Prestandatestning av webbapplikationers serversida

Jonas Granvik

Kandidatarbete
Handledare: Ivan Porres
Laboratoriet för Programvaruproduktion
Tekniska Fakulteten
Åbo Akademi

Sammandrag

Write your abstract here, or move this section to a separate file named “abstract.tex”, and uncomment the include line below.

Innehåll

Figurer

Förkortningar

Här listas de förkortningar och begrepp som är mest centrala för innehållet i avhandlingen.

Ajax	Asynchronous JavaScript and XML
CSRF eller XSRF	Cross Site Request Forgery, ett sätt att lura servern att tro att anrop kommer från en legitim klient.
DOM	Document Object Model, den trädmodell av en webbsidas innehåll som en webbläsare har.
GWT	Google Web Toolkit
XHR	XMLHttpRequest, en method i moderna webbläsares API som möjliggör synkrona eller asynkrona anrop till servern.
JSON	

Kapitel 1

Inledning

All nödvändig bakgrundsinformation tas upp i detta kapitel och även motivation till ämnesval och ingreppsvinkel förklaras.

1.1 Syfte och tes

Syftet med detta arbete är att undersöka närmare hur man utför prestandatester på webbapplikationer. Den går dels in på hur man traditionellt brukar testa prestandan och hur man idag, med lite modernare teknologi, kan utföra tester och kringgå vissa nya tekniska utmaningar.

Avhandlingen försöker påvisa att det är fullt möjligt att utföra prestandatestning även på moderna webbapplikationer så länge man beaktar vissa saker som bl.a. Ajax-teknologins användning fört med sig.

1.2 Begränsningar

Jag har valt att inte behandla prestandatestningen av klientsidan alls, utan fokuserar helt på serversidan. Detta för att klientsidan hos webbapplikationer snabbt blir så mångfacetterad och varierande att det är svårt att vid sidan om serversidan fokusera även på klienten. Vaadin används som exempelramverk för moderna webbapplikationer för att det är ett relativt nytt ramverk som använder Ajax håller på och vinner mark hos användare allt mera, ett exempel på detta är att Vaadin inkluderas i Gartners rapport ”MarketScope for Ajax Technologies and RIA Platforms” för 2011.

TODO: citera Gartner artikeln rätt

1.3 Uppläggning

Avhandlingens första kapitel inleder läsaren till frågeställningen och förutstättningarna för avhandlingen. Även nödvändiga bakgrundsinformationen om webbapplikationer, prestandatestning och Vaadin ramverket presenteras. Andra kapitlet innehåller mer detaljerad information och jämförelse mellan prestandatestning på traditionella och moderna webbapplikationer.

1.4 Webbapplikationer

Webbapplikationer är applikationer som är tillgängliga över nätverk. De tjänar ofta ett specifikt syfte för användaren genom att nära agera tillsammans med ett gränssnitt av något slag. Användargränssnittet kan bestå av en webbsida man öppnar i en webbläsare eller så kan de använda någon tilläggsmodul i webbläsaren för att rendera själva utseendet korrekt.

Oftast använder man sig av HTTP protokollet för att sända och ta emot information mellan klienten och servern. Själva protokollet är tillståndslöst (eng. stateless), men genom att från klientsidan skicka med en sessions-id för identifikation, kan man upprätthålla ett tillstånd mellan parterna så att de kommer ihåg varandra mellan anropen. [3]

Vanligen körs webbapplikationen i en applikationscontainer eller tolkas av en scriptmotor. Vanliga exempel på applikationscontainers är Java-specifikationens servlet container och ASP.Net som körs på Microsoft IIS server. Skriptmotorer används bl.a. av PHP och Python.

FIXME: ref till definitioner

1.4.1 Vaadin

Vaadin kommer att användas som exempelramverk för prestandatesterna. Vaadin är ett Java-baserat ramverk för att skapa dynamiska internetapplikationer som fungerar i webbläsare utan insticksprogram. Vaadin skiljer sig från andra ramverk genom att man endast behöver kunna programmera Java för att använda ramverket. De flesta andra ramverk kräver en blandning av olika programmeringsspråk för att få applikationerna kompletta. Vanliga exempel i på teknologier som brukar behövas utöver det primära programmeringsspråket är är HTML, JavaScript och XML.

