분산공유 메모리를 위한 적응적 프로토콜

이성우[0] 류시룡 김용국 김현철 유기영
경 북 대 학 교 컴 퓨 터 공 학 과
{swlee, dryice, lrunner, hskim}@purple.knu.ac.kr, yook@knu.ac.kr

Adaptive Protocol for Distributed Shared Memory

Sung-Woo Lee[0] Shi-Ryong Ryu Yong-Kuk Kim Hyun-Chul Kim Kee-Young Yoo
Dept. of Computer Engineering, Kyungpook National University

요 약

본 연구에서는 분산 공유 메모리 시스템을 위해 기존의 두 프로토콜, Lazy Multiple Writer 프로토콜[3]과 Home-based 프로토콜[6]을 혼합한 적응적 프로토콜, Adaptive Lazy Multiple Writer(ALMW)을 제안한다. ALMW는 두 프로토콜 중 실시간에 접근패턴을 보고 각 페이지 별로 적절한 프로토콜을 선택하여 성능향상을 꾀한다. 본 프로토콜을 CVM[10] 패키지에 구현하고 기존의 두 프로토콜과 함께 IBM-SP2머신의 8개 프로세서상에서 4개의 응용 프로그램에 대해 성능 평가를 수행하였다.

1. 서론

최근에 높은 속도의 네트워크의 출현으로 리눅스를 기반으로 한 클러스터 시스템은 병렬 컴퓨팅을 위한 효율적인 환경으로 인식되고 있다. 이들 클러스터 시스템상에서 좀 더 편리한 프로그래밍 환경을 위해 공유 메모리 모델을 제공하는 소프트웨어 분산 공유 메모리 시스템(Software Distributed Shared Memory System, DSM)에 대해 많은 연구가 이루어 지고 있다[1,4,6].

DSM 일관성 모델 중 가장 우수하다고 알려진 Lazy Release Consistency (LRC)모델[3]을 따르는 두 프로토콜, Rice 대학의 Lazy Multiple Writer(LMW) 프로토콜[3]과 Princeton 대학의 Home-based (HOME) 프로토콜[6]은 최근 DSM 연구의 두 주류를 이루고 있다. 이 두 프로토콜을 비교하는 연구[8,9]의 결과를 보면 이들 프로토콜들은 특정 응용프로그램의 메모리 접근 패턴에 따라서 그것의 성능이 좌우됨을 알 수 있다. 결국, 최적의 성능을 얻기 위해서는 한 시스템 내에 이 두 프로토콜들을 동시에 구현하고, 시스템이 응용 프로그램의 메모리 접근 패턴을 관찰하여 특정 데이터 영역별로 독립적으로 하나의 프로토콜을 선택, 적용시키는 연구가 필요하다. 따라서 본 연구에서는 이들 두 프로토콜을 혼합한 Adaptive Lazy Multiple Write(ALMW) 프로토콜을 제안하고 기존의 프로토콜과 성능을 분석한다.

2. 관련연구

LRC는 Release Consistency(RC) 메모리 모델[2]의 한 종류이다. LRC 알고리즘은 한 프로세스가 임계영역의 시작부분을 표시하는 **요구**(acquire)작업을 수행할 때까지 그 프로세서에게 수정 사실을 전파하는 것을 지연시켜 프로세스 p가 프로세스 q로부터 **요구**를 끝내기 전에, p가 happen-before-1 partial order[2]에 의해 앞선 모든 프로세스들의 구간들 내에 발생한 수정사실을 알도록 요구한다. 그러므로 p가 **요구** 메시지를 q에게 보내면, q는 필요한 모든 구간들의 쓰기통보(write-notice)들을 수집하여 **해제-요구**(release-acquire) 메시지에 실어 p에게 보낸다. 쓰기통보는 생성한 프로세서 번호, 페이지 번호, 구간 번호로 구성되어 한 페이지가 특정 구간 내에서 수정되어졌음을 알려주지만 실제 수정을 포함하지 않는다. 쓰기통보를 받은 p는 해당 페이지를 무효화시킨다.

