MPI상에서 분산공유메모리(DSM) 시스템의 구현

장우현○ 이성우 유기영
경북대학교 컴퓨터공학과

Implementation DSM system over MPI

Woohyun Jang Sungwoo Lee Keeyoung Yoo
Department of Computer Engineering Kyungpook Nat'l University

요 약

본 논문에서는 MPI를 이용하여 분산공유메모리 시스템을 구현한다. 또한 사이클이 없는 방향성 그래프를 기반으로 한 분산 락 알고리즘을 기반으로 네트웍 환경에 적당한 알고리즘을 제안하고 구현한다. 사용된 MPI는 분산 메모리 시스템의 메시지 교환의 표준이므로 MPI가 구현되어 있는 대부분의 분산메모리 시스템에서 활용이 가능하여 높은 이식성을 가진다

1. 서론

최근에 많은 계산이 필요한 응용프로그램들이 많이 나타남에 따라 멀티프로세서(Multiprocessor)에 관한 많은 연구와 시스템들이 개발되어 왔다 이들은 메모리의 구조에 따라 크게 공유메모리 시스템과 분산 메모리 시스템으로 분류될 수 있다. 이 중 공유메모리 시스템은 모든 프로세서가 동일하게 접근 가능한 메모리를 사용한다. 이 시스템은 쉽고, 편리한 프로그램 개발환경을 제공하지만 프로세서의 수를 늘일수록 메모리 접근 시간이 떨어지고, 메모리의 구조가 복잡해지는 문제가 있다. 반면에 분산메모리 시스템은 각각 독립적인 메모리를 가지는 노드들이 네트웍으로 연결되어 있다. 이런 분산메모리 시스템은 확장성이 좋아 높은 컴퓨팅 능력을 만들 수 있지만, 다른 노드에 있는 자료를 접근하기 위해서는 메시지 전달 (Message Passing)을 사용해야 하기 때문에 프로그래밍 하기가 어렵다

그 이후 이 두 가지 모델의 장점만을 결합한 분산공유메모리 [4] 모델이 등장했다 이 분산공유메모리(Distributed Shared Memory, DSM) 시스템은 물리적으로는 분산된 메모리 시스템 상에서 논리적으로는 공유된 메모리 시스템을 제공한다. DSM 시스템의 설계자는 다양한 하드웨어/소프트웨어 방법을 이용하여 응용프로그래머에게 하부의 통신방법을 숨겨서 공유메모리 시스템처럼 프로그래밍을 쉽게 해 준다. 또한 이 시스템은 하위의 분산메모리 시스템이 가지는 확장성과 저 비용 요소를 그대로 가지고 있다.

본 논문에서는 기존의 공유메모리 시스템 환경에서 제공되는 프로그래밍 모델을 그대로 유지하고, 이기종 환경에서 강한 이식성을 가지며, 보다 나은 성능을 가질 수 있는 소프트웨어 DSM시스템을 구현한다 이를 위해 빠른 해제 일관성 모델(Eager Release Consistency Model) [3]을 기반으로 하고, 네트웍 환경에 적당한 분산 동기화 알고리즘을 제안하여, 분산메모리 시스템 상에서 메시지 패싱 인터페이스의 표준인 MPI(Message Passing Interface) [7]를 이용하여 구현한다.

본 논문의 구성은 다음과 같다 2장에서는 시스템의 전체적인 구성을 설명하고, 3장에서는 구현에 필요한 자료구조와 방법 등을 설명하며, 4장에서는 실험 환경과 실험 결과를 제시하며, 4장에서 결론을 맺는다

2. 설계

본 시스템의 전체적인 구성은 그림 1과 같다. 하나의 프로세스는 다른 노드의 요청을 처리하는 서버 스레드와 병렬프로그램을 수행하며 필요에 따라서는 다른 노드로 요청을 하는 작업 스레드로 구성된다 서버 스레드와 작업 스레드는 외부노드와 자료를 교환하기 위해서 MPI를 이용한다. 프로그래머는 본 시스템이 제공하는 라이브러리를 사용함으로서 공유메모리 시스템의 병렬프로그램을 수행할 수 있다.

