성능 기반 태스크 할당을 이용한 웹 기반 병렬처리 시스템의 설계

(Design of Web-based Parallel Processing System using Performance-based Task Allocation)

한 연 희 † 박 찬 열 †† 정 영 식 ††† 황 종 선 ††††

(YounHee Han) (Chan Yeol Park) (Young-Sik Jeong) (Chong Sun Hwang)

요 약 최근 인터넷 환경에서의 기술 향상으로 인하여 다양한 정보를 공유하고, 네트워크로 연결된 여러 시스템 자원을 이용하는 것이 용이하게 되었다. 특히, 자바의 애플릿(applet)을 이용한 코드 이동(code migration) 기술은 인터넷의 웹 환경에서 프로그램의 분산을 용이하게 하며, 그 애플릿을 수행하는 브라우저는 이동된 코드의 신뢰성을 보장해준다. 본 논문은 웹 환경에서 이동 가능 코드를 분산시키고, 대량의 연산수행을 지닌 작업을 분배하여 병렬적으로 수행시킨 뒤, 그 결과를 취합하는 웹 기반 병렬 시스템의 설계 및 구현에 관하여 기술한다. 또한, 이 시스템에 참여하는 이질적인 다수의 호스트들은 인터넷이라는 환경에서 지리적으로 떨어져 연결되어 있으므로 성능의 차이 및 가변성을 예상하기 힘들다. 그러므로, 그들 간의 성능 차이를 고려한 태스크 할당 알고리즘 및 심각한 가변성에 대한 적응력이 요구된다. 이 논문에서는 시스템의 구현에 사용될 적응성 향상 기법을 제시하고, 시스템의 작업 처리 성능 및 제안하는 알고리즘들의 효율을 나타내는 성능 평가 결과를 제시한다.

Abstract Recent advances of technologies make easy sharing various information and utilizing system resources on the Internet. Especially, code migration using applets of Java supports the distribution of programs on the web environment, and also browsers executing the applets guarantee the reliability of a migrated codes. In this paper, we describe the design and implementation of a web-based parallel processing system, which distributes migratable codes of a large job, makes the distributed codes to execute in parallel, and controls and gathers the results of each execution. The hosts participate in the computation reside on the Internet, spreaded out geographically, and the heterogeneity and the variability among them are severe. Thus, task allocation considering the performance differences and the adaptability to the severe variability are necessary. We present an adaptive task allocation algorithm applied to our system and the performance evaluation.

1. 서 론

병렬 처리의 목적은 방대한 연산 수행이 필요한 작업을 여러 컴퓨팅 자원이 나누어 수행함으로써 전체 작업의 수행 시간을 단축시키는데 있다. 네트워크로 연결된 컴퓨팅 자원을 이용하는 병렬 처리에 대한 연구가 오래 전부터 이루어져왔으나, 이들은 지역적으로 한정된 곳에서 LAN에 연결된 여러 호스트들을 이용한 연구들이 대부분이었다[1]. 또한 이러한 방식의 병렬 처리는 병렬 처리에 참여하는 각 호스트들이 미리 설정되어 있는 상태에서 이루어지므로 한정된 범위의 자원들만을 이용할 수 있게 한다. 보다 많은 컴퓨팅 자원을 이용하고자 하는 노력으로 컴퓨팅 환경을 LAN에서 WAN으로의 확장이 시도되어 왔다.

바로 이러한 시점에서, 전세계적으로 널리 퍼져 구축

· 이 연구는 정보통신부의 대학기초연구 지원사업에 의하여 수행되었음.

† 학생회원 : 고려대학교 컴퓨터학과
yhhan@disys.korea.ac.kr
†† 비 회 원 : 고려대학교 컴퓨터학과
chan@disys.korea.ac.kr
††† 종신회원 : 원광대학교 컴퓨터정보통신학부 교수
ysjeong@wonkwang.ac.kr
†††† 종신회원 : 고려대학교 컴퓨터학과 교수
hwang@disys.korea.ac.kr

논문접수 : 1998년 11월 17일
심사완료 : 2000년 1월 18일

되어 있는 인터넷이 그러한 시도의 성공을 기대하게 해주는 좋은 환경이 되고 있다. 인터넷은 최근에 급속도로 성장하고 있으며, 그 중 월드 와이드 웹(World Wide Web, 이하 웹)은 무수히 많은 호스트들을 연결하는 가상 시스템이 될 수 있다. 전 세계에 걸친 매우 커다란 이 가상 시스템에 연결된 호스트들은 지역적으로 상당히 먼 거리에 분산되어 있으며, 그에 따라 각 호스트들이 활발하게 이용되는 시간도 다르다. 즉, 어느 한 지역에서 활발하게 해당 지역의 호스트들을 이용하는 동안, 웹으로 연결된 지구 반대편 지역의 호스트들은 이용되지 않는 경우가 대부분이다. 현재는 이렇게 매우 넓은 지역에 걸쳐 분산된 여러 호스트들의 자원들을 대부분 한정된 범위에서만 이용하고 있을 뿐이다. 그래서, 이러한 자원들을 효율적으로 이용할 수 있는 기반의 필요성이 제시되고 있으며, 이러한 필요성에 의해 제시된 분야가 전역 컴퓨팅(global computing)이다[2].

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)을 보이므로, 코드의 개발, 분산 및 설치가 쉽지 않다. 이러한 이질성으로 인하여 기존 연구들은 미리 설정되어 있는 호스트들만이 병렬 처리에 이용할 수 있는 제약사항이 있었다. 또한 병렬 처리만을 위해 사용되는 호스트들이 아니라 각각 나름대로의 목적에 이용되는 호스트들이므로 그 기종 및 성능면에서 매우 큰 다양성을 지니며, 병렬 처리에 이용될 때 할당된 연산의 수행에 대한 응답 시간을 예상할 수 없다. 유휴 상태라고 파악된 인터넷의 한 호스트가 병렬 컴퓨팅에 참여하여 태스크를 수행할지라도 갑작스런 환경 변화가 발생할 가능성이 매우 높다.

이 논문에서는 이러한 문제들을 해결하는 웹 기반 병렬 처리 시스템의 설계와 구현을 제안한다. 각 호스트들의 이질성은 웹 환경에서 이미 널리 사용되고 있는 자바(JAVA)를 이용하여 해결한다. 자바는 어느 플랫폼에서도 동작이 가능하며, 네트워크 관련 API(Application Programming Interface)가 풍부하고, 클라이언트-서버 형태의 시스템 개발에 적합한 언어이다. 특히 웹 환경에서 단말 사용자(end user)가 이용하는 브라우저(browser)로 적재되어 안전하게 연산 수행을 가능하게 해주는 이동 바이트 코드인 애플릿(applet)을 이용하여 웹에 연결된 호스트들에게 프로그램을 쉽게 분산시키고 그 컴퓨팅 자원을 쉽게 이용할 수 있다. 그리고 분산된 애플릿은 내재된 API나 RMI(Remote Method Invocation)를 통하여 다른 애플릿간에 정보 교환을 쉽게 할 수 있다. 그리고 급변할 수 있는 환경에 적응하기 위해 분배된 작업의 수행 중에 적응적으로 태스크를 재할당하는 적응적 성능 기반 태스크 할당(Performance-based Adaptive Task Allocation, PAPT) 기법을 제안한다. 태스크의 수행에 최초로 참여할 때 각 호스트에 분배된 애플릿에 의해 간단한 성능 평가(benchmark)가 수행되고, 그 결과에 기반하여 서버는 최초 태스크 할당을 수행하고, PAPT에 의해 태스크 수행 중 태스크의 수행 능력에 적응적으로 태스크 재할당이 이루어진다. 구현된 웹 기반 병렬 처리 시스템의 활용 및 성능 평가를 위한 응용으로써 프랙탈 이미지 처리가 사용되었다.

다음 장에서는 인터넷을 이용하는 병렬 처리에 관련된 연구들을 소개하고, 그 연구들이 제시하는 시스템과 우리가 구현한 시스템과의 차이를 설명한다. 3장에서는 시스템에 대한 설계 및 구성 요소들을 기술하며, 4장에서는 성능 기반 태스크 할당에 대한 알고리즘을 제시하고 이에 대한 분석을 보인다. 5장에서는 적응력을 갖기 위한 적응성 향상 기법을 제안하고 이를 통해 적응적 태스크 할당 알고리즘을 제안한다. 그리고 6장에서 구현된 시스템에 대한 성능 평가를 보이고, 마지막 7장에서 결론을 맺는다.

