論文94-31B-3-1

효율적 분산자원 관리를 위한 글로벌 스케쥴러 구현

(An Implementation of Global Scheduler for Efficient Distributed Resource Management)

丘 龍 完*

(Yong Wan Koo)

要 約

본 연구에서는 분산 시스템하에서 효율적인 분산자원 관리를 위한 글러벌 스케쥴러, 즉, 부하 균등화라는 공용의 목적을 성취할 수 있는 완전 대칭형 글로벌 스케쥴러를 설계, 구현하였다.

시스템 부하를 효율적으로 균등화하기 위해서는 각 노드는 정확한 의사 결정을 내려야 하는데, 이에 장애가 되는 통신 네트워크상의 지연, 통신 주기, 기타 의사 결정을 위한 계산 시간등을 충분히 고려하였다.

본 연구에서 제시된 부하 균등화 기법을 IBM PC/AT 들이 Ethernet으로 연결되어 구성된 분산 시스템에서 구현하였다. 대상 운영체제는 UNIX 운영체제를 IBM PC/AT상에 적합한 형태로 기본 구성을 인식하였고, 통신 계층은 Amoeba에서 구현한 통신 형태로 채택하였다. 프로세스간의 통신(IPC) 방식은 계층화된 다단계 접근방식을 취하는 프로토콜의 비효율성을 피하기 위해 직접 통신 방식을 사용하여 구현이 쉽고 프로세스간 통신(IPC)을 위하여 소요되는 사전 준비 시간이 적기 때문에 빠른 응답시간을 보장한다.

Abstract

In this study under the distributed system for efficient distribution resource completely symmetic global scheduler was designed and implemented to obtain the general global scheduler, that is load balancing as sharing objectives.

To balance the system's load efficiently 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. Load balancing mechanism which suggested in this study was implemented in the distributed system which IBM PC/AT linked to and composed with Ethernet.

The target operating system was composed of IBM PC/AT as a basic construction in which proper type of UNIX operating system were ported and communication layer chose communication type implemented from Amoeba. The method of IPC employing layered multilevel access method to avoid inefficient protocol using direct communication mode guarantees rapid response due to short ready time for IPC.

* 正會員, 水原大學校 電子計算學科
(Dept. of Computer Science, Suwon Univ.)

※ 본 연구는 한국과학재단의 1992년도 연구 지원에 의하여 이루어졌음.

接受日字 : 1993年 3月 19日

Ⅰ. 서 론

1. 부하 균등화의 정의

부하 균등화(Load Balancing) 혹은 부하 공유화(Load Sharing)는 분산 시스템의 효율성과 성능을 향상시키기 위하여 부하를 적절한 노드-과부하 노드에서 저부하 노드로-에 재분배하는 일련의 작업을 의미한다.

부하 균등화 혹은 부하 알고리즘의 동작은 효율적 Global 정보의 교환에 의존하며, 이러한 정보 메세지의 전송은 부하 균등화 알고리즘의 적절하고 정확한 수행을 돕지만, 이로 인하여 시스템 통신 효율에 부정적인 영향을 미친다. 즉 Global 정보의 정확성과 통신비용은 Trade-off 관계에 있으며, 이를 Livny [1] 는 "Communication(Transmission) Dilema"라고 언급한 바 있다.

부하 균등화는 여러 개념으로 정의 되어질 수 있는데, Zhou [2] 는 부하균등화를 노드간 부하의 불균형을 피하기 위해 컴퓨터 네트워크에 주어진 작업부하(Workload)를 재분배하는 작업이라고 정의하였고, Eager [3] 는 분산시스템 전체의 처리능력을 이용하여 각 노드 부하의 과체증(High congestion)을 작업부하의 이동을 통해 완하시킴으로써 시스템 전체의 성능을 향상시키는 작업이라고 정의하였다. 또한 Krueger [4] 는 부하균등화를 분산시스템 내 각 노드들이 거의 같은 작업부하를 갖도록 하기 위해 노드들, 혹은 프로세서들에게 프로세스가 할당되어지도록 하는 정책이라고 정의하였고, Livny [1] 는 부하균등화를 분산 시스템의 자원들의 할당을 제어하는 분산된 결정과정 (Distributed Decision Process)이라고 정의하였다.

엄밀한 의미에서는 본 연구에서의 부하 균등화는 부하 공유화와는 다르다. 이렇게 두 의미를 상이하게 구별하는 이유는, 두 용어 공히 시스템 내 노드들의 WI 상태 (WI상태란 Wait-while-Idle state의 약자로서 시스템 내 임의의 프로세서가 유휴상태(idle time)로 존재하고, 이와 동일 시각에서 타 노드의 프로세스(작업)가 실행되기를 기다리며 대기상태에 있는 상태를 의미한다)의 제거를 그 공동 목적으로 하지만, 부하 균등화가 부하 공유화보다 더욱 강한 의미의 부하 불균형의 제거를 요구한다는 데서 비롯된다. 반면에 현실적으로 완벽하거나, 또는 거의 완벽에 가까운 부하 불균형 제거 노력에 소모되는 비용 문제로 인하여, 일반적으로 부하 균등화는 그 의미가 Near-Optimal한 의미로 흐르는 경향 [5] [6] [7] 이 있으며, 따라서 앞으로 언급되어지는 "부하균등화"는 Near-Optimal한 부하균등의 의미를 지니게 된다.

결론적으로 부하 균등화란 분산 시스템의 자원 공유 측면에서 각 노드들의 프로세서를 하나의 자원으로 보고 작업들을 상황에 맞게 분배시킴으로써, 프로세서의 공유를 행하여 주는 행위라 할 수 있다. 비분산 시스템에서의 작업 스케줄링 정책들(FIFO, LRU, Round Robin등)을 로컬 스케줄링이라 한다면, 부하 균등화 기법은 작업의 할당을 분산된 프로세서들 모두를 그 대상으로 한다는 점에서 글로벌 스케줄링, 또는 분산 스케줄링이라 할 수 있는데, 사용자 입장에서 보면 시스템을 하나의 거대한 가상 프로세서(Powerful Virtual Processor)로 생각할 수 있게끔 해 주어 전체적인 처리율(Throughput)을 최대화하고 응답시간(Response Time)을 최소화하는 목적을 지니고 있다.

부하 균등화 기법은 이러한 시스템을 위해 분산 시스템 전체의 처리 능력을 이용하여 각 노드의 부하의 적체 현상을 작업 부하의 이주를 통해 완화시킴으로써 시스템 전체의 성능을 향상시키는 작업이라고 할 수 있다.

