PC 기반 그리드 환경에서 저 성능 자원의 활용도 향상을 위한 에이전트 기반 자원 관리 시스템 구현

이준돈[O] 길아라*
숭실대학교 대학원 컴퓨터학과
darthvader@archi.ssu.ac.kr[O], ara@computing.ssu.ac.kr*

An Implementation of Agent based Resource Management System for Improving Low-Performance Resource Utilization in PC based GRID

Joohn Dhone Yeeh[O] Ara Khil*
Dept. of Computer Science, Soong Sil University

요 약

그리드 컴퓨팅의 기본 개념은 여러 대의 저 성능 컴퓨터 자원을 통합하여 고성능 컴퓨팅 환경을 구축하는 것이다. 이런 환경을 관리 하기 위해서는 요청된 작업에 대해 자원 관리 시스템의 효율적인 자원 할당 기능이 중요하다. 본 논문에서는 효과적인 자원의 선택을 위해 CPU의 종합적인 성능을 평가하는 UC 단위 모델을 제안하고, 제안된 모델을 기준으로 자원관리 시스템에서 저 성능 컴퓨터 자원을 효율적으로 할당 할 수 있도록 저 성능 자원 우선 알고리즘을 제안하며, 이를 이용한 에이전트 기반 자원관리 시스템을 구현한다.

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 정보를 공유하거나, 동일한 문제 처리를 위해, 물리적인 위치에 구애 받지 않는 자원 교류와 협업 환경을 실현하는 것이다.[1]

그리드 컴퓨팅의 기본 목적은 여러 대의 컴퓨터 자원을 통합 관리하여 고성능 컴퓨팅 환경을 구축하는 것에 있다. 계산 그리드(Computational Grid)는 과학 분야 및 엔지니어링 분야의 고속 계산이 필요한 다양한 문제를 신속히 해결하기 위한 분야에 적용되며, 데이터 그리드(Data Grid)는 대용량 데이터를 분산 저장하여 처리하기 위한 분야에 적용된다. 특히, 계산 그리드는 유휴 자원을 최대한 이용하여 가능한 빠른 시간 안에 계산 결과를 얻어야 하므로 효율적인 자원관리 시스템(Resource Management System)의 역할이 중요하다.[2]

개인용 컴퓨터(이하 PC) 기반 그리드는 기존의 그리드 컴퓨팅 환경과 달리, 일반 사용자가 범용적으로 사용하는 PC 를 이용한다. 사용자가 PC 를 사용하지 않는, CPU 유휴상태의 자원을 검색하여 계산 그리드의 한 노드로써 활용을 한다. 이미 존재하는 범용 PC 를 사용함으로써, 기존의 그리드 환경과 달리, 물리적 환경을 구축함에 있어 수월 하고 비용이 적다는 장점이 있다. 그러나 중앙 집중형 관리가 불가능 하여, 각 자원(PC)의 유휴 상태에 대한 관찰과 관리가 필요하고 유휴자원 검색 및 활용이 어려워지는 단점이 있다. 이러한 단점을 본 논문에서는 각 자원에 에이전트 어플리케이션을 두고 에이전트 어플리케이션을 통한 자원 관리 시스템을 구현하여 해결한다.

자원 관리 시스템의 기능 중, 작업의 자원 요청에 대해 효율적으로 자원을 선택하여 해당 작업에 할당하는 기능은 자원의 관리 및 작업 실행 성능에 큰 영향을 미치고, 나아가 계산 그리드 전체의 성능에도 영향을 미친다.

일반적으로, 계산 그리드에서 컴퓨터 성능은 CPU 의 처리속도가 주요 요소로 평가 되나[3], 같은 처리 속도의 CPU 라고 해도 L1/L2 캐시(cache)의 존재 유무 및 용량, 메모리(Ram)의 가용 량, 하드디스크(Hard Disk)의 동작속도, 운영체제 등과 같은 여러 주변 환경 요소에 따라 CPU 의 작업 처리에 대한 결과는 다르다. 또한 여러 대의 PC(자원) 성능은 단순히 CPU 처리속도를 가산하는 방법으로, 산술연산 평가하는 것이 불가능 하다. 그렇기 때문에, 효율적으로 자원 선택을 하기 위해서는, 성능 평가를 통해 자원에 대한 정확한 정보를 파악해야 하며, 자원의 전체적인 성능을 평가할 수 있는 기준이 필요하다.

