주기성을 이용한 제한된 부하균형 기법

임 종 규† · 박 한 규† · 장 순 주†† · 구 용 완†††

요 약

분산 시스템에서 부하 균형의 목적은 한 프로세서로부터 다른 프로세서로 작업을 적절히 전송함으로서 전체 시스템의 성능을 향상시키는 것이다.

본 논문에서는 수행 주기성과 탐색 한계를 이용한 제한된 부하균형 기법을 제안하였다. 본 알고리즘은 시스템에 있는 각 노드의 최신 상태를 기록하기 위해 탐색 과정에서 수집된 정보를 사용한다. 이러한 정보는 각 노드에서 관리되는 데이터 구조에 저장된다. 본 데이터 구조를 사용함으로써, 어느 노드를 탐색할 것인가를 판단할 수 있어 무차별하게 탐색하는 것을 방지한다.

시뮬레이션 결과, 제안한 주기성 부하균형 알고리즘은 평균응답시간을 단축시킨다는 것과 성능이 수행 주기에 의존함을 발견하였다. 따라서, 훌륭한 수행 주기는 낮은 평균 응답시간과 부하 이주를 위한 적당한 노드가 존재한다면 그 노드를 발견할 높은 확률을 제공한다는 결론을 내렸다.

A restricted load balancing scheme using a periodicity

Jong-kyu Im† · Han-kyu Park† · Soon-ju Chang†† · Yong-wan Koo†††

ABSTRACT

In a distributed system, the goal of the load balancing is to improve the performance of th whole system by appropriately transferring work from one processor to another.

In this paper, we presents a restricted load balancing algorithm using the perform period(Pt) and the probing limits(Lp). Our algorithm utilizes the information gathered during probing to keep track of the recent state of each node in the system. This information is stored in a data structure that is maintained at each node. Using this data structure, the algorithm decides which nodes to probe, thus preventing indiscriminate probing.

Using simulation, We found that algorithm based on the proposed periodic load balancing is capable of reducing the mean response times and the performance is strongly dependent upon the perform period. Therefore, We conclude that the good perform period provides short mean response time and high probability of finding a suitable counterpart if one exists.

1. 서 론

분산시스템(distributed systems)은 통신망에 의해 상호 연결된 자율적인 컴퓨터들의 집합이다. 통신망을 통해 시스템의 자원들은 다른 곳에 위치한 사용자에 의해 공유될 수 있어, 전체 시스템의 부하 균형을 이루는 것은 분산시스템에서 가장 중요한 현안들 중 하나가 되었다.

분산시스템은 많은 잠재적인 처리용량을 가지고 있으나, 각 프로세서의 이용도(utilization) 분석 결과에

† 준 회 원 : 수원대학교 전자계산학과
†† 정 회 원 : 수원대학교 전자계산학과
††† 종신회원 : 수원대학교 전자계산학과
논문접수 : 1997년 9월 25일, 심사완료 : 1998년 4월 27일

따르면 대부분의 처리 용량(processor capacity)이 낭비되고 있다[1]. 따라서, 사용되지 않는 처리 용량을 다른 사용자에게 제공할 수 있다면 프로세서의 이용률을 개선 할 뿐 아니라 향상된 작업 반환 시간(turn-around time)도 보장받을 수 있다. 이를 위해서 사용자의 작업중 일부를 유휴(idle) 프로세서로 적절히 분배시키는 부하 균형 기법(load balancing schemes) 등이 연구되어 왔다.

부하 균형 기법은 시스템의 환경에 적응하는 형태에 따라 정적 방식(static methods)과 동적 방식(dynamic methods)으로 나눌 수 있으며, 작업 전송을 요구하는 프로세서의 부하 상태에 따라 송신자 주도 방식(sender-initiative methods), 수신자 주도 방식(receiver-initiative methods), 대칭 방식(symmetric methods)으로 분류한다.

송신자 주도 탐색은 과부하 상태로 전이된 노드가 저부하 상태인 노드를 찾아서 그 노드로 부하를 이주시키기 위한 기법이며, 이에 반해 수신자 주도 탐색은 저부하 상태로 전이된 노드가 과부하인 노드를 찾아서 그 노드로부터 부하를 이주 받기 위한 기법이다. 관련 연구에 따르면,

시스템의 부하가 낮은 경우에는 과부하 노드가 부하균형 알고리즘의 협상을 주도하는 송신자 주도 탐색 방식이, 시스템의 부하가 높은 경우에는 저부하 노드가 주도하는 수신자 주도 탐색 방식이 더 나은 성능을 보이며, 대칭 탐색 방식은 이들을 혼형(hybrid)한 방식으로 모든 시스템의 부하 상태에서 상기 두 가지 방식에 비해 우수한 성능을 보인다[2][4].

하지만, 지금까지 연구된 부하균형 알고리즘은 부하균형 알고리즘 수행 자체에 대한 비용은 완전히 무시하고 타스크들의 평균응답시간을 비교/평가하였다. 이러한 평가는 그렇지 않아도 과부하 상태인 노드에 불필요하게 많은 부하균형 세션(session)으로 인해 더욱 시스템을 과부하로 이끌 수 있는 상황을 간과하였기 때문에 정확한 성능 평가로 삼기에는 무리가 있다.

본 논문에서는 이와 같은 기존의 부하균형 알고리즘 평가에서 무시했던 부하균형 알고리즘 수행 비용을 고려하여 적절한 노드의 탐색을 어느 임계적 주기(periodic)를 두어, 그 주기마다 탐색을 수행하는 주기적 부하균형 알고리즘을 설계하였다.

