그리드 환경에서 MPI 작업을 위한 스케줄링 시스템의 설계 및 구현

오영은[O] 김진석
서울시립대학교 컴퓨터과학부
{iceize, jskim}@venus.uos.ac.kr

Design and Implementation of Scheduling System for MPI Job on GRID Environment

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

요 약

여러 지역에 산재되어 있는 고성능의 컴퓨팅 자원들을 이용하는 그리드에서는 어떠한 자원을 선택하느냐에 따라 응용 프로그램의 성능이 좌우된다. 본 논문에서는 단일 클러스터의 노드 개수에 우선한 스케줄링 방법과 노드의 속도를 우선으로 한 스케줄링 방법을 고려하였다. 이러한 실험을 위하여 스케줄링 구조를 설계하고 그리드 자원에서 실제 실험하여 결과를 비교하였다.

1. 서론

여러 지역에 산재되어 있는 고성능의 컴퓨팅 자원들을 이용하는 그리드(GRID)에서는 그리드 자원을 효율적으로 이용하는 것이 무엇보다 중요하다. 이렇게 하기 위해서는, 그리드 자원에서 자원에 대한 상세한 정보를 제공하여야 하며, 자원 이용자는 그 정보를 이용하여 작업을 실행시키기에 가장 적당한 자원을 찾아야 한다.

본 논문에서는 그리드 환경에서 동작하는 스케줄러를 설계하고 구현하여 자원의 정보 표시 및 자원 선택의 기준에 대하여 실험하였다.

2. 관련 용어 및 연구

2.1. GQS (Global Queueing System)

GQS는 분산되어 있는 그리드 자원을 사용자가 쉽게 사용할 수 있는 인터페이스와 작업에 대한 로드 밸런싱을 제공하여 자원을 효과적으로 사용하기 위해 제안되었다. 즉 GQS를 이용하여 사용자가 원하는 최적의 자원을 제공하고 스케줄링 정책을 이용하여 자원을 효율적으로 사용할 수 있다[1]. GQS에서는 이와 같은 목적을 달성하기 위해 다음과 같은 사항을 고려해야 한다.

- 그리드 자원 간의 인증 문제:
  MyProxy[2]를 이용하여 인증 문제를 해결할 수 있다.
- 사용자에게 투명한 스케줄러 제공:
  자원 상태를 실시간으로 모니터링할 수 있어야 하며, 사용자 작업을 실행하기 위한 자원을 선택하고, 실행 결과를 사용자에게 보고해야 한다.

2.2. 스케줄링 알고리즘

사용자 작업을 자원에 할당하는 동적인 스케줄링 방식은 스케줄링 시기에 따라 온라인 방식과 배치 방식으로 나눌 수 있다. 온라인 방식은 작업이 도착하는 즉시 스케줄링 하는 방식이며 배치 방식은 일정 시간에 도착하는 작업들을 임시로 저장해 두었다가 스케줄링 이벤트가 발생할 때 저장했던 작업들을 모두 스케줄링하는 방식이다[3]. 온라인 방식의 알고리즘으로는 대표적으로 MET (Minimum Execution Time), MCT (Minimum Completion Time), SA (Switching Algorithm), KPB (K-Percent Best) 등이 있으며[4], 배치 방식의 알고리즘으로는 완료 시간, 준비 시간, 실행 시간을 기준으로 한 스케줄링 방법, Genetic 알고리즘을 이용한 방법 등이 있다[5]. 또한, 그리드 환경에서 제안된 알고리즘으로는 비용 최적화 및 시간 최적화 알고리즘이 있다[6].

3. 실험 방법 및 환경

3.1. 실험 방법

그리드 환경에서 스케줄링 실험을 하기 위해서 다음과 같은 가정을 세웠다.

- 클러스터는 동일 아키텍처로 구성됨
- 트래픽으로 인한 시스템 오버 헤드는 없음
- 사용자 작업 파일 (실행 파일, 라이브러리, 데이터 파일)의 전송은 없음
- 기대 실행 시간에 대한 프로파일이 제공되지 않음
- 지역 작업 관리자는 단일 큐를 사용함

그리고 사용자 작업을 실행하기 위한 노드를 선택하기 위해, 다음과 같은 MDS 값을 검색하였다.

