**Start**

hallo und herrzlich willkommen zu meinem seminarvortrag. mein name ist daniel bin masterstudent im 3ten semester und präsentiere euch heute das thema lasttests von webseiten mit jmeter.

hier ein überblick des inhalts. ich möchte zuerst mit einer einführung in die welt von lasttests starten danach zu apache jmeter übergehen, anschließend 2 live demos zeigen und abschließend ein fazit zu dem ganzen geben.

**einleitung:**

als einleitung fand ich das bild hier ganz passend. es stammt aus einem work life balance artikel aber trifft das thema lasttest ganz gut, da man auch schauen muss, dass das system unter der last nicht zusammenbricht.

worum geht es in der einleitung? ich will kurz die motiviation ansprechen, wozu man braucht lasttests benötigt, danach basiswissen von lasttests vermitteln und was für anforderungen an lasttests gestellt werden.

**motivation:**

Bestes beispiel dass für lasttests spricht und dass dem herrn vogelsang wohl ewig nachhängen wird, ist der berühmte ausfall der intranet seite, als das erste mal die seminare und projekte online aushingen und man sich dort auf der seite anmelden musste. dort hat sich das system aufgrund der großen anzahl gleichzeitiger zugriffe irgendwann verabschiedet.

Es gibt aber auch weitere beispiele wie etwa ein neuer Online Shop der viele Besucher erwartet, hat dann plötzlich große Latenzzeiten oder ist irgendwann gar nicht mehr erreichbar. Wäre natürlich der Worst Case. Gerade zum Thema Latenzzeiten gibt es eine Studie von Greg Linden (damals noch bei amazon) – Es wurde ermittelt, dass Amazon pro 100ms verzögerung 1\% weniger verkaufserlöse hat.

Daraus folgt Zeit = Geld / schneller = besser für den umsatz.

**grundlagen:**

Bei einem lasttest, wird wie der name es vermuten lasst hohe last auf einem system erzeugt und das verhalten dessen untersucht. lasttests haben dabei folgende ziele: 1 2 3

lasttests sind oft die letzten schritte der entwicklung bevor die software eingeführt wird, was dazu verleitet diese so schnell wie möglich hinter sich zu bringen. man sollte sich dennoch genügend zeit für sie nehmen um am ende nicht noch eine böse überraschung zu erleben.

**testen:**

Es gibt manuelle und automatisierte tests. Beim manuellen testen benötigt man einige Benutzer, die sich durch die anwendung klicken. In kleineren Firmen oft die Entwickler selbst, die in dem Zeitraum nicht zur Verfügung stehen. ist ein problem weil in der zeit nicht entwickelt wird. das manuelle testen ist daher sehr zeit und resourcenintensiv. beim automatisierten testen werden programme verwendet. etwa jmeter gatling oder the grinder. sind 3 kostenlose tools die alle lasttests machen. diese tools simulieren die zugriffe von vielen benutzern.

kleine anmerkung: da lasttests häufig am ende der entwicklung gemacht werden wenn die anwendung bereits gehostet ist, verwechselt man lasttests häufig mit UATs. UATs fokusieren sich allerdings mehr auf funktionalität und ob die anwendung funktioniert.

**anforderungen:**

Ein lasttest sollte mindestens folgende fälle abdecken können: verschiedene lastvariationen simulieren, z.b. viele gleichzeitige anfragen über einen konstanten zeitraum; ab einer bestimmten tageszeit sehr viele zugriffe – sogenannte spitzenzeiten

die antwortnachrichten sowie antwortzeiten und der datendurchsatz sollte einsehbar sprich loggbar sein, die festlegung auf welcher infrastruktur der test laufen soll sprich cloud dienste oder lokal.

**jmeter:**

All dies und weitere funktionalität bietet das open source Lasttestprogramm JMeter von Apache. version 1.0 wurde von Stefano Mazzocchi im jahr 1999 entwickelt um ursprüunglich die performance von Apache Tomcat zu testen. Hat sich daraufhin stetig weiterentwickelt und ist heute der de facto standard bei großkonzernen wie sap und 1und1 wo es regelmäßig im einsatz ist und den resourcenbedarf von anwendungen ermittelt. auf die hier abgebildete gui wird später genauer eingegangen

**jmeterübersicht:**

jmeter ist open source, kann weiterentwickelt werden. es ist platformunabhängig d.h. läuft auf windows linux und mac.

gui um aufzeichnung zu erstellen und eventuell einfache tests zu machen. non gui in der cloud, performanter. man kann automatisiert ablaufen lassen

**gui**

sobald man jmeter heruntergeladen und entpackt hat navigiert man in das bin verzeichnis. hier kann man jmeter entweder via kommandozeile starten oder durch ausführen der jmeter exe. die kommandozeile hat den vorteil dass man mehrere kommandos mitgeben kann, und z.b. ein html dashboard generieren kann auf das ich noch später zurückkomme. auch ist der resourcenverbrauch deutlich geringer als mit der gui. gui hat den vorteil, dass es für beginner ziemlich einsteigerfreundlich ist. man sieht hier die debug console unten und den testscript recorder sowie auf der linken seite den hierarchischen aufbau einzelner elemente auf die ich jetzt zu sprechen komme.

