이종 분산 시스템을 위한 동적 부하균등 알고리즘기법

The Dynamic load Balancing Algorithm Method for Heterogeneous Distributed Systems

장 순 주*
Soon-Ju Chang

요 약

본 논문은 이종 분산시스템을 위한 동적 부하균등 알고리즘을 제안하였다. 본 알고리즘은 타스크들이 재배치할 수 있게 한다. 알고리즘의 핵심은 송신자로부터 수신자에게로 적절한 처리 요구량을 전송하는데 있다. 본 전송량은 동적으로 송신자와 수신자간 협상과정에서 결정된다. 이러한 전송량이 결정될 때 고려되는 요소들은 다른 노드의 처리속도, 현재 송신자와 수신자간의 부하상태, 재배치하기에 적절한 처리요구 등이다.

본 연구에서는 또한 이종시스템을 위해 특별히 설계된 부하상태 전략도 제안하였다. 연구결과는 제안한 알고리즘이 현존하는 알고리즘에 비해 뛰어난 성능을 보였고 시스템 속성이 다양한 측면에서 안정적이다.

Abstract

This paper propose an dynamic load balancing algorithm for heterogeneous distributed systems. The algorithm allows this tasks to be relocated. The key of the algorithm is to transfer a suitable amount of processing demand from senders to receivers. This amount is determined dynamically during sender-receiver negotiations. Factors considered when this amount is determined include processing speeds of different nodes, the current load state of both sender and receiver, and the processing demands of tasks eligible for relocation. This paper also propose a load state measurement scheme which is designed particularly for heterogeneous systems.

This results of the study show that the proposed algorithm outperforms the existing algorithms and is stable over a range of system attributes.

☞ 키워드 : Dynamic load Balancing, Distributed Systems, task

1. 서 론

분산시스템(Distributed Systems)은 통신망에 의해 상호 연결되어진 자율적인 컴퓨터들의 집합이다. 통신망을 통해 시스템의 자원들은 다른 곳에 위치한 사용자에 의해 공유될 수 있기 때문에, 이들 자원들의 효율적인 활용이라는 측면에서 볼 때, 전체 시스템의 부하 균등(Load Balancing)을 이루는 것은 분산시스템에 가장 중요한 현안들 중 하나가 되었다.

본 논문에서 제시한 대칭적(Symmetric)부하균등 알고리즘은 과부하 노드가 타스크를 전송하기를 원할 때 저 부하 노드를 찾을 뿐 아니라 저 부하 상태인 수신 노드가 몇 개의 타스크를 과부하 상태인 노드로부터 위임받아 관리한다. 적절한 노드의 선정은 각 부하균등 polling 메시지에 실어보내는 보증 값(gur)과 예상 값(res)에 따른다.

본 연구의 목표는 커다란 overhead를 유발함 없이 무분별한 탐색으로 시스템이 불안정상태에 빠지는 것을 방지하는데 있다. 여기서 제시한 알고리즘은 전체 시스템에 연결되어있는 각 노드들의 최근 부하 상태정보를 유지(sender state, receiver state, neither)키 위해 탐색 과정에서 수집되는 정보를 각 노드가 저장하며 노드 탐색 시 이를 이용한다.

* 정 회 원 : 한신대학교 교육대학원 강사
sjcchang1@hanmail.net (제1저자)

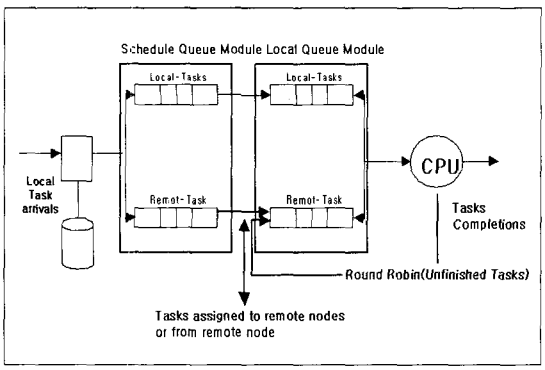
본 논문에서는 다른 서비스 요구시간을 갖는 작업들과 다른 처리 능력을 갖는 노드를 이종(Heterogeneous)으로 정의하였으며 이를 위한 대칭적 동적부하균등 알고리즘을 설계하였으며 시뮬레이션을 통하여 성능을 분석하였다. 성능평가결과 제시한 알고리즘은 송신자와 수신자 주도형의 혼합형태인 대칭형으로 설계되었기 때문에 한쪽 노드만이 주도되는 일방 주도형 알고리즘보다 여러 시스템 부하상태에서, 특히 노드의 특성이 다르고 부하 도착 분포가 다른 이종 상태에서는 탁월한 성능을 보임을 보았다.

