LinuX Enterprise

4 | Dezember 2000
www.linuxenterprise.de
D52897

DM 9,50
SFr 9,50
Ös 80,–
LFr 231

LinuX Enterprise

DAS MAGAZIN
FÜR LINUX & OPEN SOURCE
IM UNTERNEHMEN

mit CD

Interviews:

- Martin Schulze, Kern-Entwickler der Debian-Distribution
- Chuck Jazdzewski, Inprise Chief Architect für Kylix

Schwerpunkt

embedded systems

Handies, PDAs und Waschmaschinen
Grenzenloser Linuxeinsatz

Kommt Zeit, kommt Linux
Linux etabliert sich als Echtzeitsystem

Linux IS Business
Erste LinuxWorld Conference & Expo in Deutschland

Team mit Zukunft
Geschäftskritische COBOL-Anwendungen unter Linux

Special Startups

„Realistisch und überzeugend"
Interview mit Eric Allman, Sendmail

Wie beim Skateboarden
Linux-Startups stellen sich vor

4|2000 LINUX & OPEN SOURCE IM UNTERNEHMEN www.linuxenterprise.de

4 395289 709505 04

NFS und NIS in großen Installationen

von Christian Hartmann

Einsparungen durch zentral verwaltete Peer-to-Peer-Netzwerke

Die Vernetzung von Arbeitsplatzrechnern erfolgt zumeist entweder in einer Client-Server- oder einer Peer-to-Peer-Struktur. Entgegen der weitverbreiteten Ansicht, dass sich letztere nur für kleine Arbeitsgruppen eignet und oberhalb einer gewissen Anzahl an Systemen bestimmte Aufgaben, wie Benutzerverwaltung und Speicherung von Nutzdaten einem zentralen Server vorbehalten sein sollten, lassen sich Unix-Systeme auch so miteinander vernetzen, dass sich die Vorteile beider Konzepte nutzen lassen, ohne die jeweiligen Nachteile in Kauf nehmen zu müssen.

Motivation

Bei der Vernetzung von Arbeitsplatzrechnern bieten sich im Wesentlichen zwei Strukturen zur logischen Gliederung an: Auf der einen Seite so genannte Peer-to-Peer (P2P)-Netzwerke, in denen alle Rechner gleichberechtigt sind und jeder seine lokalen Ressourcen anderen zur Verfügung stellt, oder andererseits eine Client-Server- Struktur, in der alle Nutzdaten auf einem oder einigen wenigen Servern liegen und auf Client-Rechnern nur noch Anwendungen installiert werden. Im Allgemeinen übernehmen diese Server auch noch andere Aufgaben, wie die Bereitstellung von Druckdiensten oder die Authorisierung von Benutzern.

Die vorherrschende Meinung geht dabei davon aus, dass P2P-Netze mit der Anzahl der Systeme schnell unübersichtlich, chaotisch und deswegen unverwaltbar werden. Aus eben diesem Grund wird bei größeren Installationen meist einer strikten Client-Server-Struktur der Vorzug gegeben, obgleich man damit auch prinzipbedingte Nachteile in Kauf nimmt. Im Wesentlichen ist das neben den zusätzlichen Kosten für Hard- und Software der Server die hohe Belastung des LANs, dessen Verfügbarkeit eine enorme Bedeutung für die Arbeitsproduktivität an Bildschirmarbeitsplätzen zukommt. Hingegen fallen in P2P- Netzen diese Extrakosten nicht an, und sie zeichnen sich dadurch aus, dass die Nutzdaten auf lokalen Festplatten gespeichert werden können, auf die auch noch bei Ausfall der Netzinfrastruktur zugegriffen werden kann. Der lokalen Speicherung kommt eine nicht zu unterschätzende Bedeutung zu, insbesondere dann, wenn schon eine einzelne Datei – wie bei komplexen CAD-Systemen durchaus üblich – schon mal die 100MB-Grenze überschreitet.

Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

Als Hauptargument für eine serverorientierte Vernetzung wird gebetsmühlenartig immer wieder die leichtere, weil zentrale Verwaltung z.B. von Benutzeraccounts oder Druckern ins Feld geführt. Dieser Artikel zeigt auf, wie auch ein sehr großes Netzwerk von Unix-basierten Arbeitsplatzrechnern so konfiguriert werden kann, dass die Vorteile beider Strukturen voll zum Tragen kommen, ohne die jeweiligen Nachteile in Kauf nehmen zu müssen und so gleichermaßen den Ansprüchen von Administratoren, Benutzern und denen an die Datensicherheit Rechnung getragen werden kann.

