이질적 계산 능력을 가진 NOW를 위한 공간 공유 스케쥴링 기법

(Space-Sharing Scheduling Schemes for NOW with Heterogeneous Computing Power)

김 진 성 † 심 영 철 ††

(Jin-Sung Kim) (Young-Chul Shim)

요 약 NOW(Network of Workstations)는 병렬 프로그램들을 수행하기 위한 플랫폼으로 많이 고려되어지고 있다. NOW에서 병렬 프로그램이 좋은 성능으로 실행되기 위해 해결되어야할 기본적인 문제들 중 하나가 작업의 스케쥴링 문제를 효율적으로 결정하는 것이다. 현재 NOW에 관한 대부분의 연구는 NOW를 구성하는 모든 워크스테이션이 같은 처리 능력을 가지고 있다고 가정하고 있다. 본 논문에서는 NOW를 구성하는 워크스테이션들이 다른 계산 능력을 가지고 있는 것을 고려한다. 이질적인 계산 능력을 가지고 있는 워크스테이션들로 구성된 NOW에 적용할 수 있는 10가지 공간 분할 스케쥴링 방법을 제시하고, 시뮬레이터를 통하여 이 스케쥴링 정책들을 비교한다. 시뮬레이터는 합성된 순차/병렬 부하를 입력으로 받아 병렬 작업의 응답 시간과 기다림 시간을 성능 지표로 발생시킨다. 실험 결과 워크스테이션의 계산 능력에 비례하여 병렬 프로그램을 이질적으로 분할하는 경우가 균등 분할하는 경우보다 성능이 우수함을 알 수 있었다. 병렬 프로세스를 수행하는 워크스테이션에 소유자가 돌아온 경우 병렬 프로세스를 새 유휴 워크스테이션에 이주하는 것보다는 단지 우선 순위를 낮추는 것이 높은 성능을 보여 주었다. 우선 순위 낮춤을 사용하는 이질적 분할의 경우 적응 할당 정책이 넓은 범위의 병렬 프로그램 도착시간에서 좋은 성능을 보이나 부하 불균형이 높아지는 경우 수정된 적응 할당 정책이 높은 성능을 보여준다.

Abstract NOW(Network of Workstations) is considered as a platform for running parallel programs by many people. One of the fundamental problems that must be addressed to achieve good performance for parallel programs on NOW is the determination of efficient job scheduling policies. Currently most research on NOW assumes that all the workstations in the NOW have the same processing power. In this paper we consider a NOW in which workstations may have different computing power. We introduce 10 classes of space sharing-based scheduling policies that can be applied to the NOW with heterogeneous computing power. We compare the performance of these scheduling policies by using the simulator which accepts synthetically generated sequential and parallel workloads and generates the response time and waiting time of parallel jobs as performance indices of various scheduling strategies. Through the experiments the case when a parallel program is partitioned heterogeneously in proportion to the computing power of workstations is shown to have better performance than when a parallel program is partitioned into parallel processes of the same size. When the owner returns to the workstation which is executing a parallel process, the policy which just lowers the priority of the parallel process shows better performance than the one which migrates the parallel process to a new idle workstation. Among the policies which use heterogeneous partitioning and process priority lowering, the adaptive policy performed best across the wide range of inter-arrival time of parallel programs but when the load imbalance among parallel processes becomes very high, the modified adaptive policy performed better.

· 이 논문은 1997년 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음

† 비 회 원 : 한국통신하이텔연구소 연구원
truestar@hitel.net
†† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
shim@cs.hongik.ac.kr

논문접수 : 1999년 9월 27일
심사완료 : 2000년 5월 13일

1. 서 론

컴퓨터 하드웨어 기술의 발달과 네트워크 기술의 발달로 인해 컴퓨터의 성능이 급속히 좋아지고 있으며 컴퓨터간 통신 속도도 ATM, Myrinet과 같은 기술로 인해 대단히 빨라지고 있다. 최근 이러한 조건을 기반으로 자치적으로 운영되고 있는 워크스테이션들을 네트워크로 연결시켜 MPP(Massively Parallel Processors)와 같은 수퍼 컴퓨터의 성능을 가질 수 있도록 하는 연구가 진행중이다. 이러한 시스템을 가리켜 NOW(Network of Workstations)[1]라고 한다.

어느 시점에서나 네트워크에 연결된 워크스테이션들 중에 많은 워크스테이션들이 사용되지 않고 있다. 한 통계에 의하면 하루 중 아무리 바쁜 시간대에도 20~60%의 워크스테이션들이 사용되지 않고 있다는 통계가 있다[2]. 이렇게 사용되지 않는 유휴 워크스테이션을 이용하는 것에 대한 연구가 최근 들어 활발히 진행 중에 있다. 이 대표적인 예로는 Berkeley에서 Solaris 운영체제를 기반으로 진행중인 NOW 프로젝트가 있다.

위에서 말한 Berkeley의 NOW를 비롯해 NOW에 대한 대부분의 연구들은 동일한 구조와 계산 능력을 가진 워크스테이션들로 구성된 환경 하에서 연구가 진행되었다[3-10]. 하지만 실세계에서 로컬 네트워크에 연결된 워크스테이션들은 모두 동일한 구조나 계산 능력을 지니고 있는 것은 아니다. 다양한 종류의 구조와 계산 능력을 가지는 워크스테이션들이 로컬 네트워크에 연결되어 있는 이질적인 환경이 대부분일 것이다. Carnegie Mellon의 Dome[3]이 이런 이질적 환경에서 연구중인 대표적인 예라고 할 수 있다. 따라서 이질적 구조 및 계산 능력을 가진 워크스테이션으로 구성된 NOW에서, 병렬 프로그램을 수행시키기 위한 스케쥴링 방법을 연구하는 것이 중요하다. 본 논문에서는 우선 이질적인 계산 능력을 가진 NOW 환경에 적합한 스케쥴링 방법들을 제시하고, 시뮬레이션을 통하여 성능을 비교했다.

NOW에서의 스케쥴링 문제를 고려할 때 다음과 같은 사항을 고려해야 한다.

- 작업 분할 : 하나의 병렬 작업은 다수의 프로세스들로 나뉘어 진다. 이렇게 나뉘어진 수를 병렬도(parallelism)라 한다. 이 병렬도는 병렬프로그램의 컴파일 시간(compilation time) 혹은 병렬프로그램이 제출되는 시간(submission time)에 결정된다. 이렇게 나뉘어진 프로세스들은 할당되어질 워크스테이션에 따라 그 크기가 같을 수도 있고, 다를 수도 있다.
- 워크스테이션 공유 방법 : 두 개 이상의 병렬 프로그램들이 동시에 수행될 때 각각의 병렬 프로그램을 유휴 워크스테이션들에 할당하는 방법은 두 가지가 있을 수 있다[4]. 하나는 공간 분할(space sharing) 방법이고 다른 하나는 시간 분할(time sharing) 방법이다. 공간 분할 방법은 시스템을 여러 개로 분할시키고, 각각의 분할된 영역에 하나의 병렬프로그램만이 수행된다. 시간 분할 방법은 여러 개의 병렬프로그램들이 코스케쥴링(coscheduling)[5] 등을 통하여 전체 시스템을 공유한다. 대부분의 연구에서 공간 분할 방법이 시간 분할 방법보다 우수하다는 것이 알려져 있다[6]. 이에 본 논문에서는 공간 분할 방법만을 고려한다.
- 순차/병렬 작업 사이의 상호관계 : NOW 시스템에서는 병렬 프로그램을 수행시키기 위하여 유휴 워크스테이션을 사용하기 때문에, 병렬 작업이 순차 작업의 성능에 영향을 끼치지 않도록 하는 것이 중요하다. NOW 시스템에서 병렬 프로그램을 수행 중이던 워크스테이션에 주인이 돌아와 순차 작업을 시작할 경우, 병렬 작업이 순차 작업의 성능에 영향을 끼치지 않도록 하기 위해서는 공격적인 방법(aggressive strategy) 혹은 보수적인 방법(conservative strategy)이 고려되어질 것이다. 공격적인 방법에서는 병렬 프로세스가 다른 유휴 워크스테이션으로 이주되어지는 반면 보수적인 방법에서는 병렬 프로세스의 우선 순위를 낮추어 순차 작업에 영향을 끼치지 않도록 한다. 본 연구에서는 다른 유휴 호스트로 이주시키는 것과 우선 순위를 낮추는 것을 모두 고려한다.

본 논문에서는 위 사항들을 고려하여 이질적인 계산 능력을 가진 NOW 환경에 적합한 10가지의 공간 분할 스케쥴링 방법을 제시하였다. 본 논문에서 제시한 공간 분할 스케쥴링 방법이 기존의 병렬 시스템에서의 스케쥴링 방법과 다른 점은 시스템을 구성하고 있는 CPU의 속도가 서로 다른 환경을 고려하였다는 것이다. 제시한 공간분할 스케쥴링 방법들의 성능을 비교하기 위해 시뮬레이터를 구현하였다. 시뮬레이터는 순차 부하와 병렬 부하를 입력으로 받아 병렬 프로그램 각각의 실행 시간(execution time)과 대기 시간(waiting time)을 결과로 내보낸다. 순차 부하는 [7]과 [8]을 참고하여 합성하였으며, 병렬 부하는 [9]를 참조하여 Feitelson의 병렬 부하 모델을 사용하였다.

본 논문의 구성은 다음과 같다. 2장에서는 먼저 병렬 시스템에서의 기존의 스케쥴링 방법에 대해 살펴보고 계산 능력의 이질성을 고려한 10가지 공간 분할 스케쥴링 방법을 새로이 제시한다. 3장에서는 제시된 스케쥴링

방법들을 평가하기 위하여 구현된 시뮬레이터에 대하여 기술한다. 4장에서는 시뮬레이션 및 구현을 통한 결과를 분석하고, 마지막으로 5장에서 결론을 맺는다.

