# IBM Aglets를 기반으로 하는 가상 병렬 컴퓨팅 시스템에서 작업 할당 기법과 성능 비교

## (Work Allocation Methods and Performance Comparisons on the Virtual Parallel Computing System based on the IBM Aglets)

김 경 하 † 김 영 학 †† 오 길 호 ††
(Kyong-Ha Kim) (Young-Hak Kim) (Gil-Ho Oh)

**요 약** 최근에 다중 에이전트를 기반으로 하는 가상 병렬 컴퓨팅 시스템에 관한 적극적인 연구가 이루어지고 있다. 가상 병렬 컴퓨팅 시스템은 많은 계산을 요구하는 복잡한 문제들을 해결하기 위해 고비용 수퍼 컴퓨터를 사용하는 대신에, 인터넷상에 산재되어 있는 개인용 컴퓨터 혹은 워크스테이션을 사용한다. 가상 병렬 컴퓨팅 시스템은 인터넷상에 이용 가능한 자원에 의존하여 동질 혹은 이질형의 컴퓨터들로 구성될 수 있다. 본 논문에서는 IBM Aglets를 기반으로 하는 가상 병렬 컴퓨팅 환경에서 작업자 에이전트와 작업 패키지를 효율적으로 분산하는 새로운 방법을 제안한다.

기존의 방법들은 작업자 에이전트와 작업 패키지를 분산하기 위해서 주로 마스터/슬레이브 유형을 사용한다. 그러나 이러한 방법에서 에이전트의 수가 증가하게되면 중앙 마스터에서 작업부하가 급속하게 증가된다. 이러한 문제에 대한 해결로서 본 논문에서는 작업자 에이전트와 작업 패키지의 분산을 작업자 에이전트에게 위임하는 방법을 사용한다. 제안된 방법은 가상 병렬 컴퓨팅 시스템에서 다양한 방식으로 평가되었으며, 그 결과는 기존의 방법에 비해서 괄목할 만한 수준으로 개선되었다.

**키워드** : IBM Aglets, 조정자 에이전트, 작업자 에이전트, 가상병렬컴퓨팅 시스템

***Abstract*** *Recently, there have been active researches about the VPCS (Virtual Parallel Computing System) based on multiple agents. The VPCS uses personal computers or workstations that are dispersed all over the internet, rather than a high-cost supercomputer, to solve complex problems that require a huge number of calculations. It can be made up with either homogeneous or heterogeneous computers, depending on resources available on the internet. In this paper, we propose a new method in order to distribute worker agents and work packages efficiently on the VPCS based on the IBM Aglets.*

*The previous methods use mainly the master-slave pattern for distributing worker agents and work packages. However, in these methods the workload increases dramatically at the central master as the number of agents increases. As a solution to this problem, our method appoints worker agents to distribute worker agents and workload packages. The proposed method is evaluated in several ways on the VPCS, and its results are improved to be worthy of close attention as compared with the previous ones.*

**Key words** : IBM Aglets, Coordinator agent, Worker agent, Virtual parallel computing system

· 본 연구는 2000년도 금오공과대학교 학술연구비 및 기성회 기자재 지원에 의해서 연구되었음.

† 비 회 원 : 쌍용정보통신 NI기술팀 연구원
kimkha@sicc.co.kr
†† 종신회원 : 금오공과대학교 컴퓨터공학부 교수
kimyh@knut.kumoh.ac.kr
gilho@knut.kumoh.ac.kr

논문접수 : 2001년 2월 21일
심사완료 : 2002년 4월 29일

## 1. 서 론

컴퓨터 하드웨어 기술과 네트워크 기술이 급속하게 발전하면서 많은 계산을 필요로 하는 복잡한 문제를 해결하기 위해 고성능 수퍼 컴퓨터를 사용하는 대신에, 네트워크 상에 분산되어 있는 컴퓨터를 이용하여 병렬처리하는 많은 연구들이 진행되고 있다[1-6]. 이러한 시스템

은 네트워크 상에 분산되어 존재하는 컴퓨터들을 비용대 성능비가 높은 가상 병렬 컴퓨팅 환경으로 구성하고, 하나의 큰 작업을 분할하여 다수의 컴퓨터 상에서 효율적으로 분산처리 한다. 본 논문에서 고려한 가상 병렬 컴퓨팅 시스템은 PVM[7] 혹은 MPI[8]를 기본으로 하는 NOW(Network Of Workstation)의 개념과 유사하나 다음과 같은 두 가지 차이점이 있다. 하나는 전용 클러스터링을 사용하지 않고 인터넷상에서 분산된 자원을 이용한다는 점이고, 다른 하나는 다중 에이전트(multiple agents) 시스템을 기반으로 하고 있는 점이다.

최근에 다양한 분야의 응용 프로그램에서 에이전트의 개념이 이용되고 있으며[9-11], 자바를 기반으로 하는 다수의 이동 에이전트(mobile agent) 시스템들이 이미 개발되어 사용 중이 있다[12-15]. 이동 에이전트 시스템은 병렬 컴퓨팅 분야에도 도입되어 응용되고 있으며, [16, 17]에서는 이러한 시스템을 기반으로 하는 병렬 계산에서 작업 분산을 위한 방법과 성능 모델을 제안하였다. 현재 다중 에이전트 시스템을 기반으로 하는 가상 병렬 컴퓨팅 시스템에서 병렬 계산을 위한 작업 할당 기법으로는 마스터/슬레이브 개념을 응용한 방식이 주로 사용된다.

먼저 마스터 역할을 하는 중앙의 조정자 에이전트(coordinator agent)가 작업자 에이전트(worker agent)를 생성하여 슬레이브에 파견하고, 또한 전체 작업 패키지를 단위 작업 패키지로 분할하여 슬레이브의 작업자 에이전트에 분산한다[16]. 각 작업자 에이전트는 자신에 할당된 작업 패키지를 수행하여 그 결과를 조정자 에이전트에 보내고, 조정자 에이전트는 전체 수행 결과를 통합하게 된다. 이러한 방법은 구조가 단순하고 프로그램 구현이 쉽다는 장점이 있지만, 계산할 작업량이 많고 작업자 에이전트 수가 일정 크기 이상이 되면 조정자 에이전트를 담당하는 마스터의 과도한 부하로 인해 성능이 급속도로 저하되는 문제점을 가지고 있다.

본 논문에서는 이러한 조정자 에이전트의 과부하 문제를 해결하기 위해 조정자 에이전트의 일부 기능을 작업자 에이전트에게 위임하여, 작업자 에이전트와 작업 패키지를 효율적으로 분산하여 문제를 해결하는 새로운 방법을 제안한다. 이러한 방법은 같은 성능을 갖는 컴퓨터로 구성된 가상 병렬 컴퓨팅 시스템과 서로 다른 성능을 갖는 컴퓨터로 구성된 가상 병렬 컴퓨팅 시스템에서 각각 평가된다. 본 논문에서 제안된 방법의 성능 평가를 위해 IBM에서 개발한 Aglets[15]를 기반으로 하는 가상 병렬 컴퓨팅 시스템이 구성되었고, 성능 평가에 대한 벤치마크로 행렬 곱셈 문제가 자바로 구현되었다. IBM Aglets 시스템은 자바로 구현되어 있기 때문에 플랫폼에 독립적이며 사용자가 그 기능을 확장할 수 있는 효율적인 기능을 가지고 있다.

또한 서로 다른 능력을 갖는 컴퓨터로 구성된 가상 병렬 컴퓨팅 환경에서 컴퓨터의 능력에 따른 작업 패키지의 분할 방법이 제시되었고, 그 결과가 실험적으로 평가되었다. 본 논문에서 제안된 방법과 기존의 방법을 다양한 매개변수에 따라 실험적으로 평가한 결과, 제안된 방법들의 성능이 기존의 방법들에 비해서 괄목할만한 수준으로 개선되었다. 따라서 본 논문의 결과는 다중 에이전트를 기반으로 하는 가상 병렬 컴퓨팅 시스템의 작업 할당 기법에 널리 사용될 수 있을 것으로 기대된다. 기존 방법에 비해서 매개변수에 따른 실험 결과에 의한 각각의 개선 비율은 4장에서 상세하게 기술된다.