2. 관련 연구

Chandy 등[3]은 자바와 인터넷을 활용한 분산 시스템의 구성방법과 고려해야 할 여러 사항들에 대하여 제시하였으며, 메시지와 스레드를 활용하여 분산 응용프로그램을 구축할 때 발생할 수 있는 여러 제반 사항들을 고려한 시스템 예를 제시하였다. 한편, Vahdat 등[4]이 제안한 WebOS에 대한 연구에서는 웹 환경에서 응용프로그램의 수행에 필요한 여러 고려사항들, 즉, 자원 관리, 전역 이름공간(namespace), 원격 프로세스 수행, 인증 및 보안에 대한 내용을 기술하였다.

PVM[5]과 MPI[6]은 서로 다른 주소 호스트에 존재하는 프로세스간에 메시지를 전달하는 기본적인 근간을 마련해주고 있으며, JPVM[7], JAPE[8]는 기존의 PVM과 MPI를 자바로 이식하여 이질적 환경을 극복하는 방안을 제시하였다. 하지만 이들에서는 모두 LAN에 한정되어 있기 때문에, WAN에서 중요하게 생각되어야 하는 부하 균등 방법은 제시되지 않았다. 그리고, 프로그램 코드를 각 호스트에 배분하기 위해 그 호스트에

대한 계정(account)을 필요로 하는 등 사전 작업이 필요한 단점이 있다.

JavaParty[9]는 임의의 호스트에서 생성된 객체를 여러 호스트에 자동으로 분산시키는 원격 객체에 대한 메커니즘을 기반으로 자연스럽게 공유 메모리 공간을 형성해준다[10]. 하지만, 이것은 웹으로 그러한 시스템을 적용할 수 있는 해결책은 제시하지 못하고 있다.

근래에는 자바의 플랫폼 독립성의 장점에 기반하여 자바를 활용하여 인터넷 환경에서 병렬 시스템을 구축하려는 연구가 진행중이다. ATLAS[2]는 전역 컴퓨팅을 위한 여러 고려 사항을 항목별로 자세하게 제시하였다. ParaWeb[11] 은 자바 병렬 클래스 라이브러리(JPCL)를 구축하면서 웹의 컴퓨팅 자원을 활용할 수 있는 해결책을 제시하였지만, 자바 병렬 런타임 시스템(JPRS)이 기존 자바 가상 기계의 수정을 요구하기 때문에, 호환성 측면에서 단점이 있다. 그러므로, JavaParty, ATLAS와 ParaWeb은 오히려 LAN에 적당한 시스템으로 평가를 받고 있다.

Charlotte[12]는 Droutine이라는 원격 호스트에서 수행되는 스레드 클래스를 구현하고, 프로그래머로 하여금 웹을 공유 메모리 공간으로 생각할 수 있도록 만들어 주었다. 하지만, 태스크 균등 할당과 각 작업 호스트간의 성능 차이문제를 단순히 열정적 스케줄링(eager scheduling) 기법으로 해결하고 있어서, 같은 작업을 여러 곳에서 동시에 수행시킴으로서 발생하는 과도한 자원 낭비를 초래한다.

국내 연구로서 박지민 등[21]이 제안하는 시스템은 본 논문에서 제시하는 것과 유사한 의뢰인(Client)/마스터(Master)/작업자(Worker)의 요소들로서 구성되는데, 이 시스템의 장점으로는 마스터의 분산 그룹을 제공하는 것으로, 간단한 URL 변환을 통하여 작업자와 마스터와의 링크를 다른 부마스터(Sub-Master)와의 링크로 바꾸는 작업을 통하여 수행된다. 한편 작업자간의 통신을 마스터에 공유 변수를 두어 그 변수값을 갱신 및 참조하며 수행하는데, 이는 작업자의 수가 많은 상황에 대비하여 마스터의 부하를 줄이기 위해 그 마스터를 분산시키는 의도와 배치된다. 즉, 많은 작업자들이 그 공유 변수를 참조하기 위해 마스터에 접근한다면 마스터의 부하가 가중된다. 그러므로, 마스터의 중재없이 작업자간에 통신을 하는 것이 필요하다.

이 논문에서 구현한 웹 기반 병렬 시스템과 유사하게 웹을 통하여 병렬 처리를 수행하고자 하는 목적으로 구현된 시스템들로서 Javelin[13], SuperWeb[14], JET[15], POPCORN[16] 등이 있다. Javelin은 웹 기반 병렬 시스템의 원형을 제시하며, 구축된 시스템에 광선 추적법(RayTracing) 및 메르젠(Mersenne) 소수의 탐색 프로그램을 탑재하여 그 수행 결과를 보였다. 또한, 낙관적 부하 전파(optimistic load propagation)에 기반한 간단한 부하 균등 기법을 제시하였지만, 형식적인(formal) 성능 분석은 제시하지 않았다. 그리고, 브라우저가 지닌 보안 정책 때문에, 작업자들 사이의 통신 및 작업자와 의뢰인 사이의 통신이 모두 소켓을 통하여 연결되므로 서버가 반드시 그 통신을 중재해야 하는 단점이 존재한다. SuperWeb은 기본적으로 Javelin과 비슷한 정책을 택하면서, 향상된 구조를 제시하고 서버에게 더 이상 작업자와 의뢰인들간의 통신 중재를 요구하지 않지만, 부하 균등에 대한 어떠한 알고리즘도 제시하지 않았다. JET는 서버/작업자 구성의 웹 기반 병렬 라이브러리를 구축하고, 웹 환경에서 동작하는 여러 시스템의 결함(fault)을 고려하여 서버 및 작업자들의 상태를 주기적으로 데이타베이스에 기록하는 방법을 이용하였다. 또한, 구축한 라이브러리를 이용하여 세일즈맨 여행 문제 및 암호문 해독 문제에 대한 성능 평가를 수행하였다. 하지만, 역시 인터넷의 다양한 환경 변화에 적응하는 부하 균등에 대한 알고리즘은 존재하지 않았다. POPCORN은 원격지에서 객체를 생성하여 작업을 수행한 후 결과를 돌려주는 Computelet 이라는 클래스를 정의하여, 이를 기반으로 프로그래밍 모델을 제시하고 있지만, 역시 부하 균등에 대한 알고리즘은 전혀 언급하지 않았으며, 결함이 감지되자마자 다시 그 작업을 다른 작업자에게 전해준다는 간단한 결함 해결책만을 제시하였다.

이 논문에서 구현한 시스템은 JPWM(Java Parallel Web Machine)이라 명명하였으며, JPWM은 Javelin과 유사한 의뢰인/서버/작업자 구성을 채택하여 웹 기반 병렬 시스템을 구성한다. 하지만, Javelin과는 다르게 RMI를 이용하여 의뢰인의 원격 참조(remote reference)를 작업자에게 전달 후, 직접 그 결과를 의뢰인에게 돌려주는 방법을 이용한다. 즉, 서버는 작업을 각 작업자들에게 할당한 후, 작업자들과 의뢰인 사이에 어떠한 통신 중재도 하지 않음으로써 서버의 부담을 크게 줄여줄 수 있다. 또한 제안하는 성능 기반 작업 할당 알고리즘 및 적응성 향상 기법을 통해 올바른 적응적 부하균등 해결책을 제시한다. 즉, 매우 가변적인 인터넷 환경에서 작업의 수행을 진행하는 도중에 발생하는 이질적인 작업자들의 실제 수행 능력 변화와 네트워크 지연시간의 변화 등에 적합하도록 작업의 분배량을 바꾸는 적응성을 지닌 태스크 할당 기법을 제시한다. 각 작업자들은 실제

시스템에서 수행하는 작업만을 목적으로 수행되는 작업자들이 아니며, 인터넷에 연결된 이질적인 컴퓨팅 자원들이므로 매우 심한 환경 변화를 겪게 되며 이에 대한 적응성은 반드시 필요한 요소이다.

표 1 관련 연구와의 비교

시스템 / 고려사항	JET	POPCORN	Javelin	Super Web	JPWM
통신 기법	Socket	RMI	Socket	언급없음	**Socket + RMI**
통신에 대한 서버 중재	필요	불필요	필요	불필요	**불필요**
부하 균등	×	×	○	×	○
적응성	×	×	×	×	○

3. JPWM의 설계

3.1 기본적 요소

JPWM은 의뢰인(Client), 작업자(Worker), 병렬처리 서버(Parallel Processing Server), 그리고 웹 서버(Web Server)의 4가지 요소들로 구성되어 있다.