2. 공간 분할 스케쥴링 방법

본 장에서는 먼저 병렬 시스템에서의 기존의 스케쥴링 방법에 대해 살펴본 다음 1장에서 언급한 고려사항들을 바탕으로 계산 능력이 다른 워크스테이션들로 구성된 NOW에 적용할 수 있는 공간 분할 스케쥴링 방법에 대하여 기술한다.

NOW를 포함한 병렬 시스템 상에서 여러 개의 작업이 동시에 수행되는 다중프로그래밍(multiprogramming) 환경에서는 병렬 작업 스케쥴링이 매우 중요한 문제가 된다. 앞에서 이미 언급한 바와 같이 병렬 작업 스케쥴링은 크게 시간 분할(time sharing) 정책과 공간 분할(space sharing) 정책으로 나눌 수 있으며 많은 연구 결과 공간 분할 정책이 시간 분할 정책보다 높은 성능을 보인다는 것이 잘 알려져 있다[6,10]. 공간 분할 정책은 다시 정적(static) 정책과 동적(dynamic) 정책으로 나눌 수 있다[11]. 정적 정책에서는 한 개의 병렬 작업의 병렬도가 수행 중 변하지 않으나 동적 정책의 경우에는 하나의 병렬 작업의 병렬도가 시스템의 부하 상태에 따라 수행 중 변경될 수 있다[12]. 그러나 동적 정책의 경우 작업의 수행 중 작업을 재분할(repartition)함으로써 발생하는 오버헤드가 너무 커서 큰 성능 저하가 발생한다는 단점이 잘 알려져 있다[13]. 그러므로 본 논문에서는 정적인 공간 분할 정책만을 고려하였다. 정적인 공간 분할 정책은 다시 고정(fixed), 가변(variable), 적응(adaptive) 정책의 셋으로 나뉜다[14]. 고정 정책에서는 병렬 시스템을 같은 크기의 파티션(partition)으로 분할하고 하나의 병렬 작업이 하나의 파티션에 할당된다. 그러므로 병렬 작업의 병렬도는 고정된 파티션의 크기에 따라 결정된다. 가변 정책에서 파티션의 크기는 사용자의 요청에 따라 병렬 작업 제출 시에 결정된다. 병렬 시스템의 상태와 무관하게 사용자가 작업을 원하는 수의 병렬도로 나눈 후 제출하면 이 병렬도의 크기와 같은 수의 CPU를 가지는 파티션이 이 작업에 할당된다. 마지막으로 적응 정책의 경우에는 시스템의 부하 상태를 고려하여 시스템에 의해 파티션의 크기가 결정된다. 여러 종류의 적응 정책이 가능하나 가장 단순한 경우로는 가용한 CPU를 현재 큐에서 기다리고 있는 작업들에게 균일하게 분할하는 방법이 있다[15]. 가변 정책의 경우 먼저 병렬 작업이 사용자에 의해 병렬 프로세스로 분할되고 이에 따라 병렬 시스템의 파티션 크기가 결정된다. 고정 정책과 적응 정책의 경우에는 시스템에 의해 병렬 시스템의 파티션 크기가 결정되고 다음 병렬 작업의 병렬도가 결정됨을 알 수 있다.

그러나 이러한 연구 결과들은 모두 병렬 시스템을 구성하는 CPU가 모두 같은 계산 능력을 가지고 있다고 가정하고 있다. MPP와 같은 시스템에서는 이 가정이 유효하나 여러 종류의 워크스테이션으로 구성된 NOW의 경우에는 이 가정이 유효하지 않은 경우가 더 많다. 그러므로 NOW에서의 스케쥴링 문제에서는 계산 능력의 이질성을 고려하는 것이 매우 중요하다. 최근 이질적인 계산 능력을 가진 CPU로 구성된 병렬 시스템에서의 스케쥴링에 관한 연구가 있었으나 다양한 스케쥴링 정책을 충분히 고려하지 못했다[16,17].

본 논문에서는 이질적인 계산 능력을 가진 워크스테이션으로 구성된 NOW와 같은 병렬 시스템에 적용할 수 있는 다양한 스케쥴링 정책들의 성능을 비교하였다. 동일한 계산 능력을 가진 워크스테이션으로 구성된 NOW를 파티션으로 분할하는 경우 파티션의 크기는 단순히 파티션 내의 워크스테이션의 수로 결정되었다. 그러나 본 논문에서는 이질적인 계산 능력을 가진 워크스테이션으로 구성된 경우에 파티션의 크기를 파티션 내의 모든 워크스테이션들의 계산 능력의 합으로 정의하였다.

표 1 공간 분할 스케쥴링 방법의 구분

분할 및 할당 방법 / 병렬/순차 부하 충돌	균등 분할 (Homogeneous Partitioning)		이질적 분할 (Heterogeneous Partitioning)		
	균등 할당 (Homogeneous Allocation)	이질적 할당 (Heterogeneous Allocation)	고정할당 (Fixed)	적응할당 (Adaptive)	수정된 적응 할당 (Modified Adaptive)
이주 (Migration)	HoHoMi	HoHeMi	HeMi-F	HeMi-A	HeMi-M
우선 순위 낮춤 (Priority Lowering)	HoHoLo	HoHeLo	HeLo-F	HeLo-A	HeLo-M

본 논문에서는 정적인 공간 분할 스케쥴링 방법을 병렬 작업 분할 및 할당 방법과 병렬/순차 부하 충돌 시 해결 방법에 따라 표 1과 같이 10가지 방법을 고려하였다. 먼저 공간 분할 스케쥴링 방법을 작업의 분할 방법과 할당 방법에 따라 균등 분할과 이질적 분할의 둘로 나누었다. 균등 분할의 경우에는 작업이 사용자에 의해 균등한 크기의 병렬 프로세스들로 분할되는 것으로 가

변 정책에 해당한다. 균등 분할은 다시 균일한 계산 능력을 가진 경우의 가변 정책과 같이 하나의 워크스테이션에 하나의 병렬 프로세스가 할당되는 균등 분할과 워크스테이션의 계산 능력에 따라 다른 수의 병렬 프로세스가 할당되는 이질적 할당의 두 경우로 나눈다. 이질적 분할의 경우는 고정 할당, 적응 할당, 수정된 적응 할당의 셋으로 나눈다. 고정 할당은 균일한 계산 능력을 가진 병렬 시스템에서의 고정 정책에 해당하며 적응 할당과 수정된 적응 할당은 적응 정책에 해당한다. 그러나 균일한 계산 능력을 가진 경우와는 달리 파티션의 크기를 결정할 때 파티션 내의 워크스테이션의 수를 고려하는 대신 파티션 내의 워크스테이션의 총 계산 능력을 고려한 점이 다르다. 그리고 각각의 분할 및 할당 방법은 병렬 부하와 순차 부하가 충돌할 때 이주 정책을 쓰는지 또는 우선 순위 낮춤 정책을 쓰는가에 따라 다시 두 가지로 나눈다.

균등 분할의 경우는 병렬 프로그램의 병렬도가 컴파일 시간에 결정되는 형태의 병렬 작업을 고려한 방법이다. 이런 병렬 작업을 고정된 작업(rigid job)이라고 한다. 한편 이질적 분할의 경우는 병렬 프로그램의 병렬도가 병렬 프로그램이 제출된 시간에 결정되는 형태의 병렬 작업으로서 이런 병렬 작업을 가변 작업(moldable job)이라고 한다[18].

이후 2장의 하부 절에서는 표 1과 같이 구분된 각각의 공간 분할 스케쥴링 방법에 대하여 자세히 설명한다.

2.1 균등 분할 및 균등 할당 방법 (HoHoMi, HoHoLo)

HoHoMi 방법은 병렬 프로그램이 도착하였을 때 사용자에 의해 정해진 병렬도의 개수로 병렬 프로세스들을 만들고 각 프로세스는 똑같은 크기의 데이타를 처리하도록 나누어 하나의 워크스테이션에 하나의 프로세스를 할당하는 방법을 말한다. 이때 할당되어지는 워크스테이션은 유휴 워크스테이션 중 가장 계산 능력이 큰 것부터 선택되어지도록 하였다. 할당된 워크스테이션에 순차부하가 생겼을 경우에는 병렬 프로세스를 다른 유휴 워크스테이션으로 이주시킨다. 이주될 대상이 되는 유휴 워크스테이션을 찾는 순서는 현재 수행 중이던 워크스테이션과 같은 계산 능력을 갖는 워크스테이션을 먼저 찾고, 없을 경우 다음으로 그보다 좋은 계산 능력을 갖는 워크스테이션을 오름차순으로 검색해 본다. 이 경우에도 실패하면 마지막으로 현재 수행 중이던 워크스테이션의 계산능력보다 낮은 워크스테이션을 내림차순으로 검색하여 유휴 워크스테이션을 찾는다. 이렇게 해도 이주될 워크스테이션이 없을 때는 이주 큐(migration queue)에서 새로운 유휴 워크스테이션이 생길 때까지 기다리게 된다.

HoHoLo 방법은 병렬 프로세스들이 할당된 워크스테이션에 순차부하가 생겼을 경우에 병렬 프로세스의 우선 순위를 순차부하보다 낮추는 것을 제외하고, 나머지 분할 및 할당 방법은 HoHoMi 방법과 동일하다.

