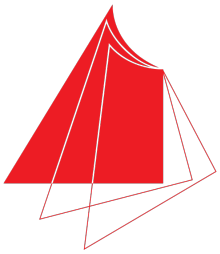

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Studiengang Informatik (Master)

Last-Test von Web-Seiten mit JMeter

Seminararbeit - Ausarbeitung

Daniel Schäfer (60118)

Betreuer: Prof. Dr. Holger Vogelsang

Sommersemester 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Mit dieser Ausarbeitung zur Seminararbeit wird das Performance und Last-Test Programm Apache JMeter ausführlich unter die Lupe genommen. Es werden seine Funktionen und Möglichkeiten erläutert und kurz Alternativen dazu erwähnt. Zusammenfassend werden Vor- und Nachteile aufgelistet, sowie ein Fazit gegeben.

1.1 Motivation

Man erlebt es immer wieder, das Webseiten unter zu hoher Last in die Knie gezwungen werden. Sei es nun Amazon, Netflix und Konsorten durch DDoS-Angriffe [8], oder auch hochschulinterne Seiten, die durch gegebene Anmeldefristen eine hohe Anzahl von gleichzeitigen Benutzern zu bewältigen haben.

Gerade letzteres Szenario lässt sich im voraus gut vermeiden, da man die grobe Anzahl der Studierenden kennt und somit präventiv die Webseite auf die eingehende Last testen kann. Dabei werden sehr viele nebenläufige Anfragen an einen Anwendung gestartet und die Antwortzeiten ausgewertet. Diese Art von Tests nennt man Last- oder Performance-Tests und wird in die Klasse der Systemtests kategorisiert [13].

Das frei verfügbare, unter Public-License-Key stehende Java Programm JMeter bietet diese und weitere Funktionalität, um die Leistung einer Webseite bis ins kleinste Detail zu analysieren, zu testen, auszuwerten und zu visualisieren.

1.2 Manuelle und automatisierte Tests

Bevor es an das eigentliche Thema JMeter geht, gibt es einen kleinen Exkurs in die sogenannten Systemtests. Diese wurden im vorherigen Abschnitt kurz erwähnt und haben die Besonderheit, dass sie sowohl manuell, als auch automatisiert ausgeführt werden können.

1.2.1 Manuelles Testen

Beim manuellen Testen werden bestimmte Aktionen und Anfragen von mehreren Benutzern nach bestimmten Vorgaben gestartet und die resultierenden Ergebnisse protokolliert. Es liegt auf der Hand, dass diese Art des testens ziemlich zeit -und resourcenaufwändig ist. Zusätzlich muss das Personal, sprich die Tester organisiert, betreut und verpflegt werden.

Falls es sich dabei um die Programmierer selbst handelt, stehen diese im Zeitraum auch nicht für andere Tätigkeiten zur Verfügung. Der Faktor Mensch spielt natürlich bei dieser Art der Tests eine Rolle, wodurch nicht ausgeschlossen werden kann, dass Fehler während der Testphasen gemacht und diese entsprechend nicht erfasst werden. [9, S. 11]

Trotz dieser Nachteile sollte man auf zusätzliche manuelle Tests[1] einer Anwendung nie verzichten, da eine „reale" Person über den Tellerrand schauen kann und es dadurch möglich ist, diverse Bugs ausfindig zu machen, die mit dem eigentlichen Test unter Umständen gar nichts zu tun haben.

1.2.2 Automatisiertes Testen

Automatisiertes Testen funktionieren immer dann recht gut, wenn häufige Wiederholungen auftreten. Beispielsweise bei Regressiontests. Aber auch bei Lasttests ist die Testautomatisierung ein gängiges Verfahren. Die Tests laufen in erster Linie mit Software, was zur Folge hat, dass sie ziemlich statisch sind und dadurch beispielsweise keine Ästhetik geprüft werden kann, wie dies etwa bei menschlichen Testpersonen der Fall wäre.

Jedoch haben automatisierte Tests den großen Vorteil, dass sie kosteneffizienter sind. Man benötigt lediglich ein Budget für etwaige Lizenzgebühren der Software oder für den Support. Des weiteren ist man durch Testsoftware in der Lage, sehr viele Benutzer gleichzeitig zu erzeugen und parallel auf ein System zugreifen zu lassen. Mit Testskripten sogar rund um die Uhr [12]. Dadurch werden Systeme bis zur ihren Grenzen und darüber hinaus getestet.

2 Apache JMeter

Wen man sich nun für die Testautomation entschieden hat, steht man vor der Wahl einer entsprechenden Software. Diese befinden sich in einem Preissegment von kostenlos bis hin zu fünfstelligen Bereichen. Die Wahl hängt letzten Endes von den eigenen Anforderungen ab [9, S. 15]. Da sich die Arbeit auf JMeter bezieht richten wir unser Augenmerk auf diese spezielle Software.

Wie man in Abbildung 1 erkennen kann, erscheint Apache JMeter mit

[1] Das sogenannte User Acceptance Testing (UAT) ist gerade im finalen Entwicklungsstadium einer Anwendung unabdingbar. [16]

seinen „Metal-Look-and-Feel-Widgets“ [15] wie das Relikt aus einer längst vergangenen Zeit. Man sollte sich jedoch von der veralteten und trägen Swing-basierten GUI nicht täuschen lassen, denn hinter der Oberfläche steckt ein mächtiges Werkzeug, mit dem man alle möglichen Arten von Tests erstellen und ausführen kann. [10]

Tatsächlich stammt JMeter aus einer Zeit als das Web und die Application-Server noch in den Kinderschuhen steckten. Ursprünglich wurde es entwickelt um den Application-Server Tomcat zu testen. Seitdem wurde das Programm fortlaufend weiter entwickelt, so dass man heute in der Lage ist diverse Tests zu realisieren. Von verteilten Tests in der Cloud, über das Testen von dynamischen Webseiten bis hin zu Javas JUnit-Tests. [7]

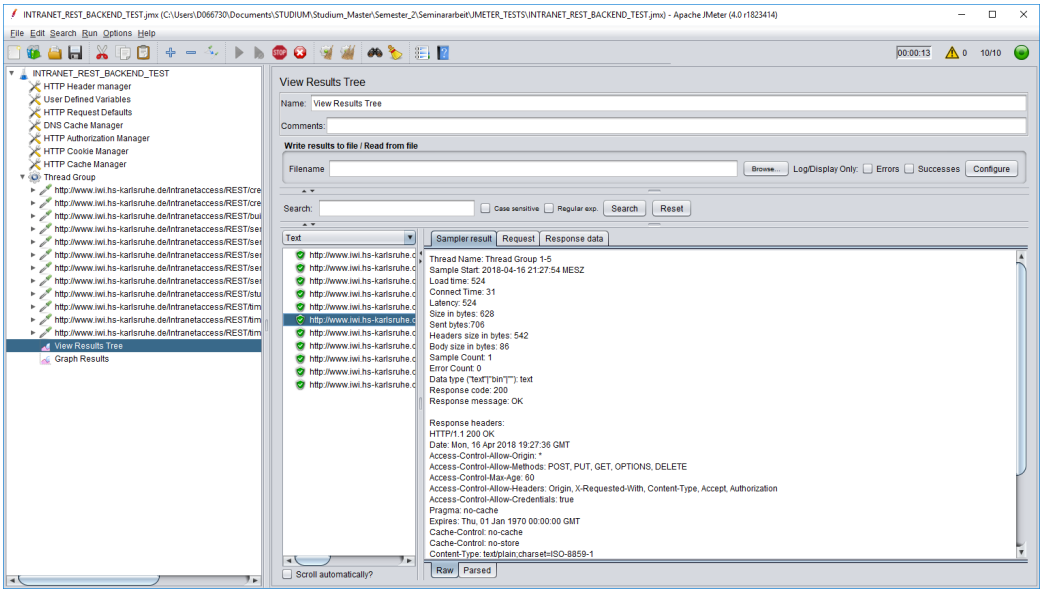

Abbildung 1: Apache JMeter

Apache JMeter wurde in Java geschrieben und kann durch seine plattformunabhängigkeit unter jedem beliebigen Betriebssystem, welches eine Java-Runtime-Environment (JRE) installiert hat, verwendet werden. Neben der GUI kann man Apache JMeter auch mit der Kommandozeile bedienen. Dies ist gerade bei speicherintensiven Lasttests empfehlenswert, da die GUI schnell an ihre Grenzen gerät und sich die Testergebnisse dadurch nicht mehr genau erfassen lassen. Die Verwendung der JMeter Kommandozeile wird in Abschnitt 2.5 genauer betrachtet.

Apache JMeter bietet zusätzlich eine gut dokumentierte API an, mit der es möglich ist das Programm selbständig zu erweitern.

