論文 94-19-6-4

병렬 프로그램 실행을 위한 ELI 기반 동적 부하 균등화

正會員 裵 仁 漢*

An ELI-based Dynamic Load Balancing for Parallel Program Executions

Ihn Han Bae* *Regular Member*

要 約

본 논문에서 분산 시스템의 각 노드들은 노드들간에 주기적으로 교환되는 시스템 상태 정보를 히스토리에 저장하고, 그 히스토리 정보에 Newton의 후향 보간법의 5차 보간 다항식을 사용하여 다음 주기의 예측 부하 지수(ELI)를 계산한다. 계산되어진 ELI를 동적 부하 균등화 시스템의 로케이션 정책에 이용하였다. 그 결과 ELI 기반 동적 부하 균등화 시스템은 기존의 부하 균등화 알고리즘에 비해 성능이 우수함을 시뮬레이션을 통하여 알 수 있었다.

ABSTRACT

In this paper, we have studied load balancing problems in distributed systems. The nodes of distributed systems exchange periodically system state information each other. The information is stored in history. Based on the information, we compute an expected load index(ELI) using a five-degree interpolating polynomial in Newton's backward difference interpolation formula. A new location policy of dynamic load balancing systems makes use of the ELI. We show that its performance is better than that of the existing load balancing algorithms through a simulation study.

I. 서 론

분산 시스템에서 태스크들은 life time 동안에 다수의 단계 : 태스크 정의(task definition), 태스크 할당(task allocation), 태스크 스케줄링(task scheduling), 태스크 이주(task migration)를 거쳐서 처리된다.[1] 본 논문에서는 처리기들과 통신망의 부하 변화에 따라 태스크들을 처리기로 동적 재할당하는 태스크 이주 문제에 대해 연구한다.

분산 시스템에서 병렬 처리를 위해, 프로그램들은 태스크 할당에 의해 다수의 태스크들로 분할되어 분산된 처리기에 적재되어진다. 각 노드가 이러한 태스크를 실행할 때, 초기 프로세스는 많은 차일드 프로세스를 생성할 수 있다. 이러한 태스크들의 실행 조

*曉星女子大學校 電子計算學科
Dept. of Computer Science, HyoSung Women's Univ.
論文番號 : 93230
接受日字 : 1993年 12月 4日

건은 자료에 종속적이고 예측할 수 없으므로 좋은 태스크 할당만으로 좋은 부하 균등화를 이룰 수 없다. 그러므로 분산 자원을 효율적으로 이용치 못하여 태스크 응답 시간이 늦어진다. 따라서 분산 자원의 이용율을 향상시키기 위해서는, 실행시 생성된 프로세스들을 분산된 처리기에 균등하게 매핑하여야 한다.[17] 이러한 동적 부하 균등화는 분산 시스템의 상태 변화에 적응적이어야 하므로, 불확실한 상태에서의 시스템 상태판단과 의사 결정을 포함한다.[13] 즉, 처리할 프로세스가 많은 노드는 프로세스를 유휴한 노드로 이주시키기 위해서 다른 노드들의 상태를 판단하여 목적 노드를 결정한다. 그러므로 의사 결정을 위한 시스템 상태 판단은 동적 부하 균등화의 성능을 좌우한다. 만일 각 노드에서 판단한 리모트 노드들의 상태와 그 리모트 노드들의 실제 상태가 같다면, 최적인 부하 균등화를 이룰 수 있다. 이런 목적을 만족시키는 가장 간단한 방법은 각 노드에서 프로세스 이주가 발생할 때 마다, 각 노드는 리모트 노드들에게 시스템 상태 정보에 관한 요청 메세지를 보내고 응답 메세지를 받는 것이다.[12] 그러나 이 방법은 많은 통신 비용 때문에 오히려 태스크의 평균 응답 시간을 더 느리게 만들 수 있다. 다른 방법은 각 노드들은 시스템 상태 정보를 주기적으로 교환하여, 이러한 정보를 기초로 시스템 상태 판단을 하는 것이다. 그러나 이 방법 역시 각 노드에서 예측한 리모트 노드들의 시스템 상태와 실제 리모트 노드의 시스템 상태 사이에는 여전히 오차가 존재한다.[12] 따라서 적은 통신 비용으로 저부하의 리모트 노드를 찾아내는 효율적인 부하 공유 방법이 연구되었다.[8,12,17]

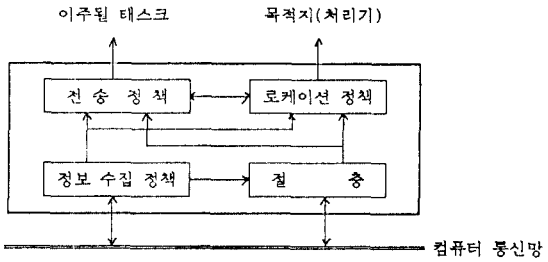
본 논문에서는 분산 시스템의 각 노드들은 시스템 상태 정보를 주기적으로 교환하고, 이 정보를 기초로 저부하의 리모트 노드를 효율적으로 탐색하는 예측 부하 지수(Expected Load Index : ELI)를 이용한 동적 부하 균등화 방법을 제시한다. 여기서는 분산 시스템의 각 노드에서 예측한 리모트 노드들의 부하와 현재 리모트 노드들의 부하의 차이를 줄이기 위하여 SSE(System State Estimater)를 동적 부하 균등화 시스템에 도입하였다. 이를 위하여 분산 시스템의 노드들은 주기적으로 시스템 상태 정보를 교환하며, 각 노드의 SSE에서는 주기적으로 교환되는 리모트 노드들의 시스템 상태 정보를 히스토리에 저장하고, 그 히스토리에 저장된 정보에 보간법을 적용하여 다음 주기의 리모트 노드들의 시스템 상태를 예측한다. 각 노드는 리모트 노드들의 예측 부하 지수를 사용하여 부하 균등화의 로케이션 정책을 결정한다. 본 논문에서는 ELI 기반 동적 부하 균등화 시스템을 설계하고 시뮬레이션을 통하여 그것의 성능을 평가한다.

