

## 15 NFS: Verteiltes Unix-Dateisystem

NFS ist die Abkürzung für *Network File System* (Netzwerkdateisystem). Es ist unter Unix *die* Methode schlechthin, Dateisysteme von anderen Rechnern einzuhängen und so Dateien innerhalb eines Netzes gemeinsam zu benutzen. Das Szenario dabei ist, dass ein oder mehrere NFS-Server Verzeichnisse mit Dateien vorhalten. Typischerweise sind dies Heimatverzeichnisse von Benutzern, Softwareverzeichnisse – etwa mit `/usr/local` – oder andere gemeinsam zu benutzende Daten. Ein NFS-Server muss nicht unbedingt ein ausgewachsener Unix-Rechner sein. NAS – Network Attached Storage – einst unter dem Namen NFS-Appliance Spielwiese größerer Rechenzentrums-umgebungen – ist mittlerweile auch in kleineren Bereichen vorzufinden und selbst preislich für das kleine Büro zu Hause erschwinglich.<sup>1</sup> All diese Geräte machen nichts anderes als Ihre mehr oder weniger zahlreichen Platten via NFS zu exportieren.

Gemessen an Samba ist NFS der Veteran schlechthin, er hat mehr als 20 Jahre auf dem Buckel, damit also sogar mehr als die Urversion von Windows. Sun Microsystems hat NFS in den frühen 80ern mit seiner lediglich intern erhältlichen Version 1 ins Leben gerufen. Der Nachfolger NFS Version 2 kam 1985 auf den Markt und lag damals verschiedenen Unices bei. 10 Jahre lang haben Hersteller mit dessen Schwächen gekämpft und eigene Ergänzungen ins Leben gerufen, bevor NFS Version 3 das Licht der Welt erblickte. Diese Version ist zwar performanter, aber hat wie seine Vorgänger nicht allzu viel am Hut mit Authentifizierung (alles beruht auf IP-Adressen oder vermeintlichen DNS-Adressen), Sicherheit (alle Daten gehen unverschlüsselt übers Netz) und echter gemeinsamer Datenbearbeitung. Die relativ aktuelle Version 4 ist da vielversprechender, aber komplizierter zu handhaben und noch nicht weit verbreitet. Aus den beiden letzten Gründen beschränke ich mich im Folgenden auf die Vorgängerversionen NFSv2 und NFSv3.

---

1. Wer NFS unter Windows benutzen will, sei auf die SFU (*Services for Unix*) verwiesen.

## 15.1 Konfigurieren der NFS-Server-Dienste

Die serverseitige Konfiguration von NFS benötigt schon ein paar Schritte. Also los geht's.

Als Erstes sollten Sie überprüfen, ob im Kernel die Unterstützung für NFS (**File systems** --> **Network File Systems** --> **NFS file system support**) und die Unterstützung für den Server (**NFS server support**) und mindestens Version 3 (**Provide NFSv3 server support**) eingeschaltet ist, was bei einem Kernel vom Distributor der Fall sein wird. Abseits des Kernels sind drei Dämonen wichtig, die alle durch Init-Skripte gestartet werden:

- Der Portmapper `portmap`, der alle Aufrufe vom Client an die serverseitig registrierten Dienste (siehe Abschnitt 6.6) delegiert<sup>2</sup>,
- der Mount-Dämon `rpc.mountd`, der für die Anfragen, Dateisysteme ein- oder auszuhängen, verantwortlich ist,
- und schließlich `rpc.nfsd`, der für alle eigentlichen dateibasierten Anfragen zuständig ist und sie an das NFS-Kernel-Modul weiterleitet. Dies erledigt heutzutage die Schwerstarbeit der Dateitransaktionen. In der Prozessliste sehen Sie daher keinen `rpc.nfsd`<sup>3</sup>.

Die ersten beiden Dienste unter Linux können durch den TCP-Wrapper geschützt werden. Sie sollten an dieser Stelle die Gelegenheit ergreifen und den Zugriff gemäß Ihren Vorstellungen einschränken, die entsprechenden Zeilen fangen mit `portmap`: und `mountd`: an.

