그리드 컴퓨팅 환경에서 효율적인 자원 활용을 위한 성능 계량 모델 및 자원 선택 알고리즘 제안

이준돈[O] 정윤미[O] 길아라* 윤현주**
[O]숭실대학교 대학원 컴퓨터학과
*숭실대학교 컴퓨터학부
**한국과학기술원 전산학과
{darthvader[O],bopeep[O]}@archi.ssu.ac.kr ara@computing.ssu.ac.kr* juyoon@camars.kaist.ac.kr**

A Performance Measurement Model and Resource Selection Algorithm for Efficient Resource Utilization in Grid Computing

Joohn Dhon Yeeh[O] Youn Me Jung[O] Ara Khil* Hyeon-Ju Yoon**
[O]Dept. of Computer Science, Soong Sil University
*School of Computer Science, Soong Sil University **CS Dept., KAIST

요 약

그리드 컴퓨팅은 네트워크 상의 유휴 자원 및 다수의 저성능 자원을 활용함으로써 보다 고 성능의 컴퓨팅 환경을 요구 하는 응용 문제를 해결할 수 있다. 따라서, 그리드 컴퓨팅의 자원 관리 시스템의 자원 선택 및 할당 기능은 주어진 응용 문제에 대하여 보다 높은 성능의 그리드 컴퓨팅 환경을 제공하기 위한 매우 중요한 요소이다.

본 논문에서는 보다 효율적으로 자원을 선택하기 위하여 환경 내 자원들의 종합적인 CPU 성능을 평가하는 UC 단위 모델을 제안하고, 보다 효율적인 자원 할당을 위하여 그리디 방식(Greedy Method)을 변형한 최적 자원 우선(Best-Fit-First) 알고리즘을 제안한다. 또한, 기존의 자원선택, 할당방식과 비교하는 모의실험을 통하여 제안하는 모델 및 알고리즘의 향상된 성능을 나타내 보인다.

1. 서론

그리드 컴퓨팅(Grid Computing)이란, 지리적으로 분산된 고성능 컴퓨터, 대용량 저장 장치 및 데이터베이스, 첨단 실험 장비 등의 자원들을 고속 네트워크에 연결해 상호 공유, 이용할 수 있도록 하는 기반구조이다.[1]

그리드 컴퓨팅의 기본 개념은 여러 대의 저성능 컴퓨터 자원을 통합하여 고성능 컴퓨팅 환경을 구축하는 것이다. 이러한 통합된 환경을 효율적으로 관리하기 위해서는 자원관리 시스템(Resource Management System)의 역할이 중요하다.[2]

자원 관리 시스템의 기능 중, 자원의 요청에 대해 효율적으로 자원을 선택하여 할당하는 기능은 자원 관리 및 작업 실행 성능에 큰 영향을 미치고, 나아가 그리드 컴퓨팅 전체의 성능에도 영향을 준다. 본 논문에서는 기존의 잘 알려진 그리드 시스템에서 자원 선택 알고리즘을 개선하는 방법을 제시하였다.

자원의 요청과 선택에 앞서, 각 자원의 성능 또는 용량을 어떻게 평가하고 표현하는가도 역시 중요한 요소다. 용량이 정확하게 표현되는 메모리나 저장 장치와는 달리, CPU 의 성능이나 네트워크의 성능은 하나의 자원 요소뿐만 아니라 그를 둘러싼 다른 자원에 의해서도 그 성능이 영향을 받는다.

정확한 자원의 성능 평가를 통한 효율적인 자원 선택을 하기 위해서는 각 자원의 전체적인 성능을 평가할 수 있는 기준이 필요하다.

본 논문에서는 우선 CPU 의 성능을 종합적으로 평가하여 표현하는 UC (Unit of CPU) 단위 모델을 제시하고, UC 단위로 평가된 자원들을 요청되는 작업에 대해 효율적으로 할당할 수 있도록 그리디 방식을 변형한 최적 자원 우선 알고리즘을 제안한다.

