보장/예약 기법을 이용한 프로세서 쓰레싱 감소 방안

이준연*, 임재현**
*동명정보대학교 멀티미디어공학과
**국립공주문화대학 컴퓨터정보과
e-mail:jylee@tmic.tit.ac.kr
nolboo@munhwa.kongju-c.ac.kr

Reducing the frequency of the processor thrashing using guarantee/reservation

Jun-Yeon Lee*, Jae-Hyun Lim**
*Dept of Multimedia Engineering, Tongmyong University of Information Technology
**Dept of Computer Information, Kongju National Culture College

요약

동적 부하 분산 정책은 시스템의 상태를 고려하여 부하 균등화를 결정한다. 이를 위하여 분산형 정책에서는 각 노드가 부하 균등화를 결정하기 전에 적절한 부하 전송 대상을 찾기 위하여 현재의 시스템 상태 정보를 수집한다. 그러나 이 과정에서 프로세서 쓰레싱이 발생하게 된다. 본 논문에서는 프로세서 쓰레싱의 발생 빈도를 감소시키기 위한 새로운 알고리즘을 제안하였다. 그리고 모의실험을 통하여 제안된 알고리즘의 성능 개선을 증명하였다.

1. 서론

일련의 프로세싱 노드들이 네트워크로 연결된 분산 컴퓨팅 시스템에서 일부 노드들은 다른 노드들에 비해 작업 도착율이 높은 경우가 자주 발생한다[1][2]. 이러한 경우 부하의 불균형이 발생하며, 이때 유휴 상태이거나 혹은 저부하 상태의 각 프로세서들의 사용율을 높이기 위하여 작업 부하를 균등화함으로써 작업의 평균 응답 시간을 최소화하고, CPU 사용율과 전체 시스템의 성능이 최대화될 수 있다.

동적 부하 분산 정책중 가장 성능이 좋은 방법인 분산형 정책에서는 시스템 상태가 모든 노드로 분산되므로 중앙 집중형에서 발생하는 단점을 줄일 수 있다. 그러나 분산형 정책에서의 문제는 도착 시간 간격만큼이나 서비스 시간의 변화에 민감하다는 것이다[3]. 이러한 문제를 해결하기 위한 분산 정책은 부하 분산의 시작 위치에 따라 송신 노드 시작 정책과 수신 노드 시작 정책, 그리고 대칭형 정책의 세 가지가 있으나 각각 문제점으로 인하여 현재 대칭형 정책이 사용되고 있다.

대칭형 정책은 송신 노드나 수신 노드가 모두 작업 전송의 대상을 찾기 위하여 알고리즘을 시작할 수 있는 방법이다[4]. 대칭형 알고리즘은 현재 시스템의 부하 상태에 따라 위치 정책에 가장 적합한 것이 될 것이다. 대칭형 알고리즘은 송신자 시작 정책과 수신자 시작 정책을 동적으로 변경할 수 있어야 한다. 다른 형태의 위치 정책에서 대칭형 알고리즘은 시스템의 성능과, 이러한 성능을 얻기 위하여 지불하여야 할 오버헤드를 고려할 때 가장 유리한 정책이다. 그러나 이 정책은 프로세서 쓰레싱을 감수하여야 한다[5].

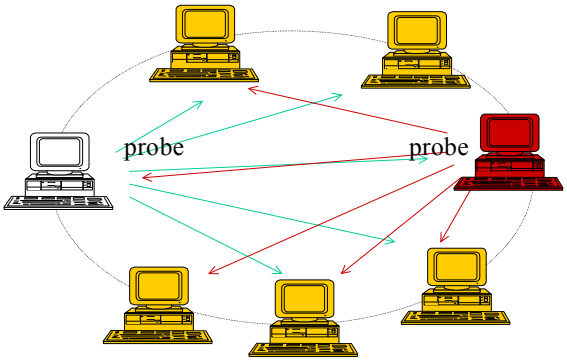
따라서 본 논문에서는 대칭형 알고리즘에서 프로

세서 쓰레싱의 빈도를 줄일 수 있는 새로운 알고리즘을 제안하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서 관련 연구를 소개한 후, 3장에서 본 논문에서 제안한 보장/예약 알고리즘을 기술한다. 4장에서는 시뮬레이션의 결과를 분석하고, 마지막으로 5장에서 결론을 맺는다.

