

Ingenieurbüro David Fischer GmbH
Lindenstrasse 21, CH-4123 Allschwil,
Schweiz

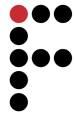
Technisches Büro:
Mühlemattstrasse 61, CH-3007 Bern

<http://www.d-fischer.com>

<http://www.proxy-sniffer.com>

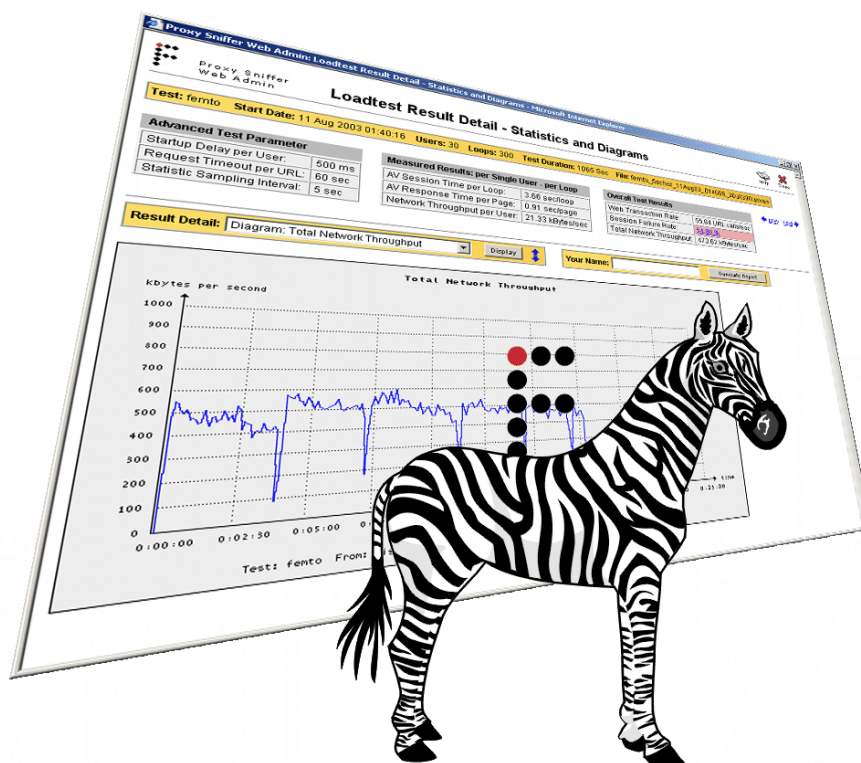
E-Mail: direct@d-fischer.com

INGENIEURBÜRO
DAVID FISCHER



Leitfaden zum erfolgreichen Durchführen von Lasttests

Deutsche Ausgabe



Inhaltsverzeichnis

1	Einleitung	2
2	Was Sie zuerst wissen müssen – ein wenig Theorie	3
2.1	Kollaspunkt – Totalausfall bei Überlast	4
3	Auswahl der Testfälle und Mengengerüst	5
4	Hinweise zur Testausführung – die 8 wichtigsten Punkte	6
5	Ungenügende Antwortzeiten – ist das Netzwerk die Ursache?	7
6	Tuning Tipps	7
6.1	Optimieren der Antwortzeiten	7
6.2	Beheben von Stabilitätsproblemen	8
6.3	Entfernen des Kollaps-Punkts	8
7	Weitere Testarten	9
7.1	Dauertest	9
7.2	Anfahren unter Last	9
7.3	Degradationstest	9
7.4	Kurschlusstest	9
7.5	Stripped Test	10
8	Weitere Informationen & Hersteller	10

1 Einleitung

Dieser kurze Leitfaden richtet sich an Personen, welche noch nicht „ausgefuchste“ Lasttest-Profis sind und soll Ihnen helfen, erste Schritte im anspruchsvollen Gebiet der professionell ausgeführten Lasttests zu unternehmen.

Irgendein beliebiger Lasttest ist recht schnell zusammengestellt und ausgeführt. Spätestens bei der Interpretation der Messresultate stellt sich jedoch die Frage, was diese nun konkret bedeuten und ob die gemessenen Resultate auch einen realen Bezug haben.

