INBANCA기법을 이용한 웹 서버 장애 진단 및 복구기법

윤 정 미[†] · 안 성 진[††] · 정 진 욱[†††]

요 약

이 논문에서는 웹 서버 장애 항목을 정의하고, 장애를 진단하기 위한 규칙 및 복구 방법을 INBANCA 기법을 이용하여 제안하였다. 장애 항목으로 프로세스 장애, 서버 과부하, 인터페이스 장애, 구성 및 성능 장애를 정의하였으며, 각 장애 항목을 진단하기 위한 지식을 활성 네트워크기법을 적용하여 표현하고, 이를 시스템 레벨 장애 진단 생성규칙과 서비스 레벨 장애 진단 생성규칙으로 정형화하였다. 또한 사례기반의 웹 서버 장애 복구를 위한 복구 규칙 정의 및 사례 라이브러리 구축을 통하여 환경 변수에 적응적인 복구 메커니즘을 제안하였다. 그리고 제안한 장애 진단규칙 및 복구 메커니즘의 타당성을 증명하기 위한 장애 환경 구성 및 각 장애 환경에 대한 생성규칙 적용과 복구과정을 실험을 통하여 제시하였다. 이 논문에서는 기하 급수적으로 증가하는 웹 서버의 장애 관리를 위한 메커니즘을 제안함으로써 웹 서버 관리를 위한 관리자의 노력을 최소화하고, 지능적인 장애 관리를 위한 방법론을 제시하고자 한다.

Web Server Fault Diagnosis and Recovery Mechanism Using INBANCA

Jung-Mee Yun[†] · Seong-Jin Ahn[††] · Jin Wook Chung[†††]

ABSTRACT

This paper is aimed at defining items of fault, and then constructing rules of fault diagnosis and recovery using INBANCA technology for the purpose of managing the web server. The fault items of web server consist of the process fault, server overload, network interface fault, configuration and performance fault. Based on these items, the actual fault management is carried out fault referencing. In order to reference the fault, we have formulated the system-level fault diagnosis production rule and the service-level fault diagnosis rule, conjunction with translating management knowledge into active network. Also, adaptive recovery mechanism of web server is applied to defining recovery rule and constructing case library for case-based web server fault recovery. Finally, through the experiment, fault environment and applicability of each proposed production rule and recovering scheme are presented to verify justification of proposed diagnosis rules and recovery mechanism for fault management. An intelligent case-based fault management scheme proposed in this paper can minimize an effort of web master to remove fault incurred web administration and operation.

1. 서 론

1992년 웹이 첫 선을 보인 이후 2년 만에 웹 서비스가 인터넷을 독점했다고 할 정도로 사용자들 사이에서 돌풍을 일으키기 시작하였다[1]. 웹이 인터넷에 존재하는 많은 소프트웨어 가운데 하나이며, 여기에 사용되는 HTTP프로토콜도 TCP/IP프로토콜 스택의 응용계층에 해당하는 프로토콜에 지나지 않을 뿐인데도 불구하고 이제는 웹이 곧바로 인터넷을 의미할 정도로 그

† 준 회 원 : 성균관대학교 대학원 전기전자 및 컴퓨터 공학부
†† 종신회원 : 성균관대학교 컴퓨터교육과 교수
††† 종신회원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
논문접수 : 2000년 3월 10일, 심사완료 : 2000년 8월 22일

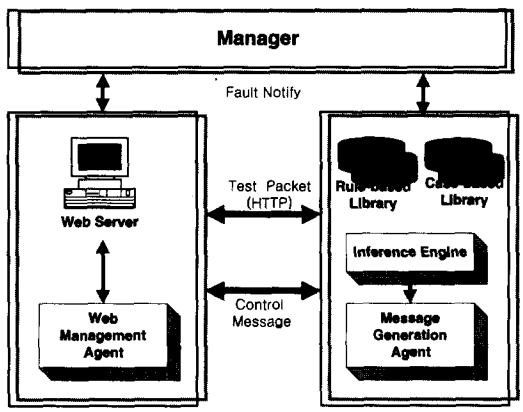
사용이 급격히 증가하고 있다. 이것이 가능하게 된 이유로는 여러 가지를 꼽을 수 있으나 크게 손쉽게 사용할 수 있다는 것과 웹 서버의 설치가 용이하다는 것을 들 수 있다. 그 결과 국내 현존하는 웹 서버의 개수만도 십 만개 정도에 이르나[2], 웹 서버의 증가와 비교하여 장애에 대한 적절한 대처가 이루어지고 있지는 않다. 이와 관련하여 웹 서비스 신뢰성을 향상시키기 위하여 웹 서버 성능 튜닝 기반의 다양한 방법론이 제안되고 있으나, 대부분의 연구가 웹 서버 자체의 성능 평가나 SNMP를 기반으로 한 자원 관리 중심으로 이루어지고 있으며[3-5, 12], 부분적으로 응용 서비스의 트래픽 패턴 분석을 통한 성능관리 및 장애 관리방법론에 대한 접근이 이루어지고 있는 실정이다[6]. 그러나 현재의 접근 방법은 웹 서버의 장애 관리에 대한 접근이라기보다는, 단순히 웹 서버의 성능 향상 위주의 관리기법들만 다루어지고 있는 실정이다.

