결함모형기반 고속결함검출기법

김유호, 김성수
아주대학교 정보통신전문대학원
e-mail:{daybreak001, sskim}@ajou.ac.kr

A Fast Fault Detection Method Based on Fault Model

You-Ho Kim, Sungsoo Kim
Graduate School of Information and Communication,
Ajou University

요 약

결함허용시스템을 위한 연구는 유비쿼터스 컴퓨팅에 대한 연구와 더불어 가속화되고 있다. 결함에 대한 검출시간과 유지보수시간을 줄이는 연구가 중요시되고 있는 시점에서, 본 연구는 시스템의 평균유지보수시간을 줄이고자 결함모형을 기반으로 한 고속결함검출기법을 제안한다. 본 연구의 목표는 기존의 일반적인 시스템에서 결함 모형을 적용한 결과를 소개하여, 이 결과에 소립단위재시작(Microreboot)을 결합시킨 결과를 분석하였다. 그 분석 결과 결함검출시간을 단축시킬 수 있었으며, 이를 통해 기존의 연구보다 높은 가용성을 보였다.

1. 서론

결함허용 시스템에 대한 연구는 유비쿼터스 컴퓨팅에 대한 연구가 가속화되는 현 상황에서 더욱 중요시 되고 있다. 이는 유비쿼터스 컴퓨팅의 다양한 환경에 대한 철저한 시스템 설계 및 구현에도 불구하고 예기치 못한 결함의 발생에 대한 모든 대비가 완벽하게 되는 것은 불가능하기 때문이다[1]. 따라서, 결함을 견딜 수 있는 시스템, 즉 고가용성을 보장하기 위한 연구가 필연적으로 대두된다.

가용성은 평균고장시간(Mean Time To Failure :MTTF)에 비례하며 평균유지보수시간(Mean Time To Repair: MTTR)에 반비례한다. 고가용성을 위해서는 0.99999의 가용성, 즉 일년에 5분 내외 고장시간을 보장하는 가용성을 제공해야하고 이를 달성하기 위한 방안으로 평균고장시간의 최소화 혹은 평균유지보수시간의 극대화가 요구된다.

본 연구는 정보통신부 21세기 프론티어연구개발사업의 일환으로 추진되고 있는 유비쿼터스컴퓨팅및네트워크원천기반기술개발사업의 지원에 의한 것임

평균유지보수시간은 결함발생에 대한 검출 및 검출된 결함을 처리하기 위한 보수시간이 요구된다. 유지보수를 위한 방법으로는 시스템 재시작이 대표적인데, 이는 적은 비용에 비해 높은 가용성을 가질 수 있기 때문이다. 관련된 최신 연구로, 소립단위재시작(Microreboot)에 관한 연구가 이루어지고 있다.[2, 3]. 본 논문에서는 결함처리를 위해 소립단위재시작을 사용하는 시스템 환경위에 고속결함검출을 위한 결함모형을 적용함으로서 평균유지보수시간을 최소화 하고, 이를 통해 99.99999%의 가용성을 보장하는 결함허용(fault-tolerance)시스템을 구축하고자 한다.

2. 관련 연구

평균유지보수시간을 줄이기 위한 대표적인 연구는 SWIG(SoftWare Infrastructure Group)에서 이루어지는 소립단위재시작을 들 수 있다. 소립단위재시작은 인터넷과 관련한 응용 서비스가 증가됨에 따라 나타난 큰 규모의 시스템의 결함을 대상으로 하고 있으며, 이에 따르면 예상하지 못한 결함이 발생되

었을 때에는 시스템 전체를 재시작하는 유지보수 방법은 낭비라고 언급하였다[3]. 따라서 이를 개선하기 위해 시스템을 컴포넌트 단위로 나누어 결함발생에 따라 각 컴포넌트 별로 재시작하는 모델을 제안하였다. 하위 컴포넌트에 결함이 발생하더라도 전체 시스템을 재시작할 필요가 없기 때문에 성능면에서 개선된 결과를 보였다. 그러나 결함이 발생한 수준을 구별하지 않고 각 단계별로 재시작을 하기 때문에, 최악의 경우에는 처음부터 전체 시스템 단위의 재시작을 하는 경우보다 좋지 못한 성능을 보일 수 있다. 앞에서 언급한 문제를 해결하기 위하여 소립단위재시작에 결함모형을 적용할 수 있으며, 이를 통해 평균유지보수시간을 단축시킬 수 있다. 결함모형에 관련된 기존 연구로는 결함모형실험을 통해 클러스터 기반 인터넷 서비스의 가용성을 높이는 연구가 있다[4]. 이에 따르면, 각 노드들의 상하 및 수평관계를 관리하는 Group Membership과 멤버들의 상태를 체크하는 Queue Monitering을 결합하여 결함모형에 적용하고 있다. 본 논문에서는 이를 이용하여 컴포넌트들의 모니터링을 소립단위재시작에 적용하고 성능평가를 수행하였다.

