Analyse und Vorhersage der Aktualisierungen von Web-Feeds

Dissertation

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Inf. Sandro Reichert
geboren am 02.06.1980 in Dresden

Gutachter:

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill	TU Dresden
Prof. Dr. Peter Mandl	Hochschule München
Jun.-Prof. Dr.-Ing. Thomas Schlegel	TU Dresden

Tag der Verteidigung: 02.02.2012

Dresden, im März 2012

Danksagung

Die vorliegende Dissertation entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl Rechnernetze der Fakultät Informatik an der Technischen Universität Dresden. Wenngleich die Arbeit einzig meinen Namen trägt, so haben zahlreiche Personen auf unterschiedliche Weise zum Gelingen dieser Dissertation beigetragen.

Ich bedanke mich bei meinem Doktorvater Prof. Alexander Schill für die Unterstützung in den letzten Jahren. Die regelmäßigen Vorträge im Rahmen der Klausurtagungen sowie das stets sehr zügige Feedback zu Manuskriptentwürfen halfen, zielführend zu arbeiten. Darüber hinaus trugen die flexiblen Arbeitszeiten und die unkomplizierte Finanzierung meiner Arbeit in den Projekten Theseus/Texo (BMWI), Aletheia (BMBF) und Intellix (BMBF) sehr zu einem angenehmen Arbeitsklima bei. Des Weiteren danke ich meinem externen Gutachter Prof. Peter Mandl und meinem Fachreferenten Jun.-Prof. Thomas Schlegel für ihre Hinweise und hilfreichen Kommentare.

Einen weiteren wichtigen Baustein bildete das kollegiale Umfeld. An erster Stelle danke ich Daniel Schuster, der als Coach und langjähriger Zimmergenosse jederzeit bereit war, meine Arbeit zu hinterfragen und Entwürfe des Manuskripts zu diskutieren. Mein Dank gilt David Urbansky, Klemens Muthmann und Philipp Katz für ihre Bemühungen rund um das Toolkit Palladian, das die Basis der Implementierung der vorliegenden Arbeit bildet. Die Synergieeffekte des gemeinsamen Arbeitens und fachlicher Diskussionen erwiesen sich als sehr ertragreich. Klemens Muthmann gilt – stellvertretend für alle anderen Kollegen – mein Dank für sein Engagement im Projekt Intellix während der Endphase meiner Arbeit.

Das dritte Standbein der Arbeit bildete mein privates Umfeld aus Familie und Freunden. Jakob Schumann verdanke ich eine Vielzahl fruchtbarer, fachlicher Diskussionen. Des Weiteren haben er sowie Daniel Schott das vorliegende Manuskript einem sorgfältigen Lektorat unterzogen und somit zu einer besseren Allgemeinverständlichkeit beigetragen. Nicht zuletzt möchte ich mich bei meiner Freundin Sylvie Kunz für ihre verständnisvolle Art bedanken, mit der sie mich stets aufmunterte.

Abstract

Feeds werden u. a. eingesetzt, um Nutzer in einem einheitlichen Format und in aggregierter Form über Aktualisierungen oder neue Beiträge auf Webseiten zu informieren. Da bei Feeds in der Regel keine Benachrichtigungsfunktionalitäten angeboten werden, müssen Interessenten Feeds regelmäßig auf Aktualisierungen überprüfen. Die Betrachtung entsprechender Techniken bildet den Kern der Arbeit.

Die in den verwandten Domänen Web Crawling und Web Caching eingesetzten Algorithmen zur Vorhersage der Zeitpunkte von Aktualisierungen werden aufgearbeitet und an die spezifischen Anforderungen der Domäne Feeds angepasst. Anschließend wird ein selbst entwickelter Algorithmus vorgestellt, der bereits ohne den Einsatz spezieller Konfigurationsparameter und ohne Trainingsphase im Durchschnitt bessere Vorhersagen trifft, als die übrigen betrachteten Algorithmen.

Auf Basis der Analyse verschiedener Metriken zur Beurteilung der Qualität von Vorhersagen erfolgt die Definition eines zusammenfassenden Gütemaßes, welches den Vergleich von Algorithmen anhand eines einzigen Wertes ermöglicht.

Darüber hinaus werden abfragespezifische Attribute der Feed-Formate untersucht und es wird empirisch gezeigt, dass die auf der partiellen Historie der Feeds basierende Vorhersage von Änderungen bereits bessere Ergebnisse erzielt, als die Einbeziehung der von den Diensteanbietern bereitgestellten Werte in die Berechnung ermöglicht.

Die empirischen Evaluationen erfolgen anhand eines breitgefächerten, realen Feed-Datensatzes, welcher der wissenschaftlichen Gemeinschaft frei zur Verfügung gestellt wird, um den Vergleich mit neuen Algorithmen zu erleichtern.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Wissen ist Macht. Diese bekannte Redewendung trifft insbesondere auch auf Unternehmen zu, liegt doch ein guter Teil des unternehmerischen Potentials im Wissen einer Firma. Je mehr nützliche Informationen einem Entscheidungsträger zur Verfügung stehen, desto besser kann er in der Regel fundierte Entscheidungen treffen. Unternehmen sind an allen relevanten Informationen interessiert, welche ihre Produkte, Dienstleistungen, sie selbst oder Mitbewerber betreffen. Relevante Informationen sind in diversen Quellen innerhalb und außerhalb der administrativen Grenzen eines Unternehmens enthalten, sowie häufig mit irrelevanten Informationen vermischt.

Eine interessante Informationsquelle stellt das World Wide Web (WWW) dar. Kunden nutzen das WWW als Kommunikationsplattform, um untereinander Erfahrungen und Meinungen über Produkte und Unternehmen auszutauschen. Für Unternehmen ist das WWW eine wichtige Basis ihrer Öffentlichkeitsarbeit, um Interessenten über aktuelle oder geplante Produkte zu informieren. Potentiell relevante Informationen sind auf eine Vielzahl von Quellen im Web verteilt, auf denen neben klassischen Webseiten verschiedene Techniken wie Blogs, Foren oder ein Content-Management-System (CMS) zum Einsatz kommen. Die Integration von Informationen aus diesen Quellen in unternehmensinterne Systeme ist Gegenstand aktueller Forschungen im Bereich der Informationsintegration [AAB+09]. Einige der Quellen haben die Gemeinsamkeit, dass sich interessierte Leser über Aktualisierungen oder neue Beiträge auf den Seiten informieren können, indem sie zur Verfügung gestellte *Web Feeds* abonnieren. Web Feeds, im Folgenden als Feeds bezeichnet, enthalten eine begrenzte Anzahl von Einträgen bestehend aus einem Titel, einer Beschreibung und einem Link zum vollständigen Beitrag. Feeds fanden im Verlauf des letzten Jahrzehnts eine zunehmende Verbreitung. Sie bieten den Vorteil, dass Neuigkeiten in aggregierter Form und einem einheitlichen Format verteilt werden. Nutzer verwenden üblicherweise einen Feed Reader zum Lesen und zur Verwaltung der von ihnen abonnierten Feeds. Unglücklicherweise werden bei Feeds in der Regel keine Benachrichtigungsfunktionalitäten (push) angeboten, sodass Abonnenten einen Feed regelmäßig auf Aktualisierungen überprüfen müssen (pull). Da verschiedene Feeds in der Regel unterschiedliche Aktualisierungsmuster aufweisen, birgt der Zugriff per regelmäßigem pull zwei Nachteile. Ist die Frequenz der Abfragen zu niedrig, werden Aktualisierungen der Quelle nur verspätet festgestellt oder gar verpasst. Ist die Frequenz hingegen zu hoch, wird eine Quelle unnötig oft abgefragt und somit wertvolle Systemressourcen vergeudet. Neben klassischen Feed Readern wächst die Anzahl von Systemen wie [HC09, LB08], die Informationen aus Feeds automatisiert extrahieren und somit zusätzliche Last auf den Quellen erzeugen.

Diese Gegebenheiten lassen Feeds in den Fokus der Arbeit rücken. Es werden verschiedene adaptive Techniken auf ihre Eignung hin untersucht, die bisher verwendeten statischen Abfrageintervalle zu ersetzen. Aus Sicht eines Nutzers ist es erstrebenswert, wenn er keinen Feed-Eintrag verpasst und jeden neuen Eintrag ohne nennenswerte Zeitverzögerung erhält. Aus technischer Sicht wird die Skalierbarkeit der adaptiven Techniken betrachtet, sodass die Algorithmen zur Überwachung einer großen Zahl von Feeds effizient eingesetzt werden können.

1.2. Szenario

Ein Geschäftsreisender checkt spät abends im Hotel ein. Der Portier hatte einen schlechten Tag und verhält sich dem Gast gegenüber äußerst unfreundlich. Der Reisende verkündet darauf hin seinen Unmut in Form einer Kritik auf einer Hotelbewertungsseite, wodurch die Reputation und tendenziell auch der künftige Umsatz des Hotels sinken. Nimmt der Hotelmanager rechtzeitig Kenntnis von dem Vorfall, hat er die Möglichkeit, sich bei dem Gast zu entschuldigen, ihm einen Preisnachlass zu gewähren oder in sonstiger Weise entgegen zu kommen. Der Gast kann das Hotel zufrieden verlassen, kommt vielleicht wieder oder überarbeitet die negative Bewertung.

Eine naheliegende Frage ist, wie der Manager die relevanten Informationen erlangt. Einerseits könnte er die entsprechenden Bewertungsseiten kennen und regelmäßig auf neue Einträge hin untersuchen. Alternativ kann er sich an einen Dienstleister wenden, der diese Aufgabe für ihn übernimmt. Es gibt Unternehmen bei denen Kunden Anfragemuster (engl. query pattern) hinterlegen können. Diese Dienstleister durchsuchen in regelmäßigen Zyklen das Web bzw. Feeds auf diese Muster und übermitteln neue Informationen z. B. per E-Mail an ihre Kunden. Um diese Aufgabe zu erfüllen, müssen sie u. a. neue Quellen ausfindig machen, Änderungen an bereits heruntergeladenen Dokumenten erkennen, sowie die erhaltenen Dokumente einer Informationsextraktion unterziehen. Es ist davon auszugehen, dass das Verhältnis relevanter Informationen zu Feeds einer zipf-Verteilung unterliegt: nur wenige Feeds enthalten ausschließlich Bewertungen vom Hotels, hingegen enthält ein Großteil der Feeds nur selten oder sporadisch Einträge mit relevanten Informationen. Um einen möglichst großen Anteil dieser Einträge zu erfassen, d. h. einen hohen *Recall* des Systems zu erlangen, muss eine große Anzahl Feeds regelmäßig verarbeitet werden. Mit steigender Anzahl zu überwachender Quellen steigt der Bedarf an Systemressourcen für deren Verarbeitung. Eine Möglichkeit der Optimierung ist, die Abfragen der Quellen effizient zu gestalten und möglichst nur genau dann abzufragen, wenn sich die Quelle geändert hat. Dieser Aspekt ist Gegenstand der Arbeit.

Verallgemeinert man das Hotelszenario zu generischen Produktinformationen, so gibt es u. a. im Bereich E-Commerce oder Versandhandel eine Vielzahl von Unternehmen, die Produktbewertungen in verschiedenen Phasen des Produktlebenszyklus nutzen, etwa um Marketing und Vertrieb zu optimieren. Größere Unternehmen verwalten ihre Daten üblicherweise in Produktinformationssystemen wie Data Warehouses (DWHs), deren Aufbau in Abschnitt 1.3 erläutert wird. Der eben eingeführte Dienstleister kann im Umfeld des Produktinformationsmanagements Daten aus öffentlichen Quellen vorverarbeiten

und dem Produktinformationssystem (PIS) des Versandhandels zur Verfügung stellen. Überlappen sich die Geschäftsfelder der Kunden des Dienstleisters, so ergeben sich für ihn Synergieeffekte, wenn Quellen jeweils zur Beantwortung mehrerer Anfragemuster genutzt werden können und hierfür nur ein mal pro Zyklus abgefragt werden müssen.

1.3. Verwaltung von Produktinformationen

Das im Szenario betrachtete Beispiel wird nun in die Verwaltung von Produktinformationen eingegliedert. Abbildung 1.1 zeigt ein PIS mit den drei Ebenen Datenzugriff und -integration; Extraktion, Verwaltung und Föderation von Informationen sowie der Präsentation von Informationen. In einigen Anwendungsbereichen können einzelne Schichten verschmelzen, da Integrationskomponenten wie der im Szenario beschriebene Dienstleister bereits eine Extraktion vornehmen. Für ein grundlegendes Verständnis der Architektur sind diese Details jedoch nicht relevant.

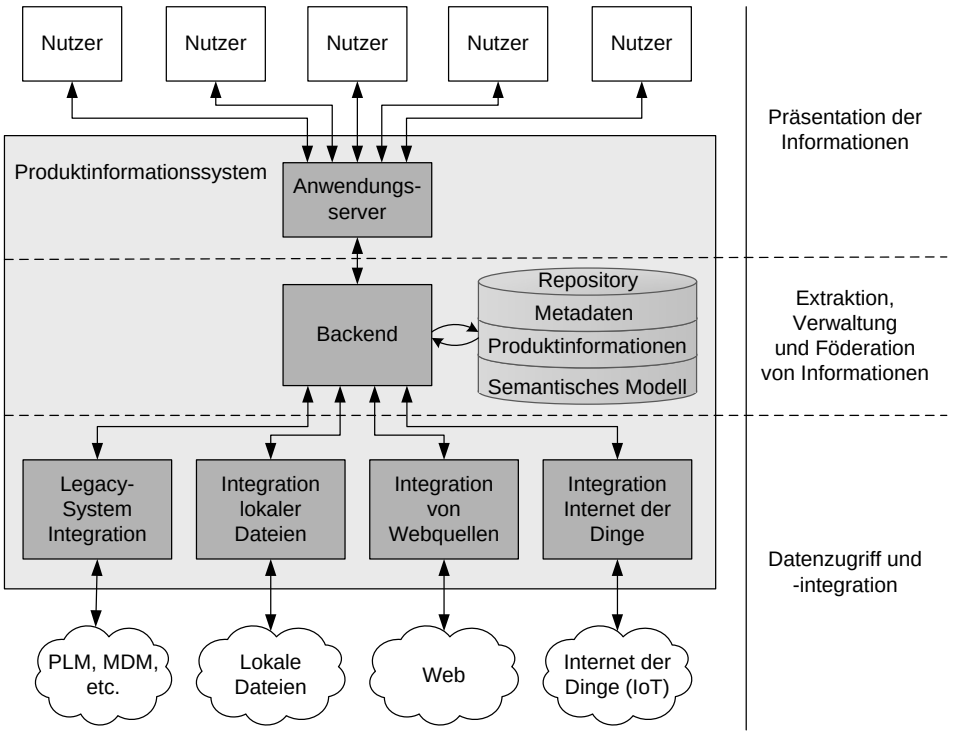

Abbildung 1.1.: Architektur eines Produktinformationssystems.

Zur Integration von Daten sind mehrere Schritte notwendig, bei denen die drei Aspekte Autonomie, Heterogenität und Verteilung der Datenquellen zu berücksichtigen sind. Der erste Schritt, in dem die Quellen der gesuchten Daten ausfindig gemacht werden, wird als Source Localization bezeichnet und ist ein Teilgebiet des Information Retrieval. Zu den meist historisch gewachsenen Datenquellen zählen u. a. bestehende Systeme wie Master Data Management (MDM) oder Product-Lifecycle-Management-(PLM)-Systeme,

Office-Dokumente, Web-Quellen wie Webseiten, Feeds, Foren und Wikis, sowie das Internet der Dinge (engl. Internet of Things (IoT)). Da gleichermaßen firmeninterne und öffentliche Quellen betrachtet werden, können sich die Datenquellen und das PIS sowohl in der gleichen als auch in verschiedenen administrativen Domänen befinden. Für die meisten Quelltypen wird der Schritt Source Localization initial zur Inbetriebnahme des PIS durchgeführt, im Bereich des Web ist er ein kontinuierlicher Prozess. Das Auffinden neuer Quellen ist ein Aspekt der untersten Ebene in Abbildung 1.1 und wird u. a. in [WHSS10] speziell im Kontext von Produktinformationen betrachtet.

Wurden die Quellen identifiziert, müssen die enthaltenen Daten in geeigneter Weise dem integrierenden System zur Verfügung gestellt werden. Die Daten liegen strukturiert, semi-strukturiert oder unstrukturiert in den Quellen vor. Sie werden von spezialisierten Komponenten zur Weiterverarbeitung in ein einheitliches, internes Format überführt und dem Backend des PIS übergeben. Ein wichtiger Bestandteil dieser Aufgabe ist, in den Quellen neu hinzukommende Daten zeitnah in das PIS zu integrieren. Dies ist ein kontinuierlicher Prozess. Ein bedeutendes Unterscheidungsmerkmal der Quellen ist, ob diese über geeignete Mechanismen verfügen, um dem PIS das Vorhandensein neuer Daten anzuzeigen. Stellt eine firmeninterne Quelle diese Funktionalität nicht bereit, so kann sie gegebenenfalls modifiziert werden. Bei externen Quellen ist dies meist nicht möglich, sodass diese in geeigneten Abständen auf das Vorhandensein neuer Daten zu untersuchen sind (pull).

Im Anschluss folgen die Informationsextraktion und Föderation. Bei den zur Informationsextraktion eingesetzten Techniken wie Natural Language Processing (NLP) und Named Entity Recognition (NER) erfolgen eine Interpretation und Bewertung der Daten, wobei ein Informationsverlust oder Fehlinterpretationen auftreten können. Zur persistenten Speicherung werden die Informationen in ein zentrales Repository integriert. Hierfür werden sie zunächst anhand eines semantischen Modells harmonisiert, um z. B. verschiedene Synonyme auf das Vokabular einer domänenspezifischen Ontologie abzubilden. Anschließend werden die Informationen föderiert, in Folge dessen z. B. Duplikate einer Information aus verschiedenen Quellen erkannt und entsprechend abgelegt werden.

Einer der Vorteile eines PIS besteht in der einheitlichen Schnittstelle, über welche auf die umfassenden Produktinformationen zugegriffen wird, um sie ganzheitlich zu präsentieren. Abbildung 1.2 zeigt die grafische Nutzungsschnittstelle eines Prototyps des Forschungsprojekts *Aletheia – semantische Föderation umfassender Produktinformationen* [Ale11], in dessen Rahmen die Dissertation entstand. Das in Aletheia entwickelte PIS wurde als Grundlage für die in der Dissertation entwickelten Konzepte verwendet, die in der eigenen Arbeit gewonnenen Erkenntnisse beschränken sich jedoch nicht auf das im Projekt entwickelte System.

1.4. Ziele der Arbeit

Ein wichtiger Aspekt eines PIS ist die Aktualität der bereitgestellten Informationen. Das Ziel der Aktualisierung ist, den gegenwärtigen Stand der Daten externer Systeme, die durch Data Provider angebunden sind, im zentralen Repository widerzuspiegeln. Hierzu

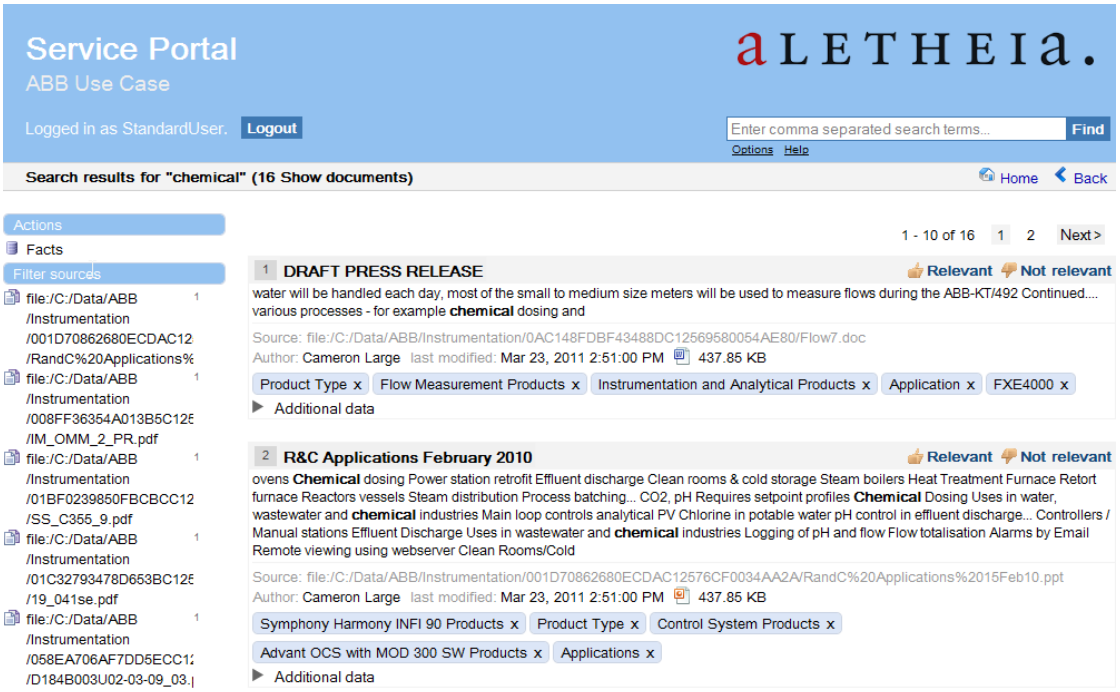

Abbildung 1.2.: Screenshot des Produktinformationssystems Aletheia.

muss erkannt werden, wenn in den Quellen neue Daten hinzukommen und bestehende aktualisiert oder gelöscht werden. Die in Abbildung 1.1 exemplarisch aufgeführten Datenquellen können als Produzenten und das PIS als Konsument von Produktdaten betrachtet werden. Quellen wie klassische Datenbanken stellen z. B. mit Triggern Mechanismen bereit, mit denen externe Systeme wie ein PIS über Änderungen informiert werden können. Dem gegenüber stehen Systeme wie Feeds, die in der Regel keine Mechanismen zur Notifikation externer Systeme bereitstellen. Um dennoch von neuen, geänderten oder gelöschten Daten Notiz zu nehmen, müssen diese Quellen entweder regelmäßig überprüft oder um Notifikationsmechanismen erweitert werden. Eine Modifikation hat den Nachteil, dass die Quellenautonomie eingeschränkt wird. Bei externen Quellen im öffentlichen Web ist dies unmöglich, sodass diese Quellen nur über spezielle Wrapper angebunden werden können.

Der Fokus der Arbeit liegt auf dem Zugriff auf Feeds, da Webdatenquellen im Allgemeinen als immer bedeutender angesehen werden [LN07], im Fall von Feeds jedoch überwiegend *pull*-basierte Zugriffstechniken erfordern.

Hierfür werden geeignete Strategien untersucht, die das Ziel haben, genau dann auf eine Quelle zuzugreifen, wenn sich die enthaltenen Daten geändert haben. Die Problematik ist nicht gänzlich neu und wurde bereits in Domänen wie Web Caching oder Web Crawling untersucht. Diese Techniken werden vorgestellt und ihre Eignung bezüglich der Vorhersage neuer Feed-Einträge in einer umfassenden Evaluation eruiert. Für den direkten Vergleich werden drei geeignete Metriken eingeführt: Der *Delay* ist die Zeitverzögerung, mit der neue Einträge erkannt werden; die Anzahl Abfragen, die zum Auffinden eines neuen

Eintrags benötigt werden, sowie der Recall, d. h. das Verhältnis erkannter Änderungen zu allen erfolgten Änderungen. Ein wichtiger Unterschied zwischen Feeds und den meisten anderen Datenobjekten ist, dass Feeds zumeist mehrere Einträge beinhalten und somit eine begrenzte Historie aufweisen, während z. B. bei Webseiten nur der aktuelle Stand ersichtlich ist. Die Anzahl der Einträge wird als Fenstergröße bezeichnet. Die Vorhersage neuer Einträge basiert zunächst auf der Analyse der beobachteten Änderungshistorie der Quelle und wird im weiteren Verlauf um quellenspezifische Eigenschaften verfeinert.

1.5. Thesen

Bewertung von Vorhersagen Algorithmen werden zumeist anhand mehrerer Qualitätskriterien wie dem Delay, der Anzahl benötigter Abfragen oder der Anzahl verpasster Einträge beurteilt.

Es wird die These aufgestellt, dass sich diese Qualitätskriterien zu einem Kennwert zusammenfassen lassen, der einen einfachen Vergleich der Algorithmen ermöglicht.

Änderungshistorie Die Abfragen von Feeds basieren bisher entweder auf statischen, vorab festgelegten Intervallen oder auf einem Modell der Änderungshistorie, das vorab trainiert werden muss.

Es wird die These aufgestellt, dass die in einem Fenster enthaltene partielle Historie eines Feeds im Durchschnitt genügend Informationen enthält, um eine Vorhersage zu tätigen, deren Qualität äquivalent zu aufwendigeren Verfahren ist, in denen vorab ein Modell trainiert werden muss.

Nutzung abfragespezifischer Attribute Einige Feed-Formate verfügen über optionale, zur Optimierung des Abfrageintervalls vorgesehene Attribute wie skipHours oder ttl. Diese ermöglichen es Diensteanbietern, Zeiträume ohne Aktivitäten zu spezifizieren, in denen Nutzer keine Aktualisierungen zu erwarten haben und den Feed idealerweise nicht abfragen.

Es wird die These aufgestellt, dass diese Attribute im Mittel unzuverlässige Werte enthalten und die Vorhersage auf der puren Historie durchschnittlich bessere Ergebnisse erzielt, als es mit der Auswertung abfragespezifischer Attribute möglich wäre.

1.6. Gliederung der Arbeit

Die Arbeit ist in sechs Kapitel unterteilt, von denen die Kapitel zwei bis vier den Kern bilden. Zu Beginn wird in Kapitel 2 der Stand der Technik analysiert und bestehende Verfahren hinsichtlich der Eignung für die Vorhersage neuer Feed-Einträge bewertet. Das übergeordnete Ziel der Arbeit ist die Integration von Feeds in Produktinformationssysteme, sodass diese sowie Datenbanken im Allgemeinen zuerst betrachtet werden. Anschließend werden die gängigen Feed-Formate vorgestellt und das zur Übertragung der Dokumente eingesetzte Hyper Text Transfer Protocol (HTTP) hinsichtlich der Vorhersage dienlicher Eigenschaften untersucht. Mit diesem Grundwissen erfolgt die Betrachtung der Domänen

Web Crawling und Web Caching sowie bestehender Systeme zur Verarbeitung von Feeds. Die aussichtsreichsten Techniken werden abschließend zusammengefasst und später in Kapitel 4 aufgegriffen und miteinander verglichen.

Die Basis der umfassenden Evaluation wird in Kapitel 3 geschaffen. Zunächst werden die Anforderungen an den zu verwendenden Datensatz formuliert und verfügbare Datensätze in Augenschein genommen. Die Bewertung wird ergeben, dass keiner der vorhandenen Datensätze den Anforderungen gerecht wird, sodass ein eigener umfangreicher, realer Feed-Datensatz erstellt wird. Hierbei werden einige wichtige Erkenntnisse bezüglich der Verwendung realer Daten erlangt, die beim alternativen Einsatz eines synthetischen Datensatzes unerkannt geblieben wären. Hierzu zählen vor allem die Behandlung von Zeitstempeln der Einträge sowie die Identifikation neuer Einträge. Auf die Beschreibung der Vorgehensweise folgt eine statistische Auswertung des erlangten Datensatzes sowie die Bewertung der Erfüllung der aufgestellten Anforderungen.

In Kapitel 4 werden die in Kapitel 2 identifizierten Algorithmen aufgegriffen und sofern notwendig an die Domäne Feeds adaptiert. Aus den bekannten Stärken verschiedener Algorithmen wird zudem ein eigener Algorithmus entwickelt, der adaptiv auf Variationen der Änderungsraten reagiert und keine explizite Trainingsphase benötigt. Anschließend werden die in der Evaluation zu verwendenden Metriken vorgestellt und die Güte als zusammenfassendes Kriterium hergeleitet. Basierend auf den Algorithmen und Qualitätskriterien erfolgt eine ausführliche Evaluation der Algorithmen, wobei zunächst die Parameter einiger Algorithmen separat bestimmt und im Anschluss alle Algorithmen untereinander verglichen werden. In der zusammenfassenden Diskussion der Ergebnisse werden Möglichkeiten der Optimierung identifiziert.

Mit der Verfeinerung der Vorhersage werden die Betrachtungen in Kapitel 5 abgerundet. Es erfolgen Variationen verschiedener Parameter wie der Begrenzung des Intervalls zwischen zwei Abfragen und der Anzahl der für die Vorhersage betrachteten Feed-Einträge. Mit der Auswertung abfragespezifischer Attribute wird gezeigt, dass diese keine besseren Vorhersagen ermöglichen. Die Generalisierung des in Kapitel 4 aufgestellten Güte-Kriteriums ermöglicht eine anwendungsspezifische Bewertung der Algorithmen und spannt den Bogen zu Kapitel 6, in dem die Arbeit zusammengefasst wird und ein Ausblick auf mögliche Erweiterungen erfolgt.

2. Analyse des Stands der Technik

In diesem Kapitel werden der Stand der Technik aufgearbeitet und der Forschungsbedarf im Bereich der Integration von Quellen ohne push-Mechanismen aufgezeigt. Einige Techniken werden hierbei detailliert vorgestellt, um sie im weiteren Verlauf der Arbeit an die Domäne Feeds anzupassen und anhand eigener Evaluationsergebnisse zu vergleichen. Die Einordnung der Arbeit in die verwandten Themengebiete erfolgt in Abbildung 2.1. Die im Folgenden sequentiell betrachteten Themengebiete sind vertikal dargestellt, wobei pro Abschnitt die horizontal abgebildeten Aspekte betrachtet werden. Im Fokus der Analysen stehen Web Feeds, sodass sich die herausgearbeiteten Beziehungen und Überlappungen der Domänen nur auf Feeds beziehen. Weitere Zusammenhänge wie die zwischen Datenbanken und Web Caching werden im Text betrachtet, aus Gründen der Übersichtlichkeit jedoch nicht in Abbildung 2.1 dargestellt.

