상대적 부하 색인을 기반으로 한 이기종 워크스테이션 클러스터의 부하 균형

(Load Balancing of Heterogeneous Workstation Cluster based on Relative Load Index)

지 병 준 † 이 광 모 ††
(ByoungJun Ji) (KwangMo Lee)

요 약 이기종 워크스테이션 클러스터링은 응용 프로그램의 병렬 처리에 유용하며 비용 측면에서 효과적이다. 이기종 워크스테이션 클러스터링 환경에서 총작업반환시간을 최소화하기 위해서는 부하 균형 시스템이 필요하다.

기존의 부하 균형 방식은 각 워크스테이션의 처리능력에 가중치를 미리 부여하여 작업을 분배하는 정적 방식이거나, 각 워크스테이션의 상대적 처리능력을 얻기 위해서 성능 테스트 프로그램을 수행하는 동적 방식이 있다. 수행되는 응용 프로그램과는 관계없는 성능 테스트 프로그램은 계산시간을 소비하고 총작업반환시간을 지연시킨다.

이 논문은 상대적 부하 색인에 기초한 효과적 작업 분배 방식과 작업 이주 방식을 제안하였으며 이기종 워크스테이션 클러스터 환경에서 부하 균형 시스템을 설계 구현하였다. 이 논문에서 제안한 방식의 총작업반환시간을 실험을 통하여 부하 균형을 하지 않은 라운드 로빈 방식의 총작업반환시간과 성능 테스트 프로그램에 의한 부하 균형 방식의 총작업반환시간과 비교하였다. 실험 결과는 비교한 방식보다 제안 방식의 결과가 우수함을 보였다.

키워드 : 부하균형, 부하색인, 워크스테이션 클러스터

Abstract The clustering environment with heterogeneous workstations provides the cost effectiveness and usability for executing applications in parallel. Load balancing is considered a necessary feature for a cluster of heterogeneous workstations to minimize the turnaround time.

Previously, static load balancing that assigns a predetermined weight for the processing capability of each workstation, or dynamic approaches which execute a benchmark program to get relative processing capability of each workstation were proposed. The execution of the benchmark program, which has nothing to do with the application being executed, consumes the computation time and the overall turnaround time is delayed.

In this paper, we present efficient methods for task distribution and task migration, based on the relative load index. We designed and implemented a load balancing system for the clustering environment with heterogeneous workstations. Turnaround times of our methods and the round-robin approach, as well as the load balancing method using a benchmark program, were compared. The experimental results show that our methods outperform all the other methods that we compared.

Key words : Load Balance, Load Index, Workstation Cluster

1. 서 론

병렬 응용 작업을 워크스테이션 클러스터에서 처리하는 목적은 총작업 반환시간을 최소화 하기 위한 것이다. 클러스터를 구성하는 워크스테이션들이 동일 기종이나 다른 기종이라도 총작업 반환시간을 최소하기 위해서는 부하균형이 필요하다. 부하 균형을 결정하는 요인으로 현재의 부하 정도를 나타내는 부하색인을 사용한다.

† 비 회 원 : 한림정보산업대학 전산정보처리과 교수
bjji@sun.hallym-c.ac.kr
†† 종신회원 : 한림대학교 컴퓨터공학과 교수
kmlee@sun.hallym.ac.kr
논문접수 : 2001년 11월 1일
심사완료 : 2002년 1월 4일

동기종 워크스테이션 클러스터 시스템의 각 워크스테이션들의 처리기는 같은 처리성능을 가지므로 현재의 부하 색인을 부하 균형에 바로 사용할 수 있다. 그러나 이기종 워크스테이션 클러스터 시스템에서 부하 색인은 바로 부하 균형에 사용 할 수 없다. 이기종 워크스테이션 클러스터에서 부하 색인은 다른 워크스테이션과 상대적으로 비교된 객관적 비교값을 사용한다.

이기종 워크스테이션 부하 균형에 관한 기존 연구에서 각 워크스테이션의 상대적 처리 능력을 부하 색인으로 적용하기 위한 방법은 두 가지로 구분된다. 첫 번째는 이기종 워크스테이션 클러스터 환경에서 부하색인 정도를 시스템 설정시 미리 가중치로 결정하는 정적 방식이다[1]. 두 번째는 워크스테이션 클러스터에서 처리하려는 병렬 응용 프로그램과 상관없는 간단한 성능 테스트 프로그램(Banchmark)을 반복적이거나 필요시에 수행하는 동적 방식이 있다[2,3,4,5]. 특히, 이 방식은 성능 테스트 프로그램 의존적 부하 균형 방식이며 전체 반환시간을 지연하는 하나의 요소가 된다.

이 논문에서는 부하 균형을 위한 부하 색인 결정을 동적으로 하면서 성능 테스트 프로그램 독립적 부하 균형 방식을 수행하기 위해서 자체 응용 프로그램을 수행하면서 발생하는 여러 요소들을 상대적으로 비교하고 그 값을 상대적 부하 색인으로 사용하는 부하 균형 방식을 제안한다. 상대적 부하 색인에 기초한 부하 균형 방식에는 부하를 고려한 균등한 작업 분배(Distribute) 방식과 과부하 워크스테이션과 경량부하 워크스테이션 간의 작업이주(Migration) 방식을 포함한다.

2. 관련 연구

2.1 부하 색인

일반적으로 워크스테이션의 부하정도를 나타내는 부하정보는 0이상의 양수값으로 표현하는 부하 색인(load index)을 사용한다. 부하색인은 시스템 성능 표현과 직접적인 관계를 가지며 잘 표현되는 항목을 선정해야 한다. 부하 색인은 최신의 부하를 표현하기 위해서 변경되어야 하며, 변경 간격이 너무 자주이면 부하 색인 수정의 효과를 감소시키고 너무 오랜 변경간격은 시스템의 현재부하를 못 나타내므로 부하색인의 변경 기간은 중요한 고려 사항중의 하나이다.

부하 색인은 CPU의 준비 큐 길이, 일정 시간의 평균 준비 큐의 길이, 워크스테이션의 현재 또는 일정간격의 CPU 이용률, 작업의 정규화된 응답시간, 아직까지 남아있는 총 작업 처리시간 및 작업의 수, 현재 수행중인 작업들의 총 처리 시간등을 이기종 워크스테이션 클러스터에서 부하 균형을 위한 부하색인으로 바로 사용 할 수 있다. 일반적으로 워크스테이션에서 부하 정보는 uptime, vmstat, finger, top, utmp등의 unix 유틸리티를 통해서 구할 수 있다. 준비 큐의 평균 길이 즉, 평균 작업의 수인 평균 부하(load average)는 동기종 워크스테이션 클러스터에서 부하균형을 위한 가장 적절한 부하색인이다[6].

이기종 워크스테이션 클러스터에서는 자체 워크스테이션의 부하 색인들이 다른 워크스테이션과 비교된 부하 정보를 표현하지 못한다. 그러므로 상대적으로 비교된 부하 색인이 필요하다. 그러므로 기존 연구에는 상대적으로 비교된 부하 색인을 유도하기 위해서 클러스터 구성시 가중치로 부하 색인을 결정하는 정적방식과 주기적으로 성능 테스트 프로그램의 수행 결과를 부하 색인으로 적용하는 동적 방식이 있다.

