분산시스템에서 상태 정보 추론을 이용한 그룹 부하 균등 알고리즘

정진섭* · 이재완*

Group Load Balancing Algorithm Using State Information Inference in Distributed System

Jin-Seob Jeong* · Jae-Wan Lee*

요 약

분산 시스템에서 전체 시스템의 부하 균형을 이루어 시스템의 성능을 향상시키는 것이 주요 목표 중 하나이다. 시스템간의 부하를 균등하게 함으로써 처리기의 가동률을 높이고 작업 반환 시간도 줄일 수 있다. 본 논문은 지식 기반 메카니즘을 이용하여 각 노드에서 과거 및 현재의 정보를 기반으로 추론한 미래 부하 상태 정보를 서로 공유하여 최적의 부하 균등화를 이루는 의사 결정 규칙과 정보 교환 규칙을 설계하였다. 성능 평가 결과 각 노드의 가동률이 균등해지고 처리 속도의 향상을 보였으며, 시스템의 신뢰성과 가용성이 향상되었다. 본 논문에서 제안한 기법은 분산 운영 체제의 부하 조절 알고리즘 설계에 활용될 수 있다.

ABSTRACT

One of the major goals suggested in distributed system is to improve the performance of the system through the load balancing of whole system. Load balancing among systems improves the rate of processor utilization and reduces the turnaround time of system. In this paper, we design the rule of decision-making and information interchange based on knowledge based mechanism which makes optimal load balancing by sharing the future load state information inferred from past and present information of each nodes. The result of performance evaluation shows that utilization of processors is balanced, the processing time is improved and reliability and availability of systems are enhanced. The proposed mechanism in this paper can be utilized in the design of load balancing algorithm in distributed operating systems.

키워드

Distributed system, Load balancing, state information, rule of decision-making.

*군산대학교 전자정보공학부 박사과정

**군산대학교 전자정보공학부 교수

접수일자 : 2002. 10. 15

I. 서 론

통신망에 의해 서로 연결되어 있는 자율적인 컴퓨터들의 집합인 분산시스템은 여러 개의 처리노드로 구성되어 있으며, 각 처리기는 지역메모리(local memory)를 갖고서 상호 연결된 네트워크를 통하여 정보를 교환할 수 있는 시스템이다. 또한 단일 시스템과는 달리 하나의 노드가 결함이 발생해도 전체 시스템에는 큰 손상을 가져오지 않는 특성을 가지고 있기 때문에 신뢰성과 가용성(availability)을 동시에 지니고 있다. 그러나, 잠재적인 처리 용량을 많이 가지고 있는 분산시스템은 대부분의 처리 용량이 낭비되고 있으며[1] 다른 곳에 위치한 사용자들이 시스템의 자원들을 통신망에 의해 공유할 수 있기에 분산시스템에서 전체 시스템의 부하 균등을 이루는 것은 가장 중요한 사항이 되었고, 이를 위해서 작업들 중 일부를 적절히 분배시키는 부하 균등 기법(load balancing scheme) 등이 수없이 연구되어 왔다.

부하균등 기법은 작업 전송을 요구하는 처리기의 부하 상태에 따라 송신자 주도 탐색(sender-initiative probing), 수신자 주도 탐색(receiver-initiative probing), 대칭 탐색(symmetric probing) 방식으로 분류한다.

송신자 주도 탐색은 과부하 상태로 바뀌어진 노드가 저부하 상태인 노드를 찾아서 그 노드로 부하를 이주시키기 위한 탐색이며, 이에 반해 수신자 주도 탐색은 저부하 상태로 바뀌어진 노드가 과부하인 노드를 찾아서 그 노드로부터 부하를 이주받기 위한 탐색이다. 대칭탐색 방식은 이들을 혼합(hybrid)한 기법으로 모든 시스템의 부하 상태에서 상기 두 가지 방식에 비해 우수한 성능을 보인다[2]. 하지만, 대칭적 알고리즘은 무분별한 탐색으로 인해 그렇지 않아도 과부하 상태인 노드에 불필요하게 많은 탐색메시지가 도착함으로써 더욱이 그 시스템을 과부하로 만드는 단점이 있다[3].

분산시스템은 각자의 정보를 기반으로 하여 수행하는 분산 의사-결정자(Distributed Decision-Maker)들의 모임으로 생각할 수 있는데[4], 본 논문에서는 분산시스템이 확대됨에 따라 증가하는 부하 상태 정보를 효과적으로 획득, 시스템의 상태 변화를 즉각적으로 예측하여 새로운 환경에 능동적으로 대처할 수 있도록 하고[5][6] 정확한 이주의 결정으로 의사-결정이 불일치하는 빈도수를 줄일 수 있는 방법[7]에 초점을 맞추어 연구가 진행되었으며 이러한 불일치의 해결 방안으로 분산된 제어 메카니즘을 지식 기반 분산 자원 관리자(Knowledge-Based Distributed Resource Manager)가 수행토록 하였다. 각 의사 결정자는 보다 적은 통신을 행하면서 적은 비용으로 노드들에 대한 상태 정보를 추론할 수 있어야 한다. 따라서 각 노드간의 조정은 전적으로 통신에 의존하기보다는 추론된 정보를 공유함으로서 얻어질 수 있는 메카니즘을 구상하였다. 또한 시스템의 상태에 따라 임계치를 정하여 임계치 이하의 작업은 지역 노드에서 처리하고 임계치 이상의 작업들만 부하조절기에서 처리하게 하여 적당한 노드를 찾기 위해 소요되는 탐색 비용과 통신비용을 줄였다.

