다중 클러스터 시스템을 위한 Co-Allcation 알고리즘

권석면[O] 김진석
서울시립대학교 컴퓨터과학부
{leonids1[O], jskim}@venus.uos.ac.kr

A Co-Allocation Algorithm for a Multi-Cluster System

Seok Myun Kwon[0] Jin Suk Kim
School of Computer Science, University of Seoul

요 약

본 논문에서는 Grid 시스템에서 작업을 할당하기 위해서, 가능한 모든 작업 분할 방법을 찾는 기법에 대해 연구 하고자 한다. 즉 다수의 노드를 가진 여러 클러스터 사이에 많은 노드를 필요로 하는 작업을 할당하기 위해 변형된 깊이 우선 탐색 트리를 사용하여 작업 분할 리스트를 찾는 방법을 만들고, 그 알고리즘의 속도 및 효율을 평가하였다. 제안된 알고리즘은 전체 리스트를 검색하는 방법에 비해 속도 측면에서 빠른 효율을 보이는 것을 알 수 있다. 따라서 제안된 알고리즘은 동기화된 작업의 스케줄링에 사용할 경우 유용할 것으로 생각된다.

1. 서 론

최근 과학 기술이 발전함에 따라 여러 복잡한 문제를 해결하기 위해서 고성능의 연산 처리를 요구하고 있다. 이러한 요구를 충족시키기 위해 같은 공간에 동질의 자원을 묶어 사용하는 방법을 사용하였으나, 이 방법은 많은 비용과 물리적인 공간을 요구하게 된다. 이 문제를 해결하기 위해 연구되고 있는 Grid 시스템에서는 존재하는 여러 개의 클러스터가 가진 노드에 작업을 할당하여 이 문제를 해결하고자 하고 있다[1]. 그러나 Grid 시스템에서 작업을 할당하기 위해 사용하는 알고리즘들은 여러 개의 클러스터에 작업을 할당하기 위해 필요한 클러스터의 목록을 모두 만들거나, 할당할 수 있는 클러스터의 개수를 제한하고 있다[2,3,5].

여러 개의 클러스터에 작업을 할당하는 경우에 주어진 클러스터의 목록에서 작업이 요구하는 노드의 개수를 만족하는 조합을 찾아내는 과정이 필요하다. 모든 조합을 다 고려한다고 하면 클러스터의 개수를 n이라고 할 때 2^n-1 만큼의 조합을 고려해야 한다. 이 경우 n이 커지면 커질수록 처리해야할 조합의 수가 많아질 뿐만 아니라 불필요한 연산을 수행하는 경우가 증가하게 된다.

반대로 클러스터를 최대 m개에만 할당한다고 제한을 하게되는 경우에는 더 좋은 성능을 낼 수 있는데도 불구하고 나쁜 성능을 내는 클러스터에 할당하는 경우가 생기게 된다. 예를 들어, 4개의 클러스터가 각각 다음과 같이 노드를 가지고 있고, 스케줄러가 클러스터 조합의 수를 줄이기 위해, 최대 2개의 클러스터에만 할당 가능하도록 제한한 경우를 생각해 보자.

$$C_1(4), C_2(5), C_3(3), C_4(6)$$

이때 사용자가 16개의 노드를 요구하면, 작업을 4개의 클러스터 모두에 분할 할당하면 충분히 가능하지만, 스케줄러가 2개의 자원에 할당하는 것으로 제한되어 있기 때문에 할당을 하지 못하게 된다.

따라서 클러스터에 작업을 할당할 때는 스케줄러의 알고리즘도 중요하지만, 유용한 클러스터의 조합을 얼마나 효율성 있게 만들어 내는가 하는 문제도 중요하게 된다. 2절에서는 변형된 깊이 탐색 트리를 이용하여 클러스터의 분할 조합을 만들어 내는 방법을 설명하고, 3절에서는 노드 분할 알고리즘의 효율성에 대해 살펴본다.

2. 노드 분할 알고리즘

2.1 알고리즘의 기본 가정

알고리즘을 실행하기 위해서 다음의 3가지 기본 가정을 만족해야 한다.

가정 1. 각각의 클러스터가 가진 노드의 성능은 모두 동일하다.

가정 2. 실행할 작업을 J, 작업 J가 분할 할당된 클러스터의 집합을 C, C에 할당된 작업 J가 실행되는데 걸리는 시간을 E(J, C)라고 하면 다음의 조건을 만족한다.

$$C_1 \subset C_2 \text{ 이면 } \quad E(J,C_1) \le E(J,C_2)$$

가정 3. 클러스터들은 각각 작업을 할당할 수 있는 노드의 개수가 많은 것부터 적은 순으로 정렬되어 있다.

