분산 공유메모리 시스템을 위한 실시간 제한 프로토콜

(Real-Time Restricted Protocol for Distributed Shared Memory Systems)

전 상 준 † 김 재 훈 †† 김 성 수 †††

(Sang-Joon Jun) (Jai-Hoon Kim) (Sungsoo Kim)

요 약 본 논문에서는 분산 공유 메모리(distributed shared memory) 시스템에서 데이타의 일치성(consistency)을 마감시간(deadline) 이내에 완료하기 위해서 새로운 실시간 프로토콜을 제안한다. 분산 공유 메모리 시스템에서 사용되는 공유 데이타는 여러 시스템에 복제를 할 수 있기 때문에 이들간 일치성을 효과적으로 유지시키기 위한 여러 방법이 연구되어 왔다. 기존의 방법들은 평균 액세스 비용을 단축시키기 위한 것이고 프로토콜에 따라 서로 다른 복사본의 개수를 갖게 되며 일치성 유지를 위한 비용은 시스템이 갖고 있는 복사본의 개수에 따라 증가한다. 각 노드가 서로 다른 데드라인을 갖는 실시간 분산시스템에서는 일치성 유지를 주어진 데드라인이내에 완료하기 위해서는 일치성 유지를 위한 비용을 줄이는 것도 요구되지만 데드라인이 상대적으로 급한 노드에 대한 우선적인 처리가 요구된다. 실시간 프로토콜에서는 각 분산 시스템에서 데드라인이 상대적으로 급한 노드가 항시 복사본을 갖게 하고 전체 복사본의 개수를 제한한다. 시뮬레이션을 통해서 실시간 프로토콜의 성능향상을 확인하였다.

Abstract We suggest a new real-time protocol that can complete the data consistency in a specified deadline on the distributed shared memory system. Various schemes have been studied to keep the data consistency effectively in the distributed shared memory system where data can be copied in different nodes. Because the previous researches just focus on reducing the average access cost and the cost for the data consistency is increased according to the number of copies that depends on the protocols. In order to complete the data consistency within the deadline in the real time distributed system in which each node has different deadline, processing the node having the least slack time first is more important than reducing the average cost for the data consistency. Real-time protocol restricts the number of copies of shared data to reduce the consistency overhead while allowing the node(s) having a little slack to keep the copy of shared data. Simulation results show the performance improvement by using the real-time protocol.

1. 서 론

DSM(Distributed Shared Memory)시스템은 물리적인 공유메모리가 없는 분산시스템 환경에서 가상의 공유메모리 영역을 제공하기 때문에 메시지 패싱(message passing)과 달리 프로그래머가 분산 환경을 크게 고려하지 않고 마치 지역(local) 메모리를 사용하듯 쉽게 분산 프로그램을 작성할 수 있다. 또한 기존의 공유 메모리를 기반으로 한 멀티프로세서 시스템에서 개발된 많은 애플리케이션 프로그램이 별다른 수정 없이 DSM에서 동작할 수 있다. DSM 시스템에서는, 여러 노드들이 네트워크로 연결되고 각 노드들은 프로세서와 지역 메모리로 이루어져 있다[그림 1]. 지역 메모리는 페이지 단위로 구분되고 각 페이지는 여러 노드들에 복사 될 수 있다[1,3].

· 본 연구는 2000년도 교육부 두뇌한국 21 지원 사업에 의한 결과임

† 정 회 원 : 안양과학대학 사무정보처리학부 교수
seeup@netian.com
†† 정 회 원 : 아주대학교 정보통신전문대학원 교수
jaikim@madang.ajou.ac.kr
††† 종신회원 : 아주대학교 정보통신전문대학원 교수
sskim@madang.ajou.ac.kr

논문접수 : 1999년 6월 7일
심사완료 : 2000년 7월 10일

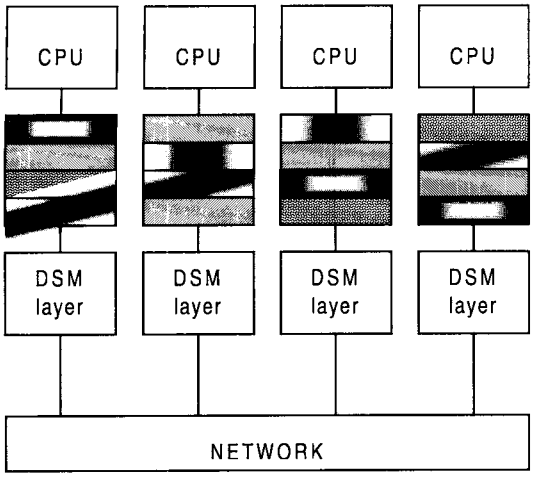

그림 1 분산 공유메모리 구조

이러한 분산 메모리 모델에서는 기본적으로 다음과 같은 두 개의 수행(operation)이 필요하게 된다.

data: =read(address)
: 주어진 주소(address)에 저장된데이타(data)를 리턴(return)한다.
write(data, address)
: 주어진 주소에 데이타를 세트(set)한다.

지역 메모리를 참조할 때는 최대 메모리 접근(access) 속도로 참조가 가능하지만, 참조하려는 페이지가 지역에 복제되지 않았을 때 페이지 오류(fault)가 발생하게 되어 해당 페이지를 다른 노드로부터 전송 받아야 한다. 여러 다른 노드에 동일한 데이타의 복제들이 존재 할 수 있기 때문에 어느 하나의 복사본에 쓰기(write) 수행을 하게 되면 이 수행에 따라 갱신된 정보를 복사본을 보유한 다른 노드에 전송하여 여러 복제의 일치성을 유지시켜야 한다. DSM에서 일치성을 효과적으로 유지시키기 위한 많은 연구가 되어 왔다. 전통적인 방법으로 쓰기-무효(write-invalidate) 또는 쓰기-갱신(write-update) 프로토콜이 사용되었고 성능향상을 위하여 이들 프로토콜을 기반으로 여러 다양한 프로토콜들이 제안되어 왔다[1,2,3,4,5,6,7].