2. 이종시스템 기술

2.1 시스템 모델 설계

본 논문에서는 그림 1에서 보인 것 같은 시스템 모델을 갖는 노드가 분산시스템 형태를 이루고 있다.

상기 시스템 모듈에서, 각 노드는 2개의 큐 모듈과 1개의 타스크 해독기, 1개의 타스크 리스트로 구성된다. 스케줄 큐 모듈(SQM ; Schedule Queue Module)은 무한한 용량의 큐를 갖으며, 로컬 큐 모듈(LQM ; Local Queue Module)은 Qi로서 정의된 서비스 큐 용량을 갖는다. 로컬 타스크가 도달했을 때, 타스크는 타스크 할당을 위한 후보가 되기 위해 스케줄 큐 모듈로 들어간다. 이때, 타스크



(그림 1) 이종 시스템 모델

해독기는 타스크 리스트를 이용하여 원격으로 처리 가능한 타스크인 경우 원격 타스크 큐에 할당하고, 로컬로 처리하기 적당한 타스크인 경우 로컬 타스크 큐에 할당한다.

타스크가 로컬 SQM에서 또는 원격 노드로부터 이주된 것이든 간에 일단 로컬 큐에 들어갔다면, 그 타스크는 또 다른 노드에게 재할당될 수 없다. 이러한 전략은 무분별한 탐색(probing)에 의한 스레싱(thrashing)을 방지하기 위함이다. 로컬 큐 모듈에 있는 타스크들은 다중 레벨(multi-level) 큐잉 기법을 사용하여 스케쥴된다. 이 기법은 로컬타스크 큐에 어떠한 타스크도 없는 경우만이 원격타스크 큐에 큐잉되어 있는 타스크를 RR 전략으로 처리한다.

2.2 타스크 특성 분석

본 논문에서 제시한 부하균등 알고리즘의 성능평가를 위하여 실제 시스템 환경에서 자주 사용되고 있는 작업부하를 분류할 필요가 있다. 이러한 분류는 제시한 시스템 모델에서 살펴보았듯이 각 노드에 도착하는 작업부하는 타스크 리스트를 참조하여 타스크 해독기가 LQM내의 로컬이나 원격 큐에 적절히 큐잉되도록 하기 위함이다. 따라서, 다음 표 1은 UNIX SVR4상에서 자주 사용되는 명령어들을 추출하였고, 표 1의 원격란에서의 Y:Yes, N:NO 로 표기하였으며, 이들의 수행 특성에 따라 분류한 것이다.

3. 대칭 부하균등 기법

3.1 이주 결정 규칙 정의

송신자측에서 보증 값 gur를 유도하는 전략은 알고리즘 성능을 크게 좌우한다. 즉, 매우 큰 gur은 송신자가 전송하기로 한 하나의 노드를 제외하고 다른 잠정적인 수신자에게 전송하는 것을 막는다. 그러므로 작업부하 분산양은 제한될 것이다.

