

Michael Scharkow und Steffen Müller

# Frisiert und aufgebohrt

## Bessere Performance mit TYPO3

**Idealerweise sollte die Performance einer Website so lange kein Thema sein, wie ihre Anwendung mit Hilfe von mehr und besserer Hardware skalierbar bleibt.**

Warum sich mit aufwändigen Optimierungsmaßnahmen herumärgern, wenn man das Problem auch einfach mit zusätzlicher Hardware lösen kann?



Fast alle großen dynamischen Websites, wie z. B. Wikipedia, Slashdot und ganz besonders Google, vertrauen in puncto Performance auf die ständige Verbesserung der Hardware. Aus der Sicht einer öffentlich finanzierten Einrichtung haben wir uns hingegen mit der Frage beschäftigt, ob man die Leistung von TYPO3 auch ohne finanzielle Investitionen verbessern kann. Im Fokus standen dabei die Antwortzeit und die Auslastung des Servers sowie die Suche nach dem effektivsten Mittel zur Optimierung. Das Ziel war, eine einfache und schnelle Lösung mit dem größtmöglichen Effekt zu finden.

### Die Testumgebung

Der Weg zu einem optimierten System führt in der Regel über Testreihen, so genannte Benchmarks. Diese führten wir auf derzeit verbreiteter Hard- und Software durch: ein Athlon XP 2200 mit 1 GB RAM, gewissermaßen die Standard-Hardware im Shared- und Rootserver-Segment. Da wir die Wirkung unterschiedlicher Software-Versionen testen wollten, benutzten wir für unsere Tests die folgenden Plattformen:

- GNU/Linux: Debian Sarge und Ubuntu Breezy, Linux-Kernel 2.4 und 2.6
- MySQL 4.0 und 4.1
- Apache 1.3 und 2.0, lighttpd 1.4.8
- PHP4 und PHP5
- TYPO3 3.8, Quickstart-Paket (FC Bigfeet)

Auch wenn wir bei den Tests Linux verwendet haben, gelten die meisten Empfehlungen auch für Windows.

Bei unseren Versuchen zur Verbesserung der Performance haben wir uns auf zwei Fragen konzentriert: Wie schnell werden gleichzeitige Anfragen bewältigt und wie groß ist dabei die Systemlast? Für diese Zwecke haben wir verschiedene Tools im Netz gefunden. Unser Favorit war das in den Apache Utilities enthalte-

ne Apachebench, das dazu diente, den HTTP-Output von Websites zu messen [1].

Ausgangspunkt für unsere Testreihe war ein Worst-Case-Szenario: ein neu installierter Debian-Linux-Server mit einem standardmäßigen Apache 1.3 Webserver, einer MySQL-Datenbank, mod\_php4 und einem jungfräulichen TYPO3 3.8 Quickstart-Paket mit ausgeschaltetem TYPO3-Caching. Zusätzlich benutzten wir einen zweiten Rechner als Client, der sich am selben 100-Mbit-Switch wie der Server befand.

Um einen Vergleichswert zu erhalten, bestand unser erster Test aus einer Anfrage nach einer 24 KB großen statischen Datei, also ohne Einbeziehung von MySQL, PHP und TYPO3. Auf diese Weise haben wir die Maximalleistung unseres Webservers ermittelt. Wir starteten Apachebench mit 1000 Anfragen, je 100 davon gleichzeitig.

```
SHELL
$ab -n 1000 -c 100 http://yourserver/static_file.txt
```

Listing 1

Die durchschnittliche Zeit für eine Anfrage betrug ca. 3 ms. Das bedeutet mehr als 300 gleichzeitige Anfragen pro Sekunde (300 Req/s). Im nächsten Schritt wendeten wir uns TYPO3 zu:

```
SHELL
$ab -n 1000 -c 100 http://yourserver/index.php?no_cache=1
```

Listing 2

Neben einem zeitweise nicht mehr erreichbarem System war das Ergebnis ein recht enttäuschender Wert von 4 Req/s – kein Vergleich zum statischen Test: Zeit für eine Optimierung.

### Betriebssystem

Die Wahl des Betriebssystems macht in manchen Fällen einen erkennbaren Performance-Unterschied aus. Einige bekannte Benchmark-Vergleiche betonen, dass Linux 2.6 sowohl in Hinsicht auf Apache als auch MySQL eine höhere Leistung bietet als die Version 2.4 [2] [3]. Auch skaliert Linux besser als die freien BSD-Derivate, ganz zu schweigen von Windows oder Solaris [4].