이러한 연구들은 DSM의 일치성을 유지시키는 데 필요한 평균 비용을 단축시키는 데 그 목적이 있기 때문에 시간제약 이내에 업무를 완료 시켜야 하는 실시간 환경에서는 적합하지 않다. 본 논문에서는 (1) 슬랙시간(slack time)이 상대적으로 짧은 타스크(task)가 있는 노드에서 항시 DSM의 복제본을 갖도록 하고, (2) 전체적으로 되도록 복제본의 수를 제한하여 쓰기 수행 시 일치성 유지비용을 줄이어, 시간 제약이내에 일치성을 유지시킬 수 있도록 하는 제한 프로토콜을 제안하고 이의 성능을 평가한다.

일치성 유지를 위하여 DSM에서 기본적으로 사용되는 프로토콜을 요약하면 다음과 같다.

무효 프로토콜(invalidate protocol)

무효 프로토콜은 쓰기에 의해서 데이타의 복제본의 값이 변할 때 그 데이타의 복사본을 가지고 있는 다른 노드에게 무효 메시지를 보내서 무효화 시키는 방식이다. 무효 메시지를 받은 노드에 있는 데이타의 복사본은 다시 사용할 수 없게 된다. 무효화된 복사본을 가진 노드는 다시 데이타를 사용하려 할 때 페이지 오류가 발생되고 이때 복사본을 가진 다른 노드로부터 페이지를 전송 받는다. 읽기(read) 수행에 대해서는 데이타가 바뀌지 않기 때문에 지역에서 데이타를 읽어 들이면 된다.

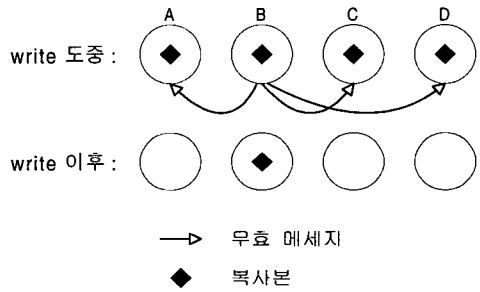

그림 2 무효화 프로토콜

[그림2]는 무효 프로토콜을 사용한 예로서 노드 B에서 쓰기를 할 때, 복사본을 가지고 있는 노드 A, C, D에 무효 메시지를 보냄으로 노드 B만이 그 복사본을 가지게 된다[6].

갱신 프로토콜(update protocol)

갱신 프로토콜은 쓰기 수행에 의해서 데이타가 변하게 되면, 다른 노드에 존재하는 그 데이타의 복사본을 무효화 시키는 것이 아니라 그들을 모두 갱신하는 방식이다. 무효화하는 것보다는 비용(cost)이 많이 발생하지만 다른 노드에 갱신된 복사본은 나중에 페이지 오류 없이 접근 할 수 있게 된다.

[그림 3]은 갱신 프로토콜을 이용한 것으로 노드 B에서 쓰기 할 때 변화된 내용을 복사본을 가진 다른 노드 A, C, D에게 보냄으로써 각 노드에 있는 복사본들이 갱신되게 된다[6].

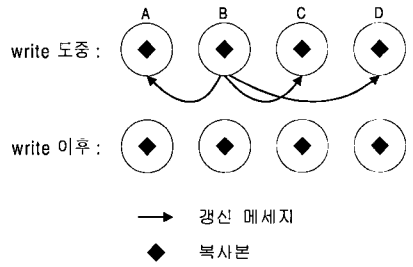

그림 3 갱신 프로토콜

경쟁적 갱신 프로토콜(competitive update protocol)

무효 프로토콜과 갱신 프로토콜의 성능은 애플리케이션 프로그램에서 데이타 사용 패턴과 시스템 변수등에 의해 영향을 받는다. 즉, 어느 한 프로토콜이 모든 경우에 다른 프로토콜보다 성능이 항상 우수할 수 없다. 경쟁적 갱신 프로토콜은 두 가지 프로토콜의 장점을 취한 것으로 사용되지 않는 복사본이 계속 갱신 메시지를 받는 것을 막기 위한 것이다[7]. 이를 위해서 가장 최신 지역노드의 메모리 접근 시점에서 시작하여 다른 노드로부터 갱신 받은 회수에 제한값(limit)을 두어 이 값 한도까지만 갱신 메시지를 받는다. 예를 들어, 갱신 제한값이 3으로 주어지면, 한 노드가 데이타의 사용 없이 다른 노드로부터 3회를 초과하여 갱신 메시지를 받을 경우 자신의 복사본을 무효 시키고 3회 이하면 갱신 시킨다. 중간에 자신이 데이타를 사용하면 갱신된 회수는 0으로 리셋(reset)된다.

갱신 제한값이 0일 경우는 복사본 갱신이 허용되지 않으므로 항시 무효화되어 무효프로토콜과 유사하고 제한값이 무한대이면 존재하는 복사본이 항시 갱신되므로 갱신 프로토콜과 동일하게 된다.

[그림 4]는 경쟁적 갱신 프로토콜의 한 예로서 제한

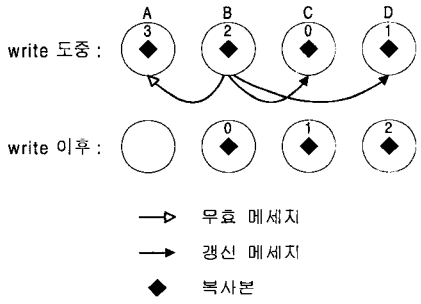

그림 4 경쟁적 갱신 프로토콜

값이 3으로 주어질 경우, 갱신된 회수가 3인 노드 A는 새로운 갱신 메시지를 받을 때 갱신회수가 4로 되어 갱신 제한값 보다 크게 되므로 복사본을 무효화시키고 갱신된 회수가 각각 0, 1인 노드 C, D는 갱신 메시지를 받으면 갱신 회수는 각각 1과 2로 증가되어 갱신 제한값 보다 작으므로 복사본을 갱신시킨다.

확장가능 프로토콜(scalable protocol)

확장가능 프로토콜을 읽기 작업 동안에 복사본의 개수(Nr)를 다음과 같이 제한한다[2]

$r_{min} \leq Nr \leq r_{max}$ with $r_{min}, N_r, r_{max} \in N$

r_{min}, r_{max}은 각각 읽기 작업 동안의 최소 또는 최대 복사본의 수를 나타낸다.

