그리드 컴퓨팅의 다중 큐 하이브리드 작업스케줄링 기법

(Multi-queue Hybrid Job Scheduling Mechanism in Grid Computing)

강 창 훈 † 최 창 열 †† 박 기 진 ††† 김 성 수 ††††
(Changhoon Kang) (Changyeol Choi) (Kiejin Park) (Sungsoo Kim)

요 약 그리드(Grid) 컴퓨팅은 지리적으로 분산된 컴퓨팅 자원들을 네트워크로 연동시켜 서로 공유될 수 있도록 해주는 서비스이다. 본 논문에서는 그리드 컴퓨팅 시스템을 구성하는 전체 노드를 대상으로 작업을 분배하는 메타 스케줄링 정책과 특정 한 개의 노드 내에서 작업을 분배하는 작업 스케줄링을 동시에 고려하는 하이브리드 스케줄링 기법을 제안한다. 그리드 컴퓨팅 노드로 제출된 작업을 필요 프로세서 수와 예상 작업수행 시간에 따라 구분하여 우선순위가 높은 작업은 작업 큐(Job Queue)로, 우선순위가 낮은 작업과 원거리 작업은 백필 큐(Backfill Queue)로 할당시킴으로써, 그리드 컴퓨팅 시스템의 성능을 높이는 방법을 연구한다. 다양한 실험을 통하여 제안된 기법들의 성능을 평가하며, 그리드 컴퓨팅 시스템의 이용률이 높아지고, 작업 지연시간이 감소됨을 확인한다.

키워드 : 그리드 컴퓨팅, 메타 스케줄링, 백필 스케줄링, 예약

Abstract Grid computing is a service that share geographically distributed computing resources through high speed network. In this paper, we propose hybrid scheduling scheme which considers not only meta-scheduling scheme to distribute the job between the nodes of grid computing system but also the job scheduling to distribute the job within the local nodes. According to the number of processors needed and expected execution time, the job with high priority is allocated to job queue while the one with low priority and remote job are allocated to backfill queue. We evaluate the proposing scheme through the various experiments and the results show that the utilization of grid computing system increases and the job slowdown decreases.

Key words : Grid Computing, Meta-scheduling, Backfill Scheduling, Reservation

1. 서 론

최근 과학 기술이 발전함에 따라 복잡한 문제를 해결하기 위하여 고성능의 계산 자원이 필요하게 되었으며, 이러한 요구를 충족시키기 위하여 지역적으로 분산되어 있는 이질적인(Heterogeneous) 고성능 컴퓨팅 자원을 하나로 묶어 거대한 시스템을 구성하는 그리드 컴퓨팅(Grid Computing)에 대한 연구가 활발하게 이루어지고 있다. 그리드 컴퓨팅은 지리적으로 분산되어 있는 고성능 컴퓨터, 대용량 저장장치, 첨단과학 설비 등 컴퓨팅 자원을 네트워크로 연동하여 병렬 분산처리 환경을 제공하는 방법이다[1]. 현재 그리드 컴퓨팅 기술은 병렬 컴퓨팅 기술의 새로운 패러다임으로 평가를 받고 있으며, 크게 1)그리드로 연결된 유휴 자원들을 찾아내는 자원 검색 서비스, 2)할당된 작업들의 처리 순서를 결정하여 분산시키는 스케줄링 서비스, 3)시스템 안정을 위한 그리드 보안 서비스, 4)컴퓨팅 자원들을 사용할 때 발생하는 비용 처리를 위한 사용자 계정 서비스 등을 지원하기 위한 그리드 기반 기술 들을 개발 중이다[2]. 또한 그리드 컴퓨팅을 지원하기 위한 Globus[3]와 Legion[4] 등의 미들웨어 개발에 대한 연구가 활발히 진행되어, 그리드 응용 프로그램 개발이 용이해 졌다. 향후에는 이러한 미들웨어를 사용한 다양한 그리드 컴퓨팅 응용 서비

· 본 연구는 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행되었음. (KRF-2006-331-D00438)
† 정 회 원 : 극동정보대학 방송영상미디어과 교수
chkang@kdc.ac.kr
†† 정 회 원 : 숭실대학교 정보미디어기술연구소 전인연구원
cychoi@otlab.ssu.ac.kr
††† 종신회원 : 아주대학교 공과대학 산업정보시스템공학부 교수
kiejin@ajou.ac.kr
†††† 종신회원 : 아주대학교 정보통신전문대학원 교수
sskim@ajou.ac.kr
논문접수 : 2006년 4월 24일
심사완료 : 2007년 4월 27일

스가 개발될 것으로 예상된다.

그리드 컴퓨팅 환경에서의 작업 스케줄링은 기존의 병렬 컴퓨팅 환경에서의 병렬 작업 스케줄링 기법을 그리드 컴퓨팅 환경에 그대로 적용시키는 방법이 시도되고 있다[5]. 병렬 컴퓨팅의 스케줄링 기능은 크게 자원들의 분산 처리에 관한 관리와 작업 스케줄링 기법으로 구분할 수 있다[6]. 대표적인 예로, 이기종 병렬 컴퓨팅 환경에 적합한 스케줄링 기법인 백필 방식을 그리드 컴퓨팅에 적용하여 보다 빠른 응답 시간을 제공하기 위한 스케줄링 기법들이 소개되었다[7]. 하지만, 기존 백필을 기반의 스케줄링 기법은 작업의 특성과 우선순위에 따라 작업 분배 결과가 달라지기 때문에 성능 요구조건 사이에 상충관계(Trade-Off)가 존재한다. 즉, 시스템의 자원 이용률(Utilization) 높이고자 작업의 우선순위를 정하면 그로 인해 작업의 지연시간(Slowdown)이 늘어나고, 작업의 지연 시간을 줄이고자 작업을 분류하면 시스템 자원 이용률이 낮아지는 결과를 초래한다.

작업의 특성을 고려하여 우선순위를 조정함으로써 전체 시스템의 자원 이용률을 증가시키거나 작업의 지연시간을 단축 시킬 수 있지만, 이러한 스케줄링 문제는 NP-Complete로 해결하기 어렵다[8]. 현재까지 여러 가지 병렬 컴퓨팅 스케줄링 알고리즘이 연구되어 왔으나, 대부분 작업 계산량을 중심으로 다루어 지역적으로 분산된 그리드 시스템에 적합하지 않은 문제점을 가지고 있다.

한편 그리드를 구성하는 각 노드에 작업들을 분배하는 메타 스케줄링 방법들이 많이 연구되고 있지만, 작업들을 그리드 시스템의 임의의 노드에 분배 한 후에 각 노드 내에서 작업 스케줄링을 병행 처리하는 방식은 거의 연구되지 않고 있는 실정이다. 따라서 전체 시스템의 성능 향상을 위해서는 노드 간 성능 향상을 위한 메타 스케줄링과 노드 내의 작업 스케줄링을 동시에 고려한 통합 스케줄링 방법이 필요하다.

이에 따라 본 논문에서는 그리드 컴퓨팅 시스템 상의 임의의 노드로 제출된 작업을 실행 시키는데 필요한 프로세서 수와 예상 작업수행 시간에 따라 작업들을 구분하여, 우선순위가 높은 작업은 예약 후에 실행되는 작업 큐(Job Queue)로 분배하고, 우선순위가 낮은 작업과 다른 노드로부터 발송 (Dispatch) 되어진 원거리(Remote) 작업은 백필 큐(Backfill Queue)로 할당시킴으로써, 시스템의 이용률을 높이면서, 동시에 작업 지연시간을 단축시키는 방법을 연구하며, 제안된 기법들을 다양한 시뮬레이션 실험을 통하여 성능을 평가한다. 본 논문의 2장에서는 연구 배경을 언급하고, 3장에서는 하이브리드 스케줄링 구조를 제시하고, 4장에서는 백필 기반의 다중 큐 스케줄링 기법을 제시하며, 5장에서는 제안한 방법의 성능 평가를 수행하고, 6장에는 결론을 내린다.

2. 연구배경

본 논문은 그리드 시스템에서 De-centralized 스케줄링 기법을 기본으로 하는 그리드 작업스케줄링 기법의 연구로, 기존의 스케줄링 기법들은 그리드를 구성하는 노드로만 작업을 분배하는 메타스케줄링 정책에만 초점을 맞추었다. 그러나 이런 정책은 노드 내의 작업과 다른 노드에서 보내진 원거리 작업간에 스케줄링에 관해서는 고려하지 않았기 때문에, 해당 지역 노드의 작업과 원거리 노드의 우선순위를 고려한 효율적인 스케줄링이 이루어질 수 없다. 따라서 본 논문에서는 그리드 노드 내의 작업 스케줄링과 병행한 그리드 메타 스케줄링 기법을 연구한다.

그리드 노드 내의 작업 스케줄링 방법은 기존의 병렬 시스템의 병렬 작업스케줄링 기법 중 가장 많이 사용하는 백필 스케줄링을 변형한 기법을 사용한다. 백필 스케줄링은 Conservative 백필과 EASY 백필 스케줄링으로 구분한다. 두 방법을 변형한 많은 연구들이 이루어졌다. 두 방법은 상충관계에 잇는 방법으로, Conservative 백필은 시스템 이용률이 낮은 반면 작업시간 지연률은 낮은 방법고 이에 반해 EASY 백필은 시스템 이용률은 높은 반면에 작업시간 지연률이 높아지며 작업의 무한대기가 발생할 수 있다.

따라서 본 논문에서는 상기 두 방법의 단점을 보완하여 시스템 이용률을 높이고 무한대기를 방지한 새로운 스케줄링 방법을 그리드 시스템의 메타스케줄링 방법에 적용한 작업 스케줄링 정책을 연구한다. 본 장에서는 그리드 시스템의 메타스케줄링 방법과 병렬 스케줄링 방법의 관련연구를 기술하고 연구의 방향을 제시한다.

