분산 공유 메모리 내에서 적응적 중복 객체에 의한 캐쉬 일관성

°장재열* 이병관**

*동우대학 사무자동화과 **관동대학교 컴퓨터공학과

Adaptive Replicated Object with for Cache Coherence in Distributed Shared Memory

Jang, Jae Youl Lee, Byung Kwan

Dept. of Office Automation, Dong-U college
Dept. of Computer science, University of KwanDong

<요약>

분산 공유 메모리 상에서 클라이언트들은 네트워크를 통해 원격 공유 메모리 상으로 접근하게 된다. 접근 시에 클라이언트들은 접근 정보를 자신의 지역 캐쉬에 저장해 두었다가 필요시에 인출해서 사용한다. 그러나 시간이 경과함에 따라 다른 클라이언트들에 의해서 데이터 갱신이 이루어질 수 있다. 이에 본 논문에서는 원격 데이터 정보를 객체로 설정하여 이 객체를 관리하여 분산 공유 메모리 상에서 데이터 일관성을 유지하고자 한다.

객체 중복을 통해서 분산 객체 시스템을 구성하였을 때 기존의 중복 기법에서 사용하는 일관성 비용 이외에 별도의 추가 비용이 없이도 제한적으로 병렬 수행의 효과를 볼 수 있다. 또한 중복 기법에 있어서 가장 큰 오버헤드로 알려진 일관성 유지비용을 최소화시키기 위하여 이 비용을 결정하는 가장 핵심적인 요소인 객체의 복사본의 수를 동적으로 변화시키면서 관리함으로써 전체 수행 시간의 측면에서 많은 향상을 가져왔다.

1. 서론

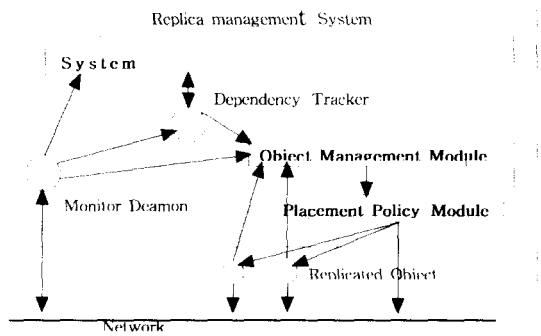
분산 시스템에서 이루어지는 작업의 형태는 대체로 다수의 사용자들이 시스템의 자원을 서로 공유하며 여러 작업이 동시에 이루어지는 복잡한 형태를 띄고 있다. 이러한 작업들을 수행하기 위하여 운영체제는 작업의 효율적 할당이나 시분할(time sharing) 방식 등 다양한 기법들을 도입하게 되었으며 또한 이 기능들은 전체 시스템의 성능에 크게 영향을 미쳤다. 이러한 대표적인 예로서 작업의 지능적 할당을 통한 시스템의 성능 향상 정책이나 다수의 데이터 복사본(replica)을 분산 유지시킴으로서 가용성을 높이고 고장감내(fault tolerance)의 효과를 기대하는 정책 등을 들 수 있겠다[1,2].

분산된 복사 본을 유지하는데 있어서는 이들 분산 저장된 데이터의 복사 본들 사이에 논리적 일관성을 유지시키는 것이 매우 중요한데, 이를 위해서 강세 복제에 의하거나, 지연 복제, 프로세스의 그룹화 등의 기법이 이용되고 있다. 그러나 이러한 방법들은 대체로 일관성 유지를 위한 방법으로 메시지를 사용하기 때문에 각 노드들 사이에는 추가적인 메시지의 발생을 수반하게 되며, 이에 따른 부가 비용 또한 적지 않아 전체 성능에 큰 영향을 미치게 된다.

최근에는 분산 시스템에서도 객체 기반의 개념을 도입하여
시스템의 구성을 용이하게 하고 성능을 높이려는 연구가 많이 진행되고 있다.

