Home 기반 소프트웨어 DSM 에서 프로토콜 오버헤드를 줄이는 방법

이상권
삼성전자 디지털미디어 연구소
e-mail : sngkwn@samsung.com

Reducing Protocol Overheads in Home-based Software DSM

Sang-Kwon Lee
Digital Media R&D Center, Samsung Electronics Co., Ltd.

요 약

소프트웨어 분산공유메모리 시스템은 특별한 하드웨어의 도움 없이 구현이 용이하다는 장점이 있지만, 큰 페이지 크기로 인한 메모리 접근 지연 문제와 diff 처리로 인한 프로토콜 오버헤드가 크다는 단점이 있다. 본 논문에서는 diff 처리 오버헤드를 효과적으로 줄일 수 있는 Proportional Home Migration 및 Adaptive Direct Diff 기법에 대해 제안한다.

1. 서론

최근 고성능 마이크로 프로세서와 고속 네트워크의 등장으로 인해 NOW(Network Of Workstations)와 같은 클러스터 시스템에 관한 연구가 활발히 진행되었다. 소프트웨어 분산공유메모리(DSM: Distributed Shared Memory)는 가상 메모리 하드웨어만으로 공유메모리를 제공할 수 있다. 하지만, 원거리 메모리 접근 지연이 길다는 문제점과 diff 처리 오버헤드가 크다는 단점을 가진다. 이런 문제점을 해결하기 위해서 다양한 메모리 모델들이 제시되었다.

HLRC(Home-based Lazy Release Consistency) 는 가장 많이 쓰이는 메모리 모델의 하나로, twinning 과 diffing 기법에 기반한 multiple-writer 모델을 사용한다 [4]. HLRC 프로토콜의 특징은 모든 공유 페이지에 대해서 지정된 홈을 가지는 것이고, 홈의 위치가 전체 성능에 큰 영향을 미친다는 점이다: 홈은 항상 페이지에 대한 최신본을 가지기 때문에, 페이지에 대한 접근은 별도의 프로토콜 처리를 필요치 않는다. 즉, 페이지 폴트, twin 생성, diff 생성 및 diff 계산이 필요 없다(이것을 홈 효과[4]라 부른다). 홈이 아닌 노드에서는 수정된 페이지에 대해서 diff 를 계산해서 홈으로 전송해야 하고, 홈에서는 전송된 diff 를 적용해야 한다.

홈 효과를 활용하기 위해서 다양한 Home Migration(HM) [3] 기법들이 제안되었는데, 일반적으로 다음과 같은 방식을 이용한다: “어떤 페이지가 single writer multiple reader 공유 패턴을 보인다면, 해당 페이지의 홈을 single writer 노드로 이전한다”. 하지만, 응용 프로그램의 특성에 따라 홈이 전체 노드들 중 특정 노드로 몰리게 되면, 오히려 성능 저하가 일어날 수 있다.

Direct diff(DD)[6] 기법은 사용자 수준 통신(ULC: User Level Communication)이 지원되는 소프트웨어 DSM 에 적용할 수 있다. 페이지를 수정한 노드에서 원거리의 홈 노드로 직접 수정된 내용을 갱신함으로써 프로토콜 오버헤드를 줄일 수가 있다. 하지만, 페이지 내의 수정된 세그먼트 마다 하나의 갱신 메시지가 전송되기 때문에, 한 페이지가 여러 개의 세그먼트로 구성된다면 성능 저하가 발생할 수 있다..

본 논문에서는 위와 같은 문제점을 극복하기 위해서, 특정 노드로 페이지의 홈이 몰리는 것을 방지하기 방법인 Proportional Home Migration(PHM)을 제안하고, 페이지 당 평균 세그먼트 수에 따라 적응적으로 direct diff 방식을 사용하는 Adaptive Direct Diff (ADD) 기법을 제안한다.

제안된 기법의 성능을 평가하기 위해서, KDSM (KAIST Distributed Shared Memory)[1] 시스템 상에서 7 개의 SPLASH-2 응용을 실행시켰다. 그 결과, 응용 프로그램의 특성에 따라 PHM 및 ADD 기법을 적용 시 성능향상을 가져왔으며, 그 기법을 합쳤을 때 시너지 효과가 나는 것을 발견하였다.

논문의 구성은 다음과 같다. 2 절에서 실험에 사용된 Linux 클러스터 구성요소에 대해 설명한다. 3 절에서는 diff 처리 오버헤드를 줄이는 방법을 기술하고, 4 절에서는 그 성능 측정 결과를 설명한다. 5 절에서 결론을 맺는다.