2.1 그리드 시스템의 작업 스케줄링

그리드 작업 스케줄링은 수행될 작업을 처리하는 관점에 따라 크게 두 가지로 나눌 수 있다. 첫째는, 수행해야 할 작업을 적절한 그리드 구성 노드로 분배하는 메타 스케줄링 과정이고, 둘째는 그리드 구성 각 노드 내에서 적절한 스케줄링 정책에 의해 작업을 분배하는 과정으로 나눌 수 있다. 현재까지 대부분의 연구는 이 두 과정을 서로 다른 관점에서 다루었으나, [5]에서는 다중 큐를 이용하여 로컬(Local) 작업과 원거리 작업을 서로 다른 큐에 배치/예약하는 방법을 연구하였다.

그림 1은 Centralized 스케줄링에 대한 설명으로 그리드 상의 모든 노드의 상태 정보를 관리하며 제어하는 메타 스케줄러가 있고, 그리드의 모든 작업은 메타 스케줄러에게 제출되고, 제출된 작업이 바로 수행되지 못할 경우 중앙 작업 큐(Central Job Queue)에 저장되어 대기하며, 이후에는 메타 스케줄러가 적절한 노드에 분배

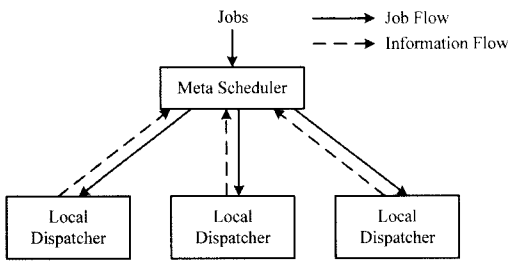

그림 1 Centralized 스케줄링

하고 지역 노드는 스케줄링 결정에 아무런 영향을 미치지 않는 방법이다. 이 방법은 메타 스케줄러가 가용 가능한 리소스에 대한 모든 정보를 알고 있기 때문에 개념적으로 매우 효과적인 스케줄링이 가능한 장점이 있다. 그러나 이 방법은 메타 스케줄러가 모든 정보를 관리하며 작업을 분배하기 때문에 Scalable 하지 못하고, 네트워크 장애(Network Failure)로 인해 메타 스케줄러와 지역 노드와 차단될 경우 병목(Bottleneck) 현상이 발생하는 단점이 존재한다[9].

그림 2는 De-centralized 스케줄링 정책으로 하나의 메타 스케줄러만 존재하는 것이 아니라 모든 노드에 메타 스케줄러가 존재하고, 모든 작업은 그 작업이 생성된 노드의 메타 스케줄러에 제출된다. 메타 스케줄러는 서로 서로 질의(Query)를 주고받으며 일정 기간 동안 부하 정보를 모집한다. 그 결과를 작업 부하가 가장 적은 노드로 작업을 발송한다. 이 방법은 Scalable하고, 병목 현상이 발생하지 않으며, 한 노드에서 장애가 발생하더라도 전체 그리드 시스템에는 영향을 미치지 않는다. 또한 지역 노드의 독자적인 지역 스케줄링 정책이 가능한 장점이 있다. 그러나 구현하기 어렵고 동기화하기 어렵기 때문에 최적의 스케줄링을 제공하기 어려운 단점을 가지고 있다[9].

현재 대부분의 그리드 작업 스케줄링 연구들은 De-centralized 스케줄링에 의해 로컬 노드로 제출된 작업을 해당 노드에서 실행하지 못할 경우 다른 노드에 적절하게 분배하는 스케줄링 방식에 대한 연구가 주를 이루고 있다[10-12].

작업의 백필율을 높여 시스템의 이용률을 높이기 위한 방법으로, 작업을 복사하여 여러 개의 노드에 분배하여 실행하는 방법이 연구되었다[10]. 작업이 임의의 노드에 제출되면 지역 메타 스케줄러는 k개의 주변 노드에 해당 작업을 복사하여 분배한다. 작업이 복사된 k개의 노드들은 각각 노드의 지역 스케줄링 정책에 따라 스케줄링한다. 어느 시점에서 작업이 k개의 노드 중 임의의 노드에서 실행 가능할 때, 그 노드의 메타 스케줄러는 k-1개의 메타 스케줄러에게 작업이 실행됨을 통보하고, k-1개의 지역 큐에서 작업을 삭제할 것을 요청한다. 이 방법은 여러 노드에서 작업이 백필 될 기회가 부여되기 때문에 백필이 용이하고, 시스템 이용률이 개선되었으며, 평균 작업 반환(Average Job Turnaround) 시간을 줄였다. 그러나 작업을 여러 노드에 복사하여 스케줄링 함으로써 지역 스케줄러의 작업량이 증가하는 문제가 발생하고, k의 크기에 따라 내부 노드 사이의 통신량과 동기화(Synchronization) 량이 증가하는 단점이 발생한다.

또 다른 De-centralized 메타 스케줄링 방법으로 그리드 전체 노드들을 서브그리드(Subgrid)로 나누고 작업이 끝날 때마다 서브그리드 내에서 예상 응답시간이

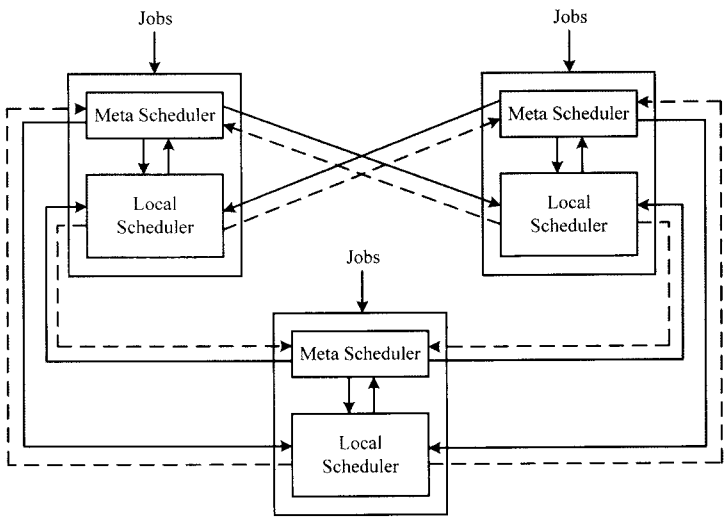

그림 2 De-centralized 스케줄링

가장 작은 노드로 작업을 이동(Migrate)하여 백필하는 분산백필(Distributed Backfill) 기법이 연구되었다[11]. 이 방법은 그리드 상에서 기다리는 작업의 노드 선택을 동적으로 개선하기 위하여 연구된 방법이다. 또한 지역 노드 내에서의 백필은 기존의 Conservative 백필 기법을 사용하는 것을 가정하였고, 단일 노드에서 실행되는 작업이며 동적 스케줄링(Dynamic Scheduling)을 전제로 하였다.

이질 그리드 환경을 위한 De-centralized 스케줄링과 동적 부하 균형(Dynamic Load Balancing) 알고리즘[12]이 제시되었다. 작업이 임의의 노드로 제출되면 작업의 실행 여부를 판단하여 실행 가능할 경우 내부 큐(Internal Queue)에 배치하고, 그렇지 못할 경우 외부 큐(External Queue)에 배치한다. 외부 큐에 배치된 작업을 이웃 노드(Neighbor Node)중에서 부하가 가장 적은 노드를 찾아 발송을 요청 한다. 이웃 노드들의 구성 방법으로 해당 노드와 이웃 노드에 포함시키고자 하는 임의의 노드 사이의 통신 지연(Communication Delay)이 임계치 이내의 값을 갖는 노드들로 구성한다. 이웃 노드 내에서 발송 요청을 받은 노드는 실행 가능할 경우, 실행 가능 응답을 보내고 작업을 실행한다. 이 방법 역시 노드 사이의 작업을 분배하는 메타 스케줄링 정책만을 다루었다.

그리드를 구성하는 노드에 작업을 분배하고 분배된 노드 내에서 작업을 분배하는 두 가지 과정을 고려한 스케줄링 정책의 연구는 미비한 실정이다. 계산 그리드 시스템의 한 노드로 구성될 수 있는 병렬 시스템의 작업 스케줄링 기법으로, 다중 큐를 이용하여 지역 작업과 원거리 작업을 서로 다른 우선순위에 의해 배치하여 그리드 시스템 환경에 적용한 방법이 연구되었다[5].

시스템은 네 개의 파티션으로 나누어지고 각 파티션 당 하나의 큐가 할당된다. 초기에 각 파티션은 임의의 프로세서가 할당되며 시간이 지남에 따라 작업부하의 정도에 따라 각 파티션의 프로세서 수가 동적으로 변화한다. 각 노드의 작업 스케줄링 정책은 EASY 백필 방법을 사용하는 것을 전제로 하였다. 지역 사용자에 의해 제출된 작업은 높은 우선순위를 가지며 외부 노드로부터 발송된 작업은 낮은 우선순위를 갖는다. 작업이 노드로 제출되면 작업의 예상 실행시간에 따라 4개의 큐 중 하나의 큐에 할당되고 우선순위에 따라 높은 우선순위 작업은 큐의 앞쪽에 낮은 우선순위의 작업은 높은 우선순위 작업보다 큐의 뒤쪽에 저장한다.

이 방법은 작업의 수행에 필요한 프로세서 수를 고려하지 않고 예상 실행시간만을 고려하였기 때문에 시스템의 작업 지연시간은 전반적으로 감소하였지만 시스템의 이용률은 고려되지 않았다. 또한 그리드의 외부 노드로부터 임의의 노드로 보내는 정책은 고려하지 않았고, 외부로부터 보내진 작업이 낮은 우선순위에 의해 큐의 뒤쪽에 배치되기 때문에 높은 우선순위의 지역작업에 의해 무한대기가 발생할 수 있는 문제점을 가지고 있다.

