트랜잭션 부하 분산 알고리즘을 위한 웹 기반 시뮬레이션
Web-based Simulation for Transaction Load Balancing Algorithm

남 영 환, 김 기 형, 조 행 래
영남대학교 컴퓨터 공학과

요 약

이 기종의 다중 시스템 환경에서의 트랜잭션 처리는 전체 참여 노드의 성능을 최대로 이용하기 위해 효과적인 트랜잭션의 분배가 반드시 필요하다. 이를 위해 트랜잭션 처리에 의해 발생하는 다양한 부하들을 참조해야 한다. 본 논문에서는 효율적인 트랜잭션 분배를 위해 분산 로킹 정보인 주사본 권한(PCA) 정보를 이용한다. 주사본 권한 정보를 이용해 트랜잭션 처리 데이터의 참조 지역성을 극대화하고, 이를 통해 캐쉬 효과를 증가시킨다. 즉 각 노드에 저장된 데이터를 자신의 노드와 나머지 노드에 분배해 캐싱하고 이를 이용해 디스크 접근을 최소화한다. 트랜잭션 분배기는 전역적인 캐싱 정보를 관리하고 이를 이용해 전체 노드들이 평균적인 트랜잭션 처리율을 유지할 수 있도록 트랜잭션을 분배한다. 제안된 기법의 검증을 위해 트랜잭션 처리 시스템을 웹 기반 환경에서 시뮬레이션할 수 있도록 Simjava를 이용해 모델을 구성한다.

1. 서론

1)부하 분산은 생산 설계, 운송, 동력 공급 등의 산업 전반에 적용 가능한 폭 넓은 분야에서 연구되어 왔으며, 기존의 부하 분산 기법은 임의 분할(random splitting) 방법과 최소 사용 노드에 트랜잭션을 분배하는(send to least utilized node) 방법으로 구분될 수 있다[1].

분산 트랜잭션 처리 시스템의 경우에 사용자로부터 요청되는 트랜잭션 처리를 적절한 노드에 할당하는 역할을 수행하는 것을 트랜잭션 분배(transaction routing)이라 한다. 트랜잭션의 분배를 수행하는 노드가 트랜잭션이 실행될 워크스테이션을 결정하기 위해서 동적인 부하 상태만을 고려한다면 가장 적은 부하 상태 값을 가지는 노드에 트랜잭션을 할당하면 된다. 그러나 실제 분산된 트랜잭션 처리 시스템에서는 동일한 트랜잭션 타입들을 동일한 노드에서 실행하도록 함으로써 노드의 참조 지역성을 높이고 캐쉬 효율성을 극대화하며, 로킹에 소요되는 메시지 오버헤드를 최소화하기 위해 각 노드의 주사본 권한(PCA) 정보[2]를 이용한다.

본 논문에서는 트랜잭션 분배기가 두 가지의 정보에 의해 트랜잭션을 분배할 워크스테이션을 결정한다고 가정한다. 하나는 각 워크스테이션이 보유하고 있는 데이터에 대한 주사본 권한(PCA)이

1) 1) 본 논문은 1998년도 정보통신부 초고속정보통신 응용기술개발사업에 의하여 연구되었음

고, 나머지 하나는 워크스테이션의 부하 상태이다. 그러나 이 기종 환경에 기반한 분산 트랜잭션 처리 시스템의 경우, 부하 상태를 토대로 효과적으로 트랜잭션 라우터를 구현하기 어렵다. 왜냐하면 서로 이질적인 시스템에 대한 일관적인 부하의 측정과 정량화를 수행하기가 쉽지 않기 때문이다. 또한 트랜잭션 처리 시스템에 참여하는 노드의 수와 연동되는 디스크의 수, 네트워크의 대역폭, 동시에 동작하는 쓰레드의 수 등에 의해 트랜잭션 처리 효율은 변화되며, 이를 고려한 효과적인 트랜잭션 분배 방법을 구현하는 것은 대단히 복잡하다.

따라서 본 논문에서는 이질적인 환경의 트랜잭션 처리 시스템간의 효과적인 부하 분배를 위해 트랜잭션 처리에 필요한 데이터의 참조 지역성을 극대화하고 전체 노드가 일정한 부하량을 유지할 수 있는 트랜잭션 분배 방법을 제안한다. 또한 제안된 기법을 검증하기 위해 이 기종 환경의 분산 트랜잭션 처리 시스템에서의 트랜잭션 처리와 트랜잭션의 분배를 위한 시뮬레이션 모델을 구성한다. 이는 트랜잭션 처리 시 발생할 수 있는 다양한 환경 변수등을 간단하게 표현할 수 있고 필요에 따라 수시로 변화시킬 수 있기 때문이다.

