韓國 컴퓨터情報學會 論文誌
第11卷 第5號, 2006. 11.

2006-11-5-1-10

계산 그리드 컴퓨팅에서의 자원 성능 측정을 통한 그리드 스케줄링 모델

박 다 혜*, 이 종 식*

Grid Scheduling Model with Resource Performance Measurement in Computational Grid Computing

Da Hye Park*, Jong Sik Lee*

요 약

그리드 컴퓨팅은 지리적으로 분산된 이기종의 자원들을 상호 연결하여 대용량의 컴퓨팅 문제들을 해하기 위해 개발되었다. 그리드 컴퓨팅은 다양한 자원들로 구성되어 있기 때문에 효율적이고 안정적인 작업 처리를 위해서는 자원 스케줄링 모델이 필요하다. 그래서 우리는 각 자원들의 성능을 측정하여 작업을 할당하는 자원 성능 측정 스케줄링 모델을 제안하였다. 우리는 자원 성능 측정 수식을 이용하여 자원들을 평가하였고, DEVS 시뮬레이션 모델링 환경에서 자원 성능 측정 스케줄링 모델을 실험하였다. 그리고 우리는 자원 성능 측정 스케줄링 모델의 효율성을 증명하기 위해 자원 성능 측정 스케줄링 모델의 실험 결과들을 기존 스케줄링 모델들과 비교하였다. 이 실험 결과들은 자원 성능 측정 스케줄링 모델이 자원 관리를 개선하고 안정적인 작업 처리를 보장해 줄 수 있음을 증명해 줄 수 있었다.

Abstract

Grid computing has been developed for resolving large-scaled computing problems through geographically distributed heterogeneous resources. In order to guarantee effective and reliable job processing, grid computing needs resource scheduling model. So, we propose a resource performance measurement scheduling model which allocates job to resources with resource performance measurement. We assess resources using resource performance measurement formula, and implement the resource performance measurement scheduling model in DEVS simulation modeling.

▸ Keyword : 계산 그리드 컴퓨팅(Computational Grid Computing), 그리드 스케줄링 모델(Grid Scheduling Model), 그리드 자원 성능 측정(Grid Resource Performance Measurement)

• 제1저자 : 박다혜
• 접수일 : 2006.10.12, 심사일 : 2006.10.24, 심사완료일 : 2006.11.18
* 인하대학교 컴퓨터 정보공학과
* 본 연구는 정보통신부 및 정보통신 연구진흥원의 대학 IT연구센터 육성・지원사업의 연구결과로 수행되었음.

Ⅰ. 서 론

빠른 컴퓨팅 기술의 발전으로 새로운 기술들이 등장하고 발전하고 있는 가운데 지리적으로 분산되어 있는 여러 컴퓨팅 자원들을 공유하여 많은 양의 데이터를 처리하는 기술인 그리드 컴퓨팅[1][2][3]이 등장하게 되었다. 즉, 그리드 컴퓨팅은 지리적으로 분산된 컴퓨팅 자원들을 공유하여 자원의 사용을 극대화한 차세대 컴퓨팅 기술을 의미한다. 기존의 클러스터 컴퓨팅이나 분산 컴퓨팅은 제한된 네트워크 상에서 소유한 자원의 이용이 가능했지만, 그리드 컴퓨팅은 자신이 소유하지 않은 지리적으로 분산된 자원들을 지정된 시간 동안 빌려서 사용할 수 있으며, 이러한 휴면 자원의 공유와 활용을 통해 하나의 슈퍼 컴퓨터급 가상 시스템을 구축함으로써 비용 절감과 대량의 데이터 처리가 가능해진다.

그리드 컴퓨팅에는 애플리케이션 특징에 따라 크게 4가지로 분류 할 수 있다. 데이터 그리드(Data Grid)는 대용량의 정보를 생산하는 장비, 대용량 DB및 고성능 컴퓨터를 사용자 중심으로 연결하는 그리드이고, 액세스 그리드(Access Grid)는 계산 그리드(Computational Grid)와 데이터 그리드에서 생성된 정보를 원거리의 연구자들이 공유할 수 있도록 협업 환경을 제공하는 그리드이다. 그리고 계산 그리드[4][5]는 많은 자원들을 연결하여 계산을 해결할 수 있는 그리드이며, 마지막으로 모바일 그리드(Mobile Grid)는 모바일이라는 이동성 개념이 그리드 컴퓨팅에 접목이 된 그리드이다.

