Load Management

am Beispiel des Projektes *FARSITE*: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment

Jörg Diesinger

Ausarbeitung zum Vortrag im Rahmen des Seminars
„Peer-to-Peer Information Systems“
Max-Planck-Institut für Informatik
AG5: Databases and Information Systems Group
Prof. Dr.-Ing. G. Weikum

Universität des Saarlandes, Saarbrücken
Wintersemester 2003/04

Inhaltsverzeichnis

1 Motivation

Für gewöhnlich ist der Datenaustausch in großen Unternehmen oder Universitäten über das lokale Dateisystem eines jeden Desktop-Rechners oder über zentrale Daten-Server realisiert. Dabei weisen beide Varianten Schwächen auf, bieten aber auch jeweils Vorteile im direkten Vergleich miteinander: Der Server ermöglicht eine zuverlässige sowie transparente Datenhaltung. Hochwertige Hardware und der Einsatz von Cluster-Technologie können höchsten Lastansprüchen von seiten der Clients gerecht werden. Stellt man Dateien über einen Desktop-PC im Netzwerk zur Verfügung, ist zwar kaum zusätzlicher finanzieller Aufwand zur technischen Realisierung erforderlich, da direkte Kosten für teuere Server-Komponenten (RAID-/high-performance I/O-Systeme, CPU) und kompetente und zuverlässige Systemadministratoren entfallen, bezüglich der auftretenden Systemlasten sind hier jedoch schnell Grenzen offensichtlich, wenn man berücksichtigt, dass die Rechner zusätzlich einer kontinuierlichen Last lokaler Arbeitsprozesse ausgesetzt sind. Die Fehleranfälligkeit beider Systeme geht nicht zuletzt auf Kosten der Datenverfügbarkeit und Datenzuverlässigkeit: Während Server möglichst voneinander unabhängige Datenspeicher tragen und eventuell zusätzliche Backup-Lösungen bieten, besteht bei Desktop-Systemen eine größere Gefahr, dass durch Hardware-Fehler der gesamte Datenbestand vernichtet wird. Das normale Verhalten eines Benutzers, den Rechner bei Abwesenheit auszuschalten, macht die lokalen Daten für andere unzugänglich. Aber hier stellen auch Server einen „Single-Point-of-Failure“ dar, betrachtet man die Möglichkeit, dass beispielsweise durch Fehler im Netzwerk die Verfügbarkeit aller Daten verhindert wird oder diese in ihrer Gesamtheit im Fokus böswilliger Angriffe von außen liegen, d.h. ausspioniert oder zerstört werden können.
Ziel des von Microsoft[1] initiierten Projektes FARSITE ist es, die Vorteile eines zentralen Daten-Servers und die des lokalen Desktop-Dateisystems zu einem skalierbaren, verteilten und sicheren Dateisystem zu vereinen.

Die vorliegende Arbeit ist im Wesentlichen in drei Hauptabschnitte gegliedert: Der an diese Motivation anknüpfende Teil gibt einen Überblick über die Zielsetzungen des Projektes sowie dessen grundlegenden Konzepte in Design und Implementierung. Der Abschnitt „Merkmale von FARSITE“ erläutert Details der Realisierung wichtiger Aspekte wie Datenverfügbarkeit, Sicherheit, Konsistenz und Skalierbarkeit, aber auch die Fähigkeit, administrative Prozesse selbstständig einleiten zu können. FARSITE's Möglichkeiten der Lastenverteilung bzw. Reduzierung sind Inhalt des Abschnittes „FARSITE und Load Management“.

[1] Microsoft Systems and Networking Research Group

2 FARSITE im Überblick

FARSITE[2] ist ein serverloses, verteiltes Dateisystem, d.h. es benötigt keinen zentralen, verwaltenden Server und läuft ausschließlich und vollständig auf den beteiligten Desktop-Clients. Es funktioniert somit innerhalb eines Netzwerkes aus kooperierenden, aber nicht zwangsläufig vertrauensvollen Client-Rechnern.
Die Zielsetzungen, die im Einzelnen dem Design von FARSITE zugrundeliegen, werden nachstehend zusammengefaßt.