2.2 부하 균형

부하 균형의 목적은 총작업반환시간을 줄이기 위해서 '어떻게 부분 작업들을 워크스테이션 클러스터에 분배하는가'이다. 부하 균형을 위한 연구는 공유 메모리 다중처리 시스템 접근과 분산 시스템 접근으로 구분 할 수 있다. 이것은 다시 워크스테이션 종류에 의해서 동기종 환경과 이기종 환경으로 구분하고 부하 균형 관리자의 분배 결정의 방법에 따라서 정적 방식과 동적 방식으로 구분된다. 이 논문은 이기종 분산 시스템에서 동적 접근 방식에 관한 연구이며, 특히 '이기종 워크스테이션 클러스터 환경에서 동적으로 워크스테이션과 작업들의 분배와 재분배를 어떻게 하는가'의 문제이다. 즉 '여러 작업과 여러 워크스테이션들이 많은 변화가 계속해서 발생되는 상황에서 가장 좋은 분배와 재분배 선택은 어떤 것인가'이다. 그러나 2개 이상의 프로세서 클러스터에서 최소 실행 비용을 산출하기 위한 문제는 NP-complete 이다[7].

동기종의 정적 부하 균형 방식을 다룬 [8]에서는 모든 워크스테이션에 남아있는 부하는 대략 같아야 하며 통신비용을 줄이기 위해서 선형 프로그램이나 유전자 알고리즘을 사용하였다. 동기종의 동적 부하 균형 방식을 다룬 [9]에서는 주기적으로 부하의 변화를 확인하여 기준값과 비교하고 자체 수행과 분배를 결정한다. 또한 [10]에서는 시스템을 작은 도메인 단위로 나누고 부하의 교환은 같은 도메인내의 워크스테이션들간에만 가능하도록 함으로서 [9]의 방식을 향상시켰다.

이기종 정적 부하 균형 방식은 작업 제출시에 워크스테이션에 작업을 분배하는 방식으로 동적으로 작업 이주를 지원하지 못하는 방식으로 Load sharing

Facility(LSF), Utopia, Distributed Quening System (DQS), 그리고 Prospero Resource Manager(PRM)등이 있다. 현재 많은 연구가 되고 있는 이기종 동적 부하 균형 방식은 여러 시스템들이 있다. 그러나, 매우 확장성이 뛰어난 PVM은 동적 분배가 적용되어 있지 않으며, Condor은 단일 작업만을 분배하도록 구성되어 있으며, MPVM[11], MIST[12]와 DPVM[13]은 비동기적으로 같은 작업의 이주만을 지원한다. 이 논문은 다중 사용자 이기종 워크스테이션 클러스터 시스템에서 동적으로 정수의 분배 작업 유도 알고리즘과 정수의 이주 작업 유도 알고리즘을 제안한다.

3. 균형적 작업 분배

3.1 작업 분배시 상대적 부하색인

워크스테이션 클러스터 시스템 Φ_{system}에서 총작업반환 시간 T은 워크스테이션 p_i에 할당된 병렬 작업의 반환 시간 t_i들 중에서 가장 큰 값이며 다음과 같다.

$$T = \underset{\forall_i \in \Phi_{system}}{Max} \{t_i\}$$

작업 분배 알고리즘은 균형적 작업 분배를 통하여 최소의 T을 얻고자 한다.

X는 Φ_{system}에서 분산 처리할 병렬 처리 프로그램이며 SPMD(single-program multiple-data)의 동일 실행 코드의 병렬처리 작업들의 개수이다. Φ_{system}내의 각 워크스테이션 p_i에 분배되는 총 작업 개수는 상대적 부하 색인을 평가하기 위해서 사용하는 $Seedx_i$와 분배 알고리즘에 의해 분배될 x_i의 합이다. Φ_{system}에서 분배 알고리즘에 의해서 분배될 작업의 개수 $DisX$는 X에서 상대적 부하 색인을 유도하기 위해서 사용한 $Seedx_i$을 제외한 값이다.

$$X = (x_1 + seedx_1) + \ldots + (x_i + seedx_i) + \ldots + (x_n + seedx_n), x_i > seedx_i$$

$$DisX = X - \sum_{\forall_i \in \Phi_{system}} seedx_i$$

이기종 워크스테이션 클러스터 시스템에서 병렬 응용 작업의 일부이면서 상대적 부하 색인 평가를 위한 n개의 $Seedx_i$작업이 워크스테이션 p_i에 분배하는 t시간의 워크스테이션 부하색인 $La_i^{(t)}$는 분배된 병렬 작업에 의한 부하 $LocalLa_i^{(t)}$뿐만 아니라 p_i 자체에서 발생되는 외부작업의 부하 $ExLa_i^{(t)}$을 포함한다. 또한 분배된 작업을 수행하고 난, $t+1$ 시간이후의 워크스테이션 부하색인 $La_i^{(t+1)}$는 분배작업에 의한 $LocalLa_i^{(t+1)}$과 $t+1$시간 현재, 외부작업 부하 $ExLa_i^{(t+1)}$을 포함한다. t 시간 동안의 부하색인의 변화 $\Delta La_i^{(t+1)}$은 다음과 같다.

$$La_i^{(t+1)} = LocalLa_i^{(t+1)} + ExLa_i^{(t+1)}$$
$$La_i^{(t)} = LocalLa_i^{(t)} + ExLa_i^{(t)}$$
$$\Delta La_i^{(t+1)} = La_i^{(t+1)} - La_i^{(t)}, \forall_i \in \Phi_{system}$$

여기서 $\Delta La_i^{(t+1)}$의 값이 양의 값을 가진다면 p_i는 그만큼의 부하가 증가함을 보인다. $\Delta La_i^{(t+1)}$의 값이 0이라면 현재의 상태를 유지하고 있는 것이고 음의 값을 가지면 워크스테이션의 현재 부하는 감소함을 보인다. $\Delta La_i^{(t+1)}$양의 값이 가장 크다는 것은 Φ_{system}에서 현재 부가된 작업을 처리하는데 가장 많은 부하가 생성되었음을 나타낸다. 즉 현재 가장 처리 능력이 떨어지는 워크스테이션으로 간주할 수 있다.

