韓國컴퓨터情報學會 論文誌
第15卷 第2號, 2010. 2.

2010-15-2-1-1

그리드 컴퓨팅을 위한 실시간 작업 스케줄링 정책

최 준 영*, 이 원 주**, 전 창 호***

Real-Time Job Scheduling Strategy for Grid Computing

Jun Young Choe *, Won Joo Lee**, Chang Ho Jeon***

요 약

본 논문에서는 그리드 환경을 구축할 때 자원 비용 면에서 효율적인 스케줄링 정책을 제안한다. 이 스케줄링 정책은 로컬 컴퓨팅 자원을 효율적으로 할당하기 위해 자원 비용과 작업 실패율을 고려한다. 이 스케줄링 정책의 특징은 원격 스케줄러와 로컬 스케줄러를 사용하여 2단계 스케줄링을 수행한다. 원격 스케줄러에서는 자원 데이터베이스에 저장된 네트워크와 로컬시스템의 정보를 사용하여 작업의 총실행시간이 최소인 로컬시스템을 선택하여 작업을 할당한다. 로컬 스케줄러에서는 할당된 작업의 대기시간과 처리시간을 재계산한 후, 작업을 데드라인 내에 처리할 수 있다면 로컬시스템에서 수행한다. 하지만 데드라인을 초과하면 다른 로컬시스템으로 이주시켜 처리함으로써 작업 실패율과 자원 비용을 최소화한다. 제안한 스케줄링 정책은 기존 Greedy 정책에 비해 작업 실패율은 높지만, 자원 비용을 줄이는 면에서 더 우수함을 보인다. 본 논문에서는 시뮬레이션을 통하여 제안한 스케줄링 정책이 기존 Greedy 스케줄링 정책에 비해 컴퓨팅 자원 비용을 줄이는 면에서 효과적임을 보인다.

Abstract

In this paper, we propose a scheduling strategy for grid environment that reduces resource cost. This strategy considers resource cost and job failure rate to efficiently allocate local computing resources. The key idea of our strategy is that we use two-level scheduling using remote and local scheduler. The remote scheduler determines the expected total execution times of jobs using the current network and local system status maintained in its resource database and allocates jobs with minimum total execution time to local systems. The local scheduler recalculates the waiting time and execution time of allocated job and uses it to determine whether the job can be processed within the specified deadline. If it cannot finish in time, the job is migrated other local systems. Through simulation, we show that it is more effective to reduce the resource cost than the previous Greedy strategy. We also show that the proposed strategy improves the performance compared to previous Greedy strategy.

▸ Keyword : 그리드 환경(Grid environment), 자원비용(Resource cost), 작업실패율(Job failure rate).

• 제1저자 : 최준영 교신저자 : 이원주
• 투고일 : 2010. 02. 01, 심사일 : 2010. 02. 08, 게재확정일 : 2010. 02. 22.
*삼성전자 정보통신사업부 연구원 ** 인하공업전문대학 컴퓨터정보과 부교수 *** 한양대 전자컴퓨터공학부 교수

Ⅰ. 서 론

그리드 컴퓨팅은 지리적으로 분산된 네트워크 환경에 수많은 이기종 컴퓨터와 대용량 저장장치, 데이터베이스 시스템, 인공위성 등과 같은 다양한 자원들을 고속 네트워크로 연결하여 그 자원들을 상호 공유할 수 있도록 하는 차세대 컴퓨팅 환경이다[1]. 이러한 그리드 컴퓨팅은 이기종 컴퓨팅 자원들을 자원 대 비용 측면에서 가장 효율적으로 사용할 수 있고, 하나의 가상 환경으로 구성하여 관리할 수 있다. 또한 대량의 데이터나 고속 연산을 필요로 하는 작업의 처리 방법을 결정하고, 그 환경을 구성할 수 있다. 그리드 컴퓨팅 환경과 기존의 인터넷 웹 서비스 환경을 비교해 보면 표 1과 같다.

그리드 컴퓨팅에서는 작업 크기에 따른 네트워크 전송시간과 그리드 자원의 상태 정보를 실시간으로 사용하기 어렵다. 따라서 실시간 작업의 총실행시간을 정확하게 예측할 수 없기 때문에 그리드 자원 할당의 효율성이 떨어지는 문제점이 있다[2][3].

표 1. 그리드 컴퓨팅과 인터넷 서비스 비교
Table 1. Grid computing vs. internet service

기능	인터넷	그리드 컴퓨팅	차이점
정보제공	정적	동적	DNS : LDAP
자원할당	단일자원	다중자원	단일 경로 : 동일논리공간
보안	비단일인증	단일인증	
아키텍처	단일계층	다중계층	mapping : mix
서비스유형	연결	가상조직	ISP : User

본 논문에서는 이러한 그리드 자원 할당의 문제점을 최소화 할 수 있는 새로운 스케줄링 정책을 제안한다. 이 스케줄링 정책의 특징은 원격 스케줄러와 로컬 스케줄러를 사용하여 2단계 스케줄링을 수행한다. 원격 스케줄러에서는 자원 데이터베이스에 저장된 네트워크 환경과 로컬시스템의 정보를 사용하여 작업의 총 실행시간을 예측한다. 그리고 총 실행시간이 데드라인에 근접한 로컬시스템에 작업을 할당하여 자원비용을 최소화한다. 로컬 스케줄러에서는 할당된 작업의 대기 및 처리시간을 실시간으로 계산한 후, 그 시간이 데드라인 내에 처리할 수 있다면 로컬시스템에서 처리한다. 하지만 데드라인을 초과하면 다른 로컬시스템으로 이주(migration)시켜 처리함으로써 자원비용을 최소화한다.

