GRID 환경을 위한 포인트 기반 스케쥴링 알고리즘

오영은° 김진석
서울시립대학교 컴퓨터과학부
{iceize, jskim}@venus.uos.ac.kr

A Point-based Scheduling Algorithm for GRID Environment

Young-Eun Oh[0] and Jin Suk Kim
School of Computer Science, University of Seoul

요 약

지역적으로 분산되어 있는 고성능의 컴퓨팅 자원들을 한 데 묶어 이용하는 그리드 기술이 발전함에 따라, 그리드 환경에서의 스케쥴링이 절실히 필요하게 되었다. 본 논문에서는 그리드 자원 정보를 이용하여 그리드 자원을 효율적으로 이용할 수 있는 스케쥴링 알고리즘에 대하여 고찰해보고, 자원의 성능을 고려한 스케쥴링 알고리즘을 설계하여 실험하였다. 이러한 실험을 위하여 그리드 환경을 시뮬레이션 하였으며, 이 결과를 다른 알고리즘과 비교하였다.

1. 서론

지역적으로 분산되어 있는 고성능의 컴퓨팅 자원들을 한 데 묶어 이용하는 그리드(GRID)에서는 자원을 효율적으로 이용하는 것이 무엇보다 중요하다. 그리드 미들웨어로 많이 사용하고 있는 Globus Toolkit [1]에서는 사용자가 그리드 자원에 직접 접속하여 자원의 상태를 조회한 후, 작업을 실행시키기에 가장 적당한 자원을 찾아야 한다. 그러나 이것은 번거로운 일이며, 자원의 상태가 바뀔 경우에 대하여 잘못된 결과가 도출될 수 있다.

본 논문에서는 그리드 환경에서의 사용자 작업을 효율적으로 처리할 수 있도록 최적의 자원을 선택하는 스케쥴링 알고리즘을 설계하였다. 또한 실험을 통하여 기존의 라운드 로빈 스케쥴링 방식에 비해 성능이 향상되었음을 보였다.

2. 관련 연구

2.1. 동적 스케쥴링 알고리즘

동적 스케쥴링 알고리즘은 스케쥴링을 언제 수행하는지에 따라 온라인 방식과 배치 방식의 알고리즘으로 나눌 수 있다[2]. 온라인 방식의 스케쥴링 알고리즘이란 사용자 작업을 큐에 저장하지 않고 작업이 도착하는 즉시 스케쥴링 하는 방식을 말한다. 대표적인 온라인 방식의 스케쥴링 알고리즘으로는 MET (Minimum Execution Time)[3], MCT (Minimum Completion Time), SA (Switching Algorithm) 등이 있다[4]. 또한 배치 방식의 알고리즘으로는 완료 시간, 준비 시간, 실행 시간을 기준으로 한 스케쥴링 방법 및 유전자 알고리즘을 이용한 방법 등이 있다[5]. 그러나 그리드 환경에서는 사용자 작업마다의 기대 실행 시간과 기대 완료 시간 등을 알기 어려우므로, 그리드 환경에 적합한 알고리즘이 필요하게 되었다.

2.2. 그리드 환경에서의 스케쥴링 알고리즘

그리드 환경에서의 스케쥴링 알고리즘으로는, 자원의 상태를 고려한 K-Windows 알고리즘과 PSS 알고리즘 등이 있으며[6], 사용자 작업에 대한 비용 최적화 및 시간 최적화 알고리즘이 있다[7]. 이 외에도 성능을 예측하기 위한 여러 알고리즘이 있다.

3. 실험 방법 및 환경

3.1. 고려 사항 및 가정

그리드 환경에서 사용자 작업과 자원에 대한 고려 사항은 다음과 같다.

- 사용자 작업: 요구 CPU 시간, 요구 노드 수, 요구 네트웍량, 분산된 자원간의 동기화의 필요 여부
- 자원: CPU 속도, 전체 노드 수, 사용 가능한 노드 수, 네트웍 대역폭

또한 스케쥴링 알고리즘을 실험하기 위하여 다음과 같은 가정을 세웠다.

- 그리드 자원은 동일한 아키텍쳐로 구성
- 선택된 자원에 대해서는 반드시 하나의 작업만이 실행됨
- 스케쥴러는 자원, 네트웍의 상태와 사용자 작업의 요구사항을 알 수 있음
- 스케쥴러는 사용자 작업에 대한 기대 실행 시간 및 기대 완료 시간을 알 수 없음

그리고 온라인 방식과 배치 방식을 실험하기 위하여 작업 사이에 유휴 시간 (idle time)을 두었으며, 유휴 시간은 이전 작업

이 제출되고, 그 다음 작업이 제출될 때까지의 시간을 말한다. 본 논문에서는 스케줄링 방식에 따른 성능을 평가하기 위하여 유휴 시간은 포아송 분포를 따르고, 요구 CPU 시간 및 요구 노드의 수, 요구 네트웍량, 자원 네트웍 대역폭 등은 모두 정규 분포를 따르도록 하였다.

3.2 시뮬레이션 환경

<그림 1>은 그리드 환경을 시뮬레이션 하기 위하여 사용한 구조이다.

