무상태 소프트웨어의 리부팅을 통한 자가 치유 방법

홍일선, 이은석
성균관대학교 휴대폰학과
e-mail : ishong@skku.edu, leees@skku.edu

Self-Healing Method of Stateless Software by Rebooting

Ilsun Hong, Eunseok Lee
Department of Mobile Systems Engineering, Sungkyunkwan University

요 약

컴퓨팅 시스템이 복잡해 지면서 기존의 관리자에 의한 유지 보수는 한계에 직면하였다. 이에 따라 시스템이 스스로 상태를 감시하고 문제가 발생하였을 경우 스스로 문제를 해결하는 자율 컴퓨팅은 컴퓨팅 시스템의 유지 및 운영을 위한 대안으로 기대되고 있다. 자율 컴퓨팅 중 하나인 자가 치유 방법은 시스템의 문제가 발생하였을 때 스스로 치유하여 시스템을 정상 상태로 되돌리는 기법이다.

리부팅은 간단하고 실용적이며 효율적으로 다양한 시스템의 문제를 해결하는 자가 치유 방법 중 하나이다. 리부팅은 시스템의 문제 발생 원인과 위치를 알지 못해도 시스템을 빠르고 쉽게 복구할 수 있다. 그러나 리부팅 전략은 예기치 못한 데이터의 손실을 가져올 수 있으며 복구 시간이 예상보다 길어지는 등의 문제가 발생한다. 본 논문에서는 이러한 문제를 해결하기 위한 방법으로 무상태 소프트웨어와 마이크로리부팅을 이용한 소프트웨어 자가 치유 방법론을 제안한다.

1. 서론

컴퓨팅 시스템 환경은 나날이 점점 더 복잡해지고 있다. 시스템 관리자가 시스템을 수동으로 감시하는 기존의 방식은 가용한 인력자원 확보와 비용문제에 직면하게 되었다[2]. 이에 따라 시스템이 스스로 시스템의 상태를 감시하고 문제가 발생했을 경우 스스로 문제를 해결하는 자율 컴퓨팅(Autonomic Computing)은 컴퓨팅 시스템의 유지 및 운영을 위한 대안으로 기대되고 있다[1].

자율 컴퓨팅 중 하나인 자가 치유(Self-Healing) 방법은 시스템에 문제(Fault)가 발생하였을 때, 스스로 치유하여 시스템을 정상 상태로 되돌리는 기법이다[3]. 이러한 자가 치유 방법은 일반적으로 시스템의 상태를 수집하는 모니터링(Monitoring) 단계, 수집한 정보를 분석하는 분석(Analysis) 단계, 문제가 발생하였을 경우 치유 정책을 결정하는 계획(Plan) 단계, 마지막으로 치유 정책을 수행하는 실행(Execution) 단계를 거친다[2][4][5]. 이와 같은 방식은 문제가 되는 위치를 판단하여 해당되는 부분을 치유하므로 핀 포인트 치유 전략(Pin point healing strategy)이라고 부른다.

핀 포인트 치유 전략은 시스템의 문제를 정확히 파악하여 문제가 되는 컴포넌트만을 치유하므로 효율적인 치유가 가능하다. 그러나 일반적인 시스템에서 이러한 핀 포인트 치유 전략은 문제를 발생시킨 컴포넌트를 찾기가 어렵고, 그 컴포넌트를 치유하였을 경우 발생하는 또 다른 문제의 파악이 용이하지 않으며, 컴포넌트 간의 의존성으로 인해 해당 컴포넌트만을 따로 치유하는 것이 불가능한 경우가 많다. 이러한 문제로 인해 상용화 시스템에서 주로 사용하는 치유 방식으로 리부팅(Rebooting) 전략이 있다.