따라서 이 논문에서는 응용 서비스, 특히 응용 트래픽의 80%정도를 점유하는 웹 서비스 장애 관리의 중요성을 인식하고 장애 관리를 체계적으로 수행하기 위한 장애 검출 규칙 정형화 및 복구 기법을 제안하고자 한다. 특히, 사례 기반 추론기법에서 전문가의 지식표현(Expertise)에 사용되는 INBANCA(INtegrating BAyesian Networks with CAse-Based Reasoning) 기법을 사용하여 지식을 체계적으로 표현함으로써 지능적인 장애 검출 및 복구 관리방법을 제시한다. 또한 실험을 통한 장애환경 조성 및 장애 규칙 진단을 통하여 실제 환경에서의 장애 진단 규칙 타당성을 평가하고, 복구 기법의 적합성에 대해서 살펴보도록 하겠다.

2. 시스템 구조 및 장애 항목정의

2.1 시스템 구조

웹 서버 장애 관리 시스템은 웹 서비스를 사용하는데 있어서 발생할 수 있는 장애를 검출하고 이를 환경 적응적으로 복구하는 시스템으로 전문가 지식을 체계화된 진단 규칙으로 표현하고, 실측 데이터와 장애 발생 파라미터의 사례베이스화를 통하여 적응적으로 장애를 추론한다. 이때 장애 진단 및 복구에 적용되는 시스템의 구조는 (그림 1)과 같이 이루어진다. 전체 시스템은 웹 서버가 설치되어 있는 시스템 내에 위치하여 웹 서버의 자원과 관련된 장애발생 여부를 모니터링하는 Web Management Agent(WMA)와 주기적으로 웹 서버로 서비스 요청 패킷을 보내고 이 검사 패킷에 대한 서버의 응답을 이용하여 서버의 서비스 제공과 관련된 장애발생 여부를 모니터링하는 Message Generation Agent(MGA)로 이루어진다.



(그림 1) 웹 서버 장애 관리 시스템의 구조

검사 패킷에 대한 응답으로 웹 서버는 HTTP 버전, 상태코드, 응답 구문으로 이루어진 상태라인을 포함하는 응답 메시지를 보내게 되며, 이때의 상태코드를 이용하여 서버의 상태를 진단하게 된다. 서버가 구동 가능한 상황에서 서버 장애가 발생하는 경우 에러 코드를 상태코드에 추가하여 보내게 되며, 이 응답 메시지의 분석을 통하여 MGA시스템에서의 장애 진단 추론 과정이 활성화된다.

2.2 웹 서버 장애 항목

응용서비스 분야에 있어서의 장애란 물리적인 장애뿐만이 아니라 서비스를 제공받는 사용자 관점에서의 장애까지를 포함하는 것으로서 서비스 제공이나 응답 지연 시간 등의 사용자 QoS를 지원하지 못하는 경우도 장애가 발생한 것이라 정의할 수 있다[7].

<표 1>에서 보는 바와 같이 웹 서버의 장애 항목은 크게 프로세스 장애, 서버 과부하, 인터페이스 장애, 구성정보 오설정으로 정의할 수 있으며, 각 장애 항목별로 발생하는 진단 증상을 도출할 수 있다. 첫번째 장애 항목인 프로세스 장애의 경우 이와 수반하여 발생될 수 있는 증상들로 프로세스 구동이 안 되는 경우와 하부 프로세스의 폭주가 발생하는 상황들을 들 수 있으며, 이러한 장애는 성능과 관련된 구성정보의 오 설

정으로 인한 원인들에 기인하여 발생될 수 있다. 다음으로 서버 과부하는 웹 서버 시스템 상태가 과부하가 되거나 서비스를 제공하는 서버 프로세스수의 폭주, 서비스 제공 시간의 증가 등의 장애 상황이 발생하는 것을 말하는 것으로 이 경우 웹 서버에서는 서버 장애 코드 중 Service Unavailable를 이용하여 서버의 장애를 알리게 된다. 관련하여 발생할 수 있는 장애 증상으로는 시스템의 CPU사용량 증가, 응답 시간 증가, HTTP 트래픽의 급증 등이 있을 수 있다. 세 번째로 인터페이스 장애는 내부적으로 웹 서버의 장애가 없는데도 불구하고 서비스를 제공하지 못하는 것으로 논리적으로 시스템의 인터페이스가 다운되거나 서버의 네트워크 오설정이 그 원인이 된다. 마지막 구성정보 오설정은 올바르지 못한 구성환경 설정으로 인하여 웹 서버의 성능 저하가 발생할 수 있으며 이는 장애를 초래하는 원인으로 작용할 수 있다.