Was müssen Sie nun tun, um Verzeichnisse zu exportieren? Legen Sie als Root zunächst eine Datei namens `/etc/exports` (Modus 644) an. Jede Zeile darin hat folgendes Format:

- (a) Zu exportierendes Verzeichnis
- (b) Definition von Rechnern, die das Verzeichnis einhängen dürfen
- (c) Optionen in runden Klammern angegeben

In (b) kann zum Beispiel ein einzelner Hostname (IP-Adresse oder Hostname), eine Domäne sein (wie `*.bin.ichmuede.de` oder `ich*.aberauch.de`) oder ein Netz in CIDR-Notation bzw. in Netz/Netzmasken-Schreibweise stehen. Zwischen (a) und (b) muss mindestens ein Leerraumzeichen (»Whitespace«) sein, zwischen (b) und (c) darf keins sein. Nach (c) dürfen jeweils nach einem Leerraumzeichen wieder Paare mit (b) und (c) stehen. Damit Ihnen das nicht zu trocken erscheint, ein Beispiel aus der Praxis:

- 
2. Die Kommunikation bei NFS-Version 2 und 3 findet bei Linux ausschließlich über RPC-Aufrufe statt.
  3. Nur eine Anzahl von `nfsd` mit eckigen Klammern und 0 Byte residentem Hauptspeicherverbrauch (RSS).

---

```

/data/mp3    riker(ro, all_squash) 192.168.110.13(rw, sync)
/home       data(rw, async, root_squash) picard(rw, async, root_squash)
/usr/local  *.stng.wa(ro, sync, no_root_squash)
/media/dvd  192.168.111.0/24(ro, async, root_squash)

```

---

Hier können die MP3s von dem Knoten riker lesend (ro) und von 192.168.110.13 auch schreibend (rw) eingehängt werden. Die Heimatverzeichnisse dürfen data und picard jeweils mit Schreibberechtigung einhängen. async bedeutet, dass der Server beim Schreiben über NFS sein »Ok« als Rückmeldung an den Client schickt, bevor dieser die Daten auf seine Platte geschrieben hat; das ist zwar normalerweise der Geschwindigkeit zuträglich, aber unter Umständen nicht der Datenintegrität. /usr/local dürfen alle Rechner der Domäne stng.wa<sup>4</sup> nur zum Lesen einhängen, in gleicher Weise ist /media/dvd exportiert an das Klasse-C-Subnetz 192.168.111.\*. Eine wichtige Besonderheit hat es mit root\_squash (Root zerquetschen/zermantschen) auf sich: Es bedeutet, dass Root auf dem Client nicht die via NFS vom Server exportierten Dateien als Root, sondern als unprivilegierter Benutzer nobody (UID 65534) liest, schreibt oder ausführt (dies ist die Voreinstellung). no\_root\_squash bildet Root auf Root ab, all\_squash wie bei riker in der ersten Zeile bildet alle Benutzerkennungen vom Client als nobody auf dem Server ab.

Wenn Sie nun eine eigene /etc/exports erstellt haben, sollten Sie als Erstes sicherstellen, dass der Portmapper auf dem Server gestartet ist. Das geschieht bei allen Distributionen mit:

---

```
/etc/init.d/portmap start
```

---

Die danach zu startende Init-Datei für mountd und nfsd heißt unter Suse-Linux nfsserver, unter Red Hat nfs und unter Debian nfs-kernel-server. Falls der Start nun glatt gegangen ist, können Sie mit exportfs -v kontrollieren, ob alles so exportiert ist wie beabsichtigt. Änderungen an /etc/exports machen Sie am besten mittels exportfs -rv bekannt.

So weit, so gut. Nun wollen Sie sicherlich auch lernen, wie man diese Verzeichnisse auf dem Client einhängt.

## 15.2 Maßnahmen auf dem NFS-Client

Auf dieser Seite ist die einfache Konfiguration sehr unspektakulär. Auch hier muss im Kernel die Unterstützung für NFS und Ihre gewünschte Version eingeschaltet sein (**File systems** --> **Network File Systems** --> **NFS file system support** +

---

4. Die Top-Level-Domäne wa = Weltall war zum Zeitpunkt der Drucklegung leider noch nicht vergeben.

**Provide NFSv3 client support**), was beim Distributor-Kernel der Fall sein wird. Häufig wird clientseitig der Portmapper vergessen. Starten Sie ihn und stellen Sie vom NFS-Server aus mit

---

```
rpcinfo -p nfsclient
```

---

sicher, dass der Client erreichbar ist und durch keine Firewall oder durch den TCP-Wrapper blockiert wird. Die Ausgabe sollte ungefähr so aussehen:

---

program	vers	proto	port	
100000	2	tcp	111	portmapper
100000	2	udp	111	portmapper

---

Was jetzt noch fehlt, ist ein `mount`-Kommando auf dem Client, um das vom NFS-Server freigegebene Verzeichnis nach `/mnt` einzuhängen:

