에이전트 기반 분산 컴퓨팅 환경 설계 및 구현

박 권[0] 김명호
숭실대학교 컴퓨터학과
terius@amin.ssu.ac.kr, kmh@comp.ssu.ac.kr

Design and Implementation of Agent-based Distributed Computing Environment

Park Kweon[0] Kim Myoung-Ho
Dept. of Computing, Soongsil University.

요 약

컴퓨터 네트웍 기술의 발달로 고성능 컴퓨팅을 위해 물리적으로 분산된 자원들을 사용하려는 노력의 일환으로 현재 많은 프로젝트가 진행되어 왔다. 이런 기술들은 과학 계산처럼 복잡하고 큰 계산을 위한 소프트웨어 라이브러리의 구현이 주류를 이룬다. 하지만 이런 라이브러리를 사용하기 위해서는 사용자에게 많은 프로그래밍 능력을 요하고 세부사항까지 알아야만 프로그래밍이 가능한 것이 많다. 본 논문에서는 사용자에게 사용하기 쉬운 인터페이스를 제공하고, 고성능 컴퓨팅이 가능한 시스템을 제시한다.

1. 서론

오늘날 과학계산처럼 매우 복잡하고 큰 문제를 해결하기 위한 효율적 해결책에 대한 연구는 계속 진행되어 오고 있다. 이런 다양한 메커니즘들은 서로 다른 플랫폼을 통해 계산을 수행하기 위해 개발되어져 왔고, 이들의 공통점은 소프트웨어 라이브러리를 제공한다는 점이다. 어떤 소프트웨어 라이브러리는 특정 플랫폼에 대해서 최적화 되어있거나, 다른 컴퓨터 시스템에 대한 인터페이스는 제공되지 않는 것도 있다. 또한 사용자로 하여금 상당한 프로그래밍 노력을 요구하는 것도 있다[1]. 이런 환경들에 있어서 근본적인 문제점은 사용자가 복잡한 인터페이스 구조나 세부사항들을 모두 알아야만 프로그래밍이 가능하다라는 것이다. 본 논문에서는 사용자들에게 보다 쉬운 인터페이스를 제공하여, 네트워크 상에 있는 자원들에게 효율적으로 접근을 할 수 있도록 하기 위한 시스템(InterCom)을 제시하고 구현에 대해 설명한다.

2. 기존 연구

현재까지 네트워크 컴퓨터의 분산 컴퓨팅을 이용하여 프로그래머에게 효율적인 병렬 환경을 제공해 주기 위한 연구와, 과학계산처럼 복잡하고 큰 문제를 해결하기 위한 연구가 진행되고 있다. 대표적인 예로는 MPI[4], PVM[5], P4, Express, Linda, Netsolve[1], Ninf[3]등이다. 이러한 시스템들의 문제점은 사용자에게 많은 프로그래밍 노력 부담이 존재하게 되고, 복잡한 인터페이스에 대한 API들을 모두 알아야만 프로그래밍이 가능하다. 이런 단점을 극복하기 위해서 본 논문에서는 사용자에게 보다 쉬운 인터페이스를 제공할 수 있도록 최소한의 API를 사용하여 고성능 컴퓨팅이 가능한 시스템을 제시 한다.

3. InterCom(Internet Computing) 시스템

InterCom시스템은 로컬 네트워크상이나 인터넷에 모든 구성원들이 연결될 수 있다. InterCom은 일반 C 언어 프로그램에서 함수 부분을 원격에서 실행하여 결과를 받아오기 위해, 사용자에게 사용하기 쉬운 인터페이스를 제공하여 일반 프로그램 함수 호출에 의한 수행처럼 사용할 수 있게 만든 시스템이다. 즉, 일반 프로그램에 필요한 부분만 인터페이스를 사용하여 호출을 하면 내부적으로는 에이전트가 작업을 적절히 분배를 해준다. 이때 사용자는 내부 구동 동작을 모르고도 단지 인터페이스만 사용하여 호출함으로써 사용자에게 프로그래밍 부담을 줄일 수 있다.

그림1은 InterCom 시스템의 전체 개념적인 그림이다. 이 그림에서 3가지 주요구성요소를 볼 수 있다.

- ◆ InterCom Agent:서버리스트를 관리하고 클라이언트의 작업을 적절히 분배.
- ◆ InterCom Server:실제 클라이언트의 작업 수행.
- ◆ InterCom Client:사용자에게 인터페이스 제공.

각 구성요소들의 동작은 내부적으로 다음 4단계로 동작한다.

1) 클라이언트는 에이전트에게 리퀘스트를 보내고,
2) 에이전트는 최상의 InterCom 서버를 선택해주고,
3) 클라이언트작업을 선택된 서버에게 전달하여 해결하고,
4) 결과를 직접 클라이언트에게 돌려준다.