2.1 Historisches

Ursprünglich wurde Apache JMeter von Stefano Mazzocchi, seiner Zeit Entwickler bei der Apache Software Foundation, programmiert um die Performance von Apache Tomcat (damals noch Apache JServ) zu testen. Kurz danach wurde das Programm neu entworfen und mit einer verbesserten GUI und zusätzlichen Möglichkeiten von Funktionstests ausgestattet. Im Jahr 2011 wurde JMeter zu einem sogenannten Top-Level-Apache-Project, was ein Projekt-Management-Committee und eine offizielle Homepage mit sich brachte [6].

Mittlerweile ist Apache JMeter ein wichtiger Bestandteil von Performance-Tests in sehr vielen Firmen geworden. Großkonzerne wie SAP oder 1&1 verwenden regelmäßig JMeter um die Verfügbarkeit ihrer Produkte unter hoher Last zu prüfen.

2.2 Was kann JMeter

Apache JMeter dient in einer Client/Server Landschaft als Client und kann dadurch Anfragen an bestimmte Anwendungen absetzen. Dadurch erhält man unter anderem die Antwortzeit, Antwortmessage oder aber auch den Speicherverbrauch der Anfrage.

Anfragen können sowohl an statische als auch an dynamische Resourcen erfolgen. Darunter fallen unter anderem statische Dateien, Servlets, FTP-Server, Datenbanken, Java-Objekte und Skripte. Um diese Anfragen in einer entsprechend großen Anzahl abzufeuern, bietet das Programm die Simulation von vielen gleichzeitigen Benutzern an. Diese sogenannten Threads lassen sich in einzelne Thread-Gruppen unterteilen.

In JMeter ist es ebenfalls möglich, den Test in eine Cloud Infrastruktur auszulagern und die Tests unabhängig vom eigenen System bzw. Hardware laufen zu lassen. Ebenfalls kann man mehrere Systeme dazu bringen, die Tests verteilt auszuführen, was den Vorteil bringt, deutlich mehr Resourcen zur Verfügung stehen zu haben.

Nach den Tests bietet JMeter ein HTML-Dashboard an, in dem sehr viele Informationen über die Anfragen und Ergebnisse in Tabellen und Statistiken grafisch aufbereitet dargestellt werden. Abbildung 2 zeigt einen Ausschnitt des Dashboards, welches im Abschnitt 5.2 genauer untersucht wird.

Eine weitere nützliche Funktion ist das Aufzeichnen von Userinteraktionen. Dieser Http-Test-Script-Recorder wird in JMeter mit bestimmten Filtern und Pattern vorkonfiguriert und gestartet. Jede Aktion im Browser

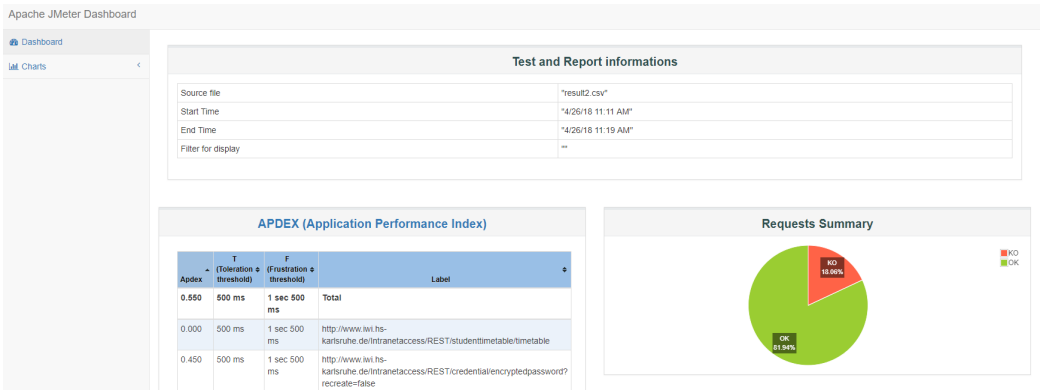

Abbildung 2: HTML-Dashboard-Report

mit der entsprechenden URL und Port wird dann als einzelner HTTP-Request-Sampler angelegt. Der Recorder wird initial einmal ausgeführt und kann dann für diverse Tests recycelt werden.

Seit der JMeter Version 2.1.2 ist es außerdem möglich in Java geschriebene JUnit-Tests als JUnit-Sampler zu importieren und auszuführen. Diese nützliche Funktion wird in Abschnitt 8 untersucht.

2.3 Installation

Im folgenden Kapitel werden notwendige Schritte zur Inbetriebnahme von Apache JMeter erklärt.

2.3.1 Einrichten der Umgebung

Apache JMeter wurde als reine Java Anwendung entwickelt. Sie ist dadurch platformunabhängig und benötigt keine zusätzlichen Treiber oder Installation. Alle notwendigen Abhängigkeiten und Klassen sind im entsprechenden Archiv hinterlegt. Zum Starten muss man lediglich die JAR-Datei aus dem `\bin` Verzeichnis ausführen.

Auf der offiziellen Seite von Apache JMeter[2] muss man sich zunächst eine Release-Version herunterladen. Den Build entpackt man dann in ein beliebiges Verzeichnis, beispielsweise `C:\\Program Files\jmeter\` . Es sollte bereits eine aktuelle JRE vorhanden sein, da diese Grundvorausstzung ist um JAR-Dateien auszuführen. Prüfen kann man dies, indem man in

[2] http://jmeter.apache.org/download_jmeter.cgi

der Kommandozeile den Befehl `java -version` eingibt. Wenn keine JRE installiert ist kommt eine entsprechende Meldung, dass der Befehl nicht gefunden wurde. In diesem Fall hilft eine Installation der Java runtime environment von der offiziellen Oracle Seite unter http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Im Installationspfad befinden sich mehrere Unterverzeichnisse. Wichtig sind hier der `\bin` Ordner und der `\lib` Ordner. Im ersteren befindet sich die ApacheJMeter.jar, welche die GUI von JMeter startet. Im `\lib` Verzeichnis sind alle Bibliotheken und Erweiterungen hinterlegt. Dies ist auch der Ort an dem zusätzliche Third-Party-Plugins hineinkopiert werden müssen.

2.3.2 Starten von JMeter

JMeter lässt sich wie bereits erwähnt als GUI bzw. als Kommandozeilenprogramm verwenden. Um JMeter als GUI zu starten kann man direkt die ApacheJMeter.jar ausführen. Sitzt man hinter einer Firewall oder einem Proxy-Server empfiehlt es sich die GUI via Kommandozeile zu starten, da man hier zusätzlichen Funktionsumfang hat um den Proxy zu übergeben. Folgende Parameter sollten dabei verwendet werden:

Parametername	Bedeutung
-H	Proxy-Server-Hostname bzw. IP-Adresse
-P	Port des Proxy-Servers
-N	Host ohne Proxy-Server (localhost)
-u	Benutzername für den Proxy-Server
-p	Passwort für den Proxy-Server

Tabelle 1: Befehle für Firewall Einstellungen

Beispiel: Aus der Kommandozeile ins `\bin` Verzeichnis von JMeter navigieren und folgenden Befehl ausführen:

```
1 jmeter -H 196.168.178.1 -P 1337 -u lindan -a hispassword -N
    localhost
```

Alternativ dazu die Befehle der JMeter Kommandozeile ohne GUI:

Parametername	Bedeutung
-n	non-GUI; keine GUI-Oberfläche
-t	JMX-Datei, dass die Testfälle enthält
-l	JTL-Datei, für die Protokolle
-r	Verwende alle Remote-Server aus jmeter.properties

Tabelle 2: Befehle für JMeter Kommandozeile

Beispiel um die test.jmx Datei auszuführen und in logtest.jmx zu speichern: Aus der Kommandozeile ins `\bin` Verzeichnis von JMeter navigieren und folgenden Befehl ausführen:

```
1 jmeter -n -t test.jmx -l logtest.jtl
```

2.4 JMeter-GUI

Die GUI von JMeter ist die erste Anlaufstelle für Anfänger und interessierte Benutzer. Hier kann man schnell und einfach Testaufrufe erzeugen und Ergebnisse anhand von Graphen und Tabellen anzeigen.

Die GUI wird von erfahrenen Anwendern hauptsächlich dazu verwendet um die Test-Sampler zu generieren, die dann später aus der Kommandozeile heraus als JMX-Dateien gestartet werden.

Hier kann man auch den HTTP-Test-Script-Recorder konfigurieren und starten, mit dem es möglich ist, diverse HTTP Anfragen an Webseiten aufzuzeichnen. Abbildung 3 zeigt die GUI beim Abarbeiten mehrerer gleichzeitiger Anfragen an eine Webseite. Der Graph kann das ganze zusätzlich visualisieren. Dadurch erkennt man, an welchen Stellen noch optimiert werden kann.

2.5 JMeter Kommandozeile

Möchte man nun einen komplexeren Lasttest machen, eignet sich die GUI ab einer gewissen Anzahl von gleichzeitigen Benutzern nicht mehr. Durch die steigende Anzahl der Anfragen wird auch der Resourcenbedarf immer größer und der ohnehin schon recht hohe Speicherbedarf der GUI selbst kann sich dann negativ auf die Messergebnisse auswirken und diese verfälschen.