Φ_{system}의 각 워크스테이션들은 $\Delta La_i^{(t+1)}$을 가지며, $\Delta La_i^{(t+1)}$값에서 워크스테이션 p_i의 상대적 부하 색인 rli_i는 다음과 같다. 가장 처리 성능이 떨어지는 p_i의 rli_i는 1이 되고 나머지 워크스테이션들이 1보다 큰 값을 갖게 된다.

$$\Delta La_{MAX} = MAX[\Delta La_{i-1}^{(t+1)}, \Delta La_i^{(t+1)}, \Delta La_{i+1}^{(t+1)}, \ldots, \Delta La_n^{(t+1)}]$$
$$\Delta La_{MIN} = MIN[\Delta La_{i-1}^{(t+1)}, \Delta La_i^{(t+1)}, \Delta La_{i+1}^{(t+1)}, \ldots, \Delta La_n^{(t+1)}]$$

라면, 각 워크스테이션들의 $\Delta La_i^{(t+1)}$값이 모두 양의 값을 가지는 경우에는

$$rli_i = \frac{\Delta La_{MAX}}{\Delta La_i^{(t+1)}}, \forall_i \in \Phi_{system}$$

이고, 각 워크스테이션의 $\Delta La_i^{(t+1)}$의 값 중에서 음의 값을 가지는 경우에는 다음과 같이 처리하여 현재 처리 성능이 가장 떨어지는 워크스테이션의 rli_i를 1로 한다.

$$rli_i = \frac{\Delta La_{MAX} + |\Delta La_{MIN}| + 1}{\Delta La_i^{(t+1)} + |\Delta La_{MIN}| + 1}, \forall_i \in \Phi_{system}$$

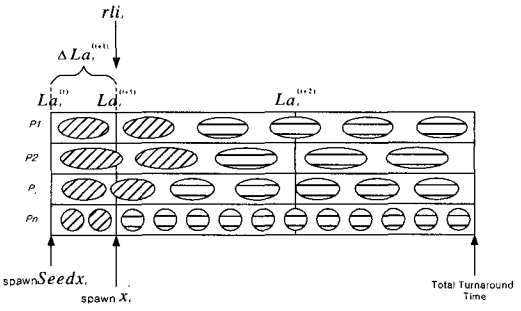

그림 1 상대적 부하색인과 작업분배

3.2 작업 분배 알고리즘

상대적 부하색인 유도를 위한 각 $Seedx_i$의 합을 제외한 $DisX$는 $DisX \geq \Phi_{system}$라면 분배 알고리즘의 대상이다. 워크스테이션 p_i에 상대적 부하색인에 의한 분배작업의 수 x_i는 다음과 같다.

$$x_i = \alpha \cdot rli_i \quad , \forall_i \in \Phi_{system} \qquad (A)$$

α는 다음과정에서 얻을 수 있다. Φ_{system}의 각 워크스테이션 p_i는 병렬 수행할 전체 작업에서

$$\frac{rli_i}{\sum_{\forall_i \in \Phi_{system}} rli_i}$$

만큼의 비율로 작업을 할당받는다. 즉, p_i에 할당되는 작업의 수 x_i는 다음과 같다.

$$x_i = DisX\left[\frac{rli_i}{\sum_{\forall_i \in \Phi_{system}} rli_i}\right] \qquad (B)$$

(A)와 (B)식에서 α는 다음과 같으며 Φ_{system}에서 얻어 질 수 있는 단위 작업의 최소 반환 시간으로 볼 수 있다.

$$\alpha = \frac{DisX}{\sum_{\forall_i \in \Phi_{system}} rli_i}$$

x_i와 α는 이상적인 값이므로 실제 구현에서 분배할 정수개의 작업 x_i'값을 유도하는 과정이 필요하다[3]. 우선 각각의 워크스테이션 p_i에 분배할 최적의 작업 개수인 x_i의 하한 정수 값을 취한다.

$$x_i' = \lfloor x_i \rfloor \quad , \forall_i \in \Phi_{system}$$

다음은 각 워크스테이션 p_i에게 정수개의 분배 작업 x_i'을 부여하고도 아직 남아있는 작업들을 구한다. x_i에서 하한 정수 값 x_i'을 제외한 소수들의 합 Y는 아직 분배가 결정되지 않은 작업의 수이다. Y는 병렬 분배처리 해야할 총 작업 $DisX$에서 각 워크스테이션들에 분배 결정된 Φ_{system}내의 정수 작업 x_i'의 합을 뺀 값이다.

```
load_distributed( )
{
    for(i=0; i<# of Φ_system; i++)
            check La_i^(t);          /*초기 부하 색인 체크
    for(i=0; i<# of Φ_system; i++)
            Spawn( Seedx_i → p_i );      /*자체 작업의 일부 분배
    for(i=0; i<# of Φ_system; i++)
            check La_i^(t+1);        /*부하 색인 체크
    ΔLa_i^(t+1) = La_i^(t+1) − La_i^(t) ;          /*부하 색인 변화률 계산
    caculate ΔLa_MAX, ΔLa_MIN;
    for(i=0; i<# of Φ_system; i++)
            if( La_i^(t+1) >= 0 )
                    rli_i = ΔLa_MAX / ΔLa_i^(t+1)   ; /*상대적 부하 색인 계산
                else
                    rli_i = (ΔLa_MAX + |ΔLa_MIN| + 1) / (ΔLa_i^(t+1) + |ΔLa_MIN| + 1)   ; /*상대적 부하색인 계산
    α = DisX / Σ_{∀i∈Φ_system} rli_i ;/*단위 작업 최소 반환시간 계산
    for(i=0; i<# of Φ_system; i++)
            x_i = α · rli_i ;                /*분배할 작업 계산
            x_i' = ⌊ x_i ⌋ ;                        /*분배할 정수 작업 유도
    Y = DisX − Σ_{∀i∈Φ_system} x_i';
    for(i=0; i<# of Φ_system; i++)
            Plusa_i = (⌈ x_i ⌉ − x_i) / rli_i ;      /*추가 작업 분배을 위한 워크스테이션 선정
    for(i=0; i<# of Φ_system; i++)
            if( Plusa_i smallist in Φ_system)
                    x_i'++;                          /*추가 작업에 따른 반환시간 변화가 가정 적은
    for(i=0; i<# of Φ_system; i++)                       워크스테이션에 작업추가
            Spawn( x_i' → p_i);                      /*각 워크스테이션에 작업 분배
}
```

그림 2 상대적 부하 색인과 작업 분배 알고리즘

$$Y = \sum_{\forall i \in \Phi_{system}} x_i - \sum_{\forall i \in \Phi_{system}} x_i' = DisX - \sum_{\forall i \in \Phi_{system}} x_i'$$

다음은 Y개의 추가 작업을 분배하기 위한 대상 워크스테이션을 선정한다. 먼저, x_i가 분배된 워크스테이션 p_i의 단위 작업의 최대 반환시간은 $\frac{\lceil x_i \rceil}{rli_i}$이고 단위 작업 반환시간은 $\frac{x_i}{rli_i}$이다. 이 들의 차이를 $Plusa_i$로 했을 때, $Plusa_i$가 작은 워크스테이션은 추가 작업분배에 따른 반환시간의 증가 변화가 적다고 볼 수 있다.

$$Plusa_i = \frac{(\lceil x_i \rceil - x_i)}{rli_i} \quad , \forall i \in \Phi_{system}$$

그러므로 Φ_{system}내의 워크스테이션들 중에 $Plusa_i$가 작은 것에서부터 Y개의 작업을 하나씩 워크스테이션 p_i에 분배한다.

상대적 부하 색인에 기초한 작업 분배 알고리즘은 그림 2와 같다.

4. 부하 균형 작업 이주

4.1 작업 이주시 상대적 부하색인

이기종 클러스터의 현재 상황을 가장 적절히 파악하고 균형적인 작업 분배를 하였더라도 불규칙한 외부 작업 유입으로 현재 분배된 작업의 부하가 부담스럽거나 아니면 더 많은 작업의 분배도 가능할 수 있다. 워크스테이션 클러스터 시스템에서 동적으로 각 워크스테이션의 부하 변화를 파악하고 현재에서 가장 최선의 부하균형을 이루기 위해서는 과부하의 노드에서 경량부하의 노드로 작업을 재분배하는 작업이주 방법이 필요하며, 이 때 필요한 세 가지 상대적 부하색인 방법을 제안한다.

