Journal of Korea Multimedia Society Vol. 7, No. 2, February 2004(pp. 264-271)

인터넷 기반 클러스터 시스템 환경에서 효율적인 부하공유 기법

최인복[†], 이재동[††]

요 약

인터넷기반의 클러스터 시스템 환경에서 부하공유 알고리즘은 네트워크의 특성 및 노드의 이질성에 따른 부하 불균형에 효과적으로 대처할 수 있어야 한다. 본 논문에서 제안하는 효율적인 부하공유 기법은 Weighted Factoring 알고리즘을 기반으로 스케줄러를 생성하고 여기에 적응할당정책과 개선된 고정 분할 단위 알고리즘을 적용하여 작업을 분배하는 것이다. 본 논문에서 적용한 적응할당정책은 상대적으로 작업속도가 느린 종노드의 작업을 빠른 종노드가 대신 수행하도록 하는 기법이며, 개선된 고정 분할 단위 알고리즘은 종노드의 계산시간과 데이터전송에 필요한 네트워크 통신시간을 겹치도록 하는 것이다. 제안된 알고리즘의 성능 평가를 위한 시스템 환경에서 멀티미디어 응용에 많이 사용되는 행렬의 곱셈 프로그램을 PVM을 통하여 실험한 결과, 본 논문에서 제안한 알고리즘이 NOW 환경에서 우수한 Send, GSS, Weighted Factoring 알고리즘보다 각각 75%, 79%, 그리고 17% 효율적임을 보였다.

An Efficient Load-Sharing Scheme for Internet-Based Clustering Systems

In-Bok Choi[†], Jae-Dong Lee[††]

ABSTRACT

A load-sharing algorithm must deal with load imbalance caused by characteristics of a network and heterogeneity of nodes in Internet-based clustering systems. This paper has proposed the Efficient-Load-Sharing algorithm. Efficient-Load-Sharing algorithm creates a scheduler based on the WF(Weighted Factoring) algorithm and then allocates tasks by an adaptive granularity strategy and the refined fixed granularity algorithm for better performance. In this paper, adaptive granularity strategy is that master node allocates tasks of relatively slower node to faster node and refined fixed granularity algorithm is to overlap between the time spent by slave nodes on computation and the time spent for network communication. For the simulation, the matrix multiplication using PVM is performed on the heterogeneous clustering environment which consists of two different networks. Compared to other algorithms such as Send, GSS and Weighted Factoring, the proposed algorithm results in an improvement of performance by 75%, 79% and 17%, respectively.

Key words: Cluster System(클러스터 시스템), Load-Sharing(부하공유), Message Passing(메시지전달) Weighted Factoring Algorithm(WF 알고리즘)

※ 교신저자(Corresponding Author) : 이재동, 주소 : 서울시 용산구 한남동 산 8번지(140-714), 전화 : 02)709-2854, FAX : 02)709-2854, E-mail : letsdoit@dku.edu
접수일: 2003년 5월 30일, 완료일 : 2003년 7월 11일
[†] 준회원, 단국대학교 대학원 컴퓨터과학 및 통계학과 박사과정
(E-mail : pluto612@dku.edu)
[††] 정회원, 단국대학교 정보컴퓨터학부 교수
※ 이 연구는 2002학년도 단국대학교 대학 연구비의 지원으로 연구되었음.

1. 서 론

고성능 컴퓨팅의 중요성이 증대되면서 1980년대 이후 슈퍼컴퓨터와 대용량 병렬 컴퓨터가 폭넓게 활용되어 왔다. 최근에는 비행 시뮬레이션, 가상현실, 그리고 GIS와 같은 대규모 멀티미디어 정보를 처리하기 위하여 병렬처리 기법이 많이 활용되고 있다[5]. 그러나, 고성능 컴퓨팅의 요구가 늘어나면서 이

러한 시스템들의 높은 가격, 유지보수의 어려움 등 여러 가지 문제점이 드러나고 있다. 최근 마이크로프로세서의 뛰어난 성능향상과 고속 네트워크의 보급으로 단일 컴퓨터들을 네트워크로 연결함으로써 하나의 병렬 컴퓨터처럼 작동하는 클러스터 컴퓨팅이 가능해졌다. 클러스터의 대표적인 예로는 140개의 알파시스템을 연결한 미국 로스알라모스 국립연구소의 아발론 클러스터, 버클리대학의 NOW(Network of Workstation) 클러스터, 그리고 미국항공우주국(NASA)의 Beowulf 클러스터 등이 있다. 클러스터를 사용하는 이유는 무엇보다도 가격대 성능비가 우수하고 자체제작이 가능하여 문제발생시 해결이 용이하기 때문이다[4,7].