이 논문에서는 특히 계산 그리드 컴퓨팅에서의 스케줄링 모델에 대해 제안하였다. 계산 그리드 컴퓨팅에서는 많은 자원들을 연결하여 작업을 처리하므로 작업 처리시 어떠한 자원에 작업을 할당하느냐에 따라 작업의 처리 능력은 큰 차이를 보일 수 있다. 특히 서로 다른 성능을 가진 자원들로 구성되어 있는 계산 그리드 컴퓨팅 상에서는 각 작업을 할당할 때 그 자원들의 성능에 따라 작업을 할당함으로써 큰 성능 차이를 보일 수 있다. 바로 우리는 계산 그리드 컴퓨팅을 구성하는 각각의 자원들의 성능을 측정하는 자원 성능 측정 수식을 제안하여 각 그리드 자원들의 성능을 측정하고, 그 측정된 결과를 통하여 작업을 할당하는 자원 성능 측정 스케줄링 모델을 제안하였다. 이 자원 성능 측정 스케줄링 모델을 통해 각 자원에 작업을 할당함으로써 효율적이고, 안정적인 작업 처리를 보장해 줌으로써 계산 그리드의 이용률을 높이고, 신뢰성을 보장해 줄 수 있다.

논문의 구성은 다음과 같다. 2장에서는 이 논문의 관련 연구를 설명하고, 3장에서는 우리가 제안한 계산 그리드 컴퓨팅에서의 각 자원들의 성능을 측정하는 자원 성능 측정 수식과 이 수식을 응용한 자원 성능 측정 스케줄링 모델을 소개한다. 그리고 4장에서는 다른 기존 스케줄링 모델들과 비교 실험을 통하여 우리가 제안한 자원 성능 측정 스케줄링 모델의 효율성을 증명하고, 마지막 5장에서는 결론을 맺는다.

Ⅱ. 관련 연구

2.1 그리드 스케줄링 모델

그리드 컴퓨팅은 분산된 이기종의 자원들로 구성되어 있기 때문에 각 자원들을 관리하고 작업을 할당하기 위해 그리드 스케줄링 모델[6][7][8]이 필요하다. 특히 그리드 컴퓨팅이 발전하면서 그리드 스케줄링 모델의 필요성이 증대되고 있는데 이는 그리드 사용자에게 안정적이고 빠른 작업 처리를 보장하기 위해서이다. 또한 스케줄링 모델은 시스템 내에서 사용 가능한 자원을 누구에게 얼마나 할당할지를 결정하는 일을 하기 때문에 다양한 성능을 가지고 있는 자원들로 구성되어 있는 그리드 컴퓨팅에서는 특히 중요한 역할을 하게 된다. 즉, 스케줄링 모델이 더 빠르고 안정적인 성능을 보이는 자원을 선택하여 작업을 처리함으로써 그리드 컴퓨팅의 전체 성능을 향상시킬 수 있는 것이다.

그리드 컴퓨팅에서는 그리드 사용자의 작업을 수행하기 위해 크게 정적 스케줄링 모델(Static Scheduling Model)과 동적 스케줄링 모델(Dynamic Scheduling Model)로 나눌 수 있다.

먼저 정적 스케줄링 모델은 작업의 개수와 자원의 상태 등이 고정되어 있을 때 사용하는 방법으로 작업을 처리하기 전에 모든 자원과 파라미터들을 요구한다. 자원의 성능을 처음 평가한 그대로 계속 작업을 할당하기 때문에 동적으로 변화하는 그리드 컴퓨팅 환경에서 효율적인 작업 처리를 보장하기가 어렵다는 단점이 있다.

동적 스케줄링 모델은 자원의 상태가 유동적일 때 사용하는 방식으로 스케줄러가 작업을 처리하는 중에도 즉시 자원의 상태 변화에 따라 작업을 유동적으로 할당할 수 있다는 장점이 있다. 그러나 동적 스케줄링 모델은 실시간으로 자원을 평가하여 작업을 할당하기 때문에 오버헤드가 증가할 수 있고, 구현이 복잡하다는 단점이 있다.

그러므로 우리는 이 두 스케줄링 모델들의 단점을 보완하여

그리드 컴퓨팅 환경에서 안정적이고 빠른 작업 처리를 보장해 줄 수 있는 자원 성능 측정 스케줄링 모델을 구현하였다.

2.2 그리드 자원 성능 평가 정책

계산 그리드 컴퓨팅 환경에서 분산된 자원들의 효율적인 활용을 위해서는 각 자원들의 성능을 평가하여 더 빠르고 안정적인 작업을 처리해 주는 것을 보장해 주어야한다. 즉, 자원 성능 평가를 통해 그리드 사용자에게 QoS(Quality of Service)[9]를 보장해 주고, 향상된 작업 처리를 제공하게 됨으로써 그리드 컴퓨팅의 이용률을 높일 수 있다. 그래서 최근들어 많은 그리드 컴퓨팅 연구에서는 실시간으로 자원의 성능이 변화하는 그리드 컴퓨팅 환경에 적합한 자원 성능 평가 방법들을 연구하고 있다.

그리드 자원들의 성능을 평가하는 정책들로 먼저 자원의 신뢰성을 측정하여 자원 성능을 평가하는 정책이 있다. 이 정책에서 중요한건 바로 자원의 신뢰성을 어떻게 평가할지를 결정하는 것인데, 대표적인 방법으로는 각 자원에 Credit을 부여하여 자원의 성능을 부여된 Credit을 이용하여 평가하는 Trust Mechannism[10]이 있다. 그러나 이 정책은 Credit을 부여받은 Grid Resource Provider가 자신이 부여받은 Credit을 속일 수 있는 문제가 발생할 수 있다.