Category	MDS DN
Available node count	Mds-Computer-Total-Free-nodeCount, Mds-Job-Queue-name=default, Mds-Software-deployment=jobmanager-pbs
Total node count	Mds-Computer-Total-nodeCount, Mds-Job-Queue-name=default, Mds-Software-deployment=jobmanager-pbs
CPU speed	Mds-Cpu-SpeedMHz, Mds-Device-Group-name=processors
System load	Mds-Cpu-Free-5minX100, Mds-Device-Group-name=processors

표 1. 자원의 상태를 검색하기 위한 MDS 값

본 논문에서는 온라인 방식과 배치 방식의 스케줄링 실험을 하기 위하여 유휴 시간 (idle time)을 고려하였다. 그리고 각각의 스케줄링 방식에 따른 성능을 측정하기 위해 프로그램의 인자, 유휴 시간 및 각 작업에서 요구하는 노드 수를 동일한 평균 및 편차값을 갖는 균일 분포 변수로 선정하였다.

3.2. 실험 환경 및 구조

본 논문에서는 그리드 미들웨어로 글로버스 툴킷 2.0을 사용하였으며, 병렬 처리 라이브러리로 MPICH-G2, 지역 작업 관리자로는 OpenPBS를 사용하였다. <표 2>는 실험에서 사용한 자원에 대한 정보이며, <그림 1>은 스케줄링 알고리즘을 실험한 구조를 나타낸다.

Clustername	Speed (MHz)	Node count	Location
grid	933	7	A
gridtest	1800	6	B
hpctest	800	2	A

표 2. 실험에서 사용한 자원에 대한 정보

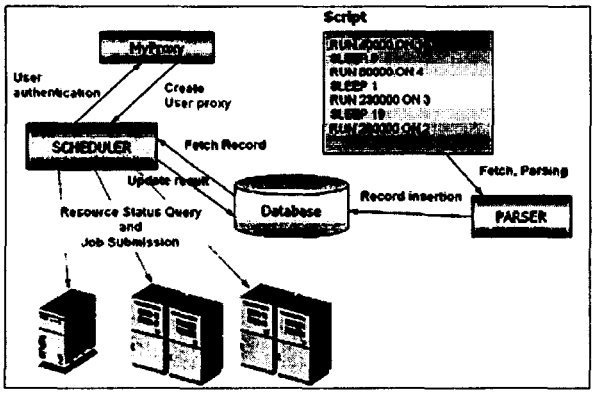

그림 1. 실험 구조

<그림 1>에서 스케줄러는 데이터베이스에서 사용자 작업에 대한 정보를 가져와 스케줄링 방법에 맞는 그리드 자원을 선택한 후 실제로 할당한다. 만약 사용자 작업이 요구하는 노드 수보다 사용한 가능한 노드의 수가 작다면, 작업이 PENDING 상태로 놓이게 된다. 그리고 사용자 작업이 요구하는 노드 수가 전체 노드의 수보다 크다면 작업은 취소된다.

3.3. 스케줄링 방법

본 논문에서는 노드 수에 우선 순위를 둔 스케줄링 방법과 CPU 속도에 우선 순위를 둔 스케줄링 방법을 고려하였다. 노드 수를 우선으로 한 스케줄링에서는 그리드 환경 내의 단일 클러스터에 대한 사용 가능한 노드 수, 전체 노드의 수, CPU 속도, 시스템 부하량을 고려하였다. 예를 들어, 사용자의 첫 번째 작업은 여러 개의 클러스터 중 사용 가능한 노드 수가 가장 많은 클러스터에 할당된다. 이러한 스케줄링 방법에서는 MPI 작업 시에 통신 비용이 낮으므로, WAN 환경이나 높은 트래픽의 작업에 적합하다. 반면, 노드의 속도를 우선으로 한 스케줄링에서는 CPU 속도, 시스템 부하량, 전체 노드의 수, 사용 가능한 노드의 수를 고려하였다. 이러한 스케줄링 방법에서는 일반적으로 대기 시간과 실행 시간이 짧기 때문에, LAN 환경이나 빠른 실행을 요구하는 작업에 적합하다. <표 3>은 각각의 스케줄링 방법을 비교하기 위한 예제 스크립트이며, <그림 2>는 각각의 스케줄링 방법에 대한 작업 할당 그래프이다.

```
RUN 40000 ON 15
SLEEP 9
RUN 50000 ON 4
SLEEP 1
RUN 230000 ON 3
SLEEP 19
RUN 290000 ON 2
SLEEP 12
RUN 300000 ON 10
SLEEP 7
RUN 90000 ON 8
```

