고가용성 유비쿼터스 컴퓨팅 시스템을 위한 오토노믹 자가 관리 메커니즘

(An Autonomic Self-management Mechanism for High-available Ubiquitous Computing System)

최 창 열 † 김 성 수 ††
(Changyeol Choi) (Sungsoo Kim)

요 약 유비쿼터스 컴퓨팅은 네트워크로 상호연결된 프로세서로 구성되며, 하나 이상의 컴퓨터로 이루어진다. 따라서 한 시스템의 단일 컴포넌트 뿐만 아니라 전체 시스템의 가용도를 유지할 수 있어야만 한다. 본 연구에서는 사람의 간섭을 최소화하면서 예기치 못한 결함이 발생하여도 서비스를 지속적으로 제공할 수 있는 능력을 시스템에 부여하기 위한 오토노믹 자가 관리 기법을 제안하고 설계한다. 또한 시스템 로그 분석 결과를 통한 서비스 부하 모델을 근간으로 제안한 메커니즘의 효율성을 검증한다.

키워드 : 유비쿼터스 컴퓨팅, 오토노믹 컴퓨팅, 자가관리, 결함 예방

Abstract A ubiquitous computing system could be constructed not only with one computer but with networks of computers (or other devices with computing power) embedded in everyday objects. For this, dependability must be guaranteed for each single component of a system and for the whole system which might be more than just a sum of its components. This paper proposes an autonomic self-management mechanism that can operate for 24 hours/day with the minimum human intervention. In addition, we validate the autonomic fault management scheme based on a workload model derived from the system log analysis.

Key words : Ubiquitous Computing, Autonomic Computing, Self-management, Fault-prevention

1. 서 론

장비의 소형화 및 처리 능력 확대에 더불어 컴퓨팅 시스템을 구축하기 위한 비용이 절감되면서, 다양한 산업 분야에서 컴퓨팅 능력을 활용하고 있다. 더욱이 유비쿼터스 컴퓨팅 환경이 도래하면 언제 어디서나 컴퓨팅 자원을 사용할 수 있게 되고, 이를 위해 도처에 컴퓨팅 장치들이 탑재되어 유·무선으로 서로 연결된 네트워크를 통해 상호작용을 한다. 다시 말해서 사람, 컴퓨터와 사물이 통합된 환경에서 기능의 최적화를 위해 이질적인 시스템의 연계성을 제공하고자 하는 것이 유비쿼터스 컴퓨팅의 목표이다[1]. 하지만 모든 장치를 사용자 및 관리자가 관리할 수 없으므로 장치들은 스스로 관리할 수 있는 능력을 보유하여 사람의 간섭을 최소화하면서 1년 12개월 24시간 동안 동작할 수 있어야만 한다. 다시 말해서 한 시스템의 단일 컴포넌트 뿐만 아니라 전체 시스템에 대한 고가용성이 제공되어야만 한다. 또한 예기치 못한 결함을 감지하고 효율적으로 복구하기 위해서는 전체 시스템이 약연결(loosely coupled)되어야 하고 독립적으로 수행할 수 있는 최소한의 단위로 컴포넌트를 구분하고 모듈화하여 이들을 구조적으로 정의할 수 있어야만 한다. 하지만 OSGi, J2EE, Eclipse 등과 같은 플랫폼을 개발하기 위한 연구 뿐만 아니라 국내·외 대다수 관련 연구들은 이질적 환경의 기능적 통합 및 연결성(functional integration and connectivity)에만 치중하여 비기능성(high-availability and reliability) 관련 연구는 미미하여 기본적인 기술만 제공하고 있어 예기치 못한 결함 발생으로 인한 서비스 중단에 대한 대비책 마련이 시급하다[2,3].

시스템 결함은 하드웨어에 의한 결함과 소프트웨어에 의한 결함으로 구분할 수 있는데, 주로 예기치 못한 결

· 본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스컴퓨팅 및 네트워크 원천기반기술개발 사업의 지원에 의한 것임
† 학생회원 : 아주대학교 정보통신전문대학원
clchoi@ajou.ac.kr
†† 종신회원 : 아주대학교 정보통신전문대학원 교수
sskim@ajou.ac.kr
논문접수 : 2005년 1월 24일
심사완료 : 2005년 8월 12일

함은 소프트웨어에 의한 것이 대부분을 차지하며, 대표적인 결함 원인으로 소프트웨어 노화현상을 들 수 있다[4-6]. 소프트웨어 노화현상이란 소프트웨어에 잠재적으로 존재하는 것으로 메모리 유출(memory leakage), 버퍼 오버플로우, 데이타 원형 손상, 수학적 에러 누적 등으로 인해 데이타의 유실, 통신 장애 및 서비스 불가능을 야기시키는 현상이다. 이에 대한 대안으로 결함 예방(fault prevention) 기술인 소프트웨어 재활 기법이 최근 연구되고 있는데, 이는 심각한 시스템 오류를 발생시킬 수 있는 상황을 분석 및 예측하여 사전에 시스템 내부 상태를 깨끗한 초기 상태와 같은 상태로 전환시켜 지속적인 서비스가 가능하도록 하는 일련의 과정을 포함하는 기술이다[7-11]. 여기서 내부 상태를 깨끗이 하는 과정이란 쓸데없는 자료 수집(Garbage collection), 운영체제 커널 테이블의 재정리, 내부 자료 구조의 초기화 등이다. 하지만 소프트웨어 재활 기술의 최근 관련 연구[7-11]들은 기술 적용에 따른 시스템의 가용도 변화에 대한 분석, 또는 결함에 따른 손실 비용 분석, 단일 시스템과 클러스터와 같은 동종 서비스를 제공하는 분산 시스템에 적용 가능성을 타진하는 정도로만 연구가 수행되었을 뿐 이질적인 컴퓨팅 환경인 유비쿼터스 컴퓨팅에 접목하기 위한 시스템 및 장치 관리 모듈의 구조 도출 및 관리 기능을 적용함에 있어 사람의 간섭을 최소화하기 위한 연구는 수행되지 못했다. 이로 인해 시스템을 관리하는데 있어 사람의 간섭을 최소화해야 되는 유비쿼터스 컴퓨팅 환경에 기존 연구들을 직접 접목하는 것은 부적절하므로 이를 해결하기 위한 오토노믹 컴퓨팅(autonomic computing) 관련 기술 개발이 필요하다.

오토노믹 컴퓨팅 기술이란 기존에 개발된 시스템 관리 기술을 자동화하여 시스템 관리 비용을 줄이기 위한 기술로써, 기본 개발 철학은 인간의 신체가 체온이 저하되면 열의 방출을 억제하기 위해 근육을 수축하고, 어두워 보이지 않을 경우 동공을 확장하는 것과 같이 뇌의 활동에 의한 명백한 사고없이 몸을 관리하는 자율신경계의 동작 원리를 컴퓨터 시스템 관리에 적용하자는 것이다[12]. 오토노믹 컴퓨팅을 위한 제어 루프 구조는 크게 4부분(M.A.P.E.)으로 나뉠 수 있는데, 자원의 상태 정보를 자동으로 수집하는 모니터링 부분(monitor), 수집된 정보를 근간으로 상황 분석을 수행하는 분석 부분(analyze), 분석된 자료를 바탕으로 사용자의 요구사항 및 서비스 수준 계약에 명시된 비기능적 요소를 만족하기 위한 정책을 계획하는 부분(plan), 수립된 계획에 준하여 적절한 조치를 수행하는 부분(execute)이다. 따라서, 본 연구에서는 일시적인 결함 및 사람의 운영상 실수에 대처하여 중단없는 서비스를 제공할 수 있는 메커니즘을 제안하고 이를 M.A.P.E. 구조를 기반으로 각각 필요한 기법 및 동작원리를 설명한다. 또한 예기치 못한 결함이 발생하여 이를 해결하고자 하는 일련의 과정에서 사람의 개입을 최소화할 수 있도록 시스템 스스로 자원을 관리할 수 있는 기법을 제안한다. 또한 메모리 유출에 의한 결함을 자동으로 감지하고 관리할 수 있는 자원모델링을 통해 제안한 기법의 실효성을 살펴본다.

