이질형 환경에서 네트워크 트래픽 감소를 위한 유전 알고리즘을 이용한 부하 균형 기법

A Load Sharing Scheme to Decrease Network Traffic Using Genetic Algorithm in Heterogeneous Environment

이성훈*, 조광문**

천안대학교 정보통신학부*, 목포대학교 경제통상학부 전자상거래학 전공**

Seong-Hoon Lee(shlee@cheonan.ac.kr)*, Kwang-Moon Cho(ckmoon@mokpo.ac.kr)**

요약

송신자 개시 부하 균형 알고리즘에서는 전체 시스템이 과부하일 때 송신자(과부하 프로세서)가 부하를 이전하기 위해 수신자(저부하 프로세서)를 발견할 때까지 불필요한 이전 요청 메시지를 계속 보내게 된다. 따라서 이 같은 상황에서는 저부하 상태인 수신자 프로세서로부터 승인 메시지를 받기까지 불필요한 프로세서 간 통신으로 인하여 프로세서의 이용률이 저하되고 또한 태스크의 처리율이 낮아지는 문제점이 발생한다. 본 논문에서는 이질형 분산 시스템에서의 동적 부하 균형을 위해 유전 알고리즘을 기반으로 하는 접근 방법을 제안한다. 이 기법에서는 불필요한 요청 메시지를 줄이기 위해 요청 메시지가 전송될 프로세서들이 제안된 유전 알고리즘에 의해 결정된다.

■ 중심어 : | 이질형시스템 | 부하 균형 알고리즘 | 유전 알고리즘 |

Abstract

In a sender-initiated load sharing algorithms, sender(overloaded processor) continues to send unnecessary request messages for load transfer until receiver(underloaded processor) is found while the system load is heavy. Therefore, it yields many problems such as low CPU utilization and system throughput because of inefficient inter-processor communications until the sender receives an accept message from the receiver in this environment. This paper presents an approach based on genetic algorithm(GA) for dynamic load sharing in heterogeneous distributed systems. In this scheme the processors to which the requests are sent off is determined by the proposed GA to decrease unnecessary request messages.

■ Keyword : | Heterogeneous system | Load Sharing Algorithm | Genetic Algorithm |

Ⅰ. 서 론

분산 시스템에서 부하 균형의 목표는 프로세서의 이용률을 극대화하고 평균 반응 시간을 최소화하기 위하여 각 프로세서에 태스크를 균형 있게 할당하는 것이다. 이러한 부하 균형 알고리즘(load sharing algorithm)

접수번호 : #050215-001

접수일자 : 2005년 02월 15일

심사완료일 : 2005년 04월 25일

교신저자 : 이성훈, e-mail : shlee@mokpo.ac.kr

은 크게 3가지(정적, 동적, 적응적 방법)로 분류할 수 있다. 본 논문에서는 이러한 방법들 중 동적 부하 균형 알고리즘을 기반으로 한다. 동적(dynamic) 부하 균형 알고리즘은 실행 중에 과부하(overloaded) 프로세서가 시스템의 부하 정보를 이용하여 초과된 태스크를 저부하(under-loaded) 프로세서로 보낸다. 기존에 연구되어 온 많은 부하 균형 알고리즘들은 대부분 동형 분산 시스템(homo-geneous distributed systems) 환경에서 부하를 각 프로세서에 균형 있게 유지하기 위한 연구들이었다[1, 2, 6, 8]. 이러한 기존의 알고리즘들은 각 프로세서들이 서로 다른 처리 속도를 갖는 이질형 분산 시스템에서는 적합하지 않다. 최근에 Chin Lu는 이질형 분산 시스템에서의 적응적 부하 균형 문제를 다루었다[11]. 본 논문에서는 이같은 이질형 분산 시스템에서의 동적 부하 균형을 위해 유전 알고리즘(genetic algorithm)을 기반으로 하는 접근법을 제안한다.