Damit Sie am Schluss auch einen Erfolg erzielen, sind etwas Know-How und ein strukturiertes Vorgehen unbedingt nötig. Dieses versuchen wir Ihnen auf den folgenden Seiten zu vermitteln.

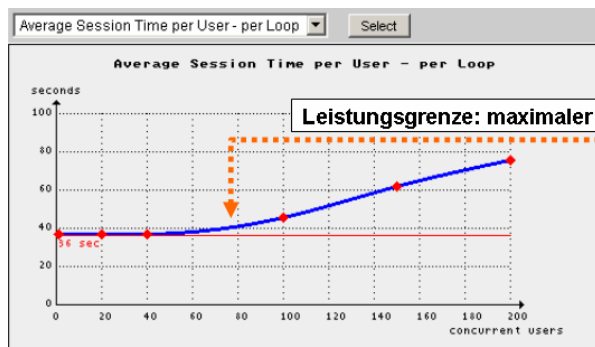
2 Was Sie zuerst wissen müssen – ein wenig Theorie

Die wichtigsten Messresultate, welche Sie bei einem Lasttest messen, sind die so genannten Lastkurven. Diese geben auf einen Blick eine Übersicht über das Systemverhalten bei unterschiedlichen Belastungen. Dabei sind 3 Arten von Lastkurven besonders interessant:

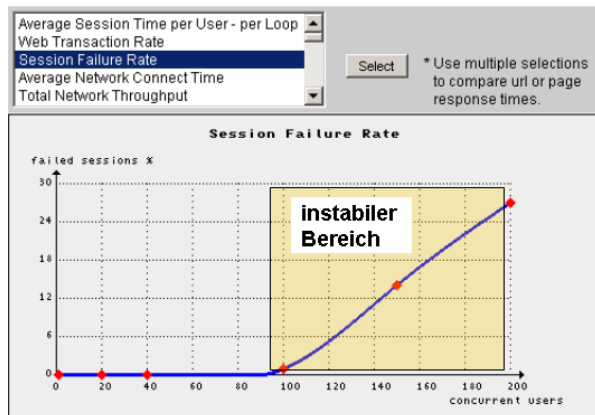
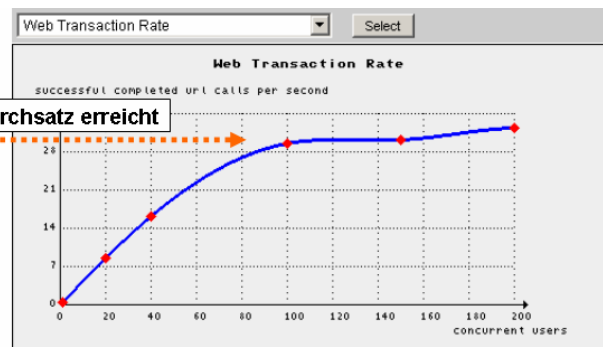
- Die mittlere Zeitdauer einer Web Surf-Session pro Benutzer.
Diese Kurve widerspiegelt das Antwortzeitverhalten unter Last.
- Die gesamte Anzahl erfolgreicher URL-Aufrufe pro Sekunde, gemessen über alle Benutzer.
Diese Kurve widerspiegelt die Leistungsfähigkeit der Web Applikation.
- Den Prozentwert der fehlgeschlagenen Web Surf-Sessions, gemessen über alle Benutzer.
Diese Kurve widerspiegelt die Stabilität der Web Applikation.

Das nachfolgende Bild zeigt ein Beispiel typischer Lastkurven. Die unterschiedliche Anzahl gleichzeitiger Benutzer (d.h. die Last) ist jeweils auf der horizontalen X-Achse abgebildet:

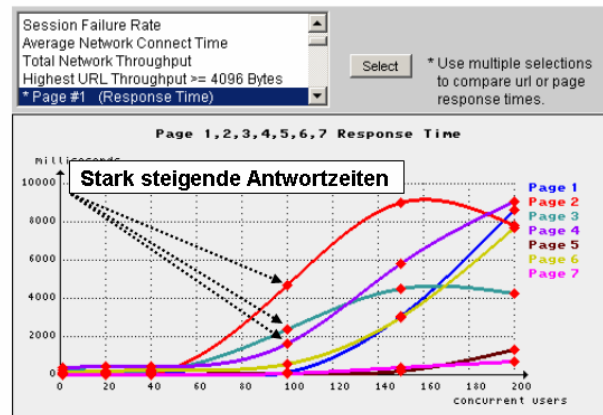
Mittlere Zeitdauer einer Web Surf-Session pro Benutzer