제안된 알고리즘의 설계 목표는 노드의 상태 정보를 서로 교환하기 위한 협상의 시기를 서로 동기화 하는 수행 주기(P_t)를 두도록 하였다. 이는, 기존 대칭적 알고리즘이 새로운 작업이 도착하거나 수행의 완료로 시스템을 떠날 때 부하 상태가 변경되어 협상을 시작하기 때문에 발생할 수도 있었던 과도한 부하균형의 문제점을 예방할 수 있다. 본 알고리즘은 평균 작업 응답 시간 측면에서 비교된 기존의 알고리즘에 비해서 개선된 성능을 보였으며 성능평가에 사용된 파라메타는 사용자가 정의 가능하며 시스템의 확장 등에 잘 적용될 수 있을 것으로 여겨진다.

2. 관련연구

부하균형 알고리즘의 유형은 위치정책에서 부하를 공유하기 위한 권한을 갖는 전략에 따라 송신자 주도(Sender-initiated), 수신자 주도(Receiver-initiated), 대칭적 주도(Symmetrical-initiated)전략으로 나눌 수 있다[9].

송신자 주도 전략은 자신의 노드가 과부하 상태가 되었을 때만이 부하를 공유하기 위해서 유휴 또는 저부하 상태의 노드를 찾는 반면, 수신자 주도 전략은 자신의 노드가 유휴 또는 저부하 상태가 되었을 때만이 부하를 공유하기 위한 과부하 상태의 노드를 찾는 방법이다.

대다수의 관련 연구들은 부하균형 알고리즘 설계시 송신자 또는 수신자 주도 전략중 어느 하나를 선정하였다. 〈표 1〉에서도 볼 수 있듯이 이러한 기반으로 연구된 알고리즘은 각 주도 전략의 실행 특성상 장, 단점을 여전히 내포하고 있다[3][5][6].

대칭적 주도 전략은 송신자 주도와 수신자 주도 전략을 조합한 형태로서 송신자와 수신자 노드 모두가 부하 분산을 요청할 수 있는 방법이다. 따라서, 송신자와 수신자 주도형에서 나타나는 단점을 보완하자는 데 그 설계 목적이 있다.

[10][13]에 따르면, 성능평가 결과 송신자나 수신자 주도중 한가지만을 이용하는 알고리즘에 비해 대칭적 주도 부하균형 알고리즘이 상대적으로 시스템이 과부하 상태와 저부하 상태 모두 훌륭한 성능을 보임을 밝혔다. 흥미로운 것은, 전체 분산시스템에 들어오는 작업 도착율이 시스템 처리능력 이하로 도착할 경우 송신자 주도 방식이 수신자 주도 방식보다 빠른 응답시간을 보

〈표 1〉 부하균형 알고리즘 비교
〈Table 1〉 Comparing load balancing algorithms

기법	Graph	Bidding	Drafting	Gradient	Soh
제안자	H.S.Stone	D.J.Farber	L.M.Ni	F.C.H.Lin	J.W.Soh
분류	정적기법	동적기법			
		송신자주도	수신자주도		
부하균형 기준	Cutset	Bid	Draft Standard	Gradient surface	Light load 기준치
부하이주 대상	소규모, 단순시스템	최선의 Bid	최대 Draftage	최대 전달 압력	최대 과부하 프로세서
적응성	소규모, 단순시스템	소규모 시스템	대규모, 과부하시스템	대규모, 경부하시스템	대규모, 과부하시스템
장/단점	-단순성,정확성 보장 -자원이용율에 대한 정확한 예측이 필요	-정확성 필요 -부하이주를 위한 판단 시간 소요 -단순성 결여	-공정성,안정성 보장 -과도한 통신으로 부하이동에 시간이 소요됨	-안정성, 공정성 보장 -부하이주를 위한 판단 빠름	-공정성,안정성 보장 -부하 테이블이 필요없으며, 항상 부하를 감시 할 필요 없다
제어방식	중앙 제어		분산제어		

인 반면, 작업 도착율이 높은 경우에는 저부하 노드가 주도하는 수신자 주도 방식이 더 나은 성능을 보였다.

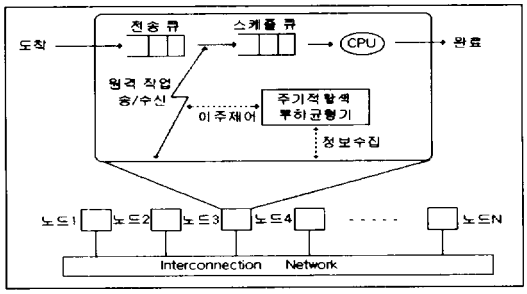
이와 같이, 시스템의 작업 도착율에 따라 다른 응답 시간을 보이는 알고리즘 특성에 착안하여 제시된 대칭적 주도 방식은 이들 두 가지 방식을 혼형 하였기 때문에 모든 시스템의 부하 상태에서 적응적으로 동작하도록 설계되었으며 시뮬레이션으로 입증되었다[2][4]. 하지만, 본 알고리즘에도 몇 가지 문제점이 발생했는데, 이들중 한가지 중요한 것은 시스템의 부하 상태가 변할 때마다 부하균형 알고리즘을 수행해야 함으로써, 그렇지 않아도 과부하 상태인 노드가 더욱 과부하에 빠지게 하며 마침내 전체 분산 시스템의 안정성을 해치는 결과를 초래할 수 도 있다는 사실이다.

주기적 부하균형 알고리즘은 이와 같은 시스템 불안정성(instability)을 방지하기 위해 시스템의 상태정보가 변경될 때마다 부하균형을 시도하는 것이 아니라 협상 시기인 수행주기가 되었을 때 부하균형을 시도하자는 데 그 설계 목적이 있다.