Ⅱ. 부하 균등화

2.1 구성 요소

부하 균등화 문제는 "처리기들 사이에 작업 부하를 균세하기 위하여 네트워크로 연결된 처리기들간에 프로세스를 어떻게 분배할 것인가?"하는 것이다. 즉, 프로세스 이주를 통하여 처리기간에 시스템 작업 부하를 분배하는 분산 스케줄링 정책의 한 분야이다.[9]

부하 균등화를 위한 다수의알고리즘들이 제시되었다.[2,7,8,12,14,15,17] 이러한 알고리즘들은 일반적으로 (그림 1)과 같은 4개의 기본 요소로 구성되어 있다.[9]



(그림 1) 부하 균등화 알고리즘의 구성 요소

(1) 정보 수집 정책

시스템 상태 정보(정보의 종류와 양)를 저장하고 이 정보(처리기 부하, 큐 길이, 프로세스 정보, 시스템 부하의 히스토리)를 관리한다. 정보 수집 정책은 전송 정책과 로케이션 정책에 정보를 제공한다.

(2) 전송 정책

이 구성 요소는 프로세스가 로컬에서 또는 리모트에서 실행되는지를 결정한다. 이 결정은 태스크 패러미터, 프로세스 부하, 그 외의 정보에 기초하여 만들어진다.

(3) 로케이션 정책

이 구성 요소는 프로세스가 옮겨질 적당한 처리기를 결정한다.

(4) 절 충

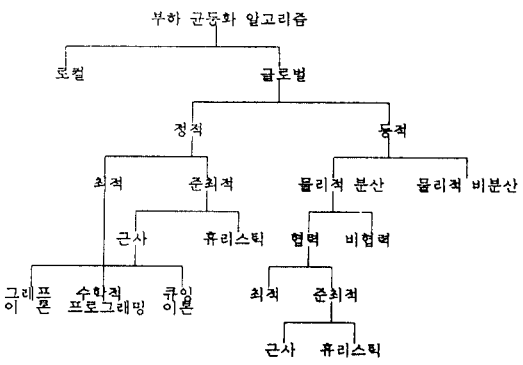
분산 부하 균등화 알고리즘에서 위의 3가지 구성 요소는 다른 처리기의 유사한 구성 요소와 분리될 수

없다. 그것들은 일반적으로 메세지 전달을 사용하는 절충 구성 요소를 통하여 주기적 또는 비주기적으로 상호 작동한다.

2.2 분 류

부하 균등화 알고리즘의 계층적 분류는 (그림 2)와 같다. 정적 부하 균등화는 모든 프로세스가 실행되기 전에 처리기를 찾는데 사용된다. 프로세스가 실행되었다면 처리기로 부터 제거되지 않고, 프로세스 전송 결정은 결정적 혹은 확률적으로 만들어진다. 전체 분산 시스템의 현재 상태는 고려되지 않는다. 정적 부하 균등화의 장점은 시스템 상태 정보가 관리될 필요가 없기 때문에 간단하고 결정을 하기 전에 작업 부하가 충분히 잘 설명될 때 효과적이다. 그러나 정적 부하 균등화는 시스템 부하 변동의 조절에 실패하고 자원 이용율도 나쁘다.[9]

동적 부하 균등화는 프로세스 생성 및 소멸에 따라 동적으로 각 처리기의 부하를 같게한다. 그러한 알고리즘들은 프로세스가 실행되었어도 리모트 처리기에 이주할 수 있다. 이주의 목적은 분산 시스템에 존재하는 모든 프로세스를 위한 최적 처리기를 찾는데 있다. 결정은 현재 시스템 상태에 관한 정보를 사용하여 만들어진다. 부하 균등화에 대한 동적 해를 찾는 것은 정적 해를 찾는 것 보다 더욱 복잡하다. 그것은 시스템 상태 정보의 수집과 관리를 요구한다.[9] 본 논문에서 제안하는 동적 부하 균등화 방식은 현재 시스템 상태 부하 지수를 이용하여 프로세스 이주 결정을 하기 때문에 성능이 우수하다.



(그림 2) 부하 균등화 알고리즘의 계층적 분류[3]

2.3 관련 연구

부하 균등화에 대한 이전의 연구를 살펴보면 다음과 같다. Stankovic은 부하 분배 결정을 만들기 위하여 컴퓨터간에 협력을 요구하는 글로벌·동적 알고리즘을 제시하였다.[15] 해는 준최적이고 Bayesian 결정 이론에 기초한 휴리스틱 방법을 이용하여 구해진다. 이 알고리즘은 중앙 모니터 장치에 의해 지원되는 물리적 분산 부하 균등화 알고리즘이다. 모니터가 하는 일은 상태 정보를 대조하고 환경 변화를 개개의 컴퓨터에 알린다. 이 알고리즘은 두가지 흥미있는 특징을 가지고 있다. 첫째, 모니터의 고장은 전체 시스템의 성능에 크게 영향을 주지 않는다. 왜냐하면 이 시스템은 두개의 모니터가 있는데 항상 하나의 모니터가 작동하고, 단지 첫번째 모니터가 고장일 때 두번째 모니터가 사용되기 때문이다. 둘째, 부하 할당 기능은 두개의 부분 : decentralized job scheduler(DJS)와 decentralized process scheduler(DPS)로 나누어 진다. 분산된 다수의 객체로 구성된 DJS의 역할은 각 컴퓨터에 대기하는 작업의 개수를 거의 같게 하고 네트워크 상태에 계속 적응하기 위하여 네트워크에 대한 상태 정보를 관리한다. DPS는 DJS의 대기 큐로 부터 활성화되는 작업을 결정한다. DPS는 작업의 다중 모듈, 그들의 자원 요구사항, 클러스터링 관계, 그리고 프로세스 실행 동안의 할당을 처리한다.

Ramarithm은 데드라인과 자원요구를 갖는 태스크를 스케줄링하기 위한 휴리스틱 알고리즘을 연구하였다.[12] 여기서는 협동할 수 있는 4가지 알고리즘 : random scheduling, focused addressing, bidding, flexible 알고리즘들을 평가하였다. 시뮬레이션에서는 서로 상대적인 알고리즘들의 성능 뿐만 아니라 두가지 기초 알고리즘과도 비교하였다. 첫번째 기초 알고리즘은 로컬에서 처리할 수 없는 태스크는 어떤 다른 노드로 보낼 수 없는 noncooperative 알고리즘이고, 두번째 기초 알고리즘은 bidding 알고리즘처럼 정확한 행동을 취하는 이상적인 알고리즘이나 통신 비용은 발생치 않는 것이다. 그리고 통신 지연, 태스크 laxity, 노드간의 부하 차이, 태스크 처리 시간이 알고리즘 성능에 어떠한 영향을 미치는가를 조사하였다. 그 결과 분산 스케줄링은 실시간 환경에서도 효율적이고, flexible 알고리즘의 성능은 focused addressing과 bidding 알고리즘보다 우수하여 이러한 알고리즘들의 성능은 시스템 상태 정보에 좌우된다는 것을 알 수 있었다.

