Geheime Profi-Tricks für Schnellere Websites!

Leseprobe

Das vollständige Buch finden Sie auf
http://schnellere-websites.bitpalast.net

Peter Debik M.A.

In zahlreichen Abschnitten dieses Buchs werden Beispiele zu Skripten und Serverbefehlen gezeigt. Alle Beispiele stammen aus der Praxis und sind getestet. Manche Befehle sind mächtige Werkzeuge, die mit wenigen Buchstaben die Einstellungen einer Software so verändern können, dass sie nicht mehr fehlerfrei arbeitet. Informieren Sie sich deshalb stets vor der Nutzung eines Befehls, wie der Befehl genau funktioniert, welche Optionen er hat (z.B. bestimmte Parameter, die sein Verhalten steuern) und was er bewirkt. Besonders hilfreich ist dazu der Linux-Befehl `man` (="Manual", zu Deutsch „Handbuch"), mit dem Sie sich viele wichtige Befehle erklären lassen können.

Wenn Sie sich nicht sicher sind, welche Auswirkungen eine bestimmte Einstellung hat, führen Sie sie nicht aus oder führen Sie sie nur auf einem Computer aus, auf dem Sie alles risikofrei testen können.

Die Informationen in diesem Buch wurden sorgfältig erarbeitet und überprüft. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Verlag und Autor übernehmen keine juristische Verantwortung oder Haftung für vielleicht verbliebene Fehler und deren Folgen.

Geheime Profi-Tricks für schnellere Websites!
ISBN 3-98050723-8
EAN 9783980507233

Copyright © 2014 Bitpalast GmbH. Alle Rechte vorbehalten.
Hintergrundbild des Umschlags: © alphaspirit - Fotolia.com
1. Auflage 2014

Lektorat, Satz, Gestaltung: Bitpalast GmbH.

2.4 Crontab- und atd-Zeitplanung

Viele Serveraufgaben werden in Linux selbsttätig zeitgesteuert erledigt. Zum Beispiel kann eine MySQL-Datenbank-Sicherung automatisch angefertigt werden, können Log-Dateien rotiert, verwaiste Dateien entfernt und Anti-Viren-Updates aus dem Internet geladen werden. Solche Aufgaben werden regelmäßig bearbeitet, manche mehrfach täglich, manche sogar mehrfach stündlich.

Mit dem Linux `at`-Befehl können Sie ein Skript zu genau einem vorher geplanten Zeitpunkt, d.h. nicht wiederholt, selbsttätig ausführen lassen. `at`-Aufträge können mit dem Linux `atd`-Daemon so gesteuert werden, dass sie zum geplanten Zeitpunkt nur dann starten, wenn die CPU-Last des Servers einen festgelegten Wert unterschreitet (siehe 2.4.3 „Termin-Aufträge nur bei niedriger CPU-Last ausführen“). Damit können Sie verhindern, dass CPU-Zeit für Wartungsskripte verwendet wird, wenn die CPU gerade mit voller Kraft Website-Aufgaben bearbeiten soll.

Wiederkehrende zeitgesteuerte Aufträge hingegen werden in Linux vom `crontab`-Programm verwaltet und heißen „Crontab Jobs“ oder kurz „Cronjobs“. Selbstredend sollten Sie Cronjobs möglichst nur zu Zeiten einplanen, in denen ansonsten wenig auf dem Server los ist, damit Ihrer Website ausreichend Host-Ressourcen zur Verfügung bleiben und sie auch dann schnell geliefert werden kann, wenn der Host zusätzlich die Cronjobs abarbeitet. Cronjobs können nämlich nicht darauf achten, dass die CPU unbelastet ist, wenn sie gestartet werden. Manche Cronjobs finden häufig statt, zum Beispiel alle 15 Minuten, und können deshalb nicht in Zeiten verlegt werden, in denen der Host von anderen Aufgaben unbelastet ist. Mit einer geschickten Terminplanung können Sie trotzdem auch diese Aufträge so einrichten, dass Sie CPU und RAM schonen.

