韓國 컴퓨터情報學會 論文誌
第13卷 第5號, 2008. 9.

2008-13-5-1-6

고성능 컴퓨팅 시스템을 위한 효율적인 동적 작업부하 균등화 정책

이 원 주*, 박 말 순**

An Efficient Dynamic Workload Balancing Strategy for High-Performance Computing System

Won Joo Lee*, Mal Soon Park**

요 약

본 논문에서는 고성능 컴퓨팅 시스템의 성능 향상을 위한 효율적인 동적 작업부하 균등화 정책을 제안한다. 이 정책은 시스템 자원인 CPU와 메모리를 효율적으로 사용하여 고성능 컴퓨팅 시스템의 처리량을 최대화하고, 각 작업의 수행시간을 최소화한다. 또한 이 정책은 수행중인 작업의 메모리 요구량과 각 노드의 부하 상태를 파악하여 작업을 동적으로 할당한다. 이때 작업을 할당 받은 노드가 과부하 상태가 되면 다른 노드로 작업을 이주시켜 각 노드의 작업부하를 균등하게 유지함으로써 작업의 대기시간을 줄이고, 각 작업의 수행시간을 단축한다. 본 논문에서는 시뮬레이션을 통하여 제안하는 동적 작업부하 균등화 정책이 기존의 메모리 기반의 작업부하 균등화 정책에 비해 고성능 컴퓨팅 시스템의 성능 향상 면에서 우수함을 보인다.

Abstract

In this paper, we propose an efficient dynamic workload balancing strategy which improves the performance of high-performance computing system. The key idea of this dynamic workload balancing strategy is to minimize execution time of each job and to maximize the system throughput by effectively using system resource such as CPU, memory. Also, this strategy dynamically allocates job by considering demanded memory size of executing job and workload status of each node. If an overload node occurs due to allocated job, the proposed scheme migrates job, executing in overload nodes, to another free nodes and reduces the waiting time and execution time of job by balancing workload of each node. Through simulation, we show that the proposed dynamic workload balancing strategy based on CPU, memory improves the performance of high-performance computing system compared to previous strategies.

▸ Keyword : 작업부하(workload), 이주(migration), 부하균등화(load balancing)

• 제1저자 : 이원주 교신저자 : 박말순
• 접수일 : 2008. 4. 28, 심사일 : 2008. 6. 3, 심사완료일 : 2008. 9. 25.
* 인하공업전문대학 컴퓨터정보과 부교수 ** 한양대 컴퓨터공학과 연구원

Ⅰ. 서 론

현재 인터넷이 급속하게 발전하면서 뉴스 사이트, 전자상거래, 검색엔진 등의 인터넷 응용 분야에서 웹 서버, DB 서버, 게임 서버, VOD 서버 등의 수요가 늘어나면서 고성능 컴퓨터 시스템에 대한 관심이 증가하고 있다[1][2]. 고성능 컴퓨터 시스템은 각 제조사의 고유한 설계 기술에 따라 캐비넷 형태로 생산하는 고가의 장비이다. 따라서 구입 및 활용에 상당한 어려움을 가지고 있다. 이러한 문제를 해결하기 위한 하나의 방안은 클러스터 컴퓨팅 시스템을 구축하는 것이다.

클러스터 컴퓨팅 시스템은 저가의 PC 및 워크스테이션을 고속 네트워크 또는 전용 네트워크를 사용하여 연결하여 구축한 시스템이다. 이 시스템의 장점은 저가의 상용 제품을 사용하여 기존 고성능 컴퓨터에 비해 적은 비용으로 동일한 성능의 시스템 구축할 수 있기 때문에 가격 대 성능비가 높다는 것이다. 또한, PC 및 워크스테이션의 개발 환경을 그대로 사용할 수 있기 때문에 프로그램 개발이 용이하다는 것이다[3].