- 의뢰인(Client) : 대량의 연산이 필요한 작업을 병렬처리 서버에게 의뢰하고, 작업의 수행에 필요한 인자값(arguments)을 제공한다. 작업의 수행이 시작되면 수행 작업의 결과를 받을 준비가 필요하다.
- 작업자(Worker) : 서버로부터 전체 작업의 일부분인 태스크를 할당받아 수행한다. 태스크는 애플릿이 수행하며, 태스크 수행 종료 즉시 의뢰인에게 그 결과를 보내준다.
- 병렬 처리 서버(Parallel Processing Server) : 의뢰인과 작업자의 등록 및 관리를 수행하며, 의뢰인과 작업자간의 통신을 중재한다.
- 웹 서버(Web Server) : 작업자는 자바 애플릿에 의해 태스크를 수행하므로, 이 애플릿을 포함하는 HTML 문서를 공급해줄 수 있는 웹 서버가 필요하다. 이 웹 서버는 병렬 처리 서버와 같은 호스트에 존재해야 한다.

3.2 요소들간의 통신 프로토콜

그림 1은 의뢰인, 작업자와 서버와의 통신 프로토콜을 나타낸다. 의뢰인과 작업자는 병렬 처리 서버와 같은 호스트에 존재하는 웹 서버에 접속하여 HTML 문서를 브라우저로 받는다. 이 문서에는 병렬 처리를 위한 애플릿이 포함되어 있다. 의뢰인이 받은 애플릿은 의뢰인의 인터넷 주소 등 간단한 정보들을 서버에게 보냄으로써 준비 상태임을 알린다. 그리고 나서 서버로부터 의뢰인 식별자를 받는다. 작업자가 받은 애플릿은 간단한 성능 평가 연산(benchmark)을 수행하여 자신의 연산 능력을 서버에게 알린다. 성능 평가 연산은 가우스 소거법에 의거한 선형방정식의 해답을 구하는 연산으로 그 능력을 MFLOPS(mega-floating point operation per second)로 전환하는 전통적인 방법[17, 18]을 이용한다. 서버는 이를 토대로 각 작업자들에게 나누어줄 태스크의 양을 결정할 것이며, 작업자에게 고유의 식별자를 부여하여 알려준다.

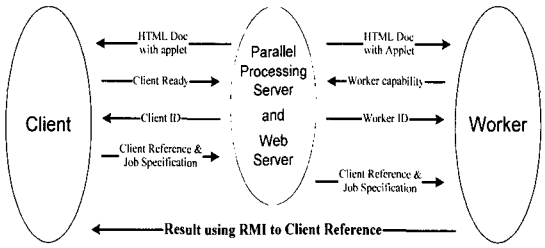


그림 1 시스템 통신 프로토콜

여기까지의 통신은 임의의 시점에서 의뢰인이나 작업자들이 동시에 수행될 수 있다. 이러한 과정이 끝난 후, 의뢰인은 작업에 관련된 명세를 서버에게 제출한다. 서버는 이 명세를 받아 각 작업자가 보낸 성능 평가 정보를 활용하여 성능비대로 작업을 분할한 뒤 거의 같은 시간에 전체의 작업이 끝나도록 각 작업자들에게 배분한다. 분할된 태스크는 작업의 명세와 함께 각 작업자들에게 보내지고, 이를 받자마자 작업자들은 연산 수행을 시작한다. 한편 의뢰인이 작업 명세를 서버에게 보낼 때, 자신의 원격 참조 정보(Client Remote Reference Information)를 같이 보내는데 이것은 다시 각 작업자들에게 작업의 명세가 보내질 때 같이 전달된다. 이는 작업자가 작업 명세를 받아 작업을 수행한 후 그 결과를 곧바로 의뢰인에게 보내기 위해 필요한 정보이다. 자바의 애플릿은 보안 정책인 원천 호스트(host-of-origin) 정책 때문에, 자신이 애플릿을 받아온 서버로만 직접 소켓 연결을 허용한다. 그러므로, 원격 참조 정보 및 RMI를 이용하여 임의의 호스트에 상주하는 의뢰인 객체의 메소드를 호출하는 콜백(callback) 방법을 이용하여 직접 의뢰인과 통신을 한다. 이를 통해 서버가 작업자의 결과를 다시 중재하고 관리하는 부담을 덜어주며, 동시에 의뢰인은 보다 빠른 응답을 받을

수 있는 이점을 얻을 수 있다.

3.3 시스템 구성

본 시스템은 총 3개의 층으로 구성된다. 가장 하위층에는 세션(Session)층이 존재하며, 중간층에는 병렬 처리(Parallel Process)층이 존재하며, 마지막으로 가장 상위층에는 응용(Application)층이 존재한다. 세션층은 기존에 JAVA에서 제공하는 Socket과 ServerSocket 및 이들로부터 얻어내는 입력스트림(InputStream)과 출력스트림(OutputStream)을 함께 하나의 클래스인 Session의 구성원으로 묶어서 상위층에게 일관된 관점을 주며, 좀 더 향상된 기능을 이용할 수 있도록 해준다.

병렬 처리층은 이 시스템이 요구하는 프로토콜을 구현하며, 특히 서버에서는 의뢰인으로부터 온 작업을 현재 등록된 여러 작업자들에게 분배하는 역할을 수행한다. 이미 구현이 완성된 일반 클래스도 존재하지만, 상위의 응용층의 클래스에서 상속을 하고, 추상 메소드의 구현을 완성하여 이용하는 추상 클래스 및 인터페이스도 존재한다. 병렬 처리층은 의뢰인 부분, 작업자 부분 및 서버 부분으로 나뉜다.

다음은 각 클래스 및 인터페이스에 대한 설명이다.

- 의뢰인 부분

abstract class JPWMClientApplet : 서버와의 세션연결을 구축하여 자신을 서버에 등록

interface JPWMClient : JPWMClientApplet의 추상 메소드와 동일한 메소드를 선언

interface JobHandler : 작업 명세를 다루는 메소드들을 선언하여 응용층에게 공개

class ResultReceiver : 작업자가 보내는 결과를 받는다.

- 서버 부분

class JPWMServer : 의뢰인과 작업자의 등록 및 관리

class JobSpecBroker : 의뢰인이 보낸 작업 명세를 받아 JobScheduler에 넘김

class JobScheduler : 스케줄링 후에 등록된 작업자들에게 해당 명세를 보냄

- 작업자 부분

abstract class JPWMWorkerApplet : 서버와의 세션연결을 구축하여 자신을 서버에 등록

interface JPWMWorker : JPWMWorkerApplet의 추상 메소드와 동일한 메소드를 선언

class JobSpecProcessor : 서버로부터 작업 명세를 받아 JobHandleProxy에게 전달 JobHandleProxy에게 결과를 받아 의뢰인에게 전달.

abstract class JobHandleProxy : 해당 문제에 관련된 알고리즘을 수행 후 결과 추출

interface JobHandleMediator : JobHandleProxy의 추상 메소드와 동일한 메소드를 선언

class BenchMarker : 작업자가 로드된 호스트의 성능을 평가

가장 상위층인 응용층은 이 시스템을 이용하는 각 응용 문제에 따라 다르게 구현해야 하는 부분이다. 우선 병렬 처리층에서 제공하는 각 추상 클래스들을 상속받으면서, 동시에 관련 인터페이스를 구현하는 새로운 클래스들을 구성한다. 주어진 문제를 해결하기 위해 만든 기존의 일반 코드를 분석하여 새로 구성된 클래스의 적절한 위치로 배분한다.

프랙탈 이미지 처리를 위한 시스템 클래스 구성 및 관계도는 그림 2와 같다.

4. 성능 기반 태스크 할당

인터넷상의 호스트들은 매우 다양하며 모두 다른 성능을 보여준다. 그러므로, 하나의 작업을 여러 태스크들로 나누어 서로 다른 호스트들이 병렬적으로 수행할 때 일반적으로 전체 작업의 수행 시간은 작업 수행 시작 이후 태스크로의 분할과 여러 작업자 호스트로의 할당이 이루어지고, 가장 나중에 응답을 보내주는 시간까지일 것이다. 따라서 각 작업자들의 성능을 고려하여 적절한 분배가 이루어지지 않는다면 성능이 낮은 작업자의 늦은 응답으로 인하여 전체 작업 시간이 길어지게 될 것이다. 각 작업자들의 성능에 기반하여 태스크들을 분배함으로써, 모든 작업자들이 태스크를 할당받은 이후 특정 작업자로 인하여 전체 작업에 대한 수행 완료 시간이 길어지지 않도록, 각 작업자들이 같은 시간 내에 각자의 태스크들을 모두 처리할 수 있도록 해야 한다. 이 논문에서는 각 태스크간의 종속 관계는 고려하지 않고, 각 작업자들이 독립적으로 할당받은 태스크들을 수행한다고 가정한다. 서버는 주어진 작업을 일정한 크기를 지닌 태스크들로 나누며, 각 작업자들은 성능비율에 따른 개수의 태스크들을 할당받아 수행한다. 또한, 주어진 작업의 수행이 시작된 후 이 수행에 참여하는 작업자의 수는 작업의 완료시점까지 일정함을 가정한다. 표 2는 이 논문에서 제안하는 알고리즘의 분석과 성능 평가에 사용되는 기호들이다.