Hier kommt der sogenannte „Non-GUI-Mode“ ins Spiel. Dies ist die Bezeichnung um JMeter in der Kommandozeile auszuführen. Mit Hilfe des

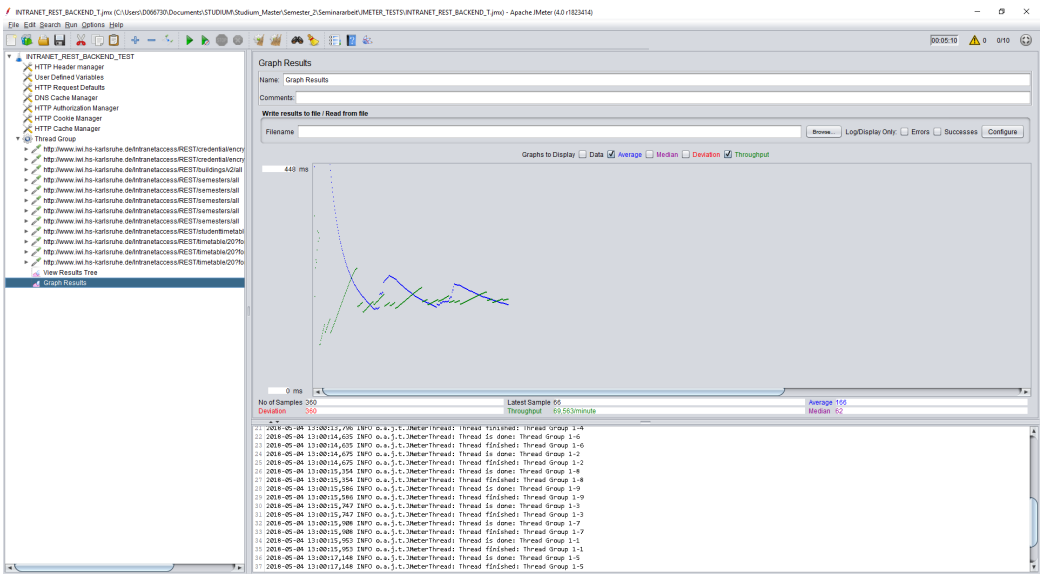

Abbildung 3: JMeter-GUI mit Graph

Kommandozeilenaufrufes lassen sich am Ende des Tests CSV oder XML Dateien erstellen, die Testergebnisse beinhalten.

Auch der HTML-Dashboard-Report lässt sich nur hier generieren.

Zuerst sollte ein Testscript verfügbar sein. In diesem sind alle Anfragen und Konfigurationen wie Cache/Cookie-Verhalten und Auth-Management enthalten. Man kann hier eine vorhandene Datei verwenden oder in JMeter selbst eine neue JMX-Datei erzeugen. Minimale Voraussetzungen sind eine Thread-Gruppe sowie ein Sampler, beispielsweise ein HTTP-Request, mit entsprechender URI und Operation.

Nun öffnet man die Kommandozeile und navigiert in das `\bin` Verzeichnis von JMeter. Dort startet man den Test via folgendem Befehl:

```
jmeter -n -t [Pfad zur JMX-Datei] -l [Pfad zur Ergebnisdatei] -e
    -o [Pfad zum HTML-Dashboard Ordner]
```

In der folgenden Tabelle sieht man einige häufig verwendete Parameter und ihre Bedeutung. Um eine Übersicht aller Parameter zu erhalten, kann man in der Kommandozeile den Befehl `jmeter -?` ausführen. Ein zweiter sehr nützlicher Befehl ist `jmeter -h` , welcher eine Vorauswahl von Kommandozeilenbefehle von JMeter anbietet.

Parametername	Bedeutung
-n	Non-GUI-Mode
-t	Pfad zum JMeter Skript; in Verbindung mit -n
-l	Pfad zur Result-Datei
-?	Hilfe; zeigt alle Parameter an
-h	Beispielbefehle, die man verwenden kann
-L	Log-Level - Wann soll etwas protokolliert werden
-e	Um HTML-Reports zu erzeugen
-o	Pfad zum Output-Ordner; in Verbindung mit -e or -g
-g	HTML-Report wird aus einer CSV-Datei erzeugt
-j	Protokoll-Datei nicht überschreiben, neue generieren

Tabelle 3: Befehle der JMeter Kommandozeile

Pauschal kann man sagen, dass man die GUI bei kleineren Tests bzw. Testplanerstellung verwendet. Für alles andere sollte der Non-GUI-Mode die erste Wahl sein.

3 Der Testplan

Nach dem ganzen Vorgeplänkel geht es nun an die Erstellung eines ersten Testplans für JMeter. Doch was ist überhaupt ein Testplan? Jeder JMeter Test hat genau einen Testplan, der als Wurzel in der Hierarchie angelegt ist. Man kann ihn sich als einen Container vorstellen, der weitere Komponenten beinhaltet kann, die nötig sind um einen Testlauf erfolgreich zu starten. Der Testplan ist ein Skript in einem speziellen JMX-Format. Betrachtet man die Datei in einem Texteditor erkennt man, dass es sich hier um eine XML-Datei handelt (Siehe Abbildung 4). Nach Anlegen und Abspeichern des Testplans kann man diesen via Kommandozeile oder GUI starten. Falls man auf einem Testplan die Option „Functional-Test-Mode“ aktiviert, sorgt das dafür, dass JMeter die Antwort jedes Sample-Aufrufs in eine Protokoll-Datei speichert. Diese Protokolldateien lassen sich in den Listenern konfigurieren.

