결함 검출 기법들의 성능 평가를 위한 테스트베드의 설계*

윤영원° 이효순 신현식
서울대학교 컴퓨터공학부
{forever, fanta, shinhs}@cselab.snu.ac.kr

Design of Testbed for Performance Evaluation of Fault Detection Techniques

Young-Won Yun° Hyosoon Lee Heonshik Shin
School of Computer Science and Engineering, Seoul University

요 약

결함의 검출은 결함 허용 시스템의 결함 허용성과 신뢰도 분석에 있어서 기초가 된다. 결함 검출 기법들은 각기 다른 특성을 가지고 있어 결함의 종류에 따라 다른 검출 능력을 가지기 때문에 효율적으로 시스템의 신뢰도를 향상시키기 위해서는 결함의 종류에 따라 적절한 기법들을 선별하여 적용해야 할 필요가 있다. 하지만 기존의 연구에서는 결함 검출 기법들에 대해 비교 검토에 대한 연구가 미흡하다. 따라서 결함의 종류에 따른 결함 검출 기법들의 성능을 평가하기 위한 테스트베드가 요구된다. 본 논문에서는 결함 검출을 위해 사용되고 있는 기법들의 종류를 분류하고 특성을 서술한다. 그리고, 리눅스 환경에서 소프트웨어로 구현된 결함 삽입 도구를 이용하여 각 결함 검출 기법들의 성능을 비교하기 위한 테스트베드를 설계한다.

1. 서론

유도 무기 미사일이나 원자력 발전소와 같은 중요한 작업을 수행하는 시스템에서는 시스템에 고장이 발생했을 때 정상적으로 동작하거나 또는 외부로 고장의 효과가 나타나지 않도록 시스템을 구성해야 한다. 이러한 결함 허용 시스템은 개발 시간과 비용의 제한으로 인해 점점 더 COTS(Commercial Off-The-Shelf) 구성요소를 기본 설계 블럭(building block)으로 사용하여 설계되는 추세에 있다. 그런데, 이런 COTS 구성요소들은 대개 결함 허용성을 위한 기능을 가지고 있지 않거나 미흡하다. 따라서, 소프트웨어 모듈을 통하여 COTS 시스템의 고장이나 결함을 검출하고 대처하는 방법들이 사용되고 있다[5]. 결함의 검출은 시스템의 결함 허용성과 신뢰도 분석에 있어서 시작점(baseline) 역할을 하는 기능이다. 결함 검출 기법들은 각기 다른 특성을 가지고 있어 결함의 종류에 따라 다른 검출 능력을 가지기 때문에 효율적으로 시스템의 신뢰도를 향상시키기 위해서는 결함의 종류에 따라 적절한 기법들을 선별하여 적용해야 할 필요가 있다[1,2]. 하지만 기존의 연구에서는 결함 검출 기법들에 대해 비교 검토에 대한 연구가 미흡하다. 따라서 결함의 종류에 따른 결함 검출 기법들의 성능을 평가하기 위한 테스트베드가 요구된다. 테스트베드에는 사용자의 요구에 따라서 새로운 검출 기법을 추가하거나 재정의할 수 있어야 하므로 개방성과 확장성이 요구된다. 결함 검출 기법의 성능은 결함 삽입 도구를 통하여 인위적으로 결함을 시스템에 삽입시킨 후 결함 검출 기법에 의해 검출되는지의 여부를 통하여 평가될 수 있다. 따라서, 본 논문에서는 이러한 결함 검출 기법의 성능을 평가하기 위한 테스트베드를 제안하고 설계한다. 본 논문은 다음과 같이 구성된다. 먼저, 2장에서는 기존 시스템에서 사용되고 있는 결함 검출 기법들을 분류하고 특성을 설명한다. 3장에서는 이전 연구에서 구현된 결함 삽입 도구 SFIDA(A Software Implemented Fault Injection Tool for Distributed Dependable Applications)[4]에 대해서 소개하고 삽입 가능한 결함 모델을 설명한다. 그리고 SFIDA를 이용하여 결함 검출 기법의 성능 평가를 위한 테스트베드를 설계하고, 그 구조에 대하여 설명한다. 마지막으로 4장에서는 본 논문을 요약한다.