2.2 균등 분할 및 이질적 할당 방법 (HoHeMi, HoHeLo)

HoHeMi 방법은 병렬 프로그램이 들어왔을 때 사용자에 의해 정해진 병렬도의 개수로 병렬 프로세스들을 만들고, 각 프로세스가 똑같은 크기의 데이타를 처리하도록 나누는 과정까지는 HoHoMi 방법과 동일하다. 하지만 이후 유휴 워크스테이션에 병렬 프로세스를 할당하는 과정에서 HoHoMi 방법과 달라지게 된다. 본 논문에서 고려하는 NOW 시스템은 이질적 계산 능력을 가지는 워크스테이션들로 구성되어 있다. 따라서 HoHoMi 방법과 같이 하나의 워크스테이션에 동일한 크기로 나뉘어진 하나의 병렬 프로세스만을 할당하는 것이 아니라, 할당되어질 유휴 워크스테이션의 계산 능력에 비례하여 병렬 프로세스를 할당하는 것이다. 예를 들어 병렬도가 8인 병렬 프로그램을 워크스테이션에 할당하려고 할 때, 유휴 워크스테이션이 800MIPS 1개, 400MIPS 1개, 200MIPS 2개, 100MIPS 5개가 있었다고 하자. 이때 선택되어지는 유휴 워크스테이션은 가장 계산 능력이 큰 유휴 워크스테이션부터 선택되어 800MIPS 1개, 400MIPS 1개, 200MIPS 2개가 선택되어 지고, 병렬 프로세스들은 선택된 워크스테이션의 계산 용량의 비율에 따라 4개, 2개 1개씩 할당되어지게 된다. 결국 각각의 병렬 프로세스가 200MIPS의 계산 용량을 할당받게 되어 모든 병렬 프로세스가 균등한 계산 용량을 할당받게 된다. 위의 예에서와 같이 유휴 워크스테이션의 선택은 가장 계산 용량이 큰 것부터 선택되어지되 각각의 병렬 프로세스들이 균등한 계산 용량을 할당받을 수 있도록 선택되어 진다. 그리고 할당된 워크스테이션에 순차부하가 생겼을 경우에는 병렬 프로세스를 다른 유휴 워크스테이션으로 이주시킨다.

HoHeLo 방법은 병렬 프로세스들이 할당된 워크스테이션에 순차부하가 생겼을 경우에 병렬 프로세스의 우선 순위를 순차부하보다 낮추는 것을 제외하고, 나머지 분할 및 할당 방법은 HoHeMi 방법과 동일하다.

2.3 이질적 분할 및 고정 할당 방법 (HeMi-F, HeLo-F)

HeMi-F 방법은 이름에서도 짐작할 수 있듯이 병렬 프로그램이 할당받을 수 있는 계산 용량을 고정시키는

방법이다. 즉 NOW 시스템이 가지고 있는 총 계산 용량(Total Computing Power)을 TCP라고 할 때, 임의의 수 N으로 나뉘어진 TCP/N 만큼의 계산 용량이 하나의 병렬 프로그램에 할당되어지도록 하는 방법이다. 즉, NOW 시스템의 총 계산 용량이 10000MIPS 라고 할 때, N이 5로 주어졌다면 하나의 병렬 프로그램이 받을 수 있는 계산 용량은 *10000 / 5* 가 되어 *2000MIPS* 가 된다. 이렇게 해서 하나의 병렬 프로그램이 받을 수 있는 계산 용량이 결정되면 이 계산 용량과 같거나 최대한 가까운 계산 용량을 가지는 유휴 워크스테이션들의 집합을 선택한다. 예를 들어 위와 같이 하나의 병렬 프로그램이 받을 수 있는 계산 용량이 2000MIPS로 결정되었을 때, 유휴 워크스테이션이 800MIPS 1개, 400MIPS 2개, 200MIPS 4개, 100MIPS 10개가 있었다고 하자. 그렇다면 가장 계산 능력이 큰 유휴 워크스테이션부터 선택되어 800MIPS 1개, 400MIPS 2개, 200MIPS 2개가 선택되어진다. 이렇게 선택되어진 워크스테이션들 각각에게 하나의 병렬 프로세스가 할당되어진다. 각각의 병렬 프로세스들의 크기 비율은 할당되어진 워크스테이션의 계산 용량에 비례하게 된다. 즉 병렬 프로그램은 8:4:4:2:2 비율의 크기를 갖는 5개의 병렬 프로세스로 분할되고, 8 크기의 프로세스는 800MIPS 워크스테이션에, 4 크기의 프로세스는 400MIPS 워크스테이션에, 2 크기의 프로세스는 200MIPS 워크스테이션에 할당된다. 그리고 할당된 워크스테이션에 순차부하가 생겼을 경우에는 병렬 프로세스를 다른 유휴 워크스테이션으로 이주시킨다.

HeLo-F 방법은 병렬 프로세스들이 할당된 워크스테이션에 순차부하가 생겼을 경우에 병렬 프로세스의 우선 순위를 순차부하보다 낮추는 것을 제외하고, 나머지 분할 및 할당 방법은 HeMi-F 방법과 동일하다.

2.4 이질적 분할 및 적응 할당 방법 (HeMi-A, HeLo-A)

HeMi-A 방법은 HeMi-F 방법과는 달리 NOW 시스템의 가용 계산 용량에 따라 병렬 프로그램이 할당받을 수 있는 계산 용량이 달라지는 것이다. HeMi-F 방법의 경우, 수행되어지기 위해 큐(Queue)에서 기다리고 있는 다른 병렬 프로그램이 없을 지라도 수행될 병렬 프로그램은 고정된 계산 용량만을 할당받게 된다. 반면 HeMi-A 방법은 현재 수행되어지기 위해 큐에서 기다리고 있는 병렬 프로그램들이 가용 계산 용량을 동등하게 모두 활용하는 방법이다. 하지만 너무 많은 병렬 프로그램들이 큐에서 기다리고 있을 때에는 각 병렬 프로그램이 HeMi-F 방법에서 받는 양만큼의 계산 용량을 받을 수 있도록 보장해 준다. 예를 들어 유휴 워크스테이션들의 계산 용량 총 합을 나타내는 가용 총 계산 용량(Available Total Computing Power, ATCP)이 800 MIPS이고, 병렬 프로그램이 받아야 하는 최소 계산 용량이 200 MIPS라 할 때, 현재 수행되어지기 위해 큐에서 기다리고 있는 병렬 프로그램이 2개라면 각각의 병렬 프로그램은 400 MIPS의 계산 용량을 할당받게 된다. 만약 큐에서 기다리고 있는 병렬 프로그램이 5개 이상이라면 먼저 큐에 들어와서 기다리고 있던 4개의 병렬 프로그램에게 200 MIPS씩 계산 용량을 할당하게 되고, 나머지 병렬 프로그램들은 새로운 유휴 워크스테이션이 생길 때까지 기다리게 된다. 이렇게 하나의 병렬 프로그램이 할당받을 수 있는 계산 용량이 결정되면 이후 유휴 워크스테이션의 선택 및 병렬 프로세스들의 할당은 HeMi-F 방법과 동일하다. 그리고 할당된 워크스테이션에 순차부하가 생겼을 경우에는 병렬 프로세스를 다른 유휴 워크스테이션으로 이주시킨다.

HeLo-A 방법은 병렬 프로세스들이 할당된 워크스테이션에 순차부하가 생겼을 경우에 병렬 프로세스의 우선 순위를 순차부하보다 낮추는 것을 제외하고, 나머지 분할 및 할당 방법은 HeMi-A 방법과 동일하다.

2.5 이질적 분할 및 수정된 적응 할당 방법 (HeMi-M, HeLo-M)

HeMi-M 방법은 HeMi-F 방법과 HeMi-A 방법의 중간 형태라 할 수 있다. HeMi-M 방법에서는 최대 계산 용량(*Maximum Computing Power*, MaxCP)과 최소 계산 용량(Minimum Computing Power, MinCP)을 나타내는 두 개의 상수를 정의한다. MaxCP는 하나의 병렬 프로그램이 최대로 할당받을 수 있는 계산 용량이고, MinCP는 하나의 병렬 프로그램이 최소한 할당받아야 하는 계산 용량을 말한다. MaxCP는 TCP보다는 작거나 같고 MinCP보다는 크거나 같은 값이고, MinCP는 HeMi-F 방법에서의 TCP/N 값이 된다. 결국 하나의 병렬 프로그램이 할당받을 수 있는 계산 용량은 MaxCP와 MinCP 사이의 값이 된다. HeMi-A 방법에서는 MinCP 값만을 두어 하나의 병렬 프로그램이 받아야하는 최소 계산 용량만을 고려했지만, HeMi-M 방법에서는 하나의 병렬 프로그램이 받을 수 있는 최대 계산 용량을 더 고려했다는 것이 HeMi-A 방법과 HeMi-M 방법의 차이점이다. 예를 들어 MaxCP가 1000MIPS이고, MinCP가 200MIPS라고 하자. ATCP가 3000MIPS일 때 현재 2개의 병렬 프로그램이 수행되어지려 한다면, HeMi-A 방법의 경우 2개의 병렬 프로그램 각각이 1500MIPS씩 계산 용량을 할당받을 것이다.

하지만 HeMi-M 방법의 경우 MaxCP가 1000MIPS로 제한되어 있기 때문에 2개의 병렬 프로그램 각각이 1000MIPS씩 계산 용량을 할당받고 나머지 1000MIPS는 다음에 도착되는 병렬 프로그램을 위해 남겨두는 것이다. 이렇게 하나의 병렬 프로그램이 할당받을 수 있는 계산 용량이 결정되면 이후 유휴 워크스테이션의 선택 및 병렬 프로세스들의 할당은 HeMi-F 방법과 동일하다. 그리고 할당된 워크스테이션에 순차부하가 생겼을 경우에는 병렬 프로세스를 다른 유휴 워크스테이션으로 이주시킨다.