이러한 클러스터 컴퓨팅 시스템의 성능을 향상시키는 한 방법은 작업 부하를 균등하게 유지 하는 것이다. 기존의 작업 부하 균등화 정책은 CPU 활용도를 기준으로 작업을 할당하기 때문에 메모리 접근 빈도수가 높고, 대용량의 메모리를 요구하는 응용 프로그램이 증가하면서 클러스터 컴퓨팅 시스템의 성능을 향상시키는데 한계가 있다. 실제 클러스터 컴퓨팅 시스템에서는 모든 노드들이 메모리를 소유하고 있지만 일부 노드의 메모리만을 사용한다. 따라서 유휴 메모리를 가진 노드에 작업을 할당하여 메모리 사용의 균형을 이룰 수 있는 작업 부하 균등화 정책이 필요하다.

따라서 본 논문에서는 CPU와 메모리 부하를 기반으로 클러스터 컴퓨팅 시스템의 성능을 향상시킬 수 있는 새로운 작업 균등화 정책을 제안한다. 이 정책은 먼저 각 노드의 CPU와 메모리 사용량에 따라 과부하 상태가 아니면 작업을 할당한다. 그리고 수행중인 작업의 메모리 요구량이 가용 메모리 크기를 초과하여 페이지 폴트가 발생하면 수행 중인 작업의 실행을 중지하고, 그 작업을 다른 노드로 이주시킴으로써 작업의 대기시간과 수행시간을 단축한다.

본 논문의 구성은 다음과 같다. 2장에서는 클러스터 컴퓨팅 시스템의 구성과 기존의 작업부하 균등화 정책에 대하여 살펴본다. 3장에서는 제안한 작업부하 균등화 정책에서 사용하는 용어 및 작업부하 균등화 알고리즘에 대하여 자세히 설명한다. 4장에서는 시뮬레이션 환경과 시뮬레이션 결과를 분석한다. 그리고 5장에서 결론을 맺는다.

Ⅱ. 관련 연구

2.1 클러스터 컴퓨팅 시스템 구성

일반적인 클러스터링 시스템은 관리 노드, 계산 노드, 스토리지, 게이트웨이 등으로 구성된다.

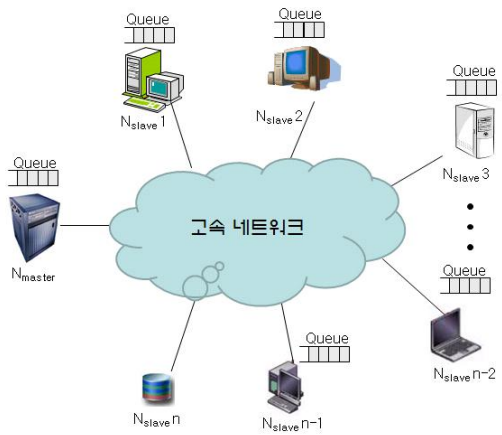

그림 1. 클러스터 컴퓨팅 시스템
Fig. 1. Cluster computing system.

그림 1의 클러스터 컴퓨팅 시스템에서 마스터 노드 Nmaster는 관리 노드 기능을 한다. 관리 노드는 원격 설치, DHCP, 하드웨어 관리를 포함한 클러스터 관리, 작업량 관리, 백업 등의 서비스를 제공한다. 슬레이브 노드 Nslave는 계산 노드와 스토리지 기능을 한다. 계산 노드는 연산을 담당하는 노드이고, 스토리지는 데이터 저장 서비스를 제공한다. 게이트웨이는 관리 노드 상에 존재하며, 외부 네트워크와 연결하는 서비스를 제공한다.

또한 클러스터 컴퓨팅 시스템의 성능은 각 노드를 연결하는 네트워크 성능과 자원 관리 소프트웨어에 따라 달라진다. 자원 관리 소프트웨어는 글로벌 접근, 원격 시스템 모니터링 및 관리, 네트워크와 각 노드의 상태를 진단하는 기능을 제공한다. 또한 다른 소프트웨어로는 각 노드에 운영체제를 설치하는 소프트웨어인 인스톨 유틸리티와 기존 어플리케이션을 간단한 지시어와 프리 컴파일러를 사용하여 병렬화 시키는 기능을 제공하는 병렬 프로그래밍 환경 등이 있다. 그리고 병렬 프로그래밍 라이브러리를 사용하여 메시지 패싱 방법으로 복수 노드에 걸쳐 실행되는 병렬 어플리케이션을 개발하는 것을 지원한다.