(표 1) 명령어 특성

명령어	원격	기능	명령어	원격	기능
hostname	N	host information	du	N	disk usage
cat	N	view a file	f77	Y	Fortran Compiler
gcc	Y	Gnu C compiler	lint	Y	C program checker
cp	N	file coping	grep	Y	text pattern search
date	N	current time	finger	N	user information
df	N	file system usage	users	N	list of current users
lp	Y	printer queue check			

반대로, 적은 gur은 단일배치에서 작업 부하 전송량을 제한함으로써, 배치 할당의 장점을 유도하게 된다. gur을 유도하기 위한 기법은 다음과 같다.

$$gur(s) = \min\left[\begin{array}{c}(T_{up} - T_{low}) \cdot \mu_r^5 \\ W_s^{TQ}\end{array}\right] \qquad (식\ 1)$$

윗첨자(s)는 송신자 노드 P_s를, 아래첨자(r)은 수신자 노드 P_r를 의미한다. 식 1의 유효성을 증명키 위한 첫 번째 규칙은 다음과 같다.

◦ [규칙 1] : 저부하 노드 P_r은 과부하 노드로부터 수신한 서비스 요구 $\alpha^{(r)}$만큼 타스크 이주가 이루어진 후 수신자 노드P_r의 상태가 과부하로 되지 않아야 한다.
($VW^r + \alpha^{(r)} \rightarrow$ H - load)

3단계 부하 측정 기법에 따라, 규칙 1은 다음과 같이 유도할 수 있다.

$$\begin{array}{l}(VW_R + \alpha^{(r)}) \le T_{up} \\ \alpha(r) \le (T_{up} - VW_r)\end{array} \qquad (식\ 2)$$

수신노드의 가상부하는 0~T_{low} 사이의 값이어야 한다. 따라서 VW_r의 두 가지 한계치(bound)는 다음과 같다.

$$\begin{array}{l}Minimum: VW_r = 0 \\ Maxinum: VW_r = T_{low}\end{array}$$

따라서, 다음과 같이 두 가지 한계조건(bound condition)을 갖는다.

$$\alpha^{(r)} \le \left[\begin{array}{c}T_{up} \text{ if } VW_r = 0 \\ (T_{up} - T_{low}) \text{ if } VW_r = T_{low}\end{array}\right] \qquad (식\ 3)$$

식 3에서 $VW_r = T_{low}$ 인 경우를 보면, 송신자 P_s는 수신자 P_r이 최대로 받아들일 수 있는 작업 부하의 상한치는 (T_{up} - T_{low})라는 것을 예상할 수 있다.

이것은 송신자 노드의 CPU 시간으로 표현했을 때 (T_{up} - T_{low}) · μ_r^s로 나타낼 수 있다. 그러므로, gur(s)은 (T_{up} - T_{low}) · μ_r^s보다 크지 않도록 해야 한다. 즉, 식 1의 첫 번째 조건을 설명한 것이다. 식 1의 두 번째 조건은 송신자 P_s 의 큐 크기보다 많은 양의 타스크를 보증할 수 없다는 것이다.

gur(s)가 도달했을 때 수신자 P_r은 일치하는 예약값 Res를 유도해야만 한다. 규칙 1과 실제 송신자 노드로부터 이주를 시작한 타스크가 수신되기 전에 로컬로 새로운 타스크가 도착할 가능성이 없다고 보았을 때 수신 가능한 최대 값은

$$\alpha^{(r)} = T_{up} - VW_r \qquad (식\ 4)$$

명백히, P_r은 송신자가 보증한 gur보다 더 많은 양으로 자신의 처리 능력을 예약(res)하지는 않을 것이다. 그러므로, 다음과 같은 식을 얻을 수 있다.

$$res^{(r)} = \min\left[\begin{array}{c}\alpha^{(r)} = T_{up} - VW_r \\ gur^{(s)} - \mu_r^s\end{array}\right] \qquad (식\ 5)$$

하지만, 식 1에 따라 $gur^{(s)}$의 최대값은 P_r의 CPU 시간으로 측정할 때 $[(T_{up} - T_{low}) \cdot \mu_r^s]$ 또는 $(T_{up} - T_{low})$이다. 이에 반해, 수신자 P_r에 대한 VW_r의 최대값은 T_{low}이며 $\alpha^{(r)}$의 최소값은 $(T_{up} - T_{low})$이다. 따라서, $gur^{(s)}$는 최소한 항상 $\alpha^{(r)}$만큼 가지고 있다. 그러므로, 식 5는 $res^{(r)} = gur^{(s)} \cdot \mu_r^s$로써 간략히 될 수 있다. 식 4에 의하면 수신자-주도형 polling인 경우, $gur^{(s)}$는 간단히 이용할 수 없으므로 식 $res^{(r)} = (T_{up} - VW_r)$ 을 얻게 된다.