2.2 병렬 시스템의 작업 스케줄링

병렬 작업 스케줄링 방법 중 가장 많이 사용하는 가변 분할(Variable Partitioning) 방법은 작업의 수행시간을 가로축, 필요 자원 공간을 세로축의 직사각형 모양으로 표현하여 작업을 할당해주는 방식이다. 이 과정에서 사각형을 제외한 여백 부분이 많아지면 시스템의 자원 이용률이 떨어지게 된다. 또한 가변 분할 방식은 작업이 들어오는 순서에 의한 우선순위를 갖는 방식을 따르기 때문에 시스템의 유휴 자원으로 수행 가능한 경우에도, 우선 순위가 낮으면 수행되지 못하는 문제가 발생한다. 이 경우 작업은 수행되지 않고 자원만 남는 단편화(Fragmentation) 문제가 발생하여 시스템 성능이 저하된다. 단편화 문제를 해결하기 위해 Dynamic Partitioning[15]이나 Gang 스케줄링[16] 방법이 제시 되었는데, 각 노드의 클럭(Clock) 동기화 문제나 통신상의 오버헤드 등 구현 시에 발생하는 제약 점이 많아 거의 사용되지 않는다.

단편화 문제를 해결하기 위한 또 다른 방법으로 여백의 공간에 작업 대기 큐에 있는 작업 중 우선순위를 변경하여 먼저 수행 될 수 있도록 끼워 넣는 개념인 백필 방법이 제시되었으며, 이 기법은 현재 유휴 자원으로 수행이 가능하지만 우선순위가 낮아 수행되지 못하는 작업을 자신보다 우선순위가 높은 작업을 지연시키지 않는 경우 먼저 수행시키는 스케줄링 방법이다[7].

백필 스케줄링 기법 중 Conservative 백필은 현재 큐의 뒤쪽에 있는 우선순위가 낮은 작업이 그 작업보다 먼저 수행되어야 하는 모든 작업을 지연시키지 않는 경우에만 백필하여 작업을 수행시키는 방법이다. 그림 3은 Conservative 백필 과정의 예로 스케줄링을 시작할 때 현재 시스템에서 수행중인 작업을 끝나는 시간을 기준으로 하여 가장 늦게 끝나는 작업부터 정렬하여 일정한 시간 간격으로 사용 가능한 자원을 확인한다. 그 후 첫 번째 작업이 사용 가능한 시간(그림 3의 Anchor Point)을 확인하고, 두 번째 작업이 그 전에 수행 가능한지 확인한다. 두 번째 작업은 자원이 충분하지 않고 수행 가능 하여도 첫 번째 작업을 지연시키기 때문에 첫 번째 작업 뒤에 예약된다. 세 번째 작업은 백필이 가능함으로 수행된다. Conservative 백필 방식은 우선순위가 낮은 작업을 백필하여 수행하고자 할 때 자신보다 더 앞에 도착한 어떤 작업도 지연시키지 않으면서 수행되는 방식이다.

EASY 백필은 우선순위가 낮은 큐의 뒤쪽에 있는 작

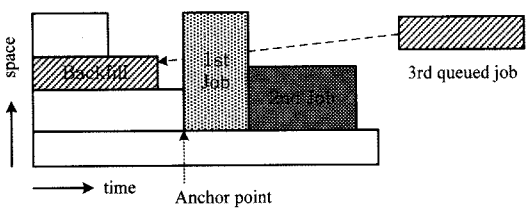

그림 3 Conservative 백필 스케줄링

업들이 큐의 맨 처음에 있는 작업을 지연시키지만 않으면 백필하여 수행시키는 방법이다. 그림 4는 EASY 스케줄링 방식을 설명한 것으로, 사용 가능한 자원과 큐의 첫 번째 작업이 처음으로 수행 가능한 시간(그림 4의 Shadow Time)을 확인한다. 그런 후에 큐에 있는 작업들을 차례대로 확인하면서 스케줄링 작업을 하게 된다. 그림 처럼 현재 사용 가능한 프로세서보다 적은 프로세서를 요구하면서 Shadow time 전에 수행이 끝나는 작업이거나 혹은 Shadow time 전에 수행이 끝나지는 않지만 첫 번째 작업이 수행중인 경우에 남는 프로세서보다 더 적은 프로세서를 요구하는 작업의 경우에는 먼저 수행한다. 이와 같이 EASY 스케줄링 방식은 n번째 작업이 큐에 있는 첫 번째 작업만 지연시키지 않는다면 백필하여 바로 실행시키는 방법이다.

앞에서 살펴본 두 가지 백필 방법은 각각의 장단점이 존재하는데, EASY 백필 방식을 사용하면 Conservative 백필 방식보다 더 많은 작업을 백필하여 실행시킬 수 있고, 이것은 전체 시스템의 효율을 높여준다. 그러나 Conservative 백필에서처럼 작업의 실행을 약속할 수 없고, 두 번째 이후의 작업이 오랫동안 대기하는 무한대기(Unbounded Delay)가 발생하게 되고 이러한 점은 시스템의 성능 저하를 초래하게 된다.

백필 스케줄링을 기반으로 하여 시스템의 성능을 높이고자 여러 방법이 연구되었는데, 작업의 수행 시간이 짧은 작업부터 스케줄링하는 방식이 제안되었는데, 수행 시간이 짧은 작업들의 우선순위가 높아지면, 짧은 작업들이 먼저 수행되게 되어 전체적으로 작업들이 오래 기다리지 않아도 되어 긴 작업의 지연 시간을 줄일 수 있다. 그러나 이런 방법은 백필 가능성이 높은 짧은 작업을 먼저 실행하게 되어 백필 확률이 낮아져서 시스템의 자원 효율이 낮아진다[17]. 또 다른 방법으로 작업이 제출되어 현재까지 수행되지 못하고 기다린 시간이 길수록 우선순위를 높게 하여 작업이 오래 기다리는 경우 일찍 수행되도록 하여 작업의 지연시간을 줄인 방법도 연구되었다[18]. 또한 사용자가 결정한 우선순위를 기준으로 우선순위를 결정한 후에, 기다린 시간을 기준으로 Slack을 부여하여, 스케줄링 시에 작업이 갖고 있는 슬랙(Slack) 만큼 우선순위가 높은 작업을 지연시키는 것을 허용하여, 사용자가 결정한 우선순위를 반영하여 전체적인 작업지연 시간을 줄이는 방법도 연구 되었다[19]. 시스템 자원 효율을 높이기 위한 방법으로 Relaxed 백필 방법은 작업의 우선순위를 조절하여, 현재 백필시켜 수행하려는 작업이 우선순위가 높은 작업을 지연시키더라도 어느 정도의 지연을 허용하는 개념으로 좀 더 많은 백필을 수행함으로써 시스템의 자원 효율을 높이는 기법이다[20].

이외에도 작업의 우선순위를 적절히 조절하여 스케줄링되는 형태를 바꿈으로써 시스템의 효율을 높이고 지연시간을 낮추는 방법이 제시되었다. 그러나 기존의 많은 연구들은 시스템의 자원 이용률과 작업의 지연 시간 사이의 상충관계를 고려하지 않고 한쪽에만 초점을 맞추었기 때문에 시스템의 자원 이용률을 높이면 작업의 지연 시간이 늘어나고 작업의 지연 시간을 줄이면 시스템의 자원 이용률이 낮아진다는 단점을 공통적으로 가지고 있다.

2.3 연구방향

본 논문에서는 그리드 컴퓨팅 시스템의 노드 간 작업을 분배하는 메타 스케줄링 정책과 지역 노드 내의 작업을 분배하는 작업 스케줄링을 동시에 고려하는 하이브리드 작업 스케줄링 기법을 제안한다. 즉, 1)노드 내의 스케줄링을 고려한 노드 간의 메타 스케줄링 정책과 2)노드 내에서 그리드 시스템의 이용률을 높이고 작업

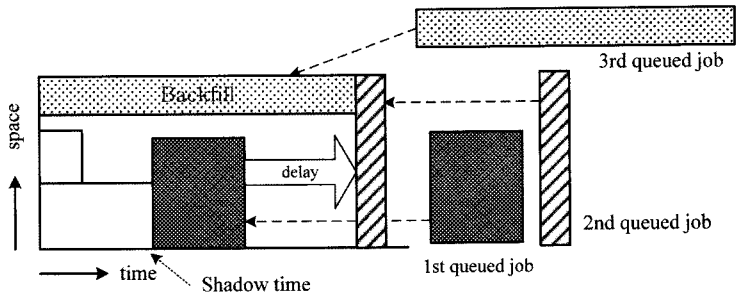

그림 4 EASY 백필 스케줄링

지연 시간을 줄이는 스케줄링 정책을 동시에 고려하여 연구를 진행한다. 제안한 그리드 컴퓨팅 시스템의 하이브리드 작업 스케줄링 정책은 고속으로 연결된 계산 그리드 시스템에서 그리드 시스템 전체의 성능을 높이기 위한 방법으로, 노드 사이의 작업 이동에 의하여 노드에서 부하를 다른 노드로 분배하며, 노드 내의 다중 큐 사용으로 많은 백필이 가능해짐에 따라, 시스템 자원 이용률을 높이면서 작업 지연 시간을 줄여 응답시간을 감소시키는 새로운 작업 스케줄링 기법이다.

