Софийски университет „Св. Климент Охридски”
Факултет по Математика и Информатика
Катедра „Информационни технологии”
Специализация „Софтуерни технологии”

ДИПЛОМНА РАБОТА

Тема:

Тестване за производителност, анализ и оптимизация на Web приложения

Дипломант: Биляна Милкова Петрова, ФN№ M21753

Научен ръководител: доц. Силвия Илиева,
катедра „Информационни технологии”, ФМИ

Консултант: Илина Манова, Рила Солюшънс

Съдържание

1 Въведение

1.1 Въведение в предметната област

В съвременния свят на информационните технологии, тестването е неразделна част от процеса на създаване на софтуерен продукт или услуга. Това е процесът на изследване на дадена софтуерна система, за да се докаже, че тя отговаря на изискванията и критериите за качество, поставени към нея.
Съществуват различни видове методи на тестване, всеки от които обхваща специфичен аспект в осигуряването на качеството. Най-основните са:

- **Black Box Testing** – Тестване, базирано на изискванията и функционалността на продукта, а не на знания за неговата вътрешна структура или код.
- **White Box Testing** – Тестване, базирано на вътрешната логика на системата (изпълнение на условия, цикли и т.н.).
- **Unit Testing** - Тестване на най-малките единици на системата.
- **Integration Testing** – Тестване на две или повече части на системата, за да се провери как те работят съвместно.
- **Functional Testing** – black box тестване, целящо да провери дали системата отговаря на изискванията за предназначението и.
- **Regression Testing** – Тестване, което се извършва повторно след извършени модификации за да се провери дали не е засегната преди това съществуваща и работеща функционалност.
- **Performance Testing** – Тестване на системата при натоварване с голям брой (виртуални) потребители.
- **User-Acceptance Testing** – определя дали системата покрива очакванията на нейните клиенти/потребители към нея.
- **Usability Testing** – Тестване за използваемост и удобство на системата.
- **System Testing** – включва различни видове тестове за цялостно изследване на системата в условия максимално близки до реалните.

1.2 Цел на дипломната работа

Настоящата дипломна работа има за цел да представи един от методите за тестване - тестването за производителност (Performance Testing) и неговото място в софтуерния процес. Тя покрива основните дефиниции, понятия, тестови подходи и най-добри практики за тестване на производителност. Подробно се разглеждат различните метрики, които се следят по време на изпълнение на тестовете и техните връзки. Дипломната работа включва обзор на някои от най-известните съвременни инструменти за тестване на производителност и техните характеристики. В по-голям детайл се разглежда инструмента Microsoft ACT и се показва как той може да се

използва на практика за тестване и анализ на производителността на примерно Web приложение.

1.3 Полза от реализацията

Дипломната работа съдържа информация и знания за тестването за производителност, които могат да послужат на всички тестови инженери, разработчици, анализатори, ръководители на проекти, които се интересуват от тази тема и биха искали да научат повече. Дипломната работа е разработена, за да отговори на потребността за подробна и систематизирана информация за тестването на производителност. Освен теоритична обосновка, тя включва и практическа част, където читателите ще могат да се запознаят с един конкретен тип приложения и проследят примерна стратегия за изследване на производителността с помощта на инструмента Microsoft ACT.

1.4 Структура на дипломната работа

Дипломната работа постепенно въвежда читателя в процеса на тестване на производителност и е организирана в следните секции:

2) Тестване за производителност

В тази част, читателят се запознава със същността, основните дефиниции и целите на тестването за производителност, както и неговото място в софтуерния процес. След това се разглеждат детайлно най-важните метрики, които се следят по време на изпълнение на тестовете – време за отговор от страна на системата (Response Time, Latency), капацитет (throughput) и използване на системните ресурси (utilization).

3) Тестове, подходи и практики

В тази част, първоначално читателят се запознава с четири подхода, които могат да се следват при тестването за производителност на дадена Web система – Benchmarking, Capacity Planning, Soak и Peak-Rest. След това се разглеждат най-често използваните видове тестове за отчитане на метриките за производителността. В края на тази част се споменават и някои добри практики, които се препоръчват при разрешаване на проблеми, свързани с производителността.

4) Инструменти за тестване за производителност

В тази част, читателят се запознава с характеристиките на най-известните инструменти за тестване на производителност, предлагани в момента на софтуерния пазар. В по-големи детайли се разглеждат особеностите и графичният интерфейс на инструмента Microsoft ACT.

5) Тестване на производителност на приложението XML Firewall. Планиране и стратегия

Тази част от дипломната работа е с практическа насоченост и в нея читателят се запознава с примерното приложение XML Firewall, неговите характеристики и принцип на работа. След това се дефинира и стратегията за тестване на производителност на приложението XML Firewall - как ще протече процеса, какви тестове ще се изпълняват, какъв инструмент ще се използва и как ще се разработят съответните тестове.

6) Изпълнение на тестовете. Отчитане на резултатите. Анализ.

Тази част се базира на стратегията и примерния тестов скрипт, дефинирани в предната секция. Тук се демонстрира настройката, изпълнението и отчитането на резултатите от проведените тестове. На базата на получените резултати се извършва анализ и се определят посоки за подобрение на системата.

В дипломната работа са използвани следните фигури, таблици и съкращения:

1. Индекс на използваните фигури

2. Индекс на използваните таблици

3. Списък на използваните съкращения

Съкращение	Пълно наименование
TTLB	Time To Last Byte
TTFB	Time To First Byte
RPS	Requests Per Second
TCA	Transaction Cost Analysis
HTTP	HyperText Transfer Protocol
SSL	Secure Socket Layer
XML	Extensible Markup Language
SOAP	Simple Object Access Protocol
JMS	Java Message Service
LDAP	Lightweight Directory Access Protocol
XSLT	Extensible Stylesheet Language
DTD	Document Type Definition

2 Тестване за производителност

2.1 Същност на тестването за производителност и неговото място в софтуерния процес

Тестването за производителност представлява процеса на изпитване на поведението на дадена софтуерна система при различно натоварване с голям брой (виртуални) потребители. Освен качественият анализ, за производителността на системата са важни и количествените измервания, които могат да послужат за анализирането на дефекти и да помогнат за правилното им отстраняване.

Основната цел е не само да се установят възможностите и ограниченията на системата по отношение на производителността, но същевременно да се стимулира нейното развитие и усъвършенстване. По време на тестовете се събира количествена информация за различни показатели, които позволяват да се определи какви промени могат да се направят в системата и съответно как те ще помогнат за постигането на поставените цели. Измерванията могат да покажат дали системата се доближава или отдалечава от поставените изисквания по отношение на производителността. Таблица 2-1 съдържа най-основните показатели за производителността на едно Web приложение:

Обект	Показател	Описание
Процесор	%Processor Time%	Използване на процесора
Памет	Available Bytes	Наличната памет на сървъра
Мрежов интерфейс	Bytes/Sec	Мрежовия трафик от клиента към сървъра
Web приложение	Avg. Requests/Sec	Среден брой заявки за секунда.
Web приложение	Avg. Time to First Byte (TTFB)	Средното време, изминало от изпращането на заявката до пристигането на първия байт на отговора
Web приложение	Avg. Time to Last Byte (TTLB)	Средното време, изминало от изпращането на заявката до пристигането на последния байт на отговора

Таблица 2-1: Основни показатели за производителността

Тестването за производителност води до няколко задачи, свързани с оптимизация на различни аспекти от системата:

1) Извършване на анализ, дефиниране на изисквания по отношение на производителността и съответно определяне на допустимо поведение на системата.
2) Изпълнение на тестовете за измерване на производителността.
3) Идентифициране на компонентите, критични за производителността (bottlenecks)
4) Извършване на оптимизационни промени с цел отстраняване на откритите проблеми
5) Повторение на тестовете за производителност след промените и анализ за техния ефект върху поведението на системата

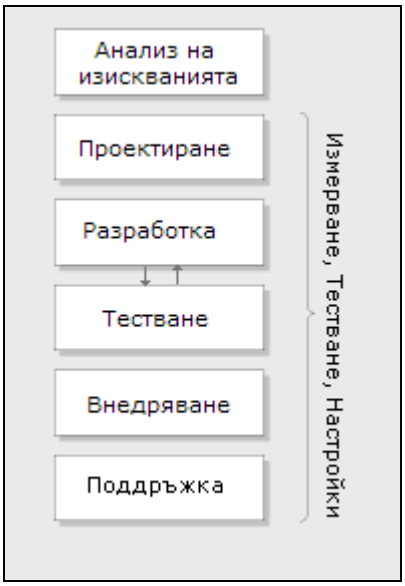
Тестването за производителност само по себе си е повече изкуство, отколкото наука. Няма общо установен подход, който да се прилага при всяко едно приложение/система. По време на процеса на тестване за производителност, е важно да се следи системата за определени грешки, които биха могли да се проявят само когато тя е подложена на натоварване. Тестовете могат да помогнат за изолиране на „коварни" дефекти, които могат да не бъдат забелязани преди пускане на системата в реална експлоатация. Един от най-важните моменти е планирането на тестовете и определяне на кои показатели (метрики) ще бъдат измервани. Добър подход е отначало да се извърши контролен тест и да се запишат базовите стойности на метриките, които да послужат за сравнение при следващите тестове. Така може да се провери дали има подобрение от извършените промени в системата или пък те са имали обратен ефект. Съществено изискване по отношение на дизайна на самите тестове, е те да могат да се възпроизвеждат отново. „Ефективният процес на тестването за производителност трябва да бъде итеративен"*(Adam Kolawa,2001).* Следователно, един тест е валиден само ако може да бъде възпроизведен след това. В противен случай няма да могат да се съпоставят резултатите от тестовете с извършените промени в системата. Основна стъпка в процеса е определянето на нивото на натоварване на системата. Трябва да се вземе под внимание, че то може да е променливо. Повечето приложения не са подложени на постоянно натоварване. Например, приложение, което е свързано със събиране на суми за дължими сметки, може да е подложено на по-голямо натоварване в края на месеца, когато фактурите на сметките са готови. При такива случаи, за да се доближат максимално до реалността, тестовете за производителност трябва да симулират пиковото натоварване на системата. Според някои автори *(Leslie Segal,cmp.3),* доброто познаване на системата и нейните потребители е от първостепенна важност за определяне на какви тестове е необходимо се извършат.

Като естествено разширение на тестването за производителност е процесът на настройка (Performance Tuning), при който може да се направят някои оптимизационни промени в приложението, да се донастройват различни системни параметри, да се добавят нови ресурси с цел подобряване на производителността му. От съществена важност при този процес е да се модифицира само по един параметър в даден момент и това да се извършва в контролирана среда. Например, ако се тества J2EE приложение и трябва да се определи дали увеличаването на паметта на JAVA виртуалната машина ще окаже влияние върху производителността, то трябва

натоварването на системата да е едно и също при всеки тест, а увеличението да се извършва поетапно (например 1024 MB, 1224 MB, 1524 MB, 2024MB). В края на всеки тест се спира и се записват съответните данни и резултати за анализ и сравнение. След евентуална донастройка на системата, тестовете трябва да се повторят отново и да се оценят отново получените резултати. Този процес се повтаря докато се постигнат поставените отначало изисквания за производителността на системата.

Как замерванията се прилагат в жизнения цикъл на проекта?

Процеса на замерване трябва да започне веднага след като има ясно поставени цели за производителността на системата. Препоръчва се това да стане още по време на фазата на проектиране на системата, както е показано на фигура 2-1:



Фигура 2-1: Жизнен цикъл на софтуерна система

Планирането на тестването на производителността именно в тази фаза има следните предимства:

- Производителността на системата става част от нейния дизайн
- Дава се отговор на въпроса „Ще може ли дизайна на системата да покрие поставените изисквания за производителност?”. Така още преди да се напише кода, могат да се анализират недостатъците на дизайна.
- Може да се разбере кои части от дизайна са най-уязвими по отношение на производителността

Замерванията на производителността трябва да се прилагат по време на целия процес на разработка на системата, за да се определи съответно дали тя се доближава или отдалечава от поставените цели.

Планиране на производителността

Процеса на планиране на производителността включва създаването на план за производителността. Състои се от следните стъпки:

1. **Определяне на ключови сценарии**

   Ключовите сценарии могат да бъдат един от следните два вида:

   А. **Критични сценарии** – това са сценарии, които имат специално поставени изисквания за производителността. Това са още така наречените „Service Level Agreements". Например такъв сценарий може да бъде: „Изпълняването на поръчка не трябва да отнема повече от 3 секунди".

   Б. **Важни сценарии** - това са сценарии, които са най-често използвани от потребителите и покриват основни функционалности на системата.

2. **Определяне на натоварването на системата**

   На тази стъпка се дефинират:

   a) Общ брой потребители
   b) Брой конкурентни потребители (взаимодействащи със системата в един и същи момент)
   c) Обем данни, преминаващи през системата

   Определените стойности трябва да се съобразят и с конкретните сценарии. Например могат да поставят следните изисквания: „Трябва да се поддържат 1000 потребители, които да разглеждат системата и 100 потребители, които едновременно правят поръчки."

3. **Определяне на стойности на метриките, които системата трябва да покрие**

   За всеки от определените сценарии на стъпка 1, се определят стойности на метриките. Тези стойности се дефинират съобразно поставените бизнес изисквания. Обикновено се дават стойности за:

   a) **Време за отговор**. Например „Каталога с продукти трябва да се покаже на екрана за не повече от 3 секунди"
   b) **Капацитет.** Например „Системата трябва да обработва 100 транзакции за 1 секунда"
   c) **Консумация на системни ресурси** (процесор, памет, използване на дисковото пространство и други).

Планът се оформя като документ, в който се включват поставените изисквания, определените тестови сценарии и натоварване на системата. Също така в него се включва информация за необходимите ресурси за извършване на тестовете – времеви, човешки, както и за възможните рискове и начините за тяхното минимизиране или избягване.

2.2 Метрики

При тестовете за производителност се следят различни количествени метрики. Всички те са базирани на поведението на системата през определен интервал от време. Най-често се отчитат характеристиките: време за отговор от страна на системата (Response Time, Latency), капацитет (Throughput) и използването на системните ресурси (Utilization).

2.2.1 Време за отговор от страна на системата (Response Time, Latency)

Тази метрика служи за измерване на времето от стартирането на дадена заявка до получаването на съответния отговор/резултат. Дефиницията на тази метрика предполага наличието на начална точка, от която да започне отчитането на времето и крайна точка, в която да спре. Като примери за тази метрика могат да бъдат посочени:

a. Времето, което изминава от изпращането на заявка за дадена страница от Web браузър и нейното пълно визуализиране на екрана.
b. Времето, за което мрежов рутер задържа даден пакет, преди да го препрати в дадено направление.

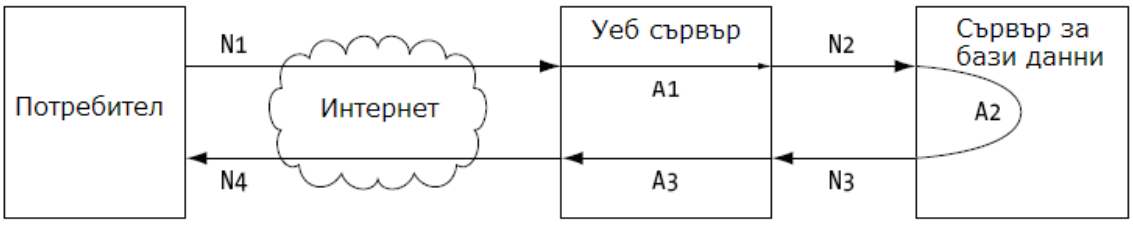
В Web приложенията, тази метрика отчита времето между заявка (request) и съответния и отговор (response). В много инструменти за тестване за производителност, времето за отговор от страна на системата се представя още с метриката „**времето до последния байт**" (Time To Last Byte, TTLB), която измерва времето от изпращането на заявка до момента на пристигане на последния байт на отговора. Друга важна характеристика е „**времето до първия байт**" (Time To First Byte, TTFB), която отчита времето, изминало от началото на заявката до пристигане на първия байт на отговора (*Raj Jain 2001, глава 3.3*). Размера на отговора и мрежовото забавяне са фактори които оказват влияние при тази метрика.

В повечето приложения, времето за отговор от страна на системата отначало расте линейно с увеличаването на натоварването на системата. В даден момент обаче(при достигане на определено натоварване), то нараства експоненциално, както е показано на фигура 2-2:

Фигура 2-2: Изменение на времето за отговор спрямо натоварването

Най-често внезапното нарастване на времето за отговор от системата е знак, че системата е достигнала лимита на някои от своите ресурси. Web сървърите обикновено се сблъскват с този сценарий, когато броят на заявките надхвърли максималния брой конкурентни сесии. Това става, когато всички свободни нишки са вече заети и всички допълнителни заявки вместо да бъдат обработени веднага, се поставят в чакаща опашка. Времето, прекарано в опашката оказва влияние върху времето за отговор от страна на системата.

Факторите, които влияят на времето за отговор от дадена Web система могат най-общо да се класифицират в две категории: към едната се причисляват **мрежови/системни** забавяния (**network latency**), а към другата – **приложни** забавяния (**application latency**). **Мрежовите** забавяния се отнасят до времето, което е необходимо за пренасяне на данните между отделните сървъри. **Приложните** забавяния се отнасят до времето, което е необходимо на даден сървър да обработи съответната информация. Обикновено в едно Web приложение се срещат по няколко инстанции и от двата вида забавяния. Фигура 2-3 илюстрира нагледно различните видове забавяния при обработката на една Web заявка:



Фигура 2-3: Забавяния при обработката на една Web заявка

- N1, N4 – времевите забавяния между клиента и сървъра през Интернет.
- N2, N3 – времената за комуникация между отделните сървъри/приложения/сървъри за бази данни, изграждащи системата.

- A1, A2, A3 – времената, нужни на съответните сървъри/приложения да обработят съответната информация

Общото време за отговор от страна на системата, тогава се изчислява по следната формула:

$$\textit{Време за отговор от страна на системата} = \sum N_n + \sum A_n,$$

където N_n е n-тото мрежово забавяне, а A_n е n-тото приложно забавяне.

Мрежовите забавяния през Интернет са присъщи за всяко едно Web приложение. Потребителите, които използват приложението могат да имат различен достъп – от най-висикоскоростен интернет до dial-up достъп. За минимизирането на мрежовите закъснения, като опция може да се препоръча разполагането на Web сървърите на хостове на големи Интернет доставчици, които имат висикоскоростни връзки с много хостове в Интернет. По този начин може да се намали броят на хоповете, през които преминава заявката, докато се намира в „облака" на Интернет.

