계산 그리드를 위한 서비스 예측 기반의 작업 스케줄링 모델

장성호*, 이종식*

Service Prediction-Based Job Scheduling Model for Computational Grid

Sung-Ho Jang, Jong-Sik Lee

Abstract

Grid computing is widely applicable to various fields of industry including process control and manufacturing, military command and control, transportation management, and so on. In a viewpoint of application area, grid computing can be classified to three aspects that are computational grid, data grid and access grid. This paper focuses on computational grid which handles complex and large-scale computing problems. Computational grid is characterized by system dynamics which handles a variety of processors and jobs on continuous time. To solve problems of system complexity and reliability due to complex system dynamics, computational grid needs scheduling policies that allocate various jobs to proper processors and decide processing orders of allocated jobs. This paper proposes a service prediction-based job scheduling model and present its scheduling algorithm that is applicable for computational grid. The service prediction-based job scheduling model can minimize overall system execution time since the model predicts the next processing time of each processing component and distributes a job to a processing component with minimum processing time. This paper implements the job scheduling model on the DEVS modeling and simulation environment and evaluates its efficiency and reliability. Empirical results, which are compared to conventional scheduling policies, show the usefulness of service prediction-based job scheduling.

Key Words : Computational Grid, Prediction-based Job Scheduling, DEVS Modeling and Simulation

* 인하대학교 컴퓨터공학부

1. 서론

최근 인터넷의 보편화와 컴퓨터 네트워크의 발달로 인하여 클러스터를 대체하는 차세대 기술로서 그리드 컴퓨팅이 각광받고 있다[1][2]. 그리드란 단순 자료 공유만을 하는 분산 컴퓨팅 개념과는 달리 지리적으로 분산된 컴퓨터, 애플리케이션, 대용량 저장장치 등의 고성능 자원들을 네트워크로 상호 연동하여 사용자가 단일 시스템처럼 모든 그리드 자원들을 쉽게 사용할 수 있도록 하는 정보통신 인프라를 말한다[3][4]. 그리드는 크게 계산 그리드(Computational Grid), 데이터 그리드(Data Grid), 접근 그리드(Access Grid)로 분류된다[5]. 시간이 갈수록 기존의 클러스터 시스템이나 슈퍼컴퓨터로 해결할 수 없는 대용량의 복잡한 연산 집약적인 문제들이 늘어감에 따라 계산 그리드에 대한 연구와 개발이 한창 진행되고 있으며 현재 과학 연구, 금융, 물리, 행정 및 기계 항공등의 산업 전 분야 걸쳐 적용되고 있다. 계산 그리드는 고속, 고성능 연산을 요구하는 컴퓨팅을 위하여 그리드 상의 사용가능한 많은 자원들을 연결하여 서비스 실행 시간의 최소화를 목표로 한다.

계산 그리드의 핵심은 바로 계산속도 향상과 처리량 분산을 위한 효과적 스케줄링 기술이다[6][7]. 이는 전체 시스템 성능의 최적화 문제와 직결된다. 그러므로, 계산속도 향상을 위해서는 언제 어떠한 프로세서에 얼마만큼의 작업을 할당하고 작업들의 처리 순서를 결정하여 처리량을 분산시키기 위한 스케줄링 기법이 필수적이다. 계산을 위해 사용되는 그리드 컴퓨팅 자원은 지리적, 조직적으로 독립적인 객체들로 구성되어 있고 개개의 관리 정책을 가지고 있다. 또한, 계산 그리드에서는 그리드에 존재하는 프로세서들의 수와 계산 처리를 필요로 하는 서비스의 수가 거의 무한대에 이른다. 이로 인해 그리드는 매우 동적인 환경 특성을 가지게 된다. 이러한 환경 특성으로 인해 그리드의 작업 할당과 스케줄링은 매우 어렵고 복잡하다.

본 논문에서는 계산 그리드에서 적용 가능한 서비스 예측 기반의 작업 스케줄링 모델과 실행 시간 예측 알고리즘을 제시한다. 작업 스케줄링 모델은 서비스 유형을 예측하여 프로세서가 제공하는 경험적 데이터를 이용한 실행시간 측정 및 예상 실행시간 계산과 그룹화 과정을 통해 적합한 프로세서에 작업을 할당하여 전체 시스템 성능을 향상시킨다. 우리는 DEVS 모델링 & 시뮬레이션 환경에서 시뮬레이션 모델을 설계하고 랜덤 스케줄링 모델과 라운드로빈 스케줄링 모델과의 비교를 통하여 모델의 성능 평가를 실시한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 배경이 되는 스케줄링 기법과 그리드 스케줄러에 대해 알아보고 3장에서는 서비스 예측 기반의 작업 스케줄링 모델과 실행 시간 예측 알고리즘을 제시한다. 4장에서는 시뮬레이션의 결과 분석을 통하여 모델의 효율성과 효과를 입증하고 마지막으로 5장에서는 결론을 맺는다.