2.2 클러스터 컴퓨팅 시스템의 분류

클러스터 컴퓨팅 시스템은 CPU, 메모리, 입출력 등의 시스템 구성 요소와 클러스터 컴퓨팅 시스템에 설치된 운영체제의 동질성 여부에 따라 동질(homogeneous)적 또는 이질(heterogeneous) 적 클러스터 컴퓨팅 시스템으로 분류할 수 있다.

클러스터 컴퓨팅 시스템의 분류 기준은 세 가지로 나눌 수 있다. 첫 번째는 CPU 속도나 메모리 크기, 저장 장치 크기 등의 시스템 구성 요소가 동일한지의 여부에 따라 정해진다. 두 번째는 시스템 구성 요소가 어떤 구조로 이루어져 있는지 여부에 따라 구분하는 것이다. 마지막으로는 시스템에 설치된 운영체제에 따라 구분할 수 있다.

2.3 작업부하 균등화 정책

작업부하 균등화(load balancing) 정책은 각 노드들의 CPU와 메모리, 또는 다른 자원들의 성능을 고려하여 작업을 균등하게 할당하는 것이다. 작업부하 균등화 정책은 CPU 기반 정책과 메모리 기반 정책으로 분류 할 수 있다.

CPU 기반 정책[4][5][6][7][8]은 메모리 크기가 충분하다는 가정 하에 CPU 처리속도나 CPU 작업 큐의 길이를 기준으로 작업을 할당한다. 이 정책은 작업 크기와 실행시간을 미리 예측해야 한다는 어려움이 있다.

메모리 기반 정책[9][10]은 CPU 처리속도에 여유가 있다는 가정 하에 가용 메모리의 크기를 기준으로 작업을 할당한다. 기존의 작업부하 균등화 정책은 가용 메모리 또는 CPU 처리속도가 충분한 시스템에 적용해야만 좋은 성능을 얻을 수 있다는 문제점이 있다.

본 논문에서는 이러한 문제점을 해결하기 위해 CPU 처리속도와 메모리 크기에 기반한 작업부하 균등화 정책을 제안한다.

III. 동적 작업부하 균등화 정책

본 논문에서 제안하는 동적 작업부하 균등화 정책은 이질적 클러스터 컴퓨팅 시스템에서 CPU와 메모리 자원을 고려하여 작업을 할당한다. 이 정책은 클러스터 컴퓨팅 시스템의 성능 향상을 위해 과부하 노드의 작업을 유휴 노드로 이주시켜 수행함으로써 작업의 대기시간을 줄인다. 또한 작업의 메모리 요구량을 반영하여 동적으로 작업을 할당하거나 유휴 노드로 이주시켜 수행함으로써 메모리 부족으로 인한 페이지 폴트 발생을 줄인다.

3.1 용어 정의

본 논문에서 제안하는 동적 작업부하 균등화 정책에서 사용하는 용어를 먼저 정의한다.

정의 1. CPU 한계점(CPU Threshold)은 CPU가 처리할 수 있는 최대 작업의 수이다.

정의 2. 메모리 한계점(Memory Threshold)은 전체 메모리에서 시스템 파일이 차지하는 메모리를 제외한 가용 메모리이다.

CPU 한계점과 메모리 한계점은 각각 CTi와 MTi로 표현한다. CPU의 처리속도는 MIPS (Millions of Instructions Per Second)로 나타낸다.

정의 3. CPU 부하 인덱스(CPU Load Index)는 CPU에서 수행되고 있는 작업의 수이다.

정의 4. 메모리 부하 인덱스(Memory Load Index)는 수행중인 작업에 할당된 메모리의 총합이다.

각 노드의 CPU 작업 큐에는 대기, 수행, 전송, 페이지 폴트 상태의 작업들이 존재한다. 이러한 작업의 수를 CPU 부하 인덱스라 하며 Q_i로 표현한다. 메모리 부하 인덱스는 ML_i로 표현하며, ML_i이 MT_i보다 크다면 메모리 과부하로 인해 페이지 폴트가 발생한다.

정의 5. 가용 공간(Free Space)은 작업을 할당 할 수 있는 공간이다.

