

Web Site Performance Bottleneck Checklist Top 25+

개요

이 문서는 IBE 컨설턴트들의 수 많은 테스트 경험에 근거하여 발견한, 일반적인 웹 애플리케이션 퍼포먼스 Bottleneck에 대해 말하고 있습니다. 그리고 이러한 Bottleneck을 찾아내는 방법에 대해서도 가이드를 제공하며 다양한 웹 애플리케이션에서 나타나는 Bottleneck을 용이하게 찾아내기 위한 지침이 되는 것을 목적으로 일반적인 문제들을 분류하였습니다.

퍼포먼스 Bottleneck은 모든 애플리케이션에 존재합니다. 그것들을 IBE의 e-TEST suite과 같은 툴이나 테스트 서비스를 이용하여 애플리케이션이 실제 출시하기전에, 발견하여 수정하기 위해서는 애플리케이션 개발팀의 협력이 중요합니다. IBE 고객들은 이 툴을 이용하여 수집되는 결과를 최대한 이용하여 그들의 웹사이트의 성능을 현격히 향상시킬 수 있습니다.

Bottleneck이란 무엇인가?

테스트 대상이 될 수 있는 모든 웹 시스템에는 퍼포먼스 Bottleneck이 존재합니다. 그러면 이 Bottleneck이란 무엇인지에 대해 아래 내용에서 이러한 원인이나 결과 또는 문제 해결에 대해 표현될 때 사용되는 용어를 살펴보겠습니다.

Bottleneck - 테스트중인 시스템 자원에 대해 한계가 나타났을 때 사용됩니다. Bottleneck은 일반적으로 허용된 범위 외의 퍼포먼스 및 느려지는 원인이 되는 것을 말합니다. 이것은 테스트 하에 있는 시스템의 성능과 확장성 양쪽 모두에 직접 작용합니다. 그리고 단지 한번의 테스트로 하나의 Bottleneck이 특정 트랜잭션 처리 문제로 인해 시스템 성능이 떨어진다고 말할 수 있으므로 유념할 필요가 있습니다.

성능 - 시스템 테스트에 있어, 일반적으로 테스트에 앞서 사전에 정의된 요구 조건에 맞는 시스템의 능력을 나타냅니다. 시스템의 테스트 결과를 미리 설정된 기준에 비추어 성능이 좋다 혹은 나쁘다고 평가합니다. 또, 단순하게 동시 실행 가상 유저의 수, 하루 동안의 트랜잭션수, 평균 응답시간등을 퍼포먼스라고 표현하기도 합니다.

확장성 - 동시 접속자의 수, 또는 리퀘스트 수가 증가 할때 시스템이 허용 가능한 능력을 표현할 경우에 이용됩니다. 시스템이 지정된 수의 유저수나 리퀘스트에 문제 없이 대응할 수 있는 경우 그 시스템은 좋은 확장성을 가지고 있다고 표현할 수 있습니다.

징후 혹은 문제 - 징후 또는 문제는 테스트에 대해 시스템의 Bottleneck이 어떻게 나타나는지를 나타냅니다. 징후는 드물게는 Bottleneck이 원인 그 자체인 경우도 있습니다. Bottleneck의 원인을 찾아내기 위해서 징후는 다른 데이터와 함께 사용됩니다. 예를 들면 응답 시간의 늦은 페이지, 에러, 서버 다운등의 현상등이 있습니다.

Bottleneck의 원인 혹은 지표 - 테스트중에 나타나는 시스템 문제의 원인이 되는 포인트의 하나로 퍼포먼스의 저하가 있습니다.

솔루션 - 이 말은 Bottleneck 문제를 수정, 해결 하는 것을 표현할 경우에 이용됩니다.

계속 포인트 - 언제, 어디에 Bottleneck이 존재했는지를 결정하기 위해서, 특정 계측 기준을 정해 둘 필요가 있습니다. 예를 들어 CPU 사용율이나 사용 가능한 메모리 용량, 8초 룰의 응답 시간, 에러 발생을 등입니다.

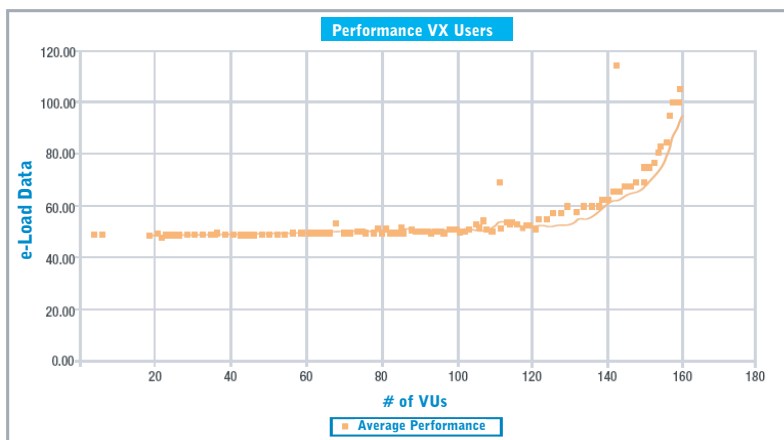


그림 1 : 징후를 측정하는 계속 포인트의 예

고객들은 IBE 툴을 이용하여 수집되는 결과를 최대한 이용하여 그들의 웹사이트의 성능을 현격히 향상시킬 수 있습니다.

왜 bottleneck은 존재 하는가?

시스템에 Bottleneck이 존재하는 원인은 아래에서 보여주듯이 여러가지의 경우가 있습니다.

- 요구되는 사양에 대한 부적절한 스펙의 하드웨어 구성
- 요구되는 사양에 대한 부적절한 대역의 네트워크 구성(인터넷/인트라넷)
- 시스템 아키텍처 및 디자인에 의한 요인
- 부적절한 데이터베이스 모델 및 실행, 튜닝의 부족
- 애플리케이션 개발시 퍼포먼스에 대한 고려 부족
- 프로젝트 진행시 시스템 퍼포먼스의 전문적 지식의 부족
- 불충분한 개발 스케줄 및 요구 사항 변경에 의한 요인

이 리스트는 정정되어야 할 것들 그 자체를 있는 그대로 말하고 있습니다. 그러나 시스템 개발 동안에 이러한 것을 간과하는 경우가 많습니다. 또, 뛰어난 디자인을 바탕으로 설계, 구축, 실행된 시스템이 단위 테스트 및 시스템 테스트에 통과를 했다고 하더라도, 단순한 버그가 아닌 문제를 포함하고 있는 일이 있습니다. 부하 테스트는 그러한 성능에 관련되는 문제를 애플리케이션이 릴리즈 되기전에 확인, 분리 그리고 해결하기 위한 하나의 수단이라고 할 수 있습니다.

