

Samba 3 und AFS: Storage wirklich skalierbar

7. Juni 2004

Rechtlicher Hinweis

Dieser Beitrag ist lizenziert unter der GNU Free Documentation License.

Zusammenfassung

Das Andrew File System AFS ist ein Netzwerkdateisystem, das ganz grob als Konkurrenz zu NFS gesehen werden kann. Im Vergleich mit NFS zeigen sich jedoch einige wesentliche Unterschiede, die in der unterschiedlichen Ausrichtung von AFS und NFS begründet sind.

NFS ist dafür gemacht, die Semantik des POSIX-Dateisystems so weit wie möglich im Netz nachzubilden. Ein Benutzer oder Anwendungsprogramm soll den Unterschied zwischen einer lokalen und einer Datei im Netz praktisch nicht merken.

AFS ist auf Skalierbarkeit ausgelegt. Diese Skalierbarkeit bezieht sich sowohl auf den zu Verfügung gestellten Plattenplatz als auch auf die Anzahl der Benutzer, die gleichzeitig zugreifen können.

Was hat Samba mit dem ganzen zu tun? Man kann direkt von Windows aus auf AFS-Server zugreifen, wenn man die entsprechenden Clients installiert. Die Clients nicht anfassen zu müssen, wenn man Unix-Server in eine Landschaft mit Windows-Clients integriert, ist aber die einzige Begründung, überhaupt Samba einzusetzen. Daher liegt es nahe, Samba als Proxy zu einer AFS-Infrastruktur einzusetzen. Die unterschiedliche Semantik von AFS und SMB macht jedoch Änderungen an Samba notwendig, wenn man eine nahtlose Integration anstrebt.

In Samba 3 gibt es einige Erweiterungen, die eine nahtlose Integration einer AFS-Umgebung in eine Windows-Umgebung möglich und einfach machen.

Dieser Vortrag wird AFS kurz vorstellen, die Probleme der Integration darstellen und die Lösungen, die in Samba 3 für diese Probleme gefunden wurden. Konkrete Themen sind die Kerberos-Authentifizierung von AFS, die Access Control Lists von AFS und ihre Abbildung in der Windows-Welt.

1 AFS im Überblick

Das Andrew File System AFS ist ein Netzwerkdateisystem, das ganz grob als Konkurrenz zu NFS gesehen werden kann. Im Vergleich mit NFS zeigen sich jedoch einige wesentliche Unterschiede, die in der unterschiedlichen Ausrichtung von AFS und NFS begründet sind.

- NFS ist dafür gemacht, die Semantik des POSIX-Dateisystems so weit wie möglich im Netz nachzubilden. Ein Benutzer oder Anwendungsprogramm soll den Unterschied zwischen einer lokalen und einer Datei im Netz praktisch nicht merken.
- AFS ist auf Skalierbarkeit ausgelegt. Diese Skalierbarkeit bezieht sich sowohl auf den zu Verfügung gestellten Plattenplatz als auch auf die Anzahl der Benutzer, die gleichzeitig zugreifen können.

NFS macht den Versuch, die lokale Semantik der Posix-Dateisystemaufrufe im Netz nachzubilden. Die volle Posix-Semantik nachzubilden ist extrem schwer, und wäre nur mit grossen Einbußen bei der Leistung möglich. Beispielsweise ist die Behandlung von geöffneten, aber bereits gelöschten Dateien nur über den Umweg von obskuren, temporären Dateinamen möglich.

Die AFS-Designer haben sich von der Idee, die lokale Semantik effizient abbilden zu können, relativ früh verabschiedet und sind dann den Weg konsequent weiter gegangen: Wenn wir schon keine lokale Semantik bekommen, wo können wir die Kompromisse in Richtung Skalierbarkeit verschieben?