Die hier beschriebene Konfiguration ist in der Praxis in einem Netzwerk erprobt, das circa 500 Arbeitsplatzrechner und 1.000 Benutzer umfasst, auf denen zusammen etwa vier Terabyte Daten gespeichert sind – eine Datenmenge, bei der auch die leistungsstärksten Server schon mal an ihre Grenzen kommen. Dank der Peer-to-Peer Struktur arbeiten aber nahezu alle Benutzer mit lokal gespeicherten Daten, wodurch etwa zehn Mal schneller zugegriffen werden kann als über ein Netzwerk. Dieser Effekt wird dann noch verschärft, wenn Client und Server nicht im selben Subnetz liegen, die Daten also zusätzlich über Router laufen müssen.

User, NIS und der Automounter

Die Problematik verteilter Datenhaltung wird schnell offensichtlich, wenn man die Vorraussetzungen betrachtet, die erfüllt sein müssen, damit ein Benutzer an einem beliebigen Arbeitsplatz (Client) seine Daten, die in einem Verzeichnis des Servers gespeichert sind, im Zugriff hat. Neben einer Kennung (engl. Account) auf Client und Server müssen sowohl der Name des Servers im Netz als auch der Pfadname des Verzeichnisses bekannt sein. Im Gegensatz zu einem Windows-Netzwerk müssen zusätzlich bei der Verwendung von NFS die Accounts auf Client und Server identisch sein, da dieses keine Autorisierung unter einer anderen Kennung vorsieht, wobei wir hier die Möglichkeit einer automatischen Umsetzung auf dem Server außer Acht lassen. Für die Administration ergibt sich daraus die Notwendigkeit, auf allen Rechnern Accounts für alle Benutzer einzurichten, eine Aufgabe, die in großen (Windows-) Installationen mit mehreren hundert Rechnern schlicht nicht mehr zu bewältigen ist.

Für dieses Problem gibt es bei Unix-basierten Systemen mit NFS, NIS und dem Automounter eine außerordentlich elegante Lösung. Der „Trick" besteht nun darin, zwar die Daten lokal auf den Arbeitsplatzrechnern zu speichern, die Benutzerverwaltung hingegen zentral auf einem Server zu realisieren und den Clients derart zur Verfügung zu stellen, dass es für sie unerheblich ist, ob die Metadaten eines Accounts lokal gespeichert sind oder über das Netz bezogen werden. Aber der Reihe nach. Beginnen wir mit der Benutzerverwaltung.

Der Network Information Service (NIS) wird häufig als verteiltes Datenbanksystem beschrieben, was aber wenig zu unserer Erkenntnis beiträgt und leider auch nicht dem Nutzen gerecht wird. Dem nähert man sich schon eher, wenn man NIS als einen Transportmechanismus begreift, in dem ein Server nahezu beliebige Informationen über ein Netzwerk einer Vielzahl von Clients zur Verfügung stellt. Das geschieht in einer Art und Weise, dass die Inhalte in der Regel in der gleichen Form abrufbar sind, wie sie auch in lokalen Dateien gespeichert wären. Client-Systeme nutzen nun die Informationen teils ergänzend zu den eigenen, teils werden die lokalen Dateien aber auch komplett ignoriert. Seine vermutlich häufigste Verwendung besteht genau darin, die Informationen über Benutzer, die von Unix zur Autorisierung benötigt werden, bereitzustellen. Traditionell sind diese Metadaten in der lokalen Datei */etc/passwd* gespeichert. In einer Standard-NIS-Installation wird auch aus eben dieser Datei auf dem so genannten Master-Server eine Datenbank erzeugt, auf die – weil indiziert – schnell über das Netzwerk zugegriffen werden kann. Insbesondere in großen und zumeist auch heterogenen Netzen wird auf dem Master zweckmäßigerweise aber gerne eine separate Datei verwandt, wie zum Beispiel */etc/yp/passwd*.

Dadurch können z.B. nur solche Accounts über NIS verteilt werden, die von realen Personen zur Anmeldung an einem System benötigt werden, aber die – nennen wir sie technische oder non-login Accounts – plattformabhängig verwaltet werden. Danach stehen allen Clients alle Informationen aus der Quelldatei in einer so genannten Map zur Verfügung, die ebenfalls den Namen *passwd* besitzt. Der Zugriff eines Clients auf diese *passwd*-Map wird auf modernen Unix-Systemen durch das Hinzufügen des Schlüsselwortes *nis* in der Konfigurationsdatei des Name Service Switch (NSS) */etc/nsswitch.conf* aktiviert: *passwd: files nis*.