Viele dieser Benchmarks waren entweder „low-level“-Tests oder wurden auf extrem leistungsfähiger Hardware (große Multi-processor-Systeme) durchgeführt. Für unsere Zwecke überprüften wir, ob ein einfaches Ersetzen des vorhandenen 2.4.x Kernels mit der Version 2.6.14.x und das Aktivieren der Native Posix Threads Library (NPTL) die Performance erhöhen. Die Auswirkungen waren jedoch kaum messbar und nicht einmal auf einem beständigen Niveau. Das Ergebnis lag immer noch im Bereich von 4 Req/s.

Bedeutet das, dass eine Anpassung des Betriebssystems sinnlos ist? Nein, aber in unserem Fall war dies einfach nicht der signifikante Engpass. Dennoch sollten Sie Ihr System NPTL-fähig machen, um Threads statt Prozesse zu nutzen. Im Vergleich zu Prozessen reduzieren Threads zum Beispiel die durch Apache 2.0

verursachte Systemauslastung deutlich. Das bedeutet aber keinesfalls, dass Seiten automatisch schneller ausgeliefert werden (siehe unten).

## MySQL-Datenbank

Oft wird angenommen, die Datenbank sei der Grund für eine schlechte Performance. Wir wissen, dass TYPO3 erstere ziemlich intensiv nutzt. Es führt für jede einzelne Anfrage mehrere Abfragen durch, zum Beispiel, um Daten aus dem Cache zu holen und Sessions zu aktualisieren. Die Optimierung von MySQL ist eine Wissenschaft für sich und es existieren dazu sowohl unzählige Mythen als auch nützliche Rezepte [5]. Wir haben uns lediglich darauf konzentriert, einige Server-Parameter zu verändern und die Auswirkungen zu messen, hier am Beispiel der MySQL-Version 4.1.

Zunächst erhöhten wir die maximale Anzahl gleichzeitiger DB-Verbindungen auf 100 – da TYPO3 als Voreinstellung persistente DB-Verbindungen benutzt, entspricht die Zahl ungefähr der Anzahl der MaxClients in Apache (siehe unten). Diese Veränderung brachte zwar keine Leistungssteigerung, schaffte aber Reserven für schnellere Anfragen im späteren Verlauf der Testreihe.

```
CONFIG
my.cnf:
max_connections = 100
```

Listing 3

Weil wir genug freien RAM-Speicher hatten, bot sich der Einsatz von „Query Caching“ [6] an, um die vielen sich wiederholenden Abfragen von TYPO3 leichter bewältigen zu können.

```
CONFIG
my.cnf:
query_cache_limit = 2M
query_cache_size = 64M
query_cache_type = 1
table_cache = 256
key_buffer_size = 64M
```

Listing 4

Anschließend deaktivierten wir das Logging von DB-Abfragen, weil wir keine weiteren MySQL-Datenbanken zur Replikation einsetzten und unsere Backup-Strategie nicht darauf basierte, jeden Schreibzugriff aufzuzeichnen.

```
CONFIG
my.cnf:
log-bin
```

Listing 5

Auch nach den Änderungen an der Datenbank hatte sich die Leistung nicht wesentlich verändert: 4 Req/s. Ein Blick auf die Ausgabe von `mysqldadmin status` zeigte, dass das Caching sehr gut funktionierte. Wo war also das Problem? Wie zuvor beim Betriebssystem stellte sich heraus, dass die Datenbank nicht der vermutete Engpass war. Und zwar aus zwei Gründen:

1. Mit einer einzelnen Website hatten wir nicht so viele gleichzeitige Verbindungen, sodass das Erstellen von Threads für Anfragen kein Problem darstellte. Das wäre bei mehreren parallel installierten Websites anders.
2. Die Quickstart-Website lieferte (größtenteils) statische Inhalte, die ausgezeichnet von MySQL zwischengespeichert werden konnten. Das würde sich mit dynamischeren Inhalten ändern.

## Webserver – Apache und Alternativen

Der folgende Abschnitt befasst sich mit der Optimierung und dem Vergleich von Apache 1.3 und 2.0. Zusätzlich schauen wir uns den verhältnismäßig neuen und schlanken Webserver `lighttpd` [7] an.