그리고, 기억에 남겨 두어야 하는것 중 하나로 무한의 확장성을 가지는 시스템은 존재하지 않는다는 것입니다. 테스트 담당자는 비즈니스 유저나 시스템의 오너로부터의 피드백을 통해, 테스트하는 시스템이 성능 요건을 채우고 있는지 어떤지를 평가, 결정하기 위한 기준을 제공합니다. 또 성능에 관련된 개발 담당자는, 도입 직후의 성능 요구와 도입 후의 성장에 수반하는 성능 요구를 비교해 이해해 둘 필요가 있습니다. 개발 시점에서의 평가 기준을 깨끗이 통과한 시스템을 릴리즈 했을 경우에도 시간과 함께 비즈니스가 성장해 가면서 시스템에 대한 리퀘스트가 증가하는 일은 자주 발생합니다. 설계, 실행 되었을 때와 같이 항상 시스템의 피크나 한계점이라고 하는 시스템의 레벨을 파악하는 것이 필요합니다. 이 성능 허용 레벨은 동시 처리 가능 유저수, 초당 트랜잭션수, 일일 매출 또는 그 외 비즈니스와 관련 있는 애플리케이션을 지원하는 지표에 의해서 정의되어야 할 것입니다.

시스템 퍼포먼스 Bottleneck의 식별과 분리

개 요

Bottleneck은 시스템 아키텍처의 다양한 장소에서 존재하고 있을 가능성이 있습니다. 우리는 경험에 의해 그러한 Bottleneck이 대부분 어디에 존재하는지를 알 수 있었습니다. 웹 시스템으로부터 발견되는 문제의 상당수는 개발자에 의해서 기술된 코드, 또는 데이터베이스와 주로 관련되어 있습니다. 서버의 설정, 네트워크의 셋업은 설정 자체가 복잡하게 되는 것은 적고 컴퍼넌트에 포함되는 "알려지지 않은" 설정 항목도 그다지 많지 않아 문제가 될 가능성이 비교적 낮다고 말할 수 있습니다.

가장 일반적으로 볼 수 있는 문제 영역 :

- 데이터베이스 커넥션과 쿼리
- 애플리케이션 서버 코드
- 웹 서버 하드웨어
- 네트워크

이 부분이 다른 Bottleneck의 요인과 비교하여 그 가능성이 높다고 하는 것을 기억해 둘 필요가 있습니다. 그리고 한 번의 테스트에서는 하나의 Bottleneck을 발견하는 경우가 많아, 테스트와 애플리케이션 테스트에 대한 수정의 사이클을 반복해서 실행할 필요가 있습니다. 이것은 Bottleneck을 찾는 것을 돕는 중요한 아이디어입니다. 이것을 항상 의식하여 Bottleneck을 일으키지 않은 요소를 찾는데 시간을 줄이고 보다 빠른 시간에 성능을 나쁘게 만들고 있는 주요한 원인을 찾는것이 바람직합니다. 여기서 key Point는 한 번의 테스트로부터 하나의 Bottleneck이 밝혀진다고 하는 것입니다. 특정 Bottleneck을 수정하면 또 다른 Bottleneck에 직면하게 될 것입니다. 이 반복적인 프로세스를 통해 성능과 확장성 테스트를 형성합니다.

임의의 한 시점에서, 하나의 Bottleneck이 시스템의 특정 처리를 늦게 만드는 원인이 되기도 합니다.

다음의 그래프는 IBE의 컨설턴트 및 엔지니어가 테스트한 결과로부터 얻을 수 있었던 시스템에 Bottleneck이 존재하는 부분에 대한 비율을 나타내고 있습니다.

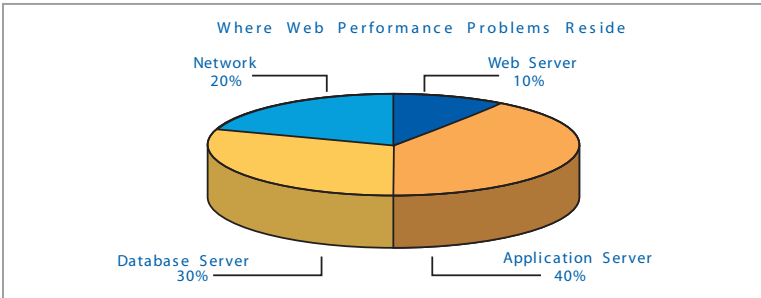


그림 2 : 일반적으로 문제가 존재하는 부분

아래의 내용에서부터 단시간에 효과적으로 퍼포먼스 Bottleneck을 발견, 분리 그리고 해결하는 방법론을 설명합니다.

STEP 1-테스트 접근 : 스크립트의 준비와 방법

전형적인 접근

대부분의 웹 애플리케이션에 대해, 애플리케이션 담당자는 애플리케이션이 지원하는 성능에 대한 기대치를 가지고 있습니다. 이 기대치는 일반적으로 유저의 수로 표현됩니다. 예를 들면 시스템이 피크시에 3,000 명의 동시 접속 접속자가 일반적인 트랜잭션 처리를 할 수 없으면 안 된다고 하는 것입니다. Empirix의 부하 테스트 툴인 e-Load는 가상의 유저를 사용하여 실제 유저의 행동처럼 시뮬레이션하여 성능 요구 수준의 목표를 달성하는지를 검증합니다.

일련의 애플리케이션의 순서를 기록한 스크립트가 목표한 가상 유저수에 이르는 것이 중요합니다. 각 스크립트는 웹 사이트 상에서 발생할 수 있는 실제 유저의 행동을 에뮬레이션 합니다. 각각의 가상 유저는 다른 데이터를 사용하여, 동시에 다양한 유저가 액세스 하고 있는 상태를 만들어 냅니다. 그 결과, 웹 애플리케이션은 실제 유저에 의해 부하가 걸렸을 때와 같은 행동을 나타냅니다. 테스트를 시작한 후, 특히 성능 테스트가 거의 없었던 시스템에서 퍼포먼스 Bottleneck의 징후는 현저하게 나타나는 일이 있습니다. 그 징후는 데이터베이스의 높은 사용율 혹은 특정 트랜잭션에 대한 페이지 응답 시간이 저하되거나 하는 것입니다. 그리고 다음의 단계는 징후를 나타내는 원인이 되는 Bottleneck을 분리 하는 것이지만 이것은 간단하지는 않습니다. 왜냐하면 대부분의 웹 애플리케이션은 동시에 많은 트랜잭션의 발생을 수반하고 있기 때문입니다. 동시에 많은 스크립트를 실행하여, 수백 수천의 트랜잭션을 발생시킨 상태에서 문제가 발생한 트랜잭션에 초점을 맞추는 것은 대단한 일입니다. Empirix는 다음의 과정들을 통해 성능 문제를 간결하게 정의, 분리하는 것으로 이 문제를 해결했습니다.