2. 사례 연구

분산 환경하에서의 작업 스케줄링은 아주 많은 연구가 이 분야에서 수행되어 왔지만 [8] [9] [10] [11] [12] 그중에서 본 연구의 응용 분야와 관련된 몇가지의 연구들에 대해서만 살펴보겠다.

Chou [13] 는 서로 다른 속도와 신뢰성의 특성을 지닌 프로세서들에게 타이스클 할당하는 문제의 해결을 위해 의사-결정 이론의, 접근 방식을 제안하였다. 비록 이 접근 방식이 정적(Static)이고 타스크에 대한 사전 정보의 유용성을 가정하긴 했지만 처음으로 의사-결정 이론을 명백하게 사용한 것이며 좋은 실험적인 결과치를 얻을 수 있었다.

Kumar [14] 는 게임 이론과 퍼지 집합 이론을 분산 의사-결정에 도입하였고 퍼지 프로세스와 시간이 지남에 따른 확률을 제안하였다. 모델의 응용 분야로서 분산 전문가 시스템(Distributed Expert System)을 부하 균등화를 위해 제시하였으며 프로토타입이 구현되었다. 또한 의사-결정 규칙과 정보 교환 규칙을 설계하여 상태 정보의 효과적인 교환 정책으로 여러가지 기법을 상황에 맞게 택할 수 잇게 하였고, 시간이 지남에 따른 상태 정보의 신뢰도를 관리하였다.

Stankovic [15] 은 분산된 작업 스케줄링에 베이지안 의사-결정 이론에 기반을 둔 휴리스틱을 응용하였다. 상태 정보의 효율적인 유지를 위해 동적인 휴리스틱이 사용되었으며 5개의 노드로 구성되어 있는 각 노

드에는 분리된 의사결정 관리자인 휴리스틱에 필요한 정보를 계산하기 위한 두개의 모니터가 존재한다. 하나의 모니터는 사용되고 나머지 하나는 백업용 이다. 낮은 오버헤드로 효율적으로 수행되며 지난 상태 정보의 존재에도 장점이 있지만, 효용(Utility)계산과 확률 분포가 집중화 되어 있어 분산 시스템의 크기가 커질수록 액티브 모니터(active monitor)의 경로에 통신 병목 현상이 생기는 문제점이 있다.

Wang [17] 은 프로세서들을 논리적으로 서버(Server)와 소스(Source)로 구분한 다음, 서버가 주도하여 자신에게 할당시켜 수행 하고자 하는 프로세서를 결정하는 기법과 프로세스가 발생한 소스에서 주도하여 그 프로세스를 수행할 수 있는 프로세서를 할당하는 기법으로 분류 하였다. 소스주도(Source initiative)기법은 새로운 프로세스가 발생 하거나, 또는 과부하 상태임이 밝혀질때 어떤 프로세서로 부하를 나누어 보낼 것인지를 결정하는 기법이고, 서버주도(Server initiative)기법은 자신의 부하가 매우 작다고 생각하는 프로세서가 그와 같은 사실을 다른 프로세서 들에게 알려줌으로써 부하를 나누어 받는 기법이다. Eager [5] 의 결과에 의하면 시스템의 부하가 중간 정도일때 소스 주도형이 더 좋고, 반면에 시스템의 부하가 높을때는 서버 주도형이 더 좋다고 알려져 있다.

본 연구는 시스템의 논리적인 링을 구성하고 이를 따라 상태정보를 담고 순환하는 VISITOR라는 메시지 패킷을 두어, 이를 기반으로 작업을 수행시킬 가장 적절한 노드를 선택하게 하였다. 또한 부하 균등화 정책으로 이를 소스 주도형의 알고리즘을 설계해서 시스템을 구현했으며, 이를 기존의 bidding 알고리즘에 의해 동작하는 시스템과 성능을 비교 평가하였다.

Ⅱ. 부하 균등화 시스템 설계

1. 부하균등화 모델을 위한 형식 기술

$N_i \in N$: 작업(Job)을 실행시키는 하나의 컴퓨터 시스템

$w \in W$: 작업부하(Work)

$d_i \in D$: 부하 균등화 결정(Load-balancing decision)

여기서 각 노드들은 서로 협력하며, 전체 시스템의 평균응답시간을 최소화 시키는데 목적을 둔다.

본 모델을 설명하기 위해서는 몇가지 가정이 필요한데, 그 내용은 다음과 같다.

① 모든 노드들은 소유하고 있는 자원들이 동일하고 처리 능력이 같은 이른바 동종의 시스템이다.

② 부하의 이주 단위는 작업(Job) 단위로 한다.

③ 작업들 간에는 상호간의 통신을 필요로 하지 않는다.

④ 어떠한 작업이나 임의 노드에서 수행이 가능하다.

2. 로컬 상태 및 갱신 모듈

로컬 노드의 CPU 준비 큐 길이를 주기적으로 관찰하여 현재의 로컬 노드의 부하값을 갱신하고, 상태 정보 교환 모듈로부터 받은 리모트 노드의 부하값 또한 갱신한다.

(그림 1)에서 만약 주기적인 관찰주기가 너무 빈번하게 되면, 이를 위한 오버헤드가 증가되고, 반대로 관찰주기가 너무 크면 상태정보의 불확실성의 증가로 인한 결정손실(Decision Loss)이 커진다. 따라서 시스템 상황에 따른 적당한 조절이 필요하다.

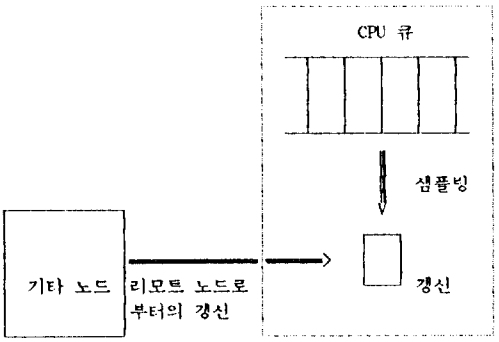

그림 1. 로컬 상태 관찰 및 갱신 모델

Fig. 1. Local state observation and updating model.

