부하 균등 정책에 대한 객체 이동 시스템의 설계
Design of object migration system for load ballancing policy

조승한 (전자계산과), 서상훈 (사무자동화과)
Seung-Han Cho (Dept. of Computer Science), Sang-Hoon Suh (Dept. of Office Automation)

Key Words : 전역 컴퓨팅(global computing), 객체 이동(object migration)

ABSTRACT : Recent popularization of Internet and advances of technologies make easy sharing various information and utilizing system resources on Internet. Especially, for large computational problems to need excessive computation time with super computer, there have been numerous efforts to solve these problems with computers distributed on Internet. Code migration of java applet make possible distributing computational programs on Internet and we can get the solution of large computational problems by using distributed collaborative objects. Thus, we need a object distribution policy and a object migration method to get high performance. In this paper, we propose and design the technique to migrate a object from low performance computer to high performance computer.

1. 서 론

대량의 연산을 필요로 하는 문제에 대해 그 처리 성능을 높이기 위한 방법으로 병렬 시스템을 오래 전부터 연구해 왔었다. 또한 지역적으로 분산되어 있으며 네트워크로 연결된 컴퓨팅 자원을 효율적으로 이용하기 위한 분산 처리에 대한 연구들이 있었다. 더욱이 이들의 결합으로 처리 성능을 극대화 하는 병렬 분산 처리에 대한 연구가 있었으나 이는 참여하는 각 호스트들이 미리 설정되어 있는 상태에서 이루어지고 한정된 범위의 자원들만을 이용할 수 있도록 한다. 따라서 보다 많은 컴퓨팅 자원을 이용하고자 하는 노력으로 컴퓨팅 환경을 LAN에서 WAN으로의 확장이 시도되고 있다.

그러한 시도에 성공을 기대하게 해주는 좋은 환경이 바로 최근에 급속도로 성장하고 있는 인터넷이며 그 중 웹(Web)은 무수히 많은 호스트들을 연결하는 가상 시스템이 될 수 있다. 인테넷에 연결된 호스트들은 지역적으로 상당히 먼 거리에 분산되어 있고 그에 따라 각 호스트들이 활발하게 이용되는 시간도 서로 다르다. 즉, 어느 한 지역에서 활발하게 해당 지역의 호스트들을 이용하는 동안, 웹으로 연결된 지구 반대편 지역의 호스트들은 이용되지 않는 경우가 있을 수 있다. 현재는 이렇게 매우 넓은 지역에 걸쳐 분산된 여러 호스트들의 자원들을 대부분 한정된 범위에서만 이용하고 있을 뿐이다. 따라서, 이러한 자원들을 효율적으로 이용할 수 있는 기반의 필요성이 제시되고 있으며, 이러한 필요성에 의해 제시된 분야가 전역 컴퓨팅(global computing)이다[(1)].

전역 컴퓨팅의 근간으로 웹을 이용하려는 시도는 이미 전 세계적으로 구축되어 있는 인터넷을 이용하여 적은 비용으로 유휴 상태(idle state)에 있는 컴퓨팅 자원을 이용함으로써 활용도(utilization)를 높이고 병렬 처리를 가능하게 하게 한다. 그러나 웹 환경에 연결된 호스트들은 컴퓨터 구조나 운영 체제 등이 다양한 이질성(heterogeneity)으로 인하여, 코드의 개발, 분산 및 설치가 쉽지 않다.

이러한 이질성을 극복하는 수단을 제공하는 것으로 웹 환경에서 이미 널리 사용되고 있는

자바(JAVA)가 있다. 자바는 어느 플랫폼에서도 동작이 가능하며, 네트워크 관련 API(Application Programming Interface)가 풍부하고, 클라이언트-서버 형태의 시스템 개발에 적합한 언어이다. 특히 웹 환경에서 이동 바이트 코드인 애플릿(applet)은 브라우저(browser)로 적재되어 연산이 수행되는 구조를 갖고 있다. 이것은 프로그램의 분산을 쉽게하고 그 컴퓨팅 자원을 쉽게 이용할 수 있도록 한다. 또한 분산된 애플릿은 내재된 API나 RMI(Remote Method Invocation)를 통하여 다른 애플릿간에 정보 교환을 쉽게 할 수 있다[(2)].