Anzahl erfolgreiche URL-Aufrufe pro Sekunde / Durchsatz



Prozentwert der fehlgeschlagenen Web Surf-Sessions



Mittlere Antwortzeit pro Web-Page

Im Bild unten rechts sehen Sie noch weitere Lastkurven, welche die Antwortzeiten nach Web-Pages aufschlüsseln. Dabei sieht man, dass bei 100 gleichzeitigen Benutzern die Antwortzeiten der Web-Pages 2, 3, und 4 ungewöhnlich stark ansteigen.

Die wichtigste Aussage ist jedoch die Leistungsgrenze der Applikation. Diese liegt bei diesem Beispiel bei ca. 80 gleichzeitigen Benutzern. Wird die Leistungsgrenze noch nicht erreicht, so sind die Antwortzeiten nahezu unabhängig von der Anzahl Benutzer konstant (Bild oben links). Steigt nun die Anzahl Benutzer über 80 an, so erhöhen sich jetzt ab diesem Punkt die Antwortzeiten linear, da der Durchsatz der Web Applikation nicht weiter steigen kann (Bild oben rechts).

Unten links im Bild sieht man auch, dass ab ca. 100 Benutzern erste geringe Instabilitäten auftreten, welche sich danach mit zunehmender Last linear erhöhen. Dies bedeutet, falls man die steigenden

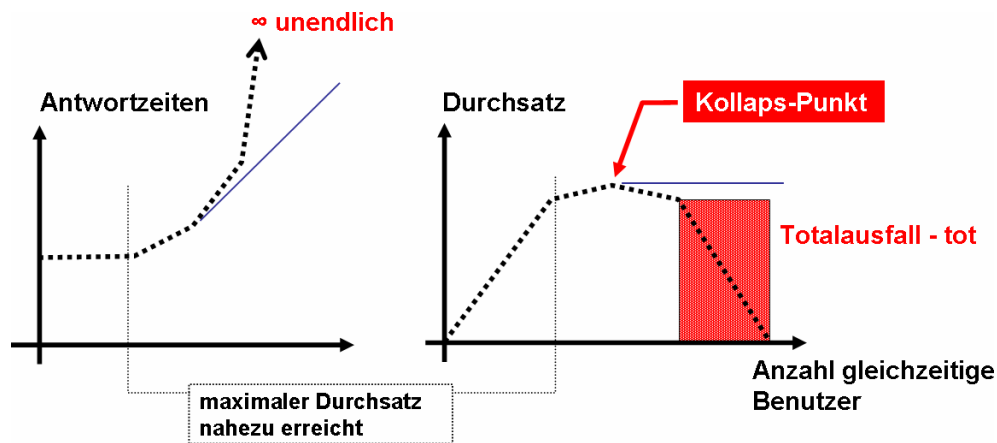
Antwortzeiten ab 80 Benutzern akzeptiert, dass die Applikation höchstens 95 gleichzeitige Benutzer fehlerfrei bedienen kann.

2.1 Kollapsunkt – Totalausfall bei Überlast

Nicht jede Web Applikation zeigt ein Lastverhalten, bei denen die Antwortzeiten beim Überschreiten der Leistungsgrenze „nur“ linear ansteigen. Bei vielen Applikationen sinkt sogar der Durchsatz des Gesamtsystems beim Überschreiten dieser Grenze. Der Messwert, bei der der Durchsatz trotz weiterer Erhöhung der Last wieder sinkt, ist der Kollaps-Punkt.

Wird dieser überschritten, so konvergieren die Antwortzeiten gegen unendlich – was in der Praxis bedeutet, dass die Applikationen nicht mehr verfügbar ist (tot). Da die Benutzer im Web jedoch vorerst immer noch hoffen, mit wiederholten Maus-Klicks oder mittels Browser-Refresh eine Antwort von der Web Applikation zu erhalten, bleibt diese hinter dem Kollaps-Punkt gefangen, und verbleibt dort so lange, bis die Mehrheit der Benutzer eingesehen hat, dass weiteres Klicken zu nichts führt (Verlassen der Applikation).