각 노드의 가용 공간은 CPU 가용 공간과 메모리 가용 공간으로 분류 할 수 있으며 각각 $Free(C)_i$와 $Free(M)_i$로 표현한다. $Free(C)_i$와 $Free(M)_i$는 각각 식 (1)과 (2)로 구할 수 있다.

$$Free(C)_i = CT_i - Q_i \quad (1)$$

$$Free(M)_i = MT_i - ML_i \quad (2)$$

각 노드의 가용 공간 정보는 가용 공간 정보 테이블(Free Space Information Table)에 저장된다. 가용 공간 정보 테이블은 FSI_table 로 표현한다. FSI_table의 각 레코드는 $\langle i, Free(C)_i, Free(M)_i \rangle$로 구성되며 $Free(C)_i$와 $Free(M)_i$를 기준으로 내림차순으로 정렬된다. 여기서 i 는 노드 번호

를 의미한다.

3.2 동적 작업부하 균등화 알고리즘

동적 작업부하 균등화 알고리즘은 그림 2와 같이 마스터 작업 균등화 (Master_LoadBalance)와 슬레이브 작업 균등화(Slave_LoadBalance) 모듈로 구성된다.

Master_LoadBalance() 모듈은 작업의 초기 할당을 담당한다. 이 모듈에서는 작업의 메모리 요구량(MRi)을 미리 예측할 수 없기 때문에 FSI_table에서 첫 번째 조건 (Free(C)$_i$〉0 AND Free(M)$_i$〉0)을 만족하는 FSI_table 레코드의 슬레이브 노드에 작업을 할당한다. 만약 첫 번째 조건을 만족하는 레코드를 찾지 못하면, 두 번째 조건 (Free(C)$_i$〈=0 OR Free(M)$_i$〉0)을 만족하는 슬레이브 노드에 작업을 할당한다. 이 경우는 CPU가 과부하 상태이기 때문에 CPU가 유휴 상태가 될 때까지 대기해야만 한다. 만약 두 번째 조건을 만족하는 레코드가 없으면 작업은 마스터 노드의 작업 큐로 복귀한다.

슬레이브 노드에 할당된 작업은 수행에 필요한 메모리를 요구한다. Slave_LoadBalance() 모듈에서는 조건(ML$_i$ + MR$_i$ 〉= MTi)을 만족하면 새로운 작업이 슬레이브 노드에 할당되는 것을 막고, FSI_table의 정보를 수정한다. 그리고 작업을 이주할 최적 할당(best-fit) 슬레이브 노드를 선택한다.

만약 최적 할당 슬레이브 노드가 존재하면 현재 수행중인 작업을 이주시킨다. 그러나 최적 할당 슬레이브 노드가 존재하지 않으면 메모리가 유휴 상태로 될 때까지 대기한다. 슬레이브 노드의 작업 균등화 과정은 이미 예측된 메모리 요구량을 가지고 최적의 노드에 작업을 할당하기 때문에 빈번한 작업 이주를 줄일 수 있다. 또한 메모리 한계점을 초과하지 않기 때문에 페이지 폴트 발생률도 줄일 수 있다.

```
Master_LoadBalance() //Master 노드 작업균등화
{
  while (N_master_Queue 길이 > 0)
  {
    if ( Free(C)_i > 0 ANDFree(M)_i > 0 ) {
      // i-th 슬레이브 노드에 작업(J_m) 할당
      Allocation_N_slave(J_m);
      // 메모리 유휴 결정, MR_i: 메모리 요구량
      if (Free(M)_i > MR_i){
          Execution( J_m);
          Receive(J_m+1);
      }else{
          //In slave node, load balancing
          Slave_LoadBalance(J_m);
      }
    } esle if ( Free(C)_i <= 0 OR_i > 0 ) {
      Allocation_N_slave(J_m);
      // J_m은 CPU가 유휴 상태로 될 때까지 대기
      Waiting(J_m);
      if (Free(M)_i > MR_i ){//MR_i: 메모리 요구량
          Executing( J_m);
          Receive(J_m+1);
      }else{
          //In slave node, load balancing
          Slave_LoadBalance(J_m) ;
      }
    } else {
      //N_master_Queue로 J_m 복귀
      Return N_master_Queue(J_m);
    }
  }
}

Slave_LoadBalance(J_m) //Slave 노드 작업균등화
{
  if ( ML_i + MR_i >= MT_i )
  {
    Block_New_Job(); //새로운 작업 할당 저지
    Update_FSI_table(); //FSI_table의 정보 수정
    //Best-fit 슬레이브 노드 선택
    Best-fit_N_slave = Select_Best-fit_Node();
    If ( Best-fit_N_slave = ){
      //메모리가 유휴 상태로 될 때까지 대기
      waiting();
    }else{
      // migrate J_m to Best-fit_N_slave
      Job_Migration(J_m);
    }
  } else {
    // N_master_Queue로 J_m 복귀
    Return N_master_Queue(J_m) ;
  }
}
```

