Zero-copy 기술을 이용한 PVM의 성능 개선

정회원 임 성 택*, 심 재 홍*, 최 경 희*, 정 기 현**, 김 재 훈*, 문 성 근***

Performance Improvement for PVM by Zero-copy Mechanism

Sung-taek Lim*, Jae-hong Shim*, Kyung-hee Choi*, Gi-hyun Jung**, Jai-hoon Kim*, Kyung-Duk Moon*** *Regular Members*

요 약

PVM(Parallel Virtual Machine)은 네트워크 상에 분산되어 있는 여러 시스템들을 투명하게 활용하여 사용자에게 고성능 병렬 컴퓨팅을 지원하는 단일 가상 시스템(single virtual system)으로 인식되게 하는 middle-ware 소프트웨어이다. 초고속 통신망을 기반으로 하는 PVM에서는 임의의 태스크로부터 원격 태스크로 하나의 메시지를 전송하기 위해 세 번의 메시지 복사가 필요하며, 이는 PVM의 성능 저하를 가져오는 주요 원인이 된다. 이러한 문제점을 개선하기 위해 이 논문에서는 zero-copy PVM 통신 모델을 제안한다. 제안된 모델은 PVM 태스크, PVM 데몬, 네트워크 인터페이스 보드 등에 의해 동시 접근이 가능한 전역 공유 메모리(global shared memory)를 이용하며, 초고속 통신망을 기반으로 하고 있다. 이 모델에서 PVM 태스크는 전송하고자 하는 메시지를 전역 공유 메모리에 저장하고, 메시지를 보낼 준비가 되었음을 PVM 데몬에게 알리며, 데몬은 해당 메시지를 커널을 통하지 않고 바로 초고속 통신망으로 전송함으로써, 메시지의 메모리 복사 횟수를 감소 시킨다. 실험 결과 두 시스템간의 메시지 왕복 시간은 제안된 모델을 사용함으로써 현저히 줄어 들었음을 확인하였다.

ABSTRACT

PVM provides users with a single image of high performance parallel computing machine by collecting machines distributed over a network. Low communication overhead is essential to effectively run applications on PVM based platforms. In the original PVM, three times of memory copies are required for a PVM task to send a message to a remote task, which results in performance degradation. We propose a zero-copy model using global shared memory that can be accessed by PVM tasks, PVM daemon, and network interface card (NIC). In the scheme, a task packs data into global shared memory, and notify daemon that the data is ready to be sent, then daemon routes the data to a remote task to which it is sent with no virtual data copy overhead. Experimental result reveals that the message round trip time between two machines is reduced significantly in the proposed zero-copy scheme.

I. 서 론

병렬 계산은 과학 계산, 일기 예보, 화상 처리와 같은 입력 데이터가 크거나 복잡도가 매우 높은 문제를 해결하는데 널리 사용된다. 병렬 처리를 위한 두 가지 주요 기반 시스템으로 다중 병렬 프로세서(MPP)시스템과 분산 컴퓨팅 시스템을 들 수 있다. MPP 시스템은 계산 측면에서는 탁월한 성능을 발휘하나 고가의 장비 가격으로 인해 일반 사용자가 널리 사용하지 못하고 있는 실정이다. 대안으로 네트워크 상의 일반 컴퓨터들로 구성된 분산 컴퓨팅 시스템이 더욱 일반적으로 사용된다.

분산 컴퓨팅 환경에서 병렬 계산을 지원하는 좋

* 아주대학교 정보 및 컴퓨터공학부(ist0123@chollian.net), ** 아주대학교 전기전자공학부
*** 한국전자통신연구소
논문번호 : 00003 -0107, 접수일자 : 2000년 1월 7일

은 예가 PVM 시스템[1]이며, 이는 네트워크 상에 분산되어 있는 여러 시스템들을 투명하고 효율적으로 활용하여 사용자에게 고성능 병렬 컴퓨팅을 지원하는 단일 가상 시스템(single virtual system)으로 인식되게 하는 middle-ware 소프트웨어이다.

PVM 시스템은 PVM 데몬, PVM 태스크, 그리고 PVM 라이브러리로 구성된다[1]. PVM 라이브러리는 응용 프로그램 개발자에게 점대점(point to point) 통신, 메시지 중계(broadcast), 상호 배제(mutual exclusion), 프로세스 컨트롤, 다양한 동기화(synchronization) 등과 같은 편리한 프로그램 개발 환경을 제공한다. PVM 데몬은 (1)메시지의 제어 및 경로 설정 (2)네트워크 상에 연결된 여러 시스템으로의 투명한(transparent) 작업(태스크) 분배, 사용자에게 단일 가상 시스템으로서의 이미지 인식 (3) 새로운 호스트(단위 시스템)의 추가 및 기존 호스트의 삭제 등과 같은 다양한 기능들을 제공한다.

PVM 라이브러리를 사용하는 PVM 응용 프로그램은 여러 개의 PVM 태스크들로 구성되며, 각각의 태스크는 네트워크 상에 투명하게 분산되어 수행된다. 많은 요소들이 PVM 성능에 영향을 미치는데 이 중 가장 중요한 요소가 PVM 시스템을 구성하는 호스트들간의 통신 메카니즘(mechanism)이다[2,6,7,8,10]. PVM 은 태스크 병렬 및 데이터 병렬과 같은 두 가지 형태의 병렬 작업을 지원한다. 이러한 병렬 작업을 수행하는 응용 프로그램을 위해서는 낮은 데이터 통신 지연(low communication latency) 및 태스크 상호간 빠른 동기화(synchronization)가 필수적이다.

통신 지연을 초래하는 주요 요인으로서 사용자 주소 공간의 메시지를 커널 주소 공간으로 옮기고, 커널 내의 여러 계층적 통신 프로토콜들을 통과하면서 발생하는 중복된 메시지 복사 부하를 들 수 있다. 이러한 메시지 복사 부하는 저속 네트워크 상에서는 큰 영향을 미치지 않지만, 초고속 네트워크의 출현과 더불어 통신 지연의 중요한 요인으로 부각되었다. 따라서 최근 메시지 복사 부하를 줄이기 위한 방안으로 zero-copy 기술이 여러 연구에서 제안되었다[3,4,5,17,18].

Zero-copy 기술은 기존의 커널 내에 구현된 통신 프로토콜을 사용하지 않고, 사용자 주소 공간에서 초고속 네트워크 인터페이스 보드 (NIC: Network Interface Card)로 바로 메시지를 전송하는 기술이다. Francis O'Carrol[4]은 기존 MPI(Message Passing Interfaces)를 zero-copy 기술을 활용하여 재 구현하였다. Zero-copy 기술은 또한 ATM상의 U-Net을 구현하는 데에도 적용되었다[5].

PVM 시스템은 Myrinet, FDDI, HiPPI, SONET, 그리고 ATM과 같은 초고속 네트워크의 출현과 더불어 고성능 병렬 컴퓨팅을 위한 기반 시스템으로 부각되기 시작했다. 그러나 PVM은 여전히 범용 socket API를 이용한 커널 내의 TCP/IP 프로토콜에 의존하고 있으며, 이는 초고속 통신망을 기반 네트워크로 사용할 경우 통신 지연의 주요한 요인으로 작용한다. 즉, 여러 번에 걸친 메시지 복사로 인해 커널 자체가 메시지 전송 경로상의 병목 현상을 초래하게 된다.

따라서 이 논문에서는 초고속 통신망을 기반으로 하는 PVM 시스템의 성능 향상을 위한 zero-copy PVM 통신 모델을 제안한다. 제안된 모델은 PVM 태스크, PVM 데몬, 네트워크 인터페이스 보드(NIC)에 의해 동시 접근이 가능한 전역 공유 메모리(global shared memory)를 이용하며 초고속 네트워크를 기반으로 하고 있다. 이 모델에서 PVM 태스크는 전송하고자 하는 메시지를 전역 공유 메모리에 저장하고, 메시지를 보낼 준비가 되었음을 PVM 데몬에게 알리며, 데몬은 해당 메시지를 커널을 통하지 않고 바로 초고속 네트워크 보드로 전송함으로써, 메시지의 메모리 복사 횟수를 감소 시킨다. 두 시스템간의 메시지 왕복 시간은 제안된 모델을 사용함으로써 현저히 줄일 수 있음을 실험을 통해 확인해 본다.

이 논문의 구성은 다음과 같다. 2장에서는 PVM의 성능 향상과 관련한 다양한 연구결과를 살펴보고, 3장에서 기존의 PVM 통신 모델을 분석한다. 초고속 네트워크 상에서의 PVM 성능 향상을 위한 zero-copy PVM 통신 모델을 4장에서 제안하며, 이의 구체적인 구현 방안과 알고리즘을 5장에서 논의한다. 구현된 모델의 타당성을 확인하기 위해 다양한 실험을 수행하였으며, 실험 방법 및 실험 결과 그리고 이의 분석 등이 6장에서 논의된다.