In „Shared Hosting“ Webspace-Paketen stehen häufig keine Cronjobs und schon gar kein `atd`-Zeitplaner zur Verfügung. Dort werden die zeitgesteuerten Aufträge zentral vom Webhosting-Anbieter verwaltet. Mieter „virtueller privater Server“ („VPS“) können zusätzlich eigene Cronjobs und meistens auch den `atd`-Daemon nutzen, Betreiber dedizierter Server können und müssen alle `at`- und `crontab`-Jobs selbst planen. Website-Betreiber sind sich selten bewusst, dass schon nach der Bereitstellung eines neuen Servers selbsttätig zeitgesteuerte Aufträge vorhanden sein können. Weil Cronjobs die Leistung Ihres Hosts beeinflussen, sollten Sie prüfen, welche Aufträge zu welcher Zeit geplant sind.

Cronjobs werden u.a. in der Datei `/etc/crontab` gespeichert. Änderungen an `/etc/crontab` müssen nicht ausdrücklich an `crontab` mitgeteilt werden. `crontab` schaut selbst nach, ob sich die `/etc/crontab`-Konfiguration geändert hat. Die dort eingetragenen Aufträge werden sofort nach jeder Änderung der Datei und auch nach jedem Neustart des Hosts neu geladen. Aufträge können aber auch nutzerbezogen in die Zeitsteuerung geladen werden. Die Jobliste des aktuell angemeldeten Nutzers sehen Sie mit dem Befehl `crontab -l`.

Ich empfehle, dass Sie keine einzelnen crontab-Befehle in die Zeitplanung einfügen, sondern sich eine Textdatei aller Cronjobs erstellen und diese Datei als Konfiguration komplett in `crontab` laden. So können Sie alle Cronjobs zentral in einer Datei verwalten und behalten den Überblick. Mit `crontab [Dateiname]` können Sie eine neue `crontab`-Konfiguration laden, mit `crontab -e` die aktuelle Konfiguration löschen. Falls Sie die

`crontab`-Syntax noch nicht kennen, finden Sie dazu in vielen Internetbeiträgen und in der `man crontab` Anleitung alle weiteren Informationen.

Beispiel: Sie möchten sehen, welche Cronjobs geplant sind. Geben Sie dazu auf der Linux Kommandozeile den Befehl `crontab -l` ein. Eine typische Jobliste könnte in etwa so aussehen:

```
24 1 * * * /usr/local/bin/php /home/rottbi/unlinkoldbackups.php
26 1 * * * rm -R /var/tmp/*
10 2 * * * /usr/bin/find /home -name bayes_toks | xargs rm -f
46 4 * * * perl /db_backup.pl --compress=1 --backup_dir=/home/rottbi
3 5 * * * /home/rottbi/restart_vsapd.sh
12 12,13,14,22,23 * * * /usr/local/urchin5/bin/urchinctl start
1,16,31,46 * * * * /bin/chmod -R 775 /home/meinkonto
2,17,32,47 * * * * /bin/chown -R meinkonto:apache /home/meinkonto
*/20 * * * * /usr/local/sbin/restart_apache
```

Die Zahlen und Wildcard-Asteriske `*` legen fest, wann die Skripte ausgeführt werden. Wildcards werden häufig verwendet, sind aber vor allem im Minuten- und Stundenbereich gefährlich für die CPU-Last, weil sie veranlassen, dass Aufträge sehr häufig gestartet werden.

2.4.1 Aufträge ohne Wildcard planen

Mit `crontab` können Sie Aufträge planen, die in festen zeitlichen Abständen bearbeitet werden sollen, indem Sie eine Wildcard durch einen Schrägstrich und der Zahl an Zeiteinheiten, die zwischen zwei Programmausführungen verstreichen sollen, schreiben. Die Angabe `*/15` in der `crontab`-Konfiguration würde einen Auftrag alle 15 Minuten ab Beginn der vollen Stunde ausführen. Das führt jedoch dazu, dass mehrere gleichartige mit Wildcard geplante Aufträge zur gleichen Zeit gestartet werden[55] und dadurch vorübergehend hohe Last auf dem Server erzeugen. Hohe Last verlangsamt andere Vorgänge und ist deshalb schlecht für Ihre Website.

