6

분산 환경에서의 온라인 게임 서버 플랫폼 테스트 자동화

NHN ▌이강희 · 유석문

1. 서 론

게임의 온라인화와 분산 환경에서의 운영이 증가함에 따라 효과적인 게임 개발 플랫폼에 대한 요구도 함께 증가하고 있다. NHN의 게임 포탈인 한게임에서는 게임 개발에 공통적으로 요구되는 기능을 플랫폼으로 제공하고 있다. 게임 플랫폼은 로깅, 통신, 병렬처리 등 게임 개발 시 필요한 공통 기능을 제공하는 게임 개발 플랫폼과 게임 서버간 중복 접속 방지, 게임 채널 정보 제공, 분산 서버 간 메시지 처리, 분산 서버 자원 할당 등 게임을 서비스하기 위한 기능을 제공하는 게임 서비스 플랫폼으로 분류된다. 게임 플랫폼은 게임 제작과 운영의 필수 기능을 제공하기 때문에 높은 수준의 안정성과 성능이 요구되며 특히 한게임에서 사용되고 있는 게임 플랫폼은 분산 환경을 지원하고 있어 분산 환경에서 테스트가 필수적이다.

본 논문에서는 분산 환경에서의 효과적 테스트 수행을 위하여 NTAF(NHN Test Automation Framework)[1]을 이용하여 테스트 환경 구성 및 제거를 자동화하는 방안을 제시하고 그 실 적용 사례 및 효과를 검증한다.

NTAF은 NHN에서 2008년 개발되어 오픈 소스로 운영되고 있는 End-to-end Testing Automation Framework이다. 편리한 테스트 작성을 위하여 Wiki를 지원하며 시나리오 테스트 구성을 위한 제어구문을 제공한다. 이에 더하여 분산 환경 구성과 테스트 수행을 위한 다양한 서비스를 제공하고 있어 효과적으로 분산 환경에서 다양한 테스트를 수행할 수 있다.

분산 환경에 대한 테스트 환경 구성을 자동화함으로 반복적인 환경 구성 및 제거에 들어가는 개발자/테스터/QA의 리소스를 절약할 수 있으며 Human Error의 개입을 미연에 방지할 수 있어 보다 정확하고 빠른 품질 검증을 수행할 수 있게 된다. 또한 한 번 구축된 테스트 환경은 테스트 라이브러리로 재사용 될 수 있다.

2. 분산 환경에서의 게임 플랫폼 테스트 자동화

최근 들어 소프트웨어 개발 생산성 향상을 위하여 다양한 테스트 자동화 도구들이 이용되고 있다. 그러나 테스트 자동화 도구의 도입의 증가에도 불구하고 테스트의 상당 부분은 수동으로 수행되고 있으며 특히 분산 환경에서의 테스트 환경 구성 작업은 다양한 운영체제, 플랫폼, 어플리케이션 등이 포함되어 효과적으로 자동화 되지 못하고 있는 실정이다. 이러한 문제점을 해결하기 위하여 본 논문에서는 FitNesse[2]와 STAF(Software Testing Automation Framework)[3]을 결합하여 분산 환경 테스트를 지원하는 NTAF을 분산 환경에서의 게임 서버 플랫폼 테스트 자동화에 적용하였다.

2.1 테스트 자동화를 위한 방안

Fit[2]은 Ward Cunningham에 의하여 개발된 통합 테스트 자동화 프레임웍이며 FitNesse는 Fit 프레임웍 위에 Wiki를 구현하여 테스트 작성 및 수행을 보다 편리하게 할 수 있도록 한 테스트 자동화 프레임웍이다. Fit과 FitNesse의 최대 장점은 테스트 작성 및 수행이 용이하여 다양한 관련자들이 쉽게 협업을 할 수 있다는 점이다. 개발 분야에 전문적인 지식이 없는 기획자와 사용자도 쉽게 자신들이 원하는 기능에 대한 테스트를 테이블 형태로 작성할 수 있으며 이를 바탕으로 개발자/QA/Tester들은 자동화된 테스트를 작성 및 실행할 수 있게 된다. 이는 통상의 문서를 바탕으로 한 의사소통에 비하여 보다 명확하고 적은 비용이 들어가며 개발 단계에 사용자/기획자/QA/Tester들이 함께 참여할 수 있도록 하여 품질을 높일 수 있게 도와준다[1]. Fit과 FitNesse의 경우 테스트 작성을 위한 프레임웍을 제공하나 분산 환경에서의 테스트 환경 구성/실행에 관련된 기능을 제공하여 주지 못하는 한계를 가지고 있다. 이런 제약을 극복하기 위하여 STAF이 활용될 수 있다. STAF은 분산 환경에서의 테스트

