InterCom : 에이전트 기반 인터넷 컴퓨팅 환경 설계 및 구현

김 명 호† · 박 권††

요 약

네트워크와 컴퓨터 기술의 발달로 물리적으로 분산된 컴퓨터를 하나의 자원으로 사용하려는 연구가 많이 진행되고 있다. 일반적으로 이러한 연구들은 메시지 패싱을 기반으로 하는 환경을 개발하는 것이 주류를 이루고 있다. 이러한 환경은 보통 과학계산용 문제를 풀기 위해 많이 사용되고 주어진 문제의 내부병렬성을 이용하여 병렬처리 하게 된다. 따라서 보통 이러한 환경에서는 고도의 병렬성을 얻을 수 있다는 장점이 있는 반면에, 프로그래밍이 어렵고, 사용하기가 어려우며, 분산된 컴퓨터에 사용자의 계정이 있어야 한다는 단점이 있다. 그런데 만일 주어진 문제가 완전히 독립적인 작은 문제로 분할된다면 더욱 효율적인 환경을 만들 수 있다. 이러한 문제 유형은 생물정보학, 3차원 애니메이션, 그래픽스 등에 많이 존재하며, 이를 위한 새로운 환경 개발은 매우 중요한 연구라 할 수 있다. 따라서 본 논문에서는 이러한 문제를 효율적으로 처리하는 프록시 컴퓨팅 기반의 InterCom이라는 새로운 환경을 제안하고, 이를 구현한 것에 대해서 설명한다. 이 환경은 에이전트, 서버, 클라이언트로 구성되어 있다. 이 환경의 장점은 프로그래밍하기가 쉽고, 분산된 모든 컴퓨터에 사용자 계정이 없어도 되며, 분산되는 코드를 자동 컴파일해 줌으로써 사용하기 쉽다는 것이다.

InterCom : Design and Implementation of an Agent-based Internet Computing Environment

Myung Ho Kim† · Kweon Park††

ABSTRACT

Development of network and computer technology results in many studies to use physically distributed computers as a single resource. Generally, these studies have focused on developing environments based on message passing. These environments are mainly used to solve problems for scientific computation and process in parallel using inside parallelism of the given problems. Therefore, these environments provide high parallelism generally, while it is difficult to program and use as well as it is required to have user accounts in the distributed computers. If a given problem is divided into completely independent subproblems, more efficient environment can be provided. We can find these problems in bio-informatics, 3D animation, graphics, and etc., so the development of new environment for these problems can be considered to be very important. Therefore, we suggest new environment called InterCom based on a proxy computing, which can solve these problems efficiently, and explain the implementation of this environment. This environment consists of agent, server, and client. Merits of this environment are easy programming, no need of user accounts in the distributed computers, and easiness by compiling distributed code automatically.

키워드 : 인터넷 컴퓨팅 환경(Internet computing environment), 분산처리 환경(Distributed processing environment), 에이전트(Agent), 프록시 컴퓨팅(Proxy computing)

1. 서 론

컴퓨터가 개발되면서부터 매우 복잡하고 큰 문제를 해결하기 위한 여러 가지 방법들이 연구되어 왔다. 이러한 연구의 결과로 슈퍼컴퓨터나 병렬컴퓨터가 개발되었다. 그러나 문제의 크기가 기하급수적으로 커지면서 단일 시스템으로는 한계가 있음을 알게 되었고, 이에 다른 접근 방법이 필요하게 되었다. 그 결과 네트워크 컴퓨터가 등장하게 되었다. 네트워크 컴퓨터는 로컬네트워크에 연결된 여러 컴퓨터들을 마치 하나의 컴퓨팅 자원처럼 사용할 수 있게 하는 것이다. 이것은 네트워크 속도의 증가로 더욱 그 진가를 발휘하게 되었다. 네트워크 컴퓨터에서 병렬 프로그래밍을 할 수 있게 하는 대표적인 소프트웨어 라이브러리로는 PVM, JaNeC, MPICH, LAM, P4 등을 들 수 있다. 이러한 소프트웨어 라이브러리는 보통 MPI를 기반으로 하고 있다[1-6].

병렬컴퓨터나 네트워크 컴퓨터에서 처리하는 문제는 보통 과학 계산용 문제이다. 이러한 문제들은 문제의 특성상 일반

※ 이 연구는 2000년도 숭실대학교 교내연구비에 의하여 수행되었음.
† 종신회원 : 숭실대학교 컴퓨터학부 교수
†† 정 회 원 : LG캐피탈 IT서비스팀 정보분석파트(DW) 사원
논문접수 : 2001년 1월 16일, 심사완료 : 2001년 8월 10일

적으로 독립된 작은 문제로 분할하여 병렬 처리하는 것이 아니라 문제 자체의 내부 병렬성을 기반으로 병렬 처리하게 된다. 따라서, 여러 프로세서 또는 컴퓨터에 할당되는 태스크(task)들은 독립적으로 자신의 해를 구하는 것이 아니라 다른 여러 프로세서 또는 컴퓨터와의 통신에 의해 중간 결과를 교환하면서 해를 구하게 된다. 즉, 분할된 태스크들은 서로 협력관계에 있다. 이로 인해 대부분의 네트워크 컴퓨터는 메시지 패싱 기반의 환경을 사용하고 있는 것이다.

메시지 패싱 기반의 환경은 다음과 같은 세 가지 문제점을 가지고 있다. 첫째는 병렬 프로그래밍을 위한 API가 복잡하여 프로그래밍하기가 어렵다는 것이다. 또한 보통 병렬 프로그래밍을 하기 위해서는 병렬 프로그램을 작성하는 기술에 대해서도 많이 알고 있어야 한다. 그 다음 단점은 작성된 병렬 프로그램이 수행되기 위해서는 네트워크 컴퓨터에 미리 컴파일 된 실행 파일이 존재해야 한다는 것이다. 이것은 프로그램을 개발할 때에는 매우 성가신 일이다. 마지막으로, 사용자가 프로그램을 수행하기 위해서는 수행될 모든 네트워크 컴퓨터에 계정을 가지고 있어야 한다. 즉, 계정이 없는 컴퓨터를 네트워크 컴퓨터에 포함시켜서 사용할 수는 없다. 이상과 같이 메시지 패싱 기반의 환경을 사용하면 여러 단점이 있음을 알 수 있다. 그러나 만일 분할된 작은 문제들이 완전히 독립적이라면 메시지 패싱 기반 환경보다 더욱 사용하기 쉽고 간편한 환경을 만들 수 있다.

최근 인간의 염색체 구조를 밝히는 게놈 프로젝트가 성공적으로 종결되면서, 이 정보를 분석, 해석하는 생물정보학(Bio-Informatics)에 많은 관심이 집중되고 있다. 그런데, 이 분야는 대량의 컴퓨팅 자원이 절실히 요구되는 분야이다. 특히, 이 분야의 많은 문제들은 독립된 작은 문제로 분할하여 풀 수 있다. 예를 들어, 유전자 지도를 사용하여 신약을 개발하는 Virtual screening 방법은 신약 후보 물질을 찾아내기 위해 주어진 한 단백질과 매우 많은 수의 화학물질간의 상호작용을 계산한다[7]. 즉, 상호작용을 계산하는 하나의 소프트웨어 또는 함수에 10만개 이상의 데이터를 적용하여 신약물질 후보를 생성하게 된다. 이와 같이 독립적인 여러 작은 문제로 분할할 수 있는 문제는 그래픽스, 3차원 애니메이션 등과 같은 분야에 많이 찾을 수 있다. 따라서 이러한 종류의 문제를 풀기 위해 메시지 패싱 기반의 환경보다 더욱 효율적인 환경을 개발하는 것은 매우 중요한 연구 분야라고 할 수 있다[8, 9].