1) 추상화 상태 공간

각 노드의 각종 의사결정에 영향을 주는 혼합된 저수준 상태정보들은 어떠한 값(수)들이나 큰 상태 공간을 가지고 있으므로 이를 고 수준 상태(즉 추상화 상태)로 바뀌어야 한다. 이를 지식 추상화라 하겠다. 이를 위해 본 연구에서는 저 수준 상태 x_i를 추상화 상태 y_i로 바꾸어 주는 지식 추상화 함수 A가 필요하다. 지식 추상화 함수 A는 저수준상태 x_i중에 현재 CPU 준비 큐에 대기하고 있는 Job들의 수 x_i만을 고려하며, 고거 값을 반영한 값을 돌려준다.

$$y_i(t) = A(x_i(t)) = x_i(t) + x_i(t-1)/2 + x_i(t-2)/4 + \cdots$$

여기서 t는, 사실은 정확한 현 시점 t는 아니지만, 가장 최근에 샘플링 또는 타 노드에게서 전달된 값이

며, t-1은 그 전 시점에서의 값을 의미하며, t-2는 또 그의 전 시점을 의미한다. 과거 정보를 t-1 시점에서 1/2만큼, t-2시점에서 1/4만큼 반영한 이유는 과거 정보 반영시 가급적이면 함수 A의 수행시간을 줄일 수 있게 하기 위한 배려이다. (왜냐하면 1/2, 1/4등의 곱은 단 한번의 Shift operation으로 구현 가능하기 때문이다)그러므로 추상화 상태 Yi는

$$Y_i = \{y_i(t) | 0 \le y_i(t) \le y_i(t), \max\}$$ 이다.

2) 우선순위 리스트

로컬 상태 관찰 및 갱신 모듈에서 보유되어야할 또 하나의 정보는 (그림 2)와 같은 각 노드에 대한 우선순위 리스트이다. 본 연구에서는 각 노드는 동종의 시스템이라고 가정하였기 때문에, 우선순위 리스트를 시스템 토폴로지에 따른 노드 i에 대한 리모트 노드의 링크에 대한 가중값 인수(Weighting factor) 값에 그 기준을 두었다. 이 가중값 인수는 만약 모든 노드들간의 통신 비용이 다른 경우에는 일정 상수값들이 각기 상이하게 존재하지만, 느슨히 결합된 시스템 또는 견고히 결합된 시스템등에서는 각 노드들간의 직접통신일 경우 통신비용은 일정하다고 볼 수 있으므로, 이러한 토폴로지를 가진 시스템에서는 가중값 인수를 노드들간의 Hop 수로 정해질 수도 있다.

어쨌든 가중값 인수는 정적인 성질을 가지고 있으므로 시스템 초기 상태에 정의 되어진다.

노드 / 우선순위	1	2	...	i	n
0	1	2		i	n
1	2	3		i+1	1
2	n	1		i-1	n-1
3	3	4	...	i+2	2
.	.	.		.	.
.	.	.		.	.
.	.	.		.	.

그림 2. 우선순위 리스트
Fig. 2. Priority List.

3. 상태 정보 교환 모듈

(그림 3)에서와 같이 리모트 노드로부터의 상태 정보가 보내어지면 이를 받아 로컬상태 관찰 및 갱신 모듈에게 전달하고, 로컬 상태 관찰 및 갱신 모듈에게 로컬 및 리모트 상태들을 갱신하라는 신호를 보낸다.

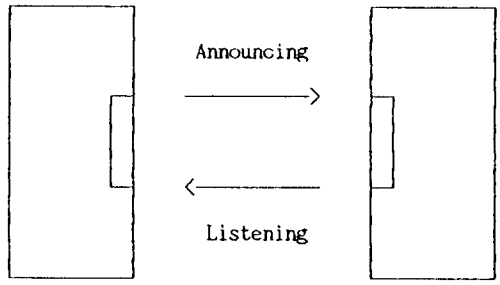

그림 3. 상태 정보 교환 모듈
Fig. 3. Status information switching module.

또한 비주기적으로 로컬노드의 부하값을 시스템상의 모든 노드의 상태 정보 교환 모듈에게 브로드캐스트(Broadcast)한다. 그리고 의사-결정 모듈로부터 부하 균등화를 위한 어떠한 조치가 행해지며, 대상 리모트 노드의 상태 정보 교환 모듈 및 로컬 노드의 로컬 상태 관찰 및 갱신 모듈에게 부하값 변경을 요청한다.

Announcing 및 Listening 주기는 비 주기적으로 행해지며, 이에 대한 주기 T*의 결정은 의사결정 모듈에서 행해진다.

4. 의사 결정 모듈

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 $\gamma \in R$과 퍼지 함수들에 의해서 행해지는데 규칙 (rule)의 추론은 퍼지 추론 (Fuzzy reasoning) 및 전진, 후진 연쇄 (Forward, Backward chaining)에 의해 의사 di ∈ Di가 결정되어 질 수 있다.

1) 의사 결정 모듈의 상태정보 퍼지 집합

① $low = \mu_{low}(y_i(t)) = 1-(y_i(t)/y_{max})^{1/2}$

② $high = (y_i(t)/y_{max})^2$

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

이 ① ~ ③ 을 그림으로 나타내면 다음 (그림 4)와 같다.

④ Threshold α

각 노드가 상태를 천이함에 있어서, 어느 기준 α 이상의 효용 차이가 있는 경우에만 상태 천이를 허용함으로써, 너무 잦은 상태천이로 인한 오버헤드를 막을 수 있다.

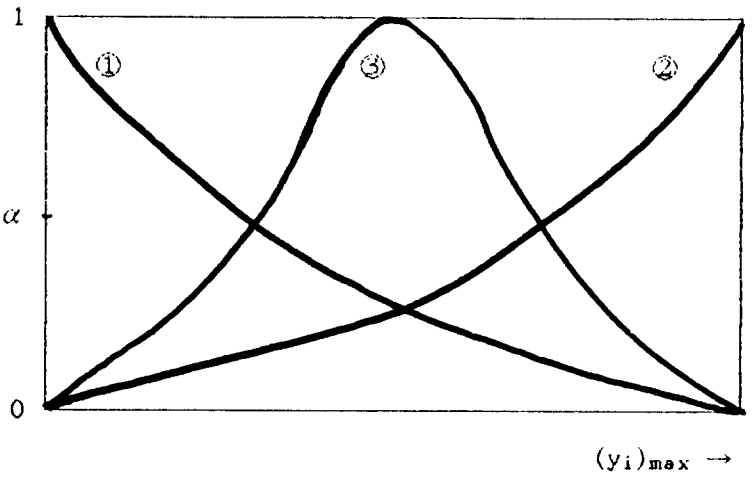

그림 4. 상태정보 퍼지집합
Fig. 4. Status Information Fuzzy Set.