Ⅱ. 관련 연구

앞서 언급한 바와 같이 많은 요소들이 PVM 성능에 영향을 미치고 있으며, 가장 중요한 요소가 PVM 시스템을 구성하는 호스트들간에 사용되는 통신 프로토콜이다. 또한 PVM이 지원하는 병렬 프로그래밍 모델을 위해 낮은 데이터 통신 지연 및 태스크 상호간 빠른 동기화가 필수적이다. 이러한 이

유로 인해 PVM 호스트간의 통신 성능을 향상시키려는 노력이 많은 연구팀에 의해 진행되었으며, 주로 통신 프로토콜 개선과 초고속 네트워크 사용 등 2가지 분야에서 활발한 연구가 진행되었다.

2.1 통신 프로토콜 개선

H. Xu[6]는 초고속 네트워크인 Myrinet상에서 고속의 PVM 통신 프로토콜인 ATOMIC LAN을 설계하고 구현하였다. 사용자들은 Myrinet상에서 다음과 같은 두 가지 통신 프로토콜 중 하나를 이용할 수 있다: (1)기존 socket을 이용한 TCP/IP 프로토콜 (2)사용자 수준의 ATOMIC LAN API를 이용한 ATOMIC LAN 프로토콜. 기존에 이미 구현된 커널 내의 TCP/IP 프로토콜은 커널 버퍼로의 데이터 복사를 요구하는 반면, ATOMIC LAN 프로토콜을 이용할 경우 커널의 간섭 없이 태스크가 직접 네트워크 인터페이스 보드에 접근할 수 있다. ATOMIC LAN 프로토콜은 Myrinet-API를 사용하며, 제어 메시지 전달 경로와 일반 데이터 메시지 전달 경로를 서로 다르게 이용한다는데 그 특징이 있다.

S. L. Chang[7]은 Fore System의 ATM API를 사용한 ATM 네트워크 상에 PVM 시스템을 구현하였다. 그러나 Fore System의 ATM API는 데이터 흐름 제어(flow control)를 지원하지 않으므로, 이로 인한 데이터 분실을 방지하기 위해 선택적 재전송 방법(selective retransmission mechanism)을 사용하는 흐름 제어 방안을 제안하였다. 연구자들은 이를 활용할 경우 성능 저하 없이 신뢰성 있게 메시지를 전달할 수 있다고 주장한다.

H. Zhou[8]는 ATM 네트워크 상에서 PVM을 위한 별도의 메시지 전달 경로인 PvmRouteAtm을 제안하였다. 이 경로는 socket API를 사용하되 기존의 커널을 통하는 TCP/IP 프로토콜(이를 PvmRoute Direct 경로라 함)을 이용하는 대신 사용자 수준에서 직접 ATM API를 사용한다. 이 경로를 이용할 경우 태스크는 PVM 데몬과 커널을 통하지 않고 바로 원격 태스크에 데이터를 보낼 수 있다.

2.2 초고속 네트워크 사용

Myrinet, ATM, 그리고 HiPPI 등과 같은 초고속 통신망이 PVM 시스템의 기반 네트워크로 주로 활용되었다. Marcia[9]는 Myrinet을 기반으로 하는 워크스테이션 연동 환경하에서 두 가지 PVM을 제안하였다. 하나는 커널 내의 TCP/IP를 사용하는 것이고(커널 내의 Myrinet 드라이브는 Myrinet API를 사용), 다른 하나는 Fast Message(FM) 라이브러리를 사용하는 것이다. J. Hsieh[10]는 HiPPI LAN에서 계층적 프로토콜 스택을 통과하는 부하를 줄이기 위해 소켓 대신에 LLA HiPPI를 사용하였다.

Ⅲ. PVM 통신 모델

PVM은 태스크간 사용되는 통신 모델에 따라 크게 socket을 이용하는 버전과 태스크별 공유 메모리를 이용하는 버전으로 나눌 수 있다[1]. 편의상 전자를 OPVM이라 지칭하고, 후자를 SPVM이라 지칭한다.

OPVM은 모든 통신을 socket을 이용하여 수행하며, 근거리 통신망에 기반을 둔 분산 컴퓨팅 환경에 주로 활용된다. 반면, SPVM은 지역 태스크들(동일 호스트 내에서 수행되는 태스크들) 사이에 보다 빠른 메시지 교환을 위해 태스크별 공유 메모리(shared memory)를 사용하며, 주로 MPP 시스템에서 활용된다.

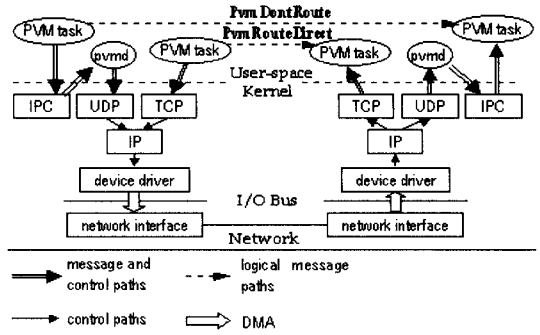


그림 1. 기존의 PVM 통신 모델

그림 1은 OPVM의 통신 모델을 도시화한 것이다. OPVM은 확장성을 유지하기 위해 각 호스트에 하나씩 존재하는 PVM 데몬(pvmd)들 사이에는 UDP/IP socket을 사용하여 통신하고, 호스트 내의 지역 태스크들과 PVM 데몬간 통신에는 UNIX domain stream socket을 사용한다. 그러나 태스크 상호간 통신을 위해서는 PvmDontRoute와 Pvm RouteDirect와 같은 두 가지 메시지 전송 경로를 지원하고 있다.

PvmDontRoute는 상호간 위치를 모르는 태스크간에 pvmd를 통한 투명한 원격 통신을 지원해 주는 메시지 전송 경로이다. 이를 위해 pvmd는 각 태스크의 현재 위치를 정확히 파악하고 있어야 하며, 태

스크간의 통신에 있어 모든 메시지는 pvmd를 통해서 전달된다. 통신을 원하는 지역 태스크(local task)는 위치를 모르는 원격 태스크(remote task)에게 메시지를 전송하기 위해, 먼저 동일 호스트에 상존하는 pvmd에게 수신 태스크 정보와 메시지를 함께 전송한다. 메시지를 수신한 pvmd는 이를 수신 태스크가 존재하는 시스템의 pvmd에 전송하고, 수신 시스템의 pvmd는 받은 메시지를 다시 수신 태스크에게 넘겨준다. 이 방법은 통신을 원하는 태스크 상호간에 투명성을 제공하고 이로 인한 태스크 이주(task migration)를 가능하게 하는 장점은 있으나, 모든 메시지가 반드시 pvmd를 통해서 전달되어야 하는 단점이 있다. 이는 곧 과다한 IPC로 인한 운영 체제의 부하를 가중시키고 잦은 메시지 복사로 인한 통신 지연을 초래한다.

반면, PvmRouteDirect는 PvmDontRoute와는 달리 pvmd를 경유하지 않고 태스크간에 직접적으로 TCP/IP 소켓을 이용하여 통신하는 방법이다. 따라서 초기 태스크간의 연결 설정 시에만 pvmd가 개입하고, 나머지는 pvmd의 협조 없이 태스크 상호간 독자적인 통신이 가능하다. 그러나 상호간 연결이 설정된 태스크는 연결 설정 당시의 시스템에서 다른 시스템으로 이주할 수 없으며, 계속해서 같은 시스템에 상주해야 한다. 이 경우 상대적으로 부하가 적은 새로운 유휴 시스템이 생겨도 상호 연결된 태스크들은 유휴 시스템으로의 이주를 할 수 없으며, 이는 곧 시스템 자원을 효율적으로 사용하지 못하는 결과를 초래할 수 있다. 또한, 하나의 PVM 태스크가 동시에 여러 PVM 태스크들과 메시지 교환을 원할 경우 통신하고자 하는 각 태스크마다 하나의 TCP 소켓을 필요로 한다. 이 경우 파일 테이블(file table)이나 파일 기술자(file descriptor) 등과 같은 시스템 자원 부족으로 인하여, 소켓 기술자(socket descriptor) 할당에 제약이 따를 수도 있다. 이상의 이유로 인해 PvmRouteDirect는 본 연구에서 다루지 않는다.

문제는 이들 두 OPVM 메시지 전송 경로는 반드시 커널내의 TCP/IP 프로토콜 스택을 경유한다는데 있다. 이는 사용자 영역(커널)에서 커널(사용자 영역)로의 메시지 복사 부하를 유발하게 된다. 예를 들어, 태스크가 PvmDontRoute를 이용하여 원격 태스크에게 메시지를 전송할 경우 세 번의 메시지 복사가 일어난다. 태스크로부터 커널로, 커널로부터 pvmd로 메시지를 보내는데 두 번의 복사가 발생하고, pvmd에서 다시 목적지 호스트의 pvmd로 전송하는데 또 한번의 복사가 발생한다. 후자의 경우 실제로 두 번의 메시지 복사가 일어난다: pvmd에서 커널로, 커널에서 다시 NIC로. 그러나 커널에서 NIC로의 메시지 복사는 NIC 내의 DMA 컨트롤러에 의해 CPU 실행 시간의 낭비 없이 투명하게 수행되므로 실제 메시지 복사로는 간주하지 않는다. (이 정의는 메시지 복사 수를 세는 다른 모델에도 적용된다.) 뿐만 아니라, 목적지 호스트에서 해당 메시지를 수신 태스크로 전달하는데 또한 세 번의 메시지 복사가 필요하다. 따라서 PVM이 상대적으로 통신 지연이 큰 TCP/IP에 기반을 두고 구현되는 한, PVM 내의 나머지 통신 지연 요소(slow list implementation, packing/unpacking)에서 발생하는 시간 손실은 통신 지연의 큰 제약 사항이 되지 않는다. 따라서 기반 네트워이 초고속 통신망일 경우 메시지 복사 부하는 PVM의 전체적인 성능을 감소시키는 주된 요인이 되며, 이러한 메시지 전송 경로를 수정할 경우 상당한 통신상의 성능 향상이 기대된다[11].

