OSL
UNIX Pfadfinder

NFS unter Solaris

Server und Client

Version 1.3

Copyright und Handelsmarken

Beschränkungen

Versionen dieses Dokumentes

Version	Datum	Author	e-mail	Inhalt / Änderungen
1.0	20.01.2004	ZM	pathfinder@osl-it.de	Erste Version
1.1	02.02.2004	ZM	pathfinder@osl-it.de	Korrekturen der Rechtschreibfehler
1.2	22.06.2004	HO	pathfinder@osl-it.de	OSL Hinweis, Copyright 2004
1.3	11.01.2007	HO	pathfinder@osl-it.de	Fehlersuche mehrere Netzwerkinterfaces

Inhaltsverzeichnis

1. Einführung

Dieser Artikel stellt sich zum Ziel, den Systemadministrator mit grundlegenden Merkmalen sowie spezifischen Besonderheiten des NFS-Dateisystems unter Solaris vertraut zu machen. Besonderes Augenmerk wird dabei auf die verschiedenen Mount-Optionen des NFS-Clients gelegt, wie sich dieser bei Nichterreichbarkeit des Servers verhalten soll.

Alle Konfigurationen wurden mit Solaris 9 als NFS-Server und -Client ausführlich getestet.

1.1 Allgemeines zum NFS-Protokoll

Das Network Filesystem (NFS) wurde als eine Client-Server-Anwendung entwickelt. Seine Implementation unterscheidet sich in einen Client-Teil, welcher Dateisysteme von anderen Rechnern importiert, und einen Server-Teil, welcher lokale Dateisysteme nach anderen Rechnern exportiert.

NFS ist ein sogenanntes zustandsloses Protokoll, was es robuster gegenüber Ausfällen bzw. Fehlern von Client oder Server macht, da es keine Zustände gibt, welche zu erhalten oder wiederherzustellen sind. Solche Zustände wären z.B. um welchen Client es sich handelt oder welche Dateien er gerade geöffnet hält. Aus Sicht des Clients ist ein Server, der nach einem Absturz wieder anläuft, nur ein sehr langsam arbeitender Server. Analog ist ein nach einem Absturz wiederanlaufender Client für den Server lediglich ein Client, der längere Zeit keine Anforderungen mehr gesandt hat.

Das NFS-Protokolldesign ist transportunabhägig, d.h. obwohl es ursprünglich nur für das UDP-Protokoll entwickelt wurde, kann es mittlerweile auch über TCP genutzt werden. Es ist zudem möglich, NFS auch über zahllose andere nicht IP-basierte Protokolle laufen zu lassen.

Bei Solaris ist zu beachten, dass es standardmäßig das erste vorhandene verbindungsorientierte Protokoll (im Regelfall TCP) für NFS einsetzt und lediglich bei expliziter Konfiguration von Server und Client (bzw. Nichtverfügbarkeit von TCP auf einem oder beiden Rechnern) das UDP-Protokoll nutzt.

1.2 Struktur und Arbeitsweise von NFS

Die Kommunikation zwischen Client und Server basiert auf sog. RPC-Anforderungen (remote-procedure-call) des Clients an den Server. Es handelt sich hierbei um 15 mögliche Anforderungen zur Operation an Dateien und Verzeichnissen (u.a Ermitteln und Setzen von Dateiberechtigungen, Lese- und Schreibvorgänge auf Dateien, Löschen, Anlegen, Umbennen von Dateien und Verzeichnissen).

Diese Anforderungen werden vom Client als eine Nachricht über das Netzwerk an den Server gesendet, welche diese als eine lokale Dateisystem-Operation ausführt. Das Resultat wird an den Client zurückgesandt, welcher dieses an den aufrufenden Prozess zurückgibt, so als ob das Resultat von einer lokalen Prozedur zurückgegeben wurde.

Jede Datei auf dem Server wird durch ein eindeutiges Filehandle identifiziert, mit dessen Hilfe die Clients sich auf die entsprechende Datei des Servers berufen. Dieses Filehandle wird erzeugt, sobald der Client eine sog. Lookup-Anforderung an den Server stellt. Findet der Server die Datei bzw. das Verzeichnis und stellt er sicher, dass der anfordernde Benutzer die nötigen Berechtigungen hat, gibt er das entsprechende Filehandle an den Client zurück. Das Filehandle identifiziert nun die Datei bei zukünftigen Zugriffs-anforderungen durch den Client.

Die Zustandslosigkeit von NFS wird nun dadurch erreicht, dass jede RPC-Anforderung auch alle zusätzlich benötigten Informationen enthält, um die Anforderung zu erfüllen.

Beispiel: Read-Anforderung enthält

- Benutzeridentifizierung
- Filehandle, auf welches sich die Anforderung bezieht
- Offset in der Datei, von dem ab gelesen werden soll

- Anzahl der zu lesenden Bytes

Mit diesen Angaben ist der Server in der Lage,

- die Datei zu öffnen
- zu überprüfen, ob der Benutzer die Datei lesen darf
- den richtigen Offset zu finden
- den gewünschten Inhalt zu lesen
- die Datei zu schließen.

In der Praxis werden aktuell im Zugriff befindliche Dateien im Cache zwischengespeichert. Falls hohe Aktivitäten das Ausschieben der Datei aus dem Zwischenspeicher bedingen, kann trotzdem jederzeit der Server über das Filehandle ausreichend Informationen erhalten, um die Datei erneut zu öffnen.

Außerdem speichert der Cache Wiederholungsversuche bereits eingegangener Anforderungen. Falls nämlich die Bestätigung für die Ausführung einer Anfrage den Client nicht erreicht, wird dieser nach einer Zeitüberschreitung die Anfrage erneut senden. Der Server kann jetzt durch seinen Cache feststellen, dass die gesandte Anforderung bereits ausgeführt wurde. Somit wird die Operation nicht wiederholt, stattdessen wird lediglich die Bestätigung zurückgesandt.

Ein Nachteil des zustandslosen Protokolls ist allerdings die Tatasache, dass hiermit das Setzen von Sperren nicht möglich ist. Um dennoch eine überwachte Sperrung zu ermöglichen, wird der Dämon lockd eingesetzt. Dieser Prozess läuft sowohl auf dem Client als auch auf dem Server. Wird eine Datei über das Netzwerk verfügbar gemacht, leitet der lockd des Client die von den Anwendungen angefordeten Dateisperren an den lockd des Servers weiter, der anschließend eine lokale Sperre auf die Datei aktiviert.