본 논문의 구성은 다음과 같다. 2장에서는 그리드 컴퓨팅의 기존 스케줄링 정책과 기존의 스케줄링 정책의 문제점에 대하여 설명한다. 그리고 3장에서는 제안하는 스케줄링 정책에 대하여 자세히 기술한다. 4장에서는 제안하는 스케줄링 정책의 성능을 평가하고, 그 결과를 분석한다. 그리고 5장에서는 결론 및 향후 연구 과제를 제시한다.

Ⅱ. 관련 연구

2.1 그리드 미들웨어

그리드 환경을 만들기 위해 사용하는 대표적인 것이 미들웨어이다. 미들웨어는 네트워크에 연결된 다양한 장비를 공동으로 활용할 수 있도록 연동시키는 일종의 소프트웨어로써 분산되어 있는 컴퓨팅 자원을 가상의 컴퓨팅 자원으로 보이게 하여 활용할 수 있도록 하는 기술이다. 주로 사용되는 미들웨어로는 Globus, Legion, MOL, AppLes 등이 있다.

1) 글로버스 툴킷(Globus ToolKit)

글로버스 툴킷은 그리드 서비스를 제공하는 미들웨어로써 그리드에서 필요로 하는 다양한 서비스를 독립적인 요소로써 제안하고 있기 때문에 그리드 개발 과제에서 가장 많이 사용되고 있다. 글로버스의 장점은 기존의 각 시스템 및 네트워크의 관리 정책이나 운영 도구를 무시하지 않고 각 요소들과 협력해 그리드를 이룬다는 점이다. 따라서 글로버스 툴 킷의 요소들도 하드웨어 측면이나 소프트웨어 측면에서 상이한 시스템 간에 성능 저하를 줄이면서 통합하기 위한 기능을 위한 것이 대부분이다. 글로버스 툴킷은 크게 그리드 보안, 정보 서비스, 자원 관리, 데이터 관리 등으로 나눠진다.

2) 객체지향 기법을 적용한 미들웨어 'Legion'

Legion은 다양한 컴퓨팅 자원과 네트워크를 연결해 단일 시스템과 같은 모양을 내는 객체지향 메타시스템이다. Legion은 객체지향 방법의 미들웨어라는 면에서 CORBA와 비슷하지만 CORBA와는 달리 고성능 컴퓨팅을 필요로 하는 사용자를 위해 개발되었다. Legion은 사용자 운영체제에 위치하며 컴퓨팅 자원과 애플리케이션 사이를 중재한다. Legion은 자원의 스케줄링을 처리하고 보안 문제를 다루고 있으며, 컨텍스트 공간이라는 네이밍 시스템이 사용되고 있다. 컨텍스트 공간은 사용자가 쉽게 객체를 모니터링하고 사용할 수 있게 해준다. Legion에서는 컴퓨팅 자원과 스토리지 자원을 모두 객체로 표현하는데 호스트 객체, 볼트 객체 등으로 표현한다. Legion의 IDL은 자바, C, C++, 포트란, CORBA IDL로 변환

가능하다. 여러 자원을 사용할 때 발생하는 문제 중 가장 중요한 것은 인증과 권한 부여에 관한 문제다. 일반적으로 각 사이트는 인증에 관한 자신만의 정책을 가지고 있다. 이러한 정책은 SSH나 Kerberos와 같은 보안 서비스를 이용하거나 단순한 패스워드를 이용한다.

3) 유휴 컴퓨팅 자원 관리 미들웨어 'Condor'

Condor는 분산된 워크스테이션을 통합하여 사용하는 초생산성 컴퓨팅 환경이다. 현재 대부분의 유닉스 시스템에 설치가 가능하고, NT 시스템 기반은 개발 중에 있다. Condor의 중요한 특징은 ClassAd 기법에 있다. 각 사용자는 자신의 작업을 수행하기 위해 적합한 컴퓨팅 자원을 찾기 원한다. ClassAd는 각 컴퓨팅 자원에 대해 프로세서 타입, 메모리 양, 운영체제 타입 등 모든 가능한 특징에 관한 정보를 담고 있다. 또한, 사용자는 자신의 작업이 부동소수점 연산에 있어 더 나은 컴퓨터에서 수행되거나 특정한 컴퓨터에서 수행되기를 원할 수 있는데 이와 같은 모든 것을 ClassAd에서 기술할 수 있다. Condor는 사용자 작업에 관한 ClassAd와 각 컴퓨팅 자원의 ClassAd를 일치시킴으로 적당한 자원을 찾을 수 있다.