**testplan**

als root element in der hierarchie steht der testplan. diesen gibt es pro jmx file nur 1 mal. hier kann man optionen für thread gruppen einstellen (ob hintereinander oder alle auf einmal starten, man kann hier libraries importieren usw. auch user variablen können hier global festgelegt werden

**thread gruppe**

testplan ist eine jmx datei die im xml format gespeichert wird. jeder testplan kann 1 oder mehrere thread gruppen enthalten. diese steuern die anzahl der threads, sowie wiederholungen. außerdem kann man hier das fehlerverhalten steuern. (also was soll bei einem fehler passieren

**sampler:**

in den thread gruppen kann man dann die sampler erstellen, welche gegen ein system gefeuert werden. jdbc, http, java usw.. JUnit

**controller:**

der controller nur im thread-group steuert zeitlichen ablauf. wie oft wann etc. den durchsatz und unterstüzt if while foreach bla

**config:**

config elemente können sowohl im sampler selbst, der threadgruppe oder auch global im testplan liegen. hier kann man variablen deklarieren und platzhalter für gewisse werte definieren http header oder cookies verwalten

**listener:**

Listener sind für die ausgabe der ergebnisse zuständig. diese gibts es z.b. als results tree wo die ergebnisse nur aufgelistet werden, oder als graph schön visualisiert oder auch als aggragation report. wichtig ist zu wissen, dass es sich bei den unterschiedlichen listenern immer um die gleichen daten handelt. lediglich die darstellung ist eine andere

**fazit:**

jmeter ist ein super tool um die lasttests an webseiten auszuführen. es platformunabhängig und kostenlos. Man hat viele tutorials und how-tos auf der offiziellen seite von apache. dadurch entsteht eine flache lernkurve und man bleibt stehts motiviert sich weiter mit dem thema zu beschäftigen. Testerstellung ist dank GUI sehr einfach und unkompliziert. Der testlauf kann sowohl in der GUI als auch in der Kommandozeile erfolgen. Die visualisierung der ergebnisse durch das html dashboard sind zusätzlich ein super schmankerl um auch außenstehende z.b. POs oder Vorgesetzten von lasttests zu überzeugen.

Einige nachteile sollten auch nicht unerwähnt bleiben. So etwa die veraltete Swing Oberfläche, stört zwar nicht sonderlich aber rein optisch betrachtet könnte man sich da ein update wünschen. zweiter nachteil: die passwörter die beim recording gespeicehrt werden sind im klartext hinterlegt. sollten also möglichst fix in variablen verpackt und in z.b. base64 codiert werden. Die testpläne sind alle im JMX format gespeichert, wobei es sich dabei um eine XML datei handelt. daher ist es sehr unhandlich sampler werte direkt in der datei zu editieren. die gui frisst zudem ziemlich viel an hardware resourcen was sich beim testen von mehreren threads irgendwann bemerkbar macht.

als neutral wollte ich hier auch noch 2 punkte nicht unerwähnt lassen. beide punkte sind eher allgemeine probleme. zum einen können die REST endpunkte einer anwendung nicht einfach mal via brute force herausgefunden werden und alle in ein jmx file gespeichert werden. dies wäre bei sehr vielen endpunkten eine deutlich erleichterung. es gibt nur die möglichkeit mit swagger z.b. die api zu dokumentieren oder eventuell mit WADL die endpunkte und ihre vor und nachfolger zu definieren. dies obliegt jedoch dem programmierer der REST API.

Der zweite punkt ist die captcha erkennung. ein captcha dient ja dazu eine reale person von einem bot oder computer zu unterscheiden. dementsprechend ausgelegt sind die captchas. und jmeter als programm ist es nicht möglich diese zu erkennen. ist aber ein generelles problem aller programme. hier muss man beim testen captchas deaktivieren oder ein vordefiniertes verwenden.

gibt alternativen zu jmeter wie z.b. gatling oder selenium oder the grinder. wer lust hat kann sich gerne damit befassen. oder als nächste seminararbeit

**abschluss:**

in den zeiten von TDD ist der fokus vermehrt wieder testen gerückt. allerdings nur in richtung unit/component/integration testing. lasttests fristen nach wie vor ein nischendaschein und JMeter und konsorten werden leider viel zu selten in der entwicklung eingesetzt.

das hat auch einen einfachen grund? und zwar ist es in der heutigen zeit und der immer größer werdenen programme wichtiger dass das programm fertig ist.

an dieser stelle möchte ich noch herrn pape zitieren, der mal in seiner optimierung von programmen vorlesung gesagt hat: VORLESEN.