HeLo-M 방법은 병렬 프로세스들이 할당된 워크스테이션에 순차부하가 생겼을 경우에 병렬 프로세스의 우선 순위를 순차부하보다 낮추는 것을 제외하고, 나머지 분할 및 할당 방법은 HeMi-M 방법과 동일하다.

3. 시뮬레이터의 설계

본 장에서는 본 논문에서 제시한 공간 분할 스케쥴링 방법들의 성능을 평가하기 위해 구현된 시뮬레이터의 구조에 대하여 살펴보고, 시뮬레이터의 입력 값으로 사용되어지는 순차부하 및 병렬 부하에 대하여 알아본다.

3.1 시뮬레이터의 구조

시뮬레이터의 구조는 그림 1과 같으며, 시뮬레이터는 이벤트(Event) 처리 방식으로 진행된다. 시뮬레이터는 순차 부하와 병렬 부하를 입력으로 받아들여 각각의 스케쥴링 방법에 따라 병렬 프로그램들을 수행시킨 뒤 실행 시간(Execution Time)과 대기 시간(Waiting Time)을 출력으로 내보낸다. 대기 시간은 병렬 작업이 제출되어 시스템 자원을 할당받기까지 대기 큐에서 기다리는 시간이고 실행 시간은 시스템 자원을 할당받아 실제로 수행되어지는 시간이다.

그림 1 시뮬레이터의 구조

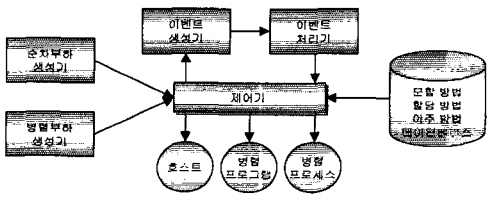

그림 2 시뮬레이터의 세부구조

그림 2는 시뮬레이터의 세부구조를 나타내며 각 부분에 대해서 간략하게 살펴보겠다.

- 순차 부하 생성기 : 제어기에서 한 호스트의 상태(BUSY, IDLE)를 넘겨받아 3.2절의 내용에 따라서 다음 상태(BUSY면 IDLE, IDLE이면 BUSY)와 상태가 유지되는 시간을 넘겨준다.
- 병렬 부하 생성기 : 3.3절과 같은 병렬 부하를 생성하여 제어기의 요구에 따라 넘겨준다.
- 이벤트 생성기 : 제어기로부터 새로 생성되어야 할 이벤트를 넘겨받아 이벤트 리스트에 추가한다.
- 이벤트 처리기 : 이벤트의 종류에 따라 제어기가 해야 할 일을 정해준다.
- 제어기 : 이벤트 처리기로부터 받은 지정된 작업들을 수행하며 호스트, 병렬 프로그램, 병렬 프로세스들을 관장한다. 또한 병렬 프로그램의 스케쥴링을 담당한다.
- 호스트 : IDLE이나 BUSY 상태를 가지며 상태 유지 시간 등을 갖는다.
- 병렬 프로그램 : 수행중인 병렬 프로그램의 시작 시간, 종료 시간, 병렬도 등을 갖는다.
- 병렬 프로세스 : 수행중인 병렬 프로그램에 대한 병렬 프로세스들로써 COMPUTING, COMMUNICATING, MIGRATING, READY2RUN, BLOCK4YNC, BLOCK4SWITCH, BLOCK4MIG등의 상태를 가진다.
- 데이타베이스 : 각각의 스케쥴링 방법에 대한 분할 및 할당 방법 그리고 이주 정책에 대한 정보를 가진다.

본 논문에서 구현한 시뮬레이터는 Sun Ultra Enterprise 3000, Solaris 2.5.1에서 5000라인의 C 코드로 구현되었다.

3.2 순차 부하 모델

순차 부하란 병렬 프로그램에 의해서 생기는 부하를 제외한, 사용자가 실행한 순차 프로그램에 의해 생기는 부하를 말한다. 본 논문에서 시뮬레이터의 입력으로 사용되어지는 순차 부하는 합성에 의하여 만들어졌다. 이에 대해 간단히 살펴보면 다음과 같다.

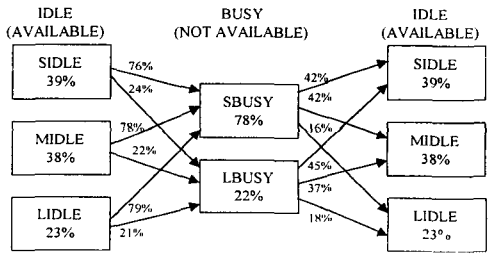

그림 3 상태 전이 확률

워크스테이션은 순차 부하가 전혀 없는 유휴(IDLE) 상태와, 순차 부하가 있는 바쁜(BUSY) 상태중의 하나에 있고, 각각의 워크스테이션이 현재의 상태를 유지할 확률과 다른 상태로의 전이가 일어날 확률은 그림 3과 같다[2].

그림 3에서와 같이 유휴 상태와 바쁜 상태를 평균 지속 시간에 따라 각각 3가지와 2가지로 세분화하였다. 유휴 상태는 짧은 유휴 상태(SIDLE)와 긴 유휴 상태(LIDLE) 그리고 이들의 중간 정도의 유휴 상태(MIDLE)로 나누었다. 그리고 바쁜 상태는 짧은 바쁜 상태(SBUSY), 긴 바쁜 상태(LBUSY)로 나누었다. 이때 상태 지속 시간은 다음 식에 의해 계산된다[2].

$T = - t_{avg} \cdot ln(1 - R)$

where T : 상태 지속 시간

$_{avg}$: 평균 지속 시간

R : $0 < R < 1$인 난수

위 식에서의 평균 지속 시간은 SIDLE 180초, MIDLE 1500초, LIDLE 18000초, SBUSY 320초, LBUSY 2600초로 하여 평균 유휴 워크스테이션의 비율은 80%로 하였다.

3.3 병렬 부하 모델

병렬 부하란 병렬 프로그램에 의해서 생기는 부하를 말한다. 본 논문에서는 병렬 프로그램들 중에서 Single Program Multiple Data(SPMD)를 위주로 설명한다. 본 논문에서 고려하는 SPMD 프로그램은 그림 4와 같은 방식으로 동작한다[19]. 병렬 프로그램을 구성하는 각 프로세스들은 각각의 계산을 마치고 통신이 필요한 프로세스들끼리 통신을 한다. 이러한 계산과 통신을 합쳐 한 단계(phase)라고 한다. 병렬 프로그램은 정해진 수만큼의 단계를 반복하여 모든 계산을 마치면 종료한다.

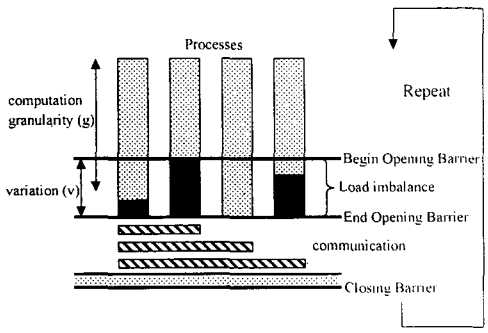

그림 4 병렬 프로그램 SPMD 모델

병렬 부하는 병렬 프로그램의 특성에 대한 자료를 가지고 있다. 통신 형태, 평균 계산 시간(g), 부하 불균형(load-imbalance), 통신 시간, 반복 회수, 병렬도 등이다. 통신 형태는 병렬 프로그램을 구성하는 프로세스들이 어떤 형태로 통신하는지에 관련된 것으로 BARRIER, NEWS, TRANSPOSE 등을 고려할 수 있다[19]. 평균 계산 시간(g)은 한 단계 내에서 각 프로세스가 수행해야 할 계산의 평균 시간을 말한다. 각 프로세스는 처리해야할 데이타가 다르기 때문에 g는 처리해야할 데이타의 상대적인 비율(d)과 그 프로세스가 수행되어지고 있는 워크스테이션의 상대적인 계산 용량(p)에 따라 달라지게 된다. 부하 불균형(v)은 한 단계에서 가장 짧은 계산 시간과 가장 긴 계산 시간의 차이를 만드는 변수로서 v는 0 ~ 2g 사이의 값을 갖게 된다. 그래서 한 프로세스가 한 단계에서 가지는 계산 시간은 다음과 같이 결정된다.

$T_{comp} = (g \pm v/2) * d * p$

where T_{comp} : 계산 시간

g : 평균 계산 시간

d : 처리 해야할 데이타의 비율

p : 계산 용량의 비율

v : 계산 시간 변화량

통신 시간은 한 단계 내에서 모든 프로세스가 통신을 완료하는데 걸리는 시간이다. 반복 회수는 병렬 프로그램이 몇 번의 단계를 거쳐야 종료하는가를 나타낸다. 병렬도는 하나의 병렬 프로그램이 몇 개의 프로세스들로 이루어져 있는가를 나타낸다.