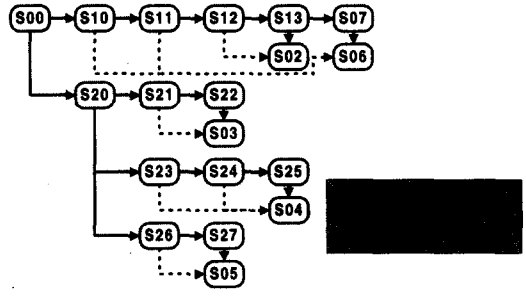
〈표 1〉 웹 서버의 장애 항목

장애 항목	발생 증상
프로세스 장애	Process Down(프로세스 다운)
	Flooding Chile Process(하부 프로세스 급증)
서버 과부하	Flooding Request(서비스 폭주)
	System Overload(시스템 과부하)
	Increasing Response Time(응답 지연시간 증가)
	Increasing Http Traffic(HTTP트래픽 급증)
인터페이스 장애	Routing Configuration Error(라우팅 구성장애)
	Interface Down(인터페이스 다운)
구성정보 오설정	Impossible Running Status(서버 구동 불가)
	Lowering of Performance(성능 저하)

3. 장애 진단 규칙 및 복구 기법

3.1 장애 진단 규칙

웹 서버의 장애 발생은 고정적인 요인보다는 다양한 환경 요인들에 의해 발생한다. 영향을 미치는 환경 요인으로는 시스템 자원 가용성, 서버 평균 부하, 서비스 요청 횟수 등 다양한 원인이 있으며 이러한 원인들이 복합적으로 작용하여 장애가 발생한다. 그러므로 응용 서비스 장애 관리를 하기 위해서는 시스템 의존적인 환경 요인들을 감안하여 장애 상황들을 검출할 수 있는 관리기법이 필요하며, 본 논문에서는 이러한 동적인 장애상황에 대처하기 위하여 사례기반추론(CBR : Case-Based Reasoning)기법을 사용한다. 사례기반 추론은 복잡한 문제를 해결하기 위해서 과거의 비슷한 사례를 추출하여 문제를 풀고자 하는 유사 추론, 혹은 경험적 추론을 하는 것에서 기인한 추론 기법으로 경험에 의해 스스로 발견된 지식을 이용하면서도 개념이 잘 정리되지 않거나, 수집 데이터의 편차를 이용한 추론 기법등 규칙추출이 어려운 분야에 주로 사용되는 추론 기법이다. 이 논문에서는 INBANCA(INtegrating BAyesian Networks with CAse-Based Reasoning)기법을 사용한 관리지식 규칙표현 및 추론을 기반으로 한다[8]. INBANCA는 확률 기반의 사례 지식표현에 사용되는 Bayesian Network을 네트워크 관리분야에 접목시킨 추론기법으로 규칙 결정 파라미터를 통한 상태천이를 기본으로 한다. INBANCA기반의 진단 규칙 표현을 위해서는 달성 목표의 정의, 활성 네트워크(Active Network)표현, 생성 규칙정의 및 수집 데이터의 사례 라이브러리화가 필요하다. 여기서 달성 목표란 진단하고자 하는 장애 유형들을 의미하는 것으로 (그림 2)에서 집합 {S01, S02, S03, S04, S05, S06, S07}이 이에 해당되며 이는 장애 진단의 최종 노드를 의미한다. INBANCA기법에서는 달성목표에 도달하기 위한 추론 과정을 활성 네트워크를 이용하여 표현하게 되는데 이때 달성 목표에 도달하기 위해서는 초기 진단 노드인 S00노드로부터 시작하여 상태 조건 만족 여부에 따라 활성 네트워크의 노드를 방문하며 달성목표 집합에 도달 시 장애 진단과정 이 완료되게 된다.



(그림 2) 장애 검출 및 복구 활성네트워크

각 상태 노드들에 대한 정의는 <표 2>와 같으며 각 상태들의 이동은 장애 진단규칙에 의하여 정의되어진다. 표에서의 α, β, γ는 수집 데이터에 의해서 설정되는 임계치를 나타내며, 상태 집합은 장애 항목을 표현하는 상태집합({S02, S03, S04, S05, S06, S07})과 장애 진단 과정을 표현하는 상태집합({S00, S01, S10, S11, S12, S13, S20, S21, S22, S23, S24, S25, S26,

S27))으로 나누어진다. 탐색 경로는 Bayesian Network 탐색과 동일하게 규칙 결정 파라미터에 의한 상태 값이 FALSE값을 가질 때는 장애 진단 아크를 따라서 이동하고, TRUE 값을 가질 경우에는 장애 복구 아크를 따라 이동하게 된다.