⑤ Threshold κ

한 노드가 의사결정 과정에서 자신보다 높은 μ $Eu_{ij}(y_j(t^{(+1)}))$를 찾는 과정에서 Threshold κ 를 설정함으로써, 별 가치가 없는 작업 이주를 줄인다. 즉, 어느 한노드가 작업이주를 위해 임의의 적당한 노드 j를 선택하는데 있어서, 노드 i는 $t^{(+1)}$ 시점에서 가장 기대되는 유틸리티가 높은 멤버쉽값을 가진 노드(Max expected utility를 가진 노드)를 선택한다. 이때 그판단 기준은 $t^{(+1)}$ 시점에서의 μ $Eu_{ij}(y_j(t^{(+1)}))$이며 이 멤버쉽 함수는 과거 리모트 노드의 유틸리티 정보를 가지고, 좋은 의사결정을 해 주게하는 기준을 설명해 준다.

⑥ 노드 i에서 노드 j를 볼때 기대되는 유틸리티(μ $Eu_{ij}(y_j(t^{(+1)}))$ 함수의 정의)

i) $\triangle_j \leq \triangle_{max}$ 일때

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

ii) $\triangle_j \geq \triangle_{max}$ 일때

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

여기서 $\triangle_{ji} = \triangle^{(1)}{}_j + \triangle^{(2)}{}_j$ 이며, 그 의미는 $\triangle^{(1)}{}_j$는 상태정보를 획득 했을때까지의 과거 시간의 경과를 의미하며, $\triangle^{(2)}{}_j$ 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.

$\triangle_{max}$는 부하 균등화를 위한 상태정보의 불확실 정도 μ uc_{ij}가 1이 될때까지의 시간을 나타낸다.

2) 의사결정 규칙

① 만약 $y_j(t)$ = high ($1 \leq j \leq n$, $i \neq j$)이면, Job $S_i(t)$는 로컬에서 처리

② 만약 yii(t) = low이면, Job Si(t)는 로컬에서 처리

③ 만약 $Y_{ji}(t)$ = middle 또는 high 이면, 불확실성(Uncertainty) 및 Threshold κ 를 고려하여 미래시점에서의 기대 효용성 (Expected Utility)이 자신보다 높은 노드들을 구하고, 이들중 우선순위를 고려하여 가장 적당한 노드를 선택하여 작업(work)을 이주한다.

여기서의 우선순위란 노드간의 거리(Hop수), 노드들간의 Computing power, 또는 노드들의 구룹, 기타 여러 요인들이 우선순위에 포함되어질 수 있다.

3) 통신주기 결정

다음 상태정보 교환을 위해서, μ $Eu_i(y_i(t))$에 대한 통신주기 T^*를 우선순위를 고려하여 결정하고 상태정보 교환 모듈에게 통지한다.

여기서 주기 결정은 각 노드들간의 최종 기대되는 효용성들간의 평균에 대한 편차에 대한 함수값으로 결정되어지며, 이 결정지어진 주기 T^*를 가지고 상태정보 교환 모듈은 그 이전 주기 $T^*{}_{-1}$가 끝난 시점부터 T^* 시간이 경과한후 모든 노드들에게 현재의 상태 정보를 브로드캐스트 한다.

Ⅲ. 구 현

1. 구현 환경

본 연구에서 제시된 부하 균등화 기법은 IBM PC/AT들이 Ethernet으로 연결되어 구성된 분산 시스템에서 구현하였다. 대상 운영 체제는 UNIX운영체제를 IBM PC/AT상에 적합한 형태로 기본 구성을 이식하였고, 통신 계층은 Amoeba에서 구현한 통신 형태를 채택하였다. 통신 방식은 계층화된 다단계 접근방식을 취하는 프로토콜의 비효율성을 피하기위해 직접 통신방식을 사용하여 구현이 쉽고 프로세스간 통신을 위하여 소요되는 사전 준비시간이 적기때문에 빠른 응답시간을 보장한다. 통신용 보드는 웨스턴 디지탈사의 WD8003E모델이며 전송속도는 10Mbps이다.

전체적인 시스템 구성에 관한 모듈간의 블럭도는 (그림 5)과 같다.

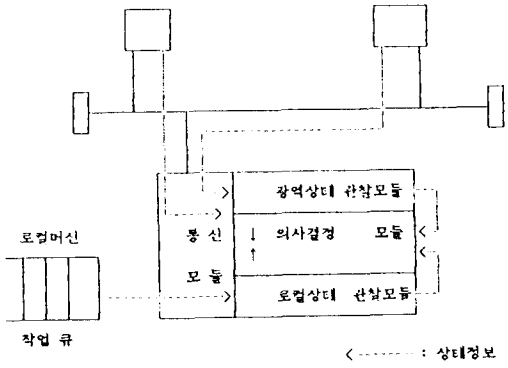

그림 5. 전체적인 시스템 구성 블럭도
Fig. 5. Block diagram of system configuration.

2. 로컬 상태 관찰 모듈

본 연구에서는 로컬 부하의 측정을 커널내에서 수행하였다. 주기적인 부하의 측정을 위해서 규칙적으로 발생하는 클럭 인터럽트를 이용한다. 본 운영 체제는 1초에 60회의 클럭 인터럽트가 발생하므로 클럭 인터럽트 처리 루틴에 일정 상수값을 정의하여 인터럽트가 발생할 때마다 그 값을 1씩 감소시켜서 그 값이 0이 될 때마다 부하 측정루틴을 호출하여 부하를 측정하고 다시 상수값을 정의하도록 하였다. 부하의 측정은 CPU 준비 큐의 길이를 조사하는 것으로 하였다. 각 노드의 커널 마다 CPU 준비 큐와 프로세스 테이블을 유지하고 있는데 이 프로세스 테이블에는 CPU 준비 큐의 시작을 가리키는 rdy head와 CPU준비 상태의 프로세스를 가리키는 포인터로 p_nextready가 있다. 따라서 CPU 준비 큐 길이의 측정은 rdy_head가 가리키는 프로세스 부터 연결된 프로세스의 p_nextready가 NULL일때까지 검사해서 그 길이를 구한다. 이러한 알고리즘은 다음과 같다.

```
if (rdy_head[CPU_READY_Q] !=NULL)
        { q_length = 1;
          next_p = rdy_head[CPU_READY_Q]->p_nextready;
          while(next_p != NULL)
                { q_length++;
                  next_p = next_p->p_nextready;
                }
        }
else q_length = 0;
```

3. 광역 상태 관찰 모듈

본 연구의 구현 대상 운영체제에서 프로세스들 간의 통신은 고정된 크기의 메세지를 서로 교환 함으로써 이루어진다. 메세지를 주고, 받기위해서 다음과 같은 프리미티브를 사용한다.

