원격 로깅 기법을 이용하는 고장 허용 소프트웨어 분산공유메모리 시스템의 구현

(An Implementation of Fault Tolerant Software Distributed Shared Memory with Remote Logging)

박 소 연 † 김 영 재 † 맹 승 렬 ††
(Soyeon Park) (Youngjae Kim) (Seung Ryoul Maeng)

요 약 최근에 소프트웨어 분산공유메모리 시스템은 그 성능이 높아짐에 따라 큰 규모의 클러스터 상에서 사용되는 경우가 많아졌다. 그러나 시스템 규모가 커지면서 고장이 발생하는 가능성도 높아졌다. 시스템의 가용성을 높이기 위하여 고장 허용 기능을 제공하는 분산공유메모리 시스템이 요구되었으며 메시지 로깅에 대한 많은 연구가 이루어져 왔다. 본 논문에서는 고속의 네트웍을 이용하여 복구에 필요한 메시지들을 원격 노드의 메모리에 로깅하는 방법을 제안한다. 원격 로깅은 정상 수행 동안 빈번한 디스크 접근을 요구하지 않으므로 오버헤드가 적다. 또한 로그를 유지하는 백업 노드들이 고장나지 않은 경우 다중 노드의 고장을 허용하여, 분산공유메모리 시스템의 신뢰성을 높인다. 본 논문에서는 FT-KDSM (Fault Tolerant KAIST DSM) 시스템을 설계하고 구현하여 원격 로깅의 성능을 보이고 고장으로부터의 복구 시간을 보인다.

키워드 : 소프트웨어 분산공유메모리, 고장 허용, 메시지 로깅, 클러스터 시스템

Abstract Recently, Software DSMs continue to improve its performance and scalability. As Software DSMs become attractive on larger clusters, the focus of attention is likely to move toward improving the reliability of a system. A popular approach to tolerate failures is message logging with checkpointing, and so many log-based rollback recovery schemes have been proposed. In this work, we propose a *remote logging* scheme which uses the volatile memory of a remote node assigned to each node. As our remote logging does not incur frequent disk accesses during failure-free execution, its logging overhead is not significant especially over high-speed communication network. The remote logging tolerates multiple failures if the backup nodes of failed nodes are alive. It makes the reliability of DSMs grow much higher. We have designed and implemented the FT-KDSM(Fault Tolerant KAIST DSM) with the remote logging and showed the logging overhead and the recovery time.

Key words : Software Distributed Shared Memory, Fault Tolerance, Message Logging, Cluster System

1. 서 론

소프트웨어 분산공유메모리(Software Distributed Shared Memory) 시스템은 각 노드에 분산된 메모리를 공유메모리로 인식할 수 있도록 사용자에게 가상의 이미지를 제공해준다. 최근까지 분산공유메모리 시스템의 성능, 확장성, 프로그램의 용이성을 향상시키기 위해 많은 연구들이 이루어져 왔으며, 그 결과 분산공유메모리 시스템은 큰 규모의 클러스터 상에서 효율적인 병렬처리를 위해 이용될 수 있었다.

한편, 클러스터 시스템의 규모가 점차 커짐에 따라 시스템에서 고장이 발생할 가능성이 높아지게 되었다. 분산공유메모리 시스템에서는 노드들 사이에 데이타가 공유되므로 한 노드에서 고장이 발생해도 전체 노드의 재수행이 요구된다. 따라서 오랜 기간 동작하거나 높은 가용성을 요구하는 응용 프로그램들을 위해 분산공유메모리 시스템에서 고장 허용성을 제공해 주는 것이 중요하다.

고장 허용을 위해 일반적으로 많이 사용되는 방법은

† 비 회 원 : 한국과학기술원 전자전산학과
sypark@calab.kaist.ac.kr
yjkim@calab.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
maeng@calab.kaist.ac.kr
논문접수 : 2003년 5월 16일
심사완료 : 2004년 3월 11일

체크포인팅(checkpointing)과 롤백(rollback) 이다[1-5]. 이는 프로그램 수행 중 주기적으로 시스템 이미지를 안전한 저장 장치에 저장하고, 고장이 발생했을 때 체크포인팅 시점으로 되돌아가서 수행을 계속하는 방법이다. 분산공유메모리 시스템에서는 공유 페이지를 매개로 여러 노드 사이에 의존 관계(dependency)가 형성되므로 올바른 시스템 상태로 복구하기 위해 한 노드의 롤백이 다른 노드의 롤백을 연속적으로 요구하는 도미노 현상이 발생할 수 있다. 이를 방지하기 위하여, 각 노드가 독립적으로 체크포인팅을 수행하고 더불어 각 노드 간에 교환되는 메시지를 로깅(logging)하는 방법들이 많이 연구되었다[1-5]. 그러나 로그 저장을 위해 디스크를 이용하는 방법들은 빈번한 디스크 접근으로 인하여 정상 수행 시 성능을 크게 감소시키는 문제를 가진다[1,2]. 반면, 로그를 메시지 전송 측의 메모리에 저장하는 방법들은 한 노드의 고장만을 가정함으로 적용에 제한적이다[3,4].