대부분의 클러스터 컴퓨터는 보다 나은 성능향상을 위해 Myrinet, SCI 또는 기가비트 이더넷과 같은 고속네트워크에 연결하거나, VIA나 MyrinetGM 등과 같이 사용자 수준의 특수한 프로토콜을 이용하고 있다. 하지만 이러한 방법들은 지역적으로 한정된 특별한 네트워크를 구성해야 하므로 확장성에 문제가 있을 뿐 아니라 추가적인 비용이 필요하다[1,13].

인터넷이 발달하면서 대부분의 컴퓨터는 TCP/IP 프로토콜에 의해 네트워크에 연결되어 있으므로 인터넷에 연결되어 있는 컴퓨터들을 추가적인 네트워크의 구성이나 특수한 프로토콜 없이 클러스터를 구성하는 것이 가능해졌다. 하지만, 인터넷 기반의 클러스터 환경은 기존의 클러스터 환경과 많은 차이가 있다. 인터넷에는 여러 종류의 다양한 네트워크 환경들이 혼재되어 있고 이질적인 노드들로 구성되어 있다. 따라서, 인터넷 기반의 클러스터 환경에서는 기존의 클러스터 환경에서 고려하지 않았던 다양한 네트워크의 특성, 부하불균형 그리고 노드의 이질성과 같은 다양한 수행 환경의 변화에도 효과적으로 적응할 수 있도록 해야 한다[4,7].

따라서, 본 논문에서는 인터넷 기반의 클러스터 환경에서 Message Passing 방식의 고성능 클러스터 컴퓨팅 작업시 최적의 효율을 얻을 수 있도록 하는 부하공유기법을 제안한다. 제안하는 부하공유 기법은 이기종 클러스터 환경에서 효율적이라고 연구된 Weighted Factoring 알고리즘[9]을 기반으로 상대적으로 느린 종노드의 작업을 빠른 종노드가 빠른 종노드가 대신 수행하도록 하는 적응할당 정책을 적용하는 동시에 종노드의 계산시간과 데이터 전송에 필요한 네트워크 통신시간을 겹치게 하는 개선된 고정 분할 단위 알고리즘[1]을 적용하는 것이다. 본 논문에서 제안한 부하공유 기법이 인터넷기반의 클러스터 환경에서 효율적임을 보이기 위해 두 개의 네트워크로 구성된 클러스터 환경을 구성하고, PVM을 이용한 행렬의 곱셈을 계산하는 프로그램을 알고리즘별로 구현하여 성능을 측정한다[8,14].

본 논문은 다음과 같이 구성된다. 2장에서는 클러스터 환경에서의 부하공유를 위한 관련연구들을 소개하고, 3장에서는 본 논문에서 제안하는 효율적인 부하공유 기법을 살펴본다. 4장에서는 인터넷기반 클러스터 환경에서의 부하공유 알고리즘들의 성능을 실험을 통해 비교해 보고, 5장에서는 결론 및 향후 발전 과제를 언급하도록 한다.

2. 관련 연구

고성능 클러스터 시스템에서 스케줄링은 크게 시간분할(time sharing) 방법과 공간분할(space sharing) 방법으로 나눌 수 있다. 시간분할 방법은 여러 개의 프로그램들이 코스케줄링(coscheduling) 등을 통하여 전체 시스템을 공유하는 것이고, 공간분할 방법은 시스템을 여러 개로 분할시켜 각각의 분할된 영역에 하나의 프로그램을 수행하는 것이다. 많은 연구에서 공간분할 방법이 시간분할 방법보다 우수하다고 알려져 있다[2,6].

부하공유란 공간분할 스케줄링 방법에서 빠른 실행결과를 얻기 위한 방법을 말한다. NOW(Network of Workstation) 환경에서 부하공유를 위한 알고리즘으로 Send 알고리즘과 GSS 알고리즘이 좋은 성능을 보인 것으로 Piotrowski와 Dandamudi의 연구결과에 나타났다[1,10-12]. Flynn Hummel의 연구 결과에 의하면 이기종 클러스터 환경에서는 Weighted Factoring 알고리즘이 좋은 성능을 나타냈다[9].