쓰기 수행에 있어서도 다음의 제한값을 갖게 된다.

$w_{min} \leq N_w \leq w_{max}$ with $w_{min}, N_w, w_{max} \in N$

확장가능 프로토콜은 복사본 개수의 하한선과 상한선을 정하여 이 범위를 벗어나지 않게 한다. 하한선을 두는 이유는 일부 노드의 장애 발생시 가용성을 높일 수 있고 상한선을 두는 이유는 복사본을 갱신할 때 발생하는 과다한 오버헤드(overhead)를 줄이어 확장이 용이하기 때문이다[2].

실시간 처리와 관련된 연구들

실시간 DSM 시스템 뿐만 아니라 실시간 데이타베이스 시스템(RTDB)에서도 공유된 데이타를 접근하는 트랜잭션이 올바르게 수행되어야 할 뿐만 아니라 마감시간이라는 시간제약이내에 수행을 완료해야 한다. 트랜잭션 관리가 실시간 데이타베이스 시스템에서 가장 중요한 문제이기 때문에 이에 대한 연구가 많이 되어 왔다. 대부분은 트랜잭션과 그에 따른 마감시간을 집합(set)으로 가정하여 물리적인 자원에 대한 가용성과 스케줄링 문제를 다룬다. 마감시간 실패가 최소로 하기 위해서 어떤 트랜잭션이 주어진 마감시간을 만족시키면서 수행되는지를 알아내는 방식을 따르고 있다. [8,9,10].

또한 실시간 시스템에서 캐쉬를 사용할 때에는 캐쉬의 특성상 예측 불가능성(unpredictability)문제가 있게 된다. 이것은 주로 캐쉬내에서 서로 다른 타스크들의 메모리 블록이 다른 메모리블럭과 충돌하는 타스크간 방해(interference)가 주요한 원인이다. 이를 해결하기 위해서 캐쉬 분할방법과 타이밍(timing) 효과를 분석하기 위한 방법을 이용한다. 캐쉬 분할 방법은 캐쉬 메모리를 서로 분리된(disjoint) 파티션으로 나누어서 하나 또는 그 이상의 파티션이 각 실시간 타스크에 할당하게 된다. 타이밍 효과를 분석하기 위해서 각 수행시점에서 캐쉬 관련 선점 비용을 결정하기 위해서 각 타스크를 분석하

는 방법과 선점으로 인한 시간지연을 분석하는 방법이 있다[15].

캐쉬를 가진 멀티프로세서를 사용하는 시스템에서는 공유데이타의 캐쉬 일치성을 해결하기 위한 하드웨어적인 지원을 제공하기도 한다. 이것은 캐쉬 일치성을 위해서 저장된 캐쉬 태그나 디렉토리을 검사하는데 소요되는 오버헤드도 요구된다[16].

다음은 실시간 시스템에서의 데이타의 일치성을 위한 새로운 프로토콜을 제안한다.

2. 실시간 프로토콜

분산 공유메모리 시스템에서 데이타를 사용할 때 복제된 데이타들 사이의 일치성을 유지시켜야 한다. 앞에서 설명한 무효, 갱신, 경쟁적 갱신, 또는 확장가능 프로토콜을 사용해서 이를 해결하게 된다. 이때 발생하는 통신 비용으로 인해서 데이타에 대한 읽기 또는 쓰기 수행에 대한 처리가 주어진 마감시간 이내에 완료하지 못하게 되는 경우가 생긴다. 기존의 데이타 일치성을 위해 사용되었던 방식은 평균 통신 오버헤드를 줄이는데 초점을 맞추었기 때문에 마감시간을 고려한 실시간 이용에는 적합하지 못하다. 본 논문에서는 평균 수행시간보다는 마감시간 이내에 끝내야 하는 실시간 응용을 위한 DSM 일치성 유지 방안을 제안한다.

본 논문에서는 실시간 시스템에서의 주기적인 타스크를 모델로 한다. 주기적인 타스크는 일정시간 간격으로 동일한 일(job)을 수행하는 계산 모델로서 일정한 비트 비율의 비디오 스트림(stream)의 전송, 센서 데이타의 주기적인 샘플링 또는 샘플링 데이타의 처리등이 이러한 주기적인 타스크로서 정의 될 수 있다. 주기적 타스크의 주기는 인접한 일들의 릴리즈 타임(release time) 간격중 최소길이로 정해지고 실행시간(execution time)은 모든 일들 중 최대 실행시간으로 정해진다[12].

본 논문에서 제안하는 실시간 프로토콜의 적용 가능성을 알아보기 위하여 우선적으로 다음과 같은 간단한 시스템 모델을 적용하였다.

i) 각 노드마다 한 개의 타스크가 수행되며 모든 타스크는 동일한 주기를 갖는다.

ii) 각 타스크의 수행시간(일치성을 위한 통신 오버헤드는 제외)은 동일하지 않다. 따라서 각 타스크는 다른 슬랙시간을 갖는다.

iii) 각 타스크는 주기마다 DSM을 이용하여 공유 데이타를 읽기나 쓰기를 한다. 이때 데이타의 일치성을 유지하여야 한다.

시스템 모델에서 설명한 것과 같이 슬랙시간이 각 노드별로 다르기 때문에 각 노드에서 읽기 또는 쓰기 수행에 대한 처리에서 마감시간을 고려하지 않을 경우 슬랙시간이 상대적으로 적은 노드에 요청된 수행에 대한 처리는 마감시간을 실패 할 가능성이 다른 노드들 보다 높게 된다. 페이지 오류가 발생하게 될 경우 이를 위한 비용이 마감시간까지의 시간보다 크게 되면 이를 마감시간안에 처리하기는 불가능하게 된다. 따라서 슬랙시간이 상대적으로 적은 노드에 우선순위(priority)를 주어야 하는 것이 필요하다. 우선순위를 주는 방법으로 슬랙시간이 가장 적은 노드에 데이타의 복사본을 항상 존재하게 함으로써 다른 노드보다 데이타 일치성을 위한 비용을 줄이게 하여 마감시간을 실패할 확률을 줄일 수 있다.

