉬 영역이나 디스크 공간에 참조 데이터를 보유하고 있는 노드에 트랜잭션을 요청하는 것이 유리하다.
② 편중된 데이터 참조 경향을 가진 트랜잭션들의 처리가 요청되는 경우에 특정 노드가 과부하 상태에 빠지지 않도록 해야 한다.

데이터베이스 공유시스템의 성능(트랜잭션 처리율)을 극대화하기 위해서는 각 노드의 성능을 극대화 하여야 하며, 각 노드의 성능을 극대화하기 위해서 트랜잭션 라우터는 각 노드마다 적정량의 트랜잭션 부하를 갖도록 해야 한다. 또한, 각 노드에서의 처리율을 높이기 위해서는 트랜잭션 처리시간을 줄여야 한다. 트랜잭션 처리시간은 데이터베이스 로킹 과정(그림 2)을 포함한다. 각 노드의 트랜잭션 처리 성능이 같다고 가정하면, 트랜잭션 처리시간을 최소화하기 위해서는 데이터베이스 로킹시간을 최소화하여야 한다. 이를 위해서는 (그림 2)의 네트워크 접근시간((그림 2)의 로크요청 ①, 로크응답 ③, 페이지요청 ④, 페이지전송 ⑥)을 최소화하여야 하며, 이는 트랜잭션을 해당 PCA 노드가 처리함을 의미한다. 이것이 바로 위의 첫 번째 요구조건이다.

편중된 데이터 참조 경향을 가진 트랜잭션에 대해서 PCA에 기반하여 분배를 하면 특정 PCA 노드에게 트랜잭션이 집중되게 된다. 이는 특정 PCA 노드가 과부하상태에 빠지게 되고 다른 노드는 과소부하 상태에 빠지게 된다. 따라서 이 경우 PCA 노드의 트랜잭션의 일부를 다른 노드들에게 분배하게 된다. 이때 데이터베이스 로킹과정(그림 2)에서 로킹처리시간(①, ③)은 피할 수 없게 되고, 페이지 요청 ④과 페이지 전송 ⑥ 시간을 최소화하기 위해서는 참조 데이터베이스 페이지를 캐싱하고 있는 노드에게 트랜잭션이 분배되어야 한다. 이것이 바로 위의 두 번째 요구조건이다.

제안된 트랜잭션 분배 알고리즘은 토큰(token)이라는 개념을 이용한다. 토큰은 부하 분배의 기본 단위로서 일정 데이터를 캐싱할 수 있는 권한을 의미한다. 데이터 참조경향이 균등한 경우에는 PCA 노드가 토큰을 가지고 있는 노드(이하 이를 토큰노드라고 한다.)된다. 따라서 토큰기반 트랜잭션 라우팅은 노드간 부하균형이 이루어지는 상황에서는 PCA 기반 라우팅과 같다. 만일, 특정 PCA 노드에 트랜잭션이 집중될 때는 일부 트랜잭션들을 다른 노드에게 분배하여야 하는데, PCA는 고정되어 있으므로 다른 노드에게 이양될 수 없다. 토큰은 로킹 권한이 아니고 캐싱 권한을 의미하므로 PCA와 달리 다른 노드에게 이양될 수 있다. 따라서 이 경우 토큰기반 트랜잭션 라우팅은 친화도 기반 라우팅이 된다. 즉, 특정 노드가 과부하 상태에 도달하기 전에는 PCA 노드가 토큰노드가 되지만, 만일 특정 노드가 과부하 상태에 도달하게 되면, 그 노드의 일부 토큰을 다른 노드에게 양도하게 된다. 이후 트랜잭션 라우터는 도달하는 트랜잭션들을 과부하상태의 PCA 노드가 아닌 다른 토큰노드에게 전달되게 된다. 이때 토큰노드는 해당 데이터를 캐싱하고 있기 때문에 로크처리만 PCA 노드에게 요청하고 나면 토큰 노드 내부에서 트랜잭션을 처리할 수 있게 된다.

토큰이 지정하는 데이터의 크기(TSize)는 식 (1) 과 같이 구할 수 있다.