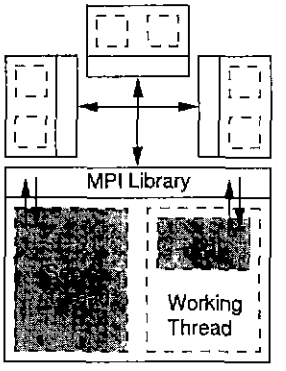

그림 1. 시스템 구성도

2.1 공유메모리

DSM시스템은 크게 공유메모리의 종류가 페이지인 페이지 기반 시스템(Page Based System)과, 공유메모리가 변수인 변수 기반 시스템(Variable Based System)으로 나뉜다. 본 시스템은 기본 통신 단위가 변수인 MPI를 이용하기 때문에 공유메모리의 단위로 변수를 사용한다.

본 시스템에서 제공하는 공유변수는 두가지 형태이다. 하나는 단일 변수, 즉 정수형 또는 부동소숫점형과 같은 하나의 변수이며, 다른 하나는 이들의 벡터, 즉 동일한 형이 여러개 모여 있는 공유변수이다.

2.2 DSM 알고리즘

DSM시스템은 공유변수의 접근을 빠르게 하기 위해 공유변수를 지역메모리에 캐싱(Caching)한다. 따라서 이 경우 여러 노드에 걸쳐 분포하는 캐쉬의 값을 일관되게 유지해 주는 DSM알고리즘이 필요하다.

DSM알고리즘은 크게 3가지로 나눌 수 있다 [8]. 공유메모리가 접근할때 마다 여러 시스템들 사이를 옮겨다니는 SRSW(Single Reader Single Writer), 여러 노드들에 읽기만 가능한 공유메모리를 두고, 단지 하나의 노드만 공유메모리에 쓸 수 있는 MRSW(Multiple Reader Single Writer), 마지막으로 본 시스템이 사용하고 있는

MRMW(Multiple Reader Multiple Writer)가 있다. 완전-중복 알고리즘이라고 불리기도 하는 MRMW알고리즘은 읽기권한과 쓰기권한 둘 다 가진 데이터 블럭의 중복을 허용한다. 그러므로 공유메모리의 접근시 페이지를 다른 노드에서 가져오는 동의 작업이 없기 때문에 다른 알고리즘에 비해 성능이 우수하다. 이 알고리즘에서는 일관성을 유지하기 위해 공유변수의 수정을 멀티 캐스트나 방송 메시지로 보내어 다른 노드의 캐쉬들의 값을 변경한다. 하지만 이 알고리즘은 특별히 갱신 빈도가 높고 복사된 사본의 수가 많은 경우에 있어서는 일관성 유지를 위한 통신의 횟수가 많아질 수 있다.

2.3 일관성 모델

DSM시스템에서 가장 중요한 내용은 자료의 일관성을 유지하면서 공유메모리의 접근에 소요되는 평균시간을 최소화하는 메모리 일관성 모델(Memory Consistency Model)이다. 좀더 효율적인 일관성 모델을 위해서 많은 연구들이 행해져 왔으며, 그 대표적인 모델로는 순차적 일관성 모델(Sequential Consistency Model) [5], 빠른 해제 일관성 모델(Eager Release Consistency Model) [3], 늦은 해제 일관성 모델(Lazy Release Consistency Model) [9], 등록 일관성 모델 (Entry Consistency Model) [1]등이 있다

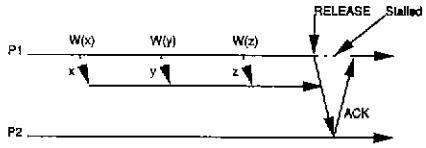

그림 2. 빠른 해제 일관성 모델

빠른 해제 일관성 모델은 프로그래머는 공유변수를 사용할 때 동기화변수를 사용한다는 사실을 이용한다. 즉 어떤 노드가 임계영역에서 일을 수행하고 있을 때 다른 노드는 임계영역에 들어가지 못하기 때문에 다른 노드가 사용하고 있는 공유변수를 접근할 수 없다. 즉 어떤 노드가 임계영역에서 수정한 내용은 임계영역을 벗어날때까지 다른 노드의 캐쉬는 갱신되지 않아도 상관없다. 그러므로 만일 프로그램이 모든 공유변수의 접근에 동기화를 해 준다면, 빠른 해제 일관성 모델은 순차적일관성모델과 동일한 결과를 낸다. 그림 2에서 보듯이 P1에서 사용하는 x,y,z는 P1의 임계영역이 끝날때 까지 P2는 사용할 수 없기 때문에 P2의 값이 갱신되는것은 임계영역이 끝날때 일어나면 된다. 이렇게 함으로서 원래 3번의 메시지가 전달되어야 하는 것을 1번의 메시지 전달로 줄일 수 있다.