- Lokationstransparenz: In einem AFS-System gibt es potentiell sehr viele Server, die Plattenplatz zur Verfügung stellen. Es wäre für einen Benutzer sehr aufwendig, den richtigen Server zu finden, auf dem seine Daten gerade liegen. Daher enthält AFS die sogenannte Volume Location Database, die den Aufenthaltsort aller Daten beinhaltet. AFS-Clients befragen transparent die vldb, um den aktuellen Aufenthaltsort der benötigten Daten herauszufinden. Dem Benutzer wird ein einheitlicher Namensraum unter /afs präsentiert.
- Einfache Datenreplikation: Man kann zur Lastverteilung für jedes Volume Replikas anlegen. AFS betreibt hierbei jedoch keine schwarze Magie in Form von transparenter Multi-Master Replikation. Es kann für jedes Volume nur eine einzige Schreib-Lese Replika geben, und diese werden auch nur auf explizite Anforderung repliziert.
- Verschieben von Volumes: Mit Hilfe der vldb ist es einfach möglich, Volumes von einem Server auf einem anderen zu verschieben. Dies kann im laufenden Betrieb geschehen, während Benutzer auf die Daten zugreifen. Zusammen mit detaillierten Statistikfunktionen kann so eine gleichmäßige Lastverteilung über alle Fileserver hinweg erreicht werden. Auch können Server zu Wartungszwecken aus dem Netz herausgenommen und neue einfach hinzugefügt werden.
- NFS versucht, unterschiedlichen Client-Workstations eine konsistente Sicht aller Schreibvorgänge im Netz vorzuspiegeln. Dies steht im Konflikt mit jeder Art von Caching. NFSv4 führt hier sogenannte Leases ein, die exklusiven Zugriff auf eine Datei gewähren, bis jemand anders die Datei ebenfalls öffnen möchte. SMB kennt diese Leases unter dem Namen Opportunistic Locks oder oplocks bereits seit mehr als einem Jahrzehnt.

AFS arbeitet so, dass Änderungen an einer Datei zunächst auf dem Client selbst durchgeführt werden. Weitere Clients, die die Datei ebenfalls öffnen, bekommen eine statische Version der Datei präsentiert, die noch keine Änderungen enthält. Erst wenn der schreibende Client die Datei schliesst, bekommt der nächste Client, der die Datei öffnet, die neue Version der Datei.

Dieses Verhalten kann sowohl als Vorteil als auch als Nachteil aufgefasst werden: Hält ein Prozess eine Datei lange offen, dann kann es passieren, dass viele Änderungen nicht persistent auf einer Platte landen, sondern beim Neustart des Clients verloren sind. Andererseits bekommen Datensicherungsprogramme immer eine konsistente Sicht der Dateien. Es besteht nicht das Problem binärer Datenbanken, dass geöffnete Dateien nicht konsistent gesichert werden können. Auf die eine oder andere Art muss die Applikation ohnehin dem Betriebssystem mitteilen, dass die Datenbank in einem konsistenten Zustand auf Platte liegt und gesichert werden kann. Bei AFS ist diese Mitteilung durch Schliessen und Wieder-Öffnen der Datei durchzuführen.

- Persistente Client-Caches: AFS-Clients sind so geschrieben, dass sie die angeforderten Daten cachem. Dies kann sogar auf der Platte des Clients geschehen, so dass der Cache einen Reboot überlebt. Dadurch ist die Übertragungsleistung von AFS im Netz nicht so entscheidend wie bei NFS und SMB, da viele Dateitransfers insbesondere von statischen Daten überhaupt nicht mehr anfallen. Die Konsistenz der Caches ist über interne Versionsnummern gewährleistet.
- Access Control Lists: AFS hat seine eigenen Vorstellungen davon, was eine Access Control List ist. Mit AFS kann man Berechtigungen nur für Verzeichnisse, und nicht für Dateien vergeben. In einem Verzeichnis können sehr wohl Berechtigungen für Dateien vergeben werden, diese gelten dann aber gleichzeitig für alle Dateien in diesem Verzeichnis. An dieses Konzept muss man sich gewaltig gewöhnen, aber wo ausser in /etc setzt man wirklich Berechtigungen für einzelne Dateien?

Um die Administratoren zu entlasten, gibt es zusätzlich noch die Möglichkeit, dass Benutzer selbst bis zu einer Grenze Gruppen anlegen können. So können Benutzer selbst sehr einfach Rechte auf ihren Verzeichnissen organisieren.