동적 부하 균형 기법은 송신자 개시 방법(sender-initiated), 수신자 개시 방법(receiver-initiated), 대칭 개시 방법(symmetrically-initiated)의 3가지로 세분화 할 수 있다. 본 논문에서는 이들 중 송신자 개시 방법을 기반으로 한다. 송신자 개시 방법은 송신자 프로세서(sender: overloaded processor)가 수신자 프로세서(receiver: underloaded processor)에게 태스크를 전송하기 위해 부하 균형 알고리즘을 수행한다[1, 2]. 이러한 송신자 개시 방법에서는 태스크를 전송하기 위한 이전 요청(request) 메시지가 송신자 프로세서로부터 임의로 선정된 다른 프로세서로 보내진다. 만일 선정된 프로세서가 저부하 상태이면 송신자 프로세서에게 승인(accept) 메시지를 보낸다. 반면에 선정된 프로세서가 과부하 상태이면 송신자 프로세서에게 거절(reject) 메시지를 보내게 된다. 송신자 프로세서는 수신자 프로세서로부터 승인 메시지를 받을 때까지 반복적으로 임의로 선정되는 다른 프로세서를 조사한다.

분산 시스템 내의 전체적인 시스템 부하가 과부하 상태이면 태스크를 이전 받을 수 있는 프로세서를 쉽게 찾을 수 없다. 왜냐하면 대부분의 프로세서들이 송신자 프로세서로부터 태스크를 이전 받을 수 있는 저부하 상태를 유지하지 못하기 때문이다. 따라서 이러한 과정을 승인 메시지를 받을 때까지 반복적으로 임의로 선정된 다른 프로세서들에 태스크 이전 요청 메시지를 보내야 한다. 그러므로 많은 이전 요청 및 거절 메시지들이 발생하고 이로 인하여 실행 전에 많은 시간이 소모된다. 따라서 실질적인 태스크 처리 시간이 낭비된다.

본 논문에서는 이질형 분산 시스템에서 효율적인 부하 균형을 위하여 송신자에 의해 발생되는 이전 요청 메시지들이 전송될 프로세서를 유전 알고리즘을 통하여 결정한다. 이 같은 유전 알고리즘을 사용하는 목적은 수신자 프로세서를 결정하기 위한 시간을 단축하여 CPU의 이용률을 증대하고 분산 시스템의 전체적인 처리율(throughput)을 증대하기 위한 것이다.

2장에서는 유전 알고리즘을 기반으로 하는 부하 균형 기법 등을 알아본다. 3장에서는 제안한 유전 알고리즘을 기반으로 하는 동적 부하 균형 알고리즘에 대한 내용 즉, 코딩(coding) 방법 및 적합도 함수, 알고리즘 등을 기술한다. 4장에서는 기존 방법과의 비교 분석을 위한 시뮬레이션 결과를 기술하고 분석한다. 마지막으로 5장에서 결론을 기술한다.

II. 관련 연구

본 논문에서는 이질형 분산 시스템에서의 부하 균형 문제를 해결하기 위하여 유전 알고리즘을 이용한다. 이러한 유전 알고리즘을 기반으로 하여 부하 균형 문제를 해결하기 위한 연구로서 Terence C. Forgarty 등은 사탕무우 공장(sugar beet pressing station)에서 프레스(press)들 간의 부하 균형 문제를 위하여 2단계의 유전 알고리즘을 이용하였다[9]. Garrison W. Greenwood 등은 실시간 분산 컴퓨팅 시스템(real-time distributed computing system)에서 휴리스틱 알고리즘의 적응성을 보이기 위해 진화 전략(evolutionary strategy)을 이용한 스케줄링(scheduling) 문제를 연구하였다[10]. 기본적으로 이 논문은 동형 분산 시스템을 기반으로 하였으며 진화 전략을 이용한 스케줄링 문제가 실시간 분산 시스템에 적용될 수 있음을 나타낸 연구라 할 수 있다. David B. Fogel 등은 이질형 컴퓨팅 환경에서 태

스크들을 스케줄링하기 위하여 진화 프로그래밍(evolutionary programming) 및 greedy 알고리즘을 이용하였다[12]. 이 논문에서는 이질형 컴퓨팅 환경에서의 스케줄링 문제에 진화 프로그래밍 및 greedy 알고리즘을 이용한 접근법이 적용될 수 있음을 보여 주고 있다.