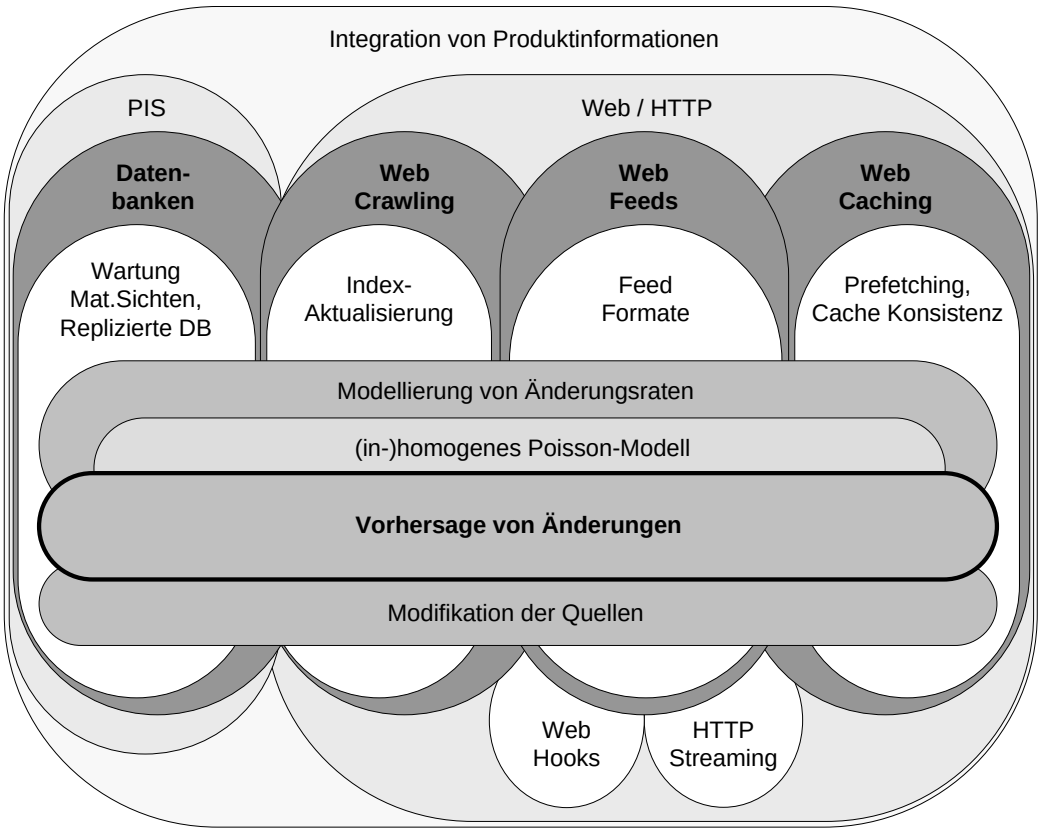

Abbildung 2.1.: Einordnung der Arbeit in verwandte Themengebiete.

Zu Beginn werden die verwendeten Termini eingeführt sowie in Abschnitt 2.1 generische Eigenschaften von Feeds als Datenquellen im Web betrachtet. Anhand dieses Wissens erfolgt in Abschnitt 2.2 die Analyse von PIS sowie ihre Eignung zur Integration von Feeds. Die Erkenntnisse werden zeigen, dass die Systeme nur bedingt zur Integration geeignet sind, sodass der Bogen der Betrachtungen weiter gespannt und Techniken zur Replikation und Synchronisation aus dem Bereich Datenbanken betrachtet werden. Im weiteren Verlauf werden in Abschnitt 2.3 Feeds als zentrales Objekt der Arbeit im Detail analysiert und der Vorhersage dienliche Attribute eruiert. Zur Identifikation weiterer Attribute erstreckt sich die Analyse zudem auf das zur Übertragung von Feeds eingesetzte HTTP. Anhand der gewonnenen Erkenntnisse werden in den Abschnitten 2.4 und 2.5 Systeme aus den Domänen Web-Crawling und Web-Caching betrachtet. Schwerpunkte sind die Aktualisierung von Indizes sowie Techniken zur Wahrung der Cache Konsistenz und zum Prefetching. In einem Exkurs in sonstige Themen im Web-Umfeld in Abschnitt 2.6 werden u. a. Web Hooks und Arbeiten zu HTTP Streaming vorgestellt. In der abschließenden Betrachtung der Domäne Feeds in Abschnitt 2.7 werden vor allem Arbeiten untersucht, die ebenfalls das Ziel verfolgen, neue Einträge in Feeds zeitnah den Interessenten zur Verfügung zu stellen. Die gewonnenen Erkenntnisse bezüglich der in Abbildung 2.1 horizontal dargestellten Aspekte sowie die offenen Forschungsfragen werden in Abschnitt 2.8 zusammengefasst.

Die bereits stillschweigend verwendeten und im weiteren Verlauf häufig benötigten Termini *Produktinformationen* und *Datenquelle* werden zunächst definiert.

Produktinformation Unter dem Begriff Produktinformation werden alle Informationen zusammengefasst, welche für ein Unternehmen von Interesse sind. Hierzu zählen zunächst alle Informationen über ein Produkt, welche innerhalb des Produktlebenszyklus anfallen. Informationen aus unternehmensinternen Quellen wie der Forschungs- und Entwicklungsabteilung oder der Fertigung sind ebenso relevant, wie externe Quellen, in denen sich Nutzer zu einem Produkt äußern. Der Begriff Produkt wird dabei nicht nur auf physische Produkte wie Kleidung oder nicht anfassbare Produkte wie Software beschränkt, er erstreckt sich im erweiterten Rahmen auch auf Dienstleistungen, die von einem Unternehmen erbracht werden. Des Weiteren fallen alle Informationen die sich auf ein Unternehmen im Allgemeinen beziehen unter den Begriff Produktinformationen, da auch Meinungen über ein Unternehmen, die nicht in direktem Zusammenhang mit einem Produkt oder einer Dienstleistung stehen, von wirtschaftlichem Interesse für ein Unternehmen sind. Es wäre daher ebenfalls korrekt, statt Produktinformationen den Begriff Unternehmensinformation zu verwenden, die Bezeichnung Unternehmensinformationssystem ist jedoch weniger gebräuchlich, sodass im weiteren Verlauf der Arbeit Produktinformationen und Unternehmensinformationen als Synonym betrachtet werden.

Datenquelle Die Definition einer Datenquelle, kurz Quelle, ist eng an die Definition in [LN07, S. 7] angelehnt. Die Autoren beschreiben eine Datenquelle als einen beliebig aufgebauten Informationsspeicher, dessen enthaltene Daten in ein integriertes System integriert werden sollen. Der Begriff ist zudem hierarchisch zu verstehen, da ein

integriertes System selbst als Datenquelle für ein weiteres System eingesetzt werden kann. Die Autoren unterscheiden dabei nicht zwischen Daten und Informationen.

2.1. Analyse von Datenquellen im Web

Im Allgemeinen beruhen die Probleme bei der Integration von Datenquellen auf deren Verteilung, Heterogenität und Autonomie. Der in dieser Arbeit im Fokus stehende Aspekt der Aktualisierung ergibt sich aus der Autonomie – genauer gesagt der Schnittstellenautonomie – der Datenquellen, da die Betreiber selbstständig entscheiden können, mit welchen Verfahren auf welche Daten zugegriffen werden kann. Die Autonomie ist vor allem bei Quellen im Web-Umfeld stark ausgeprägt [LN07, S. 54ff].

Aufgrund der (Schnittstellen-)Autonomie der Datenquellen liegt meist eine technische Heterogenität zwischen den Quellen und dem PIS vor, die sich in unterschiedlichen Zugriffsmöglichkeiten auf die Quellen äußert. Dabei kann zwischen Schnittstellenheterogenität und Heterogenität in den Anfragemechanismen unterschieden werden [LN07, S. 62ff]. Unterstützt eine Quelle eine deklarative Anfragesprache wie die Structured Query Language (SQL), so ist die Integration dieser Quelle meist deutlich einfacher als bei einem Web-Service oder einer Programmierschnittstelle (engl. Application Programming Interface (API)), die nur eine Menge an Funktionsaufrufen bereitstellen, welche bestenfalls parametrisiert sind. Quellen müssen daher unterschieden werden zwischen denen mit Mechanismen zur Notifikation über erfolgte Änderungen und jenen ohne. Im Bereich von DWHs ist es z. B. üblich, dass innerhalb des DWH Änderungstabellen des zu integrierenden Quellsystems vorliegen, in denen das Quellsystem selbstständig alle Änderungen in strukturierter Form ablegt, sodass sie jederzeit für das DWH zugreifbar sind. Wie im weiteren Verlauf gezeigt wird, ist dieser Ansatz für einige Web-Quellen nicht praktikabel. Mit den in dieser Arbeit vorgestellten Verfahren wird eine Möglichkeit aufgezeigt, diese Lücke zu schließen, indem ein DWH um Module erweitert wird, welche die Änderungstabellen für Quellen anlegen, die über keine Notifikationsmechanismen verfügen.

2.1.1. Allgemeine Eigenschaften von Quellen

Administrative Grenzen des PIS

Datenquellen können anhand der Eigenschaft unterschieden werden, ob sie innerhalb oder außerhalb der administrativen Grenzen des PIS liegen. Betrachtet man in der verallgemeinerten Architektur eines PIS in Abbildung 1.1 die unterste Ebene, so liegen die ersten beiden Klassen von Quellen – klassische, in Unternehmen bereits bestehende (Legacy-) Systeme und lokale Dateien der Mitarbeiter – primär innerhalb der administrativen Grenzen des integrierenden Systems, während Quellen im Web und Internet der Dinge sowohl innerhalb als auch außerhalb der Unternehmensgrenzen liegen können. In dieser Arbeit liegt der Fokus auf Teilaspekten der Integration von Feeds, sodass im weiteren Verlauf nur Quellen im Web betrachtet werden.

Web-Systeme sind im Internet bereits fest etabliert und erfahren auch innerhalb von Unternehmen eine immer stärkere Verbreitung. Für die Integration dieser Systeme be-

dingt das Unterscheidungsmerkmal der administrativen Grenzen zwei wichtige Merkmale: die *Bekanntheit* und die *Relevanz* der Quellen. Innerhalb der administrativen Grenzen existieren meist organisatorische Strukturen wie Abteilungen mit eigenen Administratoren, über die zu integrierende Quellen ausfindig gemacht werden können. Bei diesen Quellen kann davon ausgegangen werden, dass sie Produktinformationen enthalten und als relevant eingestuft werden können bzw. nicht zu integrierende Bereiche den jeweiligen Administratoren bekannt sind und von der Integration ausgeschlossen werden. Außerhalb der administrativen Grenzen sind diese Voraussetzungen nicht gegeben. Es sind weder alle Quellen bekannt, noch enthalten die bekannten Quellen ausschließlich relevante Informationen. Dies hat zur Folge, dass ein vermeintlich relevantes Dokument zunächst heruntergeladen und analysiert werden muss, um festzustellen, ob es Produktinformationen enthält. Die in der Arbeit vorgestellten Techniken leisten einen Beitrag zur Reduzierung der hierfür benötigten Ressourcen, indem potentiell relevante Dokumente möglichst nur dann heruntergeladen werden, wenn sie sich geändert haben. Der nachgelagerte Schritt der Relevanzbewertung selbst sowie das Auffinden neuer Quellen sind nicht Gegenstand der Arbeit.

Je nach Herkunft der Quelle sind unterschiedliche Aktualisierungsmuster zu erwarten. Befindet sie sich im öffentlichen Web, so können rund um die Uhr Änderungen auftreten, während in unternehmensinternen Quellen Aktualisierungen typischerweise zu den Bürozeiten stattfinden. Genauer betrachtet sind hier Unterschiede zwischen den regulären Bürozeiten, produzierenden Abteilungen mit Mehrschichtbetrieb und vor allem lokal oder global agierenden Unternehmen zu erwarten.

Nicht zuletzt sei an dieser Stelle noch einmal an die Autonomie der Quellen erinnert. Je loser die Quellen administrativ an das zu integrierende System gebunden sind, desto schwerer bis unmöglich ist es, ihre Schnittstellen zu modifizieren. Innerhalb der administrativen Grenzen sind Anpassungen zwar theoretisch einfacher möglich, dennoch ist es häufig nicht gewünscht, überhaupt oder gar stark in bestehende Systeme einzugreifen, getreu dem Leitspruch: „Never touch a running system.“

Zugriffsmechanismen: push und pull

Wie bereits in Abschnitt 2.1 exemplarisch erwähnt, arbeiten DWH auf speziellen Änderungstabellen, in denen neue, geänderte oder aktualisierte Daten der angebundenen Quellen vorliegen. Um diese Tabellen zu befüllen, gibt es zwei Mechanismen: *push* und *pull*. Unterstützt eine Quelle push-Mechanismen, so propagiert sie Änderungen eigenständig an das DWH. Werden diese Mechanismen nicht unterstützt, so muss das DWH eigene Module bereitstellen, um die Daten zu integrieren.

Bei den zur Verfügung stehenden pull-Mechanismen treten deutliche Unterschiede auf. In der einfachsten Form wird bei einem pull das komplette Dokument übertragen. Neben dieser rudimentären Unterstützung bieten viele Systeme ausgefeiltere Mechanismen an, sodass ein Nutzer z. B. anfragen kann, ob ein Dokument seit einem in der Abfrage übergebenen Zeitpunkt geändert wurde oder noch in einer bereits bekannten Version vorliegt. Ein Beispiel für diese Techniken ist HTTP [FGM+99] mit den Header-Elementen Last-Modified und ETag. Darüber hinaus unterstützen einige Systeme nicht nur diese

Booleschen Abfragen, sondern auch die Übertragung von Änderungen, sodass dem Abfragenden bereits bekannte Teile des Dokuments nicht erneut übertragen werden müssen. Ein weiterer interessanter Aspekt sind Mechanismen, mit denen das Quellsystem Angaben zur Gültigkeitsdauer der Dokumente liefern kann. Ist einem Abfragenden bekannt, dass ein Dokument eine Gültigkeitsdauer von einem Tag hat, so kann er den Inhalt bis dahin als aktuell betrachten und das Quellsystem erst nach Ablauf der Gültigkeitsdauer erneut nach Änderungen anfragen.

Ungeachtet der zur Verfügung stehenden Mechanismen muss stets beachtet werden, dass jeder unnötige pull Ressourcen verbraucht, da z. B. eine HTTP-Verbindung aufgebaut werden muss. Werden nur wenige Quellen in das PIS eingebunden, ist dies wohl vernachlässigbar, sind jedoch mehrere Millionen von Quellen angebunden, so muss auch dieser Aspekt betrachtet werden.

2.1.2. Grundlegende Eigenschaften von Feeds

Feeds sind eine Technologie, um Nutzer über Aktualisierungen oder Änderungen von Webseiten zu informieren. Ihr Einsatzgebiet ist sehr breit gefächert und umfasst unter anderem klassische Webseiten, Seiten von Nachrichtenagenturen, Blogs und soziale Netzwerke, Foren und Content Management Systeme im Allgemeinen. In Foren oder auf Entwicklerseiten wie sourceforge[1] werden Feeds eingesetzt, um Nutzer über neue Einträge, Versionen oder aktuelle Aktivitäten in der Entwicklung zu informieren. Nachrichtenagenturen verwenden Feeds, um über aktuelle Geschehnisse zu berichten, in Content Management Systemen werden Nutzer z. B. über neue Produkte oder aktuelle Angebote informiert.

Die Eigenschaft, dass Feeds Aktualisierungen verschiedenartiger Quellen kapseln und somit als Abstraktionsschicht für Aktualisierungen von Webseiten im Allgemeinen gesehen werden können, rückt sie in den Fokus der Betrachtung dieser Arbeit. Bevor in Abschnitt 2.2 bestehende PIS auf ihre Eignung hin untersucht werden, Webseiten im Allgemeinen oder Feeds im Speziellen einzubinden, werden an dieser Stelle noch einige wichtige Eigenschaften von Feeds betrachtet, welche für die Analyse bestehender Systeme von Bedeutung sind. Die hier knapp beschriebenen Eigenschaften werden in Abschnitt 2.3 ausführlich untersucht sowie in Kapitel 3 anhand eines gewonnenen Datensatzes belegt.

Zugriff per pull Auf Feeds wird üblicherweise mit pull-Mechanismen zugegriffen, das push-Verfahren wird nur von einem sehr geringen Anteil der Feeds unterstützt.

Zeitliche Begrenzung Feeds sind üblicherweise nach dem First-In, First-Out (FIFO)-Prinzip aufgebaut, sodass neue Nachrichten am Anfang eingeschoben und in der Reihenfolge des Erscheinens auch wieder aus dem Feed entfernt werden. Im Allgemeinen enthält ein Feed 10–25 Einträge. Ein Eintrag ist daher nur so lange in einem Feed vorhanden, bis er von neueren Einträgen verdrängt wurde.

Aktualisierungsmuster Die Muster, nach denen Feeds aktualisiert werden, weisen eine hohe Diversität auf. Sie unterscheiden sich hauptsächlich in der Länge des Intervalls,

[1] `http://sourceforge.net`

in dem sie aktualisiert werden, und ob dieses konstant oder variabel ist. Die Analysen eines Datensatzes von 200.000 Feeds in Abschnitt 3.7.4 werden zeigen, dass einige Feeds mehrere neue Einträge pro Sekunde haben können, während andere durchschnittlich weniger als einen neuen Eintrag pro Jahr verzeichnen. Das Intervall ist zudem meist variabel.

Diese Merkmale von Feeds untermauern die Anforderung an ein integrierendes System, pull-Mechanismen mit adaptiven Intervallen einzusetzen, um sich an die jeweiligen Eigenschaften eines Feeds anpassen zu können.

2.2. Produktinformationssysteme und Datenbanken

Die Bedeutung der Integration semi- und unstrukturierter Daten aus dem Web wurde von zahlreichen namhaften Persönlichkeiten aus der Domäne Datenbanken in einem aktuellen Bericht untermauert, in dem die Integration als eine der wichtigsten aktuellen und zukünftigen Forschungsrichtungen in diesem Feld dargestellt wird [AAB+09].

2.2.1. Genereller Aufbau von Produktinformationssystemen

Grundlegend wird zwischen *Informationsintegration* und *Anwendungsintegration* unterschieden. Bei der Informationsintegration werden die Datenbestände in ein einheitliches System integriert, bei der Anwendungsintegration findet eine Integration von IT-Prozessen statt, sodass ein Austausch von Nachrichten zwischen Systemen erfolgt [LN07].

Systeme zur Informationsintegration werden in *virtuelle* und *materialisierte* Integration aufgesplittet. Bei der materialisierten Integration werden die Daten in das integrierende System kopiert. Data Warehouses sind der typischste Vertreter dieser Klasse von Informationssystemen. Bei der virtuellen Integration verweilen die Daten in den Datenquellen und werden erst zur Anfragezeit aus den Quellen geladen, ein typischer Vertreter sind föderierte Datenbanken. Im weiteren Verlauf werden DWHs genauer betrachtet. Laut [LN07, S. 372] sind DWHs heute[2] die wichtigsten auf Informationsintegration basierenden Systeme in Unternehmen.

In einem klassischen materialisierten Informationsintegrations-System werden Änderungen der Quelldaten in geeigneter Weise bereitgestellt. Eine Integrationskomponente des Informationsintegrations-System hat Zugriff auf diese und bestimmt eigenmächtig, welche Änderungen in welcher Reihenfolge in das zentrale System übernommen werden. Zur Integration dieser Daten sind zwei Aspekte von Interesse: die Form, in der Änderungen bereitgestellt werden und die Zugriffsmethoden auf diese Daten. Die Aspekte werden zunächst einzeln und anschließend am Beispiel vorgestellt.

2.2.2. Darstellung von Änderungen

Änderungs-Logdateien enthalten alle Modifikationen (hinzufügen, ändern, löschen) einer Quelle, sodass diese lückenlos nachvollzogen werden können. Die Dateien werden von den

[2] Die zitierte Quelle wurde 2007 veröffentlicht.

Quellen selbst erstellt und der Integrationskomponente zur Verfügung gestellt [JD08].

Als *Audit Columns* werden zusätzliche Spalten einer Datenbank bezeichnet, in denen zu jedem neuen oder geänderten Tupel ein Zeitstempel oder eine andere adäquate Versionsinformation geschrieben wird. Die Integrationskomponente hat Zugriff auf die Datenbank und kann entsprechende Abfragen stellen, um z. B. alle Änderungen seit der letzten Abfrage zu erhalten. Der Nachteil des Verfahrens ist, dass der Integrationskomponente nicht angezeigt werden kann, wenn Tupel gelöscht werden. Je nach Einsatzszenario des DWH ist dies nicht relevant [JD08].

Unterstützt eine Quelle weder Logdateien noch Audit Columns, so muss ein *Schnappschuss* (engl. Snapshot) ihrer Daten erstellt und in das System integriert werden. Um Änderungen zu detektieren, werden Schnappschüsse mit geeigneten – unter Umständen quellenspezifischen – *diff*-Algorithmen miteinander verglichen [JD08].

Legacy-Systeme, die z. B. auf COBOL oder PL/1 basieren, sind in der Regel Quellen mit beschränktem Funktionsumfang. Es wird empfohlen, die regelmäßig in den Quellen erstellten Reports zyklisch zu materialisieren [LN07, S. 250ff]. Vor gut zehn Jahren war es üblich, die Daten eines DWH nur täglich, wöchentlich oder gar monatlich zu aktualisieren [ZK01].

Webdatenquellen werden als immer bedeutender angesehen [LN07, AAB+09], die Meinungen zur Integration von Daten aus dem Web divergieren jedoch. Arbeiten zur materialisierten Integration erfolgen im Bereich Web Crawling und werden in Abschnitt 2.4 betrachtet. Dem gegenüber stehen Argumente zur virtuellen Integration von Webdatenquellen, da bei diesen über Formulare oder Web-Services verfügbaren Daten normalerweise nicht auf den kompletten Datenbestand zugegriffen werden kann [LN07, S. 250ff]. Aus diesem Grund wird der Einsatz von Wrappern vorgeschlagen, deren Architektur in Abbildung 2.2 dargestellt ist. Der Einsatz eines solchen Wrappers wird u. a. in [ZAL08] in Form eines Data-Capture-Service vorgestellt, welcher Daten verschiedener Webquellen einheitlich zur Verfügung stellt. Angaben, mit welcher Strategie Quellen angefragt werden, fehlen. Bereits in den frühen 90er Jahren wurden zudem Mediatoren vorgestellt, um das Problem der Integration heterogener Quellen anzugehen [Wie92]. Mediatoren agieren zwischen Integrationssystem und Datenquelle bzw. Integrationssystem und Wrapper.

2.2.3. Aktualisierungsstrategien im Umfeld von Datenbanken

Die in DWHs eingesetzten Aktualisierungsstrategien werden analog in verschiedenen Bereichen wie der Wartung materialisierter Sichten (engl. maintenance of materialized views) und replizierten Datenbanken eingesetzt [OV11]. Der Unterschied, dass in DWHs meist die komplette Historie der Daten vorgehalten wird, materialisierte Sichten hingegen nur den aktuellen Zustand der Basisrelationen reflektieren [JD09], ist für den Blickwinkel dieser Arbeit nicht relevant.

Zur Wartung materialisierter Sichten existiert in der Regel eine *view maintenance policy*, in der angegeben ist *wie* und *wann* eine Sicht zu aktualisieren ist. Fokus der Forschungsgebiete Wartung materialisierter Sichten und DWHs sind datenbankspezifische Verfahren zur Beantwortung der Frage, *wie* die integrierenden Systeme aktualisiert werden sollen. Diese sind für die vorliegende Arbeit nicht von Interesse. Die Frage nach dem Aktualisierungszeitpunkt wird auch als Change Data Capture (CDC) bezeichnet [JD08].

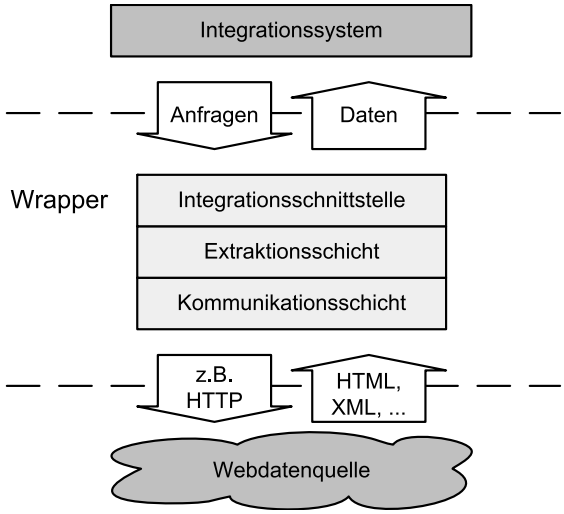

Abbildung 2.2.: Aufbau eines Web-Wrappers nach [LN07, S. 254].

Der Zeitpunkt beeinflusst die Konsistenz zwischen der Sicht und den Basisdaten sowie die Effizienz des Gesamtsystems. Zwei Klassen werden unterschieden: die *synchrone* (sofortige) und die *asynchrone* (verzögerte) Aktualisierung. Bei der synchronen Aktualisierung werden die Sichten gemeinsam mit den Basisdaten in einer Transaktion modifiziert. Werden die Basisdaten und die Sicht von verschiedenen, unter Umständen verteilten, Datenbank-Management-Systemen verwaltet, so muss ein Protokoll wie das Zwei-Phasen-Commit-Protokoll eingesetzt werden, um verteilte Transaktionen zu realisieren.

In der Praxis wird häufig die asynchrone Aktualisierung bevorzugt, um die Performanz der Transaktionen auf den Basisdaten zu erhöhen. Das Laden neuer Daten geschieht meist über herstellerspezifische Bulk-Load-Verfahren, da diese viel schneller sind, als die Tupel einzeln zu laden. Drei Modi der asynchronen Aktualisierung sind zu unterscheiden: *träge* (engl. lazy), periodisch und verzögert (engl. deferred). Bei einer trägen Aktualisierung wird eine Sicht erst direkt vor der Bearbeitung einer Abfrage auf diese Sicht aktualisiert. Periodische Aktualisierungen finden zu vordefinierten Zeiten statt, meist zu Zeiten geringer Systemlast. Das Konzept von verzögerten Aktualisierungen ist, die Anzahl der Änderungen der Basisdaten zu zählen und beim Erreichen einer vordefinierten oberen Schranke die Sichten zu aktualisieren. Die Modi periodisch und verzögert werden auch als Schnappschüsse bezeichnet. Ein Nachteil dieser Methoden ist, dass die Schnappschüsse häufig erstellt werden, weshalb einen hohes Volumen zu verarbeitender Daten entsteht [JD08]. Ein weiterer Nachteil ist, dass neue, in den Quellen bereits vorhandene aber noch nicht integrierte Daten nicht zur Beantwortung von Anfragen herangezogen werden, weshalb Ansätze zur materialisierten Integration häufig eine mangelnde Aktualität der Daten aufweisen [LN07, S. 372].

Die Verfahren zur Aktualisierung replizierter Datenbanken ähneln den soeben eingeführten zur Wartung materialisierter Sichten. Änderungen an der Primärkopie oder einer ihrer Replikate müssen an alle anderen propagiert werden. Die eingesetzten Verfahren

wie Read-One-Write-All (ROWA), Read-One-Write-All-Available (ROWAA), Primary Copy (PC), Voting-Verfahren, Schnappschüsse, Lazy Update, Quasi-Kopien und Epsilon-Serialisierbarkeit basieren entweder auf der aktiven Teilnahme der Quelle, periodischen Abfragen oder Abfragen zu Zeiten geringer Systemlast [Dad96, BN97, OV11].

Zur Verarbeitung von kontinuierlichen Datenströmen, die z. B. aus dem Internet der Dinge stammen, werden Data Stream Management Systeme (DSMS) eingesetzt [OV11]. In dem DSMS zugrundeliegenden Modell wird angenommen, dass Daten kontinuierlich und in fester Reihenfolge generiert werden. Wie bereits von DWHs bekannt, wird auch bei DSMS ein push-basierter Ansatz verfolgt, sodass Datenquellen neue Daten selbstständig dem System übergeben. Aus diesem Grund werden keine Strategien zum intelligenten Abfragen der Quelle benötigt, wie es beim pull-basierten Zugriff auf Feeds der Fall ist.

Ein verwandtes Gebiet sind *real-time* DWHs, die ebenfalls zur Verarbeitung eingehender Datenströme entwickelt werden. Zu den Forschungsaufgaben zählt u. a. die Frage, in welcher Reihenfolge die kontinuierlich eingehenden Daten in das DWH eingepflegt werden und ob Abfragen zwischenzeitlich oder erst nach der Ausführung aller relevanten Schreiboperationen beantwortet werden [TL09].

Zusammenfassend kann festgestellt werden, dass die Methoden, *wann* eine Sicht oder ein Replikat zu aktualisieren ist, nur bedingte Überschneidungen mit den Zielen dieser Arbeit haben. Feeds weisen eine begrenzte Historie von einem Fenster auf, sodass die Methoden zur Wartung materialisierter Sichten nicht zur Integration von Feeds geeignet sind. Eine synchrone Aktualisierung ist aufgrund der Quellenautonomie nicht möglich. Bei der trägen Aktualisierung besteht die Gefahr, dass Einträge verpasst werden, periodische Aktualisierungen mit statischen Intervallen bergen die Gefahr, bei zu kleinen Perioden eine zu hohe Last zu erzeugen und bei zu großem Intervall Änderungen zu verpassen. Verzögerte Aktualisierungen sind aufgrund der Quellenautonomie ebenfalls nicht möglich, da die Quelle über keinen Änderungszähler verfügt.

Frequent Pattern Mining (FPM) – das Auffinden häufiger Muster – ist ein Teilgebiet des Data Mining. Ziel ist es z. B. Assoziationsregeln, Korrelationen, Sequenzen, Episoden, Klassifikatoren oder Cluster zu finden, wobei das Auffinden von Assoziationsregeln eines der bekanntesten Probleme ist. Mit Assoziationsregeln werden u. a. itemsets beschrieben. Itemsets sind Produkte, die gemeinsam in einem Supermarkt gekauft werden. Das Auffinden von itemsets und Ableiten von Assoziationsregeln ist der Fokus in [Goe02], das die Erkenntnisse eines Jahrzehnts in diesem Bereich zusammenfasst. Sequenzen und Episoden werden nicht betrachtet.

Sequential Patterns sind Muster von Ereignissen, die in einer bestimmten Reihenfolge auftreten. Beispiel eines Ereignisses ist „Produkt X wurde von Kunde Y gekauft“. In Arbeiten wie [RP08] werden neben der puren Reihenfolge auch diskrete Zeitstempel als Eigenschaften der Ereignisse erfasst, eine Auswertung bezüglich der Vorhersage eines zukünftigen Auftretens des Ereignisses wird nicht getroffen. Ein verwandtes Gebiet ist die Analyse von Verkehrsströmen. Die von Sensoren generierten Datenströme werden u. a. auf Muster untersucht und darauf aufbauend Vorhersagen von Ereignissen wie Staus getroffen [LWC08]. Im Unterschied zum eigenen Ansatz, in dem die Zeitpunkte neuer Einträge von Feeds von Interesse sind, werden in [LWC08] keine Vorhersagen über einzelne

Fahrzeuge getroffen, sondern vielmehr verschiedene Verkehrsdichten analysiert, die zu Beeinträchtigungen führen können.