또 다른 자원 성능 평가 정책으로는 각 자원들의 Status Data를 바탕으로 자원의 성능을 측정하는 방법[11]이 있다. 이 방법은 각 자원들이 받은 자원의 개수에 따라 자원을 세 가지 Status로 평가한다. 할당된 작업이 가장 적을 때는 Normal Status, 작업의 처리가 지연될 때는 Degradation Status, 그리고 자원에 작업이 많이 할당되어 작업 처리를 보장해 줄수 없는 Fail Status로 자원의 상태를 평가하고, 시간 흐름에 따라 자원이 상태에 머문 시간을 측정하고, 측정한 값을 이용하여 각 그리드 자원들의 성능을 평가한다. 그러나 이 정책에서는 그리드 유저로부터 발생하는 작업들이 다양할 수 있다는 그리드 컴퓨팅의 특징을 고려하지 않고 단순히 할당된 작업의 개수를 이용하여 상태를 측정하기 때문에 자원의 성능을 평가하는데 조금 비효율적일 수 있고, 그리드 스케줄러가 모든 자원들의 상태 정보를 저장하고 있기 때문에 오버헤드가 발생할 수 있다는 단점이 있다.

III. 자원 성능 측정 스케줄링 모델

이번 절에서는 계산 그리드 컴퓨팅상에 있는 각 자원들의 성능을 측정하는 자원 성능 측정 수식에 대해 소개하고, 이 자원 성능 측정 수식을 이용한 자원 성능 측정 스케줄링 모델의 구성에 대해 알아본다.

3.1 자원 성능 측정

우리는 계산 그리드에 있는 각각의 자원들의 성능을 측정하기 위해 그리드 컴퓨팅 환경의 특징을 이용하였다. 계산 그리드 컴퓨팅은 서로 다른 성능을 가진 자원들로 구성되어 있으며, 각 자원들에게 할당되는 작업의 크기도 서로 다양하게 존재한다. 바로 우리는 이 특징을 이용하여 계산 그리드 컴퓨팅에 있는 각 자원들의 성능을 측정하였다. 먼저 각 자원들에게 할당되는 작업의 크기는 크게 3 가지로 구성하였다. 각 작업은 Small Job, Medium Job, Large Job으로 나누어지며, Small Job은 세 가지 작업 중에서 가장 짧은 처리 시간을 요구하며, Medium Job은 Small Job보다는 긴 처리 시간을 요구하지만 Large Job보다는 짧은 처리 시간을 갖는다. 그리고 마지막으로 Large Job은 다른 작업들 중 가장 긴 처리 시간을 요구하는 작업을 의미한다. 우리는 이 세가지 작업들을 통해 각 자원의 *JAS*(*Job Allocation Status*)를 측정하는데, 그 측정식은 아래의 (3.1)과 같다.

$$JAS_i = x \times LS_i + y \times MS_i + z \times SS_i \quad \cdots\cdots (3.1)$$

$$(x > y > z)$$

JAS_i : i번째 자원의 작업 할당 상태 정보
x, y, z : 각 작업에 따른 변수, 처리 시간이 높을수록 그 수치는 높음
LS_i : i번째 자원에 할당된 Large Job의 개수
MS_i : i번째 자원에 할당된 Medium Job의 개수
SS_i : i번째 자원에 할당된 Small Job의 개수

우리는 (3.1)의 수식을 통하여 각 자원들에 할당된 작업의 상태를 측정하였고, 이 수식을 이용하여 각 자원들의 성능을 측정하였다. 각 자원들의 성능을 측정할 때에는 앞의 수식에서 측정한 할당된 작업의 상태 정보뿐만 아니라 각 자원들이 처리 가능한 작업 처리 한도를 고려하여야 한다. 즉, 계산 그리드 컴퓨팅상에 있는 자원들은 각각 자신이 처리 할 수 있는 작업의 개수가 정해져있다. 이를 작업 처리 한도라 하며 만약 적재된 작업의 개수와 자원이 처리 할 수 있는 작업의 개수가 같으면 더 이상 그 자원은 작업을 받을

수 없다. 바로 이것을 이용하여 각 자원들의 성능을 측정하는 식이 (3.2)의 측정식이다.

$$Resource\,Performance = \frac{JPL_i}{JAS_i + 1} \quad \cdots\cdots\cdots (3.2)$$