---

```
mount -t nfs nfsserver:/data/mp3 /mnt
```

---

`/mnt` ist ein lokales, am besten leeres Verzeichnis. `-t nfs` ist meistens überflüssig. Falls das Kommando nicht geklappt hat, überprüfen Sie vom Client mittels

---

```
showmount -e nfsserver
```

---

ob Sie das entsprechende Verzeichnis auch einhängen dürfen oder ob irgendetwas den Zugriff behindert. Falls das Kommando nicht sofort zurückkommt oder Sie mit der Fehlermeldung konfrontiert werden, dass der Portmapper nicht kontaktiert werden konnte, überprüfen Sie die TCP-Wrapper-Konfiguration auf dem Server und schauen Sie in die Log-Dateien des Servers. Sie können die Kommunikation bzw. die zur Verfügung gestellten NFS-Protokolle mit `rpcinfo -p nfsserver` überprüfen, dabei sollten Sie ungefähr folgende Ausgabe erhalten:

---

program	vers	proto	port	
100000	2	tcp	111	portmapper
100000	2	udp	111	portmapper
100003	2	udp	2049	nfs
100003	3	udp	2049	nfs
100003	4	udp	2049	nfs
100003	2	tcp	2049	nfs
100003	3	tcp	2049	nfs
100003	4	tcp	2049	nfs
100024	1	udp	32959	status
100021	1	udp	32959	nlockmgr
100021	3	udp	32959	nlockmgr
100021	4	udp	32959	nlockmgr
100024	1	tcp	32788	status
100021	1	tcp	32788	nlockmgr

---

---

100021	3	tcp	32788	nlockmgr
100021	4	tcp	32788	nlockmgr
100005	2	udp	812	mountd
100005	2	tcp	813	mountd
100005	3	udp	812	mountd
100005	3	tcp	813	mountd

---

Uns interessieren hier vorrangig die Zeilen mit `nfs` und `mountd` am Ende. In der zweiten Spalte sehen Sie bei `nfs` die NFS-Version, in der dritten das Protokoll, gefolgt von der Portnummer, an den der Portmapper die Aufrufe weiterleitet, und schließlich den Programmnamen. Hier werden also NFS-Version 2 und 3 via UDP und TCP für das NFS- und `mount`-Protokoll angeboten (neben NFS-Version 4 via TCP), NFS über seinen Standardport 2049, `mountd` über 812 und 813. Der Zugriff auf diese Ports sowie auf die anderen im Beispiel sollte nicht durch eine hostbasierte Firewall blockiert sein. Kommen Sie allerdings nie auf die Idee, diese Ports zum Internet hin zu öffnen, es sei denn, Sie haben ein besonders exhibitionistisches Bedürfnis, was Ihre Dateien und die Sicherheit Ihres NFS-Servers angeht. Seien Sie auch sonst sehr vorsichtig, wann und wo Sie überhaupt den NFS-Dienst starten. Wie anfangs erwähnt, NFSv2 und v3 haben intrinsische Schwächen.

Normalerweise werden zwischen Client und Server die NFS-Version und das Transportschichtprotokoll ausgehandelt. Clientseitig lassen sich Vorgaben machen<sup>5</sup>, falls man andere Präferenzen hat:

---

```
mount -o nfsvers=2,udp,intr nfsserver:/data/mp3 /mnt
```

---

Hier wird für `/data/mp3` NFS-Version 2 über UDP vorgegeben.

Sie sehen dort aber noch eine weitere Option namens `intr`. Damit hat es Folgendes auf sich: `hard`, `soft` und `intr` sind dazu da, clientseitig etwaige Probleme des NFS-Servers in den Griff zu bekommen: `hard` bedeutet, dass der Client bis zum Sankt Nimmerleinstag versucht, eine Antwort auf seine Dateioperation vom NFS-Server zu bekommen, wenn dieser oder sein angebotener Dienst »tot« ist. Damit hängt auch der Client, sobald er auf ein NFS-Verzeichnis nur zugreift. Sobald der Server allerdings wieder einsatzbereit ist, sollte alles wieder wie gehabt laufen. `intr` ist wie `hard`, nur ermöglicht es Benutzern, mit CTRL-C Ein-/Ausgabeoperationen auf Dateien im NFS abzubrechen. Bei `soft` hingegen versucht der NFS-Client eine gewisse Zeit zu warten, bevor das aufrufende Programm einen I/O-Fehler zurückgeliefert bekommt.