3. 주기적 부하균형 시스템 설계

3.1 시스템 모델

본 논문에서는 (그림 1)에서 보인 것과 같은 시스템 모델을 갖는다. 시스템 모델을 위한 가정은 다음과 같다. 노드간의 통신은 메시지 전달(message passing) 기법을 사용하며, 네트워크에는 결함이 존재하지 않기 때문에 어떠한 메시지 손실도 발생하지 않으며 각 메시지는 전송된 순서대로 도착한다.



(그림 1) 시스템 모델
(Fig. 1) System Model

네트워크는 자치적인 처리 능력을 가지고 있는 N개의 노드들로 구성되어 있으며, 각각의 노드는 타스크를 저장하기 위한 두 개의 큐를 가지고 있다. 전송 큐(transfer queue)는 새로 로컬에 도착한 타스크가 저장이 되며 무한한 용량의 크기로 구성되었고, 이곳에 저장된 타스크는 FIFO 형태로 서비스되는데 로컬 또는 원격으로 처리가 가능하다. 스케줄 큐(schedule queue)는 로컬에서 처리하기 위한 모든 작업들이 서비스를 위해 대기하는 큐이다. 스케줄 큐는 프로세서의 처리 한계로 인해 프로세서 처리 용량만큼의 크기로 제

한한다. 이 큐에 일단 작업이 도착하면 무조건 로컬에서 처리하는 단일 이주(single migration) 기법이 적용된다. 이는 최적의 노드를 찾아 작업이 계속적으로 이주되는 것을 예방하기 위하여 고려되어진다.

3.2 부하의 3상태

본 주기적 부하균형 알고리즘에서는 부하의 상태를 큐의 길이로 계산하여 다른 노드에게 방송(broadcast)하는 것이 아니라, 부하 정도를 3가지 상태로 분류하여 방송하게 된다. 이러한 전략은 부하계산에 따른 전송지연을 줄이며 불필요하게 부하균형이 이루어지는 것을 막는데 있다.

◉ 과부하(H-Load) :
부하의 정도가 높아서 다른 프로세서로 프로세스를 이주시켜야 하는 경우
◉ 적정부하(N-Load) :
부하의 정도가 적절하여 프로세서간 이주가 불필요한 경우
◉ 저부하(L-Load) :
부하의 정도가 낮아 다른 프로세서들로부터 프로세스를 받아들 일 수 있는 경우

〈표 2〉 3-레벨 부하 측정 전략
〈Table 2〉 3-level load measurement scheme

부하상태	의 미	평가기준
L-load	저부하 상태	큐길이 ≤ 하한임계치
N-load	적정부하 상태	하한임계치 〈 큐길이 ≤ 상한임계치
H-load	과부하 상태	큐길이 〉 상한임계치

3.3 주기적 부하 균형 알고리즘

3.3.1 상태벡터(State Vector)

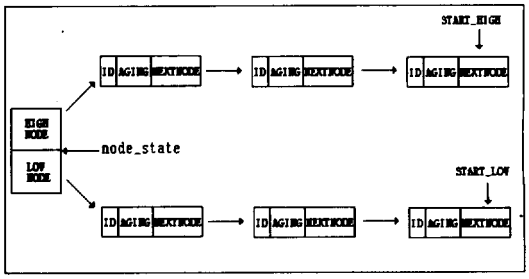
본 절에서는 주기적 부하균형 알고리즘이 탐색 과정에서 적절한 노드를 선정하기 위해 이용하는 자료구조인 상태 벡터(state vector)를 (그림 2)에서 기술하였다.

수행 주기에 의해 시작된 부하균형 알고리즘은 탐색 과정에서 수신한 응답메시지에 실려있는 노드의 부하상태를 수집하여 SV의 구분된 노드의 링크에 연결한다. (그림 3)는 상태 벡터가 어떻게 작동하는가 이해를 돕기 위해 그림으로 나타낸 것이다. 이때, 수집된 노드의 상태에 따라 과부하 또는 저부하 노드 리스트의 앞에 삽입이 되므로 LIFO(Last In, First Out)로 작동된다. 이는 최근의 부하 정보에 따라 적절한 노드에게 먼저 probing하기 위함이다. 오래된 상태 정보는 그 만큼 잘못된 전송을 이끌 수 있는 확률이 높아진다. 탐색 주기마다 aging은 리스트의 마지막까지 1씩 증가하며 새로 삽입된 링크의 기본 값은 1이다. 만약, 정의된 링크의 한계치에 이르면 aging이 가장 높은 링크를 제거(free)하는데, 이 과정은 리스트의 마지막 리스트를 삭제하면 된다.

```
typedef struct SV *noderef, *START_HIGH, *START_LOW;
typedef struct Dispatcher  *node_state : /* 상태 리스트 링크 */
struct SV {
   noderef  *PRENODE :   /* 이전 노드의 링크 */
   int      ID :         /* 노드의 식별자 저장 */
   int      AGING :      /* 경과된 탐색 주기 */
   noderef  *NEXTNODE : /* 다음 노드의 링크 */
 } :

struct Dispatcher {
   noderef  *HIGHNODE : /* 과부하 리스트 노드 링크 */
   noderef  *LOWNODE :  /* 저부하 리스트 노드 링크 */
 } :
```

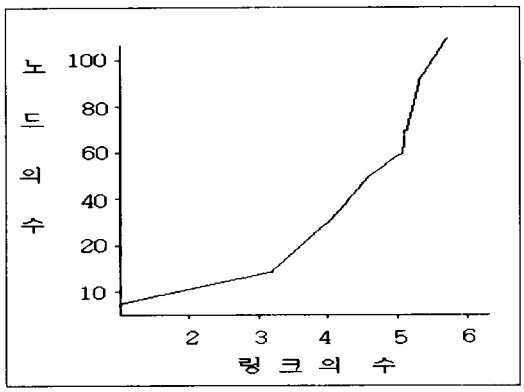
(그림 2) 상태 벡터를 위한 자료 구조
(Fig. 2) Data structure for state vector