(3.2)의 수식에서 JPL_i은 i번째 자원이 할당 받을 수 있는 작업 처리 한도를 의미하는데, 자원들은 자신이 처리할 수 있는 작업의 개수가 한정적이기 때문에 작업을 할당 받는데 제한을 받을 수 있다. 즉, JPL_i은 바로 각 자원들이 현재 할당 받을 수 있는 작업 한도를 의미한다. JAS_i는 위의 (3.1) 식에서 계산한 i번째 자원의 작업 할당 정보를 의미한다. 기존 연구에서는 Random 방식이나 Round Robin 방식을 이용하여 작업을 할당하지만, 우리는 위의 (3.1)식을 이용하여 계산한 작업 할당 상태(JAS) 값과 작업 처리 한도(JPL)를 이용하여 그리드 자원의 성능을 측정하였다. 우리는 이 수식을 통하여 각 자원의 상태를 측정하였고, 이렇게 측정된 값 중 가장 높은 값을 가지는 자원에게 작업을 할당하여 처리하였다. 그리고 만약 그 측정된 값이 0일 때에는 그 자원은 작업 처리 한도에 도달한 것이므로 어떠한 작업도 받지 못하게 된다. 이 자원 성능 측정 수식을 사용하면 각 자원들의 성능을 측정하여 작업을 할당하는 것이 용이할 수 있다.

3.2 자원 성능 측정 스케줄링 모델

우리는 위의 절에서 설명한 자원 성능 측정 수식을 이용하여 계산 그리드 컴퓨팅에서 이용할 수 있는 스케줄링 모델을 구성하였다. 먼저 기본 구성은 다음과 같다.

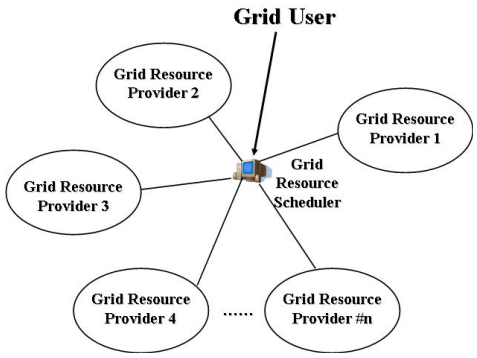


그림 1. 자원성능측정 스케줄링 모델의 기본구성
Fig 1. Basic Architecture of the Resource Performance Measurement Scheduling Model

그리드 유저(Grid User)가 일정한 간격으로 작업을 발생하여 그리드 자원 스케줄러(Grid Resource Scheduler)에게 작업을 보내면, 그리드 자원 스케줄러는 각각의 그리드 자원 제공자(Grid Resource Provider)들의 성능을 측정하여 성능이 더 높은 그리드 자원 제공자에게 작업을 할당한다. 그리드 유저는 세 가지 종류의 작업들을 그리드 자원 스케줄러에게 보내는데, 처리 시간에 따라 Small Job, Medium Job, Large Job으로 나눈다. 각 세가지 작업들이 어떻게 자원에 할당 되느냐에 따라 각 자원의 성능이 측정될 수 있다. 각 작업들이 자원에 할당되는 예는 아래 그림 2와 같다.

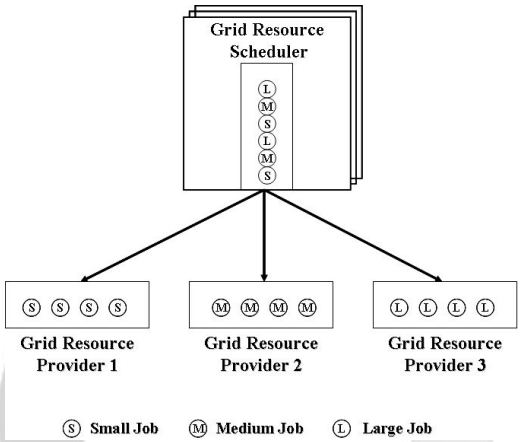


그림 2. 작업 할당의 예
Fig 2. Example of Job Allocation

그리드 자원 스케줄러는 그리드 유저로부터 받은 세 종류의 작업을 자신의 큐에 보관한다. 이 보관된 작업들은 각 자원들에게 할당되는데 위의 그림 2와 같이 세 개의 그리드 자원 제공자에게 다음과 같이 작업이 할당되어 있다면, 그 다음 작업은 가장 처리 시간이 적은 Small Job들만을 할당 받은 그리드 자원 제공자1에게 작업이 할당될 것이다. 즉, 각 자원에 어떤 종류의 작업이 할당되었느냐에 따라 그 자원의 성능은 달라지게 된다. 각 자원 제공자들의 상태를 측정하기 위해 우리는 위의 자원 성능 측정 수식을 사용하여 각 자원 제공자들의 성능을 측정한다. 위의 자원 성능 측정 수식을 이용한 자원 성능 측정 스케줄링 모델의 알고리즘은 다음과 같다.

```
j_large[i] = job_large_i;
j_medium[i] = job_medium_i;
j_small[i] = job_small_i;

//자원의 작업 상태 정보 측정
JAS[i] = x×j_large[i] + y×j_medium[i]
        + z×j_small[i];

//자원 성능 측정
RP[i] = JPL[i] × (1/(JAS[i]+1));
```

그림 3. 자원성능측정 스케줄링 모델의 알고리즘
Fig 3. Algorithm of the Resource Performance Measurement Scheduling Model