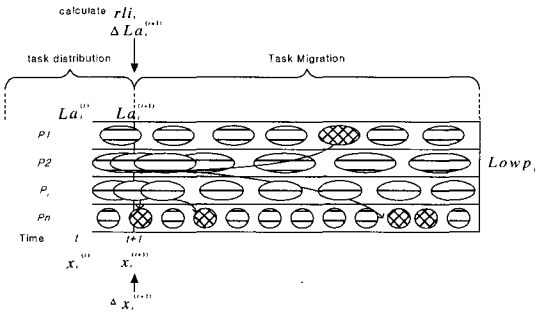

그림 3 상대적 부하색인과 작업이주

첫 번째 상대적 부하색인으로는 처음 작업 분배시에 사용하였던 상대적 부하색인 방식을 이용한다. 즉, $\Delta La_i^{(t+1)}$을 이용하여 Φ_{system}의 각 워크스테이션 p_i의 rli_i 값을 구한다.

두 번째 상대적 부하 색인으로는 처리된 작업량의 상대 비교를 사용할 수 있다. 초기 균형 분배 알고리즘에 의해 워크스테이션 p_i에 분배된 $x_i'^{(t)}$와 $t+1$ 시간이후, 일부의 작업이 수행된 $x_i'^{(t+1)}$에서 처리된 작업을 나타내는 $\Delta x_i'^{(t+1)}$와 부하 색인으로 적용되는 상대적 부하색인 rli_i는 다음과 같다.

$$\Delta x_i'^{(t+1)} = x_i'^{(t)} - x_i'^{(t+1)} \quad , x_i'^{(t)} \geq x_i'^{(t+1)} \quad , \forall i \in \Phi_{system}$$

$$\Delta x'_{MIN} = MIN[\Delta x_{i-1}'^{(t+1)}, \Delta x_i'^{(t+1)}, \Delta x_{i+1}'^{(t+1)}, \ldots, \Delta x_n'^{(t+1)}], \text{ 이면}$$

$$rli_i = \frac{\Delta x_i'^{(t+1)}}{\Delta x'_{MIN}} \quad , \forall i \in \Phi_{system}$$

이다.

세 번째로 제안하는 상대적 부하 색인은 위의 두가지 방식을 복합하여 처리한다. 먼저, 워크스테이션 클러스터에서 가장 처리성능이 떨어지는 워크스테이션 $Lowp_i$을 찾는다.

$Lowp_i$을 찾기 위해서 3단계를 거친다. 첫 번째 단계는 $t+1$ 시간이후, 처리된 작업 $\Delta x_i'^{(t+1)}$들 중에서 최소 값을 갖는 워크스테이션 그룹 $Lowp_i''$을 구한다. 두 번째 단계는 $Lowp_i''$들 중에서 t시간 이전의 부하 $La_i^{(t)}$가 최대인 워크스테이션 그룹 $Lowp_i'$을 구한다. 세 번째 단계에서 $Lowp_i'$중에서 $t+1$시간 현재의 $La_i^{(t+1)}$가 최대인 워크스테이션 p_i를 $Lowp_i$로 한다.

단계 1 : $Lowp_i'' = \{i \mid \forall i \in \Phi_{system} \wedge MIN\Delta x'\}$

단계 2 :

$Lowp_i' = \{i \mid \forall i \in Lowp_i'' \wedge MAX[La_i^{(t)}, \forall i \in Lowp_i]\}$

단계 3 :

$Lowp_i = \{i \mid \forall i \in Lowp_i' \wedge MAX[La_i^{(t+1)}, \forall i \in Lowp_i]\}$

다음은 워크스테이션 $Lowp_i$가 다른 워크스테이션 p_i에 분배된 작업 x_i'을 대신해서 처리했을 경우, $t+1$시간 현재의 부하 $La_i^{(t+1)}$의 예상값 $LowLa_i^{(t+1)}$을 구한다.

$$LowLa_i^{(t+1)} = \frac{La_{Lowp_i}^{(t+1)} \cdot (x_i' + 1)}{x'_{Lowp_i} + 1} \quad , \forall i \in \Phi_{system}$$

워크스테이션 p_i에서 $t+1$시간 현재 발생한 실제 부하색인 $La_i^{(t+1)}$는 $Lowp_i$워크스테이션의 $LowLa_i^{(t+1)}$와 $DiffLa_i^{(t+1)}$만큼의 차이를 보인다.

$$DiffLa_i^{(t+1)} = LowLa_i^{(t+1)} - La_i^{(t+1)} \quad , \forall i \in \Phi_{system}$$

$DiffLa_i^{(t+1)}$가 양이면 그 차이만큼 $Lowp_i$에 비하여

```
load_migration( )
{
    load_distributed( La_i^(t), x_i' );      /*분배 알고리즘
    for(i=0; i<# of Φ_system; i++)
        check La_i^(t+1);   /*부하 색인 체크
        check x_i';         /*수행전 남아 있는 작업 체크
    calculate ΔLa_i^(t+1), Δx_i'^(t+1);      /*부하 색인 변화율, 처리 작업율 계산
    for(i=0; i<# of Φ_system; I++)           /*현재 상황에서 가장 저성능 워크스테이션 선택
        if( ΔLa_i^(t+1) smallist in Φ_system)
            Lowp_i'' = p_i;
            for(i=0; i<# of Lowp_i''; I++)
                if( La_i^(t) largest in Lowp_i'')
                    Lowp_i' = p_i;
                    for(i=0; i<# of Lowp_i'; i++)
                        if( La_i^(t+1) largest in Lowp_i')
                            Lowp_i = p_i;
    for(i=0; i<# of Φ_system; i++)
        LowLa_i^(t+1) = La_{Lowp_i}^(t+1) · (χ_i' + 1) / (χ'_{Lowp_i} + 1)   ; /* Lowp_i가 각 워크스테이션을 대신해서
    for(i=0; i<# of Φ_system; I++)                                  처리했을 경우 부하색인*/
    DiffLa_i^(t+1) = LowLa_i^(t+1) − La_i^(t+1) ; /*최대 부하색인과 비교
    calculate DiffLa_{i MIN}^(t+1);
    for(i=0; i<# of Φ_system; i++)
        rli_i = DiffLa_i^(t+1) / DiffLa_{i MIN}^(t+1) ;          /*상대적 부하색인 계산
    α = (Σ_{∀i∈Φ_system} x_i'^(t+1)) / (Σ_{∀i∈Φ_system} rli_i);          /*평균 부하 계산
    for(i=0; i<# of Φ_system; i++)
        migx_i = x_i'^(t+1) − rli_i · α ; /*이주 작업 계산
    for(i=0; i<# of Φ_system; i++)                  /*이주 정수 작업 유도
        if( migx_i > 0)
            migx_i' = ⌊ migx_i ⌋ ;
            if( migx_i < 0)
            migx_i' = ⌈ migx_i ⌉ ;
    migy' = Σ_{∀i∈Φ_system} migx_i';
    if( migy' > 0)
        for(i=0; i<# of Φ_system; i++)
            if( migx_i' smallist in migx_i'< 0)
                migx_i'--;
    if( migy' < 0)
        for(i=0; i<# of Φ_system; i++)
            if( migx_i' largest in migx_i'>0)
                migx_i'++;
    for(i=0; i<# of Φ_system; i++)
        Spawn( migx_i' → p_i);                  /*작업 이주
}
```