그림 2. 작업부하 균등화 알고리즘
Fig. 2. Workload balancing algorithm.

IV. 시뮬레이션

본 논문에서는 시뮬레이션을 통하여 제안한 동적 작업부하 균등화 정책이 기존의 CPU 기반 정책에 비해 시스템의 성능 향상 면에서 유리함을 검증한다. 시뮬레이션 방법은 참고문헌 〔11〕과 동일하다.

4.1 시뮬레이션 환경

본 논문에서는 표 1과 같이 32개의 이질적인 노드로 구성한 클러스터 컴퓨팅 시스템을 대상으로 시뮬레이션을 수행한다.

표 1. 클러스터링 컴퓨팅 시스템
Table 1. Cluster computing system.

시스템	CPU 처리 속도 (MIPS)					메모리 크기 (MB)			
	100	200	300	400	500	32	64	128	256
1	2	2	2	13	13	4	4	12	12
2	2	2	2	13	13	12	12	4	4
3	13	13	2	2	2	4	4	12	12
4	13	13	2	2	2	12	12	4	4
5	6	6	8	6	6	8	8	8	8

표 1에서 시스템 1은 CPU 처리 속도가 높고, 메모리 용량이 큰 노드들이 많은 시스템이다(평균 CPU 처리속도=403, 평균 메모리 크기=156). 시스템 2는 CPU 처리 속도가 높고, 메모리 용량이 작은 노드들이 많은 시스템이다(평균 CPU 처리속도=403, 평균 메모리 크기=84). 시스템 3은 CPU 처리속도가 낮고, 메모리 용량이 큰 노드들이 많은 시스템이다(평균 CPU 처리속도=196, 평균 메모리 크기=156). 시스템 4는 CPU 처리속도가 낮고, 메모리 용량이 작은 노드들이 많은 시스템이다(평균 CPU 처리속도=196, 평균 메모리 크기=84). 시스템 5는 CPU 처리 속도와 메모리 용량이 평균인 시스템이다(평균 CPU 처리속도=300, 평균 메모리 크기=120).

또한 이러한 클러스터 컴퓨팅 시스템을 구성하는 각 노드의 시뮬레이션 환경은 표 2와 같다.

표 2. 각 노드의 시뮬레이션 환경
Table 2. Simulation environment of each node.

항목	값
CPU 처리 속도	100 ~ 500 MIPS
메모리 크기	32 ~ 256 MB
시스템 파일 크기(Usys)	16 MB
유휴 메모리 크기(RAMi)	메모리 크기 - Usys
메모리 페이지 크기	4 KB
네트워크 속도	100 Mbps
페이지 폴트 수행시간	10 ms
작업 CPU 처리 시간	10 ms
문맥교환시간	0.1 ms

표 2에서 각 노드는 100-500MIPS 범위의 CPU 처리속도와 32-256MB 범위의 메모리 크기를 가진다. 또한 페이지 폴트 처리시간(10ms), 작업 수행시간(10ms), 네트워크 속도 (100Mbps), 메모리 페이지 크기(4 KB), 문맥교환시간(0.1ms) 등의 항목은 고정된 값으로 설정한다.

