이기종 클러스터 환경에서 부하공유를 위한 Enhanced Weighted Factoring 알고리즘

최인복[0] 이재동
단국대학교 전자계산학과
(pluto612[0], jdlee)@cs.dankook.ac.kr

Enhanced Weighted Factoring Algorithm For Load-Sharing In Heterogeneous Clustering Systems

In-Bok Choi[0] Jae-Dong Lee
Division of Information and Computer Science, Dankook University

요 약

최근 인터넷이 발달하면서 인터넷 상의 다양한 컴퓨터들을 연결함으로써 이기종 클러스터 환경 구축이 용이해졌다. 이러한 이기종 클러스터 환경에서 알고리즘의 이식성을 높이기 위해서는 네트워크의 특성 및 노드의 이질성에 따른 부하 불균형에 효과적으로 적응할 수 있어야 한다. 본 논문에서는 이기종 클러스터 환경에서 Message Passing 방식을 이용한 고성능 클러스터 컴퓨팅 작업 시 최적의 효율을 얻을 수 있는 Enhanced-WF 알고리즘을 제시한다. Enhanced-WF 알고리즘은 부하공유를 위하여 Weighted Factoring 알고리즘을 기반으로 적응할당정책을 적용하는 동시에 네트워크 통신시간과 계산시간을 겹치게 한다. Enhanced-WF 알고리즘의 성능을 측정하기 위해 이기종 PC클러스터 환경에서 PVM을 이용한 행렬곱셈 프로그램을 이용하였다. 그 결과, Enhanced-WF 알고리즘이 이기종 클러스터 환경에서 Send, GSS, Weighted Factoring 알고리즘과 같은 기존의 부하공유 알고리즘보다 효과적임을 보였다.

1. 서 론

최근 마이크로 프로세서의 뛰어난 성능향상과 고속 네트워크의 보급으로 단일 컴퓨터들을 네트워크로 연결함으로써 하나의 병렬 컴퓨터처럼 작동하는 클러스터 컴퓨팅이 가능해졌고, 나아가 인터넷이 발달하면서 인터넷 상의 다양한 성능의 컴퓨터들을 TCP/IP 프로토콜에 의해 연결함으로써 이기종의 클러스터를 구성할 수 있게 되었다.

지금까지의 고성능 클러스터에서 부하공유 알고리즘은 주로 공유 메모리 구조의 시스템에서 노드들의 계산 능력이 서로 비슷하다는 가정에서 개발되었으며 네트워크의 특성을 고려하지 않았다. 그러나, 인터넷 기반의 이기종 클러스터 환경에서 알고리즘의 이식성을 높이기 위해서는 노드의 이질성에 따른 부하 불균형 그리고 네트워크의 특성과 같은 다양한 수행 환경의 변화에도 효과적으로 적응할 수 있도록 해야 한다.

따라서, 본 논문에서는 인터넷 기반의 이기종 클러스터 환경에서 Message Passing 방식을 이용한 고성능 클러스터 컴퓨팅 작업 시 최적의 효율을 얻을 수 있도록 Weighted Factoring 알고리즘을 확장한다. 알고리즘의 확장 방법은 효과적인 부하공유를 위하여 느린 종노드의 작업을 빠른 종노드가 대신 수행하는 적응할당정책을 적용하는 동시에 다음에 처리해야 할 작업을 미리 보냄으로써 네트워크 통신시간과 계산시간을 겹치도록 한다.

이기종의 PC를 연결한 클러스터 환경에서 행렬의 곱셈을 계산하는 프로그램을 알고리즘별로 구현하여 기존의 부하공유 알고리즘들과 비교함으로써 본 논문에서 제안한 Enhanced-WF 알고리즘이 효과적임을 보인다.

본 논문은 다음과 같이 구성된다. 2장에서는 클러스터 환경에서의 부하공유를 위한 대표적인 연구들을 소개하고 3장에서는 본 논문에서 제안하는 Enhanced-WF 알고리즘의 부하공유 방안을 살펴본다. 4장에서는 이기종

클러스터 환경에서의 부하공유 알고리즘들의 성능을 실험을 통해 비교해 보고, 5장에서는 결론 및 향후 발전 과제를 언급함으로써 논문을 마친다.

2. 관련 연구

부하공유란 공간분할 스케줄링 방법에서 빠른 실행결과를 얻기 위한 방법을 말한다[2]. Piotrowski와 Dandamudi의 연구결과에 의하면 NOW 환경에서 대체적으로 Send 알고리즘과 GSS 알고리즘이 좋은 성능을 보인 것으로 나타났다[3][1].