Send 알고리즘은 주노드에서 응답이 빠른 종노드에게 스케줄큐에 있는 분할 단위만큼의 다음 작업을 먼저 할당하는 FCFS(First-Come-First-Serve)방식의 대표적인 비선점형 스케줄링 알고리즘이다.

GSS(Guided Self-Scheduling) 알고리즘은 전체 데이터 중에서 아직 남아있는 데이터의 크기에 대하여 일정비율의 데이터를 할당하여 데이터의 크기를 점점 줄여나간다. N개의 데이터와 P개의 종노드에 대하여 다음 종노드에게 스케줄링할 i번째 덩어리의 크기(G_i)는 (식 1)과 같이 결정된다.

$$G_i = \left\lceil (1-\frac{1}{P})^i \frac{N}{P} \right\rceil \quad \text{(식 1)}$$

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 크기(F_{ij})는 (식 2)와 같이 결정된다[9,10].

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

3. 효율적인 부하공유 알고리즘의 설계

이 장에서는 인터넷기반 클러스터 시스템 환경에서의 효율적인 부하공유 알고리즘을 설계한다. 지금까지 이기종 클러스터 환경에서의 부하공유 기법으로는 Weighted Factoring 알고리즘이 우수하다고 연구되어져 있다[9]. 따라서, 여기서는 Weighted Factoring 알고리즘에 기반으로 한 효율적인 부하공유 알고리즘을 설계한다.

3.1 전역 스케줄러 및 서브루틴 설계

Weighted Factoring 알고리즘에 적응할당정책을 적용하기 위해서는 각 종노드의 성능 및 작업할당에 대한 정보를 관리할 수 있는 스케줄러를 운영하는 것이 필요하며(전역 스케줄러), Message Passing 방식의 기본적인 명령인 send/receive 명령 수행에 대하여 추가적인 스케줄러 정보변경 작업이 필요하다.

N개의 데이터와 P개의 종노드에 대한 전역 스케줄러의 데이터 구조는 다음과 같다.

```
struct slave_node
{
  int job[ ⌊ log2N ⌋ ]; //할당된 작업
  int status[ ⌊ log2N ⌋ ]; //수행상태(0:미수행,
  1:전송/수행중, 2:완료)
  float weight; //가중치(종노드의 성능)
  int remain; //미수행중인 job의 크기
} schedule[P];
```

이 전역 스케줄러에서 각 종노드에는 $\lfloor \log_2 N \rfloor$ 개의 작업이 할당된다. 각 작업의 상태는 status 변수에 의해 관리되며, weight은 종노드의 상대적인 성능을 나타낸다. 그리고 remain 변수는 적응할당정책을 적용하기 위해 수행해야할 작업량을 유지한다.

주노드가 종노드에 작업을 할당할 때에는 주로 send 함수를 이용하며, 주노드는 종노드의 상태를 관리하기 위하여 전역 스케줄러에 다음과 같은 추가적인 작업들을 수행한다.

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

SEND 서브루틴에서는 해당 job의 상태를 '전송/수행중'으로 설정하고(2행), '미수행'작업량을 줄인다(3행).

종노드로부터 임의의 부분적인 결과데이터를 전달받기 위해서는 receive 함수를 이용하게 되는데, 이 때에도 전역 스케줄러에 다음과 같은 추가적인 작업들을 수행하여 종노드의 상태를 관리한다.

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

RECEIVE 서브루틴에서는 종노드로부터 부분적인 결과를 전송 받으면(1행), 해당 작업에 대한 상태를 '완료'로 설정한다(2행).

3.2 적응할당정책 적용 기법

Weighted Factoring 알고리즘은 작업 초기에 수행된 종노드의 성능평가에 의한 가중치만을 이용하여 부하를 조절하기 때문에 작업 도중에 발생하는 종노드의 변화에는 대처하기 어려운 단점을 가지고 있다. 일반적으로, 클러스터 시스템에서는 부하가 큰 노드에서 일부 작업을 부하가 작은 노드로 이동시키는 work stealing(작업 이주) 기법이 사용되고 있지만, work stealing 기법은 데이터 재분배를 위한 추가

적인 통신 오버헤드와 이로 인한 또 다른 부하 불균형을 초래할 수 있는 단점을 가지고 있다. 이질적인 계산능력을 가진 NOW 환경에서의 연구에 의하면, work stealing 기법보다는 우선 순위를 낮추는 적응할당정책이 좋은 성능을 보였으며[2], Hummel은 가장 느린 몇 개의 종노드들을 인위적으로 제외시킴으로써 수행시간이 단축됨을 보였다[9].