환경 구축/실행/분석을 위한 기능을 제공하는 오픈 소스 프레임웍이다. 또한 시나리오 테스트 수행을 위하여 테스트 시나리오를 작성하고 실행할 수 있는 STAX (STAF eXecution engine)을 제공하고 있다. STAX는 XML과 Python으로 작성된 시나리오 테스트를 수행하는 구조로 되어 있어 XML과 Python에 대한 지식이 없는 경우 테스트 작성이 어려우며 테스트 작성 과정에서 발생하는 오류를 쉽게 발견 및 수정할 수 없는 문제점을 가지고 있다.

NTAF는 FitNesse와 STAF의 장점을 결합하여 개발된 테스트 자동화 프레임웍이다. NTAF에서는 FitNesse에서와 같이 테이블을 이용하여 시나리오 테스트를 작성할 수 있으며 이를 위한 다양한 제어 구문을 제공하고 있다. 이러한 기능은 개발 지식이 없는 사용자도 쉽게 테스트를 작성할 수 있게 지원한다. 또한 STAF에서 제공되는 모든 분산 환경 기능을 NTAF에서 사용할 수 있어 분산 환경 테스트가 매우 용이하다.

그림 1은 NTAF의 아키텍처이다. NTAF은 Fit과 FitNesse의 테이블 형태의 테스트를 수행하는 부분과 NTAF 고유의 제어구문을 처리하는 부분 그리고 STAF의 서비스와 연계하여 분산 환경을 지원하는 요소로 구성되어 있다. BVT(Build Verification Test) Service는 STAF의 외부 서비스로 개발되어 연동되어 있으며 분산 환경에서 테스트를 실행시키고 해당 결과를 취합하는 역할을 수행한다.

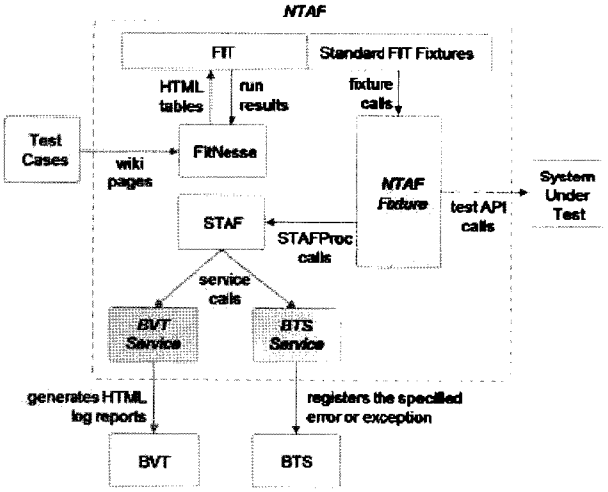

그림 1 NTAF 아키텍처[1]

BTS(Bug Tracking System) Service는 테스트 수행 중 오류 발생 시 자동으로 JIRA[4]에 해당 오류를 등록하고 담당자에게 통보하는 기능을 수행하며 STAF의 외부 서비스로 개발되었다.

NTAF은 HUDSON CI 서버와 MAVEN에 대한 Plug-in을 제공하여 보다 쉽고 반복적으로 테스트 수행 결과를 확인할 수 있는 기능을 제공한다. NTAF의 자세한 특성과 사용법은 NHN의 개발자 센터[7]에서 확인할 수 있다.

NTAF은 분산 환경에서의 테스트 수행을 위해 필요한 필수 기능을 모두 제공하고 있으며 본 논문에서는 해당 기능을 활용하여 게임 플랫폼의 테스트 자동화를 구현하였다.

2.2 게임 서버 플랫폼의 End-to-end Testing Automation 적용

2.2.1 테스트 환경 구축 자동화

테스트 자동화는 "환경 구성", "테스트 수행", "결과 분석", "환경 제거"의 순서로 구성된다[5].