이기종 클러스터 환경에서의 대표적인 부하공유 알고리즘에는 Flynn Hummel의 Weighted Factoring 알고리즘이 있다. *Weighted Factoring* 알고리즘은 종노드의 계산 성능의 비율에 따라 가중치(weight)을 부여하고 그 가중치에 따라 반복적으로 데이터의 크기를 동적으로 줄여나간다. N개의 데이터와 P개의 종노드에 대하여 i번째 묶음에 있는 j번째 데이터 덩어리의 크기를 F_{ij}라고 하면, F_{ij}는 아래의 [식1.1]과 같이 결정된다.

$$F_{ij} = \left\lceil \frac{(1-\frac{1}{x})^{i}\frac{N}{x}W_j}{\sum_{k=1}^{P}W_k} \right\rceil = \left\lceil \frac{(\frac{1}{2})^{i+1}N\frac{W_j}{\sum_{k=1}^{P}W_k}}{} \right\rceil \quad [식1.1]$$

일반적으로 x의 값은 2정도가 다양한 응용에 대하여 좋은 성능을 보이는 것으로 알려져 있다[4][5].

3. Enhanced-WF 알고리즘

Enhanced-WF 알고리즘은 이기종 클러스터 환경에서의 효과적인 부하공유의 목적을 위하여 설계되었다. 효과적인 부하공유를 위해서 Weighted Factoring 알고리즘에 적응할당정책을 적용함과 동시에 네트워크 통신시간과 계산시간을 겹치게 하였다. 이를 위해, Enhanced-WF 알고리즘에서는 전역적인 스케줄러를 운영한다. N개의 데이터와 P개의 종노드에 대한 스케줄러의 데이터구조는 다음과 같다.

```
struct slave_node {
  int job[⌊log2N⌋];
  int status[⌊log2N⌋]; //job상태 (0:미수행, 1:전달/수행중, 2:완료)
  float weight; // 가중치
  int remain;   // 미수행 상태인 job의 크기
} schedule[P];
```

주 노드에서 임의의 i번째 종노드로부터 부분적인 결과 데이터 schedule[j].job[k]를 전송 받았을 때, slave_node 구조체 내의 변수들은 메시지 전달방식 고성능 클러스터에서의 기본적인 명령인 send/receive 명령 수행 시 아래의 서브루틴과 같이 변화하게 된다.

```
Procedure SEND (schedule[j].job[k], i)
1 send (data of schedule[j].job[k] to ith node);
2 schedule[j].status[k] = 1;
3 schedule[j].remain -= size of
    schedule[j].job[k];
```

```
Procedure RECEIVE (schedule[j].job[k], i)
1 wait (arbitrary partial result data
        schedule[j].job[k] from ith node);
2 schedule[j].status[k] = 2;
```

효과적인 부하공유를 위한 적응할당정책의 방법은 자신의 job을 먼저 마친 종노드가 성능에 비해 가장 느리게 작업을 수행하는 종노드의 작업을 대신 처리하도록 한다. 임의의 종노드 i로부터 schedule[j].job[k]의 부분적인 결과 데이터를 전송 받았을 경우, 주노드는 적응할당정책을 위하여 해당 종노드의 다음의 작업선정을 아래의 서브함수와 같이 수행한다.

```
Function SELECT_JOB (schedule[j].job[k], i)
Input : schedule[j].job[k], received partial result
        data; i, index of a arbitrary slave node
Output : schedule[m].job[n], next partial job for
        i th slave node

1  if (schedule[i].remain != 0) {
2    m = i;
3    n = k+1;
4  }
5  else if (Exist schedule[0...(P-1)].remain != 0) {
6    m = 0;
7    max_remain = schedule[0].remain *
         schedule[0].weight;
8    for (index=1; index<P; index++) {
9      temp_remain = schedule[index].remain *
         schedule[index].weight;
10     if (max_remain < temp_remain) {
11       max_remain = temp_remain;
12       m = index;
13     }
14   }
15   for (index=-1; index>0; index--) {
16     if (schedule[m].status[index] == 0) {
17       n = index;
18       break;
19     }
20   }
21 }
22 return schedule[m].job[n];
```

마지막으로 Enhanced-WF 알고리즘은 위의 서브함수를 적용함과 동시에 네트워크시간과 계산시간을 겹치게 하기 위하여 다음의 작업을 미리 전송(아래 3~4행)하도록 함으로써 효율적인 부하공유를 수행한다.

```
Algorithm Enhanced-WF
Input : N, size of job; P, number of slave nodes
Output : result, merged partial data are received from
         slave nodes (e.g. array, a variable)

1 EVALUATE (performance of slave nodes);
2 CREATE (scheduler by Weighted Factoring Alg.);
3 SEND (schedule[0...(P-1)].job[0], 0...(P-1));
4 SEND (schedule[0...(P-1)].job[1], 0...(P-1));
5 while (all partial results are not gathered by master
  node) {
6   RECEIVE (schedule[j].job[k], i);
7   if (schedule[i].remain != 0 or
    Exist schedule[0...(P-1)].remain != 0) {
8     schedule[m].job[n] =
        SELECT_JOB (schedule[j].job[k], i);
9     SEND (schedule[m].job[n], i);
10  }
11  MERGE (pratial result data of schedule[j].job[k]
    to the result);
12 }
```

