서버 클러스터 시스템을 위한 고장 감내 소프트웨어 개발 환경

함명호[0] 김진용 신현식
서울대학교 컴퓨터공학부
(mhham[0], sugar)@cslab.snu.ac.kr, shinhs@snu.ac.kr

Fault-Tolerant Software Development Environment for Server Cluster Systems

Myung-Ho Ham[0] Jin-Yong Kim Heon-Shik Shin
School of Computer Science and Engineering, Seoul National University

요 약

분산 시스템 환경에서 하드웨어나 소프트웨어 자원의 가용성이나 신뢰성을 높이기 위한 노력으로 가용성이 높은 클러스터 시스템이나 고장 감내 소프트웨어 개발 환경들이 연구되어 왔다. 본 연구에서는 하드웨어의 신뢰성을 높이기 위해 서버 클러스터 시스템을 구축하였고, 이 클러스터 시스템에 기반한 고장 감내 소프트웨어 개발 환경을 구축하였다. 사용자는 고장 감내 소프트웨어 개발 환경을 이용하여 쉽게 고장 감내 소프트웨어를 작성할 수 있고, 원하는 소프트웨어 고장 감내 기법을 간단하게 기술할 수 있다. 특히, 소프트웨어 개발과 소프트웨어 고장 감내 기법의 적용을 논리적으로 분리시켜 소프트웨어 개발 과정을 단순화 시켰고, 이미 개발된 소프트웨어 모듈의 변경 없이 다양한 고장 감내 기법을 적용할 수 있게 하였다. 이러한 개발상의 논리적인 분리, 소프트웨어 모듈의 동적 노드 결정, 그리고 작업 스케쥴링 등의 일을 처리하기 위해 실행 시간 제공 요소(Run-time supports)들이 노드와 네트웍 고장을 감내하기 위해 개발된 미들웨어 계층 위에서 구현되었다.

1. 서 론

분산 시스템 환경에서 고장 감내 기법에 대한 연구는 오랫동안 중요하게 다루어졌고 다양한 형태의 연구들이 진행되어 왔다. 시스템의 가용성이나 신뢰성을 높이기 위해서는 하드웨어의 고장 감내뿐만 아니라 그 시스템에서 실행되는 소프트웨어의 고장도 감내 할 수 있어야 한다. 일반적으로 많이 사용되는 C언어나 JAVA와 같은 고급 프로그래밍 언어를 사용해서 분산 고장 감내 소프트웨어를 개발하는 것은 투자한 노력에 비해 비효율적이다[1]. 분산 환경을 위한 프로그램은 Ada와 같은 언어나 OMG의 CORBA 또는 OSF의 Distributed Computing Environment(DCE)와 같은 미들웨어 수준의 환경 지원으로 보다 쉽게 개발될 수 있다[2]. 그러나 이러한 개발 환경들 역시 고장 감내 소프트웨어를 개발하기 위한 환경이 지원되지 않기때문에 개발시간이 지연될수 밖에 없다.

이러한 비효율성을 극복하기 위하여 고장 감내 소프트웨어를 손쉽게 개발할 수 있는 환경들이 연구되어 왔다. 고장 감내 소프트웨어 개발환경은 크게 3가지 형태로 나눠볼 수 있다[2]. 첫째, 기존에 존재하는 프로그래밍 언어에 라이브러리 형태의 함수들을 지원하는 것이다. 이 방법은 좋은 성능을 낼수 있지만 프로그래머에게 고장 감내 논리에 대한 투명성을 제공하지 못하여 프로그래밍을 복잡하게 만든다. 프로그래머는 고장 감내 프로그래밍에 대한 전문가적인 지식이 있어야만 한다. 둘째, 고장 감내 소프트웨어 개발을 위한 새로운 언어이다. 이 방법은 고장 감내 기법에 대한 추상화를 제공함으로써 프로그램 작성을 용이하게 만든다. 그러나 이 언어에서 제공하는 방법 이외의 기법을 적용하는데 어려움이 있다. Argus[3]나 Plits[4]가 이러한 언어의 예이다. 셋째, 분산 프로그래밍을 위한 언어를 고장 감내 기법을 기술할 수 있도록 확장한 언어이다. Ada 언어를 확장한 replicAda[2], SR 언어를 확장한 FT-SR[1]등이 대표적인 예이다.