본 논문에서는 자원의 성능을 상대적으로 평가하는, 성능평가 모델 UC 단위를 제안한다. 그리고 UC 단위를 이용하여, 작업 할당 시, 등한시 될 수 있는 저 성능 자원을 조합하여 효율적인 작업 처리가 가능한, 저성능 자원 우선 알고리즘을 제안하고, 에이전트 어플리케이션을 이용하여 Globus 상에서 구현한 자원 관리 시스템이, 알고리즘을 통해 자원을 효율적으로 선택하고 조합하는 것을 실험 결과로 보인다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구에 대하여 설명하며, 3 장 에서는 에이전트 어플리케이션(Agent Application)과, UC 단위 모델을 제안하고, 에이전트 기반 자원 관리 시스템 구현 방법에 대해 설명하며, 4 장에서는 저성능 자원 활용에 대한 실험결과를 나타내며 5 장에서 결론을 맺는다.

2. 관련 연구

일반적인 그리드 환경과 PC기반 그리드의 운영상 차이점은, 대부분의 그리드 환경은 순수연구를 목적으로 하기 때문에 연구관련 작업(계산작업) 외, 기타 작업에는 자원(CPU)사용이 거의 없으나, PC기반 그리드는 기본적으로 사용자의 업무처리를 위해 자원이 사용되다가 자원의 유휴 상태에 따라 제공된다는 것이다. 또한, 그림 1과 같이, 그리드 환경에서 그리드 어플리케이션으로 표현되는 작업의 흐름상으로도 차이가 있다.

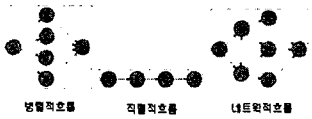

그림 1: 그리드 상에서 작업의 흐름

병렬적 흐름의 경우 각 작업은 다른 작업의 결과에 영향을 받지 않기 때문에, 작업 완료시간은 문제가 되지 않는다. 그러나 직렬적 흐름 및 네트웍적 흐름은 다른 작업의 결과를 바탕으로 실행을 해야 하기 때문에, 이전 작업이 완료되지 않으면 작업을 진행 할 수 없다. PC 기반 그리드의 경우 병렬적 흐름 형태의 작업을 대상으로 한다.

PC 기반 그리드의 대표적인 프로젝트는 SETI@Home: Search for Extra Terrestrial Intelligence로써, 400만 대 이상의 PC가 참여하는 대규모 프로젝트 이다.

3. 저 성능 자원의 효율적 활용

3.1 UC단위 모델

계산 그리드에서 그리드 환경을 구성하고 있는 각 자원은, 실행되는 작업의 일부를 할당 받아 처리하므로 그리드 환경 전체의 성능과 밀접한 연관이 있다. 그러므로 효율적인 스케줄링을 지원하기 위해서는 자원 관리 시스템이, 동적으로 변화하는 각 자원의 성능을 정확히 파악할 수 있어야 하며, 파악된 각 자원의 성능은 절대 평가의 기준뿐만 아니라, 전체 자원 성능 대비 어느 정도의 성능을 갖고 있는지를 상대적으로 평가 할 수 있는 기준이 필요하다.

본 논문에서는 정확한 자원의 성능을 평가하기 위하여, 전세계 슈퍼컴퓨터의 성능 순위를 판정 할 때 쓰이는 Linpack 벤치마크를 이용 한 후, 자원의 유휴 CPU 이용률(Free CPU Load)을 적용하는 성능평가 모델인 UC 단위 모델을 제안한다.

UC 단위 모델은 작업 J가 임의의 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 표현 할 수 있다. 또한 작업 J 가 실행되는데 10UC 가 필요하다면 5UC 의 컴퓨터 2대를 이용해 작업을 마칠 수 있다

각 자원의 성능을 상대적 평가하기 위해, CPU=Pentium4-2.66GHz, RAM=256MB, 등의 하드웨어로 구성된 PC에 설치된 Linux 환경에서 커널 재 컴파일을 통해 불필요하게 CPU를 소모할 수 있는 모듈을 제거한 후, Linpack 벤치마크 java 버전으로 100회 벤치마킹한 평균값 45.778 Mflops/s 을 기준으로 하며, 이 값을 1000UC로 정한다.

