MASTERARBEIT

Titel der Masterarbeit

Design eines sicheren, verteilten Dateisystems

Verfasser

Stefan Sommer, Bakk. rer. soc. oec.

angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

Wien, 2011

Studienkennzahl lt. Studienblatt: **A 066 926**
Studienrichtung lt. Studienblatt: **Magisterstudium Wirtschaftsinformatik**
Betreuer: **O.Univ.Prof. Dipl.-Ing. Dr.techn. A Min Tjoa**

Widmung

Meiner Frau Martina
und ihrer unendlichen Geduld gewidmet.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die mich durch mein Studium begleitet und unterstützt haben.

- Mein Dank gilt meinem Betreuer Dipl.-Ing. Mag. Dr. techn. Thomas Neubauer, für seine Unterstützung und vorbildliche Betreuung.
- Ich danke meinem Arbeitgeber PDTS AG, Frequentis Group, für die Möglichkeit der flexiblen Arbeitszeitgestaltung in den letzten Monaten.

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt zu haben. Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

Wien, Oktober 2011

Stefan Sommer

Inhaltsverzeichnis

Zusammenfassung

„Cloud“ ist derzeit ein geflügeltes Wort in der IT-Branche. Diese Technologie ermöglicht es Firmen, ihre Dienste und damit auch deren Betreuung auszulagern und schnell auf sich ändernden Bedarf an Rechenleistung zu reagieren. Auch Endnutzer profitieren von der Cloud, sie können ihre Daten bei vielen Anbietern abspeichern und von überall auf der Welt darauf zugreifen. Das wird zumindest von den jeweiligen Anbietern behauptet.

Leider entspricht die Realität nicht den Werbeversprechen der Anbieter, es kommt zu Ausfällen von Diensten, Daten gehen verloren und bei dem Wunsch nach Migration der eigenen Daten zu einem anderen Anbieter treten plötzlich vorher nicht bekannte Schwierigkeiten auf.

Diese Arbeit beschäftigt sich aus diesem Grund mit der Frage der Überwindung dieser Probleme. Es werden die notwendigen Grundlagen für die redundante Speicherung von Daten vorgestellt. Außerdem werden bestehende Dateisysteme und bestehende wissenschaftliche Arbeiten auf diesem Gebiet beschrieben. Danach wird ein Abriss über die Fähigkeiten einzelner ausgewählter Dienste-Anbieter gegeben und deren Güte bewertet.

Es folgt das Design eines Systemes, welches die Schwächen und die oben genannten Probleme nicht oder nur in vermindertem Maße aufweist. Es wird beim Design darauf geachtet, dass im Lichte der steigenden Kriminalitätsrate im Cyberspace auch das Bedürfnis des Benutzers nach Sicherheit vor unbefugtem Zugriff befriedigt wird.

Anhand von Beispielen werden die wichtigsten Punkte des Programmes erläutert und die korrekte Implementierung verdeutlicht. Zu guter Letzt wird eine Validierung der für das Design aufgestellten Requirements anhand von Versuchen durchgeführt.

Es zeigt sich, dass das hier implementierte System TDFS die Speicherung von Daten bei mehreren Anbietern unterstützt und sie gleichzeitig vor dem Zugriff Unbefugter abschottet.

Abstract

„Cloud“ is one of the most hyped buzzwords in the IT-business nowadays. Supposedly this technology enables enterprises to outsource services and maintenance, and to react to fast changing consumption of computational ressources. The „Cloud“ is even beneficial for end-users who are able to save their data at multiple storage providers and can access it from everywhere in the world. At least this is what providers want us to believe.

However, reality is not quite congruent to the providers' promises. Service breakdowns, lost data and other unexpected problems may occur, resulting in the necessity to migrate one's data to a more suitable service provider.

The focus of this thesis is the creation of a solution for these problems. Background information on information on redundant saving of data is provided to the reader. Furthermore existing file systems and scientific work on this matter is described. A short survey on the capabilities of a few chosen service providers is given and their service level is measured.

Next the design of a system aimed at reducing these limitations is explained. Due to the rising number of crimes in cyberspace the design also focuses on providing a means of securing one's data from unsolicited access.

The most important facts about the system are being presented by examples and their correct implementation is shown. Last but not least, the fulfillment of the requirements, which were created before designing the solution, is validated.

The implemented system TDFS („Transparent Distributed File System“) proves itself of being capable of saving data at multiple storage providers and securing its data from unwanted access.

Kapitel 1

Einleitung

Die wissenschaftliche Bearbeitung eines Themas beinhaltet die Analyse der Problemstellung, die Definition eines Problemlösungs-Ansatz und die Bewertung der Ergebnisse. Dieses Kapitel beschreibt die Gründe für die Untersuchung der Thematik rund um Cloud, Sicherheit und einem zu implementierenden Dateisystem.

1.1 Motivation und Ziele

Cloud-basierte Dienste sind derzeit ein gefragter Untersuchungsgegenstand wissenschaftlicher und kommerzieller Einrichtungen rund um den Erdball[1][2][3]. Von der Auslagerung verschiedener Dienste versprechen sich Unternehmen Kosteneinsparungen in relevanter Höhe. Diese Einsparung sollen durch geringeren Wartungsaufwand für selber betriebene Hardware, geringere Personalkosten und nicht zuletzt höhere Ausfallsicherheit in den ausgelagerten Rechenzentren erreicht werden[1].

Viele der in der Cloud angebotenen Dienstleistungen sind bereits seit längerem existierende Dienste, welche nun mit einer neuen Bezeichnung versehen erneut verstärkt durch die Marketing-Abteilungen des jeweiligen Anbieters beworben werden. Dazu zählen Infrastructure-as-a-Service (ehemals mietbare Serverhardware mit physikalischem Zugriff), Platform-as-a-Service (ehemals Root-Server) und Application-as-a-Service (klassische Dienstleistungen wie z.B. Microsoft Exchange)[4].

Ein weiterer dieser Services ist die sogenannte Cloud Storage, in der unter anderem auch über das REST-Protokoll Datenübertragungen stattfinden. Eine Cloud-Storage eines Anbieters stellt externen Speicherplatz für die Auslagerung von Daten zur Verfügung. Bei „Cloud-Storage“ handelt es sich, vereinfacht gesprochen, um die Auslagerung der Festplatte in das Internet. Für den Normalverbraucher ergibt sich dadurch die Möglichkeit des Zugriffes auf seine Daten auf jedem beliebigen Computer, der Zugriff auf das Netz des Providers hat. Prominente Anbieter wie beispielsweise Amazon mit dem S3-Dienst versprechen dabei hohe Verfügbarkeitsraten von 99,9999% bei gleichzeitig hoher Performance und geringen Kosten. Die Sicherheit der Daten vor dem Datenverlust wird von den Anbietern beworben[4].

Die Realität zeigt, dass die bei einem Cloud-Storage-Provider gespeicherten Daten nicht gegenüber möglichen Störszenarien gefeit sind. Probleme wie beispielsweise Ausfälle der Internetkonnektivität im Zugriff auf bestimmte Provider und auch Störfälle in der Speicherung der Daten beim Provider selbst werfen die Frage auf, ob der

Werbung der Anbieter im beworbenen Ausmaß vertraut werden kann [1].

Zusätzliche Gefahren für die Daten des Benutzers ergeben sich aus der teilweise mangelhaften Absicherung der Daten gegen unbefugten Zugriff. Aktuelle themenbezogene Artikel[2,3,4] beschreiben Szenarien, in denen Vollzugsbehörden der Staaten in denen Daten aufbewahrt werden, nach entsprechender Anordnung Zugriff auf diese erhalten. Dies begünstigt missbräuchlichen Zugriff und dessen Folgeerscheinungen wie zum Beispiel Wirtschaftsspionage[5].

Eine Lösung für die oben genannten Probleme ist die Auffächerung der eigenen Daten über mehrere Dienstanbieter. Dies vermindert die Wahrscheinlichkeit eines Datenausfalles drastisch. Die manuelle Ausführung einer solchen Tätigkeit ist jedoch zeitraubend, und leider ist es aufgrund restriktiver Zugriffsmechanismen mancher Anbieter oft auch gar nicht möglich, größere Mengen an Daten ohne weiteres von einem Provider zum nächsten zu transferieren [1].

Im Bereich der Dateisysteme gibt es als existierende Lösung solcher Probleme verteilte Dateisysteme, welche Daten über mehrere Datenträger und auch Maschinen verteilen, die mitunter große physikalische Entfernungen trennen[5]. Das Ziel derartiger Systeme ist jedoch zumeist der einfache Zusammenschluss mehrerer physikalischer Datenträger zu einem einzelnen logischen Datenträger. Auf die im Themenbereich „Cloud“ wichtigen Fragen der redundanten Aufbewahrung der Daten und deren Schutz vor unbefugtem Zugriff wird weniger Wert gelegt.

Gegenstand dieser Magisterarbeit ist die Untersuchung, ob sich das Prinzip des verteilten Speichers auch auf die Cloud-Storage-Provider anwenden lässt. Diese bieten teilweise recht unterschiedliche Zugriffsmechanismen an. Außerdem wird ein genereller Überblick über den Themenbereich des Cloud-Computing gegeben.

Zu diesem Zweck sollen unter anderem folgende Punkte untersucht werden:

- Datensicherheit (im Sinne von Redundanz)
- Gleichzeitiger Ressourcenzugriff
- Datenschutz (im Sinne von Zugriffsschutz)
- Data Sharing capabilities der Provider

Ein weiterer Fokus dieser Arbeit liegt auf der Frage welche Systeme zum Zugriff auf verteilte Dateisysteme derzeit verfügbar sind. Etliche kommerzielle Anbieter sind in diesem Feld tätig und auch Forschungsarbeiten beschäftigen sich mit dem Thema. Ihre Konzepte werden untersucht und ein Vergleich mit dem Stand der Technik vorgenommen.

Ein wichtiges Ziel der Arbeit ist die Sicherung der Daten vor dem Zugriff Unbefugter. Es wird bestimmt, mit welchen Mitteln diese Sicherung gewährleistet werden kann.

[1] http://aws.amazon.com/de/message/2329B7/

[2] http://www.nasuni.com/news/nasuni-blog/how-government-mandated-backdoors-could-threaten-cloud-security/

[3] http://www.nytimes.com/2010/09/27/us/27wiretap.html

[4] http://www.wired.com/threatlevel/2010/09/fbi-backdoors/

[5] http://futurezone.at/produkte/2774-dropbox-legt-daten-fuer-us-behoerden-offen.php

Redundante Speicherung von Daten geht immer auch mit einem gewissen Leistungsverlust aufgrund des erhöhten Verwaltungsaufwandes und des erhöhten Bandbreitenbedarfes einher. Es wird bestimmt, welcher Ansatz das beste Verhältnis von Geschwindigkeit zum betriebenen Aufwand erreicht.

Zu guter Letzt soll festgestellt werden, wie in der Cloud der sogenannte "Lock-In-Effekt [6], also die Einschränkung der Migrierbarkeit einer Softwarelösung oder der Migration von Daten aufgrund spezialisierter APIs zu beheben bzw. zu umgehen ist.

1.2 Methodik und Aufbau

Für die Bearbeitung der erwähnten Punkte wird folgendermaßen vorgegangen:

- Zuerst werden die notwendigen **Grundlagen** beschrieben, die für die redundante Speicherung von Daten notwendig sind:

  Darunter sind Technologien wie „RAID“ oder „Service Oriented Architecture“ zu verstehen. Bestehende Dateisysteme und Lösungen für die Speicherung von Daten in der Cloud oder weiter gefasst in verteilten Systeme werden überprüft. Als Vorreiter dieser Systeme dienen Dateisysteme wie das „Network File System“ oder auch das „Google File System“, moderne Lösungen wie HAIL[7] oder auch das als Web-Dienst aufgebaute eprint[6].

- Es wird festgestellt, welche **Anbieter von Speicherdienstleistungen** derzeit auf dem Markt verfügbar sind:

  Eine große Anzahl von Dienstleistern kämpft um diesen Wachstumsmarkt. Sie werden anhand zu definierender Metriken untersucht und dazu eine Bewertung abgegeben. Provider unterscheiden sich beispielsweise in Qualität des Angebotes, Quantität des angebotenen Speicherplatzes und Höhe des Preises.

- Aus den gewonnenen Erkenntnissen werden die **Requirements** für ein sicheres, verteiltes Dateisystem abgeleitet:

  Diese sind notwendig, um die Fähigkeiten des zu erstellenden Programmes zu definieren, und die Überprüfung des Erfolges zu vereinfachen. Das Erstellen einer Requirements-Liste gehört zu den „Best Practices“ der Software-Entwicklung.

- Im nächsten Schritt wird daraufhin das **Design** eines Systems durchgeführt, das die vorweg erwähnten Probleme behebt oder zumindest deren Auswirkungen stark einschränkt:

  Im Detail wird auf die einzelnen Komponenten des Programmes eingegangen und deren Funktionalität beschrieben. Um die Übersicht über das Programm zu gewährleisten werden außerdem Übersichtsdiagramme dargestellt. Einzelne wichtige Klassengruppen werden hervorgehoben und ihre Bedeutung für die Gesamtheit des Programmes beschrieben.

- Darauf aufbauend wird auf Basis dieses Designs eine **Implementation** vorgenommen, welche die Verbesserungen exemplarisch aufzeigt:

[6] http://www.eprints.org/

Im Implementationskapitel wird exemplarisch auf die Lösung einiger Probleme bei der Erstellung des Programmes eingegangen und auch Beispiele für die Bedienung desselben vorgeführt. Es dient vor allem dem leichteren Verständnis des Progamms im Falle einer allfälligen Erweiterung der Software durch andere Personen.

- Zu guter Letzt werden die Eigenschaften bereits vorhandener Systeme zur Neuimplementierung in Relation gesetzt:

  Diese Validierung dient zur Überprüfung der gestellten Anforderungen. Es wird mittels Messungen gezeigt, in welcher Güte die vorliegende Lösung die Anforderungen implementiert.

Kapitel 2

Grundlagen

Der Begriff „Cloud“ ist ein Modewort und kennzeichnet die Vermengung beziehungsweise Aufwertung existierender Basistechnologien, auf die später vertiefend eingegangen wird. Für jemanden, der mit der Materie nicht vertraut ist, ist er ein schwer zu qualifizierender Terminus und wird als solcher zurecht oft als nebulös bezeichnet[7]. Dieses Kapitel erklärt Historie und Definition dieses Begriffes gemäß dem aktuellen Forschungsstand und beschäftigt sich des Weiteren mit den verfügbaren Arten von Cloud-Services. Abschließend wird auf die inhärenten Risken bei der Verwendung von Cloud-Diensten eingegangen und Metriken zur Messung der Dienstgüte vorgestellt.

2.1 Definition der Cloud

Als Cloud (zu deutsch Wolke) wird in der Informationstechnik die Abstraktion von Ressourcen bezeichnet, auf die über bestimmte standardisierte APIs zugegriffen werden kann.

Das National Institute of Standards and Technology (NIST) beschreibt in seinem Paper „The NIST definition of cloud computing“ [8] die Charakteristika des Cloud-Computing. Demnach ist Cloud-Computing

> *(...) ein Modell um einfachen und zeitunabhängigen Netzwerkzugriff auf einen gemeinsamen Ressourcen-Pool (z.B. Netzwerke, Server, Speicher, Applikationen und Dienste) zu ermöglichen, der in kurzer Zeit bereitgestellt und ohne großen Verwaltungsaufwand oder Eingriff des Dienstleisters ablaufen kann (...)*

Cloud-Computing hat einen langen historischen Entstehungsprozess hinter sich (vgl. [4]). Am Anfang standen Mainframe-basierte Architekturen. In diesem Umfeld wurde dem wissenschaftlichen oder kommerziellen Benutzer ein bestimmter Anteil der Rechenzeit zugesprochen, welcher dieser zu bestimmten Zeiten verbrauchen durfte. Der Verwaltungsaufwand war dementsprechend hoch, entsprach aber den Kosten und dem schwierigen Zugang zu Rechnerressourcen zu dieser Zeit.

Den nächsten Schritt stellten dann um 1980 die sogenannten Personal-Computer (PCs) dar. Da nun auch Privatpersonen Zugang zu Rechnern hatten, vergrößerte

[7] http://www.gartner.com/it/page.jsp?id=766215

sich die Anzahl der Benutzer exponentiell und ließ den Bedarf an Kommunikationslösungen für Computer wachsen.

Auf die Phase der PCs folgte die Phase der (lokalen) Netzwerke, wobei es erstmals für Endbenutzer möglich war auf Ressourcen außerhalb des eigenen Computers zuzugreifen. Der Zugriff wurde wie heute (wieder) üblich über Netzwerkschnittstellen durchgeführt.

Den nächsten Schritt markieren die (semi-)globalen Netze, wobei am Beginn lokale Netze zu größeren Einheiten verbunden wurden. Aus der Verbindung dieser autonomen Systeme miteinander entstand letztendlich das heutige Internet.

Deutlich höheren Fokus auf die auf diese Weise miteinander verbundenen Computer legt das Grid Computing, welches bereits eine starke Entwicklung zur Abstraktion von Rechner-Ressourcen nach außen beinhaltet. Bekannte Vertreter dieses Paradigmas sind wissenschaftliche Auswertungen, die die Rechenkapazität eines einzelnen Rechners übersteigen würden. Im Rechnerverbund sind diese bewältigbar. Ein Beispiel für einen solchen Rechnerverbund stellt das LHC [9] dar. Die riesigen Datenmengen, die beim Betrieb des Teilchenbeschleunigers gewonnen werden (etwas mehr als ein Petabyte im Monat), werden von Rechnerverbünden, die aus günstiger Standardhardware zusammengestellt sind, verarbeitet.

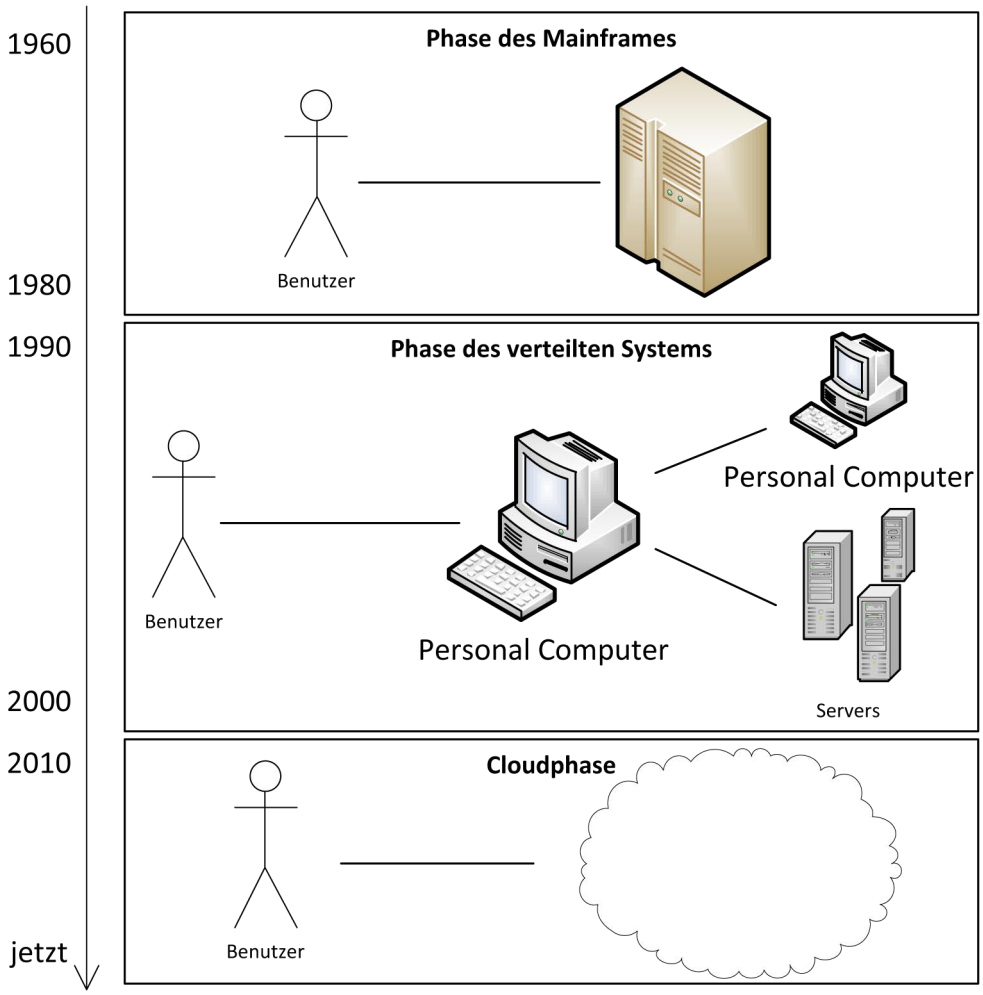

Abbildung 2.1: Phasen der Netzentwicklung

Weiters beschreibt das NIST in [8] fundamentale Eigenschaften von Cloud-basierten Systemen. Dazu zählen

- **Benutzer-induzierte Provisionierung** von Ressourcen ohne Anbieter-Eingriff
- **Standard-gestützte Übertragungsverfahren**, die den Zugriff durch unterschiedlichste Clients erlauben.
- **Ressourcen-Pooling:** Dienstleisterressourcen werden dem Benutzer über mehrschichtige Modelle zur Verfügung gestellt, wobei die Zuteilung dynamisch passiert und für den Benutzer transparent ist. Der Benutzer ist über Ort und Entfernung der Aufbewahrung seiner Daten in diesem Zusammenhang generell unwissend, allerdings gibt es bei manchen Dienstleistern Möglichkeiten in beschränktem Umfang Ortsvorgaben zu machen (z.B. um Antwortzeiten zu minimieren).
- **Elastizität:** Dem Benutzer ist es möglich, Art und Umfang seiner Dienstleistung in der Cloud transparent und unmittelbar zu verändern (z.B.: seiner Cloud-gestützten Festplatte mehr Speicherplatz zuzuweisen). Für den Benutzer wird in vielen Fällen der Anschein von Unbegrenztheit dieser Ressourcen erweckt.
- **Messbarkeit:** Cloud-basierte Ressourcen unterstehen einer permanenten Überwachung. Durch diese Überwachung ist es dem Dienstleister möglich sein System zu optimieren, auch der Benutzer hat Zugriff auf diese Daten.

2.2 Cloud-Dienst-Arten

Beim Cloud-Computing werden folgende Dienstarten unterschieden [8]:

Cloud-Software-as-a-Service (SaaS) ist ein Dienstmodus in dem vom Benutzer verwendete Applikationen nicht am lokalen Rechner sondern in der Cloud ausgeführt werden. Auf die Applikation zugegriffen wird über eine Benutzerschicht welche zumeist Browser-basiert ist und nur geringe Einstellmöglichkeiten bietet. Der Benutzer hat keine weitergehenden Einflussmöglichkeiten auf die der Ausführung der Applikation zu Grunde liegenden Ressourcen. Der Vorteil dieser Art von Auslagerung von Applikationen ist der verminderte Wartungsaufwand, der beinahe gänzlich zum Dienstleister verlagert werden kann.

Bei der **Cloud-Platform-as-a-Service (PaaS)** kann der Benutzer eigene Applikationen auf einer vom Dienstleister zur Verfügung gestellten Plattform ausführen. Der Benutzer bleibt Herr über die Einstellungen des Programms und kann sogar begrenzt Einfluss auf die Konfigurationsparameter der Plattform nehmen. Wie bei SaaS hat der Benutzer keinen Einfluss auf die darunter liegenden Schichten.

Die unterste hier beschriebene Schicht ist die **Cloud-Infrastructure-as-a-Service (IaaS)**. Hierbei wird vom Dienstleister dem Benutzer eine Möglichkeit zur Provisionierung von Ressourcen gegeben. Der Benutzer kann also im vom Dienstleister erlaubten Rahmen selbsttätig über Speicher, Netzwerkverbindungen, Prozessoren, Betriebssysteme und ähnliches entscheiden. Die Einstellmöglichkeiten können sogar gewisse Freiheiten im Bezug auf die Firewalls eines Dienstleisters beinhalten.

Als weiteres Derivat ist **Data-Storage-as-a-Service (dSaaS)** zu erwähnen [4](Seite 5), welches reine Speicherdienste zur Verfügung stellt. Es ist als Teil- oder Untergruppe von IaaS einzuordnen.

Weitere Begriffe in diesem Zusammenhang sind **Hardware-as-a-Service**[10], diese weißt große Überschneidungen mit IaaS auf, ist allerdings auf die reine Server-

Hardwaresicht präzisiert (Prozessoren, Speicher usw.). Außerdem ist **Communications-as-a-Service (CaaS)** anzuführen, welches Kommunikationslösungen wie Telefonanlagen in die Cloud auslagert.

Diesen Schichten ist gemein, dass sie aufeinander aufbauen, die Gliederung ist wie in Abb. 2.2 gezeigt.

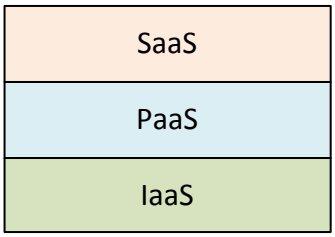

Abbildung 2.2: Cloud-Dienst-Schichten

Clouds sind nicht nur im Internet vorhanden, sondern können auch im lokalen Gebrauch verwendet werden. Generell werden von [8] 4 Arten von Clouds unterschieden:

- Die **private Cloud** ist zumeist bei Einzelunternehmen anzutreffen und wird auch von diesen verwaltet. Die Hardware befindet sich ebenfalls dort.
- Die **gemeinschaftliche Cloud** wird von mehreren Unternehmen oder Organisationen verwaltet und unterhalten.
- Die **öffentliche Cloud** wird zumeist von größeren Unternehmen betrieben und stellt ihre Dienste der Allgemeinheit zur Verfügung, meist gegen Bezahlung.
- Als Mischform existiert auch die **hybride Cloud**. Sie besteht aus mehreren der oben genannten Cloud-Formen, die einzelnen Clouds sind untereinander durch standardisierte Protokolle interoperabel.

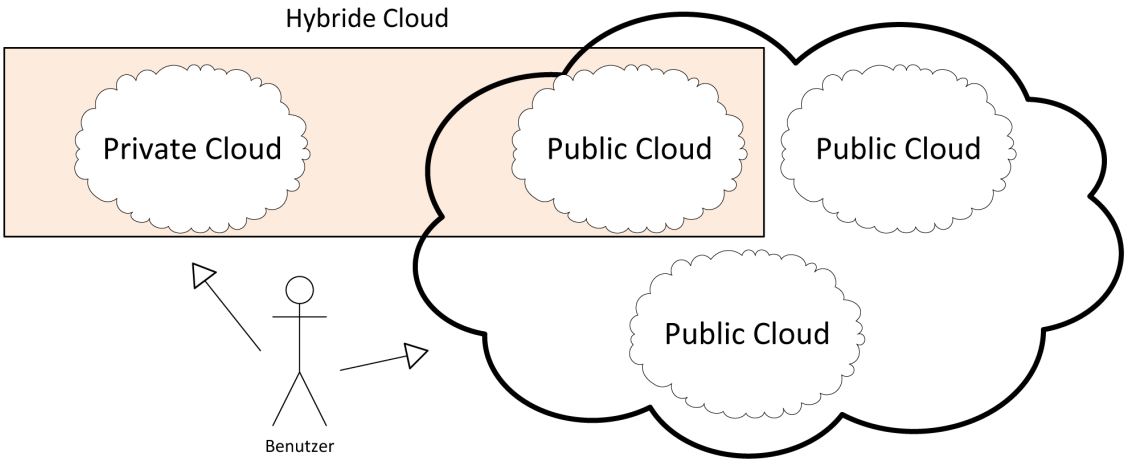

Abbildung 2.3: Beispiele für Cloud-Typen

Zur Illustration der obigen Angaben dient folgendes Beispiel, angelehnt an [4]:

In herkömmlichen IT-Abteilungen wird Beschaffung und Instandhaltung der Hardware und die Beschaffung und Wartung der darauf betriebenen Software intern durchgeführt. Das bedeutet im Detail, dass zuerst ein Ort für das Aufstellen der Hardware gefunden werden muss, dadurch ergeben sich Mietkosten. Ist ein geeigneter Platz vorhanden und der Einkauf der Hardware getätigt, so kann ein Mitarbeiter mit dem Einbau der Hardware und der Einrichtung einer darauf laufenden Software beginnen. Hierfür fallen Personalkosten zur Ausschreibung der Hardware an. Aufgrund der Länge der Beschaffungsprozesse müssen Reserven miteinkalkuliert werden, welche wiederum für höhere Kosten sorgen. Ist die Software betriebsbereit, so können die Benutzer beginnen damit zu arbeiten. Ist die Rechnerkapazität später nicht ausreichend, muss der Prozess von Beginn an wiederholt werden. Außerdem fallen der Austausch defekter Hardware und Stromkosten ins Gewicht.

Diese Kosten kann die Leitung einer Unternehmens-IT mithilfe von Clouds minimieren, falls sie den Großteil der Agenden an einen spezialisierten Dienstleister auslagert. Platzvorhaltung und Einkauf von Hardware sind nicht mehr notwendig, Cloud-**IaaS** lässt den Benutzer innerhalb kurzer Zeit Hardware zur Benutzung bereitstellen. Diese Dienstleistungen stellen zum Beispiel Fujitsu oder Amazon zur Verfügung. Um auf der nun ausgelagerten Hardware sinnvoll arbeiten zu können, kann die IT-Leitung entweder den vorhin beschriebenen Prozess weiter verfolgen, oder lässt sich von einem Cloud-Dienstleister **PaaS** anbieten. Dem Kunden wird bei PaaS eine Plattform angeboten, d.h. er bekommt auch noch eine fertige Betriebssystemumgebung für seine Software zur Verfügung gestellt (Beispiel: Microsoft Azure). Gibt es keine besonderen Anforderungen an die Plattform oder Infrastruktur, ist es möglich bei der Auslagerung noch weiter zu gehen und sich **SaaS** anbieten zu lassen. Damit gibt der Kunde große Teile der Einstellungsmöglichkeiten auf, bekommt aber vom Dienstleister garantierte Serviceleistungen zugesichert, deren Einhaltung er nur mehr kontrollieren muss (soweit sie nicht offensichtlich sind). Ein Beispiel aus der Praxis ist hier das Auslagern von Exchange-Diensten (Beispielfirmen: Salesforce, Cortado, United Internet, Microsoft Online).

Ein grundlegendes Problem im Zusammenhang mit der Auslagerung von Diensten (Hardware, Plattform oder Software) stellt natürlich die Auslagerung selbst dar. Ist aus externen Gründen (eventuell höhere Gewalt) kein Zugang zur Cloud des Dienstleisters möglich, so ist unter gewissen Bedingungen (kein lokaler Cache) kein Betrieb bzw. Zugriff auf die Dienste in der Cloud möglich. Dies kann bis zum Stillstand des Tagesgeschäfts eines Unternehmens führen. Die genaue Abwägung zwischen auslagerbaren Diensten und nicht auslagerbaren Kerndiensten gehört daher zu den schwierigen Entscheidungen für die Leitung eines Unternehmens.

Überblicksartig wird in Abbildung 2.4 das Zusammenspiel der einzelnen Elemente einer Cloud beschrieben.

2.3 Basistechnologien

Cloud-Computing ist ohne die Zuhilfenahme von einheitlichen Basistechnologien nicht möglich, da ohne sie keine Interoperabilität gegeben ist. Es haben sich im Laufe der Zeit De-Facto-Standards etabliert, die heute vom Großteil der Dienstleister unterstützt bzw. benutzt werden.

Die bereits erwähnte Eigenschaft der kurzfristigen Provisionierbarkeit basiert auf

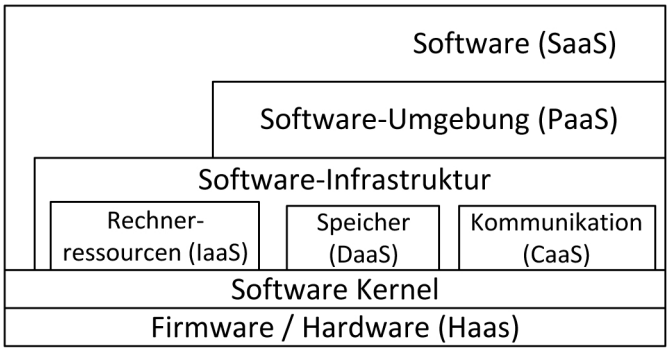

Abbildung 2.4: Elemente eines Cloud-Dienstes, aus [10]

der Fähigkeit der Dienstleister zur **Virtualisierung** von Rechnerressourcen. Prinzipiell basiert Virtualisierung im Rechnerkontext darauf Aufrufe des Betriebssystems an die Hardware abzufangen, zu validieren und gegebenenfalls unter der Schirmherrschaft eines Hypervisors durchzuführen. Der Hypervisor (vgl. [11]) ist für den reibungslosen Ablauf der Virtualisierung zuständig und schirmt das Betriebssystems einer virtuellen Maschine von der eigentlichen Hardware auf der sie läuft ab. Eine virtuelle Maschine verhält sich im Betrieb beinahe ident zu einem physikalischen Pendant.

Beispiele für im industriellen Betrieb befindliche Virtualisierungssysteme sind XEN, RedHat KVM, Microsoft Hyper-V und VMWare. Diese Technologien reichen von der vollen Virtualisierung wie bei VMWare zur Bereitstellung geeigneter APIs für eine hochperformante Virtualisierung unter Xen. Während schnittstellenbasierte Virtualisierer höhere Performance bereitstellen können [11], sind die vollen Virtualisierungslösungen out-of-the-Box einsetzbar (sofern deren Standardhardware vom Gast-Betriebssystem unterstützt wird). Nachdem die Spezifikation von XEN öffentlich einsehbar ist, wird diese im Folgenden als Beispiel herangezogen.

Eine Besonderheit stellt die Paravirtualisierungs-Unterstützung moderner Prozessoren dar, wie „AMD-V“ oder „Intel VT“, welche auch Betriebssystemen, die nicht für eine schnittstellenbasierte Virtualisierung angepasst sind, wie zum Beispiel für XEN nötig, die Fähigkeit zur vollen Abstraktion der Hardware-Aufrufe geben ohne dass dies das Gastbetriebssystem erkennt.

