14

실용적인 성능 테스팅

■ 제임스 불 (QA 컨설턴트) 지음

■ 이대엽 옮김

형편없이 수행되는 소프트웨어는 사람들의 삶을 편하게 만들어 주기보다는 사람들을 불쾌하게 만들고 조직에 걸림돌이 된다. 이러한 소프트웨어는 전달된 소프트웨어의 기능적 수준과는 상관없이 사람들이 인식하는 소프트웨어의 품질에 대해 굉장히 부정적인 인상을 준다. 여러분이 근무하는 회사에서 출시한 소프트웨어에 대해 좋지 않은 경험을 한 사람들은 다음 번에는 여러분이 더 잘할 수 있는가를 보려고 기다려 주지 않을 것이며 더 나은 소프트웨어가 있는지 찾아보고 다른 곳에 돈을 쓸 것이다.

확장이 원활하고 빠르고 신뢰할 수 시스템과 그렇지 않은 시스템 중에서 골라보라고 한다면 어떤 시스템을 선택할지는 명백하다. 성능 테스팅에 관해 이야기할 때 보통 우리는 성능 테스팅에 성능은 물론 확장성과 신뢰성도 포함시키는데 그 까닭은 성능과 확장성, 신뢰성에 대한 테스팅이 종종 같은 시간에, 같은 도구를 사용하여 이루어지는 경우도 있기 때문이다. 이 글에서는 우리가 통틀어 성능performance이라고 지칭하는 이 같은 특성들이 완성된 제품에 갖추어져 있음을 어떻게 보장하는가에 관해 이야기한다.

14.1 성능 테스팅이란 무엇인가?

이쯤 되면 아마 앞서 언급했던 소프트웨어의 성능 특성이 그저 좋은 것이 아니라 돈을 들일 만한 가치가 충분히 있다는 데 모두 동의할 것이다. 이제 질문은 다음과 같다. 그렇다면 성능 테스팅에 가장 적합한 곳은 어디인가? 어떻게 이러한 성능 테스팅이 적절한 성능 수준을 갖춘 소프트웨어를 만드는 목표를 달성하는 데 도움을 주는가?

성능 테스팅은 출시된 제품이 충분한 성능을 갖췄음을 보장하는 데 필요한 활동을 모두 포괄해야 한다. 여기에는 요구사항, 제품 성능 측정자료, 의사소통, 프로세스라는 4가지 핵심요소가 있다.

만약 이러한 4가지 요소 가운데 하나라도 누락되면 성능 테스팅의 효과가 상당히 줄어들 것이다. 그리고 여러분이 맡게 된 일이 테스팅밖에 없다면 그리 유리한 입장에 있다고 볼 수 없다. 그 이유는 테스팅을 얼마나 빨리 진행해야 할지에 관해 알려진 바가 없기 때문이다. 이런 이유로 여러분은 요구사항 수집도 해야 한다. 그리고 만약 테스팅 결과가 다른 이들에게 전해지지 않는다면 문제가 있다는 사실을 아무도 모를 것이며, 따라서 아무런 행동도 취할 수 없을 것이다. 또, 요구사항이 수립되고 제품을 테스트했으며 테스팅 결과를 전달했다고 해서 끝나는 것은 아니다. 성능 관련 버그를 수정할 계획을 세울 여유가 없거나 다른 이들과 공유한 테스팅 결과를 토대로 필요한 작업을 계획할 프로세스가 수립되지 않았다면 실제로 출시되는 소프트웨어에 거의 영향을 주지 못한다. 이렇게 되면 종국에는 성능과 관련된 실패나 성공의 정도를 정확히 파악하는 데 상당한 비용이 들게 된다.

우리가 원하는 바는 단순히 성능과 관련된 성공과 실패의 정도를 잘 파악하는 것뿐만 아니라 개발 중인 소프트웨어에도 영향을 줄 수 있도록 정보를 모으는 것이다. 이를 통해 우리는 원하는 성능 수준에 다다르거나 아니면 못해도 그 정도 수준에 근접할 수가 있다. 그럼 지금부터 4가지 핵심요소에 관해 자세히 살펴보자.

14.2 요구사항 수집

요구사항 수집은 종종 과소평가되어 생략되곤 한다. 이 절에서는 우리가 측정하는 것이 무엇이고 어떻게 원하는 것을 알아내는지, 그리고 어떻게 실제로 도움되고 업무 능률을 높이는 데 이바지하는 측정 자료를 찾아내는지에 관해 설명할 것이다.

우리가 측정해야 할 것은 무엇인가?

성능에 대한 핵심적인 측정요소는 최대 처리량maximum throughput과 주어진 처리 수준에서 보여주는 응답시간response time이다. 서로 다른 처리량에 따른 응답 시간을 측정해 보는 것은 시스템의 부하가 응답 시간에 미치는 영향을 알아내는 데 도움이 된다. 목표로 삼은 응답시간이 굉장히 절박한 수준이라면 아마 그와 같은 목표치를 유지하는 동안에는 달성 가능한 처리량이 최대 처리량에 비해 굉장히 낮을 것이다. 여러분은 허용할 만한 수준의 응답시간에서 보여주는 처리량과 원하는 수준의 처리량을 보이면서 수행될 때 얻게 되는 응답시간을 알아내고 싶을 것이다.

확장성scalability에 대한 핵심 측정요소는 데이터 집합의 크기와 사용자 수, 또는 시스템이 수행되고 있는 하드웨어가 달라짐에 따라 초기 성능 측정값이 어떻게 변화하느냐 이다.

신뢰성reliability에 대한 핵심 측정요소는 비정상적으로 높은 부하에서도 시스템이 계속해서 올바르게 기능하느냐와 오랜 시간 수행되는 경우에도 계속해서 올바르게 기능하느냐 이다.

수치를 어떻게 산정할 것인가?

시스템이 주어진 과업을 완수하는 데 필요로 할 처리량을 알아내려면 해당 시스템을 얼마나 많은 사용자가 사용하고, 그러한 사용자들이 해당 시스템

을 사용하는 패턴이 어떠할지 알아야 한다. 사용자가 주어진 기능을 얼마나 자주 수행하는가, 그리고 주어진 기능은 얼마나 빨리 완료되어야 하는가? 와 같은 질문이 이 같은 사항들을 파악하는 데 도움이 될 것이다.