그림 4 상대적 부하 색인과 작업 이주 알고리즘

부하 부담이 적었음을 나타낸다. 여기서 상대적 부하 색인 rli_i는 다음과 같다.

$$DiffLa_i^{(t+1)}{}_{MIN} = MIN[DiffLa_{i-1}^{(t+1)}, DiffLa_i^{(t+1)}, DiffLa_{i+1}^{(t+1)}, \ldots, DiffLa_n^{(t+1)}]$$

이면

$$rli_i = \frac{DiffLa_i^{(t+1)}}{DiffLa_i^{(t+1)}{}_{MIN}}, \forall_i \in \Phi_{system}$$

이다.

4.1.1 작업 이주 알고리즘

rli_i는 워크스테이션 p_i의 동적 부하 균형을 위한 작업 이주의 부하색인으로 사용한다. 또한 각 워크스테이션에 분배된 작업 x_i'에서 $t+1$시간 이후에 처리되고 남은 작업 $\Delta x_i'^{(t+1)}$는 현재까지 아직 처리되지 않으면서 p_i에 영향을 미치는 부하이고 이주의 대상이다.

단위 작업의 최소 반환시간 α는 Φ_{system}내에서 평균 작업 부하로 볼 수 있다. 즉 α는 클러스터 워크스테이션들의 상대적 부하색인 rli_i의 합과 현재 작업 부하 $x_i'^{(t+1)}$의 합과의 비율이다.

$$\alpha = \frac{\sum_{\forall_i \in \Phi_{system}} x_i'^{(t+1)}}{\sum_{\forall_i \in \Phi_{system}} rli_i}$$

워크스테이션 p_i의 최적의 작업 부하상태 $x_i'^{(t+1)}{}_{opt}$는 평균 작업 부하와 상대적 부하색인으로 얻어진다.

$$x_i'^{(t+1)}{}_{opt} = rpc_i \cdot \alpha \quad , \forall_i \in \Phi_{system}$$

$t+1$시간 현재 p_i의 작업 부하 $x_i'^{(t+1)}$는 $x_i'^{(t+1)}{}_{opt}$에 가까울 때 최적의 상태이다. 그러나 p_i의 현재 작업 부하 상태는 $x_i'^{(t+1)}{}_{opt}$상태가 아니며 $x_i'^{(t+1)}$와 차이를 보인다. 그 차이만큼이 현재 부하 정도를 나타내며 이주할 작업 부하 $migx_i$이다.

$$migx_i = x_i'^{(t+1)} - x_i'^{(t+1)}{}_{opt}$$

$$migx_i = x_i'^{(t+1)} - rli_i \cdot \alpha \quad , \forall_i \in \Phi_{system}$$

워크스테이션 p_i에서 $migx_i$가 양의 값을 가지면 그만큼 과부하 상태이며 경량부하 상태의 워크스테이션 p_i으로 이주한다. $migx_i$가 0이면 현재 상태를 유지한다. $migx_i$가 음의 값을 가지면 그 만큼 경량 부하 상태이며 과부하 상태의 p_i에서 $migx_i$만큼의 작업을 이주 받는다. 즉 Φ_{system}내의 모든 $migx_i$의 합은 0이다.

$$0 = \sum_{\forall_i \in \Phi_{system}} migx_i$$

$migx_i$을 실제 구현에 적용할 정수 작업 $mig\chi_i'$값을 유도하는 과정이 필요하다. $mig\chi_i$가 양수이면 $mig\chi_i$의 하한 정수를 $mig\chi_i'$값으로 하고 음수이면 상한 정수를 $mig\chi_i'$값으로 한다.

$$mig\chi_i' = \lfloor mig\chi_i \rfloor \quad , mig\chi_i > 0 \ , \forall_i \in \Phi_{system}$$

$$mig\chi_i' = \lceil mig\chi_i \rceil \quad , mig\chi_i > 0 \ , \forall_i \in \Phi_{system}$$

클러스터의 모든 $mig\chi_i'$의 합을 $migy'$로 한다.

$$migy' = \sum_{\forall_i \in \Phi_{system}} mig\chi_i'$$

$migy'$가 0이 아니라면 이주 할 작업의 수와 이주 받을 작업의 수를 같게 해야 한다. 즉 $migy'$가 0이 아니고 양수라면 $migx_i$가 최대 음수인 워크스테이션 p_i에서부터 $migy'$만큼 차례로 음수를 증가시킨다. 또한 $migy'$가 음수라면 $migx_i$가 최대 양수인 워크스테이션 p_i에서부터 $migy'$만큼 차례로 양수를 증가시킨다. 상대적 부하색인에 기초한 작업 이주 알고리즘 과정은 그림 4와 같다.

과부하 워크스테이션과 경량부하 워크스테이션간의 $migx_i$의 작업 이주 알고리즘 방법 자체는 이 논문의 중요내용이 아니며, 성능 평가를 위하여 최적 적격(best fit) 방식으로 연결하여 교환한다.

5. 성능 평가

이 논문에서 제안한 상대적 부하 색인으로 작업당 수행 시간 변화율과 평균 부하값의 변화율을 사용하였다. 이 항목들은 이기종 워크스테이션의 상대적 성능 정도를 표현할 수 있어야 하며, 그것을 증명하기 위한 실험은 다음과 같다.

첫 번째, 동일한 일정 작업당 수행 시간 변화율은 직접적으로 현재의 워크스테이션의 부하정도를 나타내므로 이기종 상대적 성능 정도를 표현한다. 그러나 병렬 응용 작업 자체를 수행하면서 작업의 진행 수행시간을 파악하야 하는 어려움이 있다.

두 번째, 평균 부하값은 워크스테이션 자체의 부하 정보를 표현하지만 이기종 워크스테이션 클러스터에서는 평균 부하값 자체로는 상대적 부하 색인으로 사용될 수 없다. 그러나 시간당 평균 부하값의 변화가 기종간에 차이를 보인다면 상대적 부하 색인으로 적용할 수 있다. 다음 실험은 평균 부하값의 시간당 변화율과 시스템 성능과의 관계를 보인다.

이기종의 각 워크스테이션에서 완전 유휴 상태인 0의 평균 부하값 상태에서 동일한 작업 부하를 주었을 때, 시간당 평균 부하값의 변화가 기종간에 차이를 보이며 그림 3과 같다. 그림 5의 기울기는 변화율이고 각 워크스테이션의 상대적 처리능력을 나타낸다. 평가를 위한

워크스테이션 처리기들의 SPECint와 SPECfp성능평가 [14] 결과는 표 1과 같다.