테스트 환경 구성은 테스트를 수행하기 위하여 SUT (System Under Test)와 SUT가 사용하는 DOC(Depended-On Component)를 테스트 장비에 설치하고 초기 상태로 설정하는 단계를 말한다. 복잡한 테스트 환경 구성을 테스트마다 반복적으로 실행하는 것은 테스트 수행 간의 간섭에 의한 오류를 배제하기 위한 목적이다. 그러나 복잡한 구성을 갖는 서비스의 경우 환경 구성을 수동으로 수행할 경우 사용자 오류가 개입될 수 있다. 또한 수동 작업으로 인한 민첩성의 저하로 신속한 테스트 환경 구축이 필요한 경우 효과적으로 대응할 수 없는 한계점을 가진다.

환경 구축은 그림 2의 순서도와 같이 수행된다. 첫 번째 단계는 테스트 장비가 서비스가 가능한 상태인지를 확인하는 단계이다. 테스트 장비가 정상적인 경우 다음 단계에서 필요한 SUT와 DOC 설치를 진행할 수 있으며 정상적이지 않을 경우 제어문을 이용하여 테스트를 종료하고 오류 보고를 하거나 해당 테스트 장비는 테스트 수행에서 배제하도록 구성할 수 있다.

다수의 테스트 장비에 SUT와 DOC를 설치해야 하는 경우 순차 설치에는 많은 시간이 소요된다. 이를 효

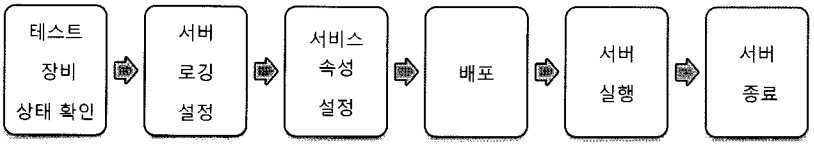

그림 2 테스트 환경 구축 순서도

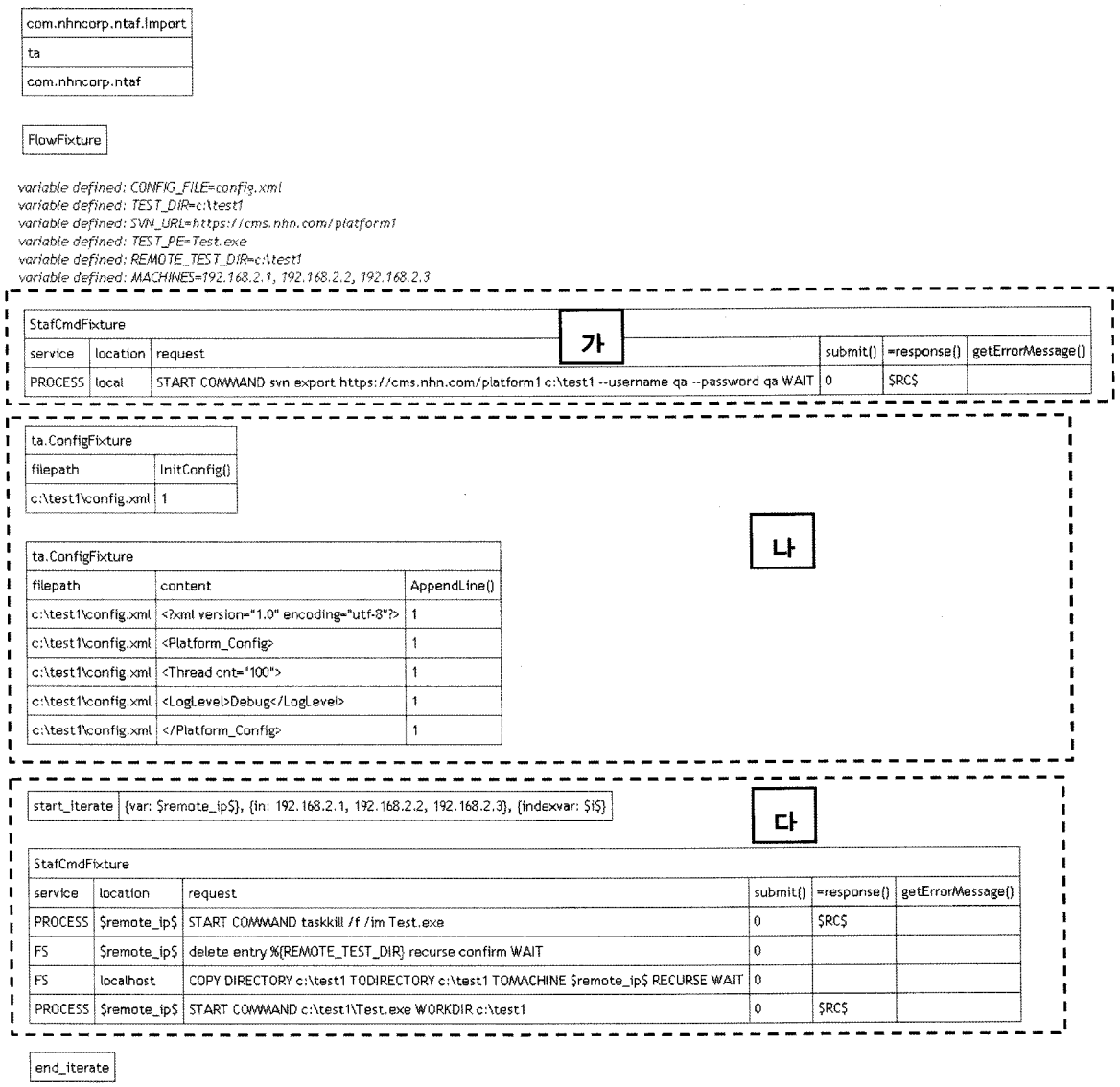

그림 3 테스트 환경 구성의 NTAF 예제