4) 모듈화를 중심으로 한 미들웨어 'Cactus'

Cactus는 그리드 환경에서 과학기술 분야 연구를 위한 문제 해결 환경으로 구조가 모듈화 되어 있다는 것이 특징이다. Cactus는 단일 시스템뿐만 아니라 클러스터나, 슈퍼컴퓨터 등 다양한 플랫폼에 구축될 수 있으며 현재 애플리케이션 사용자가 사용하는 도구를 쉽게 연결해 사용할 수 있는 환경을 제공한다. 예를 들면, 글로버스 툴킷, HDF5 병렬파일 I/O, PETSc 과학라이브러리, 가시화 도구 등이다.

2.2 그리드 컴퓨팅 환경을 위한 스케줄링 정책

1) 그리드 스케줄링 시스템

그리드 스케줄링 정책은 실시간 작업처리를 필요로 하는 시스템에 주로 사용한다. 스케줄링 정책의 최소실행시간 예측 모듈의 정확성에 따라 시스템의 성능이 결정된다. 하지만 이질적이고 다양한 자원이 분산된 그리드 환경에서는 응용프로그램의 실행시간을 예측하는 것은 매우 어려운 일이다[4]. 그리드 컴퓨팅에서 자원 할당을 위한 스케줄링 시스템의 구성은 그림 1과 같다[5].

그림 1에서 모니터는 네트워크 모니터와 자원 모니터로 구성된다. 네트워크 모니터는 네트워크의 전송속도 및 사용 상태를 모니터링하고, 자원 모니터는 로컬 시스템의 사용 상태를 모니터링 하여 그 결과 데이터를 자원 데이터베이스에 저장한다. 원격스케줄러는 그리드 환경의 상태 정보를 바탕으로 현재 그리드에서 수행 중인 작업의 실행시간을 예측한다. 그리고 예측한 정보를 바탕으로 대기 중인 작업의 실행시간을 최소화할 수 있는 노드에 작업을 할당한다.

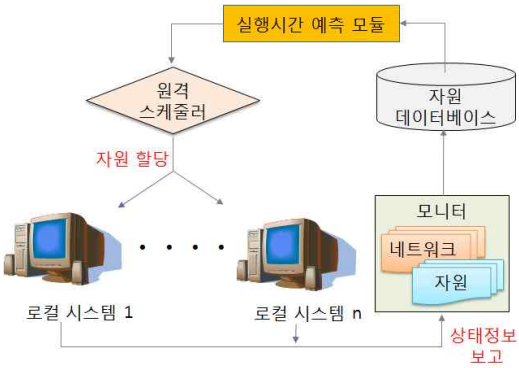

그림 1. 그리드 스케줄링 시스템
Fig. 1. Grid Scheduling System

예측모듈은 자원 데이터베이스의 정보를 사용하여 작업 총실행시간을 예측한다. 자원 데이터베이스의 자원정보는 네트워크 상태에 영향을 받기 때문에 로컬시스템 자원이 가진 실제 정보와 차이가 있다. 따라서 작업에 대한 컴퓨팅 자원할당은 작업 총실행시간 예측에 대한 오차를 고려하여 할당해야 한다[6].

작업 총실행시간 $T_{estimated}$는 식(1)과 같이 구한다[5].

$$T_{estimated} = T_{network} + T_{computation} \quad \cdots\cdots\cdots\cdots (1)$$

식(1)에서 $T_{network}$는 작업을 전송하는데 소요되는 시간이고, $T_{computation}$은 할당된 자원에서 작업을 처리하는 시간이다. $T_{network}$는 식(2)와 같이 구한다[7].

$$T_{network} = \frac{W_{send\_data}}{P_{send\_throughputs}} + \frac{W_{recv\_data}}{P_{recv\_throughputs}} \quad \cdots\cdots (2)$$

식(2)에서 W_{send_data}와 $P_{send_throughputs}$은 송신시 작업 크기와 처리량이며, Wrecv_data와 $P_{recv_throughputs}$은 수신시 작업 크기와 처리량이다. $T_{computation}$은 식(3)과 같이 구한다[7].

$$T_{computation} = \frac{W_{logical\_computation\_cost}}{P_{performance\_of\_resource}} \quad \cdots\cdots (3)$$

식(3)에서 $W_{logical_computation_cost}$는 처리해야 할 명령어의 수이며, $P_{performance_of_resource}$는 로컬시스템의 성능을 의미한다.

작업의 총실행시간을 예측할 때 자원 데이터베이스의 정보와 네트워크 및 로컬시스템간의 실제 정보는 차이가 있다. 이러한 차이에 따른 오차를 고려하여 작업의 총실행시간을 보정해주면 좀더 정확한 작업의 총실행시간을 구할 수 있다.

2) 그리드 스케줄링 정책

지리적으로 분산된 컴퓨팅 환경인 그리드 환경을 위한 스케줄링 정책은 이기종 컴퓨팅 자원의 유동성과 활용을 고려해야 하기 때문에 기존의 분산 컴퓨팅을 위한 스케줄링 정책을 적용하는 것은 적합하지 않다. 따라서 그리드 환경을 위한 스케줄링 정책 개발을 위해 여러 가지 프로젝트가 진행되고 있다. 그리드 컴퓨팅을 위한 스케줄링 정책 개발 프로젝트는 표 2와 같다.