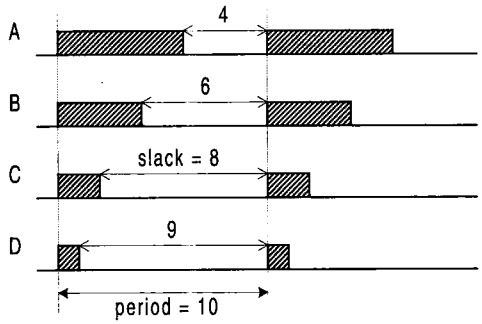

그림 5 주기적 타스크

[그림 5]와 같이 모든 노드는 동일한 주기를 가지며 각 주기마다 하나의 태스크가 수행된다. 각 노드의 슬랙시간을 각각 4, 6, 8 그리고 9라고 하면 노드 A의 슬랙시간이 가장 작게 되므로 이 노드를 항상 복사본을 갖는 노드로 택하게 된다.

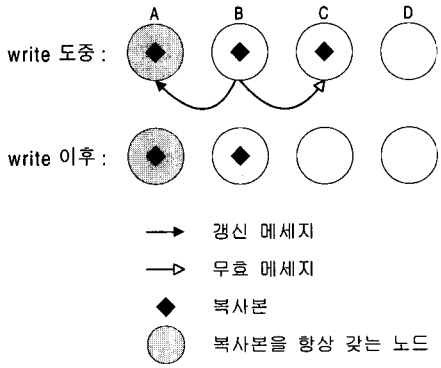

그림 6 제한 프로토콜

[그림 6]과 같이, 노드 B에 쓰기가 하게 되면, 복사본을 갖고 있는 노드 C,D에 각각 무효 메시지를 보내 복사본을 무효화 시킨다. 노드 A도 역시 복사본을 가지고 있지만, 항상 복사본을 갖는 노드로 지정되어 있으므로 이 노드에게는 갱신 메시지를 보내게 된다. 즉, 현재 메모리를 사용하는 노드와 복사본을 항시 갖는 노드(슬랙시간이 적은 노드)만이 복사본을 소유하도록 복사본의 개수를 제한하는 방법을 사용한다.

제한 프로토콜의 주요함수를 정리하면 다음과 같다. 분산 페이지소유자 관리방법[3]을 가정하였고 첫번째 읽기(같은 페이지를 다른 노드가 사용한 후 해당 노드의 첫번째 읽기)도 감지하기 위하여 보호모드(protected mode)를 사용한다.

```
NC: 복사본을 항시 갖는 노드
page_fault() /* 페이지 오류 */
{
  1. 페이지 소유자에게 페이지를 요구하고
     페이지를 수신 받는다.
  2. 페이지 소유자는 NC가 아닌 원격 노드
     가 복사본을 갖고 있으면 무효화 시킨다.
}
read_interrupt_handler()
/* 첫번째 읽기에서 발생 */
{
  1. 복사본이 없으면 page_fault
  2. NC가 아닌 원격 노드에 복사본이
     존재하면 무효화 시킨다.
     /* handler routine 후 읽기 수행 */
}
write_interrupt_handler()
/* 첫번째 쓰기에서 발생 */
{
  1. 복사본이 없으면 page_fault
     /* handler routine 후 쓰기 수행 (원격 NC
     는 갱신하고 NC가 아닌 원격 노드에 복
     사본이 존재하면 무효화 시킨다) */
}
```

3. 제한 프로토콜의 비용 분석

복사본 제한 프로토콜에서 메모리 사용 시 일치성 유지를 위해 필요한 비용을 수학적으로 분석하였다. 비용 계산에 관련된 파라미터는 다음과 같이 정의된다.

R : 읽기 비율

W : 쓰기 비율 ($W = 1-R$)

S : 슬랙시간

N : 전체 노드의 수

C : 항상 복사본을 갖는 노드의 수

i : 무효 메세지 또는 제어 메시지 전송 비용

u : 갱신 메세지 전송 비용

p : 페이지 오류 시 페이지를 수신 받는데 필요한 비용

각 노드의 메모리 시용 빈도는 동일하다고 가정하였고 각 노드는 현재 복제 노드의 집합과 항상 복제를 갖는 노드의 집합에 관한 정보를 갖고 있다고 가정하였다. (실제로는 페이지 소유자(owner)가 이를 관리하고 각 노드는 페이지 소유자를 통하여 무효 또는 갱신 메시지를 보내며 페이지 오류발생시 페이지 소유자에게 페이지 전송을 요구하지만 본 수학적 분석에서는 페이지 소유자를 통하는 비용은 고려하지 않는다.)

- 복사본을 항시 갖는 노드의 읽기

$$\frac{N-C}{N} i$$

바로 이전에 복사본을 항상 갖는 노드가 아닌 노드가 메모리를 사용했을 경우(확률= $(NC)/N$) 의 복사본을 무효화 시켜야 한다(비용 = i). 이전에 복사본을 항시 갖는 노드가 메모리를 사용했다면 추가 비용은 없다.

- 복사본을 항시 갖는 노드의 쓰기

$$(C-1)u + \frac{N-C}{N} i$$

복사본을 항시 갖는 다른 노드에게 갱신 메시지를 보내야 하고(비용= $(C-1)u$), 바로 이전에 복사본을 항시 갖는 노드가 아닌 노드가 메모리를 사용했다면(확률= $(N-C)/N$)) 그 복사본은 무효화 시켜야 한다 (비용 = i).

- 복사본을 항시 갖는 노드가 아닌 노드의 읽기

$$\frac{N-1}{N} p + \frac{N-C-1}{N} i$$

바로 이전에 자신이 메모리를 사용하지 않았다면(확률 = $(N-1)/N$) 페이지 오류가 발생한다(비용 = p), 바로 이전에 항시 복사본을 갖는 노드가 아닌 노드(자신을 제외)가 메모리를 사용했다면(확률 = $(N-C-1)/N$) 그 복사본을 무효화 시킨다(비용 = i).

