□특 집□

소프트웨어 결함 허용 기법에 대한 고찰

한양대학교 전자계산학과 허 신*

목 차

I. 서 론
II. 기존의 소프트웨어 결함 허용 기법
2.1 소프트웨어 적응 검사
2.2 일관성 검사
2.3 능력 검사
2.4 N-Version Programming
III. 후향 오류복구의 개념을 기초로 한 소프트웨어 결함 허용 기법
3.1 복구블록 기법
3.2 분산환경에서의 후향 오류복구 기법
3.3 분산 복구블록 기법
IV. 결 론

I. 서 론

기존 컴퓨터의 불충분한 오류 복구 능력은 높은 신뢰도를 요구하는 실시간 응용 프로그램에 큰 문제점을 제시한다. 결함 허용 컴퓨터 시스템(Fault-Tolerant Computer Systems, FTCS)은 중단없는 실시간 서비스를 위해 시스템의 하드웨어나 소프트웨어 결함 하에서도 시스템을 복구할 수 있는 능력을 갖춘 컴퓨터 시스템이다[1, 2]. 이 FTCS는 실시간 응용 프로그램의 신뢰도와 안정도를 향상시킬 수 있다. 이들 실시간 응용 프로그램은 막대한 재원이나 귀중한 생명를 제어하는 것으로 그 주된 사용에는 우주항공기 제어, 공항 관제탑의 제어, 원자로 리액터의 제어 및 중환자실의 제어 등이다.

하드웨어의 신뢰도를 증가시키기 위한 노력은 오래전부터 계속되어 왔고 하드웨어의 설계는 그 결함률을 감소시키는 결함 허용 구조를 개발하는 방향으로 발전되었다. 그러나 최근에는 소프트웨어 신뢰도의 중요성을 인식하게 되었으며 또한 대부분의 소프트웨어 엔지니어들은 완전히 검증된 소프트웨어 개발은 불가능하다는 것도 인식하게 되었다. 높은 신뢰도를 갖춘 소프트웨어 개발 툴의 사용은 물론 소프트웨어 개발에 의욕적이고 재능을 겸비한 엔지니어들이 참여한 NASA의 우주 항공 발사 프로젝트가 소프트웨어 결함으로 연기되는 경우를 우리는 자주 경험하였다[3,4]. 현재의 소프트웨어 공학의 기술은 증가되는 소프트웨어 시스템의 크기와 그 복잡성을 극복할 수 없는 실정이다. 높은 신뢰도를 갖춘 소프트웨어를 구현하는 비용은 소프트웨어 시스템의 크기와 복잡도에 비례한다. 대형 소프트웨어의 완전한 검증은 불가능하므로, time-critical한 응용프로그램의 결함 허용에 관한 연구는 점차 활발해 질 것이다.

시스템의 수행중 발생하는 결함을 극복하기 위한 결함 허용 시스템 설계시 다음과 같은 4가지 과정들은 필수적이다[2].

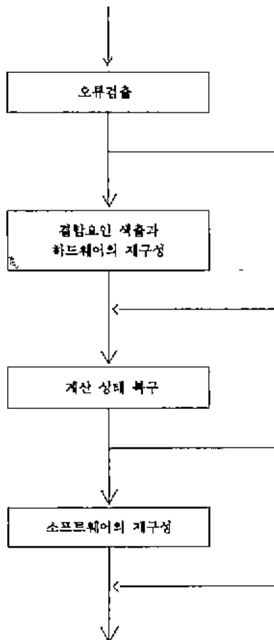
(1) 오류 검출(Error Detection)
(2) 결함 요인 색출과 하드웨어의 재구성(Fault Location and Hardware Reconfiguration)

* 종신회원

(3) 계산 상태 복구(Computation Recovery)
(4) 소프트웨어의 재구성(Software Reconfiguration)