2.1 Zielsetzungen

Design und Implementierung von FARSITE umfassen hauptsächlich drei als Projektziele aufzufassende Aspekte:

- FARSITE bietet – basierend auf gewöhnlichen Desktop-Systemen – einen Dateiservice mit höchster Datenverfügbarkeit und Datenzuverlässigkeit, ungeachtet der dabei vorauszusetzenden geringen Erreichbarkeit einzelner Rechner, verglichen mit der eines Servers. Durch Replikation sowohl der Daten als auch ihrer (Verzeichnis-)Metadaten funktioniert FARSITE logisch als zentraler, verlässlicher Datei-Server in einer heterogenen Hard-und Software-Umgebung.
- Die Sicherheit der Daten und Metadaten im FARSITE-System ist ebenso gegeben wie die Möglichkeit, die Vertraulichkeit von privaten Daten eines jeden Benutzers zu gewähren und aufrechtzuerhalten. Verschlüsselung und Hash-Funktionen sichern die Datenintegrität der Dateien. Während Metadaten dagegen jederzeit vom System nachvollziehbar und überprüfbar sein müssen, unterliegen sie byzantinisch-fehlertoleranten Operationen [CaLi99].
- Nicht zuletzt ist es Ziel von FARSITE, ein System zu repräsentieren, welches eigenständig auf eine sich ändernde Umgebung reagiert, d.h. durch automatische Konfiguration Systemausfälle unter den Clients ausgleicht und bezüglich einer Lastenverteilung selbstständig optimierende Operationen anstößt.

Es ist hier zu erwähnen, dass FARSITE sich zweier technologischer Entwicklungstrends (nicht ausschließlich auf Desktop-Systeme bezogen) bedient, die Datenreplikation und -verschlüsselung als primäre Techniken bei der Realisierung rechtfertigen: Dem stetigen Ansteigen von Speicherkapazitäten generell steht eine langsamere Steigerung der tatsächlich vom Benutzer genutzten Kapazitäten gegenüber [DoBo99]. Außerdem können mit zunehmender Rechenleistung heutiger Desktop-Systeme Verschlüsselungsalgorithmen mit geringerem Rechen- und somit Lastenaufwand durchgeführt werden als die in einem Dateisystem unvermeidbar auftretenden I/O-Operationen.

[2] Federated, Available, and Reliable Storage for an Incompletely Trusted Environment

2.2 Voraussetzungen

Vor dem Hintergrund, FARSITE als Datenspeicherung bzw. Dateisystem auf Desktop-Workstations zu realisieren, können einige Annahmen bezüglich des Designs getroffen werden:

- Es wird eine maximale Skalierbarkeit von etwa 10^5 Clients angenommen, deren maximales Datenaufkommen etwa 10^{10} Dateien bzw. ca. 10^{16} Bytes beträgt.
- Es wird davon ausgegangen, dass die Client-Workstations über ein Netzwerk hoher Bandbreite und geringen Latenzzeiten verbunden sind.
- Die Verfügbarkeit der Rechner im Netzwerk wird als geringer eingestuft als die eines Servers, aber als höher verglichen mit der eines Internet-Hosts: Die Mehrzahl der beteiligten Clients ist in der überwiegenden Zeit verfügbar.
- Lediglich eine Minderheit aller authorisierten Benutzer hat die Absicht, Dateien oder Metadaten mutwillig zu zerstören.
- Viele Benutzer versuchen unabhängig voneinander, lesend auf gleiche Daten ohne entsprechende Berechtigung zuzugreifen.

Anhand der ersten vier Punkte ist erkennbar, dass der angestrebte Einsatzbereich von FARSITE nicht im offenen Internet liegt. Die Verwendung innerhalb einer Systemlandschaft von Unternehmen oder Universitäten rechtfertigt jedoch diese Aussagen.

2.3 Das Basis-System

Zwei wesentliche Konstrukte bilden das Grundgerüst der Implementierung des FARSITE Dateisystems:

- ***Namespace Roots***
  Namensräume stellen eine hierarchische Verzeichnisstruktur und damit das logische Repository des Dateisystems dar. Die Wurzel dieses Verzeichnisbaumes wird als *Namespace Root* bezeichnet. FARSITE erlaubt die Verwaltung mehrerer Namespace Roots und kann somit mehrere virtuelle Datei-Server repräsentieren. Workstations, die die Verwaltung der Namespace Roots übernehmen, werden in einer *Directory Group* zusammengefaßt (vgl. Abschnitt 2.4).
- ***Certificates***
  Zertifikate stellen in FARSITE eine Sicherheitskomponente dar: Ein Zertifikat ist eine Datenstruktur, die mit einem privaten Schlüssel signiert ist. Es werden drei Typen von Zertifikaten unterschieden:
  - *Namespace Certificates* authentifizieren Clients als Verwalter eines Verzeichnisbaumes.

– *User Certificates* binden einen Benutzer an seinen individuellen öffentlichen Schlüssel.

– *Machine Certificates* ordnen Clients einen öffentlichen Schlüssel zu und identifizieren sie somit als gültige Datenquelle.