따라서,

$$res^{(r)} = \begin{bmatrix} gur(s) - \mu_s^r & \text{if } sender \\ T_{up} - VW_r & \text{if } receiver \end{bmatrix} \quad \text{(식 6)}$$

타스크 할당 크기 계산의 다음 단계는 $res^{(r)}$에 의해 부과된 제약에 따라 송신자 P_s는 수신자 노드 P_r에게 전송되기 적당한 최대 타스크 서비스 양을 판단하는 것이다. 이 양을 최대 허용 타스크 할당이라 부르며 θ로 표시한다. θ을 유도하기 위해 규칙 2와 규칙 3을 제시하였다.

- [규칙 2] : 이주시킨 타스크의 CPU 시간 $\beta^{(s)}$를 제거할 경우 송신자 노드 P_s를 L-load 상태로 변화시켜서는 안되어야 할 것이다.
  ($VW_r + \beta^{(s)} \rightarrow$ L - load)
  규칙 2는 부하의 3상태에 의거해서 ($VW_s + \beta^{(s)} T_{low}$)가 된다.

$$\beta^{(s)} < VW_s - T_{low} \quad \text{(식 7)}$$

전체 CPU 시간이 $\beta^{(s)}$인 타스크를 이주한 결과 P_s가 P_r보다 낮은 가상 부하상태로 변경되어서는 안 된다는 사실이다.

- [규칙 3] : 타스크 할당 크기의 재배치는 P_s에서 결과가 P_r 보다 더 낮은 가상노드를 갖지 않는 전체 CPU 시간 $\beta^{(s)}$를 갖는다.

$$\begin{aligned} VW_r - \beta^{(s)} &\geq VW_r' + \beta^{(s)} \\ \mu_s^r \beta^{(s)} &\leq \frac{VW_s - VW_r'}{(1+\mu_s)} \end{aligned} \quad \text{(식 8)}$$

여기서 VW_r'는 P_s의 Polling 메시지에 따른 응답으로 수신자 P_r에서 송신자 P_s에게로 전송되는 ACK 메시지에 piggyback 되는 P_r의 가상 부하이다.

부등식 식 7의 조건을 완화하고 식 8을 만족하는 최대 가능 값을 얻음으로써, 식 9를 유도할 수 있다.

$$gur^{(s)} = \min \begin{bmatrix} VW_s - T_{low} \\ \dfrac{VW_s - VW_r'}{(1+\mu_s^r)} \end{bmatrix} \quad \text{(식 9)}$$

송신자 P_s는 수신자가 예약한 작업부하 보다 더 많은 것을 전송할 수 없으므로 다음과 같은 식을 얻을 수 있다.

$$\theta = \mu_{T(P_r)} \min \left[\frac{\beta^{(s)}}{res^{(r)}} \cdot \mu_r^s \right] \quad \text{(식 10)}$$

식 6으로부터 $res^{(r)}$이 최대로 가질 수 있는 값은 $VW_r = 0$을 갖는 수신자 주도형 T_{up}이다. 식 10에 따라 θ의 최대값은 $T_{up} \cdot \mu_{T(P_r)}$이다.

3.2 알고리즘 기술

각 노드에서는 부하 균등 클러스터에 있는 모든 노드의 부하 상태 정보를 관리하기 위해 구조체 자료형을 다음과 같이 사용한다. 노드에서 관리되는 정보는 3개의 순서리스트로 구성되어 있다.