2. 관련연구

대칭형 위치 정책의 목적은 과부하 노드가 작업을 전송하려 할 경우에는 저부하 노드를 찾고, 저부하 노드가 작업을 수신하려 할 경우에는 송신 노드를 찾도록 하는 것이다[4]. 이 정책은 Shivaratri와



<그림 1> 대칭형 정책

Krueger가 제안한 알고리즘이므로 이를 SK 알고리즘이라 한다. 이를 그림으로 표시하면 <그림 1>과 같다.

3. 알고리즘

3.1 보장/예약 프로토콜

보장/예약 프로토콜은 프로세서 쓰레싱을 해결하기 위하여 고안된 것으로 기본적인 방법은 다음과 같다. 송신 노드와 수신 노드의 쌍이 결정되면 송신 노드는 폴링 메시지에서 수신 노드로의 전송을 보장할 수 있는 작업의 수(보장값:Guarantee value)를 정의한다. 이 값을 근거로 수신 노드는 송신 노드로부터 받아들일 작업의 수(예약값:Reservation value)를 결정한다. 예약값은 송신 노드로부터 약속된 양만큼 받기 위한 할당량으로서, 그리고 다른 송신 노드로부터의 작업 송신을 막기 위하여 사용된다[6].

예약/보장 프로토콜은 각 노드에서 두가지 속성을 사용한다.

정의 1: 노드 P_i의 총 예약값(RES_i)은 모든 송신 노드로부터 자신에게 예약된 작업의 수이다.

정의 2: 노드 P_i의 총 보장값(GUR_i)은 모든 수신 노드가 전송을 보장한 작업의 수이다.

3.2 작업부하 상태

GUR_i와 RES_i, 그리고 수행되는 모든 작업의 수를 근거로 하여 노드P_i의 부하를 다음과 같이 정의할 수 있다.

정의 3: 노드 P_i의 실제 부하(K_i)는 작업 큐, 서비스 큐, 그리고 임계 큐에 존재하는 작업의 총 수를 의미한다.

정의 4: 노드 P_i의 가상 부하 $VW_i = K_i + RES_i - GUR_i$ 이다.

부하를 정확하게 측정하기 위하여 여러 가지 부하 요소를 고려하는 것이 간단한 부하 인덱스에 비해 현저한 성능 개선을 가져오지는 못한다[7][8]. 따라서 본 논문에서는 각 노드의 처리 용량의 수준을 나타내기 위하여 3단계의 부하 측정 기법을 사용한다. 이 3단계는 H-load(과부하), N-load(적정 부하), L-load(저부하 상태)이다. 한 노드의 부하 상태는 가상 부하 VW_i에 의하여 결정된다. <표 1>은 부하 상태와 가상 부하 VW_i의 관계를 나타낸다.

<표 1> VW_i를 기준으로 한 부하 상태

부하 상태	기준
L-load	$VW_i \leq lower\_threshold$
N-load	$lower\_thershold < VW_i \leq upper\_threshold$
H-load	$VW_i > upper\_threshold$

3.3 위치 정책

보장/예약 알고리즘은 [3]에서 제안된 적응성있는 대칭형 위치정책을 사용한다. 이 정책의 특징은 다른 노드의 최근 부하 상태를 탐지하기 위하여 폴링 시에 수집된 부하 정보를 사용하는 것이다.

노드 P_i는 부하 정보를 위하여 3개의 리스트 구조를 유지한다. $SList_i$는 잠정적으로 송신 노드로 간주된 모든 노드의 id를, $RList_i$는 잠정적으로 수신 노드로 간주된 모든 노드의 id를, 그리고 $NList_i$는 적정 부하를 가진 노드의 id를 포함한다.

