Josef Spillner

Untersuchungen zur Risikominimierungstechnik Stealth Computing für verteilte datenverarbeitende Software-Anwendungen mit nutzerkontrollierbar zusicherbaren Eigenschaften

– Habilitationsschrift –

zur Erlangung der Lehrbefähigung für das Fachgebiet Informatik,
Fakultät Informatik, 28. April/18. Dezember 2015

Technische Universität Dresden

Inhaltsverzeichnis

Kapitel 1
Problemdarstellung

Zusammenfassung Software wird zunehmend in Form von verteilten Komponenten und Diensten sowie Anwendungen, welche ihrerseits auf Diensten von Drittanbietern aufbauen, bereitgestellt. Sie kann dadurch von mehreren Geräten aus komfortabel genutzt werden, unter Einsatz von Cloud-Diensten elastisch mit der Verarbeitungslast skalieren und nutzungsabhängig berechnet werden. Diese Vorteile hängen jedoch von einer stabilen und vertrauenswürdigen Infrastrukturgrundlage ab, welche im Internet und teilweise selbst in lokalen Netzen in den wenigsten Fällen gegeben ist. Somit weisen die ausgeführten Dienste und Anwendungen entweder keine oder anwendungsspezifische, oft nachträglich implementierte und unvollständige Schutzmechanismen auf, die zudem wenig portabel und nur zu einem geringen Grad zusicherbar sind. Dieses Kapitel führt in das Problemfeld ein und motiviert eine systematische Herangehensweise an eine zwar eingeschränkte, aber dafür praktisch einsetzbare und für den Anwender kontrollierbare Lösung des Problems durch inhärente Laufzeit-Risikominimierung als Bestandteil des Entwicklungsprozesses für solche Anwendungen.

1.1 Einführung

Die ubiquitäre Nutzung von Diensten und Datenübertragungsprotokollen durchdringt zunehmend alle Anwendungsdomänen und Lebensbereiche [Lam15]. Beobachten lässt sich der stetige Zuwachs vor allem bei Anwendungen, die über das Internet weit verbreitet sind und kritische Massen an Benutzern erreichen, auch trotz gelegentlicher Popularitätseinbußen einzelner Angebote [BK14b].

Einhergehend mit der Dienstorientierung werden Anwendungen aus unterschiedlichen Gründen zumeist in mehrere Anwendungsbestandteile und Anwendungsdienste aufgeteilt und auf eine Vielzahl von Plattformen verteilt [LAS+15, BDR14]. Dadurch wird dem Benutzer die effektive Kontrolle über den Umgang mit den verarbeiteten Daten entzogen, wodurch ohnehin vorhandene Risiken der Datenverarbeitung erhöht werden. Anwendungsentwickler entgegnen dieser Einschränkung mit

vereinzelten zusätzlichen Sicherheitsfunktionen [GKL15], mitunter jedoch auch mit blindem Vertrauen in die Sicherheit und Robustheit der Infrastruktur mit optionaler nachgelagerter Risikosymptombehandlung [RC15, FGM13]. Zur Verbesserung der Lage wird, wie in Abbildung 1.1 gezeigt, für jede Anwendungsfunktionalität ein anwendungsbezogener sicherer Infrastrukturbereich über die Vertrauensdomäne hinweg benötigt.

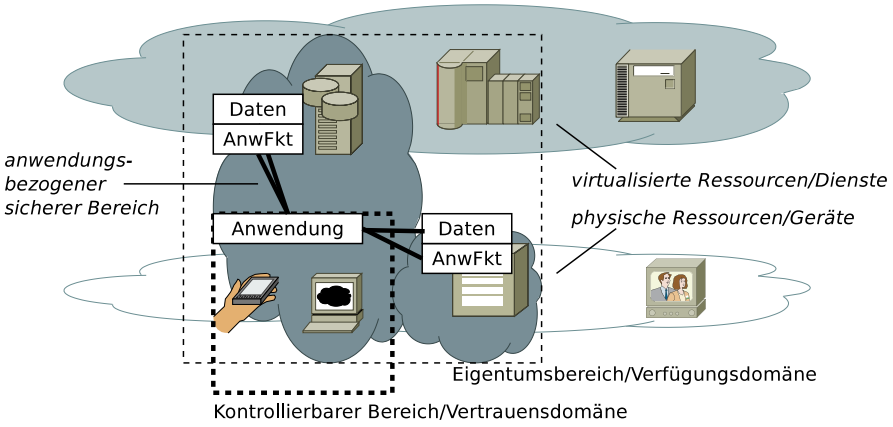

Abb. 1.1 Topologische Darstellung nutzerkontrollierbar sicherer verteilter Anwendungen

Diese Arbeit widmet sich den Prinzipien der Konstruktion sicherer Bereiche mit zusicherbaren Eigenschaften zur Minimierung der Risiken, denen Anwendungen und Anwender ausgesetzt sind. Dazu werden zuerst grundlegende Betrachtungen zu verteilten Architekturen, speziell zu Cloud- und Dienstarchitekturen sowie darauf verteilten Anwendungen, vorgebracht. Anschließend wird das zu lösende Problem definiert, eingeordnet und abgegrenzt, so dass Anspruch und Umfang der Arbeit mit dem Abschluss des Kapitels feststehen.

1.2 Grundlegende Betrachtungen

Zur Hinführung auf die Kernproblematik ist es erforderlich, zuerst die Dienststrukturen von Cloud-Angeboten mit ihren funktionalen und nichtfunktionalen Eigenheiten zu verstehen.

Zum Dienstspektrum zählen zum einen IT-gestützte Dienstleistungen [Spi11a], zum anderen auch vollständig IT-basierte Dienste [FGJ+11, SKS12], die in unterschiedliche Dienstklassen bis hin zu Spezialisierungen wie *Forensic-as-a-Service* [DdQ11], *Sensing-as-a-Service* [PZCG14], *CDN-as-a-Service* [DHR+14] oder *Data-as-a-Service* [VPT+12] eingeteilt sind. Das sich aus solchen Dienstklassen ergebende Dienstspektrum wird als *Everything-as-a-Service* (XaaS) zusammenge-

fasst [KSHD13, SS13a] und zumeist klassenspezifisch, mitunter jedoch auch generisch klassenübergreifend verwaltet [SSZ13, SZS13]. Orthogonal dazu existieren viele Dienstdomänen, welche die Anwendungsfelder beschreiben, in denen ein Dienst genutzt wird. Beispiele dafür sind Logistik, industrielle Datensammlung und -aufbereitung, Kommunikation, soziale Kontakte, persönliche Datenverwaltung, Steuererklärung und Auftragserteilung in Geschäftsprozessen [BBF+11, SS13a, PP15, uRLW+15, JL13].

Viele der Dienstklassen und Dienstdomänen werden in funktionale Schichten eingeordnet, die wiederum als geschichtete Modelle zusammengefasst sind. Die Referenzmodelle zum Cloud Computing sind typische Vertreter einer solchen Einordnung [MG09, KSHD13]. Das Anwendungs- und Forschungsfeld Cloud Computing hat sich dabei aus unterschiedlichen Ursprüngen wie den dienstorientierten Architekturen bzw. Service Computing, dem ausgelagerten Betrieb von Anwendungen (Hosting bzw. Application Service Provisioning), dem Cluster-, Grid- und High-Performance-Computing, der Virtualisierung, dem Volunteer Computing, netzwerkintegrierten Diensten sowie der Softwaretechnologie und der Ökonomie entwickelt [CR12, CBA+06, FZRL08, BYV+09]. Cloud-Anwendungen laufen in dedizierten Rechenzentren, in isolierten oder virtualisierten Teilen derselben, verteilt über personenübergreifende gleichberechtigt vernetzte Rechner (Peer-to-Peer und Friend-to-Friend Cloud [BM14a], Nebula Computing [ROCW14], Edge Cloud, Mobile Cloud), verteilt über die Geräte eines Benutzers (Personal Cloud [SZS13]) oder verteilt über eingebettete Systeme (Fog Computing [YMSG+14]).

Die Cloud-Referenzmodelle enthalten als Spezialisierung des XaaS-Modells zumeist drei Klassen: Softwareanwendungen als Dienst (*Software-as-a-Service*, SaaS), Plattformbestandteile als Dienst (*Platform-as-a-Service*, PaaS) und Infrastrukturressourcen als Dienst (*Infrastructure-as-a-Service*, IaaS). Auf der Anwendungsebene laufen Webanwendungen auf Webservern, automatisierte Prozesse auf Workflowservern oder grafische Softwareanwendungen auf Endgeräten unter Benutzerkontrolle wie Mobiltelefonen, Notebooks und Personalcomputern. Sollen sie cloudfähig sein, so werden sie als SaaS angeboten, wobei auf dem Endgerät ein Zugang entweder als service-spezifischer Client (Service-Frontend) oder als generische Eingabe-Ausgabe-Übertragung durch Remote-Desktop-Protokolle verbleibt. Aufgrund der Schichtung sind SaaS-Anwendungen, deren sichere und robuste Nutzung aufgrund von risikominimierenden Entwurfsentscheidungen im Entwicklungsprozess Gegenstand dieser Arbeit ist, am stärksten von den darunter liegenden Plattform- und Infrastrukturdiensten abhängig. In dieser Arbeit werden ausschließlich XaaS-Dienste betrachtet, welche im Sinne der Web-Service-Definitionen bzw. des Programmable-Web-Ansatzes eine uniforme Schnittstelle und eine uniforme Beschreibung aufweisen [ACKM04, PTDL08, BGF11], auch wenn letztere oftmals nicht vorhanden oder von niedriger Qualität ist und somit manuell ergänzt werden muss [Spi10]. Weitergehende Betrachtungen, etwa die Überführung einer aus Sicht des automatisierten Zugriffs fragilen Interaktion mit einer Webseite in einen Web Service [SPW+12], sind allerdings aus Gründen der Praktikabilität im Validierungskapitel 6 ergänzend enthalten.

Die Softwaredienste nutzen dabei Plattformdienste, welche bereits im Entwicklungsprozess der Anwendungen festgelegt worden sind, zur Erzielung einer skalierbaren und robusten Teilfunktionalität [SBS11, Spi11b]. Beispiele für Funktionen, die durch explizit von Anwendungen angesprochene Plattformdienste gekapselt werden, sind Datenbanken, zeitabhängige Aufrufe, Build-Systeme für die Softwareentwicklung, Ereignisverarbeitungen, Nachrichtenwarteschlangen, Nachrichtenverteilsysteme, Dienstkomposition und Anwendungsintegration. Diese Funktionen entsprechen somit zu weiten Teilen dem älteren Konzept einer verteilten dienstorientierten Middleware [AJM12, Spi10, Spi09]. Teilweise sind die Funktionen jedoch explizit ressourcengebunden, insbesondere für die Ausführung von Anwendungen und Anwendungsdiensten sowie für die Übertragung und Speicherung von Daten. In diesem Fall greifen die Plattformdienste in einer Cloud ihrerseits wiederum auf ressourcenkapselnde Infrastrukturdienste zurück, um eine elastische Skalierung und eine bedarfsgerechte Zuteilung der Ressourcen zu erreichen. Darin werden virtuelle Maschinen, teilweise auf mehreren Ebenen [SBBS12a, SCB+14, SBBS12b, SCB+13], sowie Container ausgeführt. Manche dieser Funktionen werden zudem als kombinierte *Stacks* angeboten [YNMT15, ST10], mit denen eigene, vornehmlich nichtöffentliche Cloud-Umgebungen konstruiert werden können. Weitere Plattformdienste, die meist in einer auf Entwickler und Dienstanbieter zielenden Dienstplattform zusammengefasst sind und eher nebenläufig zur Anwendung im Hintergrund arbeiten, stellen eine Ausführungsumgebung für Dienste unterschiedlicher Implementierungstechniken bereit, prüfen die Zugriffe, übernehmen die aufrufabhängige Abrechnung (*pay-as-you-go*), skalieren die Dienste nach Bedarf (*on-demand*) und bieten Schnittstellen zur Einbringung neuer Diente an (*deploy-to-market*).

Abbildung 1.2 beinhaltet die vereinfachte Darstellung einer Cloud-Architektur mit den Dienstschichten SaaS, PaaS und IaaS und den genannten Funktionen, Abhängigkeiten und Zugriffsmöglichkeiten. Anhand dieser Abbildung sollen nun die Herausforderungen in der Betrachtung zusicherbarer Eigenschaften erläutert werden.

Analog zu Softwarebibliotheken weisen Dienste sowohl eine wohldefinierte Schnittstelle als auch eine davon getrennt zu betrachtende Implementierung auf. Die Beschreibung der Schnittstelle beschränkt sich jedoch in den meisten Fällen auf eine funktionale Sicht, ohne wichtige nichtfunktionale Parameter zu spezifizieren. Diese sind zur Laufzeit bereits bei Nutzung eines Dienstes demzufolge schwer vorhersagbar oder gar zusicherbar. Bei Nutzung mehrerer unabhängiger oder teilweise voneinander abhängiger Dienste fehlt jegliche Einschätzbarkeit des Gesamtverhaltens insbesondere im Hinblick auf Risiken, denen die Anwendung und je nach Übertragung persönlicher Daten zudem der Anwender bei zunehmender Dienstnutzung auch vermehrt ausgesetzt ist. Die nichtfunktionalen Eigenschaften (NFE) betreffen teilweise das gesamte Dienstspektrum, teilweise aber auch nur eine Auswahl an Domänen und Klassen. Beispiele sind die Antwortzeit, die durchschnittliche Verfügbarkeit, der Preis, die Kapazität eines Speicherdienstes, die Aktualität eines Nachrichtendienstes, die Treffsicherheit einer Partnersuchanwendung oder der Durchsatz eines integrierten Routing- und Switchingdienstes für Netzwerknachrichten.

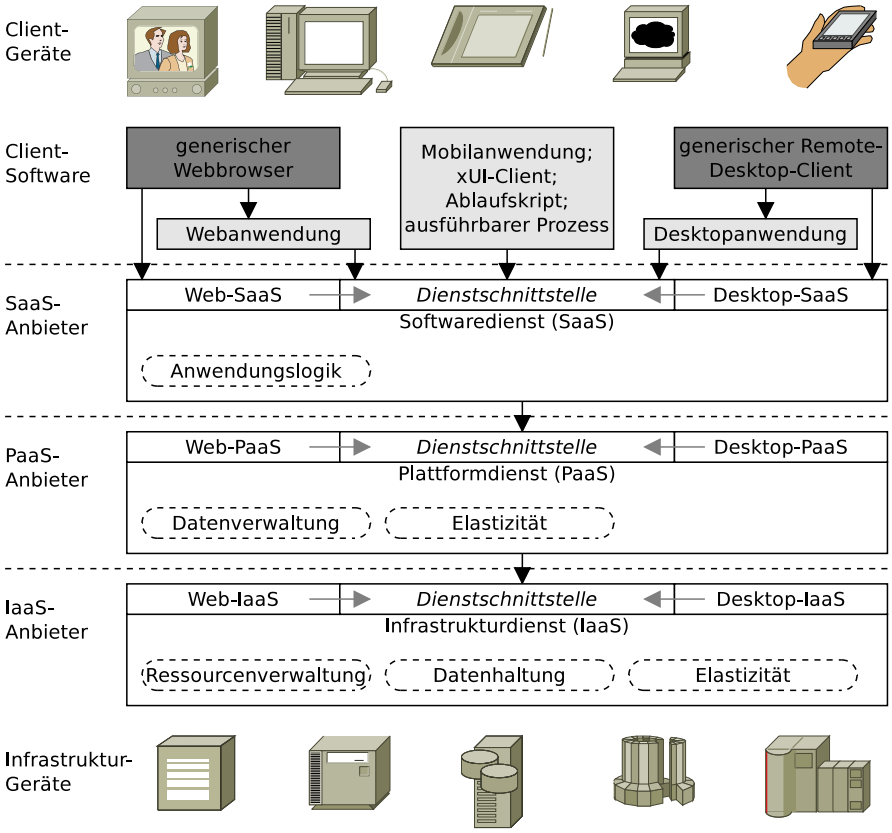

Abb. 1.2 Cloud-Schichtenarchitektur mit Zugriffsmöglichkeiten und Abhängigkeitsbeziehungen

Die nichtfunktionalen Eigenschaften lassen sich unterschiedlich klassifizieren und ordnen, beispielsweise über Taxonomien, Ontologien oder Qualitätsmodelle [SS13a, TPH+14, VBS11]. Es lässt sich aber über die Ordnungsrelation der Eigenschaften auch vereinfachend eine Zweiteilung in offensive und defensive Eigenschaften bzw. Eigenschaftsausrichtungen erzielen. Defensiv bedeutet in diesem Kontext, in Analogie zum Mantra *never change a running system*, dass bei einer einmal erzielten Funktionsgüte diese zumindest immer beibehalten werden soll. Beispiele dafür sind die Zuverlässigkeit, Verfügbarkeit und Verlässlichkeit, die Gewährung von Sicherheit und Datenschutz, die Stabilität und die Robustheit. Offensiv heißt im Gegensatz, dass neue Entwicklungen notfalls auf Kosten der Dienstgüte einbezogen werden sollen, beispielsweise eine erhöhte Skalierbarkeit als Beitrag zum Geschäftswachstum, höhere Performance, mehr Speicherplatz und erweiterte Funktionsmerkmale. Abwägend können weitere NFE wie die Energieeffizienz der Dienstausführung oder deren Preis der einen oder anderen Ausrichtung zugeordnet werden. Allgemein gilt, dass Abwägungen (*trade-offs*) und relative Priorisierungen

nötig sind, da sich beispielsweise die Verfügbarkeit mit hohem Aufwand asymptotisch steigern lässt, der Zugewinn an Verfügbarkeit aber recht schnell in keinem günstigen Verhältnis mehr zum entstehenden höheren Preis steht [YYC13].

Die Nichtbeachtung der nichtfunktionalen Ziele führt häufig genug zu einer auf die Funktion reduzierten *good-enough*-Integration, deren Entwicklung als abgeschlossen angesehen wird, wenn die reine Funktion erfüllt ist. Die Gestaltung und Dokumentation von Bibliotheken, Frameworks und Werkzeugen zur Dienstintegration, oftmals als *software development kit* (SDK) direkt vom Dienstanbieter bereitgestellt, trägt in den meisten Fällen nicht zu einer Verbesserung bei. Aufgrund von Qualitätsdefiziten, insbesondere Sicherheitsdefiziten, in der *good-enough*-Integration von Diensten in Anwendungen gemäß derzeitiger Softwarearchitekturen und -entwicklungsmethoden stellt sich die vorliegende Arbeit der Herausforderung, durch ein methodisches Vorgehen die Abwägungen zwischen mehreren nichtfunktionalen Eigenschaften einschließlich der Sicherheitseigenschaften derartiger datenverarbeitender Anwendungen und Anwendungsdienste flexibel, transparent und kontrollierbar zu gestalten. Dazu wird im Folgenden eine Problemdefinition erarbeitet. Anschließend wird eine Einordnung der Arbeit in laufende Forschungsströmungen im Bereich verteilte Systeme sowie Service- und Cloud Computing gegeben und motiviert, das Prinzip des Schutzes und der Risikominimierung von Anwendungen und Anwendern in nicht vertrauenswürdigen und nicht vollständig kontrollierbaren Umgebungen als *Stealth Computing* im Sinne eines neuen Datenverarbeitungsparadigmas zu bezeichnen.

1.3 Problemdefinition

Die Nutzung von ressourcenbezogenen Diensten auf nicht vom Nutzer kontrollierbaren Ressourcen und damit einhergehend die Akzeptanz des Cloud-Computing-Modells erfordert die Zusicherung von Dienstfunktionen und nichtfunktionalen Eigenschaften gleichermaßen. Im Fokus dieser Arbeit stehen die defensiven NFE, insbesondere die Sicherheit und Stabilität einer Cloud-Anwendung über einen langen Zeitraum bei gleichzeitiger Veränderlichkeit der äußeren Rahmenbedingungen einschließlich der Verfügbarkeit von Diensten. In der Praxis sind genau diese Eigenschaften das größte Hemmnis für den großflächigen Einsatz von Cloud-Diensten [PB10, WLMS14].

Zur Gewährleistung der generischen und dienstspezifischen technischen Eigenschaften von Cloud-Diensten werden von den jeweiligen Anbietern mehrere Verfahren eingesetzt. Beispielsweise werden Daten, die von den Diensten gehalten werden, verschlüsselt und ausfallsicher auf unabhängige Speicherorte repliziert. Aufrufschnittstellen werden ebenfalls mehrfach angeboten und per Lastverteilung oder Fehlerrückfall hochverfügbar gestaltet. Die Dienstimplementierung setzt auf vollredundante Hardwarelösungen auf [VN10]. Ergänzt werden die technischen Eigenschaften um nichttechnische Zusicherungen, zumeist nicht verhandelbare Dienstgütevereinbarungen (*service level agreements*, SLAs) [SIS13], Garantien, Zertifikate,

Prüfsiegel und um allgemeine Popularität und Reputation [TPH+14], die jedoch allesamt nicht Grundlage für eine systematische Anbieterwahl sein können oder sollten und demnach in dieser Arbeit auch nicht berücksichtigt werden.

Dennoch bleibt das Problem bestehen, dass die Qualität einzelner Dienste trotz Vorsichtsmaßnahmen oder aufgrund manipulativer Eingriffe stets unter dem Maximum erreichbarer Dienstgüte bleibt. Die Ursachen dafür, die im Abschnitt 2.1 genauer analysiert werden, umfassen technische Probleme, aktive Angriffe, wirtschaftliche Entscheidungen, menschliches Fehlverhalten und äußere Einflüsse auf den Betrieb von Rechenzentren und Verbindungsnetzen. Nur durch die gezielte Kombination mehrerer voneinander unabhängiger Dienstaufrufe in einem Integrationspunkt, der sich unter weitgehender oder gar vollständiger Kontrolle der dienstaufrufenden Software befindet, sowie durch die a-priori-Transformation von Daten lässt sich die Qualität über das Maß der von den jeweiligen Anbietern gebotenen Dienstgüte hinaus steigern. Insbesondere gilt dies für die Robustheit, Zuverlässigkeit, Verlässlichkeit und Sicherheit von Diensten, sowie teilweise auch für die Antwortzeit.

Durch einzelne Techniken ist dies bereits heute vereinzelt in Anwendungen, Anwendungsdiensten und Frameworks realisiert [YPLL14, APST12]. Auch existieren mehrseitig einsetzbare Monitoringlösungen durch vertrauenswürdige Dritte, die Schutzzielverletzungen zumindest erkennen können [NSHS14]. Ein generisches Vorgehen zum Entwurf, zur Entwicklung und zur Laufzeitkonfiguration der Anwendungen bezogen auf eine abgewogene und priorisierte Menge an zusicherbaren Ziel-NFE im Kontext nicht vertrauenswürdiger Multi-Cloud-Umgebungen ist allerdings noch nicht umfassend erforscht oder gar umgesetzt worden. Dieses Ziel erfordert eine neuartige Form der Anwendungskonstruktion mit datentransformierenden und dienstbündelnden Frameworks basierend auf XaaS-Schnittstellen, welche die langfristige Evolution der Dienste und ihrer dynamischen Qualitätscharakteristiken sowie die Eigenschaften der an die Dienste übertragenen Daten berücksichtigt.

Das vorliegende Werk stellt algorithmische und systemische Ansätze vor, die zur Dienstkombination und zur darauf aufbauenden Entwicklung von Anwendungen mit zusicherbaren Eigenschaften genutzt werden können, und bewertet diese anhand von Implementierungs- und Experimentierergebnissen. Es trägt damit perspektivisch dazu bei, dass zukünftige Cloud-Anwender im privaten, geschäftlichen, institutionellen oder behördlichen Umfeld sich geringeren Risiken aussetzen, in einem höheren Maß die Kontrolle über ihre datenbezogenen Aktivitäten behalten und im Rahmen dieser Aktivitäten verstärkt ihr Recht auf informationelle Selbstbestimmung ausüben können.

1.4 Einordnung und Abgrenzung

Die vorliegende Arbeit tangiert unter dem eher breit aufgefassten Begriff der Cloud-Anwendung mehrere wissenschaftliche Arbeitsgebiete. Sie bezieht sich auf die herkömmlichen Verfahren der Entwicklung von dienstnutzenden Anwendungen und

deren Migration in die Cloud oder auf cloud-gebundene Endgeräte und stellt alternative Ansätze zur nativen Cloud-Anwendungsentwicklung vor, welche den Migrationsschritt in seiner Komplexität reduzieren. Ferner wird die Sicherheit, Robustheit, Fehlertoleranz und Adaptivität von Diensten und Anwendungen angesprochen, wobei jedes dieser Themen für sich genommen eine Untersuchung wert ist und dies auch jeweils in dedizierten wissenschaftlichen Communities erfolgt. Für den Erfolg dieser Arbeit ist jedoch eine holistische Sicht auf diese Eigenschaften notwendig. Schließlich liegt es in der Natur der datenverarbeitenden Anwendungen in der Cloud, dass Themen wie Algorithmen und Datenstrukturen, Informationstheorie und Kanalkodierung, Cloud-Datenbanken, Data Management, Ereignisverarbeitung, Cloud-Migration und Software-Entwicklung ebenfalls betrachtet werden.

Die Arbeit erhebt nicht den Anspruch, insbesondere auf der Ebene der prototypischen Umsetzung alle Facetten der genannten Themen abzudecken. Auch bei Berücksichtigung der vorgestellten Verfahren verbleiben Risiken hinsichtlich der Sicherheit, Robustheit und anderer Eigenschaften entwickelter Anwendungen. Der Anspruch umfasst dennoch die Erläuterung der Notwendigkeit der Einführung einer dienstübergreifenden und anwendungsintegrierten Schutzschicht anhand praktisch lauffähiger und nachvollziehbarer Beispielanwendungen und komplexer Szenarien.

Zum besseren Verständnis seien an dieser Stelle noch die wesentlichen bereits genutzten Begriffe als Definition wiedergegeben. Während die Literatur mehrere Ausprägungen des Cloud-Begriffs kennt, wird in dieser Arbeit eine auf die Problematik bezogene Definition aus Sicht des Anwenders gewählt, welcher sich nachrangig eine technische Sicht anschließt. Eine Cloud ist demnach eine stets zur Verfügung stehende, sichere Bereitstellung von Ressourcen auf beliebige Geräte. Technisch wird eine Cloud als eine Menge an Ressourcendiensten aufgefasst, welche sich durch ihre einheitliche Beschreibung und Aufrufkonvention ohne Mehraufwand in die Software des Anwenders integrieren lassen, gegebenenfalls ergänzt um lokale Ressourcen ohne Dienstcharakter, aber dennoch mit einheitlichen Schnittstellen. Über die Aufrufe der Schnittstellen werden kodierte Daten und Berechungsinstruktionen übergeben, welche Gegenstand des Kapitels 3 sind. Dadurch entstehen native oder hybride Cloud-Anwendungen und -Anwendungsdienste, welche im Kapitel 4 hinsichtlich ihrer Architektur und ihrer Laufzeiteigenschaften und Zusicherbarkeiten beschrieben werden. Bei Aktivierung der Schutzschicht werden sie zu Stealth-Anwendungen erweitert, die datenverarbeitungsbezogene Risiken in der Cloud zwar nicht vollständig eliminieren, jedoch zu weiten Teilen reduzieren.

Kapitel 2
Vorgehensweise und Problemlösungsmethodik

Zusammenfassung Zur Vermeidung von Risiken bei der Verlagerung von Daten und Software in die Cloud und beim Betrieb von Anwendungen in unkontrollierbaren verteilten Umgebungen ist es wichtig, zuerst die Risiken und Fehlerursachen zu kennen und anschließend funktionierende Vermeidungs- und Minimierungsstrategien bereits während der Entwurfs- und Implementierungsphase von Anwendungen zu berücksichtigen. Dieses Kapitel stellt ausgehend von Annahmen, Risiken und Thesen zu Cloud-Anwendungen den Lösungsprozess und die Beiträge der Arbeit vor, bespricht die wissenschaftlichen Methoden bei der Ausführung des Prozesses zur Problemlösung, und leitet schließlich daraus die Struktur der weiteren Kapitel ab.

2.1 Annahmen und Beiträge

2.1.1 Annahmen dieser Arbeit

Die Erarbeitung eines fundierten, wiederverwendbaren und praxistauglichen Konzepts für die Entwicklung von nativen Cloud-Anwendungen mit den vorgestellten wünschenswerten Eigenschaften erfordert eine thematische Eingrenzung und eine Festlegung auf Annahmen. Diese treffen möglicherweise in ihrer Kombination nicht auf jede Cloud zu, lassen jedoch trotz Einschränkung einiger genereller Betrachtungen sowohl eine prototypische Umsetzung mit vertretbarem Aufwand als auch den tatsächlichen Einsatz aufbauend auf verbreiteten Dienstschnittstellen und darüber hinaus in vielen üblichen Cloud-Umgebungen zu.

Im Folgenden sind die gewählten Annahmen aufgeführt. Dabei wird ein abstrakter Schutzbegriff zum Schutz der Anwendung wie auch des Anwenders eingesetzt, welcher sich prinzipiell auf alle technischen Schutzziele wie Vertraulichkeit, Verfügbarkeit und Verlässlichkeit, Integrität, Authentizität, Verbindlichkeit, Zurechenbarkeit, Nicht-Verkettbarkeit, Intervenierbarkeit, Transparenz und Anonymität be-

zieht [BM12, Sho02]. Aufgrund der Dienstorientierung und des vorrangigen Bezugs der Problemstellung auf nicht vertrauenswürdige und nicht kontrollierbare Cloud-Umgebungen werden diese um weitere Ziele wie Preisstabilität, Erwartungskonformität, Client-Neutralität, Bindungssymmetrie und Portabilität ergänzt. Die konkrete Betrachtung erfolgt allerdings zumeist für die zwei erstgenannten Schutzziele.

Die Annahmen lauten im Einzelnen:

1. Daten sind schutzwürdig. Dies betrifft einerseits persönliche Daten, die oft einen schwer quantifizierbaren ideellen Wert aufweisen und somit subjektiv betrachtet stets bis auf expliziten Widerruf für den Eigentümer lesbar, für unbefugte Dritte hingegen nie lesbar, als auch in industriellen Prozessen auftretende Daten, welche mitunter einen über eine Risikoanalyse festgestellten monetären Wert aufweisen.
2. Datenverarbeitung ist schutzwürdig. Die Verarbeitung von Daten in der Cloud wird entweder zeitabhängig, ereignisabhängig oder durch explizite Dienstaufrufe von Anwendungen durchgeführt. Selbst bei angenommener Sicherheit der Eingabedaten können die Ergebnisse in der Form von Ausgabedaten fehlerhaft oder verfälscht sein. Die Entwicklungskonzepte für Cloud-Anwendungen müssen diese Risiken demnach mit berücksichtigen und minimieren.
3. Clouds sind risikobehaftet. Infrastrukturdienste in der Cloud sind vor allem unter Einbindung von Drittanbieterdiensten inhärent unsicher, nicht vertrauenswürdig, nicht vollständig kontrollierbar und unzuverlässig. Diese Annahme wird von den Cloud-Anbietern selbst als unnötig charakterisiert. Die immer wieder auftretenden Problemfälle mit öffentlichen Cloud-Anbietern [FGM13, PB10, WLMS14] führen jedoch zwangsweise zu ihrer Aufnahme in diese Liste.
4. Die Anwender möchten, ohne ein tieferes Sachverständnis für die Risiken ausweisen zu müssen, eine selbstbestimmte Abwägung zwischen der Schutzhöhe und Einschränkungen der Funktionalität oder offensiver nichtfunktionaler Eigenschaften treffen [SS12b, TPH+14]. Dies setzt bestimmte soziotechnische Vorkehrungen voraus, deren Diskussion keinen Lösungsbeitrag zu dieser Arbeit leisten würde und die deshalb als gegeben angenommen werden.
5. Cloud-Anwendungen sind sowohl solche, die als Dienst auf einer Cloud-Plattform bzw. Cloud-Infrastruktur betrieben werden als auch solche, deren Funktionalität von ersteren abhängt. Die reine Migration von nicht cloud-fähigen Anwendungen auf eine Cloud-Plattform bringt jedoch nur wenige Vorteile bei gleichzeitig vollem Risiko. Aus diesem Grund wird hierbei zwischen nativen und nichtnativen bzw. cloud-fähigen und nicht cloud-fähigen Anwendungen differenziert, wobei die Fähigkeiten mindestens dienstorientiert auszulegen sind und unter anderem eine erneute Bindung an veränderliche Endpunkte erlauben müssen. Native Cloud-Anwendungen nutzen zudem das Spektrum an Cloud-Funktionen von elastischer Skalierung bis hin zur losen Ankopplung von Speicherbereichen aus.

Es sei darauf hingewiesen, dass aus Annahme 4 abgeleitet werden kann, dass die angestrebte Problemlösung mitunter Komfortmerkmale von Cloud-Anwendungen wie günstige Laufzeiteigenschaften zugunsten des Schutzes von Daten und Datenverarbeitung aufgibt, was sich jedoch vor allem vor dem Hintergrund der oftmals

geringen Volumen besonders schützenswerter Daten mitunter kaum auf die vom Anwender letztlich wahrgenommene Leistung niederschlägt. Als wichtig und wesentlich wird vielmehr eine flexible Änderungsmöglichkeit innerhalb eines Bereichs zwischen den Extremwerten der Eigenschaften ohne unnötige Einschränkungen angesehen.

2.1.2 Risiken in Verbindung mit Cloud-Anwendungen

Die Risiken von verteilten dienstorientierten Softwarearchitekturen im Allgemeinen und von Clouds im öffentlichen Raum im Speziellen lassen sich nicht vollumfänglich aufzählen oder gar für konkrete Cloud-Dienste bewerten. Die folgende Liste entstand jedoch unter dem Einfluss vorhandener Studien, Analysen und eigener Erfahrungen und Ansichten. Sie folgt der vorsichtigen Herangehensweise nach dem Motto *»don't trust the cloud«* und benennt eine repräsentative Auswahl an Risiken, welche bei jeder datenverarbeitenden Anwendung auftreten können. Dies erfolgt in klarer Abgrenzung von solchen Ansätzen, die eine Vertrauensbasis etwa im Sinne von *trusted computing* über eine enge selbstdefinierte Vertrauenszone hinaus voraussetzen [Mic15, MM13]. Für alle aktiven Angriffe auf das System von innen wird ein Angreifermodell vorausgesetzt, welches mindestens die *honest-but-curious*-Eigenschaften erfüllt [BSSV10], während Angriffe von außen nach gängigem Vorgehen als nicht verhinderbar eingestuft werden.

1. Temporäre Nichtverfügbarkeit eines Dienstes. Dieses Risiko kann durch den Client, die Verbindung oder den Dienstanbieter ausgelöst werden. Clientseitig können fehlerhafte Verbindungseinstellungen oder fehlkonfigurierte Firewalls eine Kommunikation mit dem Dienst behindern. Netzwerkseitig können sporadische Verbindungsausfälle wie auch Kommunikationssperren auftreten. Dienstseitig können neben Überlastungserscheinungen auch Software- und Hardwarefehler sowie unangekündigte Endpunktänderungen durch physischen Umzug einen Dienstausfall verursachen.
2. Permanente Nichtverfügbarkeit eines Dienstes. Ökonomische Gründe führen oft zur Einstellung von Diensten oder zur Marktbereinigung von nicht profitablen Dienstanbietern.
3. Langsamkeit eines Dienstes: Aufgrund von Systemheterogenitäten, unterschiedlichen Systemzuständen und auch Softwarefehlern antworten Dienste mitunter mit geringer temporaler Vorhersagbarkeit beliebig langsam. Dieser Effekt ist von einer Nichtverfügbarkeit nur selten zu unterscheiden.
4. Preisänderung eines Dienstes. Zwar sind diese meist für den Dienstnutzer vorteilhaft, insbesondere bei Ressourcendiensten angesichts nahezu stetig sinkender Hardwarekosten. Mitunter werden jedoch kostenfreie Angebote kostenpflichtig oder bereits kostenpflichtige verteuert.
5. Temporärer Datenverlust: Durch Hardware- oder Bedienfehler kann auf Daten zeitweise nicht mehr zugegriffen werden, wobei eine Rücksicherung oder ein Umkehreffekt nach einiger Zeit diesen Verlust rückgängig machen können.

6. Permanenter Datenverlust: Ist keine Datensicherung vorhanden oder schlägt die Rücksicherung fehl, dann ist oftmals ein irreversibler Datenverlust die Folge.
7. Aktiver lesender Zugriff: Während der Datenzugriff für autorisierte Nutzer oder Anwendungen problemlos möglich ist, ist dies ohne Einverständnis des Dateneigentümers auch für unautorisierte Dritte möglich. Dies kann einerseits der Anbieter selbst über einen Insider-Angriff, andererseits auch eine Spionagesoftware in der Cloud oder eine externe Spionage auf der Leitungsebene sein.
8. Aktiver schreibender Zugriff: Ähnlich dem akiven lesenden Zugriff sind hierbei unautorisierte Dritte beteiligt, wobei diese neben dem Zugriff auf die Daten auch deren Modifikation (*malicious modification*) vornehmen, welche mitunter nur schwerlich zu entdecken oder gar rückgängig zu machen ist.
9. Aktive Täuschung: Dieses Risiko beinhaltet Aufrufe an den Cloud-Anbieter, bestimmte Operationen auszuführen, die zwar entsprechend bestätigt, nicht jedoch tatsächlich ausgeführt werden. Ein typisches Beispiel ist die Weigerung von Anbietern, Daten sicher zu löschen (*refusal to delete*). Bestimmte Täuschungsversuche lassen sich aufdecken, so beispielsweise die bewusst fehlerhafte Berechnung über eine byzantinische Mehrfachberechnung oder über einen vertrauenswürdigen Attestierungsdienst. Da diese jedoch nur Randbereiche einer Grauzone sind, wird dieses Risiko über die Annahme *honest-but-curious* für die weiteren Betrachtungen ausgeschlossen, was jedoch nicht bedeutet, dass es nicht existent und in der Tat hochproblematisch wäre.
10. Bindung an einen Dienstanbieter: Durch die Bindung an einen einzelnen Dienstanbieter können über die Zeit zunehmende *vendor-lock-in*-Effekte entstehen. Insbesondere sind die Aufwände zur Dienstanbindung und die Transferkosten oft asymmetrisch verteilt. Die Anmeldung zu einer Dienstnutzung *sign-up* ist relativ aufwandsarm über Webformulare möglich, die Abmeldung erfordert jedoch die Schriftform und oft zudem die Aufgabe von bereits bezahlten Restkontingenten. Der Datentransfer in die Cloud ist günstig oder kostenfrei. Um jedoch hohe Datenmengen aus der Cloud heraus oder zu einem anderen Anbieter zu migrieren, ist er ab einem bestimmten Punkt exorbitant hoch und verhindern die Mitnahme von für den Nutzer vorteilhaften Preisanpassungen.

Teilweise existiert mehr als eine Ursache für jedes Risiko. So kann ein permanenter Datenverlust auch durch seltene Naturereignisse oder durch politische Aktionen ausgelöst werden. Die Ursachenbetrachtung ist für diese Arbeit jedoch nebensächlich. Gleichfalls treffen einige der Risiken auf mehr als einen Dienst zu. Ein aktiv lesender Zugriff kann sowohl auf den Speicherbereich (Cloud Storage) als auch auf die Netzverbindung zwischen Bestandteilen der Anwendung erfolgen.

Das Ziel der Arbeit ist die Minimierung von einigen der vorgestellten Risiken. Konkret wird angestrebt, den Einfluss temporärer Ausfälle durch erhöhte Verfügbarkeit und unerlaubter Zugriffe durch erhöhte Vertraulichkeit weitgehend transparent für den Anwender durch geeignete Datenkodierungen zu unterbinden. Weiterhin wird durch architektonische Maßnahmen bei hinreichend großer Auswahl an Dienstalternativen die Bindung an einen Dienstanbieter und die aktive Täuschung unterbunden.

2.1.3 Thesen und Beiträge

Die zentrale These dieser Arbeit lautet:

> Die Sicherheit und Zuverlässigkeit von Anwendungen, welche schutzwürdige Daten verarbeiten, lässt sich durch die geschützte Verlagerung in die Cloud mit einer Kombination aus zielgrößenabhängiger Datenkodierung, kontinuierlicher mehrfacher Dienstauswahl, dienstabhängiger optimierter Datenverteilung und kodierungsabhängiger Algorithmen deutlich erhöhen und anwenderseitig kontrollieren. Die Kombination der Verfahren zu einer anwendungsintegrierten *Stealth-Schutzschicht* ist eine notwendige Grundlage für die Konstruktion sicherer Anwendungen mit zusicherbaren Sicherheitseigenschaften im Rahmen eines darauf angepassten Softwareentwicklungsprozesses.

Für diese Kombination werden einzelne Beiträge geliefert, die sich in der Kapitelstruktur zu daten- und plattformspezifischen Konzepten widerspiegeln. Insbesondere enthält die Arbeit eine Generalisierung des Kodierungsverfahrens *Dispersed Coding* sowie die Erweiterung *Stealth Coding*, die darauf adaptierten Paradigmen *Dispersed Computing*, *Dispersed Cloud Computing*, *Stealth Computing* und *Stealth Cloud Computing* sowie weitergehende Plattformkonzepte zur dynamischen Verwaltung von Daten und Diensten in einer geschützten Umgebung aufbauend auf ungeschützten Ressourcendiensten.

Die Zusicherung der sicherheitsbezogenen Zielgrößen als nichtfunktionale Eigenschaften (NFE) erfolgt über die nutzerkontrollierte Anhebung unterer Grenzen über die von einzelnen Diensten hinaus angebotenen informell gegebenen oder per Vereinbarung zugesicherten Eigenschaften. Weitere, nicht sicherheitsbezogene Eigenschaften wie beispielsweise die Energieeffizienz ließen sich ebenfalls aushandeln und technisch erzwingen [GIC+14a, GIC+14b], was allerdings eine Kooperation des Dienstanbieters über die gegebenen Annahmen hinaus erfordert.

Als Gegenthese sei gewählt, dass physische Geräte weniger Risiken ausgesetzt sind und somit per se bei guter Konnektivität eine höhere Zuverlässigkeit bieten können. Allerdings existieren hierfür weitere Risiken wie etwa Diebstahl und langsamere Entdeckung und Behebung von Hardwareschäden, Sicherheitsverlust durch veraltete Software. Desweiteren ist ab einer größeren Datenmenge, die im Laufe der Zeit beispielsweise kontinuierlich angewachsen ist, möglicherweise eine lokale Berechnung auf einem ressourcenbeschränkten Endgerät bezogen auf die Laufzeit und weitere NFE ungünstiger als selbst eine verlangsamte, da geschützte, verteilte dienstübergreifende Berechnung. Nach aktuellem Stand lässt sich nicht abschließend beurteilen, ob diese Risiken schwerer wiegen. Der Anspruch der Arbeit ist jedoch, für Nutzer privater bzw. persönlicher Cloud-Umgebungen mit selbst kontrollierten Geräten ebenfalls einen Mehrwert zu liefern. Aus diesem Grund erstreckt sich die Cloud-Definition der These auch auf solche Modelle.

Als grundlegendes Modell wird eine Prozesskette angenommen, über welche zuerst die zu verarbeitenden Daten vorverarbeitet und anschließend verteilt werden. Diese Kette lässt sich als funktionale Nutzungsvertikale auffassen, welche die Auslagerung von Daten auf verteilte Dienste und die darauf aufbauende Auslagerung oder Ankopplung von Anwendungen an diese Dienste subsumiert. Neben der Vertikalen zur Dienstnutzung existieren weitere zur vorherigen Konfiguration oder zur

parallelen Überprüfung und Kontrolle, die zu einer Gesamtsicht beitragen. Abbildung 2.1 stellt einführend den Prozess der Vorverarbeitung von Daten mit anschließender Übertragung, Speicherung oder Verarbeitung als Vertikale grafisch dar. Die markierten Flächen repräsentieren Beiträge in dieser Arbeit mit ihrer Unterkapitelzuordnung, während die Konfigurations- und Kontrollverfahren bereits Gegenstand eigener Vorarbeiten gewesen sind [STS14, SCB+14, SS13b, SZS13, SS12a, SS12b, Spi10].

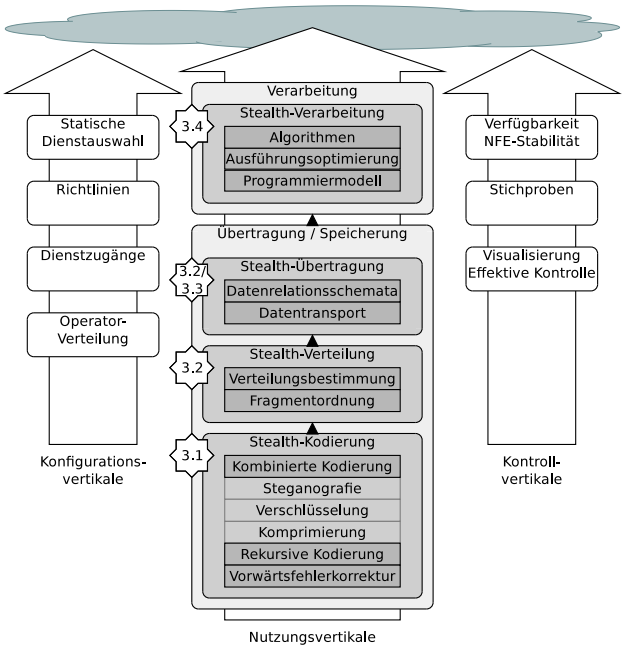

Abb. 2.1 Funktionale Vertikalen in nativen Cloud-Anwendungen

Die geleisteten Beiträge umfassen neue Denkmodelle und formalisierte Modelle, Algorithmen und Verfahren, deren Kombination in neuartiger Form, technische Umsetzungen in Form von Spezifikationen, Architekturen und Implementierungen sowie vielfältige experimentell erzielte Ergebnisse. Tabelle 2.1 stellt die wichtigsten Beiträge zur Theorie des sicheren verteilten Speicherns und Rechnens in einer Übersicht zusammen. Diese sind unabhängig von einer bestimmten Ausführungsarchitektur einsetzbar und damit für weiterführende Fragestellungen systemunabhängig nachnutzbar.

Die weiteren Beiträge bauen auf den Theoriebeiträgen auf, wobei sie an die Architektur gemäß den Annahmen und Definitionen zu Cloud-Umgebungen gebunden sind. Sie bestehen aus einer Vielzahl an Schnittstellen-, Format-, Sprach- und Protokollspezifikationen, Architekturen zur Datenverwaltung und zur Dienstintegrati-

Tabelle 2.1 Theoriebeiträge der Arbeit

Beitrag	Beitragsart	Abschnitt	Seite
Kombinierte Kodierung	Kombination	3.1.2 Überblick zur kombinierten Datenkodierung	18
Bitsplitting-Aufteilungskodierung	Verfahren	3.1.3 Aufteilung von Daten per Dispersion	22
Bitexpansion-Geheimnisteilung	Verfahren	3.1.3 Aufteilung von Daten per Dispersion	22
Kodierungsketten	Kombination	3.1.6 Kombinierte Verfahren	36
PICav+: Präzise iterative Verteilungsbestimmung	Verfahren	3.2.4 Verteilung zur Erzielung einer Mindestverfügbarkeit	41
DRS, FRS: Daten- und Dateirelationsschemata	Schema	3.3.1 Datenrelationsschemata	55
dispergiertes Rechnen	Verfahren	3.4.1 Verarbeitung dispergierter Daten	61
Redundanzsuche	Verfahren	3.4.1 Verarbeitung dispergierter Daten	61
Stealth-Rechnen	Kombination	3.4.3 Verarbeitung dispergierter und verschlüsselter Daten	64
Stealth-Algorithmen	Verfahren	3.4.4 Konstruktion und Analyse von Algorithmen	68

on, Definitionen von Szenarien, Softwareanwendungen und Experimenten, sowie schließlich deren Umsetzung respektive Durchführung.

2.2 Wissenschaftliche Methoden

Die Bearbeitung des Themas erfolgt abschnittsweise mit unterschiedlichen etablierten wissenschaftlichen Methoden. Die beiden Konzeptkapitel 3 und 4 enthalten für die herausgestellten Beiträge jeweils eine Literaturrecherche in Form eines *literature review* sowie eine fundierte Analyse der vorhandenen Arbeiten. Diese werden jeweils im Text mit aufgeführt, so dass sich für den Leser eine gesamtheitliche Darstellung ergibt. Die Validierung der Konzepte im Kapitel 6 erfolgt durch Umsetzung in Form von verfahrensbezogener synthetischer Software mit anschließender experimenteller Evaluierung dieser und in Vorarbeiten entstandener verfahrensenthaltenden Softwareanwendungen in unterschiedlichen realen und simulierten Cloud-Umgebungen. Die Rohdaten und extrahierten Ergebnisse werden gemäß den Konventionen des *scientific data management* und der *recomputability* zusammen mit den Experimentvoraussetzungen für die Gewährleistung der Reproduzierbarkeit dauerhaft archiviert und öffentlich zur Verfügung gestellt. Ein informelles Projekt mit der Bezeichnung *Cloud Storage Lab* bündelt die Software- und Experimentieraktivitäten zum Thema. Es dient als greifbare und nachhaltige Projektionsfläche für die Ergebnisse.

Eine neue wissenschaftliche Methode wird nicht vorgeschlagen. Es wird stattdessen der Begriff *systems-centric research* genutzt, um zu betonen, dass sich die Forschungsbeiträge zwar auf die Zusicherung von Eigenschaften für Anwender beziehen, aber keine Anwenderstudie durchgeführt wurde. Stattdessen werden die konstruierten Systeme selbst zum Gegenstand weiterführender Forschungsarbeiten einschließlich der Anwenderstudien.

2.3 Struktur der Arbeit

Die Beiträge der Arbeit beginnen mit theoretischen Betrachtungen zu Modellen, Schemen, Ontologien und Algorithmen zu verteilten Daten und Anwendungen und führen über Architektur- und Infrastrukturthemen zu prototypisch implementierten und experimentell evaluierten Systemen. Grundlagenbetrachtungen und Gegenüberstellungen verwandter Arbeiten als Stand der Technik sind bewusst nicht zentral zusammengefasst, sondern stets in den jeweiligen Unterkapiteln enthalten, um genauere Einordnungen und Abgrenzungen zu ermöglichen.

Die weitere Arbeit ist wie folgt gegliedert. Im dritten Kapitel (3. Stealth-Konzepte für die abgesicherte Datennutzung) werden Konzepte zur Kodierung und Verteilung von Daten erörtert, die von hoher Wichtigkeit für alle datenintensiven Anwendungen sind. Im vierten Kapitel (4. Stealth-Konzepte für zuverlässige Dienste und Anwendungen) folgt die Einführung von Plattformkonzepten, mit denen ähnlich zu existierenden PaaS-Schnittstellen die Entwicklung und die Inbetriebnahme von Diensten und Anwendungen vereinfacht wird. Im fünften Kapitel (5. Szenarien und Anwendungsfelder) werden Szenarien mit unterschiedlichen Zielgruppen vorgestellt, welche bisher nicht oder nur umständlich, mit den erarbeiteten Konzepten jedoch einfach und schnell realisiert werden können. Das sechste Kapitel (6. Validierung) bespricht die Umsetzung der Konzepte in Form von Software und validiert die Konzepte experimentell. Schließlich wird die Arbeit im siebenten Kapitel zusammengefasst.

Kapitel 3
Stealth-Konzepte für die abgesicherte Datennutzung

Zusammenfassung Datenverarbeitende Anwendungen unterliegen mehreren datenbezogenen Risiken. Zu deren Minimierung müssen die Daten derartig aufbereitet werden, dass einerseits die berechtigte Verarbeitung weiterhin ohne größeren Mehraufwand möglich ist, der Schaden für die Anwendung oder den Anwender im Fall beeinträchtigter Daten jedoch weitgehend ausgeschlossen werden kann. In diesem Kapitel werden diese Bedingung erfüllende Kodierungs- und Verteilungsmechanismen, Verteilungsrelationen sowie daran angepasste Verarbeitungsvorschriften auf algorithmischer Ebene vorgestellt. Als übergreifendes Schutzkonzept wird damit das Prinzip des *Stealth Computing* herausgearbeitet.

3.1 Datenkodierung

3.1.1 Konzeptioneller Ansatz

Die ausgelagerte, mithin verteilte, Verarbeitung von Daten unter Zusicherung nichtfunktionaler Eigenschaften als Zielgrößen erfordert eine zielgrößenoptimierte Vorverarbeitung in einem nutzerkontrollierten System sowie eine auf die vorverarbeiteten Daten adaptierte Verarbeitungsmethode in einem als nicht notwendigerweise kontrollierbar angenommenen separaten System. Die Datenkodierung steht am Anfang der in Abbildung 3.1 gezeigten Verknüpfung von Konzepten respektive Ablaufschritten zur Vorverarbeitung mit dem Ziel der sicheren und zuverlässigen Datennutzung in verteilten Anwendungen. Die Existenz der Daten und ihre Herkunft aus einer beliebigen Quelle, welche neben dem nutzerkontrollierten System auch ein sicheres externes System sein kann, wird vorausgesetzt. Die Übergabe der Daten von der Datenquelle an das Vorverarbeitungssystem durch eine passende Datenübertragung entspricht weitgehend der Übertragung vom Vorverarbeitungssystem an die Datensenke, beispielsweise einen Dienst, und wird zusammen mit dieser im Anschluss beschrieben.

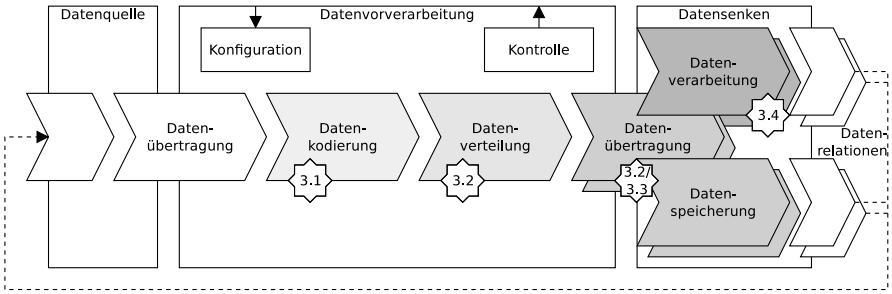

Abb. 3.1 Konzeptbestimmende Ablaufschritte für die zuverlässige systemübergreifende Datennutzung

Für die Vorverarbeitung wird ein generischer Ansatz gesucht, um eine wachsende Zahl von datenproduzierenden und datenkonsumierenden Systemen so miteinander verbinden zu können, dass die Eigenschaften des Gesamtsystems zusicherbar sind und dennoch der Entwicklungsaufwand für Anwendungen durch einfache Integration des Vorverarbeitungssystems niedrig bleibt.

Die Erarbeitung der Lösung erfolgt in vier komplexen Beiträgen in Form von Unterkapiteln. Im weiteren Text des Abschnitts 3.1 werden zuerst Datenkodierungstechniken untersucht und um zwei eigene Verfahren zur Datenaufteilung ergänzt. Desweiteren werden kombinierte Verfahren vorgestellt, tiefer untersucht und formalisiert. Der zweite komplexe Beitrag im Abschnitt 3.2 untersucht anschließend geeignete Verteilungen der kodierten Daten und führt kapazitiv gestaffelte Verfahren ein. Im dritten komplexen Beitrag im Abschnitt 3.3 wird eine Repräsentationsform derartig verteilter Datenbestände mit semantischen Beziehungen in Form von Relationenschemata entwickelt. Der vierte komplexe Beitrag untersucht die Verarbeitung verschlüsselter aufgeteilter Daten unter Nutzung adaptierter Verarbeitungsalgorithmen. Der Schwerpunkt ist dabei vor allem der Einfluss der Vorverarbeitung und der Verteilung auf die Verarbeitung. Drei konkrete Verfahren und eine kombinierende Sicht werden erarbeitet. Mit diesen insgesamt zehn Einzelbeiträgen stehen in hinreichender Zahl und Tiefe abstrakte Konzepte und theoretische Betrachtungen zur zuverlässigen Datennutzung und Anwendungsausführung unter der Sammelbezeichnung Stealth Computing bereit, bevor diese Konzepte im Folgekapitel in verteilte Architekturen eingesetzt werden.

3.1.2 Überblick zur kombinierten Datenkodierung

Als Daten werden im Folgenden Symbolmengen betrachtet, welche in Systemen zur Verarbeitung, Übertragung oder Speicherung vorkommen und zu einem bestimmten quantifizierbaren Grad Informationen enthalten, welche wiederum eine nicht weiter bestimmte Menge an Wissen repräsentieren [Ack89]. Die Informationsdichte oder Entropie wird über den Anteil redundanter Daten bestimmt und beträgt im Normal-

fall $1,0$ ohne Berücksichtigung inhärenter Redundanzen des Datenformats oder der statistischen Verteilung der Symbole. Die äußere Form der Daten ist die einer Datei, eines Datenstroms oder eines Tupels. Die Formfrage ist für wenige algorithmische Verfahren während der Vorverarbeitung, jedoch wesentlich für die Datenverarbeitung zur Laufzeit von Interesse.

3.1.2.1 Kodierungsstrukturen

Die Kodierung von Eingangsdaten d in Ausgangsdaten d' lässt sich im einfachen Fall durch eine Kette und im generalisierten Fall durch einen Baum von z Transformationen T_i $(i,z \in \mathbb{N}; 0 \leq i \leq z-1; z \geq 1)$ repräsentieren. Jede Transformation bildet eine Menge von Daten entweder auf eine andere Menge oder kodierungsspezifisch auf mehrere Mengen in einer Multimenge der Kardinalität n ab, d.h. $d'|\{d'_1,...d'_n\} := T_{i_j}(d)(n \in \mathbb{N}; n \geq 1)$. Dabei ist d' mitunter unvollständig, falls redundante Mengen durch Nichtverfügbarkeit oder Verzicht ausgeschlossen werden. Dann gilt für den Informationsgehalt I der ausgeschlossenen Mengen: $I(d'_{ex}) = 0$. Es gilt jedoch für alle verlustfreien Transformationen stets bezüglich des gesamten Informationsgehaltes: $I(\sum_{j=1}^{n} d'_j) = I(d)$, womit die Bildung der inversen Transformation $T_{i_j}^{-1}$ gesichert ist. Ist eine Transformation verlustbehaftet mit dem Qualitätsfaktor Q $(Q \in \mathbb{R}; 0 \leq Q \leq 1)$, so gilt näherungsweise $I(\sum_{j=1}^{n} d'_j) \cong QI(d)$.

Eine grafische Darstellung der Transformations- bzw. Kodierungskette erfolgt in Abbildung 3.2. Sie enthält eine optionale Aufteilung, wodurch eine Kodierungsbaumstruktur entsteht. Vollständige gerichtete Graphstrukturen werden an dieser Stelle nicht betrachtet, obwohl sie durch die Zusammenführung einer Multimenge als Definitionserweiterung vorstellbar sind.

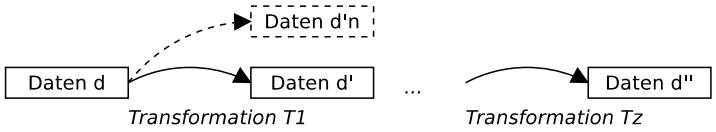

Abb. 3.2 Abstrakte Datenkodierungskette mit z Transformationen

Bekannte Verfahren der Datenkodierung umfassen die Vervielfältigung (einfache und mehrfache Replikation, balancierte Hashring-/Hashtable-Replikation), die einfache Aufteilung (Partitionierung, Blockbildung, Chunking, Slicing), und die Informationsdispersion mit selektiver Redundanz und den Ausprägungen Vorwärtsfehlerkorrektur/Löschkodierung (*Erasure Codes*) und Geheimnisaufteilung (*Secret Sharing*). Der Schwerpunkt dieses Kapitels liegt wegen ihrer flexiblen Parametrisierung auf Dispersionsverfahren, worauf im Folgenden genauer eingegangen wird. Neben dieser Grundkodierung, welche die Dateninhalte respektive die in den Datensymbolen enthaltene Information unberücksichtigt lässt, sind weitere formatunabhängige aber datenauswertende Kodierungs- und Transformationsverfahren zur

Komprimierung, zur Verschlüsselung und zur Steganographie sowie formatspezifische Transformationen bekannt, welche teilweise ebenfalls einen Beitrag zur Problemlösung leisten und dazu kurz vorgestellt werden. Abbildung 3.3 fasst mehrere Kodierungsverfahren ohne Anspruch auf Vollständigkeit in drei Kategorien zusammen. Aus ihr geht hervor, dass einige Verfahren neben den eigentlichen Daten zusätzliche Eingabeparameter besitzen, welche als *Metadaten* bezeichnet werden. Weitere Verfahren auf der Ebene der Datenorganisation umfassen Deduplizierung, Versionierung und Archivierung. Sie sollen an dieser Stelle nicht weiter betrachtet werden.

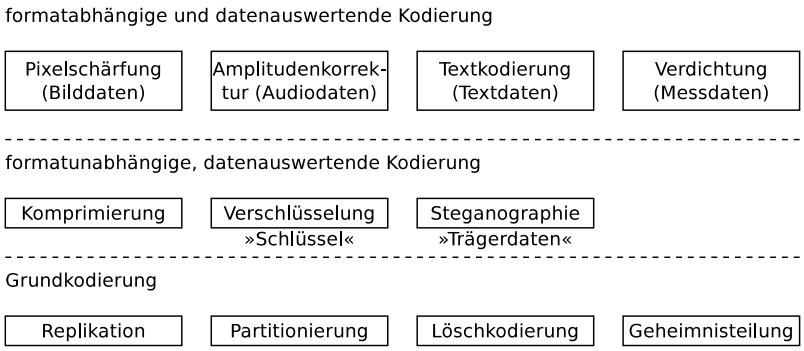

Abb. 3.3 Kategorien der Kodierung von Daten: Grundkodierung und erweiterte Kodierung

Jeder Kodierungsschritt besitzt einen Inversalgorithmus T^{-1} zur Wiederherstellung von ursprünglichen Daten. Sind Metadaten für die Kodierung genutzt worden, müssen diese zumeist für die Dekodierung bereitgehalten werden.

Man unterscheidet unabhängig von der Interpretation von Inhalten zwischen Kodierungsverfahren ohne Berücksichtigung der Datentypsemantik, welche die Verarbeitung ohne Datentypinformationen, also ungetypt, auf Basis eines Bytestroms vornehmen, und denen, die typisiert arbeiten. Desweiteren wird zwischen strukturerhaltender und strukturzerstörender Kodierung unterschieden. Bei letzterer ist es mitunter nicht mehr möglich, neben dem Inversalgorithmus weitere Verarbeitungsschritte durchzuführen. Ungetypte Verfahren arbeiten mitunter zwar strukturerhaltend, aber nichtdeterministisch, so dass ähnliche Probleme auftreten. Die Relevanz der verarbeitungsabhängigen Strukturerhaltung wird deutlich, wenn über die kodierungsunabhängige Übertragung oder Speicherung von Daten hinaus eine Verarbeitung der kodierten Daten erfolgen soll. Aus diesem Grund leistet diese Arbeit einen Beitrag zur risikomindernden *holistischen Datenkodierung* als Überlagerung einer Grundkodierung mit mehreren formatunabhängigen oder formatabhängigen zusätzlichen Kodierungen.

3.1.2.2 Stand der Technik: Kombinierte Datenkodierung

Der Stand der Technik zur kombinierten Kodierung von Daten für Anwendungen mit zusicherbaren Eigenschaften umfasst mehrere anwendungsspezifische Ansätze mit diversen Unzulänglichkeiten hinsichtlich der Anforderungen. Sobe und Peter schlagen zur sicheren Speicherung eine Datenkodierung mit überlagerten Techniken bestehend aus vier Phasen vor: Aufteilung von Daten, Anreicherung um fehlertolerante Kodierung, Komprimierung und Verschlüsselung [SP07]. Das vorgeschlagene Modell sieht bereits eine flexible Zusammenstellung der Kodierungsschritte per Superposition vor. Die Autoren haben es hinsichtlich der Brauchbarkeit, der Laufzeiteigenschaft und des Mehrbedarfs an Speicherplatz der $4! = 24$ möglichen Zusammenstellungen validiert. Die Nachteile dieses Ansatzes sind die Einschränkung auf die Datenspeicherung ohne eine Untersuchung nachgelagerter Verarbeitungsschritte, die Annahme homogen charakterisierter Datensenken, speziell in Form von Netzwerkdiensten, sowie die fehlende Berücksichtigung der Strukturerhaltung. Demgegenüber sieht der Ansatz in dieser Arbeit die Berücksichtigung von Datenstrukturen und Diensteigenschaften sowie eine bei Bedarf auf die weitere Verarbeitung zielende Auswahl an Techniken vor. García et al. geben einen Überblick über die Vorverarbeitung von Daten (*preprocessing*) für die nachgelagerte Datenanalyse [GLH15]. Derartige Verfahren verbessern die Datenqualität an sich, sind jedoch ohne Ergänzung nicht geeignet, den Zugriff auf die Daten in verteilten Umgebungen abzusichern. In dieser Arbeit werden die Datenqualitätsdimensionen bzw. -metriken wie Vollständigkeit, Aktualität oder Dichte nicht berücksichtigt. Lediglich die Präzision von Daten ist im Zusammenhang mit der Verteilung eine Metrik, deren Berücksichtigung die Möglichkeiten zur Berechnung im Fehlerfall ausweitet. Mancill und Pilskalns kombinieren Verschlüsselung und Komprimierung zur Datenübertragung in drahtlosen Sensornetzwerken, berücksichtigen jedoch keine Verteilung [MP11]. Shomorony et al. untersuchen die verteilte Komprimierung von Netzwerknachrichten, allerdings ohne Berücksichtigung der dezentralen Verarbeitung und ohne weitere Absicherung der Nachrichten durch Verschlüsselung [SAAW12].

Die verwandten Arbeiten werden in der Tabelle 3.1 zusammengefasst. Vor allem die fehlende Betrachtung der weiteren Datenverarbeitung begründet die Notwendigkeit einer eigenen Untersuchung der kombinierten Kodierung in dieser Arbeit.

Tabelle 3.1 Vergleich existierender Betrachtungen von Mehrfachkodierungen

Ansatz	Verteilung	Verschlüsselung	Komprimierung oder weitere	Verarbeitung
Sobe und Peter	x	x	x	-
García et al.	-	-	x	-
Mancill und Pilskalns	-	x	x	-
Shomorony et al.	x	-	x	-

Neben diesen zumindest teilweise holistischen mehrfachkodierungsbezogenen Betrachtungen existieren viele Ansätze in den tangierten einzelnen Kodierungsthemen der Datendispersion, der Verschlüsselung, der Komprimierung und der

Steganographie, welche in den nachfolgenden Abschnitten vor allem hinsichtlich der direkten Verarbeitung im kodierten Zustand separat ausgewertet werden, bevor die Analyse der Verarbeitung kombiniert kodierter Daten folgt. In summa steht somit fest, dass eine ganzheitliche Sicht auf die sichere Datenverwaltung und -verarbeitung in verteilten Systemen bisher nicht gegeben ist und eine solche somit als Grundlage für sämtliche Zusicherungen in Anwendungen und Anwendungsdiensten in diesem Kapitel zu erarbeiten ist.

3.1.3 Aufteilung von Daten per Dispersion

Die Informationsdispersion bezeichnet ein Verfahren, bei dem informationsenthaltende Daten so auf k signifikante Teile aufgeteilt werden, dass der bloße Zugriff auf $k-1$ Teile nicht ausreicht, um deterministisch die Daten vollständig wiederherzustellen und somit die kodierte Information in Gänze erlangen zu können [Rab89]. Vielmehr müssen sämtliche k Teile kombiniert werden. Dies sichert die praktische Vertraulichkeit unter der Annahme einer Verteilung auf k unterschiedliche Systeme, Dienste oder Vertrauensdomänen nicht kooperierender Besitzer und Anbieter zu. Einige Verfahren lockern diese Einschränkung auf und erfordern lediglich $\tilde{k} \leq k$ für eine zwar nicht vollständige, aber zumindest näherungsweise Wiederherstellung der Information. Andere Verfahren erzielen eine höhere Vertraulichkeit der kodierten Daten durch Secret-Sharing-Algorithmen, für die ausnahmslos gilt: $\tilde{k} \geq k$. Zusätzlich ermöglichen mehrere Algorithmen zur Informationsdispersion (*information dispersal algorithm*, IDA) die Generierung von m redundanten Fragmenten zur Gewährleistung einer m-fachen Ausfallsicherheit [Pla13]. Dabei gilt für die Gesamtzahl der Teile n: $n = m + k$, und somit ein kapazitiver Mehraufwand (*overhead*) für die Speicherung oder Übertragung der Daten von $\frac{n}{k}$, mitunter auch notiert als $\binom{n}{k}$ im Hinblick auf die kombinatorische Aufgabe des erneuten Abrufs der signifikanten Fragmente ohne Fehlerfall. Die resultierenden Teile werden in der Literatur als Fragmente, Blöcke, Chunks oder Slices bezeichnet, wobei in diesem Text die Bezeichnung Fragmente konsistent eingesetzt wird, da bei einer holistischen Betrachtung die weiteren Begriffe an anderer Stelle benötigt werden. Treten Fehler auf und reicht auch die Redundanz nicht aus, so sind die durch die Daten repräsentierten Informationen verloren. Es sei die Zahl der tatsächlich zur Verfügung stehenden signifikanten Fragmente als $\hat{k} \leq k$ bezeichnet. Die nachfolgenden Abschnitte stellen die einzelnen Dispersionsverfahren Blockbildung/Chunking, Löschkodierung/Erasure Coding, Replikation, Secret-Sharing, Interpolation und Bitsplitting stereotypisch und exemplarisch mit jeweils einem Kodierungsbeispiel vor.

3.1.3.1 Aufteilung in Blöcke, Chunks, Slices und Partitionen

Die einfachste Variante eines IDA ist die redundanzlose Aufteilung eines Datenfeldes in k Fragmente fester Länge ohne Berücksichtigung der Inhalte. Abbildung 3.4

veranschaulicht das Verfahren. Hierbei wird klar, dass ungetypte Verfahren nicht immer anwendbar sind, da die Aufteilungsgrenzen unter Umständen Daten ungewollt separieren, sofern sie nicht Vielfache der Datentypsgrenzen sind. Hingegen lässt sich mit der getypten Dispersion die Aufteilung innerhalb eines Datentyps vorteilhaft ausnutzen. Im Fall von Zeichenketten liegt beispielsweise eine variable Bytelänge eines Zeichens abhängig vom Zeichensatz bzw. der Codetabelle vor.

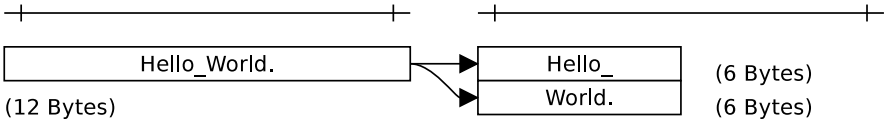

Abb. 3.4 Aufteilung von Daten in Chunks für $k = 2$

Werden die Fragmente nicht auf mehrere Systeme verteilt, sondern bilden sie in einer zweiten Stufe die jeweilige Basis für eine weitere Dispersion, so werden diese Fragmente als Eingabe-Chunks oder -Slices betrachtet, andernfalls als zu verteilende Datenfragmente. In beiden Fällen sind die nachgelagerten Kodierungs- oder Verarbeitungsschritte parallelisierbar, was in Kombination mit dem möglichen Einsatz von Interleaving- und Streamingtechniken das Chunking zu einer wichtigen ressourcenschonenden und beschleunigenden Funktion einer Datenvorverarbeitung werden lässt. Diese Charakteristik wird in der Abbildung 3.5 anhand eines Datenstroms verdeutlicht. Dieser wird regelmäßig in Chunks aufgeteilt, wobei diese wiederum per Dispersion kodiert und anschließend verteilt werden. Somit ist der Ressourcenbedarf durch die Datenreplikation mit der Größe eines Chunks erst geringfügig höher, bleibt anschließend aber konstant gegenüber einer vollständigen Zwischenspeicherung des Datenstroms.

3.1.3.2 Löschkodierung und Replikation

Im Gegensatz zur Blockaufteilung beziehen Löschkodes (*erasure codes*) einen frei wählbaren Redundanzfaktor mit ein. Redundante Blöcke werden auf Basis der signifikanten Blöcke errechnet, welche wiederum mit der Blockaufteilung oder einem anderen Verfahren aus den Ausgangsdaten gebildet werden. Die Abbildung 3.6 stellt das Verfahren grafisch dar. Der Name Löschkode bezieht sich dabei auf die mögliche Korrektur durch Ignorieren und Substituieren von gelöschten, also nicht mehr verwendbaren, Fragmenten. Damit sind Löschkodes die primäre Technik zur Gewährleistung einer hohen Datenverfügbarkeit in verteilten Systemen.

Löschkodes haben die Eigenschaft, äquilonge Fragmente, also Fragmente gleicher Größe, zu produzieren, wobei im Fall eines verbleibenden Divisionsrests, also $len(d)\%k \neq 0$, die Ausgangsdaten um Füllzeichen (Padding) erweitert werden. Im historischen Kontext wurden diese Kodierungen vorrangig als Schutzmaßname gegen Medienausfälle in lokalen Speicherverbünden mit mehreren Festplatten sowie für die Netzwerkübertragung von Daten genutzt, sind jedoch mit dem Aufkommen

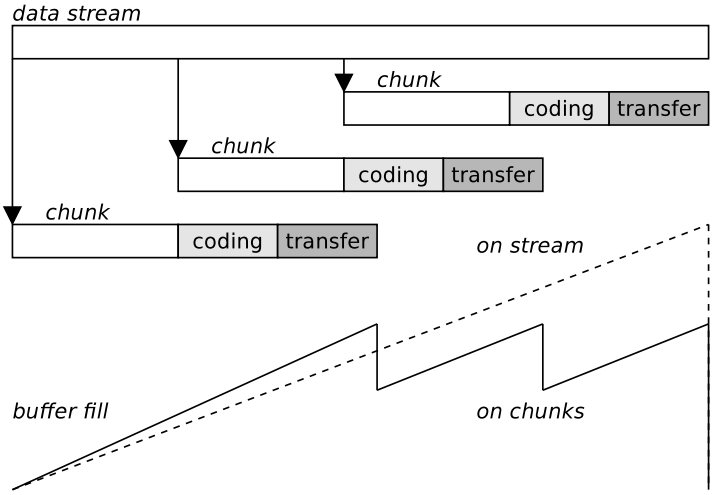

Abb. 3.5 Chunking, Parallelisierung und Interleaving während der Datenstromkodierung

von Online-Speicherdiensten gerade für diesen Anwendungsfall weiter optimiert, als Schutzmaßnahme auch gegen Ausfälle erkannt und in datenverarbeitende Systeme integriert worden.