그림 1은 시스템상에서 발생하는 결함 허용을 위한 4가지 과정을 설명한다. 소프트웨어 결함 허용의 중요성은 time-critical한 응용에서 잦은 결함들을 경험한 많은 연구가들과 소프트웨어 엔지니어들에 의해서 오래전부터 인식되어 왔다. 소프트웨어 결함 허용을 이룩하는 근본적인 매개체는 대체 모듈이다. 대부분의 결함은 모듈 설계시 생기는 residual한 오류에 의한 것이다. 소프트웨어 결함허용은 결함 모듈에서 발생된 결함때문에 훼손된 시스템을 대체 모듈로 대체함으로써 이룩된다. 소프트웨어 결함 허용을 위한 대체 모듈의 설계는 부수적인 노력, 즉 기능적으로 동일한 다른 모듈의 설계가 필요하다. 만일 모듈 설계시 residual한 오류에 의한 결함이 모듈 수행시 발생할 경우 복사된 동일한 모듈의 재수행은 시스템 결함 허용에 도움을 주지 못한다. 이런 이유때문에 기능적으로는 동일하나 설계가 다른 모듈을 대체 모듈로 사용한다.

본 논문에서는 소프트웨어 결함 허용기법의 개요를 설명하고 특히 후향오류복구(Backward Error Recovery)의 개념을 기초로 한 결함허용 기법인 복구블록(Recovery Block) 및 그 기법을 확장한 결함허용 기법들을 설명하고자 한다.



(그림 1) 결함허용을 위한 기본적인 과정

II. 기존의 소프트웨어 결험허용 기법

2.1 소프트웨어 적응검사(Acceptance Test)

소프트웨어 적응검사 기법은 시스템 250에서 사용되었다. 소프트웨어 적응검사 기법에서 사용되는 소프트웨어 오류 검사 루틴은 시스템의 전 과정을 통해 수행된다. 이 오류 검사 루틴들은 오류를 검출하여 복구 시스템에 보고함으로써 복구 시스템의 수행이 시작된다. 복구는 다음 세가지 재시작 프로시듀어에 의해 이루어진다.

(1) 프로세스 재시작

수행을 실패한 프로세스가 처리하지 못한 트랜잭션의 수행을 실패한 프로세스의 재 초기화 및 재 수행을 함으로써 처리하지 못한 트랜잭션의 수행을 이룬다. 재 수행시에는 정상적인 시스템의 기능을 사용한다.

(2) 국부적 재시작(warm start)

대부분의 처리하지 못한 트랜잭션의 수행을 무시하고 시스템이 갖고있는 중요한 redundant한 정보만을 가지고 재수행을 한다.

(3) 전면적 재시작(cold start)

모든 트랜잭션의 수행을 무시하고 실패한 프로세스의 전 시스템을 초기화하고 재수행을 시작한다. 이기법의 주된 장점은 소프트웨어 결함을 처리하는 데 있으나 그 수행시간이 길다는 단점이 있다.

2.2 일관성 검사(Consistency Checks)

일관성 검사는 사용되는 정보의 특성을 알고 있을 경우 이를 활용하여 그 정보에 오류가 존재하는지를 검사하는 기법이다. 예를 들면 프로그램의 한변수 var에 주어지는 값이 어떤 값 t 보다는 작다고 하면 변수 var의 값을 검사함에 의해 오류가 발생하였는지를 감지할 수 있다. 즉, 변수 var의 값이 t보다 크면 오류가 발생하였음을 알 수 있다.

2.3 능력검사(Capability Checks)

능력검사는 시스템이 기대했던 능력을 보유하고 있는지를 검사하기 위해 사용되는 기법이다. 예를 들면, 시스템내의 메모리가 전부 사용 가능한지를 검사하거나 (메모리 테스트) 다중 프로세스 시스템에서 모든 프로세서가 제대로 동작하는지를 검사하는 경우가 능력검사의 일종이다. 능력검사에는 여러 형태가 있는데 그 중 몇가지를 살펴보면 아래와 같다.

- 단순 메모리 테스트 : 프로세서가 단순히 특정 데이타를 메모리에 저장하였다가 읽어 그 값이 저장한 값과 동일한지를 검사하여 메모리의 이상 여부를 검사한다.
- ALU 테스트 : 시스템 내에 ALU가 하나 이상 있을 때 주기적으로 모든 ALU에 같은 데이타를 가지고 같은 명령어를 수행하게 하고 그 수행 결과를 비교함으로써 ALU의 이상 여부를 검사한다.
- 다중 프로세서 시스템에서의 프로세서 테스트 : 시스템 내에 존재하는 프로세서들간의 통신 능력을 검사하는 것으로 한 프로세서에서 다른 프로세서로 주기적으로 특정한 정보를 교환함으로써 이루어진다.

