소프트웨어 오류 탐지를 위한 아키텍처 기반의 다계층적 자가적응형 모니터링 방법

(An Architecture-based Multi-level Self-Adaptive Monitoring Method for Software Fault Detection)

윤 현 지 † 박 수 용 ††
(HyunJi Youn) (SooYong Park)

요 약 Mission-critical 시스템의 경우 자가 치유는 신뢰성을 보장하기 위한 기술 중 하나이다. 자가 치유는 오류 탐지와 오류 회복으로 이루어져 있으며 오류 탐지는 오류 회복을 가능하게 하는 자가 치유의 중요한 첫 단계이지만 시스템에 과부하를 주는 문제가 있다. 모델 기반의 방법 등으로 오류를 탐지할 수 있는데 시스템의 모든 행위를 통지하고 정상 행위 모델과 통지된 시스템의 행위를 비교하여야 하므로 그 양이 많고 부하가 크기 때문이다. 본 논문에서는 모델 기반의 오류 탐지 방법을 보완하는 아키텍처 기반의 다계층적 자가적응형 모니터링 방법을 제안한다. 소프트웨어 아키텍처 상에서 오류 탐지의 중요도는 컴포넌트 마다 다르다. 각 컴포넌트마다 발생하는 오류의 심각도와 빈도가 다르기 때문이다. 모니터링 중요도가 높은 컴포넌트에는 강도가 높고 모니터링 중요도가 낮은 컴포넌트에는 강도가 낮도록 모니터가 적응한다면 오류 탐지의 부하는 줄이고 효율은 유지시킬 수 있다. 또한 소프트웨어의 환경 변화 및 아키텍처 상의 변화 등에 따라 오류 발생 빈도가 변화하여 컴포넌트의 오류 탐지 중요도가 변화하기 때문에 학습을 통해 이를 추적하여 자가적응적으로 중요도가 높은 컴포넌트를 집중 모니터링 한다.

키워드 : 자가 치유, 오류 탐지, 과부하, 모니터링, 아키텍처

Abstract Self-healing is one of the techniques that assure dependability of mission-critical system. Self-healing consists of fault detection and fault recovery and fault detection is important first step that enables fault recovery but it causes overhead. We can detect fault based on model, the detection tasks that notify system's behavior and compare normal behavior model and system's behavior are heavy jobs. In this paper, we propose architecture-based multi-level self-adaptive monitoring method that complements model-based fault detection. The priority of fault detection per component is different in the software architecture. Because the seriousness and the frequency of fault per component are different. If the monitor is adapted to intensive to the component that has high priority of monitoring and loose to the component that has low priority of monitoring, the overhead can be decreased and the efficiency can be maintained. Because the environmental changes of software and the architectural changes bring the changes at the priority of fault detection, the monitor learns the changes of fault frequency and that is adapted to intensive to the component that has high priority of fault detection.

Key words : Self-healing, Fault detection, Overhead, Monitoring, Architecture

† 학생회원 : 서강대학교 컴퓨터공학과
dpwisdom@empal.com
†† 정 회 원 : 서강대학교 컴퓨터공학과 교수
sypark@sogang.ac.kr
(Corresponding author임)
논문접수 : 2010년 5월 3일
심사완료 : 2010년 5월 31일

정보과학회논문지: 소프트웨어 및 응용 제37권 제7호(2010.7)

1. 서 론

시스템이 정상적으로 동작하는 데 실패하는 경우 커다란 위험 및 손실이 따르는 Mission-critical 시스템의 경우 소프트웨어 신뢰성은 가장 중요한 품질 속성 중 하나이다[1]. 자가치유는 이러한 신뢰성을 보장하기 위한 기술 중 하나이며 소프트웨어 스스로 자신의 오류를 탐지하고 오류를 회복하도록 하는 것이 그 목적이다[2].

이 중 오류 탐지는 자가 치유를 위한 첫 번째 단계로

오류 탐지 단계에서 오류를 정확히 탐지하여야 해당 오류를 치유할 수 있기 때문에 매우 중요한 단계이지만 탐지를 위해 추가된 작업 때문에 시스템에 과부하를 가져오는 문제가 있다[3,4]. 모델 기반으로 오류를 탐지하는 경우 정상 행위 모델을 구축하여 시스템이 이 모델과 벗어나는 행위를 보이는 경우 오류를 탐지한다. 이를 위해서는 시스템의 모든 행위를 모니터에게 통지해야 하고 모니터에서는 통지된 정보를 바탕으로 시스템의 행위를 모델과 비교해야 한다[5]. 이러한 작업은 그 양이 많고 부하가 크며 결과적으로 시스템의 성능을 감소시킨다. 이러한 문제를 해결하기 위해서 본 논문에서는 오류 탐지 메커니즘의 부하를 감소시키면서 탐지 효율은 유지하는 것을 목적으로 아키텍처 기반의 다계층적 자가적응형 모니터링을 제안한다.