리부팅 전략은 간단하고 실용적이며 효율적으로 다양한 시스템의 문제를 해결하는 방법으로 알려져 있다[8]. 이러한 리부팅 전략은 시스템의 문제 발생 원인과 위치를 알지 못해도 시스템을 빠르고 쉽게 복구할 수 있는 장점을 가진다. 실제로 현재의 인터넷 서비스의 대부분은 클러스터(Cluster)를 구성하는 개별 서버 중 문제가 되는 서버를 다른 서버로 대체하고 문제가 되는 서버를 리부팅 한 후, 다시 클러스터에 합류하는 방식으로 서비스를 지속시킨다. 그러나 예기치 않은 리부팅은 데이터 손실을 야기시킬 수 있으며 이전 상태로의 복구가 용이하지 못할 경우 복구 시간이 예상보다 길어지는 문제가 발생할 수 있다.

따라서, 본 논문에서는 이러한 문제를 해결하기 위한 방법으로 다음과 같은 방법론을 제시한다.

- 무상태 소프트웨어(Stateless Software)
- 트랜젝셔널 데이터베이스(Transactional Database)
- 마이크로리부팅(Micro-Rebooting)

무상태 소프트웨어는 소프트웨어의 상태 유지를 스스로 할 필요가 없는 소프트웨어를 말하며 트랜젝셔널 데이터베이스를 통해 소프트웨어의 컨텍스트(Context)를 저장한다. 이를 통하여 리부팅을 통한 데이터의 손실을 예방한다. 또한 저장된 컨텍스트를 이용하여 시스템에 문제가 발생하였을 경우 마이크로리부팅을 통해 치유전략을 수행하여 시스템을 복구할 수 있다.

* 이 논문은 2010 년도 정부(지식경제부)의 재원으로 한국전자통신연구원(ETRI)의 지원을 받아 수행된 연구임(10035708)

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구에 관하여 기술하고, 3 장에서는 무상태 소프트웨어에 대하여 설명한다. 4 장에서는 본 논문에서 제안하는 자가 치유 방법론에 대하여 설명하고, 마지막으로 5 장에서 결론을 기술한다.

2. 관련 연구

Gray[9]는 컴퓨팅 시스템에서 오류가 발생되는 원인이 무엇이 있는지에 대하여 조사하였다. 소프트웨어 버그 중에 하이젠버그(Heisenbug)와 보어버그(Bohrbug)에 대하여 설명하고 이러한 버그에 대한 특징과 그 해결책을 제시하였다.

Fox[12]는 복구 지향 컴퓨팅(ROC, Recovery Oriented Computing)을 제시하면서 소프트웨어의 개발초기부터 하드웨어 오류나 소프트웨어 오류, 관리자에 의한 오류에 대한 복구 방안을 고려할 것을 주장하였다. 이는 오류가 발생할 때까지 걸리는 시간(MTTF, Mean Time to Failure)을 늘리는 것이 아니라 복구에 걸리는 시간(MTTR, Mean Time to Repair)을 줄이는 것이 중요하고 이를 통하여 높은 가용성(Availability)를 제공할 수 있다고 하였다.

Candea[7]는 크래쉬-온리(Crash-Only) 소프트웨어의 개념을 소개하였다. 저자는 클린 리붓(Clean Reboot)과 크래쉬 리붓(Crash Reboot)의 속도 중 크래쉬 리붓의 속도가 빠른 것을 보고, 크래쉬로 인한 데이터 손실이 없다면 크래쉬를 통해 보다 빠른 복구 전략이 가능할 것이라고 하였다. 이를 위하여 상태를 저장하는 데이터베이스가 따로 존재하여야 하며, 각각의 상태는 지속시간에 따라 저장형태를 달리해야 한다고 말하였다.

또한 Candea 의 다른 논문[6]에서는 전체 시스템을 리부팅하기 위해서는 초기화 등에 많은 시간이 걸리므로 효율적인 복구를 위해 마이크로리부팅이 필요하고 이러한 마이크로리부팅을 위해서는 개별 컴포넌트들이 독립적이어야 하며 복구를 위한 상태 저장소가 필요함을 주장하였다.