2. 연구배경

2.1 스케줄링 기법

그리드 환경에서는 많은 수의 프로세서들이 존재하여 기존의 시스템으로 해결할 수 없는 대용량의 복잡한 문제들을 처리한다. 따라서, 그리드 스케줄링을 위해서는 다음과 같은 문제점들을 고려해야 한다[8]. 그리드 컴퓨팅 자원들의 이종성을 극복해야 하고, 시스템 성능을 향상시켜야 한다. 또한, 다양한 애플리케이션을 지원해야 하고 자원 활용도를 높여야 한다. 이런 문제를 해결하기 위한 그리드 스케줄링 기법에 관한 연구는 작업 할당의 결정 시점에 따라 정적 스케줄링(Static job scheduling)과 동적 스케줄링(Dynamic job scheduling)으로 구분된다[9].

정적 스케줄링은 컴파일 시간동안 스케줄러

역할을 맡은 컴파일러가 자원 할당을 결정한다. 즉, 주어진 자원들과 작업들을 이용하여 컴파일러가 실행시간 전에 예상 실행 스케줄을 계산한다. 이 방식은 부하의 이동에 따른 코드 변환이 불필요하며 동기화와 오버헤드가 적다는 장점을 가지고 있지만 실행시간에 앞서 시스템의 모든 자원 정보와 성능 파라미터들에 대한 정보가 알려져야 하고 프로세서의 활용도가 낮다는 단점을 가지고 있다. 또한, 예외상황을 처리할 대처능력이 부족하므로 컴파일 시에 예측한 스케줄이 실행 시에도 항상 유지되어야 하는 단점이 있다.

동적 스케줄링은 실행 시간동안 시스템 부하를 유지하기 위해 프로세서가 직접 작업을 할당하고 처리하는 방식이다. 동적 스케줄링은 실시간으로 변화하는 그리드 환경의 변화에 맞춰 즉각적인 대처가 가능하고 부하의 불균형을 최소화하지만 통신상의 오버헤드가 증가한다는 단점이 있다. 동적 스케줄링은 스케줄링 정보의 저장위치와 스케줄링의 실행 장소에 따라 중앙 스케줄링(centralized scheduling)과 분산 스케줄링(distributed scheduling)로 분류된다[10][11]. 중앙 스케줄링은 중앙에 스케줄러 역할을 담당하는 프로세서를 선정하여 전체 정보를 저장하고 시스템을 구성하는 다른 프로세서들이 작업을 전송하거나 수신하기 위해 스케줄러에 접속하는 방식이다. 이 방식은 간단한 구조와 편리한 유지보수성, 불가결성을 제공하지만 스케줄러에 병목(bottleneck)현상이 발생할 가능성이 매우 크다. 분산 스케줄링은 스케줄러 없이 각 프로세서들이 스케줄링 작업과 정보 저장을 담당하는 방식이다. 이 방식은 공유된 전체 큐에 접근하여 휴면상태의 프로세서가 실행시간동안 작업을 할당받는 것을 허가하고 동기화 메커니즘을 통해 큐의 접근을 하나의 프로세서에만 허락한다. 또한, 작업 부하의 분산으로 인해 시스템이 강화되고 예외처리 능력이 뛰어나지만 오버헤드가 자주 발생한다.

본 논문에서 제안하는 서비스 예측 기반의 작업 스케줄링 모델은 정적 스케줄링의 경험적 데이터 분석을 적용하여 프로세서의 실행시간을 예측하고 실행시간동안 프로세서에 작업을 할당하는 동적 스케줄링의 실시간 특성을 제공함으로서 시스템의 유연성을 보장한다. 작업 스케줄링 모델은 중앙 스케줄링 방식과 같이 중앙에 스케줄러가 존재하지만 스케줄러의 작업부하를 줄이기 위해 실행시간 예측과 정보 저장은 프로세서들이 담당한다.

2.2 그리드 스케줄러

그리드 컴퓨팅을 위해 많은 스케줄러들이 개발되었고 현재 그리드 오퍼레이터들에 의해 다양한 범위에서 이용되고 있다. 채택되어 사용되고 있는 대표적인 스케줄러에는 Nimrod-G, GrADS, Condor-G가 있다.

Nimrod-G[12]는 최근 그리드 메타 스케줄러(Grid meta-scheduler)로서 각광을 받고 있으며 GRACE(GRid Architecture of Computa-tional Economy) 프로젝트에서 경제 기반 스케줄러로서 이용되고 있다. GrADS [13]은 계산 그리드환경에서 응용프로그램을 쉽게 개발할 수 있는 환경 구축을 목적으로 하며 가용 자원의 변화에 따라 응용 프로그램을 적응시킨다. Condor-G[14]는 계산 그리드의 초입부에 위치하는 작업 브로커이며 사용가능한 노드에서 작업을 실행시키기 위한 그리드의 진입점(entry point)으로서 동작한다.