Als Besonderheit kommt bei XEN eine sogenannte „Dom0“ zum Einsatz. Diese „Domänen“ sind bei XEN Container für die jeweiligen Gastbetriebssysteme und die Kapselung der Anfragen. Die Dom0 hat eine privilegierte Stellung und übernimmt die Verwaltungsfunktionen des Systems. Abbildung 2.5 zeigt den Aufbau eines Virtualisierungssystems mit XEN, es bestehen offensichtliche Ähnlichkeiten zum Aufbau von IaaS, PaaS, etc.

Die für die Virtualisierung gebräuchlichste Hardware-Architektur X86 stellt vier Privilegierungsniveaus zur Verfügung, welche auch Ringe genannt werden [11](Seite 167). Auf Ring 0 wird das Betriebssystem mit dem höchsten Niveau an Berechtigungen ausgeführt. Auf Ring 3 laufen unter herkömmlichen Betriebssystemen die Benutzerapplikationen, Ring 2 und 3 bleiben durchgehend ungenutzt. Diesen Umstand machen sich Virtualisierungslösungen wie XEN zu Nutze und schieben ihre Virtualisierungsschicht zwischen Applikationscode-Schicht und Betriebssystemschicht, womit sie effizient eine Umgehung der Virtualisierung verhindern können.

Der Ablauf eines einzelnen virtualisierten Aufrufs wird in Abbildung 2.6 am Beispiel

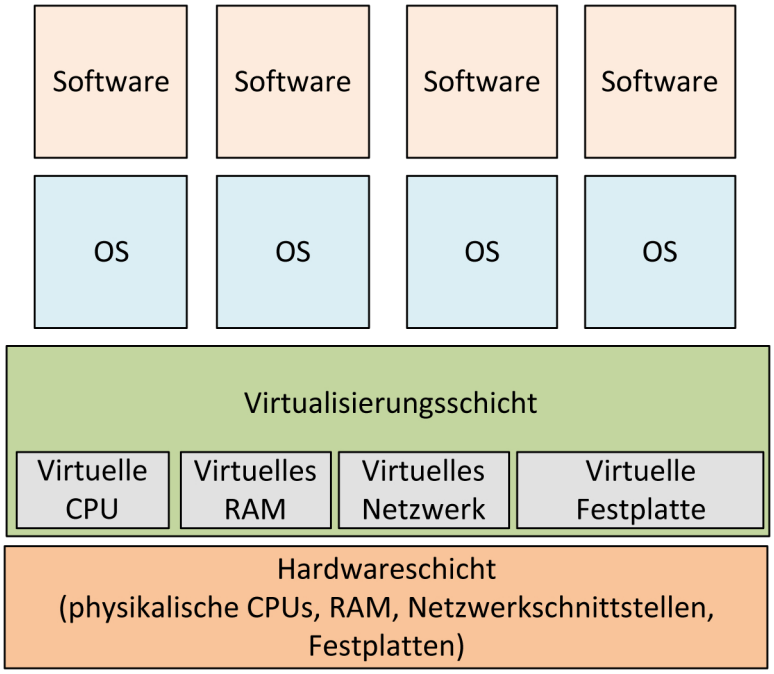

Abbildung 2.5: Virtualisierungs-Layer, adaptiert aus [11]

eines Zugriffs auf eine Harddisk gezeigt. Hierbei kommt ein Ringpuffer zum Einsatz, in welchem die Aufrufe abgespeichert werden, bis sie validiert und verarbeitet sind. Der Hypervisor arbeitet diese nicht notwendigerweise sequenziell ab, sondern kann sie zu Performance-Optimierungen auch umgruppieren bzw. ohne Rücksicht auf ihren Einfügungszeitpunkt vorzeitig zur Ausführung bringen. Antworten werden dort ebenso verarbeitet, bis sie an das Gastbetriebssystem zurückgegeben werden.

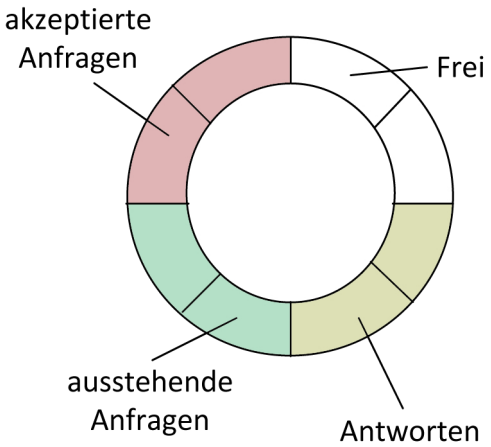

Abbildung 2.6: Ringpuffer für IO-Virtualisierung bei XEN, adaptiert aus [11]

Zur Virtualisierung von Hauptspeicher und CPU kommen ähnlich komplexe Techniken zum Einsatz. Beispielsweise werden von der Virtualisierungsschicht gegenüber dem Gastbetriebssystem durchgehende Hauptspeicherbereiche angegeben, obwohl diese auf der physikalischen Ebene nicht zusammenhängend sein müssen. Gelöst wird das über spezielle Übersetzungstabellen, welche vom Betriebssystem unterstützt werden müssen. Für die Zuteilung von Prozessorzeit implementiert der Hypervisor bei XEN einen „Borrowed Virtual Time (BVT)“-Algorithmus, welcher über

die gleichmäßige Verteilung an alle Gastsysteme wacht, wobei es standardmäßig eine Präferenz in der Zuteilung für neu aufgewachte Gastsysteme gibt [11].

Virtualisierungssysteme wie XEN können durch diese Mechanismen, welche von den Herstellern unterschiedlich implementiert werden, auf Hardware welche im nicht virtualisierten Betrieb nur die Ausführung eines einzelnen Betriebssystems zulassen würden, mehrere Gastbetriebssysteme zur Ausführung bringen. Bei in der Praxis vorkommender geringer Auslastung von Servern für spezielle Dienste, wie beispielsweise einem Testserver für ein gerade ruhendes Entwicklungsprojekt, bedeutet das im Regelfall wesentlich geringere Kosten durch einen geringeren Bedarf an Hardware und deren Wartung.

Ein anderer wichtiger Punkt bei der Bereitstellung von Cloud-Diensten ist die Verwendung von **Standardprotokollen**. Die Cloud ist in ihrem Inneren und im Zusammenspiel mit dem Kunden ein Konglomerat an verschiedenen Softwaresystemen. Da die Verwendung von proprietären binären Formaten zur Datenübertragung die Komplexität eines solchen Systems exponentiell ansteigen lässt, wurde eine Architektur entwickelt, die dieser Entwicklung Einhalt gebietet. Sie trägt den Namen **„Service Oriented Architecture“** (kurz SOA). Die Benutzung von standardisierten, teilweise auch vom Menschen lesbaren Formaten (vor allem dem XML-Format) spielt eine große Rolle. Die Notwendigkeit für einen derartigen Ansatz belegt die Darstellung der Abhängigkeiten der Softwarekomponenten in einem Elektrounternehmen in Abb. 2.7 aus [12].

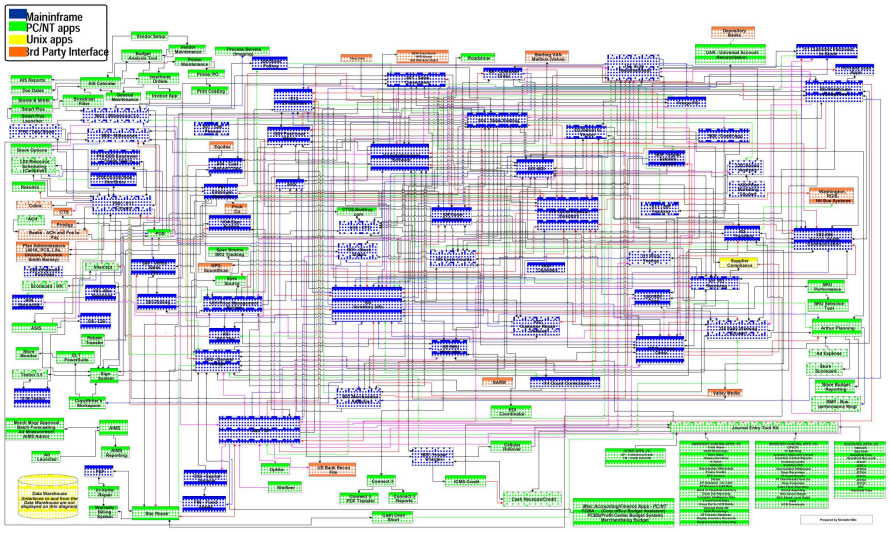

Abbildung 2.7: Abhängigkeiten der Softwarekomponenten eines Elektrounternehmens

Alonso und Canati definieren in ihrem Werk „Web services and service-oriented architectures“ [13] folgende Charakteristika einer SOA:

- Softwarebestandteile einer SOA sind lose gekoppelt
- Der Aufenthaltsort eines Softwarebestandteils ist transparent, das bedeutet der Benutzer einer Schnittstelle kann diese ohne Wissen um ihren Aufenthaltsort benutzen
- Kommunikation zwischen Bestandteilen einer SOA ist protokollunabhängig

Weiters wird ein Dienst in diesem Zusammenhang als autark definiert, d.h. er hat keine Abhängigkeiten zu anderen Diensten oder deren Stati. Eine SOA ist eine Sammlung solcher autarker Dienste die miteinander kommunizieren.

Die bei SOA vorgesehene Technik zum Auffinden eines Dienstes zeigt Abbildung 2.8.

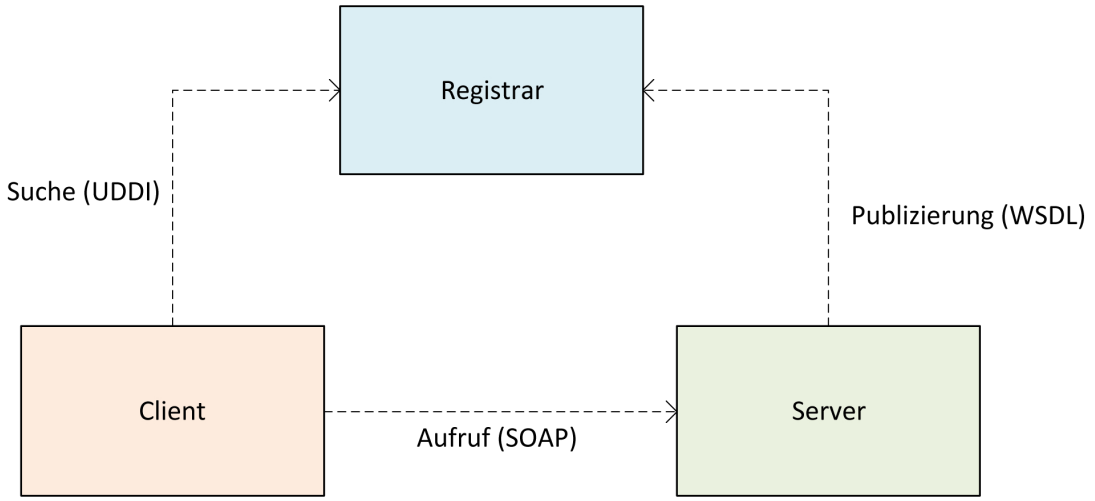

Abbildung 2.8: Client-Server-Registrar-Beziehung bei einer SOA

Die Implementierung der oben genannten Forderungen hat zur stärkeren Modularisierung von Software geführt, wobei die Migration von bestehender Software auf ein SOA-Konzept ein längerer Prozess sein kann. Als protokollunabhängige Kommunikationsformen haben sich in der Praxis [14] diverse Standards durchgesetzt, wie beispielsweise:

- SOAP: Das Service-Oriented-Architecture-Protokoll überträgt über HTTP und mittels XML Daten über das Netzwerk.
- JSON: Die „JavaScript Object Notation“ stellt ein auf JavaScript basierendes Protokoll dar, welches eine Alternative zu XML darstellt.
- XML-RPC: Im Gegensatz zu früheren RPC-Implementierungen basiert die „Extensible Markup Language Remote Procedure Call“-Implementierung auf Verwendung des Protokolls HTTP und des Datendarstellungsformates XML.
- REST: Dieses Akronym steht für „Representational State Transfer“ und beschreibt den Zugriff auf eine API mittels typischer HTTP-Befehle wie POST oder GET.

Bis auf JSON handelt es sich um XML-basierte Kommunikationsprotokolle, welche als Transportprotokoll HTTP oder dessen verschlüsselte Variante HTTPS verwenden. Durch die Verwendung des XML-Standards bleiben die Nachrichten prinzipiell für einen Menschen lesbar, dadurch wird die Fehlersuche erleichtert. Durch die Flexibilität der Elemente einer XML-Nachricht sind sehr verschiedene Einsatzmöglichkeiten gegeben, wodurch sich das Protokoll als Basis der meisten Nachrichtenübermittlungen im Rahmen einer SOA etabliert hat. Allerdings wird diese erhöhte Interoperabilität durch einen Mehraufwand an übertragenenem Speichervolumen erkauft, weswegen Binärprotokolle vor allem in hochperformanten Systemen nicht gänzlich obsolet sind.

Webservices machen die externen Schnittstellen von SOA-basierter Software über die oben genannten Protokolle nach außen zugänglich. Sie werden gemäß [15] wie folgt definiert:

> *Ein Webservice ist ein Software-System, das dazu entworfen ist Interaktionen zwischen Maschinen über ein Netzwerk zu unterstützen. Seine Schnittstelle(n) sind in einem maschinenlesbaren Format beschrieben. Andere Systeme interagieren mit dem Webservice über SOAP-Nachrichten auf eine Weise, die von der Dienstbeschreibung vorgegeben ist, typischerweise auf Basis von HTTP und XML-Serialisierung, und/oder mittels anderer Web-Standards".*

Im Idealfall ist eine externe Schnittstelle bei SOAP mittels der „Web Service Description Language" beschrieben und auf diese Weise leicht für Softwareentwicklungsprojekte zugänglich. Ansonsten muß anhand einer Schnittstellenbeschreibung die Anpassung der Client-Software vorgenommen werden.
In ihrer Gesamtheit ermöglichen die in Kapitel 2.1 beschriebenen Technologien den effizienten Einsatz von Software in einer Cloud.

2.4 Risiken und Metriken

Die Auslagerung von Software bzw. Hardware in eine Cloud hat nicht nur positive Eigenschaften sondern birgt auch Risiken, die vor einem verantwortungsvollen Einsatz von Software in der Cloud bemessen werden müssen. Aus diesem Grund wird in diesem Kapitel näher auf diese bei einer Entscheidungsfindung für die Auswahl des richtigen Dienstleisters wichtigen Kriterien eingegangen.
Eine Studie des Europäischen Netzwerk- und Informationssicherheitsdienstes (ENISA) beschreibt die wesentlichen Risiken im Zusammenhang mit der Verwendung einer Cloud [16], hier wird eine Auswahl genannt:

- **Verlust der Kontrolle:** Unterschiedliche Einschätzungen von Prioritäten im Bezug auf die Eigenschaften einer Auslagerung in die Cloud kann Nachteile nach sich ziehen, da die alleinige Kontrolle über die Dienstleistung nach der Auslagerung nicht mehr beim Auslagernden liegt. Auch genaue Beschreibungen der Dienstgüte auf Basis eines Service-Level-Agreements (dt. Dienst-Güte-Abkommens) können durch Deutungsunterschiede zu Diskrepanzen zwischen erwarteter und tatsächlicher Dienstgüte führen.

- **Lock-In-Effekt**: Durch diesen Effekt kommt es bei der Anpassung einer Softwarelösung an bestimmte Eigenschaften eines Cloud-Dienstleisters zu einer einseitigen Abhängigkeitsbeziehung zu diesem Dienstleister. Durch schlechtes Design kann es dazu kommen, dass ohne hohe Kosten keine Loslösung von einem den Anforderungen des Kunden nicht mehr entsprechenden Dienstanbieter mehr möglich ist.

- **Fehlerhafte Trennung**: Durch die Virtualisierung werden oft mehrere virtuelle Maschinen oder Dienste auf derselben physikalischen Hardware betrieben. Dadurch besteht die Gefahr, dass durch ausgefeilte Angriffe die Trennung zwischen diesen Diensten oder Maschinen überwunden werden kann (vgl. Abbildung 2.9).

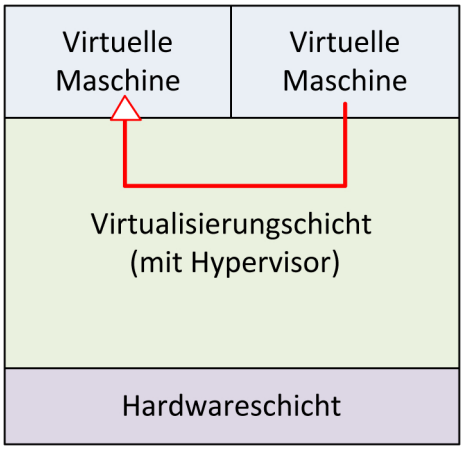

Abbildung 2.9: Möglicher Angriffsvektor über einen kompromittierten Hypervisor

- **Zertifizierungsprobleme**: Im betrieblichen Umfeld ist es für sicherheitsrelevante Anwendungen notwendig die Erfüllung gewisser Kriterien im Rahmen einer Zertifizierung nachzuweisen. Bei Clouds besteht das erhöhte Risiko, sicherheitsrelevante Umgebungsbedingungen aufgrund mangelnder Kontrolle oder Sicherheitsproblemen nicht erfüllen zu können.
- **Angriffe auf Cloud-Verwaltungswerkzeuge**: Bei öffentlich zugänglichen Clouds besteht wie bei jedem öffentlich zugänglichen Webdienst die Gefahr einer Kompromittierung durch einen nicht ausreichend abgesicherten Webserver. Durch den in diesem Fall gegebenen administrativen Zugriff besteht ein hohes Schädingungspotenzial.
- **Unvollständige Löschung**: Durch den verteilten und Orts-agnostischen Charakter von Cloud-Diensten besteht die Gefahr, dass nach einer benutzerinduzierten Löschung eines Inhaltes bei schlechter Implementierung des Löschalgorithmus durch den Dienstleister unerwünscht Kopien des Inhaltes beim ihm verbleiben.
- **Insider-Angriff**: Mitarbeiter des Dienstleisters, die mit administrativen Aufgaben betraut sind, können (eventuell unerlaubt) Zugriff auf die Daten eines Kunden haben. Dadurch entsteht die Gefahr des Geheimnisverlustes oder der Datenbeschädigung.

Neben den inhärenten Risiken des Cloud-Computing gibt es andere Eigenschaften eines Cloud-Dienstes, die beachtet werden müssen. [1] nennt dazu folgende Punkte:

- **Serviceverfügbarkeit:** Sie zeigt die Verfügbarkeit eines Dienstes über einen längeren Zeitrahmen hinweg. Im optimalen Fall sind das 99,999%.
- **Zuverlässigkeit:** Die Zuverlässigkeit bemisst sich als das Maß der Wahrscheinlichkeit des Verlustes von Daten bzw. der Ausfallwahrscheinlichkeit eines Dienstes
- **Support:** Dieser beschreibt die Möglichkeit des Kunden im Falle von Problemen auf Unterstützung des Dienstleisters zurückgreifen zu können und dessen Ausmaß sowie die Geschwindigkeit der Antwort des Dienstleisters (sog. Response-Zeit)
- **Performance:** Diese zeigt die Schnelligkeit eines Dienstes, also Antwortzeiten bzw. Übertragungsgeschwindigkeiten

- **Sicherheit:** Daten dürfen auch im Cloud-Umfeld nicht in unautorisierte Hände geraten. Unter Sicherheit fallen Maßnahmen wie Verschlüsselung, Authentifizierungsmethoden, Isolation der Daten von den Daten anderer Kunden etc.
- **Preise:** Ein relevantes Kriterium stellt im wirtschaftlichen Umfeld auch der Preis dar.
- **Verantwortlichkeiten:** Für eine effiziente Zusammenarbeit sind klare Zuständigkeiten und Ansprechpartner auf Seiten der Dienstanbieter vonnöten.
- **Metriken:** Zur Beurteilung des Betriebsverhaltens eines Cloud-Dienstes sind Messmethoden (wie CPU-, Speicher-, Festplattenauslastung, etc.) notwendig.

Anhand der hier genannten Punkte wird im Kapitel 4 eine Evaluation bestehender Speicherdienstleister vorgenommen.

Kapitel 3

Arten der Datenspeicherung

Zur Entwicklung eines für die Cloud sinnvollen Dateisystems ist es notwendig, sich einen Überblick über bereits bestehende Formen des Datenzugriffs bei verteilten Systemen zu verschaffen.

Prinzipiell basieren alle diese Arten auf dem einfachen Konzept der Blockspeicherung. Der Datenstrom aus Bits und Bytes, der den Inhalt einer Datei ausmacht, wird in logische Blöcke eingeteilt. Diese logischen Einheiten werden von den Speicherlesegeräten im Ganzen eingelesen und verarbeitet. Durch die Optimierung der Lesegeräte auf diese Art der Datenverarbeitung hat die Organisation von Daten in Blöcken weite Verbreitung gefunden. Geräte, welche diese Art der Datenspeicherung unterstützen, werden Blockspeichergeräte genannt.

Zum Zugriff auf Blockspeichergeräte haben sich Dateisysteme etabliert. Diese Dateisysteme ordnen einzelne Blöcke einer größeren Einheit zu, welche Datei genannt wird. Das Dateisystem hat die Aufgabe die richtige Zuordnung von Blöcken zu Dateien zu überwachen. Abbildung 3.1 stellt den Zusammenhang schematisch dar. Weitere mögliche Fähigkeiten eines Dateisystems sind:

- Zugriffskontrolle
- Verschlüsselung
- Replikation
- Organisation in Ordnern
- durchsuchbare Indexierung

Zur Beschreibung der Eigenschaften einer Datei wie Größe, Erstellungsdatum usw. kommen Metadaten zum Einsatz. Diese Metadaten sind zumeist in einem speziellen Metadatenbereich des Dateisystems verborgen und für den Benutzer nicht direkt einsehbar.

Ausfallsicherheit wird auf Blockebene durch eine Technologie namens RAID sichergestellt, welche hard- oder softwaregestützt Lese- und Schreibvorgänge auf weiteren Blockspeichergeräten repliziert. Sie wird in Kapitel 3.1 näher beschrieben.

Dateisysteme können in mehrere Kategorien unterteilt werden, welche in unterschiedlichem Maße den Anforderungen an eine Datenspeicherung in der Cloud entsprechen[8]:

[8] http://en.wikipedia.org/wiki/File_system

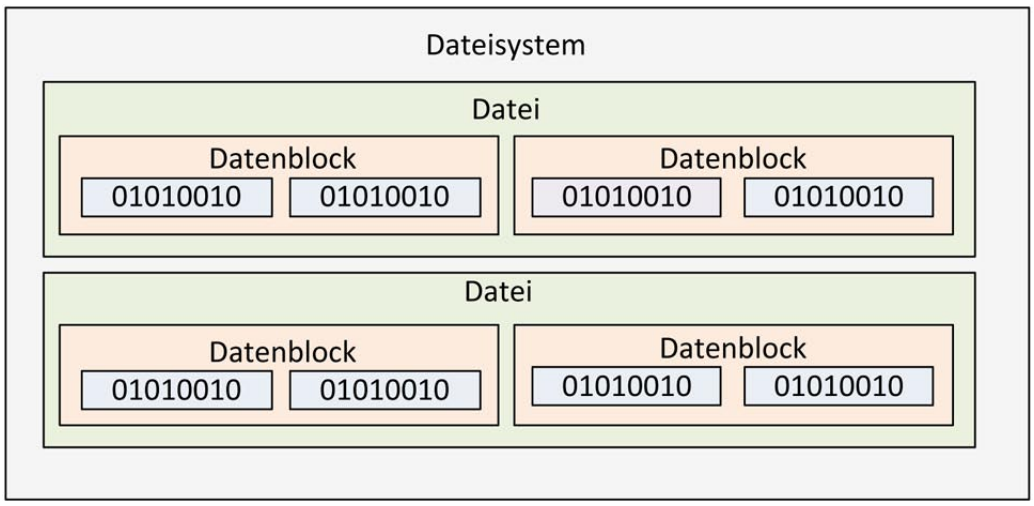

Abbildung 3.1: Zusammenhang Dateisystem - Blockspeichergeräte

- Dateisysteme für Festplatten
- Dateisysteme für Flashspeicher
- Dateisysteme für Bänder
- Datenbankdateisysteme
- Transaktions-Dateisysteme
- Netzwerk-Dateisysteme
- Dateisysteme für Disksysteme, auf die gleichzeitig von verschiedenen Stellen aus zugegriffen wird
- Dateisysteme für Spezialanwendungen

Am Ende des Kapitels werden bestehende Lösungsansätze zu diesem Problem untersucht. Hierbei gibt es Ansätze, die mit konventionellen Dateisystemen große Ähnlichkeit besitzen und sich mittels eines Treibers in die Kommunikation zwischen Benutzer und Festplatte einhängen, wie auch Ansätze mittels lokaler Überwachung durch einen Service oder Bereitstellung eines Interfaces durch einen Web-Server.

3.1 RAID

RAID, in der Langform **„Redundant Arrays of Inexpensive Disks“** (im Deutschen in etwa „Redundante Zusammenstellung billiger Festplatten“) ist laut [17] ein Mittel gegen das Risiko des Datenverlustes, das bei der Aufbewahrung von Daten im Allgemeinen, bei der Speicherung von Daten auf mehreren Datenträgern aber in höherem Maße auftritt.

Die Speicherung von Daten in einer Cloud entspricht diesem Risikoprofil, es müssen daher Vorkehrungen für den Fall des Datenverlustes getroffen werden.

Die Ausfallsicherheit eines RAID-Systems bemisst sich laut [18] als

$$R_{RAID} \approx 1 - \sum_{j \geq n} a_j \in^j \approx 1 - a_n \in^n .$$

Im Folgenden werden die bei RAID gebräuchlichen Ansätze zur Verminderung des Ausfallrisikos beschrieben. [17] nennt dazu folgende Arten:

Während **JBOD („Just a bunch of disks“)** im engeren Sinne kein RAID darstellt, ist es in diesem Zusammenhang trotzdem zu erwähnen. Hierbei wird durch einen Verwaltungsdienst eine Gruppe von (möglicherweise unterschiedlichen) Festplatten zu einem Festplattenverbund zusammengefasst. Im Gegensatz zu den anderen RAID-Arten geht es hierbei nicht um Faktoren wie Geschwindigkeit oder Ausfallsicherheit, sondern um die Zusammenfassung von mehreren physikalischen Datenträgern zu einem logischen Datenträger.

Dies geschieht für den Nutzer transparent, für ihn stellt sich das JBOD in der Benutzung als eine einzige Festplatte dar.

Der Nachteil dieser Form eines RAIDs ist die stark erhöhte Ausfallwahrscheinlichkeit, da diese ohne Duplizierung der Daten, wie oben beschrieben, stark ansteigt.

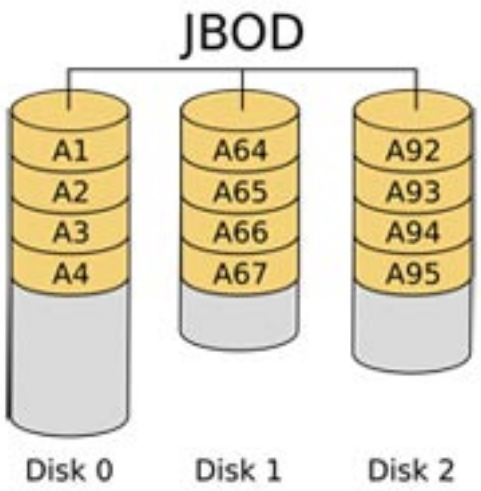

Abbildung 3.2: Verteilung von Daten bei JBOD, aus [19]

Als **RAID 0** gilt eine Zusammenstellung von Speichern, wenn sie das Prinzip des Data Striping verwendet. Data Striping bedeutet bei block-basiertem Striping laut [17](Seite 470) die Aufteilung der Bestandteile eines Blockes auf mehrere Datenträger. Beim Einlesen der Datei kann durch diesen Parallelismus, das Senden bzw. Empfangen mehrerer Bestandteile einer Datei zum selben Zeitpunkt eine höhere Geschwindigkeit bei der Datenübertragung erreicht werden. RAID 0 setzt keine Maßnahmen zur Sicherung vor Datenträgerausfall, sondern ist nur auf Geschwindigkeit ausgerichtet. Die Anzahl der benötigten physikalischen Datenträger liegt bei $n * 2$, bei $n \geq 2$, wobei n die Anzahl der logischen Datenträger ist.

RAID 1 ist eine Form des RAIDs, welche Funktionen für sichere Datenaufbewahrung bietet. Jede Schreiboperation auf einem Datenträger wird durch den RAID-Controller, welcher Schreib- und Lesezugriffe auf die Datenträger eines RAIDs überwacht, auf die anderen Datenträger im RAID 1 repliziert.

Durch diese Maßnahme beträgt die Ausfallwahrscheinlichkeit für ein bestimmtes Dateisystemobjekt, abgesehen von benutzerinduzierten Einflüssen oder Stromaus-

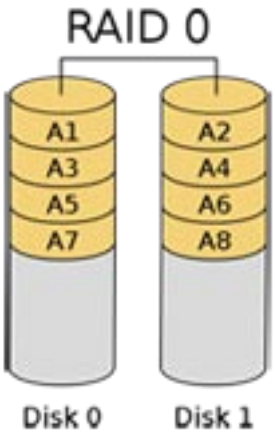

Abbildung 3.3: Verteilung von Daten bei RAID 0, aus [19]

fällen, also bezogen rein auf die plötzliche Fehlfunktion eines Datenträgers selbst, statt $1/n$ dann $1/n^2$.

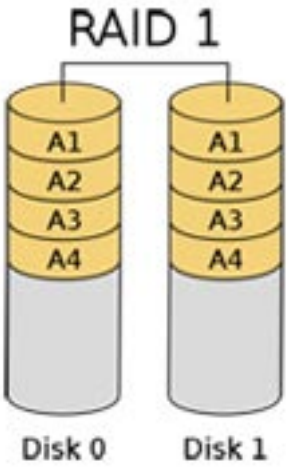

Abbildung 3.4: Verteilung von Daten bei RAID 1, aus [19]

RAID 2 benutzt den Mechanismus der Fehlerkorrektur durch Parity-Bits. Diese speziellen Bits zeigen an ob ein Nutzdatenbit gerade oder ungerade ist. Sollte das Nutzdaten- oder Paritybit beschädigt sein, so ist dies sofort ersichtlich und kann korrigiert werden. Ein Byte kann so beispielsweise bitweise auf Datenträger eines RAIDs aufgeteilt werden, die Parity-Bits werden auf zusätzlichen Datenträgern untergebracht. Fällt einer der Datenträger aus, so ist es dem RAID-Controller möglich anhand der verbliebenen Datenträger und der Parity-Informationen die restlichen Daten zu rekonstruieren. Vorteil dieses Verfahrens ist, dass es weniger Datenträger zur Sicherung der Daten benötigt als RAID 1.

RAID 3 führt den Gedanken von RAID 2 weiter, nutzt allerdings die Eigenschaft von Harddisk-Controllern, das korrekte Auslesen eines Sektors verifizieren zu können. Dazu stellt es den Ort des korrumpierten Bits fest und vergleicht dessen Parity-Daten mit den korrespondierenden Daten auf anderen Datenträgern. Es wird für jedes Bit in einem Sektor aufgrund der Paritätsdaten in diesem Sektor sein binärer Zustand (0 oder 1) festgestellt und das korrupte Bit mit den verbleibenden verglichen. Wenn die Parität dieser Bits gleich der für sie gespeicherten Parität ist, ist das

fehlende Bit 0, ansonsten 1. Durch diese bessere Erkennung von fehlerhaften Daten verbraucht RAID 3 im Gegensatz zu RAID 2 weniger Speicherplatz.

RAID 4 operiert nicht wie RAID 2 und 3 auf Bit-/ oder Byte-Ebene sondern auf Blockbasis. Es verteilt Blöcke wie RAID 0 auf mehrere Datenträger und hält wie RAID 2 und 3 Parity-Blöcke zur Datenrekonstruktion bereit, allerdings auf einer dafür designierten Festplatte. RAID 4 hat einen höheren Datendurchsatz als RAID 2 und 3 beim Lesen, allerdings einen geringfügig geringeren Schreibdurchsatz. Aufgrund der Tatsache, dass an einer Schreiboperation immer die Paritätsfestplatte beteiligt ist, führt dies zu einer erhöhten Belastung und höheren Ausfallraten. Die RAID-Level 2, 3 und 4 werden in der Praxis selten verwendet.

RAID 5 führt das Konzept von RAID 4 weiter, verteilt allerdings die Paritätsinformationen auf alle Festplatten. Durch diese Kombination von Striping, einer Block-Basis und verteilten Paritätsinformationen kann die Fehlerhäufigkeit im Vergleich zu RAID 4 vermindert werden. Gleichzeitig bleiben die Vorteile eines erhöhten Durchsatzes erhalten. Aus diesem Grund ist RAID 5 in Unternehmen das am meisten genutzte RAID-Format.

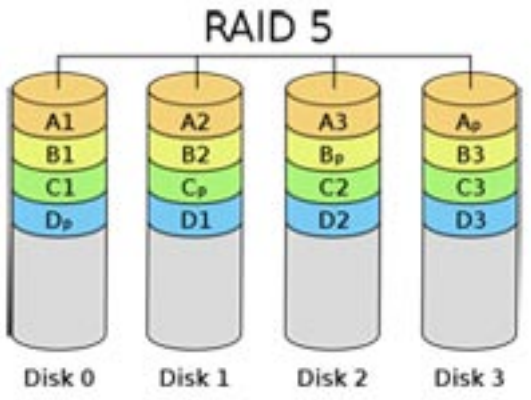

Abbildung 3.5: Verteilung von Daten bei RAID 5, aus [19]

RAID 6 fügt RAID 5 weitere Paritätsdaten hinzu um gegen doppelte gleichzeitige Datenträgerausfälle geschützt zu sein. Die Implementierungen sind unterschiedlich, eine Variante setzt anstatt der Paritätsinformationen „Reed-Solomon-Code“ ein. Durch die mehrfache Speicherung der Paritätsinformationen steigen die Kosten im Gegensatz von RAID 5 und der Datendurchsatz vermindert sich

Als **RAID 10** wird eine Kombination von RAID 1 und RAID 0 bezeichnet, bei der wie bei RAID 1 zuerst Festplatten gespiegelt werden. Die resultierenden Festplatten-Spiegel-Paare werden dann wiederum wie bei RAID 0 in einem Stripe zusammengehängt. Diese Kombination bietet die Möglichkeit die Geschwindigkeit von RAID 0 mit der Redundanz von RAID 1 zu verbinden. Ein Nachteil ist der erhöhte Bedarf an Speichergeräten im Vergleich zu RAID 5.

