자기학습을 이용한 결함허용 시스템의 부하조절 알고리즘 설계

장 순 주[†] · 구 용 완[††]

요 약

본 논문에서는 분산시스템 환경에서 n개의 노드가 결함일 경우, 결함을 허용해 주고, 시스템의 안정성을 유지하면서, 결함 노드의 부하를 정상 노드로 조실하기 위하여 부하 조절 알고리즘인 전송 정책, 위치 정책, 선정 정책을 제안하였다. 이러한 메카니즘은 부하 상태의 정보를 효과적으로 획득하고, 응답 시간을 줄이기 위하여 자기 학습 경험을 기반으로 하는 최적의 알고리즘을 선정할 수 있었다. 결과적으로 이를 기반으로 유사한 상황에서도 최적의 알고리즘을 선정할 수 있음을 알 수 있었다.

각 기법들의 효율성에 영향을 미칠 수 있는 매개변수를 적용하여 성능평가를 하였다. 성능평가 결과 작업 도착 율, 서비스 율, 노드의 결함 율은 서로간에 영향을 주지 못하고, 다만 결함 수리 율과 특히 부하의 이주에 대한 통신 지연 시간의 크기에 민감한 영향을 주었다

A Design of Load Conditioning Algorithm In Fault-Tolerant System using Self-learning

Soon-Ju Chang[†] · Yong-Wan Koo[††]

ABSTRACT

In this paper, we proposed for load control algorithm which is transfer policy, location policy and selection policy in case of n node's failure in a distributed system environment, allowing to failure, supporting system's safety, to translate failure node into no failure node. This mechanism gains load state's information efficiently and the best algorithm can select in the background of self-learning experience in order to reduce response time. And then we can select the best algorithm in a resemble appearance. We estimate system's affect to each algorithm which is applying to parameter. In a result, it is not affected to job arriving rate, service rate and failure of node rate, but we know that it is many affected to failure's repair rate and communication delay size with respect to load migration

1. 서 론

1.1 연구 목적

반도체 기술의 발전으로 컴퓨터 시스템은 일상생활 및 산업에 점차 중요한 도구로 자리잡게 되었다. 따라서, 이러한 시스템들의 고장은 많은 사용자들에게 심각한 문제로 대두되었기 때문에 신뢰성 있는 시스템설계를 위한 움직임이 학계와 산업계의 연구실을 중심으로 활발히 추진되어 왔다.

오늘날의 결함허용 시스템은 높은 신뢰도와 가용성(availability)이 요구되는 응용뿐만 아니라 일반정보처리 분야로까지 확대되어 가고 있다.

분산 시스템은 단일의 운영체제로 제어되는 처리기

† 정 회 원 수원대학교 지연피학연구소 연구원
†† 종신회원 수원대학교 전자계산학과 교수
논문접수 2000년 4월 27일, 심사완료 2000년 10월 23일

와 기억장치들이 지리적으로 분산되어 이들 상호간에 통신회선을 통해 데이터를 협력해서 처리하므로 하나의 자원 또는 노드에 결함이 발생한다 해도 전체 시스템에는 큰 손실을 입히지 않는 특성을 가지고 있으며, 또한 큰 관심 분야로 떠오른 신경망 모델은 대형의 중앙 집중식 보다 계산 능력이 낮고 상대적으로 저렴한 워크스테이션급의 처리기들이 모여서 서로의 연결망을 통해 정보를 주고받으면서 작업을 처리하는 병렬 분산 처리 방식이기 때문에 결함허용 시스템 설계를 위한 좋은 조건을 가지고 있다고 할 수 있다.

1.2 연구 내용

본 논문에서는 분산 실시간 시스템에서 n개의 노드에 결함이 발생했을 때 자기학습 결함진단 필터를 내장하여 새로운 작업을 맡아들여 프로세스의 결함을 진단하여 그 유형에 따라 결함 노드의 부하조절을 통하여 결함을 허용하기 위한 부하조절 기법을 설계, 평가한다. 부하조절 후에도 모든 노드들의 부하가 조절을 이루도록 정상 노드들의 결함전의 부하상태를 고려하였다. 각 기법들에 대한 평가를 위하여 기법의 성능에 영향을 미칠 수 있는 작업 도착율, 서비스율, 프로세서의 고장율과 수리율, 부하 이주에 따른 통신지연 등을 매개변수로 하여 성능평가를 하였다. 2장에서는 기존의 관련연구들을 기술하고, 3장에서는 자기학습 부하조절기 모델 및 결함허용을 위한 부하조절 기법 모델을 설계하고 4장에서는 각 정책에 대한 성능평가와 결과를 분석하여 부하조절 기법의 효율성을 평가함으로써 부하조절 기법의 선정지침을 제시하고 끝으로 5장에서는 본 논문의 결론을 기술하고, 향후의 연구방향을 제시하였다.

2. 관련 연구

2.1 결함허용 분산 시스템 분석

2.1.1 결함허용 시스템의 목적

결함허용 시스템이란 하드웨어 오동작, 소프트웨어 에러 또는 정보오염이 일어날지라도 주어진 임무를 올바르게 수행할 수 있는 시스템을 말한다. 최근 결함허용 시스템에 대한 중요성이 트랜잭션처리, 실시간 제어, 그리고 인간의 안전에 관련된 응용분야에서 급속히 증대하고 있다[Kim92]. 결함허용 컴퓨팅 시스템(Fault-Tolerance Computing System : FTCS)의 설계는 디지털 시스템들이 여러 종류의 결함이 일어날수 있다는 가정하에 수행하였는데, 이는 시스템에 중복(redundancy) 또는 여분(spare)의 장치를 추가함으로써 결함의 발생에도 불구하고 기능(functions)들의 올바른 수행을 제공하는 신뢰성 있는 시스템을 설계하고자 하는 것이다.

2.1.2 결함허용 시스템의 기본기능 및 기법

결함허용 컴퓨터 시스템은 중단 없는 실시간 서비스를 위해 시스템의 하드웨어나 소프트웨어의 결함 하에서도 시스템을 복구할 수 있는 능력을 갖춘 컴퓨터 시스템이다[Arms84][Huan80]. 결함허용 컴퓨터 시스템이 가지야 할 필수요건으로는 결함탐지(fault detection), 결함분리(fault isolation), 그리고 결함복구(fault recovery)기능이다[Bren88][Emme84]. 결함탐지 기능이란 하드웨어나 소프트웨어 메카니즘을 사용하여 결함을 탐지하는 기능을 말하고, 결함분리 기능은 발생된 결함이 시스템의 다른 부분에 영향을 미치지 않도록 시스템으로부터 분리시키는 것으로 결함봉쇄(fault containment)라고도 불린다. 결함복구는 결함으로부터 시스템을 복구하는 기능을 말하며, 이것은 다시, 탐지된 결함의 원인을 분석하는 결함진단(fault diagnosis)단계와 결함모듈을 시스템으로부터 제거하고 시스템을 재구성하는 재구성(reconfiguration)단계, 그리고 시스템을 결함발생 직전의 상태에서부터 다시 수행을 계속하게 하는 복구(recovery) 단계로 이루어진다. 영구결함인 경우에는 시스템으로부터 제거시킨다.

2.1.3 결함 유형

결함유형은 다음과 같은 다섯가지 관점에서 구분하여 볼 수 있다.

- 결함원인 : 결함 원인은 여러 가지가 있다. 즉, 설계상의 실수, 외부 환경으로부터의 영향, 구현시의 실수, 또는 부품결함 등을 들 수 있다.
- 결함속성 : 결함이 하드웨어에서 발생하였는지 소프트웨어에서 발생하였는지에 의해 구분될 수 있다.
- 결함지속시간 : 결함은 영구적일 수도 있고, 일시적일 수도 있다. 또한 주기적이든 비 주기적이든 반복적으로 나타날 수도 있다.
- 결함범위 : 결함이 미치는 영향이 국부적 일수도 있고, 시스템 전체 동작에 영향을 미칠 수 있다.
- 결함의 상태 : 발생된 결함 상태가 항상 일정할 수

도 있고, 시간에 따라 그 상태가 변할 수도 있다.

2.1.4 결함허용의 종류

결함허용 기법에는 하드웨어 결함허용기법, 소프트웨어 결함허용 기법 혼합 결함허용기법 등으로 시스템의 구성요소별로 구분할 수 있다

2.2 분산 시스템에서 부하조절 알고리즘 분석

2.2.1 부하조절 기법의 목적

부하조절의 목적은 프로세서 사이에 부하를 균형 있게 재 분산시켜 줌으로써, 어떤 프로세서는 처리할 수 있는 양 이상의 작업을 수행하고 다른 프로세서는 수행할 작업이 적어서 쉬는 시간이 많아지는 경우가 없도록 하여, 모든 프로세서의 수행능력을 최대화 시켜주며 그에 따라서 전체 시스템의 성능을 향상시키자는데 있다[Hsu86]. 부하조절에 대한 연구 방향은 크게 두가지로 나뉘어진다. 하나는 부하균형(load balancing)이며, 또 다른 하나는 부하공유(load sharing)로서 축소된 의미를 가지는 연구이다. 부하조절 정책에서는 가능한 한 모든 노드의 부하를 균형 있게 하기 위한 방법이며, 후자의 축소된 의미에서 부하공유 정책에서는 전체 시스템 내에서 유휴(idle)한 노드가 있음에도 불구하고 처리를 기다리는 작업이 존재하는 경우를 제거하는데 있다. 부하조절을 위해 부하가 많이 있는 어떤 프로세서의 작업을 부하가 적은 다른 프로세서로 옮기는데, 이런 동작을 작업이주(job migration)라 한다

2.2.2 부하조절기법의 기본기능

(1) 단계별 분류[Bara85]

① 정보정책(information policy)

프로세서의 부하를 측정하는 것은 아주 어렵고 다양하다 가장 많이 쓰이는 방법은 각 노드에 대해 주기적으로 준비상태에 있는 부하의 수를 계산하여 그것을 부하 값으로 취하는 것이며, 또다른 방법은 잔여 수행시간을 계산해서 그 노드가 끝날 때까지 필요한 프로세서 처리시간을 부하 값으로 삼는 것이다.