Ramverkets struktur

Vaadin applikationer är delade i två klara lager: en klientsida och en serversida. Klientens komponenter baserar sig på GWT som är ett ramverk utvecklat av

Google. Vaadin skiljer sig från GWT genom att själva applikationers affärslogik sparas helt och hållet på servern när man i GWT oftast även har logiken i JavaScript-koden i klienten. Klienten, webbläsare, innehåller således inget utöver enskilda komponenters presentationslogik. Basen av serversidan utgörs av en Java Servlet som kommunicerar med klienten över HTTP. Servleten kräver en servlet-container vilket finns inkluderat med webbservrar som möjliggör Java-baserade webbteknologier som t.ex. JSP-sidor och Servlets. Kommunikationen mellan klienten och servern är av JSON format. JSON används för att syntaxen är lite lättare än XML och det finns snabba JSON-läsare gjorda med JavaScript att använda i klienten.

1.5 Prestandatestning

Det är skäl att utföra prestandatestning på en webbtjänsts alla delar, dvs. klient-sidan som oftast finns i användarens webbläsare samt server-sidan av tjänsten som körs på tjänsteleverantörens servrar.

1.5.1 BUSS testning

Ett vanligt förekommande begrepp att tackla prestandatestandet är att använda sig av den s.k. BUSS-testningsmetoden (eng. LESS: Load-, Endurance, Stress-, Spiketesting) som Subraya nämner[7]. Det innebär att man delar upp själva prestandatestandet i fyra olika infallvinkel vilka alla ger information om systemets beteende ur olika aspekter och tillsammans bildar de en nästan heltäckande bild av prestandan.

Belastningtestning testar systemet på en normal och förväntad belastning. Den hjälper till att verifiera att allt fungerar som man har tänkt att det ska. Uthållighetstestningen kontrollerar att applikationen står ut med en belastning under en längre period. Ibland finns det buggar och brister som framkommer först efter en längre tids anvädning.

Stresstestning testar gränserna för hur stor belastning applikationsmiljön klarar av. I praktiken ökar man gradvis belastningen och följer med när systemet blir obrukbart, då vet man att gränsen är nått.

När man vill se hur systemet reagerar på en plötslig kortvarig ökning av belastningen används spetstestning. Det verifierar att att belastningsspetsen inte får systemet ur balans och att det klarar av att återhämta sig till det normala efteråt. [7]

1.5.2 Profilering

I samband med att man utför prestandatester brukar man även vara intresserad av att förutom reda ut de utåt synliga egenskaperna på en applikation, även

undersöka applikationers interna egenskaper vid belastning. Med webbapplikationer som körs på en server har man oftast möjlighet att koppla verktyg rakt i applikationens underliggande lager på servern. Dessa profileringsverktyg kan sedan ge en god överblick över resursanvändningen i värdsystemet och i applikationen. Man ser bl.a. minnes- och processoranvändingen i olika skeden av exekveringen. Statusen för filsystems- och nätverkshanterar kan uppföljas och man ser även hur olika objekt används och städas bort. På detta sätt kan man relativt lätt få fast minnesläckor och andra processdelar som upptar en oansenligt stor del av de tillgängliga resurserna.

I Java brukar man man koppla profileringsverktygen mot JVM som kör den till bytekod kompilerade javakoden och andra omgivningar har motsvarande abstraherande lager att profilera i t.ex. CLR i .Net ramverket.

Kapitel 2

Prestandatestning av webbapplikationer

Här går jag mera i detalj igenom vad man speciellt bör ta i beaktan när man testar webbapplikationer.

2.1 Koncept och terminologi

Webbapplikationer är beroende av prestandan av tjänsterna. Vid större belastningar är det möjligt att serverns kapacitet inte räcker till att behandla alla förfrågningar och då bör man ta till vidare åtgärder för att se till att alla system fungerar som de bör . Ifall man inte utför prestandatestning på tjänsten, så kan man inte känna till gränsen för nuvarande uppsättningens kapacitet gällande responstid, genomströmming och resursanvänding. [6] Oftast är det för sent att reagera på den hårda belastningen först sedan när den redan är igång. Ett effektivare sätt är att proaktivt reda ut i vilka situtationer man bör reagera och hur man ska reagera. Detta är kärnan i prestandatestandet.

Man kan dela in webbsidor i fem kategorier [7]:

1. Statiska webbsidor
2. Statiska webbsidor med formulär
3. Webbsidor med dynamisk data
4. Dynamiskt genererade webbsidor
5. Webbaserade applikationer

Av dessa kategorier kan man generellt konstatera att de tre första kan betraktas som traditionella och de två senare som modernare webbapplikationer.

FIXME: referera listan korrekt åt subraya eller powell

2.1.1 Utförande

Utförandemönstret av prestandatester varierar lite beroende på implementationen och eventuella begränsningar hos testprogramvaran, men man kan ändå karaktärisera vissa gemensamma drag.