본 논문의 구성은 다음과 같다. 2장에서는 다중 에이전트를 기반으로 하는 가상 병렬 컴퓨팅 시스템과 기존의 연구들을 개괄적으로 설명하고, 3장과 4장에서는 각각 동일환경과 이질적인 환경에서 분산 기법을 제안한 후에 실험적인 방법을 통하여 기존의 결과와 성능을 비교한다. 마지막으로 5장에서 결론을 맺는다.

## 2. 관련연구

이 장에서는 다중 에이전트를 기반으로 하는 가상 병렬 컴퓨팅 시스템의 개요를 개략적으로 설명하고, 이러한 환경에서 작업 분산에 관한 기존의 연구들을 검토한다.

### 2.1 다중 에이전트 기반의 가상 병렬 컴퓨팅 시스템

현재 다중 에이전트 시스템을 지원하는 소프트웨어로는 독일 Stuttgart 대학에서 개발한 Mole[14], IBM의 Aglets[15], 그리고 FIPA AP(Agent Platform)[18] 등이 존재한다. 본 논문에서는 IBM의 Aglets을 기반으로 하는 컴퓨터들로 구성된 가상 병렬 컴퓨팅 시스템을 고려한다. Aglets는 IBM에서 개발한 자바 기반의 자율적인 이동 에이전트 시스템으로 서로 다른 네트워크상의 컴퓨터들에 설치되어 있는 에이전트 서버 사이를 에이전트가 쉽게 이주할 수 있으며, 자바를 기반으로 하기 때문에 컴퓨터의 플랫폼에 독립적인 장점을 가지고 있다.

다중 에이전트 시스템은 하나의 에이전트로 해결하지 못하는 복잡한 문제를 여러 에이전트간의 협동을 통해 효과적으로 수행하기 위해 제안되었다. 또한 여러 응용 에이전트 외에 조정자 에이전트라는 중재자를 통해 메시지의 전달과 각 에이전트의 제어를 수행하게 된다. 이 경우 모든 응용 에이전트간의 통신 메시지는 조정자 에이전트를 통해 다른 에이전트로 전달된다. 따라서 각 응용 에이전트는 조정자 에이전트와만 연결된다. 각 에이전트가 어떤 처리 능력이 있는지에 대한 정보는 응용 에이전트 생

성 시에 조정자 에이전트에 등록되며, 조정자 에이전트는 이를 바탕으로 통신 메시지를 해당 응용 에이전트에게 보낸다. 응용 에이전트는 다른 에이전트가 어떤 일을 할 수 있는 지에 대하여 알 필요가 없고, 단지 필요시 통신 메시지를 조정자 에이전트에게 보내어 맡기면 된다.

다중 에이전트를 기반으로 하는 가상 병렬 컴퓨팅 시스템은 다음과 같은 장점들을 갖는다. 첫째, 전체적인 통신회수를 줄여 네트워크 트래픽을 감소시킬 수 있다. 분산 시스템은 주어진 작업을 성취하기 위해 여러 번의 상호작용을 하는 통신프로토콜에 의존한다. 이것은 네트워크 트래픽을 많이 발생시키는 원인이 된다. 이동성을 가진 다중 에이전트 시스템의 경우는 데이타와 코드가 목적지로 이동되어서 수행되기 때문에 통신회수를 줄일 수 있게 한다. 둘째, 시스템 구성이 편리하다. 메시지 전달 방식과 공유메모리 방식은 병렬 컴퓨팅에 참여하는 모든 컴퓨터에 관련된 프로그램을 일일이 설치해야하므로 병렬 프로그램 수정이 발생할 때마다 설치비용이 많이 든다. 그러나 다중 에이전트 시스템 기반인 경우는 초기에 모든 컴퓨터에 에이전트 서버만 설치하면 되고 조정자 에이전트가 있는 중앙의 하나의 컴퓨터에만 응용 프로그램을 설치하면 된다. 셋째, 플랫폼 독립적인 병렬 컴퓨팅 환경을 구성할 수 있다. 자바 언어로 구현된 에이전트 프레임워크를 이용하면 서로 다른 플랫폼의 컴퓨터들로 가상 병렬 컴퓨팅 환경을 구성할 수 있다.

그림 1은 다중 에이전트 시스템을 기반으로 하는 가상 병렬 컴퓨팅 시스템 구성의 예를 보여준다. 현재 이동 에이전트를 이용한 다중 에이전트 시스템은 기존의 RPC(Remote Procedure Call)[19]를 사용한 네트워크 프로그래밍 개념의 확장으로, 컴퓨터간의 원격 상호 통신을 줄여 네트워크 부하를 감소시킬 수 있으므로 분산 및 병렬 시스템의 새로운 패러다임으로 제시되고 있다.

### 2.2 에이전트와 작업 패키지 분산을 위한 기존의 방법

에이전트와 작업 패키지 분산을 위한 가장 단순하면서 널리 사용되는 방법은 마스터/슬레이브 방식을 들 수 있다[12]. 이 방법에서는 가상 병렬 컴퓨팅에 참여하는 컴퓨터들 중에서 하나의 컴퓨터가 마스터가 되고 나머지는 슬레이브가 되며, 개략적인 수행 절차는 다음과 같다. 마스터가 가상 병렬 컴퓨팅에 참여하는 각 컴퓨터에 작업자 에이전트를 순서적으로 보내고, 다음에 전체 작업 패키지를 참여하는 슬레이브 수만큼 균등하게 분할하여 각 슬레이브에 순서적으로 분할된 작업 패키지를 보낸다. 각 슬레이브는 자신에 할당된 작업을 작업자 에이전트에 의해 독립적으로 수행한 후에 결과를 마스터에 보내고, 마스터는 전체 결과를 취합하게 된다. 그러나 이 방법은 가상 병렬 컴퓨팅에 참여하는 슬레이브의 수가 많아 질 경우 마스터에 부하가 집중되며, 슬레이브 수가 적고 전체 작업 패키지가 클 경우 마스터가 슬레이브 수만큼의 작업 패키지를 순서적으로 전송하기 때문에 각 슬레이브의 에이전트는 많은 휴식(idle) 시간을 갖게되어 성능이 저하되는 단점을 가지고 있다.

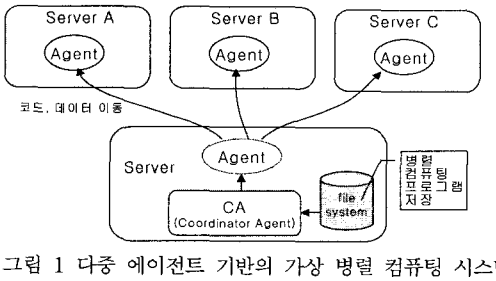


그림 1 다중 에이전트 기반의 가상 병렬 컴퓨팅 시스템의 구성도

마스터/슬레이브 방법에서 발생하는 단점을 일부 보완한 방법이 [16]에서 제안되었다. 여기서는 마스터/슬레이브 방식에서처럼 전체 작업 패키지를 슬레이브 수만큼 균등하게 분할하지 않고 단위 작업 패키지로 분할하여 풀에 유지하고, 이 풀을 이용하여 마스터와 슬레이브 간의 상호 패키지 교환에 의해서 슬레이브가 대기하는 휴식 시간을 줄였다. 이 방법은 다중 에이전트인 Mole 시스템 기반으로 평가되었으며, 개략적인 절차는 다음과 같다. 전체적인 병렬 컴퓨팅 시스템은 어플리케이션, 조정자 에이전트, 작업자 에이전트의 3개 부분으로 구성된다. 어플리케이션은 병렬 컴퓨팅이 필요한 응용 프로그램을 의미하며, 어플리케이션이 수행되면 조정자 에이전트가 자동으로 생성된다. 조정자 에이전트는 어플리케이션의 처리 작업에 따라 적합한 정책을 결정하고 작업자 에이전트를 생성하여 각 컴퓨터에 파견한다. 조정자 에이전트는 작업 패키지를 작은 단위 작업으로 분할하여 작업자 에이전트에게 전송하고, 작업자 에이전트는 전송 받은 단위 패키지를 수행하여 그 결과를 조정자 에이전트에 보낸다. 조정자 에이전트는 결과를 취합하고 단위 작업이 남아 있으면 다시 작업자 에이전트에게 전송한다. 조정자 에이전트는 모든 단위 작업의 계산이 완료될 때까지 작업자 에이전트에게 작업을 보내고 결과를 취합하는 반복작업을 수행한다.