유휴 CPU율은 Mds-Cpu-Total-Free-15minX100: 096 과 같이 Globus 의 GRAM 에서 제공하는 정보를 사용하며, 현재를 기준으로 15분 전까지의 평균 유휴 CPU율이다. Linpack 벤치마킹 결과가 45.121 Mflops/s 의 성능인 어떤 자원의 유휴 CPU율이 70% 일때 UC단위로 나타내면,

(42.121*1000UC)/45.778 = 920.114 을 통해,

920.114*70% = 644.080UC 로 나타낼 수 있다.

3.2 저 성능 자원 우선 알고리즘

저 성능 자원 우선 알고리즘은 UC 단위 모델을 기준으로 저 성능 자원을 보다 많이 활용하여 전체 시스템의 성능을 개선할 수 있도록 한다. 자원 선택 시 고려해야 할 자원의 서술 방법이나 종류 등은 UC로 표현된 CPU 성능에만 국한한다

저 성능 자원 우선 알고리즘은 요청된 작업의 UC(Rq_UC)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. Rq_UC 와 유사한 UC 값을 먼저 찾지 않고, 가장 작은 UC값부터 조합하여 Rq_UC 값을 만족하는 방법은, RQ 가 큰 값일 때 성능 저하가 발생하기 때문에, 하나의 가장 유사한 UC 값을 먼저 찾는다.

저 성능 자원 우선 알고리즘의 의사코드는 다음 그림 2와 같다.

```
입력 :    자원 집합 M={ M1, M2, M3, … , Mn}에 대한
          각 UC 값 Mac_Pool[n].uc
          작업J의 실행에 대한 요청 UC 값 Rq_UC
출력 :    Rq_UC 를 만족하는 자원 집합 Candi_List
LPF_Algorithm()
{
  BigOne = FindBigOne()
  SumResource = SumResource + BigOne.uc
  for(i =0 ; i<BigOne.index; i++)
  {
    SumResource = SumResource + Mac_Pool[i].uc          (1)
    if(SumResource == Rq_UC)
    {
      Add Mac_Pool[i] to Candi_List
      Return Candi_List
    }else if(SumResource < Rq_UC)                       (2)
    {
      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_UC)                       (3)
    {

      tempMac = FindSmallMac(Recently_AddMac)

      result = Delete_fromCandiList(tempMac)

      if(!result) return Candi_List

    }
  } // for loop
} // Func End
```

그림 2: 저 성능 자원 우선 알고리즘의 의사코드

최적 자원 우선 알고리즘은 Rq_UC 를 만족하는 자원들의 집합 Candidate_List 를 구성하기 위해 3가지 과정으로 동작한다.

(1) GetSumResouce : BigOne 을 제외한 각 자원의 UC 값

을 합한 결과 값, SumResource 에 따라, ADD 동작 및 DROP 동작을 수행한다.

(2) SumResource < Rq_UC 의 경우 - ADD 동작
Candidate_List 에 자원을 추가한다. 추가되는 자원이 최근에 Drop 되었던 자원과 UC값이 같을 경우, Candidate_List 에 추가 한 후, 해당 자원 보다 작은 UC 값을 갖는 자원 중 가장 큰 UC값의 자원 하나를 삭제한다. 이때, 자원이 존재하지 않을 경우, 바로 이전의 Candidate_List 로 결정 한다.

(3) SumResource > Rq_UC 의 경우 - DROP 동작
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_List 로 결정 한다.