Beispiel: Die PHP-Skripte `meins.php`, `deins.php` und `unsers.php` sollen alle 15 Minuten ausgeführt werden. Normalerweise schreiben Anwender dafür in die `crontab`-Konfiguration:

```
*/15 * * * * /usr/local/bin/php /home/meinkonto/meins.php
*/15 * * * * /usr/local/bin/php /home/meinkonto/deins.php
*/15 * * * * /usr/local/bin/php /home/meinkonto/unsers.php
```

Syntaktisch ist das richtig. Aber für die Serverlast ist es eine Katastrophe. Denn mit dieser Cronjob-Planung werden die drei Skripte zur gleichen Zeit ausgeführt. Je nach Last, die

[55] Crontab-Aufträge, die zur gleichen Zeit geplant sind, werden ungefähr zur gleichen Zeit ausgeführt, aber nicht ganz genau pünktlich. Da Skripte jedoch meist einige Sekunden, manche einige Minuten laufen, befinden sich regelmäßig zeitgesteuerte Prozesse trotz etwas unterschiedlicher Startzeiten gleichzeitig in der Prozessliste und belasten die CPU stark.

jedes einzelne Skript erzeugt, steigt die Serverlast um 0, 15, 30 und 45 Minuten jeder vollen Stunde stark an. Zu diesen Zeiten werden alle anderen Vorgänge auf dem Server verlangsamt.

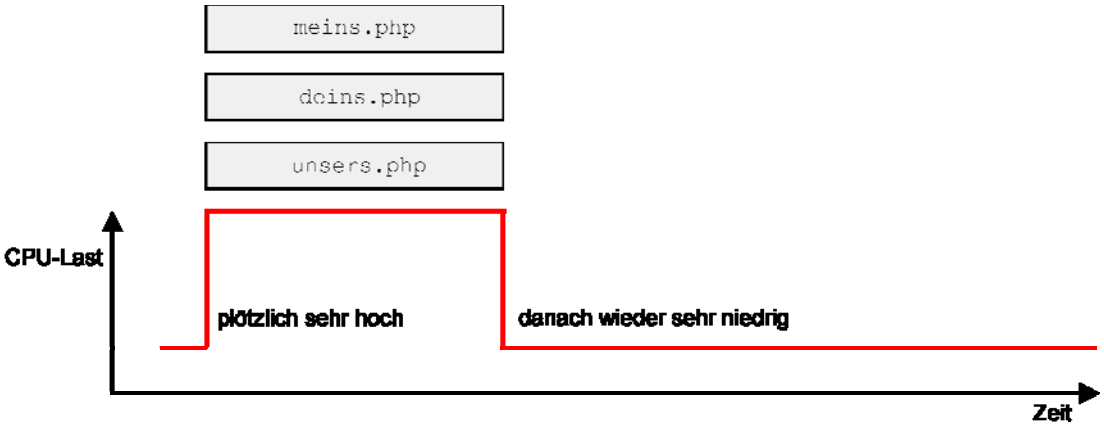

Abbildung 8: Gleichzeitig startende Programme erzeugen plötzlich eine hohe CPU-Last

Auf einem Host gibt es aber nicht nur eine handvoll zeitgesteuerter Vorgänge, sondern viele Dutzend. Die bequeme `*/`-Schreibweise kann deshalb zu sehr hoher Last zu einer bestimmten Zeit führen, während zu anderen Zeiten kaum Last entsteht. Für Ihre Website bedeutet das, dass Surfer gelegentlich eine sehr schnelle Website erleben, gelegentlich aber auch eine langsame. Zu den 15-minütigen Aufträgen des Beispiels können tägliche, wöchentliche, monatliche hinzu kommen, die noch mehr CPU-Zeit beanspruchen.

Es ist deshalb viel besser, den genauen Ausführungszeitpunkt von Aufträgen festzulegen. Eine gute Zeitplanung automatisch zeitgesteuerter Aufträge hilft, jederzeit ausreichend Serverkapazitäten frei zu halten. Die Laufzeit von Skripten ist meistens bekannt. Für unser Beispiel nehmen wir an, dass die Skripte je ungefähr 2 ¼ Minuten laufen. Deshalb wird jedes Skript zwar weiterhin alle 15 Minuten ausgeführt, aber mit einem Abstand von zwei Minuten zueinander. So muss die CPU nicht mehrere zeitintensive Aufgaben gleichzeitig erledigen:

```
1,16,31,46 * * * * /usr/local/bin/php /home/meinkonto/meins.php
3,18,33,48 * * * * /usr/local/bin/php /home/meinkonto/deins.php
5,20,35,50 * * * * /usr/local/bin/php /home/meinkonto/unsers.php
```

