論文
5-9-1

부하 균등화 기법 연구
(A Study on the Load Balancing Strategy)

김광휘(Kyang-hyu Kim)[1] 정구영(Gu-Young Jung)[2]

요약

본 논문은 분산 시스템하에서 효율적인 분산자원 관리를 위하여, 각 노드는 정확한 의사 결정을 내려야 하는데, 이에 장애가 되는 통신 네트워크상의 지연, 통신 주기, 기타 의사 결정을 위한 계산 시간등을 충분히 고려하였다. 또한, 직접 통신 방식을 사용하여 전체 시스템의 성능을 향상시킬 수 있다.불확실한 부하정보는 시간변화에 따라 부하 정보에 대한 유용성을 변화 시켜 완화하였다.

ABSTRACT

In this paper under the distributed system for efficient distribution resource to system's each node must be designed to get right decision making. Thus we considered computing time to estimate fault such as delay on communication network, communication period and other decision making. Aiso, using direct communication mode improve the availity of total system.

논문접수 : 2004. 10. 14.
심사완료 : 2004. 10. 29.

1) 정회원 : 우송정보대학
2) 정회원 : 우송정보대학

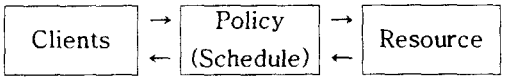
Ⅰ. 서 론

1.1 부하균등화

부하 균등화(load balancing)는 분산 시스템의 자원 공유 측면에서 각 노드들의 프로세서를 하나의 자원으로 보고 작업들을 상황에 맞게 분배시킴으로써, 프로세서의 공유를 행하여 주는 행위라 할 수 있다.
비 분산 시스템에서의 작업 스케쥴링 정책들(FIFO, LRU, Round Robin등)을 로컬 스케쥴링, 또는 분산 스케쥴링이라 할 수 있는데, 사용자 입장에서 보면 시스템을 하나의 거대한 가상 프로세서(powerful virtual processor)로 생각할 수 있게끔 해주어 전체적인 처리율(throughput)을 최대화하고 응답시간(response time)을 최소화하는 목적을 지니고 있다. 분산 시스템은 지역적으로 자치적(autonomous)인 노드들이 산재되어 있고, 메시지 전달에 의하여 계산 능력, 자료, 하드웨어 장치와 같은 여러 컴퓨터와 연관된 자원들의 공유를 허용하는 독립적인 프로세서들의 집합으로 이루어져 서로 통신하는 시스템이라 할 수 있다. 분산 시스템은 지역적으로 떨어져 있는 여러 처리 요소들을 제어 함에 있어서 크게 분산(distributed) 제어기법 및 집중(centralized) 제어기법으로 분류한다.

1.2 연구 배경

본 논문에서는 분산 제어기법에 대하여 연구 초점을 맞추었으며, 이유는 분산 제어기법이 다음과 같은 장점을 가지고 있기 때문이다. 중복되어 있는 제어로 인하여 한 노드의 고장시에도 나머지 노드들은 정상적인 제어가 가능한 신뢰성과 각 노드에 분산된 제어로 여러 결정들이 병렬적으로 이루어 질 수 있으므로, 시스템의 효율은 그만큼 빠르게 되며, 분산형 제어기법은 집중형 제어기법에 비해, 한 노드에 집중된 제어로 인한 병목 현상 및 통신 경로의 병목 현상을 최소화 할 수 있고 다른 노드에만 의존적이 아닌 자신이 자기자신에 대한 모든 제어도 가지므로, 자신의 일을 자치적으로 행할 수 있다. 그리고 시스템 전체의 설계를 바꾸지 않고도 시스템의 성능과 기능을 바꾸거나 확장하기가 용이하며, 또한 시스템의 전반에 걸친 제어로 인해 시스템의 모든 자원은 공유(resource sharing)될 수 있다. 이러한 분산 시스템의 효율성과 성능을 향상시키기 위하여 어느 한 노드의 작업 부하를 적절한 노드(과부하 노드에서 저부하 노드로)로 재분배하는 부하 균등화는 작업부하를 균등 시키는 네트워크에 의해 연결된 프로세서들 가운데 분산 프로세스 방법이며 적재 공유는 거기에 어떤 유휴 프로세스가 되었을 때 프로세스 이주(process migration)를 사용할 수 있고, 작업 부하 균등은 노드가 유휴되지 않을때 이주를 요구한다. [그림 1]