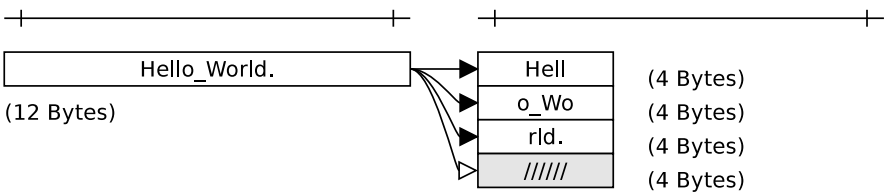

Abb. 3.6 Aufteilung von Daten in Löschkode-Fragmente für $k = 3, m = 1$

Es existieren mehrere algorithmische Umsetzungen von Löschkodes, die sich teils im Platzbedarf und teils in der Gewichtung der Aufteilungs- und Zusammenführungszeit unterscheiden [Pla13]. Desweiteren sind einige von ihnen auf bestimmte Implementierungsformen oder Ausführungsumgebungen optimiert. Die Algorithmen umfassen die Reed-Solomon-Kodierung [RS60], die Cauchy-Reed-Solomon-Kodierung [BKK+95], sowie mehrere speziell für ausfallsichere Datenträger konzipierte Verfahren.

Eine wichtige Eigenschaft hinsichtlich der Kodierungseffizienz ist die Zunahme an redundanten Daten proportional zur Tolerierbarkeit an Ausfällen. Sind beide Werte stets identisch, weist die Kodierung die Eigenschaft *maximum-distance separable* (MDS) auf und ist damit optimal. Sowohl übliche RAID-Kodierungen (RAID-4..6) als auch die meisten Reed-Solomon-Code-Varianten besitzen die MDS-Eigenschaft. Deren Aufgabe stellt eine Abwägung über die Laufzeiteigenschaften dar,

unter anderem durch eine Implementierbarkeit ohne rechenintensive Nutzung von Galois-Feldern und dafür mit performanten XOR-Aufrufen [Pla13]. Die Verfahren Tornado, Raptor, HoVeR, WEAVER und GRID sind im Gegensatz dazu (bis auf Ausnahmen, z.B. WEAVER mit $m = 2, n = 2$) Vertreter MDS-freier Kodierungen. Array-Kodierungen wie Cauchy-Reed-Solomon, RDP, EVENODD, sowie für bestimmte Konfigurationen auch Blaum-Roth, Liberation und STAR besitzen hingegen die MDS-Eigenschaft, lassen sich aber dennoch mit XOR-Aufrufen implementieren. Auch *locally-repairable codes* (LRC) mit sowohl lokalen als auch globalen redundanten Fragmenten sind MDS-frei, bieten dafür jedoch einen höheren Wiederherstellungsdurchsatz an. Diese werden vorrangig von Speicherdienstanbietern eingesetzt. Vertreter sind Partial-MDS [BHH13], welches die physischen Zusammenhänge von Fehlern auf dem Speichermedium berücksichtigt, sowie Azure-Coding und Xorbas [SAP+13].

Die Replikation von Daten wird meist durch 1 : 1-Kopiervorgänge erreicht, stellt jedoch algorithmisch auch einen Spezialfall der Löschkodierung dar und kann somit bei Nichtberücksichtigung des erhöhten algorithmischen Aufwands unter diesen subsumiert werden. Hierbei gilt: $k = 1$ und $m \geq 1$. Aufgrund der verhältnismäßig hohen Kapazitätsanforderungen für die redundanten Daten, welche bei Nutzung von Fremdsystemen zudem in hohe Kostenanforderungen umschlagen können, wird die vollständige Datenreplikation zunehmend zugunsten optimierter Löschkodes aufgegeben, was wiederum eine Herausforderung für die weitere dezentrale Verarbeitung der Daten darstellt [AC15]. Gleichzeitig eröffnet diese Transition weitreichende Möglichkeiten zur feingranularen Aufteilung der Daten anhand heterogener Einzelkapazitäten, zur Erhöhung der Vertraulichkeit und Verringerung der Datenmigrationskosten, sowie zu aufteilungsspezifischen Verarbeitungsoptimierungen, welche in dieser Arbeit vorgestellt werden.

3.1.3.3 Aufteilung per Secret-Sharing

Während Löschkodes allein durch die Auslassung von Informationen eine zwar praktische, aber nicht informationstheoretische Sicherheit gegen unberechtigtes Auslesen bieten, füllen Secret-Sharing-Algorithmen durch Gewährung von Vertraulichkeit diese Lücke. Diese existieren sowohl in einfacher Form ($m = 0$) als auch mit Redundanz ($m > 0$). Allerdings wird durch die Redundanz der Platzbedarf enorm gesteigert, da jedes Fragment mindestens die Größe des gesamten ursprünglichen Datenfeldes annimmt (Abbildung 3.7).

Bekannte Verfahren existieren seit einiger Zeit [Sha79, Bla79]. Mit der zunehmenden Popularität von Online-Speicherdiensten und multiplen Netzwerkanbindungen werden sie verstärkt untersucht und genutzt.

Begrifflich eng verwandt mit dem Secret-Sharing ist die All-Or-Nothing-Transformation (AONT) [Riv97], welche strikt betrachtet eine Verschlüsselungstechnik darstellt. Mit dem Verfahren AONT-RS existiert allerdings eine vorteilhafte Verbindung aus der Reed-Solomon-Löschkodierung und AONT [RP11], welche jedoch einschränkend eine feste Fragmentgröße voraussetzt. Eine alternative Kombinati-

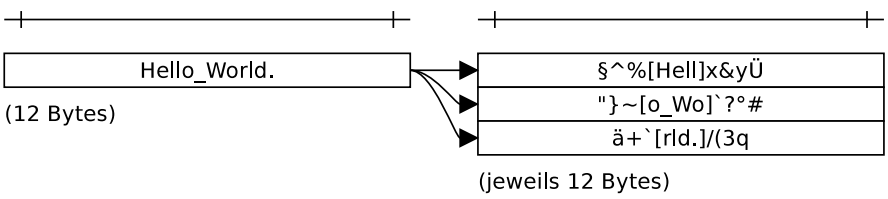

Abb. 3.7 Aufteilung von Daten in Secret-Sharing-Fragmente mit $k = 3$

on ist AONT-LT unter Nutzung einer Luby-Transformation, welche eine variable Fragmentgröße zulässt, dafür jedoch den Zugriff auf mehr als k Fragmente für die Wiederherstellung benötigt [BMMC14].

Secret-Sharing-Schemen sind per Definition nicht strukturerhaltend, da jede verbleibende Information über die Struktur die unberechtigte Wiederherstellung der Ursprungsdaten erlauben würde. Sie können damit nicht die Basis für eine schützende Kodierung von zu verarbeitenden Daten bilden; gleichwohl sollten sie als ergänzende Maßnahme in eine gesamtheitliche Kodierung für besonders schutzwürdige sekundäre Daten eingebunden werden.

3.1.3.4 Aufteilung mit Interpolationsausgleich

Das duale Ziel, den Platzbedarf aufgeteilter Daten gering zu halten und gleichzeitig die Verfügbarkeit der Daten zu gewährleisten, unterliegt durch die proportional zur gewünschten Verfügbarkeit steigende notwendige Menge an redundanten Daten einem Konflikt und ist somit schwierig zu erreichen. Abhilfe schaffen Kodierungsverfahren, die redundanzfrei fehlende Fragmentinformationen durch Interpolation aus den übrigen Fragmenten näherungsweise ausgleichen können. Diese Möglichkeit besteht nur bei wenigen Datentypen, ist also nicht generisch für beliebige Daten anwendbar, und ist stark abhängig vom Verwendungszweck. Handelt es sich bei den Daten beispielsweise um Bilder, die vom Benutzer nur betrachtet werden sollen, dann ist selbst eine starke visuelle Interpolation kaum auffällig und kann demnach selbst bei Ausfall vieler Speicherorte eine hohe wahrgenommene Datenverfügbarkeit leisten. Ähnliches gilt für andere multimediale Daten. Selbst ohne Durchführbarkeit einer Interpolation können komposite Datenformate wie etwa Dokumente mit einzelnen Seiten bei entsprechender Aufteilung im Fehlerfall mit $\tilde{k} < k$ Fragmenten näherungsweise oder teilweise genutzt werden. Hierfür ist jedoch die Berücksichtigung der Datensemantik zwingend.

Abbildung 3.8 veranschaulicht die Funktionsweise der Aufteilung mit Interpolationsausgleich im Fehlerfall.

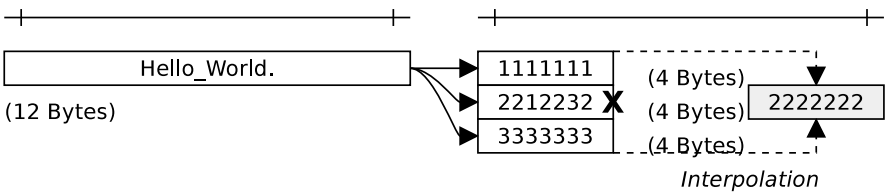

Abb. 3.8 Aufteilung von Daten mit Interpolationsmöglichkeit

3.1.3.5 Aufteilung nach Bits

Während die bisher vorgestellten Verfahren der Aufteilung in Blöcke, mit Löschkodierung, mit Replikation oder mit Secret-Sharing mitunter keine Struktur- und Typeigenschaften der Daten berücksichtigen und nur grobgranular mit Fragmentgrößen von mindestens 8 Bit arbeiten, in der Praxis oftmals auf mehrere Kilobytes erweitert, stellt eine flexible feingranulare Aufteilung der Daten in Fragmente beliebiger Bitbreite eine lückenfüllende Erweiterung der Kodierungsoptionen dar. Dadurch kann vermieden werden, dass einzelne Werte vollständig in den Fragmenten auftauchen und dem Risiko des unberechtigten Lesens ausgesetzt sind. Gleichzeitig ergeben sich durch die selektive Verarbeitung einzelner entstehender Fragmente neue Möglichkeiten der sicheren verteilten Verarbeitung. Abbildung 3.9 veranschaulicht die bitweise Aufteilung.

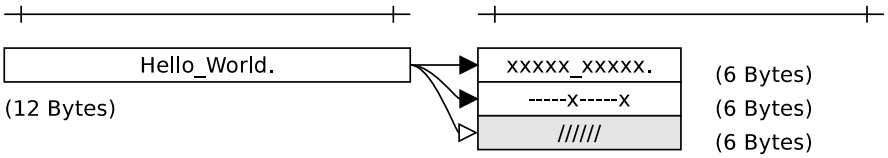

Abb. 3.9 Aufteilung von Daten in Bitketten für $k = 2, m = 1$

Wie in Abbildung 3.10 zu sehen, kann die Aufteilung nach Bits in verschiedenen Variationen vorgenommen werden. Ein bitvariabler oder byteweiser ($b = 8$ Bits) Datenstrom wird iterativ in Blöcke bzw. Chunks mit $w = 6$ Bits eingeteilt. Jeder Block wird in $k = 3$ Fragmente zerlegt. Die Fragmente werden in ihre jeweiligen Fragmentströme eingefügt. Für Anwendungen, die keine bytegrenzen-überschreitenden Fragmente verarbeiten können, lässt sich die Einhaltung der Grenzen über Füllfelder realisieren. Für alle $k = 2^x$ ist dies unnötig, während ansonsten bis zu $37,5\%$ des Fragmentstroms für Füllfelder benötigt werden, was einem zusätzlichen Platzbedarf von 60% gegenüber einer Optimalkodierung entspricht. Ist die Berechnung einer einfachen Redundanz per XOR-Verknüpfung über alle Fragmentströme vorgesehen, so müssen alle Fragmente die gleiche Größe aufweisen oder per Füllbits gleich groß betrachtet werden. Die Größe des redundanten Fragments entspricht dann der größten Fragmentlänge.

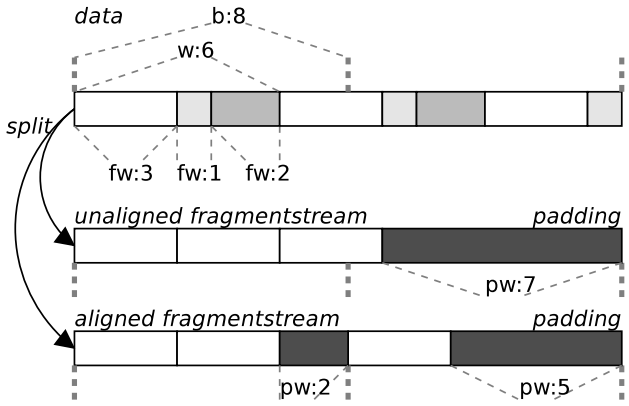

Abb. 3.10 Bitgrenzenanordnung und Füllfelder

3.1.3.6 Bitexpansion

Existierende Aufteilungsverfahren erlauben nur eine begrenzte Zahl verketteter Aufteilungen. Beispielsweise wurde JigDFS als datenschutzförderndes rekursiv verkettbares Verfahren speziell für dezentrale Anwendungen konzipiert, nutzt aber eine Löschkodierung mit $k = 4, m = 3$, was die Verkettungstiefe einschränkt [Bia10]. Hingegen sollte ein Verfahren so konzipiert sein, dass die Anwendung und somit der Anwender selbst über die Rekursionstiefe bestimmen kann. Aus diesem Grund wird in diesem Abschnitt ein Lösungsvorschlag zu dieser Problematik eingebracht und untersucht.

Das Aufteilungsverfahren nach Bits lässt sich mit Secret-Sharing-Ansätzen verbinden, um es rekursiv ohne inherente Abbruchbedingung durchführen zu können. Abbildung 3.11 zeigt exemplarisch die Aufteilung eines minimalen Fragments mit der Größe $fw_0 = 1$ Bit in 3 imaginäre *Drittelbits*, welche durch ein Byte repräsentiert werden, deren Maximum der Summen aller belegten und unbelegten Bits bei der Zusammensetzung wiederum den Wert des 1-Bit-Fragments ergeben. Dafür muss die Belegung eindeutig erfolgen. Ein Maß für die Belegung einer konkatenierten Fragmentmenge F ($F = \frown \{f_1, \ldots, f_n\}$) ist das Hamminggewicht $\mathscr{W}(F)$ als Spezialfall der Hammingdistanz zwischen der Fragmentmenge und einer Nullfragmentmenge $d(F, F_0)$.

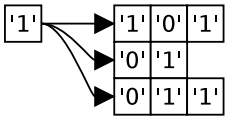

Abb. 3.11 Rekursive Aufteilung beliebig großer Fragmente

Die erste Anforderung an eine derartige rekursive Aufteilung ist somit ein resultierendes Hamminggewicht $\mathscr{W}$ verschieden von 50% der Blockgröße w als Summe aller Fragmentgrößen. Das Gewicht gibt in der Binärdarstellung die Zahl der belegten Bits an. Somit gilt für das Maximal- bzw. Mindestgewicht: $\mathscr{W}_{min}(0) < \frac{w}{2}$ und $\mathscr{W}_{min}(1) > \frac{w}{2}$. Zur Vereinfachung der Darstellung wird die Belegung des Ausgangsbits als $\mathfrak{B}$ und die Nullbelegung von F_0 als $\neg\mathfrak{B}$ bezeichnet. Somit muss stets gelten: $\mathscr{W}_{min}(\mathfrak{B}) > \frac{w}{2}$ (Anforderung 1).

Eine zweite Anforderung ist eine passende Verteilung der belegten Bits auf die Fragmente, so dass selbst bei Zusammenfügung von $k-1$ Fragmenten kein Rückschluss auf das tatsächliche Hamminggewicht ziehbar ist: $\forall q : \mathscr{W}(F) - \mathscr{W}(f_q) < \mathscr{W}(F), 1 \leq q \leq k, \mathscr{W}(F) = \sum_{i=1}^{k} \mathscr{W}(f_i)$ (Secret-Sharing-Regel). Jedes Fragment muss somit Träger einer signifikanten Information sein: $\mathscr{W}(fw_q) \geq 1$ (Anforderung 2). Dies bedeutet auch: $fw_q \geq 1$.

Gleichzeitig muss zur Einhaltung der Secret-Sharing-Regel auch gelten, dass bei Wegnahme eines Fragments die Zahl der verbleibenden Bits nicht mehr ausreicht, um $\mathscr{W}$ abzubilden: $w - fw_q < \mathscr{W}$. Das bedeutet aber, dass jede unvollständige Fragmentkonkatenation $F \setminus f_q$ mindestens ein Nullbit benötigt und somit $\mathscr{W} \neq w$ sein muss. Es muss also auch ein maximales Hamminggewicht $\mathscr{W}_{max}$ geben (Anforderung 3). Die Existenz eines Nullbits bedeutet weiterhin: $\exists q : f_q \geq 2$ sowie insbesondere $\mathscr{W}(fw_q) \geq fw_q$.

Für nahezu beliebige Parameter $w \in \mathbb{N}$ und $k \in \mathbb{N}$ lässt sich somit ein permissibler Hamminggewichtskorridor $\ulcorner\mathscr{W}\urcorner(\mathfrak{B}, k, w) = \{x \in \mathbb{N} | \mathscr{W}_{min} \leq x \leq \mathscr{W}_{max}\} = [\mathscr{W}_{min}, \mathscr{W}_{max}]$ erstellen, aus dem $\mathscr{W} \in \ulcorner\mathscr{W}\urcorner$ stets zufällig ausgewählt sein sollte, um die Vorhersage zu erschweren. Die zweite Anforderung impliziert als Vorbedingung eine das minimale Hamminggewicht nicht übersteigende Fragmentanzahl $k \leq \mathscr{W}_{min}$ (Bedingung 1). Weitere Implikationen sind $w > 2$, um eine Entscheidbarkeit zu erreichen, sowie $k > 1$, um die Information tatsächlich aufteilen zu können (Bedingungen 2 und 3). Für $k = 2$ ist die Ausführung des Algorithmus redundant, da ohnehin $fw < \mathscr{W}$ zusammen mit $\forall q : fw_{k-q+1} = w - fw_q$ gilt, d.h. $\mathscr{W}_{max} = w$.

Eine Formel zur Bestimmung von $\mathscr{W}_{max}$ oder ein vollständiger Algorithmus zur Generierung von k Fragmenten mit der Gesamtgröße w als Repräsentation für ein Bit $\mathfrak{B}$ sind aus der Literatur noch nicht bekannt. Aufgrund der potentiellen Nützlichkeit wird an der Stelle der Algorithmus zur Fragmentgenerierung eingeführt. Eine Erweiterung auf die Generierung von Fragmenten der Gesamtgröße w aus einem Fragment der Größe $w_{orig} < w$ wäre möglich, wird hier allerdings abgesehen von $w_{orig} = 1$ nicht betrachtet bzw. lässt sich stets durch vorherige Dispersion auf diesen Minimalfall abbilden.

Die Formel zur Berechnung von $\mathscr{W}_{max}$ leitet sich aus der Ungleichung 3.1 her, welche sich aus der Secret-Sharing-Regel ergibt. Die Ungleichung sagt aus, dass im optimalen Fall die Bits annähernd gleich verteilt sind und somit der Wegfall eines Fragments aus k im statistischen Mittel auch den Wegfall von $\frac{1}{k}$ belegten Bits bedeutet, womit gemäß der Secret-Sharing-Regel kein vollständiges Wissen selbst über $\mathscr{W}_{min}$ mehr vorhanden sein darf.

$$\mathscr{W}_{max} - \frac{\mathscr{W}_{max}}{k} < \mathscr{W}_{min} \tag{3.1}$$

In diese Ungleichung wird als Freiheitsgrad-Hilfsvariable die reverse Hamminggewichtskorridorbreite *mod* als Maß für die Anzahl der erforderlichen Nullbits $\neg\mathfrak{B}$ eingeführt: $w = \mathscr{W}_{max} + mod$. Damit entsteht Ungleichung 3.2. Je kleiner *mod*, desto höher die Variabilität durch einen breiten Korridor.

$$w - mod - \frac{w - mod}{k} < \mathscr{W}_{min} \tag{3.2}$$

Die Ungleichung sagt aus, dass nach Subtraktion der Anzahl aller mindestens notwendigen Nullbits und der weiteren Subtraktion der durchschnittlichen Zahl notwendiger Nullbits in einem Fragment (dem nach Annahme wegfallenden f_q) weniger als $\mathscr{W}_{min}$ Bits verbleiben. Jedes Bit, welches bis zur Darstellung von $\mathscr{W}_{min}$ noch fehlt, kann somit frei mit $\mathfrak{B}$ oder $\neg\mathfrak{B}$ belegt werden.

Durch Umstellung der Ungleichung nach *mod* ergibt sich die Ungleichung 3.3.

$$mod < \frac{k\mathscr{W}_{min} - (k-1)w}{-(k-1)} \tag{3.3}$$

Durch die Annahme eines minimalen Maximums für *mod* wird aus der Ungleichung eine Gleichung. Dabei muss im Fall, dass diese einen Modulus von 0 aufweist, die Zahl 2 subtrahiert werden, ansonsten 1, um die Gleichung sicher und nahe an der Ungleichungsgrenze zu erfüllen.

Schließlich ergibt sich daraus eine Berechnungsvorschrift für $\mathscr{W}_{max}$, die in mehreren Schritten algorithmisch umgesetzt werden kann, allerdings auch als einzelne Formel darstellbar ist, wie die Gleichung 3.4 zeigt.

$$\mathscr{W}_{max} = \left(\frac{k\mathscr{W}_{min} - (k-1)w)}{-(k-1)} + \begin{cases} 2 & \text{für}(w - ((\frac{k\mathscr{W}_{min}-(k-1)w)}{-(k-1)}))\%k = 0 \\ 1 & \text{sonst} \end{cases}\right. \tag{3.4}$$

Das Quelltextbeispiel 3.1 enthält den Algorithmus zur sicheren Bitexpansion in Pseudocode angelehnt an eine Python-Syntax. Er garantiert, dass im Rahmen der Einschränkungen die maximale Zufälligkeit der Bitverteilungen erreicht wird. Die Laufzeitkomplexität beträgt $\mathscr{O}(1)$ für die Berechnung der beiden Werte $\mathscr{W}_{min}$ und $\mathscr{W}_{max}$ und die Bestimmung einer Zufallszahl sowie $\mathscr{O}(w)$ für die Zuweisung der Bits. Der Faktor k hat keinen wesentlichen direkten Einfluss auf die Laufzeit, da er in Abhängigkeit von w begrenzt wird.

Die Brauchbarkeit des Algorithmus für die sichere Datenspeicherung oder -übertragung lässt sich aus der Befüllung der drei Korridore von F bestimmen. Der erste Korridor ist *Det* mit der Größe $|Det| = \mathscr{W}_{min}$. Der zweite Korridor ist $\ulcorner\mathscr{W}\urcorner$ mit der Größe $|\ulcorner\mathscr{W}\urcorner| = \mathscr{W}_{max} - \mathscr{W}_{min}$. Der dritte Korridor is *Mod* mit der Größe $|Mod| = mod$.

Umfassen die belegten Bits vollständig *Det*, so ist der Bitwert $\mathfrak{B}$ deterministisch entscheidbar, andernfalls nicht. Wird die Belegung von $\ulcorner W\urcorner$ echt zufällig durchgeführt, so handelt es sich um einen Secret-Sharing-Ansatz. Sind die Bits vorhersagbar

Listing 3.1 Fragmentgenerierung

```
def expandbit(𝔅, k, w):
  # Anf.1: minimales Hamminggewicht
  𝒲min = ⌊w/2⌋ + 1
  # Bed.1: nicht mehr Fragmente als minimales Hamminggewicht
  # Bed.2+3: Minimalwerte für k und w
  if k > 𝒲min or k < 2 or w < 3:
    return None
  # Anf.2: mindestens 1 gesetztes Bit pro Fragment
  fragments = [x + [𝔅] for x in [[]] * k]

  # Anf.3: Bestimmung des Hamming-Freiheitsgrades zwischen fixem
      Minimum und dynamischem Maximum
  premod = (k * 𝒲min - (k - 1) * w) / (k - 1) * -1
  mod = premod + (1, 2)[(w - premod) % k == 0]

  # Randomisierte Auswahl des endgültigen Hamminggewichts
  𝒲 = random.randint(𝒲min, w - mod) - k
  # Erzeugung der Bitzuweisungsliste, mind. 'mod' inverse Bits am
       Anfang
  bitlist = [¬𝔅] * (w - 𝒲 - k - mod) + [𝔅] * 𝒲
  bitlist = [¬𝔅] * (mod) + bitlist
  # Zuweisung der Bits im Rotationsverfahren
  for i in range(len(bitlist)):
    fragments[i % k].append(bitlist[i])
  # Weitere Randomisierung
  [random.shuffle(x) for x in fragments]
  random.shuffle(fragments)
  return fragments
```

oder sogar gleich $\mathfrak{B}$, so kann immer noch Information versteckt werden, es wird jedoch nur eine abgeschwächte (praktische) Vertraulichkeit erreicht. Sind schließlich auch die Bits von *Mod* belegt, findet eine unsichere Aufteilung statt.

Beispiele für unterschiedlich breite und belegte Korridore sind in der Abbildung 3.12 zu sehen.

Zur portablen Übertragung von Fragmenten mit der Größe $fw\%8 \neq 0$ bietet sich eine Umkodierung in eine menschenlesbare Alphabetsform an. Dies kann durch Verfahren wie Base16 (Hexadezimaldarstellung), Base64 oder BaseN geschehen [Jos03]. Für die Verarbeitung der Datenströme in verteilten Umgebungen wird dies beispielsweise dann notwendig, wenn die Daten in einem textorientierten Protokoll wie HTTP übertragen werden sollen. Dies erhöht wiederum den zusätzlichen Platzbedarf. Für eine Kodierung mit Base64 entspricht dieser Wert 33%, was unter Umständen eine Alternative zur Bytegrenzeneinhaltung mit Füllfeldern darstellt.

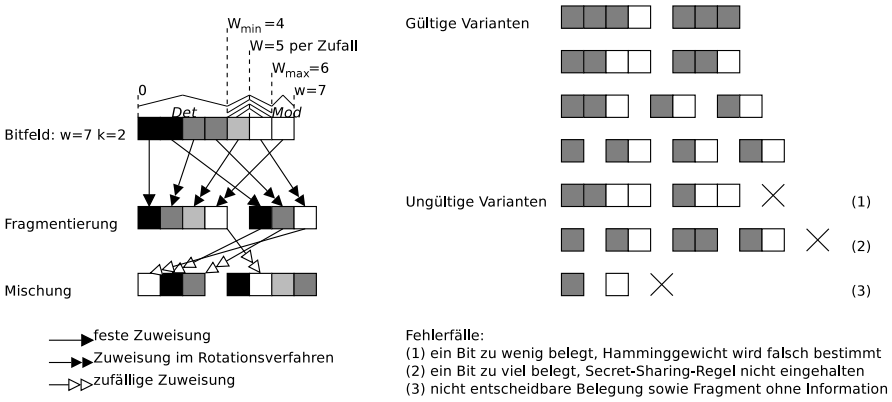

Abb. 3.12 Variationen des Hamminggewichtskorridors

3.1.3.7 Bewertung der Dispersionsverfahren

Die erstmalige weit gefasste Untersuchung unterschiedlichster Grundkodierungsverfahren verdeutlicht ein Spannungsfeld nichtfunktionaler Charakteristiken mit den Eckpunkten Ressourcenbedarf, Verfügbarkeit, Vertraulichkeit, Vollständigkeit, Strukturerhaltung und Interpretation von Datentypen und Dateninhalten.

Die Vielzahl an Kodierungs- und Aufteilungsverfahren mit ihren spezifischen Eigenschaften zeigt deutlich, dass je nach Anforderungen und Nutzungskontext eine geeignete Auswahl und Parametrisierung erforderlich ist. Aufgrund dynamischer Kontextwechsel muss dies ebenfalls dynamisch und flexibel zu beliebigen Zeitpunkten der Datennutzung geschehen können. Demnach lautet die Empfehlung, eine softwareseitige Unterstützung der Auswahl und Parametrisierung für datenverarbeitende Anwendungen und Dienste zu leisten. Mit dem beigetragenen Verfahren des Bitsplitting sowie der visuellen Aufteilung und anderer Interpolationsverfahren wird diese Schlussfolgerung gestützt.

Die Verfahren sind unterschiedlich strukturerhaltend bezüglich der Verarbeitung von durch sie kodierte Daten. Für feingranulare Datentypen wie Ganzzahlen, Fließkommazahlen oder Zeichenketten ist das Bitsplitting sowohl aus Vertraulichkeits- und Verfügbarkeitsgründen als auch hinsichtlich der möglichen nachgelagerten Verarbeitung empfehlenswert. Zudem beeinträchtigen sie teilweise auch die Verwaltung der Daten. Anfügeoperationen auf Chunks sind unter der Voraussetzung äquilonger Fragmente beispielsweise rechenintensiver als identische Anfügeoperationen auf Bitströmen, da im letzteren Fall nur die anzufügenden Teilfragmente berechnet werden müssen.

Tabelle 3.2 fasst die in diesem Abschnitt vorgestellten Kodierungsverfahren hinsichtlich ihrer Eigenschaften bezogen auf die weitere sichere Verarbeitung zusammen. Gesondert wird dabei das Bitsplitting-Verfahren mitsamt der rekursiven Bitexpansion und der einfachen Redundanz herausgestellt. Dessen Charakteristik kombiniert die vorteilhaften Kapazitäts- und feingranularer Verteilungseigenschaften der

Löschkodierung mit der vorteilhaften Vertraulichkeit von Secret-Sharing-Ansätzen und der nachgelagerten Verarbeitung replizierter Daten.

Tabelle 3.2 Vergleich von Verfahren zur Grundkodierung von Daten

Kodierungsverfahren	Strukturerhaltung	Rekursion	Verarbeitung	Redundanz
Blockbildung/Chunking	nichtdeterministisch	nein	bedingt	0%
Löschkodierung/Erasure Coding	nichtdeterministisch	nein	bedingt	0-100%,...
Replikation	ja	ja	ja	100%,200%...
Secret-Sharing	nein	nein	nein	0% / >0%
Interpolation	ja	nein	ja	0%
Bitsplitting	teilweise	ja	teilweise	1x (50%,...)

Ergänzend fasst Tabelle 3.3 weitere Eigenschaften der Dispersionsverfahren, insbesondere bezogen auf die Wiederherstellung der Daten, zusammen.

Tabelle 3.3 Eigenschaften von Verfahren zur Grundkodierung von Daten

Kodierungsverfahren	Minimale Fragmentzahl $\tilde{k}$	Ausfalltoleranz
Blockbildung/Chunking	$= k$	0
Löschkodierung/Erasure Coding	$= k$	m
Replikation	= 1	m
Secret-Sharing	$\geq k$	0 / m
Interpolation	$\leq k$	$k-1$
Bitsplitting	$= k$	1

3.1.4 Verschlüsselung von Daten

Zur Absicherung von Daten insbesondere mit Hinblick auf die Vertraulichkeit sind Verschlüsselungsverfahren unabdingbar. Sie tragen zudem bei der Verarbeitung personenbezogener Daten zur Ausweitung der Privatsphäre bei und sind somit Grundlage für diverse *privacy-enhancing techniques* (PETs). Im Folgenden werden nach einem kurzen Überblick über die Grundlagen solche Verfahren angesprochen, die beim Einsatz in verteilten Anwendungen auf nicht vertrauenswürdigen Ressourcen und Diensten für die Anwendungsarchitektur und -topologie von Bedeutung sind. Neben der konventionellen symmetrischen und asymmetrischen Verschlüsselung und den davon durch geeignete Parametrisierung abgeleiteten Spezialisierungen der konvergenten und suchenermöglichenden Verschlüsselung werden die Aspekte der ordnungserhaltenden und der homomorphen Verschlüsselung vorgestellt, welche strukturerhaltend eine nachgelagerte Verarbeitung in bestimmten Grenzen zulassen und somit in Kombination mit ebenfalls strukturerhaltenden Dispersionsverfah-

ren vorteilhaft gegenüber Secret-Sharing-Ansätzen ohne Verarbeitungsmöglichkeit sind.

3.1.4.1 Symmetrische und asymmetrische Verschlüsselung und Spezialfälle

Symmetrische und asymmetrische Verschlüsselungsverfahren (Chiffren, *ciphers*) sichern die Vertraulichkeit von Daten zu. Sie existieren in vielen Ausprägungen mit hinreichenden Untersuchungen [Sim79, SGI13], die sich jedoch mehr auf die Verschlüsselung an sich und weniger auf die Kombinierbarkeit mit weiteren Kodierungsverfahren beziehen.

Die symmetrische Verschlüsselung benötigt einen Schlüssel für die Ver- und Entschlüsselung von Daten. Da diese mitunter länger als der Schlüssel sind, lässt sich ein logischer Schlüssel durch Ringkonkatenierung des Ausgangsschlüssels (Blockchiffrier-Modus ECB), durch Einbeziehung in vorherigen Schritten chiffrierter Daten (Stromchiffrier-Modi CBC/CFB) oder anderen Techniken mit der Länge der Daten erzeugen. Der Schlüssel muss an alle geheimnistragenden Anwender und Anwendungen übermittelt werden, was neben digitalen und nichtdigitalen Übertragungen auch Aufgabe verschiedener Schlüsselverteilungsmechanismen sein kann [BD05]. Weiterhin muss er dauerhaft sicher aufbewahrt werden, wofür in verteilten Systemen wiederum Secret-Sharing-Ansätze geeignet sind.

Die asymmetrische Verschlüsselung zeichnet sich durch ein Schlüsselpaar mit zwei getrennten mathematisch zusammenhängenden Schlüsseln aus. Ein öffentlicher Schlüssel eines Empfängers wird für die Verschlüsselung auf Senderseite und die Entschlüsselung auf Empfängerseite genutzt. Ein privater Schlüssel des Senders kann zudem für eine kryptografisch gesicherte digitale Signatur genutzt werden. In der Praxis werden oft hybride Verfahren eingesetzt, wobei zuerst asymmetrisch der Schlüssel ausgetauscht und mit diesem dann symmetrisch verschlüsselt wird.

Die konventionellen symmetrischen und asymmetrischen Verfahren sind in der Regel per definitionem nicht strukturerhaltend, da jegliche Strukturübertragung eine Angriffsfläche bieten würde. Dennoch wird für bestimmte Szenarien verminderte Vertraulichkeit akzeptiert, wenn gleichzeitig durch die verbleibenden Strukturen funktionale Vorteile entstehen, womit ein Bedarf für Verfahren mit flexiblerer Abwägung zwischen Vertraulichkeit und Verarbeitungspotenzial generiert wird. Teilweise sind dafür angepasste Verschlüsselungsverfahren notwendig, teilweise ist hingegen die Parametrisierung der konventionellen Verfahren ausschlaggebend.

Die konvergente Verschlüsselung (*convergent encryption*) ist dafür ein Beispiel. Sie kommt zum Einsatz, indem bei einer symmetrischen Verschlüsselung ein stabiler Wert, der den zu verschlüsselnden Daten zugeordnet ist, als Schlüssel genutzt wird. Trivialerweise nutzt man den Hashwert der Daten. Die resultierende Funktion ist die Erkennung von Duplikaten. So kann beispielsweise ein Speicherdienstanbieter nutzerübergreifend Duplikate erkennen, durch Verweise ersetzen und somit Speicherplatz einsparen [SGLM08].

Die klassische suchenermöglichende Verschlüsselung (*searchable encryption*) ist ein weiteres Beispiel. Sie verbindet die symmetrische Verschlüsselung mit einem

speziell kodierten Schlüssel, so dass eine sichere Suche nach Schlüsselwörtern in einer unsicheren Umgebung ermöglicht wird [SWP00].

3.1.4.2 Ordnungserhaltende Verschlüsselung

Die ordnungserhaltenden Verschlüsselungsverfahren (*order-preserving encryption*, OPE) zielen auf numerische Daten mit zudem ganzzahliger Typisierung ab. Sie garantieren, dass die Ordnung von Zahlen durch die Verschlüsselung nicht gebrochen wird. Somit sind Vergleiche auf Gleichheit oder Ungleichheit, die Auswahl statistischer Momente wie Minimum, Maximum und Median sowie die Bildung von Verteilungsfunktionen direkt auf den verschlüsselten Daten möglich. Es existieren symmetrische ordnungserhaltende Verschlüsselungen wie das Boldyrewa-Schema [BCLO09] und das ältere OPES [AKSX04]. Eine grundlegende Eigenschaft dieser Verfahren ist es, dass die relativen Abstände zwischen den Zahlen im ordnungserhaltend verschlüsselten Raum ohne Kenntnis des Schlüssels nicht vorhersagbar und somit arithmetische Operationen auf ihnen nicht durchführbar sind.

3.1.4.3 Homomorphe Verschlüsselung

Mit der homomorphen Verschlüsselung (*homomorphic encryption*, HE) werden ebenfalls numerische Daten strukturerhaltend transformiert. Hierbei sind jedoch sowohl Ganzzahlen als auch Festkommazahlen, und damit per Konvertierung auch Fließkommazahlen, zulässig [FDH+10]. Einschränkungen gibt es jedoch unter anderem für negative Zahlen. Die Verfahren lassen eine bestimmte Menge an arithmetischen Operationen auf den verschlüsselten Zahlen zu, wobei diese im Vergleich mit ihren auf unverschlüsselten Daten arbeitenden Pendants mitunter weniger präzise und zumeist weniger effizient arbeiten. Ein Grund dafür ist die schlüssellängenabhängige Vergrößerung der Datentypen und damit auch der Datenmenge insgesamt. Ein asymmetrisches homomorphes Verschlüsselungsverfahren ist das Paillier-Cryptosystem [Pai99], auf welchem Additionen und Multiplikationen definiert sind. Homomorphe Verfahren haben die grundlegende Eigenschaft, aufgrund der Verschlüsselung keine Vorhersage über die unverschlüsselte Zahl zuzulassen, was impliziert, dass im homomorphen Raum die Ordnung nicht erhalten werden kann.

3.1.4.4 Bewertung der Verschlüsselungsverfahren

Die Erforschung kryptografischer Verfahren und Systeme (*cryptosystems*) umfasst zunehmend in Hinblick auf die weitere Verarbeitung der verschlüsselten Daten flexibilisierte Techniken, deren strukturerhaltende Eigenschaften bereits auf einzelne Verarbeitungsoptionen spezialisiert sind. Die durchführbaren Operationen auf ordnungserhaltenden Verfahren und homomorphen Verfahren schließen einander sogar

wechselseitig aus. Dies führt zur Notwendigkeit einer Mehrfachverschlüsselung, die bereits vorgeschlagen, aber bisher noch kaum umgesetzt worden ist [TKMZ13].

Wenige Untersuchungen gibt es bisher zur Verknüpfung der strukturerhaltenden Merkmale der Verschlüsselung mit denen der Aufteilung von Daten. Ein wichtiger Aspekt ist die einheitliche Verwaltung der Verfahrensparameter als Metadaten für die Durchführbarkeit der Reverskodierung.

3.1.5 Komprimierung und Steganografie

Die Komprimierung von Daten führt nicht unmittelbar zu einem Zuwachs an Sicherheit, aber durch die Reduktion des Kapazitätsbedarfs sowohl zu einer auch in ökonomischer Hinsicht signifikanten Optimierungsmöglichkeit bei der Auswahl von Diensten und Ressourcen als auch zu einer zusätzlichen Optimierung der Verfügbarkeit durch flexiblere Verteilungsstrategien. Ein Standardverfahren ist die Lauflängenkodierung (*run-length encoding*, RLE), bei der Folgen von Bytes mit einer bestimmten Länge so komprimiert werden, dass die Längenangabe als ein Wert mit einer gesetzten oberen Bitfolge und einem einzelnen Wertbyte erfolgt [Ans91].

Ähnlich zu den Verschlüsselungsverfahren gilt auch für die Komprimierungsverfahren eine prinzipielle Abwägung zwischen einerseits der Qualität, in dem Fall also primär der Komprimierungsrate und sekundär der Komprimierungs- und Dekomprimierungslaufzeit und des zugehörigen Speicherbedarfs, und andererseits der Verarbeitbarkeit der erzeugten komprimierten Daten [NT05].

Die Steganografie trägt zur Sicherheit bei, indem die Vertraulichkeit von Daten durch deren verborgene Einbringung in Trägerdaten erhöht wird. Somit entstehen hochgradig redundante Daten, die durch Dritte interpretiert werden können, ohne dass die verborgenen Daten dadurch sichtbar oder erfassbar würden [PH03, ZMS14, Wes06]. Eine nachgelagerte Verarbeitung kann diesen Schutz nur aufrecht erhalten, wenn auch die Verarbeitungslogik selbst verborgen bleibt oder ihren eigentlichen Zweck verschleiert (*obfuscation*).

3.1.6 Kombinierte Verfahren

Abbildung 3.13 fasst die vorgestellten Datenkodierungsverfahren Dispersion, Verschlüsselung, Komprimierung und Steganografie schematisch zusammen. In der Abbildung ist der Schwerpunkt der Arbeit auf dem Verfahren Dispersion sowie den Verfahrenskombinationen Dispersion mit Verschlüsselung und der vollständigen Überlagerung aller Verfahren markiert.

Prinzipiell lassen sich diese Verfahren beliebig kombinieren, um die Schutzwirkung zu maximieren. Wichtig ist dabei jedoch die Berücksichtigung von Einschränkungen insbesondere in Hinblick auf die Strukturerhaltung und die Selbstähnlichkeit von Datenstrukturen.

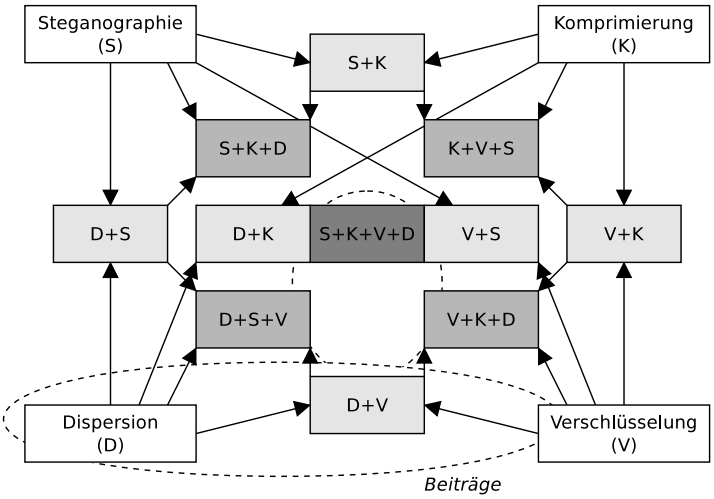

Abb. 3.13 Übersichtsschema zu Datenmodifikationsvarianten

So ist, wie in Abbildung 3.14 zu sehen, die vorherige Komprimierung zur Datenmengenreduktion, gefolgt von der Verschlüsselung und darauffolgend der Aufteilung, optimal bezogen auf die Geschwindigkeit der Kodierungsoperationen. Soll jedoch auf den entstehenden Fragmenten gerechnet werden, müssten sowohl die Komprimierung als auch die Verschlüsselung Selbstähnlichkeitseigenschaften aufweisen, so dass die Berechnung auf den Fragmenten möglich wäre. Dies ist jedoch nur sehr eingeschränkt der Fall. Für Lauflängenkomprimierung mit einer festen, zyklisch wiederholbaren, Kodierungslänge entsprechend der Fragmentlänge funktioniert dies. Für homomorph verschlüsselte Ganzzahlen funktioniert dies nicht, da die entstehenden Teilzahlen nicht mehr notwendigerweise im homomorphen Raum $\mathbb{H}$ liegen. Hingegen liegen Fragmente von Ganzzahlen stets wieder im gleichen Raum $\mathbb{Z}$.

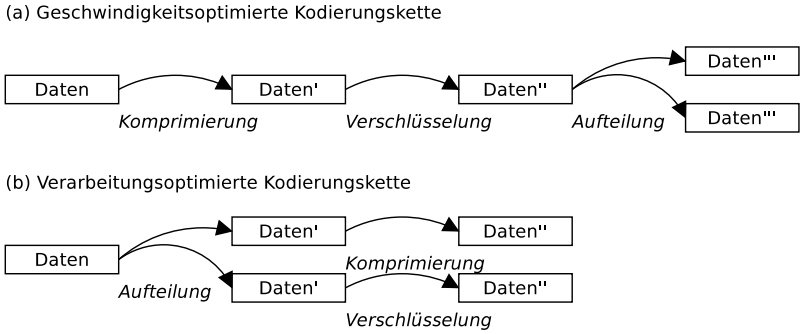

Abb. 3.14 Gegenüberstellung zweier Datenkodierungsketten *a* und *b*

Möglicherweise auftretende Fehler bei der Verknüpfung von Datenkodierungsoperationen sind in der Abbildung 3.15 abgebildet. Hierbei zerstört eine nachfolgende Dispersion die Struktureigenschaften *vergleichbar* und *dekomprimierbar*. Es ist allerdings möglich, Algorithmen für die Verschlüsselung oder die Komprimierung so zu entwerfen, dass diese Eigenschaften erhalten bleiben. Trivial ist dies möglich, indem bei der einfachen Lauflängenkodierung die Operation am letzten Bit vor der Fragmentgrenze nicht durchgeführt wird und bei der ordnungserhaltenden Verschlüsselung der verschlüsselte Wert mit sich selbst konkateniert wird. Diese Erweiterungen sind jedoch nicht optimal. Die Kombinierbarkeit von Kodierungsverfahren kann somit als neuartiges und noch nicht abschließend gelöstes Problem definiert werden, welches eine eigene Bezeichnung erhalten soll.

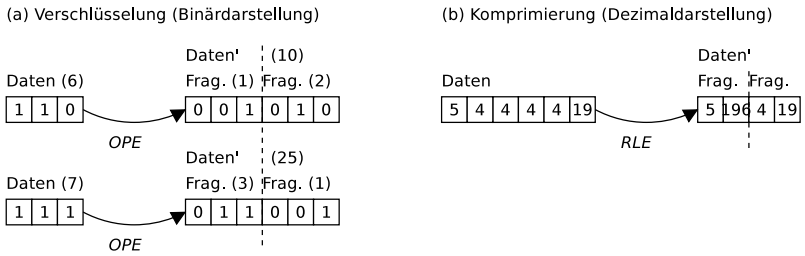

Abb. 3.15 Fehlerquellen bei der Verknüpfung zweier Kodierungsoperationen in einer Kette

Die gegen mehrere Risiken schützende verarbeitbare kombinierte Kodierung oder Mehrfachkodierung von Daten sei hiernach als *Stealth-Kodierung* bezeichnet.

3.2 Datenverteilung

Die kodierten Daten stehen als Menge von n Fragmenten bzw. Replikaten zur Verfügung. Diese sollen anschließend auf n aus h Speicherzielen oder äquivalenten Zieldiensten als Datensenken transferiert werden. Dabei gilt es, zuerst die Probleme der Ordnung und der Verteilung zu lösen, bevor anschließend der tatsächliche Transfer stattfinden kann. Die Ordnung bezeichnet dabei die Anordnung von zusammengehörigen Fragmenten aus einer Fragmentmenge, die prinzipiell jeder Mutation der geordneten Elementliste der Menge entsprechen kann. Dabei wird der Begriff Fragmentstrom eingeführt. Die Verteilung bezeichnet anschließend die Zuweisung von Fragmentströmen an Speicherziele.

3.2.1 Ordnung der Datenfragmente

Ein Fragmentstrom sei definiert als ein Datenstrom, der mit oder ohne Füllzeichen eine Folge von Fragmenten erhält. Fragmentströme werden durch eine Ordnungsfunktion aus den Fragmentmengen gebildet.

Das kanonische Verfahren weist stets das erste Fragment einer Fragmentmenge dem ersten Strom, das zweite dem zweiten u.s.w. zu. Dies hat Vorteile und Nachteile. Ein Vorteil ist die Vorhersagbarkeit, dass beispielsweise bei Einsatz des Bitsplitting-Verfahrens stets die Fragmente mit der höchsten Bit-Signifikanz auf einem Speicherziel liegen. Ein zweiter Vorteil ist die einfache Verwaltung. Ein Nachteil ist hingegen, dass im Falle einer Beeinträchtigung des signifikantesten Fragmentstroms möglicherweise ein vollständiger Datenverlust eintritt. Das Helixverfahren balanciert hingegen die Fragmente so aus, dass jeder Fragmentstrom in etwa die gleiche Menge an signifikanten Fragmenten erhält. Durch die Regelmäßigkeit der Helixstruktur ist der Verwaltungsaufwand gleichsam gering. Der Vollständigkeit halber sei noch eine Zufallsordnung erwähnt, die im statistischen Mittel eine ähnlich balancierte Verteilung erreicht, jedoch für jede Fragmentmenge einen separaten Metadaten-Verwaltungseintrag zur Wiederherstellung erfordert.

Abbildung 3.16 veranschaulicht die drei unterschiedlichen Ordnungsverfahren für Datenfragmente sowohl schematisch als auch anhand exemplarischer Fragmentzuteilungen auf die Fragmentströme. Die Ströme sind jeweils horizontal, die k signifikanten Fragmente aus jedem Kodierungsschritt jeweils vertikal eingetragen. Die Verfahren funktionieren analog für die Ordnung aller n Fragmente.

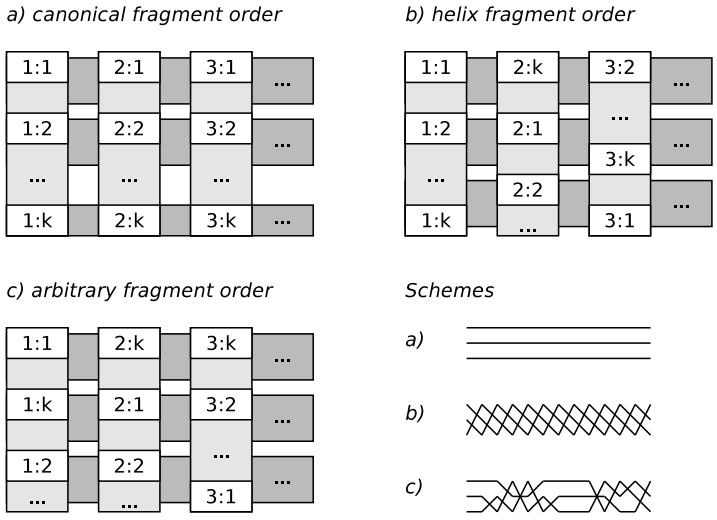

Abb. 3.16 Ordnungsmethoden für Datenfragmente

3.2.2 Einzelplatzierung und Round-Robin-Verteilung

Die Einzelplatzierung als triviale Verteilung leitet sich aus einer $1:1$-Verteilung aus einer Datenquelle und einem Speicherziel ab. Hierbei werden die Fragmente aus einem geordneten Fragmentstrom an ein einzelnes Speicherziel übermittelt, wobei $k = n$ und insbesondere $k = n = 1$ sinnvoll ist, da Redundanzen stets zusammen mit den signifikanten Daten von Ausfällen betroffen sind. Eine Variante der Einzelplatzierung ist die Round-Robin-Verteilung, welche aus einer $1:n$-Verteilung von Speicherzielen hervorgeht. Hierbei werden die n Fragmente in einer statischen und zyklischen Zuteilung an die n Speicherziele übertragen.

Die Einzelplatzierung wird in Abbildung 3.17 für $h = 5(s_1,...,s_5)$ und $n = 5(f_1,...,f_5)$ veranschaulicht. Sie wird nur der Vollständigkeit halber dargestellt, da sie nicht zu einer verteilten Datenhaltung führt und somit keinen Mehrwert liefert. Die Round-Robin-Verteilung wird in Abbildung 3.18 gegenübergestellt.

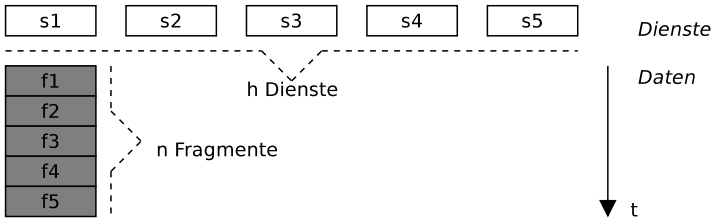

Abb. 3.17 Verteilung von Daten per Einzelplatzierung

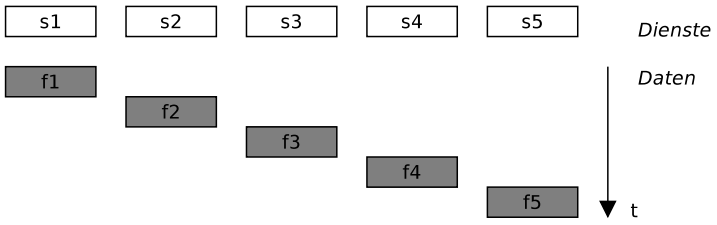

Abb. 3.18 Verteilung von Daten per Round-Robin-Schema

3.2.3 Mehrfachplatzierung und Replikation

Die Mehrfachplatzierung bildet jedes Fragment auf mehr als ein Speicherziel ab. Die Daten werden entweder parallel an alle n Speicherziele oder an eine balancierte Auswahl von k aus n Zielen übermittelt. Zur Balancierung können Hashringe

respektive Hashtabellen verwendet werden, welche die Zielauswahl basierend auf dem Hashwert des zu übermittelnden Wertes durchführen. Die beiden Verteilungsverfahren werden in Abbildung 3.19 gegenübergestellt.

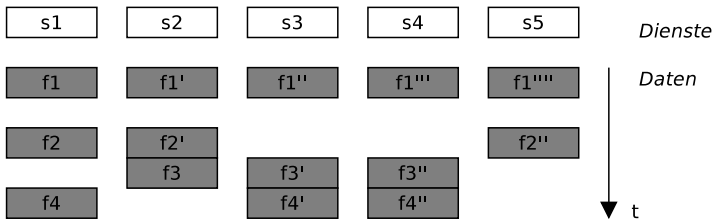

Abb. 3.19 Verteilung von Daten per Vollreplikation und per Hashring

3.2.4 Verteilung zur Erzielung einer Mindestverfügbarkeit

In nahezu allen offenen verteilten Umgebungen unterscheiden sich die datenverarbeitenden Speicher-, Rechen- und Kommunikationsdienste erheblich in ihren nichtfunktionalen Eigenschaften. Um bei der Verteilung von aufgeteilten Daten die bestmögliche Kombination zu erzielen, müssen die in Frage kommenden Dienste ausgewählt, priorisiert und parametrisiert werden. Die vorgestellten einfachen Platzierungsverfahren, welche unabhängig von den Diensteigenschaften arbeiten, sind dafür ungeeignet.

Für die Speicherung dispergierter Daten wurde hierbei das PICav-Verfahren entwickelt, welches präzise und iterativ basierend auf bekannten oder angenommenen Werten für die Verfügbarkeit und die Kapazität der Speicherziele bzw. -dienste eine Verteilung und eine daran gekoppelte Gesamtverfügbarkeit ermittelt [SM14a]. An dieser Stelle soll es zusammen mit weiteren Verfahren untersucht werden und aufbauend ein verbessertes Verfahren mit dem Namen PICav+ konstruiert werden.

Abbildung 3.20 zeigt die wechselseitigen Abhängigkeiten zwischen der Gesamtanzahl der Speicherziele n, der Anzahl der signifikanten und redundanten Speicherziele k und m, sowie der resultierenden Verfügbarkeit a und der resultierenden Nettokapazität $\hat{c}$ für eine angenommene mittlere Verfügbarkeit jedes Speicherziels von $a = 0,5$. Die entstehenden charakteristischen Verfügbarkeitskurven steigen einerseits monoton gegen $a = 1,0$ konvergierend an, andererseits sinken sie gespiegelt monoton gegen $a = 0,0$ konvergierend ab. Die Ableitung der gedachten Funktionen a' repräsentiert dabei das Optimierungsziel: hohe Verfügbarkeit ($a' > 0$) oder geringer Platzbedarf ($a' < 0$). Offensichtlich existiert dabei ein nichtlinearer Zusammenhang zwischen a und c, der als dediziertes Hilfsverfahren die Bestimmung der Gesamtverfügbarkeit für jede Konfiguration erfordert.

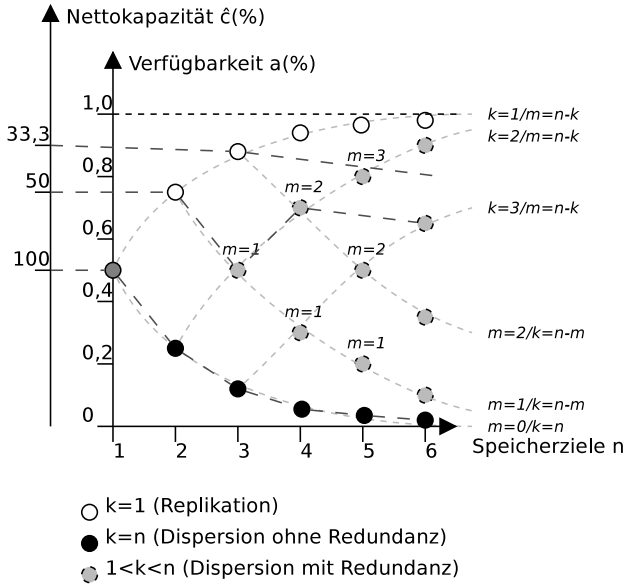

Abb. 3.20 Charakteristische Verfügbarkeits- und Kapazitätskurven abhängig von Redundanz und Zahl der Speicherziele

3.2.4.1 Bestimmung der Gesamtverfügbarkeit

Für Mengen an Speicherzielen mit der Kardinalität n lassen sich basierend auf deren individuellen Eigenschaften die Gesamteigenschaften bestimmen. Dabei gilt es zu unterscheiden, ob die individuellen Eigenschaften homogen oder annhähernd homogen ausgeprägt sind, oder ob eine starke Heterogenität anzunehmen ist.

Die Gesamtkapazität lässt sich im homogenen Fall über $c = nc_1$ berechnet. Ähnlich wird der Gesamtpreis über $p = np_1$ berechnet. Im heterogenen Fall muss zuerst ein Durchschnittswert oder die vollständige Summe gebildet werden, so dass gilt: $c = n\bar{c} = \sum_{i=1}^{n} c_i$ und $p = n\bar{p} = \sum_{i=1}^{n} p_i$.

Die Verfügbarkeit homogener Dienste wird nach [PJGLSAH11] über die Formel 3.5 bestimmt. Für die Verfügbarkeit heterogener Dienste gilt nach [PJGLSAH11] bzw. [SM14a] eine komplexere Berechnungsvorschrift nach Formel 3.6. Über die Potenzmenge aller als verfügbar angenommenen Dienstkombinationen C mit mindestens k Diensten $\mathscr{P}_f(C)$ wird iteriert, wobei in jedem Schritt die Verfügbarkeit als Produkt der Einzelverfügbarkeiten der Menge S und der Einzelverfügbarkeitskomplemente der als nicht verfügbar angenommenen Komplementärmenge $C \setminus S$ gebildet wird.

$$availability = \sum_{i=k}^{n} \binom{n}{i} a_1^i (n - a_1)^{n-i} \tag{3.5}$$

$$availability = \sum_{S \in P_{>k}(C)} \left(\prod_{i \in S} a_i \cdot \prod_{i \in C \setminus S} 1 - a_i \right) \tag{3.6}$$

Aufgrund der Berechnungskomplexität von Formel 3.6 existiert das Hilfsverfahren zur Bestimmung der Gesamtverfügbarkeit in zwei Varianten. Einerseits wird das Ergebnis mittels des kombinatorischen Ansatzes der Potenzmenge exakt ermittelt, andererseits wird es durch eine Monte-Carlo-Simulation approximiert [PJGLSAH11].

3.2.4.2 Formulierung des Optimierungsproblems

Gesucht ist eine optimale Verteilung aller Datenfragmente auf alle Kandidatendienste zur Erzielung von nutzerdefinierten Mindest- oder Höchstwerten der Verfügbarkeit, des Kapazitätsbedarfs und der Kosten bei gleichzeitiger Einschränkung der Laufzeit für die Suche. Somit entsteht ein multikriterielles Optimierungsproblem, welches als *Multiple Service Multiple Data Multi-Objective Optimisation (MS-MD-MOO)* bezeichnet werden soll. Konkret handelt es sich um ein gewichtetes trikriterielles Problem mit einer Optimierungszielfunktion und bis zu vier Nebenbedingungen gemäß Gleichung 3.7. In der Gleichung werden die Priorisierungen der Verfügbarkeit a, der Kapazität c und des Preises p mit den Parametern w gewichtet den einheitenlosen Minimal- und Maximalwerten dieser Größen sowie der angenommenen maximal zulässigen Laufzeit der Verteilungsbestimmung $\tilde{t}$ entgegengesetzt.

$$\begin{aligned} MS-MD-MOO := & max & cw_c + aw_a - pw_p \\ & unter & t \leq \tilde{t} \\ & & a \geq \tilde{a} \\ & & c \geq \tilde{c} \\ & & p \leq \tilde{p} \end{aligned} \tag{3.7}$$

Aufgrund der bedingt durch die Nichtlinearität zu erwartenden Berechnungskomplexität [UU12] ist eine vollständige Verteilungsbestimmung in Echtzeit ($t \leq \tilde{t}$) mit steigender Dienstzahl zunehmend nicht mehr möglich. Desweiteren lassen sich existierende generische MOO-Lösungsverfahren wie beispielsweise der linearen Optimierung (*multi-objective integer linear programming*, MOILP) oder der Software-Qualitätsanpassung *Multi-Quality Auto-Tuning* (MQuAT) [Gö13] nur bedingt anwenden, da sie die spezifischen Merkmale der Verteilungsbeschränkungen signifikanter und redundanter Fragmente und die Nichtlinearität der Bestimmung von a, c und p nicht berücksichtigen.

Aus diesem Grund existieren mehrere dedizierte Verfahren, die im Folgenden zusammen mit verständlichen Trivialverfahren und der vollständigen Kombinationssuche vorgestellt und bewertet werden, bevor anschließend ein verbessertes Verfahren vorgestellt wird. Die Untersuchung erfolgt exemplarisch für $n = 4$ Fragmen-

te, wobei das jeweilige Verteilungsverfahren die Fragmentierung in k und m sowie die Anzahl $\hat{n} \leq n$ der letztlich genutzten Dienste bestimmt, und $h = 4$ Kandidatendienste mit den in Tabelle 3.4 aufgeführten Parametern, die bewusst ohne Einheiten angegeben sind. Die Werte sind dennoch realistisch gewählt; so stellt D_3 etwa einen nur sporadisch verfügbaren Dienst auf einem mobilen Gerät dar. Der Preis bezieht sich jeweils auf eine Kapazitätseinheit pro (nicht betrachteter) Zeiteinheit. Neben diesem nutzungsgradabhängigen Kostenmodell wären noch Festpreisangebote mit ebenso festgelegter Kapazität in die Optimierung aufzunehmen, worauf im Beispiel und in der Erarbeitung des Algorithmus nicht weiter eingegangen wird. Gilt $n \neq \hat{n}$, so ist eine sichere Datenverteilung mit Secret-Sharing-Eigenschaften nicht möglich.

Tabelle 3.4 Beispielsdienste zur Untersuchung der Fragmentverteilung

Dienst	Kapazität c_i	Verfügbarkeit a_i	Preis p_i
D_1	10	0,96	1,00
D_2	40	0,3	0,00
D_3	92	0,99	0,30
D_4	120	0,98	0,24

3.2.4.3 Staffelung zur Berücksichtigung heterogener Kapazitäten

Nicht nur die Verfügbarkeiten, sondern auch die Kapazitäten werden als heterogen angenommen. Damit wird ein Verfahren benötigt, welches gestaffelt die Teilverfügbarkeiten für diejenigen Dienstmengen bestimmt, welche in der jeweiligen Kapazitätsstaffel beteiligt sind. Da ein solches Verfahren noch nicht bekannt ist, wird es an dieser Stelle erarbeitet.

Die Gesamtkapazität liegt im Beispiel bei $c = 262$. Daraus geht die maximalverfügbarkeitsabhängige Effektivkapazität $\hat{c} = 40$ durch die gleichmäßige unabhängige Verteilung von $k = 2$ und $m = 2$ hervor, bei deren Überschreitung die Fragmente nicht mehr derart anteilig gespeichert werden können und somit zumeist eine Reduktion der Verfügbarkeit die Folge wäre. In solchen Fällen muss die veränderliche und zumeist degradierte kapazitätsabhängige effektive Gesamtverfügbarkeit $\hat{a}$ als Summe abschnittsweise veränderlicher Einzelverfügbarkeiten berechnet werden, was in Abbildung 3.21 skizziert ist. Dieser Ablauf sei als gestaffelte Verteilungsbestimmung respektive als gestaffelte effektive Verfügbarkeits-, Kapazitäts- und Kostenbestimmung bezeichnet.

3.2.4.4 Trivialverfahren: Gleich-, Proportional- und Absolutverteilung

Das Gleichverteilungsverfahren berücksichtigt keine angestrebten Garantien oder Optimierungsziele, sondern weist jeder Datensenke gleiche Datenmengen und somit bei angenommenen gleichgroßen Fragmenten auch eine gleiche Fragmentanzahl zu.

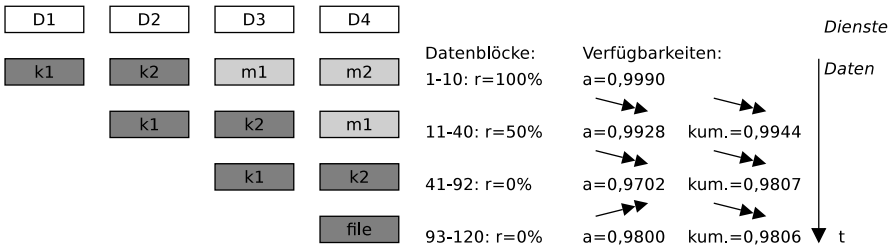

Abb. 3.21 Veränderlichkeit und Degradierung der Verfügbarkeit bei höherer Auslastung der Kapazität, resultierend im gewichteten kumulativen Mittelwert $\hat{a} = 0,9806$

Die Gleichverteilung mit maximaler Redundanz ($k = 1, m \geq 1$) entspricht der vollständigen Replikation, die Gleichverteilung ohne Redundanz ($k \geq 1, m = 0$) hingegen einer reinen Aufteilung. Das Proportionalverteilungsverfahren stellt demgegenüber zumindest insofern eine Verbesserung dar, als dass die Zuweisung proportional zu bestimmten Diensteigenschaften, also den zur Verfügung stehenden Kapazitäten, Einzelverfügbarkeiten oder Preisen, bestimmt wird. Das Absolutverteilungsverfahren weist bei vorhandener Kapazität die Daten ausschließlich dem Dienst mit der höchsten Verfügbarkeit, der höchsten Kapazität oder des niedrigsten Preises zu.

Das Gleichverteilungsverfahren erzielt bezogen auf das Beispiel eine maximale Auslastungsquote C von $15,2\%$ ($c = \hat{c} = 40$), bevor der kleinste Dienst mit $c_1 = 10$ voll ausgelastet und eine weitere Gleichverteilung nicht mehr möglich ist. Die Kosten liegen dabei bei $p = 15,40$ absolut und $\frac{p}{c} = 0,385$ normiert. Das Proportionalverteilungsverfahren nach Kapazität erzielt durch feingranularere Aufteilung auf $n = 40 * 0,1$ nach dem Gewichtungsschlüssel $1/6/14/19$ letztlich $C = 91,6\%$ ($c = 240$, $c_i = 6$) bei $p = 58,56$ und $\frac{p}{c} = 0,244$, was bei Nichtberücksichtigung der Fragmentsemantik sowohl wesentlich günstiger (36%) als auch leistungsstärker (500%) als das Ergebnis der Gleichverteilung ist, andernfalls aber mit $\hat{C} = 9,2\%$ ($\hat{c} = 24$) schlechter abschneidet. Demgegenüber wird für das Proportionalverteilungsverfahren nach Einzelverfügbarkeit der Gewichtungsschlüssel $12/4/12/12$ angesetzt und aufgrund der geringen Kapazität c_1 auf $10/3/10/10$ gekürzt. Dadurch wird mit $C = 13,0\%$ ($c = 33$, $c_i = 1$) nur eine sehr geringe Auslastung und mit $p = 15,40$ und $\frac{p}{c} = 0,467$ ein höherer Kostenfaktor im Vergleich mit der Gleichverteilung erreicht. Die Effektivauslastung beträgt hier lediglich $\hat{C} = 4,58\%$ ($\hat{c} = 12$). Wird schließlich die Proportionalverteilung nach dem absoluten Preis vorgenommen, ergäbe sich im Beispiel der Sonderfall, dass mit der Gewichtung $0/40/0/0$ ausschließlich D_2 berücksichtigt wird und trivialerweise $C = 15,2\%$ ($c = 40$) sowie $p = 0,00$, $\frac{p}{c} = 0,000$ und $a = a_{min} = 0,3$ erzielt werden. Wenn man die Preise hingegen abstandsproportional zum Höchstpreis ansieht, würde sich die Gewichtung auf $0/16/11/13$ ändern. Mit ihr wird immerhin $C = 30,5\%$ ($c = 80$, $c_i = 2$), absolut $p = 9,72$ und normiert $\frac{p}{c} = 0,122$ erzielt. In beiden Fällen der Proportionalverteilung nach Preis findet keine unabhängige Verteilung aller Fragmente statt, woraus $\hat{C} = 0,0\%$ ($\hat{c} = 0$) resultiert.

Die Gesamtverfügbarkeit aller Daten unter Berücksichtigung der Semantik redundanter Fragmente liegt bei der Gleichverteilung bei $a = a_{max} = 0,99903$ respektive $99,903\%$, hingegen bei der abschnittsweisen Degradierung der redundanten Fragmente bei $a = 0,9806$. Für die Proportionalverteilungen ist die entsprechende vollständige Fragmentverteilung nicht einheitlich bestimmbar und die Verfügbarkeit somit nicht einheitlich berechenbar. Eine zweiteilige Berechnung ist dann möglich, wenn man die Annahme trifft, dass alle Daten über die bestimmte Verteilung innerhalb der Effektivkapazität $\hat{c}$ hinaus redundanz- und fragmentfrei abgelegt werden, also $k = 1$ und $m = 0$ gilt. Die Effektivverfügbarkeit wird in dem Fall über die Formel 3.8 bestimmt, wobei $\hat{c}_i = \hat{c}/h$ gilt. Die Berechnung mit der Formel liefert eine Verfügbarkeit für die Proportionalverteilung nach Kapazität $\hat{a} = 0,8965$, nach Einzelverfügbarkeit $\hat{a} = 0,8877$ und nach Preis $\hat{a} = a_{min} = 0,8789$.

$$\hat{a} = \frac{\hat{c}a_{max} + \sum_{i=1}^{h}(c_i - \hat{c}_i)a_i}{c} \tag{3.8}$$

Tabelle 3.5 fasst die mit den zusicherungsfreien Verteilungsverfahren erzielten Verfügbarkeiten zusammen.

Tabelle 3.5 Verteilungen ohne Zusicherung einer Mindestverfügbarkeit mit der Gleich-, Proportional- und Absolutverteilung

Verteilungsverfahren	Erreichte Verfügbarkeit a	Preis	Kapazitätsoverhead
Gleichverteilung ohne Redundanz	0,2794	1,54	0,0
Gleichverteilung mit Redundanz	0,9990	1,54	1,0
Proportional nach Verfügbarkeit	0,9989	1,54	1,0
Proportional nach Kapazität	0,9857	1,54	1,0
Proportional nach Kosten	0,9791	1,54	1,0
Proportional nach Verfügbarkeit effektiv	0,8877	1,54	variabel
Proportional nach Kapazität effektiv	0,8965	1,54	variabel
Proportional nach Kosten effektiv	0,8789	1,54	variabel
Absolut nach Verfügbarkeit	0,9900	0,30	0,0
Absolut nach Kapazität	0,9800	0,24	0,0
Absolut nach Kosten	0,3000	0,00	0,0

Die mit den Gleich- und Proportionalverteilungsverfahren bestimmten Verteilungen werden anhand ihrer Eigenschaften in der Abbildung 3.22 visuell gegenübergestellt und verglichen.

Durch eine adaptive Verschiebung der Gleichverteilungsgrenze können die erzielten Charakteristiken der Proportionalverteilungen nutzerkontrollierbar verändert und ineinander überführt werden. Im Beispiel liegen die erzielbaren Gesamtverfügbarkeiten stets zwischen $\hat{a}_{min} = 0,8789$ und $\hat{a}_{max} = 0,8965$. Niedrigere Verfügbarkeiten (bis $a_{min} = 0,3$) und Kosten sind dann möglich, wenn zu Lasten der Kapazität nicht alle Dienste eingebunden werden, und höhere (bis $a_{max} = 0,99903$), wenn Überkapazitäten unberücksichtigt bleiben und nur die Effektivkapazität ausgenutzt wird. Abbildung 3.23 stellt den Zusammenhang vereinfacht für Dienste mit identischen Einzelverfügbarkeiten dar.

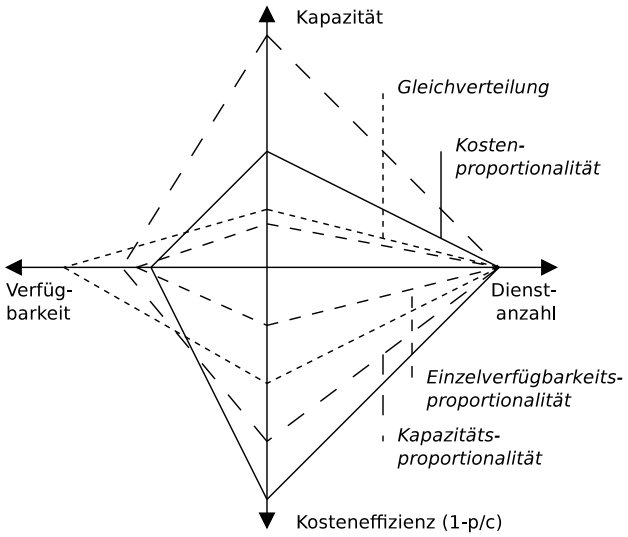

Abb. 3.22 Mehrdimensionale Sicht auf durch Verteilungsverfahren erzielbaren wünschenswerten NFE

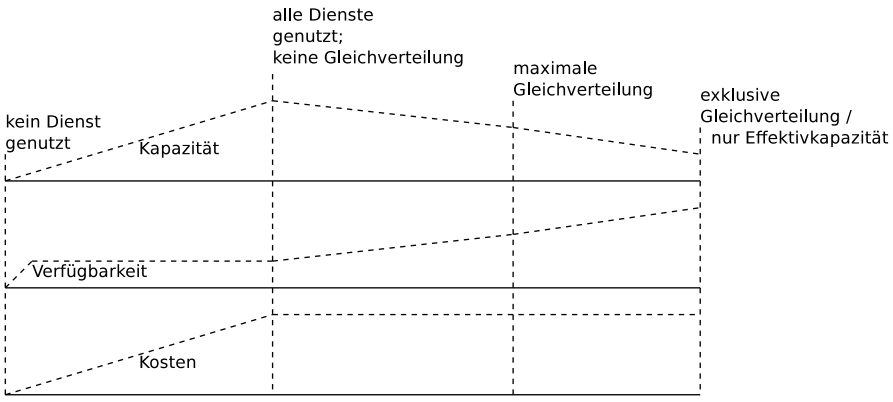

Abb. 3.23 Verteilungsbezogener Zusammenhang zwischen Kapazität, Verfügbarkeit und Kosten

Der Vergleich der gestaffelten Gleichverteilungen mit Redundanz am Beispiel ist im Abbildung 3.24 zu sehen.