아키텍처 기반의 다계층적 자가적응형 모니터링에서는 소프트웨어 아키텍처 상에서 오류 탐지의 중요도는 컴포넌트마다 다르다는 것을 기반으로 모니터가 컴포넌트마다 오류 탐지의 중요도에 따라 자가적응하여 모니터링 강도를 조정한다. 또한 소프트웨어의 환경 변화 및 아키텍처 상의 변화에 따른 오류 발생 빈도의 변화에 적응하여 컴포넌트 별 모니터링 강도를 조정한다.

본 논문의 구성은 다음과 같다. 2장에서는 컴포넌트 모니터링 중요도에 대해서 분석하고 3장에서는 다계층적 자가적응형 모니터링 방안을 설명한다. 4장에서는 제안한 다계층적 자가적응형 모니터링 방안의 검증 결과를 설명하며 5장에서는 관련연구를 소개하고 6장에서는 결론 및 향후 연구를 제시한다.

2. 컴포넌트 모니터링 중요도

아키텍처를 구성하고 있는 각 컴포넌트는 각기 다른 모니터링 중요도를 갖고 있기 때문에 각기 다른 강도로 모니터링을 수행해야 한다. 본 논문에서는 모니터링 중요도를 오류 심각도와 오류 발생 빈도로 정의한다. 오류 심각도와 오류 빈도를 바탕으로 모니터링 중요도를 결정하며 그림 1처럼 각 컴포넌트 별로 대응되는 이 세 가지 정보를 유지한다.

핵심이 되는 작업 및 실패할 경우 치명적인 영향을 유발하는 작업을 수행하는 핵심 컴포넌트의 경우에는 다른 컴포넌트에 비해 발생하는 오류의 위험성이 크기 때문에 오류 심각도는 다음과 같은 기준으로 결정되며 아키텍처 모델 상에서 직접 지정된다.

- 핵심 컴포넌트, 주변 컴포넌트 여부

컴포넌트 별 오류 발생 빈도는 각각 다르며 오류 발생 빈도가 높은 경우 모니터링의 주요 대상이 된다. 오류 발생 빈도는 다음과 같은 통계 정보를 바탕으로 결정되며 실행 중 변경될 수 있으므로 동적으로 학습되어 학습된 이후 다음 학습 주기 전까지 컴포넌트의 오류 발생 빈도를 나타내는 수치로써 사용된다.

- 오류 탐지 횟수/전체 모니터링 횟수

모니터링 중요도는 다음과 같은 기준으로 결정되며 오류 심각도와 오류 빈도에 따라 결정된다.

- 집중적 모니터링, 느슨한 모니터링 여부

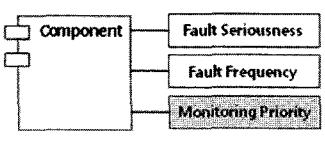

그림 1 컴포넌트 모니터링 중요도

3. 다계층적 자가적응형 모니터링

본 논문에서는 그림 2와 같이 모니터가 각 컴포넌트마다 모니터링 중요도에 따라 자가 적응하여 모니터링 강도를 변경하는 자가적응형 모니터링을 통해 오류탐지에서의 과부하문제에 대한 향상 방안을 제안한다.

3.1 개요

컴포넌트 별 모니터링 중요도에 따라 모니터링 강도를 조정하기 위해서는 중요도가 높은 경우 상세한 모든 행위를 모니터에 통지하고 중요도가 낮은 경우 개략적인 행위만을 통지한다. 그리고 시스템의 실제 행위와 비교 대상이 되는 정상 행위 모델의 상세한 정도를 조절한다. 그림 2와 같이 강도를 높이기 위해서는 정상 행위에 대한 상세한 모델을 이용하며 강도를 낮추기 위해서는 정상 행위에 대한 간략한 모델을 사용한다. 소프트웨어 환경 변화 및 아키텍처 상의 변화로 인한 컴포넌트 별 오류 발생 빈도의 변화 역시 지속적으로 학습하여 적응한다.