본 논문의 구성은 다음과 같다. 2 장에서는 UC 단위 모델에 대해 설명하고, 3 장에서는 기존 시스템의 자원 선택 알고리즘을 살펴본 후, 이들의 문제점을 해결하기 위한 최적 자원 우선 알고리즘에 대해 설명한다. 4 장에서는 시뮬레이션 및 결과를 보이고 5 장에서 결론을 맺는다.

2. UC 단위 모델을 통한 자원의 성능 평가

일반적으로, 계산 그리드(Computational Grid) 에서 컴퓨터 성능은 CPU 의 처리속도가 주요 요소로 평가되나[3], 같은 처리 속도의 CPU 라고 해도 L1, L2 cache 의 존재 유무 및 용량, 메모리의 가용량, 하드디스크의 작업속도, 운영체제 등에 따라 CPU 의 작업 처리 성능은 다르며, 여러 대의 컴퓨터 성능은 단순히 CPU 처리속도를 산술 연산하여 평가하는 것이 불가능하다. 본 논문에서는 이러한 성격의 CPU 성능을 보다 정확하게 평가하기 위한 방법을 제시하고, 그에 따라 도출된 UC 단위를 CPU 성능 표현에 쓸 것을 제안한다.

UC 단위 모델은 작업 J가 시간 t 안에 실행 완료 할 수 있는 컴퓨터 CPU의 성능으로 정의한다.

작업 J 가 컴퓨터 A 에서 실행완료 시간이 1초 이고, 컴퓨터 B 에서 실행완료 시간이 2초 라고 가정하면, 컴퓨터 A는 2UC, 컴퓨터 B는 1UC 라고 할 수 있다. 또한 작업 J 가 실행되는데 10UC 가 필요하다면 5UC 의 컴퓨터 2대를 이용해 작업을 마칠 수 있다.

작업 J 는 정수 연산, 부동소수점 연산 등을 종합적으로 수행하는 에이전트 어플리케이션 형태로 구현 할 수 있으며, 그

리드 컴퓨팅 환경이 HPC 환경인지, HTC 환경인지, 혹은 계산 그리드 환경인지 데이터 그리드 환경인지에 따라 자원 평가 시 각 요소에 우선순위를 부여할 수 있다.

각 자원에는 에이전트 어플리케이션이 존재하고, 에이전트 어플리케이션은 각 자원이 유휴상태에 들어갈 때 성능평가를 실시하여 자원 관리 시스템으로 결과를 보고하는 역할을 한다.

3. 자원 선택 알고리즘

3.1 관련 연구

Condor System 은 대용량 자료의 장시간 처리를 위한 그리드 환경으로써 연구 되었다. Condor System 의 자원관리 방법은, 각각의 자원이 자신의 상태를 광고한 내용과, 요청된 자원을 평가하여 서로 일치하는 것을 할당하는 ClassAds Match-Making 을 이용하고[4] 있으나, 자원 요청에 대해 하나의 자원만 관리하는[5] 구조적 한계가 있다.

RedLine System의 경우, ClassAds Match-Making 을 개선한 set-extended ClassAds 스크립트와 set-matching algorithm[2]을 이용하여, 자원 요청에 따라 다수의 자원을 통합 관리하여 Condor System의 한계를 극복하고 있으며, 자원 관리에 있어 Greedy 알고리즘을 사용한다[5]. 자원 요청이 들어오면 가용 자원 중 가능한 고성능 자원부터 할당을 하기 때문에 상대적으로 저 성능의 자원은 사용될 기회가 적으며, 고성능 컴퓨터의 잉여 자원을 쓸 수 있도록 하지 않으면 자원 낭비의 우려도 있다.