Ergänzend sei auch an dieser Stelle die zufällige Verteilung von Fragmenten erwähnt. Die Verfahren der Gleich-, Proportional-, Absolut- und Zufallsverteilung zeichnen sich allesamt durch mangelnde Zusicherung, gleichzeitig jedoch geringe Laufzeitkomplexität aus.

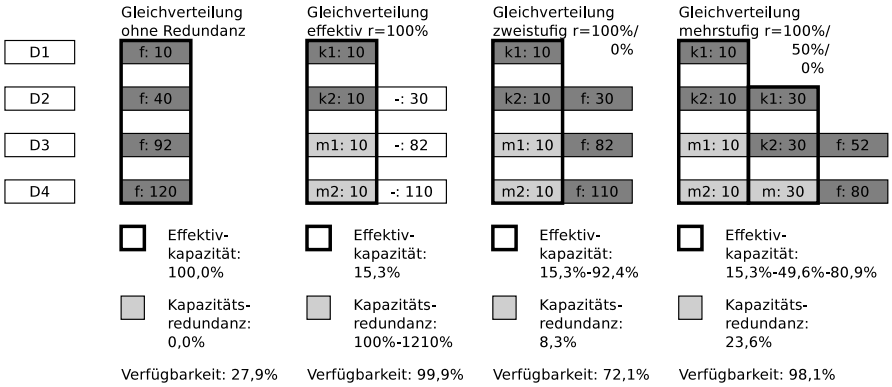

Abb. 3.24 Vergleich von Verteilungen mit verfügbarkeitsabhängigen Effektivkapazitäten

3.2.4.5 Kombinatorische Verteilungssuche

Im Gegensatz zum Trivialverfahren stellt die kombinatorische Suche den Gegenpol der Optimierung dar. Das Optimum wird durch sie stets gefunden, jedoch nur nach vollständiger Evaluierung sämtlicher Verteilungskombinationen. Jedoch kann eine nicht kostenoptimale erste Lösung schnell gefunden werden, so dass sich das Verfahren für den Einsatz in einer inkrementellen Verbesserung der Verteilung empfiehlt.

Wird nur die Verfügbarkeit bei Vernachlässigung von Preis und Kapazität als Optimierungsziel eingesetzt, so ergeben sich für eine gewünschte Mindestverfügbarkeit $\tilde{a}$ die in der Tabelle 3.6 eingetragenen Verteilungen. Sofern ein Dienst, im Beispiel D_4, die Mindestverfügbarkeit allein erfüllt und zudem am günstigsten ist, gibt es aus der Optimierungsperspektive keinen Grund, die Daten über mehrere Dienste zu verteilen.

Tabelle 3.6 Verteilungen zur Zusicherung einer Mindestverfügbarkeit mit der Kombinationssuche

Mindestverfügbarkeit $\tilde{a}$	Tatsächliche Verfügbarkeit a	Preis	Kapazitätsoverhead
$\geq$ 0,0000	0,9800	0,24	0,0
$\geq$ 0,9801	0,9860	0,24	0,25
$\geq$ 0,9861	0,9900	0,30	0,0
$\geq$ 0,9901	0,9930	0,30	0,25
$\geq$ 0,9931	0,9998	0,54	0,25
$\geq$ 0,9999	˜1,0000	1,54	0,50

Die gestaffelte Kombinationssuche ist zudem in der Lage, stets die optimale Verteilung für eine gewünschte Mindestkapazität über Dienste mit heterogenen Kapazitäten zu finden. Dieses Verfahren ist somit qualitativ ohne Berücksichtigung der Laufzeit die Referenz für mögliche Optimierungen der Bestimmungsverfahrens an sich bezogen auf deren Laufzeit und schnelle Konvergenz.

3.2.4.6 Stand der Technik: Optimierte Verteilungssuchverfahren

Mit Hilfe einer Partikelschwarmoptimierung erreichen Pamies-Juarez et al. eine Verteilungsstrategie für heterogene Dienstlandschaften mit unterschiedlichen Verfügbarkeiten [PJGLSAH11]. Es erfolgt zwar keine explizite Zusicherung einer Mindestverfügbarkeit, der Algorithmus findet aber mit hoher Wahrscheinlichkeit das Maximum bei gleichzeitig größtmöglicher Reduktion der Redundanz um bis zu 70%. Darüber hinaus werden Kapazitäten, Preise oder Laufzeiteinschränkungen jedoch nicht berücksichtigt. Durch die Formulierung als Bin-Packing-Problem schlägt Hadji ebenfalls eine Optimalverteilung vor, welche die Speicherkosten berücksichtigt [Had15]. Sind die Kapazitäten unterschiedlich, gelten die Garantien allerdings nur bis zur kleinsten Kapazität. Desweiteren berücksichtigt das Verfahren nur die vollständige Replikation, d.h. $k = 1, r \geq 1$. Die eigene Vorarbeit *Precise, Iterative and Complement-based Cloud Storage Availability Calculation Scheme* (PICav) verbessert die Berechnung gegenüber der Schwarmoptimierung in einem zweistufigen Verfahren, welches zuerst inkrementell exakte Werte liefert und, falls ohne Erreichen des Optimumgs keine weitere Verbesserung schnell erzielt werden kann, auf Näherungswerte zurückfällt [SM14a]. Im ersten Verfahren werden jedoch heterogene Kapazitäten überhaupt nicht, im den zwei weiteren Verfahren ebenfalls nur teilweise berücksichtigt. Unterschiede im Preis werden nur in einem Verfahren als Kriterium hinzugezogen.

Tabelle 3.7 zeigt vergleichend zur Kombinationssuche die Resultate der PICav-Verteilungssuche an. Der Vergleich zeigt deutlich, dass die gefundenen Kombinationen nicht vollständig sind und bereits eine geringe Verfügbarkeit von 28% mit einem Nettokapazitätsverlust bezahlt werden muss. Ein Grund dafür ist, dass PICav von vornherein alle Dienste als Kandidaten in die Verteilung einbezieht. Günstiger wäre es, die iterative Verteilungsbestimmung mit der kombinatorischen Zusammenstellung der Kandidatendienste zu verknüpfen. Somit ergibt sich die Notwendigkeit, ein verbessertes Verfahren zu konstruieren, welches jedoch im Kern auf PICav aufbauen sollte, um die Verteilung in wenigen Iterationsschritten bestimmen zu können.

Tabelle 3.7 Verteilungen zur Zusicherung einer Mindestverfügbarkeit mit der PICav-Suche

Mindestverfügbarkeit $\tilde{a}$	Tatsächliche Verfügbarkeit a	Preis	Kapazitätsoverhead
$\geq$ 0,0000	0,2794	1,54	0,0
$\geq$ 0,2795	0,9986	1,54	0,75
$\geq$ 0,9987	-	-	-

3.2.4.7 Konstruktion des Verfahrens PICav+

Gesucht wird ein Verfahren, welches ein Optimierungsproblem der Klasse MS-MD-MOO lösen kann. Ein Nutzer soll also angeben können, inwiefern eine hohe Verfügbarkeit, eine hohe Kapazität und ein niedriger Preis situationsbezogen wichtig sind,

und für diese Kombination in einer ebenfalls spezifizierten Höchstzeit eine zumindest passende, idealerweise sogar optimale Lösung erhalten. Das Verfahren sollte also iterativ arbeiten, um schnell ein lokales Optimum und in annehmbarer Zeit ein globales Optimum zu ermitteln.

In Anlehnung an die Verfahren PICav und Staffelverteilung ist das iterative Vorgehen wie folgt: Zuerst wird in jedem Iterationsschritt eine Dienstkombination gebildet. Danach werden iterativ die Staffeln bestimmt. Für jede Staffel wird in einer eingebundenen Iteration eine Gruppierung (Clustering) der Dienste gemäß ihrer Verfügbarkeiten durchgeführt. In jedem Iterationsschritt werden clusterproportional Fragmentmengen zugewiesen. Danach werden die resultierenden Zielgrößen Verfügbarkeit, Kapazität und Preis bestimmt. Ist die Frist zur Ausführung des Algorithmus überschritten oder die maximale Iterationszahl in der letzten Staffel erreicht, wird der Iterationsschritt abgebrochen und liefert das bis dahin passendste Ergebnis zurück.

Quelltext 3.2 stellt den kombinatorischen Teilalgorithmus in Pseudocode dar. Die Verteilungsbestimmung erfolgt vereinfachend anhand der primären Zielgröße apc. Ergänzend sei erwähnt, dass eine als erforderlich betrachtete Mindestdienstanzahl $\tilde{n}$ und weitere Parameter nicht als Anforderungen im Verfahren enthalten sind, diese aber analog zur Überprüfung des Preises in jeder Iteration als zusätzliche Ausschlusskriterien für gefundene Kombinationen ergänzt werden können.

Listing 3.2 Kombinatorischer Teilalgorithmus zu PICav+

```
def picavplus(C, apc, ã = 0.0, c̃ = 0, p̃ = ∞, t̃ = ∞):
  t0 = time()
  𝔓 = 𝔓(C) # Potenzmenge über die Dienstmenge
  V = ∅ # alle gültigen Verteilungen (valid)
  F = ∅ # Verteilungen mit Optima für a/p/c (final)
  ∀C ∈ 𝔓: # alle Teilmengen der Potenzmenge
    if |C| > 0:
      V = V ∪ picavplusstaggered(C, apc, ã, c̃, p̃, t̃)
      if t̃ ≠ ∞:
        if time() − t0 > t̃:
          break
  if |V| = 1:
    F_default = V0
  else:
    ∀v ∈ V:
      ∀prop ∈ {a, p, c}:
        if ¬F_prop ∨ v.prop > F_prop.prop:
          F_prop = v
  return F
```

Die Quelltexte 3.3 und 3.4 repräsentieren die weiteren Teilalgorithmen zu PICav+.

Listing 3.3 Staffelnder Teilalgorithmus zu PICav+

```
def picavplusstaggered(C, apc, ã = 0.0, c̃ = 0, p̃ = ∞, t̃ = ∞):
  V = ∅ # alle gültigen Verteilungen
  X = ∅ # Konfigurationen für alle Staffeln
  𝔄 = ∅ # Staffelkonfiguration: Dienstmenge, k, Nettokapazität
  a = 0.0
  p = 0.0
  c = 0
  slicecaptotal = 0
  while C ≠ ∅:
    slicecap = (min s.c ∀s ∈ C) − slicecaptotal
    p_slice = |C|slicecap ∑_s^C s.p
    if p̃ ≠ ∞ ∧ p_slice > p̃:
      return V
    slice = picavplusslice(C, apc, ã, c̃, p̃)
    if slice:
      k, a_slice = slice
      c_slice = kslicecap
      if c_slice = 0:
        c_slice = ∞
      X = X ∪ (a_slice, c_slice, p_slice, C, k)
    slicecaptotal += c_slice
    C = C \ {s|s.c = slicecaptotal}
    if c_slice ≥ c̃:
      C = ∅
  if |X| > 0:
    c = ∑_x^X c_x
    a = 1/c ∑_x^X a_x c_x
    p = 1/c ∑_x^X p_x c_x
    ∀x ∈ X:
      𝔄 = 𝔄 ∪ (C_x, k_x, c)
    if a ≥ ã ∧ c ≥ c̃ ∧ (p̃ = ∞ ∨ p ≤ p̃):
      V = (𝔄, a, p, c)
  return V
```

3.2.4.8 Bewertung der Verteilungsbestimmungsverfahren

Abbildung 3.25 veranschaulicht die Hierarchie aller vorgestellten optimalen und optimierenden Verteilungsbestimmungsverfahren. Es wird deutlich, dass das Verfahren PICav+ die Merkmale mehrerer Einzelverfahren vereinigt und somit einerseits stellenweise heuristisch, andererseits auch stärker laufzeitunabhängig von der Dienstanzahl n als die gestaffelte Kombinationssuche ausgeführt werden kann.

Durch die explizite Berücksichtigung der Verfügbarkeit und des Preises als risikominimierende Zielgrößen können Anwendungen bei der Generierung einer erfolgreichen Verteilung mit PICav+ Zusicherungen leisten. Auch die Kapazität kann als risikominimierende Größe eingeschätzt werden, falls beispielsweise bekannt ist, dass eine Datenquelle eine bestimmte Ausgabemenge pro Zeiteinheit liefert, wofür

Listing 3.4 Gruppierender Teilalgorithmus zu PICav+

```
def picavplusslice(C, apc, ã = 0.0, c̃ = 0, p̃ = ∞):
  C.sort(apc) # Sortierung anhand der Präferenzen (a/p/c)
  gi = [s[0].a, s[−1].a] # Verfügbarkeitsintervall
  factor = 1 # Multiplikator für die Zahl logischer Fragmente
  ∀i ∈ {1,...,maxiterations(10)}:
    intervals = ∅
    classes = ∅
    elements = 0
    ∀j ∈ {0,...,i}:
      interval = (gi[0] + j·(gi[1]−gi[0])/i, gi[0] + (j+1)·(gi[1]−gi[0])/i)
      intervals = intervals ∪ interval
      ε = 0.0001
      class = {s ∈ C | interval[0] − ε < s.a < interval[1] + ε}
      classes = classes ∪ class
      if |class| > 0:
        elements++
        ∀s ∈ class:
          s.redundant = elements − 1
    k = |C| ∗ factor
    m = Σ_s^C s.redundant
    a = av(k)
    if a ≥ ã:
      return k, a
  return ⊥
```

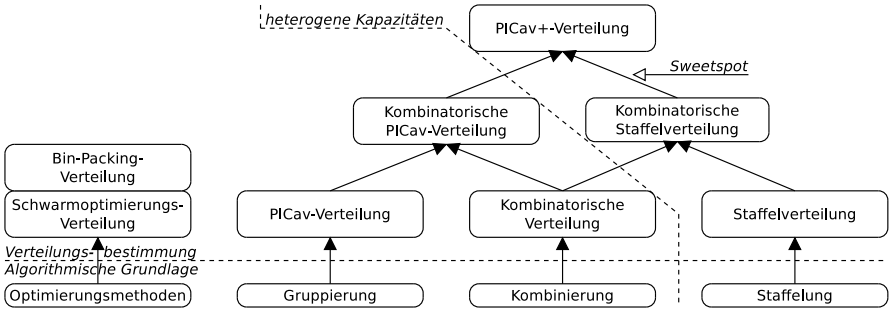

Abb. 3.25 Hierarchische Einordnung von Verteilungsbestimmungsmethoden

genügend Kapazität bereitstehen muss, da sonst das Risiko der Nichtverfügbarkeit der Speicheroperation besteht. PICav+ ist somit neben der gestaffelten Kombinationssuche das einzige Verfahren, welches zuverlässig und zusichernd eine Verteilung mit den vier Anforderungen Kapazität, Preis, Verfügbarkeit und Verfahrenslaufzeit erreicht, und ist zudem bei Abstinenz der maximalen Laufzeit für eine große Dienstanzahl im Schnitt schneller als dieses.

Tabelle 3.8 fasst die Verfahren zur Bestimmung der Fragmentverteilungen und den aufgrund der algorithmischen Komposition oder vorhandener Implementierun-

gen angenommenen Komplexitäten zusammen. Mitunter konnte die Komplexität mangels dieser Informationsquellen nur annähernd durch Annahmen über algorithmische Details abgeschätzt werden. Hierbei setzen einige Verfahren die Sortierung der Dienste nach Verfügbarkeit mit der Komplexität $\mathcal{O}(n*log(n))$ oder die Suche nach einem Maximalwert mit der Komplexität $\mathcal{O}(n)$ voraus. Die zusichernden Verfahren enthalten zudem stets die mehrfache Berechnung der Verfügbarkeit $av(2^n-1)$, welche im Fall der Abschätzung mit dem Monte-Carlo-Verfahren auf $\mathcal{O}(av(i_{max}*n*\omega))$ reduziert werden kann, so dass kein Verfahren mehr in der Klasse der exponentiellen Komplexität liegt. Weitere relevante Größen sind mit i_{max} eine jeweils verfahrensspezifische Zahl an Iterationen sowie mit s die Zahl der kapazitätsabhängigen Belegungsscheiben (*slices*).

Tabelle 3.8 Vergleich der Verfahren zur Fragmentverteilung

Verfahren	Zusicherung	Laufzeitkomplexität (Schätzung)
Gleichverteilung	keine	linear: $\mathcal{O}(n)$
Zufallsverteilung	keine	linear: $\mathcal{O}(n)$
Proportionalverteilung	keine	linear: $\mathcal{O}(n)$
Absolutverteilung	keine	konstant bis linear: $\mathcal{O}(1)\dots\mathcal{O}(n)$
Kombinatorische Suche	Verfügbarkeit, Preis	exponentiell: $\mathcal{O}(av(2^n-1)*\frac{n^2}{2})$
PICav	Verfügbarkeit bei hoher Kapazität	$\mathcal{O}(\frac{i_{max}^2}{2}*(n+av(2^n-1)))$
Schwarmoptimierung	Verfügbarkeit bei hoher Kapazität	$\mathcal{O}(i_{max}*(n+av(2^n-1)))$
Bin-Packing	Verfügbarkeit, Preis	polynomiell: $\mathcal{O}(n^h)$
Staffelverteilung	Verfügbarkeit, Kapazität, Preis	$\mathcal{O}(av(2^n-1)*s_{max}*\frac{n^2}{2})$
PICav+	Verfügbarkeit, Kapazität, Preis	$\mathcal{O}(av(2^n-1)*s_{max}*\frac{i_{max}^2*n}{2})$

Für die Validierung der Verfahren sind neben der Prüfung der Korrektheit auch zwei *sweetspots* wesentlich. Zum einen muss bestimmt werden, ab welcher Dienstzahl PICav+ schneller arbeitet als die gestaffelte Kombinationssuche, da iterative Verfahren mit Gruppierung stets einen initialen Mehraufwand beinhalten. Zum zweiten gilt dies ebenso für die präzise Berechnung und die Approximierung der Gesamtverfügbarkeit.

3.2.5 Durchführung der Verteilung

Nach der Festlegung der Verteilung erfolgt die eigentliche Verteilung, also die Übertragung an die Zieldienste. Aus diesem Grund wird in diesem Abschnitt zuerst die Regelung der Verteilung und anschließend die Datenübertragung an sich zusammen mit der Speicherung betrachtet. Der Verarbeitung ist anschließend aufgrund mehrerer Beiträge das nächste Unterkapitel gewidmet.

3.2.5.1 Regelung der Kodierung und Verteilung

Technische Eigenschaften ausgewählter Dienste für die Datenübertragung, -speicherung und -verarbeitung sollten nicht alleiniges Kriterium für die Verteilung der Daten sein. Vielmehr sollte auch die Natur der Daten, also ihr Inhalt oder Metadaten, zusammen mit weiteren Dienstnutzungskontextinformationen für die Regelung der Verteilung wie auch der vorgelagerten Kodierung berücksichtigt werden.

Die Repräsentation der Datenkodierungsketten und der Verteilung lässt sich zweckmäßig über Speicherflüsse bewerkstelligen. An derartige Speicherflüsse lassen sich Richtlinien anbringen, die zumeist in domänenspezifischen Sprachen formuliert sind. Ein Beispiel für eine solche Richtliniensprache ist die Flexible Data Distribution Policy Language (FlexDDPL) [SS12a]. Mit der Sprache lässt sich ausdrücken, wann wer welche Daten wohin und mit welcher Vorverarbeitung übertragen darf. Die Durchsetzung derartiger Richtlinien betrifft alle Vorverarbeitungsschritte, angefangen von der Kodierung bis zur Festlegung der Verteilung.

3.2.5.2 Datenübertragung und Datenspeicherung

Die Übertragung dispergierter Daten wird als *Dispersed Communication* oder alternativ *Dispersed Transmission* bezeichnet. Neben einer $1 : n$-Abbildung ist auch eine $n_1 : n_2$-Abbildung für fehlertolerante und hochverfügbare Netzwerkanwendungen sinnvoll [SS14b].

Werden Daten in Form von Fragmentströmen dienstübergreifend gespeichert, so lässt sich dieser Umstand mit dem Begriff dispergierte Speicherung oder *Dispersed Storage* ausdrücken. Hierzu existieren viele Grundlagenbetrachtungen und anwendungsbezogene Analysen [SBM+11, SMS13, SS14b]. Als Spezialfall von allgemein verteilter Datenspeicherung (*distributed storage*) treffen deren Eigenschaften auch auf die dispergierte Speicherung zu.

Die Metadaten zu gespeicherten Daten müssen ebenfalls sicher gespeichert werden. Hierfür bietet sich ein rekursives Verfahren an, welches eine linear anwachsende Menge an lokalen Metadaten auf einen konstanten Eintrag reduziert, welcher die weiteren Metadaten referenziert, womit der Datenabruf zweistufig wird [SMS13].

3.3 Semantische Verknüpfung verteilter kodierter Daten

Über Datenspeicherflussbeschreibungen (*storage flows*) lässt sich formal ausdrücken, unter welchen Bedingungen Daten auf welche Art und Weise kodiert und verteilt werden sollen, wenn das Ziel eine persistente Speicherung ist [SS13b]. Ihre Konstruktion obliegt bestimmten Richtlinien, welche die Freiheitsgrade der Gestaltung von Speicherflüssen reguliert [SS12a]. Der Austausch von verteilt gespeicherten und möglicherweise stealth-kodierten Daten erfordert die Reversinterpretation der Speicherflüsse und insbesondere der durch sie entstehenden finalen Datenrelationen. Im

Folgenden werden dafür Formalismen zur Repräsentation von Relationsschemata sowie Algorithmen zu deren Verarbeitung eingeführt.

3.3.1 Datenrelationsschemata

Aufgrund der vielen Konfigurationsmöglichkeiten zur Aufteilung und verteilten Speicherung und Verarbeitung von Daten wird für anwendungsübergreifende Zugriffe ein Konstrukt benötigt, um logisch verknüpfte Datenorte mit eindeutiger Semantik für den Anwender nachvollziehbar und kontrollierbar auszudrücken. Auf der Modellebene wird hierzu das Relationenschema *Data Relation Scheme* (DRS) eingeführt. Die Verarbeitung von Instanzen des Modells ermöglichen dabei einen sicheren Datenzugriff in verteilten Umgebungen bis hin zur Migration in neue Umgebungen. Die Daten können in der Form von Dateien, Datenströmen oder Tupeln vorkommen, wobei die weiteren Betrachtungen auf die Dateiform zugeschnitten sind. Vom DRS abgeleitet wird dazu das *File Relation Scheme* (FRS) eingeführt. Ein solches Schema zur Formalisierung der Beziehungen zwischen Daten oder Dateien ist bisher aus der Literatur nicht bekannt. Allenfalls in Peer-to-Peer-Netzen sowie im Webumfeld gibt es ähnliche Ansätze, die jedoch eher inhaltsbezogen als relationenbezogen und eher netzzentrisch als nutzerzentrisch angelegt sind. So ist es in neueren P2P-Systemen möglich, virtuelle stabile Knoten zu etablieren, welche ihrerseits die abgefragten Daten unter Ausnutzung von Fehlertoleranzmechanismen von mehreren weiteren Knoten beziehen können, und anschließend stabile und hochverfügbare Verweise auf die stabilen Knoten zu setzen. Dabei entscheidet jedoch letztlich der stabile Knoten und nicht der Benutzer über die Wahl der genutzten Knoten [AUS+09]. Im Web-Umfeld und in Dokumentenverwaltungen werden Dateien und Dokumente auf Ähnlichkeit geprüft, wobei die genutzten Metriken rein quantitative und inhaltsabhängige Ergebnisse liefern und beispielsweise Dispersionsschritte nicht explizit als Ähnlichkeitsmaß abbilden können [WWG14, Bro97]. Ähnliches gilt für bisherige Ansätze aus dem Linked-Data-Bereich, die auf einer inhaltlichen Ebene Verknüpfungen erfassen und weniger eine reine Datenzusammengehörigkeitssemantik vermitteln, auch wenn diese mit den eingesetzten Sprachen wie RDF erfassbar wären. Forschungsarbeiten und praktische Anwendungen im Netzwerkbereich kennen aufgeteilte oder replizierte Datenströme unter den Bezeichnungen Multihoming, IP Flow Mobility und Multipath TCP und zugehörige Priorisierung und Scheduling-Optimierungen [SMK15]. Für die Beschreibung der Zusammenhänge der Datenströme existiert aber ebenfalls bisher kein Schema. Somit stellen DRS und FRS das erste generische Modell zur qualitativen Angabe von Relationen zwischen Originaldaten und abgeleiteten Daten dar. Eine menschen- und maschinenlesbare Darstellung von DRS- und FRS-Instanzen wird benötigt, um die Relationen interoperabel auswerten zu können.

Das FRS sieht vier grundlegende Relationen vor: Äquivalenz ($\doteq$) für die vollständige Übereinstimmung zwischen zwei Dateien, interpretierbar auch als Originaldatei mit Replikat oder als Fragment mit Replikat, Teilmenge ($\subset$) für die teil-

weise Übereinstimmung im Sinne einer Teilmengenrelation zwischen Fragment und Ausgangsdatei, Komplement (∞) für komplementäre Fragmente, welche zusammen eine vollständige Datei ergeben, sowie Redundanz ($\propto$) für redundante Fragmente, die im Fall des vollständigen Vorliegens aller komplementären Fragmente ignoriert werden können und andernfalls für die Wiederherstellung der nicht vorliegenden Fragmente genutzt werden.

Daraus ergibt sich, dass $\subset$ und $\propto$ asymmetrische bzw. gerichtete Relationen sind, während ∞ und $\doteq$ symmetrisch bzw. ungerichtet und damit mathematisch kommutativ sind. In Tabelle 3.9 werden die Relationen und ihre Charakteristiken zusammengefasst. Weitere Relationen, die sich aus Kodierungen und Transformationen neben der Aufteilung noch ergeben, einschließlich Verschlüsselung, Komprimierung und inhaltsspezifischer Transformationen wie Textkodierung oder Bildqualitätsreduktion, werden im FRS nicht berücksichtigt.

Tabelle 3.9 Relationen im FRS

Relation	Anwendung	Kommutativität
$\doteq$	Datei $\doteq$ Datei, Fragment $\doteq$ Fragment	ja
$\subset$	Fragment $\subset$ Datei	nein
∞	Fragment ∞ Fragment	ja
$\propto$	redundantes Fragment $\propto$ signifikantes Fragment	nein

Ein Merkmal von FRS ist, dass die Art der Datei nicht explizit angegeben muss, sondern aus den Relationen inferiert werden kann. Die zusammenhängenden Inferenzregeln 3.9 und 3.10 sowie 3.11 und 3.12 machen dies deutlich. Sie bestimmen jede Datei als vollständige Datei, signifikantes Fragment oder redundantes Fragment. Da die vier Inferenzregeln vollständig sind, ist die explizite Angabe der Art der Datei als Annotation auf dem Graph und somit die Nichtanwendung der Regeln nur für die Situationen empfehlenswert, in denen die Ressourcenoptimierung die Verarbeitung gegenüber der Speicherung und Übertragung von Daten bevorzugt.

$$\frac{A \doteq B \wedge A \subset C}{A,B \in fragments, C \in files} \tag{3.9}$$

$$\frac{A \doteq B \wedge \neg(A \subset C)}{A,B \in files} \tag{3.10}$$

$$\frac{A \propto B}{A \in redundant fragments, B \in fragments} \tag{3.11}$$

$$\frac{A \infty B}{A,B \in fragments} \tag{3.12}$$

Die konkrete Ausprägung von Relationen gemäß FRS lässt sich stets als kantengefärbter Graph darstellen, womit dieser auch als Ontologie aufgefasst werden kann. Abbildung 3.26 zeigt beispielhaft einen FRS-Graphen. Aus der Darstellung geht hervor, dass einige Kanten explizit gesetzt sind (durchgehende Linie), während

andere implizit abgeleitet werden können (Punktlinie). Allgemeiner betrachtet besitzen FRS-Graphen einen Grad an Vollständigkeit zwischen einem minimalen und einem vollständigen Grad, wobei die beiden Extrema stets aus jedem Grad durch Anwendung weniger Regeln erreicht werden können.

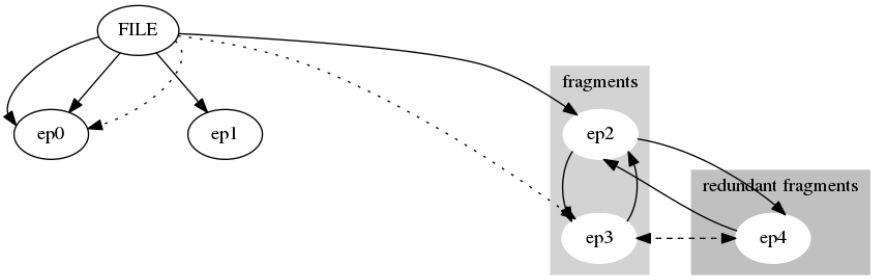

Abb. 3.26 Beispiel für einen FRS-Graphen, dessen Kanten aus technischen Gründen gerichtet dargestellt werden

Die Minimierung und Vervollständigung eines beliebigen FRS-Graphen erfolgt durch das iterative Applizieren von Transformationsregeln bis zum Erreichen des jeweiligen Extrems. Der Graph gilt als vollständig, wenn zwischen zwei beliebigen Knoten genau jeweils eine Kante existiert und somit eine weitere Iteration keine Änderung hervorruft, sowie als minimal, wenn ebenfalls keine Änderung mehr erzielt werden kann. Die Vervollständigungsregeln lauten im Einzelnen o.A.d.V.:

1. Transitive Redundanz. Sind die Fragmente A und B komplementär, sowie C zu A redundant, dann ist auch C zu B redundant.
2. Transitive partielle Replikate replizierter Dateien. Sind die Dateien A und B Replikate, sowie C zu A ein partielles Replikat, dann ist auch C zu B ein partielles Replikat.
3. Transitive partielle Redundanz einer Datei. Sind die Dateien A und B komplementär, sowie A zu C ein partielles Replikat, so ist auch B zu C ein partielles Replikat.

Die Gleichungen 3.13, 3.14 und 3.15 formalisieren diese Regeln. Für die Minimierung werden die Regeln invers angewendet. Trifft die Konklusion zu, dann wird sie unter Beibehaltung der Prämissen aufgelöst.

$$\frac{A \infty B \wedge A \propto C}{B \propto C} \tag{3.13}$$

$$\frac{A \doteq B \wedge C \subset A}{C \subset B} \tag{3.14}$$

$$\frac{A \infty B \wedge A \subset C}{B \subset C} \tag{3.15}$$

Jeder Graph kann hinsichtlich Vollständigkeit und Validität überprüft werden. Die Vollständigkeitsprüfung erfolgt anhand der Formel $A * B = A\infty | \propto | \subset | \doteq B \forall A, B \in locations$. Die Validierung erfolgt mit Auschlussregeln, die besagen dass bestimmte Relationen unzulässig sind. Dies betrifft vollständige Dateien, die als Replikat eines Fragments angegeben sind (Gleichung 3.16), sowie weitere undefinierte Fälle (Gleichungen 3.17, 3.18, 3.19 und 3.20) o.A.d.V. Hierbei steht das Relationssymbol $*$ für eine beliebige Relation. Die rekursive Aufteilung von Fragmenten in Teilfragmente wird durch FRS nur insofern berücksichtigt, als dass das Fragment wiederum als Ausgangsdatei betrachtet werden muss.

$$\frac{A \doteq B \wedge A \subset C \wedge B \doteq C}{\neg} \tag{3.16}$$

$$\frac{A \propto B \wedge B \subset C \wedge A * C}{\neg} \tag{3.17}$$

$$\frac{A \propto B \wedge (C \subset A \vee C \subset B)}{\neg} \tag{3.18}$$

$$\frac{A \infty B \wedge (C \subset A \vee C \subset B)}{\neg} \tag{3.19}$$

$$\frac{A \propto B \wedge A * C}{\neg} \tag{3.20}$$

FRS weist somit vier Relationen, vier Inferenzregeln, drei Vervollständigungsregeln und fünf Ausschlussregeln auf. Auf dieser Grundlage lassen sich komplexe verteilte Speicherzielkonfigurationen mit je einem FRS-Graphen pro Datei formal ausdrücken. Eine vorteilhafte Eigenschaft von FRS besteht darin, dass Knoten die Orte, also die Verteilung, der Daten repräsentieren, während die Kanten die Kodierung ausdrücken, und beides unabhängig voneinander durch Substitutionen die langfristige Evolution von Daten unterstützt, dadurch aber auch neue Herausforderungen hinsichtlich der Konsistenz mit sich bringt. Somit sind diese Graphen einerseits dateiabhängig, andererseits auch zeitabhängig, was für die Nutzung beachtet werden muss.

Die Nutzung von FRS-Instanzen zur Laufzeit ist für drei Vorgänge von Belang: der sichere Zugriff auf Dateien per Download, die Freigabe von verteilt gespeicherten Dateien für andere Benutzer, sowie die a-priori prüfbare Migration komplexer Datenbestände per Up- und Download mit optionalem Crossload. Die dafür notwendigen Algorithmen werden in den nachfolgenden Abschnitten vorgestellt.

3.3.2 Algorithmen auf Basis von Datenrelationsinstanzen

3.3.2.1 Dateizugriff

Der lesende Zugriff auf eine Datei mit höchstmöglicher Verfügbarkeit erfolgt durch Iteration über den FRS-Graphen mit seriellen Zugriffsversuchen auf alle notwendigen vollständigen Dateien, signifikanten Fragmente und redundante Fragmente bis zur erfolgreichen Erzielung eines Dateiabbilds. Sind die Netzwerkressourcen vorhanden und die für die Verwendung der Ressourcen anfallenden Kosten nicht relevant, können die Zugriffsversuche auch parallel durchgeführt werden, anderweitig muss eine Reihenfolge festgelegt werden. Sind die zu erwartenden Einzelverfügbarkeiten der Speicherorte bekannt, können diese zur Priorisierung herangezogen werden, andernfalls gilt im Grunde die Vorrangregel der vollständigen Dateien vor den signifikanten Fragmenten und nachrangig die redundanten Fragmente. Dieser Vorrang lässt sich durch die Vermeidung zusätzlicher Netzwerkanfragen sowie der inversen Kodierung fehlender Fragmente begründen.

Im Quelltext 3.5 wird der Algorithmus für den Dateizugriff in Pseudocode in Anlehnung an eine Python-Syntax beschrieben. Der Graph G wird hierbei als Klasse mit den Attributen $files$, $fragments$, $redundantfragments$ und $relations$ repräsentiert, wobei die ersten drei Attribute inferiert werden können und somit nur aus Gründen der Darstellung angegeben werden.

Listing 3.5 Zugriff auf eine verteilt gespeicherte Datei

```
def download(g, dir):
  for f in g.files:
    if download(f, dir):
      return dir/f
  nfragments = 0
  for f in g.fragments:
    if download(f, dir):
      nfragments += 1
  if nfragments < g.k:
    for f in g.redundantfragments:
      if download(f, dir):
        nfragments += 1
        if nfragments == g.k:
          break
  if nfragments == g.k:
    lf = decodefragments(dir)
    return lf
  return None
```

3.3.2.2 Dateifreigabe

Durch Weitergabe der FRS-Instanz kann der lesende Zugriff auf eine verteilt gespeicherte Datei durch beliebige Anwendungen und Geräte erfolgen. Dazu ist es vorteilhaft, den Graphen zu minimieren und in eine maschinell auswertbare Syntax zu transformieren. Die Überführung des abstrakten Graphen in konkrete JSON-, XML- und URL-Repräsentationen wird im Abschnitt 4.7 behandelt.

Zur Gewährleistung der Interpretierbarkeit muss die FRS-Instanz um zwei weitere Informationen angereichert werden. Erstens muss im Fall der Bereitstellung von Fragmenten das Kodierungsverfahren mitsamt dessen Parametern bekannt sein, um die Dekodierung ohne Heuristiken durchzuführen. Zweitens sollte ein Hinweis auf den resultierenden Dateinamen gegeben werden, welcher aus den Replikaten oder Fragmenten nicht ohne weiteres Hervorgeht.

Ein weitergegebener FRS-Graph hat demnach die Form $< G, codec, filename >$.

3.3.2.3 Dateimigration

Die Migration von Daten zwischen zwei Speicherorten lässt sich unterteilen in eine dateibezogene Migration, bei der ein oder mehrere Speicherorte einer FRS-Instanz substituiert werden, sowie eine speicherortsbezogene Migration, in der der betreffende Ort auf sämtlichen betroffenen FRS-Instanzen substituiert werden muss. Eine Migration resultiert in einer Verteilung mit geänderten Eigenschaften. Dies betrifft insbesondere die Verfügbarkeit und die entstehenden Speicherkosten. Zudem können für die Migration selbst Kosten anfallen. Für den kontrollierbaren Ablauf empfiehlt sich ein Vergleich der beiden FRS-Graphen, welche den Ist-Zustand und den Soll-Zustand der Datenverteilung repräsentieren. Dazu ist es notwendig, die Graphen hinsichtlich ihrer Struktur zu vergleichen, wofür diverse generische Algorithmen existieren [FV01]. Es ist aber noch kein Algorithmus bekannt, der die Migrationsentscheidung mit Einschränkungen seitens des Benutzers auf Zulässigkeit prüft und im Ergebnis eine vergleichbare, bessere oder schlechtere Verteilung beurteilen kann. Desweiteren ist noch kein Algorithmus zur tatsächlichen Durchführung der Migration bekannt. Hierfür wird in abgewandelter Form der Zugriffsalgorithmus genutzt, wobei nicht vorhandene Replikate, signifikante oder redundante Fragmente unter Ausnutzung der Redundanz durch Neuberechnung ersetzt werden können. Somit lässt sich auch hier eine Abwägung auf Ressourcenebene zwischen Datenübertragung und Datenverarbeitung einbringen.

Bisherige Verfahren der Datenmigration über mehrere Speicherdienste berücksichtigen das Platzierungsproblem, gehen aber von gleich großen und stets signifikanten Datenblöcken aus [PT09] und berücksichtigen keine weiteren Optimierungen etwa für dienstinterne Migrationen [HCC11].

Die strukturvergleichende Überprüfung zweier Graphen ist im Quelltext 3.6 dargestellt. Hierbei gilt der strikte Modus der strukturellen Übereinstimmung, so dass die Daten ohne Umkodierung migriert werden können.

Listing 3.6 Migrationsprüfung

```
def compatible(gsrc, gdst):
  if len(gsrc.files) != len(gdst.files) or len(gsrc.fragments) !=
       len(gdst.fragments) or len(gsrc.redundantfragments) != len
      (gdst.redundantfragments):
    return False
  if len(gsrc.relations) != len(gdst.relations):
    return False
  for srcrelation, dstrelation in zip(sort(gsrc.relations), sort(
      gdst.relations)):
    if srcrelation != dstrelation:
      return False
  return True
```

Die eigentliche Migration ist algorithmisch im Quelltext 3.7 dargestellt. Sie ruft aus Gründen der Robustheit zuerst intern die Migrationsprüfungsfunktion auf, lädt danach alle Dateien, welche nicht per Crossload innerhalb eines Dienstes migriert werden können, auf einen Zwischenspeicher herunter, und lädt sie anschließend wieder gemäß der neuen Speicherzielkonfiguration wieder hoch. Der Algorithmus sieht keinen Transaktionskontext vor, so dass es Aufgabe der Anwendung ist, nach Fehlerfällen die Konsistenz der Daten zu sichern. Desweiteren sieht der Algorithmus keine Löschung der Daten an den nunmehr ungenutzten Speicherorten vor, da diese mitunter ohnehin vollständig deaktiviert werden.

3.4 Verarbeitung verteilter kodierter Daten

Nach der Kodierung, Ordnung und Festlegung der Verteilung sowie Übertragung oder Speicherung der Daten soll abschließend deren verteilte Verarbeitung betrachtet werden. Hierbei ist von Interesse, mit welchen Techniken dies erfolgt, so dass sie nicht nur während der Übertragung oder Speicherung, sondern auch in weiteren Verarbeitungsschritten langfristig geschützt sind.

In diesem Abschnitt werden mit der erweiterten Betrachtung der Datenverarbeitung die Konzepte *Dispersed Computing*, *Encrypted Computing* und *Stealth Computing* eingeführt. Abschließend werden vertiefend algorithmische Aspekte aus dem Stealth Computing diskutiert.

3.4.1 Verarbeitung dispergierter Daten

Der Begriff *Dispersed Processing* oder alternativ *Processable Dispersion* bezeichnet im weiteren Sinne die Verarbeitung beliebig dispergierter Daten [SS14b]. Hier-

Listing 3.7 Migration einer verteilt gespeicherten Datei

```
def migrate(gsrc, gdst, dir, mindownload):
  if !compatible(gsrc, gdst):
    return False
  nfiles = 0
  for f in gsrc.files:
    # Markierung Crossload
    if f in gdst.files:
      continue
    if download(f, dir):
      nfiles += 1
      if mindownload:
        break
  replicatefiles(gsrc.files, dir)
  nfragments = 0
  for f in gsrc.fragments:
    if download(f, dir):
      nfragments += 1
  if nfragments < gsrc.k + gsrc.m:
    for f in gsrc.redundantfragments:
      if download(f, dir):
        nfragments += 1
      if mindownload and nfragments == gsrc.k:
        break
  if nfiles == 0 or nfragments < gsrc.k:
    return False
  decodefragments(dir)
  for f in gdst.files:
    if f in gsrc.files:
      # Durchführung Crossload
      if not crossload(f):
        return False
    elif not upload(dir/f):
      return False
  for f in gdst.fragments:
    if not upload(dir/f):
      return False
  for f in gdst.redundantfragments:
    if not upload(dir/f):
      return False
  return True
```

unter fallen per definitionem auch unkodierte sowie replizierte Daten, für welche, von Fehlertoleranzmechanismen abgesehen, keine gesonderten Verarbeitungsuntersuchungen zu führen sind. Einschränkend wird deshalb Dispersed Processing im engeren Sinne als Verarbeitung fragmentierter Daten resultierend aus Löschkodierung, Bitsplitting oder interpolierter Aufteilung verstanden. Die Herausforderung und gleichzeitig der Vorteil ist hierbei, dass lokale Algorithmen nicht stets auf die vollständigen Daten zugreifen können oder müssen, sondern das globale Verarbei-

tungsresultat nur durch eine geeignete Koordinierung der lokalen Verarbeitungsschritte zustandekommt. Der weiter gefasste Begriff *Dispersed Computing* bezeichnet somit ein Paradigma der Datenverarbeitung, welches die dispergierte Übertragung und Speicherung von Daten mit der passenden Verarbeitung kombiniert und somit erweiterte Sicherheitszusagen für alle an der Datenverarbeitung beteiligten Systeme zulässt.

Die Übermittlung dispergierter Daten über n unterschiedliche Kanäle schützt vor den Risiken, die den Kanal betreffen. Gleichermaßen werden die Daten bei der Speicherung auf n unterschiedlichen Speicherzielen oder der Verarbeitung auf n unterschiedlichen Rechnern geschützt. Zwischen den drei Infrastruktursegmenten existieren mehrere Abhängigkeiten. Zum einen wird die Netzwerkkommunikation benötigt, um die Daten zum Speicher- oder zum Verarbeitungsort zu transportieren. Zum anderen kann die verteilte datenpositionsaffine Verarbeitung sowohl in-situ, also auf gespeicherten Daten, als auch in-transit, also in Echtzeit auf übertragenen Daten, ausgeführt werden.

3.4.2 Verarbeitung verschlüsselter Daten

Der Begriff *Encrypted Processing* oder alternativ *Processable Encryption* bezieht sich auf die Verarbeitung von verschlüsselten Daten ohne Entschlüsselung direkt im Verschlüsselungsraum. Lediglich das Ergebnis wird für eine sinnvolle Weiterverarbeitung letztlich wieder entschlüsselt. Das Ziel der Verschlüsselung ist es, dass zu jedem beliebigen Zeitpunkt nur der legitimierten Aufrufer einer Operation die ursprünglichen Daten oder davon abgeleitete Daten einsehen kann, wobei dieses Ziel insofern eingeschränkt wird, als dass oft durch Verknüpfung mehrerer Aufrufe ein Rückschluss auf den Dateninhalt ermöglicht wird, sofern nicht bestimmte datenschützende Vorkehrungen getroffen werden [JKF12].

Mehrere Algorithmen leiten sich direkt aus den Kodierungsformen ab. So sind Rechen- und Sortieralgorithmen auf homomorph und ordnungserhaltend verschlüsselten Daten definiert, allerdings mit Einschränkungen hinsichtlich der Datentypen und der Datentypgrößen. Im Paillier-Kryptosystem, einer Ausprägung der *computable encryption*, ist eine Addition zweier Zahlen im homomorphen Raum beispielsweise folgendermaßen definiert: $add(s1,s2) = H^{-1}(H(s1)h(s2)\%(pq)^2)$ [Pai99]. Dabei sind H und H^{-1} die Ver- und Entschlüsselungsoperationen, die einen Wert vom ganzzahligen in den homomorphen Raum und wieder zurück transformieren. Wird hingegen eine Konstante aufaddiert, so lautet die Definition: $add(s1,K) = H^{-1}(H(s1)((pq+1)^K\%(pq)^2)\%(pq)^2)$. Schließlich ist noch die Multiplikation mit einer Konstanten definiert: $mul(s1,K) = H^{-1}(s1^K\%(pq)^2)$.

Weitere Algorithmen umfassen die verschlüsselte Suche nach Daten unter dem Begriff *searchable encryption* [SWP00, BHJP15]. Neben der Suche nach Schlüsselwörtern (*public-key encryption with keyword search*, PEKS) können auch reguläre Ausdrücke mit einer geringen Laufzeitverzögerung von ca. 6% und keinen Einschränkungen hinsichtlich der Genauigkeit gesucht werden [BCOP04, SCF+14].

Sind hingegen Genauigkeitseinschränkungen zulässig, so kann eine ungenaue Suche nach Schlüsselwörtern (*fuzzy search*) stattfinden. Hierbei wird die Suchkomplexität von linear auf logarithmisch oder gar, bei gleichzeitiger Reduktion der Vertraulichkeit, auf konstant reduziert [Lu12, CGKO06].

3.4.3 Verarbeitung dispergierter und verschlüsselter Daten

Während die Dispersion der Daten erhebliche Vorteile für die Verfügbarkeit bei geringem zusätzlichen Platzbedarf, jedoch nur bestimmte Verbesserungen für das Laufzeitverhalten oder die Vertraulichkeit von Daten erzielt, lässt sich die kombinierte *Stealth-Kodierung* zusammen mit weiteren Stealth-, Sicherheits- und Fehlertoleranzverfahren zur gesamtheitlichen Absicherung von Daten und Anwendungen nutzen. Daraus entsteht das Paradigma des *Stealth Computing*, welches infrastrukturabhängig einen weit gefassten Schutz vor datenbezogenen Risiken bietet. In diesem Abschnitt werden zunächst ähnliche kombinierte Verfahren verglichen und anschließend die Stealth-Kodierung und darauf anwendbare Algorithmen vorgestellt.

3.4.3.1 Begriffsdefinition Stealth Computing

Stealth Computing bezeichnet im weiteren Sinne ein mehrdimensionales Spektrum zur sicheren Nutzung von Speicher- und Rechenressourcen in verteilten Systemen. Im engeren Sinne bezieht es sich auf das sichere Ende des Spektrums, welches jedoch hinsichtlich einer unbeschränkten Datenspeicherung und Programmausführung nur als theoretischer Punkt existiert und von Systemumsetzungen nicht erreicht wird. Stattdessen werden abhängig von den erforderten Funktionen und von der Datenkodierung Zwischenwerte auf den einzelnen Dimensionen erreicht, die zudem weitere Einschränkungen etwa der Ausführungszeit oder der benötigten Speicherkapazität zur Erzielung der Sicherheit implizieren.

Zur Beschreibung und Einordnung von Systemen bietet sich eine textuelle oder grafische Notation an. Beide Notationen werden an dieser Stelle eingeführt und an entsprechenden weiteren Stellen wieder aufgegriffen.

Die Notation in Textform repräsentiert die vereinfachte dreidimensionale Kombination aus Verarbeitungsfunktion, Verteilung und Verschlüsselung. Diese sei als *Stealth-3V*-Notation bezeichnet. Die Verarbeitungsfunktionen gliedern sich in solche zur reinen Datenspeicherung unter Einbeziehung CRUD-Operationen (*create, read, update, delete*) sowie möglicher weiterer generischer, nicht dateninhaltsbezogener, Datenverwaltungsfunktionen, und in solche zur beliebig komplexen Verarbeitung der Daten. Die Funktionen werden demnach gebündelt als *S* für *storage* und *C* für *computation* bezeichnet. Die Verteilungen gliedern sich in *L* für *local*, also unverteilt, und *D* für *distributed*. Schließlich gliedern sich die Verschlüsselungen in *P* für *plain*, also unverschlüsselt, und *E* für *encrypted*.

Somit ist die Textnotation ein Tripel zusammengefasst durch Formel 3.21. Die konkret genutzten Algorithmen werden durch die Notation nicht ausgedrückt. Sie eignet sich vielmehr für die ungefähre Einordnung eines Systems mit teilweisen oder vollständigen Stealth-Eigenschaften. Auch die Berechnung über redundante Daten ist demnach nicht Gegenstand der Notation. Als Kurzbezeichnung für *DEC*, welches den vollen Umfang repräsentiert, wird $\mathscr{S}$ (kalligrafisch-S) vorgeschlagen.

$$stealthsystem := \{L,D\}x\{P,E\}x\{S,C\} \tag{3.21}$$

Die acht möglichen Textnotationen sind in der Tabelle 3.10 gegenübergestellt und jeweils mit einer beispielhaften Anwendung erläutert. Daraus geht hervor, dass sich viele existierende Systeme in die $**S$-Klassen einordnen lassen, jedoch nur wenige in die $**C$-Klassen jenseits von *LPC*. Deren Ergründung im Rahmen dieser Arbeit ermöglicht die Komplettierung des Spektrums und eröffnet damit flexiblere Architekturoptionen für die Konstruktion sicherer verteilter Anwendungen.

Tabelle 3.10

3V-Notation	Beispielanwendung
LPS	Speicherung auf unverschlüsselten Datenträger
LES	Speicherung auf verschlüsselten Datenträger
DPS	Speicherung per RAID/RAIC
DES	verschlüsselte Speicherung per RAID/RAIC
LPC	konventionelle Programmausführung
LEC	Berechnungen im homomorphen Raum; *processable encryption*
DPC	Berechnungen im dispergierten Raum; *processable dispersion*
DEC	Berechnungen im Stealth-Raum ($\mathscr{S}$)

Die grafische Notation für Systeme aus der Perspektive des Stealth Computing wird mit Abbildung 3.27 eingeführt. In dieser sind die Dienst- oder Ressourcenklassen sowie die Kodierungs- und Verteilungsaspekte jeweils als Dimension berücksichtigt. Sie sei als *Stealth-Cube*-Notation bezeichnet. Die Extremkoordinaten des Kubus sind $\mathscr{V}$ für *vulnerable* als Bezeichnung eines Systems ohne Schutzeigenschaften und $\mathscr{S}$ für *stealth* als Bezeichung eines umfänglich geschützten Systems. Um die Hypothese auszudrücken, dass der vollständige Schutz nicht erreichbar ist und nur ein Konvergenzziel für die Systemkonstruktion darstellt, wird zudem eine $\mathscr{S}-\varepsilon$-Umgebung eingeführt, deren Umfang zu untersuchen ist.

Beispielhaft wird die grafische Notation zur Einordnung von drei Systemklassen in den Abbildungen 3.28 (a-c) gebraucht. In der Textnotation werden Systeme nach (a) als DPS, nach (b) als LEC und nach (c) als DEC bzw. $\mathscr{S}$ klassifiziert.

Die Zulässigkeit der Ausführung von Funktionen respektive Operationen zur Laufzeit ist direkt von der gewählten Datenkodierung und der Datentypisierung abhängig. So lässt sich beispielsweise die Addition über zwei beliebige Ganzzahlen definieren; werden diese Summanden jedoch mittels eines 32-Bit-Schlüssels in den homomorphen Raum transformiert, limitiert diese Kodierung auch die zulässige Größe der Summanden und damit gleichzeitig die Zulässigkeit der Addition

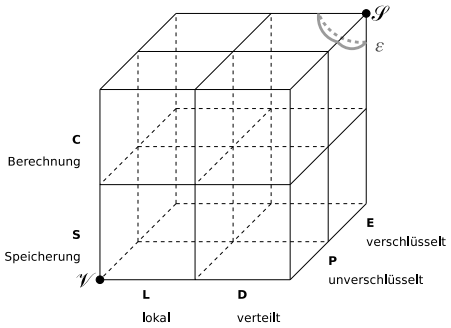

Abb. 3.27 Schema zur grafischen Notation für Stealth-Systeme

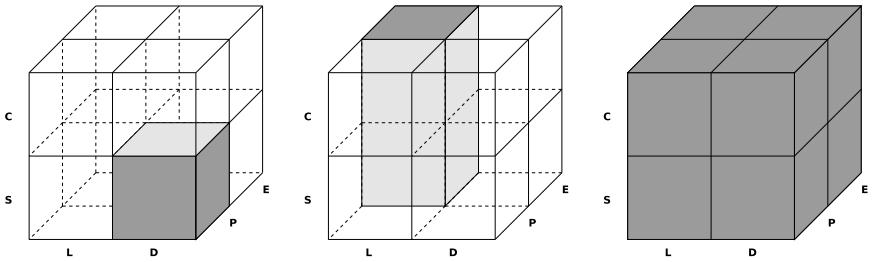

Abb. 3.28 Ausgewählte grafische Darstellungen für (a) redundante Datenspeicherung in RAID- und RAIC-Systemen, (b) sichere Datenverarbeitung in verschlüsselten Datenbanken, (c) vollumfängliche sichere und verteilte Datenspeicherung und -verarbeitung

größerer Summanden. Hierfür wird der Begriff des *Stealth-Filters* eingeführt. Mehrere Filter lassen sich in einer geometrischen Repräsentation ähnlich einer Schablone übereinanderlegen und limitieren somit zunehmend die zulässigen Datentypen und Operationen. Stealth-Systeme, welche sich frei konfigurieren lassen, benötigen demnach die Filter zur Entscheidung über die Zulässigkeit der Gesamtkonfiguration und zur Prüfung auf Laufzeitfehler. Damit wird deutlich, dass die Definition der Filter eine Herausforderung für die Konstruktion von Stealth-Systemen darstellt.

Abbildung 3.29 stellt die überlagerte Anwendung von Filtern abstrakt-schematisch dar. Konkrete Ausprägungen folgen im Anschluss an die Konstruktion und Analyse von Algorithmen zur Verarbeitung von Stealth-Daten, womit die Grundlage für die Untersuchung von ε gelegt wird.

3.4.3.2 Verwandte Ansätze zur sicheren verteilten Datenverarbeitung

Zur sicheren Datenverarbeitung in unsicheren Systemen existieren mehrere Verfahren. Diese nehmen jedoch in der Mehrzahl ein einziges Verarbeitungssystem, also beispielsweise einen Dienstanbieter, oder eine einzelne Kodierung an und fallen damit nicht in die Klasse der $\mathscr{S}$-Systeme nach strikter Definition oder zumindest in die

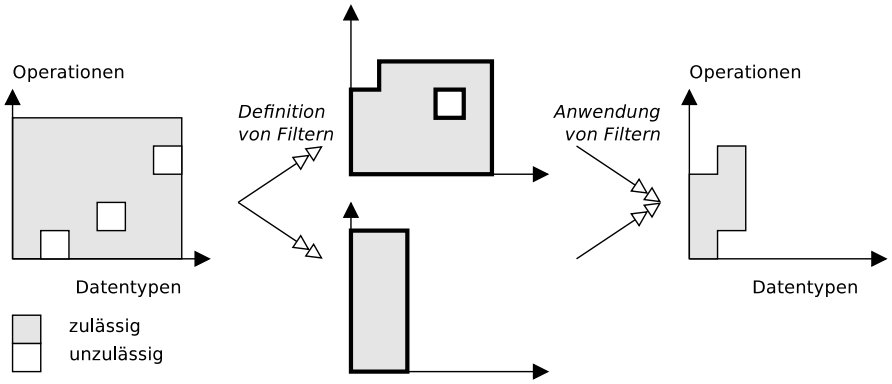

Abb. 3.29 Schema zur Filteranwendung über zulässige Operationen in Stealth-Systemen

Klasse der $\mathscr{S} - \varepsilon$-Systeme. In diesem Kapitel werden hingegen primär kodierungs- und funktionsübergreifende Berechnungen betrachtet. Weiterhin sind die existierenden Verfahren oft auf spezielle Algorithmen zugeschnitten, so beispielsweise auf Datenbankoperationen. Diese Einschränkung wird für die nachfolgende Konstruktion und Analyse von Stealth-Algorithmen beibehalten.

Hacıgümüş et al. nutzen einen Privacy-Homomorphismus zur sicheren Aggregation von Daten [HIM14]. Die Daten liegen dabei aber nicht verteilt vor. Halang et al. beschäftigen sich mit sicherer Auslagerung von Anwendungen auf Dienstarchitekturen allgemein [HKS14], nehmen aber ebenfalls einen einzelnen Anbieter an. Kerschbaum überprüft Entwurfskriterien für vertrauliche Informationssysteme in nicht vertrauenswürdigen Umgebungen [Ker11]. Eine Ausführung nach erweiterten Sicherheitskriterien einschließlich der Verfügbarkeit wird nicht untersucht.

Agrawal und Emekci schlagen Algorithmen für die verteilte Verarbeitung von Secret-Sharing-Fragmenten unter Ausschluss schwergewichtiger Rechenoperationen auf verschlüsselten Daten vor [AAEM09, EMAA14]. Die Fragmente werden dabei als Polynome vom Grad $k-1$ zusammen mit einem Schlüssel über Ganzzahlen gebildet. Das Verfahren ist damit informationstheoretisch sicher. Es ist zudem ordnungserhaltend, wenn die Polynome sorgfältig konstruiert werden [EMAA14]. Auf die verteilten Fragmenten können verschiedene Anfragen (*exact match, range, aggregation queries*) gestellt werden. Sowohl Ausfälle als auch byzantinische Fehler werden bis zu einer bestimmten Grenze (m von n) toleriert. Allerdings wurde das Verfahren bisher nur theoretisch betrachtet und nur durch eine Simulation validiert. In dieser Arbeit soll hingegen ein Laufzeitsystem dieser Art konstruiert werden. Zudem soll in Hinblick auf die Konstruktion von sicheren verteilten Anwendungen die Konstruktion von auf die Datenkodierung adaptierten Algorithmen detailliert analysiert werden.

Die existierenden Arbeiten werden in der Tabelle 3.11 gegenübergestellt und hinsichtlich des Stealth-Grades und der bereits erfolgten praktischen Validierung durch Umsetzung in ein ausführbares System mit allgemeiner Verfügbarkeit bewertet.

Tabelle 3.11 Vergleich existierender sicherer verteilter Ausführungen

Ansatz	3V-Notation	Umsetzung
Hacıgümüş et al.	LEC	nicht bekannt
Halang et al.	LES	nicht bekannt
Agrawal und Emekci	¯DEC	nicht bekannt
Kerschbaum	LEC	nicht bekannt

Gemäß der Tabelle 3.11 fällt kein System in die Stealth-Klasse $\mathscr{S}$. Stealth-Systeme stellen damit einen neuen Beitrag zur sicheren Operationsausführung in verteilten Umgebungen dar.

Zur weiteren Untersuchung soll im nächsten Schritt untersucht werden, welche Operationen sich auf stealth-kodierten Daten mit welchen Datentypen unter welchen Bedingungen ausführen lassen. Die in den nächsten Abschnitten vorgestellten Algorithmen werden sowohl für den Fall nur dispergierter Daten als auch für den Fall stealth-kodierter, also dispergierter und individuell verschlüsselter, Daten analysiert.

3.4.3.3 Zusicherungen nichtfunktionaler Eigenschaften

Die Stealth-Datenverarbeitung erlaubt es, weitgehend unabhängig von den nichtfunktionalen Zusicherungen einzelner Dienstanbieter bestimmte Mindestzusicherungen für das resultierende dienstübergreifende Gesamtsystem zu erhalten. Abbildung 3.30 stellt den Einfluss der Stealth-Kodierung in einer vierdimensionalen Sicht auf die erzielbaren NFE dar. Erkennbar ist deren Flexibilisierung und somit die effektive Kontrollierbarkeit durch den Anwender. Unabhängig von der Verschlüsselung ergibt sich allein durch die Aufteilung der Daten ein bestimmtes Maß an Vertraulichkeit.

3.4.4 Konstruktion und Analyse von Algorithmen

Ähnlich wie die Datenkodierung ist auch die Gestaltung und Anwendbarkeit von Algorithmen auf dispergierte und stealth-geschützte Daten stark typ- und zudem operationsabhängig. Ein passendes Programmiermodell für verteilte Algorithmen ist Map-Reduce, das basierend auf Sprachkonstrukten funktionaler Programmiersprachen einen verteilbaren und parallelisierbaren Map-Schritt sowie einen letztlich zentral arbeitenden Reduce-Schritt aufweist [UMJ+14, AGLP01]. Dabei sollte letzterer unter der vollständigen Kontrolle des Aufrufers stehen, während der Map-Schritt auf geschützten Daten in unsicheren Umgebungen ablaufen kann [SS14a].

Generell gilt, dass die Algorithmen stets höchstens auf den k signifikanten Fragmenten arbeiten, da redundante Informationen mehrdeutig sein können. Beispielhaft sei dies bei der surjektiven XOR-Redundanz für $k = 2$ für die Werte $a = 72$ und $b =$

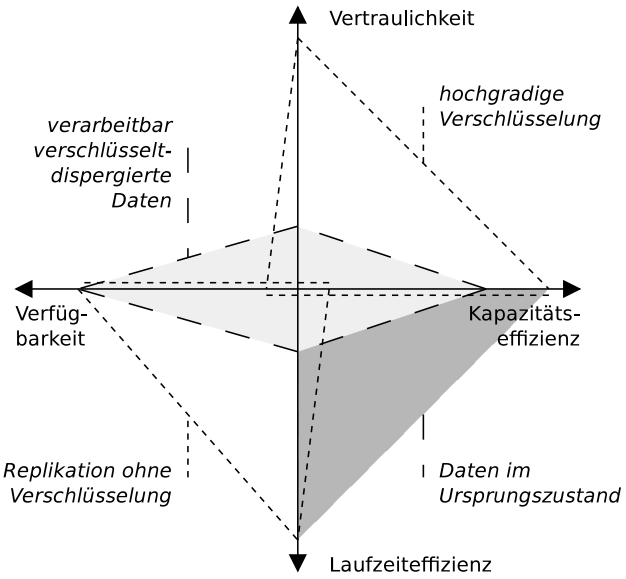

Abb. 3.30 Mehrdimensionale Sicht auf durch Kodierungsverfahren erzielbaren wünschenswerten NFE

132 in Hexadezimalschreibweise gezeigt: $a = \{0x04, 0x08\}; b = \{0x08, 0x04\}; a_r = b_r = 0xC0\square$. Dies impliziert, dass die Algorithmen nur bedingt ausfallsicher arbeiten und entweder eine Blockierung während der Wiederherstellungsphase oder eine qualitative Degradierung der Ergebnisdaten bezogen auf die Präzision oder Vollständigkeit verursachen [SS14a]. Andererseits lässt sich für bestimmte Algorithmen die Verteilung der Fragmente ausnutzen, um eine Beschleunigung der Ausführung gegenüber dem alternativen Verfahren, also der Zusammenführung aller Daten für eine lokale Berechnung, oder sogar eine absolute Beschleunigung gegenüber bereits lokal vorliegenden Daten erreichen.

Im Folgenden werden adaptierte sowie neu entworfene Algorithmen zur Verarbeitung dispergierter und teilweise stealth-geschützter Daten vorgestellt. Die Algorithmen umfassen den Vergleich von Daten, davon abgeleitet die Suche über Daten, arithmetische Operationen und Funktionen sowie wiederum abgeleitet Aggregationen. Anschließend werden die Algorithmen auf einer theoretischen Ebene hinsichtlich ihres Einsatzes in sicheren Anwendungen innerhalb von unsicheren Umgebungen bewertet.

3.4.4.1 Einteilung der Algorithmen

Abbildung 3.31 unterscheidet die Dispersed-Processing-Algorithmen nach ihren Anforderungen hinsichtlich der Dienstnutzung. Die 1-suffizienten Algorithmen erzielen das endgültige Ergebnis mit den Daten eines Dienstes, welcher meist derjeni-

ge ist, der den Fragmentstrom mit den höchstsignifikanten Bits (*most-significant bit*, MSB) beinhaltet. Die $[1-k]$-suffizienten Algorithmen erzielen bereits mit einem Dienst ein erstes Ergebnis, welches aber unvollständig oder unpräzise ist. Durch Hinzunahme weiterer Dienste lässt sich das Ergebnis vervollständigen. Schließlich sind die k-suffizienten Algorithmen diejenigen, die selbst bei $k-1$ Diensten keine verwertbaren eindeutigen Ergebnisse liefern.

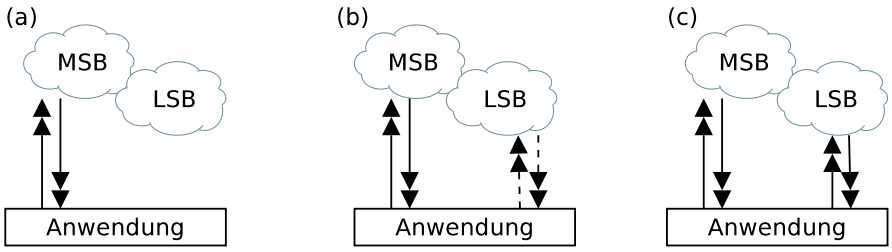

Abb. 3.31 Varianten der Dienstnutzung für die Verarbeitung dispergierter Daten: (a) 1-suffizient, (b) $[1-k]$-suffizient, (c) k-suffizient

Die Algorithmen werden vorrangig für drei grundlegende Datentypen analysiert: Ganzzahlen (*integer* bzw. *int*), Fließkommazahlen (*float*) und Zeichenketten (*string*). Tabelle 3.12 gibt einen ersten Überblick über die zu erwartende Einteilung. Der Laufzeitgewinn gegenüber einer nichtdispergierten Verarbeitung ergibt sich durch die eingesparte Netzwerkübertragungszeit abzüglich der erhöhten lokalen Ausführungszeit und ist mitunter positiv für aggregierende Algorithmen, hingegen für nichtaggregierende stets negativ, und für inhaltsunabhängige Funktionen stets Null.

Tabelle 3.12 Einteilung der Algorithmen zur Verarbeitung dispergierter Daten nach ihren Laufzeiteigenschaften

Algorithmus	Fragmentanzahl	Datentypen	Laufzeitgewinn
CMP()	k	int,float,string	< 0
SEARCH()	k	string	< 0
COUNTUNICODE()	1	string	> 0
ADD(),SUB(),MUL(),DIV()	k	int,float	< 0
LENGTH()	1	string	$< 0 / > 0$
ROUND(), FLOOR(), CEIL()	$1 \ldots k$	int,float	< 0
ABS()	k	int,float	< 0
SIN(), COS(), TAN()	k	int,float	< 0
SQUARE(), SQRT()	k	int,float	< 0
UPPER(), LOWER()	k	string	< 0
SUM(), AVG()	k	int,float	> 0
MEDIAN()	k	int,float	> 0
MIN(), MAX()	k	int,float	> 0
COUNT()	1	int,float,string	$= 0$

3.4.4.2 Vergleich von Daten

Die Gleichheit bzw. Ungleichheit zweier Daten gleicher Länge lässt sich für den Fall dispergierter Daten auf die Und-Verknüpfung der Gleichheit bzw. Ungleichheit aller geordneten Fragmente zurückführen. Es gilt: $cmp(a,b) = \bigwedge_{i=1}^{k} cmp_{disp}(a_i,b_i)$. Die Funktion cmp_{disp} kann dabei als Map-Schritt verteilt ablaufen, wohingegen cmp als Reduce-Schritt zentral abläuft.

3.4.4.3 Suche in Daten

Algorithmen zur Suche lassen sich mehrdimensional klassifizieren. Eine gängige Unterscheidung ist die Art der Suche nach einem Schlüsselwort, einem frei im Text vorkommenden Textmuster oder einem regulären Ausdruck. Zwar basieren Suchen auf Vergleichen, als Rückgabewert wird jedoch eine Position oder eine Liste von Positionen zurückgegeben. Damit ergibt sich bei der Suche über dispergierter Daten das Risiko falsch positiver Teilantworten, so dass bei der Zusammenführung der Ergebnisse diese durch die Bildung einer Schnittmenge aussortiert werden müssen. Es gilt demnach: $search(needle) = \cap_{i=1}^{k} search_{disp}(needle_i)$.

Im Fall von Stealth-Daten muss die Suchfunktion auch an die Verschlüsselung der Daten angepasst sein.

3.4.4.4 Arithmetik

Das Rechnen mit dispergierten Daten bedingt die Festlegung auf die Ordnung der Fragmente. Im Folgenden wird davon ausgegangen, dass bei k durch Bitsplitting entstandenen Fragmenten f_1 das höchstwertige Bit und f_k das niederwertigste Bit enthält. Die Fragmente werden durch Verschiebung und Überlagerung zum Ursprungswert zusammengeführt. Da aufgrund von Rechenoperationen der Ergebnisanteil jedes Fragments beliebig viele Bits groß sein kann, ist stattdessen die Verschiebung jedes Fragments f_i um die kumulierte Fragmentbreite $\sum_{j=i+1}^{k} fw_j$ in Kombination mit einer funktionsabhängigen Zusammenfügung wie der Summenbildung $\sum$ vorgesehen. Zur Vereinfachung der Notation wird die kumulierte Fragmentbreite als Verschiebungsfaktor $\overleftarrow{f_i}$ eingeführt.

Die betrachteten arithmetischen Algorithmen setzen die vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division auf dispergierte Daten um. Neben dem Trivialfall mit null dispergierten Elementen respektive zwei Konstanten sind ein Element mit einer Konstante sowie zwei Elemente zu betrachten, so dass insgesamt acht Fälle entstehen.

Auf Basis des Assoziativgesetzes lassen sich die Fälle der Addition und Subtraktion mit einem Element konstruieren. Somit gilt für ein Element a und für eine Konstante K: $add(a,K) = \sum_{i=1}^{k} \overleftarrow{a_i} + K = add_{disp}(a_k,K) + \sum_{i=1}^{k-1} \overleftarrow{a_i}$ und $sub(a,K) = \sum_{i=1}^{k} \overleftarrow{a_i} - K = sub_{disp}(a_k,K) + \sum_{i=1}^{k-1} \overleftarrow{a_i}$.

Auf Basis des Distributivgesetzes $(a+b)c = ac+ac$ lassen sich die Fälle der Multiplikation und Division mit einem Element in nahezu allen Varianten konstruieren. Allerdings ist zu beachten, dass die Division nicht kommutativ und auch nur rechtsdistributiv ist und somit zwingend der Divisor eine Konstante sein muss, also $\frac{a}{K}$. Somit gilt: $mul(a,K) = mul(K,a) = \sum_{i=1}^{k} \overleftarrow{mul_{disp}(a_i,K)}$ und $div(a,K) = \sum_{i=1}^{k} \overleftarrow{div_{disp}(a_i,K)}$. Neben einzelnen Zahlen lassen sich auch Matrizen derartig multiplizieren, sofern eine dispergierte Matrix und eine konstante nichtdispergierte Matrix vorliegen.

Sind zwei dispergierte Elemente a und b zu verarbeiten, gilt die folgende Additionsregel für $k \geq 2$: $add(a,b) = \sum_{i=1}^{k} \overleftarrow{add_{disp}(a_i,b_i)}$. Für die Multiplikation einer Zahl mit sich selbst ist hingegen mindestens $n = 3(k = 2, m = 1)$ erforderlich, für weitere Multiplikationen sogar $k = 4$, was einen höheren Kapazitätsbedarf bewirkt. Abgeleitet von der ersten binomischen Formel $(a_1+a_2)^2 = a_1^2+2a_1a_2+a_2^2$ wird jede zu quadrierende Zahl in zwei signifikante Fragmente a_1 und a_2 sowie ein redundantes Fragment $r = a_1a_2$ zerlegt. Dann gilt für eine Bitsplitting-Kodierung mit der niederbittigen Fragmentbreite fw_2: $mul(a,a) = (a_1 \ll fw_2)^2 + a_2^2 + r \ll (fw_2+1)$.

3.4.4.5 Abbildungsfunktionen

Funktionen bilden einen Wert aus einem Definitionsbereich auf einen Wert aus einem Wertebereich ab, wobei sowohl numerische als auch nichtnumerische Werte zulässig sind. Prädikatsfunktionen sind solche, deren Wertebereich die Menge der booleschen Zahlen $\{0,1\}$ ist.

Die Funktionen *upper* und *lower* beziehen sich auf Zeichenketten und bilden deren Buchstaben auf die Groß- und Kleinschreibung ab, sofern diese alphabetisch definiert ist. Beide Funktionen arbeiten mit dispergierten Daten, wenn die Dispersion mit $w = 8$ auf Buchstabenebene bei Annahme einer ASCII-Kodierung erfolgt, wobei der Reduce-Schritt die Konkatenation ist. Es gilt: $upper(text) = concat_{i=1}^{k} upper_{disp}(text_i)$. Erfolgt hingegen die Dispersion mit $w = 4$, müssen diese Funktionen zentral nach dem Abruf der Daten ausgeführt werden.

Die Funktionen *round*, $floor$ und *ceil* sind dann geeignet realisierbar, wenn die Dispersion auf Fließkommazahlen so angewandt wird, dass die Fragmentbildung mit $k \geq 2$ an der Kommaposition erfolgt. Es gilt dann: $floor(number) = number_1$, $ceil(number) = number_1 + 1$.

Die Quadrierfunktion *square*, mitunter auch *pow* genannt, ergibt sich direkt aus der Multiplikation einer Zahl mit sich selbst. Hingegen erfordert die Wurzelbestimmung *sqrt* ein iteratives mathematisches Verfahren, das die Einteilung einer Zahl in Zweiergruppen von Ziffern einer bestimmten Basis voraussetzt. Unter der Annahme einer dekadischen Basis muss somit für eine Zahl z die Voraussetzung $k = \lceil lg(z)/2 \rceil$ erfüllt sein. Ebenso muss das dekadische Dispersionsverfahren eingesetzt werden. Der Algorithmus zur Wurzelbestimmung ist im Quelltext 3.8 schematisch dargestellt. Auffällig ist, dass zwei Daten in der Iteration übertragen werden müssen.

Listing 3.8 Berechnung einer Quadratwurzel auf dispergierten Daten

```
fragments = [5, 43, 21, 00, 00, 00]
rest = 0  # global
result = 0  # global
for f in fragments:
  sub = (result * 2) * 10 + 1
  count = 0
  f += rest * 100
  while f >= sub:
    f -= sub
    sub += 2
    count += 1
  result *= 10
  result += count
  rest = f
```

Abhängig von der Kodierung wird die Funktion *abs* ausgeführt. Ist die Zahl mit einem Vorzeichenbit versehen, gilt vorteilhaft: $abs(number) = \overleftarrow{abs_{disp}(number_1)} + \sum_{i=2}^{k} \overleftarrow{number_i}$ oder alternativ per Überlagerung mit einer OR- oder XOR-Verknüpfung ohne Summierung $\overleftarrow{abs_{disp}(number_1)} \oplus \bigoplus_{i=2}^{k} \overleftarrow{number_i}$. Ist sie andernfalls in der Zweierkomplementdarstellung kodiert, so muss revers iteriert werden, um die Interpretation des Vorzeichens durch Änderung der Bitfolge umzukehren. Es gilt dann: $abs(number) = \sum_{i=k}^{1} \overleftarrow{abs_{disp}(number_i)} = \bigoplus_{i=k}^{1} \overleftarrow{abs_{disp}(number_i)}$, wobei die Umsetzung von abs_{disp} ab dem ersten belegten Bit dieses und alle weiteren invertiert, so dass dieser Zustandswechel in der Iteration übergeben werden muss.

