소켓 및 RMI 기반 자바 메시지 전달 시스템의 구현 및 성능평가☆

Implementation and Performance Evaluation of Socket and RMI based Java Message Passing Systems

방 승 준* 안 진 호**
Seung-Jun Bang Jin-Ho Ahn

요 약

본 논문은 자바 언어로 완성된 MPI(Message Passing Interface) 스펙인 MPJ(Message Passing in Java) 스펙을 준수하여 병렬 컴퓨팅 환경에서 메시지 통신 인터페이스를 제공하는 JMPI(Java Message Passing Interface) 라이브러리를 설계하고 구현하였다. 이 라이브러리는 간단하면서도 매우 편리한 GUI(Graphical User Interface) 도구를 제공하여, 사용자가 손쉽게 병렬 컴퓨팅 환경을 구성할 수 있다. 또한, 본 논문에서는 두 가지 전형적인 분산 시스템 통신 메커니즘인 소켓과 RMI(Remote Method Invocation)방식을 이용하여 두 가지 버전의 시스템을 구현하였고, 기존의 JPVM 시스템과의 성능을 벤치마크 애플리케이션들을 통하여 컴퓨터 대수의 증가에 따른 처리 속도를 비교해 보았다. 실험 결과로는 본 논문에서 제시한 JMPI 시스템이 JPVM시스템보다 다양한 측면에서 높은 성능을 발휘한다는 것과 컴퓨터의 가장 효율적인 처리 속도는 애플리케이션에 따라 컴퓨터의 수를 증가시킨다고 해서 일정하게 처리속도가 증가하는 것이 아니라 네트워크의 트래픽을 고려하여 컴퓨터의 수를 증가시켰을 때 얻을 수 있다는 것을 보여준다. 마지막으로 컴퓨터의 수가 증가함에 따라 RMI를 사용하여 메시지를 전달하는 것이 소켓에 부속된 객체 스트림을 사용하여 메시지를 전달하는 것보다 효과적이라는 것을 알 수 있다.

Abstract

This paper designs and implements a message passing library called JMPI (Java Message Passing Interface) which complies with MPJ (Message Passing in Java), the MPI standard Specification for Java language. This library provides some graphic user interface tools to enable parallel computing environments to be configured very simply by their administrators and JMPI applications to be executed very conveniently. Also in this paper, we implement two versions of systems using Socket and RPC which are both typical distributed system communication mechanisms and with three benchmark applications, compare performance of these systems with that of an existing system JPVM depending on the increasing number of the computers. Experimental results show that our systems outperform JPVM system in terms of various aspects and that the most efficient processing speedup can be obtained by increasing the number of the computers in consideration of network traffic through processing evaluation. Finally, we can see that, as the number of computers increases, using RMI to transmit a message is more effective than using object streams attached to sockets to transmit a message.

☞ keywords : Parallel computing, Java language, MPJ, RMI, Socket

1. 서 론

최근 네트워크 및 컴퓨터 프로세서 기술들이 매우 빠르고 지속적으로 발전함에 따라 현재의 일반 개인용 컴퓨터가 불과 10년 전 슈퍼컴퓨터의 성능을 발휘하거나 그 성능을 능가하고 있다 [18]. 이에 따라 병렬 및 분산 프로그래밍의 관심이 높아져 네트워크로 연결된 컴퓨터에서 데이터

* 준 회 원 : 경기대학교 대학원 전자계산학과 석사과정
ppangas@hotmail.com
** 종신회원 : 경기대학교 정보과학부 컴퓨터학부 조교수
jhahn@kyonggi.ac.kr(교신저자)
[2007/04/12 투고 - 2007/04/23 심사 - 2007/6/27 심사완료]
☆ 이 연구는 경기도지역협력연구센터 사업(GRRC)의 지원에 의해 연구되었음

와 명령의 분산처리를 이용한 병렬 프로그래밍 환경을 제공하는 연구가 활발히 진행되어 e-Science, e-Business, 데이터마이닝 등과 같은 많은 분야에서 적용되고 있다[4]. 그러나 각 연구들이 병렬 컴퓨터나 분산 환경에서 각각 다른 프로그래밍 인터페이스로 구현되었기 때문에 이들 시스템 간의 사용자가 작성한 응용 프로그램들이 서로 호환되지 않는 문제가 있다.

이러한 응용 프로그램의 호환성 문제를 해결하기 위해 병렬 및 분산 프로그래밍 모델로서 MPI(Message Passing Interface)[8, 13]과 PVM (Parallel virtual Machine)[20]이 있다. 두 모델은 서로간의 장단점을 가지고 있으나, 메시지 전달 인터페이스 표준 모델의 현재 선호도 및 발전경향에 따라 본 논문에서는 MPI 표준스펙에 초점을 맞추고자 한다. MPI는 프로세스들 간의 통신을 위해 코드에서 호출해 사용하는 서브루틴(Fortran 언어에서) 또는 함수(C 언어에서)들의 라이브러리 표준이다. MPI는 기존의 연구에서 사용하던 라이브러리들을 참조하여 각각의 시스템에서 공통된 내용의 형식을 받아들여 메시지 기반의 병렬 프로그램에서 필요한 명령어를 표준화해서 인터페이스로 정의하였다[8, 13]. MPI 표준에 따라 구현된 대표적인 시스템들로는 MPICH, LAM, CHIMP 등이 있다[5]. 이 시스템들과 PVM은 표준 인터페이스를 사용하여 호환성은 있지만 각기 다른 플랫폼을 기반으로 구현되어 서로 다른 기종의 컴퓨터에 이식하는데 상당한 어려움이 있다[19].