Времетраенето на комуникацията между отделните сървъри, използвани от самото Web приложение, също е от значение за неговата производителност. От съществена важност тук е вида на физическата мрежа и нейната конфигурация. За да се постигне оптимална производителност, когато е възможно се препоръчва тези сървъри да са в една и съща подмрежа и да са с конфигурирани с фиксирани IP адреси (по този начин би се избегнала възможността тези сървъри да бъдат недостъпни, съответно когато някои DHCP сървър е в неизправност или не е достъпен). Когато забавяне от такъв тип се окаже реален проблем за производителността на системата като цяло, може да се обмисли възможността за подобряване на мрежовата инфраструктура (например добавяне на рутери и т.н.)

Времето, което отнема на самото приложение да завърши дадена задача също е от съществено значение за неговата производителност. За да се минимизира това време, във фазата на разработка на едно Web приложение се препоръчват следните практики:

- Минимизиране на всички задачи, които биха довели до забавяне (например преминаването на дадена заявка последователно през няколко сървъра)
- Ефективност при писането на кода – използване на техники за кеширане на често използвани данни или такива, които са трудни за изчисление.
- Използване на техники за оптимизация при работа с бази от данни като съхранени процедури (stored procedures) и индекси с цел подобряване на производителността на базата от данни.
- Избягване на транзакции, когато те не са необходими.
- Използване на такива системни ресурси, които покриват нуждите на приложението и предполагаемия брой потребители.

2.2.2 Капацитет (Throughput)

Капацитетът е метрика, която служи за измерване на работата, извършена за единица време. В софтуерните системи, това е скоростта, с която се обслужват заявките на потребителите. Като примери за дефиницията на тази метрика могат да бъдат посочени:

- Брой транзакции, извършени за минута
- Брой обръщения към паметта за секунда
- Количество данни, предадени по мрежата за секунда

В Web системите, капацитетът се дефинира като броя на потребителските заявки, обслужени за секунда (Requests per second, RPS). Капацитетът е една от най-полезните метрики, която се отчита при тестовете за производителност, тъй като играе важна роля при идентифицирането на дефекти.

Фигура 2-4 илюстрира как обикновено се изменя капацитетът спрямо натоварването на системата:

Фигура 2-4: Изменение на капацитета спрямо натоварването

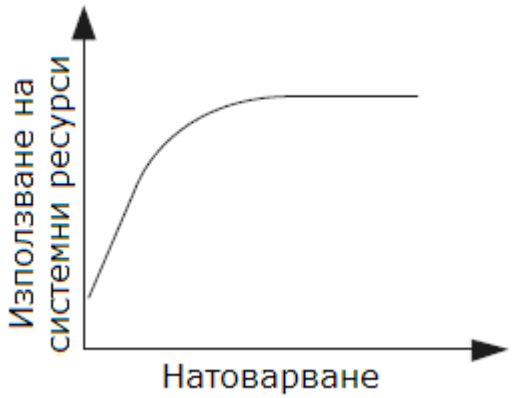
Броят на заявките, обслужени за секунда отначало расте линейно с увеличаването на натоварването на системата. В даден момент обаче, при определено натоварване, се достига максималния капацитет на системата. Допълнителните заявки ще се поставят в чакаща опашка, а това от своя страна ще доведе до намаляване на капацитета. Важно е да се отбележи, че системата не само трябва да може да обслужва адекватно максималния брой потребители, но в същото време да функционира коректно съобразно поставените и изисквания.

Капацитетът е метрика, която силно зависи от сложността на системата. Например, едно динамично Web приложение ще има различен капацитет в сравнение с Web приложение, състоящо се само от статични HTML страници. На практика извършване на сравнение между такива приложения е безсмислено. Когато обаче се извършват многократни тестове на едно и също приложение, се получават резултати, които могат да се използват за анализ на това как извършените промени в системата са се отразили на нейната производителност.

2.2.3 Използване на системните ресурси (Utilization)

Тази метрика, отчита в проценти използваните от приложението системни ресурси като процесор (CPU), памет, твърд диск. Допустимите стойности са в интервала от 0 до 100 процента. Когато някой от системните ресурси се използва на 100 процента, той не може да извършва никакви допълнителни задачи и съответно се превръща в проблем за приложението (bottleneck). В такава ситуация рязко се забавя времето за отговор от страна на системата. Затова обикновено стремежът е да не се заема даден ресурс на повече от 70 или 80 процента.

Използването на системните ресурси отначало нараства пропорционално с увеличението на броя на потребителите, взаимодействащи със системата. В определен момент се достига една максимална граница и тя не се променя с нарастване на натоварването на системата. Това се вижда от фигура 2-5:



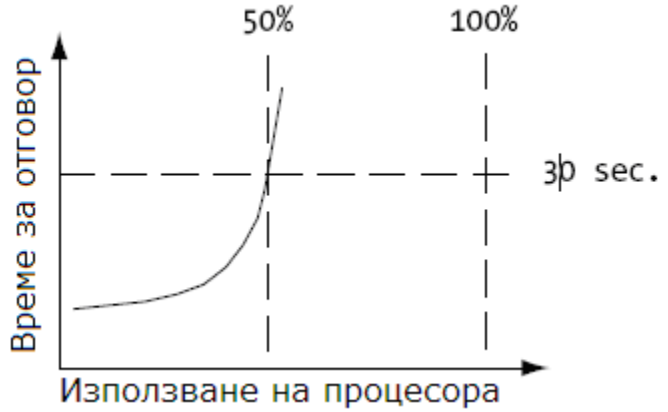
Фигура 2-5: Използвани системни ресурси, в зависимост от натоварването

Когато използването на някои от системните ресурси действително се превърне в проблем за производителността на системата, като вариант може да се обмисли замяната му с по-добър (например по-голям твърд диск, по-мощен процесор и т.н.). Друга възможност за подобряване на производителността е, да се разпредели натоварването с помощта на допълнителни сървъри (това е така наречената „load balancing" техника). Освен добавянето на нов или подобряването на съществуващия хардуер, използван от приложението, трябва да се обърне внимание на някои аспекти от самото приложение. Например, използването на техники за кеширане и оптимизации при работа с данни могат да намалят значително използването на системните ресурси.

2.2.4 Връзки между отделните метрики

За да се анализира цялостно производителността на една Web система, е необходимо не само да се следят и трите метрики по време на тестовете, но и да

се разбират връзките между тях. Например, когато някои от системните ресурси бъде зает на 100%, това неминуемо ще доведе до драстично забавяне на времето за обработване на една заявка от системата. Съществуват ситуации обаче, при които може някои основен системен ресурс (като например процесор) да не се използва на 100 процента, но същевременно времето за отговор от страна на системата да е станало неприемливо (фигура 2-6):



Фигура 2-6: Времето за отговор спрямо използване на процесора

От фигурата се вижда, че времето за отговор от страна на системата е достигнало до 30 секунди, докато процесора се използва все още само на 50 процента. В такива случаи, за да се идентифицира проблема, трябва да се изследва как приложението консумира останалите системни ресурси и отново да се направи съпоставката от графиката.

От друга страна, нарастването на времето за обработване на една потребителска заявка от системата, съответно ще намали и броят на заявки, които тя ще успее да обработи за секунда. Може да се случи така обаче, че времето на отговор на системата да е станало неприемливо много преди да се е намалил броят на обработените заявки за секунда. Всичко това, показва, че по време на тестовете трябва да се следят и трите метрики и техните съотношения една спрямо друга, за да могат да се направят точни и коректни анализи за поведението на системата.

2.3 Обобщение

Тестовете за производителност имат за цел да изследват поведението на една система, подложена на различно натоварване. Те могат да помогнат за намиране и изолиране на дефекти още по време на разработване на дадено приложение, а не след пускането му в реална експлоатация. Преди да бъдат разработени тестовете, трябва да се извърши анализ и да се дефинират изисквания по отношение на производителността. Препоръчва се това да стане още по време на фазата на проектиране на системата. При изпълнението на тестовете се следят стойностите на различни метрики, които имат за цел да покажат дали системата

се доближава или отдалечава от поставените изисквания по отношение на производителността. Най-често използваните метрики при тестването на Web приложения са: време за отговор от страна на системата (Response time), капацитет (Throughput) и използване на системните ресурси (Utilization).

3 Тестове, подходи и практики

Тестването за производителност може да се извърши по различни начини, в зависимост от конкретните особености на тестваното приложение и резултатите, които се иска да се постигнат. В тази глава ще се разгледат най-често използваните подходи, видове тестове и добри практики, които е необходимо да бъдат познавани при определяне на стратегиите за изследване на производителност на дадено приложение/система. Някои от тях са лесно изпълними, докато други изискват значителни времеви, технически и човешки ресурси.

3.1 Подходи

Тестването за производителност може да се окаже трудна и неясна задача, ако не се следва ясно дефиниран подход. От съществена важност е да се съберат необходимите изисквания и да се анализират бизнес нуждите много преди започване на самите тестове. Изискванията могат да се базират на данни от предишни тестове или версии на системата, или на оценка за очакваната производителност на системата. След като ясно се дефинира какво ще се тества, може да се премине към решаване на въпроса как да се тества.

3.1.1 Подход „Benchmarking”

В ранните етапи на процеса на разработка на системата, обикновено се предпочитат така наречените „benchmark” тестове. С тях могат да се постигнат повторяеми(възпроизводими) резултати за относително кратко време и да се следи за възможни дефекти при разработката на системата. Повторяемите резултати спомагат „да се придобие увереност в тестваното приложение и получените резултати” *(Kumaran Systems, 2006)*.

При този подход идеята е да се променя само един параметър между отделните тестове. От съществено значение за тестовите резултати е и това как се осъществява натоварването на системата. Според някои автори *(Matt Maccaux, 2005),* натоварването се предизвиква от два фактора – от една страна това е броят на виртуалните потребители, които взаимодействат със системата едновременно, а от друга - това е времето, което е нужно на всеки потребител между заявките му към системата (т.нар. „think time”). Колкото повече потребители взаимодействат по едно и също време със системата, толкова повече трафик ще се генерира към нея. Също така, колкото е по-кратко времето между заявките на всеки потребител към системата, толкова по-голямо ще е натоварването на системата. Така, комбинирайки тези два фактора по различен начин, може да се генерира различно натоварване на системата.

3.1.2 Подход „Capacity Planning”

При подхода „Capacity Planning” основната цел е да се провери поведението на системата и докъде тя може да издържи на натоварване при различни обстоятелства. За разлика от „benchmark” тестовете, тук повторяемостта не е от толкова голяма важност, тъй като често в тестовете се включва случаен фактор. При този подход сценариите се стремят да се доближат максимално до реалността. Най-често целта е да се определи колко потребителски заявки може да обработи системата едновременно, преди времето за отговор на всяка една от заявките да надвиши определена стойност. Например, може да бъде поставено следното изискване: „Колко сървъра ще са необходими за инсталиране на системата, за да може тя да обработва едновременно заявките на 5000 потребители, като времето на всяка от тях да не надвишава 5 секунди?”. За да се отговори на такъв въпрос е необходимо да се знае допълнителна информация за конкретната система. Определянето на изискванията за производителността е от съществена важност при този подход. Според някои автори *(Adobe Systems Incorporated,2006)*, ако те не се планират добре това може да се окаже причината системата да не може да издържи на натоварване и да не оправдае потребителските очаквания към нея.

За да се определи капацитета на една система, трябва да се вземат под внимание няколко фактора. Според някои автори *(Matt Maccaux,2005)*, на първо място е броят на потребителите, които се очаква да могат да работят конкурентно със системата. След това е нужно и да се проучи какво ще бъде времето, нужно на всеки потребител между отделните му заявки към системата (т.нар. “think time”). За производителността на системата, това време е от критична важност – колкото е по-кратко, толкова по-малко потребителски заявки ще могат да бъдат обработени конкурентно от системата за единица време. Например, ако времето между заявките на всеки потребител към системата е 1 секунда, то системата ще може вероятно да обработи няколко стотици заявки едновременно. Но ако това време е 30 секунди, същата тази система ще може да поеме дори няколко десетки хиляди заявки. На практика, когато системата е пусната в реална експлоатация е много трудно да се предвиди какво време ще е необходимо на всеки потребител между заявките му към системата. Също така трябва да се отбележи, че това време не е едно и също между всеки две заявки.

Поради тези причини, в тестовете трябва да се предвиди фактор на случайност. Например, ако се знае, че средно на един потребител са нужни около 5 секунди между всеки две негови заявки към системата, то когато се съставят тестовете е нужно да се заложи по 5 секунди време между всяко действие (натискане на бутон, въвеждане на данни в текстово поле и т.н.). Допълнително, може всеки потребител след като изпълни определен брой заявки да изчаква случаен период от време (например 2 секунди) преди да продължи по-нататък. Така, комбинирайки тези два метода, тестовете могат да се доближат повече до действителността.

Следващият важен момент при “Capacity Planning” подхода е пускането на самите тестове. Тук идва въпроса как да се добавят виртуалните потребители, които ще симулират натоварването към системата. Това зависи от особеностите на конкретната система. Ако се очаква натоварването на системата да става постепенно, то се използва подход при който на всеки x секунди се добавят по y

на брой потребители. Това е така наречения "**ramp-up**" стил. Ако се очаква, че всички потребители за даден период от време ще взаимодействат със системата по едно и също време, то тогава виртуалните потребители трябва да се добавят заедно. Това е така наречения "**flat-run**" стил. Двата стила са различни и съответно дават различни резултати, които сами по себе си са несравними. Например, ако с "ramp-up" стила се открие, че изследваната система може да поддържа 5000 потребители с време за отговор на всеки от тях не повече от 4 секунди и се повтори същия тест с "flat-run" метода, то вероятно средното времето за отговор за 5000 потребители ще е по-голямо от 4 секунди. Разликата се дължи на характерната за "ramp-up" стила неточност, която го прави неподходящ за прецизно изследване на броят на конкурентните потребители, поддържани от системата. Това обаче не означава, че този метод не трябва да се използва. Той е особено подходящ за системи, при които се очаква натоварването да става постепенно за голям период от време, защото така системата ще може непрекъснато да се настройва през времето. Тук възниква въпроса кой от двата метода трябва да се използва, за да се определи капацитета на системата. На практика, ако се комбинират и двата подхода и се проведат серии от тестове, ще се получат най-добрите резултати. Например, с "ramp-up" може да се определи някакъв приблизителен интервал на броя поддържани от системата потребители. След това, със серия от "flat-run" тестове с променлив брой конкурентни потребители, който е число от този интервал, може да се определи по-точно и прецизно капацитета на системата.

3.1.3 Подход "Soak'

При подхода "Soak", тестовете за производителност се извършват за дълъг период от време и са със статичен брой конкурентни потребители. Те имат за цел да проверят цялостното поведение на системата през времето, нейната устойчивост и стабилност. По време на тестовете се следи за спад в производителността, дължащ се например на неосвободена памет или други проблеми в системата. Такива проблеми могат да се открият именно при този подход, за разлика от останалите подходи, където тестовете се извършват за сравнително кратко време *(RPM Solutions Pty, 2004)*. Колкото по-дълъг е един тест, толкова повече сигурност ще се придобие към системата. Според някои автори *(Matt Maccaux,2005)*, добра практика е тестовете да се пускат два пъти – веднъж със среден брой конкурентни потребители (но под максималния капацитет на системата, така че да няма нужда от чакаща опашка с необработени заявки) и веднъж със голям брой конкурентни потребители (такъв, че да се образува чакаща опашка с необработени заявки).

Продължителността на "Soak" тестовете трябва да бъде от порядъка на поне няколко дни, за да се добие представа какво е състоянието на системата във времето.

Тестовите сценарии трябва да се доближават максимално до действителността и ако е възможно, да изследват всички функционалности на системата.

3.1.4 Подход “Peak-Rest”

При подхода “Peak-Rest”, тестовете са хибрид от “capacity planning” и “soak” тестове. Целта е да се провери как системата се възстановява между периоди, в които е достигнала върховото си натоварване (т.нар. “peak” периоди) и такива, в които тя стои неизползвана (т.нар. “rest” периоди). Според някои автори *(Matt Maccaux,2005)*, най-добрият начин за провеждане на тестовете, е първо да се пуснат серия от “capacity planning” тестове и се достигне максимално натоварване, а след това да се освободи натоварването на системата. След кратка пауза, тестовете трябва да се повторят отново и да се анализира дали след достигането на максимално натоварване, системата винаги се възстановява по един и същи начин, или все повече се забавя производителността и (например системата може да не освобождава напълно заетата памет и в един момент да се окаже, че няма налична свободна памет). Колкото повече се повтарят тестовете, толкова повече информация ще има за това какво е състоянието на системата и дали тя е стабилна за по-дълъг период от време.

Кой от всички подходи да се избере при провеждането на тестовете за производителност силно зависи от бизнес нуждите, естеството на приложението, и процеса на разработка, който се използва. Същевременно, трябва да се потърси отговор на следните основни въпроси, които ще подскажат кой метод е най-подходящ за съответното приложение:

- Трябва ли резултатите да са повторяеми?
- Колко цикъла на провеждане на тестовете биха могли да се извършат?
- В коя фаза от цикъла на разработка сме в момента?
- Какви са бизнес изискванията?
- Колко време е необходимо да работи системата, когато тя се пусне в реална експлоатация, преди да се спре временно за поддръжка?
- Какво е средното очаквано натоварване през един работен ден?

След като се даде отговор на тези въпроси и се анализира кой от подходите най-добре ще ги покрие, може да се състави ясен план за цялостния процес на тестване на производителност на съответната система.

3.2 Видове тестове

За да се събере информация за производителността на приложението и да се отчетат стойностите на метриките, се използват различни видове тестове. Най-известните и често използвани са следните:

- Load тестове
- Scalability тестове
- Transaction Cost Analysis (TCA)

Load тестовете са най-популярните видове тестове. С тях може да се замерят и трите метрики (време за отговор от страна на системата, капацитет и използване

на системните ресурси) при различно натоварване на системата. Целта на този вид тестове е не само да се събере информация за метриките и да се анализират характеристиките на приложението, но също така да се определи предела на неговите възможности (т.е. да се определи каква е горната граница по отношение на производителността). На практика се извършват поредица от едни и същи тестове, всеки от които генерира все по-голямо натоварване на системата и това се прави докато тя започне да дава грешки или се срине напълно. Анализът на резултатите от тестовете помага да се идентифицират трудни проблеми, които при нормални тестове не биха могли да бъдат открити. Така може да се предотврати възможността те да се случат когато приложението се пусне в реална експлоатация. Тъй като най-често дефектите се дължат на грешки в дизайна на системата, то този вид тестове трябва да започнат да се планират и изпълняват още в ранните фази на процеса по разработка.

Scalability тестовете са набор от тестове, при всеки от които се следи как се изменя поведението на системата при различна хардуерна конфигурация. Целта е да се определи най-добрата възможна конфигурация на системата, когато тя е подложена на натоварване. Има два подхода за извършването на тези тестове – **vertical scalability** и **horizontal scalability**.