2. Linux 클러스터 구성요소

본 연구는 Myrinet 스위치로 연결된 8 대의 PC Linux 클러스터를 이용하였다. 각 PC 는 850MHz Pentium III CPU, 2GB 메모리, 133MHz LANai9.2 RISC 프로세서를 탑재한 Myrinet NIC M2L-PCI64B 카드를 가진다. PC 들은 1.28Gbps full-duplex Myrinet 스위치로 연결되어 있다. 사용된 OS 는 Linux 2.4.7-10 이다.

클러스터 노드 간의 통신은 사용자 수준 통신(User-

Level Communication)을 지원하는 SVIA[2] 라이브러리를 사용하였다. SVIA 는 Send/Receive 와 RDMA-Write 모델을 지원한다. 본 클러스터 환경에서 SVIA 의 기본 연산 속도는 다음과 같다:

- 4B round-trip latency: 35.8 μs
- 4KB round-trip latency: 229.2 μs
- 최대 ping-pong 대역폭: 786 Mbits/sec

KDSM[1]은 HLRC[4] 프로토콜을 지원하는 완전한 소프트웨어 DSM 시스템이다. KDSM 은 Linux 상에서 사용자 수준 라이브러리로 구현되었고, 통신을 위해 SVIA 를 사용한다. SVIA 는 RDMA-Write 를 통해 무복사 전송(zero-copy transfer) 기능을 제공한다. 무복사 전송은 다음 두 곳에서 사용된다:

- Page fetch: 한 노드에서 페이지 폴트가 발생하면, 해당 노드는 홈 노드에 페이지 요청 메시지를 보낸다. 홈 노드는 RDMA-Write 를 이용해서 최신 페이지 내용을 요청한 노드에게 직접 전달한다.
- Direct diff: 한 노드에서 페이지를 수정하면, 수정된 내용을 diff 로 만들어 홈 노드에 전달해야 한다. 해당 노드에서 홈 노드로 diff 를 보낼 때, 페이지의 수정된 각 세그먼트 마다 RDMA-Write 를 이용해서 홈 노드의 페이지를 직접 갱신한다.

KDSM 은 다음과 같이 home migration 을 지원한다. 배리어에 도착했을 때, 각 프로세스들은 배리어 관리자에게 write-notice 를 전달한다. 배리어 관리자는 모든 프로세스로부터 write-notice 를 받은 후, 분석을 통해 어떤 페이지가 single-writer 페이지인지 판단한다. Single-write 페이지 중 현재 홈 노드가 single-writer 노드와 다르다면 홈을 이동하도록 home migration 메시지를 전체에 전송한다.

3. 프로토콜 오버헤더를 줄이는 방법

3.1 PHM(Proportional Home Migration)

HM 기법을 적용 시, single-writer 특성을 보이는 특정 노드에 홈이 몰리는 경우가 발생할 수 있다. 이 경우, 해당 노드는 다른 노드들로 부터 오는 페이지 요청에 응답하기 위해서 많은 시간을 소요하기 때문에 전체 시스템의 병목 지점으로 동작한다. 이와 같은 현상을 막기 위해서, 본 논문에서는 하나의 홈 노드가 가질 수 있는 최대 페이지 수를 다음과 같이 제한하고자 한다.

$$\max\left(0, G/N(1-\alpha)\right) \le K \le \min\left(G, G/N(1+\alpha)\right), \text{where } \alpha \ge 0$$

G 는 전체 공유 페이지 수, N 은 전체 노드 수, K 는 하나의 홈 노드가 보유할 수 있는 최대 페이지 수, α 는 K 값을 조정하는 비율값을 의미한다. G/N 은 노드당 할당된 평균 페이지 수를 의미한다.

PHM 기법의 구현을 위해서, 배리어 관리자는 HM 기법 적용 시 각 노드 별로 위 수식을 만족하는 범위 내에서 홈 페이지를 할당시킨다.

3.2 ADD(Adaptive Direct Diff)

DD 기법은 페이지를 수정한 노드에서 원거리의 홈 노드로 직접 수정된 내용을 갱신함으로써 프로토콜 오버헤드를 줄일 수가 있다. 하지만, 페이지 내의 수정된 세그먼트 마다 하나의 RDMA-Write 메시지를 보내서 내용을 갱신하기 때문에, 페이지 내에 여러 개의 세그먼트가 수정되었다면 성능 저하가 발생할 수 있다.