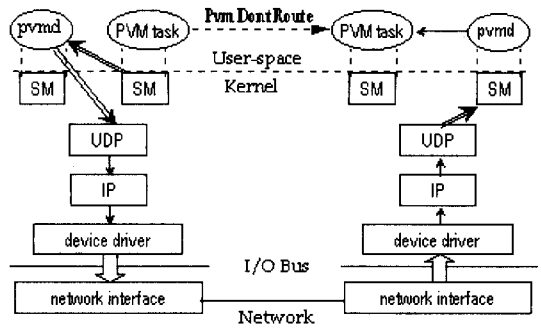


그림 2. 공유 메모리를 이용한 PVM 통신모델

그림 2는 태스크별 공유 메모리를 사용하는 SPVM의 통신 모델을 도시화한 것이다. 이 통신 모델에서, 각 호스트에 하나씩 존재하는 PVM 데몬들 사이에는 OPVM과 마찬가지로 UDP/IP socket을 사용하며, 호스트 내의 지역 태스크 상호간 또는 지역 태스크와 PVM 데몬간 통신에는 공유 메모리를 사용한다. 태스크 상호간 통신을 위해서는 PvmDontRoute 메시지 전송 경로만 지원될 뿐, PvmRouteDirect는 지원되지 않는다.

이 모델에서 각 태스크는 초기화시 자신의 공유 메모리를 일정 크기(1MB 정도)로 생성하고, 이후 다른 태스크들이 자신의 공유 메모리에 접근할 수 있게 이에 관한 정보를 pvmd에 통보한다. 각 태스

크의 공유 메모리는 전송하고자 하는 메시지를 보관하는 데이터 영역과 자신에게 전송된 메시지에 관한 정보만 담고 있는 메시지 헤드 큐 영역으로 나누어 진다. 따라서 메시지 헤드 큐는 자신에게 수신된 메시지의 간단한 정보(위치, 크기 등등)만 가지고 있을 뿐, 메시지 자체는 전송 태스크 공유 메모리 내의 데이터 영역에 존재한다.

SPVM의 PvmDontRoute는 OPVM과는 달리 수신 태스크가 동일 시스템에 존재하느냐 아니면 원격 시스템에 존재하느냐에 따라 메시지의 이동 경로가 달라진다. 동일 호스트 내에 존재할 경우, pvmd를 통하지 않고 수신 태스크에게 직접 메시지가 전달된다. 즉, 송신 태스크는 자신의 공유 메모리 내의 데이터 영역에 해당 메시지를 준비(packing) 시키고, 이 메시지를 기술하는 헤드 정보를 수신 태스크 공유 메모리(pvmd로부터 사전에 이 위치를 파악 해둠) 내의 메시지 헤드 큐에 삽입한다. 수신 태스크는 이 헤드 정보를 이용해 해당 메시지의 위치를 찾고, 전송 태스크 공유 메모리 내의 데이터 영역으로부터 자신의 지역 메모리로 해당 메시지를 가져(unpacking) 온다. 이 경우 메시지 복사는 일어나지 않는다.

그러나 수신태스크가 원격 호스트에 존재할 경우, 메시지는 pvmd를 통해야 되며 이 과정에서 두 번의 메시지 복사가 필요하게 된다. 즉, 송신 태스크로부터 pvmd로의 메시지 전송은 앞서 언급한 바와 같이 공유 메모리를 이용하므로 메시지 복사는 일어나지 않는다.

그러나 pvmd에서 수신시스템으로의 전송은 UDP/IP를 이용하며, 이를 위해 pvmd는 해당 메시지를 송신 태스크의 공유 메모리에서 자신의 지역 메모리로 한번 복사하고, 이를 다시 시스템 함수를 이용하여 지역 메모리에서 커널로 보내는 과정에서 또 한번의 복사가 일어난다. 이 같은 pvmd 지역 메모리로의 비효율적인 복사는 메시지의 분실 방지를 위해 반드시 필요하다. 예를 들어, 태스크가 메시지를 보낸 후 작업을 끝냈을 경우(아직 이 메시지는 pvmd에 의해 전송이 되지 않은 상태), 그 태스크의 공유 메모리도 함께 삭제되며, 이 경우 삭제된 공유 메모리의 데이터 영역에 남아 있던 전송 메시지는 분실하게 된다.

이 연구에서는 유사한 다른 연구에서와 같이 커널로부터 NIC로 메시지를 옮기는 작업과 메시지 packing 및 unpacking 작업 등은 실제 메시지 복사로 간주하지 않는다.

Ⅳ. Zero-copy PVM 통신 모델

태스크별 공유 메모리를 이용하는 SPVM은 OPVM에 비해 메시지 복사 횟수가 세 번에서 두 번으로 줄었지만, 여전히 두 번의 메시지 복사로 인한 통신상의 지연을 초래한다. 따라서 이러한 문제를 해결하기 위해 이 논문에서는 초고속 통신망을 기반으로 하고, 전역 공유 메모리(global shared memory)를 이용한 zero-copy PVM 통신 모델을 제안한다.

이 모델은 PVM 태스크, PVM 데몬, 그리고 NIC에 의해 동시 접근이 가능한 전역 공유 메모리를 필요로 한다. 그림 3은 이를 도시화한 것이다.

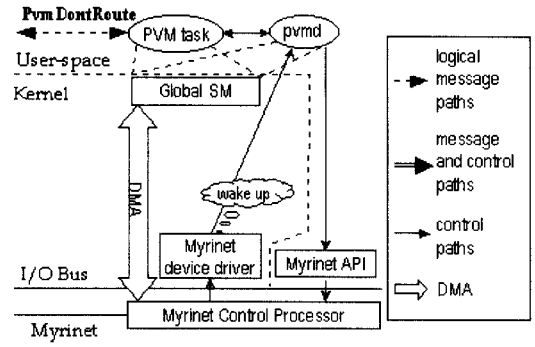


그림 3. 전역공유 메모리와 zero-copy 기술을 이용한 PVM 통신모델

그림에서 PVM 태스크와 PVM 데몬은 일반 응용 프로세스이므로 자신의 가상 기억 주소(virtual address)를 이용하여 전역 공유 메모리에 접근하고, NIC는 DMA 접근을 위해 실 기억 주소(real address)를 필요로 한다. 제안한 모델이 기반으로 하는 초고속 통신망(Myrinet)은 일반적으로 커널을 경유하지 않고 사용자 수준에서 직접 NIC로 접근 가능케 하는 자체 NIC API(Myrinet API)를 제공하고 있다. 따라서 pvmd가 NIC API이용하여 NIC에 직접 접근하기 위해서는 전송 데이터의 실 기억 주소를 매개 변수로 넘겨 주어야 하며, 이를 위해 전역 공유 메모리의 가상 기억 주소를 실 기억 주소로 변환하기 위한 변환 테이블을 관리해야 한다. 전역 공유 메모리는 고정된 크기를 가지고 재배치(relocation) 없이 항상 주기억 장치에 상주(pin-down or memory-lock) 해야 하며, 이는 복잡한 알고리즘 없이 쉽고 빠르게 NIC가 접근 가능하도록

하기 위해서 이다.

메시지를 전송하고자 하는 PVM 태스크는 해당 메시지를 전역 공유 메모리에 packing 시키고, 이를 pvmd에게 통보한다. pvmd는 해당 메시지의 실 기억 주소를 구한 후, 이를 NIC API을 이용해 커널을 통하지 않고 NIC에 직접 넘겨 주면서 전송을 의뢰한다. 이후 NIC내의 DMA 제어장치는 해당 메시지를 전역 공유 메모리로부터 통신망으로 전송한다. 이 과정에서 커널의 도움을 전혀 받지 않으며, 메시지 복사 또한 한번도 일어나지 않는다.

메시지가 도착된 경우, NIC는 DMA 제어 장치를 이용하여 특정 주소(수신용 큐에 사전에 정의된 전역 공유 메모리내의 주소)로 해당 메시지를 전달한 후, 인터럽트를 발생하여 커널내의 NIC 드라이브를 구동한다. 여기서 해당 메시지를 기다리는 프로세스(pvmd)를 깨워 준다. pvmd는 다시 수신 태스크를 찾아 이를 통보하며, 최종적으로 수신 태스크가 전역 공유 메모리내의 해당 메시지를 unpacking하고 처리하게 된다. 이 과정에서 pvmd(수신 프로세스)를 깨우기 위해 커널의 협조를 받지만, 통신 프로토콜 스택을 거치는 것이 아니므로 메시지 복사는 발생하지 않는다.