그러나 이 방법은 다음과 같은 문제점을 가진다. 첫째, 작업의 크기에 따라 성능 향상을 위해 작업자 에이전트의 수를 증가시키지만, 그 수가 일정 범위 이상이 되면 더 이상의 성능 향상이 없다. 둘째, 조정자 에이전트의 분산 수

행시간이 각 작업자 에이전트의 평균 수행시간보다 더 길게 되면 조정자 에이전트에서 병목현상이 발생된다. 이것은 단위 작업 패키지로 작업을 분산하여 전송함으로서 조정자 에이전트와 작업자 에이전트간에 잦은 통신 때문에 나타난 결과이다. 그리고 이 방법은 다른 능력을 갖는 컴퓨터로 구성된 가상 병렬 컴퓨팅 시스템에서 작업 패키지 분산을 위한 부하 균형 문제는 고려하지 않았다. 따라서 논문에서는 이러한 문제를 보완할 수 있는 새로운 방법론을 제안하고, IBM Aglets을 기반으로 하는 가상 병렬 컴퓨팅 환경에서 실험적으로 성능을 평가하여 이들 결과와 비교하여 성능이 향상됨을 보인다.

## 3. 동일환경에서 분산기법

이 장에서는 동일 능력을 갖는 컴퓨터로 구성된 가상 병렬 컴퓨팅 환경에서 새로운 분산기법의 개념을 소개하고, 이에 대한 평가 모델과 실험적인 평가를 통하여 기존의 결과와 성능을 비교한다.

### 3.1 개요 및 알고리즘

다음은 본 논문에서 사용될 가상 병렬 컴퓨팅 시스템의 구성요소와 그에 관련된 용어들이다.

- 어플리케이션(Application)

가상 병렬 컴퓨팅 시스템에서 수행할 응용프로그램을 의미한다. 본 논문에서 고려한 응용 프로그램은 기존의 방법에서와 같이 SPMD(Single Program Multiple Data) 형식을 갖는다.

- 조정자 에이전트(Coordinator Agent: CA)

조정자 에이전트는 가상 병렬 컴퓨팅에 참여하는 컴퓨터들의 주소 테이블을 유지하며 어플리케이션 수행을 위한 에이전트의 생성 및 파견, 작업 패키지의 분할 및 전송, 최종 결과의 통합 등 전반적인 흐름을 관장한다.

- 작업자 에이전트(Worker Agent: WA)

작업자 에이전트는 조정자 에이전트가 어플리케이션 수행을 위해 각 컴퓨터에 파견한 에이전트를 의미한다. 효율적인 분산을 위해 조정자 에이전트로부터 일부 권한을 위임받을 수 있다.

- 작업 패키지(Work Package: WP)

작업 패키지는 응용프로그램이 처리하게 될 전체 데이타의 집합을 의미한다. 작업 패키지는 조정자 에이전트 혹은 작업자 에이전트에 의해서 분할되어 가상 병렬 컴퓨팅에 참여하는 컴퓨터들에 전달된다.

- IP 테이블

IP 테이블은 가상 병렬 컴퓨팅에 참여하는 컴퓨터들의 고유 주소 값(인터넷 주소)을 유지하는 테이블을 의미한다. 이 테이블에는 각 컴퓨터의 주소 값과 더불어 그 컴퓨터의 성능지수가 동시에 유지된다. 성능지수는 동적으로 수집되며 후에 이질 환경에서 작업 패키지를 분할하는데 사용된다. 성능지수는 4장에서 설명된다.

본 논문에서 제안하는 기본적인 분산 기법은 다음과 같다. 먼저 어플리케이션이 주어지면 조정자 에이전트가 수행되어 문제에 적합한 작업자 에이전트를 생성하고, IP 테이블을 이용하여 가상 병렬 컴퓨팅에 참여하는 컴퓨터들을 이진트리와 유사한 형식으로 구성한다. 다음에 조정자 에이전트가 작업자 에이전트에게 자신의 일부 권한을 위임하여 하위 컴퓨터들에게 작업자 에이전트를 파견한다. 작업 패키지의 분산도 이와 유사한 절차를 사용하여 자신의 부모 컴퓨터로부터 전송 받은 작업 패키지를 재분할하여 하위 컴퓨터로 전송한다. 이러한 방법은 가장 하위 컴퓨터에 도달할 때까지 반복된다.

이제 위에서 기술한 개념적인 절차를 좀더 구체적으로 설명한다. 먼저 가상 병렬 컴퓨팅에 참여하는 컴퓨터의 수가 $m$이라 가정하고, 병렬 계산을 위한 응용 프로그램의 전체 작업 패키지(WP)의 크기가 $n$이라 가정한다. 응용 프로그램이 주어진 컴퓨터가 조정자 에이전트(CA)와 컴퓨팅에 참여하는 컴퓨터들에서 수행하게 될 작업자 에이전트(WA)를 생성한다. 이때 조정자 에이전트는 자신의 일부 권한을 작업자 에이전트에게 위임하여, 작업자 에이전트가 자율적으로 작업자 에이전트 및 작업 패키지를 분산하는 데에 참여할 수 있도록 한다. 작업자 에이전트의 파견 절차는 그림 2와 같이 IP 테이블을 이용하여 매 단계마다 작업자 에이전트를 갖는 모든 컴퓨터는 자신의 하위 컴퓨터들에 작업자 에이전트를 파견하는데 참여하게 된다.

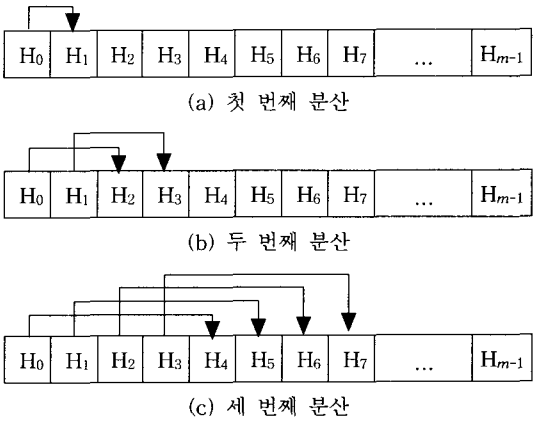


그림 2 IP 테이블을 이용한 작업자 에이전트 분산과정

그림 2에서 첫 번째의 경우는 IP 테이블의 $H_0$의 주소 값을 갖는 컴퓨터가 $H_1$의 주소 값을 갖는 컴퓨터에 작업자 에이전트를 파견하고, $H_0$의 작업자 에이전트가 $H_1$의 작업자 에이전트에게 작업 패키지를 $n/2$으로 분할하여 전송한다. 같은 방법으로 두 번째 단계에서는 $H_0$와 $H_1$이 분산에 참여하여 $H_2$와 $H_3$에 작업자 에이전트와 자신이 가지고 있는 작업 패키지의 반인 $n/4$의 작업 패키지를 전송한다. 단계 3에서는 $H_0$, $H_1$, $H_2$, $H_3$가 분산에 참여하여 $H_4$, $H_5$, $H_6$, $H_7$에 작업자 에이전트와 $n/8$의 작업 패키지를 전송한다. 이러한 방법을 이용하면 전체적으로 $\lceil \log_2 m \rceil - 1$ 단계 후에 작업자 에이전트와 작업 패키지 분산이 완료된다. 그림 2의 분산 과정을 이진 트리 형식으로 표현한 예가 그림 3에 보여졌다.

