# 그리드에서 균등 분배 MPI 병렬 프로그램을 위한 자원 선택 알고리즘

이원재[O]* 이상권** 임민열*** 맹승렬** 조정완**
* 한국전자통신연구원, ** 한국과학기술원 전자전산학과 전산학전공, *** 한국과학기술정보연구원
russell@etri.re.kr[O], sklee@camars.kaist.ac.kr, mylim@hpcnet.ne.kr, {maeng, jwcho}@camars.kaist.ac.kr

# Resource Selection Algorithm for Uniformly Partitioned MPI Parallel Programs on the Grid

Wonjae Lee[O]*, Sang-Kwon Lee**, Minyoul Lim***, Seungryoul Maeng**, Jung-Wan Cho**
*Electronics and Telecommunications Research Institute
**Div. of Computer Science, Dept. of EECS, KAIST
***Korea Institute of Science and Technology Information

요 약

고속 네트워크의 등장으로 관리 영역을 초월한 계산 자원의 공유가 가능하게 되었고, 그리드 컴퓨팅이 등장하였다. 그리드 환경에 포함된 각 자원들은 이질적이고, 이질적인 환경에서 고성능을 얻기 위해서는 효과적인 자원 발견 및 자원 선택이 중요하다. 본 논문에서는 균등 분배 MPI 프로그램들에 대한 성능 예측 알고리즘과 자원 선택 알고리즘을 제안한다. 성능 예측 알고리즘은 자원의 이질성, 네트워크 성능, 복수 노드에 있는 CPU 부하, 응용 프로그램의 특성을 고려해 성능을 예측 한다. 자원 선택 알고리즘은 k개의 후보 자원 집합들을 생성한 후, 이들 중 최적의 집합을 선택한다. 이를 통해 기존 그리디 알고리즘의 약점이던 지역성을 극복했다.

## 1. 서 론

최근 들어 Grand Challenge Problems와 같이 대량의 계산을 필요로 하는 응용 프로그램들이 개발되었다. 이런 프로그램들을 실행하기 위해서는 여러 대의 슈퍼컴퓨터를 동시에 사용할 필요성이 있다. 고속 네트워크를 통해 지리적으로 떨어진 계산 자원들이 통합된 시스템을 Metacomputing System 혹은 계산 그리드(Computational Grid)[1]라고 부른다.

계산 그리드를 구성하기 위해서는 이질적인 자원들을 투명하게 활용할 수 있게 하는 미들웨어가 필요하다. Globus[2]와 같은 미들웨어가 개발됨으로써 그리드 상의 여러 계산 자원들을 사용하는 응용 프로그램의 개발이 가능해 졌다.

이질적인 그리드 환경에서는 동종의 구성요소들로 이루어졌던 기존의 시스템과 달리 자원 발견 및 자원 선택 문제가 중요하게 된다. 응용 프로그램의 특성에 맞지 않는 자원들을 선택하였을 경우, 성능이 매우 떨어질 수 있다. 따라서 자원 선택 기능은 미들웨어에 포함되어야 하지만, Globus는 자원 선택 기능을 제공하지 않는다. set-extended ClassAds언어와 set-matching algorithm[3]의 경우 set-expression을 통해 병렬 프로그램을 위한 자원 선택을 한다. 그러나 set-extended ClassAds는 프로그램의 성능을 정밀하게 모델링하기에는 표현력이 부족하고, 복잡한 모델링이 필요하다. Jaspal Subhlok[4] 등은 병렬프로그램을 위한 네트워크 상의 자원 선택 연구를 수행하여, 노드와 네트워크의 사용가능성(availability)에 따라 자원 선택을 하는 알고리즘을 제안하였다. 이 알고리즘은 자원의 이질성을 고려하지 않고, 응용 프로그램의 특성을 반영하기 어려운 단점이 있다.