② 전송정책(transfer policy)

각 노드가 자신의 부하를 계산한 뒤, 다른 노드의 부하상태를 알아보거나 각 노드의 정보를 교환하는 통신방법이 필요하다. 현재 방송(Broadcasting)과 폴링(Polling)방법이 대표적으로 쓰인다

③ 배치정책(placement policy)

수집된 부하정보를 이용해서 어느 노드에서 어느 노드로 부하이주가 일어날것인가를 결정해야 한다. 부하이주를 결정하는 방법에는 절대적인 부하값을 이용하는 경우와 상대적인 부하값을 이용하는 경우가 있다. 전자는 대체적으로 부하분산을 위용하는 임계치를 미리 설정해놓고 그 이상인 경우는 부하의 이주를 허용하고, 그 이하일 경우는 지역 노드에서 그 부하를 처리한다.

④ 선정정책(selection policy)

전송할 작업을 선정하는 선정정책은 전송 정책인 한 노드를 송신자로 결정한 즉시 전송할 작업을 선정한다. 선정 정책이 전송에 적당한 작업을 찾는데 실패한다면, 전송 정책이 다시 송신자 노드로 결정될 때까지 그 노드는 송신자로 더 이상 고려되지 않는다 전송을 위해 작업을 선정하는 기본적인 기준은 작업을 전송하는데 초래되는 오버헤드가 그 작업에 의해 실현된 응답시간의 감소로 보상될 것이다 일반적으로 긴 실행시간이 걸리는 작업은 이러한 기준을 만족한다 뿐만 아니라, 작업의 유형에 따른 측정된 평균 실행 시간이 어떤 실행 시간 임계치 보다 클 때 원격실행을 위해 선정될 수 있다

(2) 정책별 분류

정책별 분류는 일반적으로 4가지로 대비할 수 있다[강88].

① 정적과 동적

정적은 시스템이 현재 상태보다 어떤 평균값에 의존하여 부하의 이주 경로가 시스템 설정시 이미 결정되어 있는 정책이다. 시스템내에 있는 각 노드의 부하가 정의될수 있을 때 이 정책이 유용하며 시스템 상태정보의 통신이나 부하이주 장소에 대한 계산이 필요 없는 대신에 동적인 변화에 대한 적응이 어려운 단점이 있다. 이 방법은 부하조절이 이루어지는 동안 어느 특정 프로세서 처리능력이 전체적으로 다른 것에 비해 월등할 경우 적용될 수 있다. 동적 부하조절 기법은 분산체제에서 사용되는 대표적인 알고리즘이며 이 정책에서는 부하이주 결정을 위해서 각 노드간의 정보를 수시로 또는 주기적으로 교환함으로써 최신의 부하상태 정보를 이용해 최적의 노드를 선택해서 부하를 이주시키는 정책이다. 이 기법은 어느 프로세서도 시간

의 변화에 따라 부하의 상태가 바뀜에 따라서 서버 혹은 클라이언트 프로세서가 될 수 있다. 그러므로, 모든 프로세서가 동일하거나 거의 비슷한 정도의 프로세스 처리능력을 갖는 것으로 하는 가정이 필요하게 되고 프로세서간의 처리능력에 있어서 최소한의 일반화를 필요로 한다.

② 소스주도와 서버주도

부하의 이주에 있어서 어느 노드가 주도하느냐에 따른 분류로써, 부하가 많은 노드가 다른 노드들 중에서 부하가 적은 노드를 찾아서 그 노드로 부하를 이주시키는 정책을 소스주도(sender-initiative)라 한다. 반면 부하가 적은 노드가 부하가 많은 노드를 찾아서 그 노드에게서 부하를 이주해 오는 정책을 서버주도(receiver-initiative)라 한다

③ 선네와 비선네

부하가 실행되고 있는중 이라도 그 부하가 이주될 수 있는 정책이 선네(preemptive)이고, 부하가 생성되는 순간에만 이주가 허용되는 정책이 비선매(non-preemptive)로 분류된다

④ 집중형과 분산형

집중형 정책은 한 노드만이 모든 노드의 부하 량에 대한 정보를 갖고 부하조절 알고리즘을 이용해 부하를 이주시킬 수 있다.

3. 부하조절 기법의 모델 설계

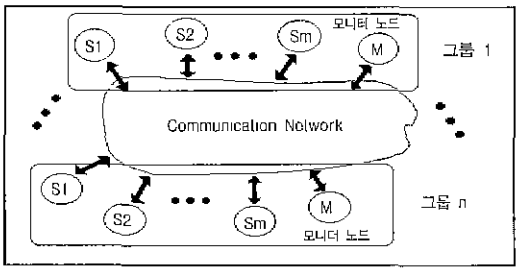
3.1 모델의 특성

기존의 부하균형(load balancing) 알고리즘에서 부하균등의 의미는 분산시스템에서 지리적으로 분산된 자원을 최대로 활용하기 위해 전체 시스템 내에서 처리되어야 할 작업부하를 각 프로세스에게 균등하게 배정함으로서 시스템의 성능을 향상시키고자 하는 목적이 있다[Hsu86]. 하지만 다양한 컴퓨터의 보급이 확산되고 통신기술이 발달함에 따라, 분산 시스템으로 연결된 각 노드들은 각각의 통신비용이나 처리능력 자체가 다른 다양하고 이질적인 컴퓨터들이 네트워크로 연결된 분산시스템으로 변화하여 왔다. 따라서 각 노드들의 부하를 균등하게 배정하는 기존의 부하균등화의 모델로서는 실제적인 시스템 성능 향상에 이바지 할 수 없었다.

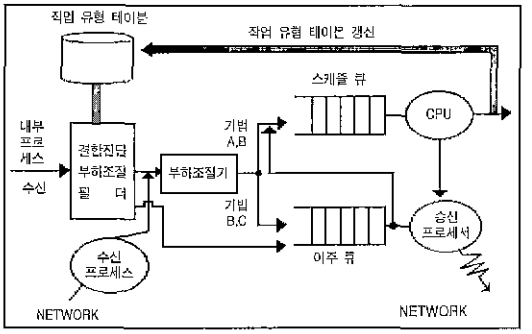
복잡하고 다양한 분산환경에서 통신지연이나 각 노드들의 다양한 시스템 성능을 고려하기에는 기존의 부하균형 알고리즘으로 구현이 현실적으로 불가능하다. 따라서 본 논문에서는 자신의 과거 실행한 경험으로부터 실행한 작업의 행위를 학습하고 다음 번 동일한 유형의 작업이 제출되었을 때 과거 실행한 경험으로부터 정확한 이주에 대한 적절성의 결정을 내릴 수 있는 시스템을 설계하기로 한다

3.2 자기학습 시스템 모델

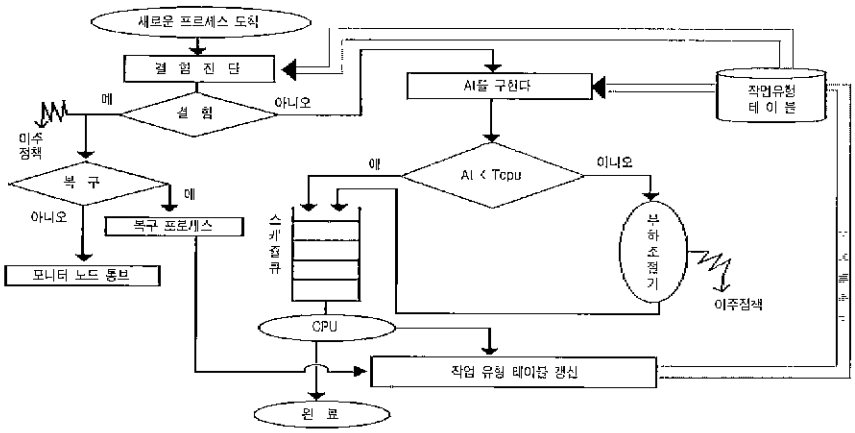
본 연구에서 제시한 자기 학습 부하 조절기 시스템 모델은 (그림 3-1)과 같은 구조를 가지며, 시스템의 각 노드는 처리 능력과 속도가 다른 환경을 갖는 시스템으로 가정하였다. 통신 네트워크에 연결된 각 노드는 여러 그룹으로 나누어지며, 각 그룹 $G_i(i=1,2,\cdots,n)$는 인접한 m개의 노드 $S_i(i=1,2,\cdots,m)$들과 한 개의 모니터 노드로 구성되며, 노드 $S_j(j=1,2,\cdots,m)$는 각각의 노드내에 내장되어있는 실험진단 부하조절 필터를 가지고 있으며, 필터를 위한 작업 유형 테이블을 가지고 있다. 또한 스케쥴 큐(SCHEDULE-Q)와 이주큐(TRANS-Q)를 가지고 있다. (그림 3-2)는 그룹 $G_i(i=1,2,\cdots,n)$



(그림 3-1) 자기 학습 부하 조절기 시스템 모델



(그림 3-2) 시스템 큐



(그림 3-3) 결함진단 부하조절 필터의 수행 과정

내의 노드 S_i와 모니터 노드의 시스템 튜닝 모델을 나타내고 있다.