2 Interner Aufbau von AFS

AFS besteht intern aus einer Reihe von unterschiedlichen Dämonen:

- Der **kaserver** ist die zentrale Authentifizierungsinstanz von AFS. Er ist eine Implementation des Kerberos Protokolls Version 4. Kerberos bot sich beim Design von AFS an, da AFS für viele Server ausgelegt ist, und man von einem Benutzer nicht verlangen kann sich gegenüber jedem Server von neuem zu authentifizieren. Kerberos ist entworfen worden, exakt dieses Problem zu lösen: Man meldet sich einmal gegenüber dem Kerberos an und erhält Zugriff zu allen Ressourcen, zu denen man berechtigt ist.
- Der Protection Server **ptserver** sorgt für die Autorisierung der Benutzer. Der kaserver stellt sicher, dass der Benutzer vl wirklich derjenige ist, der er behauptet zu sein. Die wesentliche Aufgabe des ptservers ist es, die Liste der Gruppen zu pflegen, in denen vl Mitglied ist. Zusammen mit Access Control Lists im Dateisystem kann ein Fileserver feststellen, was ein Benutzer darf.
- Der **vldb**-Server stellt die Volume Location Database zur Verfügung. Aus dieser Datenbank können Clients ersehen, wo sich bestimmte Volumes gerade befinden, welchen Fileserver sie also befragen müssen.
- Der **fileserv** nimmt die eigentlichen Dateisystemanfragen entgegen, und steht insbesondere mit dem ptserver in Kontakt, um die Gruppenliste für einen Benutzer herauszubekommen.

Es gibt noch eine Reihe weiterer Server, wie beispielsweise den Backup Server, die aber für die Betrachtung hier nicht von Belang sind.

Ein Grund, warum AFS die Welt nicht im Sturm erobert hat, ist die Anzahl der Server, die aufgesetzt und unter Kontrolle gehalten werden müssen. Selbst wenn man wirklich Übung darin hat, benötigt man für eine AFS-Installation nicht unter einer halben Stunde, die erste Installation kann leicht einen Tag in Anspruch nehmen. Es gibt einen AFS Quick Start Guide mit einer Schritt-für-Schritt Anleitung. Man tut gut daran, sich sehr genau an diese Anleitung zu halten und die einzelnen Schritte erst später zu hinterfragen.

Auch ist die Administration von Anfang an etwas aufwendiger als die eines kleinen NFS-Servers. Benutzer müssen allein im AFS zweimal gepflegt werden. In der Datenbank des kaservers und in der des ptservers. Und davon hat man noch keinen Unix-Benutzer, mit dem man sich bei einer Workstation anmelden möchte. Dieser erhöhte Aufwand lohnt sich erst wenn man mehr als zwei oder drei Server zu administrieren hat. Dann ist es nicht mehr schwer, zusätzliche Server völlig gleichberechtigt in das System aufzunehmen und mit den relevanten Daten zu versorgen.

3 Samba und AFS

Was hat Samba mit dem ganzen zu tun? Man kann direkt von Windows aus auf AFS-Server zugreifen, wenn man die entsprechenden Clients installiert. Die Clients nicht anfassend zu müssen, wenn man Unix-Server in eine Landschaft mit Windows-Clients integriert, ist aber die einzige Begründung, überhaupt Samba einzusetzen. Daher liegt es nahe, Samba als Proxy zu einer AFS-Infrastruktur einzusetzen. Die unterschiedliche Semantik von AFS und SMB macht jedoch Änderungen an Samba notwendig, wenn man eine nahtlose Integration anstrebt.

3.1 Kaserver

Wenn Samba anstelle von Windows-Clients auf einen AFS-Server zugreifen soll, dann geschieht dies auf dem gleichen Weg, wie ein normaler Benutzer auf AFS zugreifen würde. Das heißt, der Samba-Server muss sich gegenüber AFS als Benutzer ausweisen, und zwar unter Vorlage eines Kerberos-Tickets. Der kaserver gibt aber mit einer gewissen Berechtigung Tickets nur gegen Vorlage von Passwörtern heraus.

Das Problem von Samba ist, dass es zur Anmeldung des Benutzers das Passwort zu keiner Zeit im Klartext besitzt. Als Domänencontroller oder Stand-alone Server kann Samba nur die Antwort des Clients im Challenge-Response Verfahren anhand der smbpasswd überprüfen. Als Domänenmitglied wird selbst diese Überprüfung an den Domänencontroller delegiert.

Als Active Directory Mitglied bekommt Samba vom Client ein Kerberos-Ticket geliefert. Man könnte auf die Idee kommen, dieses Ticket für AFS zu recyceln. Dieser Weg ist jedoch aus mehreren Gründen nicht praktikabel.