그림 3의 알고리즘에서 j_large, j_medium, 그리고 j_small은 각각 그 자원에 해당된 Large 작업, Medium 작업, 그리고 Small 작업의 개수를 의미한다. 이 값들은 해당 자원의 상태가 변경될 때마다 실시간으로 값을 입력을 받는다. 그리고 이 값들을 이용하여 작업 상태 정보[JAS]를 계산하고, 자원의 성능[RP]을 각 자원의 작업 상태 정보[JAS]와 작업 처리 한도[JPL]를 이용하여 계산하는데 여기서 작업 처리 한도는 자원들이 현재 할당 받을 수 있는 작업 한도를 수치화한 값이다. 이렇게 측정된 자원 성능 값이 가장 높은 자원에게 작업을 할당하여 작업을 처리한다. 이 알고리즘을 통해 자원 성능 측정 스케줄링 모델은 자주 상태가 변경되는 자원들의 성능을 실시간으로 측정하여 가장 높은 성능을 보이는 자원에게 작업을 할당함으로써 안정적이고 빠른 작업 처리를 그리드 사용자에게 제공할 수 있게 된다.

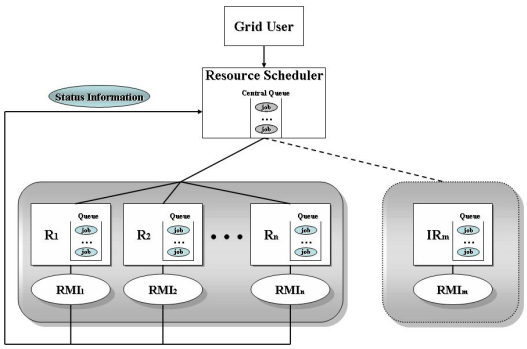


그림 4. 자원 성능 측정 스케줄링 모델의 구성
Fig 4. Architecture of the Resource Performance Measurement Scheduling Model

전체적인 자원 성능 측정 스케줄링 모델의 구성은 그림 4와 같이 크게 5개의 컴포넌트로 구성되며 각 컴포넌트의 기능은 다음과 같다.

그리드 유저(Grid User) : 그리드 유저는 일정한 간격으로 세가지 종류(Small job, Medium job, Large job)의 작업을 발생시키며, 발생된 작업은 자원 스케줄러에게 전달된다.

자원 스케줄러(Resource Scheduler) : 자원 스케줄러는 그리드 유저로부터 받은 작업을 자신의 큐(Central Queue)에 먼저 저장하고, 각각의 자원들의 성능을 측정하여 가장 높은 성능을 보이는 자원에 자신의 큐에 있는 작업을 보낸다. 자원 성능 측정과 작업 할당은 동시에 이루어져 할당에 따른 지연을 없앤다. 만약 모든 자원들이 적재된 작업의 개수가 자신의 작업 처리 한도에 도달하여 더 이상 어떠한 작업도 처리하지 못할 때에는 자원 스케줄러는 유휴 자원(IR)을 추가하여 자신의 큐에 있는 작업을 처리한다.

자원(R) : 자원 스케줄러로부터 받은 작업을 처리한다. 기본적으로 큐가 존재하며, 큐의 크기가 바로 자원의 작업 처리 한도이며, 각 자원들은 서로 다른 작업 처리 한도를 가진다. 그리고 자신의 상태가 변경될 때마다 해당 자원 측정기(RMI)에 자신의 상태를 전달한다.

자원 측정기(RMI) : 각 자원들에는 자신의 상태를 측정하는 자원 측정기가 존재한다. 자원 측정기는 해당 자원의 상태 변화를 측정하고, 만약 해당 자원의 상태가 변경되면 자원 스케줄러에게 변경된 자원 상태 정보를 전달한다.

유휴 자원(IR) : 만약 기존 자원들이 자신의 작업 처리 한도만큼 작업을 할당받아서 어떠한 작업도 할당 받을 수 없는 상태 일 때, 자원 스케줄러의 큐에 저장되어 있는 작업들을 처리하기 위해 추가하는 자원들이 바로 유휴 자원들이다. 유휴 자원은 기존 자원들과 같이 각각 자원 측정기를 가지고 있으며, 자신의 상태가 변경되면 자원 스케줄러에게 변경된 상태 정보를 전달한다.