자바 언어는 JVM(Java Virtual Machine)로 인하여 플랫폼에 독립적이라는 특징을 가지고 있어 분산 프로그래밍 언어로서 큰 장점을 가지고 있다. 따라서 자바 언어로 완성된 MPI 스펙인 MPJ(Message Passing in Java) 표준 스펙이 제시되었는데, 본 논문에서는 이러한 MPJ 표준 스펙을 준수하여 병렬 컴퓨팅 환경에서 메시지 통신 인터페이스를 제공하는 JMPI(Java Message Passing Interface) 라이브러리를 설계하고 구현하였다[2, 4]. 따라서 JMPI는 호환성과 이식성이 뛰어나 추가적인 하드웨어나 시스템의 변경 없이도 JVM만 설치되어 있으면 어떠한 환경에서도 병렬 애플리케이션에 라이브러리를 추가하여 간단하지만, 편리하게 사용할 수 있다는 특징이 있다. JMPI는 MPI와 동일한 프로그래밍 인터페이스를 제공하여 기존의 사용자들이 손쉽게 사용할 수 있다. 또한 이 라이브러리는 간단하면서도 매우 편리한 GUI(Graphical User Interface) 도구를 제공하여, 사용자가 손쉽게 병렬 컴퓨팅 환경을 구성할 수 있다.

특히, 본 논문에서는 두 가지 전형적인 분산 시스템 통신 메커니즘인 소켓과 RPC(Remote Procedure Call)방식을 이용하여 두 가지 버전의 시스템을 구현하였다. 소켓의 경우 일반적인 분산 시스템에서도 유연성 있게 사용되지만 구현이 RPC와 비교하여 까다롭다는 단점을 지니고 있다[7]. 다른 메커니즘인 RPC의 경우 객체 지향 분산시스템에서 사용할 수 없는 단점을 가지고 있다. 따라서 객체 지향 분산 시스템에 적절한 통신 메커니즘이 필요한데, 그 중 하나가 자바 언어에 기반한 RMI(Remote Method Invocation)[17]이다[11, 14, 15]. 본 논문에서는 MPJ 표준 스펙을 따라 자바 언어로 구현된 소켓기반 메시지 전달시스템과 RMI기반 메시지 전달시스템 간의 성능을 PVM의 라이브러리를 오직 자바로 구현한 JPVM(Java Parallel Virtual Machine) 시스템[19]을 벤치마크 애플리케이션들을 통하여 컴퓨터 대수의 증가에 따른 처리 속도를 비교하고 분석하고자 한다.

본 논문의 구성은 다음과 같다. 2절은 관련 연구를 설명하고, 3절에서는 본 논문에서 제시한 JMPI 시스템의 설계와 구현에 대하여 설명한다. 4절에서 세 개의 벤치마크 애플리케이션의 수행을 통한 두 시스템의 성능 평가와 실험 결과를 제시하고, 5절에서는 결론을 맺는다.

2. 관련연구

소켓 통신을 이용해서 분산 시스템을 구축하려면, 많은 시간과 노력이 필요하다[7, 11, 14, 15].

왜냐하면, 실제적인 분산 시스템의 기능보다는 소켓 프로그래밍을 하는데 많은 노력이 들기 때문이다. 이러한 어려움을 줄이기 위해, 분산 시스템을 구축하는데 있어서, 널리 사용된 방법으로 RPC라는 기술이 있다. RPC는 프로그래머가 소켓 프로그래밍을 하지 않고도, 다른 컴퓨터 내에서 실행되는 컴퓨터 프로그램의 함수를 호출할 수 있도록 지원해준다. 따라서 RPC를 사용하면 분산 시스템 구축 시 프로그래머가 복잡한 소켓 프로그래밍을 할 필요없이, 마치 로컬 시스템의 함수 라이브러리를 호출하듯이 프로그램을 작성해도 되었기 때문에 널리 사용되어져 왔다[7].

자바 언어에서는 JDK 1.1에서 소개된 RMI라는 메커니즘을 이용하면, RPC가 제공하는 분산시스템을 손쉽게 구축할 수 있다. 더욱이, RPC의 기능은 단순한 원격지 프로그램의 함수 호출인데 반해, RMI는 원격지 객체의 메서드 호출을 가능하게 함으로써 분산객체 시스템(Distributed Object System)에 적합한 메커니즘이다.