- Alle Clients vor Windows 2000 wären prinzipiell vom Zugriff ausgeschlossen, da sie nichts von Kerberos wissen.

- Selbst Windows XP und 2003 nutzen nicht unter allen Umständen Kerberos. Wenn Sie eine Verbindung zu einem Server nicht unter seinem Namen, sondern unter seiner IP-Adresse aufbauen, dann wird wie vorher die Anmeldung per NTLM ablaufen, ohne Kerberos.
- Windows verschickt Tickets Version 5, AFS arbeitet mit Version 4. Dieses Problem könnte man durch den krb524d umgehen, das ist aber bestenfalls ein Hack.
- Das größte Problem mit dieser Idee ist ein prinzipielles: Ein Kerberos-Ticket wird immer spezifisch für einen Dienst ausgestellt. Das Ticket, das eine Windows-Workstation einem Samba-Server präsentiert, ist für den Zugriff auf den Samba-Server und nicht zum Zugriff auf den AFS-Server ausgestellt.

Samba müsste unter Vorlage dieses Tickets zum AD-Domänencontroller gehen und ein weiteres Ticket zum Zugriff auf den AFS-Server beantragen, dieses dann dem krb524d übergeben und dieses dann verwenden. Damit kann der Samba-Server aber auf jeden Dienst im Netz anstelle des Benutzers zugreifen. Diese Forderung ist der Idee, in eine bestehende AD-Domäne integriert zu werden, nicht gerade förderlich.

Windows 2003 kennt das Prinzip der Constrained Delegation. Damit wäre dieses Problem "richtig" zu lösen, aber eben nur im Falle von modernen Clients, die ihre Server unter dem Servernamen ansprechen.

Wenn man eine Lösung mit dem krb524d installiert, muss man diesem den AFS-Serverkey übergeben, damit er Tickets anstelle des kaservers bauen kann, die der AFS-Server als gültig akzeptiert.

Die von Samba 3 implementierte Variante löst das ganze Problem ähnlich: Der AFS-Serverkey wird dem Samba-Server übergeben.

Was hat das für Konsequenzen?

- Dem Samba-Server wird vertraut. Samba kann gegenüber AFS jede gewünschte Identität annehmen. Das heißt, Samba wird in dem Sinne vertraut, dass der smbd nur dann Tickets ausstellt, wenn er sich von der Identität des verbindenden Benutzers überzeugt hat.
- Samba übergibt dem AFS-Server ein Ticket, das einen Benutzernamen enthält. Dieses Ticket wird vom AFS-Server ausgepackt. Der enthaltene Benutzername wird dem ptserver übergeben, der diesen Benutzernamen zu einer Gruppenliste macht, anhand derer die Zugriffsrechte im Dateisystem festgelegt werden.

Das heißt, Samba muss wissen, wie aus der Angabe der Windows-Domäne und dem Benutzernamen ein Benutzername gemacht wird, der dem ptserver bekannt ist. Dafür gibt es den Parameter `afs username map`.

- Soll AFS nur als Infrastruktur für eine verteilte Samba-Umgebung implementiert werden, so ist es nicht mehr notwendig, die Datenbank des kaservers zu pflegen. Samba kann die Tickets selbst erzeugen. Daher der Name dieser Option im `configure` von Samba: `-fake-kaserver`.

Eine noch nicht eingetragene Erweiterung des `winbind` macht es auch möglich, ein Ticket nach erfolgter Authentifizierung für normale Unixbenutzer zu erzeugen.

3.2 Ptproxy

Mit dem fake-kaserver ist die eine Benutzerdatenbank von AFS eliminiert. Wenn man AFS in eine existierende Windows-Infrastruktur als Storage-Lösung integrieren möchte, muss man die zweite Benutzerdatenbank des AFS noch pflegen. Der ptserver muss die NT-Domäne widerspiegeln. Insbesondere die Gruppen und die Mitgliedschaften darin müssen von Windows ins AFS oder umgekehrt nachgepflegt werden.

Dabei versieht der ptserver eigentlich Aufgaben, die der Domänencontroller der Domäne genau so gut vollziehen kann:

- Benutzer- und Gruppennamen in numerische IDs umsetzen und umgekehrt.
- Auf Anfrage die Liste der Gruppen herausgeben, in denen ein Benutzer Mitglied ist.

