DomainDrivenArchitecture
Lasttest für Webapplikationen

Author(en): Michael Jerger
Version 0.3 vom 03.07.2012
Status: Gültig

Änderungsnachweis

Version	Autor / Aktor	Datum	Änderung / Aktivität	Status
0.1	M. Jerger	21.06.2012	Initiale Erstellung	Gültig
0.2	M.Mörike, T. Wieger, T. Frech, M.Jerger	25.06.2012	Datenauswertung hinzugefügt; Feedback von M.Mörike, T. Wieger und T.Frech eingearbeitet	Gültig
0.3	M.Jerger	03.07.2012	Kostenbetrachtung, Erfahrungen und vollständige Messdaten hinzugefügt.	Gültig

Legende

Neuer Status	Konsequenz
Entwurf	Inhalt nach Ansicht der Autoren noch nicht vollständig.
Gültig	Der Inhalt ist nach Ansicht der Autoren vollständig und richtig
Abgestimmt (mit → Aktor)	Der Inhalt ist auch nach Meinung weitere Personen gültig
Freigegeben (durch → Aktor)	Der Auftraggeber / Verantwortliche erklärt den Inhalt für gültig

Inhaltsverzeichnis

1 Rahmenbedingung für dieses Dokument

1.1 Anlass und Ziel

Sobald ein Lasttest für ein produktionsnahes System nicht nur ein prinzipielles „tut“ / „tut nicht“ liefern soll, steigt die Wahrscheinlichkeit einen Fehler zu machen schnell an. Leider steigt damit aber auch der Testaufwand rasant.

Das vorliegende Dokument beschreibt Ansätze und die Erfahrungen damit, um den ein oder anderen erfolglosen Testlauf zu vermeiden.

1.2 Zielgruppe

Die Zielgruppe für dieses Dokument sind jeweils erfahrene Tester, Software-Entwickler, IT-Betreuer, IT-Architekten und IT-Projektmanager.

1.3 Umfang und Abgrenzung

In diesem Dokument werden Webanwendungen im Java-Umfeld betrachtet.

1.4 Ressourcen und weiterführende Informationen

In diesem Dokument gezeigte Beispiele zur Logfile Transformation und die verwendeten Lasttest-Scripts finden Sie unter https://www.domaindrivenarchitecture.org zum Download.

2 Beteiligte Systeme

An einem Lasttest sind meist viele Systeme beteiligt. Die folgende Abbildung zeigt eine beispielhafte Anordnung.

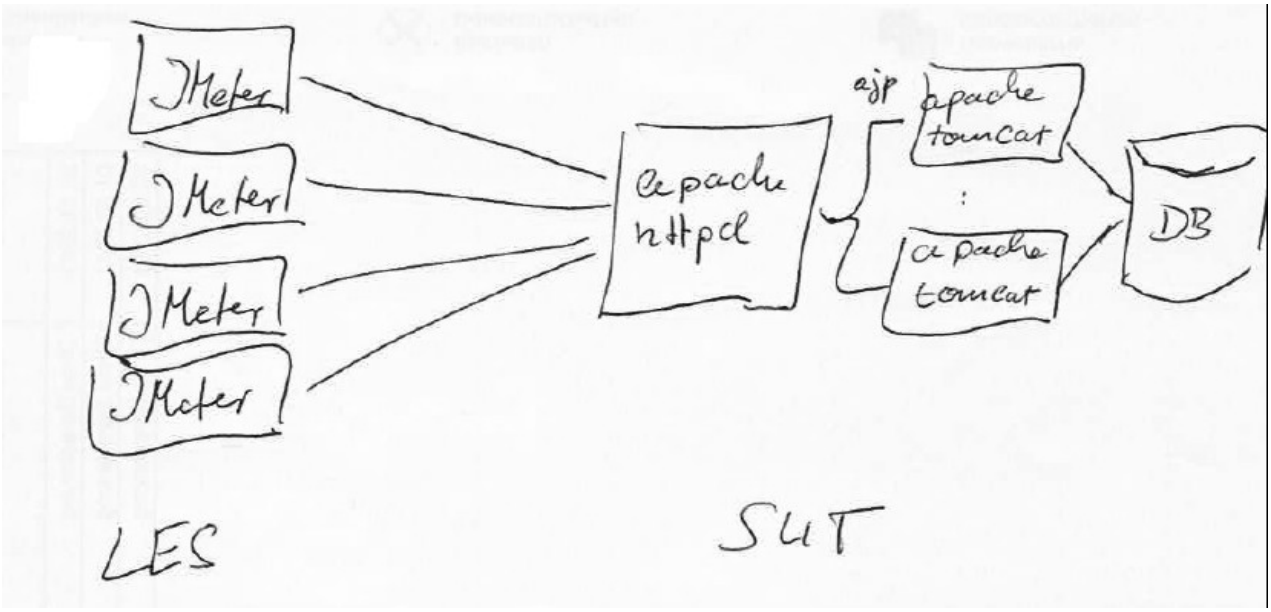


Abbildung 1: Die am Lasttest beteiligten Systeme

2.1 Last-Erzeugendes-Systeme (LES)

Die Last-Erzeugenden Systeme simulieren Benutzerinteraktion für viele parallele Benutzer. Je nach dem, für wie viel Last des zu testende System ausgelegt ist, werden hier unterschiedlich viele Instanzen zur Last-Erzeugung benötigt.

2.2 System-Unter-Test (SUT)

Im verwendeten Beispiel besteht das zu Testende System aus einem Webserver (apache httpd) und mehreren geclusterten Applikationsservern (apache tomcat).

Der Webserver ist in dieser Konstellation für die Verteilung der Last und für die robuste Begrenzung der Zugriffe in Überlastsituationen zuständig.

3 Anforderungen an den Lasttest

3.1 F1: Der Test bestätigt oder widerlegt die spezifizierte Performance

Die Performance wird im Umfeld von Webanwendungen oft als

- ausgelieferte Seiten / Zeit
- maximale Antwortzeit
- maximale Anzahl von parallel arbeitende Benutzer / Seiten / Requests

definiert. Weitere perfomancerelevante Anforderungen kommen typischerweise aus dem IT-Betrieb.

3.2 NF1: Der Test ist wiederholbar

Das bedeutet, alle für den Test relevanten Größen sind so beschrieben, dass der Test ein zweites mal durchgeführt werden kann. Die Resultate sind dabei ähnlich (genug).

3.3 NF2: Der Test unterstützt Analysen und Voraussagen für Lastzuwachs

Oft ergibt ein Lasttest, dass an der ein oder anderen Ecke die Performance noch optimiert werden muss. Der Lasttest soll die dazu nötige Analyse unterstützen.

3.4 NF3: Der Test testet das SUT

Das ist auf den ersten Blick eine triviale Anforderung. Aber schon in der Abbildung 1 „Die am Lasttest beteiligten Systeme„ ist ersichtlich, dass die LES Systeme zwar parallel Last erzeugen können, diese Last aber nur dann am SUT ungebremst ankommt, wenn die Netzverbindungen ausreichend dimensioniert sind. In dieser Art sind noch viele weitere Details zu beachten.

4 Performance Spezifikation

Um die unter „3.1 F1: Der Test bestätigt oder widerlegt die spezifizierte Performance„ angegebene Performance zu spezifizieren werden viele verschiedenen Größen angegeben – meist sind die folgenden Gruppen mit jeweils eigenen Vorstellungen vertreten:

- Kunden,
- den IT-Betrieb und
- die Tester.

4.1 Performance aus Kundensicht

4.1.1 Seiten pro Tag

Die Größe „Seiten pro Tag“ tauchen in vielen Webstatistiken auf und ist daher eine allgemein verständliche Größe zur Last-Spezifikation. Da die Abschätzung pro Tag recht grob ist, versuche ich zumindest auf eine Größe „Seiten pro Stunde“ zu präzisieren.

4.1.2 Maximale Ladezeit einer Seiten

Eine weitere gängige Größe ist „eine Seite soll innerhalb von 1,5 Sekunden geladen werden“.