2. 오토노믹 결함 관리 메커니즘

DMTF(distributed management task force)에서 권고하는 CIM(common information model) 표준은 관리를 위해 시스템 간에 많은 정보를 교환해야 되는 컴퓨팅 환경에서 사용 가능한 정보를 체계적으로 설계하고 구조화하기 위한 효율적인 대안이다[13]. 또한 모델링을 하는데 있어 균등한 방식을 채택하고 있는데, 기본적으로 객체지향 개념에서 착안한 시나리오를 준수하며, 다중 환경에서 객체 지향 스키마의 협업적인 개발을 지원한다. 따라서 본 연구에서 제안하는 기법을 유비쿼터스 컴퓨팅 환경에 적용하기 위한 모델을 CIM과 UML(unified modeling language)로 설명한다. 그림 1에서 하나의 사각형으로 표현된 것은 시스템 관리를 위해 필요한 개념을 표현하기 위한 클래스이다. 기본적으로 클래스에는 클래스 이름, 정의를 위해 최소 필요한 자료구조, 각 클래스를 동작시키기 위한 연산들이 포함된다. 또한 각 클래스를 연결하는 선은 클래스간 관계를 표현하기 위한 것으로 다중성에 대한 정의도 포함되는데, 예를 들면, "하나의 의존도 관리자는 하나 이상의 다중 관찰 관리자를 제어한다.; 하나의 관찰 관리자는 하나의 의존도 관리자에 의해서 관리된다."와 같은 의미적인 표현을 내포하고 있다.

오토노믹 결함 관리 메커니즘은 크게 두 부분으로 나뉘는데, 하나는 관리를 위한 계획 수립과 실행을 제어하기 위한 의존도 관리자(dependability manger)이며, 다른 하나는 모니터링과 분석을 수행하기 위한 관찰 관리자(observation manager)이다(그림 1 참고).

2.1 의존도 관리자

의존도 관리자는 자율적인 결함 관리를 위한 메커니즘의 중추 역할을 담당하며, 유비쿼터스 컴퓨팅 시스템에 소프트웨어 재활 기술을 적용하기 위한 기본 모델이다. 따라서 사전에 정의된 서비스 계약 수준을 만족하기 위한 응용 프로그램과 전체 시스템의 품질을 평가하고, 필요하다면 발생한 문제를 해결하기 위한 메커니즘을 작동시킨다. 또한 계획을 수립하거나 방안을 적용시키기 위해서 관찰 관리자와 협업한다. 이 협업을 통해 자원의 상태에 대한 지시자를 생성하고 모델에 통합된 각 객체의 특성을 분석하는데, 기본적인 정보는 다음과 같다.

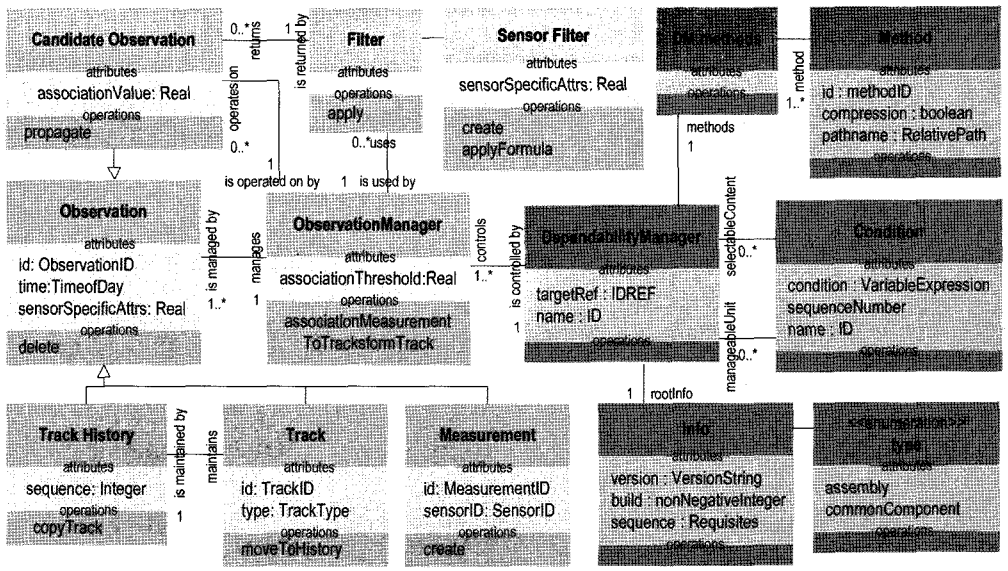


그림 1 오토노믹 결함 관리 메커니즘을 위한 UML 기반 설계도

- **조건(condition):** 유비쿼터스 컴퓨팅 시스템 내에 정의된 컴포넌트의 상태를 표현하기 위한 일반적인 지시자로써 이진(boolean) 표현 방식으로 표기하며 조건이 거짓일 경우 의존도 관리자는 해당하는 메커니즘을 동작시킨다.
- **기법(method):** 고유의 인식자(identifier)를 가지며, 수행해야 하는 일련의 작업들로 주어진 요구사항을 만족하기 위해서 코드를 실행한다.
- **정보(Info):** 시스템에 설치된 컴포넌트의 버전과 시스템 정보에 대한 특성을 묘사하고, 소프트웨어 요구사항을 만족하기 위한 플랫폼에 대한 정보 또한 포함한다.

2.2 관찰 관리자

관찰 관리자는 유비쿼터스 컴퓨팅 시스템의 상태에 대한 지시자를 생성하는 역할을 담당하며, 이를 위해 관련된 측정값을 수집, 통합, 필터링, 추적하는 기능을 수행하며, 오토노믹 컴퓨팅 기술의 모니터링과 분석 부분과 동일한 역할을 담당한다.

- **필터(filter):** 모든 센서 필터에서 필요한 기본 정보를 제공하기 위한 클래스로 필터에 의해서 정의된 추적과 측정을 수행하여 정보를 생성한다.
- **측정(measurement):** 관찰 관리자가 수집하게 되는 정보로써 하나의 센서 또는 센서들간의 통신 인터페이스에서 제공된 결과값으로 다양한 센서와 인터페이스가 존재하므로 범용적으로 사용할 수 있는 부분에 대해서만 정의한다.
- **관찰(observation):** 결함 관리 영역에서 관리되는 객체 데이타로써 후보 관찰(candi-date observation)은 필터의 기준에 부합되는 각 추적과 측정을 위해 각각 생성된다.
- **추적(track):** 추적 객체의 상태 자료로써 특정 추적과 관련된 과거 정보를 기록한다.

2.3 M.A.P.E 구조

2.3.1 모니터링 부분(monitor part)

예측하지 못한 결함을 감지하기 위한 자료를 수집하는 부분으로 소프트웨어의 장시간 작동에 의해 발생하는 소프트웨어 노화현상[3]인 메모리 부족, 버퍼 오버플로우, 수학적 에러 누적 등을 감지하기 위해 필요한 상태 정보를 수집한다. 이 중 메모리 유출에 의한 메모리 부족 현상으로 인한 결함을 감지하기 위해 필요한 상태 정보를 표 1과 같이 정의한다. 또한 특정 시스템에서 발생할 수 있는 결함에만 국한되지 않고 각 시스템 및 장치의 관찰 관리자간 협업을 통해 전체 시스템 자원의 전반적인 상태변화를 수집한다. 표 1은 메모리 유출을 검출하기 위해 자원의 상태를 감시해야 할 객체(object)와 수집한 정보의 의미(report)를 정리하였으며, 메모리 유출이 발생하였을 경우 해당 측정값의 변화(change)를 보여주고 있다. 즉, 결함이 발생하여 시스템에서 적절한 조치를 필요로 하는 상황이라 결론짓기 위해서 필요한 정보를 시스템 및 자원으로부터 선별·수집하는 것이다.

