클러스터 컴퓨팅 시스템에서 CPU와 메모리 부하에 기반한 작업부하 균등화 정책

박말순°, 이원주, 전창호
(주)애버커스 기술연구소, 두원공과대학 인터넷프로그래밍과, 한양대학교 전자컴퓨터공학부
pms6028@iabacus.co.kr°, wonjoo@doowon.ac.kr , chjeon@cse.hanyang.ac.kr

Load Balancing Policy Based on CPU and Memory Workload in Cluster Computing System

Malsoon Park°, Wonjoo Lee, Changho Jeon
Abacus co., Ltd., Technology Research Institute°,
Dept. of Internet programming, Doowon Technical College,
School of Electrical and Computer, Hanyang University

요 약

본 논문에서는 이질적인 클러스터 컴퓨팅 시스템에서 CPU와 메모리 자원을 효율적으로 사용하는 작업부하 균등화 정책을 제안한다. 이 정책의 특징은 CPU 부하 상태와 수행중인 작업의 메모리 요구량을 고려하여 작업을 동적으로 할당하는 것이다. 먼저 각 노드는 CPU와 메모리 사용량에 따라 과부하 상태가 아니면 작업을 할당받아 수행한다. 그리고 수행중인 작업의 메모리 요구량이 가용 메모리 크기를 초과하여 페이지 폴트가 발생하면 수행 중인 작업을 다른 노드로 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 향상 면에서 유리함을 검증한다.

1. 서론

인터넷의 급속한 보급으로 웹 서버, 전자상거래 서버, 게임 서버 등의 여러 분야에서 빠르고, 안정적인 고성능 컴퓨터 시스템의 요구가 증가하고 있다. 그러나 고성능 컴퓨터 시스템은 각 회사의 고유한 구조에 기반하여 개발되므로 높은 가격으로 인해 활용 면에서 많은 어려움이 있다. 이러한 문제점을 해결하기 위한 하나의 방법이 클러스터 컴퓨팅 시스템을 구현하는 것이다. 클러스터 컴퓨팅 시스템은 저가의 PC와 워크스테이션을 고속 네트워크나 전용 네트워크로 연결한 시스템이다. 이 시스템의 장점은 적은 비용으로 고성능 컴퓨팅 시스템을 구현할 수 있기 때문에 가격 대 성능비가 높다는 것이다. 또한 사용자가 직접 상용 부품을 사용하여 업그레이드를 할 수 있고, 시스템 관리 및 유지 비용을 줄일 수 있다는 것이다. 그리고 사용이 편리한 PC 및 워크스테이션의 개발 환경을 그대로 사용할 수 있기 때문에 프로그램 개발이 용이하다는 것이다. 클러스터 컴퓨팅 시스템을 위한 기존의 작업부하 균등화 정책은 CPU나 메모리만을 기반으로 작업을 할당하기 때문에 빈번한 메모리 접근과 메모리 요구량이 많은 응용프로그램 처리에는 한계가 있다.

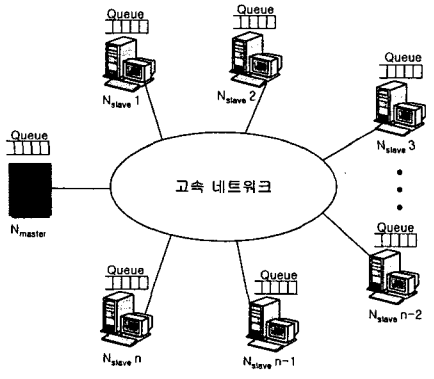
이러한 한계를 극복하기 위해 본 논문에서는 CPU와 메모리 부하를 기반으로 한 작업 균등화 정책을 제안한다. 이 정책에서 먼저 각 노드는 CPU와 메모리 사용량에 따라 과부하 상태가 아니면 작업을 할당 받아 수행한다. 그리고 수행중인 작업의 메모리 요구량이 가용 메모리 크기를 초과하여 페이지 폴트가 발생하면 수행 중인 작업을 다른 노드로 이주시킴으로써 작업의 대기시간과 수행시간을 단축한다.