한 노드에서의 부하 전송의 협상은 노드의 부하 상태가 변경되는 순간에 결정된다.

폴링 기반 부하 균등화 알고리즘에서 무제한의 폴링이 CPU 시간과 네트워크 대역폭을 낭비할 수 있다. 따라서 본 논문에서 제안한 알고리즘에서는 부하 균등화 과정에서 만들어질 수 있는 폴링의 횟수에 제한을 둔다. 이 제한을 $probe_limit$라 한다.

3.4 전송량 결정

적절한 전송량의 크기는 알고리즘의 성능에 따라 결정되기 때문에, 송신 노드와 수신 노드 사이의 상호 동의를 구하기 위하여 세가지 규칙을 정하여 사용한다. 전송량의 협상은 보장/예약 프로토콜에 포함하여 실행할 수 있다. *max*는 수신 노드가 송신 노드로부터 받아들이고자 하는 최대 작업수이며, *MaxAssign()* 함수에 의해 결정된다. *t*는 송신 노드가 수신 노드로 전송하고자 하는 작업수이며, *NumAssign()* 함수에 의해 결정된다.

이 두 함수는 <프로그램 1>과 <프로그램 2>에서 정의한다.

VW_r은 수신 노드 P_r의 현재 가상 부하라 하고, VW'_r은 작업 전송후 예측되는 수신노드 P_r의 가상 부하라 한다.

P_r이 *max*개의 작업을 받았다면

$$VW'_r \approx EI_r + max$$

$$VW'_r \le upper\_threshold$$

따라서

$$max \le upper\_threshold - VW_r$$

*max*의 가능한 최대값은

$$max = upper\_threshold - VW_r \quad (1)$$

<프로그램 1> *MaxAssign()* 함수

VW_s를 송신 노드 P_s의 현재 가상 부하라 할 때,

$$VW_s - t > lower\_threshold$$

$$t < VW_s - lower\_threshold \quad (2)$$

공식 (1)을 사용하여 P_r의 가상 부하 VW_r''는

$$VW_r'' \approx upper\_threshold - max \quad (3)$$

식 (3)은 가정 3에 의하여

$$VW_r'' + t \le VWs - t$$

$$\therefore\ t \le \frac{VW_s + \max - upper\_threshold}{2} \quad (4)$$

여기서

$$t \le max \quad (5)$$

따라서 *t*의 값은 (2), (4), (5)를 만족하는 최대 정수값이다.

<프로그램 2> *NumAssign()* 함수

VW''_r을 P_r의 가상 부하라 하고,

VW'''_r을 *b*만큼 수신후의 새로운 가상 부하라면

$$VW'''_r = VW''_r + b$$

여기서 공식(3)에 의하여

$$VW'''_r \approx upper\_threshold - max$$

따라서 $VW'''_r = upper\_threshold - max + b$

<프로그램 3> *ReceiverNewVW()* 함수

일련의 작업이 수신 노드로 전송된 후, 송신 노드는 *ReceiverNewVW()* 함수를 호출하여 수신 노드의 새로운 가상 부하를 평가한다. 이러한 평가는 *max*의 값과 전송할 작업의 크기 *b*의 값을 기초로 하여 결정된다. *b*의 최종값은 *t*와 *res*중 더 작은 값이 선택된다.

3.5 송신 노드 협상 시작 알고리즘

송신 노드를 P_s라 하고, 수신 노드를 P_r이라 할 때, 보장값 *gur*과 예약값 *res*를 고려한 송신 노드의 협상을 시작하는 알고리즘은 다음 <프로그램 4>와 같다.