따라서 방대한 연산 수행을 필요로 하는 문제에 대해 자바 애플릿을 사용하여 전 세계에 걸친 유휴 상태 컴퓨터에서 그 연산을 병렬 처리하는 것이 가능하게 되었다. 이러한 웹 기반의 병렬 처리는 기존의 대용량 컴퓨터가 갖는 처리 능력 이상의 잠재적인 능력을 갖고 있다 하겠다. 실제로 유럽에서는 1997년 1월에 3,500 대의 워크스테이션으로 48 비트 RSA 코드가 해독되었다[(3)]. 또한 1997년 6월에는 대략 78,000대의 컴퓨터로 56 비트 DES(Data Encryption Standard)가 해독되었다[(4)].

이러한 웹 기반 병렬 처리 시스템에 참여하는 컴퓨터들은 전 세계적으로 분산되어 있고 유휴 컴퓨터의 사용자가 자신의 컴퓨터를 사용함에 따라 전체적인 병렬 연산 처리 성능이 급격히 떨어질 수 있다. 따라서 처리 성능이 저하된 컴퓨터는 다른 컴퓨터로 대체되는 것이 바람직하다. 이와 같이, 참여하는 컴퓨터의 처리 부하를 균등하게 유지하며 연산 처리를 서로 분배하는 부하 균등 정책이 전체적인 성능에 중요한 영향을 미친다. 또한 이러한 분배 정책에 따라 연산 처리에 사용되는 객체가 적은 부하를 갖는 컴퓨터로 이동하여 연속적인 연산이 이루워질 수 있게 하는 것이 필요하다.

이 논문에서는 병렬 처리 애플릿이 사용하는 문제 해결 객체에 대해 부하 균등 정책에 따라 다른 호스트로의 이동이 요구되는 상황에서 원격 객체 참조(Remote Object Reference)를 갖는 애플릿에게 다른 호스트로의 이동이 발생하였는지 모르고 메소드 호출을 이룰 수 있게 하기위한 객체 이동에 대한 기법을 제시하며 그에 대한 시스템을 설계하고자 한다.

2. 관련 연구

오늘날 자바의 플랫폼 독립성을 이용하여 인터넷 환경을 분산 병렬 시스템으로 구축하려는 연구가 진행중에 있다. 자바 환경을 사용한 이러한 시스템 중에서 Charlotte는 Droutine이라는 원격 호스트에서 수행되는 스레드 클래스를 구현하며 열정적 스케줄링(eager scheduling) 기법을 사용해 각 작업 호스트간의 성능 차이문제를 해결하고 있다[(5)]. 하지만 이것은 동일한 작업이 여러 곳에서 동시에 수행되므로 과도한 자원 낭비를 초래한다.

Javelin은 웹을 기반으로 분산 병렬 시스템 구축에 대한 해결책을 제안하며 전역 시스템을 구축해 광선 추적법(Ray Tracing) 및 메르젠(Mersenne) 소수의 탐색과 같은 연산을 처리하였고, 그 수행 결과를 보였다[(6)]. 여기에는 낙관적 작업 전파(optimistic load propagation)에 기반한 부하 균등 기법을 제시하고 있다. SuperWeb은 기본적으로 Javelin 보다 향상된 구조를 제시하고 있으나 부하 균등에 대한 알고리즘없이 처리되고 있다. JET는 주기적으로 결함(fault)을 검출하여 연산을 재분배하는 방법을 사용하고 있으며 이것을 사용하여 세일즈맨 여행 문제 및 암호문 해독 문제를 해결하고 있다[(7)]. 또한 POPCORN은 원격 객체에 대해 작업을 의뢰하고 결과를 돌려 받기 위한 Commputelet 클래스를 정의하였으나, 부하 균등 정책 없이 결함만을 감지해 연산 작업을 다른 작업자에게 다시 처리 의뢰한다는 간단한 결함 해결책을 제시하고 있다[(8)].