4.1 성능 기반 태스크 할당 기법

제안하는 알고리즘은 인터넷 상의 이질적인 작업자들

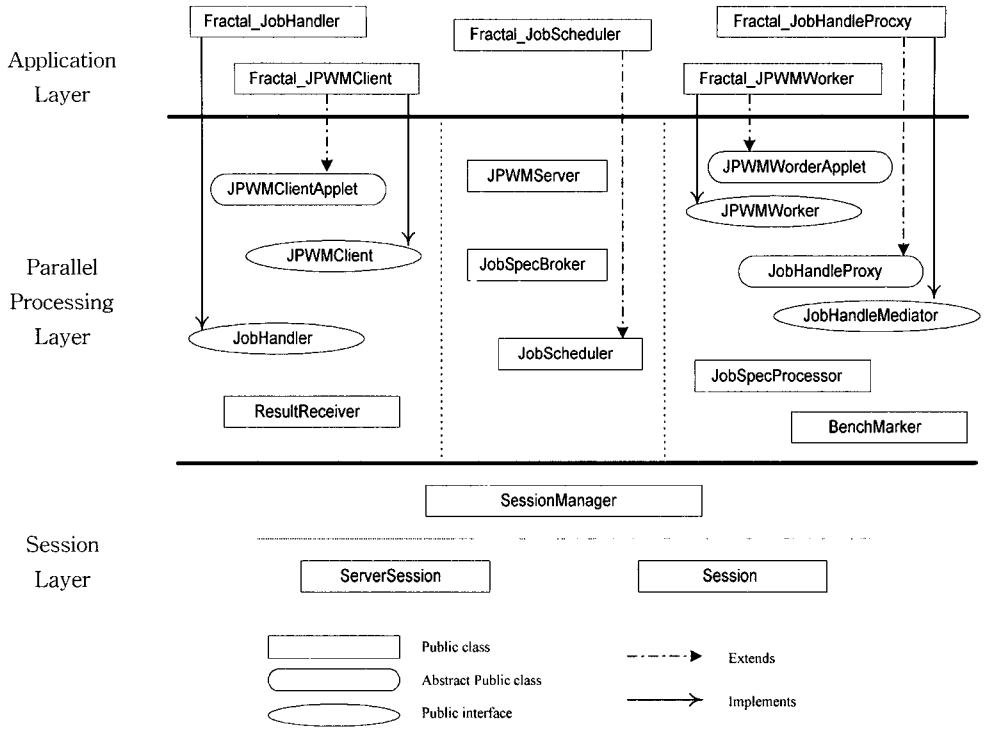


그림 2 프랙탈 이미지 처리를 위한 JPWM 클래스 구성도

표 2 성능 평가와 분석에 사용되는 기호들

기 호	의 미
	작업자 i
W_i	주어진 작업의 총 연산량
U	서버에 의해 나누어진 총 태스크 수
$TotalNumTask$	주어진 작업을 처리하는 작업자들의 수
$NumW$	시간 t 에 W_i 에 할당되는 태스크 수
$NumTaskW_i(t)$	시간 t 에 W_i 의 성능 (MFLOPS)
$P_i(t)$	의뢰인에서 서버까지 메시지 전송 시간
$TimeCS$	서버로부터 W_i 까지 메시지 전송 시간
$TimeSW_i$	W_i 로부터 서버까지 메시지 전송 시간
$TimeW_iS$	W_i 로부터 의뢰인까지 메시지 전송 시간
$TimeW_iC$	W_i가 할당받은 모든 태스크들을 수행하는 시간
$PTimeW_i$	서버가 태스크들을 각 작업자들에게 할당하는 시간
$TimeS_p$	작업의 총 수행시간
$TotalPSTime$	

중에 수행 능력이 좋은 작업자에게는 좀 더 많은 태스크량을 할당하며, 수행 능력이 나쁜 작업자에게는 할당되는 태스크 양을 줄임으로써 전체 작업의 수행 시간을 최적화한다는 간단하고도 직관적인 생각에서 비롯된다.

이를 위해 본 시스템은 (1) 작업자들이 서버에 연결하여 애플릿을 가져간 후 (2) 서버에게 자신을 등록할 때 스스로 자체 작업 수행 능력 평가를 수행하여 그 결과를 서버에게 보내준다(그림 1 참조). 각 작업자들이 수행하는 능력 평가는 행렬을 이용한 선형 방정식($A\vec{x} = \vec{b}$)을 푸는 문제가 이용된다. 이와 같은 방법은 Linpack Benchmark로 수년동안 컴퓨터의 부동 소수점 연산능력을 평가했던 고전적 방법이다[17, 18]. 방정식의 해는 부분 피벗팅을 이용한 가우스 소거법을 이용한다. 이러한 계산을 하면서, 단위 시간당 부동 소수점 덧셈과 곱셈 연산의 수행 횟수를 검사하여, 성능 평가 결과로 초당 몇 백만번의 부동소수점 연산을 수행하는지에 대한 MFLOPS 단위의 값을 돌려준다. 작업자들의 성능을 평가하기 위한 수단으로서의 벤치마킹 기법은 여러 가지 기법이 있을 수 있으나 가능하면 수행 시간이 가능하면 적으며, 애플릿의 크기가 작은 것이 필요하다. 또한 보다 정확한 성능 평가를 위해서는 단순한 연산 수행 결과를 이용하기 보다는 일반적으로 검증된 기법을 이용하는 것이 바람직하다. 이에 따라 이 논문에서

는 고전적으로 이용되어 오던 Linpack 벤치마킹을 선택하였으며, 이외에 다른 기법을 이용하는 것도 가능하다.

(3) 서버는 이러한 성능 평가 정보를 각 작업자마다 유지하고, 전체 작업을 일정한 크기를 지닌 태스크들로 나눈다. 그리고 (4) 성능 평가 정보를 바탕으로 각 작업자의 성능비에 따라 각 작업자들이 처리할 태스크의 개수를 정한다. (5) 각 작업자들은 서버로부터 할당받은 태스크들의 개수대로 스레드를 생성시켜 독립적으로 각 태스크를 수행한다. 이는 각 작업자들의 CPU 활용을 극대화시키고 전체 작업의 최종 완료 시간을 줄이고자 함이다.

다음 절에서는 작업 개시 전에 이러한 MFLOPS 정보를 이용하여 작업을 할당하고, 작업의 완료 시까지 할당된 태스크는 변하지 않는 성능 기반 태스크 할당(Performance-based Task Allocation) 기법에 대한 분석을 보인다. 그리고 5장에서 인터넷의 극심한 변화 가능성에 적응력 있는 태스크 할당 기법을 제안하고 분석한다.

4.2 성능 기반 태스크 할당 기법 분석

모든 작업자들은 서버에게 등록할 때 자신의 성능 평가 결과를 함께 전해주며, 이를 바탕으로 전체 작업이 각 작업자들에게서 거의 같은 시간에 완료되도록 태스크들을 할당받는다. 의뢰인이 작업을 개시하여 서버가 메시지를 받았을 때의 시간을 t_0 라고 가정하고 그 시간에 서버에 등록된 각 작업자의 성능 $P_i(t_0)$을 이용하여, 전체 작업이 같은 크기의 태스크들로 분할된 후의 태스크 개수 $TotalNumTask$를 $P_i(t_0)$의 비율로 각각 나누어 각 작업자들에게 태스크들을 할당한다. 즉, 작업자 W_i에게 할당된 태스크의 개수는

$$NumTaskW_i(t_0) = \left(\frac{P_i(t_0)}{\sum_{k=0}^{NumW-1} P_k(t_0)} \right) \cdot TotalNumTask \qquad (1)$$

로 결정된다. 이 때, 임의의 W_i가 할당받은 모든 태스크를 처리하는 시간을 $PTimeW_i$ 라하고, 서버가 이러한 성능 기반 태스크 할당을 수행하는 시간을 $TimeS_P$라고 가정하자.

