공유 디스크 클러스터에서 친화도 기반 동적 트랜잭션 라우팅

(Affinity-based Dynamic Transaction Routing in a Shared Disk Cluster)

온 경 오 † 조 행 래 ††
(Kyungoh Ohn) (Haengrae Cho)

요 약 공유 디스크(Shared Disks: SD) 클러스터는 온라인 트랜잭션 처리를 위해 다수 개의 컴퓨터를 연동하는 방식으로, 각 노드들은 디스크 계층에서 데이타베이스를 공유한다. SD 클러스터에서 트랜잭션 라우팅은 사용자가 요청한 트랜잭션을 실행할 노드를 결정하는 것을 의미한다. 이때, 동일한 클래스에 속하는 트랜잭션들을 가급적 동일한 노드에서 실행시킴으로써 지역 참조성을 극대화할 수 있으며, 이를 친화도 기반 라우팅이라 한다. 그러나 친화도 기반 라우팅은 트랜잭션 클래스의 부하 변화에 적절히 대처할 수 없으며, 특정 트랜잭션 클래스가 폭주할 경우 해당 노드는 과부하 상태에 빠질 수 있다는 단점을 갖는다. 본 논문에서는 친화도 기반 라우팅을 지원하면서 SD 클러스터를 구성하는 노드들의 부하를 동적으로 분산할 수 있는 동적 트랜잭션 라우팅 기법을 제안한다. 제안한 기법은 지역 버퍼에 대한 참조 지역성을 높이고 버퍼 무효화 오버헤드를 줄임으로써 시스템의 성능을 향상시킬 수 있다.

키워드 : 트랜잭션 처리, 클러스터, 공유 디스크, 트랜잭션 라우팅, 캐쉬 일관성

Abstract A shared disk (SD) cluster couples multiple nodes for high performance transaction processing, and all the coupled nodes share a common database at the disk level. In the SD cluster, a transaction routing corresponds to select a node for an incoming transaction to be executed. An affinity-based routing can increase local buffer hit ratio of each node by clustering transactions referencing similar data to be executed on the same node. However, the affinity-based routing is very much non-adaptive to the changes in the system load, and thus a specific node will be overloaded if transactions in some class are congested. In this paper, we propose a dynamic transaction routing scheme that can achieve an optimal balance between affinity-based routing and dynamic load balancing of all the nodes in the SD cluster. The proposed scheme is novel in the sense that it can improve the system performance by increasing the local buffer hit ratio and reducing the buffer invalidation overhead.

Key words : transaction processing, cluster, shared disk, transaction routing, cache consistency

1. 서 론

클러스터는 다수 개의 연결된 노드들이 트랜잭션 처리를 위해 하나의 노드처럼 협력하는 시스템으로, 온라인 트랜잭션 처리와 전자 상거래, 그리고 병렬 데이타베이스 시스템 등의 다양한 분야에 적용되고 있다[1]. 클러스터의 구성방식은 데이타를 액세스하는 방법에 따라 모든 노드에서 디스크를 공유하는 방식(Shared Disk: SD)과 메모리나 디스크를 공유하지 않는 방식(Shared Nothing: SN)으로 분류할 수 있다. SN 클러스터에서 각 노드는 할당된 데이타베이스 분할에 대해서만 직접 액세스가 가능하다. 그러나 SD 클러스터에서는 각 노드들이 디스크 계층에서 전체 데이타베이스를 공유함으로써, 동적 부하 분산이 용이하고 새로운 노드의 추가로 인한 확장성이 향상된다는 장점을 가진다. 뿐만 아니라, SAN(storage area networks)이나 NAS(network attached storage) 기술의 발전으로 높은 가용성과 데이타 액세스의 융통성을 지원하는 SD 클러스터의 장점이 한

· 본 논문은 2002년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2002-002-D00151)
† 학생회원 : 영남대학교 컴퓨터공학과
ondal@yumail.ac.kr
†† 종신회원 : 영남대학교 전자정보공학부 부교수
hrcho@yu.ac.kr
논문접수 : 2003년 6월 4일
심사완료 : 2003년 8월 21일

층 부각되고 있다. 이런 관점에서 IBM과 Oracle에서는 SD 클러스터를 위한 병렬 데이타베이스 시스템으로 IBM DB2 Parallel Edition[2]과 Oracle 9*i* Real Application Cluster[3] 등의 제품을 출시하고 있다.

SD 클러스터에서 각각의 노드는 자신의 버퍼에 최근에 액세스한 데이타들을 캐싱한다. 데이타에 대한 효율적인 캐싱은 각 노드에서 발생하는 디스크 액세스 수나 노드들 간의 데이타 전송량을 줄임으로써 SD 클러스터의 성능을 크게 향상시킬 수 있다. 그러나 노드들이 최신의 데이타를 항상 사용할 수 있기 위해서는 버퍼에 캐싱된 데이타의 일관성이 유지되어야 한다. 이를 위해 각 노드에 존재하는 버퍼 관리자는 캐쉬 일관성 기법을 지원하여야 하는데, 기본 개념은 한 노드에서 특정 데이타를 갱신할 경우 그 데이타의 이전 버전을 캐싱하고 있는 다른 노드들의 버퍼에서 무효화를 시킨다는 것이다[4,5].

본 논문의 주제는 사용자가 요청한 트랜잭션을 실행할 노드를 결정하는 트랜잭션 라우팅에 관한 것이다. SD 클러스터의 경우, 트랜잭션들을 데이타베이스 액세스 유형에 따른 트랜잭션 클래스들로 분류하고, 각 트랜잭션 클래스마다 특정 노드를 할당한다. 즉, 임의의 트랜잭션 클래스 TC_i에 대해 이를 실행하는 노드 N_k를 정의하고, N_k를 TC_i의 "친화도 노드"라고 정의한다. 이후 트랜잭션이 요청되면 트랜잭션 라우터는 그 트랜잭션이 속한 클래스의 친화도 노드에게 트랜잭션을 할당하여 실행하도록 한다. 동일한 트랜잭션 클래스에 속하는 트랜잭션들은 데이타베이스의 특정 부분을 빈번히 액세스하므로, 라우팅의 결과 각 노드의 지역 버퍼 히트율이 높아지며 버퍼 무효화와 로크 동기화에 의한 노드들 사이의 간섭을 줄일 수 있다. 이러한 개념을 친화도 기반 라우팅(affinity-based routing)이라 한다[6-10]. 그러나 친화도 기반 라우팅은 각 노드의 부하를 고려하지 않기 때문에 각 노드들 사이의 부하 편차가 클 경우 시스템의 성능이 현저하게 감소한다는 단점을 갖는다. 즉, 특정 데이타베이스 분할을 액세스하는 트랜잭션 클래스의 부하가 폭주할 경우, 그 트랜잭션 클래스의 친화도 노드는 과부하 상태가 되지만 다른 노드들은 유휴 상태가 된다.

시스템의 부하가 동적으로 변할 때 친화도 기반 라우팅의 이러한 문제를 해결하기 위해 트랜잭션 클래스에 할당된 친화도 노드를 동적으로 변화시킬 수 있는 동적 부하 분산 기법이 함께 고려되어야 한다. 이러한 관점에서 본 논문에서는 친화도 기반의 동적 트랜잭션 라우팅 기법(Dynamic Affinity Cluster Allocation: DACA)을 제안한다. DACA는 트랜잭션 클래스와 노드간의 친화도 관계를 동적으로 변화시킴으로써 각 노드의 부하를 균등히 유지하면서, 그에 따르는 버퍼 무효화 효과를 최소화할 수 있다는 장점을 갖는다.

본 논문의 구성은 다음과 같다. 2절에서는 기존에 제안된 트랜잭션 라우팅 기법에 대해 살펴보고, 3절에서는 본 논문에서 제안한 동적 트랜잭션 라우팅 기법인 DACA를 설명한다. 4절에서는 성능 평가 모형을 설명하고, 5절에서 성능 평가 결과를 분석한다. 6절에서 결론을 맺는다.

2. 관련 연구

친화도 기반 라우팅의 문제를 해결하기 위하여 트랜잭션 클래스에 동적으로 노드를 할당하는 친화도 기반 동적 부하분산 분야는 SD 클러스터의 다른 연구 분야에 비해 상대적으로 관련 연구가 많지 않다. 대표적인 관련 연구로는 [6]과 [10]을 들 수 있는데, 두 연구의 공통점은 시스템의 부하가 균등히 분산될 경우에는 친화도 기반 라우팅을 수행하고, 특정 트랜잭션 클래스가 폭주할 경우에는 해당 트랜잭션들을 다른 노드로 분산한다는 것이다. 부하를 분산하는 방법에는 다소 차이가 있는데, [6]의 경우에는 현재 부하가 가장 낮은 노드에게 폭주 클래스의 트랜잭션을 할당하며, [10]의 경우에는 폭주 클래스의 트랜잭션들을 모든 노드에게 round-robin 방식으로 분산한다. 그러나 폭주 클래스를 제외한 나머지 트랜잭션 클래스들의 부하가 비슷할 경우에는 [6]에서 제안한 기법도 결국 [10]과 유사한 방식으로 동작한다.