2.4 N-Version Programming

소프트웨어 결함은 하드웨어에 의해 여겨지는 것이 아니라 부정확한 설계나 코딩의 실수에 의해 발생한다. 그러므로 소프트웨어에서의 결함 검출 기법은 설계시 발생되는 결함을 검출해야 한다. 간단한 복제나 비교는 소프트웨어 모듈이 아주 동일하다면 소프트웨어 결함을 검출하지 못한다. 왜냐하면, 양 모듈에 설계상의 결함이 똑같이 있기 때문이다.

N-version programming의 기본 개념은 소프트웨어 모듈을 N번 설계하고 코드화하는 것이고 이 모듈들에 의해 생성된 N개의 결과를 비교하는 것이다. N개의 모듈 각각은 프로그래머 개개의 그룹에 의해 설계되고 코드화 된다. 각 그룹은 같은 기능을 수행하는 N개의 모듈을 각각과 같은 집합으로부터 설계한다. 이것은 독립적으로 N번의 설계를 수행함으로써 여러 그룹에서 같은 실수가 일어나지 않는 것을 기대한다. 그러므로 결함이 발생하면 모듈들에 의해 생성된 결과는 다를 것이다.

N-version programming 방법에서 가장 어려운 점은 두가지가 있다. 첫째, 소프트웨어 설계자와 프로그래머는 유사한 실수를 하는 경향이 있다. 그러므로 한 프로그램의 두 개의 완전한 독립 version이 동일한 결함을 갖지 않을 것이라는 보장을 못한다. 둘째, 프로그램의 N-version은 일반적인 명세로부터 개발되어 진다. 그래서 N-version 방법은 명세 실수의 검출을 하지 못한다.

N-version programming에 연관된 많은 문제들을 극복하는 것은 소프트웨어 설계자가 결함이 발생하지 않도록 시도하기 위해 확정된 설계 법칙과 방법을 사용하는 것이다. 이 방법은 결함 피하기로 알려져 있고, 신뢰할 수 있는 소프트웨어의 설계에서 매우 중요한 것이다. 만약 소프트웨어가 첫번째에 올바르게 설계되어 졌다면 소프트웨어를 위한 결함 허용 기법은 소용이 없게 된다.

III. 후향 오류 복구의 개념을 기초로 한 소프트웨어 결함 허용 기법

후향 오류 복구(Backward Error Recovery) 기법은 오류 발생시 보관된 복구점(Recovery oint)으로 돌아가서 재시도(Rollback and retry)하는 소프트웨어 결함 허용 기법이다[4,5,6]. 복구점은 그 당시의 시스템의 상태를 보관하는 시점으로, 결함 복구를 위한 시스템의 재수행은 결함 발생 시점에서 바로 전 복구점으로 돌아가서 실행된다.

3.1 복구블록(Recovery Block; RB) 기법

후향 오류 복구 대표적인 기법인 복구블록 기법은 경제적인 소프트웨어 결함 허용을 위하여 제시되었다. 복구블록 기법에서 사용된 추상적 언어구조는 오류 검출 및 복구행위 설계를 용이

```
ENSURE T
BY A1
ELSE BY A2
. . .
. . .
ELSE BY An
ELSE ERROR
```