Sie werden wahrscheinlich fragen, was nun am besten ist. Darauf gibt es leider keine allgemein gültige Antwort. Voreingestellt ist `hard`, was zwar meistens

---

5. Möchten Sie auf der Serverseite nicht alle NFS-Versionen für `mount` unterstützen, schauen Sie in der Hilfeseite `rpc.mountd(8)` nach.

eine gute Lösung ist, aber problematisch wird, wenn Sie keinen zuverlässigen Server bzw. kein zuverlässiges Netz und/oder eine Menge Clients dranhängen haben. In dem Fall ärgern sich vermehrt die Benutzer, weil auf ihrem Client auch alles steht, sobald diese ein importiertes Verzeichnis »anfassen«. `intr` ließe Ihnen dann noch die Möglichkeit, CTRL-C zu drücken und ein Stückchen weiter zu kommen, und ist damit meistens zu bevorzugen. `soft` eignet sich, wenn Sie – wie in manchen nicht anstrebenswerten Umgebungen – viele Verzeichnisbäume von verschiedenen Rechnern kreuz und quer eingehängt haben.

Egal, was Sie benutzen: Sie sollten Ihren NFS-Server niemals nach Lust und Laune hoch- und runterfahren. Abgesehen von den beschriebenen temporären Problemen, kann sich als Folge davon auch der Client etwas verschlucken (die Meldung »stale NFS file handle« ist ein typisches Symptom), so dass hier sogar ein Neustart nötig werden kann. Glücklicherweise sind diese Fälle unter Linux seltener geworden.

Eine Linux-Eigenart ist die Blockgröße: Sie sollten diese der Performance zuliebe hochdrehen:

---

```
mount -o nfsvers=3, tcp, intr, rsize=8192, wsize=8192 nfserver: /data/mp3 /mnt
```

---

Die Werte fürs Lesen (`rsize`) und Schreiben (`wsize`) werden immer als Potenzen von zwei angegeben. 8 KByte (8192) sind ein guter Wert, unter Umständen mag 32 KByte auch helfen (32768). Falls Sie keine Schreibberechtigung auf `/data/mp3` haben, erübrigt sich die Angabe von `wsize`. Wenn es Ihnen wirklich auf Durchsatz ankommt, sollten Sie hiermit experimentieren, ebenso mit der NFS-Version und mit UDP versus TCP.

## 15.3 /etc/fstab und Automounter

Falls Sie permanent Verzeichnisse via NFS exportieren wollen, werden Sie sicherlich schon die Init-Skripte in den entsprechenden `rc?.d`-Verzeichnissen (siehe Abschnitt 3.1.3) Ihres NFS-Server verankert haben. Auf dem Client möchten Sie aber wahrscheinlich auch nicht Ihre Verzeichnisse immer manuell mit `mount` einhängen, oder?

Wie Sie der Überschrift entnehmen können, gibt es hier zwei Alternativen.

### 15.3.1 /etc/fstab

Um sicherzugehen, dass ein via NFS exportiertes Verzeichnis gleich während des Boot-Prozesses eingehängt wird – ein so genannter *fixed mount* –, kann man dieses Verzeichnis in der Dateisystemtabelle `/etc/fstab` eintragen:

---

```
nfserver: /data/mp3 /mnt nfs auto, rw, nfsvers=3, tcp, intr, rsize=8192, wsize=8192 0 0
```

---

Dieser Eintrag würde entsprechend dem `mount`-Befehl aus dem vorhergehenden Beispiel funktionieren. Beachten Sie, dass Sie den Schwung NFS-Optionen auch übergeben können. Die letzten beiden Felder sind 0, da weder `dump` noch `fsck` auf dem NFS-Client relevant sind.

### 15.3.2 Automounter

Allerspätstens in Rechenzentren sollte anstelle eines solchen *fixed mount* ein *Automounter* benutzt werden, der dafür sorgt, dass bei einem Zugriff auf ein NFS-Dateisystem dieses automatisch ein- und später wieder ausgehängt wird.