Den Autoren in [CC09] zufolge ist es eine weit verbreitete Meinung, dass Zeitreihen von Ereignissen mit feingranularen Zeitstempeln mit einer Tag- oder gar Stunden-genauen Auflösung zu viele Werte enthalten, von denen viele nur Rauschen sind. Die Analyse von Trends, die sich aus diesen Zeitreihen ableiten lassen, ist daher unnötig aufwendig. Aus diesen beiden Gründen schlagen sie vor, die diskreten Zeitpunkte auf Zeitintervalle abzubilden und die Vorhersagen auf deren Basis zu berechnen. Dieses Vorgehen ist auch unter dem Begriff Temporale Segmentation bekannt. Die Autoren beschränken sich jedoch auf die Vorstellung eines Algorithmus zur Generierung der Zeitintervalle und treffen keine Aussage zu Vorhersagen.

2.2.4. Betrachtung einiger Systeme im Detail

Das Stanford Data Warehousing Project (WHIPS) [HGW+95] hatte das Ziel der Integration von Informationen aus heterogenen, autonomen Quellen wie Legacy Systemen. Hierfür schlagen die Autoren in ihrer Architektur die in Abbildung 2.3 als Monitor bezeichneten Komponenten vor, die zu integrierende Quellen überwachen und Änderungen an einen Integrator weiterleiten. Für Quellen, welche nicht modifizierbar sind, über keine Notifikationsmechanismen wie Trigger verfügen und keine Änderungs-Log-Dateien bereitstellen, werden zwei Optionen vorgeschlagen. Die erste Idee ist, statt die Quelle selbst zu modifizieren, alle Anwendungen zu erweitern, welche Änderungen an der Quelle vornehmen, sodass sie auf Anwendungsebene Nachrichten an den Monitor schicken. Alternativ schlagen die Autoren vor, mit Hilfe von zusätzlicher Software periodische Abbilder (dumps) der Quellen zu erstellen und diese dann in das DWH zu integrieren. Die Autoren bevorzugen diesen zweiten Vorschlag und wollen in künftigen Arbeiten auch Webquellen betrachten, Vorhersagen von Aktualisierungen der Quellen werden jedoch weder in dieser Arbeit noch später im Projekt[3] getroffen.

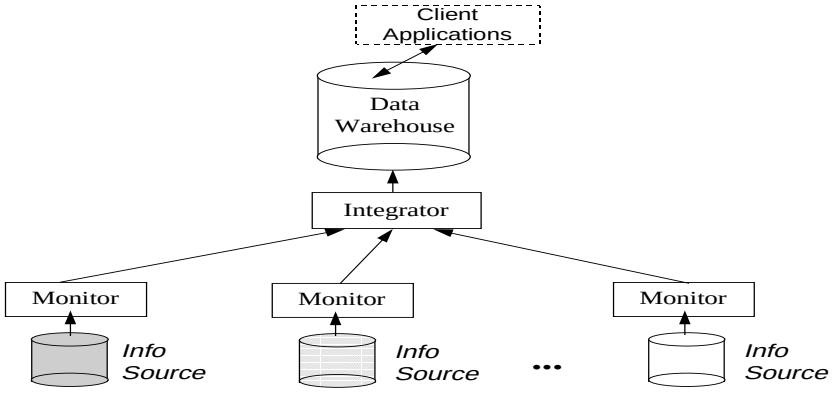

Abbildung 2.3.: Architektur des WHIPS-Projekts [HGW+95].

[3] `http://infolab.stanford.edu/warehousing/publications.html`, letzter Zugriff Juni 2011

Ziel des Projektes „TSIMMIS“ – The Stanford-IBM Manager of Multiple Information Sources – war die Integration heterogener Quellen. Das System basiert auf Mediatoren, welche den Funktionsumfang der Datenquellen kennen und die Abfragen entsprechend stellen. Aktualisierungsstrategien wurden nicht betrachtet [GPQ+97].

Der Fokus von „Jedi“ – Java based Extraction and Dissemination of Information – liegt in der Erstellung von Wrappern und Mediatoren sowie Algorithmen zum Extrahieren von Informationen aus Webseiten. Der Problematik des Zugriffs auf Quellen wird keine Beachtung geschenkt [HFAN98].

In [SA01] wird das Framework „World Wide Web Wrapper Factory (W4F)“ zur Generierung von Wrappern für Webseiten vorgestellt. Den für die vorliegende Arbeit interessanten Aspekt, wann und wie eine Webseite aufgerufen wird, erledigt in der W4F-Architektur ein Retrieval Agent, der jedoch als einzigen Parameter einen Uniform Resource Locator (URL) übergeben bekommt. Anhand des geschilderten Anwendungsszenarios ist zu vermuten, dass es sich um eine virtuelle Integration handelt. Die Autoren schlagen zwar eine Verwendung des W4F-Frameworks in einem DWH vor, den hierfür relevanten Aspekt der Aktualisierung ignorieren sie jedoch. Im Ausblick ihrer Arbeit erwähnen die Autoren, dass Änderungen von Webseiten betrachtet werden müssen, diese werden jedoch auf Modifikationen des von der Seite verwendeten Templates eingeschränkt. Die Betrachtung von Änderungen des Inhalts selbst wird explizit ausgeschlossen.

Aktuelle Arbeiten zur Datenintegration – vor allem aus dem *Deep Web* – sind eher in die Domäne Web Crawling einzuordnen und werden in Abschnitt 2.4 beleuchtet.

2.3. Grundlagen zu Feed-Formaten und HTTP

Im vorangegangenen Abschnitt 2.2 wurde gezeigt, dass aktuelle PIS nur bedingt zur Integration von Feeds geeignet sind. Aus diesem Grund werden im aktuellen Abschnitt Details zu den zu integrierenden Feeds sowie des zur Übertragung eingesetzten Protokolls betrachtet. Auf diesen bauen sowohl verwandte Arbeiten als auch eigene Techniken auf, welche im Mittelpunkt der Untersuchungen in den folgenden Kapiteln der vorliegenden Arbeit stehen.

2.3.1. Feedspezifische Eigenschaften

Feeds werden hauptsächlich in einem der beiden Formate Really Simple Syndication (RSS) und Atom publiziert, wobei beide Formate mehrere Unterformate bieten. Allen Formaten und Unterformaten ist gemein, dass sie als XML-Dokumente verbreitet werden. RSS 2.0 ist die am weitesten verbreitete Version des RSS-Formats, Atom 1.0 der häufigste Vertreter des Atom-Formats. Im Folgenden werden nur die beiden häufigsten Vertreter RSS 2.0 und Atom 1.0 vertieft betrachtet, da die anderen Unterformate nur einen geringen Anteil von fünf Prozent in der Verteilung der Formate ausmachen, wie später in Abschnitt 3.7.1 (Seite 96) gezeigt wird.

Allgemein besteht ein Feed aus einer Reihe von Feed-Einträgen, welche in RSS 2.0 item und in Atom 1.0 entry genannt werden. Im weiteren Verlauf der Arbeit werden beide ge-

nerisch als Feed-Einträge bezeichnet und nur dann die formatspezifischen Bezeichnungen item und entry verwendet, wenn Besonderheiten der Formate vorgestellt werden.

2.3.2. RSS 2.0

Wie bereits eingeführt ist RSS 2.0 die am weitesten verbreitete Version des RSS-Formats. Die folgende knappe Beschreibung des seit 2002 fixierten, jedoch bisher nicht standardisierten Formats fokussiert auf strukturelle Eigenschaften und Elemente, die einer Vorhersage neuer Feed-Einträge dienlich sind. Für eine umfassende Beschreibung sei auf [Win03a] verwiesen. Listing 2.1 zeigt einen einfachen, exemplarischen Feed.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rss version="2.0">
3   <channel>
4     <title>Example Feed</title>
5     <description>Insert witty or insightful remark here</description>
6     <link>http://example.org/</link>
7     <lastBuildDate>Wed, 24 Dec 2003 12:14:12 GMT</lastBuildDate>
8     <ttl>60</ttl>
9     <skipHours>
10      <hour>0</hour>
11      <hour>1</hour>
12    </skipHours>
13    <skipDays>
14      <day>Sunday<day>
15    </skipDays>
16    <item>
17      <title>Atom-Powered Robots Run Amok</title>
18      <description>Some text.</description>
19      <link>http://example.org/2003/12/13/atom03</link>
20      <guid>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</guid>
21      <pubDate>Sat, 13 Dec 2003 18:30:02 GMT</pubDate>
22    </item>
23    <item>
24      <title>Santa is in town</title>
25      <description>Some other text.</description>
26      <link>http://example.org/2003/12/24/atom24</link>
27      <guid>http://www.example.com/weblogItem2412</guid>
28      <pubDate>Wed, 24 Dec 2003 12:14:12 GMT</pubDate>
29    </item>
30  </channel>
31 </rss>
```

Listing 2.1: Beispiel eines RSS 2.0 Feeds, basierend auf [Mei06].

Ein RSS 2.0 Feed besteht aus genau einem channel-Element, welches theoretisch beliebig viele item-Elemente enthalten kann, die jeweils einen Eintrag des Feeds repräsentieren. Die Anzahl dieser Elemente bezeichnet man auch als *Fenstergröße* des Feeds, im Beispiel in Listing 2.1 hat der Feed eine Fenstergröße von zwei. Gebräuchliche Fenstergrößen liegen bei 10-25 Einträgen, wie später in Abschnitt 3.7.2 gezeigt wird. Die RSS 2.0 Spezifikation trifft keine Aussage zu empfohlenen typischen, minimalen oder maximalen Fenstergrößen.

Sowohl der channel als auch die items müssen über die Elemente title, description und link verfügen. Des Weiteren umfasst das Format einen Satz fakultativer Elemente um weitere Informationen und Metadaten zu übertragen. Die für diese Arbeit interessanten, fakultativen Elemente eines channels sind pubDate, lastBuildDate, cloud, ttl, skipHours und skipDays, bei items sind pubDate und guid von Interesse. Weder die Reihenfolge dieser Elemente noch die der items innerhalb des channel werden in der RSS 2.0 Spezifikation festgelegt, sodass ein Konsument insbesondere keine chronologische Reihenfolge der Einträge erwarten darf.

title enthält den Titel des channels oder eines einzelnen items.

description Bezieht sich das Feld auf den channel, so sollte es eine kurze und aussagekräftige Beschreibung des zu erwartenden Inhalts des Feeds enthalten. Ist description ein Kind-Element eines items, so enthält es den eigentlichen Inhalt dieses items, der üblicherweise aus einem knappen Text besteht, welcher häufig mittels Hyper Text Markup Language (HTML) formatiert und um eine Grafik angereichert ist. Es ist üblich, dass das description-Feld eines items nur eine Kurzfassung oder einen Ausschnitt des Textes enthält, der auf der korrespondierenden Webseite verfügbar ist.

link Der Link enthält einen URL und verweist entweder auf die dem channel oder item entsprechende Webseite.

pubDate Das Publikationsdatum kann sowohl als Subelement des channels als auch eines items verwendet werden. In beiden Fällen enthält es einen Zeitstempel, welcher der Spezifikation RFC 822 [Cro82] unterliegt, wobei die Einschränkung getroffen wurde, dass die Jahreszahl statt mit vier alternativ auch mit zwei Stellen angegeben werden kann.

Wenn sich das pubDate auf ein item bezieht, so gibt es an, wann dieses item publiziert wurde. Liegt der Zeitpunkt in der Zukunft, sollte das item von einem Feed Reader auch erst ab diesem Zeitpunkt dem Nutzer angezeigt werden. Aussagen zur Interpretation in der Zukunft liegender Daten durch andere Konsumenten werden nicht getroffen.

Bezieht sich das pubDate auf den channel, ist seine Semantik nicht eindeutig spezifiziert. In [Win03a] wird angegeben, dass es das Publikationsdatum des Inhalts des channels enthält. Als Beispiel wird die New York Times angegeben, die auf einer täglichen Basis publiziert und sich folglich das Datum alle 24 Stunden ändert. Dieses Beispiel ist jedoch inkonsequent, da in einem Feed fortlaufend neue items eingestellt werden, sodass je nach der Fenstergröße und der Anzahl neu hinzukommender items auch Einträge mehrerer Tage in einem channel enthalten sein können. Anders betrachtet bedeutet es, dass gleiche Einträge an unterschiedlichen Tagen mit unterschiedlichen pubDate-Werten aufgefunden werden können.

lastBuildDate enthält den Zeitstempel, an dem der channel das letzte Mal geändert wurde. Das Format entspricht dem des pubDate. Im aktuellen Beispiel in Listing 2.1 entspricht das lastBuildDate dem pubDate des neuesten items. Dieser Zusammenhang

ist jedoch nicht immer gegeben, da sich zum Beispiel Metadaten wie der Titel des channels unabhängig vom Erscheinen neuer items ändern können und somit das lastBuildDate auch neuer als das aktuellste pubDate eines items sein kann. Ebenso kann es jedoch vorkommen, dass alle pubDate-Werte der items in der Zukunft liegen und das lastBuildDate den ältesten Zeitstempel aufweist.

cloud Das cloud-Element spezifiziert eine Web-Applikation des Feed Providers, bei der sich ein Konsument registrieren kann, um über Aktualisierungen eines channels informiert zu werden. Dieser publish-subscribe-Mechanismus basiert wahlweise auf eXtensible Markup Language (XML)-Remote Procedure Call (RPC) [Win99] oder SOAP [BEK+00]. Dieser Mechanismus hat sich in der Praxis nicht durchgesetzt, nur 3,7 % der 200.000 Feeds des gewonnenen Datensatzes verfügen über dieses Element, sodass an dieser Stelle auf die Vorstellung weiterer Details verzichtet wird. Dem geneigten Leser sei [Win03b] empfohlen.

Würde jeder Diensteanbieter diesen Mechanismus bereitstellen, wäre eine intelligente Vorhersage für Feeds im Format RSS 2.0 nicht nötig. Ungeachtet dessen bliebe das Problem der intelligenten Aktualisierung für alle anderen Formate bestehen.

ttl Die time to live gibt die Zeitspanne in Minuten an, die der Feed in einem Cache vorgehalten werden kann. Üblicherweise fragt ein Konsument die Ressource erst nach Ablauf dieser Zeitspanne erneut an, im aktuellen Beispiel nach 60 Minuten.

skipHours enthält bis zu 24 hour-Elemente, welche je genau einen der Werte 0-23 annehmen können. Der Betreiber des Feeds gibt damit an, dass innerhalb dieser Stunden keine Updates stattfinden, weshalb ein Konsument die Ressource in dieser Zeit üblicherweise nicht auf Aktualisierungen überprüft. Auf diese Weise können z. B. Nachtstunden ohne Aktivitäten des Feeds überbrückt werden.

skipDays enthält analog zu skipHours maximal sieben day-Elemente mit je genau einem der Werte Monday, Tuesday, Wednesday, Thursday, Friday, Saturday oder Sunday. Das skipDays-Element kann verwendet werden, um z. B. Wochenenden zu überbrücken.

RSS 2.0 schließt die Angabe *aller* sieben day- bzw. 24 hour-Elemente innerhalb eines channels nicht aus, sodass ein Diensteanbieter vorgeben kann, dass sich der Inhalt des Feeds nie ändern wird.

guid Der Globally Unique Identifier (GUID) ist ein global eindeutiger Identifikator eines items. Die Syntax unterliegt keinen Bildungsvorschriften, sodass die guid wie in Listing 2.1 eine 128 Bit HEX-Zahl, eine gültige URL oder eine beliebige Zeichenkette sein kann. Die guid kann von Konsumenten verwendet werden, um neue items innerhalb eines channels zu erkennen.

Die RSS 2.0 Spezifikation trifft keine Aussage, wie sich einander widersprechende Werte verschiedener Felder zu interpretieren sind. Aus den Metadaten des channels in Listing 2.1 ist zu lesen, dass der Feed 60 Minuten im Cache vorgehalten werden kann

(ttl-Feld) und weder Sonntags noch in den Nachtstunden zwischen Mitternacht und zwei Uhr morgens aktualisiert wird (skipDays und skipHours). Angenommen der Feed wird an einem Samstag um 23:30 Uhr von einem Konsumenten angefragt, so ist nicht eindeutig spezifiziert, wann der Konsument frühestens erneut anfragen sollte. Intuitiv kann davon ausgegangen werden, dass die Parameter ttl, skipHours und skipDays separat zu betrachten sind und der späteste Zeitpunkt zu verwenden ist, dies ist aber nicht spezifiziert. Im aktuellen Beispiel würde der Konsument den Feed demnach frühestens am darauffolgenden Montag 2:00 Uhr nachts erneut anfragen.

2.3.3. Atom 1.0

Im Gegensatz zu RSS 2.0 wurde Atom 1.0 bereits im Jahr 2005 von der Internet Engineering Task Force als RFC 4287 standardisiert [NS05]. Listing 2.2 zeigt einen einfachen Atom 1.0 Feed, wobei nur grundlegende sowie der Vorhersage neuer Feed-Einträge dienliche Elemente erläutert werden. An der kompletten Spezifikation interessierte Leser seien auf [NS05] verwiesen. Der Standard ist erweiterbar, es sind jedoch keine Erweiterungen bekannt, die für eine adaptive Vorhersage neuer Einträge eingesetzt werden können.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <feed xmlns="http://www.w3.org/2005/Atom">
3   <title type="text">dive into mark</title>
4   <updated>2005-07-31T12:29:29Z</updated>
5   <id>tag:example.org,2003:3</id>
6   <entry>
7     <title>Atom draft-07 snapshot</title>
8     <link rel="alternate" type="text/html" href="http://example.org
          /2005/04/02/atom"/>
9     <id>tag:example.org,2003:3.2397</id>
10    <updated>2005-07-31T12:29:29Z</updated>
11    <published>2003-12-13T08:29:29-04:00</published>
12    <author>
13      <name>Mark Pilgrim</name>
14      <uri>http://example.org/</uri>
15      <email>f8dy@example.com</email>
16    </author>
17    <content type="xhtml" xml:lang="en" xml:base="http://diveintomark.
          org/">
18      <div xmlns="http://www.w3.org/1999/xhtml">
19        <p><i>[Update: The Atom draft is finished.]</i></p>
20      </div>
21    </content>
22  </entry>
23 </feed>
```

Listing 2.2: Beispiel eines Atom 1.0 Feeds, basierend auf [NS05].

Grundsätzlich hat Atom 1.0 einen ähnlichen Aufbau wie RSS 2.0. Ein atom:feed-Element entspricht dem channel in RSS 2.0 und das atom:entry-Element ist das Pendant des items in RSS 2.0. Auf obligatorische, selbsterklärende Elemente wie atom:title oder atom:content wird nicht näher eingegangen, zudem wird im Folgenden der in der Spezifikation stets angegebene Präfix atom: zur Beschreibung der Elemente nicht aufgeführt.

Ein Atom 1.0 Feed besteht in der Regel aus genau einem feed-Element, welches einen Satz von Metadaten sowie theoretisch beliebig viele entry-Elemente enthalten kann. Der Standard sieht zudem Atom 1.0 Feeds vor, welche kein feed-Element, sondern ausschließlich ein einzelnes entry-Element enthalten. Diese Unterscheidung ist für die Vorhersage von Aktualisierungen nicht relevant, da diese auf entry-Elementen basiert. Jedes entry-Element repräsentiert einen individuellen Eintrag des Feeds, der sich aus verschiedenen Daten- und Metadatenfeldern zusammensetzt. Die Reihenfolge der entry-Elemente innerhalb eines Feeds ist beliebig, womit sie insbesondere keiner chronologischen Ordnungsrelation unterliegen. Das Atom 1.0 Format trifft – wie bereits bei RSS 2.0 beobachtet – keine Aussage zu empfohlenen typischen, minimalen oder maximalen Fenstergrößen.

updated Sowohl das feed-Element als auch jedes entry-Element müssen über genau ein updated-Element verfügen, welches den Zeitstempel der letzten signifikanten Änderung des Feeds oder Eintrags enthält. Wird ein bereits veröffentlichter Eintrag modifiziert, so obliegt es dem Diensteanbieter, ob er die Änderung als signifikant einschätzt und den Zeitstempel ändert oder ob er den bisherigen unverändert beibehält. Zeitstempel müssen konform zu RFC 3339 [KN02] sein, wobei die zusätzliche Anforderung gilt, dass Datum und Uhrzeit durch ein „T“ voneinander getrennt sind und die fehlende Angabe einer (numerischen) Zeitzone durch ein „Z“ gekennzeichnet ist. Zeitstempel sollten so genau wie möglich sein, sodass Feed-Einträge die zu verschiedenen Zeitpunkten veröffentlicht wurden auch verschiedene Zeitstempel haben sollten.

published Mit einem published-Element kann ein Diensteanbieter angeben, wann ein Feed-Eintrag zum ersten Mal in diesem Feed veröffentlicht wurde. Es ist ein fakultativer Kind-Knoten des entry-Elements und der enthaltene Zeitstempel unterliegt den gleichen Anforderungen wie der des updated-Feldes.

id Sowohl das feed-Element als auch jedes entry-Element müssen über genau ein id-Element verfügen, welches einen einmaligen (engl. unique) Internationalized Resource Identifier (IRI) gemäß RFC 3987 [DS05] enthält. Dieser darf nie geändert werden. Der Definition, dass eine id einen entry zwar eindeutig identifiziert, laut Spezifikation [NS05] aber mehrere entry-Elemente die gleiche id haben dürfen, ist zunächst verwirrend. entry-Elemente mit gleicher id werden als ein einziges Element betrachtet, sie sollten jedoch verschiedene updated-Zeitstempel haben und können als verschiedene Revisionen eines Eintrags interpretiert werden. Einem Konsumenten wie z. B. einem Feed-Reader steht es frei, ob er dem Leser alle Elemente, eine Teilmenge oder nur einen Eintrag präsentiert. Die alleinige Anzeige des aktuellsten Eintrags wird empfohlen.

source Wenn ein Feed-Aggregator einen Eintrag eines Feeds in einen anderen übernimmt, sollte er dem übernommenen Eintrag im Ziel-Feed ein source-Element hinzufügen, welches die Metadaten wie id, title und updated des Quell-entry enthält sowie ggf. weitere Metadaten des Quell-Feeds selbst. Das source-Feld ist jedoch fakultativ, sodass aus der Abwesenheit des source-Feldes nicht geschlossen werden kann, dass der betrachtete Feed auch tatsächlich die erste Quelle des Feed-Eintrags ist.

2.3.4. Kritische Betrachtung der Feed-Formate

Basierend auf dem vermittelten grundlegenden Verständnis der Formate folgen einige kritische Betrachtungen, die für die Erstellung des Datensatzes von Bedeutung sind.