Außerdem kann eine unüberlegte Terminplanung schnell zu unerwünscht hoher Dauer-Rechenlast führen. Die drei unscheinbaren Konfigurationszeilen dieses Beispiels verteilen zwar die CPU-Last gleichmäßig, starten die drei Skripte aber jede Woche insgesamt 2 016 Mal (24 Stunden x 4 Ausführungen/Stunde x 7 Tage x 3 Skripte = 2 016 Skriptausführungen), folglich pro Jahr mehr als 105 000 Mal!

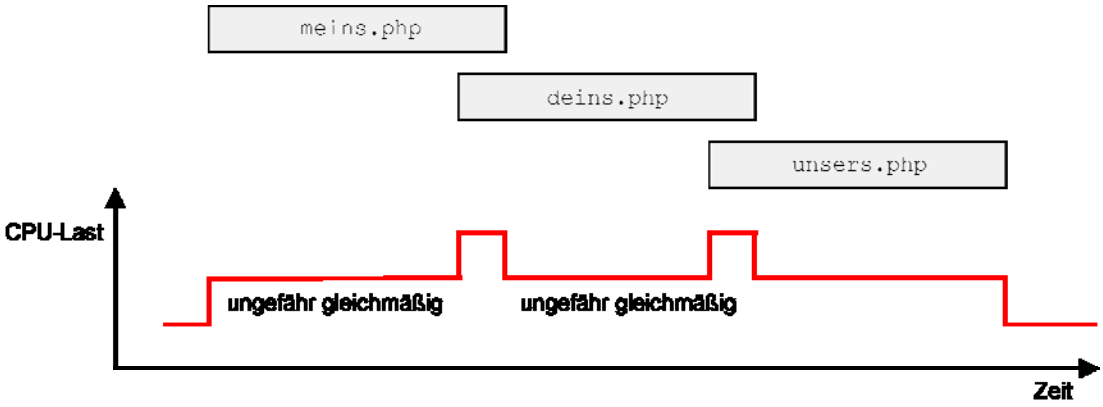

Abbildung 9: Unterschiedliche Skript-Startzeiten entlasten die CPU

2.4.2 Aufträge nur ausführen, wenn es wirklich nötig ist

Vielleicht ist es gar nicht nötig, Aufträge an jedem Tag und rund um die Uhr auszuführen. In verkehrsstarken Zeiten sollte schließlich möglichst viel Rechenzeit für die Bearbeitung von HTTP-Anfragen verfügbar sein. In verkehrsschwachen Zeiten von typischerweise 1 bis 5 Uhr morgens könnten Aufträge ausgeführt werden, die viel Rechenlast erzeugen. Zum Beispiel könnten Datensicherungen angefertigt, Logdateien rotiert und neue Antiviren-Definitionen heruntergeladen werden.

Vielleicht genügt es auch, regelmäßige Aufträge nur alle zwei Stunden laufen zu lassen? Außerdem sollten Sie genau überlegen, ob ein Skript wirklich so häufig ausgeführt werden muss, wie Sie anfangs dachten. Vielleicht entdecken Sie dabei, dass ein Skript eigentlich nur mittwochs laufen muss, weil sich an anderen Wochentagen Daten gar nicht so oft geändert haben, dass sie selbsttätig gewartet werden müssen?

Je seltener ein Skript ausgeführt werden muss, desto mehr Rechenzeit bleibt über den Tag verteilt für andere Aufgaben verfügbar. Für ein Rechenbeispiel nehmen wir einfach mal an, dass Ihre Skripte nur zweimal täglich zu jeder Viertelstunde, das `unsers.php`-Skript nur mittwochs laufen soll:

```
1,16,31,46 1,13 * * * /usr/local/bin/php /home/meinkonto/meins.php
3,18,33,48 3,15 * * * /usr/local/bin/php /home/meinkonto/deins.php
5,20,35,50 5,17 * * 3 /usr/local/bin/php /home/meinkonto/unsers.php
```