이러한 연구들을 기반으로 볼 때, 작업도중 발생하는 종노드의 변화에 효율적으로 대처하기 위해서는 스케줄러 상의 작업을 모두 마친 종노드가 성능에 비해 가장 느리게 작업을 수행하는 종노드의 작업을 대신 처리하도록 하는 적응할당정책을 적용하는 것이 바람직하다. 주노드는 임의의 종노드로부터 부분적인 결과 데이터를 전송 받았을 경우, 해당 종노드의 다음 작업을 선정해야 한다. 이 때, 주노드는 이러한 적응할당 정책을 적용하여 해당 종노드의 다음 작업을 아래의 Job_by_AGS와 같이 선정한다.

임의의 종노드 i로부터 schedule[j].[k]의 부분적인 결과 데이터를 전송 받았을 경우, 주노드는 종노드 i의 다음 작업을 선정하기위하여 아래의 함수와 같이 수행한다.

```
Function Job_by_AGS (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= ⌊ log2N ⌋ -1; index>0; index--) {
16 if (schedule[m].status == 0) {
17 n = index;
18 break;
19 }
20 }
21 return (schedule[m].job[n]);
```

해당 종노드의 스케줄러에 아직 작업이 남아 있는 경우(1행)에는 스케줄에 따라 차례로 작업을 수행한다(2~3행). 해당 종노드에 대한 작업을 모두 마쳤을 경우에는 다른 종노드의 작업상황을 검색한다. 다른 종노드에 아직 '미수행'작업이 남아 있는 경우(5행)에는 성능(weight)에 비해 작업 수행이 가장 늦은 종노드를 선택하고(6~14행), 선택된 종노드의 아직 전송되지 않은 작업 중 가장 마지막 작업을 선택하여(15~19행) 대신 수행하도록 한다. 이렇게 함으로써 자연스럽게 성능에 비해 느린 종노드의 계산량을 줄이는 적응할당정책을 적용하게 된다.

3.3 개선된 고정 분할 단위 알고리즘 적용 기법

NOW 환경에서 Send 알고리즘을 개선한 preSend 알고리즘에서는 네트워크 통신시간과 계산시간을 겹치게 함으로써 수행시간을 단축시켰다. preSend 알고리즘에서 주노드는 종노드에게 분할 단위만큼 작업을 전송한 후 종노드로부터 결과를 수신할 때까지 휴지상태를 유지하는 것이 아니라, 적당한 시간이 지나면 종노드로부터의 결과 수신 여부와 관계없이 다음 작업을 미리 전송하도록 하여 종노드의 계산시간과 통신시간을 겹치도록 하였다[1].

인터넷 환경은 다양한 네트워크와 이질적인 노드들로 구성되어 있어서 NOW 환경에서의 preSend 알고리즘과 같이 적당한 시간간격마다 작업을 전송하는 것은 어려운 문제이다. 따라서, 여기에서는 주노드가 각 종노드에 우선 두개씩의 작업을 연속적으로 전송한 후, 결과를 전송한 종노드에 처리하고자하는 다음 작업을 전송하도록 한다. 이렇게 함으로써 각 종노드는 하나의 작업을 수행하여 결과를 주노드에 전송한 후, 다음 작업의 수신을 위해 휴지 상태로 가지 않고 즉시 다음 작업을 수행할 수 있다. 이러한 방법을 통하여 종노드의 계산시간과 네트워크 통신시간을 겹치도록 하여 전체적인 수행시간을 줄일 수 있다.

이러한 방법에 기반을 둔 LS_by_preSend 알고리즘은 다음과 같다.

주노드는 처음 하나의 작업을 종노드에 보낸 후(1행), 종노드가 첫 번째 작업을 수행하는 동안 두 번째 작업을 보낸다(2행). 이후, 종노드에서는 하나의 작업이 완료되면 주노드로부터 새로운 작업을 기다릴 필요 없이 작업을 수행하는 동안 전송받은 다음 작업

Algorithm LS_by_preSend
- **Input** : *P*, number of slave nodes;
  schedule[*P*], a scheduled array for *P* slave nodes
- **Output** : *result*, merged partial data which are received from slave nodes (e.g. array, a variable)

1 **SEND** (*schedule*[0...(*P*-1)].*job*[0], 0...(*P*-1));
2 **SEND** (*schedule*[0...(*P*-1)].*job*[1], 0...(*P*-1));
3 **while** (all partial results are not gathered by master node) {
4 **RECEIVE** (*schedule*[j].*job*[k], i);
5 *schedule*[m].*job*[n] = **Job_by_AGS** (*schedule*[j].*job*[k], i);
6 **SEND** (*schedule*[m].*job*[n], i);
7 **MERGE** (pratial result data of *schedule*[j].*job*[k] to the *result*);
8 }

을 수행할 수 있으므로 계산시간과 네트워크 시간을 겹치도록 하여 전체 계산시간을 줄일 수 있다.

