

소프트웨어 분산공유메모리 시스템을 위한 HLRC 프로토콜의 설계 및 구현

윤 희철^o, 이상권, 이준원

한국과학기술원 전자전산학과 전산학전공

Design and Implementation of HLRC Protocol for Software Distributed Shared Memory System

Hee-Chul Yun^o, Sang-Kwon Lee, Joonwon Lee

Division of Computer Science, Department of Electrical Engineering & Computer Science, KAIST

hcyun@camars.kaist.ac.kr

요 약

통신 오버헤드 및 거짓 공유(false sharing) 등의 문제를 해결하기 위하여 소프트웨어 분산공유메모리 시스템을 위한 다양한 메모리 모델들이 제안되었다. HLRC(Home based Lazy Release)[1]는 Keleher에 의해 제안된 LRC [2] 모델에 home 개념을 도입한 모델로서 최근의 소프트웨어 분산공유 메모리 시스템에서 널리 채용되고 있다. 본 논문에서는 HLRC 모델을 기반으로 한 메모리 일관성 프로토콜의 설계, 구현, 그리고 성능 측정 결과에 관하여 기술한다.

1. 서론

최근 고성능 마이크로 프로세서와 고속 네트워크의 등장으로 인해서 NOW(Networks Of Workstations)와 같은 클러스터 시스템을 병렬 처리에 사용하고자 하는 연구들이 활발히 진행중이다. 소프트웨어 분산공유메모리(Software Distributed Shared Memory)는 특별한 하드웨어를 필요치 않고 구현이 비교적 용이하기 때문에 NOW 상에서 공유메모리를 제공하는데 효과적이다.

일반적으로 소프트웨어 분산공유메모리는 주소공간을 페이지 단위로 분할하며, 각 페이지 단위로 복제(replication) 및 이동(migration)을 시킨다. 페이지 기반의 소프트웨어 분산공유메모리 시스템의 문제점은 (1) 높은 통신 오버헤드와 (2) 캐쉬 및 통신 단위(granularity)인 페이지 크기가 크다는 점이다. 특히 두 번째 문제는 거짓 공유(false sharing) 문제를 야기시킨다. 이와 같은 문제들을 해결하기 위해서 다양한 알고리즘 및 메모리 모델들이 제안되었다 [3, 4, 5, 1, 6].

IVY [3]는 최초의 소프트웨어 분산공유메모리 시스템으로 SC(Sequential Consistency) 모델을 사용하였다. SC는 직관적이라는 장점을 가지지만 프로토콜 오버헤드가 크다는 단점을 가진다. TreadMarks [4]는 LRC(Lazy Release Consistency) 모델을 사용하여 이러한 거짓 공유문제를 해결함과 동시에 multiple writer를

가능하게 하였다. 최근의 DSM 시스템들은 HLRC 혹은 보다 더 완화된 메모리 모델을 채용하거나 adaptive한 프로토콜을 통해 통신상의 오버헤드를 줄이는데 초점을 맞추어 연구되고 있다.

본 논문에서는 KDSM(KAIST Distributed Shared Memory) 시스템에 사용된 HLRC 프로토콜의 설계, 구현, 그리고 성능에 대해서 설명한다.

논문의 구성은 다음과 같다. 2절에서는 KDSM 시스템의 개요를 기술한다. 3절에서는 HLRC 프로토콜의 설계 및 구현 사항에 관해 기술하고, 4절에서는 그 성능 측정 결과를 설명한다. 그리고 5절에서 결론을 맺는다.

2. KDSM 시스템 개요

KDSM(KAIST Distributed Shared Memory) 시스템은 Linux 2.2.3 커널에서 구현되었으며 사용자 수준 프로세스(user-level process)로 동작하고, TCP/IP를 사용해서 프로세스 간 통신을 수행한다. 캐쉬 일관성 프로토콜은 페이지 기반 무효화 프로토콜(page-based invalidation protocol)과 홈 기반 프로토콜(home-base protocol)을 기반으로 해서 다중 읽기 다중 쓰기(multiple reader multiple writer) 프로토콜을 사용한다. 또한 KDSM 시스템은 HLRC [1]와 ScC(Scope Consistency) [6] 두 가지 메모리 모델을 제공한다.