Efe는 다른 서버들의 부하 수준을 결정하는데 사

용되는 제어 메세지로 부터 발생하는 비용을 최소화 하는 두가지 알고리즘을 개발하였다.[8] 첫번째 알고리즘은 노드중의 하나를 조절기(controller)로 지정한다. 그 조절기는 어떤 외부 태스크를 받지 않는다. 서버들은 처리할 수 없는 태스크들을 조절기에 보낸다. 태스크를 받았을 때, 조절기는 자신의 서비스 큐를 검사하고 그 서비스 큐내의 태스크 개수가 임계값보다 큰 full 상태가 아니면 그 태스크 실행을 시도한다. 만일 조절기의 서비스 큐가 full이면, 그 태스크는 다른 서버에 전송하기 위하여 조절기의 전송 큐에 넣어진다. 서버는 조절기로 부터 받은 태스크의 재전송을 허용치 않는다. 조절기는 태스크 전송할 곳을 선택하기 위하여 서버 이름들의 queue data structure(QDS)를 관리한다. 조절기가 태스크 전송을 원할 때 마다, 그것은 QDS내의 선두 원소를 조사함으로써 목적지 서버를 결정한다. 태스크가 서버에 전송되면, 목적지 서버의 이름은 QDS의 끝으로 간다. 조절기가 서버로 부터 태스크를 받았을 때, 그 서버의 이름 역시 QDS의 끝으로 간다. 즉, 조절기는 전송 태스크를 그 네트워크상에서 가장 예전에 태스크를 보냈거나 받은 노드에 전송한다. 두번째 알고리즘에서는 서버가 조절기에 직접 태스크를 전송하지 않는다. 대신에, 그것은 조절기에 최적 목적지를 문의하는 짧은 제어 메세지(probe)를 보낸다. 그런 probe 메세지가 도착하면, 조절기는 그 서버의 이름을 QDS의 끝으로 옮긴다. 그 때, 그 요청에 대한 응답 메세지를 생성하고, 이것을 전송 큐에 삽입한다. 전송 태스크를 위한 최적 목적지를 결정하기 위하여, 조절기는 먼저 자신의 서비스 큐를 검사한다. 만일 그 서비스 큐가 full이 아니면, 그것은 자신을 최적 목적지로 선정한다. 만일 조절기의 서비스 큐가 full이면, 전송 태스크를 위한 목적지는 QDS에서 선두 원소를 조사함으로써 결정되어진다. 그 엔트리는 QDS의 끝으로 옮겨진다.

Xu는 중앙 감독자에 의해서 노드간의 빈번한 부하 정보 교환을 피하는 동적 부하 균등화를 위한 4가지 휴리스틱 방법을 제안하였다.[17] 제안된 많은 부하 균등화 방법들은 과부하 노드를 결정하기 위하여 고정된 임계값을 사용하였다. 만일 이 임계값이 너무 작다면, 지나친 부하 균등화 활동으로 발생되는 스래싱(thrasing)은 성능을 저하시킨다. 만일 그 임계값이 너무 크다면, 효율적인 부하 균등화를 수행할 수 없다. 이 연구에서는 주기적으로 임계값을 수정하기 위하여 중앙 감독자의 조정을 받는 적응적 모델을 제안하였다. 멀티컴퓨터의 노드들간에 시스템 상태를 교환하는 시간 간격인 time window를 정의하였다. 그 time window는 전체 시스템이 안정 상태에 들어가면 길어지고, 전체 시스템 부하가 급격히 변하면 time window는 짧아진다. 제안된 방법의 성능 평가를 위해 32-node Intel iPSC/2 하이퍼큐버 컴퓨터상에서 프로토타입 동적 부하 균등화 시스템을 개발하였다. 프로세스 이주를 위해 localized round-robin (LRR), global round-robin(GRR), localized minimum load(LML), global minimum load(GML) 휴리스틱 방법들이 사용되었다. 실험 결과 RR 방법은 시스템 크기가 클 때 우수하고, ML 방법은 시스템 크기가 작을 때 우수함을 알 수 있었다.

Sevensson은 글로벌 실행을 고려할 가치가 없는 일시적(short-lived)인 작업을 탐지하기 위한 필터를 제안하였다.[14] 히스토리라 부르는 필터는 작업 이름과 이전의 실행에 기초한 통계를 사용하여 일시적인 작업을 탐지한다. 여기서는 세가지 종류의 필터 : history, optimal, random을 제시하였다. 추적 방식 시뮬레이션은 두가지 초기 할당 알고리즘 : shortest, central이 필터에 의해 어떤 영향을 받는지를 평가하는데 사용되었다. 히스토리 필터의 성능은 optimal 필터의 성능에 거의 근접함을 알 수 있었고, 히스토리를 사용함으로써 평균 작업 부하 비율이 필터를 사용치 않는 것과 비교할 때 상당히 감소되고 있음을 보이고 있다.

Dandamudi는 분산 대기 큐 구조의 성능에 대한 태스크 스케줄링 정책의 영향을 연구하였다.[7] 특히, 적응적 정책이 사용될 때 분산 대기 큐 구성의 성능과 집중 대기 큐 구성을 비교하였다. 태스크 스케줄링은 크게 두가지 : oblivious 정책, adaptive 정책으로 분류할 수 있다. Oblivious 그룹에 속하는 정책들 즉, random과 round robin은 시스템 상태와 독립적으로 결정을 만든다. Adaptive 그룹에 속하는 정책들 즉, shortest queue와 shortest response time queue는 현재 시스템 상태에 대한 정보를 사용한다. 시뮬레이션 결과 shortest response time queue 정책이 다른 정책에 비해 성능이 우수함을 확인하였다.