При **vertical scalability** тестовете, хардуерната конфигурация се заменя с по-бързи и по-мощни компоненти. Типичен пример е да се увеличи броят на процесорите (CPUs) или паметта на сървъра, на който е инсталирана системата.
Ако след извършване на тестовете се докаже, че производителността на системата расте пропорционално с добавянето на нов ресурс (в конкретния пример, с нов процесор), то се казва че системата е вертикално скалируема. Една полезна настройка при Windows операционните системи, е възможността да се посочи колко от наличните процесори на машината да са активни. Това става като се конфигурира стойността на параметъра *NUMPROC* във файла boot.ini.

При **horizontal scalability** тестовете се увеличава броят на сървърите, върху които е инсталирана системата. Това е още така наречената техника на разпределяне на натоварването (load balancing). При извършване на тестовете се конфигурира клъстер от машини с помощта на специален софтуер или хардуер (load balancer).Изпълняват се няколко теста с различен брой сървъри, участващи в клъстера. Ако производителността на системата нараства пропорционално с добавянето на нов сървър, то се казва че изследваната система е хоризонтално скалируема.

Transaction Cost Analysis (TCA) е вид тест, който цели да формулира връзката между капацитета на изследваната система и консумираните от нея ресурси. С негова помощ може да се определи колко ресурси се изразходват за завършването на една транзакция от страна на системата. Транзакцията само по себе си е абстрактно понятие и то се дефинира в зависимост от спецификите на конкретната система – това може да е определена функционалност, или пък задача, която системата извършва (например добавяне на артикул към кошница, ако системата е електронен магазин). Такъв анализ най-често се използва при планиране на капацитета на една Web система. Например, ако се изчисли, че е нужен 20 MHz процесор за обработката на една заявка за секунда, то тогава трябва

да се планира поне 500 MHz процесор ако се изисква да се обработват конкурентно по 25 заявки за секунда.

Кой от видовете тестове да се избере тук зависи не само от особеностите на конкретната система, но също така от наличния бюджет и времевите ресурси.
Всеки един вид тест отчита статистики за производителността на дадено приложение, които обаче са предназначени за различни цели и дават различна гледна точка за състоянието му *(Adam Neat, 2004)*. Тъй като scalability тестовете изискват допълнителни или по-мощни машини, то те няма да могат да се изпълнят, ако такива не са налични или е невъзможно да се закупят. За TCA тестовете трябва да се предвиди време за проучвания, анализ, така че да може най-добре да се дефинира какво ще представлява една транзакция за конкретната система. На практика, в зависимост от изследваната система, поставените изисквания, времеви ограничения могат да се предпочетат само някои от трите вида тестове, а при необходимост от цялостно тестване във всички аспекти – да се използват всички тестове.

3.3 Практики

Въпреки, че проблемите, свързани с производителността на една Web базирана система могат да имат най-различни причинители, резултатите от тях винаги са едни и същи – неустойчива, ненадеждна и бавна система. При разрешаването на такива проблеми, свързани с производителността, най-добре е да се предприеме цялостен подход, където системата, заедно със съпровождащите я софтуер, хардуер и мрежова инфраструктура се разглеждат и оценяват.

Проблемите, свързани с производителността могат да бъдат причинени от различни фактори: лошо проектирана система, слаби системни ресурси, не добре конфигурирана мрежа или комбинация от изброените. Например, ако системата бъде натоварена повече от очакваното, то тогава лесно могат да се изчерпат наличните системни ресурси. Не винаги обаче по-голямото натоварване ще доведе до проблеми с производителността. Ако системата е зле проектирана и не обработва правилно заемането и освобождаването на системни ресурси, това може да доведе системата до безизходно положение (deadlock) дори и натоварването да е нормално.
Независимо какво би причинило проблеми по отношение на производителността на една Web базирана система, първата стъпка по разрешаването им е да се състави план за производителността – дори и това да е съвсем кратък документ. В съставянето на плана трябва да участват най-значимите хора по проекта – ръководителя на проекта, старши разработчиците и инженерите по качество.

При анализът на производителността трябва да се следва цялостен подход. Една Web базирана система е не само един сървър, база от данни и хиляди редове код, а едно свързано цяло. Например, оптимизирането на SQL заявките или закупуването на оптика за мрежата няма да подобрят производителността, ако проблемите в системата се дължат на липса на памет или слаб процесор. Затова,

най-добрият подход е този, който изследва всички части на системата и как те си взаимодействат по между си от гледна точка на производителността.
В основата на всяка една Web базирана система стоят три компонента - софтуер, хардуер и мрежа. Дори и само един от тях да не е настроен добре, това би довело до проблеми в производителността. Следователно, за производителността на една Web базирана система са нужни:

- добре проектирана софтуерна архитектура
- бърз хардуер
- добре конфигурирана мрежа

Добра практика по време на проекта, е планирането и отделянето на време за прегледи на дизайна на системата и разглеждане на потенциални сценарии, които биха я довели до безизходно състояние (deadlock). Също така при разработката на самата система е полезно да се записва служебна информация, която би помогнала значително при евентуално разрешаване на проблем в производителността.

В идеалният случай, проблемите в производителността трябва да се открият и разрешат още във фазата на тестване, преди пускането на самата система в реална експлоатация. На практика обаче, тестовата среда и тази, в която реално ще работи системата не са еднакви. Дори и в случаи, когато те са идентични, натоварването към системата, генерирано от един тест е само едно приближение на реалното натоварване, на което ще бъде подложена системата на практика. Затова, често проблемите по производителността на системата се появяват след като тя бъде пусната в реална експлоатация. Ключов момент тук, е способността да се разрешат евентуалните проблеми, без да се губи твърде много време за предположения и догадки коя част на системата има нужда от по-нататъшно изследване. По този начин биха се избегнали загубата на време, излишните разходи и да се предотвратят потенциални бизнес загуби. Ето защо предварителното планиране на възможните проблеми по производителността на една система е от изключителна важност. Самият план трябва да бъде цялостна стратегия, в която ефектите от евентуални промени в системата са анализирани и съобразени със съответния хардуер и мрежова инфраструктура. Ето една примерна стратегия, която може да се следва при разрешаването на проблем в производителността *(Arash Barirani and Jeffrey Blake,2004):*

1. **Установяване на проблема**
    a. **Класификация на проблема**: сриване на системата, отказ да работи, забавяне и т.н. Това е важно, тъй като всеки сценарий има свои специфични признаци, които биха помогнали при изясняване на причините за проблема.
    b. **Възпроизвеждане на проблема**: Ако проблемът се случва в реална среда, то трябва да се направи всичко възможно, той да се възпроизведе в контролирана тестова среда. Възпроизвеждането на проблема ще позволи разбирането на същността на проблема и съответно неговото тестване.
    c. **Записване на важна системна информация**: Проследяването на консумираните от системата ресурси като процесор, памет и дисково пространство е жизнено важно и може да бъде много полезно при

анализиране на поведението на системата. Добре е тази информация да се събира преди и след появяване на изследвания проблем, тъй като може да даде насоки за разрешаването му.

d. **Съставяне на графики на данните**: За да се разбере добре поведението на системата и събраните данни, е хубаво те да бъдат изразени визуално (с диаграми, графики и т.н.). По този начин данните са лесно достъпни за всички от екипа, разработващ системата.

2. **Включване на всички членове от екипа**: При цялостния подход за разрешаване на проблем в производителността на една система, трябва да се включат всички участници в проекта – системни архитекти, администратори на базата данни, инженери по качество и разработчици.
3. **Съставяне на план за действие**: В него трябва да са отразени анализа от евентуални промени в дизайна или кода на системата. Ако съществуват няколко варианта за разрешаване на проблема, то всеки от тях трябва да се отрази и след това да се избере най-добрия според конкретната ситуация.
4. **Точно и детайлно отразяване на промените в плана и предприетите стъпки**

Цялостната стратегия и планирането на усилията на всички участници в проекта са ключови елементи в разрешаването на проблеми по производителността и оптимизацията на Web базирани системи.

3.4 Обобщение

Независимо от особеностите на конкретната изследвана Web система, има някои доказали се подходи, видове тестове и праткики, които трябва да се познават, когато се определя как да бъде извършено тестването. Най-често използваните подходи са Benchmarking, Capacity Planning, Soak и Peak-Rest. Изборът на подход силно зависи от бизнес нуждите, естеството на приложението, и процеса на разработка, който се следва. За отчитане на метриките за производителността се използват различни тестове. Основните видове са Load, Scalability и Transaction Cost Analysis (TCA) тестове. Кой от тях да се избере зависи както от особеностите на конкретната система, така и от наличния бюджет и времевите ресурси. Добра практика при процеса на тестване за производителност е следването на цялостен подход и включването на всички участници в проекта, при взимането на решения и разрешаването на проблеми.

4 Инструменти за тестване на производителност

В процеса на разработване и внедряване на всеки един продукт, който има поставено изискване да може да работи стабилно при големи потребителски натоварвания, се използват инструменти за тестване на производителност. На пазара има голямо разнообразие от такива инструменти. Всеки от тях предлага функционалност, с помощта на която могат да се установят, изследват и подобрят характеристиките за производителност на конкретното приложение/система.

Всеки от производителите на такъв тип инструменти, предлага продукт който позволява на потребителите да създават и изпълняват пълноценни тестове за производителност, независимо дали тестваното приложение е Web услуга, Web приложение работещо по различни протоколи или използващо различни технологии. Тези инструменти могат да симулират реални потребителски натоварвания, максимални или продължителни натоварвания, да записват резултатите и да ги преобразуват в най-разнообразни графики, които да са в помощ на инженерите по качество, изследващи и анализиращи поведението на тестваната система.

Повечето инструменти за тестване на производителност работят по сходен начин:

1. Създаване на тестови скриптове на език, специфичен за всеки инструмент – това най често се прави чрез записване на реална потребителска работа с тествания продукт и при необходимост – ръчна модификация на записания скрипт.
2. Допълнителни настройки за създадените скриптове – виртуални потребители (брой, потребителско име/парола), продължителност, поведение на теста по време на изпълнението му (променливо или постоянно натоварване). Комбинирайки тези настройки могат да се дефинират разнообразни тестови сценарии.
3. По време и след изпълнение на тестовете се предлагат разнообразни резултати и графики.

Различните инструменти се различават по езика за тестовите скриптове, по допълнителните настройки и резултати които предлагат, по това дали предоставят възможност за използване на повече от една машина за изпълнение на тестовете и т.н.

В тази секция, ще бъдат разгледани някой от най-популярните инструменти, предлагани на пазара. В по-големи детайли ще се разгледа инструмента Microsoft ACT, тъй като той ще се използва в практическата задача.

4.1 OpenSTA (http://opensta.org/)

OpenSTA е инструмент за тестване на производителност на Web приложения използващи протокола HTTP. Разработен е на езика C++. С помощта

на скриптов език се симулират действията на даден потребител (като попълване на форми, достъп до файлове, времето между отделните заявки – think time). Скриптовете могат да се генерират автоматично, с помощта на механизъм , който записва заявките към системата и съответните им отговори. За да се симулира голям брой потребители, всеки тестов скрипт(сценарий) може да бъде изпълнен по едно и също време от много на брой виртуални потребители, като за всеки от тях се отделя по една нишка.

Цялата тестова конфигурация се управлява през графичен интерфейс. Преди да се пусне даден тест се определят:

- скриптовите сценарии
- компютър, който ще ги изпълнява
- броят на виртуалните потребители

По време на изпълнение на тестовете, могат да се проследят различни системни показатели, като използване на процесора и дисковото пространство, мрежовия трафик, грешките в HTTP отговорите на системата и други. Един тест може да бъде разпределен на няколко машини, като предварително се конфигурира коя машина кой тестов скрипт ще изпълнява. Резултатите могат да бъдат разглеждани с графики и да се запишат във файл.

Положителни страни:

- Лесен за употреба графичен интерфейс
- Записващ механизъм и автоматично генериране на скриптове

Отрицателни страни:

- Създаден само за Windows операционна система
- Поддържа само HTTP протокол

4.2 TestMaker

TestMaker е програма с отворен код, разработена на Java. За писане на тестови сценарии се използва скриптовият език Jython (хибриден език, базиращ се на езиците Java и Python). За разлика от повечето инструменти за тестване на производителност, TestMaker не поддържа автоматично генериране на скриптове - те трябва да се програмират ръчно. Затова се изискват и добри познания по Jython и Java. Представянето на резултатите зависи от конкретния избор на разработчика на теста, но най-често се препоръчва да се използват графичните библиотеки на Java.

По време на тестовете, не се извлича много информация за тестваната система. Единствените налични резултати са свързани със параметри на отговора, връщан от системата при отправена към нея заявка. Възможно е извличането на следните параметри:

- Целия отговор, върнат от системата в резултат на дадена заявка
- Времето, което изминава от изпращането на дадена заявка до получаване на целия отговор
- Кодът на отговора, върнат от системата

Положителни страни:

- Поддръжка на множество платформи

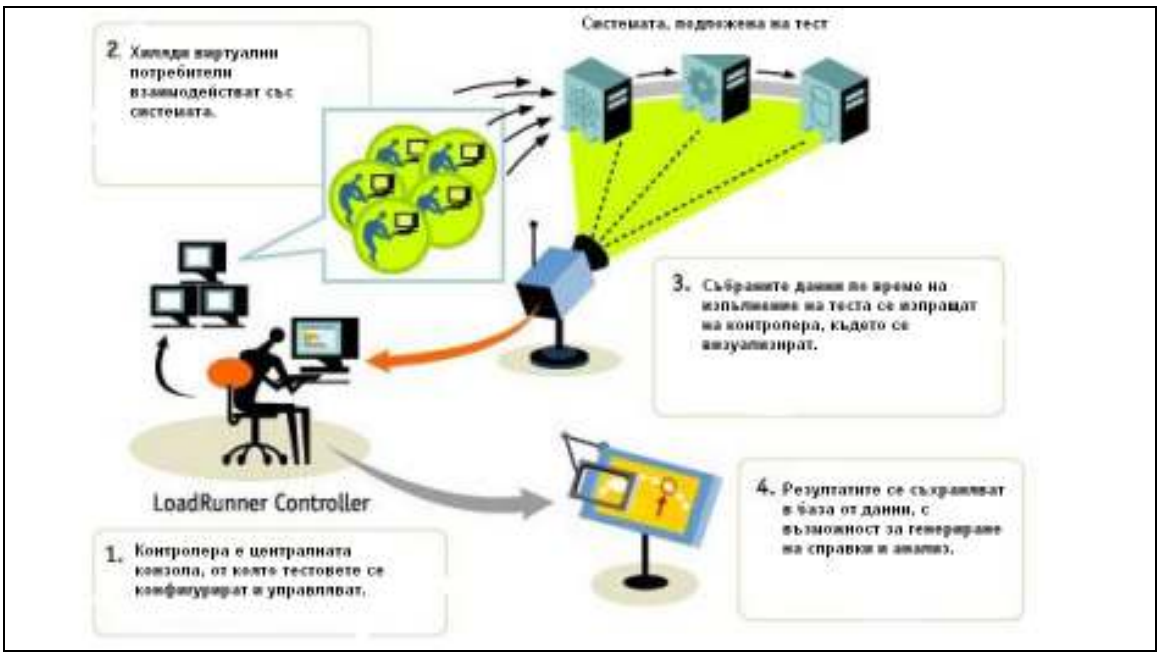
- Всички предимства на езика Java

Отрицателни страни:

- За да се създаде един тестов сценарий се изискват солидни познания по Java и Jython, както и той да се напише изцяло. Това може да се окаже трудна и времеемка задача.
- По време на тестовете се извлича информация само за отговорите, които системата връща при генерирането на заявки към нея.

4.3 LoadRunner(http://www.mercury.com/us/products/performance-center/loadrunner/)

LoadRunner е комерсиален инструмент, предназначен за анализ на производителността и поведението на разнообразни приложения (Web, бази от данни, Java и др.). Поддържат се както Windows, така и Linux платформи. Фигура 4-1 илюстрира как работи LoadRunner:



Фигура 4-1: Принцип на работа на инструмента LoadRunner

Създаването на скриптове с този инструмент е сравнително лесно, благодарение на записващия механизъм, който проследява заявките на браузъра и автоматично генерира тестов скрипт. LoadRunner е изключително подходящ за „Capacity planning" тестове, тъй като има възможност виртуалните потребители да се добавят постепенно (на определен период от време), а не всички заедно.

Инструментът може да работи и в разпределена среда на няколко машини. След като тестът завърши, резултатите от всички машини, на които той се е изпълнявал се обединяват в един документ. В него се обобщават стойностите на някои ключови показатели за системата, като общ брой виртуални потребители,

средно време за отговор от страна на системата, данни, преминали през системата за единица време, информация за отделните транзакции и други. Наред с това се записва и информация за системните ресурси като използвано процесорно време, памет, брой нишки и други. Събраните данни могат да се представят и в графичен формат.

Положителни страни:

- Лесен за употреба графичен интерфейс
- Записващ механизъм и автоматично генериране на скриптове
- Поддръжка на множество платформи

Отрицателни страни:

- Комерсиален лиценз

4.4 LoadSim (http://www.openware.org/loadsim/)

LoadSim е инструмент с отворен код, разработен на езика Java. Предназначен е за тестване на производителност на Web приложения, използващи протокола HTTP. LoadSim може да прихваща заявките на браузъра и автоматично да генерира тестови скриптове. Това, което го отличава от повечето инструменти за тестване на производителност, е че няма графичен интерфейс. Един тест може да се изпълнява на няколко машини, които предварително се дефинират в конфигурационен файл. Конфигурацията на всяка от машините може да е различна (например различен брой виртуални потребители, различен тестов скрипт). Резултатите от тестовете се записват във файл. Не се поддържа анализ на данните, както и представяне в графичен формат.

Положителни страни:

- Записващ механизъм и автоматично генериране на скриптове
- Всяка машина може да има различна тестова конфигурация

Отрицателни страни:

- Няма графичен интерфейс
- Няма графично представяне на данните

4.5 Microsoft ACT

Microsoft Application Center Test (ACT) е инструмента за тестване на производителност на Web приложения на компанията Microsoft. Разпространява се заедно с Visual Studio .NET. Използва се за изследване на поведението на Web системи, подложени на натоварване, както и за анализ на проблемите, свързани с производителността.

Инструментът ACT се характеризира със следните особености:

- Симулация на голям брой виртуални потребители, взаимодействащи едновременно със изследваната система

- Възможност за автоматично генериране на скриптове, както и тяхното ръчно програмиране
- Съвместимост с всички Web приложения, базирани върху протокола HTTP
- Поддръжка на протокола SSL (Secure Socket Layer) и възможност за тестване на защитени сайтове.

По време на изпълнение на даден тест се събира информация за изпращаните към системата заявки. Метриките TTFB (времето до първия байт на отговора), TTLB (времето до последния байт на отговора) и RPS (брой заявки за секунда) са полезни за определянето на капацитета на системата. Същевременно се отчитат консумираните от системата ресурси като процесор, памет, дисково пространство и други.