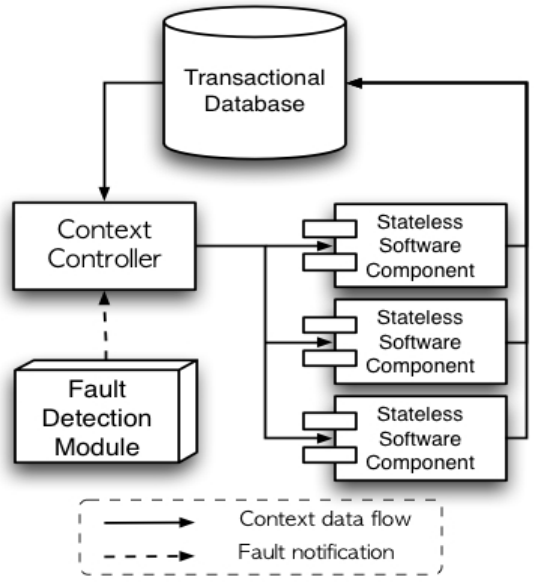
이 방법은 본 논문에서 제안하는 방법과 가장 유사한 형태를 지닌다. 그러나 인터넷 서비스와 같은 특정 도메인에 종속적인 부분이 많으며, EJB, J2EE 와 같은 다른 외부 조건을 필요로 하는 단점이 있다. 또한 컴포넌트들간의 종속성으로 인해 마이크로리부팅이 불가능한 경우 전체 시스템을 리부팅해야 하는 단점이 있다.

White[11]와 Shi[10]은 마이크로리부팅으로 리부팅해야 하는 컴포넌트를 최소화하면서 리부팅해야 하는 컴포넌트를 결정하는 방법을 제시하였다. White 는 상위 구조 모델 중 하나인 특징 모델(Feature Model)과 MMKP(Multidimensional Multiple-choice Knapsack Problem)을 이용하여 리부팅해야 하는 컴포넌트를 다항 시간(Polynomial Time)에 찾아내는 방법을 제안하였으며, Shi 는 컴포넌트간의 의존 관계와 공유 자원을 나타내는 트리 형태의 모델을 이용하여 특정 컴포넌트에 리부팅이 필요한 경우 함께 리부팅해야 하는 컴포넌트를 결정하는 방법을 제안하였다.

3. 무상태 소프트웨어

무상태 소프트웨어(Stateless Software)는 소프트웨어의 상태 유지를 스스로 할 필요가 없는 소프트웨어를 말한다. 일반적으로 소프트웨어는 지속적으로 저장해야 하는 정보(Permanent Information)와 프로그램의 수행을 위해 일시적으로 필요한 정보(Temporary Information)를 가진다. 그 중 지속적으로 저장해야 하는 정보는 손실 없이 보존되어 소프트웨어가 필요로 하는 시점에 사용되어야 한다. 이러한 시점(when)과 정보(what)를 컨텍스트(Context)라고 한다.

무상태 소프트웨어는 컨텍스트 정보를 안전한 저장소에 저장하고 그것을 이용하여 작업을 수행한다. 따라서 무상태 소프트웨어는 오류(Fault)가 발생했을 경우 그에 따른 데이터 손실이 없으며 저장된 컨텍스트 정보를 이용하여 시스템을 복구(Recovery)할 수 있다.