RAID 50 bezeichnet die Kombination von RAID 5 und RAID 0. Wie bei RAID 10 werden die Vorteile von RAID 5 (mäßiger Speichergerätebedarf, Redundanz, Geschwindigkeit) mit den Vorteilen von RAID 0 kombiniert. Dadurch wird im Vergleich zu RAID 10 eine größere Geschwindigkeit erreicht. Nachteil ist der größere Bedarf an Festplatten.

Die komplexeste hier vorgestellte RAID-Art ist **RAID 5E**. Diese Variante verbindet RAID 5 mit einem Hot-Spare-Laufwerk. Ein Spare-Laufwerk ist in einem RAID nor-

malerweise eine leere Festplatte, welche für den Fehlerfall bereit gehalten wird. Kann diese Festplatte sofort vom RAID-Controller in Betrieb genommen werden, spricht man von einem Hot-Spare. Bei RAID5E ist die Festplatte jedoch nicht im Standby, sondern der für den Fehlerfall benötigte Speicherplatz wird dynamisch auf allen Datenträgern vorgehalten und die Datenübertragsungskapazität der Spare-Festplatte kann mit genutzt werden, wodurch sich insgesamt eine höhere Geschwindigkeit des Systems ergibt.

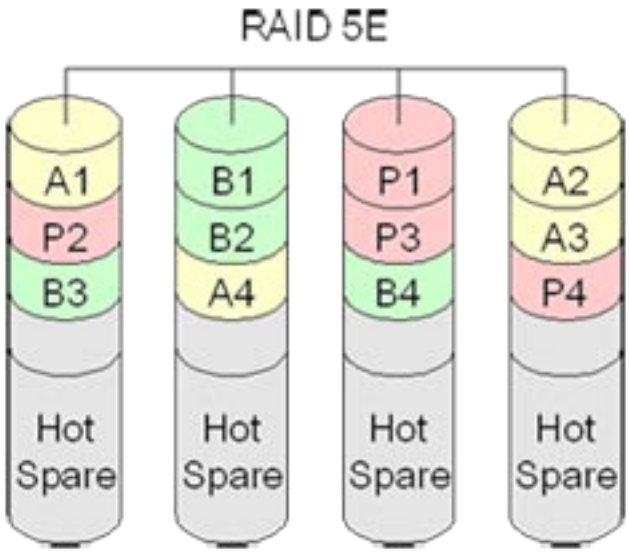

Abbildung 3.6: Verteilung von Daten bei RAID 5E, aus [19]

Neben diesen RAID-Formen existieren noch eine Vielzahl anderer Mischformen und Abwandlungen, die unterschiedliche Verhältnisse von Redundanz zu Geschwindigkeit aufweisen. Sie basieren auf den gleichen Prinzipien und werden deshalb hier nicht weiterführend beschrieben.

Abschließend stellt Tabelle 3.1 eine Übersicht über die gebräuchlichsten RAID-Level und ihre Eigenschaften dar.

3.2 Dateisysteme

Es wird von Dateisystemen auf unterschiedlichste Art und Weise versucht, die Probleme beim Aufbau von Dateisystemen zu lösen. Im Speziellen hat sich in letzter Zeit der Fokus bei der Erstellung neuer Dateisysteme weg vom lokalen Datenspeicher hin zu einem auf mehrere Orte verteilten Datenspeicher entwickelt. Etliche davon werden bereits heute von Cloud-Dienstleistern im Produktiveinsatz betrieben. Diese und andere Ansätze werden im folgenden beschrieben, um die am meisten geeigneten Prinzipien zum Aufbau eines solchen Systemes herauszuarbeiten.

3.2.1 Verteilte Dateisysteme

Ein verteiltes Dateisystem wird gemäß [21](Seite 3) als

RAID-Typ	0	1	2	3	4	5	6
Anzahl Laufwerke	n > 1	n = 2	N = 10	n > 2	n > 2	n > 2	n > 3
Redund. Laufwerke	0	1	2	1	1	1	2
Größe Overhead (%)	0	50	20	100 / n	100 / n	100 / n	200 / n
Parallele Leseoperationen	n	2	8	n - 1	n - 1	n -1	n - 2
Parallele Schreiboperationen	n	1	1	1	1	n / 2	n / 3
Max. Lesedurchsatz	n	2	8	n - 1	n - 1	n - 1	n - 2
Max. Schreibdurchsatz	n	1	1	1	1	n / 2	n / 3

Tabelle 3.1: Vergleich von RAID-Varianten, aus [20]

> *[...] ein Filesystem, dessen Clients, Server und Speichergeräte über die Maschinen eines verteilten Systemes verteilt sind [...]*

definiert.
Ein verteiltes Dateisystem definieren Levy und Silberschatz als

> *[...] eine Ansammlung von lose gekoppelten Maschinen [...], die durch ein Kommunikationsnetzwerk verbunden sind.*

Ein prominenter Vertreter eines solchen verteilten Dateisystems neuester Bauart ist das **Google File System**[22]. Das Google File System revidiert einige gängige Annahmen bei der Erstellung verteilter Dateisysteme. Beispielsweise gehen die Architekten des Dateisystems davon aus, dass Ausfälle von Bestandteilen des verteilten Systems die Regel und nicht eine Ausnahme sind, da bei steigender Anzahl von Komponenten auch die Wahrscheinlichkeit eines Ausfalls mitskaliert. Es wird im Gegensatz zu normalen Blockdateisystemen von einer höheren Größe eines Files ausgegangen, da der durchschnittliche Platzverbrauch einer Datei im Vergleich zum

Zeitpunkt der Erstellung älterer Dateisysteme angestiegen ist. Veränderungen an den Daten werden im Gegensatz zu anderen Systemen auf das Anhängen von Daten hin optimiert, da bei großen Files diese Operationen weitaus häufiger vorkommen als bei kleineren. Außerdem wurde beim Design des GFS auch auf das Design der Programme, die es benutzen, Rücksicht genommen.

Die Architektur des Google File Systems besteht aus einem einzelnen Master-Server, welcher die Metadaten des gesamten Dateisystems verwaltet und andere Bestandteile, wie die Zugriffskontrolllisten oder Chunks verwaltet, und den Chunk-Servern. Chunks werden die Einzelteile einer bestimmten Datei genannt, welche auf die Chunk-Server verteilt werden. Datensicherheit wird durch mehrfache Speicherung der Chunks auf verschiedenen Chunk-Servern erreicht, wobei der veränderbare Standardwert für die Replizierungshäufigkeit 3 ist. Nachdem das GFS kein POSIX-Dateisystem ist, d.h. Standard-Dateisystem-Operationen nicht in vollem Umfang unterstützt, ist der Support für dieses Dateisystem in die Client-Programme eingebaut.

Einen Überblick über die Architektur des GFS bietet Abb. 3.7.

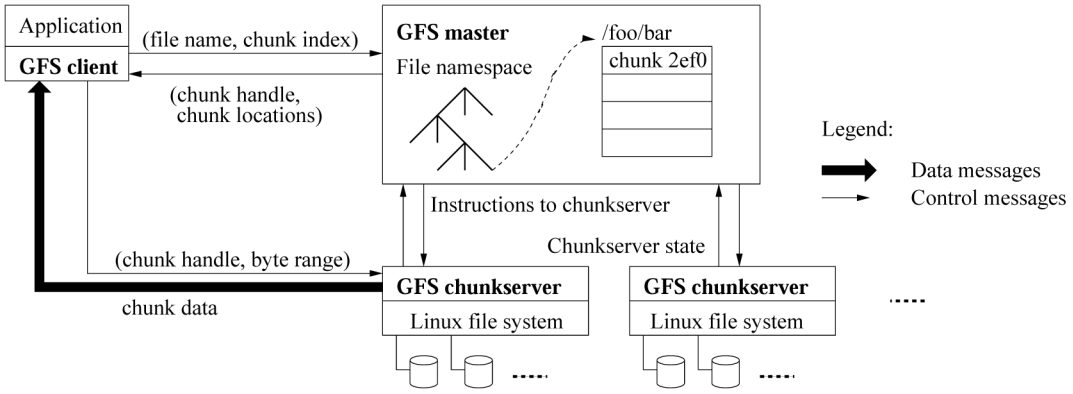

Abbildung 3.7: Google File System - Architektur, aus [22](Seite 31)

Die Teilnahme des Master-Servers wird bei Dateisystemoperationen des GFS minimiert, um keine Engpässe zu verursachen. Beispielsweise wird der Master-Server nur zu Beginn einer Leseoperation nach dem Ort der Chunks auf den Chunk-Servern befragt, er liefert daraufhin den Aufenthaltsort der Repliken und teilweise auch Informationen zu angrenzenden Chunks zurück. Nachdem diese Information am Client gecached wird, sind bis zum Ablauf der Gültigkeitsdauer des Caches in dieser Sache keine weiteren Interaktionen des Clients mit dem Masterserver mehr notwendig.

GFS ist auf große Dateien und Dateimengen optimiert, die Standardgröße für Chunks liegt bei 64MB, dies erfordert laut den Architekten weniger Zugriffe auf das Dateisystem.

Die Metadaten bestehen aus Datei-zu-Chunk-Maps, Chunk-Aufbewahrungsorten und den eigentlichen File- und Chunk-Tabellen. Veränderungen an den Tables und den Maps werden ähnlich einem Journaling-Filesystem in einem Log abgespeichert, die Aufbewahrungsorte nur im Hauptspeicher. Die Aufbewahrung der Daten im Hauptspeicher erlaubt schnellen Zugriff. Durch die Flüchtigkeit von Daten im Hauptspeicher werden die Chunk-Aufbewahrungsorte bei jedem Start und ab dann periodisch eingelesen.

Weiters bedient sich das GFS einiger weiterer Methoden um seine Ziele erreichen zu können:

- Namensräume und Verzeichnisse: Dateien sind nicht nach Ordnern sortiert, sondern die Ordner sind in die Dateinamen kodiert. Durch eine Präfixfilterung kann ein ressourcensparender Locking-Mechanismus gefahren werden.

- Replika-Management: Beim Erstellen von Replikas einzelner Chunks wird darauf geachtet, dass diese auf Servern mit unterdurchschnittlicher IO-Belastung durchgeführt werden. Bei der Wiederherstellung bei Chunk-Verlusten (z.B.: durch einen Hardwareausfall) werden die Chunks mit weniger Replikas höher priorisiert.

- Löschung: Der Masterserver löscht Daten nicht sofort, sondern erst wenn keine Referenzen von Dateinamen auf Chunks mehr vorhanden sind. Das erspart komplizierte Algorithmen zur sicheren Löschung.

- Hohe Verfügbarkeit: Diese wird durch eine Kombination von Replikation und schneller Wiederherstellung erreicht. Chunks und die Daten des Master-Servers werden über mehrere Rechner repliziert. Bei Fehlern läuft der Master-Server-Prozess nach wenigen Sekunden wieder, sollte ein größeres Gebrechen vorliegen wird auf einem anderen Rechner auf Basis der replizierten Masterdaten ein neuer Master-Server-Prozess gestartet.

- Datenintegrität: Diese wird durch das Errechnen von Prüfsummen festgestellt. Jeder Chunkserver überprüft für sich diese Prüfsummen, stimmt ein 64KB-Block eines 64MB-Chunks nicht, so wird er von einer anderen Replika wiederhergestellt.

- Diagnose: Es werden Logs zur Nachverfolgung von Fehlern geschrieben, soweit dies der Platz auf den Servern zulässt.

3.2.2 Cluster-Dateisysteme

Cluster-Dateisysteme, wie z.B. das **GPFS** von IBM [23], besitzen Ähnlichkeiten zum Google File System, allerdings sind sie auf spezielle Hardware fokussiert. Markantester Unterschied ist beim GPFS die Benutzung eines Shared-Disk-Systems. Die Switching-Infrastruktur besteht in der Regel entweder aus einem Storage Area Network (SAN), einem Fibre-Channel-System oder einem iSCSI-System und bindet die einzelnen Cluster-Nodes, aus denen das gesamte Clusterdateisystem besteht, an das Shared-Disk-System.

Das GPFS verlässt sich zur Optimierung von Übertragungsgeschwindigkeiten vor allem auf Parallelismus. Dazu wird auf bewährte RAID-Techniken zurückgegriffen und die Daten in Stripes auf mehrere Datenträger verteilt. Um eine aufwändige Serialisierung von Zugriffen auf einzelne Bereiche des Disksystems zu verhindern, wird ein Byte-Bereichs-Locking verwendet, das paralleles Schreiben in einer Datei erlaubt. Beim Lesen wird versucht, die maximale Bandbreite der Switch-Infrastruktur durch eine Vielzahl an Lesekommandos an die Clusternodes auszunutzen.

Für die Unterstützung großer Verzeichnisse wird ein sogenanntes „erweitertes Hashing“ verwendet. Durch eine spezielle Hashfunktion kann aus dem Dateinamen und dem Hash der Datei der Block mit dem Verzeichniseintrag für die Datei gefunden werden.

Auch das GPFS bedient sich bei den Dateisystemoperationen des Journaling-Mechanismus, bei dem die Operationen zuerst in einem Log zwischengespeichert werden und erst

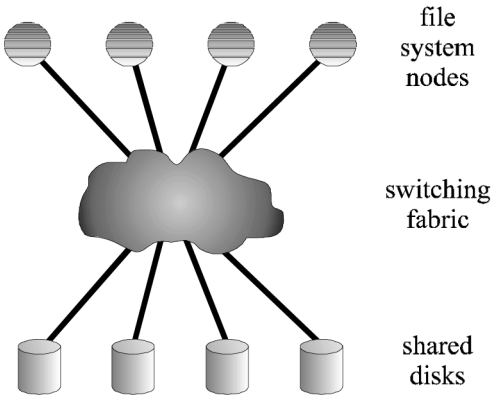

Figure 1: Shared Disk Environment

Abbildung 3.8: Aufbau GPFS aus [23]

bei erfolgreicher Durchführung der Operation aus diesem wieder gelöscht werden. Im Falle eines Fehlers braucht die Operation nur wiederholt zu werden.

Als Locking-Mechanismus zur Verhinderung von Dateisystem-Inkonsistenzen verfolgt das GPFS zwei verschiedene Ansätze, das zentralisierte Management und verteiltes Locking. Je nach besserer Eignung für die aktuelle Situation wird eines der beiden verwendet. Beim verteilten Locking werden bei Filesystemoperationen Locks erworben, die Operationen auf anderen Nodes während der aktuellen Operation verhindern. Dies geschieht zwischen den einzelnen Clusternodes. Beim zentralisierten Management übernimmt diese Aufgabe eine einzelne Node, welche die Operationen nacheinander ausführt. Das Lock-Regime auf einer Clusternode hat ein „Token-Manager“, welcher die Byte-Range-Locks verwaltet. Zur Kommunikation zwischen den Nodes kommt ein speziell dafür geschriebenes Locking-Protokoll zum Einsatz, welches auf minimierten Kommunikationsaufwand ausgerichtet ist.

Durch Zuweisungs-Tabellen (engl. Allocation Maps) werden beim GPFS die Informationen über freien und belegten Speicherplatz verwaltet. Ein spezieller Sektor pro Clusternode hat die Aufgabe diese Daten zu beinhalten. Weiters werden an alle in diesem Bereich tätigen Clusternodes Zuweisungs-Hinweise (sog. Allocation-Hints) mitverschickt, diese Informationen nutzen die Clusternodes zur Optimierung von Zugriffen.

3.2.3 Netzwerk-Dateisysteme

Vorgänger zahlreicher verteilter Dateisysteme ist das „Network File System“ (kurz NFS), welches auf der RFC 3530 [24] basiert. Im Designdokument des Dateisystems [5] beschreiben die Erfinder des Dateisystems die Grundlagen von NFS:

Wichtigster Bestandteil ist die Trennung des vom NFS verwendeten virtuellen Dateisystems (VFS) von der eigentlichen im Kernel befindlichen Implementation des Datenträgerdateisystems. Somit können durch die Verwendung des VFS je nach Maßgabe der Fähigkeiten des eingesetzten Kernels eine Vielzahl an unterschiedlichen Datenträger-Dateisystemen im Netz bereitgestellt werden.

Das NFS stellt ein Interface zur Verfügung, welches den Zugriff auf das virtuelle Dateisystem und die in ihm befindlichen Dateien regelt.

Wichtige Design-Grundsätze gemäß [5] sind für das NFS:

- Unabhängigkeit von Betriebssystem und Hardware
- Möglichkeit der Wiederaufnahme des Betriebs nach einem Crash
- Transparenter Zugriff, d.h. keine Spezialprogramme oder -bibliotheken
- brauchbare Performance

Auf Basis dieser Designziele wurde ein Design erstellt, welches aus 3 Hauptteilen besteht:

- Protokoll
- Server
- Client

Das Protokoll arbeitet in den Versionen 2 und 3 ohne vordefinierten Status, beinhaltet also alle für einen Vorgang notwendigen Daten. Dadurch wird die Fehlerbereinigung vereinfacht, da bei einem Fehler einfach der fehlgeschlagene Vorgang wiederholt wird, bis er erfolgreich abgearbeitet werden kann. Ab Version 4 des Protokolls ist das Protokoll nicht mehr statuslos, da es auch die Sperrung einzelner Dateien und Dateiteile (sog. „Locking“) unterstützt.

Als Basis des Protokolls dient das „Remote Procedure Call“-Protokoll (RPC, Remote-Prozeduren-Aufruf, [25]), welches es ermöglicht, Aufrufe eines Clientprogramms auf einem Server auszuführen. Situationsabhängige Parameter werden übermittelt. Auf Client-Seite wird auf die Verarbeitung des Aufrufs und die Rückmeldung des Servers gewartet. In älteren Varianten des NFS basierte RPC auf UDP, welches keine Erfolgsgarantie bei der Übermittlung bietet. Dieses wurde bei NFS in seiner neuesten Version 4 durch das TCP-Protokoll ersetzt.

Der Server ist für die Speicherung der Dateien am physikalischen Datenträger zuständig, wie auch in neueren Versionen für Locking und Bereitstellung der Daten im Netzwerk.

Der Client bindet das Netzwerkdateisystem transparent in die lokale Datenträgerstruktur ein (unter Unix/Linux-Systemen als Mountpunkt) und stellt so einen reibungsfreien Zugriff auf die Dateien sicher.

3.3 Unterschiede von Block- und Datei-basierten Speicherlösungen

Eine wichtige Design-Entscheidung für ein verteiltes Dateisystem stellt die Wahl eines geeigneten Verfahrens zur Verteilung der Daten dar. Die traditionelle Form der Speicherung der Daten auf einem Datenträger ist das Block-basierte Speicherungsverfahren. Es bildet die Basis für die meisten derzeit verfügbaren Dateisysteme die in Computersystemen Verwendung finden (und wird im Kapitel 3 exemplarisch beschrieben).

Die Vorteile des Block-basierten Datenträgerzugriffs sind gemäß [26]:

- große Verbreitung in allen Computersystemen

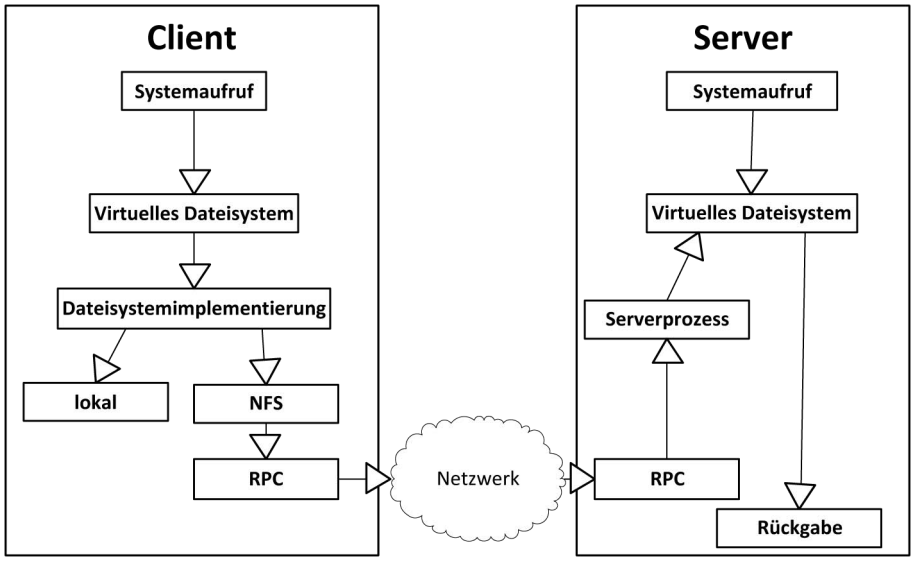

Abbildung 3.9: Schematische Darstellung eines Aufrufs beim NFS in Anlehnung an [5](Seite 122)

- Verwendbarkeit für große Dateimengen
- geringerer Prozessorzeitverbrauch
- bessere Vorhersagbarkeit von Operationen

Nachteile bilden die geringe Wirkreichweite und die Einschränkung auf bestimmte Betriebssystemfamilien (beispielsweise NTFS-Windows, EXT-Linux, etc.).

Objekt- bzw. Datei-basierte Dateisysteme stellen im Vergleich zu block-basierten Systemen eine Entwicklung neueren Datums dar. Die in [27] beschriebenen Eigenschaften eines solchen Systems sind:

- sicherer direkter Zugriff auf gemeinsam genutzte Speichersysteme ohne Vermittlungssysteme (z.B.: Fileserver)
- leichterer Zugriff mittels IP-basierter Übertragungsverfahren (im Vergleich zu z.B. Fibre-Channel)
- verminderter Verwaltungsaufwand durch gemeinsame Aufbewahrung von Meta- und Nutzdaten.

Außerdem sind Objekt-basierte Speichersysteme in der Lage über weitere Entfernungen betrieben zu werden und können von verschiedenen Betriebssystemen angesprochen werden.

Der größte Unterschied von block- und file-basierten Dateisystemen liegt im Abstraktionsniveau. Bei block-basierten Systemen muss für eine Dateisystemoperation in der Dateisystemschicht einiger Aufwand für die richtige Durchführung auf allen Blöcken einer Datei betrieben werden. Die Verwaltung der Block-Datei-Zuordnungstabellen (welche auch korrupt sein können) gehört auch dazu. Im Vergleich dazu bilden objektbasierte Dateisysteme eine Datei als eine Einheit ab, die mittels eindeutiger ID und einem Dateisystemoffset angesprochen wird.

Bereits im großflächigen Betrieb befindliche Beispiele sind das SMB/CIFS-Protokoll von Microsoft oder NFS von Sun. Sie greifen nicht direkt auf Datenträger zu, sondern setzen eine Abstraktionsebene weiter oben an.

Im wissenschaftlichen Bereich beschäftigen sich einige Projekte mit der Weiterentwicklung von file- bzw. objektbasierten verteilten Dateisystemen. Dazu gehört das **Cumulus-Dateisystem** [28], welches als Fokus die sichere Speicherung von Dateien in einer Cloud hat. Als Ansatz wurde von den Erstellern ein Thin-Cloud-Ansatz gewählt, d.h. es wird nur eine dünne Schicht an Kommandos am Interface unterstützt. Dadurch wird die Implementierung unabhängig von Anbieter-spezifischen Optimierungen in den Datacentern der Speicherprovider.

Cumulus versteht sich als Snapshot-basiertes Backupsystem. Als solches speichert es im Falle eines Backups Segmente, welche aus mehreren Dateien bestehen können, bei einem Storageprovider. Es bedient sich einer Write-Once-Strategie, bei der ein bestimmtes Objekt nur als Ganzes zu einem Speicherprovider geladen wird und nicht Abschnitte modifiziert werden. Durch diesen Ansatz werden bei einem Fehler nicht alle Backups korrumpiert, sondern maximal der in der Verarbeitung befindliche Snapshot. Beim nächsten Snapshot werden ähnlich zu Rsync[9] nur veränderte Daten hochgeladen, bei unveränderten Daten werden Verweise auf die Originalversion angelegt.

Ein Cumulus-Client hält die Status-Informationen lokal vor, dazu benutzt er eine SqlLite-Datenbank, welche die Metadaten und andere Information beinhaltet. Diese Informationen dienen zur Beschleunigung von Operationen und zur Unterstützung eines Segment-Reinigungsvorganges, welcher auf den in der Datenbank gespeicherten Altersinformationen einzelner Dateien basiert. Ein Fehlen dieser Daten führt nur zur Neuerstellung der Datenbank aus den online vorliegenden Informationen, nicht zur Operationsunfähigkeit.

Ein weiteres Forschungsprojekt ist die **Posix-Cloud** [29], welches ähnlich dem in Kapitel 6 vorgestellten **TDFS** (Transparent Distributed Filesystem, ein transparentes verteiltes Dateisystem) bereits eine Abstraktionsschicht für verschiedene Storageprovider enthält. Im Gegensatz zu dem in Kapitel 3.2.1 vorgestellten GFS werden die Metadaten nicht zentral auf einem Server vorgehalten, sondern in einer verteilten Datenbank abgelegt. Nutzdaten werden in sogenannten Chunks abgelegt, d.h. Dateien werden in Einzelteile bestimmter Größe zerlegt.

Bei der Durchführung von Dateisystemoperationen werden diese von einem Überwachungsprozess (dem Cloud-Monitor) an Metadaten- und Daten-Agenten (den Data- und Meta-Dispatchern) weitergeleitet, welche auf den einzelnen Storageprovidern die angeforderten Operationen auf den einzelnen Chunks ausführen. Aufgrund der Linux-Basis des Posix-Cloud-Projekts wird als Benutzerschicht ein virtuelles Dateisystem (FUSE[10]) verwendet, welches die einzelnen Dateisystemoperationen nach oben POSIX-konform abbildet.

Hail, „A High-Availability and Integrity Layer for Cloud Storage“ [7], ist fokussiert auf die Abwehr eines Angriffes auf die in einer CloudStorage befindlichen Daten. Es verteilt ähnlich einem Block-basierten Dateisystem Blöcke einer Datei mitsamt zugehöriger Paritätsdaten in den Cloudspeicher. Es ähnelt somit in dieser Eigenschaft konventionellen RAID-Systemen (siehe dazu Kapitel 3.1). Zusätzlich werden spezielle Identifizierungsmerkmale (sogenannte „message authentication codes“ (MACs)

[9] http://www.samba.org/ftp/rsync/rsync.html
[10] http://fuse.sourceforge.net/

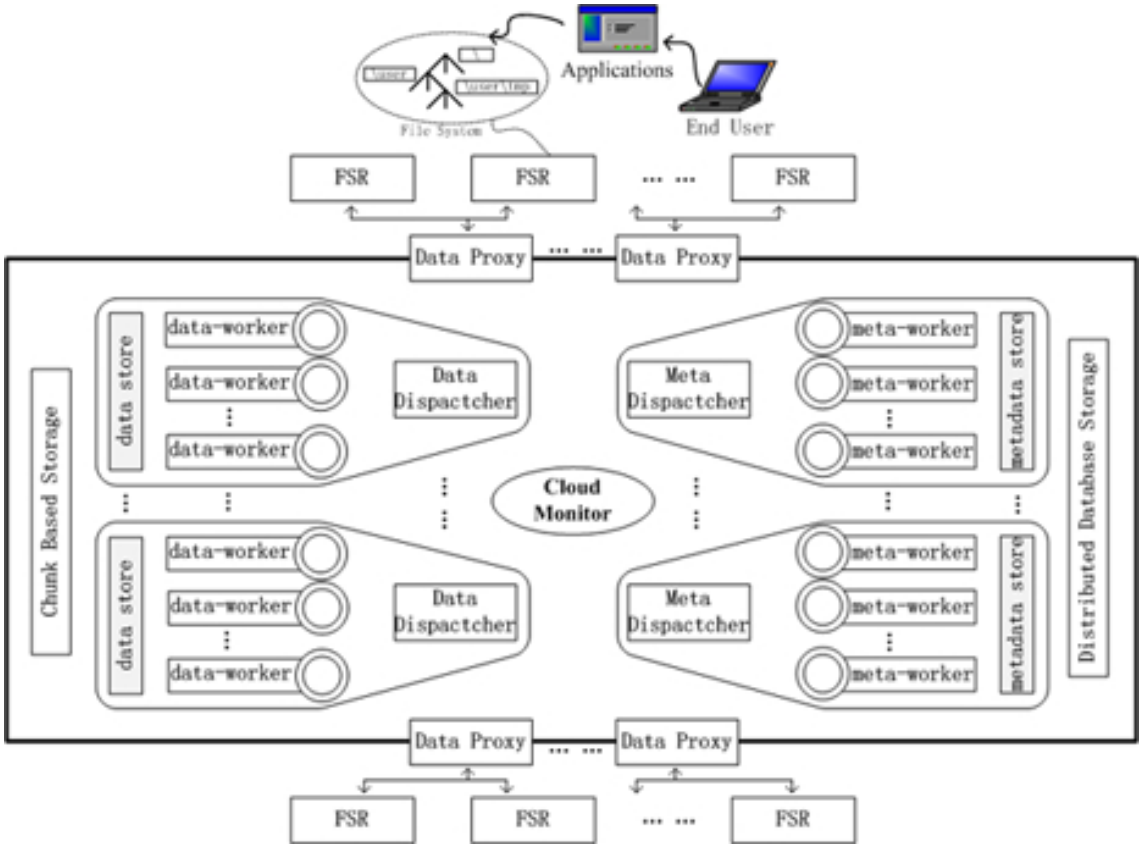

Abbildung 3.10: Architektur der Posix-Cloud aus [29]

in die Daten eingebunden. Diese Codes werden bei Abrufen decodiert und verifiziert. Bei der Entdeckung eins Fehlers wird der fehlerhafte Block durch eine korrekte Kopie auf einem anderen Server in der Cloudstorage wiederhergestellt. Die En- und Dekodierung basiert auf der Verwendung der auch bei RAID genutzten Reed-Solomon-Codes.

Das **RFS** [2] ist ein Clouddateisystem, welches sich auf mobile Geräte spezialisiert. Als Designziele des Projekts wurden Datensicherheit und Abstraktion von Cloud-Storage-Anbietern und bestmögliche Ausnutzung der Bandbreitenressourcen gewählt. Dies wird durch ein Client-Server-Model erreicht, bei dem der RFS-Server die Interaktion mit der Cloud übernimmt, und der Client sich von ihm die Daten abholt bzw. sie einschickt. Zur Verminderung von Interaktionen mit dem Server wird ein lokaler Cache vorgehalten. RFS sieht keinen Schutz gegen gleichzeitigen Zugriff mehrerer Parteien auf dieselbe Datei vor.

Als Methode zur Gewährleistung der Sicherheit der Daten vor unbefugtem Zugriff wird Verschlüsselung verwendet. Allerdings werden nur Nutzdaten verschlüsselt, die Metadaten bleiben unverschlüsselt. Die Integration mit der Datenverwaltung des Benutzers geschieht als Linux-Kernel-Modul (ohne virtuelle Fuse-Zwischenschicht) und ist somit für den Benutzer transparent. In definierbaren Intervallen wird eine Synchronisation mit der Cloud initiiert.

Außerhalb des wissenschaftlichen Diskurses existieren Projekte, die ähnliche Ziele verfolgen. Ein prominentes Beispiel ist **eprint**[11], eine Software der Open-Access-Bewegung. Diese tritt für einen offenen Zugang zu wissenschaftlichen Materiali-

[11] http://www.eprints.org/

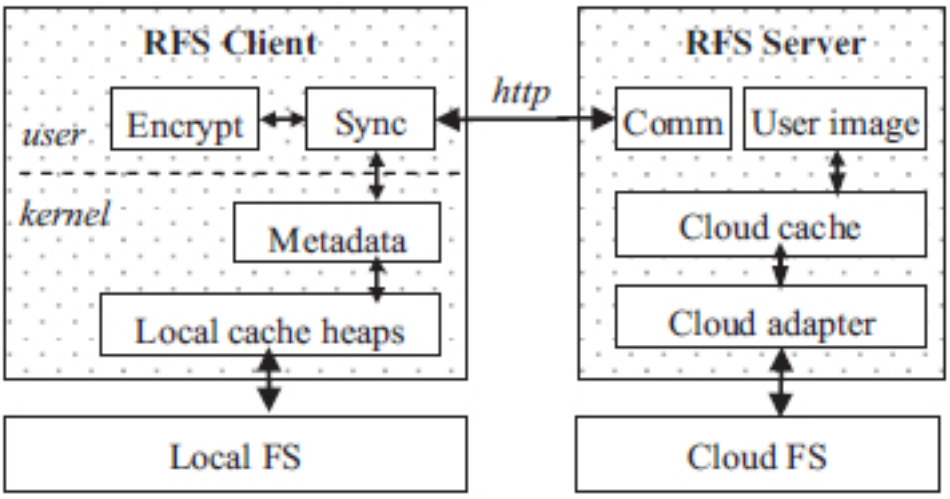

Abbildung 3.11: RFS Architektur aus [2]

en ein. Als Herausgeber wissenschaftlicher Artikel ist sie auf eine leistungsfähige Repository-Software angewiesen. In der neuesten Version 3.2 wurde ein Storage-Abstraktionslayer eingebaut, welcher es ermöglicht Publikationen und ihnen zugehörige Materialien auch auf einer Cloud-Storage zu speichern.

Ein weiteres Projekt ist **Akubra Project**[12], welches Cloud-Speicher abbildet. Es soll nach seiner Fertigstellung für das Fedora-Projekt als Basis für die Speicherung der Fedora-Daten dienen.