За да се изследва и замери правилно производителността на една Web система, е необходимо тестовете да се извършват в среда, максимално близка до реалната. В идеалния случай тестовете трябва да се извършват в истинската работна среда на системата, но на практика това рядко е възможно тъй като това може да попречи на бизнеса. Задължително е сървърът и клиентът да са на различни машини, за да се отчете правилно консумацията на системните ресурси. Най-често се препоръчва да се използва ACT на клиентската машина за генериране на HTTP заявки към системата и замерване на метриките TTLB (времето до последния байт на отговора) и RPS(брой заявки за секунда). В същото време, на машината на която се намира изследваната система може да се използва Windows Performance Monitor за да се отчете как тя консумира системните ресурси докато е подложена на натоварване.

Процесът на създаване и изпълнение на тест с ACT включва следните основни стъпки:

4.5.1 Създаване на проект

Проектът е контейнер за един или няколко свързани тестове, формиращи тестова сюита. Всеки проект има лог файл, в който се съдържат резултатите от използваните **Test.Trace** методи в тестовите скриптове (ако има такива, разбира се).

4.5.2 Създаване на тест

Тестовият скрипт може да се създаде както ръчно, така и автоматично с помощта на записващ механизъм, които записва заявките, изпратени към системата. Когато се използва автоматичното генериране на скрипт, може да се наложи допълнително да се направят модификации за нуждите на тестовия сценарий.

4.5.3 Модификация на тест

Обикновено, за нуждите на конкретния сценарий се налага да се модифицира предварително записания тестов скрипт. Ето параметрите, които най-често се нуждаят от допълнителна настройка:

a) Think Time

Когато ACT изпълнява тестовете, той използва пълния капацитет на машината на която е стартиран. Така тестваното приложение приема максимален брой заявки, но на практика това не отразява реалната му работна среда. За симулиране на среда, максимално близка до реалната е добре да се използват малки забавяния между отделните заявки на всеки виртуален потребител. Това се постига с извиквания на метода **Test.Sleep**.

b) Authentication

ACT поддържа различни методи за автентикация – Anonymous, Basic, Integrated Windows, Passport. Последните два механизма не се поддържат автоматично по време на записването на тестовия скрипт и затова се налага те да се конфигурират ръчно след това.

c) Tracing

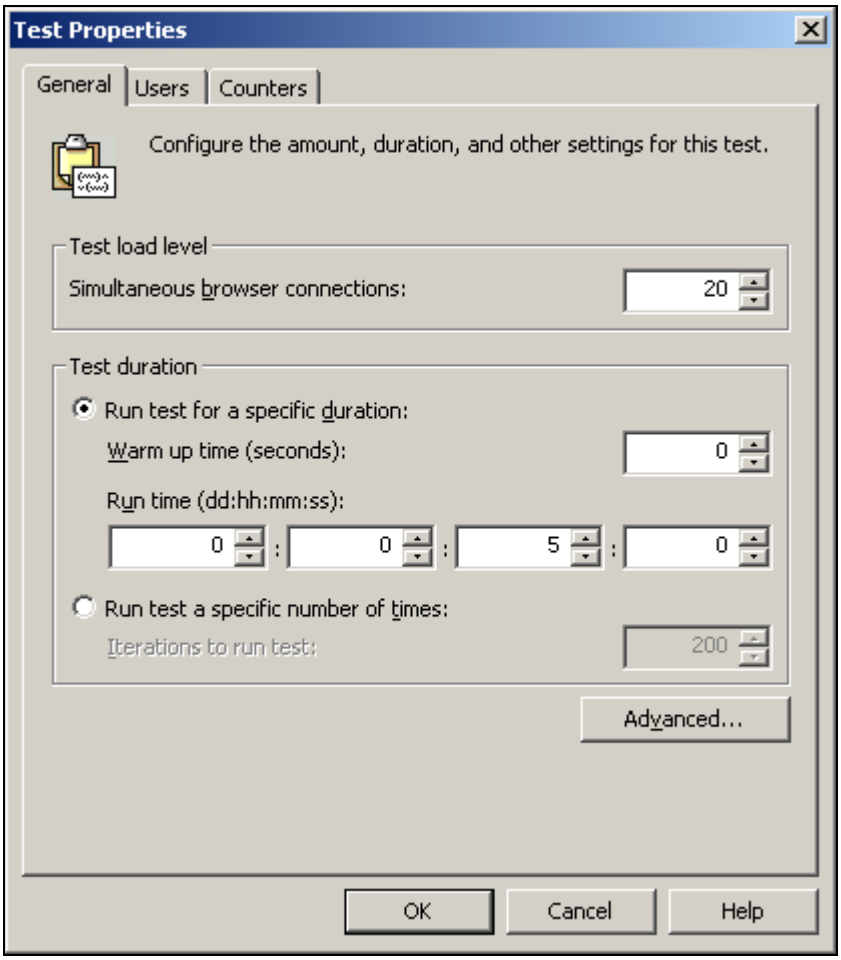
След записването на скрипта могат да се вмъкнат Test.Trace методи, които са особено полезни при диагностицирането на проблеми. С помощта на параметър може да се определи колко да е детайлна информацията

4.5.4 Настройка на параметрите на теста

Преди пускането на даден тест е нужно да се конфигурират някои параметри, съобразно поставените изисквания. Параметрите, които могат да бъдат настроени са следните:

a) Simultaneous Browser Connections (Test Load Level)

Този параметър определя броя на паралелните връзки към системата и съответно генерираното натоварване. Колкото са връзките, толкова копия на тестовия скрипт ще се изпълняват едновременно по време на теста. Необходимо е, броят на виртуалните потребители да е по-голям или равен на броя на паралелните връзки, тъй като всяка инстанция на браузъра взаимодейства със системата с различен виртуален потребител. Ако например се изисква да се симулират едновременно 20 връзки към системата, то стойността на този параметър трябва да е 20, както е показано на фигура 4-2:



Фигура 4-2: Настройка на параметрите на теста

b) Test Duration

За продължителността на даден тест може да се избира измежду две възможности:

i. **Run test for a specific duration** – този параметър се използва, когато е нужно тестът да се изпълнява за определен период от време (например 20 мин.). Периодът от време трябва да се определи много внимателно, в зависимост от дължината и степента на сложност на скрипта. Задължително е обаче да могат да се извършат основните операции със системата. Обикновено, ако извършването на набор от действия със системата отнема време t1, то се препоръчва тестът да се изпълнява за време 2* t1. Когато се изследват сериозни проблеми като например изтичане на памет е нужно тестът да се изпълнява за по-дълго време.

ii. **Run test a specified number of times** – този параметър се използва, когато е нужно тестът да се изпълни определен брой пъти. Трябва да се има предвид обаче, че броят на итерациите и броят паралелни връзки към системата (simultaneous connections) са взаимно свързани. Например, ако са конфигурирани 10 паралелни връзки, които да бъдат отворени към системата и тестът се изпълни 5 на брой итерации, то общо ще се отворят 500 връзки към системата.

c) Warm-up Time

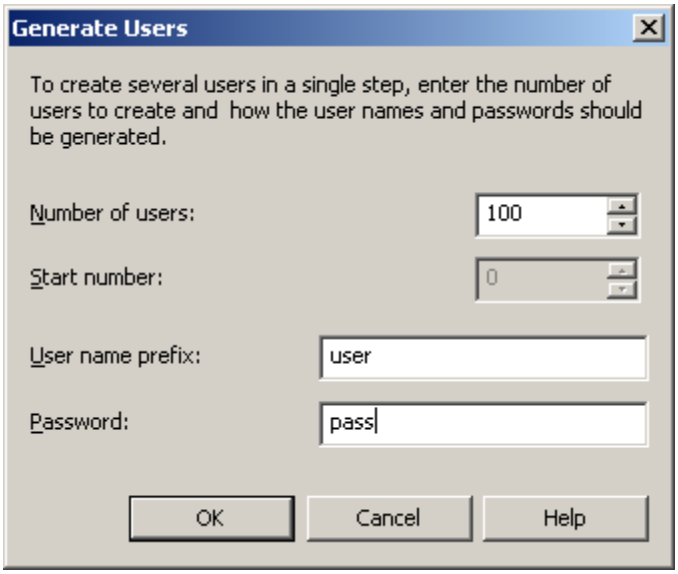
Обикновено, при пускането на даден тест отначало има период на забавяне на времето за отговор от страна на системата. Причините за това са, че е нужно време за да се инициализират всички обекти и се достигне състояние на готовност. Това време може да изкриви тестовите резултати. Именно за това, ACT дава възможност за определяне време на изчакване преди да започне събирането на същинските данни за производителността на системата. Обикновено 10 секунди е достатъчно време. За по-голяма прецизност, може да се проследи колко време изминава докато стойността на използване на процесора(CPU) от системата се стабилизира и съответно това време да се посочи като warm-up time.

d) Number of Users

Този параметър определя броят виртуални потребители, взаимодействащи едновременно със системата. И тук има две възможности: потребителите могат да се генерират автоматично от ACT по време на теста, или пък да се конфигурират ръчно (но задължително броят им трябва да е по-голям или равен на броя паралелни връзки към системата, тъй като всяка инстанция на клиентския браузър използва различен потребител). Когато е избрана втората възможност, потребителите могат да се добавят поотделно, да се генерира определен брой, или пък да се вземат от предварително зададен файл. Всеки ред от файла трябва да е във формата:

Име, Парола

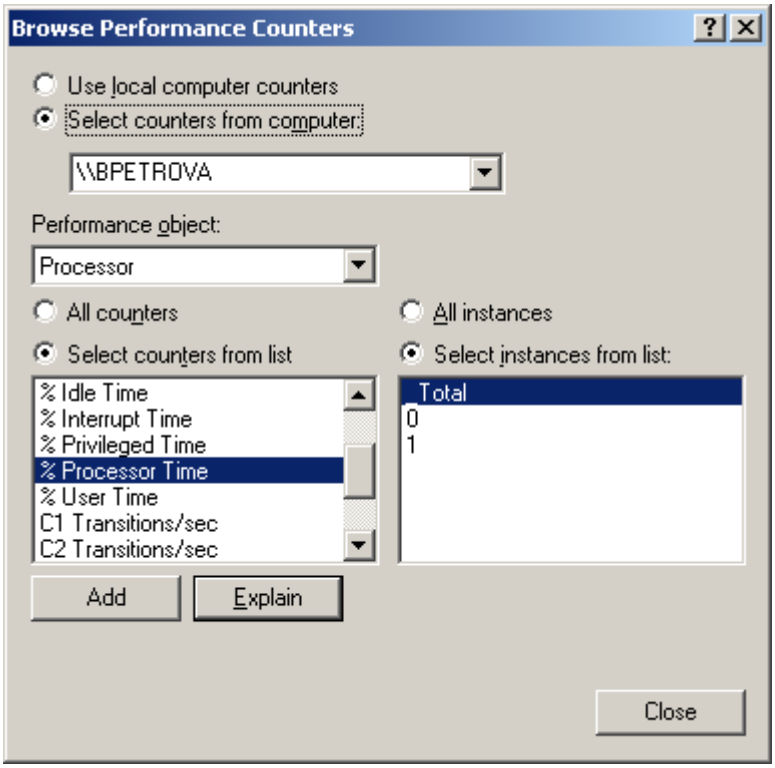
На фигура 4-3 е показано, как могат да се генерират 100 виртуални потребителя, като се използват префиксите „user” и “pass”, съответно за потребителско име и парола:



Фигура 4-3: Генериране на виртуални потребители

e) Performance Monitor Counters

Една от характеристиките на ACT е възможността за интегриране на метрики за производителността в тестовите резултати. Това, което е особено полезно е, че може да се събира информация за производителността на отдалечени машини. Така може едновременно да се наблюдават производителността на клиентската и сървърната машина (тази, на която е инсталирана системата, подложена на тест), а резултатите да са агрегирани в един документ. Фигура 4-4 показва интерфейса, чрез който се избират различните метрики които предоставя ACT.



Фигура 4-4: Добавяне на метрики за производителността

Най-важните метрики, които се отчитат на клиентската машина са използването на процесора и паметта (таблица 4-1):

Processor \ % Processor Time	Не трябва да се използва повече от 75%.
Memory \ Available Mbytes	Не трябва да се заема по-малко от 20 – 25 % от клиентската памет.

Таблица 4-1: Метрики, отчитани на клиентската машина

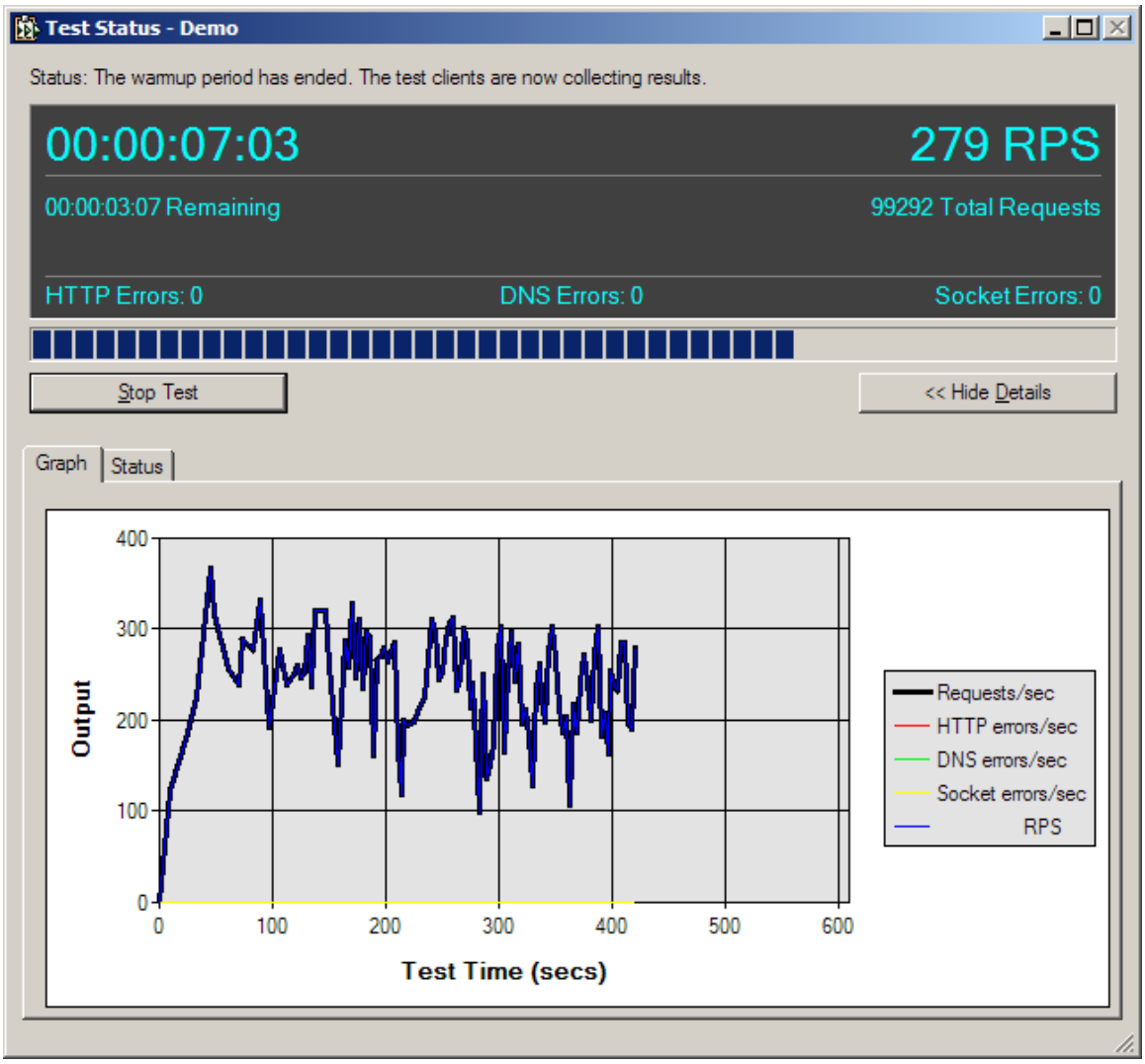
4.5.5 Изпълнение на теста

След като тестовият скрипт и параметрите са настроени, тестът се пуска за изпълнение срещу системата. Това може да бъде ръчно или автоматично.

4.5.6 Анализ на резултатите

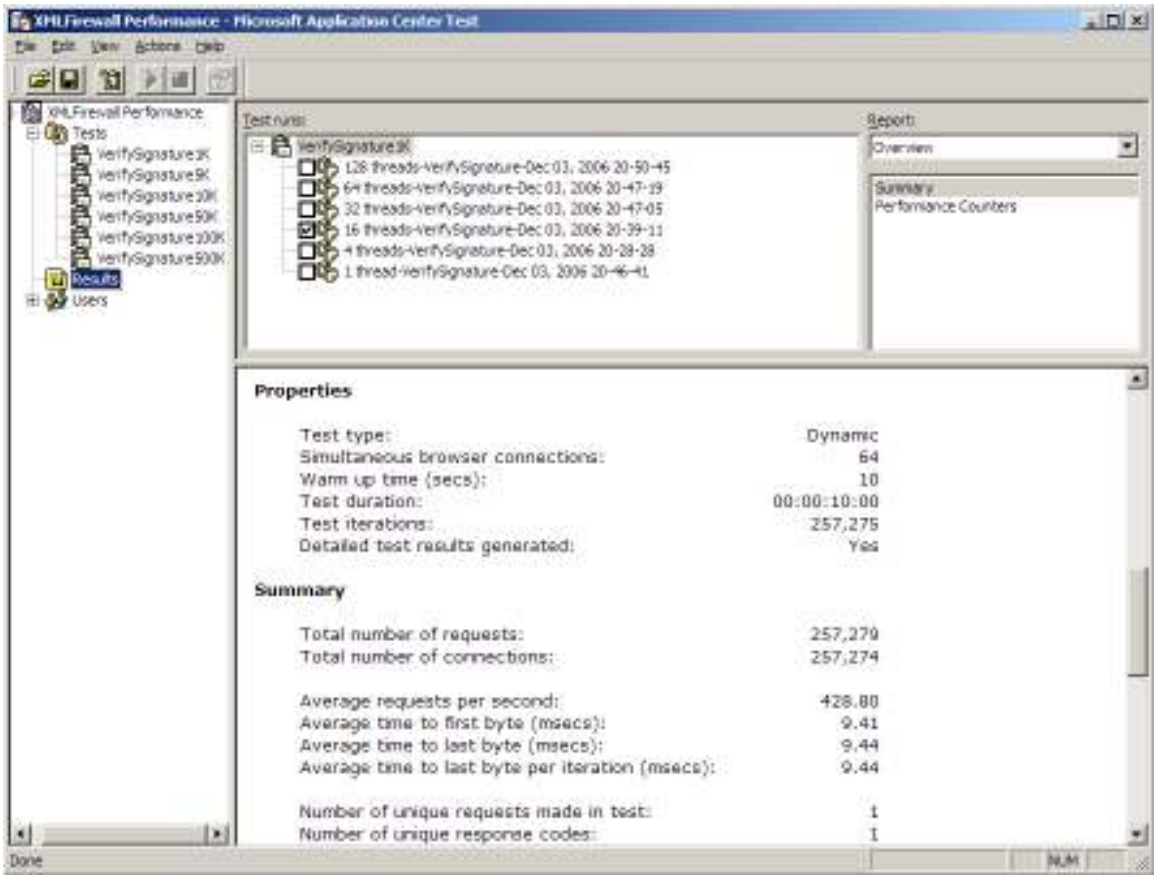
Резултатите се показват веднага със започването на изпълнението на теста и непрекъснато се обновяват (фигура 4-5). Отчитат се изминалото и оставащото

време, брой обработени заявки за секунда (RPS), както и три типа грешки: HTTP, DNS и Socket.