이러한 정보는 현업의 누군가로부터 알아낼 수 있어야 한다. 여러분은 이런 정보들이 정기적으로 필요할 거라는 점을 주지시켜 최소한의 노력을 들이고도 이 같은 정보들을 수집할 수 있도록 프로세스를 수립해야 한다.

여러분은 필요한 정보를 안정적으로 얻을 수 있고 특정 시간에 업무를 지원하는 데 필요한 수치들을 금방 알아낼 수 있는 프로세스를 마련하고 싶을 것이다. 만약 이러한 수치들을 주기적으로 평가하지 않는다면 결국 엉뚱한 목표를 향해 달려가게 될 수도 있다.

현재 필요로 하는 처리량을 결정했다면 이제 응답 시간에 관해 생각해 볼 수 있다. UI의 경우 사용자 인터페이스가 순식간에 응답하길 바라기 때문에 실행되고 있는 기능도 반드시 이 시간 안에 완료되어야 한다고 생각하는 함정에 빠지기 쉽다. UI는 요청이 현재 처리되고 있음을 사용자에게 보여주면서 즉각적으로 응답해야 한다. 반면, 현재 실행되고 있는 기능에 의존하지 않는 애플리케이션의 나머지 부분은 여전히 사용할 수 있어야 한다.

응답 시간 목표는 사용자 인터페이스에 우선해야 하며 낮아야 한다. 응답 시간 목표는 주어진 모든 기능이 그 시간 안에 완료되어야 한다고 기대하게 만들어서는 안 된다.

다음은 상황을 명확하게 판단하기 힘든 경우에 즉시 적용해 볼 수 있는 예를 보여준다.

어떻게 이것을 일반 소프트웨어 개발 프로세스에 접목시키는가?

이상적으로 금주 업무에 대한 성능 요구사항을 결정하기 위한 회의에는 프로젝트 관리자, 성능 관련자, 선임 개발자, 성능 테스터가 참석해야 한다.

개발자는 회의에 참석하여 검토 중인 요구사항 가운데 성능 요구사항에 명백히 부합되지 않거나 터무니없는 것이 있지는 않은지 지적한다. 성능 관련자는 업무에 관한 정보와 지식을 제공하여 요구사항을 결정하는 데 도움을 준다. 프로젝트 관리자는 어떠한 의사결정이 이루어지는지 파악하여 방향을 제시해야 하며, 말할 것도 없이 성능 테스터는 회의에 참석하여 테스팅 대상을 파악할 필요가 있다.

다음으로는 누구와 이런 논의를 해야 할지 결정해야 한다. 중요한 점은 성능 요구사항 결정에 대한 책임을 공유할 연락책이 업무부서에 있어야 한다는 것이다. 이를 통해 고객이 자신이 원하는 바가 무엇인지 파악하고, 개발자도 자신이 원하는 바가 무엇인지 확실히 파악할 수 있다. 그리고 앞서 회의에서 도출한 요구사항을 불가침한 것으로 여겨서는 안 된다. 모든 요구사항과 마찬가지로 그것들도 무엇을 달성할 수 있는가에 관해 고객과 대화를 나눌 수 있는 시발점이어야 한다.

일단 요구사항이 정해지면 이러한 요구사항이 성능 요구사항에 부합한다는 사실을 어떻게 보여줄지에 관해 합의한 다음, 다른 작업 계획 산정과 마찬가지로 이러한 테스트를 진행하기 위한 작업 계획을 산정할 수 있다.

개발자들도 성능 테스팅의 요구사항을 갖고 있지 않는가?

개발자들이 가진 요구사항은 다양하겠지만 실제로 개발자들에게 영향을 주는 것은 특정 코드를 재작업할 필요가 있느냐 하는 것이다. 따라서 개발자들은 재작업할 코드에서 무슨 일이 일어났는지에 관한 추가정보를 필요로 할 것이다. 그런 정보들은 코드 프로파일러code profiler의 출력결과에서부터 스레드 덤프thread dump나 단순 로깅logging까지 매우 다양할 수도 있다. 또, 개발자들은 애플리케이션 서버 대비 데이터베이스 사용량이나 최대 부하peak load가 일어나는 시간 동안의 네트워크 사용량을 알고 싶어할 수도 있다.

이러한 질문에 대한 답을 사전에 모두 구하려고 애쓸 필요는 없다. 그렇

게 하려면 상당한 양의 일을 해야 하기 때문이다. 그렇지만 여러분이 할 수 있는 것은 재작업할 필요가 있는 코드가 있는 곳에서 개발자들이 효율적으로 문제를 해결하는 데 필요한 정보가 어떤 것인지 알아낸 다음, 이를 클라이언트와 함께 조사하여 작업 목록에 추가하는 것이다. 이 시점에서 여러분은 지금부터 모든 테스트에 이렇게 하는 것이 쉬울지, 아니면 이처럼 특별한 경우에만 하는 일회성 작업에 그칠지를 고려해 볼 수 있다.

이런 식으로 개발자들의 요구사항이 요구사항 회의의 안건으로 제시되면 모든 이들이 그러한 사항에 관해 알게 되며, 추후 우선순위를 산정하고 작업 계획을 수립할 때 개발자들의 요구사항이 참작될 것이다. 결국 완료된 성능 테스팅이 양측의 요구사항을 모두 만족하게 될 것이다. 이렇게 되면 고객은 개발 중인 소프트웨어를 신뢰하고, 개발자들은 발생하는 문제들을 조사하여 해결하는 데 도움을 받을 수 있을 것이다.

성능 요구사항에 대한 관련자를 찾을 수 없다면?

만약 성능 요구사항에 대한 관련자를 찾을 수 없다면 몇 가지 위험이 따른다. 첫 번째 위험은 업무에 적합하지 않은 소프트웨어를 만들게 되어 프로젝트가 실패하게 될 것이라는 점이다. 두 번째 위험은 제품의 실질적인 적합성suitability과는 상관없이 고객은 성능 요구사항에 대한 의견을 구하지 않았다고 생각하기 때문에 소프트웨어의 적합성 여부에 합의하지 않을 거라는 점이다. 세 번째 위험은 고객이 요청한 일이 아니기 때문에 개발 팀에서 불필요하다고 생각하는 일을 여러분이 개발 팀에서 하도록 종용함으로써 팀 내 긴장을 조성할 수도 있다는 점이다. 이것은 여러분의 관여가 타당한가 여부와는 관계없이 일어날 수 있는 일이며, 필요한 일이 완료되지 않거나, 아니면 반대로 불필요한 일을 하느라 시간을 허비하는 결과를 초래할 수 있다.

