Linux Server Hacks

Rob Flickenger

HAC #22 hdparm: Parameter von IDE-Laufwerken optimieren

Holen Sie aus Ihrer IDE-Hardware die beste Performance heraus.

Wenn Sie ein Linux-System mit mindestens einer (E)IDE-Festplatte betreiben und noch nie von *hdparm* gehört haben, dann lesen Sie weiter!

Defaultmäßig verwenden die meisten Linux-Distributionen die voreingestellten Kernel-Parameter beim Zugriff auf Ihre IDE-Controller und Festplatten. Diese Einstellungen sind sehr konservativ und so gewählt, dass Ihre Daten unter allen Umständen geschützt sind. Aber wie viele von Ihnen bestimmt schon einmal festgestellt haben, ist »sicher« so gut wie nie identisch mit »schnell«. Und bei Anwendungen, die große Mengen von Daten bearbeiten, kann es nie schnell genug gehen.

Wenn Sie aus Ihrer IDE-Hardware die größte Performance herausholen wollen, sehen Sie sich das Kommando *hdparm(8)* an. Es sagt Ihnen nicht nur, wie gut sich Ihre Festplatten gerade verhalten, sondern lässt Sie auch nach Herzenswunsch daran drehen.

Es sollte darauf hingewiesen werden, dass diese Kommandos unter gewissen Umständen UNERWARTETE DATENVERLUSTE VERURSACHEN können! Verwenden Sie sie also auf eigene Gefahr! Machen Sie zumindest ein Backup und bringen Sie Ihren Rechner in den Einzelbenutzer-Modus, bevor Sie weitermachen.

Fangen wir an! Da wir nun im Einzelbenutzer-Modus sind (den wir im Hack »Verzicht auf ein Konsolen-Login« [Hack #2] besprochen haben), wollen wir einmal herausfinden, wie gut die Performance der primären Festplatte gerade ist:

```
hdparm -Tt /dev/hda
```

Sie sollten hier ungefähr Folgendes sehen:

```
/dev/hda:
Timing buffer-cache reads: 128 MB in 1.34 seconds =95.52 MB/sec
Timing buffered disk reads: 64 MB in 17.86 seconds = 3.58 MB/sec
```

Was sagt uns das? Das -T bedeutet, dass das Cache-Speichersystem getestet werden soll (z.B. Speicher-, CPU- und Puffer-Cache). Das -t besagt, dass eine Statistik über die fragliche Platte ausgegeben werden soll, und zwar mit

Daten, die nicht im Cache liegen. Beide Angaben zusammen, ein paar mal hintereinander im Einzelbenutzer-Modus ausgeführt, geben Ihnen eine Vorstellung von der Performance Ihres Festplatten-I/O-Systems. (Die obigen Zahlen stammen von einem PII/350/128M Ram mit EIDE-Festplatte; Ihre Zahlen werden davon abweichen.)

Aber selbst mit abweichenden Zahlen ist 3,58 MB/Sek. kläglich für die obige Hardware. Hatte die Anzeige zu der Festplatte nicht etwas von 66 MB pro Sekunde gesagt!!?!? Was ist hier los?

Finden wir mehr darüber heraus, wie Linux diese Platte anspricht:

```
# hdparm /dev/hda

/dev/hda:
multcount = 0 (off)
I/O support = 0 (default 16-bit)
unmaskirq = 0 (off)
using_dma = 0 (off)
keepsettings = 0 (off)
nowerr = 0 (off)
readonly = 0 (off)
readahead = 8 (on)
geometry = 1870/255/63, sectors = 30043440, start = 0
```

Das sind die Voreinstellungen. Nett und sicher, aber nicht unbedingt optimal. Was heißt hier überhaupt 16-Bit-Modus? War es damit nicht beim 386er schon vorbei!?

Diese Einstellungen funktionieren garantiert auf jeder Hardware, auf die man sie loslässt. Aber da wir wissen, dass wir etwas Besseres als eine staubige, acht Jahre alte 16-Bit-Multi-IO-Karte haben, betrachten wir einmal die interessanten Optionen:

multcount

Abkürzung für »multiple sector count«. Steuert, wieviele Sektoren in einem einzigen I/O-Interrupt von der Platte gelesen werden. Fast alle modernen IDE-Platten unterstützen diese Option. Die Manpage behauptet:

> Wenn dieses Feature aktiviert ist, reduziert sich normalerweise der Aufwand des Betriebssystems für den Festplatten-I/O um 30-50%. Auf vielen Systemen vergrößert sich auch der allgemeine Datendurchsatz um 5-50%.