In Verbindung mit dem lockd wird in der Regel der Dämon statd eingesetzt, um den Status eines Systems zu überwachen. Der Prozess informiert die anderen Systeme, wenn ein System nach einem Absturz wieder erfolgreich hochgefahren wurde. Anschließend wird versucht, sämtliche Sperren wiederherzustellen.

Beide Dämonen werden durch das Skript /etc/init.d/nfs.client gestartet.

1.3 NFS-Versionen

Gegenwärtig unterstützt Solaris die NFS-Protokollversionen 2 und 3.

Wichtigste Unterschiede von Version 3 gegenüber Version 2:

- Puffergröße für Schreib- und Lesevorgänge standardmäßig 32 KB statt 8 KB
- asynchroner Schreibmodus (sofortige Empfangsbestätigung durch den Server)
- Einsparung von Aufrufen zur Ermittlung von Dateiattributen
- Dateien größer als 2 GB möglich
- Anlegen von Geräteknoten möglich (mknod)

Erläuterungen zum Schreibmodus (Abläufe):

synchron: Speichern der Schreibvorgänge im lokalen Cache des Clients, Weiterleiten an den Server, Empfangen der Schreibanforderungen, Ausführen des Schreibvorgangs, Senden der Bestätigung, Löschen des Cache

asynchron: Speichern der Schreibvorgänge im lokalen Cache des Clients, Weiterleiten an den Server, Empfangen der Schreibanforderungen, Speichern im lokalen Cache des Servers, Senden der Bestätigung usf.

Nach Abschluss des Schreibvorgangs bzw. Cache voll:
Senden einer Commit-Anforderung an den Server, Ausführung des Schreibvorgangs, Senden der Bestätigung, Löschen des Cache

2. Konfiguration eines NFS-Servers

Für Betrieb und Verwaltung von NFS sind auf dem Server folgende Prozesse notwendig:

- nfsd - Entgegennahme und Bearbeitung der Lese - und Schreibanforderungen der Clients
- mountd - Überprüfung von Mountanforderungen der Clients
- lockd - Überwachung von Dateisperren
- statd - Systemstatusüberwachung, Wiederherstellung von Sperrtabellen nach Rechnerabstürzen

2.1 Start des NFS-Servers

Im Regelfall wird beim Hochfahren eines Solaris-Systems lediglich die NFS-Client-Funktion (mit Start der Prozesse lockd und statd) durch das Run-Control-Skript /etc/init.d/nfs.client aktiviert.

Den Start des NFS-Server-Dämons übernimmt das Skript /etc/init.d/nfs.server, es startet die benötigten Prozesse nfsd und mountd allerdings nur, wenn in der Datei /etc/dfs/dfstab überhaupt Ressourcen freigegeben wurden. Zumindest ein entsprechender Eintrag ist also in dieser Datei erforderlich.

Die Einträge in /etc/dfs/dfstab besitzen folgendes Format:

```
share -F nfs [-o Optionen] [-d Beschreibung] pfadname
```

Folgende wichtige Optionen seien aufgeführt:

- log=*tag* Protokollierung aller Lese-und Schreibzugriffe mit in /etc/nfs/nfslog.conf definiertem*tag* als Ablageort der Protokolldateien (wird allerdings nur bei Start/Reboot des Rechners aktiviert!)
- nosub kein Mounten untergeordneter Verzeichnisse eines freigegebenen Verzeichnisses
- nosuid Verbot des Erstellens von Dateien mit setuid- bzw. setgid-Rechten
- ro=*liste* nur Lesezugriff, Möglichkeit der Angabe verschiedener Clients
- rw=*liste* Lese- und Schreibzugriff, Möglichkeit der Angabe verschiedener Clients
- root=*liste* root-Zugriff von aufgeführten Clients aus erlaubt

Hinweis zur Option »root«:

Standardmäßig wird dem Benutzer root nicht gestattet, sich unter dieser Benutzerkennung in einem nfs-gemounteten Dateiverzeichnis zu bewegen. Stattdessen wird ihm automatisch der Benutzer nobody zugewiesen, d.h. er kann nur Dateien lesen bzw. schreiben, auf die nobody die entsprechenden Rechte hat. Um root zu erlauben, sich unter der normalen root-Kennung im Verzeichnisbaum zu bewegen, muss die root-Option für den entsprechenden Client gesetzt werden.

Möchten Sie den NFS-Server mit einer eigenen Konfiguration starten, so lassen sich die entsprechenden Werte durch Editieren der Datei /etc/default/nfs ändern. Hier kann z.B. festgelegt werden, welches Protokoll (tcp, udp oder ALL) für NFS eingesetzt werden soll.

2.2 Befehle zur Administration

Ist der NFS-Server bereits gestartet, kann eine weiteres Verzeichnis auch direkt von der Befehlszeile aus freigegeben werden:

`share -F nfs [-o Optionen] [-d Beschreibung] pfadname`

Dieses Verzeichnis ist dann allerdings nur so lange freigegeben, wie die NFS-Serverprozesse nicht neu gestartet werden.

Zum Entfernen einer freigegebenen Ressource verwenden Sie den Befehl unshare:

`unshare pfadname`

oder entfernen Sie alle freigegebenen Ressourecen mittels:

`unshareall`

Umgekehrt lassen sich mittels **`shareall`** sämtliche Ressourcen wieder freigeben, allerdings nur diese, welche in /etc/dfs/dfstab angegeben sind (und nicht welche vormals über Befehlszeile freigegeben wurden).