본 논문에서는 계산 집약적이고 통신 집약적인 균등 분배(uniformly partitioned) MPI 프로그램들에 대한 성능 예측 알고리즘과 자원 선택 알고리즘을 제안한다. 성능 예측 알고리즘은 자원의 이질성, 네트워크 성능, 복수 노드에 있는 CPU 부하, 응용 프로그램의 특성을 고려해 성능을 예측 한다. 자원 선택 알고리즘은 k개의 후보 자원 집합들을 생성한 후, 이들 중 최적의 집합을 선택한다. 이를 통해 기존 그리디 알고리즘(greedy algorithm)의 약점이던 지역성을 극복했다. 그리고 기존의 연구들과 다르게 소스 코드 수정이나 소스 코드 분석이 필요 없기 때문에 사용하기가 용이하다.

본 논문의 구성은 다음과 같다. 2장에서는 CPU 부하와 네트워크 대역폭이 병렬처리 시의 성능에 끼치는 영향들에 대해 기술한다. 3, 4장에서는 2장의 결과에 바탕을 둔 성능 예측 알고리즘과 자원 선택 알고리즘에 대해 기술한다. 마지막 5장은 본 논문의 결론과 향후 연구 과제에 대해 기술한다.

## 2. CPU 부하와 네트워크 대역폭이 성능에 끼치는 영향

CPU 부하와 네트워크 대역폭이 균등 분배 MPI 프로그램에 끼치는 영향을 알아보기 위하여 여러 가지 실험을 수행하였다. 실험에 사용된 MPI 라이브러리는 MPICH-G2(MPICH 버전 1.2.4)[5]이고, Numerical Aerodynamics Simulation Parallel Benchmarks[6] 중 통신양이 많은 IS, CG에 대해 실험을 하여 다음과 같은 결과를 얻을 수 있었다.

단일 노드에서 CPU 사용가능성(전체 CPU 시간 중 해당 프로그램에 할당되는 CPU 시간의 비율)과 성능은 비례한다. *CPU_Mops*는 부하가 있을 때 노드 i의 유효 성능, *Raw_Mops*는 CPU 사용가능성이 100%일 때 노드 i의 성능, *CPU_Avail*는 노드 i의 CPU 사용가능성을 나타낼 때, 다음이 성립한다. Mops는 Million Operations Per Second의 약자로 성능의 단위이다.

$$CPU\_Mops_i \approx Raw\_Mops_i \times CPU\_Avail_i$$

그림 1은 IS를 동일한 사양의 2개 노드에서 실행시켰을 때의

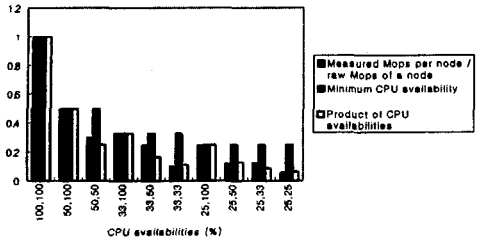


그림 1 IS를 2개 노드에서 실행한 결과

결과이다. 두 노드에 모두 CPU 부하가 있는 경우, 측정된 성능은 CPU 부하를 고려한 느린 노드의 유효 성능(Minimum CPU availability × raw Mops of a node)보다 더 낮음을 알 수 있다. 이는 2개 노드에 모두 CPU 부하가 있으면 동기가 맞지 않아서 통신 성능이 떨어지기 때문이다. 그림 1에서 보면

$$Measured\ Mops\ per\ node \approx raw\ Mops\ of\ a\ node \times Product\ of\ CPU\ availabilities$$

이 성립하고, 이로부터 다음 수식이 성립한다. 여기서 $Eff_CPU_Mops_{ij}$는 CPU부하와 그로 인한 통신 성능 저하를 고려한 각 노드 i, j의 유효 성능(두 노드는 동기화 되어 같은 속도로 동작하게 된다), $Raw_Mops_i$와 $Raw_Mops_j$는 CPU 부하가 없을 때 각 노드의 성능, $CPU_Avail_i$와 $CPU_Avail_j$는 각 노드의 CPU 사용가능성을 뜻한다.

$$Eff\_CPU\_Mops_{ij} \approx MIN(Raw\_Mops_i, Raw\_Mops_j) \times CPU\_Avail_i \times CPU\_Avail_j$$