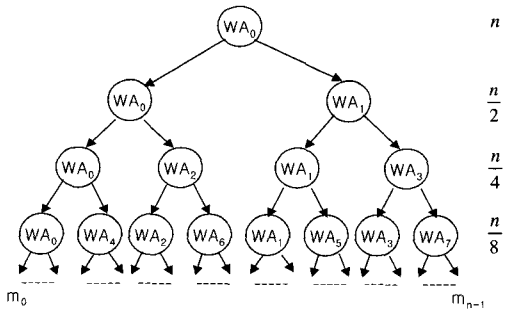


그림 3 이진트리 형식으로 표현된 분산과정

가상 병렬 컴퓨팅 시스템을 구성하는 각 컴퓨터는 작업자 에이전트와 작업 패키지를 전송 받으면 즉시 자신에 할당된 작업을 수행한다. 전체적인 결과의 통합은 위에서 설명한 분산과 반대의 절차를 사용한다. 즉 각 컴퓨터는 자신의 결과를 자신의 부모에 해당하는 컴퓨터에 전송하고, 부모 컴퓨터는 이 결과를 취합하게 된다. 이러한 과정은 조정자 에이전트를 갖는 컴퓨터에 도달할 때까지 반복된다.

다음은 이러한 개념을 수행하기 위한 구체적인 알고리즘들을 개략적으로 기술한다. 알고리즘에 사용된 변수 중에서 $m$은 컴퓨터의 수, $n$은 전체 작업 패키지의 크기, ip_table은 병렬 컴퓨터에 참여하는 컴퓨터들의 IP를 보관하는 테이블, 그리고 worker_agent는 최초에 조정자 에이전트로부터 생성된 작업자 에이전트를 의미한다. 또한 self_result는 현재의 작업자 에이전트에서 계산된 결과의 값, child_result는 자식 컴퓨터로부터 전송받은 결과 값을 의미한다. ip_table에서 가장 위쪽으로부터 아래쪽까지 병렬 컴퓨팅에 참여하는 컴퓨터들의 주소 필드를 $P_0, P_1, \ldots, P_{m-1}$로 정의한다.

알고리즘 1은 분산을 수행하기 위한 작업자 에이전트의 메인 프로시저로 최초에 조정자 에이전트에서 수행된다. 또한 알고리즘 1은 알고리즘 2를 호출하여 작업자 에이전트와 작업 패키지를 모든 컴퓨터에 전송하고, 알고리즘 4를 호출하여 각 컴퓨터에서 수행된 결과들을 취합한다.

[알고리즘 1] 메인 프로시저

```
procedure PARALLEL( )
    Agent_Distribution(ip_table, worker_agent)
    Result_Integration(self_result, child_result)
end procedure
```

[알고리즘 2] 작업자 에이전트 분산

```
procedure Agent_Distribution(ip_table, worker_agent)
  for i=0 to ⌈ log2m ⌉ − 1
    for j=0 to 2^i-1 in parallel
      1) worker_agent 생성한다.
      2) 컴퓨터 Pj가 worker_agent를 컴퓨터 P2^i+j에
         파견한다.
      3) Package_Distribution(n, i, j)
    end for
  end for
end procedure
```

[알고리즘 3] 작업 패키지 분산

```
procedure Package_Distribution(n, i, j)
  1) work_package = n/2^i
  2) 컴퓨터 Pj가 work_package를 컴퓨터 P2^i+j에
     전송한다.
end procedure
```

[알고리즘 4] 결과통합

```
procedure Result_Integration(self_result, child_result)
  for i = ⌈ log2m ⌉ − 1  to 0 step -1
    for j=0 to 2^i-1 in parallel
      컴퓨터 Pj가 자신의 계산결과 (self_result)와 컴퓨
      터 P2^i+j의 계산결과(child_result)를 통합한다.
    end for
  end for
end procedure
```

알고리즘 2는 매 단계마다 작업자 에이전트가 새로운 작업자 에이전트를 생성하여, IP 테이블을 이용하여 하위 컴퓨터에 파견하는 과정을 보여준다. 여기서 첫 번째 for 루프는 그림 2에서 설명한 절차에 따라 $m$개의 컴퓨터에 분산하기 위한 단계를 의미하고, 두 번째 for 루프는 매 단계마다 병렬로 분산에 참여하는 과정을 보여준다. 알고리즘 3은 알고리즘 2에서 호출된 서브 프로시저로서 매 단계마다 자신이 가지고 있는 작업 패키지를

반으로 분할하여 자신의 하위 컴퓨터에 전송하는 과정을 보여준다. 알고리즘 4는 각 컴퓨터의 작업 에이전트에 의해서 수행된 결과를 통합하는 프로시저이다. 결과의 통합은 작업자 에이전트의 분산과정과 반대로 수행되며, 하위 컴퓨터의 결과를 통합하여 그 결과를 자신의 상위 컴퓨터에 전송하게 된다.

### 3.2 성능모델 및 평가

다음은 앞 절에서 설명한 방법에 대한 성능모델을 분석하여 평가 식을 유도하고 기존의 방법과 성능비교를 한다. 성능평가를 위한 기본적인 환경은 [16]과 동일한 가정을 사용한다. 먼저 네트워크는 완전 연결 상태로 가정하여 성능평가 모델에서는 네트워크 부하는 무시된다. 그러나 후에 3.3절에서 실제 실험적 평가 시에는 이러한 네트워크 부하가 고려된다. 모든 작업자 에이전트들은 같은 속도로 수행되며, 작업자 에이전트의 시작 및 이동시간, 단위 작업 패키지가 작업자 에이전트에 전송되는 시간, 수행된 결과를 받는 시간 및 통합시간, 그리고 단위 작업 패키지를 수행하는 시간 등이 이미 알려져 있다고 가정한다. 표 1에서 이러한 알려진 값들에 대한 기호들의 정의되었다.

표 1 성능모델에 사용된 기호 목록

| 기 호 | 의 미 |
|---|---|
| $t_{start}$ | 작업자 에이전트의 시작시간 |
| $t_{mig}$ | 작업자 에이전트의 이동시간 |
| $t_{send}$ | 단위 작업 패키지가 작업자 에이전트에 전송되는 시간 |
| $t_{rec}$ | 결과를 받는 시간 |
| $t_{int}$ | 결과를 통합하는 시간 |
| $t_{exe}$ | 단위 작업 패키지를 수행하는 시간 |

3.1절에서 기술한 방법에 대한 성능모델은 다음과 같이 표현될 수 있다. 아래의 식에서 $n$은 작업 패키지의 크기이고 $m$은 작업자 에이전트의 수이다.

▶ 각 컴퓨터에 작업자 에이전트를 배치하는 시간

$$t_{ini} = (t_{start} + t_{mig}) \times \log_2^m \quad \text{(식 1)}$$

▶ 모든 작업 패키지가 작업자 에이전트에 전송되는 시간

$$t_{pack} = n(1/2 + 1/4 + 1/8 + \cdots + 1/(2^{\log_2^m})) \times t_{send}$$
$$= n(1 - (1/2)^{\log_2^m}) \times t_{send} \quad \text{(식 2)}$$