Procedure *S1*

1. select target node:
   target node P_r = head of $RList_s$
2. determine guarantee value *gur*:
   gur = (# of tasks in task queue) - GUR_s
3. $GUR_s = GUR_s + gur$
4. $VW_s = K_s + RES_s - GUR_s$
5. send a polling message to P_r

<프로그램 4> 송신 노드의 협상 시작 알고리즘

송신 노드 P_s로부터 폴링 메시지 *POLLING(gur)*를 받은 잠정적인 수신 노드 P_r은 먼저 P_s의 정보가 포함된 리스트를 수정한 후, 자신의 가상 부하 상태를 검사한 후, 자신이 수신 노드의 상태라면 전송받을 수 있는 작업의 양을 알리는 예약값 *res*와 현재의 가상 부하를 계산한 후, 응답 메시지 *ACK(gur, max, res)*를 송신 노드에게 전달한다.(<프로그램 5>)

Procedure *R1*

1. List update
   Remove P_s from whatever list it is in
   add it to the head of $SList_r$.
2. If P_r is NOT a receiver(i.e. $VW_r \neq$ L-load)
   goto procedure *R1'*
   endif
3. Determine reservation value *res*:
   max = *MaxAssign()*
   if *max* ≤ *gur* then *res* = *max*
   else *res* = *gur*
4. Accumulate reservation value *res*:
   $RES_r = RES_r + res$
5. $VW_r = VW_r + RES_r - GUR_r$
6. Send an *ACK* message to P_s

<프로그램 5> 수신 노드의 예약값 계산 알고리즘

P_r로부터 *ACK(gur, max, res)*을 받은 P_s는 수신 노드로부터 받은 예약값 *gur*과 자신이 결정한 보장

값 *res*를 비교하여 전송할 작업의 크기를 결정한다. 그리고, 작업을 전송한 후에 달라질 수신 노드의 부하 상태를 수정하고, 자신의 보장값 *GUR*과 예약값 *RES*을 수정한 후, 작업을 전송하고 수행을 종료한다.(<프로그램 6>)

```
Procedure S2
1. Release guarantee value gur:
   GUR_s = GUR_s - gur
2. Determine job size b:
   t = NumAssign()
   if t ≤ res then b = t
               else b = res
3. Select b tasks from task queue:
   If no task can be selected
        goto procedure S2'
   endif
4. Transfer b tasks to P_r:
   K_s = K_s - b
5. VW_s = K_s + RES_s - GUR_s
6. List update:
   Estimate P_r's new effective load state
        by calling ReceiverNewVW().
   Move P_r to the head of appropriate list of P_s.
7. Reset probe_limit counter and stop
```

<프로그램 6> 송신 노드의 작업 전송량 결정 알고리즘

송신 노드 P_s로부터 폴링 메시지 *TRANSFER(tasks, b, res, VW_s)*를 받은 P_r는 송신

```
Procedure R2
1. Lists update:
   Move P_s to the head of the appropriate list
according to VW_s
2. Append b tasks onto the threshold queue:
   K_r = K_r + b
3. Release reservation value res:
   RES_r = RES_r - res
4. VW_r = K_r + RES_r - GUR_r
```

<프로그램 7> 수신 노드의 예약값과 가상부하 재설정

노드의 가상 부하 상태 VW_s에 따라 리스트를 수정하고, 전송받은 작업을 임계 큐에 추가한 후, 예약값과 자신의 가상 부하를 재설정한다. (<프로그램 7>)

프로시저 *R1'*으로부터 전송된 폴링 메시지 *NACK(gur, VW_r)*를 받은 송신 노드 P_s는 현재의 잠정적인 수신 노드가 수신 상태가 아님을 감지하고, 현재의 P_r 노드의 *id*를 $RList_s$로부터 제거한 후 VW_r 상태에 따라 적절한 리스트에 추가한 후, 잠정적인 수신 노드를 다시 선택하기 위하여 프로시저 *P1*을 다시 시작한다. (<프로그램 8>)