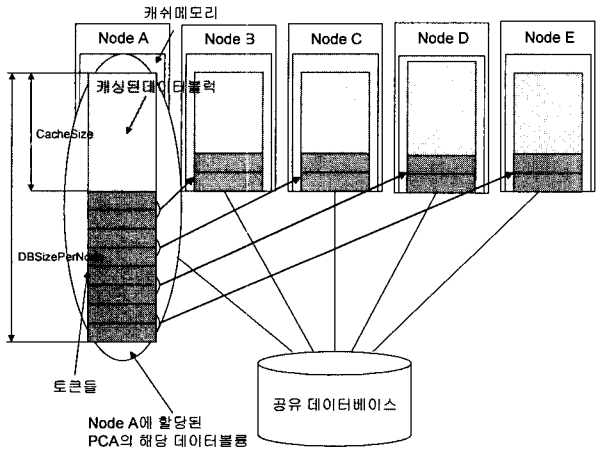
$$TSize = \frac{DBSizePerNode - CacheSize}{(N-1)\times 2} \quad (1)$$

단, 위의 수식에서 N은 데이터베이스 공유시스템의 노드의 수, CacheSize는 각 노드의 메인 메모리의 크기를 의미하며, DBSizePerNode는 각 노드마다 할당된 PCA에 해당하는 데이터베이스의 크기를 의미한다.
또한 각 노드가 가지는 토큰의 수(NTokens)는 식 (2)와 같이 구할 수 있다.

$$NTokens = (N-1)\times 2 \quad (2)$$

특정 PCA 노드가 과부하 상태에 빠지게 되면 트랜잭션 라우터는 해당 PCA 노드의 토큰 2개를 다른 노드에게 이양한다. 이양한 후에도 계속 PCA 노드가 과부하상태에 있게 되면, 계속해서 다른 노드들에게 토큰 2개씩을 이양하게 되며 (그림 4)는 모든 토큰을 다른 노드들에게 이양하는 예를 보여준다. TSize의 의미는 한 PCA 노드에 할당된 데이터 볼륨 중에서 PCA 노드의 메모리 캐쉬에 들어가지 못하는 데이터(즉, 트랜잭션의 처리를 위해 디스크 접근이 필요한 데이터)를 NTokens로 나눈 것이다. NTokens가 (N-1)×2인 것은 한번에 다른 노드에게 이양할 토큰이 2개씩이

기 때문이다. 반면 토큰을 되돌려 받을 때는 1개의 토큰을 되돌려 받게 되는데 그 이유는 급격한 로드의 변화를 막기 위해서이다. 즉, PCA 노드가 토큰을 이양할 때에는 2개씩의 토큰을 주는 반면에, 되돌려 받을 때에는 1개씩의 토큰을 받게 된다.