성능 기반 태스크 할당에서 의뢰인이 작업 요청 개시 후 모든 결과를 받을 때까지의 총 수행 시간 $TotalPSTime$은 다음으로 정의된다.

$$TotalPSTime = TimeCS + TimeS_P + MAX\{T_0, T_1, T_2, \ldots, T_{NumW-1}\} \qquad (2)$$

이 때, $T_i = TimeSW_i + PTimeW_i + TimeW_iC$

식 (2)는 성능 기반 태스크 할당에서 $TotalPSTime$을 줄이기 위해 $MAX\{T_0, T_1, T_2, \ldots, T_{NumW-1}\}$을 줄여야 함을 보여준다. 인터넷 환경에서 $TimeCS$, $TimeSW_i$, $TimeW_iC$ 들은 태스크 할당에서 조절 가능한 변수들이 되지 못한다. 따라서 $PTimeW_i$가 $TotalPSTime$를 결정하는 가장 중요한 요소가 됨을 알 수 있다.

한편, $U/TotalNumTask$는 단위 태스크의 크기가 되므로 임의의 시간 t_0 에서 임의의 W_i 에 할당되는 태스크들이 지닌 총 연산량은

$$\frac{U}{TotalNumTask} \cdot NumTaskW_i(t_0) \qquad (3)$$

이 된다. 따라서 $PTimeW_i$ 는 (1)과 (3)에 의해 다음과 같이 결정된다.

$$PTimeW_i = \frac{U \cdot NumTaskW_i(t_0)}{TotalNumTask \cdot P_i(t_0)} = \frac{U}{\sum_{k=0}^{NumW-1} P_k(t_0)} \qquad (4)$$

식 (4)는 이 태스크 할당 기법에 의해 모든 작업자들이 동일한 시간에 할당받은 작업들을 처리할 수 있음을 보여준다. 즉, 최적의 작업 수행 시간을 얻기 위한 태스크 할당 목적에 부합된다. 그러나, 식 (4)는 작업자의 성능이 처음에 평가된 후 작업 수행 중에 그 성능이 고정적으로 변하지 않는다는 가정 하에서만 적용된다. 따라서 시간에 따라 작업자의 성능이 크게 변할 수 있는 웹 환경의 호스트들에게는 식 (4)의 $PTimeW_i$ 에 예측할 수 없는 가변적인 ΔT_i가 더 추가될 수밖에 없다.

[정의 1] 성능 기반 태스크 할당에서 임의의 W_i 가 할당받은 모든 태스크를 처리하는 시간 $PTimeW_i$ 는 다음과 같이 정의된다.

$$PTimeW_k = \frac{U}{\sum_{i=0}^{NumW-1} P_i(t)} + \Delta T_i \qquad \square$$

작업 수행 개시 이후 호스트의 성능이 오히려 좋아지는 경우부터 호스트가 고장을 일으켜서 응답을 못 보내는 경우까지 고려해 보았을 때, ΔT_i 의 범위는 임의의 음수 값에서 무한대(+∞)까지 일 수 있다. 성능 기반 태스크 할당에서 $TotalPSTime$ 은 가장 큰 ΔT_i 값을 보이는 호스트에 전적으로 의존된 값을 보인다.

5. 적응성 향상 기법

5.1 적응적 성능 기반 태스크 할당

웹에서 의뢰인이 개시한 작업을 대신 수행해 주는 작업자 호스트들은 지리적으로 멀리 떨어져 있을 수 있으며, 여러 사용자들에게 개방되어 있기 때문에 그 성능에 대해 예측 불가능한 성질을 보여준다. 호스트 사용자들이 그 호스트들을 이용하는 정도에 따라 그 호스트에 상주하는 프로세스의 수가 증감될 수 있다.

따라서 단순하게 의뢰인이 작업을 개시하는 시점에 서버가 지닌 각 작업자의 성능 정보만을 이용해서 작업을 할당한다면, 시간이 흐름에 따라 변하는 각 호스트의 수행 능력 변화에 대처할 수 없다. 더욱이 수행 도중에 작업에 참여하는 어느 한 호스트에 결함이 발생하여 응답을 하지 못하는 상황에 대처하지 못하고, 의뢰인이 결함이 발생한 작업자의 응답을 무한히 기다려야하는 상황이 발생할 수 있다. 그러한 수행능력 변화 및 결함을 인식하여 태스크들을 재할당하여 준다면 더욱 효율적인 성능을 기대할 수 있다. 이 절에서는 적응적으로 각 호스트의 수행능력을 인식하여 태스크를 재할당하는 적응성 향상 기법(Performance-based Adaptive Task Allocation)을 제안한다.

Ω : 작업자들의 집합
W_c : Complete 메시지 보낸 작업자
Π_i : 작업자 i의 미수행 태스크의 집합
Γ : 태스크들의 집합
Γ_i : 작업자 i에게 할당되는 태스크들의 집합
Complete: 임의의 작업자가 할당받은 태스크를 모두 처리한 후 서버에게 보내는 메시지
Stop : 서버가 임의의 작업자에게 작업 수행을 종료하라는 메시지

Server

$\forall W_i \in \Omega$, set Γ_i and allocate Γ_i to W_i ;
$\Gamma \leftarrow \varnothing$;
isReceiveAll = True ;
while ($\bigcup_{i=0}^{NumW-1} \Pi_i = \varnothing$) {
  receive(Complete) and reset W_c ;
  $\forall$($W_i \in \Omega$ and $W_i \neq W_c$) send(Stop) to W_i ;
  $\forall W_i \in \Omega$, receive Π_i ;
  $\Gamma \leftarrow \Gamma \cup \bigcup_{i=0}^{NumW-1} \Pi_i$;
  $\forall i$, reset Γ_i as the ratio of size of Π_i ;
  $\forall W_i \in \Omega$, allocate Γ_i to W_i ;
}

그림 3 적응적 태스크 할당 알고리즘

그림 3은 서버에서 수행되는 알고리즘으로서 4장에서 설명한 태스크 할당 기법에 적응력이 추가된 적응적 태스크 할당 알고리즘이다. 기본적으로 최초에는 4장의 성능 기반 태스크 할당 기법을 이용한다. 하지만, 임의의 작업자가 자신에게 할당된 모든 태스크를 마치면, 서버에게 Complete 메시지를 보낸다. 서버는 Complete 메시지를 받으면 이 메시지를 보낸 작업자를 제외한 모든 작업자들에게 Stop 메시지를 보낸다. Stop 메시지를 받은 작업자들은 작업을 중단하고 아직 처리하지 못한 태스크들의 대한 정보(태스크 식별자)를 보내준다. 서버가 모든 작업자들로부터 이러한 미수행 태스크들에 대한 정보를 받으면, 모든 작업자들에게 지금까지 수행한 태스크의 개수의 비에 따라 그 미수행 태스크들을 재할당한다.

5.2 적응성 향상 기법의 성능 분석

제시된 알고리즘에서 최초의 할당을 포함하여 서버에 의해 태스크를 재할당하는 횟수를 ν라고 하였을 때, ν는 호스트의 성능 변화에 따라 달라진다. 즉, 작업자들 간의 성능 변화가 클수록 각 작업자간의 작업 부하를 균등하게 맞추기 위한 재할당이 많이 이루어진다. 한편, 서버에서 작업자 W_i로, 작업자 W_i에서 서버로의 메시지 전송 횟수인 τ_1과 τ_2는 그림 3의 알고리즘에 의해 다음과 같이 정해진다.

$$\begin{cases} \tau_1 = \nu + \nu_i', & \text{이 때 } \nu_i'\text{는 } W_i\text{가 stop 메시지를 받은 횟수} \\ \tau_2 = \nu \end{cases}$$

임의의 W_i가 태스크를 처리하는 모든 시간을 $APTimeW_i$로, 서버가 할당할 태스크들을 결정하는 시간을 $TimeS_{AP}$로 하고, 적응적 성능 기반 태스크 할당을 이용할 때 의뢰인이 작업개시 후 모든 결과를 받을 때까지의 총 수행 시간 $TotalAPSTime$은 다음과 같이 얻어질 수 있다.

$$TotalAPSTime = TimeCS + \nu TimeS_{AP} + MAX\{T_0, T_1, T_2, \ldots, T_{NumW-1}\} \quad (5)$$

이 때, $T_i = \tau_1 TimeSW_i + \tau_2 TimeW_iS + APTimeW_i + TimeW_iC$

식 (5)를 살펴볼 때, 식 (2)와 같이 $MAX\{T_0, T_1, \ldots, T_{NumW-1}\}$을 줄이는 것이 총 수행 시간 $TotalAPSTime$을 줄이기 위해 필요하다.