- Slisti : 전송된 타스크들의 잠재적 송신자로서 노드의 ID를 저장
- Rlisti : 잠재적인 수신자로서 판명된 노드의 ID 저장
- OKi : 송신자, 수신자가 아닌 노드리스트

본 알고리즘의 기본 개념은 송신자로부터 그 송신자의 목적지(수신지)에게로 폴링 메시지가 gur과 함께 piggyback 된다는 것이다. 폴링 메시지가 송신된 이후, 송신자의 작업부하는 인위적으로 gur의 양만큼 줄어들 것이다. 이러한 방법은 송신자가 gur의 양만큼 작업부하를 이미 예정된 수신자에게 이주시켜 처리를 의뢰할 것이므로 송신자 자신은 가상적으로 부하를 그만큼 줄일 수 있게 된다.

3.2.1 타스크 전송 모듈

본 모듈은 알고리즘의 주(main)부분에 해당하는 타스크 전송 모듈로서 자신의 3가지 시스템 부하 상태에 따라 적절한 부 함수를 호출하여 부하균등을 시도한다. 부하균등은 시스템의 부하 상태가 변경될 때에만 행한다.

```
Task_Transfer_Module( )
   evaluation( );
   while(load_state_change)
   {
    if( VWr > Tup) then
{ call Receive_Initiated_LIM( );
  call RES( ); }
     else if( VWR > Tup) then {
call Sender_Initiated_LIM( );
call RES( );
                                  }
  }
}
```

3.2.2 송신자 주도 위치 정보 모듈

송신자가 주도하는 협상에서는 송신자가 작업을 받을 수신자를 찾는다. 이때, 타스크 전송 모듈과 송신자 노드는 다음과 같은 과정을 거친다.

```
Sender_Initiated_LIM( )
{
  iter=0, j=Rlist[head];
  do {
        iter++;  /*probe iteration*/
        PROBE(Rlist[j]);
        if(RECEIVER_OK)
          then {
                return(ID, GR);
                stop;
             }
          else {
                Remove(Rlist[j]);
                if( SENDER_OK)
                then Append(Slist[j], IDj);
                else Append(OKlist[j], IDj);
             }
 j++
      } while(Probelimit > iter  ||  Rlist
                ≠ Ø  ||  No longer sender)
stop
return(FAILURE)
}
Sender_Initiated_LIM( )
{
 receive the probe message from a Heavily node( NODEi
)
 evaluation( );
 reply its load_state message to NODEi ;
 }
```

3.2.3 수신자 주도 위치정보 모듈 $VW_r > T_{up}$

수신자가 주도한 협상과정에서는 작업을 전송할 송신자를 찾기 위해 수신자에서 탐색을 시작한다. 이러한 수행을 위해, 수신자 노드에서 위치정책은 다음과 같은 과정을 거친다.

```
Receive_Initiated_LIM( ) {
iter = 0, j = Slist[head];
do {
    iter++;
    IF(Slist)
    then probe Slist[j];
    else {
     if(OKlist) then
     PROBE(Rlist[last]);
          else
     PROBE(Rlist[last]);
        }
    if(SENDER_OK && will remain a sender even after
                                  migrating a task)
          then {
               Append(Slist[head], j);
               return(ID, GR);
               stop;
```

```
            }
        else {
            if (SENDER_OK && will no longer be a
                    sender after migrating a task)
              then {
                    Remove j from
                    {Slist[] or Rlist[] or OKlist[];}
                    Append(Oklist[head], j);
                    return(ID, GR);
                                        }
                                } /* else */
            if(RECEIVER_OK)
                then {
                    Remove j from
                    {Slist[] or Rlist[] or OKlist[];}
                    Append(Rlist[head], j);
                    } j++; /* Probe next node */
            } while(No longer receiver state ||
              Probelimit > iter || iter <=   all nodes)
stop;
return(FAILURE);} /* End of MAin */
RECEIVER_Initiated_LIM( ) {
  receive the request message from nodei;
  evaluation( );
 if (!SENDER_OK)
then {
     Informing i of j's
     state(receive or OK);
     Remove j from { Slist[ ] or Rlist[ ] or OKlist[ ] ;}
}
   else {
          migrate a task to node I;
          inform I of j's state after the migration;
       }
stop;
}
```

4. 평 가

본 절에서 행해진 실험은 Sun 워크스테이션 상에서 SMPL 시뮬레이션 tool을 사용하여 프로그래밍을 행하였다.