표 2. 그리드 컴퓨팅을 위한 스케줄링 정책
Table 2. Scheduling Strategy for Grid computing

프로젝트	스케줄링 방식
I-SOFT	중앙 스케줄러에 의한 집중형 작업 스케줄링
MARS	최소실행시간을 기준으로 한 작업 스케줄링
Prophet	실행시간예측을 통해 최소실행시간을 가지는 시스템 자원 선택
VDCE	태스크와 자원타입 매칭을 통한 적정 자원 할당 스케줄링
IOS	반복 실행에 의한 Genetic 스케줄링
AppLes	사용자 성능 분석에 의한 스케줄링 선택
SEA	프로그램 그래프 안에서 'ready task'기준 스케줄링
SPP(X)	여러 스케줄링 중에서 최소실행시간 기준 스케줄링 선택
Dome	글로벌 또는 로컬 로드밸런싱 위주 스케줄링

표 2의 여러 그리드 프로젝트에서 사용하고 있는 스케줄링 정책들은 주로 Greedy 및 최소실행시간을 기준으로 컴퓨팅 자원을 할당한다. Greedy 스케줄링 정책은 작업을 가장 빨리 처리할 수 있는 컴퓨팅 자원을 선택하고, 최소자원비용 스케줄링 정책은 자원비용을 우선적으로 고려하여 컴퓨팅 자원을 선택하고 작업을 할당한다. Greedy 스케줄링 정책은 작업 실패율이 가장 낮지만 반대로 자원비용이 높다. 최소자원비용 스케줄링 정책은 낮은 자원비용에 비해 실패율이 높고 로컬 컴퓨팅 자원간의 작업이주가 빈번하게 발생할 수 있다. 작업이주가 빈번하게 발생하면 네트워크 부하는 증가하고, 불필요한 자원 소모가 발생하는 문제점이 있다.

이러한 문제점을 해결하기 위해 본 논문에서는 작업 송신에 소요되는 네트워크 부하시간을 고려하여 컴퓨팅 자원을 선택함으로써 작업 실패에 따른 작업이주와 자원비용을 동시에 줄일 수 있는 스케줄링 정책을 제안한다.

III. 제안하는 스케줄링 정책

그리드 환경에서는 컴퓨팅 자원들 간의 특징이 다르고, 작업 데이터의 전송에 따른 컴퓨팅 자원 정보의 시간차가 존재하기 때문에 단일 스케줄링 방식보다 다단계 스케줄링 방식을 선호한다. 따라서 본 논문에서 제안하는 스케줄링 정책도 원격 스케줄러와 로컬 스케줄러를 적용한 2단계 스케줄링을 수행한다. 또한, 작업 전송에 대한 부하시간을 고려하여 작업의 데드라인과 작업 종료 시간과의 차가 최소인 자원을 선택하여 할당한다. 작업이주 발생 빈도를 줄임으로써 작업실패율을 줄이고, 컴퓨팅 자원의 활용도를 높여 자원비용을 줄인다.

3.1 원격 스케줄링 정책

원격 스케줄러는 자원 DB의 로컬시스템과 네트워크의 실시간 정보를 이용하여 작업실행시간 예측하고, 작업실행시간이 최소인 로컬 컴퓨터에 자원을 할당한다. 이 때 네트워크 전송시간에 따른 작업실행시간을 예측하는데 오차가 발생한다. 제안하는 원격 스케줄러에서는 작업실행시간 예측의 오류를 줄이기 위해, 작업 크기에 대한 전송시간 계산을 추가하여 전송시간을 재계산하고, 이 값을 적용하여 실제로 컴퓨팅 자원을 선택, 작업 데이터를 전송한다. 원격 스케줄링 알고리즘은 그림 2와 같다.

그림 2에서 데드라인까지 남아있는 유효시간 T_{target}은 $T_{deadline}$과 작업 요청시간 $T_{requested}$의 차로 구하고, n개의 로컬시스템을 대상으로 작업을 할당했을 때 예측되는 총실행시간 $T_{estimated}$을 구한다. 먼저 작업 전송시간 $T_{network_i}$와 실행시간 $T_{computation}$을 구한다. 작업 전송시간 $T_{network_i}$는 작업 크기에 따라 달라진다. 따라서 작업 크기와 평균 작업크기의 비율을 가중치로 구하여 $T_{network_i}$에 곱한다. 그리고 데드라인 $T_{deadline}$과 총 실행시간 $T_{estimated}$의 차인 $T_{resource_i}$를 구하고, n개의 로컬시스템 중에 최소의 $T_{resource_i}$를 가지는 로컬시스템을 선택하