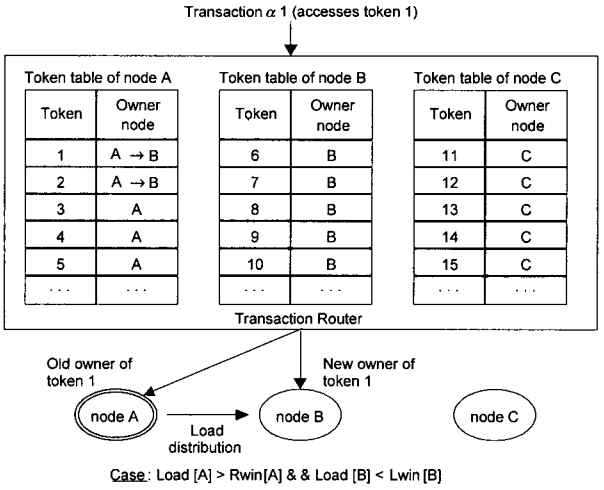


(그림 4) 트랜잭션이 PCA 노드 A에게 집중될때의 토큰의 분배

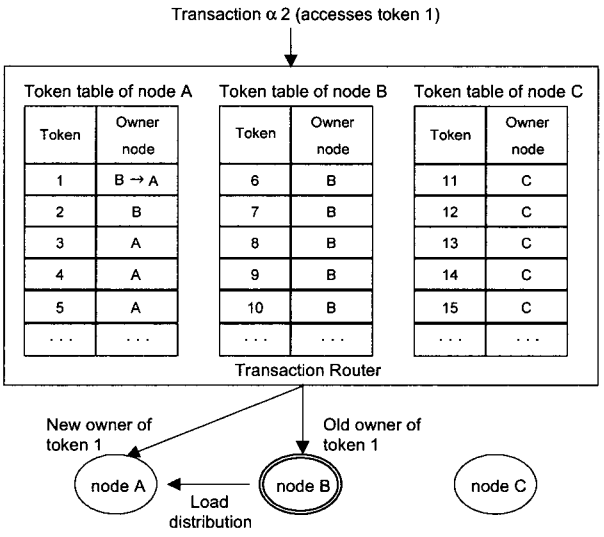
(그림 5)는 토큰을 이용한 부하 분배 알고리즘을 나타낸다. 알고리즘은 먼저 노드 i의 부하가 RWin(Right Window) 값보다 큰 경우((그림 5)의 라인 3)에는 노드 i가 보유하고 있는 토큰 중에 이전에 다른 노드로부터 분산된 토큰이 있는지를 판단한다. 만일 원래 자신이 소유하던 토큰만 존재한다면((그림 5)의 라인 16) (그림 6)와 같이 i 노드를 제외한 전체 노드 중 부하가 LWin(Left Window) 보다 작은 노드를 토큰의 소유자로 고친다((그림 5)의 라인 18~22). 이때 LWin 값보다 작은 노드가 존재하지 않는다면 토큰 정보는 갱신되지 않는다.

```
1 : LoadBalance() {
2 :   for (i = 0; i < NumOfNodes ; i++) {
3 :     if (load[i] > RWin[i]) {        // Node i is overloaded
4 :       if (node i has tokens of other nodes) {
5 :         select a token t to transfer ;
6 :         j = PCA_node(t) ;        // Node j is a PCA node of
                                        token t
7 :         if (load[j] < LWin[i]) { // Node j is underloaded
8 :           freeNode = j ;     // Select node j as a target node
9 :         } else {
10 :          // Find a free node to transfer the token t
11 :          if (∃ a node k such that load[k] < LWin[k]) {
12 :              freeNode = k ; // Node k is the
                                    underutilized node
13 :          } else {
14 :              freeNode = j ;  // Node j is the PCA
                                    node of token t
15 :          }
16 :        }
17 :        transfer the token t to freeNode ;
18 :      } else { // Overloaded node i does not have tokens
                    of other nodes
19 :        select two tokens to transfer ;
20 :        if (∃ a node k such that load[k] < LWin[k]) {
21 :          freeNode = k ;
22 :          transfer two tokens to freeNode ;
23 :          numTransferredTokens [i] += 2 ;
24 :        }
25 :      }
26 :    } else if (load[i] < LWin[i] && numTransferredTokens[i]
                > 0 ) {
27 :      // Node i is is underloaded
28 :      // and it has already transferred token(s) to other nodes.
29 :      u = the maximum overloaded node among the nodes
30 :          which have tokens of node i ;
31 :      get back token(s) from node u ;
32 :    }
33 :  }
34 : }
```

(그림 5) 토큰 기반 동적 부하 분배 알고리즘



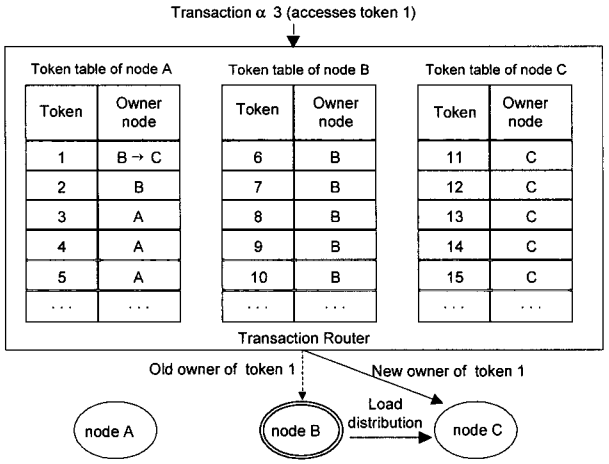
(그림 6) 노드가 자신의 토큰만을 보유한 경우



(그림 7) 다른 노드의 토큰을 보유한 경우(PCA 노드로의 분산)

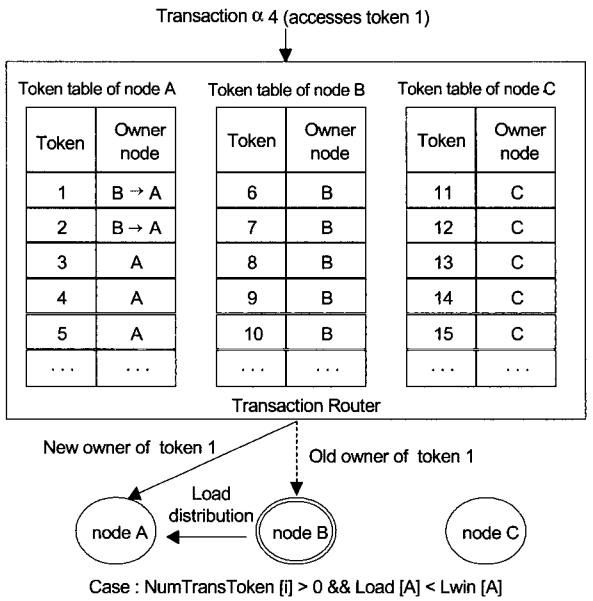
다음으로 노드 i가 다른 노드의 토큰을 보유한 경우((그

림 5)의 라인 4)에는 PCA를 보유한 노드, 즉 PCA 노드의 부하가 LWin보다 작을 때((그림 5)의 라인 7)나, PCA 노드와 다른 노드들의 부하가 LWin보다 클 때((그림 5)의 라인 12) (그림 7)과 같이 PCA 노드가 토큰의 보유노드가 된다. 반면 PCA 노드의 부하가 LWin 보다 크고 다른 노드 중 하나의 부하가 LWin보다 작은 경우((그림 5)의 라인 9~10)에는 (그림 8)과 같이 그 노드가 토큰의 보유노드가 된다.