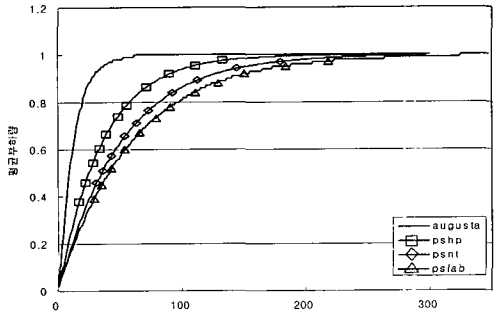

그림 5 시간당 평균 부하값 변화율

SPEC 성능 평가의 결과를 A라하고 그림 3의 기울기를 B라 했을 때 이들의 관계는

$$A = 2.96 \times \varepsilon^{-13.36 \times B}$$

의 비선형적 관계를 가지며 0.88의 상관계수를 가진다. 그러므로 평균 부하값의 변화율은 상대적 성능 정도을 표현한다고 할 수 있으며 이 것을 이용한 상대적 부하 색인 부하 균형이 가능하다.

표 1 처리기 SPEC 성능 평가

CPU Name	호스트 이름	SPECint	SPECfp
Pentium Pro 200	psnt	8.20	6.21
Pentium II 233	pslab	9.38	7.40
SPARCstation 10	augusta	1.13	1.38
HP 712/80	pshp	3.12	3.55

표 2 클러스터 구성 워크스테이션의 사양

Name	CPU	Memory
master(au)	Pentium 200	64MB
Node3(n3)	Pentium III 550 × 2	256MB
Node4(n4)	Pentium III 550 × 2	256MB
Node5(n5)	Pentium III 550	256MB
Node6(n6)	Pentium III 550	256MB
Node7(n7)	Pentium 200	80MB
Node8(n8)	Pentium 200	64MB

부하 균형 알고리즘 성능 평가를 위한 워크스테이션 클러스터 시스템은 리눅스 레드햇 7.1의 2.4.2.2-2hl 커널버전을 사용하여 PVM을 기반으로 클러스터링을 하였고 8개의 워크스테이션들은 스위칭 허브를 사용해서 10Mbps Ethernet으로 연결하였다. 시스템 구성과 워크스테이션의 사양은 표 2와 같다.

표 3 클러스터 시스템

시스템 명	구성 워크스테이션
1a	au
2a	au+n8
2b	au+n3
3a	au+n6+n7
4a	au+n5+n6+n7
5a	au+n3+n5+n6+n7
6a	au+n3+n4+n5+n6-n7
6b	au+n3+n5+n6+n7-n8
7a	au+n3+n4+n5+n6+n7+n8

제안한 알고리즘을 적용하기 위해서 클러스터에서 주 워크스테이션에 부하 관리자와 종 워크스테이션에 부하 모니터를 가지는 주종 관계의 부하 균형 시스템을 구현하였다[15]. 주 워크스테이션에는 조정자 역할을 하는 부하 관리자를 구현하였고, 시스템의 초기화와 종 워크스테이션의 부하 모니터로부터 부하 정보를 수집하여 상대적 부하를 평가한다. 종 워크스테이션에 구현한 부하 모니터는 부하 평균값과 같은 자체의 부하 정보를 주 워크스테이션의 요구 시에 전송한다.

master는 주 워크스테이션으로 나머지들은 종 워크스테이션으로 구성하고 300×300 행렬 곱셈 40개 작업과 60개 작업을 대상으로 부하 균형 실험을 하였다. 실험에 사용된 클러스터 시스템들은 표 3과 같으며 기존 방식과 제안 방식의 성능 분석을 위한 항목들은 표 4와 같다.

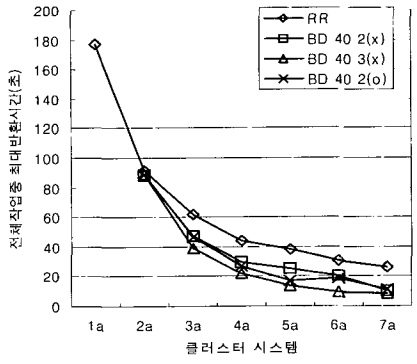

그림 6 작업 분배 성능 평가

표 4 성능 분석 항목

방식 \ 색인	분배시 부하색인	이주(재분배)시 부하색인	비교
RR	Round-Robin	×	외부부하(×)
BD	평균부하값 $\Delta La_i^{(t+1)}$	×	외부부하(○)
LADM1	평균부하값 $\Delta La_i^{(t+1)}$	$DiffLa_i^{(t+1)}$	외부부하(○)
LADM2	평균부하값 $\Delta La_i^{(t+1)}$	평균부하값 $\Delta La_i^{(t+1)}$	외부부하(○)
LADM3	평균부하값 $\Delta La_i^{(t+1)}$	$\Delta x_i'^{(t+1)}$	외부부하(○)
TTDM1	작업수행시간 $\Delta La_i^{(t+1)}$	$DiffLa_i^{(t+1)}$	외부부하(○)
TTDM2	작업수행시간 $\Delta La_i^{(t+1)}$	평균부하값 $\Delta La_i^{(t+1)}$	외부부하(○)
TTDM3	작업수행시간 $\Delta La_i^{(t+1)}$	$\Delta x_i'^{(t+1)}$	외부부하(○)
BLB	banchmark 처리 결과	banchmark 처리 결과	외부부하(○)

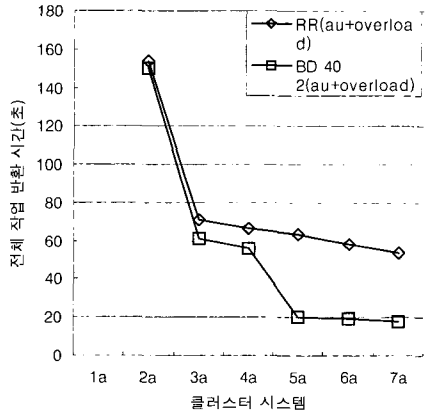

그림 7 과부하 상태에서 작업 분배

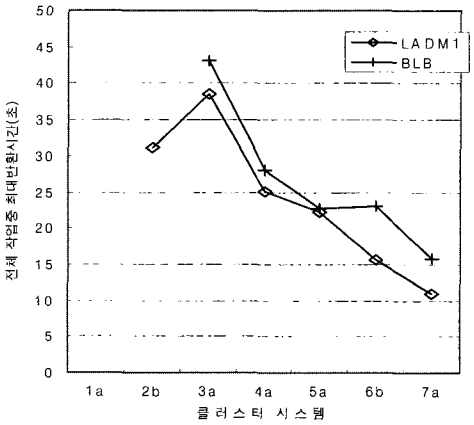

그림 8 BLB와 LADM1

그림 6에서 상대적 부하색인을 적용한 작업 분배 알고리즘(BD)방식은 PVM의 작업 분배 방식인 라운드 로빈(RR)방식과 전체 작업중 최대 반환 작업 시간을 비교하였다. 특히 BD 방식은 외부 작업 유입 환경에서도 외부 작업 유입이 없는 환경과 변화를 보이지 않았으며 RR방식보다 월등히 단축된 시간을 보였다. 클러스터를 구성하는 워크스테이션들 중에서 저성능의 워크스테이션에 특히 과부하가 발생되는 환경에서의 BD방식과 RR방식간의 반환시간 차이는 그림 7처럼 더욱 큰 차이를 보였다.