Aggregation von Atom 1.0 Feeds

Feed Aggregatoren können mehrere Atom 1.0 Feeds zu einem einzigen Atom 1.0 Feed zusammenfassen, die Spezifikation [NS05] trifft jedoch explizit keine Aussage, wie solch eine Aggregation im Detail auszuführen ist. Ein sich für die Vorhersage neuer Einträge ergebendes Problem ist, dass bei der Aggregation von Einträgen auch deren Metadaten unverändert übernommen werden können, sodass die Einträge des aggregierten Feeds unter Umständen veraltete Werte in den updated- und published-Feldern enthalten können. Angenommen der Feed aus Listing 2.2 wird in einem anderen Feed aggregiert, der in Listing 2.3 verkürzt dargestellt ist.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <feed xmlns="http://www.w3.org/2005/Atom">
3   <title type="text">dive into mark</title>
4   <updated>2005-08-31T00:00:00Z</updated>
5   <id>tag:example.org,2005:8</id>
6   <entry>
7     <title>Atom draft-07 snapshot</title>
8     <id>tag:example.org,2003:3.2397</id>
9     <updated>2005-07-31T12:29:29Z</updated>
10    <published>2003-12-13T08:29:29-04:00</published>
11    <content type="xhtml" xml:lang="en" xml:base="http://diveintomark.
          org/">
12      <div xmlns="http://www.w3.org/1999/xhtml">
13        <p><i>[Update: The Atom draft is finished.]</i></p>
14      </div>
15    </content>
16  </entry>
17 </feed>
```

Listing 2.3: Vereinfachtes Beispiel eines aggregierten Atom Feeds, basierend auf [NS05].

Der einzige Eintrag aus 2.2 wird unverändert in 2.3 übernommen, sodass insbesondere die Zeitstempel identisch sind. Der aggregierte Feed wird zum ersten Mal am 31.08.2005 veröffentlicht, dieser Wert ist im updated-Feld eingetragen, welches ein direktes Subelement des feed-Elements ist. Ein Konsument, der diesen Feed anfragt, kann nicht erkennen, dass der Feed und der Eintrag erst am 31.08.2005 zum ersten Mal verfügbar waren. Für ihn hat es den Anschein, dass der Eintrag bereits im Dezember 2003 erstellt und Ende Juli 2005 verändert wurde. Da das updated-Element des Feeds einen späteren Zeitpunkt als den des Eintrages enthält, könnte er intuitiv daraus schließen, dass sich die Metadaten wie der Titel des Feeds geändert haben könnten. Das Beispiel verdeutlicht, dass diese Schlussfolgerung falsch sein kann. Es ist ebenso möglich, dass zu dem Zeitpunkt ein Eintrag gelöscht wurde oder – wie im Beispiel – ein aggregierter Feed vorliegt, der zu diesem Zeitpunkt erstellt wurde.

Fallstricke bei der Verarbeitung von Feeds

Die Betrachtung der Grundlagen zu Feeds wird mit einer Zusammenfassung möglicher Fehlerquellen abgeschlossen, die bei der Verarbeitung der Feeds auftreten können.

Keine Ordnung der Einträge Wie in Abschnitt 2.1.2 bereits eingeführt, ist ein Feed im Allgemeinen nach dem FIFO-Prinzip aufgebaut. Folglich wird ein neuer Feed-Eintrag als erstes item bzw. entry-Element im XML-Dokument eingefügt wird. Alle bisherigen Einträge werden um eins nach unten verschoben und sollte die maximale Fenstergröße bereits zuvor erreicht worden sein, so wird der letzte Eintrag aus dem Dokument entfernt. Dieses Verhalten ist jedoch weder in RSS 2.0 noch in Atom 1.0 vorgeschrieben, sodass bei der Verarbeitung von Feeds weder eine Ordnung der Einträge erwartet werden darf noch dass Einträge nach dem FIFO-Prinzip in den Feed aufgenommen und aus ihm verdrängt werden.

Variable Fenstergröße In keinem der Formate wird die Fenstergröße genau spezifiziert, sodass weder eine Aussage über gebräuchliche oder maximale Fenstergrößen getroffen wird, noch dass die Fenstergröße konstant ist. Ein Feed verhält sich daher auch dann Standard konform, wenn die Fenstergröße mit jedem neu hinzukommenden Eintrag um eins wächst und somit die Einträge nie entfernt werden oder wenn die Fenstergröße komplett variabel ist und bei jeder Abfrage eine andere Anzahl von Einträgen zurückgegeben wird, z. B. alle Einträge des aktuellen Tages.

Unzuverlässige Datumsangaben Bei RSS 2.0 darf das pubDate in der Zukunft liegen, sodass anhand dieses Wertes nicht zwangsläufig auf das tatsächliche Publikationsdatum eines items geschlossen werden kann. In der Atom 1.0 Spezifikation steht diese Option zwar nicht zur Verfügung, entry-Elemente dürfen jedoch vom Anbieter auch nach ihrer Veröffentlichung noch verändert werden, wobei es dem Anbieter frei steht, den Wert des updated-Elements zu aktualisieren oder nicht. Daher ist es z. B. möglich, dass der Inhalt eines entry-Elements mehrfach aktualisiert wird, der Zeitstempel hingegen unverändert bleibt. Treten bei einem Feed Aktualisierungen dieser Art auf, so ist es potentiell schwer zu erkennen, ob man als Konsument alle Aktualisierungen aufgezeichnet hat. Zudem können Feeds aus anderen Feeds unverändert aggregiert werden, sodass z. B. der Wert des updated-Feldes des Quell-Feeds in den aggregierten Feed übernommen wird und nicht dem tatsächlichen Zeitstempel des Erscheinens entspricht.

In einer Analyse eines Datensatzes von 23.000 Feeds im Jahr 2004 wurde festgestellt, dass die im HTTP-Header enthaltenen Zeitstempel meist korrekt waren, da sie das Änderungsdatum der zugrunde liegenden Datei angeben [Vog04]. Nur in knapp einem Viertel der Feeds hatten die Einträge Zeitstempel, von denen mehr als fünf Prozent in der Zukunft lagen.

2.3.5. Spezifische Eigenschaften des Übertragungsprotokolls HTTP

HTTP 1.1 [FGM+99] stellt bereits seit mehr als einem Jahrzehnt im Header einige Metadaten bereit, die Informationen über die Aktualität und Änderungen des Inhalts

enthalten und z. B. in Form von bedingten GET-Abfragen (engl. conditional GET requests) verwendet werden können. Diese Eigenschaften stehen allen Dokumenten zur Verfügung, die per HTTP übertragen werden.

Date Das Date-Feld enthält das aktuelle Datum sowie die Systemzeit des Servers, zu der die HTTP-Antwort generiert wurde. Alle Antworten eines Servers müssen dieses Feld enthalten, Ausnahmen bilden interne Fehlermeldungen wie 500 (Internal Server Error) oder Server, die über keine angemessene Methode zur Bestimmung der Systemzeit verfügen. Das Datumsformat ist in RFC 1123 [Bra89] bzw. RFC 822 [Cro82] spezifiziert, z. B. „Date: Thu, 30 Apr 1987 12:00:00 CEST“. Dieses Format wird ebenso in allen folgenden Feldern, die ein Datum enthalten, vorgeschrieben. In dieser Arbeit wird das Date-Feld ausgewertet, um Zeitverschiebungen zu Servern zu berechnen, deren Systemzeit nicht mit Verfahren wie dem Network Time Protocol (NTP) [Mil92] mit einer zentralen Uhr synchronisiert sind.

Last-Modified Das Last-Modified-Feld ist optional und enthält das Datum und die Uhrzeit, die der Server als Zeitpunkt der letzten Änderung der Ressource annimmt. Bei statischen Dokumenten ist dies in der Regel der Zeitstempel aus dem Dateisystem, enthält das Dokument dynamische Anteile, so sollte der Zeitstempel mit dem des neuesten Inhaltes übereinstimmen – am Beispiel von Feeds ist dies in der Regel der Zeitstempel des neuesten Eintrags. Da die Ermittlung des Zeitstempels des neuesten Inhalts nicht immer möglich ist, wird in diesem Fall häufig die aktuelle Systemzeit verwendet. Generell sollte die Ermittlung des Last-Modified-Zeitstempels so nah wie möglich an der Ermittlung des Date-Zeitstempels liegen, um es Nutzern zu ermöglichen, die Zeitdifferenz auszuwerten. Last-Modified darf nicht in der Zukunft liegen, d. h. neuer sein als der Wert des Date-Feldes.

ETag Das optionale Entity-Tag-Feld enthält eine in Anführungszeichen gesetzte, beliebige Zeichenkette und kann zum Vergleich verschiedener Instanzen einer Ressource verwendet werden. Innerhalb einer Ressource ist es ein eindeutiger Identifikator, sodass alle Revisionen aller Instanzen einer Ressource entweder verschiedene oder gar keine ETags aufweisen. Die Einzigartigkeit des ETags kann mit sogenannten *schwachen* ETags eingeschränkt werden. Zwei Instanzen einer Ressource können das gleiche schwache ETag besitzen, wenn sie semantisch äquivalent sind. Schwache ETags sind durch den Präfix ”W/”gekennzeichnet. „ETag: W/"xyzzy"“ ist ein valides schwaches ETag. Ein präfixloses ETag wird auch als *starkes* ETag bezeichnet. Besitzen zwei Instanzen einer Ressource das gleiche starke ETag, so müssen sie eine oktale Gleichheit aufweisen.

Expires Das Expires-Feld enthält ein Datum, nach welchem die Nachricht (Response) als veraltet zu betrachten ist. Es muss jedoch beachtet werden, dass aus dem Vorhandensein des Expires-Feldes nicht geschlossen werden darf, dass die originale Ressource vor, nach oder zum Zeitpunkt des Datums geändert oder entfernt wird. Das Feld muss genau ein Datum enthalten, ungültige Werte wie die Zahl Null sind wie bereits in der Vergangenheit liegende Zeitstempel zu interpretieren, sodass die

Nachricht als verfallen zu betrachten ist. Soll eine Nachricht „niemals“ verfallen, kann der Urheber ein Datum in das Expires-Feld eintragen, das ca. ein Jahr in der Zukunft liegt, weiter entfernte Daten sollen nicht gesetzt werden. Enthält der Header neben dem Expires-Feld ein Cache-Control-Feld mit der max-age-Direktive, überschreibt diese den Wert des Expires-Feldes.

cache-control Das cache-control-Feld wird zur Steuerung und Übertragung von Statusinformationen von Web Caches eingesetzt und an geeigneter Stelle in Abschnitt 2.5.3 im Detail vorgestellt.

Mit der detaillierten Betrachtung der Feed-Formate und des zur Übertragung eingesetzten HTTP wurden wichtige Grundlagen vermittelt, um weitere Bereiche und Systeme auf Techniken hin zu untersuchen, mit denen eine Vorhersage der Zeitpunkte neuer Einträge von Feeds möglich ist. Hierzu erfolgen in den nächsten Abschnitten zunächst Betrachtungen verschiedener Systeme im WWW in den Bereichen Web Crawling (2.4) und Web Caching (2.5). Sonstige verwandte Ansätze (2.6) werden nur am Rande betrachtet. Anschließend wird der Fokus weiter eingegrenzt und Systeme zur Verarbeitung von Feeds analysiert (2.7). Die Reihenfolge der Betrachtungen ist an die zeitliche Entstehung der betrachteten Arbeiten angelehnt – vor allem Arbeiten im Umfeld von Feeds basieren auf denen der überlappenden Domänen Web Crawling und Web Caching, die zur übersichtlicheren Darstellung sequentiell betrachtet werden.

2.4. Web Crawling

Der bekannteste Einsatzbereich des Web Crawlings ist die Erstellung eines Index des WWW. Hierzu müssen u. a. bisher unbekannte Webseiten aufgefunden und in den Index aufgenommen sowie bereits enthaltene Webseiten regelmäßig aktualisiert werden. In dieser Arbeit sind ausschließlich die Techniken zum Aktualisieren von Interesse, dem Auffinden neuer Webseiten bzw. Feeds sowie der Indizierung selbst wird keine Aufmerksamkeit geschenkt. Die Optimierung der Aktualität eines Index wird seit mehr als einem Jahrzehnt untersucht und ist Gegenstand aktueller Arbeiten [DTV09]. Im Allgemeinen werden zunächst Annahmen über die Änderungsraten und das Verhalten von Webseiten getroffen, anschließend Parameter wie Alter, Frische oder zur Verfügung stehende Ressourcen definiert und diese in eine Optimierung des Gesamtsystems einbezogen. Eine gute, einführende Übersicht bietet [DTV09]. Neuere Arbeiten im Bereich *Deep Web Crawling* beschäftigen sich mit der Forschungsfrage, wie Daten aus dem Deep Web indiziert werden können, auf die ausschließlich über HTML-Formulare zugegriffen werden kann. Diese werden im Anschluss an die grundlegenden Arbeiten des Web Crawlings betrachtet.

2.4.1. Grundlegender Ablauf

Eine der grundlegenden Arbeiten des Bereichs stammt von den Autoren Cho und Garcia-Molina [CG00]. Die von ihnen beschriebenen Ansätze fanden vor allem in der Domäne Web Crawling weite Verbreitung und werden im Detail vorgestellt. Als Basis ihrer Ausführungen zeigen sie, dass die Änderungen von Webseiten mit einer Poisson-Verteilung

beschrieben werden können. In einem Poisson-Prozess treten Ereignisse zufällig, unabhängig und mit fixer Rate über die Zeit auf. Die Gleichung 2.1 wird einzig durch den Parameter λ bestimmt, der sowohl Mittelwert als auch Varianz der Verteilung ist [GKHK71] und dient der Ermittlung der Wahrscheinlichkeit P einer Poisson-verteilten Zufallsgröße, dass diese k mal eintritt.

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda} \tag{2.1}$$

Im Kontext der Aktualisierungen von Webseiten ist die Änderungsrate λ die in der Historie der Webseite beobachtete, durchschnittliche Änderungsrate. Angenommen es gilt $\lambda = 3/Tag$, so wird mit $P(X = k = 2)$ die Wahrscheinlichkeit ermittelt, dass an einem Tag exakt 2 Änderungen erfolgen. Es ergibt sich ein Wert von 0,22. Zur Vorhersage, mit welcher Wahrscheinlichkeit im Intervall λ mindestens eine Änderung eintritt, ergibt sich Gleichung 2.2 aus 2.1. Die Wahrscheinlichkeit, dass bei $\lambda = 3/Tag$ mindestens eine Änderung an einem Tag eintritt, beträgt 95 %. Zur Veranschaulichung sind in Abbildung 2.4 Poisson-Verteilungen für verschiedene Werte von λ dargestellt.

$$\begin{aligned} P(X = k > 0) &= 1 - P(X = k = 0) \\ &= 1 - \frac{\lambda^0}{0!} e^{-\lambda} \\ &= 1 - e^{-\lambda} \end{aligned} \tag{2.2}$$

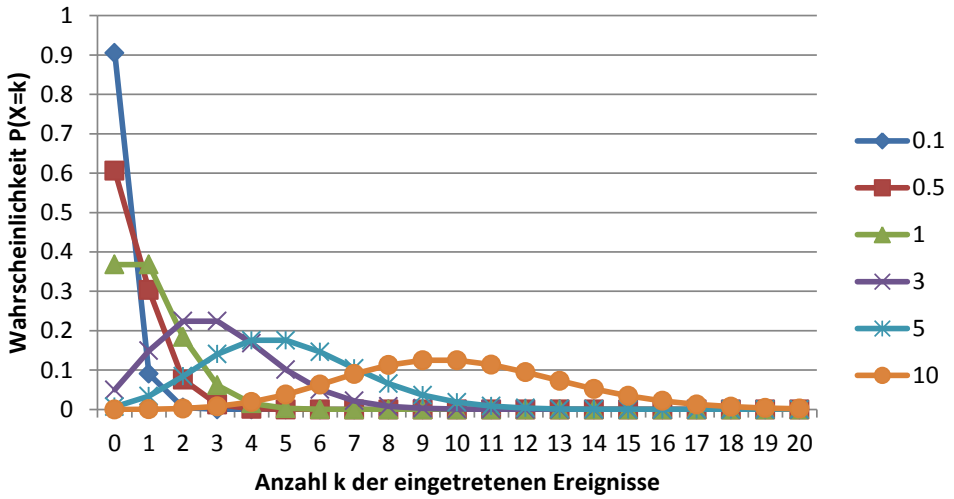

Abbildung 2.4.: Poisson-Verteilungen für verschiedene Werte von λ.

Zur Synchronisation des Index mit den Quellen schlagen die Autoren eine *Synchronization Policy* vor, die aus den vier Bestandteilen *Synchronization Frequency*, *Resource Allocation*, *Synchronization Order* und *Synchronization Points* zusammengesetzt ist.

Synchronization Frequency Die Aktualisierung des Index erfolgt zyklisch. In der Synchronization Frequency wird indirekt über die Periodendauer die Länge eines Zyklus spezifiziert. Es wird angenommen, dass in einem Durchgang immer die gleiche feste Anzahl von Quellen abgefragt wird.

Resource Allocation Die Resource Allocation gibt an, welche Elemente des Index in einem Zyklus synchronisiert werden. Hierfür schlagen die Autoren zwei Varianten vor. Bei der gleichmäßigen Verteilung der Ressourcen werden alle Elemente gleich häufig synchronisiert. Diese Methode wird in Kapitel 4 als *Fix* bezeichnet. Alternativ werden die Ressourcen proportional zur jeweiligen Änderungsrate λ eines Elements verteilt, sodass Webseiten mit kleinerem λ seltener aktualisiert werden als Webseiten mit einer großen Änderungsrate. Dieser Ansatz wird in Kapitel 4 als *Fix Learned* wieder aufgegriffen.

Synchronization Order Mit der Synchronization Order wird die Synchronisationsreihenfolge der einzelnen HTML-Seiten innerhalb einer Domain bestimmt. Diese ist für Feeds aufgrund der geringen Anzahl Feeds pro Domain nicht relevant.

Synchronization Points sind für die vorliegende Arbeit ebenfalls nicht relevant.

Zur Beurteilung der Aktualität des Index werden in [CG00] die zwei Metriken *Freshness F* und *Age A* eingeführt. Die Freshness ist das Verhältnis der Anzahl aktueller Elemente e_i zu allen N Elementen. Ein Index S ist zum Zeitpunkt t frischer als ein anderer Index, wenn er eine größere Anzahl aktueller Elemente enthält.

$$F(S;t) = \frac{1}{N} \sum_{i=1}^{N} F(e_i;t) \tag{2.3}$$

$$F(e_i;t) = \begin{cases} 1 & \text{wenn } e_i \text{ zum Zeitpunkt } t \text{ aktuell ist.} \\ 0 & \text{sonst.} \end{cases} \tag{2.4}$$

Hierbei gilt es einige getroffene Annahmen zu beachten. Die Entscheidung in Gleichung 2.4, ob eine Webseite aktuell oder veraltet ist, ist binär. In anderen Arbeiten wird hier feiner unterschieden [OP08]. Zudem werden alle Webseiten als gleich wichtig betrachtet.

Neben der Frische wird das Alter einer Seite betrachtet. Vergleicht man zwei Webseiten e_1 und e_2, für deren Freshness $F(e_1;t) = F(e_2;t) = 0$ gilt, so hat diejenige ein geringeres Alter, deren letzter Synchronisationszeitpunkt jünger ist. Die Berechnung erfolgt analog zu den Gleichungen 2.3 und 2.4, wobei t_{L,e_i} der Zeitpunkt der letzten Aktualisierung der Webseite e_i ist.

$$A(S;t) = \frac{1}{N} \sum_{i=1}^{N} A(e_i;t) \tag{2.5}$$

$$A(e_i;t) = \begin{cases} 0 & \text{wenn } e_i \text{ zum Zeitpunkt } t \text{ aktuell ist.} \\ t - t_{L,e_i} & \text{sonst.} \end{cases} \tag{2.6}$$

Der Zusammenhang zwischen Freshness und Age ist in Abbildung 2.5 über einen zeitlichen Verlauf für ein Element skizziert. Das alternierende Verhalten der Freshness sowie der lineare Anstieg des Alters sind eindeutig zu erkennen. Diese beiden Metriken lassen sich jedoch nicht eins zu eins auf das Ziel der Integration von Feeds übertragen, sodass in Kapitel 4 eigene Metriken aufgestellt und mit denen aus [CG00] verglichen werden. In [CG03a] erweitern die Autoren die beiden Metriken um individuelle Gewichte, um Webseiten entsprechend ihrer Bedeutung stärker bzw. schwächer in die Berechnung der Durchschnittswerte eingehen zu lassen.

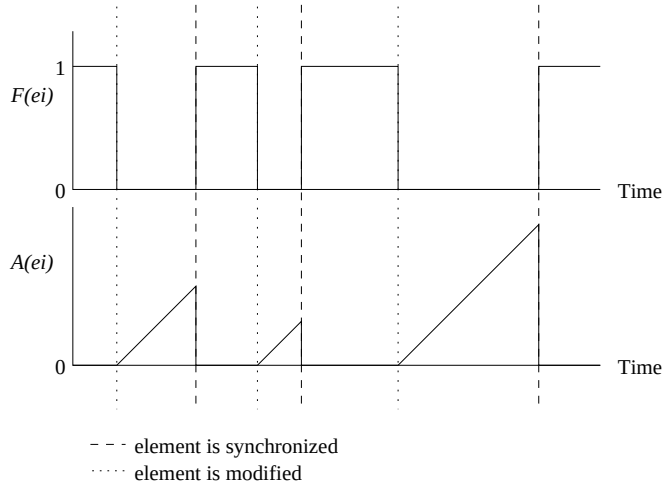

Abbildung 2.5.: Beispielhafter Verlauf von Freshness und Age einer Webseite [CG00].

In der Evaluation zeigen die Autoren zunächst, dass die Änderungsraten von Webseiten Poisson-verteilt sind. Hierzu verwenden sie einen Datensatz von 720.000 Webseiten, die gleichverteilt aus nur 270 Domains gewählt wurden. Diese wurden über einen Beobachtungszeitraum von vier Monaten jeweils täglich abgerufen. Aufgrund der in [CG00] im Detail beschriebenen Auswahl der Domains hat der Datensatz eine gewollte Verschiebung hin zu häufig aktualisierenden Webseiten. Die Autoren weisen explizit darauf hin, dass der Datensatz nicht erkennbare Lücken hat, wenn eine Webseite häufiger als einmal täglich aktualisiert wurde. Zudem schreiben sie, dass Webseiten existieren können, die nicht von einem Poisson-Prozess simuliert werden können, treffen jedoch keine Aussagen, wie deren Verhalten aussieht bzw. wie groß der Anteil solcher Webseiten ist.

Von besonderem Interesse ist das Evaluationsergebnis bezüglich der Resource Allocation. Wie bereits bekannt, hatten die Autoren zwei Verfahren vorgeschlagen, Webseiten entweder mit der gleichen Frequenz oder entsprechend ihrer beobachteten Änderungsrate zu aktualisieren. Basierend auf den in den Gleichungen 2.3 bis 2.6 dargelegten Metriken Freshness und Age sowie synthetischen Daten zeigen die Autoren in [CG99, CG00] formal, dass bei einer gleichmäßigen Verteilung der Ressourcen *immer* eine höhere Qualität des Index erreicht wird als wenn die Ressourcen entsprechend der Änderungsraten verteilt werden. Dieses Resultat ist überraschend und auf die verwendeten Metriken zurückzuführen. Im Gegensatz zu den im weiteren Verlauf der eigenen Arbeit eingeführten

Metriken, in der *jeder neue Eintrag* in einem Feed in die Berechnung eingeht, wird in [CG00] nur der *aktuelle Zustand* einer Webseite betrachtet. Stehen einem Crawler begrenzte Ressourcen zur Verfügung, um alle Elemente zu aktualisieren, dann wird die durchschnittliche Freshness des Index erhöht, wenn die verfügbaren Ressourcen verwendet werden, um sich selten ändernde Elemente zu überwachen. Dies führt jedoch dazu, dass sich häufig ändernde Webseiten nie aktualisiert werden.

Abschließend bleibt festzustellen, dass in [CG00] keine Aussage zur Ermittlung der Änderungsrate λ aus einer beobachteten Historie getroffen wird. Somit verbleibt ungeklärt, ob die Ermittlung von der Länge des Beobachtungszeitraums, der Anzahl Abfragen oder der Anzahl beobachteter Änderungen abhängig ist. Zudem berücksichtigen die Autoren keine Variationen der Änderungsrate über die Zeit.

Der grundlegende Ablauf des Web Crawlings ist jetzt bekannt, sodass im Anschluss zunächst einige Beobachtungen der Änderungsraten von Webseiten vermittelt werden. Darauf folgend werden Arbeiten vorgestellt, die sich mit der Ermittlung der Änderungsraten und einer hierauf aufbauenden Vorhersage zukünftiger Aktualisierungen beschäftigen.

2.4.2. Analyse der Änderungsraten von Webseiten

Änderungsraten von Webseiten werden in zahlreichen Arbeiten kurz betrachtet, wobei meist keine repräsentativen Datensätze verwendet werden oder die Analysen knapp sind. Viele Arbeiten beziehen sich auf die in [CG00] vermittelten Grundlagen, dass Änderungsraten im wesentlichen Poisson-verteilt sind, es jedoch Ausnahmen gibt. Andere weichen von dieser Annahme ab und untersuchen Änderungsraten, die im Verlauf der Zeit variieren [Mat05, Sin07] oder saisonale Muster aufweisen [GO08]. Analysen häufig geänderter Webseiten ergaben, dass nur ein kleiner Anteil sehr stark vom Poisson-Modell abweicht [GO08]. Von den häufig aktualisierten Webseiten werden viele täglich oder stündlich aktualisiert, zudem sind Unterschiede zwischen Tag und Nacht sowie Werktagen und Wochenenden festzustellen. Konkrete Hinweise zur Vorhersage zukünftiger Änderungen werden jedoch nicht getroffen [GO08]. Die Untersuchungen in [Mat05, Sin07] werden im nachfolgenden Abschnitt 2.4.3 betrachtet.

Eine sehr detaillierte Analyse wird in [FMNW03] anhand eines umfangreichen Datensatzes durchgeführt. Der Fokus der Autoren liegt auf Fragen, die aus Sicht des Web Crawling interessant sind: Wie häufig ändern sich Webseiten? Welcher Anteil des Inhalts einer Seite ändert sich? Kommen nur neue Inhalte hinzu, oder werden auch bestehende geändert? Korrelieren die Änderungen an der Seite mit anderen Eigenschaften der Seite?

Zur Beantwortung dieser Fragen sammelten die Autoren zunächst rund 150 Millionen URLs von Webseiten, die sie anschließend über einen Zeitraum von 11 Wochen wöchentlich abfragten. Anschließend untersuchten sie u. a. Abhängigkeiten zwischen den Dateigrößen, Änderungsraten und Top-Level Domains (TLDs). Dokumente waren im Median 16 Kilobyte (kB) groß, einzig .edu-Webseiten aus dem amerikanischen Bildungsbereich hatten im Median nur eine Größe von 8 kB.

Zur Darstellung der Verteilung von Änderungen führen die Autoren sechs Klassen auf, denen ein Dokument in Abhängigkeit von der anteiligen Änderung seines Inhalts zugeordnet wird. Diese Klassen enthalten verschiedene Abstufungen zwischen „vollständig geän-

dert“ und „keine Änderung“ und werden im weiteren Verlauf der Arbeit in Abschnitt 3.7.5 aufgegriffen. Insgesamt änderten sich rund zwei Drittel der Seiten innerhalb von 11 Wochen gar nicht und nur 3 % wurden sehr stark oder vollständig modifiziert. Die Verteilung der Änderungen variierte sehr stark zwischen den TLDs: Während nur rund die Hälfte der .de-Domains unverändert blieb und ca. ein Viertel stark oder komplett verändert wurden, wiesen rund 90 % der .edu keine Änderungen und nur ca. 1 % deutliche Änderungen auf. Dieser Unterschied ist beachtlich. Einen weiteren Zusammenhang ermittelten die Autoren zwischen der Dateigröße und der Wahrscheinlichkeit einer Änderung: um so größer ein HTML Dokument ist, desto höher ist die Wahrscheinlichkeit, dass es verändert wird.

Das Resümee der Autoren ist, dass in der Historie beobachtete Änderungen an Webseiten sich gut für die Vorhersage von zukünftigen Änderungen eignen. Ändern sich in Woche n zehn Prozent des Inhalts einer Seite, so ist auch in der Woche $n + 1$ eine zehn prozentige Änderung des Inhalts zu erwarten. Im gleichen Zug stellen die Autoren jedoch fest, dass diese Aussage für eine große Anzahl Seiten *nicht* zutrifft und bei zwei aufeinander folgenden Wochen in einer der beiden gar keine Änderung auftritt. Für die Betrachtungen der eigenen Arbeit wird geschlussfolgert, dass Webseiten im Jahr 2003 recht kontinuierliche Änderungsraten aufwiesen, wenn man diese auf einer wöchentlichen Basis betrachtet. Aussagen zu verpassten Änderungen und den (vermuteten) tatsächlichen Änderungsraten werden von den Autoren leider nicht getroffen.

2.4.3. Ermittlung von Änderungsraten und Vorhersage von Aktualisierungen

Im Folgenden werden ausgewählte Arbeiten vorgestellt, in denen wichtige Konzepte zur Ermittlung der Änderungsraten von Webseiten sowie darauf aufbauende Vorhersagen zukünftiger Aktualisierungen betrachtet werden.

Ermittlung der Änderungsrate auf Basis einer lückenhaften Historie

Die in Abschnitt 2.4.1 in [CG00] aufgezeigten Lücken wurden einige Jahre später in [CG03b] betrachtet. Im Unterschied zur eigenen Arbeit, in der auf Basis der Änderungsrate eine kommende Aktualisierung vorhergesagt wird, liegt der Fokus in [CG03b] zunächst in der Bestimmung der Änderungsrate von Objekten, zu denen keine vollständige Historie vorliegt. Im Bezug auf Feeds liegt eine lückenhafte Historie vor, wenn der Feed eine Fenstergröße von einem Eintrag hat oder zwischen zwei Abfragen Einträge verpasst wurden. Eine detaillierte Betrachtung wird in Abschnitt 3.5.2 nachgereicht.

Naiver Ansatz Wie bereits in vorangegangenen Arbeiten [CG99, CG00, CG03a] nehmen die Autoren an, dass die Poisson-Verteilung die Verteilung der realen Änderungen hinreichend gut approximiert. Ihr Ziel ist, die Änderungsrate λ (siehe Gleichung 2.1) aus dem wiederholten Zugriff auf ein Objekt zu bestimmen. Sie unterscheiden zunächst, ob a) eine lückenlose Historie ermittelt werden kann und b) Datumsangaben vorliegen. Im Falle von a) ist die Ermittlung trivial, die durchschnittliche Änderungsrate entspricht dem Quotient aus der Anzahl der ermittelten Änderungen der Quelle und der Länge des Beobachtungsintervalls. Der von den Autoren als naiv bezeichnete Ansatz wird in dieser

Arbeit ab Kapitel 4 als *FixLearned* bezeichnet und in den Evaluationen zum Vergleich betrachtet. Cho und Garcia-Molina grenzen sich unter Berufung auf [TK98, Win72] von der Betrachtung vollständiger Historien ab, da der naive Ansatz bereits eine gute Näherung sei. Liefert eine Quelle wie eine Webseite hingegen nur den aktuellen Zustand, so kann bei einer Änderung nicht ermittelt werden, ob weitere Modifikationen zwischen der letzten und der aktuellen Abfrage erfolgt sind. In diesem Fall unterscheiden sie in b), ob das Datum der letzten Änderung aus der Quelle extrahiert werden kann oder ob kein Datum angegeben ist, sodass eine Signatur der Quelle errechnet und mit letzten bekannten Signatur verglichen werden muss, um eine Änderung zu bemerken. Ist kein Datum angegeben, so wird der Änderung stattdessen der Zeitpunkt des Zugriffs zugewiesen. Im Gegensatz zur eigenen Arbeit, in der in Abschnitt 3.5.5 eine Vorverarbeitung fehlerhafter Datumsangaben vorgeschlagen und umgesetzt wird, treffen die Autoren keine Aussage zu Fehlern der realen Daten, die bei einigen Methoden zwangsläufig zu Fehlern führen.

