Systemtests
Projekt BierIdee

Danilo Bargen, Christian Fässler, Jonas Furrer

24. Mai 2012

Inhaltsverzeichnis

Änderungshistorie

Version	Datum	Änderung	Person
v1.0	07.05.2012	Dokument erstellt, Load Tests dokumentiert	dbargen
v1.1	24.05.2012	Korrekturen	jfurrer

1 Vorwort

Der Systemtest ist die Teststufe, bei der das gesamte System gegen die gesamten Anforderungen (funktionale und nicht funktionale Anforderungen) getestet wird. Gewöhnlich findet der Test auf einer Testumgebung statt und wird mit Testdaten durchgeführt.

2 Funktionale Systemtests

2.1 Android App

Funktionale Systemtests für Android werden mithilfe von Robotium[1] automatisiert umgesetzt. Sie funktionieren grundsätzlich gleich wie die Integration Tests, werden jedoch nicht auf kleinen Einheiten getestet. Stattdessen werden ganze Use Cases automatisiert ausgeführt und verifiziert.

Der Reallife-Einsatz der App wird durch von Entwicklern begleitete User Tests abgedeckt.

2.2 REST API

Da die REST API stateless ist und keine konkreten Workflows existieren, werden hier keine funktionalen Systemtests durchgeführt, da das Testen der Funktionalität bereits durch die Integration Tests komplett abgedeckt wird.

3 API Load Tests

Die Load Tests dienen dazu, die Grenzen des Testsystems auszuloten und zu sehen, ob die gesetzten Performanceziele erreicht werden können.

Um die Load Tests durchzuführen, wurde der Service von `http://blitz.io/` verwendet. Die Seite ermöglicht es, eine API mit Sprints (einfache Analyse der Seite) und Rushes (steigende Belastung durch concurrent Requests) auszulasten. Die Auslastungstests wurden jeweils auf der Beerlist-Ressource durchgeführt.

3.1 50 Concurrent Users

Um unser Ziel von 50 concurrent Users zu verifizieren, haben wir einen Rush durchgeführt, der während 60 Sekunden die Anzahl paralleler Requests von 1 auf

[1] `http://code.google.com/p/robotium/`

50 steigert und Timeouts sowie Fehler loggt.

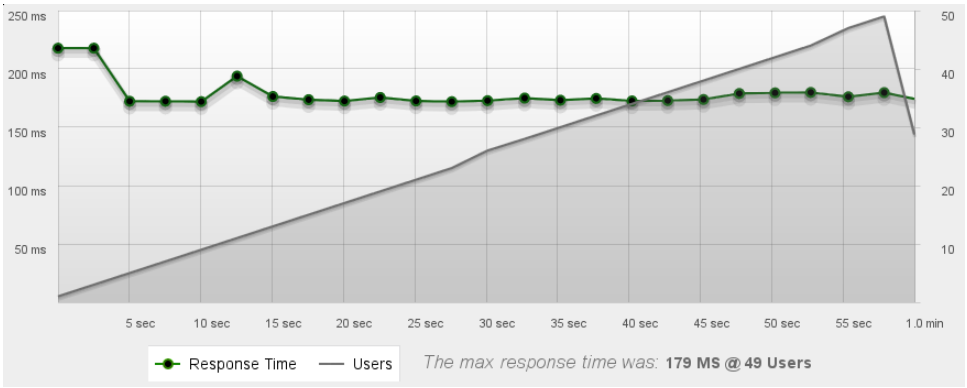
Query

```
-p 1-50:60 http://brauhaus.nusszipfel.com:8080/beers
```

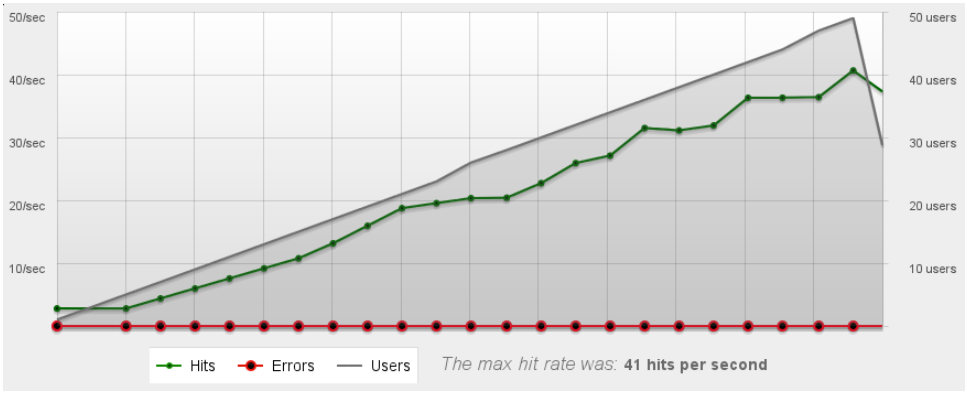
Analyse

Der Rush generierte 1'277 erfolgreiche Hits während einer Minute. Dabei wurden 7.21 MB Daten übertragen. Die Durchschnitts-Hitrate von 20 Hits / Sekunde bedeutet eine erfolgreiche Bewältigung von ca. 1'757'055 Hits / Tag.

Response-Zeiten



Hit Rate



3.2 200 Concurrent Users

Um die Grenzen des Systems zu erkennen, haben wir noch einen Test mit 200 concurrent Users durchgeführt. Dabei wurde während 60 Sekunden die Anzahl paralleler Requests von 1 auf 200 gesteigert und die Timeouts sowie Fehler geloggt.