Lediglich bei der aussterbenden Gattung von Systemen, die noch nicht über NSS verfügen, muss auf die traditionelle Methode zurückgegriffen werden, bei der in der lokalen Datei */etc/passwd* ein + (das Pluszeichen) eingetragen wird. Ansonsten ist von dieser auch bei Systemen mit NSS möglichen Variante aber dringend abzuraten, wird doch dabei bei jeder Autorisierung die gesamte Map übertragen, ansonsten jedoch nur der jeweils abgefragte Datensatz. Fortan beschränkt sich die administrative Tätigkeit darauf, auf dem Master-Server den Inhalt der Datei zu pflegen, wobei wir vorgreifend erwähnen wollen, dass prinzipiell anstatt einer einfachen Datei auch jede andere beliebige Datenquelle denkbar ist. So könnten die Daten genauso gut aus einem „echten" Datenbanksystem oder auch aus einem modernen Verzeichnisdienst kommen. Neben einigen anderen Informationen wie dem gekrypteten Passwort oder der numerischen Identifikation eines Benutzers, enthält der Eintrag in der *passwd*- Map auch das so genannte Heimatverzeichnis (engl. Home Directory). An der Stelle kommt nun der Automounter und – bei neueren Implementierungen – sein ständiger Begleiter, das virtuelle Dateisystem *autofs* ins Spiel.

Denn die netzwerkweit zur Verfügung stehende Information ist naturgemäß immer die gleiche und somit auch der Pfad des Home Directories. Der Automounter wird in diesem Zusammenhang als ein „name-to-location-

resolver" aufgefasst, d.h. seine Aufgabe besteht darin, auf Basis eines nach festgelegten Regeln geformten Pfadnamens (hier dem Home Directory), ein auf einem entfernten Server freigegebenes Verzeichnis via Network File System (NFS) lokal zur Verfügung zu stellen. Man spricht dabei im Unix Jargon von „*Mounten*", zu deutsch „Einhängen".

Betrachten wir zunächst einmal eine vergleichsweise einfache Konfiguration des Automounters, um das Funktionsprinzip aufzuzeigen. Diese ist zwar auch in Abwesenheit von NIS operabel, zieht auch keinen weiteren Nutzen daraus, dürfte aber auch heute noch gelegentlich anzutreffen sein.

Das Home Directory eines beispielhaften Benutzers *tom*, dessen Daten auf einem Rechner *sonne* im dortigen (lokalen) Verzeichnis */home/tom* liegen, wird in der NIS- Map *passwd* auf */auto/home/sonne/tom* gesetzt. Der Automounter nun – sehr frei formuliert – legt bei seinem Start gewissermaßen die Hand auf das Verzeichnis */auto/home* (indem er dort ein Filesystem vom Typ *autofs* mountet) und wartet darauf, dass ein beliebiger Prozess darauf zugreift. Im Moment der Anmeldung des Benutzers *tom* fängt er den Zugriff auf eben dieses Verzeichnis ab, extrahiert daraus den Namen des Servers und mountet dessen Verzeichnis */home* lokal unter */auto/home/sonne*, in dem danach das Home Directory */auto/home/sonne/tom* zur Verfügung steht. Ein großer Vorteil ist dabei bereits schon jetzt ersichtlich: Dem einzelnen „Otto Normal"-Benutzer muss nicht mehr die lästige und fehleranfällige Pflicht auferlegt werden, sich den Speicherort seiner Daten – also den Namen des Servers und den dortige Pfad – zu merken, denn genau diese Information steckt schon im Pfad des Home-Directories.

Aber genau in dieser Notation lauert auch eine gern übersehene Gefahr, die eine derartige Konfiguration als wenig ratsam erscheinen lässt. Anwendungen haben im Allgemeinen die „Unart", verschiedenste Pfade in diversen Konfigurationsdateien abzuspeichern, und obwohl das Prinzip eines benutzereigenen Home-Directories so alt wie Unix selbst ist und noch dazu (genauso lange) dieser Pfad in der Umgebungsvariablen *$HOME* einem jeden Programm zur Verfügung steht, sind noch lange nicht alle Anwendungen willens oder im Stande, dieses auch zu nutzen. Ein prominentes Beispiel dafür ist der Internetbrowser Netscape, der sich die Freiheit nimmt, statt der Variablen *$HOME* deren Inhalt in seinen Konfigurationsdateien mehrfach abzuspeichern, in unserem Beispiel also */auto/home/sonne/tom*. Und als sei das noch nicht genug, kann der leidgeprüfte Anwender noch nicht einmal alle diese Pfade interaktiv über den Voreinstellungsdialog (Preferences) verändern. Wenn nun die Daten unseres Benutzers auf eine andere Maschine umziehen, weil ihm ein neueres, schnelleres System zur Verfügung steht oder er an einen Arbeitsplatz im benachbarten Büro zieht, nicht aber der Kollege, mit dem er sich bislang die Festplatte „geteilt" hat, nimmt das Drama unerbittlich seinen Lauf ...