[그림 1] 자원 관리의 관계

클라이언트는 빠르게 효율적으로 이주자원을 접근하고 실행하므로 거대한 오버헤드가 발생하는 것을 방지해야 하며, 정책 기능을 수행하는 시스템은 다음의 두가지 클라이언트 요구를 이행해야 한다.[15]

①성능(performance) - 적당한 자원들은 시스템에 의해 관리하는 방법.

②효율성(efficiency) - 이것을 접근하는데 어렵거나 costly할 때 시스템을 통해서 자원을 사용하는 방법.

위와 같이 부하 균등화 기법의 설계는 분산 시스템 전체의 처리능력을 이용하여 각 노드의 부하 적체 현상을 작업 부하의 이주를 통해 완화시킴으로써 전체의 성능을 향상시키는 작업이여, 상호 네트워크로 연결된 분산 처리 시스템에서의 각 노드들은, 자원을 충분히 활용하기 위해 모든 작업들이 빠른 응답 시간을 얻기 위하여 프로세스 이주를 행한다. 프로세스 이주(process migration)란 프로세스의 실행기간(life-time)중에 하나의 기계에서 다른 기계로 프로세스 문맥 (context)을 옮겨서 실행하는 것을 의미하며, 이러한 프로세스 이주 기능을 제공하는 시스템은 다음과 같은 다양한 잇점을 제공할 수 있다.

① 원거리 machine의 많은 양의 데이터 참조
② 부하 공유화
③ 특정 하드웨어 참조
④ 특정 소프트웨어 참조
⑤ 고장 허용(fault tolerance)

프로세스 이주 기능은 프로세스 이주 정책과 프로세스 이주 메카니즘으로 구성된다. 프로세스 이주 정책은 "언제, 어느 프로세스를 어디로 옮겨야 하느냐"에 관련된 문제를 결정하며 이주 메카니즘은 프로세스 이주 정책에서 결정된 사항을 이행 하게 된다. 따라서 분산 시스템 상에서 프로세스 이주 방법으로 부하가 많이 걸린 프로세서나 상대적으로 부하가 적은 프로세서가 발생하지 않게 하여 모든 프로세서가 유사한 부하량을 가지므로서 전체 시스템의 이용도(Utilization)를 향상 시켜서 전체적인 분산 시스템의 성능을 향상시키도록 하여야 한다. 여기서 분산된 각 노드들은 공동의 목적을 성취하기 위해 상호협력(cooperating)또는 경쟁(competing)하는데, 각 노드들은 이러한 행위를 위한 의사 결정(decision making)을 위해 시스템 상태에 대한 지식(knowledge)을 사용한다. 이때 한 노드가 바라보는 시스템 상태정보는 예상할 수 없는 통신 네트워크 상의 지연(delay), 통신 주기, 기타 의사 결정을 위한 계산시간(computing time)등으로 인해 부정확해질 수 밖에 없으며, 분산 시스템에서, 임의의 한 노드에 처리할 작업량이 많은 과부하 상태와 서로 협력하여 처리해야 할 작업이 발생할 경우 각 노드들의 자원을 최대한 효율적으로 이용하기 위한 부하 균등화 알고리즘의 기법을 연구하였다.