(그림 2) 복구블록(Recovery Block)의 구조

하게 한다. 복구블록은 주 처리블록, 여러개의 대체 처리블록 및 적응검사(Acceptance Test; AT)로 구성되어 있고, 그림 2는 복구블록 기법의 구문구조를 설명한다. 복구블록내의 모든 처리블록은 다르게 설계되어야 하고 그 수행 결과는 동일하거나 수용할 수 있는 근접된 결과이어야 한다. 적응 검사는 처리블록이 수행 결과를 판정하는 논리식으로 그 결과의 수용여부를 판정한다. 적응검사의 논리식은 처리블록 수행 직후에 처리되며 처리블록의 결과가 성공적이면 복구블록으로부터 벗어나고, 처리블록의 수행 결과를 수용할 수 없다고 판정되면 계산 상태를 처리블록 수행전의 계산 상태로 복구하여 다음 대체 처리블록을 재 수행한다. 매 처리블록 수행 완료시에는 언제나 적응 검사를 행한다. 주 처리블록과 대체 처리블록의 설계는 기존 시스템의 설계방식을 사용하며, 언어 사용의 제약 즉, 전역변수, 자료구조, 또는 프로시듀어 사용상의 제약은 없다. Randell이 제안한 예는 아래와 같다 [7].

그는 n개의 수를 sorting하는 문제에 복구블록 기법을 적용시켰다. 주 처리블록으로는 Quick Sort를 사용했다. Quick Sort는 효율적이기는 하지만 완전히 검증된 알고리즘은 아니다. 반면에 대체 처리블록으로 사용된 Bubble sort는 덜 효율적이기는 하지만 완전히 검증된 알고리즘이다. 적응 검사로는 Sorting된 n개의 수를 사용하였다. 그가 사용한 복구블록의 구조는 다음과 같다.

```
Ensure 〈Acceptance Test〉
By 〈Quick Sort〉
Else By 〈Bubble Sort〉
Else error.
```

다른 예로는 계산상에서 나타나는 오류 검출에 대한 예이다.

```
Ensure 〈Acceptance Test〉
By ……
      Result=T*H/Z
Else By……
      Result=(T* (H/M)/Z)*M
      ……
Else error.
```

모든 처리블록에 있어 수행하기 전의 계산상태는 동일해야 한다. 대체 처리블록을 수행할 때에는 반드시 주 처리블록을 수행하기 직전의 상태와 동일한 조건이 형성되어야 한다. 동일한 조건을 제공하기 위해서는 주 처리블록 수행 직전의 계산 상태를 저장하여야 한다. 복구블록 기법에서는 non-local 변수들을 저장하는 방법으로 복구 캐쉬(Recovery Cache)방법이나 Backup 변수 방법을 주로 이용한다.

적응검사는 연속된 조건검사문(Condition Checking Statment)로 처리 블록 직후에 처리되며, 참, 거짓으로 그 결과를 나타내어 그 수용여부를 확인하는 것이다. 적응 검사는 처리블록 결과의 완전한 검증이 아니고 효율적으로 상태를 검사하는 것이다. 실제적으로 적응검사는 처리블록 직후에 non-local 변수의 결과값이 허용치 범위내에 존재하는지를 검사한다. 예외적인 오류 즉, arithmetic overflow가 발생하면 복구블록에서는 적응검사의 실패로 간주한다. 이런 경우에는 처리 부분의 나머지 부분을 건너 뛰고 다음 대체 블록을 수행한다.

다른 형태의 적응검사는 처리블록의 수행시간을 검사하는 것이다. 처리블록 수행 직전에 시간을 기록해 놓고 실제 처리시간이 사전에 정해놓은 허용시간을 초과할 경우 적응검사의 실패로 간주한다. 시간적응검사는 하드웨어로 구현된 Watchdog Timer의 지원으로 설계된다. 복구블록

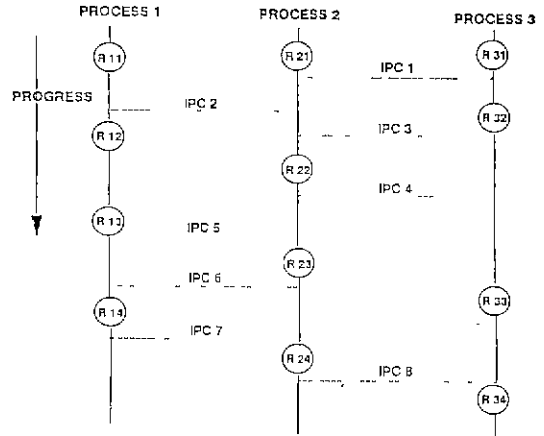
기법 사용의 주된 장점은 그 기법의 단순성에 있다. 성공적인 복구블록 기법을 구현하기 위해서는, 효율적인 대체블록의 설계와 효율적인 적응검사 설정이 필요하다. 프로그램 명세(program specification)나 Diversity programming의 연구와 실험에 의해서 복구 블록 기법은 소프트웨어 엔지니어들에게 결함 허용을 위한 실제적인 도구로 제공될 수 있을 것이다.