Viele mathematische Funktionen, unter ihnen die elementaren trigonometrischen Funktionen *sin*, *cos* und *tan*, lassen sich aufgrund fehlender Distributivgesetze nicht auf dispergierte Daten anwenden. Für $k = 2$ gilt jedoch zumindest das Additionstheorem $sin(a_1 + a_2) = sin(a_1)cos(a_2) + sin(a_2)cos(a_1)$, welches genutzt werden kann, wenn konträr zum Bitsplitting-Verfahren keine Bitverschiebung der Fragmente stattfindet.

3.4.4.6 Aggregation

Aggregationsfunktionen entsprechen Reduce-Schritten, indem sie eine Liste von Werten auf einen einzelnen Wert abbilden. Eine dispergierte Aggregation ist demnach eine mehrfach parallel und hintereinander ablaufende Folge von Reduce-Schritten. Dabei ist der letzte Reduce-Schritt von den ersten abhängig. Wird beispielsweise verteilt die Summe berechnet, so werden erst Teilsummen bestimmt, die anschließend wieder per Addition zusammengeführt werden. Es gilt: $sum(list) = \sum_{i=1}^{k} \overleftarrow{sum_{disp}(list_i)}$. Hingegen ist die Bestimmung der Länge einer Liste auf bereits einem Fragmentstrom möglich und erfordert nur zur Gewährleistung der Integrität

eine Anwendung auf die weiteren Ströme. Es gilt: $count(list) = count_{disp}(list_i), 1 \leq i \leq k$ beliebig. Die Bestimmung des arithmetischen Mittels ist ein abgeleitetes Prädikat nach der Formel $avg(list) = sum(list)/count(list)$. Es kann allerdings auch direkt bestimmt werden: $avg(list) = \sum_{i=1}^{k} \overleftarrow{avg_{disp}(list_i)}$.

Weitere statistische Aggregatsionsfunktionen einschließlich Minimum, Maximum und Median lassen sich auf dispergierten Daten definieren, erfordern jedoch die Aufgabe der strikten Isolation der Fragmentströme und der beliebigen Parallelisierbarkeit. Stattdessen wird ein iteratives Vorgehen benötigt, welches auf dem erweiterten Programmiermodell Map-Carry-Reduce aufbaut.

3.4.4.7 Programmiermodell Map-Carry-Reduce

In verschiedenen Konfigurationen lässt sich die Berechnung über verteilte Daten parallel durchführen. Sind die Daten beispielsweise per Round-Robin-Schema auf mehrere Dienste verteilt, so liefert eine Abbildungsfunktion wie $SIN()$ zu jedem Wert eines jeden Dienstes ein Resultat. Somit entsteht für jeden Dienst eine Ergebnisliste. Clientseitig werden diese Listen konkateniert oder anderweitig, etwa per Reißverschlussverfahren, zusammengeführt. Dies entspricht dem Programmiermodell *Master-Slave*. Sind die Daten jedoch dispergiert, ist die rein parallele Ausführung nicht mehr ausreichend. Stattdessen müssen die Ergebnislisten weiterverarbeitet werden. Hierfür bietet sich das angepasste Programmiermodell *Map-Reduce* an, welches im Reduce-Schritt eine lokale Berechnung auf der Liste der Ergebnislisten durchführt.

Für einige Funktionen ist jedoch auch *Map-Reduce* nicht ausreichend, da der Grad der Parallelität eingeschränkt ist. Ein Beispiel hierfür sind die ordnungsabhängigen Berechnungen. Durch ein Übergabeschema müssen teilweise sekundäre Ergebnisdaten erst von einem Dienst an den Client zurückgegeben werden, woraufhin der Client diese dann an den nächsten Dienst zur weiteren Berechnung überträgt. Dies erzeugt das iterative Programmiermodell *Map-Carry-Reduce*. Als Übergabedaten kommen Positionsangaben oder Positionsbereiche, Zwischenergebnisse oder weitere funktionsspezifische Werte in Frage. Abbildung 3.32 zeigt das Map-Carry-Reduce-Modell auf der abstrakten Ebene für den Fall $k = 3$. Die Ergebnisliste α für eine Aggregation hat anfänglich einen hohen Umfang $|\alpha|$. Der Umfang schrumpft zunehmend, so dass bereits für die zweite Ergebnisliste β gilt: $|\beta| \leq |\alpha|$. Schließlich verbleibt die finale Ergebnisliste $\hat{X}$. Wird die Iteration vorzeitig abgebrochen, so kann gezielt eine verbesserte Laufzeit gegenüber einer verringerten Präzision des Ergebnisses forciert werden.

Zwei konkrete Beispiele werden in den Abbildungen 3.33 und 3.34 erläutert. Abbildung 3.33 zeigt die Funktionen *MIN* und *MAX* im Map-Carry-Reduce-Modell. Hierbei werden zuerst die Werte und Positionen mit dem kleinsten respektive größten MSB-Wert durch Ordnung aller Werte bestimmt. Treten diese Werte mehrfach auf, wird die Ergebnisliste α auf die Duplikate erweitert. Anschließend wird α gegebenenfalls über Zwischenstationen (β etc.) an die letzte Station übergeben. Dort

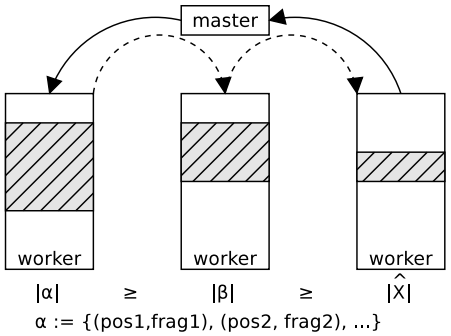

Abb. 3.32 Schema für die Anwendung des Map-Carry-Reduce-Protokolls über dispergierte Fragmente

wird die finale Ergebnisliste $\hat{X}$ analog durch Sortierung und Bestimmung des kleinsten und größten LSB-Wertes erstellt.

Abbildung 3.34 zeigt die Funktion *MEDIAN* im Map-Carry-Reduce-Modell. Eine Besonderheit ist hierbei, dass die finale Ergebnismenge zwei Werte enthalten kann, also $|\hat{X}| = 2$ gilt, wonach der letzte Rechenschritt $MEDIAN = (\hat{X}) = 5{,}5$ erfolgen muss.

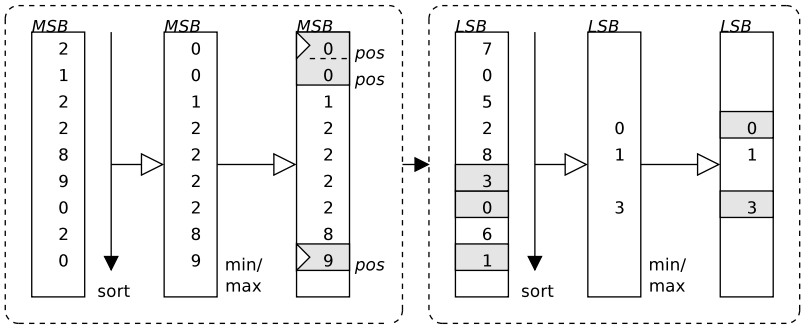

Abb. 3.33 Anwendungsbeispiel des Map-Carry-Reduce-Protokolls für die ordnungsabhängigen Funktionen MIN und MAX

3.4.4.8 Verarbeitung redundanter Daten

Üblicherweise werden redundante Daten zwar geschrieben, jedoch nur im Fehlerfall für die Wiederherstellung unrettbarer signifikanter Daten wieder eingelesen. Für die direkte Verarbeitung redundanter Daten existieren keine Ansätze. Ein Hauptgrund dafür ist, dass die Redundanzberechnung zumeist über die XOR-Verknüpfung zweier oder mehrerer Datenteile erfolgt, was ähnlich der Anwendung einer Hashfunktion

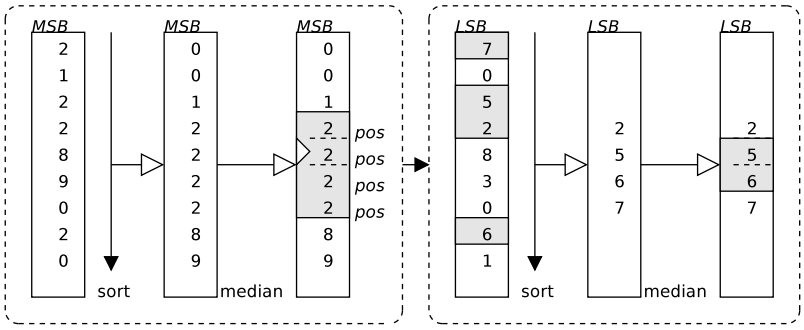

Abb. 3.34 Anwendungsbeispiel des Map-Carry-Reduce-Protokolls für die ordnungsabhängige Funktion MEDIAN

einen Informationsverlust mit sich bringt. Abbildung 3.35 zeigt, dass mehrere unterschiedliche XOR-Verknüpfungen auf die selben XOR-Ergebniswerte abgebildet werden. Damit ist das Verfahren nicht strukturerhaltend.

```
A XOR A = 0     A XOR B != 0     000 = {AAA XOR AAA, BBB XOR BBB, CCC XOR CCC,
B XOR B = 0     B XOR C != 0           AAB XOR AAB, ABA XOR ABA, BBA XOR BBA,
C XOR C = 0     C XOR A != 0           ABB XOR ABB, BAB XOR BAB, BBA XOR BBA, ...}
```

Abb. 3.35 XOR-Redundanzdaten für Buchstaben und Buchstabenfolgen

Dennoch können auch redundante Daten einen Beitrag zur dispergierten Verarbeitung leisten, indem unabhängig von der Struktur bestimmte statistische Merkmale ausgenutzt werden. Dies führt in einem ersten Schritt zu einer semantisch interpretierbaren Redundanzdatenmenge, die als *meaningful redundancy* bezeichnet werden soll. Lassen sich darüber anschließend Algorithmen konstruieren, so spreche man von einer verarbeitungsfähigen Redundanzmenge oder *processable redundancy*. Diese wird aufgrund des Informationsverlustes stets heuristisch arbeiten, kann damit jedoch für bestimmte Aufgaben dennoch geeignet sein, falls darauf geachtet wird, dass nur falsch-positive, nicht jedoch falsch-negative, Ergebnisse entstehen, da sich die falsch-positiven Resultate in einem nachgelagerten Schritt bei Kenntnis von $k-1$ weiteren Datenmengen erkennen und eliminieren lassen. Vorgeschlagen wird dafür ein Text-Suchalgorithmus, welcher sich in seiner Funktionsweise auf die sprachabhängigen statistischen Unterschiede in der Buchstabenverteilung in Wörtern stützt [BFP14, SK76]. Als Größe der Verarbeitungseinheit ist damit $w = 8$ Bit gesetzt, wobei auch andere Größen wie $w = 4$ zulässig sind.

Der naive Suchalgorithmus für $w = 8$ ist im Quelltext 3.9 in Python-Notation angegeben. Er wird auf redundanten Daten *content* mit dem ebenfalls als Redundanzkodierung aus dem Suchbegriff entstandenen *keyword* angewandt, die beide durch die Verknüpfung von je zwei aufeinander folgenden Buchstaben entstanden sind. Ein Suchwort kann nun für jede Position an erster oder zweiter Stelle gestanden haben, so dass der Algorithmus zur Verhinderung falsch-negativer Ergebnisse stets auch auf $keyword'$ ausgeführt werden muss, das entsteht, indem vom ursprüng-

lichen Suchbegriff der erste Buchstabe entfernt wird. Dies entspricht dem Vorgehen der dispergierten Suche [SS14a]. Gleichzeitig führt dieses Vorgehen zu einer deutlich höheren Rate an falsch-positiven Ergebnissen, wobei das Verhältnis derer zu den tatsächlichen Ergebnissen noch einer Untersuchung bedarf.

Listing 3.9 Heuristische Suche über redundante Daten

```
def redundantsearch(content, keyword):
  matches = 0
  cpos = 0 # content position
  kpos = 0 # keyword position
  while cpos < len(content):
    if content[cpos] == keyword[kpos]:
      kpos += 1
      cpos += 1
      if kpos == len(keyword):
        # match position in original text: (cpos - kpos) * 2
        matches += 1
        kpos = 0
    else:
      cpos = cpos - kpos + 1
      kpos = 0
  return matches
```

Die duale Kodierung mit dem XOR-Operator^der redundanten Suchbegriffdaten *searchword* in *keyword* und *keyword*′ wird im Quelltext 3.10 ergänzend dargestellt. Die Kodierung von *content* entspricht dabei der von *keyword*.

3.4.4.9 Bewertung der Algorithmen

Anhand der Untersuchungen zu den Algorithmen wird deutlich, dass das Rechnen auf dispergierten und verschlüsselten Daten zu starken funktionalen Einschränkungen und zudem zu teilweise extremen Laufzeiteinbußen führt. Ein allgemeiner Einsatz kann demnach nicht empfohlen werden. Dennoch gibt es Situationen, in denen die auf eine bestimmte Kodierung adaptierte Berechnung benötigt wird. Das ist dann der Fall, wenn eine Neukodierung nicht durchführbar ist oder wenn die Eigenschaften der Kodierung von höchster Bedeutung sind.

Der Grad der Adaptivität der Algorithmen wird durch die Dispersion erhöht. Damit entsteht eine nutzerkontrollierbare Flexibilität hinsichtlich der Abwägung zwischen Laufzeit, Präzision und Umfang, womit bestimmte Methoden des *approximate computing* und des *imprecise/significance-based computing* anwendbar werden [MPP15, NVB+14].

Die Laufzeiteinbußen können unter Umständen durch Einsparungen in der Datenübertragung über Netzwerke aufgefangen werden. Dieser Effekt lässt sich zumindest für Aggregationsfunktionen prognostizieren. Es existiert demnach ein *per-*

Listing 3.10 XOR-Kodierung redundanter Textdaten

```
def encoderedundantkeywords(searchword):
  searchwordalt = searchword
  if len(searchword) % 2 == 1:
    searchword = searchword + " "
  if len(searchwordalt) % 2 == 0:
    searchwordalt = searchwordalt + " "
  keyword = []
  for i in range(len(searchword) / 2):
    c1 = searchword[i * 2]
    c2 = searchword[i * 2 + 1]
    r = ord(c1) ^ ord(c2)
    keyword.append(r)
  keyword' = []
  for i in range(len(searchwordalt) / 2 - 1):
    c1 = searchword[i * 2 + 1]
    c2 = searchword[i * 2 + 2]
    r = ord(c1) ^ ord(c2)
    keyword'.append(r)
  return keyword, keyword'
```

formance sweetspot zwischen dem Abrufen und anschließendem lokalen Berechnen und dem direkten dispergierten Berechnen. Analog zur Existenz von weiteren algorithmischen Sweetspots, beispielsweise zur Energieeffizienz [GIC+14a], ist die Auswertung zur Laufzeit und die Adaption der Ausführung als vorteilhaft für die Gewährleistung zusicherbarer Eigenschaften anzunehmen.

3.5 Zusammenfassung der Beiträge

Im Vergleich mit existierenden Ansätzen liefert diese Arbeit neue Beiträge im Bereich Datenkodierung (Dispersion mit bitweiser Aufteilung und mit visueller Aufteilung), in der Datenverteilung (Erzielung einer Mindestverfügbarkeit) und schließlich in der Applikation von Algorithmen auf derartig aufgeteilten Daten. Die Konzepte Dispersed Computing und Stealth Computing nutzen diese, um robuste und ausfallsichere Anwendungen in nicht vertrauenswürdigen Umgebungen bei minimalem zusätzlichen Platzbedarf zu ermöglichen. Die Datenrelationsschemen DRS und FRS stellen zudem eine erste formalisierte Grundlage für den interoperablen Informationsaustausch über derartige Verteilungen dar.

Kapitel 4
Stealth-Konzepte für zuverlässige Dienste und Anwendungen

Zusammenfassung In diesem Kapitel werden Schnittstellen, Architekturen, Plattformen und plattformäquivalente Integrationsmerkmale für langlebige native Anwendungen und Anwendungsdienste konzeptionell erarbeitet. Diese sind an existierende zentrale Cloud-Plattformen für die Entwicklung und den Betrieb von zumeist gering abgesicherten Anwendungen angelehnt, sichern jedoch durch die optimale Ausnutzung von Datenrelationen, -kodierung, -verteilung und -verarbeitung aufbauend auf den im vorherigen Kapitel dargelegten Konzepten langfristig Garantien in unzuverlässigen Ausführungsumgebungen zu. Fünf datenbezogene Plattformdienstklassen werden identifiziert und mit einer Stealth-Äquivalenz versehen.

4.1 Überblick über Plattformkonzepte und -dienste

Im Umfeld von Cloud-Computing ist eine geschichtete Anwendungsarchitektur üblich. Anwendungen und Anwendungsdienste in der Cloud laufen in einer Ablaufumgebung [SBS09, SBS11], wobei die Bestandteile der Architektur entweder vom Bereitsteller selbst als Teil der Anwendung mitgeliefert werden, beispielsweise als Hilfsdienste innerhalb einer virtuellen Maschine, oder zur feingranulareren Skalierbarkeit oder verstärkten Auslagerung von Funktionalität als Plattformdienste (PaaS) eingebunden werden. Alternativ kann die Anwendung vollständig oder teilweise außerhalb der Cloud laufen und über die PaaS-Schnittstellen Daten oder Funktionen nutzen.

Es existiert keine abgeschlossene Definition von Plattformdiensten. Stattdessen sind sie in ihrem Funktionsumfang und ihrer Namensgebung stark von Trends aus der Softwareentwicklung abhängig. Beispielhaft steht *Mobile-Backend-as-a-Service* (MBaaS) für die Entlastung mobiler Anwendungen durch eine reichhaltige Datenhaltung und -analyse in der Cloud [GS14, Nit13]. Weitere Plattformdienste, die sich an Anwendungsentwickler richten, sind Datenbankdienste (DBaaS) [LS10, AAEM09], Ereignisverarbeitung (EPaaS bzw. CEPaaS) [MBF14], Speicherplatz (StaaS, sicherer Speicherplatz: SSaaS; teilweise IaaS) [SG14], Workflows

[MPP14] und Netzwerkfunktionen wie Proxys oder Routen (NaaS) [KR10]. Weitere Plattformdienste wie CDNaaS oder FaaS beziehen sich eher auf die Auslieferung und den Betrieb von Anwendungen und sind für den Anwendungsentwicklungsprozess nicht von Bedeutung. In diesem Abschnitt wird nicht auf konkrete PaaS-Anbieter eingegangen; umfangreiche Analysen und Vergleiche existieren jedoch bereits [WTJ14, BMW11] ebenso wie diverse Architekturvorschläge für zuverlässige und anbieterübergreifende PaaS-Angebote [CPV+13, Kri13, SPM12].

In der Abbildung 4.1 werden komplexe PaaS-Konstellationen für Anwendungen und Anwendungsdienste aufgezeigt. Dabei wird deutlich, dass der Anwender einer höheren Zahl an Dienstanbietern hinsichtlich der Einhaltung von Garantien und Zusagen entweder vertrauen oder per vertraglicher Regelung rechtssicher begegnen muss. Zwar ziehen derartige PaaS-Angebote Vorteile aus der vertieften Integration mit IaaS-Ressourcendiensten, eine kontrollierbare und flexible Nutzung insbesondere über Anbietergrenzen hinweg wird hingegen erschwert.

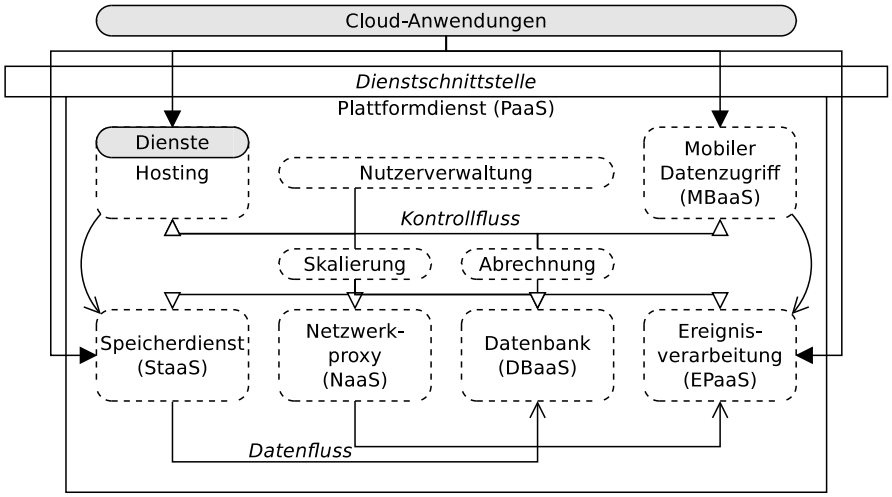

Abb. 4.1 Nutzung konventioneller PaaS-Dienste in Cloud-Anwendungen

Alternativ bietet es sich an, die PaaS-Funktionen in der Vertrauens- und Kontrolldomäne des Anwenders nachzubilden und sie somit anbieterübergreifend nutzen zu können, ohne gleichzeitig die wünschenswerten Cloud-Eigenschaften wie elastische Skalierbarkeit, nutzungsbezogene Abrechnung und ständige Verfügbarkeit aufgeben zu müssen [SS12b, TPH+14]. Dieser Ansatz wird beispielsweise bereits für die automatische Skalierung von Ressourcen ausprobiert und führt dabei verglichen mit anbieterseitigen PaaS-Angeboten zu einer effizienteren Lösung [ZLHD14]. Weiterhin ist der Ansatz im Umfeld persönlicher oder hybrider vertrauenswürdiger Cloud-Umgebungen mit anbieterübergreifenden Verwaltungsoberflächen im Sinne eines *vendor-relationship management* als integrierte Software, Gateway-Dienst oder Hardware-Appliance bereits untersucht worden [SSZ13, SZS13]. Im

Rahmen dieser Arbeit soll der prinzipielle Ansatz der Trennung von Daten- und Kontrollfluss nun auch für den Aufruf von durch Stealth-Techniken abgesicherter datenverarbeitender Funktionen genutzt werden. Dies bedingt eine alternative verteilte Anwendungsarchitektur und zu deren Realisierung auch eine alternative Software-Entwicklungsmethodik gegenüber den bisherigen Service-Engineering-Ansätzen [SKU+11]. Die vormals abstrakt betrachteten Daten werden nun auf konkrete Ausprägungsformen wie Dateien, Ströme oder Tupel konkretisierend abgebildet.

Abbildung 4.2 zeigt exemplarisch, wie eine fiktive Anwendung X mit Stealth-Plattformkonzepten für den Betrieb in unzuverlässigen Umgebungen strukturiert wird. Die Anwendung benötigt in der Annahme eine Datenbank. Anstelle der Nutzung eines DBaaS-Dienstes eines einzelnen Anbieters läuft die Datenbankverwaltungsfunktionalität als Schnittstelle in der Vertrauens- und Kontrolldomäne des jeweiligen Benutzers der Anwendung. Die Schnittstelle spricht korrespondierende verteilte Speicher- und Rechendienste auf der Plattform- oder direkt auf der Infrastrukturebene mehrerer Dienstanbieter an und ermöglicht somit eine relative Unabhängigkeit von den Anbietern und eine relativ erhöhte Dienstgüte hinsichtlich Verfügbarkeit, Vertraulichkeit und weiterer Schutzziele. Allerdings steigt der Verwaltungsaufwand an: Für jeden Anbieter müssen a-priori-Zugangsdaten sowie a-posteriori-Metadaten über die Datenverteilung verwaltet werden. Desweiteren müssen Informationen über die Anbieter selbst bekannt sein, etwa in Form eines Dienstverzeichnisses [SS13a, BSS+13], um sowohl beim erstmaligen Betrieb der Anwendung (*bootstrapping*) passende Dienste zu finden als auch über die Zeit unter Berücksichtigung der Dienstevolution ausgefallene oder unpassend gewordene Dienste ersetzen oder in einer horizontalen Skalierung ergänzen zu können. Schließlich ist ein Mehrfachbetrieb (*multiplexing*) der Dienste und ein darauf angepasstes Daten-Scheduling notwendig.

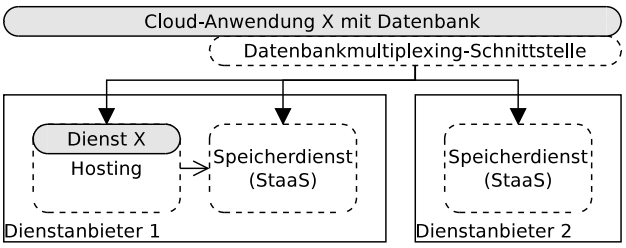

Abb. 4.2 Exemplarische Darstellung einer über Vertrauens- und Kontrolldomänen und Anbietergrenzen hinweg verteilten Cloud-Anwendung

In diesem Kapitel werden die Multiplexing-Schnittstellen für die relevanten lokalen datenverarbeitenden Plattformdienste Netzwerkproxy, Dateispeicherung, Stromspeicherung, Datenbank und Ereignisverarbeitung erläutert und passende Äquivalenzarchitekturen unter Nutzung reeller Infrastrukturdienste vorgestellt, so dass die Umsetzung der Verfahren keine oder nur minimale Anforderungen an die gemeinhin

als Cloud-Dienste bezeichnete Infrastrukturumgebung stellt. Anschließend wird aus der Softwareentwicklungsperspektive eine Darstellung der Integration der Dienste in die Anwendung sowie des Anwendungsentwicklungsprozesses insgesamt gegeben, bevor schließlich noch die Anforderungen an Funktion und Entwurf unterstützender Laufzeitmechanismen insbesondere für die Dienstverwaltung untersucht werden. Die nächsten drei Abschnitte erläutern dazu noch Betrachtungen zu Multiplexer-Architekturen, Stealth-Techniken und ergänzenden Absicherungsverfahren für Ressourcendienste auf der Infrastrukturebene.

4.1.1 Multiplexer-Architekturen für Ressourcendienste

Die bewusst gleichzeitige Nutzung mehrerer Ressourcen oder Dienste zur Umsetzung von Speicher- und Berechnungsfunktionen mit günstigen, mithin zusicherbaren, Eigenschaften sei nachfolgend als RSM-Prinzip (*Resource and Service Multiplexing*) bezeichnet. Der Begriff wird bewusst von der eher weitläufig interpretierbaren Terminologie der Multi-Cloud, föderierter Cloud-Dienste oder der Overlay-Cloud abgegrenzt [SCB+14]. Die Ressourcen umfassen dabei primär Datenspeicher (HDD, SSD, RAM-Disks, Wechselmedien) und Verarbeitungseinheiten (CPU und RAM) sowie beliebige Netzwerkadapter zur Kommunikation. Die betrachteten Infrastrukturdienste kapseln die Ressourcen mit standardisierten oder anbieterspezifischen Dienstschnittstellen. Analog werden Anwendungen, welche direkt oder über Bibliotheken oder Proxy-Dienste das RSM-Prinzip einsetzen, RSM-Anwendungen genannt. Stealth-Anwendungen sind demnach immer stets auch RSM-Anwendungen, wohingegen auch RSM-Anwendungen ohne Stealth-Eigenschaften existieren. Cloud-Anwendungen sind mitunter, allerdings nicht stets, RSM-Anwendungen, da sie teils nur mit zumindest scheinbar ressourcenunabhängigen Diensten interagieren. RSM-Dienste sind als Anwendungsdienste RSM-Anwendungen, die selbst wieder eine Dienstschnittstelle zur Verfügung stellen. RSM-Plattformdienste sind hingegen solche, die die Funktionalität eines PaaS über eine lokale Schnittstelle bereitstellen, dabei jedoch per Multiplexing nicht nur auf weit verteilte Ressourcendienste, sondern auch auf lokale Ressourcen ohne Dienstschnittstelle zugreifen können. Die Bezeichnung plattformäquivalente Dienste trifft somit ebenfalls auf sie zu.

Die generische Architektur von RSM-Anwendungen mit ihren Verbindungsoptionen, von denen jedoch nicht stets alle genutzt werden müssen, wird in der Abbildung 4.3 zusammenfassend dargestellt. Aus der Abbildung geht hervor, dass die Einbindung von Dienstschnittstellen für Berechnungsdienste stets, für Speicherdienste hingegen mitunter die vorherige Installation und Konfiguration einer Dienstimplementierung erfordert. Je nach Zielumgebung kann dies beispielsweise ein ausführbarer Anwendungscontainer, eine virtuelle Maschine oder eine auf einen vorhandenen Container passende Anwendung sein.

Der eigentliche Transport von Daten wird vom Client initiiert und läuft über eine variable Zahl an Zwischenstufen bis zum datenverarbeitenden Dienst. An dieser

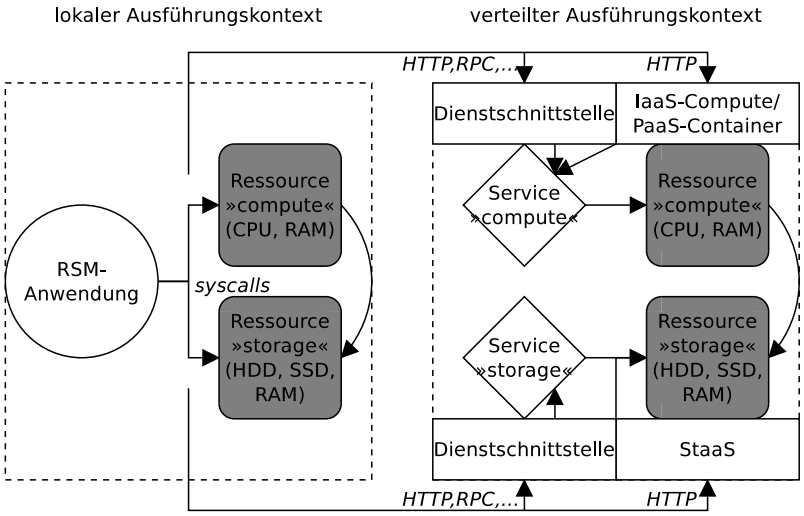

Abb. 4.3 Verbindungsoptionen von RSM-Anwendungen

Stelle soll der Transportweg für dateibezogene Speicherdienste genauer betrachtet werden. Die Daten können sowohl synchron als auch asynchron übertragen werden, wobei im ersten Fall die Anwendung erst dann eine Statusmeldung zum Schreibvorgang erhält, wenn die Daten physisch auf den Datenträger geschrieben oder zumindest im Fall einer Netzwerkübertragung an die Netzwerkschicht übergeben worden sind. Abbildung 4.4 zeigt den Unterschied der beiden Verfahren auf. Durch die Dispersion der Daten mit k signifikanten und m redundanten Fragmenten kann dieser Unterschied zur Sicherstellung der Wiederherstellbarkeit der Daten berücksichtigt werden.

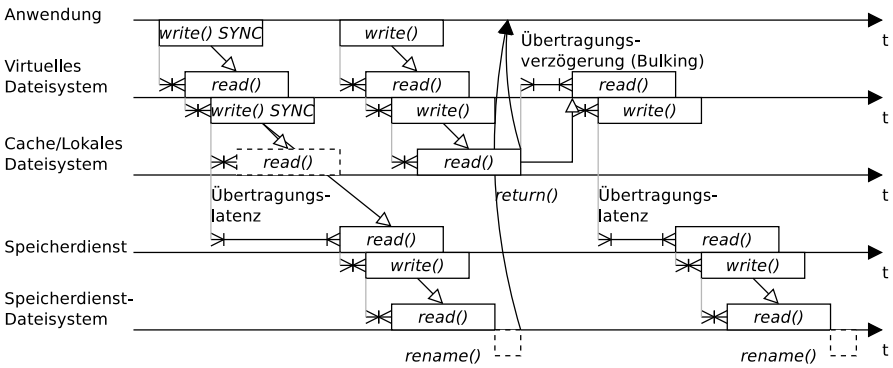

Abb. 4.4 Sequenz des Datentransports mit und ohne Synchronität

Im Fall synchroner Übertragung kann optional ein Cache eingebunden werden, um beim darauffolgenden Lesen Netzwerkzugriffe einzusparen; im Fall asynchroner Übertragung ist ein Cache zwingend notwendig. Weiterhin optional ist das Schreiben in eine temporäre Datei auf Seite des Dienstanbieters mit anschließender Umbenennung der Datei aus Konsistenzgründen. Sollen hingegen Datenströme übertragen werden, so entfällt dieser Schritt.

Für die zeitlich geordnete Übertragung der Daten und Fragmente stehen unterschiedliche Scheduling-Strategien vom seriellen bis zum parallelen Modus zur Auswahl. Falls nicht gesondert herausgestellt, wird der parallele Modus angenommen.

4.1.2 Stealth-Techniken für Ressourcendienste

Das RSM-Prinzip allein verschafft dem Anwender zwar mehr Kontrollmöglichkeiten, erhöht aber für sich genommen weder die Vertraulichkeit noch die Verfügbarkeit von Anwendungen. Dies erfordert den Einsatz stealth-kodierter Daten und daran angepasste Multiplexing-Verfahren, die im Folgenden generell als Stealth-Techniken bezeichnet werden sollen.

In der Praxis bedeutet das RSM-Prinzip mit Stealth-Techniken, dass ein Vorteil von Cloud-Infrastrukturdiensten, die oft zumindest vorgeblich kostenfreie Nutzung, ohne unerwünschte nachteilige Seiteneffekte und Risiken, welche zumeist versteckte nichtmonetäre Kosten nach dem Motto *the user is the product* enthalten, nutzen lassen. Vor allem für Privatanwender liefert dieser Aspekt einen Grund zum Einsatz von Stealth-Anwendungen.

Somit bezieht sich der abgeleitete Begriff *Dispersed Cloud Computing* auf die parallele Einbeziehung der Infrastrukturressourcen Speicherplatz, Netzwerkdurchsatz und Rechenleistung, welche jeweils durch elastische und kostenfreie oder nutzungsabhängig abgerechnete Infrastruktur- oder Plattformdienste gekapselt sind, mit Datendispersion, und *Stealth Cloud Computing* analog auf deren Einbeziehung mit Stealth-Techniken.

Die Verteilung von Daten erfolgt generell aus unterschiedlichen Gründen. Während die Dispersion vor allem dem Schutz der Daten dient, kann sie je nach Konstellation auch zur Optimierung weiterer nichtfunktionaler Eigenschaften beitragen. Beispiele für Optimierungen umfassen die Performance, welche zumeist über eine parallelisierte Ausführung erreicht wird, sowie die Energieeffizienz und Netzwerkoptimierung, welche man durch Auslagerung von Codebestandteilen über das Netzwerk, z.B. durch mobiles Code-Offloading in die Cloud, erreicht [BDR14]. Dies führt generell zu der Frage, wie Anwendungen topologisch über die Cloud konstruiert sein sollen. Abgesehen von reinen Peer-to-Peer-Architekturen, die zwar sehr skalierbar sind, bietet sich für Cloud-Umgebungen aufgrund der unterschiedlichen Eigenschaften von Cloud-Diensten und Geräteressourcen ein Ansatz an, der einen Client unter der Kontrolle des Benutzers mit nicht vertrauenswürdigen, instabilen und unzuverlässigen Diensten verbindet. Dies impliziert wiederum bestimmte

Programmiermodelle bei der Anpassung von Algorithmen auf die dispergierten und verschlüsselten Daten.

Im Cloud-Umfeld ergeben sich somit insgesamt für die Infrastruktur- und Plattformebene fünf hauptsächliche Dienstklassen, auf welche sich eine dispergierte respektive Stealth-Datenhandhabung vorteilhaft auswirkt. Diese sind im Einzelnen:

1. Speicherdienste. Ein Verbund von unabhängigen Speicherzielen bildet die Grundlage für sicheres Cloud Storage. Meist sind diese Ziele Dateispeicherdienste [MSMF14, SGS11], wobei auch die Einbindung lokaler Ressourcen möglich ist. Da bei einer reinen Datenspeicherung keine nachgelagerte Verarbeitung erfolgt, wird hierbei oft eine Dispersion mit Löschkodierung mit einer vorgelagerten beliebigen, meist symmetrischen, Verschlüsselung genutzt. Die entstehende Anwendungstopologie entspricht einer Y-Form ($1 : n$-Form), da ein Client mehrere Speicherziele anspricht.
2. Netzwerkproxys. Ein Verbund mehrerer unabhängiger Kommunikationskanäle wird für die sichere Übertragung von einzelnen Nachrichten oder Datenströmen genutzt. Viele Geräte sind mit mehreren physischen Netzwerkadaptern ausgestattet und erlauben somit eine Ende-zu-Ende-Dispersion der Übertragung. Ist dies nicht gegeben, so kann ein Proxy im lokalen Netzwerk dem Gerät diese Funktionalität zur Verfügung stellen. Die Anwendungstopologie hat eine eine Y- oder eine X-Form ($m : n$-Form), da bei letzterer die Empfängerseite die dispergierten Daten mehrerer Clients entgegennimmt und zusammenführt.
3. Stromspeicherdienste. Sie vereinen die Funktionen von dateibasierten Speicherdiensten mit strombasierten Netzwerkproxys und weisen damit eine Doppelform X/Y auf. Dadurch können Daten sowohl permanent gespeichert als auch in Echtzeit für Streaming- oder Publish-/Subscribe-Funktionen an mehrere Anwendungsinstanzen (Clients) bereitgestellt werden.
4. Datenbanken. Die Verarbeitung statischer Daten mit dynamischen Anfragen wird in verteilten Datenbanken meist über eine Replikation oder eine Partitionierung gelöst. Die Aufteilung der Daten ist abhängig vom Datenformat [BM14b]. Ein heterogener Verbund mehrerer unabhängiger Datenbank- und reiner Speicherdienste kann zur Erhöhung der Sicherheit genutzt werden. Für den Datenabruf stehen in dem Fall mehr Dienste als für die Verarbeitung zur Verfügung. Die zu speichernden und zu verarbeitenden Daten haben die Form einzelner Werte oder Tupel. Die Topologie des Datenbankdienstes wird als Y* bezeichet, um auszudrücken, dass Verarbeitungsroutinen als Code auf Ausführungsdienste installiert werden können.
5. Ereignisverarbeitung. Die Verarbeitung dynamischer Daten mit statischen Anfragen erfolgt nach ähnlichem Muster und mit ähnlicher Datenklassifikation wie in Datenbanken, allerdings in Kombination mit Netzwerkproxys. Hieraus geht die Topologie X* hervor.

Die fünf RSM-Plattformdienstklassen sind in Abbildung 4.5 schematisch mit der Einordnung in die Ressourcenklassen und mit ihren wechselseitigen Abhängigkeiten dargestellt.

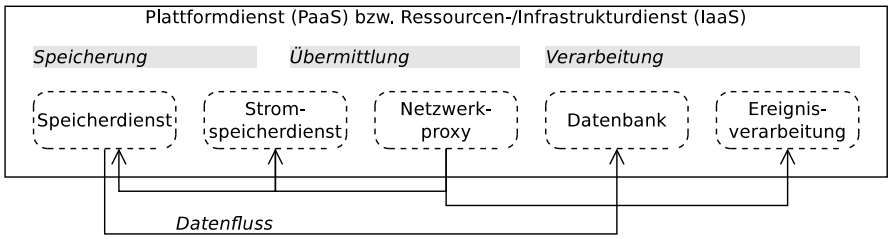

Abb. 4.5 Ausgewählte Plattformdienstklassen als Zielkonfiguration für RSM

Auf die fünf RSM-Plattformdienstklassen teilen sich fünf Datenflusstopologien auf, die durch die charakteristische Aufteilung per Dispersion und die erneute Zusammenführung entweder am Aufteilungsort oder in einer anderen Anwendung gekennzeichnet sind [SS14b]. Sie werden in Abbildung 4.6 grafisch miteinander verglichen.

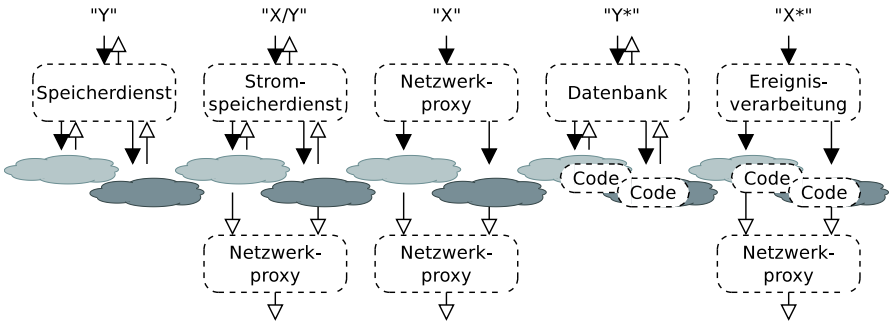

Abb. 4.6 Datenflusstopologien der RSM-Plattformdienstklassen

4.1.3 Ergänzende Absicherung der Nutzung von Ressourcendiensten

Im weiteren Text werden ausschließlich Schutzkonzepte während der eigentlichen Datenverarbeitung betrachtet. Ergänzend soll in diesem Abschnitt kurz dargelegt werden, wie eine ganzheitliche Absicherung von Cloud-Anwendungen hinsichtlich aller Schutzziele über Stealth-Verfahren hinaus erzielen lässt.

Die programmatische Absicherung über die sichere Verarbeitung der Daten hinaus umfasst beispielsweise Overlay-Clouds, um sich von Änderungen einzelner Cloud-Anbietern weitgehend unabhängig zu machen [SCB+14, SBBS12b, SCB+13], sowie der Einsatz clientseitiger Dienstaufrufproxys zur Erhöhung der Fehlertoleranz [SUSS13]. Weiterhin existieren eher nebenläufige Verfahren, also *out-of-band-*

Verfahren zusätzlich zur eigentlichen Datenverarbeitung, zur sporadischen stichprobenartigen Überprüfung von sowohl der kontinuierlichen Existenz als auch der Integrität ausgelagerter Daten. Diese Verfahren umfassen eine belegbare Datenbestands- und übermittlungszusage, welche als *proof-of-retrieval* (PoR) oder *proof-of-possession* (PoP) bezeichnet wird, sowie Inferenzkontrolle, *remote attestation* und *data integrity protection* (DIP) [CL14, KML14, SD13, KBS14].

Sie setzen voraus, dass der Client zumindest eine gewisse Informationsmenge über die durch Dienste gespeicherten Daten beibehält. Über hierarchische Prüfsummenbäume oder Bäume kryptografisch gesicherter Hash-Werte (Merkle-Bäume) lässt sich dies platzsparend realisieren. Neuere Forschungsarbeiten beinhalten schiefsymmetrische Merkle-Bäume, um inhomogene Datenzugriffsbereiche abzusichern, was neben der sicheren Datenspeicherung in Diensten auch auf Speicherhardware anwendbar ist [SB14]. Diese Verfahren sollten durch Stealth-Anwendungen in zufälligen Zeitintervallen angewandt werden, um mögliche Sicherheitsrisiken zur Laufzeit auszuschließen. Komplementär können diese Prüfungen anwendungsübergreifend durchgeführt und ihre Ergebnisse über Verzeichnisdienst-Aktualisierungen allen Anwendungen bereitgestellt werden.

4.2 Netzwerkmultiplexerschnittstelle

Die Aufgabe eines Netzwerkmultiplexers ist es im allgemeinen Fall, empfangene Daten von einem oder mehreren Sendern auf n unterschiedliche Empfänger zu verteilen. Dafür bietet sich eine Proxy-Architektur an. Mit dieser Architektur können insbesondere im Revers-Modus mehrere Datenströme auf $n = 1$ Empfänger zusammengeführt werden [SS14b]. Die übertragenen Daten können die Form von einzelnen Nachrichten wie auch, sofern vom zugrundeliegenden Übertragungsprotokoll unterstützt, von zusammenhängenden Strömen haben. Als Protokoll bieten sich somit TCP oder SCTP, mit Einschränkungen auch UDP-Sequenzen, sowie HTTP, WebSocket und XMPP an. Das Datenformat ist zudem anwendungsabhängig zu wählen. Für einzelne Nachrichten sind neben Binär- und Textkodierungen auch strukturierte Nachrichten mit JSON oder XML geeignet.

Abbildung 4.7 veranschaulicht die Integration mehrerer Übertragungswege. Sie erfolgt für die Anwendung transparent durch eine vorgeschaltete Netzwerkschnittstelle, welche gegebenenfalls mit Adaptern für unterschiedliche Protokolle ausgestattet und als Proxydienst voreingestellt ist, oder durch eine Überladung der Netzwerkfunktionen von Systemaufrufen und Netzwerkbibliotheken.

Eine Nachrichtenwarteschlange (*message queue*) ist eine besondere Ausprägung eines Netzwerkproxys. Das Protokoll Streaming Text-Oriented Messaging Protocol (STOMP) legt zwar keine Größenbeschränkungen für Datenübertragungen fest, sieht diese aber für Implementierungen vor und erfordert insbesondere die explizite Angabe der Größe im Kopffeld einer Nachricht, so dass STOMP trotz des Namens nicht für die Übertragung längerer Datenströme geeignet ist. Das Advanced Message Queuing Protocol (AMQP) sieht in ähnlicher Form Nachrichtenübertragungsrah-

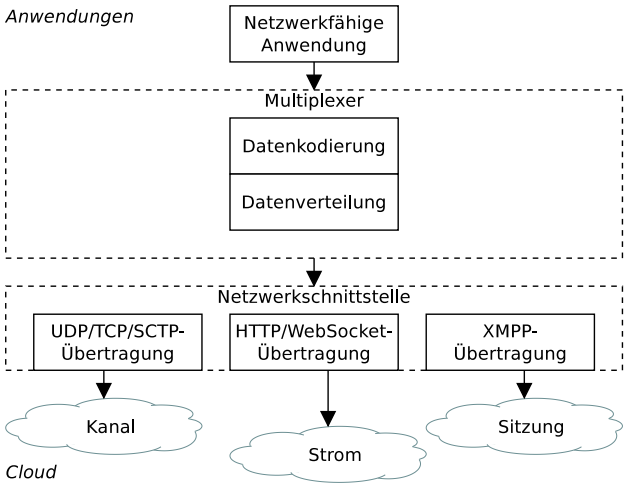

Abb. 4.7 Abstrakte Architektur einer Netzwerkmultiplexerintegration

men mit einer explizit anzugebenden Größe vor. Somit bleibt festzustellen, dass die Dienstklasse Nachrichtenstromwarteschlange (*stream queue*) nicht weit verbreitet ist und sich die Stealth-Untersuchungen auf Warteschlangen mit diskreten Nachrichten beschränken können.

4.3 Dateispeicherschnittstelle

Die Aufgabe eines Dateispeicherdienstes ist die Verwaltung von Dateien und Verzeichnissen über das Netzwerk. Neben den vier grundlegenden Dateioperationen Erzeugung, Veränderung, Löschung und Abruf sind zumeist weitere Operationen wie Suche oder Berechtigungsverwaltung funktionaler Bestandteil solcher Dienste. Aufbauend auf den rudimentären Speicherdiensten werden erweiterte Funktionsmerkmale wie Speicherplatzerweiterung, Datensicherung einzelner Geräte (Backup), selektive Bereitstellung von Dateien für den Lese- oder Schreibzugriff durch weitere Personen (Sharing) und Abgleichung von Dateien zwischen Geräten eines Benutzers (Synchronisierung) realisiert. Die Schnittstelle zu den Diensten ist mitunter als anbieterspezifische Anwendung realisiert, meist in Form von webbasierten Online-Laufwerken oder mobilen Anwendungen, vorteilhaft ist jedoch ein normierter Dienstzugriff über ein wohldefiniertes und standardisiertes Protokoll. Für den Online-Bereich jenseits lokaler Netze existieren mehrere solcher Protokolle, beispielsweise die HTTP-Erweiterungen für verteiltes Erstellen und Versionieren von Dokumenten im Web (WebDAV) [Dus07], welches in der Diskussion der Umsetzung vorrangig betrachtet wird.

Die Nutzung eines einzelnen Dateispeicherdienstes verlagert mehrere Sicherheitsabwägungen auf diesen. Während sich einige Anbieter beispielsweise mit

Mehrfachreplikation gegen Hardwareschäden absichern, verlangt der weiter gefasste Schutz gegen die temporäre und permanente Nichtverfügbarkeit eines Anbieters eine Kontrollmöglichkeit durch anbieterunabhängige Verteilung.

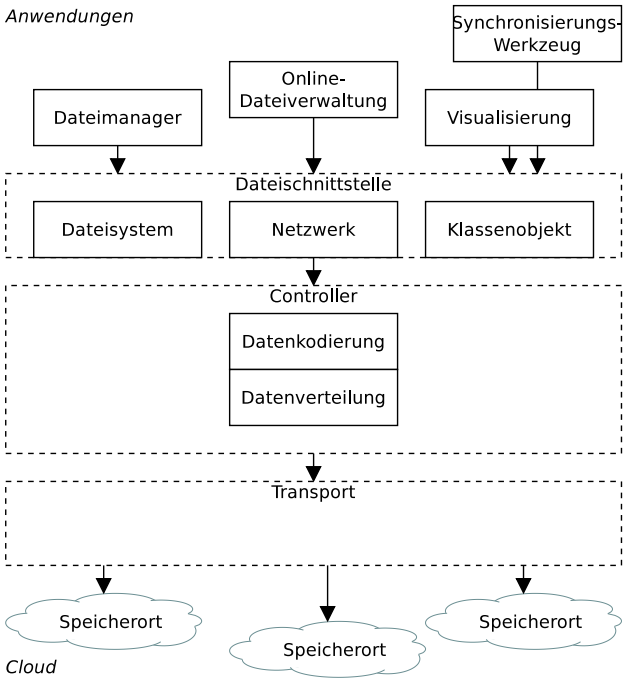

Abb. 4.8 Abstrakte Architektur einer Cloudspeicherdienstintegration

Die Umsetzung einer nutzerkontrollierten, sicheren und verteilten Online-Speicherung lässt sich vorteilhaft mit einer Dreischichtenarchitektur vornehmen. Die Schichten entsprechen der Schnittstelle für die Ein- und Ausgabe von Dateien seitens der Anwendung oder des Benutzers, der zentralen Dateikodierung sowie der Transportschnittstelle zu den Speicherdiensten, welche sich außerhalb der Vertrauens- und Kontrolldomäne der Anwendung befinden. Die resultierende abstrakte Architektur wird in Abbildung 4.8 gezeigt. Ihre Abbildung auf eine konkrete Architektur einer Speicherdienstintegration kann auf vielfältige Art und Weise erfolgen, etwa als virtuelles Dateisystem zur größtmöglichen Kompatibilität mit existierenden Anwendungen, als objektorientierter Inhaltsanbieter auf mobilen Plattformen, oder als Netzwerkproxy für den zentral administrierten Einsatz in Mehrbenutzerumgebungen.

Parallel zu dem dargestellten Datenspeicherfluss durch die Architektur existiert ein Kontrollfluss, welcher eine Konfiguration in das System gibt und über die Laufzeit Metadaten und Cache-Daten als Ausgabe erhält. Die Konfiguration enthält dabei Angaben zur Datenkodierung und Datenverteilung, zur Dienstauswahl sowie zu

weiteren Entscheidungen entlang des Speicherflusses. Die Metadaten enthalten unter anderem Angaben zur verwendeten Kodierung und den Speicherorten der Fragmente einer jeden Datei, um diese wiederherstellen zu können. Auch kryptografische Schlüssel, die bei der Absicherung der Fragmente zum Einsatz gekommen sind, sind Teil der Metadaten.

Die Auswahl von Zieldiensten für eine Konfiguration wird notwendigerweise Teil der Architektur, wenn das Ziel die dynamische Rekonfiguration im Fall einer sich verändernden Menge an Speicherdienstanbietern umfasst. Zudem ist es vorteilhaft, neben den zwingend notwendigen Angaben zu jedem Dienst wie Endpunkt und Zugangsdaten weitere Attribute wie Kapazität, Durchsatz und Preis abfragen zu können, um die Einbindung und Ansteuerung der Dienste zu optimieren. Auf diese Anforderungen wird gesondert in Abschnitt 4.7 eingegangen.

Entlang des Datenspeicherflusses sind mehrere Entscheidungen in Abhängigkeit von den Daten, den gewählten Speicherdienstanbietern sowie den Nutzeranforderungen zu treffen. Die Mehrzahl der Entscheidungen trägt dazu bei, Übertragungs- oder dienstseitige Verarbeitungsfehler zu reduzieren oder die Übertragung und den Abruf der Daten zu beschleunigen. Diesen Vorteilen stehen jedoch zumeist Aufwände entgegen, deren Umfang abgewogen und eingeplant werden muss. Tabelle 4.1 fasst einige grundlegende Entscheidungen zur Kodierung und Benennung von Dateien und daraus abgeleiteten Fragmenten zusammen und stellt die resultierenden Vorteile den jeweiligen Aufwänden gegenüber.

Tabelle 4.1 Kodierung und Benennung von Dateien und Fragmenten

Entscheidung	Auswirkung	Aufwand
Zufallsnamen/Anonymisierung	Vermeidung von Rückschlüssen auf Dateiinhalte oder Ordnerstrukturen; Umgehung von Einschränkungen hinsichtlich Dateinamen und -typen	mehr Metadaten
Partitionierung/Chunking	Umgehung von Upload-Größenbeschränkungen; geringerer Speicherbedarf des Controllers	mehr Metadaten
Zusammengefasste Schreibzugriffe/Batching	Verminderte Übertragungszeiten; Umgehung von Einschränkungen hinsichtlich Dateinamen durch Archive	höherer Platzbedarf
Lokaler Zwischenspeicher/Caching	niedrigere Latenzen, Schutz gegen Kommunikationsmusteranalyse	Cache-Verwaltung, höherer Platzbedarf
Datenströme/Streaming	Echtzeitzugriff auf Daten im Schreibvorgang	Delta-Updates
Parallelisierung	beschleunigte Datenkodierung und -transfers	Thread- oder Prozesskoordinierung

Orthogonal zur Konfiguration stehen Richtlinien, nach denen sich der Datenspeicherfluss richten muss. Um Inkonsistenzen zwischen der Konfiguration und den Richtlinien zu vermeiden, existieren Verfahren, um die Konfiguration teilweise aus den Richtlinien abzuleiten und dabei den jeweiligen Ausführungskontext zu berücksichtigen [SS12a], was zu der Anforderung der dynamischen Rekonfigurierbarkeit des Speicherdienstmultiplexers führt.

Zur Erhöhung der Ausfallsicherheit der Speicherdienstintegration ist es notwendig, die entstehenden Metadaten hinreichend zu sichern. Hierbei ist ein mögliches Verfahren, diese analog zu den Daten verteilt zu speichern. Zwar fallen hierbei wieder Metadaten an, jedoch sind diese in ihrer Größe konstant und weitaus einfacher zu handhaben.

Desweiteren kann es notwendig sein, eine portable maschinenlesbare Zusammenstellung der Fragmentspeicherorte im Fall einer Dateifreigabe in normierter Form aus den Metadaten zu bilden. Hierfür sind dedizierte Beschreibungsformate und URL-Bezeichner geeignet, welche im Abschnitt 4.7 erörtert werden.

4.4 Datenbankschnittstelle

Die Aufgabe eines Datenbankdienstes ist die administrationsfreie Verwaltung strukturierter Daten. In diesem Sinne ist ein Datenbankdienst als erweiterer Speicherdienst zu betrachten. Zusätzlich zur Speicherung feingranularer Daten sind komplexe datentypgebundene Abfragen möglich.

Als Abfragesprache wird aus Darstellungsgründen σ-SQL definiert. Diese stellt eine erweiterte Untermenge der *Structured Query Language* (SQL) respektive eine Variante abgeleiteter Sprachen für nicht vollständig oder nicht rein relationale Daten wie die *Google Query Language* (GQL) [ZSLB14] und die *Continuous Query Language* (CQL) oder weiterer *Event Query Languages* (EQL) [EBB+11] um RSM-spezifische und stealth-spezifische Syntaxelemente dar. Alternativ könnte die Anwendung die Auswahl und Konfiguration der Cloud-Dienste mit einer eigenen Syntax oder spezifischen Befehlen durchführen. Im Sinne der Portabilität der Datennutzungskonfiguration bietet sich jedoch eine Einbindung in einer SQL-artigen Syntax an.

Die Sprache zeichnet sich dadurch aus, dass Abfragen hinsichtlich priorisierter nichtfunktionaler Eigenschaften mit Einfluss auf deren Ausführung markieren lassen. Dies erlaubt beispielsweise, die Ausführungsgeschwindigkeit einer Abfrage gegenüber der Präzision der Antwortdaten abzuwägen (*fuzzy query* [MPP15]) oder die Geschwindigkeit gegenüber der Energieeffizienz zurückzustellen (*sweetspot query* [GIC+14b, GIC+14a]). Hierfür wird das Syntaxelement `SELECT OPTIMIZE FOR <parameter-list>` in Analogie zur Inhaltsaushandlung mit Parameterlisten im Kopfteil einer HTTP-Nachricht (*content negotiation* [FR14]) vorgeschlagen.

Ein weiteres Sprachmerkmal ist die Angabe der Datenkodierung und -verteilung. Hierfür wird das Syntaxelement `USE CLOUDS <distribution-list> WITH`

<encoding-list> vorgeschlagen. Dieses kann entweder ohne weitere Qualifizierung global für die aktuelle Sitzung hinsichtlich neu erzeugter Tabellen gelten, oder angewandt auf existierende Tabellen oder einzelne Tabellenspalten deren Inhalte neu kodieren und umverteilen.

Die Quelltextbeispiele 4.1 und 4.2 stehen exemplarisch für die Erweiterungen von σ-SQL.

Listing 4.1 σ-SQL-Syntaxbeispiel zur Auswahl und Nutzung von Speicherbereichen

```
USE CLOUDS 'mem://primary' AND 'mem://secondary' AND 'file:///
    home/user/.db' AND 'cloud://gcp' AND 'cloud://ec2' WITH '
    dispersion,encryption';
ALTER TABLE table ALTER COLUMN id USE CLOUDS ...;
```

Listing 4.2 σ-SQL-Syntaxbeispiel zur Abfrage mit Garantiezielen

```
SELECT id FROM table OPTIMIZE FOR 'reliability,performance';
```

Abbildung 4.9 veranschaulicht den Entwurf einer Stealth-Datenbankschnittstelle. Neben der dem RSM-Prinzip folgenden Vereinheitlichung von Zugriffen auf den Arbeitsspeicher, auf Dateien und auf generische Speicher- und Verarbeitungsdienste sieht die Schnittstelle eine adaptive Programmausführung aufgeteilt zwischen den lokalen Ressourcen und den angebundenen Verarbeitungsdiensten mit Hilfe von Map-Reduce und Map-Carry-Reduce vor.

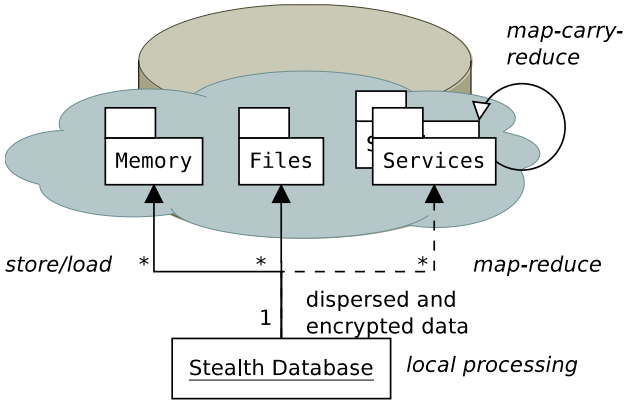

Abb. 4.9 Entwurfsarchitektur einer Stealth-Datenbankschnittstelle

4.5 Stromspeicherdienstschnittstelle

Sollen Daten skalierbar zwischen produzierenden und konsumierenden Systembestandteilen ausgetauscht werden, so sind Nachrichtenwarteschlangen und -verteiler dafür geeignete Komponenten. Je nach Anwendungsfall stehen unterschiedliche, auf Publish-Subscribe-Verfahren optimierte Protokolle wie XMPP oder PubSubHubbub bereit. Diese sind jedoch primär auf die Kommunikation und weniger auf die Archivierung der übermittelten Daten oder den wahlfreien Zugriff auf selbige ausgelegt. Alternativ bietet es sich an, Speicherdienste auf ihre Nutzbarkeit für den kategorisierten Datenaustausch mit mehreren Empfängern in Echtzeit hin zu untersuchen. Dies erfordert eine erweiterte Sicht auf die Dienste. Dafür soll nun der Begriff Datenstromspeicherdienste stehen, welcher grundlegende Datenverwaltungsmethoden mit solchen zur Stromübermittlung und zwecks kollaborativen oder öffentlichen Zugriffs zudem mit solchen zur Freigabe einzelner Datenströme kombiniert.

Als offenes und weit verbreitetes Protokoll sei hierfür primär WebDAV Gegenstand der Betrachtungen. Um WebDAV-Dienste als Grundlage für eine Publish-Subscribe-Architektur für Datenströme nutzen zu können, müssen diese Datenströme an sich und somit als Grundlage dafür Delta-Aktualisierungen unterstützen. Diese werden für HTTP auch als Byte-Serving bzw. Bereichsanfragen bezeichnet. Vergleichend werden dazu reine HTTP-Dienste ohne WebDAV-Erweiterungen sowie als neueres Protokoll GCS-JSON und GCS-XML (abgeleitet von Google Cloud Storage [MSMF14, Ali15]) betrachtet.

Die Methoden und Operationen der Speicherdienstprotokolle HTTP, WebDAV und Google Cloud Storage (GCS) werden in Tabelle 4.2 miteinander verglichen. Dabei sind für Datenströme die partiellen Aktualisierungen von Interesse, bei denen nur Deltas des Datenstroms übertragen und dienstseitig angehangen werden. Ein Asterisk (*) gibt an, dass diese Methode datenübertragungsintensiv arbeitet, während die Methoden ohne Asterisk zumeist auf Identifikatoren arbeiten.

Tabelle 4.2 Vergleich von Speicherdienst-Operationen

Funktionalität	HTTP-Methode	WebDAV-Methode	GCS-Methode
Anlegen	POST/PUT (*)	(POST)/PUT *	cp
Aktualisieren	PUT *	PUT *	cp
Löschen	DELETE	DELETE	rm
Abrufen	GET *	GET *	cp
partielles Aktualisieren	PUT mit Content-Range*	PUT mit Content-Range*	-
Kopieren	-	COPY	cp
Umbenennen	-	MOVE	mv
Konkatenieren	-	-	concat
Freigabe	-	(PROPPATCH)	acl
Ordner erzeugen	-	MKCOL	(implizit)

4.6 Ereignisverarbeitungsschnittstelle

Die Kombination einer Schnittstelle zur Übermittlung von Datenströmen und einer weiteren Schnittstelle zur Registrierung von Abfragen in σ-SQL und dem asynchronen Empfang von Ergebnissen führt zur Dienstklasse *dispersed event processing* respektive *stealth event processing* angelehnt an ESP/CEP-Dienste.

Abbildung 4.10 zeigt die grundlegende Architektur der Ereignisverarbeitungsschnittstelle.

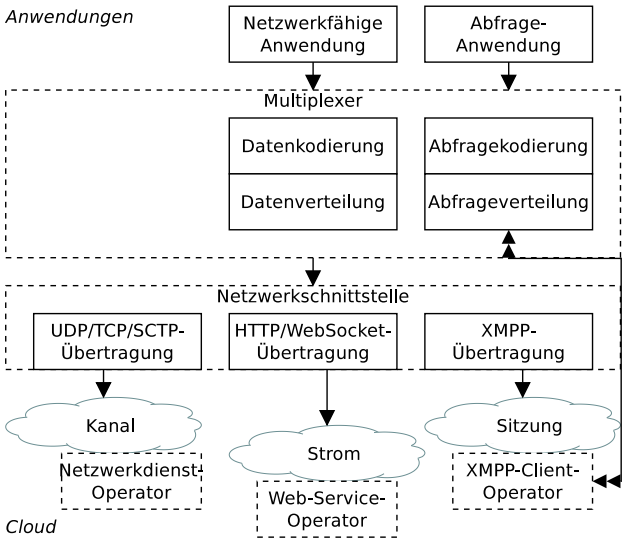

Abb. 4.10 Abstrakte Architektur einer Ereignisverarbeitungsintegration

4.7 Dienstintegration

Die Verwaltung von Diensten und Daten in dienstorientierten Architekturen und Umgebungen setzt üblicherweise eine maschinell verarbeitbare Beschreibung der Dienste und Daten sowie Protokolle zum Publizieren und Suchen dieser Beschreibungen voraus [SS13a, ACKM04, PTDL08]. Desweiteren werden Mechanismen benötigt, um technische Zusicherungen der Dienstgüte ausdrücken zu können. Hierfür werden oft Dienstgütevereinbarungen (*service level agreements*, SLA) genutzt, welche von den Dienstbeschreibungen und der Dienstausführung entkoppelt erstellt und somit wenig repräsentativ für die tatsächlich verfügbare Dienstgüte sind [SIS13]. Zusätzlich sind die existierenden Ansätze mehrheitlich auf einzelne Dienste oder aufrufbare Dienstkompositionen beschränkt. Hingegen gilt für eine clientseitige Bündelung von Diensten, dass die erzielbare Dienstgüte aufgrund von Red-

undanzen und anderen Effekten wesentlich höher als die Summe der Einzelverfügbarkeiten sein kann. In diesem Abschnitt werden deshalb Techniken für eine auf sichere Cloud-Anwendungen passende Dienstverwaltung und Dienstintegration erarbeitet und vorgestellt. Diese umfassen Dienstbeschreibungen, Datenbeschreibungen, Dienstendpunktkonfigurationen sowie Dienstgütekonversationen.

4.7.1 Dienstbeschreibungen und deren Auswertung

Die automatisierte Integration datenverarbeitender Dienste erfordert eine Beschreibung der Leistungsmerkmale. Dadurch wird es möglich, Dienste nach ihrer Funktionalität auszuwählen und nach ihren nichtfunktionalen Eigenschaften zu priorisieren. Für die Beschreibung von Diensten ist eine Vielzahl von Ansätzen verfügbar und bekannt. Neben reinen Schnittstellenbeschreibungssprachen wie der Web Service Description Language (WSDL) oder der Web Application Description Language (WADL) existieren Sprachen und Dialekte, welche nichtfunktionale Eigenschaften, das Dienstverhalten oder Vor- und Nachbedingungen des Aufrufs mit abdecken können. Beispiele hierfür sind WSMO4IoS aufbauend auf der Web Services Modelling Language (WSMO) [SS13a], die Unified Service Description Language (USDL) und Linked-USDL [SKS12, PCL14, Obe14], sowie domänenspezifische Sprachen für Cloud-Dienste [SK14].

Beispielhaft sei im Quelltextbeispiel 4.3 ein Ausschnitt aus einer Beschreibung eines Cloud-Compute-Dienstes gezeigt. Die Beschreibung entspricht einer Instanzontologie von WSMO4IoS.

Listing 4.3 Cloud-Ontologie

```
  /* ------------------------ */
  /* Boilerplate and profile  */
  /* ------------------------ */

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { ...
cloud _"http://localhost:8080/Matchmaker/ontologies/
    CloudComputation.wsml#", ...}
webService CloudComputeDienst
 importsOntology { _"http://localhost:8080/Matchmaker/ontologies/
     CloudComputation.wsml#" }
capability ServiceCapability
 postcondition definedBy ?serviceType memberOf cloud#
     CloudComputation .
interface CloudComputDienstSmallInterface
 importsOntology { CloudComputeDienstSmallOntology }
ontology CloudComputeDienstOntology
 importsOntology { _"http://localhost:8080/Matchmaker/ontologies/
     CloudComputation.wsml#" }
  instance CCCPU memberOf qos#FrequencyUnit
    qos#conversionFactor hasValue 3355443200.0
ontology CloudComputeDienstSmallOntology
```

```
importsOntology { _"http://localhost:8080/Matchmaker/ontologies/
    CloudComputation.wsml#",
                  CloudComputeDienstOntology }

 /* ---------------------- */
 /* Computation properties */
 /* ---------------------- */

 instance SmallDisk memberOf { res#Disk, qos#ServiceSpec }
   qos#value hasValue 2
   qos#unit hasValue qos#GB

 instance SmallMemory memberOf { res#Memory, qos#ServiceSpec }
   qos#value hasValue 128
   qos#unit hasValue qos#MB

 instance SmallProcessors memberOf { res#Processor, qos#
     ServiceSpec }
   qos#value hasValue 1
   qos#unit hasValue CCCPU

 instance SmallArchitecture memberOf { res#Architecture, qos#
     ServiceSpec }
   qos#value hasValue 32
   qos#unit hasValue qos#Bit

 /* ------------------------ */
 /* Main instance definition */
 /* ------------------------ */

 instance Small memberOf { cloud#CloudComputation }
   hasName hasValue "CloudComputeDienst small"
   hasIcon hasValue "???"
   hasWebsite hasValue "http://ccdienst/"
   hasCountry hasValue "Germany"
   hasComputePlan hasValue { SmallPlan }

 instance SmallPlan memberOf { cloud#CloudComputePlan }
   hasMemory hasValue SmallMemory
   hasProcessors hasValue SmallProcessors
   hasDisk hasValue SmallDisk
   hasArchitecture hasValue SmallArchitecture
   hasBursts hasValue _boolean("false")
```

4.7.2 Datenbeschreibungen und deren Auswertung

Im Abschnitt 3.3 wurden Relationenschemata zur Repräsentation der Orte und wechselseitigen Beziehungen von Daten mitsamt Regelsätzen und Algorithmen

eingeführt. In diesem Abschnitt werden nun konkrete Repräsentationsformen und Werkzeuge für die Nutzung der Schemata in Cloud-Umgebungen vorgestellt.

4.7.2.1 Datenbeschreibung mit Ressourcenbezeichnern

Die bereits erläuterte Minimierung des FRS-Graphen ist speziell für eine kompakte textuelle Notation wichtig. Insbesondere gilt dies für den interoperablen Zugriff durch Anwendungen, wenn der Graph in Form einer URL ausgedrückt werden soll. Hierfür wird ein neues speicherortsübergreifendes URL-Schema benötigt, welches bisher in der Form noch nicht existiert. URL-Schemata sind global für das Internet definierte syntaktische Regeln zum Aufbau von Ressourcenbezeichnern (*uniform resource locator*, URL). Für den Datenzugriff sind bereits mehrere URL-Schemata definiert, unter ihnen `file://` und `http`, wobei der eigentliche Datenabruf über komplementär definierte Protokolle abläuft, so beispielsweise per GET gemäß HTTP (RFC 7230) [FR14] für den lesenden Zugriff auf *http*-URLs. Das Meta-Schema für URLs und verwandte Konstrukte wie URIs und IRIs (*international resource identifier*) gemäß RFC 4395 und 3986 [HHM06] sieht eine singuläre Netzadresse als Autorität vor.

Das vorgeschlagene neue URL-Schema mit dem Namen `fileaccess://` verweist demgegenüber nicht nur auf einen Speicherort, sondern auf beliebig viele, welche wiederum als URL-kodierte Zeichenketten gemeinsam mit einer Kodierung der Relationen Teil einer `fileaccess`-URL sind. Die empfohlene Länge für URLs sollte 255 Zeichen nicht überschreiten. Aufgrund der multiplikativen Natur von `fileaccess` wird hierfür der empfohlene Grenzwert $n * 255$ Zeichen angesetzt.

Die Spezifikation des URL-Schemas sieht die im Quelltext 4.4 angegebene Syntax vor.

Listing 4.4 URL-Schema für die Darstellung von Fragmentrelationen

```
fileaccess://e0=qp(url)[&e1=qp(url)...]?R=eX,eY[&R=eX,eY...]}
```

Hierbei ist *e*0...*eN* ein fortlaufend nummerierter Bezeichner und *R* eine Darstellung der Relation. R nimmt die Werte *r* für Replikate, *c* für Komplemente, *b* für redundante Komplemente sowie *p* für partielle Replikate an. Die Funktion $qp()$ drückt die Prozentkodierung der URL aus, damit sie syntaktisch als Teil einer anderen URL zulässig wird.

Ein Beispiel zur URL-Kodierung von Fragmentdaten wird in Tabelle 4.3 gegeben. Hierbei sei eine Datei dreimal, nämlich als Original mit Replika im Web sowie einer weiteren mit 50% Redundanz dispergierten (fragmentierten) lokalen Replika gegeben.

Tabelle 4.3 Beispiellokatoren zu Datenfragmenten

URL	Bezeichner	Semantik
`http://provider1/api/x.pdf`	e0	Datei, 100% Information
`http://provider2/dav/x.pdf`	e1	Datei, 100% Information
`file://tmp/x_partA.pdf`	e2	Fragment, 50% Information
`file://tmp/x_partB.pdf`	e3	Fragment, 50% Information
`file://tmp/x_partR.pdf`	e4	Fragment, 50% Redundanz

Die Darstellung der Relationen erfolgt im Quelltext 4.5. Diese sind bereits minimal, so dass keine Relation ausgelassen werden kann, ohne einen Informationsverlust zu erleiden, und keine Relation hinzugefügt werden kann, welche einen Informationsgewinn brächte.

Listing 4.5 Fragmentrelationen im Beispiel

```
ep0 ≐ ep1; replica
ep2 ∞ ep3; complement
ep2 ⊂ ep0; partial replica
ep2 ∝ ep4; redundant complement
```

Die resultierende vom minimalen FRS-Graphen abgeleitete URL ist im Quelltext 4.6 abgebildet. In der Praxis sind derartig lange URLs für die maschinelle Verarbeitung und den teilautomatisierten Austausch etwa als QR-Tag unproblematisch, für den manuellen Austausch zwischen Benutzern etwa als Kurznachricht oder per Ansage jedoch ungeeignet. Hierfür lassen sich allerdings geeignete Dienste zur zeitweisen Abbildung auf kürzere URLs nutzen.

Listing 4.6 URL-Darstellung der Fragmentrelationen im Beispiel

```
fileaccess://e4=file%3A%2F%2Ftmp%2Fx_partR.pdf&e1=http%3A%2F%2
    Fprovider2%2Fdav%2Fx.pdf&e0=http%3A%2F%2Fprovider1%2Fapi%2Fx.
    pdf&e3=file%3A%2F%2Ftmp%2Fx_partB.pdf&e2=file%3A%2F%2Ftmp%2
    Fx_partA.pdf?r=e0,e1&c=e2,e3&p=e2,e0&b=e2,e4
```

4.7.2.2 Auswertung von Ressourcenbezeichnern

Abbildung 4.11 demonstriert drei alternative Nutzungsoptionen für *fileaccess*-URLs. Der erste Weg sieht ein lokales Programm oder einen Netzwerkproxy vor, an welches eine URL übergeben wird. Das Programm zerlegt und interpretiert die URL,

lädt iterativ die benötigten Dateien oder Fragmente, und setzt sie anschließend zu einer vollständigen Datei zusammen, die anschließend an eine Anwendung übergeben wird. Der zweite Weg beginnt ähnlich mit der Zerlegung und Interpretation der URL, delegiert das Laden allerdings an einen existierenden Speicherdienst-Controller. Hierfür werden in dessen Datenbasis zu den Fragmentorten gespeicherter Dateien neue Einträge hinzugefügt, so dass anschließend die durch die URL repräsentierte Datei über die Schnittstelle des Speicher-Controllers, beispielsweise ein virtuelles Dateisystem, abgerufen werden kann. Der dritte Weg ähnelt den ersten beiden, führt jedoch keine Kopie zum lokalen System, sondern eine Migration zu einer geänderten Speicherzielkonfiguration durch. Das lokale Programm erhält dazu noch eine zweite URL und nutzt diese für den eigenständigen erneuten Transfer der Dateien oder die indirekte Instruktion des Speicherdienst-Controllers anhand einer geänderten Speicherzielkonfiguration.