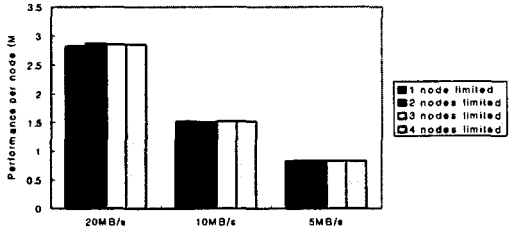


그림 2 네트워크 대역폭이 제한되었을 때 IS의 실행결과

그림 2는 4개 노드에서 노드들의 나가는 대역폭을 제한한 경우 IS의 실행결과를 보여주고 있다. x축은 Token Bucket Filter[7]를 이용하여 대역폭을 제한한 노드의 나가는 대역폭을 나타낸다. 각 경우에 대해 1, 2, 3, 4개 노드의 대역폭을 제한하여 보았다. 대역폭 제한 정도가 같은 경우, 1개 노드의 대역폭이 제한되었을 때의 성능과 2, 3, 4개 노드들의 대역폭이 제한되었을 때의 성능이 같음을 알 수 있다. 즉, 여러 개의 연결(link)이 있을 때 가장 느린 연결이 전체 성능의 병목이 된다. 따라서 노드 i와 다른 노드(노드 i와 통신하는) 사이의 대역폭 중에서 최소인 대역폭을 $Min\_Band_i$, 노드 i와 통신하는 노드들의 수를 $\#\_of\_Comm\_Nodes$, 노드 i의 네트워크 인터페이스가 제공하는 대역폭을 $Band_of_Network_Interface_i$라고 할 때, 노드 i가 실제로 사용할 수 있는 대역폭 $Avail_Band_i$는 다음과 같다.

$$Avail\_Band_i \approx MIN(Min\_Band_i \times \#\_of\_Comm\_Nodes, Band\_of\_Network\_Interface_i)$$

2개 노드에서 대역폭에 제한을 주었을 때, IS의 실행결과 측정된 성능은 그림 3에 나와 있다. x축은 대역폭을 나타낸다. 대역폭이 제한되었을 때 각 노드의 성능은 대역폭 사용량에 비례함을 알 수 있다. 즉, 대역폭이 요구량보다 작을 때 각 노드의 성능은 대역폭에 비례한다. 대역폭이 요구량 보다 클 때 노드 i의 성능을 $Raw\_Mops_i$, $App_Band$는 프로그램이 요구하는

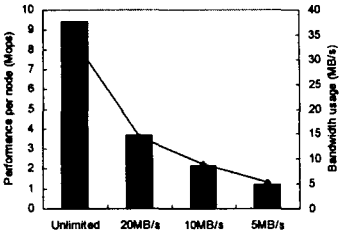


그림 3 대역폭을 제한한 2개 노드에서 IS를 실행했을 때 성능과 대역폭 사용량

대역폭이라면, 제한된 대역폭을 고려한 노드 i의 유효 성능 $Band_Mops_i$는 다음과 같다.

$$Band\_Mops_i \approx Raw\_Mops_i \times MIN(\frac{Avail\_Band_i}{App\_Band}, 1)$$

노드 2개만이 있을 때, $Eff_Comb_Mops_{ij}$를 대역폭, CPU 부하, CPU 부하로 인한 통신 성능 저하를 모두 고려한 각 노드 i, j의 유효 성능이라 하면, 다음이 성립한다.

$$Eff\_Comb\_Mops_{ij} \approx MIN(Band\_Mops_i, Band\_Mops_j) \times CPU\_Avail_i \times CPU\_Avail_j$$

2개 이상의 노드가 있을 때, 대역폭과 CPU 부하, 그리고 CPU 부하로 인한 통신 성능 저하를 고려한 노드 i의 유효 성능을 $Eff\_Comb\_Mops_i$, j는 i와 통신하는 노드들이라 하면, 다음이 성립한다. $AVG_j$는 여러 j들에 대한 값들의 평균을 구하는 함수이다. 따라서 여기서의 평균은 i가 고정되어 있을 때, 여러 j들에 대한 값들의 평균이다. i와 통신하는 노드 중 하나를 i와 짝 맺어서, 그 짝만을 생각했을 때의 유효 성능을 앞의 수식을 통해 구한다. 노드 i의 유효 성능은 각 짝들이 얻을 수 있는 유효 성능의 평균이 될 것이다.