Ⅱ. 의사 결정 모듈

2.1 모듈 선택

의사 결정 모듈은 로컬 상태 관찰 및 갱신 모듈로부터 생성된 상태 정보를 기반으로 미래의 효율성을 예측하고, 가장 기대되는 효율이 좋은 의사 결정을 취한다. 의사 결정은 일련의 규칙 $r \in R$ 과 퍼지 함수들에 의해서 행해지는데 규칙(rule)의 추론은 퍼지 추론(Fuzzy reasoning) 및 전진, 후진(Forward, Backward) 연쇄에 의해 의사 $d_i \in D_i$가 결정될 수 있다. 정상적인 시스템 행위중 갑자기 실제 최적 결정(Ld)에 비해 통신 오버헤드로 인한 손실의 정도(Lc)에 비해 커졌다는 사실은, 시스템이 어떠한 사건(event)들의 출현으로 인해 시스템 상태가 바뀌어지고 있는 상황이다. 이는 Ld를 크게 하는 요인이 된다. 임의의 노드 I는 상기 상황을 자신이 소유하고 있는 타노드에 대한 μEu_{ij} $(1 \le j \le n)$들의 평균에 대한 편차를 계산해서 알 수 있다. 즉 μEu_{ij}의 평균에 대한 편차가 일정 기준치 β보다 커지면 커질수록 전체시스템 상태는 큰 변화를 가지면서 바뀌어가고, 이는 Ld를 점점 크게 만드는 요인이 된다. 따라서 Ld는 Lc에 비해 상대적으로 크므로 Ld를

줄이기 위해서는 보다 빈번한 상대정보 교환이 요구된다. 또한 정상적인 시스템 행위 중 갑자기 Lc가 Ld에 비해 커졌다는 사실은, 상태 정보 교환을 많이 행했음에도 불구하고 시스템의 상태가 거의 바뀌지 않는, 즉 시스템 상태가 규칙적이면서도 안정된 상태에 있을 때 발생될 수 있는 상황이다. 이런 상황은 Ld는 적음에도 불구하고 상대적으로 Lc가 크므로 전체적인 시스템 손실이 커지는 요인이 된다. 이런 상황 역시 임의의 노드i를 기준으로 자신이 소유하고 있는 타 노드에 대한 $\mu Eu_{ij}(1\le j\le n)$들의 평균에 대한 편차를 계산해서 알 수 있다. 즉, μEu_{ij}의 평균에 대한 편차가 일정 기준치 β보다 작으면 작을수록 전체 시스템 상태는 작은 변화를 가지면서 바뀌어 가고, 이는 결국 Ld를 점점 더 작게 만드는 요인이 된다. 따라서 Ld는 Lc에 비해 상대적으로 작으므로 Lc를 줄이기 위해서는 불필요한 빈번한 상태정보 교환을 줄일 필요가 있다.

① low = $\mu_{low}(yi(t))=1-(y_i(t)/y_{max})^{1.2}$

② high = $(y_i(t)/y_{imax})^2$

③ middle = $\mu_{middle}(y_i(t))$
$=1/(1+(y_i(t)-y_{max}/2)^2/(ymax/2))$

④ Threshold α 각 노드가 상태를 천이함에 있어서, 어느 기준 α 이상의 효용 차이가 있는 경우에만 상태천이를 허용함으로써, 프로세스의 로컬 혹은 글로벌 처리를 판단하는 기준이 되는 Threshold α 값을 계산하여 너무 빈번한 상태천이로 인한 오버헤드를 없앤다.

⑤ Threshold κ 한 노드가 의사결정 과정에서 자신보다 높은 $\mu Eu_{ij}(y_{ji}(t(+1)))$를 찾는 과정에서 Threshold κ를 설정함으로써, 별 가치가 없는(로컬 위주의 작업들) 작업 이주를 줄인다.

⑥ 노드i에서 노드j를 볼 때 기대되는 유틸리티는

i) $\Delta_{ji}\le\Delta_{max}$일 때

$\mu Eu_{ij}(y_{ji}(t^{(+1)}))=\mu u_{ij}(y_{ji}(t-\Delta^{(1)}{}_{ji}))$
$+(0.5-\mu u_{ij}(y_{ji}(t-\Delta^{(1)}{}_{ji})))*\Delta_{ji}/\Delta_{max}$