소수의 핵심 컴포넌트와 오류 발생 빈도가 높은 컴포넌트만 집중적으로 모니터링 하기 때문에 이를 통해 오류 탐지의 부하를 줄이면서 효율을 유지할 수 있다.

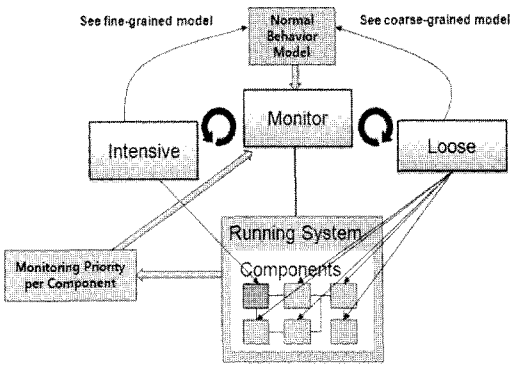

그림 2 다계층적 자가적응형 모니터링 개요

3.2 집중적/느슨한 모니터링(Intensive/Loose Monitoring)

모델 기반의 오류 탐지에서는 상태도를 이용하여 시스템의 정상 행위를 모델링 할 수 있다[5]. 이 상태도는 시스템 설계 시 사용하는 상태도와는 그 사용법이 다르며 컴포넌트의 동적 행위와 다른 컴포넌트와의 상호작용을 모델링 하여 시스템 행위에 대한 순서도와 같은 개념으로 명세 한다. 시스템은 각 행위가 완료되면 모니터에게 해당 상태에 해당하는 행위가 완료되었음을 통지한다. 시스템의 통지와 대응되는 상태도의 상태 정보는 모델에서 유지한다. 모니터는 통지를 받고 상태도에서 다음 행위를 나타내는 상태로 전이시킨다. 만약 정해진 시간 내에 명세된 대로 상태 전이가 일어나지 않으면 시스템이 모델과 벗어난 행위를 보이는 것으로 간주하여 오류를 탐지한다. 이를 통해 모니터는 정확하고 즉각적으로 문제가 생긴 위치를 찾아낼 수 있다.

본 논문에서 시스템의 행위는 그림 3의 Intensive Monitoring의 모델과 같이 계층적인 구조로 모델링한다. 그림 3에서 UpdateInfo라는 상태는 정보를 수정하는 행위를 나타내는 상태이며 하위에는 각각의 정보를 수정하는 행위를 나타내는 하위 상태가 있다. Intensive Monitoring의 경우 이 세 가지 하위 상태가 나타내는 모든 행위를 통지하고 상태도에서 상태 전이를 시킴으로써 모니터링 하지만 Loose Monitoring의 경우 UpdateInfo라는 최상위 상태만 인식하고 모든 정보 수정이 완료된 시점에서 한 번의 행위만 통지한다 이를 통해 모니터링 작업에 수반되는 통지 및 모델과 시스템 행위와의 비교 빈도를 줄임으로써 부하를 줄인다. 반면 정확한 오류 발생 위치를 탐지하기 보다는 UpdateInfo의 세 가지 하위 상태 중 하나라는 근사 지점을 탐지하고 탐지 속도도 느리다. 이 경우 치유 시에 보다 정확한 정보를 전달할 수는 없다. 하지만 탐지되지 않는 오류는 없으며 Loose Monitoring은 모니터링 중요도가 낮은 컴포넌트에 적용되어 부하를 줄이고 Intensive Monitoring은 모니터링 중요도가 높은 컴포넌트에 적용되어 탐지 효율은 유지시키게 된다.

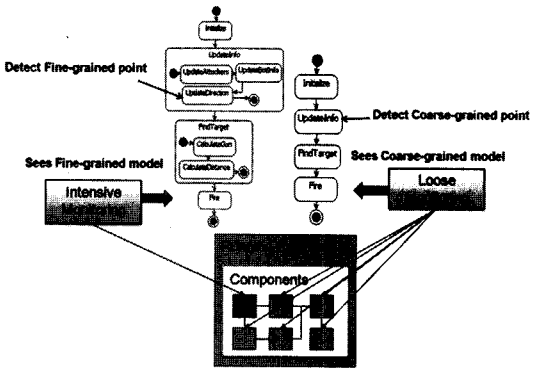

그림 3 Intensive/Loose Monitoring

3.3 자가적응형 모니터링

환경 및 시스템 변화에 따른 모니터링 중요도의 변화를 추적하기 위해 실행 중 동적으로 오류 발생 빈도를 학습하여 그에 따라 모니터가 자가적응한다.