Projekt	Verschlüsselung	Multi-Provider-Backup	Ansatz
Cumulus	ja	nein	Service
PosixCloud	nein	nein	Dateisystem
Hail	ja	nein	Service
RFS	ja	nein	Dateisystem
eprint	nein	ja	Web-Repository
lokad	nein	nein	Service
sharpbox	nein	nein	Service
TDFS	ja	ja	Service/Dateisystem

Tabelle 3.2: Vergleich Related Work

Lokad[13] ist ein Objekt-Cloud-Mapper in C#, der Features wie Speicher-, Queue- und Tabellen-I/O beinhaltet. Der Speicher-I/O bietet Speicher-Operationen zum Speicher-Provider. Beim Queue-I/O können mittels Implementierung eines speziellen Interfaces Rechenoperationen in der Cloud ausgeführt werden (vgl. MapReduce von Google [30]). Tabellen-I/O versteht sich als Ado.Net-gestützter Zugriff auf Windows Azure, den Basis-Cloud-Provider des Lokad-Projekts, und bietet die Möglichkeit einfacher Abfragen bezüglich der Struktur der in der Cloud gespeicherten Daten. Leider ist dieses System nicht auf andere Provider übertragbar. Eine kurze Untersuchung des Codes der Implementation offenbart einen exzellent geschriebenen Code, der auf die exakte Einhaltung von Patterns und architekturellen Vorgaben

[12] https://wiki.duraspace.org/display/AKUBRA/Akubra+Project
[13] http://code.google.com/p/lokad-cloud/

Wert legt.

Sharpbox ist ein auf .NET basierter Abstraktionslayer für den Cloudstorage-Provider Dropbox. Er zeichnet sich durch relativ einfachen Aufbau und die Existenz eines Synchronisationsframeworks aus. Storage-Plugins ermöglichen den Zugriff auf unterschiedliche Provider. Als Basis der Kommunikation mit den Providern wird REST verwendet. Der einfache Aufbau und die einfache Erweiterbarkeit bei gleichzeitiger Fokussierung auf das Wesentliche geben den Ausschlag für die Wahl dieses Projektes als Basis für das in dieser Arbeit vorgestellte sichere, verteilte Dateisystem TDFS.

Allen hier vorgestellten Ansätzen ist gemein, dass sie zwar teilweise Ansätze zur Loslösung von Zugriffsspezifika einzelner Provider anbieten, allerdings ist die meistens verwendete Abstraktionsschicht auf eine Entweder-Oder-Methodik ausgerichtet. Bei Verwendung dieser Lösungen wird immer nur einen einzelner Provider angesprochen, das Lock-In-Problem ist somit nicht vollständig gelöst. TDFS wird zeigen, dass ein sicheres, verteiltes Dateisystem auch gleichzeitig mit mehreren CloudStorage-Providern operieren kann, und dass Datensicherheit mittels eines RAID-artigen Mechanismus sichergestellt werden kann.

Kapitel 4

Evaluation von Storage Providern

Der Markt für Cloud-basierte Dienstleistungen ist jung und daher ist die Anzahl der um die Vorherrschaft kämpfenden Unternehmen im Ansteigen begriffen. Die Leistungen und Preise der einzelnen Angebote unterscheiden sich mitunter drastisch. Wie in Kapitel 2.4 beschrieben, gibt es eine Anzahl von Risken und Metriken, anhand derer eine qualitative Einschätzung des Angebotes der Anbieter erfolgen kann. Dieses Kapitel widmet sich der Evaluation der zur Erstellung der Magisterarbeit untersuchten Anbieter und stellt deren Ergebnisse dar.

4.1 Amazon S3

Amazon, ein großes Fernhandelsunternehmen, stellt mittlerweile auch eine Anzahl von Web-Diensten zur Verfügung. Diese gehen vom Grid-Computing (Elastic MapReduce) über Datenbanken (SimpleDB, ElastiCache), Zahlungslösungen (DevPay) bis zu Inhalts-Anbieter-Lösungen wie Amazon Cloudfront. Eine vollständige Liste ist der Amazon-WebServices-Seite[14] zu entnehmen. Die im Zusammenhang dieser Magisterarbeit hervorzuhebenden Dienste sind Amazon Elastic Block Storage (EBS), Amazon CloudWatch und Amazon Simple Storage Service (auch: **Amazon S3**).

Bei Amazon Elastic Block Storage handelt es sich um einen Service zur Bereitstellung von block-basiertem Speicher in Verbindung mit Amazon Elastic Compute Cloud (auch: Amazon EC2). Während also EC2 den in Kapitel 2.2 PaaS entspricht, ist die EBS ein zugehöriger Speicherdienst (und somit zwischen IaaS und SaaS einzustufen). Seine Dienste sind über spezielle Web-Schnittstellen abzurufen.

Im Unterschied dazu bietet Amazon S3 eine objekt-basierte Speicherlösung. Mit ihrer Hilfe ist es möglich über das HTTP-Protokoll Daten in den Amazon-Rechenzentren abzuspeichern. Es gibt (wie bei den anderen Speicherdiensten Amazons) die Möglichkeit zur Festlegung einer bestimmten geographischen Zone als Aufbewahrungsort für die Daten, nach der dann ein dort befindliches Rechenzentrum automatisch ausgewählt wird.

Laut [31] besteht der Amazon-S3-Service aus einer zweischichtigen Architektur. Die oberste Ebene bilden die sogenannten „Buckets“, eine Art Haupt-Ordner oder Laufwerk. In diesen befinden sich eine beliebige Anzahl von Dateien und Ordnern, wobei eine einzelne Datei nicht größer als 5 Terabyte sein kann. Diese Größe ist im

[14] http://aws.amazon.com/

Datenmenge	Standardspeicherung	Reduzierte Redundanz
1 TB / month	$0.140 pro GB	$0.093 pro GB
nächste 450 TB / Monat	$0.110 pro GB	$0.073 pro GB
nächste 500 TB / Monat	$0.095 pro GB	$0.063 pro GB
nächste 4000 TB / Monat	$0.080 pro GB	$0.053 pro GB
mehr als 5000 TB / Monat	$0.055 pro GB	$0.037 pro GB

Tabelle 4.1: Preisübersicht Amazon S3

Anfragen:	
PUT, COPY, POST,LIST	$0.01 pro 1,000 Anfragen
GET, andere Anfragen	$0.01 pro 10,000 Anfragen
DELETE	0

Tabelle 4.2: Preisübersicht Amazon S3 - Anfragen

Zusammenhang mit derzeit verfügbaren Übertragungstechnologien (mit Übertragungstechnologien am Internet-Backbone, wie zum Beispiel OC-192 mit 10 Gbps[15] und im Verhältnis drastisch geringeren Übertragungsraten für End-Benutzer) ausreichend dimensioniert. Die Buckets dienen in weiterer Folge auch als Messpunkt für die Metriken, die Amazon im Zusammenhang mit S3 anbietet.

S3 bietet eine Möglichkeit der unentgeltlichen Benutzung an, den „Amazon S3 Free Usage Tier“. Dieser hat folgende Eigenschaften:

- 5 GB Speicherplatz
- 20,000 „get“ Anfragen pro Monat
- 2,000 „put“ Anfragen pro Monat
- 15 GB ausgehender Datentransfer pro Monat

Innerhalb dieser Grenzen fallen für den Dienst keine Kosten an. Allerdings muss bei der Anmeldung zu Amazon S3 eine Zahlungsart (zum Beispiel Kreditkarte) angegeben werden. Werden die eben genannten Grenzen überschritten, so werden automatisch die darüber hinausgehenden Transfers in Rechnung gestellt. Derzeit sind folgende Preise wirksam (es gibt zwar minimale Unterschiede zwischen den oben genannten Zonen, aber deren Unterschiede sind vernachlässigbar[16]):

Sicherheitstechnisch ist bei Amazon S3 nur die Anmeldung über Access Key ID und Secret Access Key (dies entspricht Benutzername und Passwort) möglich, andere Dienste von Amazon bieten auch die Möglichkeit zur Authentifizierung über X.509 und Schlüsselpaare an, diese sind aber für Amazon S3 derzeit nicht freigeschaltet. Als Übertragungsmethoden stehen SOAP, REST und BitTorrent zur Verfügung. Verschlüsselung wird als Transport-Verschlüsselung (HTTPS) angeboten, eine interne Verschlüsselung der Daten besteht nicht (wobei über das SDK in Java eine clientseitige Verschlüsselung angeboten wird). Für den Transfer großer Datenmengen steht außerdem ein Import/Export-Service (physikalischer Art) zur Verfügung.

[15] http://searchnetworking.techtarget.com/definition/Optical-Carrier-levels-OCx
[16] http://aws.amazon.com/s3/

Datentransfer	
eingehend	0
ausgehend	
1 GB / month	$0.000 pro GB
nächste 10 TB / month	$0.120 pro GB
nächste 40 TB / month	$0.090 pro GB
nächste 100 TB / month	$0.070 pro GB
nächste 350 TB / month	$0.050 pro GB
ab 524 TB / month	auf Anfrage

Tabelle 4.3: Preisübersicht Amazon S3 - Datentransfer

Für die Regelung des Zugriffes auf einzelne Objekte stellt Amazon ACLs (Access-Control-Lists) zur Verfügung, diese entsprechen in etwa den Berechtigungen in herkömmlichen Dateisystemen. Amazon S3 und die anderen Web-Dienste Amazons waren im April und August 2011 von Ausfällen betroffen[17,18], bei letzterem kam es sogar zum unwiederbringlichen Verlust von Daten. Ansonsten sind keine gravierenden Störungen bekannt. Über die Leistungsfähigkeit von Amazon S3 und anderen Anbietern wird in Kapitel 8 Auskunft gegeben.

Für den Support stehen Diskussionsforen, aber auch ein kostenpflichtiger Premium-Support-Vertrag zur Verfügung. In der teuersten Variante stellt dieser direkten Kontakt zu einem Techniker von Amazon zur Verfügung[19].

4.2 Google Storage

Bei Google Storage handelt es sich um eine vom Suchmaschinenbetreiber Google betriebene Speicherdienstleistung. Diese ist im Gegensatz zu Amazon S3 nicht für die direkte Benutzung durch Endkunden, sondern für die Integration in Firmenapplikationen gedacht (der volle Name des Dienstes lautet „Google Storage for Developers“). Google bietet wie Amazon eine Reihe weiterer Dienstleistungen an. Diese stehen zwar prinzipiell mit Google Storage in Verbindung (Google stellt pro angemeldetem Konto eine Menge an Speicherplatz zur Verfügung, die sich prinzipiell alle verwendeten Dienste teilen müssen), haben aber keinen direkten Einfluss auf Google Storage.

Google Storage stellt ebenfalls Buckets bereit, die Dateien mit einer Maximalgröße von 1 Terabyte aufnehmen können. Es ist interoperabel mit Amazon S3, kann also mit den gleichen Client-Applikationen oder -bibliotheken betrieben werden, die auch für Amazon S3 verwendet werden (und ähnelt dem Dienst daher auch sehr stark). Google Storage unterstützt weiters auch OAuth als Authentifikationsmechanismus und feiner granulierbare ACLs zum Steuern des Zugriffs auf die einzlenen Objekte. Google Storage verschlüsselt Daten nicht selbst.

Als Zugriffsmechanismus außerhalb Amazon S3-kompatibler Bibliotheken ist REST vorgesehen.

Der Free-Usage-Tier von Google Storage bietet folgende Limits:

[17] http://aws.amazon.com/message/65648/
[18] http://aws.amazon.com/de/message/2329B7/
[19] http://aws.amazon.com/premiumsupport/

Speicherplatz	$0.17/GB/Monat
PUT, POST, LIST Anfragen	$0.01 per 1,000 Anfragen
GET, HEAD Anfragen	$0.01 per 10,000 Anfragen
Upload-Volumen	$0.10/GB
Download-Volumen	$0.15/GB Amerika, Europa, Mittlerer Osten, Afrika
	$0.30/GB Asien-Pazifik

Tabelle 4.4: Preisübersicht Google Storage

Typ	Speicherplatz	Preis $ /Monat
Basic	2	0
Pro 50	50	9,99
Pro 100	100	19,99
Teams	350+	66,25
zus. 100 GB		16,6

Tabelle 4.5: Preisübersicht Dropbox

- 5 GB Speicherplatz
- 25 GB Download-Volumen
- 25 GB Upload-Volumen
- 25,000 GET, HEAD Anfragen
- 2,500 PUT, POST, LIST* Anfragen

Darüber hinaus verrechnet Google folgende Raten für verbrauchte Ressourcen:

Obwohl es Support-Verträge für Googles Applikationdienste gibt, besteht eine solche Möglichkeit nicht für den Google-Storage-Dienst. Als einzige Möglichkeit bietet Google diesbezüglich die Benützung eines Support-Forums an[20].

Ausfälle größeren Ausmaßes hat es seit dem Start von Google Storage Mitte 2011 nicht gegeben, allerdings haben die Cloud-Dienste Googles (allen voran GMail) bereits eine Historie von Dienstunterbrechungen in den Jahren 2008 und 2009[21].

4.3 Dropbox

Dropbox ist ein auf Amazon S3 basierender, aber eigenständiger Speicherdienst. Dropbox stellt einen eigenen Client für den Zugriff auf die in der Cloud befindlichen Daten zur Verfügung, auch ist die Verbindung über die angebotene REST-API möglich[22]. Auch Dropbox bietet die Möglichkeit zur Authentifikation via OAuth an. Zur Moderierung von Zugriffsrechten stehen Gruppenberechtigungen zur Verfügung.

Preislich betreibt Dropbox ein relativ einfaches Modell:

Teams sind hierbei auf 5 Benutzer eingegrenzt, zusätzliche Benutzer müssen für 10,40$/Monat zugekauft werden. Dropbox bietet sowohl zur sicheren Verwahrung

[20] http://code.google.com/apis/storage/forum.html
[21] http://www.pcworld.com/businesscenter/article/172614/google_outages_damage_cloud_credibility.html
[22] https://www.dropbox.com/developers/docs

Typ	Speicherplatz (GB)	maximale Filegröße	Preis/Monat
Personal	25	1 GB	9,99
	50	1 GB	19,99
Bussiness	500	2 GB	9,99
Enterprise	Unlimitiert auf Anfrage	2GB	Auf Anfrage

Tabelle 4.6: Preisübersicht Box.Net

der Daten sowohl die übliche Transport-Verschlüsselung als auch eine auf AES-256 basierende Content-Verschlüsselung an. Diese Verschlüsselung führt in diesem Zusammenhang allerdings nicht zu einer erhöhten Sicherheit der Daten. Der Grund dafür ist, dass der Schlüssel mit dem die Daten verschlüsselt werden von Dropbox zur Verfügung gestellt wird und unveränderbar ist. Das bedeutet, dass die Mitarbeiter der Firma Dropbox Zugriff auf die einzelnen Dateien des Kunden haben. Auch kann nicht verifiziert werden, ob die Daten serverseitig überhaupt verschlüsselt sind, da weder am Web-Interface noch über die angebotene Client-Software ein Zugriff auf die Rohdaten möglich ist, sondern transparent die entschlüsselten Daten angeliefert werden. Für Uploads, die nicht mittels der DropBox-Software durchgeführt werden (sondern über die DropBox-API), gilt eine Maximal-Größe von 300MB pro Datei.

Premium-Support bietet Dropbox in seinem Teams-Paket an, für andere Anfragen existiert ein Support-Forum. Nachdem DropBox auf Amazon-S3 als Speichermedium aufbaut, sind die Ausfälle mit denen von S3 eng gekoppelt. Darüber hinaus sind nur wenige Dienstunterbrechungen bekannt[23].

4.4 Box.Net

Box.Net ist ein weiteres Unternehmen im Cloud-Storage-Umfeld. Laut eigenen Aussagen[24] fokussiert es sich stark auf die Sicherheit seines Angebots, allerdings gilt diese Aussage nur für die Daten, die im Rahmen des Enterprise-Angebots transferiert werden. Das Enterprise-Angebot ist eine von 3 Speicherdienstleistungen, die Box.Net zur Verfügung stellt. Wie bei anderen Anbietern existiert auch bei Box.Net ein Free Usage Tier zu folgenden Konditionen:

- 5 GB Speicherplatz
- 25 MB maximale Filegröße
- 1 Benutzer

Bei Überschreiten dieses Limits gelten folgende Konditionen:

Außerhalb des Enterprise-Angebotes wird nur SSL für den Transport angeboten, eine Verschlüsselung der Daten beim Anbieter findet nicht statt. Auch alle weiteren Angebote wie erweiterter Support oder Berechtigungsmanagement mit Gruppen sind auf das Enterprise-Angebot beschränkt. Von Seiten der API kann auf Box.Net mittels REST, SOAP oder XML zugegriffen werden. Als Authentifikationsmechanismus bedient sich Box.Net eines modifizierten OAuth-Mechanismus der zum Standard nicht

[23] http://forums.dropbox.com/tags.php?tag=outage
[24] http://www.box.net/features/security/

	1 Node (pro GB und Monat)	2 Nodes	3 Nodes
Speicherplatz	$0.25	$0.48	$0.71
Uploads	$0.10	$0.20	$0.30
Downloads	$0.15	$0.15	$0.15
Uptime (%)	99.9	99.99	99.999

Tabelle 4.7: Preisübersicht Nirvanix

vollständig kompatibel ist[25]. Dieser beinhaltet die Umleitung des Benutzers auf eine Login-Seite. Nur in Spezialfällen wird ein direkter Zugriff gewährt. Über Ausfälle des Betriebes ist nichts Belegbares bekannt.

4.5 Nirvanix

Nirvanix ist einer der ältesten Cloud-Storage-Anbieter. Als solcher bietet er eine detaillierte Beschreibung des Zugriffs auf seine Dienste mittels der Nirvanix-API an[26]. Die Anmeldung an den Service erfolgt klassisch über Benutzername und Passwort. Nach erfolgter Anmeldung können mit dem erhaltenen Autorisierungs-Ticket Anfragen an den Dienst gestellt werden. Eine spezielle Möglichkeit dieses Dienstes ist es, unter dem Master-Account (einer Art Administrator-Konto für eine bestimmte Menge Speicherplatz) weitere Unter-Accounts zur Laufzeit eines Programmes über die API anlegen zu können. Diese Unter-Accounts können zur Segmentierung des Speicherplatzes und zur Zugriffskontrolle genützt werden.

Die maximal mögliche Größe für den Upload einer Datei beträgt 2 GB. Persönlichen Support gibt es wie bei anderen Dienstleistern nur in der Enterprise-Version des Dienstes, zu dem preislich keine näheren Informationen vorliegen. In der „Self-Service“-Art des Dienstes gelten folgende Bedingungen[27]:

Als Node gilt hier ein einzelnes Rechenzentrum, dementsprechend kostet die verteilte Lagerung mehrer Replikas mehr als die Standard-Redundanz.

Daten können bei der Übertragung verschlüsselt werden, eine transparente Verschlüsselung beim Upload zur verschlüsselten Speicherung im Cloud-Speicher findet ohne spezielle Client-Software allerdings nicht statt.

4.6 Windows Live Skydrive / Windows Azure

Skydrive ist ein von Microsoft angebotener Dienst für Speicher in der Cloud. API im offiziellen Sinne gibt es keine, daher gibt es einige Projekte wie zum Beispiel SkydriveApiClient [28], welche Zugriff auf diesen Dienst ausserhalb von Microsofts eigenem Web-Client ermöglichen. Da diese Art des Zugriffes allerdings auf der Verarbeitung von Webseiten passiert und nicht auf einer fix definierten API, unterbrechen Veränderungen an der Webseite des Dienstes sehr leicht die Konnektivität einer solchen

[25] http://developers.box.net/w/page/12923915/ApiAuthentication
[26] http://developer.nirvanix.com/sitefiles/1000/API.html
[27] http://www.nirvanix.com/how-to-buy/self-service-pricing.aspx
[28] http://skydriveapiclient.codeplex.com/

Speicherplatz	$0.15 per GB
	$0.01 pro 10,000 Anfragen
Datentransfer	$0.20 pro ausgehendem GB

Tabelle 4.8: Preisübersicht Azure

Client-Software. Weiters ist die Verbindung mittels WebDAV möglich. Verschlüsselung ist Server-seitig nicht vorgesehen.

Der Dienst ist für seine Benutzer kostenlos. Der verfügbare Speicherplatz beträgt 25GB[29] und die maximale Dateigröße 100MB[30].

Als kostenpflichter Dienst steht Windows Azure dem kostenlosen Windows Live Skydrive gegenüber. Auch bei diesem ist keine transparente Verschlüsselung vorgesehen. Windows Azure verfolgt Preispolitik welche auf dem Durchschnittswert der Benutzung des Speicherdienstes in einem Monat basiert (mit der täglichen Benutzung als Maßeinheit):

Der Dienst ist mittels von Microsoft bereitgestellter Bibliothek oder auch über REST zugreifbar. Authentifizierung ist mittels Username und Passwort oder OAuth-Wrap (einer Vorversion des derzeit gültigen OAuth Version 2) möglich.

Ausfälle sind für Azure und Skydrive jeweils einmalig für 2009[31] und 2011[32] bekannt.

Die von den Herstellern für ihre Dienste verlangten Preise ergeben ein heterogenes Bild. Mit dem Eintritt weiterer Mitbewerber ist eine Preisminderung zu beobachten. Diese wird durch den steigenden Bedarf an Speicherplatz der Benutzer kompensiert. Während bei geringen Datenmengen das Teilnehmerfeld eng beieinander liegt, ist Amazon bei Datenmengen ab 1 Terabyte mit billigster Anbieter. Der in Abbildung 4.1 gezeigte Preisverlauf für den Anbieter Box.Net ist nur bedingt aussagekräftig. Bis zu einem Platzbrauch von 500 GB gilt bei Box.Net ein mit 15$ pro Monat pauschalierter „Business“-Tarif. Darüber hinaus kommt der „Enterprise“-Tarif zum Einsatz, über welchen keine genaue Informationen vorliegen.

Eine Gefahr für die eigenen Daten besteht nicht nur durch die durchschnittlich 1-2x pro Jahr auftretenden Service-Ausfälle, sondern auch durch Verlust der Daten bei Schließung eines kleineren Speicheranbieters (wie zum Beispiel dem Anbieter „Iron Montain“). Die meisten Anbieter geben den Kunden vor Beendigung des Dienstes allerdings die Möglichkeit zur Auslagerung der Daten.

Bemerkenswert ist auch der unterschiedliche Umgang mit der Verschlüsselung von Daten. Es gibt Angebote mit keiner server-seitigen Verschlüsselung, ein Angebot mit einem fest definierten unbekannten Schlüssel (Dropbox) und Anbieter mit serverseitiger Verschlüsselung. Bis zur Einführung von sicherer Verschlüsselung durch alle Provider ist es daher unerlässlich, Client-seitig für die Verschlüsselung zu sorgen.

Über die hier kurz beschriebenen Anbieter hinaus existieren eine Reihe weiterer Dienstleister, welche allerdings in der Regel entweder mangelhaft dokumentiert sind oder gewisse Funktionalitäten nicht enthalten, welche zur Verwendung in einer Spei-

[29] http://explore.live.com/office-web-apps-control-permissions

[30] http://windowsteamblog.com/windows_live/b/windowslive/archive/2011/06/20/introducing-skydrive-for-the-modern-web-built-using-html5.aspx

[31] http://social.msdn.microsoft.com/Forums/en-US/windowsazure/thread/6c1cd8a2-8d9d-43e9-a1d8-928e0ca4de78/

[32] http://windowslivehelp.com/thread.aspx?threadid=844565bd-8ad5-497b-bc32-9fceab52a86d

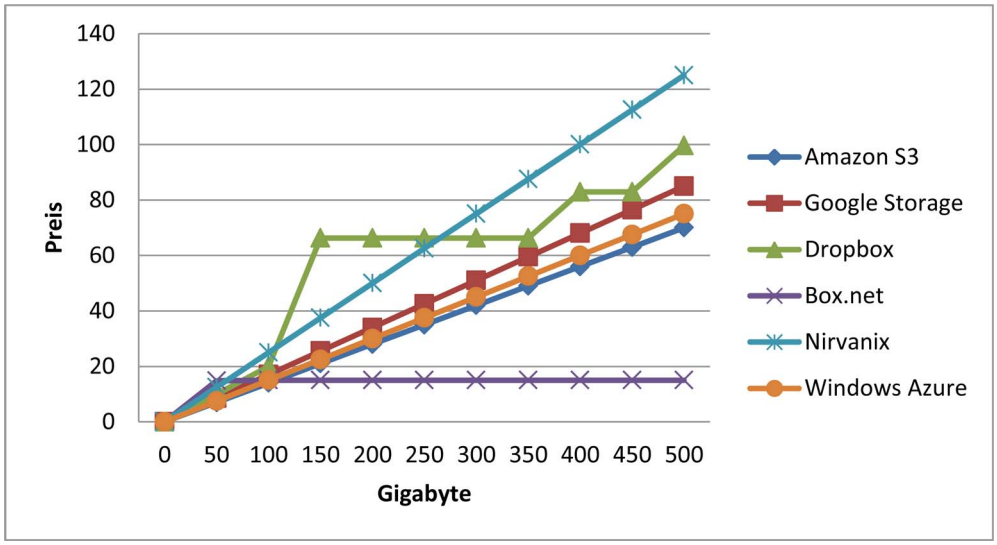

Abbildung 4.1: Preisvergleich der Anbieter

cherlösung notwendig sind. Eine Auflistung einiger Beispiele samt ihrer jeweiligen Fähigkeiten ist im Appendix auf Seite 88 zu finden.

Kapitel 5

Requirements für ein sicheres, verteiltes Dateisystem

Für die Implementierung der im Kapitel 3 über die Arten der Datenspeicherung und im Kapitel über die Evaluation von Storage-Provider (Kapitel 4) untersuchten Konzepte ist es notwendig, eine Reihe von Anforderungen zu definieren, welche die Implementierung bewertbar machen. Da in den vorangegangenen Kapiteln bereits über die Grundlagen der verwendeten Technologien berichtet wurde, wird in diesem Kapitel nicht gesondert auf deren Einzelheiten eingegangen.

Bezugnehmend auf [32] besteht ein Software-Requirements-Specification-Dokument (Abkürzung: SRS) aus folgenden Themenbereichen:

- Interfaces
- Functional Capabilities
- Performance Levels
- Data Structures/Elements
- Safety
- Reliability
- Security/Privacy
- Quality
- Constraints and Limitations

Diese Themengebiete werden in diesem Kapitel mittels des in [32] vorgestellten Aufbaus abgehandelt. Nur die relevanten Themen werden behandelt und die für diese Aufgabenstellung nicht wichtigen Themata außen vor gelassen (sogenanntes „Tailoring“).

5.1 Begriffsklärung

Computer-Programme, die von einem Installationsmanager (z.B.: Windows-/Microsoft-Installer) als eine zusammengehörige Einheit gesehen werden, werden im Fachjargon „Computer-Software-Configuration-Item“ genannt [33]. Dieser Begriff trifft auf die hier zu programmierende Software zu, diese wird also im weiteren Verlauf mit ihrer Abkürzung „CSCI“ bezeichnet. Gemäß dem IEEE Recommended Practice for Software Requirements Specifications [34] wird auch die Wichtigkeit der einzelnen Requirements bewertet.

Zur Übersichtlichkeit und Referenzierbarkeit werden Requirements-Nummern in der Form Rxx vergeben, wobei „xx“ für die Nummer des Requirements steht.

Generell soll das CSCI über eine definierte Schnittstelle Zugriff auf Cloud-basierte Dateisysteme bereitstellen. Zu diesem Zweck soll auf existierende Schnittstellen zu Cloud-Service-Anbietern zurückgegriffen werden. Der Zugriff soll transparent geschehen, d.h. für den Benutzer möglichst wenig Eingriff (abgesehen von der Konfigurationsphase) erfordern.

5.2 Funktionale Requirements

R1 Das CSCI soll Dateien bzw. Ordner mit oder ohne Dateien auf verschiedene Cloud-Storage-Anbieter verteilen können. Wichtigkeit: Hoch

R2 Das CSCI soll diese Dateien bzw. Ordner transparent gleichzeitig auf mehrere Anbieter verteilen können. Das bedeutet dass die Software über eine Möglichkeit zum Splitten und Mergen einzelner Dateibestandteile verfügen soll. Wichtigkeit: Hoch

R3 Das CSCI soll unterschiedliche und/oder bei verschiedenen Storage-Providern lagernde Daten in einer einheitlichen Ansicht zur Verfügung stellen. Wichtigkeit: Hoch

R4 Das CSCI soll eine Möglichkeit zur redundanten Aufbewahrung der Dateien bieten, d.h. es soll möglich sein eine Datei/einen Ordner bei mehreren Providern gleichzeitig zu lagern. Wichtigkeit: Hoch

R5 Das CSCI soll zumindest einen direkten Zugriff auf die online gelagerten Daten bieten. Es soll mittels Plugin-System eine Möglichkeit geschaffen werden für eine Erweiterung auf ein System, das lokale Disk und die Storage-Provider synchronisiert. Wichtigkeit: Hoch

R6 Das CSCI soll zumindest folgende Dateisystemoperationen unterstützen (Wichtigkeit: Hoch):

- Lesen von Dateien / Ordnern
- Schreiben von Dateien / Ordnern
- Auflistung von Dateien / Ordnern
- Löschen von Dateien / Ordnern
- Kopieren von Dateien / Ordnern

- Umbenennen von Dateien / Ordnern

Details zu diesen Operationen sind im Kapitel 9 nachzulesen.

R7 Zur Darstellung der Daten soll beispielhaft eine Zugriffsmöglichkeit geschaffen werden, vorzugsweise über ein Netzlaufwerk oder ähnliches. Wichtigkeit: Mittel

5.3 Stati and Modi

Das CSCI soll 2 Stati unterstützen:

R8 Einen Operationsmodus, in dem alle Funktionen verfügbar sind. Wichtigkeit: Hoch

R9 Einen Recovery/Synchronisationsmodus, in dem eingeschränkte Funktionalität bereitstehen kann, aber versucht wird den Operationsmodus wiederherzustellen. Wichtigkeit: Hoch

5.4 Interface Requirements

R10 Das CSCI soll die verschiedenen Interfaces der einzelnen Cloud-Storage-Anbieter nach außen zu einem einheitlichen Interface abstrahieren. Das Interface soll einem Client bidirektionalen Zugriff auf die dahinter liegenden Daten ermöglichen. Wichtigkeit: Hoch

R11 Das CSCI soll intern die vorhin genannten Interfaces abstrahieren, um verschiedenen Business-Logik-Modulen einen einheitlichen Ansatzpunkt zu liefern. Wichtigkeit: Hoch

R12 Zur Abstraktion der Storage-Provider sollen soweit möglich vor allem etablierte Technologien wie Representational State Transfer (REST), Simple Object Access Protocol (SOAP) oder das Hypertext Transfer Protocol (HTTP) eingesetzt werden. Diese Programmiertechniken ermöglichen Interoperabilität und sind somit anderen vorzuziehen. Wichtigkeit: Hoch

5.5 Sicherheits- und Privatsphären-Requirements

R13 Das CSCI soll die mit ihr verwalteten Daten vor unautorisiertem Zugriff schützen. Wichtigkeit: Hoch

R14 Zu diesem Zweck soll die Software gängige Authentifikationsmechanismen unterstützen, soweit dies von den eingesetzten Providern unterstützt wird. Wichtigkeit: Mittel

R15 Daten, die bei einem Storage-Provider abgespeichert werden sollen nur vom Besitzer des Files einsehbar sein. Wichtigkeit: Hoch

R16 Zur Verhinderung von Schreibzugriffsproblemen soll ein Mechanismus integriert werden, der nur einmaligen Zugriff im Schreibmodus auf ein Dateisystemobjekt erlaubt. Wichtigkeit: Mittel

R17 Außerdem soll die Software transparente Verschlüsselung der Daten ermöglichen, um möglichen Sicherheitslecks der Cloud-Storage-Provider vorzubeugen. Wichtigkeit: Mittel

5.6 Softwarequalitäts-Faktoren

R18 Das CSCI soll modular aufgebaut sein, d.h. es soll möglich sein durch Einhängen weiterer Module die Unterstützung für weitere Cloud-Storage-Anbieter ohne übermäßigen Aufwand bzw. Neudesign der Software zu erlangen. Dies soll auch für eventuelle Funktionserweiterungen (Beispiel: transparente Verschlüsselung) gelten. Wichtigkeit: Hoch

R19 Das CSCI soll reproduzierbare Ergebnisse liefern, d.h. bei vergleichbarem Ausgangszustand und Umgebungsbedingungen soll der Endzustand ident sein. Wichtigkeit: Hoch

R20 Das CSCI soll Sicherheit und Robustheit über Geschwindigkeit stellen. D.h. es soll versucht werden, die Datensicherheit unter allen Umständen zu gewährleisten. Wichtigkeit: Mittel

R21 Die Software soll in einer objektorientierten Programmiersprache geschrieben werden, vorzugsweise aus der .NET Sprachfamilie. Wichtigkeit: Mittel

R22 Das CSCI soll die zum Betrieb notwendigen Daten innerhalb des eigenen Wirkungsbereiches (Storage-Provider und CSCI selbst) verwalten, d.h. es soll keine externe Datenbank zum Betrieb notwendig sein. Wichtigkeit: Hoch

5.7 Benutzungs-Requirements

R23 Das CSCI soll ohne weitere spezielle Schulungen durch Lesen der beigelegten Dokumente (Release Notes oder ähnliches) installierbar, konfigurierbar und betreibbar sein. Wichtigkeit: Gering

R24 Das CSCI soll in der Prototypenphase durch technisch versierte Benutzer bedienbar sein. Das CSCI soll derart aufgebaut sein dass Fehlbedienung nicht zu Datenverlust führen kann (mit Ausnahme einer vom Benutzer induzierten Löschoperation). In der Endphase soll es durch Normalbenutzer verwendbar sein. Wichtigkeit: Mittel

R25 Das CSCI soll einen Installer bereitstellen, welcher die zum Betrieb der Software benötigten Komponenten beinhaltet. Eventuell zum Betrieb notwendige Komponenten von Drittherstellern sind davon nicht betroffen und separat zu installieren. Wichtigkeit: Gering

R26 Die zur Installation notwendigen Schritte und externen CSCIs sollen in Release Notes aufgeführt werden. Wichtigkeit: Hoch

R27 Das CSCI soll die zum Verbinden notwendigen Daten über Konfigurationsdateien bereitgestellt bekommen und diese von dort verarbeiten. Eine automatische Registrierung ist nicht vorgesehen. Wichtigkeit: Mittel

5.8 Andere Requirements

R28 Das CSCI soll mit Microsoft Visual Studio 2010 entwickelt werden. Wichtigkeit: Gering

R29 Soweit möglich soll zum Zugriff auf die verschiedenen Storage-APIs vorhandene Mittel der einzelnen Provider verwendet werden. Wichtigkeit: Gering

R30 Das CSCI soll auf einem Computer lauffähig sein, der in mindestens folgenden Leistungsdaten entspricht:

- CPU: 2 Ghz Single oder Dual-core
- RAM: 2 GB RAM
- Bit: 32-bit
- Grafik: integrierte Grafik mit min. 128 MB RAM

Wichtigkeit: Gering

R31 Das CSCI soll keinen direkten Einfluss auf die physikalische Sicherheit von Personen oder Objekten in seiner unmittelbaren Umgebung haben. Wichtigkeit: Gering

R32 Die Software soll auf Systemen der Windows-Betriebssystem-Familie mit installiertem .NET Framework 4 lauffähig sein. Wichtigkeit: Hoch

Kapitel 6

Design eines sicheren, verteilten Dateisystems

Gemäß dem internationalen Standard für System-Design-Dokumente (im Englischen „software design description“ (SDD)) IEEE 1016-1998 [35] beinhaltet ein solches Dokument folgende Punkte:

1. Einleitung
2. System-Architektur
3. Detaillierte Beschreibung der Komponenten
4. Benutzerschnittstellen-Design
5. Übriges Material

Dieser Abschnitt der Arbeit beschreibt den prinzipiellen Aufbau des CSCIs, seine Architektur, die Hintergründe für getroffene Designentscheidungen, eventuelle Einschränkungen, seine Ziele und nicht zuletzt auch die Details der Lösung. Das Zielpublikum dieses Abschnittes sind interessierte Entwickler und restliches interessiertes Publikum, welches einen Überblick über Funktion und Erweiterungsmöglichkeiten der Software bekommen möchte.