(그림 8) 다른 노드의 토큰을 보유한 경우(부하가 LWin 이하인 노드로 분산)

이후 노드의 부하가 LWin보다 작고 자신의 토큰 중 일부가 다른 노드에 존재하는 경우((그림 5)의 라인 23)에는 (그림 9)과 같이 자신의 토큰을 회수하게 된다. 이는 상대적으로 감소한 노드의 부하량을 LWin 값 이상으로 증가시키는 효과를 갖는다.



(그림 9) 토큰의 회수

(그림 7)과 (그림 8)의 경우에 토큰은 한 개씩 분산되며, (그림 6)와 (그림 9)의 경우에는 두 개씩의 토큰을 분산시킨다. 왜냐하면 전자의 경우에 노드의 부하가 어떤 토큰에 의한 것인지 명확하지 않기 때문에 토큰의 분산으로 인해 급격한 부하량의 변화가 발생할 수 있기 때문이다.

요청된 트랜잭션은 참조 데이터가 포함된 토큰을 보유하고 있는 노드로 분배되며, 이때 트랜잭션 분배기는 토큰 보유 노드가 상위 경계 이상의 부하를 보이면 이를 제안된 방법에 의해 다른 노드로 분산시킨다. 이런 기법은 캐싱 데이터의 참조 지역성을 높여 줄 수 있으며, 요청된 트랜잭션이 참조하는 데이터에 대한 로킹 정보를 가진 노드(PCA 노드)에 많은 부하량이 주어지는 경우에 부하를 효과적으로 분배할 수 있도록 할 수 있다. 트랜잭션 분배는 요청된 트랜잭션의 정보를 이용해 토큰을 보유하고 있는 노드를 찾고 토큰 보유 노드에 트랜잭션을 요청하도록 하면 비교적 간단하게 트랜잭션 처리 노드를 결정할 수 있다.

3.3 장 · 단점 분석

제안한 동적 트랜잭션 분배 알고리즘은 특정 트랜잭션 클래스가 폭주할 경우 PCA 기반의 정적인 트랜잭션 분배 알고리즘에 비해 과부하 노드의 부하를 다른 노드에게 동적으로 이전함으로써 노드들간의 부하 분산을 이룰 수 있다는 장점을 갖는다. 뿐만 아니라, 토큰이라는 개념을 도입하여 부하를 이전할 노드를 선택적으로 결정함으로써 부하 분산에 따른 캐쉬 무효화 효과를 줄일 수 있다는 장점을 갖는다. 이와는 달리 기존의 동적 트랜잭션 분배 알고리즘들은 과부하 노드의 부하를 모든 노드들에게 즉시 분산시킴으로써 과부하 노드의 트랜잭션들이 액세스하는 데이터들이 모든 노드에 중복되어 캐싱될 수 있고, 그 결과 빈번한 캐쉬 무효화 현상이 발생한다.

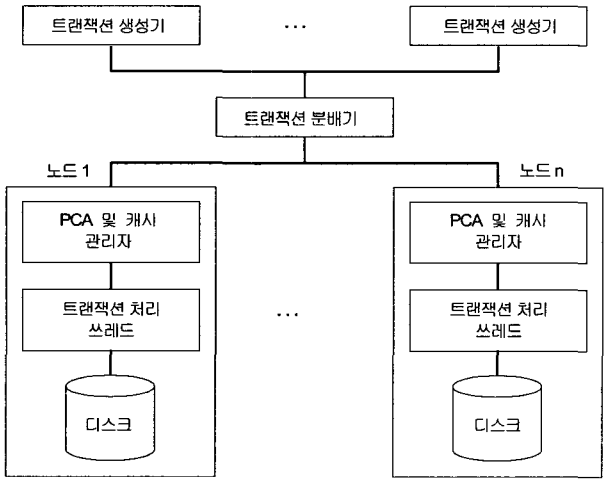
토큰 기반의 동적 트랜잭션 분배 알고리즘이 효과적으로 동작하기 위해서는 각 노드의 유휴 상태나 과부하 상태를 나타내는 LWin과 RWin을 정확하게 정의하여야 한다. 기본적으로 본 논문에서는 LWin과 RWin의 값을 사전에 파악할 수 있다고 가정하였다. 이를 위하여 단일 시스템 환경에서 각 노드별로 벤치마킹을 통하여 각각의 값을 구하는 과정이 필요하다. 또한 본 논문에서는 LWin과 Rwin을 노드에서 실행 중인 트랜잭션의 수로 표현하였는데, CPU나 디스크의 이용률, 혹은 트랜잭션 대기 큐의 길이 등을 기준으로 각 변수의 값을 정의하는 것도 가능하다. 그러나 이 경우에는 시스템 상태에 대한 동적인 감시 기능이 추가되어야 한다.