RMI도 RPC와 마찬가지로 프로그래머가 소켓 통신에 대한 고려 없이, RMI관련 클래스만으로 프로그램을 작성하면, RMI 인프라에서 네트워크와 관련된 복잡한 처리를 담당해 준다. 결론적으로 RMI를 사용하면, 원격지 객체의 메서드를 호출할 때, 마치 같은 로컬 JVM 내의 객체 메서드를 호출하는 것과 거의 유사한 호출이 가능하다. 이때, 메서드의 인자로 사용되는 것은 자바 기본형뿐만 아니라, 객체직렬화가 가능한 모든 클래스 객체들이 전달될 수 있으며, 반대로 반환 값으로 자바 기본형과 객체들이 리턴될 수 있다. 따라서 네트워크 응용 프로그램을 구현할 때 RMI를 이용하면, 소켓과 비교하여 간단하게 프로그래밍을 할 수 있다[7, 11, 14, 15].

메시지 전달 라이브러리는 네트워크 컴퓨터로 구성된 분산 시스템에서 분산 애플리케이션이 실행되는 것을 가능하게 한다[1, 6, 16]. 가장 널리 사용되는 표준 메시지 전달 API는 MPI포럼의 MPI와 PVM이다[13, 20]. 이 API에 의해 프로그래머들은 쉽게 병렬 프로그래밍을 할 수 있게 되었다. 그러나 이전의 MPI와 PVM 표준은 Fortran, C 그리고 C++로 구현 되어 있어서 플랫폼에 독립적인 대규모의 분산 시스템을 제공할 수 없었다. 이러한 문제점 때문에 자바로 바인딩 하려는 많은 시도가 있었다. MPI에 관련된 첫 번째 시도는 MPI 라이브러리를 JNI(Java Native Interface)로 래핑(wrapping)시킨 것이다[3, 12]. 이 방법은 비교적 빠르다는 장점이 있었지만 자바 보안 모델과 충분한 이식성을 지원하지 못한다[7]. 두 번째 시도는 오직 자바로 MPI를 구현하는 것이다[10]. 첫 번째 방법에 비교하여 기존의 문제점은 해결하였지만 성능이 저하된다는 단점을 가지고 있다[7].

자바 그랜드 포럼(Java Grande Forum)의 메시지 전달 그룹(Message-Passing Group)은 MPI와 같은 라이브러리를 자바 언어로 바인딩 시킨 여러 프로토타입이 등장함에 따라 1998년에 구성 되었다[2]. 자바 그랜드 포럼(Java Grand Forum)은 MPJ(Message Passing in Java)[2, 4]라 불리는 자바 언어로 구현된 MPI를 제시하였다. MPJ는 기존의 연구와 다양한 요구사항을 반영하는데 기반하고 있다. 대표적인 예로 MPJ/Ibis[5]는 최초로 성능을 위한 최적화된 기술과 높은 네트워크 스피드를 목적으로 하였으며, 오직 자바로만 구현되었다. 그러나 이 시스템은 통신 메커니즘으로 소켓과 RMI을 모두 이용하여 메시지 전달 시스템의 성능에 대한 비교 및 분석을 하지 않았다.

PVM에 관련된 자바로의 바인딩에 대한 시도는 PVM 라이브러리를 오직 자바로 구현한 JPVM 시스템[19]이 있다. JPVM은 기존의 PVM과 유사한 라이브러리를 가지고 있어 PVM을 사용해본 프로그래머라면 쉽게 사용법을 배울 수 있다[19]. 하지만 JPVM은 병렬컴퓨팅 환경을 구성하기위한 편리한 GUI를 제공하지 않아, 많은 컴퓨터를 이용하여 병렬처리를 하려고 한다면 그 환경을 구성하는데 많은 시간과 노력이 필요하다. 또한, 본 논문에서 구현된 JMPI 시스템에 비해 다양한 측면에서 시스템 성능이 떨어진다.

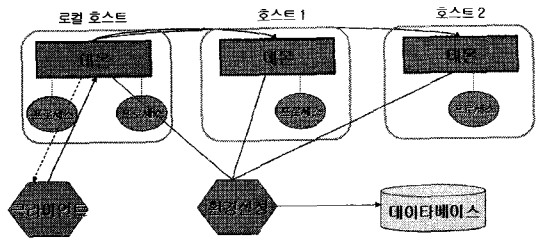
본 논문에서는 병렬컴퓨팅 환경의 구성을 위해 간단하지만 매우 편리한 GUI를 제공하고, 소켓과 RMI를 이용하여 구현된 메시지 전달 시스템을 기술한다. 또한 구현된 시스템의 성능을 벤치마크 애플리케이션을 통해서 JPVM과 비교 및 분석을 하고자 한다.

3. JMPI의 설계와 구현

JMPI는 MPJ 스펙을 준수한 메시지 전달 형식의 병렬 프로그램을 위한 라이브러리와 소프트웨어가 통합된 시스템이다. JMPI 라이브러리는 모든 클래스를 순수하게 자바만을 이용해서 구현하였다. 따라서 플랫폼에 대하여 독립적이며 이식성이 좋다는 장점을 갖는다. 이는 JMPI의 가장 큰 특징이라고 할 수 있다.