* 본 연구는 국방과학연구소 및 서울대 자동제어특화연구센터의 연구비 지원에 의한 연구결과입니다.

2. 결함 검출 기법

이 장에서는 단일 혹은 분산 응용에서 결함을 검출하는데 자주 사용되는 결함 검출 기법들을 분류하고 장단점을 기술한다. 결함을 검출하는 기법들을 분류해보면 다음과 같다[2,3].

- 하드웨어 기반 결함 검출 기법
  - 자원과 시간의 중복을 이용한 복제와 비교
  - 영속적 결함 검출을 위한 스케줄 된 오프라인 테스팅, 또는 다중 프로세서 구성 요소의 표준 회로 수준 혹은 모듈 수준 코딩 그리고 자기 검사 기법
- 소프트웨어 기반 결함 검출 기법
  - 진단과 코딩 기법
  - 알고리즘 기반 방법
  - 프로그램 감시 기법

하드웨어 기반 결함 검출 기법은 시스템에 의존적이기 때문에 다른 시스템으로의 이식성과 확장성이 부족하다. 하지만 소프트웨어 기반 결함 검출 기법은 여러 시스템에 이식할 수 있고 새로운 결함 검출 기법을 테스트베드에 쉽게 추가할 수 있는 장점이 있다. 본 논문에서는 소프트웨어 기반 결함 검출 기법에 초점을 둔다. 소프트웨어 기반 결함 검출 기법들의 장단점은 다음과 같다.

- 진단과 코딩 기법을 이용한 결함 검출

진단 프로그램은 프로세서, 메모리, 그리고 상호연결 네트웍 요소들의 하드웨어 고장을 감지하는데 사용된다. 진단 체크는 영속적 결함은 검출 할 수 있지만 진단 중에 결함이 재생성 될 수 없기 때문에 일시적이거나 간헐적인 결함을 검출하는 데에는 사용하지 못한다. 코딩기법은 버스, 메모리, 레지스터의 결함을 적은 비용으로 검출할 수 있다.

- 알고리즘 기반 방법을 이용한 결함 검출

알고리즘 기반 결함 허용과 검출은 처음에 배열 프로세서와 시스톨릭 배열에서 응용을 위해 제안되었다. 워드에서 비트에 영향을 미치는 에러를 방지하기 위해 작업 수준에서 널리 알려진 데이터 암호화가 수행된다. 결함이 있는 모듈은 동작하는 워드의 모든 비트에 영향을 미칠 수 있기 때문에, 더 높은 수준에서 데이터를 암호화할 필요가 있다. 이 전의 알고리즘은 이 암호화된 데이터에 동작하고 암호화된 출력 데이터를 구하기 위해 다시 설계되어야 한다.

- 프로그램 감시를 이용한 결함 검출

프로그램이 수행될 때, 프로그램 감시 기법은 프로그램에서 발생하는 결함들을 검출한다. 일반적으로 결함을 검출할 수 있는 방법은 다중 버전 프로그램을 작성하는 것이다. 예를 들어 다른 버전의 프로그램들 을 병렬적으로 수행시키고, 각 버전들의 상태를 서로 비교하여 결함을 검출할 수 있다.