Kommen laufend neue Benutzer hinzu, z.B. bei einer E-Banking Applikation oder bei einem Konzert-Ticket-Shop, so kann die Web Applikation auch viele Stunden hinter dem Kollaps-Punkt gefangen sein.



Der Grund, wieso eine Applikation einen Kollaps-Punkt hat, liegt fast immer in der Behinderung innerer, paralleler Abläufe (Architektur-, Programmier- oder Konfigurations-Fehler). Wie z.B. zu starke Synchronisation des Programmcodes oder nicht optimierte Datenbank Schreib-Operationen, oder auch zu wenig Ressourcen des Betriebssystems.

3 Auswahl der Testfälle und Mengengerüst

Oft stellt der Auftraggeber einer Web-Applikationen grundlegende Anforderungen, welche erfüllt werden müssen. Zum Beispiel:

1. Die Web Applikation soll 200 gleichzeitige Benutzer bedienen können
2. Die Antwortzeiten pro Web-Page sollen im Durchschnitt nicht mehr als 3 Sekunden, jedoch maximal 5 Sekunden betragen.

Damit steht jedoch noch nicht fest, über welche Web-Seiten der Test gefahren werden soll. Auch steht nicht fest, wie viel Zeit die einzelnen (realen) Benutzer zum Betrachten einer Web-Page verwenden, bevor diese auf den nächsten Hyperlink klicken.

Falls die Web Applikation schon in Betrieb ist, so können Sie ein Teil der fehlenden Anforderungen aus der Analyse der Log-Files gewinnen. So können z.B. die 15 am häufigsten aufgerufenen Web Pages aus den Log-Files bestimmt werden, worauf ein Lasttest vorbereitet wird, welcher genau diese Web-Pages umfasst.

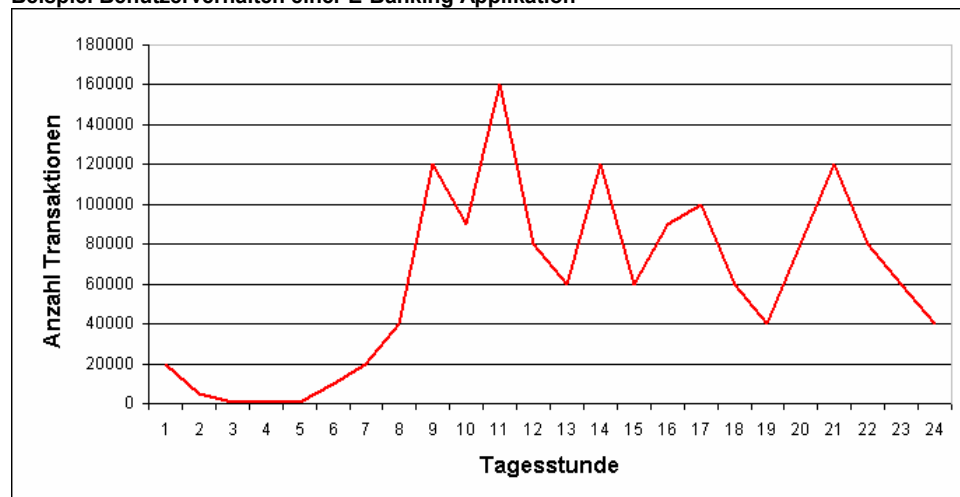
Bei einigen Web-Applikationen ist es jedoch so, dass besonders wichtige Menü-Punkte nicht unter den häufigsten Web-Pages vorkommen. Zum Beispiel bei einem Online-Shop. Dort Surfen die Web-Benutzer oft sehr lange im Warenkatalog, bevor eventuell eine Bestellung und Zahlung erfolgt. Trotzdem muss der Bestell- und Bezahl-Vorgang natürlich auch noch mit einigermaßen vertretbaren Antwortzeiten erfolgen.

Hier ist es oft günstiger, anstelle nur einer Lasttest-Session deren zwei gleichzeitig zu fahren. Zum Beispiel ein Test, welcher für eine Grundlast steht und mit vielen Benutzern ausgeführt wird – und ein zweiter, gleichzeitig durchgeführter Test, bei welchem mit wenigen Benutzern die die wichtigsten Spezialfälle abgedeckt werden.