[6]과 [10]에서 제안한 라우팅 기법들은 동적으로 부하를 분산할 수 있지만, 낮은 지역 버퍼 히트율과 노드들 간의 빈번한 버퍼 무효화로 인해 성능 개선에 한계가 있다. 이런 문제점의 원인으로 다음의 두 가지를 들 수 있다. 첫째, 과부하 노드(폭주 클래스의 친화도 노드)를 제외한 노드들은 두 개의 데이타베이스 분할을 액세스한다. 즉, 기존에 할당된 트랜잭션 클래스와 폭주 클래스의 데이타베이스 분할을 함께 액세스해야 하기 때문에 버퍼 경쟁이 증가하게 되고, 그 결과 노드들의 지역 버퍼 히트율이 떨어진다. 둘째, 폭주 클래스의 트랜잭션들이 여러 노드에서 동시에 실행되므로 특정 노드에서 페이지 갱신이 발생할 경우 이전 버전을 캐싱하고 있는 다른 노드들에서 버퍼 무효화가 발생한다. 폭주 클래스의 트랜잭션들을 실행하는 노드들의 수가 많아질수록 버퍼 무효화 빈도수는 커지며 지역 버퍼 히트율은 더욱 감소한다. 이러한 관점에서 폭주 클래스의 부하를 여러 노드에 균등히 배분하면서, 지역 버퍼 히트율의 감소나 과도한 버퍼 무효화 발생 현상을 피할 수 있는 새로운 트랜잭션 라우팅 정책이 요구된다.

최근 들어, Oracle 9i Real Application Cluster에서

친화도 기반 라우팅을 채택할 때 로킹 오버헤드를 최소화하기 위한 "동적 자원 재소유 기법"을 제안하였다[3]. 기본적인 개념은 각 데이타베이스 분할을 가장 빈번히 액세스하는 노드가 해당 분할의 로크 정보를 관리하여 그 분할에 대한 로크 연산을 지역적으로 처리함으로써 로킹 오버헤드를 줄일 수 있도록 한다는 것이다. 그러나 동적 자원 재소유 기법은 친화도 기반 트랜잭션 라우팅의 장점을 이용하는 부수적인 정책으로 새로운 트랜잭션 라우팅 정책은 아니다.

일반적인 분산 시스템에서의 동적 부하분산 알고리즘은 널리 연구되어온 분야이다[11]. 대부분의 알고리즘들은 노드의 부하를 균등히 배분하는 것을 주요 목적으로 하고 있으며, 캐싱과 관련한 친화도 기반 라우팅은 포함하지 않고 있다. 그러나 본 논문에서 가정하고 있는 SD 클러스터 기반 고성능 트랜잭션 처리 시스템에서는 트랜잭션 응답 시간이 CPU 처리 시간보다는 I/O 시간이나 로크 지연, 그리고 통신 지연과 같은 데이타베이스 관련 요소에 의해 결정되므로[9], 동적 부하분산과 친화도 기반 라우팅을 함께 고려하여야 한다. 분산 시스템의 특수한 예로서 공유 메모리를 갖는 다중 처리기 구조의 경우에는 캐쉬 친화도와 부하분산을 함께 고려한 알고리즘들이 제안되었다[12,13]. 기본 개념은 하나의 프로세스가 여러 처리기에 의해 교대로 실행될 경우 이전 처리기의 캐쉬 메모리에 남아있던 프로세스의 실행 내역들을 활용할 수 없으므로, CPU 스케줄링 단계에서 하나의 프로세스가 가급적이면 동일한 처리기에서 실행되도록 한다는 것이다. 그러나 프로세스가 동일한 처리기에서 실행될 수 없을 경우에는 단순히 유휴 상태인 다른 처리기에 할당된다는 관점에서 SD 클러스터에서 제안된 [6]의 라우팅 기법과 개념적으로 동일하다.

뿐만 아니라, 기존의 동적 부하분산 알고리즘[11,14]이나 실시간 의뢰자-서버 데이타베이스 환경의 부하 분산 알고리즘[15] 들의 경우, 실행 중인 프로세스를 다른 노드로 이전하거나 프로세스의 자원 사용 유형에 따라 하나의 프로세스를 여러 개의 서브 프로세스들로 분할하는 방법들을 이용한다. 그러나 본 논문에서는 라우팅 정책에 의해 특정 노드에 할당된 트랜잭션들은 실행 중에 다른 노드로 이전하는 것을 고려하지 않도록 한다. 그 이유는 [6]과 [7]에서 지적한 바와 같이, 실행 중인 트랜잭션을 이전하는 것은 많은 비용을 초래하며 구현도 복잡하기 때문이다. 뿐만 아니라, 사용자 트랜잭션을 초기에 분할하여 실행할 경우 트랜잭션의 원자성을 유지하기 위한 2단계 프로토콜이 필요하므로, 본 논문에서는 이를 고려하지 않도록 한다.

3. 친화도 기반 동적 트랜잭션 라우팅

본 절에서는 친화도 기반 동적 트랜잭션 라우팅 알고리즘인 DACA(Dynamic Affinity Cluster Allocation)를 제안한다. DACA의 목표는 두 가지로 요약할 수 있다. 첫 번째 목표는 버퍼 히트율이 급격하게 감소하지 않는 부하 분산을 수행한다는 것이다. 이를 위하여, 특정 트랜잭션 클래스가 폭주할 때 여러 개의 노드들을 폭주 클래스의 친화도 노드로 동적으로 정의할 수 있도록 하고, 새로운 친화도 노드에서 기존에 할당되어 있던 트랜잭션 클래스의 친화도 관계를 다른 노드로 변경한다. 그 결과 폭주 클래스의 친화도 노드들은 하나의 트랜잭션 클래스만 실행하므로 버퍼 히트율이 적정 수준을 유지할 수 있으며, 다른 노드들은 폭주 클래스의 트랜잭션들을 실행하지 않으므로 버퍼 무효화 현상도 줄일 수 있다. DACA의 두 번째 목표는 라우팅 오버헤드를 최소화한다는 것이다. 이를 위하여, 특정 데이타베이스 분할을 높은 확률로 액세스하는 여러 트랜잭션 클래스들을 하나의 친화도 클러스터(Affinity Cluster: AC)[8]로 정의한 후, AC들의 부하 분산을 고려한 라우팅 정책을 제안함으로써 라우팅 알고리즘의 복잡성을 단순화하도록 한다.

3.1 시스템 모델

표 1은 DACA의 라우팅 정책에 사용되는 매개변수들이다. 본 논문에서는 AC의 수(#AC)가 전체 노드의 수(#N)보다 작거나 같다고 가정한다. 그 이유는 [8]에서 지적한 바와 같이 트랜잭션 부하의 대부분은 소수의 트랜잭션 클래스들에 의해 발생하므로, AC의 수가 너무 많아질 경우 각 AC의 평균 부하를 유사한 수준으로 유지하도록 트랜잭션 클래스를 할당하는 것이 불가능하기 때문이다.

본 논문에서는 편의상 SD 클러스터에서 하나의 트랜잭션 라우터가 존재한다고 가정하고 있으며, 트랜잭션 라우터는 라우팅 매개변수들을 유지하여 사용자 트랜잭션들을 여러 노드에 할당한다. 즉, AC_i에 속하는 트랜잭

표 1 라우팅 매개변수

매개변수	설 명
#AC	AC의 수
#N	노드의 수 (#N ≥ #AC)
$Mem(N_p)$	노드 N_p의 메모리 크기
$HotSet(AC_q)$	AC_q에 의해 액세스되는 hot set의 크기
$\#T(N_p)$	N_p에서 실행되는 트랜잭션의 수
$\#T(AC_q)$	실행 중인 AC_q 트랜잭션의 수
$\mathbb{P}(AC_q)$	AC_q에 할당된 친화도 노드의 집합
$Card(\mathbb{P}(AC_q))$	AC_q에 할당된 친화도 노드의 수
$\mathbb{P}^{-1}(N_p)$	N_p를 친화도 노드로 하는 AC들의 집합
$Card(\mathbb{P}^{-1}(N_p))$	N_p를 친화도 노드로 하는 AC들의 수
$\overline{L}(N)$	노드의 평균 부하 $(\sum_{i=1}^{\#N}(\#T(N_i))/\#N)$

션이 트랜잭션 라우터에 의해 노드 N_j에 할당될 때 $\#T(AC_i)$와 $\#T(N_j)$값을 증가하고, 그 트랜잭션이 완료되어 라우터에 알려지면 이 값들을 감소한다. 복수 개의 트랜잭션 라우터를 두는 경우와 라우터와의 통신에 관련된 오버헤드는 3.3절에서 설명하도록 한다.