- send(caller, dest, msg_ptr) :
  dest로 메세지를 보내는데 사용
- receive(caller, source, msg_ptr) :
  source로 부터 메세지를 받는데 사용
- sendrec(caller, dest, msg_ptr) :
  메세지를 보내고 응답이 올때까지 대기함

통신에 사용되는 고정된 크기의 메세지 구조는 다음과 같다.

```
typedef struct {
    int m_source; /* 메세지를 보내는 프로세스 */
    int m_type; /* 메세지의 성격 */
    { /* 메세지 성격에 따라 필요한 여러가지 정보들 */
    } m_u;
} message;
```

(그림 6)에는 분산 IPC 구성 관계를 제시 했다. 분산 IPC는 같은 프리미티브에 의해 커널내의 프로세스들간 또는 지역망을 통한 프로세스들간의 통신이 이루어진다.

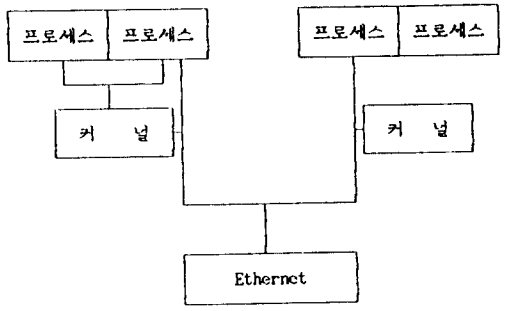

그림 6. 분산 IPC
Fig. 6. Distributed IPC.

Ethernet을 통한 IPC과정을 자세히 나타내면 (그림 7)과 같다.

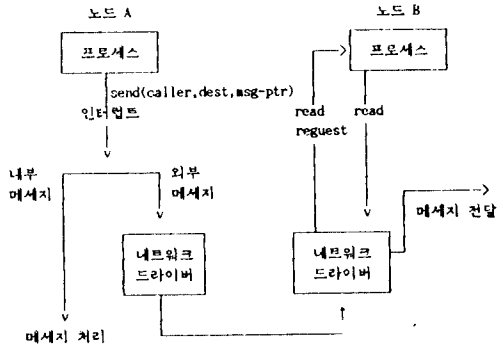

그림 7. 분산 IPC의 흐름
Fig. 7. Flow of Distributed IPC.

노드 A의 프로세스가 노드 B의 한 프로세스와 통신하고자 할 때 send 프리미티브 호출에 의해 소프트웨어 인터럽트로 IPC를 처리하는 루틴으로 간다. 이때 프리미티브의 메세지 타입을 통해 내부 통신인지 외부 통신인지를 검사하여 내부와의 통신이면 메세지를 복사해 주고 외부 통신이면 네트워크 드라이버로 보낸다. 그러면 네트워크 드라이버는 노드 B의 네트워크 드라이버에게 메세지를 보내고, 노드 B의 네트워크 드라이버는 해당 프로세스에게 메세지가 왔음을 알리는 request를 보낸다. 그 프로세스가 메세지를 받을 준비가 되어 있으면 네트워크 드라이버는 메세지를 서버로 복사해 준다.

네트워크 드라이버의 역할을 보다 자세히 살펴보면 다음과 같다. Ethernet을 통한 통신 메카니즘은 클라이언트 포트에서 서버 포트로의 요구 메세지와 서버 포트에서 클라이언트 프트로의 회답 메세지의 쌍으로 구성이 된다. 포트는 get_port와 put_port로 나타내어 지는데, get_port는 메세지를 받기 위한 것이고 put_port는 메세지를 보내기 위한 것이다.(그림 8)

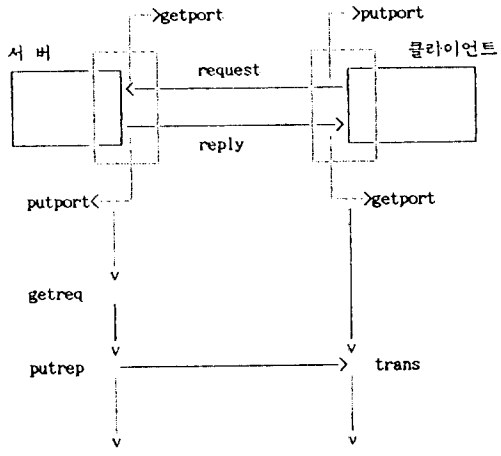

그림 8. 통신 포트
Fig. 8. Communication Port.

요구 메세지나 회답 메세지는 헤더(header)와 실제 데이타가 저장되는 부분으로 구성된다. 메세지의 몸체 부분은 전송하고자 하는 실제 데이타를 저장하는 버퍼 포인트와 버퍼의 길이가 명시된다. 이런 원격 통신을 위해 커널내에 구현된 프리미티브들은 다음과 같다.

getreq(hdr, buffer, size)

3개의 파라메터는 헤더, 버퍼, 버퍼 크기를 나타낸다. hdr 파라메터는 header structure에 대한 포인터로서 서버에게 listen할 포트를 지정할 수 있도록 허용한다. 버퍼 파라메터는 들어오는 메세지를 보관하기 위한 버퍼를 지정한다. 이 버퍼는 3번째 파라메터에 의하여 지정된 최대 바이트 크기를 보관할 수 있다.

putrep(hdr, buffer, size)

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

trans(hdr1, buffer1, size1, hdr2, buffer2, size2)

이 프리미티브는 두개의 다른 파라메터 집합을 가지고 있다. 첫번째 3개의 파라메터는 요청 메세지를 전송할 때 사용되고, 나머지 파라메터들은 응답 메세지를 받을 때 사용된다.

trans( )는 커널로 하여금 합당한 서버에 대한 요구를 보내게 되고 회답이 오기를 기다린다. 요구가 수행되고 난 후 서버는 클라이언트에 대해 putrep( )를 통해 회답을 보낸다.

본 연구에서는 위에서 설명한 바와같은 통신 메카니즘을 이해해서 즉, 버퍼에 노드 이름과 부하값을 저장

해서 주기적으로 부하 정보의 교환을 수행한다. 이때 교환 되는 메세지의 형태는 (그림 9)과 같다.

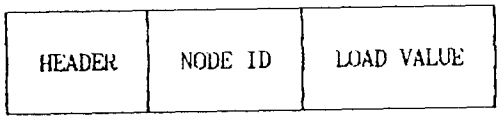

그림 9. 부하 정보 교환을 위한 메세지 형태
Fig. 9. message type for Load Information switching.