4.1.3 Parallel arbeitende Benutzer

Wird spezifiziert, wie viel Benutzer parallel arbeiten können sollen, dann ist wieder eine Abschätzung nötig: Wie viele Seiten lädt ein typischer Benutzer pro Zeit?

Damit kann dann ebenfalls auf Parallel geladene Seiten umgerechnet werden.

4.2 aus IT-Betriebssicht

Der IT-Betrieb definiert Performance naturgemäß noch deutlich genauer, meist mit den folgenden Rahmenbedingungen und Kenngrößen:

- Speicherverbrauch,
- CPU-verbrauch,
- Disk- und Netz-IO,
- maximale Requestgöße,
- Disk-Speicherbearf und
- Robustheit des Systems.

4.2.1 Systemanforderung: Robustheit

Eine weitere typische IT-Betriebsanforderung ist die „Robustheit“ eines Systems – klar, wer möchte schon dauernd für Notfälle aus dem Schlaf gerissen werden. Robustheit bedeutet dabei im Kontext eines Lasttests:

- Das System bewältigt die spezifizierte Last, auch über lange Zeit.

- Bei Überlast regeln die zuständigen Komponenten zuverlässig den Verkehr ab und ermöglicht den bestehenden Verbindungen ein fehlerfreies Arbeiten. In hier betrachteten Beispiel übernimmt der Webserver diese Aufgabe.
- Durch Störung des Netzwerks schaukelt sich das System nicht auf.
- Nach Lastrückgang kommt das System wieder in den Ausgangszustand zurück.

4.3 aus Sicht der Last-Tester

Mit den oben genannten Größen kann ein Last-Tester meist noch nicht wo wirklich arbeiten. Wichtige Größen in der Sprache eines Testers sind

- Requests parallel
- Requests pro Sekunde (Durchsatz)
- Requestgrößen
- Mit dem System interagierende Benutzer pro Zeit

4.4 Überlegung zur Größen-Umrechnung

Im Kern muss immer die Größe Seite / Zeit in parallele Requests pro Zeit umgerechnet werden. Für diese Umrechnung helfen die folgenden Überlegungen.

4.4.1 Browser Caching

Alle aktuellen Browser bieten Caches für häufig benutzte Ressourcen. Ob Caching für Ihren Test relevant ist hängt von mehreren Faktoren ab, wie

- wie viel Seiten Rufen Benutzer in Ihrer Webapplikation in Folge auf,
- wie viel Ressourcen benötigen die getesteten Seiten,
- erlaubt die Webanwendung Caching.

Zum Testen unter der Berücksichtigung von Caching gibt es drei Strategien:

- Caching komplett ignorieren und alle Ressourcen einer Seite laden. Damit hat die Webanwendung bei bestandenem Test sicher noch Puffer.
- Das Caching von Hand einschätzen. Ein Workaround für einfache Testtools.
- Ein Lasttest-Tool verwenden, das auch Caching simulieren kann.

4.4.2 Requests pro Seite

Wie viele Requests benötigt eine Seite der Webanwendung XY?

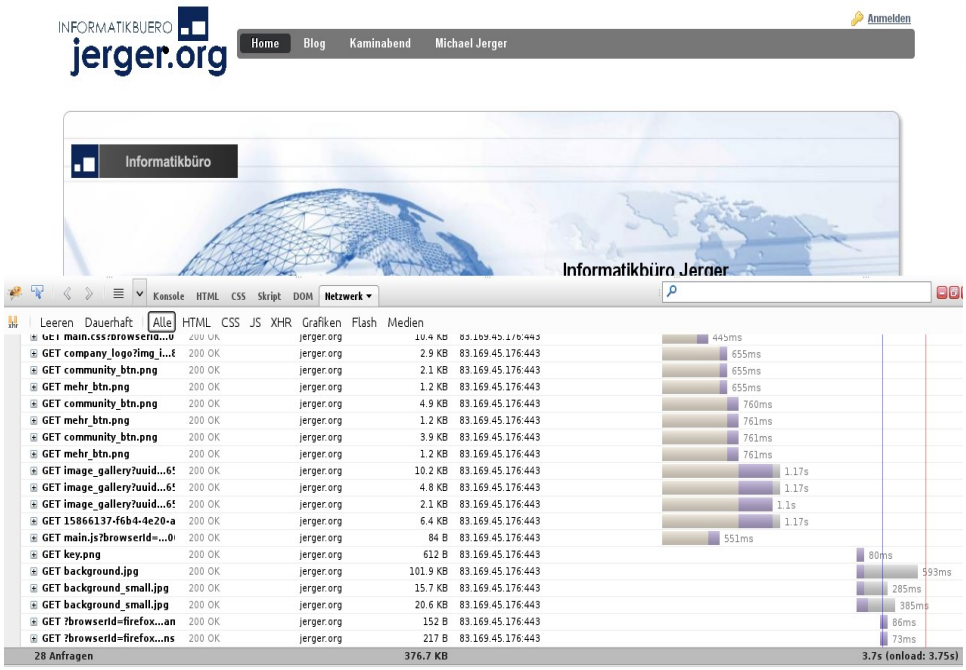


Abbildung 2: Für diese Seite werden 28 Anfragen benötigt

Das Web-Entwicklungstool Firebug bietet hier wertvolle Hilfe. Schauen Sie sich die Seiten Ihres Testszenarios an und bekommen Sie ein fundiertes Gefühl (wenn ein Gefühl nicht ausreicht können Sie natürlich auch eine genaue Statistik erstellen) für die benötigten Request.

4.4.3 Request parallel pro Browser

Ein zweiter Effekt, der sich in Firebug schon erahnen lässt, Browser stellen nur eine gewisse Anzahl von Requests parallel (beim aktuellen Firefox13 sind das 6). Unter http://www.browserscope.de finden Sie eine sehr gute Übersicht über alle möglichen Browserparameter – in diesem Fall interessieren uns die „Connections per Hostname“.

Top Browsers name	score	PerfTiming	Connections per Hostname	Max Connections	|| Script Script	|| Script Stylesheet	|| Script Image	|| Script Iframe	Async Scripts	|| CSS	|| CSS + Inline Script	Cache Expires
Chrome 18 →	13/16	yes	6	21	yes	yes	yes	no	yes	yes	yes	yes
Firefox 11 →	14/16	yes	6	28	yes	yes	yes	no	yes	yes	yes	yes
Your Test Results ↓												
Firefox 13.0	14/16	yes	6	25	yes	yes	yes	no	yes	yes	yes	yes
IE 8 →	7/16	no	6	35	yes	yes	no	no	no	yes	no	yes
IE 9 →	12/16	yes	6	35	yes	yes	yes	no	no	yes	yes	yes
Opera 11 →	7/16	no	6	32	no	no	no	no	no	yes	no	yes
RockMelt 0.9 →	13/16	yes	6	35	yes	yes	yes	no	yes	yes	yes	yes
Safari 5.1 →	12/16	no	6	35	yes	yes	yes	no	yes	yes	yes	yes
Chrome 19 →	12/16	yes	6	17	yes	yes	yes	no	yes	yes	yes	yes
Chrome 20 →	12/16	yes	6	16	yes	yes	yes	no	yes	yes	yes	yes
Firefox Beta 13 →	14/16	yes	6	40	yes	yes	yes	no	yes	yes	yes	yes
IE Platform Preview 10 →	10/15		2	35	yes	yes	yes	no	no	yes	yes	yes

Abbildung 3: Übersicht über aktuelle Browsereigenschaften: http://www.browserscope.org

5 Der Lasttest

Ein Lasttest, der all den oben genannten Anforderungen gerecht werden soll, ist naturgemäß etwas komplizierter. Daher stellt sich recht schnell die Frage, ob es Sinn macht, alle Aufgaben in einen großen Lasttest zu packen oder statt dessen viele kleine modulare Lasttests zu verwenden.

Meiner Erfahrung nach verursachen bei einem Lasttest die Vorbereitung, Synchronisation und Verwaltung der vielen beteiligten Systeme bei jedem Teststart einiges an Mühe.