Jeder Rechner in FARSITE ist angewiesen, alle Zertifikate als Authentifizierung zu akzeptieren, die über einen oder mehrere öffentliche Schlüssel validiert werden können.
Das Signieren von Zertifikaten, d.h. die Erzeugung eines Paares von privatem und zugehörigem öffentlichen Schlüssel, beschreibt den einzig notwendigen administrativen und manuell durchzuführenden Aufwand in FARSITE. Aus Sicherheitsgründen werden die privaten Schlüssel außerhalb des FARSITE-Systems von sogenannten *Certification Authorities (CA)* gehalten und verwaltet.

2.4 Die System-Architektur

Jeder Rechner in FARSITE kann bis zu drei Rollen übernehmen:

- ***Client***
  Der Client ist die Komponente in FARSITE, die direkt mit anderen Workstations in Interaktion tritt.

- Mitglied einer ***Directory Group***
  Die Directory Group ist eine Gruppe von Rechnern, die gemeinsam einen Teil eines Namespace Roots verwalten. Jedes Mitglied der Gruppe führt ein Replik eines hierarchisch organisierten, namensbasierten Kataloges von Verzeichnis-Metadaten und Dateien sowie deren Speicherort in FARSITE. Die Anfrage eines Clients an die Directory Group wird von allen Mitgliedern der Gruppe deterministisch bearbeitet und basierend auf einem byzantinisch-fehlertoleranten Protokoll beantwortet. Die Konsistenz der Verzeichnisinformationen ist garantiert, solange weniger als ein Drittel der Mitglieder der Directory Group fehlerhaft sind.[3]

- ***File Host***
  Während die Directory Groups die Verwaltung der Verzeichnisstrukturen übernehmen, halten File Hosts die replizierten Dateiobjekte. Ein lokaler Datencache eines File Hosts sorgt dafür, dass bereits angeforderte Dateien sofort und garantiert verfügbar sind.

Abbildung 1 zeigt schematisch die Rollenverteilung der Rechner im FARSITE-System. Die gepunkteten Linien verdeutlichen die separate Verwaltung von Daten- und Metadatenobjekten und den damit verbundenen Request/Response-Fluss.

[3] vgl. hierzu auch das Seminar-Thema *„Failure Resilience“* von Corinna Richter sowie [BrTo95] mit einem intuitiven Beweis dieser Aussage.

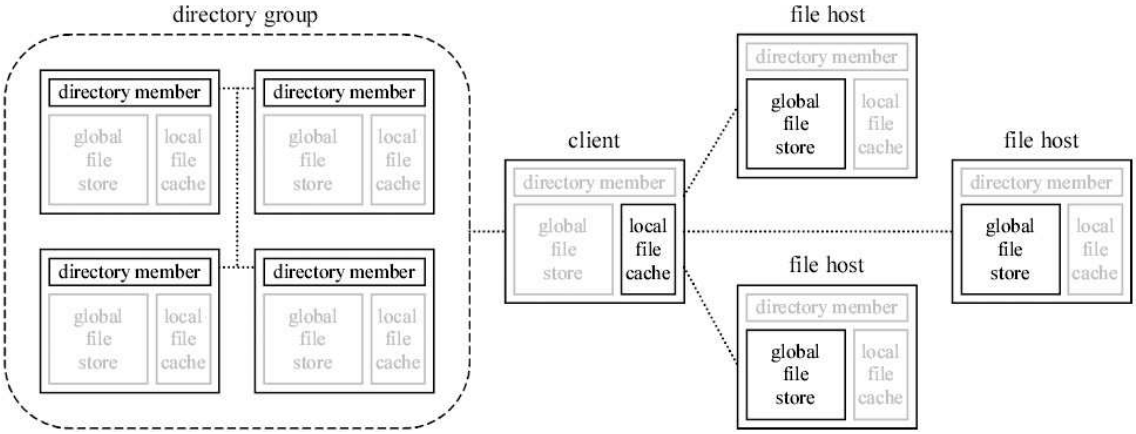

Abbildung 1: Teil der FARSITE System-Architektur aus der Sicht *eines* Clients [DoWa01]

Nachdem das Basis-System von FARSITE – die zugrundeliegende Architektur sowie die elementaren Konzepte von Namespaces und der Implementierung der Zertifizierungsmechanismen – vorgestellt wurde, befaßt sich der folgende Abschnitt mit der Realisierung einiger charakteristischer Merkmale von FARSITE, die nicht zuletzt einer Lastenverteilung im System dienen.