임의의 z번째 재할당되는 시점 t_z에서 미수행된 작업의 총 연산량을 $U(z)$라 하고, 그 때 W_i의 단위 시간당 연산 수행량인 성능을 $P_i(t_z)$라 하자. $P_i(t_z)$는 $z-1$번째에 작업자 W_i가 이미 수행한 태스크의 양을 이용하여 정해진다. 이 때 $APTimeW_k$는 식 (4)를 이용해 다음과 같이 정의된다.

[정의 2] 적응적 성능 기반 태스크 할당에서 임의의 W_i 에 할당되어진 모든 태스크를 처리하는 시간

$APTimeW_i$ 는 다음으로 정의된다.

$$APTimeW_i = \sum_{z=0}^{k-1} \frac{U(z)}{\sum_{k=0}^{NumW-1} P_k(t_z)} \quad \square$$

[정의 2]에는 **[정의 1]**에서처럼 작업자의 성능 변화에 따른 예측 불가능한 값 ΔT_i가 포함되어있지 않다.

하지만 재할당을 위하여 $TotalPSTime$ 보다 $TotalAPSTime$에 더 추가되는 요소들이 있다. 각 작업자 W_i마다 서버와의 메시지 교환을 위해 (τ_1-1) $TimeSW_i + \tau_2 TimeW_iS$가 더 필요하며, 서버에서는 태스크 재할당량을 결정하기 위하여 $\nu\, TimeS_{AP}$-$TimeS_P$가 더 필요하다. 이 둘을 함께 $MoreTimeW_i$이라고 할 때, $MAX\{PTimeW_i - APTimeW_i\}$가 $MAX\{MoreTimeW_i\}$ 보다 크다면 적응성 향상 기법은 전체 응답 시간을 단축시킬 수 있다.

작업 개시 후 각 작업자의 성능을 변하지 않고 일정하게 유지시킬 수 있도록 각 호스트들의 시간에 따른 환경 변화를 직접 제어 가능하다면 적응성 향상 기법이 오히려 네트워크와 서버에 더 많은 부담을 주지만, 만약 서비스 개시 이후에 그 성능이 예측할 수 없이 자주 변할 수 있으며 제어 가능하지 않은 웹에 연결된 호스트들을 작업자로 이용하는 경우에 극심한 환경 변화에 대처할 수 있는 적응성 향상 기법을 이용하여 전체 응답 시간을 크게 줄일 수 있다.

6. 성능 평가

본 연구의 성능 평가를 위하여 구현된 시스템에 프랙탈 이미지 처리를 수행하는 작업을 응용층에 적용하였다. 프랙탈 알고리즘은 간단한 복소변환($z = z^2 + c$) 의 규칙이 복잡한 구조를 만들어 내는 맨델브로트(Mandelbrot) 방법을 사용한다[19]. 그림의 크기는 256 ×256 픽셀로 하고 알고리즘의 특성상 각 라인마다 독립성이 유지됨을 이용하여, 256개의 라인을 4 라인씩 묶어서 하나의 태스크로 만든다. 병렬 처리와 웹 서버는 SunSPARC 20에서 수행하고, 각 작업 호스트는 SunSPARC 20, 10 및 펜티엄 233, 200, 150 MHz등 다양한 기종들에서 시험되었으며, 각 호스트는 한국의 학술망에 연결되어 지역적으로 약 400킬로미터 이상 떨어진 두 곳에 위치해 있다. 하지만, 이러한 기종의 차이보다는 벤치마크에 의한 성능평가 수치(MFLOPS)가 더 실질적으로 신뢰성이 있는 자료이다.

그림 4와 5에서는 클라이언트의 화면을 보여준다. 브라우저에서 클라이언트가 작업을 요청한 뒤 작업자들로부터 태스크 수행 결과를 일부 받아온 상태가 그림 4이며, 모든 작업이 완료된 후의 상태가 그림 5이다.

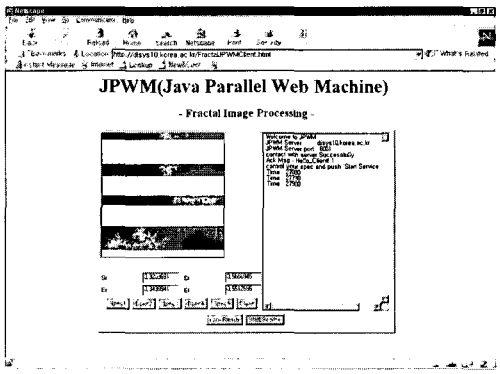


그림 4 작업이 진행 중인 과정

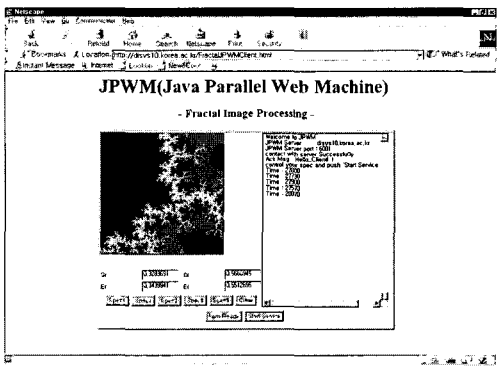


그림 5 작업 완료

6.1 작업자 수의 증가에 따른 성능 향상

첫 번째 성능 평가는 작업에 참여하는 작업자의 수에 따른 성능 향상을 실험하여 제시한다. 이 결과(그림 6)는 [13, 14]에서 수행한 결과와 유사하다. 즉, 작업자의 수가 증가할수록 성능 향상을 얻을 수 있지만 선형적이지는 않다. 이러한 성능 향상율은 응용 문제에 따라 다르게 나타날 수 있다.

6.2 균등 기반과 성능 기반 태스크 할당

두 번째 성능 평가에서는 작업자의 수를 4대로 고정시키고, 성능 기반 태스크 할당 기법을 이용했을 때의 성능 향상 정도를 알아본다. 균등 태스크 할당 기법을 이용해 모든 작업자에게 동일한 수의 태스크가 할당되도록 한 결과와 적응성이 없는 성능 기반 태스크 할당 기법을 이용한 결과를 비교한다. 할당된 태스크들은 각

작업자에서 독립적으로 수행되어 그 결과를 즉시 의뢰인에게 보내게 된다. 각 작업자의 MFLOPS 검사 결과와 각 방법에 따른 작업 호스트의 할당 태스크의 수를 표 3에 제시하였다.

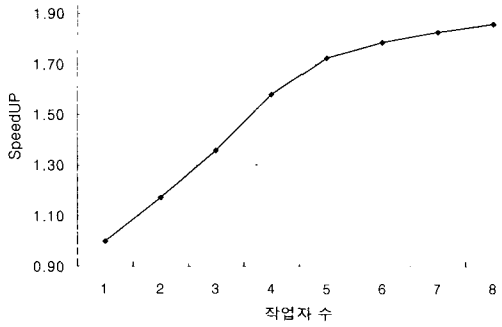


그림 6 작업자의 수 증가에 따른 성능 향상

표 3 작업 호스트의 성능에 따른 할당 태스크의 수

작업자 ID	MFLOPS	태스크 수	
		균등	성능 기반
1	1.503	16	26
2	0.517	16	9
3	0.843	16	14
4	0.847	16	15

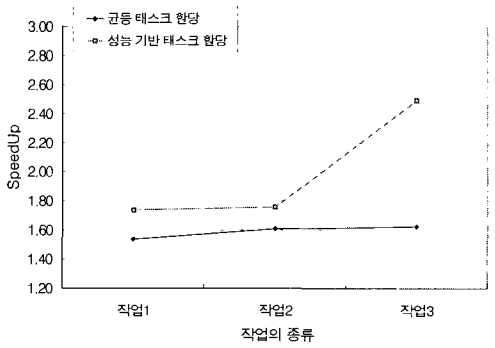


그림 7 균등과 성능 기반 태스크 할당의 SpeedUp 비교

그림 7은 균등 태스크 할당과 성능 기반 태스크 할당 기법간의 응답 시간 차이를 보여준다. 각 작업들은 그 순서대로 더 많은 계산량을 수행하도록 구성된다. 작업자 4대의 성능비가 약 0.358의 표준 편차를 보일 때, 균등 태스크 할당보다 성능 기반 태스크 할당이 20.18% 성능 향상을 보임을 알 수 있다. 또한 계산량이 상대적으로 많은 작업 3은 34.71%의 높은 성능 향상을 보여준다.

