소프트웨어 분산공유메모리 시스템을 위한 Scope Consistency 프로토콜의 설계 및 구현

이 상권°, 윤희철, 이준원

한국과학기술원 전자전산학과 전산학전공

Design and Implementation of Scope Consistency Protocol for Software Distributed Shared Memory System

Sang-Kwon Lee°, Hee-Chul Yun, Joonwon Lee

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

sklee@camars.kaist.ac.kr

요 약

거짓 공유(false sharing) 및 추가적인 통신으로 인한 성능 저하를 방지하기 위해서 소프트웨어 분산공유메모리 시스템을 위한 다양한 메모리 모델들이 제안되었다. Scope Consistency 모델은 Brazos, JIAJIA와 같은 최근 시스템에 채택된 모델로서, consistency scope 개념을 사용해서 데이타와 동기화 변수 간의 관계를 자동으로 인식한다. 본 논문에서는 Scope Consistency 프로토콜의 설계, 구현, 그리고 성능 측정 결과에 관해서 설명한다.

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 구현하기 쉽기 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 [1, 2, 3, 4, 5].

IVY [1]는 최초의 소프트웨어 분산공유메모리 시스템으로 SC(Sequential Consistency) 모델을 사용하였다. SC는 직관적이라는 장점을 가지지만 프로토콜 오버헤드가 크다는 단점을 가진다. TreadMarks [2]는 LRC(Lazy Release Consistency) 모델을 사용하여 불필요한 통신양을 줄이려고 하였다. JIAJIA [3]는 ScC(Scope Consistency) [5] 모델을 사용해서 LRC와 동일한 인터페이스를 사용하면서도 거짓 공유로 인한 성능 저하를 줄이고자 하였다.

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

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

2. KDSM 시스템 개요

KDSM(KIAST Distributed Shared Memory) 시스템은 Linux 2.2.3 커널에서 구현되었으며 사용자수준 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(Home-base Lazy Release Consistency) [4]와 ScC(Scope Consistency) [5] 두 가지 메모리 모델을 제공한다.

*본 연구는 국가지정연구실 사업의 지원을 받았다.

3. Scope Consistency 프로토콜의 설계 및 구현

ScC [5]는 데이터와 동기화 사건들 간의 암식적인 관계를 사용해서 consistency scope를 유추한다. 같은 lock에 의한 임계영역은 하나의 consistency scope를 이루고, barrier는 전체 프로그램을 포함하는 전역적인 consistency scope를 이룬다. scope 내에서 데이타에 가해진 수정은 그 scope 내에서만 보여지고, scope 밖에서 가해진 수정은 보여지는 것이 보장되지 않는다.

3.1 Lock 프로토콜의 설계 및 구현

ScC 프로토콜은 페이지 기반의 무효화 프로토콜(page-based invalidation protocol)과 홈 기반의 프로토콜(home-based protocol)을 사용해서 구현되었다.

각 lock 홈의 lock manager는 해당 lock에 관해서 다음의 정보를 유지한다: (1) 현재 incarnation 번호 [5] (incarnation 번호는 초기에 모든 프로세스에서 0으로 설정되고, lock이 release 될 때마다 1씩 증가한다) (2) 각 incarnation에서 수정된 페이지 리스트, 즉 write notice (3) lock 요청을 순서적으로 서비스하기 위한 lock request queue.

프로세스들은 자신이 얻었던 lock의 마지막 incarnation 번호를 기억한다. 프로세스는 ACQ 메시지를 보낼 때, 해당 lock에 대한 자신의 incarnation 번호를 함께 보낸다. lock manager는 이 정보를 기반으로 lock을 요청한 프로세스에게 현재 incarnation 번호와 write notice를 ACQGRANT 메시지와 함께 보낸다. ACQGRANT를 받은 프로세스는 자신의 incarnation 번호를 원래값에 1 더한 값과 ACQGRANT 메시지와 함께 온 incarnation 번호에 1 더한 값 중 큰 값으로 설정한다. 그리고, write notice에 표시된 페이지들을 무효화시킨다.