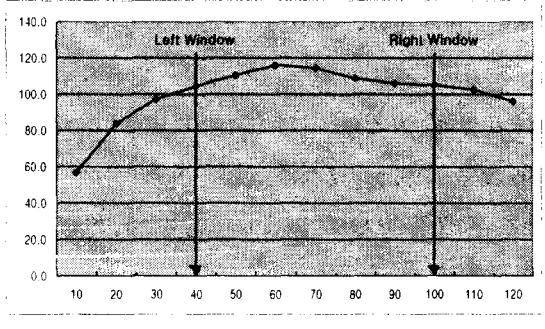
시뮬레이션 모델 구성을 위해 웹 기반 환경에서의 시뮬레이션 기능을 제공하는 SimJava 시뮬레이션 라이브러리[3]를 사용한다. 이를 통해 트랜잭션 처리 노드 프로세스와 쓰레드, 네트워크, 디스크, 라우터 등을 모델링한다. 이때 트랜잭션 라우터는 제안된 알고리즘에 대해 구현된다.

본 논문의 구성은 다음과 같다.

먼저 2장에서는 이 기종 환경의 다중 트랜잭션 처리 시스템에 대해 설명하며, 3장에서는 제안 기법을 설명한다. 4장에서는 시뮬레이션 모델과 환경에 대해 설명하고, 마지막으로 5장에서 본 논문의 결론을 맺는다.

2. 이 기종 환경의 다중 트랜잭션 처리 시스템

본 논문에서 사용한 다중 트랜잭션 처리 시스템은 PHLOX[4] 시스템이다. PHLOX는 데이터를 공유하는 다중 시스템 환경의 고성능 트랜잭션 처리 시스템이며, 수퍼 컴퓨터나 MPP에 비해 저렴한 워크스테이션을 다수 개 연동해 고속의 트랜잭션 처리 능력을 실현할 수 있도록 설계되어 있다. 또한 PHLOX는 이 기종 환경을 고려하고 있으므로 다양한 형태의 연동이 가능하며, 각 시스템의 성능과 데이터의 저장 형태에 따라 전체 시스템의 트랜잭션 처리 성능이 유동적일 수 있다. 따라서 트랜잭션의 적절한 분배를 통해 자원의 효율적인 활용이 가능해진다. 이를 위해 본 논문에서는 단일 시스템의 최적의 트랜잭션 처리 능력을 측정했다. <그림 2-1>은 단일 시스템의 성능을 나타낸다.



<그림 2-1> 단일 시스템의 성능

이때 가로축은 동시에 실행되는 트랜잭션의 수를 나타내며, 세로축은 단위 시간당 트랜잭션 처리량(TPS)를 나타낸다. PHLOX 시스템의 경우에 동시에 트랜잭션이 동시에 실행된다는 것은 각 트랜

잭션에 해당하는 쓰레드의 수를 의미하며, 일정 범위 이상의 트랜잭션 수에 대해서는 쓰레드들간의 문맥 교환으로 인해 성능의 저하 현상이 나타난다.

그림에 표시된 Left Window와 Right Window는 각각 일정 수준 이상의 트랜잭션 처리율을 보이는 최소와 최대의 부하량을 나타낸다. 이렇게 구해진 값은 하위 경계가 40개의 트랜잭션이 동시에 실행되는 시점에서 104.4 TPS, 100개의 트랜잭션에서 104.8 TPS, 그리고 상위 경계는 60개의 트랜잭션의 시점에서 116.3 TPS를 나타내었다. 따라서 단일 PHLOX 시스템은 트랜잭션 수가 40 ~ 100개의 범위에서 최적의 성능을 나타낸다고 할 수 있으며, 이때 Left Window는 40, Right Window는 100으로 한다.

본 논문에서는 측정된 Window 값을 이용해 트랜잭션 처리 시스템의 각 노드들의 단위 시간당 트랜잭션 처리율은 일정한 수준 이상으로 유지되도록 한다.

3. 제안 기법

본 논문에서는 부하량의 적절한 분산을 위해 주기적인 부하 조정 작업을 수행하도록 한다. 이를 위해 단일 시스템 성능의 하위 경계와 상위 경계 값을 이용하며, 동시에 캐싱 효과를 극대화하기 위해 각 노드가 보유하고 있는 데이터의 주사본 권한 정보를 이용한다.