Um einen Überblick über alle lokal derzeit freigebenen NFS-Ressourcen mit allen festgelegten Optionen zu erhalten, nutzen Sie:

```
share
```

Es lassen sich auch auf anderen Rechnern verfügbare NFS-Ressourcen anzeigen (allerdings ohne Anzeige festgelegter Optionen):

```
dfshares rechnername
```

Schließlich kann noch ermittelt werden, welche Client-Rechner derzeit welche Resourcen auf einem bestimmten NFS-Server gemountet haben:

```
dfmounts rechnername
```

Für statistische Informationen steht der Befehl nfsstat zur Verfügung, so z.B. zur Anzeige von NFS-Infos des Servers, d.h. Angaben über Lesevorgänge auf Dateiattribute, Lese- und Schreibvorgänge auf Dateien, Löschen, Anlegen, Umbennen von Dateien und Verzeichnissen usw.:

`nfsstat -ns`

oder zur Anzeige von RPC-Infos (Anzahl der Aufrufe, Anzahl zurückgewiesener Aufrufe):

`nfsstat -rs`

Zum Zurücksetzen der Statistiken auf Null nutzen Sie:

`nfsstat -z`

3. Konfiguration eines NFS-Clients

Das Einhängen von NFS-Freigaben geschieht mittels eines einzigen Befehls: mount. Dieser Befehlt hält eine große Anzahl von Optionen bereit, die es erlauben, das Verhalten des Clients optimal an bestimmte Situationen anzupassen.

3.1 Allgemeine Syntax des mount-Befehls

Um eine NFS-Freigabe in den lokalen Verzeichnisbaum einzuhängen, kann im einfachsten Fall die bekannte Vorgehensweise wie bei einem lokalen Dateisystem genutzt werden, lediglich bei der Pfadangabe der zu mountenden Ressource ist der NFS-Servername voranzustellen.

Beispiel des Anmountens der Freigabe /share/docu auf nfs-serv1 nach /mnt/docu:

```
mount -F nfs nfs-serv1:/share/docu /mnt/docu
```

Die Option -F nfs spezifiziert den Dateisystemtyp. Dieser braucht dann nicht mit angegeben zu werden, wenn die Datei /etc/dfs/fstypes einen entsprechenden Eintrag für NFS enthält.

Möchten Sie ein NFS-Dateisystem jedes Mal beim Hochfahren des Systems anmounten, so nehmen Sie einen entsprechenden Eintrag in der Datei /etc/vfstab vor:

```
#device         device          mount           FS      fsck    mount   mount
#to mount       to fsck         point           type    pass    at boot options
.
.
nfs-serv1:/share/docu  -        /mnt/docu       nfs     -       yes     -
```

Für spezielle Konfigurationen wird die allgemeine Syntax jedoch in den meisten Fällen nicht ausreichen, weshalb im Folgenden alle wichtigen Optionen des mount-Befehls intensiv beleuchtet werden sollen.

3.2 Option proto

proto=<netid> Auswahl des Transportprotokolls

Der Client kann festlegen, welches Transportprotokoll für den NFS-Mount verwendet werden soll. Folgende Voraussetzungen müssen allerdings erfüllt sein:

1. Das Protokoll muss in /etc/netconfig vorhanden sein
2. Der NFS-Server muss das Transportprotokoll unterstützen.

Standdardmäßig ist der NFS-Server unter Solaris mit Unterstützung für beide Protokolle gestartet. Wird die Option proto weggelassen, so wird TCP bevorzugt gewählt, nur bei Nichtunterstützung UDP.

3.3 Option retry

retry=<zahl> Anzahl von Wiederholungen nach einem misslungenen Mountversuch

Mit dieser Option kann angegeben werden, wie oft nach dem ersten fehlgeschlagenen Mountversuch nochmals versucht werden soll, das Dateisystem anzumounten. Standardmäßig ist hier der Wert 10000 vorgegeben. Abhängig vom Grund des nicht möglichen Mountens zeigten sich in Tests unterschiedlich lange Wartezeiten, bis die Anzahl an Wiederholungsversuchen abgearbeitet war:

1. Das zu mountende Verzeichnis ist auf dem Server nicht freigegeben

In diesem Fall kehrt mount sofort mit der Fehlermeldung *Permission denied* zurück.

2. Der nfsd ist auf dem Server nicht gestartet

Es erscheint in allen Fällen sofort die Meldung *RPC not registered, retrying.* Die anschließenden Wartezeiten wurden examplarisch wie folgt ermittelt:

retry=2 Abbruch nach 5 Sekunden

retry=3 Abbruch nach 15 Sekunden

retry=4 Abbruch nach 35 Sekunden

retry=5 Abbruch nach 1min 15s

retry=6 Abbruch nach 2min 35s

retry=7 Abbruch nach 4min 35s

retry=8 Abbruch nach 6min 35s

retry=10 Abbruch nach 10min 35s

retry=100 Abbruch nach 190min 35s

Aufgrund dieser ermittelten Werte lässt sich folgendes Schema zur Ermittlung der Wartezeit bei Anzahl von retry=anz bestimmen:

Wartezeit in Sekunden= $5 * (2^{anz-1} - 1)$ **für alle w >=2 und w<=6**

Wartezeit in Minuten= 2 + 2 * (anz - 6) + 35 Sekunden **für alle w>=7**

Die Wartezeit beim Standardwert von retry=10000 ist demnach 13d 21h 10m 35s!

3. Der Server ist im Netzwerk nicht erreichbar

Hier erscheint die erste Meldung nach exakt 1 Minute: *RPC bind failure, Timed out, retrying.* Der Hinweis *RPC bind failure, Timed out* erscheint in bestimmten (immer größer werdenden) Zeitabständen abhängig vom gewählten Wert für retry. Auch hier seien Testergebnisse aufgeführt:

retry=2 Abbruch nach 3min 5s

retry=3 Abbruch nach 4min 15s

retry=4 Abbruch nach 5min 35s

retry=7 Abbruch nach 12min 35s

Die Logik hinter diesem System zeigt große Ähnlichkeit zum oben beschriebenen. Hier addiert sich lediglich noch die Anzahl der Wiederholungsversuche + 1 (als Minuten) zur oben berechneten Wartezeit.

3.4 Optionen fg | bg

fg Mounten im Vordergrund (Standardeinstellung)

bg Mounten im Hintergrund

Ist der erste Mountversuch erfolglos, wird solange weiterversucht, das Dateisystem anzumounten, wie in der Option *retry* angegeben. Werden diese weiteren (evtl. erfolglosen) Mountversuche im Hintergrund durchgeführt, lässt sich vermeiden, dass der aufrufende Prozess die in der Datei /etc/vfstab anschließend aufgeführten Mountvorgänge blockiert.