ii) $\Delta_{ji}\ge\Delta_{max}$일 때

$\mu Eu_{ij}(y_{ji}(t^{(+1)}))=u^*$

여기서 $\Delta_{ji}=\Delta^{(1)}{}_{ji}+\Delta^{(2)}{}_{ji}$이며, 그 의미는 $\Delta^{(1)}{}_{ji}$는 상태정보를 획득 했을 때까지 과거 시간의 경과를 의미하며, $\Delta^{(2)}{}_{ji}$는 생성될 작업 부하 또는 정보가 미래에 사용될 시점까지의 미래 시간의 경과, 즉 Eu_{ij}의 편차에 의한 식에 의해 산출된 최적 통신주기 T^*를 의미한다. Δ_{max}는 부하 균등화를 위한 상태정보의 불확실 정도 μuc_{ij}가 1이 될 때까지의 시간을 나타낸다. 즉 임의의 노드 i가 j에 대한 상태 정보를 보유할 가치가 있는 최대 시간을 의미한다. 부하 균등화의 경우, 한 노드가 의사 결정시 가장 최적의 저부하 상태의 노드 R을 선택하고 부하를 이주시, 다른 과부하 상태의 노드들도 노드 R을 최적의 저부하 노드라고 판단하고 부하를 이주할 가능성이 많다. 따라서 이에 대한 적절한 메카니즘이 필요하다. 본 논문에서는 이를 위해 우선순위에 근거한 작업의 분산화기법을 제시한다. 각 노드는 시스템 초기 설정시 정적인 성질을 가진 우선순위 리스트를 유지한다. 우선순위 리스트는 자신의 노드에 인접한 노드들을 우선으로 우선순위가 책정될 수 있으며 또 한편으론 사용될 서버의 계산 능력 내지는 기억용량 등이 그 기준이 될 수 있다.

최상 선택 노드=$MAX[EU_{ij}-(EU_{ii}+\kappa))*(\text{우선순위}/n)^{1.2}]$, $1\le j\le n$, $EU_{ij}-(EU_{ii}+\kappa)\ge 0$

i:자신의 노드 , j:원격 노드 , κ:임계 값 , EU:미래시점의 기대 효용성 , n:노드 수

여기서 임계치 κ를 설정함으로써, 너무 빈번한 의사 결정 및 작업 이송 오버헤드 줄일 수 있다. 이러한 의사 결정에서의 규칙은 다음과 같다.

① 만약 $y_{ji}(t)$ = high $(1\le j\le n, i\ne j)$이면, Job

$S_i(t)$는 로컬에서 처리.

② 만약 $y_{ii}(t)$ = low이면, Job $S_i(t)$는 로컬에서 처리.

③ 만약 $Y_{ii}(t)$ = middle 또는 high이면, 불확실성(Uncertainty) 및 Threshold κ를 고려하여 미래시점에서의 기대 효용성(Expected Utility)이 자신보다 높은 노드들을 구하고, 이들 중 우선순위를 고려하여 가장 적당한 노드를 선택하여 작업(work)을 이주한다. 여기서의 우선순위란 노드간의 거리(Hop수), 노드들간의 Computing power, 또는 노드들의 그룹, 기타 여러 요인들이 우선순위에 포함될 수 있다.

부하 균등화를 위한 최적노드 선정을 위한 의사결정에 필요한 정보는 각 노드마다 보유하고 있어야하고, 이 테이블에 유지되고 있는 정보를 기초로 하여 수행에 가장 적합한 노드를 선정하는 알고리즘은 [알고리즘1]과 같다.

```
decide(){
  if local node state is low
        then best_node is local node
  if all node states are high
        then best_node is local node
  if local node state is middle or high
        then best_node=best_select()
  return(best_node)
      }
best_select(){
 check currunt time;
 calculate utility for each node;
 calculate the difference between
        load_timeand current time;
 calculate expected utility for each node;
 select highest-utility node;
      }
```

[알고리즘1] 최적 노드 선정 알고리즘

2.2 프로세스들의 통신

다음 상태정보 교환을 위해서, 통신주기 T^*를 우선순위를 고려하여 결정하고 상태정보 교환 모듈에게 통보한다. 여기서 주기 결정은 각 노드들간의 최종 기대되는 효용성들간이 평균에 대한 편차에 대한 함수값으로 결정되며, 이 결정지어진 주기 T^*를 가지고 상태정보 교환모듈은 그 이전 주기 T^*_{-1}가 끝난 시점부터 T^*시간이 경과한 후 모든 노드들에게 현재의 상태정보를 브로드캐스트 한다. 프로세스들 간의 통신은 고정된 크기의 메시지를 서로 교환함으로써 이루어진다. 메시지를 주고, 받기 위해서 다음과 같은 프리미티브를 사용한다.