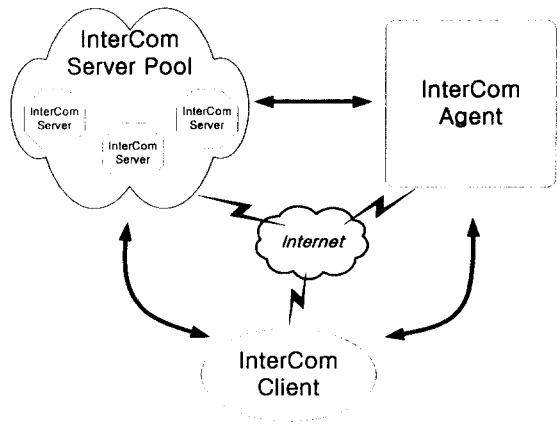

그림 1 InterCom 시스템 구조

3.1 InterCom 에이전트

3.1.1 RequestForServer 데몬

서버와 통신하여 서버들의 리스트를 등록, 관리해주는 부분으로서 서버 자신의 로드정보가 변경될 때마다 로드정보를 보내준다. 보내준 정보는 기존의 서버리스트들에 삽입할 때 로드정보가 좋은 순서로 정렬하여 서버리스트를 유지한다.

3.1.2 RequestForclient 데몬

클라이언트의 요청이 들어오면 문제를 해결할 수 있는 서버를 선택해 주는 부분으로서 위 데몬에서 유지하고 있는 정렬된 서버리스트를 모두 전송한다.

3.2 InterCom 서버

서버는 클라이언트의 요청 문제를 실제 해결하는 곳으로서 클라이언트의 작업을 처리해주는 부분과, 에이전트와 통신하여 자신의 로드정보를 전달해주는 부분으로 구성한다.

3.2.1 서버 데몬

서버데몬은 클라이언트로부터 전달된 정보들을 가지고 작업을 처리 한다. 전달된 함수가 실행되기 위해서는 가지고 온 정보를 파싱하여 main함수를 구성한다. 구성된 프로그램을 컴파일하고 실행하여 결과를 파일로 저장한 후 클라이언트에게 전송한다.

3.2.2 서버로드 데몬

서버로드데몬은 에이전트와 통신하는 부분으로서 처음 서버가 구동될 때 에이전트에게 등록을 한 후, 주기적으로 자신의 로드정보를 체크하여 자동으로 로드정보를 전송하는 역할을 한다. 서버 시스템 로드정보로는 CPU 이용율, DISK 이용율에 대한 값이다.

3.3 클라이언트

3.3.1 클라이언트 인터페이스

사용자가 작업하고자 하는 함수 부분을 일반 프로그램에서 함수 호출 처럼 호출할 수 있도록 인터페이스를 제공한다. 사용자는 일반프로그램을 작성하다가 필요한 함수 부분만 인터페이스를 사용하면 되므로 내부적으로 네트워크 통신 프로그램을 몰라도 되고, 어떤 서버에서 실행하라고 지정하지 않아도 된다. 이 부분은 에이전트가 알아서 적절히 처리를 해준다. 인터페이스에서는 단지 클라이언트 데몬과 통신을 하고, 모든 작업은 클라이언트 데몬에서 이루어 진다.

3.3.2 클라이언트 데몬

클라이언트 인터페이스에 의한 호출을 하면, 모든 작업은 이 데몬에서 수행한다. 여기서 블럭킹, 넌블럭킹 두가지 모드 함수 호출에 대해서 각각 수행을 달리 한다. 작업할 함수에 대한 아규먼트 변수에 대한 파싱을 하여 파일로 저장하고, 에이전트에게 요청하여 작업할 서버리스트를 받아온다. 선택된 서버로 클라이언트의 정보를 전송하고 나중에 서버로부터 결과를 받는다. 블럭킹 호출일 경우에는 서버로부터 결과가 올 때까지 기다리고 있고, 넌블럭킹 호출일 경우에는 서버에게 정보를 전송한 후 바로 클라이언트 코드로 리턴 한다. 나중에 이 결과를 서버데몬으로부터 작업이 끝났을 때 자동으로 결과를 받아서 특정 플래그를 세팅해준다. 이를 위해 결과가 도착했는지 여부를 알 수 있는 인터페이스를 통해 결과를 검사하고 받아온다.

4. InterCom 시스템 구현

InterCom 시스템은 동일한 리눅스 환경에서 C 언어 프로그램을 사용하여 구현하였다. 클라이언트에서의 인터페이스 호출방법은 블럭킹, 넌블럭킹 호출 모두 지원한다. 블럭킹 호출은 해당 함수 호출이 임의의 서버에서 작업을 마치고 결과값이 돌아올때까지 기다리는 호출을 말하고, 넌블럭킹 호출은 함수가 호출되었을 때 함수의 결과를 받을 때까지 기다리지 않고 바로 리턴하는 함수 호출을 말한다. 그러므로 나중에 결과를 검사하고 받아오는 인터페이스가 존재한다. 즉, 호출함수의 결과를 기다리지 않고 다음 코드를 수행할 수 있으므로 사용자는 함수 호출을 병렬로 수행할 수 있다.