여 할당한다. 이러한 할당정책을 사용하면 데드라인에 근접한 자원에 작업을 할당하여 처리함으로써 자원비용을 줄일 수 있다.

```
// T_target: 작업 데드라인까지의 유효시간
// T_requested: 사용자가 작업실행을 요청한 시간
// T_network_i: 작업을 자원 i와 송수신 소요시간
// T_computation_i: 자원 i에서 작업실행시간
// T_estimated_i: 예측한 총 실행시간
// T_resource_i: 데드라인과 총실행시간의 차이

T_target=T_deadline − T_requested;
for (i=1; i<=n; i++) // n은 자원의 개수
{
    //작업크기를 고려한 전송시간 가중치 추가 재계산
    T_network_i=T_network_i*(jobsize/averagesizeofjobs);
    T_estimated_i=T_network_i+T_computation_i;
    // 데드라인과 근접한 실행시간을 갖는 자원 선택
    T_resource_i =T_target − T_estimated_i;
    Min(T_resource_i);
}
Allocate(job, resource_i); //선택된 i번째 자원 할당
```

그림 2. 원격 스케줄링 알고리즘
Fig. 2. Remote Scheduling Algorithm

3.2 로컬 스케줄링 정책

로컬 시스템은 로컬 시스템의 부하 상태를 고려하여 작업을 할당한다. 원격 스케줄러에서 계산한 작업 총실행시간의 오차에 따른 작업이주 발생 빈도를 최소화 하기 위해 로컬 스케줄러에서 총실행시간을 재계산한다. 로컬 스케줄러에서는 로컬시스템의 대기 큐에서 대기한 시간을 총실행시간에 합산하여 총실행시간을 재계산한다. 또한 로컬 스케줄러에서는 재계산한 값에 따라 실행 또는 작업이주를 결정한다. 로컬 스케줄링 알고리즘은 그림 3과 같다.

그림 3에서 $T_{estimated}$는 원격 스케줄러에서 예측한 총실행시간과 로컬 시스템의 대기시간을 합산하여 구한다. 그리고 $T_{estimated}$가 $T_{deadline}$보다 작거나 같으면 해당 로컬시스템에서 작업을 처리한다. 하지만 $T_{estimated}$가 $T_{deadline}$보다 클 경우에는 작업을 다른 로컬 시스템으로 이주시킨다. 이때 나머지 n-1개의 다른 로컬 시스템에 대한 총 실행시간 $T_{estimated}$을 재계산한다. $T_{estimated}$을 재계산 할 때 다른 로컬 시스템으로 이주하는데 소요되는 네트워크 전송시간을 추가로 고려해야 한다. 네트워크 전송시간으로는 작업 송신시간 $T_{send_data_i}$와 수신시간 $T_{recv_data_i}$이다. 특히 $T_{send_data_i}$는 원격 스케줄러에서 현재 로컬시스템으로 작업을 전송하는 송신시간과 현재 로컬시스템에서 다른 로컬시스템으로 이주시키는 송신시간을 고려하여 $T_{send\_data\_i}$*2로 구한다. 그리고 원격 스케줄러와 같이 작업 크기에 대한 가중치를 고려하여 $T_{estimated}$를 구한다. $T_{resource_i}$는 $T_{deadline}$과 $T_{estimated}$을 고려하여 구한다. 마지막으로 만약 $T_{resource_i}$가 0 보다 크면 최소의 $T_{resource_i}$를 가진 로컬시스템을 선택하여 작업을 이주시킨다. 그러나 작업이 데드라인 내에 처리되지 않으면 실패한 작업으로 처리한다.

```
//T_estimated_i: 원격 스케줄러에서 예측된 총실행시간
//T_wait_in_queue: CPU 대기큐에서의 대기시간
//T_network_i: 작업을 자원 i에 주고받을 때의 전송시간
//T_send_data_i: 원격에서 자원 i에 작업을 전송하는 시간
//T_recv_data_i: 자원 i에서 결과를 원격으로 전송하는 시간
//T_computation_i: 자원 i에서 작업실행시간
//T_resource_i: 데드라인과 총 실행시간과의 차이
//작업에 대한 현 시점의 대기시간 추가
T_estimated_i = T_estimated_i + T_wait_in_queue;
if ( T_estimated_i <= T_deadline )
{
    Execute(job);
} else {
    For (i=1; i<n-1; i++) //네트워크 전송시간 재계산
    {
        T_network_i = ((T_send_data_i*2 + T_recv_data_i)
                * (job size / average size of jobs));
        T_estmated_i = T_network_i + T_computation_i;
        T_resource_i = T_deadline − T_estimated_i;
    }
    Min(job, resource_i); //다른 적정 자원 검색, 결정
    if (T_resource_i > 0)
        Migration(job, resource_i);
    else
        Exit(fail);   // 데드라인을 초과한 경우
}
```

그림 3. 로컬 스케줄링 알고리즘
Fig. 3. Local Scheduling Algorithm

로컬 시스템에서 작업을 일정시간 내에 처리할 수 없는 경우 작업을 다른 컴퓨팅 자원으로 재전송하는 작업이주가 발생한다. 작업 이주에 따른 컴퓨팅 자원의 소모와 네트워크 부하 증가는 전체적으로 자원 비용을 상승시킨다. 따라서 로컬 컴퓨팅 자원을 선택할 때, 작업 이주가 발생하지 않도록 정확하게 작업실행시간을 예측하는 것이 중요하다.

