사용자 편의성을 고려한 VIA 라이브러리 개발에 관한 연구

정회원 이 상 기*, 이 윤 영**, 서 대 화***

Development of Easy-to-use VI Programming Library

Sang-ki Lee*, Yun-young Lee**, Dae-wha Seo*** ***Regular Member***

요 약

클러스터 내의 노드들 사이에서 대용량의 데이터들을 보다 빠르게 전송하기 위해서 경량 메세징(Lightweight Messaging) 기법이 등장하였다. 이 기법들 중 VIA(Virtual Interface Archi- tecture)는 사용자 수준에서 커널을 거치지 않고 네트워크 장치와 직접적으로 통신을 할 수 있게 하여 클러스터 시스템의 프로토콜로 자리를 잡아가고 있다. 그러나 이러한 장점에도 불구하고 프로그래밍이 어려워 제대로 숙지하기 까지는 많은 시간의 투자가 필요한 것이 사실이다. 이 논문에서는 개발자들이 좀더 쉽게 VIA에 접근할 수 있는 EVIL(Easy-to-use Virtual Interface Library)을 제안 하였다. 그리고 EVIL을 Native VIA, TCP/IP와 각각 성능을 비교, 평가하였다.

ABSTRACT

To transfer the large size of data more quickly among cluster nodes, the lightweight messaging scheme has been developed. VIA(Virtual Interface Architecture) allows that user can directly communicate with network devices without any interference of kernel and has become a communication protocol for clusters. But one must spend a lot of time to be skillful with it because of difficulties of programming. Therefore, we propose an easier library called EVIL(Easy-to-use Virtual Interface Library) that developers can easily deal with. We evaluated the performance of EVIL, Native VIA, TCP/IP respectively

Ⅰ. 서 론

하드웨어와 소프트웨어 기술이 발전함에 따라, 컴퓨팅 환경은 급속히 변화하고 있다. 컴퓨터 시스템에서 처리해야 할 데이터의 크기 및 그 수도 점차 늘어나게 되었다. 이에 따라 컴퓨터의 구조 및 성능도 하루가 다르게 변해서, 대용량의 데이터를 주고받는 일이 많아졌다. 특히 SAN(System Area Network)에서는 안정적인 데이터 통신이 가능해져, TCP/IP의 다양한 흐름 제어 메커니즘들은 오히려 오버 헤드로 작용하기도 한다[1][5]. 이러한 점들 때문에 Berkeley NOW 시스템의 Active Message (AM), Cornell대학의 U-Net, Princeton 대학의 SHRIMP(Scalable High performance Really In-expensive Multi Processor) 프로젝트에서 제안한 VMMC(Virtual Memory- Mapped Communication), Illinois 대학의 FM(Fast Message)등의 다양한 경량 메세징(Lightweight Messaging)기법들이 등장하게 되었다[8].

VIA(Virtual Interface Architecture)도 역시 이러한 경량 메세징 기법들 중의 하나로써 Microsoft, Compaq, Intel, 등의 3사가 제안한 클러스터 통신 프로토콜이다[1][8].

* 경북대학교 정보통신학과 (leesky@palgong.knu.ac.kr),
** 넷컴스토리지 (yylee@netcomstorage.com),
*** 경북대학교 전자공학과 (dwseo@ee.knu.ac.kr)
논문번호 : 010192-0723, 접수일자 : 2001년 7월 23일

이들 새로운 프로토콜들에 있어서 주요 관심사는 패킷 전송 중의 불필요한 복사를 줄여서 메시지의 전송 속도를 높이는 것과 기존 프로토콜을 단순화 시키는 것에 있다 [5][9].

VIA를 사용하면 TCP/IP등의 프로토콜을 대신해 CPU가 담당한 네트워크 통신 오버 헤드를 제거해 줄 수 있고, 실제 기업 환경에 사용할 수 있는 시스템 수준의 고속 통신을 가능하게 하며, 응용 프로그램 수준에서 CPU를 거치지 않고 네트워크 장치와 직접적으로 통신을 할 수 있도록 해줄 수 있다. 이런 점 때문에 현재 VIA가 널리 사용되고 있기는 하지만 프로그래밍이 어려워 제대로 숙달되기까지는 많은 시간의 투자가 필요한 것이 사실이다. 그래서 일반적인 개발자들이 쉽게 사용하면서도 성능을 최대한 발휘할 수 있는 인터페이스가 필요하다. 이 논문에서는 EVIL(Easy-to-use Virtual Interface Library)이라는 VIA 프로그래밍 라이브러리를 제안한다.