3.4 Efficient-Load-Sharing 알고리즘의 설계

여기에서는 본 논문에서 제안한 인터넷기반의 클러스터 환경에서 부하공유에 효율적인 Efficient-Load-Sharing 알고리즘에 대하여 살펴보도록 한다.

Efficient-Load-Sharing 알고리즘은 Weighted Factoring 알고리즘에 의하여 스케줄링 함으로써 기본적인 부하공유를 수행한다. 여기에 앞 절에서 설명한 Job_by_AGS 함수를 이용하여 적응할당정책기법을 적용하고, LS_by_preSend 알고리즘을 이용하여 개선된 고정 분할 단위 알고리즘을 적용함으로써 추가적인 효율을 얻을 수 있도록 한다.

이러한 Efficient-Load-Sharing 알고리즘은 다음과 같다.

주노드는 초기 수행된 종노드의 성능평가에 의해 결정된 가중치를 이용하여 각 종노드마다 Weighted Factoring 알고리즘에 의한 스케줄러를 운영한다(1,2행). 주노드는 처음 하나의 작업을 각 종노드에게 보낸 후(3행), 응답을 기다리지 않고 미리 두 번째 작업을 보냄으로써 개선된 고정 분할 단위 알고리즘을 적용한다(4행). 그리고, Job_by_AGS 서브함수를 이용하여 상대적으로 느린 종노드의 작업을 빠른 종노드가 수행하게 하는 적응할당정책을 적용한다(7행).

Algorithm Efficient-Load-Sharing
- **Input** : N, size of job; P, number of slave nodes
- **Output** : result, merged partial data which are received from slave nodes (e.g. array, a variable)

1 **EVALUATE** (performance of slave nodes);
2 **CREATE** (scheduler by Weighted Factoring algorithm);
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 *schedule*[m].*job*[n] = **Job_by_AGS** (*schedule*[j].*job*[k], i);
8 **SEND** (*schedule*[m].*job*[n], i);
9 **MERGE** (pratial result data of *schedule*[j].*job*[k] to the *result*);
10 }

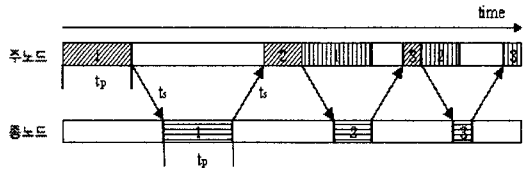
3.6 Efficient-Load-Sharing 알고리즘의 분석

Efficient-Load-Sharing 알고리즘에서는 프로그램의 총 수행시간을 줄이기 위해서 종노드의 계산시간과 데이터 전송에 필요한 네트워크 통신시간을 겹칠 수 있도록 다음의 작업을 미리 전달하도록 하였다. 이렇게 수행하였을 때 얻을 수 있는 총 수행시간의 이득은 주노드 및 종노드가 하나의 작업에 대해 계산을 수행하고 있는 시간에 대해 다른 작업이 전송되는 네트워크시간이다. 아래의 그림 1은 하나의 작업에 대해 주노드와 종노드의 수행시간(t_p)이 같고 네트워크 전송시간(t_s)이 같을 때, 네트워크시간과 계산시간을 겹치지 않았을 경우와 겹치게 했을 경우의 총 수행시간 이득을 보여주는 예이다.

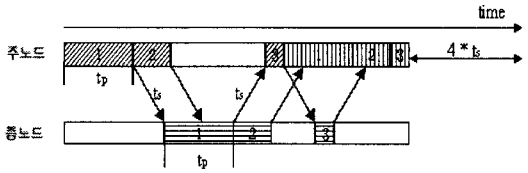
그림 1의 (b)경우와 같이 최대의 시간 이득을 위해서는 맨 처음 주노드에서 종노드로 전송되는 시간과 마지막 종노드에서 주노드로 전송된 시간을 제외한 모든 네트워크 전송시간이 계산시간과 겹쳐져야 한다.