본 논문에서는 오버헤드가 적은 원격 로깅 기법과 복구 기법을 제안한다. 원격 로깅은 각 노드 마다 로그 홈(log home)이라는 특정한 노드를 할당하여 로그 홈의 메모리에 로깅하는 방법이다. 본 방법은 정상 수행 중 빈번한 디스크 접근을 요구하지 않는다는 장점을 가진다. 특히, 고속의 네트웍 환경에서 사용자 계층 통신 프로토콜의 원격 메모리 쓰기 기능[7]을 활용할 때 로깅으로 인한 성능 감소를 최소화 할 수 있다. 또한, 원격 로깅에서는 한 노드와 그의 로그 홈을 제외한 다른 노드들에서 동시에 고장이 발생한 경우에도 복구가 가능하다.

본 논문에서는 원격 로깅을 사용하는 FT-KDSM (Fault Tolerant KAIST DSM) 시스템에 대해 설명하며 실험을 통하여 원격 로깅의 오버헤드와 고장 난 노드 수의 증가에 따른 복구 시간을 보인다. 논문의 구성은 다음과 같다. 2장에서는 시스템의 개요를 기술한다. 3장에서는 원격 로깅에 대해 설명하고 4장에서는 복구 방법을 설명한다. 5장에서는 시스템의 성능과 복구 시간을 보이고 6장에서 결론 및 향후 연구 계획에 대해 기술한다.

2. Fault Tolerant KDSM(FT-KDSM)의 개요

2.1 홈 기반 분산공유메모리 시스템

소프트웨어 분산공유메모리 시스템은 물리적으로 분산된 프로세서 상에서 병렬 프로그램을 수행할 수 있도록 가상의 공유 메모리 이미지를 사용자에게 제공해 준다. 사용자는 공유 메모리가 할당된 노드 위치 및 노드 간의 명시적인 데이타 전송에 관여하지 않고 프로그램을 작성할 수 있다. 이 때, 하부의 분산공유메모리 시스템은 가상 메모리 매카니즘과 노드 간의 명시적 메시지 전송을 통하여 공유 메모리의 일관성을 유지한다. 본 논문에서 제안하는 FT-KDSM은 기존에 구현된 분산공유메모리 시스템인 KDSM-V[6](KAIST DSM over VIA)를 바탕으로 한다. KDSM-V에서는 메모리 일관성 유지를 위해 Home-based Lazy Release Consistency (HLRC)[8] 모델을 사용하고 노드 간의 통신을 위해 VIA 표준에 의거하여 구현된 VI-GM[7]을 이용한다. HLRC는 동기화 시점에서 메모리의 일관성이 보장되는 모델의 일종으로써, 일관성 유지를 위한 네트웍 오버헤드가 적고 사용자 계층 통신 프로토콜과 함께 효과적으로 구현 가능하므로 기존 분산공유메모리 시스템에서 많이 채택된다.

그림 1은 HLRC의 동작을 나타낸다. HLRC에서 공유 페이지는 지정된 페이지 홈을 가지고 있으며 홈은 항상 최신의 내용을 유지한다. 홈이 아닌 노드는 페이지 폴트 시에 홈으로부터 페이지를 얻어오고, 쓰기를 수행한 경우 동기화 시점에서 diff 메시지를 홈에게 보내 페이지를 갱신한다. diff는 페이지 중 변경된 부분의 정보만을 가지는 데이타 구조로써, 쓰기를 수행하기 전 만들어둔 복사본(twin)과 쓰기 후의 페이지와의 비교를 통해 생성된다. HLRC에서는 동기화 연산들 사이의 구간을 하나의 인터벌(interval)로 정의하는데 diff 메시지에는 쓰기가 수행된 때의 인터벌 정보가 포함된다. 홈 노드는 페이지 마다 diff 버전 벡터를 두어 각 노드들이 언제 마지막으로 페이지를 갱신했는지의 인터벌 정보를 유지한다. 페이지 요청을 받았을 때 홈 노드는 요청된 버전 벡터와 diff 버전 벡터를 비교하여 페이지가 원하는 시점까지 갱신이 되었는지, 아니면 다른 노드로부터 diff 메시지가 도착하기를 기다려야 하는지를 판단한다.

