그리드 컴퓨팅에서 자원 할당을 위한 실시간 스케줄링 정책

최준영° 이원주* 전창호**
한양대학교 전자컴퓨터공학부°,두원공과대학 인터넷프로그래밍과*, 한양대학교 전자컴퓨터공학부**,
jychoe@cse.hanyang.ac.kr°, wonjoo@doowon.ac.kr*, chjeon@cse.hanyang.ac.kr**

Real-Time Scheduling Strategy for Resource Allocation in Grid Computing

Junyoung Choe° Wonjoo Lee*, Changho Jeon**
School of Electrical and Computer Engineering, Hanyang University°,
Department of Internet Programming, Doowon Technical College*,
School of Electrical and Computer Engineering, Hanyang University**

요 약

그리드 컴퓨팅에서는 자원의 상세정보를 실시간으로 사용하기 어렵기 때문에 자원 관리와 할당이 기존 시스템에 비해 비효율적이다. 따라서, 본 논문에서는 작업의 총 실행시간을 예측하여 그리드 자원을 할당하는 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 최소인 로컬시스템에 작업을 할당한다. 로컬 스케줄러에서는 할당된 작업의 대기시간과 처리시간을 계산한 후, 작업을 데드라인내에 처리할 수 있다면 로컬 시스템에서 처리한다. 하지만 데드라인을 초과하면 다른 로컬시스템으로 이주시켜 처리함으로써 작업실패율(failure rate)과 자원비용(resource cost)을 최소화한다.

1. 서론

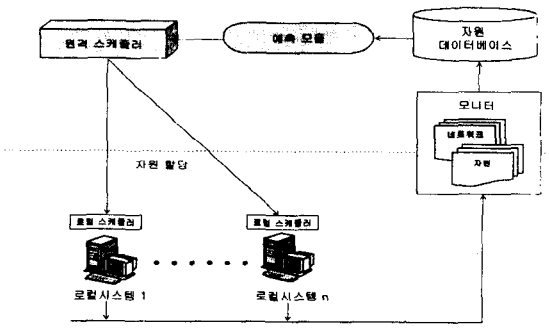
그리드 컴퓨팅은 지리적으로 분산되어 있는 컴퓨팅 자원을 공유하며 많은 양의 데이터 접근을 요구하는 과학 기술이나 컴퓨터 처리 사이클을 필요로 하는 분야에 널리 사용된다. 그리드 컴퓨팅에서는 작업 크기에 따른 네트워크 전송시간과 그리드 자원의 상태 정보를 실시간으로 사용하기 어렵다[1]. 따라서 실시간 작업의 총 실행시간을 정확하게 예측할 수 없기 때문에 그리드 자원 할당의 효율성이 떨어지는 문제점이 있다[2][3].

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 데드라인내에 처리할 수 있다면 로컬시스템에서 처리한다. 하지만 데드라인을 초과하면 다른 로컬시스템으로 이주(migration)시켜 처리함으로써 자원비용을 최소화한다[4].

본 논문의 구성은 다음과 같다. 2장에서는 그리드 컴퓨팅의 스케줄링 과정과 관련 연구에 대하여 기술한다. 또한, 기존의 스케줄링 정책의 문제점에 대하여 설명한다. 3장에서는 제안한 스케줄링 정책에 대하여 자세히 기술한다. 그리고 4장에서 결론 및 향후 연구과제를 제시한다.

2. 관련 연구

일반적으로 그리드 컴퓨팅의 스케줄링과정은 [그림 1]와 같이 스케줄러(scheduler), 예측모듈(predictor), 자원 모니터(resource monitor), 그리고 자원 데이터베이스(resource database)로 구성된다.



[그림 1] 그리드 컴퓨팅의 스케줄링 과정

[그림 1]에서 모니터는 네트워크 모니터와 자원 모니터로 구성된다. 네트워크 모니터는 네트워크 전송속도 및 상태를 모니터링하고 자원모니터는 로컬시스템의 상태를 모니터링 한다. 자원 데이터베이스는 모니터링한 정보를 저장한다. 예측모듈은 자원 데이터베이스의 정보를 사용하여 작업의 총 실행시간을 예측한다.

작업의 총 실행시간 $T_{estimated}$는 식(1)과 같이 구한다[5].

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

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[5].