Фигура 4-5: Текущи резултати от изпълняващия се скрипт

След завършване на теста могат да се видят и окончателните резултати (фигура 4-6):



Фигура 4-6: Обобщени резултати от изпълнения тестов скрипт

За да се определи дали тестът е бил обективен е нужно първо да се проверят следните показатели:

- **Web Server processor usage**: За да се постигне максимално натоварване и съответно ефективен тест, трябва процеса на уеб-сървъра да е използвал поне 80% от процесорното време на машината на която е инсталиран в рамките на изпълнението на теста. Ако условията не са изпълнени, следва да се увеличи натоварването към системата.

- **ACT client processor usage**: Ако процесорът на клиентската машина се използва максимално (близо до 100%), докато този на сървърната не, то тогава клиентската машина няма да е способна да генерира достатъчно натоварване, за да определи капацитета на системата.

- **HTTP Errors**: Тестът ще се счита за невалиден ако се срещат голям брой грешки, дефинирани в HTTP протокола 4xx, 5xx (http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10).
  Например:

  - HTTP 403 (Server Too Busy)

- HTTP 404 (Not Found)
- HTTP 500 (Internal Server Error)

Грешки от този тип обикновено се появяват когато броят на конкурентните потребители е много голям и системата не може да се справи с обработването на генерираните заявки. На практика, всеки тест при който грешките са повече от **2.5 %** от общия брой заявки трябва да се счита за невалиден.

Таблица 4-2 илюстрира какво се съдържа в резултата от един тест:

Test Run Name	Име на теста
Date Started	Дата, на която тестът е стартиран
Total Run Time	Продължителността на теста в секунди
Total Iterations	Общ брой итерации на тестовия скрипт
Total Requests	Общ брой заявки, генерирани по време на теста
Connections	Брой на конфигурираните паралелни връзки към системата
Avg Requests/sec	Среден брой заявки за секунда
Avg Time To First Byte (msecs)	Средното време, изминало до получаване на първия байт от отговора (в милисекунди)
Avg Time To Last Byte (msecs)	Средното време, изминало до получаване на последния байт от отговора (в милисекунди)
HTTP Errors	Общ брой на HTTP грешки. Това са отговори, чиито кодове са в интервалите [400;499] и [500;599]
DNS Errors	Общ брой DNS грешки
Socket Errors	Общ брой Socket грешки

Таблица 4-2: Описание на метриките, съдържащи се в резултатите

Въпреки, че ACT предоставя много данни в тестовите резултати, ключовият момент е те да се анализират и съответно да се идентифицират проблемните места по отношение на производителността. В таблица 4-3 са изброени най-важните метрики, използвани при анализа:

Метрика	Измерена на	Описание
Browser Connections	Клиентска машина	Измерва натоварването
Requests per second	Сървърна машина	Измерва капацитета
Time To Last Byte (TTLB)	Клиентска машина	Измерва времето за отговор от системата
%Processor Time	Сървърна машина	Измерва консумацията на системни ресурси
Available Bytes (Memory)	Сървърна машина	Измерва консумацията на системни ресурси

Таблица 4-3: Метрики, използвани за анализ

Една от характеристиките на ACT е и възможността за генериране на графики от събраните данни по време на теста. Могат да се генерират разнообразни графики, но най-полезни са две от тях, показващи съответно как се изменя времето за отговор спрямо натоварването (response time vs.user load) и броят обработени заявки за секунда спрямо натоварването (throughput vs.user load).

Това, което отличава ACT от останалите инструменти за тестване на производителност са лесният/интуитивният му графичен интерфейс, бързото и точно генериране на желаното натоварване, поддръжка на няколко механизми за автентикация, възможността за тестване на защитени сайтове. Добрите статистики в резултат на изпълнение на тестовете, както и генерирането на разнообразни графики по зададени от тестовия инженер критерии помагат за организирането и анализа на тестовите резултати. Именно тези негови характеристики покриват най-добре целите, поставени за тестването на производителността на примерното предложение, което е предмет на следващата глава.

4.6 Обобщение

От разгледаните инструменти, може да се заключи, че повечето са предназначени за тестване на Web приложения, базирани на протокола HTTP. Някои се поддържат на много платформи, а други са предназначени само за Windows операционна система. Комерсиалните инструменти са много по мощни и по-добре

проектирани, отколкото отворените програми. По отношение на създаването на тестовите сценарии и скриптове има две възможности:

- Използването на записващ механизъм с автоматично генериране на скриптове (OpenSTA, LoadRunner, LoadSim)
- Изцяло ръчно писане на тестовите сценарии (TestMaker)

Изпълнението на самите тестове е сходно за повечето инструменти – обикновено за всеки виртуален потребител се отделя по една нишка. Резултатите представят важни характеристики като: брой виртуални потребители, средно време за отговор от страна на системата и др. Най-детайлна информация за поведението на изследваната система и консумираните от нея системни ресурси, се дава от инструмента LoadRunner.

Всеки от разгледаните инструменти има своите предимства и недостатъци. Изборът на инструмент за тестване на производителност трябва да бъде съобразен с особеностите на конкретната система. Все пак, ако изследваната система е голяма и сложна, от съществена важност е той да има следните ключови характеристики:

- Лесно и бързо създаване на тестови скриптове/сценарии, както и възможност за тяхната модификация
- Точни данни, анализи и поддръжка на графики
- Възможност за настройка на тестовата среда според особеностите на конкретната система

5 Тестване на производителност на приложението XML Firewall. Планиране и стратегия

5.1 Въведение

В сферата на съвременните информационни технологии все по-голямо значение и роля заемат Web услугите (Web Services). Web услугите са вид програмни компоненти, които предоставят дадена функционалност (услуга). Те имат следните характеристики:

- Достъпни са отдалечено през Web, с модел за изпълнение "заявка-отговор". Клиентът изпраща заявка, услугата я изпълнява и връща резултат.
- Използват отворени стандарти за комуникация:
  - HTTP (Hypertext Transfer Protocol)
  - XML (Extensible Markup Language)
  - SOAP (Simple Object Access Protocol)
- Сами описват интерфейса си за достъп чрез езика WSDL (Web Service Description Language). WSDL е XML базиран отворен стандарт на W3C (http://www.w3.org/), описващ интерфейса за достъп до дадена Web услуга, състоящ се от:
  - имената на поддържаните методи
  - входни и изходни параметри
  - използвани типове данни
- Работят на принципа на обмяна на SOAP съобщения (съдържат структурирана информация, състояща се от данни и метаданни).
- Независими са от операционната система, платформата и езика за програмиране

Целта на Web услугите е да се преодолеят различията между езиците, технологиите и продуктите на отделните производители, така че всички единици в мрежата да могат да комуникират пълноценно.

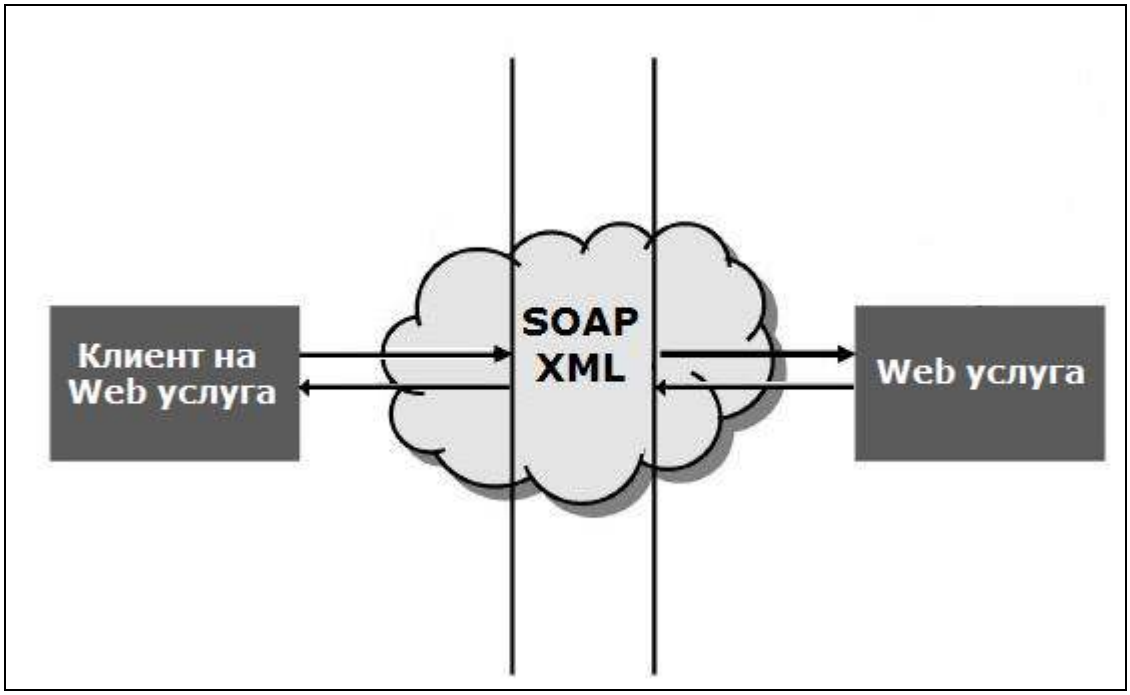
От друга страна, във всяка една организация са инсталирани и работят различни приложни програми. Същевременно, все по-голяма става нуждата въпреки различията помежду им, те да могат да си комуникират една с друга. Би било прекалено скъпо и време отнемащо обаче, те да се пренапишат изцяло. Тази интеграция е била почти невъзможна до момента на появата на Web услугите. Оказва се, че най-лесният и практичен вариант е, за всяка една такава програма да се създаде интерфейс от Web услуги. Това е така, именно защото Web услугите са платформено и езиково независима технология, която може да се използва навсякъде. По този начин се предоставя един напълно нов начин за интеграция, който много бързо се развива и се налага на пазара *(A. Zisman, R. Summers, S. Katz, F. Servan, and J. Chelsom, 2002)*.

Използвайки модела на Web Services, приложенията най-често комуникират през мрежата посредством протокола SOAP. SOAP също е един от отворените

стандарти на W3C консорциума. SOAP съобщенията се формират с помощта на езика XML. Едно SOAP съобщение се състои от две части:

- Заглавна част (SOAP Header) – описва параметри на съобщението (метаданни)
- Тяло (SOAP Body) – съдържа самото съобщение (изпратена заявка или отговор на заявка)

Обикновено SOAP съобщенията се предават по протокола HTTP. Фигура 5-1 илюстрира употребата на протокола SOAP за комуникацията между клиент и Web услуга:

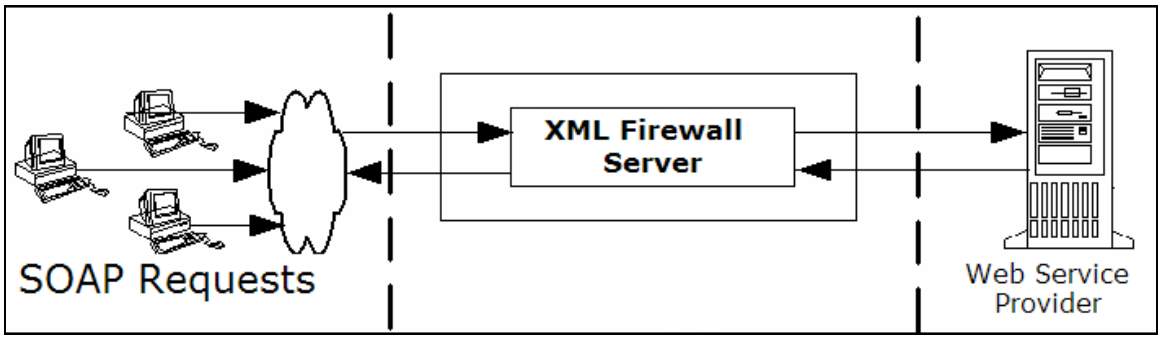


Фигура 5-1: Протоколът SOAP като средство за комуникация

На практика, изискванията към продуктите в областта на информационните технологии стават все по-големи и все повече системи ще трябва да се интегрират една с друга. Все повече организации използват Web услугите с тази цел и са изградили големи мрежи от Web услуги. Мрежите от своя страна ще стават все по-разнородни и сложни. Така постепенно идва нуждата от централизираното им управление, контрол на достъп и наблюдение. Всяка Web услуга може да има различни механизми за защита и контрол. Поради това, администрирането на всички Web услуги от едно централизирано място се превръща в една почти невъзможна задача. За да не се променя една вече изградена инфраструктура от Web услуги на дадена организация, се създават продукти които дават възможност за осигуряване на централизирана защита, управление и контрол. От такъв тип е и приложението XML Firewall, което е обект на този пример. Примерът демонстрира колко важна е ролята на тестването за производителност в процеса на разработване на този продукт.

5.2 Описание на XML Firewall

XML Firewall е продукт, чиято цел е да централизира, улеснява и защитава корпоративни мрежи от Web услуги. Той работи на принципа на прокси като защитна стена, намираща се между клиентите и мрежата от Web услуги. Комуникацията между тях не се осъществява директно, а минава през XML firewall-а (фигура 5-2).



Фигура 5-2: XML Firewall, действащ като защитна стена между клиентите и мрежата от Web услуги

Това има следните предимства:

- Преодоляват се различията между клиентите и мрежата от Web услуги по отношение на формат на данните, протоколи и механизми за защита.
- Предоставя се възможност за защита на Web услугите от неправомерен достъп, без да се налагат каквито и да е промени по вече създадената инфраструктура.
- Могат да се прилагат най-различни трансформации върху преминаващите заявки, съобразно поставени критерии.
- По време на преминаващия трафик през XML firewall-а се съхранява важна информация, която в последствие може да послужи за анализ.

Преминаващите през XML firewall-а SOAP съобщения се обработват съобразно предварително определени правила в неговия административен графичен интерфейс. Той е контролната точка, от която се администрират всички управлявани Web услуги.

Принцип на работа на системата:

Всяка съществуваща Web услуга, която ще се използва през системата XML Firewall първо трябва да се регистрира в нея. В резултат на това, описанието на регистрираната Web услуга в нейния WSDL файл се модифицира от системата. Променениият WSDL файл се предоставя на клиентите и те ще се обръщат към регистрираната в системата услуга, а не директно към първоначалната. Регистрираните Web услуги могат да се обединяват в групи, за които могат да се дефинират общи правила, включващи следната функционалност:

1. Поддръжка на протоколите HTTP, HTTPS, JMS
2. Сигурност (Security): Постига се със следните механизми:
    a. Контрол на достъпа (Access Control): Постига се чрез поддръжката на съвместимост с най-масовите средства, предоставящи автентикация. Това са така наречените автентикиращи директории (authentication directories), които са базирани на някои от протоколите за достъп до директории, каквито са например LDAP и SAML. Достъпа до даден ресурс може да се позволи/забрани с помощта на дефиницията на потребителски роли. Така защитените ресурси могат да се достъпят само от потребители, които се идентифицират като валидни потребители на някоя директория и имат роля, за която е разрешен достъп.
    b. Криптиране/електронно подписване на съобщенията
    c. Дешифриране/Валидация на електронно подписани съобщения с цел да се провери тяхната цялост
    d. Поддръжка на защитена комуникация посредством протокола SSL
    e. Поддръжка на спецификацията WS-Security
3. Изследване и модифициране на съдържанието. Постига се със:
    a. Възможност за извличане на съдържанието (Content Inspection) на важни полета от SOAP съобщенията - най-често, с помощта на езика XPATH.
    b. Модифициране на SOAP съобщенията по зададени критерии, използвайки езика XSLT.
    c. Валидиране на SOAP съобщенията по зададена XML схема или DTD дефиниция.
4. Записващ механизъм (Logging): Възможност за съхраняване на преминаващите заявки заедно с времето им за изпълнение и друга полезна информация. Събраната информация се записва в база от данни.

5.3 План/Стратегия за тестване на производителност на системата XML Firewall

По своята същност, XML Firewall е сървърна система, която приема и връща като отговор SOAP съобщения по всички поддържани протоколи (HTTP, HTTPS, JMS). Нефункционалните изисквания като бързодействие (concurrency), надеждност (reliability) и скалируемост (scalability) са жизнено важни за успеха на такъв вид продукт.

За тестването на производителността на системата XML Firewall е необходимо:

1. Първоначално да се направят проучвания и да се дефинират очаквани стойности
2. Да се дефинират цели, заедно с подходи и видове тестове за постигането им
3. Да се избере подходящ инструмент за тестване и писане на скриптове
4. Да се разработят тестовите скриптове

5. Да се извършат тестовете и се анализират получените резултати (обикновено се извършват няколко итерации докато се постигнат поставените цели)

5.3.1 Проучване и дефиниране на очаквани стойности

Системата XML Firewall действа като прокси между клиента и извикваната от него Web услуга. За да се дефинират подходящо изискванията относно производителността на системата могат да се вземат предвид статистиките събрани при работата на най-големите мрежи от Web услуги в големи корпорации. Съществената част от тези данни са пиковите натоварвания – колко заявки за секунда се обработват в мрежата, какво количество данни се обменя в тези моменти. Планирането на XML Firewall така, че да може да работи при подобни и по-големи натоварвания е жизнено важно за неговия успех на пазара.

За да бъде тестването на системата обективно, трябва да са налични достатъчно бързи Web услуги, които да бъдат регистрирани в продукта.
Тестването за бързодействието на Web услугите се извършва преди да е започнало същинското тестване на XML Firewall системата. За целта се използват най-масовите продукти, предоставящи база за създаване на Web услуги – Microsoft, BEA, IBM, AXIS. На базата на получените резултати може да се определи кои от тези продукти е най-добре да се използва при тестовете.

5.3.2 Цели, подходи и видове тестове за постигането им

Изследването на поведението на системата при натоварване ще включва следните тестове за производителност:

a. Определяне на бързодействието на системата

Цел: Да се измери времето за отговор от страна на системата, когато тя е подложена на натоварване.