- 복사본을 항시 갖는 노드가 아닌 노드의 쓰기

$$\frac{N-1}{N} p + Cu + \frac{N-C-1}{N} i$$

바로 이전에 자신이 메모리를 사용하지 않았다면(확

률 = $(N-1)/N$) 페이지 오류처리를 먼저 해야 한다(비용 = $((N-1)/N)p$). 복사본을 항시 갖는 노드에게 갱신 메시지를 보내야 하고(비용 = Cu) 만일 바로 이전에 항시 복사본을 갖는 노드가 아닌 노드(자신은 제외)가 메모리를 사용했다면(확률 =$(N-C-1)/N$) 그 복사본을 무효화 시킨다(비용 = i).

위의 네가지 경우에서 평균 메모리 사용 비용은 다음과 같다.

$$\frac{C}{N}\left[R\frac{N-C}{N}i+W(C-1)u+W\frac{N-C}{N}i\right]+$$

$$\frac{N-C}{N}\left[\frac{R(N-1)}{N}p+\frac{R(N-C-1)}{N}i+W\frac{N-1}{N}p+WCu+\frac{W(N-C-1)}{N}i\right]$$

복사본을 제한할 때 고려사항으로 (1) 복사본 제한한도를 얼마로 해야 하고, (2) 어는 노드가 복사본을 갖는지 결정해야 한다.

상대적으로 슬랙이 적은 노드가 복사본을 항시 갖도록 하면, 지역 메모리 접근을 빨리 할 수 있는 장점이 있다. 그러나 너무 많은 노드가 항시 복사본을 갖게 되면 슬랙이 적은 노드가 쓰기 동작을 할 때 갱신 메시지를 많은 노드에게 개별적으로 보내야 하므로 일치성 유지를 위해 많은 비용이 필요하여 제약시간을 준수하지 못하는 결과를 초래할 수 있다. 따라서 적정수준의 복사본 개수제한을 설정해야 한다.

최적의 C 를 구하는 알고리즘은 위에서 분석한 메모리 사용 비용을 바탕으로 다음과 같이 고안된다.

슬랙시간이 적은 C 개의 노드가 항시 복사본을 갖는다고 가정하고 모든 $C(1\le C\le N)$에 대하여 평균 마감시간 성공비율을 계산한 후 최적의 C 값을 구한다.

1. 노드 j $(1\le j\le N)$의 읽기 비율이 R_j, 쓰기 비율이 W_j, 슬랙이 S_j 라고 가정하고 각 노드에 대하여 마감시간 성공비율을 마감시간 성공비율 계산 알고리즘에 의하여 계산한다.

2. 각 노드 j에 대하여 계산된 마감시간 성공 비율을 바탕으로 평균 마감시간 성공비율을 계산한다.

마감시간 성공비율 계산 알고리즘
입력: N, C, R_j, W_j, S_j, i, u, p
I. 복사본을 항시 갖는 노드의 경우
- 읽기 (R_j의 비율로 계산)
  ① $i\le S_j$: 항시 마감시간 성공
  ② $S_j<i$: C/N의 비율로 마감시간 성공
- 쓰기 (W_j의 비율로 계산)
  ① $i+(C-1)u\le S_j$: 항시 마감시간 성공
  ② $(C-1)u\le S_j<(C-1)u+i$: C/N의 비율로 마감시간 성공
  ③ $S_j<(C-1)u$: 항시 마감시간 실패

II. 복사본을 항시 갖는 노드가 아닌 노드의 경우
- 읽기 (R_j의 비율로 계산)
  ① $p+1\le S_j$: 항시 마감시간 성공
  ② $p+S_j<p+i$: C/N의 비율로 마감시간 성공
  ③ $i\le S_j<p$: I/N의 비율로 마감시간 성공
  ④ $S_j<i$: I/N의 비율로 마감시간 성공
- 쓰기 (W_j의 비율로 계산)
  ① $p+Cu+i<S_j$: 항시 마감시간 성공
  ② $p+Cu\le S_j<p+Cu+i$: C/N의 비율로 마감시간 성공
  ③ $i+cu\le S_j<p+Cu$: I/N의 비율로 arkatlrks 성공
  ④ $Cu\le S_j<i+Cu$: I/N의 비율로 마감시간 성공
  ⑤ $S_j<Cu$: 항시 마감시간 실패

각 C($1\le C\le N$) 에 대하여 모든 노드 ($1\le j\le N$)의 마감시간 성공비율을 계산해야 하므로 제한 프로토콜의 시간 복잡도는 $O(N^2)$이다.

4. 성능 평가 및 분석

실시간 프로토콜의 성능을 측정하기 위하여 시뮬레이션 방법을 이용하였다. 시뮬레이션을 위해서는 자체로 제작한 프로그램을 사용하였다. 실시간 분산 공유메모리의 비용에 영향을 주는 변수들은 다음과 같다.

R : 전체 공유메모리 접근(읽기와 쓰기) 중 읽기가 발생하는 비율

S : 각 노드의 슬랙시간

C : 항상 복사본을 갖는 노드의 수

성능평가를 위해서 다음과 같은 시스템 변수를 사용하여 실시간 프로토콜의 성능을 기존 프로토콜(무효, 갱신, 경쟁적 갱신)의 성능과 비교하였다. 사용된 환경변수는 다음과 같다.

- 노드 개수 : 4, 8, 16개
- 주기(P) : 50 (time unit)
- 노드의 개수별 슬랙시간(S)

노드수	노드별 슬랙시간	평균슬랙
4	5, 10, 15, 20	12.5
8	5, 7, 9, 10, 12, 14, 16, 20	11.625
16	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20	12.5

- 비용

수행	비용
무효	1
갱신	3
페이지 오류	8

• 총 메모리 접근 수 : 10000 회

위의 환경 변수에서 한 주기당 수행시간의 비율을 90%에서 60%로 가정하였다. 따라서 각 주기당 슬랙시간은 주기의 10%에서 40%로 정해진다. 주기가 50이므로 한 노드가 갖는 가장 짧은 슬랙시간은 5이고 가장 긴 슬랙시간은 20이다. 또한 무효 비용을 1, 페이지 오류 비용을 8로 설정한 이유는 8노드로 구성된 워크스테이션 클러스터(10M bps Ethernet으로 연결)에서 DSM(Quarks, [13, 14])상에서 작은 메시지를 보내는데 소요시간이 평균 3.8 msec, 페이지(4kbytes) 오류시 수신시간이 30 msec정도 되므로 상대적인 비용을 1 대 8로 하였다.