Ⅲ. ELI 기반 동적 부하 균등화

3.1 예측 부하 지수의 계산

본 논문의 주된 내용은 각 노드의 히스토리에 저장된 주기적으로 교환된 리모트 노드들의 시스템 상태 정보에 보간 다항식을 적용하여 리모트 노드들의 상

태인 ELI를 계산하고, 그 ELI를 이용하여 프로세스 이주에 관한 의사 결정을 하는 동적 부하 균등화 시스템을 설계하고, 그것의 성능을 시뮬레이션을 통하여 분석·평가하는 것이다.

본 논문의 동적 부하 균등화 시스템의 히스토리는 현재 시점 부터 과거 다섯 시점 동안의 주기적으로 교환된 리모트 노드들의 시스템 상태 정보 즉, 부하 지수(L)를 저장하고 있다. 이러한 부하 지수들은 그 정보가 교환된 시점에서는 글로벌 시스템 상태 정보와 정확히 일치하지만 다음 시스템 상태 정보 교환 시점까지 시간이 흐를수록 틀려지게 된다. 따라서 본 논문에서는 다음 주기 시점의 리모트 노드의 예측 부하 지수, ELI(E)를 각 노드의 히스토리를 기초로 Newton의 후향 보간법[6]의 5차 보간 다항식을 사용하여 구한다. 그 다음 L와 E를 사용하여 현재 리모트 노드들의 예측 부하 지수, ELI(C)를 구할 수 있다.

$$C_i = \alpha \times E_i + (1-\alpha) \times L_i \quad (1)$$

C_i : 노드 i의 현재 ELI

L_i : 노드 i의 최근 시점에 교환된 부하 지수

E_i : 노드 i의 다음 주기의 ELI

$$\alpha = \frac{\text{현재 시간(C\_time)} - \text{최근 부하지수가 교환된 시간(P\_time)}}{\text{부하 지수가 교환되는 시간 간격(T\_int)}}$$

$0 \le \alpha \le 1$

식 (1)에서 알 수 있듯이 C 값은 부하 지수가 교환된 직후에는 최근 시점에 교환된 부하 지수 L 값에 근사하고, 부하 지수 교환 직전에는 예측된 부하 지수 E 값에 근사하다.

3.2 동적 부하 균등화 시스템 설계

본 논문에서 설계된 각 노드 N_i의 동적 부하 균등화 시스템은 (그림 3)과 같은 구조를 갖는다. 여기서 설계된 동적 부하 균등화 시스템은 글로벌·동적 부하 균등화 알고리즘이고 과부하인 노드가 시스템의 동적 상태 정보를 기초로 저부하인 노드를 탐색하는 적응적이고 소스 주도형 알고리즘이다. 제안된 동적 부하 균등화 시스템의 구조에서 scheduler와 SSE의 설계가 본 연구의 핵심이다.

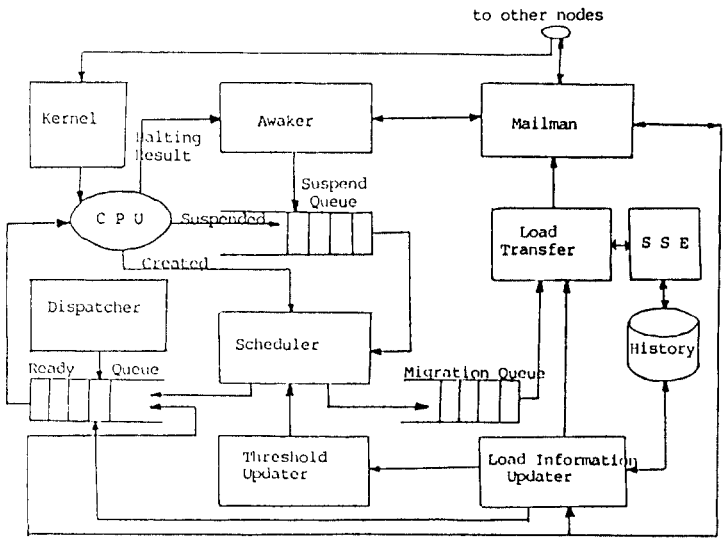
(1) Load Information Updater 모듈

정보 수집 정책으로 시스템 상태 정보 즉, 부하 지수를 계산하고 관리한다. 노드 N_i의 부하 지수 L_i는 다음과 같이 계산하였다.

$$L_i = \sum_{p_k \in R_i} F(p_k) \quad (2)$$

$F(p_k)$ = p_k의 CPU time

여기서 p_k는 프로세스 k, R_i는 노드 i의 준비 큐를 나



(그림 3) 설계된 동적 부하 균등화 시스템의 구조

타낸다. 그리고 주기적으로 교환되는 정보를 히스토리에 저장하고 필요한 정보를 threshold updater 모듈과 load transfer 모듈에 제공한다.

(2) Threshold Updater 모듈

Threshold updater 모듈은 프로세스의 로컬 혹은 리모트 처리를 판단하는 기준이 되는 임계값을 계산한다. 먼저 모든 로컬 부하 지수들을 주기적으로 모아서 시점 t에서의 시스템 평균 부하 지수 L_t를 구한다.

$$L_t = (\sum_{i=0}^{n-1} L_i)/n \qquad (3)$$

여기서 n은 분산 시스템내의 노드의 갯수를 나타낸다. 그리고 노드 i의 임계값, δ_i는 다음과 같이 계산되어 진다.

$$\delta_i = (1+\beta) * L_t \qquad (4)$$

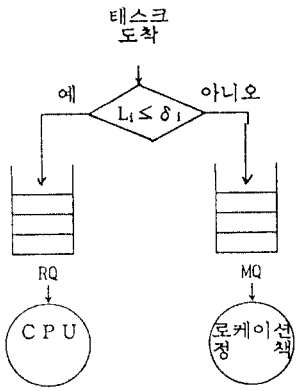
여기서 β값이 작을수록 임계값은 작아지고 과부하 노드가 많아져서 메세지 교통량이 많아질 뿐만 아니라 부하에 따라 노드의 상태 변화가 민감해 진다. 그리고 β값이 클수록 임계값이 커지고 과부하 노드가 작아져서 부하 균등화를 사용치 않는 것과 같다. 따라서 임계값을 시스템 평균 부하 지수보다 조금 크게 $0 \le \beta \le 0.2$로 정의하였다. 계산되어진 δ_i는 scheduler 모듈에 제공되어 진다.