2.4 분산 동기화 알고리즘

보통 공유메모리 시스템에서 락이나 배리어를 구현할 때는 공유메모리 자체를 이용하지만, 분산공유메모리 시스템에서는 공유메모리를 이용할 수 없기 때문에 다양한 방법의 알고리즘들이 연구되어져 왔다.

분산메모리 시스템에서 락을 구현하는 방법은 크게 허가에 의한 방법(Permission Based Algorithm)과 토큰에 의한 방법(Token Based Algorithm)이 있다. 본 시스템은 그 중에서 토큰에 의한 알고리즘 중의 하나인 사이클이 없는 방향성 트리(Direct Acyclic Graph, DAG)에 기초한 알고리즘을 개선한 알고리즘을 제안하고 구현한다.

2.4.1 락(lock)

락은 사이클이 없는 방향성 그래프(Direct Acyclic Graph, DAG)를 기반으로 하는 알고리즘 [6]을 변경해서 구현했다. 기존의 DAG기반 알고리즘은 처음에 만들어진 논리적인 트리를 유지하며, 토큰이 이동함에 따라 간선의 방향만 변하여 토큰이 있는 노드를 가리킨다. 이 방법은 병렬시스템과 같이 네트웍의 형태가 일정하여 노드간의 거리가 노드의 위치에 따라 다른 경우에는 뛰어난 성능을 발휘하지만, 일반 네트웍과 같이 노드의 위치에 상관없이 같은 거리를 가지는 환경에서는 적당하지 않다

따라서 본 시스템에서는 정적인 DAG기반 알고리즘을 변형하여 논리적 트리가 시간이 흐름에 따라 변하는 동적인 DAG기반 알고리즘을 사용한다. 동적인 DAG기반 알고리즘에서는 정적인 DAG기반 알고리즘과 마찬가지로 각 노드에 3개의 변수를 사용한다. 만일 현재 노드가 토큰을 가지고 있다면 HOLDING은 참이 되며, 만일 없다면 LAST기 가리키는 노드로 요청메시지를 보낸다. 토큰을 가지고 있는 노드는 LAST가 자기 자신을 가리킨다. 마지막으로 NEXT는 현재 노드가 임계영역을 수행한 후 토큰을 넘겨줘야 하는 노드를 가리킨다. 노드가 임계영역을 수행한 후 NEXT변수를 검사해서 토큰을 기다리는 노드가 있다면 기다리는 노드로 PRIVILEGE메시지를 보내준다. DAG기반 알고리즘 에서는 PRIVILEGE메시지가 토큰의 역할을 한다. 그리고 모든 노드들은 LAST의 값에 따라 논리적인 트리를 이룬다

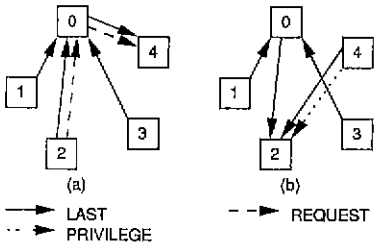

그림 3. 락의 논리적 트리

그림 3 은 락의 논리적 트리가 변하는 모습을 보여주고 있다. 만일 현재의 논리적 트리가 그림 3 (a)와 같고 2번 노드가 락을 요청 했을 때는 2번 노드의 LAST는 0을 가리키고 있으므로 REQUEST를 0으로 보낸다. 0번 노드는 현재 LAST가 4를 가리키므로 0번 노드의 LAST를 2번으로 바꾸고, 4번으로 REQUEST를 보낸다. 현재 4번 노드가 임계영역을 수행중이라면 4번 노드의 NEXT를 2로 설정해 둔다. 4번 노드가 임계영역의 수행을 마쳤을 때 NEXT가 2번 노드 를 가리키고 있기 때문에 2번 노드로 PRIVILEGE 메시지를 보내고, LAST를 2번을 가리키게 한다. 결과적으로 그림 3 (b)와 같은 형태가 된다.

정적인 DAG기반 알고리즘에서는 위와 같은 경우 4번노드가 2번노드를 바로 가리키지 않고 방향만 변하여 4번이 0번을 가리킨다. 이 경우 다음에 4번 노드가 락을 요청한 경우 REQUEST메시지를 바로 2번노드로 보내지 않고, 0번 노드를 거쳐가게 된다.