트랜잭션 라우터는 트랜잭션 라우팅을 위해 라우팅 함수($\mathbb{R}$)를 사용하며, $\mathbb{R}$은 각 AC에 할당된 친화도 노드들의 집합을 의미한다. $\mathbb{R}(AC_i)$가 여러 노드들을 포함할 경우, AC_i에 속하는 입력 트랜잭션들은 round-robin 방식으로 친화도 노드들에게 라우팅 된다. $\mathbb{R}^{-1}$은 $\mathbb{R}$의 역함수로 각 노드에 할당된 AC를 의미한다. #AC와 #N이 동일할 경우에는 $\mathbb{R}(AC_i) = \{N_i\}$로 초기화되며, #AC가 #N보다 작을 경우에는 시스템 부하에 많은 부분을 차지할 것으로 예상되는 AC에 대해서 더 많은 노드들을 할당함으로써 초기 설정을 최적화할 수 있다[8].

동적 라우팅을 위해 DACA는 과부하 종류를 'AC 과부하'와 '노드 과부하'로 구분한다. AC 과부하는 특정 AC의 트랜잭션들이 폭주하는 것을 의미하고, 노드 과부하는 특정 노드가 여러 개의 AC들에 할당되었을 때 노드에서 실행되는 트랜잭션 수가 평균을 초과한 상태를 의미한다. DACA는 AC와 노드의 상태에 따른 라우팅 정책을 수행하며, 라우팅 정책을 변경해야 할 AC와 노드의 상태에 대해서 다음과 같이 정의한다.

정의 1 (AC 과부하): $\#T(AC_q)/(Card(\mathbb{R}(AC_q)) + 1) \geq \overline{L}(N)$일 경우 AC_q는 AC 과부하 상태이다.

정의 2 (노드 과부하): $Card(\mathbb{R}^{-1}(N_p)) > 1$이고, $\#T(N_p) \geq \overline{L}(N) \times \alpha$ $(1 < \alpha \leq 2)$일 경우 N_p는 노드 과부하 상태이다.

정의 3 (AC 부하미달): $Card(\mathbb{R}(AC_q)) > 1$이고, $\#T(AC_q)/(Card(\mathbb{R}(AC_q)) - 1) < \overline{L}(N)$일 경우 AC_q는 AC 부하미달 상태이다.

AC 과부하는 AC_q의 친화도 노드 수를 1 증가하더라도 친화도 노드에 할당된 트랜잭션의 수가 전체 노드에 할당된 평균 트랜잭션 수보다 클 경우를 의미한다. 이와는 달리, 노드 과부하는 노드 N_p에 여러 개의 AC들이 할당되어 있고 현재 N_p에서 실행되는 트랜잭션의 수가 전체의 평균보다 일정 비율 이상일 경우를 의미한다. 노드 과부하의 정의에서 α는 민감도 인자(sensitivity factor)로서 라우팅 정책의 적용 빈도수를 결정하는데, 3.2.2절에서 노드 과부하에 대한 라우팅 정책과 연계한 α의 설정 방식에 대해 설명하도록 한다. AC 부하미달은 현재 AC_q의 친화도 노드 수를 1 감소하더라도 친화도 노드의 부하가 전체 노드의 평균 부하보다 작을 경우를 의미한다.

본 논문에서는 각 노드에서 동시에 실행되는 트랜잭션 수를 이용하여 AC 과부하와 노드 과부하를 정의하였다. 이와는 다른 기준으로 노드의 CPU 이용률이나 트랜잭션의 평균 응답시간, 혹은 평균 대기시간 등을 사용할 수 있으며, 여러 기준들을 혼합하여 사용하는 것도 가능하다. 그러나 [16]에서 지적한 바와 같이 단순한 부하 기준을 사용함으로써 거의 최적의 부하 분산 효과를 거둘 수 있으며, 정교하고 복잡한 부하 기준을 사용하더라도 성능 개선의 정도가 크지 않다. [16]의 연구에서는 동시에 실행되는 작업의 수를 부하 기준으로 삼는 것이 가장 최적임을 실험으로 증명하였으며, 이를 바탕으로 SD 클러스터를 위한 기존의 트랜잭션 라우팅 알고리즘들은 본 논문과 동일하게 동시에 실행되는 트랜잭션 수를 부하의 기준으로 가정하였다[6,10].

동시에 실행되는 트랜잭션 수를 부하 기준으로 정할 경우, 동시성 제어로 의한 '데이타 경쟁'(data contention) 현상이 무시될 수 있다는 문제점이 존재한다. 즉, 동시에 실행되는 트랜잭션 수는 동일하더라도 길이가 긴 트랜잭션이 존재하거나 특정 데이타에 대한 기록 연산이 많을 경우, 로킹에 의한 대기 트랜잭션의 수가 증가하며 빈번한 교착 상태 발생으로 인한 쓰래싱(thrashing) 현상이 발생한다. 데이타 경쟁 현상은 라우팅 정책에 의해 해당 트랜잭션들을 다른 노드로 분산하는 방식으로는 해결할 수 없다. 이를 해결하기 위해서는 입력 부하 제어 정책이 필요한데, 예로서 [17]의 경우 현재 실행 중인 트랜잭션 수에 대하여 대기 트랜잭션 수가 일정 비율을 넘을 경우 데이타 경쟁 상태로 간주하고 데이타 경쟁 현상이 해소될 때까지 해당 트랜잭션 클래스에 속하는 신규 트랜잭션들을 더 이상 입력받지 않는다. 입력 부하 제어 정책과 본 논문에서 제안한 DACA의 라우팅 정책과는 상호 독립적인 개념이지만, 실제 트랜잭션 라우터를 구현할 경우에는 이들을 모두 고려하여야 하며 이를 위하여 라우터는 대기 트랜잭션의 수를 추가로 파악하고 있어야 한다.

3.2 동적 트랜잭션 라우팅

DACA는 과부하 상태에 따라 각 노드의 부하를 분산시킨다. 만약 AC_q가 과부하라면, AC_q에 노드를 추가 할당하여 $\mathbb{R}(AC_q)$를 확장하며 이를 '노드 확장' 정책이라 한다. 만약 AC 과부하가 없고 노드 N_p가 과부하일 경우 'AC 분산' 정책을 수행하여 N_p에 할당된 일부 AC를 다른 노드로 분산시킨다. 마지막으로 AC_q가 부하미달이 되면 '노드 축소' 정책에 따라 AC_q에 할당된 노드를 축소한다. 본 절에서는 각각의 라우팅 정책에 대해 자세히 설명하도록 한다.

3.2.1 노드 확장

AC_q가 AC 과부하 상태이고 $\mathbb{R}(AC_q)$가 $\{N_q\}$라고 가정했을 때, AC_q의 새로운 트랜잭션을 N_q에 라우팅 한다

면 응답시간이 증가하므로 트랜잭션 라우터는 $\mathbb{R}(AC_q)$에 친화도 노드를 추가시켜 N_q의 부하를 분산시킨다. 이때 추가될 후보 노드로 SD 클러스터에서 부하가 가장 적은 노드를 선택한다. 최소 부하 노드를 N_k라고 하면 $\mathbb{R}(AC_q)$는 $\{N_q, N_k\}$로 확장되고, AC_q의 새로운 트랜잭션들은 round-robin 방식으로 N_q와 N_k에 라우팅 된다. 만약 N_k에 기존에 할당된 AC_k가 있을 경우, AC_k의 친화도 노드를 다른 노드로 변경한다. AC_k가 부하미달이면 3.2.3절에서 설명하는 노드 축소 정책에 의해 $\mathbb{R}(AC_k)$에서 N_k를 제외시키고 N_k를 AC_q에 할당한다. 만약 AC_k가 부하미달이 아니라면 3.2.2에서 설명하는 AC 분산 정책과 유사한 방식으로 AC_k의 친화도 노드를 변경한다. 결국 AC_k의 부하미달 여부에 관계없이 N_k는 AC_q에 전적으로 할당되고 AC_k는 다른 노드에 할당된다.

[예 1] 노드 확장 정책의 동작 과정을 이해하기 위하여 #AC와 #N이 각각 4인 SD 클러스터를 가정하자. 초기에 $\mathbb{R}(AC_i) = \{N_i\}$, $1 \le i \le 4$로 설정된다고 가정하며, 각 AC별로 50개의 트랜잭션들이 입력되어 실행된다고 가정하자. 즉, $\#T(AC_i) = \#T(N_i) = 50$, $1 \le i \le 4$이다. 만약 AC_1에 포함되는 트랜잭션들이 폭주하여 150개의 트랜잭션들이 입력되어 실행될 경우, 노드의 평균 부하 $\overline{L}(N) = (150 + 50 + 50 + 50) / 4 = 75$가 되고 AC_1에 대해 $150 / (1 + 1) = 75$이므로 AC_1은 AC 과부하 상태가 된다. N_1을 제외한 나머지 노드들의 부하 상태는 동일하므로 트랜잭션 라우터는 임의의 노드를 확장 후보 노드로 선택하는데, N_2가 후보 노드로 결정된다고 가정하자. 노드 확장의 결과 $\mathbb{R}(AC_1) = \{N_1, N_2\}$로 변경되며, $\mathbb{R}(AC_2) = \{N_3\}$ 또는 $\{N_4\}$로 변경 가능한데 N_3와 N_4의 현재 부하가 동일하므로 $\{N_3\}$로 변경된다고 가정하자. 이러한 부하 상태가 지속될 경우, 각 노드별 라우팅 정보와 부하 내용은 다음과 같다.