Hinweise zum Einsatz von bg bei Einträgen in /etc/vfstab:

Nach Aufruf von mountall -r (z.B. beim Ausführen des Startskripts /etc/rc2.d/S73nfs.client) werden nach dem ersten erfolglosen Mountversuch die Mount-Befehle in den Hintergrund geschoben. Wird versucht, mehrere Freigaben desselben NFS-Servers zu mounten, so werden alle gleichzeitig in den Hintergrund geschickt (d.h. die Gesamtwartezeit bis zum Schieben in den Hintergrund beträgt pro Server höchstens 1 Minute).

Das Problem liegt nun in der Tatsache, dass sämtliche Mount-Befehle mit der bg-Option nach dem Schicken in den Hintergrund nur noch exakt 1 Minute in der Prozesstabelle vorhanden sind und anschließend beendet werden. Mit anderen Worten: Kann eine Freigabe mit bg-Option nach dem ersten Wiederholungsversuch nicht eingehangen werden, so werden weitere Mountversuche nicht mehr vorgenommen; eine Meldung über den beendeten Mountvorgang erfolgt nicht!

Der Grund für dieses Verhalten scheint darin zu liegen, dass beim Aufruf von mountall (einem Shellskript) der Befehl mount -a aufgerufen wird, welcher versucht, die in /etc/vfstab vorhandenen Freigaben einzuhängen. Hierzu werden die entsprechenden Mount-Befehle als »Kinder« des mount -a-Prozesses aufgerufen. Nach Ende der *RPC bind failure*-Zeitüberschreitung von 1 Minute wird mount -a aus der Prozesstabelle entfernt, die Kindprozesse erhalten als Vaterprozess den Prozess init. Nach exakt einer weiteren Minute werden die Kindprozesse anscheinend von init beendet.

Es liegt auf der Hand, dass ein derartiges Verhalten am eigentlichen Ziel des Hintergrundmountens vorbeigeht. Als Alternative bietet sich an, die gewünschten NFS-Mountvorgaben zum Einhängen im Hintergrund nicht in /etc/vfstab anzugeben, sondern die entsprechenden Mount-Befehle einzeln in einem Startskript anzugeben, welches nach /etc/rc2.d/S73nfs.client gestartet wird.

Ansonsten zeigten sich bei dem »normalen« Mount-Befehl mit bg-Option die erwarteten Resultate bei Nichterreichbarkeit des Servers. Die Wartezeiten für Werte der retry-Option entsprechen denen der im Abschnitt 3.3 aufgeführten.

3.5 Optionen intr | nointr

intr	Mountversuche und Anforderungen an den NFS-Server lassen sich durch einen Tastatur-Interrupt (Strg + C) abbrechen. (Standardeinstellung)
nointr	Abbrüche von Mountversuchen nicht erlaubt

Die Option *intr* ist unabhängig davon, ob das Dateisystem *hard* oder *soft* eingehängt wurde. Bitte beachten Sie, das sich bei *nointr* nicht nur der eigentliche Mountversuch nicht abbrechen lässt, sondern auch nach bereits erfolgten Mount Anforderungen auf diesem Dateisystem sich nicht mehr abbrechen lassen.

3.6 Optionen ro | rw

Auch der Client kann angeben, ob das Dateisystem im nur im Lese- oder Lese-/Schreibzugriff (Standard) eingehängt werden soll. Die Option *ro* wird allerdings automatisch gewählt, wenn der Schreibzugriff für den Client durch den Server verwehrt wird (siehe 2.1).

3.7 Option vers

vers=<zahl>	Angabe der NFS-Versionsnummer (Standardwert: 3)

Lesen Sie hierzu den Abschnitt 1.3 »NFS-Versionen«.

3.8 Optionen hard | soft

hard fortgesetzte Anfragen an den NFS-Server, bis dieser antwortet (Standardeinstellung)

soft Anfragen nach einer bestimmten Anzahl von Übertragungswiederholungen bzw. Zeitüberschreitung beenden und Fehlercode ausgeben

Zur Beachtung:

Diese Optionen (und alle im Folgenden aufgeführten) haben keinerlei Auswirkung auf das Verhalten des eigentlichen Mountvorganges, sondern beziehen sich auf Operationen in einem bereits eingehängten NFS-Dateisystem.

Die Manpages warnen vor dem Einsatz der soft-Option auf Dateisystemen mit Lese-/Schreib-Zugriff bzw. mit ausführbaren Dateien. Anwendungen auf solchen Systemen können unerwartete I/O-Fehler, beschädigte Dateien und Coredumps verursachen.

3.9 Option retrans

retrans=<anzahl> Anzahl von Übertragungswiederholungen, bevor eine Fehlermeldung ausgegeben wird

Diese Option hat nur Auswirkungen, wenn als NFS-Transportprotokoll UDP eingesetzt wird. Der Standardwert ist 5.

3.10 Option timeo

timeo=<zahl> Anzahl der Zehntelsekunden für Wiederholungsversuche eines Zugriffs auf den Server

Der Standardwert für TCP beträgt 600, für UDP ist er 11.

3.11 Testergebnisse zum Einsatz von hard/soft, retrans und timeo

Analog zu den Tests über das Antwortverhalten beim Anmounten wurden entsprechende Tests beim Zugriff auf ein bereits gemountetes NFS-Dateisystem mittels des Befehls **`ls <mountpoint>`** durchgeführt.

Das Verhalten beim Zugriff auf ein Dateisystem, dass auf dem Server nicht mehr freigegeben war (NFS-Server aber weiterhin erreichbar), zeigte sich bei allen Optionen einheitlich: **Es erfolgte keinerlei Reaktion!** (keine Fehlermeldung, kehrt nicht zurück)

3.11.1 Einsatz von NFS über TCP

1. nfsd ist nicht gestartet

hard: Es erscheint unabhängig vom Wert für *timeo* nach 10 Sekunden die Meldung: *NFS-Server not responding, still trying.* Der Zugriffsversuch wird fortgesetzt, bis er gelingt.

soft: Es erscheint unabhängig vom Wert für *timeo* nach 10 Sekunden die Meldung: *NFS getattr failed, Received disconnect.* Der Zugriffsversuch wird mit dem Fehlercode 27 abgebrochen.