본 연구에서는 미들웨어에 기반한 고장 감내 소프트웨어 개발 환경을 구축하였다. 이 미들웨어는 분산된 노드들의 고장과 네트웍 고장을 발견하고, 고장 발생시 또 다른 경로를 이용하여 통신할 수 있도록 보장해 준다. 미들웨어에서 제공되는 API를 이용하면 각 노드들의 상태 정보를 알 수 있고, 노드간의 통신을 추상화 시켜 손쉽게 분산 프로그램을 작성 할 수 있다[5]. 고장 감내 소프트웨어 개발 환경을 지원하기 위해 실행 시간 제공 요소(Run-time Supports)들이 설계되었다. 실행 시간 제공 요소는 작업의 스케쥴링, 작업의 동적 노드 할당, 다른 노드로의 작업 전송, 작업의 시작과 종료 조정, 그리고 작업의 실행을 담당한다. 소프트웨어의 디자인 결함을 감내하기 위해서 중복된 소프트웨어 모듈을 작성해서 이를 정해진 소프트웨어 고장 감내 기법을 적용하게 된다. 이때 소프트웨어 전체가 중복될 필요는 없다. 이는 시간적인 자원과 공간적인 자원의 낭비를 초래하여 효율적이지 못할 뿐만 아니라 소프트웨어 성능에도 비효율성을 초래할 것이다. 그 대신 신뢰성이 요구되는 모듈을 따로 분리하여 소프트웨어 고장 감내 기법을 적용하는 것이 효율적이다.

기존 연구에서는 고장 감내 소프트웨어 개발시 해당 모듈이 실행되는 노드가 컴파일시 정적으로 정해지는 문제

가 있다. 이 경우 예측하지 못한 노드나 네트웍의 결함에 능동적으로 대처하지 못하는 문제점이 있다. 본 연구에서는 실행 시간 지원 요소(Run-time Supports)들의 도움으로 이러한 동적인 하드웨어 자원의 결함에 대처하는 방법을 제공한다. 또한 기존에 존재하는 다양한 종류의 소프트웨어 고장 감내 기법을 적용할 수 있을뿐만 아니라, 사용자가 정의 하는 임의의 기법을 기술할 수 있는 방법론을 제시한다.

2. 서버 클러스터 시스템의 구성

본 연구에서 기반을 두고 있는 클러스터 시스템은 노드의 고장 뿐만 아니라 네트웍 고장에 대한 신뢰성을 더욱 높이기 위하여 이중 채널을 사용하여 구축되었다[6]. 특히 다양한 네트웍상의 고장을 극복하기 위해 고안된 구조로 (1) 허브 결함, (2) 네트웍 링크 결함, (3) 네트웍 인터페이스 카드(NIC)의 고장을 극복할 수 있다(그림 1).

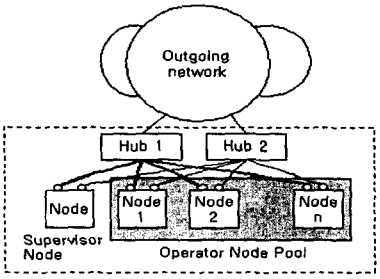

그림 1 이중 채널 기반 서버 클러스터의 구조

본 시스템은 기능별로 관리자 노드와 작업 노드로 나뉘어 진다. 관리자 노드는 작업 노드 풀을 관리하며 외부로부터 전달되는 요청을 선택된 작업 노드에게 전달하는 역할을 한다. 작업 노드풀은 현재 서비스 가능한 노드들이며 같은 동작을 수행하는 프로세스들이 준비된다. 실행 결과는 관리자 노드에게 반환된다. 만약 관리자 노드가 고장이 난다면 노드풀의 또다른 노드가 새로운 관리자 노드로 선택되어 동작을 계속하게 된다.

각 노드의 고장이나 네트웍의 고장을 발견하고 이러한 정보를 유지하기 위하여 고장 감내 미들웨어를 개발하였다. 고장 감내 미들웨어는 여러 개의 관리자로 구성이 된다. 고장 검출 관리자(Fault Detect Manager)는 UDP 하트비트(Heartbeat) 메시지를 사용하여 노드들의 상태를 검사하고 AST(Active Stations Table)를 유지한다. 고장 감내 통신 관리자(FT Communication Manager)는 네트웍 채널의 고장이나 노드의 고장시에도 관리자 노드와 작업 노드간의 신뢰성 있는 통신을 보장한다. 시스템 자원 관리자(System Resource Manager)는 관리자 노드와 작업 노드 풀에 대한 갱신된 정보를 관리하며, 상위 애플리케이션 계층에 정보를 제공한다.

3. 고장 감내 소프트웨어 개발 환경

3.1. 고장 감내 소프트웨어 프로그래밍 모델

본 연구에서 개발한 고장 감내 소프트웨어 개발 환경은 손쉬운 고장 감내 소프트웨어의 개발, 항상 발생할 수 있는 하드웨어 자원의 고장에 대비한 동적인 소프트웨어 모듈의 할당, 그리고 고장 감내 기법의 적용을 동적으로 기술할 수 있는 모델을 목표로 하였다. 그림 2는 이러한 목표를 위해 설계된 시스템의 구조도이다.

