볼륨 테스트를 위한 케이스 구조 및 볼륨 증가 패턴

이복연[O] 신석종 전성희
삼성전자 소프트웨어센터
{ wegra.lee[O], sjshin, seongheej}@samsung.com

Test Case Structure and Volume Increment Pattern for Volume Test

Bokyeon Lee[O], Seogjong Shin, Seonghee Jeon
Software Laboratory, Samsung Electronics

요 약

비기능적 테스트 기법 중 하나인 볼륨 테스트의 목적은 테스트 대상 시스템이 명세에 정의된 최대 한도까지 자원을 활용하는 환경에서도 안정적으로 동작하는가를 검증하는 것이다. 본 문서는 이런 볼륨 테스트의 대상과 고려사항을 정의하고, 테스트 우선 순위, 테스트 케이스 구조와 볼륨 증가 패턴 및 그 구현 방법을 상위 레벨에서 기술하여, 재활용 가능한 볼륨 테스트 패턴을 제시한다.

1. 서 론

비기능적(Nonfunctional) 테스트 기법 중 하나인 볼륨(Volume) 테스트[1]는 테스트 대상 시스템(System Under Test)이 명세에 정의된 최대 한도까지 자원(Resource)을 활용하는 환경에서의 정확성(Correctness)과 안정성(Reliability), 성능(Performance) 등의 검증을 목적으로 한다.

볼륨 테스트는 상대적으로 고가용성이 요구되는 시스템이나, 어떠한 어플리케이션이 수행될 지 알 수 없는 플랫폼 입장에서는 반드시 고려되어야 한다. 이러한 관점에서 PC/서버 환경 등은 상대적으로 많이 성숙해 있으나, 최근 급격히 증가하고 있는 모바일 어플리케이션, 홈 네트워크 시스템 등은 도입 단계이기 때문에 관련 결함이 쉽게 발견되는 추세이다. 따라서 볼륨 테스트를 위한 체계적이고 구체적인 가이드라인의 필요성이 대두되고 있으며, 본 논문은 이러한 요구에 대응하여 실제 수행된 과제에 기반하여 재활용 가능한 볼륨 테스트 패턴을 제시한다.

2. 볼륨 테스트 대상 자원

자원의 기본적인 의미가 '소모되는 것'임을 상기하면 어떤 것이 자원에 속하고 어떤 것이 아닌지 판단하는데 도움이 될 것이다. 본 글에서 자원이란, 그 증감에 따라 값(value)의 변화가 아니라 물리적인 양의 변화가 일어나는 개체이다. 따라서 명시적이든 묵시적이든 반드시 최대 한도가 존재하며 최소값은 '0'이라는 특성을 갖게 된다.

이에 따라 볼륨 테스트의 대상이 되는 자원에는 메모리(Stack, Heap, 기타 특수 목적 메모리 등), 프로세스, 스레드, 네트워크 커넥션, 파일, 데이터베이스 커넥션, GUI 컴포넌트, 저장소(Repository, Store 등), 각종 핸들러 등 다양하다.

자원에는 명시적으로 한계치가 정해져 있는 경우와 '서브시스템에서 지원해주는 한계까지'와 같이 불명확한 경우가 있다. 또한 논리적 한계와 물리적 한계가 구분되는 경우도 적지 않다. 예를 들어 32비트 OS는 논리적으로 4GB의 메모리 영역을 활용할 수 있지만 메모리가 제한적인 임베디드 기기에서 물리적 한계에 의해 그보다 적은 메모리만 사용하기도 한다.

3. 테스트 고려사항과 우선순위

본 문서에서는 볼륨 테스트의 주요 고려 사항을 다섯 가지로 제한한다(표 1).

표 1 볼륨 테스트 고려 사항

① 명시된 한계치까지 자원을 활용할 수 있는가?
② 한계를 초과한 자원 요청 시 정의된 예외처리를 하는가?
③ 한계치까지의 자원 활용 중, 오동작이 일어나지 않는가?
④ 한계치까지의 자원 활용 중, 성능 요구사항을 만족하는가?
⑤ 사용된 자원이 적절히 회수되는가?