6.1 Systemüberblick

Folgende Abbildung zeigt den Zusammenhang zwischen externen und internen Komponenten der Lösung:

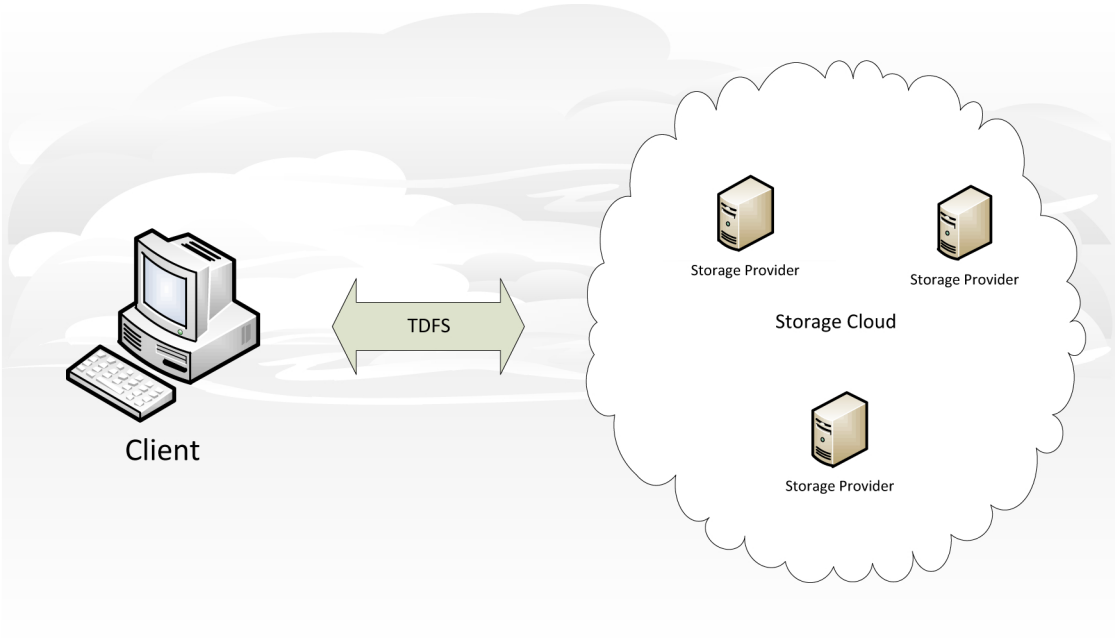

Abbildung 6.1: Systemüberblick

Ziel des CSCIs ist es, die unterschiedlichen APIs der einzelnen Storage-Provider zu abstrahieren und um zusätzliche Funktionen wie z.B. Splitten von Files und Verschlüsselung zu erweitern.

6.2 Annahmen und Abhängigkeiten

Das Design dieses CSCIs geht davon aus, dass die Kommunikationsformen für den Zugriff (REST, SOAP, HTTP, ...) konstant bleiben. Obwohl die Architektur Möglichkeiten zur Anpassung vorsieht, kann nicht gänzlich ausgeschlossen werden, dass bei einschneidenden Veränderungen Teile der Funktionalität verloren gehen.

Erwartet wird außerdem dass das CSCIs in Zukunft Zugriff auf zusätzliche Storage-Provider bieten soll. Aus diesem Grund wird die Zugriffsschicht mittels einer Plugin-Architektur ausgeführt.

6.3 Architektur

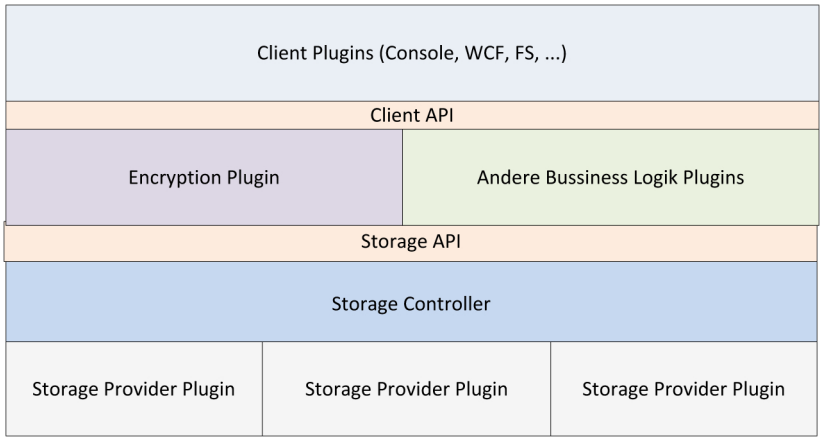

Abbildung 6.2: Programmschichten TDFS

Um den Requirements gerecht zu werden, werden die in Abbildung 6.2 gezeigten Programmschichten verwendet.

Die Basis des Systems bilden die einzelnen Storage-Provider, die Verbindung zwischen den APIs der einzelnen Storage-Provider und dem Storage-Controller herstellt.

Der Storage-Controller übernimmt die eigentliche Logik zum Verteilen von Daten auf die einzelnen Storageprovider, die Konvertierung zu Bitstreams, Redundanz- und Synchronisationsfunktionen und ähnliches.

Die Storage-API stellt eine Schicht dar, die nach oben hin die Sprache von Filesystemen spricht, d.h. Parameter wie Metadaten und Bitströme bereitstellt.

Als nächstes befindet sich darüber eine Business-Logik-Schicht, welche die Möglichkeit zum Einbau weiterer Plugins, wie beispielsweise einem Verschlüsselungs-Plugin, bereitstellt.

Darauf aufbauend stellt die Client-API ein Interface für Client-Plugins zur Verfügung. Als Client-Plugins kommen Konsolenapplikationen, Programme mit Graphischem User-Interface, Windows-Services oder auch Webservices in Betracht.

Im Folgenden soll auf die Details der einzelnen Schichten eingegangen werden.

6.4 Komponentenbeschreibung

6.4.1 Storage-Plugins

Wie bereits erwähnt stellen die Storage-Plugins die logische Verbindung zu den einzelnen Storageprovidern her. Die einzelnen Provider stellen ihre Dienste mit unterschiedlichen Zugriffsmöglichkeiten bereit. Die von ihnen genutzten Varianten (siehe 2.1) sind:

- SOAP
- REST
- JSON

Je nach Reifegrad der API sind mehr oder weniger komplexe Use-Cases abbildbar. Folgender Ablauf für Kommandos an die Storage Provider ist vereinfacht und generalisiert gegeben:

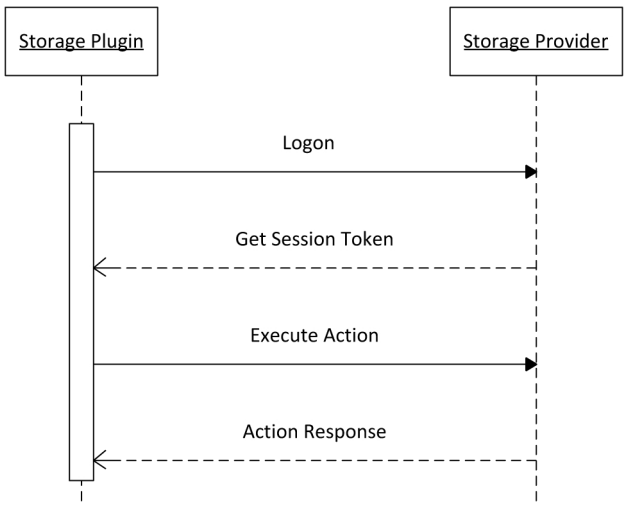

Abbildung 6.3: Einfacher Aufruf einer Funktion beim Storage Provider

Wie ersichtlich, wird zuerst bei Durchführung einer Aktion ein Access Token vom Provider eingeholt (oder im Falle eines bereits bestehenden Tokens dieses verwendet). Daraufhin wird die vom Storage-Controller (genaue Beschreibung siehe 6.4.2) angeforderte Aktion durch das Plugin beim Provider durchgeführt. Nachdem sich Art und Weise des Rückgabewertes von Provider zu Provider unterscheiden und vom Storage-Plugin interpretiert werden muss, konnte der Vorgang in obiger Abbildung nur vereinfacht dargestellt werden.

Für die Anbindung an die Storage-Provider werden nach Möglichkeit die bereits vorhandenen API-Pakete (SDKs, Bibliotheken, etc.) der Hersteller der APIs benutzt. Dies bedingt zwar eine mögliche leichte Einschränkung im Funktionsumfang (wenn beispielsweise das SDK den Möglichkeiten der API nicht entsprechen sollte), wird allerdings durch die vom Hersteller im Eigeninteresse durchzuführende Qualitätssicherung größtenteils wettgemacht.

Nachfolgend das Beispiel eines Zugriffs auf den Amazon S3 Service mittels des Amazon S3 API-SDKs (Listing aus der Amazon S3-Transfer-Utility-Beispielapplikation [36]):

```
1  this._transferUtility = new TransferUtility(appConfig["AWSAccessKey"], appConfig["
       AWSSecretKey"]);
2  // Make sure the bucket exists
3  this._transferUtility.S3Client.PutBucket(new PutBucketRequest().WithBucketName(this
       .Bucket));
4
5  TransferUtilityUploadRequest request = new TransferUtilityUploadRequest()
6            .WithBucketName(this.Bucket)
7            .WithFilePath(this.UploadFile)
8            .WithTimeout(FIVE_MINUTES)
9            .WithSubscriber(this.uploadFileProgressCallback); // Pass in a callback
                 so upload progress can be tracked.
10 this._transferUtility.Upload(request);
11
12 displayMessageBox("Completed file upload!", "Success", MessageBoxButton.OK,
       MessageBoxImage.Information);
```

Listing 6.1: Beispielcode

Es zeigt die einfache Möglichkeit des Uploads einer Datei zu Amazon. Ein Bucket ist in diesem Zusammenhang eine virtuelle Festplatteneinheit bei Amazon. Der Filepath bezeichnet die lokal abliegende Datei. Ein Timeout kann für den Uploadversuch eingestellt werden. Es steht auch eine Notifizierung für den Fortschritt des Vorganges zur Verfügung. Zu Beginn müssen Access-Key und Secret-Key übergeben werden.

Als bevorzugtes Design-Pattern wird in diesem Fall „Dependency Injection“ verwendet [37]. Dieses trennt die Implementierung eines Interfaces von ihrer Definition und ermöglicht so lose gekoppelte und leicht erweiterbare Plugin-Architekturen. Ein Assembler setzt zur Laufzeit die implementierenden Klassen in Relation zum Interface.

6.4.2 Storage-Controller

Der Storagecontroller bildet das Herz der Lösung. Er verwaltet die von anderen Schichten angelieferten Daten. Außerdem liefert er eine integrierte Sicht auf bei den Storage-Providern liegende Daten.

Folgende Operationen werden zum Aufbau der integrierten Ansicht des verteilten Dateisystemes durchgeführt:

- Logon bei den Storage-Providern
- Listen des Inhaltes
- Auslesen der Metadaten
- Erzeugung einer einheitlichen Ansicht

Daraus ergibt sich folgender Ablauf:

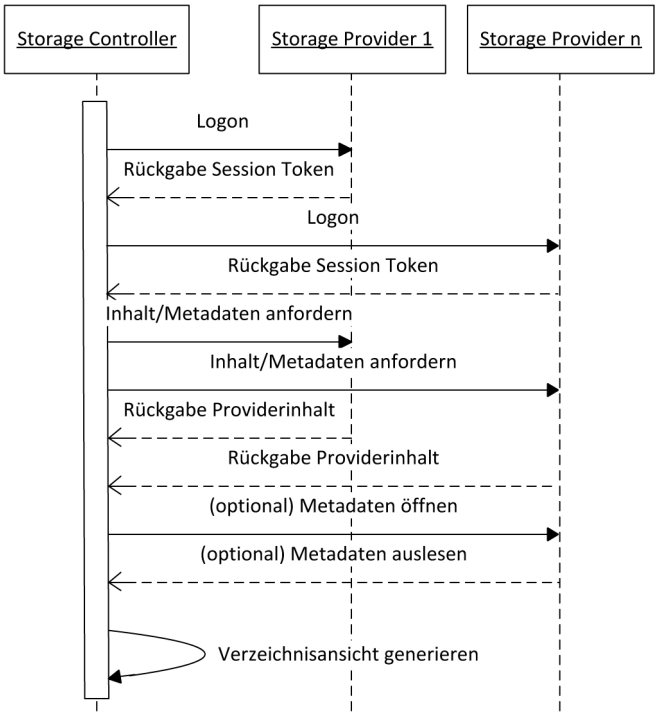

Abbildung 6.4: Sequenzdiagramm Aufbau Storageansicht

Zuerst wird eine Verbindung mittels vorher bekanntgegebener Einlogdaten zu den Providern hergestellt und deren Inhalt angefordert. Falls unterstützt, werden die Metadaten des Dateisystems angefordert. Bei Providern, die Metadaten nicht direkt unterstützen, werden diese in Dateiform beim Provider abgelegt. Beim Vorgang der Ansichtserstellung werden diese also, falls vorhanden, zur Ermittlung weiterer Information herangezogen.

Durch Kombination aller Daten ist es dem Storage-Controller möglich, für den Benutzer transparent Dateien, die bei verschiedenen Storageprovidern liegen, in einer einheitlichen Ansicht zur Verfügung zu stellen.

6.4.3 Rulesets

Die Rulesets, also Regelsätze sind ein Bestandteil des Storage-Controllers. Mittels ihrer Hilfe kann man dem Storage-Controller gewisse Mindestkriterien bei der Verwaltung der Daten auferlegen. Eine ähnliche Idee wird in [38] verfolgt. Es erlaubt dem Benutzer eine granulare Abstimmung der Funktionsweise seiner Software.

Diese Kriterien werden in einer Konfigurationsdatei definiert. Eine solche Datei könnte beispielsweise so aussehen:

```
1  <storagecontroller>
2          <MinimalProviderCount>2</MinimalProviderCount>
3          <UseEncryption>yes/no</UseEncryption>
4          <SplitLargeFiles>yes/no</SplitLargeFiles>
5          <includeFileTypes>
6                  <Type>xml</Type>
7                  <Type>doc</Type>
8                  <Type>pdf</Type>
9          <includeFileTypes>
10 </storagecontroller>
```

Listing 6.2: Beispiel-Ruleset

Der Storage-Controller baut sich aus diesen Regelsätzen eine Präferenzordnung auf, auf deren Basis er die Daten auf die verschiedenen Storage-Provider verteilt. Ist eines der Kriterien nicht erfüllbar, so wird dies dem Benutzer zur Kenntnis gebracht (zum Beispiel in einem Logfile), aber trotzdem ein Best-Effort-Ansatz durchgeführt.

6.4.4 Splitten/Mergen

Um es dem CSCI zu ermöglichen, das Requirement des verteilten Datenträgers zu erfüllen, muss das CSCI eine Funktion zum Teilen und Zusammenfügen von Dateien beinhalten (sogenanntes „Splitten/Mergen“). Aufgrund der vorliegenden Metadaten, d.h. hauptsächlich dem auf dem Storage Provider verbleibenden Restspeicherplatz, entscheidet der Storage-Controller über die Teilung einer Datei.

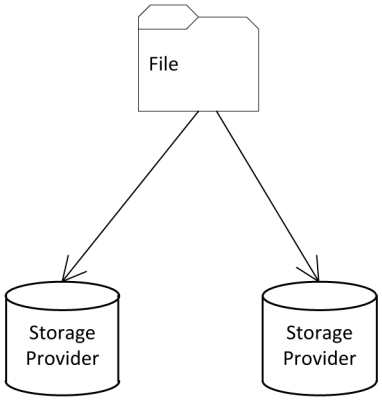

Abbildung 6.5: Teilen einer Datei

Ist die Entscheidung zum Teilen einer Datei erfolgt, so speichert das CSCI die zum Zusammensetzen des Files notwendigen Daten in den Metadaten der Datei ab. Hierbei handelt es sich um den Ursprungsdateinamen und die Liste der zum Zusammensetzen des Files notwendigen Teildateien. Sollten die in dieser Liste notwendigen Teildateien nicht vorhanden sein, ist das Zusammensetzen der Datei ohne Dateikorruption nicht möglich und dem Benutzer wird eine Fehlermeldung angezeigt.

Der programmiertechnische Ablauf ist wie folgt:

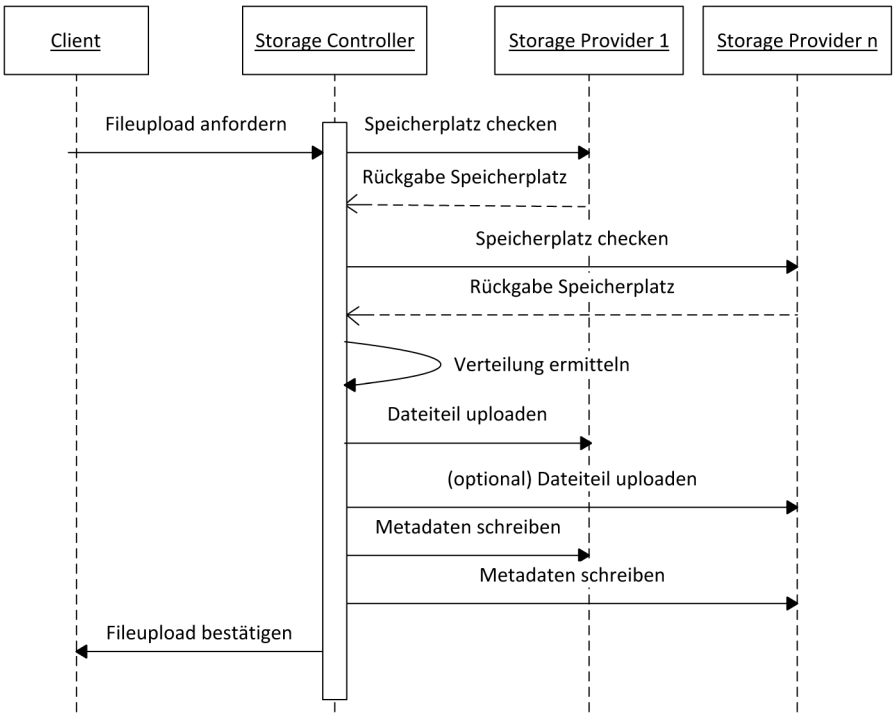

Abbildung 6.6: Verteilung einer Datei

Das Zusammensetzen der Datei aus ihren Einzelteilen verläuft ähnlich:

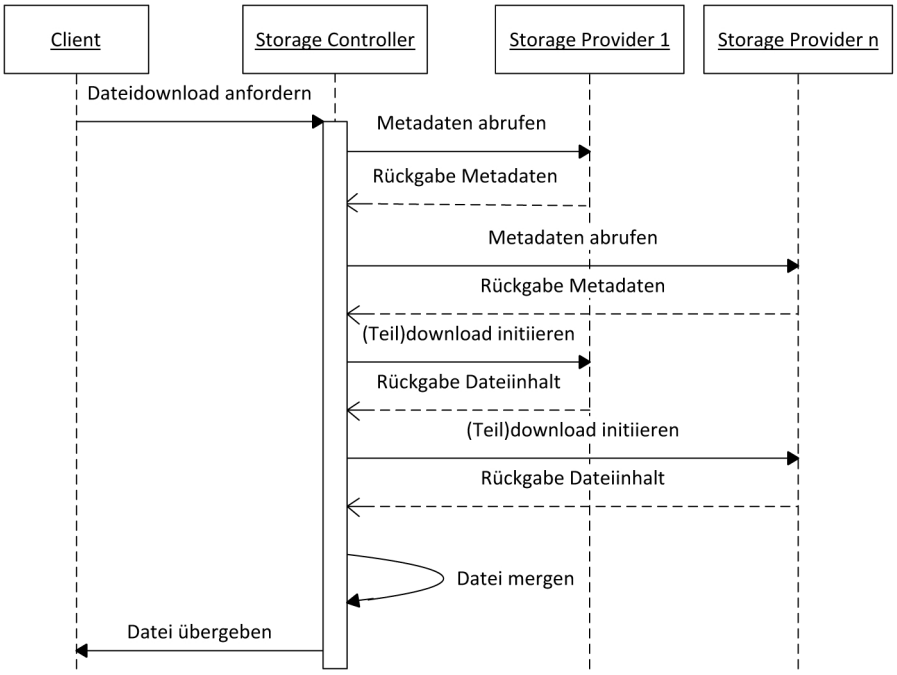

Abbildung 6.7: Zusammenfügen einer Datei

Mittels dieser Methoden lässt sich für den Benutzer trotz unterschiedlicher Storage-Provider der Anschein einer einheitlichen Storage-Schicht erwecken, die Benutzung ist also für den Benutzer transparent.

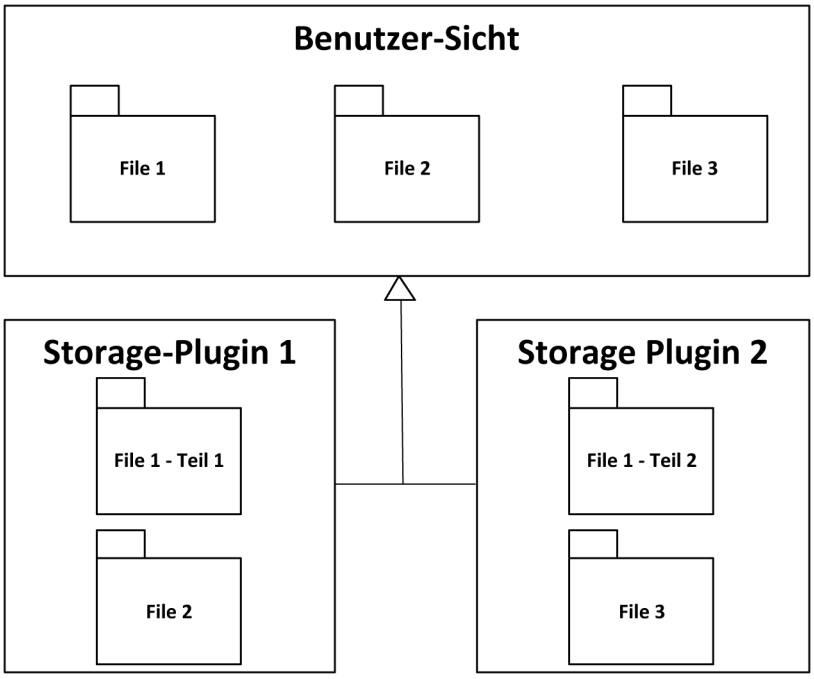

Abbildung 6.8: Transparente Dateisystemsicht

6.4.5 Redundanz

Um die im Evaluationskapitel 4 vorgestellte Forderung nach Datensicherheit zu erfüllen, muß das CSCI Funktionen zur mehrfachen Speicherung von Nutzdaten bereit-

stellen. Der Storage-Controller nimmt die Verteilung der Daten vor, zusätzlich wird allerdings darauf geachtet, dass die in den Rulesets(6.4.3) vorgestellten Bedingungen erfüllt werden.

Für den Fall, dass im Ruleset die eine Redundanz-Bedingung definiert ist, z.B. dass eine Datei zumindest auf 2 Storage-Providern vorzuhalten ist, beginnt der Storage-Controller diese auf die geforderte Art zu verteilen. Der Vorgang ist in diesem Fall analog zu Abbildung 6.6 (Verteilung einer Datei).

Nachdem Redundanz eine Sicherheitsmaßnahme gegen Speicherausfälle ist, ist auch der Fehlerfall kurz zu umreißen. Laut Kapitel 6.4.2 baut der Storage-Controller beim Starten des CSCIs aus den auf den Storage-Providern vorliegenden Daten eine Ansicht auf. Die dazu notwendigen Informationen werden weiter vorgehalten. Das betrifft vor allem die Details zum Aufbewahrungsort und zur Aufbewahrungshäufigkeit der einzelnen Dateien.

Sollte nun ein Storage-Provider nicht mehr erreichbar sein, so wird vom Storage-Controller eine Evaluation der Erfüllung der Rulesets vorgenommen. Sind für bestimmte Dateien die Bedingungen nicht mehr erfüllt, so versucht der Storage-Controller den von den Rulesets verlangten Zustand zu erfüllen und verteilt die Daten dementsprechend.

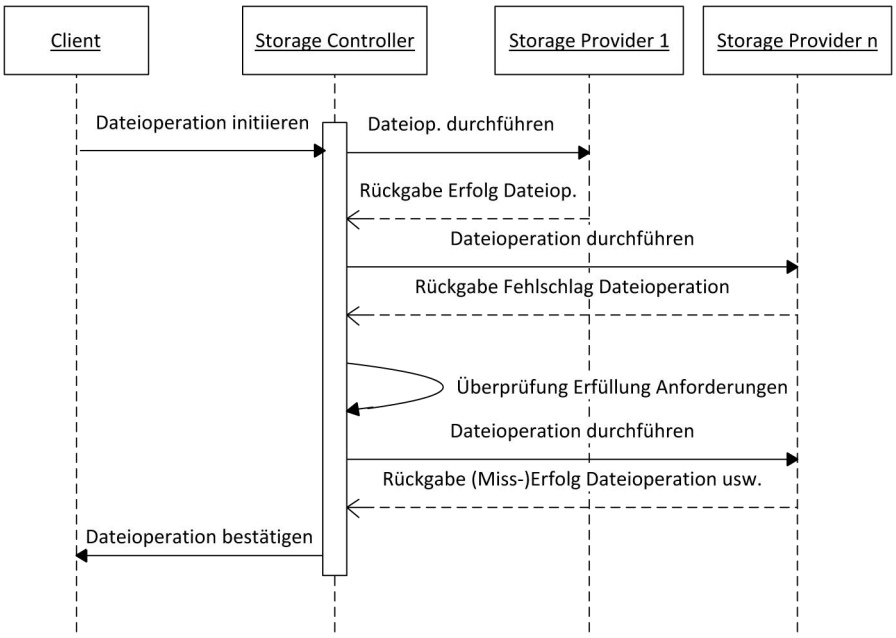

Abbildung 6.9: Sequenzdiagramm Fehlerwiederherstellung

6.4.6 Authentifikation

Ein Schlüsselthema des CSCIs ist auch die sichere Authentifikation. Die Einlogdaten sollen nicht für andere Teilnehmer einsehbar sein. Außerdem muss das System gegen Man-in-the-Middle-Angriffe geschützt sein. Während die Transportverschlüsselung HTTPS einigermaßen gegen Man-In-The-Middle-Angriffe schützt, lässt sie dennoch die Endpunkte der Kommunikation ungeschützt.

Um diesem Nachteil entgegenzutreten wird im CSCI das Authentifizierungsprotokoll OAuth unterstützt. Dieses gewährt durch einen Authentifizierungsmechanismus,

bei dem anstelle einer Username/Passwort-Kombination sogenannte Request- und Access-Tokens ausgetauscht werden, (Details siehe Kapitel 2.1) ein höheres Sicherheitsniveau.

Zum Nachteil des CSCIs wird dieser Standard nicht von allen Storage-Providern unterstützt (siehe Kapitel 4). Daher betreibt die Software in diesem Fall einen Best-Effort-Mechanismus. In diesem Fall kann das Storage-Plugin auch als Fallback auf unsichere Methoden zurückgreifen, der Storage-Controller wird allerdings immer versuchen zuerst eine Methode der Anmeldung mit höheren Sicherheitsmerkmalen zu wählen. Für den Fall der Benutzung unsicherer Anmeldemethoden wird eine Warnungsmeldung an den Benutzer abgeschickt.

Für optimale Sicherheit bei der Authentifikation (nach heutigem Kenntnisstand) sollten alle Storage-Provider auf dem OAuth-Standard basierende Authentifikationsverfahren anbieten.

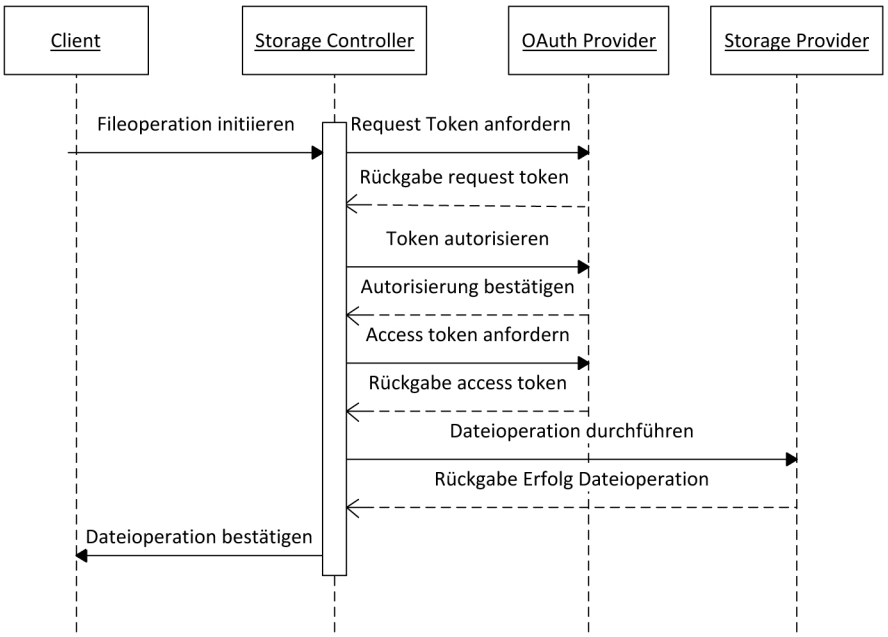

Abbildung 6.10: Sequenzdiagramm Oauth Anmeldung

6.4.7 Verschlüsselung

Um der Forderung nach sicherer Verwahrung der Nutzdaten nachzukommen muss das CSCI einen Mechanismus zur Verschlüsselung von Daten bieten. Daher benutzt das CSCI dafür ein auf einem asymmetrischen oder symmetrischen Verschlüsselungsverfahren basierendes Verfahren. Als Basis des asymmetrischen Verfahrens dient die RFC 4880 [39], in der die genauen Mechanismen zur Verschlüsselung von Nutzdaten beschrieben sind.

Mittels Ordner-basierter Verschlüsselung wird sichergestellt, dass nur berechtigte Benutzer Zugriff auf die Daten haben. Außerdem wird ausgeschlossen, dass durch Mitarbeiter der Storage-Provider Sicherheitslecks entstehen können.

Basierend auf 3.3 wird zur Verschlüsselung folgende Funktion herangezogen:

$$M_{encrypted} = F_s(M_{unencrypted}, K_{public}) \tag{6.1}$$

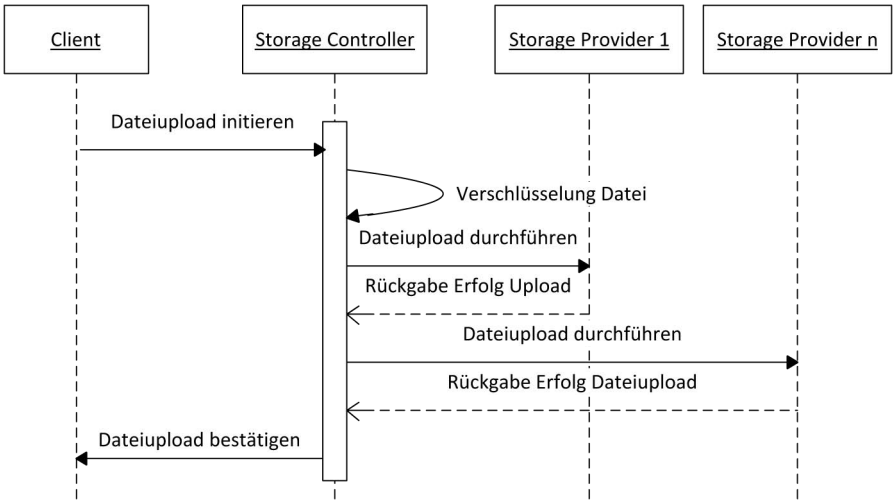

Abbildung 6.11: Sequenzdiagramm Verschlüsselung

Die verschlüsselten Daten M_b sind die Funktion der unverschlüsselten Daten M_s, verschlüsselt mit dem öffentlichen Schlüssel K_s.

Zur Entschlüsselung wird der private Schlüssel des Benutzers herangezogen um wieder den Ursprungszustand der Daten herstellen zu können.

$$M_{unencrypted} = F_s(M_{encrypted}, K_{private}) \tag{6.2}$$

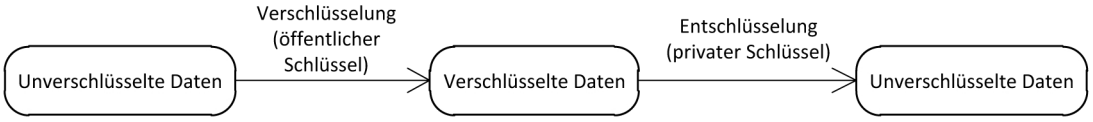

Abbildung 6.12: Asymmetrische Verschlüsselung

Eine Ordner-basierte Verschlüsselung bietet den Vorteil, dass Einzelobjekte flexibel und eventuell auch mit verschiedenen Schlüsseln verschlüsselbar sind. Somit ist eine punktuelle Aktualisierung von Daten möglich, die bei einem Datenträger-Verschlüsselungs-System (wie z.B. Truecrypt[33]) nicht gegeben wäre. Durch die Verschlüsselung erhöht sich zwangsläufig die verbrauchte Prozessorzeit, dies stellt bei heutigen Mehrprozessorsystemen allerdings kein relevantes Hindernis mehr dar.