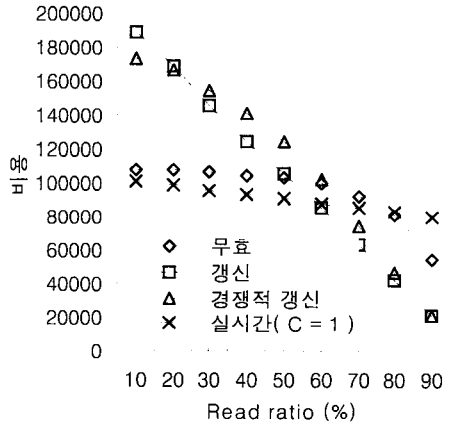

그림 7 총비용

[그림 7]은 8개 노드를 이용해서 실험한 것으로 기존 프로토콜(무효, 갱신, 경쟁적 갱신)과 실시간 프로토콜을 사용할 때 발생하는 총 비용을 측정한 것이다. 실시간 프로토콜의 복사본개수(C=1)는 1개로 가정하였다. 경쟁적 갱신 프로토콜은 읽기의 비율이 적을 때 무효 프로토콜보다 비용을 많이 발생시키고 갱신 프로토콜보다는 적은 비용이 필요하다. 반면 읽기 비율이 많을 때는 갱신 프로토콜과 같아 지면서 무효 프로토콜보다 적은 비용이 소요되는 것을 보여준다. 실시간 프로토콜을 사용하면 읽기의 비율에 따른 비용의 변화가 크지 않음을 알 수 있다. 읽기 비율이 많을 때에 비용이 다른 프로토콜과 같이 낮아지지 않는 이유는 읽기 할 때도 복사본의 개수를 조절하기 위해서 다른 노드에 있는 복사본을 제거하기 위한 메시지를 보내기 때문이다. 실시간 프로토콜에서 성능평가의 기준은 메모리 액세스를 위한 비용이 아니라 시간 제약이내에 완료하는 것이기 때문에 다음 실험에서는 이를 비교하였다.

실시간 시스템에서 데드라인의 만족여부가 가장 중요한 성능평가의 요소중 하나이다. 노드의 개수를 4, 8, 그리고 16개로 변화시키면서 기존의 프로토콜과 실시간 프로토콜간의 데드라인 준수 여부를 측정하였다. 각 노드의 슬랙시간은 앞에서 가정한 것을 사용하였다.

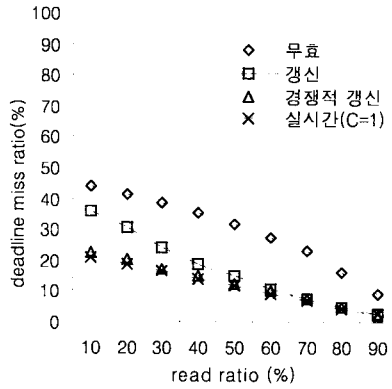

그림 8 마감시간 실패 비율(4 노드, 불균등 슬랙)

[그림8]은 4개의 노드에서 읽기 비율을 변화시키면서 각 프로토콜별 마감시간 실패비율을 보여준다. 실시간 프로토콜의 C(항시 복사본을 갖는 노드의 수)는 1개이다. 무효프로토콜을 사용한 경우는 다른 프로토콜을 사용한 경우보다 전반적으로 많은 마감시간 실패 비율을 발생시키다. 경쟁적 갱신과 실시간 프로토콜을 사용한 경우는 거의 동일하게 적은 마감시간 실패 비율을 보여준다. 노드의 수가 4개이므로 무효프로토콜을 사용하면 페이지 오류로 인한 마감시간 실패 비율이 다른 프로토콜을 사용한 것보다 많이 발생하게 된다.

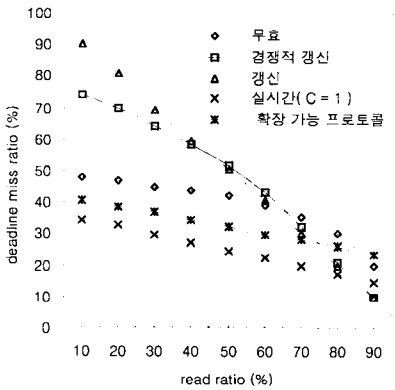

그림 9 마감시간 실패 비율(8 노드, 불균등 슬랙)

[그림 9]는 8개 노드를 이용해서 실험한 것이다. 무효 프로토콜을 사용한 경우는 읽기 비율이 높을 때 갱신 프로토콜보다 마감시간 실패가 많이 발생하고 읽기의 비율이 적을 때 갱신 프로토콜보다 마감시간 실패가 줄어든다. 한계값이 3인 경쟁적인 갱신 프로토콜은 무효 프로토콜과 갱신 프로토콜이 서로 보완된 형태를 보여준다. 즉 읽기 빈도가 낮을때는 갱신프로토콜보다 적은 마감시간 실패를 발생하고 읽기 비율이 높을 경우는 무효 프로토콜보다 적은 마감시간 실패 비율을 보여준다.

읽기 비율이 적을 때, 즉 쓰기 비율이 많을 때는 페이지 오류로 인한 마감시간 실패보다 각 노드에 존재하는 복사본을 갱신하기 위해 필요한 비용으로 인한 마감시간 실패가 많게 된다. 따라서 한계값에 의해서 제한된 복사본의 수를 갖는 경쟁적 갱신 프로토콜을 이용할 때 갱신 프로토콜보다 적은 마감시간 실패를 갖는다. 반대로 읽기 비율이 많을 때는 복사본의 수가 상대적으로 많은 갱신 또는 경쟁적 갱신 프로토콜이 무효 프로토콜 또는 실시간 프로토콜보다 적은 마감시간 실패 비율을 갖게 된다.