Zusammenspiel von NIS und Automounter

Die Anwendung versucht nun weiterhin, auf Dateien oder Verzeichnisse zuzugreifen, deren Lage sie vermeintlich ihrer Konfiguration entnimmt. Da diese aber noch „alte" Pfade enthält, führt das beim Zugriff darauf dazu, dass erneut der Automounter aktiv wird und sich seinerseits bemüht, das in dem Pfad steckende entfernte Verzeichnis nach den gleichen Regeln wie oben zu mounten. Bedauerlicherweise ist dieser Versuch aber von vornherein zum Scheitern verurteilt, da sich die Daten ja mittlerweile auf einem anderen Server befinden, wovon der Automounter indessen nichts wissen kann. Ältere Implementierungen des Automounters können bei diesem Versuch das System bis zum Ablauf eines Timeouts sogar vermeintlich „einfrieren". Oft genug sind aber die alten Pfade auch noch in (womöglich binären) Cache-Dateien enthalten. Die Ursache der ungewollten Mountversuche ist dann nur sehr schwer zu lokalisieren.

Die Lösung des Dilemmas steckt in einem engen Zusammenspiel von NIS und Automounter, bei dem für das Home Directory ein sich nicht mehr ändernder Pfad verwandt wird, der weder den Namen des Serversystems noch den des dortigen Verzeichnisses enthält. Vielmehr wird beides unter dem Namen des Benutzers in einer weiteren NIS-Map gespeichert.

Dabei wird anstatt */auto/home/sonne/tom* der Pfad des Home Directories auf ein schlichtes */home/tom* gesetzt. Für jeden Benutzer existiert in der neuen Map ein Eintrag in der allgemeinen Form:

```
user   server:/path/to/users/
  home/directory
```

Die Vorteile eines mit NFS und NIS verwalteten P2P- Netzes auf einen Blick

- Vollständige zentrale Verwaltung
- Zentrale Kontrolle der Freigaben
- Optimale Ausnutzung der sowieso vorhandenen Ressourcen
- Keine zusätzlichen Kosten für Server
- Einheitliche, transparente und dauerhafte Sicht auf Daten
- Aussagekräftige, leicht zu merkende Speicherorte
- Erheblich reduzierte Netzwerklast
- Arbeitsplatzrechner unempfindlich gegenüber Netzwerkausfällen
- Clients unempfindlich gegenüber Server-Ausfällen

In unserem Beispiel also:

```
sonne:/home/tom
```

Die Map kann im Prinzip einen beliebigen Namen tragen, im Allgemeinen aber *auto.home*. Der Automounter wird nun so konfiguriert, dass er den Namen des Benutzers als Schlüssel beim Zugriff auf die *auto.home* benutzt, was dann entsprechend so aussieht:

```
/home  auto.home
```

Unglücklicherweise würde aber auf dem Server, der ja gleichzeitig auch Client ist, ebenfalls ein Automount-Prozess gestartet, der genau in dem Verzeichnis, in dem das Home Directory unseres Benutzers *tom* liegt, sein (virtuelles) Dateisystem mountet, und damit wären die ursprünglich dort liegenden Dateien und Verzeichnisse nicht mehr sicht- und also auch nicht zugreifbar. Aus eben diesem Grunde legt man das eigentliche (physische) Home auf dem Server in ein beliebiges anderes Verzeichnis, solange es nicht unterhalb von */home* liegt. Eine gute Wahl ist z.B. */export/home*. Wobei es auf einem System einer anderen Plattform im selben Netz auch genau so gut */usr/people* sein kann. Damit lautet also der Eintrag für unseren User *tom* in der *auto.home* Map:

```
tom    sonne:/export/home/tom
```

oder für einen anderen User „*tim*", dessen Daten auf „*sand*" liegen:

```
tim    sand:/usr/people/tim
```