3.1 Elemente eines Testplans

Um einen Test zu starten benötigt man mindestens einen Testplan, darunter eine Thread-Gruppe sowie einer oder mehrere Sampler. Nachfolgend eine Auflistung der wichtigsten Elemente und ihre Funktion, die in einem Testplan verwendet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="4.0" jmeter="4.0 r1823414">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="MyFirstTestPlan" enabled="true">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.tearDown_on_shutdown">true</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments" guiclass="ArgumentsPanel"
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Users" enabled="true">
        <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
```

Abbildung 4: JMX-Datei in Notepad++ geöffnet

3.1.1 Thread-Gruppe

Zu jedem Testplan gehört mindestens eine Thread-Gruppe. Diese steht stellvertretend für die Anzahl der gleichzeitigen Benutzer und Wiederholungen, die simuliert werden. Abbildung 5 zeigt die wichtigsten Einstellungen der Thread-Gruppe. Neben den selbsterklärenden Bezeichnungen und Werte gibt es noch die Ramp-Up-Period. Diese gibt an, in welcher Zeitspanne die Threads erzeugt werden sollen. Angenommen man gibt 10 Threads an und die Ramp-Up-Period stellt man auf 20 Sekunden, wird alle zwei Sekunden ein neuer Thread erzeugt. Dies verhindert ein gleichzeitiges Erzeugen aller Threads und dadurch eine zu hohe Systemauslastung zu Beginn des Tests.

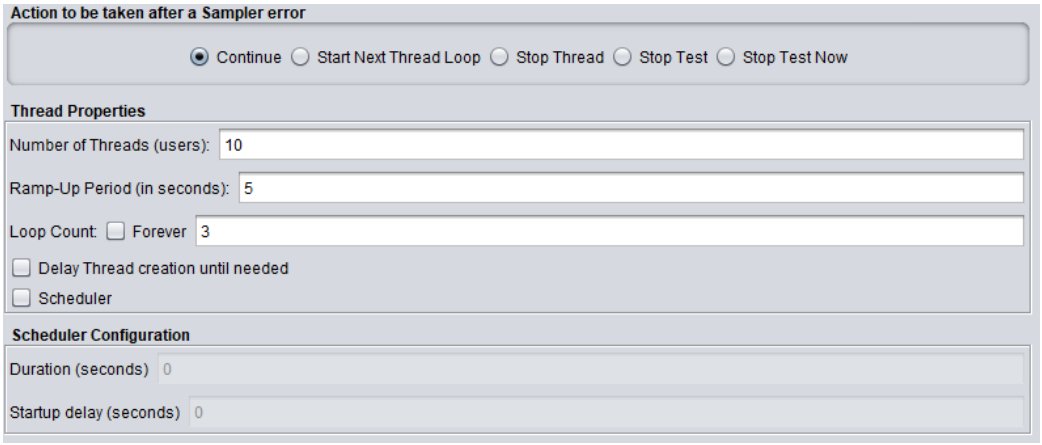

Abbildung 5: Wichtige Einstellungen der Thread-Gruppe

Thread-Gruppen kann man zusätzlich auch als SetUp- oder TearDown-

Gruppen anlegen. Diese werden analog zu JUnit-Tests entsprechend vor der eigentlichen Thread-Gruppe gestartet oder eben nach Beendigung aller Thread-Gruppen. Sehr hilfreich, wenn man beispielsweise Werte aus einer Datenbank auslesen und diese in Variablen speichern möchte, bevor der eigentliche Test beginnt und das Auslesen nicht in den eigentlichen Test mit einfließen sollen [3].

3.1.2 Sampler

Sampler und Logic-Controller steuern die Abarbeitung eines Tests. Dabei sind Sampler für das Senden einer Anfrage verantwortlich und die Logic-Controller für den zeitlichen Aspekt, also wann diese Anfrage gesendet wird. Innerhalb der Thread-Gruppe lassen sich mehrere Sampler anlegen. Diese Sampler sind die eigentlichen Anfragen des Tests und stellen die Verbindung zu den zu testenden Objekten her. Es gibt Sampler für die unterschiedlichsten Protokolle wie HTTP, FTP, JDBC SOAP/XML und Java-Objekte.

HTTP-Sampler sind die wohl am häufigsten eingesetzten Sampler. Mit ihnen lassen sich unter anderem REST Aufrufe an entsprechende Server starten. Abbildung 6 zeigt einen HTTP-Request-Sampler an die hochschulinterne API mit der GET-Methode.

Abbildung 6: HTTP-Request-Sampler

3.1.3 Logic-Controller

Der Logic-Controller steuert den zeitlichen Ablauf einzelner Teile, insbesondere der Sampler und kann man unter anderem angeben, wie oft ein Sampler wiederholt werden soll oder wann dieser gestartet werden soll. Aber auch komplexere Szenarien, etwa durch reguläre Ausdrücke bei beim If-Controller sind machbar.

3.1.4 Timer

Durch einen Timer kann man eine Verzögerung nach jedem Thread einbauen, so dass der Test für kurze Zeit angehalten wird. Es gibt konstante- und Zufallstimer.

3.1.5 Config-Elemente

Zu einzelnen Samplern oder Thread-Gruppen gibt es noch diverse Config-Elemente. Diese dienen dazu Default-Parameter einmalig anzulegen und darauf zuzugreifen. Wird dann in den jeweiligen Controllern kein Wert in die entsprechenden Felder eingetragen, wird auf den Wert im Config-Element zurückgegriffen. Speziell für HTTP-Sampler gibt es darüber hinaus noch weitere Config-Elemente, wie Header, Authorization oder Cookie-Informationen, die man zentral für den ganzen Testplan abspeichern kann [1].

3.1.6 Listener

Listener enthalten Informationen über die Ergebnisse der Performance-Tests. Es gib verschiedene Arten von Listenern, wie etwa den Graph-Result-Listener, den View-Results-Tree-Listener oder einen Aggregation-Report. Wichtiger Hinweis an der Stelle: Egal welcher Listener verwendet wird, die Daten die gespeichert werden sind immer die selben. Lediglich die Darstellung dieser auf dem Ausgabemedium unterscheidet sich. Man kann Listener an jede beliebige Stelle im Test hängen; auch unter einem Testplan. Es werden jedesmal die Daten gesammelt, die ab der eingehängter Stelle präsent sind.

3.1.7 Assertions

Mit JMeter lassen sich auch Zusicherungen, sogenannte Assertions untersuchen. Dazu wird die Antwort einer Anfrage auf ihren Statuscode hin überprüft. Auch Antwortzeiten lassen sich mit Hilfe von Assertions prüfen. Liegt die Antwort in einer bestimmten Zeitspanne oder überschreitet sie diese Zeitspanne, kann man entsprechend darauf reagieren.

4 Aufzeichnen von Aktionen

Es leuchtet ein, dass wenn man eine Webseite auf ihre Performance untersuchen will, sehr viele, möglichst unterschiedliche Anfragen an die Seite stellen möchte. Es soll natürlich eine Testabdeckung von nahezu 100% erreicht werden und so gut es geht alle Pfade innerhalb der URL zumindest einmal besucht werden. Um in JMeter nicht jeden Sampler einzeln anzulegen, gibt es diverse Tools, die Aktionen auf Webseiten aufzeichnen, die in den folgenden Kapiteln kurz erwähnt werden.

4.1 Chrome-Plugin Blazemeter

Ein wohl recht bekanntes Tool ist Blazemeter. Das kostenlose platformunabhängige Chrome-Plugin ist in der Lage Aktionen von bestimmten URLs aufzunehmen und danach zu exportieren. Ein kleiner Wermutstropfen muss man hier noch zusätzlich erwähnen. Für das exportieren der Aufzeichnung in das JMX-Format ist eine kostenlose registrieren für Blazemeter erforderlich. Wem dies nichts ausmacht, erhält mit Blazemeter ein solides Recorcing-Tool, dass eng mit JMeter zusammenarbeitet.

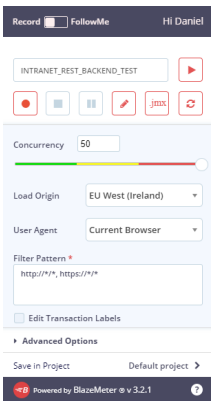

Abbildung 7: Blazemeter

4.2 Spezielle Software

Natürlich gibt es auch die Möglichkeit speziell für Recording entwickelte Software zu verwenden. Diese gibt es in Hülle und Fülle im Internet. Sowohl kostenpflichtig als auch Open-Source-Lösungen. Für Windows Nutzer gibt es beispielsweise die Software BadBoy. Unter http://www.badboy.

com.au kann man sich die neuste Version unter Angaben eines Namen und Email Adresse herunterladen und installieren. BadBoy bietet wie Blazemeter die Möglichkeit Abrufe bestimmter URLs aufzunehmen und abzuspeichern. In Abbildung 8 sieht man die Oberfläche von BadBoy, bei der gerade die API des Intranets aufgerufen und aufgezeichnet wird.

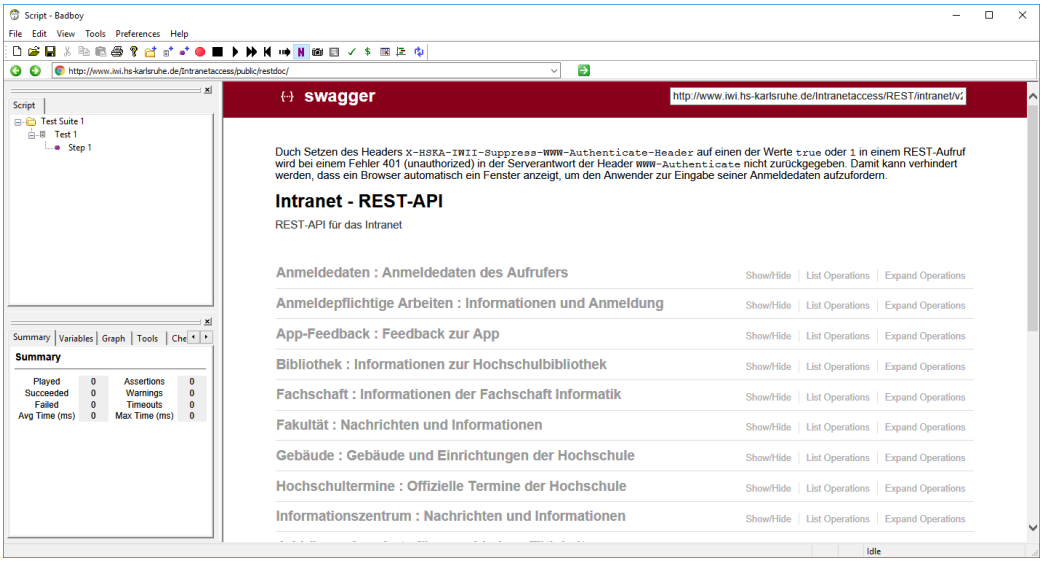

Abbildung 8: BadBoy Recording Software

4.3 Build-in Lösung

Für JMeter Enthusiasten gibt es einen integrierten Testscript-Recorder. Diesen findet man entweder beim Testplan unter folgendem Menüpunkt → Add → Non-Test-Elements → HTTP-Test-Script-Recorder oder über den Weg File → Templates... → Recording-Template → Create. In Abbildung 9 sieht man den erstellten Recorder. Falls man den Weg direkt über den Testplan gehen möchte, muss man zusätzlich eine Thread-Gruppe sowie einen Recording-Controller erstellen.

Nachdem der Port angepasst wurde klickt man auf Start und muss nun im Browser den entsprechenden ProxyServer konfigurieren. Als Adresse wählt man `localhost` und der Port entsprechend den JMeter Settings. Jede Aktion die nun im Browser getriggert wird, wird als Sampler im JMeter Testplan unter der entsprechenden Thread-Gruppe angelegt. Zusätzliche Cookies und Variable Konfigurationen werden automatisch angelegt.

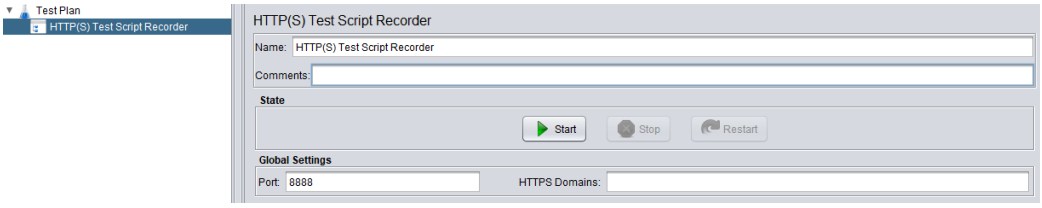

Abbildung 9: HTTP-Test-Script-Recorder

5 Lasttests von Webseiten

Nach dem Aufzeichnen der Ablauffolgen und der Sampler kann man nun einen ersten intensiven Lasttest starten. Es wird zunächst empfohlen, die Java Heap-Größe zu erhöhen, da der Standard 1GB Speicher für Lasttests ab einer gewissen Anzahl von Threads nicht ausreichend ist. Zum permanenten Ändern der Heap-Größe muss man eine zusätzliche Datei `setenv.sh` im `\bin` Ordner anlegen. Diese Datei wird ausgeführt, sobald man JMeter startet.