본 논문의 제 2장은 VIA, TCP/IP등에 대한 전체적인 배경에 관한 연구, 제 3장은 제안한 EVIL의 특징과 API의 구조, 제 4장은 실험 및 결과, 마지막으로 제 5장은 결론으로 이루어져 있다.

Ⅱ. 배경 연구

1. Virtual Interface Architecture

VIA는 Microsoft, Compaq, Intel등 3개의 선두 기업들이 클러스터 환경이나 SAN (System Area Network)에서 사용할 고성능의 네트워크 기술을 위한 인터페이스 표준의 필요에 따라서 제안한 것이다[5][8].

VIA의 주목적은 실제 기업 환경에 사용할 수 있는 시스템 수준의 고속 통신을 가능하게 하기 위한 것이며 이는 응용 프로그램 수준에서 CPU를 거치지 않고 네트워크 장치와 직접적으로 통신을 할 수 있도록 해 현재 범용으로 사용되는 TCP/IP등의 프로토콜에서 CPU가 담당한 네트워크 통신 오버 헤드를 제거한다[7]. VIA의 특징은 다음과 같다.

- 각각의 VI는 통신상의 종점을 나타내며 이 한 쌍의 종점은 점대점 통신을 논리적으로 연결시켜 준다.
- 하나의 프로세스는 하나 이상의 VI를 소유할 수 있다.
- 네트워크 어댑터는 통신상의 종점을 가상화시키는 역할을 하며, VI사이의 신뢰성 있는 데이터 전송을 담당한다.
- VI 구조는 기본적으로 가상 인터페이스, 완료 큐(Completion Queue), VI 제공자 (VI Provider), VI 소비자 (VI Consumer)의 4개의 모듈로 구성된다[7].

2. VIA와 TCP/IP의 비교

그림 1은 VIA와 TCP(UDP)/IP의 차이점을 설명해 놓은 것이다. 전통적인 TCP(UDP)/IP 프로토콜에서는 단지 커널 만이 네트워크 인터페이스에 접근할 수 있고 사용자의 입장에서는 불가능한 것이었다. 따라서 네트워크를 통해서 통신을 하려면 사용자 버퍼 영역과 시스템 버퍼 영역 사이에서 데이터의 복사가 일어나게 된다. 그러나 VIA는 가상 인터페이스(VI)를 통해 사용자 프로세스가 독점적으로 네트워크 인터페이스를 소유한 것처럼 보이게 한다. 사용자 프로세스는 VI를 통하여 운영 체제의 개입이 없이 안전하게 네트워크 인터페이스에 직접 접근할 수 있고, 사용자 버퍼와 시스템 버퍼 사이의 데이터의 복사 과정을 없애 무복사 전송(zero-copy)을 수행한다[5].

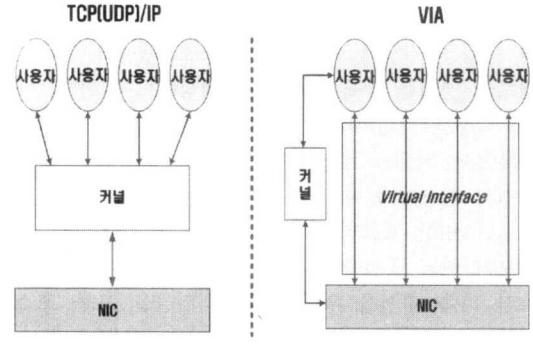


그림 1. TCP(UDP)/IP와 VIA의 비교

TCP/IP에서는 커널 만이 네트워크 디바이스에 접근할 수 있기 때문에 보호 기능이 커널 수준 에서 제공되었으나, 이와 같이 사용자 프로세스가 직접 디바이스를 접근하는 경우에는 가상 주소의 해석과 보호 기능을 VIA가 제공하게된다.