가정 1은 각각의 클러스터들은 모두 다른 성능을 가지고 있지만, 클러스터 내부의 노드는 동일한 성능을 가지고 있다는 것을 의미한다.

가정 2에서 동일한 작업을 클러스터의 집합 C에 작업을 할당하는 경우, C의 서브 클러스터의 집합에 할당이 가능하다면 서브 클러스터의 집합에 할당할 경우 더 좋은 성능을 낼 수 있다는 것을 의미한다. 예를 들어 $C=\{c_1,c_2,c_3\}$라고 하면, 실행하려는 작업이 $C_1=\{c_1,c_2\}$에서 실행이 가능한 경우, C_1에 실행시키는 것이 좀 더 나은 성능을 기대할 수 있다는 것을 의미한다. 따라서 만일 더 좋은 성능을 내는 서브 클러스터의 집합이 있다면, C는 고려하지 않고, C_1만을 고려하면 된다.

2.2 알고리즘

기본적인 알고리즘은 변형된 깊이 탐색 트리를 이용하여 구현된다[4]. 작업을 실행시킬 수 있는 모든 클러스터의 조합을 만

들기 위해서는 우선 노드의 개수로 정렬을 한 다음, 아래의 NodeFind 함수를 실행시킨다.

```
NodeFind(selectNum, selectData, nodeNum, maxNode)
Input selectNum : 선택된 Cluster의 개수
      selectData: 선택된 Cluster의 목록
      nodeNum     : 선택된 Node의 개수
      maxNode     : 최대로 포함될 수 있는 Node의 개수

  if (needNode ≤ nodeNum)  AddList(selectData);
    else if (needNode=maxNode)
        AddList(selectData+remainNode);
    else if (needNode>maxNode)      return;
    else
    {
        if (selectNum=0)  sp=0;
        else sp = selectData[selectNum-1] + 1;
            for (i = sp to clusterNum)
            {
              selectData[selectNum] = i;
              NodeFind(selectNum+1, selectData,
                 needNode+clusterNodeNum[i], maxNode);
              maxNode=maxNode - clusterNodeNum[i];
            }              //end_if
    }    // end_else
```

그림 1. 알고리즘

NodeFind 함수에서, remainNode는 추가로 할당할 수 있는 클러스터의 목록, needNode는 작업을 할당하기 위해 필요한 노드의 개수, clusterNum은 클러스터의 총 숫자, clusterNodeNum은 각각의 클러스터가 가지고 있는 노드의 개수를 의미한다. 또한, AddList 함수는 선택된 클러스터의 정보를 추가하는 함수를 의미한다. 최종적인 결과는 AddList 함수를 통해 추가된 클러스터의 목록이 된다.

Example 1:

5개의 클러스터가 각각 다음과 같이 구성되어 있다고 할 때,

$C_1(10)$, $C_2(7)$, $C_3(6)$, $C_4(4)$, $C_5(3)$

10개의 노드가 필요한 작업을 할당하는 경우를 생각해 보자. $C_1(10)$은 첫 번째 클러스터가 10개의 노드를 가지고 있다는 것을 의미한다. 이 시스템의 스케줄링 알고리즘이 다음과 같은 공식에 의해 실행 시간이 가장 적게 걸리는 클러스터에 작업을 할당한다고 하자.

$$E(J,C) = \max(E(J,C_i)) * \frac{1}{\min(N_{ij})}, \quad C \supset C_i$$

여기서 $\max(E(J,C_i))$는 작업이 할당된 클러스터 중에서 가장 오랜 실행 시간을 가지는 클러스터의 실행 시간을 의미하고, N_{ij}는 작업이 할당된 클러스터 사이의 네트워크 가중치로 0에서 1사이의 값을 가진다. 이 예제에서는 <표 1>과 같은 실행 시간과 <표 2>와 같은 네트워크 가중치를 가진다고 가정한다.