3. 구현

3.1 자료구조(Data Structure)

그림 4는 사용된 자료구조를 대략적으로 보여준다. 이 자료구조는 모든 노드들에 존재한다. 본 시스템에서 사용되는 중요한 자료구조는 일반 공유변수를 관리하는 단순공유변수배열(Shared Object Array), 벡터 공유변수를 관리하는 벡터공유변수배열(Vector Shared Object Array), 락에 관한 정보를 저장하는 락배열(Locks Array), 배리어에 관한 정보를 저장하는 배리어배열(Barriers Array), 마지막으로 공유변수갱신레코드이다. 공유변수갱신레코드는 현재 노드에서 일어나는 공유변수의 수정을 저장하고 있는 자료구조로서 동기화때 다른 노드들로 보내져 공유변수의 일관성을 유지한다.

단순공유변수배열의 하나의 요소가 하나의 단순공유변수를 가리키고 있다. 하나의 요소에는 단순공유변수의 형과 실제 메모리의 주소를 가지고 있다. 벡터공유변수배열의 하나의 요소는 하나의 벡터공유변수를 가리키고 있다. 하지만 단순공유변수와 다르게 벡터공유변수는 크기가 큰경우에는 페이지로 나누어서 관리한다. 이렇게 페이지를 따로 관리하고, 각각의 페이지에 현재는 사용하고 있지 않지만, 페이지의 소유주와 페이지의 상태를 저장할 영역을 남겨 두었기 때문에 현재의 변수기반 시스템에서 페이지 기반 시스템으로의 변환도 쉽게 하고 있다.

프로그램이 공유변수에 값을 쓴 경우, 그 정보를 공유변수갱신레코드에 추가한다. 프로세스는 동기화가 일어날때 까지 공유변수갱신레코드를 유지하다가 동기화가 일어날 때 다른 노드들로 갱신메시지를 보내줘서 공유변수의 일관성을 유지한다.

락과 배리어의 구현을 위해 락배열과 배리어배열을 사용한다. 락배열과 배리어배열의 하나의 요소는 하나의 락과 배리어를 관리한다.

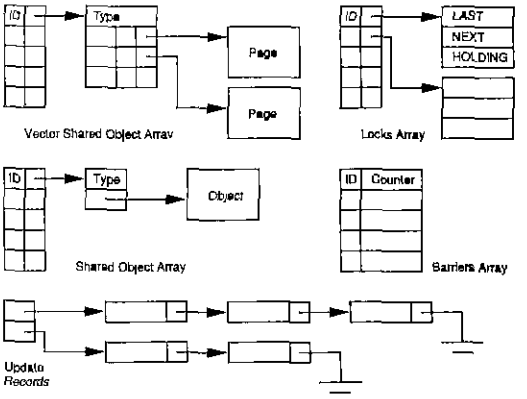

그림 4. 자료구조

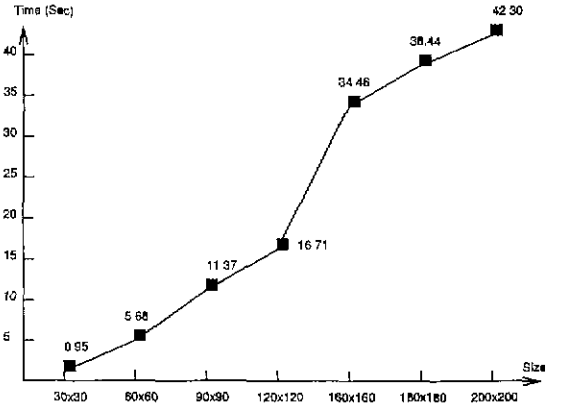

그림 5. 실험결과

3.2 락(lock)

본 시스템에서는 위에서 제안한 동적인 DAG기반 알고리즘을 사용하며, 여러개의 락을 관리하기 위해서 락배열을 유지한다. 이 락배열은 모든 노드가 가지고 있으며, 이 락배열의 각각의 요소는 하나의 락을 관리하는 자료구조를 기리킨다. 하나의 락을 관리하는 자료구조는 동적인 DAG기반 알고리즘에서 필요한 HOLDING, LAST, NEXT를 가진다

3.3 배리어(barrier)

본 시스템에서 배리어는 중앙집중식 방법을 사용했다. 또한 락과 마찬가지로 프로그램에서 하나의 배리어가 아니라 여러개의 독립적인 배리어를 쓸 수 있도록 하기 위해 락과 유사하게 모든 노드가 배리어배열을 가지며, 배열의 각각의 요소는 그 배리어가 유지하는 자료구조인 COUNTER를 가진다

4. 실험 및 결과

4.1 실험환경

본 논문에서 제시한 분산공유메모리 시스템을 이더넷(Ethernet)으로 연결된 Sun 워크스테이션 클러스트에서 구현했다 본 시스템에서 사용하는 MPI는 오하이오 주립대학(Ohio State University) 수퍼컴퓨터센터에서 개발한 LAM 6.1이다 [2].