III. 부하 균형

1. 부하 척도(load measure)

분산 시스템 내 각 프로세서의 부하 상태를 측정하기 위한 척도로 많은 방법들 중에서 CPU 큐 길이를 선정하여 사용하였다. 이 같은 이유는 기존의 연구에서 가장 적합한 것으로 알려져 있기 때문이다[5]. 하지만 이질형 분산 시스템의 가장 큰 특징은 동일한 태스크에 대하여 각기 다른 프로세서들이 다른 처리 시간을 갖는다는 것이다. 따라서 기존의 부하 척도 방법인 CPU 큐 길이는 이질형 분산 시스템의 부하 상태를 적절하게 반영하지 못한다고 할 수 있다. 이러한 배경 하에 이질형 분산시스템의 부하 상태를 적절히 나타낼 수 있는 VCQL(Virtual CPU Queue Length)이라는 부하 척도 방법을 사용한다.

$$VCQL = T_{no} \times P_{tu} \quad (1)$$

매개변수 T_{no}는 CPU 큐에 있는 태스크의 개수를 의미하고 P_{tu}는 각 프로세서에서 하나의 태스크를 처리하는데 소요되는 처리 시간을 나타낸다. 이질형 분산 시스템에서의 각 프로세서는 동일한 태스크를 처리할 때 상대적으로 다른 프로세서들 보다 처리 시간이 빠른 프로세서가 있는 반면에 느린 처리 시간을 갖는 프로세서가 있다. 따라서 VCQL을 각 프로세서의 부하를 측정하는 척도로 이용하는 것이 이질형 분산 시스템의 부하 상태를 더 적절히 나타낸다고 할 수 있다.

태스크를 다른 프로세서로 이전하기 위한 이전 정책(transfer policy)은 VCQL을 기준으로 하여 이전을 결정하는 임계값 정책(threshold policy)을 기반으로 한다. 이러한 이전 정책은 태스크가 이질형 분산 시스템 내 어느 특정 프로세서에 들어올 때 발생한다. 만일 한 프로세서에 들어온 새로운 태스크가 이 프로세서의 VCQL 값을 상한 임계값(T_{up})을 초과하도록 한다면 프로세서는 송신자가 된다. 반면에 해당 프로세서의 VCQL 값이 하한 임계값(T_{low})을 초과하지 않으면 태스크를 받을 수 있는 수신자 프로세서가 된다.

2. 개요

본 논문의 알고리즘에서는 전체 분산 시스템의 부하 균형을 위해서 모든 프로세서의 부하 상태를 3단계(저부하, 정상 부하, 과부하)로 분리하여 사용하며 이들은 각 프로세서에서의 VCQL로 결정된다. 이같은 3단계의 부하 표현 방법은 [표 1]과 같으며 이를 위해 2개의 임계값 즉, 하한(T_{low}) 및 상한 임계값(T_{up})을 사용한다.

표 1. 3단계 부하 표현

부하 상태	의미	기준(Criteria)
Light	저부하 상태	$VCQL \leq T_{low}$
Middle	정상 부하 상태	$T_{low} < VCQL \leq T_{up}$
Heavy	과부하 상태	$VCQL > T_{up}$

한 프로세서가 과부하 상태일 때 다른 프로세서로 태스크를 전송하기 위하여 부하 균형 알고리즘을 수행한다. 알고리즘은 태스크를 전송하는데 있어 요구 지향(demand-driven) 기법을 사용한다.

본 논문은 송신자 개시 정책을 기반으로 하기 때문에 부하가 증가하는 시점인 태스크가 어느 한 프로세서에 들어오는 시점에서 부하 균형 알고리즘을 수행한다. 먼저 해당 프로세서는 부하 정보를 요청하는 메시지들을 각 프로세서에 보낸다. 이 때 메시지가 전송될 프로세서가 임의로 선정되는 것이 아니라 유전자 연산 후 스트링의 내용에 따라 메시지가 전송될 프로세서들이 결정된다. 이 같은 유전 알고리즘을 이용하여 부하 균형 과정을 간략한 예를 통하여 기술하면 다음과 같다.