(3) Scheduler 모듈

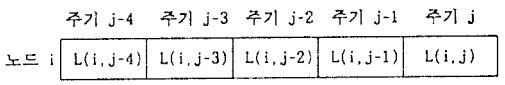
Scheduler 모듈의 전송 정책은 (그림 4)와 같이 구성되어 진다. Scheduler 모듈은 2개의 큐 : 준비 큐(ready queue : RQ)와 이주 큐(migration queue : MQ)를 가지고 있다. 준비큐는 국부 처리가 결정된 태스크들이 CPU를 할당받을 때 까지 기다리는 큐이고, 이주큐는 리모트 처리가 결정된 태스크들이 이주할 목적지 노드가 결정될 때 까지 기다리는 큐이다. 태스크가 노드 N_i에 도착 했을 때 scheduler는 L_i와 δ_i를 비교하여 L_i가 δ_i 보다 작거나 같으면 그 태스크는 준비 큐에 입력하고, 아니면 그 태스크는 이주 큐에 입력한다.

(4) SSE 모듈

SSE 모듈은 로케이션 정책인 load transfer를 지원하는 구성 요소로서 히스토리에 저장되어 있는 부하 지수를 기초로 Newton의 후향 보간법의 5차 보간 다항식을 적용하여 다음 주기 시점의 리모트 노드들의 예측 부하 지수인 ELI를 계산한다. 히스토리는 (그림 5)와 같이 현재 시점부터 과거 다섯 주기까지 교환된 리모트 노드들의 부하 지수를 저장하고 있다. 여기서 L(i, j)는 j 주기 시점에서 노드 i의 부하지수를 나타낸다.



(그림 4) 노드 i의 scheduler의 구조



(그림 5) 히스토리의 구조

(5) Load Transfer 모듈

Load transfer 모듈은 FIFO 방식으로 운영되는 이주 큐에 있는 프로세스가 이주할 목적 노드를 결정하는 로케이션 정책을 수행한다. Load information updater 모듈에 있는 가장 최근 주기 시점의 리모트 노드 부하 지수와 SSE 모듈에서 구해진 다음 주기 시점의 리모트 노드의 부하지수를 이용하여 현재 시점의 리모트 노드들의 예측 부하 지수를 식(1)로 구한다. Load transfer 모듈는 이주 큐가 비어있지 않으면 그 큐의 선두에 있는 하나의 프로세스를 최소 예측 부하지수를 갖는 노드에 전송한다.

Ⅳ. 시뮬레이션

4.1 시뮬레이션 환경

본 논문에서 제안하는 ELI 기반 동적 부하 균등화

시스템의 시뮬레이션 프로그램은 C 언어로 작성하였으며, IBM-PC 486/DX 시스템에서 실행하였다. 분산 시스템의 모델은 (M/M/1) 노드가 4개로 구성된 형태인 4×(M/M/1) 시스템을 대상으로 하였다. 그리고 시뮬레이션에 필요한 가정들은 다음과 같다.

(1) 분산 시스템은 loosely coupled 시스템이고, 각 프로세서는 homogeneous하다.

(2) 부하 균등화는 프로세스 단위로 수행된다.

(3) 모든 프로세스는 서로 독립적이며, 같은 순위를 갖는다.

(4) 프로세스간의 통신은 고려치 않았다.

(5) 준비 큐는 FIFO 방식으로 운영된다.

(6) 통신 서브시스템은 신뢰성이 있다.

시뮬레이션에 필요한 패러미터는 〈표 1〉과 같이 정의되어 졌다.

〈표 1〉 시뮬레이션 패러미터

패러미터	데이터 값		패러미터	데이터 값
노드의 갯수	4 개		평균 태스크 크기	5 Kbyte
각 노드의 평균 태스크 도착 시간	노드 1	17.0 초	평균 태스크 크기의 편차	2 Kbyte
	노드 2	17.0 초		
	노드 3	5.0 초	태스크 처리 결과의 크기	1 Kbyte
	노드 4	5.0 초	패킷의 크기	1 Kbyte
각 노드의 평균 태스크 서비스 시간	노드 1	6.0 초	제어 메세지의 크기	500 byte
	노드 2	6.0 초		
	노드 3	6.0 초	패킷의 전송 지연 시간	500 msec
	노드 4	6.0 초		
시스템 상태 정보 교환 주기	100 초		제어 메세지의 전송 지연 시간	250 msec

각 노드의 태스크 도착 시간과 서비스 시간은 실세계의 환경이 매우 다양하여 그 값을 결정하기가 매우 어렵다. 실제 구현된 시스템에서 동적 상황을 추적하여 얻어진 수치로 도착시간과 서비스 시간을 입력으로 하면 현실적이나 그러면 적용 시스템 환경이 한정되고 실행에 옮기는데 많은 시간과 노력을 필요로 한다. 본 논문에서의 시뮬레이션 모델이 M/M/1이고 이들의 도착시간과 서비스시간이 모두 지수분포에 따르므로 이들에 대한 평균값을 패러미터로 지수분포 확률 발생함수에 의한 값을 입력값으로 하였다. 여기서 평균 도착시간이 평균 서비스시간보다 크면, 그 노드는 잠재적으로 저부하 상태가 많을 것이며, 반대의 경우에는 과부하 상태가 많을 것이다. 따라서 본 논문에서는 시스템을 flow balance 상태로 유지하기 위하여 2개의 노드(노드 1, 2)는 저부하 상태로, 나머지 2개의 노드(노드 3, 4)는 과부하 상태로 하였다. 그리고 본 논문에서는 각 노드들이 동종의 시스템이라는 점을 감안하여 평균 서비스시간을 같게 하였다. 시스템에 제출되는 태스크의 크기를 어떤 수식으로 결정적으로 산출하거나 또는 어떤 값으로 고정한다는 것을 실환경에 어긋나게 된다. 그러나 다수의 태스크의 크기들은 일반적으로 정규분포 형태를 취할 것이다. 따라서 태스크의 크기는 평균 5 Kbyte, 표준편차 2 Kbyte인 정규분포로 하였다. 처리후 반환되는 결과의 크기는 대부분 제출된 태스크의 크기보다 작을 것이다. 이 또한 정규분포를 갖고 그 크기가 태스크에 비해 상대적으로 작을 것이므로 태스크의 평균 크기를 참조하여 1 Kbyte로 고정하였다. 그리고 평균 태스크 크기, 평균 태스크 크기의 편차, 태스크 처리 결과의 크기, 패킷의 크기, 그리고 제어 메세지의 크기는 메세지 전송 지연 시간을 계산하는데 필요하다.