다음은 이 논문에서 제안한 상대적 부하색인을 적용한 작업 분배와 작업 이주 알고리즘들을 성능 테스트 프로그램 수행후 부하 균형 알고리즘을 수행하는 기존 방식(BLB)과 전체 작업중 최대 반환 시간을 비교한다. 그 중에서 안전적인 결과를 보이는 LADM1와 TTDM1의 결과는 그림 8과 그림 9와 같다.

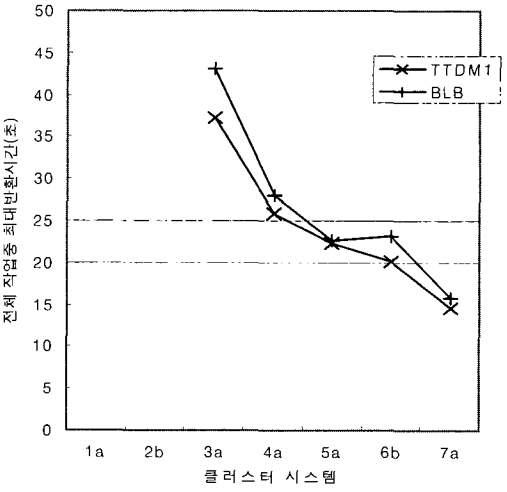

그림 9 BLB와 TTDM1

그림 8에서 작업 분배을 위한 상대적 부하 색인으로 평균 부하값의 변화 $\Delta La_i^{(t+1)}$를 사용하고 작업 이주를 위한 상대적 부하색인으로 $DiffLa_i^{(t+1)}$를 사용하는 LADM1방식은 BLB방식의 반환시간보다 빠른 결과를 보인다. 3a 시스템보다 2b 시스템에서 반환시간이 더 빠른 이유는 2b시스템중에서 듀얼 CPU을 가진 워크스테이션 때문이다. 작업 분배을 위한 상대적 부하 색인으로 작업당 수행 시간 변화값을 사용하고 작업 이주를 위한 상대적 부하색인으로 $DiffLa_i^{(t+1)}$을 사용하는 TTDM1방식과 BLB방식의 비교는 그림 9와 같다. LADM1와 유사하게 각 클러스터 시스템에서 안정적으로 BLB방식보다 빠른 반환 시간을 보인다.

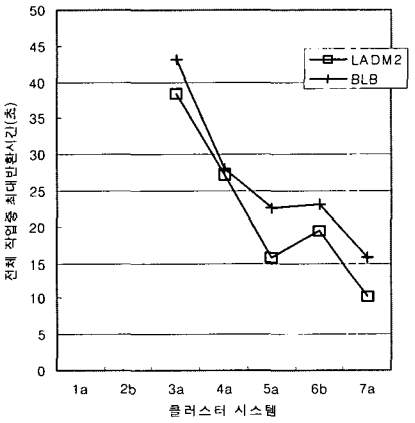

그림 10 BLB와 LADM2

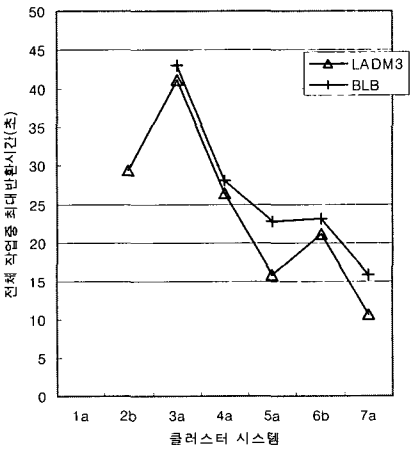

그림 11 BLB와 LADM3

LADM2와 LADM3의 경우는 그림 10과 그림 11과 같으며 BLB보다 빠른 반환 시간을 보이나 5a 시스템에서 저성능 워크스테이션이 추가된 6b 시스템에서는 안정적 반응을 보이지 못하고 있다.

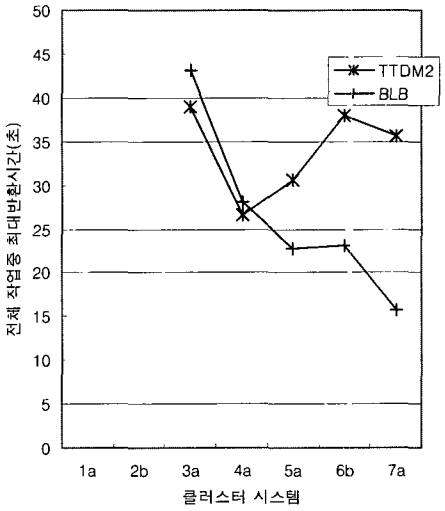

그림 12 BLB와 TTDM2

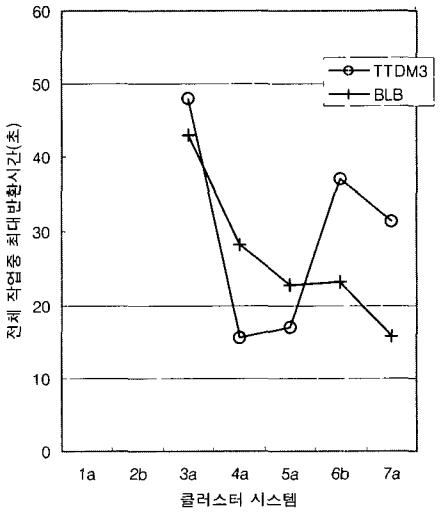

그림 13 BLB와 TTDM3

TTDM2와 TTDM3의 그림 12와 그림 13의 결과에서는 3a와 4a 경우의 클러스터 시스템에서는 비교적 빠른 결과는 보이지만 불안정하고 이기종 클러스터 시스

템에서 적절히 작용하고 있지 못하다. 작업 이주를 위한 부하색인으로 단순한 $\Delta La_i^{(t+1)}$를 사용하는 TTDM2와 TTDM3방식은 분배 알고리즘으로 분배된 작업이 고성능의 워크스테이션으로 분배되었지만, 이주시점에서 파악한 $\Delta La_i^{(t+1)}$과 $\Delta x_i'^{(t+1)}$는 과부하의 워크스테이션에게 많은 작업을 이주시키는 결과로 인해서 반대의 결과를 보인다.

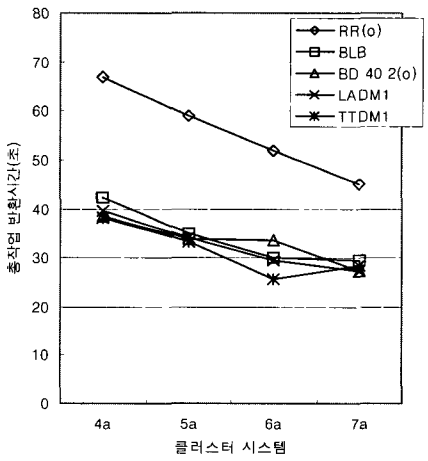

그림 14 BLB와 제안 방식

전체적으로 모든 환경에 잘 적응하는 부하 균형 알고리즘은 외부 작업이 유입되는 환경에서 LADM1과 TTDM1이며 성능 테스트 프로그램 의존적 부하 균형 방식인 BLB방식보다도 빠른 전체작업반환시간을 보인다.