Daher tendiere ich dazu, möglichst alle Anforderungen mit einem Test abzudecken. Das führt im Gegenzug aber auch dazu, dass der eine große Test erst nach mehreren Anläufen bis zum Ende erfolgreich durchläuft. Wie Sie sehen, hat also auch der vorgeschlagene Weg durchaus Schwächen.

Die Bestandteile des Lasttests im Einzelnen (in der Annahme, dass 50 Benutzer parallel arbeiten können sollen):

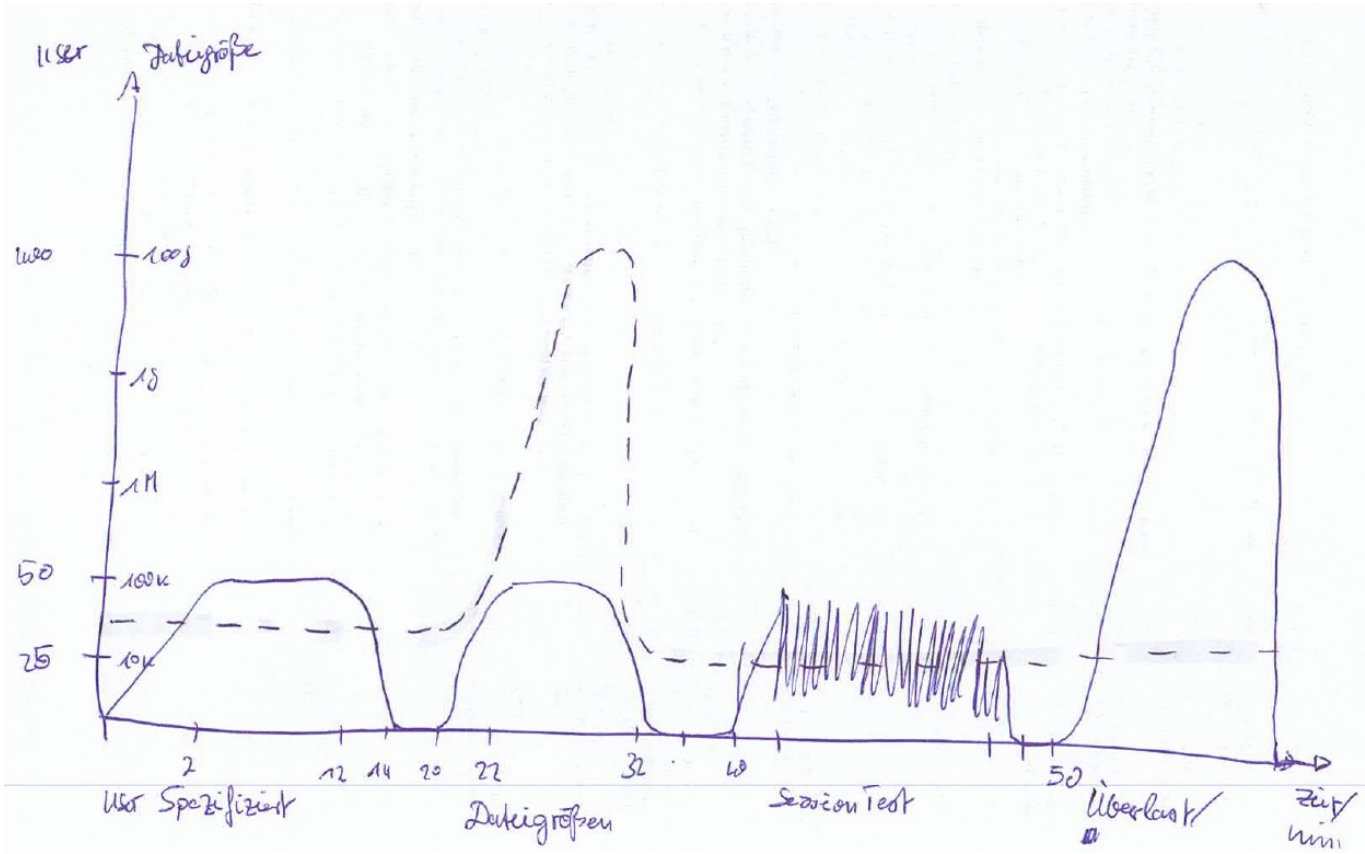


Abbildung 4: Ein Lasttest mit vier Phasen

Dieser Lasttest teilt sich dabei in vier Phasen auf:

5.1 Spezifiziert

In diesem Testabschnitt werden die maximal parallel spezifizierten Benutzer (oder genauer die umgerechnete maximale Anzahl paralleler Requests) getestet.

Die RampUp- und RampDown-Zeit betragen jeweils 2 Minuten.

Alle Tests werden mit durchschnittlichen Seiten und mit normaler (File-) Größe durchgeführt.

Als Ergebnis wird erwartet, dass

1. die Tests fehlerfrei durchgeführt werden können,
2. die Seiten-Antwortzeit eingehalten wird und
3. das System in der Ruhepause wieder den Normalzustand erreicht.

5.2 Dateigrößen

In dieser Testphase werden gezielt große Down- und wenn passend auch Up-loads getestet. Diese Testphase findet mit der spezifizierten Anzahl von Nutzern statt.

Als Ergebnis wird erwartet, dass

1. die Tests fehlerfrei durchgeführt werden können und
2. das System in der Ruhepause wieder zu seinem Normalzustand zurückfindet.

5.3 Hit and Run

In der Phase Session-Test besuchen viele Benutzer (im Schnitt immer die spezifizierte Menge) die Webanwendung ganz kurz und „sehen sich nur eine Seite an“.

Erwartete Ergebnisse sind:

1. Die Tests führen zu keinen Fehlern,
2. die spezifizierte Antwortzeit für eine Seite wird eingehalten und
3. das System kehrt in der Ruhepause wieder zu seinem Normalzustand zurück.

5.4 Überlast

In der letzten Phase „Überlast“ werden bewusst massiv mehr Requests auf das SUT abgeschickt. Die Erwartungshaltung hierbei ist, dass

1. die das SUT weiterhin Anfragen fehlerfrei ausführen kann (zumindest die, die angenommen werden),
2. die zuständige Komponente die Überlast (in diesem Szenario der apache httpd) abfängt und
3. das System in der Ruhepause wieder zu seinem Normalzustand zurückfindet.

6 Statische Messgrößen

Die Anforderung „NF1: Der Test ist wiederholbar“ führt direkt dazu, dass der Kontext des Lasttests möglichst genau beschrieben werden muss – für alle beteiligten Systeme.

In virtualisierten Betriebsumgebungen fällt es allerdings recht schwer, die Rahmenbedingungen exakt festzuhalten. Was für den jeweiligen Einsatzzweck exakt genug ist, muss in diesem Fall jeder für sich entscheiden.

6.1 LES

	Hardware LES	
	Bei Virtualisierung: VMGuest	Bei Virtualisierung: VMHost
Betriebssystem		
Speicher		
CPU		
Netzverbindungen		
Disk-Größe		-
Disk-IO-Benchmark		-
VM Hersteller		
VM-Benchmark		
	Software LES (z.B. für JMeter)	
Java Version		
JMeter Version		
Java-VM Konfiguration		

Tabelle 1: Kontext der LES Systeme

6.2 SUT

6.2.1 Webserver (z.B. apache httpd)

Hardware Webserver		
	Bei Virtualisierung: VMGuest	Bei Virtualisierung: VMHost
Betriebssystem		
Speicher		
CPU		
Netzverbindungen		
Disk-Größe		-
Disk-IO-Benchmark		-
VM Hersteller		
VM-Benchmark		
Software Webserver (z.B. apache httpd)		
Apache httpd Version		
maxclients		
timeouts		
ajp-connectoren		
LimitRequestBody		
LoadBalancing		

Tabelle 2: Kontext des Webservers

6.2.2 Applikationsserver (z.B. apache tomcat)

Hardware Applikationsserver		
	Bei Virtualisierung: VMGuest	Bei Virtualisierung: VMHost
Betriebssystem		
Speicher		
CPU		
Netzverbindungen		
Disk-Größe		-
Disk-IO-Benchmark		-
VM Hersteller		
VM-Benchmark		
Software Applikationsserver		
Java Version		
Java-VM Konfiguration		
Tomcat Version		
MaxThreads		
Timouts		
DB-Connectoren		
Cluster-Synchronisation		

Tabelle 3: Kontext der Applikationsserver

6.2.3 Datenbankserver (z.B. DB2)

Hardware Datenbankserver		
	Bei Virtualisierung: VMGuest	Bei Virtualisierung: VMHost
Betriebssystem		
Speicher		
CPU		
Netzverbindungen		
Disk-Größe		-
Disk-IO-Benchmark		-
VM Hersteller		
VM-Benchmark		
Software Datenbankserver		
DB Version		
DB Konfiguration		

Tabelle 4: Kontext des Datenbankservers

7 Dynamische Messgrößen

Nach den statischen Messgrößen stellen die dynamischen Daten den deutlich interessanteren Teil des Lasttests dar. Da ein Lasttest selten im ersten Wurf gelingt, kommt der Anforderung „NF2: Der Test unterstützt Analysen und Voraussagen für Lastzuwachs“ eine wichtige Rolle zu.