①~④번 항목은 테스트 우선 순위에 따라 기술된 것이다. 즉, 명시된 한계치 이내까지는 자원 활용이 반드시 보장되어야 하며①, 그 이상 요청이 되면 적절히 예외 처리하여 시스템이 불안정해지는 상황은 방지해야 한다②. 이상 두 항목은 테스트의 가장 기본이 된다. 다음으로, 한계치까지 자원을 활용할 수 있을 뿐 아니라, 그 상태에서도 전체 시스템의 기능적 정확성(Correctness)을 보장하며③, 사용자가 수용할 수 있는 반응 시간(Response Time)을 보장해야 한다④. 마지막 항목 ⑤는 모든 테스트에서 만족되어야 하는 공통 고려 사항이다.

테스트 케이스 작성 면에서 보면, ①, ②번 목적의 테스트 케이스는 구현이 쉽고, 하나의 테스트 케이스에서 동시에 검증해도 크

게 우리가 없다. 반면, ③, ④번 목적의 테스트 케이스는 테스트 대상 자원의 성격에 따라 다양해지며, 하나의 테스트 케이스에서 하나씩 검증하는 것을 권장한다. 물론 ⑤번 항목은 모든 테스트 케이스에서 검증해야 한다.

4. 테스트 케이스 구조

본 절에서는 모든 테스트에서 공통적으로 고려해야 하는 '⑤ 사용된 자원이 적절히 회수되는가?' 항목을 해결하기 위해 다음과 같은 구조의 볼륨 테스트 케이스를 제안한다.

표 2 볼륨 테스트 케이스 구조 (ASN.1에 따름[2])

```
VolumeTestCase ::= SEQUENCE {
  volumeTest      TestCase,
  normalTest      TestCase }
```

위와 같이 볼륨 테스트 케이스는 순차적으로 수행되는 두 개의 독립된 테스트 케이스로 구성되며, 이 중 하나의 테스트 케이스만 실패(Fail)되더라도 전체 볼륨 테스트 케이스는 실패로 처리된다.

앞의 *'volumeTest'*는 볼륨 테스트 고려 항목 ①~④번 중 하나를 검증하기 위한 완전한 형태의 테스트 케이스이다. 완전한 형태라 함은 독립적으로도 수행될 수도 있음을 의미한다.

뒤이어 수행되는 *'normalTest'*는 반드시 짝지어진 *'volumeTest'*와 동일한 자원을 소모해야 하며, 복잡한 시나리오 없이 대상 자원을 한계치까지 얻어낼 수 있는지 검증한다. 즉, 앞서 수행된 *'volumeTest'*가 소모한 자원이 모두 회수되었는지를 검증한다.

참고로, 자원 회수에 시간이 소요되는 경우 두 *'TestCase'* 사이에 충분한 대기 시간을 주어야 한다.

5. 볼륨 증가 패턴

자원의 볼륨 증가 패턴은 크게 선형(linear) 증가와 임의(random) 증가로 구분된다. 전자의 예로는 스레드, 윈도우 객체, 소켓 등이 있으며, 메모리 할당은 후자의 예가 된다. 이상의 두 패턴을 세분화하면 다양한 패턴이 생성되며, 복합적인 패턴들 역시 가능하다. 다음은 생각할 수 있는 대표적인 패턴들이다.

Linear Increasing

초기값에서 최대값까지 일정한 증가값씩 상승한다. 가장 단순한 형태의 볼륨 증가 패턴으로, 가장 정밀하게 모든 값에 대해 검증할 수 있지만 그만큼 많은 시간이 소요된다. 특히 자원의 특성상 선형 증가만 지원되는 경우, *'normalTest'*용 볼륨 증가 방식으로 사용하기 적합하다.

- 입력: 초기값, 최대값, 증가량
- 관련 고려 사항: ①, ②, ③, ④, ⑤

Exponential

지수 형태로 증가한다. Linear Increasing 패턴에 비해 빠르게 최대값에 도달하나, 그만큼 정밀성이 떨어진다. 특히, 최대값 근처에서 급격히 증가하는 특성이 있는데, 일반적으로 최대값 근처에서 실패가 많이 발생한다는 경험적 사실에 비추어, 본 패턴의 단점으로 지적할 수 있다.

- 입력: 초기값, 최대값
- 관련 고려 사항: ①, ②, ⑤

Aggressive (Exponential + Linear)

지수 형태로 증가하다가, 최대값 근처에서 선형 증가한다. Linear Increasing과 Exponential 패턴의 장점을 합친 증가 방식으로, 볼륨 테스트는 한계값 부근에서 오류가 발생할 확률이 높다는 점에 착안하여 만들어졌다. 초기에는 빠르게 볼륨을 증가시킨 후 한계값 근처에서 정밀 검사를 수행한다. 본 논문의 기반 SQA 과제에서 가장 기본적인 볼륨 증가 패턴으로 활용되었다.