I/O support

Das ist etwas Größeres. Dieses Flag steuert, wie Daten vom PCI-Bus an den Controller übergeben werden. Fast alle modernen Controller-Chipsets unterstützen den Modus 3 bzw. einen synchronen 32-Bit-Modus, einige sogar asynchronen 32-Bit-Modus. Wenn Sie diese Option aktivieren, wird das Ihren Durchsatz höchstwahrscheinlich verdoppeln (siehe unten).

unmaskirq
: Wenn Sie das einschalten, erlauben Sie Linux, andere Interrupts während der Abarbeitung eines Platteninterrupts einzublenden. Was bedeutet das? Es erlaubt Linux, anderen Interrupt-bezogenen Aufgaben nachzugehen (z. B. dem Netzwerkverkehr), während es darauf wartet, dass Ihre Platte die Daten holt, nach denen sie gefragt wurde. Das sollte die allgemeine Antwortzeit des Systems verbessern. Aber Vorsicht: Nicht alle Hardware-Konfigurationen werden damit umgehen können. Sehen Sie sich dazu die Manpage an!

using_dma
: DMA kann eine knifflige Angelegenheit sein. Wenn es Ihnen gelingt, Ihren Controller und die Platte dazu zu bewegen, einen DMA-Modus zu benutzen, tun Sie es! Ich habe allerdings schon mehr als einen hängenden Rechner gesehen, während mit dieser Option gespielt wurde. Sehen Sie sich auch hierzu die Manpage an!

Probieren wir einige Turbo-Einstellungen:

```
# hdparm -c3 -m16 /dev/hda

/dev/hda:
setting 32-bit I/O support flag to 3
setting multcount to 16
multcount = 16 (on)
I/O support = 3 (32-bit w/sync)
```

Sehr gut! 32-Bit klingt vernünftig. Einige mehrfache Leseoperationen könnten auch funktionieren. Starten wir erneut den Benchmark:

```
# hdparm -tT /dev/hda

/dev/hda:
Timing buffer-cache reads: 128 MB in 1.41 seconds =90.78 MB/sec
Timing buffered disk reads: 64 MB in 9.84 seconds = 6.50 MB/sec
```

Hhmm, fast der doppelte Plattendurchsatz, ohne große Experimente! Unglaublich.

Aber Moment, es gibt noch mehr: Wir haben noch keine Interrupts maskiert, kein DMA verwendet oder auch nur einen halbwegs anständigen PIO-Modus benutzt! Natürlich wird es riskanter, diese einzuschalten. Die Manpage erwähnt Multiword DMA-Mode2, probieren wir es einmal damit:

```
# hdparm -X34 -d1 -u1 /dev/hda
```

Leider scheint das auf diesem speziellen Rechner nicht unterstützt zu werden (er hängt wie eine NT-Kiste, die eine Java-Anwendung ausführt). Nach einem Reboot (wieder im Einzelbenutzer-Modus), bin ich hier angelangt:

```
# hdparm -X66 -d1 -u1 -m16 -c3 /dev/hda

/dev/hda:
setting 32-bit I/O support flag to 3
setting multcount to 16
```

```
setting unmaskirq to 1 (on)
setting using_dma to 1 (on)
setting xfermode to 66 (UltraDMA mode2)
multcount = 16 (on)
I/O support = 3 (32-bit w/sync)
unmaskirq = 1 (on)
using_dma = 1 (on)
```

Dann wieder der Benchmark:

```
# hdparm -tT /dev/hda

/dev/hda:
Timing buffer-cache reads: 128 MB in 1.43 seconds =89.51 MB/sec
Timing buffered disk reads: 64 MB in 3.18 seconds =20.13 MB/sec
```

20,13 MB/Sekunde. Eine deutliche Verbesserung zu den winzigen 3,58 MB/Sekunde, mit denen wir angefangen haben.