율적으로 수행하기 위해 NTAF에서 제공하는 PARALLEL 기능을 활용하여 해당 작업을 병렬로 수행하여 수행 시간을 단축할 수 있다.

그림 3은 게임 플랫폼 테스트 환경 구축의 NTAF 사용 예이다. [가]에서 선언한 StafCmdFixture는 STAF에서 제공하는 서비스를 호출하기 위한 Fixture[6]로 도스 명령어를 호출하여 SVN의 코드를 테스트 장비에 다운로드 하는 명령어이다. [나]는 "c:₩test" 폴더의 "config.xml"이라는 서버 플랫폼의 설정 파일을 자동 생성하는 Fixture의 예이다. [다]는 [가]와 [나]에서 다운로드 받고 설정된 전체 플랫폼 파일들을 "MACHINES"라는 변수에 선언된 분산 테스트 장비들에 자동 배포하기 위한 명령어이다. [다]는 다음과 같은 순서로 파일들을 배포한다.

1) 분산 테스트 장비의 "Test.exe"라는 프로세스를 강제 종료
2) 분산 테스트 장비의 "c:₩test1" 폴더를 삭제
3) 현재 테스트 장비의 "c:₩test1" 폴더의 파일들을 각 분산 테스트 장비에 복사
4) 새롭게 배포된 "Test.exe" 프로세스를 분산 테스트 장비에서 실행

위 사례와 같이 복잡한 스크립트 언어 등을 이용하지 않고 NTAF 키워드만으로 서버 플랫폼을 원격지에 자동으로 쉽게 설치가 가능하다. 그러므로 NTAF은 개발에 대한 전문적인 지식이 없는 경우에도 쉽게 사용할 수 있다. 만일 테스트 장비의 수를 늘리고 싶은 경우라면 그림 3에서 "MACHINES"라는 변수에 IP 주소만 추가하면 된다.