페이지 홈이 아닌 노드에서의 페이지 무효화는 락(lock)을 획득할 때나 베리어(barrier) 연산 때에 수행된다. 각 노드는 인터벌 종료 시에 인터벌 내에서 변경된 페이지들의 번호를 기록하여 저장해 두는데 이를 write notice라고 한다. 락을 허가하는 노드는 요청한 노드가

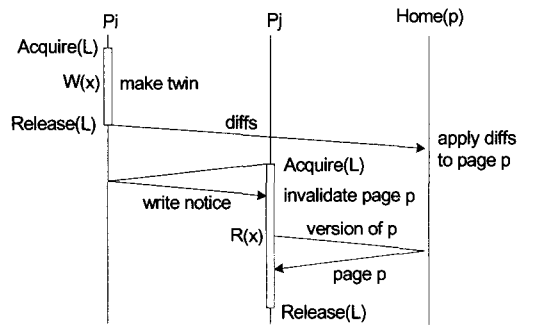


그림 1 HLRC의 동작

모르고 있는 인터벌들에 해당하는 write notice를 전달해 준다. 락을 획득한 노드는 이를 통해 페이지들을 무효화 시키고 다음에 페이지 폴트가 발생 했을 때 최신의 페이지를 홈으로부터 가지고 온다. 베리어의 경우에도 한 노드가 모든 노드의 write notice를 모은 후 노드들 간에 서로 모르고 있는 인터벌의 write notice를 보내줌으로써 페이지를 무효화 시킬 수 있게 한다.

2.2 고장 모델(failure model) 및 고장 허용

FT-KDSM은 다음과 같이 고장이 일어나는 상황을 제한한다. 이는 일반적인 고장 허용 시스템에서 가정하고 있는 고장 모델과 유사하다. 첫째, 고장-정지(fail-stop) 모델을 가정한다. 즉, 한 노드에서 고장이 발생하면 다른 노드의 상태에 영향을 미치지 않고 정지 한다. 둘째, 노드 사이의 네트웍은 안정적이라고 가정한다. 최근 개발되는 통신 계층들은 그 자체적으로 고장 허용성을 지원하기 때문에 본 논문에서는 네트웍의 고장은 고려하지 않는다. 마지막으로 디스크는 안정적이라고 가정한다.

FT-KDSM에서는 각 노드가 독립적으로 체크포인팅을 수행한다고 가정하며 메모리 상태 및 내용을 변화시키는 메시지를 다른 노드들로부터 비동기적으로 받았을 때 이를 수신 노드의 로그 홈 노드에 로깅한다. 각 노드는 응용 프로그램을 수행하는 동시에 또 다른 노드의 로그 홈 역할을 한다. 로깅은 메시지의 종류에 따라 메시지 송신 노드 또는 수신 노드가 수행하는데, VI-GM에서 제공하는 원격 메모리 쓰기 기능을 이용하여 적은 비용으로 이루어진다. 또한 VI-GM이 지원하는 안정적인 통신 레벨을 사용하므로 송신 노드는 로그 메시지에 대해 별도의 확인 메시지(ack)를 받지 않아도 좋다. 즉, 송신 노드는 로그 홈으로의 메시지 송신 완료를 로깅의 완료라고 간주 할 수 있다. 로그 홈에서 로그의 양이 특정 한계를 넘게 되면 상대 노드를 체크포인팅 하도록 하는데, 로그 홈에 저장된 로그는 오로지 상대 노드의 복구를 위해서 필요한 정보이므로 체크포인팅 시에 메모리에서 삭제되어도 좋다.

한 노드에서 고장이 발생했을 경우 그 나머지 노드들은 고장 난 노드로의 메시지 전송에 실패 했을 때, 또는 동기화 시점에 이르렀을 때 고장을 발견하게 된다. 고장 난 노드는 마지막 체크포인팅 지점으로 롤백하고 그 이후에 받았던 메시지들을 자신의 로그 홈으로부터 가져와 적용시킴으로써 노드의 상태를 고장 발생 이전과 동일하게 변화시켜 간다. 모든 로그를 적용한 후에는 전체 시스템의 메모리 일관성이 깨어지지 않는 지점까지 복구된다. 복구가 완료되면 나머지 노드들에게 완료 사실을 알려서 고장으로 인해 송신에 실패했던 메시지들을 다시 전송 받고, 이후의 프로그램을 계속 수행한다.

3. 원격 로깅(Remote Logging)

다른 노드에 의해 지역 메모리의 내용과 상태가 변화하는 경우는 홈 페이지를 갱신하는 메시지(diff)와 원격 페이지를 무효화 시키는 메시지(write notice)를 받았을 때이다. 고장으로부터 복구할 때에는 이러한 메시지들을 메모리에 다시 적용 시켜야 하므로, 정상 수행 동안 노드 간에 교환된 diff와 write notice 메시지들을 로깅해야 한다.