작업은 작업발생율[10]에 따라 일정한 시간 간격으로 생성하며 작업번호, 작업 도착시간, CPU 수행시간 등의 속성을 가진다. 작업 도착 시간은 작업이 마스터 노드의 작업 큐에 도착하는 시간을 의미한다. 슬레이브 노드에서 수행중인 작업의 메모리 요구량은 Pareto 분포[6][10]를 따른다. 작업은 발생빈도와 평균 메모리 요구량에 따라 Trace 1, Trace 2, Trace 3로 구분한다. Trace 1은 1,365초 동안 503개의 작업을 발생시키는 작업 발생율을 가진다. Trace 2는 2,503초 동안 420개의 작업을 발생시키는 작업 발생율을 가진다. Trace 3은 3,634초 동안 359개의 작업을 발생시키는 작업 발생율을 가진다. 또한, Trace 1과 Trace 2, Trace 3의 작업 평균 메모리 요구량은 각각 1MB, 2MB, 4MB 이다.

작업의 할당 노드 또는 메모리 요구량은 사전에 알 수 없다. 작업은 가용 공간 정보 테이블의 정보를 사용하여 조건을 만족하는 특정 노드에 먼저 할당된 후에 이주할 노드를 탐색한다.

4.2 결과 분석

작업 균등화 정책의 성능 평가를 위해 본 논문에서 사용하는 척도는 slowdown[5]이며, 아래 식(3)으로 구한다.

$$\text{slowdown} = \frac{\text{작업의 총 수행시간}}{\text{작업의 CPU 수행시간}} \quad \cdots\cdots (3)$$

식(3)에서 작업의 총 수행시간은 CPU 수행시간과 큐에서의 대기시간, 이주시간, 페이지 폴트 처리시간의 합이다. slowdown은 각 Trace에 있는 모든 작업의 slowdown의 평균값이다. slowdown과 작업의 CPU 수행시간은 반비례하기 때문에 CPU 수행시간이 총 수행시간의 대부분을 차지한다면 작업을 수행하는 동안 작업 이주나 큐에서의 대기시간, 페이지 폴트 처리시간은 감소한다. 그러므로 slowdown이 작을수록 시스템의 작업처리율은 증가한다.

첫 번째 시뮬레이션은 32개 노드로 구성된 클러스터 컴퓨팅 시스템의 평균 처리속도와 평균 메모리 용량을 가지도록 노드 수를 균등하게 설정한 표 1의 시스템 5를 대상으로 slowdown을 측정한다. 측정 결과는 그림 3과 같다.

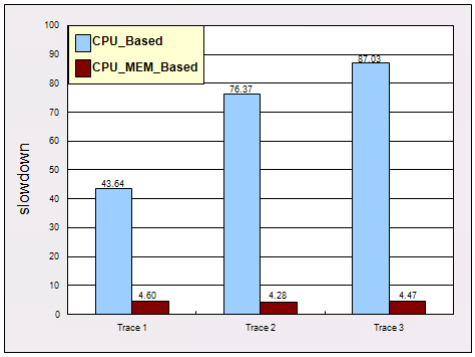

그림 3. 시스템 5의 시뮬레이션 결과
Fig. 3. Simulation result of system 5.

그림 3에서 CPU_Based는 기존의 CPU 기반작업부하 균등화 정책이고, CPU_MEM_Based는 본 논문에서 제안한 동적 작업부하 균등화 정책이다. 그림 3을 살펴보면 Trace 1, Trace 2, Trace 3에서 CPU_MEM_Based가 CPU_Based에 비해 우수함을 볼 수 있다. CPU_MEM_Based는 메모리 크기를 고려하여 페이지 폴트 발생을 줄이기 때문에 CPU_Based에 비해 작은 slowdown을 얻는다. 하지만 CPU_Based는 메모리 크기를 고려하지 않기 때문에 작업의 메모리 요구량이 증가하면서 페이지 폴트 발생이 증가한다. 따라서 페이지 폴트 처리시간이 증가하면서 slowdown도 함께 증가한다.

두 번째 시뮬레이션은 시스템 1~4를 대상으로 CPU_MEM_Based 정책을 적용하여 slowdown을 측정한다. 시뮬레이션 결과는 그림 4와 같다.

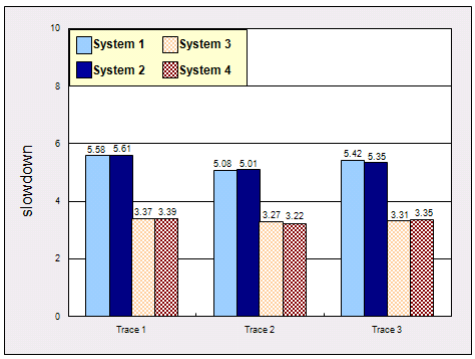

그림 4. 시스템 1~4의 시뮬레이션 결과
Fig. 4. Simulation result of system 1~4.