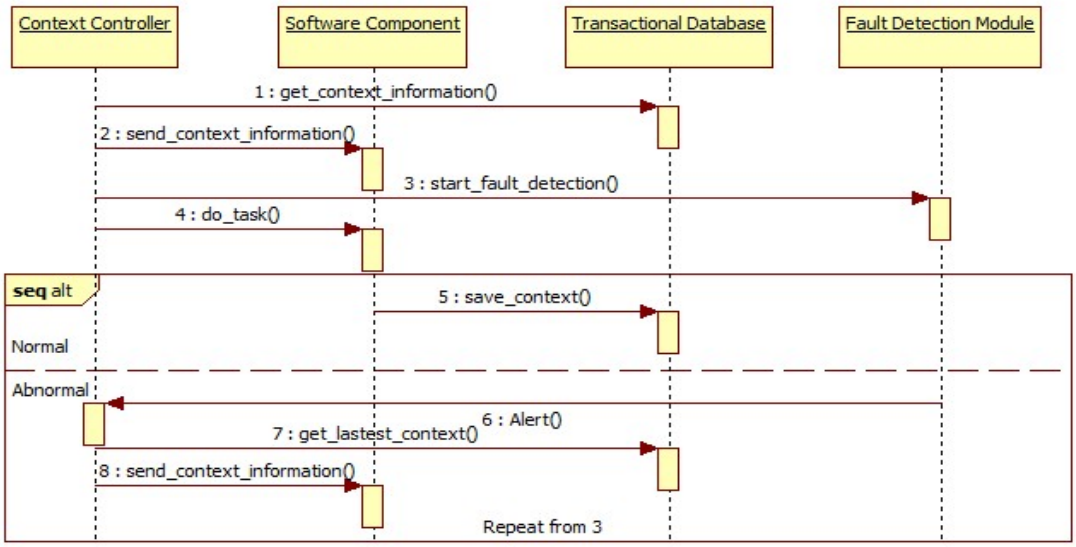


(그림 1) 무상태 소프트웨어의 구성

무상태 소프트웨어는 그림 1 과 같이 무상태 소프트웨어 컴포넌트(Stateless Software Components), 컨텍스트 컨트롤러(Context Controller), 트랜젝셔널 데이터베이스(Transactional Database), 오류 탐지 모듈(Fault Detection Module)로 구성되어 있다.

무상태 소프트웨어 컴포넌트는 실제로 소프트웨어가 기능적 요구사항을 만족시키기 위해 해야 할 일을 수행한다. 무상태 소프트웨어 컴포넌트는 컨텍스트 컨트롤러로부터 컨텍스트 정보를 받아 그 컨텍스트에 해당하는 작업을 수행한다. 작업을 정상적으로 수행한 경우 작업을 통해 갱신된 컨텍스트를 데이터베이스에 기록하고 종료한다.

컨텍스트 컨트롤러는 트랜젝셔널 데이터베이스로부터 컨텍스트 정보를 가져와 어떠한 소프트웨어 컴포넌트를 수행시켜야 할 지를 결정한다. 또한 소프트웨어에 문제가 발생하였을 경우 마이크로리부팅을 통하여 문제를 해결한다.



(그림 2) 무상태 소프트웨어의 동작 방식

트랜젝셔널 데이터베이스는 무상태 소프트웨어 컴포넌트와 통신하며 보존해야 할 정보를 저장한다. 데이터베이스의 트랜젝션(Transaction)은 무상태 소프트웨어 컴포넌트가 동작을 정상적으로 완료할 경우에만 데이터베이스에 기록하게 한다. 이러한 방식을 통해 소프트웨어 컴포넌트의 실행 도중 문제가 발생하는 경우 롤백(Roll Back)되어 문제가 발생한 경우의 정보를 데이터베이스에 저장하지 않도록 한다.

오류 탐지 모듈은 무상태 소프트웨어 컴포넌트에 문제가 발생하였을 경우 이를 감지하고 컨텍스트 컨트롤러에게 알리는 역할을 한다. 본 논문에서 오류 탐지 방식은 워치독(Watchdog) 방식을 사용하였다, 이 방식은 컨텍스트 컨트롤러가 워치독 타이머를 설정하고 프로그램 수행 중 문제가 발생하여 정상적인 시간 내에 프로그램의 수행을 종료하지 못하면 오류가 발생한 것으로 판단한다.

정상적인 상태의 무상태 소프트웨어의 실행 순서는 그림 2 와 같다.