그러나 이러한 장점에도 불구하고 프로그래머의 관점에서 본다면 VIA를 이용한 프로그래밍은 소켓과 같은 기존의 네트워크 프로그램 인터페이스보다 쉽지가 않고 흐름 제어, 수신측의 디스크립터 준비와 같은 문제들을 프로그래머가 직접 해결해야만 한다[2]. 이런 이유로 보다 쉽게 VIA프로그래밍을 할 수 있도록 하기 위해서는 프로그래머가 사용하기 쉬운 라이브러리가 필요하게 되었다.

Ⅲ. 사용자 편의성을 고려한 VIA 라이브러리

본 장에서는 사용자가 VIPL(Virtual Interface Programming Library)[7]과 VIA에 대한 자세한 이해가 없이도 쉽고 간단하게 VIA를 이용한 프로그램을 작성할 수 있는 EVIL 라이브러리의 버퍼 관리 기법 및 사용자 인터페이스에 대해서 기술한다.

1. EVIL API의 구성

EVIL API를 구성하는 데 가장 중점을 둔 것은 사용의 편의성이며, 이를 위해 VIA를 이용한 고성능 통신을 위해 필요한 여러 단계들을 직관적인 단위로 묶어 단순한 함수 세트를 정의하였다. 그림 2는 EVIL을 구성하는 함수들과 각 함수들의 역할을 간단히 나타낸 것이다.

1) OPEN

OPEN은 채널을 생성하는 역할을 하며 via_open()함수로 구현되었다. VIA 채널을 생성하는 단계에서는 NIC(Network Interface Card)에 접근하여 사용 권한을 획득하고, VI를 생성하는데 필요한 여러 속성을 지정해 주며, 응용 프로그램에서 필요한 메모리 영역을 등록하여 VI를 생성하도록 한다. 이때, 사용자가 정의하는 개수의 메모리 버퍼 공간을 등록해서 라운드 로빈(Round-robin) 방식으로 돌아가면서 사용한다. 여기서, 송신과 수신을 위한 디스크립터 또한 메모리 공간을 할당하고, 등록하여 각 메모리 버퍼 공간을 가리키도록 작성하도록 하였다. 그런 다음 수신을 위한 디스크립터들을 모두 포스팅한다. 이것은 송신측에서 데이터를 보냈을 때 수신 디스크립터가 준비되어있지 않을 경우 데이터가 유실되는 것을 막기 위하여, 항상 수신을 위한 디스크립터들이 준비되어 있도록 하기 위함이다.

2) WAIT

WAIT는 개설된 채널을 통해서 다른 노드로부터의 연결 대기 및 수락의 역할을 하며 VipConnectWait(), VipConnectAccept()함수를 한 함수 내에서 처리하도록 via_wait()함수로써 구현하였다. 즉, 다른 호스트로부터의 접속 요청을 기다리다가 이를 수락하는 역할을 하도록 구현하였다.

3) CONNECT

CONNECT는 via_connect()함수로 구현하였는데 VIA 채널을 통해서 연결을 요청하는데 사용한다.

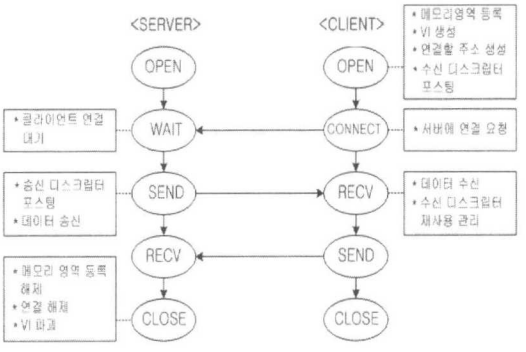


그림 2. EVIL 함수들의 역할

다른 호스트에 연결 하고자 할 때 사용하는 함수로 내부적으로 VipConnectRequest()함수를 호출하여, 접속 대기 중인 상대방 호스트에 연결을 요청한다.

4) SEND와 RECV

SEND와 RECV는 각각 via_send()와 via_recv()로 구현하였고 VIA 채널을 통해서 데이터를 송수신하는 역할을 한다. via_open()에서 송수신을 위한 메모리 버퍼 공간을 할당하고 등록한 것을 이용하여, 노드간의 데이터를 전송하도록 해준다.