1) 결함진단 부하조절 필터

본 논문에서 제시한 자기 학습 부하 조절기의 노드 Si에 내장되어 있는 결함진단 부하조절 필터는 새로운 작업을 모두 받아들여서 프로세스의 결함을 진단한 후 결함 발생시 작업유형 데이블을 참조하여 과거 같은 유형의 결함 발생시 수행한 경험을 바탕으로 다른 적당한 원격 노드로 프로세스를 이주시키도록 한다.

한편 결함의 유형에 따라 일시적 결함으로 자체 복구가 가능한 경우로 판단될 경우 복구 프로그램을 실행토록 유도하여 자체결함복구를 하도록 하고, 영구결함으로 판단될 경우 자신의 노드가 결함임을 모니터 노드에 알림으로써 새로운 작업이 들어오지 못하도록 한다.

결함이 탐지되지 않은 경우 오랜 수행 시간이 걸리는 작업들을 부하 조절기에 넘겨 부하 균등화를 이루도록 하며, 짧은 수행 시간이 소요되는 작업들은 지역적으로 처리하기 위해 스케줄 큐에 들어가도록 함으로써 적당한 원격노드를 찾기 위한 통신비용과 탐색 비용을 줄일 수 있다. 즉, 가능하면 긴 작업만이 원격으로 실행될 수 있게 함으로써 성능이 개선될 수 있기 때문이다.

2) 부하 조절기

자기 학습 부하 조절기 모델은 분산형과 집중형을 혼합한 하이브리드 알고리즘으로 설계하였다. 컴퓨터 통신망에 연결된 인접한 노드들을 묶어 하나의 그룹으로 만들고, 각 그룹은 m개의 노드와 하나의 모니터 노드로 구성되어 있으며, 알고리즘은 그룹간 알고리즘과 그룹내 알고리즘으로 나누어 설계하였다. 인접한 노드들로 묶어진 그룹내 알고리즘에서는 집중형으로 부하를 수집하며, 그룹간 알고리즘은 통신 지연 시간을 고려해 분산형으로 부하 정보를 수집한다. 노드의 수가 많은 시스템에서 집중형의 정보 수집 정책이 병목 현상(bottleneck)을 초래하고, 분산형의 정보 수집 정책이 통신 지연으로 인한 많은 오버헤드를 일으키는데 반해, 이러한 부하 조절기 모델은 시스템의 노드의 수에 상관없이 시스템이 안정적이다.

3.2.1 결함진단 부하조절 필터측 알고리즘

결함진단 부하조절 필터는 새로 제출된 작업을 수행하기 위해 자체 결함진단 프로세스를 수행하여 결함여부를 판단하여 결함 발생시는 제출된 작업과 결함 유형에 따라 적당한 원격 노드를 작업을 이주토록 하고, 결함의 유형에 따라 자신의 프로세스를 고립할 것인지, 자체적으로 수리할 것인지를 판단한다. 또한 결함이 발생하지 않은 경우 부하 조절기에 넘겨줄지의 여부를 판별하는 평가 기준을 사용한다.

평가 기준 =At < Tcpu (식 3-1)

여기서
At : 작업 유형 t에 대해 필요한 평균 CPU 실행 시간
Tcpu 필터 실행 시간 임계치

평가 기준(식 3-1)이 다음 조건을 만족하면 입력된 작업은 로컬에서 처리하기 위해 스케줄 큐에 입력되고,

그렇지 않으면 부하 조절을 수행하기 위해 부하 조절기에 삽입된다.

작업 유형은 같은 작업 유형을 갖는 모든 작업들로 구성된다. 이것은 option들이나 file들과는 달리 사용자나 파라메타 들이 무시된다는 것을 의미한다. 작업 유형 t에 대해 필요한 평균 CPU 실행 시간(At)을 구하기 위해 각 노드 Si에 중복된 테이블을 사용하는데 각각의 엔트리는 작업 유형 t에 대한 이름과 At로 구성된다.

결함진단 부하조절 필터의 수행 과정은 (그림 3-3)과 같다.

필터 실행 시간 임계치 설정 방법은 다음 (식 3-2)에 따른다.

$$\overline{X} = (\sum_{i=1}^{n} A_i)/n$$

$$\sum_{i=1}^{n}((X_i - \overline{X})^2 - (n-1))$$

$$T_{cpu} = \overline{X} - \varepsilon \times (S/\sqrt{n}) \qquad \text{(식 3-2)}$$

여기서

$\overline{X}$ 작업 유형에 따른 평균 CPU 실행 시간의 평균값
S^2 작업 유형에 따른 평균 CPU 실행 시간의 분산 추정치
T_{cpu} 필터 실행 시간 임계치
ε : (n-1)의 자유도를 갖는 t분포의 $1-\alpha/2$의 값

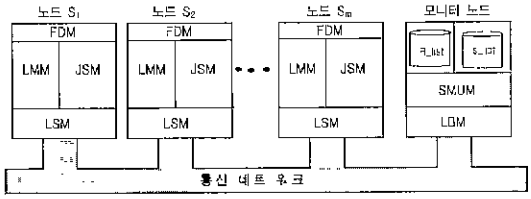
```
Algorithm Filter_routine()
{
  Wait(ProcessEnter);
  if(ProcessEnter){
      Fd(process id);
    if FR then
        FR(process id)
    else
        SEND Message()
    end
/*작업 유형 테이블에서 At와 Tcpu를 구한다 */
 JTT(Process_id),
 if(At<Tcpu)
   input_SCH_Q(Process_id),
 else
    LB(Process_id),
    }
}
```

(알고리즘 1) 결함진단 부하조절 필터측 알고리즘

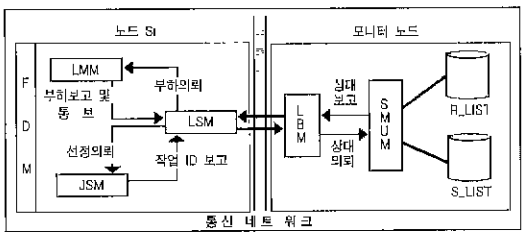
3 2.2 자기 학습 부하 조절기 구조

1) 노드 Si측 부하 조절기 구조

(그림 3-4)는 본 논문에서 제시한 자기 학습 부하 조절기 구조도를 나타내고 있다. 노드 Si(i = 1, 2, …, m)은 결함진단 모듈(Fault Diagnosis Module FDM)과 두 개의 병행 모듈(부하감시모듈(Load Monitor Module : LMM), 작업 선정 모듈(Job Selection Module JSM))과 부하 상태 관리자(Load State Manager : LSM)를 가지고 있으며, (그림 3-5)은 자기 학습 부하 조절기의 각 모듈 간 상호 관계를 보여주고 있다 본 절은 각 모듈의 상호 역할을 설명한다



(그림 3-4) 자기 학습 부하 조절기 구조도



(그림 3-5) 부하 조절기의 각 모듈간 상호 관계

(1) 결함 진단 모듈

결함 진단 모듈(FDM)은 새로운 작업이 입력되면, 결함진단 프로세스를 수행하여 결함을 탐지한다. 결함 발생시는 작업 유형 테이블을 탐색하여 결함 상태와 동일한 작업 유형이 있는지를 찾고, 과거 수행한 경험을 바탕으로 어느 노드로 작업을 이주시킬 것인가를 판단하여 해당 프로세스로 작업을 이주시킨다 또한 결함 상태에 따라 과거 수행한 경험을 바탕으로 결함을 자체 수리를 할 것인지 아니면 자신의 결함 상태를 모니터 노드를 송신하여 다른 작업이 들어오지 못하도록 하여 자신의 결함노드를 고립시킬 것인지를 결정한다. 자체수리를 할 것으로 판단되면 작업유형테이블에서 결함상태를 검색하여 적당한 결함복구 프로세스를 수행토록 하여 결함을 복구하고, 영구 결함으로 판단될 경우 모니터 노드로 자신의 결함상태를 통보하여 작업 스케줄러에게 자신의 노드로 작업을 보내지 말 것을 통보한다.