위의 스케줄러들은 모두 다음과 같은 특징을 가지고 있다. 전체 그리드 노드 상의 작업에 대한 모든 정보와 노드별 스케줄링 정책에 관한 정보를 완벽히 인식하고 있어야 하고, "top down" 방식의 중앙 스케줄링 기법을 채택하고 있다는 점이다. 그러나, 작업 정보를 완벽히 인식한다는 것은 현실적으로 불가능하고 스케줄러의 병목현상으로 인해 효과적 자원 분배를 제공할 수 없으므로 대용량 그리드의 특성에 적합하지 않다. 본 논문에서 제안하는 스케줄링 모델은 전체 작업 정보가 아닌

약간의 작업 정보만으로도 스케줄링이 가능하며 스케줄링 작업의 분산과 계층적 구조를 통해 병목현상의 발생 가능성을 최소화함으로서 광범위하고 다양한 자원을 포함하는 계산 그리드의 특성에 알맞은 스케줄러를 제공한다.

3. 서비스 예측 기반의 작업 스케줄링 모델

우리는 그리드의 스케줄링 문제점 해결을 위해 서비스 예측 기반의 작업 스케줄링 모델을 제시한다. 서비스 예측 기반의 작업 스케줄링 모델은 네트워크상에서 발생 가능한 작업 손실(job loss)을 최소화하고 작업 처리량(throughput)과 시스템 내의 프로세서 활용도(utilization)를 최대화함으로서 전체 시스템 실행시간을 단축시키는 것을 목표로 한다. 빠른 실행시간을 위해서는 각각의 프로세서의 성능에 맞는 크기의 작업을 할당할 수 있는 방법이 필요하다. 본 모델에서는 시스템의 예상 실행시간을 예측하여 작업을 할당함으로서 이를 해결한다. 시스템의 실행시간은 시스템 용량, 프로세서의 활용도, 작업의 크기, 서비스의 복잡도 등에 영향을 받는다. 그리드에 포함된 프로세서의 성능이 모두 동일하다면 스케줄링에 아무런 어려움이 없지만 현실적으로 프로세서들의 성능은 매우 다양하므로 실제 실행시간의 정확한 예측은 아주 어렵고 복잡하다. 작업 크기의 다양성과 서비스의 복잡도 역시 실행시간의 예측을 더욱 어렵게 만드는 요소이다. 우리는 이러한 예측 방해 요소를 간소화하기 위해 서비스의 유형을 파악하여 적합한 프로세서에 전달하고 실행시간 예측 알고리즘을 제공한다. 또한, 모델에서는 일정 시간동안 시스템이 동일한 서비스를 실행하였던 경험을 데이터베이스화하여 실제 서비스가 시스템에 입력되어 실행되기 전 축적된 데이터베이스를 토대로 시스템의 다음 서비스 실행 시간을 계산하여 예측한다. 데이터가 축적될수록 시스템의 예측시간 계산은 더욱 더 정밀해지고 오차 또한 줄일 수 있다. 목표인 시스템 성능의 최적화를 위해서는 주어진 시간 안에 최대한 많은 작업을 처리해야 한다. 즉, 모델은 효율적 스케줄링을 위해 단위 시간 당 작업 처리량의 최대화와 작업 손실의 최소화를 목표로 한다.

```
if (not exist T_expectedtime[1] )
{
   predict_T = ((1-alpha) * T_expectedtime[0])+(alpha *T_realtime[0]);
   T_expectedtime[1] = predict_T;
}
else if (exist T_expectedtime[1] )
{
   for (i=1;i<n-1;i++)
   {
      expect[i] = T_expectedtime[i]/ T_expectedtime[i-1];
      real[i] = T_realtime[i]/ T_realtime[i-1];
      predict_T = ((1-alpha) * expect[i])+(alpha * real[i] ));
      total_pre_T * = predict_T;
   }
   T_expectedtime[n] = total_pre_T * T_expectedtime[1];
}
```

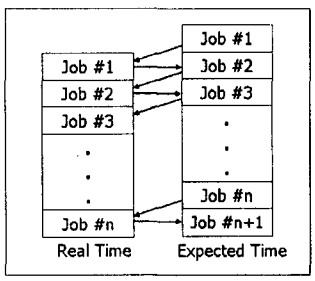
<그림 1> 실행시간 예측 알고리즘

<그림 1>은 실행시간 예측 알고리즘[15]을 의사코드어로 표현한 것이다. predict_T는 기준 값이 되는 시스템 상에 실행되는 서비스의 최초 예상 실행시간이다. T_expectedtime []는 예상 실행시간을 축적하는 배열 요소이고 T_realtime[]는 실제 실행시간을 축적하는 배열 요소이다. 이 두 배열 요소는 일정 시간동안 시스템이 동일한 서비스를 실행하였던 경험의 데이터베이스를 나타낸다. alpha는 예상 실행시간의 오차를 최소화하기 위한 학습 비율을 나타낸다. 우리는 alpha값을 변화시키고 <표 1>과 같이 예상 실행시간의 평균 오차 변화를 측정하여 최소 오차율을 제공하는 0.99라는 값을 alpha로 채택하였다.