IV. 성능 평가

본 논문에서는 SimGrid[8] 툴 킷을 사용하여 제안한 원격 및 로컬 스케줄링 알고리즘의 성능을 구하고, 기존 Greedy 및 최소자원비용 스케줄링 정책과 성능을 비교한다.

4.1 시뮬레이션 환경

본 논문에서는 시뮬레이션을 통하여 제안하는 스케줄링 정책의 성능을 평가한다. 이기종 컴퓨팅 환경에서 분산 애플리케이션의 시뮬레이션을 위한 핵심적인 기능들을 제공하는

SimGrid 툴을 이용하여 그리드 환경을 가상으로 구현한다. 시뮬레이션을 위한 환경은 표 3과 같다.

표 3. 시뮬레이션 환경
Table 3. Simulation Environment

항 목		설정값
네트워크	WAN Bandwidth	100 ~ 1000
	LAN Bandwidth	10 ~ 100
	노드 지연시간	0 ~ 10
컴퓨팅 자원	CPU power(MIPS)	10 ~ 200
	원격 시스템 수	1
	로컬 시스템 수	4/8
작업	작업 크기	100/200/300
	작업 수	16/32/64/128/256
	작업 실행시간	1500/3000/10000

표 3에서 그리드 환경을 구성하는 네트워크 설정은 WAN, LAN으로 구별하고, 각각의 네트워크 대역폭은 임의로 정하여 실제로 데이터 전송에 따른 유동성을 고려한다. 또한 로컬 컴퓨팅 자원의 수 및 연산능력, 작업 크기 및 실행시간 설정 또한 난수 발생에 의한 임의로 정한다.

4.2 시뮬레이션 결과 분석

본 논문에서는 작업 총실행시간, 작업 실패율, 자원비용을 성능 평가의 척도로 사용한다. 작업 총실행시간은 모든 작업이 완료되는 시간을 의미하고, 작업 실패율은 원격 스케줄러에서 할당한 작업을 로컬 컴퓨팅 자원에서 작업 완료시간 내에 처리하지 못하는 작업 수를 의미한다. 자원 비용은 요청된 전체 작업 크기에 대한 로컬 컴퓨팅 자원의 사용량을 의미한다. 이러한 성능 평가 척도를 적용하여 본 논문에서 제안하는 스케줄링 정책과 기존의 Greedy 스케줄링 정책의 성능을 비교한다.

먼저 그리드 환경에서 네트워크 전송시간과 CPU의 작업수행시간 및 작업이주시간이 작업의 총실행시간에 어떠한 영향을 주는지 알아보기 위해 작업 수 증가에 따른 작업의 총실행시간을 측정하였다. 그 결과는 그림 4와 같다.

그림 4를 살펴보면 작업 크기 100, 200, 300에서 로컬 컴퓨팅 자원의 수가 4 또는 8 일 때에 관계없이 일정한 변화를 보였다. 즉 동일한 작업크기에서 로컬 컴퓨팅 자원 수의 변화는 작업의 총실행시간에 미치는 영향이 적다. 하지만 동일한 로컬 컴퓨팅 자원의 수에서는 작업 크기의 변화에 따라 작업의 총실행시간의 차이가 큼을 볼 수 있다. 또한 작업의 수가 증가할수록 작업의 총실행시간도 증가함을 알 수 있다. 따라서 작업의 총실행시간은 로컬 컴퓨팅의 자원 수보다는 작업 크기, 작업 수에 더 큰 영향을 받는다는 것을 알 수 있다.

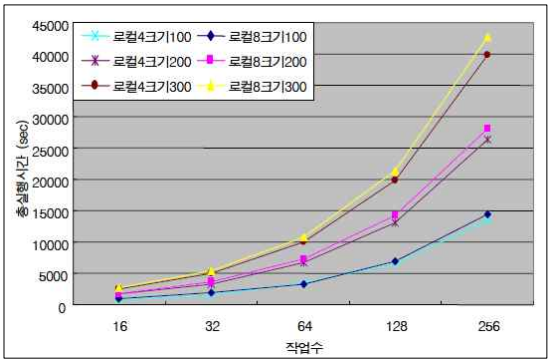

그림 4. 로컬 컴퓨팅 자원 수 변화에 따른 작업의 총실행시간
Fig. 4. The number of local computing resource vs. Total execution time of job

작업의 수를 증가시키면서 로컬 컴퓨팅 자원의 CPU 실행시간 1500, 10000에서 작업의 총실행시간을 측정한 결과는 그림5와 같다.

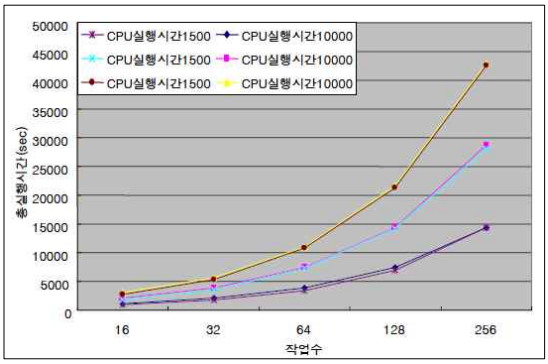

그림 5. CPU 실행시간에 따른 작업의 총실행시간
Fig. 5. CPU execution time vs. Total execution time of job

그림 5를 살펴보면 CPU 실행시간 1500, 10000에서 작업 총실행시간이 거의 유사함을 볼 수 있다. 즉, CPU 실행시간은 작업 총실행시간에 영향이 크지 않음을 알 수 있다.