▶ 전체 작업 패키지를 수행하는 시간

$$t_{work} = n/m \times t_{exe} \quad \text{(식 3)}$$

▶ 계산결과를 받고 통합하는 시간

$$t_{fin} = (t_{rec} + t_{int}) \times \log_2 m \quad \text{(식 4)}$$

▶ 전체 소요 시간

$$t_{total} = t_{ini} + t_{pack} + t_{work} + t_{fin} \quad \text{(식 5)}$$

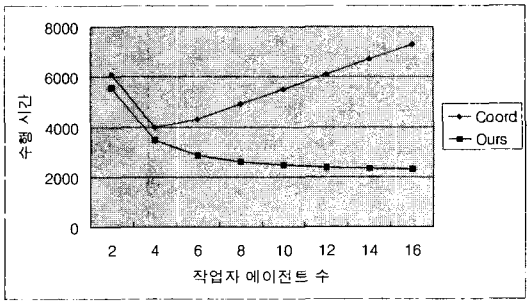


그림 4 작업자 에이전트의 수에 따른 성능 비교

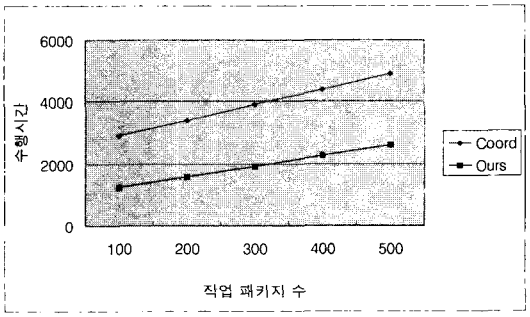


그림 5 작업 패키지 수에 따른 성능 비교

위의 식에 대한 성능 평가를 위해 작업자 에이전트의 수를 2에서 16까지를 고려하며, 작업 패키지의 크기는 500으로 가정한다. 그림 4는 $t_{exe}$=20, $t_{start}$=200, $t_{mig}$=100, $t_{send}$= $t_{rec}$=1, $t_{int}$=3으로 가정했을 때, [16]의 방법과 본 논문에서 제안한 방법의 성능을 비교한 결과를 보여준다. 여기서 시간은 ms 단위이며, 각 상수의 값은 [16]에서와 같은 값을 사용한다. 그래프에서 Coord로 표기된 부분은 [16]에서 제안한 방법의 결과를 나타내고, Ours로 표기된 부분은 본 논문에서 제안한 방법의 결과를 나타낸다. 그림에서 보듯이 기존 방법은 작업자 에이전트의 수가 4개 이상이 되면 더 이상의 성능 향상이 없다.

이것은 작업자 에이전트의 수행시간 보다 조정자 에이전트의 수행시간이 더 많이 걸리게 되어 조정자 에이전트에 부하가 집중되기 때문이다. 반면, 개선된 방법은 작업 패키지의 수에 따라 작업자 에이전트의 수를 증가

하면 더 좋은 성능을 낸다는 것을 알 수 있다. 이것은 조정자 에이전트의 부하 문제를 해결하기 위해 각 작업자 에이전트에게 조정자 에이전트의 기능을 위임하여 부하를 분산함으로써 얻어진 결과이다.

그림 5는 작업자 에이전트의 수를 8, 단위 작업 패키지의 수행 시간을 20으로 가정하고 작업 패키지의 수를 100에서 500까지 100씩 증가시켰을 때 결과를 비교 한 그래프이다. 그림에서 보는 바와 같이 작업 패키지 수의 증가에 따른 성능 또한 본 논문에서 제안한 방법이 더 좋은 결과를 나타내었다. [16]의 방법을 기준으로 했을 때 본 논문에서 제안한 방법이 평균적으로 51% 정도의 성능향상을 보였다.

본 논문에서 제안한 방법은 가상 병렬 컴퓨팅에 참여하는 모든 컴퓨터가 작업 패키지를 병렬로 전송하기 때문에, 초기 단계에서는 같은 작업 패키지가 반복 전송되는 단점을 갖는다. 그러나 제안된 방법은 병렬 컴퓨팅에 참여하는 모든 컴퓨터에게 작업 패키지 전송을 위임하는 정책을 사용하여, 기존의 마스터/슬레이브 구성에서 마스터에 집중되는 부하 때문에 발생되는 성능저하 문제를 개선하였다.

### 3.3 네트워크 트래픽을 고려한 실험적 성능평가

3.2에서 보여진 결과는 성능모델에 의한 두 방법의 평가이고 실제 환경에서 발생할 수 있는 네트워크 부하는 고려하지 않았다. 다음에서 네트워크 트래픽을 고려한 실제 환경에서 실험적인 성능평가 결과가 설명된다. 실험적 평가를 위해 Solaris 2.5.1 운영체제가 설치된 Sun Sparcstation (110㎒, 32MB) 워크스테이션을 사용하였으며, 각 워크스테이션에는 IBM Aglets 1.1.b3을 설치하였다. 각 워크스테이션은 이더넷 10Mbps LAN으로 연결되어 있으며, 실험에서는 1에서 16대까지의 워크스테이션을 사용하였다. 성능평가를 위한 벤치마크로는 행렬 곱셈 문제를 사용하였고, 이 프로그램은 Aglets 상에서 이동 에이전트 언어로 사용되는 자바로 구현하였다.

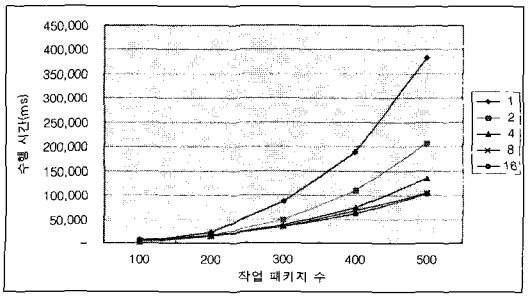


그림 6 작업 패키지와 작업자 에이전트 수에 따른 성능

그림 6은 1, 2, 4, 8, 16개의 워크스테이션에 대해서 각각 작업 패키지를 100에서 500까지 100씩 증가하면서 실험한 결과를 보여준다. 그림에서 작업 패키지의 크기가 적을 경우는 에이전트 수가 전체 성능에 크게 영향을 미치지 않는 것을 확인할 수 있다. 하지만 작업 패키지의 크기가 클 경우는 작업자 에이전트 수가 많을 수록 더 좋은 성능을 얻을 수 있음을 알 수 있다. 그림에서 작업 패키지의 크기가 500이고 작업자 에이전트 수가 16인 경우는 작업자 에이전트 수 1에 비해 대략 4배 정도의 성능향상이 있음을 확인할 수 있다.

표 2 동일 성능 환경의 실험 결과(시간 : millisecond)

| 대수 | WP크기 / 방법 | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|---|
| 2 | Ours | 4,122 | 21,439 | 59,061 | 124,194 | 232,571 |
| | Coord | 4,864 | 25,298 | 69,692 | 146,549 | 274,434 |
| | MS | 4,477 | 19,966 | 56,397 | 133,581 | 229,755 |
| 4 | Ours | 7,138 | 20,811 | 45,940 | 88,620 | 162,277 |
| | Coord | 9,279 | 27,054 | 59,722 | 115,206 | 210,960 |
| | MS | 5,496 | 20,464 | 41,121 | 82,521 | 144,191 |
| 8 | Ours | 8,333 | 21,209 | 44,219 | 82,047 | 120,897 |
| | Coord | 14,999 | 38,176 | 79,594 | 147,685 | 217,615 |
| | MS | 10,215 | 23,356 | 42,310 | 84,269 | 127,975 |
| 16 | Ours | 10,098 | 22,287 | 43,625 | 75,906 | 125,070 |
| | Coord | 22,216 | 49,031 | 95,975 | 166,993 | 275,154 |
| | MS | 17,039 | 35,654 | 70,395 | 112,417 | 165,901 |

표 2는 동일 환경에서 작업자 에이전트 수와 작업 패키지 수에 따른 성능관계를 알아보기 위해 본 논문의 결과와 기존방법의 결과를 구현하여 실험한 결과자료를 보여준다. 표에서 실험결과의 수행시간은 작업자 에이전트 파견에서부터 최종결과를 통합하는 시간까지를 측정한 자료이다. 표에서 Coord는 [16]의 방법, MS(마스터/슬레이브)는 [12]의 방법, 그리고 Ours는 본 논문의 방법을 표시한다.