5) CLOSE

CLOSE는 개설된 VIA채널을 닫는 역할을 하며 via_close()함수를 사용하여 구현하였다. VIA채널을 닫기 위해서는 데이터의 전송이 끝난 후에는 VIPL을 통해 할당받은 자원들을 시스템에 돌려주는 과정이 필요하다. 먼저 등록한 메모리를 반환하여야 하며, 보호 테그(protection tag)도 반환하여야 한다. VIA 채널을 통해 연결도 끊어주어야 하며, VI를 파괴하고, NIC의 사용 권한도 반환하여야 한다. 이러한 많은 작업들을 동시에 하나의 함수를 통해서 수행하도록 하였으므로, 응용 프로그램의 작성이 한층 더 용이해 진다.

2. 다중 버퍼의 관리

VIPL에서 데이터의 수신을 위해서는 수신 큐에 데이터를 받아서 저장할 사용자 영역의 버퍼를 지정한 다음 그 수신 큐를 미리 포스팅한 다음에 데이터 수신이 종료되기를 기다리게 된다. 이러한 과정은 여러 가지 방법으로 구현될 수 있고, VIPL API 수준에서는 흐름 제어를 하지 않기 때문에 이를 보장하기 위해서는 다양한 방법으로 사용자가 프로그램을 하여 주어야 한다. 이것을 적당한 방법으로 구현하여 사용자에게는 하나의 함수로 사용할

수 있게 해 준다면, 사용자의 편의를 상당히 높일 수 있다.

그림 3은 EVIL에서의 다중 버퍼 할당을 보여주고 있다. EVIL에서는 데이터 수신을 위해 다중 버퍼를 할당하고, 이를 라운드 로빈 방식으로 사용한다. via_send()는 응용 프로그램으로부터 송신을 요청 받은 데이터가 있는 메모리 공간을 등록한 후 이를 가리키는 디스크립터를 포스트해 주도록 작성하였다. via_recv()에서는 대기 중인 여러 개의 디스크립터 중 데이터 수신이 끝나 종료되는 디스크립터가 가리키는 메모리 버퍼의 데이터를 응용 프로그램에 넘겨주도록 구현하였으며, 사용이 끝난 디스크립터는 다시 수신을 위해 대기하도록 포스트하였다.

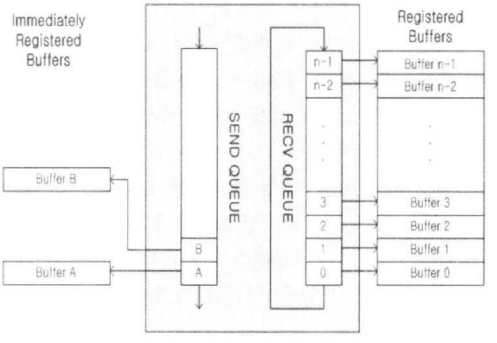


그림 3. EVIL의 다중버퍼

실제로는 이렇게 복잡한 과정들이 응용 프로그램에 코딩되어야 하지만 이는 상당히 복잡한 과정이므로 이런 과정들을 숨겨주어 응용 프로그램 작성이 상당히 용이해지도록 하였다.

3. EVIL의 라이브러리

EVIL 프로그램을 작성하기 위해서는 우선 EVIL을 구성하는 파일들을 make를 통하여 설치한다. 그리고 응용 프로그램에 evil.h를 include한 후 다음의 함수들을 사용하면 된다. Make하기 전에 각 메모리 버퍼 영역의 크기와 개수, VIA를 지원하는 네트워크 장치에 대한 설정들을 미리 수정해 주어야 한다.

1) int via_open
(ViaChann *vc, char *peer, int port);

via_open 함수는 채널을 생성하는 역할을 하며 NIC(Network Interface Card)에 접근하여 사용 권한을 획득하고, VI를 생성하는데 필요한 여러 속성을 지정해 주며, 응용 프로그램에서 필요한 메모리 영역을 등록해 준다. 이 함수를 사용하기 전에 채널의 정보를 포함하는 ViaChann 구조체를 선언해 주어야 한다. peer는 접속하고자 하는 노드의 이름을 나타내고, port는 TCP/IP의 port와 같으며 1~255까지의 정수로 표현된다. 이 함수를 통해서 ViaChann의 구조체에서 가리키는 peer와 연결될 채널의 정보를 얻어온다.