2. Server nicht erreichbar

hard: Nach der durch *timeo* festgelegten Zeit (standardmäßig also 1 Minute) erscheint die Meldung: *NFS-Server not responding, still trying.* Der Zugriffsversuch wird fortgesetzt, bis er gelingt.

soft: Nach der durch *timeo* festgelegten Zeit (standardmäßig also 1 Minute) erscheint die Meldung: *NFS getattr failed, Received disconnect*. Der Zugriffsversuch wird mit dem Fehlercode 5 abgebrochen.

Bei Angaben für *timeo*, die kleiner als 10 sind, zeigten sich in der Mehrzahl der Fälle gleichbleibende Wartezeiten von 5 Sekunden. Es traten mitunter jedoch auch Fälle auf, die exakt die vorgegebene Zeitüberschreitung von unter einer Sekunde einhielten. Der Grund für diese scheinbare Willkür konnte von uns bisher nicht ermittelt werden, ein Zusammenhang mit der Wahl von *hard* oder *soft* besteht aber offensichtlich nicht.

Testbeispiele:

timeo=33 Meldung bzw. Abbruch nach 3,3 Sekunden

timeo=77 Meldung bzw. Abbruch nach 7,7 Sekunden

timeo=3 Meldung bzw. Abbruch nach 5 Sekunden

3.11.2 Einsatz von NFS über UDP

Die Tests zeigten ein identisches Verhalten für die beiden Fälle »nfsd nicht gestartet« und »Server nicht erreichbar«. Somit erübrigt sich an dieser Stelle eine getrennte Beschreibung der Testergebnisse.

Grundsätzlich gilt auch hier, das nach Ende der Wartezeit bei *hard* weiterversucht wird, die Anforderung abzusetzen, während bei *soft* der Vorgang schließlich abgebrochen wird.

Die Wartezeit für die Ausführung einer RPC-Anforderung ist bei Einsatz von UDP ergibt sich gemäß den Erläuterungen der Manpage zu mount_nfs folgendermßen: Die NFS-Anforderung wartet zunächst *timeo* Zehntelsekunden auf Antwort. Trifft keine Antwort ein, wird die Wartezeit mit 2 multipliziert, und die Anforderung wird erneut übertragen. Hat die Anzahl der Wiederholungen die in der Option *retrans* angegebene Zahl erreicht, wird eine Meldung ausgegeben. Bei *hard* wird nun die Anforderung weiterhin wiederholt, bei *soft* wird der Fehlercode ausgegeben und die Anforderung beendet.

Unsere Tests ergaben allerdings, dass dieses Berechnungsschema nur dann tatsächlich angewandt wird, wenn die so ermittelte Gesamtwartezeit 40 Sekunden nicht übersteigt. Ist zudem die erste Zeitüberschreitung kleiner als 0,75 Sekunden, so wird ein (fiktiver) timeo-Wert von 7,5 gewählt.

Ein Beispiel für retrans=2, timeo=8

Theoretisch ermittelte Gesamtwartezeit:

0,0s Start der Anforderung, Wartezeit: 0,8 Sekunden

0,8s 1. Wiederholungsversuch, Neue Wartezeit: 1,6 Sekunden

2,4s 2. Wiederholungsversuch, Neue Wartezeit: 3,2 Sekunden

5,6s Fehlermeldung, bei Option *soft* Abbruch

Ergebnisse des praktischen Tests:

(die Werte für retrans und timeouts wurden mittels *nfsstat -rc* ermittelt)

1. Anforderung mit Abbruch nach 5,6 Sekunden retrans: 2 timeouts: 3

2. Anforderung mit Abbruch nach 6,4 Sekunden retrans: 0 timeouts: 1

3. Anforderung mit Abbruch nach 12,8 Sekunden retrans: 0 timeouts: 1

4. Anforderung mit Abbruch nach 20,0 Sekunden retrans: 0 timeouts: 1

5. Anforderung mit Abbruch nach 20,0 Sekunden retrans: 0 timeouts: 1

Ein weiteres Beispiel für retrans=4, timeo=8

Theoretisch ermittelte Gesamtwartezeit:

0,0s	Start der Anforderung, Wartezeit: 0,8 Sekunden
0,8s	1. Wiederholungsversuch, Neue Wartezeit: 1,6 Sekunden
2,4s	2. Wiederholungsversuch, Neue Wartezeit: 3,2 Sekunden
5,6s	3. Wiederholungsversuch, Neue Wartezeit: 6,4 Sekunden
12,0s	4. Wiederholungsversuch, Neue Wartezeit: 12,8 Sekunden
24,8s	Fehlermeldung, bei Option *soft* Abbruch

Ergebnisse des praktischen Tests:

1. Anforderung mit Abbruch nach 24,8 Sekunden	retrans: 3	timeouts: 4
2. Anforderung mit Abbruch nach 20,0 Sekunden	retrans: 0	timeouts: 1
3. Anforderung mit Abbruch nach 20,0 Sekunden	retrans: 0	timeouts: 1

Ein weiteres Beispiel für retrans=1, timeo=1

Theoretisch ermittelte Gesamtwartezeit:

0,0s	Start der Anforderung, Wartezeit: 0,75 Sekunden (Mindestwartezeit!)
0,75s	1. Wiederholungsversuch, Neue Wartezeit: 1,5 Sekunden
2,25s	Fehlermeldung, bei Option *soft* Abbruch

Ergebnisse des praktischen Tests:

1. Anforderung mit Abbruch nach 2,25 Sekunden	retrans: 1	timeouts: 2
2. Anforderung mit Abbruch nach 3,0 Sekunden	retrans: 0	timeouts: 1
3. Anforderung mit Abbruch nach 6,0 Sekunden	retrans: 0	timeouts: 1
4. Anforderung mit Abbruch nach 12,0 Sekunden	retrans: 0	timeouts: 1
5. Anforderung mit Abbruch nach 20,0 Sekunden	retrans: 0	timeouts: 1

Die Testergebnisse lassen sich wie folgt interpretieren:

Offenbar wird die Anzahl von Versuchen, welche über *retrans* vorgegeben wurde, nur beim erstmaligen Auftreten eines Übertragungsfehlers berücksichtigt. Die erste fehlgeschlagene Anforderung führt die erwartete Anzahl an Übertragungswiederholungen aus. Nach Abbruch der ersten Anforderung wird offenbar bei folgenden fehlgeschlagenen Anforderungen nach der Zeitüberschreitung keine Übertragungs-wiederholung mehr veranlasst, sondern sofort abgebrochen. Die Wartezeit erhöht sich für die zweite Anforderung um den timeo-Wert (mind. 0,75s), alle folgenden ergeben sich jeweils aus dem verdoppelten Wert der vorangegangen Wartezeit, bis zur Obergrenze von 20 Sekunden.