4.2 실험결과

그림 5는 행렬곱셈프로그램을 구현한 시스템의 성능을 측정하기 위해서 작성하고, 그 성능을 나타낸다. 성능 측정에 사용된 환경은 Sun워크스테이션 3대를 10Mbps의 이더넷으로 묶어서 사용했다.

사용한 행렬곱셈 프로그램은 다음과 같다. 0번 노드는 메모리를 초기화하며, 0번 노드의 초기화가 끝나면 모든 노드들이 행의 $\frac{1}{3}$씩 계산하여 공유변수에 쓴다. 모든 노드의 계산이 끝나면, 0번 노드는 공유변수를 읽어서 화면에 출력한다.

5. 결론

본 논문에서는 기존의 공유메모리 시스템 환경에서 제공되는 프로그래밍 모델을 그대로 유지하고, 이기종 환경에서 강한 이식성을 가지며, 보다 나은 성능을 가질 수 있는 소프트웨어 DSM시스템을 구현했다. 이를 위해 빠른 일관성 모델을 기반으로 하고, 네트웍 환경에 적합한 분산 동기화 알고리즘을 제안하여, 분산메모리시스템의 메시지교환표준인 MPI를 이용하여 구현하였다. 또한 페이지관리등의 일들을 독립적인 서버로 만들지 않고, 스레드의 표준인 POSIX스레드를 이용하여 구현하였다.

본 시스템은 메시지교환의 표준인 MPI를 이용하고, 스레드의 표준인 POSIX를 이용함으로서 높은 이식성을 가지며, 서버의 기능을 독립적인 프로세스로 두지 않고, 스레드를 이용함으로서 높은 성능을 가진다. 또한 기존의 시스템들이 대부분 분산락을 구현할 때 중앙집중식 방법을 사용하거나, 각각의 관리자를 지정해서 사용했는것에 반해, 본 시스템은 동적인 DAG기반 알고리즘을 사용하여 많은 프로세스가 사용될 때 높은 성능을 발휘한다.

하지만, 대부분의 MPI 라이브러리들이 프로세스기반으로 제작되었기 때문에 멀티스레드 환경에 적합하지 않아서 얼마의 오버헤드를 가지며, 갱신 프로토콜의 최적화가 이루어지지 않아서 오버헤드를 가진다. 하지만, 곧 각종 MPI라이브러리들도 스레드를 지원할것이며, 갱신프로토콜 역시 최적화 되면, 성능은 상당히 향상되리라고 본다.

참고문헌

[1] B. N Bershad, M. J. Zekauskas, W.A. Sawdon, "The Midway Distributed Shared Memory System," *Proc. of the 38th IEEE Int'l Computer Conf.*(COMPCON Spring'93), pp. 528-537, February 1993.

[2] *MPI Primer / Developing With LAM*, The Ohio State University, 1996.

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

[4] J Protic, M. Tomasevic, and V Milutinovic, "Distributed Shared Memory: Concepts and Systems," *IEEE Parallel & Distributed Technology*, pp 63-79, Summer 1996.

[5] L Lamport, "How to make a multiprocessor computer that crrrectly executes multiprocess programs," *IEEE Trans. Computer* C-28, 9(Sept.), pp. 690-691.

[6] M. L. Neilsen, M. Mizuno, "A Dag-Based Algorithm for Distributed Mutual Exclusion," *Proc. 11th Int'l Conf. Distributed Computer Systems*, pp. 354-360, May 1991.

[7] M. Snir et al., *MPI: The Complete Reference*, The MIT Press, 1996.

[8] M. Stumm and S. Zhou, "Algorithms Implementing Distributed Shared Memory," *IEEE Computer*, Vol. 23, No. 5, pp 54-64, May 1990.

[9] P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy Release Consistency for Software Distributed Shared Memory," *Proc. 19th Ann. Int'l Symp. Computer Architecture*, CS Press, pp. 13-21, 1992