-send(caller, dest, msg_ptr);
 destination으로 메시지를 보내는데 사용
-receive(caller, source, msg_ptr);
 source로부터 메시지를 받는데 사용
-sendrec(caller, dest, msg_ptr);
 메시지를 보내고 응답이 올때까지 대기함

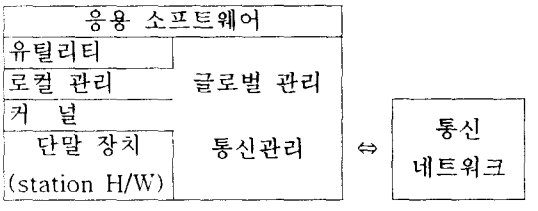
통신에 사용되는 고정된 크기의 메시지 구조는 [알고리즘2]와 같다.

```
typedef struct {
int m_source; /* 메시지를 보내는 프로세스 */
int m_type;   /* 메시지의 성격 */
 { /* 메시지 성격에 맞는 여러 가지 정보들 */
   /* system_id, buffer_size, queue_length, run_
   time, min_noad_name,buffer_pointer...etc*/
} m_union;
 } message;
```

[알고리즘2] 통신을 위한 메시지 구조

분산 시스템의 사용자들은 자원에 대한 물리

적인 위치를 의식하지 않고 시스템을 이용하는 투명성(transparency)이 제공되어야 한다. 사용자들의 프로세세들이 통신 네트워크를 통하여 다른 노드로 이주되어 실행되더라도 사용자는 작업의 결과에만 의식하고, 작업이 어느 곳에서 실행되는지에 대해서는 의식할 필요가 없어야 된다. 워크스테이션 사용자는 자신의 노드가 과부하 상태인 것을 알게 되면 사용자는 분산 시스템의 등록부(machine registry)에 시스템 상태를 문의한다. 문의 결과 사용되지 않는 워크스테이션에 대한 위치 정보를 얻은 후 위치 의존적(location dependent)인 접근을 한다[10]. 프로세스를 다른 노드에서 실행시에는 명령어 레벨(command-level)에서 다른 노드의 위치를 명시해야 한다[11]. 이런 방법 역시 사용자가 다른 노드에 대한 상태 정보를 알고 있어야 하므로 Nichols[16]의 연구와 마찬가지로 사용자에게 불편한 단점이 있다. 이런 기존의 연구들은 정적(부하 균형 시점이 프로세스 실행 개시 전)이라는 점과 분산 시스템의 내부 구성에 대한 지식을 요구하고 단일 사용자용 워크스테이션들로 구성된 시스템을 대상으로 한다는 문제점을 갖는다. 본 논문에서는 동종의 소형(micro) 또는 중형(mini)급 이상의 컴퓨터들로 대부분 시분할 시스템으로 워크스테이션과 같이 사용되지 않는 경우는 거의 없고 여러 사용자가 늘 시스템에 존재하는 특징을 갖는다. 따라서 [그림2]와 같이 점선 부분을 주요 부분으로 다루며, 연구 내용으로 하였다.