3.2 최적 자원 우선 알고리즘

본 논문에서는 2장에서 제시한 UC 단위 모델을 기준으로 저 성능 자원을 보다 많이 활용하여 전체 시스템의 성능을 개선할 수 있도록, 그리디 방식을 변형한 최적 자원 우선 알고리즘을 제안한다. 자원 선택 시 고려해야 할 자원의 서술 방법이나 종류 등은 UC로 표현된 CPU 성능에만 국한한다

최적 자원 우선 알고리즘은 요청된 작업의 UC(RQ) 보다 작은 UC값을 갖는 자원 중, 가장 큰 UC값을 갖는 자원(BigOne)을 찾고, 부족한 자원은, 가능한 작은 UC값을 갖는 자원을 여러 개 조합하여 요청을 만족한다. RQ와 유사한 UC 값을 먼저 찾지 않고, 가장 작은 UC값부터 조합하여 RQ 값을 만족하는 방법은, RQ 가 큰 값일 때 성능 저하가 발생하기 때문에, 가장 유사한 UC 값을 먼저 찾는다.

최적 자원 우선 알고리즘은 RQ를 만족하는 candidate_List 구성에 대해 3가지 operation을 갖는다.

1) GetSumResouce : 각 자원의 UC 값에 대해 중복에 상관없이, BigOne 보다 작은 자원들의 합을 구한다.

2) SumResource < RQ 의 경우 ADD operation

candidate_List 에 자원을 추가한다. 추가되는 자원이 최근에 Drop 되었던 자원과 UC값이 같을 경우, candidate_List 에 추가 한 후, 해당 자원 보다 작은 UC값을 갖는 자원 중 가장 큰 UC값의 자원 하나를 삭제한다. 이때, 자원이 존재하지 않을 경우, 바로 이전의 candidate_List로 으로 결정한다.

3) SumResource > RQ 의 경우 DROP operation

최근 추가된 머신 보다 작은 UC 값을 갖는 자원들 중 가장 큰 UC값을 갖는 자원을 candidate_List 에서 하나 삭제한다. 이때, 자원이 존재하지 않을 경우, 바로 이전의 candidate_List로 결정 한다.

자원 배열이 Descending sort 되어 있다고 가정 할 경우,

RQ = BigOne + M_n + M_{n-1} + M_{n-2} + ... 로 나타낼 수 있다.

자원 M1, M2, M3, M4, M5, M6 의 UC 값이 4, 3, 3, 2, 2, 1 로, 중복하여 존재하는 자원의 집합, MachineSet= [M1(4), M2(3), M3(3), 4(2), M5(2), M6(1)] 에서 요청된 작업의 UC 값이 8일 경우, 그리디 방식으로써 해를 찾으면,

RQ(8) : M1(4) + M2(3) + M3(3)

과 같이, UC값이 큰 자원 위주로 해를 찾지만, 최적 자원 우선 알고리즘은,

RQ(8) : M1(4) + M6(1)+ M5(2)+ M4(2) 까지 ADD Operation이 발생한 후, RQ(8) 보다 SumResource(9) 가 크므로 DROP Operation 이 발생하여 M6(1) 자원이 DROP 된다. 그 결과,

RQ(8) : M1(4) + M5(2) + M4(2) 로 구성되며, 그리디 방식보다 정확히 RQ.(8)을 만족한다.

최적 자원 우선 알고리즘의 주요 동작을 의사코드로 나타내면 그림 1 과 같다.

```
Candi_List BFF_Algorithm(RQ, Mac_Pool[], Candi_List[])
{
  BigOne = FindBigOne()
  SumResource = SumResource + BigOne.uc
  for(i =0 ; i<BigOne.index; i++)
  {
    SumResource = SumResource + Mac_Pool[i].uc
    if(SumResource == RQ)
    {
      Add Mac_Pool[i] to Candi_List
      Return Candi_List
    }else if(SumResource < RQ) //ADD Operation
    {
      Add Mac_Pool[i] to Candi_List
      If(Mac_Pool[i] == Recently_DropMac)
      {
        tempMac = FindSmallMac(Mac_Pool[i])
        result = Delete_fromCandiList(tempMac)
        if(!result) return Candi_List
      }
    }else if(SumResource > RQ) //DROP Operation
    {
      tempMac = FindSmallMac(Recently_AddMac)
      result = Delete_fromCandiList(tempMac)
      if(!result) return Candi_List
    }
  }
  return NULL
}
```