시뮬레이션에서는 제안하는 ELI를 이용한 동적 부하 균등화 시스템의 성능을 다음 3가지 참조 정책[12]과 비교하였다.

(1) Focused Addressing 알고리즘

태스크가 발생한 노드의 부하지수가 임계값보다 큰 과부하 상태이면, 주기적으로 교환된 시스템 상태 정보를 기초로 최소 부하지수를 가지고 있다고 예측되는 노드에 그 태스크를 보낸다.

(2) Bidding 알고리즘

태스크가 발생한 노드가 과부하 상태이면, 리모트 노드들에 Request-For-Bid 메세지를 전송하고 그 응답으로 받은 리모트 노드들의 상태 정보를 기초로 이주할 노드를 선택한다.

(3) Flexible 알고리즘

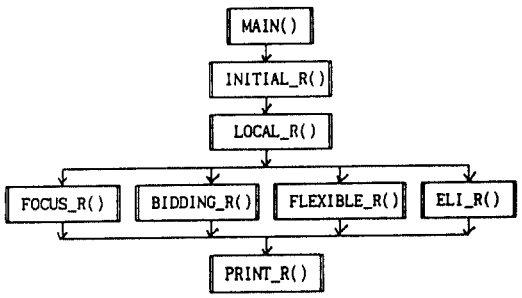
태스크가 발생한 노드가 과부하 상태이면, 먼저 focused addressing 방법으로 이주할 목적지 노드를 선정한다. 그리고 그 목적지 노드의 부하 지수가 $(1-\gamma)*L_t$ 보다 크다면 bidding 방법으로 새로운 목적지 노드를 찾는다. 여기서 γ는 $0 \le \gamma \le 0.2$이다. 즉, flexible 알고리즘은 focused addressing 알고리즘과 bidding 알고리즘을 혼합한 부하 균등화 정책이다.

그리고 시뮬레이션에서 필요한 각 노드의 초기 시스템 상태 정보와 임계값을 구하기 위하여 0~100초 동안에 태스크의 로컬 처리만 가능하고 글로벌 처리

가 불가능한 noncooperative 알고리즘을 실행한 후 focused addressing, bidding, flexible, 그리고 ELI를 이용한 동적 부하 균등화 알고리즘들을 각각 실행하였다.

4.2 시뮬레이션 프로그램

ELI를 이용한 동적 부하 균등화 시스템의 시뮬레이션 프로그램은 (그림 6)과 같이 8개의 루틴으로 구성되어 진다. 각 루틴의 이름과 기능은 다음과 같다.



(그림 6) 시뮬레이션 프로그램의 구성

(1) MAIN( ) : 시뮬레이션에 필요한 자료구조를 선언하고 시뮬레이션 패러미터와 시뮬레이션 시간을 입력하고 나머지 7개의 루틴을 제어한다.
(2) INITIAL_R( ) : 각 노드가 시뮬레이션 패러미터에 따라 주어진 시뮬레이션 시간까지 태스크를 발생시키는 루틴이다.
(3) LOCAL_R( ) : noncooperative 알고리즘을 실행하는 루틴이다.
(4) FOCUS_R( ) : focused addressing 알고리즘을 실행하는 루틴이다.
(5) BIDDING_R( ) : bidding 알고리즘을 실행하는 루틴이다.
(6) FLEXIBLE_R( ) : flexible 알고리즘을 실행하는 루틴이다.
(7) ELI_R( ) : ELI 기반 동적 부하 균등화 알고리즘을 실행하는 루틴이다.

4.3 성능 분석 및 평가

시뮬레이션을 통하여 나타난 ELI 기반 동적 부하 균등화 시스템의 성능을 분석하고 평가한다. 여기서는 아래와 같은 항목에 대하여 4 가지 알고리즘 : focused addressing(FOCUS), bidding(BID), flexible (FLEX), 그리고 ELI 기반 동적 부하 균등화(ELI)를 분석하고 평가한다.

(1) 제어 메세지 통신량

(2) 평균 태스크 응답 시간

$$\text{평균 태스크 응답시간} = \frac{\sum_{i=1}^{m} (CT_i - AT_i)}{m}$$

CT_i ; 태스크 i의 완료 시간
AT_i ; 태스크 i의 도착시간
m ; 완료된 태스크의 갯수

(3) 부하 표준 편차

$$\text{부하 표준 편차} = \sqrt{(\sum_{i=0}^{n-1} (L_i - L_t)^2)/(n-1)}$$

L_t ; 평균 부하 지수
L_i ; 노드 i의 부하 지수
n ; 노드의 갯수

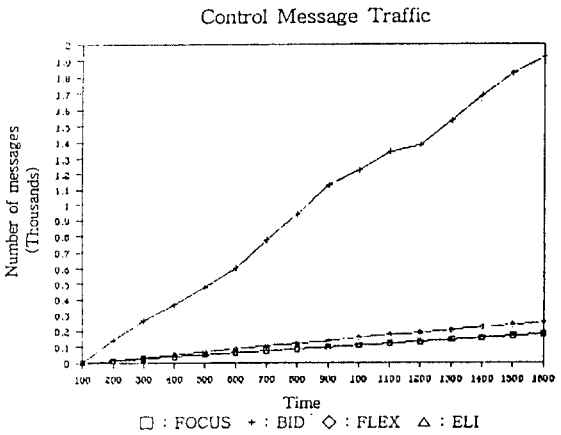
(4) 태스크의 국부 처리율

$$\text{태스크의 국부 처리율} = \frac{\text{국부 처리된 태스크의 갯수}}{\text{전체 완료된 태스크의 갯수}}$$

위의 각 성능 평가 항목별 시뮬레이션 결과를 살펴보면 다음과 같다.