Apache 1.3 für TYPO3 zu optimieren war nicht gerade eine hohe Wissenschaft, da wir hauptsächlich mit den Parametern herumspielten, die mit dem Verteilen auf Unterprozesse zu tun hatten. Das Hauptproblem war die maximale Anzahl der erstellten Unterprozesse (`MaxClients`). Der Standard-Wert von `MaxClients` liegt bei Debian bei 150, was für statische Inhalte auch vollkommen angemessen ist. Sobald jedoch TYPO3 ins Spiel kommt, verarbeitet jeder dieser Apache-Prozesse über `mod_php` auszuführenden Code, was eine immense Belastung für CPU und Speicher darstellt. Unser System war nicht leistungsstark genug, um eine derartige Menge an gleichzeitigen Anfragen zu bewältigen und daher innerhalb von Sekunden überlastet. Die Anzahl gleichzeitig erlaubter Prozesse musste also so weit reduziert werden, dass das System noch unterhalb seiner Leistungsgrenze arbeitet.

```
CONFIG
httpd.conf:
MaxClients 32
```

Listing 6

Eine bessere Lösung versprach Apache 2.0, mit dem ein neues Prozessmodell vorgestellt wurde, welches keine rechenintensiven Prozesse, sondern vergleichsweise wenig aufwändige Threads erzeugt. Das so genannte „`mpm-worker`“-Modul verbraucht Berichten zufolge deutlich weniger Ressourcen [7]. Leider arbeitet es zurzeit noch immer nicht vollständig stabil mit `mod_php` zusammen, obwohl sich auch positive Erfahrungsberichte im `www` und `usenet` finden. Hinreichend stabil dagegen ist das „`mpm-prefork`“-Modul (ohne Threading), welches mit TYPO3 in etwa die gleiche Systemlast erzeugte wie Apache 1.3.

Zur Steigerung der Performance von Apache haben wir an einigen gängigen Stellen der Konfiguration Änderungen vorgenommen, allerdings ohne nennenswerte Verbesserungen. Das Logging wurde minimiert. Im Produktivbetrieb wird ohnehin empfohlen, das Fehlerprotokoll zu reduzieren:

```
CONFIG
httpd.conf:
LogLevel warn
```

Listing 7

Um die Zahl der Dateizugriffe zu verringern, kann man das Überschreiben von Konfigurationsdirektiven verbieten. Andernfalls wird bei jeder Anfrage rekursiv nach `htaccess`-Dateien auf der Festplatte gesucht:

```
CONFIG
httpd.conf:
AllowOverride None
```

Listing 8

Verzögerungen durch DNS-Rückfragen für das Zugriffs-Log sollten stets abgeschaltet werden. Der Logfile-Analyzer kann dies problemlos im Nachhinein übernehmen:

## CONFIG

```
httpd.conf:
HostnameLookups Off
```

Listing 9

Schließlich kann die Server-Ausgabe mit `mod_gzip` bzw. `mod_deflate` komprimiert werden. Das hilft vor allem, wenn die Netzwerkverbindung den Flaschenhals darstellt.

Eine Alternative zu Apache stellt `lighttpd` unter Verwendung der FastCGI-Schnittstelle [8] dar. Beim Ausliefern von statischen Daten übertrifft `lighttpd` Apache bei weitem. Wir verglichen für unsere Zwecke `lighttpd+FastCGI+PHP` mit `Apache1.3+mod_php`. `Lighttpd` war etwa 15 Prozent langsamer als Apache, obwohl wir aufgrund anders lautender Berichte das Gegenteil erwartet hatten. Möglicherweise lag es daran, dass wir TYPO3 verwendeten und nicht einfach nur `phpinfo()` ausgeben ließen. In Sachen Serverlast war allerdings `lighttpd` der Genügsamere. Nur vier FastCGI-Prozesse schafften fast den gleichen Durchsatz wie Apache 1.3, bei einem sichtbar geringeren Ressourcenverbrauch. In bestimmten Umgebungen scheint `lighttpd` aufgrund seines schlanken Designs, eingängiger Konfiguration und Skalierbarkeit eine gute Wahl zu sein, zum Beispiel durch die Aufteilung von FastCGI-Prozessen auf mehrere Rechner und integrierte Lastverteilung. Es empfiehlt sich, vor einer Entscheidung die jeweiligen Konfigurationen ausgiebig zu testen.