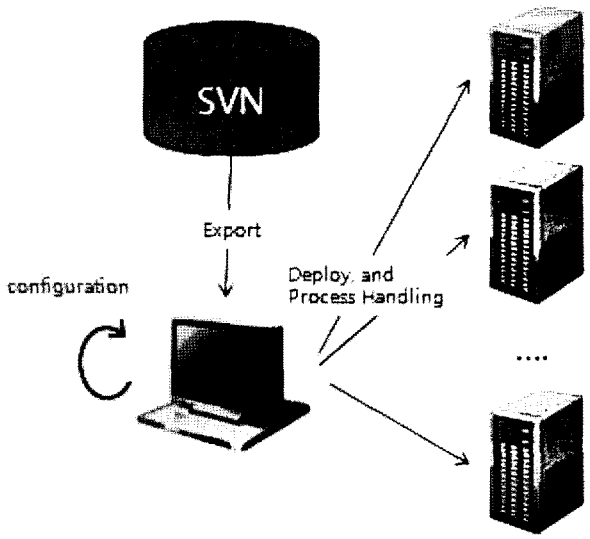

그림 4 테스트 환경 구성도

그림 3에서 작성된 자동 테스트 환경 설정 시나리오는 그림 4와 같이 구성된다. 테스트 환경을 구성하기 위해서는 우선 SVN(Subversion)에서 테스트 장비에 설치할 게임 플랫폼을 설치를 담당 하는 컴퓨터로 Export 받는다. 그리고 각 설치 환경에 맞추어 게임 서버 플랫폼의 설정 정보를 변경한 후 테스트 장비에 배포하게 된다. 효과적인 테스트 구성을 위하여 시간이 많이 소요 되는 파일 전송 부분은 최소화하고 각 환경에 맞추어 변경된 설정 파일만을 배포하도록 구성하였다.

2.2.2 기능 테스트 자동화

게임 서버 플랫폼의 기능 테스트를 자동화하기 위해서는 테스트 용도에 맞추어 Fixture를 제작하여야 한다. Fixture 작성에는 약간의 개발 지식이 필요하므로 통상 개발자 또는 QA/Tester에 의하여 작성된다. Fixture는 Test Case와 SUT를 연결하는 기능만을 수행하므로 가능한 간단하고 유지보수가 용이하도록 작성되어야 하며 코드 내에 검증을 위한 코드를 제외하고는 어떠한 비즈니스 처리 로직도 가져서는 안된다[2].

```
//채널의 속성 값을 변경하는 함수
bool CClsConActionFix::updateChannelScope()
{
        //데이타 핸들 생성
        Data::Handle hData        = Data::CreateHandle();

        //데이타 핸들에 채널 ID, 서버 주소, 속성 값 할당
        if( NULL == hData || !Data::SetChannelScope(hData, &m_lszAddr,
                        (char *)m_szChannelID.c_str(), (char *)m_szScope.c_str()) )
        {
                std::cout << "Fail to set data  " << std::endl;
                if(hData != NULL) Data::DestroyHandle(hData);
                        return false;
        }

        //데이터 핸들을 사용해서 서버에 속성값을 갱신함
        if( false ==Connector::Update(m_hCon, hData))
        {
                std::cout << "Fail to update  " << std::endl;
                if(hData != NULL) Data::DestroyHandle(hData);
                        return false;
        }

        std::cout << "Success to update  "    << std::endl;
        //사용한 데이타 핸들 자원 해지
        if( hData != NULL) Data::DestroyHandle(hData);
        return true;
}
```

그림 5 Fixture 작성 예

ActionFixture		
start	CClsConActionFtx	
enter	setChannelID	KOR.BZBADUK.0100A.088
enter	setMRSAddr	a:0:3005:0:0:0
check	addMRSAddr	1
enter	setScope	G
check	createConnection	1
check	updateChannelScope	1
check	disconnect	1

ClsViewerRowFtx		
m_strChnlId	m_strIp	m_strScope
KOR.BZBADUK.0100A.088	127.0.0.1	G
KOR.BZBADUK.0100A.001	119.205.239.47	Z
KOR.BZBADUK.0200A.001	119.205.239.49	Z
KOR.BZBADUK.0300A.001	119.205.239.48	Z
KOR.BZBADUK.0300A.011	10.8.15.251	G
KOR.BZBADUK.0500A.001	119.205.239.46	Z
KOR.BZBADUK.0600A.001	119.205.239.45	Z
KOR.BZBADUK.0700A.001	119.205.239.44	Z
KOR.BZBADUK.0800A.001	119.205.239.43	G
KOR.BZBADUK.0800A.002	10.8.14.147	G

그림 6 게임 서버 플랫폼의 기능 테스트 자동화 예

그림 5는 게임 플랫폼의 서버에 ChannelScope라는 속성 값을 변경하는 기능에 대한 Fixture 코드의 예이다.

그림 5와 같이 서버 플랫폼에서 제공되는 각각의 기능에 대한 Fixture를 작성한다. 이와 같이 작성된 Fixture를 조합하여 Test Case 별 시나리오를 구성한다. 그림 6은 그림 5에서 작성된 Fixture의 updateChannelScope라는 함수를 이용해서 서버의 속성 값을 변경하고 이 값이 정상적으로 변경되었는지를 서버에 조회하는 Test Case를 자동화한 예이다.