그림 7과 그림 8은 표 2의 결과에서 작업자 에이전트의 수가 각각 4와 16인 경우를 그래프로 도식한 예를 보여준다. 그림에서 보는 바와 같이 작업자 에이전트와 작업 패키지 수가 적을 때는 성능차이가 크게 없지만, 작업자 에이전트와 작업 패키지 수가 많아지면 본 논문에서 제안한 방법이 훨씬 더 좋은 성능을 낸다는 것을 알 수 있다. 기존 방법에서는 에이전트 파견과 작업 패키지 전송을 조정자 에이전트나 마스터가 전담한다. 하지만, 제안한 방법에서는 작업자 에이전트가 이동하면서 자신과 동일 역할을 하는 작업자 에이전트를 파견하고

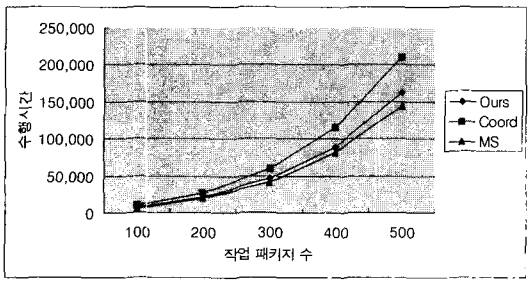


그림 7 작업자 에이전트가 4개일 때 결과

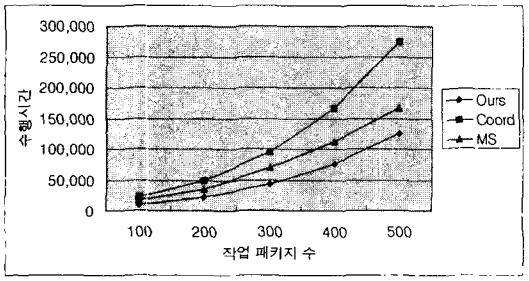


그림 8 작업자 에이전트가 16개일 때 결과

작업 패키지를 전송함으로서 조정자 에이전트나 마스터에 집중된 부하를 분산하기 때문에 이러한 결과를 얻을 수 있다.

그림 8의 결과를 참조하면 작업자 에이전트의 수가 16이고 작업 패키지의 크기가 500일 경우에 본 논문의 방법이 Coord[16]에 비해서 2.2배, MS[12]에 비해서 1.3배의 성능 향상이 있음을 확인할 수 있다. 표 2에서 작업 패키지의 크기가 500일 경우 본 논문에서 제안된 결과를 관찰하면, 가상 병렬 컴퓨팅에 참여하는 컴퓨터가 8일 때의 수행시간이 16일 때의 수행시간에 비해서 근소한 비율로 적음을 확인할 수 있다. 이러한 결과는 작업 패키지의 크기가 클 경우 작업 패키지를 병렬로 전송하는 과정에서 발생되는 네트워크 오버헤드로 인하여, 컴퓨터의 대수를 늘리더라도 오히려 성능이 저하되는 경우를 보여준다.

## 4. 이질환경에서 분산기법

이 장에서는 가상 병렬 컴퓨팅 환경에 참여하는 컴퓨터들의 능력에 따라 작업 패키지의 크기를 달리하여 작업 패키지를 분산하는 방법을 제안하고, 실험을 통하여 제안한 결과에 대한 성능을 분석하고 평가한다.

### 4.1 개요 및 알고리즘

병렬 컴퓨팅에 참여하는 컴퓨터가 이질적인 계산 능력을 가질 경우 모든 컴퓨터에 동일한 작업 크기를 할당하면 자원활용 면에서 효율성이 떨어지게 된다. 작업 패키지의 크기를 균등하게 분할하여 분산할 경우 성능이 좋은 컴퓨터에서 수행하는 작업자 에이전트는 결과를 빨리 응답하고, 상대적으로 성능이 좋지 않은 컴퓨터에서 수행하는 작업자 에이전트는 결과를 늦게 응답하게 된다. 따라서 각 컴퓨터들의 성능을 고려하여 적절한 작업 크기를 할당하지 않는다면, 성능이 좋지 않은 컴퓨터의 느린 결과 응답으로 인하여 전체적인 수행시간이 길어지게 된다. 이러한 문제를 해결하기 위한 한 방법으로 가상 병렬 컴퓨팅에 참여하는 각 컴퓨터의 성능을 동적으로 판단하여, 성능에 기반 하는 작업 패키지 할당 정책을 사용할 수 있다.

제안된 방법의 전체적인 개념은 3장에서 설명한 동일 환경과 같고 여기서는 각 컴퓨터의 성능에 기반 하여 작업 패키지를 할당하는 방법을 추가한다. 각 작업자 에이전트는 자신이 수행되는 컴퓨터의 성능에 따라 적절한 비율로 분할된 독립적인 작업을 수행하며, 자신에 할당된 작업 패키지의 계산이 완료될 때까지 작업자 에이전트의 수와 분할된 비율은 변경되지 않는다. 작업자 에이전트를 각 컴퓨터에 파견하기 전에 조정자 에이전트는 먼저 각 컴퓨터의 성능측정을 위해 에이전트를 각 컴퓨터로 파견하고 그 결과를 수집한다. 그런 다음 각 컴퓨터의 성능 값을 전체 합으로 나누어 각 컴퓨터의 성능 비를 계산한다. 이러한 성능 비를 본 논문에서는 성능지수라고 명하며 성능지수의 전체 합은 1이 된다. 성능지수는 각 컴퓨터에서 수행되는 작업자 에이전트에게 할당할 작업크기를 결정하는데 이용된다. 각 컴퓨터의 성능을 평가하는 방법은 여러 가지 방법이 있으나, 본 논문에서는 수행시간이 짧고 간단한 방법으로 성능을 측정하는 Linpack 벤치마킹[21]을 이용한다.

이 벤치마킹 방법은 행렬을 이용한 선형방정식(Ax=b)의 해를 구하는 알고리즘을 적용하여 컴퓨터의 부동소수점 연산능력을 평가한다. 방정식의 해는 부분 피벗팅을 이용한 가우스 소거법을 이용하며, 단위 시간당 부동 소수점 덧셈과 곱셈 연산의 수행 횟수를 검사하여 초당 몇 백만 번의 부동소수점 연산을 수행하는지에 대한 MFLOPS 단위의 값을 측정한다. 조정자 에이전트를 수행하는 컴퓨터는 각 컴퓨터의 성능 결과를 받아서 성능이 좋은 순서로 IP 테이블을 정렬하고, 전체 성능에 대한 각 컴퓨터의 성능지수를 계산해서 IP 테이블에 저장한다. 후에 이러한 IP 테이블은 각 컴퓨터에 분산되어 자신에 할당된 작업 패키지의 양과 자신의 하위 컴퓨터에 할당할 작업 패키지의 양을 계산하는데 이용된다. 표

3은 성능지수에 관련된 내용을 구체적으로 설명하기 위해 사용될 기호들을 보여준다.

표 3 작업 패키지 할당 정책에 사용될 기호

| 기 호 | 의 미 |
|---|---|
| $w_i$ | $i$번째 작업자 에이전트 |
| $m$ | 가상 병렬 컴퓨팅에 참여하는 컴퓨터 수 |
| $n_i$ | $w_i$ 에 할당되는 작업 패키지 크기 |
| $N$ | 전체 작업 패키지 크기 |
| $c_i$ | 각 컴퓨터의 성능(MFLOPS) |
| $c_{total}$ | 전체 컴퓨터들의 성능 합계 |
| $r_i$ | $i$번째 컴퓨터의 성능지수 |

$c_{total}$은 병렬 컴퓨팅에 참여하는 전체 컴퓨터들의 성능 합을 의미하며 $c_{total} = \sum_{i=0}^{m-1} c_i$로 표현된다. $r_i$는 $i$번째 컴퓨터에 대한 성능지수를 의미하며 $r_i = c_i / c_{total}$로 계산되고, $n_i$는 $i$번째 컴퓨터에 할당될 작업 패키지의 크기를 의미하며 $n_i = N \times r_i$로 계산된다. 다음은 3장에서 설명한 알고리즘들을 기본으로 하여 각 컴퓨터의 성능에 따라 작업 패키지를 분산하도록 재구성한 프로시저들을 보여준다. 알고리즘에 사용되는 변수들 중에서 $m$은 병렬 컴퓨팅에 참여하는 컴퓨터의 수이고 package는 작업 패키지를 나타낸다. 나머지 변수들은 3장에서 사용된 것들과 동일하게 정의된다. ip_table은 $m$개의 컴퓨터에 대한 주소 값과 성능지수를 보관하는 필드를 가지고 있으며, 테이블의 가장 위쪽으로부터 아래쪽까지 주소 필드는 $P_0, P_1, \ldots, P_{m-1}$로 정의하고, 성능지수 필드는 $r_0, r_1, \ldots, r_{m-1}$로 정의한다.