표 1. 결함 모델

자원 결함 종류 결함 위치	
레지스터	
주소라인 결함	
버퍼 결함	
제어 흐름 결함	
메모리 배치 결함	
레지스터	
- 베이스 주소	
- 데이터 버퍼	
제어 흐름	
- 프로그램 카운터	
	메모리
	단일 비트
	여러 비트
비트 값 변경	
- 한 비트	
- 한 바이트	
	디바이스
	장치 고장
	데이터 결함
파일 열기	
파일로부터 읽기	
파일에 쓰기	

3. 테스트베드 설계

3.1 결함 삽입 도구

이 장에서는 결함 검출 기법들의 성능을 평가하기 위해 시스템에 결함을 인위적으로 삽입하는 방법에 대해 설명한다. 결함은 이전 연구에서 구현된 SFIDA라는 결함 삽입 도구에 의해서 삽입된다[4]. 이 결함 삽입 도구는 이더넷으로 연결되고 리눅스 6.0 버전이 설치된 펜티엄 350MHz 시스템에서 C 언어로 구현되었다. SFIDA는 프로그램의 실행 환경에서 하드웨어 결함에 의해 발생할 수 있는 에러 상태를 에뮬레이트함으로써 하드웨어의 일시적결함과 영속적 결함을 삽입할 수 있다. 표 1은 SFIDA가 지원하는 결함 종류를 요약한 것이다.

- **프로세서** : 프로세서에서의 에러는 ALU나 내부 레지스터와 같은 유닛에서 발생하는 결함을 유발시킨다. 버퍼 레지스터, 베이스 레지스터, 프로그램 카운터가 그 대상 자원이다. SFIDA는 일시적인 결함과 영속적인 결함 삽입을 모두 제공하고 있지만, 이러한 자원에서의 영속적인 결함을 삽입하는 데에는 너무 많은 시간이 소요된다. 특히, 버퍼 레지스터는 임시 데이터를 저장하기 위해서 사용되는데 응용 프로그램의 실행 초기부터 종료시점까지 계속 사용될 수 있다. 그런데 이러한 레지스터에 영속적인 결함을 삽입하려 하면 버퍼 레지스터에 접근되는 모든 명령어에 트랩이 발생하게 되므로 상당한 시간적 오버헤드가 발생한다.

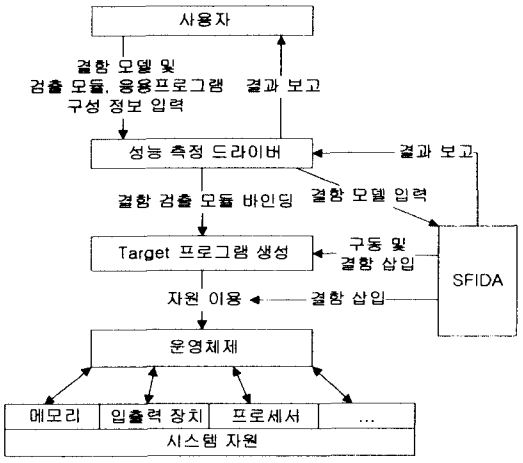

그림 1. 테스트베드의 구조와 동작

하여 프로그램이 사용하는 세그먼트 영역 공간에서 임의로 선택될 수 있다.

- **디바이스** : SFIDA는 입출력과 관계된 상위레벨의 에러 상태를 인위적으로 생성함으로써 통신상의 결함을 포함한 입출력 장치의 결함을 다룰 수 있다. 이는 파일을 열거나 파일에 데이터를 쓸 때, 그리고 파일에서 데이터를 읽어올 때에 결함을 삽입하는 방식으로 수행된다.