Zur Approximation der tatsächlichen Änderungsrate schlagen die Autoren neben dem naiven Ansatz zwei Alternativen vor. Der naive Ansatz hat den Nachteil, dass er um so schlechter ist, je mehr Einträge verpasst werden: Erfolgen in einem Zeitraum von 20 Stunden drei Änderungen, von denen nur zwei ermittelt werden, ermittelt der naive Ansatz eine Änderungsrate von 2 Modifikationen/20 Stunden. Dieser Nachteil wird mit zwei Schätzfunktionen minimiert.

Abfrage in gleichmäßigen Abständen Gleichung 2.7 zeigt die als *improved estimator* bezeichnete Schätzfunktion der Änderungsrate λ_q einer Quelle q. X_q ist die Anzahl gefundener Änderungen, n_q die Anzahl der Abfragen und f_q die Frequenz, mit der die Quelle abgefragt wurde. Die Konstanten von 0,5 wurden eingeführt, da der Logarithmus von Null nicht definiert ist, sodass die Gleichung andernfalls kein gültiges Ergebnis liefert, wenn bei jeder Abfrage eine Änderung festgestellt wird. Die Autoren zeigen, dass die mit Gleichung 2.7 bestimmte Änderungsrate um so genauer ist, je größer n_q ist.

$$\lambda_q = -log\left(\frac{n_q - X_q + 0{,}5}{n_q + 0{,}5}\right) f_q \tag{2.7}$$

Neben den zahlreichen formalen Betrachtungen der Schätzfunktionen erfolgen Evaluierungen auf realen sowie synthetischen Daten. Die Autoren simulieren die Aktualisierungen mit Hilfe einer Gamma-Verteilung als Verallgemeinerung der Poisson-Verteilung, schreiben jedoch, dass sich nicht alle realen Webseiten so verhalten. Der reale Datensatz ist eine Teilmenge des in [CG00] erstellten Datensatzes, bestehend aus rund 720.000 Webseiten, die gleichmäßig über 270 Domains verteilt sind. Diese wurden für eine Zeitdauer von vier Monaten ein Mal täglich abgefragt. Die Kritikpunkte an dem Datensatz sind der Auswahlprozess der Domains und die nachträgliche Filterung. Der Korpus entstand nicht durch eine zufällige Auswahl der Webseiten, stattdessen wurden die 400 populärsten Domains aus einem Repository von mehr als 25 Millionen Webseiten selektiert. Nach der Aufzeichnung der Historie wurden alle Webseiten aussortiert, die mehr als eine Änderung in drei Tagen oder weniger als drei Änderungen im Beobachtungszeitraum hatten. Dies stellt für die eigene Arbeit unzulässige Einschränkungen dar, welche in einer

Verzerrung der Ergebnisse resultiert. Die Autoren räumen diese Verzerrung ein, meinen jedoch mit anderen Schranken ähnliche Ergebnisse erzielt zu haben. Die Frage nach der Quantifizierung von „ähnlich“ bleibt unbeantwortet.

Die Evaluation eines simulierten Crawlers zeigt, dass mit dem naiven Ansatz rund doppelt so viele Aktualisierungen gefunden werden als mit einem statischen Abfrageintervall von einer Woche. Die verbesserte Schätzfunktion aus Gleichung 2.7 bringt einen weiteren Gewinn von rund 35 % gegenüber dem naiven Ansatz. Bei allen drei Methoden wurden alle Webseiten im globalen Mittel einmal die Woche abgefragt, bei der naiven und der verbesserten Methode erfolgte eine Verschiebung entsprechend der einzelnen Änderungsraten. Die Änderungsraten wurden nach den ersten fünf Abfragen auf Basis der bis dahin beobachteten Daten ermittelt und anschließend nicht mehr angepasst. Wenngleich das Resultat gut ist, so fehlt dennoch die sehr wichtige Angabe des Optimums, d. h. der Anzahl Aktualisierungen, die ein perfekter Algorithmus erreicht hätte. In Kapitel 4 wird die Eignung statischer Abfrageintervalle als Heuristik empirisch untersucht.

In Gleichung 2.7 wurde vorausgesetzt, dass q immer in gleichen Abständen f_q abgefragt wird. Da dies in realen Szenarien nicht generell umsetzbar ist, schlagen die Autoren einen Maximum Likelihood Estimator (MLE) vor, mit dem die Länge aller Intervalle (sowohl mit, als auch ohne entdeckten Änderungen) berücksichtigt und entsprechend gewichtet wird. Der Ansatz wird in [CG03b] jedoch nicht vertieft.

Die in [CG03b] vorgestellte Heuristik zur Ermittlung der Änderungsrate anhand einer lückenhaften Historie basiert auf statischen Intervallen, mit dem alle Quellen gleichermaßen abgefragt werden. Diese werden in Abschnitt 4.2.2 analysiert.

Ermittlung des Änderungszeitpunktes einer Webseite

Das Problem der Ermittlung des Änderungszeitpunktes einer Webseite wurde in [CG03b] pragmatisch gelöst und in [Sin07] vertieft. Es wird unterschieden, ob der Zeitpunkt der letzten Änderung anhand des Last-Modified-Feldes des HTTP-Headers (siehe Abschnitt 2.3.5) ermittelbar ist oder nicht. Ist kein Datum angegeben, so muss es geschätzt werden. Der in [CG03b] gewählte pragmatische Ansatz ist, den aktuellen Zeitpunkt des Zugriffs auf die Quelle zu verwenden. Unter der Voraussetzung, dass die Quelle periodisch und mit einem optimalen Intervall auf Änderungen überprüft wird, schlagen die Autoren in [Sin07] vor, den mittleren Zeitpunkt zwischen zwei sequentiellen Zugriffen zu verwenden. Alternativ kann auch eine geeignete Verteilungsfunktion als Heuristik eingesetzt werden. Leider nennen die Autoren keine weiteren Details zu den genannten Voraussetzungen und verzichten auf eine Evaluation der Ansätze.

Ermittlung der Änderungsrate durch sampling

Ein weiterer Ansatz zur Ermittlung der Änderungen von Webseiten wird in [CN02] vorgestellt. Grundannahme der Arbeit ist, dass es pro Domain eine Vielzahl von Webseiten gibt. Die Autoren führen zunächst ein *sampling* durch, wobei sie von jeder Domain einen Bruchteil der Seiten herunterladen und auf Aktualisierungen überprüfen und auf dieser Basis dann entscheiden, welche Domains in einem zweiten Durchlauf komplett

überprüft werden. Der Ansatz ist derzeit[4] nicht auf Feeds übertragbar, da das Verhältnis von Domains zu Feeds und Domains zu einzelnen HTML-Seiten ungleich ist und es vergleichsweise durchschnittlich nur sehr wenige Feeds pro Domain gibt.

Modellierung der Änderungsrate als inhomogener Poisson-Prozess

Neben den zahlreichen Arbeiten, die Änderungsraten als homogenen Poisson-Prozess modellieren, existiert ebenso ein Vielzahl von Arbeiten wie [GE01, Mat05, BGR06, Sin07] zur Modellierung als *in*homogener Poisson-Prozess, welcher in [GE01] als flexibler beschrieben wird. In [Mat05] werden einige Möglichkeiten formal eruiert, Änderungsraten von Quellen wie Webseiten zu schätzen, zu denen nur eine lückenhafte Historie vorliegt. Diese sind für die eigene Arbeit von geringem Interesse, da bei Feeds nur selten Lücken in der Historie der Einträge vorkommen, wie in Abschnitt 3.7.8 empirisch belegt wird. Die auf dem Maximum Likelihood Estimator basierenden Schätzfuktionen haben die Voraussetzungen, dass die Quellen in gleichmäßigen Abständen abgefragt werden und eine möglichst große Menge an Trainingsdaten vorliegt. Diese beiden Voraussetzungen widersprechen den Zielen der eigenen Arbeit, in der Feeds ohne Trainingsphase und mit unterschiedlichen Intervallen abgefragt werden sollen.

In [Sin07] werden Änderungsraten als inhomogener Poisson-Prozess beschrieben, um (über die Zeit) *variierende* Änderungsraten zu berücksichtigen. Hierfür werden die Änderungshistorien zunächst in Fenster aufgesplittet, sodass die Änderungsrate innerhalb eines Fensters entweder stetig steigt, sinkt oder gleich ist. Zur Vorhersage werden vier Methoden miteinander verglichen. Neben der aus [CG03b] bereits bekannten Berechnung aus Gleichung 2.7 werden Schätzfunktionen auf Basis der Weibull-Verteilung und Duane Plots vorgestellt sowie die Heuristik aus [Mat05] eingesetzt.

Die zahlreichen auf dem inhomogenen Poisson-Modell beruhenden Algorithmen werden fortan unter dem Begriff PostRate-Algorithmen zusammengefasst. Zwei Vertreter der Klasse werden in den Abschnitten 2.5.4 und 2.7.3 im Detail vorgestellt.

Feingranulare Betrachtung von Versionsunterschieden

Neben den Arbeiten, in denen die Frische einer Webseite binär betrachtet wird, gibt es einige Ansätze, Änderungen feingranular zu betrachten und unterschiedlich zu gewichten [OP08, ATDE09]. In [OP08] werden Webseiten in verschiedene *content regions* unterteilt und die Divergenz zweier Versionen einer Webseite anhand dieser Fragmente ermittelt. Hierbei stellten die Autoren zwei typische Ausprägungen von Änderungen fest: Bestehende Inhalte werden entweder durch neue ersetzt oder nach unten verschoben, wenn neue Inhalte hinzukommen. Werden diese Beobachtungen auf Feeds übertragen, so ist zu erwarten, dass beinahe ausschließlich der zweite Fall vorliegt und ältere Einträge von neueren verdrängt werden (FIFO-Verhalten). Die zur Vorhersage neuer Änderungen vorgeschlagenen Algorithmen setzen voraus, dass sich das Verhalten von Webseiten

[4] Das Verhältnis könnte sich in der Zukunft verschieben, sodass ein Großteil der Webseiten über einen separaten Feed verfügt. In diesem Fall könnte ein sampling in Betracht gezogen werden, aktuell erscheint es jedoch nicht aussichtsreich.

nicht ändert. Die Autoren konzentrieren sich auf die Berücksichtigung feingranularer Modifikationen der Seiten. Ihre Evaluation zeigt, dass die Algorithmen eine Trainingsphase von einem Monat benötigen, in dem sie kaum bessere Ergebnisse erzielen als ein universeller Algorithmus, der jede Webseite in gleichen Intervallen abfragt. Die Idee, Änderungen anhand der inhaltlichen Relevanz zu gewichten, wäre sicher eine interessante Erweiterung der eigenen Arbeit, wird aber nicht betrachtet.

Vorhersage von Änderungen ohne Trainingsphase des Modells

Im Gegensatz zu zahlreichen anderen Arbeiten, in denen die Algorithmen zur Vorhersage die durchschnittliche Änderungsrate in einer Trainingsphase ermitteln, betrachten die Autoren in [FGT08] unter anderem den Fall, dass neue Webseiten dem Korpus jederzeit hinzugefügt werden können und bereits nach einem einzigen Crawling-Vorgang eine Vorhersage für den Zeitpunkt der nächsten Änderung der Quelle getroffen werden muss. Da in diesem Fall eine Historie mit nur einem einzigen Eintrag vorliegt, nehmen sie zwei virtuelle Abfragen an, wobei nach einer Stunde eine Änderung gefunden wurde und bei der zweiten Abfrage nach 57 Stunden keine Änderung vorlag. Diese Werte ermittelten sie empirisch aus einem Datensatz von 10.000 Webseiten [GF08]. Darüber hinaus optimieren sie das von einem Crawler zu verwendende Abfrageintervall anhand zweier fixer Kostenfaktoren: den Kosten für einen Crawl C_C und den entstehenden Kosten C_S, wenn eine Seite für eine Zeiteinheit wie eine Stunde *stale* (nicht aktuell) ist sowie der bekannten Änderungsrate der Seite.

Die Autoren treffen keine Aussage zur maximalen Länge der betrachteten Historie. Bei den hypothetischen Betrachtungen mit verschiedenen Werten für diese Kosten bleibt unklar, wie diese in realen Umgebungen berechnet werden und wie das Verhältnis C_C zu C_S ist. Die Autoren stellen z. B. fest, dass wenn $C_C = C_S$ gilt und sich die Quelle öfter als einmal pro Stunde ändert, dann ist es am günstigsten, diese Seite gar nicht im Index zu aktualisieren. Für die Aktualisierung eines Index scheint dies geeignet, die Ziele der Autoren weichen jedoch von denen der eigenen Arbeit ab: Wie bereits eingeführt, liegt bei der Integration von Informationen aus Feeds keine binäre Entscheidung der Aktualität vor, sodass die Kosten mit zunehmender Anzahl verpasster Einträge über die Zeit immer schneller ansteigen.

2.4.4. Deep Web Crawling

Zahlreiche Arbeiten beschäftigen sich mit der Indizierung des Deep Web [MCD+07, MKK+08, MAAH09, YWLW10]. Unter dem Begriff werden Daten zusammengefasst, die nicht direkt auf statischen Webseiten zur Verfügung stehen, sondern nur über HTML-Formulare abgefragt werden können. Zur Indizierung dieser Daten stehen zwei Ansätze zur Verfügung. Die erste Lösung sind vertikale, domänenspezifische Suchmaschinen. Sie basieren auf Mediatoren pro Webseite sowie Mappings zwischen den Konzepten der individuellen Datenquellen [MKK+08]. Dem gegenüber stehen eine Reihe von Verfahren, die häufig als *surfacing* bezeichnet werden [MCD+07, AAB+09]. Hierzu werden den Formularen eine Reihe generierter Anfragen übergeben und die zurückgegebenen

Ergebnisseiten mit den bereits zur Verfügung stehenden Techniken des Web Crawlings in den Index aufgenommen [MKK+08]. Dies hat den Vorteil, dass bereits vorhandene Techniken – z. B. zum Aktualisieren des Index – wiederverwendet werden können. Die Forschungsfragen in der Domäne Deep Web Crawling fokussieren u. a. auf Strategien, geeignete Abfragen zu generieren, um die Anzahl der benötigten Zugriffe auf Formulare zu minimieren und gleichzeitig den Recall zu maximieren. Der Recall ist hier das Verhältnis der auf den Antwortseiten zurückgegebenen Daten zu allen Daten, die in der Datenbank des Betreibers vorgehalten werden. Zu den weiteren Forschungsfragen zählt die Semantik der Daten [AAB+09], die ein menschlicher Nutzer bereits aus den Formularen und der Präsentation der Daten auf den Antwortseiten ableiten kann, die jedoch mit herkömmlichen Techniken der Indizierung verloren geht. Diese Fragen und Themengebiete sind für die eigene Arbeit nicht relevant.

Die Autoren in [YWLW10] verfolgen das Ziel, die Aktualität aus dem Deep Web integrierter Daten zu optimieren. Sie erweitern die binäre Betrachtungsweise der Freshness, um auch zwischenzeitlich verpasste Aktualisierungen zu berücksichtigen. Ihre in Gleichung 2.8 dargestellte Berechnung der Freshness ähnelt der aus Gleichung 2.6 bekannten Berechnung des Alters einer Quelle [CG00]. Der entscheidende Unterschied ist die Summierung der Zeiten aller M Aktualisierungen des Elements e, die noch nicht in der lokalen Datenbasis L eingepflegt sind. Zur Berechnung der durchschnittlichen Freshness von L führen die Autoren zudem für jedes Element ein Gewicht $w(e_i)$ ein. Der Verlauf der Freshness ist in Abbildung 2.6 veranschaulicht. Nach der Aktualisierung zum Zeitpunkt 5 ist deutlich zu erkennen, dass die Freshness mit jeder weiteren Aktualisierung steiler ansteigt. Die von den Autoren gewählte Bezeichnung Freshness ist hier leider missverständlich, da das Alter bisher nicht synchronisierter Änderungen aufsummiert wird. Von der Bezeichnung abgesehen, ist die Metrik aus Sicht der eigenen Arbeit sehr interessant. In Kapitel 4 wird die gleiche Metrik unter dem Namen *Delay* aufgegriffen. [YWLW10] und die eigene Arbeit entstanden zeitlich nahezu parallel, sodass sich diese Überschneidungen ergaben.

$$F(e;t) = \begin{cases} 0 & \text{wenn } e \text{ zum Zeitpunkt } t \text{ aktuell ist.} \\ \sum_{j=1}^{M} t - t_j & \text{sonst.} \end{cases} \tag{2.8}$$

$$F(L;t) = \frac{1}{N} \sum_{i=1}^{N} w(e_i)\ F(e_i;t) \tag{2.9}$$

Der Kern der in [YWLW10] vorgestellten Methoden besteht in sogenannten *keyword probe-queries*, um pro Abfrage eines HTML-Formulars möglichst viele Daten zurückgegeben zu bekommen. Die vorgestellte Optimierung der Freshness ihrer lokalen Datenbasis baut auf diesen Methoden auf. Ihr Grundsatz ist, dass die Priorität eines Elements um so höher ist, je länger die letzte Aktualisierung zurück liegt, je häufiger sich das Element ändert und je mehr Nutzer an dem Element interessiert sind. Diese Grundsätze decken sich mit den Zielen der eigenen Arbeit. In einer Evaluation zeigen die Autoren schließlich,

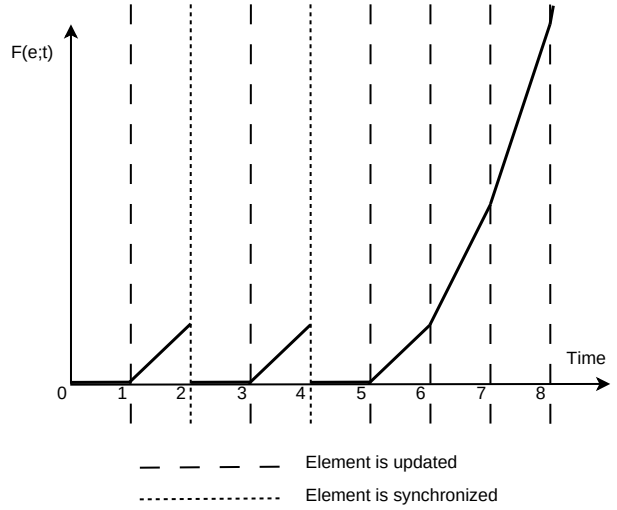

Abbildung 2.6.: Beispielhafter Verlauf der Freshness einer Quelle [YWLW10].

dass ihr Ansatz der naiven Methode überlegen ist und eine deutlich bessere Freshness erreicht wird, da pro Abfrage mehrere Elemente synchronisiert werden können. Dieser Ansatz ist nicht auf Feeds übertragbar, da hier keine solchen Abfragen gestellt werden können und immer nur ein Feed pro Abfrage synchronisiert werden kann.

2.4.5. Zusammenfassung Web Crawling

Den in Abschnitt 2.4 betrachteten Arbeiten können einige Anleihen entnommen werden, sie betrachten das Problem der Ermittlung der Änderungsrate einer Quelle und die Vorhersage neuer Versionen jedoch aus einem anderen Blickwinkel. Web Crawling und der eigene Ansatz haben zwei bedeutende Unterschiede:

Änderungsrate Bei Webseiten muss die Änderungsrate anhand einer lückenhaften Historie ermittelt werden, bei Feeds steht in der Regel bei jedem Zugriff eine begrenzte Änderungshistorie zur Verfügung, die auf den Zeitstempeln der Einträge basiert und deren Länge der Fenstergröße entspricht.

Aktualität Bei der Indizierung von Webseiten ist die Entscheidung, ob eine Webseite im Index aktuell oder stale ist, binär. Ist der Eintrag im Index nicht mehr aktuell, so spielt es keine Rolle, wie viele Modifikationen seit der letzten Synchronisation erfolgt sind. Bei der Integration von Feeds in ein PIS ist diese Entscheidung hingegen nicht binär. Jeder verpasste Eintrag kann wichtige Informationen enthalten, sodass sowohl die Anzahl verpasster Einträge, als auch die Zeit zwischen der Veröffentlichung eines Eintrags und der Abfrage des Feeds beachtet werden müssen.

Fokus der Domäne Deep Web Crawling sind die als surfacing bezeichneten Methoden, Daten zu erfassen, die nur über HTML-Formulare abrufbar sind. Die Indizierung der

Ergebnisseiten erfolgt mit bereits zur Verfügung stehenden Techniken des Web Crawlings, zu denen Strategien zur Ermittlung der Änderungsrate zählen.

Den betrachteten Arbeiten können dennoch einige wichtige Ansätze und Metriken entnommen und in den Kapiteln 4 und 5 (modifiziert) wiederverwendet werden. Hierzu zählen u. a. Metriken wie Age und Freshness [CG00, CG03a, CG03b, YWLW10], die Ermittlung des Änderungszeitpunktes, wenn die Quelle diesen nicht bereitstellt [CG03b, Sin07], variable Änderungsraten [Sin07], der Verzicht auf eine Trainingsphase durch Einsatz von Heuristiken [FGT08] und nicht zuletzt detaillierte Analysen eines realen Datensatzes, um ein besseres „Gefühl“ für die Daten zu erhalten [FMNW03].

2.5. Web Caching

Ziel des Web Cachings ist es, mit Hilfe von *Caches* Nutzeranfragen schneller zu beantworten, die Anzahl der Abfragen und somit die entstehende Last auf den Datenquellen zu minimieren und gleichermaßen den Kommunikationskanal zu entlasten. Caches werden häufig als *Proxy Server* bezeichnet, da sie in der Topologie zwischen dem Abfragenden und der angefragten Datenquelle platziert werden. Das Thema Web Caching wird bereits seit mehr als zwei Jahrzehnten untersucht. Die an einer über diesen Abschnitt hinausgehenden, vertiefenden Darstellung interessierten Leser seien auf [RS02] verwiesen.

Im Folgenden wird die Domäne Web Caching aus zwei Blickwinkeln beleuchtet. Einerseits werden geeignete Techniken untersucht, denen Anleihen zur Vorhersage von Aktualisierungen entnommen werden können. Andererseits werden Feeds über das Internet übertragen, sodass Caching-Mechanismen unter Umständen negative Auswirkungen auf die Aktualität der abgefragten Feeds haben können. In Abschnitt 2.5.1 werden zunächst einige Basiskonzepte anhand eines Beispiels eingeführt. Aus diesen lassen sich einige Fragen ableiten, die in den nächsten Abschnitten untersucht werden. Die wichtigste Frage und somit der Schwerpunkt der Betrachtungen ist die der Cache-Konsistenz: kann sichergestellt werden, dass ein Cache keine veralteten Objekte enthält – und wenn ja, wie? Hierzu werden in den Abschnitten 2.5.2 und 2.5.3 zahlreiche Ansätze vorgestellt. In Abschnitt 2.5.5 wird das aus der Domäne Datenbanken bereits bekannte Problem der Replikation am Beispiel von Content Distribution Networks (CDNs) kurz aufgegriffen. Um Objekte aus einem vollen Cache zu verdrängen, werden in Abschnitt 2.5.6 Ersetzungsstrategien umrissen. Ein weiterer interessanter Aspekt ist *prefetching*, das in Abschnitt 2.5.7 vorgestellt wird. Mit diesem Begriff werden Methoden bezeichnet, mit denen Vorhersagen getroffen werden, welche Objekte potentiell als nächstes von einem Nutzer angefordert werden, um diese proaktiv von der Quelle zu laden, sofern sie noch nicht im Cache enthalten sind.

2.5.1. Basiskonzepte

Web Caching ist meist hierarchisch aufgebaut und erfolgt in der Regel transparent für den Nutzer. Ein einfaches Beispiel ist ein lokaler Cache im Browser des Nutzers und ein weiterer Cache beim Internet Provider des Nutzers. Initial sind alle Caches leer. Die erste Abfrage des Nutzers wird zunächst gegen den Cache im Browser gestellt. Da dieser

die Abfrage nicht beantworten kann, leitet er sie an den Cache des Providers weiter. Man spricht von einem *cache miss*. Der Cache des Providers leitet die Abfrage an die vom Nutzer angefragte Quelle weiter, speichert das in der Antwort enthaltene Objekt und leitet es an den Abfragenden weiter, der analog agiert. Stellt der Nutzer nun die selbe Anfrage erneut, so kann sie bereits von seinem lokalen Cache beantwortet werden, man spricht von einem *cache hit*. Wurde das aus dem Cache zurückgegebene Objekt zwischenzeitlich auf dem Quellsystem verändert, so wird die in den Caches vorhandene Kopie als *stale*, d. h. veraltet bezeichnet. Je nach eingesetztem Konsistenzmechanismus ist dies akzeptabel, eine ausführliche Betrachtung erfolgt im anschließenden Abschnitt [RS02]. In dem Beispiel wird deutlich, dass ein Cache sowohl die Rolle des Servers als auch des Clients einnehmen kann, je nachdem, ob er eine eingehende Anfrage beantwortet oder an einen weiteren Knoten im Netzwerk weiterleitet.

2.5.2. Typen von Cache-Konsistenzen

Die Konsistenz eines Caches ist der Schwerpunkt der Betrachtungen des Web Cachings. Hierfür werden zunächst vier Typen von Cache-Konsistenzen eingeführt: *strong*, *delta*, *weak* und *mutal* [INST04].

strong Es wird sichergestellt, dass ein Objekt auf dem Server nicht neuer ist als in jedem Cache. Dieser Typ ist primär akademischer Natur, da er nur im kleinen Rahmen, jedoch nicht in Szenarien wie dem Web umsetzbar ist.

delta Mit dem Parameter delta wird eine Zeitspanne angegeben. Die Version des im Cache vorgehaltenen Objekts darf maximal delta Sekunden älter sein als die aktuelle Version auf dem Server. Ein verwandter Ansatz wurde in der Domäne Datenbanken in Abschnitt 2.2.3 als verzögerte Aktualisierung eingeführt.

weak Bei diesem Typ ist es zulässig, dass Objekte im Cache stale sind, bereits mehrere Aktualisierungen der Quelle erfolgten und somit veraltete Versionen als Antwort zurückgegeben werden. [INST04] zufolge war dies vor einigen Jahren der verbreitetste in Caches im Internet eingesetzte Typ.

mutal Eine *mutal* Konsistenz garantiert, dass eine Gruppe von Objekten das gleiche Alter hat.

Zur Realisierung der Cache-Konsistenz existieren mehrere Ansätze, die im Folgenden vorgestellt werden. Unabhängig von diesen Ansätzen kann aus den vier betrachteten Typen bereits geschlossen werden, dass beim (naiven) Senden von Anfragen an einen Server veraltete Inhalte von einem Cache zurückgegeben werden können. Für das eigene Szenario kann dies zu Folge haben, dass die Aktualisierung eines Feeds zwar korrekt vorhergesagt, ein bereits veröffentlichter, neuer Eintrag jedoch nicht zurückgegeben wird, da ein Cache noch veraltete Daten enthält. Um dieses Problem zu umgehen, kann das in Abschnitt 2.3.5 erwähnte cache-control-Feld des HTTP-Headers verwendet werden, welches eine Liste von Parametern aufnehmen kann. Die Details hierzu werden im Anschluss an die Mechanismen zur Realisierung der Cache-Konsistenz dargelegt.

2.5.3. Mechanismen zur Realisierung der Cache-Konsistenz

Die Verfahren zur Realisierung der Cache-Konsistenz werden in *validation* und *invalidation* unterteilt. Validation geht vom Cache aus und wird auch als *client polling* bezeichnet, bei der invalidation muss die Quelle des gecachten Objekts aktiv werden, weshalb der Ansatz auch als *server-driven* bezeichnet wird [RS02, INST04]. Die nachfolgenden Ausführungen beruhen auf einem einfachen Szenario mit nur einem Cache zwischen dem Nutzer und dem Server, auf dem das Original vorliegt. Die Verfahren sind ohne Beschränkung der Allgemeinheit auf hierarchische Caches übertragbar.

Validation

Bei der vom Cache ausgehenden validation fragt der Cache für ein vorgehaltenes Objekt die Quelle an, ob eine Änderung erfolgte. Zur Ermittlung der Frequenz der validation-Abfragen muss zwischen der Wahrscheinlichkeit, dass ein veraltetes Objekt zurückgegeben wird, und den durch häufiges Abfragen entstehenden Kosten abgewägt werden. Neben den Extrema, bei jeder Nutzeranfrage oder nie zu validieren, existieren Lösungen, die auf dem Konzept der time to live (TTL) aufbauen. Die TTL kann entweder explizit vom Server zur Verfügung gestellt oder implizit vom Cache ermittelt werden [RS02].

explizite TTL Zur Realisierung der expliziten TTL wird der HTTP-Header verwendet (siehe Abschnitt 2.3.5). Der Server berechnet das „Ablaufdatum“ der Gültigkeit des Objekts entweder selbst und trägt den Wert in das Expires-Feld ein oder spezifiziert im cache-control-Feld mit der max-age-Direktive die Zeitdauer, nach deren Ablauf ein gecachtes Objekt verifiziert werden muss, bevor es zur Beantwortung von Nutzeranfragen verwendet wird. Um die Verifikation effizient zu gestalten, wird dem Cache zudem der Last-Modified-Zeitstempel übergeben. Zur Verifikation des Objekts stellt der Cache eine bedingte GET-Abfrage an den Server, welcher in der Antwort entweder bestätigt, dass das Objekt nicht verändert wurde, oder die aktuelle Version des Objekts schickt [RS02]. Alternativ kann in der Abfrage das ETag verwendet werden.

Der Ansatz der expliziten TTL birgt zwei Nachteile. Zur Bestimmung der TTL benötigt der Server Kenntnisse über das Änderungsverhalten des Objekts – diese sind in der Regel nicht vorhanden. Ein weiterer, sich hieraus ergebender Nachteil betrifft den Cache und den Nutzer. Wird ein Objekt innerhalb der TTL auf dem Server aktualisiert, so hat der Cache keine Kenntnis darüber und gibt Nutzern in dieser Zeit veraltete Objekte zurück, die aus Sicht des Caches noch valide sind.

implizite TTL Sind die Voraussetzungen der expliziten TTL nicht erfüllt, muss die Verweildauer eines Objekts vom Cache mit einer Heuristik selbst bestimmt werden. Hierzu existieren u. a. verschiedene Ansätze, die unterschiedlichen Typen von Objekten wie Bildern oder Webseiten verschiedene typische Verweildauern zuordnen. Diese sind für die Bestimmung der Änderungsraten von Feeds nicht relevant.