Die Speicherung der Schlüssel erfolgt über die Windows-Registry. Diese erschwert die Übertragung der Benutzerprofile auf andere Computer. Die Profile werden mittels der von Windows bereitgestellten API eingetragen und zuvor verschlüsselt, um einfaches Auslesen zu verhindern.

6.4.8 Gleichzeitiger Dateizugriff

Bei einem Cloud-Dateisystem wie TDFS muss auch die Möglichkeit des parallelen Zugriffs mehrerer Benutzer auf dieselbe Datei bedacht werden. Ohne weitere Behandlung dieses Themas kommt es im schlimmsten Fall zur Überschreibung der Datei

[33]http://www.truecrypt.org/

mit der lokalen Version des jeweiligen Benutzers, weiters ist auch eine Beschädigung der Datei möglich.

Für die Lösung dieses Problems sind mehrere Ansätze denkbar. Eine Möglichkeit wäre die Singularisierung des Zugriffs über ein bestimmtes Gateway. Diese Lösung wird von etlichen kommerziellen Anbietern im betrieblichen Umfeld benutzt. Da alle Clients nur mehr sequentiell auf die Daten zugreifen können, ist das Problem somit behoben. Nachteil dieser Lösung ist der Single-Point-Of-Failure im Server und die Bindung an eine oder mehrere lokale Instanzen der Software, welche Nachteile im Bezug auf Geschwindigkeit und Stabilität bringen können.

TDFS wird ähnlich wie RFS in Kapitel 3.3 als Client-zentriertes System designt, allerdings ohne zentralen Server. Das bedeutet, dass es zwischen den Storage-Providern und der Client-Software keine Zwischenschicht gibt. Somit kann aufgrund der Unterschiedlichkeit der einzelnen Storage-Provider das Problem des gleichzeitigen Zugriffs nicht am Server gelöst werden sondern muss am Client gelöst werden.

Eine weitere Möglichkeit wäre wie bei NFS ein Locking-Protokoll einzuführen und den Zugriff zu synchronisieren. Diese Methodik ist für die Synchronisation ungeeignet, weil der Server als Koordinator ausfallen kann und die einzelnen Clients zueinander nicht in jedem Fall Konnektivität besitzen.

Die plausibelste Idee ist die Verwendung eines SVN-ähnlichen Algorithmus (siehe Abbildung 6.13). Hierbei werden Metadaten in das Cloud-Datei-System integriert, welche es dem TDFS-Service ermöglichen über die Aktualität der veränderten Datei zu entscheiden. Weiters kann in diese Metadaten auch ein Hash der Daten miteingebracht werden, womit das System gegen Korruption der Daten bei der Übertragung gehärtet wird.

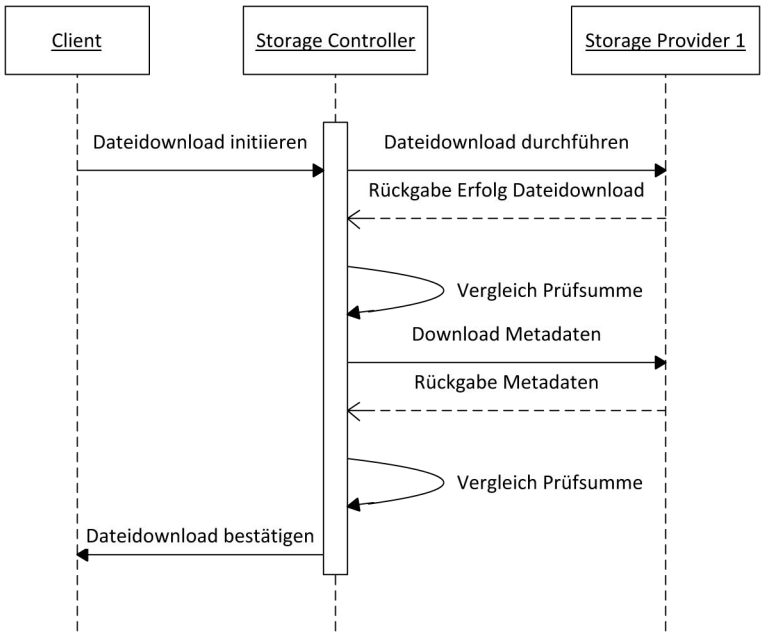

Abbildung 6.13: Sequenzdiagramm Verhinderung Korruption bei gleichzeitigem Datenzugriff

6.4.9 Klassendiagramme

Aus dem vorgestellten Design ergibt sich folgendes generelles Klassendesign:

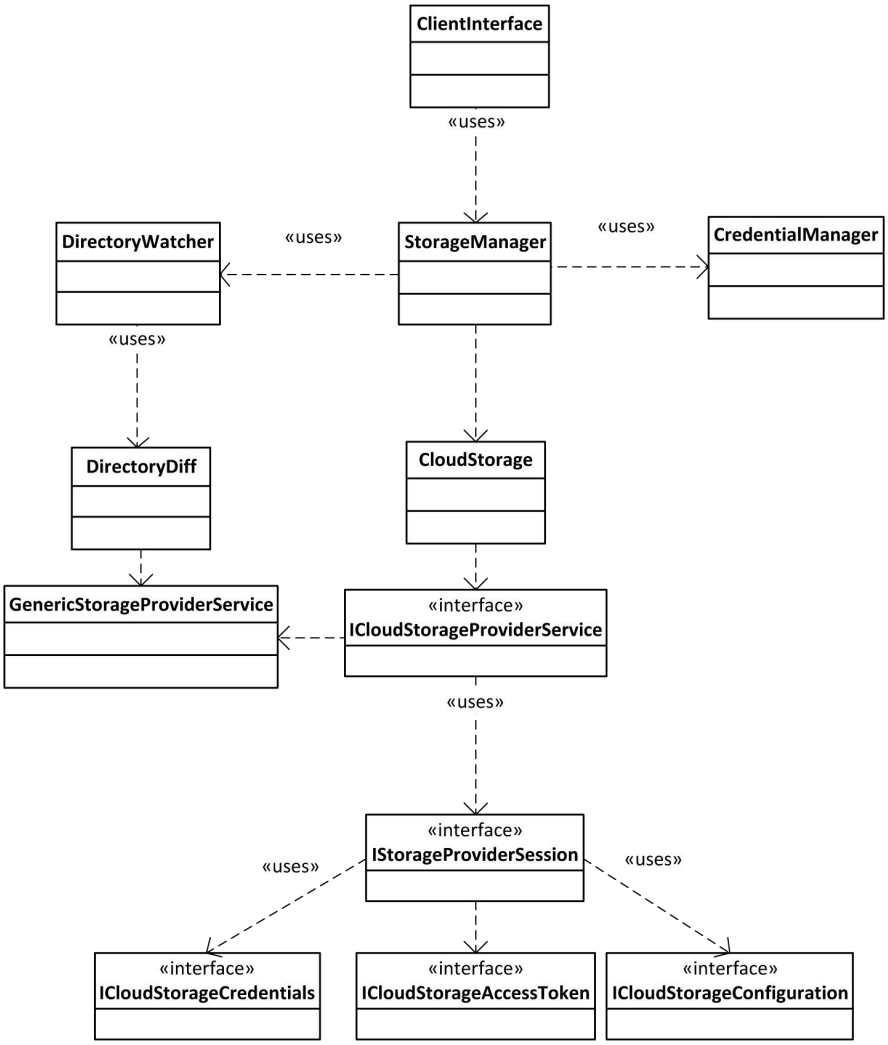

Abbildung 6.14: Paket-Abhängigkeiten

Kernstück der Designs ist der StorageManager, welcher Zugangsdaten und Dateioperationen verwaltet. Zur Erfüllung dieser Aufgaben bedient er sich der Klasse CredentialManager, welche Schnittstellen zur Speicherung und zum Auslesen der Zugangsdaten beinhaltet. Die Klasse DirectoryWatcher überwacht das Verzeichnis mit den Cloud-Dateien und ist mittels der von ihr benutzten Klasse DirectoryDiff für die Synchronisation der Daten mit den Cloudprovidern zuständig. Sobald ein Synchronisationskommando abgeschickt wird, werden mit Hilfe der zuständigen CloudStorage-Instanz die jeweiligen Daten in Up- und Download-Richtung aktualisiert. Bestandteile des für eine CloudStorage zuständigen CloudStorageProviderServices sind die Access-Token, welche bei der OAuth-Authentifikation genutzt werden, die CloudStorage-Credentials (zuständig für die Legacy-Authentifizierung), und die jeweilige CloudStorageConfiguration (sie enthält Daten wie providerspezifische URLs oder auch Limits, die eingehalten werden müssen). Nach außen hin stellt die Klasse ClientInterface die Dateisystemoperationen zur Verfügung.

Ein Storage Provider wird dabei durch sein Plugin vertreten, und kann (sinnvollerweise mit verschiedenen Konfigurationen) beliebig oft instanziert werden. Es sind folgende Abhängigkeiten geplant:

Klar ersichtlich ist die Generalisierung gemeinsamer Eigenschaften der einzelnen Sto-

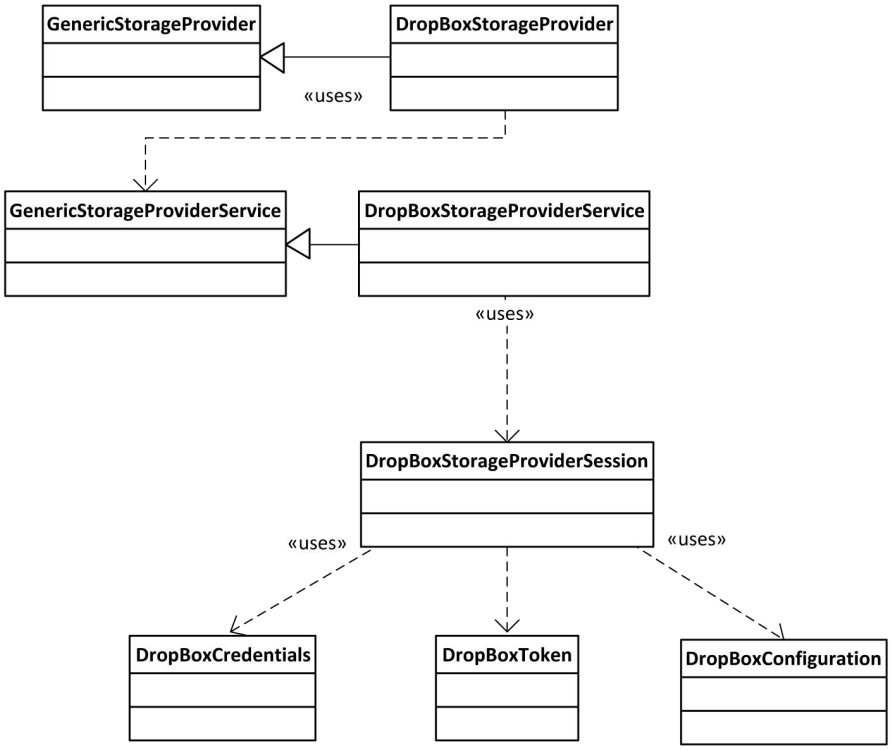

Abbildung 6.15: Design Storage Provider

rageprovider in den generischen Klassen, um Redundanzen und damit Fehler durch Code-Vervielfachung zu vermeiden. Bei der Initialisierung des Storage-Providers wird ein Sitzungs-Objekt erstellt, welches für die Dauer der Sitzung die zur Sitzung gehörigen Informationen wie OAuth-Token, Konfiguration und Credentials beihaltet. Das ist insofern wichtig, als (vor allem OAuth-basierte) Dienste ein Token benötigen um Dateisystemoperationen in der Cloud überhaupt zuzulassen. Permanente Re-Authenfizierung würde den Synchronisationsprozess unnötig verlangsamen.

Jedes Dateisystem-Objekt kann in eine von zwei Klassen fallen, Datei oder Verzeichnis. Diese bilden die Basis für die Dateisystemstruktur, welches sich als folgender Graph darstellen lässt:

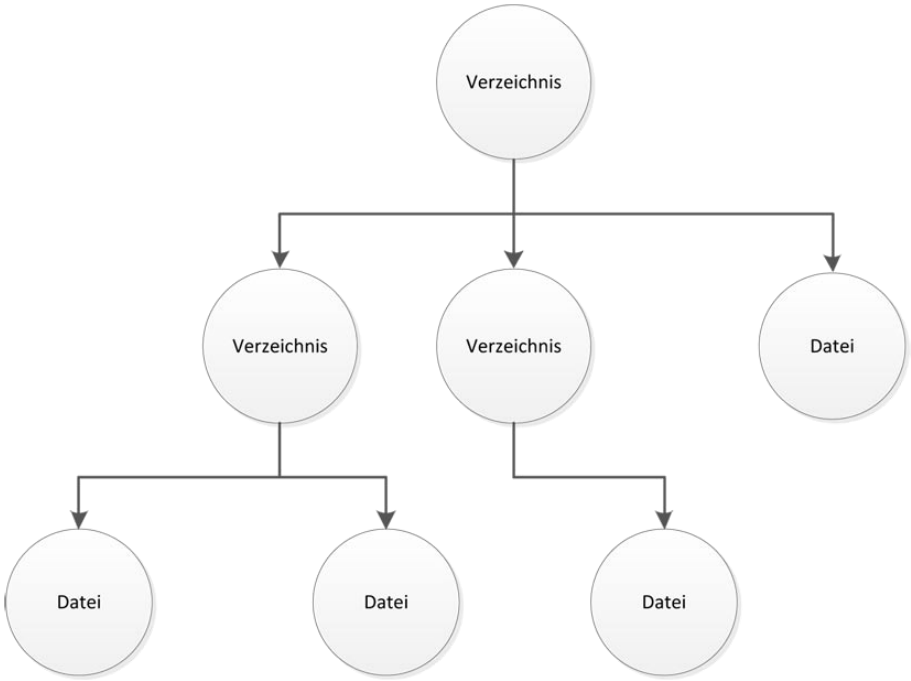

Abbildung 6.16: Design File System

Bei Betrachtung der einzelnen Dateisystemobjekte als Knoten und der Kanten als Namen der jeweiligen Objekte ergeben sich aus dem Baum die Dateinamen der Dateisystemobjekte. Hierbei spielt der Wurzelknoten eine besondere Rolle, da er keinen Namen besitzt. Beispielsweise wird auf diese Art eine Datei „\Testverzeichnis \Testdatei“ aus dem Wurzelknoten, der Namenskante „Testverzeichnis“, dem zugehörigen Verzeichnisknoten, der Namenskante „Testdatei“ und deren zugehörigen Dateiknoten abgebildet.

Im Klassendiagramm schlägt sich diese Idee in folgender Weise nieder:

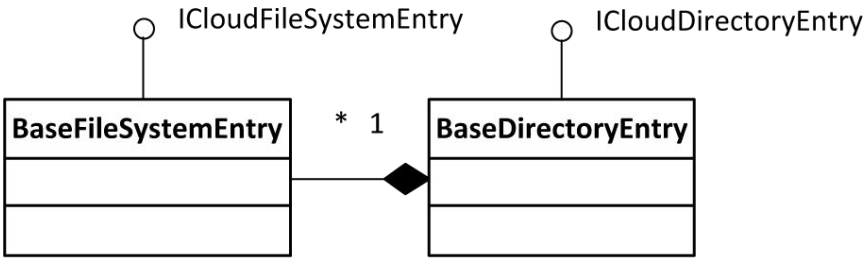

Abbildung 6.17: Design einzelner Dateiobjekte

Die Objekte gehen Eltern-Kinder-Beziehungen ein. Ein Verzeichnis (Elternteil) kann als Eigenschaft eine beliebeige Aufzählung von Dateien (Kinder) beinhalten. Auf diese Art und Weise kann der Verzeichnisbaum korrekt abgebildet und der Name einer Datei durch Durchlaufen des Baumes und Zusammenfügen der einzelnen Namen errechnet werden.

Ein wichtiger Aspekt des Datei-Objektes ist noch seine Eigenschaft als durchführende Entität einer Dateioperation. Im Gegensatz zu anderen Dateisystemoperationen wie Verschieben, Löschen oder Umbenennen einer Datei sind der Up- bzw. Download einer Datei nicht nur mit den Metadaten einer Datei, sondern auch mit ihrem

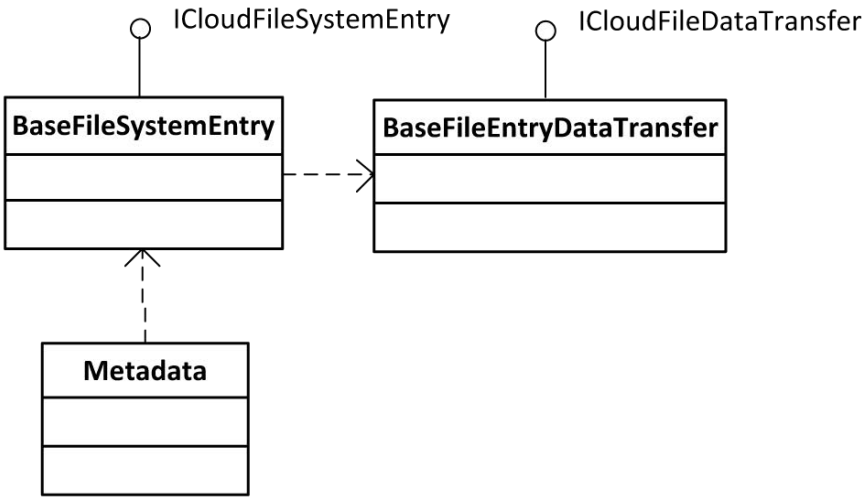

Abbildung 6.18: Daten-Transfer und Metadaten

Inhalt verbunden. Um diesem Umstand Rechnung zu tragen werden diese Transfer-Operationen als Bestandteile der Dateisystem-Objekte implementiert.

Nicht alle Dienstanbieter die Möglichkeit zur Speicherung benutzerdefinierter Metadaten zur Verfügung. Aus diesem Grund wird für Anbieter, welche eine solche Möglichkeit nicht bieten, bei jedem Dateisystem-Objekt eine dazugehörige Metadaten-Datei angelegt, welche die Metadaten eines Dateisystem-Objekts enthält.

Kapitel 7

Implementation

Dieses Kapitel beschreibt die Implementierung des in Kapitel 6 vorgestellten Designs. Im Zuge der Erstellung des Designs und dem Beginn der Implementation wurden dazu mehrere öffentliche und im Quelltext zugängliche Cloud-Speicher-APIs und deren Funktionsumfang untersucht. Zu diesen zählen Lokad[34], Spaceblock[35], Apache Libcloud[36], JClouds [37], Dasein-Cloud[38], CloudLoop[39] und auch Sharpbox[40]. Nachdem der Fokus dieser Arbeit auf der Implementierung der API in einer Windows-nahen Programmiersprache liegt, wird als Implementierungssprache C# herangezogen. Die umfangreichen Standardbibliotheken von .NET erleichtern die Erstellung des nötigen Codes. Da diese Bibliotheken durch umfangreiche Qualitätsprüfungen gegangen sind, verringert sich die Fehlerwahrscheinlichkeit im Programmcode. Aus diesem Grund wurde auch der Ansatz gewählt, ein bestehendes Projekt um die geforderten Funktionen zu erweitern, anstatt ein gänzlich neues Projekt zu erstellen. Ein bestehendes Projekt hat bereits getesteten Code und ist nach der Erweiterung hochwertiger als ein Proof-Of-Concept-Prototyp.

Die Java-basierten Multi-Cloud-APIs (Dasein, JClouds, Cloudloop) weisen eine sinnvolle interne Struktur auf. Sie bezeichnen den Objekt-basierten Speicherdienst der von ihnen implementierten Provider als BlobStorage und führen Operationen auf ihnen aus. Allerdings fehlen ihnen Funktionen wie zum Beispiel die Synchronisation, Verschlüsselung ist wiederum (via BouncyCastle[41]) vorhanden. Wie bereits in Kapitel 3.3 beschrieben, ist Lokad in qualitativ hochwertigem Code geschrieben, allerdings zu diesem Zeitpunkt stark spezialisiert und mit vielen Funktionen versehen, die für den Betrieb als Dateisystem für die Cloud nicht notwendig sind. LibCloud befindet sich in einem zu frühem Stadium um damit ein Projekt zu implementieren. Bei den C#-basierten APIs existiert mit SpaceBlock eine nicht mehr aktiv weiterentwickelte, auf Amazon S3 spezialisierte API. Sharpbox hingegen weist eine einfache und leicht zu erweiternde interne Struktur auf, implementiert bereits einige Provider-APIs und bietet Unit-Tests zur Überprüfung der Ergebnisse und ist auch sonst dem gewünschten Design sehr nahe. Daher wird im Rahmen der Implemen-

[34] http://code.google.com/p/lokad-cloud/
[35] http://spaceblock.codeplex.com/
[36] http://libcloud.apache.org/
[37] http://code.google.com/p/jclouds/
[38] http://sourceforge.net/projects/dasein-cloud/
[39] http://www.java.net/project/cloudloop
[40] http://sharpbox.codeplex.com/
[41] http://www.bouncycastle.org/

tierung das Sharpbox-Framework um die fehlende Funktionalität erweitert (Verschlüsselung, Synchronisation über mehrere Anbieter, Client-Interface über einen Webservice). Der Rest des notwendigen Codes wird als eigenständiges Programm implementiert, welches auf das Framework zugreift.

Die folgenden Unterkapitel nehmen einige wenige wichtige Punkte heraus und beschreiben bei der Implementierung entstandene Entscheidungen und dazugehörende Hintergründe.

7.1 Programmaufbau

Wie im Design beschrieben, besteht das TDFS aus mehreren Schichten, im Groben dem Client-Interface, dem Storage-Manager und den einzelnen Storage-Providern. Um eine permanente Überwachung der Cloud-Dienste gewährleisten zu können, ist TDFS nicht als Konsolen-Applikation oder GUI-Applikation, sondern als Windows-Dienst konzipiert. Damit wird gewährleistet, dass bei einem Fehler im Programm welcher zur Terminierung des Dienstes führt, dieser sofort wieder gestartet wird.

Beim Starten des Dienstes werden die benötigten Anmeldedaten aus der Registry abgerufen und mittels ihrer Hilfe die Verbindung zu den Storage-Providern hergestellt. In weiterer Folge werden die Dateibäume der einzelnen Anbieter heruntergeladen und aus ihrer Gesamtheit ein globaler Dateibaum erstellt. Im lokalen Cache fehlende Dateien werden heruntergeladen. Dateien, die nicht auf der notwendigen Anzahl von Provider-Speichern vorhanden sind, werden auf diesen abgespeichert. Der Zugriff auf die Funktionen des Multi-API-Frameworks geschieht dabei über das Client-Interface.

Das Client-Interface ist als WCF-Web-Service implementiert und publiziert im Wesentlichen die von den Storage-Providern bekannte API für Client-Applikationen. Das publizierte Interface hat folgendes Aussehen:

```
13 ICloudStorageAccessToken CurrentAccessToken { get; }
14 ICloudStorageAccessToken Open(ICloudStorageConfiguration configuration,
       CloudStorageCredentials credentials);
15 ICloudStorageAccessToken Open(ICloudStorageConfiguration configuration,
       ICloudStorageAccessToken token);
16 void Close();
17 ICloudDirectoryEntry GetRoot();
18 ICloudFileSystemEntry GetFileSystemObject(String path, ICloudDirectoryEntry parent,
        bool isDir);
19 ICloudDirectoryEntry CreateFolder(String name, ICloudDirectoryEntry parent);
20 Boolean DeleteFileSystemEntry(ICloudFileSystemEntry fsentry);
21 Boolean MoveFileSystemEntry(ICloudFileSystemEntry fsentry, ICloudDirectoryEntry
       newParent);
22 Boolean RenameFileSystemEntry(ICloudFileSystemEntry fsentry, String newName);
23 ICloudFileSystemEntry CreateFile(ICloudDirectoryEntry parent, String name);
24 Uri GetFileSystemObjectUrl(String path, ICloudDirectoryEntry parent);
25 String GetFileSystemObjectPath(ICloudDirectoryEntry fsObject);
26 void StoreToken(List<SecureString> tokendata, ICloudStorageAccessToken token);
27 ICloudStorageAccessToken LoadToken(List<SecureString> tokendata);
28 List<ICloudFileSystemEntry> ListAllObjects();
29 event CloudStorage.LogHandler OnLog;
```

Listing 7.1: Client-Interface

Der Zugriff auf das Client-Interface ist direkt möglich (mittels Anlieferung eines File-Streams), als auch indirekt über die Angabe einer Datei im lokalen Cache.

7.2 Speicherung der Anmeldedaten

Ein eigenes Problem stellt die sichere Speicherung der Anmeldedaten auf einem Computer dar. Die einfache Abspeicherung in einer Konfigurationsdatei stellt in mehrfacher Hinsicht ein Sicherheitsproblem dar. Sollte ein Angreifer in den Besitz der Konfigurationsdatei kommen, so kann er mit ihrer Hilfe Zugriff auf die Dateien des Benutzers erhalten. Bei der redundanten Aufbewahrung von Daten mittels des TDFS würde der Zugriff auf einen einzelnen Provider genügen, um Zugang zu einem wesentlichen Teil der Daten zu erhalten. Dieser Umstand erhöht die Notwendigkeit einer sicheren Speicherung.

Mehrere Varianten stehen zur Auswahl:

- Sicherung in einer Isolated Storage
- Sicherung in einem verschlüsselten File
- Sicherung in einer Datenbank
- Sicherung in der Registry mittels DPAPI

Sicherung in einer Isolated Storage bedeutet die Abspeicherung der Anmeldedaten in einem speziellen von .NET angebotenen Speicherort. Dieser ist von anderen Applikationen getrennt und kann bei Bedarf auch verschlüsselt sein. Aus diesem Grund wird diese Variante oft von Programmen benutzt. Ein Nachteil dieser Variante ist die Zugriffsmöglichkeit durch Administratoren, Unmanaged-Code, und Programmen, welche in einem Full-Trust-Context (d.h. einem Administrator-Kontext) laufen. Isolated Storage ist aufgrund der Kopiermöglichkeit daher kein sicherer Aufbewahrungsort für Anmeldedaten.

Eine andere Möglichkeit ist die Speicherung in einer verschlüsselten Datei. Die Anmeldedaten werden im XML-Format in einer Datei abgespeichert. Diese Datei wird dann verschlüsselt und ist mittels eines im Programmes abgespeicherten Schlüssels zugreifbar. Nachteil dieser Variante ist die Möglichkeit der Kopierens der Datei und damit der Zugriff von einem unautorisierten Computer aus.

Weiters können die Anmeldedaten auch in einer SQL-Datenbank abgespeichert werden. Dies entspricht der Abspeicherung in einer Datei, wobei hier das Masterpasswort mit den Zugriffsdaten für die Datenbank vergleichbar ist. Diese Variante bietet also ebenfalls keine erhöhte Sicherheit.

Als letzte Variante gibt es die Speicherung in der Registry. Da die Speicherung im Klartext unsicher ist, wird zur Verschlüsselung die vom Windows-Betriebssystem angebotene DPAPI verwendet. Diese verschlüsselt die übergebenen Daten und setzt als Schlüssel Anmeldedaten und SID des Benutzers ein. Es ist theoretisch möglich die Daten aus der Datenbank zu exportieren und in eine andere Registry zu importieren. Allerdings sind die Daten ohne den richtigen Benutzerkontext (welcher nur auf der speichernden Maschine besteht) nicht ohne weiteres dekodierbar. Im TDFS wird aus diesem Grund diese Kombination aus Speicherung in der Registry und DPAPI verwendet.

Zur Veranschaulichung zeigt Listing 7.2 einen Teil von gespeicherten Anmeldedaten.

```
[HKEY_CURRENT_USER\Software\TDFS\2]
"Username"=hex:01,00,00,00,d0,8c,9d,df,01,15,d1,11,8c,7a,00,c0,4f,c2,97,eb,01,\
  00,00,00,80,16,e8,de,2f,44,0c,4b,90,9f,82,f3,a4,a7,82,c6,00,00,00,00,02,00,\
  00,00,00,00,10,66,00,00,00,01,00,00,20,00,00,00,18,34,f8,51,43,1e,5d,d3,af,\
  27,6a,dd,bf,b2,65,d3,38,a2,ee,17,5e,c0,59,24,b8,b3,9e,37,4b,f0,0b,6b,00,00,\
  00,00,0e,80,00,00,00,02,00,00,20,00,00,00,2d,e9,15,5c,65,62,ab,43,4b,02,85,\
  32,d7,81,7d,95,c3,e4,10,be,1e,26,d8,f4,5a,8d,c9,b6,78,f4,aa,42,20,00,00,00,\
  a9,ce,53,e5,2a,ef,46,20,fe,5d,9a,09,17,26,a8,46,0a,fc,06,64,b6,cb,15,9d,3f,\
  cb,3b,43,b9,72,9f,32,40,00,00,00,6b,23,9d,f8,ab,d3,f6,13,c0,59,38,78,88,b4,\
  12,c4,2c,43,c6,fc,69,bd,2d,bd,3b,df,24,38,27,e8,fe,c0,64,c3,79,c0,5d,b7,06,\
  2c,d0,31,33,84,9f,f3,5f,0a,1c,7c,60,17,32,64,43,f5,cd,81,91,21,2f,09,2d,47
"Password"=hex:01,00,00,00,d0,8c,9d,df,01,15,d1,11,8c,7a,00,c0,4f,c2,97,eb,01,\
  00,00,00,80,16,e8,de,2f,44,0c,4b,90,9f,82,f3,a4,a7,82,c6,00,00,00,00,02,00,\
  00,00,00,00,10,66,00,00,00,01,00,00,20,00,00,00,6a,29,ca,5a,95,f2,6f,f5,43,\
  63,d5,0f,1b,08,2c,b9,90,43,54,6b,eb,53,a9,ac,15,78,46,a2,66,c4,c5,dd,00,00,\
  00,00,0e,80,00,00,00,02,00,00,20,00,00,00,ea,d7,d5,aa,aa,d7,23,39,13,65,f5,\
  2e,6b,ef,2b,77,27,ae,12,53,9b,02,b3,08,2a,a7,e8,e8,1d,57,6e,aa,10,00,00,00,\
  8d,ed,17,ea,7c,31,1b,3b,61,2e,71,a3,cd,99,c7,b6,40,00,00,00,ca,f1,54,98,36,\
  16,22,6a,45,f0,51,c6,d8,b2,db,da,9d,c5,99,2d,94,79,75,d7,c1,6d,e5,15,f3,f7,\
  77,68,4e,1c,0b,22,cb,eb,df,e7,c4,68,f4,5c,57,a8,f9,a8,15,c4,d0,02,5b,d6,34,\
  a0,68,79,f2,54,08,de,8b,2c
"Type"=hex:01,00,00,00,d0,8c,9d,df,01,15,d1,11,8c,7a,00,c0,4f,c2,97,eb,01,00,\
  00,00,80,16,e8,de,2f,44,0c,4b,90,9f,82,f3,a4,a7,82,c6,00,00,00,00,02,00,00,\
  00,00,00,10,66,00,00,00,01,00,00,20,00,00,00,f8,5c,f9,28,13,36,93,9c,df,87,\
  09,6f,13,06,9d,ca,9f,14,3d,54,7c,43,85,f9,21,8a,13,40,31,b7,87,fe,00,00,00,\
  00,0e,80,00,00,00,02,00,00,20,00,00,00,e4,5a,05,99,50,89,8e,05,c9,81,38,88,\
  aa,b9,02,0c,76,91,19,a4,e0,60,06,ad,67,f1,61,2f,1a,87,db,78,60,00,00,00,a1,\
  3d,fd,48,2c,ea,43,e0,dc,59,f7,17,5a,9c,98,49,5c,ef,89,5f,05,bb,fd,87,c6,aa,\
  94,1e,6f,97,5d,5c,e1,27,57,59,5f,49,ef,f7,9d,03,04,11,58,a9,1f,1b,ac,0f,a5,\
  5d,d4,fe,d5,64,f3,c4,94,4e,13,fa,30,0c,43,ea,f2,45,a9,73,23,bd,56,75,8e,ee,\
  51,f7,bd,6d,95,3b,59,52,c2,09,05,cf,66,67,84,ba,0f,4d,52,9d,40,00,00,00,83,\
  f4,d7,f5,58,1a,94,46,e0,8a,66,61,b8,b7,e3,a5,41,33,94,55,19,b9,c7,94,09,26,\
  b4,16,f9,43,de,65,ec,1b,20,43,57,f6,c9,43,cb,e3,8c,75,c8,15,8e,d5,ad,89,11,\
  b8,99,fd,da,a4,40,c5,c0,4b,c3,89,8b,5c
"Key"=hex:01,00,00,00,d0,8c,9d,df,01,15,d1,11,8c,7a,00,c0,4f,c2,97,eb,01,00,00,\
  00,80,16,e8,de,2f,44,0c,4b,90,9f,82,f3,a4,a7,82,c6,00,00,00,00,02,00,00,00,\
  00,00,10,66,00,00,00,01,00,00,20,00,00,00,c5,c3,e6,13,59,0e,56,47,11,f9,d0,\
  17,ab,5f,0c,9a,22,14,9d,93,68,41,0a,5a,69,fa,ba,ff,83,7a,06,3b,00,00,00,00,\
  0e,80,00,00,00,02,00,00,20,00,00,00,7b,2e,52,bf,e2,77,e1,6c,c6,7a,a6,fe,ff,\
  c0,12,2f,22,4d,b3,dc,a3,d7,dd,78,22,89,8a,98,6a,87,1d,c1,20,00,00,00,dc,00,\
  a1,f3,8d,cb,b6,4c,7a,67,86,bb,11,2d,98,18,3c,db,26,ef,da,9c,26,5a,fc,41,9b,\
  b4,6f,40,c7,0c,40,00,00,00,4c,06,99,9a,c4,ce,e5,76,a9,e7,00,f7,52,0f,78,b7,\
  e3,90,65,13,ef,3b,ee,69,22,f7,83,ee,ea,d5,fa,ef,44,5d,e4,07,1b,72,1e,b5,cb,\
  d4,b1,6c,17,f5,da,4a,c8,a1,ac,e2,19,a4,d3,07,80,3b,84,b9,f8,08,75,2c
"Secret"=hex:01,00,00,00,d0,8c,9d,df,01,15,d1,11,8c,7a,00,c0,4f,c2,97,eb,01,00,\
  00,00,80,16,e8,de,2f,44,0c,4b,90,9f,82,f3,a4,a7,82,c6,00,00,00,00,02,00,00,\
  00,00,00,10,66,00,00,00,01,00,00,20,00,00,00,e1,16,17,23,6e,7a,c3,64,d0,ee,\
  d8,1c,63,4c,dd,77,27,4e,20,5f,01,e5,53,24,72,8c,0e,b7,4e,88,b9,b3,00,00,00,\
  00,0e,80,00,00,00,02,00,00,20,00,00,00,8e,bb,9b,0a,0e,93,13,04,a3,3d,39,70,\
  c9,8f,3e,6b,61,00,76,4f,46,30,5f,6b,31,3c,99,08,ad,52,b7,ac,20,00,00,00,e0,\
  66,30,a5,24,11,ec,3b,e1,ea,6b,a6,c8,37,16,ad,d3,4f,6e,2a,bc,a9,cf,04,a3,c3,\
  a3,32,f0,24,7a,86,40,00,00,00,c7,57,d3,82,01,4c,45,2b,77,97,da,59,1a,ee,ac,\
  58,1e,18,5f,13,da,e7,d3,e2,4d,0e,3e,b8,70,7e,b0,f8,96,a3,7b,44,c7,59,fd,9d,\
  1f,44,3d,26,a3,b2,7c,d7,f3,39,c3,e5,7f,45,0c,1d,c0,1f,33,45,bd,a6,d5,b8
```

Listing 7.2: Credential-Speicherung in der Registry

Die Anmeldedaten werden dazu unter dem in Listing 7.2 angegebenen Registrierungsschlüssel in aufsteigender Nummerierung abgespeichert. Beim Start des Programmes wird der Prozess reversiert, die Daten dekodiert und mit ihnen die Verbindung zu den Providern hergestellt.