고객이 그다지 전문적이지 않고 우리가 불가능하다고 생각하는 것을 원한다면?

고객이 원하는 제품의 성능 수준이 달성 불가능한 것이거나 달성하자니 수지타산이 맞지 않는 위험도 있다. 여러분은 타당성 있는 질문을 하여 업무에 실제로 필요한 방향으로 대화를 이끌어감으로써 이러한 위험요소가 발생하지 않게 하고 싶을 것이다.

처리량과 관련하여 고려해 볼 질문으로는 다음과 같은 것들이 있다. 일과시간에 처리되는 트랜잭션의 양은 얼마인가? 이러한 트랜잭션들은 어떻게 분산되어 있는가? 트랜잭션은 고르게 발생하는가, 아니면 최고점peak과 최저점trough이 있는가? 매주 금요일 오후에 폭주하는가, 아니면 폭주가 일어나는 특정한 패턴이 없는가?

응답시간과 관련하여 고려해 볼 질문으로는 다음과 같은 것들이 있다. 사용자 인터페이스의 응답시간이 시스템에서 완료할 수 있는 작업 양에 어떤 영향을 주는가? 인터페이스를 실제 수행되는 행위로부터 분리할 수 있는가? 예를 들면 사용자가 데이터를 입력하고 나면 데이터 처리에 오랜 시간이 걸리는 시나리오가 있을지도 모른다. 사용자는 다음 데이터를 입력하기 전까지 데이터 처리가 완료되길 기다리고 싶지는 않을 것이다. 그러므로 데이터 처리가 곧 완료되길 기대하기보다는 인터페이스를 처리과정에서 분리하여 사용자가 인터페이스를 통해 데이터를 계속해서 입력할 수 있게 하면서 시스템에서는 백그라운드에서 이 데이터를 활용할 수 있게 한다.

이런 식으로 우리는 양측이 실질적으로 업무에 도움을 줄 정도의 성능에 초점을 맞추게 하고, 절실히 필요한 것과 단순히 갖게 되면 좋은 것을 구분할 수 있다. 또, 가장 필요로 하는 것이 현재 프로젝트상에 놓인 제약 때문에 달성 가능하지 않거나 또는 비용이 매우 많이 들 것으로 밝혀질 수도 있다. 이 경우 클라이언트는 이러한 요구사항 분석이 완료되지 않았다면 추후에도 가능하겠지만 초기 단계에서 진행 여부를 결정할 수 있다.

가질 수 없는 무언가를 원하는 고객의 안 좋은 점은 최종적으로 만들어진 시스템이 업무 요건에 부합할 정도로 잘 수행되더라도 그 시스템에 실망할 거라는 점이다. 이러한 사항들을 논의하는 것은 두 가지 작용을 할 것이다. 이런 논의를 함으로써 실질적인 업무 요구사항이 밝혀지고, 동시에 클라이언트는 자신이 원하는 바가 무엇인지 알게 된다. 따라서 클라이언트는 자신이 원하는 바를 시스템이 정말로 수행하고 있다면 그 시스템에 만족할 것이다. 결국 클라이언트는 터무니없는 기대를 훨씬 덜 하게 될 것이며, 그래도 만약 시스템에 만족하지 않는다면 아마 그에 맞는 합당한 이유가 있을 것이다.

업무 분석가들도 이러한 요구사항을 수집하게 하면 어떨까?

업무 분석가가 회의에 참석할 필요가 없는 데는 몇 가지 이유가 있다. 첫 번째는 기능적 요구사항 수집은 이미 완료되었을 것이기 때문이다. 두 번째는 업무 분석가가 회의에 참석해 있다고 해서 개발자가 회의에 참석할 필요성이 없어지는 것은 아니기 때문이다. 왜냐하면 개발자들은 성능 문제를 조사하기 위해 요구사항에 대한 의견을 표명해야 할 뿐만 아니라 회의에 참석하여 요구사항 가운데 무언가가 문제를 일으키거나 다른 접근법을 요구할 소지가 있는지를 알려주어야 하기 때문이다. 성능 테스터들은 앞에서 제시한 사항들을 질문하여 회의를 이끌어 나갈 수 있을 것이며, 각 기능들에 대한 테스트가 얼마나 쉽거나 어려울지에 관해서도 이야기할 수 있다. 그러므로 이러한 경우 업무 분석가에게 주어진 시간은 다른 곳에 쓰는 편이 더 나을 것이다.

요약

요구사항 수집의 목적은 업무 목표를 지원하는 데 필요한 제품의 수행 능력을 파악하는 것이다. 이 과정에 고객을 참여시키는 이유는 고객이 해당 업무를 가장 잘 알고 있기 때문이다. 이는 수집하는 요구사항의 정확성을

보장하는 데 도움이 된다. 또한 이런 과정을 통해 고객은 이 영역에 속한 요구사항이 어떤 것인지 분명하게 알게 되는 이점도 누릴 수 있고, 고객들이 최종 시스템에 대해 기대하는 바도 관리할 수 있다.

14.3 테스트 수행하기

이제부터는 수행할 테스트의 종류와 그런 테스트를 언제 수행할지에 관해 간략히 설명할 것이다.

어떤 테스트를 다시 수행할 것인가?

여러분은 사용자가 자주 수행하는 행동들을 모두 테스트하길 바랄 것이다. 이러한 테스트는 처리량과 오류 비율, 응답시간과 같은 측정지표들을 기록해야 한다. 그리고 다음으로는 이런 테스트를 활용하여 좀 더 복합적인 테스트를 만들고, 이상적인 분산환경에서 이것들을 모두 함께 수행하는 테스트도 필요할 것이다. 이러한 테스트들을 바탕으로 여러분은 제품 성능에 관한 정보를 충분히 얻을 수 있을 것이다.

이 같은 과정들을 모두 거치고 나면 이제 이상적인 테스트를 가지고 사용자 수와 데이터 집합의 크기를 바꿔가면서 테스트를 수행하여 제품의 확장성을 확인해 볼 수 있다. 만약 가능하다면 장비의 수를 바꿔가면서 시스템을 실행하여 더 많은 하드웨어가 추가될 때마다 성능이 어떻게 변하는지도 보여주고 싶을 것이다. 이런 테스트를 통해 여러분은 확장성에 관한 정보를 얻게 될 것이다.