3 Merkmale von FARSITE

Wie bereits erwähnt implementiert FARSITE einen Caching-Mechanismus, der Daten in einem lokalen Speicherbereich der Festplatte eines Clients hält. Operationen auf diesen Dateien können somit direkt und ohne vorherigen Download von einem File Host durchgeführt werden. Lediglich eine Zugriffsberechtigung für die auszuführende Operation auf der Datei ist von der zuständigen Directory Group anzufordern. Diese werden in Form sogenannter *„Leases“* – zeitlich beschränkter Zugriffsberechtigungen – an die Clients vergeben. Abschnitt 3.3 erläutert das Prinzip.
Lokales Caching und die Tatsache, dass Updates auf Dateien zeitverzögert von Clients an die verantwortliche Directory Group geschickt werden, wirken sich einerseits positiv auf die Performance von Dateizugriffen aus, andererseits kann die Auslastung des Remote-Systems reduziert werden: Untersuchungen haben gezeigt, dass ein Großteil von durchgeführten Datei-Updates nach kurzer Zeit vom Client wieder überschrieben oder rückgängig gemacht werden [BHKSO91, Vog99]. Der durch redundante Dateioperationen entstehende Workload auf die Directory Group kann so vermieden werden.
Die getrennte Verwaltung von Dateien und deren Meta-Informationen in FARSITE hat zur Folge, dass Dateioperationen nicht dem byzantinischen Fehlerprotokoll der Directory Group unterliegen. Dies ist auch nicht erforderlich, da Dateiinhalte dem System verborgen sind und nicht wie Metadaten direkt vom System genutzt werden und damit vollständig offenbart sein müssen. Die Anzahl der Replikationen innerhalb einer Directory Group, die notwendig ist, um dem byzantinischen Protokoll zu genügen (höchstens ein Drittel aller Mitglieder dürfen fehlerhaft sein), kann somit bezüglich der Daten auf File Hosts erheblich reduziert werden: Ein verschlüsselter Hash-Tree in der Directory Group erlaubt die Validierung der Dateiinhalte (vgl. Abschnitt 3.2.3), und ein Fehlerzustand von allen bis auf einen File Host ist tolerierbar (vgl. 3.1).
Um einem steigenden Datenaufkommen und einem damit zunehmenden Workload gerecht zu werden, hat eine Directory Group die Möglichkeit, Teile ihres zu verwaltenden Namespaces an eine andere Directory Group zu übertragen. Entsprechend entsendete und signierte Namespace Certificates authentifizieren diese Directory Group als Verwalter der (Unter-)Verzeichnisstruktur.

3.1 Datenverfügbarkeit und Zuverlässigkeit

FARSITE's Konzept zur Verwirklichung langfristiger Datenpersistenz und sofortiger Datenverfügbarkeit liegt maßgeblich in der Datenreplikation.
Hinsichtlich einer persistenten Datenhaltung erlaubt Replikation den dauerhaften „Fehlerzustand“ eines Rechners, beispielsweise hervorgerufen durch Hardwarefehler oder Stilllegung durch den Benutzer. Kurzfristigen Ausfällen einzelner Rechner etwa aufgrund von Systemabstürzen und der damit nicht mehr gegebenen Verfügbarkeit der bereitgestellten Daten ist vorgebeugt.
Wie vorangehend erläutert bleiben die Metadaten einer Directory Group mit R_D Mitgliedern diesbezüglich erhalten, wenn nicht mehr als $\lfloor (R_D - 1)/3 \rfloor$ fehlerhaft sind (byzantinisch-fehlertolerantes Protokoll). Aus dargelegten Gründen werden von R_F registrierten File Hosts $(R_F - 1)$ fehlerhafte toleriert.

Stehen im FARSITE-System Rechner für eine festgelegte Zeitspanne nicht zur Verfügung, kann mit Hilfe der replizierten (Meta-)Daten der Directory Group und anderer File Hosts dieser Rechner ersetzt werden, d.h. seine Funktionalität in FARSITE wird von anderen Rechnern übernommen. Daten sind daher nur dann dauerhaft verloren, wenn zu viele Rechner innerhalb eines Zeitfensters ausfallen, so dass eine vollständige Migration der Funktionalität nicht durchgeführt werden kann.

Betrachtet man die Migration innerhalb einer Directory Group, so ist diese bereits nach relativ kurzer Ausfallzeit eines Mitgliedes notwendig, um gemäß dem byzantinischen Protokoll eine korrekte Arbeitsweise garantieren zu können. Daher werden bei der Migration von Metadaten die Rechner mit hoher Verfügbarkeit, d.h. geringer Ausfallzeit, bevorzugt.

Ein als Hintergrundprozess implementierter Mechanismus in FARSITE sorgt dafür, dass die Datenreplikationen, die selten dem System zugänglich sind, weil etwa die Rechner abgeschaltet sind, auf File Hosts mit höherer Verfügbarkeit „umgesiedelt“ werden. So werden alle Daten mit möglichst gleicher Verfügbarkeit bereitgestellt. In Abschnitt 3.5 wird diese Art der selbstständigen Konfiguration dargestellt.