Falls Sie gar keine oder nur pauschale Eckdaten erhalten, so müssen Sie die restlichen Werte schätzen. Dabei sollten Sie jedoch das reale Benutzerverhalten so gut wie möglich nachempfinden.

Wenn beispielsweise bei einer E-Banking Applikation nur bekannt ist, dass 1.5 Millionen Finanz-Transaktionen pro Tag durchgeführt werden, so sollten Sie auch Berücksichtigen, dass diese sich nicht gleichmässig über den ganzen Tag erstrecken:

Beispiel Benutzerverhalten einer E-Banking Applikation



Dies bedeutet, das – falls nur Mittelwerte der zu erreichenden Last bekannt sind – Sie diese mit einem Faktor für Spitzenzeiten multiplizieren müssen. Reale Multiplikationsfaktoren liegen Erfahrungsgemäss in einem Bereich zwischen 3 und 12 (typisch bei 5).

Um nochmals auf das E-Banking Beispiel zurückzukommen: nur aus den Anzahl Finanz-Transaktionen können Sie nicht schliessen, wie viele Benutzer nun während eines Lasttests benötigt werden. Durch das Ausmessen der Lastkurven ermitteln Sie jedoch die maximale Leistungsfähigkeit der Web-Applikation, und damit auch die maximale Anzahl mögliche Finanz-Transaktionen. Da bei Lastkurven

ein direkter Bezug zwischen dem Durchsatz und den Antwortzeiten besteht, können Sie nun auch die erwarteten Antwortzeiten bei geringer, mittlerer und hoher Last bestimmen.

4 Hinweise zur Testausführung – die 8 wichtigsten Punkte

- Um die Lastkurven zu erhalten führen Sie denselben Test mehrmals aus, jedoch mit einer unterschiedlichen Anzahl Benutzer (z.B. 1, 2, 5, 10, 20, 50, 100, 200, 500 ...). Empfohlen wird eine schrittweise, logarithmische Erhöhung der Last, um eine schnelle Übersicht zu gewinnen.
- Führen Sie eine einzelne Testmessung nicht zu kurz aus. Die Testdauer sollte ab 50 Benutzer mindestens 5 Minuten betragen (Faustregel: Anzahl Benutzer dividiert durch 10 ergibt die minimale Testdauer in Minuten). Bei sehr hohen Benutzerzahlen sollten sie mitberücksichtigen, dass die einzelnen Benutzer nicht sofort erzeugt werden und es darum einige Zeit dauert, bis die angeforderte Last erreicht wird.
- Es kommt hin und wieder vor, dass eine Web Applikation während eines Lasttests stirbt, und auch nach dem Ende des Tests sich nicht wieder von selbst erholt. Sorgen Sie dafür, dass während des Tests jemand erreichbar ist, welcher die Web Applikation jederzeit neu starten kann – damit Sie Ihre Tests fortsetzen können.
- Der Lasttest selbst ist ungültig, wenn das Last-auslösende System (d.h. das System das die Last erzeugt) selbst überlastet ist – da so falsche Antwortzeiten gemessen werden. Behalten Sie darum dessen CPU-Auslastung während des Tests etwas im Auge. Falls das Last-auslösende System zu wenig CPU-Kapazität hat, so sollten Sie mehrere Last-auslösende Systeme zu einem Load-Injector Cluster zusammenschalten, wodurch das Lasttest-Programm auf mehrere Rechner verteilt wird.
- Falls die Applikation einen geschützten Login-Bereich hat, so sollte jeder Lasttest-Benutzer einen eigenen Login-Account verwenden, da sonst der Test gegenüber dem Lieferanten bzw. gegenüber den Entwicklern nicht „fair“ ist. Wird für mehrere Benutzer nur ein Account verwendet, so können unter Umständen Fehler gemessen werden, welche im realen Betrieb so nicht auftreten.
- Beauftragen Sie jemanden während des Lasttests, auch die CPU-Auslastung des Web Applikations-Servers zu betrachten oder zu messen. Sie benötigen diese Information um später beurteilen zu können, ob beim Erreichen des maximalen Durchsatzes bzw. der Lastgrenze die CPU des Applikationsservers auch zu 100% ausgelastet war. Liegt die CPU-Auslastung des Servers auch bei höchster Last unter 70% so bedeutet dies, dass die Web Applikation die Ressourcen des Servers nicht richtig nutzen kann und dass ein entsprechendes Tuning-Potential besteht.
- Achten Sie darauf, dass die Netzwerk-Kapazität zwischen den Last-auslösenden Systemen und dem Web Applikations-Server genügend hoch ist (100 MBit oder grösser). Messen Sie nicht über langsame DSL-Leitungen.
- Idealerweise ist der System-Administrator sowie der Datenbank-Administrator und auch ein Entwickler während des Lasttests vor Ort, so dass Sie gemeinsam die Lasttests-Resultate analysieren und gleich darauf die dazu nötigen Tuning-Massnahmen umsetzen können. Danach erfolgt eine weitere Messung zum „Beleg“ der Tuning-Fortschritte und gegebenenfalls eine weitere Analyse der Messresultate und so fort – bis die Web Applikation zufrieden stellend läuft. Durch die Anwesenheit aller massgebenden Personen sparen Sie erheblich Zeit und können so rasch einen Erfolg erzielen.