3.2 분산 환경에서의 후향 오류 복구 기법

분산처리 환경에서 상호 정보를 교환하는 프로세스들의 결함 허용은 interacting 프로세스들 간의 정보교환의 협력 및 동기화를 요구하고 있다.

3.2.1 도미노 효과(Domino Effect)

Interacting 프로세스들 간의 정보교환과 프로세스들의 복구점이 조화되지 않으면 도미노 효과(Domino Effect)가 일어난다. 그림 3은 interacting 프로세스 간의 부조화로 일어난 도미노 효과의 예이다. 그림은 세개의 프로세스, 그들의 복구점 및 정보 교환을 나타내고 화살표는 프로세스 수행의 진행을 표시한다. 만일 프로세스 3의 진행이 실패한 경우, 이 프로세스는 복구점 R34로 롤백하고 이 롤백은 복구점과 롤백한 시점 사이에 다른 프로세스와 정보 교환이 없기 때문에 다른 프로세스에 영향을 주지 않는다. 만일 프로세스 2의 실행이 실패한 경우, 이 프로세스는 복구점 R24로 롤백하고 프로세스 3과의 정보교환 IPC8 때문에 프로세스 3을 롤백하게 한다. 프로세스 2의 롤백이 프로세스 3의 롤백의 원인이 되었으며 프로세스간의 정보교환과 복구점의 부조화로 롤백이 다른 프로세스로 전달되었다. 그러나 프러세스 1의 수행이 실패한 경우는 세 프로세스 모두가 시작점 R11, R21 및 R31으로 롤백하게 된다. 이와 같이 프로세스 간의 정보교환과 복구점의 부조화로 인하여 프로세스 사이에 계속적인 롤백 전달의 사태가 일어나는 것을 도미노 효과라고 한다. 그러므로 interacting 프로세스들은 그들의 복구점 설정을 다른 프로세스들과 조화를 이루며 설정해야 한다.



(그림 3) 도미노 효과

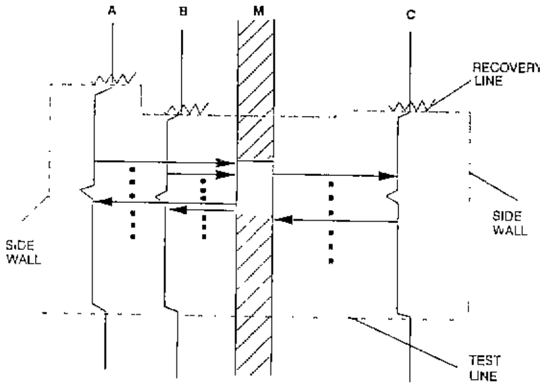
3.2.2 Conversation

Randell은 그가 제안한 결함 허용 기법 Conversation에서 복구선(Recovery line)과 검사선(Test line)을 사용하여 interacting 프로세스 간의 2차원적인 복구 행위를 제시하였다[7]. 복구선은 interacting 프로세스들의 복구점의 집합이며 검사선은 interacting 프로세스들의 적응검사의 집합으로 정의된다. 그림 4는 Conversation의 기본구조를 설명한다.

Conversation의 기본구조는 복구선, 검사선 및 Conversation에 참가하고 있지 않는 프로세스들과의 경계를 나타내는 Side Wall로 되어 있다. Conversation에서는 한 프로세스라도 적응검사를 실패하면 Conversation에 참여한 모든 프로세스가 동시에 각자의 복구점으로 롤백하여 각자의 대체 블록을 수행함으로써 도미노 효과를 방지하고 있다. Conversation에서는 참여한 모든 프로세스가 적응검사를 성공한 경우에만 패스한 것으로 간주한다. Conversation 역시 복구블록 기법이 지닌 문제점인 대체 블록 설계와 적응검사 설계에 따른 어려움을 갖고 있다. 또한 Conversation에 참여한 프로세스들 간의 동기화가 이 기법을 실시간 분산 처리 시스템에 구현하는데 문제점이 된다.