예를 들어 10개의 프로세서들로 구성된 분산 시스템에서 프로세서 P_0가 송신자 프로세서로 결정된 후의 수행 과정은 다음과 같다. 먼저 유전자 연산을 수행한 후

집단 내 스트링들 중에서 가장 높은 적합도를 갖는 스트링 <-,1,0,1,0,0,1,0,0,0>이 선정되었다고 가정한다. 그러면 송신자 프로세서 P_0는 이전 요청 메시지를 수신자 대상 프로세서 즉, P_1, P_3, P_6로 보낸다. 이전 요청 메시지를 받은 이들 3개의 프로세서 각각은 자신들의 VCQL를 기반으로 부하 상태를 측정하게 된다. 만일 이들 중에서 프로세서 P_3가 저부하 상태라면 프로세서 P_3로부터 송신자 프로세서 P_0로 승인 메시지가 보내진다. 그러면 프로세서 P_0는 프로세서 P_3로 태스크를 이전하게 된다. 수신자 대상 프로세서들로부터 두 개 이상의 승인 메시지가 보내지면 임의로 하나를 선정한다.

3. 코딩 방법

분산 시스템 내에 있는 전체 프로세서의 개수가 n일 때 i번째 프로세서는 P_i로 표현되며 i는 $0 \le i \le n-1$의 범위를 갖는다. 이러한 분산 시스템 내 각 프로세서는 자신의 집단을 가지며 각 집단은 여러 개의 스트링들로 구성된다. 이러한 스트링을 표현하는 코딩 방법으로는 이진 코딩(binary coding), 트리 코딩(tree coding), 프로그램 코딩(program coding) 방법 등이 있다[7]. 본 논문에서는 이진 코딩 방법을 이용한다. 따라서 하나의 스트링은 이진코드 벡터(binary-coded vector)로 정의할 수 있으며 $<v_0, v_1, v_2, \ldots, v_{n-1}>$과 같이 표현된다. 따라서 분산 시스템 내 프로세서가 10개 있을 때 이와 같은 코딩 결과는 [그림 1]과 같이 표현할 수 있다.

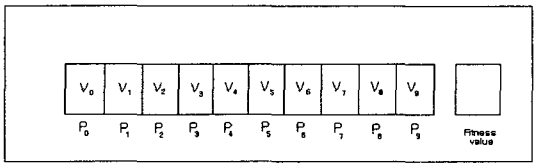

그림 1. 스트링에 대한 코딩

4. 적합도 함수

한 집단에 포함된 각 스트링은 식 (2)로 정의된 적합도 함수(fitness function)로 평가된다. 유전자 연산자가 적용된 하나의 집단 내 모든 스트링 중에서 가장 적합도가 높은 스트링이 선정되어 선정된 스트링의 내용 즉, 해당 스트링 내 '1'로 설정된 비트에 대응되는 프로세서로 태스크를 이전하기 위한 이전 요청 메시지를 보내게 된다.

$$F_i = \left(\frac{1}{\alpha \times TMPT + \beta \times TMTT + \gamma \times TTPT} \right) \quad (2)$$

식 (2)에서 사용된 α, β, γ는 각 매개변수 TMPT(Total Message Processing Time), TMTT (Total Message Transfer Time), TTPT(Total Task Processing Time)에 대한 가중치로서 4절의 성능 평가에서 다룬다. TMPT는 전송될 요청 메시지들의 처리 시간으로서 전송될 요청 메시지의 개수(request message number)와 각 메시지 처리 시간의 곱(product)으로 정의할 수 있다. 이 때 각각의 메시지 처리 시간은 태스크 처리 시간과 동일한 것으로 한다. 식 (3)에서 사용된 $ReMN_k$는 [그림 1]과 같이 표현된 스트링에 대한 코딩에서 k번째 비트가 '1'로 설정되어 있음을 의미하며 따라서 대응되는 프로세서로 요청 메시지를 보낸다.

$$TMPT = \sum_{k \in x} (ReMN_k \times \text{Time Unit}),$$
$$\text{단, } x = \{ i \mid v_i = 1 \text{ for } 0 \le i \le n-1 \} \quad (3)$$

매개변수 TMTT는 송신자로부터 선정된 스트링 내 '1'로 설정된 각 비트에 대응되는 수신자 대상 프로세서들까지의 메시지 전송 시간의 합을 의미한다. 이 매개변수는 식 (4)로 정의된다.

$$TMTT = \sum_{k \in x} (EMTT_k),$$
$$\text{단, } x = \{ i \mid v_i = 1 \text{ for } 0 \le i \le n-1 \} \quad (4)$$

식 (4)에서 사용된 매개변수 $EMTT_k$(Each Message Transfer Time)는 송신자 프로세서로부터 k번째 프로세서까지의 메시지 혹은 태스크 전송 시간을 나타낸다.