Empirix의 접근

IBE의 접근방법은 퍼포먼스 Bottleneck을 식별하는 전형적인 방법과는 달리, 트랜잭션 혹은 몇 개의 트랜잭션 처리를 타겟으로 하는 짧은 스크립트, 지시받은 스크립트 및 특정의 스크립트를 이용합니다. 이 방법은 스크립트가 짧은 만큼 특정 문제를 분리하여 식별하는데 용이합니다. Bottleneck을 분리, 식별하는 테스트 공정을 완료한 후에 한층 더 큰 규모로 테스트를 실시하여 애플리케이션의 성능이 기준을 만족 시키고 있는지를 검증합니다.

성능의 수준은, 동시 유저, 초당 트랜잭션, 일일 매출 혹은 다른 기준등으로 정의될지도 모릅니다.

계획

우선 작업을 시작하기 전에 부하 테스트의 계획을 세우는 것이 중요합니다. 다음에 설명되고 있는 몇가지 단계는 테스트 계획시 검토되는 것입니다.

- ▶ **계획** - 애플리케이션 테스트에 대해서 해야 할 계획 및 준비
- ▶ **목표** - 부하 테스트의 목적을 조직내에서 설정 또는 적절한 외부 업체로부터 입수
- ▶ **범위** - 부하 테스트에 의해서 테스트되는 범위와 그렇지 않는 범위의 명확화
- ▶ **성능 요구** - 테스트가 합격/불합격을 결정하기 위해 사용 될 시스템 성능 요구 조건을 문서화
- ▶ **테스트 케이스** - 테스트케이스와 스크립트의 문서화
- ▶ **자원** - 테스트에 필요한 시스템 자원 및 인적 자원의 문서화
- ▶ **테스트 데이터** - 테스트에 필요한 데이터(데이터베이스에 입력되는 master data 및 유저 인터페이스로부터 입력되는 데이터)의 수집 및 문서화
- ▶ **필요한 툴** - 테스트 중에 이용되는 툴의 문서화 (이용 방법 포함)
- ▶ **문제 발견시의 대응** - 테스트중에 중대한 에러나 성능상의 문제가 발견되었을 경우, 문제에 대해 어떻게 대응하는지, 또 문제를 해결하기 위해서는 어느 조직의 어떠한 자원을 준비해야 할 것인가에 대한 적절한 계획 수립

테스트 팀의 결성

테스트 팀을 정의하는 것은 테스트 계획을 세울때 간과하기 쉬운 부분입니다. 성능 테스트는 전형적인 기능 및 회귀 테스트보다 종합적인 기술 지식이 요구됩니다. 그래서 다음은 각 테스트를 담당자에 대한 설명입니다.

- ▶ **상급 테스트 담당자** - 상급 테스트 담당자는 성능 테스트를 조정, 실행하여 최종적으로 그 결과에 대해서 책임을 집니다. 그 때문에 성능 테스트에 대한 높은 이해와 테스트의 실시에 필요한 툴에 대한 이용 지식을 갖추고 있는 것이 바람직합니다.
- ▶ **데이터베이스 관리자** - 데이터베이스 관리자는 테스트중에 있어 데이터베이스의 성능 문제를 발견시, 설정의 변경등 회피하는 방법을 찾을 수 있습니다.
- ▶ **네트워크 관리자** - 네트워크 관리자는 네트워크, 방화벽, 허브, 스위치, 라우터, 로드 밸런스, SSL 가속기등과 설정에 의한 문제의 해결등을 담당 합니다.
- ▶ **개발 프로젝트 관계자(리더)** - 웹 시스템을 실제로 개발한 관계자는 시스템의 기능에 대한 모든 문의에 대한 응답과 프로그램의 원시 코드에 관련하는 문제가 생겼을 경우, 해결 하는 것을 담당합니다.

스크립트의 기록

스크립트의 작성을 쉽게 생각해서는 안됩니다. 각 스크립트는 개발자의 협력을 얻어 충분한 고찰하여 시스템의 특정 기능에 맞도록 작성하지 않으면 안됩니다. 다음의 가이드 라인을 참고합니다.

- ▶ 액세스가 가장 집중되는 페이지, 또는 집중이 예상되는 페이지의 테스트
- ▶ 사이트에서 액세스가 적더라도, 빠른 성능을 유지하는 것이 바람직한 페이지
- ▶ 가장 일반적인 트랜잭션의 테스트
- ▶ 가장 중요한 기능의 테스트
- ▶ 개발자가 Bottleneck이 발생할 가능성이 있다고 하는 페이지의 테스트

테스트팀을 정의하는 것은
테스트 계획을 세울때
간과하기 쉬운 부분입니다

대부분의 스크립트는 아래 3개 카테고리의 하나에 해당합니다.

- End-to-end - 광범위의 기능을 처리하는 긴 트랜잭션
- 모듈 - 특정 트랜잭션의 간결한 스크립트
- 단일 페이지 또는 몇 안 되는 페이지 - 특정 트랜잭션의 매우 간결한 스크립트. 이 경우의 트랜잭션은 웹사이트에서 사용자의 행동을 말합니다.

모듈 테스트, 단일 페이지 또는 몇 안 되는 페이지의 테스트는 Bottleneck 식별에 용이합니다. 이것은 원하는 테스트를 실시하기 위해서 필요한 스크립트입니다. e-Tester를 사용하여 문제페이지 또는 기능에 초점을 둔 간결한 테스트를 작성합니다.(적절한 곳에서 데이터뱅크를 사용) 그리고 시스템의 최종적인 성능과 기능을 검증하기 위해서는 Bottleneck 식별 과정 후 end-to-end 테스트가 행해져야 합니다.

환경과 부하 테스트 틀의 셋업

다음 순서는 테스트에 사용되는 하드웨어 및 소프트웨어의 셋업입니다. 만약 테스트가 원격지로부터의 테스트 서비스라면, 필요에 따라서 서버 컴퍼넌트에 시스템 관리 설정이나 애플리케이션의 액세스 경로 확보도 이 셋업에 포함됩니다. 그리고 모든 서버가 가동할 수 있는 상태에 있는지, 데이터베이스가 운영을 위해 적절히 분류가 되어 있는 데이터량이 실려있는지 등, 테스트 대상 시스템의 테스트 준비가 되어 있는 것을 확인합니다.

실제 환경에서의 테스트가 현실적으로 가능하지 않고 임의의 가상의 환경에서 테스트를 실시했을 경우, 그 테스트 결과에 따라서 증설이 요구되어질수 있는 하드웨어의 확장성에 대해서 최대한의 주의를 기울이지 않으면 안됩니다. 예를 들면 하나의 서버에서 행한 테스트 결과를 통해, 시스템을 2배로 증설한다고 해도 2배의 성능을 얻을 수 있다고 하는 가정은 성립되지 않습니다. 일반적으로, 서버가 2개일 경우 단일 서버일 경우에 비해 보다 좋은 성능을 나타내지만, 실제로 어느 정도의 성능을 나타내는지 실제환경에서 테스트를 할 필요가 있습니다.