Подход: Тестовете се изпълняват в среда, максимално близка до реалната среда на работа на системата, следвайки подхода “Benchmarking” описан в точка 3.1.1.

Реализация: За постигане на качествени и добри резултати е необходимо да се покрият колкото се може повече от сценариите, върху които се очаква да работи системата. Има две фази на протичането на тестовете:

В **първата** фаза всяка една функционалност трябва самостоятелно да бъде подложена на тест за производителност. Така могат да се идентифицират проблемните компоненти на системата (т.нар. “bottlenecks”). Съобразно характеристиките на системата, могат да се дефинират тестове, покриващи следните функционалности:

- Директно преминаване на заявка през системата (“pass-through”)

- Анализ на съдържанието на дадена заявка посредством валидация с XML схема или DTD дефиниция.

- Контрол на достъпа: Идентифициране на потребител с потребителско име и парола. Те могат да бъдат зададени по два начина:

    - HTTP Basic Authentication (Потребителското име и парола се предават на ниво протокол в HTTP заглавната част)

    - WS-Security Username token (Потребителското име и парола се предават в самата SOAP заявка в нейната заглавна част)

- Наблюдение на преминаващия трафик - записване на съобщенията или определени части от тях, използвайки различни бази от данни.

- Модифициране на самата заявка с помощта на XSLT трансформация

- Използване на различни комбинации от протоколи в рамките на една заявка, например:

    - Регистрираната услуга се достъпва по протокол HTTP, а XML Firewall достъпва оригиналната по протокол HTTPS

    - Регистрираната услуга се достъпва по протокол JMS, а XML Firewall достъпва оригиналната по протокол HTTPS

- Конфиденциалност – криптиране/декриптиране на заявки

- Предпазване от промени на съдържанието на заявката – електронно подписване на съобщенията и верификация на електронен подпис

Втората фаза включва тестване на потребителски сценарии - комбинации от вече тестваните сами по себе си функционалности, изследващи как те работят заедно. Целта тук е да се проверят най-важните и най-често използваните от потребителите сценарии.

Анализ: Първоначалното изпълнение на тестовете дава начални резултати за производителността на системата при натоварване. Сравнявайки ги с предварително изчислените очаквани резултати се набелязват тези компоненти на системата, които имат нужда от промени с цел подобряване на производителността.

b. Определяне на максималния капацитет на системата

Цел: Определяне на максималния капацитет на системата при определена конфигурация, когато тя е подложена на натоварване. Така се идентифицират

границите, при които системата може да функционира нормално до определено ниво на натоварване.

Подход: Използват се някои от тестовете за производителност измерващи времето за отговор, които се пускат последователно като всеки път постепенно се увеличава натоварването към системата. Това се прави докато системата започне да проявява дефекти.

Анализ: В резултат на тестовете се идентифицира един от случаите:

i. Системата престава да работи и не може да се възстанови (Този случай обикновено е неприемлив)
ii. Системата започва да се бави много, но след спиране на натоварването тя отново може да продължи да работи

След това се извършват анализи за определяне дали получените резултати удовлетворяват изискванията/очакванията към системата.

c. Изследване за скалируемост(scalability) на системата

Цел: Изследване на поведението на системата при натоварване с използване на различни хардуерни конфигурации и определяне на оптималната от тях.

Подход: Използват се два подхода за тестовете:

i. **Vertical Scalability**: Хардуерната конфигурация се заменя с по-бързи и по-мощни компоненти (увеличава се паметта на сървъра или броя на процесорите). След всяка промяна в хардуерната конфигурация, тестовете се извършват отново, за да се провери съответно дали това е оказало положително влияние върху производителността на системата.
ii. **Horizontal Scalabiltiy**: Използва се техниката на разпределяне на натоварването („load-balancing”). XML Firewall системата се инсталира на две машини (впоследствие може и на повече от две). С помощта на софтуер load balancer се конфигурира клъстер от двете машини. По същество load balancer се състои от виртуален сървър, идентифициран от хост и порт. Този виртуален сървър има връзка с всяка една от физическите машини, на които е инсталирана системата. Клиентите, взаимодействащи със системата изпращат заявките към виртуалния сървър, който на свой ред (по предварително зададен алгоритъм) определя коя от машините да поеме обработката на заявката и съответно я препраща към нея.

Анализ: В резултат на тестовете системата се определя като скалируема ако подобряването на хардуера (при vertical scalability) или добавянето на нова инстанция на системата (при horizontal scalability) води до пропорционално нарастване на производителността на системата.

d. Изследване за надеждност (reliability) на системата

Цел: Изследване на цялостното поведение на системата за дълъг период от време, с натоварване малко над очакваното. Тестовете са проверка за устойчивостта и надеждността на системата.

Подход: Тестовите сценарии трябва да се доближават максимално до действителността и ако е възможно, да изследват всички функционалности на системата. Продължителността им трябва да бъде от порядъка на поне няколко дни, за да се добие представа какво е състоянието на системата, когато тя се използва продължително (например не се спира в продължение на 1 месец). Това на практика е единия вариант на подхода „Soak" описан в частта *„3. Тестове, подходи и практики"*.

От друга страна, добавянето на фактор на случайност като например:

- Заявки с различен размер (1 KB, 2KB, 5KB и т.н.)
- Заявки с валидни и невалидни потребителско име/парола

ще доближи максимално тестовете до реална среда, а също така ще провери дали системата обслужва по същия начин легитимните потребители, докато е подложена на атаки.

Анализ: В резултат на тестовете могат да се идентифицират проблеми, които се проявяват при продължително използване на системата като например:

- Неосвободена памет (останали извън контрола – "memory leaks")
- Неосвободени нишки (останали извън контрола на системата)
- Синхронизационни проблеми
- Проблеми с оптимизирани данни (cache)
- Други необработени ситуации

5.3.3 Избор на инструмент за тестване и писане на скриптове

XML Firewall е система, която приема и връща като отговор SOAP съобщения. От своя страна, SOAP съобщенията могат да се предават по един измежду протоколите HTTP, HTTPS или JMS. За разработката на тестовите скриптове ще се използва инструмента Microsoft ACT, представен подробно в секция "*4.5 Microsoft ACT"*. Характеристиките на ACT, които са особено полезни за тестването на производителността на системата XML Firewall и поради които е избран са следните:

- Както ACT, така и XML Firewall системата поддържат протоколите HTTP и HTTPS.
- Инструментът ACT не е обвързан със свой собствен скриптов език, а използва един от езиците Javascript или Visual Basic. По този начин, тестовите инженери разработващи скриптовете имат възможността да се възползват от всички предимства на тези скриптови езици.
- Централизирано съхранение и управление на резултатите от проведените тестове: ACT пази информация за всеки един пуснат тест. Това е особено полезно, когато трябва да се направят сравнения и

анализи за производителността на системата между всеки две итерации от тестове.

- По време на тестовете, ACT събира информация за метриките за производителността като например използвано процесорно време, налична памет и ги интегрира в тестовите резултати.
- ACT поддържа генериране на графики от тестовите резултати по зададени критерии.

5.3.4 Разработване на тестовите сценарии и скриптове.

5.3.4.1 Определяне на бързодействието на системата

В рамките на този пример ще бъдат разгледани стъпките, които се следват при тестването на една от функционалностите на системата изброени в точка *5.3.2* - верификация на електронно подписана заявка.

Стъпки на изпълнение на задачата:

1. **Осигуряване на необходимите ресурси:**

    a. Инсталация на ACT на определена клиентска машина
    b. Инсталация на XML Firewall на стандартна конфигурация на отделна машина
    c. Инсталиране на Web услуга (избрана на базата на резултатите от изпълнението на първоначалните тестове – виж т.1)
    d. Генериране на ключове за електронен подпис – публичен и частен.
    e. Създаване на тестова конфигурация в системата:
        - Регистриране на избраната Web услуга.
        - Регистриране на публичния ключ (системата може да валидира електронни подписи, подписани само с регистрирани ключове – всички останали ще бъдат отхвърлени).
        - Създаване на правило за групата на регистрираната услуга, което да указва, че се очаква всички входящи заявки да бъдат подписани, да се провери валидността на подписа и ако е валиден – заявката да продължи. Във всички останали случаи, системата ще върне като грешка като отговор.
    f. Конструиране на заявките към регистрираната Web услуга, необходими за тестовете. За целта са необходими заявки със следните размери: 1K, 5K, 10K, 50K, 100K, 500K. След създаването на тези заявки, всяка една от тях трябва да бъде подписана (по-точно нейното тяло – SOAP Body) с частния ключ.
    g. Създаване на скриптове за всяка една заявка. За целта се взима информация от WSDL описанието на регистрираната Web услуга.

Таблица 5-1 показва как ще изглежда кода на скрипта за заявка с размер 1K, написан на езика Visual Basic (на всеки ред има коментар в зелен цвят, обясняващ подробно тестовите инструкции):

```
" Този скрипт изпраща подписана заявка към порта на които XML Firewall
" е настроен да приема входящия трафик.
" Регистрираната Web услуга се състои от една "exo" операция.
" По този начин, изпращайки низа "demo", очаквания резултат е в отговор
" на изпратената заявка да бъде върнат същият низ.

Option Explicit

Dim debug                "определя дали тест да бъде изпълнен в "debug" режим
debug = FALSE
'Test.TraceLevel = -1    "Записва всичката налична информация в ACTTrace.log
                         "когато теста се изпълнява в debug режим .Да се разкоментира когато
                         "има нужда от тази информация.
Dim httpConnection       "Обекта с който се осъществява HTTP връзката към XML Firewall
Dim SOAPRequest          "Съдържа самата заявка
Dim serverHostName       "Името или IP адреса на машината на който работи XML Firewall
Dim serverPortNumber     "Номера на порта на който XML Firewall приема заявки
Dim endpointRelativePath "Относителния път към регистрираната Web услуга
Dim SOAPActionHeader     "Съдържа стойността на SOAPAction, който трябва се включи
                         " в заглавната част на HTTP заявката.

Dim expectedAnswer       "Съдържа низа който се очаква в отговора. Спрямо него се
                         "определя дали заявката е изпълнена успешно

Dim httpRequest          "Обект на HTTP заявката
Dim httpResponse         "Обект на HTTP отговора


"--------------------------------------------------------------------------
" Задаване на стойности на променливите за конкретния тест
" Подписана заявка с размер 1K
"--------------------------------------------------------------------------

serverHostName      =    "xmlfperf"
serverPortNumber    =    8080
endpointRelativePath =   "/xmlf/runtime/echo_signed"
SOAPActionHeader    =    "http://echo"
SOAPRequest         =    "<?xml version="1.0" encoding="UTF-8"?>
                         <SOAP-ENV:Envelope xmlns:SOAP-
                         ENC=http://schemas.xmlsoap.org/soap/encoding/
                         xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
                         xmlns:impl=http://localhost:8080/axis/EchoService.jws-impl
                         xmlns:intf=http://localhost:8080/axis/EchoService.jws
```

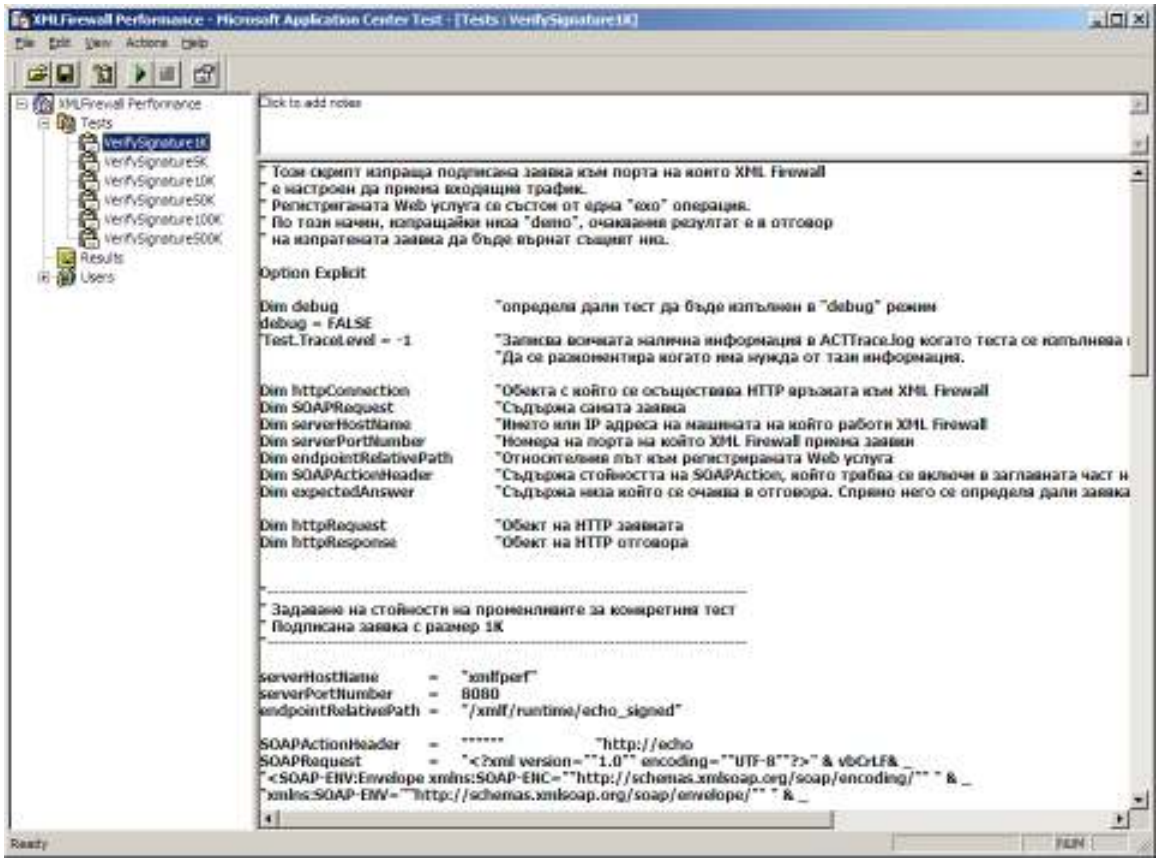
```
                                    xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
                                    xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
                                    xmlns:xsd=http://www.w3.org/2001/XMLSchema
                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                   <SOAP-ENV:Header>
                                   <wsse:Security SOAP-ENV:actor="http://demo" SOAP-ENV:mustUnderstand="1"
                                   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
                  xmlns:wsse=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
                  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
                                  <wsse:BinarySecurityToken
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
wsu:Id="SecurityToken-d7e4b257-6880-057b-4a29-7e485822f4f1">
MIICZzCCAiSgAwIBAAIERXKy0zALBgcqhkjOOAQDBQAwejEfMB0GCSqGSIb3DQEJARYQc
GVyZm9ybWFuY2VAZGVtbzELMAkGA1UEBhMCQkcxDjAMBgNVBAgTBVNvZmlhMQ0wCwYDVQ
QHEwREZW1vMQ0wCwYDVQQKEwREZW1vMQ0wCwYDVQQLEwREZW1vMQ0wCwYDVQQDEwREZW1
vMB4XDTA2MTIwMzExMTk0N1oXDTA3MTIwMzExMTk0N1owejEfMB0GCSqGSIb3DQEJARYQ
cGVyZm9ybWFuY2VAZGVtbzELMAkGA1UEBhMCQkcxDjAMBgNVBAgTBVNvZmlhMQ0wCwYDV
QQHEwREZW1vMQ0wCwYDVQQKEwREZW1vMQ0wCwYDVQQLEwREZW1vMQ0wCwYDVQQDEwREZW
1vMIHxMIGoBgcqhkjOOAQBMIGcAkEA/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864
WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFwIVAJYu3cw2nLqOuyYO5rahJ
tk0bjjFAkBnhHGyepz0TukaScUUfbGpqvJE8FpDTWSGkx0tFCcbnjUDC3H9c9oXkGmzLik
1Yw4cIGI1TQ2iCmxBblC+eUykA0QAAkEAx383whuwHdZc/Mp+e3nNas/wVYqSXFFmlxfSUP
1/iqHTfq6+RXZd85HLnISH0SxoS2pUURWcCniV4TBp+gGKczALBgcqhkjOOAQDBQADMAAwL
QIUdI5sJSvqvWk6dK73qMShnxfffKICFQCJoWovXIYVpNf2Yp1nWqoGBCuEgA==
</wsse:BinarySecurityToken>
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
<dsig:SignedInfo>
<dsig:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-  c14n#"/>
<dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
<dsig:Reference URI="#Id-2d5313db-87bb-c227-7dc8-cfef5f797d0c">
<dsig:Transforms>
<dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</dsig:Transforms>
<dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<dsig:DigestValue>PcaFHlU/G4DCjrWUteuy0NPY5iU=</dsig:DigestValue>
</dsig:Reference>
</dsig:SignedInfo>
<dsig:SignatureValue>dVRSipsSVxghex5ciEm9zhdeBs1xYF5twA6z++KVYFTze2TWYROjSQ==</dsig:SignatureValue>
<dsig:KeyInfo>
<SecurityTokenReference xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<wsse:Reference URI="#SecurityToken-d7e4b257-6880-057b-4a29-7e485822f4f1"/>
</SecurityTokenReference>
</dsig:KeyInfo>
</dsig:Signature>
</wsse:Security>
</SOAP-ENV:Header>
   <SOAP-ENV:Body wsu:Id="Id-2d5313db-87bb-c227-7dc8-cfef5f797d0c" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
   <m:echo SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:m="http://localhost:8080/axis/EchoService.jws">
   <in0 xsi:type="xsd:string">demo</in0>
</m:echo>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>"
```

```
expectedAnswer                  =         "demo"

"-------------------------------------------------------------------------------
" Помощни функции
"-------------------------------------------------------------------------------

"Намира дали даден низ се съдържа във върнатия отговор
Function IsStringInResponseBody(response, string)

        If InStr(response.Body, string) Then
                IsStringInResponseBody = True
        Else
                IsStringInResponseBody = False
        End if

End Function

" Премахва от HTTP заявката, зададена по име, заглавна част и след това я добавя отново
с нова стойност, подадена като параметър

Sub ReplaceHeader(request, httpHeaderName, httpHeaderNewValue)
        Call request.Headers.Remove(httpHeaderName)
        Call request.Headers.Add(httpHeaderName, httpHeaderNewValue)
End Sub


" Записва в ACTTrace.log зададения низ, само ако тестът е пуснат в debug режим

Sub Log(message)
        If debug Then
        Test.Trace(message)
        End if
End Sub

"-------------------------------------------------------------------------------
" Създаване на необходимите обекти и изпълняване на теста
"-------------------------------------------------------------------------------

"Създаване на обекта който ще изпрати HTTP заявката
Set httpConnection = Test.CreateConnection(serverHostName, serverPortNumber)

If (httpConnection is Nothing) Then
   Call Log("Error creating the httpConnection")       "Неуспешно създаване

Else
        Set httpRequest = Test.CreateRequest          "Създаване на обекта на HTTP заявката.
```

```
        httpRequest.CodePage = 65001                "UTF-8 encoding
        httpRequest.EncodeBody = False              "Тялото на заявката не е кодирано
        httpRequest.Verb = "POST"                   "POST метод за изпращане

        "Добавяне на SOAPAction в заглавната част на заявката
        httpRequest.Headers.Add "SOAPAction", SOAPActionHeader

        "Задаване на SOAP съобщението като тяло
        httpRequest.Body = SOAPRequest

        httpRequest.PATH = endpointRelativePath   "Указване на пътя на заявката

        "Затваряне на връзката след получаване на отговора
        Call ReplaceHeader(httpRequest, "Connection", "Close")

       "Изпращане на заявката и получаване на отговор
        Set httpResponse = httpConnection.Send(httpRequest)


        "Отговора на заявката не е получен успешно
        If (httpResponse Is Nothing) Then
                Call Log("Error receiving the response")
        Else

                "Получен е отговор
                If IsStringInResponseBody(httpResponse, expectedAnswer) Then

                        "Отговорът съдържа  очакваната стойност
                        Call Log("SUCCESS: " & httpResponse.Body)


                Else
                        "Отговорът не съдържа  очакваната стойност. Има грешка, която
                        " трябва да бъде изследвана
                        Call Log("ERROR: " & httpResponse.Body)

                End If
        End If
End If
```

Таблица 5-1: Кодът на тестовия скрипт

В средата на ACT, този скрипт изглежда по следният начин (фигура 5-3):



Фигура 5-3: Кодът на тестовия скрипт

h. Определяне на комбинациите от тестове, които трябва да бъдат изпълнени (Таблица 5-2):

Големина на заявката → Конкурентни потребители ↓	**1K**	**5K**	**10K**	**50K**	**100K**	**500K**
1	V	V	V	V	V	V
4	V	V	V	V	V	V
16	V	V	V	V	V	V
32	V	V	V	V	V	V
64	V	V	V	V	V	X
128	V	V	V	V	X	X

Таблица 5-2: Тестовите комбинации, които трябва да се изпълнят (V- теста ще се изпълни; X- няма да се изпълни)

Целта е да се събере информация за голяма комбинация от заявки и натоварвания и да се анализира как работи продукта в повечето случаи от реалния свят.