따라서, N개의 데이터와 P개의 종노드에 대해 네트워크시간과 계산시간을 겹치게 함으로써 얻을 수 있는 총 수행시간의 최대 이득은 아래의 (식 3)과 같다.

$$t_{profit} = P*((\lfloor \log_2 N \rfloor * 2t_s) - 2t_s) \qquad \text{(식 3)}$$



(a) 계산시간과 네트워크 통신시간을 겹치지 않았을 경우



(b) 계산시간과 네트워크 통신시간을 겹쳤을 경우

▨: send ▤: make result ▥: merge result

그림 1. 종노드의 계산시간과 데이터전송에 필요한 네트워크 통신시간을 겹칠 때의 시간이득

하나의 종노드에 대해 $\lfloor \log_2 N \rfloor$ 개의 작업이 할당되며, 각 작업은 주노드와 종노드 사이에서 2번씩 전송되게 된다. 여기에서 처음과 마지막의 전송시간을 제외하면 하나의 종노드에 대해 얻을 수 있는 최대 시간이득이 된다. 따라서, 이를 P개의 종노드에 대하여 적용하면, (식 3)과 같은 결과를 얻을 수 있다.

4. 성능 평가

여기에서는 3장에서 제안한 Efficient-Load-Sharing 알고리즘이 인터넷기반 클러스터 환경에서 다른 알고리즘보다 효율적임을 보이기 위해 Send 알고리즘, GSS 알고리즘, 그리고 Weighted Factoring 알고리즘과의 성능을 비교평가 하였다. 성능평가를 위하여 멀티미디어 응용 분야에서 많이 이용되는 연산인 행렬의 곱셈 프로그램을 PVM3.4.4 라이브러리를 이용하여 작성하였다.

4.1 실험 환경

실험을 위하여 그림 2와 같이 총 11개의 컴퓨터를 이용하여 클러스터 환경을 구성하였으며, 인터넷기반의 분산된 환경을 위하여 두 개의 네트워크에 분산시켜 배치하였다.

하나의 주노드와 10개의 종노드로 구성하였으며, 주노드와 종노드사이에는 메시지 전달을 이용하여

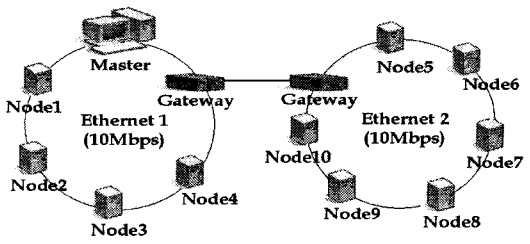


그림 2. 실험환경

통신함으로써 프로그램을 수행한다. 그리고 각 종노드의 가중치(weight)의 차이를 늘리기 위해 임의적으로 성능이 낮은 노드들을 주노드와는 다른 네트워크에 위치하도록 하였다. 그림 2의 실험환경에서 각 Node명, CPU 성능, 메모리 용량 및 운영체제는 표 1과 같다. 이기종의 환경을 구성하기 위하여 각 노드의 성능을 다르게 설정하였으며, 운영체제도 Linux와 Solaris를 혼합하여 배치하였다.

주노드는 A. Piotrowski의 연구[12]에서 얻어진 결과에 따라 전체적인 계산성능을 향상시키기 위하여 비교적 낮은 CPU 성능의 PC를 선택하였으며, 주노드의 운영체제는 커널 및 네트워크성능에서 우수하다고 측정된 Linux를 선택하였다[2,3].

실험에서 주노드 프로그램은 각 알고리즘별로 다르게 구현하였고, 종노드 프로그램은 하나로 구현하여 동일하게 적용하였다. 구현된 실험 프로그램들에서 주노드는 부분적인 결과행렬의 index들을 종노드에 전송하고, 종노드는 이에 해당하는 결과 값들을 index와 함께 주노드에 전송하도록 하였다. 따라서, Efficient-Load-Sharing 알고리즘에서 표현된

표 1. 하드웨어/소프트웨어 구성표

Node명	CPU	메모리	운영체제
Master	Pentium3 450	320M	Linux(kernel 2.4)
Node1	Pentium3 733	128M	Linux(kernel 2.4)
Node2	Pentium3 733	128M	Linux(kernel 2.4)
Node3	Pentium3 450	128M	Linux(kernel 2.4)
Node4	Pentium3 300	128M	Solaris 8.0
Node5	Pentium3 300	128M	Solaris 8.0
Node6	Pentium3 450	128M	Linux(kernel 2.4)
Node7	Pentium pro 133	64M	Linux(kernel 2.0)
Node8	Pentium pro 133	32M	Linux(kernel 2.0)
Node9	Pentium pro 133	32M	Linux(kernel 2.0)
Node10	Pentium pro 133	16M	Linux(kernel 2.0)

schedule[j],*job*[k]은 결과행렬의 index와 결과 값의 배열이 된다.