Dabei ist der ptservers auch nur ein Dämon, der dem Fileserver mit einem definierten Protokoll übers Netz ein paar Anfragen beantwortet. Mit dieser Motivation ist der ptproxy entstanden, der Anfragen des AFS-Fileservers entgegennimmt und an einen laufenden winbind weitergibt. Dieser Winbind übernimmt die Kommunikation mit den Domänencontrollern. Der ptproxy kann auf den AFS-Datenbankservern anstelle des ptservers eingesetzt werden. Damit ist auch die zweite Benutzerdatenbank des AFS obsolet geworden.

3.3 vfs_afsac1

Ein von den beiden vorhergehenden Projekten unabhängiges Modul in Samba ermöglicht die Pflege der AFS-ACLs von Windows aus. Dazu ist das VFS von Samba benutzt worden, mit dem man sich einfach in die getfacl und setfacl Systemaufrufe einklinken kann. Der Benutzer bekommt die AFS-ACLs in einer Weise präsentiert, dass das Windows-GUI diese versteht. Die ACLs von Windows werden umgekehrt so reduziert, dass AFS damit klarkommt.

3.4 strict AFS locking

Einer der grossen Vorteile von AFS sind seine persistenten Caches auf Clientsystemen. Man könnte nun in Versuchung kommen, dies für verteilte Fileserver über Standortgrenzen hinweg zu nutzen und Dateien für den Zugriff aus verschiedenen Standorten effizient zur Verfügung zu stellen. Dieses Ziel ist der heilige Gral der Clustersysteme. Das Problem hieran ist: es kann nicht funktionieren. Ein wesentlicher Punkt hieran ist das fehlende Locking über Servergrenzen hinweg.

Windows-Clients setzen voraus, dass Locking-Anfragen konsistent beantwortet werden. Wenn nun die gleiche Datei über verschiedene Samba-Instanzen mehrfach an Clients ausgegeben werden, dann muss das schiefgehen, da Dateisperren, die ein Client setzt, vom anderen nicht gesehen werden. Auch spricht die AFS-Semantik, dass Änderungen erst nach dem Schliessen der Datei von anderen Clients gesehen werden, dagegen.

Das SMB-Protokoll kennt im Gegensatz zu Unix jedoch nicht nur die fein granulierten Byte Range Locks, sondern noch die sogenannten Share Modes. Mit diesen Share Modes ist es möglich, dass ein Client eine Datei für sich exklusiv öffnet und kein anderer Client Zugriff darauf bekommt. Das Protokoll kennt sogar eine entsprechende Fehlermeldung: NT_STATUS_SHARING_VIOLATION.

Einer der Kompromisse von AFS ist, keine Byte Range Locks zu unterstützen. AFS kennt jedoch analog zu Windows Dateisperren, die exklusiven Zugriff auf eine ganze Datei für einen Client sicherstellen. Damit dies kontrolliert passiert, kennt AFS sogar ein Dateirecht, das das Sperren erlaubt oder verbietet. Diese Sperren funktionieren über AFS-Clients (also Samba-Server) hinweg.

Eine ebenfalls noch nicht eingetragene Erweiterung von Samba nutzt genau diese beiden Dinge, um gesichert den gleichen Filespace in verschiedenen Standorten zur Verfügung zu stellen. Die entsprechende Option ist mit dem Parameter „afs exclusive locks“ zu aktivieren.

Ein Client in Standort A öffnet eine Datei, Samba besorgt sich eine Sperre beim AFS. Ein weiterer Client in Standort B möchte die gleiche Datei öffnen, Samba bekommt die Sperre nicht und gibt eine Sharing Violation an den Client weiter. Dies ist eine Verletzung der Windows-Semantik, ist aber das beste, was mit den gegebenen Protokollen erreicht werden kann. Damit ist eine gemeinsame Dateiablage über Standortgrenzen hinweg möglich, die Benutzer bekommen vernünftige Fehlermeldungen bei einem gemeinsamen Zugriffsversuch. Eine gemeinsame Bearbeitung einer Access-Datenbank geht aber nicht. Dafür ist ohnehin Access denkbar ungeeignet.