(2) 부하 감시 모듈

부하 감시 모듈(LMM)은 부하 조절 알고리즘의 구성 요소중 선송 정책에 해당된다 지금까지 제안된 것들중에 대부분이 전송 정책의 임계치 정책을 사용한다 [Eage86][Lin87]. 하지만, 본 논문에서는 각 노드가 처리 능력과 속도가 서로 다른 환경을 갖는 시스템으로 설계하고자 하므로 임계치 정책은 본 논문에 적합하지 않다 따라서 본 논문에서는 스케쥴 큐와 이주 큐에 제출되어 있는 작업의 과거 서비스 시간을 기초로 각 노드의 CPU 처리 능력에 따라 부하의 상태가 결정된다.

LMM은 주기적으로 노드의 부하상태를 파악하여 통신 서브시스템의 LSM에게 보고한다. 결함 진단/복구 모듈은 결함 여부를 탐지하고 복구를 담당한다. 만일 결함이 탐지되면 결함의 발생(또는 수리완료)사실을 부하 조절기와 통신 서브시스템의 LSM에게 통보한다. 그러면 LSM은 모니터노드에게 자신의 결함사실을 통보하고 타이머를 작동시킨다. 방송 메시지를 수신한 정상 노드들은 자신이 저부하 상태인 경우(노드 자신의 최대 부하용량으로부터 현재의 부하량과 비교결과) 수용가능 부하량과 노드 식별자를 메시지로서 반송한다. LSM는 타이머가 작동하는 동안에 정상 노드들로부터 수신한 응답 메시지를 메시지 큐에 넣은후 들어온 순서대로 수용가능 부하량 만큼씩 작업이주 큐를 통해 조절한다. 이때 조절의 원칙은 메시지 큐에 들어온 정상노드들 중 저부하 노드들로부터 받은 수용가능 부하량의 총합에 따라 적절한 부하 조절 기법이 설정된다. 이떤 노드에서 부하 발생시 그 노드만 제외시키고 정상으로 작동하는 노드들의 집합만으로 재구성하고 결함노드의 부하는 정상 노드들의 수용 가능 부하량에 따라 조절하여 수행하고 결함노드가 복구되면 정상노드의 집합에 포함시켜 재구성하게 되며 일단 조절된 부하는 원상 복구시키지 않는 것으로 한다. (그림 3-2)은 부하조절 시스템의 논리도이다 어떤 노드의 결함은 1개 이상의 지원노드에 의해 지원될 수 있다. 즉 노드가 결함일 때 대기 및 처리하고 있던 작업을 하나의 지원 노드에게 집중하면 시스템의 불안전 상태가 유발될 수 있으므로 여러개의 정상 노드에게 작업을 조절하기 위한 부하조절 알고리즘이 필요하다 본 장에서는 노드의 작업 부하를 정상노드에게 조절하기 위한 가능한 기법들을 설정하고 이들을 적용하는 부하조절 알고리즘을 설계하였다

```
Algorithm LMM( )
{
  Wait(LB(ProcessEnter OR ProcessExit)),
  if(ProcessEnter) {
     Reset(ProcessEnter),
     if(LocalLoad() > CPU 처리능력 × UB) {
          Status = HEAVYLOAD;
          SendMessage(LSM, n1),
       } else if(LocalLoad() > CPU 처리능력 × LB
            and LocalLoad() < CPU 처리능력× UB) {
          status = MODERATELOAD;
       sendMessage(LSM, n1),
        }
   } else if(ProcessExit) {
     Reset(ProcessExit);
     if(LocalLoad() < CPU 처리능력 × LB) {
          status = LIGHTLOAD;
          sendMessage(LSM, n1),
     } else if(LocalLoad() > CPU 처리능력 × LB
            and LocalLoad() > CPU 처리능력 × UB) {
          status = MODERATELOAD;
          SendMessage(LSM, n1),
     }
   }
}
```

(알고리즘 2) 부하 감시 모듈 알고리즘

(다) 작업 선정 모듈

작업 선정 모듈(Job Selection Module ; JSM)은 부하 상태 관리자가 프로세스 이주 메카니즘을 기동시키기 위해 작업의 선발을 요구하게 되면 이주 큐에 제출되어 있는 작업들중의 하나를 선발하여 선발된 작업의 ID를 부하 상태 관리자에게 통보해 주는 역할을 담당하는 모듈이다 부하 조절 알고리즘의 구성 요소중 선정 정책의 하나로써, 본 논문에서 제안한 작업 선정 기법은 다음과 같다.

여기서는 Weight Climbing이라는 기법이 사용되었다 초기에 JSM은 독단적으로 어느 작업을 전송할 것이냐를 설정한다 한 작업의 수행이 종료된 후 JSM은 수행시간 결과를 평가한다. 그래서 만약 작업의 수행시간길이가 전송하지 않고 로컬에서 실행했을 때 기대되는 실행 시간 보다 더 좋은 성능을 보였다면 JSM은 그 작업에 관한 자신의 지식을 갱신하고 다음번 그와 같은 작업을 또다시 만날 경우 전송하도록 한다. 하지만 그 수행 시간 길이가 로컬에서 처리했을 경우의 기대되는 수행 시간 보다 더 오래 걸렸다면 JSM은 다음번엔 그러한 작업을 로컬에서 처리하도록 지식을 갱신한다 이주 큐에 제출되어 있는 작업중 전송할 작업을 결정하기 위해 다음 (식 3-4)을 사용한다.

d가 커지면 커질수록 그만큼 작업이 전송될 기회는 커진다

$$d = \sum_{i=1}^{n} S_i W_i \qquad \text{(식 3-4)}$$

여기서, S_i : 센서(감지기)

W_i 가중치 벡터

본 알고리즘은 이질적인 환경으로써 각 센서들은 CPU의 큐 길이, 기계유형, 이용 가능한 장치들, 이용 가능한 메모리 크기 등과 상태를 감지한다. 한 작업의 전송이 종료되었을 때, 그 작업의 실행 시간은 그 유형의 작업에 대해 기대되는 실행 시간과 비교된다. 만약 전송된 작업의 실행 시간이 로컬에서 처리되었을 때의 기대되는 실행 시간보다 길었다면, 그 작업을 전송하는 것은 아마도 나쁜 결정이었을 것이고, 따라서 그 판단에 영향을 준 해당 가중치의 값을 감소시킨다. 그렇지 않고 판단이 옳았다면 그 가중치의 값을 증가시킴으로써 다음번 동일한 유형의 작업에 대해 전송할 확률을 높인다. 가중치의 조정 규칙은 같다. 초기에 JSM은 작업을 임의적으로 전송한다. JSM은 전송한 작업의 실행을 마쳤을 때 만약 그 작업이 지역적으로 실행해서 기대되는 실행 시간보다도 오래 걸렸다는 사실을 발견하면 언제나 그 대응되는 가중치의 값을 줄인다. 그렇지 않으면, 그 가중치는 증가된다. 가중치를 증가시키는 것은 한 작업을 전송할 기회를 증가시키고 가중치를 감소시키는 것은 작업을 전송할 기회를 줄이기 때문에, 결국 JSM은 대부분의 시간을 올바른 판단을 내릴 수 있도록 학습하는데 사용한다. 만약 메모리 크기 확장이나 프로세서 파워 증진 등과 같은 시스템 구성 요소가 변경되었다면 인간의 간섭 없이도 대응하는 가중치는 조정되고 JSM는 새로운 상황을 재학습한다.

(라) 부하 상태 관리자

부하 상태 관리자(LSM)는 노드 $S_i(i = 1, 2, \cdots, m)$측 부하 조절기의 최하위 계층에 위치하여 모니터 노드와 통신하면서 자신의 부하 상태가 과부하 또는 저부하 상태가 되었을 때, 자신의 상태를 모니터 노드에게 알린다. 이 때 모니터 노드는 수신 또는 송신할 프로세서의 주소를 부하 상태 관리자에 통보한다. 모니터 노드로부터 수신 또는 송신할 프로세서의 주소를 통보받은 부하 상태 관리자는 JSM에 프로세스의 선발을 의뢰하고 통보받은 프로세서의 주소와 작업 선정 모듈로부터 선정된 프로세스의 ID로서 프로세스 이주 메카니즘을 가동시켜 부하 조절을 이루도록 하는 역할을 담당하는 모듈이다.