Prestandatesterna behöver förberedas på något sätt. Sällan går det varken tekniskt eller innehållsmässigt att köra ett prestandatest omedelbart och förvänta sig tillförlitliga resultat. Man behöver veta exakt vad man testar och hur man mäter det. Testdata som används måste genereras och omgivningen där applikationsservern körs, bör sättas i ordning. När man kör testerna måste man veta hur länge det skall köras och i vilka eventuella fall man avbryter testandet. Efter att alla tester har körts, så ska man givetsvis analysera resultaten och dra sina slutsatser om prestandan.

2.1.2 CSRF

Cross site request forgery (CSRF), även kallad Session Riding och XSRF, är en angreppsmetod och går ut på att man kan komma åt att skicka ett eller flera anrop till servern i den leigitima användarens namn och session. Ifall man har cookie-baserad sessionshantering, så är det möjligt att en tredje part lyckas lura användaren att t.ex. öppna en länk med ett anrop till servern vilket användarens sessionsinformation skickas med. Servern har i normala fall ingen möjlighet att skilja de legitima och angripande anropen från varandra.

Efter att ha prövat en del halvdåliga alternativ för att lösa detta problem, har man kommit fram till att det troligen mest effektiva sättet att bekämpa detta är att servern genererar en token som endast de nämda parterna känner till och denna token sänds med varje anrop till servern som kan verifiera att klienten hade som avsikt att sända anropet. Denna metod kräver en del mera jobb på implementationen för att få clienten och servern att sända med igenkänningstecknet och verifiera det varje gång. [**?**]

FIXME: Fixa referensen

Ifall tokens är ibruktagna hindras GET, POST och övriga HTTP anrop till servern helt och håller ifall man inte är behörig till sessionen. Det ställer också till en del besvär ur prestandatestandets synvinkel t.ex. i Vaadin. Ifall man i Vaadin vill lyckas få alla anrop att tas i beaktan när man använder testverktyg som spelar upp en förinställd scenario, måste man antingen inaktivera XSRF kontrolller i konfigurationsfilen eller så parametriserar man tokenen så att man ur första anropets svar sparar den och sedan injicerar den till alla andra anrop efter det. Då uppnår man samma konfiguration på servern under testandet, men själva testklienten förbrukar mera tid och resurser när man måste har en parser som letar fram tokenvärdet vid varje ny simulerad användare och sätta till rätta värdet i de övriga anropen för den sessionen.

2.2 Traditionella webbapplikationer

Som traditionella webbapplikationer räknas, som tidigare konstaterat, statiska sidor samt sidor med dynamisk data som presenteras. Själva webbklienten är dock fortfarande statisk, vilket betyder att man även på serversidan har en verklig bild om hur sidan ser ut när man ger ett svar på en anrop till servern. Själva webbläsarinnehållet är statiskt och ändras inte under tiden den presenteras för användaren.

Dessa sidor på så sätt lätta att prestandatesta iom. att man vet klientens tillstånd hela tiden utan att ha en verklig klient. Svaren på HTTP-anropen innehåller alltid hela webbsidans DOM-träd. Ur denna är det sedan lätt att parsea vidare och hitta alla länkar och formulär man kan klicka och fylla i för att generera belastning på servern.

Vanlig är att man ordnar själva testerna i olika slags användarscenarier vilka simulerar verkliga arbetsflöden för en mera realistisk bild av relationen mellan användingen och prestandan.

2.3 Moderna webbapplikationer

Största delen av webbapplikationerna idag kan anses vara moderna och använder sig av Ajax, HTML5 samt vissa instickspprogram så som Flash och Silverlight för att få applikationen att verka så lika vanliga skrivbordsapplikationer som möjligt. När man inte längre ha en traditionell applikationsmodell där varje händelse orsakar en omladdning av hela sidan, utan belastningen på servern är mera distribuerat i klienten, så ändras också den belastningsprofil som servern har. Istället för stora uppdateringar mera sällan har man istället mindre uppdateringar

Man behöver inte simulera klientsidan av applikationen alls, utan man inte kan vara säker på när en sida har laddat klart i samband med

2.3.1 Ajax

Synkronisera sidladdning

Ett problem med Ajax-baserade sidor är att man inte kan avgöra webbapplikationens fulla beteende utan att även simulera webbläsarklienten innehållandes dess Javascript samt DOM. I en typisk Ajax-applikation är initialiseringen av en sida mycket lätt och kompakt, medan det sedan innehåller anrop till Javascript-kod som gör en del DOM-manipulationer och XHR anrop i bakgrunden efter att webbsidan laddats. [5] [1]

2.4 Verktyg för prestandatestning

2.4.1 HtmlUnit

Ett alternativ att prestandatesta moderna webbapplikationer som använder sig t.ex. av Ajax är att man simulerar även hela klientwebbläsaren i testerna. Man kan gott ignorera att rendera sidorna, men genom att hålla en uppdaterad DOM i minnet kan man agera så som om man surfade med en vanlig webbläsare. Med ett eget testprogram som använder sig av HtmlUnit, kan man lätt åstadkomma detta.