2.1 Lazy Multiple Writer 프로토콜

무효화된 페이지에 대해 접근하게 되면 페이지 실패(page fault)를 발생한다. 이때 페이지 실패를 발생한 프로세스는 현재 구간보다 부분순서(partial order)에 의해 앞서는 구간들 동안 수행된 수정내역들을 압축한 차이본(diff)들을 모아 그 페이지에 적용(apply)시켜야 한다. 이를 위해 프로세스 p는 차이본를 가지고 있지 않은 쓰기통보들을 모아 가장 최근에 도착한 쓰기통보를 생성한 프로세스 q에게 이들에 대한 차이본들을 요청하는 메시지를 보낸다. 이를 받은 q는 요청 받은 쓰기통보들 중 자신이 가지고 있는 차이본들을 모아 p에게 보낸다. 만약 p가 q로부터 요청한 차이본들을 모두 받았으면 그들을 순서대로 페이지에 적용시켜 유효한 페이지로 만든다. 그러나 다중 쓰기(multiple write)로 인해 p가 q로부터 요청한 차이본을 모두 받지 못했다면, 차이본을 획득하지 못한 쓰기통보들 중에서 가장 최근에 생성된 것을 발생시킨 프로세스에게 차이본들을 요청하는 메시지를 보낸다. 필요한 차이본을 모두 획득할 때까지 앞의 과정을 반복한다.

2.2 Home-based 프로토콜

Home-based 프로토콜에서는 각각의 공유 페이지는 프로그램에 의해 정적으로 홈 프로세스가 할당되어진다. 임계영역의 끝을 표시하는 해제(release)작업을 할 때, 한 프로세서는 자신의 직전 해제이후에 수정된 모든 페이지들에 대해 차이본들을 생성한다. 그리고 생성된 차이본들을 해당페이지의 홈 프로세서들에게 전송한다. 여기서, 홈 프로세서의 해당 페이지들은 쓰기 보호 상태(읽기만 가능한 상태)는 될 수 있지만 무효화 상태는 될 수 없다.

본 연구에서 구현한 HOME 프로토콜은 LRC의 순서를 보장하기 위한 방법으로 페이지 버전을 표시하기 위해 벡터스템프 기법을 사용하였다. 이 기법은 각 페이지 마다 flush_stamp와 page_stamp라는 두개의 벡터 스템프를 유지한다. 프로세서 p가 차이본을 홈 프로세서의 페이지에 복원시킬 때마다 p의 현재 구간 값을 p의 flush_stamp와 홈 프로세서의 flush_stamp에 기록한다. 그리고 모든 프로세서가 동기화 시점에서 쓰기통보를 통합할 때 그것의 구간을 해당프로세서의 page_stamp에 기록한다.

한 프로세서가 무효화된 페이지를 접근해 페이지 실패를 발생시키면 유효한 페이지를 얻기 위해 그 페이지의 홈 프로세서에게 page_stamp를 실어보내 페이지 요구 메시지를 보낸다. 이 메시지를 받은 홈 프로세서는 전송된 page_stamp와 flush_stamp와 비교 후 유효한 페이지임을 검사한 후 그 페이지 전체를 해당 프로세서에게 보내준다.

3. 새로운 프로토콜의 제안

이 두 프로토콜의 성능을 비교하는 연구들[6,8,9]중에서 본 연구와 관련있는 접근 패턴에 따른 장단점을 보면 다음과 같다.

(1) 이주적 접근 패턴을 가지는 데이터들에 대해서 LMW는 마지막 쓰기 프로세서에서 다음 쓰기 프로세서에게 바로 차이본을 전송하면 된다. 그러나 HOME은 마지막 쓰기 프로세서에서 차이본을 홈 프로세서로 전달된 후 홈 프로세서로부터 페이지 전체가 다음 쓰기 프로세서에게 전달되어야 하므로 LMW에 비해 메시지 수와 전송량이 늘어난다.

(2) 생성자/소비자 접근 패턴에서는 HOME이 집합적 데이터 전송을 할 수 있어 LMW에 비해 유리하다. 그러나 생성자나 소비자의 프로세서가 변화하는 경우에도 LMW는 이주적 접근 패턴에서처럼 생성자에서 소비자로 바로 전달되지만 HOME은 부가적인 메시지가 발생한다.