본 논문의 2장에서는 기존의 작업부하 균등화 정책에 대하여 살펴본다. 3장에서는 본 논문에서 대상으로 하는 클러스터 컴퓨팅 시스템의 구성을 설명한다. 그리고 제안한 작업부하 균등화 정책에서 사용하는 용어를 정의하고, 작업부하 균등화 알고리즘에 대하여 자세히 설명한다. 4장에서는 시뮬레이션 환경과 시뮬레이션 결과를 분석한다. 그리고 5장에서 결론을 맺는다.

2. 관련 연구

기존의 작업부하 균등화 정책[1-5]은 CPU 기반 정책과 메모리 기반 정책으로 분류 할 수 있다. CPU 기반 정책[1-4]은 메모리 크기가 충분하다는 가정하에 CPU 처리속도나 CPU 작업 큐의 길이를 기준으로 작업을 할당한다. 이 정책은 작업 크기와 실행시간을 미리 예측해야 한다는 어려움이 있다. 메모리 기반 정책[5]은 CPU 처리속도에 여유가 있다는 가정하에 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. 이러한 문제점을 해결하기 위해 본 논문에서는 CPU 처리속도와 메모리 크기에 기반한 작업부하 균등화 정책을 제안한다.



<그림 1> 클러스터 컴퓨팅 시스템 구성

3. 제안하는 작업부하 균등화 정책

3.1 클러스터 컴퓨팅 시스템 구성

본 논문에서 제안하는 작업부하 균등화 정책은 <그림 1>과 같은 클러스터 컴퓨팅 시스템을 대상으로 한다. <그림 1>에서 N_{master}와 N_{slave}는 각각 마스터와 슬레이브 노드를 의미한다. 마스터 노드는 각 슬레이브 노드의 CPU 부하와 메모리 사용량에 대한 정보를 관리하며, 이 정보를 이용하여 작업을 슬레이브 노드에 할당한다. 그리고 각 슬레이브 노드는 할당 받은 작업을 수행한다.

3.2 용어 정의

제안한 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 CT_i와 MT_i로 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.

$$Free(C)_i = CT_i - Q_i \quad \text{..... (1)}$$

$$Free(M)_i = MT_i - ML_i \quad \text{..... (2)}$$

각 노드의 가용 공간 정보는 가용 공간 정보 테이블(Free Space Information Table)에 저장된다. 가용 공간 정보 테이블은 FSI_table로 표현한다. FSI_table의 각 레코드는 <i, $Free(C)_i$, $Free(M)_i$>로 구성되며 $Free(C)_i$와 $Free(M)_i$를 기준으로 내림차순으로 정렬된다. 여기서 i 는 노드 번호를 의미한다.