6.3 적응적 태스크 할당 기법의 성능 평가

세 번째 성능평가는 적응적 성능 기반 태스크 할당 기법의 효율성을 보이기 위하여 수행되었다. 두 번째 성능평가와는 달리 각 작업자들에게 10초보다 적은 임의의 시간동안 임의의 연산을 수행하는 스레드를 10개 이내로 임의의 간격마다 실행되도록 함으로써 작업자에 임의의 부하(CPU Worm)를 줌으로써 성능 변화가 발생하는 모의 상황을 만들었다. 그림 ?에서와 같이 이러한 환경에서의 성능 기반 태스크 할당 기법(PTA)은 성능이 비교적 일정하게 유지되는 환경보다 훨씬 긴 응답시간을 보여주었다. 하지만, 적응적 태스크 할당 기법(PATA)을 이용한 경우 오히려 성능이 일정하게 유지되는 환경에서 적용한 성능 기반 태스크 할당 기법보다 더 빠른 응답시간을 보여준다. 이것은 CPU의 부동 소수점 계산 능력이 메모리 I/O 등을 포함한 작업자의 전체 성능을 완전히 대변하지 못한다는 것도 보여준다. 각 작업자의 성능이 일정한 환경에서 적응적 태스크 할당 기법은 단순한 성능 기반 태스크 할당 기법보다 13.07% 성능 향상을 보이며, 작업자의 성능이 수행 도중 변하도록 만든 환경에서 적응적 태스크 할당 기법은 단순 성능 기반 태스크 할당 기법보다 17.38% 성능 향상을 보인다. 그림 9에서 작업의 종류 1, 2, 3의 순서대로 전체적인 계산량이 많은 작업들이다.

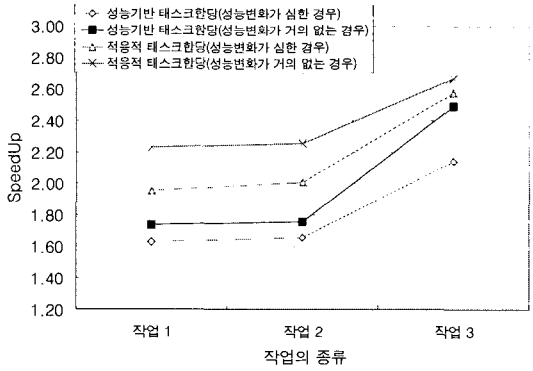
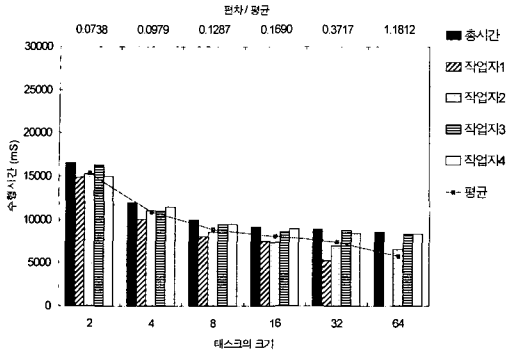
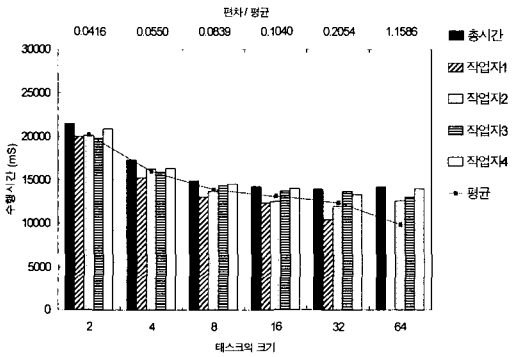


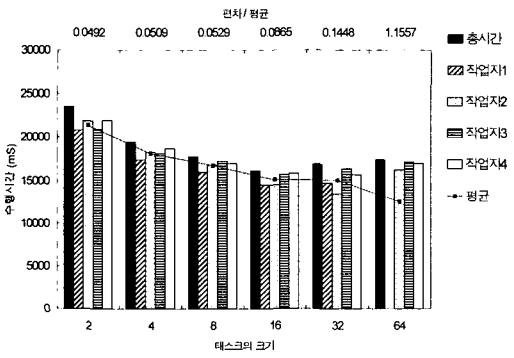
그림 8 성능기반과 적응적 태스크 할당의 태스크 할당의 SpeedUp 비교



(a)



(b)



(c)

그림 9 태스크 입도에 따른 수행시간 변화

6.4 태스크 입도에 따른 적응적 태스크 할당 기법

다음 네 번째 성능 평가는 각 작업자에게 할당하는 태스크의 입도 변화에 따른 총 수행 시간 변화를 알아보기 위해 수행한다. 그림 9에서는 4개의 작업자에게 적응적 태스크 할당 기법을 적용하면서 각 태스크의 크기를 다르게 한 결과를 보여준다. 그림 9의 (a)는 작업 1, 즉, 계산량이 비교적 적은 태스크들을 수행시킬 때이며, (b), (c)의 순서로 태스크에 포함된계산량이 많은 작업 2 작업 3을 수행한 결과를 보여준다. 세 그림 모두에서 보이듯이, 태스크의 크기가 작은 경우 통신 회수가 많아짐으로써 태스크의 크기가 큰 경우보다 보다 좋지 않은 결과를 볼 수 있다. 한편, 각 태스크에 포함된 계산량이 많은 경우에, 통신 회수를 줄이기 위해 태스크의 크기를 크게 하는 것은 가변적인 성능 변화를 반영하지 못하여 오히려 좋지 못한 결과를 가져온다. 각 작업자들의 개별적인 수행 시간을 고려해 보았을 때, 각 작업자들의 개별적인 수행 시간들의 평균에 대한 분산 정도를 보면 태스크 크기가 작은 경우가 더 작은 값을 가짐을 알 수 있다. 즉, 평균적으로 태스크의 크기가 작은 경우 각 작업자들은 쉬는(idle) 시간 없이 골고루 작업을 수행하는데 비해, 태스크 크기가 큰 경우 작업의 분배가 적절히 이루어지지 않음을 알 수 있다. 따라서, 통신 회수의 증가로 인한 성능 저하 및 각 태스크에 포함된 실제 계산량등을 고려한 적절한 태스크 입도(granularity)를 결정하는 것이 필요하다.

7. 논의 및 결론

우리는 웹 환경에 연결된 여러 자원을 이용하여 주어진 작업을 병렬로 처리할 수 있는 시스템인 JPWM의 설계 및 구현에 대하여 다루었다. JPWM은 기존에 이미 구현된 여러 동종의 시스템들과는 다르게 웹 환경에 알맞은 부하 균등 알고리즘을 지니는 시스템이다. 웹은 매우 많은 이기종들이 연결된 가상 시스템이 이기 때문에 그들간의 성능에 맞게 태스크를 분배하는 정책은 타당성 있는 접근 방법이다. 하지만, 예측할 수 없는 그들의 성능 변화에도 올바르게 대처할 수 있는 알고리즘이 필요하며, 그 알고리즘의 하나로 우리는 적응적 성능 기반 태스크 할당 기법을 제시하였다.

인터넷에 연결된 수많은 컴퓨터들에게 작업을 분배하고, 그들로부터의 결과를 의뢰자(client)에게 전달하는 등의 많은 일이 서버에 집중되어 성능 저하에 큰 영향을 주게 될 수 있다. 구현된 시스템에서는 이러한 서버의 부담을 줄이는 한가지 방법으로 서버가 작업의 결과를 취합하여 의뢰자에게 전달하는 중재 작업의 부담을 줄여, 작업자(worker)들이 직접 의뢰자에게 결과를 전달할 수 있도록 RMI(Remote Method Invocation)을 이용하도록 하였다. 자바의 보안 정책으로 인하여 애플릿은 자신이 원래 존재했던 곳으로만 네트워크 연결이 가능하지만, 그러한 보안에 문제를 발생시키지 않으면서

도 서버의 부담을 줄일 수 있도록 하였다. 또한 인터넷에 연결되어 있는 컴퓨터들은 매우 이질적이며, 그들의 성능 및 네트워크의 통신 지연 차이가 매우 심하게 변하는 상황을 보인다. 그러므로 이들을 어떤 공통의 작업을 위한 컴퓨팅 자원으로 이용하기에는 연산 결과에 대한 예측이 매우 힘들어지게 되는데, 이를 해결하기 위해서는 매우 심하게 변화는 상황에 대처할 수 있는 적응성이 요구된다. 본 논문에서는 이러한 적응성을 가진 작업의 분배 정책을 제안하고, 이를 통해 보다 빠른 작업의 수행 완료를 얻을 수 있음을 보였다.