프로그램의 내부 병렬성을 적용하여 병렬 프로그램을 작성하는 것이 아니라 프로그램의 기능별 또는 함수별 병렬성을 사용할 수 있도록 하는 환경으로는 NetSolve와 Ninf가 있다[10, 12]. NetSolve와 Ninf는 과학 계산용 라이브러리를 가지고 있는 분산된 컴퓨터에 수행해야될 함수명과 데이터를 보내고 그 결과를 돌려 받을 수 있도록 하는 환경이다. 따라서 클라이언트에 특정 라이브러리가 없거나 컴퓨팅 파워가 약하더라도 복잡한 문제를 풀 수 있게 한다. 또한 동시에 여러 컴퓨터에 작업을 보냄으로써 함수 단위의 병렬성을 얻을 수 있다. 이들은 함수 단위의 병렬성을 제공하기 때문에 제공되는 주요 API는 메시지 패싱 환경에 비해 매우 적다. 이것은 사용자가 쉽게 사용할 수 있다는 것을 의미한다. 그러나 이들은 기본적으로 과학 계산용 라이브러리의 공유에 관점을 맞추고 있기 때문에 사용자 코드(함수)를 수행시간에 분산하여 처리하기에는 어렵다는 단점이 있다.

따라서, 본 논문에서는 주어진 문제를 많은 수의 독립적인 작은 문제로 분리할 수 있을 때 유용하게 사용할 수 있는 InterCom(Internet Computing)이라는 새로운 환경을 제안한다. InterCom은 인터넷에 연결된 가용 컴퓨터들을 마치 하나의 자원으로 사용할 수 있게 하는 환경이다. 이 환경의 특징은 사용자에게 몇 개의 단순한 API를 제공하고, 분산된 컴퓨터로 보내지는 프로그램을 자동으로 컴파일하여 수행하며, InterCom을 구성하는 각 컴퓨터에 계정이 없어도 프로그램을 수행할 수 있는 환경이다.

본 논문의 구성은 다음과 같다. 2장에서는 InterCom 시스템의 설계에 대해서 설명하고, 3장에서는 InterCom 시스템의 구현에 대해서 설명한다. 그리고 4장에서는 실험을 통해 구현된 환경에 대해서 설명하고, 마지막으로 5장에서 결론을 맺는다.

2. InterCom 시스템 설계

본 장에서는 InterCom의 구조에 대해서 설명한다. 먼저 문제 해결을 위한 컴퓨팅 방법과 모델을 제시하고 InterCom의 요구 사항을 살펴본다. 그리고 내부 설계에 대해 설명한다.

2.1 네트워크 컴퓨팅 방법과 작업 모델

네트워크에 연결된 원격 컴퓨터에서 문제를 해결할 때에는 그 결과를 구하기 위해 필요한 프로그램(또는 코드)의 위치와 그 프로그램의 수행 위치에 따라 세 가지로 분류할 수 있다 : 프록시 컴퓨팅(proxy computing), 코드 이동(code shipping), 원격 컴퓨팅(remote computing)[11].

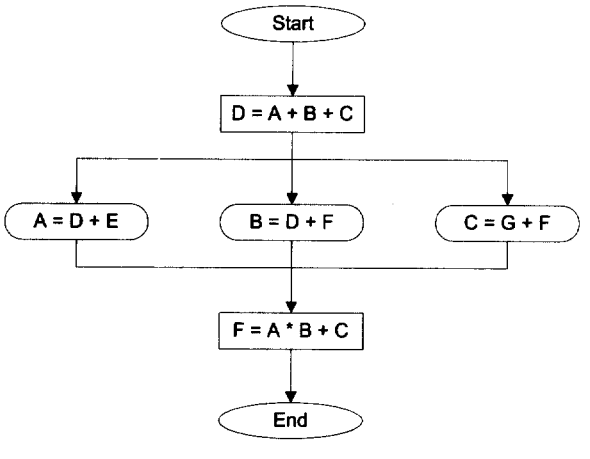
프록시 컴퓨팅은 데이터와 프로그램이 사용자 컴퓨터에 있고 이들을 모두 원격 컴퓨터로 보내어 수행을 한 후 그 결과를 사용자 컴퓨터로 리턴하는 방법이다. 코드 이동은 프로그램이 원격 컴퓨터에 있고 이 프로그램을 사용자 컴퓨터로 다운로드하여 사용자 컴퓨터에서 수행하는 방법이다. 자바 애플릿이 이러한 방법을 사용한다. 원격 컴퓨팅은 프로그램이 원격 컴퓨터에 있고 데이터를 그 원격 컴퓨터로 보내서 수행한 후 결과를 사용자 컴퓨터로 리턴 하는 방법이다. NetSolve와 Ninf는 원격 컴퓨팅을 기반으로 하고 있다[11]. 프록시 컴퓨팅과 원격 컴퓨팅은 원격에서 프로그램을 수행하는 모델이고, 코드 이동은 클라이언트에서 수행

하는 모델이다. 따라서 분산 컴퓨팅에 적용할 수 있는 모델은 프록시 컴퓨팅과 원격 컴퓨팅이다. 그런데 원격 컴퓨팅은 원격 컴퓨터에 이미 등록된 프로그램을 이용하여 문제를 해결하는 방법이므로, 해결할 수 있는 문제가 원격 컴퓨터에 의해 한정된다. 물론, 새로운 프로그램을 등록할 수도 있겠지만, 일반적으로 프로그램을 등록하는 것은 그렇게 쉬운 일이 아니다. 반면, 프록시 컴퓨팅은 프로그램과 데이터를 모두 원격 컴퓨터로 가져가서 수행하기 때문에 다른 방법들 보다 추가적인 통신이 필요하지만, 원격 컴퓨터에 등록된 프로그램의 종류에 영향을 받지 않기 때문에 원격 컴퓨팅 보다 분산 처리에 더 적합한 모델임을 알 수 있다. 따라서 본 논문에서는 데이터와 프로그램을 모두 원격 컴퓨터로 보내 실행 한 후, 결과를 받아 오는 프록시 컴퓨팅 모델을 사용한다.

이러한 프록시 컴퓨팅 방법을 토대로 프로그래머가 쉽게 분산 프로그래밍을 할 수 있는 작업 모델에 대해서 살펴보자. 보통 프로그래머는 주어진 문제를 여러 가지 작업으로 분할할 수 있다. 분할된 작업은 순차적으로 처리되어야 하는 것도 있고, 여러 작업이 동시에 처리 가능한 것도 있다. 예를 들어, 행렬 F를 다음과 같이 계산한다고 가정해 보자 :

```
Compute_F(A, B, C, F)
Begin
    1. D = A + B + C
    2. A = D + E
    3. B = D + F
    4. C = G + F
    5. F = A * B + C
End
```

이 문제를 풀기 위해서는 위의 문장을 차례로 수행하여 풀 수도 있겠지만, 각 문장의 종속 관계를 고려하면 더욱 효율적으로 이 문제를 풀 수 있다. 즉, 문장 1이 계산된 이후에는 문장 2, 3, 4가 동시에 계산될 수 있고, 그 다음 문장 5를 계산하면 될 것이다. 이것을 그림으로 표현하면 (그림 1)과 같다.