서버의 수가 성능에 어떻게 작용하는지 측정하는 비교 벤치마크를 확인하려면, 각각의 아키텍처 환경에 동일한 소프트웨어를 설치하여, 동일한 테스트를 행하는 것이 중요합니다. 그래야 테스트 후에 2개의 시스템간의 합리적인 비교 요소를 얻을 수 있습니다. 또 하드웨어 환경이 100% 일정한 상태인것을 가정 할 수 있다면 동일한 소프트웨어가 도입되고 있는 시스템간의 비교 요소는 알기 쉬워집니다. 그러나 하드웨어나 소프트웨어의 구성에 대해 변경이 행해지면 비교 요소에 작용할 가능성이 높아집니다. 이것은 일반적인 이야기이며, 항상 성립되는 것은 아닙니다.

STEP 2-시스템의 아키텍처를 안다

퍼포먼스 Bottleneck 식별의 두번째 단계는 테스트를 실행하는 시스템의 아키텍처를 아는 것입니다. 그것은 물리적인 컴퍼넌트(서버, 라우터, 방화벽등) 및 시스템을 구성하는 논리적인 컴퍼넌트(예를 들면 관련하고 있는 소프트웨어나 그것이 설치되어 있는 장소)를 이해하는 것입니다. 이것은 어느 카운터를 봐야 하는것인지 그리고 어디에서 문제를 찾아야 하는지 알기 위해 중요합니다.

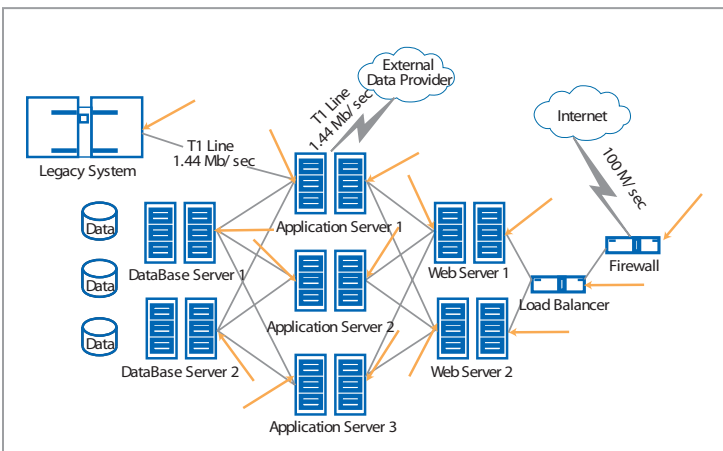


그림 3 : 웹 시스템 아키텍처

모듈 테스트, 단일 페이지 또는 몇 안 되는 페이지의 테스트는 Bottleneck 식별에 용이합니다.

테스트 대상 시스템의 이해에 있어서는 셋업 하고자 하는 모든 시스템이나 모니터를 기록해 둡니다. 여기에는 웹 서버, 데이터베이스 서버, 애플리케이션 서버 등 시스템의 중요 컴퍼넌트를 포함해야 합니다. 가능한 경우, 네트워크 장비가 제공하는 모니터 툴을 이용해 방화벽이나 로드 밸런스의 성능을 감시하면 좋을 것입니다. 그림 3은 웹 시스템 구성의 복잡함을 나타내는 일례입니다. 화살표는 Bottleneck이 발생할 가능성이 있는 장소를 나타내고 있습니다.

STEP 3-서버 관리

서버 관리의 단계에서는 시스템의 어떤 컴퍼넌트를 모니터링 해야할 것인가를 결정합니다. 이것은 테스트 툴에 의해 모아진 성능 카운터를 보충하는 데이터를 제공합니다. 시스템에 포함되는 각각의 컴퍼넌트는 2단계에서 수집된 데이터에 기재되어 있습니다. 이 정보로부터 각 시스템의 어느 카운터를 참조해야할 것인가의 리스트를 작성합니다. 아래의 내용들은 IBE가 추천하는 카운터의 최소 항목입니다. IBE의 ServerStats는 카운터를 간단하게 설정, 모니터링 하는 것을 가능하게 하는 툴입니다. ServerStats는 몇 개의 일반적인 서버에 대해서, 추천하는 카운터를 제시합니다. 이 리스트에는 프론트엔드 모니터링을 통하여 클라이언트가 받는 에러를 포함하고 있지 않습니다. e-Load가 프론트엔드에서 받는 에러를 핸들링 합니다.

Firewalls

- ❖ 토탈 접속 수 - 동시에 접속이 확립되어 있는 수치를 나타냅니다. 이 수치가 방화벽에 제한 되고 있는 최대 접속 수를 넘지 않는 것을 확인합니다.
- ❖ 토탈 SSL 접속 수 - 이 수치가 웹 서버 한 대당 100초를 넘는 경우, SSL Bottleneck이 발생할 가능성이 있습니다. 그 경우 SSL 가속기의 도입을 검토해야 합니다.
- ❖ 시스템 사용율(CPU) - 방화벽의 처리 능력이 부하에 의해 한계에 이르고 있는지 확인하는 것을 가능하게 합니다.
- ❖ Throughput - Throughput이 방화벽의 한계를 넘지 않는 것을 확인합니다.

Load Balancers

- ❖ 토탈 접속 수 - 동시에 접속이 확립되어 있는 수치를 나타냅니다. 이 수치가 로드 밸런스의 라이선스를 넘지 않은 것을 확인합니다.
- ❖ load sharing 분할 - 로드 밸런스가 이용 가능한 서버에 올바르게 부하를 분배 하는지 확인합니다.
- ❖ 시스템 사용율(CPU) - 로드 밸런스의 처리 능력이 부하에 의해 한계에 이르고 있는지 확인하는 것을 가능하게 합니다.
- ❖ Throughput - Throughput이 로드 밸런스의 한계를 넘지 않은 것을 확인합니다.

Web Servers

- ❖ 시스템 사용율(CPU) - CPU의 한계를 넘지 않은 것을 확인합니다. 70% 이상의 사용율은 잠재적인 문제가 있는 것을 나타냅니다.
- ❖ 메모리 사용율 - 웹 시스템이 사용 가능한 메모리가 충분히 여유가 있는지를 확인합니다. 애플리케이션이 SSL 통신을 요구하는 경우, 특별히 주의를 기울입니다.
- ❖ Throughput - 네트워크 대역이 충분히 이용되고 통신이 안정되어 있는 것을 확인합니다.
- ❖ 동시 접속 수 - 접속 및 유저 세션이 모든 웹 서버에 대해서 균형있게 분산되어 있는 것을 확인합니다.
- ❖ 디스크 I/O - 웹 상의 이미지 파일이나 정적 콘텐츠를 테스트하는 것으로 디스크에 빠른 속도로 접근이 가능한지를 확인합니다.