(그림 3) 상태 벡터의 예
(Fig. 3) An example of State Vector

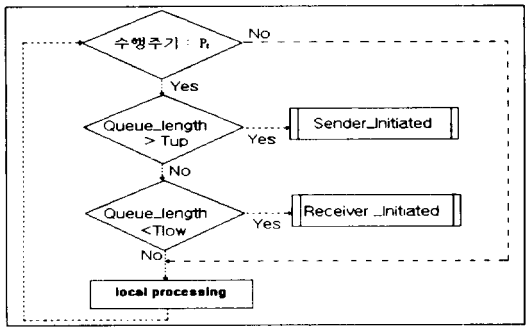
연구결과 노드의 링크 한계치(bound)는 전체 노드의 개수가 n 인 경우 최대 $\log n + 1$이면 적절한 노드를 탐색할 수 있는 탐색한계를 표현하기 충분하다. 그

러므로, 탐색한계만큼 노드의 정보를 유지하고 있다면 임의적으로 탐색하는 것보다 효율적이다. (그림 4)는 노드의 개수와 링크 수와의 관계를 나타낸 것이다.



(그림 4) 노드의 개수와 링크 수와의 관련성
(Fig. 4) Relationship between the number of node and link

3.3.2 알고리즘 흐름도



(그림 5) 주기적 부하균형 알고리즘 흐름도
(Fig. 5) The diagram of periodic load balancing algorithm

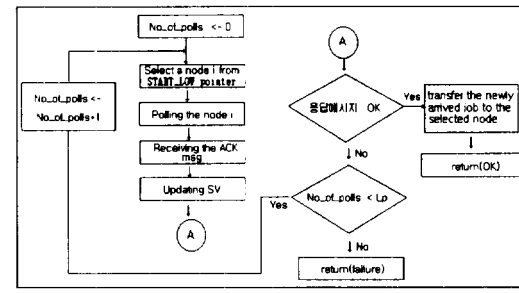
(그림 5)에서 볼 수 있듯이, 제안한 주기적 부하균형 알고리즘은 송신자 주도 알고리즘과 수신자 주도 알고리즘으로 구성된다.

주기적 부하균형 알고리즘은 노드들의 부하가 높은 경우에는 수신자 주도 알고리즘의 효율성을 취하고 노드들의 부하가 낮은 경우에는 송신자 주도 알고리즘의 효율성을 취하므로 개선된 성능을 보인다.

부하균형 방법은 탐색도중에 적절한 노드가 발견되면 탐색을 중단하는 최초발견(first-found)방법, 탐색을 한 후 탐색된 노드 중에서 조건에 부합하는 부하가 가장 작은 노드를 선정하는 최단발견(shortest-found) 방법, 그리고 수신자 주도 시에 전송될 작업을 미리 예약하는 예약(reservation) 방법으로 분류된다. 송신자 주도 탐색에서는 탐색비용이 전체 작업의 처리 시간에 끼치는 영향에 따라서 최초발견 방법과 최단발견 방법이 사용되고 있다. 탐색비용이 작업의 처리시간에 끼치는 영향이 작으면 최단발견 방법이 우수한 작업반환시간을 보이며, 수신자 주도의 경우에서는 최초발견 방법과 예약 방법이 사용될 수 있다. 본 연구에서는 송신자 주도와 수신자 주도는 최초발견 방법을 사용한다.

송신자 주도와 수신자 주도에서 최초발견 방법을 사용하는 이유는 탐색을 받은 프로세서에서 전송 가능하다는 응답을 한 작업이, 탐색이 종료되기 전에 지역적으로 처리될 수 있는 가능성을 허용하기 위함이다. 본 연구에서는 탐색에 소요되는 시간이 작업의 전송시간에 비해서 상대적으로 작지만 전송 대상으로 선정된 작업이 실제 탐색이 종료되기 전에 지역적으로 처리될 가능성도 있다.

(그림 6)은 주기적 부하균형 알고리즘에서 노드가 과부하로 판명되어 송신자 주도 탐색 알고리즘이 호출되었을 때 처리 과정을 보인 것이다.



(그림 6) 송신자 주도 알고리즘 흐름도
(Fig. 6) The diagram of Sender Initiated algorithm

(그림 6)의 송신자 주도 알고리즘의 작업 전송 시도 과정의 단계별 설명은 아래와 같다.

i) 본 루틴은 주기 탐색에 의해 호출되며 대기중인 작업의 수가 임계치보다 크면 작업전송을 시작한다. 그렇지 않으면 전송을 시도하지 않고 로컬로 처리한다.

ii) 작업 전송이 타당하면 상태벡터 SV의 START_LOW 포인터가 가리키는 노드부터 탐색한다. 탐색 시도 횟수가 탐색 한계(L_p)를 넘지 않으면

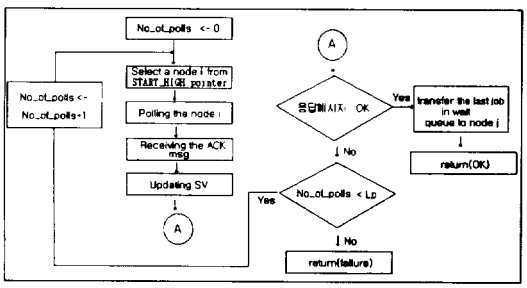
작업의 수신 요구를 위해 SV에서 선택된 노드를 계속 탐색한다.

iii) 탐색 패킷에 대한 응답을 기다려서 긍정적으로 응답한 저부하 노드(Node i)로 작업을 전송한다.

iv) 긍정적인 응답이 하나도 없으면 탐색은 실패하고 로컬로 처리되며 다음번 수행 주기가 될 때까지 기다린다.