2) int via_wait(ViaChann vc);

via_wait()함수는 채널을 통한 연결 대기 및 수락의 역할을 하며 이미 선언된 via_open()을 통해 얻은 채널의 정보를 사용하여, 다른 노드로부터 접속이 올 때까지 대기한다.

3) int via_connect(ViaChann vc);

via_connect 함수는 VIA 채널을 통해서 연결을 요청하는데 사용하며 이전에 선언하여 via_open()을 통해 얻은 채널의 정보를 사용하여, via_open() 시에 peer로 지정한 노드로 접속을 시도한다.

4) int via_send(ViaChann vc, char *data, int size);

via_send는 VIA 채널을 통해서 데이터를 송신하는 역할을 한다. 이 때 size 인자에 보내고자 하는 데이터의 크기를 지정해 주어야 한다.

5) int via_recv(ViaChann vc, char *data, int ize);

via_recv는 VIA 채널을 통해서 데이터를 수신하는 역할을 한다. 이 때 size 인자에는 수신자 측에서 받고자 하는 데이터의 크기를 지정해 주어야 한다.

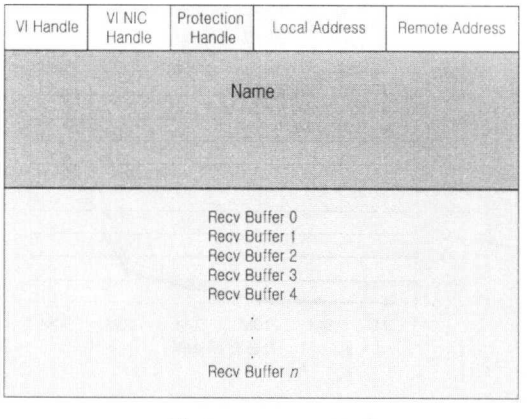


그림 4. ViaChann 구조체

6) int via_close(ViaChann vc);

via_close는 개설된 VIA채널을 닫는 역할을 하며 할당 받은 자원들을 반환한다.

채널의 정보를 가리키는 ViaChann 자료 구조는 그림 4와 같이 구성되어 있다.

채널의 이름(name)과 Virtual Interface의 핸들(viHand), NIC의 핸들(nicHand), 그리고 메모리 보호 태그(ptag)와 로컬 호스트와 리모트 호스트의 주소(localAddr, remote Addr)가 있으며, 여러 개의 등록된 메모리 버퍼 영역(mem_buff)을 채널이 가지게 된다.

Ⅳ. 실험결과

1. 실험 환경

실험을 위하여 4대의 Intel Pentium III-600Mhz 시스템을 사용하였다. 실험에 사용된 리눅스는 커널 2.2.14버전으로 VIA를 제공하는 환경으로는 Giganet사의 cLAN을 사용하였다. Giganet 사의cLAN은 cLAN1000 NIC상에서 동작하는 것으로 하드웨어적으로 VI를 지원하는 NIC이다.

2. 성능 평가

성능 평가를 위하여 EVIL과 Native VIA인 VIPL을 이용하여 동일한 크기의 메시지를 지정한 양만큼 반복하여 전송하여 그 성능을 측정하는 프로그램을 작성하였다. 그리고 VIA상에서 IP를 에뮬레이트하도록 Giganet 사에서 제작한 cLAN 어뎁터를 위한 드라이버인 LANE (LAN Emulation)[3]을 이용하는 TCP/IP socket을 이용하여 같은 일을 하도록 프로그램을 작성하여 EVIL, Native VIA로 작성된 프로그램과 성능을 비교하였다.

그림 5에 나타난 결과는 전송하는 데이터의 총량을 1MB에서 500MB로 늘여가면서 평균 전송 대역폭을 측정하였다. 이때 한번에 전송되는 메시지의 크기는 16KB로 고정하였다.