〈표 2〉 상태 정의

상태	정 의	분 류	달성목표
S00	Generation test packet	초기 상태	{S01, S02, S03, S04, S05, S06, S07}
S01	No failure	유효상태	Final normal state
S02	Network failure between MGA and WS	네트워크 장애 발생	Final fault state
S03	Internal server error	응용 서비스 장애	Final fault state
S04	Server overload	서버 자원 고갈	Final fault state
S05	DNS failure	DNS 장애	Final fault state
S06	Inoperable http daemon	웹 서버 구동 장애	Final fault state
S07	Process fault	응용 서비스 장애	Final fault state
S10	No response(HTTP)	응용 계층 장애	{S02, S06, S07}
S11	Response time> α	응답 지연 시간 초과	{S02, S06, S07}
S12	Non exist http daemon	구성 설정 장애	{S02, S06, S07}
S13	No response(ICMP)	링크 레벨 장애	{S02, S06, S07}
S20	Include error code in status line	서비스 레벨 장애	{S03, S04, S05, S22, S25, S27}
S21	Include configuration error	구성 설정 장애	{S03, S22}
S22	Contain CGI error in log file	응용 서비스 장애	{S03, S22}
S23	Excess maximum process	서비스 폭주	{S04, S25}
S24	Resource usage of http daemon process> β	서비스 레벨 장애	{S04, S25}
S25	Resource usage of entire system process> γ	시스템 레벨 장애	{S04, S25}
S26	Abnormal state of DNS resolution	DNS 장애	{S05, S27}
S27	Abnormal state of forwarding DNS	DNS 장애	{S05, S27}

활성 네트워크를 이용하여 최종 장애 진단 집합인 달성목표에 도달하기 위해서는 규칙의 정형화가 필요하게 되며, 본 고에서는 웹 서비스 장애진단 규칙을 시스템레벨 생성 규칙과 서비스레벨 생성규칙으로 계층화였다. 여기서 사용되는 규칙표현 기법은 INBANCA의 규칙 표현 기법을 이용하였다. 추론 과정은 case library와 활성 네트워크를 이용하여 진행되며, 사건Px에 의해서 추론 과정이 활성화되며, 이전 상태와의 연관성을 이용하여 활성네트워크의 범위를 줄여나가며 달성 목표를 획득하게 된다. 이때 case library에서 새로운 진단 상태를 추출하는 retrieve_case()와 활성 네트워크와 이전 상태를 반영하여 장애를 추론하는 execute(), 장애 진단 시 환경 변수들을 습득하는 learn() 함수를 이용하여 추론 과정을 수행한다.

<표 3>은 시스템 레벨에서의 장애 발생 여부를 판별하기 위한 시스템 레벨 장애 진단 생성규칙은 사건 *P1*에 의해서 규칙*R1*이 활성화되며, 이 진단 규칙에 따라 어떠한 장애가 발생하였는지를 파악하게 된다.

〈표 3〉 시스템 레벨 장애 진단 생성규칙

진단 규칙	R1
사건	P1 *P1:no response from web server
활성 네트워크	{S07, S10, S11, S12, S13}

생성 규칙	**Key:**
	L: Case Library = {P_1,P_2,P_3}
	S: Current State
	A: Current Active Network = {,S07,S10,S11,S12,S13}
	$S_{initial}$:Initial State = {S_{00}}
	NMBANCA(A,L,S) =
	WHILE not(stoping_criterion_satisfied(S))
	a = retrieve_case(A,S,L)
	DO(1≤i≤rule_length(A))
	$S_{previous}$:= S
	S := execute(action(a,i),$S_{previous}$)
	IF(failure(S))
	THEN RETURN{A,L}
	IF(S_{07})
	{A,L} := learn(S, $S_{initial}$, a, i, A,L)
	RETURN {A,L}

<표 4>의 생성 규칙*R2*는 웹 서버의 구성 및 성능 장애를 진단하는 서비스 레벨 장애 진단 생성규칙으로 상태코드 분석을 통하여 사건 *P2, P3*가 발생한 경우 활성화되며, 이 진단 규칙에 따라 어떠한 장애가 발생하였는지를 파악하게 된다.

<표 3>과 <표 4>는 INBANCA 기법에서 정의하고 있는 생성규칙 표현방식에 따라 기술한 생성규칙으로 추론과정은 진단할 활성 네트워크의 범위를 줄여나가면서 최종 목적 노드에 도달하게 된다. 이 때 수집 데이터의 임계치가 필요한 경우 사례베이스에서 가장 적

합한 사례를 추출하여 사용하며, 만약 규칙을 만족하지 않는 경우 사례 표현규칙에 따라 장애사건을 기술하여 사례베이스에 추가하게 된다.