[그림2] 스테이션에서의 네트워크 환경

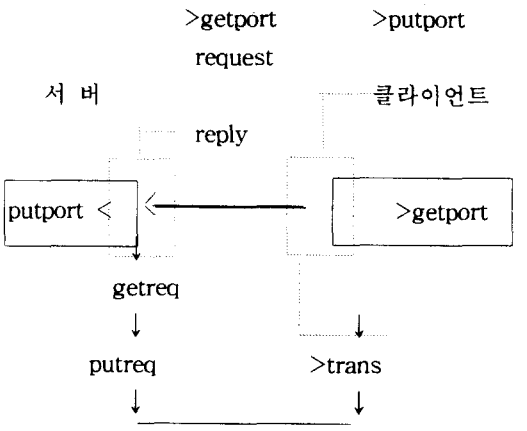
분산된 각노드의 작업이 원격 수행에 적합한지를 판별하고 원격 수행에 적합한 작업일 경우 의사 결정 모듈을 호출하여 수행에 적합한 노드를 선정하여 작업을 수행시키는 글로벌 쉘(Global SHell)은 명령어 해석기로 원격 수행에 적합한 작업들의 이름이 기록된 파일인 작업명 리스트를 읽어들인다. 작업명 리스트를 글로벌 쉘 내에서 삽입하지 않은 이유는 이 리스트에 새로운 작업을 첨가하고자 할때 이 리스트가 글로벌 쉘 내에 삽입되어 있을 경우 글로벌 쉘 자체를 수정, 재 컴파일링 하는 작업을 해야 하기 때문에 이를 방지하기 위해 따로 독립된 파일로 이 리스트를 유지하여 새로운 작업의 첨가시 추가 부담이 없도록 하였다. 이 리스트에 첨가될 수행시간이 짧은 작업들을 구별하기 위해서 time이라는 명령어를 사용했다. 이러한 알고리즘은 [알고리즘3]와 같다.

```
GSH()
{ IF best_node is local module
        THEN execution process
    ELSE {msg_passing;
          /*로컬 상태 모듈로부터 메시지 수신
*/
q_point; /*큐에서 command의 마지막에 위치
*/
msg_decision;
      /*command를 의사 결정 모듈에게 알림
*/
   exe_decision;   /*의사 결정 모듈 실행*/
   cpu_time;          /*CPU 점유시간을 측정
*/
pri_list; }  /*현 프로세스를 우선순위에 등록
*/
```

[알고리즘3] 글로벌 쉘

Ⅲ. 네트워크 환경

Ethernet을 통한 통신 메카니즘은 클라이언트 포트에서 서버 포트로의 요구 메시지와 서버 포트에서 클라이언트 포트로의 회답 메시지의 쌍으로 구성이된다. 포트는 get_port와 put_port로 나타내어 지는데, get_port는 메시지를 받기 위한 것이고 put_port는 메시지를 보내기 위한 것이다.[그림3]



[그림3] 통신 포트

요구 메시지나 회답 메시지는 헤어(header)와 실제 데이터가 저장되는 부분으로 구성된다. 메시지의 몸체(body)부분은 전송하고자 하는 실제 데이터를 저장하는 버퍼 포인트와 버퍼의 길이가 명시된다. 이런 원격 통신을 위해 커널내에 구현된 프리미티브들은 다음과 같다.

getreq(hdr,buffer,size)

세 개의 파라메터는 헤더, 버퍼, 버퍼 크기를 나타낸다. hdr 파라메터는 header structure에 대한 포인터로서 서버에게 listen할 포트를 지정할 수 있도록 허용한다. 버퍼 파라메터는 들어오는 메시지를 보관하기 위한 버퍼를 지정한다. 버퍼는 세 번째 파라메터에 의하여 지정된 최대 바이트 크기를 보관할 수 있다.

putreq(her,buffer,size)

이 프리미티브는 서버가 요청 메시지에 대한 응답을 하기 위해 사용되며 결과와 상태 정보를 반송한다.

trans(hed,buffer,size,hdr2,buffer2,size2)

이 프리미트브는 두개의 다른 파라메터 잡합을 가지고 있다. 첫 번째 세 개의 파라메터는 요청 메시지를 전송할 때 사용되고, 나머지 파라메터들은 응답 메시지를 받을 때 사용된다. trans()는 커널로 하여금 합당한 서버에 대한 요구를 보내게 되고 회답이 오기를 기다린다. 요구가 수행된 후 서버는 클라이언트에 대해 putrep()를 통해 회답을 보낸다. 본 논문에서는 위에서 설명한 바와같은 통신 메카니즘을 이용해서 즉, 버퍼에 노드 이름과 부하값을 저장해서 주기적으로 부하 정보의 교환을 수행하게 되며 교환 되는 메시지의 형태는 다음과 같다.

