데이타베이스 공유 시스템에서 동적 부하 분산

정창욱*, 조행래**
*영남대학교 컴퓨터공학과
e-mail: manza@korea.com

Dynamic Load Balancing for Database Sharing Systems

Changuk Jeong*, Haengrae Cho**
*Department of Computer Engineering, Yeungnam University

요약

데이타베이스 공유 시스템(Database Sharing System: DSS)은 고성능의 온라인 트랜잭션 처리를 위해 다수 개의 컴퓨터를 연동하는 방식으로 각 노드들은 디스크 계층에서 데이타베이스를 공유한다. DSS를 구성하는 각 노드에 트랜잭션을 할당하는 정책이 잘못될 경우 특정 트랜잭션의 폭주로 노드에 과부하가 발생할 수 있으므로, 각 노드의 성능을 최적화하기 위한 부하 분산이 필요하다. 본 논문에서는 트랜잭션 클래스에서 참조하는 데이타베이스의 핫 셋 크기, 각 노드의 메모리 크기와 CPU 성능, 동시에 실행되는 트랜잭션 수의 변화에 따른 처리량 등을 고려한 동적 부하 분산 기법을 제안한다.

1. 서론

데이타베이스 공유 시스템(Database Sharing System: DSS)은 고성능의 온라인 트랜잭션 처리를 위해 다수의 컴퓨터를 연동하는 시스템이다[7]. DSS를 구성하는 각 노드는 메모리와 운영체제, 그리고 DBMS를 포함하며, 디스크 계층에서 데이타베이스를 공유한다. 즉, DSS의 모든 노드들은 데이타를 참조하기 위해 필요한 디스크를 직접 액세스할 수 있다.

DSS에서 많이 사용되는 동시성 제어 기법은 2단계 로킹 기법이며, 로킹을 구현하는 기법으로는 중앙집중형 기법과 분산형 기법이 있다. 중앙집중형 기법은 전역 로크 관리자(Global Lock Manager: GLM)라는 노드를 사용하여 공유 데이타에 대한 모든 로킹 정보를 관리한다[5, 6]. 이 경우, GLM이 모든 데이타의 로크에 대한 정보를 가지고 있으므로 관리가 단순하다는 장점이 있다. 그러나, 로크 요청 및 해제를 위하여 GLM과의 메시지 통신이 필요하므로 GLM으로의 통신 집중에 따른 성능 저하가 발생될 뿐 아니라, GLM 노드의 고장시 전체 시스템의 동작이 중단되는 단점이 있다.

이에 반해 분산 로크 관리 기법은 데이타베이스를 논리적으로 분할하여 각 분할에 대한 로킹 정보는 DSS를 구성하는 노드들에 나누어 저장된다[3]. 각 노드에 할당된 데이타베이스 분할에 관한 권한을 주사본 권한(Primary Copy Authority: PCA)이라 한다[1].

연관있는 데이타 영역을 참조하는 트랜잭션들을 해당 PCA 노드에 우선적으로 할당하는 유사성 기반 부하 분산은 참조 지역성에 의해 DSS의 성능을 향상시킨다. 그러나, 특정 데이타를 참조하는 트랜잭션 클래스가 폭주할 경우 그 PCA 노드에 과부하가 발생할 수 있으므로 후보 노드를 결정하여 부하를 분산하는 기법이 필요하다[4]. PCA 노드를 동적으로 관리하는 방법은 노드의 부하에 대한 융통성 있는 대처가 가능하지만, PCA 관리가 어려운 단점이 있다. 이와는 달리, 트랜잭션 처리량을 고려한 부하량 계산으로 관리하는 방법은 PCA 관리는 쉽지만 이러한 계산이 과부하 판단의 근거로서는 부적절한 단점이 있다.

본 논문에서는 모든 노드에서 부하가 균형을 이루는 완전 분산의 경우와 특정 트랜잭션 클래스에 부하가 편중되는 경우로 나누어, 트랜잭션 클래스에서 참조하는 데이타베이스의 핫 셋 크기, 각 노드의

메모리 크기와 CPU 성능, 동시에 실행되는 트랜잭션 수의 변화에 따른 처리량 등을 고려한 동적 부하 분산 기법을 제안한다.

2. 관련연구

정적 PCA 관리는 시스템 전체의 PCA 할당에 관한 정보를 카탈로그(catalog) 형식으로 구성하여 모든 노드에 중복시키는 방식이다[2]. 이 방식의 단점은 PCA는 항상 일정한 노드에 고정되어 있어 특정 노드에 과부하가 발생하게 되면 적절한 대처를 하지 못하여 시스템의 성능저하를 초래할 수 있다.