Wie auch schon die *passwd*-Map könnte auch diese Map beispielsweise aus einer Datei */etc/yp/auto.home* generiert werden. Allerdings muss hierbei peinlich genau darauf geachtet werden, dass dabei nicht die Konsistenz verloren geht, also dass für jeden Benutzter in *passwd* genau ein Eintrag in *auto.home* existiert. Deswegen speichert man in einer anderen Variante diese Information sozusagen als „Anhängsel" mit in der *passwd*-Quelldatei, wobei beim Erzeugen der Maps die jeweils benötigten Felder herausgefiltert werden. Auf den ersten Blick mag eine solche Konfiguration von NFS, NIS und Automounter etwas verwirrend sein, aber unter dem Strich sind es nur einige wenige konfigurierende Eingriffe, um dieses Zusammenspiel der Komponenten zu ermöglichen.

auto.comm

Die Fähigkeiten des Automounters sind natürlich nicht auf das Mounten von benutzereigenen Home Directories beschränkt. Vielmehr lassen sich beliebige Verzeichnisse mounten, solange sie auf dem Client über einen gemeinsamen Namen zugegriffen werden. Ein gutes Beispiel sind von mehreren Benutzern gemeinsam genutzte Projektverzeichnisse. Auch hier besticht der Vorteil, dass sich der Zugriff darauf nie ändert, auch dann nicht, wenn die Daten auf einen neuen Server umziehen (müssen), z.B. weil der alte zu klein geworden ist oder ein anderer der Aufgabe dank leistungsfähigerer Hardware besser gewachsen ist.

Durch das Anlegen einer so genannten non-standard-Map in NIS *auto.comm* die einen Eintrag wie *projectx servx:/export/comm/projectx* enthält und einer entsprechenden Automounter-Konfiguration, */comm auto.comm*, kann der Zugriff unabhängig vom Namen des Servers immer über den Pfad */comm/projectx* erfolgen.

Verfügbarkeit

Auch bei der Verfügbarkeit kann sich unser P2P-Netz trotz gängiger Vorurteile gegenüber einem Client-Server-Netz gut behaupten. Ganz im Gegenteil – durch weitere Konfigurationen können die Systeme sogar deutlich robuster gegenüber Ausfällen werden, als es in einer CS Struktur möglich wäre. Dabei hat man es in der Regel mit vier Arten von Ausfällen zu tun:

- Ausfall der gesamten Netzinfrastruktur;
- Ausfall von NIS;
- Ausfall des Routings zwischen den Subnetzen;
- Ausfall eines NFS- Servers.

Bei einem Ausfall des gesamten Netzes ist in einer klassischen Client-Server-Struktur eine vernünftige Arbeit am „eigenen" Rechner nicht mehr möglich, weil niemand mehr an seine Daten heran kommt. Nicht so in unserem Peer-to-Peer- Netz, in dem die überwiegende Anzahl der Benutzer an ihrem eigenen System arbeit, also dem, das ihre Daten beheimatet. Somit ist der Zugriff auf diese Daten im Grundsatz immer möglich. Durch wenige zusätzliche Konfigurationsmaßnahmen kann auch beim Ausfall von NIS oder der Netzinfrastruktur der Zugriff auf die lokalen Daten gewährleistet werden. Aber auch hier ganz in Ruhe und der Reihe nach.

Am Anfang steht die Authorisierung des sich anmeldenden Benutzers. Wir brauchen also einen Eintrag in */etc/passwd* für „lokale" User, der natürlich auch die Angabe eines Home-Directories benötigt. Hier gibt es nun zwei Möglichkeiten: Entweder trägt man dort sozusagen direkt das eigentliche Verzeichnis (in unserem Beispiel also */export/home/tom*) ein oder den gleichen Pfad, den wir auch über NIS nutzen. Mit ersterem wäre ein Arbeiten sogar ohne den Automounter möglich, aber unter leicht erschwerten Bedingungen, denn es dürften – wie oben schon erwähnt – dürften eine Reihe von Anwendungen die Zusammenarbeit gänzlich oder wenigstens teilweise verweigern, da sie nach wie vor in ihren Konfigurationen den Pfad */home/tom* gespeichert haben. Deshalb tragen wir lieber genau den auch in der lokalen */etc/passwd* ein. Das zwingt uns aber nun, den Automounter, der in unserer bisherigen Konfiguration ja ebenfalls auf das Netzwerk zugreifen möchte, so einzustellen, dass er bei einem Ausfall auf lokale Dateien „zurückfällt".