2.3.2 분석 부분(analyze part)

수집된 정보를 근간으로 결함 발생 여부에 대해 분석

표 1 자원 객체와 메모리 유출시 변화

Object\Counter	Report	Change
Memory\Available Bytes	available bytes	fall
Memory\Committed Bytes	the private bytes committed to processes	rise
Process(process_name)\Private Bytes	bytes allocated exclusively for a specific process	rise
Process(process_name)\Working Set	the shared and private bytes allocated to a process	rise
Process(process_name)\Page Faults/sec	the total number of faults (hard and soft faults) caused by a process	rise
Process(process_name)\Page File Bytes	the size of the paging file	rise

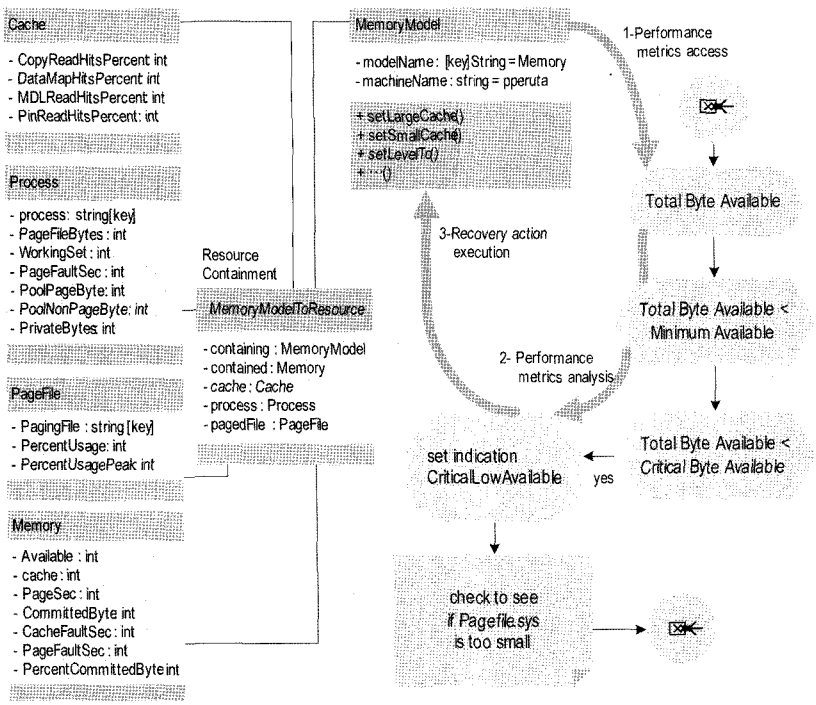


그림 2 메모리 유출 상황 분석을 위한 자원모델링

하기 위해 결함 보고의 수, 유형, 심각성에 대해 테스트해야 되고 결함 보고 비율 및 밀도(단위 크기당 문제 보고 수)를 계산하여 결함에 대한 추세를 분석해야 한다. 이를 위해서는 객체지향 패러다임에서 사용하고 있는 구조적인 기술과 개념화 기술을 적용하여 필요로 하는 정보를 구성하고 이에 대한 해석을 수행하며, 결함 발생 근원지를 찾아낼 수 있어야 한다. 그림 2는 메모리 유출 현상에 의한 결함 발생 여부를 분석하기 위한 자원모델링으로 메모리의 사용가능 용량(Total byte available)이 한 프로세스가 실행되기 위해 필요한 메모리 용량(Minimum available)보다 작으면 결함이 발생할 수 있는 상황으로 규정한다.

2.3.3 계획 부분(plan part)

시스템 구성을 위해 적용된 토폴로지는 유비쿼터스 컴퓨팅 환경에서는 시간 및 요구사항에 따라 변화할 수 있어야만 한다. 따라서 토폴로지에 사용된 컴포넌트(component, COM)와 각 컴포넌트에서 사용하고 있는 기반구조의 서비스(service, SVC)를 변화에 따라 적절히 매핑할 수 있어야만 한다. 이를 위해 이질적인 환경을 통합할 수 있는 미들웨어(예, OSGi, J2EE 등)에 대한 개발이 활발히 진행되고 있는 것이다. 하지만 예기치 못한 결함이 매핑 과정에서 일어날 수 있으므로 새로운

컴포넌트를 개발하거나 기반구조의 서비스를 추가할 경우에는 기존 시스템에 미칠 수 있는 영향에 대한 분석을 수행해야 한다. 이를 위해서 전통적인 결함주입기법(fault injection technique)이 사용될 수 있으며, 이를 각 부분별로 표현하면 그림 3과 같다.

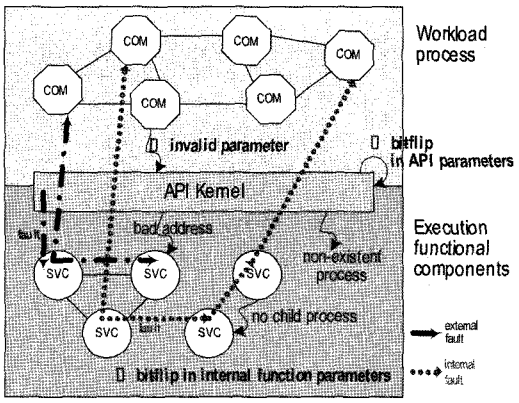


그림 3 컴포넌트 기반 시스템의 결함주입기법

결함주입기법에 의해서 한 컴포넌트 및 서비스에 결함이 발생하였을 경우 미칠 수 있는 영향에 대해 분석하고 이를 근간으로 결함이 발생하였을 경우 복구를 수행해야 되는 부분을 결정할 수 있는 정책을 결함 트리 형태로 정의하는데, 각 서비스를 제공하기 위해 필요한 최소한의 자원에 대해서도 결정한다. 예를 들어, 유비쿼터스 컴퓨팅의 대표적인 응용분야로 개발되고 있는 Follow-me 응용 서비스의 경우 사용자의 위치에 따라 사용자가 실행시킨 응용프로그램이 해당 위치의 컴퓨터로 이동하는데 필요한 컴포넌트(Follow-me Application)를 컴포넌트간 상관 관계 규명을 통해 결함에 대한 영향을 분석하고 각 컴포넌트 및 서비스에서 필요로 하는 최소자원요구량을 결함 트리 형태로 도식화하면 그림 4와 같다.

2.3.4 실행 부분(execute part)

소프트웨어 노화현상에 의해서 결함이 발생하였을 경우 계획부분에서 선정된 정책에 따라 각 컴포넌트 및 서비스를 복구하는데, 향후 심각한 시스템 장애를 방지하기 위해서 시스템의 내부 상태를 청소하여 정상적인 상태에서 서비스가 가능하게 하는 소프트웨어 재활기법을 사용한다. 하지만, 소프트웨어 재활 기법은 커널 테이블의 재정리 또는 메모리 플러싱과 같은 비교적 짧은 복구 시간을 요구할 때도 있지만, 응용 서비스를 위한 소프트웨어 및 미들웨어를 재시작해야 되는 경우도 있으며 최악의 경우 해당 시스템을 재부팅해야만 할 때도 있다. 이와 같은 경우 복구를 위한 시간으로 인해 장시간 서비스가 중단되어 사용자의 불만족을 야기할 수 있다. 따라서 수행 중이던 작업을 여분서버로 인계시켜 지속적으로 서비스를 수행하도록 한 후 결함이 발생한 주서버를 소프트웨어 재활을 수행하면 사용자 측면에서는 결함 발생 여부를 인지할 수 없게 된다. 따라서 주서버의 오동작이 발생할 것으로 예상되면 주서버에서는 주위 컴퓨팅 자원을 확보하여 작업을 인계하도록 한다. 또한, 여분서버의 역할은 주서버가 소프트웨어 재활에 의해 치유되고 있는 동안 임시적으로 서비스를 하는 것이므로 주서버가 다시 정상상태로 치유되면 주서버가 계속 서비스를 수행하며, 이를 위한 적용 방법은 3장에서 자세하게 설명한다.

3. Automation scheme

하드웨어 및 환경적인 요인에 의한 에러, 시스템 에러, 사람의 오작동에 의한 에러 중 2장에서 제안한 모델에서 주로 고려한 에러에 대한 정의를 도식화하면 그림 5와 같다.

시스템이 예기치 못하게 서비스가 불가능해지는 경우(이벤트: emergency stop)는 하드웨어 및 환경적인 요인에 의한 에러, 시스템 에러, 사람의 오동작에 의한 에러 중 하나가 발생하였지만 그것을 검출하지 못하여 그에 대한 대응책이나 복구를 수행하지 못하는 경우이다. 또한 유비쿼터스 컴퓨팅에 자가 관리 능력을 부여하기 위해 2장에서 제안한 모델에서 소프트웨어에 의한 결함에 집중하여 설명하였는데, 이는 유비쿼터스 시스템 장