탐색을 받은 노드는 대기중인 작업의 수가 임계치보다 작으면 긍정적인 응답을 나타내는 정보와 부하상태를 나타내는 정보를 응답패킷에 실어서 탐색 시도한 노드에 전송하고, 그렇지 않으면 부정적인 응답을 나타내는 정보와 부하 상태를 실은 응답패킷을 탐색 시도한 노드에 전송한다.

송신자 주도 알고리즘은 주기적 부하균형 알고리즘에 의해 호출되며 작업의 전송여부를 결정하여 전송이 타당하면 대상 노드를 선정하여 작업 처리를 의뢰한다. 이때 작업 전송여부와 작업 수신 여부를 결정하는 기준으로 임계치을 사용하며, 탐색된 노드 중에서 최초로 긍정적인 응답을 한 노드를 선정하는 최초발견 기법을 이용한다. 송신자 주도 알고리즘은 노드들의 부하가 낮으면 전송하고자 하는 작업이 적을 뿐 아니라 대상 노드를 찾을 가능성이 아주 높으므로 효율적이다. 그러나 노드의 부하가 높으면 도착하는 작업이 많아져 전송시도 횟수가 증가할 뿐 아니라 작업을 처리할 대상 노드를 찾을 수 있는 가능성도 작아지므로 노드의 부하가 높은 경우에는 비효율적이다.



(그림 7) 수신자 주도 알고리즘 흐름도
(Fig. 7) The diagram of receiver initiated algorithm

상기 (그림 7)의 수신자 주도 알고리즘 단계별 수행 과정은 다음과 같다.

i) 각 노드는 수행 주기가 되었을 때 주기적 부하균형 알고리즘에 의해 호출되어, 대기중인 작업의 수가 임계치보다 작으면 작업 전송을 요구하는 과부하 노드(Node i)를 찾는다.

ii) 탐색 시도 횟수가 탐색한계를 넘지 않으면 작업의 전송 요구를 위해서 SV의 과부하 링크에서 START_HIGH 포인터가 가리키는 노드를 탐색한다.

iii) 탐색 패킷에 대한 응답을 기다려 부정적인 응답이 도착하면 단계 ii)를 반복한다. 긍정적인 응답인 경우 저부하 노드(Node j)에게로 작업이 전송되어 오므로 작업을 이주 받고 탐색을 종료한다.

iv) 긍정적인 응답이 하나도 없으면 탐색은 실패하고 다음번 주기가 될 때까지 로컬로 부하를 처리하며 기다린다.

탐색을 받은 노드는 대기중인 작업의 수가 임계치보다 크면 탐색 시도한 노드에 작업을 전송하고, 그렇지 않으면 부정적인 응답을 나타내는 정보와 부하 상태를 실은 응답패킷을 탐색 시도한 노드에 전송한다.

수신자 주도 알고리즘은 작업이 종료되면 작업 전송 요구 여부를 결정하고 전송 요구가 타당하면 대상 노드를 선정하여 작업전송을 요구한다. 작업 전송 여부를 결정하기 위한 기준으로 임계치을 사용하며, 탐색된 노드 중에서 조건을 만족하는 최초의 노드를 대상으로 선정하는 최초발견 방법을 이용한다. 수신자 주도 알고리즘은 노드들의 부하가 높으면 작업을 요구할 대상 노드를 찾게 될 가능성이 커지므로 효율적이나 노드들의 부하가 낮으면 작업요구 횟수가 증가할 뿐 아니라 대상 노드를 찾을 수 있는 가능성도 낮아지므로 비효율적이다.

3.3.3 알고리즘

주기적 부하균형 알고리즘은 송신자 주도 알고리즘과 수신자 주도 알고리즘으로 구성된다. 이 알고리즘은 부하가 높은 경우에는 수신자 주도 알고리즘으로 부하가 낮은 경우에는 송신자 주도 부하균형 알고리즘으로 이동하여 처리한다. 주기적 부하균형 알고리즘은 (그림 8)과 같다.

(그림 9)의 송신자 주도 알고리즘은 주기적 부하균형 알고리즘에 의해 호출되며 작업의 전송 여부가 타당하면 대상 노드를 선정하여 작업 처리를 의뢰한다. 작

```
Periodic_Load_balancing_Algorithm { 주기적 부하균형 알고리즘}
{ while(1) {
      for( ; !Pt ; ) local-processing; /* Pt : 수행 주기 */
      if ( Queue_Length > Tup )      /* Tup : 상한임계치 */
        { call Sender_initiated( );   /* 과부하 상태 */
          return(OK); }
      else if ( Queue_Length < Tlow ) /* Tlow : 하한임계치 */
        { call Receiver_initiated( );   /* 저부하 상태 */
          return(OK); }
      Send a job from the transfer queue
          to the schedule queue ;  /* 적정부하 상태 */
   } /* end of while */
}
```

(그림 8) 주기적 부하균형 알고리즘
(Fig. 8) Periodic load balancing algorithm

```
Sender_Initiated( )
{  No_of_Polls = 0 ;
   while ( No_of_Polls < Lp ) { /* No_of_Polls : 탐색 개수기 */
      Select a node i from START_LOW pointer;
      Polling the node i :      /* 저부하 노드 선정 */
      Waiting the ACK msg ;
      Updating the SV ;
      if ( the ACK signal is OK )
         then { transfer the newly arrived job to the
         selected nodes ;
               return (OK) ; }
         else  No_of_Polls++ ;
   }
  return (failure) ;
}
```

(그림 9) 송신자 주도 알고리즘
(Fig. 9) Sender initiated algorithm

```
Receiver_initiated( )
{  No_of_Polls = 0 ;
   while ( No_of_Polls < Lp ) {
      Select a node i from START_HIGH pointer;
      Polling the node i :      /* 과부하 노드 선정 */
      Waiting the ACK msg ;
      Updating the SV ;
      if ( the ACK signal is OK )
         then { transfer the last job in transfer queue
               to node j ;   /* 저부하 노드에게로 부하이주 */
                return (OK) ; }
         else  No_of_Polls++ ;
   }    /* end of while */
  return (failure) ;
}
```

(그림 10) 수신자 주도 알고리즘
(Fig. 10) Receiver initiated algorithm

업 전송 여부와 작업 수신을 결정하는 기준으로 임계치을 사용한다.