한편 프로시저 *S2*에서 *NumAssign()* 함수의 실행 결과와 예약값의 크기를 기초로 하여 전송할 작업의

```
Procedure S3
1. Lists update:
   Move P_r to the head of the appropriate list
according to VW_r
2. Release guarantee value gur:
   GUR_s = GUR_s - gur
3. VW_s = K_s + RES_s - GUR_s
4. Initiate another polling session by going to S1
unless either:
   a. probe_limit is exceeded or
   b. RList_s is empty or
   c. P_s is no longer a send (i.e. VW_s ≠ H-load)
```

<프로그램 8> 잠정적인 수신 노드의 재설정 알고리즘

크기가 정해졌으나, 작업 큐에서 실제로 이 크기만큼의 작업을 선택할 수 없는 경우가 발생할 수 있다. (<프로그램 9>)

```
Procedure S2'
1. VW_s = VW_s + RES_s - GUR_s
2. List update:
   Estimate P_r's new virtual load by calling
ReceiverNewVW().
   Move P_r to the head of the appropriate list of P_s
accordingly.
3. Send a NACK message to P_r,
   maintain VW_s
4. Initiate another polling session by going to S1
unless either
   a. probe_limit is exceeded or
   b. RList_s is empty or
   c. P_s is no longer a sender (i.e. VW_s ≠ H-load)
```

<프로그램 9> 송신 노드에서 전송할 작업의 선택이 불가능한 경우

```
Procedure R3
1. List update:
   Move Ps to the head of the appropriate list
according to VWs
2. Release reservation value res:
   RES_r = RES_r - res.
3. VW_r = K_r + RES_r - GUR_r
```

<프로그램 10> 예약값의 작업을 전송받지 못한 수신노드의 알고리즘

프로시저 S2'에서 송신 노드 P_s가 자신이 보장한 값만큼의 작업을 전송할 수 없을 경우, 수신 노드 P_r는 폴링 메시지 *NACK(res, VW_s)*를 받는다. 이 경우 P_r는 자신에게 예약되었던 크기의 작업 부하가

전송될 수 없으므로, 이에 대한 처리를 한 후 수행을 종료하여야 한다. (<프로그램 10>)

3.6 수신 노드 협상 시작 알고리즘

수신 노드는 먼저 협상을 하기 위한 대상이 되는 잠정적인 송신 노드를 선택하여 대상 노드에게 자신의 보장값을 수신할 수 있는 지를 문의하는 폴링 메시지를 전송한다.(<프로그램 11>)

Procedure *R4*
1. Target node P_s is select:
   If $SList_r$ is not empty, choose the node at the head of $SList_r$.
   else, choose the last node from $RList_r$.
2. Determine reservation value *res*:
   $res = max = MaxAssign()$
3. Accumulate reservation value *res*:
   $RES_r = RES_r + res$
4. $VW_r = K_r + RES_r - GUR_r$
5. Send a polling message to P_s

<프로그램 11> 수신 노드의 협상 시작 알고리즘

P_s는 자신이 전송할 수 있는 작업의 크기를 결정한 후, 자신의 작업 큐에서 작업을 선택하여 수신 노드 P_r로 작업을 전송하고, 전송후의 P_r의 부하 상태를 검사하여 리스트의 내용을 수정한다. (<프로그램 12>)

Procedure S4
1. Determine transferring job size *b*:
   $t = NumAssign()$
   If $t \le res$ then $b = t$
   else $b = res$
2. Select *b* tasks from task queue:
   If no task can be selected, goto procedure *S4'*
3. Transfer *b* tasks to P_r
   $K_s = K_s - b$
4. $VW_s = K_s + RES_s - GUR_s$
5. List update:
   Estimate P_r's new virtual load by calling *ReceiverNewVW()*.
   Move P_r to the head of the appropriate list of P_s accordingly

<프로그램 12> 송신 노드의 작업 전송량 결정 알고리즘

협상에 참가한 송신 노드 P_s에서 전송되는 폴링 메시지의 형태는 *TRANSFER(tasks, b, res, VWs)*와 같다. 전송될 작업(*tasks*)과 작업의 크기(*b*), 예약값(*res*), 그리고 전송후의 송신 노드의 기대되는 가상 부하량(*VWs*) 등의 정보가 폴링 메시지와 함께 전달된다.