마지막으로 시스템에 예상치보다 더 많은 부하를 주어 실패 지점을 확인할 수 있는 테스트도 필요할 것이다. 또한 사용자 분포는 비슷하지만 속도가 더 빠른, 이상적인 테스트에 기반한 테스트도 필요할 것이다. 이러한 테스트들을 장기적으로 수행해 보면 시스템의 신뢰성을 파악할 수 있다.

언제 테스트를 수행해야 하는가?

말할 것도 없이 테스트는 가능한 자주 수행해야 한다. 물론 여기에도 문제는 있다. 성능 테스트 집합은 본연적으로 수행 시 긴 시간이 걸리는 경향이 있다. 특히 신뢰성 테스트가 유용하려면 오랜 시간 동안 수행해야 하므로 포괄적인 성능 테스트를 매번 빌드할 때마다 하기에는 시간이 충분하지 않다. 여러분은 개발자들에게 빠르게 피드백을 제공하면서 제품도 포괄적으로 테스트하길 바랄 것이다.

이 문제에 대한 한 가지 해법은 최신 빌드에 대해 제한적인 성능 테스트 집합을 지속적으로 수행하는 전용 장비를 한 대 마련하는 것이다. 이 장비는 테스트 결과가 이전 빌드에 대한 테스트 결과와 확연히 다르다면 실패하도록 설정되어야 한다. 여기서 얻게 되는 결과는 현실 세계의 성능을 가리키는 지표가 아니라, 개발자가 수행한 무언가가 제품 수행 방식에 중대한 변경을 초래한 것은 아닌지를 재빨리 알려주는 조기 경보 체계일 것이다.

그러므로 전체 테스트 집합은 가능한 자주, 성능 측정에 제약이 없는 환경에서 수행되어야 한다. 이렇게 하는 것은 하루에 몇 번이 될 수도 있고, 또는 그 환경에 접근하는 것이 제한된다면 야간에 테스트 집합을 수행하는 것도 적당한 절충안이 될 수 있다.

신뢰성 테스트는 분명 오랜 시간이 걸리며 종종 다른 테스트가 수행되고 있는 것과 동일한 환경에서 수행되어야 하는 경우도 있다. 이는 주중에는 신뢰성 테스트를 수행할 수 없음을 의미하기 때문에, 매주 주말에 수행하는 것이 별도의 신뢰성 테스트 전용 환경을 갖추지 않고도 바라는 만큼의 효과를 낼 수 있는 방법이다.

어디에서 테스트를 수행해야 하는가?

가능하면 실제 운영 시스템production system을 모방한 시스템을 마련하도록

노력해야 한다. 만약 실제 운영 시스템이 너무 방대해서 이렇게 할 수 없다면, 실제 운영 환경의 일부를 흉내 낸 환경을 구축하여 한 서버가 낼 수 있는 성능 측면에서 절대적인 요구사항 측정지표를 나타내야 한다.

전용 성능 테스팅 환경에 접근할 수 없다면 상황이 다소 어려워질 것이다. 그리고 만약 테스팅 환경을 기능 테스팅 팀과 공유해야 하는 경우라면 성능 테스트를 야간에 수행하는 것이 한 가지 대안이 될 수 있다. 이 경우 성능 및 기능 테스트를 위한 데이터베이스를 별도로 갖추고, 데이터베이스와 그와 같은 테스트 집합의 테스트를 수행하는 가운데 프로그램을 교체해 줄 스크립트를 만들어 두는 편이 좋을 것이다. 이는 두 개의 상충하는 테스트가 서로 방해하지 않고 시스템을 사용한다는 것을 의미한다. 물론 이것은 데스크톱 장비의 테스트에서 애플리케이션을 수행하여 그에 대한 성능 테스트를 개발할 수 있음을 가정하고 있다.

테스트를 야간에 수행할 때 한 가지 주의해야 할 문제는 네트워크 환경이 주간과 차이가 날 수 있다는 점이다. 사람들이 직장에 있지 않을 때에는 네트워크 사용량이 더 적겠지만, 백업이나 기타 야간 배치 프로세스가 네트워크 트래픽에 상당한 영향을 줄 가능성도 있다. 성능 테스트가 수행되고 있는 가운데 네트워크 사용량이 급증하면 테스트 결과에도 영향을 줄 수 있으며, 만약 트래픽을 유발하는 성능 테스트와 배치 작업이 모두 같은 시각에 시작하게 되어 있다면 테스트 결과에는 다른 시간대에 테스트를 수행하지 않고는 알 수 없는 일관성과 관련된 문제가 일어날 수도 있다. 테스트 집합을 평상시 주중에 수행되도록 조정해 보면 시간차로 인해 테스트 결과에 중대한 차이가 일어나는지 알 수 있다. 만약 둘 간에 차이가 크다면 평상시 보여주는 평균 네트워크 트래픽을 시뮬레이션해보거나 두 시간대에서 수행한 테스트 결과 사이에 나타나는 차이점이 일관성을 띠고 있는지 확인하여 단순히 차이점을 고려한 상태에서 테스트 결과를 살펴볼 수도 있다.

경험상 테스트 환경에 대한 경합이 높게 일어나는 곳에서는 해당 시스템을 누가 언제 사용하고, 어느 데이터베이스에 대해 테스트를 수행하느냐와

관련하여 꼼꼼하게 살펴야 한다. 이따금 테스트 환경에서는 테스트가 실패하지만 로컬 장비에서는 통과하는 경우도 여기에 해당된다. 이런 경우 테스트 환경에서 데이터베이스를 교체해 보고, 그 문제가 해결되기 전까지 QA 이외에는 해당 시스템을 사용하지 말아야 한다. 이러한 난관을 비롯하여 환경을 공유하는 것이 테스트 수행 빈도에 상당한 제약을 준다는 점 때문에, 가능하면 이러한 테스트를 수행하기 위한 별도의 환경을 구성해 보는 것도 도움될 것이다. 심지어 이렇게 한다고 해서 테스트를 수행하는 환경이 종종 실제 운영 환경처럼 되지는 않더라도 말이다.