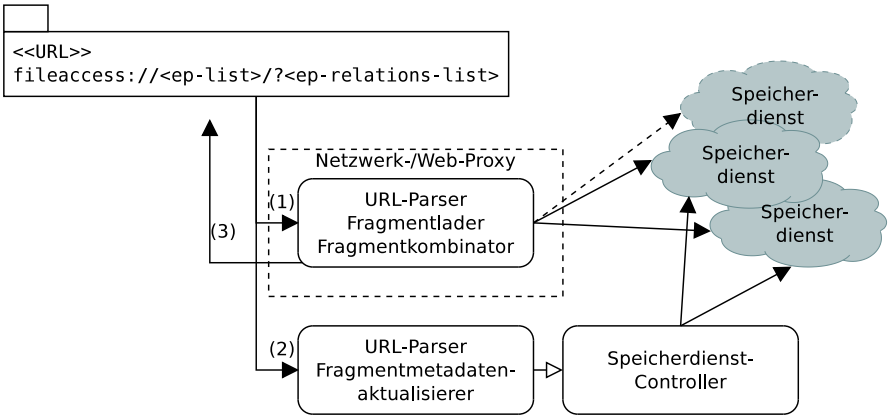

Abb. 4.11 Nutzungsvarianten zu einer FRS-Instanz-URL in einer dateiverwaltenden Netzwerkarchitektur

Für den selektiven Zugriff auf Dateien über mehrere Speicherdienste wird ein Softwarewerkzeug *Cloud-to-Cloud Copy* (`c2ccp`) vorgeschlagen, welches einerseits Daten auf das lokale System herunterladen, andererseits aber auch Daten einschließlich Crossloading von einer Menge an Speicherdiensten auf eine andere migrieren kann. Ähnliche Ansätze gibt es bereits mit dem Werkzeug `gsutil` zugehörig zu Google Cloud Storage, wobei dieses nur einen Speicherdienstanbieter unterstützt.

Für die Nutzung von Ressourcenbezeichnern in Datenbankverwaltungssystemen wird eine Syntaxerweiterung analog zum globalen oder spaltenbasierten Einsatz von *USE CLOUDS* vorgeschlagen. Die Syntax wird im Quelltext 4.7 erläutert. Es wird davon ausgegangen, dass bei einer derartig feingranularen Rekonfiguration der Nutzer darauf hingewiesen wird, dass die Flexibilität mit möglicherweise sehr hohen Laufzeiten erkauft wird. Desweiteren wird davon ausgegangen, dass eine Ortsanga-

be auf der Tupelebene ähnlich zur Unterscheidung aller Werte über eine eindeutige OID realisiert werden kann.

Listing 4.7 σ-SQL-Syntaxbeispiel zur Migration einzelner Werte

```
UPDATE table USE CLOUDS '...' WHERE x < 5;
```

4.7.3 Modell zur Konfiguration veränderlicher Endpunkte

Aufbauend auf Datenrelationsschemata lässt sich ein solches Schema auch für verteilte Anwendungsdienste mit mehreren Endpunkten konzipieren. Als Endpunkte werden innerhalb eines Protokolls gültige und eindeutige Adressen für einen Dienstaufruf oder Datenzugriff bezeichnet. Dienste können dabei mehrere Endpunkte besitzen, um über unterschiedliche Protokolle oder aus unterschiedlichen Kontexten heraus aufrufbar zu sein. Für die Erhöhung der Fehlertoleranz beim Aufruf eines nicht stets verfügbaren Dienstes und für die Erhöhung der Skalierbarkeit existieren bereits Ansätze zur Spezifikation und zum iterativen versuchsweisen Aufruf mehrerer alternativer Endpunkte desselben oder unterschiedlicher Anbieter [SUSS13, CRM06]. Dabei wird meist ein simples Round-Robin-Schema und für die Erzielung der Transparenz gegenüber der Anwendung eine Proxyarchitektur oder die Einbindung in einen ausführbaren Prozess genutzt. Es existiert jedoch bisher kein Ansatz zur deklarativen Beschreibung der Relationen zwischen mehreren Endpunkten einer Menge, die einerseits über die Relation *fallback* hinausreichen, andererseits auch einer zeitlichen Veränderung im Sinne einer globalen Dienst- und Endpunktevolution unterliegen. Zur Versionierung von Endpunkten im Zusammenhang mit einer fortschreitenden Dienstentwicklung sei auf VRESCo verwiesen [LMRD08]; der hier vorgestellte Ansatz berücksichtigt hingegen nur Laufzeitänderungen bei angenommener Stabilität und Kompatibilität der Aufrufschnittstellen an sich.

Im Folgenden wird das Modell *Evolving Endpoints Configuration* (EEC) vorgestellt. Datenverarbeitende Dienste werden als eine gebündelte Menge von physischen Diensten mit Endpunkten $\{E_i\}(1 \leq i \leq n; n \geq 1)$ repräsentiert. Im Sinne der Service-Science-Betrachtung entspricht die Bündelung der Zusammenfassung von Diensten, welche im Gegensatz zur Komposition keine strikte funktionale Abhängigkeit besitzen. Jeder Endpunkt E_i enthält entweder vollständige oder unvollständige Daten. Abhängigkeitsrelationen bestehen zwischen jeweils zwei beliebigen Endpunkten. Sie können als *complement* für Komplementärabhängigkeiten zu partiellen Daten, *parallel* für replizierte Daten etwa für Konsensabstimmungen oder *backup* für Fehlertoleranzsituationen ausgedrückt werden. Insgesamt existiert ein Maximum von $\frac{n(n-1)}{2}$ derartigen Abhängigkeiten. Die Endpunkte sind nicht notwen-

digerweise in Besitz des Wissens über die Abhängigkeiten. Es wird angenommen, dass sie nicht miteinander kooperieren und teilweise auch keine anderen Endpunkte kennen. Komplementärabhängigkeiten werden logisch gruppiert, um Abhängikeiten aufzulösen. Ein Endpunkt, der als Backup für zwei wechselseitig komplementäre Endpunkte fungiert, führt auf der logischen Ebene zu einem Primär- und einem Backup-Endpunkt.

Anwendungen sind hingegen zu jedem Zeitpunkt über die Menge der Endpunkte und ihrer Abhängigkeiten und Relationen informiert. Dies bedeutet, dass nicht nur zur Anwendungsentwicklungszeit, sondern jederzeit während der Ausführung eine Schutzfunktion derart existieren muss, dass die EEC stets sich weiterentwickelnde Infrastrukturen und veränderliche Dienstbedingungen reflektiert. Dadurch wird die Menge der Endpunkte zeitabhängig $E_i(t)$ mit einem wachsenden Unsicherheitsfaktor $\delta_i(t)$ im Fall von Aktualisierungsverzögerungen.

Eine beispielhafte Menge von Endpunkten mit vielfältigen Relationen ist in der Abbildung 4.12 zu sehen.

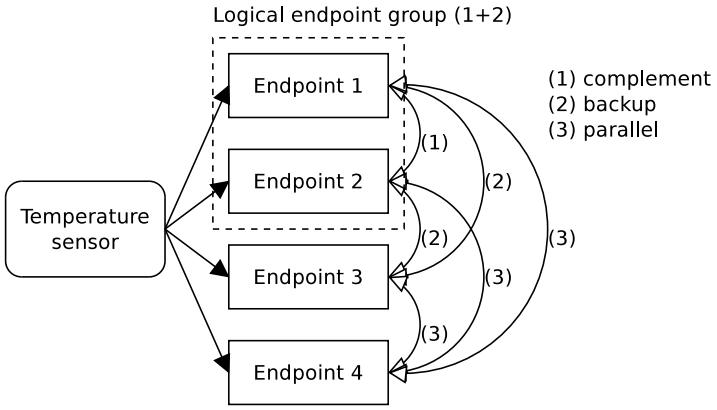

Abb. 4.12 Beispieltopologie von Endpunkten zu einem fiktiven Temperaturdatendienst

Für die Benachrichtigung über veränderte Endpunkte oder veränderte Relationen wird eine adäquate Kommunikationsinfrastruktur benötigt, welche selbst auch wieder hochverfügbar und fehlertolerant ausgelegt sein muss.

4.7.4 Protokoll zur Verständigung über Dienstgütevereinbarungen

Dienstgütevereinbarungen enthalten juristische Zusicherungen zu funktionalen und nichtfunktionalen Eigenschaften von einzelnen Diensten oder Gruppen von Diensten als Dienstbündeln. Die technische Umsetzung der Vereinbarungen ist dann möglich, wenn auch technische Zusicherungen erzielt werden können. Die dynamische Veränderlichkeit der Endpunkte erfordert dabei eine Berücksichtigung neuer

Dienstangebote und veränderter Diensteigenschaften über die Laufzeit. Somit ist es angebracht, nicht nur einmal vor der Nutzung eines Dienstes eine Dienstgüteaushandlung durchzuführen, sondern einen kontinuierlichen Aushandlungsprozess zwischen einer Anwendung und den von ihr genutzten Diensten anzustreben. Hierfür werden nebenläufige *Service Level Conversations* vorgeschlagen, welche zu jedem Zeitpunkt eine Menge von Diensten mit ihren Endpunkten und Relationen gemäß EEC sowie zugehörigen Zusicherungen zum Gegenstand haben. Quelltext 4.8 enthält in Kurzform einen möglichen SLC-Ablauf als Teil einer andauernden Konversation.

Listing 4.8 SLC

```
← request: frequency=10,cost=1,availability=99.9
→ offer: frequency=5,cost=1.2,availability=98.0
← accept
→ eec: fileaccess://...
← agree
→ eec: fileaccess://...
   # veränderte Endpunkte mit gleichen Eigenschaften
← agree
→ offer: frequency=5,cost=2.2,availability=97.0
   # veränderte Eigenschaften
← reject
```

4.8 Entwicklung von Anwendungen

Im Bereich der Softwareentwicklung existieren bereits mehrere Vorgehensweisen zur Entwicklung sicherer und zuverlässiger Cloud-Anwendungen. Rak et al. präsentieren einen komponentenorientierten Ansatz unter Berücksichtigung der Überprüfbarkeit von Sicherheitseigenschaften durch von Nutzern ausgeführte Monitoringfunktionen auf Basis der mOSAIC-Plattform [RFB+14]. Einen weiteren Ansatz liefern Almorsy et al. mit einer modellgetriebenen Einpflegung von Sicherheitseigenschaften in Cloud-Anwendungen [AGI13]. Beide Ansätze nehmen allerdings eine zentral bereitgestellte Anwendung an. Die Konstruktion von Anwendungen über Anbietergrenzen hinweg wird von SeaClouds angestrebt, wobei der Fokus allerdings stärker auf der Verwaltung und der Migrationsunterstützung liegt [BIS+14].

Einen Überblick aus der Perspektive der dienstorientierten Anwendungsentwicklung bieten Taher et al. [TNL+12]. Sie beschreiben die Konzeption und Umsetzung kompositer Anwendungen auf Basis von SaaS-Schnittstellen und weisen auf die Probleme durch deren fehlende Standardisierung hin. Viele noch offene Forschungsfragen werden von den Autoren speziell im Bereich der anbieterübergreifenden Dienstnutzung für die Komposition einer Anwendung gesehen. Die vorlie-

gende Arbeit bietet einen solchen Ansatz, der die Risiken insbesondere der Anbieterabhängigkeit verringert. Cito et al. haben in einer zweiteiligen Studie über Entwicklerinterviews und Umfragen festgestellt, dass die Entwicklung von Cloud-Anwendungen generell Einfluss auf die Entwicklungswerkzeuge und den Entwicklungsprozess, aber auch auf die Architektur der Anwendungen nimmt. Insbesondere sind Entwickler gezwungen, bei direkter Interaktion mit IaaS fehlertolerante Anwendungen zu konstruieren. Die Mehrheit der Entwickler sah zudem die größte Einschränkung in suboptimalen Entwicklungswerkzeugen [CLFG14].

Mit Cloud Types von Burckhardt et al. existiert ein explizit datenbezogener Sprachansatz, der für den Datenaustausch über fehlerhafte Cloud-Dienste konzipiert wurde [BFLW12]. Die Sprache ermöglicht dabei eine schwache Konsistenz über eine temporale Graphenstruktur ähnlich einer Revisionsverwaltung. Sowohl simple als auch komplexe Datentypen (Ganzzahlen, Zeichenketten, Tabellen, Listen) werden unterstützt, wobei Fließkommazahlen in die Festkommadarstellung überführt werden müssen. Die Daten werden jedoch stets vollständig an einen Dienst übergeben. Eine feingranulare Aufteilung ist nicht vorgesehen.

4.9 Plattformäquivalente Cloud-Integration für sichere Dienste und Anwendungen

Nach der Vorstellung von fünf Schnittstellenarchitekturen und der Betrachtung der Integrations- und Entwicklungssicht soll an dieser Stelle eine ganzheitliche Perspektive auf Anwendungs- und Dienstökosysteme gegeben werden, welche mit Hilfe von Stealth-Techniken weitgehend ohne zentrale Plattformen auskommen und stattdessen aufbauend auf dynamisch veränderlichen Ressourcendiensten plattformäquivalente Merkmale sicher nutzen können.

Abbildung 4.13 zeigt in vier Schritten die Nutzung von Stealth-Anwendungen. Im ersten Schritt wird an einen Dienstmarktplatz eine Anfrage zur Bereitstellung von Ressourcendiensten eines bestimmten Typs gestellt. Im zweiten Schritt werden diese Dienste aktiviert und eingebunden. Im dritten Schritt, der parallel zum zweiten stattfinden kann, benachrichtigt der Marktplatz die Anwendung über durchgeführte oder geplante Änderungen im Dienstangebot, woraufhin sich die Anwendung mit oder ohne Datenmigration und Neukodierung an die neue Situation adaptiert. Im vierten Schritt kann ein Entwickler eine solche Anwendung für Endgeräte oder als Mehrwertdienst über den Marktplatz an weitere Anwender bereitstellen. Ist die Anwendung auf externe Daten, etwa von Sensoren oder anderen Datenquellen, angewiesen, so dient der Marktplatz auch dem ebenfalls adaptiven Zugriff auf die Datenquellen.

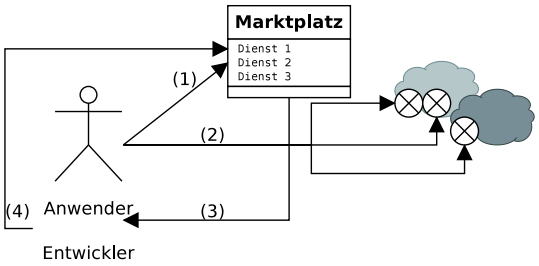

Abb. 4.13 Integrierte Prozesssicht auf die Nutzung von Stealth-Anwendungen

4.10 Zusammenfassung der Beiträge

Im Vergleich mit existierenden Ansätzen zur Bereitstellung zentraler Plattformdienste für Anwendungen in der Cloud lieferte dieses Kapitel alternative Architekturen auf Grundlage des Stealth Computing in Kombination mit Ressourcendiensten. Fünf plattformäquivalente Dienste zur Übertragung, Speicherung und Verarbeitung von Daten sind vorgeschlagen und untersucht worden. Die Architekturen und Schnittstellen haben den Vorteil, sich dynamisch an die Dienstevolution und -fluktuation anzupassen und dem Nutzer ein hohes Maß an Kontrolle über die Datennutzung zu bieten. Die vorgestellte integrierte Perspektive wird im nachfolgenden Kapitel in drei komplexen Szenarien und Anwendungsfeldern ausgebaut.

Kapitel 5
Szenarien und Anwendungsfelder

Zusammenfassung Die in den vorherigen Kapiteln vorgestellten Datenverarbeitungs- und Plattformkonzepte für zuverlässige, sichere, langlebige und native Cloud-Anwendungen sind in vielen realen Szenarien nutzbar und bringen den jeweiligen Nutzern einen Mehrwert gegenüber bisherigen risikobehafteten Cloud-Lösungen. Im Folgenden werden nun drei Szenarien und Anwendungsfelder mit unterschiedlichen Zielgruppen vorgestellt, anhand denen die Wirksamkeit der Konzepte aufgezeigt werden soll. Die Szenarien umfassen einen sicheren verteilten Dateispeicherdienst mit Suchfunktion, eine sichere persönliche Datenanalyse verteilt über öffentliche Berechnungsdienste sowie den abgesicherten Mobilzugriff auf Datenströme im Internet der Dinge. Aus der Breite der Anwendbarkeit lässt sich ableiten, dass Stealth-Architekturen zukünftig in vielen Bereichen der Software- und Serviceentwicklung berücksichtigt werden können.

5.1 Online-Speicherung von Dateien mit Suchfunktion

Die komfortable zentrale Ablage von Dateien zu verschiedenen Zwecken stellt einen wichtigen Funktionsbereich im Internet dar. Man bezeichnet derartige Dateispeicherdienste als Online-Festplatte, Online-Storage oder, sofern cloud-typische Merkmale wie elastische Skalierbarkeit und nutzungsbezogene Abrechnung vorhanden sind, dementsprechend als Cloud-Storage [MSMF14]. Im Gegensatz zu anonymen Peer-to-Peer-Ansätzen, welche eine wechselseitige Verrechnung des Aufwands zulassen, sind Cloud-Speicherdienste zumeist kostenpflichtig, bieten dafür im Gegensatz aber auch die Kontaktierbarkeit des Anbieters und Bestrebungen zu hoher Dienstgüte an. Viele Anbieter stellen zudem kostenlose Einstiegstarife oder Festkontingente speziell für Privatanwender bereit. Hierfür scheint sich die Bezeichnung Personal Cloud Storage durchzusetzen [DMM+12]. Der gleiche Begriff wird allerdings auch dann verwendet, wenn eigene Speicherressourcen beispielsweise auf einem NAS-Gerät (*network attached storage*) genuzt werden. Eine Unterschei-

dung wäre mit den Begriffen Private bzw. Public Personal Cloud Storage möglich, die aber noch keinen Eingang in die Literatur gefunden hat.

Die Zwecke der Online-Ablage von Dateien umfassen die Datensicherung (*backup*), die Erweiterung der Kapazität der eigenen Geräte (*extension*), die Synchronisierung von Dateien zwischen Geräten (*sync*), das Teilen der Dateien und weiterer Daten mit anderen Personen sowohl in eine Richtung (*sharing*) als auch in mehrere Richtungen (*collaboration*) [SBS13] sowie die Nutzung von Dateiverwaltungsmerkmalen, welche auf dem Ausgangssystem nicht angeboten werden (*management*).

Der Zugang zur Online-Speicherung erfolgt dabei client-unabhängig über dedizierte Protokolle, die entweder als standardisiert oder als proprietär eingestuft werden, sowie über vorgegebene Clients vorrangig im Web und auf Mobilgeräten. Die eigentliche Speicherung erfolgt über Software, die entweder anbieter-unabhängig, d.h. frei, verfügbar oder aber als Alleinstellungsmerkmal eines Anbieters auf diesen zugeschnitten ist.

Sowohl die Software als auch die dem Benutzer zur Verfügung gestellten Ressourcen führen zu einer starken Heterogenität im Dienstangebot für Online-Speicher. Um die heterogenen Diensteigenschaften so zu beschreiben, dass sie automatisiert für eine optimale Auswahl anhand von benutzerdefinierten Kriterien berücksichtigt werden können, sind formalisierte Dienstbeschreibungen notwendig. Dazu können entweder generische Dienstbeschreibungsformate oder domänenspezifische Formate wie beispielsweise die Cloud Storage Ontology (CSO) aus dem Ontologienkatalog WSMO4IoS genutzt werden [SS13a]. Deren Instanzen werden in Verzeichnisdiensten, auf Marktplätzen oder Spot-Märkten zum vollständigen oder inkrementellen Abruf bereitgestellt [SBBS12a, WSS14, SS13a, Spi11a]. In der Praxis sind solche Abrufschnittstellen jedoch bisher nicht verbreitet, so dass Anwender oftmals wenig systematisch anhand von Empfehlungen oder Webseiten ihre Dienstwahl treffen, wobei der Informationsabruf von den Webseiten teilweise per *web automation* programmatisch vorgenommen wird [SPW+12]. Die Tabelle 5.1 stellt ausgewählte Kombinationen von Eigenschaften bekannter Speicherdienste dar.

Tabelle 5.1 Ausgewählte Software und Dienste zur Online-Speicherung von Dateien

Name	Typ	Charakteristik	Protokolle	Kapazität	Preismodell
Dropbox	Dienst	proprietär	proprietär	ab 2 GB	abgestuft
Sugarsync	Dienst	proprietär	proprietär	ab 100 GB	abgestuft
Amazon S3	Dienste	proprietär	S3	elastisch	nutzungsabhängig
T-Online Cloud	Dienst	proprietär	WebDAV	25 GB	kostenfrei
OwnCloud	Software	open source	WebDAV	–	–
OpenStack Swift	Software	open source	OS-API	–	–

Die Nutzung eines einzelnen Online-Speicherdienstes ohne Vorverarbeitung der Daten bringt oft Komfortmerkmale mit sich. Sie hat jedoch auch gravierende Nachteile [SBM+11]. Diese resultieren aus den Risiken, die im Abschnitt 2.1 vorgestellt wurden, sowie weiteren Einschränkungen zu Dateigrößen, -typen und -inhalten. Zur Sicherheitslage von Cloud-Storage-Diensten existieren zumeist Studien ohne um-

fassende Lösungsansätze [PB10, MSMF14]. Durch den Einsatz von verteilter Speichertechniken, basierend auf der Dispersion und Verschlüsselung von Daten und dem Multiplexing von Dienstanbietern, lassen sich diese Nachteile jedoch relativieren und teilweise sogar ausschließen [SMS13, SS14b, SGS11, ALPW10]. Die Absicherung erfolgt in Produktivsystemen über die Inhaltsverschlüsselung und der Dateinamenanonymisierung vor der Übertragung, wodurch jedoch die Suche nach einem Namensmuster folgenden Dateien und die Volltextsuche nach Inhalten unterbunden wird. Gesucht wird deshalb ein nutzerkontrolliertes System zur Stealth-Speicherung von Dateien mit den gewünschten Eigenschaften hinsichtlich Sicherheit und Suchfunktionalität.

Die Datendispersion wird dabei möglichst generisch und anwendungsübergreifend in einer dedizierten Komponente durchgeführt. Diese als *Speicher-Controller* bzw. im Umfeld von Speicherdiensten als *Speicherdienst-Controller* bezeichnete Komponente kann als virtuelles Dateisystem, als Laufzeitobjekt einer Klassenbibliothek oder als Netzwerkproxy realisiert sein [SS12a]. Neben der Zerlegung und Zusammensetzung von Daten ist es die Aufgabe des Controllers, die Speicherorte und Dienste einzubinden, die Daten aufzuteilen und auf die Speicherorte zu verteilen sowie benutzerdefinierte Anforderungen wie Mindestverfügbarkeit oder Speicherrichtlinien einzuhalten [SMS13, SS13b].

Abbildung 5.1 definiert das Szenario für die Online-Speicherung von Dateien aus Benutzersicht unter Verwendung eines Speicherdienstcontrollers.

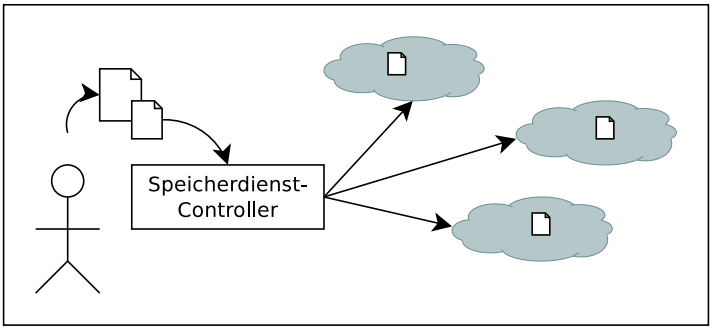

Abb. 5.1 Szenariodefinition für die Online-Speicherung von Dateien

5.2 Persönliche Datenanalyse

Die zunehmenden sensorischen Fähigkeiten von Mobiltelefonen und die zunehmende Vernetzung stationärer und tragbarer Körper- und Aktivitätsmessgeräte (*wearable computing*) hat den Trend zur Analyse persönlicher Daten verstärkt. Man spricht hierbei von *personal data analytics* und *personal data mining* [uRLW+15], *self-*

tracking und *body informatics* sowie im weiter gefassten Kontext der indirekten Aufnahme körperbezogener und biometrischer Daten in Unterhaltungssituationen von *exergames (exercise games)* [HWWF+14], in Sportsituationen vom *e-sports* und in der Gesundheit von *e-health*. Anwendungsfelder sind demnach vorrangig Fitness, Sport, Gesundheit, Unterhaltung und Soziales. Beispielhaft seien die Messung der durchschnittlichen Laufgeschwindigkeit bei einem Marathon und der zeitliche Verlauf des Blutdrucks über den Tag genannt [Mon14]. Weitere personenbezogene Daten umfassen Aufenthaltsorte, Situationen oder Aktivitäten, die mit oder ohne Sensoren per *life-logging* gesammelt werden.

Aufgrund der Popularität der persönlichen Datenanalyse wurden darauf spezialisierte Dienste im Internet eingerichtet, die mit den passenden Anwendungen für Mobiltelefone den Nutzern einen Mehrwert im Sinne einer umfassenden Speicherung und Analyse dieser Daten bieten. Allerdings handelt es sich überwiegend um persönliche und somit schutzwürdige Daten, denen eine Speicherung und Verarbeitung in nicht vertrauenswürdigen Umgebungen entgegensteht, sofern keine Sicherung der Übertragung und Begrenzung des Zugriffs erfolgt [RAG14]. Es kann davon ausgegangen werden, dass das primäre Interesse der Nutzer am Mehrwert und nicht an der meist zugrunde liegenden vollständigen Übergabe der Daten an nicht vertrauenswürdige Anbieter liegt. Mitunter wird die Balance über die Datenhoheit sogar umgekehrt: Während der Dienstanbieter die Daten erhält und sich daran Nutzungs- und Verwertungsrechte einräumen lässt, hat die datengenerierende Person ab einem bestimmten Punkt darauf keinen vollständigen Zugriff mehr. Die Übertragung aller von einem und über einen Nutzer gespeicherten Daten wird unter dem Schlagwort Datenportabilität (*data portability*) und Interoperabilität diskutiert [Gal09]. Eine Import-/Export-Schnittstelle für jegliche Daten und Metadaten ist jedoch bei weitem nicht in allen derartigen Portalen zu finden [Hey08, FK14] und sichert nur gegen die Umkehrung der Balance, nicht jedoch gegen unautorisierte Zugriffe und Auswertungen ab. In ähnlicher Form gilt dies ebenfalls für redundanzfreie und unverschlüsselte verknüpfte Daten (*linked data*).

Die Tabelle 5.2 vergleicht ausgewählte Anbieter zur Übertragung von per Sensorik oder manueller Erfassung erhobener persönlicher Daten zu Zwecken der Aufbereitung und Auswertung.

Tabelle 5.2 Ausgewählte Dienstanbieter für die persönliche Datenanalyse

Name	Domäne	Datenportabilität	Protokoll/API
Heiaheia	Sport	PDF-Export	offen
Endomondo	Sport	GPX-Export	proprietär
Sports Tracker	Sport	GPX-Export	proprietär
Movescount	Sport	keine	proprietär
Fooducate	Ernährung	keine	proprietär
Open Flights	Aktivitäten	CSV-Export	offen

Mit den Techniken des Stealth Computing lässt sich erreichen, dass die persönliche Datenanalyse zwar teilweise oder vollständig ausgelagert, aber dennoch daten-

schutzgerecht durchgeführt wird. Hierzu wird eine vollständig unter der Kontrolle des Anwenders stehende lokale Anwendung zur bidirektionalen Übertragung der Eingangsdaten und zum Abruf aggregierter Daten vorgeschlagen. Diese kommuniziert verteilt mit Anwendungsbestandteilen, die zwar unter der Kontrolle eines Dienstanbieters stehen können, jedoch auf geschützten Daten arbeiten, so dass weder die Eingangsdaten noch die aggregierten Daten für diesen einsehbar sind. Über entstehende Techniken der Code-Obfuscation können zudem vertrauliche Verarbeitungsalgorithmen zwar nicht gänzlich vor unberechtigtem Zugriff geschützt, dieser jedoch deutlich erschwert werden [SKK+14].

Abbildung 5.2 definiert das Szenario für die persönliche Datenanalyse unter Verwendung eines Stealth-Datenbankverwaltungssystems, welches Funktionen eines Speicherdienstcontrollers aufweist, darüber hinaus allerdings auch die Verarbeitung von Anfragen erlaubt.

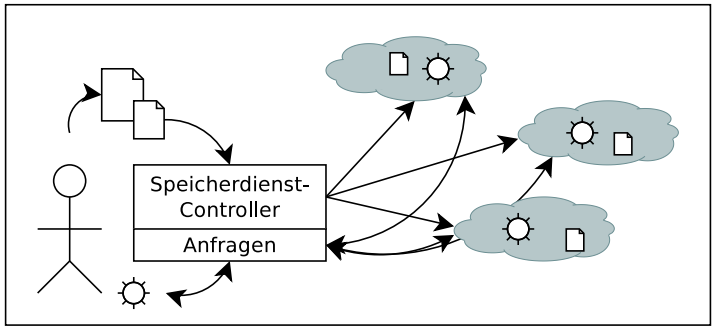

Abb. 5.2 Szenariodefinition für die persönliche Datenanalyse

5.3 Mehrwertdienste für das Internet der Dinge

Das Internet der Dinge (*internet of things*) beschreibt einen industriell getriebenen Trend zum großflächigen Einsatz von vernetzten Sensoren, beispielsweise über drahtlose Sensornetzwerke (*wireless sensor networks*) oder über passiv ausgelesene Identifikatoren wie RFID-, NFC- und BLE-Chips sowie QR-Codes [WSJ15]. Wirtschaftliche Gründe oder Kapazitätsprobleme führen oftmals dazu, dass die von den Sensoren erzeugten Daten, die oftmals die Semantik diskreter Ereignisse aufweisen, in zentral betriebenen Cloud-Diensten und externen Middleware-Plattformen verarbeitet werden [Bad09]. Meist werden diese Daten verdichtet und aufbereitet an domänenspezifische Anwendungen auf Mobilgeräten weitergeleitet. Diese Prozesse und Anwendungen dienen zum einen der Auswertung der Daten (*business intelligence*), zum anderen lässt sich mit ihnen aber auch auf die Verarbeitungskette Einfluss nehmen, womit ein Steuer- und Regelkreis entsteht. Allerdings ergeben

sich daraus mehrere Probleme hinsichtlich der Sicherheit, Zuverlässigkeit und Qualität der Verarbeitung. Ein besonderes Problem stellt der unberechtigte Zugriff auf industrielle Informationen im Rahmen von Industriespionage dar.

Mit den Verfahren des Stealth Computing und einer entsprechenden Infrastruktur lassen sich Anwendungen zur Datennutzung aus dem Internet der Dinge teilautomatisiert erstellen, verbreiten und nutzen.

Abbildung 5.3 definiert das Szenario für den Einsatz von Stealth-Anwendungen im Internet der Dinge.

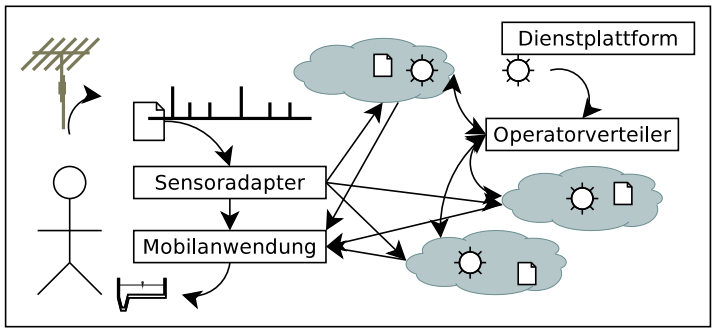

Abb. 5.3 Szenariodefinition für mobile Anwendungen im Internet der Dinge

Kapitel 6
Validierung

Zusammenfassung Die Validierung der vorgestellten Stealth-Daten- und Dienstkonzepte und die Realisierung der Szenarien wird in diesem Kapitel durch Simulationen und Experimente durchgeführt. Einführend werden die Experimentierbedingungen beschrieben. Im Anschluss werden die eingeführten Algorithmen zur Datenkodierung und -verteilung aus Kapitel 3 mit repräsentativen Parameterkombinationen geprüft und hinsichtlich ihrer Ergebnisqualität und ihres Laufzeitverhaltens bewertet. Mehrere Datenkodierungsroutinen und ein Simulationswerkzeug für Verteilungsbestimmungen werden dazu beigesteuert. Zusätzlich werden konkrete Anwendungen und Werkzeuge zur Stealth-Nutzung realer Dienste und Ressourcen, sofern nicht bereits durch eigene Vorarbeiten vorhanden, konzipiert und umgesetzt. Für ausgewählte Stealth-Schnittstellen aus Kapitel 4 kann somit eine experimentelle Bewertung stattfinden. Die Experimente umfassen die Speicherdienstanbindung und -integration sowie die Datenverwaltung mit Verarbeitungselementen und Datenströmen. Die drei letzten Unterkapitel befassen sich mit der Umsetzung der in Kapitel 5 vorgestellten Szenarien zur Online-Speicherung von Dateien, zur persönlichen Datenanalyse und zu mobilen Anwendungen im Internet der Dinge.

6.1 Infrastruktur für die Experimente

Simulationen, Emulationen, Messungen, empirische Beobachtungen und weitere Experimente in existierenden Systemen sind wesentliche Methoden zur Validierung neuer Konzepte und Systemvorschläge in der Informatik [BRNR15, ZW98]. Die Beiträge dieser Arbeit werden hauptsächlich über die Ergebnisse simulierter Abläufe sowie Benchmarks bewertet. Die Messungen zielen primär auf die Einschätzbarkeit der für eine höhere Sicherheit zu akzeptierenden Laufzeiterhöhungen. Abbildung 6.1 gibt einen Überblick über die Zuordnung der Experimente zu den vorgeschlagenen Verfahren und Schnittstellen sowie über die experimentell eingesetzte oder evaluierte Software, welche zusammen mit den jeweiligen Durchführungen in den angegebenen Abschnitten vorgestellt wird.

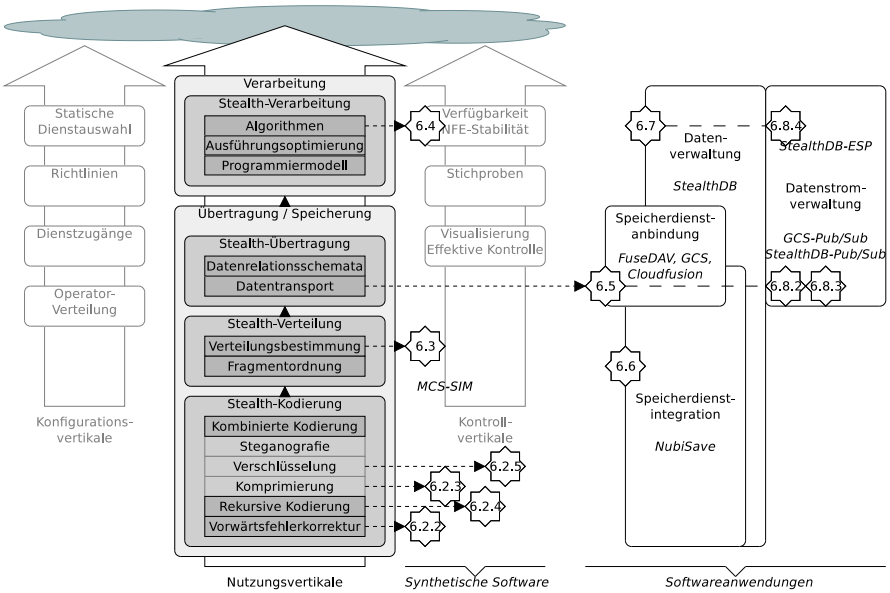

Abb. 6.1 Abdeckung der vorgeschlagenen Verfahren und Schnittstellen durch experimentelle Untersuchungen

An dieser Stelle werden zuerst die eingesetzten Experimentiersysteme beschrieben, um im weiteren Textverlauf unnötige Redundanzen zu vermeiden. Dazu zählen die eingesetzte Hardware, Systemsoftware und Infrastrukturdienste sowie grundlegende Einstellungen zu den Experimenten. Die jeweils genutzten Systeme und Verbindungen werden dementsprechend anschließend referenziert. Das Ziel der detaillierten Beschreibung ist die Gewährleistung der Reproduzierbarkeit der Ergebnisse im Sinne von *executable papers* und *recomputability*. Zu den stabilitätssichernden Maßnahmen zählen der systematische Ausschluss von Störfaktoren und die Mittelung der Messwerte über mehrere, zumeist zehn, Einzelmessungen. Im Anhang C wird auf die Rohdaten und Repositorien zu eingesetzten Werkzeugen verwiesen.

Für die Durchführung der Experimente sind die folgenden Ablaufumgebungen genutzt worden:

1. Raspberry-π-Cluster Bobo/Bobino [AHP+13]. Das System ist repräsentativ für einen Cluster, der unter der vollständigen Kontrolle und im Vertrauensbereich einer Anwendung steht. Der Cluster Bobo besteht aus 40 Knoten des Einplatinencomputers Raspberry-π vom Modell B mit jeweils einem einzelnen ARMv6-Prozessor mit 700 MHz Taktfrequenz, 512 MB SDRAM, einer SD-Speicherkarte sowie einem 100-MBit/s-Ethernet-Anschluss. Als Betriebssystemdistribution wird Raspbian basierend auf Debian 7.4 eingesetzt, wobei als weitere Systemsoftware MegaRPI installiert ist. Der Cluster Bobino vereint in einem ähnlichen Aufbau 8 Knoten.

2. HPI Future SOC Lab [ASP14]. Das Labor des Hasso-Plattner-Instituts für Softwaresystemtechnik der Universität Potsdam Lab stellt semesterweise Ressourcen für Forschungsprojekte zur Verfügung. Genutzt wurde für die Experimente zu Speicherdiensten ein Server vom Typ Hewlett-Packard DL980 G7-1 mit 8 Xeon-X7560-CPUs der Architektur Nehalem EX, welche jeweils 16 Kerne zu 2,27 GHz beinhalten, 2048 GB RAM, eine Festplatte mit 2x146 GB Kapazität im RAID-1-Betrieb zur Replikation sowie einer 4GB-Fibre-Channel-Netzwerkanbindung. Die Betriebssystemdistribution war Ubuntu 12.04 mit dem Linux-Kernel in Version 3.5.0. Für die Experimente zur Datenverwaltung wurde ein Blade-System vom Typ Hewlett-Packard Converged Cloud. Hierbei waren 4 von insgesamt 32 Blades im Einsatz. Zu jedem Blade gehörten zwei Xeon-E5-2620-CPUs mit jeweils 6 Kernen zu 2,0 GHz sowie 64 GB RAM und 10G-Ethernet. Die Speicheranbindung erfolgte über NFS auf ein 3PAR-System mit 3 TB Kapazität. Zu diesen Experimenten sind technische Berichte verfügbar [SM14b, Spi14].
3. Google Cloud [LORZ13]. Als Vertreter der Gruppe kommerzieller Infrastruktur-, Plattform- und Anwendungsdienstanbieter ist die Google Cloud für einige Experimente genutzt worden. Die aktivierten Dienste umfassen die Ausführung virtueller Maschinen mit Compute Engine, die Ausführung von Anwendungen mit App Engine, und die Datenspeicherung mit Cloud Storage.
4. Desktops und Laptops. Zur Abbildung einer realistischen Anwenderperspektive sind ein Desktop-PC und ein Notebook für die Experimente genutzt worden. Der Desktop-PC mit der Bezeichnung Bomba enthält einen Prozessor vom Typ AMD Athlon II X2 Dual-Core 240e mit 2*2,8 GHz Takt, 6 GB RAM, eine SSD vom Typ OCZ-Vertex2 mit 60 GB Kapazität sowie eine HDD vom Typ Seagate Barracuda 7200.12 mit 1 TB Kapazität. Er enthält eine WLAN-Netzwerkkarte vom Typ Realtek RTL8185 und einen Gigabit-Ethernet-Adapter vom Typ Realtek RTL8111. Als Systemsoftware läuft Debian GNU/Linux in Version 8 mit dem Linux-Kernel in Version 3.16. Das Notebook mit der Bezeichnung Rumba ist ein Lenovo-Thinkpad T510 mit Intel-Core-i7-M720-CPU mit 4*2,67 GHz Takt, 8 GB RAM, HDD mit 320 GB Kapazität, einem Gigabit-Ethernet-Adapter vom Typ Intel 82577LM und einem WLAN-Adapter vom Typ Intel Centrino Ultimate-N 6300. Die Systemsoftware ist Debian GNU/Linux Version 7.8 mit dem Linux-Kernel in Version 3.2.

6.2 Experimentelle Validierung der Datenkodierung

Im Abschnitt 3.1 sind theoretische Betrachtungen zu einer gesamtheitlichen Datenkodierung in der Kombination aus Dispersion, Verschlüsselung und weiteren Kodierungsschritten vorgestellt worden. Dazu belegt dieser Abschnitt die Funktionstüchtigkeit und die Laufzeiteigenschaften von datenkodierenden Implementierungen. Mit diesen werden die Verfahren Bitsplitting, Komprimierung durch Lauflängenkodierung und Fragmentexpansion aus Abschnitt 3.1.3 evaluiert. Obgleich kein

neues Verfahren für die homomorphe Verschlüsselung beigetragen wird, erfolgt aufgrund einer eigenen Implementierung dennoch eine umfangreiche experimentelle Untersuchung des Verhaltens, so dass die Eigenschaften aus Abschnitt 3.1.4 nachvollzogen werden können.

6.2.1 Beschreibung der Software

Zur Vorbereitung auf die Dispersion werden die Daten mit diversen Bibliotheken, welche die Algorithmen zur Löschkodierung und zum Bitsplitting implementieren, kodiert. Die Auswahl umfasst die Mehrverfahrensbibliothek Jerasure [Pla13], JigDFS [Bia10] und die eigene Implementierung Bitsplitter [SS14a]. Desweiteren existiert mit Splitter-NG eine Abstraktionsbibliothek für Java-Anwendungen, welche ein Pluginkonzept für derartige Kodierungsbibliotheken aufweist und zudem eine RAID1-Kodierung, ein Replikationsverfahren abgeleitet von *redundant array of independent disks*, als weiteres Plugin beisteuert. Die Plugins stellen einen oder mehrere Codecs bereits. Diese werden über bestimmte Parameter wie n, k oder w konfiguriert und anschließend sowohl für die Kodierung als auch für die Dekodierung verwendet.

Die Implementierung der Fragmentexpansion, der Lauflängenkodierung und der Redundanzsuche erfolgt prototypisch über Python-Skripte als Teil des Repositoriums zu Algorithmen über dispergierte Daten [SS14b]. Ebenso dort enthalten sind eine verbesserte Umsetzung eines existierenden Python-Moduls für homomorphe Verschlüsselung im Paillier-Kryptosystem [Pai99] sowie eigene Ableitungen davon als optimierte C-Bibliotheken.

6.2.2 Experimente zur Dispersion

Das erste Dispersionsexperiment vergleicht das Laufzeitverhalten der Kodierungsimplementierungen mit Hilfe des Splitter-NG-Frameworks. Dadurch wird eine nahezu homogene Vergleichsgrundlage geschaffen, welche in der Form bisher noch nicht existiert. Unterschiede existieren lediglich in der Parametrisierung, da nicht alle Verfahren beliebige Redundanzen unterstützen. Die Parameter sind $k = 1, m = 1$ für RAID1, $k = 2, m = 0$ für Bitsplitting, $k = 2, m = 1$ für Cauchy-Reed-Solomon in der JigDFS-Umsetzung sowie $k = 2, m = 2$ für Blaum-Roth in der Jerasure-Umsetzung. Abbildung 6.2 veranschaulicht die insgesamt ähnlich gelagerten Kodierungszeiten. Auffällig ist, dass für kleinere Dateien der Zeitaufwand mit dem JigDFS-Verfahren deutlich höher ist, für größere Dateien hingegen auch die Bitsplitter-Laufzeit anwächst. Dennoch ist das Bitsplitting auf Dateien mit einer Größe von wenigen MB ohne signifikanten Mehraufwand einsetzbar und somit in Hinblick auf die weitere Verarbeitung der Daten zu empfehlen.

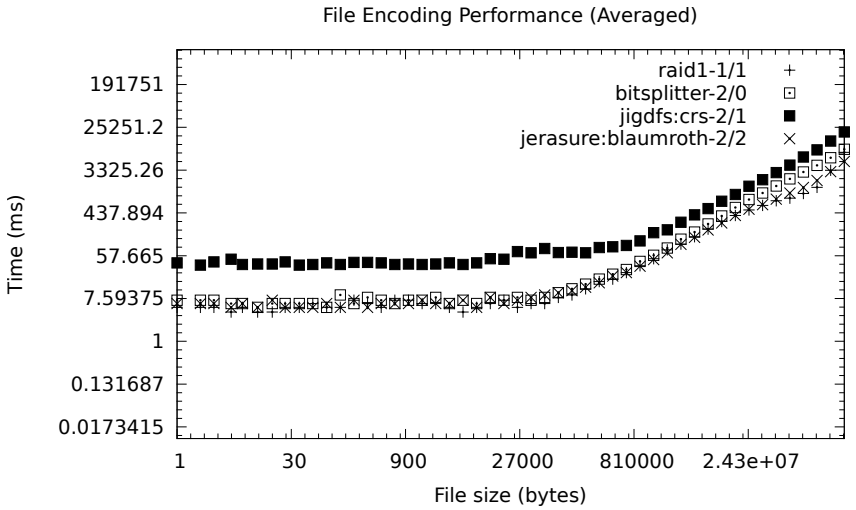

Abb. 6.2 Laufzeitverhalten von Splitter-NG mit verschiedenen Codecs (logarithmische Skala)

Am Beispiel einer Videodatei mit einer Größe von 164*MB* und einer Länge von 602*s* lassen sich die unterschiedlichen Laufzeiten mit einer erweiterten Zahl von Verfahren veranschaulichen. Die Laufzeit beträgt 1378*ms* für eine Replikation mit RAID1, 3474*ms* für Cauchy-Good in der Jerasure-Umsetzung mit $k = 3, m = 3$, 16950*ms* für Bitsplitter, 49290*ms* für JigDFS mit $k = 3, m = 3$, sowie um mehrere Magnituden langsamer 8253614*ms* für die Shamir-Secret-Sharing-Methode ebenfalls mit $k = 3, m = 3$ implementiert in JSharing. Lediglich die letzte Methode ist demnach nicht echtzeitfähig.

Abbildung 6.3 geht gesondert auf das Laufzeitverhalten der Bitsplitter-Implementierung mit verschiedenen Konfigurationen als Kombination von k und w ein. Die Generierung der redundanten Fragmente wird hierbei wiederum nicht berücksichtigt. Aufgrund der Implementierungseigenschaften ist deutlich zu erkennen, dass die Verarbeitungszeiten nicht linear mit den Parametern ansteigen oder anderweitig korrelieren, sondern mehrere lokale Minima und Maxima mit teils deutlichen Performance-Einbrüchen existieren. Ein Grund dafür ist der Einsatz von Puffern fester Größe innerhalb der Bibliothek.

Weitere Experimente belegen ein vorteilhaftes Laufzeitverhalten von Bitsplitter auch mit eingestellter Redundanz [SS14b].

6.2.3 Experimente zur Komprimierung

Die Verträglichkeit der Fragmentbildung mit einer vorherigen Datenkomprimierung wurde experimentell festgestellt. Dazu wurde ein Zufallszahlenreihengenerator mit

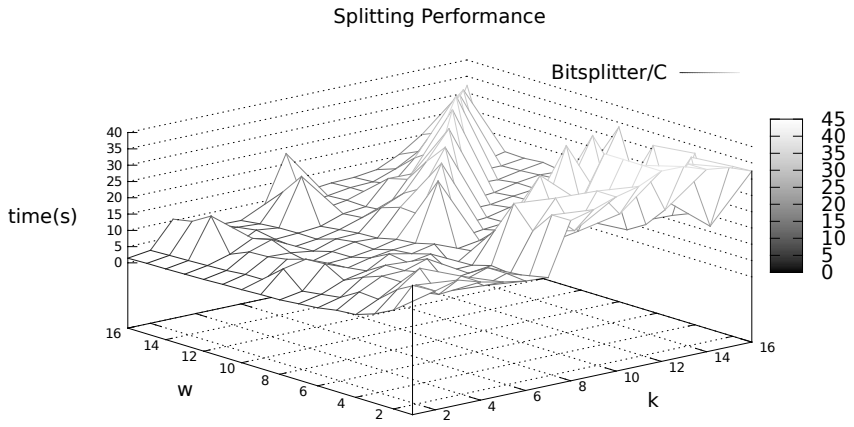

Abb. 6.3 Laufzeitverhalten des Bitsplitters für die Kombinationen von $1 \leq k \leq 16$ und $1 \leq w \leq 16$

einem stochastischen Prozess bei einer wählbaren Zustandsübergangswahrscheinlichkeit $P(trans); 0 \leq P(trans) \leq 1$ implementiert. Desweiteren wurde ein Lauflängenkodierer implementiert, welcher an die PCX-RLE-Kodierung angelehnt ist [Ans91]. Dessen Parametrisierung sieht 8-Bit-Werte (0..255), 2-Bit-Markierungen ($0xC0$) und somit komprimierbare Sequenzen mit bis zu 64 Werten vor. Für jeweils 1000 generierte Zufallszahlen wurde die Komprimierung für unterschiedliche Fragmentzielgrößen sowie ohne Beachtung der Fragmentierung durchgeführt. Abbildung 6.4 zeigt, dass das optimale Komprimierungsverhältnis bei Nichtbeachtung der Fragmentierung ($fw = 0$) sowie Dateien mit monotonem Inhalt ($P(trans) = 0$) erzielt wird. Wird die gewünschte Fragmentlänge auf den minimalen Wert $fw = 2$ gesetzt, wird das schlechteste Verhältnis erzielt, da ca. 50% aller Komprimierungsvorgänge nicht gestartet werden können. Das Verhältnis konvergiert jedoch mit steigendem fw schnell gegen den optimalen Wert.

Die Größe der komprimierten Daten liegt bei $fw = 0$ exemplarisch für $P = 0,05$ bei 14..16%, für $P = 0,50$ hingegen bei 78..81%. Der relative maximale Verlust der Komprimierungsqualität wird in der Grafik 6.5 gesondert dargestellt. Daraus geht hervor, dass mit steigender Transitionswahrscheinlichkeit $P < 0,5$ zuerst bis zu $6,5\%$ geringere Komprimierungsraten erzielt werden. Steigt P hingegen weiter an, so sinkt das Komprimierungsverhältnis ohnehin stark ab, so dass der Grad der Degradierung wieder bis auf 0% zurückgeht.

Die Schlussfolgerung lautet somit, dass bei einer geplanten Weiterverarbeitung eine Lauflängenkodierung als Komprimierungsmethode angepasst auf die Fragmentierung eingesetzt werden sollte, da ansonsten das Risiko besteht, mit fehlerhaften Daten zu arbeiten.

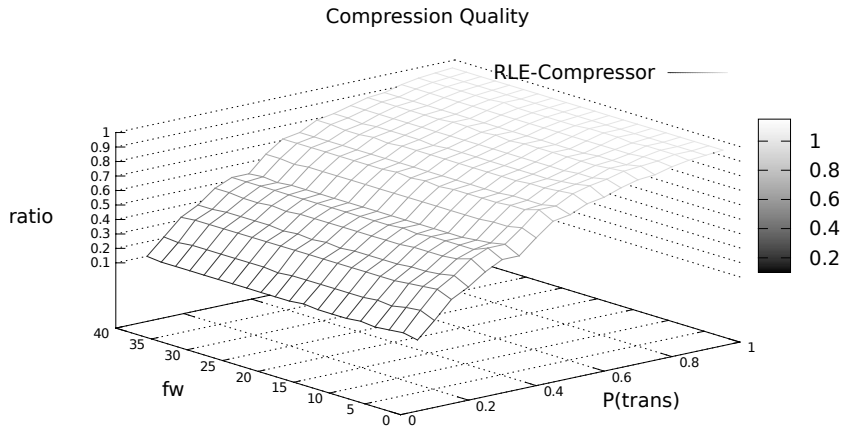

Abb. 6.4 Komprimierungsqualität einer Lauflängenkodierung in Abhängigkeit von der Fragmentgröße (niedrigere Werte bedeuten höhere Qualität)

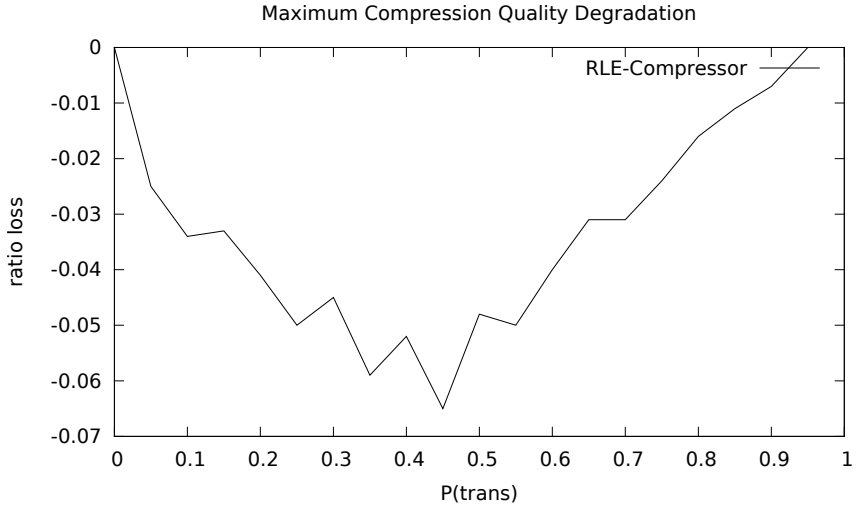

Abb. 6.5 Degradierung der Komprimierungsqualität von $fw = 0$ zu $fw = 2$

Abschließend sei noch die Laufzeit der Komprimierung abhängig von der Transitionswahrscheinlichkeit berücksichtigt. Abbildung 6.6 zeigt das Laufzeitverhalten der Komprimierung einer Liste mit einer Million 8-Bit-Zahlen. Erkennbar ist das der initiale Aufwand im Idealfall $P = 0$ von $313ms$, dem leichten, nichtlinearen Anstieg bis zum Maximum von $566ms$, und schließlich einer minimalen Reduktion

auf 560*ms* im letzten Abschnitt. Demgegenüber steht eine zwar nichtlineare, aber monoton stetige Reduktion der Komprimierungsrate von 100% auf 0%.

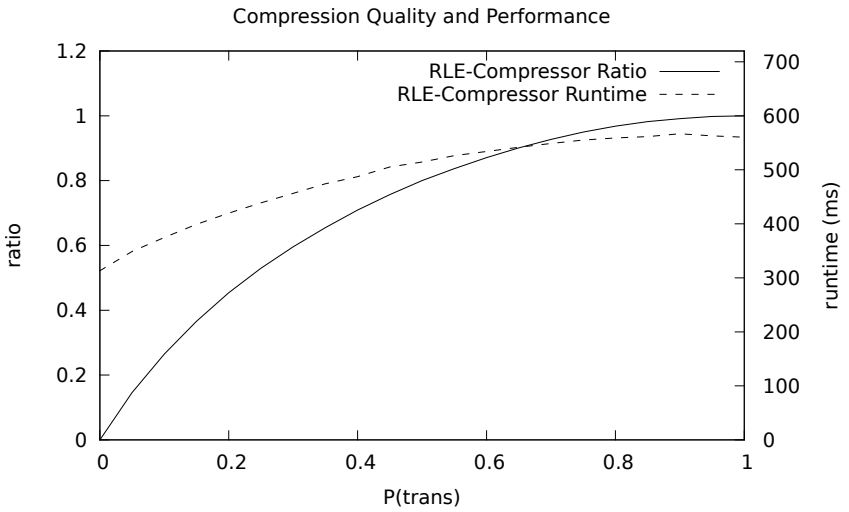

Abb. 6.6 Komprimierungsqualität und Laufzeit mit $fw = 4$

6.2.4 Experimente zur Fragmentexpansion

Die prototypische Umsetzung des Fragmentexpansionsalgorithmus belegt, dass Verfahren zur rekursiven Dispersion von Daten ohne Abbruchbedingung praktisch angewandt werden können. Die Expansion eines Bits in einen größeren Wert zwecks sicherer Verteilung wird in Abbildung 6.7 auf das Laufzeitverhalten unter mehreren Parameterkombinationen geprüft. Das Verhalten ist weitestgehend linear proportional zu den Parametern k und w. Für praktische Belange beträgt die Laufzeit mit der Python-Implementierung bei $k = 2, w = 3$ lediglich $7,66\mu s$; bei $w = 13$ bereits $12,89\mu s$; und bei der untersuchten theoretisch möglichen Kombination $k = 492, w = 993$ mithin $859\mu s$. Interpoliert auf ein Datenvolumen von $164MB$, bezogen auf die mit unterschiedlichen Dispersionskodierungen betrachtete Videodatei, beträgt die Laufzeit für $w = 13$ damit etwa $4,9h$, was einen Einsatz in Echtzeit ausschließt, insbesondere da die Dispersionszeit aufaddiert werden muss. Vergleichend enthält die Anwendung die Laufzeitmessungen für die C-Implementierung. Hierbei verringert sich die Kodierungszeit für die Videodatei immerhin deutlich auf $1,8h$, wobei dieser Wert trotz möglicher Code-Optimierungen immer noch zu hoch für eine Echtzeitkodierung ist.

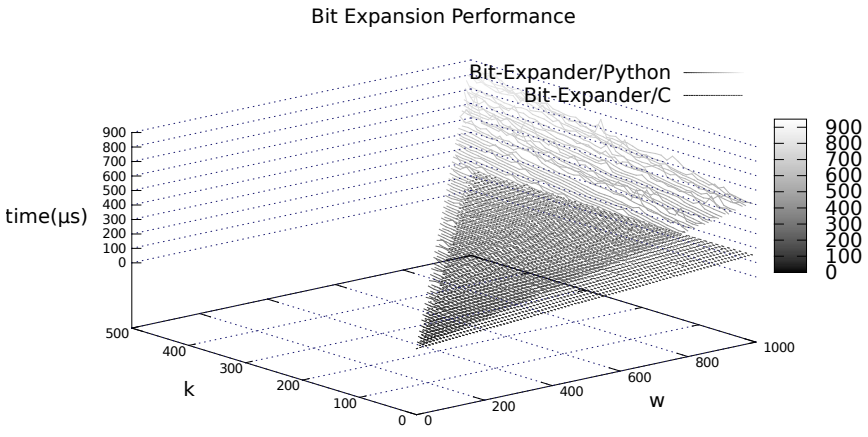

Abb. 6.7 Laufzeitverhalten des Bit-Expanders für die Kombination von $3 \leq w \leq 1000$ und $2 \leq k \leq \frac{w}{2} + 1$

6.2.5 Experimente zur Verschlüsselung

Zwar existieren bereits viele Publikationen zu homomorpher und ordnungserhaltender Verschlüsselung, eine umfassende Messung des Laufzeitverhaltens mehrerer Implementierungen zur Vergleichbarkeit wurde jedoch noch nicht durchgeführt. In diesem Abschnitt werden deshalb mehrere Varianten des Paillier-Kryptosystems und eine des Boldyreva-Kryptosystems evaluiert. Diese homomorphen Verfahren umfassen eine reine Python-Implementierung namens *Pure Python paillier*, eine beschleunigte Reimplementierung dessen in C mit Python-Wrapper namens *paillierfastenc*, sowie eine weitere unabhängige C-Implementierung namens *SRI libpaillier* in Version 0.8. Weitere Bibliotheken wie *HElib* existieren, werden aber für den Vergleich nicht berücksichtigt. Das ordnungserhaltende Verfahren wird durch das Ruby-Modul *ope.rb* repräsentiert.

Die Doppelabbildung 6.8 vergleicht die ersten zwei Implementierungen hinsichtlich der benötigten Laufzeit für die Schlüsselerzeugung (`keygen`), der Ver- und Entschlüsselung von Daten (`enc`, `dec`) sowie der Addition zweier verschlüsselter Zahlen im homomorphen Raum (`add`). Dabei werden auf dem System `rumba` Schlüssel mit einer Länge zwischen 1 und 32 Bits genutzt.

Es lässt sich beobachten, dass die Verschlüsselung den größten Zeitbedarf und die geringste Vorhersagbarkeit über alle Schlüssellängen hinweg besitzt. Auch die Schlüsselerzeugung ist relativ langsam, weist dafür aber trotz Primzahlsuche ein stabiles Verhalten auf. Berechnung und Entschlüsselung sind sehr performant, was es ermöglicht, Anwendungen auf bereits verschlüsselten Bestandsdaten oder auf Daten mit geringen Änderungen, beispielsweise Append-Only-Datenbanken basierend

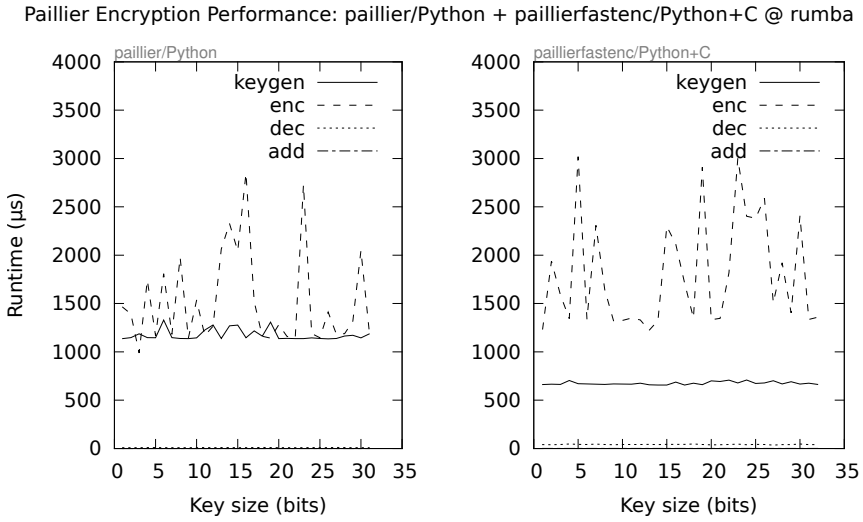

Abb. 6.8 Laufzeitverhalten des unbeschleunigten und des beschleunigten Python-Paillier-Verschlüsselungsmoduls auf dem Rumba-System

auf Datenströmen, mit ausreichender Geschwindigkeit arbeiten zu lassen. Der relative Vergleich zwischen den beiden Implementierungen zeigt zudem einen geringen Geschwindigkeitsvorteil der nativen Implementierung bei der Schlüsselerzeugung bei ansonsten wenig Unterschieden.

Zum Vergleich zeigt die Doppelabbildung 6.9 das Laufzeitverhalten auf einem Rechnerknoten in der Experimentierumgebung `bobo`. Die relativen Zeitunterschiede zwischen den beiden Systemen bleiben erhalten. Die absoluten Zeiten erhöhen sich jedoch signifikant von einem Bereich zwischen 0 und maximal $3,5ms$ auf eine Obergrenze von $140ms$.

Zur Vermeidung von Laufzeiteinbußen aufgrund der Nutzung einer Abstraktionsschicht für die Programmiersprache Python sollen abschließend auch die direkten C-Aufrufschnittstellen für sinnvolle Schlüssellängen von 16 bis 32 Bit verglichen werden. Doppelabbildung 6.10 zeigt die Ergebnisse. Es ist offensichtlich, dass mit *libpaillier* eine schlüssellängenunabhängige stabile Verarbeitungszeit für die Schlüsselgenerierung und die Verschlüsselung von ca. $1.1ms$ erzielt werden kann. Demgegenüber erreicht *paillierfastenc* die bereits vorher ersichtlichen Varianzen, ist jedoch für nahezu alle Additionsoperationen und zumindest kleinere Schlüssellängen performanter. Insbesondere für homomorph verschlüsselte 8-Bit-Zahlen ist *paillierfastenc* damit zu empfehlen.

Die Implementierung des Boldyreva-Schemas für ordnungserhaltende Verschlüsselung setzt auf OpenSSL auf. Aus diesem Grund sind die zulässigen Schlüssellängen auf 128, 192 und 256 Bit im AES-ECB-Modus begrenzt. Abbildung 6.11 veranschaulicht die Ergebnisse der Laufzeituntersuchungen. Während die Schlüsselerzeugung basierend auf einer Zufallszahl keine messbare Verzögerung verursacht,

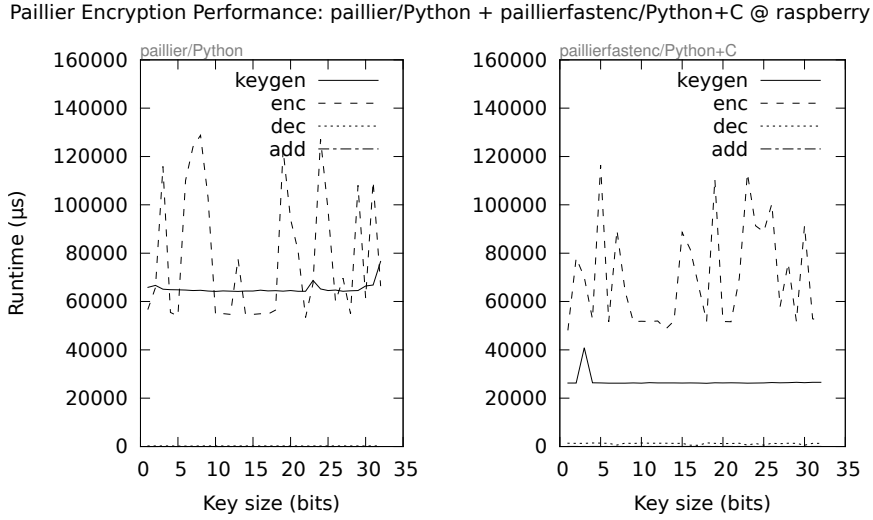

Abb. 6.9 Laufzeitverhalten des unbeschleunigten und des beschleunigten Python-Paillier-Verschlüsselungsmoduls auf dem Bobo-Raspberry-System

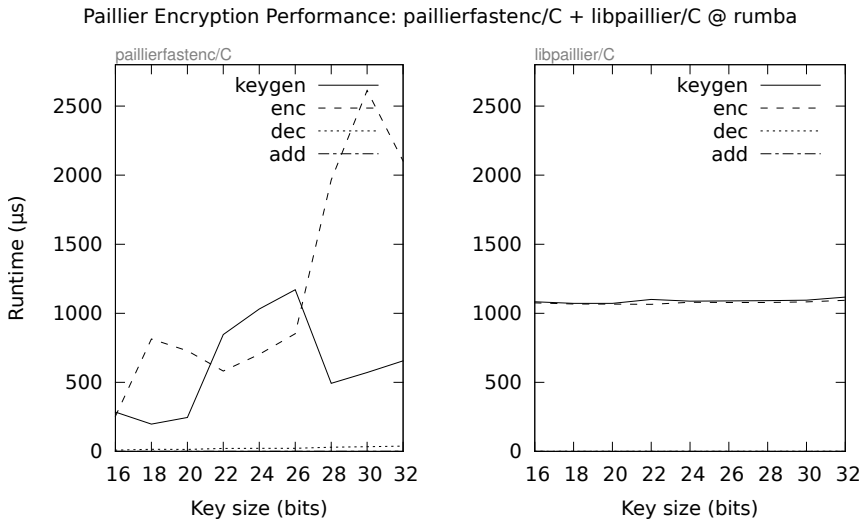

Abb. 6.10 Laufzeitverhalten der direkten C-Aufrufschnittstellen von paillierfastenc und libpaillier

ist die Verschlüsselung zwar zeitintensiv, jedoch konstant über alle Schlüssellängen. Die Entschlüsselung ist aufgrund eines Softwarefehlers im Ruby-Modul nicht Teil der Beobachtung.

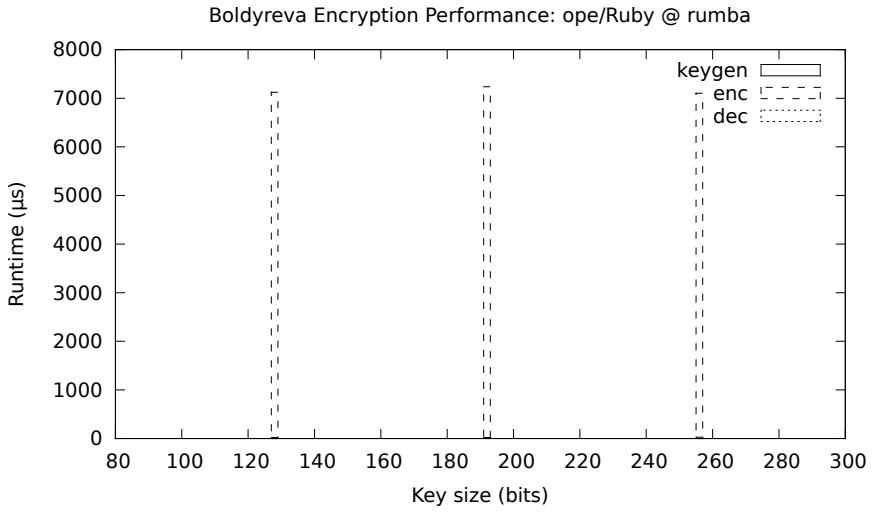

Abb. 6.11 Laufzeitverhalten der Ruby-OPE-Bibliothek

6.3 Experimentelle Validierung der Datenverteilung

Die im Abschnitt 3.2 vorgestellten und vorgeschlagenen Algorithmen zur Bestimmung einer Datenverteilung und Abbildung dieser auf eine Dienstauswahl werden sowohl anhand von Szenarien als auch vollständig über eine kombinatorische Parametrisierung der wesentlichen Parameter Verfügbarkeit, Kapazität und Preis jedes Dienstes validiert. Dazu wird ein Simulationswerkzeug eingesetzt, welches erstmalig einen direkten interaktiven Vergleich zwischen Verteilungsbestimmungsverfahren ermöglicht. Damit werden zuerst Gesamtverfügbarkeiten über Einzelverfügbarkeiten und über Varianzen bestimmt, die Laufzeit der Bestimmung gemessen und die Ergebnisse grafisch dargestellt. Die Güte der Approximierung mit der Monte-Carlo-Simulation ergänzt diese Beobachtungen. Im Anschluss werden die einzelnen Verfahren der Verteilungsbestimmung untersucht. Diese umfassen die Gleichverteilung, die Kombinationssuche, PICav, die Staffelverteilung und PICav+.

6.3.1 Beschreibung der Software

Zur Durchführung der Experimente dient das in der Programmiersprache Python implementierte Simulationswerkzeug *Multi Cloud Storage Simulation* (MCS-SIM). Es läuft als grafisches oder textausgebendes Programm und bietet mehrere Parameter zur Beeinflussung des Ablaufs an. MCS-SIM lädt Dienstbeschreibungen aus einer Datei oder generiert sie zufällig. Abbildung 6.12 beinhaltet die Implementierungsarchitektur von MCS-SIM.

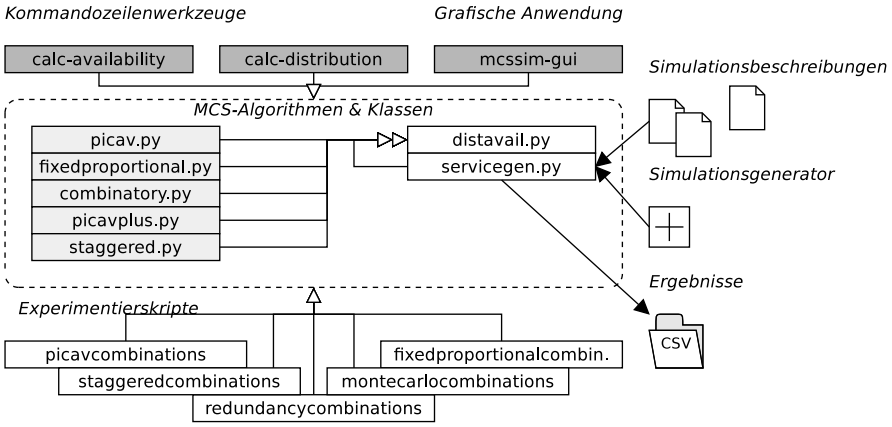

Abb. 6.12 Architektur der Simulations- und Kombinationsumgebung MCS-SIM

Im Kern werden sowohl Dienste und Dienstbündel als auch die Verfahren jeweils durch eigene Klassen innerhalb einer Klassenbibliothek repräsentiert. Zusätzlich zu den Simulationsprogrammen bietet die Bibliothek Schnittstellen für Experimentierskripte, von denen mehrere für die Durchführung der Experimente in diesem Abschnitt erstellt worden sind. Im Anhang B sind weitere Angaben zu MCS-SIM aufgeführt.

6.3.2 Bestimmung der Gesamtverfügbarkeit

Die Gesamtverfügbarkeit einer unterschiedlich großen Menge von Cloud-Diensten abhängig von homogenen Einzelverfügbarkeiten (Varianz $s^2 = 0$) wird in Abbildung 6.13 gezeigt. Hierbei fällt auf, dass der Einfluss der Einzelverfügbarkeiten bei geringer Dienstanzahl dominant ist, jedoch bei höheren Verfügbarkeiten hinzugenommene Dienste die Gesamtverfügbarkeit signifikant erhöhen.

Werden hingegen bei einer nahezu konstanten durchschnittlichen Verfügbarkeit (im arithmetischen Mittel $\mu = 50\%$) unterschiedlich hohe Varianzen ($s^2 > 0$) bis hin zu einer Bandbreite der Einzelverfügbarkeiten von $30\% \leq a \leq 70\%$ betrachtet, so fällt auf, dass nur bei einer sehr geringen Dienstanzahl $(2 \ldots 3)$ die Varianz überhaupt Einfluss auf die Gesamtverfügbarkeit nimmt. Dies bedeutet, dass selbst bei niedrigen Verfügbarkeiten Schwankungen in der Verfügbarkeit leicht durch die Hinzunahme eines weiteren Dienstes abgefangen werden können. Würde man $\mu = 90\%$ setzen, wäre der Einfluss dementsprechend noch geringer. Abbildung 6.14 zeigt den Einfluss der Varianz.