마지막으로 TTPT는 선정된 스트링 내 '1'로 설정된 각 비트에 대응되는 수신자 대상 프로세서들에서 태스크들을 처리하는데 소요되는 시간으로서 식 (5)와 같이 정의할 수 있다.

$$TTPT = \sum_{k \in x} (VCQL_k),$$
단, $x = \{ i \mid v_i = 1 \text{ for } 0 \le i \le n-1 \}$ (5)

5. 알고리즘

본 논문에서 제안하는 알고리즘은 5개의 프로시쥬어로 구성되는데 초기화(initialization), 부하 측정(load_check), 스트링 평가(string_evaluation), 유전자 연산(genetic_operation), 메시지 평가(message_evaluation) 등이다. 또한 유전자 연산은 3개의 부프로시쥬어(sub-procedure)인 지역 개선 연산(local_improvement_operation), 선택(selection), 교배(crossover) 등으로 구성된다. 이같은 프로시쥬어들은 분산 시스템 내 각 프로세서에서 시행되며 전체적인 알고리즘은 [그림 2]와 같다.

```
Genetic-based Load Sharing Algorithm
{ initialization();
  while (load_check())
  {
    string_evaluation();
    genetic_operation();
    message_evaluation();
  }
}

Procedure genetic_operation()
{
  local_improvement_operation();
  selection();
  crossover();
}
```

그림 2. 부하 균형 알고리즘

알고리즘을 구성하는 각 프로시쥬어들의 기능 및 알고리즘 단계는 다음과 같다.

● 초기화(initialization)

초기화 프로시쥬어는 전체 시스템이 작업을 시작할 때만이 각 프로세서에서 시행된다. 전체 시스템의 부하는 전반적으로 과부하 상태로 설정되고 스트링들로 구성된 집단은 어떠한 스트링의 중복 없이 임의로 만들어진다.

● 부하 측정(load_check)

어떤 하나의 프로세서에서 새로운 태스크가 들어올 때마다 부하 측정 프로시쥬어가 호출되어서 자신의 부하를 측정한다. 프로세서에서의 부하 측정은 자신의 VCQL을 측정함으로서 이루어진다. 만일 프로세서가 과부하 상태이면 스트링 평가 및 유전자 연산 프로시쥬어를 적용한 후 적합도에 비례하는 확률로 집단으로부터 하나의 스트링을 선정한 후에 선정된 스트링의 내용에 따라서 수신자 대상 프로세서들로 태스크 이전 요청 메시지를 보낸다.

● 스트링 평가(string_evaluation)

스트링 평가 프로시쥬어는 집단 내에 존재하는 스트링들의 적합도를 계산한다. 또한 태스크가 전에 적용되었던 프로세서에서 다시 들어올 때에는 바로 전에 생성된 집단 내 스트링들을 대상으로 학습을 하기 위하여 바로 이전(previous) 집단에 포함된 스트링들의 적합도를 계산한다.

● 유전자 연산(genetic_operation)

지역 개선 연산 및 선택, 교배 등을 포함하는 유전자 연산 프로시쥬어는 다음과 같은 방법으로 해당 집단에 적용된다. 먼저 분산 시스템 내 프로세서들이 지역적으로 그룹화되어 있는 환경에서는 하나의 스트링이 p개의 부분(part)으로 구성되어 있다고 가정할 수 있다. 그러면 다음과 같은 유전자 연산들이 각 스트링에 적용되며 새로운 스트링들로 구성된 집단이 만들어진다.

① 지역 개선 연산(local_improvement_operation)

먼저 스트링 1이 선정된다. 그런 다음 해당 스트링 1에 대한 하나의 복사본을 만들고 복사본의 부분 1로부터 p까지 순차적으로 선정된 후에 돌연변이(mutation) 연산자가 적용된다.