4. 의사 결정 모듈

부하 균등화를 위한 최적노드 선정을 위한 의사결정에 필요한 정보는 각 노드마다 보유하고 있어야 한다. 본 연구에서는 다음과 같은 테이블을 두어 필요한 정보를 저장하고 관리한다.

```
struct iproc{
int node_name; /* 노드명 */
int load_value; /* 부하 값 */
long load_time; /* 부하 측정 시간 */
char load_state; /* 부하 상태 정도 */
int alpha; /* 로컬 노드에서 node name이 명시하는
              노드로의 부하 이주횟수 */
float Util; /* 부하 값이 전달된 시점에서의 그 부하
               값에 대한 유용성 */
float EUtil; /* 부하 값을 사용하고자 하는 시점에서의
                그 부하 값에 대한 유용성 */
} Iproc[NR_NODES]
```

이 테이블에 유지되고 있는 정보를 기초로 하여 작업 수행에 가장 적합한 노드를 선정하는 알고리즘은 다음과 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_time and
                                        current time;
      calculate expected utillty for each node;
      select highest-utility node
      }
```

5. 글로벌 쉘

시스템이 가능될 때 수정된 명령어 해석기인 수정된 명령어 해석기인 글로벌 쉘(GSH)은 원격 수행에 적합한 작업들의 이름이 기록된 화일인 작업명 리스트를 읽어 들인다. 이 작업명 리스트를 글로벌 쉘 내에 삽입하지 않은 이유는 이 리스트에 새로운 작업을 첨가하고자 할때 이 리스트가 글로벌 쉘 내에 삽입되어 있을 경우 글로벌 쉘 자체를 수정, 재 컴파일링하는 작업을 해야 하기때문에 이를 방지하기 위해 따로 독립된 화일로 이 리스트를 유지하여 새로운 작업의 첨가시 추가 부담이 없도록 하였다.

이 리스트에 첨가될 수행시간이 짧은 작업들을 구별하기 위해서 time이라는 명령어를 수행했다. 수행 시키고자 하는 작업명 앞에 time이라는 명령어를 첨가하면 그 명령어가 수행되는데 걸리는 실제 시간과 CPU 점유시간을 측정할 수 있다. 이렇게 해서 조사된 작업의 CPU 점유 시간이 일정 시간 이상인 경우에만 작업명 리스트에 첨가 시킨다.

Ⅳ. 성능 평가

1. 시스템의 성능 향상률

부하 균등화 기법의 성능은 부하 이동에 따르는 오버 헤드와 부하 균등화를 통한 반응 시간의 이득으로 대비하여 볼 수 있다. 이때 프로세스 이전에 따른 오버 헤드로는 로컬 부하를 측정하는데 걸리는 시간과 부하 정보를 교환하는데 걸리는 시간, 최적 노드를 선정하는데 드는 시간, 또한 작업의 원격 수행으로 인한 오버 헤드가 있다.(표1)

로컬 부하를 측정하는데 걸리는 시간은 로컬 부하 측정 모듈을 호출하는 시스템 콜을 만들어 그 시스템 콜의 수행 시간을 로컬 부하 측정 시간으로 간주하였다. 이 시간은 사용자가 커널내에 포함되어 있는 실제 부하 측정 모듈을 호출하기 위하여 행하는 여러번의 메세지 전달 과정이 포함된 시간이기 때문에 실제로는 이 시간보다 짧은 시간이 소요될 것이다. 부하 정보의 교환에 걸리는 시간을 리모트 노드로의 메세지 전송을 위한 호출의 시작 시간으로부터 복귀 메세지 수신 시간까지를 측정한 호출/복귀 시간(round-trip time)으로 간주할 때 전송 데이타의 크기가 4 바이트일 경우 8msec가 걸린다. 이는 네트워크 상의 메세지 전송 시간과 프로토콜을 수행하는 시간, 클라이언트와 서버를 스케쥴링하는 시간을 내포하고 있다. 최적 노드 선정을 위한 의사 결정 시간의 측정도 의사 결정 모듈을 호출하는 시스템 콜을 만들어서 측정하였는데 실제 부하 균등화 기법을 수행 할 때에도 명령어 인식기

(GSH)에서 이 모듈을 같은 방법으로 호출 하므로 정확한 측정이라 할 수 있겠다. 같은 명령어를 로컬 노드에서 구행할 때 걸리는 시간과 리모트 노드에서 수행했을 때의 수행 시간 차이로 원격 수행으로 인한 오버 헤드를 측정했는데 약 804 msec로 상당히 크게 나타났다.

표 1. 부하 균등화로 인한 오버헤드
Table 1. Overhead due to Load balancing.

로컬 부하 측정	2 msec
부하 정보 교환	8 msec
최적 노드 선정을 위한 의사 결정	2 msec
원격 수행으로 인한 오버헤드	804 msec

일정한 예제 프로그램의 수행을 통하여 부하 균등화에 드는 오버헤드와 부하를 이전시킴으로써 얻을 수 있는 이득을 시스템의 평균 응답 시간을 지표로하여 실험적으로 측정, 부하 균등화 기법의 성능에 대한 타당성을 나타내었다.

5명의 사용자가 4개의 노드가 연결된 분산 시스템상에서 작업을 수행할 때 이들 사용자의 분포 상태를 변화 시키면서 부하 균등화 기법을 적용시켰을 때와 적용하지 않았을 때를 비교해 나타내었다. x축의 각 점들은 사용자의 초기 분포 상태를 나타내며 y축은 각 사용자들이 factorial 값을 일정한 수만큼 반복하여 구하는 예제 프로그램을 10차례 반복 수행했을 때의 평균 응답 시간을 나타낸다.(그림 10)는 시간에 따른 유용성의 변화만을 고려한 부하 균등화 기법과 부하 균등화 기법을 적용하지 않은 경우의 평균 응답 시간이다.

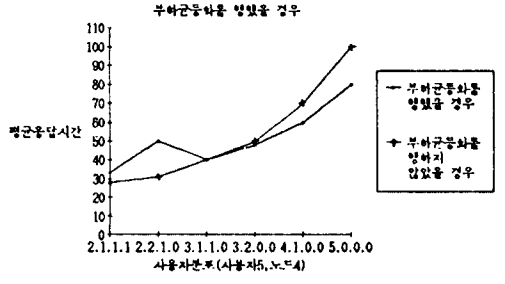

그림 10. 시스템의 평균 응답 시간
Fig. 10. Average response time of system.

그래프가 나타낸 것처럼 이 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 시간 변화에 따른 유용성의 변화를 이용해서 이러한 부하 집중 현상을 막을 수가 있다. 그러나 거의 비슷한 부하값을 갖는 경우에는 유용성의 변화 정도도 거의 같기 때문에 부하 이주가 특정 노드로 집중하게 된다.

(그림 11)은 우선 순위 노드를 사용해서 이런 현상을 해결하고자 한 것이다.

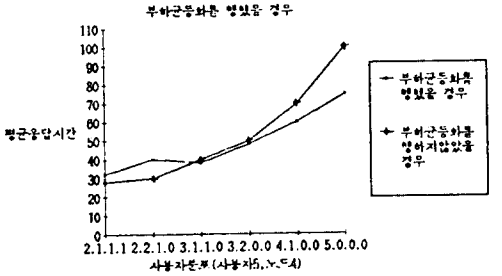

그림 11. 시스템의 평균 응답 시간
Fig. 11. Average response time of system.

이 그래프가 나타내는 것처럼 우선 순위 리스트는 과부하 노드가 여러개 존재할 경우에만 효과를 나타내고 한 노드로 많은 작업이 제출되는 경우에는 효과를 기대할 수 없다.