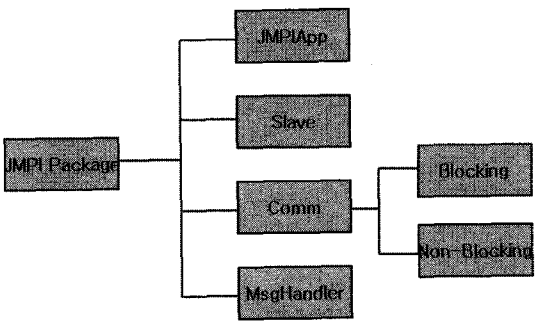
그림 1은 JMPI 시스템의 전체적인 아키텍처를 나타낸다. JMPI 라이브러리에는 프로세스간의 통신을 위한 메서드를 지원하는 클래스, 포트를 관리하는 클래스, 다른 프로세스의 정보를 관리하는 클래스, 데몬과 통신을 담당하는 클래스, 초기화와 소멸자 기능을 하는 메서드를 포함하는 클래스 등으로 구성된다. 두 프로세스 간에 메시지를 전달하기 위해서는 우선 목적지가 되는 프로세스에 연결을 설정한다. 연결이 이루어지기 위해서는 해당 목적지 프로세스에 대한 IP주소와 포트번호에 대한 정보를 알고 있어야 한다. 이는 프로세스를 관리하는 데몬이 자신의 프로세스 정보를 마스터 역할을 하는 하나의 데몬에게 모두 전송한 후 마스터 데몬이 작업에 참여하는 모든 데몬에게 프로세스의 정보를 알려주게 되고 각각의 데몬들은 자신이 관리하는 프로세스에게 모든 프로세스의 정보를 알려주게 된다. 이 정보를 이용하여 다른 프로세스와 연결이 설정될 수 있으며 두 프로세스간의 통신이 정확하게 이루어질 수 있다.

병렬 프로그램을 개발하기 위해 필수적인 JMPI 패키지의 핵심 라이브러리는 4개의 클래스로 구성되어 있다. 그림 2와 같이 JMPIApp 클래스,



(그림 1) 시스템 아키텍처

Comm 클래스, Slave 클래스, MsgHandler 클래스이다. 이 클래스들의 각 기능은 다음과 같다.



(그림 2) JMPI라이브러리의 클래스 구성도

1) JMPIApp 클래스

JMPIApp 클래스는 JMPI를 사용하는 응용프로그램을 작성하기 위해서 초기화와 소멸자 기능을 수행하는 함수를 포함한다. RMI 기반의 메시지 전달시스템에서 응용 프로그램 간에 메시지 전달은 초기화시에 메시지를 전달할 객체를 등록한다.

2) Comm 클래스

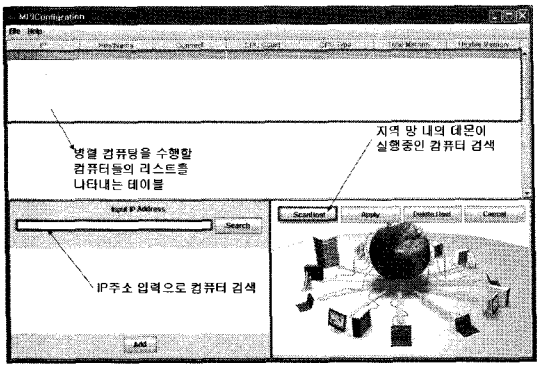
JMPI에서 가장 중요한 클래스로서 프로세스간의 통신을 담당하는 API를 제공하는 클래스이다. 통신방식은 크게 Blocking방식과 Non-Blocking방식 있다.

3) Slave 클래스

병렬 프로그램 구현 시 참여할 프로세스와 프로세스 간에 식별을 위해 사용될 번호와 포트번호를 관리한다.

4) MsgHandler 클래스

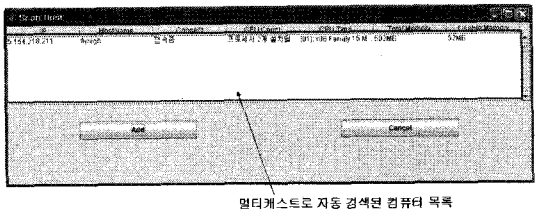
비동기 send/receive 함수에서 메시지를 구분하고 메시지 순서를 관리하는 클래스이다.



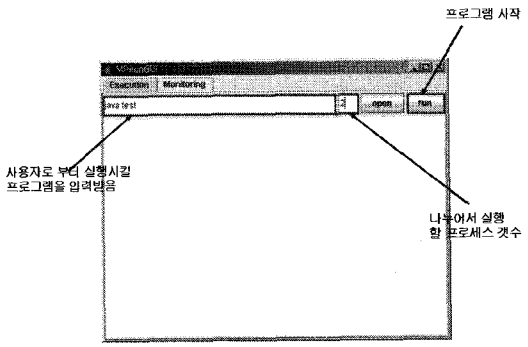
(그림 3) 환경설정 프로그램 초기화면

병렬 컴퓨팅 환경을 조성하는 것은 사용자 입장에서 생각해 보면 까다롭게 느껴질 수 있다. JPVM과 같이 편리한 GUI를 제공하지 않는 시스템에서 병렬 컴퓨팅 환경을 조성하려면 사용법과 명령어가 기술된 설명서가 필요하다. 또한 사용법을 알고 있더라도 많은 컴퓨터들을 일일이 병렬 컴퓨팅 환경을 조성하기 위해 사용자가 명령어를 직접 입력하여 시스템에 등록한다면, 많은 시간이 필요하다. 본 논문에서 제시한 JMPI 시스템에서는 간단한 애플리케이션을 이용하듯이 UI를 이용하여 병렬 컴퓨팅 환경을 조성하고 작업의 진행 상태의 결과와 진행 상태를 모니터링 할 수 있다.