PCA를 동적으로 할당하는 방법 중의 하나가 로크 보유 기법이다[8]. 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 필요 없이 바로 로크를 부여받을 수 있고, 트랜잭션 완료시에도 로크 해제 메시지를 전송할 필요가 없으므로 부하가 분산되는 효과를 가진다. 그러나 자주 참조되지 않는 데이타라 할지라도 데이타를 참조할 때마다 그 데이타에 대한 로크가 유지되기 때문에 로크 관리에 대한 많은 부담이 따른다.

트랜잭션 처리량을 고려한 부하량 계산 방법은 PCA의 관리가 용이한 점은 있지만, 트랜잭션마다 참조하는 데이타의 수 및 연산의 복잡도 등이 다른 만큼 단순히 트랜잭션의 평균 처리율이나 응답 시간을 과부하 판단의 근거로 삼는 것은 부적절하다.

특정 노드의 과부하 상태는 Half-and-Half(HaH) 알고리즘을 이용하여 계산되어질 수 있다[9]. HaH 알고리즘은 각 노드에서 실행 중인 트랜잭션에 대해, 트랜잭션이 로크를 대기하지 않고 실행될 때 *running* 상태에 있다고 정의하고, 트랜잭션이 참조할 데이타 중에서 25% 이상의 데이타에 대한 로크를 이미 보유했을 경우 *mature* 상태에 있다고 정의한다. 따라서 현재 실행 중인 트랜잭션들에 대해 다음과 같은 상태가 존재할 수 있다.

<표 1> HaH 알고리즘에서 트랜잭션 상태

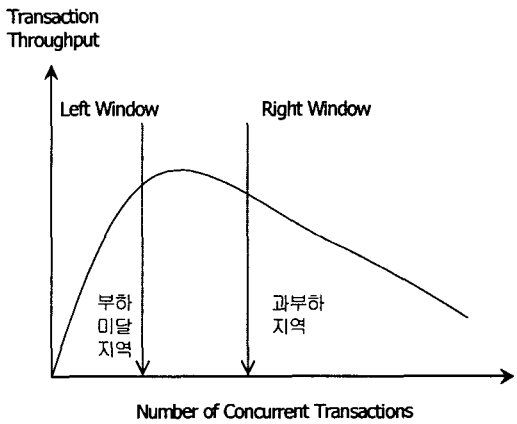
상태	Running	Mature
상태 1	Yes	Yes
상태 2	Yes	No
상태 3	No	Yes
상태 4	No	No

즉, HaH 알고리즘은 노드에서 현재 실행 중인 전체 트랜잭션 수에 대해 각 상태의 트랜잭션 수를 비교함으로써 노드의 과부하 상태를 판단하는데, 다음과 같은 식으로 나타내어진다.

$$\text{과부하} = \left(\frac{\text{상태 3의 트랜잭션 수}}{\text{실행중인 전체 트랜잭션 수}}\right) > (0.5+\delta)$$

$$\text{부하미달} = \left(\frac{\text{상태 1의 트랜잭션 수}}{\text{실행중인 전체 트랜잭션 수}}\right) > (0.5+\delta)$$

위의 식을 그림으로 표현하면 그림 1과 같다.



<그림 1> 실행 트랜잭션 수와 노드 부하 상태

동시에 실행되는 트랜잭션이 Right Window를 초과할 경우 각 노드의 부하는 급격히 하락한다. 그 이유는 로킹 기반의 동시성 제어의 경우 동시에 실행되는 트랜잭션이 증가할수록 로크 충돌과 쓰레드들간의 문맥 교환 및 디스크 IO 시간의 증가에 따른 자원 충돌이 증가하기 때문이다. 그러나, DSS가 사용되는 전형적인 OLTP 응용 분야에서는 트랜잭션 길이가 일반적으로 짧고 데이타베이스의 크기가 매우 크므로 로크 충돌 현상은 빈번하지 않을 것으로 예상된다. 즉, 자원 충돌에 의해 노드 과부하 현상이 일어나는 것이 대부분인 만큼, 과부하가 발생한 노드에서 실행될 트랜잭션들을 다른 노드에게 할당함으로써 전체 시스템의 성능을 지속적으로 향상시키는 것이 가능하다.

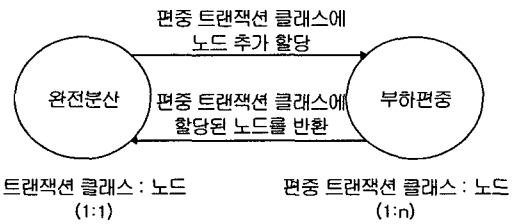
3. 부하 분산

본 절에서는 트랜잭션 클래스가 균일하게 완전히 분산된 경우와 특정 트랜잭션 클래스에 부하가 편중

되어 나타나는 경우 두 가지로 나누어 각각의 경우에 노드를 할당하는 방법에 대해 살펴본다.