폴링 메시지를 받은 수신 노드 P_r은 이러한 정보를 근거로 리스트를 수정하고, 전송받은 작업을 자신의 임계 큐에 추가하며, 가상 부하값을 수정한다.

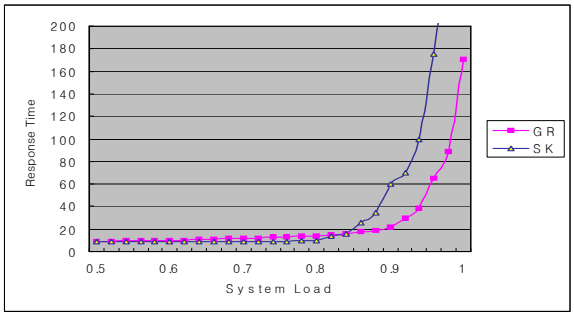
Procedure *S5*
1. List update:
   Move P_s to the head of the appropriate list according to VW_s
2. Append *b* tasks onto the threshold queue:
   $K_r = K_r + b$
3. Release reservation value res:
   $RES_r = RES_r - res$
4. $VW_r = K_r + RES_r - GUR_r$
5. Reset *probe_limit* counter and stop.

<프로그램 13> 수신 노드로의 작업 전송 과정 알고리즘

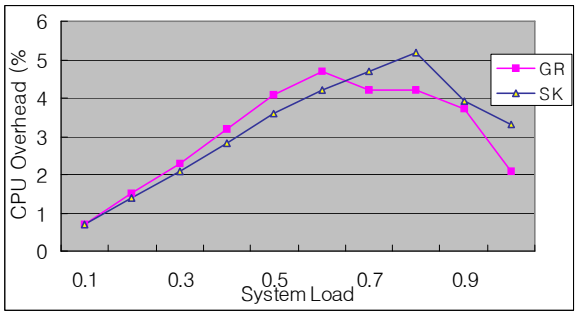
4. 시뮬레이션 결과 및 분석

본 논문에서 제안한 보장/예약 기법의 성능을 검증하기 위하여 시뮬레이션한 결과를 분석하였다. 시뮬레이션을 위하여 사용된 도구는 K. T. S. Co., Ltd의 Siman IV이며 실험 환경은 시스템은 Pentium III 400MHz, Windows 98이다.

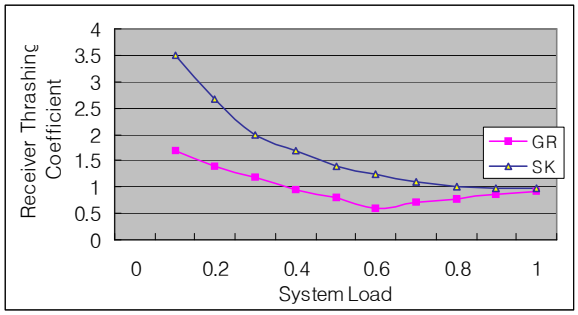
본 논문에서 제안한 보장/예약 알고리즘과 대칭형 위치 정책을 제안한 Shivaratri와 Krueger의 단일 작업 할당 알고리즘의 성능을 시뮬레이션을 통하여 비교하였다. 편의를 위하여 본 논문에서 제안한 보장/예약 알고리즘을 GR이라 하고, Shivaratri와 Krueger가 제안한 알고리즘을 SK라 한다