만약 실제 운영 환경과 유사하지 않은 환경과 실제 운영 환경과는 유사하지만 제한된 시간만 이용할 수 있는 것 가운데 하나만 골라야 하는 상황이라면 둘 다 잡아라. 배타적인 환경을 이용하면 구애받지 않고 그 환경에 접근하여 테스트를 개발하고 정기적으로 테스트를 수행할 수 있다. 이것은 지속적인 통합continuous integration 시스템에 포함될 수도 있다. 실제 운영 환경과 유사한 환경을 이용하면 해당 시스템에서 얻게 되는 결과를 좀 더 정기적으로 테스트를 수행할 수 있지만 실제 운영 환경과 유사하지 않은 환경에서 얻게 된 결과와 비교해 볼 수 있다. 이를 통해 여기서 얻게 되는 정보가 시스템의 궁극적인 성능과 어떤 관계에 있는가를 알게 될 것이다.

어떻게 자그마한 테스트 장비의 테스트 결과를 제품과 연계시키는가?

종종 발생하곤 하는 한 가지 문제는 여러분이 마련한 환경이 실제 운영 환경과 동일하지 않다는 것이다. 실제 운영 환경과 전혀 비슷한 구석이 없는 테스트 시스템을 마련한 경우 시스템이 서로 다른 하드웨어에서 어떻게 수행될지에 관한 판단을 거의 내릴 수 없다는 점을 깨닫는 것이 중요하다. 그럼 부득이하게 더 규모가 작은 환경을 이용할 수밖에 없다면 어떻게 하겠는가? 아래에서 시스템을 대표하는 부분을 통해 여기서 말하고자 하는 바를 더 자세히 살펴볼 것이다.

대용량 웹 애플리케이션의 예를 생각해 보자. 시스템의 기본적인 아키텍처는 몇 대의 애플리케이션 서버, 수많은 웹 서버, 그리고 몇몇 데이터베이스 서버로 구성될 것이다. 만약 실제 운영 시스템에 수많은 데이터베이스 서버와 데이터베이스 서버당 2배가 넘는 애플리케이션 서버, 그리고 애플리케이션 서버당 4대의 웹 서버가 있다면, 여러분은 다음과 같은 접근법을 고려해 볼 수도 있다. 먼저 실제 운영 환경의 데이터베이스 서버에 비해 절반 정도의 성능을 내는 데이터베이스 서버 한 대와 실제 운영 환경의 것과 사양이 동일한 애플리케이션 서버 한 대, 그리고 웹 서버 한 대를 구입한다.

그런 다음 애플리케이션 서버와 데이터베이스 조합을 구성하는데, 각 장비의 사양은 상대적으로 같지만 속도는 실제 운영 환경에 비해 절반 정도 수준이어야 한다. 또한, 웹 서버의 처리능력도 충분하지 않다.

이 상태에서 애플리케이션 서버에 직접 조회하면 애플리케이션의 성능이 어떤지 알아낼 수 있다. 이는 각 애플리케이션이 달성할 수 있는 수치를 보여줄 것이다. 그런 다음 단일 웹 서버를 통해 테스트를 수행하는데, 이 시나리오에서는 웹 서버가 병목구간임을 보여줄 것이다. 이러한 테스트를 통해 시스템의 나머지 부분과는 관계되지 않은, 웹 서버에 대한 서버당 성능 수치를 산출할 수 있다. 이 수치를 활용하면 실제 운영 시스템에 들어갈 서로 다른 유형의 장비 비율이 적절한지를 말할 수 있고, 어느 정도 자신 있게 말할 수 있는 실제 운영 시스템의 성능과 관련된 추정치를 뽑을 수 있을 것이다.

한 가지 기억해야 할 것은 장비의 수가 늘어남에 따라 모든 웹 요청이 단일 애플리케이션 서버/데이터베이스 조합에 의해서만 처리된다고 한다면 이것의 직접적인 결과로 처리량만 늘어날 거라는 점이다. 시스템의 부하 수준이 동일하다고 가정하면 응답시간은 CPU 처리능력이나 장비당 가용 메모리의 양이 늘어나는 경우에만 개선될 것이다. 이것은 CPU가 더 빨라지면 더 많은 요청을 더 빨리 처리할 것이며, 메모리가 더 많아지면 더 많은 항목을 캐싱할 수 있기 때문이다.

이는 물론 장비가 동일하고, CPU가 모두 같은 제조업체에서 만든 것이며, 모두 동일한 버전의 OS와 동일한 데이터베이스/웹 서버/애플리케이션 서버 조합에서 수행되고 있다고 가정한다.

그리고 사용된 장비나 소프트웨어 등 사양 측면에서 시스템이 실제 운영 환경과 멀어지면 멀어질수록 최종 시스템의 성능에 관해 여러분이 추정한 사항들을 점점 더 신뢰하기 힘들어질 거라는 점을 명심해야 한다. 앞의 예제에서 여러분은 완전한 실제 운영 환경 구성에 비해 극히 적은 비용으로도 테스트 환경을 구성할 수도 있었으며, 최종 시스템에서 예상된 수치들이 그리 많은 차이가 나지 않을 것으로 확신할 수도 있었을 것이다. 그러나 만약 Oracle 대신 MySQL, 또는 JBoss 대신 WebSphere가 사용된 전혀 사양이 다른 장비를 갖고 있다면 비록 여러분이 성능 변화를 측정할 수는 있더라도 예측은 미덥지 못할 것이다.

성능 테스팅에 적합한 데이터베이스 크기는 얼마인가?

성능 테스팅을 수행할 경우 데이터베이스의 크기가 테이블에서 행을 가져오는 데 드는 시간에 상당한 영향을 줄 수 있다는 사실에 주의해야 한다. 만약 테이블에 인덱스가 적절히 지정되어 있지 않을 경우 테스트 집합이 작다면 문제가 분명하게 드러나지 않을 수도 있다. 그러나 실제 운영 데이터가 수천 행에 이른다면 성능이 급격히 떨어질 것이다.

여러분은 실제 운영 데이터베이스의 사본을 마련하여 사본에 대해 다시 한 번 코드를 테스트할 수 있도록 성능 관련자와 상의해야 한다. 사본에 대해 테스트를 수행할 경우 데이터 보호 문제와 관련하여 사용할 데이터베이스에 포함된 개인정보를 적절히 초기화하거나 제거하는 것이 중요하다.

또한 성능 관련자와 함께 저장되는 데이터의 양이 어떻게 변화할지에 대해서도 논의해야 한다. 대략 데이터의 양이 고정되어 있을 것인가, 아니면 증가할 것인가? 만약 데이터의 양이 증가한다면, 느리게 증가할 것인가, 아니면 급격히 증가할 것인가? 이러한 사항들을 파악해 두면 현재 준비된

데이터베이스보다 훨씬 더 큰 데이터베이스를 이용하여 테스트하는 것이 적절한가를 결정하는 데 도움될 것이다.