이에 본 논문은 부하 균등 정책이 주어진 환경에서 그 정책에 따라 대상 작업자가 결정되었을 때, 객체가 갖는 현재의 상태를 유지하며 지속된 연산을 가능케 하는 객체 이동 개념을 도입하였다. 연산에 참여하고 있는 작업자의 성능이 급격히 저하된 경우 이러한 객체 이동을 통해 다른 작업자가 객체의 연산을 지속적으로 수행할 수 있다. 이러한 기법을 사용함으로써 개별 작업자의 성능 저하로 인한 전체 분산 병렬 시스템의 성능 저하를 막을 수 있다.

3. 객체 이동 시스템

3.1 기본적 요소

객체 이동을 지원하는 시스템은 의뢰인(Client), 작업자(Worker), 조정자(Coordinator)라는 3가지 요소들로 구성되어 있다.

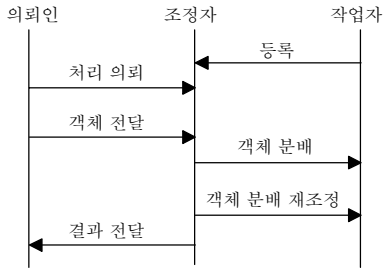
・의뢰인(Client) : 대량의 연산이 필요한 작업을 조정자에게 의뢰하고 작업에 필요한 인자값(arguments)를 제공한다. 작업의 수행이 시작되면 수행 작업의 결과를 받을 준비가 필요하다.

・작업자(Worker) : 조정자로부터 전체 작업의 일부분인 객체를 할당받아 그 처리를 수행한다. 이러한 객체는 작업자의 자원을 점유하며 연산을 수행한다.

・조정자(Coorinator) : 의뢰인과 작업자의 등록 및 관리를 수행하며 의뢰인과 작업자간의 통신을 중재한다. 또한 작업자들의 부하 균등을 주어진 정책에 따라 조정한다.

3.2 구성 요소간 처리 절차

아래 〔그림-1〕은 구성 요소간의 처리 절차이다. 작업자는 조정자에게 연산 참여에 가담하겠다라는 것은 사전에 등록한다. 의뢰인은 조정자에 접속하여 연산 처리를 위해 필요한 객체들을 전달한다. 조정자는 수신된 객체들을 자신의 부하 균등 정책에 따라 사전에 등록된 작업자들에게 그 처리를 일임한다. 원격 객체 참조를 포함하고 있는 객체 사용 작업자는 조정자에게 객체 연결을 요구한다. 이러한 요구를 받은 조정자는 어느 작업자가 그 객체를 수행하는지 찾아 연결 관계를 맺도록 유도한다.



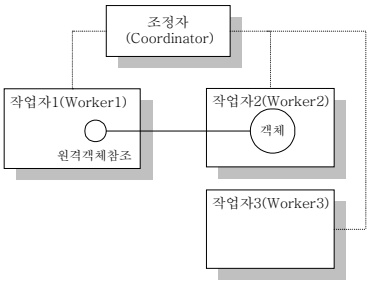
〔그림-1〕 구성 요소간 처리 절차

3.3 객체 이동의 정의 및 필요성

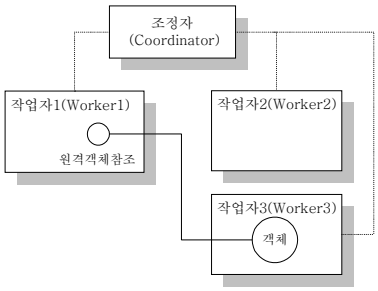
객체 이동에 대해 논하기 앞서 이 논문이 말하는 객체라는 것에 대해 정의할 필요가 있다. 여기서 말하는 객체란 일반적으로 널리 인정되는 정적 정보(자료 구조, 속성)와 동적 수단(연