Die Speicherung der Anmeldedaten ist durch den Management-Modus des Programmes möglich. In diesem können Anmeldedaten hinzugefügt und bei Bedarf auch entfernt werden. Mittels der Windows-API wird eine sichere Anmeldemaske angezeigt, in die die Anmeldedaten einzugeben sind.

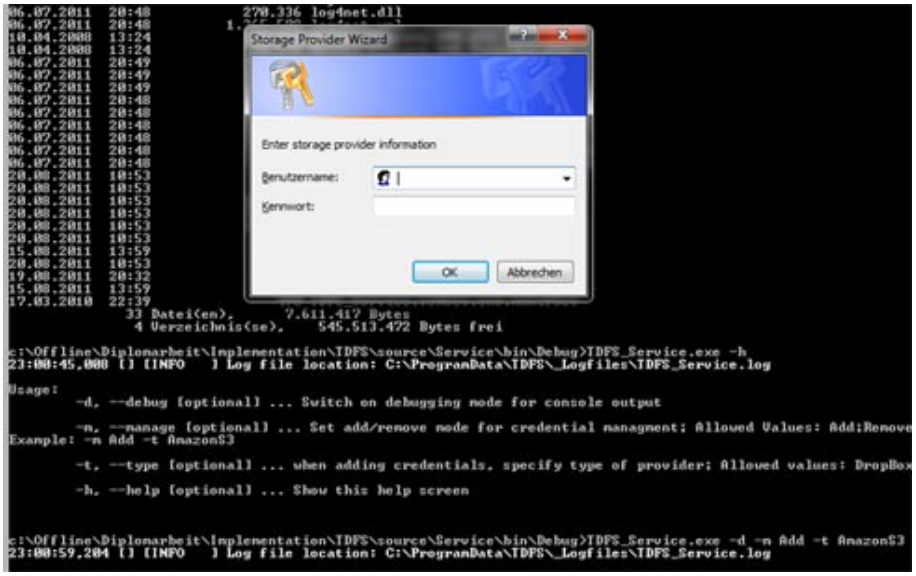

Abbildung 7.1: Eingabemaske für Anmeldedaten

7.3 Desktop-Integration

Ein weiterer wichtiger Punkt ist das Look & Feel eines Programmes, d.h. die Qualität der Bedienbarkeit durch den Benutzer. Im Falle der Implementierung eines Dateisystems ist das primäre Interface für den Benutzer in den meisten Fällen ein Laufwerk, auf das er zugreift. Für den Benutzer sind die Zugriffe transparent, die Mechaniken hinter dem Zugriff sind nicht sichtbar und für den Betrieb nicht weiter relevant.

Um diese Zugriffs-Transparenz zu erreichen, wurden bei anderen Dateisystemen, welche nicht als Kernel-Modul (Linux-Betriebssysteme) beziehungsweise Kernel-Treiber (Windows-Betriebssysteme) konzipiert sind, diese Funktionalität im User-Mode implementiert. Zu diesem Zweck wird ein Dateisystem wie FUSE[42] gewählt. Dieses User-Mode-Dateisystem besteht aus einem im Kernel implementierten Teil, welcher die vom Benutzer induzierten Operationen an einen Teil des Treibers im User-Mode weiterleitet. Der Kernelmode ist gegenüber dem Usermode privilegiert und hat erhöhte Zugriffsrechte und bessere Chancen auf Zuteilung von Prozessorzeit. Normalerweise laufen alle Treiber im Kernelmode, während die vom Benutzer ausgeführten Programme im Usermode gestartet werden. Der Vorteil dieser Architektur ist die Möglichkeit der Bearbeitung der Dateisystem-Operationen durch einen Prozess im User-Mode. Unter Linux ist diese Vorgangsweise vielfach erprobt und erfolgreich anwendbar.

Eine direkte Entsprechung für Windows, auf dessen Basis TDFS implementiert ist, existiert nicht. Die beste Entsprechung unter Windows ist das Dokan-Projekt [43], welches eine ähnliche Architektur bereitstellt. Die ursprüngliche Idee bei der Implementierung des User-Zugriffs war die Benutzung von Dokan. Dadurch wäre im Windows-Explorer ein Laufwerk angezeigt worden, auf das der Benutzer zugreifen könnte. Bei der Evaluierung und testweisen Implementierung hat sich Dokan allerdings als noch zu instabil erwiesen, um in vernünftigem Maße benutzbar zu sein.

[42] http://fuse.sourceforge.net/
[43] http://dokan-dev.net/en/

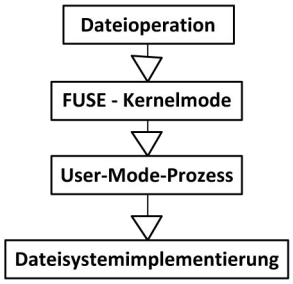

Abbildung 7.2: Dateisystemoperation in User-Mode-Dateiystem

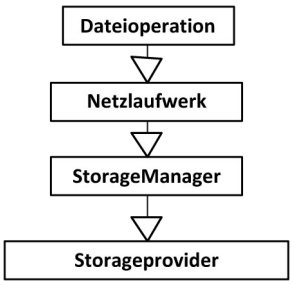

Abbildung 7.3: FUSE-Entsprechung im TDFS

Zusätzlich haben aufgrund der tiefen Verankerung im Windows-Betriebssystem (ein Teil des Treibers läuft wie bei FUSE im hardware-nahen Kernel-Mode) die Tests regelmäßig zu Abstürzen des Betriebssystems geführt. Dies ist gerade bei Dateisystem-Operationen ein gefährlicher Umstand, welcher leicht zu Dateiverlusten führen kann.

Um den Zugriff nach Art von FUSE zu emulieren, legt TDFS bei der Installation ein Netzlaufwerk an. Dieses ermöglicht den Zugriff auf den lokalen Cache von TDFS. Primärer Zweck des Caches ist die Beschleunigung von Zugriffen und des Dienst-Starts. Der Zugriff über das (lokal eingebundene) Netzlaufwerk ist aus Benutzersicht dem Dateisystemzugriff per FUSE am ähnlichsten. Ohne direkten Aufruf der Dateisystemoperationen im User-Mode-Prozess wie bei FUSE oder Dokan ist das Programm auf eine andere Art der Überprüfung auf Veränderungen im lokalen Cache angewiesen. In der vorliegenden Implementation wacht ein Filesystem-Watcher über Änderungen im Dateisystem in lokalen Cache-Verzeichnis. Bei Erstellung, Veränderung und Löschung von Dateien wird die jeweilige Operation über das Client-Interface gestartet. Sollte der Dokan-Treiber zu einem späteren Zeitpunkt eine höhere Stabilität aufweisen ist es möglich den direkten Zugriff durch diesen zu implementieren.

7.4 Synchronisation

Da der Synchronisation bei einem Cloud-Dateisystem wie TDFS ein sehr hoher Stellenwert zukommt, wird diese hier näher beschrieben. Eine eigene Sync-Klasse beinhaltet in der Implementation alle für eine erfolgreiche Synchronisation notwendigen Operationen. Bei der Synchronisation werden lokal nicht vorhandene Dateien aus der Cloud heruntergeladen. Dies ist vor allem dann der Fall, wenn auf einem anderen Computer Veränderungen am Cloud-Laufwerk vorgenommen wurden. Dateien, welche nicht in der Cloud vorhanden sind, werden geteilt, verschlüsselt und dann gemäß der definierten Redundanz-Anforderung zu einzelnen Providern hochgeladen. Die Überprüfung des Deltas des lokalen Caches zur Cloud findet mittels Dateilis-

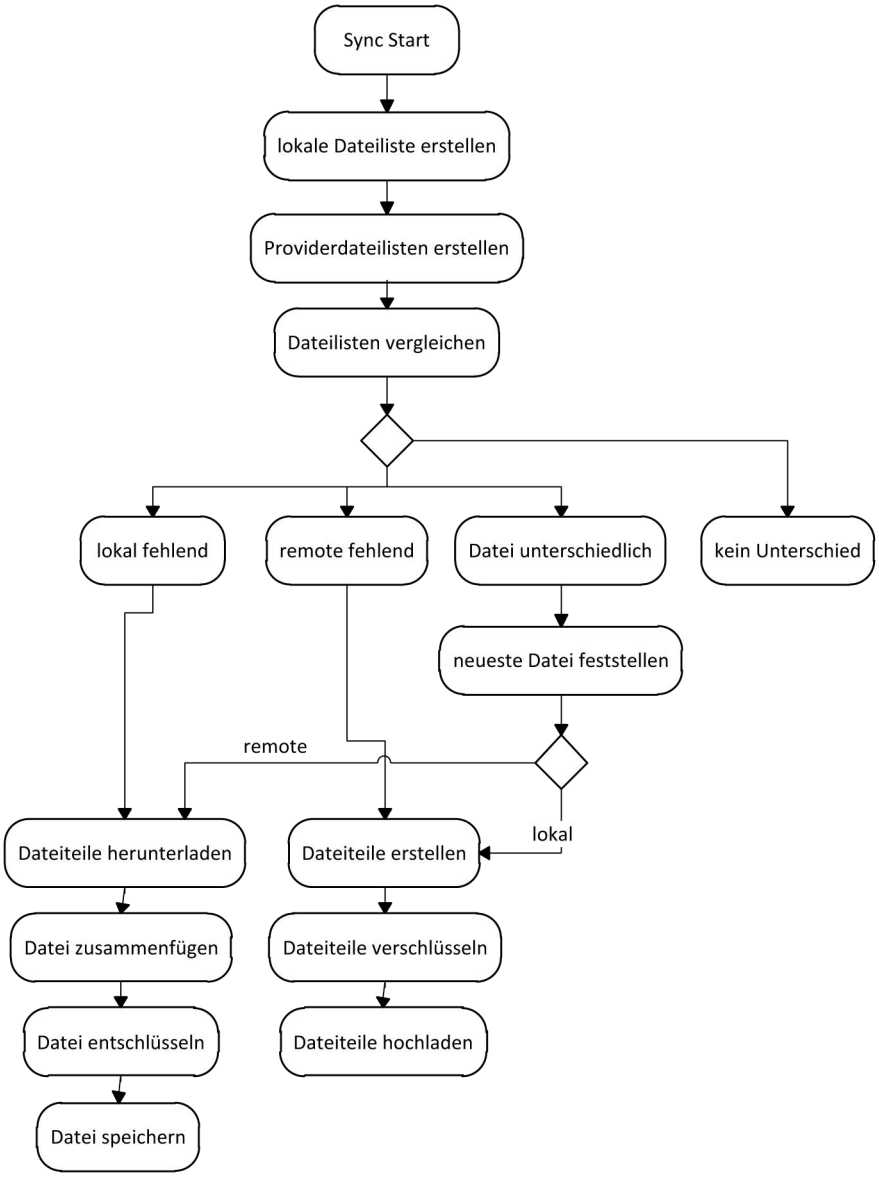

Abbildung 7.4: Synchronisation

ten statt. Zuerst wird je eine Dateiliste eines Providers und die lokale Dateiliste erstellt. Diese werden miteinander abgeglichen und die Unterschiede mit dem jeweiligen Dateinamen als zu bearbeitendes „Unterschieds“-Objekt (Diffresult im Code) abgespeichert. Die Liste mit den Resultaten wird daraufhin sequentiell abgearbeitet, bis lokaler Cache und Cloud-Provider wieder ausgeglichen sind.

Da die Provider durchgehend keinen Push-Mechanismus bei einem Update ihres Speichers anbieten (einzig Amazon S3 bietet einen Push-Mechanismus, dieser ist allerdings nur für den Fall eines Datenverlustes zuständig), ist TDFS auf einen Refresh seines Dateibaums in gewissen definierbaren Intervallen angewiesen. Dies stellt ein Problem dar, da manche Anbieter in ihren AGBs das Abgreifen der gesamten Ordnerstruktur explizit als unerwünscht deklarieren. Ohne Push-Mechanismus ist TDFS allerdings ohne Timer-gesteuerten Scan nicht in der Lage, Veränderungen am Provider festzustellen. Abbildung 7.4 zeigt die Herangehensweise.

Im Falle eines Providerausfalls wird bei der Synchronisation überprüft ob alle Teile einer Datei bei anderen Providern vorhanden sind. Ist dies nicht der Fall, so wird

die Datei als fehlend markiert und wenn möglich durch eine lokale Kopie ersetzt. Weiters wird die Dateiprüfsumme überprüft um eine unbemerkte Korruption einer Datei zu vermeiden.

7.5 Verschlüsselungs-Plugins

In Kapitel 4 wurden die Fähigkeiten von Providern untersucht und Absenz einer durchgehenden client-seitigen Verschlüsselung festgestellt. Aus diesem Grund wurde in Kapitel 6.4.7 ein Design für eine Verschlüsselung im TDFS vorgestellt. Diese Verschlüsselung ist in der modularen Architektur des TDFS als Storage-Plugin realisiert, welches bei Dateisystemoperationen aufgerufen wird.

Die Implementierung beinhaltet zwei Plugins zur Verschlüsselung:

- ein asymmetrisches Plugin, welches auf PGP/GPG basiert
- ein symmetrisches Cipher-Plugin

Zur Inbetriebnahme eines der Plugins ist es notwendig, dieses im Konfigurationsfile anzugeben. Eine Mehrfachverschlüsselung ist möglich, allerdings nicht implementiert.

```
84      <!-- Encryption plugin to use, possible Values: GPG, Symmetric -->
85      <add key="EncryptionPlugin" value="Symmetric"/>
86
87      <!-- passphrase to use for encryption key -->
88      <add key="Passphrase" value="Die Cloud ist toll!"/>
89
90      <!-- GPG encryption specific settings -->
91      <!-- Sender/Recipient used for gpg encryption, must have private key -->
92      <add key="Recipient" value="cloud@cloud.com"/>
```

Listing 7.3: Plugin-Auswahl

Bei GPG sind weiters Schlüsselpaare zur Benutzung der Verschlüsselung notwendig. Diese sind in einem Unterverzeicnis „Keys“ abzulegen und müssen den privaten Schlüssel zur Verschlüsselung der Daten enthalten. Der korrekte Empfänger (im Normalfall ist dieser ident mit dem Inhaber des privaten Schlüssels) muß angegeben werden.

Bei dem Cipher-Plugin sind umfangreichere Einstellungen zu tätigen. Der zugrunde liegende Algorithmus der verwendeten Bibliothek[44] ist AES[45]. Das Verschlüsselungsverfahren hat bereits bekannte Schwächen[46], und sollte daher bei Verwendung mit einer Schlüsselgröße von mehr als 256 Bits verwendet werden.

```
93      <!-- symmetric encryption specific settings -->
94      <!-- init vector to use, must be exactly 16 characters long -->
95      <add key="initVector" value="1234567891011121"/>
96      <!-- keysize to use, recommended value 512 -->
97      <add key="keySize" value="512"/>
98      <!-- password iterations -->
99      <add key ="passwortIterations" value="1"/>
100     <!-- salt value -->
101     <add key ="saltValue" value="test1"/>
102     <!-- hashing algorithm -->
```

[44] http://www.obviex.com/CipherLite/
[45] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[46] https://cryptolux.org/FAQ_on_the_attacks

```
103     <add key="hashAlgorithm" value="SHA512"/>
```

Listing 7.4: Cipher-Einstellungen

Wie bei jeder Verschlüsselungslösung bedingt die Veränderung der entsprechenden Konfiguration die Devalidierung sämtlicher im Cloud-Speicher vorhanden Daten. Sie sind nachher durch das TDFS nicht entschlüsselbar, allerdings sind die Daten durch Wiederherstellung der alten Konfiguration wieder zugreifbar.

Kapitel 8

Validierung

Um die Erreichung der in Kapitel 1.1 vorgestellten Ziele überprüfen zu können, werden Messungen durchgeführt. Diese Messungen orientieren sich in ihrem Aufbau an [40]. Aufgrund des verteilten Aufbaus von TDFS wird eine Messung der integrierten Zugriffszeit durchgeführt. Damit ist das Herunterladen eines in mehrere Chunks geteilten Files gemeint, im Gegensatz zur reinen Messung von Zugriffszeiten zu einzelnen Providern.

Zur Durchführung der Messung werden mehrere Dateien in unterschiedlichen Größen bereitgestellt. Die Größen der Dateien belaufen sich auf:

- 10 KB
- 1000 KB = 1 MB

Die Einstellung zur Teilung einer Datei in mehrere Teile wird auf 1 Megabyte festgesetzt.

Zur Überprüfung der Zugriffszeiten werden diese Dateien zu den einzelnen Providern hochgeladen und die Dauer des Vorgangs notiert. Als zweiter Test werden Dateien der genannten Größenklassen von den Anbietern heruntergeladen. Die Stichprobengröße beträgt n = 100. Die verwendeten Storage-Anbieter sind:

- Amazon S3
- DropBox
- StoreGate (ein WebDAV-basierter Anbieter)

Die gemessenen Zeiten beinhalten die Dauer vom Absetzen des Kommandos im Programm bis zum Abschluss der Operation unter Durchquerung sämtlicher Programmschichten. Durch den singulären Zugriff auf die einzelnen Storage-Provider wird ein Zugriff ohne die Verwendung einer RAID-artigen Technologie simuliert.

Die Auswertungen der beiden Dateigrößen-Klassen (Abbildungen 8.1, 8.3, 8.4, 8.2) zeigen keine signifikanten Unterschiede im Zugriff auf verschiedene Storage-Provider. Storegate ist im Gegensatz zu Amazon S3 und Dropbox ein schwedischer Anbieter. Es hat somit gegenüber den amerikanischen Anbietern einen Standortvorteil, welcher sich in den Übertragungszeiten niederschlägt.

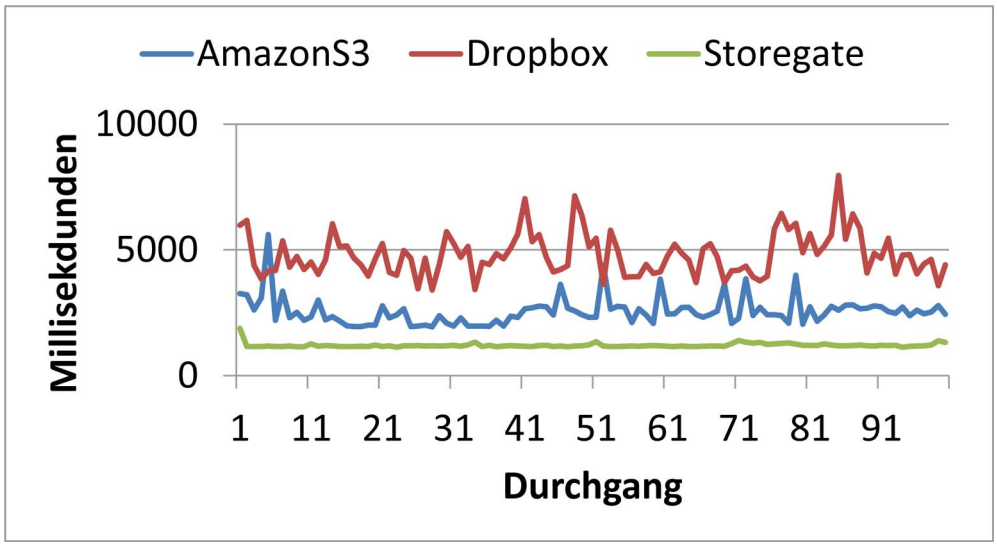

Abbildung 8.1: Übertragungszeiten - Download - 10 Kilobyte

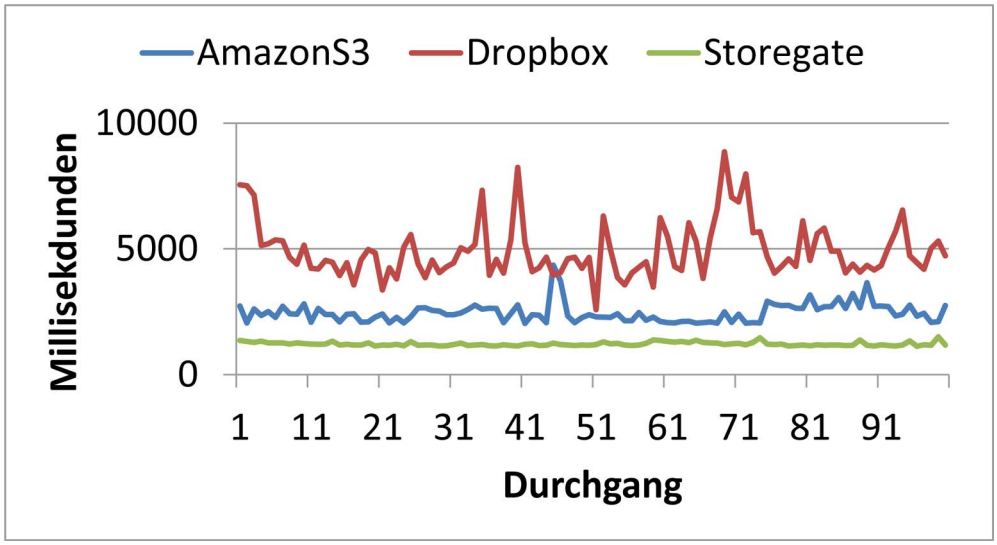

Abbildung 8.2: Übertragungszeiten - Download - 1 Megabyte

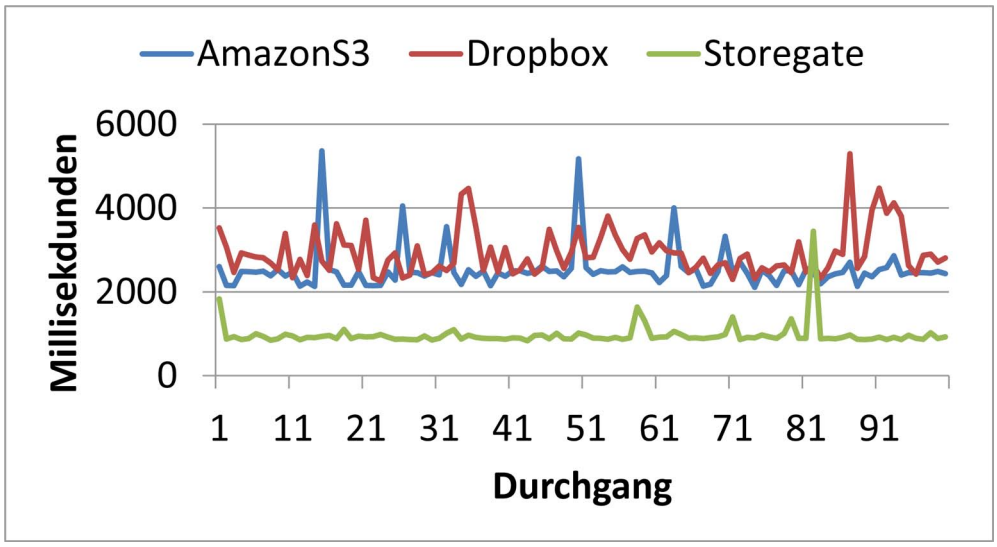

Abbildung 8.3: Übertragungszeiten - Upload - 10 Kilobyte

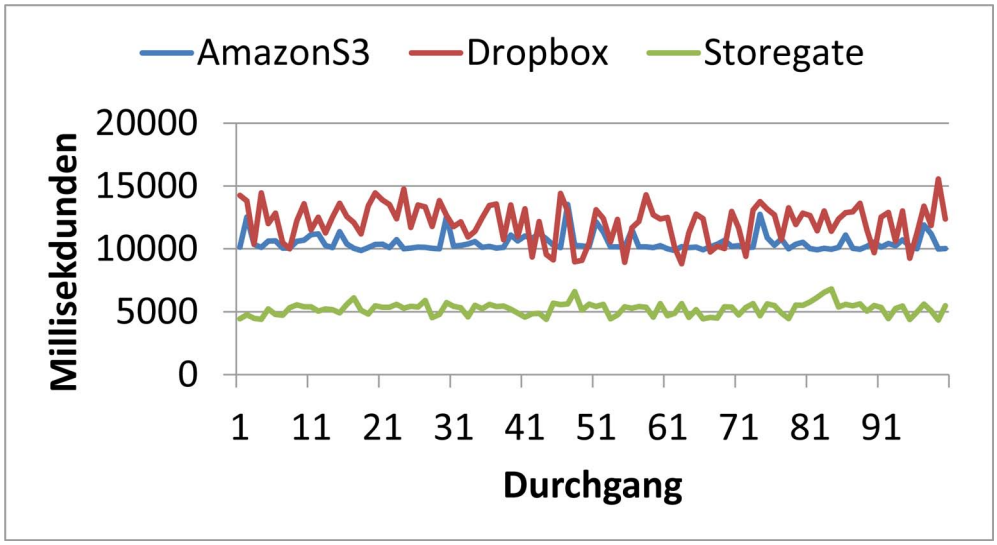

Abbildung 8.4: Übertragungszeiten - Upload - 1 Megabyte

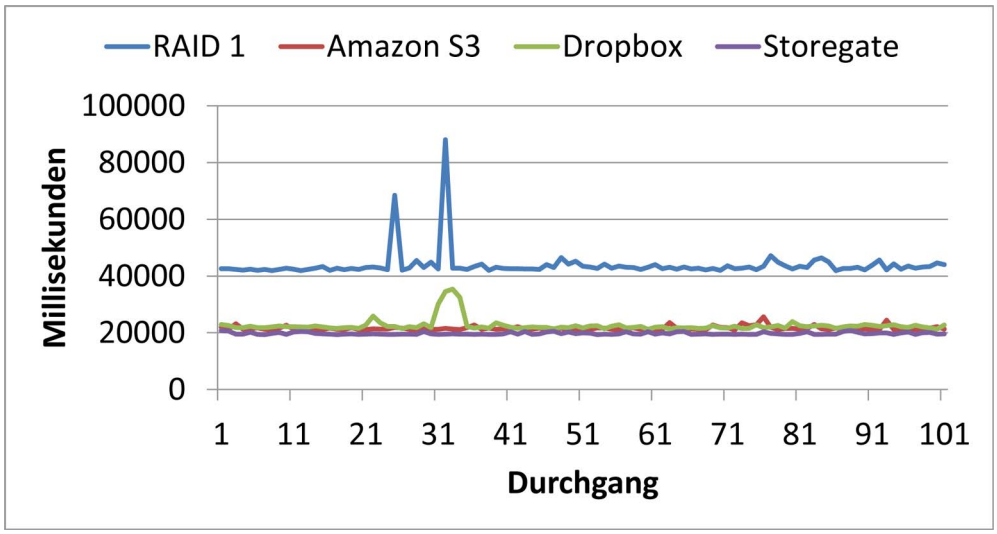

Abbildung 8.5: Übertragungszeiten - Upload RAID-Modus - 1 Megabyte

Im der nächsten Messung wird ein RAID-artiger Zugriff simuliert. Die Einstellung zur Teilung von Dateien wird auf 0,5 MB festgesetzt. Die Anzahl der Provider, zu denen Dateiteile mindestens hochgeladen werden müssen, wird auf 2 gesetzt. Diese Einstellungen entsprechen RAID 1 (vgl. Kapitel 3.1), welches einfache Redundanz auf mindestens 2 Datenträgern sicherstellt.

Abbildung 8.5 zeigt eine lineare Skalierung der benötigten Zeit zur vollständigen Übertragung der Dateiteile zu den Storage-Providern. Zum direkten Vergleich sind auch die Daten des Testlaufs mit nur einem Storage-Provider enthalten. Die Übertragungen finden sequenziell statt, bei paralleler Übertragung sind bessere Werte möglich. Im besten Fall werden alle Dateiteile gleichzeitig übertragen und so die benötigte Zeit minimiert.

Eines der Ziele des TDFS ist Ausfallsicherheit. Zur Simulation eines Provider-Ausfalls wird programmatisch beim Herunterladen einer Datei der 1 MB-Klasse einer der Storageprovider entfernt. Eine andere Möglichkeit wäre die Auflösung des Servernamens des Speicheranbieters auf das Loopback-Interface (127.0.0.1). Das hätte allerdings

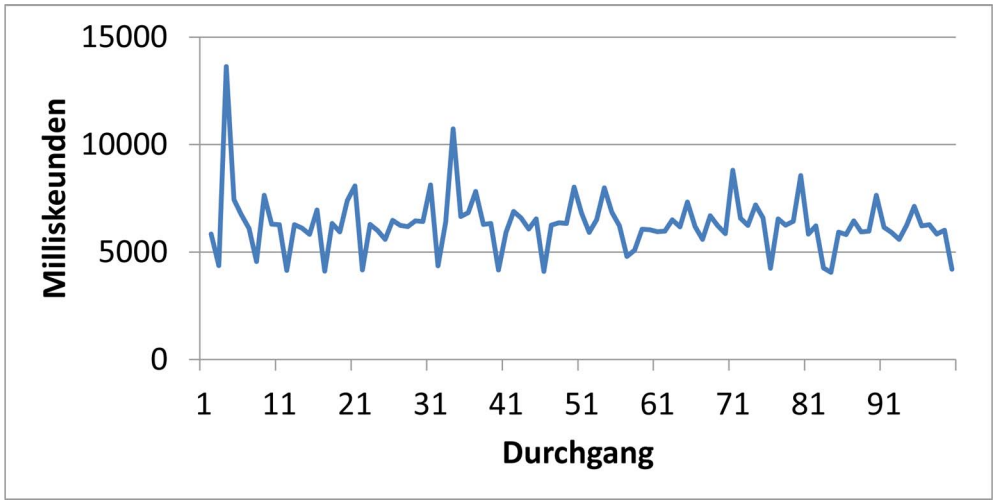

Abbildung 8.6: Übertragungszeiten - Download RAID-Modus - Fehlerfall

das Abwarten eines Timeouts zur Folge. Die programmatische Variante lässt außerdem einen Zufallsgenerator für die Ausfalls-Wahrscheinlichkeit des Providers zu.

Die benötigte Zeit zum Download einer Datei variiert nur geringfügig (vgl. Abbildung 8.6) bei Ausfall eines oder mehrerer Storage-Provider. TDFS lädt bei Ausfall eines Providers den gerade bearbeiteten Dateiteil von einem anderen Provider herunter.

Kapitel 9

Zusammenfassung und Ausblick

Im Rahmen dieser Magisterarbeit sollte ein Dateisystem („TDFS - Transparent Distributed File System“) zum Zugriff auf mehrere Speicheranbieter entworfen werden. Gründe dafür sind die Möglichkeit eines Ausfalles eines Speicheranbieters bzw. schlechte Dienstqualität. Besonderes Augenmerk war dabei auf den Schutz der Daten vor unbefugtem Zugriff zu legen.

Nach einer kurzen Darstellung der notwendigen Grundlagen (Basistechnologien, Dateisysteme, RAID-Formen) wurde ein Überblick über die derzeit vorhandenen Speicheranbieter gegeben. Preise und Angebot der Anbieter liegen eng beieinander und erschweren so die Auswahl des richtigen Anbieters. Bei großen Datenmengen ist Amazon S3 der günstigste Anbieter. Nur die wenigen im Detail vorgestellten Provider bieten eine vollständige API an. Bei den übrigen sind die zu einer Implementation notwendigen Funktionen nur teilweise vorhanden.

Auf Basis der in der Evaluation ermittelten Daten wurde im Vergleich mit der „Related Work“ anderer wissenschaftlicher Arbeiten ein Anforderungs-Katalog entwickelt, welcher die Qualität von Design und Implementation überprüfbar macht.

Das Design des Dateisystems besteht aus einem Client-Interface, einem Storage-Manager und Provider-spezifischen Konnektoren, mittels derer Dateisystemoperationen durchgeführt werden können. Zur Absicherung gegen unbefugten Zugriff können Plugins verwendet werden, welche eine Verschlüsselungs-Funktionalität auf Client-Seite bieten. TDFS ist in C# implementiert und auf einen Betrieb unter Windows ausgerichtet, dementsprechend wird die Speicherung der Anmeldedaten verschlüsselt in der Registry vorgenommen, um einen Transfer dieser Daten bzw. deren Dekodierung zu erschweren.