(그림 10)의 수신자 주도 부하균형 알고리즘은 탐색을 받은 노드는 대기중인 작업의 수가 임계치보다 크면 탐색 시도한 노드에 작업을 전송하고, 그렇지 않으면 부정적인 응답을 나타내는 정보와 부하 상태를 실은 응답패킷을 탐색 시도한 노드에 전송한다.

4. 시뮬레이션

4.1 가정

본 연구에서 제시한 주기적 부하균형 알고리즘의 성능을 평가하기 위하여 〈표 3〉과 같이 가정한 후, Pentium PC 상에서 SMPL/PC 함수와 C언어로 프로그램을 작성하여 시뮬레이션을 수행하였다[9][10][7].

〈표 3〉 시뮬레이션 파라메타 값
〈Table 3〉 Simulation parameters values

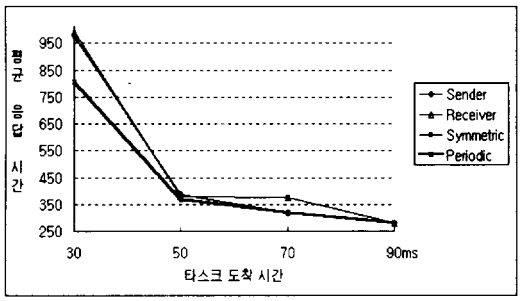
- 노드들은 동질적이다.
- 원격 작업에 대한 로컬 작업의 우선 순위는 존재하지 않는다.
- 작업 도착 및 서비스 시간은 지수분포를 따른다.
- 작업들은 FCFS 로컬 스케줄링을 기본으로 하여 실행된다.
- 시뮬레이션 시간 : 10000.0ms
- 노드 수 : 6 ~ 12
- 평균 작업 서비스 시간(Ts) : 200ms
- 타스크 도착 시간 : 10 ~ 90ms
- 전송 지연 시간 : 10ms
- 부하균형 비용 : 20ms
- 임계값 범위 : 하한치 2 ~ 5
  상한치 3 ~ 7

4.2 성능분석

4.2.1 알고리즘별 분석

본 절에서는 아래 (그림 11)에서 보인 것처럼 타스크 도착 시간을 변경시켰을 경우 각 알고리즘과 제안한 주기적 알고리즘을 타스크들의 평균 응답시간 변화 추이로 살펴보았다. 본 실험에서는 성능평가 파라메타중 하한치 3 과 상한치 5로 고정하였으며, Periodic 알고리즘 평가를 위해 필요한 수행주기로 상수 7을 사용하였다.

실험결과 본 연구에서 제시한 주기적 알고리즘은 전반적인 타스크 도착율에서 다른 알고리즘보다 빠른 응



(그림 11) 타스크 도착시간별 평균 응답시간
(Fig. 11) Mean response times under different task arrival time

답시간을 보였다. 기존의 연구에서는 대칭(Symmetric)이 타스크 도착율의 변화에 따라 적응적으로 송신자와 수신자 중 하나가 적용되어 훌륭한 성능을 보였지만 본 연구에서는 송신자와 수신자의 응답시간과 비교해볼 때 거의 같은 성능을 보이고 있다. 이러한 결과는 기존의 연구가 이들 알고리즘을 비교 평가할 때 부하균형 알고리즘 수행 자체에 대한 비용은 무시한데서 기인한다. 하지만, 실제 환경에서는 시스템이 과부하일 때 부하균형 알고리즘을 수행함으로써 노드 자신을 더욱 과부하 상태로 만드는 오류를 가져올 수 있다.

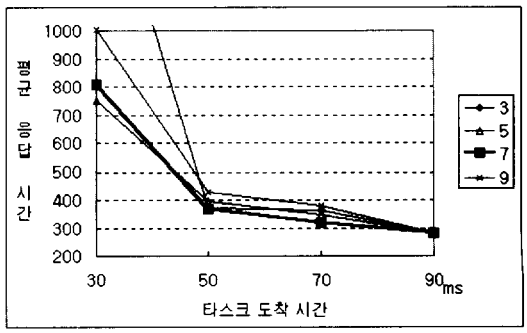
본 연구에서는 이들 부하균형 알고리즘 세션(session)에 일정한 비용을 부과하였다. 이로써, 주기적 부하균형 알고리즘은 가장 낮은 평균 타스크 응답 시간을 갖는다. 이는 다른 알고리즘보다 적은 수의 메시지 교환과 타스크의 이주에서 비롯된 결과이다. 주기적 부하균형 알고리즘의 이점은 원격 노드들에 대해 단 한번의 주기적인 탐색에 있다. 타스크 도착 시간이 90ms인 경우 모든 알고리즘은 전체 노드가 idle할 확률이 크기 때문에 이는 부하균형을 행하는 경우는 없고 모든 타스크가 로컬로 처리됨을 나타낸 것이다.

부하가 증가함에 따라 평균 응답 시간의 변화 폭에 대한 증가가 빠르게 상승하고 있는 원인은 적합한 노드를 선정함을 무분별한 탐색으로 인해 자신의 부하 상태를 유지하지 못하고 변화됨으로써 처리 능력이 감소되어지는 결과에서 비롯된다.