Die Ausführungszeit des Algorithmus zur vollständigen Gesamtverfügbarkeitsberechnung hängt, wie aus Abbildung 6.15 ersichtlich, nicht von den Einzelverfügbarkeiten, dafür exponentiell von der Dienstanzahl ab. Dies zeigt deutlich, dass für

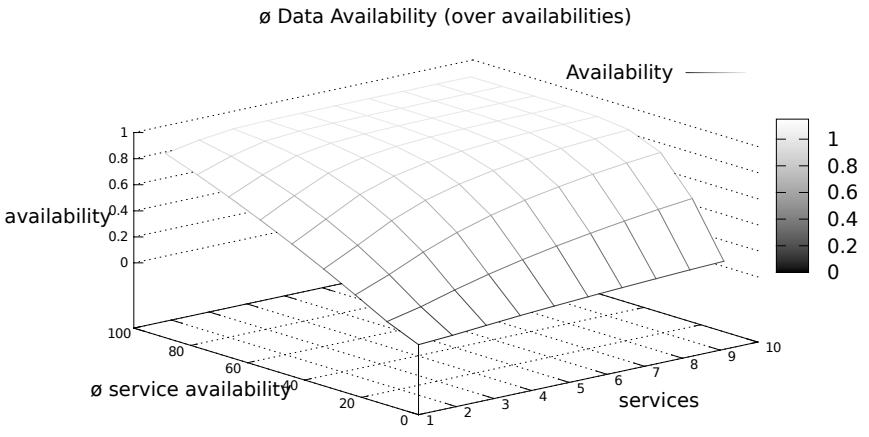

Abb. 6.13 Gesamtverfügbarkeitsbestimmung über Einzelverfügbarkeiten

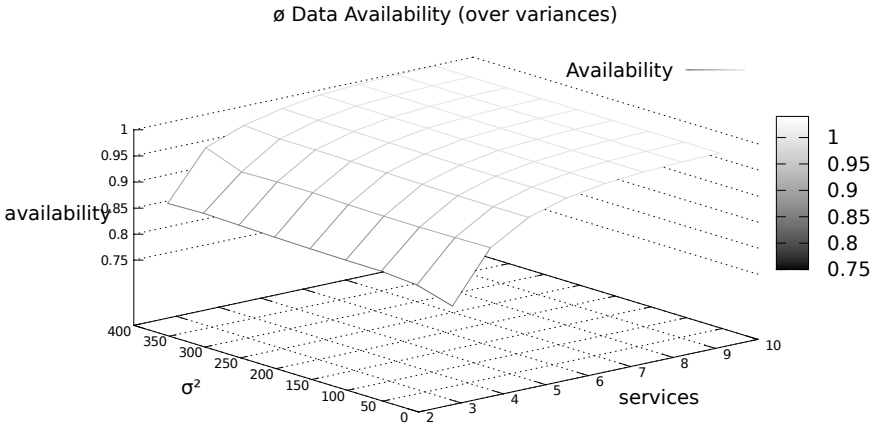

Abb. 6.14 Gesamtverfügbarkeitsbestimmung über Varianzen

eine Berechnung in Echtzeit, beispielsweise in einem grafischen Konfigurationsdialog oder einer autonomen Dienstkonfiguration mit Echtzeitanforderungen, bei sehr hoher Dienstzahl keine vollständige Berechnung insbesondere in kombinatorischen Verfahren mit vielen Berechnungsaufrufen erfolgen kann. Stattdessen muss in solchen Fällen eine Abschätzung des Wertes erfolgen.

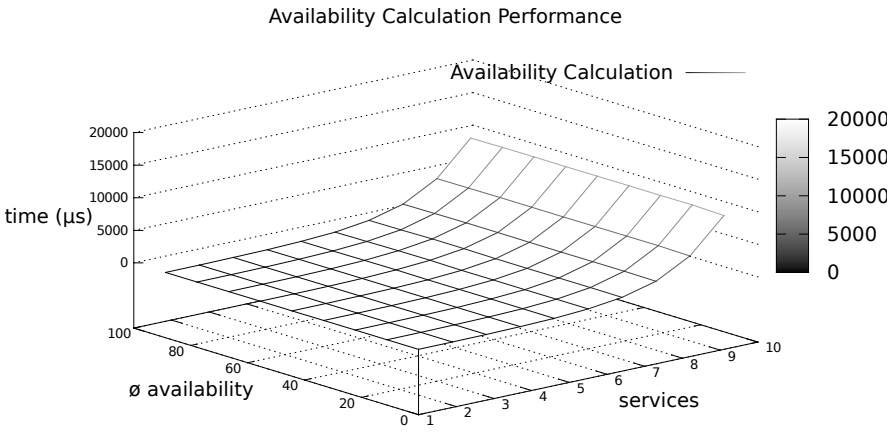

Abb. 6.15 Laufzeitverhalten der Gesamtverfügbarkeitsbestimmung

Die Abbildungen 6.16 und 6.17 zeigen vergleichend die Qualitätsverluste bei der Approximierung der Gesamtverfügbarkeit mit dem Monte-Carlo-Verfahren in zwei Konfigurationen ($k = 3/k = 4$ bei $n = 4$). Die experimentelle Bestimmung erfolgt in Abhängigkeit von einerseits der Zahl der Wiederholungen (`trials`) und andererseits dem Startwert der Iteration (ω), sowie in beiden Fällen einer ε-Umgebung. Klar ersichtlich wird die hohe Abweichung bei einer zu kleinen ε-Umgebung als Abbruchbedingung, da in diesem Fall nach dem erfolglosen Ablauf der Iteration das Ergebnis 0 geliefert wird, der Differenzwert folglich dem berechneten Verfügbarkeitswert entspricht. Eine weitere Beobachtung, wenn auch weniger deutlich ausgeprägt, ist die gleichmäßigere Abweichung für $k < n$ aufgrund des Konvergenzverhaltens der Gesamtverfügbarkeit bei Vorhandensein redundanter Elemente. Ist hingegen $k = n$, sind die erreichbaren Gesamtverfügbarkeiten insgesamt geringer und Abweichungen damit weniger eingeschränkt.

Die Reverssuche einer passenden Verteilung zur Erfüllung bestimmter Kriterien ist wesentlich komplexer, da für verschiedene Kombinationen jeweils der Algorithmus für die Bestimmung der Gesamtverfügbarkeit ausgeführt werden muss, um die Kombination entweder zu berücksichtigen oder zu verwerfen. Als Kriterien werden an dieser Stelle sowohl der zusätzliche Platzbedarf zur Einhaltung der Verfügbarkeit als auch, falls bekannt, der Preisunterschied festgelegt. Desweiteren ist das Laufzeitverhalten des Algorithmus von Interesse. Die Messungen berücksichtigen sowohl die erstbeste Verteilung (lokales Optimum) als auch die optimale Verteilung aus dem vollständigen Suchraum (globales Optimum).

Für die weiteren Experimente zur Bestimmung einer optimalen Datenverteilung wurden mittels MCS-SIM 10 datenverarbeitende Dienste mit gleichmäßig zufällig verteilten Einzelverfügbarkeiten a im Bereich $90\% \ldots 100\%$ sowie währungslosen

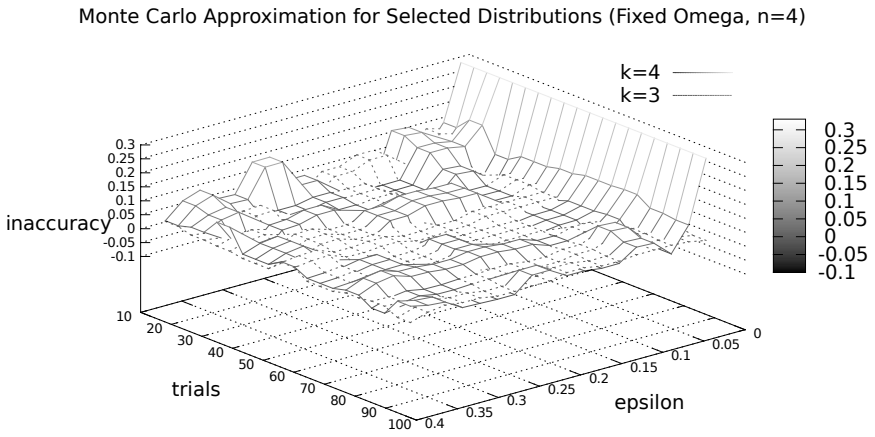

Abb. 6.16 Güte der Approximierung der Gesamtverfügbarkeit mit dem Monte-Carlo-Verfahren abhängig von ε und der Zahl der Wiederholungen

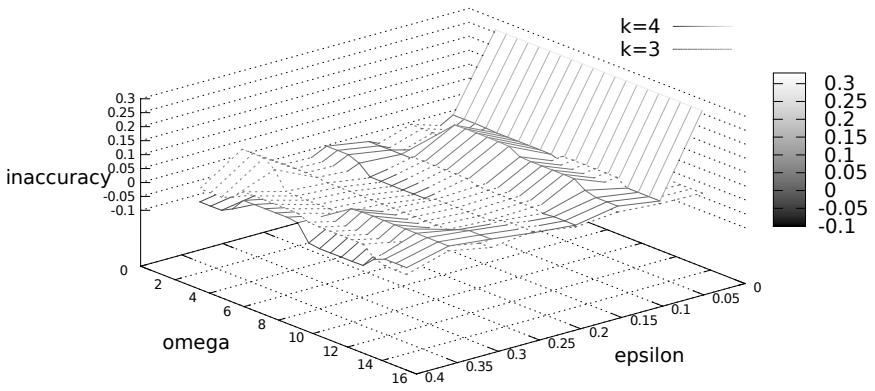

Abb. 6.17 Güte der Approximierung der Gesamtverfügbarkeit mit dem Monte-Carlo-Verfahren abhängig von ε und der Wahl des ursprünglichen ω

Preisen im Bereich $a \dots a+1$ simuliert. Üblicherweise liegen die Entgelte für Dienste mit höherer Qualität auch höher, wodurch dies eine passende Parametrisierung der Simulation darstellt. Durch kombinatorische Parameterwahl wurde die Menge der Kandidatendienste h schrittweise von 1 auf 10 erhöht und weiterhin die Ziel-

verfügbarkeit von $a = 90\%$ bis $a = 100\%$ angehoben. Im Folgenden werden nun die Verfahren Gleichverteilung, Kombinationssuche, kombinatorische Staffelsuche, PICav und PICav+ durch die Simulationsergebnisse evaluiert.

6.3.3 Bestimmung der Verteilung: Gleichverteilung

Abbildung 6.18 zeigt in der Gegenüberstellung die Gleichverteilung und die Proportionalverteilung nach Verfügbarkeit. Die observierte Größe ist dabei der Overhead als Maß für den Verlust an Nettokapazität. Somit entsteht ein Mehrbedarf, wenn proportional verteilt wird und einige Dienste zwar Ressourcen vorhalten, diese jedoch nicht genutzt werden.

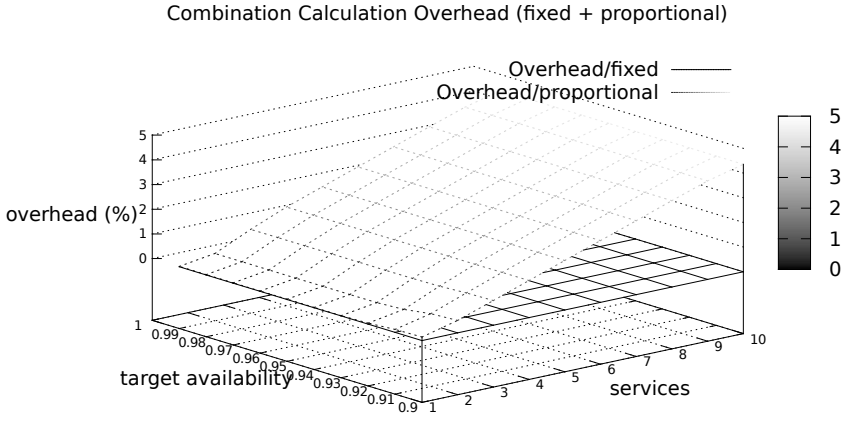

Abb. 6.18 Mehrbedarf an Kapazität einer bestimmten Verteilung zu einer Mindestverfügbarkeit mit der einfachen Verteilung

6.3.4 Bestimmung der Verteilung: Kombinationssuche

Die Implementierung der Kombinationssuche findet zuerst einen beliebigen auf das Verfügbarkeits- und das Preiskriterium passenden Dienst und optimiert das Ergebnis in weiteren Ablaufschritten auf einen günstigeren Preis, jedoch nicht auf eine höhere Verfügbarkeit. Somit entstehen zwei Optima, ein schnelles erstes und ein späteres bestes Resultat. Abbildung 6.19 vergleicht das Laufzeitverhalten zwischen

den beiden Optima. Korrespondierend dazu zeigt Abbildung 6.20 den Vergleich des Preises zwischen den beiden Optima.

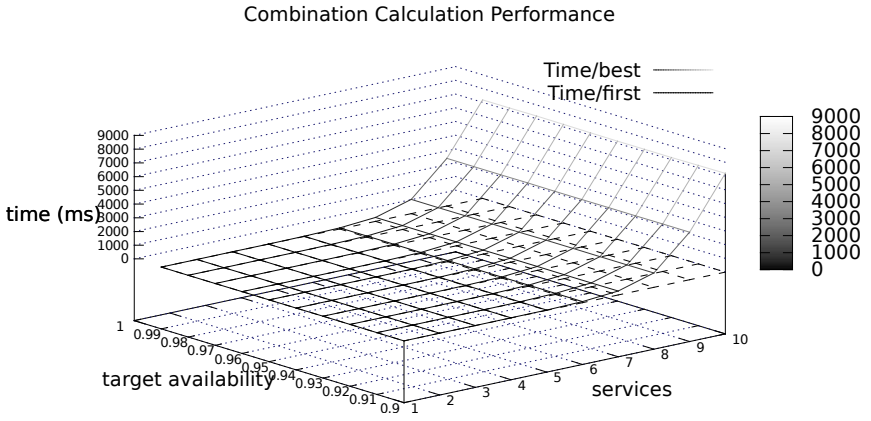

Abb. 6.19 Laufzeitverhalten der Verteilungsbestimmung zu einer Mindestverfügbarkeit der Kombinationssuche

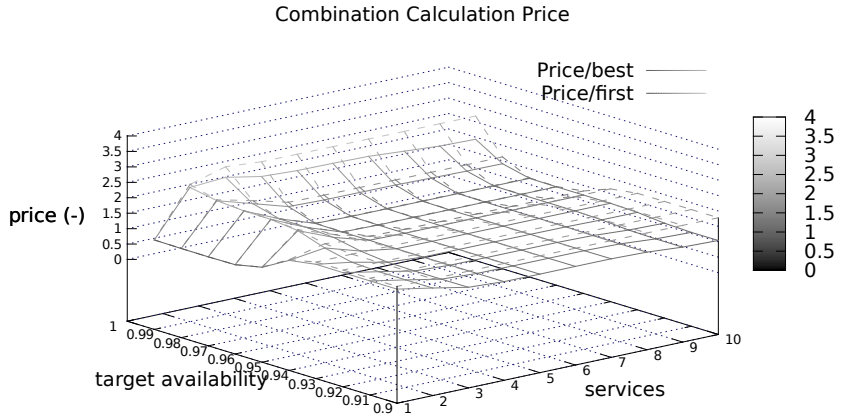

Abb. 6.20 Preis einer bestimmten Verteilung zu einer Mindestverfügbarkeit mit der Kombinationssuche

Hierbei fällt auf, dass gerade bei einer höheren Dienstanzahl für einen geringen durchschnittlichen Aufpreis eine wesentlich schnellere Verteilung gefunden werden kann. Zudem wird die Rekonfiguration einer Menge von Cloud-Diensten insgesamt eher selten vorgenommen und kann durch die zusätzliche Wartezeit, sofern sie vertretbar ist, langfristig ökonomisch günstiger ausfallen. Der Algorithmus durchsucht zuerst die Potenzmengen mit jeweils einem Anbieter und begünstigt damit einzelne Anbieter mit hoher angenommener Verfügbarkeit gegenüber einer großflächigen Verteilung. Andererseits besteht das Risiko darin, dass die angenommene Verfügbarkeit nicht eingehalten wird und dies nicht durch SLAs oder Prüfverfahren abgesichert ist, wofür es allerdings Lösungsansätze gibt [Spi10]. Die Laufzeit beträgt auf dem vorgestellten Testsystem für eine Kombination aus 10 Diensten etwa 9*s* pro Ablauf der Verteilungsbestimmung sowie für eine in den Diagrammen nicht dargestellte Kombination aus 15 Diensten bereits etwa 4500*s* pro Ablauf.

Abbildung 6.21 veranschaulicht den Mehrbedarf an Kapazität im Fall der Nutzung der preisoptimalen Datenverteilung. Auffallend ist dabei, dass auch hinsichtlich des Mehrbedarfs an Speicherplatz fast immer das Optimum erreicht wird. Jedoch gibt es Ausnahmen, welche die Verfügbarkeit nur durch Replikation erreichen und somit einen zusätzlichen Bedarf von 100% aufweisen. Insbesondere ist dies für Zielverfügbarkeiten der Fall, die höher als die maximale angenommene Verfügbarkeit aller Einzeldienste liegen.

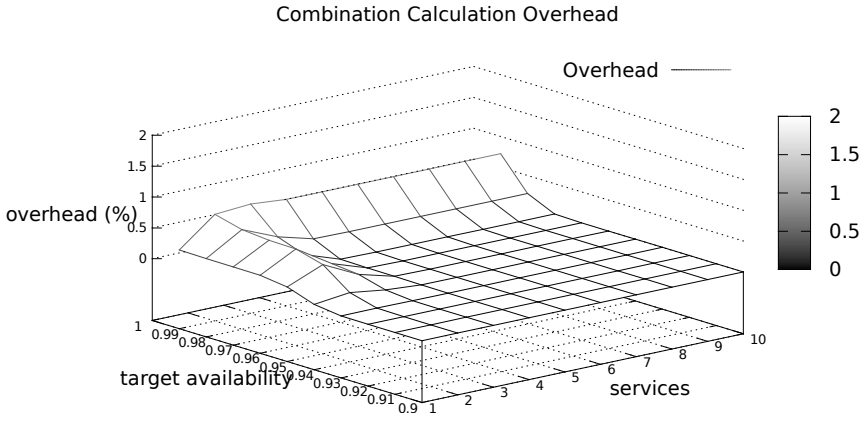

Abb. 6.21 Mehrbedarf an Kapazität einer bestimmten Verteilung zu einer Mindestverfügbarkeit mit der Kombinationssuche

6.3.5 Bestimmung der Verteilung: PICav-Verfahren

Mit dem PICav-Verfahren wird aufgrund der von der Dienstzahl unabhängigen Zahl an Iterationsschritten eine wesentlich reduzierte Laufzeit erreicht. Abbildung 6.22 enthält die Messergebnisse für eine PICav-Bestimmung, aus der ein langsamerer Anstieg der Messkurve im Vergleich mit dem kombinatorischen Verfahren hervorgeht.

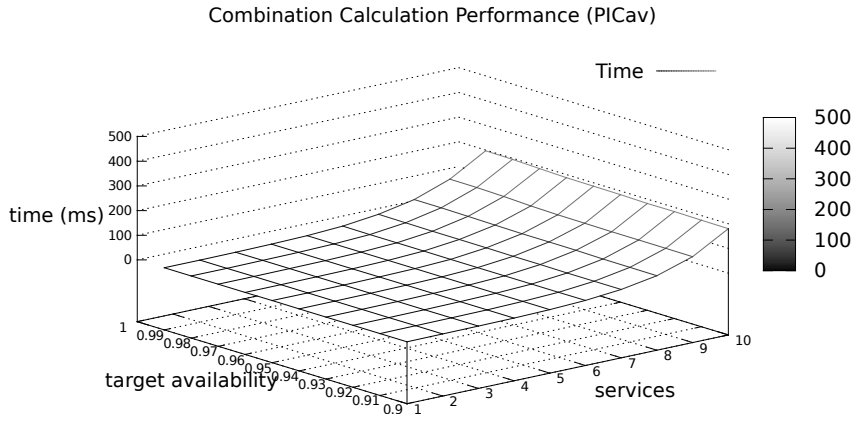

Abb. 6.22 Laufzeitverhalten der Verteilungsbestimmung zu einer Mindestverfügbarkeit mit PICav

Da das Verfahren ohne Berücksichtigung der Entgelte arbeitet, kann lediglich die benötigte zusätzliche Kapazität als Gütekriterium für die Optimierung genutzt werden. In Form eines Mehrbedarfs (Overhead) wird sie in Abbildung 6.23 dargestellt. Der Mehrbedarf beträgt in der Hälfte aller Fälle 0%, übertrifft aber in wenigen Ausnahmefällen auch 100%.

Die entstehenden Mehrkosten sind nicht anhand von Parametern durch das Verfahren selbst kontrollierbar, sie lassen sich allerdings durch MCS-SIM im Anschluss bestimmen. Die Mehrkosten im Evaluierungsbeispiel weisen dabei eine ähnliche Verteilung wie der Mehrbedarf an Kapazität auf, wie in Abbildung 6.24 dargestellt.

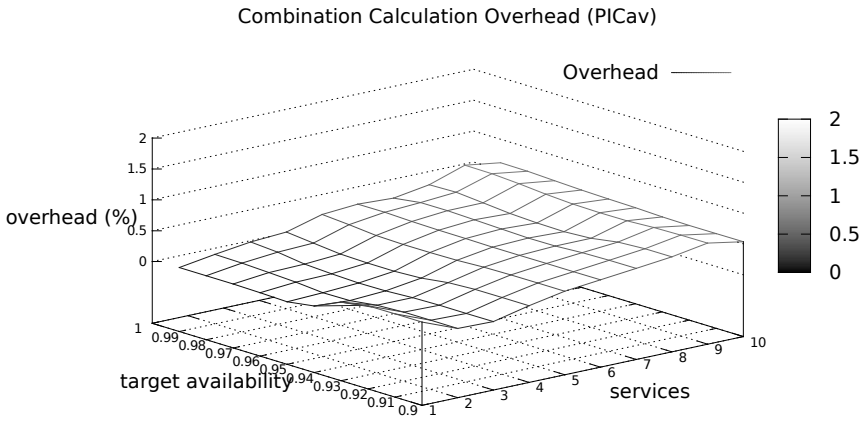

Abb. 6.23 Mehrbedarf an Kapazität einer bestimmten Verteilung zu einer Mindestverfügbarkeit mit PICav

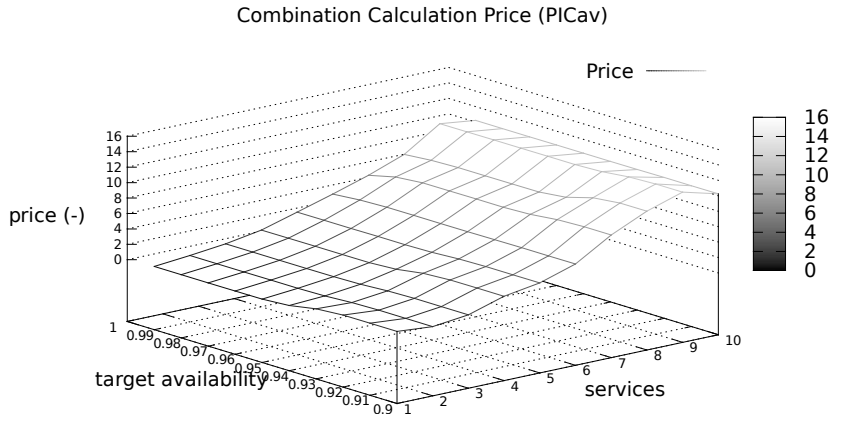

Abb. 6.24 Mehrkosten einer bestimmten Verteilung zu einer Mindestverfügbarkeit mit PICav

6.3.6 *Bestimmung der Verteilung: kombinatorische Staffelverteilung und PICav+*

In weiteren Simulationsdurchläufen werden mit dem kombinatorischen Staffelverteilungsverfahren und PICav+ zwei Verfahren analysiert, die Verfügbarkeit, Kapazität und Preis als Auswahlkriterien auswerten.

Abbildung 6.25 beinhaltet die Zahl der mit dem gestaffelten Verfahren gefundenen Verteilungen mit Zusicherung von Kapazitäts- und Verfügbarkeitsmindestwerten. Die höchste Ergebnismenge wird bei geringen Anforderungen an Kapazität und Verfügbarkeit erzielt. Bemerkenswert ist die niedrige Ergebnismengenreduktion bei erhöhter Kapazitätsanforderung gegenüber einer späten, jedoch dafür umso schnelleren, Reduktion bei sehr hohen Verfügbarkeiten.

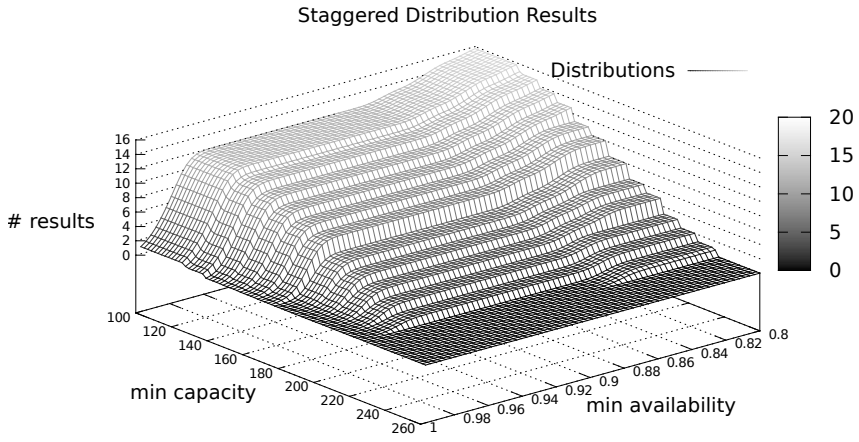

Abb. 6.25 Gestaffelte Fragmentverteilung mit Verfügbarkeits- und Kapazitätseinschränkungen (interpolierte Darstellung)

Abbildung 6.26 betrachtet die Laufzeiteigenschaften der kombinatorischen Staffelverteilungsbestimmung im Vergleich mit der zwar auch gestaffelten, aber gruppierend-iterativ arbeitenden PICav+-Verteilungssuche.

Die Laufzeit wird für zwei Zielverfügbarkeiten, $a = 0\%$ und $a = 99,9\%$, bestimmt. Aus dem Diagramm wird deutlich, dass es einen Wechselpunkt (*sweet spot*) gibt, welcher ab 8 Diensten die PICav+-Suche nach hochverfügbaren Diensten schneller werden lässt. Die Suche nach niederverfügbaren Diensten ist dabei ohnehin deutlich schneller. Somit ist das Verteilungsbestimmungsverfahren PICav+ gegenüber kombinatorisch arbeitenden Ansätzen zu empfehlen.

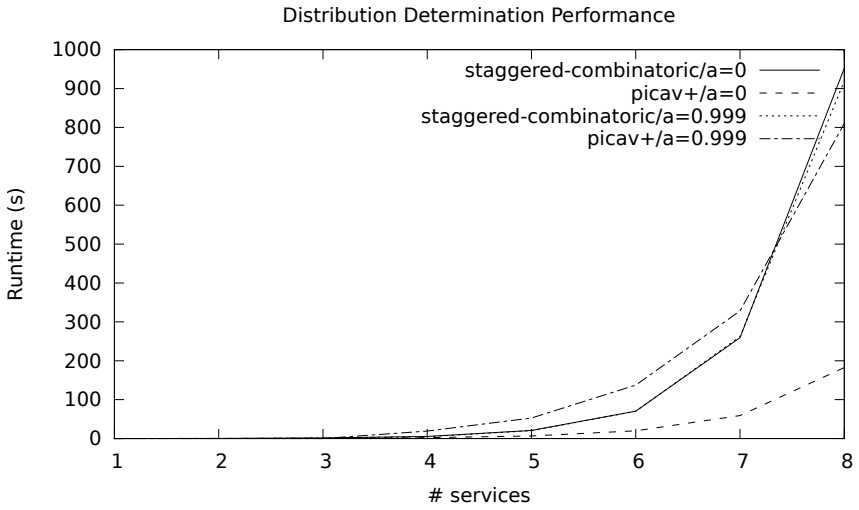

Abb. 6.26 Laufzeitverhalten der beiden Verfahren und Existenz eines Sweetspots

6.4 Experimentelle Validierung der Datenverarbeitung

In diesem Abschnitt erfolgt eine Evaluierung der Suche über dispergierte und redundante Daten aus Abschnitt 3.4.1.

6.4.1 Experimente zur Verarbeitung dispergierter Daten

Das erste Laufzeitexperiment betrachtet einen einfachen aggregierenden Funktionsaufruf auf dispergierte Textdaten. Abbildung 6.27 vergleicht die Zählung aller Unicode-Zeichen außerhalb des ASCII-Bereichs sowohl in der ursprünglichen Zeichenkette als auch auf dem höchstsignifikanten Fragmentstrom. Dieser ist durch eine kürzere Länge in Bytes und durch eine höhere funktionsrelevante Informationsdichte gekennzeichnet. Dies führt insgesamt zu einem messbaren und längenunabhängigen Geschwindigkeitsgewinn von ca. $13,2\%$.

6.4.2 Experimente zur Verarbeitung redundanter Daten

Dieser Abschnitt untersucht die Brauchbarkeit redundanter Daten nicht nur als Voraussetzung für die Wiederherstellung nicht verfügbarer signifikanter Daten, sondern auch als Grundlage für weitere Operationen. Hierzu wird die Bitsplitting-Kodierung mit einer einfachen XOR-Redundanz eingesetzt. Als Operation wird die Volltextsu-

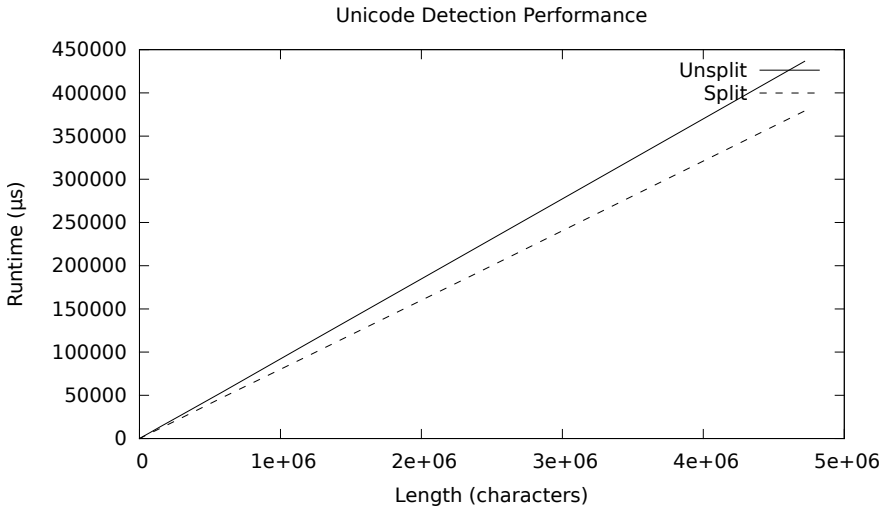

Abb. 6.27 Geschwindigkeitsvergleich der Unicode-Erkennungs-Routine auf Basis von Fragmentvergleichen

che nach Textdaten in Anlehnung an vorherige Untersuchungen zur Textsuche auf dispergierten Daten [SS14a] analysiert.

Abbildung 6.28 zeigt den charakteristischen Kurvenverlauf der Uniformitätsverteilung der XOR-Verknüpfung zweier Zahlen zwischen 0 und einem Maximum von 0 bis 128. Hierbei fällt auf, dass die redundanten Werte von Zahlen des Musters 2^i exakt gleich verteilt sind. In diesen Fällen ist es nicht möglich, aufgrund statistischer Textmerkmale vom Redundanzwert auf die beiden Ausgangswerte zu schließen. Von diesen Extrema abgesehen erscheint ein solcher Rückschluss jedoch möglich.

Abbildung 6.29 zeigt einen Ausschnitt aus der vorherigen Grafik für die Wertebereiche 65...90 und 97...122, die im ASCII-Zeichensatz den Buchstaben $A-Z$ und $a-z$ entsprechen. Die fokussierte Ansicht stützt die vorherige Beobachtung.

Werden nun konkret XOR-Redundanzen zwischen ASCII-Zeichen gebildet, so erhält man die in Abbildung 6.30 enthaltenen Verteilungen. Die Kurve beginnt bei $x = 0$, da $AXORA = 0$, $BXORB = 0$ etc. gelten. Auffällig ist, dass die großen Wertunterschiede aus den vorherigen beiden Diagrammen in eher kleinere Nuancen zwischen den Werten 20 und 26 überführt sind. Wird je ein Klein- und ein Großbuchstabe für die XOR-Operation eingesetzt, bleibt die Verteilung exakt gleich, wird jedoch um die Differenz $97-65 = 32$ auf der x-Achse in positiver Richtung verschoben.

Für praktische Belange muss jedoch die Buchstabenhäufigkeitsverteilung berücksichtigt werden. Exemplarisch soll der Originaltext von *Marco Polo* dazu dienen, sowohl die Ambiguität von Redundanzwerten als auch deren Häufung zu untersuchen. Das Ergebnis ist in Abbildung 6.31 zu sehen. Auf der x-Achse sind alle Redundanzwerte im Bereich 0...255 eingetragen. Auf der x-Achse lassen sich Ab-

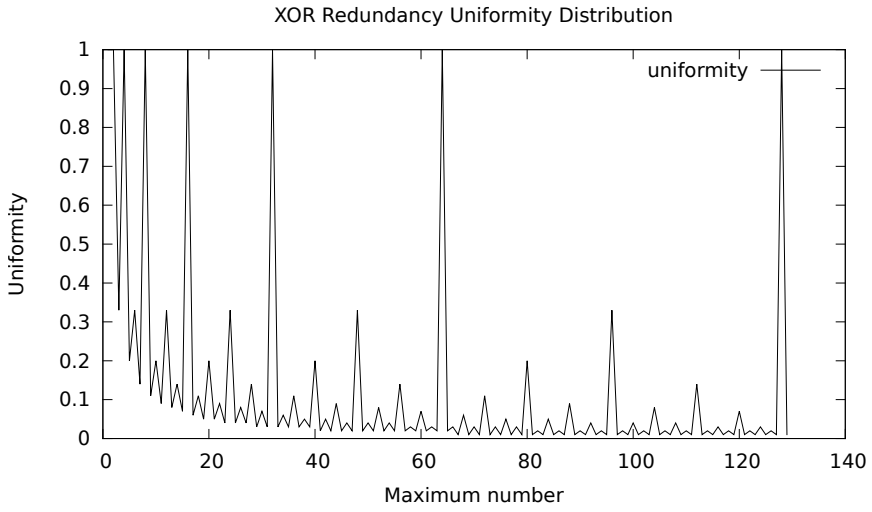

Abb. 6.28 Generelle Verteilung redundanter Werte

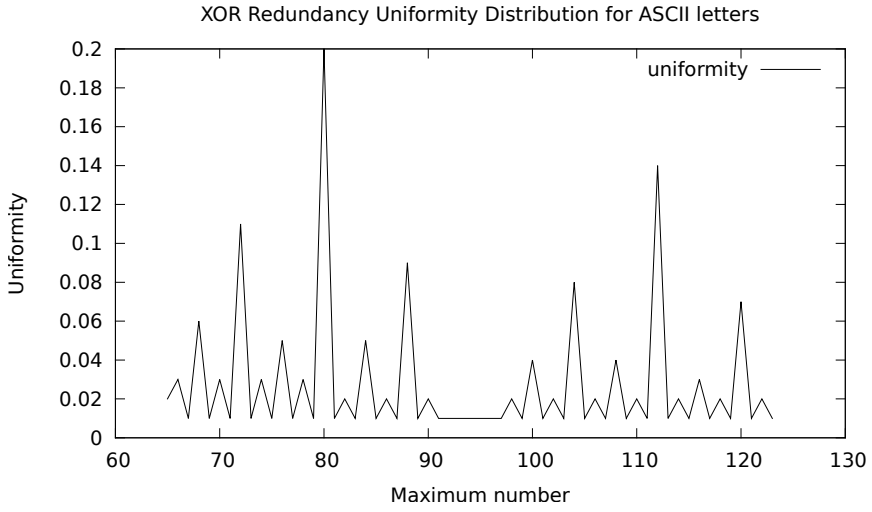

Abb. 6.29 ASCII-bezogene Verteilung redundanter Werte

schnitte bilden, in denen entweder Ambiguität und Häufung korrelieren, oder in denen die Häufung so gering ist, dass die stellenweise hohe Ambiguität kaum Einfluss nimmt, oder in denen die Häufig bei relativ niedriger Ambiguität sehr hoch ausfällt.

Der Praxistest erfolgt gemäß Abbildung 6.32. Hierbei werden alle Wörter von mindestens fünf Buchstaben Länge aus dem Text extrahiert und als Suchbegriffe

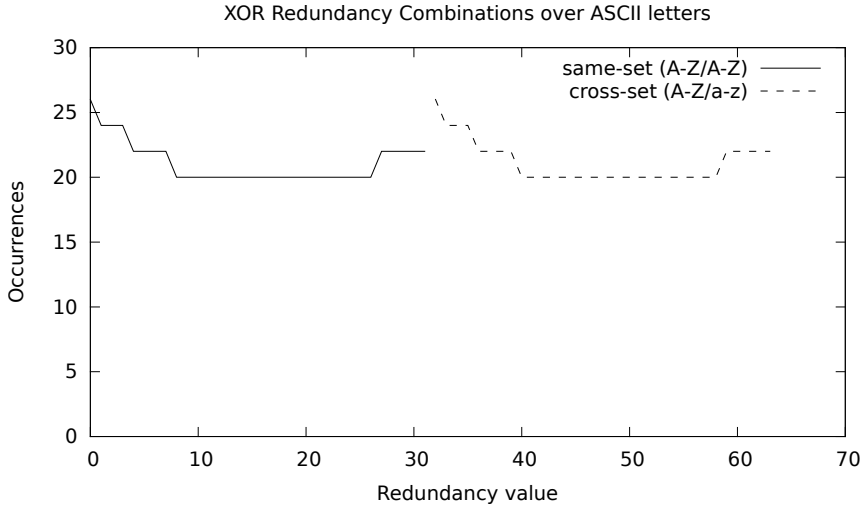

Abb. 6.30 ASCII-Redundanzkombinationen

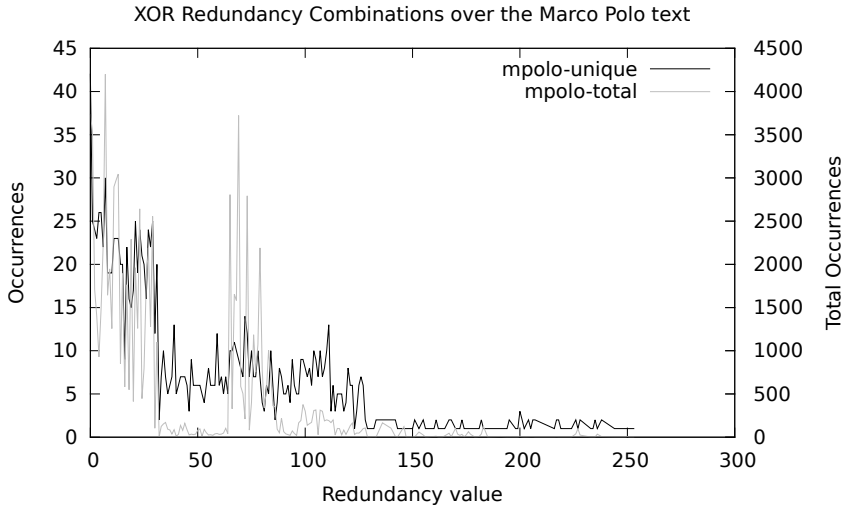

Abb. 6.31 Redundanzmetriken zum Beispieltext

eingesetzt. Abhängig von der Wortlänge ist eine durchschnittliche Suchpräzision erzielbar, die von 0% mit vielen falsch-positiven Ergebnissen, also unpräzise, bis zu 100% ohne falsch-positiv Ergebnisse, also maximal präzise, reicht. Falsch-negative Ergebnisse treten nicht auf. Damit liegt der Recall bei 1 und zusammen mit einer client-seitigen Nachfilterung kann eine vollständige und präzise Suche ermöglicht werden. Wörter ab 8 Buchstaben Länge können mit durchschnittlich über 60% Prä-

zision gefunden werden. Ab 10 Buchstaben steigt der Wert auf über 80%, ab 16 Buchstaben auf 100%. Lediglich im Fall von 17 Buchstaben ist ein Wert nicht gefunden worden, was das Ergebnis an dieser Stelle über den zulässigen Wert hinaus verfälscht hat.

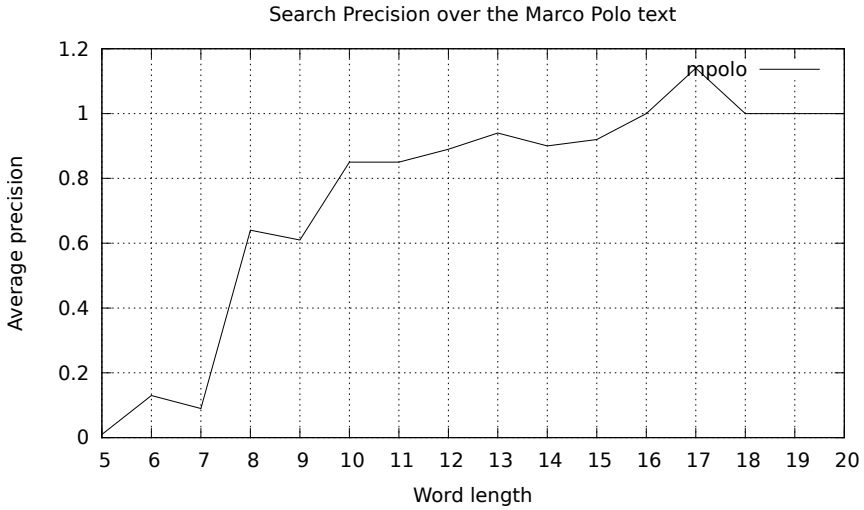

Abb. 6.32 Suchpräzision über die aus dem Beispieltext gebildeten redundanten Daten

6.5 Funktionstüchtigkeit und Eigenschaften der Speicherdienstanbindung

In diesem Abschnitt beginnt die Evaluierung der vorgeschlagenen und umgesetzten Schnittstellen auf Plattformebene zwischen Anwendungen und Anwendungsdiensten einerseits und Ressourcendiensten andererseits. Zuerst wird die Übertragung von Daten an Speicherdienste mit unterschiedlichen Transportoptimierungen evaluiert.

6.5.1 Beschreibung der Software

Im Abschnitt 5.1 wurden unterschiedliche Cloud-Speicherdienste vorgestellt. Zu deren übergreifenden Nutzung ohne dedizierte anbieterabhängige Software ist es notwendig, Anwendungen eine generische Datentransportschicht zur Verfügung zu stellen. CloudFusion, ein virtuelles Dateisystem, wurde speziell für diesen Ein-

satzzweck als Datentransportmodul konzipiert. Eine detaillierte Beschreibung der Software findet sich im Anhang B. Vergleichend werden die protokollspezifischen Transportmodule FuseDAV und Google Cloud Storage (gsutil) eingesetzt.

6.5.2 Experimente

Der Datendurchsatz über unterschiedliche Transportmechanismen wird über den Dateitransfer durch drei Speicherflussketten experimentell geprüft. Dateien in der ersten Kette werden direkt auf ein verschlüsseltes lokales Dateisystem gespeichert. In der zweiten Kette wird das FUSE-Modul FuseDAV genutzt, um einen WebDAV- oder HTTP-Server anzusprechen, welcher seinerseits die Daten auf ein unverschlüsseltes Dateisystem ablegt. In der dritten Kette wird CloudFusion anstelle von FuseDAV sowohl mit als auch ohne Protokollierungsoption (Logging) sowie sowohl mit als auch ohne Partitionierung der Dateien (Chunking) eingesetzt. Als Endpunkte der zweiten und dritten Kette werden WebDAV-Server auf dem lokalen System, im LAN verbunden per Ethernet sowie im Internet verbunden einmal per WLAN (IEEE 802.11g mit Linkqualität $\sim \frac{53}{70}$) und DSL-2000 und einmal per 1000baseT-Ethernet und DFN-Wissenschaftsnetz (Glasfaserplattform X-WiN mit Paketlaufzeit $< 10\frac{ns}{km}$) eingesetzt. Desweiteren werden in der dritten Kette Online-Speicherdienste mit anbieterspezifischen Protokollen zusätzlich zu WebDAV eingesetzt. Die für das Experiment ausgewählten Protokolle sind allesamt HTTP-basiert, um eine gewisse protokollübergreifende Vergleichbarkeit dennoch gewährleisten zu können.

Insgesamt umfasst das Experiment 1986 Einzelmessungen, deren Wert sich als arithmetisches Mittel über jeweils 10 Messpunkte zusammensetzt. Die Einzelmessungen werden wiederum in Gruppen zu 19 bis 49 Messungen insgesamt 54 Konfigurationen aus Speicherketten und Datencharakteristik zugeordnet, wobei fünf Charakteristiken zur Verfügung stehen. Jede Messung beinhaltet einen oder mehrere Uploads. Die insgesamt übertragene Datenmenge beträgt 22,22 GiB.

Tabelle 6.1 enthält die ausgemessenen Speicherflussketten ausgehend von einem Ursprungssystem ORIGIN zu einem Speicherdienst SERVICE (bzw. im lokalen Netz TARGET) mit Zugriffsprotokoll PROTOCOL über ein virtuelles Dateisystem VFS. Als Ausgangssystem wird hierbei stets Rumba genutzt, als lokales Zielsystem Bomba. Die Speicherdienste im Experiment umfassen T-Online Mediencenter, GMX Media Center, Dropbox und Google Storage.

Tabelle 6.2 enthält die dazugehörigen Charakteristiken CHAR, welche in Anlehnung an unterschiedliche Nutzungsmuster von Speicherdiensten entworfen sind.

Die Abbildungen 6.33, 6.34, 6.35 und 6.36 zeigen das unterschiedliche Verhalten sowohl der lokalen Speicherflussketten untereinander als auch einen stark reduzierten Durchsatz bei geringeren Blockgrößen.

In Abbildung 6.33 werden Dateien byteweise auf ein ext4-Dateisystem geschrieben. Der maximal erreichbare Durchsatz in der ersten Kette, wobei das Dateisystem mit AES256 im Modus `cbc-essiv:sha256;sha1` verschlüsselt ist, beträgt 865,04 kiB/s. Der Durchsatz durch die virtuellen WebDAV-Dateisysteme in-

Tabelle 6.1 Speicherflussketten

Bezeichnung	Erläuterung	Aufbau der Speicherkette
direct	lokaler direkter Festplattenzugriff	ORIGIN-fs:cryptext4
local	lokaler Speicherdienst	ORIGIN-VFS-webdav-fs:ext4
lan	Speicherdienst im LAN	ORIGIN-VFS-webdav-ethernet--TARGET-fs:cryptext4
internet	Speicherdienstzugriff über DSL	ORIGIN-VFS-PROTOCOL-wlan-dsl--SERVICE-fs:unknown
fastnet	Speicherdienstzugriff über DFN	ORIGIN-VFS-PROTOCOL-ethernet-dfn--SERVICE-fs:unknown

Tabelle 6.2 Transfercharakteristiken

Bezeichnung	Iterationen i	Blockgröße	Dateigröße	Uploads pro Iteration
direct	40-50	1 Byte	$1{,}5^i$ Byte	1
atomic	50	1 Byte/4 kiB	$\lceil 1{,}5^i * 2^{12} \rceil$ Byte	1
directlarge	30-50	1 MiB	$\lceil 1{,}5^i * 2^{20} \rceil$ Byte	1
multiplesmall	30-40	1 Byte	1 Byte	i
multiple	20-50	1 MiB	1 MiB	i

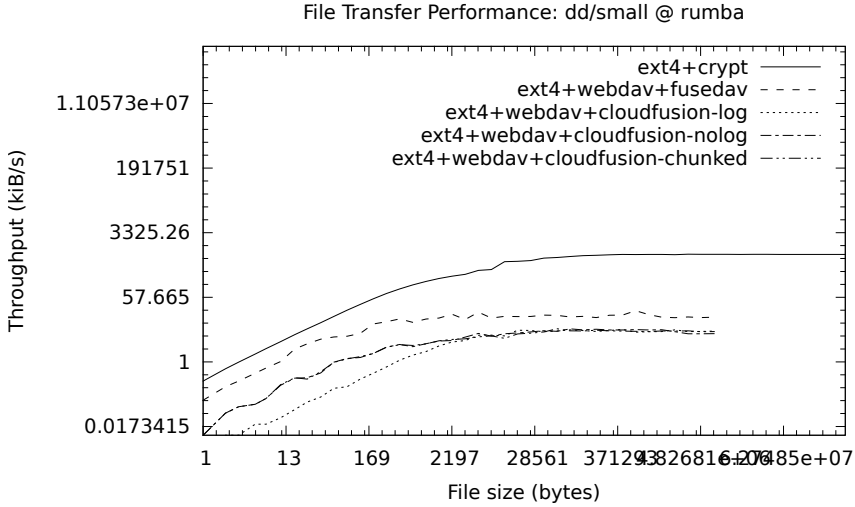

Abb. 6.33 Datentransferraten mit 1-Byte-Inkrementen (logarithmische Skalen)

klusive der lokalen Netzwerkübertragung (über die *lo*-Schnittstelle mit der IPv4-Adresse 127.0.0.1) liegt deutlich darunter. CloudFusion ist nicht für byteweise Schreibvorgänge optimiert und erreicht insbesondere bei Nutzung der Loggingfunktion maximal 7,64 kiB/s gegenüber 19,82 kiB/s bei FuseDAV. Es sei bemerkt, dass die WebDAV-Implementierung (Apache httpd 2.2 mit Modul *mod_dav*) zu jedem Schreibvorgang auf das virtuelle Dateisystem einen ebensolchen in eine temporäre Datei vornimmt, die nur bei Beendigung des Schreibvorgangs über WebDAV

sichtbar wird, so dass zusammen mit der Netzwerkübertragung ein, sofern nicht optimiert wird, theoretischer Maximaldurchsatz von etwa einem Drittel des direkten Festplattenzugriffs erzielt werden kann.

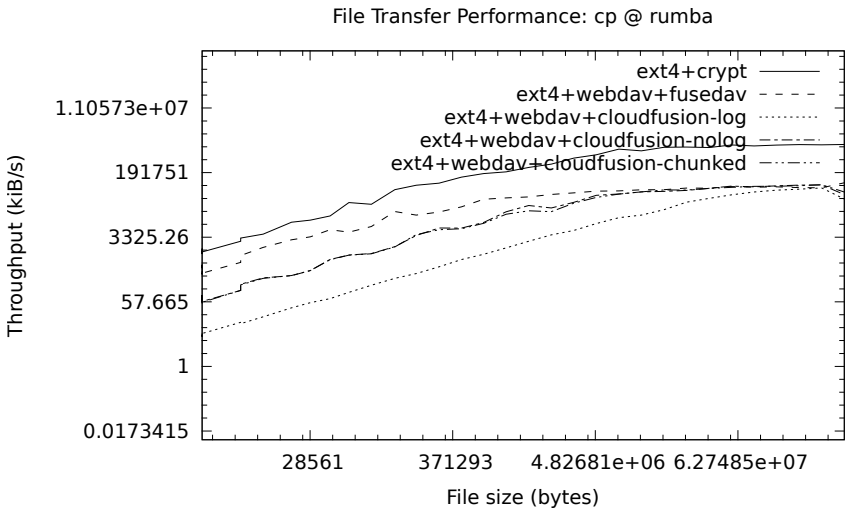

Abb. 6.34 Datentransferraten mit der Standardblockgröße von 4KiB (logarithmische Skalen)

Beträgt die Blockgröße 4 KiB, ändert sich das Übertragungsverhalten durch eine Konvergenz der Durchsätze bei einer Dateigröße von ca. 1 MiB. Steigt die Dateigröße weiter an, so ist sinkt zudem der Störeinfluss der Logging-Funktion von CloudFusion, wie aus Abbildung 6.34 ersichtlich wird.

Eine zunehmende Erhöhung der Blockgröße führt zu einem relativ höheren Durchsatz bei CloudFusion, wie in Abbildung 6.35 für die Größe 1 MiB gezeigt wird. Hierbei weist der Durchsatz von FuseDAV die höhere Konstanz, jedoch nicht mehr die höheren Absolutwerte auf.

Werden hingegen keine einzelnen Dateien mit wachsender Größe, sondern eine wachsende Zahl an Dateien mit konstanter Größe zu je 1 MiB geschrieben, stellt sich die Implementierung von FuseDAV wieder wie in Abbildung 6.36 zu sehen vorteilhaft dar. Dabei sei angemerkt, dass eine kleinere konstante Durchsatzverringerung auf die Implementierungsunterschiede zurückzuführen ist und demnach eine Reimplementierung von CloudFusion in C analog zu FuseDAV eine Verbesserung erzielen dürfte.

Unter realistischen Bedingungen, in diesem Fall einer Übertragung der Dateien über eine DSL-768k-Verbindung an einen Anbieter mit WebDAV-Schnittstelle im Internet, mit und ohne Betrachtung der Uploadzeiten, erkennt man deutlich die Vorteile der CloudFusion-Architektur.

Abbildung 6.37 zeigt, wie zwar die Übertragungszeiten beider Module übereinstimmen, jedoch der initiale Schreibvorgang in den lokalen Cache von Cloudfusion

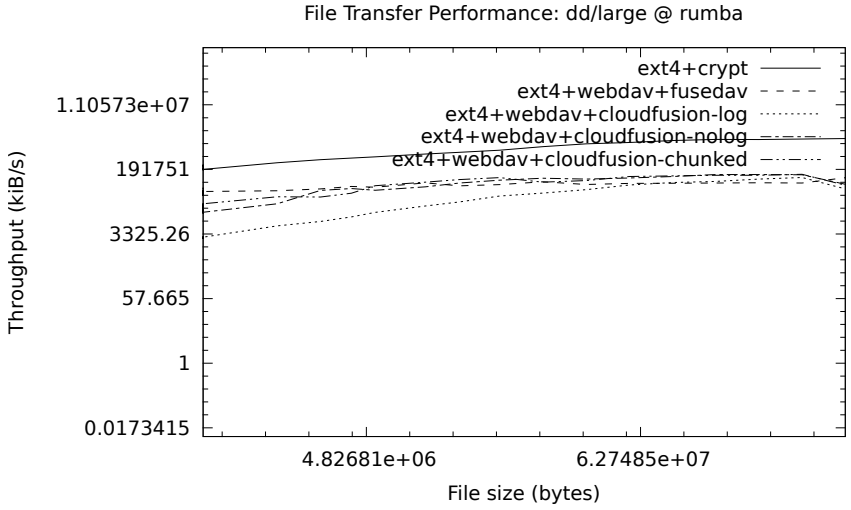

Abb. 6.35 Datentransferraten mit der bevorzugten Blockgröße von 1MiB (logarithmische Skalen)

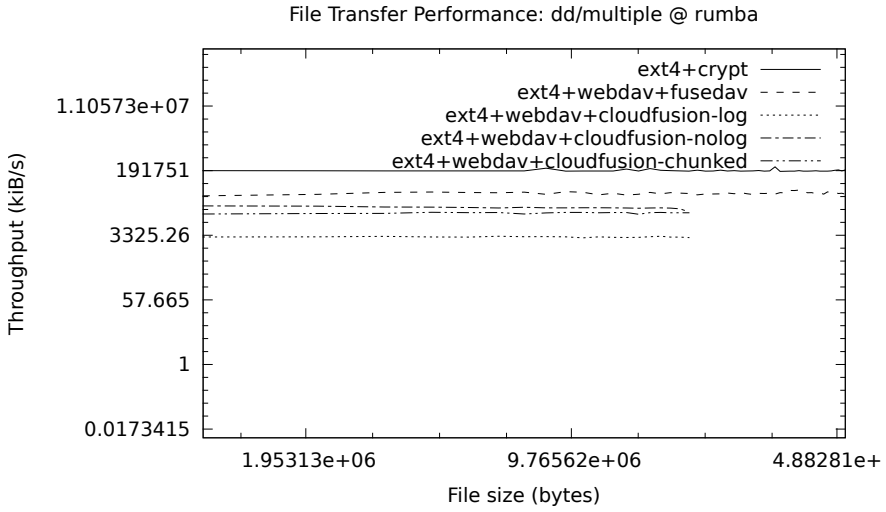

Abb. 6.36 Datentransferraten mit wachsender Dateianzahl zu je 1MiB (logarithmische Skalen)

eher zur Anwendung zurückkehrt und aus Sicht des Anwenders damit einen höheren Durchsatz erzielt. Angemerkt sei dabei, dass beide Module keine vollständig synchronen Schreibzugriffe (mit der Option `O_SYNC` bzw. `iflag=sync`) unterstützen und ohne diese Optionen keine Transaktionszusagen verletzt werden. Im günstigsten Fall, der Übertragung einer Datei, liegt das Verhältnis der lokalen Schreib-

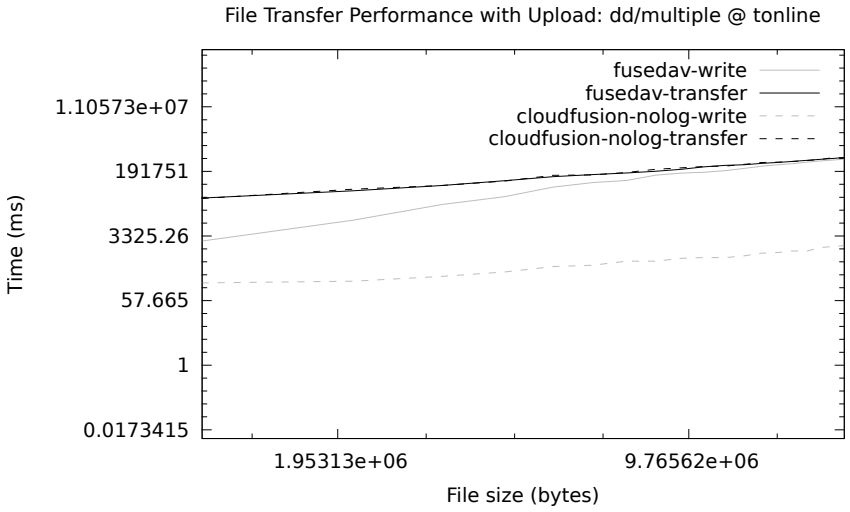

Abb. 6.37 Datentransferzeiten mit wachsender Dateianzahl zu je 1MiB (logarithmische Skalen) auf den Speicherdienstanbieter T-Online

zeiten bei 61,65s zu 2430,87s (d.h. 1:39,4), im ungünstigsten Fall, der Übertragung von 19 Dateien, bereits bei 1550,52s zu 416873,20s (d.h. 1:268,9).

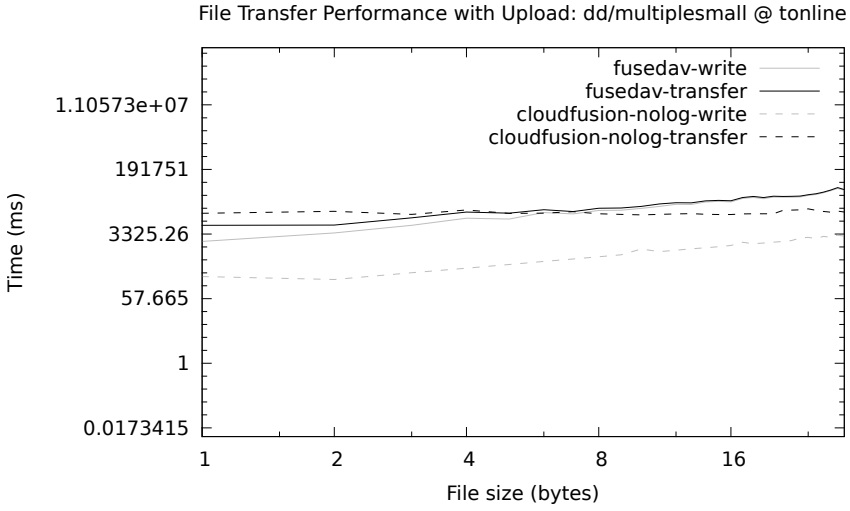

Abb. 6.38 Datentransferzeiten mit wachsender Dateianzahl zu je 1 Byte (logarithmische Skalen) auf den Speicherdienstanbieter T-Online

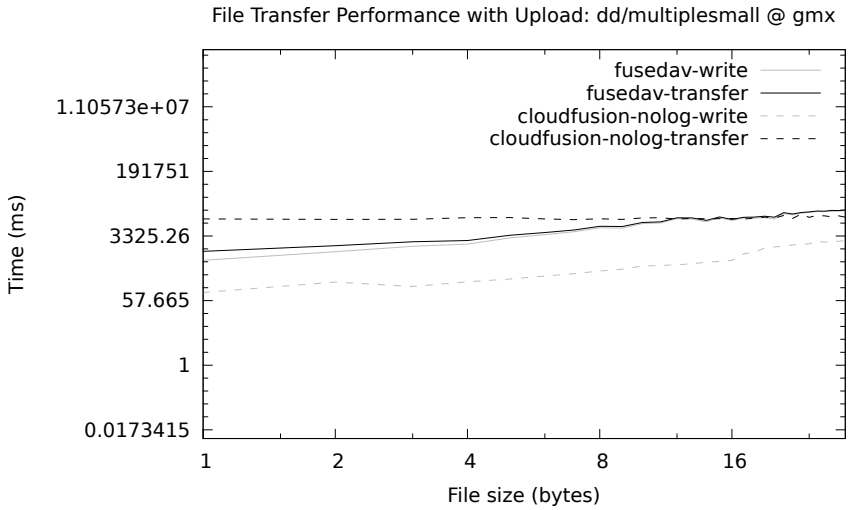

Abb. 6.39 Datentransferzeiten mit wachsender Dateianzahl zu je 1 Byte (logarithmische Skalen) auf den Speicherdienstanbieter GMX

Werden viele kleine Dateien übertragen, ein in der Praxis speziell auf mobilen Geräten häufiger Fall, wirkt sich die CloudFusion-Architektur nicht nur auf lokale Schreibzugriffe, sondern sogar auch auf die Netzwerkübertragung vorteilhaft aus. Abbildung 6.38 zeigt, wie bei dem Zugriff auf T-Online im günstigsten Fall 13238,46s gegenüber 54374,63s mit FuseDAV erzielt werden. In ähnlicher Form sind die Ergebnisse bei GMX vorzufinden, wie Abbildung 6.39 belegt, wobei hier der Schnittpunkt eine höhere Zahl an Dateien erfordert und der Unterschied zwischen beiden Modulen weniger stark ausgeprägt ist.

Schließlich soll noch der direkte Vergleich von Anbietern berücksichtigt werden. Abbildung 6.40 vergleicht den Zugriff bei maximal für das Experiment erzielbarer Bandbreite über die DFN-Verbindung zu den Anbietern T-Online, GMX und Dropbox mit verschiedenen Zugriffsmodulen. Die niedrigste Transferzeit wird für eine geringe Zahl an Dateien mit GMX erreicht, wobei dieser Anbieter gleichzeitig eine hohe Aufruffehlerquote aufweist, die jedoch der aufrufenden Anwendung aufgrund der Nutzung des virtuellen Dateisystems verborgen bleibt.

Zu beachten ist bei der Interpretation der Diagramme, dass die Konfiguration von Cloudfusion eine Uploadverzögerung von 6s für Daten und 12s für Metadaten vorsieht, welche in den Diagrammkurven enthalten sind. Eine Reduktion der Verzögerung verschiebt den Schnittpunkt nach links. Weiterhin verursacht Cloudfusion aufgrund der Python-Ummantelung der FUSE-Schnittstelle per se einen zusätzlichen Overhead, der ebenfalls im Fall einer Reimplementierung ohne Ummantelung eine weitere Optimierung zur Folge hätte.

Wenn mehrere kleinvolumige Schreibzugriffe durch verzögerte Ausführung einem größeren Datenpaket gebündelt werden, sinkt der Einfluss der Latenz und somit

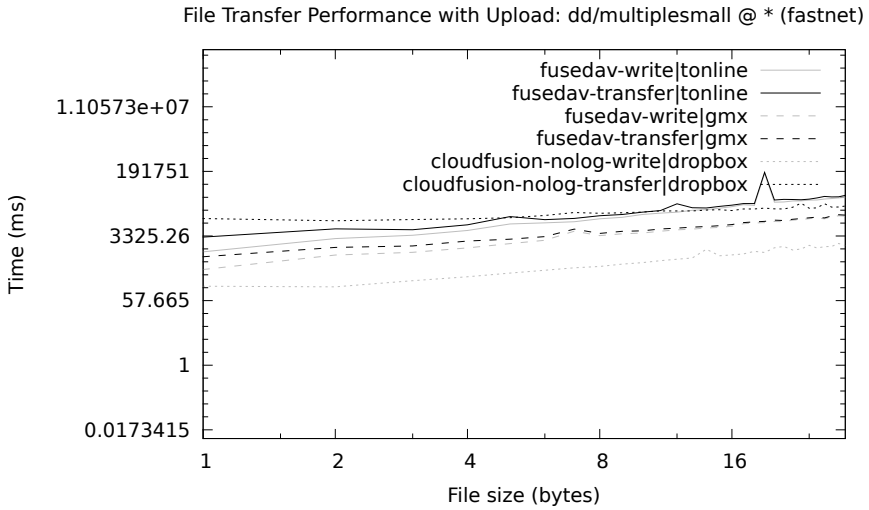

Abb. 6.40 Datentransferzeiten mit wachsender Dateianzahl zu je 1 Byte (logarithmische Skalen) auf mehrere Speicherdienstanbieter über DFN

auch die Übertragungszeit insgesamt signifikant ab. Der Effekt wird in Abbildung 6.41 deutlich. Diese als Batching, Bulking oder im Diagramm als Reverse-Chunking bezeichnete Technik ist in Cloudfusion enthalten, in FuseDAV hingegen nicht. Im Experiment wird das unterschiedliche Verhalten durch Injektion künstlicher Netzwerkverzögerungen mit dem Werkzeug *netem* provoziert und führt zu nahezu konstanter Übertragungszeit für Cloudfusion gegenüber nahezu linear anwachsender Übertragungszeit für FuseDAV.

Das Transportmodul `gsutil`, zugehörig zum Speicherdienst Google Cloud Storage, beinhaltet bereits diverse Optimierungen, unter anderem die Bündelung mehrerer Schreibzugriffe in einem Sammelzugriff sowie die Zerlegung großer Dateien während des Schreibvorgangs.

Abbildung 6.42 stellt das Verhalten von gsutil bei der Übertragung vieler kleiner Dateien mit je einem Byte Größe dar. Das Diagramm verdeutlicht, wie die Option *copy at once*, welche der Übertragung mehrerer Dateien mit einem Aufruf des Moduls entspricht, gegenüber der mehrfachen Modulausführung deutlich die Ausführungszeiten verringert, da der zeitintensive Modulstart nur einmal erfolgt. Eine weitere deutliche, wenn auch weniger stabile, Optimierung stellt die Option *combined* dar, welche die kleinen Dateien vor dem Schreibvorgang in einer großen Datei bündelt. Nachteilig ist wiederum, dass diese Dateien nicht ohne weiteres direkt weiterverarbeitet werden können.

Abbildung 6.43 stellt analog das Verhalten von gsutil mit größeren Dateien zu je einem MB Größe dar. Hierbei fällt auf, dass sämtliche Optimierungen kaum noch einen Einfluss auf die gesamte Übertragungszeit haben. Auch die zusätzliche Option *parallel* für parallele Chunk-Uploads führt zu keinen Änderungen. Teilweise,

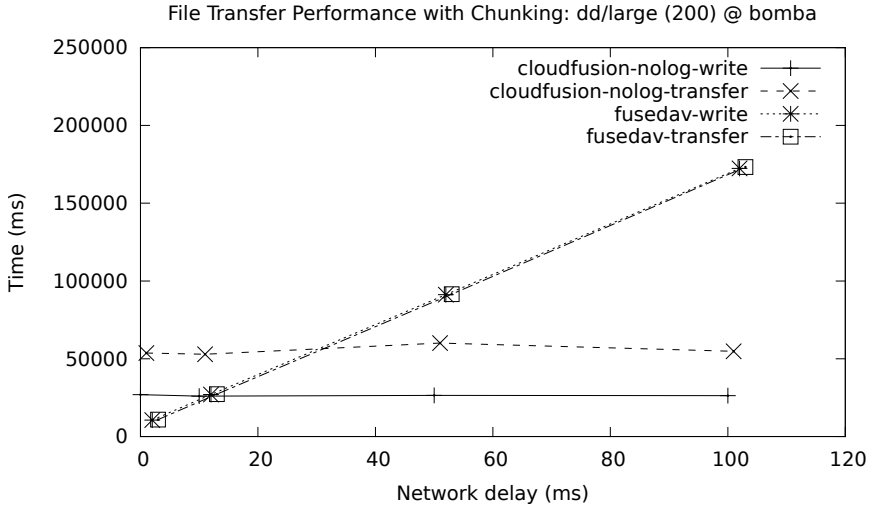

Abb. 6.41 Datentransferzeiten mit wachsender Netzwerkverzögerung

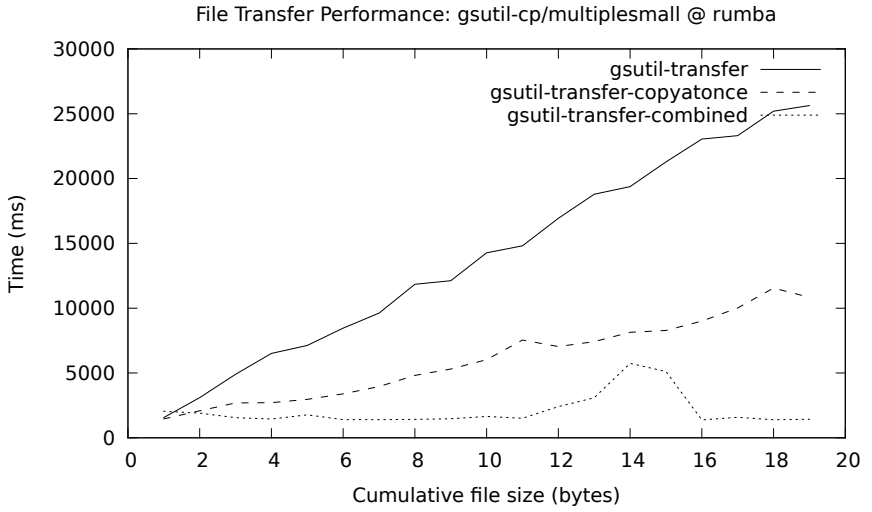

Abb. 6.42 Datentransferzeiten mit wachsender Dateianzahl zu je 1 Byte auf den Speicherdienst GCS

vor allem für kleinere Übertragungsgrößen, führt die Option *copy at once* sogar unerwünscht zu einer höheren Laufzeit.

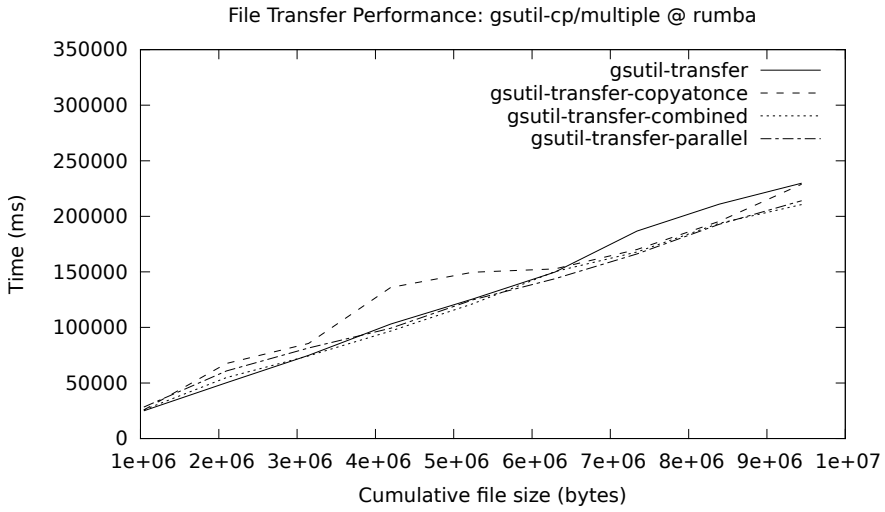

Abb. 6.43 Datentransferzeiten mit wachsender Dateianzahl zu je 1 MiB auf den Speicherdienst GCS

6.6 Funktionstüchtigkeit und Eigenschaften der Speicherdienstintegration

Die nutzerkontrollierte Kombination und Integration mehrerer Cloud-Speicherdienstanbieter erfordert Werkzeuge, welche die Auswahl, Verknüpfung, Konfiguration und Nutzung der Anbieter ermöglicht. NubiSave bietet sowohl ein virtuelles Dateisystem für Linux-Betriebssysteme mit integriertem Splitter zur Aufteilung der Daten als auch einen graphischen Speicherflusseditor [SMS13, SBM+11] mit assoziierten Werkzeugen für deren Modifikation und Ausführung. NubiVis ergänzt die Software um eine Visualisierung der Aufteilung, der Speicherorte und weiterer Attribute aller Daten [STS14]. NubiDroid bietet schließlich eine alternative Umsetzung einer Speicherdienstintegration für Android-Plattformen. Die Softwarelösungen sind im Anhang B umfänglich beschrieben. In den folgenden beiden Abschnitten wird die Speicherdienstintegration experimentell ausgewertet, wofür zuerst eine Beschreibung der Architektur von NubiSave durchgeführt wird, bevor anschließend dessen Laufzeiteigenschaften gemessen, gesammelt und ausgewertet werden. Die Experimente orientieren sich am peaCS-Framework zur Analyse des Laufzeitverhaltens und der Effizienz von Cloud-Storage-Integratoren [SQFS13].

6.6.1 Beschreibung der Software

NubiSave ist im engeren Sinne ein dispergierendes Dateisystem (NubiSave-Splitter) auf der Basis von FUSE. Dateisystemaufrufe, die über eine POSIX-Schnittstelle eintreffen, werden per Multiplexing auf mehrere Speicherorte verteilt oder mithilfe der bei Speicherung und Abruf aufgezeichneten Metadaten beantwortet.

NubiSave ist im weiteren Sinne auch eine Orchestrierung von Modulen, die als Dateisysteme repräsentiert sind, und den dazwischen existierenden Verbindungen im Sinne einer software-definierten Speicherinfrastruktur. Es bildet somit das Modell der Speicherflüsse auf Betriebssystemstrukturen ab (NubiSave-Storage-Flows). Jedes Modul weist bis zu drei Verzeichnisse auf: `data` für den bidirektionalen Datenaustausch, `config` zum Schreiben der Konfiguration und `stats` zum Auslesen von Laufzeitstatistiken und Statusinformationen. Umgesetzte Module umfassen lokalen Verzeichnisse und Speicherprotokolle als Terminierungsmodule und Deduplizierung und Verschlüsselung als Durchsatzmodule [SS13b].