일반적으로 각 태스크가 공유 메모리를 가지는 데는 두 가지 방법이 있으며, 하나는 SPVM에서 사용하는 방식으로 각 태스크마다 개별적인 공유 메모리를 생성 및 소유하게 하고 이를 다른 태스크가 사용할 수 있게 하는 방법이고, 두 번째 방법은 제안된 모델에서 사용하는 방식으로 하나의 커다란 메모리 영역을 모든 태스크로 하여금 동시에 사용 가능하게 하는 방법이다.

전자는 제어 및 구현은 간단하나, 각 태스크에 개별적으로 할당된 공유 메모리만 사용해야 하는 제약이 따른다. 또한 SPVM처럼 태스크가 메시지를 보낸 직후 작업을 끝내고 공유 메모리를 삭제했을 경우 발생하는 메시지 분실을 방지하기 위해 pvmd의 지역 메모리로의 불필요한 메시지 복사를 해야 하는 부하가 발생한다. 이러한 문제로 인해 이 연구에서는 후자의 방법을 선택했으며, 전역 공유 메모리의 전체 크기에는 제한을 받으나 개별 태스크에 배분된 메모리의 크기나 태스크의 수에는 제한 받지 않는 이점이 있다. 그리고 한번 전송된 메시지는 전송 태스크의 소멸 이후에도 전역 공유 메모리에 계속 보관되었다가 pvmd에 의해 전송되므로 전자와 같은 pvmd 지역 메모리로의 불필요한 메모리 복사를 방지할 수 있다.

전역 공유 메모리 모델에서 전송 태스크는 메시지를 packing 하기 전에 전역 공유 메모리로부터 필요한 데이터 버퍼를 사전에 할당 받아야 하며, 수신 태스크는 해당 메시지를 unpacking한 후, 버퍼를 반납해야 한다. 그러나 전역 공유 메모리의 크기는 제한되어 있으므로 여러 태스크에서 요구되는 버퍼의 수가 전역 공유 메모리내의 이용 가능한 프리 버퍼(free buffers)의 수보다 커질 경우 이를 제어할 수 있는 동기화가 필요하다. 새로이 설계된 동기화 메카니즘은 요구한 버퍼를 할당하지 못할 경우 요구 태스크를 대기하게 하고, 메시지 수신이 끝난 태스크(또는 커널)가 버퍼를 반납할 때, 대기중인 태스크를 깨워 주는 방식을 택하고 있다. 그러나 하나 이상의 태스크들이 버퍼 할당을 기다리고 있을 때, 어떤 태스크에 우선권을 주어 먼저 깨울 것인가를 판단하기 위한 태스크 스케줄링 알고리즘이 필요하다. 본 연구에서는 간단한 first-fit 알고리즘을 사용한다. 이 알고리즘은 버퍼 할당을 기다리는 태스크 대기 큐에서 현재 이용 가능한 프리 버퍼의 수보다 같거나 적은 수의 버퍼를 요구한 태스크들 중 첫번째 태스크에게 버퍼를 할당한다. 보다 좋은 성능을 위해서는 best-fit 알고리즘과 같은 다른 알고리즘들도 고려해 볼 수 있다.

V. 구 현

이 논문에서 제안된 초고속 통신망을 기반으로 한 zero-copy PVM 통신 모델은 PVM 3.3[1], Linux 2.0.30[12] 운영 체제, Myrinet MCP(Myrinet Control Program) 4.0과 Myrinet 드라이버 2.0[13]을 기반으로 하여 구현되었다. 기반 초고속 통신망은 Myricom사에 의해 공급되는 Myrinet[14]을 사용하였다. 이후 zero-copy PVM 통신 모델을 적용하여 구현된 PVM을 EPVM(Enhanced PVM)이라 지칭한다.

이를 위해 먼저 태스크별 공유 메모리를 기반으로 하는 SPVM을 Linux 시스템으로 이식하였다.(현재까지 Linux 시스템으로 이식된 SPVM 버전이 보고된 적은 없음). 그리고 전역 공유 메모리를 사전 확보하기 위한 알고리즘을 구현하고, 이를 Linux 부팅 시 호출하게 하였다. 또한 Linux에 이식된 SPVM을 수정하여, 기존 불필요한 요소들을 삭제하고, 전역 공유 메모리 관리, 대기 태스크 스케줄링 알고리즘, 태스크 동기화 메커니즘, Mynet API를 이용한 메시지 송수신 관리, 슬라이딩 윈도우 프로

토콜 등의 기능을 새로이 추가하였다. 뿐만 아니라, 기존 select() 함수를 이용한 블록킹 I/O와의 호환성을 유지하기 위해 Myrinet MCP와 Myrinet 드라이브를 수정하였다. 마지막으로 통신 지연을 최소화하기 위해 Linux 스케줄러를 수정하여 pvmd 상호간 Co-scheduling이 가능하게 하였다.
다음의 세부 절에 위의 내용을 상세히 논의한다.

5.1 전역 공유 메모리의 생성

이 연구에서는 전역 공유 메모리를 위한 실 기억 장치 공간을 확보하고 이를 관리하기 위해, 기존 Linux 시스템에 bigphysarea_setup(), bigphysarea_init(), bigphysarea_alloc() 등과 같은 함수를 새로이 추가하였다. 연속된 물리적 메모리 공간은 Linux 시스템 부팅 시 bigphysarea_setup(number_of_pages)과 bigphysarea_init()의 두 함수에 의해 확보되어 진다. 이는 전역 공유 메모리를 위한 연속된 물리적 메모리 공간을 사전에 확보하고 memory-lock(pin-down)을 해 둠으로써, 이후 Linux의 가상 기억 장치 관리 시스템에 의해 이 공간이 다른 태스크로 할당되거나 디스크로 swap-out 되지 않도록 한다. 부팅이 완료되면, Myrinet 드라이브가 설치되며, 이 드라이버는 bigphysarea_alloc(size)을 호출하여 부팅시 확보된 물리적 메모리 공간을 모두 할당받아 이후 EPVM으로부터의 서비스 요청에 대비한다. 이러한 할당 방식은 각 태스크별 별도의 고정된 물리적 메모리 공간을 할당하는 Unet[3]과는 약간의 차이가 있다.

이렇게 확보된 연속된 물리적 메모리 공간은 전역 공유 메모리로 활용되기 위해 이후 PVM 데몬과 PVM 태스크들의 가상 기억 공간으로 매핑(mapping)되어 진다. 즉, PVM 데몬과 PVM 태스크들은 각 초기화 단계에서 myri_init(file_name)을 호출하며, 이는 file_name의 Myrinet 디바이스 파일을 열고 시스템 함수인 ioclt()을 호출하여, Myrinet 디바이스 드라이브에 의해 확보된 물리적 메모리 공간의 크기 및 제어 정보를 얻는다.

이후 시스템 함수인 mmap()을 호출하여 연속된 물리적 메모리 공간을 PVM 데몬과 태스크의 연속된 가상 기억 공간으로 매핑한다. 이 과정에서 mmap()은 Myrinet 디바이스 드라이브의 해당 서비스 함수를 호출하며, 이는 또한 가상 기억 장치 관리 시스템의 커널 함수인 remap_page_range()를 호출하여 PVM 데몬과 태스크의 가상 기억 공간 중 물리적 메모리 공간과 동일한 크기의 연속된 빈 영역을 찾아 매핑하게 된다. PVM 데몬은 또한 ioctl(myri_fd, MLANAI_GET_MEN_INFO, unit)을 사용하여 전역 공유 메모리에 매핑된 연속된 물리적 메모리의 시작 주소(실기억 주소)를 얻는다. 이 시작 주소는 Myrinet API을 이용해 메시지를 보내거나 받을 때 가상 기억 주소를 실기억 주소(또는 역으로)로 변환할 때 사용된다. Myrinet 라이브러리 함수인 myri_init()과 Myrinet 디바이스 드라이브 함수인 ioctl()은 EPVM을 위해 적절히 수정되었다.

5.2 전역 공유 메모리의 관리

EPVM에 의해 생성된 전역 공유 메모리는 정보 영역, 큐(queue) 영역, 버퍼 영역 등 세 개의 영역으로 나누어진다.

정보 영역은 현재 사용되지 않는 프리 페이지(free page)들의 목록과 이들을 관리하기 위한 동기화 정보를 보관한다. 큐 영역은 각 태스크당 하나씩 할당되며 다른 태스크로부터 수신된 메시지들의 헤더에 관한 정보를 가지고 있을 뿐 아니라, 메시지 큐로서의 기능도 겸하고 있다. 마지막으로 버퍼영역은 메시지 자체를 저장하기 위한 버퍼들로 구성된다. 버퍼 영역은 8KB의 페이지 단위로 관리되며 페이지의 크기는 Myrinet MTU(maximum transmission unit) 크기와 동일하다. 따라서 하나의 버퍼는 곧 하나의 페이지를 의미한다.