(그림 1) Compute_F()의 제어 흐름

프록시 컴퓨팅을 기반으로 하는 환경을 사용할 때에는 주어진 문제를 위와 같이 함수별로 분할하는 것이 필요하다. 그 다음 병렬로 수행될 수 있는 것들은 원격의 컴퓨터들에 작업을 할당하여 동시에 수행할 수 있도록 하고, 순차적으로 수행해야 하는 작업은 그 작업량에 따라 직접 수행하든지 원격 컴퓨터에 수행하도록 해야 한다. 즉, 순차적으로 수행해야될 작업이 간단한 작업이라면 직접 수행하면 되고, 계산량이 많은 작업이라면 자신보다 성능이 좋은 원격 컴퓨터에서 수행하는 것이 좋을 것이다. 따라서, 프록시 컴퓨팅을 기반으로 하는 환경에서는 두 가지 인터페이스를 제공해야 한다. 첫 번째 인터페이스는 순차적으로 수행해야 하면서 컴퓨팅이 많은 부분을 자신의 컴퓨터 보다 더 나은 성능을 가진 원격 컴퓨터에서 실행하기 위한 인터페이스이고, 다른 한가지는 여러 작업을 동시에 처리하기 위해서 원격의 여러 컴퓨터에 분산하여 병렬적으로 실행하기 위한 인터페이스이다. 즉, 프록시 컴퓨팅 기반의 환경에서는 이 두 가지의 인터페이스만 있으면 분산처리가 가능하다.

2.2 프록시 컴퓨팅을 기반으로 하는 InterCom의 요구사항

InterCom은 프록시 컴퓨팅을 기반으로 하는 환경이기 때문에, 사용자가 인터넷에 연결된 컴퓨터에 자신의 코드와 데이터를 보내어 그 결과를 안정적으로 가져올 수 있게 해야 한다. 따라서 다음과 같은 요구사항을 만족해야 한다.

(1) 간편한 사용 : 사용자가 주어진 문제를 쉽게 분산 처리할 수 있도록 하기 위해 사용자에게는 적은 개수의 API만 제공해야 한다.

(2) 사용자 계정 없이 수행 : InterCom의 기본 취지는 인터넷에 연결된 유휴중인 컴퓨터를 사용하여 효율적으로 작업을 수행하는 것이다. 만일 InterCom을 사용하는 사용자에게 모든 컴퓨터에 계정이 있어야 한다는 것을 요구한다면 실용성이 떨어질 것이다. 따라서 InterCom은 사용자 계정이 없어도 수행할 수 있도록 설계되어야 한다.

(3) 원격 자동 컴파일 : 프록시 컴퓨팅은 프로그램 코드와 데이터를 원격 컴퓨터에 업로드해서 수행한 후 그 결과를 가져와야 하는데, 기본적으로 어떤 컴퓨터에서 그 코드가 수행될 지는 알 수 없다. 따라서 업로드되는 컴퓨터에서 수행될 수 있는 실행파일을 만드는 것은 불가능하며, 또한 설령 실행파일을 만들 수 있다고 해도 그 실행파일을 업로드하는 것은 소스 파일을 업로드하는 것에 비해 매우 시간이 많이 소요될 것이다. 따라서 업로드되는 코드를 원격 컴퓨터에서 자동으로 컴파일한 후 수행하는 기능이 필수적이다. 이것은 실행시간에 코드를 컴파일하기 때문에, 프로그램의 실행 시간이 길어진다는 문제점을 안고 있다. 하지만, 다른 환경에서도 특정 코드를 분산처리하기 위해서는 분산된 컴퓨터에서 그 코드를 컴파일해야 한다. 즉, 프로그램의 컴파일 시

점이 프로그램을 수행하기 전인가 아니면 후인가에 차이가 있지, 코드를 컴파일해야 한다는 관점에서는 모든 환경에서 공통적인 시간이 소요될 것이다. 따라서 프로그램을 수행하는 전체 시간은 다른 환경과 같을 것이다.

(4) 자원관리 및 작업 스케줄링 : InterCom은 인터넷상에 존재하는 가용 유휴 자원(컴퓨터)들을 사용하는 것이기 때문에, 이러한 자원들을 관리하고 스케줄링하는 것이 필요하다.

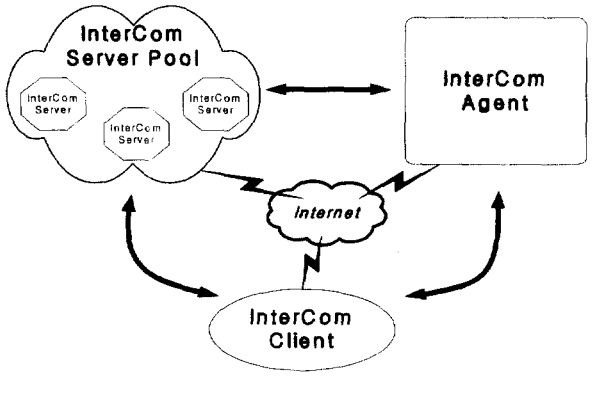
(5) 결함허용 : 결함허용은 분산, 병렬 컴퓨팅 환경에서 매우 중요한 기능이다. 만일 사용자의 작업을 받은 컴퓨터가 다운된다면, 사용자는 데이터를 잃어버리게 되거나 그 상태에서 무한정 기다리게 될 것이다. 따라서 InterCom은 서버에서 수행되는 작업이 도중에 끝나지 않고 투명한 연산이 이루어질 수 있도록 결함허용 기능을 가져야 한다.

2.3 InterCom 시스템 구성

InterCom은 2.2절에서 설명한 요구사항을 만족시키기 위해 다음과 같은 세 가지 구성요소로 구성된다 : 클라이언트, 서버, 에이전트. 클라이언트는 사용자에게 사용하기 쉬운 인터페이스를 제공하고, 작업을 서버에게 넘겨주며, 넘겨준 작업에 대한 결함허용 기능을 가진다. 서버는 클라이언트로부터 수행할 작업을 받아서 자동 컴파일하고 실행하여 결과를 클라이언트에게 돌려주는 기능과 클라이언트 작업에 대한 결함허용 기능을 가진다. 에이전트는 서버들을 관리하고 클라이언트가 요청한 작업을 스케줄링 하는 기능을 가진다.

InterCom 시스템의 개략적인 전체 구조는 (그림 2)와 같다. 각 구성 요소는 인터넷상에 존재하고, 서로간에 메시지를 교환하면서 클라이언트의 작업을 수행한다. 각각의 구성 요소가 어떤 관계를 가지는지 간략히 살펴보면 다음과 같다 :

1) 클라이언트는 에이전트에게 서버를 요청하고, 서버에게 실행할 작업을 넘겨준다.
2) 서버는 에이전트에 자신을 등록하고, 클라이언트에서 요청한 작업을 수행한다.
3) 에이전트는 서버들을 관리하고, 사용 가능한 서버리스트를 클라이언트에게 전송한다.



(그림 2) InterCom 시스템 구조

이제 InterCom의 각 구성 요소를 자세히 살펴보자.

2.3.1 InterCom 에이전트

InterCom 에이전트는 인터넷상에 존재하는 서버들을 등록, 관리하고 클라이언트로부터 요청이 왔을 때 최상의 서버를 선택해 주는 작업 스케줄러 역할을 한다. 이와 같은 작업을 위해서는 두 가지 데몬이 필요하다.