<표 1> alpha값에 따른 예상시간 평균오차

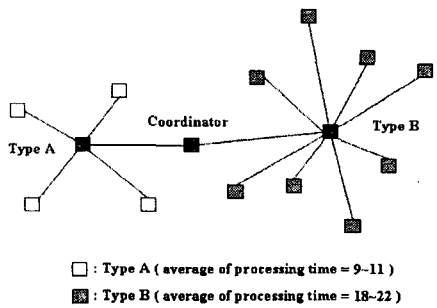
alpha	error(%)	alpha	error(%)
0.9	13.58	0.95	7.65
0.91	12.64	0.96	6.46
0.92	11.6	0.97	5.08
0.93	10.22	0.98	3.41
0.94	9.35	0.99	2.32

시스템에 서비스 작업이 입력되기 전 계산을 통해 얻어진 predict_T는 첫 번째 예상 실행시간을 나타내는 T_expectedtime[1]에 저장된다. 첫 번째 예상 실행시간을 얻은 후 서비스 작업이 시스템 상의 프로세서에서 동작 중일 때 동일한 서비스 작업이 입력되는 경우가 발생한다. 이 경우 첫 번째 예상 실행시간처럼 두 번째 작업의 예상 실행시간을 계산할 수는 있지만, 동적인 시스템 부하로 인해 정확한 예상 실행시간을 계산하는 것은 매우 어렵다. 작업의 개수가 n개로 증가할수록 시스템 부하의 영향을 받아 실제 시간과 예상 시간과의 오차가 커지기 때문이다. 그러나, 예상 실행시간의 비율 expect[]와 실제 실행시간의 비율 real[]을 계산하는 것은 아주 간단하므로, 우리는 그림 1에서와 같이 반복 계산하여 n번째 예상 실행시간인 T_expectedtime[n]을 예상할 수 있다. 이를 통해 전체 시스템 실행시간을 단축하고 최적 스케줄링을 실시함으로서 프로세서의 활용도를 높일 수 있다.



<그림 2> 예상 실행시간의 계산 순서

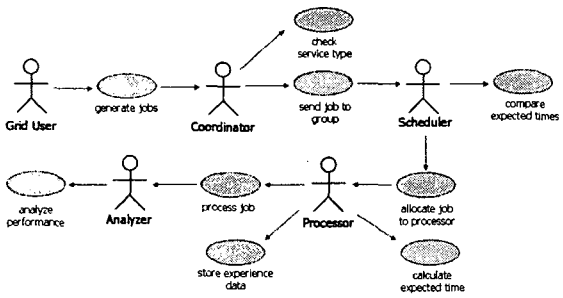
<그림 2>는 예상 실행시간의 계산 순서를 도식화한 것이다. 모델에서는 첫 번째 작업이 그리드 시스템에 도착하였을 때, 두 번째 작업의 예상시간이 예상 실행시간 비율과 실제 실행시간 비율을 이용하여 계산된다. 두 번째 작업이 그리드 시스템에 도착하였을 때는 세 번째 작업의 예상 실행시간이 계산된다. 즉, 모델에서는 n번째 작업이 프로세서에 도착함과 동시에 n+1번째 작업의 예상 실행시간이 계산된다. 계산된 예상 실행시간은 스케줄러에 전달되어 작업을 할당하는데 이용된다.



<그림 3> 멀티 서비스를 위한 그룹화

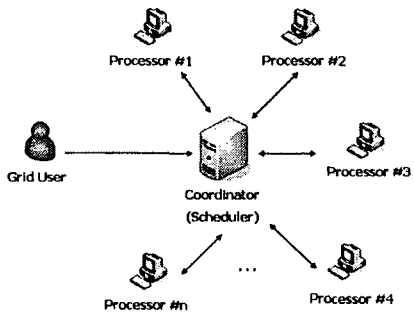
서비스 예측 기반의 작업 스케줄링 모델은 그리드 상에서 단일 유형 서비스뿐만이 아닌 다양한 멀티 유형 서비스도 지원한다. 만약 단일 유형 서비스에서와 같이 실행시간을 예측한다면 서비스 간의 실행시간 불균형으로 인해 예상 실행시간의 오차는 증가할 수밖에 없다. 본 논문에서는 이러한 멀티 유형 서비스의 실행시간 예측문제를 해결하기 위한 방법으로 그룹화(Grouping)를 적용한다. 그룹의 분류는 서비스간의 평균 실행시간 비율을 이용한다. 그리드에서 계산처리를 요하는 서비스의 유형이 두 가지이고, A유형은 평균 실행시간이 10이며 B유형은 평균 실행시간이 20이라고 가정한다. 이때, 코디네이터는 <그림 3>과 같이 이용 가능한 그리드상의 프로세서들을 1:2의 비율로 분류한다. 즉, 그룹의 수는 서비스의 수와 일치하고 그룹 내의 프로세서 수는 서비스간의 실행시간 비율과 연관된다. 코디네이터의 작업부하 집중현상을 방지하기 위해 각 유형별로 프로세서들을 관리할 스케줄러를 배치하여 작업 할당을 담당한다. 그리드 사용자에 의해 작업이 입력되면 코디네이터는 작업의 서비스 유형을 예측하여 유형에 맞는 그룹의 스케줄러에 작업을 전달하고 스케줄러는 그룹 내의 최저 실행시간을 제공하는 프로세서에게 작업을 할당한다. 작업 스케줄링 모델은 이러한 그룹화 과정을 통하여 코디네이터에 발생할 수 있는 작업부하의 집중을 막고 서비스