구현된 실험 프로그램들의 동작을 단계별로 설명하면 다음과 같다.

① 주노드는 연산에 필요한 두 행렬을 각 종노드에 전송한다.

② 주노드는 각 알고리즘에 따라 작업을 분할하고, 분할된 크기에 해당하는 결과행렬의 index배열을 종노드에 전송한다.

③ 종노드는 주노드로부터 전송받는 index에 해당 결과를 추가하여 주노드에 전송한다.

④ 주노드는 종노드로부터 전송받은 부분적인 결과를 병합한다.

4.2 실험 결과

4.1절에서 설명된 실험환경에서 각 알고리즘에 의하여 200*200, 300*300, 400*400, 500*500 크기의 행렬의 곱을 계산하는 프로그램을 수행하였다.

Weighted Factoring 알고리즘과 Efficient-Load-Sharing 알고리즘의 성능평가에서는 종노드의 성능평가와 스케줄러를 생성하는 시간을 제외하였다. 왜냐하면, 종노드의 성능평가에 의한 스케줄러의 생성은 프로그램 실행 시 매번 필요한 것이 아니라, 클러스터를 구성할 처음에만 필요하기 때문이다. 따라서, 알고리즘의 총 수행시간은 실제로 종노드에 필요한 부분적인 데이터를 전달하면서 시작하여 종노드로부터 마지막으로 전달된 부분적인 결과 데이터가 주노드에 의해 병합된 시간까지 측정하였다.

각 행렬의 크기별로 20회의 반복적인 프로그램 실행을 통하여 얻은 평균실행시간(초)을 기록한 결과, 아래의 표 2와 그림 3과 같은 결과를 얻을 수 있었다. 여기서 WF는 Weighted Factoring 알고리즘을 의미하며, ELS는 Efficient-Load-Sharing 알고리즘을 말한다.

표 2. 데이터크기별 알고리즘 수행시간(단위:초)

알고리즘 / 행렬크기	Send	GSS	WF	ELS
200*200	31.26	33.89	8.96	8.85
300*300	34.67	38.88	19.00	15.34
400*400	35.44	42.96	23.87	19.27
500*500	39.88	54.47	32.46	23.52

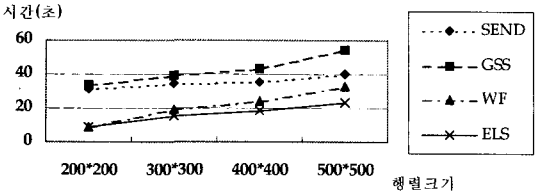


그림 3. 데이터크기별 알고리즘 수행시간

그림 3의 실험결과에 의하면, 데이터의 크기가 작을 때에는 비슷한 성능을 보이던 Weighted Factoring 알고리즘이 데이터의 크기가 커질수록 Efficient-Load-Sharing 알고리즘보다 성능이 저하되는 현상을 보인다. 이는 초기에 평가된 종노드의 성능이 변화가 심한 인터넷 환경에서는 적합하지 못하기 때문이다. 따라서, 데이터의 크기가 커져 메시지 전달의 횟수가 많아질수록 종노드의 상태 변화에 동적으로 적응할 수 있도록 개선된 Efficient-Load-Sharing 알고리즘이 높은 성능을 보인다.

행렬의 크기별 성능향상에 대한 평균값으로 결과를 분석해 보면, Efficient-Load-Sharing 알고리즘이 Send 알고리즘보다 75%, GSS 알고리즘보다 79%, WF 알고리즘보다 17%의 성능향상을 보임을 알 수 있다.

5. 결론 및 향후 과제

본 논문에서는 인터넷기반 클러스터 환경에서 부하공유에 효율적인 Efficient-Load-Sharing 알고리즘에 대하여 소개하였다. Efficient-Load-Sharing 알고리즘은 Weighted Factoring 알고리즘을 기반으로 상대적으로 작업속도가 느린 종노드의 작업을 빠른 종노드가 대신 수행하는 적응할당정책을 적용하는 동시에 종노드의 계산시간과 데이터 전송에 필요한 네트워크 통신시간을 겹치게 하는 개선된 적응할당 정책을 적용하였다.