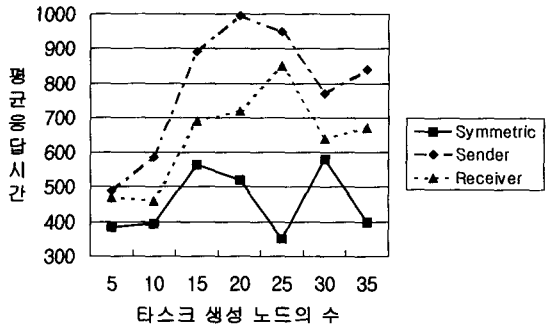
4.1 가정

다음 표 2는 알고리즘 평가에 사용된 파라메타 값이다.

(표 2) 시뮬레이션 파라메타 값

Parameter	Value	Parameter	Value
$Q_s \forall_i \in M$	30	Nodes	40
T_a	5	T_s	200
T_{low}	2	T_{up}	4

· 타스크 도착률 : 지수분포
· 시뮬레이션 시간 : 30000 msec



(그림 2) 이종 노드 개수에 따른 평균응답시간 (Ta=5)

4.2 알고리즘 분석

적절한 노드를 찾는 탐색 메시지는 무시할 수 없을 정도의 CPU overhead이다. 분산시스템에서 타스크를 전송하는데 소모되는 비용이 원격으로 처리함으로써 얻는 이득을 초과한다면 불안정성을 유발할 수 도 있다. 따라서, WAN과 같은 환경에서는 CPUtask, DELAYtask와 같은 overhead로 인해 적합치 못하며 LAN처럼 단일 건물이나 인근 건물간에 적용할 만 하다. 본 논문에서는 높은 협의 비용으로 인해 유발되는 불안정성을 피하자는 데 또 하나의 목적이 있다. 여기서 사용된 비교대상 알고리즘은 송신자 주도형(SEND)과 수신자 주도형 (RECV), 그리고 본 논문에서 제안한 대칭적 알고리즘(SYM)이다.

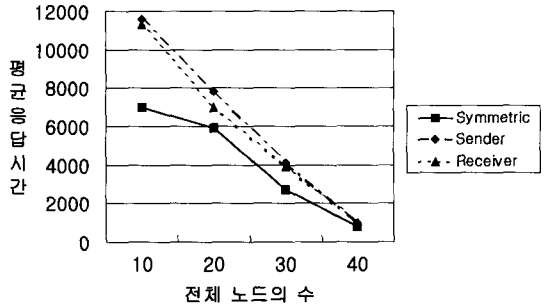
그림 2는 각 노드에 도착하는 작업 부하의 빈도가 다른 경우 어떠한 특성을 보이는가를 실험하였다. 이를 이종 프로세스 도착율이라고 한다. 본 실험에서 이종 프로세스 도착율은 전체 노드의 개수를 40으로 놓고 부하가 발생하는 노드개

수를 x 축에 나타내었다. 이때 시스템 타스크 도착율 Ta를 5ms로 고정시켰기 때문에 부하가 발생하는 노드는 자동으로 과부하가 되는 셈이다. 나머지 노드는 전혀 부하가 발생하지 않는 것으로 가정하였다. 이러한 실험은 작업 도착율이 매우 다른 이종 상황에서 시스템이 불안정성에 빠지는가를 알아보기 위해 중요하다.