우리가 차후 연구로서 더 고려할 사항은 먼저 작업자의 확보이다. 웹에서 브라우저를 이용하여 인터넷에 연결할 수 있는 전세계의 모든 호스트들이 잠재적인 작업자들이긴 하지만 얼마나 많은 호스트들이 서버에 접속하여 자신의 컴퓨팅 자원을 이용할 수 있도록 허용하는가도 문제가 될 것이다. 이에 대한 연구가 [16, 20]에서 다루어지고 있으며, 이에 대한 해결책이 더 연구되어야 할 것이다. 이를 통해 보다 많은, 즉 수백 내지 수천대의 호스트들이 이동되는 경우의 성능은 어떠할 것인가를 살펴보는 것이 필요하다.

그리고 이 논문에서 이용된 작업으로서 프랙탈 이미지 처리가 수행되었지만, 그 외 작업의 특성에 따른 성능 비교가 필요하다. 즉, 작업의 수행시간이 여러 일이나 여러 달 이상 필요한 긴 작업 시간을 갖거나, 서버로부터 작업자로의 메시지양과 작업자가 되돌려 주는 결과의 양이 다른 경우 그에 따른 영향 등을 살펴 보아야 할 것이며, 이렇게 특성이 다른 작업들에 따른 제안하는 적응적 성능 향상 기법의 성능 변화를 살펴볼 필요가 있다.

그리고 태스크들간에 종속관계가 존재하여 각 작업자들간에 통신이 필요한 경우에 대하여도 더 연구가 될 것이다. 현재 시스템은 작업자들이 서로 독립적인 태스크들을 수행하는 경우만을 고려하였지만, 작업자가 의뢰인에게 RMI를 이용하여 통신을 하듯이 서버가 적절하게 중재한다면 작업자들의 보안 문제를 발생시키지 않고 태스크들의 종속관계를 해결할 수 있을 것이다. 그러나 네트워크의 속도 및 안정성이 아직 확보되지 않은 현 단계에서는 지역적으로 매우 떨어져 있는 인터넷상의 작업자간의 통신을 가정하지 않은 문제를 해결하는데 더 적합하며, 데이타 통신량이 적으며 각 작업자의 컴퓨팅 자원을 더 필요로하는 응용에 보다 유용하다.

마지막으로 현재의 시스템에 추가되어 연구되고 있는 관련 연구로써 작업자들의 추가와 삭제이다. 현재의 시스템은 작업이 개시될 때 작업이 완료될 때까지 고정된 개수의 작업자들로만 작업을 수행하지만, 임의의 작업자에게 결함이 발생한 경우 이를 인식하고 작업자로서 더 이상 이용되지 않도록 하거나, 새롭게 작업자로서 추가된 호스트를 작업이 완료되지 않은 상태에서도 이용할 수 있도록 하는 연구가 진행 중이다.

참 고 문 헌

[1] R. A. Whiteside and J. S. Leichter, "Using Linda for Supercomputing on a Local Area Network," *Technical Report* YALEU/DCS/TR-638, Department of Computer Science, Yale Univ., New Haven, Connecticut, 1988

[2] 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

[3] K. M. Chandy, B. Dimitrov, H. Le, J. Mandleson, M. Richardson, A. Rifkin, P. A. G. Sivilotti, W. Tanaka, and L. Weisman, "A World-Wide Distributed System Using Java and the Internet," In *Proc. of the 5th IEEE Int'l Symposium on High Performance Distributed Computing*, Syracuse, NY, Aug., 1996.

[4] A. Vahdat, P.Eastham, C.Yoshikawa, E.Belani, T.Anderson, D.Culler, and M.Dahlin, "WebOS : Operating System Services For Wide Area Applications," *Technical Report* CSD-97-938, UC Berkeley, 1997.

[5] V. Sunderam, G. Geist, J. Dongarra, and R. Manchek, "The PVM concurrent system: Evolution, experiences, and trends," *Parallel Computing*, 1994.

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

[7] A. Ferrari, "JPVM - The Java Parallel Virtual Machine," In *the ACM Workshop on Java for High-Performance Network Computing*, Feb., 1998

[8] Kenji Imasaki, "JAPE : The Java Parallel Environment," In *the 1997 International Scientific Computing in Object-Oriented Parallel Environments Conference*, California, December, 1997.

[9] Michael Philippsen and Matthias Zenger, "JavaParty-Transparent Remote Objects in Java," In *the ACM Workshop on Java for Science and Engineering Computation*, 1997

[10] Nataraj Nagaratnam, Arvind Srinivasan, and Doug Lea, "Remote Objects in Java(tm)," In *the Proc. of IASTED International Conference on Networks*, Jan., 1996.

[11] Tim Brecht, Harjinder Sandhu, Meijuan Shan, and

Jimmy Talbot, "ParaWeb: Towards World-Wide Supercomputing," In *the Proc. of the 7th ACM SIGOPS European Workshop*, pp.181-188, Sep., 1996

[12] 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.

[13] 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

[14] A. Alexandrov, M. Ibel, K. E. Schauser, and C. Scheiman, "SuperWeb: Research Issues in Java-Based Global Computing," In the *Proc. of Concurrency: Practice and Experience*, Jun., 1997.

[15] 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.

[16] Noam Nisan, Shmulik London, Ori Regev, Noam Camiel, "Globally Distributedcomputation over the internet - The POPCORN project," *the 6th Int'l World Wide Web Conference*, Santa-Clara, Apr., 1997.

[17] J. Dongarra, J. Bunch, D. Moler, and G. W. Stewart, "LINPACK User's Guide," SIAM, Philadelphia, PA, 1979.

[18] J. Dongarra, "Performance of Varios Computers Using Standard Linear Equations Software," http://www.netlib.org/benchmark/performance.ps, Aug., 1998.

[19] R. L. Devaney, Chaos, Fractals, and Dynamics : Computer Experiments in Mathematics, Addison-Wesley, 1990.

[20] F. Soares, L. M. Silva, and J. G. Silva, "How to Get Volunteers for Web-based Metacomputing," In *Proc. of the Distributed Computing on the Web* (DCW'98), Rostock, Germany. Jun., 1998.

[21] 박지민, 명혜선, 한탁돈, 김신덕, 네트워크로 연결된 자바스테이션 상에서의 웹 컴퓨팅, 한국 정보과학회 봄 학술발표논문집, Vol. 25, No. 1, pp. 647-649, 1998.

한 연 희

1996년 고려대학교 수학과 졸업(학사). 1998년 고려대학교 대학원 컴퓨터학과 졸업(이학석사). 1998년 ~ 현재 고려대학교 대학원 컴퓨터학과 박사과정 재학중. 관심분야는 이동컴퓨팅, 분산시스템, 분산객체 등임.

박 찬 열

1993년 고려대학교 수학과 졸업(학사). 1995년 고려대학교 대학원 컴퓨터학과 졸업(이학석사). 1995년 9월 ~ 현재 고려대학교 대학원 컴퓨터학과 박사과정 재학중. 관심분야는 분산시스템, 이동컴퓨팅, 결함허용시스템 등임.

정 영 식

1987년 고려대학교 수학과 졸업. 1989년 고려대학교 대학원 석사학위취득(전산학). 1993년 고려대학교 대학원 박사학위취득(전산학). 1997년 1월 ~ 1998년 1월 미시간주립대학교 교환교수. 1993년 9월 ~ 현재 원광대학교 컴퓨터정보통신학부 조교수. 관심분야는 병렬분산처리, 멀티미디어 CAI, 컴퓨터 시뮬레이션 등임.

황 종 선

1978년 Univ. of Georgia, Statistics and Computer Science 박사. 1978년 South Carolina Lander 주립대학교 조교수. 1981년 한국표준연구소 전자계산실 실장. 1995년 한국정보과학회 회장. 1982년 ~ 현재 고려대학교 컴퓨터학과 교수. 1996년 ~ 현재 고려대학교 컴퓨터과학기술대학원 원장. 관심분야는 알고리즘, 분산시스템, 데이타베이스 등임.