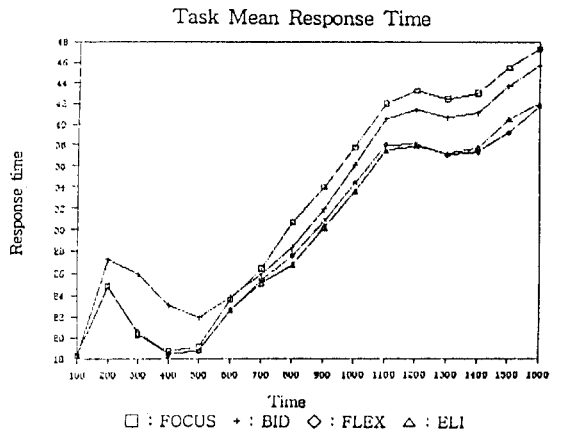
부하 균등화 시스템에서 분산 시스템의 각 노드들은 제어 메세지를 서로 송·수신하여 리모트 노드들의 시스템 상태를 알 수 있다. 따라서 제어 메세지 교통량이 많아 질수록 시스템 상태 정보의 질은 우수하지만 많은 제어 메세지 교통량으로 인한 통신 비용 때문에 오히려 태스크 응답 시간이 지연되는 상반관계가 존재한다. 태스크가 발생한 노드가 과부하 상태일 때 마다 제어 메세지(request-for-bid, bid)를 송·수신하는 BID가 제어 메세지 교통량이 가장 많다. 그리고 FLEX 역시 ELI와 FOCUS 보다는 제어 메세지 교통량이 많고, ELI와 FOCUS는 같은 제어 메세지 교통량을 갖는다(그림 7).

평균 태스크 응답 시간의 성능은 가장 적은 제어 메세지 교통량을 갖는 FOCUS가 가장 나쁜데 비해, 같은 양의 제어 메세지 교통량을 갖는 ELI가 가장 좋음을 통하여 ELI 기반 동적 부하 균등화 알고리즘의



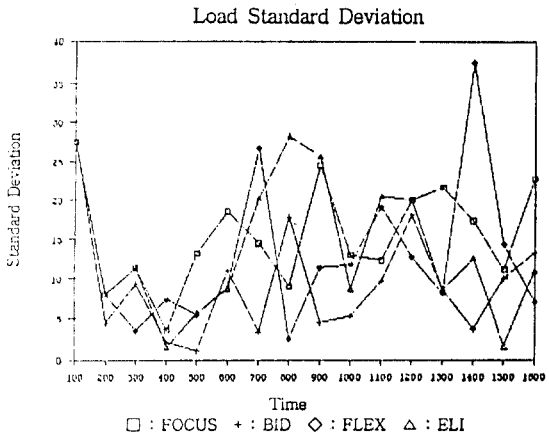
(그림 7) 부하 균등화 알고리즘별 제어 메세지 교통량

우수성을 쉽게 확인할 수 있다. 그리고 FLEX는 ELI에 비해 성능이 조금 떨어지지만 역시 우수하고, BID는 많은 제어 메세지 교통량 때문에 평균 태스크 응답시간에 지연이 발생하여 ELI와 FLEX에 비해 성능이 떨어짐을 알 수 있다(그림 8).



(그림 8) 부하 균등화 알고리즘별 평균 태스크 응답 시간

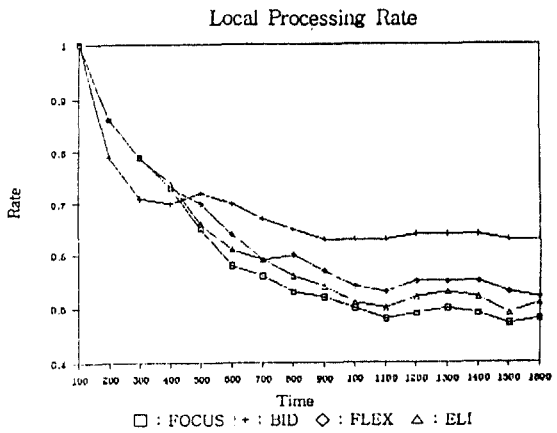
시스템 부하 표준 편차는 분산 시스템의 각 노드에 부하가 균등하게 유지되고 있는가를 살펴보기 위하여 각 노드간의 부하 차이의 정도를 나타낸다. 이 값이 클수록 부하의 균형정도가 덜된 상태이고, 작을수록 균형정도가 잘된 상태임을 나타낸다. 따라서 BID는 태스크 이주가 필요할 때 마다 제어 메세지의 교환으로 평균 태스크 응답 시간은 지연되지만 정확한 시스템 상태 정보를 알고 있으므로 부하의 분배는 잘 이루어 진다. 그리고 나머지 3가지 알고리즘들은 근사한 차이로 FLEX, ELI, 그리고 FOCUS 순으로 부하 분배가 잘 이루어지고 있다(그림 9).



(그림 9) 부하 균등화 알고리즘별 시스템 부하 표준 편차

국부 처리율은 BID가 0.65 정도로서 다른 알고리즘들에 비해 높다. 이것을 (그림 9)와 관련시켜 보면, BID는 태스크 이주가 필요할 때 마다 제어 메세지의 교환으로 정확한 시스템 상태 정보를 알고 정확한 태스크 스케쥴링을 수행한다. 따라서 국부 처리율이 0.65 정도일 때 부하 분배가 잘 이루어 진다는 것을 알 수 있다. 따라서 제어 메세지 교통량이 적고 국부 처리율이 0.65 정도일 때 가장 빠른 평균 태스크 응답 시간을 얻을 수 있다는 결론이 나온다. BID와 FLEX는 ELI에 비해 국부 처리율은 0.65에 더 가깝지만 오히려 많은 제어 메세지 교통량 때문에 평균 태스크 응답 시간이 늦고, FOCUS는 ELI에 비해 국부 처리율이 낮기 때문에 부하의 불균등이 발생하여 평균 태스크 응답 시간이 늦다(그림 10).