```
# Config file bin/setenv.sh
# Use bigger heap than the default
export HEAP="-Xms2G -Xmx2G"
```

Jetzt hat man statt den Defaultwerten einen 2GB großen Heap. Das erstellte Testskript wird als nächstes über die Kommandozeile mit folgendem Befehl gestartet:

```
jmeter -n -t IWI_PAGE.jmx -l testfolder/IWI_PAGE.csv -e -o
    testfolder/html_dashboard
```

Nach einer kurzen Rechenzeit sollte JMeter fertig sein und ein „end-of-run“ erscheinen. Siehe Abbildung 10. Es wird bereits während des Testens einiges an Informationen angezeigt, wie etwa aktive Threads, Fehlerhäufigkeit oder Dauer.

5.1 Result-File

Die interessanteren Ergebnisse befinden sich jedoch im zuvor angelegten Ordner „testfolder“. Mit dem Flag -l hat man dafür gesort, dass eine Datei IWI_PAGE.csv erzeugt wurde. Dieses sogenannte Result-File ist gleichbedeutend mit einem Listener aus dem Testplan und listet alle Anfragen

```
C:\Program Files\apache-jmeter-4.0\bin>jmeter -n -t IWI_PAGE.jmx -l testfolder/IWI_PAGE.csv -e -o testfolder/html_dashboard
Creating summariser <summary>
Created the tree successfully using IWI_PAGE.jmx
Starting the test @ Mon May 07 15:30:12 CEST 2018 (1525699812332)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +    216 in 00:00:17 =   12.6/s Avg:  2139 Min:   401 Max:  5217 Err:    32 (14.81%) Active: 245 Started: 245 Finished: 0
summary +    846 in 00:00:30 =   28.0/s Avg:  2502 Min:    12 Max:  9067 Err:   207 (24.47%) Active: 500 Started: 500 Finished: 0
summary =   1062 in 00:00:47 =   22.4/s Avg:  2428 Min:    12 Max:  9067 Err:   239 (22.50%)
summary +   2367 in 00:00:30 =   79.5/s Avg:   173 Min:    11 Max: 16921 Err:    16 (0.68%) Active: 500 Started: 500 Finished: 0
summary =   3429 in 00:01:17 =   44.5/s Avg:   871 Min:    11 Max: 16921 Err:   255 (7.44%)
summary +   1185 in 00:00:30 =   39.5/s Avg:    71 Min:    11 Max:  2622 Err:   366 (30.89%) Active: 500 Started: 500 Finished: 0
summary =   4614 in 00:01:47 =   43.1/s Avg:   666 Min:    11 Max: 16921 Err:   621 (13.46%)
summary +   1824 in 00:00:30 =   60.8/s Avg:   365 Min:    14 Max:  5317 Err:   177 (9.70%) Active: 500 Started: 500 Finished: 0
summary =   6438 in 00:02:17 =   47.0/s Avg:   580 Min:    11 Max: 16921 Err:   798 (12.40%)
summary +   1701 in 00:00:30 =   56.7/s Avg:  1073 Min:    12 Max:  8949 Err:   137 (8.05%) Active: 500 Started: 500 Finished: 0
summary =   8139 in 00:02:47 =   48.7/s Avg:   684 Min:    11 Max: 16921 Err:   935 (11.49%)
summary +   1975 in 00:00:30 =   65.4/s Avg:    82 Min:    11 Max:  2908 Err:   108 (5.47%) Active: 500 Started: 500 Finished: 0
summary =  10114 in 00:03:17 =   51.3/s Avg:   566 Min:    11 Max: 16921 Err:  1043 (10.31%)
summary +   1349 in 00:00:30 =   45.2/s Avg:    82 Min:    14 Max:  1496 Err:   392 (29.06%) Active: 250 Started: 500 Finished: 250
summary =  11463 in 00:03:47 =   50.5/s Avg:   509 Min:    11 Max: 16921 Err:  1435 (12.52%)
summary +    537 in 00:00:21 =   25.0/s Avg:    53 Min:    14 Max:  2661 Err:     0 (0.00%) Active: 0 Started: 500 Finished: 500
summary =  12000 in 00:04:09 =   48.3/s Avg:   489 Min:    11 Max: 16921 Err:  1435 (11.96%)
Tidying up ...    @ Mon May 07 15:34:21 CEST 2018 (1525700061490)
... end of run
```

Abbildung 10: JMeter Lasttest im Non-GUI-Mode

auf, die gesendet wurden, sowie zusätzliche Informationen. Ein kleiner Teil der Informationen ist in der folgenden Tabelle abgebildet. Eine Übersicht aller Befehle gibt es auf https://jmeter.apache.org/usermanual/listeners.html. Falls die Defaultwerte immer noch nicht ausreichen, hat man die Möglichkeit, die `jmeter.properties` Datei zu bearbeiten. Darin lassen sich weitere Werte hinzufügen.

Bezeichnung	**Erklärung**
elapsed	Benötigte Zeit für die Anfrage
bytes	Anzahl Bytes im Sampler
sentBytes	Anzahl der Bytes die man gesendet hat
grpThreads	Anzahl aktiver Threads in dieser Thread-Gruppe
allThreads	Anzahl aktiver Threads in allen Gruppen
latency	Zeitraum bis zur ersten Antwort
connect	Zeitraum um die Verbindung herzustellen

Tabelle 4: Informationen der CSV

Neben dem gängigen CSV-Format werden auch andere Formate, wie etwa JTL unterstützt. CSV hat allerdings den Vorteil, dass man es damit gut in Excel filtern und analysieren kann.

5.2 HTML-Dashboard

Mit dem Flag `-e -o testfolder/html_dashboard` lässt sich ein zweites tolles Feature erzeugen. Hier wird im „testfolder“ eine index.html generiert, die man sich im Browser betrachten kann. Die schön gestalteten Torten -und

Liniendiagramme geben sehr viel Auskunft über die einzelnen Aufrufe.

Auf der Dashboard Hauptseite gibt es Test und Reportinformationen mit Start- und Endzeit, verwendete Filter, sowie der Name des Result-Files, aus dem das Dashboard generiert wurde. Es gibt ein Tortendiagramm welches anzeigt, wieviel Prozent der Anfragen erfolgreich waren und wie viel schief liefen. Und es wird ein sogenannter Application-Performance-Index (APDEX) angezeigt. APDEX ist ein offener Standard, der die Leistung von Anwendungen misst. Hier wurde die durchschnittliche Dauer, einer Anfrage an einen bestimmten Endpunkt ermittelt. Anhand von vordefinierten „Toleration-treshold“ und „Frustration-treshold“-Werten, wird der Endpunkt dann als OK oder KO bewertet. Diese Werte lassen sich selbst definieren. Dazu muss man in das `\bin` Verzeichnis navigieren und Datei `user.properties` anpassen. Weiterhin gibt es Statistiktabellen und zwei Fehlertabellen für jeden Sampler.