그리드 컴퓨팅 노드로 제출된 작업을 필요 프로세서 수와 예상 작업수행 시간에 따라 구분하여, 우선순위가 높은 작업은 작업 큐로 분배하고 우선순위가 낮은 작업과 원거리 작업을 두 개의 백필 큐로 할당한다. 이 같이 다중 큐를 사용하는 이유는 그리드 노드 간의 메타 스케줄링시 지역 작업과 원거리 작업 간의 우선순위를 부여하여 지역 작업이 원거리 작업보다 먼저 실행되게 하기 위한 것이다. 즉, 다중 큐를 사용함으로써 노드 내의 스케줄링 정책과 메타 스케줄링 정책을 동시에 고려한 작업 스케줄링 정책이 가능하게 된다.

노드 내의 작업 스케줄링 정책으로 시스템의 이용률을 높이고 작업 지연 시간을 감소 시키며 작업의 무한대기를 방지하기 위한 방법으로 병렬 스케줄링의 두 가지 백필 방법의 단점을 보완한 방법을 제안한다. 앞에서 살펴본 바와 같이 Conservative 백필은 시스템 이용률이 좋지 않고 EASY 백필은 작업 지연 시간이 증가하고 무한대기가 발생할 수 있다. 따라서 EASY 백필의 단점인 무한대기를 방지 하고, Conservative 백필의 단점인 낮은 시스템 효율을 높이기 위하여, 1개의 작업 큐와 2개의 백필 큐로 나누어 작업들을 특성에 따라 서로 다른 큐에 배치하고, 작업 큐와 백필 큐에서 작업의 실행을 보장하기 위해 예약 기법을 사용한다. 지역 작업은 1개의 작업 큐와 1개의 백필 큐에 할당하고, 원거리 작업은 다른 백필큐에 저장하여 지역 작업과 원거리 작업을 구분하고, 지역 작업도 2 개의 큐에 나누어 배치하여 보다 많은 백필이 이루어질 수 있도록 한다. 또한 예약되는 작업의 수를 EASY 백필보다 많이 늘리고 Conservative 백필보다 줄여 Conservative 백필보다 시스템 이용률을 높이고 EASY 백필보다 작업 지연 시간을 감소시키 위한 방법을 연구한다.

3. 하이브리드(Hybrid) 스케줄링 구조

그림 5는 본 논문에서 제안한 하이브리드 스케줄링을 사용하는 그리드 컴퓨팅 시스템의 구조를 보여준다. 각 노드(Node)는 그리드 컴퓨팅에 참여한 컴퓨터(Worker)

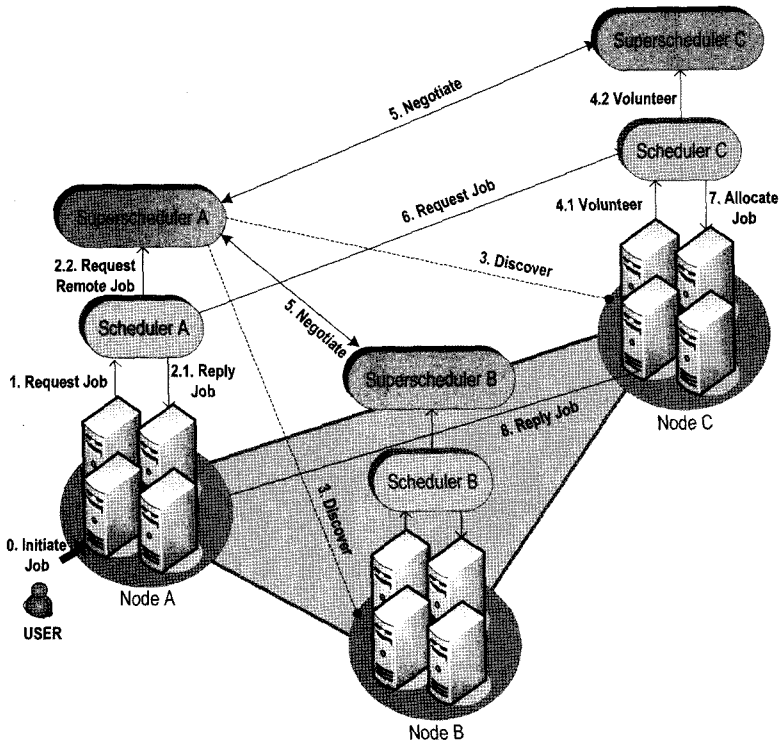

그림 5 그리드 시스템 구조도

와 지역 스케줄러 및 전역 스케줄러(Superscheduler)로 구성된다. 하이브리드 스케줄링 구조란 노드 내 스케줄링 부하를 최소화하기 위한 계층적 스케줄링 전략과 노드간 협업을 위한 전역 스케줄러간 상호작용을 지원하기 위한 그리드 시스템 구조이며 메타 스케줄링과 지역 작업 스케줄링을 병행하여 전체 시스템의 효율성 및 가용성을 증가시킬 수 있는 구조이다. 따라서 이는 전통적인 계층적 스케줄링 전략의 지역/전역 작업 스케줄링이 가능하다는 장점과 De-Centralized 스케줄링 전략의 단점인 통신 병목현상을 제거하고 SPOF(Single Point Of Failure)가 없으며, 확장성이 좋다는 장점을 모두 제공할 수 있는 구조이다.

하이브리드 구조가 적용된 시스템은 그리드 컴퓨팅에 참여하는 컴퓨터들이 인터넷을 통하여 논리적인 하나의 그룹으로 연결되며 마치 하나의 컴퓨터처럼 동작한다.

하이브리드 스케줄링 구조로 구성된 그리드 시스템은 개략적인 동작은 다음과 같다. 그리드 클라이언트들이 작업을 제출(0. Initiate Job)하면 해당 노드의 스케줄러(Scheduler A)는 1)지역 작업 스케줄링 정책에 의해 작업을 작업 큐에 분배하여 처리될 수 있도록 하거나(2.1. Reply Job) 또는 2)백필 큐에 분배하여 작업 큐에 할당되길 기다리거나 3)다른 노드에게 작업 처리를 요청(2.2. Request Remote Job)한다. 만약 해당 노드의 과부하로 인해 해당 노드에서 작업을 실행하기 어려운 경우 다른 유휴 노드가 존재하면, 그 노드로 작업을 보낸다.

작업을 원격 처리하기 위한 메타 스케줄링은 지역 스케줄러 및 해당 노드 상태 정보(Volunteer)로부터 생성된 통합 정보를 노드간에 공유(5. Negotiate)하여 전역 스케줄링 정책에 부합되도록 작업을 분배한다(6. Request Job). 작업을 인계 받은 노드의 지역 스케줄링 전략에 따라 작업을 실행하고(7. Allocate Job) 결과값을 그리드 클라이언트에게 전송한다(8. Reply Job).

그림 6은 본 논문에서 제시한 그리드 컴퓨팅 시스템의 특정 지역 노드의 다중 큐 스케줄러의 구조이며, 각 지역 노드는 스케줄러와 그리드 컴퓨팅에 참여한 컴퓨터들로 구성된다. 원격 요청 작업과 지역 요청 작업 분배를 결정하기 위한 지역 스케줄러는 성능 향상을 위하여 다중 큐 백필 메커니즘이 탑재되어 있다. 스케줄러는 순서에 의해 차례대로 실행되는 작업 큐와 백필을 하기 위해 기다리는 두 개의 백필 큐로 구성되며, 노드 내의 모든 컴퓨터들의 상태와 작업들의 상태를 파악하고 관리하며, 인터넷을 통하여 제출되는 클라이언트의 작업을 스케줄링 정책에 의해 작업 큐나 백필 큐에 저장하고, 다른 노드로 작업을 보내거나 다른 노드로부터 발송된 원거리 작업을 원거리 작업 스케줄링 정책에 의해 두 개의 백필 큐에 분배한다.

그림 6과 같은 다중 큐 스케줄링 기법에서는 임의의 노드로 작업이 제출되면 해당 노드의 작업 스케줄러에 의해 적절한 큐로 보내지며, 이 과정에서 해당 노드가 과부하로 인해 작업을 처리하기가 어려울 경우, 그리드 상에서 가장 부하가 적은 즉, 백필 큐에 가장 적은 노드로 해당 작업을 보낸다. 이처럼 작업 스케줄러는 임의의

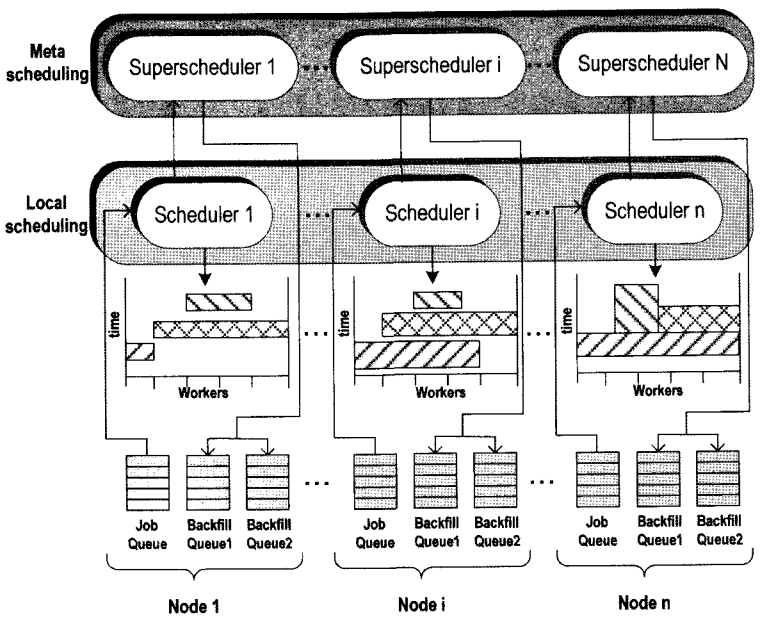

그림 6 노드 i의 다중 큐 스케줄러

노드에 제출된 지역 작업들을 작업 큐와 백필 큐로 분배하고 외부로부터 보내진 원거리 작업들은 백필 큐로 분배하는 기능을 수행한다. 작업 큐와 백필 큐의 선택은 각 작업이 수행되기 위해서 필요한 프로세서 수와 해당 작업이 수행되는데 걸리는 시간을 기준으로 이루어지며, 작업 큐는 주로 많은 프로세서를 필요로 하는 작업들이 분배되며, 이 들은 우선순위에 의해 차례대로 실행되도록 예약된다. 반면에 백필 큐는 주로 적은 수의 프로세서를 필요로 하는 작업들이 분배되어, 백필하기에 용이한 작업들로 구성된다.