결론적으로 본 논문의 시뮬레이션 모델에서는 부하의 불균등이 제어 메세지 교통량보다 평균 태스크 응답 시간의 지연에 더 많은 영향을 주고, ELI는 BID와 FOCUS에 비해 적은 메세지 교통량 때문에 평균 태스크 응답 시간이 우수하고, 그리고 FOCUS에 비해 잘 된 부하 균등 때문에 평균 태스크 응답 시간이 우수함을 알 수 있다. 따라서 본 논문에서 제시하는



(그림 10) 부하 균등화 알고리즘별 국부 처리율

ELI 기반 동적 부하 균등화 알고리즘은 기존의 다른 부하 균등화 알고리즘의 문제점인 리모트 노드의시스템 상태 정보의 질과 제어 메세지 교통량간의 상반관계를 완화할 수 있다. 그리고 시뮬레이션 패러미터 변화에 따른 다른 성능 분석 결과를 요약하면 다음과 같다. 첫째, 각 노드의 평균 태스크 도착 시간과 서비스 시간을 변화시켜 시뮬레이션한 결과 각 노드의 부하가 아주 과부하, 과부하, 저부하, 그리고 아주 저부하로 차등화된 경우보다는 그렇지 않는 경우 즉, 과부하인 노드중에서 어느 노드가 더 과부하인지 알 수 없는 상태와 저부하인 노드중에서 어느 노드가 더 저부하인지 알 수 없는 상태에서 ELI가 다른 알고리즘에 비해 평균 태스크 응답 시간이 우수하였다. 둘째, 시스템 상태 정보의 교환 주기가 길어질 수록 제어 메세지 교통량은 적어지나 평균 태스크 응답 시간은 나빠지고 짧아질 수록 평균 태스크 응답 시간이 좋아지나 아주 짧아지면 많은 제어 메세지 교통량으로 ELI의 성능은 BID와 비슷해진다. 셋째, 패킷과 제어 메세지 전송 지연 시간이 짧을 수록 BID와 FLEX의 성능이 좋아지고 길어질 수록 ELI와 FLEX의 성능이 좋아진다.

V. 결 론

본 논문에서 분산 시스템의 각 노드들은 노드들간에 주기적으로 교환되는 시스템 상태 정보를 히스토리에 저장하고, 그 히스토리 정보에 Newton의 후향 보간법의 5차 보간 다항식을 사용하여 리모트 처리기들의 다음 주기의 예측 부하 지수, ELI를 계산하였다. 계산되어진 ELI를 동적 부하 균등화 시스템의 로케이션 정책이 이용하였다. 그 결과 ELI 기반 동적 부하 균등화 시스템은 기존의 부하 균등화 알고리즘에 비해 성능이 우수함을 시뮬레이션을 통하여 알 수 있었다. 따라서 적은 통신 비용으로 리모트 노드의 현재 부하 지수와 근사한 부하 지수를 알 수 있으므로, 기존의 다른 부하 균등화 정책의 문제점인 리모트 노드의 시스템 상태 정보의 질과 제어 메시지 교통량간의 상반관계를 완화할 수 있었다. 따라서 본 논문에서는 효율적인 로케이션 정책 구성 요소를 갖는 동적 부하 균등화 시스템을 설계할 수 있었다.

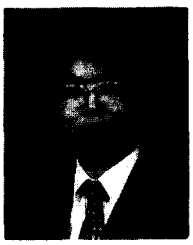
앞으로 본 논문에서 제안하는 ELI 기반 동적 부하 균등화 시스템을 더 큰 컴퓨터 시스템에서 시뮬레이션을 통한 성능 분석과 평가가 요구되며, 아울러 정확한 예측 부하지수를 구하는 방법에 대한 지속적인 연구가 요망된다.

참 고 문 헌

1. I. Ahmad, A. Ghafoor, and K. Mehrotra, "A Decentralized Task Scheduling Algorithm and its Performance Modeling for Computer Networks," Proc. of the Third IEEE Sym. on Parallel and Distributed Processing, pp. 314~321, Dec. 1991.
2. A. Barak and A. Shiloh, "A Distributed Load-Balancing Policy for a Multiprocessor," Software-Practice and Experience, 15(9), pp. 901~913, 1985.
3. T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," IEEE Trans. on Software Eng., SE-14(2), pp. 141~154, 1988.
4. L. M. Casey, "Decentralized Scheduling," The Australian Computer Journal, 13(2), pp. 58~63. 1981.
5. S. Chowdhury, "The Greedy Load Sharing Algorithm," Journal of Parallel and Distributed Computing, 9(1), pp. 93~99, May 1990.
6. S. D. Conte, C. D. Boor, Elementary Numerical Analysis, McGraw-Hill, Inc., 1980.
7. S. Dandamudi, "A Comparison of Task Scheduling Strategies for Multiprocessor Systems,"

Proc. of the third IEEE Sym. on Parallel and Distributed Processing, pp. 423~426, Dec. 1991.
8. K. Efe and B. Groselj, "Minimizing Control Overheads in Adaptive Load Sharing," The 9th Int. Conf. on Distributed Computing Systems, pp. 307~315, June 1989.
9. A. Goscinski, Distributed Operating System, The Logic Design, Addison-Wesley Pub., Co., 1991.
10. W. Korfhage, "Picking Processes For Migration," Parallel and Distributed Computing and Systems Proc. of the Fourth ISMM/IASTED Int. Conf., pp. 193~195, Oct., 1991.
11. V. M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," IEEE Trans. Computer, Vol. 37, No. 11, pp. 1384~1389, Nov. 1988.
12. K. Ramamrithm, J. Stankovic, W. Zhao, "Distributed Scheduling of Task Deadlines and Resource Requirements," IEEE Trans. on Computer, Vol. 38, No. 8, pp. 1110~1123, Aug., 1989.
13. H. G. Rotithor and S. S. Pyo, "Decentralized Decision Making in Adaptive Task Sharing," Proc. of the 10th Conf. on Distributed Computing Systems, pp. 34~41, 1990.
14. A. Sevensson, "History, an Intelligent Load Sharing Filter," Proc. of the 10th Conf. on Distributed Computing Systems, pp. 546~553, 1990.
15. J. A. Stankovic, "An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling," IEEE Trans. on Computers, C-34(2), pp.117~130, 1985.
16. Y. T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," IEEE Trans. on Computer, C-34(3), pp. 204~217, 1985.
17. J. Xu and K. Hwang, "Heuristic Methods for Dynamic Load Balancing in a Message-Passing Supercomputer," Proc. Supercomputing '90, pp. 888~897, 1990.

이 연구는 '92년도 한국과학재단 연구비 지원에 의한 결과임
과제번호 923-1100-009-1



裵 仁 漢(Ihn Han Bae) 正會員
1961年 2月 5日生
1984년 2월 : 경남대학교 전자계산학과(공학사)
1986년 2월 : 중앙대학교 전자계산학과(공학석사)
1990년 8월 : 중앙대학교 전자계산학과(공학박사)
1989년~현재 : 효성여자대학교 전자계산학과 조교수
※주관심분야 : 멀티미디어 시스템, 분산 처리 시스템, 병렬 처리 시스템