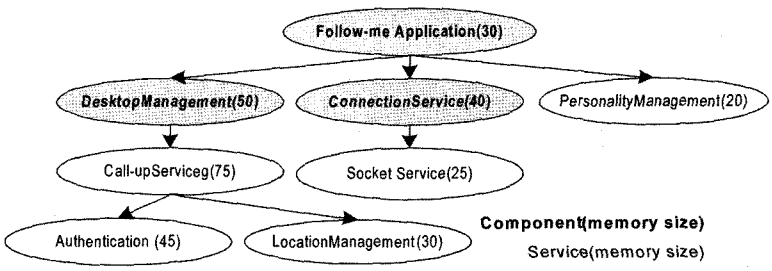


그림 4 결함 트리 및 최소자원요구량

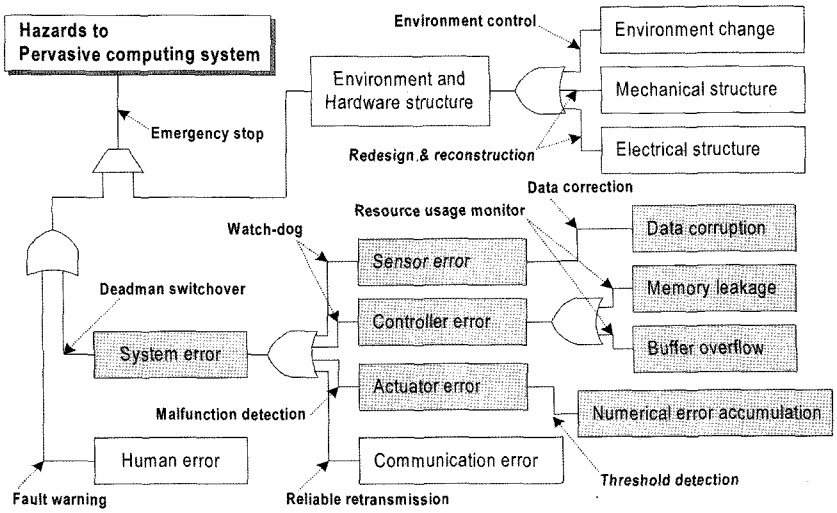
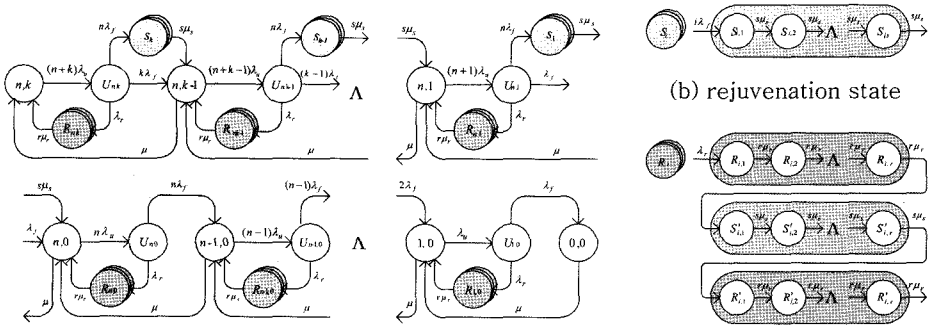


그림 5 결함 분류와 발생 가능한 이벤트

치에서 발생한 에러에 속하며, 시스템 에러는 세부적으로 센서 에러(sensor error), 제어기 에러(controller error) 및 실행기 에러(actuator error)로 분류할 수 있다. 따라서 2.3.1과 2.3.1절에서 제안한 자동적인 모니터링과 분석을 위한 기본 모델은 각 자원과 이를 관리하기 위한 제어기와 실행기에 더불어 이를 활용하는 각 컴포넌트에 대한 참조모델로 확장해야만 한다. 관찰 관리자는 참조모델을 통해 예측하지 못한 결함 현상에 대한 조건들을 검출하는데, 이는 운행시 발생한 문제를 해결하기 위한 최적(best-practices) 알고리즘인 소프트웨어 재활 기법에서 필요한 자료를 수집하고 해석한다. 소프트웨어 재활 기법은 시스템 상태를 초기화하는 것으로 운영체제 커널 테이블의 재정리, 내부 자료 구조의 초기화, 메모리 플러싱(flushing), 시스템 재부팅(reboot) 등과 같은 방안이 기존 연구[7-11]에서 개발이 되었다. 하지만, 기존 연구에서는 소프트웨어 노화 현상을 규명하는 것에 초점을 맞추거나 소프트웨어 재활을 위해 사용할 수 있는 방안 모색에만 초점을 맞추었고, 이를 시스템 스스로 판단하고 실행할 수 있는 방안에 대한 연구가 미미하다. 따라서 유비쿼터스 컴퓨팅 환경처럼 사람의 개입을 최소화해야 되는 환경에서는 소프트웨어 노화 현상을 스스로 진단하고 필요시 재활 기법을 스스로 동작시킬 수 있도록 시스템 상태 전이 모델 및 참조 모델을 근간으로 시스템을 개발하여 시스템 스스로 자신의 상태를 모니터링하고 분석하여 동작할 수 있는 능력을 제공해야만 한다. 예를 들면, 메모리 유출 현상을 검출하기 위해서 모니터링 부분은 메모리의 상태 변화를 지속적으로 측정할 수 있어야 하며, 분석 부분에서는 한 소프트웨어 컴포넌트에서 사용되고 있는 각 개별 프로세스의 특정 영역이 부족해지거나 필요 이상의 작업 집합(working set)으로 인해 가용한 메모리 공간이 부족해지는 현상이 발생하였을 경우 이에 대한 적절한 분석 및 검출을 수행한다. 이를 위해서는 과거 상태에서 수집된 정보를 근간으로 메모리 및 프로세스와 같은 시스템 관련 객체의 변화를 추적할 수 있어야만 한다[14]. 결국 추적 분석 결과 소프트웨어에 의한 결함 발생 확률이 높아지면 이에 대한 최적 대응책인 소프트웨어 재활을 통해 잠재적인 결함 발생 요인을 제거하는 것이다. 따라서 이를 위해서는 적절한 시스템 운영 모델 및 참조모델이 개발되어야 하며, 상태 전이에 대한 모델 또한 개발되어야만 하는데, 유비쿼터스 컴퓨팅 시스템의 고가용성 분석을 위한 상태전이 모델은 그림 6과 같으며, 기본적인 가정사항은 다음과 같다.

- 전체 시스템은 서비스를 제공하기 위한 n 대의 주서버로 연결되어 있으며, 각 서버는 서로 독립적으로 작업을 수행하고, 결함에 대응하기 위해 k 대의 여분서버가 존재하여 임의의 주서버에 결함이 발생할 경우 이의 역할을 대체하고 결함이 발생하거나 발생 확률이 높은 서버는 수리된다.
- 주서버와 여분서버의 고장률(λ_f), 수리률(μ)은 상수이며 결함 발생 및 수리 시간 간격은 지수분포를 따른다.
- 소프트웨어 재활 주기(λ_r)와 소프트웨어 노화현상으로 인해 서버의 상태가 불안정해지는 시간(λ_u)은 지수 분포를 따른다.
- 주서버에서 여분서버로 작업 전이에 필요한 시간은



(a) all state (b) rejuvenation state (c) switchover state

그림 6 가용도 분석 모델을 위한 유비쿼터스 컴퓨팅 시스템의 상태전이모델

일정하며, 작업 전이 시간은 평균값이 $1/\mu_s$ 인 r-stage Erlangian 분포를 따른다.

- 서버 상태가 불안정하여 재활을 수행해야할 경우 소프트웨어 재활에 필요한 시간은 일정하며, 재활 시간은 평균값이 $1/\mu_r$ 인 r-stage Erlangian 분포를 따른다.

각 시스템의 여분서버는 현 서비스를 제공하는데 포함되지 않지만 유비쿼터스 컴퓨팅 시스템 환경에는 포함되어 있는 것으로 주로 유휴(idle) 서버를 활용하면 되므로 자가 관리를 위해 추가적인 설치를 할 필요가 없으므로 전체 비용을 줄일 수 있다. 그리고 여분서버로의 작업 전이 시간과 재활에 필요한 시간을 r-stage Erlangian 분포로 가정한 것은 가용도 분석 모델링을 모든 상태가 memoryless 성질을 갖는 Markov 모델보다는 보다 현실에 근접한 가용도 계산을 가능케 하기 위해 semi-Markov 모델을 선택하기 위함이다. 또한 그림 6(a) 모델에서 정상 상태에서 가동되고 있는 서버는 (주서버수, 여분서버수)의 쌍으로 상태 변수{*(n,k)*, *(n,k-1)*, ..., *(n,0)*, *(n-1,0)*, ..., (1,0)}로 표현된다. 장시간 가동 후 소프트웨어 노화현상이 발생하여 시스템은 정상 상태에서 불안정 상태{$U_{n,k}$, $U_{n,k-1}$, ..., $U_{n,0}$, $U_{n-1,0}$, ..., $U_{1,0}$}로 전이하며, 불안정 상태에 있는 서버는 결함이 발생하여 작업 전이 상태{S_k, S_{k-1}, ..., S_1}로 들어가거나 소프트웨어 재활을 수행한다{$R_{n,k}$, $R_{n,k-1}$, ..., $Rn,0$, $R_{n-1,0}$, ..., $R_{1,0}$}. 불안정 상태에서 주서버에 결함이 발생하면(*with rate* $n\lambda_f$) 작업 전이 과정을 통해 k 개의 여분서버가 사용가능할 때까지 시스템의 가동 서버는 n 개로 유지되고 여분서버에 고장이 발생할 경우(*with rate* $k\lambda_f$) 작업 전이 시간이 불필요하므로 지연시간 없이 상태가 변한다. 주서버 중 $\lceil n/2 \rceil$ 수만 먼저 재활을 수행하는데 나머지 서버들이 수행하던 작업을 인계받은 후 수행하기 때문에 재활 가동으로 인한 서비스 중지는 발생하지 않는다. 또한 여분서버의 재활은 해당 클러스터 시스템의 재활 주기에 따라 이뤄지므로 재활 후 전체 시스템의 불안정성은 제거된다. 마지막으로, (0,0) 상태는 전체 시스템의 고장 상태를 나타내는데, 이는 예기치 못한 결함을 발생하였어도 서비스를 지속적으로 수행할 수 없는 상태로 주서버와 여분서버의 가동이 모두 중지된 경우이다.