〈표 4〉 서비스 레벨 장애 진단 생성규칙

진단 규칙	R2
사건	P2/P3 *P2: receive response message with error status code *P3 : excess response time
활성 네트워크	{S20, S21, S22, S23, S24, S25, S26, S27}

생성 규칙	**Key:**
	L: Case Library = {P,P_2,P_3}
	S: Current State
	A: Active Network = {S_{20}, S_{21}, S_{22}, S_{23}, S_{24}, S_{25}, S_{26}, S_{27}}
	$S_{initial}$:Initial State = {S_{00}}
	NMBANCA(A,L,S) =
	WHILE not(stoping_criterion_satisfied(S))
	IF(Pi∈ {S_{21}})
	THEN A = { S_{21},S_{22}}
	IF(Pi∈ { S23})
	THEN A = {S_{23},S_{24} ,S_{25}}
	IF(Pi∈ {S_{26}})
	THEN A = {S_{26},S_{22}}
	a = retrieve_case(A,S,L)
	DO(1≤ i ≤ rule_length(A))
	$S_{previous}$:= S
	S := execute(action(a,i),$S_{previous}$)
	IF(failure(S))
	THEN RETURN{A,L}
	IF(S ∈ {S_{24},S_{25},S_{23}})
	{A,L} := learn(S, $S_{initial}$, a, i, A,L)
	RETURN {A,L}

3.2 장애 복구 기법

INBANCA기법에서 장애 복구는 사례 라이브러리에서 적합한 복구 규칙을 추출하여 수행되는데, 이때 사례는 장애 진단 활성 네트워크의 탐색노드 순서쌍에 의해서 구분되며, 만약 정의되지 않은 순서쌍이 발견될 경우 유사성이 제일 높은 사례가 선택되어진다. <표 5>는 장애 복구 사례 라이브러리의 초기규칙을 나타내는 것으로 탐색노드 순서쌍 집합으로 {*q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11*}이 정의되며, 각 사례별로 활성 라이브러리인 Caseq내에서 우선순위에 따라 여러 복구방법이 수행되어진다. 이때 특정 복구방법이 실패할 경우 우선순위가 내려가며, 활성 라이브러리 내의 모든 방법이 실패할 경우 사례 라이브러리에 새로운 규칙이 추가되게 된다.

〈표 5〉 장애 복구 사례 라이브러리

Case	Recovery Rule	Case	Recovery Rule
q1	Caseq1={sol1,sol2}; if Failure(Action(C, soli,pi)) then down pi; ReactWrite(Caseq1,Recoveryq1);	*q6*	Caseq6={sol2,sol4}; if Failure(Action(C,soli,pi)) then down pi; ReactWrite(Caseq6,Recoveryq6);
q2	Caseq2={sol0 }; if Failure(Action(C, soli,pi)) then down pi; Write(Caseq2,Recoveryq2);	*q7*	Caseq7={so6,sol5 }; if Failure(Action(C,soli,pi)) then down pi; Write(Caseq7,Recoveryq7);
q3	Caseq3={sol1,sol3, sol4}; if Failure(Action(C, soli,pi)) then down pi; Write(Caseq3,Recoveryq3);	*q8*	Caseq8={sol9}; if Failure(Action(C,soli,pi)) then down pi; Write(Caseq8,Recoveryq8);
q4	Caseq4={sol1,sol3, sol4}; if Failure(Action(C, soli,pi)) then down pi; Write(Caseq4,Recoveryq4);	*q9*	Caseq9={sol5,sol7}; if Failure(Action(C,soli,pi)) then down pi; Write(Caseq9,Recoveryq9);
q5	Caseq5={sol1,sol3, sol4}; if Failure(Action(C, soli,pi)) then down pi; Write(Caseq5,Recoveryq5);	*q10, q11*	Caseq10={sol10}; if Failure(Action(C,soli,pi)) then down pi; Write(Caseq10,Recoveryq10);
q1: (S12,S02) *q2: (S13,S02)* *q3: (S10,S06)* *q4: (S11,S06)* *q5: (S22,S03)* *q6: (S21,S03)* *q7: (S23,S04)* *q8: (S24,S04)* *q9: (S25,S04)* *q10: (S26,S05)* *q11: (S27,S05)*		*sol1: 프로세스 재가동* *sol2: 기본 구성정보 수정* *sol3: 로그 기록 등의 부가적인 행위 중지* *sol4: 성능향상 구성정보 변경* *sol5: 시스템부하 스케줄링* *sol6: 최대 서비스 제공 횟수 감소* *sol7: 최대 서비스 제공 횟수 증가* *sol8: 프로세스 풀 증가* *sol9: 프로세스 풀 감소* *sol10: 사용자 통보*	

이와 같이 사례 라이브러리를 기반으로 한 장애 복구기법은 발생특성에 따라 복구 규칙이 수정되기 때문에 서비스 특성, 트래픽 특성 등 다양한 환경 변수들의 영향을 반영한 적응적인 장애 복구에 적합한 방식이라 할 수 있다.