Application Servers

- ❖ 메모리 사용율 - 메모리 누실과 JVM의 메모리 사용량을 확인합니다.
- ❖ CPU - 높은 CPU 사용율은 뛰어난 잘 설계된 시스템이 바쁘게 동작 하는지 혹은 불완전하게 설계된 시스템이 동작하려고 할 때의 지표이기도 하지만, 다른 문제에 의해 Bottleneck이 발생하는 것을 나타낼 수 있는 가능성이 있습니다.

서버 관리의 단계에서는 테스트 툴에 의해 모아진 성능 카운터를 보충하는 데이터를 제공합니다.

- Connection 풀 - 데이터베이스의 connection 풀이 올바르게 이용되고 있는 것을 확인합니다.
- 큐 - 큐를 확인합니다. 리퀘스트가 큐를 시작하는 경우, Bottleneck이 발생 할 가능성이 있습니다.
- 큐 대기 시간 - 큐의 대기 시간이 과도하지 않으면 문제는 없습니다.
- 디스크 I/O - 디스크 액세스가 고속으로 되는지 확인합니다.

Database Servers

- 메모리 사용률 - 데이터베이스가 충분한 캐시 기억 장치를 확보하고 있는지 확인합니다. 충분한 캐시 기억 공간은 보다 좋은 성능을 얻을 수 있습니다.
- CPU - 높은 CPU 사용률은 잘 설계된 데이터베이스가 바쁘게 동작던지 혹은 불완전하게 설계된 시스템이 동작하려고 하고 있는 것에도 불구하고 다른 문제에 의한 Bottleneck이 발생하는 것을 나타내고 있을 가능성이 있습니다.
- 테이블 스캔 - 모든 중요한 비즈니스 트랜잭션에 대해 낮은 테이블 스캔 수를 확인 합니다.
- 테이블 락 - 프로세스가 중요한 테이블을 잠그지 않은 것을 확인합니다. 이것은 다른 곳의 액세스 저하를 초래할 우려가 있습니다.
- 구문 분석률 - 구문 분석률이 높은 경우, 그 처리를 짧은 프로시저로 전환하고 큐 계획의 캐쉬를 갱신합니다.
- 캐쉬 히트율 - 데이터를 저장하기 위한 충분한 캐시 기억 장치가 확보 되는지 확인 합니다.

External Systems

- 접속 수 - 외부 시스템에 대한 리퀘스트를 충분히 처리 가능한 접속 수가 확보되는지 확인 합니다.
- 대역폭의 소비 - 외부 시스템에 대한 대역폭이 충분히 확보되고 있는지 확인합니다.
- 큐의 대기 시간 - 큐의 대기 시간이 과도하지 않으면 문제는 없습니다.

STEP 4 - 테스트의 실행 및 Bottleneck의 발견

이 단계에서는 지금까지의 단계에서 작성해 온 테스트를 실행합니다. 여기에서는 필요한 스크립트를 작성하여 e-Load와 ServerStats가 셋업 되어 테스트 환경이 준비되어 있는 것을 전제로 합니다. 다음은 이러한 것을 빠르게 사용하는 단계입니다. 여기에서 문제를 발견하기 위해 하나 하나의 중요한 트랜잭션을 실행하는 것이 포인트임을 기억해 두어야 합니다. 하나의 트랜잭션을 실행하여 문제가 발견되었을 경우는 그것을 수정, 그리고 재 테스트를 반복합니다.

성능이 요구 사양에 도달되면 다음의 트랜잭션 테스트로 옮깁니다. 모든 트랜잭션의 성능이 요구 사양을 채우는 것을 확인한 후, 모든 트랜잭션을 포함한 시나리오 테스트를 실행합니다. 이것은 개개의 트랜잭션의 문제를 조기에 발견, 해결하여 다음번에는 시스템 전체에 대한 테스트를 실행하는것으로 문제의 소재를 알기 쉽게하기 위한 아이디어입니다.

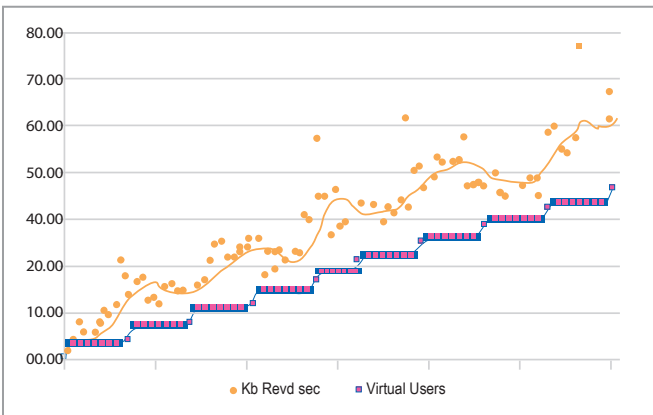


그림 4 : 주기적으로 유저수를 증가한 예

문제를 발견하기 위해 하나 하나의 중요한 트랜잭션을 실행하는 것이 포인트임을 기억해 두어야 합니다.

- 1) e-Load를 시작하고 보다 많은 가상 유저 증가율의 설정은 가상 유저가 추가되는 시점에서 시스템이 안정되는 비율로 설정하는것이 바람직합니다. 페이지간의 대기 시간은 스크립트중 어디에서 지연이 발생하는 것인지를 알기 쉽게 하기 위해, 고정값을 설정하는것이 좋습니다.
- 2) 응답 시간이나 Page/Hit rate가 떨어질 때까지 일정한 증가율을 유지합니다.
- 3) 모니터링하고 있는 서버의 CPU나 메모리가 과도하게 사용되고 있는지 확인합니다.
 - a) 만약 머신에 문제가 있는 경우, 어디서 문제가 나타나고 있는지를 확인해 코드의 어느 부분이 문제처리를 실행하고 있는지 결정합니다. DB나 개발자에게 수정을 의뢰하여 재차 테스트를 실행합니다.
 - b) 만약 자원에 문제가 없는 경우, 네트워크 대역, 외부 시스템 또는 그 외 특정 자원에 대한 요청의 큐 등에 문제가 있을 수 있으며, 문제를 확인, 수정, 재테스트를 실행합니다.

The Checklist

다음은 IBE의 컨설턴트가 경험한 많은 테스트로부터 얻을 수 있던 성능 문제에 대한 리스트를 나타냅니다. 이 리스트의 표기 순서에는 특별한 의미는 없습니다.

Application Server/Application

Problem #1

애플리케이션 서버에 높은 CPU 사용율을 볼 수 있다

Indicator : ServerStats 또는 다른 톨로부터 관찰되는 애플리케이션 서버의 CPU 사용율

Solution : 이 문제에 대한 보편적인 해결책은 없습니다. 이 경우의 원인이 되는 몇가지를 열거합니다.

- ❖ 애플리케이션의 트랜잭션에 관련하는 코드가 최적화되어 있는지 확인합니다.
- ❖ 애플리케이션 서버가 제품 사양에 따라서 최적화된 튜닝이 행해지고 있는지 확인 합니다.
- ❖ 상기와 같은 작업을 이미 행한 경우, 필요한 하드웨어를 증설하는 것에 대한 검토가 필요합니다.