Ein kleiner Bug ist uns beim Testen mit dem Standardwert für timeo=11 aufgefallen. Gibt man diesen explizit an, so entspricht das Ergebnis den Erwartungen, lässt man die Angabe weg, wird offenbar mit der Mindestwartezeit von 0,75 Sekunden gerechnet. Die Statistik für das eingehangene NFS-Dateisystem (nfsstat -m) weist jedoch in beiden Fällen gleichermaßen timeo=11 aus.

Wie bereits oben erwähnt, trifft für eine theoretische Gesamtwartezeit von mehr als 40 Sekunden für die erste Anforderung das dargestellte Berechnungsschema nicht mehr zu. Stattdessen wird als erste Wartezeit 20 Sekunden veranschlagt, darauf aufbauend wird die geforderte Anzahl an Übertragungswieder-holungen mit dem entsprechenden timeo-Wert vorgenommen:

Hier ein Beispiel (retrans=5, timeo=9) zur Verdeutlichung:

Theoretisch ermittelte Gesamtwartezeit ist mit 56,7s größer als 40s, demzufolge alternative Berechnung

0,0s	Start der Anforderung, Wartezeit: **20 (!)** Sekunden
20s	1. Wiederholungsversuch, Wartezeit: 0,9 Sekunden
20,9s	2. Wiederholungsversuch, Neue Wartezeit: 1,8 Sekunden
22,7s	3. Wiederholungsversuch, Neue Wartezeit: 3,6 Sekunden
26,3s	4. Wiederholungsversuch, Neue Wartezeit: 7,2 Sekunden
33,5s	5. Wiederholungsversuch, Neue Wartezeit: 14,4 Sekunden
47,9s	Fehlermeldung, bei Option *soft* Abbruch

Ergebnisse des praktischen Tests:

1. Anforderung mit Abbruch nach 47,9 Sekunden	retrans: 1	timeouts: 2
2. Anforderung mit Abbruch nach 20,0 Sekunden	retrans: 0	timeouts: 1
3. Anforderung mit Abbruch nach 20,0 Sekunden	retrans: 0	timeouts: 1

Legt man nun diese alternative Berechnung zu Grunde, ergibt sich eine Zunahme der Antwortzeiten von 3,1 Sekunden pro Erhöhung des timeo-Wertes um 1.

Dies gilt aber offensichtlich nicht mehr für Gesamtwartezeiten ab 1 Minute, hier nimmt die Antwortzeit um ca. 1,5 Sekunden pro Zeitüberschreitung zu, ab 1:20 Minuten nur noch um ca 0,75 Sekunden. Weitere Schwellwerte für die Abnahme der Antwortzeiterhöhung wurden von uns nicht konkret ermittelt. Es lässt sich nur festhalten, dass es keine lineare Abnahme gibt und sich die Antwortzeiten bei nur geringfügiger Änderung von größeren timeo-Werten kaum noch verändern.

Für alle vorgestellten Fälle kann aber festgehalten werden, dass für zweite und nachfolgende Anforderungen früher oder später eine Wartezeit von 20 Sekunden erreicht wird und diese dann auch kosequent beibehalten wird.

3.12 Optionen zur Pufferung von Attributen

Die ermittelten Attribute von Dateien und Verzeichnissen werden im Zwischenspeicher des Client eine bestimmte Zeitdauer gehalten, bevor vom Server neue Daten angefordert werden (Flush). Nach Ändern einer Datei werden die Dateiattribute erst nach einer Mindestzeit aktualisiert, in jedem Fall aber spätestens nach Ablauf der Höchstverbleibdauer. Analoges gilt für Verzeichnisse.

Es gilt: Falls die Datei geändert wurde, bevor die Mindestzeit für einen Flush abgelaufen ist, so wird diese erweitert um jene Zeitspanne, die seit der letzten Änderung vergangen ist. Dies erfolgt unter der Annahme, dass gerade geänderte Dateien sehr bald wieder modifiziert werden.

acregmin Mindestanzahl von Sekunden, wie lange die Attribute nach einer Dateiänderung zwischengespeichert werden (Standardwert 3 Sekunden)

acregmax entspr. Maximalanzahl von Sekunden (Standardwert 60 Sekunden)

acdirmax Mindestanzahl von Sekunden, wie lange die Attribute nach einer Verzeichnisaktualisierung zwischengespeichert werden (Standardwert 30 Sekunden)

acdirmax entspr. Maximalanzahl von Sekunden (Standardwert 60 Sekunden)

actimeo Mindest- und Maximalanzahl von Sekunden für Dateien und Verzeichnisse

Das Setzen von *actimeo=0* hat zur Folge, dass die angeforderten Attribute dem Client sofort zur Verfügung stehen, somit also höchste Aktualität gewährleistet ist. Auf der anderen Seite hat dies jedoch einen negativen Effekt auf die Performance durch erhöhte Wartezeiten sowie höhere Netzwerk- und Serverlast.

3.13 Optionen für Puffergröße der Lese- und Schreibvorgänge

rsize=<zahl> Puffergröße in Bytes für Lesevorgänge

wsize=<zahl> Puffergröße in Bytes für Schreibvorgänge

Standard- und zugleich Maximalwerte für *rsize* und *wsize*:

NFS-Version 2: 8192 Bytes

NFS-Version 3: 32768 Bytes

Im Allgemeinen lässt sich durch hohe Pufferwerte für Lese- und Schreibvorgänge eine verbesserte Performance erzielen, da die Anzahl von Plattenzugriffen sinkt. Zu beachten sind jedoch die erhöhte Netzwerklast und längere Warezeiten bei fehlgeschlagenen Anforderungen (insbesondere bei UDP, da hier die gesamte RPC-Anforderung komplett neu gesendet werden muss). TCP bringt hier Vorteile, da sämtliche Übertagungswiederholungen bereits von diesem übernommen werden anstelle des übergeordneten RPC-Levels.