3.3.1 자가적응을 위한 학습

오류 발생 빈도를 학습하기 위해 학습 데이터를 지속적으로 큐에 쌓으며 그림 4처럼 일정량의 데이터가 쌓일 때마다 가장 최근의 일정량의 데이터를 기반으로 학습하고 오래된 데이터는 큐에서 삭제한다.

학습을 위한 데이터는 다음과 같다.

- 오류 탐지 결과(정상/비정상 여부)
- 해당 컴포넌트 Id

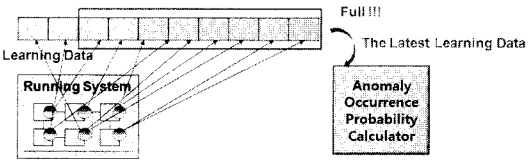

그림 4 오류 발생 확률 학습

3.3.2 모니터링 중요도에 따른 자가적응

모니터는 핵심 컴포넌트의 경우 모니터링 강도를 자가적응 하지 않는다. 항상 Intensive Monitoring을 수행하며 주변 컴포넌트의 경우에는 오류 탐지 확률 상위 25%의 컴포넌트만 오류가 뭉쳐서 나타나는 모듈로 간주하여 Intensive Monitoring을 수행하고 나머지는 Loose Monitoring을 수행하도록 적응한다. 이는 전체 모듈의 20%에서 80%의 오류가 검출되고 전체 모듈의 50% 정도는 오류가 없다는 소프트웨어에서 나타나는 현상[6,7]을 바탕으로 하며 실험에서 이러한 수치가 가장 부하 감소가 많고 탐지 효율 저하가 적은 것으로 나타났기 때문이다.

적응에 필요한 것은 모델의 상세도 조정이다. 모니터에서는 컴포넌트 별 모니터링 중요도 정보를 참조하여 각 컴포넌트 중요도에 따라 해당 컴포넌트의 정상행위 모델을 하위 상태까지 모두 인식하거나 혹은 최상위 상태만을 인식한다.

4. 검증

본 논문의 검증을 위해서 가상의 전투 장에서 다른 로봇과 전투하는 로봇을 프로그래밍하는 게임인 Robocode [8]의 우승 로봇 Fermat에 다계층적 자가적응형 모니터링 방법을 적용한 오류 탐지 기능을 추가하는 실험을 진행하였다. 실험을 위해 대부분의 프로젝트에서 오류가 발

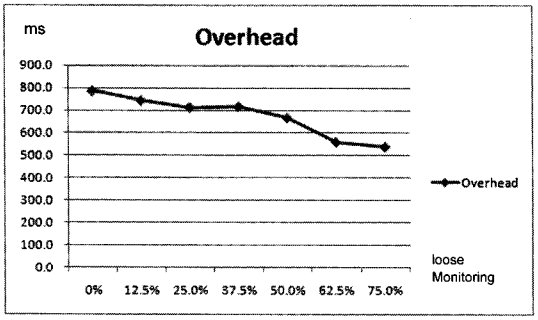

그림 5 Loose Monitoring 적용 정도에 따른 부하 감소

생하는 패턴[6]과 동일하게 오류를 삽입하였으며 모니터에서 예측할 수 없도록 무작위로 발생시켰다.

본 실험에서는 제안하는 모니터링 방법의 부하 감소 정도와 탐지 효율 저하 정도를 측정하였다. 부하 감소 정도는 그림 5처럼 Loose Monitoring의 적용 정도가 증가함에 따라 보다 효과적인 것으로 나타났다.

또한 오류 발생 시점부터 오류가 탐지되는 데까지 걸리는 시간은 Loose Monitoring이 적용됨에 따라 다소 증가하기 시작했다. 그 이유는 모델과 시스템의 행위를 자주 비교하지 않기 때문에 비교가 이루어지는 시점까지 시간이 걸리기 때문이다. 그러나 Loose Monitoring 적용 정도가 증가함에 따라 부하가 감소하고 그 효과로 인해 다시 감소하여 그림 6처럼 Loose Monitoring 75% 적용의 경우에 제안하는 방법과 일반 모니터링 사이에 큰 차이는 없었다.