하지만 그림 6(a)의 모델링은 작업 전이 상태와 재활 상태에서 머무는 시간이 일정(determinestic)하지 않으므로 무기억성(memoryless)을 만족하지 않는다. 따라서 이 모델링은 기약 재귀적(irreducible recurrent) Markov 모델에 속하지 않으므로 semi-Markov 프로세스 문제로 분류되며, 정확한 해를 구하는 간단한 방법이 존재하지 않는다. 하지만 주서버에서 여분서버로의 작업 전이가 일정 시간 내에 이루어져야 하는 조건을 만족시켜야 하는 시스템 운영 조건을 반영하는 모델을 해석하기 위해서 분할 기법을 통하여 작업 전이 상태를 s개의 부분작업 전이 상태로 나눌 수 있다(그림 6(b)). 이때, s를 무한대로 접근시키면 부분 작업 전이 상태는 단위 충격(unit impulse) 함수가 되며, 작업 전이 시간은 s개 지수분포의 합이 되므로 그림 6(a)는 Markov 프로세스로 해석될 수 있다[9,10,15]. 또한 재활 상태는 주서버가 먼저 재활을 하고 나머지 서버가 재활을 하며 중지 없는 서비스를 제공하기 위해 각 서버들 간 작업 전이가 이뤄지므로 그림 6(c)와 같이 모델링한다. 그리고 모든 상태가 안정 상태(steady-state)일 때의 균형 방정식(balance equation)을 얻는데 첨자 표현을 간략하게 하기 위해서 한 상태에 존재하는 정상으로 동작하는 총 서버수를 사용한다. 예를 들면, i=n+k라고 하면 모델에서 (n,k) 상태를 표현하며 i=n은 (n,0) 상태를 나타낸다.

$$(n+k)\cdot\lambda_u\cdot P_{n,k} = r\cdot\mu_r\cdot P_{R_{n,k}} + \mu\cdot P_{n,k-1}$$

$$(i\cdot\lambda_u+\mu)\cdot P_i = r\cdot\mu_r\cdot P_{R_i} + (i+1-n)\cdot\lambda_f\cdot P_{U_{i+1}}$$
$$+ s\cdot\mu_s\cdot P_{S_i} + \mu\cdot P_{i-1} \quad \{i=n, n+1, \ldots, n+k-1\}$$

$$(i\cdot\lambda_u+\mu)\cdot P_i = r\cdot\mu_r\cdot P_{R_i} + (i+1)\cdot\lambda_f\cdot P_{U_{i+1}} + \mu\cdot P_{i-1}$$
$$\{i=1, 2, \ldots, n-2, n-1\}$$

$$\mu\cdot P_0 = \lambda_f\cdot P_{U_1},$$

$$(\lambda_r + i\cdot\lambda_f)\cdot P_{U_i} = i\cdot\lambda_u\cdot P_i \quad \{i=1, 2, \ldots, n+k\}$$

$$r\cdot\mu_r\cdot P_{R_i} = \lambda_r\cdot P_{U_i} \quad \{i=1, 2, \ldots, n+k\},$$

$$r\cdot\mu_r\cdot P_{R_i} = s\cdot\mu_s\cdot P_{S_i} \quad \{i=2, 3, \ldots, n+k\}$$

$$s\cdot\mu_s\cdot P_{S_i} = r\cdot\mu_r\cdot P_{R_i} \quad \{i=2, 3, \ldots, n+k\},$$

$$s\cdot\lambda_s\cdot P_{S_i} = n\cdot\lambda\cdot P_{U_{i+1}} \quad \{i=n, n+1, \ldots, n+k-1\}$$

위의 균형 방정식과 각 상태에서 머물 확률의 총합이 1이 되는 보존(conservation) 방정식을 결합한 연립 방정식을 풀면, 시스템이 평형일 때, 각 상태에 머물 확률을 얻을 수 있다.

$$P_{R_i} = \frac{\lambda_r}{r\cdot\mu_r}\cdot\frac{i\cdot\lambda_u}{\lambda_r+i\cdot\lambda_f}\cdot P_i \quad \{i=1, 2, \ldots, n+k\},$$

$$P_{U_i} = \frac{i\cdot\lambda_u}{\lambda_r+i\cdot\lambda_f}\cdot P_i \quad \{i=1, 2, \ldots, n+k\}$$

$$P_{R'_i} = \frac{\lambda_r}{r\cdot\mu_r}\cdot\frac{i\cdot\lambda_u}{\lambda_r+i\cdot\lambda_f}\cdot P_i \quad \{i=2, 3, \ldots, n+k\},$$

$$P_{S'_i} = \frac{\lambda_r}{s\cdot\mu_s}\cdot\frac{i\cdot\lambda_u}{\lambda_r+i\cdot\lambda_f}\cdot P_i \quad \{i=2, 3, \ldots, n+k\}$$

$$P_{S_i} = \frac{n\cdot\mu}{(i+1)\cdot\lambda_s}\cdot P_i \quad \{i=n, n+1, \ldots, n+k-1\}$$

$$P_i = \left(\frac{\lambda_f\cdot\lambda_u}{\mu}\right)^{n+k-i}\cdot\prod_{l=0}^{n+k-1-i}\frac{(n+k-l)^2}{\lambda_r+(n+k-l)\cdot\lambda_f}\cdot P_{n+k}$$

$$\{i=0, 1, \ldots, n+k-1\}$$

기존 연구[9]에 정의한 가용도 분석은 시스템의 한 서버라도 가동이 가능하면 가용하다고 분석하였다. 하지만 이와 같은 정의는 수학적인 분석으로는 의미가 있지만 현실적으로 사용자가 느끼는 가용도로 해석하기에는 문제가 있다. 따라서 본 논문에서는 서비스 계약 수준에 명시된 성능 척도를 만족시키기 위해 필요한 최소의 수(n)가 가동되지 못하면 시스템은 가용하지 않다고 정의한다. 또한 재활을 수행하는 동안에도 처리하던 서비스나 대기하던 요청에 대한 유실은 없지만 성능 척도를 고려하면 가용하지 않은 상태이다. 따라서 가동 가능한 서버수가 n 이상일 때만 시스템이 가용함으로 차별화된 서비스 제공을 위한 전통적인 정의 방식에 따른 가용도는 아래와 같이 정의된다.

$$A = \sum_{i=0}^{k}(P_{n,i}+P_{U_{n,i}}) = 1-\left[\left(\sum_{i=2}^{n+k}P_{S'_i}+P_{R_1}+P_0\right)\right.$$
$$\left.+\sum_{i=1}^{n-1}(P_{i,0}+P_{U_{i,0}})+\sum_{i=2}^{n+k}\sum_{j=1}^{r}(P_{R_{i,j}}+P_{R'_{i,j}})+\sum_{i=n}^{n+k-1}\sum_{j=1}^{s}P_{S_{i,j}}\right]$$

이를 유비쿼터스 컴퓨팅 환경에 적합한 가용도로 재정의하기 위해 Δ_i를 i 개의 주요 서버가 동작하고 있을 때 전체 시스템의 서비스가 가용한지에 대한 여부를 판단하기 위한 지시자로 정의한다. 다시 말해서 $d \le D_u$ and $\Delta_i = 0$ 하다면 $\Delta_i = 1$이고 그렇지 않으면 $\Delta_i = 0$이다. 여기서 D_u는 사용자에 의해서 요청된 최대 대기시간을 의미하며, d는 전체 시스템에서 동작 가능한 서버의 수가 i 개일 때 평균 대기시간을 의미한다. 또한 초기 시스템 정의를 위해 기본적으로 트랜잭션의 도착 시간과 처리 시간은 지수 분포를 따른다고 가정하며, 평균율은 각각 $1/\alpha$, $1/\beta$로 정의하면, 가용도는 식 (1)과 같이 재정의할 수 있다.