Statt dem Server mit der Wildcard-Syntax 2 016 Skriptausführungen pro Woche aufzubürden, kommt die geänderte Konfiguration dieses Beispiels mit nur 64 Skriptausführungen pro Woche aus (2 Stunden/Tag x 4 Ausführungen/Stunde x 7 Tage + 2 Stunden/Tag x 4 Ausführungen/Stunde x 1 Tag = 64 Skriptausführungen), pro Jahr also nur noch ungefähr 3 300 Ausführungen statt vorher 105 000. Das heißt, fast 97 % der vormals für Wartungsarbeiten unnötig genutzten CPU-Zeit steht nun Ihrer Website zur Verfügung.

2.4.3 Termin-Aufträge nur bei niedriger CPU-Last ausführen

In den vorherigen Abschnitten haben Sie gelernt, wie Sie die Rechenlast zeitgesteuerter Aufträge verringern und gleichmäßig über den Tag verteilen können. Dadurch sollte in den meisten Fällen stets ausreichend CPU-Zeit und RAM für die Bearbeitung von Anfragen an den Webserver zur Verfügung stehen. Noch besser aber wäre es, wenn selbsttätig zeitgesteuerte Aufgaben vor der Ausführung nachschauten, ob Ihr Host tatsächlich gerade unbelastet ist (siehe 1.2.3.2 „Was ist CPU-Last und wie misst man sie?“).

Dazu können Sie den Linux `atd`-Daemon verwenden. Beachten Sie, dass `atd` kein Befehl ist, sondern ein Hintergrunddienst, der die vom `at`-Befehl geplanten Aufgaben ausführt. Damit ist es möglich, den Ausführungszeitpunkt eines Programms zu planen, aber auch einen CPU-Last Wert einzutragen, der unterschritten werden muss, damit das geplante Programm tatsächlich ausgeführt wird. Sie können damit verhindern, dass Wartungsarbeiten auf Ihrem Host ausgeführt werden, wenn die CPU gerade stark belastet ist und eigentlich dafür gar keine Zeit hat. Dadurch bleibt CPU-Zeit für die wirklich wichtigen Aufgaben frei: Die möglichst schnelle Lieferung Ihrer Website an die Anwender.

Damit `atd` genutzt werden kann, muss der Daemon in der Startkonfiguration Ihres Linux-Servers eingetragen werden. Wenn Sie den Daemon ohne Parameter starten, führt er die mit dem Linux `at`-Befehl geplanten Jobs nur aus, wenn die CPU-Last kleiner 0,8 (kleiner 80 %) ist. Falls Sie ein Mehrprozessor-System verwenden, macht das natürlich wenig Sinn. In einem Quad-Core-System – also einem Computer mit vier Prozessorkernen – müssten Sie `atd` mit dem CPU-Last-Schwellenwert von 3,2 starten, um Aufträge nur bei einer CPU-Last kleiner 80 % auszuführen. Das können Sie beim Start des Daemons mit dem `l`-Parameter angeben.

Beispiel: Der atd-Daemon wird so gestartet, dass die mit `at` geplanten Aufträge nur bei einer CPU-Last kleiner 73 % ausgeführt werden:

```
/etc/init.d/atd -l 0.73 start
```

Nachdem der Daemon gestartet ist, richten Sie einen `at`-Auftrag ein, der das Skript `meins.php` um 18.30 Uhr starten soll:

```
at -f "/usr/local/bin/php /home/meinkonto/meins.php" -v 18:30
```

Weil die Auftragswarteschlange von `atd` gesteuert wird, wird der für 18.30 Uhr geplante Auftrag nur dann ausgeführt, wenn die CPU-Last um 18.30 Uhr kleiner 0,73 ist.

`at`-Befehle können nicht für regelmäßig auszuführende Jobs genutzt werden, sondern für einmalig auszuführende Jobs. Sie können aber hilfsweise `at`-Befehle in Cronjobs verwenden und damit regelmäßig neue `at`-Befehle in die Warteschlange stellen. Dadurch profitieren Sie einerseits von der `crontab`-Fähigkeit, Aufträge selbsttätig zeitgesteuert auszuführen, andererseits von der `atd`-Eigenschaft, Aufträge nur bei ansonsten geringer CPU-Last auszuführen.