2) 모니터 노드의 부하 조절기

(가) 상태 유지 및 갱신 모듈

상태 유지 및 갱신 모듈(State Maintenance & Update Module : SMUM)은 모니터 노드의 부하 조절 관리자로부터 노드 S_i 부하 상태를 전달받아 수신자 리스트 또는 송신자 리스트 데이블을 유지 및 관리하며, 부하 조절 관리자의 요구가 있을시 부하 조절 관리자에 시스템의 상태를 알려주는 역할을 담당한다

(나) 부하 조절 관리자

부하 조절 관리자(Load Balancing Manager : LBM)는 모니터 노드측의 부하 조절기 최하위 계층에 위치하여 노드 S_i에 송신 또는 수신할 프로세서의 주소를 통보해 프로세스 이주 메카니즘을 가동시켜 부하 조절을 이루도록 하는 본 시스템의 중추 신경의 역할을 담당하는 모듈이다

3.2.3 그룹내 알고리즘

본 논문에서 제시한 자기 학습 부하 조절기의 가동 시기는 임의의 시간에 노드 S_i의 LMM이 자신의 부하 상태가 과부하 또는 저부하로 판명되었을 경우, 자신의 부하 상태를 LSM에 통보함으로써 가동된다. 이 때 과부하 노드일 경우 송신자 노드가 되며, 저부하 노드일 경우 수신자 노드가 된다

1) Si노드측 알고리즘

임의의 시점에 노드 Si의 부하 감시 모듈(LMM)로부터 자신의 부하 상태 변화를 통보받은 부하 상태 관리자(LSM)는 모니터 노드의 부하 조절 관리자에 자신의 부하 상태 정보를 알린다.(알고리즘 3)

```
Algorithm LSM ( )
{   if(Status = = LIGHTLOAD OR Status == HEAVYLOAD) {
      SendMessage(MonitorNode. Si,nl),
      Starts a timer(),
    }
   wait(ACK OR timer runs out); /*응답을 기다림*/
   if(ACK AND Status == HEAVYLOAD) {
      JSM(Selectprocess(T_Q, Process_id));
      ProcessMigration(Process_id, Receive_Address);
   } else if(timer runs out) Exit(),
   Terminates the load Balancing().
}
```

(알고리즘 3) Si 노드측 알고리즘

2) 모니터 노드측

S_i노드로부터 부하 상태 변화를 통보 받은 LBM은 통보 받은 부하 상태를 체크한다.(알고리즘 4)

```
Algorithm LBM()
{
   Wait(ReceiveMessage(Sj, nl)),
   if(Status == HEAVYLOAD){
      if(R_LIST != NULL) {
         ACKMessage(Sj, ReceiveAddress(R_LIST));
         Update(R_LIST, Si);
    } else {
        ACKMessage(Sj, MonitorNDAddress),
        Starts a timer, /* 타이머를 작동시키고 프로세스를 수신 */
        wait(ReceiveProcess_id(sj,process_id) or timer runs out);
        if(Sch_Q != FULL){
            Insert_Sch_Q(Process_id),
     } else
        insert_Trans_Q(Process_id),
        GroupLoad = HEAVYLOAD;
        Broadcasts(HEAVYLOAD),
        InsertGroupAlgorithm = SET,
      }
    }
} else if(Startus == LIGHTLOAD) {
    if(Trans_Q != NULL){
        ACKMessage(Sj, SenderAddress(Trans_Q)),

    } else if(R_LIST == NULL OR
        LOAD(R_LIST)> LOAD(Sj)) {
        if(S_LIST != NULL){
          ACKMessage(S_LIST(),Sj Address),
            Update(S_LIST, Sj), }
         else {
            ACKMessage(LSM,Si,"Moderate"),
            Update(R_LIST, Sj),
            GroupLoad = LIGHTLOAD,
            Broadcasts(LIGHTLOAD),
            InterGroupAlgorithm = SET,
            }
       }
   }
}
```

(알고리즘 4) 모니터 노드측 알고리즘

① 과부하 임을 통보 받았을 경우.

LBM은 상태 유지 및 갱신 모듈(SMUM)에게 시스템의 상태를 의뢰하여, 자신의 수신자 리스트를 탐색토록 하여 부하가 가장 적은 노드의 주소(S_i)를 목적지 노드로 결정하여 응답 메시지를 요청한 S_i노드에 보낸다 그리고 수신자 리스트의 S_i의 부하 정보를 갱신한다 만약 수신자 리스트의 모든 부하 정보가 같을 경우에는 리스트의 맨 선두에 있는 노드의 주소를 목적지 노드로 선정하여 응답한다

② 저부하 임을 통보 받았을 경우

노드 Sj로부터 저부하 상태임을 통보 받은 LBM은 우선 자신의 이주 큐를 탐색한다. 이 때 이주 큐에 제출되어 있는 직업이 있다면, 이주 큐에 있는 프로세스를 노드의 원래 주소와 함께 노드 Si에 전송한다. 만약, 이주 큐가 텅빈 상태라면, SMUM에게 시스템의 상태를 의뢰하며, 이 시스템 상태를 기반으로 노드 Sj의 부하 정보와 수신자 리스트에 있는 노드들의 부하 정보를 비교한다 만약 수신자 리스트가 비어 있거나 노드 Sj의 부하가 수신자 리스트의 부하들 보다 적으면, 모니터 노드는 노드 Sj를 수신자로 결정하고 송신자 리스트에 있는 송신자 노드(Si)에게 메시지를 전송하고 송신자 리스트에 있는 Si노드를 삭제하거나 갱신시킨다.

3.2.4 그룹간 알고리즘

그룹간 알고리즘의 가동 시기는 임의의 시간에 그룹 G_i의 모니터 노드의 부하 상태가 과부하로 판명되었을 경우, 자신의 부하 상태를 다른 그룹에 방송함으로써 가동된다.

1) 과부하 그룹측

자신의 부하 상태가 과부하임을 방송한 그룹 G_i는 타이머를 작동시키고 응답을 수집한다. 방송을 수신한 그룹 G_i의 모니터 노드는 자신의 부하 상태를 체크하여 자신의 부하 상태가 저부하라면 방송에 응답을 하고 타이머를 가동시킨다 만약 저부하가 아니라면 방송을 무시해 버린다. 타이머가 종료될 때 까지 응답이 없으면 부하 조절은 실패로 종료된다.(알고리즘 5)

```
Algorithm HEAVYLOADGroup()
{
     wait(InterGroupAlgorithm == SET);
     if(GroupLoad == HEAVYLOAD) {
     starts a time(),
     do {
       collects all the replies(), /* 응답 메시지 수집 */
     } while(time runs out),
     if(Sch_Q == FULL) {     /* 스케쥴 큐가 꽉찬 경우 */
          select one destination Group();
          selectprocess(T_Q, Process_id, Receive_Address);
       } else EXIT(),
   }
   terminates the load balancing().
}
```

(알고리즘 5) 과부하 그룹측 알고리즘

2) 저부하 그룹측

과부하 그룹으로부터 방송을 수신한 저부하 그룹의 모니터 노드는 자신의 부하 상태를 체크하여 방송에 응답하고 타이머를 작동시킨다. 타이머 종료시 까지 프로세스 이주가 없으면 부하 조절을 종료하고 과부하 그룹으로부터 프로세스가 이주해 오면 그룹 G_i의 모니터 노드를 스케줄 큐에 삽입하고 부하 조절을 종료한다 (알고리즘 6)

```
Algorithm        LIGHTLOADGroup()
{
    wait(InterGroupAlgorithm == SET),
    if(GruopLoad ~=LIGHTLOAD) {
        reply with information ();
        starts a timer (),
        do {
          receive a migrated jop()
          insert_Sch_Q(Process_id);
          exit();
        } while(timer runs out),
      } else EXIT(),
       terminates the load balancing();
}
```

(알고리즘 6) 저부하 그룹측 알고리즘

3.3 부하조절 시스템 모델

본 논문에서는 노드에 결함이 발생했을 때 동적 부하조절을 통한 Fail-Soft를 지원하는 n개의 노드로 구성된 분산 시스템으로 가정한다. 각 노드는 1개 또는 그 이상의 자원요소(CPU, I/O장치 등)를 가지며 이중으로 구성될 수도 있다. 노드 i의 작업 도착율은 λ_i 로써 지수분포 형식이다. 노드 i(source node)에 도착한 작업은 노드 i에서 처리되거나 부하조절을 위하여 통신망을 거쳐 다른 노드 j(support node)로 전송될 수 있는데 이를 위하여 동적 부하조절 알고리즘이 실행된다. 각 노드는 배정된 작업들의 대기를 위한 큐잉 메카니즘을 갖는다. 각 노드의 큐에는 원격 노드로부터 과부하 또는 결함의 발생으로 부하 분산을 위해 이주되어온 외부작업을 위한 이주작업 큐(migrant queue)와 노드에 새로 입력되는 로컬 작업인 내부작업은 작업 큐(job queue)에 대기하게 된다. 노드 i에서 노드 j로의 작업이주에 대한 결정은 노드 i의 부하상태에 따라 부하조절 알고리즘에 의해 이루어지고 전송된 작업은 노드 j에서 서비스 받고 결함이 발생되지 않는 한 다른 노드로 전송되지 않는다. 노드 j에서 처리가 끝나면 결과는 원 노드인 노드 i로 보내며 응답경로는 이주 받을때와 다를 수도 있다. 노드 i에서 작업의 서비스율은 μ_i 로, 노드 i에서 노드 j로의 작업이주는 R_{ij}로 나타낸다. 지연시간에는 큐잉과 처리에 소요되는 노드 지연시간, 작업과 관련 메시지의 전송에 소요되는 통신 지연시간으로 이루어진다. 소스 노드i와 지원노드j를 갖는 작업에 대한 평균 통신지연시간은 작업을 원 노드로부터 지원노드로 이주하는데 따른 지연시간과 응답을 소스 노드로 보내는데 따른 지연시간으로 이루어진다[Iyer84].