산, 메소드)의 결합된 형태이다. 예를 들어 스택(stack) 객체는 데이타를 표현하기 위한 자료조와 push, pop등과 같은 스택 연산이 포함된 형태인 것이다. 또한 운영체제 또는 애플리케이션 프로그램의 병렬 처리 단위인 프로세스, 스레드도 하나의 독립된 객체로 PCB(Process Control Block)와 같은 정보가 그의 자료 구조이고 연산으로는 start, stop, sleep, suspend 등과 같은 것이 있기 때문이다. 따라서 객체에 대한 참조(reference)를 생성했을 때, 메소드 호출(method invocation) 시에만 그 연산이 처리되는 구조일 수도 있고, 시작(start) 메소드에 의해 코드를 지속적으로 처리하며 멈춤(stop) 메소드에 의해 끝나는 구조일 수도 있다. 대량의 연산 처리를 위해 작성된 객체는 그의 연산 구조에 따라 앞에서 살핀 스택과 같은 형태 일 수도 있고 스레드와 같은 형태일 수도 있다.

객체 이동이라 함은 컴퓨터의 처리 성능의 저하와 같은 정책에 의해 부하 균등 조정자(coordinator)가 어느 한 컴퓨터의 객체를 동일한 객체 정보를 갖고 다른 컴퓨터로 이동하는 을 의미한다. 여기서 동일한 객체 정보를 갖는다는 의미는 객체 이동 후 원격 객체 참조를 사용하는 프로그램이 객체가 이동해 갔는지 전혀 알지 못하고 이동전과 동일하게 작업을 처리한다는 것을 의미한다. 아래 〔그림-2〕는 작업자1이 작업자2에 있는 어떤 객체에 대해 원격 참조를 하고 있는 상황을 묘사하고 있다. 조정자에 의해 작업자2의 객체가 작업자3으로 이동된 후의 상황을 〔그림-3〕이 보여주고 있다.



〔그림-2〕 객체 이동 전



〔그림-3〕 객체 이동 후

이러한 객체 이동은 웹 기반 병렬 처리 시스템에 참여하는 컴퓨터들이 전 세계적으로 분산되어 있고 연산에 참여한 유휴 컴퓨터의 사용자들이 자신의 컴퓨터를 어느 수준 이상 사용할 경우 그 컴퓨터들의 연산 처리 저하로 전체적인 병렬 연산 처리 성능이 급격히 떨어지는 것을 방지하기 위함이다. 즉, 조정자에 의해 처리 성능이 저하된 컴퓨터는 객체 이동을 통해 다른 컴퓨터로 대체되도록 하자는 것이다. 이러한 조정 과정은 부하 균등 정책에 따라 결정된다.

3.4 기존의 전략들

기존에는 조정자가 부하 균등 정책에 따라 연산을 재조정 작업이 필요할 때, 연산 처리에서 제외될 작업자가 갖는 모든 연산을 취소한다. 그리고 그러한 연산을 다른 작업자에게 새로이 의뢰한다. 이것은 특정 문제에 대해 그 연산 구조를 조정자가 알 수 없으며, 현재 어느 상태까지 연산 작업이 이루워 졌는지 알지 못하기 때문이다. 따라서 이러한 조정 작업을 연산 처리를 의뢰한 의뢰자가 담당하도록 하고 있다[6,7,8]. 다른 한편 원격 객체 참조에 대한 이동은 현재 많은 연구가 이루워져 표준[9]으로 자리잡고 있는 실정이나 객체 이동에 관한 연구는 아직 알려진바 없다.