Eine verbreitete Technik ist AdaptiveTTL [CL98, RS02, INST04]. Die Grundidee ist: Je länger ein Objekt nicht verändert wurde, umso länger wird es auch in Zukunft nicht

verändert werden [Cat92]. AdaptiveTTL wird nach Gleichung 2.10 berechnet und ist die Differenz aus dem aktuellen Zeitpunkt und dem Zeitpunkt der letzten Änderung des Objekts. Die Konstante M wird häufig mit $M = 0{,}1$ oder $M = 0{,}2$ belegt, der Schwellwert β garantiert, dass bereits seit sehr langer Zeit nicht aktualisierte Objekte dennoch in großen Abständen validiert werden. Die Berechnung ist in Abbildung 2.7 veranschaulicht, wobei auf den Schwellwert β verzichtet wird.

$$TTL = \min\left((M * (\mathsf{Now} - \mathsf{Last\text{-}Modified})), \beta\right) \tag{2.10}$$

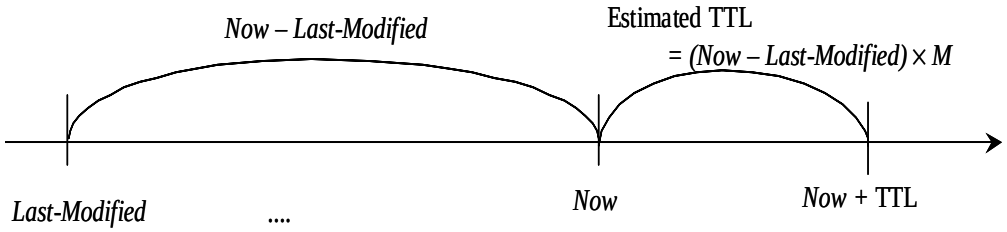

Abbildung 2.7.: Berechnung der time to live [LWLC02].

Tests ergaben, dass mit der Berechnung zwar weniger als ein Prozent der an Nutzer übergebenen Objekte veraltet waren, im Gegenzug jedoch knapp 60 Prozent der Validierungsanfragen des Caches ergaben, dass sich die Objekte nicht verändert hatten [RS02]. Wenngleich der Ansatz sehr simpel ist und keinerlei Historie des Objekts berücksichtigt, so wird er dennoch in Kapitel 4 aufgegriffen und zum Vergleich evaluiert.

Alternative Ansätze basieren auf der Berechnung der Lebensdauer eines Objekts anhand der Historie seiner Änderungsrate [OOW93, YCLL00, LWLC02]. Obwohl die Ziele der Autoren in [YCLL00] von den eigenen divergieren, ist die Idee der Berechnung interessant. Sie schlagen vor, die Lebensdauer eines Objekts auf Basis des Minimums, Maximums oder Mittelwerts der letzten n beobachteten Änderungen zu ermitteln. Leider treffen sie keine Aussage zur Größe von n. In einer Evaluation in [LWLC02] ermitteln die Autoren empirisch, dass eine Fenstergröße von zwei am geeignetsten ist, um das Verhältnis zwischen zu lang vorhergesagten TTL-Werten zu allen vorhergesagten Werten zu minimieren. Leider fehlt die aus der Sicht der eigenen Arbeit wichtige Information, wie stark die vorhergesagten Werte von den tatsächlichen Lebensdauer der Objekte abweichen. Die Autoren führen ihren Ansatz auf die Cache Ersetzungstrategie *Least Recently Used (LRU)-K* [OOW93] aus der Domäne Datenbanken zurück, in der die letzten K Zugriffe auf ein Objekt betrachtet wurden, um zukünftige Zugriffe vorherzusagen. Die Autoren in [OOW93] ermittelten ebenfalls, dass eine Länge der Historie von zwei am besten geeignet ist und längere Historien Änderungen schlechter adaptieren. Der Ansatz der begrenzten Historie wird im weiteren Verlauf in Abschnitt 4.2.3 als *Moving Average* Algorithmus aufgegriffen. Zwei weitere interessante Verfahren werden in [BGR06] vorgestellt und in Abschnitt 2.5.4 separat eruiert.

Die Ansätze der Validation bergen das Risiko, dass ein Nutzer veraltete Objekte aus einem Cache erhält. Möchte der Nutzer sicherstellen, keine veraltete Kopie zu erhalten, so gibt er im cache-control-Feld des HTTP-Headers die Direktive no-cache an und veranlasst Caches, das Objekt neu vom Server zu laden. Die no-cache-Direktive ist für die Ziele der eigenen Arbeit bedeutend und wird bei jeder Abfrage eines Feeds verwendet. Für weitere Details zum cache-control-Feld sei auf die Ausführungen in [RS02, S. 52ff] verwiesen.

Invalidation

Der Grundsatz der invalidation ist eine serverseitige Notifikation aller Caches[5], wenn sich ein Objekt ändert. Dies setzt voraus, dass der Server zu jedem Objekt eine Liste aller Knoten vorhält, die dieses Objekt cachen. Wird das Objekt geändert, so muss zunächst an jeden Cache der Liste eine invalidation-Nachricht geschickt werden. Hierfür wurden verschiedene Mechanismen wie das *Web Cache Invalidation Protocol* [LCD01] und *(adaptive) leases* [GC89, DST99] entwickelt [INST04].

Ein lease ist eine Garantie des Servers, den Cache innerhalb der im lease spezifizierten Zeitdauer über Modifikationen des betreffenden Objekts zu informieren. Nach Ablauf der Zeitspanne muss der Cache das lease erneuern. Der Vorteil besteht in der Reduktion der vom Server vorzuhaltenden Listen.

Zur lückenlosen Integration von Feeds in ein PIS erscheint es sinnvoll, auf diese Techniken des Übertragungsprotokolls zurückzugreifen. Unterstützt ein Server eines der Verfahren, so muss die Effizienz im Einzelfall betrachtet werden. Beträgt die Dauer eines lease z. B. nur wenige Minuten und die Änderungsrate des Feeds eine Woche, so ist es fraglich, ob die hohe Anzahl Nachrichten zur Erneuerung des lease den Gewinn an Aktualität rechtfertigt. Von der Betrachtung weiterer Techniken zur invalidation wird abgesehen. Eine Übertragung der Konzepte auf Mechanismen zur Benachrichtigung von Nutzern über Aktualisierungen von Feeds bedingt eine aktive Teilnahme der Server und ist aus eigener Sicht nicht realistisch umsetzbar. Verwandte Ansätze zur Benachrichtigung von Nutzern über Aktualisierungen von Feeds werden in Abschnitt 2.7.2 vorgestellt.

2.5.4. Weitere Techniken zur Validierung von Cache-Inhalten

Eine sehr umfassende Betrachtung pull-basierter Techniken in verschiedenen Einsatzszenarien wie validation erfolgt in [BGR06]. Die Autoren treffen anhand der Historien der Objekte Vorhersagen, wann die nächste Änderung bzw. wie viele Änderungen in einem Zeitintervall zu erwarten sind und evaluieren ihre Algorithmen u. a. anhand der Anzahl erfolgter Abfragen und der Zeitdifferenz zwischen der Aktualisierung auf dem Server und der nächsten Abfrage des Objekts. Diese Ziele und Metriken decken sich mit den Zielen der eigenen Arbeit und werden im Folgenden eruiert.

[5] Genauer gesagt aller Caches, die Objekte direkt von diesem Server erlangt haben.

Modellierung von Änderungen

Im Unterschied zu den meisten im Bereich Web Crawling betrachteten Arbeiten setzen die Autoren in [BGR06] die Existenz einer vollständigen Historie der Objekte voraus, welche sie durch Modifikationen der Server erreichen. Ein Nutzer bekommt in jeder Antwort des Servers die Zeitstempel verpasster Änderungen mitgeteilt. Dieser Ansatz steht zunächst im Konflikt mit dem eigenen Ansatz, die Quellenautonomie nicht einzuschränken. Beachtet man jedoch, dass Feeds zumindest eine begrenzte Historie aufweisen, so kann die Anforderung der lückenlosen Historie bei Feeds als weitestgehend erfüllt betrachtet werden.

In einer Untersuchung verschiedener Datensätze stellen die Autoren fest, dass Änderungsraten aufgrund von Faktoren wie Tageszeiten, Wochentagen und unerwarteten Ereignissen variieren, sodass Aktualisierungen deterministisch täglich zur gleichen Tageszeit erfolgen, vollkommen unvorhersehbar sind oder Muster zwischen diesen beiden Extrema aufweisen können. Im Gegensatz zu den Arbeiten wie [CG00, CG03a] nehmen die Autoren einen *piecewise constant Poisson process* als Modell der Änderungsraten an, um zyklische Variationen zu berücksichtigen. Dieses in [GE01] eingeführte Modell wird auch als *nonhomogeneous Poisson process* bezeichnet, da die Änderungsrate λ nicht mehr kontinuierlich ist. Dies ist in der linken Grafik in Abbildung 2.8 dargestellt, in der die durchschnittliche Änderungsrate pro Stunde ermittelt wurde. Die über den Tagesverlauf variierende Änderungsrate ist eindeutig zu erkennen. Dem gegenüber steht in der rechten Grafik die Darstellung der durchschnittlichen Änderungsrate des selben Objekts als homogener Poisson-Prozess, der diese feingranulare Differenzierung nicht zulässt. Zur Bestimmung der Periodendauer, d. h. ob ein einzelner Tag oder eine komplette Woche als Periodendauer modelliert wird sowie der Granularität der Abschnitte, in die eine Periode unterteilt wird, verweisen die Autoren auf das menschliche Augenmaß oder statistische Methoden wie *statistical process control (SPC)* oder *change-point theory*, vertiefen dies jedoch nicht [GE01, BGR06].

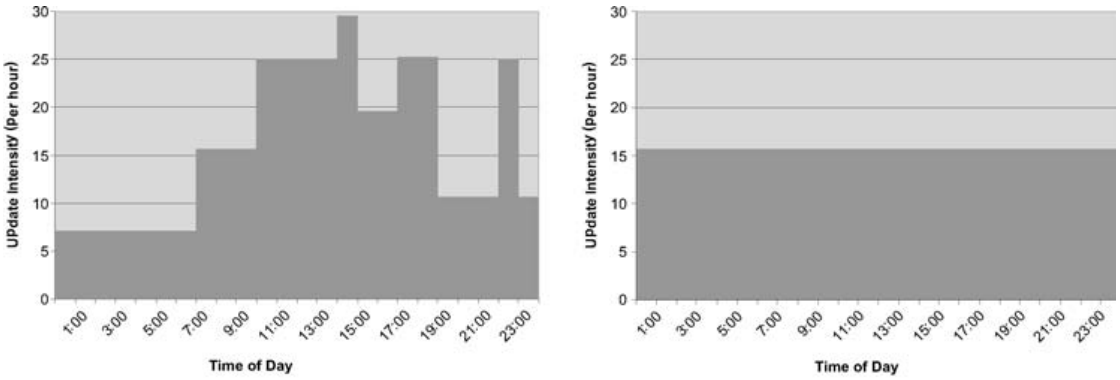

Abbildung 2.8.: Vergleich zweier Modellierungen der Änderungsrate eines Objekts. Links: inhomogener Poisson-Prozess mit Periodendauer von einem Tag und Abschnitten von einer Stunde, rechts: homogener Poisson-Prozess [BGR06].

Vorhersage auf individuellen und aggregierten Historien

Zur Vorhersage zukünftiger Änderungen schlagen die Autoren in [BGR06] die zwei Strategien IndHist und AggHist vor, die auf der individuellen Historie eines Objekts bzw. auf den aggregierten Historien von Objekten mit ähnliche Aktualisierungsmustern basieren. Leider geben sie keine Hinweise, *wie* ähnliche Aktualisierungsmuster zu identifizieren sind, sodass im Folgenden nur IndHist im Detail an einem Beispiel betrachtet wird.

Initial wird die Änderungsrate λ für ein Objekt über einen Beobachtungszeitraum von acht Tagen ermittelt. Die Historie ist Tabelle 2.1 zu entnehmen. Angenommen das Objekt wurde zuletzt um 11:30 abgefragt und um 14:00 des gleichen Tages soll ermittelt werden, ob bereits ein neue Änderung zu erwarten ist. Hierfür summieren die Autoren die λ-Werte der betreffenden Zeitabschnitte anteilig. Im Beispiel ergibt sich $1/2 * 0{,}125 + 0{,}125 + 0{,}375 = 0{,}5625$, d. h. es sind durchschnittlich 0,6 Änderungen im Zeitraum 11:30 bis 14:00 Uhr zu erwarten. Die Entscheidung, ob das Objekt erneut zu laden ist, erfolgt anhand eines Schwellwertes θ mit $\theta < 1$: Ist die Anzahl erwarteter Änderungen größer als θ, erfolgt eine Synchronisation mit dem Server, andernfalls nicht. Die Berechnung des Zeitpunktes einer nächsten Änderung erfolgt analog, sodass der Zeitpunkt ermittelt wird, an dem die erwartete Anzahl Änderungen den Schwellwert θ erreicht [BGR06]. Eine formale Betrachtung der Komplexität erfolgt in [GE01] und ergibt $\mathcal{O}(c)$, wobei c die Anzahl der Zeitabschnitte ist, die in einer Vorhersage betrachtet werden. Die Werte können zudem vorausberechnet und mit dem Modell abgelegt werden.

Zeitraum	Summe Anzahl Änderungen in 8 Tagen	λ pro Stunde
11:00 - 12:00	1	0,125
12:00 - 13:00	1	0,125
13:00 - 14:00	3	0,375

Tabelle 2.1.: Historie des Objekts nach [BGR06].

In der Evaluation kommen in [BGR06] drei Datensätze zum Einsatz, an denen die Algorithmen untereinander und mit AdaptiveTTL verglichen werden. Als Metriken kommen u. a.[6] zum Einsatz: die absolute Anzahl von Abfragen, die Anzahl Abfragen, bei denen eine Aktualisierung entdeckt wurde sowie die durchschnittliche Verzögerung, mit der Aktualisierungen entdeckt wurden. Zum Training wurden acht Tage verwendet, die verbleibenden Anteile der Datensätze wurden zur Evaluation eingesetzt. Für den realen Einsatz schlagen die Autoren vor, die Modelle in gewissen Zeitabständen neu zu trainieren, schlagen jedoch keine konkreten Werte oder Metriken zur Bestimmung der Zeitpunkte vor. Alle nachfolgenden Angaben sind Näherungswerte.

Um von einer Million Aktualisierungen *zyklischer* Objekte 90 Prozent zu erfassen, benötigt IndHist 2,25 M Abfragen, AdaptiveTTL 3,4 M Abfragen, d. h. rund 50 % mehr. Die Bestimmung der durchschnittlichen Verzögerung erfolgt leider an zwei anderen Datensätzen, deren Ergebnisse untereinander ähnlich sind. Um auf einem Datensatz mit 10 K Änderungen in einem Zeitraum von 447 Tagen eine durchschnittliche Verzögerung

[6] Die weiteren Metriken sind für die eigene Arbeit nicht von Interesse, da mit ihnen andere Ziele verfolgt werden und die Ergebnisse nicht mit denen der eigenen Arbeit vergleichbar sind.

von fünf Minuten zu erzielen, benötigt IndHist 80 K Abfragen, AdaptiveTTL hingegen 170 K, d. h. mehr als doppelt so viele. Dieser Unterschied ist beachtlich. Eine kritische Betrachtung der Werte und ein Vergleich beider Methoden mit dem sehr simplen Ansatz, die Quelle in immer gleichen Intervallen abzufragen [CG00], führt zu der Feststellung, dass der simple Ansatz diesen beiden Algorithmen überlegen ist: Um die Quelle 14 Monate alle fünf Minuten abzufragen, werden 129 K Abfragen benötigt. Mit diesem Intervall ist sichergestellt, dass die durchschnittliche Verzögerung *maximal* fünf Minuten beträgt. Bei einer zufälligen Verteilung der Änderungen ist jedoch davon auszugehen, dass die durchschnittliche Verzögerung der Hälfte der Länge des Abfrageintervalls entspricht [CG00]. Folglich muss die Quelle nur alle zehn Minuten abgefragt werden und es ergeben sich 65 K Abfragen bei einem solchen fixen Intervall. Dieses Ergebnis überrascht.

Es erscheint dennoch interessant, IndHist zu reimplementieren und in Kapitel 4 auf dem eigenen Datensatz mit anderen zu vergleichen. Hierbei muss jedoch beachtet werden, dass in [GE01, BGR06] sowohl die Periodendauer von einem Tag bzw. einer Woche als auch die gewählte Untergliederung in Abschnitte manuell vorgenommen wurde. Für die Reimplementierung wird daher vorgeschlagen, die in [GE01, BGR06] gewählten Muster zu übernehmen. Eine weitere in Kapitel 4 zu klärende Frage ist die Länge der Trainingsphase. In [GE01] erfolgte eine Unterteilung in ca. 20 Wochen Trainings- und sechs Wochen Testphase, in [BGR06] erfolgten u. a. Messungen mit einer Woche Training für eine 14-monatige Testphase.

Adaptive Vorhersage

Aufbauend auf den beiden Strategien IndHist und AggHist sowie AdaptiveTTL[7] werden zwei *adaptive* Strategien IndHist/AggHist und IndHist/TTL vorgestellt, die zur Laufzeit dynamisch zwischen IndHist und AggHist respektive IndHist und AdaptiveTTL umschalten.

IndHist/AggHist weicht auf AggHist aus, solange die individuelle Historie eines Objekts zu kurz ist, um stabil zu sein. Die Wahl des Algorithmus erfolgt anhand des Verhältnisses der Anzahl der Zeiträume eines Tages $NumSub$, in denen bereits Änderungen gefunden wurden, zu allen bekannten Änderungen $NumUpdates$. Gilt $NumSub/NumUpdates > T_{ind}$, so wird AggHist eingesetzt, andernfalls IndHist. Sie ermitteln $T_{ind} \simeq 0.5$ empirisch als geeigneten Schwellwert und zeigen in einer Evaluation eines simulierten Cache, dass der kombinierte Algorithmus bei der gleichen Anzahl Synchronisationen mit dem Server weniger stale hits erzeugt.

Wenngleich der Einsatz aggregierter Historien zur Vorhersage interessant ist, so bleibt eine essentielle Frage unbeantwortet: Wie werden ähnliche Muster der Änderungsraten identifiziert? Diese Aufgabe erscheint um so schwerer, je kürzer die zur Verfügung stehende Historie eines Objektes ist, das einem Cluster zugeordnet werden muss. IndHist/AggHist wird daher nicht in der eigenen Evaluation aufgegriffen.

[7] Die in [BGR06] verwendete Variante der AdaptiveTTL ist eine vereinfachte Version der in Gleichung 2.10 (Seite 52) vorgestellten Variante aus [RS02, INST04]. Die Autoren verzichten auf die obere Schranke β, sodass nur der Parameter M gewählt werden muss.

IndHist/TTL setzt dynamisch AdaptiveTTL ein, wenn ein nicht vorhersagbarer *burst* erkannt wird. Als burst bezeichnen die Autoren Zeitabschnitte, in denen die momentan beobachtete Änderungsrate von der im Modell hinterlegten abweicht, wobei sie sich auf ansteigende Änderungsraten konzentrieren. Die Erkennung eines bursts sei am Beispiel von Abbildung 2.9 veranschaulicht. Die Autoren führen einen frei wählbaren Zeitabschnitt ein, der z. B. eine Stunde umfasst. Das Objekt wurde zum Zeitpunkt $\tau_{\mathcal{A}}$ das letzte mal abgefragt, die acht tatsächlichen Änderungen 1–8 sind bekannt, 9 und 10 sind dem Abfragenden unbekannt, ihre Anzahl – zwei – ist gesucht.

Nach jeder Abfrage $\mathcal{A}$ eines Objekts wird die Anzahl der im vorangegangenen Zeitabschnitt $V_{\mathcal{A}}$ beobachteten Änderungen mit der von IndHist vorhergesagten Anzahl von Änderungen verglichen. Überschreitet das Verhältnis einen Schwellwert T_{burst}, so wird ein burst erkannt und AdaptiveTTL zur Berechnung des Zeitpunktes der nächsten Abfrage $\tau_{\mathcal{B}}$ gewählt, andernfalls IndHist. Im Beispiel wurden im Zeitabschnitt $V_{\mathcal{A}}$ die drei Änderungen 5–7 vorhergesagt, vier (5–8) sind eingetreten und so mit einem Verhältnis von $1{,}\overline{3}$ der von den Autoren empirisch ermittelte Wert $T_{burst} = 2$ unterschritten und IndHist zur Vorhersage eingesetzt.

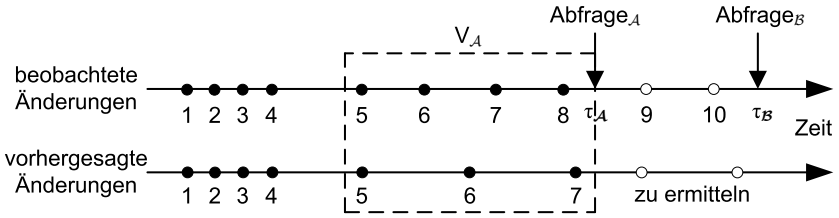

Abbildung 2.9.: Ermittlung von bursts in der Änderungshistorie eines Objekts.

In der Evaluation eines Caches zeigen die Autoren anhand ausgewählter Messungen, dass IndHist/TTL mit der gleichen Anzahl Abfragen mehr Änderungen in bursts erkennt, als IndHist oder AdaptiveTTL einzeln. Für zyklische Objekte sind IndHist und IndHist/TTL äquivalent. Basierend auf ihrem Datensatz von Webseiten ermittelten sie zudem, dass ein Zeitabschnitt V von einem Tag besser zur Erkennung von bursts geeignet ist, als nur die letzte Stunde zu betrachten. Eine weitere interessante Erkenntnis der Autoren ist, dass sie die Vorhersage für Objekte mit bursts verbessern konnten, indem statt der kompletten Historie nur wenige Tage in Form von gleitenden Fenstern betrachtet werden. Leider evaluieren sie dies nicht an zyklischen Objekten – zu diesen hatten sie die Aussage getroffen, dass IndHist umso besser arbeitet, je länger die zur Verfügung stehende Historie ist.

Zusammenfassend stellen die Autoren fest, dass ihre Historien-basierten Ansätze bei Objekten mit zyklischen Mustern besser geeignet sind, als AdaptiveTTL. Bei azyklischem Verhalten mit bursts erzielen sie ähnliche Ergebnisse. Daher sind die Algorithmen IndHist und IndHist/TTL sowie die Idee der gleitenden Fenster für die eigene Arbeit von hohem Interesse. Die Algorithmen werden im Folgenden generisch als *PostRate*-Algorithmen bezeichnet sowie in Kapitel 4 aufgegriffen und mit anderen Algorithmen verglichen.

2.5.5. Replikation am Beispiel von Content Distribution Networks

Bei der Replikation von Objekten tritt ebenfalls das Problem der Konsistenz auf, wie bereits in der Domäne Datenbanken in Abschnitt 2.2.3 erläutert wurde. Im Internet werden zur Replikation von Objekten u. a. CDNs eingesetzt, um die Inhalte näher an den „Rand“ des Internets zu verlagern. Abfragen von Nutzern werden von speziellen Komponenten zu einem Knoten des CDNs umgeleitet, der in ihrer Nähe ist. CDNs zeichnen sich durch ihre hohe Anzahl von Knoten aus – Akamai hat zum Beispiel mehr als 50.000 Server in 71 Ländern[8]. Bei diesen Größen sind Verfahren wie leases nicht umzusetzen, sodass hierarchische Verfahren eingesetzt werden [INST04]. Diese basieren auf push-Mechanismen und lassen sich daher nicht auf die Ziele der eigenen Arbeit übertragen.

2.5.6. Cache Ersetzungsstrategien

Ersetzungsstrategien werden benötigt, wenn der dem Cache zur Verfügung stehende flüchtige oder persistente Speicherplatz ausgeschöpft ist [INST04]. Die Heuristik LRU-K als Spezialfall von LRU wurde bereits in Abschnitt 2.5.3 skizziert. Weitere Algorithmen wie GreedyDual-Size basieren u. a. auf dem Zugriffsverhalten der Nutzer und lassen sich nicht direkt auf das in der eigenen Arbeit betrachtete Problem der Vorhersage neuer Einträge eines Feeds abbilden, weshalb keine vertiefte Betrachtung der Cache Ersetzungsstrategien erfolgt.

2.5.7. Prefetching

Ziel des prefetching ist eine Vorhersage der Objekte, die ein Nutzer als nächstes anfordern wird und diese proaktiv zu laden, um die Antwortzeit des Caches zu minimieren. Diese Vorhersage basiert häufig auf einem Markov-Modell oder der Verlinkungsstruktur der Webseiten [RS02]. Die auf den Hyperlinks basierenden Algorithmen lassen sich nicht auf die Vorhersage von Feeds übertragen. Markov-Modelle scheinen zunächst eher geeignet und werden im Detail betrachtet.

Zum prefetching werden verschiedene Merkmale wie Verhaltensmuster der Nutzer oder die Reihenfolge aufgerufener Objekte $x_1, ..., x_m$ ausgewertet, um die potentiell als nächstes aufgerufenen Objekte $y_1, ..., y_n$ vorherzusagen. Den Objekten werden eindeutige Identifikatoren zugeordnet und die Sequenz $x_1, ..., x_m$ anschließend dem Algorithmus als Trainingsmenge übergeben. Abbildung 2.10 zeigt den Graph eines Markov-Modells erster Ordnung mit $m = 4$ Objekten als Knoten und den Übergangswahrscheinlichkeiten als Gewichte an den gerichteten Kanten.

Reale Graphen haben eine um mehrere Größenordnungen höhere Anzahl von Knoten und weisen eine Vielzahl von Kanten pro Knoten auf. Zur Vorhersage werden die Gewichte der ausgehenden Kanten eines Objekts absteigend sortiert und anschließend die ersten i Objekte proaktiv geladen. Wie das Beispiel $x_4 \rightarrow x_3$ in Abbildung 2.10 zeigt, können unter Umständen mehrere Schritte vorhergesagt werden.

[8] `http://www.akamai.de/technology_index.html`, letzter Zugriff 25.06.2011

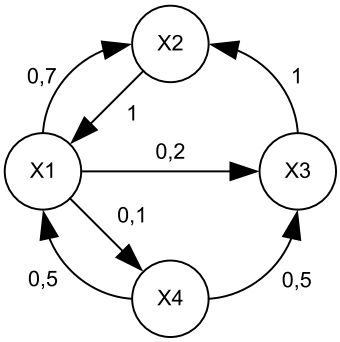

Abbildung 2.10.: Graph eines Markov-Modells erster Ordnung [RS02].

Generelle Nachteile der Markov-Modelle sind die hohen Kosten für die Berechnung und den Speicherbedarf. Eine Vorhersage ist umso besser, je länger die Trainingsphase ist und je mehr Pfade ausgewertet werden. Bei der Übertragung des Ansatzes auf die Vorhersage von Zeitpunkten, wann ein neuer Eintrag eines Feeds erscheinen wird, müssten statt konkreter Objekte Zeitpunkte oder Zeitintervalle vorhergesagt werden, in denen ein Feed aktualisiert wird. Ein möglicher Ansatz ist der Einsatz der in Abschnitt 2.2.1 vorgestellten temporalen Segmentation [CC09]. Die sich ergebende Komplexität der Berechnung einer Vorhersage wird im Vergleich zu anderen Methoden als zu hoch eingeschätzt und Markov-Modelle zur Vorhersage von Feeds nicht vertieft.

2.5.8. Zusammenfassung Web Caching

Aus den Betrachtungen der Domäne Web Caching gehen mehrere wichtige Erkenntnisse hervor, welche einerseits die Vorhersage von Änderungen und andererseits die Integration von Feeds in ein PIS unter realen Bedingungen betreffen.

Vorhersagen von Änderungen können alternativ bereits vom Server zur Verfügung gestellt oder vom Nutzer – dem Cache – getroffen werden. Bei den vom Server im HTTP-Header optional bereitgestellten Werten Expires oder max-age handelt es sich um Prognosen, die falsch sein können. Garantierte Benachrichtigungen bei Änderungen eines gecachten Objekts werden mit Verfahren wie dem Web Cache Invalidation Protocol [LCD01] oder (adaptive) leases [GC89, DST99] vorgeschlagen. Die Effizienz dieser Techniken ist im Einzelfall zu überprüfen.

Zur Ermittlung von Änderungen anhand der Historie wurden die Verfahren AdaptiveTTL [CL98, RS02, INST04], IndHist und IndHist/TTL als Kombination der beiden [BGR06] im Detail betrachtet. Diese Algorithmen werden in Kapitel 4 aufgegriffen und in einer Evaluation mit anderen Verfahren verglichen. Darüber hinaus wurde in [YCLL00, BGR06] vorgeschlagen, die Änderungsrate mit einem gleitenden Fenster, d. h. dynamisch zur Laufzeit auf einer begrenzten Historie zu ermitteln. Ebenso interessant sind die Ideen, Historien mehrerer Objekte zu aggregieren und als Heuristik einzusetzen, wenn für ein neues Objekt keine hinreichende Länge einer

Historie gegeben ist sowie kurzzeitige spontane Variationen der Änderungsrate separat zu betrachten [BGR06].

Reale Umgebungsbedingungen im Web Bei der Übertragung von Feeds im speziellen bzw. Web-Objekten im Allgemeinen müssen die Auswirkungen von (Web-)Caches beachtet werden. In Abhängigkeit von den eingesetzten Mechanismen zur Realisierung der Cache-Konsistenz besteht die Gefahr, veraltete Inhalte aus einem Cache zu erhalten (stale hit). Um dieses Risiko auszuschließen[9], muss in einer Abfrage die no-cache-Direktive im Cache-Control-Feld des HTTP-Headers angegeben werden.

Zur Minimierung der zu übertragenden Daten und der zur Verarbeitung eingesetzten Ressourcen auf Server-, Cache- und Nutzerseite sollten bedingte GET-Abfragen gestellt werden, wann immer mindestens einer der Werte Last-Modified oder ETag im HTTP-Header einer Antwort zur Verfügung stehen.

2.6. Sonstige Arbeiten im Web-Umfeld

Es folgt die Betrachtung einiger Ansätze aus dem Web-Umfeld, die keiner der bereits betrachteten Domänen direkt zugeordnet werden können. Zu den vorgestellten Techniken zählen Web Hooks und auf HTTP Streaming basierende Techniken.

2.6.1. Vorhersage der Popularität von Nachrichten

Ein zunächst zur eigenen Arbeit verwandt erscheinender Ansatz ist die Vorhersage neuer Nachrichten bei Twitter [HDD11]. Das Ziel der Autoren ist, frühzeitig populäre Nachrichten zu erkennen und die Anzahl sogenannter *retweets*[10] vorherzusagen, d. h. wie häufig diese Nachrichten von anderen Nutzern weiter verbreitet werden. Hierzu klassifizieren die Autoren Nachrichten anhand der zu erwartenden Anzahl von retweets, wobei sie vier Klassen unterscheiden: keine retweets, bis 100, bis 10.000 oder mehr als 10.000 retweets. Sie analysieren Charakteristiken aus dem Inhalt der tweets, der Topologie des sozialen Netzwerks und temporalen Eigenschaften aus der Historie wie der Zeitdifferenz zwischen der aktuellen Nachricht und ihrem ersten Auftreten oder der durchschnittlichen Zeitdifferenz zwischen aufeinander folgenden Nachrichten in einer betrachteten Reihe.

Der Unterschied zur eigenen Arbeit ist, dass retweets generell in kausaler Abhängigkeit zu vorangegangenen Nachrichten stehen, da sie sich direkt auf diese beziehen. Diese Abhängigkeit kann zwischen Einträgen von Feeds nicht vorausgesetzt werden. Darüber hinaus ist das Ziel der eigenen Arbeit die Vorhersage der Zeitpunkte der nächsten Nachrichten, während in [HDD11] nur eine grobgranulare Schätzung der Anzahl der Nachrichten erfolgt.

[9] Es wird vorausgesetzt, dass sich der Cache HTTP-konform verhält und die Direktive korrekt verarbeitet.
[10] `http://support.twitter.com/articles/104996-was-ist-ein-retweet`, letzter Zugriff 27.09.2011

2.6.2. Web Hooks

Web Hooks sind eine Art publish-subscribe-Mechanismus, der auf HTTP-POST [FGM+99] basiert und auch als *HTTP callback* bezeichnet wird. Ein Nutzer registriert bei einem Server eine URL, unter der er benachrichtigt werden möchte, wenn ein bestimmtes Ereignis eintritt. Sobald dieses Ereignis stattfindet, benachrichtigt der Server den Nutzer per HTTP-POST. Diese in der Praxis bereits eingesetzte Technik erfordert eine aktive Beteiligung des Servers der Quelle sowie einen Server auf der Nutzerseite, um die HTTP-POST-Nachrichten zu verarbeiten [Lin11].