그림 2에서와 같이, 완전 분산 상태에서 특정 트랜잭션 클래스가 편중되면 후보 노드에서 선택된 노드를 추가로 편중된 클래스에 할당하고, 반대로 부하 편중 상태에서 편중된 클래스의 빈도수가 줄어들면 할당받았던 노드를 다른 클래스에 반환하는 방식이다. 이때 고려해야 할 조건은 다음과 같다.

- 각 노드의 CPU 속도와 메모리 크기
- 트랜잭션 수(대기 큐에 있는 트랜잭션 수, 실행되면서 블록된 트랜잭션 수)
- I/O 대기 길이
- 트랜잭션 클래스가 참조하는 핫 셋 크기



<그림 2> 부하에 따른 노드 할당

3.1 정적 부하 분산

노드의 집합을 N, 트랜잭션 클래스의 집합을 TC, 노드의 수를 $\#N$, 트랜잭션 클래스의 수를 $\#TC$라 하자. 그리고, 트랜잭션 클래스 TC에 속하는 임의의 트랜잭션 클래스 t에 대해, t의 부하 $Load(t)$는 다음과 같이 정의된다.

$$\exists t \in TC,\ 0 \le Load(t) \le 1$$

즉, t에 속하는 트랜잭션이 하나도 실행되지 않을 경우 $Load(t) = 0$이며, 현재 실행되는 모든 트랜잭션들이 t에 속할 때 $Load(t) = 1$이다.

부하가 완전하게 분산된다는 것은 TC에 속하는 모든 t에 대해

$$Load(t) = \frac{1}{\#TC}$$

일 때를 의미한다. 즉, 특정 트랜잭션 클래스에 부하가 편중되는 것이 아니라 모든 트랜잭션 클래스들이 균일하게 나타나는 것이다.

임의의 노드 n의 메모리 크기를 $Mem(n)$으로 정의하고, 임의의 트랜잭션 클래스 t가 참조하는 데이타베이스의 핫 셋 크기를 $Hot(t)$로 정의한다. 이 때, 트랜잭션 클래스가 참조하는 데이타베이스의 핫 셋 크기가 할당된 노드의 메모리 크기보다 작아서 데이타를 캐쉬할 수 있을 경우, 즉,

$$\forall t \in TC,\ n \in N,\ Hot(t) \le Mem(n)$$

인 경우에는 노드의 수만큼 트랜잭션 클래스를 할당한다[1]. 반대로, 트랜잭션 클래스가 참조하는 데이타베이스의 핫 셋 크기가 노드의 메모리 크기보다 커서 데이타를 캐쉬할 수 없을 경우, 즉,

$$\exists t \in TC,\ n \in N,\ Hot(t) > Mem(n)$$

인 경우에는 다음의 식처럼, 핫 셋 크기만큼의 메모리가 되도록 노드를 여러 개 묶어서 할당한다.

$$Hot(t) \le \sum Mem(n)$$

3.2 동적 부하 분산

일반적으로 트랜잭션 클래스의 부하가 3.1에서와 같이 균일하게 나타나는 경우는 드물다. 작업 초기에 균일한 부하가 이루어 질 수는 있지만, 사용자의 작업 패턴에 따라 시간이 지날수록 특정 트랜잭션 클래스에 부하가 편중되는 것이 일반적인 현상이다. 특정 트랜잭션 클래스에 부하가 편중되면 기존에 할당되어 있던 노드에 과부하가 발생되어 전체 시스템의 성능 저하를 초래할 수 있다. 따라서, 완전 분산 상태에서 특정 트랜잭션 클래스가 편중되면 후보 노드에서 선택된 노드를 편중된 클래스에 추가로 할당한다. 추가로 할당되는 노드의 수를 $\#N'$라 할때($\#N'$는 t에 이미 할당된 노드 수를 포함한다), $\#N'$를 다음과 같이 정의한다.