4. 객체 이동 시스템의 설계

4.1 Task 객체의 설계

조정자에 의해 작업자2에서 작업자3 으로의 객체 이동이 결정됐을 때 실제로 객체 이동이 발생되는 시점에 대해 살펴보자. 조정자에 의한 주기적 성능 검사에 의해서든 개별 작업자의 측정에 의해서든 연산 취소의 결정이 내려지는 시점은 비동기적일 수 밖에 없다. 따라서 결정 즉시 객체의 이동이 있기 위해서는 객체의 현재 수행 상태를 정지시키고 객체에 대한 자료구조 뿐 아니라 어느 코드까지 수행됐는지 등과 같은 객체의 상태 정보가 필요하다. 이러한 상태 정보는 레지스터 레벨 또는 자바 가상 기계 레벨에서 얻울 수 있는데 레지스터 레벨 상태 정보는 다른 작업자로의 이동시 서로 다른 메모리 공간으로 인해 무의미한 정보이다. 자바 가상 기계 레벨의 상태 정보는 완전한 상태 정보를 가상 기계가 알려주는 인터페이스가 있어야 하며 또한 그러한 정보를 가지고 그 상태로 들어가기 위한 인터페이스의 표준화된 새로운 개발이 필요하다.

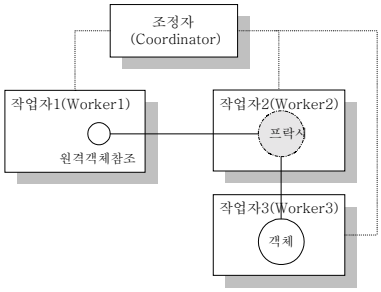
따라서 이러한 비동기적 시점에서의 객체 이동을 지양하고 특정 메소드의 호출시 발생하는 것이 바람직하다 하겠다. 이것은 스택과 같은 객체시에는 메소드 처리 결과의 리턴 시점에 호출하고 스레드시에는 중간 중간의 분기점 위치에 호출하는 것이 필요하다. 그리고 이 메소드는 모든 연산 객체에 공통적으로 포함되도록 해야 하기 때문에 스레드 객체의 상위 객체의 메소드로 정의하는 것이 바람직하다. 이 객체의 이름을 Task라 하고 다음과 같은 메소드를 포함한다.

메소드	설 명
Object invoke(String method, Object	연산 객체의 메소드를 호출한다.
void send(Object)	연산 결과를 보낸다.
Object recv()	연산 결과를 받는다.
void migration()	이동 여부를 검사하고 참이면 이동한다.

여기서 migration() 메소드는 조정자에 의해 이동 요청이 들어왔는지 살피고 요청이 있었으면 이동할 작업자에게 현재의 자신의 객체를 전달한다.

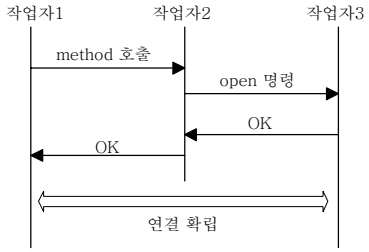
4.2 Task 객체를 이용한 객체 이동

다음 〔그림-4〕는 시간 순서에 있어 앞서 보여진 〔그림-2〕와 〔그림-3〕 사이에 위치하며 아직 작업자1의 원격 객체 참조에서 작업자3의 이동 객체로의 객체 연결이 이루워 지지 않고 있는 상태를 나타내고 있다. 이 그림에서 작업자2에 있던 객체는 프락시 객체라 하며 다음에 설명될 작업자1과 작업자3과의 결합 과정에서 단순히 중간 연결자 역할을 수행한다. 이러한 프락시 객체는 작업자1 또는 작업자3에 의한 작업자1의 원격 객체 참조와 작업자3의 이동 객체가 연결된 다음에서야 종료되는 객체이다.