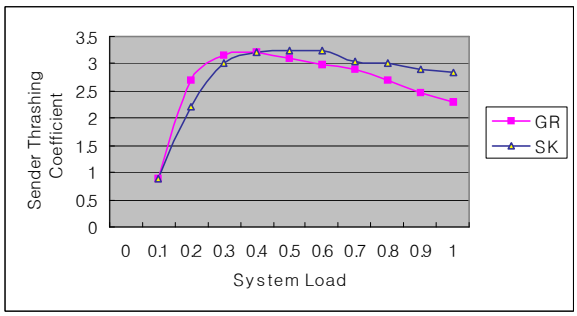
<그림 2> 평균 응답 시간



<그림 3> CPU 오버헤드



<그림 4> 수신 노드 쓰레싱 공통계수



<그림 5> 송신 노드 쓰레싱 공통 계수

5. 결론

본 논문에서는 대칭형 부하 분산 정책에서 발생할 수 있는 프로세서 쓰레싱의 빈도를 감소하기 위하여 알고리즘을 제안하였으며 알고리즘의 성능 개선을 증명하기 위하여 시뮬레이션 도구를 이용한 모의실험을 통하여 알고리즘의 성능을 분석하고 제안된 알고리즘을 기존 대칭형 정책과 비교하여 평가하였다.

평균 응답 시간은 시스템이 저부하일 경우에는 기존 방법과 큰 차이를 보이지 않으나 평균 응답 시간이 급격히 증가하는 시스템 과부하일 경우에는 기존 연구에 비해 현저한 성능 증가를 보였다. 그리고 CPU 오버헤드의 비율 또한 시스템의 과부하 상태에서는 본 논문에서 제안한 알고리즘이 기존의 알고리즘에 비해 더 낮다.

쓰레싱 공통계수는 송신 노드와 수신 노드에 따라 그 값이 다르게 나타난다. 본 논문에서 제안한 알고리즘이 수신 노드 쓰레싱 공통계수의 경우 시스템의 부하가 낮을 때 발생 빈도가 낮으며, 송신 노드 쓰레싱 공통계수는 시스템의 부하가 높을 때 더 유리하다. 따라서 시스템의 임계 큐의 상한값과 하한값을 적절하게 선택한다면 모든 시스템 상황에서 쓰레싱의 발생 빈도를 낮출 수 있다.

본 논문에서 제안한 알고리즘은 모두 동종의 시스템만을 고려하여 알고리즘을 설계하였다. 그러나 이기종 분산 시스템에서는 각 노드의 처리 성능의 차이로 인하여 송신 노드에서의 작업 전송이 수신 노드의 부하량 평가에 굉장히 큰 영향을 미치거나, 혹은 전혀 영향을 미치지 못할 수도 있다.

따라서 향후 이기종 분산 시스템에서의 알고리즘 적용을 위하여 각 노드의 처리 성능을 고려한 전송 부하량 평가에 대한 연구가 필요하다.

참고문헌

[1] Marvin M. Theimer and Keith A. Lantz. "Finding Idle Machine in a Workstation-Based Distributed systems," IEEE Transactions on Software Engineering, Vol.15 No.11, November 1989.

[2] L. M. Ni, C. W. Xu, and T. B. Gendreau. "Load Balancing form a UNIX Shell". In Proceedings, the 13th Conference on Local Computer Networks, October 1988.

[3] Thomas Kunz. "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," Technical Report TI-6/91, Institute for Theoretische Informatik, Fachbereich Informatik, Technische Hochschule Darmstadt, Dec. 1991.

[4] K. Benmohammed-Mahieddine and P. M. Dew. "A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems," SIGOPS, Vol.28 No.1 pp.66-77, Jan. 1994.

[5] Chin L. U. and Sau-Ming L.A.U. "An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple task classes," Proceedings of 16th Distributed Computing Systems, May. 1996.

[6] Jun-Yeon Lee, Young-Chan Kim. "Synchronization algorithm of migrating service object." Proceedings of the High Performance Computing Conference, pp.1407-1413, Sep. 1998.

[7] D. L. Eager and E.D. Lazowska. "Adaptive Load Sharing in Homogeneous Disributed Systems". IEEE Transactions on Software Engineering, Vol.SE-12 No.5, May 1986.

[8] C. Jacqmot and E. Milgrom. "A Systematic Approach to Load Distribution Strategies for Distirbuted Systems," In Decentralized and Distributed Systems, Sep. 1993