II. 그룹 부하균등 시스템 설계

1. 시스템 구조

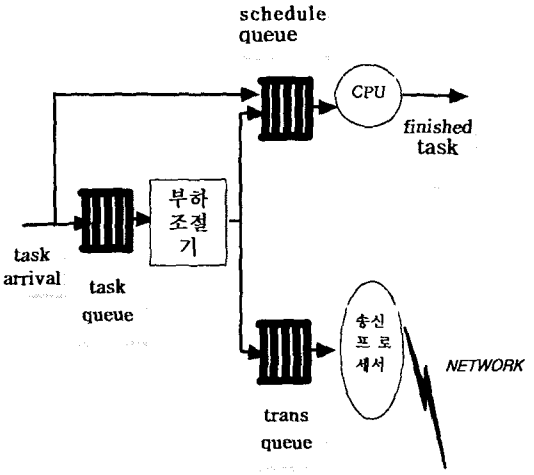

그림 1. 시스템 모델
Fig 1. System Model

시스템을 구성하는 각 노드는 처리 능력과 속도가 서로 다른 환경을 갖는 시스템으로 가정하였으며 통신 네트워크에 연결된 각 노드는 여러 그룹으로 편성되고 각 그룹은 주변 m개의 노드들과 한 개의 모니터 노드

로 구성되며 스케줄 큐(SCHEDULE-QUEUE)와 이주 큐(TRANS-QUEUE)를 가지고 있다.

2. 부하조절기 설계

다양하고 이질적인 분산 환경에서 통신 지연이나 각 노드들의 다양한 시스템 성능을 고려하여 그림과 같이 정확한 이주에 대한 적절성의 결정을 내릴 수 있는 부하조절기를 설계한다.

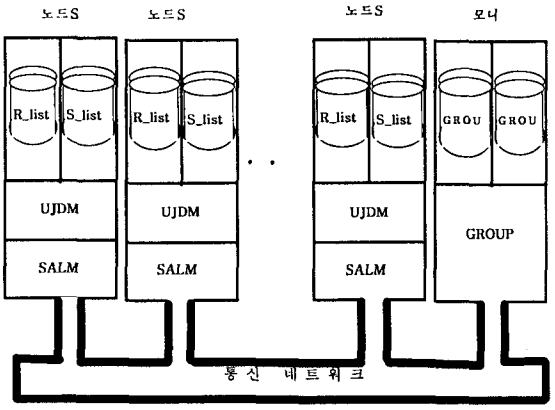

그림 2. 그룹내 부하 조절기 구조도
Fig 2. Structure of Load Controller in Group

시스템을 구성하는 각 노드는 처리 능력과 속도가 서로 다른 환경을 갖는 시스템으로 가정하였으며 컴퓨터 통신망에 연결된 인접한 노드들을 묶어 하나의 그룹으로 만들고, 각 그룹은 인접한 m개의 노드들과 한 개의 모니터 노드로 구성된다. 그룹내 여러 노드들 중 임의의 한 노드가 과부하이고 인접 노드들이 저부하일 경우 과부하 노드의 프로세스를 저부하 노드들로 이주하여 부하균형을 이룬다. 그룹간 부하균형은 각 그룹의 모니터 노드들끼리의 통신을 통해 그룹들의 부하 정보를 얻어 그룹간 부하 균형을 이루도록 하고 통신 지연 시간을 고려해 그룹간 부하 균형은 분산형으로 부하 정보를 수집하게 하였다 [8][9][10].

3. 그룹내 부하 조절

노드 Si는 상태정보 교환 모듈(State Announcing & Listening Module: SALM)과 갱신 및 작업 결정 모듈(Update & Job Decision Module: UJDM)을 가지고 있으며 수신자 리스트와 송신자 리스트 테이블을 가지고 있다. 그림 3은 각 모듈간 상호 관계 및 역할을 나타낸다.