그림2는 InterCom의 내부 동작을 상세히 나타낸 그림이다. 모든 자원들(서버)은 먼저 에이전트에 등록을 한다(1). 클라이언트 인터페이스를 통해 에이전트에게 리퀘스트를 보내면(2) 응답 메시지로써 서버리스트를 클라이언트 데몬에게 보낸다(3). 클라이언트 데몬은 선택된 서버로 접속하여 작업을 보내주면(4), 서버는 해당작업을 파싱하고, 컴파일 실행하여(5) 그 결과를 클라이언트 데몬에게 보낸다(6).

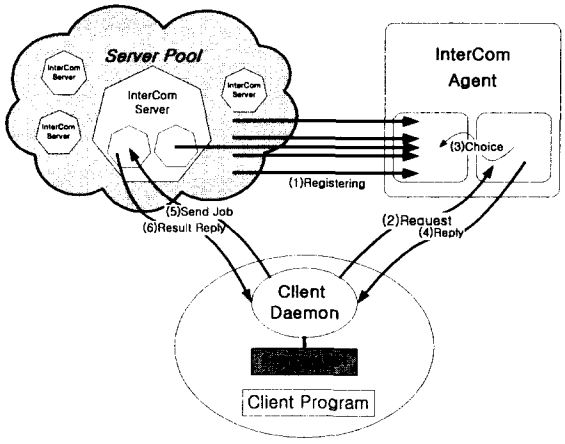

그림 2 InterCom 내부 동작도

4.1 에이전트 구현

에이전트는 클라이언트의 요청에 대해 유효한 서버리스트를 전송해주는 역할과 서버들을 등록, 관리하는 역할을 한다. 서버를 등록, 관리하는 부분에서는 자신의 로드가 변경되었다고 서버들이 자동으로 시스템 로드값을 보내준다. 이 로드값을 가지고 모든 서버리스트들을 성능이 좋은 순서대로 정렬한다. 그래서 정렬된 서버리스트들을 클라이언트 요청이 오면 바로 서버리스트정보를 클라이언트 데몬에게 전달한다. 등록된 서버를 등록, 관리 및 작업 분배를 하는 에이전트는 fault-tolerance와 load-balancing에 대한 책임이 있다.

4.2 서버 구현

서버자원들은 클라이언트에서 보낸 함수 모듈을 직접 컴파일하고 실행하는 부분이다. 서버는 서버로드데몬과, 서버데몬으로 구성되어 있다. 서버로드데몬은 주기적 간격으로 자신의 로드를 체크하여 이전에 구한 로드정보와 쓰레스홀드값(± 10)의 차이가 날때마다 자동으로 에이전트에게 자신의 로드정보를 보낸다. 서버데몬은 클라이언트에서 전달된 정보를 가지고 아규먼트정보 파일을 파싱하고 main 프로그램을 구성하여 컴파일 실행하고, 결과를 파일로 저장하여 바로 클라이언트데몬에게 전송해준다. 여기서 로드정보에 대한 계산법은 CPU 이용율에 대해서 40%의 가중치를 주고 DISK 이용율에 대해서는 60%의 가중치를 주어 계산한 로드값이다.

서버가 갑자기 다운되거나 인터럽트를 받으면 바로 에이전트에게 서버자신이 다운되었다고 보고를 하는 시그널처리를 하여 에이전트가 항상 유효한 서버리스트만 확보하도록 하였다.

4.3 클라이언트 구현

클라이언트는 사용자에게 인터페이스를 제공한다. 사용자가 일반 C 언어 프로그램을 작성하다가 필요한 부분에서 우리의 인터페이스를 통해 호출을 하면 된다. 인터페이스의 호출 방법

은 블럭킹 호출방법과 넌블럭킹 호출방법으로 나뉜다. 각각에 대해서 함수 프로토타입과 사용법에 대해서 알아보자.

4.3.1 블럭킹 호출 인터페이스

먼저 블럭킹 호출에 대한 방법을 살펴보자. 표1은 블럭킹 호출 인터페이스의 프로토타입이다.

```
void InterCom(char *function_name, char *formatlist. ...);
  • function_name : 원격으로 실행할 함수 이름
  • formatlist : 함수의 아규먼트 리스트에 대한 타입 포맷
```

표 1 블럭킹 호출 인터페이스

블럭킹 함수 호출은 함수결과가 리턴되어 오기까지 클라이언트 인터페이스 부분에서 기다렸다가 다음코드를 수행한다. InterCom()인터페이스의 두 번째 아규먼트는 printf()함수의 포맷과 비슷하다. 입력변수(%I)와 출력변수(%O)의 구분을 위해 I,O를 사용하였고, 배열일 경우에는 형식지정자 사이에 배열의 크기를 집어넣어 주면 된다. 현재 지원되는 데이터 타입은 C 언어의 기본 데이타 타입 모두 지원한다.