$\mathbb{R}^{-1}(N_1) = \{AC_1\}$, $\mathbb{R}^{-1}(N_2) = \{AC_1\}$, $\mathbb{R}^{-1}(N_3) = \{AC_2, AC_3\}$, $\mathbb{R}^{-1}(N_4) = \{AC^4\}$

$\#T(N_1) = \#T(N_2) = 75$, $\#T(N_3) = 100$, $\#T(N_4) = 50$

[예 1] 끝

DACA의 노드 확장 정책의 특징은 다음과 같이 두 가지로 요약할 수 있다. 첫째, 각 노드의 부하 편차가 일정 범위를 넘지 않는 한 과부하 AC의 친화도 노드 수를 최소로 유지한다. 이는 과부하 AC의 트랜잭션들을 모든 노드에게 즉시 할당하는 기존의 라우팅 정책과는 구분되며, 과부하 AC의 친화도 노드들 간에 버퍼 무효화 발생 확률을 최대한 줄일 수 있다는 장점을 갖는다. 동일한 데이타베이스 액세스 패턴을 갖는 트랜잭션들이 상대적으로 많은 노드에서 실행될 경우 각 노드의 지역 버퍼는 비슷한 페이지들을 캐싱하므로 버퍼 무효화 현상이 빈번해진다. 그 결과 트랜잭션 처리율이 노드 수에 선형적으로 비례하지 않음은 여러 논문에서 지적된 바 있다[4,8].

둘째, 하나의 노드에 여러 개의 AC가 할당되는 경우를 줄이도록 한다. 특히, 과부하 AC의 친화도 노드들에게는 동일한 AC의 트랜잭션들만 할당함으로써 지역 버퍼 히트율을 높일 수 있다는 장점을 갖는다. 상대적으로 부하가 작은 AC들은 하나의 노드에서 함께 실행될 수 있지만, 과부하 AC의 트랜잭션들을 효율적으로 처리하는 것이 전체 트랜잭션 처리율을 향상시키는데 보다 중요한 요인이 된다. 그러나 하나의 노드에 할당된 여러 AC들의 부하가 증가할 경우 그 노드는 노드 과부하 상태가 될 수 있고, 이때 다음 절에서 설명할 'AC 분산' 정책이 수행되어야 한다.

3.2.2 AC 분산

AC 분산 정책은 노드 N_p가 노드 과부하 상태일 때 수행된다. $\mathbb{R}^{-1}(N_p)$에 속하는 AC들 중에서 $\#T(AC_s)$가 가장 작고 $\#T(AC_l)$이 가장 크다고 가정하자. 만약 AC가 할당되지 않은 노드 N_k가 존재한다면 ($\mathbb{R}^{-1}(N_k) = \varnothing$), 트랜잭션 라우터는 $\mathbb{R}(AC_l)$을 $\{N_k\}$로 변경한다. 이와는 달리 $\mathbb{R}^{-1}(N_k) = \{AC_k\}$이며 AC_k가 AC 부하미달인 경우에도 AC_k에 대해 3.2.3 절에서 설명하는 노드 축소 정책을 수행한 후, $\mathbb{R}(AC_l)$을 $\{N_k\}$로 변경한다.

모든 노드에 AC가 할당되어 있고 부하미달인 AC도 존재하지 않을 경우, 트랜잭션 라우터는 과부하가 아닌 노드 N_k를 선택하여 AC_s를 할당한다. 만약 $\mathbb{R}^{-1}(N_k) = \{AC_k\}$일 경우, AC_s를 N_k에 추가 할당함으로써 N_k는 AC_k와 AC_s가 액세스하는 페이지들을 지역 버퍼에 모두 캐싱하게 된다. 이때 주의할 점은 서로 다른 데이타베이스 분할을 액세스하는 AC들이 하나의 노드에서 실행될 경우, 지역 버퍼가 두 AC들의 hot set을 캐싱할 만큼 크지 않다면 빈번한 캐쉬 미스가 발생할 수 있다는 것이다. 따라서 캐쉬 미스를 줄이기 위해 AC_s는 $HotSet(AC_k) + HotSet(AC_s) < Mem(N_k)$를 만족하는 N_k에 할당되어야 한다. 만약 이것을 만족하는 노드가 두 개 이상 존재한다면 트랜잭션 라우터는 그들 중 최소 부하 노드에 AC_s를 할당한다. 조건을 만족하는 노드가 존재하지 않는다면, 전체 노드 중 최소 부하 노드에 AC_s를 할당한다.

[예 2] 3.2.1절의 [예 1]에서 N_3에는 AC_2와 AC_3가 함께 할당되었는데, AC_2의 입력 트랜잭션 수가 증가하는 경우를 고려하자. 노드 과부하의 민감도 인자 α가 1.6이라고 가정하고, $\#T(AC_2)$의 값이 50에서 84로 증가하였으며 나머지 AC들의 실행 트랜잭션 수는 동일하다고 가정하자. 그 결과, $\#T(N_3)$의 값은 134로 증가하는데, 노드 평균 부하 $\overline{L}(N)$이 83.5((75+75+134+50)/4)이므로 N_3는 노드 과부하 상태에 빠지게 된다($134 > 83.5$

* 1.6). N_3를 제외한 나머지 노드들은 모두 AC가 할당되어 있으며 AC_1도 부하미달이 아니므로, N_3에 할당된 AC_2와 AC_3 중에서 부하가 적은 AC_3를 최소 부하 노드인 N_4로 할당하게 된다. AC 분산의 결과, 각 노드별 라우팅 정보와 부하 내용은 다음과 같이 변경된다.

$\mathbb{R}^{-1}(N_1) = \{AC_1\}$, $\mathbb{R}^{-1}(N_2) = \{AC_1\}$, $\mathbb{R}^{-1}(N_3) = \{AC^2\}$, $\mathbb{R}^{-1}(N_4) = \{AC_3, AC_4\}$

$\#T(N_1) = \#T(N_2) = 75$, $\#T(N_3) = 84$, $\#T(N_4) = 100$

[예 2] 끝

AC 분산 정책의 적용 빈도수는 노드 과부하의 민감도 인자 α에 의해 결정된다. α 값이 1과 거의 동일하다면 노드 부하가 평균 부하보다 조금만 클 경우에도 과부하로 판정하므로 AC 분산 정책이 빈번히 수행될 가능성이 존재한다. 특히 AC 분산 정책에 의해 과부하 노드 N_p에 할당된 AC_s의 친화도 노드를 N_k로 변경한 결과 N_k가 다시 노드 과부하 상태로 되는 문제점이 발생할 수 있다. 이러한 문제점을 해결하기 위하여 α 값을 1보다 충분히 큰 값으로 설정함으로써 AC 분산 정책의 적용 빈도수를 줄이는 것이 가능하다. 그러나 α 값이 너무 클 경우에는 노드간의 부하 편차가 커지는 상태가 오랫동안 지속되고, 일반적으로 α 값을 2이상으로 설정하는 것은 AC 분산 정책의 적용 빈도수를 줄인다는 관점에서도 큰 효과가 없다. 그 이유는 과부하 노드 N_p에 할당된 AC중에서 최소 부하를 갖는 AC(AC_s)를 다른 노드 N_k에게 할당하며, N_k는 다른 노드에 비해 부하가 상대적으로 작기 때문에 AC_s의 할당 결과로 N_k의 부하가 전체 부하 평균의 2배를 초과할 가능성이 매우 작기 때문이다. 결론적으로 노드간의 부하 편차가 크지 않아 AC 분산의 결과로 새로운 과부하 노드가 발생할 가능성이 클 경우에는 α 값을 2에 근접하게 설정하고, [예 2]와 같이 노드간의 부하 편차가 상대적으로 클 경우에는 α 값을 1과 2사이의 임의의 값으로 설정함으로써 AC 분산 정책의 적용 빈도수를 조절할 수 있다.

3.2.3 노드 축소

AC_q가 부하미달이고 $\mathbb{R}(AC_q)$가 $\{N_k, N_q\}$라고 가정하자. 만약 다른 AC에서 AC 과부하가 발생하거나 다른 노드에서 노드 과부하가 일어날 경우, 트랜잭션 라우터는 $\mathbb{R}(AC_q)$에서 제외될 노드를 임의로 선택한 후, 과부하 노드나 AC의 부하를 줄이기 위해 선택된 노드를 할당한다. 노드 축소 정책은 노드 확장 정책과 동일하게 AC 과부하가 발생하지 않는 최소 노드를 AC에 할당함으로써 AC의 친화도 노드들 간에 버퍼 무효화 발생 확률을 최대한 줄일 수 있도록 한다.