본 논문에서는 탐색주기를 미리 선정하고, 탐색 주기 내 최초로 발견된 노드를 선정하여 작업을 전송하여 처리하는 최초 발견 방식을 채택함으로써 탐색에 따르는 지연 시간을 최대한으로 줄일 수 있다.

4.2.2 주기별 분석

본 절은 주기적 부하균형 알고리즘이 사용하게될 주기에 대하여 살펴보았다. 본 실험에 사용된 고정된 입력 값들로는 하한 임계치 3, 상한 임계치 5로 주어졌으며 노드의 개수는 6 이다.



(그림 12) 부하균형 주기별 평균응답시간
(Fig. 12) Mean response time under load balancing periods.

그림에서 볼 수 있듯이, 타스크 도착시간 전체적으로 볼 때 주기가 7 인 경우 최적의 평균응답시간을 보이고 있다. 타스크 도착시간이 90ms인 경우는 앞서도 언급했듯이 부하균형이 불필요한 경우이기 때문에 모든 주기에서 동일한 성능을 보이고 있다.

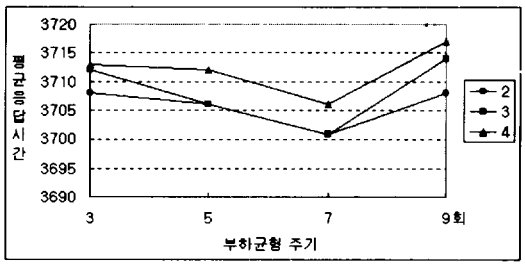
실험을 통하여 알 수 있었던 것은 주기(period)는 임계치와 시스템에 도착하는 타스크 도착율에 따라 크게 달리 결정된다는 사실이다. 이는 사용자가 자신이 사용하는 시스템의 특성을 미리 분석할 필요가 있으며 임계치와 잘 조화를 이루어서 주의 깊게 결정해야함을 의미한다.

4.2.3 임계치별 분석

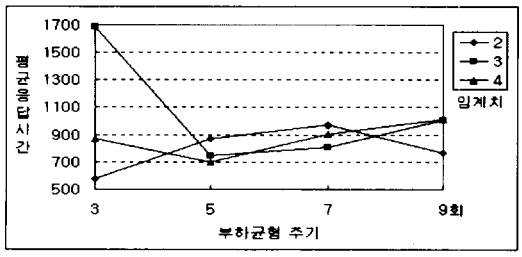
본 절은 임계치별 수행 주기율을 분석함으로써 시스템 관리자 또는 일반 사용자가 시스템을 분석하는데 도움을 주자는 데 있다. 본 실험에 이용된 노드수는 6개이며, (그림 13)과 (그림 14)는 타스크 도착시간을 T_a=10 과 30으로 각각 고정했을 때 평균응답시간 변화를 보이고 있다.

(그림 13)의 그래프를 보면, 하한치의 변화에 따라 최적의 성능을 얻기 위해서 취해야 할 부하균형 주기를 알 수 있다. 여기서는, 모든 작업 이주 임계치에서 주

기를 7 로 설정하면 최적의 타스크 응답시간을 가질 수 있다.



(그림 13) 임계치별 평균응답시간(Ta=10)
(Fig. 13) Mean response time under different low threshold (with Ta=10)



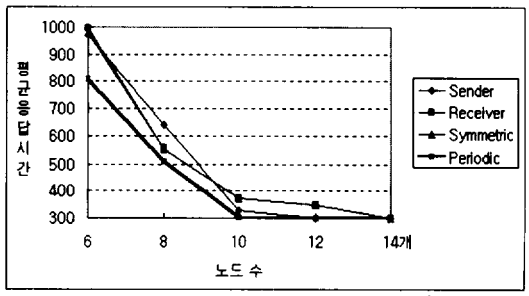
(그림 14) 임계치별 평균응답시간(Ta=30)
(Fig. 14) Mean response time under different low threshold (with Ta=30)

그에 반해 (그림 14)를 보면 타스크 도착율이 좀 더 낮아짐에 따라 임계치가 3과 4인 경우 주기를 5로 설정하면 되지만, 임계치가 2인 경우는 주기를 3으로 달리 설정해야함을 알 수 있다. 이러한 결과는 각 노드에 도착하는 타스크 비율이 낮아짐에 따라 주기를 더욱 낮게 할당함으로서 부하균형으로 인한 이득을 얻을 수 있다는 결론이다.

4.2.4 노드별 분석

본 절의 실험은 전체 부하균형 클러스터에 속한 노드의 개수 변화에 따라 비교대상 알고리즘과 주기적 알고리즘의 성능이 어떠한 변화를 보이는가를 살펴보았다. 본 실험에서는 하한치 3, 상한치 5, 타스크 도착율 T_a=30 으로 고정하였을 때 노드의 수를 변경하며 평가하였다.

본 실험에서도 역시 노드의 수 변동에 적응적으로 주기적 알고리즘이 다른 알고리즘에 비해 빠른 응답시간을 보이고 있음을 알 수 있었다. 한가지 특징은 노드의 수가 점차 증가할수록 모든 알고리즘이 거의 비슷한 성능을 보이고 있다는 것인데, 이는 노드의 수 증가는 전체적으로 각 노드에 부과되는 부하가 줄어들기 때문에 결국 부하균형을 해야할 필요성이 감소함을 의미하는 것으로 분석된다.



(그림 15) 노드별 평균응답시간
(Fig. 15) Mean response time under the different number of nodes.