3.14 Option noac

Wie bereits oben erläutert, werden Daten und Attribute in einem Cache des Clients zwischengespeichert. Dieses Verhalten lässt sich mittels der Option *noac* ausschalten. Der Zwischenspeicher für Schreibzugriffe wird zwar weiterhin gepflegt, allerdings werden in ihn kopierte Daten sofort auf den Server geschrieben.

Das Nicht-Zwischenspeichern von Attributen bringt z.B. Vorteile, wenn mehrere Clients im selben Verzeichnis permanent Schreibzugriffe haben, da Änderungen an Datei- und Verzeichnisattributen sofort aktualisiert sind. Da aber auch die Schreibanforderungen nun nicht mehr im Cache des Clients zwischengespeichert werden, ergibt sich auf der anderen Seite auch eine verschlechterte Write-Performance des Clients.

Der Unterschied von *noac* im Gegensatz zu *actimeo=0* besteht darin, dass *noac* das Zwischenspeichern sowohl von Daten als auch Attributen verhindert, während bei *actimeo=0* nur die Attribute nicht zwischen-gespeichert werden.

3.15 Option nocto

Normalerweise garantiert NFS eine sogenannte close-to-open-Konsistenz. Nach dem Schließen einer Datei werden alle modifizierten Daten an den Server übermittelt, und vom Client wird eine Bestätigung des Servers für den erfolgreichen Schreibvorgang verlangt (commit-Anforderung). Beim Öffnen der Datei werden die Daten neu vom Server in den lokalen Cache des Clients geladen, um die alte Dateiversion durch die aktuelle zu ersetzen. Dieses Verhalten stellt sicher, dass nach dem Schließen einer Datei deren Inhalt sofort für andere Clients sichtbar ist, welche diese Datei öffnen.

Beim Einsatz von *nocto* werden beim wiederholten Öffnen einer Datei keine neuen Daten vom Server angefordert in der Annahme, dass die Datei zwischenzeitlich nicht verändert wurde.

Dies verdeutlicht, dass der Einsatz dieser Option nur dann sinnvoll ist, wenn sichergestellt ist, dass Zugriffe auf ein bestimmtes Dateisystem nur *einem* bestimmten Client gestattet sind. Anderenfalls besteht die Möglichkeit des Auftretens verschiedener Kopien der gleichen Datei auf mehreren Clients.

Das Ziel für den Einsatz von *nocto* ist das Erreichen einer leichten Performancesteigerung (insbesondere bei Applikationen mit permanentem Öffnen, Schreiben und Schließen von Dateien), die durch verkürzte Wartezeiten auf neue Attributanforderungen erreicht wird.

3.16 Option sec

sec=<typ> Sicherheitsverfahren für NFS-Übertragungen
Mögliche Werte:

sys	Unix-Authentifizierung (Standard)
dh	Diffie-Hellmann-Authentifizierung
krb4	Kerberos-Authentifizierung

3.17 Überprüfen des Wertes von Optionen

Um alle derzeit gültigen Optionswerte eingehangener NFS-Dateisysteme zu überprüfen, eignet sich der Befehl **`nfsstat -m`**.

Folgende Ausgabe wäre möglich:

```
/mnt/test1 from server1:/share1
 Flags:
vers=3,proto=udp,sec=sys,hard,intr,link,symlink,acl,rsize=32768,wsize=32768,retr
ans=3,timeo=8
 Attr cache:    acregmin=3,acregmax=60,acdirmin=30,acdirmax=60

/mnt/test2 from server1:/share2
 Flags:
vers=3,proto=tcp,sec=sys,soft,intr,down,noac,link,symlink,acl,rsize=32768,wsize=
32768,retrans=5,timeo=30
 Attr cache:    acregmin=3,acregmax=60,acdirmin=30,acdirmax=60
```

Hinweis: Die Ausgabe zu /mnt/test2 enthält den Hinweis »down«, was anzeigen soll, dass der NFS-Server nicht erreichbar ist. Diese Angabe muss allerdings nicht aktuell sein, sie bezieht sich stets auf die zuletzt durchgeführte RPC-Anforderung an den Server. Konnte diese nicht erfolgreich abgeschlossen werden, wird der Serverstatus auf »down« gesetzt. Im anderen Fall erscheint kein Hinweis (d.h., ein Wert »up« o.ä. wird dann nicht explizit ausgegeben).

Die Angaben *link* und *symlink* geben an, dass der Server das Anlegen von Hardlinks und symbolischen Links erlaubt.

4. Fehlersuche

Fehlschlagenden Mountversuche und RFC-Anforderungen können verschiedenste Netzwerkprobleme zur Ursache haben und sind häufig schwer zu finden. Die folgende Checkliste soll dabei helfen, das Problem zu finden oder zumindest einzugrenzen.

1. Erreichbarkeit der Rechner mit *ping* überprüfen
2. Prüfung, ob die erforderlichen Dämonen auf dem Server laufen (mittels ps)
3. Test, ob der Dienst rpcbind überhaupt läuft: **`rpcinfo -T Protokoll Rechner rpcbind`**
4. Test, ob der Prozess *mountd* noch korrekt arbeitet: **`rpcinfo -T Protokoll Rechner mountd`**
5. Test, ob der Prozess *lockd* noch korrekt arbeitet: **`rpcinfo -T Protokoll Rechner nlockmgr`**
6. Prüfung, ob die erforderlichen Dämonen auf dem Client laufen (mittels ps)

7. Testen, ob die Freigaben des Servers korrekt sind (**`dfshares Servername`**)
8. Wird das richtige Mountverzeichnis auf dem Client verwendet?
9. Zugriffsrechte und Benutzerkennung richtig gesetzt?
10. NFS-Server zu stark belastet oder Netzwerklast zu hoch? (**`nfsstat, netstat`**)
11. Beim Einsatz von Routern und/oder Firewalls: Prüfen, ob und wie diese die NFS-Pakete weiterleiten

Sollten Clients über mehrere Netzwerkinterfaces verfügen, stellt sich die Frage, über welches Interface der Montierversuch abgesetzt wird und ob für dieses Interface eine Freigabe des Servers existiert. Prüfen können Sie dies wie folgt:

```
nfs-client:$ rsh -l <user> nfs-server
nfs-server:$ who
<user>   pts/21     Jan 11 10:28 (nfs-client)
```

Für den Client welcher unter `nfs-client` angezeigt wird, sollte die Freigabe auf Server Seite existieren.