3.3 작업부하 균등화 알고리즘

작업부하 균등화 알고리즘은 <그림 2>와 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(MR_i)을 미리 예측할 수 없기 때문에 FSI_table에서 첫 번째 조건($Free(C)_i$>0 AND $Free(M)_i$>0)을 만족하는 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($ML_i + MR_i >= MT_i$)을 만족하면 새로운 작업이 슬레이브 노드에 할당되는 것을 막고, FSI_table의 정보를 수정한다. 그리고 작업을 이주할 best-fit 슬레이브 노드를 선택한다. 만약 best-fit 슬레이브 노드가 존재하면 현재 수행중인 작업을 이주시킨다. 그러나 best-fit 슬레이브 노드가 존재하지 않으면 메모리가 유휴 상태로 될 때까지 대기한다. 슬레이브 노드에서의 작업 균등화는 이미 예측된 메모리 요구량을 가지고 최적의 노드에 작업을 할당하기 때문에 반복되는 작업 이주를 줄일 수 있다. 또한 메모리 한계점을 초과하지 않기 때문에 페이지 폴트도 줄일 수 있다.

```
Master_LoadBalance() //Master 노드 작업 균등화
{
  while (N_master_Queue 길이 > 0)
  {
    if ( Free(C)_i > 0 AND Free(M)_i > 0 ) {
      Allocation_N_slave(J_m); // i-th 슬레이브 노드에 작업(J_m) 할당
      if (Free(M)_i > MR_i){  // 메모리 유휴 결정, MR_i: 메모리 요구량
        Execution( J_m);
        Receive(J_m+1);
      }else{
        Slave_LoadBalance(J_m) ; //In slave node, load balancing
      }
    } esle if ( Free(C)_i <= 0 OR Free(M)_i > 0 ) {
      Allocation_N_slave(J_m);
      Waiting(J_m);          // J_m 은 CPU가 유휴 상태로 될 때까지 대기
      if (Free(M)_i > MR_i ){  //MR_i: 메모리 요구량
          Executing( J_m);
          Receive(J_m+1);
      }else{
          Slave_LoadBalance(J_m) ; //In slave node,  load balancing
      }
    } else {
      Return N_master_Queue(J_m) ; // N_master_Queue로 J_m 복귀
    }
  }
}
Slave_LoadBalance(J_m) //Slave 노드 작업 균등화
{
  if ( ML_i + MR_i >= MT_i )
  {
      Block_New_Job();    //새로운 작업의 할당 저지
      Update_FSI_table() // FSI_table의 정보 수정
      Best-fit_N_slave = Select_Best-fit_Node(); //Best-fit 슬레이브 노드 선택
      If ( Best-fit_N_slave  = Ø){
        waiting();          //메모리가 유휴 상태로 될 때까지 대기
      }else{
        Job_Migration(J_m); // migrate J_m to Best-fit_N_slave
      }
  } else {
    Return N_master_Queue(J_m) ; // N_master_Queue로 J_m 복귀
  }
}
```

<그림 2> 작업부하 균등화 알고리즘

4. 시뮬레이션

4.1 시뮬레이션 환경

본 논문에서는 시뮬레이션을 통하여 제안한 작업부하 균등화 정책이 기존의 CPU 기반 정책에 비해 시스템의 성능 향상 면에서 유리함을 검증한다. 시뮬레이션 방법은 참고문헌[6]과 동일하다. 시뮬레이션을 위해 32개의 이질적인 노드로 클러스터 컴퓨팅 시스템을 구성한다. 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) 등과 같은 파라미터는 모두 고정된 값으로 설정한다. 작업은 작업발생율에 따라 일정한 시간 간격으로 발생하며 작업번호, 도착시간, 수행시간 등의 속성을 가진다. 슬레이브 노드에서 수행중인 작업의 메모리 요구량은 Pareto 분포[3, 6]를 따른다. 작업은 발생 빈도와 평균 메모리 요구량에 따라 Trace 1, Trace 2, Trace 3로 구분한다. Trace 1과 Trace 2, Trace 3 작업의 평균 메모리 요구량은 각각 1MB, 2MB, 4MB 이다.

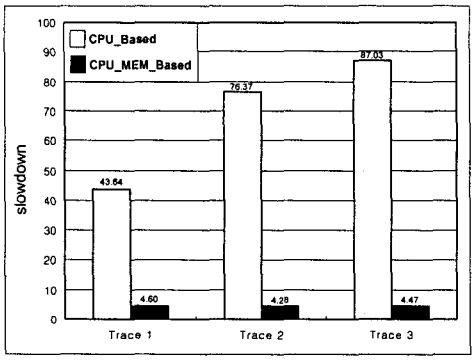
4.2 결과 분석

작업 균등화 정책의 성능 평가를 위해 본 논문에서 사용하는 척도는 slowdown이며, 아래 식(3)으로 구한다.