Das Cachen von Daten auf Client-Rechnern trägt zusätzlich dazu bei, über benötigte Dateien verfügen zu können ohne auf die Bereitschaft der File Hosts angewiesen zu sein. Hierzu definiert FARSITE die sogenannte *„Cache Retention Period“*, die festlegt, wie lange Daten im lokalen Cache gehalten werden. Sie liegt bei etwa einer Woche [BDET00].

3.2 Datensicherheit

3.2.1 Zugriffskontrolle

Um innerhalb von FARSITE kontrollieren zu können, ob ein Benutzer, der einen Schreibzugriff auf eine Datei anfordert, dazu berechtigt ist, beinhalten die von der Directory Group verwalteten Verzeichnisinformationen eine sogenannte *„Access Control List (ACL)“*. Diese enthält die öffentlichen Schlüssel derjenigen Benutzer, denen ein Schreibzugriff auf das Verzeichnis und den darin enthaltenen Dateien gewährt wird.

3.2.2 Datenschutz

Der Schutz von Dateien und nutzer-spezifischen Verzeichnisdaten vor unberechtigten Lesezugriffen steuert in FARSITE das Verfahren der *„Convergent Encryption“*. Die Verschlüsselung beim Anlegen einer Datei folgt nachstehendem Algorithmus:

1. Berechne für jeden Datenblock der Datei einen eindeutigen Hash-Wert.
2. Verschlüssele jeden Dateiblock mit Hilfe des erzeugten Hash-Schlüssels.

3. Verschlüssele jeden Hash-Wert mit einem zufällig generierten symmetrischen Schlüssel *(file key)* und verschlüssele diesen wiederum mit dem öffentlichen Schlüssel aller authorisierten Benutzer (für n authorisierte Benutzer werden n verschlüsselte file keys erstellt).

Der private Schlüssel eines berechtigten Nutzers entschlüsselt damit den file key, der den Hash-Wert dekodiert und dadurch die Entschlüsselung der Dateiblöcke ermöglicht.

Der Grund für diese Art der Verschlüsselung liegt darin, dass mehrfache Kopien einer Datei auch dann vom System aufgespürt werden können, wenn sie mit verschiedenen öffentlichen Schlüsseln kodiert wurden.[4] Verschiedene Benutzer derselben Datei haben verschiedene Zugriffsschlüssel, aber die Datei kann stets auf demselben Weg entschlüsselt werden. Identischer Dateiinhalt konvergiert zu identischem Verschlüsselungstext, unabhängig von den verwendeten individuellen Benutzerschlüsseln.

Die Verschlüsselung einzelner Dateiblöcke ermöglicht es, einen bestimmten Block zu verändern ohne die gesamte Datei schreiben zu müssen bzw. einen Block zu lesen ohne den Download des gesamten Files abzuwarten.

3.2.3 Datenintegrität

Das byzantinische Fehlertoleranz-Protokoll sichert die Datenintegrität der Metadaten einer Directory Group, solange nicht mehr als ein Drittel der Gruppenmitglieder fehlerhaft sind. Die Datenintegrität der File Hosts bleibt dadurch gewährt, dass über jeden Datenblock der Datei ein Hash-Tree berechnet wird. Der vollständige Hash-Tree wird in der Datei selbst gespeichert, während der Hash-Wert der Wurzel in der verantwortlichen Directory Group hinterlegt wird.
Dateioperationen basierend auf diesem Hash-Tree können in logarithmischer bzw. linearer Zeit in der Größe der Datei ausgeführt werden (vgl. Abschnitt 4).

3.3 Konsistenz

Datei- und Verzeichniszugriffe werden in FARSITE von den Directory Groups kontrolliert. Die Zugriffsberechtigungen werden den anfragenden Clients zeitlich begrenzt und in Abhängigkeit der auszuführenden Dateioperationen zur Verfügung gestellt. Es werden vier Ausprägungen unterschieden:

- ***Content Leases***
  Content Leases sichern Dateidatenkonsistenz. Clients können durch Anforderung eines Content Leases bei der Directory Group auf den Dateiinhalt auf zweierlei Arten zugreifen – lesend/schreibend oder ausschließlich lesend.

[4] FARSITE unterscheidet zwischen Duplikaten (logisch verschiedene Dateien mit identischem Inhalt) und vom System erstellten Repliken. Duplikate sollen vom System erkannt und eleminiert werden, so dass zusätzlicher Speicherplatz zur kontrollierten Datenreplikation zur Verfügung steht [DABST02].