$$PA = \begin{cases}\sum_{i=1}^{n-1}\Delta_i(P_{i,0}+P_{U_{i,0}})+\Delta_n\sum_{i=0}^{k}(P_{n,i}+P_{U_{n,i}}) \\ \Delta_n P_{n,0}\end{cases}$$
$$+\Delta_{n-1}(\sum_{i=2}^{n+k}\sum_{j=1}^{r}(P_{R_{i,j}}+P_{R'_{i,j}})+\sum_{i=n}^{n+k-1}\sum_{j=1}^{s}P_{S_{i,j}}) \quad \begin{matrix}if\ n>m_0 \\ if\ n=m\end{matrix}$$

$$P_{n+k} = \left[\begin{array}{l} 1+\sum_{i=0}^{n+k-1}\left\{\left(\frac{\lambda_f\cdot\lambda_u}{\mu}\right)^{n+k-i}\cdot\left(\prod_{l=0}^{n+k-1-i}\frac{(n+k-l)^2}{\lambda_r+(n+k-l)\cdot\lambda_f}\right)\cdot\left(1+\frac{\mu}{(i+1)\cdot\lambda_f}\cdot\left(1+\frac{2\cdot\lambda_r}{r\cdot\mu_r}+\frac{\lambda_r}{s\cdot\mu_s}\right)\right)\right\} \\ +\sum_{i=0}^{k}\left\{\left(\frac{\lambda_f\cdot\lambda_u}{\mu}\right)^{k-i}\cdot\left(\prod_{l=0}^{k-i}\frac{(n+k-l)^2}{\lambda_r+(n+k-l)\cdot\lambda_f}\right)\cdot\frac{\lambda_r+(n+i)\cdot\lambda_f}{(n+i)^2}\cdot\frac{n\cdot\mu}{(n+1+i)\cdot\lambda_s}\right\} \\ -\left\{\frac{n\cdot\mu}{(n+k+1)\cdot\lambda_s}+\left(\frac{\lambda_f\cdot\lambda_u}{\mu}\right)^{n+k}\cdot\left(\prod_{l=0}^{n+k-1}\frac{(n+k-l)^2}{\lambda_r+(n+k-l)\cdot\lambda_f}\right)\cdot\frac{\mu}{\lambda_f}\cdot\left(\frac{\lambda_r}{r\cdot\mu_r}+\frac{\lambda_s}{s\cdot\mu_s}\right)\right\} \end{array}\right]$$

또한 서비스 수준 계약을 만족하기 위해서 서버 처리 용량, 메모리, 네트워크 대역폭, 저장장치 등 다양한 자원 관리가 필요하지만 본 논문에서는 처리 용량 중 제공 가능한 최대 서비스 연결수를 성능 척도로 고려한다. 왜냐하면 기존 여러 연구에서 증명하였듯이 서버 부하를 측정하기 위한 매개 변수 중 최대 서비스 연결수가 대표적인 지시자(indicator)가 되기 때문이다[16,17]. 그리고 성능 분석을 위해 서비스 도착률($1/\alpha$)과 처리 시간($1/\beta$)은 상호 독립적이며, 지수 분포를 따른다고 가정한다. 먼저 서비스 수준 계약에서 정의한 지연 대기(waiting time)의 마감 시간(deadline)을 만족하기 위해 필요한 최소 서버수(m0)를 결정하는데, 이것은 위의 가정을 따르면 *Erlang's C formula*, $C(m_0, \alpha/\beta)$[15]에 얻은 평균 지연 대기 함수로부터 얻을 수 있다.

$$W_q(t) = \frac{\left(\frac{(m_0\rho)^{m_0}}{m_0!}\right)\frac{1}{1-\rho}}{\left[\sum_{k=0}^{m_0-1}\frac{(m_0\rho)^k}{k!} + \left(\frac{(m_0\rho)^{m_0}}{m_0!}\right)\frac{1}{1-\rho}\right]} \quad (2)$$

서버수가 m_0일 때 임의의 대기 시간을 나타내는 변수를 W_q, 서비스 수준 계약서에 명시된 지연 대기 마감 시간을 d, 지연 대기 마감 시간을 만족하는 비율(Ψ)이라고 하면 사용자 클래스별 요구사항에 맞게 시스템을 구성하기 위한 조건은 아래와 같다.

$$W(d) = P[W_q \le d] \ge \psi \quad (3)$$

또한 유비쿼터스 컴퓨팅 시스템에 참여하고 있는 모든 서버의 부하가 과중하여 여분 서버를 획득하기 어려울 경우에는 서비스를 제공하기 위한 프로세스에 우선순위를 두어 우선순위가 낮은 서비스를 제공하고 있는 서버를 여분 서버로 정의한다. 다시 말해서 우선순위가 낮은 서비스가 서비스 수준 계약을 일정 기간동안 만족하지 못하더라도 우선순위가 높은 클래스의 서비스 계약을 먼저 준수하기 위해 시스템 재구성을 수행한다. 이를 수식화하기 위해 우선순위가 높은 특정 서비스를 t 시간에 제공하기 위한 서버수를 $S(t)$, 가용도를 $A_s(t)$, 한 서버의 최대 서비스 연결수를 $M_s(t)$라 하고, t-1 시간에 총 서버 부하를 $L_s(t$-1)라고 하며, 가용도 요구사항을 Ω라고 하면 식 (4)와 같이 서버수를 조정한다.

$$L_s(t) > S(t)\cdot M_s(t\text{-}1)\text{이거나 } A_s(t) < \Omega\text{이면 } S(t) = S(t\text{-}1) + 1 \quad (4)$$

4. Performance evaluation

본 장에서는 제안한 적용 기법의 적용 가능성을 타진하고 실효성 여부를 판단하기 위한 실험을 수행하는데, 웹 서비스에 대한 사용자 요청 모델(http://www.cgiserver.net)을 토대로 유비쿼터스 컴퓨팅 시스템의 서비스를 모델링하였는데, 서비스 요청수의 평균과 표준편차를 일별 비교를 해보면 거의 비슷한 양상을 보인다. 이는 양일간 표준 편차가 각각 3.36, 1.14, 7.95로 적기 때문이다(표 2 참조). 즉, 서비스를 시작하면서 발생한 로그파일의 분석을 통해서 향후 발생할 사용자의 패턴을 예측하여 보다 효율적으로 유비쿼터스 컴퓨팅 시스템을 관리할 수 있음을 보여준다.

표 2에서 얻은 평균 서비스 요청수를 해당 시간으로 나누어 평균 서비스 도착률을 설정하고, 웹 서비스 트래픽 패턴의 다양성과 자기 유사성(Self-Similarity)[18]은 Pareto 분포와 같은 Heavy-Tailed 분포에 따라 디자인한다. 서비스 처리률은 정적 서비스의 경우 파일의 크기에 비례한다고 가정하고 동적 서비스의 경우 Hyper-exponential 분포를 따른다고 가정한다. 또한, 시스템 운영 파라미터는 기존 연구[8-11]에서 정의한 기본값을 근간으로 설정한다. 한 서버의 고장률은 1년에 두 번 발생하며, 수리를 위해선 12시간이 필요하다. 재활 메커니즘을 수행하는 주기는 한달에 한번이며, 소프트웨어 노화현상이 발생할 시점은 서비스 가동 후 15일이다. 작업전이와 소프트웨어 재활에 필요한 시간은 각각 10분, 20초이며, 한 서버에서 서비스 가능한 트랜잭션 수는 1200개이다. 또한, 서비스 수준 계약에 명시된 정적 서비스 응답 시간과 동적 서비스의 첫 응답시간은 최대 1분이어야 하며 이는 네트워크에 의한 지연은 제외한 시간이고, 시스템에 도착한 요청 중 90% 이상이 해당 계약을 준수해야 한다. 또한 차별화된 서비스 제공을 위한 자가치유 시스템으로 설계하기 위해 99.99%의 가용도를 유지해야 한다. 마지막으로 작업전이와 재활 상태에 머무는 시간을 일정하게 하기 위해 20개의 단계로 각각 나눈다.