각 태스크는 메시지를 전송하기 위해 우선 새로이 추가된 pvm_getbuf() 함수를 이용하여 충분한 메시지 버퍼를 할당 받아야 한다. 이후 할당된 버퍼 영역에 메시지를 packing 하여 전송한다. 메시지를 수신한 태스크는 이를 unpacking 하여 사용하며, unpacking이 끝난 메시지 버퍼를 역시 새로이 추가된 pvm_freebuf()를 이용하여 반납한다. 실제 pvm_getbuf()와 pvm_freebuf()는 각각 pvm_malloc()과 pvm_mfree()에 의해 호출되며, 이들은 다시 각각 PVM 라이브러리 함수인 pvm_send()와 pvm_recv()에 의해 호출되므로, 사용자 태스크들에게는 투명(transparent)하게 보인다.

전역 공유 메모리의 크기가 고정되어 있는 관계로 사용 가능한 버퍼의 수도 제한되어 있다. 따라서 메시지 버퍼를 요구한 태스크는 공유 메모리내의 버퍼영역에 사용 가능한 프리 버퍼가 없을 경우 pvm_getbuf() 함수 내에서 대기(blocked)해야 한다. 예를 들어, 현재 전역 공유 메모리의 프리 페이지의 수가 f이고 태스크가 요구하는 페이지의 수가 r일 때, r이 f보다 크다면 태스크는 f가 r보다 크거나 같

을 때까지 태스크 대기 큐에서 대기해야 한다. 이후 pvm_freebuf()를 통하여 다른 태스크에 의해 사용이 끝난 버퍼들이 반납되고, 이때 태스크 대기 큐에 하나 이상의 태스크가 기다릴 때, 어떤 태스크에 우선권을 주어 먼저 깨울 것인가를 결정하여야 한다. 이때 다양한 방법을 사용할 수 있으나, 본 연구에서는 구현하기 간단하면서도 성능이 뛰어난 first-fit 알고리즘을 적용하였으며, 태스크 동기화 메카니즘으로는 mutex(or semaphore)를 사용하였다.

First-fit 알고리즘은 버퍼 할당을 기다리는 태스크 대기 큐에서 현재 이용 가능한 프리 버퍼의 수보다 같거나 적은 수의 버퍼를 요구한 태스크들 중 첫번째 태스크에게 버퍼를 할당하는 전략이다. 즉, 버퍼를 요구한 모든 태스크를 요구 시간에 따라 순서적으로 관리하면서, 버퍼들이 반납될 때마다 현재 사용 가능한 프리 버퍼 수가 대기 큐의 각 태스크가 요구한 버퍼 수 보다 크거나 같은 것이 있는지 하나씩 비교한다. 요구된 버퍼 수를 만족할 수 있는 첫 번째 태스크에게 우선적으로 버퍼를 할당하고 해당 태스크를 깨워 준다. 이후 계속해서 깨어날 수 있는 태스크가 없거나 프리 버퍼가 없을 때까지 이 과정은 반복된다.

예를 들어, 그림 4에서 태스크 대기 큐에는 7개의 태스크가 버퍼를 요구한 순서에 따라 대기하고 있고, 현재 5개의 이용 가능한 프리 페이지가 존재한다. 이 경우 요구를 만족할 수 있는 태스크는 T3와 T5이고 이중 T3가 먼저 요구를 하였으므로, T3에게 3개의 버퍼가 할당된다. 그러나 이 경우 best-fit 알고리즘이 적용되었다면 T3 대신 T5 에게 5개의 버퍼가 할당된다.

Waiting Queue

Demand Pages	T_1	T_2	T_3	T_4	T_5	T_6	T_7	
Blocked Tasks	7	9	3	10	5	15	6	

Number of free pages = 5

Number of minimum demand pages = 3

그림 4. first-fit 알고리즘에 의한 버퍼 할당 예

5.3 메시지의 송수신

Myrinet API를 이용한 통신은 PVM 데몬에서 커널을 통하지 않고 직접 Myrinet 인터페이스 보드에 접근할 수 있다. 그러나 사용자 수준에서의 Myrinet API는 기본적으로 인터럽트를 이용한 event driven I/O (interrupted I/O) 방식을 지원하지 않고, 프로그램에서 직접 폴링(polling)해야 하는 programmed I/O 방식을 지원한다 (단, 커널내의 TCP/IP용 Myrinet 드라이버는 interrupt I/O 방식 사용). 즉, select() 함수와 같은 대기(blocked) 상태에서의 메시지 수신 여부를 확인하는 블록킹 I/O 방식을 제공하지 않는다.

따라서 이 연구에서는 Myrinet 디바이스 파일 기술자(file descriptor)와 select() 함수를 이용한 블록킹 I/O 방식을 지원할 수 있게 Myrinet MCP와 드라이브를 수정하였다. 사용자 수준의 Myrinet API가 이용하는 일반 통신 채널(1 or 2)로 메시지가 수신되어도 인터럽트가 발생되게 Myrinet MCP를 수정하였으며 (기존에는 TCP/IP용으로 사용하는 커널 통신 채널(0)로만 인터럽트를 발생시켰음), 일반 통신 채널로 메시지가 도착했을 경우 이를 수신할 프로세스(PVM 데몬)를 깨우도록 Myrinet 드라이브를 수정하였다. 그러나 이 경우 Myrinet 드라이브는 메시지를 처리할 PVM 데몬을 깨우는 역할만 할 뿐 메시지 자체의 처리에는 관여하지 않는다.

일반적으로 하나의 메시지는 packing 과정에서 여러 개의 패킷으로 나누어져 전송된다. pvm_send()에 의해 호출된 peer_send()는 각각의 패킷에 대한 헤드 정보를 pvmd나 수신 태스크의 메시지 헤드 큐에 삽입하고, 해당 수신 태스크를 깨어준다. 그러나 이것은 최악의 경우 매 패킷마다 두 번의 문맥 교환(context switching)이 발생할 수 있다. 따라서 이러한 과다한 문맥 교환 부하를 피하기 위해 peer_send()를 메시지 단위로 수신 태스크를 깨워주는 peer_send_message()로 교체하였다.

5.4 요구기반 동적 Co-스케줄러와 슬라이딩 윈도우 프로토콜

EPVM에서 통신망으로의 실질적인 메시지의 송수신을 담당하는 것은 PVM 데몬이고, 메시지의 길이가 통신망의 MTU보다 클 경우 이를 여러 개의 패킷으로 분리하여 전송한다. 따라서 PVM 데몬은 신뢰성 있는 통신을 위해 슬라이딩 윈도우(sliding window) 프로토콜을 사용한다. 윈도우 크기가 1일 경우 수신 데몬은 매 패킷을 받을 때마다 이를 받았다는 응답(acknowlegement) 패킷을 송신 데몬에게 보내지만, 윈도우 사이즈가 n일 경우 n개의 연속된 패킷을 받았을 때 한번의 응답 패킷을 보낸다. 그러나 기존의 PVM 데몬은 윈도우 크기가 1인 슬라이딩 윈도우 프로토콜만 사용하며, 이는 통신 지연의 한 요인으로 작용할 수 있다.

따라서 본 연구에서는 임의의 윈도우 크기를 사용할 수 있게 PVM 데몬의 슬라이딩 윈도우 프로토콜을 수정하였다. 그러나 슬라이딩 윈도우 프로토콜을 사용할 경우, 빠른 메시지의 전송을 위해선 각 패킷에 대한 재빠른 응답이 필수적이다. 만약 패킷 도착 직후 이에 응답해야 할 PVM 데몬이 이미 수행 중인 다른 태스크의 영향으로 즉시 수행되지 못하고 지연된다면, 이는 곧 도착된 패킷의 응답 지연은 물론 통신 지연의 직접적인 요인이 된다. 뿐만 아니라, 또 다른 호스트로부터 패킷이 계속해서 도착할 경우, NIC의 입력용 큐의 과부하(overflow)로 인한 패킷들의 분실을 초래할 수 있다. 슬라이딩 윈도우 프로토콜의 윈도우 크기가 증가할수록 이 문제는 더욱 심각해 진다. 따라서 본 연구에서는 이러한 패킷 수신 데몬의 스케줄링 지연시간을 줄이기 위해 기존 Linux 스케줄러를 수정 보완하여 요구기반 동적 co-스케줄러(demand-based dynamic co-scheduler)[15]를 구현하였다. co-스케줄러는 패킷이 도착할 때마다 PVM 데몬에게 가장 높은 우선 순위를 부여하여, 데몬으로 하여금 패킷 도착 즉시 응답 또는 다음 데이터 패킷을 전송할 수 있게 한다. 이러한 co-스케줄링 효과는 송수신 시스템의 두 PVM 데몬을 거의 동시에 실행하게 하여 블록킹 시간과 통신 지연을 크게 감소시킬 수 있다. 이 전략은 또한 슬라이딩 윈도우 프로토콜에서 윈도우 크기를 증가 시킬 수 있는 계기를 제공한다.