표 1. 클러스터 실행 시간

i	1	2	3	4	5
$E(J,C_i)$	50	30	45	40	35

표 2. 네트워크 가중치

i \ j	1	2	3	4	5
1	1	0.8	0.9	0.75	0.8
2	0.8	1	0.7	0.9	0.9
3	0.9	0.7	1	0.85	0.8
4	0.75	0.9	0.85	1	0.7
5	0.8	0.9	0.8	0.7	1

<그림 2>에서 트리를 구성하는 노드의 왼쪽 숫자는 현재까지 할당된 클러스터가 가진 노드의 총합을 의미하며, 오른쪽의 숫자는 현재 상태에서 추가로 검색할 수 있는 모든 클러스터가 노드에 작업을 할당할 경우 작업이 할당될 수 있는 노드의 최대 개수를 의미한다. 이 경우 <그림 2>의 (a)와 같이 C_1에 할당하는 경우 필요한 노드의 개수를 만족하기 때문에 C_1을 결과 목록에 포함시킨다. 일단 만족하는 C_1을 목록에 포함시키면, 가정 2에 의해서 C_1을 포함하는 어떤 클러스터의 집합도 $\{C_1\}$보다 빠른 속도를 가지지 못하므로 더 이상 검색을 하지 않는다. 다음으로 C_2에 할당하는 경우, C_2의 노드의 개수만을 가지고 필요한 노드의 숫자를 만족시키지 못하기 때문에, <그림 2>의 (b)와 같이 다음 클러스터인 C_3, C_4, C_5를 하나씩 추가해 가면서 클러스터의 집합을 만들어 나간다. 마지막으로 <그림 2>의 (c)와 같이 C_4의 경우 최대 기대 노드의 숫자가 필요한 노드의 숫자보다 적으므로 더 이상 검색을 하지 않게 된다.

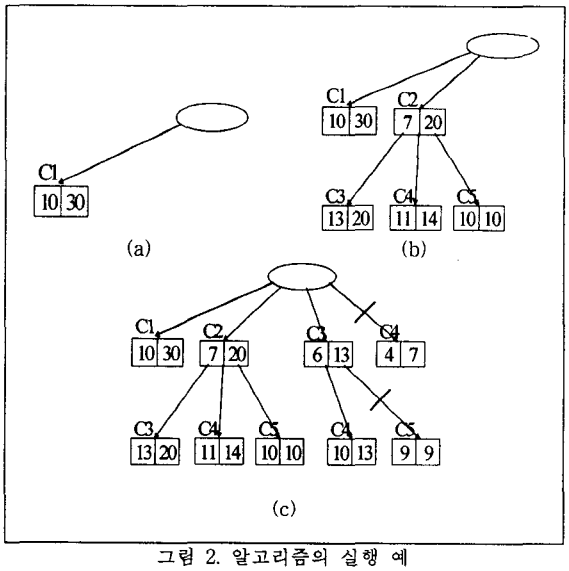

그림 2. 알고리즘의 실행 예

<표 3>은 알고리즘을 이용하여 최종적으로 만들어지는 리스트의 목록과 그 목록에 작업이 할당된다고 가정했을 경우, 스케줄링 알고리즘의 수식에 대입하여 계산한 결과이다. 계산 결과 스케줄러는 최소 실행 시간을 가지는 $C_2(7)$, $C_5(3)$에 작업을 할당하게 된다.

표 3. 최종 결과 리스트

결과 리스트	실행 시간
$C_1(10)$	50
$C_2(7)$, $C_3(6)$	64.28
$C_2(7)$, $C_4(4)$	44.44
$C_2(7)$, $C_5(3)$	38.89
$C_3(6)$, $C_4(4)$	52.94

3. 실험 결과

우선 알고리즘의 성능을 확인하기 위해서 다음과 같은 실험을 하였다. 임의의 노드를 가진 15개의 클러스터에 작업의 필요 노드 수를 1개에서 노드의 총 합까지 변경하면서 노드의 숫자에 따라 총 20개의 구간으로 나누어 그 함수 호출 회수와 생성되는 리스트의 개수의 평균값을 구하였다. 함수 NodeFind는 재귀호출을 사용하기 때문에 함수 호출 회수는 각각 작업의 크기에 따른 프로그램 수행시간에 비례하게 된다. <그림 3>은 실제로 알고리즘을 구현한 프로그램을 사용해서 리스트를 만들어 본 결과이다. <그림 3>의 그래프에서 위쪽 점선 그래프는 함수를 호출하는 회수를 나타낸 그래프이며, 아래 실선 그래프는 생성된 리스트의 개수를 의미한다. x축은 시스템의 총 노드 수를 20개로 나눈 구간을 의미한다. <그림 3>에서 알 수 있듯이 생성된 리스트의 크기에 2배정도 함수의 호출이 발생하는 것을 알 수 있다. 또한 모든 경우를 다 검색하는 경우 최대 2^n-1만큼 케이스를 검색해야 하지만, 알고리즘을 사용하는 경우, 최대값을 가지는 경우의 12%정도만을 검색하면 원하는 리스트를 모두 찾아낼 수 있음을 알 수 있다. 또한 리스트가 많이 발생하는 35%~60%구간을 제외하면, 평균 1.2%정도 검색을 하면 된다는 것을 알 수 있다.