Det negativa med det här angripssättet är det att det är minnes- och processorkrävande att simulera klienten. Även om inget ritas ut på någon skärm, så är det en avsevärt stor mängd beräkningar som måste utföras. Rhino som fungerar som Javascript-motorn i HtmlUnit klarar heller inte av de allra mesta avancerade scripterna och DOM-manipulationerna.

2.4.2 Apache JMeter

Apache jMeter är ett universalverktyg när det gäller prestanda- och belastningstestning. Det är ytterst anpassningsbart till olika situationer och protokoll, dels via inbyggda egenskaper och dels tack vare möjligheten att utveckla egna insticksprogram till det. JMeter innehåller ingen klientmotor, så man utför belastningstestningen genom att skicka anrop till servern och ta emot svaren från den.

Man kan spara testscenarion i filformat och dessa kan sedan användas som sådana eller genom att parametrisera anropen. Vidare finns det olika controllmöjligheter hur testet ska köras. [2]

2.4.3 WebLoad

2.4.4 curl-loader

2.5 Resultat

Efter att ett prestandatest har utförts, bör man noggrant analysera resultaten för att se hur systemet presterar. De måtten (eng. metrics) man oftast tittar på för en webbapplikation är:

1. Svarstid (medel)
2. Svarstid (max)

3. Felfrekvens
4. Genomströmmning
5. Anrop per sekund
6. Samtidiga användare

Med svartiden menar man den tur och returtid det tar för ett anrop från att det sänds från webbläsaren tills den efterfrågade informationen skickas tillbaka från servern. Både medelvärdet för alla anrop samt det maximala värdet är av intresse, så man lär sig känna till hur länge en användare kan tänkas bli tvungen att vänta på ett svar vid hög belastning.

Felfrekvensen innehåller både sådana fel där servern själv klarar av att meddela med HTTP statuskoder att den inte klarar av anropet och sådana som testapplikationen måste timeouta för att konstatera att det dröjer för länge för att svaet ska vara förväntad längre. Ofta vill man i vilket fall som helst få fast helt alla felsituationer.

Genomströmningen är helt enkelt traffikmängden som genereras till och från servern.

Anrop per sekund räknar skickade antalet anrop till servern för att presentera innehållet.

Virtuella användare är viktiga, speciellt ifall man har realistiska testscenarion, för att belasta systemet. Detta är bland de viktigaste testaspekterna som finns relaterade. Virtuella tänketider, som distribuerar belastningen nämnvärt, är obligatoriska.

FIXME: Avsluta från http://loadstorm.com/load-testing-metrics

[4]

Kapitel 3

Sammandrag och slutsatser

Detta kapitel diskuterar jag kort om vad jag har dragit för slutsatser i anknytning till min tes om att det är möjligt att prestandatest moderna webbappliaktioner.

3.1 Möjligt att testa

En modern webbapplikation är full möjlig att utför lika detaljerade prestandatester på som på en traditionell applikation. Det kräver visserligen att man är insatt på ett annat sätt hur logiken inom applikationen fungerar, för att man ska kunna testa alla egenskaper fullt ut.

3.2 Vad kunde förbättras

I och med att man inte lätt kan prestandatesta Ajax-baserade webbapplikationer utan att simulera även klienten så när som på sin hela funktionalitet, efterfrågas det mycket om nya testclienter där även en snabb klientsimulering är nödvändig. Ifall man inte behövde tänka på att klienterna till prestandatesterna kräver en massa resurser, utan detta fanns automatiskt inbyggd i testklienten, så kunde man spara en stor mängd manuellt arbete i samband med testandet.

Litteraturförteckning

[1] Ajax: A new approach to web applications - http://www.adaptivepath.com/ideas/e000385.

[2] Apache JMeter hemsida - http://jakarta.apache.org/jmeter.

[3] HTTP/1.1 standard - http://tools.ietf.org/html/rfc2616.

[4] Load testing metrics - http://loadstorm.com/load-testing-metrics, hämtad 11.4.2011.

[5] Testing and optimizing single page web 2.0/ajax applications – why best practices alone don't work any more - http://blog.dynatrace.com/2011/03/22/testing-and-optimizing-single-page-web-2-0ajax-applications-why-best-practices-alone-dont-work-any-more hämtad 8.4.2011.

[6] Dennis Rea J.D. Meier Carlos Farre Prashant Bansode, Scott Barber. *Performance Testing Guidance for Web Applications*. Microsoft Corporation, 2007.

[7] B.M. Subraya. *Integrated Approach to Web Performance Testing: A Practitioner's Guide*. IRM Press, 2006.