제안한 동적 트랜잭션 분배 알고리즘의 또 다른 단점은 토큰을 이용하여 다른 노드로 부하를 이전하더라도 이전된 트랜잭션이 액세스하는 데이터의 PCA는 과부하 노드에 고정되므로 로크 요청 및 응답 과정에서 메시지 전송이 발생한다는 것이다. 즉, 토큰은 캐쉬 일관성을 위한 페이지 전송 오버헤드는 줄일 수 있지만 로킹을 위한 메시지 전송은

줄어들지 않는다. 이를 해결하는 방법으로는 다음과 같은 두 가지를 들 수 있다. 첫 번째 방법은 [10, 13]에서 제안한 로크 보유 기법을 활용하는 것이다. 기본 개념은 토큰으로 인하여 트랜잭션 T가 다른 노드 N에게 이전될 경우, T가 액세스한 데이터에 대한 로크를 T의 완료 후에도 N이 계속 보유한다는 것이다. 이후 N에서 실행되는 다른 트랜잭션이 보유한 로크를 요청할 경우에는 PCA 노드와의 통신 없이 로크를 승인할 수 있다. 그러나 이 방법은 보유한 로크와 상충된 모드의 로크가 다른 노드에서 요청될 경우, PCA 노드가 이를 무효화하는 기능이 추가되어야 한다. 두 번째 방법은 Oracle 9i Real Application Cluster 등에서 제안한 방법으로 주기적으로 PCA를 재구성하는 것이다. 단, 동적으로 PCA를 재구성하기 위하여 로킹 정보나 대기 그래프 등이 노드들 간에 전송될 수 있으며, 경우에 따라서는 PCA를 재구성하는 동안 로킹 연산이 일시적으로 중단될 수 있다는 단점을 갖는다.

4. 성능 분석

제안된 트랜잭션 분배 알고리즘의 성능을 분석하기 위해 시뮬레이션 실험을 수행하였다. 시뮬레이션 모델은 SimJava [4]를 이용해 구현되었으며 (그림 10)와 같이 트랜잭션 생성기, 트랜잭션 분배기, 트랜잭션 처리 노드로 구성된다. 트랜잭션 생성기는 트랜잭션을 요청하는 노드를 표현한다. 트랜잭션 분배기는 요청된 트랜잭션을 분배 알고리즘에 의해 트랜잭션 처리 노드로 분배한다. 트랜잭션 처리 노드는 트랜잭션 처리의 기본 단위로서 분산 트랜잭션에서의 동기화를 위한 PCA 및 캐쉬 관리자, 트랜잭션 처리 쓰레드, 그리고 디스크로 구성된다. PCA 및 캐쉬 관리자는 자신이 저장하고 있는 레코드에 대한 PCA 정보와 캐쉬 정보를 관리한다. 트랜잭션 처리 쓰레드는 트랜잭션 처리의 병렬성을 위해 다수 개로 구성된다. 시뮬레이션을 위한 변수들은 <표 1>과 같다[7, 14].



(그림 10) 분산 트랜잭션 처리 시스템의 시뮬레이션 모델

〈표 1〉 시뮬레이션 변수

시스템 구성 변수		
LCPUSpeed	처리 노드의 CPU 속도	10 MIPS
NetBandWidth	네트워크의 데이터 전송 속도	100 Mbps
NumOfNodes	처리 노드의 수	5
NumTerms	시스템 전체의 터미널 수	10~700
NumDisk	공유 디스크의 수	5
MinDiskTime	최소 디스크 액세스 시간	10 ms
MaxDiskTime	최대 디스크 액세스 시간	30 ms
DBSizePerNode	각 노드에 할당된 PCA의 해당 DB 크기	20000 record
PageSize	각 페이지의 크기	4096 bytes
RecPerPage	한 페이지에 포함된 레코드 개수	20 record
CacheSize	캐쉬 버퍼의 크기	20% of DB size
트랜잭션 변수		
FixedMsgInst	메시지 처리를 위한 고정 명령수	20,000
PerPageMsgInst	메시지 길이 당 추가되는 명령수	10,000/page
ControlMsgSize	제어 메시지의 길이	256bytes
LockInst	로크 등록/해제를 위한 명령수	300
PerIOInst	디스크 IO를 위한 명령수	5000
CreationDelay	트랜잭션 생성 시 평균 대기 시간	1
제안된 알고리즘 구현을 위한 변수		
TokenSize	캐쉬 버퍼에 저장할 DB 크기	((1-0.2)×DBSizePerNode)/((NumOfNodes -1)×2)
LeftWindow	단일 노드의 성능 하한 경계 값	40 transactions
RightWindow	단일 노드의 성능 상한 경계 값	100 transactions