Die dazu nötigen Maßnahmen unterscheiden sich je nach Implementierung des Automounters, also der Plattform, auf der er läuft. Dies erfolgt entweder direkt in seinen eigenen Konfigurationsdateien oder im Name Service Switch (NSS). Dabei kann grundsätzlich zuerst auf lokale Dateien und anschließend auf NIS zugegriffen werden oder auch umgekehrt. Manche Automounter unterstützen eine zumeist „kompatibel" genannte Syntax, bei der ähnlich zu */etc/passwd* eine NIS-Map mittels des Pluszeichens (z.B.: *+auto.home*) eingeschlossen wird, andere lassen sich über */etc/nsswitch.conf* konfigurieren, und

manche unterstützen gleich beide Varianten. Via NSS kann zuerst auf die lokale Datei */etc/auto.home* und, wenn dort kein passender Eintrag gefunden wurde, auch auf NIS zugegriffen werden: ((*automount: files nis*) oder alternativ genau in umgekehrter Reihenfolge und (hier in diesem Beispiel) auf die lokale Datei nur im Falle des Ausfalls von NIS (*automount: nis [NOTFOUND=return] files*).

Die Unterschiede sind allerdings nur marginal und zwar immer dann, wenn die lokalen Einträge immer auf einem aktuellen Stand sind, weil sie z.B. von Verwaltungssoftware automatisch gepflegt werden und so sicher gestellt ist, dass bei einem Umzug der Nutzdaten eines Anwenders keine falschen Einträge auf den Systemen verbleiben.

Beim Ausfall des Routings zwischen Subnetzen kommt ein weiteres Konzept von NIS zum Tragen, das darin von Anbeginn an existiert, obgleich es nicht einmal primär für diesen Fall vorgesehen wurde. So braucht man für Clients, die sich ihren NIS- Server ohne lokale Konfiguration per Broadcast suchen, so genannte Slave-Server, die aus Sicht des Clients nicht von einem Master-Server unterscheidbar sind. Und nachdem von diesen pro Subnetz wenigstens einer, besser zwei in wenigen Minuten initialisiert wurden, bedürfen sie keiner weiteren Pflege.

Bleibt noch das Problem des Ausfalls eines einzelnen NFS-Servers zu betrachten. In unserem P2P-Netz existieren naturgemäß maximal genauso viele Server wie Clients, und damit steigt auch die Wahrscheinlichkeit (innerhalb einer betrachteten Zeitspanne) des Ausfalls eines solchen. Hinzu kommt natürlich, dass „echte“ Server schon hardwareseitig gegenüber Fehlern wie Plattencrashs gut gerüstet sind, was wir von normalen Arbeitsplatzrechnern gemeinhin nicht behaupten können. Dies wird allerdings gleich wieder dadurch relativiert, dass vom Ausfall eines Servers im P2P-Netz nur wenige Anwender betroffen sind und das auch nur dann, wenn sie „remote” arbeiten wollen, hingegen alle, wenn ein „echter“ Server in einer CS-Struktur ausfällt. Grundsätzlich lässt sich das aber – so selten es in der Realität auch vorkommen mag – nie ganz vermeiden. Alles, was man vorsorglich tun kann, ist, die Wiederanlaufzeit eines Systems so klein wie möglich zu halten. Eine – obgleich aufwändige – Lösung besteht darin, die Systemsoftware strikt, d.h. physikalisch, von den Nutzdaten der Anwender zu trennen, z.B. durch den Einbau einer zweiten Festplatte. Im Falle des Crashs der Systemplatte kann diese durch ein vorbereitetes Exemplar einfach ersetzt werden. Fällt hingegen die Platte mit den Nutzdaten aus, beschränkt sich die Wiederherstellung auf das Einspielen der Daten aus einem Backup.

Autorisierung des Benutzers

NFS versus SMB

NFS und SMB sind beides auf TCP/IP aufsetzende Protokolle, die in erster Linie dazu genutzt werden, lokalen Plattenplatz anderen Rechnern über ein Netzwerk zur Verfügung zu stellen. Auch verfügen beide zusätzlich über die Fähigkeit, daneben Druckaufträge abzuwickeln, obgleich das in der Unix-Welt außerordentlich selten genutzt wird. Unterschiede existieren jedoch hinsichtlich der konzeptionell gesetzten Prioritäten. So liegt diese bei SMB auf dem Locking von Dateien und bei NFS auf der Zustandslosigkeit des Protokolls. Konsequenzen hat dieses vermeintliche Detail für die mit einem Server verbundenen Clients. So sind diese bei Verwendung von NFS nahezu unempfindlich gegenüber Serverstörungen wie z.B. einem Reboot desselben, wohingegen bei SMB die Verbindung in jedem Fall neu aufgebaut werden muss, schlimmstenfalls der Client ebenfalls neu gestartet werden muss. Da in einem P2P-Netz der durch einen Benutzer initiierte Neustart eines Servers nie ausgeschlossen werden kann, ist NFS von Haus aus für ein solches klar die bessere Wahl.