4. 다중 큐 스케줄링 기법

본 장에서는 다중 큐 관리 전략, 지역 스케줄링 전략 및 전역 스케줄링 전략에 대해서 설명한다.

4.1 다중 큐 관리 전략

다중 큐 분배는 해당 노드로 제출되는 작업들을 특성에 따라 작업 큐와 백필 큐로 분배하는 과정이다. 백필 스케줄링에서는 얼마나 많은 작업을 효율적으로 백필하느냐가 중요한데, 백필 될 확률이 높은 작업은 필요 프로세서 수가 적은 작업들이다. 이는 필요 프로세서 수가 적을 수록 노드 내의 유휴 프로세서 확보가 용이하기 때문에 백필 될 확률이 높아지게 되고 이로 인해 전체 프로세서의 이용률을 향상시킬 수 있기 때문이다. 또한 작업 큐에 들어가는 작업들은 순서에 의해 차례대로 실행(FIFO 전략)되며 백필 큐로 들어간 작업들은 작업 큐의 작업들이 실행되는 중간에 백필 가능한 시기에 백필된다. 이와 같이 작업을 분류하는 시기는 작업이 지역 노드로 제출되거나 지역 노드의 작업이 끝나거나 다른 노드로부터 원거리 작업이 발송될 때 마다 이루어지게 된다.

기본적인 작업 분류 과정은 두 단계로 이루어지며, 첫 번째 단계에서는 작업들을 해당 노드 시스템의 전체 프로세서 수를 기준으로 세 개의 그룹으로 나누며, 두 번째 단계에서는, 앞선 단계에서 나누어진 각 그룹의 작업들을 예상 실행시간에 따라, 또 다시 두 개의 그룹으로 나눈다. 이같이 작업들을 여러 개의 그룹으로 나누어 본 논문에서 제시한 다중 큐 정책에 따라 한 개의 작업 큐와 두 개의 백필 큐에 나누어 분배하여, 많은 프로세서를 필요로 하는 작업들을 작업 큐로 배치함으로써 가능한 먼저 실행이 되도록 하였고, 두 개의 백필 큐에 분배된 작업들은 적은 수의 프로세서를 필요로 하는 작업들로써, 적합한 유휴 자원 공간을 채우는 과정을 쉽게 이루어지도록 하여 즉, 백필이 많이 이루어지도록 하여 시스템 자원의 이용률을 높일 수 있게 하고자 하였으며, 원거리 작업들을 두 개의 백필 큐에 분배함으로써 지역 작업보다 우선 순위를 낮게 한다. 또한 백필 큐를 두 개로 분리함으로써 두 개의 큐에서 백필하기에 보다 적절한 작업을 찾아 백필 확률 높임으로써 자원의 이용률을 높일 수 있도록 한다.

한편 노드 내에서 발생한 지역 작업(Local Job)이 다른 노드에서 발생한 원거리 작업(Remote Job)에 의해 지연되는 것을 방지하기 위해 작업 유형(Flag)을 두 가지로 분류 한다(그림 7 참조). 즉, 다른 노드로부터 발송된 원거리 작업은 지역 내 분배 정책에 의해 작업 큐와 백필 큐에 분배하지 않고 원거리 분배 정책에 맞춰 두 개의 백필 큐에만 분배되도록 한다. 이는 해당 노드 내에서 발생한 작업의 우선순위를 보다 높게 설정하여 원거리 작업에 의해 지역 작업이 지연되는 것을 줄이기 위한 것이다.

4.1.1 지역 작업 분배 전략

작업 스케줄링에 다중 큐 관리 전략을 적용하기 위해 지역 스케줄러와 전역 스케줄러는 작업번호 (i), 작업도착시간 (TA_i) 그리고 그리드 상의 노드 번호 (k) 등의 파라미터를 포함한 정보를 수집 및 저장한다. 또한 시스템에 제출되는 작업들은 실행에 필요한 프로세서의 수 (P_i) 와 해당 작업을 실행시키는데 필요한 예측수행시간 (TES_i) 의 파라미터를 포함한다.

$$JOB_i(TA_i, TES_i, P_i, k) \quad (1)$$

다중 큐 관리 정책을 구현하기 위한 의사코드의 세부적인 분류 기준을 설명하면 다음과 같다.

• 단계 1 - 프로세서 수에 따른 작업 분류

임의의 노드에서 총 프로세서 수를 TP_i 할 때, 해당 노드의 총 프로세서 수를 3등분하여 현재 제출된 작업을 세 가지 종류로 분류하는 과정으로 식 (2)~식 (4)와 같은 기준으로 분류할 수 있다.

$$\text{W(Wide Job)} : \left\lfloor TP_i * \tfrac{2}{3} \right\rfloor \le P_i \le TP_i \quad (2)$$

$$\text{M(Medium Job)} : \left\lfloor TP_i * \tfrac{1}{3} \right\rfloor \le P_i < \left\lfloor TP_i * \tfrac{2}{3} \right\rfloor \quad (3)$$

$$\text{N(Narrow Job)} : 1 \le P_i < \left\lfloor TP_i * \tfrac{1}{3} \right\rfloor \quad (4)$$

앞서 언급한 것처럼 작업들을 3등분 하여 한 개의 작업 큐와 두 개의 백필 큐에 배치함으로써 많은 프로세서를 필요로 하는 작업들(Wide Job)은 작업 큐에서 실행되도록 하고, 적은 프로세서를 필요로 하는 작업들(Medium Job, Narrow Job)은 두 개의 백필 큐로 분배하여 보다 많은 백필이 이루어지도록 한다.

• 단계 2 - 실행시간에 따른 작업 분류

단계 1에서 분류한 3개의 작업 그룹에 있는 작업을 실행 시간에 따라 각각 두 개의 그룹으로 나눈다. 현재 시스템에 있는 작업의 실행 시간을 평균치 보다 현재 작

```
int MultiQueue_Management(int node, struct JOB * job, int flag)
{
    if(flag==LOCAL){
        if(_Sequence[n].Seq[_Sequence[n].front].Processor <= (Num_Processor/3))
            queue_type = Queue_Backfill2;
            if( _Sequence[n].Seq[_Sequence[n].front].Service < _NODE[n].Job_mean)
                _Sequence[n].Seq[_Sequence[n].front].Priority = Narrow_Short;
            else
                _Sequence[n].Seq[_Sequence[n].front].Priority = Narrow_Long;
        else if(_Sequence[n].Seq[_Sequence[n].front].Processor <= (Num_Processor/3*2))
            queue_type = Queue_Backfill1;
            if(_JOB[TOTAL_JOB].Service < _NODE[n].Job_mean)
                _Sequence[n].Seq[_Sequence[n].front].Priority = Medium_Short;
            else
                _Sequence[n].Seq[_Sequence[n].front].Priority = Medium_Long;
        else
            queue_type = Queue_Job;
            if(_Sequence[n].Seq[_Sequence[n].front].Service < _NODE[n].Job_mean)
                _Sequence[n].Seq[_Sequence[n].front].Priority = Wide_Short;
            else
                _Sequence[n].Seq[_Sequence[n].front].Priority = Wide_Long;
    } else if(flag == REMOTE)
        if(_Sequence[n].Seq[_Sequence[n].front].Processor <= (Num_Processor/2) )
            queue_type = Queue_Backfill2;
            if(_Sequence[n].Seq[_Sequence[n].front].Service < _NODE[n].Job_mean)
                _Sequence[n].Seq[_Sequence[n].front].Priority = Remote_Narrow_Short;
            else
                _Sequence[n].Seq[_Sequence[n].front].Priority = Remote_Narrow_Long;
        else
            queue_type = Queue_Backfill1;
            if(_Sequence[n].Seq[_Sequence[n].front].Service < _NODE[n].Job_mean)
                _Sequence[n].Seq[_Sequence[n].front].Priority = Remote_Wide_Long;
            else
                _Sequence[n].Seq[_Sequence[n].front].Priority = Remote_Wide_Short;
}
```

그림 7 다중 큐 경영 전략 의사코드(Pseudocode)

업이 길면 L(Long)으로, 짧으면 S(Short)로 구분한다. 평균 실행 시간을 기준으로 예상 실행 시간이 크면 긴 작업으로 그렇지 않으면 짧은 직업으로 분류한다. 평균 실행 시간은 실제 많은 작업을 실행하기 전까지는 평균값이라 보기 어려울 수도 있지만, 많은 작업을 실행하게 되면 실제적인 평균치에 가까워지게 된다. 실행 시간에 따라 작업들을 다시 한 번 더 분류 하여 실행 시간이 적은 작업들의 우선 순위를 높게 하여, 작업 큐에서 먼저 실행되거나, 백필 큐에서 먼저 백필되도록 한다.

작업의 예상 실행 시간을 TES_i, 해당 노드에서 실행 중인 작업의 평균 실행 시간을 $TMEAN_i$이라 할 때 다음과 같이 나눈다.

$$\text{L(Long Job)} : TES_i \geq TMEAN_i \quad (5)$$

$$\text{S(Short Job)} : TES_i < TMEAN_i \quad (6)$$

그림 8은 위에서 언급한 단계 1과 단계 2과정을 거쳐 작업들을 실제 큐에 배치하기 위하여 우선순위에 맞게 배치한 형태로 가장 왼쪽에 있는 작업들이 가장 우선순위가 낮고 오른쪽으로 갈수록 우선순위가 높아지게 배치한 것이다.