토큰 기반의 트랜잭션 분배 기법을 기존의 분배 기법과 비교하기 위하여 랜덤 분배 기법과 PCA에 의한 분배 기법(PCA), 그리고 PCA에 기반하지만 PCA 노드에 부하가 폭주할 때 캐쉬효과는 고려하지 않고 최소부하노드에게 분배하는 기법(PCA/최소부하)을 시뮬레이션 하였다. 랜덤 분배 기법은 트랜잭션이 실행할 노드를 무작위로 결정하는 방법으로 무작위 함수의 특성에 따라 각 노드에서 실행되는 트랜잭션의 수는 거의 동일하다. 이와는 달리 PCA에 의한 분배 기법은 트랜잭션이 액세스하는 데이터들의 유형에 따라 메이저 PCA 노드에게 해당 트랜잭션을 할당하는 정적인 분배 방식이다. PCA/최소부하 방식은 PCA 기반의 정적인 분배 방식과 동적 부하 분산 기법을 결합한 [3]의 알고리즘을 구현한 것이다. 시뮬레이션 실행은 요청되는 트랜잭션의 개수를 변화시켜 각각 트랜잭션 분배 기법에 대한 트랜잭션 처리율을 측정하였다. 실험은 4가지 트랜잭션 분배 기법에 대해 개별적으로 수행되었다.

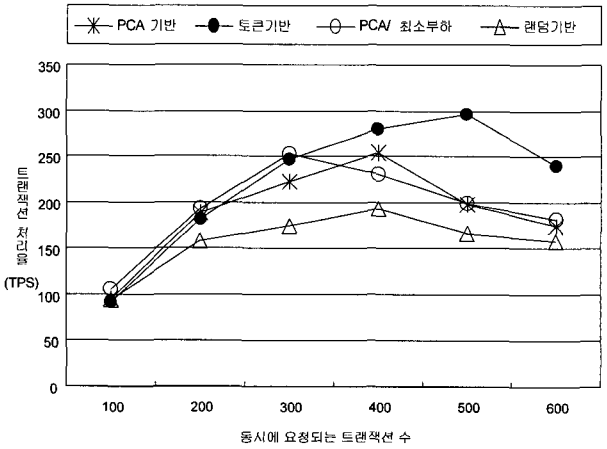
트랜잭션 분배 기법들간의 성능을 비교할 때 고려해야할 주요 사항은 분배 정책 자체에 대한 오버헤드를 분석하는 것이다. 랜덤 분배 기법이나 PCA에 의한 분배 기법은 정적인 분배 정책이므로 트랜잭션 라우터에 대한 추가적인 오버헤드가 거의 발생하지 않는다. 이와는 달리 PCA/최소부하 기법이나 본 논문에서 제안한 토큰 기반 트랜잭션 분배 기법의 경우에는 특정 노드의 과부하를 트랜잭션 라우터가 판단할 수 있어야 한다. 그러나 실험에서는 이를 특별히 고려하지 않았는데, 그 이유는 [3]에서 설명한 바와 같이 노드의 과부하 여부를 그 노드에서 실행되는 트랜잭션 수로써

판단할 수 있고 트랜잭션 라우터는 이를 쉽게 파악할 수 있기 때문이다. 즉, 모든 트랜잭션은 트랜잭션 라우터를 통하여 실행 노드에게 배정되므로 트랜잭션 라우터는 노드에게 단위 시간당 배정된 트랜잭션 수를 판단할 수 있으며, 이를 이용하여 과부하 노드를 결정할 수 있다.