3.1 Diff 로깅

올바른 복구를 위해서는 고장 난 노드가 고장 발생 이전과 동일한 데이타를 읽고 동일한 연산 과정을 거쳐서 이전과 같은 메모리 상태를 만들어야 한다. 따라서 복구를 위해 다음 두 가지 측면을 모두 고려한다. 첫째, 프로그램을 재 수행하면서 그 노드에 할당되었던 홈 페이지들을 고장 전과 동일한 과정으로 갱신해야 한다. 고장 난 노드가 자신의 홈 페이지를 갱신하는 것은 응용 프로그램을 재 수행하는 과정에서 전과 동일하게 이루어 질 수 있으나, 다른 노드에 의한 갱신은 로그를 이용하여 다시 적용되어야 한다. 이를 위해서 정상 수행 시 페이지를 수정한 노드는 동기화 시점에 diff를 페이지의 홈 뿐 아니라 페이지 홈의 로그 홈에 보내어 로깅한다.

둘째, 고장 전에 읽었던 것과 동일한 원격 페이지들을 다른 페이지 홈 노드들로부터 제공받아야 한다. 이를 위해 정상 수행 시, 모든 페이지의 홈들은 diff를 적용 시킨 후 이를 삭제하지 않고 그대로 지역 메모리에 로깅한다. 또한 페이지 홈 자신이 쓰기를 수행한 페이지에 대해서도 diff를 생성하여 이를 로깅한다. 만일 다른 노드에서 고장 났을 경우, 각 페이지 홈은 체크포인팅 된 초기 페이지에 diff 로그를 순차적으로 적용하여 고장 난 노드가 원하는 시점의 페이지를 재생성하고 서비스 해준다. 페이지 홈에서의 diff 로그는 다른 노드의 고장에 대비하기 위한 것이므로 체크포인트 시에 삭제하지 않고 디스크에 저장한다.

그림 2는 diff 로깅의 예를 보인다. diff를 송신하는 노드 P1은 페이지 홈 P2의 고장에 대비하기 위해 P2의 로그 홈 P3에 diff를 성공적으로 전송한 후 락을 해제한다. 그리고 P2는 다른 노드의 고장 발생에 대비하여 diff 적용 후에도 메시지가 메모리에 계속 저장되도록 한다. 만일 P2가 diff를 받은 후 고장 났다면(i) 동일한 diff 메시지를 P3의 로그로부터 얻을 수 있다. 만일 P1이 P3에 로깅하기 전, P2와 동시에 고장 난 경우에는 (ii), P1이 해당 인터벌을 다시 수행하고 새로운 diff를 P2에 전송하여 P2의 복구를 가능하게 한다. 이 diff는 P2의 고장 발생 전에 받았던 diff와 다를 수 있지만 P1의 고장 시점이 락을 해제하기 이전이므로 다른 노드들

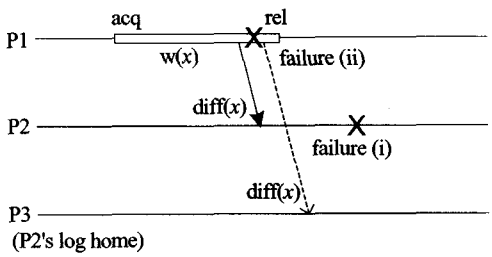


그림 2 Diff 로깅 방법

은 수정된 페이지 x에 접근하지 못하였다. 따라서 전체 메모리 일관성을 해치지 않고 새로운 diff를 이용하여 P2를 복구할 수 있다.

3.2 Write notice 로깅

HLRC에서는 락을 해제할 때 이전 인터벌에 대한 write notice를 만든다. 그리고 락을 요청한 노드가 모르고 있는 다른 노드의 인터벌들에 대해 해당 write notice를 전달해 준다. 이 메시지는 락을 요청한 노드에서 고장이 발생한 경우 동일한 write notice를 적용할 수 있도록 수신 노드의 로그 홈에 기록된다. 이 때 로그 홈은 자신이 마지막으로 받은 diff 로그가 무엇이었는지의 정보를 write notice 로그와 같이 로깅한다. 이는 락 연산을 재 수행 할 때, 락 연산 이전에 받았던 diff들을 페이지에 적용시키기 위해서 필요하다.