Existierende FUSE-Dateisysteme lassen sich nur vor der Aktivierung konfigurieren und unterstützen keine Statistiken per Dateisystemzugriff. Hingegen nutzen NubiSave und Cloudfusion die Struktur aus. Sie lassen sich deshalb zur Laufzeit flexibel mit Einstellungsdateien im Ini-Format umkonfigurieren. Dies betrifft sowohl die eigene Konfiguration als auch im Fall von NubiSave die Anbindung von Speicherzielen.

Die Standardkonfiguration des NubiSave-Splitters beinhaltet eine Aufteilung der Daten bei $n = 2$ und eine zunehmende Kodierung von Redundanzen für $n > 2$. Zudem beträgt die angenommene Verfügbarkeit des Verzeichnismoduls $a_i = 0,8$. Unter diesen Annahmen gilt für ein gregorianisches Kalenderjahr mit 365,2425 Tagen die Fragmentbildung und Verfügbarkeit gemäß Tabelle 6.3.

Tabelle 6.3 Dispersion und Verfügbarkeiten in NubiSave bei $a_i = 0,8$

n	k	m	a	Jahresausfallzeit
1	1	0	0,8	73d 01h:09m:50s
2	2	0	0,64	131d 11h:41m:42s
3	2	1	0,896	37d 23h:38m:43s
4	2	2	0,9728	9d 22h:25m:49s
5	2	3	0,9933	2d 10h:54m:22s
6	2	4	0,9984	14h:01m:31s
7	2	5	0,9996	03h:15m:13s

6.6.2 Experimente

Das erste Experiment vergleicht in Anlehnung an die Untersuchungen zum Datentransfer die Kodierungslaufzeiten von NubiSave in Verbindung mit einer Menge

lokaler Verzeichnisse als Speicherziele. Es werden dabei sowohl direkte Schreibzugriffe mit Vielfachen von $1kB$-Blöcken als auch atomare Kopieraufrufe mit einer Blockgröße von $4kB$ durchgeführt. Vergleichend werden die gleichen Schreibzugriffe auf das ext4-Dateisystem, welches NubiSave zugrundeliegt, nachgebildet. Alle Kombinationen von 1-20 Diensten, repräsentiert durch Verzeichnisse, und Dateigrößen von 2^1-2^{20} kB werden evaluiert. Insgesamt werden pro Diagramm 95,5 GB Daten an NubiSave übertragen, aufgeteilt, mit SHA-256-Prüfsumme versehen und in die angeschlossenen Speicherziele geschrieben.

Abbildung 6.44 enthält die Ergebnisse. Man erkennt für große Dateien ab 100 MB bis zu 1 GB die mit NubiSave deutlich erhöhte Laufzeit.

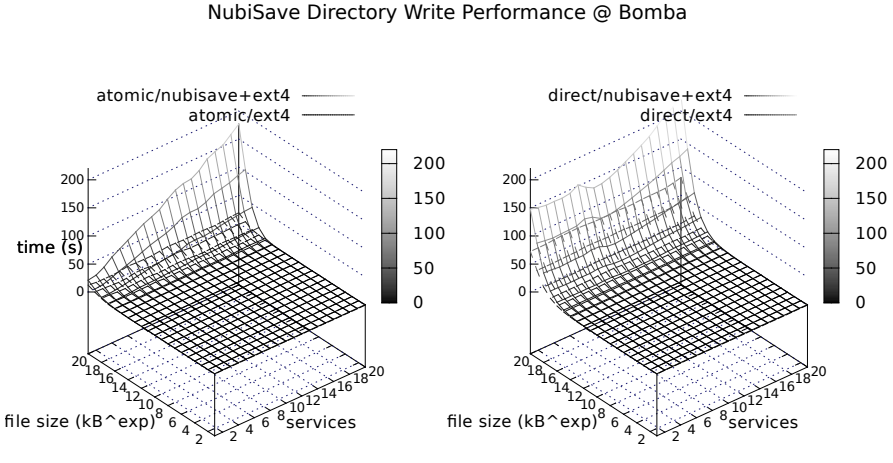

Abb. 6.44 Kodier- und Transferzeiten auf NubiSave mit angebundenen lokalen Verzeichnissen

Ergänzend soll eine experimentelle Evaluierung einer komplexeren Speicherkette basierend auf einer Aufteilung mit NubiSave-Splitter, einer Inhaltsverschlüsselung mit EncFS und einer transportverschlüsselten Speicherung über das SSH-Protokoll durchgeführt werden. Quelltext 6.1 gibt die wesentlichen Ablaufschritte wieder, welche durch das Experimentierskript ausgeführt werden. Werden die Parameter wie Benutzername, Passwort, Hostname und Zielverzeichnis für SSH und symmetrischer Schlüssel für EncFS nicht spezifiziert, erfolgt eine interaktive Abfrage.

Die Ergebnisse sind, wie in Abbildung 6.45 gezeigt, in der Form ähnlich wie die vorherigen. Allerdings sind die Absolutbeträge für die benötigte Laufzeit deutlich höher. Für den direkten Schreibzugriff ergibt sich ein Verzögerungsfaktor von ca. $3,47..3,49$ für Datenmengen $\geq 500kB$, für das atomare Schreiben ein höherer von $4,12..4,52$. Allerdings liegen beeinflusst durch das Experimentierverfahren auch die Vergleichsgrößen des Schreibens auf das lokale ext4-Dateisystem zwar initial mit

Listing 6.1 Nubisave-Modulkonfiguration

```
\$ nubisave-module add ssh
   -> ssh492749M
\$ nubisave-module add encryption
   -> encryption190241M
\$ nubisave-module connect encryption190241M ssh492749M
\$ nubisave-module mount ssh492749M manual
\$ nubisave-module mount encryption190241M % durch hot-copy in
    das NubiSave-Speicherzielverzeichnis; impliziert connect
```

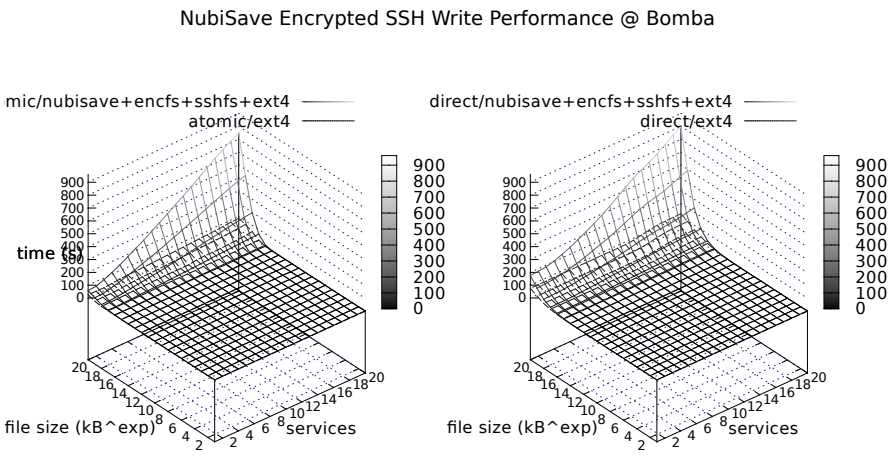

Abb. 6.45 Kodier- und Transferzeiten auf NubiSave mit komplexerer Speicherkette über EncFS und SSH

ca. 86*s* pro GB beieinander, ab 15 Speicherzielen wächst bedingt durch die insgesamt erhöhte Festplattenaktivität diese jedoch im Fall der komplexen Speicherkette auf 133*s* an. Somit ist ein Teil der Verzögerung dem Erreichen der Durchsatzgrenze auf dem Testsystem zuzuordnen.

6.7 Funktionstüchtigkeit und Eigenschaften der Datenverwaltung

Sollen Daten nicht in Form von Dateien, sondern aus der Anwendung heraus strukturiert abgelegt, und nicht nur abgerufen, sondern auch ausgewertet werden, so wird eine Datenbankschnittstelle benötigt. Mit StealthDB wurde eine Datenbank spezi-

ell für Cloud-Umgebungen so konzipiert, dass die Daten nutzerkontrolliert und sicher verteilt abgelegt und verarbeitet werden können. In diesem Abschnitt wird die StealthDB-Architektur beschrieben und deren Umsetzung experimentell ausgewertet.

6.7.1 Beschreibung der Software

StealthDB ist eine Datenbankverwaltungssoftware, welche die an sie übergebenen Daten und Datenströme spaltenorientiert und typgebunden verteilt über unterschiedliche Speicherbereichen ablegt und sie entweder in-situ auf den Speicherbereichen oder nach Abruf verarbeitet [SMS15, Spi15]. StealthDB setzt damit die im Abschnitt 4.4 dargestellten Plattformkonzepte um.

Die Software ist in ein anwendungsintegrierbares Modul zur Kapselung verschiedener Klassen (z.B. Tabelle und Spalte), ein auf Speicher- oder Recheninfrastrukturdienste installierbares Modul für ausgelagerte Operationen (StealthDB-Cloud), einen σ-SQL-Parser sowie eine Kommandozeilenschnittstelle aufgeteilt. Die Kommunikation zwischen den Modulen erfolgt über lokale Methodenaufrufe und entfernte Prozeduraufrufe.

Die Abbildung 6.46 beinhaltet die konkrete Softwarearchitektur von StealthDB. Die Implementierung basiert auf der Programmiersprache Python3 mit den Standard-Klassenmodulen sowie der zusätzlichen Bibliothek Pyro4 für die Umsetzung der Netzwerkkommunikation.

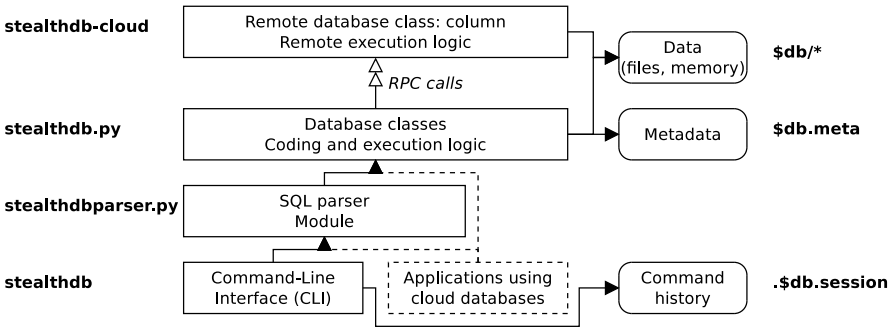

Abb. 6.46 Konkrete Architektur von StealthDB

Der Transport der Daten erfolgt entweder über eine RPC-Schnittstelle oder über eine optimierte Anbindung über ein Dateisystem, sofern ein Speicherdienstbetreiber eine dafür passende Schnittstelle wie WebDAV zur Verfügung stellt. Im zweiten Fall wird die Syntax `USE CLOUDS 'cloud://computecloud,file://storagecloud' ...` genutzt. Die Dateisystemschnittstelle kann hierbei durch CloudFusion im synchronen Übertragungsmodus oder durch ein anderes FUSE-Modul bereitgestellt werden. Sie kapselt dabei einen Speicherdienst, auf welchen auch der Verarbei-

tungsdienst Zugriff haben muss, um die Datenlokalität auszunutzen. Abbildung 6.47 zeigt die Topologie der Transportwege der Daten. Werden diese über die RPC-Schnittstelle an den StealthDB-Dienst übertragen, dann kann die Instanz der Cloud-Datenbank so eingerichtet sein, dass sie wiederum die Daten aufteilt und an weitere Speicherorte verteilt. Diese Rekursion ist aus vielen Gründen vorteilhaft [BS09].

Eine detaillierte Beschreibung der StealthDB-Software aus der Anwenderperspektive findet sich im Anhang B.

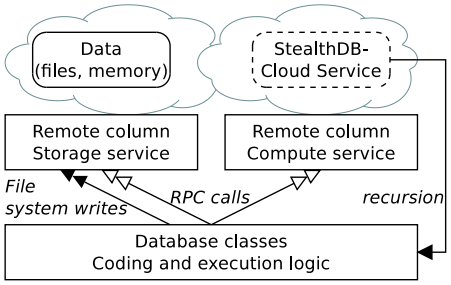

Abb. 6.47 Datentransporttopologie von StealthDB

6.7.2 Experimente

Die experimentelle Auswertung von StealthDB erfolgt anhand mehrerer lokaler und verteilter Abfragen. Im Verteilungsfall beeinflusst die Map-Carry-Reduce-Ausführung wesentlich das Laufzeitverhalten. Die Auswertung umfasst neben der Laufzeit auch die Korrektheit der Ergebnisse mit unterschiedlichen Kodierungen, beispielsweise Dispersion kombiniert mit homomorpher Verschlüsselung oder Replikation kombiniert mit ordnungserhaltender Verschlüsselung. Für die Durchführung der Experimente existiert zu StealthDB ein kombinatorisches Testskript, welches beliebige Kombinationen aus Kodierung, Zahl an Speicher- und Verarbeitungszielen und Verteilung automatisiert prüft. Desweiteren existieren dedizierte Testskripte, welche die Datenbankfunktionen als externes Programm oder als Python-Bibliothek nutzen.

Das erste Experiment vergleicht die Laufzeit von Einfüge- und Abfrageoperationen über eine unterschiedliche Anzahl und Zusammensetzung homogener Speicherbereiche. Es wird sowohl die softwareintern bestimmte Netto-Ausführungszeit der Aufrufe über die SQL-Syntax `EXPLAIN ANALYZE SELECT ...` als auch per externer Zeitmessung die Brutto-Ausführungszeit der StealthDB-Kommandozeilenanwendung gemessen.

Die Vierfachabbildung 6.48 enthält die Messergebnisse für Einfügeoperationen sowie mit deutlich verkürzter Skala für einfache, zuverlässigkeitsoptimierte und geordnete Abfragen. Die Daten sind dabei stets repliziert ($k = 1, m \geq 0$).

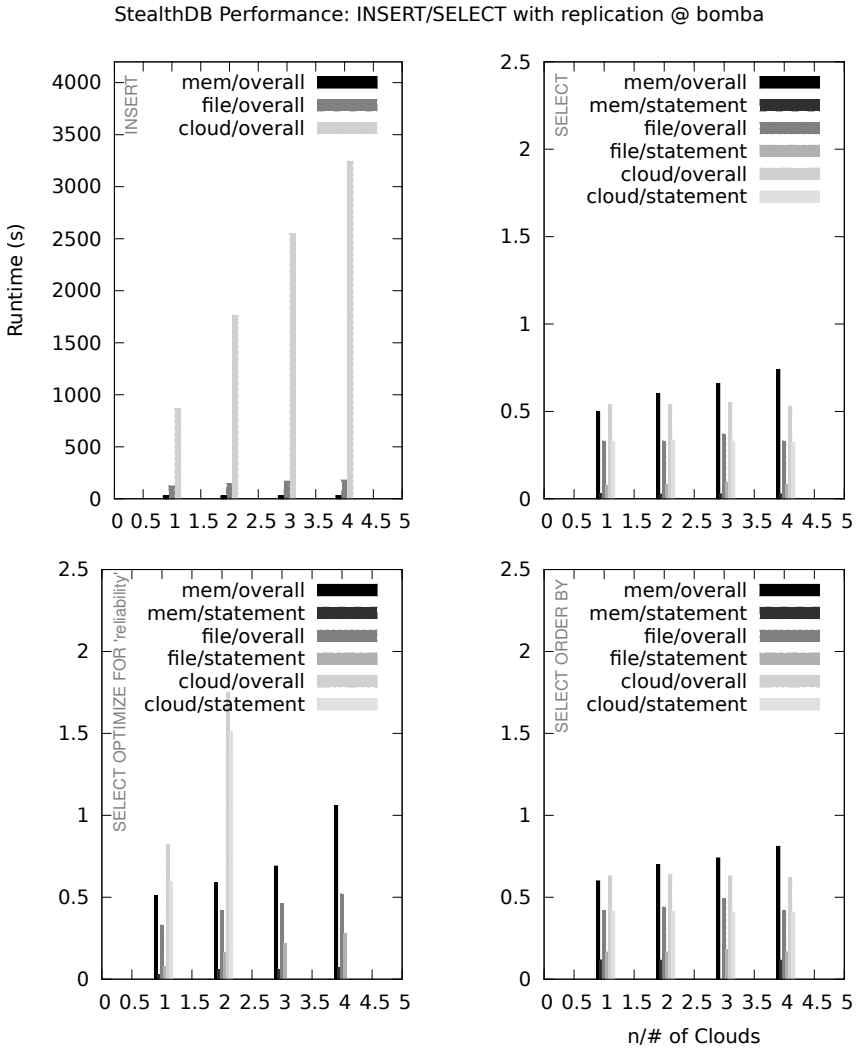

Abb. 6.48 Laufzeiten von Einfüge- und Abfrageoperationen über replizierte Clouds in StealthDB

Aus dem ersten Teildiagramm geht hervor, dass aufgrund fehlender Masseneinfügeoperationen die korrespondierende Laufzeit vor allem aufgrund der vielen generierten RPC-Anfragen stark ansteigt. Weiterhin ist im zweiten Teildiagramm zu erkennen, wie die Brutto-Abfragezeit von Arbeitsspeicherzielen anwächst, während die anderen näherungsweise konstant bleiben. Dies liegt an der Persistierung der Speicherinhalte in den Metadaten zu den Tabellen und Spalten. Bei jedem Aufruf von StealthDB werden diese mit geladen. Aus dem dritten Teildiagramm lässt sich ableiten, dass die Abfrage aller Speicherziele, die im vorliegenden Experiment se-

riell durchgeführt wurde, im Fall von Arbeitsspeicher- oder Dateispeicherzielen nur einen geringen Laufzeitanstieg mit wachsender Zahl an Speicherzielen verursacht. Hingegen steigt die Laufzeit RPC-basierten Anfragen linear an, da keine versteckten Caching-Effekte greifen.

Das zweite Experiment vergleicht die Ausführungszeiten von Aggregationsoperationen über eine und zwei Speicher- und Verarbeitungseinheiten vom Typ Arbeitsspeicher und Cloud mit unterschiedlichen Kodierungen. Die Summe und davon abgeleitet der arithmetische Mittelwert, Minimum und Maximum sowie der Median einer Liste von 10000 Ganzzahlen werden im Experiment gebildet. Abbildung 6.49 stellt links die Arbeitsspeicher- und rechts die Cloud-Messungen dar. Die Konfigurationen beinhalten die Speicherziele (`mem` und `cloud`) sowie die Kodierung (`d` für Dispersion, `e` für Verschlüsselung). Liegt eine Verschlüsselung mehrerer Ziele ohne Dispersion vor wie im Fall `e2`, so werden die Daten mit einer Vollreplikation verteilt. Offensichtlich benötigen fast alle Operationen ähnliche Zeiten für dieselbe Konfiguration. Jedoch profitieren Minimum und Maximum im Cloud-Fall von der schnelleren ordnungserhaltenden Verschlüsselung.

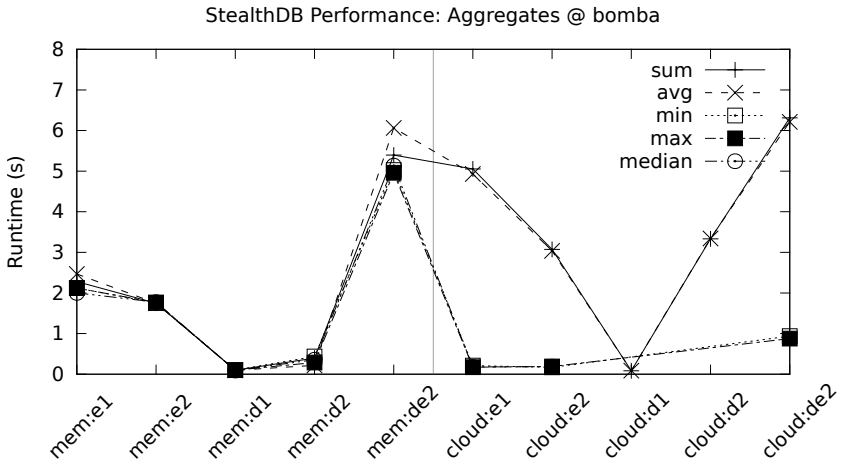

Abb. 6.49 Laufzeiten von Aggregationsoperationen über verschlüsselte und dispergierte Clouds in StealthDB

Sind Messwerte nicht vorhanden, beispielsweise erkennbar am fehlenden Maximum für dispergierte unverschlüsselte Datenverteilung auf ein oder zwei Clouds, so liegt dies an Fehlern oder inkorrekten Ergebnissen in der Implementierung, die durch den Versuchsaufbau gefiltert worden sind. Die Lücken drücken damit eine Notwendigkeit der Implementierungsverbesserung aus.

Die Wertunterschiede und deren Kompositionalität lassen sich nutzen, um eine Vorhersage und damit eine Absicherung einer Zusicherung komplexer Speicher-

flussketten durchzuführen [Spi14]. Derartige Verfahren wären für die Anwendungsentwicklung denkbar, um damit weniger ressourcenbezogen und vielmehr zielgrößenbezogen eine Vorab-Optimierung der Datenhaltung im Rahmen des Entwicklungsprozesses zu vollziehen.

6.8 Funktionstüchtigkeit und Eigenschaften der Datenstromverarbeitung

Neben separaten Betrachtungen zu der Übertragung und Speicherung von Daten ist auch die kombinierte Betrachtung der Datenstromübertragung für Stealth-Architekturen von Belang. In diesem Abschnitt werden Datenströme sowohl in erweiterten Stromspeicherdiensten als auch in um Ereignisstromverarbeitung (*event stream processing*) erweiterten Datenbanksystemen betrachtet.

6.8.1 Beschreibung der Software

Zur Überprüfung der Delta-Fähigkeit von WebDAV wurde ein Testwerkzeug in Python mit dem Modul `webdavpubsub.py` entwickelt. Es realisiert den Aufbau von HTTP- und HTTPS-Verbindungen, unterstützt die Authentifizierung mit Benutzernamen und Passwort in der Variante Basic-Auth, prüft per HEAD-Anfrage die Unterstützung für Delta-Updates ab, und nutzt im Erfolgsfall die dafür vorgesehenen HTTP-Felder `Content-Length: len` und `Content-Range: bytes pos,pos+len-1/*` für das Publizieren sowie `Range: bytes=pos,pos+len-1` für das Subskribieren. Bemerkenswert ist an dieser Stelle der Syntaxunterschied zwischen beiden Übertragungsrichtungen.

In analoger Form existiert für den Zugriff auf die Speicherdienstschnittstelle von Google Cloud Storage ein leichtgewichtiges Modul, welches das vorhandene Werkzeug `gsutil` so parametrisiert, dass Daten abschnittsweise übertragen werden. Durch die dienstseitige Konkatenierung entsteht ein Datenstrom, welcher jedoch aufgrund dokumentierter dienstspezifischer Einschränkungen nach jeweils 1023 Konkatenierungen durch die Ersetzung mit einer leeren Datei zurückgesetzt werden muss. Somit ist eine clientseitige Erkennung der Zurücksetzung und eine darauf adaptierte Delta-Parametrisierung notwendig.

Die Verarbeitung von Datenströmen im Datenbanksystem StealthDB erfordert eine Syntaxerweiterung um persistente Abfragen. Diese werden mit der Syntax `SELECT ... PERMANENT;` im Speicher der Datenbankverwaltung hinterlegt und für jede Aktualisierung der betroffenen Spalten erneut aktiviert. Die Feststellung der Aktualisierung erfolgt dabei über speicherzielspezifische Mechanismen. Daten im Speicherbereich `memory` sind nicht für externe Prozesse freigegeben und müssen somit auch nicht auf Veränderung überprüft werden. Für `file` wird der Notifikationsmechanismus des Betriebssystems (`inotify`) genutzt. Für `cloud` wird

angenommen, dass protokollabhängige Notifikationsmechanismen beispielsweise über XMPP-Nachrichten oder über permanent aufgebaute HTTP-Verbindungen mit Pipelining-Nachrichten existieren.

6.8.2 Experimente zu WebDAV-Pub/Sub

In der Tabelle 6.4 werden die WebDAV-Eigenschaften von insgesamt 17 kommerziellen Speicherdienstanbietern mit kostenlosem Zugang verglichen und das jeweilige Ergebnis der Prüfung auf Deltafähigkeit zugeordnet. Von der Prüfung ausgenommen sind Dienste, zu deren Anmeldung die Angabe von Adressen oder Zahlungswegen notwendig ist, womit jedoch immer noch eine repräsentative Menge an bekannten und populären Diensten aus mehreren Jurisdiktionen verbleibt. Im Ergebnis sind nur zwei der Dienste, SafeSync und MyDisk, deltafähig, wobei es insgesamt eine große Varität im Antwortverhalten der Dienstimplementierungen gibt. So antwortet SafeSync mit dem HTTP-Code 200, was dazu führt, dass stromspeichernde Anwendungen eine relaxierte Fehlerbehandlung implementieren müssen, welche als unerwünschten Seiteneffekt andere Fehler fälschlicherweise tolerieren könnte, oder aber eine anbieterspezifische Ausschlussliste (*blacklist*) einführen müssen, was wiederum zu einem volatilen und wartungsintensiven Satz von Metadaten führt. Einzig MyDisk hält somit die HTTP-Spezifikation in einem akzeptablen Maß ein. Weiterhin liefern neun weitere Anbieter positive HTTP-Codes (2xx) trotz der nicht implementierten Delta-Aktualisierungen. Fünf weitere Anbieter lehnen die Anfragen ab, liefern aber höchst diverse HTTP-Codes in der Antwort. Schließlich scheiterte im Versuch bei einem Anbieter die Authentifizierung, da vermutlich nicht das Basic-Verfahren, sondern das Digest-Verfahren eingesetzt wird, wofür das Testwerkzeug keine Unterstützung bietet.

Das nächste Experiment beschreibt nun den Einsatz der zwei gefundenen stromfähigen Speicherdienste für eine Pub/Sub-Übertragung dispergierter Daten. Hierfür generiert ein weiteres Testwerkzeug Zahlen im Bereich 0-255 mit einer Größe von einem Byte, zerlegt diese in zwei Fragmente von wiederum jeweils einem Byte Größe mit folglich 50% Bitredundanz, und übermittelt diese per WebDAV über HTTPS an die beiden Speicherdienste. Die Übertragung nutzt WLAN und eine DSL-Verbindung. Die Übertragung kann dabei sowohl seriell als auch parallel durch die Nutzung mehrerer synchronisierter Threads erfolgen. Eine weitere Optimierung stellt die Nutzung persistenter HTTPS-Verbindungen dar.

Abbildung 6.50 vergleicht die gemessenen Übertragunszeiten für die serielle, parallele und persistent parallele Konfiguration. Jede Konfiguration wird für vier unterschiedlich große Sammelübertragungen mit einer Batch-Größe von 1-1000 Zahlen und Bytes dargestellt. Auffällig ist, dass die serielle Übertragung mit hohen Latenzen und hohen Varianzen nicht zu empfehlen ist. Optimal für die Übertragung sind persistent parallele Verbindungen, die ab dem zweiten zu übermittelnden Wert weitgehend unabhängig von der Sammelgröße bei einer Latenz von etwa einer Sekunde pro Anfrage liegen.

Tabelle 6.4 Vergleich von WebDAV-Schnittstellen kostenloser Speicherdienste im Internet

Dienst und Anbieter	PUT-Statusmeldung	Deltafähigkeit
SafeSync	200 OK	ja
MyDisk	204 No Content	ja
T-Online Mediencenter	204 No Content	nein (falsch positiv)
DriveHQ	204 No Content	nein (falsch positiv)
StorageMadeEasy	204 No Content	nein (falsch positiv)
IDriveSync	201 Created	nein (falsch positiv)
PowerFolder	201 Created	nein (falsch positiv)
4Shared	201 Created (nur HTTP)	nein (falsch positiv)
Yandex Disk	201 Created + Socketfehler	nein (falsch positiv)
MyDrive	200 OK	nein (ignoriert)
GMX Media Center +	400 Bad Request	nein
WEB.DE SmartDrive	s.o.	s.o.
Box.net	501 Not Implemented	nein
DropDAV	501 Not Implemented	nein
Dump Truck	500 Internal Server Error	nein
Cubby	200 OK + 501 Not Implemented	nein
CloudMe	401 Unauthorized	nicht bestimmbar

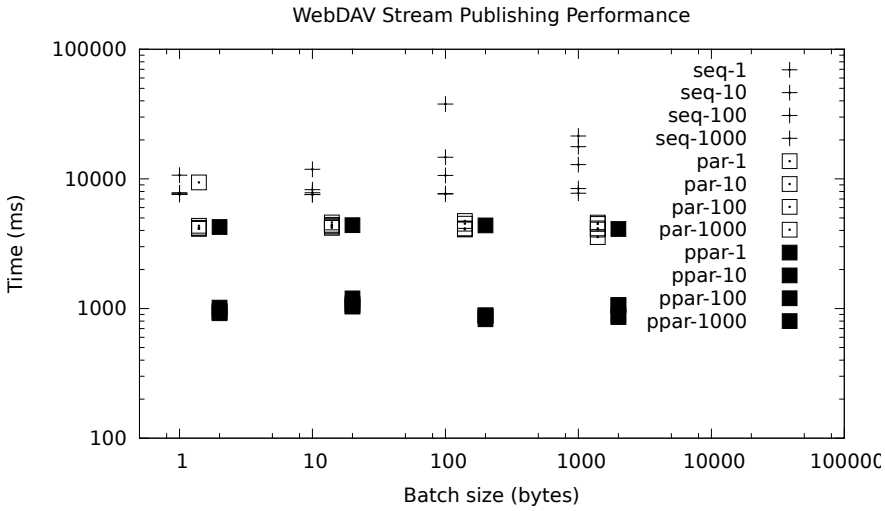

Abb. 6.50 Geschwindigkeitsvergleich der Veröffentlichung dispergierter Datenströme

6.8.3 Experimente zu GCS-Pub/Sub

Die Nutzung des kostenpflichtigen Speicherdienstes Google Cloud Storage (GCS) zur Übertragung von Datenströmen und zur Realisierung von Publish-Subscribe-Abläufen wird in zwei Experimenten unter Bestimmung der Abwägung zwischen

entstehenden Kosten und erzielter Übertragungsqualität bestimmt. Als Maß für die Übertragungsqualität gilt dabei die vollständige und rapide Übertragung von Daten in Form von Strömen und Nachrichten. Die Experimente sind mit Verzögerungen auf der Empfängerseite zu 0, 2 und 5 Sekunden versehen. Hierbei wird das Werkzeug `gsutil` genutzt, welches selbst eine Aufruf- und Authentifizierungsverzögerung von ca. 1-2 Sekunden aufweist, so dass die resultierenden Aufrufperioden sich um diesen Wert verlängern.

Aufgrund der Sichtbarkeit gespeicherter Dateien erst nach Abschluss der Speicherung ist es nicht möglich, annahmefrei beliebige Datenströme über diese Schnittstelle zu speichern. Es wird hingegen angenommen, dass der Datenstrom clientseitig unterteilt werden kann, ohne jedoch ein direktes Speichern nach jedem Datenwert im Strom vorauszusetzen. Dies impliziert einen zweiten parallelen Prozess, welcher die Datenstromabschnitte entgegennimmt und speichert.

Abbildung 6.51 zeigt die Ergebnisse für die Übertragung von Datenströmen unter diesen Annahmen. Die erste Konfiguration weist eine serielle Ausführung der Befehle `concat` und `rm` auf, was für den Fall, dass der Datenstrom zwischen diesen Befehlen aktualisiert wird, zu einem nichtdeterministischen Parallelismus (*race condition*) führt. Folglich werden ca. 18% der Daten durch Verlust korrumpiert. Die nächsten drei Konfigurationen bieten diese Lücke nicht, sind aber immer noch vom Zeitverhalten abhängig und verhindern nur mit einer hohen Abfragefrequenz und demnach mit einem hohen monetären Aufwand die Datenkorruption. Schließlich bietet das verlässliche Verfahren selbst mit einer niedrigen Abfragefrequenz einen vollständig unkorrumpierten Datenstrom zu im Vergleich mit der reinen Datenübertragung vernachlässigbaren Mehrkosten für die API-Aufrufe. Hierbei werden die Datenstromabschnitte in separate, chronologisch durchnummerierte Dateien geschrieben, welche iterativ anhand ihrer Ordnung konkateniert werden, so dass sowohl die Vollständigkeit als auch die Ordnung der Daten gewahrt bleiben.

In ähnlicher Form werden im zweiten Experiment diskrete Nachrichten mit einer festen Größe von 5 Bytes übertragen, wobei der Übertragungsstrom alle 100 Nachrichten zurückgesetzt wird, um im Gegensatz zum Maximum von 1023 Konkatenierungen keine Idealbedingungen vorauszusetzen. Die Rücksetzung muss vom Empfänger per Polling bestimmt werden. Dieses findet ebenfalls mit den Netto-Verzögerungszeiten von 0, 2 und 5 Sekunden statt.

Abbildung 6.52 zeigt die Ergebnisse der Nachrichtenübertragung. Das ideale Ergebnis, bei jeder Abfrage genau eine Nachricht zu 5 Byte zu erhalten, wird bei der Verzögerung von 0 Sekunden nahezu erreicht, allerdings auf Kosten einer sehr hohen Leerlaufrate, also Abfragen, welche keine Ergebnisse liefern, dafür aber Kosten verursachen. Demgegenüber bringt ein Wartezeitraum von 5 Sekunden nahezu keine Nachrichten zu 5 Bytes, sondern zumeist größere Bündel mit entsprechender Verzögerung, weist dafür aber auch geringere Mehrkosten auf. Die Konfiguration mit 2 Sekunden Verzögerung vergrößert die Übertragungsqualität ohne Mehrkosten und kann in diesem Fall als Optimum angesehen werden.

Die Ergebnisse sind in Tabelle 6.5 noch einmal gruppiert zusammengefasst. Die Interpretation der Ergebnisse lässt die Nutzung von Google Cloud Storage als Stromspeicherdienst und Nachrichtenverteildienst mit Einschränkungen zu.

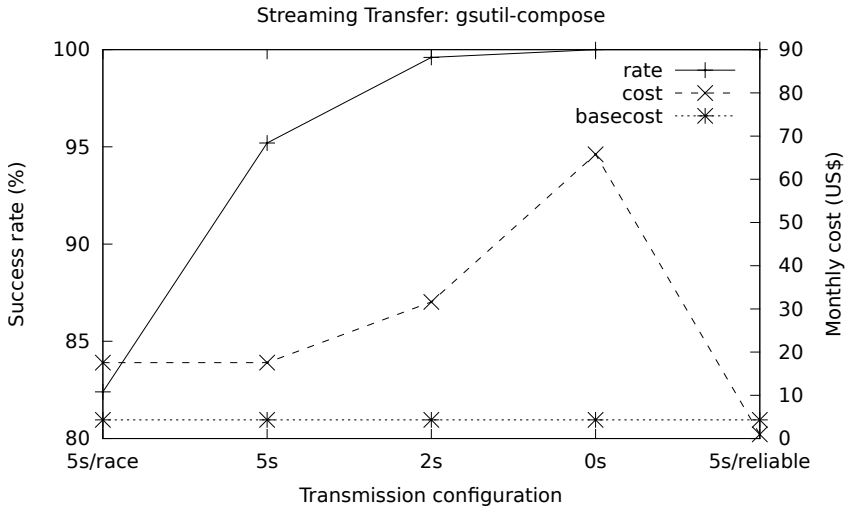

Abb. 6.51 Qualitäts- und Kostenvergleich der Veröffentlichung von Datenströmen

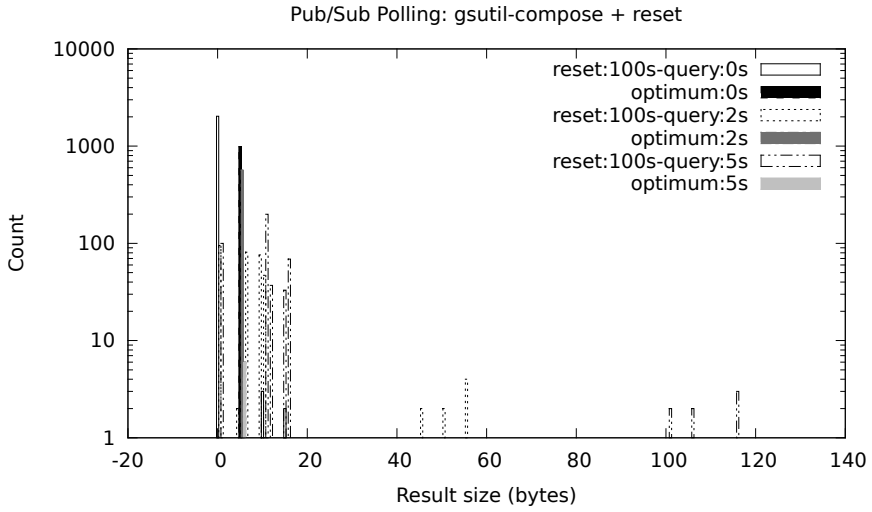

Abb. 6.52 Qualitäts- und Kostenvergleich eines Publish/Subscribe-Prozesspaares

Bemerkenswert ist die reproduzierbar beobachtbare Eigenschaft des Dienstes, trotz für sich genommen atomarer Dateioperationen (`cp` und `concat`) keine Atomizität und somit auch keine Konsistenz und keine Transaktionalität zusichern zu können. Diese Eigenschaft manifestiert sich darin, dass trotz einer festen Nachrichtengröße von 5 Bytes die Größe der empfangenen Nachrichten nicht stets ein Vielfaches von 5 ergibt.

Tabelle 6.5 Gruppierung der Polling-Ergebnisse eines Publish/Subscribe-Prozesspaares

Abrufverzögerung	kostenproblematisch (<5 Bytes)	Optimum (=5 Bytes)	qualitätsproblematisch (>5 Bytes)
0s	2031	986	6
2s	97	566	215
5s	100	6	350

6.8.4 Experimente zu StealthDB-ESP

Das mit zwei Instanzen von StealthDB durchgeführte Experiment beschreibt, wie Daten in der ersten Instanz per Dispersion aufgeteilt und in zwei `file`-Speicherziele eingetragen werden und wie in der zweiten Instanz eine permanente Abfrage zur Berechnung des Durchschnittswertes installiert wird, welche bei jedem neuen Wert ausgeführt wird. Die Daten sind dabei Temperaturdaten in °C. Die Durchschnittsberechnung findet dabei stets über alle eingetragenen Werte statt; eine Fensterbildung wie in vielen Ereignisverarbeitungen üblich [MBF14] wird nicht genutzt.

Die Quelltexte 6.2 und 6.3 zeigen die erste Instanz der Datenbankverwaltung mit den Vorbereitungs- und Einfügeoperationen. Aus Verständlichkeitsgründen sind die Debug-Ausgaben des Systems gekürzt mit aufgeführt. Demgegenüber zeigt Quelltext 6.4 die zweite Instanz mit installierter permantenter Abfrage. Das Experiment belegt, dass die Funktionalität zwar nur rudimentäre Stromverarbeitung unterstützt, dies jedoch für viele Einsatzgebiete wie der sicheren Datenpersistierung von Sensoren in Cloud-Dienste bereits hilfreich ist.

Listing 6.2 Vorbereitungsoperationen in der ersten StealthDB-Instanz

```
~~ StealthDB ›pqueries ›Wed Feb 25 18:44:05 2015 +0100 ~~
Type HELP; to get started.
Using database 'stealthdb'.
Storing all data and performing all procedures on ['mem://
    localhost'] with ['replication'].
>>> USE CLOUDS 'file:///tmp/mycloud' AND 'file:///tmp/
    anothercloud' WITH 'dispersion';
Using database 'stealthdb'.
Storing all data and performing all procedures on ['file:///tmp/
    mycloud', 'file:///tmp/anothercloud'] with ['dispersion'].
>>> CREATE TABLE temperature (celsius REAL);
Created table temperature.
(DEBUG:notifier:watch /tmp/mycloud/temperature/celsius)
Added column celsius of type REAL.
```

Listing 6.3 Einfügeoperationen in der ersten StealthDB-Instanz

```
>>> INSERT INTO temperature (celsius) VALUES (20.9);
(DEBUG:dispersion:conversion:float to fixedpoint -> 20900000)
(DEBUG:dispersion:typing=INT:b'\xa0\xe8>\x01')
(DEBUG:bitsplit-split:b'\xa0\xe8>\x01'[32 bits] => [b'...', b
    '...'])
(DEBUG:dispersion:fragment=b'...',encoded=...)
(DEBUG:dispersion:fragment=b'...',encoded=...)
(DEBUG:notifier:write event: /tmp/mycloud/temperature/celsius)
Inserted values into 1 column(s).
>>> INSERT INTO temperature (celsius) VALUES (20.5);
(DEBUG:dispersion:conversion:float to fixedpoint -> 20500000)
(DEBUG:dispersion:typing=INT:b' \xce8\x01')
(DEBUG:bitsplit-split:b' \xce8\x01'[32 bits] => [b'...', b'...'])
(DEBUG:dispersion:fragment=b'...',encoded=...)
(DEBUG:dispersion:fragment=b'...',encoded=...)
(DEBUG:notifier:write event: /tmp/mycloud/temperature/celsius)
Inserted values into 1 column(s).
>>> SELECT * FROM temperature;
Column celsius [REAL]:
(DEBUG:bitsplit-join:b'...' => [b'...', b'...'][8 bits])
(DEBUG:dispersion:conversion:fixedpoint to float)
(DEBUG:dispersion:decoded+joined=20.9)
(DEBUG:bitsplit-join:b'...' => [b'...', b'...'][8 bits])
(DEBUG:dispersion:conversion:fixedpoint to float)
(DEBUG:dispersion:decoded+joined=20.5)
% 20.9
% 20.5
```

6.9 Integriertes Szenario: Online-Speicherung von Dateien

Die sichere Online-Speicherung von Dateien mit Suchfunktion setzt das komplexe Szenario aus Abschnitt 5.1 in Form einer webbasierten Dokumentenverwaltung mit verteilter physischer Dateiablage um. Da Suchvorgänge über verschlüsselte Daten noch Gegenstand intensiver Forschung sind [KBS14, BHJP15] und aus diesem Grund von der Realisierung ausgenommen werden, wird das resultierende System mit dispergierter Volltextsuche in der Stealth-3V-Notation als *DPC* bezeichnet.

6.9.1 Beschreibung der Software und des Gesamtsystems

Eine Kombination aus mehreren kombinierten Speicher- und Verarbeitungsdiensten, einem Speicherdienst-Controller und einer Weboberfläche respektive einer Mobilanwendung unter Kontrolle des Anwenders, und einem über den Verarbeitungsdienst angebotenen Suchdienst wird zur Realisierung des Szenarios eingesetzt. So-

Listing 6.4 Abfrageoperationen in der zweiten StealthDB-Instanz

```
~~ StealthDB ›pqueries ›Wed Feb 25 18:44:05 2015 +0100 ~~
Type HELP; to get started.
(DEBUG:notifier:watch /tmp/mycloud/temperature/celsius)
Using database 'stealthdb'.
Storing all data and performing all procedures on ['mem://
    localhost'] with ['replication'].
>>> DESCRIBE temperature;
Column celsius [REAL]: {['file:///tmp/mycloud', 'file:///tmp/
    anothercloud'] with ['dispersion']}
>>> SELECT AVG(celsius) FROM temperature PERMANENT;
Permanent query registered.

>>> (DEBUG:notifier:write event: /tmp/mycloud/temperature/celsius
    )
(DEBUG:aggregate cloud:'file:///tmp/mycloud')
(DEBUG:bitsplit-join:b'...' => [b'...', b'...'][8 bits])
(DEBUG:dispersion:conversion:fixedpoint to float)
(DEBUG:dispersion:decoded+joined=20.9)
(DEBUG:bitsplit-join:b'...' => [b'...', b'...'][8 bits])
(DEBUG:dispersion:conversion:fixedpoint to float)
(DEBUG:dispersion:decoded+joined=20.5)
Average over celsius: 20.7
```

mit entstehen zwei Ausprägungen der Umsetzung, von denen eine je nach Interaktionsgerät zur Ausführung kommt.

Abbildung 6.53 beschreibt den technischen Aufbau des Szenarios. NubiSave respektive NubiDroid werden als Speicherdienst-Controller für die koordinierte Nutzung mehrerer Cloud-Speicherdienste (CS) eingesetzt. Aufgrund der Standardschnittstellen FUSE und SAF funktioniert dies transparent für die Anwendungen, wenn man von den üblichen Risiken der Netzwerktransparenz absieht [BK14a], die jedoch durch die Dispersion immerhin verringert werden.

Da es keine Standardschnittstelle für die Suche gibt, ist es Aufgabe der Anwendung, eine Suchanfrage zu kodieren, an die Cloud-Verarbeitungsdienste (CC) zu übermitteln und schließlich die Ergebnisse zu interpretieren und zusammenzuführen. Aufbauend auf der Dateisystemschnittstelle von NubiSave wird eine OwnCloud-Instanz eingesetzt, welche ein Plugin zur verteilten Suche enthält. Für die Mobilumgebung ist hingegen eine dedizierte Anwendung notwendig, da SAF keine Möglichkeit bietet, nach Dateien zu suchen.

6.9.2 Ablaufbeschreibung

Vor der erstmaligen Benutzung des Systems muss es konfiguriert werden. Dazu sind Einstellungen zur Auswahl der Speicherdienste und zur Verteilung der Daten

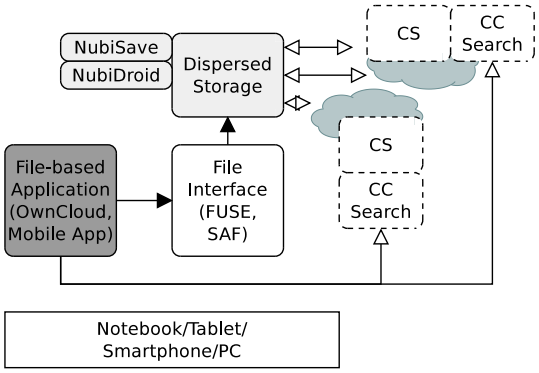

Abb. 6.53 Sichere verteilte Dokumentenverwaltung mit Suchfunktion

auf selbige ebenso notwendig wie die Erlangung von Zugangsdaten zu den Diensten. Der Konfigurationsprozess soll an dieser Stelle als abgeschlossen angenommen werden.

Die Benutzung unterteilt sich in Vorgänge zum Speichern, Laden und Suchen von Dateien. Im Fehlerfall tritt zudem eine Wiederherstellung von Dateien ein, was jedoch durch den Speicherdienst-Controller autonom realisiert wird und somit dem Anwender verborgen bleibt.

Soll eine Datei gespeichert werden, so wird diese über die Weboberfläche oder über eine beliebige Anwendung auf dem Mobilgerät durch einen Aufruf eines standardisierten Dialogs zur Auswahl der Datei abschnittsweise (per Chunking) an die Dispersionskomponente übergeben. Die Daten werden nunmehr per Bitsplitting zerlegt und an die Speicherdienste übertragen. Der Abruf erfolgt analog in umgekehrter Reihenfolge durch einen Lesezugriff auf die Dateisystemschnittstelle oder die Auswahl im SAF-Dialog.

Sollen hingegen Dateien gesucht werden, so wird die Zeichenkette entgegengenommen, per Bitsplitting aufgeteilt und die resultierenden Fragmente an die Suchdienste übergeben. Die Ergebnisse werden als Schnittmenge aller Suchpositionen zusammengeführt und dem Benutzer als geordnete Trefferliste in Form der enthaltenden Dateien angezeigt [SS14a].

6.10 Integriertes Szenario: Persönliche Datenanalyse

Das in Abschnitt 5.2 vorgestellte Szenario zur sicheren verteilten Analyse persönlicher Daten wird hier hinsichtlich seiner Umsetzung mit konkreten Geräten, Protokollen, Datenschemata und Softwarebestandteilen erörtert. In der Stealth-3V-Notation entsteht dabei ein *DEC*-System.

6.10.1 Beschreibung der Software und des Gesamtsystems

Das Szenario sieht die Auslagerung von Daten, welche personenbezogen von der jeweiligen Person selbst mit Hilfe von körper- und aktivitätsbezogenen Sensoren erhoben werden, in nicht vertrauenswürdige Dienstumgebungen vor. Ohne Stealth Computing ließe sich das Szenario nur bedingt und unter Akzeptanz von durch Vollreplikation höheren redundanten Datenmengen realisieren, während es mit Stealth Computing einfach in Anwendungen integriert und weitestgehend nichtintrusiv umgesetzt werden kann [SMS15].

Die Sensoren umfassen eine Waage vom Typ *smartLab scale w*, einen Schrittzähler vom Typ *smartLab move+*, einen Herzfrequenzmesser vom Typ *Polar WearLink+*, einen Sensor für die Messung der Raumtemperatur vom Typ *Microdia TEMPer1* und schließlich einen in einem Mobiltelefon integrierten GPS-Positionssensor vom Typ *Texas Instruments NaviLink 5030/NL5350*. Zur Anbindung eines Sensor-Gateways in Form eines Notebooks oder eines Minimal-PCs (μPC) an deren heterogenen Funkprotokolle werden USB-Adapter vom Typ *Suunto Movestick* für ANT+ und *Sitecom* für Bluetooth Low Energy (BLE), auch als Bluetooth Smart bezeichnet, genutzt. Die Datenformate und Protokolle sind teilweise offengelegt und bekannt, was beispielsweise für die Ablage des zeitlichen Verlaufs von GPS-Daten in einer GPX-Datei zutrifft, teilweise jedoch auch undokumentiert, was beispielsweise das konkrete Auslesen von Werten per BLE betrifft. In letzterem Fall ist es notwendig, die gerätespezifischen Eigenschaften per *reverse engineering* zu ermitteln, was für die vorliegende Szenarioumsetzung zur Extraktion der Daten aus der Waage, einschließlich des aktuellen Batterieladestandes, durchgeführt wurde.

Die gemessenen Daten sind zumeist kurze Zeitreihen, welche um Ausreißer, etwa verursacht durch Geräteinitialisierungsfehler, bereinigt und schließlich für die meisten Anwendungsfälle durch separate Anwendungen aggregiert werden. Dieses Vorgehen betrifft vor allem die Herzfrequenzmessung. Hingegen liefert die Waage pro Messvorgang nur ein Datum, welches mit den anderen Messungen korreliert wird, um eine Sportlichkeits- oder Fitnesseinschätzung zu erhalten.

Abbildung 6.54 erläutert den Datenfluss von Geräten und Sensoren zur Messung körperbezogener persönlicher Daten zu ausgewählten Datei- und Datenstromspeicherdiensten. Das Einfügen und Auslesen von Daten erfolgt nach Maßgabe der Stealth-Kodierung mittels der Python-Modulschnittstelle von StealthDB.

Das Datenschema umfasst jeweils eine zweispaltige Tabelle für einen Sensor mit den Namen und ersten Spalten `hearttable (heart INT)` für die Herzfrequenz, `weighttable (weight REAL)` für die Waage, `temperaturetable (temperature REAL)` für das Thermometer, `stepstable (steps INT)` für den Schrittzähler und `positiontable` für die Geopositionsdaten. Die zweite Spalte ist jeweils die Zeitangabe mit Sekundenauflösung `timestamp INT`. Die Spalten werden mit Stealth-Kodierung, also Dispersion und Verschlüsselung, auf zwei unabhängige Clouds, also zwei Speicher- und Verarbeitungsdienste, aufgeteilt.

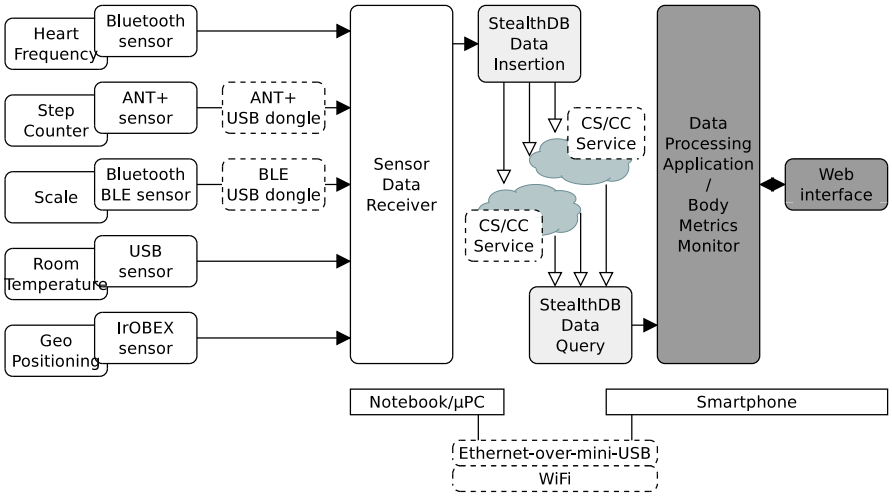

Abb. 6.54 Szenariorealisierung für die persönliche Datenanalyse

6.10.2 Ablaufbeschreibung

Die resultierende Fitness-Anwendung ist begrenzt mehrbenutzerfähig. Jeder Durchlauf wird mit der Markierung des aktuellen Benutzers über die Webschnittstelle durchgeführt. Ein Benutzer trägt dazu seinen Namen ein, woraufhin entweder ein Filter bei der Datenabfrage oder eine Rücksetzung aller gespeicherter Daten erfolgt. Die vorliegende Umsetzung enthält die zweite Variante.

Nach der asynchron durchgeführten Aufnahme aller Daten können analytische Abfragen gestellt werden. Konkret ist in der Webanwendung die Anzeige der durchschnittlichen Herzfrequenz vorgesehen, die mit dem Aufruf `SELECT AVG(heart) FROM hearttable` verteilt errechnet wird. Möglich wären im Stealth-Modus auch komplexere Abfragen wie beispielsweise bei entsprechender zusätzlicher Sensorik die Bestimmung des Körpermasseindex per Abfrage `SELECT weighttable.weight / POWER(heighttable.height, 2)`, wobei dies eine homomorphe Verschlüsselung erfordert, die im Gegensatz zum Paillier-Cryptosystem die Multiplikation eines verschlüsselten Wertes mit einer Nicht-Konstante erfordert.

6.11 Integriertes Szenario: Mobile Anwendungen für das Internet der Dinge

Diese Umsetzung realisiert das in Abschnitt 5.3 vorgeschlagene komplexe Szenario zur Entwicklung mobiler Anwendungen und Dienste für die zuverlässige Verarbeitung von Sensordaten im Internet der Dinge. Es ist dem vorherigen Szenario insofern ähnlich, als dass ebenfalls physische Sensoren als virtuelle Datenquellen

betrachtet werden. Gleichzeitig unterscheidet es sich dadurch, dass Entwicklungs- und Bereitstellungsmaßnahmen, organisationsübergreifende Zugriffe und mehrseitige Absicherungen mit Dienstgütevereinbarungen für eine stabile Integration verteilter Sensordatenströme notwendig sind.

Das Szenario zielt hierbei auf die Überprüfung von Temperaturdaten in verschiedenen Räumen eines Gebäudes, über die ein gleitender Mittelwert gebildet werden soll. Das Resultat ist ein *DEC*-System gemäß der Stealth-3V-Notation.

6.11.1 Beschreibung der Software und des Gesamtsystems

Als alternative Kommunikationsarchitektur gegenüber den vorher betrachteten RPC- und HTTP-Diensten werden für die Datenübertragung aus dem Internet der Dinge föderierte XMPP-Netze eingesetzt [BSS+13]. Die Datenübertragung wird zudem mit SLAs abgesichert, welche einen steuernden Effekt auf die Allokation von Ressourcen und die Adaption von Datensequenzen haben. Deren Verwaltung, wie auch die Verwaltung von Sensoren und weiterer Datenquellen sowie angeschlossener Empfangsgeräte mit ihren jeweiligen Anforderungen, wird durch die verteilte und cloud-fähige DaaMob-Dienstplattform vorgenommen.

In diesem Szenario ist auch die Perspektive der Entwicklung von Stealth-Anwendungen explizit verankert. Entwickler können über die Dashboard-Anwendung *SensDash* passende Sensoren zusammenstellen und ihre EEC-Beschreibungen abrufen. Diese werden in die zu entwickelnden Anwendungen als dynamische Datenquellenbeschreibungen integriert, so dass langfristige Änderungen der Endpunkte oder der Datenrelationen adaptiv ohne Änderung der installierten Anwendung und ohne Unterbrechung der Interaktion zwischen Anwender und Anwendung zur Geltung kommen können.

Abbildung 6.55 zeigt das integrierte System mit Sensoren, Adaptern für den Empfang von Daten von den Sensoren, dispergierenden Datenüberträgern und korrespondierenden Datenempfängern sowie innerhalb der XMPP-Föderation einem als XMPP-Client agierenden Aggregator, welcher Datensequenzen entgegennimmt und daraus neue aggregierte Datenströme generiert. Die DaaMob-Plattform bietet dabei die Installationsschnittstelle für Verarbeitungseinheiten an [SKU+11].

6.11.2 Ablaufbeschreibung

Mit einer Taktung von 5*s* werden diskrete Temperaturmesswerte eingelesen, aufgeteilt, verschlüsselt, base64-kodiert und in Form mehrerer dispergierter XMPP-Nachrichten (*stanzas*) an Mehrbenutzer-Chaträume (*multi-user chat*, MUC) versendet [SS14b]. In diesen befinden sich neben dem der Übertragungskomponente zugeordneten Chat-Nutzer, welche die Nachrichten lesen, aber nicht per se interpretieren können. Die Temperaturüberwachungsanwendung liest und korreliert die mit Zeit-

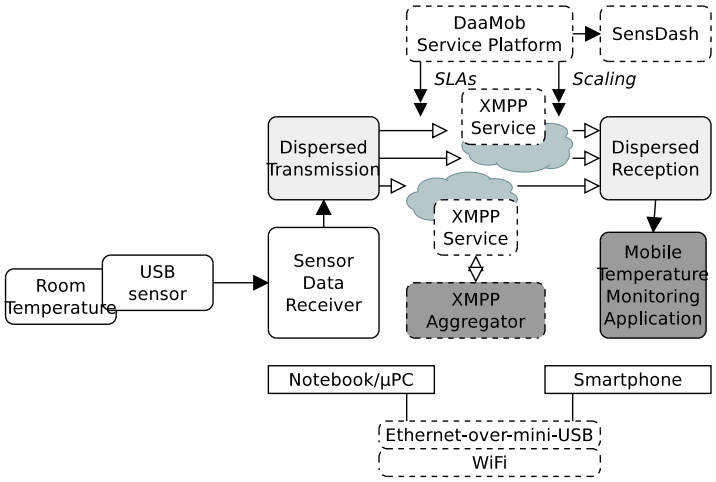

Abb. 6.55 Szenariorealisierung für Datenströme im Internet der Dinge

stempeln versehenen Nachrichten auf allen signifikanten XMPP-Servern. Sollten XMPP-Server hinzukommen oder wegfallen, so können über den gleichen Kanal Informationen über die veränderte Dienstlandschaft empfangen werden. Ebenfalls kann die Anwendung jederzeit ihre Anforderungen, etwa an die Mindestverfügbarkeit, über XMPP mitteilen. Als Teil der DaaMob-Dienstplattform übernimmt in diesem Fall eine SLA-Aushandlungskomponente die Dienstgüte-Konversation (SLC), welche letztlich die übertragende Komponente motivieren, jedoch aufgrund der organisatorischen Autarkie nicht anweisen kann, die Verteilung der Daten auf die XMPP-Server abzuändern. Schließlich erlaubt die Anwendung dem Anwender, einen gleitenden Durchschnittswert über die Temperaturdaten zu betrachten. Die Aggregation läuft dabei im Anwendungskontext auf dem Gerät oder als separater Aggregator-Client innerhalb der XMPP-Föderation.

Kapitel 7
Zusammenfassung

Zusammenfassung Die wissenschaftlichen Beiträge der Arbeit werden in diesem Kapitel kurz zusammengefasst und mit Übersichtsabbildungen zu Prozessvertikalen für die Nutzung von Cloud-Diensten und zur kodierungsabhängigen Datenverarbeitung konsolidiert. Anschließend werden sie über eine Diskussion kritisch bewertet und schließlich hinsichtlich einer weiteren Forschungsperspektive für zukünftige Arbeiten aus einem weiteren Blickwinkel betrachtet.

7.1 Zusammenfassung der Beiträge

Die Forschungsaktivitäten zu den Themen sicheres Cloud Computing, native Cloud-Anwendungen sowie zusicherbare Eigenschaften in Anwendungen bei der Nutzung von Cloud-Diensten sind vielfältig. Sie wurden deshalb in dieser Arbeit nicht in allen Facetten, sondern in einer logisch zusammenhängenden Prozesskette untersucht, welche sich als funktionale Nutzungsvertikale auffassen lässt, die um eine Konfigurations- und eine Kontrollvertikale ergänzt wird. Abbildung 7.1 stellt wiederholt die drei Vertikalen dar, wobei die Nutzungsvertikale nunmehr mit der Zuordnung der Beiträge dieser Arbeit versehen ist.

Die behandelten thematischen Schwerpunkte umfassen demnach die funktionalen und nutzungsbezogenen Aspekte der Stealth-Kodierung von Daten, die dynamische Auswahl von Diensten mitsamt der Festlegung der Verteilung der kodierten Daten, und des Transports der Daten zu den ausgewählten Diensten. Hierin wurden mittels wissenschaftlicher Arbeitsweise neuwertige theoretische Beiträge zur abstrakten Sicht auf Datenbeziehungen, zur holistischen Sicht auf die Datenkodierung unter Berücksichtigung der nachfolgenden Datenverarbeitung und zur Konstruktion von Algorithmen zur Durchführung der Datenverarbeitung geleistet.

Bezugnehmend auf die eingangs in Tabelle 2.1 zusammengefassten theoretischen Beiträge werden diese nun noch einmal rekapituliert. Im Abschnitt 3.1.3 wurde das Bitsplitting-Verfahren als Grundlage für die Datendispersion mit anschließender Verarbeitungsmöglichkeit vorgestellt. Ergänzend verschob im gleichen Abschnitt

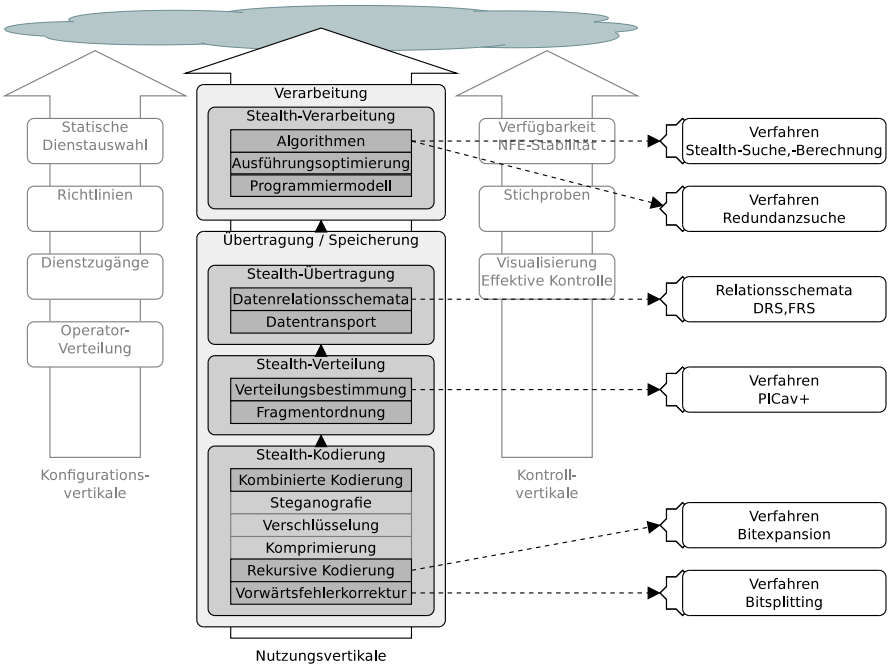

Abb. 7.1 Funktionale Vertikalen in nativen Cloud-Anwendungen

das Bitexpansions-Verfahren die Abbruchbedingung rekursiver Dispersionsschritte vom Algorithmus zum Nutzer und erhöhte somit dessen Kontrollmöglichkeiten. Ketten und Bäume zur Kombination von Kodierungsschritten bis hin zur Stealth-Kodierung wurden im Abschnitt 3.1.6 thematisiert und ebenso wie die Beziehungen zwischen den resultierenden kodierten Daten und Fragmenten im Abschnitt 3.3.1 formalisiert. Für die Verteilung aller Daten und Fragmente wurde PICav+ als Verfahren, welches nichtfunktionale Eigenschaften von Speicherdiensten berücksichtigt, vorgeschlagen 3.2.4. Die Verarbeitung der kodierten Daten wurde hinsichtlich des dispergierten Rechnens und der Suche auf redundanten Daten im Abschnitt 3.4.1, hinsichtlich des Stealth-Rechnens im Abschnitt 3.4.3 und hinsichtlich weiterer Algorithmen schließlich im Abschnitt 3.4.4 thematisiert.

Abbildung 7.2 fasst ergänzend den erarbeiteten Wissensstand bezüglich der verarbeitungsbezogenen Stealth-Kodierung zusammen. Sowohl dispergierte als auch verschlüsselte sowie verschlüsselt-dispergierte und redudante dispergierte Daten können, mit all den herausgearbeiteten Einschränkungen, ohne Zusammenführung oder Dekodierung direkt verarbeitet werden.

Über die im Kapitel 4 erarbeitete Nutzung von Stealth-Konzepten in reellen Cloud-Infrastrukturen ließen sich schließlich drei komplexe Szenarien in unterschiedlichen Anwendungsfeldern realisieren. Die nahezu vollständige Nutzung von Stealth-Konzepten ist in der Abbildung 7.3 (a-c) vergleichend dargestellt.

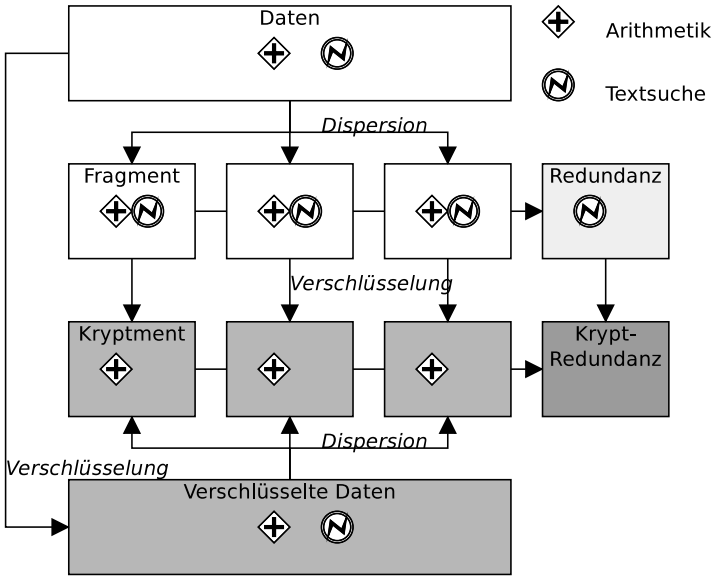

Abb. 7.2 Kodierungsabhängige Ausführbarkeit von Algorithmen

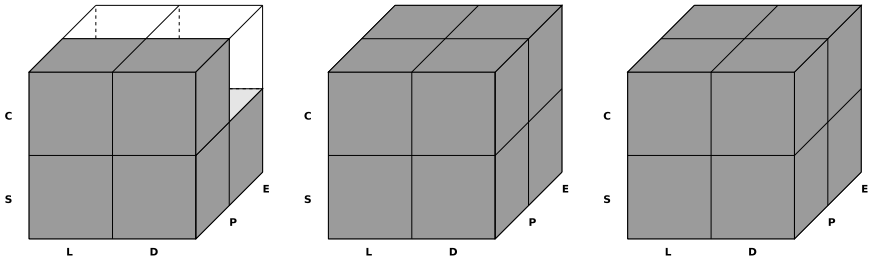

Abb. 7.3 Stealth-Cube-Darstellungen für die realisierten komplexen Szenarien (a) sichere Online-Speicherung von Dateien mit Suchfunktion, (b) persönliche Datenanalyse und (c) mobile Anwendungen für das Internet der Dinge

7.2 Kritische Diskussion und Bewertung

Hinsichtlich des Schutzes der Privatsphäre bei der Nutzung von nicht durch den jeweiligen Nutzer kontrollierbaren Diensten ist eine generelle kritische Diskussion über die Technosphäre hinaus angebracht. Technische Verfahren wie Stealth Computing können eine Schutzschicht bilden, müssen dazu aber Teil der Entwicklungsmethodik der betroffenen Anwendung sein, um einerseits eine weite Verbreitung zu finden, andererseits auch technisch dafür zu sorgen, dass Anforderungen des Nutzers bereits vor der irreversiblen Weitergabe von Daten erfüllt werden. Hieraus lässt sich die Empfehlung ableiten, dass bisher dominierende Paradigmen zur Konstruktion verteilter Client-Server-Anwendungen mit Zwei- oder Drei-Schichten-

Architekturen um eine weitere clientseitige Stealth-Schutzschicht ergänzt werden sollten. Die Ergänzung sollte entwicklerfreundlich weniger basierend auf technischen Parametern wie Redundanzgrad oder Schlüssellänge, sondern vielmehr basierend auf Zielgrößen wie erwarteter Verfügbarkeit und Vertraulichkeit vorgenommen werden. Die Arbeit enthält selbst keine Beiträge zu angepassten Software-Entwicklungsumgebungen oder Vier-Schichten-Anwendungs-Frameworks, bildet aber eine passende Grundlage dafür.

Weiterhin ist selbst bei der Anwendung von Stealth-Techniken keine absolute Sicherheit und keine unumstößliche Zusicherung von Eigenschaften möglich. Die in der Praxis auftretenden Laufzeitverschlechterungen müssen ebenso eingeplant und dem Nutzer transparent beigebracht werden. Der Anspruch, nutzerkontrollierbare Zusicherungen zu geben, kann nur durch dienst- und anwendungsspezifische Konfigurations- und Kontrollmöglichkeiten für Kombinationen nichtfunktionaler Eigenschaften erfüllt werden. Die in dieser Arbeit geschaffenen Möglichkeiten werden in Tabelle 7.1 zusammengefasst. Aus ihr wird deutlich, dass weitere Untersuchungen und zusätzliche Schnittstellen zwischen den Anwendungen und ihren Anwendern notwendig sind.

Tabelle 7.1 Konfiguration und Kontrolle nichtfunktionaler Eigenschaften

Dienstschnittstelle	Konfiguration	Kontrolle
Speicherdienst	Konfigurationsdatei, GUI	-
Stromspeicherdienst	-	-
Netzwerkproxy	Konfigurationsdatei/-parameter	-
Datenbank	σ-SQL-Syntax	-
Ereignisverarbeitung	-	-

Die Verknüpfung von Dispersion und Verschlüsselung ist für simple Algorithmen gewinnbringend, jedoch mit dem derzeitigen Forschungs- und Umsetzungsstand nicht vollständig, was durch die fehlende Suche auf verschlüsselt-dispergierten Daten im Datei-Speicherdienst-Szenario belegt wird.

Schließlich lässt sich die kritische Betrachtung auf auf die Problematik einer *dual-use*-Technologie ausweiten. Das Schutzniveau von Anwendungen und Anwendern wird durch Stealth-Techniken erhöht, während gleichzeitig solche Techniken eingesetzt werden können, um nicht identifizierbare oder zuordbare Risiken einschließlich intrusiver Programme und Angreifer zu stärken. In Analogie zum Wechselspiel zwischen Kryptographie und Kryptanalyse wird in einem solchen Fall ein Wechselspiel aus Schutzbildung und Schutzreduktion die Folge sein.

7.3 Ausblick

Der Ansatz, eine nutzerkontrollierte Datenhoheit in der Cloud zu schaffen und darüber auch weitere Aktivitäten in der Cloud mit Hilfe bewusst mit dieser Zielset-

zung entwickelten nativen Anwendungen abzusichern, wird auch in der näheren Zukunft unabhängig von der weiteren technologischen Ausprägung des oftmals verwässerten und überstrapazierten Cloud-Begriffs von hoher gesellschaftlicher Relevanz sein.

Aus der vorliegenden Arbeit lassen sich unmittelbar und instantan weitere informatikweite Forschungsfragen und Folgearbeiten ableiten. Diese reichen von der Konstruktion neuer Algorithmen zur Verarbeitung von Stealth-Daten über neuartige Software-Entwicklungsverfahren für native Cloud-Anwendungen bis hin zur Definition neuer Klassen adaptiver verteilter Systeme zur Daten- und Datenstromverarbeitung.

Eine aus wissenschaftlicher Sicht interessante Frage ist, ob Daten so kodiert oder Algorithmen so adaptiert werden können, dass jeder denkbare Algorithmus auf gesicherte Daten abbildbar ist und somit $\mathscr{S}$ erreichbar wird. Analog zur Generalisierung von partiellen zu vollständigen Verschlüsselungsverfahren, die eine Berechnung im homomorphen Raum erlauben, wobei diese aber selbst im vollständigen Fall nicht Turing-vollständige, sondern nur additive und multiplikative arithmetische Funktionen zulassen, wäre eine ähnliche Generalisierung für dispergierte oder anderweitig verteilte Daten denkbar. Während in der vorliegenden Arbeit ausgewählte Algorithmen realisiert worden sind, fehlt sowohl ihr als auch der Literatur allgemein ein Ansatz, dies automatisiert für komplexere Algorithmen durchzuführen. Darauf aufbauend müsste noch untersucht werden, ob die strukturerhaltenden Merkmale von dispergierten und verschlüsselten Daten stets ausreichen, um eine Stealth-Kombination zu bilden. Das Resultat wäre in diesem Fall von hoher Wirksamkeit für die Migration von Altanwendungen sowie den Entwurf von Neuanwendungen für Cloud-Umgebungen.

Verzeichnisse

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Literaturverzeichnis

AAEM09. Divyakant Agrawal, Amr El Abbadi, Fatih Emekci, and Ahmed Metwally. Database Management as a Service: Challenges and Opportunities. In *25th IEEE International Conference on Data Engineering (ICDE)*, pages 1709–1716, April 2009. Shanghai, China.

AC15. Michele Albano and Stefano Chessa. Replication vs erasure coding in data centric storage for wireless sensor networks. *Computer Networks*, 77:42–55, February 2015.

Ack89. Russell L. Ackoff. From Data To Wisdom. *Journal of Applied Systems Analysis*, 16:3–9, 1989.

ACKM04. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer, 1st edition, 2004.

AGI13. Mohamed Almorsy, John Grundy, and Amani S. Ibrahim. Adaptable, model-driven security engineering for SaaS cloud-based applications. *Automated Software Engineering*, 21(2):187–224, September 2013.

AGLP01. Marco Aldinucci, Sergei Gorlatch, Christian Lengauer, and Susanna Pelagatti. Towards Parallel Programming by Transformation: The FAN Skeleton Framework. *Parallel Algorithms and Applications*, 16(2):1–30, 2001.

AHP+13. Pekka Abrahamsson, Sven Helmer, Nattakarn Phaphoom, Lorenzo Nicolodi, Nick Preda, Lorenzo Miori, Matteo Angriman, Juha Rikkilä, Xiaofeng Wang, Karim Hamily, and Sara Bugoloni. Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment. In *UsiNg and building ClOud Testbeds (UNICO) workshop at the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, volume 2, pages 170–175, December 2013. Bristol, United Kingdom.