그림 4와 그림 5의 결과를 분석해보면 작업의 총실행시간은 로컬 컴퓨팅 자원의 CPU 작업 수행시간보다 원격 스케줄러에서 로컬 컴퓨팅 자원으로 작업을 할당하는데 소요되는 네트워크 전송시간에 더 많은 영향을 받는 것을 알 수 있다.

작업 크기를 10000으로 고정하고 작업의 수를 증가시키면서 로컬 컴퓨팅 자원의 수가 4와 8일 때 Greedy 스케줄링 정책과 제안한 스케줄링 정책의 작업 실패율을 측정한 결과는

그림 6과 같다.

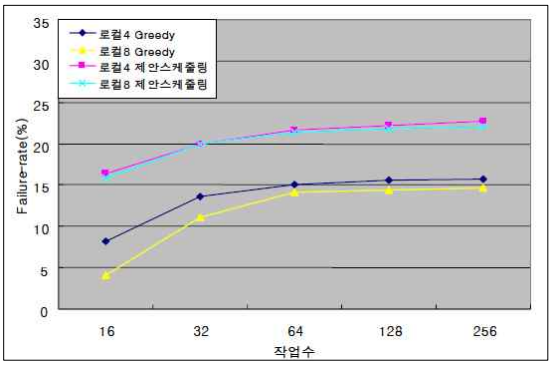

그림 6. 작업 실패율
Fig. 6. Job failure rate

그림 6을 살펴보면 작업 수 64 이상에서는 Greedy 스케줄링 정책과 제안 스케줄링 정책의 작업 실패율이 일정하게 유지됨을 알 수 있다. 또한, 로컬 컴퓨팅 자원의 수 4, 8에서 Greedy 스케줄링 정책의 작업 실패율이 제안한 스케줄링 정책보다 약 5~10% 정도 낮음을 볼 수 있다. Greedy 스케줄링 정책은 로컬 컴퓨팅 자원을 선택할 때 항상 높은 성능을 가지는 로컬 컴퓨팅 자원을 선택하기 때문에 로컬 컴퓨팅 자원의 수가 증가할수록 작업 실패율이 낮아진다. 하지만 고성능 로컬 컴퓨팅 자원으로 시스템을 구성하면 자원 비용이 증가하기 때문에 비효율적이다.

작업 크기를 10000으로 고정하고 작업의 수를 증가시키면서 로컬 컴퓨팅 자원의 수가 4와 8일 때 Greedy 스케줄링 정책과 제안한 스케줄링 정책의 자원 비용을 측정한 결과는 그림 7과 같다.

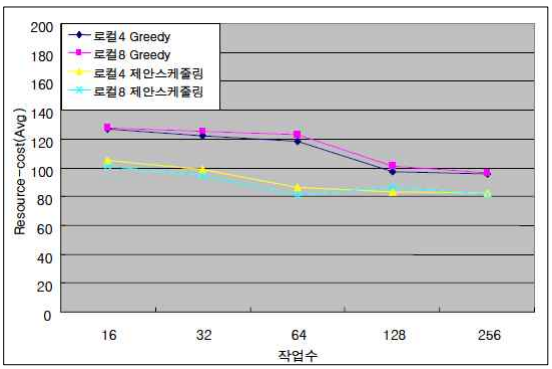

그림 7. 컴퓨팅 자원 비용
Fig. 7. Computing resource cost

그림 7을 살펴보면 로컬 컴퓨팅 자원의 수 4, 8에서 제안한 스케줄링 정책의 자원 비용이 Greedy 스케줄링 정책보다 평균 약 15~30% 정도 낮음을 볼 수 있다. 이것은 제안한 스케줄링 정책이 원격 스케줄러에서 예측한 작업 실행시간을 고려해서 로컬 컴퓨팅 자원을 선택하기 때문이다. 즉, 동일한 양의 작업을 수행하는데 사용되는 컴퓨팅 자원에 대한 자원 비용은 제안한 스케줄링 정책이 더 효율적이다. 방대한 데이터를 처리하는 그리드 환경을 구축하기 위해서는 컴퓨팅 자원에 대한 자원 비용을 고려해야 하기 때문에 제안한 스케줄링 정책이 더 적절하다.