그림 3은 락에서의 write notice 로깅의 예를 보여준다. P2는 이전에 락을 소유하고 있었던 P1으로부터 write notice를 받고 이를 자신의 로그 홈 P3에 보내어 로깅한다. 만일 로깅하기 이전에 P2가 고장 난 경우(i)에는 락 연산을 복구할 수 없지만 고장 이전에 해당 인터벌이 완결되지 않은 상태이므로 메모리 일관성을 해치지 않는다. 이 경우 P2는 새롭게 락을 얻어 인터벌을 수행한다. P2가 로깅을 완료한 후 고장 난 경우(ii)(iii)에는 P3로부터 동일한 write notice를 가져와 락 연산을 복구할 수 있다. 락을 해제하기 이전에 고장 났다면 (ii) 그 인터벌은 복구되지 않아도 메모리 일관성을 해치지 않지만 본 논문에서 제안하는 로깅 방법에서는 락의 요청 시에 로깅이 이루어지므로 복구가 가능하다.

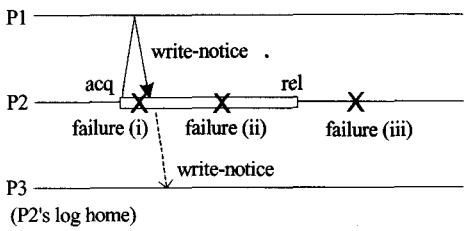


그림 3 락에서의 write notice 방법

베리어 연산에서의 write notice 로깅도 이와 유사하다. 베리어 연산은 두 단계로 나뉘는데, 우선 모든 노드는 베리어 홈에게 write notice를 보내고 더불어 베리어 홈의 로그 홈에도 동일한 메시지를 보내어 로깅한다. 베리어 홈은 로그 홈에 로깅이 모두 완료되었음을 확인한 후 write notice를 취합하여 각 노드가 인지하지 못하고 있는 인터벌의 write notice를 전송해 준다. 이 때 각 수신 노드의 로그 홈에도 동일한 메시지를 보내어 로깅한다. 즉, 모든 노드는 베리어 연산 동안 받은 것과 동일한 write notice를 고장 발생 시 자신의 로그 홈에서 얻을 수 있다.

4. 고장으로부터의 복구

한 노드에서 고장이 발생하면 다른 노드들은 그 노드와의 통신에서 실패하게 되므로 고장 발생을 인지 할 수 있다. 살아있는 노드들은 고장 난 노드와 새롭게 커넥션을 맺은 후 로그 및 페이지의 요청에 응답한다. 고장이 발생한 노드는 마지막 체크포인팅 지점으로 롤백하고 그 시점부터 응용 프로그램을 계속 수행하며 복구한다.

락 요청 및 베리어의 재 수행 :

새로운 인터벌이 시작될 때에는 그 인터벌 이전까지 받았던 diff 메시지들을 로그 홈으로부터 가져와 지역 홈 페이지를 갱신한다. 가령, 락의 경우에는 로그 홈이 해당 락 연산 이전에 받은 마지막 diff 메시지의 인덱스를 로그로부터 읽어서 해당 diff 로그를 보내준다. 마찬가지로 베리어에 도달하면 write notice 로그에 포함되어 있는 타임스탬프 값을 참조하여 베리어 이전까지 받은 diff 로그를 보내준다. 그리고 각 동기화 시점에 저장한 write notice 로그를 보내어 원격 페이지를 무효화 시킨다.

페이지 폴트 :

원격 페이지 폴트가 발생하면 해당 홈 페이지에 요청 메시지를 보낸다. 페이지 홈은 체크포인트로부터 페이지의 초기 값을 얻고 요구된 시점까지의 diff 로그를 적용시켜 페이지를 재생성 할 수 있다. 로그에는 그 diff가 생성된 때의 인터벌과 쓰기를 수행한 노드의 정보가 기록되어 있으므로, 이를 참조하여 요청된 diff 버전 벡터로 페이지를 갱신하기 위해 필요한 diff들을 로그로부터 찾을 수 있다. 본 구현에서는 요청한 페이지 전체를 페이지 홈이 재생성하여 보내주는 대신, 페이지 갱신을 위해 필요한 diff 로그만을 전송하도록 구현하였다. 이는 필요한 diff 메시지의 크기가 작은 경우 이득이 될 수 있다.