오류 탐지 확률 상위 25%의 컴포넌트를 Intensive Monitoring하도록 적응하는 경우 31.6%의 상당한 부하 감소 효과를 보였으며 6%의 적은 탐지 속도 저하를 보였다. 그러나 탐지하지 못하는 오류는 없었다.

로봇 실행 중 컴포넌트의 오류 분산을 변경하여 환경 변화 등으로 인한 모니터링 중요도의 변화를 시뮬레이션한 결과 학습 큐의 크기가 감소함에 따라 그림 7처럼

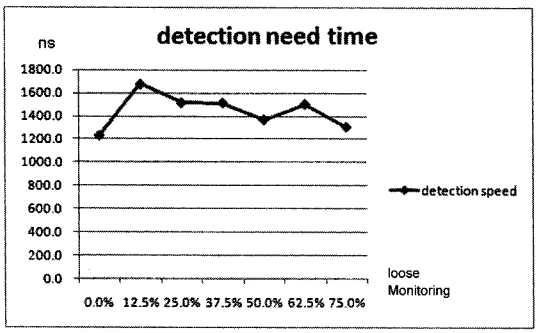

그림 6 Loose Monitoring 적용 정도에 따른 탐지 소요 시간

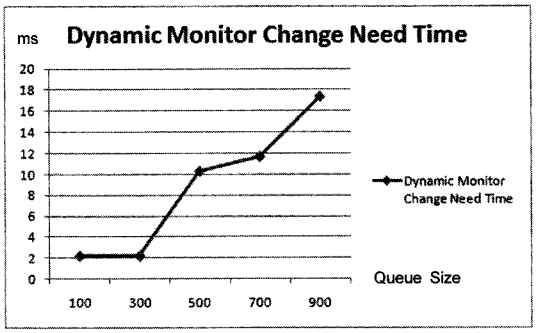

그림 7 모니터의 자가적응 소요 시간

적응에 걸리는 시간이 줄어들었으며 크기가 100인 경우 2ms 안에 적응하였다.

5. 관련연구

소프트웨어에서 오류는 뭉쳐서 나타난다는 사실은 수십 년 전부터 최근에 이르기까지 널리 알려져 있는 사실이다[6,7]. 이를 뒷받침하는 관찰 결과에는 공통적으로 오류의 50~80%가 모듈의 15~20%에서 발견되며 오류의 절반 정도는 오류가 없다고 나타나고 있다.

오류 탐지는 모델 기반으로 이루어질 수 있으며 상태도 등의 모델 기반[5]이나 아키텍처 모델 기반[3,9,10] 등으로 이루어진다. 상태도 기반의 오류 탐지에서는 시스템의 행위 하나 하나가 시작하고 끝나는 시점을 하나의 상태로 간주하여 시스템의 행위와 다른 모듈간의 상호작용을 모델링 함으로써 정상 행위 모델을 구축한다. 아키텍처 모델 기반의 오류 탐지에서는 아키텍처 모델과 시스템의 속성을 연결시키고 해당 속성을 모니터링 함으로 인해 아키텍처 상에서 어떠한 부분에 오류가 있는지 탐지한다.

오류 탐지 연구들에서는 과부하를 주지 않으면서 오류 탐지 메커니즘을 구현하는 것이 자주 이슈가 된다[3,4]. 이를 위해서 모니터를 계층적으로 구성하여 부하를 줄이거나[11,12] 모니터링 시점에 따라 오류 정보를 클러스터링 하는 방법[13], 시스템의 정상 행위 내에 모니터링 정보를 삽입하는 방법[14] 등이 제안되었다. 이러한 방법들은 모니터링 부하를 줄일 수 있었으나 아키텍처 전체에 대해 하나의 모니터링 전략 및 과부하 감소 전략을 사용하였기 때문에 모니터링의 중요도가 각각 상이한 컴포넌트 단위로 효율화를 하지 못했다. 결국 과부하를 줄임으로써 가져오게 되는 탐지 효율의 저하 효과가 아키텍처 전체에 걸쳐 영향을 끼치는 단점이 있었다.

6. 결론 및 향후 연구

자가치유의 첫 번째 단계인 오류 탐지는 그 중요성에

비해 추가되는 작업이 과다하기 때문에 과부하를 일으킨다는 문제점을 가지고 있었다.

이의 해결 방법으로 아키텍처 기반의 다계층적 자가적응형 모니터링 방안을 제안하였고 Robocode 우승 로봇인 Fermat에 적용하여 그 효과를 분석하였다.