Ⅵ. 성능 평가

이 장에서 종합적인 실험 결과를 논의하고자 한다. 메시지 교환에 따른 통신상의 지연을 객관적으로 측정하기 위해 [1]에서 제시된 벤치마크 프로그램을 사용하기로 했다. 이 프로그램은 송신 태스크에서 일정 크기의 메시지를 전송한 시간부터 수신 태스크로부터 이 메시지를 되돌려 받을 때까지 소요된 왕복 시간(round-trip time: RTT)을 측정한다.

성능 비교를 위해 이 벤치마크 프로그램을 앞서 논의한 PVM의 세가지 다른 버전에서도 실행하였다. 첫째는 기존의 socket API(커널내의 TCP/IP 프로토콜을 이용함)를 기반으로 하는 OPVM 버전이고, 둘째는 태스크별 공유 메모리(원격 태스크와의 통신을 위해 여전히 커널내의 TCP/IP 프로토콜을 이용함)를 사용하는 SPVM이며, 나머지 하나는 이 논문에서 제안한 zero-copy 통신 모델을 적용한 전역 공유 메모리를 사용하는 EPVM이다. OPVM은 원래의 PVM 버전을 그대로 사용하였고, SPVM은 실험을 위해 Linux 시스템용으로 이식하였으며, 이식된 SPVM을 기반으로 하여 새로운 EPVM을 구현하였다.

6.1절에서는 성능 평가가 이루어진 실험 환경에 대해 설명하고, 6.2절에서는 실험 결과인 세가지 PVM 버전에서의 지역 태스크간 또는 원격 태스크간의 메시지 왕복 시간을 보여주고 이를 분석한다. 메시지 왕복 시간은 메시지의 크기 뿐 아니라 PVM 데몬이나 운영 체제에 의해 사용되는 여러 알고리즘에 의해 영향을 받는다. 따라서 나머지 절에서는 태스크 동기화 부하 및 스케줄링 알고리즘과 같은 여러 민감한 사항에 대해 논의하고, 이들이 메시지 왕복 시간에 미치는 영향을 실험을 통해 알아본다.

6.1 테스트 환경

실험에는 64MB 메인 메모리와 Pentium 프로세서를 가진 PCI 버스상에 Linux 2.0.30 운영 체제를 장착한 4대의 개인용 컴퓨터를 사용하였다. 기반 초고속 통신망으로 Myricom사에 의해 공급되는 Myrinet[14]을 사용하였으며, 이는 640Mbps의 전송 속도를 제공한다. 각 시스템은 256KB SRAM을 가진 M2F-PC132 Myrinet 인터페이스 카드를 장착하였으며, 이들 시스템 상호간은 M2F-SW8 Myrinet 스위치를 통하여 연결되었다.

앞서 언급했듯이 메시지의 왕복 시간은 Linux의 프로세스 스케줄링 알고리즘과 PVM 데몬에 의해 사용되는 통신 프로토콜에 상당한 영향을 받는다. SPVM과 OPVM의 pvmd는 기본적으로 윈도우 크기가 1인 슬라이딩 윈도우 프로토콜을 사용한다. 그러나 이 논문에서 구현한 EPVM은 다양한 윈도우 크기를 가진 슬라이딩 윈도우 프로토콜을 지원하며, Linux 고유 스케줄러 대신 본 연구에서 개발한 co-스케줄러를 사용한다. 따라서 향후 각 실험 결과에서 EPVM(α ,β)라고 표기함은 α 스케줄링 알고리즘과 β 윈도우 크기를 가진 슬라이딩 윈도우 프로토콜을 사용하는 EPVM상에서 실험이 진행되었다는 것을 의미한다. 예를 들어, EPVM(O,1)은 Linux 고유 스케줄러를 수정 없이 그대로 사용했고, 슬라이딩 윈도우 프로토콜의 윈도우의 크기는 1과 같음을 의미한다. 또한 EPVM(C,5)는 요구기반 동적 co-스케줄러가 사용되었고, 윈도우 크기는 5와 같음을 의미한다.

각 실험에서 EPVM은 항상 4MB의 고정된 크기를 가진 전역 공유 메모리를 사용하였다.

6.2 메시지 왕복 시간(round-trip times: RTT)

메시지 왕복 시간 측정은 두 가지 방향으로 진행되었다. 하나는 지역 태스크 사이의 측정이고 다른 하나는 원격 태스크 사이의 측정이다. 각 실험은 EPVM, OPVM, SPVM 하에서 실시되었으며, 메시지를 보내는 송신 태스크와 이를 수신하여 되돌려 보내는 수신 태스크만 실행되었다. 메시지 왕복 시간은 시스템 호출 함수인 gettimeofday()를 사용하여 측정된다. 송신 태스크는 동일 조건의 1000개의 메시지를 반복하여 전송한 후 이들의 왕복 시간을 측정하고 평균값을 계산한다.

표 1의 왼쪽 부분은 지역 태스크 사이의 평균 메시지 왕복 시간을 보여 주고 있다. [16]의 연구에서 이미 밝혀졌 듯이 SPVM은 지역 태스크 사이의 메시지 교환에는 zero-copy scheme을 사용한다. 마찬가지로 EPVM 역시 zero-copy 통신 모델을 사용하므로, EPVM(O,1)과 SPVM의 수행 결과가 유사한 것은 명확하다. EPVM(O,1)과 SPVM 둘 다 메시지의 크기에 상관없이 OPVM의 성능을 능가하고 있다. 그 이유는 EPVM과 SPVM 상에서는 메시지 복사나 pvmd의 간섭 없이 메시지 송수신이 이루어지나, OPVM의 경우 메시지는 커널과 pvmd를 통과해야 하며 이로 인하 메시지 복사 부하를 감수해야 하기 때문이다. EPVM(O,1)(또는 SPVM)과 OPVM과의 왕복 시간 차이는 메시지의 크기가 커질수록 더 커지는데, 이것은 메시지의 크기가 커질수록 비례하여 메시지 복사 부하가 늘어나기 때문이다.

표 1의 오른쪽 부분은 원격 태스크 사이의 평균 메시지 왕복 시간을 보여주고 있다. 표에서 EPVM(O,1)은 SPVM이나 OPVM 보다 뛰어난 성능을 발휘하고, SPVM 역시 OPVM 보다 더 좋은 결과를 보여 주고 있다. 그러나 EPVM(O,1)과 SPVM 사이의 RTT 차이가 SPVM과 OPVM 사이의 RTT 차이 보다 훨씬 더 크다는 것을 확인할 수 있다. 이것은 EPVM은 무 복사(zero-copy)이고 SPVM은 여분의 메시지 복사 동작을 더 수행하기 때문이며, 이 차이는 메시지 크기가 커질수록 더 커지게 된다. EPVM(O,1)의 왕복 시간은 메시지의 크기가 4MB일 때 OPVM과 비교하여 51% 감소한다. 이 결과로부터 메시지 복사 동작이 RTT에 상당한 영향을 준다는 것을 알 수 있다. 더 나아가 SPVM이나 OPVM에서 발생하는 메시지 복사 부하 외에도 PVM 데몬과 커널이 간여함으로써 발생하는 부하 또한 무시할 수 없음을 확인할 수 있다.

표 1. 태스크 사이의 메시지 크기별 왕복 시간 (단위: micro sec)

message size	RTT between local tasks			RTT between remote tasks				
	EPVM (O,1)	SPVM	OPVM	EPVM (O,1)	SPVM	OPVM	EPVM (O,1) /OPVM	Direct route
16B	94	97	521	670	799	1,012	0.662	516
32B	94	96	507	669	799	1,009	0.663	517
64B	95	97	507	702	784	1,010	0.695	519
128B	94	96	510	675	736	1,029	0.655	521
256B	94	98	510	673	744	1,026	0.655	527
512B	95	96	520	682	758	1,046	0.652	539
1KB	99	97	525	741	907	1,077	0.688	565
2KB	96	97	542	766	961	1,129	0.797	626
4KB	97	97	584	846	1,079	1,292	0.784	722
8KB	105	104	803	1,300	1,760	1,944	0.739	1,691
16KB	110	113	1,160	1,885	2,619	2,841	0.720	2,485
32KB	123	124	2,074	3,026	4,363	4,636	0.694	3,117
64KB	151	156	4,395	5,205	8,053	9,043	0.646	6,350
128KB	218	243	9,263	9,514	15,691	18,790	0.606	13,444
257KB	519	501	18,382	18,102	32,156	37,887	0.563	27,183
512KB	893	842	34,140	34,445	64,561	74,679	0.534	55,677
1MB	1,300	1,278	67,690	65,972	127,211	146,739	0.519	107,827
2MB	2,080	2,089	130,037	129,811	253,592	290,488	0.512	221,475
4MB	3,703	3,717	236,567	257,378	505,311	571,513	0.510	398,830

6.3 태스크 동기화 부하

전역 공유 메모리의 크기는 고정되어 있고 과다한 통신량으로 인해 프리 버퍼가 모자랄 경우, 태스크들은 이를 위해 서로 경쟁하게 되며 이는 곧 RTT가 증가되는 한 요인으로 작용할 수 있다. 따라서 이들 태스크간의 동기화에 따른 부하와 이로 인한 통신 지연을 측정하기 위해 태스크 수와 메시지 크기를 다양하게 변경하면서 실험 해 보았다.