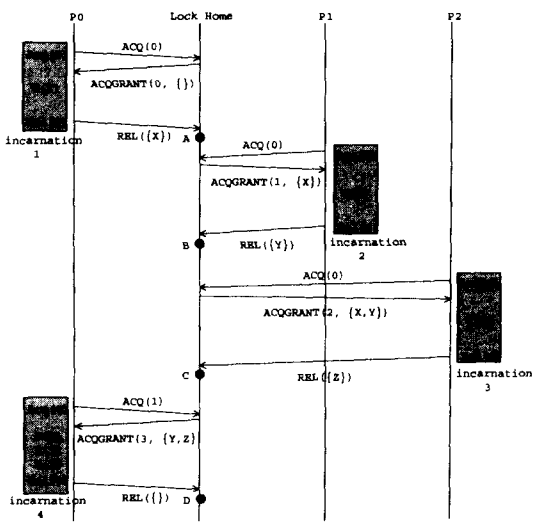


그림 1: ScC 프로토콜의 실행 예

각 프로세스들은 lock을 해제할 때, REL 메시지에 write notice 정보를 함께 lock 홈으로 보낸다. lock manager는 REL 메시지가 올 때마다 incarnation 번호를 1 증가시키고, 함께온 write notice 정보를 저장한다.

3.1.1 Lock 프로토콜의 실행 예

그림 1은 프로세스 P_0, P_1, P_2 순으로 lock을 얻어서 각각 X, Y, Z 페이지 값을 변경한 후, P_0이 다시 lock을 얻어서 X, Y, Z를 읽을 때의 실행 과정을 보여준다. ACQ 메시지 괄호 안의 수는 메시지를 보내는 프로세스가 기억하는 incarnation 번호를 표시한다. ACQGRANT 메시지 괄호 안의 수는 lock manager의 현재 incarnation 번호와 write notice를 표시한다. 그림 2는 lock 홈에서 관리되는 정보를 보여준다.

incarnation	write notice	그림 1에서의 지점
0	-	-
1	X	A
2	Y	B
3	Z	C
4	-	D

그림 2: lock 홈에서 유지하는 정보

3.1.2 Lock 프로토콜 알고리즘

lock이 중첩되어 사용될 때 각 lock의 scope 내에서 수정된 페이지들을 제대로 처리하기 위해서 스택 구조를 사용한다 [3].

`ProcessRWPages()` 함수는 임계영역에 들어 갈 때와 벗어날 때 RW 페이지들을 처리한다. RW 캐쉬 페이지들에 대해서, 해당 페이지 번호를 write notice에 추가하고, diff를 계산해서 저장하고, 페이지의 보호모드를 읽기전용으로 바꾸고, 캐쉬 상태를 RO로 바꾸고, twin 영역을 없앤다. DIFF 메시지를 사용해서 저장된 diff를 각 페이지의 홈에 전달한 후 DIFFGRANT 메시지를 기다린다. RW 지역 페이지들에 대해서, 해당 페이지 번호를 write notice에 추가한 후 페이지의 보호 모드를 읽기전용으로 바꾼다.

`DsmLock()` 함수는 먼저 `ProcessRWPages()` 함수를 호출하여 현재 RW 상태인 페이지들을 처리하고 write notice를 스택에 저장한다. lock 홈에게 ACQ 메시지를 보내고 ACQGRANT 메시지가 도착할 때까지 대기한다. 메시지가 도착하면 함께 온 write notice 정보에 따라서 해당 캐쉬 페이지들을 무효화시킨다.

`DsmUnlock()` 함수는 `ProcessRWPages()` 함수를 호출해서 현재 임계영역에서 수정된 페이지들을 처리하고, 그 결과로써 만들어진 write notice 정보를 REL 메시지와 함께 lock의 홈으로 보낸다. 스택에 저장된 이전 write notice 정보를 복구한다. 내부 임계영역에서의 수정은 외부 임계영역에서의 수정이라고 볼 수 있기 때문에 내부 임계영역의 write notice를 외부 임계영역의 write notice에 추가시킨다.

3.2 Barrier 프로토콜의 설계 및 구현

각 프로세스는 `DsmBarrier()` 함수를 호출해서 barrier를 구현한다. `DsmBarrier()` 함수는 `ProcessRWPages()`

함수를 호출해서 전역 scope에서 수정된 페이지들을 처리한다. 그 결과로 얻어진 write notice들을 BARR 메시지와 함께 미리 정해진 barrier 서버로 전송한 후, BARRGRANT 메시지가 오기를 기다린다. BARRGRANT 메시지가 도착하면 함께 온 write notice 정보에 따라서 해당 캐쉬 페이지들을 무효화시킨다.

barrier 서버는 BARR 메시지가 도착할 때마다 카운터를 1 증가시키고, BARR 메시지에 함께 온 write notice 정보를 저장한다. 카운터 값이 병렬 프로그램에 참가하고 있는 프로세스의 수와 일치하면, 모든 프로세스가 barrier에 도착한 것이므로 누적된 write notice 정보와 함께 BARRGRANT 메시지를 모든 프로세스에게 전송한다.