그림 7, 8, 9는 2장 및 3장에서 분석, 정의한 시스템

표 2 웹 서비스 근간 서비스 요청수의 평균과 표준편차

Periods	Mean number of requests		Standard deviation		Average	Standard Deviation
	2004/1/22	2004/1/23	2004/1/22	2004/1/23		
0:00 - 8:00	31.50	26.75	19.71	19.40	29.13	3.36
8:00 - 16:00	135.75	137.75	42.52	44.39	136.75	1.41
16:00 - 24:00	93.37	82.00	15.88	16.23	87.63	7.95

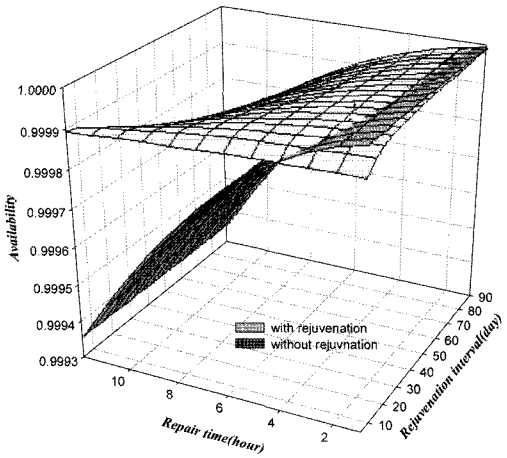


그림 7 소프트웨어 재활이 가용도에 미치는 영향

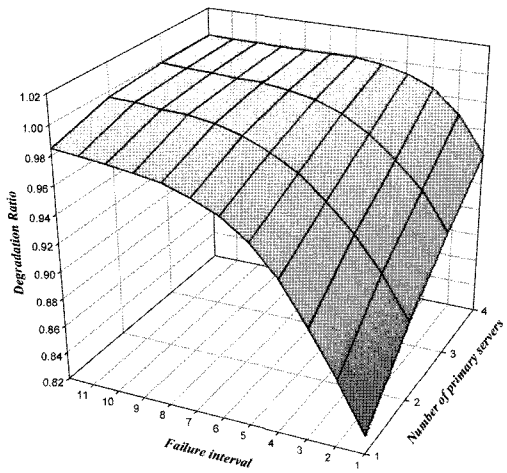


그림 8 시스템 성능 저하률

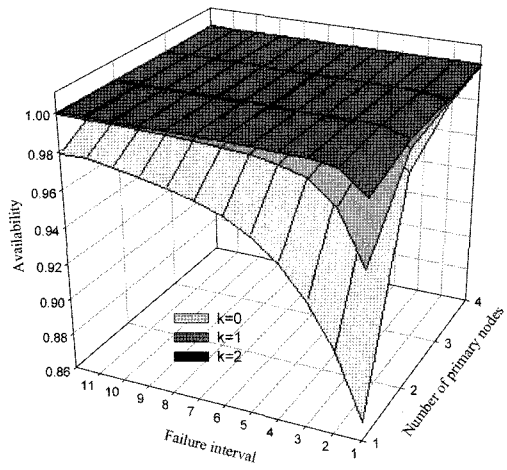


그림 9 여분서버 수에 따른 가용도 분석

모델을 가지고 실험을 통해 얻은 가용도 분석 결과 및 한 서버의 결함 발생이 전체 시스템 성능에 미치는 영향을 보여준다.

그림 7은 소프트웨어 재활 메커니즘의 장착 여부에 따라 시스템 가용도에 미치는 영향을 분석한 것이다. 결함이 발생한 서버를 수리하는데 필요한 시간이 적으면 적을수록 두 시스템 모두 가용도가 증가하지만, 소프트웨어 재활이 가능한 시스템은 긴 수리 시간이 필요한 경우에도 일정 수준의 가용도를 유지하는 것을 볼 수 있다. 그러나 재활을 수행하는 시스템일지라도 수리시간이 매우 길고 재활 주기가 길면 결함허용 시스템을 위해서 만족해야 하는 가용도 기준(99.99%)을 만족하지 못한다. 따라서 초기에 시스템을 구성하여 제공하고자 했던 성능을 유지하기 위해선 소프트웨어 재활 메커니즘과 같은 가용도 향상을 위한 메커니즘을 갖추고 있어야 하고, 재활 주기와 같은 장착한 메커니즘 관리를 위한 시스템 운영 파라미터도 적절하게 설정을 해야 한다.

그림 8은 소프트웨어 재활 메커니즘을 장착한 시스템에서 서비스 중단을 막기 위해 여분서버 및 잉여자원을 확보하지 못한 경우이다. 왜냐하면 유비쿼터스 컴퓨팅 환경은 컴퓨팅 자원(서버, 디바이스, 사람)이 유동적이므로 시간 및 상황에 따라 사용할 수 있는 여분 자원이 매우 가변적이기 때문이다. 하지만, 복구를 위한 여분자원을 확보하지 못하면 서비스를 지속적으로 수행하도록 여분서버에게 작업을 인계하지 못한 상태에서 소프트웨어 재활을 수행하여 복구 시간 동안 서비스가 불가능해진다. 이와 같은 경우 시스템에 발생한 결함이 전체 시스템의 처리량(throughput) 또는 응답시간에 영향을 미치게 된다. 따라서 본 논문에서 제안한 메커니즘을 탑재하였더라도 재활을 위한 여분 자원을 확보하지 못한다면 목표로 하는 가용성을 유지하기 어렵다. 이를 정량적으로 분석하기 위해 서버에 결함이 발생하여 서비스를 제공하지 못하기 때문에 발생하는 시스템의 전체 성능 저하률을 정의하면 성능 저하률(degradation ratio)은 (현재 제공되는 서비스 연결수)/(총 서버수 × 한 서버에서 제공가능한 연결수)로 계산할 수 있다. 결과에서 보듯이 소프트웨어 재활 메커니즘을 사용해도 여분서버의 수를 확보하지 못하면 재활을 수행하려는 서버의 작업을 인계받을 서버가 없으므로 요구하는 수준의 가용도를 유지하기 어렵고 이는 전체 시스템 성능에 영향을 미칠 수 있다는 것을 알 수 있다. 따라서 여분 자원을 확보하지 못한 상태에서 발생한 주서버의 결함이 시스템의 전체 성능에 영향을 미치므로 여분서버의 필요성을 보여준다.

그림 9는 주서버(primary server) 수와 여분서버(k) 수의 변화에 따른 QoS 인지 시스템의 가용도 분석 결

과이다. 해당 결과는 주서버 수에 따른 여분서버 수를 시스템 스스로 결정하기 위해 활용되는 것이다. 하지만 특정 서비스의 질을 유지하기 위해 불필요하게 많은 서버를 여분 자원으로 사용한다는 것은 비용 낭비를 초래하게 되므로 최소로 구성할 수 있는 서버수를 결정해야만 한다는 것을 보여주는 결과이다. 여분서버 수가 하나도 없을 경우 가용도 요구사항을 만족하지 못하고, 시스템을 구성하기 위한 소프트웨어가 안정적이지 못하여 서버 고장률이 높아질수록 여분서버를 가지고 있더라도 가용도는 떨어지게 된다. 하지만 여분서버로 2대 서버를 확보할 수 있다면 고장률이 크더라도 시스템 요구사항을 만족할 수 있다. 따라서 제공하기 위한 서비스를 유형별로 나눠 각각의 처리를 위한 시스템을 구성하여 다른 시스템의 시스템 요구사항과 서비스 수준 계약에 영향을 미치지 않는 한도에서 시스템 간에 작업전이를 하여 전체 시스템의 가용도 및 성능을 향상할 수 있다.

그림 7, 8, 9에서 얻은 결과는 기존에 성능 척도를 배제한 가용도(CA) 정의와 달리 시스템에서 제공 가능한 성능 전부를 서비스할 수 있으면 가용하다고 하며 그렇지 못할 경우 가용하지 않다고 하는 가용도(PA)를 재정의할 필요성을 뒷받침하고 있으며, 그림 8의 분석 결과를 도출하기 위해서 정의한 성능 저하률과 동일한 의미로써 가용도를 분석한 결과는 그림 10, 11과 같다. 해당 결과는 서비스 요청율에 따라 주서버의 수를 결정하고 고가용도를 유지하기 위해 소프트웨어 재활 메커니즘을 활용할 경우 여분서버의 수를 어느 정도까지 보유해야 되는지에 대한 결정을 시스템 스스로 수행할 수 있도록 하기 위한 실험 결과이다.

그림 10은 식 (1)에서 정의한 가용도를 바탕으로 여분서버수와 서비스 요청율에 따른 가용도 분석 결과이며 주서버수는 4로 설정한 것이다. 또한 4.1절에서 표본 추출한 작업 부하 모델링을 통해 얻은 서비스 도착률에 대한 결과이다(1 period = 0:00 - 8:00, 2 period = 8:00 - 16:00, 3 period = 16:00 - 24:00). 결과에서 보듯이 주서버수가 4대일 경우 여분서버가 2대 이상이면 시스템 요구사항을 만족할 수 있는데 3대 이상으로 여분서버를 추가하는 것은 큰 이득을 얻을 수 없으므로 낭비가 된다. 따라서 주서버수에 따라 필요한 최소 여분서버수를 결정해야 한다.