실시간 프로토콜(C=1)을 사용하면, 슬랙시간이 상대적으로 작은 노드에 항상 복사본을 갖게 함으로써 다른 프로토콜을 사용한 것에 비해서 마감시간 실패가 현저하게 줄어 들게 된다. 확장 가능 프로토콜을 사용한 경우는 복사본을 1개만을 갖게 제한한 것으로 복사본을 갖는 노드는 각 노드의 슬랙시간과 관계없이 8개의 노드들 중에서 임의로 하나가 정해진다. 위의 결과는 이것의 평균값을 구한 것으로 실시간 프로토콜보다 많은 마감시간 실패를 보여준다.

[그림10]은 16개의 노드를 이용해서 마감시간 실패 비율을 측정한 것이다. 무효 또는 갱신프로토콜등 기존의 프로토콜을 사용한 경우는 앞의 실험과 유사하게 데드라인 발생했다. 실시간 프로토콜을 사용한 경우는 다른 프로토콜을 이용한 것보다 적은 마감시간 실패 비율을 나타낸다. C가 1일 경우는 2인 경우에 비해서 읽기 비율이 적을 때 적은 마감시간 실패 비율을 발생시키고 읽기 비율이 높으면 C가 2인 경우가 적은 마감시간 실패 비율을 발생시킨다. 읽기 비율이 적을 때는 복사본이 많으면 갱신 비용이 증가하므로 복사본의 개수를 적게 갖는 프로토콜이 상대적으로 적은 마감시간 실패 비율을 나타낸다.

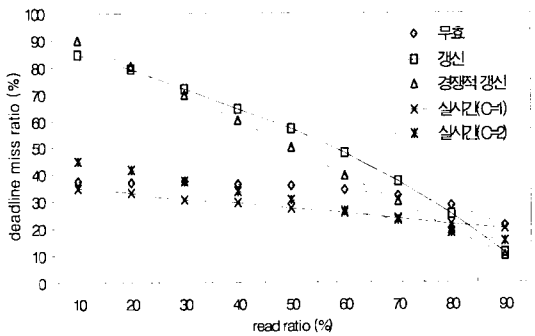

그림 10 마감시간 실패 비율(16노드, 불균등 슬랙)

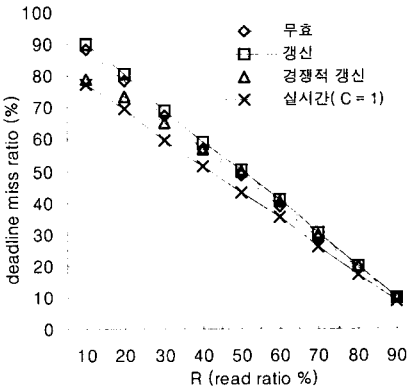

그림 11 마감시간 실패 비율(8노드, 짧은 균등 슬랙)

[그림 11]은 8개 노드의 슬랙시간을 모두 10으로 가정하고 측정한 결과값이다. 무효, 경쟁적 갱신, 갱신 프로토콜은 거의 같은 마감시간 실패를 보여주고 실시간 프로토콜을 사용한 것은 그보다 적은 마감시간 실패를

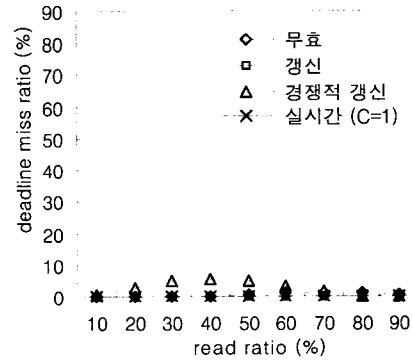

그림 12 마감시간 실패 비율(8노드, 긴 균등 슬랙)

보였다. 이는 복사본 수를 제한하여 갱신에 필요한 비용을 줄였기 때문이다.

[그림 12]는 8개 모든 노드의 슬랙시간을 26으로 정한 실험으로 거의 모든 프로토콜이 마감시간 실패를 발생시키지 않는 반면에 경쟁적 프로토콜을 사용한 경우만이 마감시간 실패를 조금 발생시킨다. 모든 노드의 슬랙시간이 충분하면 실시간 프로토콜이 아니더라도 마감시간 실패가 거의 발생되지 않음을 알 수 있다.

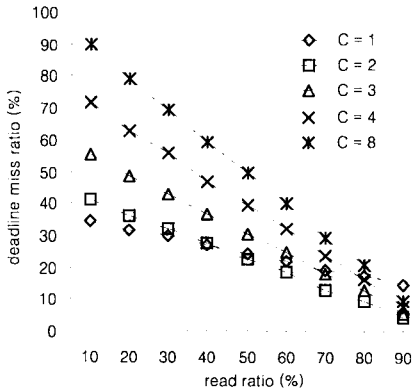

그림 13 마감시간 실패 비율(8노드, 불균등 슬랙, 복사본수 변화)

[그림 13]은 8개의 노드 중에서 항상 복사본을 갖는 노드의 개수를 변화시키면서 마감시간 실패를 측정한 것으로, 읽기 비율이 40% 이하 일 때는 하나의 노드만 복사본을 항상 갖게 하는 것이 가장 적게 마감시간 실패를 하고 복사본을 항상 갖는 노드의 개수가 증가하면, 마감시간 실패도 많아지게 된다. 읽기 비율이 40% 이상일 때는 2개의 노드가 복사본을 갖을 때 가장 좋은 결과를 얻는다. 8개의 노드 모두가 복사본을 갖게 되는 것은 갱신 프로토콜과 거의 동일한 결과를 보여준다. 또한, 항시 복사본을 갖는 최적의 노드 수는 시스템 변수에 따라 다르게 된다.

[그림14]와 같이 8개의 노드에서 갱신비용을 변화시키면서 마감시간 실패비율을 측정한 것이다. 갱신 비용을 2, 4, 6, 8로 증가시켜도 프로토콜별 상대적 성능 우위 대체적으로 전반적으로 앞의 결과와 비슷하였다. 실시간 프로토콜(C=1)을 사용한 경우는 읽기 비율이 높은 경우(90% 이상)를 제외하고는 다른 프로토콜을 사용한 경우보다 상대적으로 적은 마감시간 실패 비율을 보여준다. 또한 경쟁적 갱신 프로토콜은 무효 프로토콜과 갱신 프로토콜의 혼합형임도 보여준다.