3.3.1 부하조절 기법의 설계

결함노드의 부하조절로 인하여 특정 노드가 과부하되지 않도록 균등하게 분할하되 시스템의 안정상태가 유지되어야 한다. 그런데 노드 결함시 대치할 수 있는 방법으로는 노드가 수리, 복구 될때까지 결함 노드의 큐에서 작업들을 대기시키는 방법, 다른 정상 노드들에게 조절하는 방법, 끝으로 예비 노드로 이주하여 작업을 계속 실행하는 방법이 사용될 수 있다 그러나 마지막 방법은 과다한 비용이 소요되므로 본 논문에서는 고려대상에서 제외하고 자동으로 결함의 진단/복구를 지원하는 체제를 갖춘 시스템으로 복구 시간이 통신 지연시간에 비해 아주 짧은 경우에 적합한 첫번째 방법과 다중 처리기로 구성된 분산 시스템이나 통신기술의 발전으로 통신지연이 작을 것으로 가정하고 두번째 방법을 연구 대상으로 하였다 처음 두가지 방법은 다음과 같이 혼합하여 사용될 수 있다 여기서 기법 D의 경우는 결함 노드의 대기 큐에 대기하던 작업들이 새로 도착하는 작업들보다 처리가 늦어지므로 평균 응답시간이 길어지게 되어 실시간 시스템과 같은 곳에서는 바람직한 기법이 못되므로 연구대상에서 제외하였다.

		대기 큐에 있는 작업들	
		대 기	조 절
새로 도착하는 작업들	큐에서대기	기법A	기법B
	조 절	기법D	기법C

대기큐에 있는 작업들은 [기법A] 부하의 조절을 일체 실시하지 않는다.

결함이 발생한 노드의 대기 큐에 있는 작업은 물론 수리중에 도착하는 모든 작업들도 버퍼링 하였다가 수리가 완료된 후 재시작 한다. 이 기법은 짧은 시간내에 수리가 가능하거나 작업의 이동에 통신 지연이 큰

시스템, 또는 소프트 실시간 분산 시스템 등의 경우에 적용할 수 있을 것이다.

[기법B] 대기큐 내 부하의 부분적인 조절을 한다.

결함이 발생한 노드의 대기 큐에 있는 작업을 모두 (또는 일부) 정상 노드들에게 균형이 되도록 분배하고 (나머지와) 수리기간에 도착하는 작업들은 버퍼링 하였다가 수리가 완료된 후 재시작 한다. 이 기법은 단기간 내에 복구가 가능하거나 통신비용이 무시 못할 정도인 경우, 또는 치명적인 결과를 초래하지 않을 정도의 하드 실시간 분산 시스템에서는 고려해 볼 수 있는 기법이다

[기법C] 모든 작업을 조절한다.

결함노드의 대기큐 내의 작업은 물론 수리 중에 도착하는 새로운 작업들도 정상 노드로 조절한다 이는 결함의 발생으로 인하여 치명적인 결과를 초래하거나 복구가 불가능한 경우 통신지연의 대소에 관계없이 결함 노드에 할당된 모든 부하는 정상 노드들에게 균등하게 분할되어야 한다. 또는 작업의 이주에 따른 통신지연이 별 문제가 없을 정도로 아주 적고 모든 부하를 이주하더라도 시스템의 안정에 경향이 없다고 판단되는 상황에 적용할 수 있을 것이다. 기법 B와 기법 C를 사용할 경우 작업 큐에 대기중이던 작업들의 조절은 다음과 같이 3가지 유형이 있을 수 있다.

(1) 하나의 지원 노드를 임의적으로 선택하여 결함 노드의 부하를 일괄이주 시킨다
(2) 정상 노드의 현재 부하 상태와 관계없이 결함노드의 부하를 모든 정상 노드들에게 균등하게 분할하여한다(이는 동적 부하조절 알고리즘에 의해 결함 시점에 모든 노드가 균균형 상태이었을 것으로 가정한 것임).
(3) 결함 노드의 부하를 정상 노드들의 현재부하상태를 고려하여 조절 후에도 정상 노드들의 부하가 가능한한 균형을 이루도록 하여 시스템의 안정이 유지되도록 한다.

(1), (2)는 경우에 따라서는 노드들이 과부하 되어 시스템이 불안정상태가 될 수도 있다. 따라서 본 논문에서는 (3)의 경우만을 고려한다 이때 각 노드가 균형을 이루도록 구현하는데는 2가지 방법이 있을 수 있다

첫째는 현재 큐 길이 정보를 사용하여 큐 길이가 가장 짧은 노드로부터 차례로 결함노드의 부하를 조절하여 정상 노드들이 균형을 이루도록 한다.

두 번째 방법은 현재 모든 노드의 수용가능 부하량과 정상 노드들의 평균 부하량을 구한 후 각 정상 노드들의 현재 부하량과 평균 부하량의 차이만큼 노드의 수용가능 부하량의 범위 내에서 결함 노드의 부하를 조절하여 시스템의 안정성을 취하도록 한다. 이는 시스템의 노드들이 이종으로 구성되어 각 노드의 처리속도의 큐의 길이가 상이한 경우도 적용할 수 있다. 다음절의 부하조절 알고리즘에서는 두 번째 방법으로 설계하였으나 시뮬레이션에서 확률적 이송의 어려움으로 첫 번째 방법을 취하였는데 균등하게 분할하여 더욱 정확하도록 하기 위하여 한번에 하나의 작업씩 이주시켰다. 이 방법은 조절에 상당한 시간이 소요될 수 있다.

3.3.2 결함시 조절기법을 통한 부하조절 알고리즘

결함이 발생하면 더 이상 작업을 수행할 수 없는 상황이므로 그 노드는 자신이 현재 이떠한 상태에 있든지간에 과부하로 간주하여 저부하 노드를 찾기 위하여 결함진단 모듈은 통신서브 시스템의 LSM에게 결함과 부하 사실을 통보한다. LSM은 결함사실을 모니터노드에게 전송하면서 타이머를 작동시킨다 이후 타이머가 작동하는 동안에 통보를 받은 모니터 노드는 자신의 부하를 측정하여 과부하 상태의 경우 응답메시지를 메시지 큐에 넣는다 각 과부하노드메시지 도착순서는 시스템 상에서 노드의 상위성능 부하량 통신속도 등을 감안할 때 가장 먼저 도착한 노드의 메시지가 현재결함노드의 부하를 받기에 가장 적당한 노드라고 간주한다. 즉 거리가 가깝고 성능이 뛰어나며 부하가 적은 노드는 그만큼 다른 노드에 비해 응답이 빠를 것이다 따라서 선입선출(FIFO)방식으로 처리하기 위하여 응답메시지 수신 자료구조는 큐를 사용한다 만일 각 노드의 작업 큐의 길이가 유한하여 결함노드의 부하를 재조절 하더라도 시스템이 안정상태가 되도록 하기의 부하조절 알고리즘을 사용할 수 있다. 타이머 종료후 메시지 큐가 비어있다면 작업을 받을만한 저부하 노드가 없다는 뜻이므로 결함노드의 수리가 완료 될때까지 버퍼링하고(기법 A)부하조절은 종료된다 그러나 메시지 큐가 비어있지 않다면 결함노드의 부하를 이어받을 저부하의 정상노드가 있음을 의미한다. 만일 메시지 큐의 수용 가능한 작업의 총수가 결함 노드의 작업수 보다 작다면 일부분만을 조절하고 나머지는 결함노드가

수리 될때까지 버퍼링하여 시스템의 안정성을 유지해야한다(기법B). 반대로 결함 노드내 작업의 수가 작다면 모두 조절할 수 있음을 의미한다(기법C). 알고리즘에서 사용하는 조절 기법은 환경의 조건에 따라 시스템의 성능을 좌우할 것이므로 다음 장에서는 여러 조건 하에서 각 기법들에 대한 성능을 평가할 것이다

[단계 1] 결함이 발생하지 않았으면 정상시 부하조절 알고리즘인 동적 부하조절 알고리즘을 실행하여 노드들의 부하 균형을 유지하도록 하고 그렇지 않으면 다음과 같이 처리한다. 결함노드의 결함진단 모듈은 자신의 노드가 결함상태임을 부하조절기와 통신 서브시스템의 LSM에게 통보한다.

[단계 2] LSM는 모니터 노드에게 노드 i가 결함임을 방송하고 타이머를 set한다.

[단계 3] LSM로부터 결함발생 신호를 통보 받은 모니터 노드는 결함진단 모듈내의 LMM은 노드의 부하 상태를 점검한다
만약 저부하 상태 모니터 노드는 자신의 노드 식별자와 현재 부하상태를 통신 서브시스템의 메시지 큐에 넣는다.

[단계 4] Set된 타이머가 종료될 때까지 모니터 노드는 저부하 노드들로부터 부하상태를 전달 받은 후 , LSM은 수용이 가능하면 기법을 A로 설정하고 결함노드의 부하를 자신의 노드에 버퍼링한 후 [단계 8]로 간다.