- 입력: 초기값, 최대값, 증가 방식 전환점 (명시적으로 지정 또는 초기/최대값 사이의 비율)
- 관련 고려 사항: ①, ②

Converging

'명시되어 있지 않은' 최대값으로 수렴한다. 최대값이 알려지지 않은 경우 사용될 수 있는 패턴으로, 테스트를 수행하며 능동적으로 최대값을 찾아낸다. 내부적으로 지수 증가 모드와 수렴 모드로 구현된다. 전자는 의미 그대로 지수 형태로 볼륨을 증가시키며, 이 모드에서 처음 실패되는 시점에 수렴 모드로 전환된다. 수렴 모드에서는 자원 할당 실패 시, 가장 최근에 성공한 값과 현재 실패한 값의 중간 값으로 재시도한다. 역으로, 성공 시에는 가장 최근에 실패한 값과 현재 성공한 값의 중간 값으로 재시도한다. 이상의 두 과정에서 성공값과 실패값의 차가 지정한 정밀도 내에 들어오면 테스트가 종료된다. 최대값이 명시되지 않았으므로 성공/실패 결정을 위한 범위 설정이 추가로 필요하다.

- 입력: 초기값, 정밀도
- 관련 고려 사항: ②

Random

최소, 최대값 사이의 임의값만큼의 자원을 소모한다. 다른 패턴의 수행들을 마친 후 추가적으로 수행해보거나, 자원 회수가 적절히 일어나는지를 검증할 때 활용할 수 있다. 하지만 데이터 생성이 일정치 않기 때문에 결함 재현이 어려울 수 있다. 다음의 Marginal Stress와 같은 혼합형 패턴의 일부로도 활용된다.

- 입력: 최소값, 최대값, 반복 횟수 (또는 지속 시간)
- 관련 고려 사항: ①, ②, ⑤

Marginal Stress (Converging/Exponential + Random)

지수 증가 방식으로 시작하고 최대값 부근에서 집중적인 부하를 가한다. 이 증가 패턴은 (찾아낸) 최대값 부근에서 집중적인 테스트를 수행하여, 한계상황에서의 시스템 안정성(Reliability) 검증에 적합하다. Aggressive 증가 패턴과 같은 이유에서 시작은 지수 증가 형태이다. 단, 명시적 최대값 존재 여부에 따라 Converging 방식과 Exponential 방식 중 선택할 수 있다.

- 입력: 초기값(, 최대값), Margin, 반복 횟수 (또는 지속 시간)
- 관련 고려 사항: ③, ④, ⑤

Retry

실패 시 일정 횟수만큼 재시도한다. 자원 회수 시 시간이 소요되는 경우 고려해볼 수 있는 방식으로, 앞서 설명한 어떠한 패턴과도 함께 쓰일 수 있다.

- 입력: 재시도 횟수
- 관련 고려 사항: ①, ②, ③, ④, ⑤

6. 구현 가이드라인

앞서 설명한 다양한 볼륨 증가 패턴들을 효율적으로 적용하기 위해, '패턴에 따른 테스트 입력 데이터 생성 로직'을 분리하여 필요 시 불러 쓸 수 있는 라이브러리 형태로 제공할 필요가 있다. 표3은 Ting (Test Input Generator) 라이브러리의 설계 원칙이다.

표 3 Ting 라이브러리 설계 원칙

① 다양한 볼륨 증가 방식을 패턴화
② 모든 패턴에 일관된 인터페이스 적용
③ 개발자에게 익숙한 인터페이스 제공

표4는 이를 사용하는 간단한 예로, 설계 원칙 ③에 의거, 개발자들에게 가장 익숙한 인터페이스 중 하나를 채택했다.

표 4 사용 예 및 핵심 메서드 설명

```
Ting ting = new XxxTing( /* Input Parameters */ );
for ( ting.init(); ting.hasMore(); ting.inc() ) {
  try {
    target.method(ting.get());
  } catch ( Exception ex ) {
    // Test failed.
    break;
  }
  // Test passed.
}
```

init()	초기화
hasMore()	증가 가능 여부 판단
inc()	값 증가
get()	현재값 반환

다음 그림처럼 지금까지 언급한 모든 증가 패턴을 개개의 클래스로 제공하며①, 이들 모두는 최상위의 Ting으로부터 파생되어 통일된 인터페이스를 유지한다②.