V. 결론

본 논문에서는 그리드 환경을 구축할 때 자원 비용 면에서 효율적인 스케줄링 정책을 제안한다. 이 스케줄링 정책은 로컬 컴퓨팅 자원을 효율적으로 할당하기 위해 자원 비용과 작업 실패율을 고려한다. 이 스케줄링 정책의 특징은 원격 스케줄러와 로컬 스케줄러를 사용하여 2단계 스케줄링을 수행한다. 원격 스케줄러에서는 자원 데이터베이스에 저장된 네트워크와 로컬시스템의 정보를 사용하여 작업의 총실행시간이 최소인 로컬시스템을 선택하여 작업을 할당한다. 로컬 스케줄러에서는 할당된 작업의 대기시간과 처리시간을 재계산한 후, 작업을 데드라인 내에 처리할 수 있다면 로컬시스템에서 처리한다. 하지만 데드라인을 초과하면 다른 로컬시스템으로 이주시켜 처리함으로써 작업 실패율과 자원 비용을 최소화한다. 제안한 스케줄링 정책은 기존 Greedy 정책에 비해 작업 실패율은 높지만, 자원 비용을 줄이는 면에서 더 우수함을 보였다.

본 논문의 성능 평가 결과 Greedy 스케줄링 정책의 작업 실패율이 제안한 스케줄링 정책보다 약 5~10% 정도 우수하였다. 하지만 제안한 스케줄링 정책은 Greedy 스케줄링 정책에 비해 자원 비용이 평균 약 15~30% 정도 우수하였다. 따라서 그리드 환경을 구축할 때 지정된 시간 내에 작업을 처리하는 것이 중요하다면 작업 실패율에 비중을 두어야 하기 때문에 Greedy 스케줄링을 적용하는 것이 적합하다. 하지만 방대한 데이터 보관 및 공유, 데이터 접근 등이 중요하다면 컴퓨팅 자원의 비용에 비중을 두어야 하기 때문에 제안한 스케줄링을 적용하는 것이 적합하다.

향후 연구과제로는 작업 크기 외에 로컬 컴퓨팅 자원의 타입, 메모리 사용량, 운영체제 종류, 작업의 종류에 따른 우선순위 및 특정 자원요청 등에 대한 요소를 고려한 스케줄링 정책 개발이 필요하다.

참고문헌

[1] 최준영, 이원주, 전창호, "그리드 컴퓨팅에서 자원 할당을 위한 실시간 스케줄링 정책," 한국정보과학회 2004 봄 학술발표논문집(A), 제 31권, 제 1호, 85-87쪽, 2004년 4월.

[2] V. Subramini, R. Kettimuthu, and S. Srinivasan, S. Sadayappan, "Distributed job scheduling oncomputational Grids using multiple simultaneous requests," High Performance Distributed Computing, HPDC-11 Proceedings 11th IEEE International Symposium, pp. 359-366, July 2002.

[3] Lichen Zhang, "Scheduling algorithm for real-time applications in grid environment," Systems, Man and Cybernetics, IEEE International Conference, Vol. 5, pp. 6-9, Oct. 2002.

[4] Paul Ruth, Junghwan Rhee, Dongyan Xu, Rick Kennell, and Sebastien Goasguen, "Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure," In the 3rd IEEE International Conference on Autonomic Computing (ICAC'06), 2006.

[5] 류경후, 이원주, 전창호, "다중 그리드 사이트에서 어플리케이션 특성을 고려한 동적 작업 재배치 정책," 한국컴퓨터정보학회 논문지, 제 13권, 제 4호, 31-37쪽, 2008년 7월.

[6] Nerjes, G.; Muth, P.; Paterakis, M.; Romboyannakis, Y.; Triantafillou, P., Weikum, G.; "Scheduling strategies for mixed workloads in multimedia information servers," Research Issues In Data Engineering, 1998, Continuous-Media Databases and Applications. Proceedings. Eighth International Workshop on, pp. 121-128, 23-24 Feb. 1998.

[7] A. Takefusa, H. Casanova, and S. Matsuoka, F. Berman, "A study of deadline scheduling for client-server systems on the Computational Grid," High Performance Distributed Computing, 2001. Proceedings 10th IEEE International Symposium, pp. 406-415, Aug. 2001.

[8] 조지훈, 이원주, 전창호, "계산 그리드를 위한 효율적인 작업 스케줄링 정책," 한국정보과학회논문지: 컴퓨팅의 실제 및 레터, 제 14권, 제 8호, 753-757쪽, 2008년 11월.

저 자 소 개

최 준 영

2000 : 인하대학교 지리정보공학과 학사.
2004 : 한양대학교 컴퓨터공학과 석사.
현 재 : 삼성전자 정보통신사업부 연구원.
관심분야 : 병렬처리시스템, 모바일컴퓨팅, Grid 컴퓨팅

이 원 주

1989 : 한양대학교 전자계산학과 공학사.
1991 : 한양대학교 컴퓨터공학과 공학석사.
2004 : 한양대학교 컴퓨터공학과 공학박사
현 재 : 인하공업전문대학 컴퓨터정보과 부교수
관심분야 : 병렬처리시스템, 성능분석, Grid컴퓨팅, 클라우드컴퓨팅

전 창 호

1977 : 한양대학교 전자공학과 학사.
1982 : Cornell University 컴퓨터공학과 석사.
1986 : Cornell University 컴퓨터공학과 박사.
1977-1979 : 전자통신연구소 연구원.
현 재 : 한양대학교 전자컴퓨터공학부 교수.
관심분야 : 병렬처리시스템, 성능분석, Grid컴퓨팅, 클라우드컴퓨팅