[예 3] 3.2.2절의 [예 2]와 같이 라우팅 정책을 변경한 후 $\#T(AC_1)$이 50으로 줄어든 경우를 가정하자. 그 결과 $\#T(N_1) = \#T(N_2) = 25$로 수정되고 $\overline{L}(N)$은 58.5((25+25+84+100)/4)로 변경된다. $\#T(AC1) < \overline{L}(N)$이므로 AC_1은 부하미달 상태가 된다. 한편 $\#T(N_4)$는 100으로 $\overline{L}(N) * \alpha$의 값인 93.6(58.5 * 1.6) 보다 크므로 N_4는 노드 과부하 상태가 된다. 따라서 3.2.2절의 AC 분산 정책에 의해 $\mathbb{R}(AC_1)$에서 N_1이나 N_2를 삭제한 후, $\#T(AC_3) = \#T(AC_4)$이므로 삭제한 노드를 AC_3나 AC_4에 할당한다. 만약 N_2를 삭제하여 AC_3에 할당하였다면, 노드 축소와 AC 분산의 결과 각 노드별 라우팅 정보와 부하 내용은 다음과 같이 변경된다.

$\mathbb{R}^{-1}(N_1) = \{AC_1\}$, $\mathbb{R}^{-1}(N_2) = \{AC_3\}$, $\mathbb{R}^{-1}(N_3) = \{AC_2\}$, $\mathbb{R}^{-1}(N_4) = \{AC_4\}$

$\#T(N_1) = 50$, $\#T(N_2) = 50$, $\#T(N_3) = 84$, $\#T(N_4) = 50$

[예 3] 끝

3.2.4 알고리즘

그림 1에 DACA의 트랜잭션 라우팅 알고리즘을 요약하여 정리한다. 이 알고리즘은 신규로 사용자 트랜잭션이 입력될 때마다 호출되며, 사용자 트랜잭션이 속한 AC의 식별자를 입력 인자로 받아들인다. 그리고 사용자 트랜잭션이 실행될 노드를 결정한 후 그 노드의 식별자를 반환한다.

AC_i가 입력 AC일 경우, 먼저 AC 과부하 여부를 조사한다(단계 2). AC 과부하일 경우에는 현재 AC가 할당되지 않은 노드나 혹은 부하미달인 AC에 할당된 노드를 AC_i에 추가로 할당한다. 그러한 노드가 존재하지 않을 때에는 최소 부하인 노드를 선택하여 그 노드에 할당된 AC들을 다른 노드로 이전한 후 AC_i의 친화도 노드로 결정한다. AC 과부하가 아닐 경우에는 AC_i의 친화도 노드(N_p)에 대해 노드 과부하 여부를 다시 조사한다(단계 3). 노드 과부하일 경우 N_p의 부하를 이전할 노드를 선택한 후, N_p에 할당된 AC중에서 일부를 이전한다.

단계 2.c에서 AC_Transfer(N_k) 함수는 기존에 N_k에 할당된 AC($\mathbb{R}^{-1}(N_k)$)들을 다른 노드로 이전하는 역할을 담당하는데, 동작과정은 단계 4.e와 유사하다. 즉, $\mathbb{R}^{-1}(N_k)$의 AC들에 대한 hot set의 합을 모두 포함할 수 있는 노드 중에서 부하가 최소인 노드를 선택한 후, 그 노드를 $\mathbb{R}^{-1}(N_k)$의 AC들에 대한 친화도 노드로 변경한다. 만약 hot set을 모두 포함할 수 있는 노드가 존재하지 않을 경우에는 최소 부하 노드를 친화도 노드로 변경한다.

트랜잭션이 완료할 경우에는 그 트랜잭션이 속한 AC와 친화도 노드의 #T 값을 1씩 감소하여야 한다. 그 결과 AC 부하미달이 발생할 수 있으며, 이를 해결하기 위하여 그림 1의 단계 2.b나 4.d와 같이 노드 축소 정책을 수행하여야 한다.

TRANSACTION_ROUTING(AC_i)

1. $\#T(AC_i) = \#T(AC_i) + 1$;
2. **IF** $(\#T(AC_i)/(Card(\mathbb{R}(AC_i)) + 1) \geq \overline{L}(N))$ **// AC_i가 AC 과부하**
   a. **IF** $(\exists N_k, Card(\mathbb{R}^{-1}(N_k)) = 0)$ **THEN** $\mathbb{R}(AC_i) = \mathbb{R}(AC_i) \cup \{N_k\}$;
   b. **ELSE IF** $(\exists AC_q, Card(\mathbb{R}(AC_q)) > 1$ and $\#T(AC_q)/(Card(\mathbb{R}(AC_q)) - 1) < \overline{L}(N))$
      - Select $N_k \in \mathbb{R}(AC_q)$; // AC_q는 AC 부하미달이므로 N_k를 $\mathbb{R}(AC_q)$에서 $\mathbb{R}(AC_i)$로 변경
      - $\mathbb{R}(AC_q) = \mathbb{R}(AC_q) - \{N_k\}$; $\mathbb{R}(AC_i) = \mathbb{R}(AC_i) \cup \{N_k\}$;
   c. **ELSE**
      - Select N_k, where $\#T(N_k)$ is minimum for all nodes; // 최소 부하 노드인 N_k를 선택
      - AC_Transfer(N_k); // 기존에 N_k에 할당된 AC들을 다른 노드로 이전
      - $\mathbb{R}(AC_i) = \mathbb{R}(AC_i) \cup \{N_k\}$;
3. N_p = next node of $\mathbb{R}(AC_i)$ in round-robin order; // N_p: AC_i의 트랜잭션을 할당할 후보 노드
4. **IF** $(Card(\mathbb{R}^{-1}(N_p)) > 1$ and $\#T(N_p) \geq \overline{L}(N) \times \alpha)$ **// N_p가 노드 과부하**
   a. Select AC_l, where $\#T(AC_l)$ is maximum for all ACs $\in \mathbb{R}^{-1}(N_p)$
   b. Select AC_s, where $\#T(AC_s)$ is minimum for all ACs $\in \mathbb{R}^{-1}(N_p)$
   c. **IF** $(\exists N_k, Card(\mathbb{R}^{-1}(N_k)) = 0)$ **THEN** $\mathbb{R}(AC_l) = \{N_k\}$
   d. **ELSE IF** $(\exists AC_q, Card(\mathbb{R}(AC_q)) > 1$ and $\#T(AC_q)/(Card(\mathbb{R}(AC_q)) - 1) < \overline{L}(N))$
      - Select $N_k \in \mathbb{R}(AC_q)$; // AC_q는 AC 부하미달이므로 N_k를 $\mathbb{R}(AC_q)$에서 $\mathbb{R}(AC_l)$로 변경
      - $\mathbb{R}(AC_q) = \mathbb{R}(AC_q) - \{N_k\}$; $\mathbb{R}(AC_l) = \{N_k\}$;
   e. **ELSE**
      - Select N_k, where $HotSet(AC_s) + HotSet(\mathbb{R}^{-1}(N_k)) \leq Mem(N_k)$ and $\#T(N_k)$ is minimum
      - **IF** (there exists such N_k) **THEN** $\mathbb{R}(AC_s) = \{N_k\}$
      - **ELSE** select N_k, where $\#T(N_k)$ is minimum and set $\mathbb{R}(AC_s) = \{N_k\}$.
   f. Recalculate N_p if $\mathbb{R}(AC_i)$ is changed.
5. $\#T(N_p) = \#T(N_p) + 1$; Retrun N_p

그림 1 DACA의 트랜잭션 라우팅 알고리즘

3.3 오버헤드 분석

AC와 노드의 부하 상태는 현재 실행중인 트랜잭션의 수로 표현되므로, 트랜잭션 라우터는 트랜잭션이 입력될 때나 완료할 때 해당 트랜잭션의 #T(AC)와 #T(N) 값을 변경하여야 한다. 대부분의 OLTP 시스템이나 SD 클러스터에서 채택하고 있는 것과 같이 전체 시스템에 하나의 트랜잭션 라우터만 존재할 경우에는 이 과정에서 큰 오버헤드가 발생하지 않는다. 즉, 모든 트랜잭션들은 트랜잭션 라우터를 통하여 입력되므로 트랜잭션 라우터는 모든 AC와 노드들의 실행 트랜잭션 수를 쉽게 파악할 수 있다. 뿐만 아니라, 트랜잭션 라우터가 각 트랜잭션 클래스에 대한 평균 응답 시간을 계산한 후 이를 이용하여 #T(AC)와 #T(N) 값을 자동적으로 감소한다면 트랜잭션 완료시 발생하는 트랜잭션 라우터와의 통신 오버헤드를 제거할 수 있다. 단, 평균 응답 시간과 실제 실행 시간과의 차이는 노드의 부하 상태에 따라 달라질 수 있으므로 주기적으로 그 차이를 보정해주어야 한다[7].