Hinter der Kategorie Charts verbergen sich Diagramme, die nochmals in „Over-Time“, „Throughput“ und „Response-Times“ unterteilt sind. Wie die Namen vermuten lassen, kann man sich hier visualisierte Ergebnisse der einzelnen Samplern anzeigen lassen. Etwa über einen gewissen Zeitraum hinweg, den Datendurchsatz während der Testphase oder die Antwortzeiten. Abbildung 11 zeigt ein Liniendiagramm der prozentualen Verteilung von Antwortzeiten einzelner Sampler. Es ist auch möglich nur bestimmte Sampler in dem Diagramm, durch klicken der URL im unteren Bildrand, zu aktivieren.

Weiteres tolles Feature: Die Diagramme lassen sich alle direkt aus der HTML-Seite als PNG-Datei abspeichern und wiederverwenden.

6 Verteiltes Testen

Zunächst hört sich das Testen auf verteilten Systemen recht unkompliziert an. Man hat eine Master-Instanz von JMeter laufen und mehrere Slave-Instanzen. Über die Master-Instanz werden die Slaves gesteuert, gestartet und beendet [4]. Dadurch kann man völlig problemlos mehrere 1000+ Threads auf eine Webseite zugreifen lassen. So die Theorie. In der Praxis jedoch hat man mit einigen Problemen zu kämpfen:

- Verfügbarkeit von mehreren Servern

  Um die Performance Tests zu clustern, benötigt man natürlich mehrere Systeme. Da nicht jeder Anwender einen oder mehrere Server zu Hause stehen hat, geht man in der Regel den Weg in die Cloud

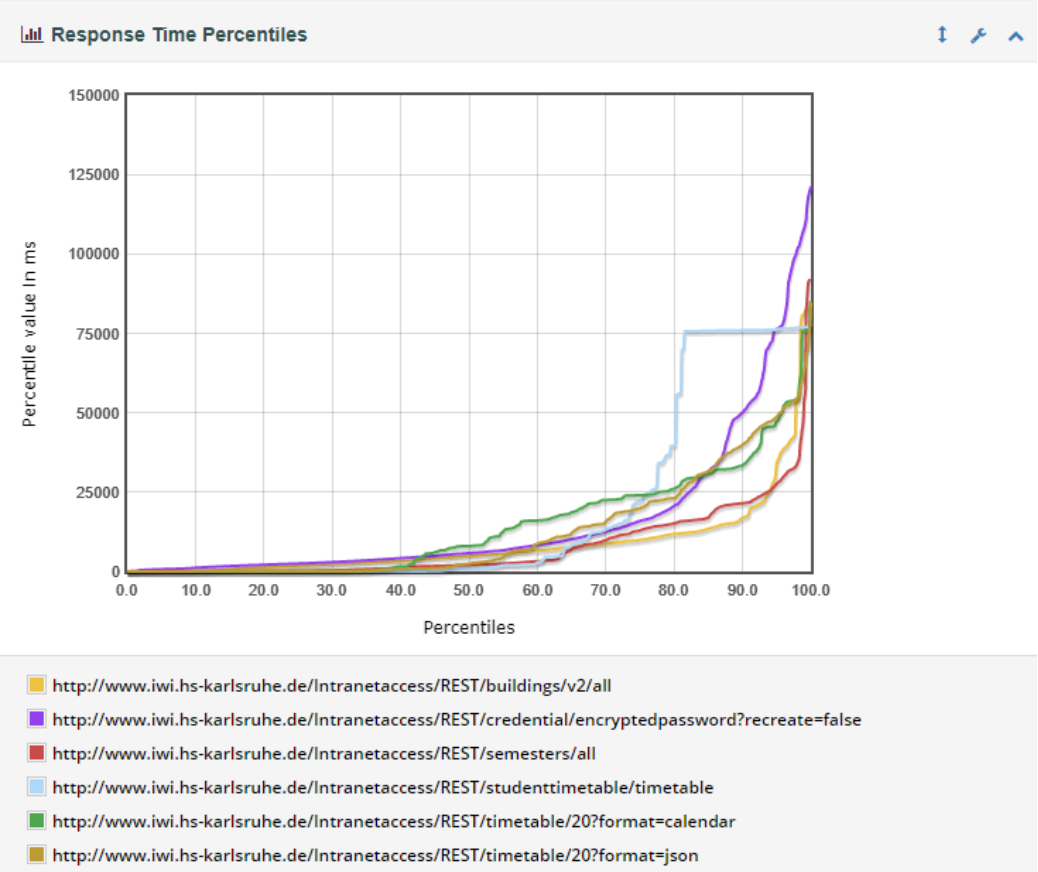

Abbildung 11: Dashboard-Diagramm der Antwortzeiten

und verwendet die dort zur Verfügung stehenden Resourcen, etwa EC2-Instanzen[3].

- Kommunikationsprotokoll von JMeter

  JMeter verwendet zur Kommunikation mit den Slave-Systemen das Java RMI (Remote-Method-Invocation)[4] und dazu müssen Master und Slave im gleichen Subnetz sein. Dies ist aber in Cloud Umgebungen nicht machbar. Unter [4] gibt es einen Workaround um genau die Subnetzproblematik zu umgehen.

- Aktive Firewall

[3] EC2 sind Compute-Units von Amazon in der Cloud https://aws.amazon.com/ec2/
[4] https://en.wikipedia.org/wiki/Java_remote_method_invocation

Falls man eine Firewall aktiviert hat, sollte man diese für Testzwecke kurz beenden oder sicherstellen, dass die JMeter Ports offen sind.

Für die Master-Instanz sollte sichergestellt sein, dass in der jmeter.properties Datei im `\bin` Ordner, die Zeile `remote_hosts=127.0.0.1` durch die entsprechenden Instanzen Komma separiert ersetzt wird. Die Master-Instanz führt lediglich das Starten der Slave-Instanzen aus und sollte keine Tests beeinhalten. Das Starten der Anwendungen geschieht dann über die Datei `jmeter-server.bat` aus dem `\bin` Verzeichnis.
Eine Ausführliche Anleitung zum verteilten Testen von JMeter findet man auch auf der offiziellen Apache Seite [5].

7 Testen der Datenbank

Apache JMeter bietet ebenfalls die Möglichkeit Lasttests auf Datenbankservern auszuführen. Dazu benötigt man lediglich entsprechende Zugangsberechtigung für die Datenbank und passende JDBC Treiber.
Ein kleines Beispiel zeigt einen Test auf eine lokale Postgres-Datenbank:

1. JDBC-Treiber

   Aktuelle JDBC-Treiber[5] für die Datenbank herunterladen und in das `\lib` Verzeichnis von JMeter kopieren.

2. Testplan und Thread-Gruppe anlegen

   In der Thread-Gruppe kann man nach einer erfolgreichen Verbindung zur Datenbank die Benutzeranzahl und gleichzeitige Verbindungen erhöhen.

3. JDBC-Connection-Configuration

   Auf der Thread-Gruppe rechtsklick → Add → Config-Element → JDBC-Connection-Configuration. Hier den Variablennamen für den Pool festlegen (myDatabase) und die Database-Connection-Configuration mit folgenden Werten füttern:

[5] https://jdbc.postgresql.org/download.html

Name	Wert
Database URL	jdbc:postgresql://hostname:ip/datenbankname
JDBC Driver class	org.postgresql.Driver
Username	username
Password	********

Tabelle 5: Postgres JDBC-Connection-Configuration

4. JDBC-Request-Sampler

   Auf der Thread-Gruppe einen JDBC-Request-Sampler adden und im Feld „Variable-Name-Bound-to-Pool“ den zuvor gewählten Variablennamen (myDatabase) eintragen. Im SQL-Query-Feld kann man nun ein entsprechendes Statement abfeuern um die Datenbank zu kontaktieren. Siehe Abbildung 12 In diesem Fall wurden die Spalten „id“ und „name“ Tabelle „country“ ausgewählt.

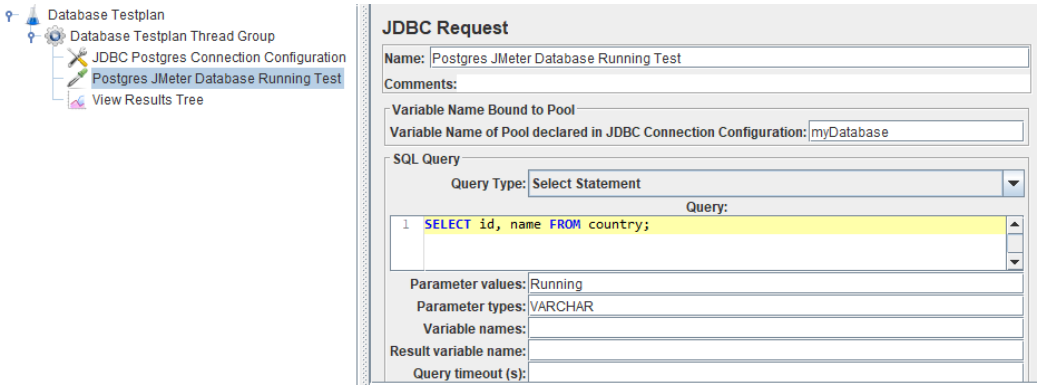

Abbildung 12: JDBC-Request-Sampler

5. View-Results-Tree

   Das Ergebnis der Anfrage lässt sich mit einem Listener, etwa View-Results-Tree anzeigen. Abbildung 13 zeigt die erfolgreiche Antwort mit drei gefundenen Ländern.

Im nächsten Schritt würde man nun die Anzahl der Benutzer und Anfragen in der Thread-Gruppe erhöhen und anpassen. Beispielsweise 50 Benutzer (Threads), zwei Anfragen und 100 Wiederholungen. Macht insgesamt 10000 JDBC-Requests und würde die Datenbank deutlich mehr belasten.

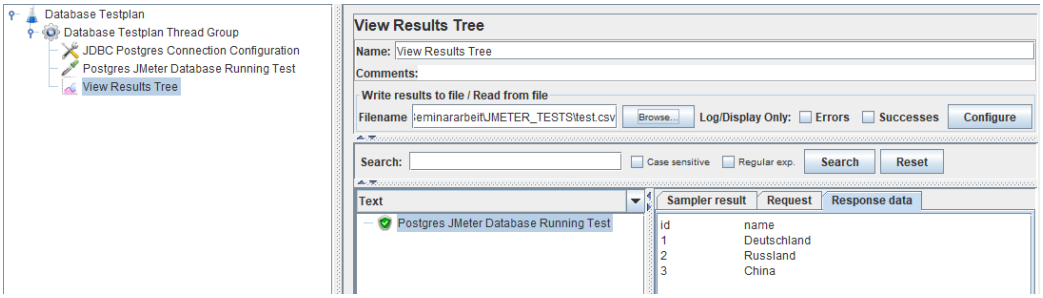

Abbildung 13: JDBC-Request-Results-Tree

8 JUnit-Tests

Ab Version 2.1.2 ist es möglich JUnit-Sampler auszuführen und damit auch JUnit-Tests zu untersuchen. Doch warum sollte man die altbewährte JUnit-Tests der IDE durch angepasste Jmeter Tests ergänzen? Es bietet sich an, durch die Möglichkeiten von JMeter ganz bestimmte Lasten und Zeiten auf individuelle Tests zu setzen und dadurch deutlich mehr Umfang im Test zu erzielen. Zunächst sollte eine JUnit Testklasse vorhanden sein. Wichtig ist hier, dass sowohl alle Testmethoden, als auch die Testklasse selbst als „public“ deklariert werden, da diese sonst nicht gefunden werden können. Abbildung 14 zeigt zwei Testmethoden. Die `addTest()` Methode schlägt fehl, die `quadTest()` Methode läuft erfolgreich durch.