Die zweite Bedingung (Server auf Nutzerseite) ist bei der Integration von Feeds in ein PIS leicht zu realisieren, eine aktive Beteiligung aller Diensteanbieter widerspricht jedoch dem eigenen Ziel, die Quellenautonomie nicht einzuschränken. Web Hooks sind bisher[11] nicht standardisiert und in der Liste der Systeme [Lin11], die Web Hooks einsetzen, ist keines der bekannten CMS aufgeführt.

2.6.3. Auf HTTP Streaming basierende Techniken

Techniken, die grundlegend auf HTTP Streaming[12] basieren, werden unter anderem von *Pushlets* [Bro02] oder Programmierkonzepten wie *Ajax – Asynchronous JavaScript and XML* und *Comet* [Rus06] aufgegriffen, um Daten asynchron zwischen einem Web Server und dem Browser des Nutzers zu übertragen bzw. neue Daten über einen bestehenden Kommunikationskanal an den Nutzer zu schicken, weshalb sie auch als *server-push* bezeichnet werden. Dies sei an einem Beispiel erläutert, welches den Bogen vom Web Crawling über grundlegende Techniken des Webs hin zu Systemen zur Verarbeitung von Feeds spannt.

Die einst in einem kommerziellen System [Six10b] eingesetzte Technik war zur Optimierung von Web Crawlern angedacht. Web Crawler sowie andere Nutzer konnten einen speziellen Atom Feed abonnieren, der alle Änderungen an Seiten enthielt, die mit der vom Betreiber angebotenen Software erstellt wurden. Im Unterschied zu konventionellen Atom Feeds hatte dieses spezielle Dokument kein explizites Ende, sodass die HTTP-Verbindung aufrecht erhalten wurde. Neue Inhalte wurden umgehend am jeweils aktuellen Ende eingefügt, sobald sie im Quellsystem angelegt wurden. Zudem wurden regelmäßig Zeitstempel in den Strom eingefügt, um die Verbindung aufrecht zu halten und Wiedereinstiegspunkte bei getrennten Verbindungen bereitzustellen [Six10b].

Im Falle von [Six10b] hat sich die Technik nicht durchgesetzt und der Anbieter ist auf das im Anschluss in Abschnitt 2.7.2 eingeführte PubSubHubbub Protokoll umgestiegen [Six10a], sodass es nicht möglich ist, den beschriebenen Ansatz für einen Teil der zu integrierenden Feeds zu nutzen. Wenngleich Streaming-Ansätze in Web-Anwendungen wie z. B. Webmail eingesetzt werden können, um Nutzer zeitnah über neu eingegangene Nachrichten zu informieren, ohne den Server regelmäßig abzufragen, so würde ein genereller Wechsel von pull-basierten Zugriffen zu server-push ein Reihe zu lösender Probleme wie die Skalierbarkeit auf Seiten des Servers mit sich bringen. Das Resümee aus Abschnitt 2.5.3

[11] Stand 26. September 2011

[12] Nicht zu verwechseln mit HTTP Live Streaming wie [PM10] zur Übertragung von Video.

gilt hier analog: stellt die Transportschicht – HTTP – Mechanismen zur Benachrichtigung von Änderungen bereit, sollten diese auf ihre Eignung zur Integration von Feeds im realen Einsatz genutzt werden.

Die Arbeit basiert auf der Annahme, dass in naher Zukunft kein genereller Wechsel von pull zu push stattfindet, Mechanismen zum intelligenten Abfragen weiterhin benötigt werden und mit Technologien wie server-push oder Web Hooks koexistieren.

2.7. Verarbeitung von Feeds

Forschungen im Feed-Umfeld, die sich mit der Thematik des Zugriffs auf Feeds und den entstehenden Fragen und Problemen bezüglich neuer Einträge beschäftigen, können in zwei Klassen unterteilt werden. Auf der einen Seite werden alternative Ansätze zur Verteilung von Feeds an Interessenten entwickelt. Der orthogonale Ansatz ist die Beibehaltung bestehender Technologien und Weiterentwicklung der Algorithmen zum Zugriff auf Feeds. Der erste Ansatz setzt eine aktive Beteiligung der Anbieter von Feeds voraus und wird in Abschnitt 2.7.2 nur kurz betrachtet. Im Fokus der Betrachtungen stehen Ansätze, welche die Quellenautonomie berücksichtigen. In Abschnitt 2.7.3 wird eine alternative Variante der in Abschnitt 2.5.4 eingeführten PostRate-Algorithmen vorgestellt, gefolgt von Betrachtungen zur Vorhersage mit begrenzten Systemressourcen in Abschnitt 2.7.4. Die Betrachtungen werden mit einem hybriden Ansatz abgerundet, in dem durch die Kombination zweier Algorithmen die in zahlreichen anderen Arbeiten benötigte Trainingsphase überbrückt werden kann.

Die Notwendigkeit der differenzierten Betrachtung von Feeds im Vergleich zu herkömmlichen Webseiten ist in [SMPD05] gut dargestellt. Die Autoren identifizierten vier Unterschiede zwischen Feeds und Webseiten, die in einem erhöhten Ressourcenverbrauch resultieren. Wenngleich sich die Autoren auf das Verhalten von Nutzern beziehen so sind ihre Aussagen im eigenen Kontext der Integration von Feeds in ein PIS gleichwohl zutreffend.

Polling Feeds werden wiederholt abgefragt, auch wenn kein neuer Inhalt vorliegt. Feed-Reader-Programme verwenden üblicherweise[13] ein Intervall von einer Stunde.

Superfluity Mit den derzeitigen Verfahren werden bei jeder Übertragung generell *alle* Einträge eines Feeds übermittelt, statt nur dem Nutzer unbekannte Einträge zu senden.

Stickiness Nutzer neigen dazu, einmal abonnierte Feeds nicht wieder aus ihrem Feed Reader zu entfernen. Im Unterschied zu Webseiten, an denen der Nutzer sein Interesse verliert und diese nicht erneut abruft, bleiben Feeds häufig abonniert und ungelesen.

24 h traffic Feed-Reader-Software läuft häufig als permanenter Prozess im Hintergrund, sodass Feeds auch dann abgefragt werden, wenn der Nutzer inaktiv ist. Dies ist ein deutlicher Unterschied zum Surfen, da Webseiten hier nur aktiv aufgerufen werden.

[13] Google's Feed Crawler verwendet ein Intervall von rund einer Stunde (`http://code.google.com/apis/ajaxfeeds/documentation/`) und viele andere Feed-Reader-Programme folgen diesem Ansatz: `http://www.therssweblog.com/?guid=20070408105324`, letzter Zugriff Januar 2011 und `http://www.allthingsdistributed.com/historical/archives/000413.html`, letzter Zugriff Juni 2011

Die ersten beiden sowie der vierte Punkt treffen uneingeschränkt auch auf das eigene Szenario zu. Die Stickiness ist nur begrenzt relevant, da irrelevante Feeds aus dem Integrationssystem entfernt werden können.

2.7.1. Analyse der Änderungsraten von Feeds

Neben zahlreichen Arbeiten, die Änderungsraten und andere Statistiken zu Feeds nur an kleinen Datensätzen betrachten, werden in [LRS05, SCC07] umfangreichere Analysen durchgeführt. Neben den Statistiken über Feeds enthalten sie Analysen des Nutzerverhaltens, die hier nicht vorgestellt werden.

Der rund 100.000 Feeds umfassende Datensatz in [LRS05] wurde über einen Zeitraum von 84 Stunden stündlich abgefragt. Als Resultat erhielten die Autoren rund 3,7 M Schnappschüsse, wobei mehr als der doppelte Wert zu erwarten wäre. Sie führen dies auf die Überlastung von Servern bzw. des Netzwerks sowie serverseitige Limitierungen des Abfrageintervalls zurück, aufgrund des sehr hohen Werts erscheint dies jedoch zweifelhaft. Die ermittelte Verteilung der Formate ist dennoch interessant und nicht durch die genannten Probleme beeinflusst. Mit einem Anteil von 98 % der Feeds war RSS unumstritten das dominierende Format, die verbleibenden 2 % waren Atom Feeds. Eine Untersuchung der Subformate ergab, dass RSS 2.0 knapp zwei Drittel des Datensatzes ausmacht, die Versionen 0.91 und 1.0 haben einen jeweiligen Anteil von einem Sechstel, andere Subformate wie RSS 0.90, 0.93 und 0.94 sind mit zusammengenommenen 0,2 % absolute Einzelfälle. Mehr als 80 % der Feeds sind kleiner als 10 kB, der Median liegt bei 5,8 kB, das arithmetische Mittel ist 10 kB und nur 0,1 % sind größer als 100 kB, sodass die Größen einer Zipf-Verteilung entsprechen. Diese Werte werden in den Abschnitten 3.7.1 und 3.7.3 mit dem eigenen Datensatz verglichen.

Bezüglich der Änderungsraten stellten die Autoren fest, dass bei der reichlichen Hälfte der Feeds innerhalb der ersten Stunde des Beobachtungszeitraums (von 84 Stunden) mindestens ein neuer Eintrag zu verzeichnen war, wohingegen ein Viertel der Feeds gar nicht aktualisiert wurden. Diese Beobachtung untermauert die Notwendigkeit unterschiedlicher Frequenzen zur Abfrage der Feeds. Untersuchungen der Korrelation zwischen Änderungsrate und Größe der Feeds führten zu dem Ergebnis, dass hier kein nennenswerter Zusammenhang zu erkennen ist [LRS05]. Weitere Analysen ergaben, dass die XML-Dokumente im Schnitt nur 7 % neue Inhalte enthalten. Konsumenten könnten daher eine Menge Bandbreite sparen, wenn statt des kompletten Feeds jeweils nur geänderte Abschnitte übertragen würden. Darüber hinaus schlagen sie vor, das Aktualisierungsintervall von FeedReadern manuell pro Feed entsprechend dessen Aktualisierungsintervall einzustellen sowie Peer-to-Peer (P2P)-Systeme einzusetzen, um die Serverlast hochfrequentierter Feeds zu verteilen. Leider schlagen sie keine Details zu den Algorithmen vor, sodass nur vermutet werden kann, dass sie den später in Abschnitt 4.2.2 als *Fix Learned* vorgestellten Algorithmus meinen, bei dem das Intervall beim ersten Zugriff auf den Feed gelernt und anschließend nie mehr geändert wird.

Ein interessanter Vergleich der Änderungsraten von Feeds ist in Abbildung 2.11 dargestellt [SCC07]. Jeder Feed ist als einzelner Datenpunkt entsprechend seiner Änderungsrate zu Beginn (Abszise) und Ende (Ordinate) des Experiments dargestellt. Die Autoren

zeigen, dass die Feeds ihres 10.000 Feeds umfassenden Datensatzes recht konstante durchschnittliche Änderungsraten aufweisen, da 50 % der Feeds (rot dargestellt) auf der Diagonalen liegen und 90 % der Feeds (gelb dargestellt) nur geringe Abweichungen der Intervalle aufweisen.

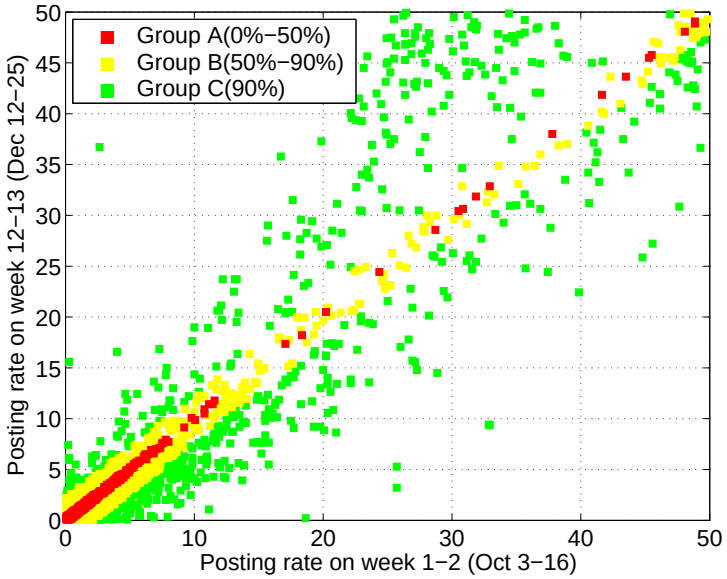

Abbildung 2.11.: Vergleich der durchschnittlichen Änderungsraten von Feeds in den Wochen 1–2 und 12–13 eines dreimonatigen Datensatzes [RMP+07].

Diese Untersuchung ist interessant, die von den Autoren gewählte Einteilung der Gruppen betrachtet jedoch nur absolute Werte und versteckt daher eine für die adaptive Vorhersage benötigte Information: das Verhältnis der Änderungsraten zueinander. Angenommen ein Feed hatte in den ersten beiden Wochen eine Änderungsrate $\lambda_{1,2} = 5$ und nach zwei Monaten $\lambda_{12,13} = 3$, dann betrug sie nur noch 60 % des ersten Wertes. Dies ist eine deutliche Änderung, die von adaptiven Algorithmen berücksichtigt werden sollte.

Die in [LSC+07] identifizierten Aktualisierungsmuster basieren hauptsächlich auf Blogs mit einem Autor. Die Autoren stellen fest, dass von (jeweils einer) Personen geschriebene Blogs in der Regel Variationen in den Tageszeiten aufweisen, zu denen neue Einträge auftreten, während automatisiert generierte Blogs entweder in festen Zeitintervallen wie festen Tageszeiten oder rund um die Uhr aktualisiert werden. Diese Beobachtung ist einer mehrerer Bausteine zur Erkennung von Spam Blogs, Vorhersagen zu Zeitpunkten neuer Einträge sind nicht Fokus ihrer Arbeit.

2.7.2. Modifikation der Feed-Techniken

Mediator-Systeme, die zwischen den Anbietern und Nutzern von Feeds agieren, werden u. a. in [RMP+07, RPS06, CH05] vorgestellt. Nutzer registrieren sich bei einem der Systeme und bekommen Aktualisierungen, sobald dem Mediator neue Einträge der abonnierten

Feeds vorliegen. Das Konzept transferiert das Problem des geeigneten Abfrageintervalls von der Nutzerseite hin zum Mediator, löst es jedoch nicht. Abbildung 2.12 zeigt die Architektur des Systems *Cobra* (Content-Based RSS Aggregator) [RMP+07], in dem eine Reihe von Crawlern nach neuen Feeds suchen und dem System bekannte Feeds regelmäßig auf Aktualisierungen überprüfen, und geänderte Artikel über die Ebenen der Filter und Reflektoren an die Nutzer weiterleiten. Nutzer erhalten nur die Feeds, deren Inhalt für sie potentiell relevant ist. Um die durch wiederholte Abfragen entstehende Last zu reduzieren, schlagen die Autoren mehrere Methoden vor. Zunächst erfolgt eine Auswertung der HTTP-Header Last-Modified und ETag zur Überprüfung, ob der Feed seit dem letzten Check verändert wurde. Als zweites verwenden sie die HTTP *Delta-Kodierung* [MKD+02], räumen jedoch ein, dass diese von *keinem* der rund 100.000 Feeds des verwendeten Syndic8 Datensatzes unterstützt wird. Beide Techniken werden zur Reduktion des Ressourcenverbrauchs beim Zugriff der Crawler auf die Feeds eingesetzt. Um Einsparungen zwischen den Crawlern und Filtern zu erreichen, werden nur neue Feed-Einträge übertragen. Zur Identifikation neuer Einträge wird ein Hash-Wert des kompletten Eintrags gebildet. Eine feinere Unterscheidung von Einträgen wird nicht getroffen, sodass z. B. doppelte Einträge mit zur Laufzeit generierten Zeitstempeln nicht als Duplikate erkannt werden. Der Einsatz intelligenter, adaptiver Algorithmen zur Ermittlung des Aktualisierungsintervalls eines jeden Feeds wird als weitere Möglichkeit zur Reduktion des Ressourcenverbrauchs vorgeschlagen, jedoch nicht weiter verfolgt.

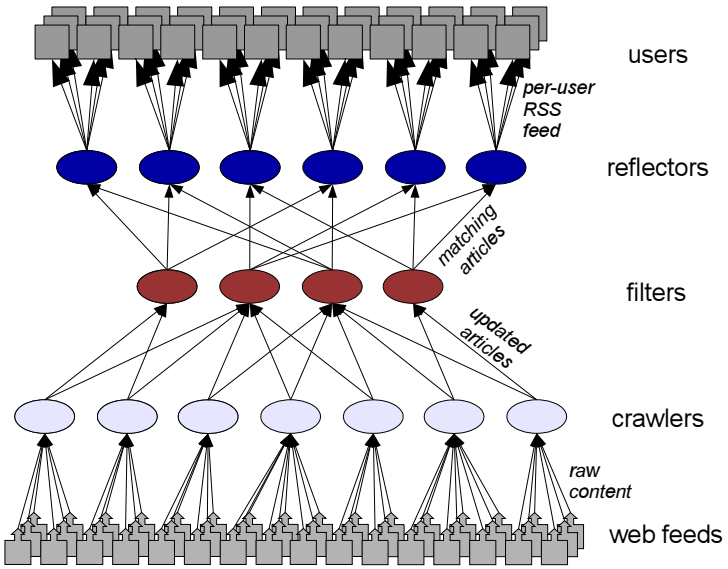

Abbildung 2.12.: Architektur des Cobra-Systems [RMP+07].

Das System *FeedTree* geht noch einen Schritt weiter, indem Feed-Anbieter Teil des vorgeschlagenen P2P-Systems zur Verteilung von Feeds werden [SMPD05]. Das Zugriffsparadigma wird von pull zu push geändert. Die Feed-Formate werden zudem um weitere Elemente ergänzt, wodurch es den Autoren gelingt, statt dem kompletten Dokument aus-

schließlich neue Einträge im P2P-System zu verteilen. Das FeedTree-System agiert zudem als Archiv, sodass Nutzer keine Einträge verpassen können, wenn sie zwischenzeitlich offline sind.

Im Rahmen des *PubSubHubbub*-Projekts [Pub11] wird ein auf Web Hooks basierendes Publish-Subscribe-System vorgestellt. Das System kann sowohl für Atom als auch RSS Feeds eingesetzt werden. In den Feeds wird ein zusätzliches Link-Element hinzugefügt, das auf den Hub verweist, an dem sich Nutzer registrieren können, um über Änderungen des Feeds informiert zu werden. Eine in Abschnitt 3.7.7 nachgereichte Untersuchung des Datensatzes ergab, dass nur knapp 7 % der Feeds des eigenen Datensatzes das PubSubHubbub-Protokoll unterstützen.

Für die eigene Arbeit wird zusammengefasst, dass das PubSubHubbub-Protokoll für den Einsatz in realen Systemen geeignet scheint, die in Kapitel 4 vorgestellten Techniken zur Vorhersage zu ergänzen, es kann sie aufgrund der geringen Verbreitung jedoch keinesfalls ersetzen. Die Auswertung der HTTP-Header wurde bereits in Abschnitt 2.5 als sinnvoll erachtet, sodass bedingte GET-Abfragen gestellt werden und Feeds nur dann übertragen werden, wenn sie geändert wurden. Der Einsatz der HTTP Delta-Kodierung [MKD+02] kann in diesen Fällen das Volumen der zu übertragenden Daten reduzieren. Das Problem der Anzahl der wiederholten Abfragen kann hierdurch jedoch nicht gelöst werden.

2.7.3. Alternative Berechnung der PostRate

Die Modellierung zyklischer Änderungsraten von Objekten wurde bereits in der Domäne Web Caching in Abschnitt 2.5.4 eingeführt. Im speziellen Kontext von Feeds findet u. a. in [LIHZ08, LH09] eine Betrachtung anhand recht selten (z. B. täglich) aktualisierter Blogs statt. Die Autoren beschränken sich implizit auf das RSS-Format und greifen auf das pubDate-Element zu. Sie treffen die Annahme, dass von (einzelnen) Personen erstellte Blogs regelmäßige Muster wie Wochentage aufweisen, die sich über die Zeit langsam ändern können. Für einen Feed wird pro Wochentag binär festgestellt, ob es an diesem Tag eine Aktualisierung gab oder nicht und der entsprechende Wert im oberen Teil von Tabelle 2.2 vorgehalten. Der Feed wurde bisher an 25 von 45 Tagen aktualisiert.

Samstag	Sonntag	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
1	0	0	0	1	1	0
1	0	0	0	1	0	1
1	1	0	1	0	1	1
1	1	1	1	1	0	0
1	1	1	0	1	0	0
0	1	0	0	1	0	0
1	1	1				
6/7	5/7	3/7	2/6	5/6	2/6	2/6

Tabelle 2.2.: Historie des Feeds nach [LIHZ08].

Die Wahrscheinlichkeit P_t, dass an einem Wochentag t ein neuer Eintrag erscheint, berechnen sie mit Gleichung 2.11. β_t ist die bisher beobachtete Wahrscheinlichkeit, dass an dem vorherzusagenden Wochentag eine Aktualisierung des Feeds stattfand, der Wert

ist in der untersten Zeile der Tabelle enthalten. δ_u ist die Anzahl aller Tage, an denen bisher eine Aktualisierung beobachtet werden konnte und δ_g ist die Summe aller Tage des Beobachtungszeitraumes. Für das Gewicht α legen die Autoren einen Wert von 0,9 fest, ohne dies näher zu erläutern.

$$P_t = \alpha\beta_t + (1-\alpha)\frac{\delta_u}{\delta_g} \tag{2.11}$$

Die Berechnung sei an einem Beispiel erläutert. Es ist Montag und der Algorithmus soll anhand der Werte in Tabelle 2.2 bestimmen, ob der Feed am darauffolgenden Dienstag abzufragen ist (leeres Feld in Tabelle). Nach Gleichung 2.12 ergibt sich $P_t = 0{,}36$. Dies liegt unter dem von den Autoren festgelegten Schwellwert von 0,5, sodass entschieden wird, den Feed nicht zu überprüfen und den Wert 0 für diesen Dienstag einzutragen. Für den nachfolgenden Mittwoch ergibt sich $P_t = 0{,}8$ und der Feed wird auf Aktualisierungen überprüft.

$$P_t = 0{,}9 \cdot \frac{2}{6} + 0{,}1 \cdot \frac{25}{45} = 0{,}36 \tag{2.12}$$

Wenngleich der Ansatz interessant ist, so bleiben leider viele Fragen ungeklärt: Wie bestimmten die Autoren den Parameter α? Wie bestimmen sie den Schwellwert für eine Überprüfung? Wird die Tabelle auch rückwirkend aktualisiert, wenn am Mittwoch festgestellt wird, dass am Dienstag eine Aktualisierung stattfand? Diese Fragen werden auch in der darauffolgenden Arbeit der Autoren nicht beantwortet, stattdessen zeigt eine Evaluation an einem Datensatz von 12.000 Feeds, deren Aktualisierungen über knapp vier Wochen aufgezeichnet wurden, dass mit den gewählten Werten 76 % der Vorhersagen korrekt waren [LH09]. Bei späteren Vergleichen dieses Wertes mit eigenen Ergbnissen muss beachtet werden, dass unklar ist, ob der Datensatz in eine Trainingsmenge und einen Teil zur Evaluation unterteilt wurde, oder ob die Parameter auf den zur Evaluation eingesetzten Datensatz hin optimiert wurden.

2.7.4. Betrachtung begrenzter Ressourcen

Die optimale Verteilung begrenzter Ressourcen wird u. a. in [SCC07, SCH+07, HLKK08] betrachtet. Die Ansätze sind orthogonal zur eigenen Arbeit, da sie ein festes Kontingent an Ressourcen veranschlagen, welche möglichst optimal in Hinblick auf eine eingesetzte Metrik über ein Zeitintervall auf die zu überwachenden Feeds verteilt werden. Dennoch sind den Methoden vielversprechende Anleihen für die eigene Arbeit zu entnehmen.

Die zum Teil aus [CG99, CG00, CG03a, CG03b] bekannten Autoren grenzen sich in [SCC07, SCH+07] von dem von ihnen in der Domäne Web Crawling vorgeschlagenen homogenen Poisson-Modell ab und verwenden ein inhomogenes Poisson-Modell. Im Gegensatz zur Aktualisierung eines Index, bei der längere Abfrageintervalle wie ein Monat für eine Vielzahl der Seiten vertretbar sind, müssen Feeds mehrfach täglich abgefragt werden, sodass eine feingranulare Betrachtung der Änderungsrate lohnenswert ist [SCC07]. Das Modell mit eintägiger Periodendauer T und einer Auflösung von einer Stunde wird über zwei Wochen trainiert.

Mit der in [SCC07] vorgestellten Metrik *Delay* kann für jeden Eintrag E, Feed F oder einen kompletten Datensatz A von Feeds die Verzögerung zwischen dem Zeitpunkt t_i der Veröffentlichung von E_i und der nächsten Abfrage τ_j des Feeds ermittelt werden. Dies ist in den Gleichungen 2.13–2.15 zusammengefasst, wobei in 2.15 jeder Feed unterschiedlich gewichtet werden kann. Der Delay stellt somit eine Erweiterung der Metrik Age aus den Gleichungen 2.5 und 2.6 (Seite 40) dar [CG00, SCC07] und wird zur Evaluation in Kapitel 4 als Basis einer eigenen Metrik verwendet.

$$D(E_i) = \tau_j - t_i \quad \text{mit } t_i \in [\tau_{j-1}, \tau_j] \tag{2.13}$$

$$D(F) = \sum_{i=1}^{k} D(E_i) \tag{2.14}$$

$$D(A) = \sum_{i=1}^{n} w_i D(F_i) \tag{2.15}$$

Ziel der zweistufigen Optimierungen in [SCC07] ist die Minimierung des anhand des inhomogenen Poisson-Modells vorhergesagten Wertes $D(A)$. In einer als *Retrieval Scheduling* bezeichneten ersten Stufe werden die in T zur Verfügung stehenden Ressourcen anhand der individuellen Änderungsraten auf die Feeds verteilt. In einem zweiten Schritt, der *Resource Allocation* werden die Abfragezeitpunkte τ_j innerhalb von T bestimmt, sodass in Abschnitten höherer Änderungsraten in kürzeren Intervallen abgefragt wird als in Abschnitten mit geringerer Aktivität des Feeds. Die Evaluation dieses im Folgenden als minDelay bezeichneten Algorithmus ergibt einen durchnittlich 35 % geringeren Delay im Vergleich zu [CG03b]. minDelay hat generell einen geringeren Delay als ein gleichmäßiger Algorithmus, der die Ressourcen gleichmäßig auf A und T verteilt. Im direkten Vergleich der beiden Optimierungsschritte Resource Allocation und Retrieval Scheduling (wenn nur einer angewandt und der andere durch einen gleichmäßigen Ansatz ersetzt wird) erzielt die Resource Allocation generell einen geringeren Delay. Als potentiellen Nachteil identifizieren die Autoren jedoch, dass aufgrund der ungleichmäßigen Verteilung der Ressourcen beim Retrieval Scheduling einige Feeds benachteiligt werden. Dies führt zu sehr hohen Werten des maximalen Delay. Als Ausweg schlagen sie vor, kein Retrieval Scheduling durchzuführen, wenn der maximale Delay begrenzt werden soll [SCC07]. Bereits vorgestellte Verfahren wie AdaptiveTTL lösen das Problem durch Einführen einer Schranke (Gleichung 2.10 Seite 53).

Für die in Kapitel 4 durchzuführenden Evaluationen bleibt festzuhalten, dass neben dem durchschnittlichen Delay auch dessen Verteilung mit betrachtet werden sollte. Die Betrachtung begrenzter Ressourcen ist wichtig, der in [SCC07] aufgeführte Ansatz hat jedoch Grenzen: Die Autoren treffen die Annahme, dass die bereitgestellten Ressourcen zu beliebigen Zeitpunkten innerhalb T (im Beispiel ein Tag) eingesetzt werden können. Im ungünstigsten Fall, wenn alle Feeds zum selben Zeitpunkt oder innerhalb einer sehr kleinen Zeitspanne abgefragt werden sollen, ist dies nicht umsetzbar.

Insgesamt weicht der Ansatz des Algorithmus minDelay von den eigenen Zielen ab, da hier kein zur Verfügung stehendes Kontingent an Ressourcen „aufgebraucht“ werden soll, sodass der Algorithmus nicht direkt übertragbar und ein fairer Vergleich in Kapitel 4 nicht möglich ist. Selbiges gilt für die in [SCH+07] vorgestellte Erweiterung, in der zusätzlich das Zugriffsverhalten von Nutzern in die Optimierung aufgenommen wird. Ungeachtet dessen stellt das vorgestellte Clustering ähnlicher Feeds einen interessanten Ansatz dar, die Anzahl der benötigten, aufwendigen Berechnungen aus [SCC07] zu minimieren. Leider werden weder die Wahl des Verfahrens (k-means) noch die der Parameter eruiert [SCH+07]. Die aufgestellte Definition eines *Miss* weicht von der Definition in dieser Arbeit ab. Ein Eintrag wird genau dann als Miss bezeichnet, wenn er bereits auf dem Server veröffentlicht, jedoch noch nicht abgerufen ist, sodass er einem Nutzer beim Zugriff auf dessen lokale Kopie des Feeds nicht angezeigt wird [SCH+07]. Für die eigene Arbeit wird die Definition eines Miss aus [HLKK08] übernommen. Als verpasste Einträge – kurz Miss – werden alle Einträge bezeichnet, die zwischen zwei aufeinander folgenden Abfragen verpasst wurden. Angenommen ein Feed hat eine Fenstergröße von zehn Einträgen und zwischen zwei Abfragen wurden zwölf neue Einträge veröffentlicht, dann werden zwei Einträge verpasst. Die Definition eines Misses wird später in Kapitel 3.5.2 verfeinert.

Fokus des im Folgenden als minMiss bezeichneten Algorithmus ist die Minimierung verpasster Einträge [HLKK08]. In Ergänzung zu Verfahren, die (ausschließlich) auf der Änderungsrate eines Feeds beruhen, berücksichtigen die Autoren zusätzlich die Fenstergröße des Feeds. Basierend auf der individuellen Änderungsrate, Fenstergröße sowie einem global zur Verfügung stehenden Kontingent an Abfragen teilt minMiss jedem Feed ein individuelles Kontingent an Abfragen zu. Es ist anzunehmen, dass das individuelle Kontingent gleichmäßig über das betrachtete Intervall T verteilt wird und jeder Feed mit einer individuellen, jedoch statischen Frequenz angefragt wird. Die Autoren äußern sich hierzu jedoch nicht konkret. Ihr Resümee ist, dass minMiss im Vergleich zu minDelay (aus [SCC07]) 23 % weniger Einträge verpasst, wobei die Verzögerung im Schnitt um 6 % zunimmt. Die Trainings- und Testphase betrug jeweils drei Wochen. Aussagen zu variierenden Fenstergrößen werden nicht getroffen, zudem wird erwähnt, dass das Modell regelmäßig zu trainieren sei, um Änderungen über die Zeit zu berücksichtigen. Aufgrund der a priori zu definierenden Ressourcen ist der Ansatz ebenfalls nicht auf die eigene Arbeit übertragbar. Er kann keinem direkten Vergleich unterzogen werden, da die in Kapitel 4 betrachteten Algorithmen selbstbestimmt jeweils unterschiedlich viele Ressourcen allokieren.

2.7.5. Hybride Vorhersage zur Überbrückung einer Trainingsphase

Das System *advaRSS* ist ein hybrider Ansatz, um die Abfrageintervalle für neue Feeds sofort adaptiv zu bestimmen und parallel das Modell eines PostRate-Algorithmus zu trainieren. Nach Abschluss der Trainingsphase wird der PostRate-Algorithmus zur Vorhersage neuer Änderungen eingesetzt [ABP10].

Die Grundidee des adaptiven Algorithmus der Trainingsphase ist, das Abfrageintervall langsam und stetig zu vergrößern, wenn bei einer Abfrage keine Änderung festgestellt wird, es jedoch umgehend stark zu verkürzen, wenn eine Änderung beobachtet wird. Wird

keine Änderung beobachtet, so wird das neue Intervall in etwa auf das Anderthalbfache des vorangegangenen gesetzt, im Falle einer Änderung erfolgt eine Reduktion auf 20 % des vorangegangenen Intervalls. Zudem verwenden die Autoren die Schranken 10 und 800 Minuten als untere, respektive obere Schranke, um nicht zu häufig oder zu selten abzufragen. Die Trainingsphase umfasst mindestens 24 Stunden und wird anschließend so lange fortgesetzt, bis mindestens ein neuer Eintrag gefunden wurde.