* 본 연구는 국가지정연구실 사업의 지원을 받았다.

3. HLRC 프로토콜의 설계 및 구현

HLRC는 DSM 응용 프로그램의 수행을 interval들로 구분하는데 한 interval은 release와 다음 release 사이를 말한다. interval은 release 마다 증가하며 interval 내에서의 모든 변경된 페이지에 대한 정보는 특정한 테이블에 기록한다. HLRC에서 여러 프로세스간의 메모리 일관성은 interval을 기본 단위로 하여 이루어진다. 일관성의 유지를 위해 각 프로세스는 각각 독립적으로 vector timestamp를 유지한다. vector timestamp는 어떤 프로세스가 알고 있는 다른 프로세스의 최신의 interval 값의 리스트이며 이 값을 통해 다른 프로세스의 어느 시간까지의 변경사항을 알고 있는지를 알 수 있다.

3.1 Lock의 설계 및 구현

일반적으로 lock은 상호배제를 위한 도구이나 HLRC에서는 상호배제 뿐만 아니라 lock을 통해 메모리 일관성에 대한 정보(diff와 write-notice)도 교환한다. 이 절에서는 HLRC 프로토콜의 lock 구현에 대해 상호배제와 일관성 유지의 순서로 기술한다.

3.1.1 상호배제 (Mutual Exclusion)

상호배제는 토큰을 기본으로 하는 분산 lock 모델을 사용하였다. 이 방법에서 lock을 acquire하기 위해서는 lock 토큰을 받아야 한다. 어떤 프로세스가 n번째로 lock을 요청했을 때 만일 lock 토큰을 가지고 있지 않다면 n-1번째로 lock을 요청한 프로세스에게서 lock 토큰을 받아온다. 여기에서 n-1번째 프로세스가 어떠한 것인가는 지정된 manager를 두어 관리한다. 이와 같은 처리를 위해서 각 프로세스는 local, held, saved의 세 필드(field)를 유지한다. local은 lock 토큰을 가지고 있는지의 여부를 나타내며 held는 lock 토큰을 사용하고 있는지의 여부를 그리고 saved는 아직 처리하지 못한 포워딩된 lock 요청 메시지를 저장하는데 쓰인다.

그림 1은 P1이 lock을 사용하고 있는 상태에서 P1이 lock을 release하고 P2가 lock 요청을 하였을 때 전송되는 메시지와 각 프로세스의 lock과 관련된 필드들의 내용을 보여주고 있다.

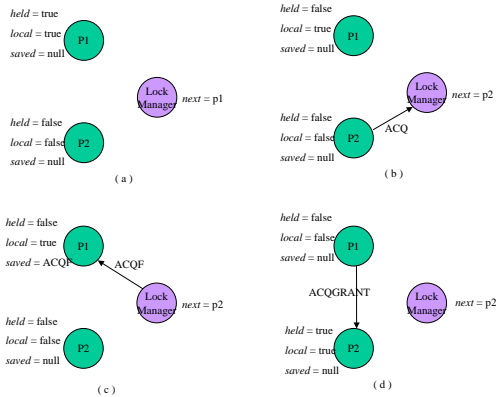


그림 1: 분산 Lock 관리

3.1.2 메모리 일관성(Consistency)

메모리 일관성 정보는 diff와 write-notice 정보이다. diff는 한 인터벌 동안 특정 페이지가 변경된 내용으로 lock acquire, release 시에 해당 페이지의 흐름으로 전송되어 흐름이 항상 페이지에 대한 최신의 정보를 유지하도록 한다. write-notice는 페이지가 변경되었는지의 유무를 알려주는 것으로 lock을 release하는 프로세스에서 lock을 acquire하는 프로세스에게로 lock grant 메시지를 통해 전달된다. write-notice는 interval 단위로 저장되므로 write-notice를 전송할 때 어느 interval까지를 전송하는가를 알아야 할 필요가 있다. 이것을 결정하는 것은 vector timestamp의 비교에 의해 이루어진다. write-notice를 받은 프로세스는 해당 페이지들을 무효화시키고 자신의 vector timestamp 갱신한다. 추후 무효화된 페이지에 대해 접근하려고 하면 페이지 폴트가 발생하여 페이지의 흐름으로부터 페이지 전체를 읽어온다.