4.3.2 넌블럭킹 호출 인터페이스

넌블럭킹 호출에 대한 방법은 인터페이스를 호출할 때 블럭되지 않고 바로 클라이언트 다음 코드를 수행한다. 표2는 넌블럭킹 호출 인터페이스의 프로토타입이다.

```
int  InterCom_nb(char *function_name, char *formatlist, ...);
  • return value : Request ID 값
  • function_name : 원격으로 실행할 함수 이름
  • formatlist
    ①넌블럭킹호출 : 함수의 아규먼트리스트에 대한 타입 포맷
    ②Probe : formatlist 에 "probe"를 입력하고 세 번째 아규먼트에
     Request ID를 인수로 갖는다.
    ③Receive : formatlist에 "receive"를 입력하고 세 번째 아규먼트
     에 Request ID를 인수로 갖는다.
```

표 2 넌블럭킹 호출 인터페이스

◆ probe 기능 : 넌블럭킹 호출로 리턴된 RequestID값을 가지고 해당 작업 결과가 도착했는지 여부를 검사한다.

◆ receive 기능 : 결과를 받아올 때 사용하는 인터페이스로서 위의 probe 기능으로 검사를 먼저 한 후 결과가 도착하지 않았으면 계속 기다리고 있다가 결과를 받으면 수행을 계속한다.

5. 실험

InterCom 시스템을 실제 C 언어프로그램에서 응용한 예를 보자. 다음은 간단한 매트릭스를 계산하는 프로그램이다.

<matrixmain.c>

```
#include "intercom.h"
main()
{ int A[10*10],B[10*10],C[10*10];
  int rid,i;
  /* A, B 배열 초기화 */
  for(i=1;i<101;i++){A[i]=i; B[i]=i;}
  rid =InterCom_nb("matrixmul","%I100d %I100d %O100d",A,B,C);
  InterCom_nb("matrixmul","probe",rid);
  InterCom_nb("matrixmul","receive",rid);
  for(i=0;i<100;i++) {
   if(i%10 ==0) putchar('Wn');
   printf("%d ",C[i]); }
}
```

이처럼 일반 C 언어 프로그램에서 사용하고자 하는 곳에 인터페이스로서 호출만 하면 된다. 위의 예제에서는 넌블럭킹호출에 대한 예제이다. 먼저 넌블럭킹으로 matrixmul이라는 함수를 호출한 후에 나중에 필요할 때 결과를 받아오면 된다. 우선 결과가 도착했는가 여부를 알아보도록 probe를 하였고 결과를 받아오기위해서 receive를 하여 결과를 받아왔다.

<matrixmul.c>

```
void matrixmul(int A[],int B[], int C[])
{
 int i,j;
 for(i=0; i < 100; i++)
   for (j=0; j < 100; j+=10)
    C[i]= *(A+i) * *(B+j);
}
```

다음은 실행결과이다.

```
$ gcc -o matrixmain matrixmain.c
$ matrixmain
probe rid [0]
Probe check::NOT YET
Receive_Check RequestID[0]
Received check::RECEIVED OK
func_out:matrixmul.out
RECEIVED RID[0] RECEIVE'S OK RETURN

91 182 273 364 455 546 637 728 819 910
1001 1092 1183 1274 1365 1456 1547 1638 1729 1820
.....
```

6. 결론 및 향후과제

InterCom 시스템은 에이전트를 기반으로 한 분산 컴퓨팅환경을 구현한 시스템이다. 사용자의 일반적 C 언어 프로그래밍 작업을 작업량이 많은 부분을 로컬에서 처리하지 않고, 즉, 로컬서버보다 더 나은 원격(서버)에서 실행하여 결과를 받아옴으로서 사용자의 작업효율을 높였다. 이런 작업을 할 수 있도록 사용자에게는 최소한의 API로 사용하기 쉬운 인터페이스를 지원함으로써 기존 연구의 단점을 극복하였다.

현재까지 클라이언트의 인터페이스는 기존의 C언어에서 지원하는 기본 데이타 타입만 가능하다. 이 부분을 파일포인터 뿐만 아니라, 사용자 데이터 타입도 가능하도록 향후 구현할 예정이다.

7. 참고 문헌

[1] Henri Casanova, Jack 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.

[2] 황석찬 "MPI에 기초한 Java 병렬 프로그래밍 환경" 숭실대학교 석사 논문, 1997.

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

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

[5] 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.