병렬 부하의 생성은 Feitelson이 6가지의 병렬 기계들의 관측을 통해 모델링한 병렬 부하 생성기를 사용하였다[9]. Feitelson의 병렬 부하 생성기로부터 병렬 작업의 실행 시간과 병렬도를 얻고, 이를 고정 작업 혹은 가변 작업의 형태에 맞도록 변형하여 사용하였다. 고정 작업의 경우에는 Feitelson의 병렬 부하 생성기로부터 나오는 병렬 작업의 실행 시간과 병렬도를 그대로 사용하였다. 반면 가변 작업의 경우에는 병렬도가 병렬 작업이 제출되는 시간에 결정이 되기 때문에 Feitelson의 병렬 부하 생성기에서 나오는 병렬도를 그대로 사용할 수 없다. 이에 본 논문에서는 먼저 [20]에서 제안된 함수로 병렬 작업이 여러 워크스테이션에서 수행되었을 때의 수행시간을 하나의 워크스테이션에서 수행될 때의 수행시간으로 변경했다. 다음 병렬도는 스케줄러에 의해 결정된 값을 사용했으며, 다시 [20]의 함수를 사용하여 병

렬 작업을 스케쥴러에 의해 결정된 병렬도만큼 병렬화 시켰을 때의 수행시간을 계산하여 병렬 작업의 수행시간으로 사용하였다. [20]에서 제안된 함수는 다음과 같다.

$$\mu_p^{-1} = [\alpha + (1-\alpha)/p]\mu_1^{-1}$$

여기서 μ_p^{-1}은 병렬 작업이 p개의 워크스테이션에서 수행되었을 때의 평균 수행 시간을 나타낸다. μ_1^{-1}은 병렬 작업이 하나의 워크스테이션에서 수행되었을 때의 평균 수행 시간을 나타낸다. p는 병렬도를 나타내며, α는 병렬화에 따른 속도 향상 계수로 0과 1사이의 값을 가진다. 그리고 병렬 프로그램들의 상호 도착 시간은 포아송 분포를 따르도록 하였다.

4. 시뮬레이션 및 구현 결과 분석

본 장에서는 2장에서 제시한 각각의 스케쥴링 방법을 시뮬레이션과 구현을 통하여 그 성능을 비교해 보고, 결과를 분석해 본다.

시뮬레이션은 3장에서 설명한 시뮬레이터를 통하여 수행하였으며, 3.3절에서 설명한 병렬 작업 1000개를 수행하여 이중 처음 100개와 마지막 100개를 제외한 800개의 병렬 작업에 대한 평균 응답 시간을 성능 지표로 삼았다. 응답 시간이란 대기 시간과 실행 시간을 합친 시간이다.

4.1 시뮬레이션 환경

워크스테이션 클러스터는 128대의 워크스테이션으로 구성되었으며, 각 워크스테이션의 처리능력은 4가지를 두었다. 가장 느린 워크스테이션을 기준으로 그에 2배, 4배, 8배가 되도록 구성하였고, 처리 능력에 따른 워크스테이션의 구성 비율은 표 2와 같다.

표 2 처리 능력에 따른 워크스테이션 구성 비율

처리 능력(MIPS)	100	200	400	800
개수	26	51	38	13

실험을 하기에 앞서 이질적 분할 및 고정 할당 방법에서의 N값과 이질적 분할 및 수정된 적응 방법에서의 MaxCP값을 결정해야 한다. 스케쥴링 방법의 성능을 극대화 할 수 있는 N과 MaxCP값을 찾기 위해 다음과 같은 실험을 하였다.

먼저 이질적 분할 및 고정 할당 방법에서 N값을 변화시켜가며 성능 변화를 관찰하였다. HeMi-F, HeLo-F 방법에서 최적의 N값을 찾기 위해 N값을 4부터 32까지

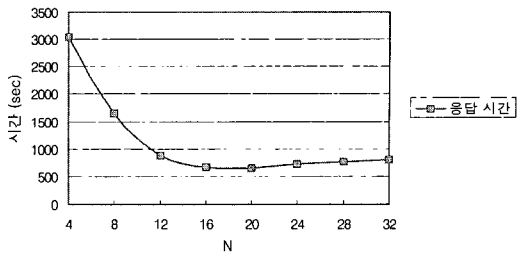

그림 5 N값 변화에 따른 HeMi-F 방법의 응답 시간

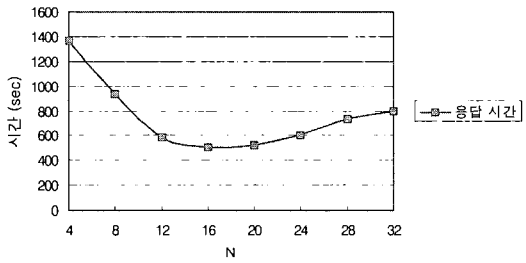

그림 6 N값 변화에 따른 HeLo-F 방법의 응답 시간

4씩 증가시켜가며 성능 변화를 관찰하였다.

그림 5는 N값 변화에 따른 HeMi-F 방법의 응답 시간을 나타내고 있다. 이 그래프에서 N값이 16까지 응답 시간이 감소하다 N값이 20을 지나면서 응답 시간이 증가하기 시작함을 알 수 있다. 그림 6은 HeLo-F 방법의 경우를 나타낸다. 이 그래프 역시 N값이 16을 기점으로 응답 시간이 감소추세에서 증가추세로 변하는 것을 알 수 있다. 이는 N값이 너무 작을 경우 하나의 병렬 프로그램이 많은 자원을 사용하여 뒤에 도착하는 병렬 프로그램들이 할당받을 자원이 없게 되고, 결국 대기 시간이 길어지게 되어 전체적인 성능 저하를 가져오는 것이고, 반대로 N값이 너무 클 경우 하나의 병렬 프로그램이 할당받을 수 있는 자원이 너무 작게 고정되어 시스템의 자원을 최대한 사용하지 못하고 결국 전체적인 성능 저하를 가져오는 것이다. 따라서 이질적 분할 및 고정 할당 방법에서 최적의 N값이 16임을 알 수 있다. 이에 본 논문의 이후 실험은 N값을 16으로 고정하여 실험을 하였다.

다음에는 이질적 분할 및 수정된 적응 방법에서 MaxCP값을 변화시켜가며 성능 변화를 관찰하였다.

그림 7과 그림 8은 MaxCP값을 TCP의 10%에서 100%까지 10%단위로 변화시켰을 때 HeMi-M 방법과 HeLo-M 방법에서의 응답 시간을 나타내는 것이다. 이

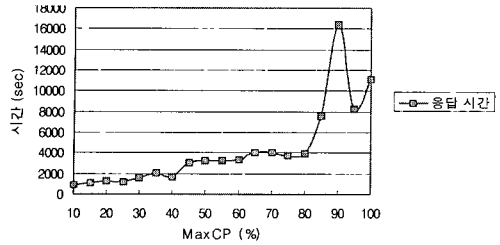

그림 7 MaxCP값 변화에 따른 HeMi-M 방법의 응답 시간

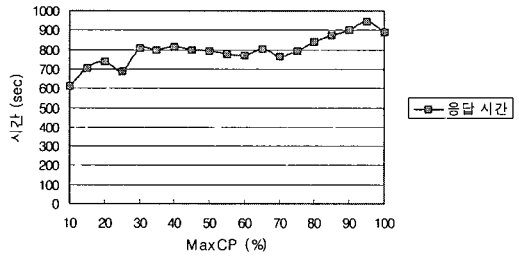

그림 8 MaxCP값 변화에 따른 HeLo-M 방법의 응답 시간

실험에서 MaxCP값을 TCP의 10%부터 시작한 이유는 이전 실험에서 결정된 N값을 바탕으로 MinCP를 계산했을 때 TCP의 9%정도가 나왔기 때문이다. 그림 7과 그림 8에서 쉽게 알 수 있듯이 MaxCP값이 TCP의 10%일 때 HeMi-M 방법과 HeLo-M 방법이 가장 좋은 성능을 나타내었다. 이는 현재 사용할 수 있는 계산 용량이 충분히 많이 있지만 다음에 도착하는 병렬 프로그램을 위해 가용 계산 용량을 다 사용하지 않고 남겨놓는 것이 좋은 결과를 낳은 것이다. 이에 본 논문에서는 이후 실험에서 MaxCP값을 TCP의 10%로 고정하였다.

4.2 균등 분할 방법과 이질적 분할 방법의 성능 평가

균등 분할 방법과 이질적 분할 방법의 성능을 평가하기 위해 앞 절에서 실험한 결과를 바탕으로 N값과 MaxCP값을 결정하였다. 먼저 이주 비용이 프로세스 이주를 사용하는 스케쥴링에 어떠한 영향을 주고 우선 순위 낮춤에 따른 느려짐 비율이 우선 순위 낮춤을 사용하는 스케쥴링에 어떠한 영향을 주는지를 알아보기 위해 이주 비용과 우선 순위 낮춤에 따른 느려짐 비율을 변화시켜가며 실험하였다.

그림 9는 이주 비용을 0.24초에서 0.54초까지 0.1초씩 증가시켰을 때, 프로세스 이주를 사용하는 스케쥴링 방법들의 응답 시간을 나타내고 있다. 병렬 프로그램의 평균 상호 도착 시간은 10초로 하였고, 부하 불균형은 없으며, 병렬 프로그램 모델은 병렬 프로그램의 계산 시간과 통신 시간의 비율이 큰 모델(coarse-grain model)을 사용하였다.

프로세스가 이주하는데 필요한 시간, 즉 이주 비용은 시스템 성능 뿐 아니라 네트워크의 속도에 따라 다르다. 예를 들면 Sprite[21]에서는 0.33초, ATM으로 연결된 GLUNIX에서는 0.25~0.5초 정도가 걸리고 [22]에서는 약 0.44초 정도가 필요함이 알려졌다. 본 논문에서는 이를 바탕으로 이주 비용의 범위를 0.44초에서 0.1초 단위로 증감시켜 GLUNIX의 이주 비용 범위와 비슷한 0.24초에서 0.54초로 결정하였다. 이 실험에서는 HeMi-F 방법이 가장 좋은 성능을 나타내었다. 그리고 이주 비용이 스케쥴링 방법들의 응답 시간에 커다란 영향을 미치지 않음도 알 수 있다.