그림 4를 살펴보면 CPU 처리속도가 낮은 시스템 3과 시스템 4의 성능이 더 우수함을 볼 수 있다. 따라서 CPU_MEM_Based 정책은 CPU 처리속도가 낮은 노드들로 구성된 클러스터 컴퓨팅 시스템에서도 좋은 성능 향상을 얻을 수 있다.

세 번째 시뮬레이션은 시스템 1~4를 대상으로 CPU_MEM_Based 정책을 적용하여 메모리 한계점 60%, 80%, 100%에서 slowdown을 측정한다. 시뮬레이션 결과는 그림 5와 같다.

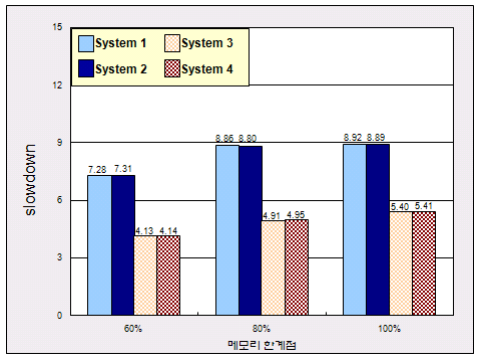

그림 5. 메모리 한계점에 따른 slowdown
Fig. 5. Memory Threshold vs. Slowdown.

그림 5를 살펴보면 메모리 한계점에 따른 slowdown의 변화는 크지 않음을 볼 수 있다. 메모리 한계점이 80%, 100%일 경우에는 그 변화의 폭이 적다. 시스템의 성능 향상을 위해서는 메모리 한계점을 60% 이하로 유지하는 것이 유리함을 알 수 있다.

표 3. 시스템 4의 노드 수
Table 3. Number of system 4 node.

노드 수	CPU 처리 속도 (MIPS)					메모리 크기 (MB)			
	100	200	300	400	500	32	64	128	256
6	2	1	1	1	1	2	2	1	1
18	6	6	2	2	2	7	7	2	2
24	9	9	2	2	2	9	9	3	3
32	13	13	2	2	2	12	12	4	4

네 번째 시뮬레이션은 표 1의 시스템 4를 대상으로 노드 수에 따른 slowdown의 변화를 측정한다. 표 3의 노드 수에

따른 시뮬레이션 결과는 그림 6과 같다.

그림 6을 살펴보면 Trace 1, 2, 3의 경우 모두 노드 수가 증가할수록 slowdown이 감소하는 것을 알 수 있다. 동일한 작업을 여러 노드에서 처리하면 시스템의 성능이 향상된다는 것을 알 수 있다. Trace 1의 경우 slowdown이 가장 큰 것은 Trace 2, 3에 비해 작업 수가 많고, 작업 이주시간과 대기시간이 크기 때문이다.

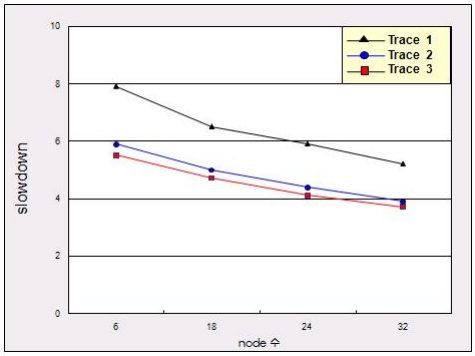

그림 6. 노드 수에 따른 slowdown
Fig. 6. The number of nodes vs. Slowdown.

V. 결론

본 논문에서는 이질적인 노드로 구성된 클러스터 컴퓨팅 시스템에서 CPU와 메모리 부하를 고려한 작업부하 균등화 정책을 제안하였다. 이 정책은 마스터 노드와 슬레이브 노드에서 각각 작업부하 균등화를 구현한다. 마스터 노드에서는 작업의 메모리 요구량(MRi)을 미리 예측할 수 없기 때문에 FSI_table에서 최대의 메모리 가용 공간을 가진 노드를 선택하여 작업을 할당한다. 슬레이브 노드에서는 수행중인 작업의 메모리 요구량이 증가하여 페이지 폴트가 발생하면 다른 슬레이브 노드로 작업을 이주시킴으로써 메모리 과부하에 따른 페이지 폴트 발생을 줄이고, 작업의 대기시간과 수행시간을 단축한다.