각 시스템의 부하량을 일정하게 유지하기 위해 Left Window와 Right Window의 범위를 이용하며, 부하 조정기는 시스템의 부하량을 측정하고 하위 경계 이하의 값이거나 상위 경계 이상의 값인 경우에 제안된 방법에 의해 부하량을 조정한다. 부하량 조정을 위해 각 노드에 분산 배치되어 있는 데이터의 로킹 정보를 참조한다. 즉 각 노드는 자신이 로킹 정보를 보유하고 있는 데이터의 캐싱 정보를 관리하며, 이 정보는 부하 조정 작업에 사용된다. <그림 3-1>은 부하 조정 알고리즘을 나타낸다

```
                                          <-------- (A)
                                          <-------- (A-1)
LoadBalance() {
  for (i=0; i<NumOfNodes; i++) {
    if (Load[i] > RWin[i]) {
      if (node(i) has tokens) {
        select a token to transfer;
        if (PCA_node(token)->Load < LWin[i])
          FreeNode = PCA_node(token);
        else {
          for (j=0; j<NumOfNodes; j++) {                    (1)
            if (Load[j] < LWin[j]) {
              FreeNode = node(j);
              if (FreeNode == previous FreeNode)
                continue;
              break;
            }
          }
          if (j > NumOfNodes)
            FreeNode = PCA_node(token);
        }
        transfer token/2 to FreeNode;
      } else {
        select a token to transfer;
        --------> 1
        if (j > NumOfNodes)                                 <- (B)
            FreeNode = null;
        if (FreeNode != null)
          transfer token to FreeNode;
      }
    } else if (node(i) gives token && Load[i] < LWin[i])
      get back token from MaxloadNode(has tokens);
  }
}
```

<그림 3-1> 부하 조정 알고리즘

각 노드가 관리하는 캐싱 정보는 페이지 단위로 관리되며 이는 다수의 레코드를 포함한다. 이러한 캐싱 정보를 이용하면 동일한 참조 정보를 갖는 트랜잭션이 동일한 노드에서 실행될 가능성을 증가시켜 노드가 보유하고 있는 캐싱 데이터의 참조 가능성을 높여줄 수 있다. 그러나 전체 시스템에 대한 페이지 단위의 캐싱 정보를 부하 조정 노드가 보유하는 것은 불가능하다. 따라서 제안 기법에서는 노드가 보유하고 있는 데이터를 캐시 영역의

크기에 맞게 분할한 토큰(token)을 이용한다. 이는 한 노드가 자신의 캐시 영역에 보유할 수 있는 크기만큼을 제외한 나머지 데이터 크기를 전체 노드의 수로 나누어서 구한 양이다.

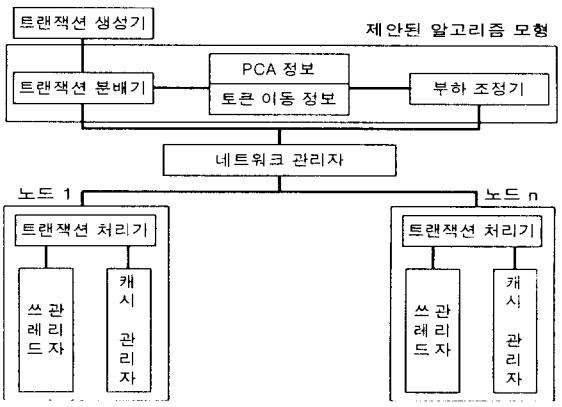
부하 조정 작업은 주기적으로 이루어지며, 전체 노드에 대해 순차적인 작업을 수행한다. 먼저 노드의 부하가 상위 경계 이상인 경우(A)에는 노드가 보유하고 있는 토큰들 중 일부를 PCA 노드나 부하량이 상대적으로 적은 노드에 전송한다. 이후부터 요청되는 트랜잭션 중 전송된 토큰을 참조하는 트랜잭션은 토큰을 전송 받은 노드로 요청되므로 부하량은 분산된다. 노드의 부하가 하위 경계 이하이고 자신의 토큰 중 일부가 다른 노드에 존재하는 경우(B)에는 자신의 토큰을 회수하게 된다. 이는 상대적으로 감소한 노드의 부하량을 하위 경계 이상으로 증가시키는 효과를 갖는다. 또한 A의 경우에서 대상 노드가 여타 노드들의 토큰을 보유하고 있을 때(A-1)는 전송할 토큰의 양을 1/2로 줄인다. 이는 노드의 부하가 어떤 토큰에 의한 것인지 명확하지 않기 때문에 토큰의 전송으로 인해 급격한 부하량의 변화가 발생할 수 있기 때문이다.

요청된 트랜잭션은 참조 데이터가 포함된 토큰을 보유하고 있는 노드로 분배되며, 이때 부하 조정기는 토큰 정보 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 분배기는 요청된 트랜잭션의 정보를 이용해 토큰을 보유하고 있는 노드를 찾고 토큰 보유 노드에 트랜잭션을 요청하도록 하면 비교적 간단하게 트랜잭션 처리 노드를 결정할 수 있다.