5 Ungenügende Antwortzeiten – ist das Netzwerk die Ursache?

Wenn bei einem Test ungenügende Antwortzeiten gemessen werden, so wird oft vermutet, dass „das Netzwerk“ die Ursache des Problems ist. In der Realität ist dies jedoch äusserst selten der Fall.

Glücklicherweise können Sie nur aufgrund der Messdaten recht einfach entscheiden, ob dies zutrifft: vergleichen Sie die Antwortzeit eines statischen Bildes mit den Antwortzeit einer dynamisch erzeugten HTML-Seite, wobei beide etwa die gleiche Grösse haben sollten.

Beispiel:

- Bei einem statischen GIF-Bild von 20'000 Bytes Grösse wurde eine Antwortzeit von 60 Millisekunden gemessen.
- Bei einer dynamisch erzeugten HTML-Seite von 40'000 Bytes Grösse wurde eine Antwortzeit von 3 Sekunden gemessen.

Wenn nun die Antwortzeit des Bildes ausschliesslich von der Netzwerk-Bandbreite abhängen würde (was natürlich nicht stimmt), so müsste die HTML-Seite eine Antwortzeit von 120 Millisekunden haben, da diese ja „nur“ doppelt so gross ist. Da in Wirklichkeit jedoch 3 Sekunden gemessen wurden, gehen mindesten $3000 - 120 = 2880$ Millisekunden im Web Applikationsserver selbst verloren.

6 Tuning Tipps

Versuchen Sie zuerst durch eine Interpretation der Messresultate die Ursache für ungenügende Antwortzeiten oder Stabilitätsprobleme zu ermitteln. Optimieren Sie Ihre Infrastruktur immer nur aufgrund real gemessener Fakten:

6.1 Optimieren der Antwortzeiten

- Falls unter steigender Last sowohl die Antwortzeiten der statischen Bilder wie auch die der dynamisch erzeugten HTML-Pages im Gleichschritt ansteigen, so liegt eventuell ein Netzwerk-Problem vor, oder noch wahrscheinlicher eine Fehlkonfiguration des Web Servers vor (zuwenig Working-Threads oder zuwenig Working-Prozesse).
- Falls nur einzelne URL-Aufrufe unter steigender Last überproportional langsamer werden liegt die Fehlerursache innerhalb der Programmierung der Web-Applikation oder bei ineffizienten Datenbank-Abfragen.
 1. Falls der Web Applikations-Server einen so genannten „DB Connection Pool“ enthält so überprüfen Sie als erstes, ob dieser genügend gross konfiguriert ist. In der Regel benötigt während eines Web-Page Aufrufs jeder Benutzer eine eigene DB-Verbindung aus dem Pool, falls innerhalb der Web Applikation keine speziellen Cache-Funktionen vorhanden sind.
 2. Überprüfen Sie als nächstes die Datenbank-Engine, in dem während eines Lasttests ein DB-Administrator eine Statistik des langsamsten SQL-Queries und deren Antwortzeiten erstellt.
 3. Führt auch dies zu keinem Ergebnis so liegt die Fehlerursache in der mangelhaften Programmierung der Web Applikation. Der häufigsten Programmierfehler ist eine übermässige Synchronisation innerer Abläufe, bei denen gewisse interne Ausführungsschritte über alle Benutzer gesehen sequenziell anstatt parallel ausgeführt werden. Als begleitendes Symptom wird die CPU des Applikationsservers auch unter hoher Last kaum benutzt, da innere Blockierungen keine CPU-Zeit verursachen.