HEADEG	NODE ID	LOAD VALUE

여기서 헤더(header)는 메시지가 올바른 프로세스에게 전송되도록 보장하며, 이는 노도의 ID와 노드의 value를 전송하게 된다.

Ⅳ. bidding기법과의 비교

부하 균등화 기법의 성능은 부하 이동에 따르는 오버 헤드와 부하 균등화를 통한 반응 시간의 이득으로 대비하여 볼 수 있다. 이때 프로세스 이주에 따른 오버 헤드로는 로컬 부하

를 측정하는데 걸리는 시간과 부하 정보를 교환하는데 걸리는 시간, 최적노드를 선정하는데 드는 시간, 또한 작업의 원격 수행으로 인한 오버 헤드가 있다. 부하 균등화 기법은 부하 불균형 정도가 심할 경우에만 좋은 효과를 나타낸다. 이와 같은 현상이 나타나는 것은 부하 정보 주기내에 여러 노드들이 특정 저부하 노드로 부하를 이주하는 현상이 발생하기 때문이다. 저부하 노드들의 부하값들 간에 어느 정도의 차이가 있는 경우에는 시간변화에 따른 유용성의 변화를 이용해서 이러한 부하집중 현상을 막을 수가 있다. 그러나 거의 비슷한 부하값을 갖는 경우에는 유용성의 변화 정도도 거의 같기 때문에 부하 이주가 특정 노드로 집중하게 된다. 우선 순위 리스트는 과부하 노드가 여러개 존재할 경우에만 효과를 나타내고 한 노드로 많은 작업이 제출되는 경우에는 효과를 기대할 수 없다. 이런 경우에 대비하기 위해서 한 노드에 제출된 작업을 수행하기에 적합한 노드로 임의의 노드가 선정되면 다음에 그 노드의 유용성을 계산할 때 감소시키는 방법을 사용한다면, 한 노드가 갑자기 많은 작업이 제출되는 경우에 적절히 대처할 수 있으므로 평균 응답 시간이 훨씬 짧아지게 된다. 본 논문에서 구현한 부하 균등화 기법은 계산 지향 방식이므로 통신 지향 방식의 대표적 알고리즘은 bidding 기법과 성능을 비교하였다. 비교 방법은 lttmxpa의 평균 응답 시간을 척도로 평가하였고 bidding 알고리즘은 다음의 [알고리즘4]과 같다.

```
begin
  a process created at P(i)
  P(i) broadcast "request for bids;
  each P(j) (j !=i) evalute their states;
   r  eturn "bid";
    do their job;
      P(i) select the lowest ont;
       send a message to the chosen P(j);
       the chosen P(j) decides whether it is
      going to the perform the jobs OR not;
     if chosen P(j) do not want to perform
     the job then it notifies P(i);
end
```

[알고리즘4] bidding 알고리즘

bidding 알고리즘은 노드 수가 적고 발생되는 작업의 수가 적을 경우에 메시지 통신량이 적어지므로 본 실험 환경상에서 가장 좋은 성능을 나타낼 수 있다. 제출되는 작업량이 증가할 경우 와 노드수가 증가할 경우에는 본 논문에서 제시한 의사 결정 기법은 통신 메시지수의 변화폭이 완만한데 비해서 bidding알고리즘의 경우 그 통신 메시지 수가 급격히 증가하는 것으로 보아서 노드의 수가 늘고 제출되는 작업량이 많을 경우 그 성능 향상율이 점차 증가될 것으로 기대된다.