3. 결함모형

본 논문에서 결함발생단계에 따른 재시작의 종류에는 1단계부터 3단계까지가 있다고 가정한다. 각 단계에서 1단계는 단일 컴포넌트 재시작을 의미하며, 2단계는 컴포넌트 컨테이너 수준의 재시작, 3단계는 전체 시스템 수준의 재시작을 의미한다. 표1에서는 각 단계에서 발생 가능한 주요 결함 및 결함유지보수의 재시작시간을 보여준다. 재시작시간은 기존 연구의 실험결과를 따르며, 각 결함에 대한 재시작시간들의 평균값을 구하였다.

표1 결함의 재시작 수준과 재시작 시간

(시간:msec)

단계	주요 결함	재시작시간
1	Application memory leak	533
2	Corrupt data inside session state	1,027
3	Bit flips in process memory	19,083

표1에서 보듯이 3단계 재시작시간은 1단계 재시작시간에 비해 20배 정도의 과부하를 요구한다. 기존 연구는 1단계부터 3단계까지 순차적으로 재시작을 수행하며 그 결과 결함이 해결되었는지를 확인하기 때문에 만약 3단계재시작이 요구되는 결함이 발생한 경우에는 추가적인 시간(1,560ms)을 낭비되게 된다. 또한, 이러한 추가적인 시간외에 각 단계 재시작에 따른 초기화 시간이 요구되기 때문에 또 다른 추가시간이 요구된다. 만일 결함 발생의 단계를 고속으로 검출할 수 있다면 바로 3단계 재시작을 수행할 수 있기 때문에 이러한 추가적인 시간을 줄일 수 있다. 따라서, 이와 같은 결함검출을 위한 결함모델 기반의 재시작이 순차적인 재시작보다 감소된 평균유지보수시간을 나타낼 수 있다.

4. 결함모형 분석 결과

본 논문에서는 결함 발생을 모니터링하기 위한 그림1과 같은 시스템 환경을 가정하였다. 결함 발생시 3단계 재시작이 필요한 상위 컴포넌트와 그와 연관된 하위 컴포넌트를 설정하고, 이를 관리하는 그룹관리자(Group Manager)를 두었다. 결함 발생으로 처리량(Throughput)에 이상이 발생하면 그룹 관리자가 이를 감지하고 응용 프로그램 관리자(Application Manager)에게 상태정보를 전송한다. 응용 프로그램 관리자는 모니터(Monitor)와 결함모형을 이용하여, 어떠한 단계의 유지보수가 필요한지 판단한다. 이러한 환경은 결함모형이 모든 결함에 대해 완벽히 정의되어 있다는 가정에서 시작된다. 이에 관한 구체적인 방안은 다음장에서 설명한다.

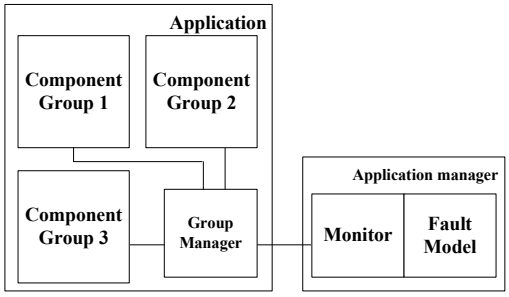

그림1 Application 모니터링 환경

그룹 관리자는 Heartbeat을 사용하여 컴포넌트들을 관리하므로, 처리량의 변동에 따른 결함상황알림시간은 0에 가깝다. 응용 프로그램 관리자와의 통신은 결함 발생시에만 연결되므로 결함모형 프로시저가 시작에서 종료까지 소요되는 시간이 전체결함검출시간과 거의 동일하다. 따라서 결과적으로 소요되는 시간도 기존시간(19,083 + 1,560 + 재시작별초기화시간ms)에 비해 1,000ms이상 빠를 것으로 예상된다.

표2에서는 재시작 단계에 따른 결함을 분류하고 있으며 이에 따라 결함모형은 3단계로 이루어진다. 이

와 같은 분류는 그룹관리자가 결함을 인식하면 단계에 적합한 유지보수를 선택하기 위해서이다.