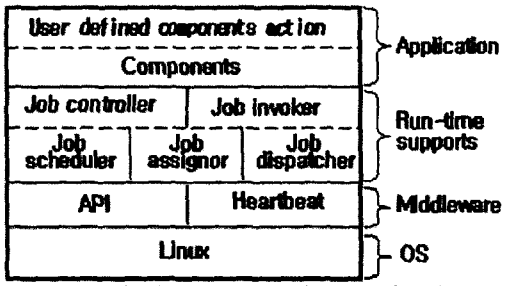

그림 2 고장 감내 소프트웨어 개발 환경의 구조

기존의 고장 감내 소프트웨어 개발 방법은 소프트웨어 개발 시 고장 감내 기법에 대한 프로그래밍도 같이 하여 컴파일 하여 실행파일을 생성하는 방식이였다. 이런 방법은 소프트웨어 고장 감내 기법을 정적으로 고정시켜 다양한 고장 감내 기법을 적용하기 힘들 뿐만 아니라, 다른 고장 감내 기법을 적용하기 위해서 전체 소프트웨어를 재컴파일해야 하는 부담이 있다. 고장 감내 기법을 동적으로 기술할 수 있게 하기 위해서는 소프트웨어 고장 감내 기법의 단위 요소들을 분리할 필요가 있다. 이들 단위 요소는 모듈(Module), 투표자(Voter), 수용 검사(Acceptance test)의 3요소로 분리하였다.

3.1.1. 동적 컴포넌트 실행 모델(Dynamic components execution model)

동적 컴포넌트 실행 모델은 고장 감내 소프트웨어 개발시 고장 감내 알고리즘과 각 모듈간의 통신 메커니즘을 컴포넌트로 추상화 시킴으로써 복잡한 고장 감내 알고리즘 및 내부 알고리즘을 논리적으로 분리시켜 소프트웨어 개발을 손쉽게 만든다. 또, 프로그램의 재작성 과정 없이 상황에 따라 다양한 고장 감내 기법을 동적으로 적용할 수 있게 한다.

동적 컴포넌트 실행 모델을 위해 고장 감내 기법의 각 단위 요소를 정의할 필요가 있다. 본 연구에서는 모듈, 투표자, 수용 검사의 3요소로 컴포넌트를 정의하여 구현하였다. 이러한 컴포넌트를 구현하기 위해 흔히 사용되는 소프트웨어 고장 감내 기법에서 각 컴포넌트의 역할을 일반화 시켜야 한다. NSCP(N self-checking programming), NVP(N-version programming), RB(Recovery blocks), CRB(Consensus recovery block)등의 일반적인 소프트웨어 고장 감내 기법[7]을 통해 각 컴포넌트의 입출력 패턴을 정의하였다. 표 1은 컴포넌트의 입출력 개수를 나타낸다.

그림 3은 정해진 컴포넌트의 입출력 패턴을 바탕으로 설계된 모듈(Module) 컴포넌트의 슈도 코드이다. 이 코드

표 1 컴포넌트(Components)의 입출력 개수

I/O / Components	Input		Output	
	Data	Control	Data	Control
Module	1	0	n	0
Voter	n	1	1	0 or 1
Acceptance test	1	1	1	0 or 1

에서 compute 함수 부분은 이 컴포넌트에서 처리해야 할 실제 계산 부분이며, 실제 코드에서는 별개의 컴파일된 프로그램으로 제공되어야 한다. 따라서 컴포넌트들은 다른 프로그램을 실행시키는 기능과 프로세스간의 통신에 관한 기능을 갖춰야 한다.

```
Module
{
    initialize(); // to use FT job controller's function
    input = receive(from input component, timeout value);
    output = compute(input, timeout value);
    send(output to voter[0, ..., v-1]);
    send(output to Acceptance test[0, ..., t-1]);
    finalize(); // the end of using FT job controller's function
}
```

그림 3 모듈 컴포넌트의 슈도 코드

3.1.2. 정적 컴포넌트 실행 모델(Static components execution model)

동적 컴포넌트 실행 모델에서 정의한 컴포넌트들은 일반적으로 많이 쓰이는 소프트웨어 고장 감내 기법들에는 적용 가능하다. 그러나 사용자가 요구하는 임의의 형태의 고장 감내 기법을 적용하기는 힘들다. 사용자는 컴포넌트를 재설계하거나 또는 완전히 새로운 컴포넌트를 정의 할 수 있다. 새로운 컴포넌트를 구현하는 어려움을 줄이기 위하여 템플릿 코드가 제공된다. 이 템플릿 코드는 사용자가 정의하는 새로운 컴포넌트의 입출력 형태를 기반으로 생성된다.