여러 노드가 동시에 고장 난 경우의 복구 :

원격 로깅 방법에서는 한 노드와 그의 로그 홈이 동

시에 고장 난 경우를 제외하면 여러 노드가 동시에 고장이 나는 경우에도 복구 가능하다. 그림 4는 P1과 P5 두 노드가 동시에 복구를 수행할 때의 예를 나타낸다. P1의 로그 홈은 P2이며 P5의 로그 홈은 P6이라고 가정한다. 동기화 연산의 경우, 두 노드는 자신이 홈인 페이지에 대한 diff 로그와 write notice 로그를 각각의 로그 홈 P2와 P5로부터 받아올 수 있으므로 다른 노드의 고장과 무관하게 재 수행 할 수 있다. 그러나 페이지 폴트 발생 시, 그 페이지의 홈이 복구를 수행 중인 노드이고 필요한 diff 로그가 아직 홈에 적용되지 않은 경우에는 바로 페이지 요청을 해결 할 수 없다. 가령, 그림 4에서 P5가 P1이 요청한 버전의 페이지를 바로 서비스할 수 없는 시점이라면 필요한 diff 들을 P6으로부터 가져오게 되는 동기화 시점까지 계속 복구를 수행한다. 그 후, 해당 diff들을 P1에 보내어 지연된 페이지 요청을 해결한다.

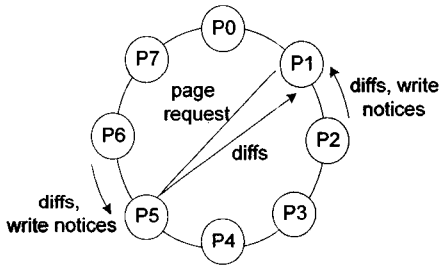


그림 4 여러 노드가 동시에 복구하는 경우

복구의 종결 :

로그를 다 적용 시키고 모든 고장 난 노드들의 복구가 끝나면 이를 다른 노드에게 알려서 정상 수행을 계속한다. 고장 발생으로 인해 통신에 실패했던 노드들은 같은 통신 메시지를 다시 보낸다.

5. 성능 평가

본 논문에서는 8대의 Pentium III 850MHz 컴퓨터로 구성된 Linux 클러스터 상에서 FT-KDSM 시스템의 성능을 측정하였다. 각 노드는 LANai9.1 프로세서를 장착한 Myrinet 통신망으로 연결된다. 벤치마크로는 SPLASH2[9]의 ocean, water, barnes, FFT와 Rice 대학에서 개발된 TSP를 사용하였다. 표 1은 각 응용프로그램에서의 입력 데이타 크기를 보인다. 본 논문에서는 제안하는 로깅 방법의 오버헤드와 고장으로부터의 복구 성능에 초점을 맞추기 때문에 실험에서 체크포인팅의 영향은 배제하였다.

5.1 원격 로깅의 오버헤드

고장 허용 분산공유메모리 시스템에서 응용 프로그램

표 1 벤치마크 프로그램과 입력 데이타 크기

ocean	258*258 ocean
water	1728 mols, 5steps
barnes	64k bodies
FFT	128 x 128 x 64 data points
TSP	20 cities

의 전체 수행 시간(E)은 다음과 같다.

$$E = Tbusy + Tmem + Tckp \quad (1)$$

(E : 전체 수행 시간, Tbusy : 연산 수행, Tmem : 메모리 일관성 유지, Tckp : 체크포인팅)

$$Tmem = Tbar + Tlock + Tpf \quad (2)$$

(Tbar : 배리어 수행, Tlock : 락 수행, Tpf : 페이지 폴트 처리)

(1)의 식에서 프로그램 내의 연산을 수행하는 시간 Tbusy와 프로세서 및 메모리 내용을 저장하는 체크포인팅 시간 Tckp은 로깅의 방법에 영향을 받지 않고 고정적으로 소요되는 시간이다. 반면, Tmem은 로깅 방법에 크게 좌우되며 고장 허용 시스템의 전체 성능에 중요한 영향을 미친다. 따라서 본 실험에서는 체크포인팅을 수행하지 않았으며, 정상 수행 시 프로그램의 전체 수행 시간을 측정하고 Tmem을 통하여 로깅 오버헤드를 분석하였다.

원격 로깅을 사용하는 FT-KDSM의 성능은 로깅을 하지 않는 KDSM-V의 성능 및 가장 직관적인 로깅 방법인 안정적 로깅을 사용했을 때의 성능과 비교된다. 안정적 로깅은 수신 노드가 동기화 시점 마다 이전 인터벌 동안 받은 메시지를 로컬 디스크에 저장하는 방식으로 구현되는데 로깅되는 데이타의 종류는 원격 로깅에서와 유사하다. 그림 5의 그래프는 왼쪽부터 로깅을 수행하지 않는 KDSM-V, 원격 로깅, 안정적 로깅의 전체 실행 시간을 나타낸다. 실험 결과, ocean을 제외한 네 개의 응용 프로그램에서 원격 로깅으로 인한 전체 성능 감소는 1~11%로서, 로깅으로 인한 실행 시간의 증가가 크