4. 실험 및 고찰

제시한 웹 서버 장애 진단 모델을 실제 환경에 적용시켜 장애를 감지하고 이에 대한 복구 수행을 살펴봄으로써 장애검출규칙과 그에 대한 복구 방법론에 대한

타당성 여부를 살펴보도록 하겠다. 웹 서버의 장애를 검출하기 위해서 본 실험에서 사용된 환경은 <표 6>에서 보는 바와 같이 1대의 서버와 3대의 클라이언트로 구성하였다. 서버 환경은 현재 웹 서버의 60%정도를 차지하는 NCSA 웹 서버를 사용하였으며[11], 운영체제는 리눅스를 기반으로 하여 실험하였다.

〈표 6〉 실험 대상 시스템

Server Machine		O/S	Memory
S1	Pentium II 300	Linux RedHat5.0	64MB
Client Machine		O/S	Memory
C1	Sparc	SunOS 5.5.1	64MB
C2	Sparc	SunOS 5.6	64MB
C3	Sparc	SunOS 5.6	128MB

<표 6>의 대상 시스템을 기반으로 장애 환경을 구성하지 위해 Webstone(SGI)을 이용하였는데, WebStone은 하드웨어 구조나 운영 체제등과 무관하게 웹 서버의 성능을 평가하는 벤치마크로 본 실험에서 사용하기에 적절하다고 할 수 있다.

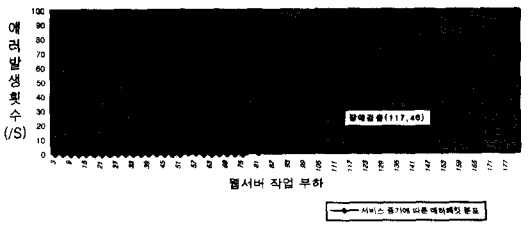
즉, WebStone을 *C1, C2, C3* 클라이언트에 설치하고 클라이언트별로 HTTP/1.0 GET 요청을 이용하여 작업 부하를 점진적으로 증가시킴으로써 장애 환경을 구성하였으며, <표 7>의 실험 기준표를 사용하여 장애 발생시점과 이에 발생한 시스템의 장애 검출 및 복구 행위들을 살펴봄으로써 규칙의 타당성을 살펴보도록 하겠다.

〈표 7〉 실험 임계치 및 사례 라이브러리

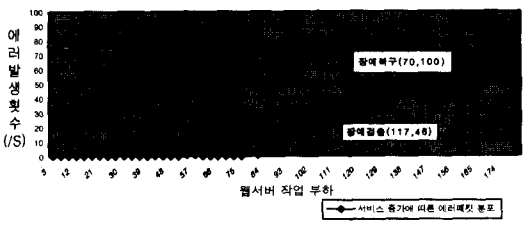
Test	Fault State	Detect Parameter	Production Rule	Case	Recovery Action	Recovery Status
T1	S04	β >80(%)	R2	q8	sol9	success
T2	S06	α>200(ms)	R2	q4	sol1	fail
					sol3	fail
					sol4	success
T3	S03	Configuration Modify	R1	q6	sol2	success
					sol4	-

4.1 서버 과부하 장애 검출 및 복구

T1 실험은 두 가지 경우로 나누어 실험을 하였으며, 먼저 첫째로, (그림 3(a))와 (그림 3(b))는 웹 서버의 작업 부하의 증가에 초점을 맞추어 실험한 것으로 *C1, C2, C3*의 초기 클라이언트의 수를 1로 설정하고 초당 1씩 클라이언트의 수를 증가시켰을 경우의 에러 코드의 발생 횟수와 이때의 상태코드를 나타내고 있다.