3.2.3 Wood의 기법

Wood는 분산된 환경에서의 후향 오류 복구 기법을 제시하였다[8]. 이는 분산 시스템에서의



(그림 4) Conversation

하나 또는 그 이상의 프로세스가 실패 후 복구된 경우에도 일관성 있는 시스템의 상태를 보존하기 위하여 불필요한 복구점을 제거하는 것이다. 그는 시스템 수행의 과정을 intra-process와 inter-process 정보교환의 흐름을 기본으로 하였으며 분산 시스템에서 서로 독자적으로 설정된 복구점들의 관계를 정의하였다.

3.2.4 Russell의 기법

Russell은 도미노 효과가 발생하지 않는 메시지 리스트를 기본으로 하는 시스템 구조를 제시하였다[9]. 그가 제시한 domino-free 시스템은 프로세스가 설정하는 복구점의 수에 제한을 두는 것이다. 즉 특정 순차적인 메시지 리스트의 수행 후에만 복구점을 설치하는 결함 허용 시스템을 제시하였다.

3.2.5 Kant와 Silberschatz의 기법

Kant와 Silberschatz는 도미노 효과를 방지하기 위하여 시스템에 의해 설정되는 복구점과 부수적으로 사용자가 정의하는 복구점의 조건을 제시하였다[10]. 그들이 제시한 시스템의 구조는 프로세스들의 정보교환을 모니터를 통하여 하는 것이다. 프로세스들의 모니터 read/write 수행 과정은 반드시 복구 블록 안에서만 이루어지게 하였다. 그리고 처리 블록 내에서의 모니터 수행의 수와 타입의 제한을 둠으로써 도미노 효과를 방지하였다. 프로세스는 처리 블록 내에서 많아야 한번의 write 수행을 하나 read 수행의 수는 제한을 두지 않았다. 또한 프로세스는 처리블록 내에서는 read 수행이 write 수행을 선행할 수 없다는 제약이 있다. 그들은 후향 오류 복구 문제점의 해결책은 제시하였으나, interacting 프로세스들이 그들이 제안하는 구조의 모니터를 사용해야 되므로 대부분의 분산처리 시스템에는 적합하지 않다.

3.2.6 Briatico의 기법

Briatico 역시 도미노 효과가 없는 시스템의 구조를 제시하였다[11]. 그가 제시한 것은 분산처리 시스템에서 설정된 복구점들이 각각 그 복구점에 관련된 하나의 복구선에 연결되어 있어 결함 발생시 모든 interacting 프로세스들이 동시에 하나의 복구선으로 롤백을 함으로써 interacting 프로세스들의 롤백에 일관성을 제시했다. 그러나 그가 제시한 결함 허용 시스템은 롤백에 의한 과부하 때문에 특히 실시간 분산 처리 시스템에는 적용할 수 없다.

3.3 분산 복구블록(Distributed Recovery Block; DRB) 기법

분산 복구블록 기법은 분산처리 개념과 복구블록의 개념을 기초로 하고 있다. Time-critical 응용에서 사용되는 컴퓨터 시스템은 하드웨어와 소프트웨어 요소들의 고장시 최소한의 복구 시간을 요구하고 있다. 실시간 컴퓨터 시스템에서 소프트웨어의 결함과 하드웨어의 결함을 구분하는 일은 간단하지 않다. 그러므로 모든 가능한 결함을 처리하는 것은 시스템의 크기만 증가시키는 요인이 된다. 비록 모든 상이한 결함을 치유할 수 있다고 하더라도 시스템의 크기 및 복잡도의 증가와 계산 수행의 오버헤드는 시스템을 비효율적이고 작동 불가능하게 만든다. 분산 복구블록은 하드웨어의 결함과 소프트웨어의 결함을 획일적으로 처리할 수 있는 장점과 대체 처리 블록을 동시에 수행하므로 복구 수행 속도가 짧은 장점을 갖고 있기 때문에 time-critical한 응용분야에 사용될 수 있다.