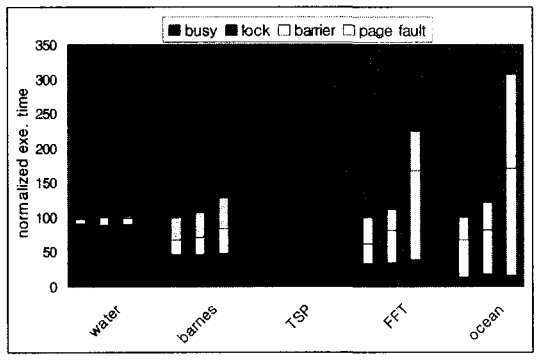


그림 5 원격 로깅과 안정적 로깅의 오버헤드

지 않음을 확인할 수 있다. 특히, FFT와 ocean의 경우에는 원격 로깅의 오버헤드가 각각 11%와 21%인 반면에 안정적 로깅의 오버헤드는 125% 와 206%로써, 안정적 로깅 보다 원격 로깅의 성능이 뛰어남을 볼 수 있다.

안정적 로깅에서는 모든 로그가 로컬 디스크에 저장되므로 원격 로깅과는 달리 동시에 발생하는 어떠한 종류의 고장으로부터라도 복구가 가능하다. 그러나 FFT와 ocean 처럼 로깅해야하는 데이타의 양이 많은 응용 프로그램의 경우, 동기화 시점 마다 디스크에 접근하는 오버헤드가 전체 성능에 크게 영향을 미침을 알 수 있다. 디스크에 로깅을 하는 동안 그 노드는 원격 노드의 페이지 요청을 처리하지 못하므로 페이지 폴트 시간이 증가한다. 또한 각 노드에서 로깅하는데 소요되는 시간이 서로 불균형하여 베리어 시간이 크게 증가함을 볼 수 있다. 반면, 제안하는 원격 로깅 방법은 수행 중 디스크로의 접근이 일어나지 않으며 빠른 통신을 통해 원격 메모리에 로깅을 수행하므로 전체 수행 시간의 감소가 크지 않다.

5.2 복구 시간

본 실험에서는 원격 로깅으로 기록된 데이타를 이용하여 시스템을 복구하는 시간을 측정하기 위해 특정 노드에 임의로 고장을 발생시켰다. 선택된 노드의 응용 프로그램은 exit() 함수를 통하여 정상 종료 바로 전에 강제 종료되며 그 외의 노드들은 고장이 어느 노드에서 발생했는지를 미리 알 수 있도록 설정하였다. 본 실험에서는 체크포인팅을 수행하지 않으므로, 고장 난 노드는 프로그램의 처음으로 롤백하고 로그를 이용하여 고장 지점까지 복구한다.

고장 허용 기능을 제공하지 않는 시스템에서는 고장 발생 후 전체 노드가 프로그램을 다시 시작해야 하는 반면, 고장 허용 시스템에서는 고장 난 노드만이 로그를 적용하여 복구한다. 그림 6에서는 고장 난 노드의 수가 각각 1,2,3개 일 때의 시스템 복구 시간을 전체 노드의 재 수행 시간에 정규화 하여 보인다. 실험 결과, 복구 중 동기화 연산에 소요되는 시간이 재 수행 때 보다 크게 감소함을 볼 수 있다. 복구 중에는 실제로 다른 노드들과의 동기화를 꾀할 필요 없이 write notice 로그를 이용하여 페이지를 무효화하고 diff 로그를 이용하여 지역 홈 페이지를 갱신하는 것으로 충분하다. 따라서 통신 시간이 줄고 동기화를 위해 기다리는 시간이 없어진다. 페이지 폴트 시간은 고장 난 노드의 개수가 증가함에 따라 커지는데, 이는 해당 페이지의 홈이 동시에 고장 난 경우 원하는 diff가 홈 노드에 적용될 때까지 페이지의 재생성이 불가능하기 때문이다.

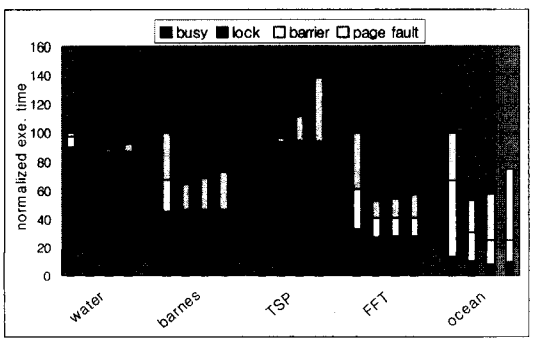


그림 6 원격 로깅과 안정적 로깅의 오버헤드

6. 결론 및 향후 연구 계획

기존의 고장 허용 시스템들은 여러 노드에서 동시에 발생하는 고장에 완벽히 대비하기 위해 높은 오버헤드를 감수하거나, 또는 적은 비용으로 단지 한 노드의 고장에만 대처한다. 그러나 시스템의 규모가 커짐에 따라 한 노드 이상에서 동시에 고장이 발생하는 경우가 많아진다. 따라서 비교적 적은 비용으로, 제한적으로나마 여러 노드의 고장을 허용할 수 있는 효율적인 방법이 필요하다.