```
public class CalculatorTest {

    Calculator c = new Calculator();

    @Test
    public void addTest() {
        assertEquals("Result does not match", 150, c.add(10, 10)); //$NON-NLS-1$
    }

    @Test
    public void quadTest() {
        assertEquals("Result does not match", 81, c.quad(9)); //$NON-NLS-1$
    }

}
```

Abbildung 14: JUnit-Tests von Calculator

Diese Klasse muss man als JAR-File exportieren und in das Verzeichnis `\lib\junit` kopieren. Jetzt kann man in JUnit einen JUnit-Request-Sampler anlegen. Darin kann man Einstellungen vornehmen wie „Success-Messages“,

„Success-Codes“ und Andere. Wenn jetzt eine Testmethode aus der Testklasse ausgewählt und gestartet wird kann man diese wiederum in einem Listener anzeigen lassen und analysieren (Siehe Abbildung 15).

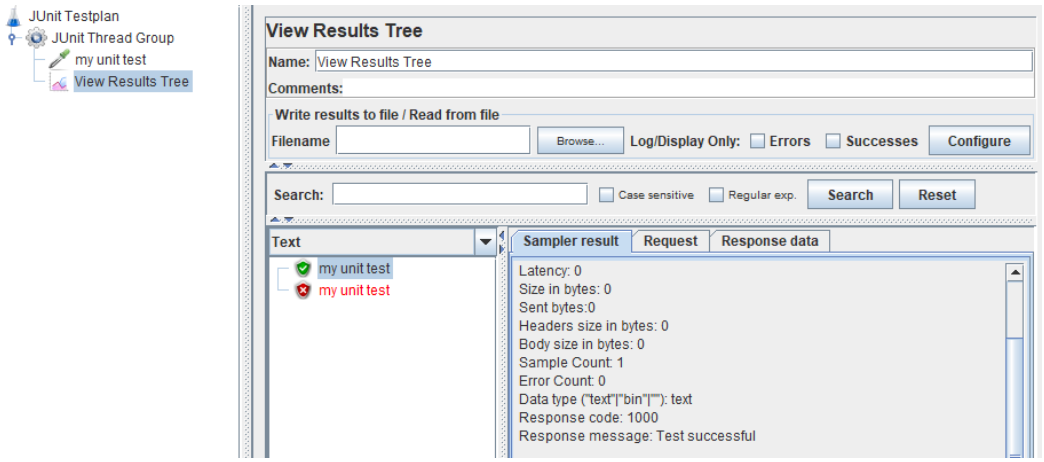

Abbildung 15: JUnit View Results Tree

9 Wissenswertes und Besonderheiten

Dieses Kapitel gibt einen kleinen Exkurs in erweiterte Funktionen und Besonderheiten, die Apache JMeter zu bieten hat.

9.1 Java Code in JMeter verwenden

Wenn man die Funktionalität von JMeter um eigene Methoden erweitern möchte, gibt es BeanShell-Sampler [2]. BeanShell ist eine leichtgewichtige Skriptsprache für Java, welche die JMeter API nutzen kann und das darunter liegende Java-SDK. Abbildung 16 zeigt eine zuvor in Java erstellte Klasse Calculator welche in JMeter bei einem Testplan als JAR hinzugefügt wurde und die Methoden daraus in der Beanshell verwendet werden.

Das Ergebnis der zwei Berechnungen sieht man in dem Log-Viewer-Panel. Falls dieser nicht aktiviert ist, genügt ein Klick auf das gelbe Dreieck mit dem Ausrufezeichen in der rechten oberen Ecke.

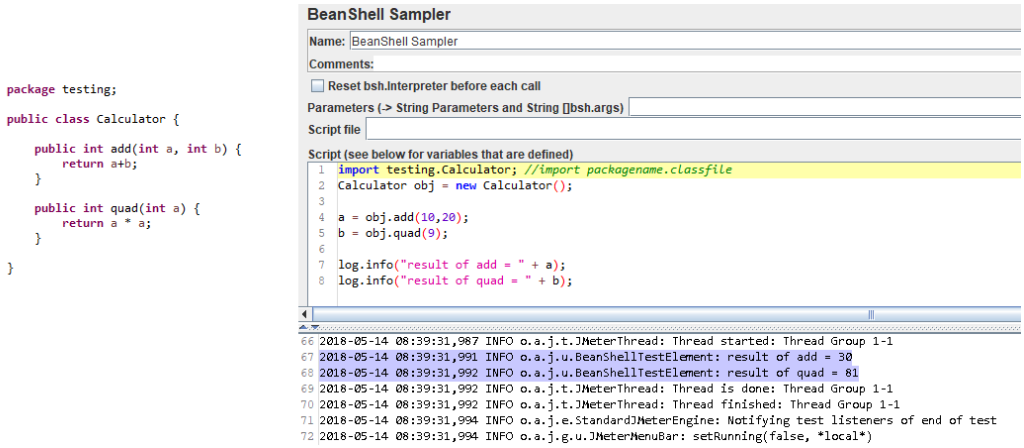

Abbildung 16: Verewndung von Java in Beanshell

9.2 JMeter-Plugin-Manager

Erwähnenswert ist der Plugin-Manager von JMeter. Anstatt sich diverse Plugins manuell zu installieren und sich mit unterschiedlichen Versionen und JARs herumzuärgern, übernimmt der Plugin-Manager die Installation, Deinstallation und Upgradeverwaltung für den Benutzer. [11]

Unter Options → Plugin-Manager gelangt man in die Erweiterung. Dort kann man oben genannten Aktionen ausführen. Um eine Übersicht der JMeter Erweiterungen zu erhalten sollte man die Webseite https://jmeter-plugins.org/ besuchen.

9.3 Extrahieren von Daten aus einer Anfrage

Sehr hilfreich sind auch die Post-Processor-Extraktoren. Diese werden nach der Anfrage ausgeführt und können zusätzliche Informationen in Form von XPath -und Regular-Expressions aus der Antwort herauslesen und etwa in JMeter-Variablen speichern. Die Sampler hierfür befinden sich unter → Post-Processors → Regular-Expression-Extractor / XPath-Extractor.

Ein praxisbezogenes Beispiel wäre hier das herauslesen einer serverseitig vergebenen ID für ein neu angelegtes Objekt, welches man dann im Folgeaufruf erneut verwenden kann [14].

10 Nachteile von JMeter

Neben der Vielzahl von positiven Features gibt es natürlich auch einige Mängel, die JMeter mit sich bringt.

- Konfigurationsfehler

  Wenn ein Variablenfeld in den JMeter-Config-Elementen (Z.B. JDBC-Connection-Configuration) „required“ ist, erkennt man dies nicht direkt im Feld, sondern erst beim Ausführen.

- Passwörter

  Ein großer sicherheitskritischer Punkt ist, dass eingegebene Passwörter bei Recorder im Klartext in der Config Datei gespeichert werden (Siehe Abbildung 17). Diese sollten so schnell wie möglich durch entsprechende Variablen, etwa ${USER} und ${PASSWORD}, ersetzt werden. In diesen Variablen muss man die entsprechenden Werte verschlüsselt hinterlegen (Stichwort Base64). Gerade bei JMeter-Tests in Cloud Umgebungen ist es enorm wichtig, dass die Klartextwerte nicht einsehbar sind.

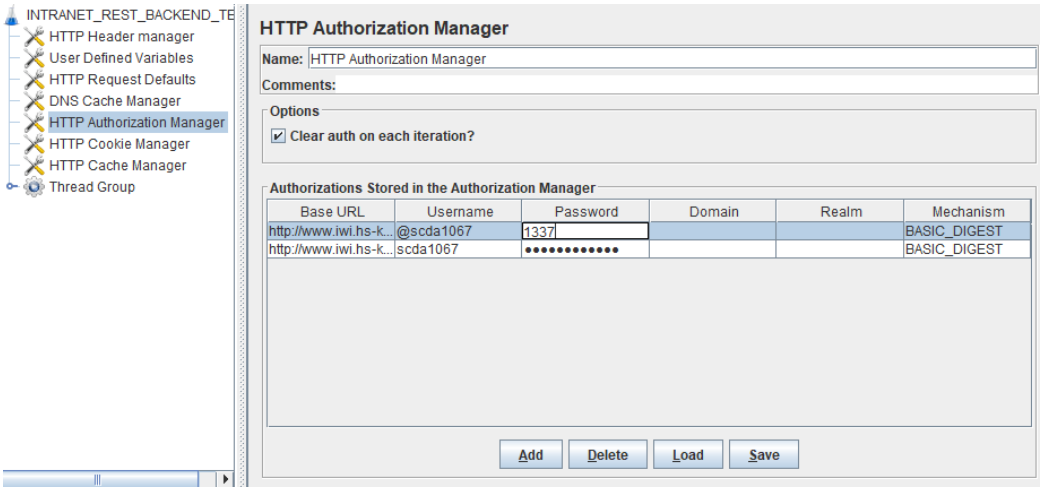

Abbildung 17: Speichern von Passwörter im Klartext

- Resourcenbedarf

  Wie in der Einführung bereits erwähnt, ist Apache JMeter im GUI Modus sehr ressourcenhungrig. Selbst auf einer Maschine mit 16GB Arbeitsspeicher hat sich die Oberfläche hin und wieder aufgehängt

und reagierte nicht mehr. Falls man JMeter über die Kommandozeile verwendet, was bei größeren Lasttests auch empfohlen wird, hat man jedoch weniger Probleme und kann diesen Punkt getrost ignorieren.

- Captcha-Erkennung

  Eine Captcha-Abfrage dient dazu, eine „reale“ Person von einem Bot oder Computerprogramm zu unterscheiden. Dementsprechend trickreich sind die Captcha-Abfragen aufgebaut. Für JMeter ist es daher auch nicht möglich diese Captchas als Programm zu erkennen. Man sollte daher die Captcha-Erkennung serverseitig entweder deaktivieren oder ein vordefiniertes Captcha mit der entsprechenden Antwort bereitstellen, auf das im Test zurückgegriffen wird.

11 Alternativen

Alternative Last- und Performancetest Produkte auf dem Markt sollen natürlich nicht unerwähnt bleiben. Die folgenden drei gelisteten Tools mit entsprechender URL sind neben JMeter relativ bekannt und können bei Bedarf weiter untersucht werden:

- Gatling https://gatling.io/
- Selenium https://www.seleniumhq.org/
- The Grinder http://grinder.sourceforge.net/

12 Fazit

Durch eine Vielzahl von Literatur und Handbüchern, auch auf der offiziellen Apache Webseite, fällt einem der Einstieg in die Welt von Performance-Tests sehr leicht und durch die flache Lernkurve bleibt man stets motiviert sich weiter mit dem Thema zu beschäftigen.

Um Anwendungen und insbesondere HTTP-Server bezüglich ihrer Performance zu testen, ist JMeter trotz des hohen Resourcenverbrauchs, ein hervorragendes Tool, welches eine große Menge an Möglichkeiten der Konfiguration des Testes bietet. Sowohl die Laststeuerung als auch Anfragensteuerung ermöglichen aussagekräftige Ergebnisse.

Tolle Visualisierungsmöglichkeiten der Ergebnisse, etwa durch den HTML-Dashboard, können dazu verwendet werden um aussagekräftige und verständliche Reports zu kommunizieren und Schwachstellen und Engpässe im System zu erkennen und entgegenzusteuern.

Durch eine breite Protokollunterstützung wie HTTP, FTP, JDBC oder das Aufrufen von Java-Klassen, kann man nicht durch typische Webanwendungen, sondern sogar eine gesamte Systeminfrastruktur testen.

Zum ausführlichen Testen von JMeter in der Cloud fehlten leider die Ressourcen, um eine aussagekräftiges Fazit geben zu können. Alle Cloud Dienste die zur Auswahl standen, etwa EC2 oder Blazemeter, waren kostenpflichtig und konnten nicht getestet werden.

Im Vergleich zu Unit oder Integrationstests fristen Last- und Performance Tests in vielen Unternehmen, aufgrund von Zeit oder Geldmangel immer noch ein Nischen-Dasein. Dort sind Lasttests eher ein wildes unkontrolliertes umherklicken von mehreren Mitarbeitern. Diese Tests können jedoch nicht reproduziert werden und haben keine Aussagekraft bezüglich Performance-Engpässen der Software oder Webanwendungen.

Man kann sich nur erhoffen, dass sich diese Sichtweise über längere Zeit ändert und die Firmen wieder verstärkt auf diesen Sektor eingehen. So hat man etwa bei kritischen Situationen und Stoßzeiten, die Gewissheit, dass die Webanwendung den Ansturm vieler Nutzer meistert und die Kundenzufriedenheit dadurch gewährleistet ist.

Literatur

[1] Tim Bardenhagen. Performance-Tool JMeter. http://www.fh-wedel.de/~si/seminare/ws02/Ausarbeitung/9.perftools/perftool2.htm#3, 2002. (Zugriff am 07.05.2018).

[2] BeanShell. Beanshell - lightweight scripting for java. http://beanshell.org/, 2018. (Zugriff am 14.05.2018).

[3] Rishil Bhatt. Setup and Teardown Thread Group in Jmeter. http://www.testingjournals.com/setup-teardown-thread-group-jmeter/, 08 2016. (Zugriff am 07.05.2018).

[4] Marcel Folaron. Performance Testing in the Cloud with JMeter & AWS. https://www.artofsoftwaredevelopment.com/performance/performance-testing-in-the-cloud-with-jmeter-aws. (Zugriff am 07.05.2018).

[5] The Apache Software Foundation. Apache JMeter - Apache JMeter Distributed Testing Step-by-step. https://jmeter.apache.org/usermanual/jmeter_distributed_testing_step_by_step.html, 2018. (Accessed on 05/07/2018).

[6] The Apache Software Foundation. Apache JMeter - User's Manual: History/Future. http://jmeter.apache.org/usermanual/history_future.html, 2018. (Zugriff am 26.04.2018).

[7] The Apache Software Foundation. Apache JMeter - What can I do with it? https://jmeter.apache.org, 2018. (Zugriff am 23.04.2018).

[8] Hauke Gierow. Amazon, Spotify, Twitter, Netflix: Mirai-Botnetz legte zahlreiche Webdienste lahm. https://www.golem.de/news/ddos-massiver-angriff-auf-dyndns-beeintraechtigt-github-und-amazon-1610-123966.html, 10 2016. (Zugriff am 23.04.2018).

[9] Emily H. Halili. *Apache JMeter*. Packt Publishing, Birmingham, 1 edition, 2008.

[10] Frank Pientka. Last- und Performance-Tests mit JMeter oder Gatling. https://www.heise.de/-3648505, 03 2017. (Zugriff am 23.04.2018).

[11] Andrey Pokhilko. Documentation - JMeter-Plugins.org. https://jmeter-plugins.org/wiki/PluginsManager/, 2018. (Zugriff am 14.05.2018).

[12] Waldemar Siebert. Vorteile der Testautomatisierung in der Praxis. https://www.testautomatisierung.org/welche-vorteile-bietet-die-testautomatisierung-in-der-praxis/, 07 2015. (Zugriff am 25.04.2018).

[13] SwissQ Consulting AG Waldemar Erdmann. Last- und Performancetest - Teil 3: Die Messkurven führen uns zu den Flaschenhälsen. https://swissq.it/de/testing/last-und-performancetest-teil-3-die-messkurven-fuehren-uns-zu-den-flaschenhaelsen/, 02 2014. (Zugriff am 23.04.2018).

[14] Björn Weinbrenner. Einen automatisierten Lasttest mit JMeter durchführen. https://www.bjoerne.com/einen-automatisierten-lasttest-mit-jmeter-durchfuehren/, 09 2013. (Zugriff am 14.05.2018).

[15] Wikipedia. Swing (Java). https://de.wikipedia.org/w/index.php?title=Swing_(Java)&oldid=171402620, 12 2017. (Zugriff am 25.04.2018).

[16] Wikipedia. Akzeptanztest (Softwaretechnik). https://de.wikipedia.org/w/index.php?title=Akzeptanztest_(Softwaretechnik)&oldid=176319126, 2018. (Zugriff am 23.04.2018).