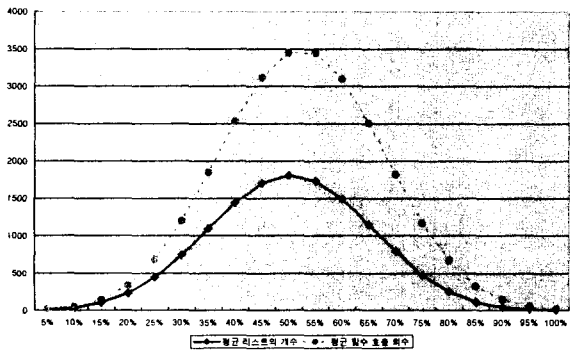

그림 3. 구간별 평균 실행 시간

또한 클러스터의 개수를 변화 시켜가면서 함수가 최대 호출 회수를 가지는 경우를 찾았다. 이때 최대 호출 회수가 의미하는 것은 알고리즘을 실행시켰을 때 발생하는 최악의 경우를 의미한다. 만약 2개의 시스템이 동일한 숫자의 클러스터를 가지고 있고, 또한 클러스터들의 노드의 합이 같은 경우가 있다고 하면, 클러스터가 균일한 숫자의 노드를 가지는 시스템이 클러스터의 노드 숫자가 다양하게 존재하는 시스템에 비해 더 많은 호출 회수를 가지는 것을 알게 되었다.

예를 들어, 클러스터의 개수가 5개이며, 총 노드의 개수가 15개인 경우를 살펴보자. 노드의 숫자가 일정한 크기로 증가하는 $\{C_1(1),C_2(2),C_3(3),C_4(4),C_5(5)\}$의 경우는 최대 호출 회수가 26회가 되는데, 이 값은 각각의 노드가 균일한 숫자의 노드를 가지는 $\{C_1(3),C_2(3),C_3(3),C_4(3),C_5(3)\}$일 때 최대 호출 회수인 20회보다 큰 값을 가진다. 따라서 시스템이 동일한 노드를 가진 클러스터의 경우보다 노드의 숫자가 다양하게 분포한 경우가 더 나쁜 결과를 보였다. 여기서는 알고리즘이 최악의 실행결과를 가지는 경우를 알기 위해서, n개의 클러스터가 각각 1~n까지의 노드의 숫자를 가지는 경우를 가지고 최대 호출 회수를 가지는 경우를 찾았다.

표 4. 최대 호출 회수

클러스터의 수	5	7	9	11	13	15
호출 회수	10	28	76	234	750	2467
근사값	7.8	24.5	76.7	240	753.1	2359.1

<표 4>에서 근사값은 $0.45*1.77^n$을 사용해서 계산한 것으로 호출 회수와 유사한 값을 가진다.

4. 결 론

Grid에서 스케줄링 알고리즘은 작업을 할당하기 위해서 클러스터가 가진 모든 노드를 사용하고 있다. 스케줄링 알고리즘의 경우 작업을 할당할 수 있는 모든 경우를 고려하여 작업을 할당하는 것이 유리하지만, 연결된 클러스터의 숫자가 많은 경우 계산 량이 많아 전체적으로 모두 검색하기 힘든 것이 사실이다. 따라서 일부의 알고리즘에서는 연산량을 줄이기 위해서 최대로 할당할 수 있는 클러스터의 숫자를 제한하고 있다. 그러나 위에 제안된 알고리즘을 사용하면 모든 경우를 검색하지 않고 검색 트리의 일부분만 검색하여 원하는 리스트들을 빠르게 찾을 수 있었다. 따라서 검색 알고리즘은 Grid 시스템과 같은 클러스터간의 스케줄링에서 유용하게 사용할 수 있다.

참고문헌

[1] I. Forster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Journal of High-Performance Computing Applications,* Vol. 15, no. 3, pp. 200-222, 2001.

[2] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Indenpendent Tasks onto Heterogeneous Computing System," *Proc. of the 8th Heterogeneous Computing Workshop,* pp. 30-44, April, 1999.

[3] O. H. Ibarra and C. E. Kim, "Heuristic Algorithm for Scheduling Independent Tasks on Nonidentical Processors," Journal of the ACM, vol. 24, no.2, pp.280-289, April, 1977.

[4] Sara Basse, Allen Van Gelder, *Computer Algorithms Introduction to Design & Analysis Third Edition,* Addison wesley, 2000.

[5]Gregory F. Pfister, *In Search of Clusters Second Edition,* Prentice-Hall PTR, New Jersey, 1998.