본 논문에서는 고장 허용성을 제공하는 소프트웨어 분산공유메모리 시스템인 FT-KDSM을 구현하였다. 본 시스템은 원격 로깅 기법을 사용하여 빈번한 디스크 접근 없이 메시지 로그를 특정 노드의 메모리에 저장함으로써 정상 수행 시의 성능 감소를 최소화 한다. 실험 결과 빠른 네트워과 효율적인 통신 계층을 이용하는 원격 로깅이 디스크에 로깅하는 방법에 비하여 오버헤드가 적음을 확인하였다. 또한 원격 로깅은 로그 홈을 제외한 여러 노드에서 동시에 고장이 발생한 경우에도 복구 가능케 하여 시스템의 가용성을 높인다.

일반적으로 로그는 고장이 발생했을 때만 이용된다. 원격 로깅에서는 로그가 메모리에 저장되므로 정상 수행 동안의 성능 향상을 위해 로그를 이용하는 것이 용이하다. 이와 관련된 연구가 현재 진행 중이다. 그리고 복구 도중 새로운 페이지를 재생성하기 위해 diff를 순차적으로 적용하지 않고 임의의 노드가 가진 페이지의 최근 복사본으로부터 diff를 역순으로 적용하여 예전의 페이지를 재생성하는 방법에 대해 연구할 계획이다.

참 고 문 헌

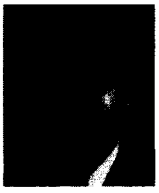
[1] G.Suri, B.Janssens, and W.K.Fuchs, "Reduced Overhead Logging for Rollback Recovery in Distributed Shared Memory", In Proceedings of the 25th Annual International Symposium on Fault-Tolerant Computing, June 1995.

[2] A.Kongmunvattana and M.F.Tzeng, "Coherence Centric Logging and Recovery for Home-based

Software Distributed Shared Memory," In Proceedings of the International Conference of Parallel Processing, September 1999.
[3] F.Sultan, T.D.Nguyen, and L.Iftode, "Scalable Fault-tolerant Distributed Shared Memory," In Proceeding of Supercomputing, 2000.
[4] S.Kanthadai and J.L.Welch. "Implementation of Recoverable Distributed Shared Memory by Logging Writes," In Proceedings of the 16th International Conference on Distributed Computing Systems, May 1996.
[5] M.Costa, P.Guedes and M.Sequeira, N.Neves, and M.Castro, "Lightweight Logging for Lazy Release Consistent Distributed Shared Memory," In Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation, October 1996.
[6] 박소연, 김영재, 이상권, 맹승렬, "VIA(Virtual Interface Architecture)를 기반으로 하는 소프트웨어 분산 공유메모리 시스템의 설계 및 구현", 제29회 한국정보과학회 춘계 학술발표논문집(A), 2002.4.
[7] http://www.myri.com/scs/index.html
[8] Y.Zhou, L.Iftode, and K.Li, "Performance Evaluation of Two Home-based Lazy Release Consistency Protocols for Shared Virtual Memory Systems," In Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation, October 1996.
[9] S.Woo, M.Ohara, E.Torrie, J.Singh, and A.Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," In Proceedings of the 22nd International Symposium on Computer Architecture, May 1995.

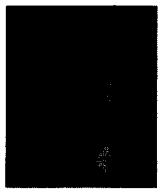
박 소 연

1998년 서강대학교 전자계산학과 학사 2000년 한국과학기술원 전자전산학과 전산학 전공 석사. 2000년~현재 한국과학기술원 전자전산학과 전산학전공 박사과정. 관심분야는 Software Distributed Shared Memory, Fault Tolerance, Parallel Processing, Computer Architecture



김 영 재

2001년 서강대학교 컴퓨터학과 학사. 2003년 한국과학기술원 전자전산학과 전산학 전공 석사. 2003년~현재 한국전자통신연구원 임베디드 S/W 센터 연구원. 관심분야는 Parallel Processing, Fault Tolerant S/W DSM, Embedded Operating System



맹 승 렬

1977년 서울대학교 공과대학 전자공학과 학사. 1979년 한국과학기술원 전산학과 석사. 1984년 한국과학기술원 전산학과 박사. 1984년~현재 한국과학기술원 전자전산학과 전산학전공 교수. 관심분야는 Computer Architecture, Parallel Processing, Cluster Computing