Für die Analyse ist es dabei essentiell, dass die Messdaten aus den beteiligten Systemen korreliert werden können. Für die Korrelation bieten sich zwei Größen an:

- Nach Zeit: Die Korrelation nach Zeit macht für alle Rquest-unabhängigen Messgrößen Sinn.
- Nach Request: Die Webkomponenten können alle mit http Requests umgehen. Hier ist es sinnvoll, einen Request durchgängig durch alle Systeme zu verfolgen. So ist ersichtlich, welche Komponenten wie viel Zeit benötigen.

7.1 Nach Zeit

Als Voraussetzung für eine Korrelation noch Zeit ist natürlich, dass alle Systeme eine synchronisierte Systemzeit verwenden.

Im beschriebenen Beispiel werden die folgenden Datenquellen für eine Korrelation nach Zeit verwendet:

7.1.1 LES

- Systemmessungen mit SAR (CPU, Speicherverbrauch, DiskIO, NetzIO) – siehe 11.4.1 Logging für Linux-Systeme.
- GC-Logg der JavaVM (falls Jmeter zur Last-Erzeugung verwendet wird) – siehe 11.4.2 Logging für Java-VMs
- Kumulierte Anzahl der Requests pro Sekunde.

7.1.2 SUT – Webserver

- Systemmessungen mit SAR (CPU, Speicherverbrauch, DiskIO, NetzIO) – siehe 11.4.1 Logging für Linux-Systeme.
- Kumulierte Zeit, die in diesem System für Requests verbraucht wurden pro Sekunde.

7.1.3 SUT – Applikationsserver

- Systemmessungen mit SAR (CPU, Speicherverbrauch, DiskIO, NetzIO) – siehe 11.4.1 Logging für Linux-Systeme.
- GC-Logg der JavaVM für den eingesetzten Tomcat, Jboss, o.ä. – siehe 11.4.2 Logging für Java-VMs
- Kumulierte Zeit, die in diesem System für Requests verbraucht wurden pro Sekunde.

7.2 Pro Request

Wenn für einen Request bekannt ist, in welchen Komponenten wie viel Zeit verbraucht worden ist hilft das bei der schnellen Analyse.

Um hier den Messfehler möglichst gering zu halten kommen die folgenden Prinzipien zum Einsatz:

- Die gesamte Request/Response Time wird gemessen (beinhaltet auch verbrauchte Zeit in nachgelagerten Systemen). Zeitanteile von nach gelagerten Messpunkten müssen dann subtrahiert werden.
- Die Messung erfolgt in den gemessenen Systemen jeweils möglichst nahe an den Systemgrenzen.

7.2.1 Request-ID

Um die einzelnen Requests durch alle beteiligten Systeme verfolgen zu können, werden im LES eindeutige Request-Id's als URL Parameter Injiziert (z.B. **https://www.jerger.org/blog?testId=12346**).

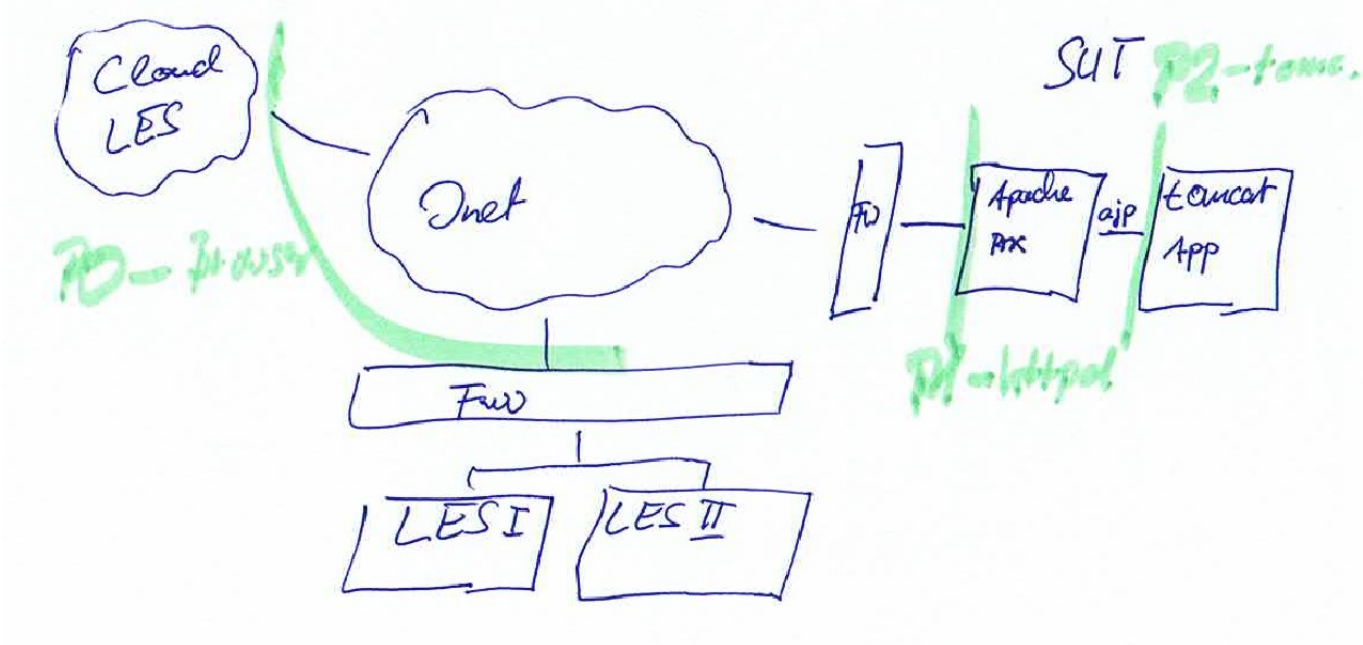
Im Browser sieht das dann so aus:

Im Webserver:
84.156.133.155 - - [08/Jun/2012:17:03:17 +0200] "GET **/blog?testId=12346** HTTP/1.1" 200 6932 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:13.0) Gecko/20100101 Firefox/13.0" 544369

Im Applikationsserver:
84.156.133.155 - - [08/Jun/2012:17:03:17 +0200] "GET **/blog?testId=12346** HTTP/1.1" 200 6932 544 33156AB922EB1CB0E93776B408CCAD92

7.2.2 Messpunkte

Im vorgestellten Beispiel bieten sich die folgenden Messpunkte an:



- **P0_Browser**: Hier wird vom Testsystem die gesamte Zeitspanne zwischen dem Request und dem Eintreffen der Response gemessen.
- **P1_httpd**: Im Punkt P1 wird die Zeit gemessen, die für ein Request/Response Zyklus im Webserver und Applikationsserver gemeinsam verbraucht wird.
- **P2_tomcat**: Im Punkt P2 wird schließlich die verbrachte Zeit im Applikationsserver gemessen.