[알고리즘 5] 메인 프로시저

```
procedure PARALLEL( )
    Host_Performance(ip_table)
    Distribution(ip_table, worker_agent, package)
    Result_Integration(self_result, child_result)
end procedure
```

[알고리즘 6] 성능측정

```
procedure Host_Performance(ip_table)
  for i=0 to m-1
     컴퓨터 Pi에 성능측정 에이전트를 파견한다.
  end for
  for i=0 to m-1
     컴퓨터 Pi에 대한 에이전트로부터 벤치마크 결과
     (MFLOPS)를 전송 받는다.
  end for
  병렬 컴퓨팅에 참여하는 컴퓨터들에 대한 성능지수를
  계산한다.
  성능지수 값에 따라 내림차순으로 ip_table을 정렬한다.
end procedure
```

[알고리즘 7] 작업자 에이전트와 작업 패키지 분산

```
procedure Distribution(ip_table, worker_agent, package)
  for i=0 to ⌈ log2 m ⌉ - 1
    for j=0 to 2^i-1 in parallel
      1) worker_agent를 생성한다.
      2) 컴퓨터 Pj가 컴퓨터 P2^i+j에 worker_agent를 파견한다.
      3) Task_Division(ip_table, package, j)
      4) 컴퓨터 Pj가 컴퓨터 P2^i+j에 성능지수에 따라 분할된
         package를 전송한다.
    end for
  end for
end procedure
```

[알고리즘 8] 작업 패키지 분할

```
procedure Task_Division(ip_table, package, j)
  1) 컴퓨터 Pj의 작업자 에이전트가 자신이 가진 작업
     package를 작업할당 정책(위에서 기술)에 따라
     package를 분할한다.
  2) 분할된 package를 새로운 package로 설정한다.
end procedure
```

알고리즘 5는 메인 프로시저로서 각 컴퓨터의 성능을 측정하는 프로시저 호출, 작업자 에이전트와 작업 패키지를 분산하는 프로시저의 호출, 그리고 결과를 통합하는 프로시저 호출로 구성된다. 여기서 결과를 통합하는 프로시저 Result_Integration은 3장의 알고리즘 4와 동일하므로 구체적인 기술은 생략한다. 알고리즘 6은 성능측정 프로시저로 조정자 에이전트는 각 컴퓨터에 성능측정 에이전트를 파견하고, 각 에이전트로부터 MFLOPS 성능 정보를 받아서 성능지수를 계산한 후에 IP 테이블을 성능지수 값의 내림차순으로 재 정렬한다.

알고리즘 7은 IP 테이블에 저장된 정보를 이용하여 병렬 컴퓨팅에 참여하는 각 컴퓨터에 작업자 에이전트와 작업 패키지를 분산하는 프로시저이다. 두 개의 for 루프에서 첫 번째 for 루프는 이진분산 형태를 이용한 단계별 증가를 의미하고, 두 번째 for 루프는 병렬로 수행되며 매 단계마다 작업자 에이전트와 작업 패키지 분산에 참여하는 컴퓨터들의 수행과정을 의미한다. 알고리즘 7은 각 컴퓨터의 성능에 따른 작업 패키지 분할을 위해 알고리즘 8의 Task_Division 프로시저를 호출하고, 알고리즘 8의 프로시저는 위에서 기술한 것과 같이 ip_table에 저장된 각 컴퓨터 성능지수를 이용하여 작업 패키지를 분할한다. 즉, 컴퓨터 $P_i$에는 $n_i = N \times r_i$ 만큼의 작업 패키지가 할당된다.

### 4.2 구현 및 성능 평가

성능 평가를 위한 실험 환경은 다음과 같다. 이더넷 10Mbps LAN 환경에 연결된 컴퓨터를 사용하였고, 각 컴퓨터에는 다중 에이전트를 지원하는 IBM의 Aglets

1.1.b3가 설치되었다. 실험에 참여한 컴퓨터는 4대로 각각의 사양은 다음과 같다. Solaris 2.5.1 운영체제의 Sun Sparcstation(110Mhz, 32M) 2대, 윈도우 98 운영체제의 펜티엄 II(400Mhz, 128M) PC 1대 및 펜티엄 II(300Mhz, 64M) 1대가 사용되었다. 이러한 환경에서 성능 평가를 위해 행렬 곱셈 문제를 벤치마크로 사용하였으며, 프로그램은 자바 및 Aglets에서 제공하는 라이브러리를 사용하여 구현하였다.

표 4 이질 환경에서 실행 결과(시간 : millisecond)

| 작업크기 / 방법 | 300 | 400 | 500 | 600 |
|---|---|---|---|---|
| O-LB | 11,980 | 18,070 | 26,970 | 35,270 |
| O-NLB | 25,210 | 58,170 | 101,400 | 173,730 |
| M-LB | 14,680 | 23,620 | 30,490 | 41,970 |
| M-NLB | 27,020 | 59,490 | 111,610 | 186,250 |

표 4는 이러한 가상 병렬 컴퓨팅 환경에서 각 컴퓨터의 능력을 고려한 작업 패키지의 할당 방법이 그렇지 않은 방법에 비해서 어느 정도의 성능 개선이 있는지를 평가한 결과를 보여준다. 성능 평가를 위해 4가지 경우를 고려하였다. 표에서 O-LB(Ours Load Balancing)는 본 논문의 방법을 의미하고, O-NLB(Ours None Load Balancing)는 본 논문의 3장에서 제안한 방법으로 작업 패키지를 균등하게 고려한 경우를 의미한다. M-LB (Mater-salve Load Balancing)는 마스터/슬레이브 방식에서 컴퓨터의 능력에 따른 작업 패키지 할당을 고려한 경우이고, M-NLB(Master-slave None Load Balancing)는 마스터/슬레이브 방식에서 균등하게 작업 패키지를 할당한 경우이다. 실험 결과에서 보여진 수행시간은 각 컴퓨터의 성능 측정 시간, IP 테이블의 재구성 시간, 그리고 작업자 에이전트의 파견에서부터 최종 결과를 통합하는 데 걸리는 시간들의 합을 표시한다. 또한 작업 패키지의 크기는 300에서 600까지 100씩 증가하여 수행시간을 측정하였다.

그림 9는 본 논문에서 제안한 방법 중에서 컴퓨터의 성능을 고려한 작업 패키지 분산 방법(O-LB)이 그렇지 않은 작업 패키지 분산(O-NLB)에 비해서 어느 정도가 개선되었는가를 보여준다. 그림에 보여진 것과 같이 이질 환경에서 각 컴퓨터의 성능을 고려하여 작업 패키지를 분산하면 그렇지 않은 경우에 비해서 작업 패키지가 클수록 훨씬 더 좋은 결과를 얻을 수 있음을 확인할 수 있다. 이 실험 결과에서 작업 패키지의 크기가 500과 600인 경우를 참조하면, 성능을 고려한 작업 패키지 할당이 그렇지 않은 경우에 비해서 각각 3.8배와 4.9배의 성능향상을 보였다