6.2 Beheben von Stabilitätsproblemen

Versuchen Sie durch eine Analyse der am häufigsten aufgetretenen Fehler zu entscheiden, ob die Ursache innerhalb oder ausserhalb der Web Applikation liegt.

- Treten Netzwerk-Fehler oder Timeouts relativ gleichmässig über alle URL-Aufrufe auf, so liegt die Ursache nicht innerhalb der Web Applikation. Versuchen Sie als erstes den TCP/IP Stack des Applikations-Servers mittels der Netzwerk- und Systemparameter zu Tunen. Weitere Fehlerquellen sind Firewalls, Loadbalancer und Reverse-Proxy. Messen Sie gegebenenfalls wiederholt von verschiedenen Netzwerk-Standorten aus – immer von innen nach aussen. Zum Beispiel zuerst möglichst nahe beim Applikations-Server – vom gleichen LAN aus – danach indirekt über den Reverse-Proxy oder den Load-Balancer und so fort ...
- Treten Stabilitätsprobleme nur bei bestimmten URL-Aufrufen auf, so liegt die Ursache innerhalb der Web Applikation. Dasselbe gilt auch, wenn Sie HTML-Pages mit eingebetteten Fehlermeldungen oder teilweise fehlendem Inhalt erhalten.

Um Stabilitätsprobleme einzelner URL-Aufrufe zu beheben benötigen Sie die Unterstützung der Entwickler. Schildern Sie den Entwicklern das Problem und motivieren Sie diese, den Sourcecode diesbezüglich zu analysieren. Um das Problem besser eingrenzen zu können müssen im Sourcecode gegebenenfalls auch Applikations-interne Messpunkte, d.h. Log-Meldungen über interne Verarbeitungsschritte inkl. Timestamps zusätzlich hinzugefügt werden. Dieses Verfahren wird auch als „White Box Test“ bezeichnet. Die internen Messpunkte werden dabei zuerst nur grob über den Code gestreut, und danach iterativ immer näher dem gefundenen Problembereich angepasst, bis nach wenigen Durchläufen die Ursache des Problems gefunden wird. „White Box Tests“ könne natürlich auch zur Optimierung der Antwortzeiten eingesetzt werden.

6.3 Entfernen des Kollaps-Punkts

Versuchen Sie zuerst, den Kollaps-Punkt durch ein Tuning der Antwortzeiten in einen so hohen Lastbereich zu bringen, dass dieser real im Betrieb nicht mehr auftritt. Ist dies nicht möglich, so besteht die Alternative, dass Sie dem Web Applikations-Server einen Reverse Proxy vorschalten (z.B. Apache) und den Revers Proxy so Konfigurieren, dass dieser als Lastbremse wirkt, so dass der Kollaps-Punkt nun nie erreicht wird. Der Reverse Proxy muss zu diesem Zweck mit wiederholten Messungen genau der Leistungsfähigkeit des Web Applikations-servers angepasst werden. Ebenso muss der TCP/IP Stack des Reverse Proxy in einem solchen Fall besonders sorgfältig eingestellt werden.

7 Weitere Testarten

Mit der Messung der Lastkurven decken Sie die Testarten ab, die üblicherweise als Last- und Stress-Test bezeichnet werden. Darüber hinaus gibt es noch weitere Testarten, welche zusätzliche Informationen ergeben.

7.1 Dauertest

Ein Dauertest gibt Auskunft über die Langzeitstabilität der Web Applikation, d.h. beantwortet die Frage, ob diese auch über mehrere Tage stabil läuft. Dazu wird üblicherweise am Abend ein Lasttest gestartet, welcher die Applikation mit ca. 50% der maximalen Leistungsfähigkeit belastet, und bis zum nächsten Morgen dauert (12 Stunden lang).