4. 성능 평가

이 장에서는 ScC 프로토콜을 사용했을 때의 KDSM 시스템의 실행 성능에 관해서 설명한다. 성능 측정은 500 MHz Pentium III 프로세서를 탑재하고 각각 128 MB의 메모리를 가진 8대의 PC를 100 Mbps Switched Fast Ethernet으로 묶은 클러스터 시스템에서 수행되었다. 성능 측정에 사용된 응용은 SPLASH2 [6] 응용 중 Water와 LU, Rice 대학에서 개발된 SOR, TSP를 선택하였다. 또한 KDSM의 성능을 다른 시스템과 비교하기 위해서, JIAJIA [3]를 비교 대상으로 삼았다.

표 1은 각 응용에 대해서 프로세서의 수를 1 개, 2 개, 4 개, 8 개씩 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	KDSM	161.68	86.55	42.72	24.05
1024×1024	JIAJIA	173.00	85.05	44.92	21.18
SOR	KDSM	29.47	25.81	17.27	15.14
1024×1024	JIAJIA	29.22	23.08	12.10	6.82
TSP	KDSM	93.22	48.86	27.12	15.84
20 cities	JIAJIA	93.18	49.54	26.37	16.30
WATER	KDSM	114.75	61.49	34.62	24.15
1728 moles	JIAJIA	102.74	53.52	28.16	15.86

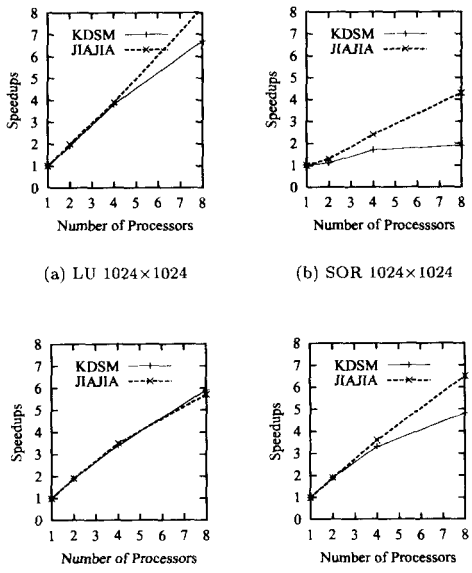
표 1: 실행 시간 (단위: 초)

TSP의 경우 KDSM은 JIAJIA와 거의 같은 성능을 보이고, LU와 WATER의 경우는 프로세서가 8개일 때만 성능이 조금 떨어진다. SOR의 경우 KDSM은 JIAJIA에 비해서 모든 경우에 성능이 떨어짐을 볼 수 있다. 이것은 JIAJIA의 ScC 프로토콜에 비해 KDSM의 ScC 프로토콜이 덜 최적화되었기 때문인데, 앞으로 해결해야 할 과제이다.

5. 결론

본 논문에서는 프로토타입 분산공유메모리 시스템인 KDSM 시스템에 사용된 Scope Consistency 프로토콜의 설계와 구현, 그리고 성능에 관해 설명하였다. 동일한 환경에서 성능 측정을 한 결과 KDSM 시스템이 JIAJIA 시스템에 비해서 비슷하거나 조금 떨어졌다.

현재 KDSM 시스템은 계속 개발중에 있으며, 메모리



(a) LU 1024×1024 (b) SOR 1024×1024

(c) TSP 20 cities (d) WATER 1728 moles

그림 3: 속도 향상

일관성 프로토콜의 최적화, SMP 기계 특성을 활용할 수 있는 구조 설계, Myrinet 네트워크을 위한 사용자수준 통신계층을 연구 중에 있다.

참고 문헌

[1] K. Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors.* PhD thesis, Yale University, Department of Computer Science, September 1986.

[2] 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.

[3] W. Hu, W. Shi, and Z. Tang. The JIAJIA Software DSM System. Technical report, Inititute of Computing Technology, Chinese Academy of Sciences, February 1998.

[4] 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.

[5] L. Iftode, J.P. Singh, and K. Li. Scope Consistency: A Bridge between Release Consistency and Entry Consistency. In *Proceedings of SPAA*, 1996.

[6] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of ISCA*, 1995.