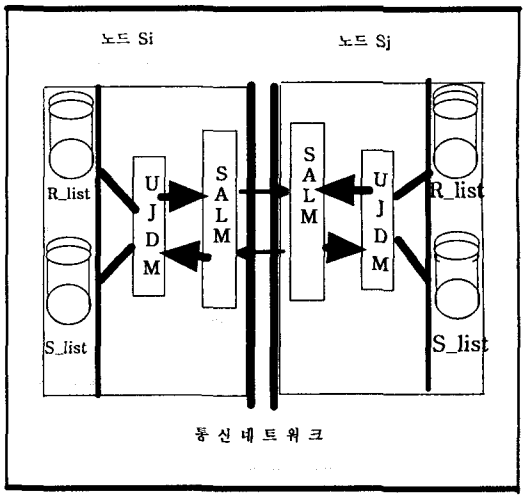

그림 3. 부하 조절기의 각 모듈간 상호 관계
Fig 3. Inter-relationship between Module in Load Controller

인접해 있는 노드들을 묶어 만들어진 그룹내 통신에서는 각 노드의 상태 정보 교환 모듈이 서로의 부하 정보를 수집하게 하였다.

(1) 상태 정보 교환 모듈

상태 정보 교환 모듈은 그룹내 리모트 노드들로부터 주기적으로 상태 정보를 받아 갱신 및 작업 결정 모듈에게 전달해 준다. 또한 로컬 노드의 상태 정보를 판단하고 예측하여 그룹내 리모트 노드들의 상태 정보 교환 모듈에게 주기적으로 전달하는 역할을 한다. 또한 모니터 노드와 통신하면서 자신의 부하 상태가 과부하 또는 저부하 상태가 되었을 때, 자신의 상태를 모니터 노드에게 알린다.

그룹내 리모트 노드로부터의 상태 정보가 보내어지면 이를 받아 갱신 및 작업 결정 모듈에게 전달하고 로컬 및 리모트 상태들을 갱신하라는 신호를 보낸다.

또한 주기적으로 로컬 노드의 부하값을 그룹내 모

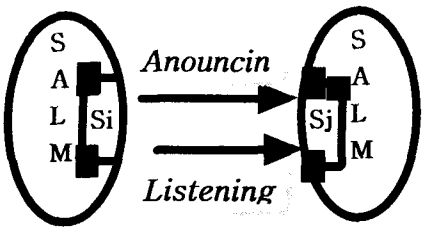

그림 4. 상태 정보 교환 모듈
Fig 4. Module of State Information Interchange

든 노드의 상태 정보 교환 모듈에게 방송한다. 그리고 부하 상태 관리자로부터 부하 균등화를 위한 어떠한 조치가 행해지면, 그룹내 대상 리모트 노드의 상태 정보 교환 모듈에게 부하값 변경을 요청한다.

상태 정보 교환 모듈은 가장 중요한 모듈로서 과거 및 현재의 정보를 가지고 미래의 상태를 추론하는 역할을 한다. 다른 노드의 현재 상태는 통신 지연 시간으로 인해 불확실하고, 자신의 정보 또한 주기적인 샘플링 때문에 즉, 그 사이에 값이 변할 수 있기 때문에 불확실하다. 그러므로 로컬 및 글로벌 상태의 과거 정보를 반영하고 또한 현재의 불확실한 상태의 정보를 평가해야 한다. 뿐만 아니라 이들 정보를 바탕으로 미래의 글로벌 상태를 평가하여 추측해야 될 것이다. 따라서 상태 정보 교환 모듈은 지식 기반 시스템 기법을 사용해서 이러한 문제를 해결하는 매카니즘이다.

상태 정보 교환 모듈은 로컬 및 그룹내 리모트 노드의 과거 및 현재의 상태 정보를 가지고, 각 노드의 미래 상태를 추론한다. 또한 이 추론 결과에 의하여, 미래의 시스템 상태를 추론한다.

상태 정보 교환 모듈은 로컬 노드 및 그룹내 리모트 노드의 과거 및 현재의 부하값을 가지고 각 노드의 상태 정보를 만든다. 또한 이 상태 정보를 기반으로 일련의 규칙을 가지고 전체 시스템의 미래 상태를 예측하는 가설을 만들고, 이 가설이 시스템의 부하 불균형을 의미하면, 취해야 할 행위를 결정하는 후진 추론 방식을 택했다.

(2) 상태 정보 교환 모듈의 상태 정보

$$\tau_{(n,m)} = \alpha LD_{(n,m)} + (1-\alpha)\alpha LD_{(n,m-1)} + (1-\alpha)^2 \alpha LD_{(n,m-2)} + \cdots + (1-\alpha)^j \alpha LD_{(n,m-j)} + \cdots + (1-\alpha)^m \tau_0$$

상태 정보 교환 모듈이 유지하는 상태 정보는 다음과 같다.

$\tau_{(n,m)}$은 node n의 m시점에서 만든 m+1시점의 상대적인 부하 예측값이다.

■ $\tau_{(n,m)} = \alpha LD_n + (1-\alpha)\tau_n$

α값에 대해 다음과 같이 정의한다.