Da normalerweise ein Dauertest viel mehr URL-Aufrufe macht, als real im Betrieb während eines ganzen Tages vorkommen, entsteht so eine Art Zeitraffer-Funktionalität. D.h. in einer Nacht können so mehrere Tage reale Belastung simuliert werden.

7.2 Anfahren unter Last

Diesen Test sollten Sie zusätzlich vornehmen, wenn es sich um eine neu entwickelte Web Applikation handelt, welche noch nie zuvor produktiv „am Netz“ war. Dazu wird die Web Applikation zuerst abgeschaltet, und ein Test gestartet, welcher die Applikation nahezu zu 100% belasten würde. Sie erhalten zuerst nur Mess-Fehler, da die Applikation ja noch nicht läuft. Nun wird die Applikation unter diesen Bedingungen, d.h. bei laufendem Test gestartet, wobei die real-time Testresultate laufend beobachtet werden. Die Web Applikation muss sich nun innerhalb von wenigen Minuten stabilisieren.

Der Hintergrund dieser Testart besteht darin, dass während dem Applikations-Startup-Vorgang sich die Web Applikation bereits am Netz anmeldet, bevor diese intern richtig initialisiert ist – und sich danach am den ersten URL-Aufrufen so „verschluckt“, dass sie gleich wieder stirbt. Dies sollte nicht möglich sein und kann mit der Testart „Anfahren unter Last“ überprüft werden.

7.3 Degradationstest

Ist die Web-Applikation mit redundanten Komponenten versehen, welche die Verfügbarkeit erhöhen wie z.B. ein Loadbalancer oder ein Web Applikations-Cluster, so werden bei dieser Testart während der Zeitdauer eines (einzigen) Tests einzelne System-Komponenten (z.B. ein Ast eines Loadbalancers oder auch ein Web Applikations Cluster-Rechner) aus- und danach wieder ein-geschaltet. Dadurch können Sie überprüfen, ob die Verfügbarkeit auch bei Teilausfällen gegeben ist und wie viel Einfluss ein Ausfall einer oder mehrere Komponente auf die Performanz und Stabilität hat.

7.4 Kurschlusstest

Ein Kurschlusstest ist eine Testart, bei der die Infrastruktur ohne die (programmierte) Web Applikation getestet wird. Dazu werden ein paar grössere statische Bilder auf dem Applikationsserver hinterlegt und die Lastkurven nur mit diesen Bildern gemessen. Dadurch erhalten Sie den maximalen Durchsatz, welcher noch ohne dynamisch erzeugte HTML-Pages möglich ist und können auch abschätzen, ob bereits „ohne Applikation“ ein Performanz- oder Stabilitätsproblem besteht.

Natürlich kann so auch der Performanz-Verlust eines Reverse Proxy gemessen werden, in dem ein Mal die Applikation direkt belastet wird und ein anderes mal die Belastung über den Reverse Proxy erfolgt.

7.5 Stripped Test

Bei einem Stripped Test werden alle URL-Aufrufe welche den statischen Content betreffen aus dem Lasttest-Programm entfernt (z.B. Bilder, Style-Sheets und Java-Script Dateien). So das nur die „nackten“, funktionalen HTML-Aufrufe verbleiben, welche die Applikationslogik darstellen.

Diese Testart wird vor allem von Entwicklern verwendet, wenn die Antwortzeiten der einzelnen, dynamisch erzeugten Web-Pagers optimiert werden sollen. Durch das Weglassen der Bilder wird die Netzwerkbandbreite und die CPU des Last-auslösenden Systems geschont, so dass auch mit beschränkten Ressourcen viele gleichzeitige Benutzer emuliert werden können. Es fallen auch weniger Messwerte an, so dass es einfacher ist, den Überblick zu behalten.

8 Weitere Informationen & Hersteller

Ingenieurbüro David Fischer GmbH
Lindenstrasse 21
CH-4123 Allschwil
Schweiz

Produkt Homepage: <http://www.proxy-sniffer.com>

Hersteller Homepage: <http://www.d-fischer.com>

E-Mail: direct@d-fischer.com

Alle Rechte vorbehalten.