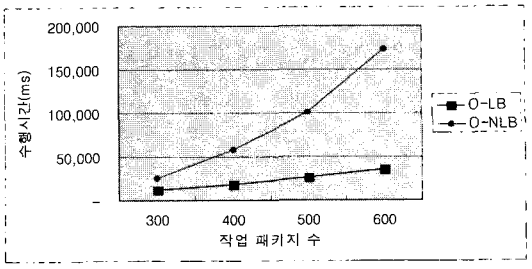


그림 9 작업 패키지 수에 따른 O-LB와 O-NLB의 성능

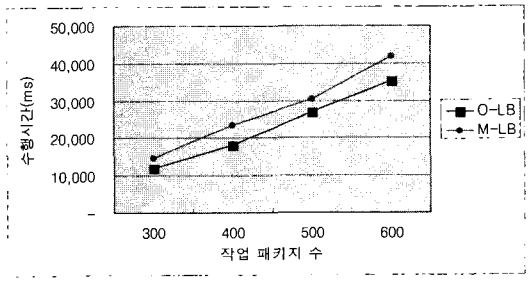


그림 10 작업 패키지 수에 따른 O-LB와 M-LB의 성능

그림 10은 본 논문에서 제안한 이질 환경의 성능 기반 작업할당(O-LB) 개념을 마스터/슬레이브 방법에 적용(M-LB)하여, 두 방법의 성능을 평가한 결과이다. 그림의 결과에서 확인할 수 있듯이 본 논문에서 제안한 성능 기반 작업 패키지 할당 방법(O-LB)이 마스터/슬레이브 방식(M-LB)에 비해서 평균적으로 17%정도의 성능 향상이 있었다. 이 결과에 의하면 마스터/슬레이브 방식에서 성능에 기반 하는 작업분산을 하더라도, 본 논문의 방법에 비해서 마스터의 부하 집중 현상으로 인하여 결국은 성능이 저하된다는 점을 확인할 수 있다.

## 5. 결 론

본 논문에서는 다중 에이전트를 지원하는 IBM Aglets를 기반으로 하는 가상 병렬 컴퓨팅 환경에서 작업자 에이전트와 작업 패키지를 효율적으로 분산하는 방법과 이에 대한 성능 평가를 하였다. 다중 에이전트 기반의 병렬 컴퓨팅 시스템에서 조정자 에이전트에 집중되는 부하를 분산하기 위해, 작업자 에이전트에게 조정자 에이전트의 권한을 위임하여 작업자 에이전트가 에이전트와 작업 패키지의 분산에 참여하는 새로운 방법을 제안하였다. 그리고 동일 성능을 가지는 컴퓨터들과 이질 성능을 가지는 컴퓨터들로 구성된 가상 병렬

컴퓨팅 시스템을 구성하여 실험적인 방법을 통하여 기존의 방법과 성능을 평가하였다.

성능 평가 결과에 의하면 동일 환경의 경우는 제안한 방법이 기존의 방법에 비해서 작업 크기가 작고 에이전트 수가 적을 경우는 성능 차이가 없지만, 작업 크기가 커지고 에이전트 수가 증가함에 따라 훨씬 더 좋은 성능을 보였다. 또한 이질 환경에서는 각 컴퓨터의 성능을 고려하여 작업을 분산한 경우가 그렇지 않은 경우에 비해서 훨씬 더 좋은 성능을 보임을 실험을 통하여 입증하였다. 최근에 다중 에이전트를 기반으로 하는 가상 병렬 컴퓨팅 시스템이 고비용 수퍼 컴퓨터의 사용을 대신하여 활용되고 있는 추세이다. 본 논문의 결과는 IBM Aglets을 기반으로 하는 병렬 컴퓨팅 분야에서 효율적인 작업 할당에 많은 도움을 줄 수 있을 것으로 기대된다.

## 참 고 문 헌


[1] 김경하, 김영균, 김영학, 오길호, "이동 에이전트 시스템 기반의 병렬 계산을 위한 효율적인 분산방법", *한국정보과학회 2000 봄학술논문집(A)*, pp. 615-617, 2000.

[2] 김경하, 김영학, 오길호, "다중 에이전트 시스템 기반의 병렬 계산을 위한 작업할당 기법과 성능비교", *한국정보과학회 2000 가을 학술논문집(III)*, pp. 502-504, 2000.

[3] L. F. G. Sarmenta, "Bayanihan: Web-Based Volunteer Computing Using Java," *Lecture Notes in Computer Science 1368*, Springer-Verlag, pp. 444-461, 1998.

[4] L. F. G. Sarmenta and S. Hirano, "Bayanihan: Building and Studying Web-Based Volunteer Computing Systems Using Java," *Future Generation Computer Systems*, Vol. 15, pp. 675-686, 1999.

[5] L. F. G. Sarmenta. "An Adaptive, Fault-tolerant Implementation of BSP for Java-based Volunteer Computing Systems," *Lecture Notes in Computer Science 1586*, Springer-Verlag, pp. 763-780, 1999.

[6] B. O. Christiansen, P. Cappello, B. O. Christiansen, M. F. Ionescu, M. O. Neary, and K. E. Schauser, "Javelin: Internet-Based Parallel Computing Using Java," *1997 ACM Workshop on Java for Science and Engineering Computation*, June 20, 1997.

[7] V. Sunderam, J. Dongarra, A. Geist, and R. Manchek, "The PVM Concurrent Computing System: Evolution, Experiences, and Trends," *Parallel Computing*, Vol. 20, no. 4, pp. 531-547, 1994.

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

[9] 최중민, "에이전트 개요와 연구방향", *한국정보과학회지*, 제15권 제3호, pp. 7-16, 1997.

[10] S. Franklin and A. Graesser, "Is it an agent, or just a program? : A taxonomy for autonomous agents," *Proc. of Third International Workshop on Agent Theories, Architectures, and Languages*, 1996.

[11] P. Maes, "Artificial life meets entertainment: Life Like autonomous agents," *Comm. ACM*, Vol. 38, no.11, pp. 108-144, 1995.

[12] B. Lange and M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison Wesley, 1998.

[13] A. Ferrari, "JPVM: The Java Parallel Virtual Machine," Available from http:// www.cs.virginia.edu/~ajf2j/jpvm, 1999.

[14] J. Baumann, F. Hohl, K. Rothermel, and M. Straber, "Mole-Concepts of a Mobile Agent System," WWW Journal 1(3), Baltzer Science Publishers, pp. 123-137, 1998.

[15] M. Oshima, G. Karjoth, and K. Ono, Aglets Specification 1.1 Draft 0.65, IBM Corp., 1998.

[16] M. Straber, J. Baumann, and M. Schwehm, "An Agent-Based Framework for the Transparent Distribution of Computations," *PDPTA*, Vol. 1, pp. 376-382, 1999.

[17] M. Starber and M. Schwehm, "A Performance Model for Mobile Agent Systems," *PDPTA*, Vol. 2, pp. 1132-1140, 1997.

[18] FIPA, "Agent Management, FIPA version 1.0," Available from http://fipa.org/spec/ fipa98.html, 1998.

[19] M. Philippsen and M. Zenger, "JavaParty-Transparent Remote Objects in Java," *In Concurrency: Practice and Experience*, Vol. 9, pp. 1225-1242, 1997.

[20] W. Yu and Alan L. Cox, "Java/DSM: A Platform for Heterogeneous Computing," *In ACM 1997 Workshop on Java for Science and Engineering Computation*, 1997.

[21] J. Dongarra, "Linpack Benchmark-Java Version," Available from http://www.netlib. org/benchmark/linpackjava, 1998.

김 경 하

1998년 금오공과대학교 컴퓨터공학과 학사. 2001년 금오공과대학교 컴퓨터공학과 석사. 2001년 ~ 현재 쌍용정보통신 NI 기술팀 연구원. 관심분야는 병렬컴퓨팅, 이동에이전트

김 영 학

1984년 금오공과대학교 전자공학과 학사. 1989년 서강대학교 대학원 전자계산학과 석사. 1997년 서강대학교 대학원 전자계산학과 박사. 1989년 ~ 1997년 해군사관학교 전산과학과 교수. 1998년 ~ 1999년 여수대학교 멀티미디어학부 교수. 1999년 ~ 현재 금오공과대학교 컴퓨터공학부 교수. 관심분야는 병렬알고리즘, 분산 및 병렬처리 등

오 길 호

1980년 서울대학교 전기공학과 학사. 1982년 서울대학교 컴퓨터공학과 석사. 1992년 University of Florida 컴퓨터공학과 박사. 1983년 ~ 현재 금오공과대학교 컴퓨터공학부 교수. 관심분야는 병렬/분산 소프트웨어공학, 실시간 시스템, 객체지향시스템