(3) w개의 프로세서에 의해 다중 쓰기가 수행된 후 바로 r개의 프로세서에 의해 읽기가 발생하는 경우, 모든 읽기 프로세서가 유효한 페이지를 얻기 위해 HOME은 2w+2r 개의 메시지가 필요하지만 LMW는 2wr 개의 메시지가 필요하다.

따라서 제안하는 프로토콜 Adaptive Lazy Multiple Writer(ALMW) 프로토콜은 앞서 언급한 두 가지 프로토콜을 내재하고 개별적인 페이지에 대해 특정 프로토콜이 유리한 공유 패턴이 발생하면 그 프로토콜로 전환한다. 여기서 전환기준은 자신을 제외한 다른 프로세서간에 다중쓰기가 발생하는지 여부이다.

3.2.1 Lmw 모드에서 Home 모드로 전환

Lmw모드로 프로세서 p가 동작중인 페이지 pg_x에 대해 접근실패가 발생하면 필요한 쓰기통보들을 모아 이들 중 가장 최근 쓰기통보의 생성자인 프로세서 q에게 차이본들을 요구한다. 이 때 q가 요청받은 쓰기통보들에 대한 차이본들을 모두 갖고 있지 않다면 pg_x를 Home모드로 전환하고 q로부터 응답받은 p 또한 pg_x를 전환한다. 이후 p는 차이본을 받지 못한 쓰기통보에 대한 다른 프로세서들에게 차이본 요구 메시지를 보낼 때 메시지에 태그를 붙여 상대에게 pg_x가 Home 모드로 전환되었음을 알린다. 이때 페이지의 홈은 p가 된다. 그리고 p로부터 차이본요구 메시지를 받은 프로세서들은 Home 모드로 전환시키기 전에 요구된 차이본 뿐 만 아니라 그 이후에 자신이 생성하거나 다른 프로세서로부터 받은 모든 차이본을 홈 프로세서 p에게 전송해야 한다. 물론 p와 메시지를 주고 받지 않은 프로세서들은 Lmw모드로 인식한다. 그림 1은 3개의 프로세서에서 페이지 pg_x가 Home모드로 변환되었고 P_2이 홈이 되었다. P_1이 차이본을 요구받았을 때 요구받은 $Write_1$에 대한 차이본 $diff_1$ 뿐만 아니라 이후에 발생된 $Write_2$에 대한 차이본 $diff_2$도 응답메시지를 싣는다. 그러나 P_0는 아직 Lmw 모드로 인식한다.

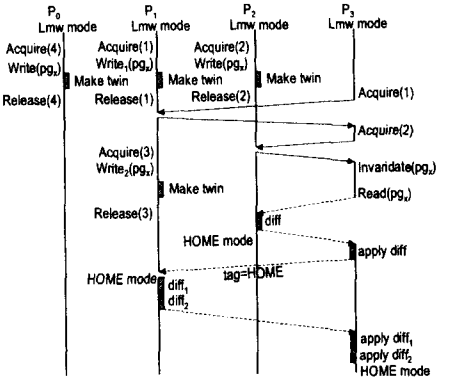

그림 1. Lmw 모드에서 Home 모드로 전환의 예

Home 모드로 전환된 페이지들에 대한 정보를 전파시키기 위해 쓰기통보 자료구조에 페이지 모드와 홈프로세서 번호 필드를 추가한다. 동기화 시점에서 Lmw모드로 인식하고 있는 프로세서 p가 Home모드인 프로세서 q로부터 Home모드 쓰기통보에 의해 Home으로 전환되어 진다. p는 쓰기통보를 받은 시점에서 즉시 Home모드로 전환되지 못하고 현재 구간이 끝날 때 자신의 차이본들을 홈 프로세서에게 전송한 후 모드를 전환한다.

기존의 HOME프로토콜에서는 페이지의 홈에서는 접근실패가 발생하지 않지만, 본 프로토콜에서는 Lmw모드로 인식하고 있는 프로세서가 존재한다면 그 프로세서의 차이본이 홈에 복원되지 않았기 때문에 홈에서도 접근실패가 발생할 수 있다. 그림 2에서 처럼 Lmw모드로 인식하는 P_0의 쓰기통보가 홈인 P_2에게 전달된다면 P_2는 그 페이지를 무효화시킨다. 이후에 홈에서 접근실패가 발생하면 flush_stamp가 page_stamp보다 작은 프로세서들에게 홈에 차이본들을 적용시킬 것을 요구하는 force_home_flush 메시지를 보내 최신 페이지로 복원한다.