$0 \le \alpha \le 1$

$\alpha = 0$ 일 때 $\tau_{(n,m)} = \tau_n$으로 최근의 부하값은 무시되고 과거값만 반영되고,

$\alpha = 1$일 때 $\tau_{(n,m)} = LD_n$으로 과거의 값은 무시되고 최근값만 반영된다.

α 와 (1-α)가 모두 1보다 적거나 같기 때문에 각각의 후속되는 항목은 앞의 것보다 가중값이 적다.

m: 현재의 event가 일어난 시점

LD(n,m): m시점에서 노드 n의 실제 부하값

LD(n,m-1): m-1 시점(바로 전 상태)에서의 노드 n의 과거 부하값

LD(n,m-j): m-j 시점에서의 노드 n의 과거 부하값

τ_0: 상수(전반적인 시스템의 부하 평균값)

(3) 상태정보 교환모듈의 상태정보 해석규칙

① 만약>OVERLOAD_FACTOR이면
LD(n,m)은 앞으로 과부하

② 만약 UNDERLOAD_FACTOR< $\tau_{(n,m)}$≦ OVERLOAD_FACTOR이면 LD(n,m)은 앞으로 적정부하

③ 만약 $\tau_{(n,m)}$≦UNDERLOAD_FACTOR이면 LD(n,m)은 앞으로 저부하

(4) 상태정보 교환모듈의 상태정보 추론규칙

위의 세 가지 규칙이 주어지면 각 노드의 상태를 결정하기 위해 상태 정보 교환 모듈은 다음의 추론 규칙을 구성한다.

① 만약 $\tau_{(n,m)}$가 증가하는 상태이고(즉, $\tau_{(n,m)}$> 0이고) LD(n,m)이 저부하이면 앞으로 LD(n,m)은 적정부하가 된다.

② 만약 $\tau_{(n,m)}$가 증가하는 상태이고 LD(n,m)이

적정부하이면, LD(n,m)은 앞으로 과부하가 된다.

③ 만약 $\tau_{(n,m)}$가 감소하는 상태이고(즉, $\tau_{(n,m)} < 0$이고) LD(n,m)이 과부하이면 앞으로 LD(n,m)은 적정부하가 된다.

④ 만약 $\tau_{(n,m)}$가 감소하는 상태이고 LD(n,m)이 적정부하 이면, LD(n,m)은 앞으로 저부하가 된다.

상태 정보 교환 모듈은 GROUP_STAGE가 균형인 경우는 아무 일도 하지 않는다. 그러나 만약 GROUP_STAGE가 불균형 상태이거나 또는 불균형 상태가 앞으로 될 것이라고 판단되면, 이를 어떻게 균형으로 만들 것인가를 결정한다.

⑤ 만약 $\tau_{(n1,m)} - \tau_{(nn,m)}$이 감소하는 상태이고, GROUP_STAGE가 불균형이라면, 앞으로 GROUP_STAGE가 균형이 되도록 한다.

현재 GROUP_STAGE(n)가 불균형이라면

$\tau_{(n1,m)} - \tau_{(nn,m)}$을 감소시켜야 한다. 이를 감소시키고자 할 때,

⑥ ACT_LB(n,m)을 행해야 $\tau_{(n1,m)} - \tau_{(nn,m)}$이 감소하는 상태가 될 수 있다.

상태 정보 교환 모듈은 ACT_LB(n,m)을 행해야 문제에 대한 해결이 가능하다고 추론하며 자신의 부하 상태와 다른 노드의 부하 상태를 비교해서 불균형 상태이거나 앞으로 자신의 부하가 증가하는 불균형 상태가 될 때 부하 조절을 이루도록 한다.

4. 그룹간 부하 조절

(1) 모니터 노드의 상태 정보 교환 모듈

노드 Si의 부하 상태들을 상태 정보 교환 모듈들로부터 전달받아 Group의 부하 상태를 측정하며 다른 그룹의 모니터 노드로부터 요구가 있을시 그룹의 시스템 상태를 알려주는 역할을 담당하며 그룹간에 송신 또는 수신할 노드의 주소를 통보해 프로세스 이주 메카니즘을 가동시켜 부하 조절을 이루도록 하는 본 시스템의 중추 신경의 역할을 담당한다.

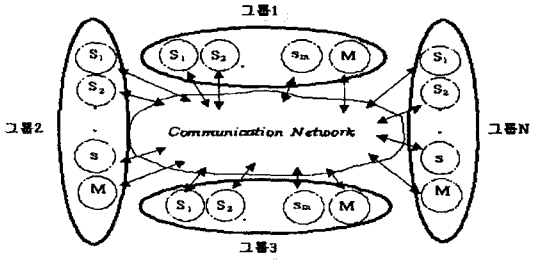

그림 5. 그룹간 부하 조절기 시스템 모델
Fig 5. System Model of Load Controller between Group

(2) 모니터 노드에서의 Group 상태 정보 교환 모듈의 상태 정보 해석 규칙

$\tau_{(G1,m)}$은 Group1 에서 m시점에서 만든 m+1 시점의 상대적인 평균 부하 예측값

$\tau_{(n1,m)}$: node 1 에서의 부하 예측값

$$\tau_{(G1,m)} = (\tau_{(n1,m)} + \tau_{(n2,m)} + \cdots + \tau_{(nn,m)}) / n$$

$\tau_{(n2,m)}$: node 2 에서의 부하 예측값

$\tau_{(nn,m)}$: node n 에서의 부하 예측값

① 만약 $\tau_{(G1,m)} - \tau_{(Gn,m)} > C + N$ 이면, SYSTEM_STATE는 불균형이다.