첫 번째는 서버들을 등록하고 관리하는 데몬이다. 등록된 서버에서 실제 클라이언트 작업이 수행되므로, 효율적인 작업 분배가 필요하다. 에이전트에서 로드 밸런싱(load balancing)을 위한 정책으로서 각 서버들의 로드 부하 정보들을 검사하여 클라이언트 작업을 분배하도록 한다. 서버에서 보낸 시스템 로드 값을 가지고 모든 서버 리스트들을 성능이 좋은 순서대로 정렬한다.

두 번째는 클라이언트의 요청을 처리하는 데몬이다. 클라이언트로부터 요청이 오면 바로 서버 리스트 정보를 클라이언트 데몬에게 모두 전달한다. 등록된 모든 서버 리스트들을 클라이언트에게 전송함으로써 클라이언트가 최상의 서버에 접속하지 못하는 경우가 발생하더라도 그 다음 서버에 접속할 수 있도록 한다.

2.3.2 InterCom 서버

InterCom 서버는 실제 클라이언트의 작업이 수행되는 곳이다. 먼저 에이전트가 서버들을 관리할 수 있도록 서버에서는 자신의 시스템 정보를 에이전트에게 주기적으로 전송해야 한다. 그리고, 클라이언트의 요청이 오면 문제를 해결하여 클라이언트에게 결과를 전송한다. 서버에서 작업이 진행되고 있는 동안 서버의 결함 여부를 검사하기 위해 클라이언트와 메시지 교환을 한다. 이를 위해 서버는 세 가지 데몬으로 구성된다.

첫 번째는 에이전트에게 서버 자신의 정보를 주기적으로 전송하는 서버 로드 데몬이다. 서버 로드 데몬은 로드 정보를 수집하여 주기적으로 에이전트에게 자신의 정보를 보낸다. 에이전트의 작업 분배 및 서버 정렬을 위해 수집되는 로드 정보로는 대기 중인 작업 수, 메모리 사용량, CPU의 사용량, 디스크의 사용량 등의 정보를 사용한다.

두 번째는 클라이언트의 작업을 수행하는 서버 데몬이다. 서버 데몬은 클라이언트에서 전달받은 코드와 데이터 파일을 가지고 적절한 파싱 작업을 거쳐 자동 컴파일하고 실행한다. 그리고, 수행된 결과를 클라이언트에게 전달한다.

세 번째는 서버의 결함 여부를 탐지하기 위한 결함허용 데몬이다. 이는 클라이언트 작업의 결함허용을 제공하기 위해 사용된다. 서버는 작업을 수행 중일 때 클라이언트의 결함허용 데몬에게 수행되는 상태 정보를 넘겨준다. 이렇게 하여 서버의 결함을 클라이언트의 결함허용 데몬에서 탐지할 수 있게 한다.

2.3.3 InterCom 클라이언트

InterCom 클라이언트는 사용자에게 분산 컴퓨팅 환경을 제공해 주는 부분이다. 클라이언트는 세 부분으로 나눌 수 있는데, 사용자에게 인터페이스를 제공해 주는 라이브러리 부분과 클라이언트의 요청을 처리하는 클라이언트 데몬 부분, 그리고 서버의 결함 여부를 탐지하기 위한 부분으로 나눌 수 있다.

첫 번째는 사용자에게 인터페이스를 제공하기 위한 클라이언트 인터페이스 라이브러리이다. 이는 2.1절과 2.2절에서 살펴본 바와 같이 다음과 같은 것들로 구성된다. 만일 순차적으로 수행해야 하는 경우에 컴퓨팅을 많이 요구하는 작업이라면, 자신의 컴퓨터보다 더 성능이 뛰어난 컴퓨터에서 수행하여 빠른 결과를 얻을 수 있다. 또한, 병렬적으로 수행해도 되는 작업들은 여러 컴퓨터에 작업을 분산하여 해결하면 훨씬 더 좋은 결과를 얻을 수 있다. 따라서 작업의 특성에 따라 호출 방법을 달리 하기 위해 두 가지 인터페이스를 제공한다 : 블러킹(blokcing) 호출, 논블러킹(non-blocking) 호출. 블러킹 호출이라 함은 요청한 작업이 서버에서 마치고 결과 값이 돌아올 때까지 기다리는 호출 방식을 말하고, 논블러킹 호출은 요청 작업의 결과를 받을 때까지 기다리지 않고 바로 사용자 코드로 리턴하여 클라이언트의 다음 코드를 수행할 수 있는 호출을 말한다. 물론 논블러킹 호출을 사용할 때에는 나중에 그 결과가 나왔는지를 검사하는 것이 필요할 것이다.

각 인터페이스는 사용하기 쉽게 하기 위해서 C의 printf() 함수 형식으로 인터페이스를 설계한다. 각 인터페이스의 첫 번째 인자는 함수 이름이고, 두 번째 인자는 그 함수의 인자들에 대한 형식 지정자이며, 세 번째 인자는 그 함수의 실제 인자로 사용되는 변수들이다. 다음 <표 1>은 InterCom 인터페이스의 프로토타입이다.

〈표 1〉 클라이언트 인터페이스 프로토타입

① 블러킹 인터페이스
- void InterCom(char *task_name, char *arg_format, *(arg_list)*) ;

② 논블러킹 인터페이스
- int InterCom_nb(char *task_name, char *arg_format, *(arg_list)*) ;
- void InterCom_nb_ok(char *type, int rid) ;

블러킹 호출인 InterCom() 인터페이스를 호출하면 문제와 데이터를 서버로 보내고 결과가 도착할 때까지 기다린다. 반면 논블러킹 호출 인터페이스는 즉시 요청 ID(request ID)를 반환하고 클라이언트로 리턴 한다. 따라서 논블러킹 호출을 할 때에는 나중에 결과를 검사하거나, 결과를 받아오는 인터페이스가 하나 더 필요하다 : InterCom_nb_ok(). 인터페이스의 인자로 사용되는 변수들은 입력 변수와 출력 변수로 구분되며, 각각 I(input)와 O(output)로 명시된다. 입력 변수는 그 함수를 수행할 때 입력 값으로 주어지는 변수이며, 출력 변수는 나중에 결과 데이터를 받을 변수이다. 배열 변수는 형식 지정자 사이에 배열 크기를 표기하도록 한다. 다음 <표 2>는 데이터 타입에 따른 입출력 변수들의 형식 지정자에 대한 것이다.

〈표 2〉 입출력 변수들의 형식 지정자

데이터 타입	단일 변수 형태	
	입력 변수(I)	출력 변수(O)
char	%Ic	%Oc
string (char *)	%Is	%Os
unsigned short	%Ius	%Ous
unsigned int	%Iu	%Ou
int	%Id	%Od
unsigned long	%Iux	%Oux
long	%Ild	%Old
float	%If	%Of
double	%Ifd	%Ofd
long double	%Ilf	%Olf
file pointer(read, write)	%IF	%OF
file pointer(append)	%IA	%OA
배열 변수 사용 형태 => %{I/O}[size]{c/d/…} 예) %I50d : int 배열 50 크기를 갖는 입력변수		

두 번째는 클라이언트의 요청을 처리하는 클라이언트 데몬이다. 작업의 효율성을 위해서 클라이언트 인터페이스와 클라이언트 데몬 서로간에 메시지를 교환하면서 클라이언트의 작업이 처리된다. 클라이언트 인터페이스로부터 함수의 이름과 호출 모드를 받으면, 클라이언트 데몬은 에이전트에 접속하여 서버 리스트를 받아 온다. 그 다음, 서버 리스트의 첫 번째 서버에 접속하여 문제를 해결하고 결과를 사용자 코드로 넘겨준다. 만일 첫 번째 서버 접속에 실패했다면, 그 다음 서버에 접속하여 처리한다. 이러한 과정은 서버 리스트의 마지막 서버를 만날 때까지 계속된다.