4.1.2 원거리 작업 분배 정책

다음으로 앞서 기본 전략에서 설명하였듯이 다른 노드로부터 발송된 원거리 작업은 작업 큐와 백필 큐에 분배하지 않고 두 개의 백필 큐에만 분배한다. 이는 지역 작업이 원거리 작업 보다 가능한 먼저 처리될 수 있도록 우선순위를 주기 위해서 이다. 원거리 작업을 분류하는 과정도 두 단계로 나누어 분류하며, 첫 번째 단계는 원거리 작업을 실행하기 위해 필요한 프로세서 수 (P_i)로 나누는 과정으로, 식 (7)과 식 (8)에 의해 RW (Remote Wide) 작업과 RN(Remote Narrow) 작업으로

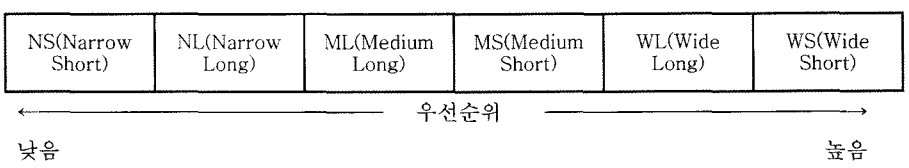

그림 8 지역 작업의 분배 및 우선순위

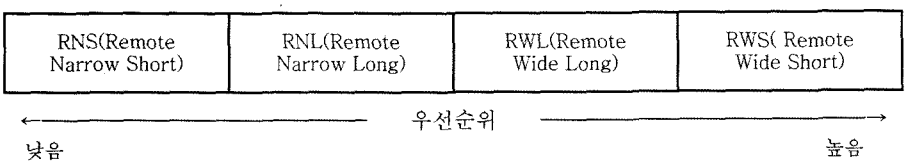

그림 9 원거리 작업의 분배 및 우선순위

구분한다.

$$\text{RW(Remote Wide Job)} : \left\lfloor TP_i * 1/2 \right\rfloor \le P_i \le TP_i \quad (7)$$

$$\text{RN(Remote Narrow Job)} : 1 \le P_i < \left\lfloor TP_i * 1/2 \right\rfloor \quad (8)$$

두 번째 단계는 실행시간에 따라 나누는 과정으로 식 (9)와 식 (10)에 의해 나눈다. 그림 9는 두 단계를 거쳐 분류된 작업들을 우선 순위에 따라 배열한 결과를 보여준다.

$$\text{RL(Remote Long Job)} : TES_i \ge TMEAN_i \quad (9)$$

$$\text{RS(Remote Short Job)} : TES_i < TMEAN_i \quad (10)$$

4.2 지역 스케줄링과 전역 스케줄링 전략

그림 10은 노드 내의 효율적인 작업 분배를 위한 지역 스케줄링 전략과 전체 그리드 시스템의 이용률을 향상시키기 위한 전역 스케줄링 전략을 보여준다. 최초 사용자가 작업 요청을 하면 전역 스케줄러는 작업을 해당 노드에서 처리하게 할 것인지, 원격 노드에서 처리하게 할 것인지에 대한 의사결정을 내리기 위해 전역 스케줄러는 현 노드 및 그리드 시스템의 상태 정보를 수집한다. 또한 전역 스케줄러는 항상 다른 노드에서의 원격 처리 작업 요청을 바로 처리하기 위해 대기 상태를 유지하며, 필요 시에 다른 전역 스케줄러와의 협업을 수행한다.

본 논문에서는 무한대기를 방지하기 위하여 기존의 병렬처리 시스템의 작업 스케줄링에서 많이 사용되는 백필 방법 중 대표적인 Conservative 백필과 EASY 백필 두 방법의 단점을 보완하여, 작업 큐에서 무한대기를 방지하는 방법과 백필 큐에서 무한대기를 방지하는 두 가지 방법의 예약 기법을 사용한다.

첫 번째 예약 기법으로 작업 큐에 있는 작업이 무한대기 하지 않도록 하기 위해 작업 큐에서 실행이 보장되는 작업의 수를 여러 개로 늘린다. 즉, 백필되는 작업에 의해 지연되지 않아야 하는 작업 들을 여러 개 예약함으로써 무한대기 하지 않고 언젠가는 실행되도록 한다. EASY 백필 방법은 예약되는 작업의 수가 큐의 처음에 있는 한 개의 작업만을 대상으로 하지만 본 논문에서는 예약 작업의 수를 여러 개로 늘려서 시스템의 이용률을 높이고 무한대기를 방지한다.

현재 작업 큐에 있는 예약되는 작업의 수를 J_r, 작업의 수를 J_n, 예약 작업의 개수를 조정하기 위한 임계치(Threshold)를 α라 할 때 예약 작업의 수는 식 (11)에 의해 구한다.

$$J_r = J_n * \alpha, \quad 0.1 \le \alpha \le 0.5 \quad (11)$$

두 번째로, 백필 큐에 있는 작업들에 대해서도 예약 기법을 적용하였다. 백필 큐에 있는 작업들은 예약되어 있는 작업들이 아니라 백필되기를 기다리고 있는 작업이며, 실행되기에 필요한 프로세서수가 확보되지 않으면 오랫동안 실행되지 못하고 무한대기에 빠질 수 있다. 특히 원거리 작업의 경우는 무한대기가 될 확률이 더 높아지게 된다. 따라서 백필 큐에서 백필 횟수를 기록하여 백필되지 못한 수치가 일정 값보다 커질 경우 이 작업을 작업 큐로 보내어 일정 시간 후에 실행을 보장 받도록 하거나 다른 노드에서 실행 가능한 경우 그 노드로 발송하여 실행되도록 함으로써 무한 대기를 방지한다.

또한 지역 스케줄러는 가용한 프로세서 수 및 자원 사용률(메모리/네트워크 사용량 등) 등 노드 상태를 종합 수집 분석하여 작업 실행 시 야기될 수 있는 시스템 지연 여부 및 예상 응답 시간 등을 고려하여 작업 실행 여부를 결정한다. 백필 큐에서 백필 가능 여부를 검사할 때 해당 작업이 백필의 기회가 부여되었으나 백필되지 못한 횟수를 J_m라 하고, 백필 횟수의 최대 임계치를 β라 할 때, 식 (12)에 의해 검사되어 예약된다.

$$J_m \ge \beta \quad (12)$$

백필 큐에서 지연된 횟수가 증가된 작업의 경우 우선 다른 노드에게 인계하여 처리 가능한지를 점검하여, 만약 그렇다면 작업을 원격 처리하기 위한 작업 이동 시간 등을 고려하여 원격 처리 여부를 결정하게 함으로써 노드의 원거리 작업 실행이 미치는 영향을 최소화되도록 한다.

5. 성능 평가

본 논문에서 제안한 하이브리드 다중 큐 스케줄링 기법의 성능 평가를 위하여 백필 비율(Backfilling Ratio)을 통해 시스템의 자원 효율성을 분석하고, 작업의 평균 지연율(Slowdown Ratio)를 통해 다중 큐와 원격 작업 처리로 인해 얻을 수 있는 성능 개선 효과에 대해 분석한다. 백필 비율이 높아짐은 단위 시간 내에 많은 프로

```
int Global_Scheduler(int node, struct JOB *job, int flag)
{
     Maintain the volunteer list of its relevant node;
     Gather the node state information;

     Listen to the remote execution request from other global scheduler;
     Negotiate the resource usage of other node between global scheduler if necessary;

     if(its resource state is to be able to process a request job)
          Make Response_Commitment response to a corresponding global scheduler;
          Enqueue_To_Backfill2_Queue(a job) after taking over the job from a corresponding;
          Request Job_Execution to its Local_Scheduler;
     else
          Reply with Response_Busy message to a corresponding global scheduler;

     if(Get Response_Busy message)
          Reservation_Counter_Backfill2++;

     if(Reservation_Counter_Backfill2 > Threshold)
          fix a job's priority into the urgent level;
}
int Local_Scheduler(int node, struct JOB *job, int flag)
{
     job.flag = (Global_Scheduler(node) == LOCAL_DONE ? LOCAL : REMOTE);

     queue_type = MultiQueue_Management(node, job, flag);

     if(queue_type == _Job_Queue)
          if(the number of available Workers <= the number of request workers)
               Allocate the Job to the workers in a node;
               Execute_Job(node, job, timestamp);
               Update the Mean Expected Execution Time of a total of Jobs in a node;
          else
               Update the Mean Waiting Time in Job Queue of a node;
               Enqueue_To_Job_Queue(job);
     else if(queue_type == Backfill_Queue1)
          if(the number of available workers <= the number of request workers)
               if(the execution of a job in Backfill Queue1 don't delay the waiting jobs in Job Queue)
                    Allocate the Job to the workers in a node;
                    Execute_Job(node, job, timestamp);
                    Update the Mean Expected Execution Time of a total of Jobs in a node;
               else
                    Update the Mean Waiting Time in Backfill Queue1 of a node;
                    job.Reservation_Counter_Backfill1++;
          else
               if(job.type == Local)
                    wait(job);
                    Update the Mean Waiting Time in Backfill Queue1 of a node;
               else
                    Move_To_Backfill2(job, timestamp);
     else if(queue_type == Backfill_Queue2){
          if(the number of available workers <= the number of request workers)
               if(the execution of a job in Backfill Queue2 don't delay the waiting jobs in other Queue)
                    Allocate the Job to the workers in a node;
                    Update the Mean Expected Execution Time of a total of Jobs in a node;
               else
                    Update the Mean Waiting Time in Backfill Queue2 of a node;
                    Reservation_Counter_Backfill2++;
          else
               Update the Mean Waiting Time in Backfill Queue1 of a node;
               if( !(Volunteer_Node = Negotiate_Global_Scheduler()) )
                    Migrate_Job( job, Volunteer_Node, timestamp);

     if(Reservation_Counter_Backfill1 > Threshold1 || Reservation_Counter_Backfill2 > Threshold2 )
          Request_Reservation_To_the_global_scheduler(this.queue.point);

     return queue_type;
}
```