Wenn der Client einen Lease auf ein File hält und diesen nicht mehr benötigt, wird er an die Directory Group zurückgegeben. Erhält die Directory Group eine Anfrage auf einen Content Lease, kann es zu Konflikten mit bereits vergebenen Leases kommen – beispielsweise wird ein Schreib-/Lesezugriff auf eine Datei angefragt, der bereits von einem Client gehalten wird. In diesem Fall benachrichtigt die Directory Group die betroffenen Clients und fordert die Leases zurück. Stimmt der Client zu, erhält die Directory Group die Leases zurück, führt eventuell Updates aus und vergibt den Lease erneut. Für den Fall, dass der Client aufgrund eines Fehlers nicht antworten kann, implementiert FARSITE eine Ablaufzeit auf Content Leases. Updates werden danach verworfen. Die Anzahl der zu vergebenen Leases pro File ist hinsichtlich einer Lastenbegrenzung auf Directory Groups beschränkt.

- ***Name Leases***
  Name Leases sichern Namespace-Konsistenz. Sie erlauben einem Client, auf einen Namen einer Datei und/oder eines (Unter-)Verzeichnisses innerhalb des Namespaces zuzugreifen. Das Erstellen von Dateien oder Verzeichnissen erfordert ebenso einen entsprechenden Name Lease wie das Umbenennen. Analog zu Content Leases können auch die Name Leases von der Directory Group zurückgefordert werden bzw. unterliegen einer Ablaufzeit.

- ***Mode Leases***
  Mode Leases implementieren in FARSITE die von Windows bekannte file-sharing Semantik: Das Öffnen einer Datei in einem verteilten System erfordert die Festlegung auf die Art des Zugriffes – lesend, schreibend, löschend – sowie auf die Möglichkeiten der verteilten Rechte – gleichzeitiges Gewähren von Lese-/Schreib-/Löschrechten. Hält ein Client eine Datei ohne geteiltes Leserecht geöffnet, ist es für andere Clients nicht möglich, Lesezugriff auf diese Datei zu erhalten.
  FARSITE's Mode Leases umfassen demnach die folgenden sechs Ausprägungen: *read, write, delete, exclude-read, exclude-write, exclude-delete.*

- ***Access Leases***
  Access Leases realisieren Windows-Semantik bezüglich des Löschens von Dateien oder Verzeichnissen: Das Löschen eines Objektes kann erst dann erfolgen, wenn keine Zugriffe mehr registriert sind, also alle Leases auf dieses Objekt zurückgegeben wurden. Zuvor muss eine entsprechende Markierung durch den Client gesetzt werden, die die Directory Group zum Löschen veranlasst. Auch wenn danach weitere Leases auf das Objekt nicht mehr vergeben werden, sind alle bis dato aktiven Leases berechtigt, diese Markierung zu entfernen, den Löschvorgang also zu verhindern.
  FARSITE unterstützt diese Funktionsweise mittels dreier Ausprägungen von Access Leases: *public, protected, private.* Ein Public Access Lease zeigt an, dass der Client die Datei geöffnet hat, während ein Protected Lease zusätzlich einen weiteren Zugriff auf die Datei unterbindet. Der Private Access Lease informiert ferner darüber, dass keine weiteren Zugriffe existieren.
  Um ein Objekt als zu löschend zu markieren, benötigt der Client einen Protected oder Private Access Lease von der Directory Group. Ein Private Access Lease gestattet dem Client, den Löschvorgang als lokale Operation auszuführen.

3.4 Skalierbarkeit

FARSITE's Directory Groups sind in der Lage, Teile ihres Verzeichnisbaumes (Namespace) an eine andere Gruppe zu übergeben, indem ein entsprechendes Namespace Certificate signiert wird. Wie kann aber ein Client wissen, welche Directory Group für die Verwaltung der von ihm gewünschten Datei zuständig ist? Grundsätzlich hat ein Client die Möglichkeit, sukzessive alle Directory Groups zu kontaktieren, um letztlich die zu finden, die den geforderten Dateipfad verwaltet. Um allerdings eine skalierbare Lösung anzubieten, implementiert FARSITE ein Verfahren, das auf gecachten Pfadangaben und deren zugehöriger Verwaltungsgruppe beruht, der sogenannten *„Hint-based Pathname Translation"*.

3.4.1 Hint-based Pathname Translation

Jeder Client pflegt einen lokalen Cache mit Pfadnamen und deren verantwortliche Directory Group. Ein File-Request des Clients initiiert folgende Vorgehensweise:

1. Finde die längste Übereinstimmung des Pfades mit denen im lokalen Cache und kontaktiere die entsprechende Directory Group. Es tritt einer der Fälle ein:
   (a) Die angesprochene Directory Group verwaltet den Verzeichnispfad und kann den Request bearbeiten. Der Suchvorgang ist beendet.
   (b) Die angesprochene Directory Group verwaltet lediglich einen Präfix dieses Pfadnamens und antwortet mit all ihren bisher übertragenen Namespace Certificates. Diese nimmt der Client in seinen Cache auf und wiederholt den Vorgang.
   (c) Die angesprochene Directory Group verwaltet keinen Präfix-Pfadnamen und teilt dies dem Client mit, der den Eintrag aus seinem Cache entfernt und den Vorgang wiederholt.