5. 결 론

본 논문은 주기성을 이용한 제한된 부하균형 알고리즘을 제시하였다. 기존의 연구는 시스템 상태의 변화에 따른 과부하 또는 저부하 노드가 부하균형을 주도하였다. 때로는 이들을 혼형한 대칭적인 기법을 이용하기도 했다. 기존의 연구결과 과부하 또는 저부하 노드만이 주도하는 부하균형보다는 대칭적인 기법을 사용하는 경우 전체 시스템 부하 변동이 심한 상태에서 작업들의 평균 응답시간이 빠름을 볼 수 있었으며, 시스템의 안정성 측면에서도 탁월한 성능을 발휘함을 알 수 있다.

하지만, 기존의 대칭 알고리즘은 몇 가지 단점을 가지고 있다. 이것의 문제는 부하상태의 변동으로 인해 부하균형 필요시 Random하게 탐색 대상 노드를 선정하는데 있다. 이러한 노드 선정 기법은 이종성이 증가하는 시스템에서는 적절한 노드 선정에 실패할 확률이 높으며 계속적인 탐색에 의한 과도한 메시지 처리는 그렇지 않아도 과부하 상태인 노드에 불필요하게 많은 탐색 메시지가 도착함으로써 더욱 과부하로 만들 수 있다. 이는 전체 시스템을 불안정 상태에 빠져드는 현상을 초래한다. 본 논문에서는 이를 보완한 기법으로, 각 노드에 상태벡터를 두어 부하 균형 메시지 교환시 부하

정보와 노드의 ID를 최신의 상태로 갱신함으로써 다음 번 수행 주기시 적절한 대상 노드를 미리 예측하여 탐색할 수 있다.

향후 연구과제로서는 제안한 알고리즘을 수행 주기가 시작될 당시에 고정적으로 설정하는 것이 아니라 각 노드의 작업 도착율의 변동에 비례하여 적응적으로 변경될 수 있도록 본 알고리즘을 확장하는 연구가 필요하다.

참 고 문 헌

[1] P. V. McGregor and R. R. Boorstyn, "Optimal Load Balancing in a Computer Network", Proc. 1975 Int. Conf. on Commun., Vol.3, IEEE, New York, pp.14-19, 1975.

[2] F. C.H. Lin and R. M. Keller, "The Gradient Model Load Balancing Method", IEEE Trans. Software Engineering, Vol.SE-13, pp.32-38, Jan., 1987.

[3] W. E. Leland and T. J. Ott, "Load-balancing Heuristics and Process Behavior", Proc. of the ACM SIGMETRICS Conference, pp.54-69, May, 1986.

[4] J. A. Stankovic, "Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms.", Computer Networks 8, pp.199-217, 1984.

[5] L.M. Ni, C.-W. Xu, and T.B Gendreau, "A Distributed Drafting Algorithm for Load Balancing" IEEE Trans. Software Engineering, Vol.SE-11, pp.1153-1161, Oct. 1985.

[6] 임종규, "결함노드의 부하재분배를 위한 부하재분배기 모델 연구", 수원대학교, 석사학위논문, 1994.

[7] 김명희, 백두권 편저, 컴퓨터시스템의 시뮬레이션, 생능출판사, 1995.

[8] D.L.Eager, E.D. Lazowska, and J.Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", IEEE Trans. Software Engineering, Vol.SE-12, May, 1986.

[9] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing ", Performance Evaluation 6, pp.53-68, 1986.

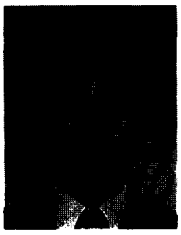
[10] P. M. Dew, and M. Kara, "A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems", School of Computer Studies, University of Leeds, pp.616-623, 1994.

[11] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancig", IEEE Trans. Software Engineering, Vol.14, No.9, pp.1327-1341, Sept. 1988.

[12] K.Efe and Groselj, "Minimizing Control Overheads in Adaptive Load Sharing.", Proc. of the 9th Int. Conf. on Distributed Computing Systems, Newport Beach, California, pp. 307-315, Jun. 1989.

[13] N. G. Shivaratri and P.Krueger, "Two Adaptive Location Policies for Global Scheduling Algorithms", Proc. of the 10th Int. Conf. on Distributed Computing Systems, pp.502-509, 1990.

[14] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems", ACM Performance Evaluation Review, Vol.11, No.1, pp.47-55, Spring 1982.



임 종 규

1989년 2월 목원대학교 컴퓨터공학과 졸업(학사)

1994년 2월 수원대학교 전자계산학과 졸업(석사)

1997년 3월 수원대학교 전자계산학과 재학중(박사)

1997년 3월~현재 수원대학교 자연과학연구소 객원연구원

관심분야 : 개방형 분산 시스템, 멀티미디어 시스템, 결함허용시스템

박 한 규

1994년 2월 수원대학교 전자계산학과 졸업(학사)
1997년 8월 수원대학교 전자계산학과 졸업(석사)
1998년 3월~현재 수원대학교 전자계산학과 재학중(박사)
관심분야 : 분산운영체제, 멀티미디어 시스템

장 순 주

1989년 2월 대전산업대학교 전자계산학과 졸업(학사)
1995년 8월 수원대학교 전자계산학과 졸업(석사)
1997년 3월 수원대학교 전자계산학과 재학중(박사)
1997년 3월~현재 수원대학교 자연과학연구소 객원연구원
관심분야 : 분산운영체제, 멀티미디어 시스템, 컴퓨터네트워크

구 용 완

1976년 2월 중앙대학교 전자계산학과 졸업(학사)
1980년 2월 중앙대학교 전자계산학과 졸업(석사)
1988년 2월 중앙대학교 전자계산학과 졸업(박사)
1983년 3월~현재 수원대학교 전자계산학과 교수 및 전자계산소장
관심분야 : 분산운영체제, 시스템 네트워크 관리, 멀티미디어 시스템등