$$slowdown = \frac{\text{작업의 총 수행시간}}{\text{작업의CPU수행시간}} \quad \text{.....(3)}$$

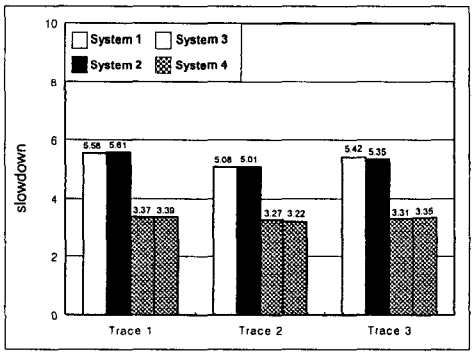
식(3)에서 작업의 총 수행시간은 CPU 수행시간과 큐에서의 대기시간, 이주시간, 페이지 폴트 처리시간의 합이다. slowdown은 각 Trace에 있는 모든 작업의 slowdown의 평균값이다. slowdown과 작업의 CPU 수행시간은 반비례하기 때문에 CPU 수행시간이 총 수행시간의 대부분을 차지한다면 작업을 수행하는 동안 작업 이주나 큐에서의 대기시간, 페이지 폴트 처리시간은 감소한다. 그러므로 slowdown이 작을수록 시스템의 작업처리율은 증가한다.

첫 번째 시뮬레이션은 CPU 처리속도와 메모리 용량이 서로 다른 32개 노드의 평균 처리속도와 메모리 용량을 가지도록 노드 수를 균등하게 설정한 클러스터 컴퓨팅 시스템에서 CPU_Based와 CPU_MEM_Based 정책을 적용하여 slowdown을 측정한다. CPU 처리속도가 100, 200, 400, 500 MIPS인 노드를 각각 6개로 설정하고, 300 MIPS는 CPU의 평균처리속도에 해당하기 때문에 8개로 설정한다. 또한, 메모리 크기가 32, 64, 128, 256 MB인 노드를 각각 8개로 설정한다. 시뮬레이션 결과는 <그림 3>과 같다.



<그림 3> CPU 처리속도와 메모리 크기가 다른 노드의 수가 균등한 클러스터 컴퓨팅 시스템

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



<그림 4> 클러스터 컴퓨팅 시스템의 종류에 따른 성능 비교

두 번째 시뮬레이션은 CPU 처리속도와 메모리 용량이 서로 다른 32개의 노드로 4 종류의 클러스터 컴퓨팅 시스템을 구성하고, 본 논문에서 제안하는 CPU_MEM_Based 정책을 적용하여 slowdown을 측정한다. System 1은 CPU 처리속도가 높고, 메모리 용량이 큰 노드들이 많은 시스템이다(CPU>300 MIPS: 26개, MEM>128 MB: 24개). System 2는 CPU 처리속도가 높고, 메모리 용량이 작은 노드들이 많은 시스템이다(CPU>300 MIPS: 26개, MEM<128 MB: 24개). System 3는 CPU 처리속도가 낮고, 메모리 용량이 큰 노드들이 많은 시스템이다(CPU<300 MIPS: 26개, MEM>128 MB: 24개). System 4는 CPU 처리속도가 낮고, 메모리 용량이 작은 노드들이 많은 시스템이다(CPU<300 MIPS: 26개, MEM<128 MB: 24개). 시뮬레이션 결과는 <그림 4>와 같다. <그림 4>를 살펴보면 CPU 처리속도가 낮은 System 3과 System 4의 성능이 더 우수함을 볼 수 있다. 따라서 본 논문에서 제안하는 CPU_MEM_Based 정책은 CPU 처리 속도가 낮은 노드들로 구성된 컴퓨팅 시스템에서도 높은 CPU 이용율을 보인다.

5. 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(MR_i)을 미리 예측할 수 없기 때문에 FSI_table에서 최대의 메모리 가용공간을 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.

6. 참고문헌

1) 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, April 22-26, 2001.
2) 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.
3) C. -C. Hui and S.T. Chanson, "Improved Strategies for Dynamic Load Sharing", IEEE Concurrency, Vol. 7, No. 3, pp. 58-67, 1999.
4) M. Harchol-Balter and A. B Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing", ACM Trans. Computer Systems, vol. 15, no. 3, pp. 253-285, 1997.
5) 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.
6) L. Xiao, S. Chen, and X. Zhang, "Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands", IEEE Trans. Parallel and Distributed System, Vol. 13, No. 3, pp. 223-240, March 2002.
7) 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.