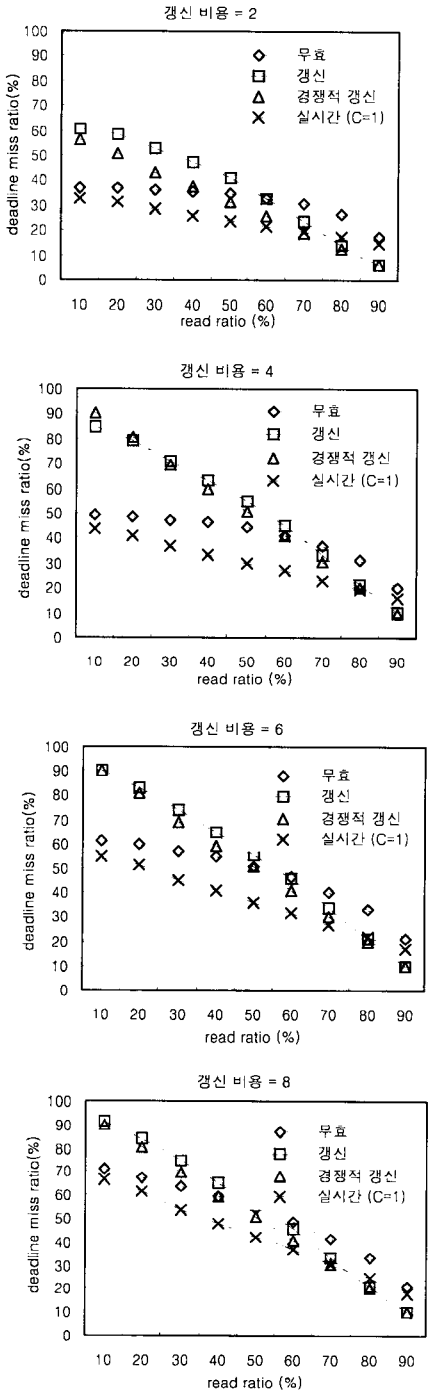

그림 14 갱신 비용이 변할 때의 마감시간 실패 비율 (8노드, 불균등 슬랙)

4. 결 론

본 논문에서는 실시간 처리 환경에 적합한 DSM 프로토콜을 제안하였다. 실시간 DSM하에서 각 노드들이 각기 다른 슬랙시간을 가질 때, 슬랙시간이 적은 노드에 항시 데이타의 복사본을 갖도록 하고 다른 노드들은 메모리 액세스를 할 때만 복사본을 갖게 하였다. 이에 따라 슬랙시간이 적은 노드는 일치성유지를 위한 비용을 줄이게 되어 마감시간 실패를 줄일 수 있다. 제안된 실시간 프로토콜을 이용할 때 마감시간을 실패하는 빈도가 줄어 들게 됨을 시뮬레이션을 통해 확인하였다.

참 고 문 헌

[1] M. Stumm and S. Zhou, Algorithms implementing distributed shared memory, *IEEE Computer*, vol. 23, pp. 54-64, May 1990.

[2] O. Theel and B. Fleisch, Design and analysis of highly available and scalable coherence protocols for distributed shared memory systems using stochastic modeling, in *Proc. of International Conference on Parallel Processing*, vol. I, pp. 126-130, Aug. 1995.

[3] K. Li and P. Hudak, Memory coherence in shared virtual memory systems, *ACM Transactions on Computer Systems*, vol. 7, pp. 321-359, Nov. 1989.

[4] H. Grahn, P. Stenstrom, and M. Dubois, Implementation and evaluation of update-based cache protocols under relaxed memory consistency models, *Future Generation Computer Systems*, vol. 11, pp. 247-271, June 1995.

[5] A. Lebeck and D. Wood, Dynamic self-invalidation: Reducing coherence overhead in shared-memory multiprocessors, in *Proc. of the 22nd Annual International Symposium on Computer Architecture*, pp. 48-59, 1995

[6] J. B. Carter, *Efficient Distributed Shared Memory on Multi-Protocol Release Consistency*, Ph. D. dissertation, Rice University, Sept. 1993.

[7] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, Competitive snoopy caching, in *Proc. of the 27th Annual Symposium on Foundations of Computer Science*, pp. 244-254, Oct. 1986.

[8] Young-Kuk Kim and Sang H. Son, Predictability and Consistency in Real-Time Database Systems, *Advances In Real-Time Systems*, pp. 509-531, Prentice-Hall, 1995.

[9] N. Soparkar, H. F. Korth, and A. Silberschatz, "Databases with Deadline and Contingency Constraints," *IEEE Transactions on Knowledge and Data Engineering* 7, 4 (August 1995), 552-565.

[10] Rhan Ha and Jane W. S. Liu, Validating timing constraints in multiprocessor and distributed real-time systems, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, 1993.

[11] Anidnya Datta and Igor R. Viguier, Providing Real-Time Response, State Recency and Temporal Consistency in Databases for Rapidly Changing Environments, Information System Vol. 22. No. 4, pp.171-198, 1997

[12] J. Liu, Real-Time Systems, Prantice Hall, 2000

[13] D. Khandekar, Quarks: Portable dsm on unix, manuscript, Dept. of Computer Science, University of Utah, 1995.

[14] M. Swanson, L.Stoller, J.Carter, Making distributed shared memory simple, yet simple, proc. of the Third International Workshop on High-Level Parallel Programming Models and Supportive Environments, March 1998.

[15] Chang-Gun Lee, Yang-Min Seo, Seongsoo Hong, Minsuk Lee, Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling, IEEE Transaction on Computers, Vol. 47, No. 6, pp. 700-713, June 1999.

[16] Alfredo Romagosa, Cache Coherence Issues for Real-Time Multiprocessing, Embedded Systems Journal, Vol. 10, No. 2, 1997.

전 상 준

1998년 경기대학교 수학과 학사. 2000년 아주대학교 컴퓨터공학과 석사. 2000년 ~ 현재 안양과학대학 전산사무정보학부 강사. 관심분야는 운영체제, 분산처리 시스템

김 재 훈

정보과학회논문지 : 시스템 및 이론 제 27 권 제 8 호 참조

김 성 수

정보과학회논문지 : 시스템 및 이론 제 27 권 제 8 호 참조