Haben Sie bemerkt, dass wir die Schalter *-m16* und *-c3* erneut angegeben haben? Das passiert deswegen, weil sich Linux Ihre *hdparm*-Einstellungen nach einem Reboot nicht merkt. Fügen Sie die oben genannte Zeile auf jeden Fall zu Ihren Skripten in */etc/rc.d/** hinzu, wenn Sie sicher sind, dass das System stabil ist (am Besten, nachdem Ihr *fsck* läuft; ein umfangreicher Check des Dateisystems mit einem übermutigen Controller mag eine gute Methode sein, die Entropie signifikant zu erhöhen, aber sicher nicht, um ein System zu verwalten. Jedenfalls nicht, ohne eine schmerzvolle Grimasse dabei zu ziehen).

Sollten Sie *hdparm* auf Ihrem System nicht finden (normalerweise in */sbin* oder */usr/sbin*), können Sie sich den Quellcode dafür bei *http://metalab.unc.edu/pub/Linux/system/hardware/* besorgen.

HACK #67 Turbo-schnelle ssh-Logins

Noch schnellere Logins von der Kommandozeile aus.

Wenn Sie gerade den vorherigen Hack gelesen haben (»Schnelle Logins mit ssh-Client-Schlüsseln« [Hack #66]), dann kennen Sie erst die halbe Lösung! Selbst mit Client-Schlüsseln müssen Sie weiterhin unnötigerweise immer wieder `ssh server` eingeben, wenn Sie mit *ssh* reinkommen wollen. Früher, in den dunklen, unsicheren und unaufgeklärten Zeiten von *rsh*, gab es ein obskures Feature, das ich sehr geliebt habe, das (noch) nicht nach *ssh* portiert wurde. Man konnte einen symbolischen Link von */usr/bin/rsh* auf eine Datei mit dem Namen des entsprechenden Servers erstellen, und *rsh* war so schlau zu erkennen, dass es nicht als *rsh* aufgerufen wurde, und stellte in diesem Fall eine Verbindung mit dem Rechner her, mit dessen Namen es aufgerufen wurde.

Natürlich ist es trivial, so etwas in der Shell zu implementieren. Erzeugen Sie eine Datei namens *ssh-to* mit folgenden beiden Zeilen darin:

```
#!/bin/sh
ssh `basename $0` $*
```

(Um den Ausdruck `basename $0` herum befinden sich Backticks.) Kopieren Sie sie irgendwo in Ihren PATH (falls *~/bin* noch nicht existiert oder nicht in Ihrem PATH ist, dann sollte es dort aber hin) und richten Sie symbolische Links zu all Ihren Lieblings-Servern ein:

```
$ cd bin
$ ln -s ssh-to server1
$ ln -s ssh-to server2
$ ln -s ssh-to server3
```

Um nun mit *ssh* auf server1 zu gelangen (unter der Annahme, dass Sie Ihren öffentlichen Schlüssel wie zuvor beschrieben hinüberkopiert haben) geben Sie einfach »server1« ein, und Sie gelangen auf magische Weise in eine Shell auf server1, ohne »ssh« und ohne Ihr Passwort einzugeben. Jenes `$*` am Ende erlaubt Ihnen, beliebige Kommandos in einer einzelnen Zeile auszuführen (statt eine neue Shell zu öffnen), z.B.:

```
server1 uptime
```

(Dieser Befehl zeigt lediglich die Uptime, die Anzahl der Benutzer sowie die durchschnittliche Last auf `server1` an, schließt dann die Verbindung. Wickeln Sie eine *for*-Schleife drumherum und lassen Sie sie über eine Liste von Servern laufen, um eine Statusübersicht all Ihrer Rechner zu erhalten.)

Ich glaube, das ist die schnellste Möglichkeit, *ssh* zu benutzen, abgesehen von einem Alias mit nur einem Buchstaben, der Ihnen die Arbeit abnimmt (was übertrieben Hack-mäßig, unwartbar und unnötig ist, obwohl es einige Leute anscheinend beeindruckt):

```
$ alias a='ssh alice'
$ alias b='ssh bob'
$ alias e='ssh eve'
...
```

Auf jeden Fall hat *ssh* viele Optionen, die es zu einem sehr flexiblen Werkzeug bei der sicheren Navigation zwischen einer beliebigen Anzahl von Servern machen. Wenn wir es nur dazu bewegen könnten, dass es sich einloggt und auch noch den Rechner für Sie repariert... dann hätten wir einen echten Hack.

Siehe auch:

- Schnelle Logins mit ssh-Client-Schlüsseln [Hack #66]
- Effektive Verwendung eines ssh-Agenten [Hack #68]

HACK #79 Verteilen der Serverlast via ringförmigem DNS

Leiten Sie Datenverkehr mit ringförmigem DNS an mehrere Server gleichzeitig weiter.