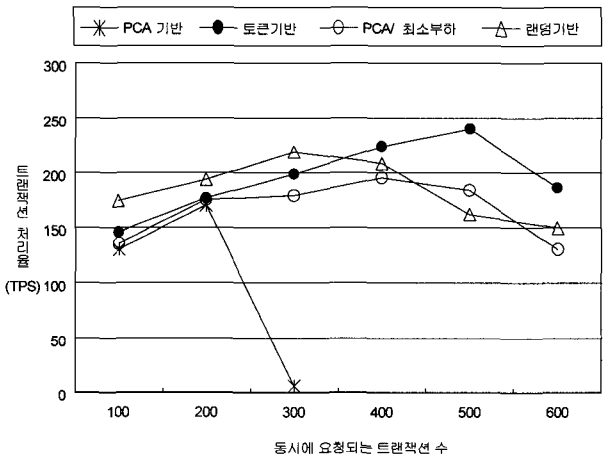
토큰 기반 트랜잭션 분배 기법의 경우 토큰 테이블을 관리하는 추가적인 오버헤드도 발생한다. 그러나 이 오버헤드도 성능에 큰 영향을 미치지 않을 것으로 판단되어 실험에서는 고려하지 않았다. 그 이유는 토큰 테이블에 대한 변경 연산은 특정 노드가 과부하가 되는 시점과 원상태로 복구하는 시점에서 각각 한번만 발생하기 때문이다. 뿐만 아니라, 토큰 테이블 변경 연산에서 토큰을 분산할 최소 부하 노드를 결정하기 위하여 전체 노드의 부하를 차례로 비교하는 과정이 발생하지만, 일반적으로 노드 수가 수십 개를 넘어가는 경우는 드물기 때문에 이 과정에서 소요되는 CPU 시간은 전체 성능에 큰 영향을 미치지는 않는다.

실험은 10만개의 레코드를 5개의 디스크에 2만개씩 분산시켜 저장한 후 이들에 대한 참조를 수행하는 트랜잭션을 생성시켜 수행했다. 이때 편중된 데이터 참조의 경우를 발생시키기 위해 10만개 중 4만개의 레코드만을 참조하는 트랜잭션을 발생시켜 각각의 트랜잭션 분배 기법의 성능을 비교했다. 또한 균등한 데이터 참조의 경우에는 10만개의 레코드를 임의로 참조하는 트랜잭션을 생성시켰다.

균등한 데이터 참조의 경우의 시뮬레이션 결과는 (그림 11)과 같다. 대부분의 동시요청 트랜잭션수의 범위에서 PCA 및 토큰기반 분배 기법들이 랜덤 분배 기법에 비해 높은 성능을 보임을 알 수 있다. 이는 랜덤 분배 기법과 비교할 때 PCA 및 토큰기반 분배 기법들이 데이터레코드의 캐싱효과를 이용할 수 있기 때문이다. 동시요청 트랜잭션 수가 많을수록 캐싱효과의 영향은 더 커지게 된다. 특히, 500개의 동시요청 트랜잭션의 경우 토큰기반 분배 기법이 랜덤기반 분배 기법에 비해서는 78%, PCA 기반 분배 기법에 비해서는 50%, PCA/최소부하기반 분배 기법에 비해서는 49%의 트랜잭션 처리율 향상을 보임을 알 수 있다. 반면, PCA/최소부하기반 분배 기법은 PCA 기반의 분배 기법과 비슷한 결과를 보였는데, 이는 동적 부하량을 고려하였지만, 데이터 레코드의 캐싱정보를 고려하지 않았기 때문으로 해석된다. 편중된 데이터 참조의 경우의 시뮬레이션 결과는 (그림 12)와 같다. PCA 기반 분배 기법은 2개의 트랜잭션 처리 노드에만 집중적으로 트랜잭션이 할당되므로 특정 처리노드에만 과도한 부하량이 발생하게 된다. 이 때 과부하 노드에서 정상적으로 처리할 수 있는 동시 요청 트랜잭션의 수(본 실험에서는 200개)를 넘어선 과부하에 대해서는 급격한 성능 저하를 보인다. 이는 본 시뮬레이션 실험 모델의 기반이 되는 PHLOX 시스템[7]에서 관측된 현상이며, 캐쉬사이즈를 넘어서는 데이터 레코드 처리시의 빈번한 디스크 I/O와 트랜잭션 처리 쓰레드의 빈번한 문맥교환이 발생하는 것이 주요한 원인으로 판단된다. 동시 요청 트랜잭션의 수가 적은 경우에는 랜덤 분배 기법, PCA/최소부하 기반 분배기법, 토큰 기반 분배 기법의 성능이 크게 차이가 나지 않지만, 트랜잭션 수가 많아질수록 제안된 토큰 기반 분배 기법이 우수한 성능을 보이는 것을 알 수 있다(500개의 동시요청 트랜잭션의 경우, 랜덤 기반 분배 기법에 비해서는 48%, PCA/최소부하 기반 분배 기법에 비해서는 30%의 성능향상을 보였다). 이는 토큰 기반 분배 기법이 캐쉬 이용률과 부하량 평준화 능력이 상대적으로 PCA/최소부하 기반 분배 기법과 랜덤 분배 기법보다 우수하기 때문이다.