(그림 12)은 이런 경우에 대비하기 위해서 한 노드에 제출된 작업을 수행하기에 적합한 노드로 임의의 노드가 선정되면 다음에 그 노드의 유용성을 계산할 때 0.05씩 감소 시키는 방법을 사용했다.

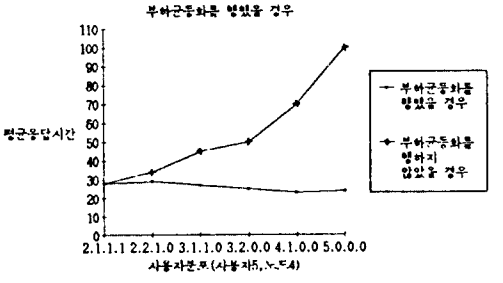

그림 12. 시스템의 평균 응답 시간
Fig. 12. Average response time of system.

이 방법을 사용한 결과, 한 노드가 갑자기 많은 작업이 제출되는 경우에 적절히 대처할 수 있으므로 평

균 응답 시간이 이전 보다 훨씬 짧아졌다. 이 방법은 특히 사용자의 분포가 (4,1,0,0)이나 (5,0,0,0)의 경우처럼 특정 노드에 작업제출의 집중현상이 심한 경우에 더욱 좋은 효과를 나타낸다.

본 연구에서 제시한 부하 균등화기법을 사용했을 경우에 얼마만큼의 시스템 성능이 향상되었는가를 살펴보기위해 다음과 같은 성능 향상률을 척도로 삼았다.

$$\text{성능 향상률}(\%) = \frac{(N-L)}{N} * 100$$

N : 부하 균등화를 행하지 않았을 경우의 평균 응답 시간

L : 부하 균등화 기법을 수행했을 경우의 평균 응답 시간

위의 공식을 이용해서 성능 향상률을 살펴본 결과 본 연구에서 제시한 부하 균등화 기법을 수행했을 경우 부하 균등화 기법을 수행하지 않았을 경우보다 35% 정도 시스템 성능이 향상되었다.

2. 타 기법과의 비교

부하 균등화 기법은 각 노드가 글로벌 상태 정보를 획득하는데 있어서 전적으로 메시지 교환에 의존하는 통신 지향(communication oriented) 방식과 불확실한 정보를 추론하여 상대적으로 메세지 교환량을 줄이는 계산 지향(computation oriented) 방식으로 구분할 수 있다. 본 연구에서 구현한 부하 균등화 기법은 계산 지향 방식이므로 통신 지향 방식의 대표적 알고리즘인 bidding 기법을 구현하여 그 성능을 비교하였다. 비교 방법은 시스템의 평균 응답 시간을 척도로하여 평가하였다.

bidding 알고리즘을 간략하게 살펴보면 다음과 같다.

> bidding 알고리즘 : 작업이 제출되면 시스템내의 모든 노드에게 수행에 관한 bid를 요구하는 메세지를 띄운다. bid 요구 메세지를 받은 모든 노드들은 요구 노드에게 bid를 띄운다. bid를 받은 노드는 최적의 bid를 선정하여 작업을 이주 시킨다.

(그림 13)는 두 기법을 사용하였을때의 평균 응답 시간을 나타낸 것이다.

본 실험에서는 구현 환경상의 한계성과 실험상의 제약으로 인하여 부하 균등화에 참여하는 노드의 갯수와 동일 시간에 제출될 수 있는 작업의 수에 제한(노드의 갯수는 4개, 동일 시간내에 제출될 수 있는 작업의 수는 5개)을 두었다. bidding 알고리즘은 노드수가 적고 발생되는 작업의 수가 적을 경우에 메세지 통신량이 적어지므로 본 실험 환경상에서 가장 좋은 성능을 나타낼 수 있다. 따라서 본 실험에서는 본 연구에서 제시한 부하 균등화 기법을 적용했을 경우가 bidding 알고리즘을 사용했을때에 비해 단지 2% 정도만의 성능 향상을 나타내고 있다. 그렇지만, 제출되는 작업수가 (그림 14)이 나타내는 것처럼 늘어날 경우와 노드수 작업수가 (그림 15)에서 보여주듯이 증가할 경우에 본 연구에서 제시한 부하 균등화 기법은 통신 메세지수의 변화폭이 완만한데 비해서 bidding 알고리즘의 경우 통신 메세지수가 급격히 증가하는 것으로 보아 노드수가 늘고 제출되는 작업수가 많을경우 그 성능 향상율이 점차 증가될 것으로 기대된다.

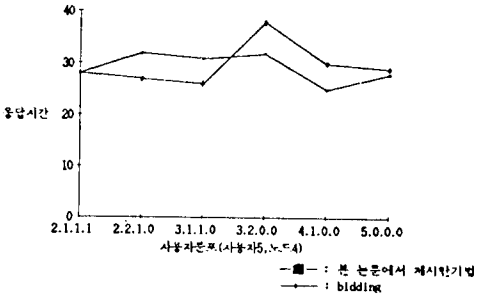

그림 13. 응답 시간 비교

Fig. 13. Comparision of response time.

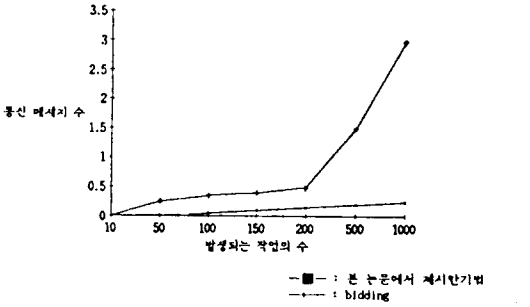

그림 14. 작업량 변화에 따른 통신메세지 수의 변화

Fig. 14. Variation of number of communication message upon workload.

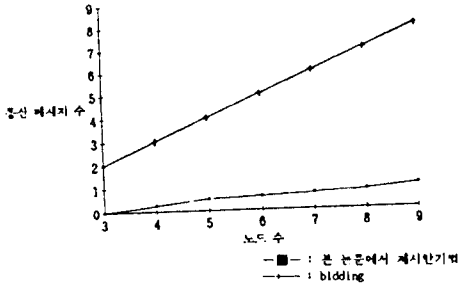

그림 15. 노드 수 변화에 따른 통신메세지 수의 변화

Fig. 15. Change of Node according to communication message.

V. 결 론

분산 시스템에서 부하 균등화 기법을 설계하고 운용하는 목적은 시스템내의 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.

본 연구에서는 부하 균등화 기법의 설계 및 구현시 시스템의 성능 저하를 가져오는 문제점들에 대한 해결 방법들을 제시하였다. 즉, 불확실한 부하 정보는 시간 변화에 따라 부하 정보에 대한 유용성을 변화시킴에 의해 완화시켰으며 특정 저부하 노드로의 부하의 집중 현상을 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 작업명 리스트를 사용하였다.

본 연구에서는 이러한 부하 균등화 기법을 Ethernet으로 연결된 IBM PC/AT상에 구현하였다. 구현 대상 운영체제는 UNIX의 기본 구성을 IBM PC/AT상에 이식하고 통신은 Amoeba의 통신형태를 이식한 운영체제이다. 본 연구에서 제시한 부하 균등화 기법을 구현한 시스템상에서 부하 균등화 기법을 운용하는데 드는 오버헤드를 측정하였고 부하 균등화 기법을 적용하였을 때와 부하 균등화 기법을 적용하지 않았을 때의 평균 응답 시간의 비교를 통하여 시스템의 성능을 비교해 보았을 때 부하 균등화 기법을 운용하는데 드는 오버헤드에도 불구하고 부하 균등화 기법을 적용했을 때 시스템의 성능이 35%정도 향상되었다. 또한 bidding 알고리즘을 적용했을 경우와도 예제 프로그램의 실험을 통해 시스템 평균 응답 시간을 비교했다. 본 연구에서 성능 평가를 위해 시행한 실험 환경이 그 한계성으로 인해 노드수가 적고 동일시에 제출될 수 있는 작업의 수에 제한이 있다. 이러한 환경은 bidding 알고리즘이 좋은 성능을 나타낼 수 있는 환경임에도 불구하고 본 연구에서 제시한 부하 균등화 기법이 2% 정도 좋은 성능을 나타내었고 노드수가 늘고 제출되는 작업수가 증가할 경우 bidding 알고리즘은 통신 메세지수가 급격히 증가하는데 비해서 본 연구에서 제시한 기법은 통신 메세지 증가 폭이 경미하므로 그 성능 향상율이 점점 커질 것으로 예상된다.