그림 10 지역 및 전역 스케줄링 전략 의사코드

세서가 보다 많은 작업을 처리한 것으로 볼 수 있으며, 평균 지연율이 낮아짐은 작업이 처리되는 대기 시간이 줄어들어 결과적으로 시스템의 성능이 좋아짐을 의미한다. 이러한 점은 기존의 연구[2,18]에서 파악할 수 있다. 또한 스케줄링 전략에 따라 발생할 수 있는 작업 지연 효과를 평균 응답 시간(Average Response Time) 척도(Metric)를 사용하여 비교한다. 작업의 평균 지연 시간은 식 (13)을 적용하며[17], 여기서 N은 전체 작업 수, S는 가장 수행이 짧은 작업의 길이, $resp(i)$는 작업이 제출되어 끝날 때까지의 시간을 의미하며, $exec(i)$는 작

업의 수행 시간, $width(i)$는 작업이 실행되기 위해 필요한 프로세서 수를 의미한다.

$$AverageSlowdown = \frac{\sum_{i=1}^{N} \frac{resp(i)}{width(i) \times \max(S, exec(i))}}{N} \quad (13)$$

이와 같이 발생 가능한 평균 지연 시간을 계산하고 다중 큐를 사용하는 경우($AverageSlowdown_m$)가 단일 큐를 이용하였을 경우($AverageSlowdown_1$)보다 얻을 수 있는 성능 이득이 어느 정도인지 분석하기 위해 식 14와 같이 평균 지연율을 정의한다[5]. 즉 Slowdown-Ratio가 0보다 크다는 의미는 단일 큐를 이용한 경우 평균 지연시간이 더욱 크다는 것을 뜻하며 따라서 다중 큐를 사용하였을 경우 평균 지연시간을 줄일 수 있다는 것을 의미한다. 반대로 0보다 작다는 것은 다중 큐를 사용함으로써 관리 오버헤드가 증가하여 평균 지연시간이 증가되었다는 것을 의미한다.

$$SlowdownRatio = \frac{AverageSlowdown_1 - AverageSlowdown_m}{\min(AverageSlowdown_1 - AverageSlowdown_m)} \quad (14)$$

기존 연구결과[6,11,12,17-20]를 분석해 보면, 동일한 기법이라 할지라도 대상 작업부하에 따라 성능이 개선될 수도 또는 저하될 수 있기 때문에 입력 데이타의 선정은 중요하다. 성능 평가를 위한 실험 데이타 생성을 위하여, 본 논문에서는 평균 작업 도착 시간(Mean Arrival Time), 평균 작업 요청 시간(Mean Estimated Execution Time), 평균 필요 프로세서 수(Mean Number of Processor, Mean Width) 등을 실험을 위해 필요한 파라미터 값을Feitelson Archive[21]의 작업 부하 로그(Workload Logs)의 집합으로부터 추출하였고, 해당 파라미터를 기본값으로 하는 확률 함수를 근간으로 만들어진 RGD(Randomly Generated Data)를 입력 데이타로 사용한다.

실험은 펜티엄 PC 상에서 비주얼 C 프로그램을 통하여 이루어졌고, 로컬 그리드 노드 내의 컴퓨터간의 통신 속도는 그리드 노드 간의 통신속도에 비해 두 배 빠르다고 가정한다. 실험을 위해 사용된 기본적인 그리드 시스템 설정은 노드 수는 8개, 노드당 평균 보유 프로세서 수는 64개, 작업 도착율은 평균 분당 0.167 (≈ 7200개/월)을 갖는 지수함수, 작업 요청 시간은 평균 100 분을 갖는 지수함수를 따르며, 작업들은 서로 의존성이 존재하지 않는 독립된 작업이며, 평균 필요 프로세서 수는 1에서 64개의 Uniform 함수를 따른다[7].

그림 11은 그리드 전체 시스템의 작업 부하량(System Utilization, 시스템 이용률)에 따라 백필되는 작업 비율을 Conservative 백필 방식과 EASY 백필 방식과 비교한 결과이다. 노드 별 작업 도착 시간과 처리 시간 평균

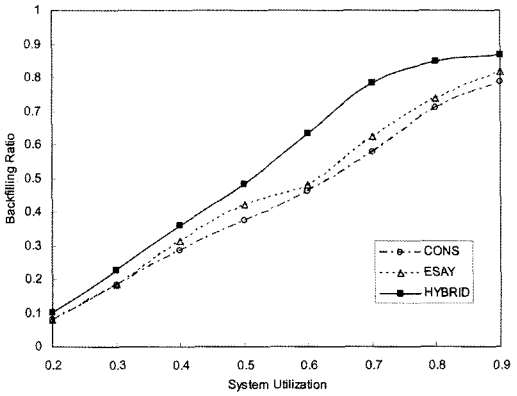

그림 11 시스템 부하량에 따른 백필 비율

에 의해 계산된 시스템 이용률($\rho_{mean}=\lambda_{mean}/\mu_{mean}$)이 증가하는 것은 단위 시간당 새로 생성된 작업 요청이 빈번하게 발생함을 뜻한다. 전체적으로 Conservative 백필 방식보다EASY 백필 방식이 백필되는 횟수가 많으며, EASY 백필 방식보다 하이브리드 다중 큐 스케줄링 방식이 백필 횟수가 많다. 이는 하이브리드 다중 큐 스케줄링 방식이 전체 그리드 시스템의 프로세서 사용률을 향상시켜 보다 자원 관리가 효율적으로 이루어짐을 보여준다. 또한 시스템 부하량이 증가할수록 백필 가능성이 높은 작업 생성 가능성이 높아지므로 세가지 방식 모두 백필 비율이 증가함을 볼 수 있다.

하지만 자원 사용률이 증가되었다는 것이 작업의 응답시간을 최소화시킬 수 있다는 것을 의미하는 것은 아니므로 작업 지연율 및 평균 응답시간과 같은 척도로 전체 시스템 성능에 관한 분석을 수행한다. 그림 12는 요구 프로세서 수 및 작업 예상 수행시간에 따라 식 (14)에서 정의한 작업의 평균 지연율을 분석한 결과로 세가지 방식 모두 단일 큐만을 사용할 때보다 다중 큐를 사용함으로써 작업 지연율을 줄일 수 있는 효과를

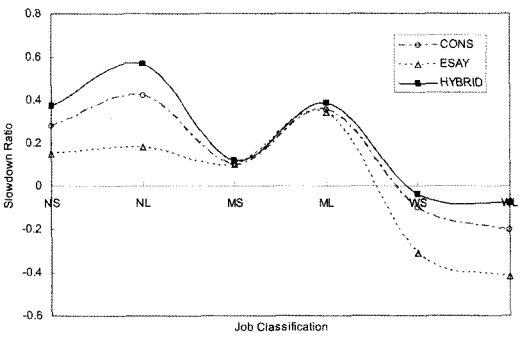

그림 12 요구 프로세서 수 및 예상 수행시간에 따른 평균 지연율

얻을 수 있다는 결과를 보여준다. 한편 다중 큐를 사용한다 할지라도 백필에 의한 대기 현상으로 발생할 수 있는 지연 현상은 상대적으로 적은 프로세서를 요구하는 작업의 경우에는 발생하지 않았지만 세가지 방식 모두 다수 프로세서를 필요로 하는 작업(WS & WL)의 경우 우선순위가 낮은 작업이 백필되는 것에 의해 지연이 발생함을 볼 수 있다.

그러나 EASY 백필 방식과 Conservative 백필 방식보다는 본 논문에서 제안한 하이브리드 다중 큐 스케줄링 방식에서 단일 큐를 사용한다 할지라도 다수 프로세서를 필요로 하는 작업의 지연율이 적게 발생함을 보여준다. 따라서 하이브리드 스케줄링 방식은 단일 큐를 사용하더라도 예약 기법을 통해 백필 비율을 높일 수 있는 EASY 백필 방식과 무한 대기 현상을 방지할 수 있는 Conservative 백필 방식의 장점을 모두 제공하고 있다는 것을 알 수 있다.

그림 13은 Conservative 및 EASY 방식의 단점을 보완하기 위해 개발된 예약 기법에서 사용하는 예약 작업의 적용 범위를 조정하기 위한 임계치 α와 무한 대기 현상을 방지하기 위한 임계치 β 값에 따른 평균 응답 시간을 분석한 결과이다.

우선 α가 증가할수록 평균 응답 시간이 증가함을 볼 수 있는데, 이는 작업 큐로 옮겨져 우선 처리될 수 있는 작업의 범위가 넓어질수록 EASY 백필 방식과 같이 빈번한 백필로 인해 다른 작업의 대기 시간을 증가시키기 때문이다. 또한 백필이 가능한 작업 중에서 백필되지 못한 횟수의 임계치를 적게 설정할수록 이 또한 비슷한 사유로 인해 다른 작업의 대기 시간을 증가시켜 전체 작업의 평균 응답시간을 증가시키는 결과를 초래한다.