[단계 5] 모니터 노드의 메시지 큐에 도착한 순서대로 부하 수용가능 범위 내에서 원격 노드로 조절한다(메세지 큐의길이 : j = 1, 2, …, n).
j > L(모니터노드에 도착한 큐의길이)이면[단계 7]로 가고, 그렇지 않으면 조절해야할 노드를 원격 노드로 이주하고 제거해야할 노드를 갱신하고 j의 값을 증가 시킨후 [단계 5]로 가고 그렇지 않으면(마지막 조절일 경우) 조절해야할 노드를 원격노드로 이주하고 정상노드와 조절해야할 노드를 갱신 한후[단계 7]로 간다.

[단계 6] 만일 결함이 복구되었음을 결함 진단/복구 모듈이 감지하면 이 사실을 부하 조절 모듈에게 통보하여 정상의 부하 조절알고리즘을 실행하고 그렇지 않으면 [단계 7]을 처리한다

[단계 7] L > 적정부하 이면 노드를 모두 결함 노드의 버퍼에 버퍼링하고 L = 적정부하 이면 기법을 B로 set하고, 그렇지 않고 L <= 적정부하 이고 일반적인 노드 > 적정부하 이면 기법을 B로 일반적인 노드 > 적정부하 이면 C로 set한다.

[단계 8] 결함노드에 새로이 도착한 작업이 있으면 조절정책에 따라서 처리한다.
만일 기법이 A또는 B인 경우 결함노드의 버퍼에 버퍼링 한후 조절정책에 따라 처리하고, 기법이 C인 경우는 원격노드로 이주하고 현재 수용할 수 있는 노드 j와 정상노드를 갱신한 후 [단계 7]로 간다

3.3.3. 정상일 때 동적 부하조절 알고리즘

분산 시스템에서 각 호스트의 LMM은 주기적(m : 갱신주기)으로 통신 서브시스템의 LSM에게 자신의 노드 식별자와 부하 상태정보를 메시지로서 보고한다. LSM는 각 호스트로부터 입수한 부하정보를 근거로 하여 시스템의 평균 부하를 계산하여 각 호스트의 LMM에게 통보하면 아래의 알고리즘을 수행한다.

[단계 1] 노드 i에 작업이 도착하면 로컬 노드에서 처리할 것인지 원격노드에서 처리할 것인지를 결정한다. 만일 큐에 대기하고있는 평균 CPU 실행시간의 합($\sum_{i=1}^{k}(A_i)$)이 CPU의 처리능력보다 크면 원격노드에서 처리하기 위하여 [단계 2]로 가고, 그렇지 않으면 작업은 로컬 노드에서 처리하도록 하고 알고리즘은 종료한다.

[단계 2] 작업을 원격호스트로 보내기 위하여 각 유형에 따른 가중치를 결정한다.

각 부하정보가 갱신된 후 가중치는 다음과 같이 결정된다.

① 과부하 : CPU실행시간이 CPU처리능력에 대한 상한비 보다 클 경우 프로세스를 이주한다.
② 적정부하 : CPU실행시간이 CPU처리능력에 대한 상한비와 하한비 사이에 존재할 때 프로세스 이주가 필요 없다.
③ 저부하 : CPU실행시간이 CPU처리능력에 대한 하한비보다 적을 경우 프로세스를 이주한다

[단계 3] 가중치에 따라서 작업을 나누어 전송(Dispatch)한다

작업전송기(Job Dispatcher)는 1단계 2)에서 얻은 가중치가 큰 곳에 도착된 작업을 나누어 전송한다.

[단계 4] 원격호스트에 대한 가중치를 조정한다.

선택된 원격호스트로 부터 나누어 전송된 노드의 로컬 작업을 받을 때마다 호스트 노드는 원격 호스트로 보낸 로컬 작업들의 가중치를 기록한다.

그렇지 않으면 가중치는 불변이고 알고리즘은 종료한다.

4. 성능 평가

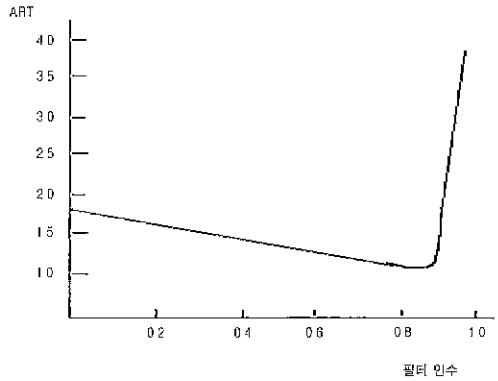
시뮬레이션은 SPSS PC+ 통계 패키지를 이용하였고, 시뮬레이터는 다음과 같은 부분들로 구성된다. 시뮬레이션에 실제 사용한 값들중 평균 결함발생 간격시간과 평균 고장 수리시간의 설정치들은 IBM 3081 프로세서의 실제 결함율과 수리율의 실측치들을 근사 적용하였다[Iyer85]

PRTF	MTBF	MTTR	MTTF	CDT
0.3	49.18	0.49	48.69	0.0
0.6	73.18	0.69	72.49	1.0
0.8	98.67	0.94	97.73	1.5

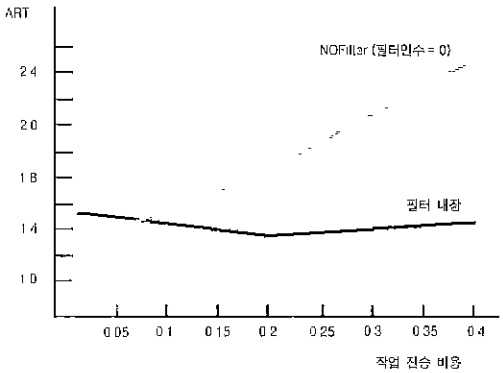
PRTF : 작업도착간격시간 MTBF 평균고장시간 간격
MTTR : 평균고장수리시간 MTTF 평균결함발생시간
CDT : 통신지연시간

정상노드 자기학습 부하조절 모델인 경우 그림에서와 같이 필터가 사용되지 않았을 때 즉 필터 인수가 "0"일 때 부하 조절을 수행하지 않은 경우 (필터 인수가 "1" 일 경우) 보다 커다란 성능 개선을 보이고 있으며, 성능은 필터 인수 F가 약 "0.8" 일 때 최고였으며, 그 이상의 필터 인수에서는 나빠지고 있음을 알 수 있다 (그림 4-1)은 필터 인수에 따른 응답 시간(ART)을 나타내고 있다

CPU처리 시간에 있어서 작업 전송 비용의 함수로, 필터를 사용하지 않은 경우(F = "0")와 필터를 사용했을 경우의 작업 응답 시간을 (그림 4-2)에서 나타내고 있다. 필터를 사용하지 않았을 경우, 작업 전송 비용을 증가시킴에 따라 급격히 나빠짐을 알 수 있다 이것은 작업 전송 비용 부담으로 인한 전송할 가치가 없는 잡은 작업을 부하 조절을 시도함으로써 나타나는 탐색 비용의 증가 때문이다.



(그림 4-1) 필터 인수에 따른 평균 작업 응답 시간, 시스템 부하 = 54.0 %



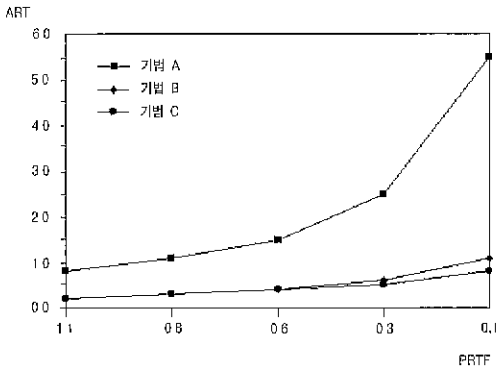
시스템 평균 부하 : 54.0 %, 필터인수 : 0.77

(그림 4-2) 작업 전송 비용에 따른 평균 작업 응답 시간

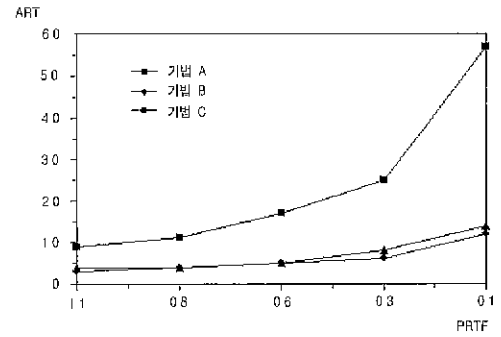
작업 도착 시간간격의 변화는 각 기법의 상대적 효율성에 별다른 영향을 주지 못하고 통신 지연시간이 1.0정도까지는 C, B, A의 순으로 기법이 우수한 것으로 나타났다.

정상노드 자기학습 부하조절 모델은 프로세스의 결함을 진단한 후 결함 발생시 작업유형 테이블을 참조하여 과거 같은 유형의 결함 발생시 수행한 경험을 바탕으로 다른 적당한 원격 노드로 프로세스를 이주시키도록 하고, 결함의 유형에 따라 일시적 결함으로 자체 복구가 가능한 경우로 판단될 경우 복구 프로그램을

실행토록 유도하여 자체 결함 복구하도록 하고, 영구 결함으로 판단될 경우 자신의 노드가 결함임을 모니터 노드에 통보하고 결함이 탐지되지 않은 경우 오랜 수행 시간이 걸리는 작업들을 부하 조절기에 넘겨 부하 균등화를 이루도록 하며, 짧은 수행 시간이 소요되는 작업들은 지역적으로 처리하기 위해 스케줄 큐에 넣어 가도록 함으로써 적당한 원격노드를 찾기 위한 통신비용과 탐색 비용을 줄일 수 있으며, 낮은 필터 실행 임계치는 낮은 시스템 부하와 낮은 전송 비용에서 적당한 반면, 높은 필터 실행 임계치는 높은 전송 비용에서 더 나은 성능을 보임을 알 수 있다.