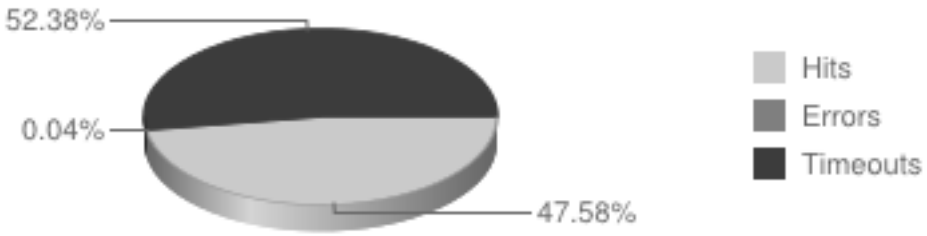
Query

```
-p 1-200:60 http://brauhaus.nusszipfel.com:8080/beers
```

Analyse

Der Rush generierte 2'600 erfolgreiche Hits während einer Minute. Dabei wurden 14.67 MB Daten übertragen. Die Durchschnitts-Hitrate von 41 Hits / Sekunde entspricht ca. 1'757'055 Hits / Tag.

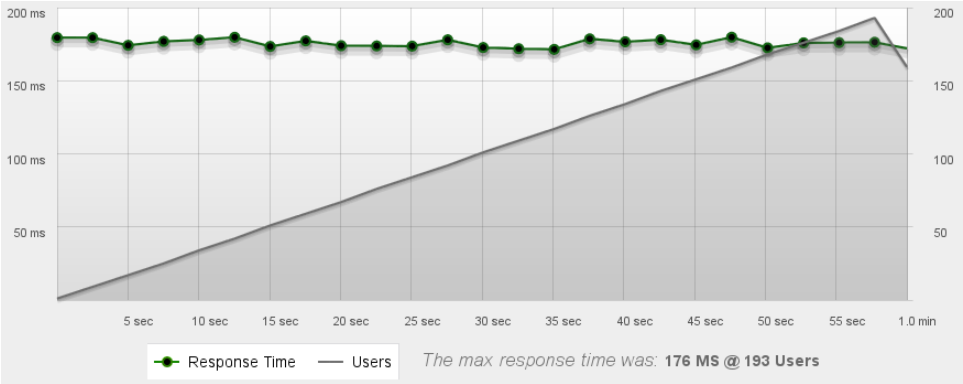
Es gab jedoch Probleme – 52.42% der User Requests resultierten in Timeouts oder Fehlern.



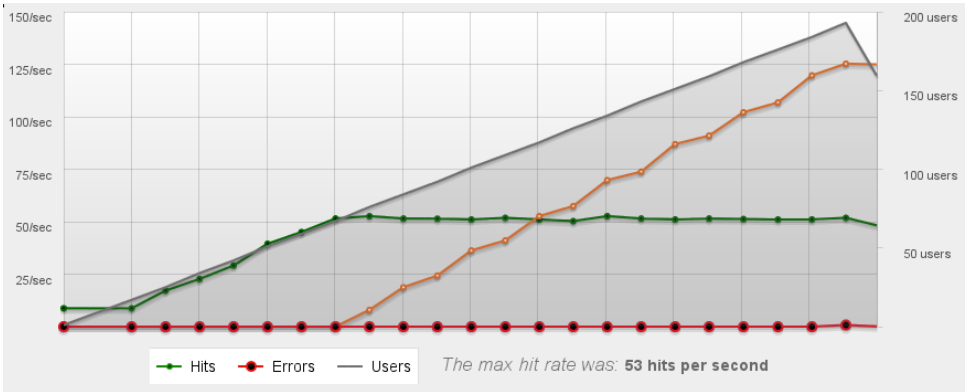
Fehler Der erste Fehler trat nach 57.72 Sekunden bei 193 concurrent Users auf. Die Fehler sind vermutlich auf Ressourcenüberlastung zurückzuführen, wie zB ungenügend freie Filedescriptors oder zu kleine Connection Pools.

Timeouts Der erste Timeout trat nach 22.56 Sekunden bei 76 concurrent Users auf. Die Timeout-Zeit war auf 1 Sekunde gesetzt. Man könnte die Timeouts mithilfe von Caching drastisch reduzieren.

Response-Zeiten



Hit Rate



3.3 Auswertung

Wie man am Test mit 50 gleichzeitien Benutzern sieht, können wir unser selbst gestecktes Auslastungs-Ziel problemlos erreichen. Die maximale Response-Zeit von 179ms bei 41 Hits / Sekunde ist äusserst tief, vor allem wenn man bedenkt dass die Requests von Virginia, USA aus erzeugt wurden. Bei dieser Menge an Anfragen trat kein einziger Timeout oder Fehler auf.

Anders sieht es aus, wenn man die Anzahl paralleler Benutzer erhöht. Wie man in den Diagrammen sehr gut sehen kann, bewältigen der Server ziemlich genau 50 Hits pro Sekunde ohne dass Probleme auftreten. Danach bleibt die Anzahl erfolgreich beantworteter Requests konstant, obwohl die Gesamtanzahl Hits weiterhin erhöht wurde. Die Anzahl der Timeouts steigt parallel zur Erhöhung der parallelen User.

Das heisst, dass die Grenze unseres Systems bei ziemlich genau **76 gleichzeitigen Benutzern** liegt. Dies ist ein sehr guter Wert, wenn man bedenkt dass auf dem Server bisher keinerlei Caching o.ä. eingerichtet wurde.

Auch bei 200 gleichzeitigen Benutzern gab es lediglich viele Timeouts, jedoch keine kritischen Fehler. Das bedeutet, dass die Benutzung der App bei einer solchen Menge an Benutzern zwar sehr langsam wäre, jedoch nicht in Fehlermeldungen resultieren würde.