이와 같은 문제점을 해결하기 위해서, 페이지 내의 수정된 평균 세그먼트의 수에 따라서 적응적으로 DD 를 적용하는 기법을 제안한다. 즉, 배리어 관리자에서 이전 배리어 구간 동안 수정된 모든 페이지의 평균 세그먼트 수를 계산한 후, 이 값이 특정 임계치(정수값)를 넘지 않으면 다음 배리어 구간에도 DD 를 적용하고, 임계치를 넘으면 다음 배리어 구간에서는 DD 를 적용하지 않는다. 사용되는 임계치의 선택은 몇 개의 세그먼트를 RDMA-Write 로 직접 전송하는 것이 하나의 diff 메시지를 만들어 Send 및 Receive 해서 처리하는 것보다 성능이 낮게 나오는 값이다.

4. 성능 평가

이 절에서는 본 논문에서 제안하는 PHM 과 ADD 기법의 성능에 관해 설명한다. 성능 평가를 위해 2 절에서 설명한 Linux 클러스터 환경을 이용하여 7 개의 SPLASH-2[5] 응용을 실행하였다 (표 1).

<표 1> 응용 프로그램 특성

Application	Problem size	Shared data (MB)	Lock	Barrier	Sequential time (sec)
LU	2046×2046	32	0	136	85.7
Ocean	514×514	57	72	330	8.5
FFT	128×128×128 points	96	0	24	46.2
Radix	16M keys	132	6	11	3.9
Raytrace	256×256 car	84	1657	1	8.4
Water-Spatial	8000 mols	16	16	21	80.5
Barnes	32768 bodys	7	0	27	58.5

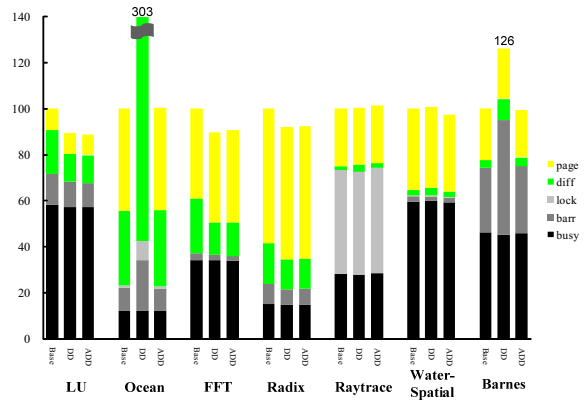
표 2 는 α값의 변화에 따른 PHM 기법을 적용한 결과를 정규화된 실행시간으로 보여준다. 표 내의 값들은 라운드 로빈 방식으로 홈을 할당한 후 HM 없이 실행했을 때의 결과("Base")를 기준으로 정규화된 값이다. 먼저, "HM"은 제한없이 HM 을 적용했을 때인데, LU, Ocean, FFT 는 성능이 향상되고 Radix, Raytrace, Water-Spatial, Barnes 는 성능이 저하됨을 볼 수 있다. PHM 기법을 적용했을 때 아주 작은 α값(=0.1)을 이용했을 때에도 LU, Ocean, FFT 는 충분한 성능 향상을 가져옴을 볼 수 있다. 반면 α가 커짐에 따라 오버헤드가 늘어 Radix, Water-Spatial 의 경우 성능 저하가 발

<표 2> PHM 에서 α값에 따른 성능

	LU	Ocean	FFT	Radix	Raytrace	Water-Spatial	Barnes
Base	100.0	100.0	100.0	100.0	100.0	100.0	100.0
PHM(0.1)	76.1	73.9	75.6	101.6	101.1	101.9	99.1
PHM(0.2)	75.1	73.3	74.7	102.2	101.5	103.3	96.0
PHM(0.3)	75.5	73.7	77.5	103.6	102.3	103.8	100.2
PHM(0.4)	75.3	73.5	74.2	104.9	101.9	104.4	99.4
PHM(0.5)	77.4	73.6	76.8	105.3	102.3	105.0	99.3
HM	75.7	73.6	74.7	105.6	103.2	111.4	106.1

생한다. 성능 측정에 사용된 응용에 대해서 α 값이 0.2 일 때 최적의 성능을 보임을 알 수 있다.

그림 1 은 ADD 기법을 적용했을 때 성능 향상을 보여준다. 그림에서 “Base”는 아무런 기법도 적용하지 않았을 때이고, “DD”는 DD 기법 적용, “ADD”는 임계치를 3 으로 했을 때 ADD 기법 적용 결과이다(3 이란 값은 본 Linux 클러스터 환경에서 선택된 값이다). DD 기법은 LU, FFT, Radix 에 대해 성능 향상을 가져오지만, Ocean 과 Barnes 는 성능 저하가 나타난다. 이것은 LU, FFT, Radix 가 페이지 당 평균 1~2 개의 세그먼트를 가지는데 반해, Ocean 과 Barnes 는 각각 15.6 과 12.5 로 세그먼트 수가 상당히 많기 때문이다. 따라서 ADD 적용을 통해 Ocean 과 Barnes 는 그림 1 과 같이 성능 저하 현상을 막을 수 있다.