유형별 예상 실행시간을 제공함으로서 더욱 정교한 예측을 가능케 한다.



<그림 4> 작업 스케줄링 모델 사례도

<그림 4>는 작업 스케줄링 모델 내에서 단위 작업들의 순서를 나타내는 사례도이다. 모델의 참여자는 크게 그리드 사용자, 코디네이터, 스케줄러, 프로세서, 분석가로 분류된다. 그리드 사용자가 계산 처리를 요하는 컴퓨팅 문제를 코디네이터에게 제공하면 코디네이터는 서비스 유형을 파악하고 유형에 적합한 그룹의 스케줄러에게 작업을 전달한다. 스케줄러는 그룹 내 프로세서들의 예상 실행시간을 비교하여 최소 예상 실행시간을 제공하는 프로세서에게 작업을 할당한다. 작업을 전달받은 프로세서는 작업 발생시간, 처리 시간, 서비스 유형 등과 같은 경험적 데이터를 저장하고 다음 작업의 예상 실행시간을 계산한다. 처리가 완료된 작업은 분석가에게 전달되어 시스템 성능 평가에 이용된다. 실제 계산 그리드 상에서 모델 참여자들은 <그림 5>와 같이 구성된다. 중앙 스케줄링 방식과 같이 중앙에 스케줄링을 담당하는 코디네이터를 선정하고 그리드 사용자와 프로세서들은 네트워크 연결을 통해 코디네이터와 통신하여 예상 실행시간을 전달하고 작업을 할당받는다. 그러나, 중앙 스케줄링 방식과는 달리 스케줄러는 단순히 작업 전달과 예측 시간 비교, 서비스 예측의 기능만을 담당하며 예측 실행시간 계산과 정보 저장은 각 프로세서들이 담당한다.

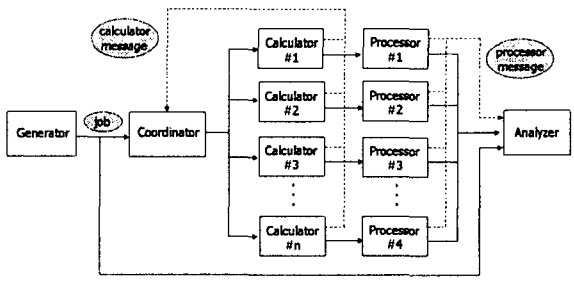


<그림 5> 계산 그리드 상의 모델 구성

4. 실험 및 결과 분석

본 논문에서 제안한 서비스 예측 기반의 작업 스케줄링 모델의 성능을 평가하기 위하여 <그림 6>, <그림 7>과 같이 시뮬레이션 모델을 구성하였다. 실험을 위한 시뮬레이션 환경은 DEVSJAVA 모델링&시뮬레이션 환경[16]을 사용하였다. 실험은 단일 유형 서비스와 멀티 유형 서비스로 실시하였고 서비스 예측 기반의 작업 스케줄링 방식 외에도 기존의 전통적인 모델인 랜덤 스케줄링(Random scheduling) 모델과 라운드로빈 스케줄링(Round-robin scheduling) 모델과의 실험 비교를 통하여 모델의 효과를 입증한다.