돌연 변이 연산자가 적용된 부분 1를 포함하고 있는 새로운 스트링에 대한 적합도를 계산한 다음 만일 새로운 스트링에 대한 적합도가 기존의 스트링 적합도보다 높으면 기존 스트링을 새로운 스트링으로 교체한다. 부분 2에 대한 연산이 끝나면 임의로 선정된 부분 2에 대한 지역 개선 연산이 수행된다. 이러한 지역 개선 연산

이 차례로 각 부분에 적용된다. 모든 부분에 대한 지역 개선 연산이 끝났을 때 하나의 새로운 스트링이 만들어진다.

스트링 1에 대한 지역 개선 연산이 끝난 후에 스트링 2가 선정되어 위에서 기술한 지역 개선 연산이 적용되며 이러한 유전자 연산이 집단에 포함된 모든 스트링에 적용된다.

② 선택(selection)

선택 연산은 스트링들에 적용되며 다음 세대의 집단을 만들기 위하여 부모가 될 스트링 쌍(pair)들을 결정하는 단계로 일반적으로 많이 이용되는 "roulette wheel selection" 기법을 사용한다.

③ 교배(crossover)

교배 연산은 새로이 만들어진 스트링들에 적용되며 교배 연산 결과 스트링들에 대한 평가가 이루어진다. 이 단계에서는 다음 세대 스트링들에 대한 부모들의 유전 형질 전이(genetic materials transfer)가 일어난다. 이러한 전이는 교배 연산자에 의해 이루어지는데 본 본문에서는 부하 균형을 위해 변형된 "one-point" 교배 연산자를 선정하여 적용한다. 기존의 순수한 "one-point" 교배 연산자는 교배를 수행할 교배 점(crossover point)을 하나의 스트링 내 전체 유전 인자들을 대상으로 하여 임의로 하나의 유전 인자를 선정하여 교배를 수행하였다. 본 논문에서의 변형된 교배 활동에서는 먼저 교배의 기본이 되는 교배 점을 선정하기 위해 전체 유전 인자를 대상으로 하지 않고 부분(p)들의 경계 점(boundary)들을 대상으로 하여 이 경계 점들 중에서 하나의 경계 점을 임의로 선정하여 교배 활동을 수행하게 된다.

● 메시지 평가(message_evaluation)

메시지 평가 프로시쥬어는 하나의 프로세서가 네트워크를 통하여 다른 프로세서로부터 메시지를 받을 때 발생한다. 프로세서 P_j가 Req_{ij} 메시지를 받을 때 프로세서 P_j가 저부하 상태이면 Acc_{ji} 메시지를 송신자 프로세서에 보낸다. 그런 다음에 태스크를 수신자 프로세서에 전송할 준비를 한다. 만일 프로세서 P_j가 과부하 상태이면 Req_{ji} 메시지를 보낸다. 프로세서 P_j로부터 받은 메시지가 Acc_{ji}일 때 j를 가용 리스트(available list)에 넣게 되며 이 리스트는 이전 요청 메시지를 받은 프로세서들 중에서 송신자 프로세서에 Acc_{ji} 메시지를 보내면서 여분의 태스크를 이전 받을 수 있는 프로세서들로 구성된다. 전에 발생된 P_i의 이전 요청 메시지에 대한 모든 응답을 받은 후에 송신자 프로세서는 수신자 프로세서에게 태스크와 함께 Mig_{ij} 메시지를 보낸다. 만일 가용 리스트에 두 개 이상의 프로세서가 존재하면 임의로 하나를 선정한다. Mig_{ij} 메시지를 받은 프로세서 P_j는 송신자 프로세서로부터 받은 태스크를 자신의 CPU 큐에 저장한다. 그런 다음에 송신자 프로세서 P_i는 수신자 프로세서 P_j에 T_{ij} 메시지를 보낸다.

Ⅳ. 성능 평가

제안된 유전 알고리즘을 기반으로 하는 동적 부하 균형 모델의 성능을 기존의 송신자 개시 알고리즘[1, 2] 및 단순 유전 알고리즘(simple genetic algorithm)과 비교하기 위하여 시뮬레이션하였다. 시뮬레이션에서 적용된 단순 유전 알고리즘 기반의 부하 균형 방법은 다음과 같은 내용에서 제안하는 방법과 차이가 있다. 단순 유전 알고리즘 기반의 방법에서는 먼저 지역 개선 연산을 수행하지 않고 선택 연산이 적용된다. 그런 다음에 교배 연산을 수행하게 되는데 이때 적용되는 교배 정책은 순수한 "one-point" 교배 정책을 이용한다. 마지막으로 돌연변이 연산을 수행하게 된다.