그림 3은 데몬이 실행되는 있다는 가정 하에서 병렬 컴퓨팅 환경을 조성하기 위하여 환경설정 프로그램 실행 시 초기화면이다. 병렬 컴퓨팅 환경을 구성하기 위한 컴퓨터들을 검색하는 방법은 두 가지 방법이 있다. 첫째는 지역 망 내의 컴퓨터를 자동검색을 하기 원한다면 "ScanHost" 버튼을 누르고, 두 번째는 추가하려는 컴퓨터의 IP주소를 직접 입력하여 컴퓨터를 검색하려 한다면 "Input IP Address" 밑에 필드에 컴퓨터의 IP주소를 직접입력한 뒤 "Search" 버튼을 누른다.



(그림 4) 자동 검색 결과 화면

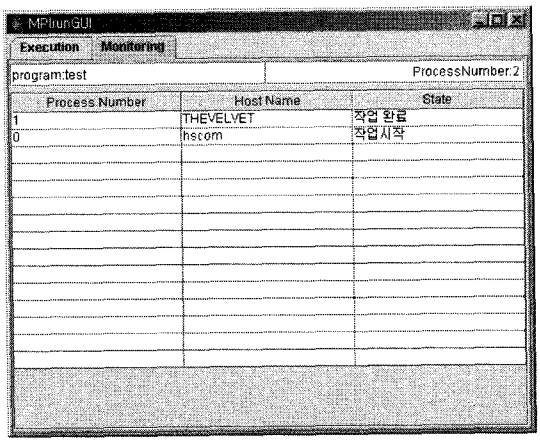


(그림 5) 클라이언트 프로그램 실행 화면

그림 4는 환경설정 프로그램에서 "ScanHost" 버튼을 눌러 지역 망 내의 컴퓨터가 자동 검색된 결과이다. 자동으로 검색된 컴퓨터를 등록하고자 한다면 컴퓨터를 선택한 뒤 "Add" 버튼을 누르면 테이블에 추가된다. 자동검색이나 수동검색으로 검색된 컴퓨터 중 선택한 컴퓨터를 테이블에 추가해 준 뒤 "Apply" 버튼을 누르면 그림 3의 테이블에 있는 컴퓨터들을 DB에 저장한 뒤 현재 등록된 컴퓨터들에게 병렬 컴퓨팅을 수행할 컴퓨터들의 리스트를 전송해 준다. 그림 5는 JMPI 시스템의 클라이언트 GUI 도구로써 실행시킬 프로그램의 이름과 실행할 프로세스의 개수를 입력한 뒤 "run" 버튼을 누르면 작업이 시작된다. 병렬 컴퓨팅을 통하여 완료된 작업의 결과는 밑의 텍스트 영역에 나타나게 된다.

작업의 진행 상태를 모니터링 하는 기능을 그림 6과 같이 클라이언트에 구현하였다. "Process Number"는 분할된 프로세스를 식별하기 위한 식별자이고, "HostName"은 현재 프로세스가 실행되

는 컴퓨터의 이름이며 "State"는 현재 작업의 진행 상태를 나타낸다.