표2 재시작 단계에 따른 결함의 분류

단계	결함
1	Deadlock, Infinite loop, Application memory leak, Transient exception, Corrupt primary keys, Corrupt transaction method map
2	Corrupt data inside session state
3	Bit flips in process memory, Memory leak outside application, Bit flips in process registers, Bad system call return values

표2와 같은 결함의 단계를 알기 위해서는 결함주입을 통한 결함분류와 더욱 구체적이고 세분화 되어진 결함정의가 필요하다. 서론에서 언급한 바와 같이 소프트웨어 노화에 따른 예측할 수 없는 결함을 모두 미리 정의하는 것은 불가능하다. 본 논문에서는 이를 위하여 추가적인 해결책을 제안한다. 일반적으로, 결함은 한번 발생한 후에는 예측할 수 있는 결함이 될 수도 있다. 그것은 결함발생시 컴포넌트가 어느 단계에서 유지보수가 되었는지를 기억하는 기법의 사용으로 활용될 수 있다. 이를 통해 결함은 특징에 따른 결함분류가 접목된 결함모형을 따라 최소한의 비용으로 재시작 단계를 정할 수 있다. 이렇게 얻어진 재시작 단계와 임의로 만들어진 결함의 고유이름으로 결함모형은 갱신된다.

5. 결함 모형 응용 시스템의 분석

본 논문에서는 결함주입에 대한 성능평가를 위하여 3단계 재시작을 요구하는 결함에 해당하는 프로세스 메모리의 비트플립(Bit flips in pro cess memory)을 설계하였다. 그림2는 프로세스 메모리의 비트플립에 대한 결함 발생 여부 분석을 위한 모델이다. 응용프로그램 관리자가 그룹관리자로부터 결함발생메시지를 받게 되면, 다른 결함유형들에 해당되는 여부를 검사한 후 그렇지 않다면 비트를 읽었는지를 확인한다. 비트를 읽은 후 문제가 있는 비트가 있는지를 패리티 검사등을 이용해 확인한다. 만일, 비트에 결함이 발생했다면, 프로세스 메모리 비트 에러 메시지를 그룹관리자에게 보내고, 그룹관리자는 그와 상관관계가 있는 컴포넌트들을 정지시킨다. 그 후 재시작으로 프로세스 비트 레벨에 해당하는 컴포넌트를 유지보수한다. 유지보수 후에 초기화 작업이 끝나면, 그룹관리자는 컴포넌트들의 상관관계를 다시 정의하고 시스템이 정상동작할 수 있도록 한다.

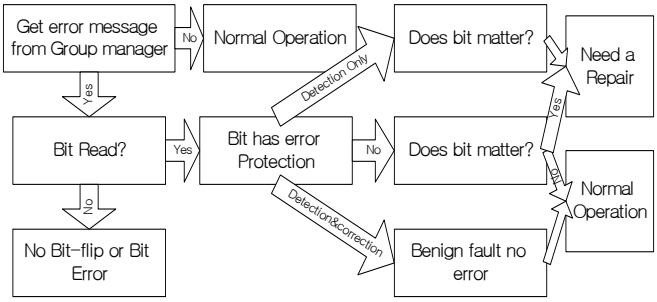

그림2 결함발생 여부 분석모형

이에 더하여, 결함검출을 위한 특징에 따른 결함분류기법이 요구된다. 특징에 따른 결함분류는 고속결함검출을 위한 첫 단계이다. 공통된 속성을 가진 결함은 하나의 집합으로 묶는다면, 더욱 신속한 결함검출이 이루어 질 수 있다. 또한, 이러한 분류기법은 재시작 단계에 따른 결함의 분류와 결합되어 매우 짧은 유지보수시간을 만들 수 있다. 그림3에서는 특징에 따른 결함분류의 기본적 아이디어를 제시하였다. 전체 결함이라는 집합 내에는 하위 단계의 응용프로그램과 네트워크, 디스크 등의 결함이 포함되어 있다.

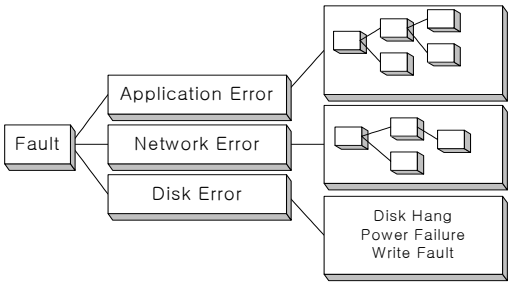

그림3. 특징에 따른 결함분류

이를 바탕으로 본 논문에서는 실제 결함검출시간을 시뮬레이션하였다. 그림4는 결함모형에 따른 결함검출시간을 시뮬레이션 한 결과이다. 앞에서 논한 결함검출기법은 결함 모형의 수와 모형에 사용된 프로시저의 수에 따라 그 시간이 결정된다. 따라서 시뮬레이션은 10개의 정의되어 있는 결함모형부터 결함정의가 확장될 경우 최대 10,000개까지의 결함모형을 분석하였다. 또한, 결함 내에 연산되어지는 프로시저의 수 및 결함정의에 사용된 연산의 수에 따라 결함검출시간의 변화도 볼 수 있다. 따라서 시뮬레이션을 위하여 부동소수점의 사칙연산시간을 바탕