본 논문에서는 시뮬레이션을 통하여 제안한 작업부하 균등화 정책이 기존의 CPU 기반 정책에 비해 시스템의 성능 향상에 유리함을 검증하였다. 클러스터 컴퓨팅 시스템에서는 각 노드의 CPU만을 고려하는 것보다 메모리 크기까지 고려하여 작업을 할당한다면 메모리 과부하에 따른 페이지 폴트 발생을 줄이고, 시스템의 작업처리율을 향상시킬 수 있음을 알 수 있었다.

참고문헌

[1] Q. Zhang, A. riska, W. Sun, E. Smirni, and G. Ciardo, "Workload-Aware Load Balancing for Cluster Web Servers," IEEE Trans. on Parallel and Distributed Systems, Vol. 16, No. 3, pp. 219-232, March 2005.

[2] J. Guo and L. N. Bhuyan, "Load Balancing in a Cluster-Based Web Server for Multimedia Applications," IEEE Trans. on Parallel and Distributed Systems, Vol. 17, No. 11, pp. 1321-1334, Nov., 2006.

[3] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez, "Adaptive Parallel Job Scheduling with Flexible Coscheduling," IEEE Trans. on Parallel and Distributed Systems, Vol. 16, No. 11, pp. 1066-1077, Nov., 2005.

[4] Karatza H. and Hilzer R.C. "Epoch Load Sharing in a Network of Workstations," Proceedings of the 34th Annual Simulation Symposium, IEEE Computer Society Press, SCS, Seattle, Washington, pp. 36-42, Apr. 22-26, 2001.

[5] Wentong Cai, Francis Lee Bu-Sung, and Alfred Heng. "A Simulation Study of Dynamic Load Balancing for Network-based Parallel Processing," in Proceedings of 1997 International Symposium on Parallel Architectures Algorithms and Networks (I-SPAN'97), pp. 383-389, IEEE Computer Society Press, Taipei, 14-16 Dec. 1997.

[6] C. -C. Hui and S.T. Chanson, "Improved Strategies for Dynamic Load Sharing," IEEE Concurrency, Vol. 7, No. 3, pp. 58-67, 1999.

[7] M. Harchol-Balter and A. B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," ACM Trans. on Computer Systems, Vol. 15, No. 3, pp. 253-285, March, 1997.

[8] 이성훈, 한군희, "이질형 분산시스템에서의 동적 부하재분재를 위한 유전적 접근법," 한국컴퓨터정보학회 논문지, 제 11권, 제 1호, pp. 1-10, Mar. 2006.

〔9〕 A. Barak and A. Braverman, "Memory Ushering in a Scalable Computing Cluster," Journal of Microprocessors and Microsystems, Vol. 22, No. 3-4, pp. 175-182, Aug. 1998.
〔10〕 A. Acharya and S. Setia, "Availability and Utility of Idle Memory in Workstation Clusters", Proc. ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems, pp. 35-46, May, 1999.
〔11〕 L. Xiao, S. Chen, and X. Zhang, "Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands," IEEE Trans. on Parallel and Distributed Systems, Vol. 13, No. 3, pp. 223-240, March, 2002.

저 자 소 개

이 원 주

1989: 한양대학교 전자계산학과 공학사.
1991: 한양대학교 컴퓨터공학과 공학석사.
2004: 한양대학교 컴퓨터공학과 공학박사
현 재: 인하공업전문대학 컴퓨터정보과 부교수
관심분야: 성능분석, 센서네트워크,병렬처리시스템, 모바일 컴퓨팅, 그리드 컴퓨팅,

박 말 순

2001: 삼육대학교 컴퓨터과학과 공학사
2003: 한양대학교 컴퓨터공학과 공학석사
관심분야: 성능분석, 병렬처리시스템, 고성능컴퓨팅,