3.3.1 분산 복구블록 기법의 구성

분산 복구블록 기법은 단지 두개의 처리블록 즉 주처리 블록과 한개의 대체 처리블록과 적응검사로 구성되어 있다. 각각의 처리블록의 수행시간에 대한 제약은 분산 복구블록 기법의 실시간 시스템을 위한 확장으로 생각할 수 있다. 처리블록 수행시간의 검사는 watchdog timer를 사용함으로써 처리 블록의 수행 허용시간의 경과 여부를 결정한다. 처리 블록 수행시간이 허용시간을 초과할 경우에는 적응검사의 실패로 간주한다. 분산 복구블록 기법에서의 적응검사는 계산 결과를 판정하는 적응검사와 수행시간의 경과 여부를 판정하는 적응검사의 복합으로 구현된다.

실시간 분산처리 시스템에 적용될 분산 복구블록 기법은 아래와 같은 시스템의 특성을 가지고 있어야 한다.

(1) 컴퓨터 시스템은 여러개의 계산국(Computing Station)으로 되어 있고 각 계산국은 단지 한개의 복구블록으로 구성된다.

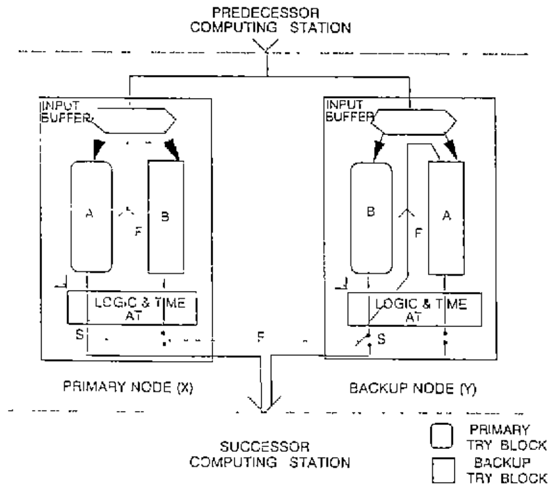
(2) 한 계산국에서 산출된 계산 결과는 다른 계산국의 입력이 된다.

(3) 계산국은 하나 또는 그 이상의 계산 노드로 구성된다.

(4) 다중 계산 노드는 여분의 노드로 사용된다.

그림 5는 분산 복구블록 기법을 두개의 노드에 적용한 경우이다. 주노드와 대체노드는 두개의 처리블록과 계산 결과와 수행시간의 수용 여부를 판정하는 적응검사로 구성된다. 각각의 노드에 있는 두개의 처리블록의 역할은 각각 다르게 할당되어 있다. 주노드에는(그림 5의 X) 처리블록 A를 주 처리블록으로 사용하고, 대체노드에서는 (그림 5의 Y) 처리블록 B를 주 처리 블록으로 사용한다. 결함이 검출되기 전에는 두 노드가 같은 입력 데이타로 서로 다른 처리블록을 수행한다. (노드 X에 찰리블록 A와 노드 Y에 처리블록 B). 두 처리 블록의 수행결과는 적응검사를 사용하여 성공 여부를 판정한다. 두 노드가 동시에 각각의 처리블록을 수행하고, 대체노드에 시간 적응검사만이 두 노드의 수행시간을 검사한다.

결함이 없는 상태에서는 두 노드가 같은 입력 데이타를 앞의 노드에서 받아 각각의 주 처리 블록만을 수행하고, 주노드의 수행 결과만이 다음 노드로 전달된다. 만일 주노드의 수행 결과가



(그림 5) 분산 복구블록 기법의 구조

적응검사에 의해 거짓으로 밝혀지고, 대체노드의 수행 결과가 참일 경우에는 대체노드가 주노드의 역할을 담당하게 된다. 만일 대체노드가 적응검사에 실패한 경우에는 주노드는 대체노드의 결과에 관계없이 처리를 계속한다. 어떤 노드가 실패한 경우, 그 노드의 데이타베이스를 보전하기 위하여 다음과 같은 조치를 취해야 한다. 적응검사를 실패한 노드는, 두 노드의 데이타 베이스를 동등하게 유지하기 위하여 대체 처리블록을 수행하여 데이타베이스를 갱신한다. 주노드가 처리블록의 수행을 실패했을 경우에 대체노드가 그 역할을 담당하기 위해서는 대체노드는 주노드와 동등한 데이타베이스와 계산 상태를 유지해야 한다.