C : 상수 (UNBALANCE 기준값)

N : Threshold 값 (네트워크 및 시스템 상태에 따라 시스템 관리자가 정함)

② 만약 $C + N \geq \tau_{(G1,m)} - \tau_{(Gn,m)} \geq C$ 이면,

SYSTEM_STAGE는 m 시점에서의 SYSTEM_STAGE 상태를 그대로 유지한다.

③ 만약 $\tau_{(G1,m)} - \tau_{(Gn,m)} < C$ 이면,

SYSTEM_STAGE는 균형이다.

④ 만약 $\tau_{(G1,m)} - \tau_{(Gn,m)}$ 이 증가하는 상태이고,
SYSTEM_STAGE가 균형이라면,
SYSTEM_STAGE는 앞으로 불균형이 될 것이다.

본 논문에서는 후진 추론 방법을 채택하였다. 이는 전진 추론에 비해 상대적으로 탐색하는 양이 작을 뿐 아니라 구현하기도 용이하기 때문이다. 상태 정보 교환 모듈은 SYSTEM_STAGE가 균형인 경우는 아무 일도 하지 않는다. 그러나 만약 SYSTEM_STAGE가 불균형인 상태이거나 또는 불균형이 앞으로 될 것이라고 판단되면, 이를 어떻게 균형으로 만들 것인가를 결정한다.

⑤ 만약 $\tau_{(G1,m)} - \tau_{(Gn,m)}$이 감소하는 상태이고, SYSTEM_STAGE가 불균형이라면, 앞으로 SYSTEM_STAGE가 균형이 될 것이다.

현재 SYSTEM_STAGE(n)가 불균형이라면

$\tau_{(G1,m)} - \tau_{(Gn,m)}$을 감소시켜야 한다. 이를 감소시키고자 할 때,

⑥ ACT_LB(G1, n)을 행하면 $\tau_{(G1,m)} - \tau_{(Gn,m)}$은 감소하는 상태가 될 수 있다.

상태 정보 교환 모듈은 ACT_LB(G1, n)을 행해야 문제에 대한 해결이 가능하다고 추론하며 자신의 부하 상태와 다른 노드의 부하 상태를 비교해서 불균형 상태이거나 앞으로 자신의 부하가 증가하는 불균형 상태가 될 때 부하 조절을 이루도록 한다.

5. 갱신 및 작업 결정 모듈

갱신 및 작업 결정 모듈(UJDM)은 상태 정보 교환 모듈이 프로세스 이주 메카니즘을 가동시키기 위하여 작업의 선발을 요구하게 되면 이주 큐에 제출되어 있는 작업들 중의 하나를 선발하여 선발된 작업의 ID를 상태 정보 교환 모듈에게 통보해 주는 역할을 담당하고 수신자 리스트 또는 송신자 리스트 테이블을 유지 및 관리하는 역할을 담당하는 모듈이다.

각 프로세스는 Migration Factor(MF)를 갖는다. MF는 프로세스에 대한 정보를 추출할 때 계산되며 계산 방법은 아래와 같다. 이 값이 0 에 가까울수록 이주시 적절한 프로세스로 간주된다.

$$MF = \frac{\text{이주비용} + \text{원거리 실행비용}}{\text{로컬 실행 비용}}$$

작업 결정 모듈은 수신 또는 송신할 프로세서의 주소를 통보받아서 프로세스 선발을 하고 상태 정보 교환 모듈로 하여금 통보받은 목적지 노드의 주소와 선정된 프로세스의 ID로서 프로세스 이주 메카니즘을 가동시켜 부하 조절을 이루도록 한다

III. 알고리즘

임의의 시간에 노드 Si의 SALM(상태 정보 교환 모듈)이 자신의 부하 상태가 과부하 또는 저부하가 될것으로 판명될 경우, 자신의 부하 상태를 인지함으로써 가동되는데 과부하 노드일 경우 송신자 노드가 되며, 저부하 노드일 경우 수신자 노드가 된다. 또한 그룹간 알고리즘의 가동 시기는 임의의 시간에 그룹 Gi의 평균 부하 상태가 과부하나 저부하가 될 것으로 판명될 경우, 모니터 노드가 자신의 그룹 부하 상태를 다른 그룹에 알림으로써 가동된다.