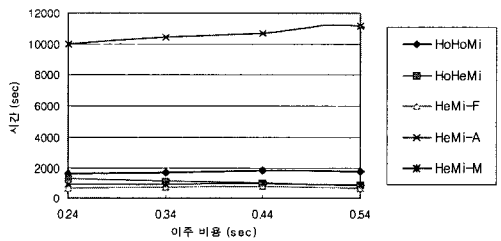

그림 9 이주 비용 변화에 따른 응답 시간

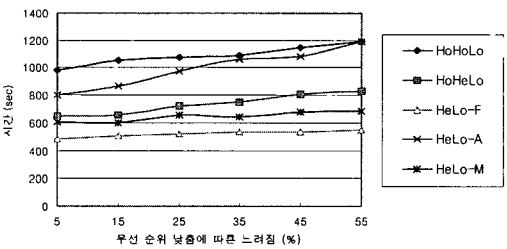

그림 10 우선 순위 낮춤에 의한 느려짐 비율 변화에 따른 응답 시간

병렬 프로세스가 수행중인 워크스테이션에 순차부하가 발생했을 때 순차부하에 영향을 주지 않기 위해 순차부하보다 병렬 프로세스의 우선 순위를 낮추면 병렬 프로그램의 수행 시간이 느려지게 된다. 이 느려짐의 비율을 알아보기 위해 본 논문에서는 간단한 실험을 하였다. 부동 소수점 연산만을 수행하는 동일한 프로세스를 우선 순위를 낮추어 실행해보고 또 낮추지 않고 실행해 본 결과 우선 순위를 낮추고 실행했었을 때의 실행 시

간이 우선 순위를 낮추지 않고 실행했을 때보다 약 10%~15%정도 느려짐을 알 수 있었다. 그림 10에서는 우선 순위 낮춤에 의한 느려짐 비율 범위를 5%에서 55%까지 확장하여 10%씩 증가시켰을 때, 우선 순위 낮춤을 사용하는 스케쥴링 방법들의 응답 시간을 나타내고 있다. 이 실험에서는 HeLo-F 방법이 다른 모든 스케쥴링 방법보다 우수한 성능을 나타내었다. 그림 9와 그림 10에서 고정 할당 방법이 좋은 성능을 나타낸 이유는 병렬 프로그램의 평균 상호 도착 시간을 10초로 하여 병렬 부하가 너무 컸기 때문으로 볼 수 있다. 즉, 적응 할당 방법은 하나의 병렬 프로그램이 많은 계산 용량을 차지하게 되어 이 병렬 프로그램의 수행 시간이 길어질 경우 다음에 들어오는 병렬 프로그램에 할당할 계산 용량이 없게 되어 이후 도착되는 병렬 프로그램들의 대기 시간이 길어지게 된다. 이러한 이유로 적응 할당 방법은 고정 할당 방법보다 나쁜 성능을 나타내었다. 수정된 적응 할당 방법은 적응 할당 방법보다는 월등히 좋은 성능을 나타내었지만 고정 할당 방법보다는 약간 나쁜 성능을 나타내었다.

표 3은 이주 비용이 0.24초일 때, 이주를 고려한 스케쥴링 방법들의 응답시간과 우선 순위 낮춤에 의한 느려짐 비율이 50%일 때, 우선 순위 낮춤을 고려한 스케쥴링 방법들의 응답시간을 비교한 것이다. 표 3을 통하여 우리는 이질적 분할을 고려한 방법이 균등 분할을 고려한 방법보다 대부분 우수한 성능을 나타냄을 알 수 있다. 이중에서도 HeLo-F 방법이 가장 좋은 성능을 나타내고 있다. 그리고 순차 부하 및 병렬 부하 충돌시 해결책으로는 이주를 고려하는 방법보다는 우선 순위 낮춤을 고려하는 방법이 좋은 성능을 나타냄을 알 수 있다. 이 실험을 바탕으로 이후 실험들은 이질적 분할 방법들에 대해서 심층적인 실험을 하였다.

표 3 모든 스케쥴링 방법들의 성능 비교

스케쥴링 방법	응답 시간 (sec)
HoHoMi	1694.938
HoHoLo	1193.026
HoHeMi	1359.702
HoHeLo	830.043
HeMi-F	659.262
HeLo-F	551.455
HeMi-A	9949.901
HeLo-A	1192.931
HeMi-M	936.701
HeLo-M	684.352

이질적 분할 방법들에 대한 심층적인 실험을 하기에 앞서 HeMi-A 방법이 나머지 이질적 스케쥴링 방법에 비하여 상당히 나쁜 원인을 분석해 보았다. HeMi-A 방법의 경우 NOW 시스템의 총계산 용량을 사용하게 된다. 따라서 HeMi-A 방법에서는 순차/병렬 부하 충돌시 이주할 유휴 워크스테이션이 없어서 유휴 워크스테이션이 생길 때까지 큐에서 기다리는 시간이 많아지게 된다. 이로 인해 적응 스케쥴링 방법의 경우 성능이 급격히 나빠지게 된 것이다.

본 논문에서는 이런 이유로 NOW 시스템에 이주를 대비하여 일정량의 계산 용량의 여유분을 두어 실험을 해 보았다. 그림 11은 NOW 시스템의 여유분 변화에 따른 이주를 사용하는 공간 분할 스케쥴링 방법들의 성능을 비교한 것이다. 이 때 병렬 프로그램간의 상호도착 시간은 과도한 병렬 부하를 피하기 위해 평균 200초로 두었고, 여유분은 TCP의 0%~10%까지 2%간격으로 변화시켜가며 실험하였다.

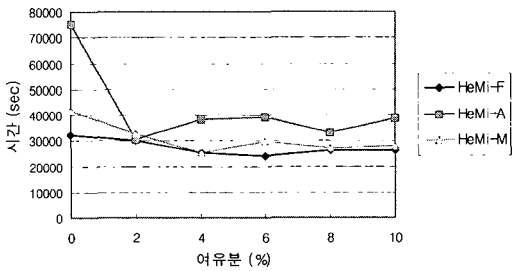

그림 11 유휴 워크스테이션 여유분 변화에 따른 공간 분할 스케쥴링 방법들의 응답시간

그림 11의 결과를 분석해 보면 HeMi-A 방법의 경우 여유분이 없을 때에는 HeMi-F 방법이나 HeMi-M 방법에 비하여 응답시간이 매우 나쁘다는 것을 알 수 있다. 하지만 여유분이 2% 이상 있을 때에는 HeMi-F 방법이나 HeMi-M 방법과도 견주어 볼만큼 성능이 향상됨을 알 수 있다. 이주를 고려한 방법에서는 이주에 대비해 여유분을 두는 것이 스케쥴링 방법의 성능향상에 도움을 주는 것을 확인할 수 있었으므로 본 논문의 이후 실험에서는 이주를 고려한 이질적 분할 방법들 모두 3%의 여유분을 두어 실험을 하였다.

4.3 이질적 분할 방법에 대한 심층적인 성능 평가

본 논문에서는 4.2절의 실험을 통하여 이질적 계산 능력을 가진 워크스테이션들로 구성된 NOW에서 이질적 분할 방법이 균등 분할 방법에 비하여 월등히 좋은 성

능을 나타냄을 알았다. 이에 본 절에서는 이질적 분할 방법들에 대한 심층적인 성능 평가를 위해 2가지 실험을 더 수행하였다. 첫째는 병렬 프로그램의 상호 도착 시간 변화에 따른 성능 변화를 살펴보았고, 둘째는 부하 불균형에 따른 성능 변화를 살펴보았다.

먼저 병렬 프로그램의 상호 도착시간을 변화시킴으로써 NOW 시스템에 가해지는 병렬 부하에 영향을 주어 이질적 분할 방법들의 성능이 어떻게 변하는지를 실험해 보았다. 이 실험은 병렬 프로그램의 계산 시간과 통신 시간의 비율이 큰 모델(Coarse-Grain Model, CGM)과 병렬 프로그램의 계산 시간과 통신 시간의 비율이 작은 모델(Fine-Grain Model, FGM)의 형태로 실험을 해 보았다. 병렬 프로그램의 상호도착시간은 평균 100초에서 500초까지 100초 간격으로 변화시켜가며 실험하였으며, 부하 불균형은 없이 실험하였다.

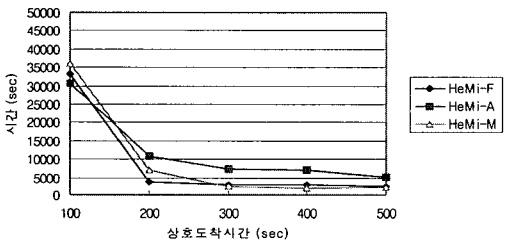

그림 12 상호 도착 시간 변화에 따른 공간 분할 스케줄링 방법들의 응답 시간(CGM)

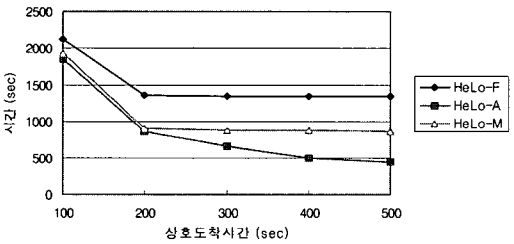

그림 13 상호 도착 시간 변화에 따른 공간 분할 스케줄링 방법들의 응답 시간(CGM)

그림 12와 그림 13에서 상호도착시간이 작아질수록, 모든 스케줄링 방법이 과부하로 인하여 성능이 나빠짐을 알 수 있다. 병렬/순차 부하 충돌 시 이주를 고려한 그림 12의 실험에서는 HeMi-A 방법이 상호도착시간이 커짐에 따라 성능이 나빠지는 것을 알 수 있다. 상호도착시간이 커진다는 것은 NOW 시스템에 가해지는 부하가 줄어드는 것을 의미한다. 따라서 부하가 줄었을 경우 HeMi-A 방법의 경우 각각의 병렬 프로그램이 보다 많은 계산 용량을 할당받게 되어 병렬도가 커지게 된다. 따라서 이주에 의한 느려짐이 커지게 되고 이는 이후 도착되는 병렬 프로그램의 대기 시간을 증가시켜 전체적으로 나쁜 성능을 나타내는 것이다. 그림 12에서 HeMi-F 방법과 HeMi-M 방법을 비교해 보면, 상호도착시간이 작을 때에는, 다시 말해 NOW 시스템에 병렬 부하가 많을 때에는 다음에 도착할 병렬 프로그램을 위해 할당받을 수 있는 계산 용량을 양보하는 HeMi-F 방법이 더 좋은 성능을 나타냈다. 하지만 상호도착시간이 커졌을 때, 즉 NOW 시스템에 부하가 줄었을 때에는 좀더 많은 계산 용량을 사용할 수 있는 HeMi-M 방법이 좋은 성능을 보였다. 이때 HeMi-A 방법이 HeMi-M 방법이나 HeMi-F의 방법보다 나쁜 성능을 나타낸 것은 이주에 의한 느려짐 때문이다. 그림 13의 실험에서는 우선 순위 낮춤을 고려하여 이주에 의한 느려짐이 없어졌기 때문에 HeMi-A 방법이 가장 좋은 성능을 나타내고 있음을 보여주고 있다. NOW 시스템에 병렬 부하가 줄수록 그 성능 차는 더 벌어지고 있다.