1. 시뮬레이션 환경

본 논문에서 시뮬레이션하는 시스템은 이질형 시스템으로서 30개의 이질형 프로세서로 구성되었다고 가정한다. 또한 처리될 각각의 태스크 크기는 동일한 것으로 가정한다. 각각의 프로세서로부터 다른 프로세서로의 메시지 전송 및 태스크 전송 시간은 2차원 배열로 표현되며 이때 각 원소의 값은 난수 발생기로부터 결정되는 3 이하의 값을 갖는다. 단위는 초(second)로 한다.

시뮬레이션을 위해 기본적으로 사용될 매개변수들은 [표 2]에 기술된 내용과 같으며 매개변수들 중에서 교배 발생 확률, 스트링 개수 등에 대한 값은 일반적으로 다양한 응용에 유용하게 적용되는 것으로 알려져 있다[3, 4].

표 2. 매개변수들의 내용

매개변수	값
프로세서의 개수	30
교배 발생 확률(Pc)	0.7
돌연변이 발생 확률(Pm)	0.1
스트링(염색체)의 개수	50
처리될 태스크 개수	5000
스트링의 부분 개수(p)	5
TMPT에 대한 가중치(α)	0.5
TMTT에 대한 가중치(β)	0.15
TTPT에 대한 가중치(γ)	0.05

2. 비교 평가

[시뮬레이션 1] 본 시뮬레이션은 분산 시스템의 전체 시스템 부하가 60%일 때 제안된 방법과 기존의 송신자 개시 방법 및 단순 유전 알고리즘을 이용한 접근법 간의 반응 시간을 알아보기 위한 시뮬레이션으로서 [표 2]를 기반으로 한 시뮬레이션 결과는 [그림 3]과 같다.

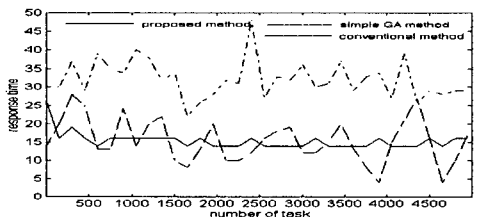

그림 3. 시뮬레이션 1에 대한 결과

시뮬레이션에서 나타난 성능은 각각의 처리 대상 태스크에 대한 반응 시간을 나타낸 결과로서 기존의 송신자 개시 방법에서는 대상 프로세서의 결정을 임의로 하기 때문에 각 태스크에 대한 반응 시간에서 심한 요동 현상을 보인다. 반면에 제안된 알고리즘에 대한 결과에서는 위에서 기술된 두 개의 방법에서 보이고 있는 심한 요동 현상을 보이지 않고 있으며 평균 반응 시간에서 가장 낮은 시간을 보이고 있다.

[시뮬레이션 2] 본 시뮬레이션은 시스템 부하에 따른 반응 시간 및 평균 반응 시간의 변화를 알아보기 위하여 전체 시스템 부하가 80%일 때의 반응 시간을 알아보기 위한 시뮬레이션이다.

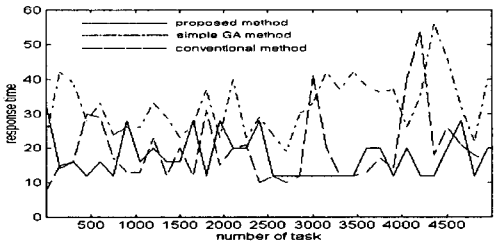

그림 4. 전체 시스템 부하 80%에서의 반응 시간

[그림 4]는 전체 시스템 부하가 80%일 때 반응 시간을 나타낸 것이다. 시뮬레이션 1에서의 결과보다 각 방법 간의 성능 차이가 두드러지게 나타나고 있으며 제안된 알고리즘에 의한 성능이 반응 시간 및 평균 반응 시간에 가장 우수한 결과를 보이고 있다.