$$\begin{aligned} Eff\_Comb\_Mops_i &\approx AVG_j(Eff\_Comb\_Mops_{ij}) \\ &\approx AVG_j(MIN(Band\_Mops_i, Band\_Mops_j) \times CPU\_Avail_i \times CPU\_Avail_j) \end{aligned}$$

모든 노드는 동기화 되어 가장 느린 노드의 속도로 동작하게 된다. 따라서 $n$개의 노드가 있고, 각 노드의 유효 성능 $Eff_Comb_Mops_i$가 주어졌을 때, 전체 실행 성능 $Total_Mops$는 다음과 같다. 여기서 $MIN_i$는 여러 i에 대한 $Eff_Comb_Mops_i$값들 중 최소 값을 구하는 함수이다.

$$Total\_Mops \approx MIN_i(Eff\_Comb\_Mops_i) \times n$$

```
1. 대역폭 제한 반영
for each node i
    min_band[i]= MIN_j( available bandwidth between i and j );
        /* for all j that communicates with i */
    Avail_Band[i]= MIN( min_band[i] * #_of_communicating_nodes,
                    bandwidth of node i's network interface );
    max_band_mops[i]= Mops_per_MBps * Avail_Band[i];
    Band_Mops[i]= MIN(max_band_mops[i], Raw_Mops[i]);
2. CPU 부하 반영
for each node i
    Eff_Comb_Mops[i] = AVG_j( MIN( Band_Mops[i], Band_Mops[j] )
                    * CPU_Avail[i] * CPU_Avail[j] );
        /* for all j that communicates with i */
3. 전체 성능 계산
Total_Mops = MIN_i(Eff_Comb_Mops) * n;
```

그림 4 성능 예측 알고리즘

### 3. 성능 예측 알고리즘

2장의 결과들에 기초한 성능 예측 알고리즘은 그림 4와 같다. 1단계에서는 대역폭이 제한되었을 경우, 그 영향을 반영한다. 우선 하나의 노드 i가 있을 때, i와 다른 노드 사이의 대역폭들 중에서 최소 값을 찾는다. 대역폭 최소 값을 통신하는 노드들의 수와 곱한 결과와, 노드 i 네트워크 인터페이스의 대역폭 중 작은 값이 노드 i가 실제로 사용할 수 있는 대역폭이다. 실제로 사용할 수 있는 대역폭이 프로그램이 원하는 값보다 작으면, 주어진 대역폭에서 나올 수 있는 최대 성능을 Band_Mops[i]에 저장한다. 여기서 Mops_per_MBps는 병렬처리 시 노드의 성능을 대역폭 사용량으로 나눈 값이다. 실제로 사용할 수 있는 대역폭이 프로그램에서 요구하는 값보다 크면, Raw_Mops[i]를 Band_Mops[i]에 저장한다. 2단계에서는 각 노드에 CPU 부하가 있을 경우 앞의 수식에 따라 그 영향을 반영한다. 3단계에서는 전체 실행 성능을 계산한다. $n$개의 노드들이 있을 때, 이 알고리즘의 복잡도는 $O(n^2)$이 된다.

### 4. 자원 선택 알고리즘

본 논문에서 제안하는 자원 선택 알고리즘은 그림 5와 같다. 우선 start_nodes 배열에 사용가능한 노드들 중에서 가장 빠른 k개의 노드들을 저장한다. 그 후 start_nodes의 각 원소들을 시작점으로 후보 집합들을 생성해서 candidate_sets에 저장한다. 후보 집합 생성은 뒤에서 설명할 후보 집합 생성 알고리즘에 의해 이루어진다. 마지막으로 candidate_sets의 각 후보 집합들을 성능 예측기로 평가하여 가장 좋은 집합을 선택한다.

```
Input  : available nodes, required number of nodes
Output : a node set
    start_nodes= { fastest k nodes of available nodes };
    for(i=0;i<k;i++)
        candidate_sets[i]=
            { generate a set from start_nodes[i] };
    select the best set from candidate_sets;
```

그림 5 자원 선택 알고리즘

이 알고리즘에서는 그리디 알고리즘의 단점을 극복하고자 k개의 노드들에 대해서 후보 집합을 생성한다. 가장 빠른 노드를 시작점으로 해서 하나의 후보 집합만을 생성하는 경우, 최적의 집합(global optimum)을 선택하지 못하고 가장 빠른 노드가 속한 지역의 최적 집합(local optimum)을 선택할 수 있다. 이러한 것을 방지하기 위해서 사용자가 적당한 k값을 제시하면 $k$개의 빠른 노드들에 대해서 후보 집합을 생성하고, 이런 경우 각 지역 최적의 후보들이 생성되므로 최적의 집합을 선택할 가능성이 높아진다.