Was gibt es in Sachen Webserver zu resümieren? Passen Sie `MaxClients` an, um ein Gleichgewicht zwischen Datendurchsatz und Serverlast zu finden. Ein zurückhaltender Startwert ist 32, der im Lauf der Optimierung wieder gesteigert werden kann. Werfen Sie einen Blick auf `lighttpd`, wenn Sie schlanke Software mögen und ohne die Module und den Support leben können, den Apache bietet. Da wir hauptsächlich mit wenigen, aber aufwändigen, dynamischen Seitenanfragen zu tun haben, steigern Änderungen an der Webserver-Konfiguration unsere Performance nur geringfügig. Weitere Details zum Thema Apache-Performance gibt es unter [9] und [10].

## PHP

Der nächste Kandidat für die Optimierung war PHP – ein äußerst geeigneter Kandidat, wie der folgende Abschnitt zeigen wird. Tatsächlich war die Optimierung von PHP ebenso trivial wie effektiv. Die einfache Installation eines PHP-Beschleunigers wie `eAccelerator` oder `Zend Optimizer` als PHP-Modul brachte gewaltige Geschwindigkeitssteigerungen. Unsere Wahl fiel auf `eAccelerator` [11], weil es kostenlose GPL-Software ist, welche die Konkurrenz nicht scheuen muss.

Zum besseren Verständnis soll die grundsätzliche Funktionsweise von `eAccelerator` hier kurz erläutert werden. Die Software besitzt zwei Hauptkomponenten, welche die Ausführung von PHP beschleunigen: Code-Optimierung und Caching. Zunächst analysiert `eAccelerator` den kompilierten Bytecode und versucht anschließend, ihn für eine höhere Geschwindigkeit zu optimieren. Danach wird der Code als Objekt im `shared memory` und/oder auf der Festplatte zwischengespeichert. Ist dies erstmalig geschehen, braucht es weitaus weniger Rechenprozesse (und somit Zeit), um den Code auszuführen. Angesichts der Menge an Code in TYPO3 ist es leicht zu verstehen, warum `eAccelerator` einen enormen Performance-Zuwachs bringt.

Wir installierten `eAccelerator` und aktivierten ihn in der zentralen PHP-Konfigurationsdatei:

## CONFIG

```
php.ini:
eaccelerator.enable = 1
```

Listing 10

Das Ergebnis des anschließend durchgeführten Tests war deutlich. Die Optimierung und das Caching des Codes ergaben 21 Req/s für den „FC Bigfeet“ – die fünffache Performance. Der Vollständigkeit halber wollten wir den Speicherbedarf von `eAccelerator` untersuchen. Das Ziel war, eine Grundregel für TYPO3 zu finden. Ein von `eAccelerator` mitgeliefertes Statistik-Werkzeug (`eaccelerator.php`) lieferte uns die gewünschten Werte: Das Frontend von „FC Bigfeet“ brauchte etwa 8 MB, das Backend zusätzliche 18 MB Speicher. Nach weiteren Tests mit anderen Websites konnten wir einige Grundregeln formulieren:

1. Je mehr Extensions eingesetzt werden, desto mehr Speicher wird benötigt.
2. Für eine durchschnittliche Website kann ein Minimalwert von 32 MB angenommen werden ( in der Datei `php.ini`: `„eaccelerator.shm_size = "32"“`).
3. Bei mehr als einer Website kann der Speicherbedarf minimiert werden, indem für alle Websites dasselbe Quellverzeichnis (`typo3_src`) genutzt wird – ein Muss für Server, die viele Websites hosten!
4. Um das Potential von Servern mit sehr wenig Speicher voll auszuschöpfen, können selten gebrauchte Objekte durch kleine TTL-Werte aus dem Speicher entfernen werden (in der Datei `php.ini`: `„eaccelerator.shm_ttl = "1800"“`).

Etwas rigoroser ist es, `eAccelerator` für das Backend ganz auszuschalten, mit folgender Konfiguration:

## CONFIG

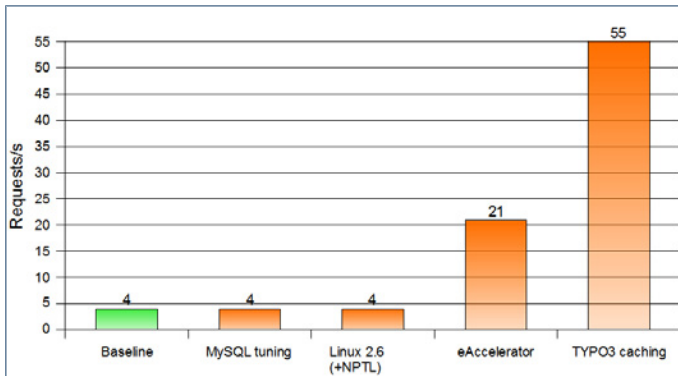
```
./typo3/.htaccess
php_flag eaccelerator.enable 0
```