Problem #2

애플리케이션 서버의 낮은 메모리 상태

Indicator : ServerStats 또는 다른 톨로부터 관찰되는 애플리케이션 서버의 이용 가능 메모리

Solution : 애플리케이션에 할당 할 수 있는 충분한 메모리가 설정되어 있는 지 확인합니다. 필요시 메모리를 증설합니다. 또 부하 테스트 실행 중에 경과 시간에 따라 사용 메모리가 일방적으로 증가하는 경우, 애플리케이션에 메모리 누수가 발생하고 있는지 확인합니다.

Problem #3

동적 페이지의 낮은 응답

Indicator : e-Load 내의 페이지/초 또는 애플리케이션 서버의 카운터 관찰

Solution : 애플리케이션 서버의 설정이 제조사의 설명서에 따라 조정되었는지 확인합니다. 또 애플리케이션의 원시 코드에 문제가 없는지 확인합니다. 많은 문제가 애플리케이션에 포함된 비효율적인 원시 코드에 의해서 일어나고 있습니다.

Problem #4

System resource의 낮은 사용율에도 불구하고 응답 시간의 악화(스레드 문제)

Indicator : 부하의 증가에 대해서 throughput이 증가하지 않는다. 그러나 시스템에는 CPU나 메모리의 높은 사용율과 같은 일반적인 스트레스 조건을 나타내지 않는다.

Solution : 큐로 인한 비동기 트랜잭션을 일으키는 조건이 없는지 확인합니다. 이것은 외부 시스템 커넥터를 통해 레거시 시스템에 접속할때 발생하는 경우가 있습니다.

Problem #5

애플리케이션 서버의 높은 CPU 사용율과 동적 페이지의 낮은 응답

Indicator : ServerStats 또는 다른 모니터링 툴로부터 관찰되는 애플리케이션 서버의 CPU 사용율. e-Load의 페이지 비율 카운터

Solution : 비효율적인 프로그램 코드에 의한 전형적인 결과 : 부족한 개발 경험, 미숙한 코딩, 비효율적인 SQL에 의한 대량 데이터의 검색 등 기능을 실행하기 위해서 불러 가는 코드, 본래 정적이어야 하는데 동적인 페이지가 없는지, 또 시스템 설정이 제조사의 추천에 따라서 최적화되고 있는지를 확인합니다. 또, Java 플랫폼을 사용하고 있는 경우는 성능에 관계하는 패치가 제조사의 추천대로 설정이 이루어지고 있는지를 확인 합니다.

Problem #6

애플리케이션 서버의 낮은 처리량, 늦은 응답 시간, 높은 부하 상태에서 가상 유저의 타임아웃이 발생.

Indicator : e-Load가 가리키는 응답 시간, ServerStats에서 관찰되는 애플리케이션 서버의 CPU 사용율

Solution : 애플리케이션 서버의 설정을 확인합니다. 특히, 리퀘스트를 처리 가능한 충분한 Threads가 준비되어 있는지 확인합니다. 많은 애플리케이션 서버는 동시 프로세스의 수를 한 번에 실행되도록 하는 세팅을 가지고 있습니다.

Problem #7

가상 유저의 늦은 응답 시간으로 데이터베이스의 낮은 사용율

Indicator : ServerStats 로부터 관찰되는 데이터베이스 서버의 CPU 사용율. e-Load가 가리키는 응답 시간.

Solution : 애플리케이션 서버의 connection pooling : 애플리케이션 서버의 풀링치가 최적의 값으로 설정되어 있는지 확인합니다.

Problem #8

튀는 응답 시간 : 서버는 안정되어 동작하고 있지만, 크게 튀는 응답 시간을 가끔 볼 수 있다.

Indicator : 일반적으로는 애플리케이션 서버의 CPU 사용율과 가상 유저의 응답 시간에 의해 관찰된다.

Solution : Java 기반의 애플리케이션 서버에서는 한 대의 머신으로 애플리케이션 서버를 클러스터로 구성합니다. 이것의 장점은 애플리케이션 서버가 JAVA Garbage 콜렉션을 일으켜도, 동적으로 생성되는 페이지를 참조할 수 있는 기능을 유지할 수 있는 것입니다. (애플리케이션 서버가 Garbage 콜렉션을 실행하는 타이밍을 늦추는 것에 있습니다) 다른 선택사항으로서 Java의 Garbage 콜렉션에 관한 런타임 파라미터를 "Frequent"로 설정, 한 번의 Garbage 콜렉션에 필요한 시간을 단축하는 것입니다. Garbage 콜렉션의 간격을 보기 위한 런타임 개시에 사용되는 스위치가 있습니다. Garbage 콜렉션이 필요할때 필요이상으로 빈번히 실행되지 않는지 확인합니다.

Problem #9

애플리케이션 서버의 높은 디스크 I/O 또는 높은 CPU 사용율

Indicator : ServerStats로부터 관찰되는 애플리케이션 서버의 디스크 I/O 또는 CPU 사용율

Solution : 애플리케이션 서버의 로그 출력 레벨이 최적화되고 있는지를 확인합니다. 운용시 필요한 최저의 로그 출력 레벨을 결정합니다. 높은 로그 출력 레벨은 시스템에 부담을 주어 결과적으로 유저의 요구 처리에 많은 시간을 필요로 합니다.

Problem #10

낮은 부하 상태에서 특정 페이지의 늦은 응답 시간

Indicator : ServerStats로부터 관찰되는 데이터베이스 서버 또는 애플리케이션 서버의 높은자원 사용율. 데이터베이스 서버로부터 애플리케이션 서버로 대량의 데이터 전송과 애플리케이션의 긴 큐 대기 시간.

Solution : 모든 늦은 동적 페이지를 최적화합니다. .asp, .jsp, JavaBeans 내의 함수, 알고리즘, 그 외의 프로그램을 어떠한 개발 환경에서도 최대한의 성능을 발휘하고 있는지를 확인합니다. 일반적인 문제로 필요한 데이터가 작음에도 불구하고, 대량의 데이터를 반환하고 있는 것을 들 수 있습니다. 필요한 데이터만을 반환하여 데이터베이스의 쿼리를 제한합니다. 예를 들면 데이터 베이스로부터 500건의 레코드를 돌려주는 대신에 20건만 반환하여, 애플리케이션 서버에 20건 단위로 처리를 시키는 알고리즘등입니다.

Problem #11

낮은 부하 상태에서 특정 페이지의 비정상적으로 늦은 응답 시간

Indicator : 특정 페이지의 처리중에 애플리케이션 서버에 관찰되는 스파이크처럼 튀는 CPU 사용율

Solution : 한계 오브젝트는 페이지 기능에 필요한 애플리케이션 코드를 포함하고 있습니다. 그것은 일반적으로 개발자가 .jsp 또는 .asp의 페이지에 필요하지 않은 오브젝트를 포함합니다. 각 페이지 생성 코드의 오브젝트 링크를 최소화하는 것으로, 애플리케이션 서버는 불필요한 페이지의 구문 분석 처리를 실행하지 않게 되어 부하가 작아집니다.