앞으로의 연구에서는 본 연구에서 제시한 부하 균등화의 성능을 보다 향상시키기 위해서 원격 수행에 의한 오버헤드를 줄일 수 있는 방법이 요구된다.

參 考 文 獻

[1] M. Livny, "The Study of Load Balancing for Decentralized Distributed Processing System", Ph. D. Degree thesis, Weizmann Institute of Science Rohovot, Israel, 1984.

[2] S. Zhou, D.Ferrari, "An Experimental study of load balancing performance", REPORT NO. UCB/CSD/87/336, Berkely, California, Jan.1987.

[3] D. L. Eager et al., "Adaptive Load Sharing in Homogeous Distributed Systems", IEEE Tran. on SE, vol. SE-12, no.5, pp. 662-675, May. 1986.

[4] P.Krueger, R.Finkel, "An Adaptive Load balancing Algorithm for a Multicomputer", Winsconsin Univ. Tech. Report,No.539, Apr.,1984.

[5] R. Alonso, "An Experimental Evalution of Load Balancing Strategies", REPORT No.CS-TR-112-87, Princeton Univ., Sep., 1987.

[6] 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.

[7] S.Zhou,D. Ferrari, "A measurement Study of Load Balancing Performance", IEEE 7th Conf5.on DCS, PP490-497, Sep.,1987.

[8] T.L. Cassavant, J.G.Kuhl, "A Taxonomy of Scheduling in General-purpose Distributed Computing Systems",IEEE Tran. on SE, vol.14,no.2,Feb.,1988

[9] C.H.Hsu,J. W. S. Liu, "Dynamic Load Balancing Algorithm in Homogeneous Distributed System", U.S.Army Report no. UIUCDCS-R-86-1261, 1986.

[10] L. M. Ni, Chong-wei xu and thomas B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing", IEEE Tran. on Software Engineering, vol. SE-11, no.10, PP.1153-1162, oct. 1985

[11] M.Scharo, K.Efe, L.Delcambre, S. Koppolue, "Heuristic Algorithms for Adaptive Load Sharing in Local Networks", proc. of tlu First Int's Conf5. on system Integration, PP. 762-770, Apr., 1990.

[12] 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.

[13] T.C.K.Chou and J.A. Abraham,"Load balancing in distributed Systems", IEEE Trans. on SE, vol. SE-8, no-4, JULY 1982, pp. 401-412.

[14] A. Kumar et al., "A Model For Distributed Decision Making: An Expert System for Load Balancing in Distributed Systems", Proc. of the 11th annual Int,l Comp. Software and Appl. Conf., Tokyo,Japan, pp. 507-513, Oct., 1987.

[15] J.Stankovic,"An Application of Bayesian Decision theory to Decentralized control of Job scheduling", IEEE Tran. on comp., vol. C-34, no.2, pp. 117-130, Feb., 1985.

[16] Barak, A. and Shillon, A., "A distributed load balancing policy for a multicomputer", soft pract. exper. pp 901-913, sept. 1985.

[17] YUXG-Terng Wang, and Robert J.T. Morris "Load sharing in distributed Systems", IEEE Trans. on computer, vol. C-34, no.3, March 1985.

[18] Chi-Yun Huang Hsu, Jane W. S. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems", IEEE the 6th International Conf. on Distributed Systems, pp.216-223, May, 1986.

[19] T. L. Cassavant, "Analysis of Three Dynamic Distributed Load Balancing Strategies with Varying Global Information Requirements", The 7th International Conf. on Distributed Computing System, pp. 185-192, Sept, 1987

[20] A. K. Ezzat, "Decentralized Control of Distributed Processing Systems", Ph. D. Degree thesis, New Hampshire Univ., Dec., 1982.

[21] T. L. Cassavant and J. G. Kuhl, "A Formal Model of Distributed Decision-Making and its Applications to Distributed Load Balancing", IEEE Tran. on Computers, pp.232-239, 1986.