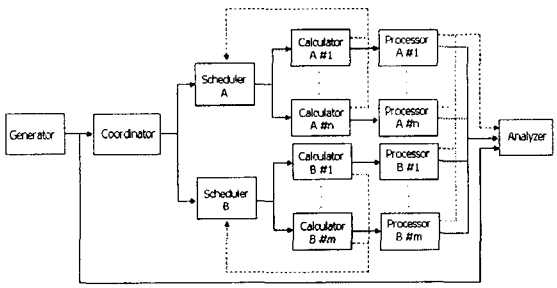
4.1 시뮬레이션 모델 구성



<그림 6> 단일 유형 서비스의 모델 구성도

단일 유형 서비스를 위한 시뮬레이션 모델은 <그림 6>과 같이 다섯 종류의 컴포넌트로 구성된다. Generator는 그리드가 처리해야 하는 컴퓨

팅 문제 즉, 작업을 생성시키는 그리드 사용자의 역할을 담당하고 생성된 작업들을 Coordinator와 Analyzer에게 전송한다. Coordinator는 작업을 입력받기 전 Calculator들로부터 예상 실행시간이 저장된 메시지를 전송받아 예상 실행시간을 기록하는 리스트를 구성하고 작업을 입력받은 후에는 예상 실행시간 리스트를 참조하여 출력포트를 지정하고 작업을 전송한다. Calculator는 3장에서 제안한 실행시간 예측 알고리즘을 사용하여 Processor의 다음 작업의 예상 실행시간을 계산한다. 계산된 예상 실행시간은 Calculator 메시지에 저장하여 Coordinator에게 전달하며 작업은 Processor에게 전달한다. Processor는 작업을 처리하고 작업손실과 프로세서 활용도 등의 데이터를 Processor 메시지에 저장하여 작업이 완료됨과 동시에 Analyzer에게 작업과 함께 전송한다. Analyzer는 Generator와 Processor에게 전달받은 작업과 메시지를 이용해 작업 손실, 작업 처리량, 평균 처리 소요시간 등을 계산하여 시스템 성능을 측정한다.



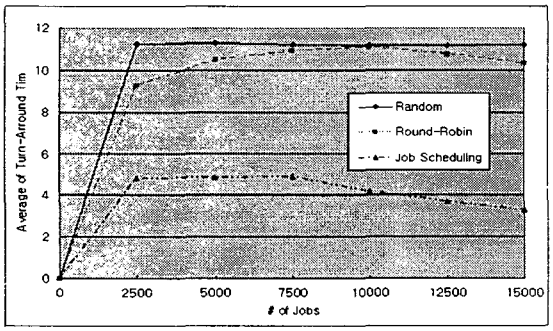
<그림 7> 멀티 유형 서비스 모델 구성도

멀티 유형 서비스를 위한 시뮬레이션 모델은 <그림 7>과 같이 그룹별 Scheduler가 추가되어 구성된다. 단일 유형 서비스와는 달리 Coordinator는 단지 작업의 서비스 유형을 체크하여 적합한 그룹의 Scheduler에게 작업을 전달하는 역할만을 수행한다. Scheduler는 그룹 내 Processor들의 예상시간 리스트를 구성하여 작업이 입력될 경우 예상시간 리스트를 이용하여 출력포트를 지정한다. 그룹 A의 Processor 수 n과 그룹 B의 Processor 수 m은 3장에서 언급한 것과 같이 서비스 유형의 평균 실행 시간의 비율로 결정된다.

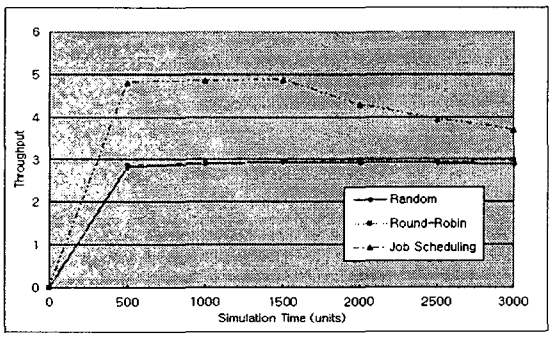
4.2 시뮬레이션 결과 분석

4.2.1 실험 1: 단일 유형 서비스

실험 1은 그리드 내의 프로세서가 동일한 서비스만을 처리한다는 가정 하에 단일 유형 서비스에 대한 평균 작업처리 소요시간과 평균 작업 처리량을 측정하는 실험이다. 우리는 신뢰성 있는 실험을 위하여 약 15000개의 작업을 생성시켜 각각의 스케줄링 모델에 따른 평균 작업처리 소요시간을 분석하였다. <그림 8>은 각 스케줄링 모델들의 작업 개수별 평균 처리 소요시간을 그래프화한 것이다. 작업의 개수가 많고 적음에 관계없이 작업 스케줄링 모델은 랜덤 스케줄링 모델, 라운드로빈 스케줄링 모델들과 비교하여 확연히 단축된 평균 작업처리 소요시간을 제공한다. <그림 9>와 <표 2>는 각 스케줄링 모델들의 시간당 평균 작업 처리량을 나타낸다. <그림 9>에서 작업 스케줄링 모델은 최대 작업 처리량을 랜덤 스케줄링 모델은 최소 작업 처리량을 제공한다. 예를 들어 <표 2>에서 1000초일 경우 작업 스케줄링 모델에서는 4,852개의 작업을 처리한 반면 랜덤 스케줄링 모델은 2,903개의 작업을 라운드로빈 스케줄링 모델은 2,925개의 작업을 처리한다. 이는 제안한 모델이 단위 시간당 가장 많은 작업을 처리한다는 것을 증명한다. 위의 결과들은 본 논문에서 제안한 작업 스케줄링 모델이 다른 모델들에 비해 작업처리 소요시간을 단축시키고 작업 처리량을 증가시킴으로서 더 나은 시스템 실행시간과 처리능력을 제공하는 것을 나타낸다. 또한, 작업 개수의 증가 즉 처리 요구량이 증가해도 작업 당 평균 처리시간이 증가하지는 않는다는 결과는 모델의 성능이 시간과 관계없이 안정적이라는 사실을 증명한다.