1. 컨텍스트 컨트롤러가 컨텍스트 정보를 트랜젝셔널 데이터베이스로부터 가져온다.
2. 컨텍스트 컨트롤러는 컨텍스트에 해당하는 무상태 소프트웨어 컴포넌트를 선택하고 컨텍스트 정보를 전달한다.
3. 컨텍스트 컨트롤러는 오류 탐지 모듈을 설정하고 무상태 소프트웨어 컴포넌트를 실행시킨다.
4. 무상태 소프트웨어 컴포넌트는 컨텍스트 정보를 읽어 해야 할 일을 결정한 후 수행한다.
5. 무상태 소프트웨어 컴포넌트는 수행한 작업이 정상 종료된 경우에 트랜젝셔널 데이터베이스에 컨텍스트 정보를 업데이트한다.
6. 위의 과정을 반복한다.

4. 무상태 소프트웨어의 자가 치유 방법

소프트웨어의 오류는 주로 소프트웨어 컴포넌트에서 발생하게 된다. 소프트웨어 컴포넌트가 해당 작업을 정상적으로 종료시키지 못하는 경우, 오류 탐지 모듈은 이를 감지하여 컨텍스트 컨트롤러에게 알린다. 오류 탐지 모듈은 상태 분석 방식(State Analysis), 오류 분석 방식(Fault Analysis) 등으로 구현될 수 있으나 본 논문에서는 워치독 방식을 사용하였다.

오류가 탐지되면 이를 해결하기 위하여 마이크로리부팅을 통한 치유가 수행된다. 마이크로리부팅은 전체 시스템을 리부팅하는 것이 아니라 문제 발생이 의심되는 컴포넌트만을 리부팅 하는 것을 의미한다. 무상태 소프트웨어 컴포넌트의 경우 리부팅에 의한 데이터 손실이 없어 안정적으로 리부팅이 가능하다.

리부팅 후에 컨텍스트 컨트롤러는 트랜젝셔널 데이터베이스로부터 마지막으로 정상 종료된 컨텍스트를 불러와 다시 그 컨텍스트에 해당하는 작업을 수행하게 함으로써 치유를 수행한다.

그러나 리부팅을 통하여 치유를 수행하였음에도 오류가 해결되지 않는 경우 트랜젝셔널 데이터베이스의 컨텍스트 히스토리(Context History)를 이용하여 치유를 수행하게 된다. 컨텍스트 히스토리는 정상적으로 종료된 컨텍스트 정보의 집합으로 시간 순으로 기록되어 있다. 소프트웨어의 치유는 시간의 역순을 따라 컨텍스트 정보를 가져와 마이크로리부팅을 통하여 수행되게 된다. 이러한 치유는 소프트웨어가 정상 작동할 때까지 반복하거나 사전 정의된 리부팅 회수만큼 치유 정책을 수행하고 종료하게 된다.

다음은 오류가 발생하였을 경우의 자가 치유 방법이다.

1. 오류 탐지 모듈이 오류를 감지하여 컨텍스트 컨트롤러에게 오류가 발생한 것을 알린다.
2. 오류 탐지 모듈은 트랜젝셔널 데이터베이스로부터 마지막으로 정상 종료된 컨텍스트를 가져온다.
3. 컨텍스트 컨트롤러는 받아온 컨텍스트에 해당하는 소프트웨어 컴포넌트를 리부팅한다.
4. 리부팅된 컴포넌트에게 컨텍스트 정보를 전달한다.

5. 무상태 소프트웨어 컴포넌트가 작업을 다시 수행한다.
6. 작업을 수행한 결과에 따라 두가지로 작동한다.
   A. 정상적으로 작업이 종료되는 경우 무상태 소프트웨어 컴포넌트가 작업을 완료한 경우 정상 상태로 복귀한다.
   B. 비정상적으로 작업이 종료되는 경우 다음을 수행한다.
7. 컨텍스트 컨트롤러가 트랜젝셔널 데이터베이스로부터 바로 이전 컨텍스트 정보를 가져온다.
8. 3 번 과정을 다시 수행한다.

5. 평가

제안된 방법은 시스템의 신뢰성을 높여주지만 치유 방법을 위한 오버헤드로 인해 성능상의 하락이 발생한다. 성능 하락은 컨텍스트 정보를 저장하기 위한 트랜젝셔널 데이터베이스와의 통신에서 주로 발생하게 된다. 따라서 이러한 오버헤드는 데이터베이스와의 접근 회수와 저장되는 컨텍스트 정보의 양에 의존적이다.