그림 2은 HLRC 프로토콜에서의 write-notice 정보의 흐름의 예를 보여준다. 이 그림은 P0, P1, P3가 차례로 lock을 얻고 페이지 X, Y, Z를 수정하고 마지막으로 다시 P0가 lock을 얻어 X, Y, Z를 읽었을 때 write-notice 정보가 어떻게 흘러가지를 보여준다. 그림에서 앞절에서 설명한 lock manager를 통해 lock 요청이 포워딩되는 과정은 생략하였다. ACQ 메시지의 괄호안의 숫자는 자신의 vector timestamp를 나타낸다. ACQGRANT 메시지는 write-notice 정보를 포함하는데 write-notice는 (프로세스, interval, 수정된 페이지 리스트)와 같은 형태로 나타내었다. 표 1은 그림 2의 각 위치에서 해당 프로세스의 vector timestamp 값을 보여준다.

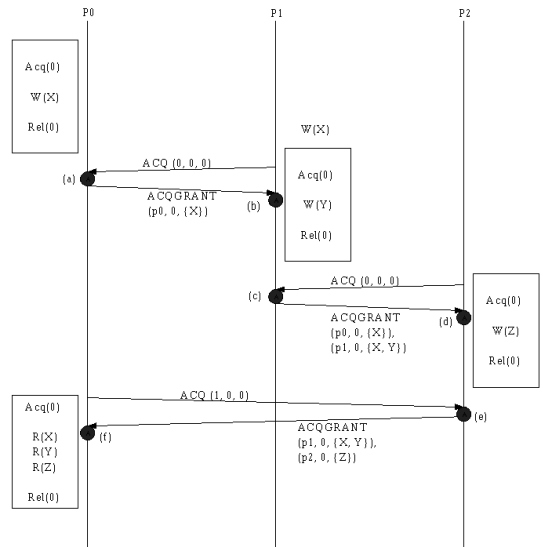


그림 2: write-notice 정보의 흐름

위치	프로세스	vector timestamp
(a)	P0	(1,0,0)
(b)	P1	(1,0,0)
(c)	P1	(1,1,0)
(d)	P2	(1,1,0)
(e)	P2	(1,1,1)
(f)	P0	(1,0,0)

표 1: 프로세스의 vector timestamp 정보

3.2 Barrier의 구현

barrier는 release와 acquire로 생각할 수 있다. 즉 barrier에 도착하는 것은 lock release, barrier를 떠나는 것은 lock acquire로 생각할 수 있다. barrier 도착 시 고정된 barrier 서버에게 자신의 vector timestamp와 write-notices를 보내면 barrier 서버는 모든 프로세스에게 이를 종합하여 각 requester의 vector timestamp와 비교하여 write-notices를 보내준다.

4. 성능 평가

이 장에서는 HLRC 프로토콜을 사용했을 때의 KDSM 시스템의 실행 성능에 관해서 설명한다. 성능 측정은 500 MHz Pentium III 프로세서를 탑재하고 128M의 메모리를 가진 8대의 PC를 100 Mbps Switched Fast Ethernet으로 묶은 클러스터 시스템에서 수행되었다. 성능 측정은 SPLASH2 [7]의 Water와 LU, Rice 대학의 SOR, TSP를 사용하였다. 또한 다른 시스템과의 성능 비교를 위해 Chinese Academy of Science에서 개발된 JIAJIA [5] 시스템을 비교 대상으로 삼았다. JIAJIA는 ScC [6] 모델을 사용하는 소프트웨어 DSM 시스템이다.