Dies hat mindestens zwei Vorteile:

- Feste Einträge in `/etc/fstab` erhöhen die Abhängigkeit vom NFS-Server quasi doppelt:
  - Falls der NFS-Server beim Booten des Clients nicht erreichbar ist, wird Letzterer gar nicht oder nur verzögert hochfahren.
  - Generell werden beim automatischen Einhängen bei Nichtbenutzung von NFS-Verzeichnissen diese nach einer gewissen Zeitspanne wieder ausgehängt, womit die verhängnisvolle Abhängigkeit bei Problemen des NFS-Servers für die Zeitdauer der Nichtbenutzung vom NFS zumindest vom Tisch ist. Wohlgemerkt nur dann, denn sobald das Verzeichnis eingehängt werden soll, und der NFS-Server ist nicht erreichbar, haben Sie wieder das gleiche Problem.
- Geringerer Pflegeaufwand für `/etc/fstab` versus Automounter-Konfiguration.

Diese Argumente sollten Sie überzeugt haben, selbst in Ihrer Heimatumgebung einen Automounter für NFS zu benutzen. Es ist auch nicht besonders schwer, diesen zu konfigurieren.

Grundsätzlich existieren unter Linux zwei Möglichkeiten, NFS-Verzeichnisse automatisch einzuhängen: Den BSD-Automounter `amd`, den es nicht nur unter Linux gibt, und das Kernel-basierte `autofs`, das vorzuziehen ist, vorzugsweise in Version 4.

Neben der Kernel-Unterstützung (**File systems --> Kernel automounter version 4 support**) sind Werkzeuge auf der Anwendungsebene Ihres Linux notwendig, das Paket hört auf den Namen `autofs`. Zentrales Programm darin ist `automount`, daneben ist die globale Konfigurationsdatei `/etc/auto.master` entscheidend. Aber alles der Reihe nach.

`automount` erwartet auf der Kommandozeile folgende Parameter: Ein am besten leeres Verzeichnis, unter dem alles eingehängt wird, eine Typbeschreibung der Konfigurationsdatei (=Map-Datei) und diese Datei selbst. Also beispielsweise:

---

```
automount /home file /etc/auto.home
```

---

`file` als Map-Typ besagt, dass die nachfolgende Map-Datei – hier `/etc/auto.home` – eine echte Datei ist. Alternativ lassen sich Map-Dateien aus Verzeichnisdiensten wie LDAP oder NIS benutzen.

Weiter geht es mit dem obigen Beispiel: Wenn die Map-Datei `/etc/auto.home` aus folgendem Inhalt besteht:

---

```
picard ncc1701d: /export/home/picard
kirk ncc1701a: /export/home/kirk
```

---

und Sie sich entweder per `cd` ins Heimatverzeichnis `/home/picard` auf dem Client begeben oder versuchen, dort eine x-beliebige Datei »anzufassen«, hängt automount das Verzeichnis `/export/home/picard` von `ncc1701d` lokal auf `/home/picard` ein und versucht, es nach einer normalerweise fünfminütigen Inaktivität wieder auszuhängen.

Wenn man dies so weiterspinnt, ist dieses Beispiel eigentlich eine Anleitung, wie man es nicht machen soll. Wenn Sie es mit mehr als einem Benutzer zu tun haben und sich die Heimatverzeichnisse jedes Benutzers auf einem anderen NFS-Server befinden, haben Sie nicht nur einen erhöhten Administrationsaufwand zur Pflege der Map-Datei, auch die Wahrscheinlichkeit des Hängens beim Client wird größer, je mehr Sie sich auf die Verfügbarkeit mehrerer NFS-Server verlassen. Daher: Versuchen Sie es besser mit nur *einem* NFS-Server. Dies erleichtert auch das Backup der Heimatverzeichnisse – Sie machen doch regelmäßige Backups, oder?

Sollen all diese Verzeichnisse von einem Server kommen, bietet sich Folgendes in `/etc/auto.home` an:

---

```
* ncc1701: /export/home/&
```

---

Dies besagt, dass alle (deswegen der Stern) Heimatverzeichnisse von der Maschine `ncc1701` zu holen sind. »&« bedeutet, dass derselbe Name, der intern für den Stern ersetzt wurde, auch für »&« gilt. `cd /home/worf` auf Ihrem NFS-Client hängt also `/export/home/worf` von `ncc1701` ein.