5. Spezielle NFS-Konfigurationen

5.1 Failover bei Ausfall eines Servers

Um bei Ausfall eines NFS-Servers trotzdem weiterhin den Zugriff auf eingehängte Ressourcen zu ermöglichen, können statt nur einem auch mehrere NFS-Server beim mount-Befehl angegeben werden. Im Regelfall wird nun die Ressource des ersten erreichbaren Servers in dieser Aufzählung eingehangen. Schlagen nun zukünftige RPC-Anforderungen fehl, so wird versucht, die Anforderung auf dem nächsten Server der Liste abzusetzen (Failover).

Dieses Vorgehen bringt aber auch einige Einschränkungen mit sich:

1. Das Dateisystem darf nicht mit Schreibzugriff eingehängt sein. Die Sicherung der Gleichheit von Daten nach deren Änderung auf mehreren Servern (Konsistenz) kann durch das NFS-Protokoll nicht geleistet werden. Wird das Dateisystem dennoch mit Schreibzugriff eingehangen, so ist die Failover-Funktion deaktiviert.
2. Die einzelnen freigegebenen Dateisysteme auf den Servern sollten möglichst die gleiche Verzeichnis-struktur und identische Dateien enthalten (der Name der freigegebenen Verzeichnisse muss nicht identisch sein). Es leuchtet ein, dass die gesamte Failover-Konfiguration nicht sehr sinnvoll ist, wenn nach Ausfall eines Servers die gerade zugegriffene Datei auf den anderen Servern nicht vorhanden ist bzw. einen anderen Inhalt hat. Das Erzeugen gleichartiger Verzeichnisstrukuren mit identischen Dateien auf mehreren Rechnern kann z.B. mit dem Befehl *rdist* erfolgen.

Beispiel für eine Failover-Konfiguration mit identischen Freigabepfaden:

```
mount -F nfs -o ro server1,server2,server3:/share/docu /mnt
```

Beispiel für eine Failover-Konfiguration mit unterschiedlichen Freigabepfaden:

```
mount -F nfs -o ro server1:/docu,server2:/share/docu,server3:/opt/docu /mnt
```

5.2 Public Filehandle und WebNFS

Normalerweise werden Filehandles durch den Server erzeugt und verwendet, um eine bestimmte Datei oder ein bestimmtes Verzeichnis eindeutig zu identifizieren. Der Client erzeugt im Regelfall keine Filehandles und besitzt kein Wissen über dessen Inhalt.

Das Public Filehandle bildet eine Ausnahme. Es enthält vordefinierte Werte, die es ermöglichen, den *rpcbind* und *mountd* des Servers nicht mehr ansprechen zu müssen, um das entfernte Dateisystem einzuhängen. Durch des Entfallen des mount-Protokolls wird der Netzwerkverkehr reduziert und das Antwortverhalten verbessert. Soll der Mountvorgang über das Public Filehandle erfolgen, so ist anzugeben:

```
mount -F nfs -o public Serverfreigabe Mountpoint
```

Achtung! Derart gemountete Freigaben lassen sich mittels *dfmounts* nicht mehr anzeigen, da das mount-Protokoll nicht genutzt wird.

WebNFS bietet zusätzlich die Möglichkeit, über eine NFS-URL freigegebene Dateisysteme einzuhängen. Der Server muss WebNFS unterstützen, was bei Solaris aber der Fall ist. Syntax für den Mountvorgang:

```
mount -F nfs nfs://Server/Freigabe Mountpoint
```

WebNFS-fähige Browser ermöglichen einen Zugriff auf Ressourcen durch Eingabe in der Adresszeile von:

`nfs://Server/Freigabe` (z.Z. nur HotJava Browser)

Auch hier lassen sich durch die standardmäßige Verwendung des Public Filehandles gemountete Freigaben auf dem Server nicht anzeigen.

Auf eine nähere Abhandlung der verschiedenen Aspekte von WebNFS muss an dieser Stelle vorerst verzichtet werden, da dies den Rahmen dieses Dokumentes sprengen würde.

6. Anhang: Performancetests

Alle Tests erfolgten unter geringer Netzwerklast, kein Routing; es griff jeweils immer nur 1 Client auf den NFS-Server zu. Die Lese- und Schreibzugriffe wurden mittels des Befehls *dd* erzeugt, wobei jeweils 8192 Blöcke mit einer Größe von 8 Kilobyte gelesen bzw. geschrieben wurden.

Server: SunBlade 150, UltraSPARC-IIe, 650 Mhz, 1024 MB

Client: SunBlade 150, UltraSPARC-IIe, 650 Mhz, 256 MB

6.1 Vergleich UDP / TCP

Werte für rsize / wsize	Vorgang	Anzahl reads / writes	UDP	TCP
1024	write	65536	11,98s	11,83s
	read	65536	25,48s	33,29s
4096	write	16384	10,54s	10,58s
	read	16384	10,43s	13,13s
8192	write	8192	10,57s	9,61s
	read	8912	7,15s	8,4s
16384	write	4096	7,02s	6,48s
	read	4099	6,3s	6,4s
32768	write	2048	5,80s	5,95s
	read	0	0,72s	0,69s

6.2 Einsatz von noac und nocto (TCP)

Werte für rsize / wsize	Vorgang	Option	Anzahl reads / writes	Zeit
4096	write	-	16384	11,01s

Werte für rsize / wsize	Vorgang	Option	Anzahl reads / writes	Zeit
		noac	16384	48,40s
		nocto	16384	10,85s
	read	-	16384	13,95s
		noac	16384	24,62s
		nocto	16384	13,91s
16384	write	-	4096	7,72s
		noac	8192	27,73s
		nocto	4096	7,40s
	read	-	4099	6,33s
		noac	0	6,30s
		nocto	4099	6,23s
32768	write	-	2048	5,86s
		noac	8192	27,65s
		nocto	2048	5,83s
	read	-	0	0,64s
		noac	0	6,17s
		nocto	0	0,64s