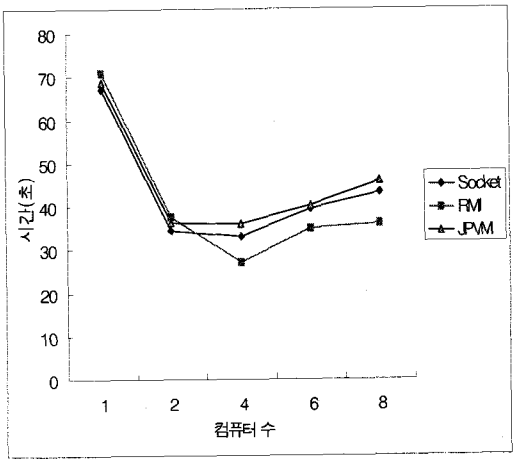


(그림 6) 클라이언트 프로그램의 모니터링 화면

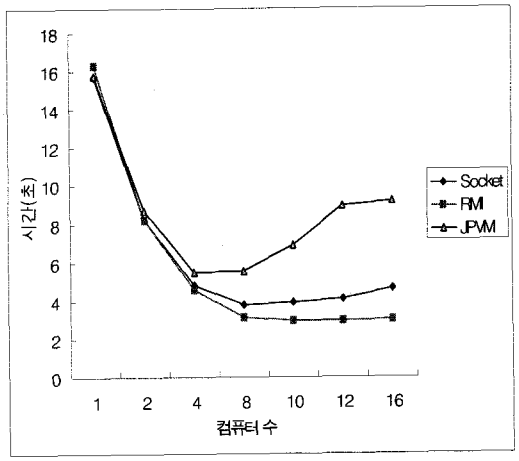
4. 성능 평가

본 절에서 분산 시스템 통신 메커니즘인 소켓 및 RMI를 기반하여 본 논문에서 구현된 시스템의 성능을 JPVM 시스템과 비교하기 위해 각각 특정한 통신 및 컴퓨팅 패턴을 보여주는 3개의 벤치마크 애플리케이션인 파이 값을 계산하는 애플리케이션과 모든 정점 사이의 최단경로를 나타내는 ASP(All-pair Shortest Paths) 애플리케이션, 분산된 라플라스 방정식을 풀기 위한 메서드인 SOR(Red/black Successive Over Relaxation) 애플리케이션을 사용했다. 시스템의 구성은 최대 16개의 노드로 된 클러스터이며, 각 노드는 Windows XP를 OS로 하는 Pentium 4 3.0GHz, 512Mbyte의 메모리로 구성되었고, 네트워크는 100Mb 이더넷으로 연결되어 있다. JMPI는 Jave SE 1.5[9]버전으로 구현되었다. 동일한 환경에서 시스템의 성능평가를 위해 동일한 성능의 컴퓨터에서 해당 애플리케이션에 따라 1대에서 최대 16대로 그 수를 늘려가면서 실험했다.

실험결과 ASP 애플리케이션에서는 1500X1500 노드에서 그림 7과 같이 소켓으로 구현된 JMPI 시스템에서는 4대로 병렬처리를 수행할 때 가장 높은 효율을 보였으며, JPVM시스템과 RMI로 구현된 JMPI 시스템의 경우에도 4대로 병렬 처리를 수행할 때 가장 높은 효율을 보였다.



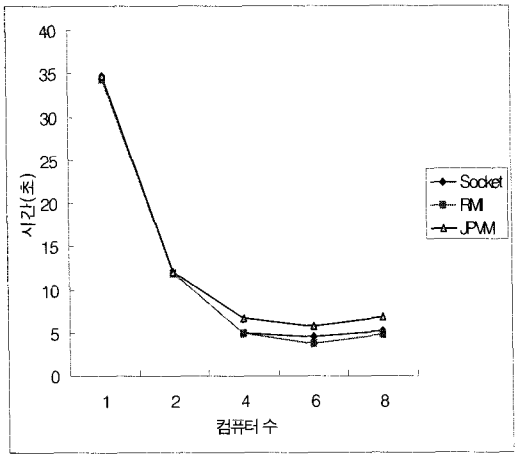
(그림 7) ASP 애플리케이션을 통한 성능 비교



(그림 8) 파이 값 계산 애플리케이션을 통한 성능 비교

파이 값을 계산하는 애플리케이션에서는 그림 8과 같이 소켓으로 구현된 JMPI 시스템에서는 8

대로 병렬처리를 수행할 때 가장 높은 효율을 보였으며, RMI로 구현된 JMPI 시스템에서는 12대로 병렬 처리를 수행할 때 가장 높은 효율을 보였다. 하지만 JPVM에서는 4대 이상으로 병렬처리를 수행할 경우 네트워크 트래픽에 의한 처리 속도의 저하가 JMPI 시스템과 비교하여 크게 나타났다.



(그림 9) SOR 애플리케이션을 통한 시스템의 성능 비교

SOR 애플리케이션에서는 1000X1000의 열과 행을 가진 상황 그림 9와 같이 모든 시스템이 6대의 컴퓨터로 병렬 컴퓨팅 환경을 조성했을 때 가장 효율적인 성능을 나타냈다. 또한 SOR 애플리케이션에서도 JPVM 시스템의 수행시간이 JMPI 시스템에 비교하여 평균적으로 높다.

(표 1) ASP 애플리케이션의 컴퓨터 수에 따른 시스템의 처리 속도

컴퓨터 수	Socket	RMI	JPVM
1	67.013	70.841	68.766
2	34.371	37.671	36.344
4	33.007	27.023	35.922
6	39.428	34.812	40.278
8	43.257	36.024	45.982

(표 2) 파이 값 계산 애플리케이션의 컴퓨터 수에 따른 시스템의 처리 속도 (단위 : 초)

컴퓨터 수	Socket	RMI	JPVM
1	15.719	16.296	15.781
2	8.203	8.187	8.718
4	4.844	4.593	5.469
8	3.828	3.125	5.551
10	3.907	2.984	6.875
12	4.094	2.938	8.941
16	4.657	3.047	9.157

(표 3) SOR 애플리케이션의 컴퓨터 수에 따른 시스템의 처리 속도 (단위 : 초)

컴퓨터 수	Socket	RMI	JPVM
1	34.829	34.310	34.797
2	11.944	12.012	12.016
4	5.012	4.969	6.766
6	4.517	3.739	5.782
8	5.297	4.857	6.912

컴퓨터 1대로 실험을 했을 경우와 가장 높은 성능을 보인 경우의 결과를 비교했을 때, ASP 애플리케이션에서는 표 1과 같이 소켓으로 구현된 JMPI 시스템에서는 병렬처리를 수행할 경우 컴퓨터 1대에 비해 컴퓨터 4대의 처리속도는 2배정도 향상되었고, RMI으로 구현된 JMPI 시스템에서는 병렬처리를 수행할 경우 2.6배정도의 처리속도 향상을 보였다. JPVM 시스템의 경우에는 1대의 컴퓨터에 비해 4대의 컴퓨터를 이용하여 병렬 처리를 수행할 경우 1.9배정도의 처리속도 향상을 보였다. 파이 값 계산 애플리케이션에서는 표 2이와 같이 소켓으로 구현된 JMPI 시스템에서는 컴퓨터 1대에 비해 컴퓨터 8대의 처리속도는 4배정도 향상되었고, RMI으로 구현된 JMPI 시스템에서는 컴퓨터 12대인 경우에 처리속도가 5배정도 향상되었다. JPVM 시스템에서는 컴퓨터 1대에 비해 컴퓨터 4대의 처리속도가 2.8배정도 향상을 보였다.