<그림 8> 작업 개수별 평균 처리 소요시간



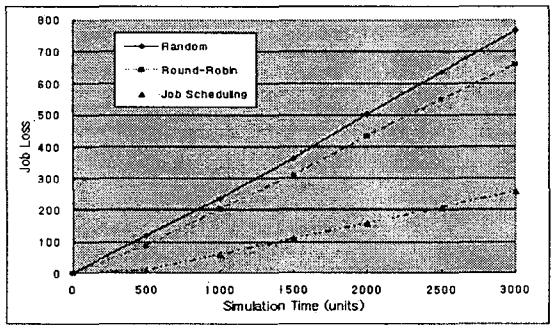
<그림 9> 모델별 시간당 작업 처리량

<표 2> 시뮬레이션 시간당 총 작업 처리량

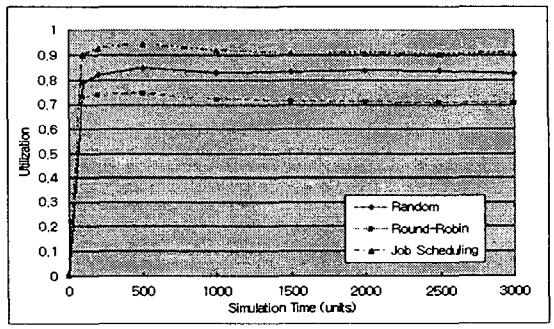
Time(s)	Prediction	Random	Round-Robin
500	2402	1412	1425
1000	4852	2903	2925
1500	7302	4403	4425
2000	8578	5838	5925
2500	9850	7311	7375
3000	11079	8688	8875

4.2.2 실험 2: 멀티 유형 서비스

실험 2는 계산 그리드의 동적인 환경 특성을 고려한 멀티 유형 서비스에 대한 작업 손실과 프로세서 활용도에 관한 실험이다. 실험에 사용된 서비스 작업들은 A유형과 B 유형으로 분류되며 각각 9~11과 28~32의 처리시간을 필요로 한다. <그림 10>은 각 스케줄링 모델들의 시뮬레이션 시간 흐름에 따른 작업 손실률을 측정하여 그래프화한 것이다. 그림에서 작업 스케줄링 모델은 랜덤 스케줄링 모델, 라운드로빈 스케줄링 모델과 비교하여 감소된 작업 손실률을 보여준다. 예를 들어 2000초에서 작업 스케줄링 모델에서는 156개의 작업 손실이 발생하여 7.8%의 작업 손실률을 기록한 반면 랜덤 스케줄링 모델과 라운드로빈 스케줄링 모델은 503개와 432개의 작업 손실이 발생하여 각각 25.15%와 21.6%의 손실률을 기록하였다. 이 결과는 작업 스케줄링 모델이 기존의 스케줄링 모델들과 비교하여 그리드 네트워크상에서 신뢰성 있는 전송을 제공한다는 것을 증명한다. <그림 11>은 각 모델들의 프로세서 활용도를 측정하여 그래프화한 것이다. 작업 스케줄링 모델이 다른 두 모델에 비해 확연히 높은 평균 90%이상의 프로세서 활용률을 보여주고 있다. 이는 휴면 자원을 줄이고 자원 이용도를 높여 전체 시스템 활용도를 증가시킴으로서 더 많은 양의 작업을 처리할 수 있다는 것을 증명한다.



<그림 10> 모델별 시간당 작업손실



<그림 11> 모델별 시간당 프로세서 활용도

5. 결론

시간이 갈수록 계산 그리드에 대한 관심과 요구가 계속 증가함에 따라 그리드 컴퓨팅 기술은 발전하고 있으며 계산 그리드를 적용한 프로젝트 역시 한창 진행되고 있다. 이러한 계산 그리드의 발전에 맞춰 그리드 스케줄링에 관한 중요성과 필요성 또한 증가하고 있다. 본 논문에서는 계산 그리드의 스케줄링 문제를 해결하기 위하여 서비스 예측 기반의 작업 스케줄링 모델을 제안하였다.

서비스 예측 기반의 작업 스케줄링 모델에서는 효과적인 그리드 자원 할당과 스케줄링을 위하여 각 프로세서들로부터 경험적인 데이터를 수집한다. 그 데이터들을 바탕으로 실행시간 예측 알고리즘을 적용하여 프로세서들의 실행시간을 예측하고 이를 정렬하여 적절한 프로세서에 작업을 할당한다. 또한, 다양한 서비스를 포함하는 그리드의 동적인 환경 특성을 고려하여 그리드 상의 프로세서들을 유형별로 그룹화하고 작업의 서비스 유형을 예측하여 적절한 그룹에 작업을 전달함으로서 예측 실행시간의 오차를 최소화하고 효율적인 작업의 분배를 실시한다.