AJM12. Jameela Al-Jaroodi and Nader Mohamed. Service-oriented middleware: A survey. *Journal of Network and Computer Applications*, 35(1):211–220, January 2012.

AKSX04. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order Preserving Encryption for Numeric Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 563–574, June 2004. Paris, France.

Ali15. Syed Ali. Hosted Cloud Solutions Using Google Cloud Platform. In *Practical Linux Infrastructure*, Apress, pages 25–51. Springer, January 2015.

ALPW10. Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. RACS: A Case for Cloud Storage Diversity. In *1st ACM Symposium on Cloud Computing (SoCC)*, pages 229–240, June 2010. Indianapolis, Indiana, USA.

Ans91. Dean Ansley. ZSoft PCX File Format Technical Reference Manual. Booklet, 1991.

APST12. Mohammed A. AlZain, Eric Pardede, Ben Soh, and James A. Thom. Cloud Computing Security: From Single to Multi-Clouds. In *45th Hawaii International Conference on System Sciences (HICSS)*, pages 5490–5499, January 2012. Maui, Hawaii, USA.

ASP14. Rami Akkad, Henning Schmitz, and André Deienno Pansani. Operating the HPI Future SOC Lab – An Infrastructure Laboratory for Researchers – A Practicioners' Report. In *44. Jahrestagung der Gesellschaft für Informatik - INFORMATIK: Big Data – Komplexität meistern*, pages 1661–1666, September 2014. Stuttgart, Germany.

AUS+09. Kota Abe, Tatsuya Ueda, Masanori Shikano, Hayato Ishibashi, and Toshio Matsuura. Toward Fault-Tolerant P2P Systems: Constructing a Stable Virtual Peer from Multiple Unstable Peers. In *The First International Conference on Advances in P2P Systems (AP2PS)*, pages 104–110, October 2009. Sliema, Malta.

Bad09. Dirk Bade. Verteilte Abfragebearbeitung von Sensordaten. 8. GI/ITG KuVS Fachgespräch Sensornetze, 2009. Hamburg, Germany.

BBF+11. Prith Banerjee, Cullen Bash, Rich Friedrich, Patrick Goldsack, Bernardo A. Huberman, John Manley, Chandrakant Patel, Partha Ranganathan, and Alistair Veitch. Everything as a Service: Powering the New Information Economy. *Computer*, 44(3):36–43, March 2011.

BCLO09. Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-Preserving Symmetric Encryption. In *28th Annual International Conference on the Theory and Applications of Cryptographic Techniques - Advances in Cryptology (EUROCRYPT)*, volume 5479 of *LNCS*, pages 224–241, April 2009. Cologne, Germany.

BCOP04. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with Keyword Search. In *Advances in Cryptology – 23rd Annual Eurocrypt Conference (Eurocrypt)*, volume 3027 of *LNCS*, pages 506–522, 2004. Interlaken, Switzerland.

BD05. Carlo Blundo and Paolo D'Arco. Analysis and Design of Distributed Key Distribution Centers. *Journal of Cryptology*, 18(4):391–414, September 2005.

BDR14. Florian Berg, Frank Dürr, and Kurt Rothermel. Optimal Predictive Code Offloading. In *11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, December 2014. London, UK.

BFLW12. Sebastian Burckhardt, Manuel Fähndrich, Daan Leijen, and Benjamin P. Wood. Cloud Types for Eventual Consistency. In *Proceedings of the 26th European Conference on Object-Oriented Programming (ECOOP)*, pages 283–307, June 2012. Beijing, China.

BFP14. Ulrike Baumann, Elke Franz, and Andreas Pfitzmann. Historische Chiffren und deren Analyse. In *Kryptographische Systeme*, pages 3–24. eXamen.press, 2014.

BGF11. Fatna Belqasmi, Roch H. Glitho, and Chunyan Fu. RESTful Web Services for Service Provisioning in Next-Generation Networks: A Survey. *IEEE Communications Magazine*, 49(12):66–73, December 2011.

BHH13. Mario Blaum, James Lee Hafner, and Steven Hetzler. Partial-MDS Codes and Their Application to RAID Type of Architectures. *IEEE Transactions on Information Theory*, 59(7):4510–4519, July 2013.

BHJP15. Christoph Bösch, Pieter H. Hartel, Willem Jonker, and Andreas Peter. A Survey of Provably Secure Searchable Encryption. *ACM Computing Surveys (CSUR)*, 47(2):18, January 2015.

Bia10. Jiang Bian. *JigDFS: A Secure Distributed File System and its Applications*. PhD thesis, University of Arkansas at Little Rock, December 2010. Department of Computer Science.

BIS+14. Antonio Brogi, Ahmad Ibrahim, Jacopo Soldani, José Carrasco, Javier Cubo, Ernesto Pimentel, and Francesco D'Andria. SeaClouds: A European Project on Seamless Management of Multi-Cloud Applications. *ACM SIGSOFT Software Engineering Notes*, 39(1):1–4, January 2014.

BK14a. Peter Bailis and Kyle Kingsbury. The Network is Reliable. *ACM Queue*, 12(7):1–13, July 2014.

BK14b. Christian Bauckhage and Kristian Kersting. Strong Regularities in Growth and Decline of Popularity of Social Media Services. arXiv:1406.6529, June 2014.

BKK+95. Johannes Blömer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. An XOR-Based Erasure-Resilient Coding Scheme. Technical Report TR-95-048, International Computer Science Institute, August 1995.

Bla79. George R. Blakley. Safeguarding cryptographic keys. *Proceedings of the National Computer Conference*, 48:313–317, 1979.

BM12. Kirsten Bock and Sebastian Meissner. Datenschutz-Schutzziele im Recht. *Datenschutz und Datensicherheit - DuD*, 36(6):425–431, June 2012.

BM14a. Ozalp Babaoglu and Moreno Marzolla. The People's Cloud – Escape From the Data Center: The Promise of Peer-to-Peer Cloud Computing. *IEEE Spectrum*, 14(10), October 2014.

BM14b. Vanessa Braganholo and Marta Mattoso. A Survey on XML Fragmentation. *ACM SIGMOD Record*, 43(3):24–35, September 2014.

BMMC14. Marco Baldi, Nicola Maturo, Eugenio Montali, and Franco Chiaraluce. AONT-LT: a Data Protection Scheme for Cloud and Cooperative Storage Systems. In *International Conference on High Performance Computing & Simulation (HPCS 2014) – 4th International Workshop on Security, Privacy and Performance in Cloud Computing (SPCLOUD)*, July 2014. Bologna, Italy.

BMW11. Daniel Beimborn, Thomas Miletzki, and Stefan Wenzel. Platform as a Service (PaaS). *Business and Information Systems Engineering (BISE)*, 3(6):381–384, December 2011.

BRNR15. Tomasz Buchert, Cristian Ruiz, Lucas Nussbaum, and Olivier Richard. A survey of general-purpose experiment management tools for distributed systems. *Future Generation Computer Systems*, 45:1–12, April 2015.

Bro97. Andrei Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, pages 21–29, June 1997. Salerno, Italy.

BS09. Jiang Bian and Remzi Seker. JigDFS: A secure distributed file system. In *IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 76–82, April 2009. Nashville, Tennessee, USA.

BSS+13. Sven Bendel, Thomas Springer, Daniel Schuster, Alexander Schill, Ralf Ackermann, and Michael Ameling. A Service Infrastructure for the Internet of Things based on XMPP. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 385–388, March 2013. San Diego, California, USA.

BSSV10. Vladimir Božović, Daniel Socek, Rainer Steinwandt, and Viktória I. Villányi. Multi-authority attribute based encryption with honest-but-curious central authority. *International Journal of Computer Mathematics*, 89(3):268–283, June 2010.

BYV+09. Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(5):599–616, June 2009.

CBA+06. Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the World, Unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.

CGKO06. Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, pages 79–88, October 2006. Alexandria, Virginia, USA.

CL14. Henry C. H. Chen and Patrick P. C. Lee. Enabling Data Integrity Protection in Regenerating-Coding-Based Cloud Storage: Theory and Implementation. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 25(2):407–416, February 2014.

CLFG14. Jürgen Cito, Philipp Leitner, Thomas Fritz, and Harald C. Gall. The Making of Cloud Applications – An Empirical Study on Software Development for the Cloud. arXiv:1409.6502, September 2014.

CPV+13. Antonio Celesti, Nicola Peditto, Fabio Verboso, Massimo Villari, and Antonio Puliafito. DRACO PaaS: A Distributed Resilient Adaptable Cloud Oriented Platform. In *IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum (IPDPSW)*, pages 1490–1497, 2013. Cambridge, Massachusetts, USA.

CR12. Simon Caton and Omer Rana. Towards autonomic management for Cloud services based upon volunteered resources. *Concurrency and Computation: Practice and Experience*, 24(9):992–1014, June 2012.

CRB+11. Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud com-

puting environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.

CRM06. Pat. P. W. Chan, Michael R.Lyu, and Miroslaw Malek. Making Services Fault Tolerant. In *Third International Service Availability Symposium (ISAS)*, volume 4328 of *LNCS*, pages 43–61, May 2006. Helsinki, Finland.

DdQ11. Dener Didoné and Ruy J. G. B. de Queiroz. Forensic as a Service - FaaS. In *Proceedings of the Sixth International Conference on Forensic Computer Science (ICoFCS)*, pages 202–210, October 2011. Florianópolis, Brazil.

DHR+14. Florian Dudouet, Piyush Harsh, Santiago Ruiz, Andre Gomes, and Thomas Michael Bohnert. A Case for CDN-as-a-Service in the Cloud: A Mobile Cloud Networking Argument. In *3rd International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 651–657, September 2014. Delhi, India.

DMM+12. Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the Internet Measurement Conference (IMC)*, pages 481–494, November 2012. Boston, Massachusetts, USA.

Dus07. Lisa M. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). IETF Request for Comments - RFC 4918, June 2007.

EBB+11. Michael Eckert, François Bry, Simon Brodt, Olga Poppe, and Steffen Hausmann. A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed. In Sven Helmer, Alexandra Poulovassilis, and Fatos Xhafa, editors, *Reasoning in Event-Based Distributed Systems*, volume 347 of *SCI*, pages 47–70. Springer, 2011.

EMAA14. Fatih Emekcia, Ahmed Methwallyb, Divyakant Agrawalb, and Amr El Abbadi. Dividing secrets to secure data outsourcing. *Information Sciences*, 263:198–210, April 2014.

FDH+10. Martin Franz, Björn Deiseroth, Kay Hamacher, Somesh Jha, Stefan Katzenbeisser, and Heike Schröder. Secure Computations on Non-Integer Values. In *2nd IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, December 2010. Seattle, Washington, USA.

FGJ+11. Roberta Ferrario, Nicola Guarino, Christian Janiesch, Tom Kiemes, Daniel Oberle, and Florian Probst. Towards an Ontological Foundation of Services Science: The General Service Model. In *10. Internationale Tagung Wirtschaftsinformatik*, page P47, February 2011. Zurich, Switzerland.

FGM13. Lance Fiondella, Swapna S. Gokhale, and Veena B. Mendiratta. Cloud Incident Data: An Empirical Analysis. In *IEEE International Conference on Cloud Engineering (IC2E)*, pages 241–249, March 2013. Redwood City, California, USA.

FK14. Oleksandr M. Fal' and Volydymyr F. Kozak. Personal Data Protection Problems Associated with Cloud Computing. *Cybernetics and Systems Analysis*, 50(5):768–773, September 2014.

FR14. Roy T. Fielding and Julian F. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. IETF Request for Comments - RFC 7230, June 2014.

FV01. Mirtha-Lina Fernández and Gabriel Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6–7):753–758, May 2001.

FZRL08. Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop (GCE)*, pages 1–10, November 2008. Austin, Texas, USA.

Gal09. Wojciech Galuba. Friend-to-Friend Computing: Building the Social Web at the Internet Edges. Technical Report LSIR-REPORT-2009-003, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2009.

GIC+14a. Sebastian Götz, Thomas Ilsche, Jorge Cardoso, Josef Spillner, Uwe Aßmann, Wolfgang Nagel, and Alexander Schill. Energy-Efficient Data Processing at Sweet Spot Frequencies. In *International Conference Cloud and Trusted Computing (OnTheMove C&TC)*, volume 8842 of *LNCS*, pages 154–171, October 2014. Amantea, Italy.

GIC+14b. Sebastian Götz, Thomas Ilsche, Jorge Cardoso, Josef Spillner, Thomas Kissinger, Uwe Aßmann, Wolfgang Lehner, Wolfgang E. Nagel, and Alexander Schill. Energy-Efficient Databases using Sweet Spot Frequencies. In *1st International Workshop on Green Cloud Computing (GCC)*, pages 871–876, December 2014. London, UK.

GKL15. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *34th Annual International Conference on the Theory and Applications of Cryptographic Techniques - Advances in Cryptology (EUROCRYPT)*, volume 9057 of *LNCS*, pages 281–310, April 2015.

GLH15. Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*, volume 72 of *Intelligent Systems Reference Library*. Springer, 1st edition, 2015.

GS14. Francis Gropengießer and Kai-Uwe Sattler. Database Backend as a Service: Automatic Generation, Deployment, and Management of Database Backends for Mobile Applications. *Datenbank-Spektrum*, 14(2):85–95, July 2014.

Gö13. Sebastian Götz. *Multi-Quality Auto-Tuning by Contract Negotiation*. PhD thesis, Technische Universität Dresden, August 2013. Faculty of Computer Science.

Had15. Makhlouf Hadji. A Mathematical Programming Approach to Multi-Cloud Storage. In *5th International Conference on Cloud Computing and Services Science (CLOSER)*, May 2015. Lisbon, Portugal.

HCC11. Ching-Hsien Hsu, Alfredo Cuzzocrea, and Shih-Chang Chen. CAD: An Efficient Data Management and Migration Scheme across Clouds for Data-Intensive Scientific Applications. In *4th International Conference Globe*, volume 6864 of *LNCS*, pages 120–134, September 2011. Toulouse, France.

Hey08. Karen Heyman. The Move to Make Social Data Portable. *IEEE Computer*, 41(4):13–15, April 2008.

HHM06. T. Hansen, T. Hardie, and L. Masinter. Guidelines and Registration Procedures for New URI Schemes. IETF Request for Comments - RFC 4395 / BCP 35, February 2006.

HIM14. Hakan Hacıgümüş, Bala Iyer, and Sharad Mehrotra. Secure Computation on Outsourced Data: A 10-year Retrospective. In *19th International Conference on Database Systems for Advanced Applications (DASFAA)*, volume 8421 of *LNCS*, pages 16–27, April 2014. Bali, Indonesia.

HKS14. Wolfgang A. Halang, Maytiyanin Komkhao, and Sunantha Sodsee. Secure Cloud Computing. In *Proceedings of the 10th International Conference on Computing and Information Technology (IC2IT)*, volume 265 of *Advances in Intelligent Systems and Computing*, pages 305–314, May 2014. Phuket, Thailand.

HWWF+14. Marc Herrlich, Dirk Wenig, Benjamin Walther-Franks, Jan D. Smeddinck, and Rainer Malaka. Raus aus dem Sessel - Computerspiele für mehr Gesundheit. *Informatik-Spektrum*, 37(6):558–566, December 2014.

JKF12. Daniel Janusz, Martin Kost, and Johann-Christoph Freytag. Privacy Protocol for Linking Distributed Medical Data. In *9th Workshop on Secure Data Management (SDM) at the 38th VLDB*, volume 7482 of *LNCS*, pages 45–57, August 2012. Istanbul, Turkey.

JL13. Kai Jander and Winfried Lamersdorf. Jadex WfMS: Distributed Workflow Management for Private Clouds. In *1st International Conference on Networked Systems (NetSys)*, pages 84–91, March 2013. Stuttgart, Germany.

Jos03. Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. IETF Request for Comments - RFC 3548, July 2003.

KBS14. Florian Kerschbaum, Martin Beck, and Dagmar Schönfeld. Inference Control for Privacy-Preserving Genome Matching. arXiv:1405.0205, June 2014.

Ker11. Florian Kerschbaum. Secure and Sustainable Benchmarking in Clouds – A Multi-Party Cloud Application with an Untrusted Service Provider. *Business and Information Systems Engineering (BISE)*, 3(3):135–143, June 2011.

KML14. Nesrine Kaaniche, Ethmane El Moustaine, and Maryline Laurent. A Novel Zero-Knowledge Scheme for Proof of Data Possession in Cloud Storage Applications. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 522–531, May 2014. Chicago, Illinois, USA.

KR10. Eric Keller and Jennifer Rexford. The "Platform as a Service" Model for Networking. In *Internet Network Management Workshop / Workshop on Research on Enterprise Networking (INM/WREN)*, pages 1–6, April 2010. San Jose, California, USA.

Kri13. Chandra Krintz. The AppScale Cloud Platform: Enabling Portable, Scalable Web Application Deployment. *Internet Computing*, 17(2):72–75, March-April 2013.

KSHD13. Steffen Kächele, Christian Spann, Franz J. Hauck, and Jörg Domaschka. Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking. In *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pages 75–82, December 2013. Dresden, Germany.

Lam15. Harald Lampesberger. Technologies for Web and cloud service interaction: a survey. arXiv:1408.2751v3, March 2015.

LAS+15. Jieyao Liu, Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani, Rajkumar Buyya, and Ahsan Qureshi. Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, 48:99–117, February 2015.

LMRD08. Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. End-to-End Versioning Support for Web Services. In *IEEE International Conference on Services Computing (SCC)*, pages 59–66, July 2008. Honolulu, Hawaii, USA.

LORZ13. Zheng Li, Liam O'Brian, Rajiv Ranjan, and Miranda Zhang. Early Observations on Performance of Google Compute Engine for Scientific Computing. In *5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 1–8, December 2013. Bristol, United Kingdom.

LS10. Wolfgang Lehner and Kai-Uwe Sattler. Database as a Service (DBaaS). In *26th IEEE International Conference on Data Engineering (ICDE)*, pages 1216–1217, March 2010. Long Beach, California, USA.

Lu12. Yanbin Lu. Privacy-preserving Logarithmic-time Search on Encrypted Data in Cloud. In *19th Annual Network & Distributed System Security Symposium*, February 2012. San Diego, California, USA.

MBF14. André Martin, Andrey Brito, and Christof Fetzer. Scalable and Elastic Realtime Click Stream Analysis using StreamMine3G. In *The 8th ACM International Conference on Distributed Event-Based Systems (DEBS)*, pages 198–205, May 2014. Mumbai, India.

MG09. Peter Mell and Tim Grance. The NIST Definition of Cloud Computing version 15. National Institute of Standards and Technology, Maryland, USA, 2009.

Mic15. James Bret Michael. Trusted Computing: An Elusive Goal. *IEEE Computer*, 48(3):99–101, March 2015.

MM13. Antonio Muñoz and Antonio Mana. Bridging the GAP between Software Certification and Trusted Computing for Securing Cloud Computing. In *Ninth IEEE World Congress on Services (SERVICES)*, pages 103–110, June 2013. Santa Clara, California, USA.

Mon14. Gregory Mone. The New Digital Medicine. *Communications of the ACM*, 57(9):18–20, September 2014.

MP11. Tony Mancill and Orest Pilskalns. Combining Encryption and Compression in Wireless Sensor Networks. *International Journal of Wireless Information Networks*, 18(1):39–49, March 2011.

MPP14. Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Evaluating Geospatial Geometry and Proximity Queries Using Distributed Hash Tables. *Computing in Science & Engineering*, 16(4):53–61, July-August 2014.

MPP15. Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Fast, Ad Hoc Query Evaluations over Multidimensional Geospatial Datasets. *IEEE Transactions on Cloud Computing*, 2015.

MSMF14. Christoph Meinel, Maxim Schnjakin, Tobias Metzke, and Markus Freitag. Anbieter von Cloud-Speicherdiensten im Überblick. Technical Report 84, Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 2014.

Nit13. André Nitze. Cloud-basierte Architekturen mobiler Anwendungen. In *8. Workshop Bewertungsaspekte service- und cloudbasierter Architekturen (BSOA/BCloud)*, pages 1–5, November 2013. Basel, Switzerland.

NSHS14. The An Binh Nguyen, Melanie Siebenhaar, Ronny Hans, and Ralf Steinmetz. Role-based Templates for Cloud Monitoring. In *7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pages 242–250, December 2014. London, UK.

NT05. Gonzalo Navarro and Jorma Tarhio. LZgrep: a Boyer–Moore String Matching Tool for Ziv–Lempel Compressed Text. *Software: Practice and Experience*, 35(12):1107–1130, October 2005.

NVB+14. Dimitrios S. Nikolopoulos, Hans Vandierendonck, Nikolaos Bellas, Christos D. Antonopoulos, Georgios Karakonstantis, Andreas Burg, and Uwe Naumann. Energy Efficiency through Significance-Based Computing. *IEEE Computer*, 47(7):82–85, July 2014.

Obe14. Daniel Oberle. Ontologies and Reasoning in Enterprise Service Ecosystems. *Informatik-Spektrum*, 37(4):318–328, August 2014.

Pai99. Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - 17th Annual IACR Eurocrypt Conference (EUROCRYPT)*, volume 1592 of *LNCS*, pages 223–238, May 1999. Prague, Czech Republic.

PB10. Siani Pearson and Azzedine Benameur. Privacy, Security and Trust Issues Arising from Cloud Computing. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 693–702, November 2010. Indianapolis, Indiana, USA.

PCL14. Carlos Pedrinaci, Jorge Cardoso, and Torsten Leidig. Linked USDL: A Vocabulary for Web-Scale Service Trading. In *The Semantic Web: Trends and Challenges – 11th International Conference on ESWC (née European/Extended Semantic Web Conference)*, volume 8465 of *LNCS*, pages 68–82, May 2014. Anissaras, Crete, Greece.

PH03. Niels Provos and Peter Honeyman. Hide and Seek: An Introduction to Steganography. *IEEE Security and Privacy*, 1(3):32–44, May 2003.

PJGLSAH11. Lluis Pamies-Juarez, Pedro García-López, Marc Sánchez-Artigas, and Blas Herrera. Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures. *Comput. Netw.*, 55(5):1100–1113, April 2011.

Pla13. James S. Plank. Erasure Codes for Storage Systems. *;login: The USENIX Magazine*, 38(6):44–50, December 2013.

PP15. Pekka Pääkkönen and Daniel Pakkala. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research*, February 2015.

PT09. Dmitry L. Petrov and Yury S. Tatarinov. Data migration in the scalable storage cloud. In *International Conference on Ultra Modern Telecommunications & Workshops*, pages 1–4, October 2009. St. Petersburg, Russia.

PTDL08. Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: A Research Roadmap. *International Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

PZCG14. Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies; Special Issue: Smart Cities – Trends & Technologies*, 25(1):81–93, January 2014.

Rab89. Michael O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the ACM*, 36(2):335–348, April 1989.

RAG14. Oscar J. Rubio, Alvaro Alesanco, and José García. Secure and efficient coding of biomedical signals, periodic measurements and contextual data in body area networks. In *2nd IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 231–234, June 2014. Valencia, Spain.

RC15. Nurul Hidayah Ab Rahman and Kim-Kwang Raymond Choo. A survey of information security incident handling in the cloud. *Computers & Security*, 49:45–69, March 2015.

RFB+14. Massimiliano Rak, Massimo Ficco, Ermanno Battista, Valentina Casola, and Nicola Mazzocca. Developing Secure Cloud Applications. *Scalable Computing: Practice and Experience (SCPE)*, 15(1):49–62, January 2014.

Riv97. Ronald L. Rivest. All-or-Nothing Encryption and the Package Transform. In *4th International Workshop on Fast Software Encryption (FSE)*, volume 1267 of *LNCS*, pages 210–218, January 1997. Haifa, Israel.

ROCW14. Mathew Ryden, Kwangsung Oh, Abhishek Chandra, and Jon Weissman. Nebula: Distributed Edge Cloud for Data Intensive Computing. In *IEEE International Conference on Cloud Engineering (IC2E)*, March 2014. Boston, Massachusetts, USA.

RP11. Jason K. Resch and James S. Plank. AONT-RS: Blending Security and Performance in Dispersed Storage Systems. In *9th USENIX Conference on File and Storage Technologies*, February 2011. San Jose, California, USA.

RS60. I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

SAAW12. Ilan Shomorony, A. Salman Avestimehr, Himanshu Asnani, and Tsachy Weissman. Worst-Case Source for Distributed Compression with Quadratic Distortion. arXiv:1208.1784, August 2012.

SAP+13. Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. *The 39th International Conference on Very Large Data Bases (VLDB)*, 6(5):325–336, August 2013. Riva del Garda, Trento, Italy.

SB14. Jakub Szefer and Sebastian Biedermann. Towards Fast Hardware Memory Integrity Checking with Skewed Merkle Trees. In *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, pages 1–8, June 2014. Minneapolis, Minnesota, USA.

SBBS12a. Josef Spillner, Andrey Brito, Francisco Brasileiro, and Alexander Schill. A Highly-Virtualising Cloud Resource Broker. In *5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pages 233–234, November 2012. Chicago, Illinois, USA (Poster).

SBBS12b. Josef Spillner, Andrey Brito, Francisco Brasileiro, and Alexander Schill. Analysis of Overhead and Profitability in Nested Cloud Environments. In *1st Latin American Conference on Cloud Computing and Communications (LatinCloud)*, pages 13–18, November 2012. Porto Alegre, Brazil.

SBM+11. Josef Spillner, Gerd Bombach, Steffen Matthischke, Johannes Müller, Rico Tzschichholz, and Alexander Schill. Information Dispersion over Redundant Arrays of Optimal Cloud Storage for Desktop Users. In *4th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pages 1–8, December 2011. Melbourne, Australia.

SBS09. Josef Spillner, Iris Braun, and Alexander Schill. Towards Unified Service Hosting. In *INSTICC-Tagungsband: 4th International Conference on Software and Data Technologies*, volume 2, pages 31–36, July 2009. Sofia, Bulgaria.

SBS11. Josef Spillner, Iris Braun, and Alexander Schill. Unified Hosting Techniques for the Internet of Tradeable and Executable Services. In José Cordeiro, AlpeshKumar

Ranchordas, and Boris Shishkov, editors, *Software and Data Technologies - Revised Selected Papers*, volume 50-2 of *Communications in Computer and Information Science*, pages 35–45. Springer Berlin Heidelberg, March 2011.

SBS13. Josef Spillner, Sven Bendel, and Alexander Schill. Towards Flexible Data Sharing in Multi-Device Mobility Scenarios. In *3rd International Conference on Selected Topics in Mobile & Wireless Networking (MoWNet)*, pages 80–85, August 2013. Montréal, Canada.

SCB+13. Josef Spillner, Andrii Chaichenko, Andrey Brito, Francisco Brasileiro, and Alexander Schill. Operating the Cloud from Inside Out. In *HPI Cloud Symposium - Operating the Cloud*. arXiv:1309.5442, September 2013. Potsdam, Germany.

SCB+14. Josef Spillner, Andrii Chaichenko, Andrey Brito, Francisco Brasileiro, and Alexander Schill. Cloud Resource Recycling: An Addition of Species to the Zoo of Virtualised, Overlaid, Federated, Multiplexed and Nested Clouds. *SDPS Transactions: Journal of Integrated Design and Process Science (JIDPS)*, 18(1):5–19, April 2014. DOI: 10.3233/jid-2014-0006.

SCF+14. Mohsen Amini Salehi, Thomas Caldwell, Alejandro Fernandez, Emmanuel Mickiewicz, Eric W. D. Rozier, Saman Zonouz, and David Redberg. RESeED: Regular Expression Search over Encrypted Data in the Cloud. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 538–539, May 2014. Chicago, Illinois, USA.

SD13. Xiuli Song and Hongyao Deng. Lightweight Proofs of Retrievability for Electronic Evidence in Cloud. *MDPI Information*, 4(3):262–282, July 2013.

SG14. Andy Sayler and Dirk Grunwald. Custos: Increasing Security with Secret Storage as a Service. In *USENIX Conference on Timely Results in Operating Systems (TRIOS)*, October 2014. Broomfield, Colorado, USA.

SGI13. Christoph Sorge, Nils Gruschka, and Luigi Lo Iacono. *Sicherheit in Kommunikationsnetzen*. De Gruyter Oldenbourg, 1st edition, 2013.

SGLM08. Mark W. Storer, Kevin Greenan, Darrell D. E. Long, and Ethan L. Miller. Secure Data Deduplication. In *4th International Workshop on Storage Security and Survivability (StorageSS)*, October 2008. Alexandria, Virginia, USA.

SGS11. Ronny Seiger, Stephan Groß, and Alexander Schill. SecCSIE: A Secure Cloud Storage Integrator for Enterprises. In *International Workshop on Clouds for Enterprises (C4E)*, pages 252–255, September 2011. Luxembourg, Luxembourg.

Sha79. Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

Sho02. Martin L. Shooman. *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*. Wiley, 1st edition, 2002.

Sim79. Gustavus J. Simmons. Symmetric and Asymmetric Encryption. *ACM Computing Surveys (CSUR)*, 11(4):305–330, December 1979.

SIS13. Josef Spillner, Stefan Illgen, and Alexander Schill. Engineering Service Level Agreements: A Constrained-Domain and Transformation Approach. In *3rd International Conference on Cloud Computing and Services Science (CLOSER)*, pages 395–405, May 2013. Aachen, Germany.

SK76. Robert L. Solso and Joseph F. King. Frequency and versatility of letters in the English language. *Behavior Research Methods & Instrumentation*, 8(3):283–286, May 1976.

SK14. Mathias Slawik and Axel Küpper. A Domain Specific Language and a Pertinent Business Vocabulary for Cloud Service Selection. In *11th International Conference on Economics of Grids, Clouds, Systems and Services (GECON) - Revised Selected Papers*, volume 8914 of *LNCS*, pages 172–185, September 2014. Cardiff, UK.

SKK+14. Sebastian Schrittwieser, Stefan Katzenbeisser, Peter Kieseberg, Markus Huber, Manuel Leithner, Martin Mulazzani, and Edgar Weippl. Covert Computation – Hiding code in code through compile-time obfuscation. *Computers & Security*, 42:13–26, May 2014.

SKS12. Josef Spillner, Ronny Kursawe, and Alexander Schill. Experience Report on Real-World Manual Service Modelling in USDL. In Alistair Barros and Daniel Oberle, editors, *Handbook of Service Description - USDL and its Methods*, pages 487–501. Springer Berlin Heidelberg, March 2012.

SKU+11. Josef Spillner, Anne Kümpel, Stefan Uhlig, Iris Braun, and Alexander Schill. An Integrated Provisioning Toolchain for the Internet of Services. In *Proceedings of the 10th IADIS International Conference WWW/Internet*, pages 566–570, November 2011. Rio de Janeiro, Brazil.

SM14a. Josef Spillner and Johannes Müller. PICav: Precise, Iterative and Complement-based Cloud Storage Availability Calculation Scheme. In *7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pages 443–450, December 2014. London, UK.

SM14b. Josef Spillner and Johannes Müller. Project Report: Statistical Analysis of Cloud Storage. Technical report, Universität Potsdam, Hasso-Plattner-Institut für Softwaresystemtechnik, 2014. Erscheint im Jahr 2015.

SMK15. Klaus M. Schneider, Kai Mast, and Udo R. Krieger. CCN Forwarding Strategies for Multihomed Mobile Terminals. In *2nd International Conference on Networked Systems (NetSys)*, March 2015. Cottbus, Germany.

SMS13. Josef Spillner, Johannes Müller, and Alexander Schill. Creating Optimal Cloud Storage Systems. *Future Generation Computer Systems*, 29(4):1062–1072, June 2013. DOI: http://dx.doi.org/10.1016/j.future.2012.06.004.

SMS15. Josef Spillner, Lorenzo Miori, and Julian Sanin. Stealth Apps for Secure Personal Data Analytics in the Cloud. In *2nd International Conference on Networked Systems (NetSys) - Communication Software Award Demo*, March 2015. Cottbus, Germany (Vorführung).

SP07. Peter Sobe and Kathrin Peter. Combining Compression, Encryption and Fault-tolerant Coding for Distributed Storage. In *21st IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, March 2007. Long Beach, California, USA.

Spi09. Josef Spillner. Service Orientation in Middleware Components for Scalable Service Marketplaces. In *19th International Crimean Conference on Microwave and Telecommunication Technology (*КрыМиКо*)*, volume 1, pages 360–361, September 2009. Sevastopol, Crimea - Ukraine.

Spi10. Josef Spillner. *Methodik und Referenzarchitektur zur inkrementellen Verbesserung der Metaqualität einer vertragsgebundenen, heterogenen und verteilten Dienstausführung*. PhD thesis, Technische Universität Dresden, September 2010. Faculty of Computer Science.

Spi11a. Josef Spillner. An Environment for Educational Service Communities. *International Research Journal of Telecommunication Sciences*, 2(2):22–27, December 2011.

Spi11b. Josef Spillner. SPACEflight - A Versatile Live Demonstrator and Teaching System for Advanced Service-Oriented Technologies. In *21st International Crimean Conference on Microwave and Telecommunication Technology (*КрыМиКо*)*, volume 1, pages 455–456, September 2011. Sevastopol, Crimea - Ukraine.

Spi14. Josef Spillner. Project Report: Dispersed Data Processing Services for Third-Party Applications. Technical report, Universität Potsdam, Hasso-Plattner-Institut für Softwaresystemtechnik, 2014. Erscheint im Jahr 2015.

Spi15. Josef Spillner. Secure Distributed Data Stream Analytics in Stealth Applications. In *3rd IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, May 2015. Constanța, Romania. Angenommen zur Veröffentlichung.

SPM12. Călin Sandru, Dana Petcu, and Victor Ion Munteanu. Building an Open-Source Platform-as-a-Service with Intelligent Management of Multiple Cloud Resources. In *5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, November 2012. Chicago, Illinois, USA.

SPW+12. Josef Spillner, Christian Piechnick, Claas Wilke, Uwe Aßmann, and Alexander Schill. Autonomous Participation in Cloud Services. In *2nd International Workshop on Intelligent Techniques and Architectures for Autonomic Clouds (ITAAC)*, pages 289–294, November 2012. Chicago, Illinois, USA.

SQFS13. Josef Spillner, Maximilian Quellmalz, Martin Friedrich, and Alexander Schill. peaCS - Performance and Efficiency Analysis for Cloud Storage. In *ESOCC Workshop on Cloud Storage Optimization (CLOUSO)*, volume 393 of *CCIS*, pages 47–58, September 2013. Málaga, Spain.

SS12a. Josef Spillner and Alexander Schill. Flexible Data Distribution Policy Language and Gateway Architecture. In *1st Latin American Conference on Cloud Computing and Communications (LatinCloud)*, pages 1–6, November 2012. Porto Alegre, Brazil.

SS12b. Josef Spillner and Alexander Schill. π-Control: A Personal Cloud Control Centre. arXiv:1202.0970, February 2012.

SS13a. Josef Spillner and Alexander Schill. A Versatile and Scalable Everything-as-a-Service Registry and Discovery. In *3rd International Conference on Cloud Computing and Services Science (CLOSER)*, pages 175–183, May 2013. Aachen, Germany.

SS13b. Josef Spillner and Alexander Schill. Orchestration of Distributed Storage Targets through Storage Flows. In *5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 349–354, December 2013. Bristol, United Kingdom.

SS14a. Josef Spillner and Alexander Schill. Algorithms for Dispersed Processing. In *1st International Workshop on Advances in Cloud Computing Legislation, Accountability, Security and Privacy (CLASP)*, pages 914–921, December 2014. London, UK.

SS14b. Josef Spillner and Alexander Schill. Towards Dispersed Cloud Computing. In *2nd IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 175–179, May 2014. Chișinău, Moldova.

SSZ13. Josef Spillner, Johannes Schad, and Stephan Zepezauer. Personal and Federated Cloud Management Cockpit. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 36(1):44, February 2013. DOI: 10.1515/pik-2012-0149.

ST10. Peter Sempolinski and Douglas Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *Second IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 417–426, December 2010. Indianapolis, Indiana, USA.

STS14. Josef Spillner, Sebastian Tilsch, and Alexander Schill. NubiVis: A Personal Cloud File Explorer. In *11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, pages 354–356, December 2014. London, UK (Vorführung).

SUSS13. Josef Spillner, Anna Utlik, Thomas Springer, and Alexander Schill. RAFT-REST - A Client-side Framework for Reliable, Adaptive and Fault-Tolerant RESTful Service Consumption. In *2nd European Conference on Service-Oriented and Cloud Computing (ESOCC)*, volume 8135 of *LNCS*, pages 104–118, September 2013. Málaga, Spain.

SWP00. Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches over Encrypted Data. In *IEEE Symposium on Security and Privacy (S&P)*, pages 44–55, May 2000. Berkeley, California, USA.

SZS13. Johannes Schad, Stephan Zepezauer, and Josef Spillner. Personal Cloud Management Cockpit with Social or Market-Driven Asset Exchange. In *Networked Systems Conference (NetSys/KiVS) - Communication Software Award Demo*, March 2013. Stuttgart, Germany (Vorführung).

TKMZ13. Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing Analytical Queries over Encrypted Data. In *Proceedings of the 39th In-*

ternational Conference on Very Large Data Bases (VLDB), volume 6(5) of *VLDB Endowment*, pages 1–12, August 2013. Riva del Garda, Italy.

TNL+12. Yehia Taher, Dinh Khoa Nguyen, Francesco Lelli, Willem-Jan van den Heuvel, and Mike Papazoglou. On Engineering Cloud Applications - State of the Art, Shortcomings Analysis, and Approach. *Scalable Computing: Practice and Experience (SCPE)*, 13(3):215–231, September 2012.

TPH+14. Yvonne Thoss, Christoph Pohl, Madlain Hoffmann, Josef Spillner, and Alexander Schill. User-friendly Visualization of Cloud Quality. In *7th IEEE International Conference on Cloud Computing (CLOUD)*, pages 890–897, July 2014. Anchorage, Alaska, USA.

UMJ+14. Jacopo Urbani, Alessandro Margara, Ceriel Jacobs, Spyros Voulgaris, and Henri Bal. AJIRA: A Lightweight Distributed Middleware for MapReduce and Stream Processing. In *34th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 545–554, July 2014. Madrid, Spain.

uRLW+15. Muhammad Habib ur Rehman, Chee Sun Liew, Teh Ying Wah, Junaid Shuja, and Babak Daghighi. Mining Personal Data Using Smartphones and Wearable Devices: A Survey. *MDPI Sensors*, 15(2):4430–4469, February 2015.

UU12. Michael Ulbrich and Stefan Ulbrich. *Nichtlineare Optimierung*. Birkhäuser, 2012.

VBS11. Elisabeth Vinek, Peter Paul Beran, and Erich Schikuta. Classification and Composition of QoS Attributes in Distributed, Heterogeneous Systems. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 424–433, May 2011. Newport Beach, California, USA.

VN10. Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing Cloud Computing Hardware Reliability. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC)*, pages 193–203, June 2010. Indianapolis, Indiana, USA.

VPT+12. Quang Hieu Vu, Tran-Vu Pham, Hong-Linh Truong, Schahram Dustdar, and Rasool Asal. DEMODS: A Description Model for Data-as-a-Service. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, March 2012. Fukuoka, Japan.

Wes06. Andreas Westfeld. Steganalysis in the Presence of Weak Cryptography and Encoding. In *International Workshop on Digital Watermarking (IWDW)*, volume 4283 of *LNCS*, pages 19–34, November 2006. Jeju Island, Korea.

WLMS14. Olga Wenge, Ulrich Lampe, Alexander Müller, and Ralf Schaarschmidt. Data Privacy in Cloud Computing – An Empirical Study in the Financial Industry. In *Proceedings of the 20th Americas Conference on Information Systems (AMCIS)*, pages 1–10, August 2014. Savannah, Georgia, USA.

WSJ15. Roy Want, Bill N. Schilit, and Scott Jenson. Enabling the Internet of Things. *IEEE Computer*, 48(1):28–35, January 2015.

WSS14. Olga Wenge, Dieter Schuller, and Ralf Steinmetz. Towards Establishing Security-Aware Cloud Markets. In *Proceedings of 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 1027–1032, December 2014. Singapore.

WTJ14. Stefan Walraven, Eddy Truyen, and Wouter Joosen. Comparing PaaS offerings in light of SaaS development. *Computing*, 96(8):669–724, August 2014.

WWG14. Kyle Williams, Jian Wu, and C. Lee Giles. SimSeerX: A Similar Document Search Engine. In *ACM Symposium on Document Engineering (DocEng)*, pages 143–146, September 2014. Fort Collins, Colorado, USA.

YMSG+14. Marcelo Yannuzzi, Rodolfo A. Milito, René Serral-Gracià, D. Montero, and Mario Nemirovsky. Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing. In *19th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 325–329, December 2014. Athens, Greece.

YNMT15. Yoji Yamato, Yukihisa Nishizawa, Masahito Muroi, and Kentaro Tanaka. Development of resource management server for production IaaS services based on OpenStack. *Journal of Information Processing*, 23(1):58–66, January 2015.

YPLL14. Hui-Shyong Yeo, Xiao-Shen Phang, Hoon-Jae Lee, and Hyotaek Lim. Leveraging client-side storage techniques for enhanced use of multiple consumer cloud storage services on resource-constrained mobile devices. *Journal of Network and Computer Applications*, 43(1):142–156, August 2014.

YYC13. Dong Yuan, Yun Yang, and Jinjun Chen. *Computation and Storage in the Cloud: Understanding the Trade-Offs*. Elsevier, 1st edition, 2013.

ZLHD14. Rostyslav Zabolotnyi, Philipp Leitner, Waldemar Hummer, and Schahram Dustdar. JCloudScale: Closing the Gap Between IaaS and PaaS. arXiv:1411.2392, November 2014.

ZMS14. Elżbieta Zielińska, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Trends in steganography. *Communications of the ACM*, 57(3):86–95, March 2014.

ZSLB14. Liang Zhao, Sherif Sakr, Anna Liu, and Athman Bouguettaya. *Cloud Data Management*. Springer, 1st edition, 2014.

ZW98. Marvin V. Zelkowitz and Dolores R. Wallace. Experimental models for validating technology. *IEEE Computer*, 31(5):23–31, May 1998.

Anhang A
Symbole und Notationen

Symbol	Bedeutung
d	Daten, Eingangsdaten
$d\prime$	Ausgangsdaten
$I(d)$	Informationsgehalt von Daten (0..1)
z	Zahl der Kodierungsschritte (1..∞)
$T_i(d)$	Kodierungsschritt i (i: 0..$z-1$)
n	Zahl der Fragmente, Kardinalität von $d\prime$
k	Zahl signifikanter Fragmente ($n-m$, 1..∞)
$\tilde{k}$	Zahl notwendiger Fragmente ($\leq k$)
m	Zahl redundanter Fragmente ($n-k$, 0..∞)
w	Block- bzw. Chunkgröße in Bits (n..∞)
fw_i	Fragmentgröße i (i: 1..n; 0 imaginär)
f_i	Fragment i (i: 1..n; 0 imaginär)
F	Fragmentmenge $\frown \{f_1,\ldots,f_n\}$ mit Größe w
$\overleftarrow{f_i}$	kumulierte Fragmentmenge $f_i..f_k$
$\mathscr{W}$	Hamminggewicht einer Fragmentmenge (0..w)
$\ulcorner\mathscr{W}\urcorner$	Hamminggewichtskorridor ($\mathscr{W}_{min}..\mathscr{W}_{max}$)
$\mathfrak{B}$	Bit mit Belegung 0/1
h	Zahl der Ressourcenziele; Speicher- und Übertragungsziele (n..∞)
s_i	Ressourcenziel oder Dienst i (i: 1..h)
a, $\hat{a}$	Verfügbarkeit, Effektivverfügbarkeit
c, $\hat{c}$	Kapazität, Effektivkapazität
p, $\hat{p}$	Preis, Effektivpreis
$\tilde{a}$	Mindestverfügbarkeit
H, H^{-1}	homomorphe Ver- und Entschlüsselung
$\mathscr{V}$	System ohne Schutzeigenschaften
$\mathscr{S}$	System mit umfänglichen Schutzeigenschaften
α, β, $\hat{X}$	Daten- und Ergebnismengen

Anhang B
Software-Beiträge für native Cloud-Anwendungen

Die in diesem Kapitel vorgestellten Softwareanwendungen, Bibliotheken, Werkzeuge, Dateisysteme und Abbilder virtueller Maschinen sind im Zeitraum 2010 bis 2015 im Cloud Storage Lab (http://lab.nubisave.org/) als konkrete und nachnutzbare Open-Source-Artefakte mit teilweise einem, teilweise mehreren Beitragenden als Grundlage für die integrierte praktische Validierung der in dieser Arbeit vorgestellten Konzepte implementiert worden [Spi11b]. Die detaillierte Autorenschaft lässt sich zusammen mit dem Umfang und dem Stand der Entwicklung aus der Versionshistorie der jeweiligen Repositorien ableiten. Die Vorführung der Anwendungen als Demonstratoren auf Fachtagungen wird durch entsprechende Referenzen belegt. Durch die interaktive Benutzbarkeit eignen sie sich für einen Transfer der Konzepte sicherer verteilter Anwendungen in die Lehre.

Im Einzelnen stellt dieses Kapitel die folgenden Softwareartefakte aus der Perspektive des Anwenders vor: Einen Simulator für Multi-Cloud-Storage (MCS-SIM), einen Editor für nutzerdefinierte Speicherflüsse respektive *Software-Defined Storage* (NubiSave Storage Flow Editor), eine Speicherdienstintegration für offene Desktopsysteme (NubiSave) und eine für restriktivere Mobilsysteme (NubiDroid), eine Visualisierung verteilter Datenbestände (NubiVis), die protokollspezifische Übertragung von Dateien an Speicherdienste (CloudFusion) und schließlich eine verteilte Datenbankverwaltung (StealthDB). Abschließend wird eine integrierte Softwareperspektive vorgestellt, welche die Mehrzahl der Anwendungen zueinander in Bezug setzt.

Verfügbarkeitssimulation für dienstübergreifende Datenspeicherung

Repositorium: git://nubisave.org/git/mcssimulation

Die Simulationsumgebung *Multi Cloud Storage Simulator* (MCS-SIM) ermöglicht die Bestimmung der Gesamtverfügbarkeit von Daten zu einer gegebenen Verteilung

auf Dienste sowie im umgekehrten Fall die Bestimmung der Nutzung von Diensten und der Verteilung auf Dienste zu einer gegebenen Mindestverfügbarkeit. Neben der Verfügbarkeit können auch weitere Parameter wie Kosten, Platzbedarf und Laufzeit des Bestimmungsverfahrens als Einschränkung eingestellt werden. Durch die Einblendung der Ausführungsschritte des jeweils eingestellten Bestimmungsalgorithmus über unterschiedlich hohe Detailstufen gestattet die Simulationsumgebung dem Benutzer die Nachvollziehbarkeit der Berechnungen und bietet somit einen didaktischen Mehrwert gegenüber vergleichbaren Softwareansätzen, welche zumeist ohne Berücksichtigung von Benutzern konzipiert worden sind oder keine Modellabbildung spezifischer Speicherdiensteigenschaften aufweisen [CRB+11].

MCS-SIM ist in Python 2.x implementiert und besteht aus mehreren Berechnungs- und Experimentierskripten sowie der gleichnamigen grafischen Oberfläche MCS-SIM. Die Software kann sowohl fiktive Dienste zufallsbasiert generieren als auch vormodellierte Dienste per Dienstbeschreibungsdatei im INI-Format einlesen.

Abbildung B.1 zeigt einen Screenshot der typischen Nutzung von MCS-SIM. Hierbei wird über das PICav-Verfahren die Fragmentverteilung zu einer Mindestverfügbarkeit von 98% bestimmt, wobei fünf Speicherdienstanbieter mit ihren realen Verfügbarkeiten zur Auswahl stehen [SM14a]. Im Ergebnis schlägt der Simulator vor, auf alle fünf Dienste signifikante und auf die vier zuverlässigsten Dienste zudem redundante Fragmente abzulegen, was einem Aufteilungsverhältnis von $k = 5$ und $m = 4$ entspricht.

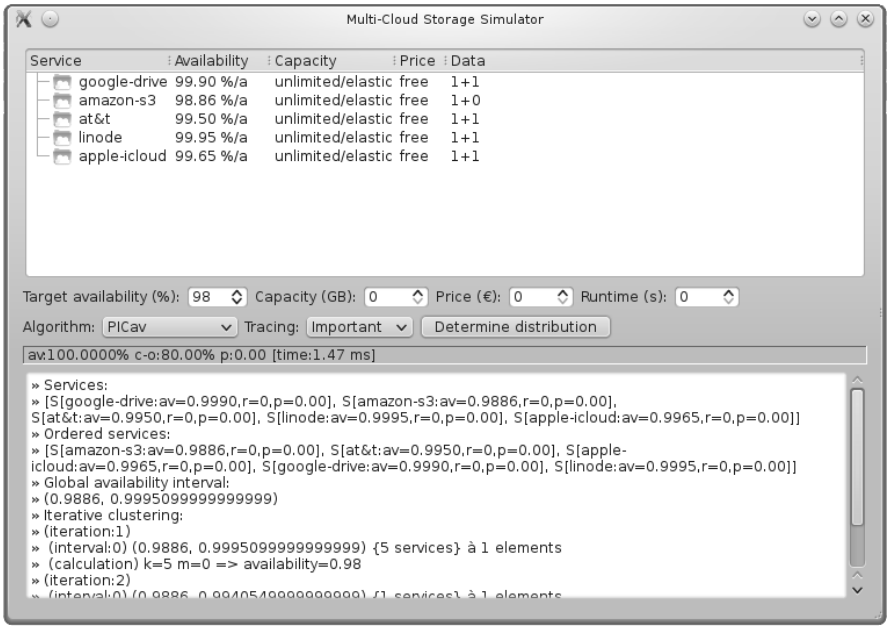

Abb. B.1 Multi Cloud Storage Simulator (MCS-SIM)

Editor für Speicherflüsse

Repositorium: git://nubisave.org/git/nubisave
Vorführungen: [SZS13, SS13b]

Der Speicherflusseditor ermöglicht Benutzern, eine verkettete Kodierung von Daten sowie eine Verteilung auf Speicherziele im Sinne einer vollständigen Speicherflusskette respektive eines Speicherflussbaumes festzulegen. Als Daten werden hierbei Dateien betrachtet, die ausgehend von einem Eingangsverzeichnis beispielsweise dedupliziert, komprimiert, verschlüsselt und anschließend dispergiert werden. Die Speicherziele umfassen lokale Verzeichnisse, bekannte Speicherdienstanbieter sowie generische Protokolle ohne vorherige Festlegung des Anbieters oder des Geräts.

Der Editor ist in Java implementiert und nutzt sowohl die Standard-AWT-/Swing-Klassen als auch Erweiterungen wie Jung oder DJNativeSwing als Widgetbibliotheken. Er interagiert mit einem Dateisystem wie dem im nächsten Abschnitt vorgestellten NubiSave-Splitter oder FlowFS [SS13b] zur Durchführung von Speicherflussrekonfigurationen basierend auf geschriebenen, umgeschriebenen oder gelöschten Dateien eines Konfigurationsverzeichnisses. Er lässt sich aber auch separat von diesen für die Erzeugung einer statischen Konfiguration mit nachgelagerter Aktivierung einsetzen.

Abbildung B.2 zeigt den grafischen Editor zur interaktiven Zusammensetzung, Konfiguration und Aktivierung von Speicherflüssen. Jedes Modul wird durch ein Dateisystem repräsentiert. Von besonderem Interesse ist hierbei das NubiSave-Splitter-Modul, welches eine $1 : n$-Aufteilung des Speicherflusses erreicht. Die Konfiguration eines jeden Moduls ruft per Reflektionsmechanismus einen spezifischen oder, falls nicht vorhanden, einen generischen Moduldialog mit Editierfeldern für alle benötigten Parameter auf. Dies ermöglicht die Einbringung neuer Module zum Installationszeitpunkt. Übliche Parameter sind beispielsweise eine Endpunktangabe in Kombination mit einer Benutzerauthentifizierung in Speicherdienstmodulen oder die Schlüssellänge in Verschlüsselungsmodulen.

Als zusätzliche Funktionalität erlaubt der Editor die Festlegung von Richtlinien zur Datenspeicherung im NubiSave-Splitter-Modul. Diesbezüglich wird per Markierung signalisiert, dass Speicherdienste voneinander abhängig sind, obwohl sie es in einer gegebenen Konfiguration nicht sein dürfen, wenn die Sicherheitseigenschaften der Dispersion zugesichert werden sollen. Temporäre Einschränkungen der Verfügbarkeit werden, sofern die zugrundeliegenden Dateisysteme diese Statusmeldungen unterstützen, ebenfalls visuell signalisiert.

Weiterhin ermöglicht der Editor die Einbindung neuer Speicherdienste per Katalogsuche. Dabei wird ein per Replikation eines Git-Repositoriums erstellter lokaler Katalog mit einfachen Dienstbeschreibungen im INI-Dateiformat geladen und seine Inhalte auf Anforderungen des Benutzers geprüft. Gefundene Dienste können anschließend als Blattknoten in den Speicherflussbaum eingefügt und aktiviert werden. Benutzer können darüber hinaus die Migration von Daten aus einem Speicherziel in ein anderes anweisen.

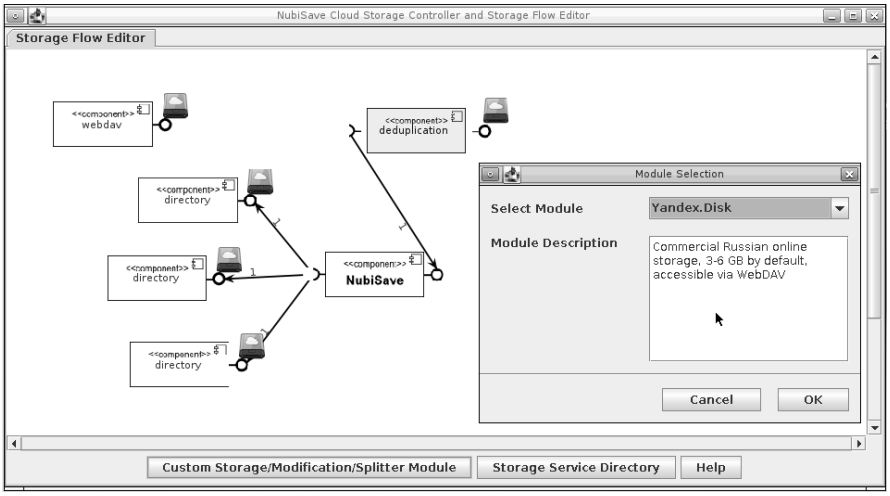

Abb. B.2 NubiSave Storage Flow Editor

Speicherdienstintegration in Gateway-, Desktop- und Mobilsysteme

Repositorium Desktop und Gateway: `git://nubisave.org/git/nubisave`
Repositorium Mobil: `git://nubisave.org/git/nubidroid`
Vorführungen: keine
Die vorgestellten Werkzeuge zur Simulation der Datenspeicherung über unabhängige Speicherziele und zur Definition von Speicherflüssen lassen sich durch Administratoren oder technisch versierte Benutzer unabhängig von der für die eigentliche Datenspeicherung vorgesehenen Laufzeitumgebung bedienen. Demgegenüber steht die laufzeitbezogene Speicherdienstintegration in die Umgebung des Benutzers, welche zumeist als Desktop- oder Mobilumgebung für einen einzelnen Benutzer oder als Gateway bzw. Proxy für mehrere Benutzer auftritt. Hierfür sind zwei dedizierte Softwarelösungen entstanden.

Der NubiSave-Splitter ist ein virtuelles Dateisystem auf Basis der Dateisystemschnittstelle FUSE, welche zumeist unter Linux und ähnlichen Betriebssystemen bereitsteht. Das Dateisystem stellt zum einen ein Datenverzeichnis zur Verfügung, welches dorthin geschriebene Dateien und Verzeichnisse anhand der Konfiguration kodiert und verarbeitet, und zum anderen ein Konfigurationsverzeichnis, welches zur dynamischen Anpassung der Konfiguration genutzt wird. Hierfür ist primär der im vorherigen Abschnitt vorgestellte Speicherflusseditor vorgesehen, wobei zu Automatisierungszwecken auch andere Konfigurationsgeneratoren genutzt werden können.

NubiSave ist in Java implementiert und benötigt das Splitter-NG-Framework für die dynamische Auswahl eines Dispersionsalgorithmus.

Die vorrangig aus Sicherheitsgründen deutliche eingeschränkte Betriebssystemvariante Android stellt die FUSE-Schnittstelle nur in bestimmten Gerätekonfigurationen zur Verfügung, welche den Benutzern meist nicht zugänglich sind. Aus diesem Grund existiert mit NubiDroid eine an das System und seine spezifischen Abläufe bei der Verarbeitung und Speicherung von Daten eine alternative Speicherdienstintegration vor allem für mobile Geräte. Neben der Verarbeitung von Dateien nimmt NubiDroid auch Text- und Multimediainhalte aus anderen Anwendungen entgegen.

NubiDroid ist in Java auf Basis des Systems Android in Version 4.4+ implementiert und nutzt intensiv das mit der Version eingeführte *Storage Access Framework* zur kontrollierten Übergabe von Dateien zwischen Anwendungen und Speicherzielen. Es setzt ebenfalls auf dem Splitter-NG-Framework auf.

Abbildung B.3 zeigt den Startbildschirm der InfinityDrive-Anwendung auf einem Mobiltelefon. Diese Anwendung ist direkt von NubiDroid abgeleitet und dient der weiteren Verbreitung der Software zur Erhöhung der Sicherheit der Benutzer durch höhere Zusicherbarkeit relevanter Dateneigenschaften bei der Einbindung von Online-Speicherdiensten.

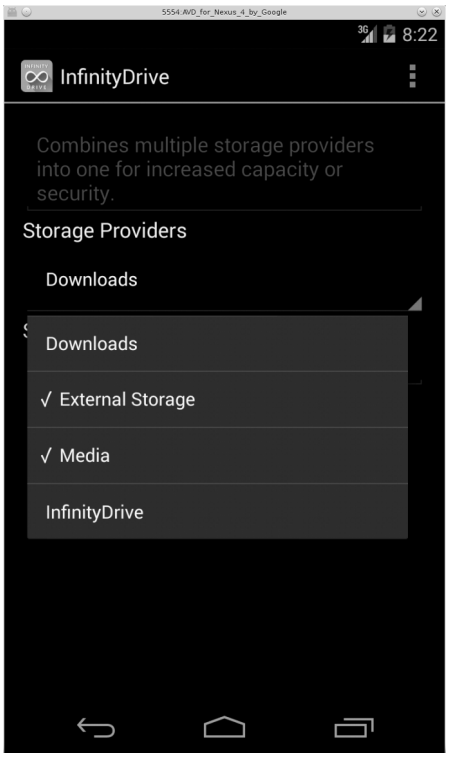

Abb. B.3 NubiDroid als InfinityDrive-Anwendung

Eine vorkonfigurierte Installation von NubiSave steht als Abbild einer virtuellen Maschine unter der Bezeichnung NubiGate zur Verfügung. Diese kombiniert einen WebDAV-Server, einen darunterliegenden NubiSave-Speicherdienst-Controller sowie eine um verteilte Datenspeicherung erweiterte Instanz der webbasierten Dateiverwaltung OwnCloud zur Netzlaufwerksanbindung mehrerer Benutzer an eine vom Administrator der virtuellen Maschine vorgegebene Speicherdienstkonfiguration.

Visualisierung verteilter Datenbestände

Repositorium: `git://nubisave.org/git/nubivis`
Vorführungen: [STS14]

Werden umfangreichere Mengen an Dateien, Archiven und Verzeichnissen über mehrere Speicherziele hinweg verteilt abgelegt, und ändert sich insbesondere diese Konfiguration über die Zeit, so wird es einem Benutzer schwerfallen, bestimmte Dateien wiederzufinden oder den Speicherzustand einer Datei zu beurteilen. Zudem ist es mit existierenden Werkzeugen nicht möglich, die Abhängigkeit von Dienstanbietern zu beurteilen und bei Bedarf eine Datenmigration zu alternativen Speicherzielen anzufordern.

Die Webanwendung NubiVis ist in Java und JavaScript mit diversen Frameworks implementiert. Sie bietet mehrere voneinander unabhängige Visualisierungsmodule zur Darstellung der verteilten Dateien und Fragmente. Filter können jedoch zwischen den Modulen übernommen werden, so dass beispielsweise in einer Zeitleiste zuerst ein Jahr eingegrenzt und anschließend in der Dateigrößendarstellung ein bestimmter Ordner ausgewählt werden kann.

Abbildung B.4 zeigt als Screenshot eine Visualisierungsvariante von NubiVis. Diese entspricht einem klassischen Dateimanager mit Baumansicht, vermittelt aber zusätzlich Informationen über die Aufteilung und Verteilung der Fragmente auf unterschiedliche Speicherorte.

Dateiübermittlung an Speicherdienste

Repositorium: `https://github.com/joe42/CloudFusion`

Zur vereinheitlichten Übertragung von Dateien an Speicherdienste mit unterschiedlichen Aufrufschnittstellen und Übertragungseigenschaften empfiehlt sich der Einsatz von CloudFusion. Das System bietet eine ähnliche Funktion wie existierende FUSE-Dateisysteme für den Zugriff auf Netzwerkdienste an, erreicht jedoch gleichzeitig wesentlich vorteilhaftere nichtfunktionale Eigenschaften.

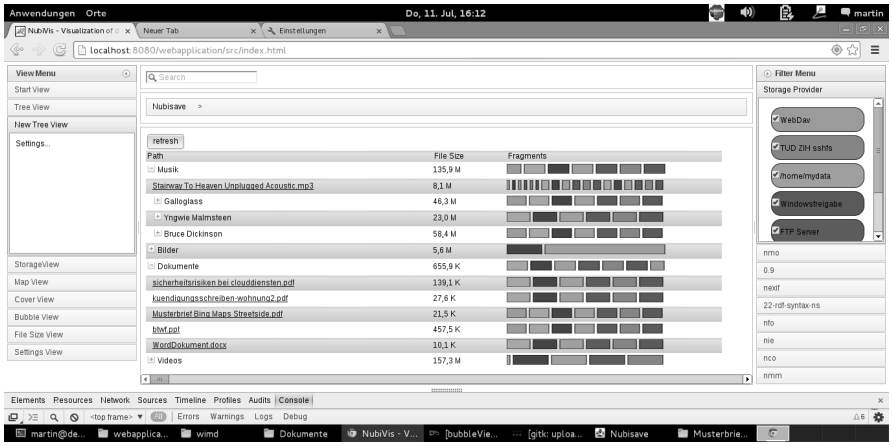

Abb. B.4 NubiVis-Webanwendung

CloudFusion läuft entweder als Dienst im Hintergrund oder als Terminalanwendung im Vordergrund. In beiden Fällen wird ein virtuelles Dateisystem bereitgestellt, welches ein Datenverzeichnis, ein Konfigurationsverzeichnis und ein Statistikverzeichnis umfasst. Das Datenverzeichnis gibt den Inhalt des Speicherdienstes als Struktur mit Dateien und Unterordnern wieder. Die Konfiguration erfolgt über eine INI-Datei. Die Statistikdateien erlauben einen dateibasierten Echtzeitzugriff auf interne Zustände von CloudFusion, insbesondere den Inhalt der Upload-Queue, die Fehlerrate sowie Übertragungsstatistiken. Die Implementierung nutzt Python mit diversen protokoll- und anbieterspezifischen Modulen. Über eine Multithreading-Architektur mit Upload-Queue, Cache und Archiven zur Zusammenfassung kleiner Dateien wird ein hoher Durchsatz und eine niedrige Latenz weitgehend unabhängig von der Größe und Muster der Schreib- und Leseoperationen erreicht.

Stealth-Datenbanksystem

Repositorium: `git://nubisave.org/git/dispersedalgorithms`
Vorführungen: [SMS15, Spi15]

StealthDB ist ein für den Einsatz in Cloud-Umgebungen mit nicht zusicherbaren Diensteigenschaften optimiertes spaltenorientiertes Datenbankverwaltungssystem. Die Daten können über mehrere Speicherorte wie dem lokalen Speicher, Dateien und Dienstanbieter per Dispersion und anderen Verfahren verteilt werden. Über SQL-ähnliche Anfragen legt der Benutzer a priori oder per Datenmigration a posteriori auf Tabellen- oder Spaltenebene fest, welche Speicherziele angesprochen und welche Kodierungsoptionen in der Speicherkette dabei genutzt werden sollen. In Anfragen werden Qualitätsanforderungen des Benutzers unter Berücksichtigung

des Speicherflusses zur Durchführung als Optimierungsziele und Nebenbedingungen herangezogen.

StealthDB ist in Python 3.x implementiert. Für die Kommunikation, also die Übertragung von Daten und die Übermittlung von Anfragen, wird Pyro als RPC-Bibliothek genutzt. Die Datenübertragung in Einfügeoperationen kann jedoch auch optimiert über CloudFusion erfolgen, um bei vielen Einzelwerten den Kommunikationsbedarf zu reduzieren.

Abbildung B.5 zeigt einen Screenshot der SQL-Kommandozeilen-Oberfläche von StealthDB beim Aufruf der Hilfefunktionen hinsichtlich implementierter Speicherziele, Verteilungsverfahren und Aggregatsfunktionen.

```
josef@rumba:/repos/space-universe/dispersedalgorithms/db$ ./stealthdb
~~ StealthDB ›finegrained ›Wed Oct 15 17:07:07 2014 +0200 ~~
Type HELP; to get started.
Using database 'stealthdb'.
Storing all data and performing all procedures on ['mem://localhost'] with ['replication'].
>>> HELP;
StealthDB Quickhelp
 HELP [<topic>]
 SHOW DATABASES|TABLES
 CREATE TABLE <table> [(<column> <column-type>, ...)]
 DESCRIBE <table>
 DROP TABLE [IF EXISTS] <table>
 CREATE DATABASE <database>
 USE DATABASE <database>
 DROP DATABASE <database>
 [EXPLAIN ANALYZE] SELECT [DISTINCT] */<column>/<aggregate>(*/<column>)/<predicate>, ... [FROM <table>]
  [WHERE <column> LIKE/=/... <value>] [ORDER BY <column> [ASC|DESC]] [OPTIMIZE FOR <goal>]
 INSERT INTO <table> (<column>, ...) VALUES (<value>, ...)
 DELETE FROM <table>
 USE CLOUDS <cloud> [AND <cloud>...][WITH <distribution>]
 ALTER TABLE <table> ALTER COLUMN <column> USE CLOUDS ...
 MODE <mode>
>>> HELP aggregate;
Aggregates: ('COUNT', 'SUM', 'AVG', 'MIN', 'MAX')
>>> HELP cloud;
Clouds: <protocol>://urlremainder, with protocols: ('mem', 'file', 'cloud'); or 'auto'
>>> HELP distribution;
Distributions: ('roundrobin', 'replication', 'dispersion', 'hashring') x ('encryption', 'compression')
>>>
```

Abb. B.5 Sicheres Cloud-Datenbankverwaltungssystem StealthDB

Integrationsperspektive

Die vorgestellten Softwareanwendungen sind weitgehend unabhängig voneinander entstanden, lassen sich jedoch kombinieren, um komplexere Datenverwaltungsvorgänge mit einzelnen Diensten oder Dienstkombinationen zu realisieren.

Eine solche Integration ist in Abbildung B.6 zu sehen. Hierbei werden Daten entweder in Form einer Datei mit NubiSave kodiert und mit CloudFusion an einen Speicherdienst übertragen, oder in Form eines Datenbankeintrags mit StealthDB erzeugt und anschließend an einen Speicher- oder Rechendienst übermittelt. Parallel dazu erfolgt die Datenübertragung mit NubiDroid. Unabhängig von der Übertragung zur Laufzeit stellt NubiVis dem Benutzer einen Überblick über die Verteilung

der Dateien dar, während der Speicherflusseditor es dem Benutzer erlaubt, ungünstige Konfigurationen abzuändern.

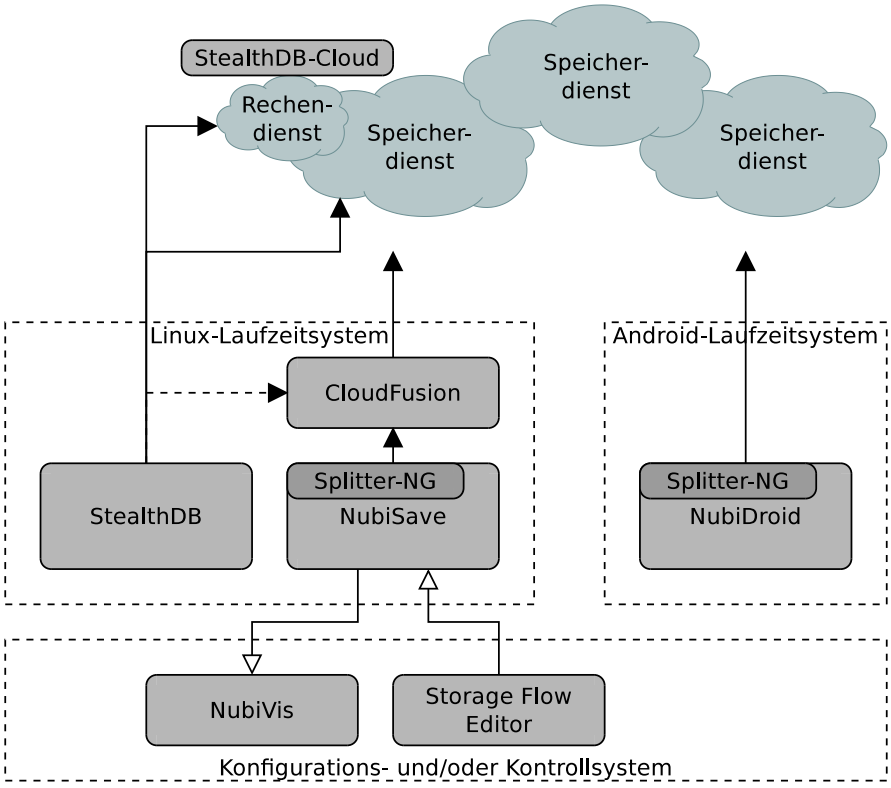

Abb. B.6 Kombinierbarkeit der Softwareartefakte als integrierte Softwarelösung für die sichere verteilte Datenverwaltung

Anhang C
Repositorien mit Experimentdaten

Alle eingesetzten Werkzeuge, Experimentier- und Simulationsskripte, erzielte Rohdaten sowie auch Diagrammdefinitionen stehen in öffentlich versionierten Git-Repositorien zum Zweck der Reproduzierbarkeit aller Ergebnisse zur Verfügung. Die Liste korrespondiert mit den Abschnitten in Kapitel 6.

- Datenkodierung: Splitter-NG / Abbildungen: 6.2
  - Repositorium: `git://nubisave.org/git/miscexperiments`
  - Ordner mit Rohdaten: `splitterng-benchmarks/plotting/`
  - Reproduzierbarkeit: `splitterng-benchmarks/splitterbench.py` mit Parametrisierungen im Skript
- Datenkodierung: Bitsplitting / Abbildungen: 6.3
  - Repositorium: `git://nubisave.org/git/dispersedalgorithms`
  - Ordner mit Rohdaten: `bitsplitter/evaluation/plotting/`
  - Reproduzierbarkeit: `bitsplitter/performancetest`
- Datenkodierung: Komprimierung / Abbildungen: 6.4 … 6.6
  - Repositorium: `git://nubisave.org/git/dispersedalgorithms`
  - Ordner mit Rohdaten: `diverse-algorithms/compression/plotting/`
  - Reproduzierbarkeit: `diverse-algorithms/compression/experiment.py`
- Datenkodierung: Fragmentexpansion / Abbildungen: 6.7
  - Repositorium: `git://nubisave.org/git/dispersedalgorithms`
  - Ordner mit Rohdaten: `diverse-algorithms/fragment-expansion/plotting/`
  - Reproduzierbarkeit: `diverse-algorithms/fragment-expansion/expander-exp.py`
- Datenkodierung: Verschlüsselung / Abbildungen: 6.8 … 6.11
  - Repositorium: `git://nubisave.org/git/dispersedalgorithms`

  - Ordner mit Rohdaten: `db/encryption/benchmark/`
  - Reproduzierbarkeit: `db/encryption/benchmark/bench.py` mit Parametrisierungen im Skript

- Datenverteilung: Gesamtverfügbarkeit / Abbildungen: 6.13 … 6.17

  - Repositorium: `git://nubisave.org/git/mcssimulation`
  - Ordner mit Rohdaten: `plotting/`, `plotting/mc/n4/`
  - Reproduzierbarkeit: `experiments/calc-availability.sh`, `experiments/montecarlocombinations.py`

- Datenverteilung: Verteilungen ohne Zusicherung / Abbildungen: 6.18 … 6.24

  - Repositorium: `git://nubisave.org/git/mcssimulation`
  - Ordner mit Rohdaten: `experiments/`, `revplotting/`
  - Reproduzierbarkeit: `experiments/*.py`

- Datenverteilung: Verteilungen mit Zusicherung / Abbildungen: 6.25, 6.26

  - Repositorium: `git://nubisave.org/git/mcssimulation`
  - Ordner mit Rohdaten: `experiments/`, `plotting/`, `plotting/picav+/`
  - Reproduzierbarkeit: `experiments/staggeredcombinations.py`

- Datenverarbeitung: Zeichenkettensuche / Abbildungen: 6.27

  - Repositorium: `git://nubisave.org/git/dispersedalgorithms`
  - Ordner mit Rohdaten: `typespecific/unicode/`
  - Reproduzierbarkeit: `typespecific/unicode/testlength`

- Datenverarbeitung: Redundanzsuche / Abbildungen: 6.28 … 6.32

  - Repositorium: `git://nubisave.org/git/dispersedalgorithms`
  - Ordner mit Rohdaten: `diverse-algorithms/redundancy/`
  - Reproduzierbarkeit: `diverse-algorithms/redundancy/*.py`

- Datenübertragung / Abbildungen: 6.33 … 6.43

  - Repositorium: `git://nubisave.org/git/miscexperiments`
  - Ordner mit Rohdaten: `storage-benchmarks/fuse-benchmarks/`, `storage-benchmarks/gsutil-benchmark/`, `storage-benchmarks/chunking/`
  - Reproduzierbarkeit: `storage-benchmarks/chunking/fsperformance.py`

- Datenspeicherung / Abbildungen: 6.44, 6.45

  - Repositorium: `git://nubisave.org/git/miscexperiments`
  - Ordner mit Rohdaten: `nubisave-benchmarks/`
  - Reproduzierbarkeit: `nubisave-benchmarks/run-nubisave-exp.sh` mit Parametrisierungen im Skript

- Datenverwaltung / Abbildungen: 6.48, 6.49
  - Repositorium: `git://nubisave.org/git/dispersedalgorithms`
  - Ordner mit Rohdaten: `db/benchmark/`, `db/benchmark/aggcsv/`
  - Reproduzierbarkeit: `db/benchmark/bench.sh`
- Datenstromspeicherung / Abbildungen: 6.50 … 6.52
  - Repositorium: `git://nubisave.org/git/miscexperiments`
  - Ordner mit Rohdaten: `stream-benchmarks/`
  - Reproduzierbarkeit: `stream-benchmarks/webdav-pubsub/measure\-dispersed-publish.py`, `stream-benchmarks/gsutil-delta/runscript.sh`