표 2은 각 응용에 대해서 프로세서의 수를 1개, 2개, 4개, 8개씩 사용했을 때 걸린 실행 시간을 보여준다. 그림 3은 각 응용 프로그램에 대해서 얻을 수 있는 속도 향상을 보여준다.

application	system	processor count			
		1	2	4	8
LU 1024×1024	KDSM	161.49	86.00	42.34	21.79
	JIAJIA	161.48	80.68	43.85	22.41
SOR 1024×1024	KDSM	28.95	31.95	19.79	17.14
	JIAJIA	30.17	29.10	20.46	16.22
TSP 20 cities	KDSM	93.84	48.82	26.01	16.25
	JIAJIA	93.18	50.58	27.43	17.51
WATER 1728 moles	KDSM	104.04	56.52	32.09	23.76
	JIAJIA	104.50	56.85	33.26	25.10

표 2: 실행 시간

모든 프로그램에 대해 KDSM은 JIAJIA와 거의 유사한 성능을 보였다. SOR과 WATER의 경우 그다지 좋은 성능향상을 얻지 못하였는데 이는 메시지의 처리로 인해 barrier와 lock에서의 대기 시간이 길어지기 때문이다. 이것은 소프트웨어 DSM에서 일반적으로 나타나는 문제로서 앞으로 해결해야 할 과제이다.

5. 결론

본 논문에서는 KDSM 시스템에 사용된 HLRC 프로토콜의 설계와 구현, 그리고 성능에 관해 설명하였다. 성능 측정 결과 널리 사용되는 벤치마크 병렬 프로그램 상에서 상당한 성능 향상을 보여 주었다.

현재 KDSM 시스템은 계속 개발중에 있으며, 메모리 일관성 프로토콜의 최적화, SMP 기계 특성을 활용할 수 있는 구조 설계, Myrinet 네트워크를 위한 사용자수준 통신계층을 연구 중에 있다.

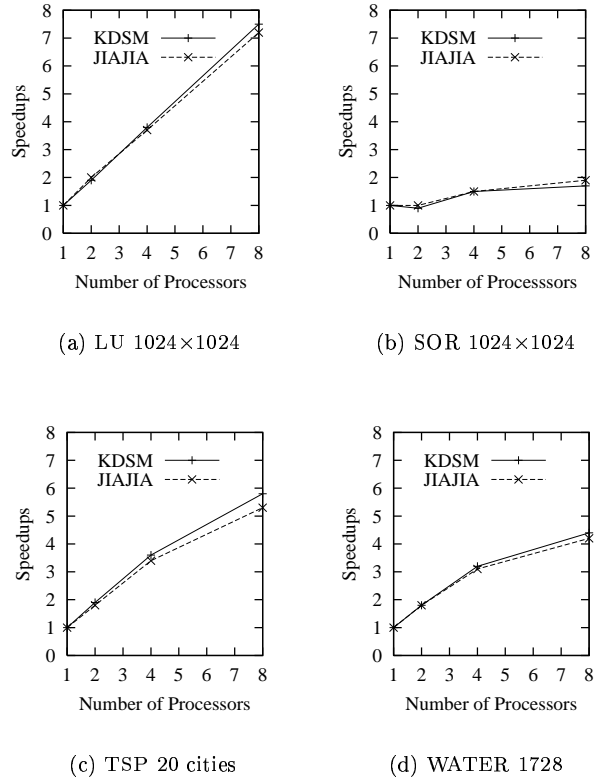


그림 3: 속도 향상 결과

참고 문헌

- [1] 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 USENIX OSDI*, October 1996.
- [2] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In *Proceedings of ISCA*, 1992.
- [3] K. Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors*. PhD thesis, Yale University, Department of Computer Science, September 1986.
- [4] C. Amza, S. Dwarkadas, P. Keleher, A. Cox, and Z. Zwaenepoel. Treadmarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, 29(2), February 1996.
- [5] W. Hu, W. Shi, and Z. Tang. The JIAJIA Software DSM System. Technical report, Institute of Computing Technology, Chinese Academy of Sciences, February 1998.
- [6] L. Iftode, J.P. Singh, and K. Li. Scope Consistency: A Bridge between Release Consistency and Entry Consistency. In *Proceedings of SPAA*, 1996.
- [7] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of ISCA*, 1995.