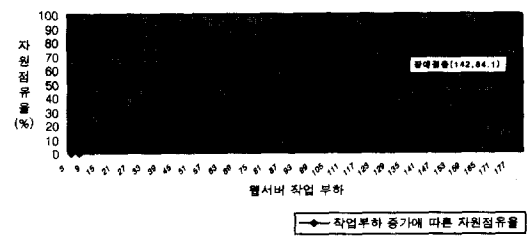
(그림 3(a)) 서비스 증가에 따른 에러 패킷 분포



(그림 3(b)) 서비스 증가에 따른 에러 패킷 분포 (복구 수행)

(그림 3(a))가 기존의 환경에서 실험한 그래프이며, (그림 3(b)) 그래프의 경우가 장애 복구를 수행했을 경우의 결과 그래프이다. (그림 3(a)) 그래프에서 보는 바와 같이 *f1*이 장애가 발생한 시점을 나타내고 있으며, 이때의 초당 클라이언트 수 117이며, 그 중 40%가 에러코드를 포함함을 알 수 있었다. (그림 3(b))에서는 <표 7>의 실험 기준표에 의하여 복구를 수행했을 경우의 에러 패킷 분포를 나타내는 것으로 장애 복구 시점 이후의 에러 패킷 발생 비율에서 알 수 있듯이 장애 복구를 수행하였을 경우 약 5%정도 에러 발생횟수가 줄어드는 것을 확인할 수 있었다.

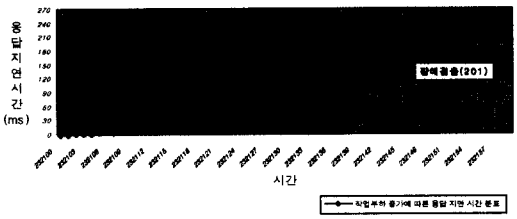
다음으로 시스템의 자원 점유율을 기준으로 웹 서버의 과부하를 실험하였는데, (그림 3(c))에서 볼 수 있듯이 자원 점유율이 80%이상인 *f2*에서 장애 S04를 검출하였으며, 이때 사용자에게 시스템 과부하 장애를 통보함을 확인할 수 있었다.



(그림 3(c)) 서비스 증가에 따른 자원 점유율 분포

4.2 서비스 제공시간 초과 검출 및 복구

T2를 실험하기 위하여 초당 클라이언트의 수를 100으로 고정하고 이때 다른 시스템 작업부하를 증가시키는 방식으로 실험 환경을 구성하였다. (응답시간의 임계치는 200ms로 설정하였다). 이 때의 시스템 작업 부하와 웹 서버의 응답시간과의 관계를 그래프로 그렸을 경우 아래의 (그림 4)와 같이 나타나는데, 시스템 작업부하가 90%이상이 되었을 경우 응답시간이 임계치를 초과하는 경우가 발생함을 확인할 수 있었으며, 이때 장애를 검출하고 관리자에게 시스템의 부하 조절이 필요함을 통보하는 것을 확인할 수 있었다.



(그림 4) 작업 부하 증가에 따른 응답 지연 시간 분포

4.3 구성 정보 오설정 검출

T3를 실험하기 위해서 NCSA 웹 서버의 구성파일 내용을 임의로 잘못 설정하고 이때 장애 진단 규칙에 의해서 이를 검출하고 복구 규칙에 따라 올바르게 복구되는지를 실험해 보았다. 이때 실험한 항목은 NCSA 웹 서버 구성항목을 기본으로 <표 8>과 같이 장애상황을 구성하였다. 결과로 웹 서버의 구동에 기본적으로 필요한 항목들의 정보가 잘못 설정되었을 경우 사례 q6가 적용되며, 성능관련 항목이 잘못되어 있을 경우 사례 q5가 적용됨을 알 수 있었으며, 두 경우 모두 장애 관리 시스템이 자동적으로 이를 검출하고 유효한 정보로 수정함을 확인할 수 있었다.

〈표 8〉 구성항목 및 장애 실험결과

Configuration item	Case	Recovery Action	Recovery Status
Port	*q6*	*sol2*	success
ServerRoot	*q6*	*sol2*	success
TimeOut	*q5*	*sol1*	fail
		sol3	success
ServerName	*q6*	*sol1*	success
PidFile	*q6*	*sol1*	success

5. 결 론

이 논문에서는 가변적인 웹 서버 환경에 적응할 수 있는 장애관리 메커니즘을 사례기반의 INBANCA기법을 사용하여 진단 및 복구 규칙으로 정의하였다. INBANCA기법의 규칙 표현 기법에 따라 달성 목표의 정의, 서버 상태감시 및 장애 진단 지식의 활성 네트워크(Active Network)표현, 생성 규칙정의 및 수집데이터의 사례 라이브러리화를 정형화하였다. 또한 진단 규칙을 웹 서버의 자원 및 구성을 주기적으로 감시하는 시스템 레벨 장애 진단 생성규칙과 검사 패킷을 이용한 성능 및 서비스 장애를 진단하는 서비스 레벨 장애 진단 생성규칙으로 계층화 하였으며, 장애 항목에 대한 복구 방법 정의 및 복구 지식에 대한 규칙을 사례 라이브러리로 표현하였다.