SOR 애플리케이션에서는 표 3과 같이 소켓으

로 구현된 JMPI 시스템에서는 컴퓨터 1대에 비해 컴퓨터 6대의 처리속도는 7.7배정도 향상되었고, RMI으로 구현된 JMPI 시스템에서는 컴퓨터 6대인 경우에 그 처리속도가 9.1배정도 향상되었다. JPVM 시스템에서는 컴퓨터 6대인 경우에 6배정도의 처리속도 향상을 보였다. 이처럼 컴퓨터의 수를 늘린다고 해서 이에 비례하여 계산이 빠르게 처리되는 것이 아니라 전달되는 메시지의 크기와 네트워크 트래픽을 고려하여 처리하려는 업무에 적절한 컴퓨터 수의 설정도 중요하다는 것이 입증되었다. 실험결과 본 논문에서 제시한 JMPI 시스템이 JPVM 시스템 보다 전체적인 측면에서 높은 성능을 보였으며, 성능 차이는 병렬 처리를 위한 컴퓨터의 수가 늘어남에 따라 차이가 명확하게 나타났다. 이 결과는 병렬 컴퓨터 사이에서 메시지를 송수신하는 경우에 발생하는 트래픽이 JMPI 시스템보다 JPVM 시스템이 더욱 크다는 것이다.

본 논문에서 제시한 JMPI 시스템에서 소켓으로 객체 메시지를 전달할 경우 자바에서 제공하는 객체 스트림을 이용하였고 RMI에서는 자체적으로 제공되는 메커니즘을 이용하여 메시지를 전달하였다. 벤치마크 애플리케이션들의 실험결과를 통하여 컴퓨터의 수가 증가함에 따라 메시지 전송 횟수가 높아질수록 소켓에 부속된 객체 스트림을 사용하여 메시지를 전달하는 것보다 RMI를 사용하여 메시지를 전달하는 것이 보다 효과적이라는 것을 알 수 있다.

5. 결론

본 논문에서는 대표적인 분산 시스템 통신 메커니즘인 소켓 및 RMI를 이용하여 자바 MPI 표준 명세인 MPJ를 따르는 메시지 통신 라이브러리인 JMPI(Java Message Passing Interface)를 각각 설계하고 구현하였다. JMPI는 JVM이 설치된 환경에서 시스템과 애플리케이션의 수정 없이 사용할 수 있으며 간단하지만 매우 편리한 GUI 도구를 제공하기 때문에 시스템 관리자는 손쉽게 JMPI 시스템을 사용할 수 있다. 그리고 본 논문에서 제시한 소켓과 RMI기반의 JMPI 시스템과 JPVM 시스템의 성능 비교는 세 개의 벤치마크 애플리케이션들을 통하여 컴퓨터 대수의 증가에 따른 처리 속도를 비교해 보았다.

본 논문의 실험 결과를 통해 애플리케이션에 따라 컴퓨터의 수를 증가시킨다고 해서 일정하게 처리속도가 증가하는 것이 아니라 최대 효율의 경계치를 넘어간 컴퓨터의 수는 네트워크 트래픽에 의하여 오히려 처리 속도를 저하시키는 것을 알 수 있다. 이것은 컴퓨터의 가장 효율적인 처리 속도는 전달되는 메시지의 크기와 네트워크의 트래픽을 고려하여 컴퓨터의 수를 증가시켰을 때 얻을 수 있다는 것을 알 수 있다. 그리고 병렬 컴퓨팅 환경을 조성하기 위한 시간과 사용자 측면에서 시스템을 이용하는데 필요한 지식을 습득하는데 소요되는 시간을 고려하고, 벤치마크 애플리케이션들의 실험결과를 통하여 병렬 처리를 위해 컴퓨터의 수가 증가할수록 본 논문에서 제시한 JMPI 시스템이 다양한 측면에서 JPVM시스템보다 효과적인 성능을 나타냈다. 또한, 본 논문에서 제시한 소켓과 RMI를 이용하여 구현된 JMPI 시스템의 성능평가를 통해서, JMPI 시스템에서 병렬처리를 위해 컴퓨터의 수가 증가함에 따라 RMI를 사용하여 객체를 포함하고 있는 메시지를 전달하는 것이 소켓에 부속된 객체 스트림을 사용하여 메시지를 전달하는 것보다 평균적으로 효과적이라는 것을 알 수 있다.