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

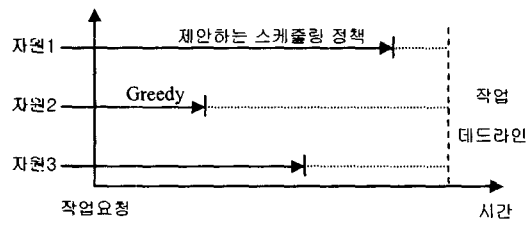
식(2)에서 W_{send_data}와 $P_{send_throughputs}$은 송신시 작업 크기와 처리량이다, W_{recv_data}와 $P_{recv_throughputs}$은 수신시 작업 크기와 처리량이다. $T_{computation}$은 식(3)과 같이 구한다[5].

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

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

작업의 총 실행시간을 예측할 때 자원 데이터베이스의 정보와 네트워크 및 로컬시스템간의 실제 정보는 차이가 있다. 따라서, 정확한 작업의 총 실행시간을 구하기 위해서는 그 차이에 따른 오차를 고려하여 작업의 총 실행시간을 보정해주어야 한다.

스케줄러는 원격 스케줄러와 로컬 스케줄러로 구성된다. 원격 스케줄러는 요청 작업의 총 실행시간을 예측한 후, 그 시간이 최소인 로컬시스템에 작업을 할당한다. 로컬 스케줄러는 할당된 작업의 대기시간과 처리시간을 계산한 후, 그 시간이 데드라인 내에 처리할 수 있으면 로컬시스템에서 처리한다. 반대의 경우 다른 로컬시스템으로 이주시킨다.



[그림 2] Greedy 스케줄링과 제안하는 스케줄링 정책

특히, 실시간 작업 스케줄링에서는 작업실패율과 자원비용을 고려해야 한다[6]. 기존의 Greedy 스케줄링은 작업실패율을 줄이기 위해서 [그림 2]와 같이 작업 완료시간이 최소인 로컬시스템을 선택하여 할당하기 때문에 자원비용이 증가하는 문제점이 있다. 이러한 문제점을 해결하기 위해 본 논문에서는 작업실패율과 자원비용을 최소화 할 수 있는 스케줄링 정책을 제안한다.

3. 제안하는 스케줄링 정책

제안하는 스케줄링 정책은 원격 스케줄러와 로컬 스케줄러에서 2단계 스케줄링을 수행한다. 이 스케줄링 정책은 작업의 데드라인에 근접해 있는 자원을 선택함으로써, 자원비용을 최소화하기 때문에 효율적인 자원할당이 이루어진다.

3.1 원격 스케줄링 정책

원격 스케줄러에서는 네트워크 전송시간에, 작업 크기에 대한 요소를 추가하여 전송시간을 재계산한다. 자원 할당을 위한 원격 스케줄링 정책은 [그림 3]과 같다.

```
// T_target : 작업 데드라인까지의 유효시간
// T_requested : 사용자가 작업실행을 요청한 시간
// T_network_i : 작업을 자원 i와 주고받을 때 걸리는 전송시간
// T_computation_i : 자원 i에서 작업실행시간
// T_estimated_i : 예측한 총 실행시간
// T_resource_i : 데드라인과 총 실행시간의 차이

T_target = T_deadline - T_requested;
for (i=1; i<=n; i++) // n은 자원의 개수
{
   //작업 크기를 고려한 전송시간 가중치 추가 재계산
   T_network_i = T_network_i * (job size / average size of jobs);
   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 3] 원격 스케줄링 정책

[그림 3]에서 T_{target}은 데드라인까지 남아있는 유효시간으로 $T_{deadline}$에서 작업 요청시간 $T_{requested}$를 제외하여 구한다. n개의 로컬시스템을 대상으로 작업을 할당했을 때 예측되는 총 실행시간 $T_{estimated}$을 구한다. 먼저 작업 전송시간 $T_{network_i}$과 실행시간 $T_{computation}$을 구한다. 작업 전송시간 $T_{network_i}$은 작업의 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 로컬 시스템 중에 최소의 $T_{resource_i}$를 가지는 로컬시스템을 선택하여 할당한다. 이러한 할당정책을 사용하면 데드라인에 근접한 자원에 작업을 할당하여 처리함으로써 자원비용을 줄일 수 있다.

3.2 로컬 스케줄링 정책

로컬시스템의 작업할당은 로컬시스템의 부하상태를 고려한다. 원격 스케줄링 정책에 의해 구한 작업의 총 실행시간을 로컬 스케줄러에서 재계산한다. 원격 스케줄러에서 예측한 시점과 할당 시점의 로컬시스템의 상태정보가 다르기 때문이다. 또한 로컬 스케줄러에서는 로컬시스템의 대기 큐에서 대기한 시간을 총 실행시간에 합산하여 총 실행시간을 재계산한다. 로컬 스케줄링 정책은 [그림 4]와 같다.