3.2 결함 검출 기법의 성능 측정을 위한 테스트베드의 설계

성능을 평가할 대상 프로그램(target program)은 실행중에 표 1에 나열된 결함을 모두 포함하거나 특정 결함만을 삽입된 상태에서 구동된다. 대상 프로그램은 각각의 결함 종류에 따라 다르게 구성할 수 있다. 결함 검출 기법의 성능 측정을 위한 테스트베드의 구조와 동작은 그림 1과 같다. 시스템의 구성은 사용자의 성능 평가를 위한 입력 정보와 결함 삽입을 담당하는 SFIDA 모듈, 삽입된 결함을 검출하기 위한 결함 검출 모듈로 이루어진다. 메모리와 같은 시스템 자원, 그리고 마지막으로 시스템 자원을 관리하는 운영체제가 있다. 사용자는 평가 하고자 하는 결함 검출 모듈과 이 모듈과 연결되어 실행될 대상 프로그램, 그리고 결함 모델에 대한 정보를 성능평가 드라이버에게 전달한다. 성능평가 드라이버는 입력정보를 바탕으로 대상프로그램을 생성하고 SFIDA에 결함 모델과 함께 전달한다. 결함 삽입 도구에 전달되는 결함에 대한 정보는 결함 종류, 결함 발생 위치, 결함 발생 시간, 결함의 지속성 등이다. 대상 프로그램과 연결되는 결함 검출 모듈들은 2장에서 설명한 소프트웨어 기반 결함 검출 기법들을 구현한 것이지만, 새로운 검출 기법의 성능 평가를 위해 정해진 API를 통하여 구동되는 라이브러리를 설정할 수 있다. SFIDA는 입력 받은 결함 모델을 바탕으로 대상 프로그램에 결함을 삽입한다. SFIDA는 결함 검출 모듈에 의해 성공적으로 결함이 극복되었는지를 검사하고 그 결과를 성능평가 드라이버에게 전달하여 사용자에 보고하도록 한다. 정확한 성능 평가에 대한 결과를 얻기 위해 결함을 삽입하고 결함을 검출하는 과정을 원하는 회수만큼 반복할 수 있다. 결함 검출 기법의 성능은 결함 검출 실패와 성공의 비율로 나타내는데, 여기에서 성공이란 삽입된 결함에 의해 시스템 고장이 발생했을 경우로 정의된다. 왜냐하면 결함이 대상프로그램의 실행시 실제 사용되지 않는 메모리 공간이나 디바이스에서 발생할 수 도 있기 때문이며, 결함 검출 모듈은 출력을 생성하는 데에 영향을 미치는 정보에 대해서 검출 작업을 수행하기 때문이다. 현재 테스트베드는 결함 검출 기법들의 결함 종류에 따른 성능을 측정함으로써, 결함 허용 시스템의 설계와 신뢰도 평가에 중요한 자료를 제공할 수 있도록 구현중이다.

4. 결론

결함의 검출은 시스템의 결함 허용성과 신뢰도 분석에 있어서 필수 불가결한 요소이다. 결함 검출 기법들은 각기 다른 특성을 가지고 있어 결함의 종류에 따라 다른 검출 능력을 가지기 때문에 효율적으로 시스템의 신뢰도를 향상시키기 위해서는 결함의 종류에 따라 적절한 기법들을 선별하여 적용해야 할 필요가 있다. 본 논문에서는 결함 삽입 도구를 이용하여 결함 검출 기법의 성능 평가를 자동화하기 위한 테스트베드를 설계하였다. 결함 삽입 도구는 이미 구현되었으며, 현재는 테스트베드를 구현하면서 결함 삽입 도구와 통합작업을 진행중이다.

5. 참고문헌

[1] Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi, and keith Whisnant,"Chameleon: A Software Infrastructure for Adaptive Fault Tolerance", IEEE Transactions on Parallel and Distributed System, June 1999.

[2] D. K. Pradhan,*Fault Tolerant Computer System Design*, Prentice Hall PTR, 1996.

[3] D. P. Siewiorek and R. S. Swarz,*Reliable Computer Systems*, Digital Equipment Corporation, 1992.

[4] Hyosoon Lee, Youngshik Song and Heonshik Shin,"SFIDA: A Software Implemented Fault Injection Tool for Distributed Dependable Applications," The Fourth International conference on High Performance Computing in Asia-Pacific Region, China, May 2000.

[5] A. Avizienis,"Toward Systematic Design of Fault-Tolerant Systems," IEEE Computer, April 1997, pp.51-58.