트랜잭션 라우터가 여러 개 존재할 경우에는 DACA 알고리즘을 다소 수정하여야 한다. 그 이유는 #T(AC)와 #T(N) 값을 모든 라우터들이 공유하기 위해서는 이를 분산 공유 메모리나 공유 디스크에 저장할 수밖에 없는데, 이 과정에서 동기화 오버헤드나 디스크 I/O 오버헤드가 과도하게 발생하기 때문이다. 제안한 알고리즘의 수정 방법은 기본적으로 다음과 같다. 먼저 친화도 라우팅 정책에 따라 각 AC를 실행할 노드를 정적으로 결정한 후, 라우팅 함수($\mathbb{R}$)를 모든 트랜잭션 라우터들이 공유한다. 이후 각 트랜잭션 라우터는 $\mathbb{R}$에 따라 입력 트랜잭션들을 각 노드에 할당하고 #T(AC)와 #T(N) 값을 독자적으로 유지한다. 특정 트랜잭션 라우터는 주기적으로 다른 라우터들의 #T(AC)와 #T(N) 값을 수집하여, 과부하 상태의 AC와 노드를 판단한 후 3.2.4절의 알고리즘에 따라 $\mathbb{R}$을 변경하고 이를 다른 라우터들에게 다시 전송한다. 라우팅 정보 수집의 주기가 짧을수록 부하 변경에 따른 빠른 대처가 가능하지만, 정보 수집에서 발생하는 트랜잭션 라우터들 간의 통신 오버헤드는 증가한다.

4. 성능 평가 모형

본 절에서는 제안한 DACA와 기존의 라우팅 기법들의 성능을 평가하기 위한 모의실험 모형을 설명한다. SD 클러스터의 성능 평가 모형은 그림 2에 나타나며, 이산-사건 실험 프로그램인 CSIM 언어[18]를 이용하여 구현하였다.

SD 클러스터는 하나의 라우터와 전역 로크 관리자(Global Lock Manager: GLM), 그리고 네트워크를 통

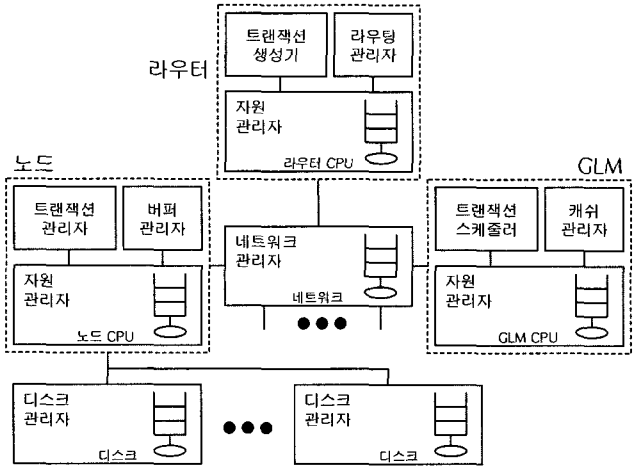

그림 2 SD 클러스터의 성능 평가 모형

해 연결된 여러 개의 노드로 구성되고, 각 노드는 모든 디스크들을 공유한다. 라우터는 트랜잭션 생성기와 라우팅 관리자로 구성된다. 트랜잭션 생성기는 트랜잭션을 생성하는 역할을 담당하고, 라우팅 관리자는 라우팅 기법을 수행한다. 각 노드는 *LRU* 버퍼를 관리하는 버퍼 관리자와 자원 관리자를 가진다. 트랜잭션 관리자는 각 트랜잭션에 대해 GLM에 로크 요청 메시지와 트랜잭션 완료 메시지를 전송한다.

GLM은 트랜잭션 스케줄러와 캐쉬 관리자를 가지고 각 노드에서 요청한 로크 관리 및 캐쉬 일관성 기법을 수행한다. 트랜잭션 스케줄러는 레코드 단위의 2 단계 로킹 기법을 지원한다. 그리고 대기 그래프를 사용하여 교착상태를 감지하며, 교착상태가 발생했을 경우 로크를 요청한 트랜잭션은 철회된다. 캐쉬 관리자는 SD 클러스터에서 대표적인 캐쉬 일관성 기법인 ARIES/SD 기법을 수행한다[5]. 본 논문에서는 GLM과 트랜잭션 라우터가 전체 시스템에 하나만 존재한다고 가정하며, 전체 시스템의 병목 현상을 방지하기 위하여 GLM은 다른 노드들에 비해 성능이 우수하다고 가정한다. 동일한 이유로 라우터의 CPU 성능도 GLM의 CPU 성능과 같다고 가정한다.

본 논문에서 시스템 구성 변수와 오버헤드를 표 2에서 요약한다. 각 매개 변수의 구체적인 값들은 [4]와 [10]에서 주로 참조하였다.

디스크 수는 10개이며 각 디스크는 FIFO 큐를 이용하여 I/O 요청을 들어온 순서대로 처리한다. 디스크 액세스 시간은 0.01초와 0.03초 사이의 일양 분포를 따른다고 가정한다. 데이타베이스는 여러 개의 논리적인 클러스터로 구성되며, 하나의 데이타베이스 클러스터는 10000개의 페이지(40 Mbytes)를 가진다. 하나의 AC는 하나의 데이타베이스 클러스터와 연관되며, AC의 수는 1 또는 8로 설정하였다. ACLocality는 트랜잭션이 연관된 데이타베이스 클러스터에서 참조하는 데이타의 비율을 의미한다. 각 데이타베이스 클러스터는 20%의 hot set(HotSize)을 가지고 연관된 AC 트랜잭션들의 80% (HotProb)가 이를 액세스한다. 각 트랜잭션에 의해 액

표 2 입력 매개 변수

시스템 구성 변수		
LCPUSpeed	노드 CPU의 속도	50 MIPS
GCPUSpeed	GLM CPU의 속도	100 MIPS
NetBandwidth	네트워크의 데이타 전송 속도	100 Mbps
NumNode	노드의 수	1 ~ 16
NumDisk	공유 디스크의 수	10
MinDiskTime	최소 디스크 액세스 시간	0.01 초
MaxDiskTime	최대 디스크 액세스 시간	0.03 초
PageSize	페이지 크기	4096 bytes
RecPerPage	페이지당 레코드 수	10
ClusterSize	클러스터의 페이지 수	10000
HotSize	클러스터의 hot set의 페이지 수	2000
DBSize	데이타베이스의 클러스터 수	8
BufSize	노드의 버퍼 크기(페이지 수)	4000
MsgInst	메시지당 명령 수	22000
LockInst	로크 획득/해제를 위한 명령 수	2000
PerIOInst	디스크 I/O를 위한 명령 수	5000
PerObjInst	레코드 처리를 위한 명령 수	15000
LogIOTime	로그 레코드 기록 시간	0.005 초
α	노드 과부하의 민감도 인자	2.0
트랜잭션 변수		
TrxSize	트랜잭션이 액세스하는 레코드 수	10
SizeDev	트랜잭션 크기 편차	0.1
WriteOpPct	레코드 갱신 비율	0.3
MPL	동시에 실행되는 트랜잭션 수	10 ~ 640
ACNum	AC의 수	1, 8
ACLocality	지역 클러스터를 액세스할 확률	0.8
HotProb	hot set을 액세스할 확률	0.8

세스되는 평균 레코드의 수는 TrxSize ± TrxSize × SizeDev 사이의 일양 분포를 따른다. WriteOpPct는 갱신되는 레코드의 비율을 의미하며 0.3으로 설정하였다. 데이타 경쟁으로 인한 쓰래싱 현상이 발생하지 않도록 하기 위하여 각 페이지마다 10개의 레코드를 저장한다고 가정하였으며 레코드 단위의 로킹을 구현하였다. 각 레코드를 처리하는 데는 15000개의 명령어(PerObjInst)가 실행된다고 가정하였다. 그리고 트랜잭션이 완료할 때마다 노드는 트랜잭션 라우터에게 트랜잭션 완료를 통보한다.

네트워크 관리자는 100Mbps의 대역폭을 갖는 FIFO 서버로 구현하였다. 네트워크를 통해 메시지를 전송하는 과정을 표현하기 위해 노드의 CPU 및 GLM의 CPU는 메시지마다 고정된 명령 수(MsgInst)를 실행한다. 모의실험에서 사용한 주요 성능 평가 지수는 트랜잭션 처리율이다. 트랜잭션 처리율은 단위 시간동안 완료한 트랜잭션의 수에 의해 측정된다. 그리고 보조 성능 지수로 트랜잭션을 실행하기 위해 요청된 데이타를 지역 버퍼나 혹은 다른 노드의 버퍼에서 발견할 확률을 나타내는 버퍼 히트율을 사용하도록 한다.