세 번째, 서버의 결함 여부를 탐지하기 위한 결함허용 데몬이다. 작업이 서버에게 잘 전달되었다고 하더라도, 클라이언트는 그 서버로부터 결과를 반드시 받을 것이라는 것을 보장받지 못한다. 즉, 그 서버가 다운될 경우 사용자는 결과 데이터를 받지 못하고 계속 기다리게 될 수도 있다. 따라서 서버측 결함허용 데몬으로부터 주기적으로 수행 상태 정보를 전달받아 서버의 결함 여부를 판단해야 한다. 만일 서버에 결함이 생기면 다른 서버에서 클라이언트의 작업이 수행되도록 처리한다.

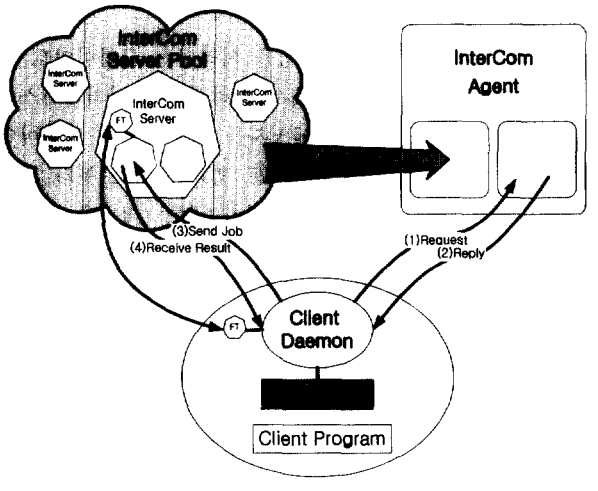
3. InterCom 시스템 구현

C 언어는 거의 모든 운영체제가 지원하고 있고, 운영체제가 바뀌더라도 약간의 수정만으로 프로그램을 이식할 수 있으며, 다른 언어에 비해 수행 속도가 빠르다는 장점을 가지고 있다. 따라서, InterCom은 C로 구현한다. 이제 InterCom의 동작을 자세히 살펴보고, 이를 위해 InterCom의 각 구성 요소들을 어떻게 구현했는지를 자세히 설명한다.

3.1 InterCom 내부 동작

InterCom은 2장에 언급한 것과 같이 클라이언트, 서버, 에이전트로 구성된다. (그림 3)은 InterCom 시스템의 내부 동작을 상세히 나타낸 것이다. 먼저 인터넷 자원(서버)들은 에이전트에 등록이 된 상태에서 클라이언트가 분산 처리 작업을 원할 때, 다음과 같은 순서로 구동이 된다 :

1) 클라이언트 데몬은 에이전트에게 서버 리스트를 요청한다.
2) 에이전트는 클라이언트 데몬에게 서버 리스트를 전송한다.
3) 클라이언트 데몬은 선택된 서버로 접속하여 코드와 데이터를 전송한다.
4) 서버는 전송 받은 정보를 가지고 자동 컴파일하고 실행하여 그 결과를 클라이언트 데몬으로 전송한다. (작업이 실행중인 동안 서버와 클라이언트간에 결함허용 데몬(FT)이 구동되어 서버의 결함을 검사한다.)



(그림 3) InterCom 내부 동작도

이제 InterCom의 구성 요소들의 세부 구현사항을 자세히 설명한다.

3.2 에이전트 구현

InterCom 에이전트는 인터넷상에 있는 서버들을 등록, 관리를 하고, 클라이언트의 요청이 왔을 때 최상의 서버를 선택해 주는 작업 스케줄러 역할을 한다. 이를 위해 에이전트는 클라이언트의 작업을 처리할 ProcessingForServer 데몬과 서버와 통신하기 위한 ProcessingForClient 데몬으로 구성된다.

3.2.1 ProcessingForServer 데몬

인터넷상에 존재하는 가용 서버들을 등록, 관리 해주는 데몬이다. 서버들은 각자 자신의 로드 정보를 주기적으로 보낸다. 이러한 정보를 받으면 로드 상태 순으로 정렬하여 서버 리스트를 유지한다. 이렇게 서버 리스트를 관리하는 목적은 클라이언트로부터 요청이 들어왔을 때 등록된 서버들 중 최상의 서버를 할당하기 위해서이다. 항상 최신의 정보로 서버 리스트가 정렬되기 때문에 등록된 서버들은 자신의 로드 정도에 따라 작업을 분배받게 된다.

3.2.2 ProcessingForClient 데몬

클라이언트의 요청이 들어오면 문제를 해결할 수 있는 서버 리스트를 전송하는 데몬이다. 서버 리스트를 클라이언트 데몬에게 전송함으로써, 첫 번째 등록되어 있는 최상의 서버가 갑자기 다운되거나 접속 할 수 없을 때, 그 다음 서버로 접속 할 수 있도록 한다.

3.3 서버 구현

서버는 클라이언트의 요청을 실제로 해결하는 곳으로써, 클라이언트의 작업을 처리 해주는 부분인 서버 데몬과 에이전트와 메시지를 주고받으며 자신의 로드 정보를 전달해주는 서버 로드 데몬, 그리고 클라이언트 작업의 결함허용을 제공하기 위한 결함허용 데몬으로 구성된다.

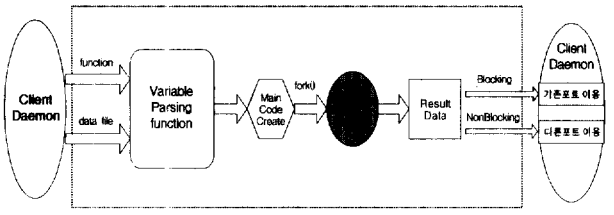
3.3.1 서버 로드 데몬

서버 로드 데몬은 에이전트와 통신하는 부분으로 처음 서버가 구동될 때 에이전트에 등록한다. 그 후, 주기적으로 자신의 로드 정보를 검사하여 자동으로 로드 정보를 전송하는 역할을 한다. 서버 로드 데몬은 주기적 간격으로 자신의 로드 정보를 검사하는 LocalLoadCheck 데몬과, 에이전트에게 로드 부하 정보를 전송하는 RequestForAgent 데몬으로 구성된다. 서버의 시스템 로드 정보로는 대기 중인 작업 수, 메모리 사용량, CPU의 사용량, 디스크의 사용량으로 계산된 수치 값이 사용된다. 본 논문에서는 CPU 사용률에 대해서 30%, 디스크 사용률에 대해서는 50%, 메모리 사용량에 대해서는 20%의 가중치를 주어 계산하였다. 그리고, 서버 로드 값이 같을 때에는 현재 대기 중인 작업수가 적은 서버가 먼저 선택되도록 하였다. 계산된 로드 값은 자신의 서버 주소(IP 주소)와 함께 에이전트에게 보내진다. 이 때, 이 값이 이전에 구한 로드 값의 임계치(±10%) 이내에 있다면 그 값을 보내지 않는다. 이는 자주 서버 로드 정보를 보내지 않음으로써, 네트워크 통신 오버헤드를 줄이고자 한 것이다.