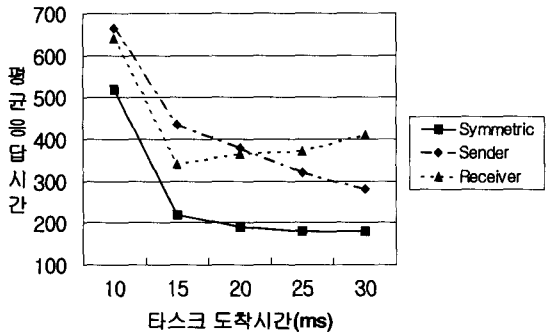
RECV는 다른 알고리즘과 비교해 볼 때 과부하 타스크가 많아질수록 평균응답시간이 나빠지다가 25개를 넘어서면서 부터는 점차 안정된 상태를 보이고 있다. 이러한 현상은 과부하 노드를 찾을 확률이 높아지므로 안정상태를 보인 것으로 분석되었다. RECV의 협상은 타스크가 송신자(과부하) 노드에 도착하면 시작되는 것이 아니라 큐에 계속 적재되어 과부하가 된 상태에서 시작되기 때문에 지연된 타스크 전송을 발생시킨다. 이는 송신자의 큐 길이를 감소시키는 효과는 볼 수 없을 것이다. SEND도 RECV와 마찬가지로 불안정성에 빠지는데, 특히 전체노드 중 정확히 절반인 20개의 노드가 과부하일 경우 가장 나쁜 성능을 보이며, 이를 정점으로 하여 점차 개선되고 있다. 이는 수신자를 찾을 확률이 그만큼 적어지기 때문에 발생한다. SYM인 경우, 모든 이종성에 있어 안정상태를 유지한다. 그 이유는, SYM 알고리즘이 자체에서 Slist, Rlist, OKlist 자료구조를 유지하기 때문에 낮은 이종성에서는 Rlist를 이용하여 수신자 노드를 찾고 높은 이종성에서는 Slist에서 송신자 노드를 쉽게 찾을 수 있기 때문이다.

그림 3은 시스템 부하를 고정시킨 상태에서 평균응답시간이 부하균등에 참여하는 노드개수의 변화에 따라 어떠한 영향을 받는가를 보았다. SYM의 그래프를 보면, 평균응답시간이 가장 좋음을 보이고 있다.

노드의 수가 적을 경우 다른 알고리즘보다 SYM이 월등한 성능을 보였으나 점차 노드의 수가 증가할수록 이들의 격차는 줄어든다. 이것은 전체 시스템에 도착하는 부하량이 고정된 상태였기 때문에 전체적으로 부하량은 감소한 효과를 갖기



(그림 3) 부하균형 노드의 수에 따른 평균응답시간 (Ta=5)



(그림 4) 시스템 부하에 따른 평균응답시간 (노드수 = 20)

때문이다. 만약, 노드의 수 증가에 비례하여 타스크 도착시간도 더욱 빠르게 하였다면 노드가 10개일 경우의 성능 격차를 계속 유지하였을 것이다.

그림 4는 시스템 부하의 변동에 따른 평균 타스크 응답시간의 관계를 그래프로 보인 것이다.

(1) 저부하 상태인 경우

부하균등 클러스터에 속한 노드에 작업들이 전체적으로 균등하게 낮은 비율로 도착함에 따라, receiver는 작업을 전송 받을 송신자를 찾을 확률은 낮다. SEND 알고리즘은 자신에 작업이 도착하면 저부하 상태일 확률이 높기 때문에 다른 노드에게 polling할 필요 없이 즉시 처리하지만, RECV는 저부하일 확률이 높기 때문에 과부하를 찾기 위한 불필요한 polling으로 인해 다른 알고리즘 보다 떨어진 평균 응답시간을 보이고 있다. 또한, SYM도 일반적으로 송신자를 찾는데 실패

할 것이다. 하지만, 다른 노드상에서 유지되는 Rlist를 최신상태로 갱신하는 긍정적인 효과를 갖는다.

이러한 polling 실패는 성능에 불리하게 작용되지 않는다. 그 이유는, polling을 시작한 노드는 저부하 노드이므로 CPU capacity측면에서 충분히 여유가 있을 것이며, polling으로 인한 불안정성에 빠질 위험은 전혀 없기 때문이다. Rlist를 갱신하는 긍정적인 효과란 시스템의 상태 정보를 더욱 정확하게 갱신하기 때문에 다음 sender가 주도한 협상과정에서 유리하게 작용할 것이다. 전체적으로, 시스템이 저부하인 경우 송신자주도형을 적용하고, 시스템이 과부하인 경우 수신자주도형을 반영하는 것이 평균응답시간을 분석한 결과 성능이 좋아짐을 알 수 있었다. 본 논문에서 제안한 대칭적 부하균등 알고리즘은 이들 2가지 경우를 모두 고려하여 설계하였기 때문에 어떠한 시스템 부하 상태에서도 훌륭한 성능을 발휘할 뿐 아니라, 시스템이 불안정 상태에 들어가는 것을 방지할 수 있다.