현재의 시스템은 프로세스간의 1:1 통신만을 구현하였는데, 향후 연구 과제로 그룹 통신이 가능한 통신 메서드에 관한 클래스를 구현하고 있으며, 다양한 통신 방법과 메시지의 형태를 통해 통합적인 실험을 하고자 한다.

참 고 문 헌

[1] A. M. Agbaria, R. Friedman, "Starfish:

fault-tolerant dynamic mpi programs on clusters of workstations", In Proc. of the High Performance Distributed Computing Symposium, pp. 31-40, Aug. 1999.

[2] M.A. Baker and D.B. Carpenter, "MPJ: A Proposed Java Message Passing API and Environment for High Performance Computing", the 2nd Java Workshop at IPDPS 2000, Cancun, Mexico, LNCS, Springer Verlag, pp. 552-559, May 2000.

[3] M.A. Baker and D.B. Carpenter, G. Fox, S.H. Ko and S. Lim, "mpi-Java: An Object-Oriented Java Interface to MPI", theWorkshop on Java for Parallel and Distributed Computing at IPPS/SPDP 1999, LNCS, Springer Verlag, 1999.

[4] M. Baker, B. Carpenter and A. Shafi, "MPJ: A New Look at MPI for Java", In Proc. of the UK e-Science All Hands Meeting 2005, April 2005.

[5] M. Bornemann, R. V. Nieuwpoort, and T. Kielmann., "MPJ/Ibis: a flexible and efficient message passing platform for Java", In Proceedings of 12th European PVM/MPI Users' Group Meeting, pp. 217-224, Sept. 2005.

[6] A. Bouteiller, F. Cappello, T. Herault, G. Krawezik, P. Lemarinier, and F. Magniette, "MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging", In Proc. of the 15th International Conference on High Performance Networking and Computing(SC2003), November 2003.

[7] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster, "Multi-paradigm Communications in Java for Grid Computing", Communications of the ACM, Vol. 44, No. 10, pp. 118-125, 2001.

[8] W. Gropp, E. Lusk and A. Skjellum, "Using MPI: Portable Parallel Programming with the Message Passing Interface," MIT Press, 1994.

[9] J2SE 1.5.0 API Specification, http://java.sun.com.

[10] G. Judd, M. Clement, Q. Snell, and V. Getov, "Design issues for efficient implementation of mpi in java", In Proc. of the ACM Java Grand Conference, pp. 58-65, 1999.

[11] R. Metkowski and P. Bala, "Parallel Computing in Java: Looking for the Most Effective RMI Implementation for Clusters", Lecture Notes in Computer Science, Springer-Verlag Berlin, Vol. 3911, pp. 272-277, 2006.

[12] S. Mintchev and V. Getov, "Towards portable message passing in Java: Binding MPI", Lecture Notes in Computer Science, Springer-Verlag Berlin, Vol. 1332, pp. 135-142, 1997.

[13] MPI: A Message Passing Interface Standard, Message Passing Interface Forum, November 2003.

[14] C. Nester, M. Phillippsen, and B. Haumacher, "A more efficient RMI for java", In Proc. of the ACM Java Grande Conference, pp. 152-159, June 1999.

[15] M. Philippsen, B. Haumacher, and C. Nester, "More efficient serialization and RMI for Java", Concurrency: Practice and Experience, Vol. 12, No. 7, pp. 495-518, 2000.

[16] G. Stellner, "CoCheck: Checkpointing and process migration for MPI", In Proc. of the Int'l Symposium on Parallel Processing, pp. 526-531, 1996.

[17] Sun Microsystems. RMI specification, available at http://java.sun.com/products/jdk/rmi/.

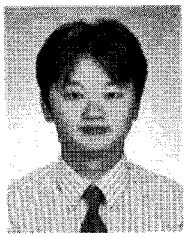
[18] J. T. Rough and A. M. Goscinski, "The

development of an effcient checkpointing facility exploiting operating systems services of the GENESIS cluster operating system", Future Generation Computer Systems, Vol. 20, No. 4, pp 523-538, 2004.

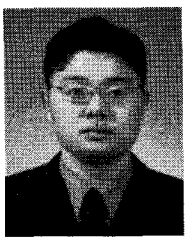
[19] A. Ferrari, JPVM: Network Parallel Cumputing in Java, Technical Report CS-97-29, University of Virginia, 1997.

[20] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. S. Sunderam, "PVM: Parallel Virtual Machine", MIT Press, 1994.

◐ 저 자 소 개 ◑



방 승 준(Seung-Jun Bang)
2006년 경기대학교 전자계산학과 졸업(학사)
2006~현재 경기대학교 대학원 전자계산학과 석사과정
관심분야 : 분산시스템, P2P, 그룹통신, 이동 에이전트 시스템
E-mail : ppangas@hotmail.com



안 진 호(Jin-Ho Ahn)
1997년 고려대학교 컴퓨터학과 졸업(학사)
1999년 고려대학교 대학원 컴퓨터학과 졸업(석사)
2003년 고려대학교 대학원 컴퓨터학과 졸업(박사)
2003~현재 경기대학교 정보과학부 컴퓨터학과 조교수
관심분야 : 분산시스템, P2P, 그룹통신, 이동 에이전트 시스템
E-mail : jhahn@kyonggi.ac.kr