본 논문을 평가하기 위하여 제안된 방법으로 소프트웨어를 설계하고 구현하였다. 또한 오버헤드를 측정하기 위하여 같은 일을 수행하는 일반 소프트웨어를 구현하였다. 수행한 결과 약 15% 정도의 성능하락이 발생함을 볼 수 있었다.

이러한 성능하락은 시스템의 유지 보수 비용 및 데이터 손실에 따른 비용 감소에 대한 트레이드오프로 상황에 맞는 접근 방식이 필요하다.

6. 결론

리부팅은 실용적이고 효과적인 자가 치유 방법으로 다른 방법에 비하여 실제 시스템에 적용하기 용이하다. 그러나 전체 시스템을 리부팅하는 경우 예상치 못한 시스템의 오동작이나 데이터의 손실을 초래할 수 있다. 본 논문에서는 이러한 문제를 해결하기 위하여 무상태 소프트웨어와 마이크로 리부팅을 통한 자가 치유 방법을 제안하였다. 제안된 방법은 성능상 약간의 오버헤드가 존재하지만 시스템의 신뢰성을 높일 수 있으며 데이터의 손실을 막을 수 있다.

현재 실제 시스템에 적용하기 위하여 설계 방법에 대한 연구가 이루어지고 있으며, 마이크로리부팅을 통한 효율적인 자가 치유 시스템을 개발 중에 있다. 또한 크고 복잡한 시스템에서 높은 신뢰성이 요구되는 부분에 선택적으로 마이크로리부팅을 적용하는 방안에 대하여 연구하고 있다.

참고문헌

[1] Jeffrey O. Kephart, “Research Challenges of Autonomic Computing”, Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference, pp.15-22, 2005.

[2] Jeffrey O.Kephart and David M. Chess, “The Vision of Autonomic Computing”, Computer, vol. 36, pp. 41-50, 2003.

[3] D. Ghosh, R. Sharman, H. R. Rao and S. Upadhyaya, “Self-healing - survey and synthesis”, Decision Support Systems in Emerging Economies, vol. 42, pp. 2164-2185, 2007.

[4] David Garlan and Bradley Schmerl, “Model-based adaptation for self-healing systems”, Proceedings of the first workshop on Self-healing systems, pp. 27-32, 2002.

[5] Matthias Rohr, Marko Boskovic, and Simon Giesecke, “Model-driven development of self-managing software”, Workshop in conjunction with MoDELS/UML2006, 2006.

[6] George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman and Armando Fox, “Microreboot - A Technique for Cheap Recovery”, OSDI ’04: 6th Symposium on Operating Systems Design and Implementation, pp. 31-44, 2004.

[7] George Candea and Armando Fox. “Crash-Only Softwrae”, Proceedings of the 9th conference on Hot Topics in Operating Systems, vol. 9, pp. 1-6, 2003.

[8] Brewer, E.A., “Lessons from giant-scale services”, Internet Computing, IEEE, vol. 5, pp. 46-55, 2001.

[9] Jim Gray, “Why Do Computers Stop and What Can Be Done About It?”, Technical Report 85.7, 1985

[10] Changting Shi, Rubo Zhang and Bailong Liu, “Layered Self-healing Software Architecture of AUV Based on Micro-reboot, Intelligent Systems and Applications, 2009. ISA 2009. International Workshop, pp. 1-4, 2009.

[11] J. White, B. Dougherty, H.D. Strowd, and D.C. Schmidt, “Using Filtered Cartesian Flattening and Microrebooting to Build Enterprise Applications with Self-adaptive Healing”, Self-Adaptive Systems, LNCS 5525, pp. 241-260, 2009.

[12] Armando Fox, “Toward recovery-oriented computing”, Proceedings of the 28th international conference on Very Large Data Bases, pp. 873-876, 2002