(a) MTTR : 0.49, CDT : 0 일 경우



(b) MTTR : 0.49, CDT : 1.0 일 경우

(그림 4-3) 작업 도착 시간간격이 작업응답시간에 미치는 영향

결함진단 부하조절 모델은 기법들간의 효율성 차이에 영향을 미치는 주요 요인으로는 통신지연시간, 결함 수리시간이다. 기존의 부하균형기법에서 통신지연시간이 큰 경우에는 자신의 노드에서 처리되어야할 작업량이 많아지므로 본 논문에서는 결함진단 부하조절 필터에서 부하조절기를 거치지 않고 직접 스케쥴큐로 보냄으로써 처리되는 속도가 향상됨을 알 수 있으며, 통신지연시간이 없거나 평균 작업처리시간과 비슷한 (1.0 정도) 경우는 기법 C가 가장 우수했으나 통신지연시간이 그 이상 증가하는 경우는 상대적으로 기법 A와 B는 유리해지나 기법 C는 상대적으로 불리하게 된다. 이는 기법 C의 경우 부하 조절량이 많기 때문이다. 즉 통신지연이 큰 경우는 조절 작업으로 인한 비효율성이 결함 집중으로 인한 비효율성을 크게 상회하기 때문이다. 또한 결함 수리시간이 증가하는 경우 결함 노드의 작업들을 수리시간동안 대기하도록 하는 기법 A는 상대적으로 불리해 지지만 모든 결함 노드의 작업들을 조절시키는 기법 C는 상대적으로 유리해진다. 이는 결함수리시간이 클수록 결함집중으로 인한 비효율성이 증가되기 때문이다.

5. 결 론

분산 처리 시스템에서는 지리적으로 분산된 자원을 최대로 활용하기 위해 전체 시스템 내에서 처리되어야 할 작업 부하를 각 프로세서에 균등하게 배정함으로써 시스템을 효율적으로 운영할 수 있다. 오늘날의 결함허용 분산 시스템에서 기존의 중복 메카니즘 외에 부하 조절 기법을 통한 결함허용도 고려 대상이 될 수 있을 것이다.

시뮬레이션 결과 작업의 도착 간격시간과 결함 발생 간격시간의 변화는 조절 기법들간의 상대적 효율성에 민감한 영향을 주었다. 통신 지연시간이 평균 작업처리시간보다 큰 경우에는 모든 작업을 조절하는 기법 C보다는 부분 조절하는 기법 B가 우수하며 통신 지연시간이 평균 작업처리시간과 유사하거나 작을 경우에는 기법 B나 조절을 일체 실시하지 않는 기법 A보다는 기법 C가 현격하게 우수하였다. 파라미터 값들의 변화에 따른 기법의 상대적 효율성이 커다란 변화 없이 일반성을 나타낸 기법은 B이었으며 기법 A는 결함수리시간이 작은 경우, 즉 신속한 수리가 가능한 경우와 통신지연시간이 큰 경우에 적합하며 기법 C는 통신지연시간이 작거나 결함수리시간이 큰 경우 적합하였다.

또한 자기 학습 부하 조절기법의 결함진단 부하조절 필터는 새로운 작업을 모두 받아들여서 프로세스의 결

함을 진단한 후 결함 발생시 작업유형 테이블을 참조하여 과거 같은 유형의 결함 발생시 수행한 경험을 바탕으로 다른 적당한 원격 노드로 프로세스를 이주시키도록 하고 원격으로 실행할 가치가 없는 짧은 작업을 찾아 로컬에서 처리하도록 하는 필터를 내장하였으며, 이러한 필터는 결함이 탐지되지 않은 경우 오랜 수행시간이 걸리는 작업들을 부하 조절기에 넘겨 부하 균등화를 이루도록 하며, 짧은 수행 시간이 소요되는 작업을 부하 조절기를 거치지 않고 스케줄 큐에 들어가도록 하여 로컬에서 처리하도록 함으로써 적당한 원격 노드를 찾기 위한 통신비용과 탐색비용을 줄일 수 있었다

본 논문에서는 스케줄 큐와 이주 큐에 제출되어 있는 작업의 과거 서비스 시간을 기초로 각 노드들의 시스템 처리능력에 따라 부하 상태를 결정토록 하였다. 작업 선정정책에 서는 인공신경망 분야에서 사용하는 Weight Climbing기법을 적용하여 작업을 선정함으로써 자신의 과거 실행한 경험으로부터 한 작업의 행위를 학습하여 그 작업을 전송하는 것이 가치가 있는가 없는가에 관한 올바를 판단을 할 수 있다 따라서 스스로 시스템을 구성 또는 프로그램의 행위들이 변경되었을 때 즉각적으로 반응하여 새로운 환경에 능동적으로 대처할 수 있는 기법을 제안하였다. 앞으로의 연구에서는 작업 선정 모듈에서 학습을 통한 적절한 작업 선택이 시스템의 성능에 미치는 영향과 각 노드간의 통신속도, 실제 분산시스템 환경에서 실행될 수 있는 부하조절알고리즘과 노드 결함시 부하조절 알고리즘을 결합한 통합알고리즘의 연구가 더 진행되어야 하겠다.

참 고 문 헌

[Arms84]Armstrong, C. V M and Fathi. E. T, "A Fault-Tolerant Multi micro processor-based Computer System for space-based Signal Processing," IEEE Micro, Vol.4. No 8, pp 54-65, 1984.

[Bara85]A. Barak & A. Shiloh, "A Distributed Load-balancing policy for a multiprocessor," Software Prac. & exper, Vol.15(9), pp.901-913, Sep. 1985.

[Bren88]P. A. Brenstein. "Sequoia : A Fault-Tolerant Tightly Coupled Multiprocessor for Transaction Processing." Computer, pp.37-45, Feb. 1988

[Eage86]Eager D L, E D. Lazowska, and J Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed System," IEEE Transactions on Software Engineering, Vol.12, No.5. pp.662-675, May 1986.

[Emme84]R Emmerson, and McGowan. M, "Fault Tolerance Achieved in VLSI," IEEE Micro, 1984

[Hsu86]C. Y H Hsu & J. W. S. Liu, "Dynamic Load Balancing Algorithm in Homogeneous Distributed Systems," The 6th Int. Conf. on Dis Comp. Sys., IEEE, pp 216-223, 1986.

[Huan80]H H Huang, "Fault-Tolerant design of a modern receiving system." Proceeding of FTCS-10, pp.375 378, 1980

[Iyer84]R K Iyer, "Reliability Evaluation of Fault-Tolerant System-Effect of Variability in Failure Rates," IEEE Trans. on computers, Vol.C-33, No.2, pp.197-200, Feb 1984.

[Iyer85]R K Iyer, D. J. Rossetti. "Effect of system workload on operating system reliability : A study on IBM 3081," IEEE SE-11. pp.1438-1448, Dec. 1985.

[Kim92]K. H Kim, "Design of Real-Time Fault-Tolerant Computing Stations," Lecture note in the NATO Advanced Science Institute on Real-Time Computing, Sint Maarten, Oct. 1992

[Lin87]F. C H Lin and R. M. Keller, "The gradient model load balancing method," IEEE on SE, Vol SE-13, No.1, pp 32-38, Jan. 1987.

[강88]강태규, 최용락, 김영찬, "분산시스템의 동적 부하균형 알고리즘 연구", 한국정보과학회 논문집, 제15권 제2호, pp.377-380, 10월, 1988.

[김94]김대수, "신경망 이론과 응용(I), (II)", 하이테크정보, 1994.

[김98]김기태, "인공지능의 기법과 응용", 기한재, 1998.

[장94]장순주, 임종규, 구용완, "분산시스템에서 결함허용성을 위한 프로세스 이주연구", 한국경보과학회 추계학술발표논문집, 제21권 제2호, pp.131-134, 1994

장 순 주

e-mail : sjcchang1@hanmail.net

1989년 대전산업대학교 전자계산학과 졸업(학사)

1995년 수원대학교 전자계산학과 (이학석사)

2000년 수원대학교 전자계산학과 (이학박사)

1997년~현재 수원대학교 자연과학연구소 객원연구원

관심분야 : 분산운영체제, 멀티미디어 시스템, 시스템 네트워크, 결함허용시스템.

구 용 완

e-mail : ywkoo@mail.suwon.ac.kr

1976년 중앙대학교 전자계산학과 졸업(학사)

1980년 중앙대학교 전자계산학과 졸업(이학석사)

1988년 중앙대학교 전자계산학과 졸업(이학박사)

1983년~현재 수원대학교 전자계산학과 교수 및 전자계산소장

관심분야 분산운영체제, 멀티미디어 시스템, 시스템 네트워크, 결함허용시스템