그림 11은 차별화된 서비스 제공을 위한 유비쿼터스 컴퓨팅 시스템을 구성할 때 고려해야할 사항들을 종합하여 얻은 결과를 바탕으로 구성에 필요한 서버수를 결정하는 예를 보여준다. 이때 서로 다른 우선순위를 서비스하기 위한 가상의 두 시스템을 위한 총 서버수는 16대라고 설정한다. 9:00 - 16:00 사이에 한 서비스를 위한 시스템(vLan1)은 서비스 수준 계약에 명시된 사항을

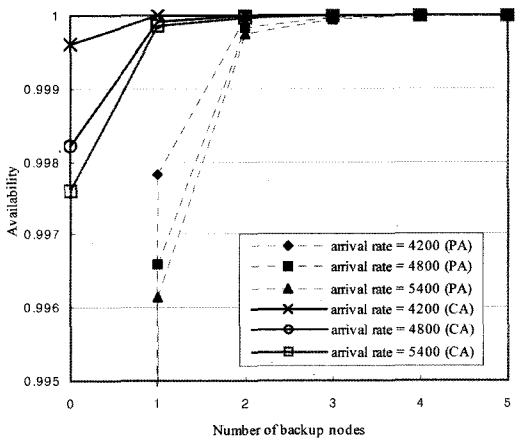


그림 10 서비스 요청률에 따른 가용도 분석

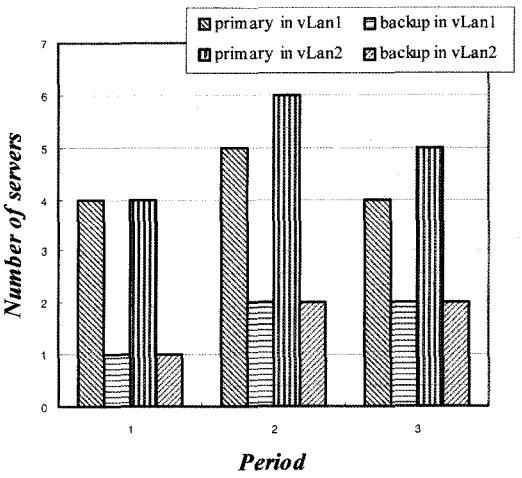


그림 11 주/여분서버수 변화

준수하기 위해서 주서버 5대, 여분서버 2대가 필요하며, 다른 서비스를 위한 클러스터 시스템은 주서버 6대, 여분서버 2대가 필요하였다. 이와 같이 필요한 주서버가 다른 이유는 한 서비스의 경우 I/O 집약적인 트랜잭션이 많기 때문에 상대적으로 적은 서버수 만으로 계약 준수가 가능하고 다른 서비스의 특징은 CPU 집약적인 트랜잭션이 많기 때문에 계약 준수를 위해서는 보다 많은 서버수가 필요한 것으로 판단된다.

5. 결 론

유비쿼터스 컴퓨팅은 동적으로 사용자에게 컴퓨팅 서비스를 제공해야만 한다. 이를 위해서는 언제, 어디서나 사용 가능해야 하므로 고가용도 요구사항을 만족해야만 한다. 더불어 사용자 측면에서는 서비스를 받고 있다 하더라도 응답시간과 같은 성능 측면의 요구사항이 만족

되지 못하면 시스템이 가용하지 못하다고 느끼는 사용자 편이 또한 고려해야만 한다. 이를 위해 본 연구에서는 가용도를 향상하기 위한 기존의 소프트웨어 재활기법을 유비쿼터스 컴퓨팅 시스템에 탑재하기 위한 기본 모델을 제안했다. 또한 제안한 모델의 실효성을 검증하기 위해 예기치 못한 소프트웨어 결함 발생 요인 중 대표적인 원인인 메모리 유출 현상을 검출, 분석하기 위한 자원 모델링을 통해 모델의 적용 가능성을 타진했다. 그리고 시스템의 자가 관리를 통해 사람의 개입을 최소화하기 위한 메커니즘을 M.A.P.E 제어 루프 구조에 근간하여 설계 및 모델링하고 제안한 메커니즘의 동작 원리 이해를 위해 실험을 통해 검증하였다. 이로써 사람의 간섭을 최소화하면서 시스템 운영 중에도 시스템 스스로 예기치 못한 결함 발생으로 인한 서비스 중단 요인을 제거할 수 있음을 보였다. 더불어 전통적인 개념으로 정의한 가용도 분석 모델링의 문제점을 제시하고 보다 유비쿼터스 컴퓨팅의 서비스 목표를 달성하는데 적합하도록 성능 측면을 함께 고려한 새로운 가용도에 대해 정의하였다. 향후 연구에서는 보다 다양한 응용 서비스에서 탑재할 수 있는 모델을 개발하고 이에 필요한 프로토타입 시스템을 구현할 것이다.

참 고 문 헌

[1] Satyanarayanan M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications (2001) 10-17.

[2] Sun_Microsystems: J2EE Platform Specification (2002) http://java.sun.com/j2ee/

[3] Microsoft: The Microsoft .NET Framework. Microsoft Press (2001).

[4] Garg S., Moorsel A., Vaidyanathan K., Trivedi K.: A Methodology for Detection and Estimation of Software Aging. Proceedings of 9th IEEE International Symposium on Software Reliability Engineering (1998) 282-292.

[5] Sullivan M., Chillarehe R.: Software Defects and Their Impact on System Availability-A Study of Field Failures in Operating Systems. Proceedings of 21st IEEE International Symposium on Fault-Tolerant Computing (1991) 2-9.

[6] Scott D.: Making Smart Investments to Reduce Unplanned Downtime. Tactical Guidelines Research Note TG-07-4033, Gartner Group (1999).

[7] Huang Y., Kintala C., Kolettis N., Fultion N.: Software Rejuvenation: Analysis, Module and Applications. Proceedings of 25th IEEE International Symposium on Fault-Tolerant Computing(1995) 318-390.

[8] Trivedi K., Vaidyanathan K., Popstojanova K.: Modeling and Analysis of Software Aging and Rejuvenation. Proceedings of IEEE 33rd Annual Simulation Symposium (2000) 270-279.

[9] Park K., Kim S.: Availability Analysis and Improvement of Active/Standby Cluster Systems using Software Rejuvenation. The Journal of Systems Software, Vol. 61, No. 2. (2002) 121-128.

[10] Choi C., Kim S.: Self-configuring Algorithm for Software Fault Tolerance in (n, k)-way Cluster Systems. Lecture Notes in Computer Science, Springer, Vol. 2667, No. 1. (2003) 742-751.

[11] J. Arlat, et al.: Dependability of COTS Microkernel-based Systems. IEEE Transactions on Computers, Vol. 51, No. 2. (2002) 138-163.

[12] Lanfranchi G., et al.: Toward a New Landscape of Systems Management in An Autonomic Computing Environment. IBM System Journal, Vol. 42, No. 1. (2003) 119-128.

[13] DMTF Inc.-Common Information Model. http://www.dmtf.org/standards/cim/

[14] 최창열, 김성수, "유비쿼터스 컴퓨팅 환경의 자동적인 결함 허용 기법 설계", 2004년 한국정보과학회 추계학술발표대회, 한국정보과학회, 31(2), pp. 379-381, 2004. 10.

[15] Kleinrock, L.: Queueing Systems Volume I: Theory. Wiley (1975).

[16] H. Chen and P. Mohapatra, "Session-Based Overload Control in QoS-Aware Web Servers," IEEE INFOCOM 2002, pp. 516-524, June 2002.

[17] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli, "Web Switch Support for Differentiated Services," ACM Performance Evaluation Reviews, Vol. 29, No. 2, pp. 14-19, 2001.

[18] Crovell M. E. and Bestavros A.: Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, IEEE/ACM Transactions on Networking, Vol. 5, No. 6. (1997) 835-846.

최 창 열

1999년 아주대학교 정보통신공학과 졸업(공학사). 2000년 아주대학교 정보통신전문대학원 졸업(공학석사). 2002년~현재 아주대학교 정보통신전문대학원 박사과정. 관심분야는 유비쿼터스 컴퓨팅, 오토노믹 컴퓨팅, 고가용성 시스템, 미들웨어 등

김 성 수

정보과학회논문지 : 시스템 및 이론
제 32 권 제 7 호 참조