1. 그룹내 부하균형 알고리즘

임의의 시점에 노드 Si의 SALM은 자신의 부하 상태 변화를 탐지하고 다른 노드 Si의 SALM에게 자신의 부하 상태 정보를 알린다. 만약, 과부하일 경우에 SALM은 프로세스를 전송할 목적지 노드의 주소를 응답 받고, UJDM(작업 결정 모듈)에 전송할 프로세스의 선발을 요구하며, UJDM은 이주 큐에 제출되어 있는 작업들 중의 하나를 선발하여 목적지 노드의 주소와 선발된 프로세스의 ID로써 프로세스 이주 메카니즘을 가동시킴으로 종료된다. 만약 저부하일 경우에 UJDM은 자신의 송신자 리스트를 탐색하여 부하가 가장 많은 노드의 주소(Si)를 목적지

노드로 결정하여 SALM으로 하여금 응답 메시지를 요청한 Si노드에게 보내도록 한다. 그리고 송신자 리스트 Si의 부하 정보를 갱신한다. 만약 송신자 리스트의 모든 부하 정보가 같을 경우에는 리스트의 맨 선두에 있는 노드의 주소를 목적지 노드로 선정하여 응답한다.

```
Si(process_id)
{
  timer_start(Si);

   states=SALM( τ(n, m));
  SALM(Si,process_id, States);
}
  function SALM( τ(n, m));
  {
   if( τ(n, m) > OVER_FACTOR)
      {States= OVERLOAD;}
   elseif (UNDERLOAD_FACTOR < τ(n, m)
          AND τ(n, m) <= OVERLOAD_FACTOR)
          {states=Busy;}
   elseif ( τ(n, m)); <= UNDERLOAD_FACTOR)
          { states=Underload;}
  return states;
}
  function SALM(Si,process_id, states)
  {
   if (states == OVERLOAD or states == UNDERLOAD)
       {
        ACK = Monitor_states_(Si, process_id, states, timer(start));
        if(ACK and states == OVERLOAD)
          { MOVEprocess(UJDM());}
        else if(timer(out)) { Exit;}
       }
  }
  function UJDM()
  {
    MF=(process_trans_cost+process_execute_cost)/Local_execute_cost;
    process_id=sort_ascend_first(MF);
    return process_id
  }
```

그림 6. Si 노드측 알고리즘
Fig 6. Algorithm of Si Node

2. 그룹간 부하균형 알고리즘

Si 노드들로부터 부하 상태 변화를 통보 받은 모니터 노드의 GROUP SALM은 통보 받은 부하 상태들을 체크하여 자기 그룹의 평균적 부하 값을 예측한다.

모니터 노드의 GROUP SALM은 그룹의 시스템 상태를 예측하여, 앞으로 그룹내 노드들의 전체적인 부하 평균치가 과부하 되어 그룹내에서 부하 조절을 위한 적당한 목적지 노드를 선정치 못할 경우 그룹간 부하 조절을 위해 그룹의 모니터 노드는 자기 그룹의 부하 상태가 과부하가 될 것임을 다른 그룹에게 알려 각 그룹 모니터 노드의 송신자 리스트에 갱신하도록 한다. 방송을 수신한 그룹 Gj 모니터 노드의 GROUP SALM은 그룹의 시스템 상태를 파악하여 자신의 부하 상태가 저부하가 될 것이라면 응답을 한다. 이때 이주 큐에 제출되어 있는 작업이 있다면 이주 큐에 있는 프로세스를 그룹 Gj의 저부하 노드 Si 에 전송한다. 파악된 시스템 상태를 기반으로

```
GROUP SALM_state( τ(Gi, m), states, timer(start))
          {
           if(states == OVERLOAD)
            {
             target_address = GROUP SALM(select(R_list));
           if(target_address = NULL)
            }
             Broadcast(Gi, OVERLOAD);
           GROUP SALM(S_list, target_address, update);
             return target_address;
          }
          Function  Group_monitor(Gi, process_id, Group_states);
           { if(Group_states==OVERLOAD)
            {
              Do timer(start){
              Collect_replies();
              } while(timer(out));
            }
          Function  Group_monitor(Gj, process_id, Group_states);
         { if(Group_states==UNDERLOAD)
            {
              Do timer(start){
                if (Load(S_list) == NULL)
                  { Send_message(Gn, Idle);
                    SALM(R_list, Si, Update);
                    Group_state=UNDERLOAD;
                    Broadcast(Gi, UNDERLOAD);
                    do Timer(start) {
                    Group_monitor(Gj,Process_id, Group_states))
                  }
                }
              } while(timer(out));
          Function Group_monitor(Gi, process_id, Group_states);
          { if(GroupLoad == OVERLOAD)
            Select_Destination_Group();
            Select_process(Monitor_TransQ, process_id);
            {
              reply_information();
              Do timer(start){
                  receive_migrated_job();
                  Insert_Group_Monitor_SchQ(process_id);
                  Exit();
              } while(timer(out));
             Exit();
             }
           }
          }
```