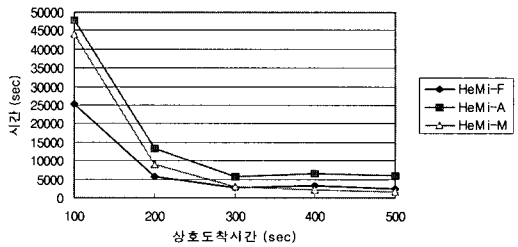

그림 14 상호 도착 시간 변화에 따른 공간 분할 케줄링 방법들의 응답 시간(FGM)

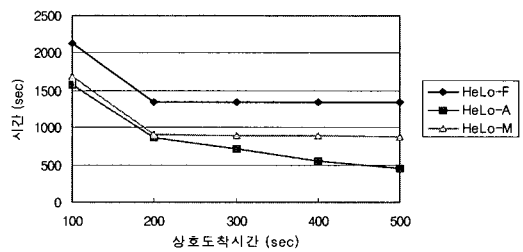

그림 15 상호 도착 시간 변화에 따른 공간 분할 스케줄링 방법들의 응답 시간(FGM)

그림 14와 그림 15는 FGM의 경우를 실험한 결과이다. 이 결과 역시 그림 12와 그림 13의 CGM 경우와 거의 같은 결과를 얻었다. 단 HeMi-A 방법과 HeMi-M 방법의 경우 병렬 프로그램의 상호 도착 시간이 100초일 때 CGM의 경우에 비하여 그 성능이 월등히 나빠졌음을 알 수 있다. 이는 병렬 프로그램의 단계(Phase)가 많아지게 되어 이주에 따른 동기화 문제 때문에 수행 시간이 느려지게 되고, 병렬 부하가 크기 때문에 이후 도착되는 많은 병렬 프로그램의 대기 시간을 증가시켜 나쁜 성능을 나타내게 된 것이다.

다음은 부하 불균형 값을 변화시켜가며 어떤 스케줄링 방법이 부하 불균형에 잘 견디는지를 비교해 보았다. 이때 부하 불균형 값은 0에서 2.0까지 0.5의 간격으로 실험하였다. 병렬 프로그램의 상호 도착 시간은 200초로 하였다. 이는 각각의 이질적 스케줄링 방법이 병렬 프로그램의 상호 도착 시간이 200초일 때 비슷한 성능을 나타내어서 이 값이 부하 불균형에 따른 성능 변화를 관찰하기 좋다고 판단했기 때문이다. 병렬 프로그램 모델은 CGM과 FGM모두를 고려하였다.

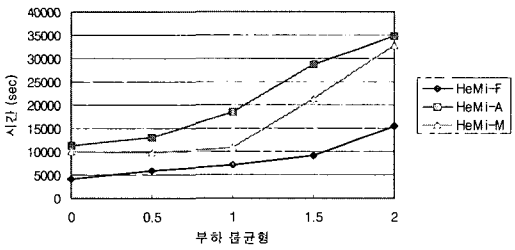

그림 16 부하 불균형 변화에 따른 공간 분할 스케줄링 방법들의 응답 시간(CGM)

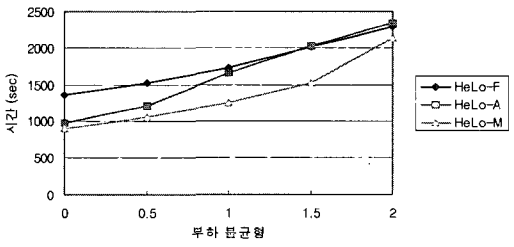

그림 17 부하 불균형 변화에 따른 공간 분할 스케줄링 방법들의 응답 시간(CGM)

그림 16과 그림 17은 CGM의 경우 각각 이주를 고려한 경우와 우선 순위 낮춤을 고려한 경우의 결과이다. 그림 16에서 부하 불균형 값이 1이상이 되면 HeMi-A 방법과 HeMi-M 방법은 HeMi-F 방법에 비하여 기울기가 좀더 급하게 올라가는 것을 볼 수 있다. 이는 HeMi-A 방법과 HeMi-M 방법보다 HeMi-F 방법이 부하 불균형에 잘 견딘다는 것을 의미한다. 이는 HeMi-F 방법에서는 하나의 병렬 프로그램이 HeMi-A 방법과 HeMi-M 방법에 비하여 적은 계산 용량을 할당받아 병렬도가 상대적으로 작기 때문에 병렬 프로세스들간 동기화에 따른 불균형이 적게 나타나게 된 것이다. 그림 17의 경우에서는 HeLo-M 방법이 HeLo-A 방법보다 부하 불균형에 잘 견딤을 보여주고 있다. 그림 13에서 상호 도착 시간을 변화시켜가며 성능을 관찰한 실험에서 상호 도착 시간이 200초일 때 HeLo-A 방법과 HeLo-M 방법이 엇비슷한 성능을 나타내었다. 이번 실험에서도 부하 불균형이 없을 때는 두 방법이 엇비슷한 성능을 나타내었다. 그림 13에서의 수치와 다소 차이가 있는 것은 두 실험에서 사용한 순차부하와 병렬부하가 같지 않았기 때문이다. 하지만 부하 불균형이 1이상이 되면서 HeLo-A 방법은 HeLo-M 방법보다 나쁜 성능을 나타내었다. 결국 하나의 병렬 프로그램이 할당받을 수 있는 계산 용량이 HeLo-A 방법보다 상대적으로 작은 HeLo-M 방법이 부하 불균형에 잘 견디는 것이다.

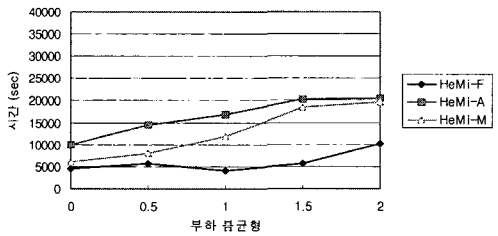

그림 18 부하 불균형 변화에 따른 공간 분할 스케줄링 방법들의 응답 시간(FGM)

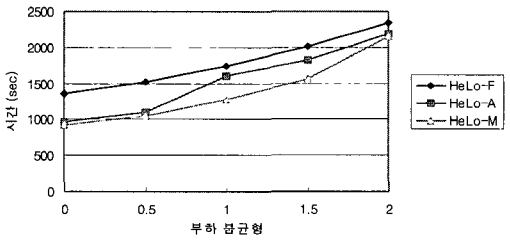

그림 19 부하 불균형 변화에 따른 공간 분할 스케줄링 방법들의 응답 시간(FGM)

그림 18과 그림 19는 FGM의 경우를 실험한 결과이다. 이 결과 역시 각각의 스케쥴링 방법들의 성능 비교에서는 그림 16과 그림 17의 CGM 경우와 거의 같은 결과를 얻었다. 한가지 주목할만한 점은 이주를 고려한 경우 CGM과 FGM을 비교해 보았을 때 부하 불균형이 1 이상이 되면 CGM의 경우가 FGM의 경우보다 월등히 나쁜 성능을 나타내고 있다는 것이다. 이는 병렬 프로그램의 한 단계(Phase) 내에서 계산 시간이 FGM에 비하여 CGM이 상대적으로 월등히 크기 때문에, CGM의 경우에서는 부하 불균형에 따른 한 단계 내에서의 BARRIER 기간이 커지게 되는 것이다. 결국 부하 불균형이 CGM의 경우 FGM에 비하여 전체적인 스케쥴링 성능에 많은 영향을 미치게 되는 것이다.

지금까지 본 절에서 실험한 2가지 실험 결과를 종합적으로 분석해보면 다음과 같다. 병렬 프로그램의 상호 도착 시간을 변화시켰을 때, HeLo-A 방법이 가장 좋은 성능을 나타내었다. 하지만 병렬 부하가 클 경우에는 HeLo-M 방법도 HeLo-A 방법과 비슷한 성능을 나타내었다. 그리고 부하 불균형이 발생하게 되었을 때는 HeLo-M 방법이 HeLo-A 방법보다는 보다는 부하 불균형에 잘 견디었다.