4. 시뮬레이션 모델 구성

본 논문에서 제안된 부하 분산 기법의 검증을 위해 Simjava 라이브러리를 이용해 시뮬레이션 모델을 구성하였다. Simjava는 웹 기반의 환경에서 시뮬레이션을 수행하고 수행 상황을 애니메이션하며 결과를 분석할 수 있는 도구를 제공하는 자바 기반의 이산 사건 시스템 시뮬레이터이다. 시뮬레이션 모형을 구성하기 위해 트랜잭션 생성기, 트랜잭션 분배기, 부하 조정기, 네트워크, 트랜잭션 처리 프로세스와 쓰레드를 모델링하였으며, 각 요소들은 자바 클래스 형태로 구현되었다. <그림 4-1>은 시뮬레이션 모형을 나타낸다.



<그림 4-1> 시뮬레이션 모형

<그림 4-1>에서 트랜잭션 생성기는 트랜잭션을 요청하는 노드를 표현하며, 다수 개의 노드를 표현하기 위해 다수 개의 쓰레드로 구성될 수 있다. 요청된 트랜잭션은 분배기에 의해 트랜잭션 처리 노드로 할당되며, PCA와 토큰 정보에 기반한 트랜잭션 분배를 위해 각 노드에 저장된 데이터의 PCA 정보와 토큰의 이동 정보를 관리한다. 이때 토큰의

이동은 부하 조정기에 의해 진행되며, 부하 조정기는 수행한 토큰 이동 정보를 트랜잭션 분배기에 전달한다. 트랜잭션 처리 노드는 자신이 저장하고 있는 데이터에 대한 PCA 정보와 캐싱 정보를 관리하며, PHLOX 시스템에서의 트랜잭션 처리 과정을 표현하기 위해 다수 개의 쓰레드를 이용해 구성되었다.

본 논문에서 사용한 시뮬레이션 구성 변수들은 <표 4-1>과 같으며, 각 변수들의 구체적인 값은 [5]을 참조하여 조정하였다.

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 ~ 100
NumDisk	공유 디스크의 수	5
MinDiskTime	최소 디스크 액세스 시간	10 ms
MaxDiskTime	최대 디스크 액세스 시간	30 ms
DBSizePerNode	각 노드에 저장된 DB 크기	20000 record
PageSize	각 페이지의 크기	4096 bytes
RecPerPage	한 페이지에 포함된 레코드 개수	20 record
CacheSize	캐시 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	256 bytes
LockInst	로크 등록/해제를 위한 명령수	300
PerIOInst	디스크 IO를 위한 명령수	3000
CreationDelay	트랜잭션 생성 시 평균 대기 시간	0.1
제안된 알고리즘 구현을 위한 변수		
TokenSize	캐시 버퍼에 저장한 DB 크기	((1-0.2) * DBSizePerNode) / ((NumOfNodes -1) * 2)
LeftWindow	단일 노드의 성능 하한 경계값	40 transactions
RightWindow	단일 노드의 성능 상한 경계값	100 transactions

<표 4-1> 시뮬레이션 변수

5. 결론

본 논문에서는 분산 환경의 고성능 트랜잭션 처리 시스템에 효과적으로 적용될 수 있는 트랜잭션 분배 방법을 논의하며, 비교적 저렴한 비용의 웹 기반 시뮬레이션을 통해 효과를 검증할 수 있도록 모델을 구성했다. 이는 기존의 시스템뿐만 아니라 향후 개발될 분산 트랜잭션 처리 시스템에 적용될 수 있으며, 이를 통해 고성능의 트랜잭션 처리 시스템을 구현할 수 있으리라 기대된다. 또한 시스템의 부하를 보다 정교하고 일관성 있게 정량화 하는 작업이 수반된다면 보다 효율적인 트랜잭션 처리 시스템을 구현할 수 있을 것이다.

참 고 문 헌

[1] A. Reuter, Load control and load balancing in a shared database management system, IEEE Computer Society Press, 1986. pp. 188-197.

[2] E. Rahm, Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems, ACM Trans. on Database Systems 18(2), 1993. pp. 333-377.

[3] F. Howell, R. McNab, Simjava package, http://www.dcs.ed.ac.uk/home/hase/simjava, April 1999.

[4] 조행래, 데이터를 공유하는 다중 시스템 환경에서 고성능 트랜잭션 처리 시스템 개발, 정보통신부 국책기술개발사업 2차년도 연차보고서, 1999.

[5] M. Zaharioudakis, M. Carey, M. Franklin, Adaptive, Fine-Grained Sharing in a Client-Server OODBMS: A Callback-Based Approach, ACM Trans. on Database Systems 22(4), 1997. pp. 570-627.