V. 결 론

본 논문에서 제안하는 유전알고리즘 기반의 방법에서는 이전 요청 메시지들이 전송될 수신자 대상 프로세서들을 결정한다. 기존의 송신자 개시 부하 균형 알고리즘 및 단순 유전 알고리즘에 의한 접근법과의 비교 분석을 위한 시뮬레이션에서는 반응 시간을 기준으로 비교하여 분석하였다. 시뮬레이션 결과는 제안된 알고리즘이 시스템 부하가 높을 때 가능한 짧은 반응 시간을 갖는 수신자 프로세서를 결정함으로써 전체적으로 다른 방법들보다 빠르게 나타났다. 본 논문에서는 이질형 분산 시스템의 특성 중에서 각 프로세서의 처리 시간이 다르다는 점만을 고려하였다. 따라서 향후 연구에서는 이질형 특성 중 보다 많은 요소들을 반영하는 연구가 필요하다.

참 고 문 헌

[1] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," IEEE Transactions on Software Engineering, Vol.12, No.5, pp.662-675, May 1986.

[2] N.G. Shivaratri, P. Krueger and M. Singhal, "Load Distributing for Locally Distributed Systems," IEEE Computer, Vol.25, No.12, pp.33-44, December 1992.

[3] J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," IEEE Transactions on System, Man and Cybernetics, Vol.SMC-16, No.1, January 1986.

[4] J.R. Filho and P.C. Treleaven, "Genetic--Algorithm Programming Environments," IEEE Computer, pp.28-43, June 1994.

[5] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load sharing Scheme," IEEE Transactions on Software Engineering, Vol.17, No.7, pp.725-730, July 1991.

[6] L.M. Ni, C.W. Xu and T.B. Gendreau, "A Distributed Drafting Algorithm for Load sharing," IEEE Transactions on Software Engineering, Vol.SE-11, No.10, pp.1153-1161, October 1985.

[7] P. D. Wasserman, Advanced Methods in Neural Computing, Van Nostrand Reinhold, New York, 1993.

[8] M. Livny and M. Melman, "Load sharing in Homogeneous Broadcast Distributed Systems," Proc. ACM Computer Network Performance Symp, pp.44-55, 1982.

[9] T. C. Fogarty, F. Vavak and P. Cheng, "Use of the Genetic Algorithm for Load sharing of Sugar Beet Presses," Proc. Sixth International Conference on Genetic Algorithms, pp.617-624, 1995.

[10] G. W. Greenwood, C. Lang and S. Hurley, "Scheduling Tasks in Real-Time systems Using Evolutionary Strategies," Proc. Third Workshop on Parallel and Distributed Real-Time Systems, pp.195-196, 1995.

[11] C. Lu and S. M. Lau, "An Adaptive Load sharing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes," Proc. International Conference on Distributed Computing Systems, pp.629-636, 1996.

[12] D. B. Fogel and L. J. Fogel, "Using Evolutionary Programming to Schedule Tasks on a Suite of Heterogeneous Computers," Computers & Operations Research, Vol.23, No.6, pp.527-534, 1996.

저 자 소 개

이 성 훈(Seong-Hoon Lee) 정회원

- 1985년 2월 : 한남대학교 전산학과(공학사)
- 1995년 2월 : 고려대학교 컴퓨터학과(이학석사)
- 1998년 2월 : 고려대학교 컴퓨터학과(이학박사)
- 1998년 3월~현재 : 천안대학교 정보통신학부 교수

<관심분야> : 분산컴퓨팅, 그리드, 유전정보학

조 광 문(Kwang-Moon Cho) 정회원

- 1988년 2월 : 고려대학교 컴퓨터학과(이학사)
- 1991년 8월 : 고려대학교 컴퓨터학과(이학석사)
- 1995년 8월 : 고려대학교 컴퓨터학과(이학박사)
- 1995년 9월~2000년 2월 : 삼성전자 통신연구소 선임연구원
- 2000년 3월~2005년 2월 : 천안대학교 정보통신학부 조교수
- 2005년 3월~현재 : 목포대학교 경제통상학부 전자상거래학 전공 전임강사

<관심분야> : 전자상거래, 콘텐츠 유통, 모바일 콘텐츠, 데이터베이스, 그리드 컴퓨팅