4.4 실제 구현을 통한 성능 평가 결과

시뮬레이션 결과를 실제적으로 검증하기 위하여 4대의 Linux PC를 10Mbps Ethernet으로 연결하고 이 위에 MPI를 설치하여 간단한 NOW환경을 만든 후 2차원 열 전도 문제를 수행하였다. 4대의 PC는 333Mhz Pentium CPU와 64Mbyte 메모리를 가지는 2대의 PC와 150Mhz Pentium CPU와 16Mbyte 메모리를 가지는 2대의 PC로 구성되었다. 333Mhz PC와 150Mhz PC는 각각 383MFLOPS와 170MFLOPS의 성능을 가짐이 실험에 의하여 밝혀졌다. 이러한 환경에서 우선 순위를 낮춤을 고려한 균등 분할 및 균등 할당 방법(HoHoLo), 우선 순위 낮춤을 고려한 이질적 분할 및 고정 할당 방법(HeLo-F), 우선 순위를 고려한 이질적 분할 및 적응 할당 방법(HeLo-A)의 세 스케쥴링 정책의 성능을 비교하였다. 입력으로 2차원 열 전도 문제의 병렬 프로그램을 세 개 수행하였다. 먼저 한 개의 프로그램 P1이 제출 즉시 수행되고 P1이 제출된 후 120초 후 그러나 P1이 끝나기 전에 두 개의 프로그램 P2와 P3가 동시에 제출되도록 하였다. HoHoLo의 경우 프로그램을 두 개의 병렬 프로세스로 균등 분할 한 경우 평균 응답시간이 496초가 되었다. 4개의 PC를 동일한 계산 용량을 가지는 두 개의 파티션으로 분할한 HeLo-F와 HeLo-A의 경우는 응답 시간이 각각 320초와 293초로 HeLo-A가 가장 좋은 성능을 보임을 알 수 있었다. 물론 HeLo-A의 경우에는 P2와 P3가 언제 제출되었느냐에 따라 응답 시간이 길어질 수도 있다. 그러나 HeLo-A의 경우 세 개의 프로그램을 모두 끝내는 시간이 480초로서 HoHoLo의 896초, HeLo-F의 640초보다 훨씬 짧았다.

5. 결 론

본 논문에서는 이질적 계산 능력을 가지고 있는 워크스테이션들로 구성된 NOW에서 다양한 스케쥴링 방법이 병렬 작업의 성능에 어떻게 영향을 주는지에 대하여 연구하였다. 본 논문에서는 공간 분할 기법에 의거한 10가지의 스케쥴링 방법을 제시하였으며 시뮬레이션을 통하여 위 스케쥴링 방법들의 성능을 비교한 결과 다음과 같은 결론을 얻었다. 워크스테이션의 계산 능력에 비례하여 병렬 프로그램을 이질적인 크기의 병렬 프로세스들로 분할하는 경우가 균등 분할하는 경우보다 성능이 우수함을 알 수 있었다. 병렬 프로세스를 수행하는 워크스테이션에 소유자가 돌아온 경우 병렬 프로세스를 새 유휴 워크스테이션에 이주하는 것보다 병렬 프로세스의 우선 순위를 낮추는 두 방법을 고려하였는데 이주를 하는 경우 이주를 위한 유휴 워크스테이션을 약 3% 정도 확보하면 그렇지 않은 경우보다 최고 2.5배의 성능 향상이 있었음을 알 수 있었고 우선 순위 낮춤을 사용하는 경우 우선 순위 낮춤으로 인하여 발생하는 병렬 프로세스의 느려짐이 스케쥴링의 성능에 크게 영향을 주지 않음을 알 수 있었다. 그러나 일반적으로 이주를 사용한 이질적 분할 방법보다 우선 순위 낮춤을 사용한 이질적 분할 방법의 성능이 약 5배 이상 우수함을 알 수 있었다. 우선 순위 낮춤을 사용한 이질적 분할 방법에서는 부하 불균형이 없을 때 적응 방법이 광범위한 병렬 프로그램 도착 시간의 범위에서 가장 좋은 성능을 보인다. 그러나 적응 방법과 수정된 적응 방법의 성능이 비슷한 병렬 프로그램의 도착 시간의 경우에는 수정된 적응 방법이 적응 방법보다 부하 불균형에 잘 견딤을 알 수 있다.

또 작은 규모의 NOW 환경을 구현하여 제안된 스케쥴링 기법의 일부를 실험한 결과 시뮬레이션과 동일한 결과를 얻었다. 그러나 앞으로 프로세스 이주 기능을 구현하고 MPI가 설치된 하나의 PC에서 동시에 여러 개의 병렬 프로세스를 수행할 수 있는 기능을 추가로 구현하고 또 많은 수의 병렬 프로그램을 수집하여 다양한 환경을 실제로 구현하여 성능을 비교하여야 할 것이다.

본 논문에서는 이전 연구에서 공간 분할 정책이 시간

분할 정책보다 우수함이 밝혀졌기 때문에 공간 분할 스케쥴링 방법에 대해서만 고려하였다. 하지만 향후에는 공간 분할 정책과 시간 분할 정책이 혼합된 혼성(hybrid) 정책과 공간 분할 정책을 비교해 봐야 할 것이다. 또한 본 논문에서의 모델링한 병렬 프로그램의 통신 모델이 너무 단순했기 때문에 이를 보다 세밀하게 모델링하여 실험을 해 봐야 할 것이다.

참 고 문 헌

[1] T. E. Anderson et al., "A Case for NOW(Network of Workstation)," IEEE Micro, vol. 15, no. 1, pp. 54-64, February 1995.

[2] M. M. Mutka and M. Livny, "The available capacity of a privately owned workstation environment," Performance Evaluation, vol. 12, no. 4, pp. 269-284, July 1991.

[3] J. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey, and P. Stephan, "Dome: Parallel Programming in a Heterogeneous Multi-User Environment," Proceedings of the International Parallel Processing Symposium, 1996

[4] S.L. Au and S.P. Dandamudi, "The Impact of Program Structure on the Performance of Scheduling Policies in Multiprocessor Systems," Int. J. Computers and Their Applications, Vol. 3, No. 1, April 1996

[5] Y.-N. Chan et al., "Experiences with Parallel Job Scheduling on a Transputer System," Performance Evaluation Review, March 1999

[6] Young-Min Chang and Young-Chul Shim, "Simulation- ased Analysis of the Performance of Parallel Programs in the NOW," IASTED Int. Conf. on Parallel and Distributed Computer and Networks, 1998.

[7] M. Mutka and M. Livny, "The Available Capacity of a Privately Owned Workstation Environment," Performance Evaluation, July 1991

[8] Anurag Acharya, et al., "The Utility of Exploiting Idle Workstations for Parallel Computation," Proceedings of ACM SIGMETRICS '97, 1997.

[9] D. G. Feitelson, "Packing Schemes for Gang Scheduling," In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1996

[10] Y.N. Chan, S.P. Dandamudi, and S. Majumdar, "Performance Comparison of Processor Scheduling Strategies in a Distributed-Memory Multicomputer Systems," IEEE Int. Parallel Processing Symposium, 1997

[11] Y.N. Chan, S.P. Dandamudi, and S. Majumdar, "Experiences with Parallel Job Scheduling on Transputer System," ACM SIGMETRICS Performance Evaluation Review, Vol. 26, No. 4, 1999

[12] C. McCann and J. Zahorjan, "Processor Allocation Policies for Message-Passing Parallel Computers," Proc. ACM SIGMETRICS Conference, May 1994

[13] M.S. Squillante, "On the Benefits and Limitations of Dynamic Partitioning in Parallel Computer Systems," Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Springer-Verlag, 1995

[14] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," Job Scheduling Strategies for Parallel Processing, D. Feitelson, L. Rudolph (eds.), Vol. 1291, Lecture Notes in Computer Science, Springer-Verlag, 1997

[15] E.W. Parsons and K.C. Sevcik, "Coordinated Allocation of Memory and Processors in Multiprocessor," Proc. ACM SIGMETRICS Conference, 1996

[16] Y. Yan, X. Zhang, and Y. Song, "An Effective and Practical Performance Prediction Model for Parallel Computing on Non-Dedicated Heterogeneous NOW," Journal of Parallel and Distributed Computing, Vol. 38, 1996

[17] J.B. Weissman and X. Zhao, "Run-time Support for Scheduling Parallel Applications in Heterogeneous NOWs," IEEE Symp. on High Performance Distributed Computing, 1997

[18] A. Downey and D. Feitelson, "Elusive Goal of Workload Characterization," Performance Evaluation Review, March 1999

[19] A.C. Dusseau et al, "Effective Distributed Scheduling of Parallel Workloads," ACM Int. Conf on Measurement and Modeling of Computer Systems, 1996.

[20] L. W. Dowdy and M. R. Leuze, "On Modeling Partitioned Multiprocessor Systems," International Journal of High Speed Computing, Vol.6, No.1, 1994

[21] A.M. Vahdat, D.P. Ghormley, and T.E. Anderson, "Efficient, Portable, and Robust Extension of Operating System Functionality," Technical Report UCB//CSD- 4-842, University of California, Berkeley, 1994.

[22] M. Harchol-Balter and A.B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," Proceedings of ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, May 1996.

김 진 성

1994년 ~ 1998년 홍익대학교 컴퓨터공학과 학사. 1998년 ~ 2000년 홍익대학교 정보공학과 석사. 2000년 ~ 현재 한국통신하이텔 연구소 연구원. 관심분야는 분산병렬시스템, 시스템 네트워크 보안, 운영체제.

심 영 철

1975년 ~ 1979년 서울대학교 전자공학과 학사. 1979년 ~ 1981년 한국과학기술원 전기 및 전자공학과 석사. 1981년 ~ 1984년 삼성전자 대리. 1984년 ~ 1991년 UC Berkeley 전산학 박사. 1991년 ~ 1993년 UC Berkeley 연구원, 1993년 ~ 현재 홍익대학교 정보컴퓨터공학과 부교수. 관심분야는 분산병렬시스템, 컴퓨터 및 네트워크 보안, 인터넷 프로토콜