객체 기반의 분산 시스템에서는 객체의 복사 본을 유지함으로써 어렵지 않게 프로세스의 병렬 수행 효과를 기대할 수 있다. 그러나 프로세스를 병렬 수행하기 위해서는 잠금 기법이나 일관성 유지 작업 등 병렬 처리를 위한 부가 작업이 수반되어야 하는데 이들 부가 작업의 비용은 복사본의 수와 밀접한 관계를 가지고 있으므로 필요 이상으로 지나친 수의 복사 본을 유지하는 경우에는 많은 자원을 부가 작업에 소모하게 되어 오히려 성능의 하락을 가져올 수도 있다. 또한 객체의 복사 본을 유지하는데 있어 복사 본의 수가 많아질수록 일관성 유지에 참여하는 노드들 중 과부하 상태인 노드가 존재할 확률을 높여 전체 시스템의 기대 성능을 저하시키는 악영향을 가져올 수도 있다[1,3,5].

본 연구에서는 시스템을 구성하는 각 노드의 상태와 객체의 사용 패턴에 근거하여 객체 복사 수와 복사 본이 위치할 노드를 동적으로 결정하여 객체를 적응적으로 배치함으로써 전체 시스템의 성능을 향상시키는 방안을 제시하였다. 또한 기존의 중복 정책과 제시한 적응적 중복 정책을 일관성 유지 측면에서 시뮬레이션 하여 성능 향상되어짐을 평가하였다.



[그림 1] 일반적인 분산 객체 시스템의 구성

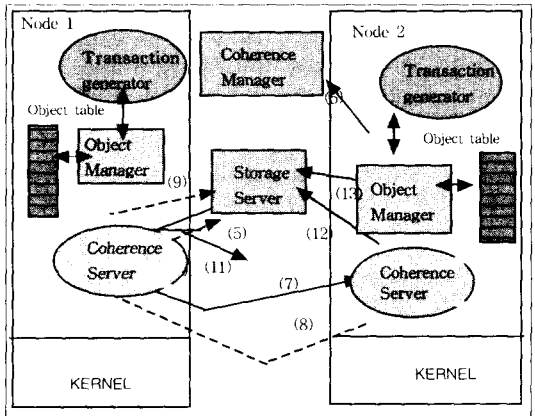
2. 일관성 제어 프로토콜

커널은 분산 공유 메모리에 대한 요구를 처리하기 위하여 객체 관리자(Object manager)에 객체 테이블(Object table)을 가진다. 이 객체 테이블은 CoherMgr에 의해 생성되어지고 파괴되어진다. 다시 말해서 객체들이 응용 프로그램에 의해서 긴 시간 동안 요구되어지지 않으면 Unmapp Object함수를 호출하게 된다. 이 함수는 CoherMgr_close Object 파라메터에 의해서 객체 테이블을 파괴시키게 됨으로써 일관성을 유지할 수 있게 한다

일관성 제어 프로토콜 흐름은 세 가지 형태로 구분할 수 있다. 첫째, 읽기 부재가 발생하는 경우이다. 노드 2가 노드 1에 이미 적재된 객체 페이지를 읽는다면 읽기 부재가 발생한다. 이때, 지역 객체 관리자는 CoherMgr_GetPage ForRead(RPC 5 ; 그림 2에서 (5)에 해당함)요구를 CoherMgr에게 전송한다.

둘째, 쓰기 접근 부재가 발생하는 경우이다. 노드 2가 동일한 페이지에 쓰기 동작을 시도할 때, 쓰기 접근 부재가 발생한다. 그리고 객체 관리자는 현재 storage server(RPC 9) 페이지를 쓰고 있는 소유자(노드 1) CoherSvr에게 CoherSvr_GetWrite Access 호출(RPC 8)을 전송하고 지역적 캐쉬와 결점 노드(노드 2)를 제외한 사본을 무효화시킨다(RPC 10).