4. 실험 및 결과

본 논문에서 제시한 Enhanced-WF 알고리즘이 이기종 환경에서 다른 알고리즘보다 효과적임을 보이기 위해 동일한 환경 하에서 Send 알고리즘, GSS 알고리즘, 그리고 Weighted Factoring 알고리즘을 각각 구현한 경우의 고성능 클러스터 프로그램과의 수행시간을 비교하였다.

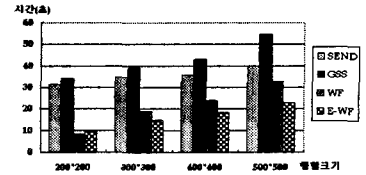
수행시간의 비교를 위해 많은 응용 분야에서 이용되는 연산인 행렬 곱셈을 PVM3.4.4 라이브러리를 이용하여 작성하였다.

실험환경은 이기종의 환경을 구성하기 위하여 [표 4.1]과 같이 각 노드의 성능과 운영체제를 다르게 설정하여 구성하였다.

노드명	CPU	메모리	운영체제
Master	P3 450	320 M	Linux(kernel 2.4)
Node1-3	P3 450	128 M	Linux(kernel 2.4)
Node4-5	P3 MMX 300	128 M	Solaris 8.0
Node6	P3 450	128M	Linux(kernel 2.4)
Node7	P-pro 133	64 M	Linux(kernel 2.0)
Node8-9	P-pro 133	32 M	Linux(kernel 2.0)
Node10	P-pro 133	16 M	Linux(kernel 2.0)

[표 4.1] 시스템 구성표

각 알고리즘별 200*200, 300*300, 400*400, 500*500 크기의 행렬의 곱을 계산하는 프로그램을 수행하였다. 각 행렬의 크기별로 20회의 반복적인 프로그램 실행을 통하여 얻은 평균실행시간(초)을 기록한 결과, 아래의 [그림 4.1]과 같이 200*200의 크기를 제외한 모든 실험에서 Enhanced-WF 알고리즘이 높은 성능과 함께 수행시간의 예측이 가능할 정도의 안정성을 보였다.



[그림 4.1] 데이터크기별 알고리즘 수행시간

5. 결론

본 논문에서는 이기종의 클러스터 환경에서 효과적인 부하공유 위하여 Enhanced-WF 알고리즘을 제안하였다.

실험의 결과를 행렬의 크기별 알고리즘별 성능향상에 대한 평균값으로 분석해 본 결과, 본 논문에서 제안한 Enhanced-WF 알고리즘이 NOW 환경에서 강건하다고 입증된 Send 알고리즘보다 73%, GSS 알고리즘보다 77%의 성능향상을 보였으며, Weighted Factoring 알고리즘보다도 15%의 성능향상을 보였다. 또한, 다른 알고리즘에서 제공할 수 없었던 안정적인 수행시간을 제공할 수 있음을 보였다.

인터넷기반의 분산된 이기종 클러스터 환경은 NOW 환경보다 더욱 동적이다. 이러한 환경에서는 효과적인 부하공유뿐 아니라, 네트워크의 고장 및 노드의 고장에 대하여 대처할 수 있어야 한다. 따라서, 향후에는 네트워크 및 노드의 고장에 대처할 수 있는 결함허용에 대한 알고리즘의 연구가 필요할 것이다.

참고문헌

[1] 구본근, "NOW 환경에서 개선된 고정 분할 단위 알고리즘", 정보처리학회논문지, Vol.8 No.2, 2001.

[2] 김진성, 심영철, "이질적 계산 능력을 가진 NOW를 위한 공간 공유 스케쥴링 기법", 정보과학회논문지, Vol.27 No.7, 2000.

[3] A. Piotrowski and S. Dandamudi, "A Comparative Study of Load Sharing on Networks of Workstations", Proc. Int. Conf. Parallel and Distributed computing system, New Orleans, Oct. 1997.

[4] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, "Load-Sharing in Heterogeneous Systems via Weighted Factoring", SPAA, 1997.

[5] Yangsuk Kee and Soonhoi Ha, "A Robust Dynamic Load-Balancing Scheme for Data Parallel Application on Message Passing Architecture," PDPTA'98 (Internation Conf. on Parallel and Distributed Processing Techniques and Applications), pp. 974-980, Vol. II, 1998.