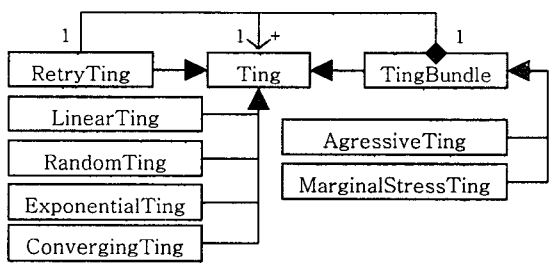

그림 1 Ting 라이브러리 클래스 연관 관계

Linear, Exponential, Random, Converging Ting은 최상위 Ting에서 직접 파생된다. 이들을 묶어 기본 Ting이라 부르자. 다른 Ting들도 물론 최상위 Ting에서 직접 파생시킬 수는 있지만, 자세히 살펴보면 이미 존재하는 기본 Ting들의 조합이라는 특성을 발견할 수 있다. 이런 이유로 두 개 이상의 Ting을 조합해 순차적으로 수행시키는 TingBundle 개념이 도입되었다(표5). 마지막 RetryTing은 종류에 상관 없이 하나의 Ting을 한 겹 감싸고(Wrap), Retry 기능을 덧붙여준다.

표 5 AggressiveTing과 MarginalStressTing의 구조

```
AggressiveTing ::= SEQUENCE {
  subTing1        ExponentialTing,
  subTing2        LinearTing }

MarginalStressTing ::= SEQUENCE {
  subTing1 ::= CHOICE {
    converging    ConvergingTing,
    exponential   ExponentialTing },
  subTing2        RandomTing }
```

7. 응 용

앞서 구현 가이드라인에서 설명한 테스트 입력값 생성 라이브러리(Ting)의 기본 설계는 볼륨 테스트뿐 아니라 단위(Unit) 테스트, 강건성(Robustness) 테스트 중 오류 삽입(Fault Injection) 테스트용으로 곧바로 적용할 수 있다.

단위 테스트용으로는 최상위 Ting에서 곧바로 파생되는 경계값(Boundary Value) Ting과 동치 분할(Equivalence Class Partitioning) Ting을 추가할 수 있다. 미리 지정된(또는 저장한) 값을 차례로 내보내는 List(or Enumeration)Ting 개념도 유용하다. ListTing은 오류 삽입 테스트용으로 쉽게 응용될 수 있다. Carnegie Mellon 대학의 Ballista 프로젝트[3]에서 얻어진 타입별 오류 유발 데이터들을 차례로 반환하는 Ting이 예가 될 수 있다. 또한, 기본 데이터 타입의 특정 bit를 바꿔주는 BitFlipTing, 문자열의 일부를 전환하는 Ting 등도 가능하다.

8. 결 론

볼륨 테스트는 비기능적 요구사항 중에서는 비교적 검증하기 수월한 테스트에 속한다. 하지만 아직까지 체계적이고 구체적인 가이드를 찾아보기란 쉽지 않다.

이러한 상황에서, 본 글은 실제 수행된 프로젝트를 바탕으로, 볼륨 테스트의 대상, 중점 고려 사항, 우선 순위, 테스트 케이스 구조, 볼륨 증가 패턴과 그 구현 가이드를 제시함으로써 향후 유사 과제 진행에 좋은 참고 자료와 지침을 마련하였다.

특히, 제시된 테스트 입력 데이터 생성 라이브러리(Ting)의 개념은 볼륨 테스트에 국한되지 않고, 단위 테스트와 오류 삽입 테스트에도 응용되고 있다.

앞으로 진행될 다양한 성격의 품질 보증 업무들에서는 본 글이 소개한 테스트 패턴을 응용, 시험하여 잘못된 것을 고치고 좋은 점은 보강하여 보다 나은 테스트 지침으로 성장시키기 위한 연구가 계속될 것이다.

9. 참고 문헌

[1] M. R. Lyu, "Handbook of Software Reliability Engineering," Mc-Grw-Hill/IEEE, 1996.

[2] X.680-0207, "Information technology -Abstract Syntax Notation One (ASN.1): Specification of basic notation," ITU-T, 2002.

[3] Phil Koopman, "The Ballista Project: COTS Software Robustness Testing", Carnegie Mellon University, http://www.ece.cmu.edu/~koopman/ballista/