으로 프로시저를 구성하여, 최소 1개의 프로시저부터 100개까지의 프로시저를 바탕으로 분석하였다. 분석 결과 약 10,000개의 결함모형에 대한 정의와 결함모형에 사용된 프로시저가 가정한 최대값이 되더라도 약 1,500ms의 시간으로 기존의 시간보다 적은 시간 소요이다. 이와 같은 결과는 결함모형에 대한 최적화와 결함모형 순차적 검색시간의 수정을 고려한다면 더 단축시킬 수 있다. 그러므로 본 논문에서 평가한 결함모형에 대한 성능은 효율적인 탐색 알고리즘이 고려된다면, 최악의 상황에서 결함이 발생하더라도 기존의 소립단위재시작보다 성능 면에서 우수함을 알 수 있다. 또한, 프로시저의 수도 특징에 따른 결함분류를 통해 크게 경감할 수 있다. 따라서 지금까지 논했던 시스템에서 결함에 대한 정보가 더 구체적으로 수집이 되어진다면 이를 바탕으로 매우 적은 평균유지보수시간이 요구되는 시스템을 구축할 수 있다. 또한, 특징 및 재시작 단계가 고려된 결함분류를 바탕으로 트리구조를 형성하여, 색인기반의 검색이 가능해 진다면, 결함검출속도의 한 변수가 될 것이다.

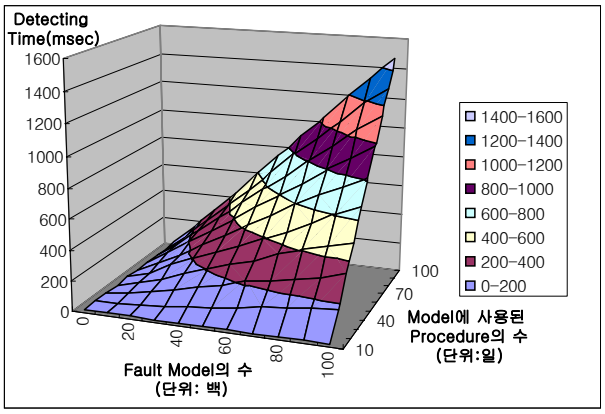

그림4. 결함모형에 따른 결함검출시간

6. 결론

본 논문에서는 결함모형기반의 응용 서비스에 대한 결함발생의 고속결함검출에 대한 단계적인 검토를 하였다. 우선, 컴포넌트별 소립단위재시작을 고려하였고, 단계별로 결함들을 나누었으며, 그에 대한 결함모형을 바탕으로 기존 기법에 비하여 평균유지보수시간을 절약하였다. 이를 위한 시스템 환경을 설계하였고, 그룹 관리자와 응용 프로그램 관리자의 상호작용으로 결함발생상황에 대한 조속한 대처가 가능하게 하였다.

향후에 결함주입으로 학습 가능한 결함분류기술과 환경에 대한 결함 단계의 구체화가 이루어져 고가용성 결함허용 시스템으로 더 나아가기 위한 꾸준한 연구가 이루어져야 할 것이다.

참고문헌

[1] D. Pradhan, "Fault-Tolerant Computer System Design", prentice Hall PTR, 1996.

[2] A. Huang and A. Fox, "Cheap Recovery: A Key to Self-Managing State" ACM Transactions on Storage, Vol. 1, No. 1, pp. 38-70, Feb. 2005.

[3] G. Candea, et al., "Microreboot - A Technique for Cheap Recovery" Proceedings of the Symposium on Operating Systems Design and Implementation, pp. 31-44, Dec. 2004.

[4] K. Nagaraja, et al. "Quantifying and Improving the Availability of High-Performance Cluster-Based Internet Services" Proceedings of the ACM/IEEE Supercomputing Conference, pp. 27, Nov. 2003.

[5] P. Broadwell, N. Sastry, and J. Traupman. "FIG: A Prototype Tool for Online Verification of Recovery Mechanisms" Proceedings of the ICS Workshop on Self-Healing, Adaptive and Self-Managed Systems, June 2002.

[6] Kiran Nagaraja, et al. "Using Fault Model Enforcement(FME) to Improve Availability" Proceedings of the Workshop on Evaluating and Architecting System Dependability, Oct. 2002.