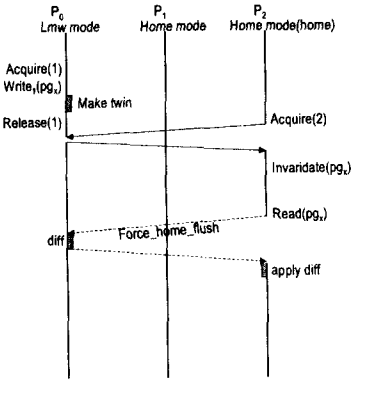

그림 2. 홈 프로세서에서 접근실패 발생에 대한 처리방법

또한 Home 모드이지만 홈이 아닌 프로세서 q에 의해 요구된 페이지의 버전이 홈 프로세서의 페이지 버전보다 더 최신것일 수도 있다. 이런 경우는 페이지와 홈 권한을 q에게 넘겨주고 q가 필요한 차이본을 다른 프로세서로부터 모아 최신 페이지로 복원시킨다.

3.2.1 Home 모드에서 Lmw 모드로 전환

Home 모드에서 Lmw 모드로 전환되는 것은 반대의 경우보다는 훨씬 더 간단하다. 모든 프로세서가 Home모드로 전환된 후, 다중 쓰기가 발생하지 않는다면 Home 모드로 전환된다. 즉, 홈 프로세서가 홈이 아닌 프로세서 q로부터 쓰기 접근실패에 의한 페이지 요구 메시지가 들어와 페이지를 전송한 후, q로부터 차이본이 복원되기 전에 그외의 프로세서로부터 쓰기접근 실패에 의한 페이지 요구 메시지를 받는다면 Lmw 모드로 전환한다. 이 사건 또한 쓰기 통보에 의해 다른 프로세서들에게 전파된다.

4. 성능평가

본 프로토콜의 성능평가를 위해 HOME 프로토콜과 ALMW 프로토콜을 CVM[10] DSM시스템상에서 구현하였고 LMW 프로토콜은 이미 CVM에 구현되어 진 것을 사용하였다. 실험은 IBM-SP2 슈퍼컴퓨터에서 8노드상에서 4개의 응용프로그램을 대상으로 수행되었다

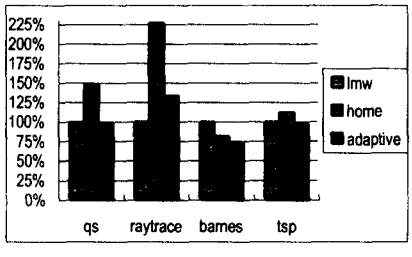

그림 3. 8개 프로세서에서의 수행 시간의 비교

테이블 1. 중요 요소들에 대한 실험 결과

	protocol	msgs	msg_diff	fault (msec)	flush (msec)
qs	lmw	25827	13836	26490	
243/1027	home	27409		11886	6407
	adaptive	26009	13389	26174	314
raytrace	lmw	160786	73893	24326	
(52/7690)	home	198128		48221	15065
	adaptive	168352	51651	30466	5099
barnes	lmw	37652	37484	40721	
(245/250)	home	11385		35101	2180
	adaptive	13027	4233	36079	2021
tsp	lmw	9228	7364	2815	
(1/197)	home	8366		4520	171
	adaptive	9721	6763	2584	108

그림 3은 전체 수행시간에 대해 3가지 프로토콜을 비교하고 있고 테이블 1은 메시지수(msgs), 차이본 요구메시지(msg_diff), 접근실패의 지연시간(fault), 홈 복원 지연시간(flush)들을 설명하고 있다. 차이본 요구 메시지는 LMW 프로토콜이나 ALMW 프로토콜의 Lmw모드에서 발생하고, 홈 복원은 ALMW의 Home모드나 HOME 프로토콜에서 발생한다. 그리고 첫번째 열에 괄호 안은 전체 사용 페이지에 대한 ALMW 프로토콜에서 Home 모드로 인식된 페이지 수를 나타낸다.