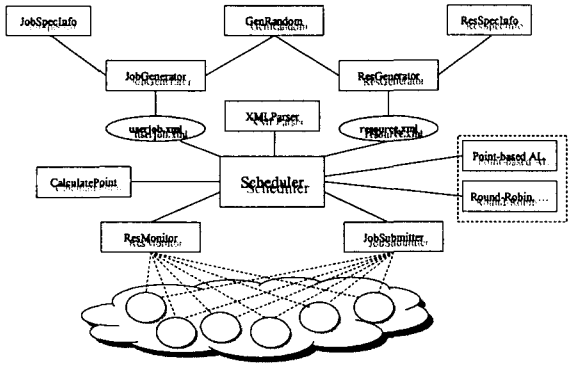

그림 1. 그리드 시뮬레이터의 구조

JobGenerator와 ResGenerator 클래스를 이용하여 만들어진 userjob.xml 파일과 resource.xml 파일의 DTD는 다음과 같다.

표 1. 사용자 작업에 대한 DTD

```
<?xml version="1.0" encoding="UTF-8">
<!ELEMENT cputime (#PCDATA)>
<!ELEMENT nodecount (#PCDATA)>
<!ELEMENT bandwidth (#PCDATA)>
<!ELEMENT idletime (#PCDATA)>
<!ELEMENT sync (#PCDATA)>
<!ELEMENT joblist (jobspec+)>
<!ELEMENT jobspec (cputime, nodecount, bandwidth,
idletime, sync)>
```

표 2. 그리드 자원에 대한 DTD

```
<?xml version="1.0" encoding="UTF-8">
<!ELEMENT cpuspeed (#PCDATA)>
<!ELEMENT nodecount (#PCDATA)>
<!ELEMENT systemload (#PCDATA)>
<!ELEMENT bandwidth (#PCDATA)>
<!ATTLIST bandwidth to CDATA #REQUIRED>
<!ATTLIST resource name CDATA #REQUIRED>
<!ELEMENT resourcelist (resource+)>
<!ELEMENT resource (cpuspeed, nodecount, systemload,
bandwidth+)>
```

본 논문에서는 시뮬레이션을 위하여 각각의 사용자 작업의 실행 시간을 계산량과 네트웍량으로 구분하였다. 각 사용자 작업의 실행 시간을 (계산량 - 네트웍량) * 타이머로 정의하였으며, 타이머가 동작할 때마다 각 작업의 CPU 시간이 감소하여 0이 되면 작업이 종료되도록 하였다. 만약 사용자 작업이 동기화를 필요로 한다면, 가장 느린 CPU와 가장 느린 네트웍 대역폭에 맞추어 감소하도록 하였다.

3.3. 스케줄링 알고리즘

다음은 그리드 자원을 평가하기 위한 모델링이며[6], P는 성능(Performance), PF는 성능 인자(Performance Factor), 그리고 α_i는 i번째 성능 인자에 대한 가중치를 나타낸다.

$$P = \alpha_1 PF_1 + \alpha_2 PF_2 + \ldots + \alpha_n PF_n$$

본 논문에서 제안한 스케줄링 알고리즘은 CPU의 속도가 빠른 노드에 높은 우선 순위를 주는 방식이며, 각 자원에 대한 포인트를 계산하는 방식은 다음과 같다.

$$P = \sum n\,RP - k\sum \frac{m}{NP}$$

위 식에서 n은 작업에 참여하는 노드의 수를 나타내며, m은 네트웍에 참여하는 노드의 수를 나타낸다. 또한 RP는 자원 포인트를 의미하며 CPU 속도를 시스템의 부하량으로 나눈 값이다. 만약 RP 값이 같다면, 노드의 개수가 많은 순으로 선택된다. 그리고 NP는 네트웍 포인트를 의미하며, 네트웍 대역폭 값을 갖는다. 사용자 작업이 계산 위주의 작업이라면 k값을 낮게 지정할 수 있으며, 사용자 작업이 네트웍 위주의 작업이라면 k 값을 높게 지정할 수 있다. 이렇게 CPU 속도와 시스템의 부하량을 고려한 이유는 시스템의 부하량이 클수록 단일 작업의 실행 시간이 길어지기 때문이며[8], 네트웍의 대역폭을 고려한 이유는 MPI 작업의 경우 대역폭에 따라 많은 성능 차이가 발생하기 때문이다[9]. 자원 선택의 기준은 다음과 같이 세가지 경우로 나누어 P_{max}을 갖는 경우를 선택한다.