작업 스케줄링 모델의 성능 평가를 위해 본 논문에서는 시뮬레이션 모델을 설계하여 단일 유형 서비스와 멀티 유형 서비스에 대한 실험을 실시하였다. 실험 결과는 본 논문에서 제안한 서비스 예측 기반의 작업 스케줄링 모델이 기존의 스케줄링 모델인 랜덤 스케줄링 모델, 라운드로빈 스케줄링 모델과 비교하여 단축된 평균 작업처리 소요시간과 높은 작업 처리율을 제공하며 작업 손실을 줄이고 프로세서 활용도를 향상시킨다는 것을 증명한다. 따라서, 서비스 예측 기반의 작업 스케줄링 모델을 계산 그리드에 적용하게 되면 그리드 스케줄링 문제를 해결함과 동시에 프로세서 활용도와 효율을 극대화하여 전체적인 시스템 성능 향상을 기대할 수 있을 것이다.

Acknowledgement

본 연구는 정보통신부 및 정보통신 연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음.

참고 문헌

[1] F. Berman, G. Fox and T. Hey, Grid computing :making the global infra structure a reality, J. Wiley, New York, 2003.

[2] P. Kacsuk, et al., Distributed and parallel systems :cluster & grid computing, Kluwer Academic Publishers, Boston, 2002.

[3] I. Foster and C. Kesselman (eds.) "The Grid: Blueprint for a new Computing Infrastructure" Morgan Kaufmann Publishers, 1998.

[4] F. Berman, G. Fox and T. Hey, Grid computing :making the global infrastructure a reality, J. Wiley, New York, 2003.

[5] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of High Performance Computing Application, Vol.15, No.3, pp. 200-222, 2001.

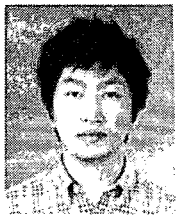
[6] R. Wolski, J. Plank, J. Brevik, and T. Bryan, "Analyzing Market-based Resource Allocation Strategies for the Computational Grid", International Journal of High performance Computing Applications, Sage Publications, Vol 15, No.3, pp. 258-281, 2001.

[7] C. Boeres et al. A tool for the Design and Evaluation of Hybrid Scheduling Algorithms for Computational Grids. Proceedings of the 2nd workshop on Middleware for grid computing, ACM International Conference, October 2004.

[8] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. Software Practice and Experience, 2:135 - 164, 2002.
[9] T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," IEEE Trans. Software Eng., vol. 14, no. 2, pp. 141-154, Feb. 1988.
[10] Q. Wang and X. Gui, et al. "De-centralized Job Scheduling on Computational Grids Using Distributed Backfilling", Lecture Notes in CS, Springer-Verlag, vol. 3251, pp. 285- 292, 2004
[11] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of Job -Scheduling Strategies for Grid Computing. In Proc. Grid '00, pages 191 -202, 2000.
[12] http://www.gridbus.org
[13] http://hipersoft.cs.rice.edu/grads
[14] http://www.cs.wisc.edu/condor
[15] Yang Gao, Hongqiang Rong, Frank Tong, et al. "Adaptive Job Scheduling for a Service Grid Using a Genetic Algorithm" Lecture Notes in CS, Springer-Verlag Vol. 3033, pp. 65-72, 2004
[16] B.P. Zeigler, et al., "The DEVS Environment for High-Performance Modeling and Simulation", IEEE C S & E, Vol. 4, No3, pp. 61-71, 1997.

주 작 성 자 : 장 성 호
논 문 투 고 일 : 2005. 06. 28
논 문 심 사 일 : 2005. 07. 18(1차), 2005. 07. 24(2차), 2005. 08. 01(3차)
심 사 판 정 일 : 2005. 08. 01

저자소개



장성호
2004 용인대학교 컴퓨터정보학과 학사
2004 ~ 현재 인하대학교 컴퓨터공학부 석사과정
관심분야: 그리드 컴퓨팅, 모델링 및 시뮬레이션

이종식
1993 인하대학교 전자공학과 학사
1995 인하대학교 전자공학과 석사
2001 아리조나대 컴퓨터공학과 박사
2001 ~ 2002 캘리포니아 주립대학교 전기·컴퓨터공학과 전임강사
2002 ~ 2003 클리블랜드 주립대학교 전기·컴퓨터공학과 조교수
2003 ~ 현재 인하대학교 컴퓨터공학부 조교수
관심분야: 시스템 모델링 및 시뮬레이션, 그리드 컴퓨팅, 미들웨어, 유비쿼터스 컴퓨팅