EVIL을 사용하였을 때 Native VIA를 사용한 경우보다 약 5~10%의 손실이 있음을 알 수 있다. 이러한 손실이 발생하는 이유는 EVIL을 사용할 경우 송신측에서 송신할 데이터가 위치한 메모리 영역을 매번 등록하고 해제해 주기 때문에 발생하는 오버 헤드 때문이다. Native VIA를 사용할 경우에는 한 번 등록한 메모리를 계속 사용할 수 있어 더 나은 성능을 보이게 된다. 그렇지만 디바이스 드라이버 수준에서 IP를 에뮬레이트한 LANE를 이용한 경우보다는 39~62%의 성능개선이 있음을 확인할 수 있었다. 그리고 전송하는 데이터의 양이 커지더라도 일정수준의 성능을 나타내고 있었다.

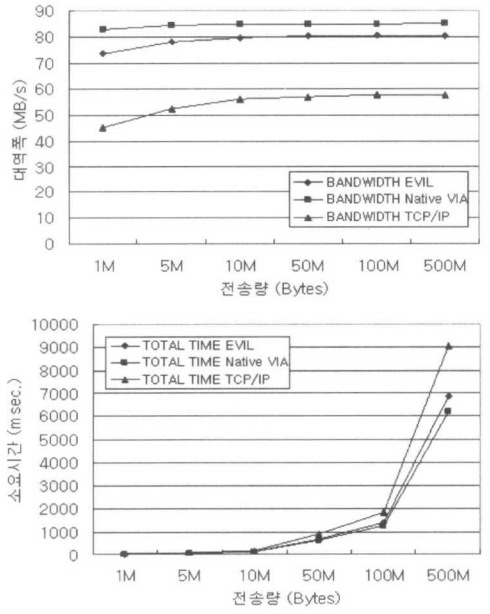


그림 5. EVIL, Native VIA, TCP/IP의 데이터 크기 변화에 따른 성능 변화

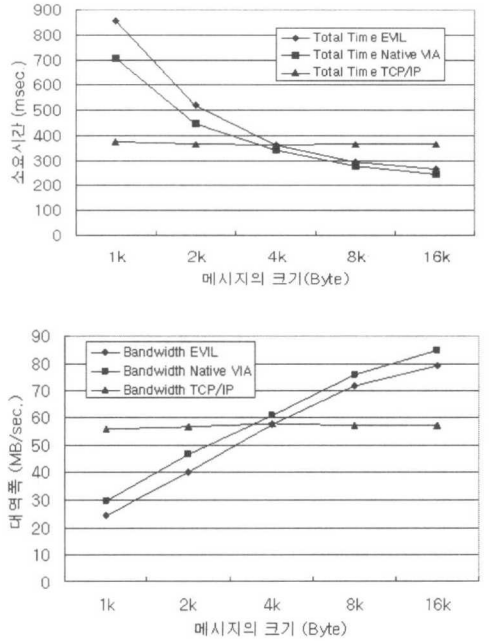


그림 6. EVIL, Native VIA, TCP/IP의 데이터 단위 변화에 따른 성능 변화

그림 6은 20MB의 메시지를 전송하는데, 전송하는 메시지의 크기를 변화시키면서 그 성능의 변화를 살펴보았다. 메시지의 크기가 클수록, 성능이 좋아지는 것을 확인할 수 있는데, 큰 메시지를 한번 보내는 것이 작은 메시지로 여러 번 나누어 보내는 것보다 효율적임을 보여준다. 한번에 전송하는 메시지의 크기를 변화시킨 결과 약 7~17%정도의 손실이 발생하였다. 적은 크기의 메시지를 보낼 때와 큰 메시지를 보낼 때 EVIL과 Native VIA의 경우 성능 차가 큰 것에 비하여, LANE의 경우 일정한 성능을 보이고 있다. 이것은 LANE 내부의 흐름 제어 메커니즘과 관련이 있으며, 향후 EVIL에도 그러한 메커니즘을 적용한다면, 적은 메시지를 전송하는 성능도 크게 보완할 수 있을 것이다.

V. 결 론

본 논문에서는 SAN환경에서 경량 메세징 기법의 일종인 VIA를 이용하는 응용 프로그램의 작성을 용이하게 해주는 라이브러리인 EVIL을 구현하고 그 성능을 평가하였다.