그림 5는 다양한 태스크의 수(N)와 전체 메시지 크기(T)의 변화에 따른 평균 왕복 시간을 보여주고 있다. 여기서 N은 동일 전송 시스템에서 똑 같은 크기의 메시지를 같은 시간대에 전송하는 전송 태

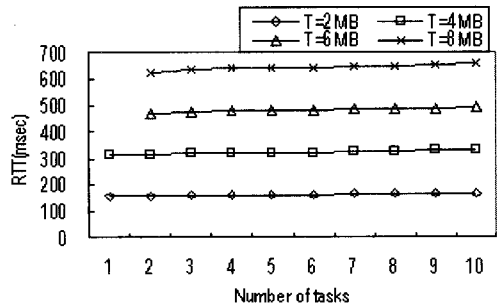


그림 5. EPVM(O,1)상에서의 태스크 수의 변화에 따른 메시지 왕복 시간 : 모든 태스크가 전송하는 전체 메시지의 크기는 고정

스크들의 수를 의미하며, T는 송신 태스크들에 의해 같은 시간대에 전송되는 메시지의 총합을 의미한다. 이 실험에서 각 태스크에 의해 전송되는 메시지의 크기는 T / N 으로 항상 일정하다. T가 전역 공유 메모리의 크기(4MB) 보다 작거나 같을 때는 태스크 동기화가 일어나지 않는다. 그러나 T가 4MB 보다 크면 태스크들은 버퍼를 할당 받기 위해 경쟁하고, 어떤 태스크는 대기(blocked)해야 한다. 이 대기 시간은 결국 메시지 왕복 시간에 추가된다.

표 2. EPVM(O,1), SPVM, OPVM상에서의 태스크 수의 변화에 따른 메시지 왕복 시간 : 모든 태스크가 전송하는 전체 메시지 크기는 고정 (단위: mili sec)

No. of tasks	EPVM(O,1)				EPVM(O,1)				EPVM(O,1)			
	2MB	4MB	6MB	8MB	2MB	4MB	6MB	8MB	2MB	4MB	6MB	8MB
1	157	313	-	-	279	566	-	-	334	634	-	-
2	158	315	472	628	279	564	840	1118	328	655	986	1248
3	160	320	485	636	278	559	839	1127	328	656	981	1317
4	162	321	481	641	280	560	844	1124	329	653	988	1324
5	162	323	481	641	279	560	843	1128	332	662	989	1322
6	162	323	481	642	279	563	845	1132	333	664	993	1321
7	164	325	485	648	280	566	853	1135	334	664	1001	1331
8	166	329	487	649	283	565	854	1137	341	667	994	1328
9	166	331	490	656	282	569	854	1140	341	673	1012	1334
10	167	334	496	657	282	567	861	1140	341	684	1005	1331

그림 6은 태스크의 수(N)와 각 태스크에 의해 전송되는 메시지의 크기(S)를 변경하며 수행한 실험 결과를 보여 준다. 이 경우 전체 메시지 크기는 N * S 이다. 위의 두 실험 모두 태스크의 수가 증가함에 따라 이로 인한 태스크 동기화 부하도 증가함을 보여주고 있다.

표 2와 표 3은 각각 그림 5와 그림 6에서 수행된 실험과 동일한 조건하에서 EPVM, SPVM, OPVM에서 소요되는 왕복 시간을 측정하기 위해 시행된 실험 결과이다.

표 3. EPVM(O,1), SPVM, OPVM상에서의 태스크 수의 변화에 따른 메시지 왕복 시간 : 각 태스크가 전송하는 메시지의 크기는 고정 (단위: mili sec)

No. of tasks	EPVM(O,1)				EPVM(O,1)				EPVM(O,1)			
	5MB	1MB	2MB	4MB	5MB	1MB	2MB	4MB	5MB	1MB	2MB	4MB
1	40	79	157	314	69	139	280	561	84	168	332	634
2	80	158	316	627	139	280	560	1118	165	329	658	1252
3	120	238	474	953	209	420	840	1708	248	493	985	1888
4	162	321	641	1282	280	560	1123	2262	330	661	1315	2504
5	202	400	804	1608	351	703	1409	2830	415	850	1654	3130
6	243	483	966	1930	420	845	1701	3410	499	995	1974	5129
7	285	565	1128	2259	495	990	1993	3987	583	1160	2308	11779
8	328	659	1301	2596	565	1135	2283	4574	671	1325	2653	-
9	368	738	1471	2939	638	1283	2582	5154	754	1495	2985	-
10	412	824	1644	3285	712	1434	2869	5775	842	1669	3333	-

표 2의 왕복 시간은 전체 메시지 크기와 태스크의 수를 변경하면서 측정한 결과이며, 표 3은 각 태스크에 의해 전송되는 메시지의 크기와 태스크의 수를 변경하며 측정한 결과이다. 어떤 태스크는 전역 공유 메모리의 빈 버퍼를 할당 받기 위해 대기해야 하고 이로 인한 동기화 부하에 따른 왕복 시간이 증가할지라도, EPVM(O,1)의 왕복 시간은 여전히 SPVM이나 OPVM의 왕복 시간 보다는 작다.

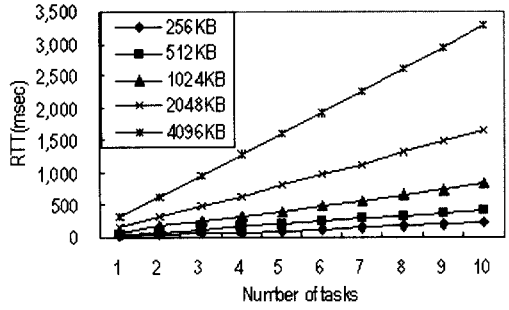


그림 6. EPVM(O,1)상에서의 태스크 수의 변화에 따른 메시지 왕복 시간 : 각 태스크가 전송하는 메시지의 크기는 고정

이는 태스크 동기화에 의한 부하보다는 메시지 크기가 크면서 발생하는 복사 부하가 더 크기 때문이다. 표 3에서 OPVM은 메시지의 크기가 4MB이고 7개 이상의 태스크가 동시에 메시지를 전송하고자 할 경우, 시스템 다운현상으로 인해 더 이상 실험을 재개할 수 없었다. 이러한 시스템 다운현상은 전송해야 할 메시지의 과부하로 인해 커널이 TCP/IP 버퍼로 사용할 실 기억 장치 페이지를 더 이상 할당 받지 못해 발생하는 오류였다.

6.4 요구기반 동적 Co-스케줄링 효과와 슬라이딩 윈도우 프로토콜의 영향

본 연구에서는 실질적인 메시지 패킷을 수신하는 PVM 데몬의 스케줄링 지연시간을 줄이기 위해 기존 Linux 스케줄러를 수정하여 co-스케줄링이 가능하게 하였다. 그림 7은 요구기반 동적 co-스케줄러에 의한 왕복 시간 성능향상을 보여 주고 있다. 이 실험에서 PVM 데몬의 즉각적인 스케줄링을 방해할 수 있는 부하 태스크(load tasks: 단순한 사칙 연산만을 반복 수행하는 CPU-bound 태스크)들을 실행함으로써, 시스템에 인위적인 부하가 걸리게 했다.

그림에서, 'N=n, Load=m'란 메시지 송수신에 참

여하는 두 대의 시스템상에 n개의 태스크 짝(송신 및 수신 태스크)이 수행되고, m개의 부하 태스크들이 수행되었음을 의미한다. 실험 결과에서 co-스케줄러상에서 수행된 EPVM이 그렇지 않은 EPVM보다 우수한 성능을 발휘함을 확인할 수 있다. 부하 태스크 수와 메시지 크기가 증가함에 따라, 요구기반 동적 co-스케줄러를 사용한 EPVM의 왕복 시간이 상대적으로 더 감소된다. 메시지의 크기가 4MB이고 4개의 부하 태스크들이 동작할 때, EPVM(O,1)상에서의 왕복 시간은 약 9초인 반면 EPVM(C,1)상에서는 3.4초로 줄어든다. 이 실험 결과는 co-스케줄러를 사용할 경우 메시지 왕복 시간 자체는 부하 태스크의 수와는 거의 무관하다는 것을 보여준다.