Sicherheit und Flexibilität

Wie sieht es bei der Verwendung von NFS und NIS mit Sicherheit und Flexibilität aus? Durch den Einsatz so genannter *netgroups*, die ein Bestandteil von NIS sind, für die es – im Gegensatz zu allen anderen Standard-Maps – kein lokales Äquivalent gibt, lassen sich beide Ansprüche realisieren. NIS-Netzgruppen sind formal betrachtet ein oder mehrere Tripel aus einem Hostnamen, einem NIS-Domainennamen und einer Benutzerkennung, wobei alle drei Elemente optional sind. Der Trick ist nun, die Namen von Netzgruppen alternativ oder zusätzlich zu einfachen Hostnamen in den Freigabe- bzw. Exportdefinitionen einzufügen.

Der entscheidende Vorteil liegt darin, dass der Name einer Netzgruppe in einer Exportdefinition erst zum Zeitpunkt des Zugriffs aufgelöst wird und dieser Inhalt dann sozusagen dynamisch aus dem Netz bezogen wird. Dadurch entfällt auch die Notwendigkeit, auf einem (Daten-)Server neu zu exportieren, eine Operation, die sonst explizit angestoßen werden müsste.

Damit können also zum einen die Freigaben von Verzeichnissen auf bestimmte einzelne Systeme und/oder Benutzerkennungen begrenzt werden (Sicherheit), und zum anderen kann gleichzeitig den unterschiedlichsten Ansprüchen der Benutzer hinsichtlich kooperativer Arbeit Rechnung getragen werden (Flexibilität). Obendrein kann nun auch die Konfiguration der Freigaben zentral verwaltet werden.

Das„Multiple Administration Authority”-Problem

Große Installationen mit mehreren hundert Systemen und vierstelligen Benutzerzahlen können nicht mehr von einer einzelnen Person verwaltet werden, und somit erfordert eine Standard NIS-Konfiguration wenigstens eine ständige Absprache zwischen den Verantwortlichen. Im ungünstigsten Fall sitzen die beteiligten Administratoren noch nicht einmal in einem gemeinsamen Raum, was eine Absprache beim Zugriff auf eine einzelne Datei praktisch unmöglich macht. Bei einer derartigen Größe sollte der Zugriff

auf diese Daten unbedingt programmgesteuert erfolgen, um den risikoreichen manuellen Zugriff zu vermeiden.

Dabei gibt es grundsätzlich zwei Lösungsansätze: Entweder verbietet man grundsätzlich den gleichzeitigen Zugriff auf die ganze Quelldatei, was das Problem aber nicht wirklich löst und obendrein bei mehreren Verwaltern schnell an seine Grenzen stößt, oder nur den auf einzelne Datensätze, was hingegen (vermeintlich) gleich den Einsatz eines ganzen Datenbanksystems nahe legt. Aber schon mit einfachen Maßnahmen – - sozusagen mit Unix-Bordmitteln – - lässt sich letzteres bewerkstelligen. So kann z.B. anstatt der Datei */etc/yp/passwd* auf dem Master-Server ein Verzeichnis genutzt werden, das dann seinerseits pro Benutzer eine Datei aufnimmt. Das verlangsamt zwar die Erzeugung der Maps, erfordert die Programmierung kleiner Skripte, die das ursprüngliche Dateiformat daraus generieren, und zuletzt auch Eingriffe in das so genannte Makefile, aber dafür ist der gleichzeitige Zugriff auf die Daten möglich, was den Aufwand schnell amortisiert.