매우 큰 데이터베이스를 마련하는 가장 좋은 방법은 안정성 테스트를 이용하여 새로운 데이터베이스를 만들어 내는 것이다. 여러분은 안정성 테스트의 일환으로 새로운 사용자와 새로운 트랜잭션을 만들어야 할 것이며, 시스템이 성공적으로 주말 내내 수행되었다면 차후에 활용할 수 있는 상당히 큰 데이터 집합이 마련될 것이다.

써드파티 인터페이스는 어떻게 다룰 것인가?

시스템에 존재하는 써드파티 인터페이스의 수가 많다면 이러한 시스템에 직접 조회하는 것은 그리 좋은 생각이 아니다. 여기엔 두 가지 이유가 있다. 첫 번째 이유는 아마 써드파티가 성능 테스트에 기꺼이 참여하지는 않을 것이기 때문이다. 두 번째 이유는 써드파티에서 테스트 환경을 제공하더라도 여러분이 통제할 수 없는 써드파티에 의존한다는 점 때문에 테스트의 신뢰성이 떨어질 거라는 데 있다.

이런 경우에는 테스트를 수행하여 이 시스템이 평균적으로 얼마나 빠르게 응답하는가를 알아낸 다음, 모의 객체mock나 스텁stub을 만들어 일정한 응답을 반환하기 훨씬 전부터 단순히 테스트를 기다리게 하는 편이 가장 좋다. 응답을 즉시 반환하게 할 수도 있겠지만, 애플리케이션 서버는 다른 것보다 더 빠를 수 없는 데이터베이스 연결이나 네트워크 연결을 거쳐 동작할 수도 있으므로 시나리오에 어느 정도 현실성이 떨어질 수 있으며, 이로 인해 최종 테스트 결과가 달라질 수도 있다.

얼마나 다양한 테스트 케이스가 필요한가?

이 질문이 매우 중요한 이유는 데이터의 양을 잘못 산정하면 사용하는 테스트 케이스가 많고 적음에 따라 테스트 결과가 심하게 왜곡될 것이기 때

문이다. 테스트 케이스를 너무 적게 사용할 경우 관련된 모든 정보가 캐싱될 것이며, 따라서 테스트 결과는 시스템이 실제보다 더 빠르다고 보여줄 것이다. 만약 사용하는 테스트 케이스가 너무 많다면 캐시를 폭발적으로 사용하게 되어 시스템이 일반적인 연산에서 보여주는 것보다 더 느린 것으로 나타날 것이다.

적절한 수의 테스트 케이스를 사용하기 위해서는 예상되는 시스템의 사용 패턴에 관해 성능 관련자와 논의하고, 가능하다면 현행 시스템의 사용 로그를 구해야 할 것이다. 예를 들어 애플리케이션에서 고객 정보를 조회하고 있다면, 조회하는 정보를 포함하는 고객의 수는 분명 일반적인 연산에서 보게 되는 고유 고객 수와 비슷하게 나타나야 한다. 만약 특정일에 시스템에서 조회되는 레코드가 전체 대비 5%라면 테스트 케이스에서 조회되는 고객의 수도 이 정도가 되어야 한다.

응답 시간과 처리량에 대해 여러 측정수단을 취하는 이유는 무엇인가?

일반적으로 유휴 상태에서 부하를 늘리기 시작하면 시스템의 응답 시간은 내려가지 않을 것이다. 계속해서 부하가 늘어나면 단위 시간당 처리되는 트랜잭션의 총계가 증가하는(다시 말하면, 처리량이 증가하는) 지점이 있을 것이나, 이는 응답 시간을 희생하여 나타난 결과일 뿐 응답시간 또한 증가하기 시작할 것이다. 서버가 최대 처리 능력에 도달하게 되면, 처리량은 처음으로 일정한 상태를 유지하는 반면, 응답시간은 최종적으로 처리량 자체가 떨어지기 전까지 눈에 띄게 증가하기 시작하는데 이것은 단순히 장비가 요청된 작업의 양을 따라가지 못하기 때문이다. 이쯤에서 응답 시간은 급속히 올라갈 것이며, 전체 시스템은 완전히 정지상태에 머무르게 될 것이다.

여러분은 다음과 같은 정보에 관심이 있을 것이다. 가장 먼저 알고 싶은 것은 시스템의 최대 처리량이 어느 지점에서 나타나느냐일 것이다. 그 밖

의 흥미로운 정보로는 응답 시간이 목표치에 부합할 때의 부하 수준, 달성 가능한 최적의 응답 시간, 그리고 이전에 측정된 최대 처리량의 80% 및 90% 수준에서 보여주는 응답 시간 등이 있다.

이러한 정보들을 활용하여 요구사항을 수집하는 동안 합의한 범위 안에서 시스템의 성능 특징을 유지하기 위해 애플리케이션 서버에서 받아들일 접속 수를 제한할 수 있다. 여러분은 최대 부하에 도달할 때 응답 시간의 변동폭이 극적으로 증가하고, 처리능력이 80%나 90%일 때 변동폭이 두드러지게 낮다는 사실을 알게 될 것이다. 여러분이 일정 수준의 성능을 보장해야 한다면 이러한 사항을 염두에 둘 필요가 있다.

시스템의 모든 기능을 테스트해야 하는가?

시스템의 모든 기능을 하나하나 테스트하는 것은 타당성이 매우 낮다고 볼 수 있다. 비록 그렇더라도 가장 자주 사용되는 기능에 대해서는 확실히 테스트를 수행해야 한다. 이를 위해서는 시스템이 사용되는 주경로 major way를 파악하여 각 시나리오에 대한 다양한 테스트를 만들어야 할 필요가 있다.

예를 들어 온라인 쇼핑 사이트에서는 주된 사용 패턴이 탐색과 구매일 것이다. 그렇지만 물건을 구입하려고 방문하는 사람이라고 해서 노력이 많이 드는 탐색 과정을 거친 다음 꼭 물건을 구입하는 것은 아니며, 그리고 모든 이들이 오랜 시간 동안 물건을 탐색하는 것도 아니다. 여러분이 해야 할 일은 탐색용 스크립트 하나와 구매용 스크립트 하나를 만드는 것이다. 이러한 스크립트가 현실성을 띠도록 만드는 데 필요한 정보로는 탐색자가 평균적으로 탐색하는 품목의 개수와 평균적으로 주문에 포함되는 품목의 개수, 하루에 사이트를 방문하는 모든 사용자가 탐색하는 모든 품목의 총 비율이 있다.