```
// Testimated_i : 원격 스케줄러에서 예측된 총 실행시간
// Twait_in_queue : CPU 대기큐에서의 대기시간
// Tnetwork_i : 작업을 자원 i에 주고받을 때의 전송시간
// Tsend_data_i: 원격에서 자원 i에 작업을 보낼 때의 전송시간
// Trecv_data_i : 자원 i에서 결과를 원격으로 보낼 때의 전송시간
// Tcomputation_i : 자원 i에서 작업실행시간
// Tresource_i : 데드라인과 총 실행시간과의 차이

//작업에 대한 현 시점의 대기시간 추가
Testimated_i = Testimated_i + Twait_in_queue;
if ( Testimated_i <= Tdeadline )
{
   Execute(job);
} else {
   For (i=1; i<n-1; i++)  // 네트워크 전송시간 재계산
   {
      Tnetwork_i = ((Tsend_data_i*2 + Trecv_data_i)
                        * (job size / average size of jobs));

      Testmated_i = Tnetwork_i + Tcomputation_i;
      Tresource_i = Tdeadline -  Testimated_i;
   }
   Min(job, resource_i);  // 다른 적정 자원 검색, 결정
   if (Tresource_i > 0)
   {
      Migration(job, resource_i);
   } else {
      Exit(fail);  // 데드라인을 초과한 경우
   }
}
```

[그림 4] 로컬 스케줄링 정책

[그림 4]에서 $T_{estimated}$는 원격 스케줄러에서 예측한 총 실행시간에 로컬시스템의 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 로컬시스템으로 이주시킨다. 이때 나머지 n-1개의 다른 로컬시스템에 대한 총 실행시간 $T_{estimated}$을 재계산한다. $T_{estimated}$을 재계산 할 때에 다른 로컬시스템으로 이주하는데 소요되는 네트워크 전송시 간을 추가로 고려해야 한다. 네트워크 전송시간으로는 작업의 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 $T_{send_data_i} * 2$로 구한다. 그리고 원격 스케줄러와 같이 작업크기에 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 실행을 완료하지 못하면 실패한 작업으로 처리한다.

4. 결론 및 향후 연구과제

본 논문에서는 그리드 컴퓨팅에서 실시간 처리를 요구하는 작업에 대해 작업실패율과 자원비용을 고려한 새로운 스케줄링 정책을 제안하였다. 이 스케줄링 정책의 특징은 원격 스케줄러와 로컬 스케줄러를 사용하여 2단계 스케줄링을 수행한다. 원격 스케줄러에서는 자원 데이터베이스에 저장된 네트워크과 로컬시스템의 정보를 사용하여 작업의 총 실행시간이 최소인 로컬시스템을 선택하여 작업을 할당한다. 로컬 스케줄러에서는 할당된 작업의 대기시간과 처리시간을 재계산한 후, 작업을 데드라인내에 처리할 수 있다면 로컬시스템에서 처리한다. 하지만 데드라인을 초과하면 다른 로컬시스템으로 이주시켜 처리함으로써 작업실패율과 자원비용을 최소화한다. 따라서 본 논문에서 제안하는 정책은 기존 Greedy 스케줄링 정책에 비해서 작업실패율과 자원비용을 줄이는 면에서 우수하다.

향후 연구과제는 시뮬레이터를 개발하여 제안된 스케줄링 정책을 검증하는 것이다.

5. 참고 문헌

1) B. Hamidzadeh, "Dynamic scheduling of real-time aperiodic tasks on multiprocessor architectures," System Sciences, Proceedings of the Twenty-Ninth Hawaii International Conference, Vol. 1, pp.469-478, Jan. 1996.

2) V. Subramini, R. Kettimuthu, and S. Srinivasan, S. Sadayappan, "Distributed job scheduling on computational 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) Junwei Cao, D.J. Kerbyson, and G.R. Nudd, "Performance evaluation of an agent-based resource management infrastructure for grid computing," Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium, pp. 311-318, May. 2001.

5) 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.

6) Ching-Chih Han, K.G. Shin, and Jian Wu, "A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults," Computers, IEEE Transactions, Vol. 52, Issue. 3, pp. 362-372, Mar. 2003.