EVIL을 사용하였을 때 Native VIA의 API 세트인 VIPL을 이용하여 작성한 프로그램의 약 1/3에 해당하는 코딩으로 같은 작업을 수행하는 프로그램을 작성할 수 있었으며, 그 성능 또한 크게 차이가 나지 않았다. 코딩의 양이 줄었다는 것은 그만큼 응용 프로그램의 개발 속도가 빨라짐을 의미하며, 디버깅이 용이하여졌음을 의미한다.

특히 EVIL에서는 매우 간단하면서도 직관적인 함수들을 이용하므로, VIA를 이용한 프로그램을 작성하는 것이 매우 간편해졌다. 간편함을 추구하므로 성능 저하가 우려되었지만, 실험을 통해 확인한 결과 Giganet 사에서 제공하는 예제 프로그램에 비해 성능이 크게 떨어지지 않았으며 따라서 경량 메세지로써의 기능을 충분히 하고 있음을 확인할 수 있었다.

향후 파일 전송을 위한 흐름 제어 방식들을 적용하여 작은 크기의 메시지의 전송에도 높은 성능을 발휘하는, 전체적으로 더 안정적이며 높은 처리율을 나타내는 연구가 계속될 것이다.

참 고 문 헌

[1] 김강호, 김진수, 김해진, "리눅스 상의 VIA 구현 비교", 한국 정보과학회 학술 발표논문집, pp. 627-629, 2000년 10월(제27권 2호)

[2] 하순회, 기양석, 김선재, "VIA기반의 병렬 라이브러리에 관한 연구", 정보과학회지, pp. 28-39, 2000년 3월

[3] "cLan for Linux, Software User's Guide", Emulex, 2001

[4] Rajkumar Buyya, "High Performance Cluster Computing", Prentice Hal, Vol. 1, 1999.

[5] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, C. Dodd. "The Virtual Interface Architecture", IEEE Micro. March/ April 1998.

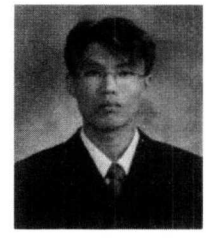
[6] "VI Architecture Software Developer's Guide", Emulex, 2001.

[7] "Virtual Interface Architecture Specification", draft revision 1.0, 1997.

[8] "Virtual Interface (VI) Architecture - The New Open Standard for Distributed Messaging Within a Cluster", Compaq, 1998.

[9] Yangsuk Kee, Soonhoi Ha, "xBSP: An Efficient BSP Implementation for cLAN", Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ ACM International Symposium on, pp. 237-244, 2001.

이 상 기(Sang-ki Lee) 정회원

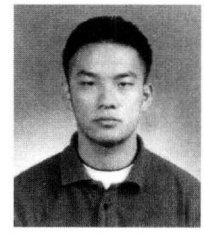


2001년 2월 : 경일대학교 영어영문학과 졸업

2001년 3월 ~ 현재 : 경북대학교 정보통신학과 석사과정

<주관심 분야> 클러스터 컴퓨팅, 병렬 파일 시스템, 운영 체제

이 윤 영(Yun-young Lee) 정회원



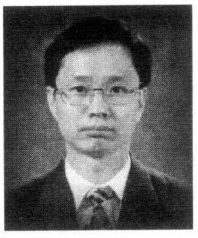
2000년 2월 : 경북대학교 전자공학과 졸업

2002년 2월 : 경북대학교 전자공학과 석사

2002년 1월 ~ 현재 : (주)넷컴스토리지 기술 연구소

<주관심 분야> 병렬처리, 분산처리, 클러스터 컴퓨팅, 암호화 시스템

서 대 화(Dae-wha Seo) 정회원



1981년 2월 : 경북대학교 전자공학과 졸업
1983년 2월 : 한국과학기술원 전산학과 석사
1993년 2월 : 한국과학기술원 전산학과 박사
1983년 ~ 1995년 : 한국전자통신연구원
1995년 ~ 현재 : 경북대학교 전자전기공학부 부교수
<주관심 분야> 병렬·분산 운영체제, 병렬 컴퓨터 구조, 병렬 파일 시스템