셋째, 쓰기 부재가 발생한 경우이다. 노드 1이 동일한 객체 페이지를 쓰고자 할 때, 쓰기 부재가 발생한다. 지역적 객체 관리자은 현재 storage server (RPC 12) 페이지를 쓰고 있는 소유자(노드 2)의 CoherSvr에 CoherSvr_Get PageForWrite를 호출한다(RPC 11). 지역적 캐쉬와 사본(RPC 13)을 무효화시키고, 사본 정보와 소유권 지시자를 지우고 난 후에, 노드 1에서 그 페이지 데이터의 복사본과 쓰기 허용을 획득한다. 노드 1에 있는 객체 관리자는 그 페이지의 소유권 지시자를 세팅(RPC 13)하고 그 페이지의 쓰기 접근을 허용하여 쓰기 부재를 해결한다.

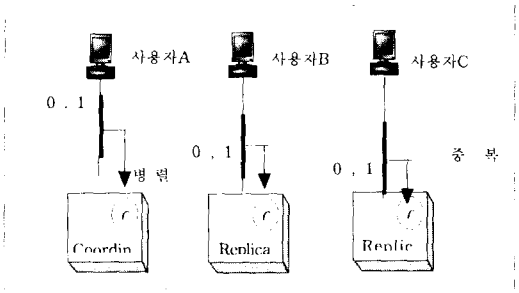


[그림 2] 분산 시스템내의 일관성 유지 제어 프로토콜

3. 적응적 중복 객체 설계 구현

3.1 중복 기법

분산 시스템에서 객체를 어느 노드에 배치시킬 것인가 하는 것은 해당 객체를 사용하는 응용 프로그램의 성능 및 신뢰도에 결정적인 영향을 미친다. 중재자와 같이 높은 신뢰도를 요구하는 객체는 해당 객체가 위치할 수 있는 노드들 중에서 신뢰성이 가장 높은 노드 상에 위치시킬 필요가 있을 것이다. 만약 시스템내의 노드들에 대한 신뢰도가 요구조건을 충족시키지 못하거나 좀 더 높은 수준의 신뢰도를 필요로 한다면 객체의 복사 본을 2개 이상 유지시키는 방법으로 요구 수준을 만족시킬 수 있다.



[그림 3] 읽기 모드 연산의 병렬 수행

중복 기법은 같은 내용의 객체를 시스템 내에 하나 이상으로 유지시키는 방법이다. 이때 복수 개로 유지되는 객체 중 본래 처음부터 존재하던 객체를 원본(original)이라 하고 복사되어 유지되는 객체를 복사본(replica)이라 한다. 각 복사본들은 그 내용에 있어서 원본과의 일치를 유지해야하는데 이를 일관성 유지의 문제라 한다.

순수한 작업 수행을 제외한 노드의 검색 및 접근 비용을 통신비용(t_{comm})이라 하면 원격지에 존재하는 객체를 사용하는 비용(t_{remote})은 다음과 같이 표현할 수 있다.

$$t_{remote} = t_{exec} + t_{comm}$$

여기서 t_{exec}는 해당 객체의 연산을 수행하는데 드는 비용으로 객체의 연산이 수행되는 노드의 부하와 반비례의 관계를 가진다. 만약 사용할 객체가 작업을 수행하는 노드 내에 존재한다면 $t_{comm}=0$ 이므로 t_{local}은 t_{exec}와 같을 것이다. 또한 k개의 노드 $n_1, n_2, n_3 \cdots n_k$로 이루어지는 시스템에서 n_i에 이상이 발생할 확률을 f(n_i)이라 한다면(여기서 $0<f(n_i)\leq 1$이다) n1이 존재하는 객체 O_{n1}에 이상이 발생할 확률 또한 f(n_i)으로 볼 수 있다. 그러나 이 객체의 복사 본을 n1, n2, n3, n4에 위치하여 둔 경우는 노드 n1에 이상이 발생하더라도 나머지 노드들만으로 작업의 수행이 가능하며 최대(전체 복사본의 수-1)개의 노드에 결함이 발생하여도 다른 정상 노드에서는 객체를 사용할 수 있게 된다. 위에서 살펴본 예에서는 단지 n_1, n_2, n_3, n_4의 노드에 이상이 발생하였을 경우에만 해당 객체를 사용할 수 없게 된다.