5. 실험결과 분석

본 절에서는 SD 클러스터에서의 동적 라우팅 기법들의 성능 평가 결과를 분석한다. 비교한 기법들은 DACA와 기존의 순수한 친화도 기반 라우팅(PAR), 그리고 폭주하는 AC에 SD 클러스터의 모든 노드를 round-robin 방식으로 할당하는 동적 친화도 기반 라우팅(DRR) 기법[10]이다. 먼저, 단일 AC에서의 성능 특성을 살펴보고 세 기법의 성능을 분석하도록 한다.

5.1 단일 AC의 성능

실험 모형의 정확성을 검증하기 위하여 노드의 수(NumNode)와 HotSize, 그리고 데이타베이스 클러스터의 크기(ClusterSize)를 변화할 경우 단일 AC의 성능 차이를 실험하였다. 그림 3은 실험결과를 나타낸다. 실선은 HotSize와 ClusterSize를 표 2의 값으로 설정하고 NumNode를 1부터 16까지 변화시키며 측정한 트랜잭션 처리율이다. 트랜잭션 라우터는 입력 트랜잭션들을 여러 노드들에게 round-robin 방식으로 할당한다. 점선은 NumNode를 1로 고정시키고, HotSize와 ClusterSize를 각각 2배와 3배로 증가하였을 때의 성능을 나타낸다. 이것은 여러 개의 AC들이 하나의 노드에 할당된 경우와 동일한 의미이다.

하나의 AC에 많은 노드를 할당할 경우 그림 3과 같이 성능개선 효과가 명확하게 나타난다. 그 이유는 그림 4에서 나타난 것과 같이 버퍼 히트율이 증가하기 때문이다. NumNode가 1이고 HotSize가 2000일 때, 버퍼

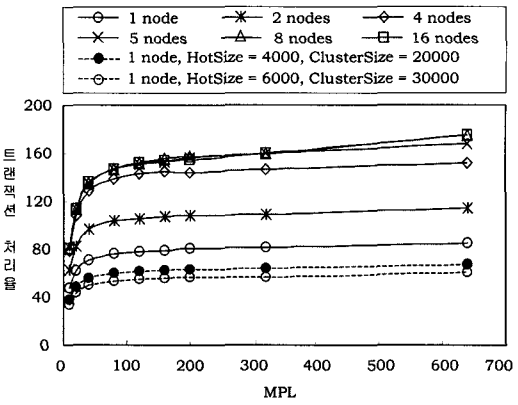

그림 3 단일 AC의 트랜잭션 처리율

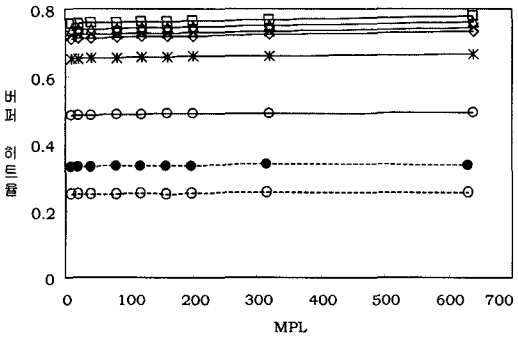

그림 4 단일 AC의 버퍼 히트율

히트율은 약 0.5이다. ACLocality와 HotProb가 모두 0.8이므로, AC의 트랜잭션들이 hot set을 액세스할 확률은 0.64이다. 따라서 버퍼 히트율이 0.5라는 것은 hot set에 속하는 데이타들도 버퍼에 완전히 캐싱되지 않는다는 것을 의미한다. 비록 버퍼 크기(BufSize)가 4000으로 HotSize의 2배 이지만, hot set을 제외한 다른 데이타를 액세스할 확률이 0.36이나 되므로 hot set이 항상 버퍼에 캐싱될 수는 없다. NumNode가 2개 이상으로 증가할 경우 대부분의 hot set의 데이타들이 적어도 하나의 노드에 캐싱되고, 따라서 버퍼 히트율은 0.64이상이 된다.

한 가지 흥미로운 결과는 하나의 AC에 할당된 노드의 수를 증가시킬 때, 노드를 추가하더라도 어느 시점에서는 더 이상 성능 개선이 이루어지지 않는다는 것이다. 본 실험에서는 4개 이상의 노드를 하나의 AC에 할당할 경우 거의 비슷한 성능을 보였다. 그 이유는 하나의 AC가 주로 참조하는 데이타베이스 클러스터의 크기(ClusterSize)가 10000이고 BufSize는 4000이므로, NumNode가 4보다 클 때 데이타베이스 클러스터의 대부분이 적어도 하나의 노드에는 캐싱되기 때문이다. 그러나

전체 데이타베이스는 8개의 데이타베이스 클러스터로 구성되므로 NumNode가 16이 되더라도 데이타베이스의 모든 페이지를 캐싱할 수는 없다. 결국, 버퍼 히트율은 거의 동일한 반면 많은 노드에 동일한 AC의 트랜잭션들이 할당됨에 따라 버퍼 무효화의 가능성이 높아져 CPU 추가에 따른 성능 개선이 상쇄된다. 이러한 결과는 성능이 좋아질 때만 추가로 친화도 노드를 할당한다는 DACA 정책의 정당성을 입증한다.

HotSize와 ClusterSize가 2배 혹은 3배로 증가할 때, 낮은 버퍼 히트율로 인해 성능이 떨어지지만 상대적으로 성능 차이는 크지 않다. 이것은 대부분의 트랜잭션에 의해 요청된 데이타들이 디스크 I/O에 의해 액세스되기 때문이다.

5.2 부하 폭주

본 절에서는 특정 AC의 부하가 폭주할 때 DACA와 PAR, 그리고 DRR 기법들 간의 성능을 비교한다. 폭주 AC의 부하에 대해서 PAR는 부하를 분산시키지 않는 정적 트랜잭션 라우팅 정책을 지원하며, DACA와 DRR은 트랜잭션 라우터에 의해 여러 개의 노드에 폭주 AC의 트랜잭션들을 분산한다.

그림 5에 폭주 AC가 존재할 경우 3가지 기법들 간의 트랜잭션 처리율이 나타난다. 세 기법을 비교하기 위해 NumNode와 ACNum을 8, WriteOpPct는 0.3으로 설정하였다. AC의 수는 8이며 동시에 실행되는 트랜잭션 수(MPL)를 320으로 설정하였다. 따라서 균등한 부하 상태에서 각 AC의 평균 트랜잭션 수는 40이 된다. 폭주 AC의 부하는 균등한 부하 상태에 대한 상대 비율(부하 폭주율)로 나타낸다. 예를 들어, 25%의 부하 폭주율은 폭주하지 않는 AC들의 부하가 25% 감소하고 감소된 부하의 총합이 폭주 AC에 추가된다. 즉, 폭주 AC의 트랜잭션 수는 110이 되고 폭주하지 않는 다른 AC들의 트랜잭션 수는 30이 된다. 100%의 부하 폭주율은 동일한 AC에 속하는 320개의 트랜잭션이 동시에 실행될 때를 나타낸다.

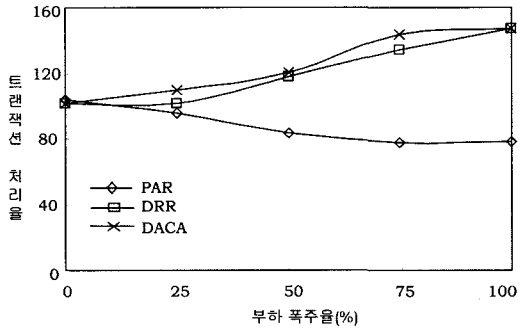

그림 5 부하 폭주에서의 트랜잭션 처리율

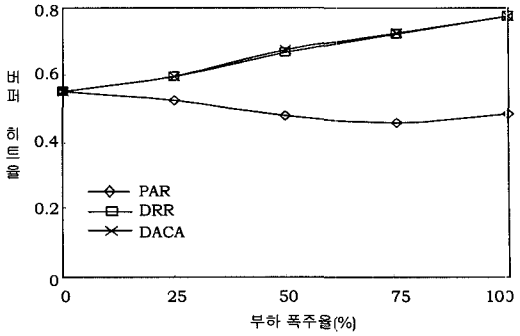

그림 6 부하 폭주에서의 버퍼 히트율

부하 폭주율이 0%일 경우는 모든 노드의 부하가 거의 동일하기 때문에 DACA나 DRR과 같은 동적 라우팅 기법들도 PAR와 동일한 방식으로 수행한다. 그러나 예상했던 것처럼 부하 폭주 비율이 증가할수록 PAR의 성능이 떨어졌다. PAR는 폭주 AC의 부하를 다른 노드들로 분산시키지 않기 때문에 그림 6과 같이 폭주 AC에 대해 낮은 버퍼 히트율을 나타낸다. 5.1절에서 설명하였듯이 한 AC에 대한 hot set의 데이타들을 한 노드의 버퍼에 모두 캐싱할 수 없기 때문에, 폭주 AC는 버퍼 히트율이 현저하게 떨어지고 결과적으로 디스크 I/O가 빈번하게 발생한다. 100% 부하 폭주의 경우, PAR의 성능은 동적 라우팅 기법들에 비해 거의 절반 수준에 불과했다. 이것은 그림 3에서 단일 AC가 하나의 노드에서 실행될 때의 성능과 같다.