그림 1 최적 자원 우선 알고리즘 의사 코드

3.3 그리디 방식과 최적 자원 우선 알고리즘의 차이

그리디 방식 와 최적 자원 우선 알고리즘의 자원 선택시 차이를 알아보기 위한 시뮬레이션으로써, 작업 J1, J2, J3 를 각각 임의의 시간 1T, 3T, 6T 에 요청한 결과이다. 각 작업은 다음과 같이 설정 되었다.

J1 : 70UC, Execution Time 8T

J2 : 55UC, Execution Time 5T

J1 : 32UC, Execution Time 3T

그림 2-1과 같이, 그리디 방식 의 경우 작업 J1, J2 까지는 요청된 작업에 필요한 자원을 할당 하였으나, 작업 J3 가 요청된 순간, 필요한 자원의 UC 값이 32 인데 반해, 사용할 수 있는 총 자원이 30UC 이므로 자원을 할당 받지 못하였다.

그리디 방식

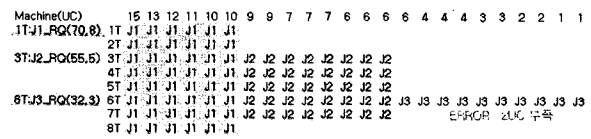

그림 2-1 그리디 방식 VS. 최적 자원 우선 알고리즘

작업 J1, J2 도 자원을 할당 받기는 하였으나, 각각 1UC,

2UC가 낭비되어 J3가 자원을 할당받지 못하는 원인이 되었다.

그림 2-2 그리디 방식 VS. 최적 자원 우선 알고리즘

그림 2-2 의 최적 자원 우선 알고리즘의 경우, 작업 J3에 자원 할당 시 1UC의 낭비가 발생하였으나 모든 작업 J1, J2, J3에 필요한 자원을 모두 할당 하였으며, 작은 자원 위주로 할당한 것을 알 수 있다.

사용된 자원 수 에서도 그리디 방식과 최적 자원 우선 알고리즘은 많은 차이를 보인다. 그리디 방식 의 경우 14대의 컴퓨터 자원을 사용하였으나, 최적 자원 우선 알고리즘의 경우 24대의 컴퓨터 자원을 사용함으로써, 저성능 컴퓨터 여러 대를 통합하여 고성능 컴퓨팅 환경을 구축하는 그리드 컴퓨팅 기본 개념에 부합 하고 있다.

4. 시뮬레이션 및 결과

최적 자원 우선 알고리즘의 성능 평가를 위해, 가상의 31대 컴퓨터 자원을 대상으로 다음과 같은 가정 하에 시뮬레이션 하였다.

- 작업 J 는 여러 대의 컴퓨터 자원에서 실행가능하고 그에 따른 성능저하는 없다
- 성능평가 보고 및 자원요청시 Network관련 부하는 없다
- 각 자원들은 50UC(1), 48UC(1), 43UC(2), 40UC(3), ..., 15(20),..., 4UC(3), 3UC(2), 2UC(1), 1UC(1) 등과 같이 정규 분포를 따른다.
- 자원 요청은 무작위(random)로 발생한다
- 동시에 자원을 요청하는 경우는 배제한다

작업의 실행 시간은 3T에서 8T까지 무작위(random)로 설정 되고 1 UC 에서 200 UC 사이의 값을 갖는 작업을, 무작위로 요청하는 시뮬레이션을 1000T 동안 실시하였다.

그림 3은 1000T 동안의 시뮬레이션을 10회 실시하여 각 자원들의 평균 사용 회수를 나타낸 그래프 이다.