그림 6은 서버에서 관리하는 데이터를 갱신하는 기능과 조회하는 기능을 조합하여 간단한 시나리오를 작성하였다. 파란색으로 나타난 부분은 각 함수나 데이터가 예상된 결과값을 반환하였음을 나타내는 것으로 정상적으로 테스트가 완료되었음을 나타내고 붉은색 부분은 예상하지 못한 데이터가 추가적으로 나타난 것을 의미하는데 위 예에서는 "KOR.BZBADUK.0100A.088"라는 이름을 가진 채널의 "Scope"값이 "G"로 정상 변경되었으므로 다른 채널의 데이터는 고려대상이 아니다. 따라서 정상적으로 테스트 결과가 나왔음을 알 수 있다. 이와 같이 각 기능에 대해서 Fixture를 작성하고 NTAF을 활용하여 구성 및 검증함으로써 소프트웨어의 변경 발생 시 회귀 테스트의 효율성을 증대할 수 있다.

2.2.3 다수의 가상 사용자 게임 환경 구성 자동화 및 테스트 수행

게임 서버 플랫폼은 다수의 게임 사용자에게 필요한 서비스를 제공하기 위한 목적으로 제작되므로 멀티 쓰레드 환경하의 데드락(Dead lock) 문제나 경쟁상태(Race condition)에 대비하여야 한다. 이를 검증하기 위해서는 다수의 사용자가 서버에 접속하여 서비스를 제공받는 테스트가 필요하나 수동으로 테스트 프로그램을 분산된 테스트 장비에 배포 및 수행하고 테스트 결과 데이터를 수집하기는 쉽지 않다.

NTAF을 이용하여 자동으로 테스트 프로그램을 분산된 테스트 장비에 배포, 설정, 실행하고 테스트 시나리오에 따라서 통제하도록 구현하였다. 테스트 프로그램은 가상의 사용자가 서버에 접속하여 서비스를 받도록 시뮬레이션 하며 설정 파일을 통해서 다양한 시나리오를 수행하고 그 기록을 남기도록 구현하였다.

그림 7은 테스트 프로그램을 통해서 다수의 사용자가 게임 서버 플랫폼을 사용하는 것과 같은 환경을 시뮬레이션 하는 구성도이다.

2.2.1장에서 설명한 테스트 환경 구축 자동화 기능을 통해서 우선 서버 플랫폼의 설치를 종료한 이후에 테스트 프로그램을 코드 저장소에서 가져와 어떤 테스트 장비에 배포할 것이고 어떤 시나리오로 시뮬레이션 할 것인지에 대한 설정을 완료한다. 그 후, 테스트 장비에 배포를 하고 주어진 시나리오에 맞추어 테스트 프로그램을 실행하며 개별적인 트랜잭션에

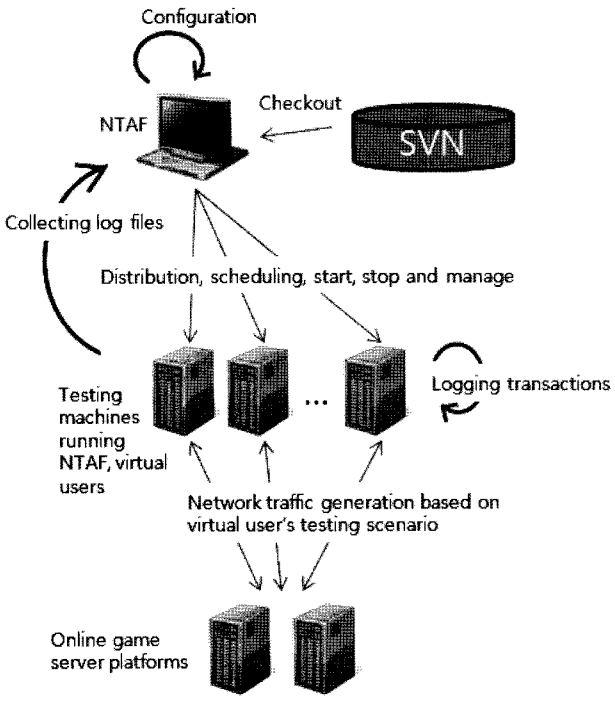

그림 7 다수의 가상 사용자 게임 환경 자동화 구성도

대해서 로깅을 한다. 이러한 일련의 과정이 자동화 프레임웍인 NTAF에서 스케줄링 된다.