3.2 적응적 중복 정책(Adaptive replication)

분산 객체 시스템에서 중복 기법을 통하여 객체의 복사본을 운영하면 일반적으로 전체 성능이 향상될 것으로 기대한

다. 전체 작업 중 갱신 작업의 비율이 높아짐에 따라 증가하게 되는 또 다른 비용은 일관성 유지비용이다. 일관성 유지비용은 해당 시스템에서 사용하는 일관성 유지 정책이나 시스템에서 수행되는 작업의 특성, 서로 다른 노드들에 존재하는 복사본의 수, 복사본을 가지는 노드들의 부하 및 성능, 분산 시스템의 구성 형태 등에 따라 크게 달라지므로 정확한 측정값을 얻기는 곤란하지만 이들 중 가장 큰 영향을 미치는 항목은 복사본의 수와 각 중복 노드의 부하이다.

본 논문에서는 잠금 기법의 비용과 일관성 유지의 비용을 구분하여 고려하는 것이 큰 의미를 가지지 못하므로 이해의 편의를 위해 두 비용의 합을 통털어 일관성 유지비용이라 칭하도록 하겠다.

갱신 작업에 있어서 기대되는 작업의 수행시간(t_{write})은 호출 작업의 수행시간(t_{exec})과 일관성 유지비용(t_{cons})에 영향을 받으므로 다음과 같이 기술될 수 있다.

$$t_{write} = t_{exec} + t_{cons} = t_{exec} + f(\text{number of replica})$$

객체의 메소드를 수행하는 데에는 일관성 유지비용만이 영향을 미치는 것이 아니므로 다른 요소를 감안한 전체 수행 시간과 복사본의 수와의 관계를 살펴보기로 한다. 객체O에 발생하는 메소드 호출 중 갱신을 요하지 않는 작업 Mr이 발생할 확률은 $\lambda read$ 라 할 때, 갱신을 요하는 작업 Mw가 발생할 확률은 $1-\lambda read$ 이다. Mr의 수행 시간(Er)은 노드 내 호출인 경우와 원격 호출인 경우로 나누어 생각해 볼 수 있으며 $\rho 1$을 작업이 발생한 노드에 객체의 복사 본이 존재할 확률, 즉 노드 내 호출일 확률이라 하고 Ttr을 객체 호출이 발생한 노드와 실제 객체가 존재하는 노드 사이에 메시지를 전송하는데 소요되는 시간, Tr을 메소드 수행하는데 실제 필요한 시간이라 한다면 $Er = \rho 1(Tr) + (1-\rho 1)(2Ttr + Tr) = 2(1-\rho 1)Ttr + Tr$이 된다.

L_n, C_n의 값을 구하려는 시도를 할 필요는 없는 것이다. 이에 따라 L_n, C_n의 일관성 유지비용을 그대로 사용하지는 않으나 이와 밀접한 관계를 가지는 λ_{read}를 이용하여 간접적으로 복사본의 수 n을 결정하는 적응적 결정 방식을 제안할 수 있다.

[알고리즘 1]에서 보는 바와 같이 이 방식에서는 기존의 시스템에서도 간단히 측정될 수 있는 읽기 모드 연산 비율 λ_{read}를 이용하는데 객체의 호출 내용을 살펴 λ_{read}가 증가하면 이에 따라 일관성 유지비용이 낮아질 것으로 판단하여 n을 증가시키고(②) 반대로 λ_{read}가 감소하는 경우에는 일관성 유지비용은 높아질 것이므로 n은 감소시키게 된다(③). 이러한 n값의 변경 결과 객체 메소드의 수행 성능이 향상되었다면 변경된 n의 값은 적절한 것이었으므로 이를 받아들이고 반대로 수행 성능이 하락되는 것으로 판단되면 오히려 변경된 n의 값이 비효율적인 값이므로 본래의 n값으로 환원시킨다(①). 이러한 방식을 통하여 정확한 L_n, C_n을 구하는 작업이 없어도 최적의 n값에 근사한 n의 값을 적응적으로 결정할 수 있을 것이다.

```
get the response time  t_response of object
if  t_response > t_previous_response                    ...①
    if direction = increased then
        decrease number of replica
    else if direction = decreased then
        increased number of replica
direction = null
get the read ratio  λ_read
if ( λ_read − λ_previous_read ) > threshola      ...②

    increase number of replica
    direction <- increased
else if ( λ_previous_read − λ_read ) > threshola     ...③
    decrease number of replica
    if number of replica < 1 then number of replica <- 1
    direction <- decreased
endif
```