Bei Bedarf können weitere Messpunkte hinzugefügt werden. Zum Beispiel an der Schnittstelle zur Datenbank oder an signifikanten Stellen in der Webanwendung.

8 Messergebnisse

Die Messergebnisse eines Lasttests benötigen mehrere Schritte der Veredlung, bis sich ein aussagekräftiges Diagramm aus den Daten generiert werden kann. Mir ist es übrigens noch nie geglückt, mit nur einem Lasttest zu guten Messwerten zu kommen – daher stelle ich mich immer darauf ein, die folgenden Schritte mehrfach zu durchlaufen.

8.1 Ablage

Alle Logfiles eines Lasttests sollten an einer Stelle versioniert und zusammengeführt werden – das ist trivial. Da Lasttests aber durchaus auch zu Systemabstürzen führen können, sollten die Logfiles auch während dem Test laufend gesichert werden.

Denn was kurz vor einem Systemabsturz passiert ist ist sicher sehr interessant.

8.2 Normalisierung

In diesem Schritt werden alle Daten aus den Logfiles extrahiert und in die gleichen Einheiten gebracht (oder auch in die gleiche Zeit – falls nicht alle Syteme synchron waren).

Ich verwende nach vielen frustrierenden Versuchen mit Monitoring und Log-AnalyseTools dafür inzwischen die BI Lösung Pentaho.

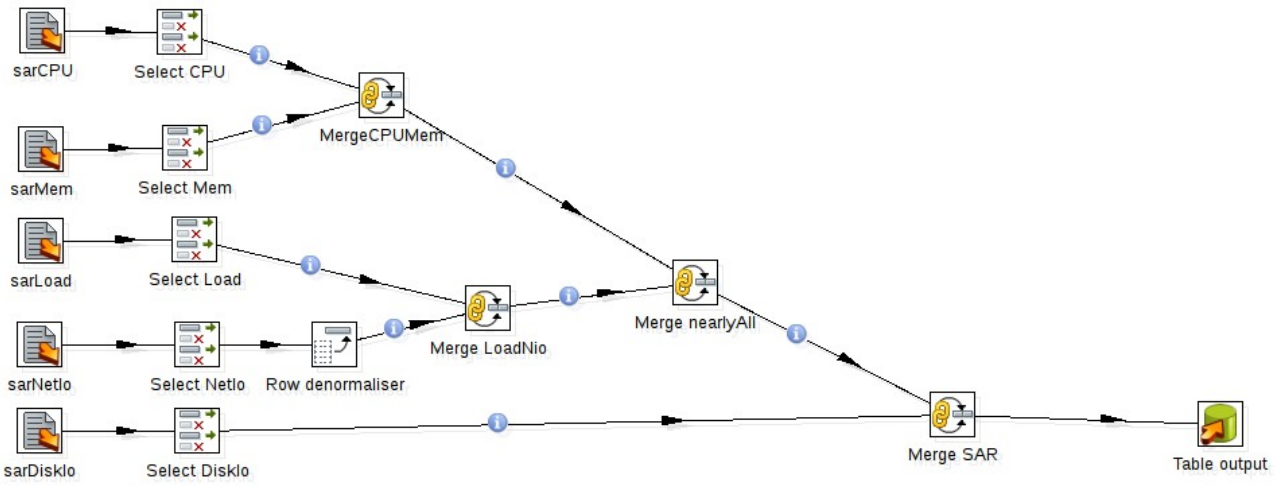


Abbildung 5: Transformationsschritte um aus den einzelnen Sar-Files wieder zusammenhängende Information zu machen.

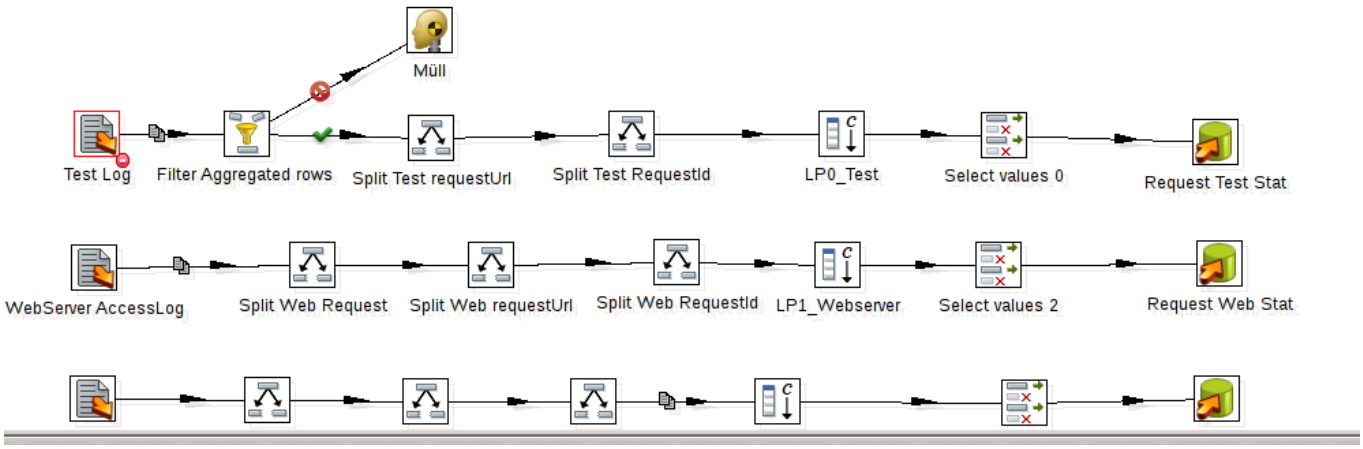


Abbildung 6: Logfiles aus den Web-Systemen Test-, Web- und Anwendungs-Systemen

Abbildung 7: Und noch das GC-Log - im Moment noch mit manuellen Vorarbeiten ...

All diese Daten werden von Pentaho in eine Datenbank geschrieben. In SQL lassen sich dann sehr einfach die benötigten Daten richtig kumulieren und weiterverarbeiten.

8.3 Darstellung

Hier finden Sie die Auswertung zu dem exemplarischen Test über alle vier Test-Phasen:

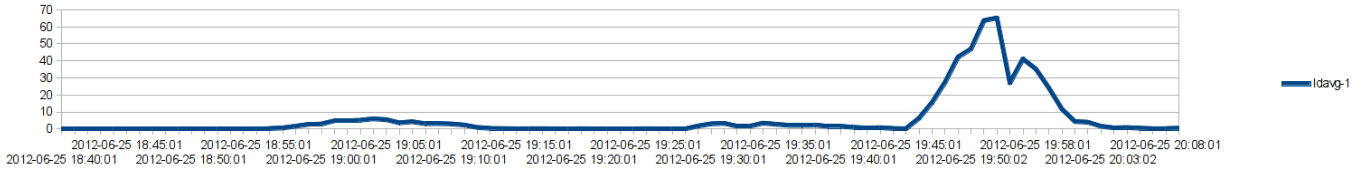


Abbildung 8: SUT-Load

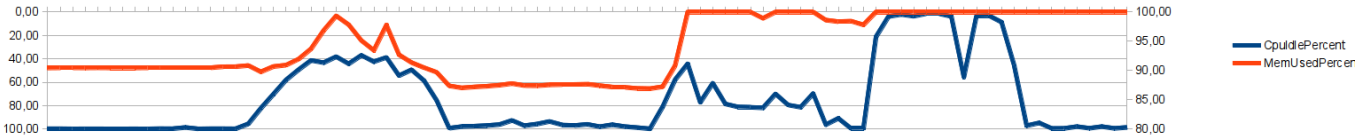


Abbildung 9: SUT: CPU und Memory

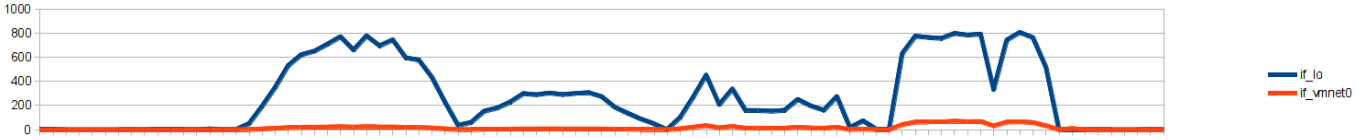


Abbildung 10: SUT Netztraffik

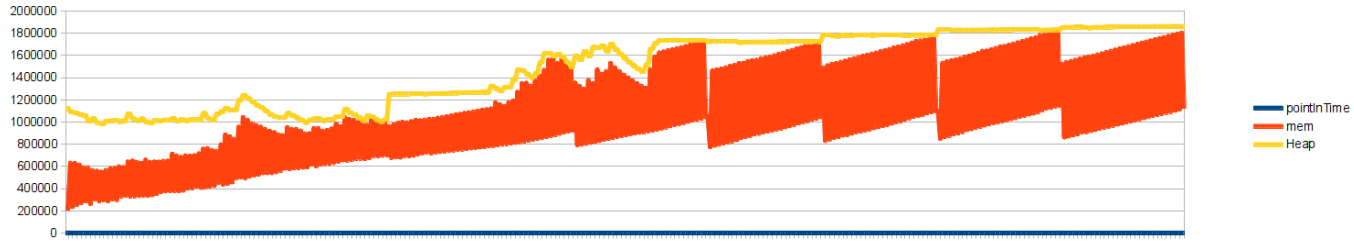


Abbildung 11: WebApp GC-Log

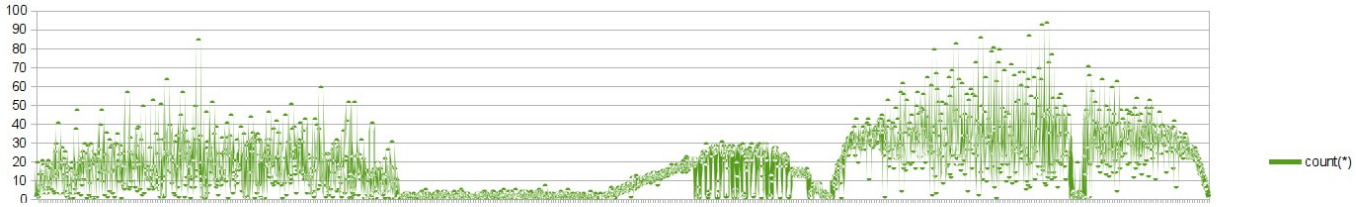


Abbildung 12: Lasttest - Request pro Sekunde

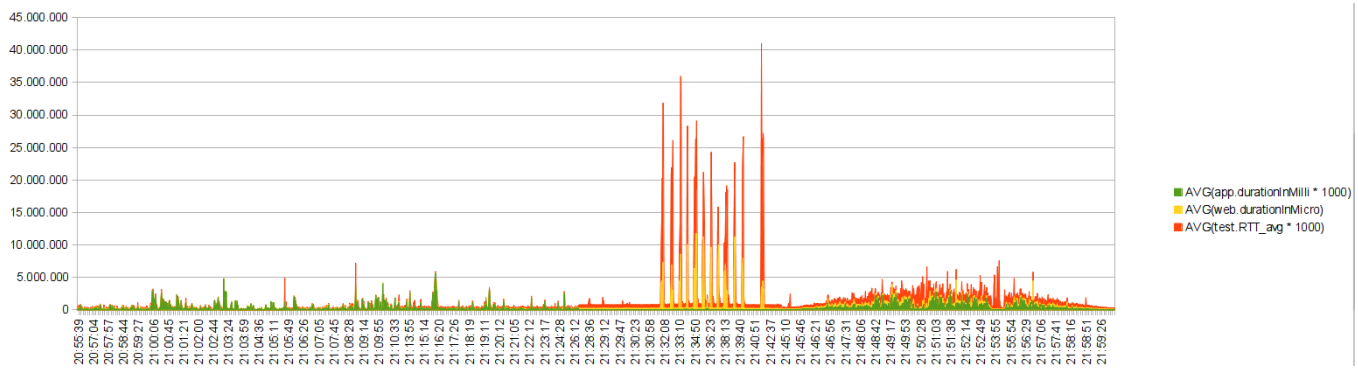


Abbildung 13: Durchschnittlicher Zeitverbrauch pro Sekunde

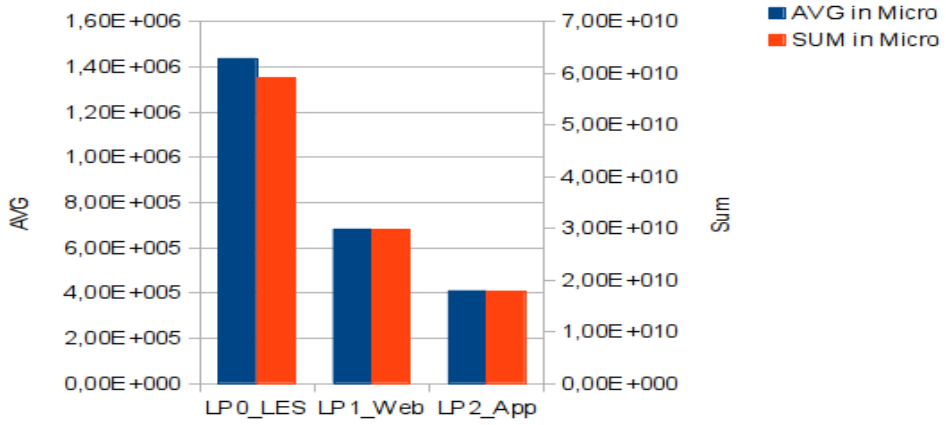


Abbildung 14: Verteilung von Summe und Durchschnitt pro Messpunkt

8.4 Analyse & Bewertung

Wie immer birgt auch dieser einfache Lasttest jede Menge Überraschungen und wirft die folgenden Fragen auf:

8.4.1 Performance-Einbruch in der Überlastphase

Nach kurzer Recherche wird klar, hier sind die File-Handles ausgegangen. Ein typischer Server-Konfigurations-Fehler.

8.4.2 Knicke während des Überlast-Phasen RampUps

Zeigen hier den Punkt des maximalen Durchsatzes an.

8.4.3 Latenzzeit in Hit&Run

In der Hit&Run Phase habe ich von einem Standort aus den USA getestet – die daraus resultierende Latenzzeit kann man unschwer erkennen. Man sieht aber auch, wie sich eine solche Latenzzeit auf die Server-Last auswirkt.

Hier wird also deutlich, dass Durchsatz sehr deutlich vom beteiligten Client abhängt.

9 Kosten eines Lasttests

Die Kosten eines Lasttest lassen sich recht schnell berechnen. Ich habe hier zwei typische Szenarien gegenübergestellt, das erste Szenario repräsentiert den hier vorgestellten Test, das zweite einen deutlich komplexeren Test einer Web-Appliance.

		Test www.jerger.org	Test Appliance
Charakteristik		Loadimpact, 3 Request-Messpunkte	JMeter, 6 Request-Messpunkte, Benutzeranmeldung
Lasttest entwickeln		1 PT	7 PT
pro Last-test		2,1 PT	3,9 PT
	Lasttest durchführen	0,5 PT	1,5 PT
	Logs sammeln	1 PT	0,2 PT
	Logs korrelieren	0,1 PT	0,2 PT
	Analyse	0,5 PT	2 PT

Abbildung 15: Exemplarische Aufwände

Über die reinen Aufwände hinaus entstehen noch weitere Kosten durch die bereitzustellende Hardware. Diese Kosten können durch Loadimpact deutlich reduziert werden – der vorgestellte Lastest liegt im Bereich von 30,- Euro (Stand 2012).

Was allerdings sehr deutlich zu Buche schlägt ist die Tatsache, dass Kosten pro Lasttest entstehen – im folgenden Kapitel sind Situationen zusammengetragen, die alle einen weiteren Lasttest nötig machen.

10 Erfahrungen

In diesem Kapitel sind Situationen zusammengetragen, die die Frage „darfs noch ein Test sein“ mit einem gequälten Ja beantworten lassen.