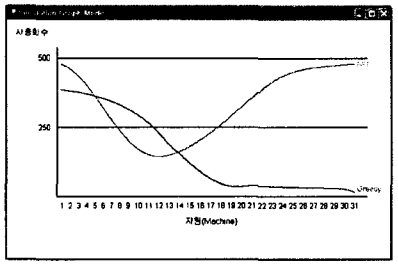

그림 3 최적 자원 우선 알고리즘 시뮬레이션

그리디 방식의 경우, 요청된 작업에 대해 자원이 얼마나 적합한지는 고려하지 않고, 충족되기만 하면 자원을 할당하기 때문에 M1, M2 와 같은 고성능 자원의 사용이 두드러지고 저성능 자원의 사용이 거의 없으나, 최적 자원 우선 알고리즘의 경우 고성능의 M1을 BigOne 으로 설정하고, 저성능의 M31 부터 조합하여 부족한 UC를 충족하므로 저성능의 자원 위주로 사용됨을 알 수 있다. 또한, 3-3 에서 보인 바와 같이, 그리디 방식보다 자원할당오류가 발생할 여지가 적으므로 고성능 자원인 M1의 사용 회수도 높게 나타났다. 이런 특성으로 인해, 큰 UC값의 자원 요청이 발생 했을 때 그리디 방식에 비해 매우 효율적으로 자원 사용을 가능하게 한다.

그림 4는 시뮬레이션 하는 동안 전체 자원에 대해 작업에 할당된 자원을 평균 사용률로 나타낸 그래프이다. 저성능 자원을 활용 함으로써 전체적인 자원 사용률도 높음을 알 수 있다

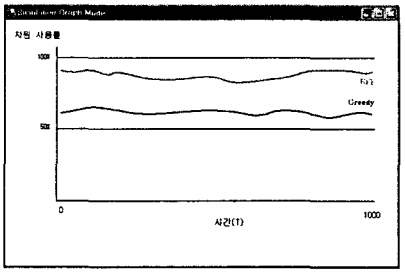

그림 4 자원 사용률

5. 결론 및 향후 과제

본 논문에서는 UC 단위 모델을 제안하고 UC 단위로 평가된 자원들을 최적 자원 우선 알고리즘을 사용하여 요청된 작업에 할당함으로써, 저성능의 컴퓨터를 통합하여 고 성능의 컴퓨팅환경 구축하는 것을 시뮬레이션 하였다.

최적 자원 우선 알고리즘이 여러 대의 컴퓨터 자원을 조합하여 요청을 만족 하는 특성을 통해, Condor System의 ClassAd Match-Making 의 자원과 요청의 1:1 할당문제를 해결하였으며, RedLine System의 Extended Set Match 에서 자원 할당에 방식에 사용된 그리디 방식을 변형하여 보다 효율적으로 저성능 컴퓨터를 활용 할수 있는 가능성을 확인하였다.

향후에는, UC 단위 모델에서, 보다 정확한 CPU 자원 평가 방법과 종합적인 자원의 성능을 평가할 수 있는 기준과 에이전트 어플리케이션의 효율적인 네트워크 사용에 대한 연구가 필요하며, ClassAds 스크립트와 같이, 최적 자원 우선 알고리즘을 적용 할 수 있는 스크립트 엔진에 대한 연구가 지속적으로 필요하다.

6. 참고문헌

[1] The Anatomy of the Grid: Enabling Scalable Virtual Organizations. I. Foster, C. Kesselman, S. Tuecke. *International J. Supercomputer Applications*, 15(3), 2001.

[2] Condor-G: A Computation Management Agent for Multi-Institutional Grids. J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke. *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.

[3] Computational Grids., I. Foster, C. Kesselman. *Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure"*, Morgan-Kaufman, 1999.

[4] Litzkow, M. Livny, & M. Mutka, Condor - A hunter of idle workstations. *In Proceedings of the Eighth Conference on Distributed Computing Systems*, San Jose, California, June 1988.

[5] Design and Evaluation of a Resource Selection Framework for Grid Applications. D. Angulo, I. Foster, C. Liu, and L. Yang. *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002.