3.3.2 서버 데몬

서버 데몬은 클라이언트로부터 전달된 정보(코드와 변수)를 가지고 실제 작업을 처리하는 부분이다. 전달된 정보는 데이터 파일에 있게 된다. (그림 4)는 서버 데몬의 동작을 나타낸 것이다.

먼저 클라이언트에서 받은 데이터 파일을 파싱하고, 실행될 수 있는 프로그램을 자동으로 생성하는 VariableParsing 루틴을 수행하여 main() 함수를 생성한다. 그리고 생성된 프



(그림 4) 서버 데몬의 내부 동작

로그램을 컴파일하고 실행한다. 실행 후 결과가 파일로 저장되고, 생성된 결과 데이터 파일을 클라이언트 데몬에게 전달한다. 결과를 전달 할 때, 블러킹 호출 일 때는 클라이언트가 결과 파일을 받을 때까지 기다리고 있으므로 이미 설정되어 있는 포트를 통해 결과를 보내면 된다. 하지만 논블러킹 함수 호출일 때에는 클라이언트가 기다리지 않기 때문에, 포트를 개설한 후 결과 파일을 전송한다. 그래서, 클라이언트 데몬에서는 논블러킹 결과 파일을 받기 위한 데몬(RequestForNonBlock)이 별도로 존재한다.

3.3.3 서버 결함허용 데몬

클라이언트의 작업에 대해서 결함허용을 제공해 주기 위한 데몬이다. 이는 클라이언트측의 결함허용 데몬에게 주기적으로 수행 상태 정보를 전송하여 클라이언트가 결함을 탐지하도록 한다. 서버에서 클라이언트의 작업이 진행되는 동안 구동된다.

3.4 클라이언트 구현

클라이언트는 사용자에게 블러킹과 논블러킹 API를 제공해 주는 부분이다. 인터페이스를 제공하기 위한 클라이언트 인터페이스 부분과 실제 에이전트와 서버간에 메시지 교환을 담당하는 클라이언트 데몬, 그리고 클라이언트 작업의 결함허용을 제공하기 위한 결함허용 데몬으로 구성된다.

3.4.1 클라이언트 인터페이스

사용자에게 인터페이스를 제공해 주고 수행 코드의 데이터 파일을 생성하고 나중에 결과를 받으면, 결과 데이터 값을 파싱하여 출력 변수에 데이터 값을 복원하는 역할을 한다. InterCom 인터페이스의 형태는 printf() 함수의 형식과 유사하게 구성하였고, 변수의 형식 지정자는 설계 시에 제시했던 형식으로 구성하였다. 현재 지원되는 데이터 타입은 C 언어의 기본 데이터 타입 모두를 지원하고, 파일 포인터 타입까지 지원한다.

클라이언트가 요청한 작업은 함수 단위로 서버에 보내지기 때문에, 클라이언트는 서버에서 수행할 함수명을 블러킹과 논블러킹 호출에서 명시한다. 그러면 클라이언트는 그 함수의 코드와 필요한 변수들을 서버로 보내게 된다. 여기서 효율적인 구현을 위해 프로그래밍할 때 약간의 제약을 주었다. 즉, 프로그래머는 서버로 보낼 함수를 그 함수명과 같은 이름을 갖는 파일에 따로 저장해 두어야 한다. 따라서, 클라이언트는 블러킹과 논블러킹 호출이 일어날 때, 그 인터페이스에 명시된 함수명을 보고 그 함수 코드를 저장한 파일을 찾아서, 그 파일의 내용과 필요한 변수들의 값을 서버에 보내게 된다.

3.4.1.1 블러킹 호출 인터페이스

블러킹 함수 호출은 결과 데이터가 리턴되어 오기까지 클라이언트 인터페이스 부분에서 기다렸다가 다음 코드를 수행한다. 다음 <표 3>은 블러킹 호출 인터페이스의 프로토타입이다.

〈표 3〉 블러킹 호출 인터페이스

```
void InterCom(char *task_name, char *arg_format, … ) ;
 o task_name : 원격으로 실행할 함수 이름
 o arg_format : task_name 함수의 인자 리스트의 형식지정자
 o … : task_name 함수의 인자 리스트
```

3.4.1.2 논블러킹 호출 인터페이스

논블러킹 인터페이스는 블러킹 인터페이스와 거의 유사하지만, 요청에 대한 결과가 나오기를 기다리지 않고 요청 ID 값을 리턴 값으로 하여 바로 클라이언트 코드로 리턴한다. 이 ID를 가지고 사용자는 나중에 요청에 대한 결과를 검색하거나 결과가 도착할 때까지 기다릴 수 있다. 이를 위해 논블러킹 함수 호출 이후에는 InterCom_nb_ok() 함수를 사용하여 결과를 받아 온다. 논블러킹 인터페이스를 사용하여 사용자는 작업을 병렬적으로 수행할 수 있다.

다음 <표 4>는 논블러킹 호출 인터페이스의 프로토타입이다.

〈표 4〉 논블러킹 호출 인터페이스

```
int InterCom_nb(char *task_name, char *arg_format, … ) ;
 o 리턴 값 : 요청 ID 값
 o task_name : 원격으로 실행할 함수 이름
 o arg_format : task_name 함수의 인자 리스트의 형식지정자
 o … : task_name 함수의 인자 리스트

int InterCom_nb_ok(char *type, int rid) ;
 o 리턴 값 : type이 probe일 때 결과 도착 여부에 따라 0 또는 1
 o type : probe 또는 receive
    - probe : 요청 ID가 rid인 요청의 결과가 도착했는지 여부
              를 검사함
    - receive : 요청 ID가 rid인 요청의 결과가 올 때까지 계속
                기다리고 있다가 결과를 받아 오는 기능
 o rid : 요청 ID
```

3.4.2 클라이언트 데몬

클라이언트 인터페이스를 통해 호출이 되면, 에이전트에게서 서버 리스트를 받아 오는 작업과 해당 서버에 클라이언트 작업을 전송하는 모든 작업이 클라이언트 데몬에서 수행된다. 클라이언트 데몬은 클라이언트 인터페이스의 작업 요청을 처리하는 RequestForClient 데몬과 논블러킹 호

출시 서버로부터 결과 데이터 파일을 받는 RequestFor-NonBlock 데몬으로 구성된다.

사용자가 InterCom 인터페이스로 호출하면 수행 작업(함수) 이름을 호출 모드와 함께 클라이언트 데몬에게 알려준다. 그러면, 클라이언트 데몬에서는 에이전트에게 서버 리스트를 요청하여 받아오고, 서버 리스트의 첫 번째 서버에게 작업을 전송한다. 이때, 블러킹 모드일 경우에는 결과 데이터가 도착할 때까지 기다리게 되고, 논블러킹 모드 일 때는 해당 요청 ID 값을 리턴하고 바로 클라이언트의 다음 코드를 수행하게 된다. 결과 데이터가 클라이언트 데몬에게 전송되면, 블러킹 모드 일 경우에는 결과 도착을 알려주고 출력 변수에 결과 데이터를 복사한다. 그리고, 논블러킹 모드 일 경우에는 RequestForNonBlock 데몬이 서버로부터 결과 데이터를 받고, 해당 요청 ID 값에 대해 결과가 도착했다는 것을 특정 플래그를 설정하여 표시한다. 사용자는 InterCom_nb_ok() 인터페이스를 이용하여 해당 요청 ID 값에 대한 결과를 검사하거나 결과 데이터가 도착할 때까지 기다려서 결과 데이터를 가져온다. 이때 논블러킹의 결과를 받는 작업의 동기화를 위해 세마포어 연산을 사용하였다.