10.1 Optimierungs-Zwang beim SUT

Beim vorgestellten Test muss hier z.B. die Anzahl der File Handles vergrößert werden.

10.2 Log-Files rotieren / löschen sich

LogFiles rotieren und haben eine Maximalgröße. Mit etwas Pech passt der gesamte Lasttest nicht in Íhre aktuelle Konfiguration.

10.3 Festplatte läuft voll

Sie haben alle Logfiles und Puffer groß genug Dimensioniert, aber nicht auf den Verfügbaren Paltz geachtet?

10.4 Netzverbindung ist zu dünn

Oder die Firewall ist nicht performant genug?

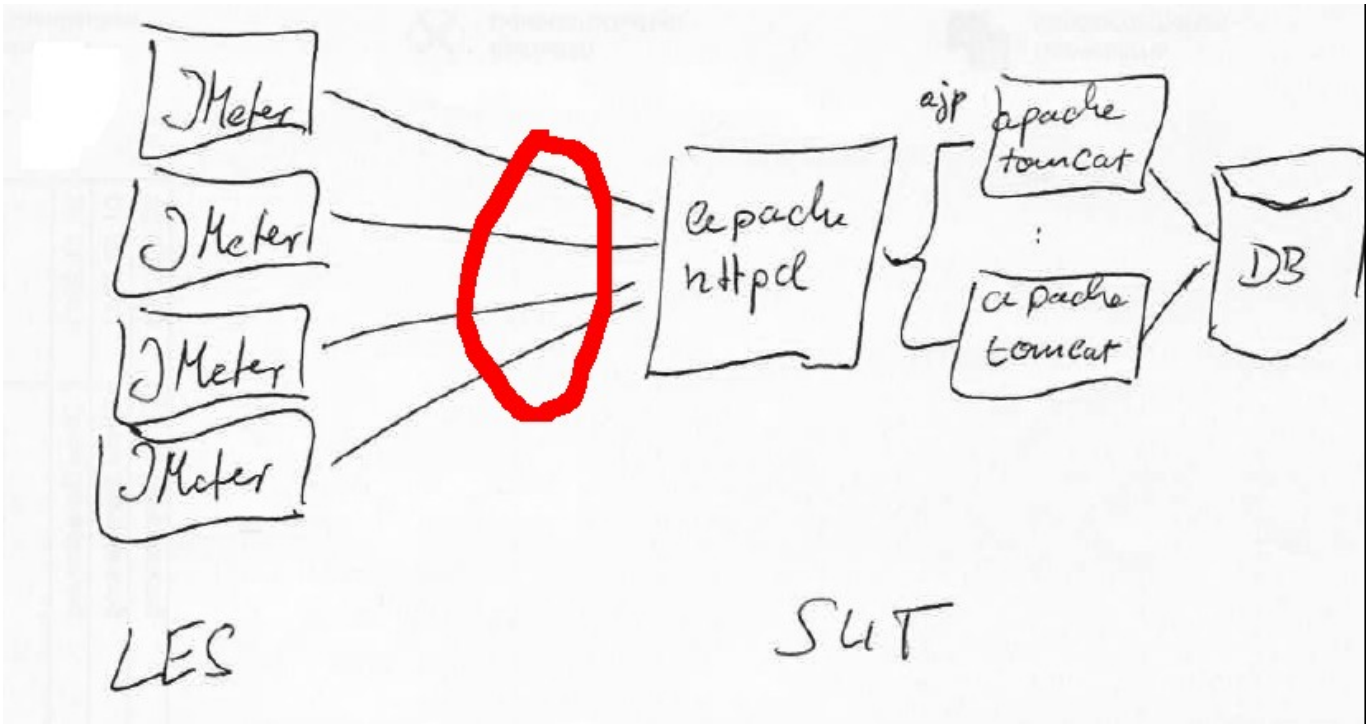


Abbildung 16: Hier ist das Netzwerk der Flaschenhals, nicht das SUT

10.5 Fehler im Test-Script

Selbst sorgfältig entwickelte oder angepasste Testscripts sind oft nicht fehlerfrei. So z.B.

- funktioniert der parallelisierte Test nicht, wie erwartet oder
- Id's werden nicht erwartungsgemäß erzeugt.

10.6 Fehler in integrierten SUT Systemen

SOA bedeutet auch dass Webapplikationen nicht mehr monolithisch und abgeschlossene Systeme sind, sonder viele weiteren Systeme integrieren. Ein Lasttest einer Webapplikation kann jetzt natürlich sehr leicht auch zu Fehlern in den integrierten Systemen des SUT führen (z.B. Benutzer-Anmeldung, Auftragsverwaltungs-System, o.ä.).

10.7 Datentyp Konvertierung

Nachdem sehr viele System an einem Lasttest beteiligt sein können, können all diese Systeme auch unterschiedlichst formatierte Daten abliefern. Die Umwandlung von Datentypen ist daher nicht so einfach, wie auf den ersten Blick gedacht:

- Punkt oder Komma: 0.15 oder 0,15
- „0“, „-“ oder „null“
- LocalZone oder GMT: 25/Jun/2012:02:01:30 +0200
- 25/Jun/2012:00:01:30
- Nachkommastellen: 0.0966070 sec
- Schei Encoding …

10.8 Sar nicht richtig konfiguriert

Im Default schreibt SAR die System-Kennzahlen nur alle 10 Minuten auf. Ein 30' Lasttest kann dann durchaus so aussehen:

```
Hostname;             interval;timestamp;                 ldavg-1;      ldavg-5;      ldavg-15
www.jerger.org;       600;    2012-06-15 18:05:01 UTC;    0.22;         0.14;         0.05
www.jerger.org;       600;    2012-06-15 18:15:01 UTC;    0.04;         0.10;         0.08
www.jerger.org;       601;    2012-06-15 18:25:01 UTC;    1.15;         0.42;         0.19
www.jerger.org;       599;    2012-06-15 18:35:01 UTC;    4.76;         5.61;         3.23
www.jerger.org;       600;    2012-06-15 18:45:01 UTC;    0.03;         0.80;         1.73
www.jerger.org;       600;    2012-06-15 18:55:01 UTC;    0.00;         0.11;         0.89
```