V. 결 론

분산 시스템에서 부하 균등화 기법을 설계하고 운용하는 목적은 시스템 내의 모든 노드들을 효율적으로 이용하여 시스템 전체의 성능을 향상시키기 위한 것이다. 그러나 부하 균등화 기법이 잘못 설계 구현되었을 경우에는 오히려 전체 시스템의 성능을 크게 저하시킬 수가 있다. 불확실한 부하 정보는 시간 변화에 따라 부하 정보에 대한 유용성을 변화시킴에 의해 완화시켰으며 특정 저부하 노드로의 부하의 집중현상을 방지하기 위해 우선 순위 리스트와 부하 교환 주기내에 한 노드에서 같은 리모트로의 계속적인 부하 이동을 억제하는 방식을 사용하였다. 또한 부하 이주에 부적합한 작업을 자동적으로 선별하기 위해서 작업명령 리스

트를 사용하였다. 본 논문에서 제시한 기법은 통신 메시지 증가 폭이 경미하므로 그 성능 향상률이 점점 커질 것으로 예상된다. 향후 연구에서는 본 논문에서 제시한 부하 균등화의 성능을 보다 향상 시키기 위해서 원격 수행에 의한 오버헤드를 줄일 수 있는 연구가 필요하다.

참고문헌

1. D.L.Eager et al., "Adaptive Load Sharng in Homogeneous Distributed System", IEEE Tran.on Software,Vol.SE-12,No.5,pp. 662-675,May.,1986.
2. F.Bonomi,A.Kumar, "Adaptive Optimal Load Balancing in a Heterogeneous Multiserver System with a Central Job Scheduler", Proc. 18th International conf. Distributed Computing System,pp.500-508,1988.
3. S.Zhou,D.Ferrari, "A measurement Study of Load Balancing Performance", IEEE7th Con5.on DCS,PP.490-497,Sep.,1987.
4. L.M.Ni.Chong-weixu and thomas B.Gendreau, "A Distributed Drafting Algorithm for Load Balancing ", IEEE Trans.on Software Engineering, Vol.SE-11,No.10, PP.1153-1162, oct.,1985
5. M.Scharo,K.Efe, L.Delcambre, S.Koppolue, "Heuristic Algorithms for Adaptive Load Sharing in Local Networks", proc. of tlu First Int's Con5 .on system Integration, PP.762-770,Apr., 1990.
6. Timothy C.L.chau,J.A.Abraham,"Distributed control of computer system", IEEE Trans. on comp., Vol.c-35,PP.564-567,NO.6,jun.,1986.
7. T.C.K.Chou and J.A.Abraham,"Load balancing in distributed Systems", Ieee Trans. on SE,Vol. SE8, No-4, JULY 1982,pp.401-412.
8. A.Kumar et al., " A Model For Distributed Decision Making:An Expert System for Load Balancing in Distributed System", Proc.of the 11th annual Int, l Comp. Software and Appl. Conf., Tokyo,Japan, pp.507-513, Oct.,1987.
9. J.Stankovic, "An Application of Bayesian Decision theory to Decentralized control of Job scheduling", IEEE Trans. on comp., Bol.C-34, No.2,pp.117-130,Feb.,1985.
10. David A. Nichols,"Wsing Idle Workstations in a Shared Computing Environment," In Proc.of 11 th Sympo. on Operating Systems Principles, pp.5-12,Nov.,1987
11. Kai Hwang, " A \Unix-Based Local Cmoputer Network with Load Balancing," IEEE Computer, Vol.12.No.4, pp55-66, Apr.,1982.
12. Barak,A. and Shillon,A., "A distributed load balancing policy for a multicomputer",soft pract. exper.pp 901-913,sept.,1985.
13. L.Cassavant, " Analysis of Three Distributed Load Balancing Strategies with Varying Global Information Requirements", The 7th International Conf. on Distributed Computing System.pp.185-192, Sept.,1987.
14. T.L.Cassavant and J.G.Kuhl, "A Formal Model of Distributed Decision-Making and its application to Distributed Load

Balancing", IEEE Tran. on Computers,pp.232-239,1986.

15. A.Goscinski,"Distuributed Operating Systems The Logical Design"., Addison-wesley publishing company,pp.481-513,1991.
16. Dual Systems Copr.,"VMPU-32 32Bit CPU Board with Memory Management User's Manual",1986.