각 자원의 UC 값이 내림차순 정렬 되어 있다고 가정 할 경우, $Rq\_UC = BigOne + M_n + M_{n-1} + M_{n-2} + \ldots$ 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 값이 8일 경우, 그리디 방식으로써 해를 찾으면, Rq_UC(8) : M1(4) + M2(3) + M3(3) 과 같이, UC값이 큰 자원 위주로 해를 찾지만, 저성능 자원 우선 알고리즘은, Rq_UC(8) : M1(4) + M6(1)+ M5(2)+ M4(2) 까지 ADD 동작이 발생한 후, Rq_UC(8) 보다 SumResource(9) 가 크므로 DROP Operation 이 발생하여 M6(1) 자원이 DROP 된다. 그 결과, Rq_UC(8) : M1(4) + M5(2) + M4(2) 로 구성되며, 그리디 방식 보다 저성능 자원을 효율적으로 사용 하여 정확히 Rq_UC(8)을 만족한다.

3.3 자원관리 시스템의 구현

그림 3과 같이, 각 자원의 성능을 평가 하기 위해 각 자원마다 Globus 미들웨어(Globus Tool Kit 3.2) 상에서 실행되는 에이전트 어플리케이션을 JavaCoG으로 구현하였으며, 에이전트 어플리케이션은 각 자원의 현재 성능을 평가하여 주기적으로, 또는 자원의 유휴 상태에 따라 자원 관리 시스템으로 보고한다.

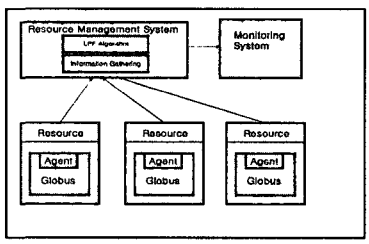

그림 3: 자원 관리 시스템의 구현

일반적인 사용자가 업무처리를 하는 도중, 자원의 유휴 상태에 따라 자원을 제공하는 경우 지속적인 관찰이 필요하기 때문에, 에이전트 어플리케이션에서는 언제 자원관리 시스템에 보고하여 자원을 제공할지, 기준이 필요하며, 사용자가 그 기준을 설정 할 수 있도록 한다. 에이전트 어플리케이션은 설정 파일의 내용중 report_freecpu 항목의 설정 값을 읽어 유휴 CPU이용률이 설정값 이상인 경우, 자원관리 시스템에 보고하여 자원을 제공한다.

모니터링 프로그램은 전반적인 운용에 대한 정보를 제공하며, 실험 결과 확인에 사용된다.

4. 자원 관리 시스템의 성능평가

성능 평가를 위해, 100Mbps 네트웍으로 연결된 PC 7대의 자원을 대상으로 다음과 같은 환경에서 실험 하였다.

- 작업 J 는 여러 대의 machine 에서 실행가능하고 그에 따른 성능저하는 없는 것으로 가정한다.
- 1개~10개까지의 작업 J가 임의로 자원을 요청 할수 있다
- 자원 요청은 10,000UC~100,000UC 의 값을 갖는다.
- 각 PC자원들은 실험이 반복될 때마다 2,000UC ~ 20,000UC의 정규분포를 따르는 UC값으로 임의 설정된다.

실험은 1000회 반복하여 시행하고, 각 자원(PC) 사용횟수를 그림 4로 나타내었다.

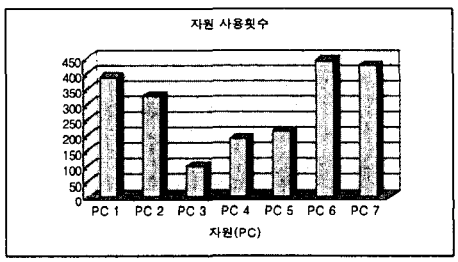

그림 4: 자원 사용횟수

저 성능 자원 PC6과 PC7의 사용회수가 높아, 저 성능 자원 위주의 자원 선택 및 조합이 이루어 진 것을 알 수 있다. PC1과 PC2는 BigOne 으로 선택되어 사용 횟수가 높다.

5. 결론

본 논문에서는 UC 단위 모델을 제안하고 UC 단위로 평가된 자원들을 저 성능 자원 알고리즘을 사용하여 요청된 작업에 할당함으로써, 저 성능의 컴퓨터를 통합하여 고 성능의 컴퓨팅환경 구축하는 것을 보였다. 향후에는, UC 단위 모델에서, 보다 정확한 자원 평가 방법과 연동 될 수 있는 스케줄러에 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] 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.

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