후보 집합 생성 알고리즘은 시작점이 주어졌을 때, 그리디 방식으로 가장 좋은 노드를 하나씩 선택해 집합에 추가한다. 제안하는 후보 집합 생성 알고리즘은 그림 6과 같다. result_set의 크기가 사용자가 원하는 노드수와 같아질 때까지, result_set에 포함되었을 경우 전체 집합의 예상 성능이 가장 좋은 노드를 result_set에 계속적으로 추가한다. 예상 성능을 구하는 데는 그림 4의 성능 예측 알고리즘이 쓰일 수 있다.

예상 성능을 구하는 데 그림 4의 성능 예측 알고리즘이 쓰였고, 사용자가 원하는 노드 수가 $n$, 그리드에서 사용가능한 노드 수가 $N$이라고 할 때, 후보 집합 생성 알고리즘의 복잡도는 $O(n^2N^2)$이 된다.

*이질적인 노드들과 네트워크를 가진 환경에서의 2, 4, 8 노드 선택 실험 결과, 제안한 자원 선택 알고리즘은 항상 최선의 선택을 하였다.*

```
Input  : start_point, available nodes, required_#_of_nodes
Output : a node set that includes start_point
    result_set={ start_point };
    node_pool={ available nodes except start_point };
    while( size_of(result_set) < required_#_of_nodes ) {
        next_node= { next_node is in node_pool &&
            ( For all other_node in node_pool,
              performance_of( result_set + next_node ) >=
              performance_of( result_set + other_node ) ) };

        Add next_node to result_set;
        Remove next_node from node_pool;
    }
    return result_set;
```

그림 6 후보 집합 생성 알고리즘

### 5. 결론

본 논문에서는 균등 분배 MPI 병렬 프로그램에 대한 성능 모델을 세웠고, 이를 바탕으로 성능 예측 알고리즘과 자원 선택 알고리즘을 제안하였다. 자원 선택 알고리즘은 성능 예측 알고리즘을 이용하여 자원의 이질성, 대역폭 제한, 복수 노드에 있는 CPU 부하, 응용 프로그램의 특성을 고려하는 효과적인 자원 선택을 한다. 그리고 그리디 알고리즘의 단점인 지역성(local optimum)을 극복하기 위해 k개의 후보 자원 집합들을 생성한 후, 이들 중 최적의 집합을 선택한다. 그리고 소스 코드 수정이나 소스 코드 분석 없이 시스템 수준에서의 분석만이 필요하기 때문에 사용하기가 용이하다.

### 참고문헌


[1] I. Foster, and C. Kesselman, ed., The Grid: Blueprint for a New Computing Infrastructure, pp. 15-51, Morgan Kaufmann Publishers, 1999

[2] I. Foster, and C. Kesselman, "Globus: A metacomputing infrastructure toolkit", International Journal of Supercomputer Applications, Vol. 11, No. 2, pp. 115-128, 1997.Summer

[3] C. Liu, L. Yang, I. Foster, AND Angulo, "Design and evaluation of a resource selection framework for Grid applications", *Proceedings of the 11th IEEE Symposium* on High-Performance Distributed Computing, pp. 63-72, *2002.7*

[4] J. Subhlok, P. Lieu, and B. Lowekamp, "Automatic Node Selection for High Performance Applications on Networks", Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel *Programming*, pp 163-172, 1999.5

[5] I. Foster, and N. Karonis, "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems", Proceedings of Supercomputing 98, 1998.11

[6] D. Bailey, T. Harris, W. Saphir, R. Wijngaart, A. Woo, and M. Yarrow, "The NAS parallel benchmarks 2.0", Technical Report NAS-95020, 1995.12

*[7] B. Hubert, Linux Advanced Routing & Traffic Control* HOWTO, 2002