3.4.3 클라이언트 결함허용 데몬

클라이언트의 작업에 대해서 결함허용을 제공하기 위한 것이며, 서버측 결함허용 데몬으로부터 주기적으로 수행 상태 정보를 받는다. 만일 서버 측에서 일정 시간 상태 정보를 보내지 않으면 클라이언트에서는 서버가 다운되었다고 판단하고 클라이언트의 작업을 새로운 서버에게로 전달한다. 에이전트에서 받은 서버 리스트에 접속할 서버가 없을 때까지 이런 작업이 계속된다.

4. 실 험

이 장에서는 3장에서 구현한 InterCom 시스템을 사용하여 실제로 C 프로그램을 작성하고 실험해 본다. InterCom은 주어진 함수와 데이터를 원격 컴퓨터에 업로드하고 원격 컴퓨터가 자동으로 그 함수를 컴파일하고 실행한 후 결과를 주는 모델이기 때문에, InterCom에서 효율적으로 수행될 수 있는 문제의 유형은 필요한 데이터 량은 작으면서 계산 시간이 많이 요구되는 문제이다. 이러한 문제는 1장에서 언급한 것과 같이 생물정보학이나 애니메이션 분야에 많이 존재한다. 그러나 본 장에서는 여기서 구현한 시스템이 잘 동작하는지 만을 실험하는 것을 목적으로 하기 때문에 식 (1)과 같은 단순한 행렬을 계산하는 프로그램을 작성한다 :

$$A = (B + C) * (D + E) \quad (1)$$

여기서 A, B, C, D, E는 모두 1000x1000의 행렬이다. 이 행렬 수식을 InterCom을 사용하여 계산하기 위해서는 이 식의 병렬성을 구해야 한다. 그런데, 이 식은 직관적으로 다음과 같이 작성할 수 있다는 것을 알 수 있고, 식 (2)와 (3)은 병렬로 계산할 수 있다는 것을 알 수 있다 :

$$F = (B + C) \quad (2)$$

$$G = (D + E) \quad (3)$$

$$A = (F * G) \quad (4)$$

즉, 식 (2)와 식 (3)의 수식을 동시에 수행하여 결과를 구한 후 그 결과를 식 (4)와 같이 곱하면 우리가 원하는 결과를 얻을 수 있을 것이다. 이제 이것을 InterCom을 사용하는 프로그램으로 작성해 보자. 먼저 해야할 일은 식 (2), 식 (3), 식 (4)를 InterCom 서버에서 수행하기 위해 식 (2), 식 (3), 식 (4)를 위한 함수를 다른 파일에 작성해야 한다. 그 다음, 프로그램 상에서 InterCom API를 사용하여 식 (2)와 식 (3)을 InterCom 서버에서 수행하도록 해야 한다. 그런데, 식 (2)와 식 (3)은 동시에 수행될 수 있지만, 식 (4)가 실행되기 위해서는 식 (2)와 식 (3)의 결과가 있어야 됨을 알 수 있다. 따라서 식 (2)와 식 (3)은 논블러킹 호출인 InterCom_nb()을 사용하여 동시에 수행하도록 하고, 식 (4)가 수행되기 전에 그 결과가 왔는지를 알아내는 InterCom_nb_ok()를 사용하면 된다. 또한 식 (4)도 시간이 많이 요하는 부분이기 때문에 성능이 더 좋은 서버에서 수행하는 것이 더 좋을 것이다. 따라서 이 수식 하나만 InterCom 서버로 보내서 수행하면 되기 때문에, 블러킹 호출인 InterCom()을 사용하여 요청하면 된다.

이제 InterCom을 사용하는 프로그램은 (그림 5)와 같이 작성할 수 있다 :

```
void matrixadd(int A[], int B[], int C[])
{
    int i;

    for ( i = 0; i < 1000 * 1000; i++)
      C[i] = *(A + i) + *(B + i);
}
```

(가) matrixadd.c 파일

```
void matrixmul(int A[], int B[], int C[])
{
   int i, j;
   for (i = 0; i < 1000 * 1000; i++)
     for (j = 0; j < 1000 * 1000; j += 1000)
       C[i] += *(A + i) * *(B + j);
}
```

(나) matrixmul.c 파일

```
#include "intercom.h"
main()
{
   int *A, *B, *C, *D, *E, *F, *G;
   int rid1, rid2, i;
```

```
    A = (int *) calloc(1000 * 1000, sizeof(int));
    B = (int *) calloc(1000 * 1000, sizeof(int));
    C = (int *) calloc(1000 * 1000, sizeof(int));
    D = (int *) calloc(1000 * 1000, sizeof(int));
    E = (int *) calloc(1000 * 1000, sizeof(int));
    F = (int *) calloc(1000 * 1000, sizeof(int));
    G = (int *) calloc(1000 * 1000, sizeof(int));

    /* B, C, D, E 배열 초기화 */
    for (i = 1; i < 1000 * 1000; i++)
      B[i] = C[i] = D[i] = E[i] = 1;

    rid1 = InterCom_nb("matrixadd",
                       "%I1000000d %I1000000d %O1000000d",
                       B, C, F);
    rid2 = InterCom_nb("matrixadd",
                       "%I1000000d %I1000000d %O1000000d",
                       D, E, G);

    InterCom_nb_ok("receive", rid1);
    InterCom_nb_ok("receive", rid2);
    InterCom("matrixmul",
             "%I1000000d %I1000000d %O1000000d",
             F, G, A);

    for (i = 0; i < 1000 * 1000; i++){
      if (i % 1000 == 0) putchar('\n');
        printf("%d ", A[i]);
    }
}
```

(다) matrixmain.c 파일

(그림 5) 수식 (1)을 위한 InterCom 예제 프로그램

이 프로그램을 좀더 자세히 살펴보자. 입력 변수는 B, C, D, E 배열이고 출력 변수는 F, G, A 배열이다. 따라서 변수의 표기를 입력 변수일 때에는 I 모드를 사용하고, 출력 변수일 때는 O 모드를 사용하였다. 두 개의 논블러킹 호출은 에이전트에 등록된 서버에서 matrixadd() 함수를 동시에 수행하게 한다. 그리고 matrixmul()을 수행하기 전에, matrixadd()의 실행 결과를 받아 오기 위해서 논블러킹 호출 인터페이스의 리턴 값인 요청 ID 값(rid1, rid2)과 "receive" 모드로 InterCom_nb_ok()를 호출하여 결과를 받아 온다. 받아 온 결과를 가지고 블러킹 호출인 InterCom()을 사용하여 최종적으로 곱셈 연산을 수행하면 모든 행렬 계산이 끝나게 된다. 이렇게 InterCom의 블러킹 인터페이스와 논블러킹 인터페이스를 조합하여 문제를 해결하면 작업을 분산하여 처리할 수 있다.