요약

성능 테스팅을 하는 동안에는 수많은 의문이 생길 수 있다. 무엇을 측정해야 하는가? 얼마나 자주? 스크립트의 개수는? 데이터의 양은? 가장 중요한 것은 이러한 의문사항들이 모든 이들에게 전해지고 필요한 정보를 수집할 수 있는 회의에서 성능관련자와 정기적으로 논의하는 것이다. 뿐만 아니라 프로젝트의 진행상황이 이러한 테스트의 효율성에 중대한 영향을 줄 것이라 생각한다면 프로젝트 관리자 및 성능관련자와 논의할 자리도 마련해야 할 것이다.

14.4 의사소통

다른 이들에게 테스트 결과를 전달하는 것은 중요하다. 그렇다면 다음과 같은 질문을 할 수 있다. 우리가 전달하는 것은 정확히 무엇인가? 테스트 결과를 전달한다는 것은 그저 결과값을 있는 그대로 보고하는 것 이상의 일이다. 만약 여러분이 단순히 결과값만을 보고하는 수준에 머무른다면, 이는 팀원들 모두가 다른 해야 할 일이 있을 때 테스트 결과 분석에 시간을 할애해 달라고 요청하는 것과 같다. 그러나 만약 테스트 결과에 기본적인 분석 및 해석, 요약정보가 제시된다고 가정한다면 모든 이들이 테스트 결과를 쉽게 이해할 수 있을 것이다.

따라서 합의된 요구사항과 현 성능수준을 결부시켜 결과를 해석할 필요가 있다. 먼저 여러분은 성능이 목표치에 얼마나 근접한지, 더 빠르고 느린지에 관해 보고하고 싶을 것이다. 두 번째로는 실제 운영 환경 측면에서 성능이 눈에 띄게 변화하는지를 보고하고자 할 것이다. 여러분은 이러한 변화가 성능 목표에서 벗어나게 만드는지를 널리 알리고 싶을 것이다. 이런 변화는 제품에 덩치가 큰 기능이 추가되어 불가피하게 제품 성능에 영향을 준 것처럼 할 수 있는 일이 많지 않은 경우일 수도 있다. 반면에 데이터베이스의 인덱스를 빠뜨린 경우처럼 쉽게 고칠 수 있는 경우일 수도 있다.

누가 알아야 하는가?

테스트 결과를 전달받아야 할 사람에는 세 부류가 있는데, 바로 개발자와 프로젝트 관리자, 그리고 클라이언트이다. 개발자와 프로젝트 관리자는 문제가 발생하는 즉시 팀에서 그것을 적절히 처리할 수 있도록 테스트가 수행되자마자 결과를 확인해야 한다. 사소한 문제를 가지고 매일 클라이언트를 번거롭게 할 필요는 없지만, 그렇게 하지 않는다면 클라이언트에게 말해야 할 무언가 중요한 사안이 있을 때 클라이언트는 기꺼이 귀담아 들으려 하지 않을 것이다. 하지만 여러분은 이러한 점을 소홀히 하고 싶지는 않을 것이며, 따라서 결과를 검토할 수 있을 때 1주일에 한 차례 회의 일정을 잡는 것이 아마 좋은 생각일 것이다.

또, 서로 다른 사람들은 관심을 갖는 정보도 각기 다르다는 사실을 고려하는 것도 도움이 된다. 클라이언트나 프로젝트 관리자는 상위 수준 관점의 정보를 보고 싶어 하는 반면, 개발자는 주어진 기간 내의 응답수 등 가공되지 않은 자료에 더 관심이 있을 것이다. 만약 여러분이 적절한 정보를 적절한 사람에게 제공할 수 있다면 제품의 상태를 전해주는 일은 훨씬 더 쉬워질 것이다.

그렇다면 여러분이 해야 할 일은 보고서를 작성하는 것뿐인가?

전혀 그렇지 않다. 단순히 보고서를 작성하여 회람시킨다면 대부분의 사람들은 보고서를 읽지 않을 것이며, 따라서 여러분이 전하고자 하는 정보가 사라질 거라는 문제가 발생한다. 여러분이 작성하는 모든 보고서는 여러분이 전하고자 하는 메시지를 알리는 데 도움을 주는 수단에 불과할 뿐, 보고서를 작성하는 것으로 끝난다는 것을 의미하지는 않는다.

사람들이 최근 성능 테스트 결과를 확인해 볼 수 있는 웹 사이트를 갖추면 유용할 것이다. 그렇게 하면 여러분이 누군가의 자리로 가서 테스트 결과를 알려줄 때, 웹 페이지를 열어 테스트 결과를 통해 알아낸 사항을 보여줄 수 있다. 그러나 대부분의 사람들은 테스트 보고서를 읽을 때 크게 관심

을 갖지 않기 때문에, 여러분에게 전하고자 하는 메시지가 있다는 사실을 확실하게 알리는 유일한 방법은 그 사람들의 자리로 가거나 전화로 그 사람과 함께 보고서를 검토하는 것뿐이다.

요약

여러분의 목표는 누구나 여러분의 요구사항이 무엇인지는 알고 있기 때문에 매번 테스트 결과가 나올 때마다 클라이언트에게 테스트 결과가 만족스러운지 문의할 필요가 없는 상태에 이르는 것이다. 프로젝트 진척 상황을 검토하기 위해 주 단위로 회의를 할 경우 여러분은 테스트 결과에서 이례적이거나 비정상적인 사항들을 지적하고 그것들을 설명할 수 있기를 바랄 것이다. 만약 소프트웨어가 특정 영역에서만 유독 상태가 좋지 않지만 그것이 심각한 문제로 평가되지 않았다면 여러분은 회의에 참석한 이들에게 소프트웨어가 느린 이유와 프로젝트 관리자가 그것의 우선순위가 높지 않다고 생각한 이유를 설명하고 그 원인을 참석자들에게 제시할 수 있어야 한다. 만약 회의에 참석한 이들이 그와 같은 결정에 동의하지 않는다면 이번에는 프로젝트 관리자와 고객이 마주앉아 현재 상황에 관해 논의해야 한다.