6. 결 론

서로 다른 사용자들과 그룹들, 다른 외부 작업 수행 요구들, 다른 처리능력을 갖는 워크스테이션으로 구성된 리눅스 워크스테이션 클러스터에서 각 워크스테이션은 병렬 수행을 위한 작업들을 부여받아서 수행하게 된다. 이 논문에서는 병렬 응용 프로그램의 총작업반환시간을 줄이기 위한 부하 균형 방법으로 상대적 부하 색인을 기초해서 작업 분배와 이주 알고리즘을 제안하였다. 또한 기존 방식과 제안 방식의 성능 비교 평가를 위해서 이기종 워크스테이션 클러스터를 구축하고 부하 균형 시스템을 중앙식의 주종관계에 의한 부하 관리자와 부하 모니터를 설계 구현하였다.

제안한 부하 균형 방법은 다음과 같은 특징을 갖는다.

첫 번째, 이기종 워크스테이션 클러스터 시스템에서 동적 부하 균형 방식을 제공한다. 두 번째, 성능 테스트 프로그램 결과를 부하색인으로 사용하는 기존 방식이 아닌 자체 작업 수행중에 발생하는 여러 요소들을 이용한 상대적 부하 색인을 사용한다. 세 번째, 부하 균형을 위한 작업 교환 방식이 이웃 노드간의 방식이 아닌 전체 클러스터 시스템 내에서 최적의 과부하 워크스테이션과 경량부하 워크스테이션간의 교환 방식이다. 네 번째, 작업 단위 부하 균형 방식으로 실제 구현에 적용되도록 정수개의 작업을 유도한다.

즉 기존의 방법이 성능 테스트 프로그램 의존적 부하 균형 방법이라면 논문에서 제안한 방법은 자체 응용 프로그램에 의한 성능 테스트 프로그램 독립적 부하 균형 방법이다.

성능 평가 결과에서 클러스터 구성 워크스테이션에 동일 작업 분배 방식인 라운드 로빈 방식과 제안한 상대적 부하 색인에 기초한 균형 분배 방식의 비교는 2배 내외의 총작업반환시간 단축을 보였으며, 외부 작업 유입 상황에서도 유입이 없는 상황과 변화가 없는 시간 단축을 보였다. 또한 평균 부하값의 변화율과 처리 작업의 변화율을 상대적 부하 색인으로 하고 이를 기초한 작업 이주 방식은 이기종 워크스테이션 클러스터 부하 균형뿐만 아니라 동기종 워크스테이션 클러스터 부하 균형에서도 향상된 작업 반환 시간을 보였다. 또한 동일한 환경에서 성능 테스트 프로그램에 기초한 작업 분배와 작업 이주 방식보다 총작업반환시간이 단축됨을 보였다.

향후연구에는 동적으로 작업 이주 시점을 결정할 수 있는 이주 결정 스케줄링과 현재의 중앙식 부하 균형 방식에서 분산식 부하 균형 방식을 연구할 수 있다.

참 고 문 헌

[1] F. Meisgen, E. Speckenmeyer, "Dynamic Load Balancing on Clusters of Heterogenous Workstations," Dept of Computer Science University of Cologne, Tech Report No. 97-261, 1997.

[2] J. C. Fabero, I. Martin, A. Bautista, S. Molina, "Dynamic Load Balancing in a Heterogeneous Environment under PVM," IEEE Procedings of PDP'96, 1996.

[3] Khaled Al-Saqabi, Steve W. Otto, Jonathan Walpole, "Gang Scheduling in Heterogeneous Distributed System," Oregon Graduate Institute, Tech Report, 1994.

[4] T.s Hsu, J.C. Lee, D.R Lopez, "Task Allocation on a Network of Processors," IEEE Trans. on Computers, Vol 49, No 12, Dec 2000.

[5] Olivier DALLE, "LoadBuilder:a tool for generating and modeling workloads in distributed workstation environment," proceeding of 9th ISCA, 1996.
[6] Domenico Ferrari, "A Study of Load Indices for Load Balancing Schemes," Report No. UCB/CSD 86/262, October 1985.
[7] Virginia Mary Lo, "Heuristic Algorithms for Task Assignment in Distributed System," IEEE Trans on Computers, 37(11), November 1988.
[8] Chao-Wei Ou, Sanjay Ranka, "Parallel Incremental Graph Partitioning," IEEE Transactions on Parallel and Distributed Systems, 8(8), August 1997.
[9] S. Zhou, "A Trace Driven Simulation Study of Dynamic Load Balancing," IEEE Trans. Software Engineering, Vol. 14, No.9, pp.1327-1341, Sept 1988.
[10] O. Kremien, J. Kramer, J. Magee, "Scalable, Adaptive Load Sharing for Distributed System," IEEE Parallel and Distributed Technology, Vol.1, No.3, pp.62-70, Aug 1993.
[11] Jeremy Casas, Jonathan Walpole, "MPVM: A Migration Transparent Version of PVM," Tech report, Dept of Computer Science and Engineering, Oregon Graduate Institute of Science & Technology, Feb 1995.
[12] J. Casas, D. Clark, P Galbiati, R. Konuru, S. Otto, "MIST:PVM with Transparent Migration and Checkpoint," Dept of computer Science and Engineering Oregon Graduate Institute, 1995.
[13] Leen Dikken, Peter Sloot, "DynamicPVM, Dynamic Load Balancing on Parallel System," High Performance Computing and Networking, Vol 797, pp273-277, 1994.
[14] http://www.specbench.org/osg/cpu2000/results/cpu2000.html
[15] 지병준, 이광모, "병렬 가상 컴퓨터에서 작업 부하를 고려한 타스크 관리자의 설계 및 구현", 정보과학회 병렬처리연구회 학술발표논문집, 제7권 2호, 1996.5.

지 병 준
1983년 2월 ~ 1987년 3월 한림대학교 전산계산학과(이학사). 1987년 9월 ~ 1993년 2월 중앙대학교 컴퓨터공학과(공학석사). 1995년 9월 ~ 2002년 2월 한림대학교 컴퓨터공학과(공학박사). 1994년 3월 ~ 현재 한림정보산업대학 전산정보처리과 부교수. 관심분야는 분산병렬처리, 클러스터 시스템, 웹 분산 처리

이 광 모
1992년 2월 서울대학교 계산통계학과(박사). 1980년 9월 ~ 1985년 1월 조선대학교 전자계산학과 조교수. 1985년 2월 ~ 현재 한림대학교 정보전자공과대학 컴퓨터공학부 교수. 1985년 3월 ~ 1989년 2월 한림대학교 전자계산소 소장. 1992년 1월 ~ 1993년 1월 Florida State University 방문교수. 1996년 3월 ~ 1998년 2월 한림대학교 정보전자공과대학 학장. 1999년 3월 ~ 현재 한림대학교 교무처장. 관심분야는 병렬처리. 프로그램 언어, 병렬 컴파일러