Noch weniger Aufwand

Aber Unix wäre nicht Unix, wenn sich der administrative Aufwand nicht noch weiter reduzieren ließe. Oben haben wir schon den Einsatz zweier Festplatten pro System nahegelegt, sind aber nicht auf die Notwendigkeit der Konfiguration des einzelnen Systems eingegangen. Aber auch hier existiert schon seit vielen Jahren die Lösung in Gestalt des Bootprotkolls *BOOTP*, das mittlerweile von *DHCP* abgelöst worden ist, wobei letzteres nur eine runderneuerte Variante des ersteren ist. Durch dessen Einsatz entfällt die Notwendigkeit, die beim Start von einem Unix-System benötigten Informationen wie Hostnamen, IP-Adresse oder DNS- bzw. NIS-Domainnamen lokal zu konfigurieren und anstatt dessen ebenfalls über Netzwerk bereitzustellen. Das alleine ist in manchen Netzwerken auch heute schon Standard, aber auch eine quasi automatische Freigabe der User-Home -Directories ist mittels eines geeigneten Boot-Skriptes realisierbar, denn zum Zeitpunkt der Aktivierung der NFS-Serverdienste ist NIS schon lange verfügbar, aus dem die pro System zu exportierenden Verzeichnisse bezogen werden können. Das Ziel, die Systeme auch in einem P2P-Netz von jedweder manueller und statischer Konfiguration zu befreien, ist mithin erreichbar.

Datensicherung

Ein prinzipbedingter Vorteil eines zentralistischen Client-Server-Netzes ist zweifelsfrei die einfachere Datensicherung, da nur eines oder wenige Systeme gesichert werden müssen, wohingegen in unserem Peer-to-Peer- Netz eine um Faktoren größere Anzahl an Systemen erreicht werden muss. In der Praxis hat sich aber gezeigt, dass moderne, für den Unternehmenseinsatz entworfene Datensicherungssysteme wie z.B. ADSM, das von IBM vertrieben wird, problemlos auch mit tausend und mehr „Servern“ zurecht kommen.

LDAP

Über Kurz oder Lang wird NIS wohl durch LDAP ersetzt, das gegenüber diesem einige Vorteile aufweisen kann (siehe auch den Artikel auf Seite 101 dieses Heftes). Zum einen ist es – zumindest in der grauen Theorie – interoperabel mit LDAP-Servern verschiedenster Hersteller, erlaubt die Speicherung beliebiger Daten und – im Gegensatz zu NIS – auch die Abbildung hierarchischer Strukturen innerhalb eines Unternehmens. Denn so verlockend eine zentrale Verwaltung auch immer wirken mag, mit der Entfernung zum verwalteten Objekt wachsen auch die Schwierigkeiten. Und mit LDAP wird es sogar möglich, auch solche Informationen zu speichern, welche die „Erfinder” von Unix vor unzähligen Jahren nicht vorrausgesehen haben oder aber einfach nicht gebraucht wurden, wie z.B. die eMail-Anschrift eines Benutzers oder seine Telefonnummer. Auch wenn alle verbreiteten Unix-Systeme mittlerweile eine direkte Unterstützung für LDAP aufweisen und teilweise damit NIS transparent über den NSS ersetzen können, so dürfte es in der praktischen Anwendung immer noch die Ausnahme sein.

Und solange, wie man NIS nicht als statisches System mit festgelegter Funktionalität betrachtet, sondern als einen trickreichen Mechanismus, über den beliebige Informationen aus beliebigen Quellen über ein Netzwerk transportiert und sogar hierarchische Strukturen aufgebaut werden können, solange lässt es sich auch mit dem bewährten und vertrautem NIS auskommen.

Christian Hartmann ist teilhabender Geschäftsführer der hartmann+hertha GbR, deren Kompetenzschwerpunkte auf der Verwaltung großer Unix- Netzwerke sowie der Entwicklung von webbasierten Anwendungen liegen. Er ist zu erreichen unter: christian.hartmann@hartmann-hertha.de

auto.project

Die Fähigkeiten des Automounters sind natürlich nicht auf das Mounten von benutzereigenen Home -Directories beschränkt. Vielmehr lassen sich beliebige Verzeichnisse mounten, solange sie auf dem Client über einen gemeinsamen Namen zugegriffen werden. Ein gutes Beispiel sind von mehreren Benutzern gemeinsam genutzte Projektverzeichnisse. Auch hier besticht der Vorteil, dass sich der Zugriff darauf nie ändert, auch dann nicht, wenn die Daten auf einen neuen Server umziehen (müssen), z.B. weil der alte zu klein geworden ist oder ein anderer der Aufgabe dank leistungsfähigerer Hardware besser gewachsen ist.

Durch das Anlegen einer so genannten non-standard-Map in NIS, die zum Beispiel den Namen *auto.projects* trägt und einen Eintrag wie

```
px        sonne:/export/projects/px
```

enthält und der entsprechenden Automounter-Konfiguration */projects auto.projects* erfolgt der Zugriff auf den Client unabhängig vom Namen des Servers und dem dortigen Verzeichnis immer über denselben vom Automounter kontrollierten Pfad */projects/px*.