Dieser Algorithmus fügt pro Schritt entweder eine weitere Informationen dem Cache hinzu oder entfernt eine ungültige. Er muss also terminieren, da im schlechtesten Fall für jeden Teilpfad die Information über die Verlagerung seiner Verwaltung auf eine andere Directory Group hinzugefügt wird, d.h. maxmimal zweimal die Anzahl der Teilpfade an Durchläufen benötigt wird.

3.5 Autonome Administration

Die Mehrzahl der bisher beschriebenen Prozesse im FARSITE-System werden direkt durch Client-Requests initiiert. Die Replikation von Daten auf File Hosts oder byzantinisch-fehlertolerante Operationen der Directory Groups sind beispielsweise Folgeprozesse, ausgelöst durch Daten-Updates von seiten der Clients. FARSITE's Administration erfordert dem entgegen jedoch auch, dass Prozesse nicht als Reaktion auf Client-Operationen veranlasst werden, sondern zeitlich wiederkehrende Abläufe sind. Hier ist die Verschiebung von Replikationen zu nennen, die eine möglichst

gleiche Verteilung der Verfügbarkeit aller Daten zum Ziel hat.
Wie sind nun solche Prozesse in Abhängigkeit eines Timers anzustoßen? Man muss bedenken, dass die Uhren der Clients nicht perfekt synchron gehalten werden können, die Prozesse aber dennoch innerhalb eines möglichst kleinen Zeitfensters ablaufen müssen, um beispielsweise die Verfügbarkeit zu erhalten (eine Änderung des Speicherortes zieht ein Update der entsprechenden Metadaten und deren replizierten Zustände nach sich).
FARSITE wendet hier *„Timed Byzantine Operations“* an.

3.5.1 Timed Byzantine Operations

Jedem replizierten Zustand in einer Directory Group mit R_D Mitgliedern sind R_D lokale Uhrzeiten zuzuordnen – eine bezüglich jeden Mitgliedes. Diese werden als *member times* bezeichnet. Die $k$te größte member time wird weiterhin als *group time* definiert, wobei $k = \lfloor (R_D - 1)/3 + 1 \rfloor$ ist.[5] Wenn eine beliebige lokale Uhrzeit eines Rechners die Ausführung einer Operation notwendig macht, veranlasst dieser das byzantinische Fehlerprotokoll, alle member times auf die lokale Zeit des Rechners anzugleichen. Dieses Update wird zwangsläufig auch die group time betreffen, was wiederum der Anstoß zur Durchführung aller geplanten Operationen zu kleinerer oder gleicher member time ist.

[5] Da $\lfloor (R_D - 1)/3 \rfloor$ Mitglieder fehlerhaft sein können und die group time die größte aller insgesamt R_D Zeiten ist, muss bezogen auf die korrekt handelnden Mitglieder der größte Wert gesetzt werden, also $\lfloor (R_D - 1)/3 + 1 \rfloor$.

4 FARSITE und Load Management

Die vorangegangenen Abschnitte haben gezeigt, dass sich aufgrund von Design und Implementierung von FARSITE die Analyse der Skalierbarkeit und des Load Managements im Wesentlichen auf die Directory Groups konzentriert.
Unterscheiden muss man hier zwischen der *direkten* Last auf die Verzeichnisgruppen durch Zugriff und Updates auf Metadaten und der *indirekten* Last, hervorgerufen durch Pfadauflösungen über alle Directory Groups hinweg.

Die direkt auftretenden Lasten werden weniger von der Anzahl der am System beteiligten Clients beeinflusst als vielmehr durch die Anzahl der Dateien pro Verzeichnis, was gemäß [BDET00] unabhängig voneinander ist. Somit kann eine obere Grenze für die direkte Last einer Directory Group gesetzt werden durch eine Begrenzung der Anzahl gleichzeitig aktiver Leases pro Datei (vgl. Abschnitt 3.3).

Der in Abschnitt 3.4.1 vorgestellte Mechanismus erlaubt eine erhebliche Reduzierung der indirekten Last auf die Directory Groups. Tritt ein Client dem System bei, ist er gezwungen, die Directory Group zu kontaktieren, die das Wurzel-Verzeichnis verwaltet. Da diese mit all ihren signierten Namespace Certificates antwortet, sollte der Client im weiteren Verlauf nicht wieder die Root-Gruppe kontaktieren müssen.

Ein weiteres Merkmal von FARSITE dient letztlich nicht ausschließlich einer verbesserten Performance: Die „faule“ Propagierung von Datei-Updates zur Directory Group kann einerseits den Netzwerk-Traffic reduzieren, da manch redundante Operation lokal bleibt. Andererseits verhindert sie eine zusätzliche direkte Last auf der Gruppe: Updates auf Metadaten werden ebenso vermieden wie deren Replikation auf alle Mitglieder der Gruppe. Auch auslastende Datenreplikationen auf File Hosts entfallen.