(그림 11) 균등한 데이터 참조의 경우 트랜잭션 처리율



(그림 12) 편중된 데이터 참조의 경우 트랜잭션 처리율

5. 결 론

본 논문에서는 분산 트랜잭션 처리 시스템에 효과적으로 적용될 수 있는 트랜잭션 분배 기법을 제시하였다. 제안 기법의 기본적 원리는 캐쉬 효과를 극대화해 디스크에 대한 접근을 최소화하고 특정 노드의 과부하를 최소화하도록 해 전체 트랜잭션 처리 시스템의 성능을 향상시키고자 하는

것이다. 이를 위해 토큰이라는 캐싱 데이터의 흐름을 관리하는 데이터를 정의하여 토큰기반의 라우팅을 하게 된다. 토큰은 PCA보다 상위 개념으로서 PCA는 일단 한 노드에게 할당되면 다른 노드에게 양도할 수 없지만 토큰은 다른 노드에게 잠시 빌려줄 수 있는 캐싱권한이다. 제안기법은 기본적으로는 PCA 기반의 트랜잭션 라우팅기법을 이용한다. 즉 모든 노드들이 정상상태(또는 과부하가 아닌 상태)일 때는 친화도 기반 라우팅을 위해 PCA 노드에게 트랜잭션을 분배한다. 하지만 일단 PCA 노드가 과부하가 되면 토큰을 다른 노드에게 넘겨줌으로써 부하를 분배한다. 제안기법의 성능을 알아보기 위해 시뮬레이션을 수행하였으며, 수행 결과 기존의 기법에 비해 전체적으로 우수한 결과를 보여주었다. 특히 트랜잭션 부하량이 높은 경우와 편중된 데이터 참조를 보이는 경우 기존의 기법들보다 트랜잭션 처리율에서 각각 49%와 30%의 성능향상을 보였다.

참 고 문 헌

[1] M. Abdelguerfi and K. Wong (ed.), "Parallel Database Techniques," IEEE Computer Society Press, 1998.

[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] S. Haldar and D. K. Subramanian, "An Affinity-based Dynamic Load Balancing Protocol for Distributed Transaction Processing Systems," Performance Evaluation, Vol. 17, No.1, pp.53-71, 1993.

[4] F. Howell and R. McNab, "Simjava Package," http://www.dcs.ed.ac.uk/home/hase/simjava, April, 1999.

[5] L. Miller, A. Hurson and S. Pakzad (ed.), "Parallel Architectures for Data/Knowledge-Based Systems," IEEE Computer Society Press, 1995.

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

[7] K. Ohn, R. Hwang and H. Cho, "Prototyping PHLOX, A High Performance Transaction Processing System on a Workstation Cluster with Shared Disks," Proc. 8th IEEE Workshop on FTDCS, pp.67-73, 2001.

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

[9] E. Rahm, "A Framework for Workload Allocation in Distributed Transaction System", Syst. Soft. Journal, Vol.18, No.3, pp.171-190, 1992.

[10] 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.333-377, 1993.

[11] A. Reuter, "Load Control and Load Balancing in a Shared Database Management System," Proc. of 2nd Int. Conf. on Data Eng., pp.188-197, 1986.

[12] P. Yu and A. Dan, "Performance Analysis of Affinity Clustering on Transaction Processing Coupling Architecture," IEEE Trans. on Knowledge and Data Eng., Vol.6, No.5, pp.764-786, 1994.

[13] 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.

[14] M. Zaharioudakis, M. Carey and M. Franklin, "Adaptive, Fine-Grained Sharing in a Client-Server OODBMS : A Callback-Based Approach," ACM Trans. on Database Syst., Vol.22, No.4, pp.570-627, 1997.

김 기 형

e-mail : kkim@yu.ac.kr

1990년 한양대학교(공학사)

1992년 한국과학기술원(공학석사)

1996년 한국과학기술원(공학박사)

2001년 AdForce, Inc 선임연구원

1997년~현재 영남대학교 컴퓨터공학과 조교수

관심분야 : Ad Hoc Networks, Multicasting, Simulation, Embedded System

조 행 래

e-mail : hrcho@yu.ac.kr

1988년 서울대학교 컴퓨터공학과 학사

1990년 한국과학기술원 전산학과 석사

1995년 한국과학기술원 전산학과 박사

1995년~현재 영남대학교 전자정보공학부 부교수

관심분야 : 분산/병렬 데이타베이스, 트랜잭션 처리, DBMS 개발 등

남 영 환

e-mail : nicki@sicc.co.kr

1996년 경북대학교 고분자공학 학사

1999년 영남대학원 컴퓨터공학 석사

1999~현재 쌍용정보통신 IT 솔루션센터 시스템통합 컨설턴트

관심분야 : 트랜잭션 프로세싱, 차세대 시스템 통합기술, eAI, Web Service, 정보기술 아키텍처(ITA), C4ISR