이와 같은 웹 서버 장애 관리를 위해 시스템 레벨 장애 추론을 수행하는 WMA와 서비스 레벨 장애 추론을 수행하는 MGA로 시스템을 구성하였으며, 추론 엔진과 규칙/사례 라이브러리를 이용하여 지능적인 장애 추론 및 복구를 수행한다. 그리고 웹 성능 벤치마크인 Webstone을 이용한 장애 환경 구성 및 각 장애 항목에 대한 장애 진단 생성규칙 적용과 사례 라이브러리를 이용한 복구과정을 보임으로써, 제안한 장애 진단규칙 및 복구 메커니즘의 타당성을 실험을 통하여 증명하였다. 이러한 웹 서버 장애 관리 기법은 급속히 증가하는 웹 서버의 관리를 위한 관리자의 노력과 비용을 줄이고, 좀더 적응적이며 지능적인 웹 서비스 장애관리가 가능하게 할 것이다.

참 고 문 헌

[1] Martin F.Arlitt and Carey L. Williamson, "Web server workload characterization," in Proceedings of the ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems, 1996.

[2] Paul E. Hoffman, "Web Wervers Survey," http://www.webcompare.com/web_server.html, 1999.

[3] 한정수, 안성진, 정진욱, "Web-based performance manager system for a Web server," Network Operations and Management Symposium NOMS 98.,

IEEE, 1998.

[4] 홍원택, 최영수, 정진욱, "효율적인 성능관리를 위한 TCP/IP 네트워크 이용률 예측에 관한 연구", 한국정보처리학회 추계학술발표논문집, 1998.

[5] Yiming Hu, Nanda, A, Qing Yang, "Measurement, analysis and performance improvement of the Apache Web server," Performance, Computing and Communications Conference IEEE International, 1999.

[6] Goldszmidt, G. S, "Load management for scaling up Internet services," Network Operations and Management Symposium NOMS 98., IEEE, 1998.

[7] Polze, A, Richling. J, Schwarz, J, Malek, M, "Towards predictable CORBA-based Web-services," Object-Oriented Real-Time Distributed Computing (ISORC '99). Proceedings. 2nd IEEE International Symposium, 1999.

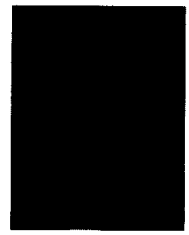
[8] Hung, C. K, Lai, E. M.-K,. "An Intelligent Assistant For The Management of Telecommunications Network Services," Expert Systems for Development, Proceedings of International Conference on , 1994.

[9] The Apache Team, "Apache HTTP server project," http://www.apache.org

[10] Roy T, Fielding, Jim Gettys et, "Hypertext Transfer Protocol-HTTP/1.1," Internet Draft draft-ieft-httpv1.1-spec-07, 1996.

[11] Lee KangChan, "국내 WWW 서버 조사(WWW Server Survey in Korea)," URL : http://sharon.comeng.chungnam.ac.kr/~dolphin/Server/compare.html

[12] Vingralek, R. Breitbart et, "A transparent replication of HTTP service," Data Engineering, 1999. Proceedings.15th International Conference, 1999.



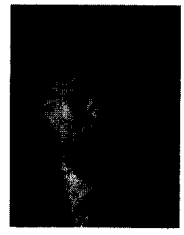
윤 정 미

e-mail : jmyun@songgang.skku.ac.kr

1999년 성균관대학교 정보공학과 졸업(학사)

1999년~현재 성균관대학교 전기전자 및 컴퓨터 공학부 대학원 석사과정

관심분야 : 응용 서비스 관리, 네트워크 관리



안 성 진

e-mail : sjahn@comedu.skku.ac.kr

1988년 성균관대학교 정보공학과 졸업(학사)

1990년 성균관대학교 대학원 정보공학과 졸업(석사)

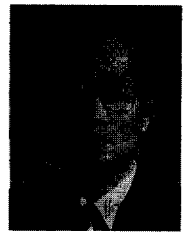
1998년 성균관대학교 대학원 정보공학과 졸업(박사)

1990년~1995년 한국전자통신연구원 연구 전산망 개발실 연구원

1996년 정보통신 기술사 자격 취득

1999년~현재 성균관대학교 컴퓨터교육과 전임강사

관심분야 : 네트워크 관리, 트래픽 분석, 보안 관리



정 진 욱

e-mail : jwchung@songgang.skku.ac.kr

1974년 성균관대학교 전기공학과 학사

1979년 성균관대학교 대학원 전자공학과 석사

1991년 서울대학교 대학원 계산통계학과 박사

1982년~1985년 한국과학기술 연구소 실장

1981년~1982년 Racal Milgo Co. 객원연구원

1985년~현재 성균관대학교 전기전자 및 컴퓨터공학부 교수

관심분야 : 컴퓨터 네트워크, 네트워크 관리, 네트워크 보안