Допълнителни тестове, които трябва да бъдат направени на базата на тестовете за бързодействие:

Тест: Намиране на оптимални настройки за системата.

Цел: намиране на ефекта върху бързодействието при промяна на настройките на XML Firewall – броя на нишките обработващи заявките, използване на хардуерни криптографски карти.

Резултат: Документ за оптимизационни настройки за продукта.

5.3.4.2 Изследване за скалируемост (scalability) на системата

За определяне на скалируемостта на продукта се използват сценариите и тестовете от предишната точка. Единственото, което се променя е броя на инсталираните XML Firewall (хоризонтална) или различните хардуерни конфигурации (вертикална).

5.3.4.3 Изследване за надеждност (reliability) на системата

Стремейки се да се доближи максимално до средата на реална работа на системата, тестовете за надеждност трябва да се базират до известна степен до случайността, като за база за сравнение се използват данни от реално работещи подобни системи и мрежи от Web услуги.

Определянето на случайния фактор при тестването на надеждността на XML Firewall се определя на три етапа, като се базира на данните в всяка една от таблиците 5-3, 5-4, 5-5 (Избраните стойности се базират на статистически данни за големината, типа и честотата на заявките в реално работещите мрежи от Web услуги):

Тип заявка	**Вероятност %**
Обикновени заявки	**40**
- валидни	*85*
- невалидни	*15*
С потребителско име и парола	**35**
- валидни	*75*
- невалидни	*25*
Подписани и шифрирани	**25**
- валидни	*80*
- невалидни	*20*

Таблица 5-3: Разпределение на заявките

Размер на заявката	**Вероятност %**

1K	30
2K	25
5K	15
10K	10
50K	10
100K	8
500K	2

Таблица 5-4: Разпределение на размера на избраната заявка

Вид протокол	**Вероятност %**
HTTP	80
HTTPS	20

Таблица 5-5: Разпределение на протоколите

Създаването на скрипт за този вид тест е аналогично на създаването на теста, даден за пример в частта за определяне на бързодействието. За симулиране на случайността в скрипта се използват конструкциите показани във Таблица 5-6:

```
"Генериране на случайно число в интервала от 0 до 1

Function GetRandom

        Dim Percent
        Call Randomize()
        Percent = Rnd(1)
        GetRandom = Percent

End Function


Dim requestTypeProbability          "Определя конкретен тип заявка и съответните
                                    "и параметри

Dim validProbability                "Определя от кой тип да бъде заявката
                                    "валидна/невалидна


"Определяне на заявката, която ще бъде изпратена, както и всичката нужна
информация за нея

requestTypeProbability = GetRandom

If requestTypeProbability <0.40 Then        "Обикновени заявки
```

```
        validProbability = GetRandom

        If validProbability <0.85 Then          "Валидна заявка
                sendValidSimpleRequest()

        Else                                    "Невалидна заявка
                sendInvalidSimpleRequest()

ElseIf Percent <0.75 Then                       "С потребителско име и парола

        validProbability = GetRandom

        If validProbability <0.75 Then          "Валиден потребител

             sendValidUsernamePasswordRequest()
        Else                                                "Невалиден потребител
                sendInvalidUsernamePasswordRequest()

Else                                                        "Подписани и шифрирани
        validProbability = GetRandom

        If validProbability <0.8 Then                       "Валидна заявка

                sendValidSingedEncyptedRequest()

        Else                                                "Невалидна заявка
                sendInvalidSingedEncyptedRequest()
End If


"Във всяка една от sendXXXRequest функция, аналогично се определя размера на
"заявката от съответния тип и протокола, по който ще бъде изпратена:

Dim requestSizeProbability          "Определя конкретен SOAPRequest(със
                                    " избрания размер) от съответния тип

Dim protocolProbability             "Определя конкретен протокол HTTP/HTTPS
```

Таблица 5-6: Симулиране на случайност в тестовия скрипт

5.4 Обобщение

XML Firewall е продукт, който централизира и защитава трафика към Web услуги в големи корпоративни мрежи. Той е проектиран да може да работи стабилно и безотказно при големи натоварвания. От неговата производителност зависи качеството на услугите, предлагани от корпорациите, които ще го интегрират в техните системи. Имайки тези изисквания и цели пред себе си, екипът от тестови инженери има задачата да планира и разработи такива тестове, които да гарантират, че продукта ще удовлетвори очакванията на неговите клиенти. За целта в плана за тестване на производителността се включва изследване на следните характеристики на системата:

- Бързодействие
- Максимален капацитет
- Скалируемост
- Надеждност

За реализирането на тези тестове е избран инструмента Microsoft ACT, в чиято среда се разработват конкретни тестови скриптове. За да се симулира среда максимално близка до реалната се избират различни тестови комбинации, а също така се добавя фактор на случайност. За всеки тест се описва каква е неговата цел, подхода за постигането и, както и какъв анализ трябва да бъде направен върху резултатите.

6 Изпълнение на тестовете. Отчитане на резултатите. Анализ.

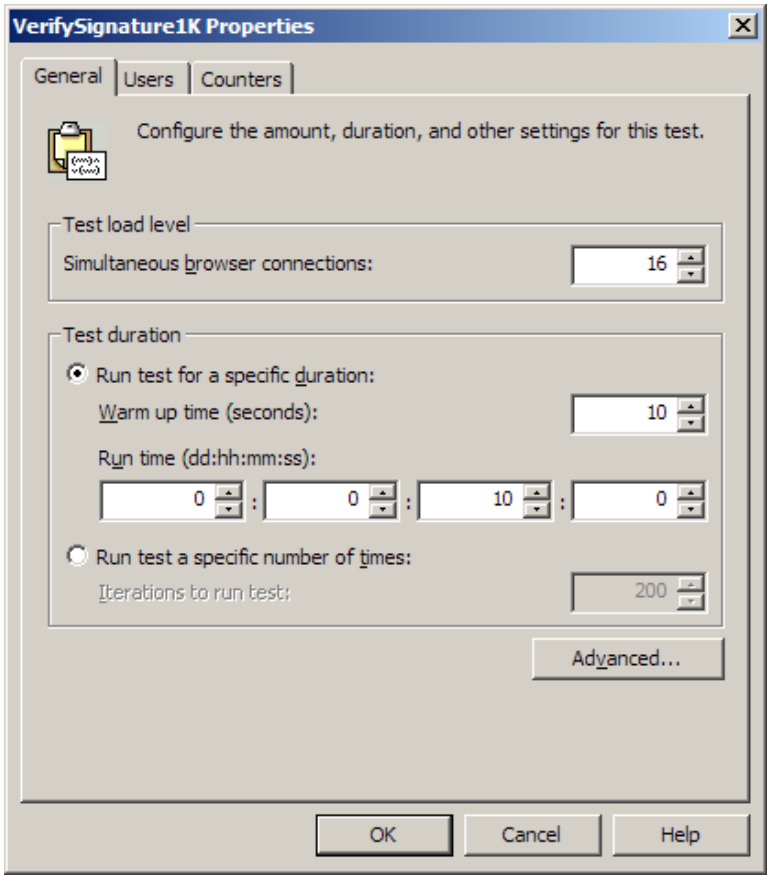
6.1 Изпълнение на тестовете.

Тестът, който ще бъде демонстриран се базира на стратегията и скрипта (*Таблица 5-1*) създаден в предишната глава. За постигане на целите и исканите резултати, тестът се стартира със следните стъпки:

a. Настройки на ACT

Тестовият скрипт ще се изпълнява със следните параметри (Фигура 6-1):

- Продължителност - 10 минути
- Брой конкурентни потребители - 16
- Време на изчакване (Warm-up Time) – 10 секунди

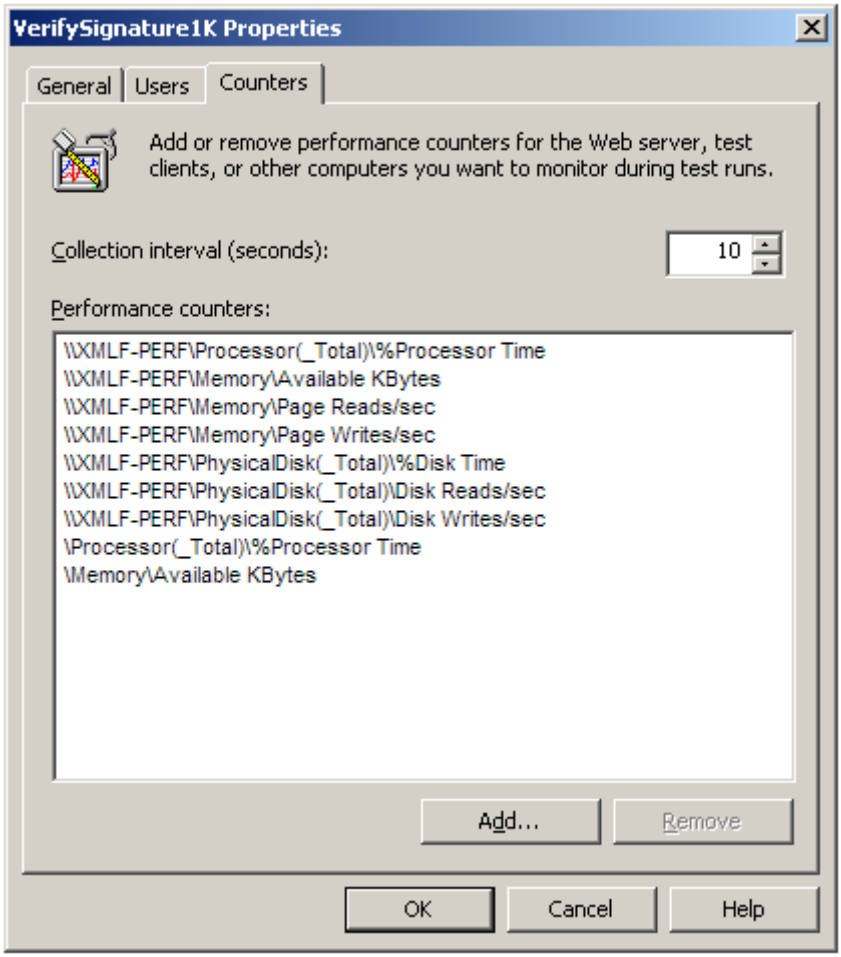


Фигура 6-1: Конфигурация на теста

- Метрики за производителността (Performance Counters) за клиентската машина - тази, на която е инсталиран ACT (Фигура 6-2):

- Използване на процесора (*\Processor(_Total)\%Processor Time*)
- Налична памет (*\Memory\Available KBytes*)

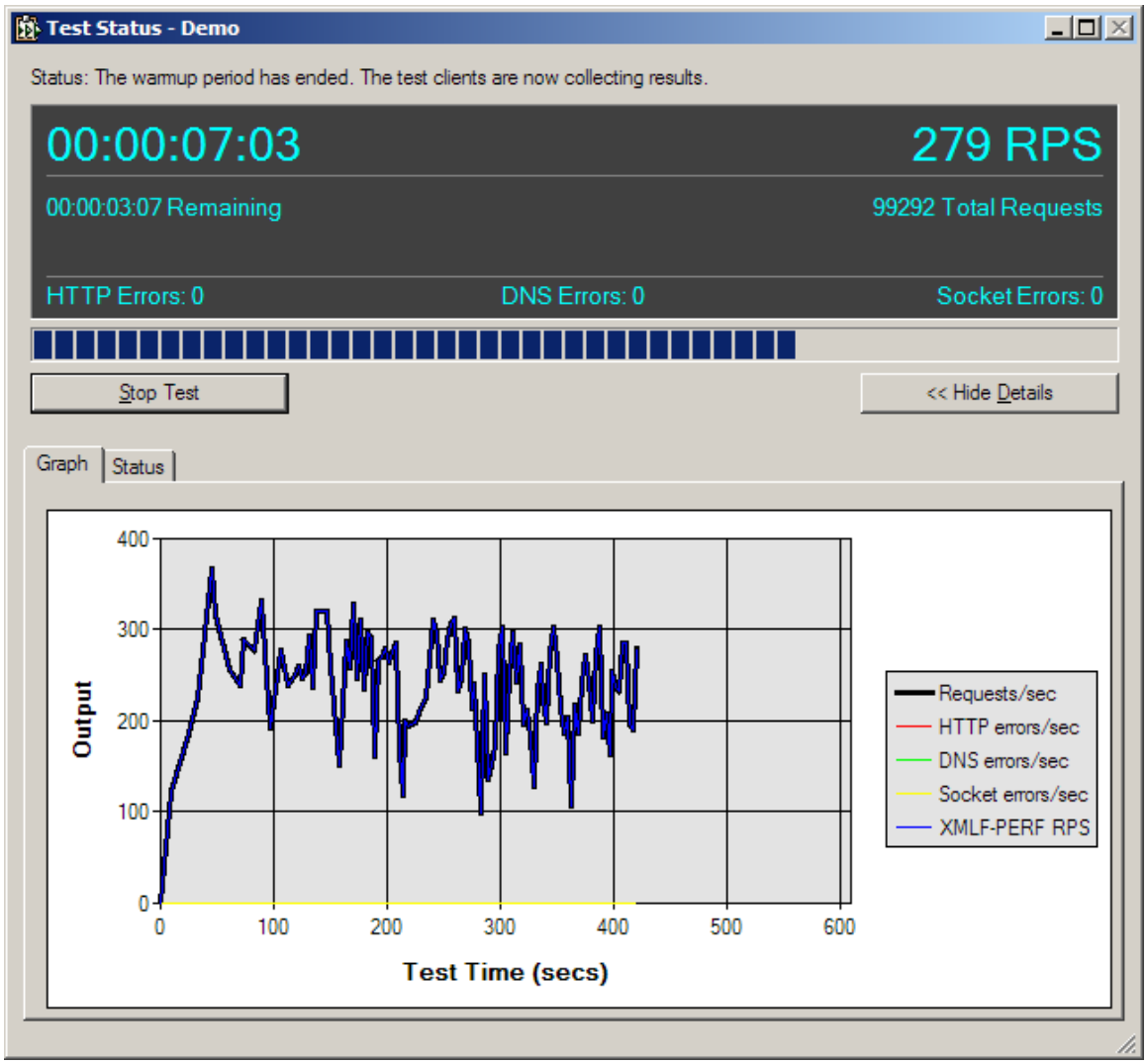
- Метрики за производителността (Performance Counters) за сървърната машина (тази, на която е инсталиран XML Firewall). Машината се достъпва по име XMLF-PERF (Фигура 6-2):
  - Използване на процесора *(\\XMLF-PERF\Processor(_Total)\ %Processor Time*)
  - Налична памет *(\\XMLF-PERF\Memory\Available KBytes*)
  - Четене от паметта *(\\XMLF-PERF\Memory\Page Reads/sec*)
  - Писане от паметта *(\\XMLF-PERF\Memory\Page Writes/sec*)
  - Време за операции с твърдия диск *(\\XMLF-PERF\PhysicalDisk(_Total)\%Disk Time*)
  - Четене от твърдия диск *(\\XMLF-PERF\PhysicalDisk(_Total)\Disk Reads/sec*)
  - Писане от твърдия диск *(\\XMLF-PERF\PhysicalDisk(_Total)\Disk Writes/sec*)



Фигура 6-2: Добавяне на метрики за производителността

b. Стартиране на скрипта

Изпълнението на скрипта започва след натискането на бутона **Start Test**. В резултат, както е илюстрирано на фигура 6-3, се появява прозорецът **Test Status**, който започва да показва временните резултати:



Фигура 6-3: Временни резултати от изпълнението на теста

6.2 Отчитане на резултатите.

6.2.1 Резултати от изпълнението на теста за определяне на бързодействието на XML Firewall.