Zur Benutzung des Dateisystems war die Verwendung von „Dokan“, einer FUSE-ähnlichen Dateisystem-Schicht geplant. Es erwies sich als für den Betrieb in TDFS zu instabil, in der Folge wurde mittels der Freigabe des Caches im Netzwerk eine gleichartige Zugriffsmöglichkeit emuliert. Mittels einer Einstellung kann bestimmt werden, bei welcher Anzahl von Storage-Providern ein Dateiteil vorhanden sein muss. Dadurch kann die Ausfallswahrscheinlichkeit in eingeschränktem Maße dem Sicherheitsbedürfnis des Anwenders angepasst werden. Falls nicht vom Benutzer eine Dateisystem-Operation induziert wird, synchronisiert TDFS in einem einstellbaren Intervall die lokalen Daten mit der Cloud und vice versa. Im Fall eines Fehlers beim Herunterladen einer Datei lädt TDFS eine Replika, die bei einem anderen Provider gespeichert ist, und vermindert so das Verlustrisiko für die in der Cloud gespeicherten Daten.

Die erstellten Messungen belegen die Fähigkeit von TDFS, Daten bei mehreren Storage-Providern zu speichern. Dazu wurden die Daten zuerst bei den Anbietern einzeln abgespeichert bzw. von diesen heruntergeladen und die benötigte Zeitdauer gemessen. Ebenfalls wurde das Verhalten des Programmes im RAID-Betrieb überprüft. Der Aufwand für den Upload steigt im vorgestellten sequenziellen Fall linear an. Das Vorliegen eines Fehlerfalles führt bei TDFS zu keiner signifikanten Abweichung bei den Download-Zeiten, diese sind mit dem Normal-Betrieb vergleichbar.

Die Erweiterung der Funktionalität des vorgestellten Programmes ist in mehrfacher Weise möglich, zum Beispiel durch die Erstellung zusätzlicher Konnektoren für den Zugriff auf weitere Storage-Provider. Durch die Messung von Parametern wie Zugriffsgeschwindigkeit, Datentransferraten, Fehler-Raten etc. im Live-Betrieb kann eine Verbesserung des Betriebs-Verhaltens erreicht werden. Eine darauf aufbauende Steuerung des Verhaltens von TDFS könnte beispielsweise die Verteilung von Dateiblöcken bzw. die optimalen Anbieter zum Download von Dateiteilen ermitteln. Sobald die Qualität des Dokan-Projekts ein stabiles Stadium erreicht hat, ist auch die Anbindung der POSIX-Funktionen des Dokan-Interfaces and das Client-Interface von TDFS denkbar.

Literaturverzeichnis

[1] H. Adelsberger and A. Drechsler, "Ausgewählte Aspekte des Cloud-Computing aus einer IT-Management-Perspektive," Universität Duisburg, Tech. Rep., 2010. [Online]. Available: http://www.icb.uni-due.de/fileadmin/ICB/research/research_reports/ICB-Report-No41.pdf

[2] Y. Dong, H. Zhu, J. Peng, F. Wang, M. P. Mesnier, D. Wang, and S. C. Chan, "Rfs: a network file system for mobile devices and the cloud," *Operating Systems Review*, vol. 45, no. 1, pp. 101–111, 2011.

[3] J. Heurix, T. Neubauer, S. Sommer, and A. Tjoa, "SecureCloud: Ein System für die verteilte Datenspeicherung in der Cloud," IFS, Tech. Rep., Okt 2011.

[4] B. Furht and A. Escalante, *Handbook of Cloud Computing*. Springer-Verlag New York Inc, 2010.

[5] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the sun network filesystem," in *Proceedings of the Summer 1985 USENIX Conference*, 1985, pp. 119–130.

[6] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[7] K. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 187–198.

[8] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Tech. Rep., Jul. 2009. [Online]. Available: http://www.csrc.nist.gov/groups/SNS/cloud-computing/

[9] B. Segal, L. Robertson, F. Gagliardi, F. Carminati, and G. Cern, "Grid computing: The European data grid project," in *IEEE Nuclear Science Symposium and Medical Imaging Conference*, 2000, pp. 15–20.

[10] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE'08*. IEEE, 2008, pp. 1–10.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, 2003, pp. 164–177.

[12] Y. Chen, "Service Oriented Architecture," 2006.

[13] G. Alonso and F. Casati, "Web services and service-oriented architectures," in *Proceedings of the international conference on data engineering*, vol. 21. IEEE Computer Society Press; 1998, 2005, p. 1147.

[14] C. Mathas, *SOA intern: Praxiswissen zu Service-orientierten IT-Systemen*. Hanser Verlag, 2008.

[15] W. W. S. A. W. Group, "Web Services Architecture," http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, Feb. 2004. [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[16] A. Perilli, A. Manieri, A. Algom, and C. e. a. Balding, "Cloud computing risk assessment," ENISA, Greece, Tech. Rep., Nov. 2009. [Online]. Available: http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment

[17] A. Silberschatz, P. Galvin, and G. Gagne, *Operating system concepts*. Addison-Wesley New York, 2004.

[18] A. Thomasian and M. Blaum, "Higher reliability redundant disk arrays: Organization, operation, and coding," *ACM Transactions on Storage (TOS)*, vol. 5, no. 3, pp. 1–59, 2009.

[19] Various, "Wikimedia Commons RAID Illustrations," 2011.

[20] J. Luther, C. Vilsbeck, and B. Haluschak, "RAID-Varianten im Überblick," 2011.

[21] E. Levy and A. Silberschatz, "Distributed file systems: Concepts and examples," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 321–374, 1990.

[22] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37. ACM, 2003, pp. 29–43.

[23] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the First USENIX Conference on File and Storage Technologies*. Citeseer, 2002, pp. 231–244.

[24] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network File System (NFS) version 4 Protocol," RFC 3530 (Proposed Standard), Internet Engineering Task Force, April 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3530.txt

[25] R. Thurlow, "RPC: Remote Procedure Call Protocol Specification Version 2," RFC 5531 (Draft Standard), Internet Engineering Task Force, May 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5531.txt

[26] S. Lowe, "Block-level-storage versus file-level-storage - a comparison," 2011.

[27] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran, "Object storage: The future building block for storage systems," in *2nd International IEEE Symposium on Mass Storage Systems and Technologies, Sardinia*. IEEE, 2005, pp. 119–123.

[28] M. Vrable, S. Savage, and G. Voelker, "Cumulus: Filesystem backup to the cloud," *ACM Transactions on Storage (TOS)*, vol. 5, no. 4, pp. 1–28, 2009.

[29] P. Xu, W. Zheng, Y. Wu, X. Huang, and C. Xu, "Enabling cloud storage to support traditional applications," in *The Fifth Annual ChinaGrid Conference*. IEEE, 2010, pp. 167–172.

[30] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[31] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in *Proceedings of the 2008 international workshop on Data-aware distributed computing*. ACM, 2008, pp. 55–64.

[32] D. Le Vie Jr, "Writing Software Requirements Specifications," 2007.

[33] J. Radatz, M. Olson, and S. Campbell, "Mil-std-498," *CrossTalk, STSC, Hill Air Force Base, Utah, US*, vol. 1, p. 9, 1995.

[34] E. Iee, "Ieee recommended practice for software requirements specifications," IEEE, Tech. Rep., 1998. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=720574

[35] I.-S. S. Board, "Ieee recommended practice for software design descriptions," IEEE, Tech. Rep., 1998. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=741934

[36] Amazon.com, "Amazon sdk example code listing: http://aws.amazon.com/sdkfornet/." [Online]. Available: http://aws.amazon.com/sdkfornet/

[37] M. Fowler, "Inversion of control containers and the dependency injection pattern," 2004.

[38] D. Tarrant, T. Brody, and L. Carr, "From the desktop to the cloud: Leveraging hybrid storage architectures in your repository," in *The 4th annual international Open Repositories Conference (or09)*, February 2009. [Online]. Available: http://eprints.ecs.soton.ac.uk/17084/

[39] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "RFC 4880-OpenPGP Message Format," 2007.

[40] J. Broberg, R. Buyya, and Z. Tari, "Metacdn: Harnessing storage clouds for high performance content delivery," *Journal of Network and Computer Applications*, vol. 32, no. 5, pp. 1012–1022, 2009.

[41] M. Johnson, "POIFS Use Cases," 2002-2010.

Appendix

Use Cases

In Anlehnung an die POIFS Use Cases [41] des Apache POI Projektes soll das CSCI folgende Use Cases Unterstützen:

Use-Case-Element	Beschreibung
Akteur	Benutzer
Umfang	TDFS
Abstraktionsebene	Übersicht
Interessenten und Interessen	Benutzer möchte eine Datei lesen TDFS - abstrahiert Dateisystemoperationen Cloud Storage Provider - führt Operationen beim jeweiligen Provider durch
Vorbedingungen	Dateien wurden durch das Use-Case „Auflistung von Dateien / Ordnern“ gelistet.
Minimalerfolgspfad	Keiner
Erfolgspfad	1. Benutzer wählt eine Datei zum Auslesen aus. 2. Benutzer öffnet Datei. 3. TDFS ermittelt Teile der Datei 4. TDFS lädt Teile über die Storage-Provider herunter. 5. TDFS stellt Datei im Speicher bereit. 6. Benutzer liest Datei.
Erweiterungen	5a. Falls verschlüsselt, TDFS entschlüsselt Datei.

Tabelle 1: Use Case Datei lesen

Use-Case-Element	Beschreibung
Akteur	Benutzer
Umfang	TDFS
Abstraktionsebene	Übersicht
Interessenten und Interessen	Benutzer möchte eine Datei schreiben TDFS - abstrahiert Dateisystemoperationen Cloud Storage Provider - führt Operationen beim jeweiligen Provider durch
Vorbedingungen	Dateien wurden durch das Use-Case „Auflistung von Dateien / Ordnern“ gelistet.
Minimalerfolgspfad	Keiner
Erfolgspfad	1. Benutzer wählt eine Datei zum Auslesen aus. 2. Benutzer öffnet Datei. 3. TDFS ermittelt Teile der Datei 4. TDFS lädt Teile über die Storage-Provider herunter. 5. TDFS stellt Datei im Speicher bereit. 6. Benutzer liest Datei. 7. Benutzer modifiziert Datei. 8. Benutzer speichert / schreibt Datei. 9. TDFS übermittelt Daten an Storage Provider.
Erweiterungen	5a. Falls verschlüsselt, TDFS entschlüsselt Datei. 9a. Falls Verschlüsselung verlangt, TDFS verschlüsselt Datei. 9b. Falls Platzverbrauch zu groß, Splitting der Daten.

Tabelle 2: Use Case Datei schreiben

Use-Case-Element	Beschreibung
Akteur	Benutzer
Umfang	TDFS
Abstraktionsebene	Übersicht
Interessenten und Interessen	Benutzer möchte den Laufwerksinhalt bzw. Ordnerinhalt einsehen. TDFS - abstrahiert Dateisystemoperationen Cloud Storage Provider - führt Operationen beim jeweiligen Provider durch
Vorbedingungen	Dateien wurden durch das Use-Case „Auflistung von Dateien / Ordnern“ gelistet.
Minimalerfolgspfad	Keiner
Erfolgspfad	1. Benutzer startet TDFS. 2. TDFS meldet sich bei den Storage-Providern an. 3. TDFS ermittelt die Inhalte der einzelnen Storage Provider. 4. TDFS aggregiert die Liste 5. TDFS stellt Daten am Interface bereit bzw. dar.
Erweiterungen	keine

Tabelle 3: Use Case Dateien/Ordner auflisten

Use-Case-Element	Beschreibung
Akteur	Benutzer
Umfang	TDFS
Abstraktionsebene	Übersicht
Interessenten und Interessen	Benutzer möchte eine Datei löschen TDFS - abstrahiert Dateisystemoperationen Cloud Storage Provider - führt Operationen beim jeweiligen Provider durch
Vorbedingungen	Dateien wurden durch das Use-Case „Auflistung von Dateien / Ordnern“ gelistet.
Minimalerfolgspfad	Keiner
Erfolgspfad	1. Benutzer wählt eine Datei zum Löschen aus. 2. TDFS ermittelt Teile der Datei 3. TDFS löscht die Teile über die Storage-Provider.
Erweiterungen	keine

Tabelle 4: Use Case Datei / Ordner löschen

Use-Case-Element	Beschreibung
Akteur	Benutzer
Umfang	TDFS
Abstraktionsebene	Übersicht
Interessenten und Interessen	Benutzer möchte eine Datei kopieren TDFS - abstrahiert Dateisystemoperationen Cloud Storage Provider - führt Operationen beim jeweiligen Provider durch
Vorbedingungen	Dateien wurden durch das Use-Case „Auflistung von Dateien / Ordnern“ gelistet.
Minimalerfolgspfad	Keiner
Erfolgspfad	1. Benutzer wählt eine Datei zum Kopieren aus. 2. TDFS ermittelt Teile der Datei 3. TDFS lädt Teile über die Storage-Provider herunter. 4. TDFS lädt Teile zu den Einzelnen Storage Providern.
Erweiterungen	3a. Falls möglich wird eine Direkt-Lade-Operation benutzt.

Tabelle 5: Use Case Datei / Ordner kopieren

Use-Case-Element	Beschreibung
Akteur	Benutzer
Umfang	TDFS
Abstraktionsebene	Übersicht
Interessenten und Interessen	Benutze möchte eine Datei umbennenen TDFS - abstrahiert Dateisystemoperationen Cloud Storage Provider - führt Operationen beim jeweiligen Provider durch
Vorbedingungen	Dateien wurden durch das Use-Case „Auflistung von Dateien / Ordnern“ gelistet.
Minimalerfolgspfad	Keiner
Erfolgspfad	1. Benutzer wählt eine Datei zum Umbenennen aus. 2. Benutzer benennt Datei um. 3. TDFS ermittelt Teile der Datei 4. TDFS benennt Teile über die Storage-Provider um.
Erweiterungen	4a. Falls kein umbenennen möglich ist, wird der Use-Case Kopieren benutzt.

Tabelle 6: Use Case Datei umbennenen

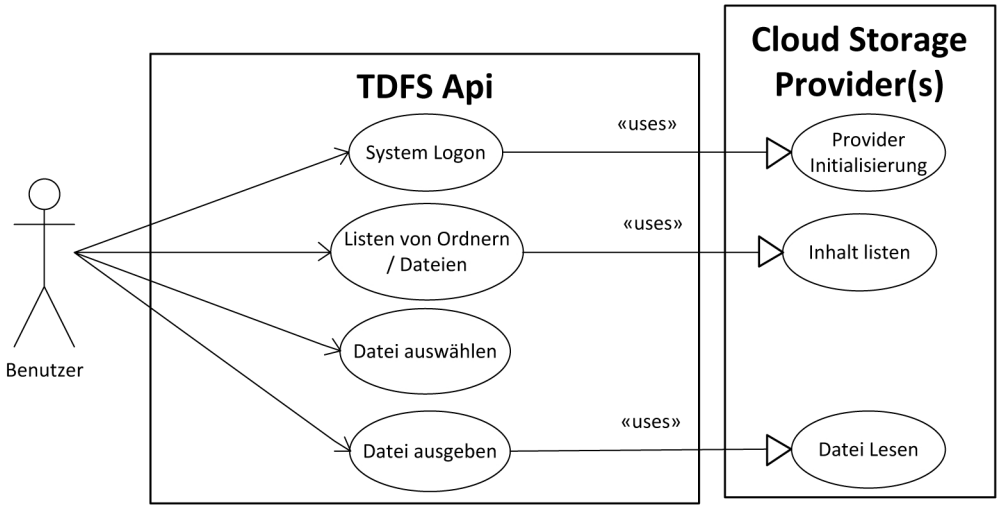

Abbildung 1: Lesen von Dateien

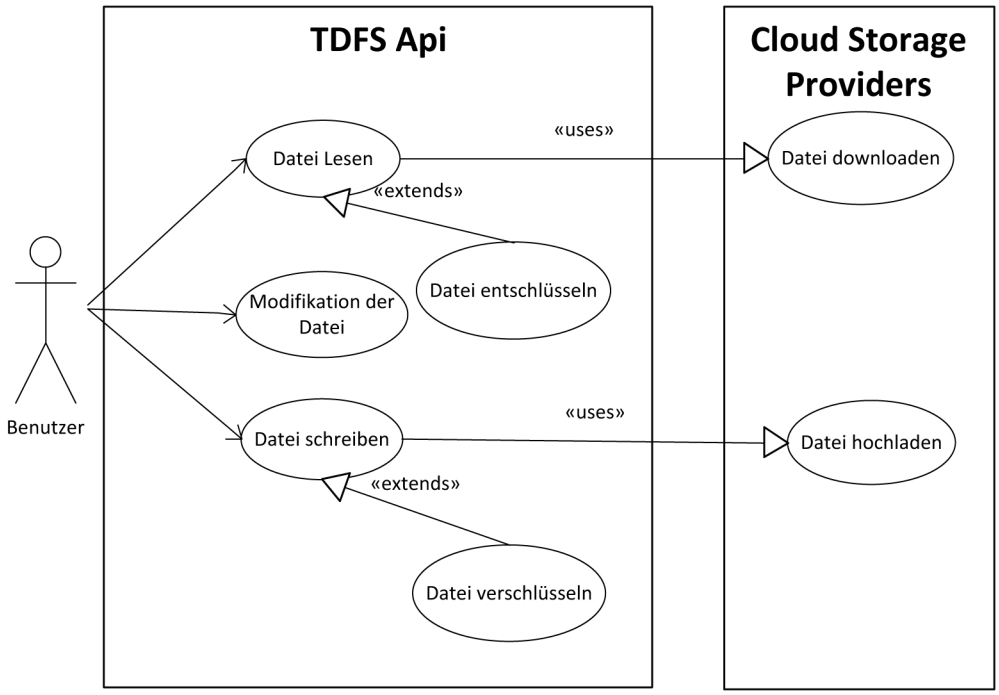

Abbildung 2: Schreiben von Dateien

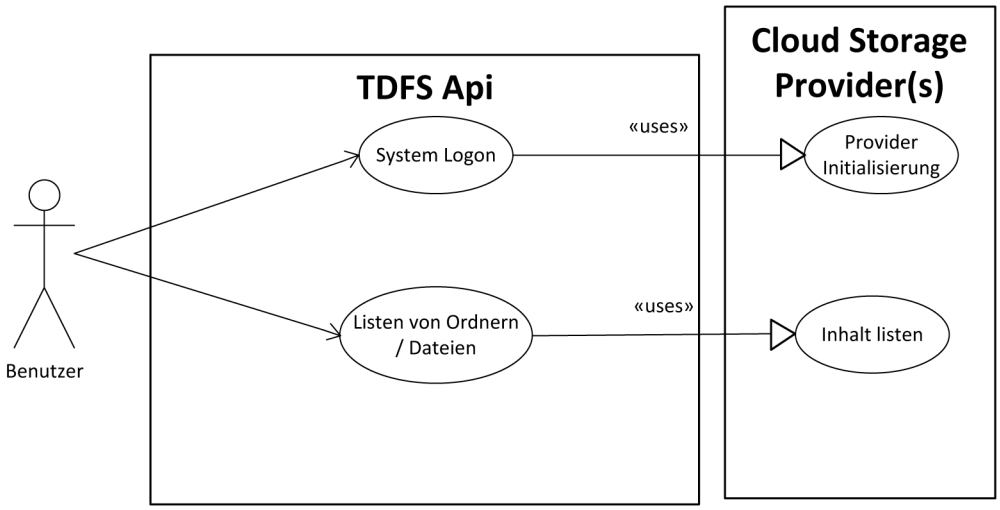

Abbildung 3: Listen von Files / Ordnern

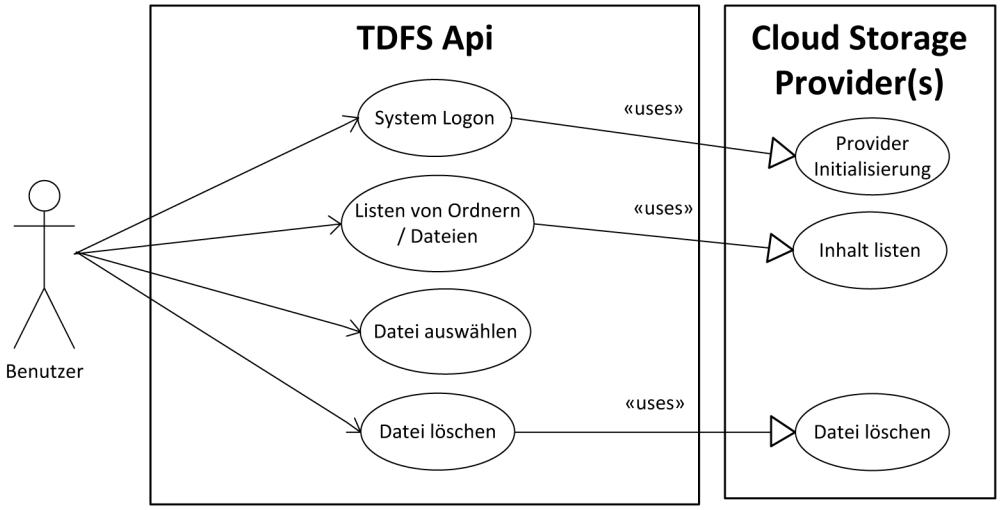

Abbildung 4: Löschen von Dateien / Orndern

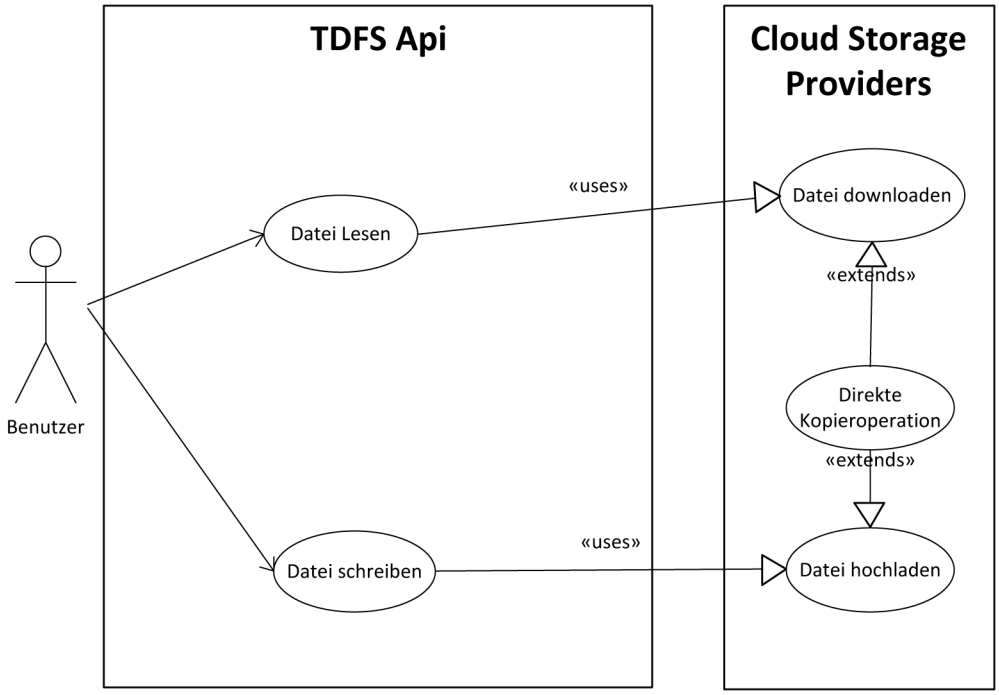

Abbildung 5: Kopieren von Dateien / Ordnern

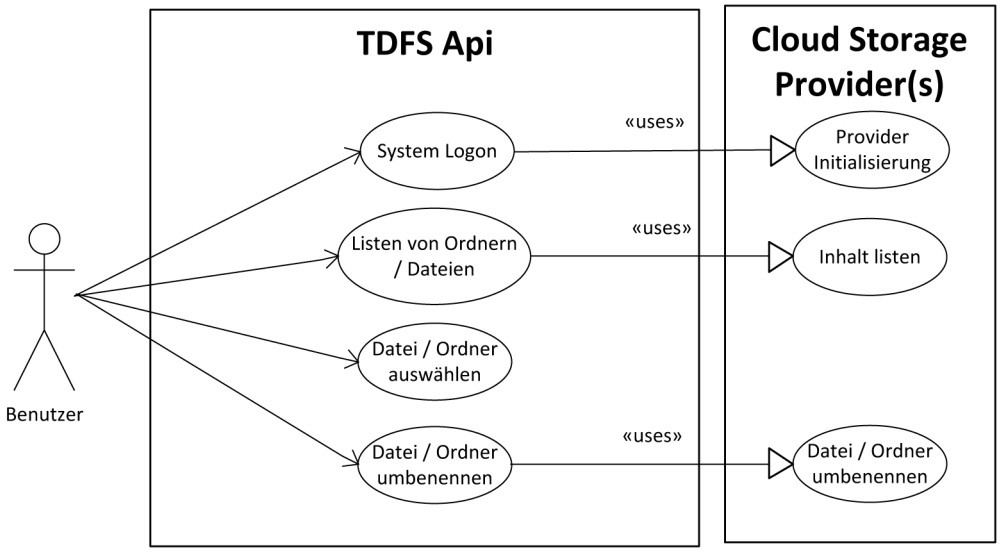

Abbildung 6: Umbenennen von Dateien / Ordnern

Requirement-To-Design-Mapping

Tabelle 7 zeigt welche Requirements es gibt und welche Komponente sie implementiert.

Req.	Storage-Pl.	Split/-Merge	Redundanz	Storage-Cont.	Client-Int.	Encr.	Auth.
R1	X	X	X	X			
R2	X	X		X			
R3	X			X	X		
R4	X		X	X			
R5	X				X		
R6	X	X	X	X	X		
R7					X		
R8				X			
R9				X			
R10					X		
R11	X						
R12	X			X			
R13						X	
R14	X						X
R15						X	
R16							
R17						X	
R18	X				X		
R19	X	X	X	X	X	X	X
R20				X			
R21	X	X	X	X	X	X	X
R22	X			X			
R23					X		
R24				X	X		
R25							
R26							
R27				X			
R28	X	X	X	X	X	X	X
R29	X						
R30	X	X	X	X	X	X	X
R31	X	X	X	X	X	X	X
R32	X	X	X	X	X	X	X

Tabelle 7: Requirements-Design-Matrix

Provider-Funktionalität

Tabelle 8 bis Tabelle 11 zeigen, soweit vorhanden, Informationen zu den Fähigkeiten des Speicherangebots einzelner Dienstanbieter.

Funktion	Read	Write	Open	Close	Dir
4shared	getFileDownloadLink	uploadStartFile	n.v.	n.v.	getDirInfo
Humyo	n.v.	n.v.	n.v.	n.v.	listTeamspace?
Magicvortex	Receive	Send	n.v.	n.v.	List
Amazon S3	yes	ja	ja	ja	ja
Box.Net	yes (REST)	UploadFile	nein	nein	GetFolderStruct.
Dropbox	DownloadFile	UploadFile	Open	Close	GetRoot()
Emc	ReadObject	CreateObject	n.v.	n.v.	ListObject
Google Storage	Retrieve	AddObject	ja	nein	bucket.Objects
Azure	DownloadWebFile	UploadWebFile	IsWebFileExists	nein	ListSubWebFolderF.
Nirvanix	DownloadToLocalFile	PerformSOAPUpload		n.v.	ListFolder()
Mezeo	yes (REST)	ja (REST)	n.v.	n.v.	Shares (REST)
Hotfile	getdirectdownloadlink	nein	ja	nein	nein
Rapidshare	DownloadFile	UploadFile	n.v.	n.v.	ListFiles
Wuala	GetRequest	PutRequest	n.v.	n.v.	GetDirectoryList
Swapdrive	n.v.	n.v.	n.v.	n.v.	n.v.
Barracuda Drive	n.v.	n.v.	n.v.	n.v.	n.v.
FTP	GetFile	PutFile	FileExists	n.v.	GetFiles/-Directories
WebDav	Windows-Webdav? Yes	ja	ja	ja	ja
Netzwerk Share	Windows / .Net API, yes	ja	ja	ja	ja
Idisk	Windows-Webdav? Yes	ja	ja	ja	ja

Tabelle 8: Provider-Fähigkeiten - CRUD

Provider / Functionality	Rename	Move	Copy	Delete	Notifications
4shared	RenameFile/Folder	nein	nein	DeleteFile/Folder	nein
Humyo	nein	nein	nein	nein	nein
Magicvortex	nein	nein	nein	nein	nein
Amazon S3	n.v.	n.v.	yes	ja	ja
Box.Net	RenameObject	n.v.	CopyObject	DeleteObject	nein
Dropbox	RenameFileSystemEntry	MoveFileSystemEntry	/fileops/copy	DeleteFileSystemEntry	nein
Emc	Rename	nein	nein	DeleteObject	nein
Google Storage	gsutil is able to	gsutil is able to	PUT Object	Delete	n.v.
Windows Skydrive / Azure	RenameWebFile/Folder	MoveWebFile/Folder	CopyWebFile	DeleteWebFile/Folder	up progress
Nirvanix	RenameFile/Folder	MoveFiles()	CopyFiles/Folder	DeleteFiles/Folder	nein
Mezeo	nein	nein	nein	ja	nein
Hotfile	renamefolder	nein	nein	deletefolder	nein
Rapidshare	RenameFile	MoveRealFolder	nein	DeleteFiles	nein
Wuala	nein	nein	nein	nein	nein
Swapdrive	n.v.	n.v.	n.v.	n.v.	n.v.
Barracuda Drive	n.v.	n.v.	n.v.	n.v.	n.v.
FTP	Rename	Rename	nein	RemoveFile/Directory	nein
WebDav	ja	ja	ja	ja	nein
Netzwerk Share	ja	ja	ja	ja	nein
Idisk	ja	ja	ja	ja	nein

Tabelle 9: Provider-Fähigkeiten - weitere Dateisystemoperationen

Provider / Functionality	Versioning	ACLs	Metadata	FileAttributes	Multipart
4shared	no (but history)	setFolderSharingProperties	ja	nein	nein
Humyo	nein	setshare	nein	nein	nein
Magicvortex	nein	nein	nein	nein	nein
Amazon S3	ja	ja	ja	ja	ja
Box.Net	nein	Private/PublicShare?	SetDescription	n.v.	nein
Dropbox	nein	nein	GetMetadata	n.v.	nein
Emc	ListVersions	Acl Class	Get/Del/SetUserMetadata	nein	ja
Google Storage	GS no / S3 yes?	ACL/AClSave	GetObjectHead	no	nein
Windows Skydrive / Azure	nein	nein	ChangeWebFile	nein	nein
Nirvanix	nein	n.v.	ja	ja	nein
Mezeo	nein	ja	ja	nein	ja
Hotfile	nein	nein	nein	nein	nein
Rapidshare	nein	nein	nein	nein	nein
Wuala	nein	nein	GetMetadata	nein	nein
Swapdrive	n.v.	n.v.	n.v.	n.v.	n.v.
Barracuda Drive	n.v.	n.v.	n.v.	n.v.	n.v.
FTP	nein	user	nein	CHMOD / STAT	n.v.
WebDav	nein	ja	ja	ja	nein
Netzwerk Share	nein	ja	ja	ja	nein
Idisk	nein	ja	ja	ja	nein

Tabelle 10: Provider-Fähigkeiten - Metadaten

Provider / Funktionalität	Authentifizierung	Encr.(http / intern)	API	Typ	Lizenz
4shared	shared secret	n.v.	4Shared SOAP API	Soap	unbekannt
Humyo	shared secret	SSL / nein	Humyo Storage API	Soap	unbekannt
Magicvortex	shared secret	n.v.	HTTP API	Soap	unbekannt
Amazon S3	n.v.	SSL / nein	Amazon S3 SDK	SDK	Apache V2
Box.Net	n.v.	SSL / AES-256 (Enterprise)	Boxsync.net / REST	SDK / Lib	MPL
Dropbox	Oauth	SSL / AES-256	Sharpbox / Dropnet	SDK / Lib	MIT License
Emc	shared secret	SSL / nein	Atmos_Dotnet	SDK / Lib	BSD License
Google Storage	shared secret	SSL / nein	Sharpg	SDK / Lib	unbekannt
Windows Skydrive / Azure	shared secret	SSL / nein	Skydrive.Net Api Client	SDK / Lib	GPLv2
Nirvanix	shared secret	n.v.	Nirvanix C# SDK	SDK	unbekannt
Mezeo	n.v.	n.v.	C# Bsp (developer.mezeo.com)	Rest	unbekannt
Hotfile	shared secret	n.v.	http://api.hotfile.com	Rest	unbekannt
Rapidshare	shared secret	n.v.	Rapidshare API (sourceforge)	Rest	unbekannt
Wuala	shared secret	n.v.	WUALA API	Rest	unbekannt
Swapdrive	n.v.	n.v.	kommerziell	unbekannt	kommerziell
Barracuda Drive	n.v.	n.v.	kommerziell	unbekannt	kommerziell
FTP	shared secret	SFTP (transport)	C# FTPlib	Lib	MIT License
WebDav	shared secret	https (opt.)	Neon (in C)	Lib	GPL
Netzwerk Share	Windows	nein	Win / .NET API	Lib	unbekannt
Idisk	shared secret	nein	Webdav	Lib	GPL

Tabelle 11: Provider-Fähigkeiten - Sicherheit & Implementierungsinformationen

Curriculum Vitae

Stefan Sommer

Personalia:	Geburtstag:	3. August 1981, Klosterneuburg, Österreich Österreichische Staatsbürgerschaft
	Addresse:	Johann-Strauss-Gasse 1/1/2 3424 Muckendorf-Wipfing
	e-mail:	stefan.sommer@aestas.at
	Sprachen:	Deutsch, Englisch, Russisch
Ausbildung	1987 - 1991	Volksschule Zeiselmauer
	1991 - 1999	Bundesgymnasium Tulln
	2000 - 2003	Bakkalaureats-Studium Wirtschaftsinformatik Schwerpunkt: Software Engineering
	seit 2003	Magisterstudium Wirtschafstinformatik Schwerpunkt: Advanced Software Engineering
Beruf	2001 -2008	Beschäftigung bei Versicherungen (EDV)
	seit 2008	Development System Engineer PDTS AG, Frequentis Gruppe
Interessen		Musik, Sport, IT