그림 7. 그룹 모니터 알고리즘
Fig 7. Group Monitor Algorithm

송신자 리스트에 있는 그룹들의 부하 정보를 비교한다. 만약 송신자 리스트가 비어 있거나 그룹 Gi의 부하가 송신자 리스트의 부하들보다 많다면, 모니터 노드는 그룹 Gi를 송신자로 결정하고 송신자 리스트에 있는 송신자 그룹 Gi의 모니터 노드에게 부하 조절을 이루도록 메시지를 전송한다. 송신자 리스트의 선택은 송신자 리스트에 있는 노드중에서 부하량이 가장 많은 그룹이 선택된다. 부하량이 같을 경우에는 송신자 리스트의 가장 선두에 있는 그룹을 선택한다. 만약, 송신자 리스트가 텅빈 상태라면 자신의 그룹이 저부하임을 다른 그룹에 알림으로 다른 그룹의 모니터 노드 수신자 리스트에 그 그룹에 대한 주소와 부하 정보를 등록하고 그룹간 부하 조절을 이루도록 한다. 프로세스 이주가 없으면 부하 조절을 종료하고 과부하 그룹으로부터 프로세스가 이주해 오면 그룹 Gj의 모니터 노드의 스케쥴 큐에 삽입하고 부하 조절을 종료한다. 응답을 수집한 그룹 Gi의 모니터 노드는 수집된 정보를 기반으로 적당한 그룹을 선발하여 그 그룹의 모니터 노드에 전송하여 그룹간 부하 조절을 이루도록 한다.

IV. 성능 분석

본 연구에서 제시한 그룹 부하균등 알고리즘의 성능을 분석하기 위하여 여러개의 노드를 한 그룹으로 묶어 각 노드마다 과부하 상태 임계치 상수인 과부하 상수를 다르게 주어서 여러 그룹으로 편성하였으며 시뮬레이션 횟수마다 모델링에 필요한 가정들은 똑같이 하고 시스템의 부하율만 증가시켜 처리하는 과정을 보여주었다.

시뮬레이션 처리 결과를 비교하여 보면 프로세서의 양이 과부하 상수보다 아주 적을 때(시스템의 부하율이 70% 이하)에는 처리시 비적용과 적용의 시간의 차가 별로 크지 않는데, 프로세서의 양이 점차 많아질수록(시스템의 부하율이 70% 이상) 처리 시간의 차이가 점점 더 벌어지는 것을 확인할 수 있다. 같은 환경에서 프로세서를 처리하는데 있어서 부하균등 비적용과 적용의 시간차가 현저하게 나타나 있음을 볼 수 있어 처리 시간 효율에서 부하처리과정을 적용시킨 부분이 우월하다는 것을 알 수 있다.

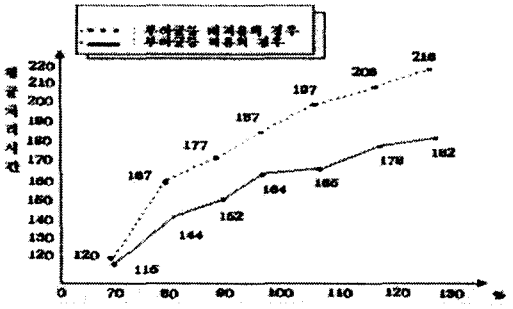

그림 8. 분산시스템의 처리 시간 비교

Fig 8. Comparison of Processing Time of Distributed System

그림 8에서 가로축은 시스템 노드들의 부하율로서 시뮬레이션 횟수가 늘어날수록 점차 처리양이 증가함을 나타내었고 시스템 노드들의 평균 소요시간을 처리 시간별로 나누어 세로축에 표시하고 부하균등 비적용(점선)과 적용(실선)의 그래프로 나타내었다.

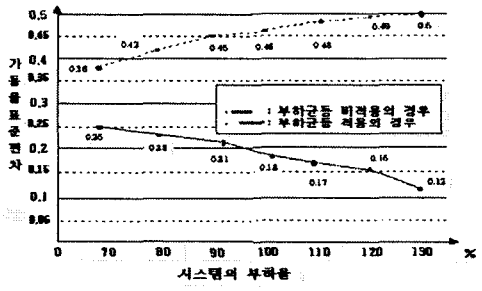

그림 9. 시스템 가동률 표준편차 비교

Fig 9. Comparison of Standardization Deviation on the Rate of System Operation

그림 9는 모든 그룹의 각 노드별 가동률(노드별 프로세서 할당양/Overload 상수)을 시스템의 부하율에 따라 아래 공식에 의해 노드 가동률 표준편차로 나타낸 것이다.