14.5 프로세스

성능 테스팅은 프로젝트가 끝날 때까지 미뤄지기도 하는데 이렇게 되면 성능 테스팅의 효율성에 심각한 영향을 주게 된다. 성능 테스팅을 수행하는 것에 관한 가장 중요한 사항은 정기적으로 성능 테스팅을 수행하는 것이다. 만약 프로젝트가 끝나기 몇 주 전까지 성능 테스팅을 미룬다면 짧은 프로젝트 기간에도 해야 할 일이 상당히 많을 것이다. 그렇게 되면 대부분의 시간을 스크립트를 작성하고 제품에서 몇몇 수치를 알아내는 데 보내게 될

것이다. 결국 시스템의 속도에 관해 대충 파악하게 될 것이며, 현 상태에 대해 거의 아는 바가 없더라도 필요할지도 모를 변화를 줄 시간이 없을 것이다.

성능 테스팅은 코드를 작성하기 시작할 때 시작되어야 한다. 비록 아직까지는 테스트할 수 있는 것이 없을지도 모르지만 그래도 여러분이 할 수 있는 일은 많다. 여러분은 개발자에게 그들이 사용할 기술에 관해 알려주고, 적절한 도구를 평가하고, 제품을 테스트하는 데 필요하게 될 기능을 제공하는 사람을 찾아볼 수 있다. 뿐만 아니라 성능 관련자를 찾아 함께 요구사항 수집 프로세스를 시작할 수도 있다.

그럼 어떻게 그것들을 연계하는가?

이때부터 여러분은 일정한 주기cycle로 진입할 수 있다. 한 주가 시작될 때 여러분은 처음으로 성능 관련자를 만나 회의를 하게 된다. 이 회의는 현재 구현 중인 기능에 대한 요구사항을 논의하고, 여러분이 만들 테스트와 어떻게 그런 테스트가 요구사항이 충족됐는지를 보여줄 수 있는지도 설명할 수 있는 자리이다. 뿐만 아니라 고객은 이 때 추가적인 테스트를 요청할 수도 있다. 그 주의 나머지 시간 동안에는 가장 최근에 완료된 기능에 대한 테스트를 구현하고 자동화 테스트를 유지보수하며, 테스트 결과를 확인할 수 있을 것이다. 한 주가 끝날 때쯤이면 성능 관련자와 다시 한 번 회의를 갖게 된다. 두 번째 회의의 목적은 두 가지로 압축된다. 첫 번째는 한 주 동안 작성한 테스트를 성능관련자에게 보여주는 것이다. 그러고 나서 제품이 앞서 논의한 요구사항에 부합하고 있음을 새로운 테스트에서 실제로 보여주는지를 클라이언트와 논의할 수 있다. 이 회의의 두 번째 목적은 현행 성능 테스트의 결과를 검토하는 것이다.

주기에 뒤처지지 않았는지 어떻게 확인할 것인가?

주 단위로 일하기 때문에 주기에 뒤처지고 있는지는 순식간에 드러난다. 다시 주기에 맞춰 가려면 성능 테스팅에 할당된 자원을 늘리거나 시도해 보고자 하는 작업량을 줄일 수 있다. 이들 중 무엇을 선택하느냐는 전적으로 이 프로젝트에서 성능 요구사항이 얼마나 중요한가에 달려있다.

이렇게 하는 한 방법은 클라이언트와 함께 결정했던 테스트를 기준으로 필요한 작업 목록을 만드는 것이다. 작업 목록을 만들고 나면 클라이언트와 테스트 목록을 검토하여 그것의 우선순위를 정할 수 있다. 그러고 나서 한 주 동안 할 수 있는 만큼 테스트를 수행한 다음 계속해서 진행하면 된다. 만약 이러한 접근법으로 인해 테스트 커버리지test coverage가 고르게 분포되지 않는다면 성능 테스팅에 더 많은 노력이 필요한 것일지도 모른다. 그러나 가장 어렵고 우선순위가 낮은 테스트는 배제함으로써 테스팅 주기에 뒤처지지 않고도 적당한 수준의 테스트 커버리지를 제공하는 테스트 집합을 개발할 수 있을 것이다.

식별된 문제에 대해 조치가 취해졌는지 어떻게 보장할 것인가?

여기서는 프로젝트가 시작될 때 성능과 관련된 수정사항을 어떻게 처리할지에 관해 프로젝트 관리자와 논의하는 것이 중요하다. 여러분은 프로젝트 관리자가 이러한 접근법에 합의하고 여러분이 수집하는 요구사항이 타당하다고 생각하는지 확인할 필요가 있다. 여러분은 프로젝트 관리자가 성능 이슈를 버그로 제기하길 바라는지를 비롯하여, 프로젝트 종반에 처리해야 할 알려진 문제의 수가 상당히 많아지는 상황에 처하지 않도록 성능 문제가 발생할 때 즉시 그것들을 처리하길 바라는지 확인해 두고 싶을 것이다. 결국, 문제를 처리해야 할 때 그것을 처리하기 위한 행동을 아무것도 취하지 않는다면 현재 성능이 받아들일 만한가를 확인하는 것은 불필요한 일이다.

14.6 요약

이러한 프로세스의 주된 이점은 여러분이 필요로 하는 것과 실제로 갖고 있는 것이 무엇인지를 파악하게 되고, 시스템의 모든 부분에 대해 테스트를 갖추고 있음을 확신할 수 있다는 것이다. 이렇게 되면 어떠한 문제가 발생하더라도 그것들을 처리할 가능성이 매우 높아진다. 각각의 기능이 개발될 때마다 그것을 테스트할 수 있다는 사실은 기능을 변경해야 할 때 그것을 변경할 시간을 확보할 수 있음을 의미한다. 여러분에게 요구사항이 있다는 사실은 변경이 필요한지를 여러분이 알고 있음을 뜻한다. 요구사항이 클라이언트에게서 나오고 그것들이 클라이언트의 업무 프로세스와 업무의 양에 기반한다는 사실은 팀 전체가 그러한 요구사항에 확신을 갖고 있음을 의미하며, 이는 곧 사람들이 성능과 관련된 버그를 수정하는 데 필요한 시간을 기꺼이 보낼거라는 점을 의미한다. 그들은 성능과 관련된 버그를 수정하는 것이 매우 가치 있는 일이라는 사실을 알고 있기 때문이다.