Problem #12

최초 테스트시 ASP서버의 성능 부족.

Indicator : 예를 들면 측정되는 페이지/초등 애플리케이션 서버의 성능이 제조사가 공개하고 있는 성능 명세에 도달하지 않는다.

Solution : ASP를 이용하고 있는 경우 Microsoft가 제시하고 있는 모든 퍼포먼스 튜닝 (MSDN Article: Q253146)를 적용합니다. 이것은 테스트 전의 모든 Microsoft ASP 이용자는 기본적으로 실행해야 할 순서입니다.

Problem #13

최초 테스트시 Java 서버의 성능 부족

Indicator : 예를 들면 측정되는 페이지/초 등 애플리케이션 서버의 성능이 제조사가 공개하고 있는 성능 명세에 도달하지 않는다.

Solution : Java 플랫폼을 이용하고 있는 경우 우선, 가장 빠른 JVM 를 사용합니다. 그리고 사이트에 불필요한 함수를 삭제하여 JVM를 최적화합니다. 또, 소비 메모리 제한, garbage 콜렉션등, JVM실행시 파라미터를 플랫폼 제조사가 추천하는 파라미터로 설정합니다. 이것은 Java 플랫폼 이용자는 기본적으로 실행해야 할 순서입니다.

Web Servers**Problem #14**

테스트 중 Web 서버의 높은 CPU 사용율

Indicator : ServerStats로부터 관찰되는 Web 서버의 CPU 사용율

Solution : 서버에 과부하가 걸려있지 않은지를 확인합니다. 이것은 많은 가상 유저가 SSL를 이용하여 트랜잭션을 실행했을 경우에 보이는 현상입니다. 해결책으로서 Web 서버를 추가하던지, SSL 통신의 처리에 SSL 가속기를 사용합니다.

Problem #15

Web 서버의 높은 디스크 I/O 발생이나 서버 스펙에서 기대하는 값보다 낮은 페이지 throughput.

Indicator : ServerStats로부터 관찰되는 높은 디스크 I/O 또는 낮은 페이지 throughput

Solution : 그래픽, 멀티미디어, HTML등의 모든 Web resource file이 로그 파일을 출력하고 있는 디스크와 다른 디스크에 있는 것을 확인합니다. 이것에 의해, Web 애플리케이션 서버상의 읽고 쓰기를 동시에 행하는 것이 가능하게 됩니다. 웹서버는 과도한 로깅의 양을 수행 할 수 있습니다. 그리고 이 I/O 는 화일을 읽는데 분열을 최소로 발생시킵니다.

Problem #16

NFS를 이용한 Web 파일이나 로그 파일, 그 외 원시 파일의 공유하고 있는 환경의 액세스 속도 저하

Indicator : 파일 공유에 의한 높은 네트워크 사용율

Solution : 파일을 로컬 드라이브에 보존합니다. 또는 고속의 SAN 디바이스를 이용합니다.

Problem #17

모뎀 접속시 페이지 다운로드의 저하

Indicator : e-Load 에서 모뎀 접속 에뮬레이션을 이용하여, 다운로드 시간에 대한 요구 사양을 넘는다.

Solution : 전체의 페이지 사이즈가 75k를 넘지 않은 것을 확인합니다. 페이지 사이즈가 적절하고 모든 네트워크 속도가 유저에게 적절한지 확인합니다.

Problem #18

Netscape iPlanet Web 서버 이용시 기대 밖의 퍼포먼스

Indicator : ServerStats로부터 관찰되는 페이지/초 또는 히트/초

Solution : Netscape(iPlanet Server)가 하나의 프로세스로 실행되고 있는 경우, 프로세스를 분할합니다. Netscape사는 동시 접속이 512 상정되는 경우, 4 개의 프로세스를 실행하는 것을 추천하고 있습니다. 단일 프로세스로 효율적으로 Netscape Server를 가동하는 것은 다소 어려울 수 있습니다.

Problem #19

Web 서버의 높은 디스크 I/O 또는 높은 CPU 사용율

Indicator : ServerStats로부터 관찰되는 Web 서버의 디스크 I/O 또는 CPU 사용율

Solution : Web 서버의 로그 레벨이 낮게 설정되어 있는지 확인합니다. 그리고 현재 로그 레벨 설정이 필요하지 검토합니다. 높은 로그 레벨의 설정은 시스템이 보다 많은 시간에 사용자 리퀘스트의 처리를 위해서 분석, 구축되어 없어지게 될 것입니다.

NETWORK

Problem #20

Web 시스템은 낮은 CPU 사용율을 나타내지만, 응답 시간이 저하한다

Indicator : 유저의 증가에 따라 응답 시간이 늦어지지만, Web 시스템에 부하가 발생하고 있는 징조를 볼 수 없다.

Solution : 외부 시스템과의 상호작용이나 네트워크 대역이 한계를 넘지 않았는지 확인합니다.

Problem #21

네트워크 상에서 과도한 패킷손실

Indicator : SNMP 카운터로부터 감시되는 관련 스위치나 라우터의 패킷 손실율.

Solution : 이 문제는 일반적으로 네트워크 장비가 올바르게 설정되어 있지 않거나 혹은 네트워크 용량 한계를 넘는 것에 의해 발생합니다. 필요한 대역폭의 네트워크를 준비하거나 네트워크의 설정이 올바르게 되어 있는지 확인합니다.

Problem #22

connection의 절단 (e-Load에서 12xxx 에러가 발생)

Indicator : e-Load 테스트중에 12xxx 에러가 발생.

Solution : LAN의 한계를 넘고 있을 가능성이 있습니다. (전형적인 예로, 네트워크가 내부 및 외부 네트워크로 분할되어 있지 않다). 이러한 문제는 네트워크의 설정이 제대로 안된 결과입니다. 그리고, 시스템에 포함되는 방화벽, 로드 밸런스, 라우터, 스위치등의 설정을 확인합니다.

Problem #23

적은 사용자 액세스로도 네트워크 대역이 넘는다.

Indicator : ServerStats로부터 관찰되는 스위치의 SNMP 카운터

Solution : 테스트시에는 네트워크상의 불필요한 트래픽을 억제하기 위해, 모든 내부 트래픽에 대해 독립된 네트워크를 사용하는 것을 추천합니다. 서버 컴퍼넌트는 서버간의 커뮤니케이션을 취하기 위해, 동작시에 많은 트래픽을 생성합니다. 이것은 통신을 하나의 네트워크 카드나 서브넷으로 웹 트래픽을 만들어 최소화할 수 있습니다.