DACA와 DRR은 부하 폭주 비율이 증가할수록 성능이 더 좋아졌다. 그 이유는 부하 폭주 비율이 높을 경우, 폭주 AC의 트랜잭션들이 여러 노드에 할당됨으로써 노드들 사이에 부하를 균등하게 분배할 수 있기 때문이다. 뿐만 아니라, 폭주 AC를 실행하는 노드 수가 증가하므로 데이타베이스 클러스터를 캐싱할 수 있는 총 버퍼 크기도 증가한다. 그 결과, 그림 6에서 나타나는 것과 같이 부하 폭주 비율이 증가할 때 동적 라우팅 기법들의 버퍼 히트율이 증가한다.

DACA는 25%에서 75%의 부하 폭주율 사이에서 DRR 보다 성능이 우수한 것으로 나타났다. 이 결과는 그림 6의 버퍼 히트율과 일치하지 않는데, 그림 6에서는 DRR과 DACA의 버퍼 히트율이 거의 동일하며 부하 폭주율이 50%일 경우에는 DRR의 버퍼 히트율이 DACA의 경우보다 조금 더 높게 나타난다. 그 이유는 그림 6의 버퍼 히트율은 지역 버퍼 히트율과 원격지 버퍼 히트율을 합한 값을 표현하기 때문이다. 여기에서 지역 버퍼 히트율은 노드에서 요청된 데이타가 지역 버퍼에 캐싱되어 있을 확률을 의미하며, 원격지 버퍼 히트율은 요청된 데이타가 다른 노드의 버퍼에 캐싱되어 있을

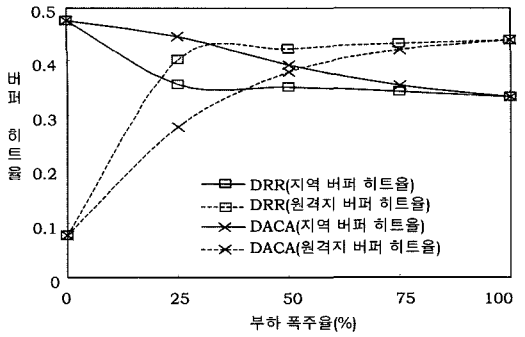

그림 7 부하 폭주에서 지역/원격지 버퍼 히트율

확률이다. 그림 6의 버퍼 히트율의 내용을 구체적으로 설명하기 위하여 그림 7에서 버퍼 히트율을 지역 버퍼 히트율과 원격지 버퍼 히트율로 분리하여 나타내었다. 25%에서 75%까지의 부하 폭주율 사이에서 DACA의 지역 버퍼 히트율은 DRR보다 높게 나타나지만 원격지 버퍼 히트율은 낮게 나타났다. 지역 버퍼 히트율이 높을 경우, 노드들 사이의 페이지 전송으로 인한 메시지 통신 오버헤드를 줄일 수 있다는 장점이 있다. DRR의 경우 폭주 AC를 모든 노드로 분산시킴으로써 버퍼 히트율을 높이고 디스크 I/O를 줄일 수 있지만, 원격지 버퍼 히트율이 상대적으로 높기 때문에 통신 오버헤드가 증가한다. 즉, 동일한 페이지가 보다 많은 노드에 캐싱되므로 버퍼 무효화 효과가 빈번하게 발생하며, 그 결과 지역 버퍼 히트율이 낮아짐으로써 DACA에 비해 DRR의 성능이 저하되었다.

6. 결 론

고성능 트랜잭션 처리를 위하여 여러 대의 컴퓨터를 연동하는 클러스터는 처리 능력과 가용성, 그리고 비용의 측면에서 장점을 가진다. 뿐만 아니라, SAN과 같은 클러스터 하드웨어 기술의 급속한 발전에 따라 큰 규모의 데이타베이스 클러스터 시스템의 개발이 가능해지고 있다. 본 논문에서는 SD 클러스터 환경에서 친화도 기반 라우팅을 지원하면서 노드들의 부하를 동적으로 분산할 수 있는 동적 트랜잭션 라우팅 기법인 DACA를 제안하였다. 기존에 제안된 동적 트랜잭션 라우팅 정책과 비교하여 DACA는 지역 버퍼에 대한 참조 지역성을 높이고 버퍼 무효화 오버헤드를 줄임으로써 시스템의 성능을 향상시킬 수 있다는 장점을 갖는다.

제안한 기법의 성능을 분석하기 위하여 SD 클러스터 모의실험 환경을 구축하였고, 다양한 데이타베이스 부하와 시스템 구성 하에서 실험을 수행하였다. 실험의 결과로 DACA는 부하가 동적으로 변화될 때 순수한 친화도 기반 라우팅 기법보다 뛰어난 성능을 보였다. 뿐만 아니라, 부하 상태에 따른 효과적인 노드 할당 정책에 의해 DACA는 기존의 친화도 기반 동적 트랜잭션 라우팅 알고리즘보다 우수한 성능을 나타내었다. 기존의 기법들은 폭주하는 트랜잭션 클래스에 전체 노드를 할당함으로써 빈번한 버퍼 무효화가 발생하고, 그 결과 지역 버퍼 히트율이 저하됨으로써 성능이 저하되었다. 이와는 달리 DACA는 AC 과부하가 되지 않는 범위에서 최소의 노드를 할당함으로써 참조 지역성을 높이고 버퍼 무효화의 오버헤드를 줄일 수 있다.

참 고 문 헌

[1] M. Yousif, "Shared-Storage Clusters," *Cluster Comp.*, Vol.2, No.4, pp.249-257, 1999.

[2] *IBM DB2 Data Sharing: Planning and Administration*, IBM SC26-9935-01, 2001.

[3] *Oracle 9i Real Application Cluster-Concepts*, Oracle Part No. A89867-02, 2001.

[4] H. Cho, "Cache Coherency and Concurrency Control in a Multisystem Data Sharing Environment," IEICE Trans. Information and Syst., Vol.E82-D, No.6, pp.1042-1050, 1999.

[5] 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 Int. Conf. on VLDB*, pp.193-207, 1991.

[6] 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.

[7] C. Nikolaou, M. Marazakis, and G. Georgiannakis, "Transaction Routing for Distributed OLTP Systems: Survey and Recent Results," *Info. Sciences*, Vol.97, No.1-2, pp.45-82, 1997.

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

[9] E. Rahm, "A Framework for Workload Allocation in Distributed Transaction Processing Systems," *J. Syst. Software*, Vol.18, pp.171-190, 1992.

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

[11] B. Shirazi, A. Hurson, and K. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press, 1995.

[12] M. Squillante and E. Lazowska, "Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling," *IEEE Trans. Parallel and Distributed Syst.*, Vol.4, No.2, pp.131-143,

1993.
[13] J. Torrellas, A. Tucker, and N. Gupta, "Evaluating the Performance of Cache-Affinity Scheduling in Shared-Memory Multiprocessors," *J. Parallel and Distributed Comp.*, Vol.24, No.2, pp.139-151, 1995.
[14] B. Hamidzadeh, L. Kit, and D. Lilja, "Dynamic Task Scheduling Using Online Optimization," *IEEE Trans. Parallel and Distributed Syst.*, Vol.11, No.11, pp.1151-1163, 2000.
[15] V. Kanitkar and A. Delis, "Site Selection for Real-Time Client Request Handling," *Proc. 19th Int. Conf. on Distributed Computing Syst.*, pp.298-305, 1999.
[16] T. Kunz, "The Influence of Difference Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Trans. Software Eng.*, Vol.17, No.7, pp.725-730, 1991.
[17] M. Carey, S. Krishnamurthi, and M. Livny, "Load Control for Locking: The 'Half-and-Half' Approach," *Proc. 9th Symposium on Principles of Database Syst.*, pp.72-84, 1990.
[18] H. Schwetman, *CSIM User's Guide for use with CSIM Revision 16*, MCC., 1992.

온 경 오

1999년 영남대학교 컴퓨터공학과 학사
2001년 영남대학교 컴퓨터공학과 석사
2001년~현재 영남대학교 컴퓨터공학과 박사과정. 관심분야는 분산/병렬 데이타베이스, 트랜잭션 처리, 저장 시스템 등

조 행 래

1988년 서울대학교 컴퓨터공학과 학사
1990년 한국과학기술원 전산학과 석사
1995년 한국과학기술원 전산학과 박사
1995년~현재 영남대학교 전자정보공학부 부교수. 관심분야는 분산/병렬 데이타베이스, 트랜잭션 처리, DBMS 개발 등