(2) 과부하 상태인 경우

부하균등 클러스터에 속한 노드에 작업들이 전체적으로 균등하게 높은 비율로 도착함에 따라, 3가지 알고리즘 모두 시스템 부하가 10을 전후로 하여 평균 응답시간이 급격히 증가함을 보이고 있다. 이는, 전체적으로 과부하상태이기 때문에 저부하 상태인 노드를 찾을 확률이 그만큼 줄어들기 때문에 시스템이 불안정한 상태에 빠짐을 의미한다.

5. 결 론

본 논문에서 이종(Heterogeneous)분산 시스템을 위한 대칭적 부하균등 알고리즘을 제안했다. 제안한 알고리즘의 핵심은 이주될 부하를 gur과 res에 따라 양자간 협의 하에 전송하여 처리토록 하는데 있다. 이때 원격 실행에 참여하는 부하는 대체로 로컬 정보를 사용하지 않고 장시간 처리를 요하는 CPU 위주의 작업이 고려된다. 이는 처리비용이 적은 작업의 전송시 CPUtask 비용으로 인한 부하균등의 이득이 상쇄되는 것을 방지하기 위함이다.

본 논문에서 제안한 대칭적 부하균등 알고리즘은 이종적인 작업 발생이나 다양한 처리능력을 갖는 노드들이 부하균등 클러스터로 구성될 때 시스템이 불안정 상태에 빠지지 않고, 훌륭한 성능을 발휘함을 알 수 있었다.

참 고 문 헌

[1] A.Goscinski, Distributed Operating Systems : The Logical Design, Addison Wesley Publishing Company, 1991.

[2] T. L. Casavant, "Analysis of Three Dynamic Load-Balancing Strategy with varing Global Information Requirements," The 7th International Conference on Distribut ted Computing Systems, Berlin, West Germany, pp. 185~192, 1987.

[3] H. Y Chang, M.Kuvnt, "Distributed Scheduling under Deadline Constraints : A Comparison of Sender-Initiated and Receiver-Initiated Approachs", Proceeding of Real-time Systems Symposium, Dec., 1986.

[4] 임종규, "결함노드의 부하 재분배를 위한 부하재분배기 모델 연구", 학위논문, 수원대학교, 1993.

[5] MacDougall, "Simulating Computer Systems", 1987, MIT Press.

[6] L. M. Ni, C. W. Xu, and T. B. Gendreau, "Drafting algorithm - a dynamic process migration protocol for distributed systems. In Proceedings". the 5th International Conference on Distributed Computing Systems, pp 539-546. IEEE, 1985.

[7] A.S.Tannenbaum & R.V.Renessee, "Distributed Operating Systems", ACM Computing Surveys,

Vol.17, No.4, Dec., 1985, pp. 439~468.

[8] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computin systems", IEEE trans. on SE.41, pp. 141~154, Feb. 1988.

[9] W. Zhao, K. Ramamrithm, "Distributed scheduling using bidding and focused addressing", proc. IEEE real-time Symp., Dec., 1985.

◐ 저 자 소 개 ◑

장 순 주

1989년 한밭대학교 전자계산학과 졸업(학사)
1995년 수원대학교 대학원 전자계산학과 졸업(석사)
2000년 수원대학교 대학원 전자계산학과 졸업(박사)
1997년~2002년 수원대학교 전자계산학과 강사
2001년~현재 : 한신대학교 교육대학원 강사
관심분야 : 분산운영체제, 멀티미디어, 데이터베이스, 결함허용 etc.
E-mail : sjcchang1@hanmail.net