표 3. 작업을 실행하기 위한 예제 스크립트

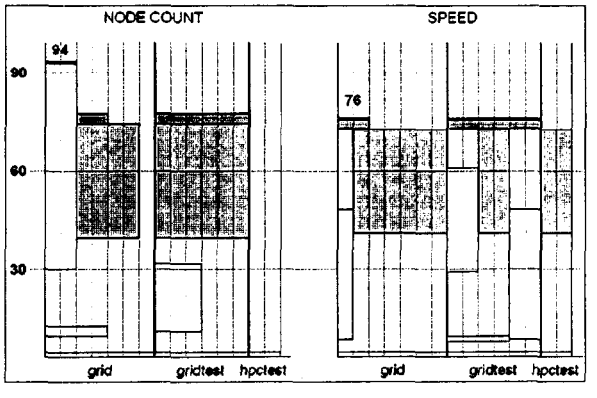

그림 2. 예제 스크립트에 대한 작업 할당 그래프

4. 실험 결과

본 논문에서는 작업 스크립트를 만들어 사용자 작업을 각각 10개, 30개, 50개, 100개로 설정하여 실험하였다. 실험 결과는 <표 4>와 같다.

		Node Count	Node Speed	Ratio
10	Ready Time	30138.5	18889.0	0.63
	CompletionTime	38311.0	25988.0	0.68
	Makespan	6030.5	4572.5	0.76
30	Ready Time	68833.7	16382.7	0.24
	Completion Time	110940.7	47772.3	0.43
	Makespan	34042.0	28254.7	0.83
50	Ready Time	447072.0	120972.0	0.27
	Completion Time	517604.0	173635.0	0.34
	Makespan	55605.0	46921.0	0.84
100	Ready Time	1103557.0	237989.0	0.22
	Completion Time	1235782.0	321024.0	0.26
	Makespan	109074.0	100312.0	0.92
Average	Ready Time	412400.3	98558.2	0.24
	Completion Time	475659.4	142104.8	0.30
	Makespan	51187.9	45015.1	0.88

표 4. 스케줄링 실험 결과

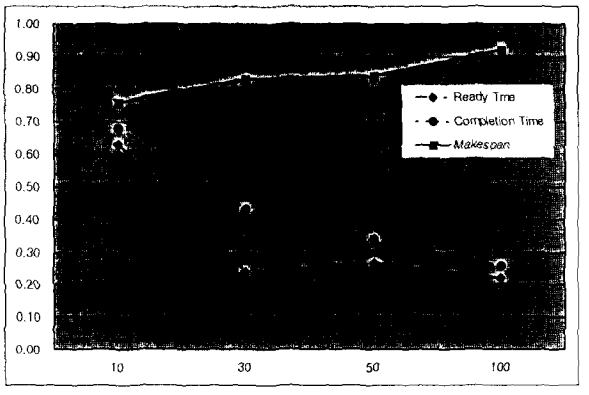

그림 3. 작업의 증가에 따른 스케줄링 실험 결과 그래프

<그림 3>은 작업 수의 증가에 따른 결과 그래프를 보여준다. 사용자 작업의 수가 증가할수록 각 작업의 대기 시간과 완료 시간의 상대값이 줄어들기 때문에 단일 클러스터의 노드 수에 우선한 스케줄링보다 노드의 속도에 우선한 스케줄링이 더 좋은 성능을 보인다는 것을 알 수 있다. 반면 사용자 작업의 수가 증가할수록 최종 완료시간의 상대값이 증가하는 이유는 스케줄링 방식에 따른 유휴 시간의 누적 때문이라고 볼 수 있다.

5. 결론

본 논문에서는 그리드 환경에서 어떠한 자원을 선택하는 것이 좀 더 효율적인지 알아보는 스케줄링 실험을 하였다. 실험 결과 작업을 할당하는 자원의 밀집도보다는 속도에 더 의존한다는 것을 알 수 있었다. 그러나 실험에서 사용한 자원은 노드의 개수가 많지 않고 지역적인 거리가 멀지 않기 때문에 이에 대한 보완이 요구된다. 향후 응용 프로그램의 프로파일을 이용한 스케줄링 방법과 과금을 고려한 스케줄링, GASS[7]를 이용한 프로그램 및 데이터 전송에 관련한 스케줄링 등을 연구, 실험하는 것을 목표로 하고 있다.

참고문헌

[1] "글로벌 큐잉 서비스 기술 개발에 관한 연구," 한국과학기술정보연구원 위탁연구과제 최종보고서, 서울시립대학교 컴퓨터과학부, 2002.

[2] J. Novotny, S. Tuecke, and V. Welch, "An Online Credential Repository for the Grid: MyProxy," *Proc. of the 10th IEEE Symp. on High Performance Distributed Computing*, 2001.

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

[4] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Proc. of the 8th Heterogeneous Computing Workshop*, 1999.

[5] T. D. Braun, H. J. Siegel, and Noah Beck, "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] Rajkumar Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing," Thesis submitted for the degree of Doctor of Philosophy, 2002.

[7] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," *Proc. of the 6th Workshop on I/O in Parallel and Distributed Systems*, 1999.