Qs에서 ALMW는 25%정도의 페이지가 Home모드로 전환했으나 차이본 요구메시지는 Lmw모드와 비슷한 것을 볼 때 이들 페이지의 접근이 그리 빈번하지 않아 수행속도는 LMW와 비슷하다. Raytrace는 HOME이 LMW 보다 상당히 많은 수의 메시지를 발생시키면서도 접근 실패의 지연시간 또한 2배정도나 길어 성능이 매우 나빠짐을 볼 수 있다. ALMW에서도 약간의 Home 모드 페이지로 인해 LMW 보다 수행속도가 35% 정도 나빠졌다. Barnes에서는 HOME에 비해 메시지 수는 증가하나 홈 복원 지연시간이 감소해 수행속도가 10%정도 향상되었다. Tsp에서는 ALMW가 단 1개의 페이지가 Home모드로 전환되었지만 이 페이지에 대한 접근이 매우 많아, LMW에 비해 접근실패 지연시간이 감소되어 그 결과 3%정도의 성능 향상을 얻었다.

5. 결론

본 논문에서는 Lazy Multiple Writer(LMW) 프로토콜과 Home -based (HOME) 프로토콜을 혼합한 Adaptive Lazy Multiple Writer(ALMW) 프로토콜을 제안하였다. 성능 평가 결과를 보면 ALMW가 4개중 2개의 응용 프로그램에서 LMW와 HOME에 비해 더 나은 성능을 보였다. 그러나 HOME 프로토콜이 LMW에 비해 현저히 수행속도가 떨어지는 응용 프로그램에 대해서는 ALMW 역시도 HOME에 비해서는 성능향상을 가져오지만 LMW보다는 나쁜 성능을 보였다. 이것은 ALMW 역시 구간이 끝날 때 마다 차이본을 생성시켜 홈 프로세서에게 전송해야 하는 HOME의 약점을 그대로 안고 있기 때문이다. 향후 이러한 패턴에 대한 적응 기법을 연구해야 한다.

6. 참고 문헌

[1] J. B. Carter, J. K. Bennett, and W. Zwaenepoel, "Implementation and Performance of Munin," Proc. the 13th ACM Symposium on Operating Systems Principles, Oct. 1991, pp. 152-164.

[2] K. Gharachooloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta and J. Hennessy, "Memory consistency and event ordering in scalable shared-memory multiprocessors," Proceedings of the 17th Annual Interbational Symposium on Computer Architecture, pp. 15-26, May 1990.

[3] P. Keleher, "Distributed Shared Memory Using Lazy Release Consistency," PhD dissertation, Rice University, Tech. Re port Rice Comp -TR-240, ftp cs.rice.edu under public/TreadMarks/papers, 1994.

[4] P. Keleher et al., "TreadMarks: Shared Memory Computing on Networks of Workstations," IEEE Computer, Feb. 1996, pp. 18-28.

[5] K. Li and P. Hudak, "Memory coherence in shared virtual memory systems." ACM Transaction of Computer Systems 7(4), 321-359, Nov 1989.

[6] Y. Zhou, L. Iftode, and K. Li, "Performance Evaluation of Two Home-based Lazy Release consistency Protocols for Shared Virtual Memory Systems," Proceedings of the Sec ond USENIX Symposium on Operating System Design and Implementation, Mar. 1996, pp. 75 - 88,

[7] L. Iftode, "Home -based Shared Virtual Memory," Ph.D. thesis, Rice Univ, Jun. 1998

[8] E. deLara, Y. Hu, A.L. Cox, and W. Zwaenepoel, "Evaluating the Effect of Con tention of Page -Based Software Shared Memory Systems,", Technical Report, Rice University, http://www.cs.rice.edu/~willy/TreadMarks/papers.html

[9] A.L. Cox, E. deLara, Y. Hu, and W. Zwaenepoel, "A Performance Comparison of Homeless and Home -based Lazy Re lease Consistency Protocols in Software Shared Memory", Proceedings of the Fifth High Performance Computer Architecture Conference, Jan. 1999.

[10] P. Keleher, "CVM:The Coherent Virtual Machine", http://www.cs.umd.edu/projects/cvm, November 1996.