[알고리즘 1] 적응적 중복기법을 통한 중재자의 n값 동적 결정 알고리즘

4. 결 론

공유 메모리를 사용하는 분산 처리 시스템에서 캐쉬 메모리의 사용은 많은 시스템의 접근 성능 향상을 가져다 주며 사용자에게 투명성을 제공하는 가장 큰 부분이다. 그러나 많은 컴퓨터들이 네트워크로 연결되어 사용 되어지므로 빈번한 데이터의 갱신과 접근이 이루어지므로 시스템에 존재하는 여러 다른 캐쉬들과 메모리 사이에서 데이터의 일관성 문제를 가져온다. 이런 데이터의 일관성 문제를 해결하기 위해서 많은 캐쉬 알고리즘들이 제안되었다.

분산 처리 시스템에서 공유 메모리 메카니즘을 제공하기 위해서 캐싱 기법을 사용하게 된다. 그런데 이 기법은 자료의 일관성 문제를 발생시키게 된다. 그러므로 분산 공유 메모리를 구축하기 위해서는 먼저 자료 일관성 문제를 해결하여야 한다. 이런 일관성 문제를 해결하기 위해서 본 논문에서는 시스템 확장이 쉬운 분산 객체 복제 기법을 이용하였다. 또한 네트워크의 객체 지명 캐쉬를 사용하여 탐색에 드는 노력을 최소화하였다. 즉, 클라이언트가 서버에게 이름의 주소 변환을 요청하면 서버는 먼저 국부적(local)인지를 확인하고, 복제 객체 캐쉬를 검사하여 최근에 사용된 이름이였는가를 확인한다.

적응적 중복 객체 기법을 통하여 분산 객체 시스템을 구성하였을 때 기존의 중복 기법에서 사용하는 일관성 관리비용 이외에 별도의 추가 비용이 없이도 제한적으로 병렬 수행의 효과를 얻을 수 있었다. 또한 중복 기법에 있어서 가장 큰 오버헤드로 알려진 일관성 유지비용을 최소화시키기 위하여 이 비용을 결정하는 가장 핵심적인 요소인 객체의 복사본의 수를 동적으로 변화시키면서 관리함으로써 전체 수행 시간의 측면에서 많은 향상을 가져왔다.

< 참 고 문 헌 >

[1] M. C. Little and S. K. Shrivastava, Replicated K-Resilient Objects in Arjuna, Proceedings of the 1st IEEE Workshop on the Management of Replicated Data, Houston, November 1997, pp53-58

[2] J. Achibald and J. L. Baer, "Cache Coherence Protocol : Evaluation Using a Mutiprocessor Simulation Model," ACM Trans. on Comp. System, vol.4, no.4, pp273-298, Nov. 1996

[3] D.C.Winsor' and T. N. Mudge, "Analysis of bus for Multiprocessors, 15th Int. Symp. on Comp. Arch. pp 100-105, 1998.

[4] P. A. Bernstein et al, Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1997

[5] G. D. Parrington, S. K. Shrivastava, S. M. Wheater and M. C Little, The Desige and Implemntation of Arjuna, WSENIX Computing Systems Journal Vo18. No3, 1995

[6] J. Achibald and J. L. Baer, "Cache Coherence Protocol : Evaluation Using a Mutiprocessor Simulation Model," ACM Trans. on Comp. System, vol.4, no.4, pp273-298, Nov. 1996