다음 (그림 6)은 클라이언트에서 위 프로그램을 수행했을 때의 결과이다 :

```
###### Global rid ###### [0]
###### Global rid ###### [1]
InterCom_nb_ok 's Function Name Check[matrixadd]
RequestID[0]
Probe check : : Not yet
Receive_Check RequestID[0]
Received check : : RECEIVED OK OK

func_out : matrixadd.out
RECEIVED RID[0] RECEIVE 'S OK RETURN
InterCom_nb_ok 's Function Name Check [matrixadd]
RequestID[1]
Probe check : : OK received okok

func_out : matrixadd.out
RECEIVED RID[1] PROBE'S OK RETURN
function name : matrixmul :
function outfile : ./matrixmul.out :
ArgFormatParsing end
ok recieve out file !!
func_out : ./matrixmul.out

4000000 4000000 4000000 4000000 4000000 4000000 4000000 4000000
4000000 4000000 4000000 4000000 4000000 4000000 4000000 4000000
4000000 4000000 4000000 4000000 4000000 4000000 4000000 4000000
.....
```

(그림 6) (그림 5) 프로그램의 수행 결과

5. 결 론

컴퓨터가 개발되면서부터 매우 복잡하고 큰 문제를 해결하기 위한 여러 가지 방법들이 연구되어 왔다. 이러한 연구의 결과로 슈퍼컴퓨터나 병렬 컴퓨터가 개발되었고, 네트워크에 연결된 컴퓨터들을 단일 시스템처럼 사용할 수 있게 하는 네트워크 컴퓨터도 개발되었다. 이러한 환경에서 처리하는 문제는 보통 과학 계산용 문제이다. 이러한 문제들은 문제의 특성상 문제 자체의 내부 병렬성을 기반으로 병렬처리하게 된다. 따라서, 여러 프로세서 또는 컴퓨터에 할당되는 태스크들은 독립적으로 자신의 해를 구하는 것이 아니라 다른 여러 프로세서 또는 컴퓨터와의 통신에 의해 중간 결과를 교환하면서 해를 구하게 된다. 따라서 대부분의 네트워크 컴퓨터는 메시지 패싱 기반의 환경을 사용한다.

최근 생물정보학, 3차원 애니메이션, 그래픽스 등의 분야에 대한 관심이 증폭되고 있다. 이들 분야는 대용량의 컴퓨팅 파워를 필요로 하는 문제를 많이 가지고 있다. 또한 이들 분야에 속하는 많은 문제들은 독립적인 작은 문제로 분할될 수 있다는 특징을 가지고 있다. 따라서 이러한 종류의 문제를 풀기 위해 메시지 패싱 기반의 환경보다 더욱 효율적인 환경을 개발하는 것은 매우 중요한 연구 분야라고 할 수 있다.

본 논문에서는 이러한 종류의 문제를 인터넷에 연결된 가용 컴퓨터를 사용하여 효율적으로 처리하는 프록시 컴퓨팅 기반의 새로운 네트워크 환경인 InterCom을 제안하였다. InterCom은 클라이언트, 서버, 에이전트로 구성된다. InterCom의 장점은 다음과 같다. 먼저 병렬 처리를 위해 사용자에게 제공되는 API 개수가 메시지 패싱 기반의 환경보다 월등히 적다는 것이다. 이는 사용자가 쉽게 병렬 프로그램을 작성할 수 있다는 것을 의미한다. 또한 수행 중에 다른

컴퓨터로 분산되는 코드는 사용자의 도움 없이 자동으로 컴파일되어 실행파일이 만들어진다. 이것은 사용자가 이 환경을 손쉽게 사용하도록 한다. 마지막으로 InterCom을 이루는 컴퓨터에 계정이 없어도 프로그램을 수행할 수 있게 하였다. 따라서 각 사용자는 손쉽게 많은 컴퓨팅 파워를 얻을 수 있다. InterCom의 추가적인 기능으로는 가용 컴퓨터를 효율적으로 관리하기 위한 로드 모니터링 기능과 사용자 프로그램을 사용자에게 투명하게 수행할 수 있도록 하는 결함허용 기능이 있다.

InterCom은 현재 C로 구현되었고, C프로그램을 위한 API만을 지원하고 있다. 그리고, 각 API에서는 C의 기본 데이터 타입만을 사용할 수 있게 하였다. 따라서 앞으로 다른 프로그래밍 언어를 위한 API를 개발하고, 기본 데이터 타입뿐만 아니라 사용자 정의 데이터 타입도 사용할 수 있도록 해야 한다. 또한 서버에서 한번 수행되었던 문제를 재사용할 수 있는 기능도 추가되어야 한다.

참 고 문 헌

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam, *PVM : Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, The MIP Press Cambridge, Massachusetts, 1994.

[2] 황석찬, 최재영, 김명호, "Java를 이용한 병렬 프로그래밍 환경", *정보과학회논문지(A)*, Vol.26, No.1, 1999.

[3] W. Gropp, and E. Lusk, *User's Guide for MPICH, a Portable Implementation of MPI*, ANL/MCSTM-ANL, Jun. 1996.

[4] *LAM/MPI Parallel Computing*, http : //ccl.osc.edu/lam.html.

[5] R. Butler, and E. Lusk, "Monitors, Messages, and Cluster s : the P4 Parallel Programming System," Technical Report Preprint MCS-P362-0493, Argonne National Laboratory, Argonne, ILI, 1993.

[6] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI : The Complete Reference*, The MIT Press Cambridge, Massachusetts, 1996.

[7] H. Kubinyi, G. Folkers, and Y. Martin, *3D QSAR in Drug Design*, Vol.2, Kluwer/Escom, 1998.

[8] I. Foster, and C. Kesselman, *Gid : Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc., 1998.

[9] H. Casanova, M. Kim, J. Plank, and J. Dongarra, "Adaptive Scheduling For Task Farming With Grid Middleware," *The International Journal of High Performance Computing Applications*, Vol.13, No.3, pp.231-240, 1999.

[10] H. Casanova, and J. Dongarra, "NetSolve : A Network-enabled Server for solving computational Science Problems," *In Proceedings of Super computing '96, Pittsburgh.* Department of Computer Science, University of Tennessee, Knoxville, 1996.

[11] H. Casanova, and J. Dongarra, "NetSolve : A Network Enabled Server, Examples and Users," *Proceedings of the Heterogeneous Computing Workshop*, Orlando, Florida, pp.19-28, 1998.

[12] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima, "Ninf : Network based Information Library for Globally High Performance Computing," *Proc. of Parallel Object-Oriented Methods and Applications(POOMA)*, Santa Fe, 1996.

김 명 호

e-mail : kmh@comp.ssu.ac.kr

1989년 숭실대학교 전자계산학과(공학사)

1991년 포항공과대학교 전자계산학과 (공학석사)

1995년 포항공과대학교 전자계산학과 (공학박사)

1995년~1995년 한국전자통신연구소 선임연구원

1995년~현재 숭실대학교 컴퓨터학부 조교수

1998년~1999년 University of Tennessee 전자계산학과 교환교수

관심분야 : GRID, 컴퓨터 보안, 리눅스, 클러스터링, 병렬/분산처리

박 권

e-mail : kpark@capital.lg.co.kr

1999년 시립인천대학교 전자계산학과 (공학사)

2001년 숭실대학교 대학원 컴퓨터학과 (공학석사)

2001년~현재 LG캐피탈 IT서비스팀 정보분석파트(DW) 사원

관심분야 : 클러스터링 웹서버, 병렬 및 분산시스템, 고성능 컴퓨팅, Data WareHousing, CRM