Die hinsichtlich der Integrität der Daten auf File Hosts verwendeten Hash-Trees über alle Datenblöcke einer Datei erlauben dem Client eine Validierung beliebiger Blöcke in logarithmischer Zeit in der Größe der Datei. Updates auf einzelnen Blöcken sind ebensfalls mit logarithmischem Aufwand möglich, auch ohne den Download der gesamten Datei abzuwarten. In linearer Zeit in der Dateigröße kann der Client die Validierung der ganzen Datei abschließen.

5 Zusammenfassung und Ausblick

FARSITE bietet ein skalierbares, dezentrales Netzwerk-Dateisystem, das auf einer Gruppe von unsicheren und unzuverlässigen Desktop-Rechnern aufbaut. Es repräsentiert virtuelle Datei-Server, die eine sichere und zuverlässige Datenspeicherung möglich machen. FARSITE erfordert minimalen administrativen Aufwand zum authentifizieren neuer Benutzer oder Rechner im System.

FARSITE's Architektur trennt die Verwaltung von Dateien und ihren Verzeichnisinformationen und realisiert so die Replikation der Metadaten innerhalb byzantinisch-fehlertoleranter Rechnergruppen sowie eine naive Replikation der Dateien auf File Hosts. Alternativ könnte FARSITE hier das Verfahren des *„Erasure Coding“* einsetzen, um den Speicherplatzbedarf der Replikationen zu optimieren. Dieser Ansatz wird beispielsweise im Projekt *OceanStore* verfolgt und beruht auf einer Fragmentierung und Kodierung der Daten.[6]

Die in dieser Arbeit beschriebenen Implementierungen haben gezeigt, dass FARSITE in der Lage ist, eine Verteilung der auftretenden Lasten im System zu erreichen – beispielsweise durch Aufteilung von zu verwaltendem Namespace unter den Directory Groups – bzw. diese möglichst gering zu halten, etwa indem effiziente kryptographische Algorithmen zum Einsatz kommen oder Dateioperationen lokal ausgeführt werden. Jedoch fehlen FARSITE Mechanismen, den Workload im System zu kontrollieren, der von einem einzelnen Benutzer verursacht werden kann: Die Tatsache, dass eine Beschränkung der Anzahl der zu vergebenden Zugriffsberechtigungen pro Datei oder Namespace existiert, hält einen Benutzer nicht davon ab, durch kontinuierliche Anfragen eine stetige indirekte Last auf den Directory Groups zu halten. Weiterhin bietet FARSITE keine Möglichkeit, einen einzelnen Benutzer daran zu hindern, den gesamten im System verfügbaren Speicherplatz unkontrolliert mit ständig neuen Dateifreigaben einzunehmen. Die Folge wäre nämlich nicht ausschließlich eine „Überflutung“ des Systems mit neuen Daten, sondern ein daraus resultierender unvermeidbarer Workload durch umfassende Datenreplikationen.

[6] Erläuterungen zu *Erasure Coding* und *OceanStore* finden sich in der Arbeit von Corinna Richter, *„Failure Resilience“*.

Literatur

[OSDI02] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, R. P. Wattenhofer: *FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment*. Proc. 5th OSDI, Boston, Dezember 2002.

[CaLi99] M. Castro, B. Liskov: *Practical Byzantine Fault Tolerance*. Proc. 3rd OSDI, USENIX, Februar 1999.

[DoBo99] J. R. Douceur, W. J. Bolosky: *A Large-Scale Study of File-System Contents*. Proc. ACM SIGMETRICS, Mai 1999.

[DoWa01] J. R. Douceur, R. P. Wattenhofer: *Optimizing File Availability in a Secure Serverless Distributed File System*. Proc. 20th IEEE SRDS, Oktober 2001.

[BHKSO91] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, J. K. Ousterhout: *Measurements of a Distributed File System*. Proc. 13th SOSP, Oktober 1991.

[Vog99] W. Vogels: *File system usage in Windows NT 4.0*. Proc. 17th SOSP, Dezember 1999.

[BDET00] W. J. Bolosky, J. R. Douceur, D. Ely, M. Theimer: *Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs*. Proc. ACM SIGMETRICS, Juni 2000.

[BrTo95] G. Bracha, S. Toueg: *Asynchron Consensus and Broadcast Protocols*. Journal of the ACM, Oktober 1995.

[DABST02] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, M. Theimer: *Reclaiming Space from Duplicate Files in a Serverless Distributed File System*. Proc. 22nd IEEE ICDCS, 2002.