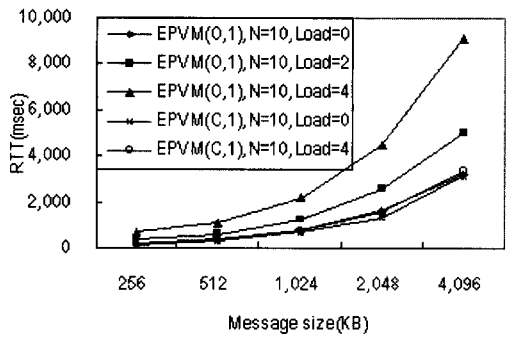


그림 7. co-스케줄링의 효과

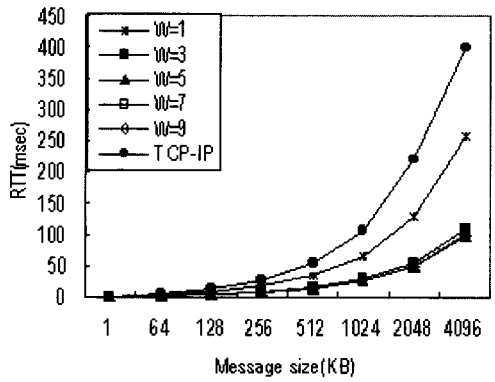


그림 8. 슬라이딩 윈도우의 효과
(각 곡선은 EPVM(O,W)에 해당함)

메시지 왕복 시간은 또한 PVM 데몬에 의해 사용되는 통신 프로토콜에 의존적이다. 앞장에서 언급한 바와 같이 PVM 데몬은 신뢰성 있는 통신을 위해 슬라이딩 윈도우 프로토콜을 사용한다. 그림 8은 윈도우 크기와 왕복 시간과의 관계를 보여준다. EPVM(O,1)이 가장 나쁜 결과를 나타내고, 윈도우 크기가 커질수록 결과는 더 좋아진다는 것을 알 수 있다. 그러나 윈도우 크기가 5를 넘어가면서 왕복 시간의 차이는 거의 없어진다. 그림에서 TCP-IP 곡선은 OPVM에서 PvmRouteDirect 메시지 전달 경로를 사용했을 때의 메시지 왕복 시간이다.

Ⅶ. 결 론

이 논문에서 PVM을 위한 zero-copy 통신 모델을 제안했다. 이 모델은 PVM 태스크, PVM 데몬, 네트워크 인터페이스 보드 등에 의해 접근할 수 있는 전역 공유 메모리를 사용하며, 초고속 통신망을 기반으로 하고 있다. 이 모델에서 PVM 태스크는 전송하고자 하는 메시지를 전역 공유 메모리에 저장하고, 메시지를 보낼 준비가 되었음을 PVM 데몬에게 알리며, 데몬은 해당 메시지를 커널을 통하지 않고 바로 초고속 통신망으로 전송한다. 이는 기존의 PVM 성능 저하의 주요 요인으로 작용하던 세 번의 메시지 복사 부하를 제거함으로써 상대적인 성능향상을 이룰 수 있게 해 준다..

제안된 모델은 기존의 태스크별 공유 메모리를 사용하던 PVM(SPVM) 시스템을 수정 보완함으로써 구현되었다. 이를 위해 기존 SPVM 시스템에 전역 공유 메모리 관리, 대기 태스크 스케줄링 알고리즘, Mynet API를 이용한 메시지 송수신, 슬라이딩 윈도우 프로토콜 등의 기능을 새로이 추가하였다. 뿐만 아니라, 기존 select() 함수를 이용한 블록킹 I/O와의 호환성을 유지하기 위해 Myrinet MCP와 Myrinet 드라이브를 수정하였다. 전역 공유 메모리를 위한 연속된 물리적 메모리를 확보하고 이를 관리하기 위한 새로운 모듈과 통신상의 지연을 줄이기 위해 특별히 설계된 co-스케줄러도 기존 Linux 시스템에 새로이 추가되었다.

제안된 모델의 실용성을 확인하기 위해 이를 적용한 PVM(EPVM)과 기존의 다른 PVM(SPVM과 OPVM) 상에서 다양한 실험을 실시하였다. 실험 결과 두 시스템간의 메시지 왕복 시간은 제안된 모델을 사용함으로써 현저히 줄일 수 있음을 확인하였다.

참 고 문 헌

[1] A. Geist, A. Beguelin, J.Dongarra, W.Jiang,R. Manchek, and V. Sunderamm, *PVM 3 User's*

Guide and Reference manual, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, Sept. 1994.

[2] Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, "Recent Enhancements to PVM," Dec. 1994, http://www.netlib.org/utk/papers/pvm-ijsa/ijsa.html.

[3] A. Basu, M. Welsh, and T. von Eicken, "Incorporating Memory Management into User-Level Network Interfaces," Department of Computer Science, Cornell University, Submitted for publication, Nov. 1996.

[4] Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, and Yuyaka Ishikawa, "The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network," Tsukuba Research Center, Japan (manuscript).

[5] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," *Proc. of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain, Colorado, December 1995.

[6] H. Xu and T. Fisher, "Improving PVM Performance Using the ATOMIC User-Level Protocol," *The High-Speed Network Computing Workshop '95*, 1995.

[7] S. L. Chang, David H.C. Du, J. Hsieh, M. Lin, and R. P. Tsang, "Enhanced PVM Communications over a High-Speed Local Area Network," *IEEE Parallel & Distributed Technology*, Fall 1995.

[8] H. Zhou and A. Geist, "Faster Message Passing in PVM," *9th International Parallel Processing Symposium : Workshop on High Speed Network Computing*, Santa Barbara, April 1995.

[9] "PVM on a Network of Workstations Based on Myrinet," http://www.cs .mcgill.ca/~marciac/gao.html.

[10] J. Hsieh, David H. C. Du, N. J. Troullier and M. Lin , "Enhanced PVM Communications over a HIPPI Local Area Network," *Proceedings of the 2nd International Workshop on High-Speed Network Computing (HiNet '96)*, Honolulu, April 1996.

[11] T. E. Anderson, D. E. Culler, and D. A. Patterson, "A Case for NOW (Network of Workstations)," *IEEE Micro*, Vol. 15, No. 1, pp. 54-64, Feb. 1995.

[12] M. Beck, H. Bohme, M. Dziadzka, U. Kunitz, R. Magnus, and D. Verworner, Linux Kernel Internals, Addison-Wesley, 1996.

[13] "MYRINET USER'S GUIDE," Myricom, http://www.myri.com:80/scs/documentation/mug.

[14] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. K. Su, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, Vol. 15, No. 1, pp. 29-36, Feb. 1995.

[15] P. Sobalvarro and W. Weihl, "Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors," *Proc. of Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 63-75, April 1995.

[16] W. Jiang, "PVM Internals - Multiprocessor Systems," Department of Computer Science, University of Tennessee, 1993 (manuscript).

[17] S. Pakin, V. Karamcheti, and A. Chien, "Fast Messages: Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors," *IEEE Concurrency*, 1997.

[18] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa, "Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication," Technical Report, Tsukuba Research Center, Real World Computing Partnership, 1997.

[19] 김인수, 심재홍, 최경희, 정기현, 김태근, 문경덕, "초고속 Myrinet 통신망에서의 PVM 성능 개선," *정보처리 논문지*, 제7권, 제1호, pp. 74-87, Jan. 2000

임 성 택(Sung-taek Lim) 정회원

1992년 2월 : 한국방송통신대학 전자계산학과공학사
1992년 2월 : 아주대학교 컴퓨터공학과 석사
1992년 3월 ~ : 아주대학교 컴퓨터공학과 박사과정

<주관심 분야> 지능형교육시스템, 원격교육, 멀티미디어시스템

심 재 홍(Jae-hong Shim) 정회원

1987년 2월 : 서울대학교 전산과학과 이학사
1989년 2월 : 아주대학교 컴퓨터공학과 공학석사
1989년~1994 : 서울시스템 공학연구소

1995년 3월~ 아주대학교 컴퓨터공학과 박사과정
<주관심 분야> 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템

최 경 희(Kyung-hee-Choi) 정회원

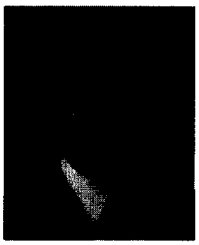
1976년 : 서울대학교 사범대학 수학교육과 이학사
1979년 : 프랑스 그랑데꼴 ENSEEIHT 공학석사
1982년 : 프랑스 Paul Sabatier대학 정보공학부 공학박사
1982년~ : 아주대학교 정보 및 컴퓨터공학부 교수
<주관심 분야> 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템

정 기 현(Gi-hyun Jung) 정회원

1990년 : 미국 Purdue대학 전기 전자공학부 Ph.D
1991~1992년 : 현대반도체 연구소
1993년~ : 아주대학교 전기전자공학부 교수

<주관심 분야> 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템

김 재 훈(Jae-Hoon Kim) 정회원



1992년 2월 : 한국대학교 전자공학과졸업
1992년 2월 : 아주대학교 컴퓨터공학과 석사
1994년 2월 : 아주대학교 컴퓨터공학과

<주관심 분야> 전자공학, 통신공학, 광통신 공학

문 경 덕(Kyung-Duk Moon) 정회원

1990년 2월 : 한양대학교 전산학과 공학사
1992년 2월 : 한양대학교 전산학과 공학석사
1997년~ : 한국전자통신연구소 선임연구원
<주관심 분야> 클러스터링, 홈 네트워크, Java