테스트 종료 후에는 여러 분산 테스트 장비에 남겨진 로그 데이터를 수집하여 기능 테스트에서 확인하기 힘든 동시성 문제나 서버의 CPU, 메모리, 네트워크 등의 자원 사용률과 처리량 등을 분석하였다.

2.2.4 로그 수집 자동화

로그 데이터는 디버깅 과정은 물론이고 QA 시점과 운영 시점 등 개발 라이프 사이클의 다양한 단계에서 분석에 사용된다. 하지만 과다한 로그를 기록하게 되면 서버 플랫폼 자체의 성능을 저하시킬 수 있는 단점이 있어서 최소한으로 기록하면서 테스트가 진행되어야 하는 경우 실시간 로그 데이터를 원격지에 취합하기 힘들다.

이때 NTAF을 이용하여 서버의 로그는 물론이고 가상 사용자 프로그램 등 테스트 시 원격지 서버에 기록된 로그 데이터를 분석할 장비로 자동 취합할 수 있다.

그림 8은 테스트 종료 후 원격지 서버에 남아 있는 로그 데이터를 분석할 장비로 카피해 오는 NTAF 명령어 예이다. 위 예는 "MACHINES"에 선언된 각 원격지 테스트 장비의 "C:₩StressTest₩logs" 폴더에 있는 "20090907.log" 파일을 분석할 장비인 "localhost"의 파일 시스템에 복사하여 취합하도록 하는 시나리오이다. 이와 같이 취합한 로그 데이터는 로그 분석기나 그 외의 방법을 이용하여 분석 데이터를 데이터베이스에 입력하고 통계 데이터를 추출한다.

2.3 테스트 자동화의 효과

표 1은 게임 플랫폼의 테스트 자동화로 인하여 테스트 과정 중 주요한 4가지의 단계의 대표적인 세부 업무 항목이 어떻게 변경되었는지를 보여준다.

표 1에서와 같이 원격 테스트 장비에 접속하는 일이나 배포, 관리 등의 업무가 자동화된 것을 확인할 수 있다.

게임 서버 플랫폼의 한 사례를 볼 경우 QA팀이 테스트를 수행한 2개월 간 약 50회 가량의 변경이 발생하였다. 서버 플랫폼의 변경뿐만 아니라 테스트를 위한 프로그램의 변경으로 인해 10여대가 넘는 테스트 장비에 환경을 반복적으로 재설정해야 하는 작업이 요구되었으나 환경 구성을 자동화함으로써 상당한 시간 및 리소스를 절감할 수 있었다.

```
#변수 선언 부
!define DATE {20090907}
!define LOG_DIR {C:₩StressTest₩logs}
!define MACHINES {192.168.2.1, 192.168.2.2, 192.168.2.3}

#사용할 클래스를 import하는 부분
!|com.nhncorp.ntaf.Import|
|ta|
|com.nhncorp.ntaf|

#NTAF의 제어문 사용을 위해서 선언
!|FlowFixture|

#반복문을 위한 선언
|start_iterate|{var: $remote_ip$}, {in: ${MACHINES}}, {indexvar: $i$}|

#$remote_ip$ 변수 값에 따라서 해당 테스트 장비의 특정 파일을 분석할
#장비인 localhost로 복사하여 취합하는 명령어
!|StafCmdFixture|
|service|location|request|submit()|response()|getErrorMessage()|
|FS|$remote_ip$|COPY FILE ${LOG_DIR}₩${DATE}₩StressTest_${DATE}.log TOFILE
C:₩logs₩${DATE}_$remote_ip$.log TOMACHINE localhost|0|||

|end_iterate|
```