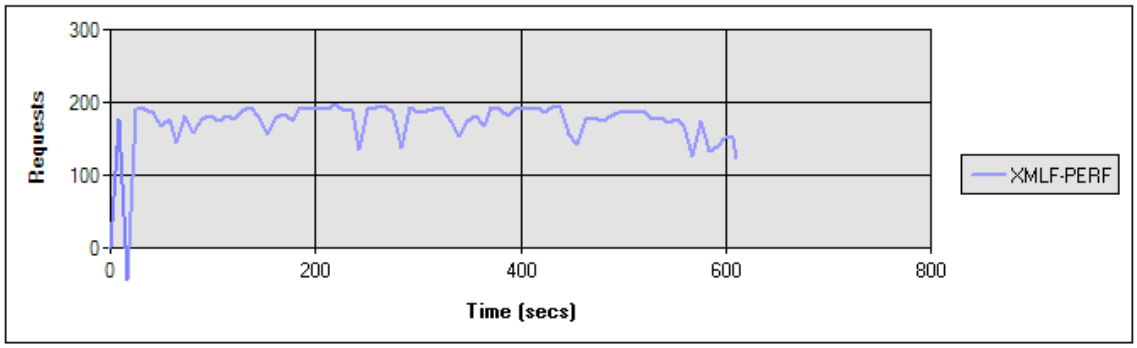
След като изпълнението на теста приключи, всички статистики за съответния тест могат да бъдат намерени в частта Резултати (Results) в проекта от Microsoft ACT. Таблица *6-1* съдържа резултатите от първото изпълнение на скрипта за XML Firewall:

Application Center Test

Overview: Summary

Test Name:	XMLFirewall Performance: VerifySignature1K
Test Run Name:	report-VerifySignature1K-Dec 06, 2006 09-16-19
Test Started:	12/6/2006 9:06:07 AM
Test Duration:	00:00:10:00
Test Iterations:	106,270
Test Notes:	-

Test Run Graph



Properties

Test type:	Dynamic
Simultaneous browser connections:	16
Warm up time (secs):	10
Test duration:	00:00:10:00
Test iterations:	106,270
Detailed test results generated:	Yes

Summary

Total number of requests:	106,270
Total number of connections:	106,270

Average requests per second:	177.12
Average time to first byte (msecs):	2.93
Average time to last byte (msecs):	2.97
Average time to last byte per iteration (msecs):	2.97
Number of unique requests made in test:	1
Number of unique response codes:	1

Errors Counts

HTTP:	0
DNS:	0
Socket:	0

Additional Network Statistics

Average bandwidth (bytes/sec):	579,702.85
Number of bytes sent (bytes):	186,078,770
Number of bytes received (bytes):	161,742,940
Average rate of sent bytes (bytes/sec):	310,131.28
Average rate of received bytes (bytes/sec):	269,571.57
Number of connection errors:	0
Number of send errors:	0
Number of receive errors:	0
Number of timeout errors:	0

Response Codes

Response Code: 200 - The request completed successfully.

Count:	106,270
Percent (%):	100.00

Requests

	Requests	**Content Length (bytes)**		**Time To First Byte (msecs)**		**Time To Last Byte (msecs)**	
	Total	**Avg**	**Std Dev**	**Avg**	**Std Dev**	**Avg**	**Std Dev**
POST xmlf-perf/xmlf/runtime/echo_signed							
	106,270	1875.00	0.00	2.93	1.34	2.97	1.37

Overview: Performance Counters

\\XML-PERF\Processor\% Processor Time_Total

Minimum:	81.63
Maximum:	100.00
Average:	97.02
25th Percentile	98.44
50th Percentile	99.02
75th Percentile	99.50

\\XML-PERF \Memory\Available Kbytes

Minimum:	966,820.00
Maximum:	989,272.00
Average:	976,105.10
25th Percentile	971,059.80
50th Percentile	976,095.50
75th Percentile	978,103.50

\\XML-PERF \Memory\Page Reads/sec

Minimum:	70.90
Maximum:	179.13
Average:	123.48
25th Percentile	106.82
50th Percentile	121.89
75th Percentile	139.10

\\XML-PERF \Memory\Page Writes/sec

Minimum:	0.00
Maximum:	1.68
Average:	0.60
25th Percentile	0.20
50th Percentile	0.49
75th Percentile	1.02

\\XML-PERF \PhysicalDisk\% Disk Time_Total

Minimum:	70.00
Maximum:	100.00
Average:	89.00

25th Percentile	70.00
50th Percentile	82.50
75th Percentile	87.00
\\ XML-PERF \PhysicalDisk\Disk Reads/sec_Total	
Minimum:	71.70
Maximum:	177.84
Average:	123.60
25th Percentile	107.32
50th Percentile	121.74
75th Percentile	138.80
\\ XML-PERF \PhysicalDisk\Disk Writes/sec_Total	
Minimum:	11.00
Maximum:	23.11
Average:	14.17
25th Percentile	4.92
50th Percentile	5.33
75th Percentile	12.53
\Processor\% Processor Time_Total	
Minimum:	58.02
Maximum:	92.98
Average:	73.31
25th Percentile	71.88
50th Percentile	71.93
75th Percentile	72.08
\Memory\Available Kbytes	
Minimum:	483,290.00
Maximum:	503,616.00
Average:	497,613.00
25th Percentile	486,940.00
50th Percentile	485,040.00
75th Percentile	498,743.00

Таблица 6-1: Резултати от ValidateSignature1K с 16 конкурентни потребители

Освен с 16 конкурентни потребители, тестът е пуснат също и със 1, 4, 32, 64 и 128 потребители. Резултатите са обобщени в таблица *6-2*:

Threads	Duration	Total requests	req/sec	Errors	Notes
1	10	68013	113.355	0	Disk usage at product machine is very high during the tests
4	10	85016	141.693	0	
16	10	106270	177.117	0	
32	10	122211	203.684	0	
64	10	128321	213.868	0	
128	10	141153	235.255	0	

Таблица 6-2: Обобщение на теста ValidateSignature1K

6.2.2 Резултати от изпълнението на теста за определяне на надеждността на XML Firewall.

За този вид тест, най-важните статистики са тези, които показват разпределението на заявките (Таблица 6-3):

	Requests	Content Length (bytes)		Time To First Byte (msecs)		Time To Last Byte (msecs)	
	Total	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
POST xmlf-perf/xmlf/runtime/schemavalidate							
	25779562	52643.21	449227.16	12.22	4.67	19.87	93.17
POST xmlf-perf/xmlf/runtime/signencrypt							
	87988043	70758.70	429242.30	29.82	13.64	32.06	197.64
POST xmlf-perf/xmlf/runtime/usernamepass							
	145870255	65829.15	434161.85	17.08	3.42	23.87	83.88
POST xmlf-perf/xmlf/runtime/transfrorm							
	30587740	49185.33	445815.67	14.68	5.90	21.91	75.94
POST xmlf-perf/xmlf/runtime/simple							
	39870123	51039.49	449961.51	9.54	4.88	19.98	45.64
POST xmlf-perf/xmlf/runtime/audit							
	28566455	55635.98	444465.02	10.68	5.07	20.58	59.40

Таблица 6-3: Разпределение на случайните заявки

ACT предоставя и детайлни данни за всеки един вид заявка от горната таблица. На базата на всички тези данни, може да се определи поведението на отделните компоненти на системата поставена в режим на постоянно натоварване от най-разнообразни заявки – да се изследва дали всеки от тях работи в рамките на очакваното. Създадения тест се пуска с продължителност от 24 часа до няколко дни, в зависимост от времето за тестове с което се

разполага. По време на изпълнение на теста, трябва да се следят основните показатели на системата – процесор, памет, процесите на XML Firewall, както и дисковите операции.

6.3 Анализ на резултатите.

- Проверка за валидност: Тестът се счита за валиден, тъй като не са отчетени HTTP, DNS или Socket грешки

- Използване на системните ресурси от клиентската машина: Процесорът се използва средно на 73.71%, което е в допустимите граници (за да са правилни резултатите, се препоръчва да не надвишава 75%)

- Резултатите за средния брой на обработени заявки в секунда (Average requests per second) от системата XML Firewall са незадоволителни и показват, че системата няма достатъчен капацитет за обработка на подписани заявки за работа в реални условия. При направените проучвания в реално работещи мрежи, за този тип трафик : 16 потребителски сесии, в пиковите моменти на работа, системата се очаква да обработи над 400 заявки за секунда. Но резултатите от теста показват, че тя може да обработи едва около 177 заявки.

- Резултатите за стандартното отклонение за времето за отговор на системата (Time to last byte, Std. Dev) показват, че има заявки за които системата се забавя повече от 40% от средното време за отговор (средно 2.97 мс, отклонение 1.37мс - ~46%). Това не отговаря на повечето стандарти за качеството на предлаганата услуга и резултатите трябва да се подобрят.

- Изследването на метриките за производителността (Performance Counters) показва, че само резултатите за два от показателите са извън нормата – операциите за четене от дисковото устройство (Disk Reads) – средно 123.60 операции за секунда и средното време за използване на процесора (Processor Time) - 97.02%. Най-вероятно, причините за завишените стойности на тези показатели, са причини и за недобрите резултати на продукта описани в горните два проблема.

- Кодът на отговорите на всички заявки е 200, което показва, че няма нито една грешка в системата в това отношение. Това потвърждава стабилността на XML Firewall.

  За по-добра визия върху извършваните тестове, за всеки тест за бързодействие се попълва шаблон, който е специално създаден за тази цел (Таблица 6-4). По този начин може бързо и лесно да се види обобщена информация за резултатите и анализа на извършените

тестове, а също така и да се проследи поправянето на намерените проблеми, записани като дефекти и кой е отговорният тестов инженер *(Ellen Friedman, 2005)*.

Тест		Подписана заявка 1К	
Дата на изпълнение		06.12.2006 г.	
Изпълнил теста		Биляна Петрова	
Валидност на теста		Да	
Машина на ACT		Perf	
Машина на XML Firewall		Xmlf-perf	
Изследван параметър	Приемлив резултат	Дефекти	Коментар
Оперативна памет	Да	-	-
Процесорно време	Не	4353	Завишено: Над 90%
Дискови операции	Не	4354	Завишено четене
Заявки за секунда	Не	4350	Под очакваното
Грешки на системата	Да	-	-
Отклонение на отделните заявки в рамките на 40% от средното	Не	4350	>46%

Таблица 6-4: Попълнен шаблон за изпълнен тест за бързодействие

Насоки за подобрение:

За намерените проблеми се създават дефекти (bugs), в които се описва детайлно тестовата конфигурация, получените резултати и направените анализи от изпълнението на тестовете.

След детайлно разглеждане и анализ на резултатите от ACT, програмистите са обърнали внимание на бележката за това, че по време на теста, продуктът е използвал диска и процесора повече от обичайно. Така, тази отчетена статистика по време на теста помага за бързото локализиране на проблема – за всяка заявка системата е прочитала съдържанието на публичния ключ от файл от диска. Тя е оптимизирана да запази най-често използваните ключове в паметта, като по този начин се намаля драстично броя на бавните операции за четене от диска.

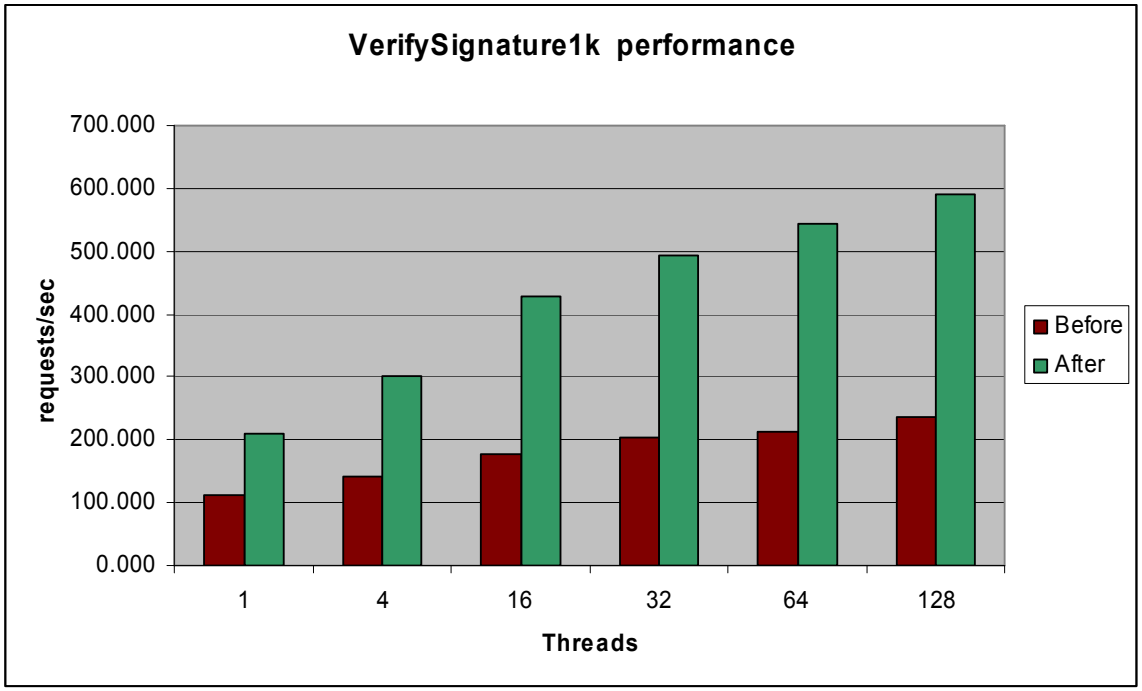
След като проблема е разрешен от програмистите, същият тест е изпълнен отново на същите машини (т.е. в същата тестова среда), за да се установи дали проблема наистина е решен и какви са новите резултати. Резултатите са обобщени в таблица 6-5:

Threads	Duration	Total requests	req/sec	Errors	Notes
1	10	126065	210.108	0	
4	10	180093	300.154	0	
16	10	257275	428.792	0	
32	10	295866	493.110	0	
64	10	325453	542.421	0	

128	10	354744	591.239	0

Таблица 6-5: Обобщени резултати след направените промени в системата

Решен е и проблема в стандартното отклонение, което е вече в рамките на 25% от средното време за отговор на системата. Сравнението на резултатите от двете итерации на теста е илюстрирано на фигура 6-4. След поправянето на проблема, бързодействието на системата се е подобрило повече от 2 пъти за по-големите натоварвания на системата (16, 32, 64 и 128 потребителски сесии) и около 2 пъти при по-слабите (1 и 4 потребителски сесии).



Фигура 6-4: Сравнение на резултатите от двете итерации

7 Заключение

Дори и най-доброто като функционалност Web приложение няма да успее да впечатли никой от своите потребители, ако страниците/компонентите му се зареждат бавно и те са принудени да чакат. Още повече, ако системата е комерсиално приложение, обслужващо голям брой потребители, това може да доведе до силно недоволство на клиентите и големи финансови загуби.

Днешните потребители на софтуерните системи, изискват не само те да извършват това, за което са предназначени, но и да са достатъчно бързи и конкурентно способни на пазара. Ето защо във все повече софтуерни проекти, тестването за производителност е неделима част от софтуерния процес. Все повече софтуерни организации осъзнават тази нужда и инвестират в инструменти за тестване на производителност. Дефинирането на изисквания по отношение на производителността трябва да стане още по време на фазата на проектиране на системата. Изборът на правилна стратегия и потребителски сценарии също е от съществена важност и само по себе е предизвикателство – тестовете трябва да имитират доколкото е възможно реалната среда, да се стремят да достигнат очакваното натоварване, да включват фактор на случайност. Анализът на рисковете, както и включването на всички участници в проекта са ключови елементи в разрешаването на проблеми по производителността. Освен установяването на възможностите на системата по отношение на производителността, процесът на тестването за производителност стимулира нейното развитие и усъвършенстване.

Въпреки, че тестовете и инструментите за производителност са различни и се обуславят от спецификата на конкретната система, общо известните подходи, практики и методи за тестване, които бяха представени в настоящата дипломна работа са доказани и дават по-голям шанс на софтуерните проекти да завършват успешно. Изборът на подход или конкретен вид тест силно зависи от бизнес нуждите, естеството на приложението, процеса на разработка, който се следва, както и от наличните времеви, човешки и парични ресурси. От разгледаните инструменти за тестване може да се заключи, че за големи Web системи е изключително важно те да притежават следните характеристики:

- Лесно и бързо създаване на тестови скриптове/сценарии, както и възможност за тяхната модификация
- Точни данни, анализи и поддръжка на графики
- Възможност за настройка на тестовата среда според особеностите на конкретната система

Познаването на характеристиките на повечето популярни инструменти на пазара би помогонало за по-бърз и правилен избор на инструмент за всеки един конкретен случай.

Освен всичко друго, чрез извършването на тестове за производителност могат да бъдат открити проблеми в дизайна на приложението и той да бъде променен още в началните фази в процеса на разработка. Това от своя страна би спестило много средства и време.

От разгледания в дипломната работа практически пример може да се заключи, че самите тестове трябва да могат да се възпроизвеждат, да се извършват в

контролирана среда и да са много добре документирани. Това се налага, тъй като в практиката обикновено се извършват няколко итерации на изпълненние на тестовете, докато се постигнат задоволителни и оптимални резултати. Характерните за тестването на Web системи метрики -време за отговор (response time), капацитет (throughput) и използване на системните ресурси(utilization), дават удобен начин за количественото представяне и измерване на производителността. Представянето на тестовите резултати във вид на графики улеснява анализа и взимането на бързи и адектватни решения от страна на ръководителите на проекта.

Тестването за производителност е необходима задача за определяне на възможностите и ограниченията на изследваното приложение/система. То е един вид „гаранция", че система използва правилните софтуерни, хардуерни и мрежови ресурси, за да задоволи очакванията на своите клиенти. На базата на получените резултати и информация от извършените тестове за производителност, могат да се предвидят действия и очаквания за това как производителността на системата би могла да се подобри в бъдеще ако има такива изисквания или нужди.

Тестването на производителността е неделима и важна част в процеса на разработване на всеки един успешен и качествен Web продукт.

8 Литература

[1] Статия в Интернет: Adam Kolawa (2001), Performance Testing: http://www.wrox.com/WileyCDA/Section/id-291625.html

[2] Статия в Интернет: Leslie Segal, Key Considerations in Performance Testing: http://www.testwareinc.com/resources/articles/Key_Considerations_in_Performance_Testing.pdf

[3] Книга: Raj Jain (2001), "The Art of Computer Systems Performance Analysis", John Wiley & Sons

[4] Статия в Интернет: Kumaran Systems (2006), Performance Testing: http://www.kumaran.com/PerformanceTesting.html

[5] Статия в Интернет: Adobe Systems Incorporated (2006), Capacity planning: http://www.adobe.com/products/flex/whitepapers/pdfs/flex2wp_fdscapacityplanning.pdf

[6] Статия в Интернет: RPM Solutions Pty (2004), Soak Tests: http://loadtest.com.au/types_of_tests/soak_tests.htm

[7] Статия в Интернет: Matt Maccaux (2005), Approaches to Performance Testing: http://dev2dev.bea.com/pub/a/2005/09/performance_testing.html

[8] Статия в Списание: Adam Neat (03, 2004), "Software Test&Performance" March 2004, "Definining and Building High-Performance Software"

[9] Статия в Интернет: Arash Barirani and Jeffrey Blake (2004), Analyzing a Web-Based Performance Problem:
http://www.artima.com/articles/webperf.html

[10] Статия в Интернет: A. Zisman, R. Summers, S. Katz, F. Servan, and J. Chelsom (05, 2002). "Information Bus: A Web Services Application", 11th International World Wide Web Conference
http://www2002.org/CDROM/alternate/XS1/

[11] Статия в Интернет: Ellen Friedman (11, 2005), "Tales from the Lab: Best Practices in Application Performance Testing":
http://www.cmg.org/measureit/issues/mit27/m_27_10.html