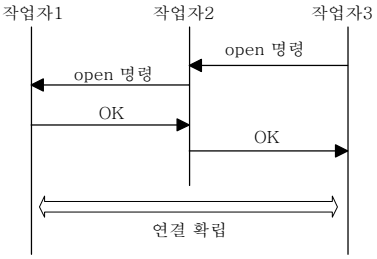
〔그림-4〕 작업자2의 프락시 객체

다음 〔그림-5〕는 객체 이동이 있은 후 작업자1의 원격 객체 참조 프로그램이 invoke(method,obj) 메소드를 호출한 경우의 프로토콜로 invoke 메소드는 이러한 일련의 과정을 갖고 있어야 한다. 그림에서 open 명령은 원격 객체 참조 프로그램과 객체간의 메시지 전달에서 연결 설정을 이루기 위해 미리 약속된 명령어 문자열이다.



〔그림-5〕 invoke method의 호출로 객체 연결 확립 프로토콜

다음 〔그림-6〕는 객체 이동이 있은 후 작업자3의 객체가 send(obj) 메소드를 호출한 경우의 프로토콜로 send 메소드는 이러한 일련의 과정을 갖고 있어야 한다.



〔그림-6〕 결과에 대한 리턴시 객체 연결 확립 프로토콜

5. 결론

오늘날 인터넷의 보급으로 대용량 컴퓨팅 파워를 능가하는 전역 컴퓨팅(global computing)을 구성하는 것이 하나의 큰 이슈가 되고 있다. 전역 컴퓨팅 환경을 조성하기 위해서는 무엇보다도 전역 정보 인프라(GII: Global Information Infrastructure)의 구축이 필요하며 코드의 이동성, 이동된 코드의 보안성, 신뢰성등을 기본으로 하고 있다. 그리고 할당된 객체 자원에 대한 부하 균등 정책, 정책에 따른 객체의 이동이 그 성능에 막대한 영향을 끼친다. 이 논문에서는 분산 객체 처리에 대한 객체 이동에 대한 방법을 제안하며 그에 대한 지원 시스템을 자바 환경을 배경으로 설계하였다.

이 논문에서는 원격 객체 참조를 갖는 작업자의 연산 참여 취소시 발생하는 문제와 그 해결에 대한 언급이 없으나 객체 이동과 같은 구조에 대칭성을 주면 쉽게 설명될 수 있다.

앞으로의 연구 과제로는 이에 대한 구현과 전체적인 성능을 향상시키는 부하 균등 정책의 개발, 그리고 이러한 것들의 결합에 의한 전역 컴퓨팅 환경의 구성이라 하겠다.

참고 문헌

(1) J. Eric Baldeshwieler, Robert D. Blumofe, and Eric A. Brewer, "ATLAS : An Infrastructure for Global Computing", *In Proc. of the 7th ACM SIGOPS European Workshop on System Support for World wide Applications*, 1996

(2) Bill Venners, "Inside the JAVA Virtual Machine", *McGraw-Hill*, 1998

(3) Inc. RSA Data Security, "The RSA Data Secret-Key Challenge", *http://www.rsa.com/rsalabs/97challenge*, 1997

(4) CNET, Inc., "48-bit Crypto latest to crack", *http://www.news.com/News/Item/0,4,7849,4000.html*, 1997

(5) Arash Baratloo, Mehmet Karaul, Zvi Kedem, and Peter Wyckoff, "Charlotte : Metacomputing on the Web", *In Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems*, Sep., 1996

(6) Bernd O. Christiansen, Peter C. Japello, Mithai F. Ionescu, Michael O. Neary, Klaus E. Shauser, and Danieal Wu, "Javelin: Internet-based parallel computing using java", *In the ACM Workshop on Java for Science and Engineering Computation*, 1997

(7) Hernani Pedroso, Luis Silva, Jose M. Tavares and Joao Gabriel Silva, "Web-based Metacomputing with JET", *In the ACM Workshop on Java for Science and Engineering Computation*, 1997

(8) Noam Nisan, Shmulik London, Ori Regev, Noam Camiel, "Globally Distributed computation over the internet - The POPCORN project", *the 6th Int'l World Wide Web Conference*, Santa-Clara, Apr., 1997

(9) Sun Microsystems., Inc., "Java Remote Method Invocation Specification", *JDK 1.2 Beta1*, Oct., 1997