자원 성능 측정 스케줄링 모델은 위와 같이 그리드 유저로부터 생성된 작업을 그리드 스케줄러가 자신의 큐에 먼저 저장한다. 그리고 실시간으로 자원들의 상태 정보를 받아서, 동적으로 각 자원들의 성능을 측정하여 가장 높은 성능을 가진 자원에게 작업을 할당한다. 그리고 만약 기존 자원들 모두가 할당 받은 작업들이 자신의 작업 처리 한도(자원의

큐 크기) 범위와 같게 되면, 자원 스케줄러는 자신의 큐에 있는 작업을 유휴 자원에 전달하게 된다. 이는 자원 성능 측정 스케줄링 모델의 가장 큰 단점이 될 수 있는 오버헤드를 방지 할 수 있다. 기존의 자원들이 모두 작업을 처리하지 못하는 상황에서 계속 그리드 유저가 자원 스케줄러에게 작업을 보내면, 이 작업들은 처리되지 못하고 계속 자원 스케줄러의 큐에 저장되어 지게 되어 오버헤드가 발생할 수 있다. 그러므로 상태 정보에 따라 유휴 자원을 추가하여 자원 스케줄러의 큐에 있는 작업들을 처리함으로써 이 문제를 해결할 수 있다.

Ⅳ. 실험 및 결과 분석

우리는 자원 성능 측정 스케줄링 모델의 효율성과 안정성을 증명하기 위해 DEVS 시뮬레이션 모델링[12]을 사용하였다. 그리고 자원 성능 측정 스케줄링 모델의 유용성을 증명하기 위해 기존의 스케줄링 모델인 Round-Robin 모델과 Random 모델을 추가로 구현하여 실험 결과를 비교하였다.

세 스케줄링 모델 모두 동일한 조건에서 실험을 하였다. 그리드 유저는 1초 간격으로 작업을 발생시키며, 작업의 종류는 총 3가지이다. 작업의 종류는 처리 시간에 따라 정해지는데, Small 작업은 처리 시간이 1초, Medium 작업은 처리 시간이 5초, 그리고 Large 작업의 처리 시간은 20초로 지정하였다. 그리고 각 작업을 처리하는 자원들은 총 5개로 구성하였으며, 각 자원들의 작업 처리 한도(큐 크기)는 서로 다르게 정하여 다양한 자원들로 실험을 구성하였다. 그리고 총 10000개의 작업을 발생시켜 각 모델들이 작업 처리시에 나타나는 Job Loss, Throughput, 그리고 Average Turn Around Time 값을 비교 측정하였다.

첫 번째 실험은 Average Turn Around Time을 측정하여 비교하였다. 그림 5는 Average Turn Around Time을 비교한 결과를 보여준다. 실험 결과를 통해 자원 성능 측정 스케줄링 모델이 다른 두 스케줄링 모델에 비해 낮은 Average Turn Around Time을 기록하는 것을 볼 수 있다. Random 스케줄링 모델과 Round Robin 스케줄링 모델이 각각 143과 151의 Average Turn Around Time을 기록한 반면 자원 성능 측정 스케줄링 모델은 141의 Average Turn Around Time을 기록하여 다른 두 모델들보다 좀더 빠르게 작업을 처리하고 있음을 보여준다. 우리가 제안한 자원 성능 측정 스케줄링 모델이 다른 기존의 두 모델들과 Average Turn Around Time의 값이 많이 차이나지 않는 이유는 자원 성능 측정 스케줄링 모델은 자원 스케줄러에 큐를 구현하여 작업을 보관하기 때문이다. 그러나 이 실험 결과는 우리가 제안한 스케줄링 모델은 다른 두 모델들보다는 계산 그리드 컴퓨팅상에서 빠른 작업 처리를 보장해 줄 수 있음을 보여준다.

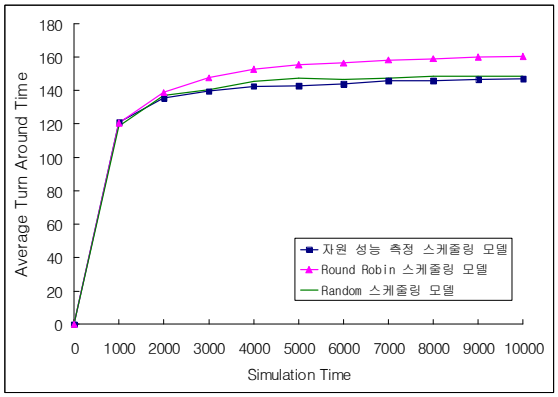


그림 5. 스케줄링 모델들의 Average Turn Around Time
Fig 5. Average Turn Around Time of the Scheduling Models

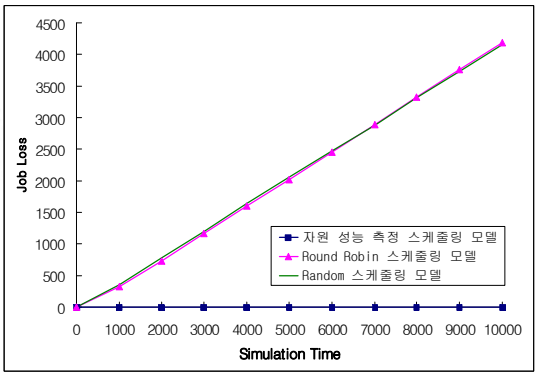


그림 6. 스케줄링 모델들의 Job Loss
Fig 6. Job Loss of the Scheduling Models

두 번째 실험은 각 스케줄링 모델들의 Job Loss를 측정하였고, 그 결과는 그림 6에 보여진다. Job Loss가 낮을수록 안정적인 작업 처리를 보장해 준다고 할 수 있는데 이 실험 결과를 보면 Random 스케줄링 모델과 Round Robin 스케줄링 모델이 각각 평균 40.4%와 39.6%의 높은 Job Loss율을 기록한 반면 우리가 제안한 자원 성능 측정 스케줄링 모델이 가장 낮은 0%의 Job Loss율을 기록하고 있음을 볼 수 있다. 자원 성능 측정 스케줄링 모델은 자원 스케줄러에 중앙 큐가 존재하여 자원들이 작업을 처리하는 것이 불가능 할 때(할당받은 작업의 개수와 작업 처리