11 Eingesetzte Tools

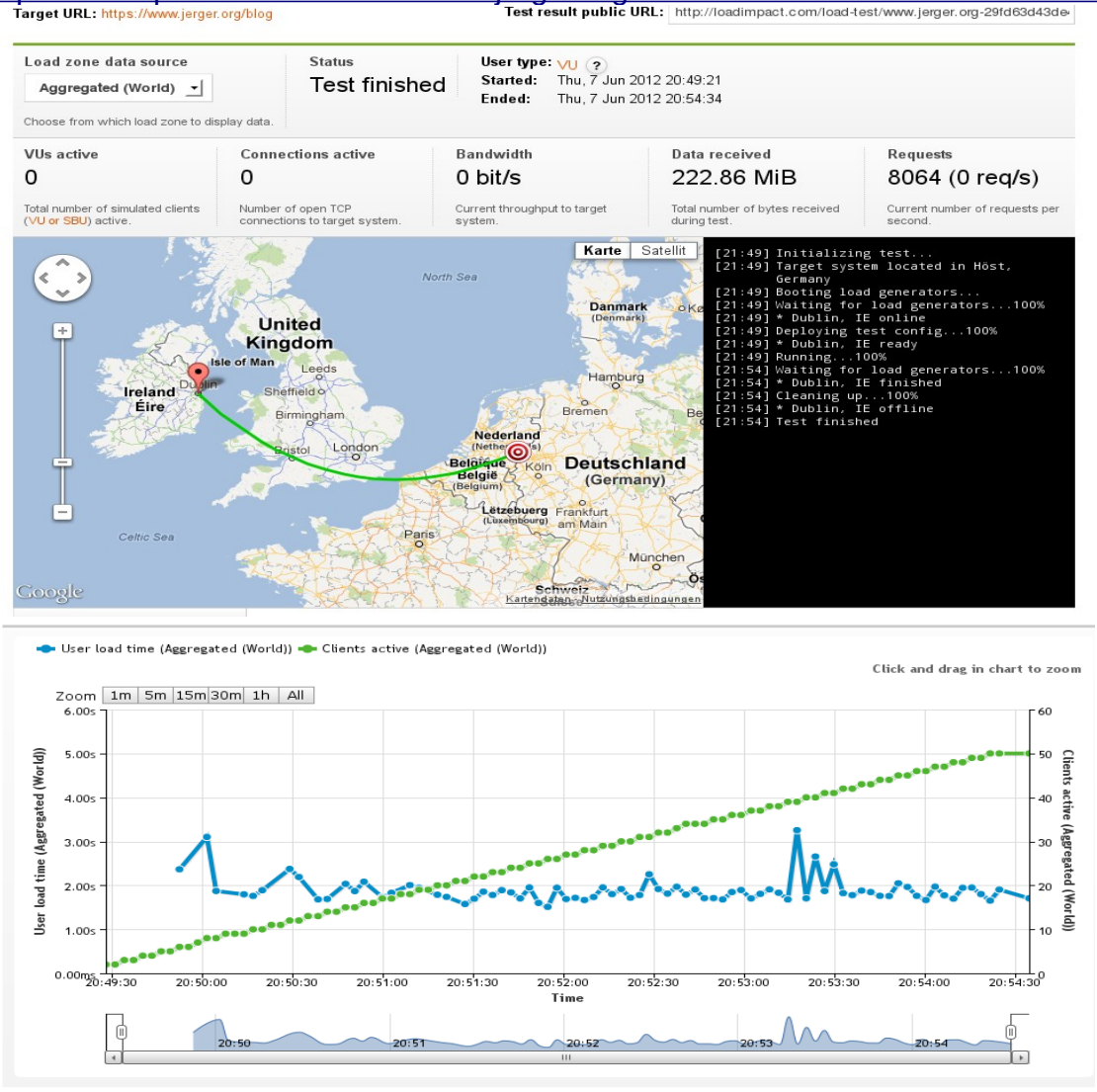
11.1 Lasterzeugung

Simples Erzeugen von Requests:

- Apache Bench (ab) http://httpd.apache.org/docs/2.0/programs/ab.html

Komplexere Szenarien:

- Lasterzeugung in der Cloud: http://loadimpact.com/, Bsp. Ergebnis: http://loadimpact.com/load-test/www.jerger.org-3fca99d44811839dc0d85ccf92b7cecf



- JMeter: http://jmeter.apache.org/

11.2 Informationen und Analysetools

- Browserverhalten für die unterschiedlichen Versionen: http://www.browserscope.org

- Firefox Webentwicklung: https://addons.mozilla.org/de/firefox/addon/firebug/

11.3 Statische Messungen

Virtualisierung mit VM-Ware:

- Angaben unter http://www.vmware.com/technical-resources/performance/benchmarks.html
- http://www.vmware.com/products/vmmark/overview.html
- www.vmware.com/pdf/VI3.5%5FPerformance.pdf

11.4 Dynamische Messungen

11.4.1 Logging für Linux-Systeme

Paket: SAR

- **CPU:** `sadf -d /var/log/sysstat/sa25 -- -u`
```
# hostname;interval;timestamp;CPU;%user;%nice;%system;%iowait;%steal;%idle
www.jerger.org;600;2012-06-15 06:55:01 UTC;-1;0.28;0.00;0.06;0.08;0.00;99.57
```
- **IO:** `sadf -d /var/log/sysstat/sa25 -- -b`
```
# hostname;interval;timestamp;tps;rtps;wtps;bread/s;bwrtn/s
www.jerger.org;600;2012-06-15 06:55:01 UTC;0.00;0.00;0.00;0.00;0.00
```
- **Mem:** `sadf -d /var/log/sysstat/sa25 -- -r`
```
# hostname;interval;timestamp;kbmemfree;kbmemused;
%memused;kbbuffers;kbcached;kbcommit;%commit
www.jerger.org;600;2012-06-15 06:55:01 UTC;85824;3059904;97.27;0;0;0;0.00
```
- **Load:** `sadf -d /var/log/sysstat/sa25 -- -q`
```
# hostname;interval;timestamp;runq-sz;plist-sz;ldavg-1;ldavg-5;ldavg-15
www.jerger.org;600;2012-06-15 06:55:01 UTC;0;207;0.09;0.08;0.03
```
- **Network:** `sadf -d /var/log/sysstat/sa25 -- -n DEV`
```
#
hostname;interval;timestamp;IFACE;rxpck/s;txpck/s;rxkB/s;txkB/s;rxcmp/s;txcmp/s
;rxmcst/s
www.jerger.org;600;2012-06-15 06:55:01
UTC;lo;13.59;13.59;12.86;12.86;0.00;0.00;0.00
```

Dokumentation und Anleitungen:

- SAR unter Ubuntu verwenden: http://aarklonlinuxinfo.blogspot.de/2010/08/how-to-setup-sar-in-ubuntu-1004.html

11.4.2 Logging für Java-VMs

```
java … -verbose:gc -Xloggc:<file> …
```

```
0.068: [GC 5244K->320K(1004928K), 0.0034960 secs]
0.072: [Full GC 320K->241K(1004928K), 0.0037430 secs]
```

Dokumentation und Anleitungen:

- Frontend für das gc – log: http://www.tagtraum.com/gcviewer.html und aktueller https://github.com/chewiebug/GCViewer
- Übersicht: http://www.codecentric.de/kompetenzen/java-performance/java-memory-leak/garbage-collection-analyse/
- Das Gclog-File-Format erklärt: http://sujitpal.blogspot.de/2006/08/charting-jvm-garbage-collection.html

11.4.3 Logging für apache httpd

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\" %D" combinedx
CustomLog /var/log/apache2/ssl_access.log combinedx

79.125.101.106 - - [15/Jun/2012:20:28:41 +0200] "GET
/documents/10725/12651/Leistungsnachweis.png?t=1339444897927&X-
requestId=spec1621339784894 HTTP/1.1" 200 36692 "-" "LoadImpactRload/2.4 (Load
Impact; http://loadimpact.com);" 341001
```

Dokumentation und Anleitung:

- http://httpd.apache.org/docs/2.4/mod/mod_log_config.html#formats

11.4.4 Logging für apache tomcat

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
            pattern="%h %l %u %t &quot;%r&quot; %s %b %D %S"
            prefix="localhost_access_log." suffix=".txt"
            resolveHosts="false"/>

79.125.101.106 - - [15/Jun/2012:20:28:42 +0200] "GET
/documents/10725/12651/Leistungsnachweis.png?t=1339444897927&X-
requestId=spec1621339784894 HTTP/1.1" 200 36692 231 21B527DFAD1800998F66066E95918CD1
```

Dokumentation und Anleitungen:

- http://tomcat.apache.org/tomcat-7.0-doc/config/valve.html

11.5 Monitoring

- Monitoring in der Cloud: http://www.keynote.com/, http://newrelic.com/

11.6 Daten Aggregation und Visualisierung

- Loggaggregatoren & Visualisierung:
  - Aggregation und Visualisierung für Monitoring Daten: http://ganglia.sourceforge.net/, http://graphite.wikidot.com/
  - Report Visualisierung in der Cloud: http://visualizefree.com/index.jsp, http://www-958.ibm.com/software/data/cognos/manyeyes/,
- Business Intelligence:

- OS BI Suite: http://community.pentaho.com/
- Cloud-Service: http://bimeanalytics.com/, http://www.gooddata.com/, http://www.youcalc.com/

12 Fazit

Wie Sie sehen ist ein Lasttest nicht sehr schwer zu verstehen. Aber durch die Komplexität der beteiligten Systeme und Komponenten ist die Möglichkeit für Fehler und Überraschungen sehr groß.

Falls auch Sie Erfahrungen zu Lasttests beizutragen haben, fühlen Sie sich herzlichst zu Kommentaren und Erweiterungen in diesem Dokument eingeladen – wir alle können davon nur profitieren.