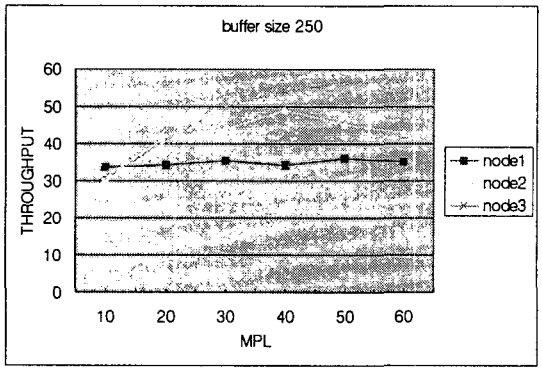
$$\#N' = \#N \times Load(t)$$

각 트랜잭션 클래스들은 초기에 $1/\#TC$의 비율로 균일하게 노드에 할당된다. 사용자의 작업에 따라 특정 트랜잭션 클래스의 부하가 $k/\#TC$로 증가하면, 나머지 트랜잭션 클래스들의 부하의 합은 $(\#TC - k)/\#TC$ 가 된다. 예를 들어, t_i, t_j, t_k, 세 개의 트랜잭션 클래스가 있다고 하자. 각 트랜잭션 클래스들의 초기 부하는 $Load(t_i) = Load(t_j) = Load(t_k) = 1/3$로 동일하다. 만약 t_i에 부하가 편중되어 $Load(t_i) = 2/3$로 증가한다면, 나머지 트랜잭션 클래스들의 부하는 $Load(t_j) + Load(t_k) = 1/3$이 된다. 즉, 전체 트랜잭션 클래스의 부하는 1이므로 특정 트랜잭션 클래스에 부하가 편중되면 다른 트랜잭션 클래스들의 부하는 상대적으로 낮아진다. 노드의 할당 방법은 $\#N'$개 노드의 부하를 합하여 그 값이 최소인 노드를 선택하여 할당한다. 그 이유는, 추가로

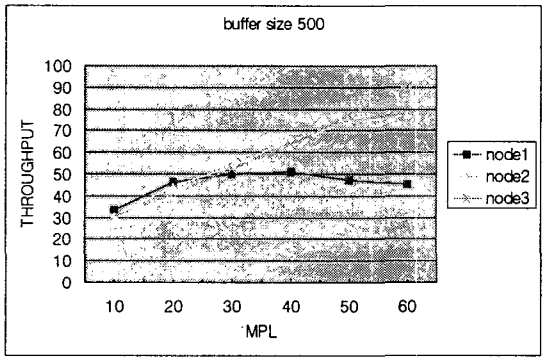
1) 본 논문에서는 $\#TC = \#N$인 경우만 가정한다. $\#TC$가 $\#N$의 정수배일 경우에도 확장이 용이하다.

할당되는 노드에 있던 기존의 트랜잭션 클래스들이 나머지 노드에 새로이 할당될 때 발생하는 부하의 증가를 최소화하기 위한 것이다. 최악의 경우, $\#N' = \#N$이면 특정 트랜잭션 클래스 t에 대해 $Load(t) = 1$일 경우이므로, DSS를 구성하는 모든 노드들이 t를 실행하고, t에 속하는 트랜잭션은 round-robin 방식으로 각 노드에게 차례로 할당한다.

그림 3과 그림 4의 경우, $Load(t) = 1$인 트랜잭션 클래스가 존재할 때 t에 많은 수의 노드를 할당함으로써 성능이 좋아지는 것을 알 수 있다.



<그림 3> 노드 수에 따른 트랜잭션 처리율
(메모리 크기 250)



<그림 4> 노드 수에 따른 트랜잭션 처리율
(메모리 크기 500)

4. 결론

GLM을 이용한 부하 분산 기법은 로크 요청 및 해제를 위하여 GLM과의 메시지 통신이 필요하므로 GLM으로의 통신 집중에 따른 성능 저하가 발생될 뿐 아니라, GLM 노드의 고장시 전체 시스템의 동작이 중단되는 단점이 있다. 그리고, 트랜잭션 처리량을 고려한 부하량 계산으로 관리하는 방법은 이러한 계산이 과부하 판단의 근거로서는 부적절한 단점이 있다. 본 논문에서는 부하 분산의 상태를 정적인 경우와 동적인 경우로 구분하여, 각각의 상태에 적합한 부하 분산 기법을 제안했다. 부하의 완전 분산인 정적인 상태에서 특정 트랜잭션 클래스가 편중되면 후보 노드에서 선택된 노드를 편중된 클래스에 추가로 할당하고, 반대로 부하 편중 상태에서 편중된 클래스의 빈도수가 줄어들면 할당받은 노드를 다른 클래스에 반환함으로써 시스템의 상태 변화에도 유동적인 부하 분산이 가능하다. 본 논문의 향후 과제는 제안한 알고리즘을 모의 실험 모형을 통해 성능을 분석하는 것이다.

참고문헌

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

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

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

[4] C. N. Nikolaou et al., "Transaction Routing for Distributed OLTP Systems: Survey and Recent Results," Information Sciences Vol.97, Issues 1-2, pp.45-82, 1997.

[5] P. Yu et al., "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.

[6] 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.

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

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

[9] Michael J. Carey et al., "Load Control for Locking: The 'Half-nad-Half' Approach," PODS, pp72-84, 1990.