한도가 같은 때)에는 큐에 보관하고 작업을 보내지 않기 때문에 0%의 Job Loss율을 기록할 수 있다. 그리고 자원 스케줄러의 중앙 큐의 오버헤드를 방지하기 위해 모든 기존 자원들이 작업 처리가 불가능 할 때에는 유휴 자원을 추가하여 작업을 처리하여 이 문제를 해결할 수 있다. 이 결과는 우리가 제안한 스케줄링 모델이 다양한 자원들이 존재하는 계산 그리드 컴퓨팅상에서 안정적으로 작업을 처리할 수 있는 자원을 선택하여 작업을 할당함으로써 계산 그리드 컴퓨팅의 신뢰성을 보장해 줄 수 있다.

세 번째 실험은 세 모델의 Throughput을 비교 측정하였고, 그 결과는 그림 7과 같다. 이 결과는 Round Robin 스케줄링 모델은 평균 57.6%의 Throughput을 Random 스케줄링 모델은 평균 57%의 Throughput을 기록하였지만, 자원 성능 측정 스케줄링 모델은 평균 78.6%의 Throughput을 기록하였다.

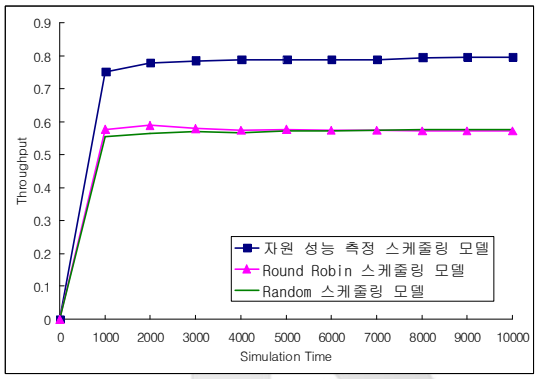


그림 7. 스케줄링 모델들의 Throughput
Fig.7 Throughput of the Scheduling Models

이 결과는 우리가 제안한 자원 성능 측정 스케줄링 모델이 다양한 자원들이 존재하는 계산 그리드 컴퓨팅 상에 있는 자원들을 효과적으로 선택하여 빠른 작업처리를 처리할 수 있음을 증명하는 결과이다.그러므로 자원 성능 측정 스케줄링 모델은 계산 그리드 컴퓨팅 환경에서 효율적이고 빠른 작업 처리를 보장해 줄 수 있다.

이 세 가지 실험을 통해 우리는 기존의 스케줄링 모델들(Random 모델, Round-Robin 모델)에 비해 자원 성능 측정 스케줄링 모델이 다양한 자원들이 존재하는 계산 그리드 컴퓨팅 환경에서 빠르고 안정적인 작업 처리를 보장해 줄 수 있음을 증명하였다.

Ⅵ. 결론 및 향후 과제

계산 그리드 컴퓨팅은 다양한 이기종의 자원들을 이용하여 슈퍼 컴퓨팅으로 처리해야 할 대용량의 작업들을 처리하는 새로운 컴퓨팅 기술이다. 이런 다양한 자원들이 존재하는 계산 그리드 컴퓨팅 환경에서 효율적이고 안정적인 작업 처리를 위해 필요한 것이 바로 스케줄링 모델이다. 분산된 다양한 자원들의 성능을 측정하여 효과적으로 작업을 할당해 줌으로써 빠른 작업 처리를 보장해주는 기능을 바로 스케줄링 모델이 담당하는데, 바로 우리는 각 자원들의 성능을 측정하여 작업을 할당하는 자원 성능 측정 스케줄링 모델을 제안하였다. 자원 성능 측정 스케줄링 모델은 다양한 자원들이 존재하고 다양한 작업들이 존재하는 계산 그리드 컴퓨팅의 특징을 이용하여 각 자원들의 성능을 측정하였고, 가장 높은 값을 가지는 자원에게 작업을 할당하여 작업을 처리하였다. 그리고 자원 성능 측정 스케줄링 모델에서 자원 스케줄러의 큐에 작업을 보관할 때 발생할 수 있는 오버헤드를 줄이기 위한 방법으로 유휴 자원을 추가하여 큐에 보관된 작업들을 처리하였다. 이렇게 유휴 자원을 추가함으로써 기존 자원들이 작업을 처리하지 못하는 경우 자원 스케줄러에 쌓이는 작업들을 처리하여 자원 성능 측정 스케줄링 모델에서 발생할 수 있는 오버헤드의 문제점을 보완할 수 있었다.