Listing 11

## TYPO3

Abschließend werfen wir noch einen Blick auf TYPO3 selbst. Die TYPO3-Entwickler haben im Laufe der Jahre eine Menge getan, um die Geschwindigkeit zu verbessern. Doch TYPO3 ist eine komplexe Anwendung und jede Funktion erzeugt Last. Aber auch ohne die Optimierung von Code lässt sich die Performance von TYPO3 in der Regel verbessern. Die Nutzung der Caching-Optionen (Page Caching) bescherte uns enorme Performance-Steigerungen. Unsere Benchmarks offenbarten eine Geschwindigkeit von 55 Req/s, also eine 2,5-fache Steigerung gegenüber Seiten ohne Caching. Wichtig ist, auch bei Extensions auf eine Aktivierung des Caching zu achten.

Sehr hilfreich war auch die Suche nach Extensions, die entweder kein Caching zulassen, bei jeder Anfrage ausgeführt werden oder solchen, die das HTML kurz vor der Ausgabe umwandeln. Diese waren in der Regel große Performance-Killer. Wendet man bei jeder Extension die oben vorgestellte Methode zur Messung von Antwortzeiten an, gestaltet sich die Suche nach den üblichen Verdächtigen aber recht einfach.



Wie hier deutlich zu sehen ist, bietet TYPO3-Caching den größten Geschwindigkeitsvorteil.

## Fazit

Nach einer ganzen Menge erfolgreicher, aber auch ergebnisloser Anpassung haben wir es letztendlich geschafft, unsere Server-Performance von 4 Req/s auf 55 Req/s zu erhöhen (siehe Diagramm). Das ist 14-mal schneller im Vergleich zur Ausgangslage und ein ordentlicher Erfolg. Im Vergleich zur maximalen Leistungsfähigkeit des Webservers von 300 Req/s ist dies jedoch noch immer sehr wenig. Um in diese Bereiche vorzudringen, müsste man über Loadbalancer, Cluster oder Proxy-Server nachdenken.

Als Maßnahme zur Optimierung scheint Caching auf den meisten Ebenen der beste Weg zu sein. In unserem Fall waren dies der eAccelerator sowie der TYPO3-Cache. Das wiederum setzt eine gewisse Größe des RAM-Speichers voraus, was uns zurück an den Ausgangspunkt unseres Artikels führt: Um gewisse Hardware-Investitionen kommt man einfach nicht herum.

## Links und Literatur

- [1] Apache HTTP Server Benchmarking Tool: <http://httpd.apache.org/docs/2.0/programs/ab.html>
- [2] Linux Kernel-Vergleich: 2.6.4 vs. 2.4.25: [http://www.2cpu.com/articles/98\\_1.html](http://www.2cpu.com/articles/98_1.html)
- [3] Kernel-Vergleich: Web-Serving auf 2.4 und 2.6: <http://www-128.ibm.com/developerworks/linux/library/l-web26/?ca=dgr-Inxw07KernelCompare>
- [4] BSD- und Linux-Benchmarking: <http://bulk.fefe.de/scalability/>
- [5] MySQL Optimierung: <http://dev.mysql.com/doc/refman/4.1/en/optimization.html>
- [6] A Practical Look at the MySQL Query Cache: <http://dev.mysql.com/tech-resources/articles/mysql-query-cache.html>
- [7] Lighttpd-Webserver : <http://www.lighttpd.net/>
- [8] FastCGI: <http://www.fastcgi.com/>
- [9] Hinweise zur Apache Performance: <http://httpd.apache.org/docs/1.3/misc/perf-tuning.html>
- [10] Apache Performance-Tuning: <http://www.xs4all.nl/~thomas/apachecon/PerformanceTuning.html>
- [11] eAccelerator: <http://eaccelerator.net/>

### DER AUTOR

Michael Scharkow und Steffen Müller sind Studenten der Freien Universität Berlin am Institut für Publizistik- und Kommunikationswissenschaft und arbeiten freiberuflich mit TYPO3. Seit sie die Website ihres Instituts 2002 mit TYPO3 neu gestalteten, engagieren sie sich aktiv in der TYPO3-Community. Sie sind Mitglieder im TYPO3-Security-Team und haben einige Extensions entwickelt.