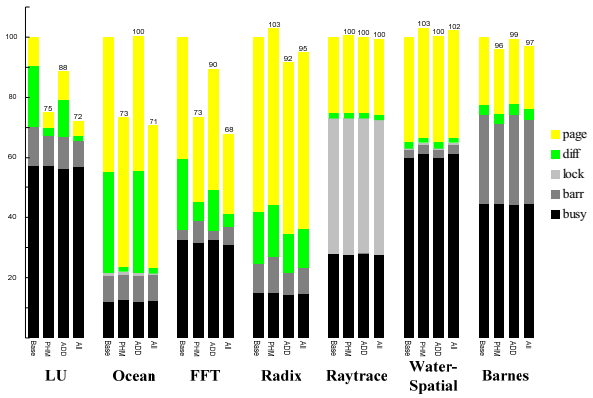


(그림 1) DD 와 ADD 의 성능 비교(임계치=3)

마지막으로, PHM 기법과 ADD 기법을 동시에 사용했을 때의 효과에 대해 살펴본다. 먼저, 표 3 은 PHM 적용 전후 diff 를 생성하는 총 페이지 수와 그 때 페이지 당 평균 세그먼트 수를 보여준다. 대부분의 응용에서 PHM 적용 후 diff 를 생성해야 하는 페이지 수 자체가 많이 줄었음을 볼 수 있고, Ocean 과 Barnes 의 경우에는 페이지 당 평균 세그먼트 수도 줄어들었음을 볼 수 있다. 그림 2 는 아무런 기법도 적용하지 않았을 때(“Base”), PHM 기법만 적용했을 때(“PHM”), ADD 기법만 적용했을 때(“ADD”), 그리고 PHP 과 ADD 기법을 모두 적용했을 때(“All”)의 결과를 보여준다. 그림 2 에서, 두 기법을 모두 적용했을 때 LU, Ocean, FFT 의 경우 개별 기법을 적용했을 때 보다 성능이 좋아졌으며, 나머지 응용은 두 기법 중 효과가 더 좋은 하나를 선택했을 때보다 성능이 조금 떨어졌다. 특히 FFT 의 경우, PHM 적용 후 ADD 를 적용하면, 평균 세그먼트는 1 이면서 diff 를 보내야 하는 페이지가 상대적으로 많기 때문에 두 기법 모두로부터 이득을 볼 수 있다.

<표 3> diff 를 생성하는 총 페이지 수 및 페이지 당 평균 세그먼트 수

		LU	Ocean	FFT	Radix	Raytrace	Water-Spatial	Barnes
Before PHM	#Page	7204	11881	21518	38979	1680	3654	5643
	#Segment	1.0	15.6	1.0	1.0	1.3	2.1	12.5
After PHM	#Page	43	21	6286	36601	234	2	768
	#Segment	1.0	1.0	1.0	1.0	1.3	2.1	5.4



(그림 2) PHM 과 ADD 를 합친 효과

5. 결론

본 논문에서는 기존 home migration 기법과 direct diff 기법의 문제점을 극복하기 위해 새로운 적응적인 기법을 제시하였다. PHM 기법은 특정 노드로 홈이 몰리는 것을 방지하였고, ADD 기법은 페이지 당 수정된 세그먼트가 많을 때 너무 많은 갱신 메시지가 발생하지 않도록 하였다. 성능 측정 결과, 응용 프로그램에 따라서 한 기법만 적용했을 때 성능이 좋은 경우와 두 기법을 모두 적용했을 때 성능이 더 좋은 경우가 있었다.

참고문헌

[1] S. Lee, H. Yun, J. Lee, and S. Maeng. “Adaptive Prefetching Technique for Shared Virtual Memory”. In Proc. of WDSM, May 2001.

[2] M. Lee, J. Yu, and S. Maeng. “An Efficient Implementation of Virtual Interface Architecture using Adaptive Transfer Mechanism on Myrinet”. In Proc. of ICPADS, June 2001.

[3] W. Hu, W. Shi, and Z. Tang. “Home Migration in Home-based Software DSMs”. In Proc. of the 1st Workshop on Software Distributed Shared Memory, June 1999.

[4] Y. Zhou, L. Iftode, and K.Li. “Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Virtual Memory Systems”. In Proc. of 2nd OSDI, October 1996.

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

[6] A. Bilas, C. Liao, and J. Singh, “Using Network Interface Support to Avoid Asynchronous Protocol Processing in Shared Virtual Memory Systems”. In Proc. of ISCA 1999.