본 논문에서는 아키텍처 상에서 각 컴포넌트의 모니터링 중요도가 각각 상이하다는 점을 토대로 모니터링 중요도가 높은 컴포넌트에는 강도가 높고 모니터링 중요도가 낮은 컴포넌트에는 강도가 낮도록 모니터가 자가 적응하여 오류 탐지의 부하는 줄이고 효율은 유지시켰다. 또한 환경 변화 및 아키텍처 상의 변화에 따라 모니터링 중요도가 변화하더라도 그에 따라 적응할 수 있도록 하였다.

실험 결과 핵심 컴포넌트 및 오류 발생 확률 상위 25%의 컴포넌트를 Intensive Monitoring 하도록 적응하는 경우 31.6%의 부하가 감소되는 등 상당량의 부하를 감소시킬 수 있었다. 또한 모든 컴포넌트를 Intensive Monitoring 한 경우와 마찬가지로 100%의 오류를 탐지할 수 있었고 탐지 속도는 6% 저하되는 적은 량의 탐지 효율 감소를 보였다.

향후 연구에서는 아키텍처 상의 변화나 환경 변화에 따라 핵심 컴포넌트가 변경되는 경우 자가적응적으로 핵심 컴포넌트를 판별하기 위해 컴포넌트 간의 상호 작용을 분석하는 방법을 추가할 예정이다.

참 고 문 헌

[1] "IEEE Standard Dictionary of Measures of the Software Aspects of Dependability," *IEEE Std 982.1-2005 (Revision of IEEE Std 982.1-1988)*, IEEE Press, 2006.

[2] Ghosh, D., Sharman, R., Rao, H.R., Upadhyaya, S.: Self-healing systems - survey and synthesis, Decision Support System, vol.42, no.4, pp.2164-2185 (2007).

[3] D. Garlan and B. Schmerl, Model-based adaptation for self-healing systems, Proceedings of the first workshop on Self-healing systems, ACM Press, Charleston, South Carolina, 2002.

[4] M.G. Merideth, P. Narasimhan, Proactive containment of malice in survivable distributed system, International Conference on Security and Management, Las Vegas, NV, 2003.

[5] Michael E. Shin, and Yan Xu, Detection of Anomalies in a Software Architecture with connectors, *International Workshop on System/Software Architectures (WSSA05)*, LasVegas, Nevada, USA, vol.61, Issue 1, pp.6-26, June 2005.

[6] C. Andersson and P. Runeson, "A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems," *IEEE Trans. Software Eng.*, vol.33, no.5, pp.273-286, May 2007.

[7] Davis, Alan M. 1995. 201 Principles of Software Development. New York: McGraw-Hill. Principle 114.

[8] http://robocode.sourceforge.net/

[9] S.W. Cheng, D. Garlan, B. Schmerl, P. Steenkiste, N. Hu, Software architecture-based adaptation for grid computing, *The 11th IEEE Conference on High Performance Distributed Computing (HPDC '02)*, Edinburgh, Scotland., 2002.

[10] G. Valetto, G.E. Kaiser, Case study in software adaptation, *Proceedings of the FirstWorkshop on Self-Healing Systems*, 2002.

[11] S. Bagchi, B. Srinivasan, K. Whisnant, Z. Kalbarczyk, and R. Iyer, Hierarchical Error Detection in a Software Implemented Fault Tolerance (SIFT) Environment, *IEEE Transactions on Knowledge and Data Engineering*, vol.12, no.2, pp.203-224, March/April 2000.

[12] Jinho Ahn, Efficient Failure Detection and Recovery Scheme for Hierarchical Distributed Monitoring, fgcn, vol.2, pp.510-515, Future Generation Communication and Networking (FGCN 2007) - Volume 1, 2007.

[13] Midori Sugaya, Yuki Ohno, Andrej van der Zee, Tatsuo Nakajima, A Lightweight Anomaly Detection System for Information Appliances, isorc, pp.257-266, 2009 *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2009.

[14] Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, Theo Ungerer, A Lazy Monitoring Approach for Heartbeat-Style Failure Detectors, ares, pp.404-409, 2008, *Third International Conference on Availability, Reliability and Security*, 2008.

윤 현 지

2004년 홍익대학교 컴퓨터공학과(공학사) 2010년 서강대학교 컴퓨터공학과(공학석사). 관심분야는 자가치유, 소프트웨어 아키텍처

박 수 용

정보과학회논문지 : 소프트웨어 및 응용 제 37 권 제 2 호 참조