인터넷기반 클러스터 환경에서의 Efficient-Load-Sharing 알고리즘의 성능을 측정하기 위하여 두 개의 분산된 네트워크 상에서 PC를 이용하여 이기종의 클러스터를 구축하고, 멀티미디어 응용에 많이 사용되는 행렬의 곱셈 프로그램을 PVM을 이용하여 200*200, 300*300, 400*400, 500*500의 크기별로 수행하였다. 실험의 결과를 행렬의 크기별 성능향상에 대한 평균값으로 분석해 본 결과, 본 논문에서 제안

한 Efficient-Load-Sharing 알고리즘이 NOW 환경에서 효율적인 Send 알고리즘보다 75%, GSS 알고리즘보다 79%의 성능향상을 보였으며, Weighted Factoring 알고리즘보다도 17%의 성능향상을 보였다.

인터넷기반 클러스터 환경은 NOW 환경보다 더욱 동적이며, 이러한 환경에서 효율적인 부하공유를 위해서는 클러스터를 구성하는 노드들의 성능과 네트워크의 성능뿐만 아니라 노드 및 네트워크의 고장까지 고려해야만 한다. 따라서, 향후에는 더욱 다양한 환경에서 적용이 가능한 부하공유기법에 대한 연구와 네트워크의 고장 및 노드의 고장에 대처할 수 있는 결함허용에 대한 연구가 필요하다.

참 고 문 헌

[1] 구본근, "NOW 환경에서 개선된 고정 분할 단위 알고리즘", 정보처리학회논문지, Vol. 8 No. 2, pp. 117-124, 2001.

[2] 김진성, 심영철, "이질적 계산 능력을 가진 NOW를 위한 공간 공유 스케쥴링 기법", 정보과학회논문지, Vol. 27 No. 7, pp. 650-664, 2000.

[3] 박윤용, 박정호, 임동선, "이종 분산 환경에서 UNIX 커널 성능 측정 방법에 관한 연구", 정보처리학회지, Vol. 6 No. 11, pp. 2954-2964, 1999.

[4] 유찬수, "리눅스 클러스터링", 정보과학회지, Vol. 18 No. 2, pp. 33-39, 2000.

[5] 정상화, 류광렬, 고윤영, 곽민석, "병렬 GIS를 위한 효율적인 분산공유메모리 시스템", 정보과학회논문지, Vol. 5 No. 6, pp. 700-707, 1999.

[6] 정훈진, 정진하, 최상방, "네트워크 기반 클러스터 시스템을 위한 적응형 동적 부하균등 방법", 정보과학회논문지, Vol. 28 No. 11, pp. 549-560, 2001.

[7] R. Buyya, "High Performance Cluster Computing", vol. 1, Prentice Hall, 1999.

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R.Manchek, and V. Sunderam, "PVM : Parallel Virtual Machine - A Users' Guide and Tutoraial for Networked Parallel", The MIT Press, 1994.

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

[10] 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.

[11] G. Pfister, "In Search of Clusters", 2nd Edtion, Prentice Hall, 1998.

[12] 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.

[13] IEEE Task Force on Cluster Computing (TFCC), http://www.ieeetfcc.org

[14] Message Passing Interface Forum, "MPI : A Message-Passing Interface Standard", May, 1994.

최 인 복

1999년 단국대학교 전자계산학과 학사
2002년 단국대학교 대학원 전자계산학과 석사
2002년~현재 단국대학교 대학원 컴퓨터과학및통계학과 박사과정
관심분야 : 분산 및 병렬처리, (모바일)인터넷 기술, 컴퓨터 네트워크

이 재 동

1985년 인하대학교 전자계산학과 학사
1991년 Cleveland State Univ. Dept. of Computer & Information Science (M.S.)
1996년 Kent State Univ. Dept. of Computer Science(Ph.D.)
1997년~1988년 대우중공업 정보관리실 시스템 분석
1992년~1996년 Kent State University R.A & T.A
1996년~1997년 (주)두루넷 기술기획팀
1997년~현재 단국대학교 정보컴퓨터학부 교수
관심분야 : (Mobile) Internet Technologies/Applications, High Performance Computing(Clustering systems etc.), GIS Technologies and Applications, Many aspects of parallel/distributed processing