- 한 자원에 작업을 할당할 수 있는 자원 중 가장 RP 값이 큰 자원을 선택
- 작업을 여러 자원에 할당하더라도, RP 값이 큰 순서대로 자원을 선택
- 작업을 여러 자원에 할당하더라도, 가장 RP 값이 큰 자원과의 NP 값이 큰 순서대로 선택

3.4. 실험 결과

실험은 유휴 시간이 0인 경우와 60인 경우로 나누어 실험하였으며, 각각마다 자원 혹은 작업을 고정하여 실험하였다. 또한 같은 방법으로 100회 실험하여, 평균값을 계산하였다. 본 논문에서 제안한 스케줄링 알고리즘의 성능을 평가하기 위하여 사용자 작업이 제출될 때마다 순서대로 자원이 할당되는 라운드 로빈 방식과 비교하였다. 다음 그림은 시뮬레이션 결과를 보여주고 있으며, 라운드 로빈 방식에 비해 대기 시간은 43%, 실행 시간은 30%, 완료 시간은 34%, makespan은 29%의 성능 향상을 보였다.

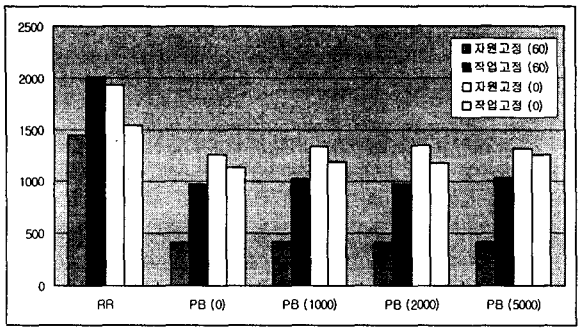

그림 2. 대기 시간

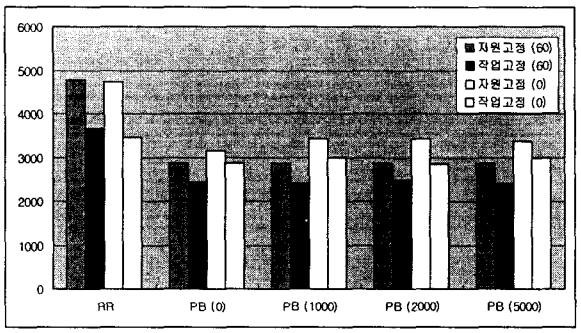

그림 3. 실행 시간

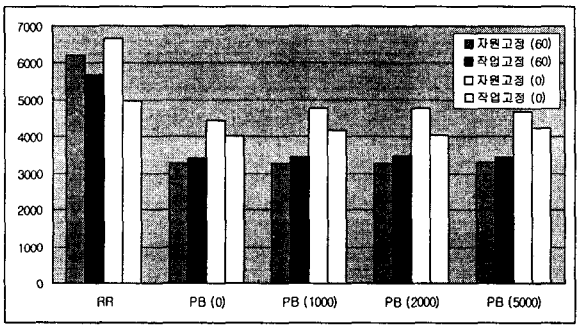

그림 4. 완료 시간

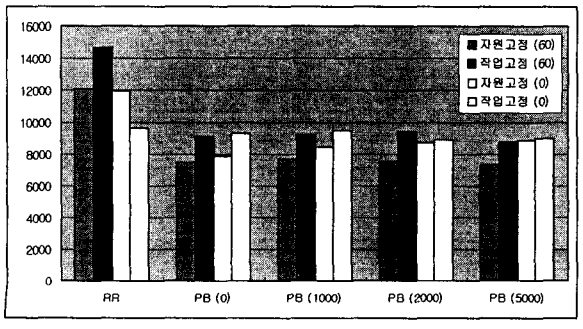

그림 5. Makespan

4. 결론

본 논문에서는 그리드 환경에서 효율적인 자원을 선택하는 스케줄링 알고리즘을 제안하였으며, 시뮬레이션을 통해 이를 실험 평가하였다. 그러나 본 논문에서는 작업의 실행 시간에 따른 변화나 자원의 노드 수에 따른 변화를 실험하지 못하였다. 향후 더 다양한 환경에서의 시뮬레이션을 통하여 제안한 알고리즘을 수정 보안하는 것을 목표로 두고 있다.

참고문헌

[1] Globus Toolkit, http://www.globus.org.

[2] 김학두, "이질적인 분산 컴퓨터 시스템을 위한 동적 스케줄링 알고리즘," 석사 학위논문, 서울시립대학교 컴퓨터과학부, 2003.

[3] R. F. Freund et al, "Scheduling Resources in Multi-User, Heterogenous, Computing Environments with SmartNet," *Proc. of the 7th Heterogenous Computing Workshop*, 1998.

[4] M. Maheswaran et al, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogenous Computing Systems," *Proc. of the 8th Heterogenous Computing Workshop*, 1999.

[5] T. D. Braun et al, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, 2001.

[6] Srisan E. et al, "Heuristic Scheduling with Partial Knowledge under Grid Environment," *Proc. of the 2nd International Symp. on Communications and Information Technology*, 2002.

[7] Rajkumar Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing," Ph. D. Thesis, Monash University, Melbourne, Australia, 2002.

[8] Shreenivasa Venkataramaiah et al, "Performance Prediction for Simple CPU and Network Sharing," *Proc. of the LACSI Symp.*, 2002.

[9] Bronis R. de Supinski et al, "Accurately Measuring MPI Broadcasts in a Computational Grid," *Proc. of the 8th IEEE International Symp. on High Performance Distributed Computing*, 1999.