Wenn Sie eine besonders beliebte Site betreiben, erreichen Sie früher oder später den Zeitpunkt, an dem Ihr Server keine weiteren Anfragen bedienen kann. In der Welt der Webserver wird das *Slashdot-Effekt* genannt, und es ist kein schöner Anklick (ähm, Anblick). Durch mehr RAM, schnellere Prozessoren, Festplatten und Busse erreichen Sie zwar eine kurzfristige Besserung, aber am Ende werden Sie wahrscheinlich feststellen, dass ein einziger Rechner unmöglich die gesamte Arbeit erledigen kann, die nun einmal anfällt.

Eine Möglichkeit, die Grenzen eines monolithischen Servers zu sprengen, liegt darin, die Last auf viele Rechner zu verteilen. Durch Hinzufügen eines zweiten (oder dritten) Servers können Sie nicht nur die Performance erhöhen, sondern auch die Stabilität des Netzwerks. Wenn Sie immer einen oder zwei Ersatzrechner laufen lassen, kann ein Computer die Arbeit übernehmen, sobald ein anderer Probleme bereitet, ohne dass es zu Ausfallzeiten kommen muss.

Die vielleicht einfachste Art, die Belastung durch öffentlichen Datenverkehr zu verteilen, liegt darin, die Verteilungsarbeit von der Allgemeinheit erledigen zu lassen. Durch die Magie des sogenannten ringförmigen DNS (engl. »Round Robin DNS«), können eingehende Anfragen an einen einzelnen Rechnernamen auf eine beliebige Anzahl von IP-Adressen umgeleitet werden.

In BIND 9 ist das genauso einfach wie das Hinzufügen mehrerer A-Einträge für einen einzelnen Rechner. Angenommen, wir benutzen z.B. Folgendes in der Zonendatei für oreillynet.com:

```
www 60 IN A 208.201.239.36
www 60 IN A 208.201.239.37
```

Wann immer nun ein Rechner im DNS nach www.oreillynet.com schaut, sieht er ungefähr in der Hälfte aller Fälle dieses Ergebnis:

```
rob@caligula:~$ host www.oreillynet.com
www.oreillynet.com has address 208.201.239.36
www.oreillynet.com has address 208.201.239.37
```

In der anderen Hälfte der Fälle sieht er Folgendes:

```
rob@caligula:~$ host www.oreillynet.com
www.oreillynet.com has address 208.201.239.37
www.oreillynet.com has address 208.201.239.36
```

Das funktioniert ziemlich gut, weil die meisten Anwendungen nur die erste vom DNS zurückgegebene Adresse benutzen. Da etwa die Hälfte der Anfragen auf je eine der beiden Adressen fällt, sollte die Last der beiden Server in etwa der halbierten Last eines einzelnen Servers entsprechen. Wir setzen den TTL-Wert niedrig an (auf 60 Sekunden), um zu verhindern, dass irgendwelche zwischengeschalteten, cachenden DNS-Server sich zu lange an einer Vari-

ante festbeißen, wodurch hoffentlich die Anzahl der Anfragen an beide Server mehr oder weniger gleich hoch sein dürfte.

Es ist nur dann sinnvoll, die Last zu verteilen, wenn alle Server denselben Informationsstand haben. Falls Ihre Daten jedoch nicht mehr synchronisiert sind, könnte ein Browser beim ersten Zugriff eine andere Variante der Webseite erhalten als beim zweiten, wenn der Benutzer die Seite erneut lädt. Wenn Sie so etwas verhindern möchten, lesen Sie in dem Hack »Teile des Dateisystems mit rsync synchronisieren« [Hack #41] nach.

Beachten Sie außerdem, dass eine echte IP-Übernahme (falls ein Rechner seine Pflichten nicht erfüllen kann) nicht gerade trivial ist. Wenn ein Server verschwindet, können Sie nicht das DNS ändern und darauf warten, dass es sich im ganzen Internet herumspricht. Dann brauchen Sie etwas, das sofort die Stelle des alten Servers einnehmen kann. Wie Sie das erreichen, können Sie in dem Hack »Billige IP-Übernahme« [Hack #63] nachlesen.

Siehe auch:

- Teile des Dateisystems mit rsync synchronisieren [Hack #41]
- Billige IP-Übernahme [Hack #63]