대체노드는 주노드가 보고하는 수행 결과와 상태를 매번 검사해야 한다. 만일 주노드가 처리블록 수행 결과의 실패를 보고하면 대체 노든 즉시 그 실패를 인지하고 주노드의 역할을 담당한다. 또 주노드의 수행시간이 미리 정해진 허용시간을 초과할 때에는 대체노드는 주노드의 수행결과를 실패로 간주하고 주노드의 역할을 담당한다.

두 노드 사이의 적응검사 수행은 비동기적으로 하기 때문에 분산 복구블록 기법에서 노드사이의 동기화에 필요한 오버헤드는 고려의 대상이 되지 않는다. 분산 복구블록 기법은 중복된 처리 블

록의 수행이 동시에 일어나기 때문에 복구시간이 극히 적다. 그러나 이 기법의 구현에 대한 연구는 깊이 이루어지지 않았다. 경제적이고 효율적인 분산 복구블록 기법의 구현을 위하여 다양한 처리블록의 설계 및 노드의 재구성이 고찰되어야 한다.

IV. 결 론

본 논문에서는 소프트웨어 결함 허용 기법에 대하여 기술하였다. 기존의 시스템에 사용되고 있는 결함 허용기법과 현재 이 분야에서 연구되고 있는 소프트웨어 결함 허용 기법들의 소개를 하였다. 컴퓨터 시스템 소프트웨어의 비대화와 복잡화로 시스템 작동시 발생하는 결함의 증가는 당연하다. 본 논문에서 소프트웨어 결함 허용기법중 많은 연구가 진행중인 후향 오류 복구 개념을 기초로한 결함 허용 기법들을 소개하였다.

참 고 문 헌

1. Avizienis, A. and kelly, J. P. J., "Fault tolerance by Design Diversity: Concepts and Experiments", IEEE Computer, August 1984, pp. 67~80.
2. Kim, K. H., "Error Detection, Reconfiguration and Recovery in Distributed Processing Systems", Proc. IEEE 1st Conf. on Distributed Computing Systems, Oct. 1979, pp. 284~295.
3. Davis, C. G. and Couch, R. L., "Ballistic Missile Defence: A Supercomputer Challenge", IEEE Computer, Nov. 1980, pp. 37~46.
4. Hecht, H., "Fault-Tolernat Software for Real-time Applications", Computing Surveys, Dec. 1976, pp. 391~407.
5. Heu, S., "Experimental Validation of Distributed Recovery Block", '91 Pacific Rim International Symposium on Fault Tolerant Systems, Sep. 1991.
6. Kim, K. H., "Issues in Design of Temporary Blockout Handling Capabilities into Tightly Coupled Computer Networks", Software for Strategic Systems Conf. Oct. 1988.
7. Randell, B., "System Structure for Software fault tolerance", IEEE Trans. on Software Engr., June 1975, pp. 220~232.
8. Wood, W. G., "A Decentralized Recovery Control Protocol", Digest of FTCS-11, 1981, pp. 157~164.
9. Russell, D. L., "State Restoration in System of Communicating Processes", IEEE Trans. on Software Engr. Vol. SE-6, No. 2, Mar. 1980, pp. 183~194.
10. Kant, K. and Silberschatz, A., "Error Recovery in Concurrent Processes", Proc. COMPSAC, Oct. 1980, pp. 608~614.
11. Briatico, d., *et al.*, "A distributed Domino-Effect Free Recovery Algorithm", Proc. 4th Symp. on Reliablity in Distributed Software and Database Systems, Oct. 1984.

허 신

1973 서울대학교 전기공학과 졸업
1979 Univ. of Southern California 전자계산학 석사
1986 Univ. of South Florida 전자계산학 박사
1986 ~1988 The Catholic University of America 조교수
1988 ~현재 한양대학교 전자계산학과 조교수
관심 분야 : 분산처리 시스템, 결함 허용 컴퓨터 시스템, 실시간 운영체세