3.2. 소프트웨어 고장 감내 기법의 기술

각 컴포넌트들(모듈, 투표자, 수용 검사등)은 실시간으로 데이터를 교환하며 상호 유기적인 동작을 한다. 소프트웨어 고장 감내 기법을 기술하기 위해서는 이 데이터들의 패턴을 분류하고 상호 동작에서의 데이터 흐름을 나타낼 수 있어야 한다. 데이터의 패턴은 각 컴포넌트의 컴퓨팅을 위해 필요한 데이터와 컨트롤을 전환하기 위한 활성화(activation) 신호가 있다.

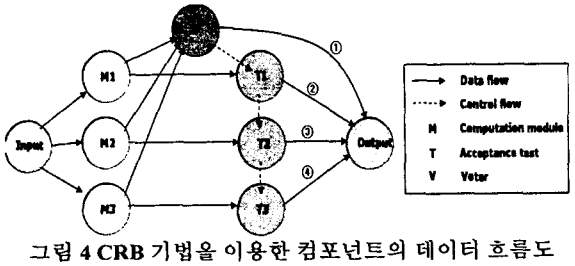

그림 4 CRB 기법을 이용한 컴포넌트의 데이터 흐름도

예를 들어, 그림 3과 같은 CRB기법을 살펴보면 실선으로 된 데이터 흐름과 점선으로 된 컨트롤 흐름을 볼 수 있다. 각 컴포넌트의 입출력 관계를 나타내기 위한 형식을

컴포넌트 = {(data input), (control input), (data output), (control output)}

와 같이 정의 하면, Voter에 대한 입출력 관계는

Voter = {(M1, M2, M3), ( ), (Output), (T1)}

와 같이 표현할 수 있다.

3.3. 실행 시간 제공 요소(Run-time supports)

실행 시간 제공 요소는 미들웨어와 응용 소프트웨어 사이에 위치하여 보다 효율적으로 고장 감내 소프트웨어가 동작하도록 지원한다. 작업 스케쥴러는 서버 클러스터에 요청된 작업의 우선 순위를 결정해 처리하는 역할을 한다. 작업 할당자는 주어진 컴포넌트들을 최적의 신뢰성을 보장하도록 각 노드에 할당해 준다. 작업 발송자는 작업 할당자에 의해 할당된 작업을 각 노드에 전송한다. 작업 조정자는 각 노드에 전송된 작업들을 스케쥴링하고 완료된 작업은 종료시키는 역할을 한다. 마지막으로 작업 실행자는 컴포넌트를 실행시키는 일을 수행한다.

4. 결과 및 향후 연구

본 연구에서는 서버 클러스터 시스템을 위한 고장 감내 소프트웨어 환경을 구축하였다. 이는 고장 감내 소프트웨어 개발 프로세스를 단순화 시켰을 뿐만 아니라 동적인 소프트웨어 모듈의 노드 할당, 그리고 고장 감내 기법의 적용을 동적으로 기술할 수 있는 모델을 제안하였다. 또한 사용자가 정의하는 임의의 고장 감내 소프트웨어 기법도 적용 가능하다.

향후 소프트웨어 고장 감내 기법의 기술을 위한 비쥬얼한 툴을 개발할 계획이며, 현 시스템의 성능 평가가 이루어질 것이다.

참고문헌

[1] Richard D. Schlichting, Vicraj T. Thomas, "Programming Language Support for Writing Fault-Tolerant Distributed Software," IEEE Transactions on Computers, pp. 203-212, 1995

[2] Pedro de las Heras-Quiros, Jesus M. Gonzalez-Barahona, Jose Centeno-Gonzalez, "Programming Distributed Fault Tolerant Systems: The replicAda Approach," proceedings of the conference on TRI-Ada `97, pp. 21-29, Nov. 09-13, 1997, St. Louis, Missouri, United States

[3] B. Liskov,"The Argus language and system," in Distributed Systems: Methods and Tools for Specification, LNCS, Vol. 190(M. Paul and H. Siegert, eds.), ch. 7, pp. 343-430, Berlin: Springer-Verlag, 1985

[4] C. Ellis, J. Feldman, and J. Heliotis, "Language constructs and support systems for distributed computing," in ACM Symp. On Prin. Of Dist. Comp., pp. 1-9, Aug 1982

[5] Sape Mullender, "Distributed Systems, 2nd Ed.", Addison-Wesley, 1993

[6] 함명호, 김진용, 최보곤, 신현식, " 이중 채널 이더넷 기반 가상서버를 위한 분산 고장 감내 미들웨어의 구현" 정보과학회 춘계학술대회, 2003

[7] Dhiraj K. Pradhan, "Fault-Tolerant Computer System Design", Prentice Hall PTR