우리는 자원 성능 측정 스케줄링 모델의 유용성과 효율성을 증명하기 위해 DEVS 모델링 환경에서 실험을 하였고, 실험 결과들을 기존 스케줄링 모델들(Random 모델, Round-Robin 모델)과 총 3개의 파라미터들(Job Loss, Throughput, 그리고 Average Turn Around Time)의 결과를 비교하였다. 이 비교 실험을 통해 자원 성능 측정 스케줄링 모델이 기존 스케줄링 모델들에 비해 계산 그리드 컴퓨팅상에서 안정적이고 빠른 작업 처리를 보장해 줄 수 있음을 증명하였다. 이것은 자원 성능 측정 스케줄링 모델이 계산 그리드 컴퓨팅 환경에서 작업을 할당하고 처리하는데 매우 안정적이고 효율적이라는 것을 증명해주는 결과이다.

향후 우리가 제안한 자원 성능 측정 스케줄링 모델을 이 논문에서 비교했던 Random 모델과 Round-Robin 모델 이외에 다른 그리드 스케줄링 모델들과 그 결과를 비교 실험해 볼 예정이며, 자원 성능 측정 스케줄링 모델에서 발생하는 오버헤드의 문제를 해결할 수 있는 다른 방법을 연구할 예정이다.

참고문헌

[1] Foster, I. (2002), "What is the Grid? A three point checklist," Argonne National Lab and University of Chicago, http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf

[2] F. Berman, G. Fox and T. (2003), Hey, "Grid computing: making the globalinfrastructure a reality," J. Wiley. New York

[3] Ian Foster, Carl Kesselman. (2003), "The Grid 2: Blueprint for a New Computing Infrastructure," Morgan Kaufmann Publisher

[4] Xiu-Chuan Wu, Liang Hu, Jiu-Bin Ju. (2003), "ACGWPRS: the active computational grid framework," Machine Learning and Cybernetics, 2003 International Conference on Volume 2, pp. 994-999

[5] Li, C., Xiao, N., Yang, X. (2003), "Application availability measurement in computational grid," Proceedings of the 2nd workshop on Grid and Cooperative Computing (GCC2003), Springer LNCS 3032 pp. 151-154

[6] Shaohua Zhang, Ning Gu, Saihan Li. (2004), "Grid workflow based on dynamic modeling and scheduling," Information Technology: Coding and Computing. Proceedings of 2004 International Conference on Volume 2, pp. 35-39

[7] Shun-Li Ding, Jing-Bo Yuan, Jiu-Bin Ju.(2004), "An algorithm for agent-based task scheduling in grid environments." Machine Learning and Cybernetics, Proceedings of 2004 International Conference on Volume 5, pp. 2809-2814

[8] Junzhou Luo, Peng Ji, Xiaozhi Wang, Ye Zhu, Feng Li, Teng Ma, Xiaopeng Wang. (2004), "Resource management and task scheduling in grid computing,"Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on Volume 2, pp. 431-436

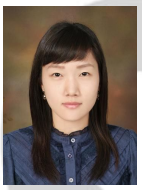
[9] Foster, I., et al. (2004), "End-to-End Quality of Service for High-End Applications," Computer Communications pp. 1375-1388

[10] Junzhou Luo, Peng Ji, Xiaozhi Wang, and Ye Zhu. (2005), "A Novel Method of QoS Based Resource Management and Trust Based Task Scheduling," CSCWD 2004, LNCS 3168 pp. 21-32

[11] Chunjiang Li, Nong Xiao, and Xuejun Yang. (2004), "Predicting the Reliability of Resources in Computational Grid," Grid and Cooperative Computing 2004, LNCS 3251, pp. 233-240

[12] Zeigler, B.P., et al. (1997), "The DEVS Environment for High-Performance Modeling and Simulation," IEEE CS & E, Vol. 4, No3 pp. 61-71

저 자 소 개



박 다 혜

2005 인하대학교 컴퓨터공학부 학사

2005~현재 인하대학교 컴퓨터정보공학과 석사과정

관심분야 : 시스템 모델링 및 시뮬레이션, 그리드 컴퓨팅

이 종 식

1993 인하대학교 전자공학과 학사

1995 인하대학교 전자공학과 석사

2001 애리조나대 컴퓨터공학과 박사

2001~2002 캘리포니아 주립대학교 전기·컴퓨터공학과 전임강사

2002~2003 클리블랜드 주립대학교 전기·컴퓨터공학과 조교수

2003~2006. 8 인하대학교 컴퓨터공학부 조교수

2006. 9~현재 인하대학교 컴퓨터공학부 부교수

관심분야: 소프트웨어공학, 시스템 모델링 및 시뮬레이션, 그리드 컴퓨팅