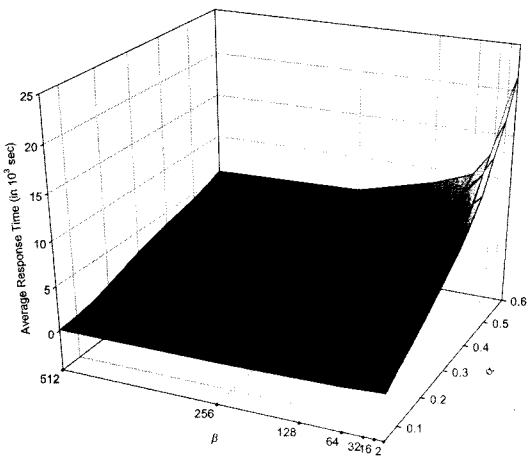

그림 13 예약 기법에 사용되는 파라미터(α,β) 기반 평균 응답 시간

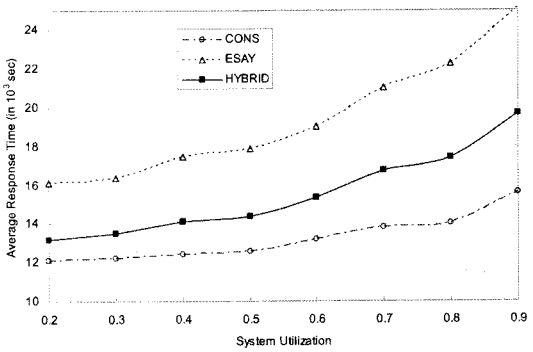

그림 14 시스템 이용률에 따른 평균 응답 시간

하지만 다양한 실험에 따라, α가 0.5인 경우에도 β 값이 적정 수준보다 크다면(예: $\beta \geq 64$) 전체 시스템의 응답 시간에 미치는 영향은 적어지는 것을 확인할 수 있고, 이는 그리드 시스템을 구성하고 있는 노드 별 평균 프로세서 수와 비슷한 값을 갖는데, 노드 별 보유하고 있는 평균 프로세서 수가 많을수록 유휴 프로세서가 발생할 확률이 높아지고 따라서 백필에 의한 지연 현상이 발생할 확률이 적어지기 때문이다.

그림 14는 전체 그리드 시스템 이용률에 따른 평균 응답 시간의 변화를 보여준다. EASY 백필 방식의 경우 시스템 이용률이 증가할수록 평균 응답 시간의 증가 비율 폭이 매우 커진다. 이는 이용률이 증가할수록 소수의 작업이지만 백필로 인해 지속적으로 대기 상태에 머무르게 되는 경우가 발생하여 전체 시스템의 평균 응답 시간을 증가시키기 때문이다. 본 논문에서 제안한 하이브리드의 경우 작업 지연율이 Conservative 방식보다 적은데 비해 평균 응답시간이 높은 현상은 하이브리드 방식이 Conservative 방식보다 다른 노드로 작업을 이동시켜야 되는 경우가 빈번히 발생하기 때문이다. 다시 말해서, 그리드 시스템의 구성 노드를 연결하기 위해 전용(Dedicated)의 네트워크를 사용하지 않는다면 원격 처리를 위한 작업 이동으로 인한 오버헤드가 백필로 통해 얻을 수 있는 이득을 감소시키기 때문이다.

6. 결 론

그리드 시스템의 작업 지연시간 성능 및 이용률을 향상시키기 위한 스케줄링 기법 개발에 대한 연구가 활발히 진행되고 있지만, 대부분의 연구들이 문제 해결 범위를 단순화하기 위해 각 노드에 작업들을 분배하는 메타 스케줄링 방법과 노드 내의 작업 스케줄링 방법에 대한 연구를 분리하여 진행하고 있다. 이런 점을 해결하기 위하여 본 논문에서는 그리드 컴퓨팅 시스템의 작업 스케줄링 정책 중 노드 간의 작업을 분배하는 메타 스케줄링 정책과 지역 노드 내의 작업을 분배하는 작업 스케

줄링을 동시에 고려하는 하이브리드 스케줄링 기법을 제안하였다. 그리드 컴퓨팅 노드로 제출된 작업을 필요한 프로세서 수와 예상 작업수행 시간에 따라 구분하여, 해당 작업을 노드들의 작업 부하에 따라 지역 노드에서 처리할지 다른 노드에서 처리할지를 결정하여, 지역 노드에서 처리할 경우 우선순위가 높은 작업은 실행 가능한 작업 큐로 분배하고 우선순위가 낮은 작업은 백필이 되어지는 백필 큐로 할당하며, 다른 노드에서 처리할 경우 해당 노드의 백필 큐로 할당하였다. 또한 제안된 기법들을 여러 가지 실험을 통하여 성능을 평가하였으며, 그리드 컴퓨팅 시스템의 이용률이 높아지고, 작업 지연 시간이 감소되는 것을 확인하였다. 추후에는 본 논문에서 제안한 기법을 보다 다양한 그리드 시스템 및 응용 그리드에 적용 가능하도록 네트워크 지연으로 야기되는 성능 저하 문제를 고려할 예정이다.

참 고 문 헌

[1] I. Foster, et al., "Grid Services for Distributed System Integration," Computer, Vol. 35, No. 6, pp. 37-46, 2002.

[2] K. Krauter, et al., "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," Software Practice and Experience Journal, Vol. 32, No. 2, pp. 135-164, Feb. 2002.

[3] I. Foster, et al., "Globus: A Metacomputing Infrastructure Toolkit," The International Journal of Supercomputer Applications and Performance Computing, Vol. 11, No. 2, pp. 115-128, Oct. 1997.

[4] http://www.legion.org

[5] B. Lawson, et al., "Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems," The 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, pp. 72-87, July 2002.

[6] H. Shan, et al., "Job Superscheduler Architecture and Performance in Computational Grid Environments," In SC2003 Conference, 2003.

[7] A. Mualem, et al., "Utilization, Predictability, Workloads and User Run time Estimates in Scheduling the IBM SP2 with Backfilling," IEEE Trans. Parallel and Distributed System, Vol. 12, No. 6, pp. 529-543, June 2001.

[8] O. H. Ibarra, et al., "Heuristic Algorithm for Scheduling Independent Tasks on Nonidentical Processors," Journal of ACM, Vol. 24, No. 2, pp. 280-289, Apr. 1977.

[9] V. Hamscher, et. al., "Evaluation of Job-Scheduling Strategies for Grid Computing," The 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000) at the 7th International Conference on High Performance Computing (HiPC-2000), LNCS 1971, pp. 191-202, 2000.

[10] V. Subramani, et al., "Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests," The 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), pp. 359-368, July 2002.

[11] Q. Wang, et al., "De-centralized Job Scheduling on Computational Grids Using Distributed Backfilling," Grid and Cooperative Computing - GCC 2004: Third International Conference, LNCS 3251, pp. 285-292, Oct. 2004.

[12] K. Li, "Job Scheduling for Grid Computing on Metacomputers," The 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), Abstracts Proceedings, Apr. 2005.

[13] B. Bode, et al., "The Portable Batch Scheduler and the MauiScheduler on Linux Clusters," in Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, Georgia, Oct. 2000.

[14] D. G. Feitelson, et al., "Theory and Practice in Parallel Job Scheduling," Job Scheduling Strategies for Parallel Processing, IPPS'97 Workshop, Geneva, Switzerland, LNCS 1291, pp. 1-34, Apr. 5, 1997.

[15] R. McCann, et al., "A Dynamic Processor Allocation Policy for Multiprogrammed Sharedmemory Multiprocessors," ACM Trans. on Computer System, Vol. 11, No. 2, pp. 146-178, May 1993.

[16] D. G. Feitelson, et al., "Improved Utilization and Responsiveness with Gang Scheduling," Job Scheduling Strategies for Parallel Processing, IPPS'97 Workshop, Geneva, Switzerland, LNCS 1291, pp. 238-261, Apr. 1997.

[17] D. Zotkin, et al., "Job-Length Estimation and Performance in Backfilling Schedulers," The 8th IEEE International Symposium on High Performance Distributed Computing (HPDC'99), Aug. 1999.

[18] S. Srinivasan, et al., "Characterization of Backfilling Strategies for Parallel Jobs Scheduling," 31st International Conference on Parallel Processing Workshops (ICPP 2002 Workshops), pp. 514-522, Aug. 2002.

[19] D. Talby, et al., "Supporting Priorities and Improving Utilization of the IBM SP2 Scheduler Using Slack-based Backfilling," In *Proceedings of the International Parallel Processing Symposium*, pp. 513-517, Apr. 1999.

[20] W. A. Ward Jr., et al., "Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy," 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'2002), pp. 88-102, July 2002.

[21] D. Feitelson, "Logs of Real Parallel Workloads from Production Systems," http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

강 창 훈

1986년 2월 충남대학교 계산통계학과(이학사). 1988년 2월 충남대학교 대학원 계산통계학과(이학석사). 2006년 8월 아주대학교 대학원 컴퓨터공학과(공학박사) 1994년~현재 극동정보대학 방송영상미디어과 부교수. 관심분야는 결함허용, 성능분석, 클러스터컴퓨팅, 그리드시스템

최 창 열

1999년 아주대학교 정보통신공학과 졸업(공학사). 2002년 아주대학교 정보통신전문대학원 졸업(공학석사). 2007년 아주대학교 정보통신전문대학원 정보통신공학과(공학박사). 2007년~현재 숭실대학교 정보미디어기술연구소 전임연구원. 관심분야는 오토노믹 컴퓨팅, 홈 네트워크 시스템, SOA, 고가용성 시스템 등

박 기 진

정보과학회논문지 : 시스템 및 이론
제 34 권 제 2 호 참조

김 성 수

1982년 서강대학교 전자공학과 졸업(공학사). 1984년 서강대학교 전자공학과 졸업(공학석사). 1995년 Texas A&M University 전산학과(공학박사). 1983년~1996년 삼성전자 수석연구원. 2002년~2003년 Texas A&M University 교환교수 1996년~현재 아주대학교 정교수. 2006년~현재 아주대학교 정보통신전문대학원 원장. 관심분야는 Dependable System & Networks, Autonomic Computing, Ubiquitous Computing & Networks 등