Problem #24

가상 유저 증가시에 방화벽 접속 불가능의 에러를 보내기 시작한다. 또는 시스템이 접속타임 아웃의 에러가 발생하기 시작한다.

Indicator : e-Load의 가상 유저가 수신한 에러, 또는 ServerStats로부터 관찰되는 방화벽의 정보.

Solution : 방화벽의 라이센스 또는 설정을 확인합니다. 또 동시 접속 유저수가 방화벽의 제한을 넘지 않았는지 확인합니다. 각 유저는 서버에 대해서 최대 4 개의 접속을 열 가능성이 있는 것을 의식할 필요가 있습니다.

Problem #25

부하 상태하에, 로드 밸런스 환경시 모든 웹 서버에 균등하게 부하가 최적화 안됨

Indicator : ServerStats로부터 관찰되는 특정 Web 서버의 높은 접속수

Solution : 로드 밸런스의 설정이 적절히 되어 있는지 확인합니다. 또 로드 밸런스 룰이 모든 웹 서버에 load sharing 되도록 조정합니다.

Problem #26

네트워크 성능의 부족

Indicator : 스위치의 성능을 확인, e-Load가 나타내는 네트워크 Throughput에 의해서도 확인 가능.

Solution : 모든 장비에 대해 100Mbps 이상의 네트워크 접속이 확립되어 있는지 확인합니다.

External Systems**Problem #27**

내부에 있는 웹 시스템은 만족스런 성능으로 동작하고 있지만, 전체적인 시스템의 성능이 떨어진다.

Indicator : 페이지/초 또는 트랜잭션/초당의 늦은 응답시간. 테스트 기준을 만족 시키지 않는 성능.

Solution : 외부 시스템의 성능과 외부 시스템의 상호작용을 확인합니다. 어떠한 처리를 하고 있는지 정확하게는 모르는 외부 시스템, 일명 "블랙 박스"에서 흔히 볼 수 있습니다. 자주 인터페이스의 개발자는 그것의 성능 확인을 위해서 이용 가능한 카운터를 만들것입니다. 시스템의 이러한 예는 웹사이트에서 세금 계산을 하거나 배송 추적, 그리고 신용카드의 유효성등이 있습니다. 외부 시스템상의 큐, 외부 처리 이벤트의 동기, 외부 시스템과의 접속에 사용되고 있는 회선의 대역등도 조사의 대상이 됩니다.

Database**Problem #28**

애플리케이션 서버상에 체류 하고 있는 대량의 리퀘스트. 그러나 애플리케이션 서버 사용율은 낮다.

Indicator : ServerStats로부터 관찰되는 애플리케이션 서버의 리퀘스트 큐

Solution : 애플리케이션 서버로부터 데이터 베이스 서버에 대한 충분한 connection이 준비되어 있는지 확인합니다. 많은 경우 JDBC 풀을 조정하는 것으로 성능의 개선을 볼 수 있습니다.

Problem #29

쿼리 응답 시간의 저하

Indicator : 특정의 페이지 또는 트랜잭션 응답 시간이 늦다

Solution : 쿼리나 저장된 프로시저가 사용하는 인덱스가 효과적인지 확인합니다. 쿼리 계획을 확인해, 쿼리를 최적화합니다. 데이터베이스 Profiling 툴을 사용해 늦은 쿼리에 주목합니다. 최소화 sorting(기본 인덱스에 의한 sorting)으로 이러한 쿼리를 최적화합니다. 불필요한 트리거, 작업용 테이블을 삭제해 가능한 한 저장된 프로시저를 사용합니다. 인덱스를 적절히 참조, 검색 결과를 최소화하고, 또 테이블의 결합이나 소트, 조항의 그룹을 최소화 되도록 SQL 문을 최적화합니다.

Problem #30

Oracle 또는 다른 데이터베이스의 전체적인 성능의 저하

Indicator : 적은 유저수로 높은 데이터베이스의 사용율.(특히 테스트 개시 직후)

Solution : DBMS에 적절한 업데이트가 적용되어 최적화가 행해지고 있는 것을 확인합니다. 또 DBMS 제조사가 공개하고 있는 튜닝이 적용되고 있는 것을 확인합니다.

Problem #31

데이터베이스 전체 성능이 느리고, 애플리케이션 서버간, 데이터 베이스 서버간에 대량의 데이터 이동이 발생하고 있다.

Indicator : 데이터베이스 서버의 높은 CPU 사용율. 애플리케이션이나 데이터베이스 서버간의 대량의 데이터 이동

Solution : 저장된 프로시저가 사용되고 있거나 또는 실행시마다 recompile 될 필요가 없는 쿼리를 사용하고 있는 것을 확인합니다. 저장된 프로시저에는 몇개의 이점이 있습니다. 우선, 저장된 프로시저는 사전컴파일 됩니다. 인라인 SQL은 매번 실행되어 질때 컴파일 또는 문장 해석을 실시하지 않으면 안됩니다. 또 인라인 SQL 코드는 최적화하거나 소스 관리가 어렵습니다. 정자된 프로시저는 DBA에 의해서 비교적 간단하게 최적화할 수 있습니다.

Problem #32

데이터베이스의 디스크 I/O 가 대량으로 발생한다.

Indicator : ServerStats 로부터 관찰되는 디스크 I/O 카운터

Solution : 데이터베이스의 로그, 데이터, 인덱스를 다른 디스크에 보존합니다. RAID 디스크를 사용하는 것이 가장 효과적입니다. 데이터베이스가 모든 쿼리에 대해 인덱스를 사용하고 있는가를 확인합니다. 또, 데이터베이스에 충분한 메모리와 캐쉬를 할당되어 있는지 확인합니다.

Problem #33

Multiusuer에 의한 데이터베이스 성능의 저하

Indicator : ServerStats로 감시되는 데이터베이스 모니터의 테이블 락 또는 트랜잭션 블록이 잦은 빈도로 발생.

Solution : 실행되고 있는 트랜잭션을 조사해 복수의 동시 유저에게 대응할 수 있도록 최적화합니다. 필요에 따라서 데이터베이스 락의 계획을 변경합니다.

Problem #34

쿼리 응답의 저하

Indicator : ServerStats 로부터 관찰되는 참조 테이블수

Solution : 필요한 인덱스의 발견 및 추가. 인덱스의 재구축 및 새로운 구성.

About IBE

IBE는 웹, 음성애플리케이션, 컨택센터, 커뮤니케이션 네트워크를 측정, 관리하고 성능을 향상시키는 테스트 및 모니터링 솔루션을 제공하는 선도적 회사입니다.

IBE는 개발 비용과 시간을 단축함과 동시에 최종 사용자의 체험의 질을 극대화하는 강력하고 혁신적인 제품과 서비스를 제공합니다. IBE는 전 세계에 3,400여 개의 고객과 2,000여 개의 기업들과 장비생산업체, 서비스업체들에게 제품과 서비스를 제공하고 있습니다.