평균 가동률 = 노드들의 가동률 합/노드의 수

가동률 편차 = 노드의 가동률 - 평균 가동률

$$\text{가동률 표준 편차} = \sqrt{\frac{(\text{가동률편차})^2\text{의합}}{\text{노드의수}}}$$

시스템의 부하율을 가로축에 놓고 이에 따른 시스템 노드의 가동률 표준편차를 세로축에 표시하여 최적의 시스템 안정도 0을 기준으로 하여 부하균등 비적용(점선)과 적용(실선)의 그래프로 나타내었다. 위의 시뮬레이션한 결과를 분석하면 다음과 같은 사항을 유추해 낼 수 있다. 할당되어진 프로세서를 수행함에 있어 초기에 시스템 부하율 70%에서 시작하여 점차적으로 부하율 130%까지 증가시켰는데, 부하율이 증가할수록 알고리즘을 적용시킨 경우, 시스템 각 노드에 프로세스가 균등하게 할당되어지므로 각 노드의 값이 균등해지고 가동률이 균등하게 되어 가동율 표준편차가 안정도 0으로 분포되어져 감을 볼 수 있어 시스템은 보다 안정적으로 나아감을 알 수 있다.

V. 결 론

분산시스템에서 사용되지 않는 처리 용량을 다른 사용자에게 제공하여 노드의 이용률을 개선하고 작업 반환 시간도 줄임으로써 전체 시스템의 부하 균형을 이루어 시스템의 성능을 향상시키는 것이 가장 중요한 현안이다. 전체 시스템 내에서 처리되어야 할 작업 부하를 각 프로세서에 균등하게 배정함으로써 지리적으로 분산된 자원을 최대로 활용하고 시스템을 효율적으로 운영할 수 있는 것이다. 각 노드들이 과거 및 현재의 정보로 미래의 상태를 추론하고 미래의 시스템 상태를 추론한 상태 정보를 가지고 서로 정보를 공유하여 부하조절을 이룸으로서 효과적인 방법을 보였다. 또한 통신망에 연결된 인접한 노드들을 묶어 그룹을 이루고 그룹내에서 서로의 상태 정보를 가지고 부하 조절을 이루게 하고 그룹 안에 있는 모니터 노드로 하여금 그룹간 정보를 공유하게 하여 그룹간 부하조절을 이루게 하였다. 시뮬레이션을 통해 성능을 분석한 결과 그룹 부하균등 알고리즘을 적용했을 때 비적용 때보다 시스템이 대단히 안정적이고 처리에 있어서 뛰어난 성능 향상을 보였다.

향후 그룹간 알고리즘에 관한 연구가 더 진행되어야 하겠고 시뮬레이션 환경 설정을 확장하여 실제 분산시스템 환경에서 실행될 수 있도록 검증하는 연구가 더 진행되어야 하겠으며 앞으로 초고속통신망을 통한 분산시스템 환경에서 사용자들에게 속도 지연이 없고 중단이 없는 다양한 멀티미디어 서비스를 제공하는 무결성 시스템이 될 것이다.

참고문헌

[1] P. V. McGregor and R. R. Boorstyn, "Optimal Load Balancing in a Computer Network", Proc. Int. Conf. on Commun., Vol. 3, IEEE, New York, pp.41.14-41.19. 1975

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

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

[4] M. Singhal, "On the Application of AI in Decentralized Control: An Illustration by Mutual Exclusion", 7th DCS Conf., pp.232-239, 1987.

[5] K. B. Mahieddine, P. M. Dew, and M. Kara, "A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed System", IEEE, Proc 14th International Conf. on Distributed Computing Systems, Jun. 1994.

[6] M. Alanyali and B. Hajek, "Analysis of Simple Algorithms for Dynamic Load Balancing," Math. Operations Research, Vol.22, No.4, 1997.

[7] F. Berman, R. Wolski, S. Figueira. J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Network," Proc. Supercomputing, 1996.

[8] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," PhD thesis, Univ. of California, Berkeley, Sept. 1996.

[9] M. Mitzenmacher, "On the Analysis of Randomized Load Balancing," Theory of Computing System, Vol.32, 1999.

[10] M. Mitzenmacher, "How Useful Is Old Information?", IEEE Trans. on Parallel and Distributed System, Vol.11, No.1, Jun, 2000.

저자소개

정진섭(Jin-Seob Jeong)

2001년 2월 군산대학교 정보통신공학(공학석사)

2001년 3월～현재 군산대학교 전자정보공학부 박사과정

※관심분야: 분산시스템, 컴퓨터 네트워크

이재완(Jae-Wan Lee)

1984년 중앙대학교 전자계산학과 졸업(학사)

1987년 중앙대학교 대학원 전자계산학과(석사)

1987년 중앙대학교 대학원 전자계산학과(박사)

1996년 3월～1998년 1월 한국학술진흥재단 전문위원

1992년～현재 군산대학교 전자정보공학부 부교수

※관심분야: 분산시스템, 운영체제, 네트워크등