그림 8 분산 로그 파일 취합 예

표 1 테스트 자동화 이전과 이후의 업무 비교

테스트 단계	단계 별 대표적 세부 업무 목록	자동화 후	자동화 전
환경 구성	테스트 장비 상태 확인	각 테스트 장비의 IP 목록을 PING 명령어로 자동 체크함	개별 장비에 PING 명령어로 상태 확인 or 원격 접속
	소프트웨어 설정 파일 수정	변수 값에 따라서 자동 생성	수동으로 직접 편집
	배포	원하는 원격지 테스트 장비에 자동 배포	FTP, SVN 등에 업로드 후 원격 접속으로 다운로드
	프로세스 관리	자동 실행&종료	원격 접속 후 실행&종료
테스트 수행	기능 테스트	각 기능별 혹은 시나리오 별로 작성된 Fixture를 통해서 자동 수행	테스터가 개별적인 기능에 대해서 각기 기능을 직접 테스트 함
	성능 테스트	10여대의 테스트 장비를 자동 통제하여 성능 테스트 수행	10여대를 직접 원격 접속하여 가상 테스트 프로그램을 수행하고 관리
결과 분석	로그 분석 출력된 화면 확인	로그 파일의 자동 수집, 로그 분석기를 통한 분석. 화면 출력의 경우 직접 확인함	테스터 혹은 QA 팀이 로그나 테스트 수행된 화면을 매번 분석함
환경 제거	플랫폼, 테스트 프로그램 등의 삭제	자동적으로 모든 테스트 프로그램들을 삭제함	각 원격지 장비를 원격접속 하여 수동으로 삭제 혹은 그대로 방치함

또한 Wiki로 작성된 테스트는 전문적인 지식이 없는 경우에도 쉽게 이해하고 수행할 수 있기에 담당 개발자나 QA의 부재 시 혹은 업무 이관 시에도 테스트 수행이 중단되거나 지연되는 문제가 발생하지 않았다.

테스트 자동화를 위해서는 자동화 이전에 비해서 초기에 많은 자원이 투입되어야 함은 명백하다. Fixture의 제작이나 시나리오 구성 그리고 Wiki 작성 등 이전에는 불필요했던 업무를 추가적으로 수행해야 하나 분명한 것은 잦은 변경으로 인해 다수의 회귀 테스트가 요구되는 경우 기민하게 대처할 수 있음으로 얻을 수 있는 효과가 상당하다는 점이다.

3. 결 론

온라인 게임 서비스를 제공하는 게임 서버 플랫폼의 테스트를 위해서는 분산 환경에서의 테스트가 필수이다. 분산 환경에서 테스트를 효율적으로 수행하기 위해서는 원격지 테스트 장비에 대한 접근이 원활해야 하며 분산된 테스트 장비를 효과적이고 쉽게 관리할 수 있는 방안이 필요하다. 본 논문에서는 테스트 자동화 프레임웍인 NTAF을 활용해서 온라인 게임 서버 플랫폼의 테스트 자동화를 한 방법과 그 사례 및 효과를 기술하였다.

분산 환경에서 "환경 구성", "테스트 수행", "결과 분석", "환경 제거"의 일련의 과정을 자동화함으로써 빈번한 소프트웨어 변경으로 인한 회귀 테스트를 빠르게 수행하였으며 다수의 가상 사용자를 시뮬레이션 하는 테스트를 용이하게 수행하여 게임 서버 플랫폼의 기능적, 비기능적 품질요소 검증의 신뢰도를 높였다.

참고문헌

[1] Eunha Kim, Jong Chae Na, and Seokmoon Ryoo, "Implementing an Effective Test Automation Framework," IEEE DOI 10. 1109/COMPSAC, pp.534-538, 2009.

[2] Rick Mugridge, and Ward Cunningham, Fit for Developing Software : Framework for Integrated Tests, Prentice Hall, 2005.

[3] "Software Testing Automation Framework (STAF)", http://staf.sourceforge.net

[4] "JIRA", http://www.atlassian.com/software/jira/

[5] Gerard Meszaros, xUnit Test Patterns, Addison Wesley, 2007.

[6] "Fixture", http://en.wikipedia.org/wiki/Test_fixture

[7] "NTAF", http://dev.naver.com/projects/ntaf

이 강 희

2007 서강대 정보통신대학원 소프트웨어공학(석사)
2007~현재 NHN, 게임QA실
E-mail : tomcatguru@nhn.com

유 석 문

2000 광주과학기술원 기전공학과(석사)
2008~현재 NHN, 기술선도랩
E-mail : seokmoon.ryoo@nhn.com