Nun wissen Sie also, wie das Programm automount funktioniert. Was nun zum Ganzen noch fehlt, ist, den Aufruf von automount plus Map-Datei und anderer Parameter im Boot-Prozess Ihres Rechners zu verankern. Hierfür gibt es das Init-Skript `/etc/init.d/autofs`, das die globale Konfigurationsdatei `/etc/auto.master` einliest und entsprechend jeweils einen automount-Prozess mit zugehörigen Map-Dateien wie unsere `/etc/auto.home` mitsamt etwaiger Parameter startet. Ein Beispiel für `/etc/auto.master`:

---

```
/net /etc/auto.net -soft
/home /etc/auto.home -rw, intr, rsize=8192, wsize=8192
```

---

Die erste Zeile startet automount mit `/etc/auto.net` und der NFS-Option `-soft` fürs Einhängen unter `/net`, die zweite Zeile ein weiteres automount mit `/etc/auto.home` und dem halben Rattenschwanz an NFS-Optionen für das Verzeichnis `/home`. Vergessen Sie nicht, bei Bedarf `/etc/init.d/autofs` in den `rc?.d`-Verzeichnissen zu verlinken.

Eine nützliche Eigenschaft des Linux-automount will ich Ihnen nicht verschweigen: Sie können ab autofs-Version 4 auch Programme übergeben. Sie haben bereits in der `/etc/auto.master` oben die Datei `/etc/auto.net` gesehen. Dies ist ein Shell-Skript, das den meisten Distributionen beiliegt.<sup>6</sup> In der Prozessliste sieht dies ungefähr so aus, achten Sie auf das Wort `program`:

---

```
/usr/sbin/automount /net program /etc/auto.net -intr
```

---

Es ermöglicht nach einem `cd` zu `/net/nfsserver/`, die von `nfsserver` exportierten Verzeichnisse zu erforschen. Einhängen lässt sich jeder `x`-beliebige NFS-Server, sofern Sie diesen erreichen können und dieser den Portmapper Ihrer Maschine.

## 15.4 Mehr gefällig?

Dieses Kapitel konnte Ihnen aus Platzgründen nur einen Einstieg in die Tiefen des NFS geben. Wie erwähnt, ist NFS Version 4 ganz unter den Tisch gefallen, auf die zahlreichen Optionen von `mount` – hilfreich ist die Hilfeseite `nfs(5)` – wurde nur ansatzweise eingegangen. Das wichtige Thema Performance ist zu kurz gekommen, und das NFS-Dateilocking (etwa: Sperren) sowie andere Problemquellen wie Sicherheit im Umgang mit NFS habe ich kaum angerissen. Aber auch automount unter Linux hat seine Tücken. In allen Problemfällen kann es hilfreich sein, Dateisysteme per Hand auszuhängen, etwaige Prozesse, die dies verhindern, mit `lsof -p Prozess-ID` oder `lsof Verzeichnisname` aufzuspüren und sie nötigenfalls abzuschließen.

Auf jedenfall habe ich Ihnen genug Lernstoff als Grundlage vermittelt, womit Sie ein sehr gutes Stück weit kommen sollten. Wie bei vielen anderen Systemthemen gilt auch bei NFS: Erfahrung macht den Meister.

Es soll an dieser Stelle nicht verschwiegen werden, dass es auch Alternativen zu NFS gibt. Eine Alternative besonders zum Vorhalten wenig veränderlicher Daten via NFS, die nur gelesen werden sollen, und in Anbetracht sinkender Festplattenpreise und steigendem Netzdurchsatz, ist `rsync`, das Ihnen im nächsten Kapitel vorgestellt wird.

---

6. Sie sollten sich `/etc/init.d/autofs` nicht anschauen. Es ist ein grauenvolles Shell-Skript, das versucht, `/etc/auto.master` zu parsen und auf unsaubere Weise etwaige Map-Dateien aus NIS, NIS+ oder LDAP zu holen; etwas, das unter anderen Betriebssystemen eigentlich Aufgabe des Dämons ist.

Es gibt aber auch aufwendigere Alternativen, die vor längerer Zeit ins Leben gerufen wurden: In Umgebungen, wo es sich vom Aufwand her lohnt, wird AFS (früher: Andrew File System) oder DFS<sup>7</sup> (Distributed File System) benutzt, die in den Bereichen Authentifizierungen, Backups, Sicherheit, Zugriffsrechte und Verfügbarkeit entscheidend besser sind. Ein lokaler Puffer auf den AFS-Clients hilft besonders bei kleinen Dateien. Jedoch ist der Einsatz aufgrund der erforderlichen Infrastruktur erst bei einer wirklich großen Anzahl von Clients sinnvoll. Das freie OpenAFS (<http://www.openafs.org>) basiert auf der ursprünglich von IBM (vorher von Transarc) kommerziell vertriebenen Version.

---

7. Es gibt auch ein Microsoft-DFS, das weitaus weniger mächtig ist und keinerlei Verbindung zum klassischen DFS hat.