Zur Erkennung von neuen Einträgen setzen die Autoren ein zweistufiges Verfahren ein, indem sie für jeden Feed den hash-Wert der jeweils letzten bekannten Version der XML-Datei vorhalten und mit dem einer neu heruntergeladenen Version vergleichen. Unterscheiden sich die Werte, so werden im zweiten Schritt neue Einträge ausschließlich anhand ihrer Titel identifiziert. Interessanterweise setzen die Autoren dieses Verfahren nur zur internen Weiterverarbeitung der Feeds ein, zur Bestimmung des Intervalls erfolgt ausschließlich eine binäre Betrachtung der Änderung, sodass nicht unterschieden wird, ob der Feed einen oder mehrere neue Einträge aufweist – dies ist verwunderlich, intuitiv ist zu erwarten, dass Intervall in Abhängigkeit zur Anzahl erfolgter Änderungen zu wählen.

Das trainierte Modell der Änderungsrate des Feeds weist Ähnlichkeiten mit dem in Abschnitt 2.5.4 vorgestellten Modell aus [GE01, BGR06] auf: Das Modell umfasst einen Tag und für jede Stunde wird die durchschnittliche Änderungsrate ermittelt. Die Vorhersage der bis zu einem Zeitpunkt t erwarteten Änderungen erfolgt ebenfalls analog. Im Unterschied zu [BGR06] umfasst das Modell in [ABP10] jedoch ausschließlich die letzten 24 Stunden und wird daher fortlaufend nach jeder Abfrage aktualisiert. Zur Berücksichtigung begrenzter Ressourcen ermitteln die Autoren für alle Feeds zunächst die Anzahl erwarteter, neuer Einträge, ordnen die Feeds absteigend nach dieser Anzahl und fragen anschließend nur die ersten k Feeds ab. Die Sortierung kann zudem durch die Anzahl der Nutzer der Feeds beeinflusst werden, sodass bei zwei Feeds mit gleicher Anzahl erwarteter Änderungen der Feed mit einer größeren Leserzahl höher gewichtet wird.

Obwohl der Ansatz und die erzielten Evaluationsergebnisse verheißungsvoll sind, werden leider einige wichtige Aspekte nicht beschrieben, sodass es nicht möglich ist, den Algorithmus anhand [ABP10] zu implementieren und mit anderen Verfahren zu vergleichen. Die Autoren geben keinen Wert für die Anzahl k pro Zyklus abgefragter Feeds an. Wurde für einen Tag keine Änderung beobachtet, so enthält das Modell ausschließlich Änderungsraten von Null. In diesem Fall verwenden die Autoren einen alternativen Wert, ohne diesen zu spezifizieren. In der Evaluation erreichen die Autoren einen zunächst beachtlich guten Wert von 14 Minuten für die durchschnittliche Verzögerung, mit der neue Einträge gefunden werden. Ohne Angaben zur Zusammensetzung des Datensatzes und der getätigten Anzahl Abfragen ist dieser Wert jedoch nicht aussagekräftig, sodass es nicht möglich ist, die nicht spezifizierten Parameter auf Basis der beschriebenen Evaluationsergebnisse zu schätzen. Eine direkte Nachfrage bei den Autoren blieb leider unbeantwortet.

2.7.6. Zusammenfassung der Systeme zur Verarbeitung von Feeds

Aus den Betrachtungen der Domäne Feeds wird geschlussfolgert, dass Methoden zur Aktualisierung von Webseiten nicht eins zu eins auf Feeds übertragbar sind [SMPD05, SCC07, SCH+07]. Die in [LRS05, SCC07] erarbeiteten Statistiken und Betrachtungen

der Änderungsraten von Feeds werden in den Abschnitten 3.7.1 und 3.7.3 mit den Charakteristiken des eigenen Datensatzes verglichen. Die untersuchten Ansätze zur Modifikation der Feed-Techniken stellen Lösungsansätze vor, die auf einer aktiven Beteiligung der Anbieter basieren und daher als nicht global realisierbar eingeschätzt werden [RMP+07, RPS06, CH05]. Die aussichtsreichste Alternative ist das PubSubHubbub-Protokoll [Pub11], dass jedoch nur von knapp 7 % der Feeds des Datensatzes unterstützt wird und daher in realistischen Szenarien mit Techniken zum adaptiven Abfragen ergänzt werden sollte.

Die in [LIHZ08, LH09] vorgestellte alternative Variante des PostRate-Algorithmus wird in Kapitel 4 aufgegriffen und mit den übrigen Techniken verglichen. Aus den Betrachtungen begrenzter Ressourcen [SCC07, SCH+07, HLKK08] gehen die Definitionen Delay und Miss hervor, die für die eigene Arbeit übernommen werden. Der Delay eines Eintrags ist die Zeitdauer zwischen der Veröffentlichung des Eintrags und der darauffolgenden Abfrage des Feeds. Ein Miss liegt vor, wenn der Konsument einen Eintrag verpasst, da er bereits aus dem Fenster verdrängt wurde, bevor der Feed abgefragt wird.

2.8. Zusammenfassung des Stands der Technik

Zur Abrundung des Kapitels werden die wichtigsten Resultate rekapituliert. Im Fokus der Betrachtungen standen die Modellierung und Bestimmung von Änderungsraten, Algorithmen zur Vorhersage neuer Einträge sowie Eigenschaften der Feed-Formate und Verfahren aus dem Web-Umfeld.

Aktuelle PIS sind nur bedingt zur Integration von Feeds geeignet, da die eingesetzten Techniken aus der Domäne Datenbanken entweder auf einer aktiven Beteiligung der Quellen (push) oder statischen, periodischen Abfrageintervallen basieren.

Die Analysen der Feed Formate und des zur Übertragung eingesetzten HTTP ergaben, dass diese einige Attribute bereitstellen, die zur Optimierung der Abfragen eingesetzt werden können. Für HTTP-Header wie Expires wurde jedoch festgestellt, dass diese von den Servern bereitgestellten Werte nicht zuverlässig sind. Für Feed-Attribute konnten keine Arbeiten zur Analyse feedspezifischer Attribute gefunden werden, sodass in Kapitel 5 eigene Studien erfolgen. Datumsangaben in RSS-Feeds dürfen in der Zukunft liegen, sodass sie nicht immer das tatsächliche Veröffentlichungsdatum beinhalten, weshalb eine Vorverarbeitung zur Korrektur als sinnvoll erachtet und in Abschnitt 3.5.5 umgesetzt wird. Zur Reduzierung des übertragenen Datenvolumens eignen sich bedingte GET-Abfragen, um einen Feed nur dann neu zu übertragen, wenn eine Änderung vorliegt. Um darüber hinaus keine als stale bezeichneten, veralteten Objekte aus Web Caches zu erhalten, kann das Cache-Control Attribut eingesetzt werden, um generell eine frische Kopie von der Quelle zu erlangen. Diese beiden auf dem HTTP-Header basierenden Mechanismen werden ebenfalls in Kapitel 3 aufgegriffen und umgesetzt. Alternative push-Techniken der Server, mit denen Nutzer über neue Inhalte informiert werden, sind aussichtsreiche Entwicklungen, werden jedoch derzeit nur selten realisiert. Sollte ihre Verbreitung in Zukunft stark zunehmen, sollten sie auf ihre Eignung hin untersucht werden.

Untersuchungen der Änderungsraten von Webseiten und Feeds werden in Abschnitt 3.7

aufgegriffen und mit Statistiken über den eigenen Feed Datensatz verglichen.

Die Modellierung der Änderungsraten basiert sehr häufig auf einem Poisson-Prozess. Hauptunterscheidungsmerkmal dieser Arbeiten ist, ob ein homogenes oder inhomogenes Modell zugrunde gelegt wird, um sich wiederholende Intervalle feingranular zu betrachten und z. B. unterschiedliche Frequenzen für Tages- und Nachtzeiten modellieren zu können. Die Schwierigkeit besteht dabei in der Wahl eines geeigneten Trainings. Wird ein zu kurzer Abschnitt der Historie zum Trainieren verwendet, so kann das Modell das tatsächliche Verhalten nicht repräsentieren, im Gegenzug birgt eine lange Trainingsdauer die Gefahr, nicht adaptiv auf Änderungen über die Zeit reagieren zu können. Einige Autoren schlagen daher vor, das Modell regelmäßig neu zu trainieren. Das azyklische Verhalten eines Objekts wird auch als burst bezeichnet und sollte nicht mit in das Modell einfließen, sondern von den Vorhersageagorithmen gesondert behandelt werden. Können Änderungen von Objekten nur binär festgestellt werden, so kann nicht entschieden werden, ob zwischenzeitlich Einträge verpasst wurden. Zur Ermittlung der Änderungsrate wurden daher in Abschnitt 2.4.3 spezielle Verfahren vorgestellt, deren Eignung im Hinblick auf die eigenen Ziele in Kapitel 3 untersucht wird.

Die Definitionen Freshness, Age, Delay und Miss bilden die Grundlagen für die eigene Arbeit. Sie werden in Abschnitt 4.3.3 erweitert und in den Kapiteln 4 und 5 zur Bewertung der Algorithmen eingesetzt. Wichtige Vertreter der Algorithmen zur Vorhersage neuer Einträge sind ein universeller Algorithmus, FixLearnd, PostRate und Moving Average. Darüber hinaus wurden Kombinationen von Algorithmen vorgestellt, um bursts und kurze Historien separat zu behandeln.

Fix Ein universeller Algorithmus ist der einfachste Ansatz. Alle Quellen werden mit der gleichen Frequenz abgefragt. Dieser Ansatz wird in der Literatur häufig als *Baseline* verwendet. Universelle Algorithmen werden in Kapitel 4 unter der Bezeichnung *Fix* aufgegriffen, wobei Fix 1h und Fix 1d Abfrageintervalle von einer Stunde respektive einem Tag haben.

Fix Learned ist die Erweiterung des universellen Algorithmus. Wird die Änderungsrate als Skalar modelliert und z. B. initial auf Basis des ersten beobachteten Fensters eines Feeds gelernt, so wird der Algorithmus als FixLearned bezeichnet.

PostRate In die Kategorie der PostRate-Algorithmen fallen alle Techniken, die auf einem inhomogenen Poisson-Modell basieren und die Änderungsrate in Abhängigkeit der Tageszeit modellieren. Dabei kann unterschieden werden, ob die Anzahl erfolgter Änderungen oder nur die binäre Entscheidung, ob überhaupt eine Änderung in einem Zeitintervall vorlag, erfasst werden. Vertreter der PostRate-Algorithmen sind IndHist [BGR06] und der in [LIHZ08] vorgestellte Ansatz.

Moving Average ist eine Erweiterung des FixLearned Algorithmus, bei der das Modell bei jedem neuen Eintrag auf Basis einer begrenzte Historie trainiert wird. AdaptiveTTL [CL98, RS02, INST04] aus der Domäne Web Caching kann als Spezialfall des Moving Average angesehen werden, wobei ausschließlich der Zeitpunkt der letzten Änderung in die Vorhersage eingeht.

Die betrachteten Algorithmen werden in Kapitel 4 aufgegriffen und im Kontext der Integration von Feeds in ein PIS miteinander verglichen. Um einen direkten Vergleich zu ermöglichen, wird ein geeigneter Datensatz benötigt. Die Gewinnung dieses Datensatzes wird im folgenden Kapitel 3 vorgestellt.

3. Gewinnung eines umfangreichen, realen Feed Datensatzes

Das Fundament dieser Arbeit bildet ein umfangreicher, realer Datensatz von 200.000 Feeds, deren Einträge über einen Zeitraum von vier Wochen nahezu lückenlos aufgezeichnet wurden. Zunächst werden die Anforderungen an den Datensatz formuliert und verfügbare Datensätze anhand dieser untersucht. Im Anschluss wird der Datensatz in zwei Schritten erstellt. Das Kapitel schließt mit einer Analyse der gewonnenen Daten, in welcher u. a. die Aktualisierungsmuster der Feeds und die Qualität des Datensatzes betrachtet werden.

Wenngleich bei der Gewinnung des Datensatzes darauf geachtet wurde, dass er so breit wie möglich aufgestellt ist, so fokussieren die aufgeführten Statistiken dennoch auf das Kernthema der Arbeit: die Vorhersage von Aktualisierungen.

Als roter Faden sei bereits das Grundprinzip zur Erstellung des Datensatzes eingeführt. In einem ersten Schritt wird eine Menge von Feed URLs gesammelt, welche im zweiten Schritt über einen Zeitraum von vier Wochen wiederholt abgefragt werden. Es wird versucht, jeden Feed so oft abzufragen, wie er Aktualisierungen aufweist und diese zu persistieren. Um festzustellen, ob ein Feed neue Einträge enthält, werden die Einträge der aktuellen Abfrage mit denen der vorangegangenen verglichen. Hierbei sind zwei Schwierigkeiten zu bewältigen: Es muss sichergestellt werden, dass der Feed häufig genug anfragt wird, um keine Einträge zu verpassen und Lücken im Datensatz zu vermeiden. Darüber hinaus müssen Duplikate, d. h. bereits empfangene Einträge, eindeutig identifiziert werden, um die selben Einträge nicht mehrfach im Datensatz zu speichern.

3.1. Anforderungen an den Datensatz

Die Grundlage der Untersuchung verfügbarer Datensätze sowie der Erstellung des eigenen Datensatzes bilden die Anforderungen, denen der in der Arbeit benötigte Datensatz genügen muss. Ziel ist es, einen *realen, d. h. nicht-synthetischen*, *breitgefächerten*, *vollständigen* und nicht zuletzt *nachnutzbaren* Datensatz zu verwenden. Die Kriterien werden im Einzelnen wie folgt definiert:

Real ist ein Datensatz, dessen Datenbestand sich aus Feeds tatsächlich existierender Quellen speist und nicht mittels einer Simulation synthetisch erstellt wird.

Breitgefächert Der Datensatz sollte möglichst alle in der Realität auftretenden Aktualisierungsmuster abdecken und nicht auf einzelne Domänen wie sehr häufig aktualisierte Webseiten von Nachrichtenagenturen oder selten überarbeitete, persönliche Blogs beschränkt sein. Sollte kein passender Datensatz in der Domäne

Produktinformationen verfügbar sein, so wird für den zu erstellenden Datensatz das Ziel formuliert, dass dieser Domänen-unabhängig sein soll. Wenngleich die vorliegende Arbeit in der Domäne Produktinformationen angesiedelt ist, hat diese „Einschränkung“ zwei entscheidende Vorteile:

Zur Erstellung eines Datensatzes für die Domäne Produktinformationen müsste sichergestellt werden, dass alle aufgenommenen Feeds tatsächlich Produktinformationen nach der erweiterten Definition aus Kapitel 2 enthalten. Hierfür wäre entweder eine inhaltsbasierte Analyse von Feeds nötig, welche kein Bestandteil der Arbeit ist, oder eine Art zentrales Register, welches „alle“ Feeds enthält, die der Domäne Produktinformationen zuzuordnen sind – ein solches Register ist nicht bekannt. Aus diesem Grund wird die Annahme getroffen, dass Feeds der Domäne Produktinformationen ähnliche Aktualisierungsmuster aufweisen, wie Feeds anderer Domänen, sodass ein breitgefächerter, Domänen-unabhängiger Datensatz gut geeignet ist, um die im weiteren Verlauf entwickelten Konzepte zu evaluieren.

Der zweite Vorteil dieser Anforderung ist, dass der Datensatz potentiell auch von anderen Wissenschaftlern genutzt werden kann, um typische Aufgaben des Information Retrievals daran zu evaluieren. Aus diesem Grund soll auch keine sprachabhängige Selektion der Feeds vorgenommen werden.

Vollständig Die Forderung nach der Vollständigkeit bezieht sich auf die (zu persistierenden) Aktualisierungen der Feed-Einträge. Um der Anforderung zu genügen, müssen zu den Feeds lückenlose Historien der Einträge vorliegen. Der Datensatz soll zudem einen Ausschnitt der existierenden Feeds beinhalten, aber – selbstverständlich – keinesfalls alle Feeds.

Die Betrachtungen in den Abschnitten 2.1.2 und 2.3.1 haben gezeigt, dass einige Feeds ein sehr hohes Aktualisierungsintervall aufweisen oder gar alle Einträge dynamisch zur Abfragezeit generieren. Aus diesem Grund kann es vorkommen, dass der Datensatz Lücken enthält, die erkannt und entsprechend gekennzeichnet werden müssen. Die Definition der Vollständigkeit wird daher erweitert, sodass der Datensatz vollständig ist, wenn die Historie der Feed-Einträge lückenlos aufgezeichnet wurde oder dies erkennbar nicht der Fall ist.

Nachnutzbar Ein Datensatz ist nachnutzbar, wenn die Daten in einem gebräuchlichen Format gespeichert, öffentlich zugänglich sowie gut dokumentiert sind. Ein sorgfältig gewonnener Datensatz sollte nicht nur der eigenen Arbeit dienlich sein, sondern auch Anderen zur Verfügung gestellt werden.

3.2. Verfügbare Datensätze

Anhand der aufgestellten Anforderungen werden frei verfügbare, bekannte, reine Feed-Datensätze sowie einige Sammlungen im Bereich des Web auf ihre Eignung hin untersucht und bewertet. In Kapitel 2 bereits aufgeführte Datensätze, die nicht frei zugänglich sind, werden nicht erneut betrachtet.

3.2.1. Datensätze im Feed-Umfeld

Nach bestem Wissen sind nur drei Datensätze im Bereich Feeds verfügbar, von denen keiner jede der in Abschnitt 3.1 aufgestellten Anforderungen nach einem realen, breitgefächerten, vollständigen und nachnutzbaren Datensatz unterstützt.

Der frei verfügbare Syndic8 Datensatz umfasst laut Angaben der Webseite mehr als 600.000 Feeds, die zur Verfügung gestellte Liste[1] enthält jedoch nur ca. 100.000 URLs. Bei Stichproben stellte sich heraus, dass viele dieser Feeds nicht mehr verfügbar sind. Generell werden die Feeds manuell von Nutzern hinzugefügt, sodass sie nicht breitgefächert sind. Der Datensatz beinhaltet keine Aufzeichnung von Historien, ein Teil der Feeds wurde ein bis vier mal täglich auf Erreichbarkeit überprüft, seit Januar 2010 scheint der Betreiber der Seite diese Aktivitäten eingestellt zu haben. Der Datensatz wurde u. a. von [LRS05] zur Erstellung allgemeiner Statistiken verwendet, die Autoren in [RMP+07] verwendeten den Datensatz, um ihr Feed Aggregationssystem Cobra zu testen.

Datensätze mit Feed-Einträgen werden im Rahmen der Text Retrieval Conference (TREC) als Blogs06 [MO06, G2006] und Blogs08 [G2008] zur Verfügung gestellt. Der Fokus beider Datensätze liegt auf *web logs (blogs)*. Blogs06 enthält ca. 100.000 Feeds, die über einen Zeitraum von elf Wochen je einmal pro Woche heruntergeladen wurden. Blogs08 enthält rund 1,3 Millionen Feeds, die über eine Zeitspanne von einem Jahr ebenfalls wöchentlich heruntergeladen wurden. Beide Datensätze erfüllen nicht die Anforderungen dieser Arbeit, da sie nicht breitgefächert sind und nur Feeds mit recht seltenen Aktualisierungen beinhalten bzw. aufgrund der seltenen Abfragen zahlreiche Lücken für Feeds aufweisen, die eine hohe Änderungsrate haben.

3.2.2. Datensätze im allgemeinen Web-Umfeld

Im breiteren Web-Umfeld sind deutlich mehr Datensätze verfügbar, zu denen u. a. ClueWeb09, 20 Newsgroups und der Reuters RCV1 Korpus zählen.

Der ClueWeb09 Datensatz [CHYZ09] wird – wie auch die Blogs06 und Blogs08 Datensätze – in mehreren, im Rahmen der TREC erschienenen Arbeiten eingesetzt. Er umfasst eine Milliarde Webseiten in zehn Sprachen, die über einen Zeitraum von zwei Monaten gesammelt wurden. Der Datensatz umfasst jeweils nur eine Revision einer Webseite, weshalb er nicht für die Vorhersage von Änderungen eingesetzt werden kann.

Der 20 Newsgroups Datensatz [Lan08] enthält ungefähr 20.000 Nachrichten aus dem Jahr 1997, die gleichverteilt 20 Newsgroups zugeordnet werden können. Die 20 Newsgroups können sechs verschiedenen Kategorien wie Computer, Wissenschaft oder Religion zugeordnet werden. Der Datensatz besteht aus E-Mails, sodass aus den in den Headern enthaltenen Zeitangaben die Zeiträume zwischen aufeinander folgenden Nachrichten rekonstruierbar sind. Dennoch sprechen mehrere Argumente gegen die Verwendung in dieser Arbeit. Der 20 Newsgroups Datensatz ist aufgrund der Zuordnung zu nur sechs Kategorien nicht breitgefächert und 20.000 einzelne Nachrichten sind im Vergleich zu aktuellen Feeds recht wenig Einträge. Zudem kann aus den Mustern, die in E-Mails vor mehr als einem Jahrzehnt aufgetreten sind, nicht auf aktuelle Feeds geschlossen werden.

[1] `http://www.syndic8.com/feedlist.php`, letzter Zugriff 22.06.2011

Ein weiterer bekannter Datensatz ist der Reuters RCV1 Korpus [RSW02]. Es handelt sich um einen zusammenhängenden Ausschnitt aus dem Archiv der Nachrichtenagentur Reuters. Er hat einen Umfang von rund 800.000 Nachrichten und umfasst alle englischsprachigen Nachrichten, die im Zeitraum von einem Jahr (1996–1997) von Journalisten der Agentur Reuters erstellt wurden. Der Datensatz verfügt zudem über Annotationen, die u. a. das Themengebiet der Nachricht oder die Region (z. B. Zuordnung eines Landes) enthalten. Mit dem Reuters RCV1 Korpus sollte die regelbasierte Klassifikation von Nachrichten verbessert werden.

3.2.3. Bewertung der verfügbaren Datensätze

Die Erfüllung der in Abschnitt 3.1 aufgestellten Anforderungen durch die verfügbaren Datensätze ist in Tabelle 3.1 dargestellt. Da ausschließlich reale, nachnutzbare Datensätze betrachtet wurden, erfüllen alle diese beiden Anforderungen. Die Datensätze aus dem Feed-Umfeld sind weder breitgefächert noch enthalten sie die vollständige Historie der Feeds. Zwei der drei Datensätze aus dem Web-Umfeld sind breitgefächert, jedoch erfüllt keiner der Datensätze die Anforderung nach einer vollständigen Historie der Feeds, da sie Feeds – sofern überhaupt – nur am Rande enthalten. Um diese Lücke zu schließen, wird ein eigener Datensatz erstellt und öffentlich zur Verfügung gestellt.

Datensatz	Real	Breitgefächert	Vollständige Feed-Historie	Nachnutzbar
Syndic8 (`www.syndic8.com`)	✓	○	×	✓
Blogs06 [MO06]	✓	×	×	✓
Blogs08 [G2008]	✓	×	×	✓
ClueWeb09 [CHYZ09]	✓	✓	×	✓
20 Newsgroups [Lan08]	✓	×	×	✓
Reuters RCV1 [RSW02]	✓	✓	×	✓
		✓/ ○/ × ≙ erfüllt / teilweise erfüllt / nicht erfüllt		

Tabelle 3.1.: Erfüllung der Anforderungen durch die verfügbaren Datensätze.

3.3. Erstellung des Datensatzes im Überblick

Die Erstellung des Datensatzes erfolgt in zwei Phasen: dem Auffinden von Feed URLs und dem kontinuierlichen Aufrufen dieser URLs, um neue Einträge zu speichern. Diese Phasen sind in Abbildung 3.1 grob skizziert und werden in den Abschnitten 3.4 und 3.5 im Detail erläutert. In der Grafik fehlende Werte der Nummerierung werden in Abbildung 3.2 nachgereicht. Die zur Gewinnung des Datensatzes benötigten Grundlagen im Bereich der Feed-Formate und dem zur Übertragung verwendeten HTTP wurden bereits in Abschnitt 2.3 erläutert.

Zum Auffinden der URLs wurde zunächst eine Menge breitgefächerter Schlüsselwörter (Liste 1) erstellt und anschließend eine konventionelle Suchmaschine angefragt. Die als Suchergebnisse zurückgegebenen Webseiten (Liste 2) wurden nach Feed URLs durchsucht. Aus der Menge der Feed URLs (Liste 3) wurden alle tatsächlichen und Beinahe-Duplikate

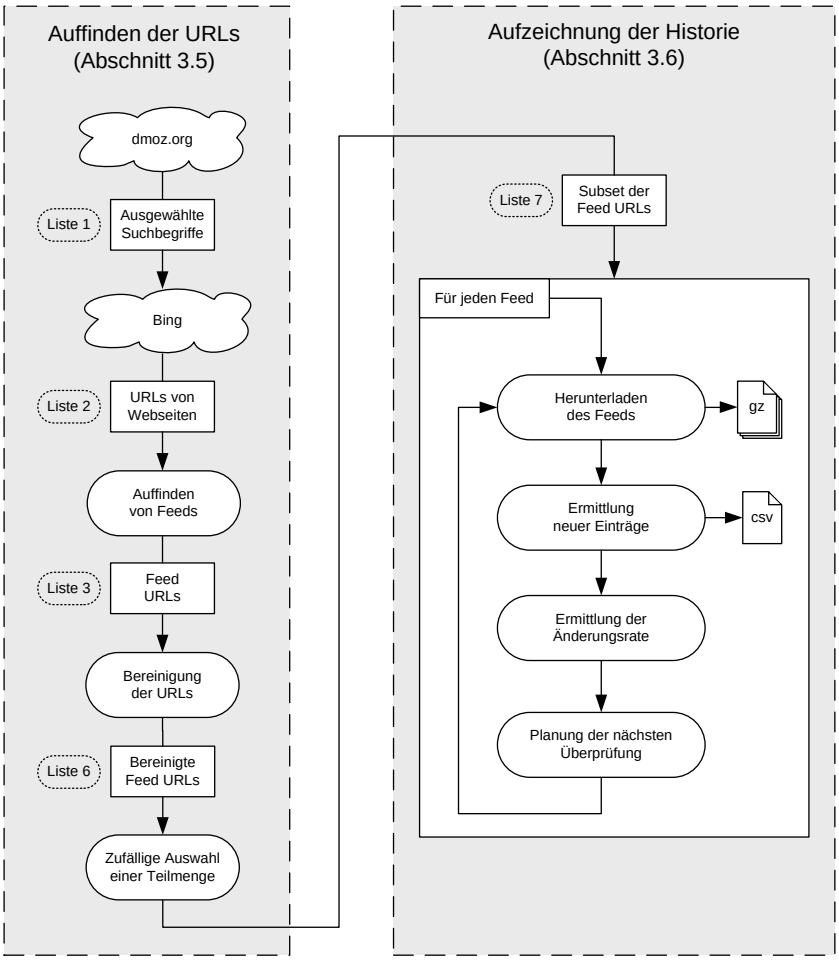

Abbildung 3.1.: Erstellung des Datensatzes im Überblick. Die Listen 4 und 5 folgen in Abbildung 3.2.

(engl. near-duplicate) entfernt (Liste 6). Die erste Phase wurde mit einer zufälligen Auswahl einer Teilmenge der Feed URLs (Liste 7) abgeschlossen, welche die Basis für die zweite Phase bildete.

Zur Aufzeichnung der Historie wurde jeder Feed regelmäßig abgerufen. In jedem Zyklus wurde überprüft, ob der Feed verändert wurde. Im Falle einer Änderung wurde das komplette XML-Dokument gemeinsam mit dem empfangenen HTTP Header in komprimierter Form gespeichert. Anschließend wurde aus den Einträgen das Aktualisierungsintervall berechnet. Enthielt der Feed neue Einträge, so wurden diese in einer eigenen Datei abgelegt, aus welcher sich der komplette Strom der Einträge rekonstruieren lässt. Abschließend wurde aus dem ermittelten Aktualisierungsintervall berechnet, wann der Feed das nächste Mal auf Änderungen zu überprüfen ist.

3.4. Auffinden von Feed URLs

Die erste Phase in der Erstellung des Datensatzes bestand aus dem Auffinden von Feed URLs. Sie ist nötig, um den Anforderungen eines realen, breitgefächerten Datensatzes gerecht zu werden (siehe Abschnitt 3.1). Als Zielgröße wurde eine Anzahl von einer Million Feed URLs festgelegt. Im Folgenden werden die Teilschritte einzeln hergeleitet und abschließend in Abschnitt 3.4.6 zusammengefasst.

3.4.1. Auswahl der Suchbegriffe

Um die Domänenunabhängigkeit des Datensatzes zu gewährleisten und nicht durch eine manuelle Auswahl von Feeds zu beeinträchtigen, wurden die Feed URLs unter der Zuhilfenahme einer Suchmaschine gesammelt. Als Ausgangspunkt wurden die Kategorien des Open Directory Projekts[2] verwendet, um eine Liste mit Suchmaschinenanfragen zu erstellen. Das Open Directory Projekt ist ein umfangreiches Verzeichnis des Web und umfasst ca. 800.000 Kategorien[3] in einer Vielzahl von Sprachen. Es wird von knapp 100.000 Nutzern gepflegt und bildet die Grundlage der Verzeichnisdienste großer Suchanbieter wie AOL Search und Google, sodass davon ausgegangen wird, dass die enthaltenen Kategorien der Anforderung eines breitgefächerten Datensatzes entsprechen.

Die Baumstruktur der Kategorien wurde „ausgerollt“ und in eine sogenannte *bag-of-words*-Repräsentation überführt. Kategorien, die nur zwei mal oder seltener in der Taxonomie enthalten sind, wurden anschließend entfernt. Auf diese Weise wurden hoch spezialisierte Kategorien, wie z. B. die Namen seltener Krankheiten, ausgeschlossen. Das Ergebnis der Filterung ist die Liste 1 (vgl. Abb. 3.1) mit ca. 57.000 Kategorien, zu denen Sportarten, Namen von Ländern, allgemeine Begriffe wie „news“ und „cold“ und nicht zuletzt Namen von Firmen und Produkten gehören.

3.4.2. Abfragen der Suchmaschine

Liste 1 der knapp 57.000 Kategorien des vorangegangenen Schrittes wurde anschließend verwendet, um unter Zuhilfenahme einer Suchmaschine Webseiten zu finden. Alternativ hätten die ca. fünf Millionen Webseiten des Open Directory Projekts verwendet werden können, hiervon wurde jedoch abgesehen, da die Webseiten im Gegensatz zu den Kategorien subjektiv sind und somit zu einer Verzerrung des Datensatzes führen könnten.

Bei der Wahl der Suchmaschine fiel die Wahl aus pragmatischen Gründen auf Microsoft Bing, da die zur Verfügung gestellte API[4] im Vergleich zum Kontrahent Google zwei entscheidende Vorteile aufweist. Zum einen ist die Anzahl der von Bing zurückgegebenen Suchtreffer nicht limitiert, während der Mitbewerber Google maximal 64 Ergebnisse liefert[5]. Darüber hinaus ist die Nutzung der API bei Bing weniger reglementiert, sodass

[2] `http://rdf.dmoz.org/rdf/categories.txt`, letzter Zugriff Mai 2011

[3] Auf der Webseite selbst wird ein Umfang von einer Million Kategorien angegeben, in der bereitgestellten Datei categories.txt sind jedoch nur ca. 800.000 Kategorien enthalten, letzter Zugriff Mai 2011.

[4] `http://www.bing.com/developers`, letzter Zugriff Mai 2011

[5] `http://code.google.com/intl/de/apis/websearch/docs/reference.html`, letzter Zugriff Mai 2011

weder die Frequenz, mit der Anfragen gestellt werden dürfen, noch deren maximale Anzahl einer Beschränkung unterliegen.

Jede der knapp 57.000 Kategorien wurde als Suchbegriff an Bing übergeben und die ersten maximal 1.000 URLs gespeichert. Da die URLs der Webseiten in der Regel auf spezifische Unterseiten verweisen, wurde jeweils nur der komplette Domain-Teil inklusive Subdomains persistiert. Das Resultat war Liste 2 mit ca. acht Millionen Webseiten.

3.4.3. Ermittlung der Feed URLs

Die ermittelten acht Millionen Webseiten wurden anschließend auf das Vorhandensein von Feed URLs untersucht. Hierzu wurde eine Technik eingesetzt, die gewöhnlich als *Feed Autodiscovery* bekannt ist. Der zugrunde liegende Mechanismus besteht in der Einbettung von speziellen Link-Elementen in den Header einer HTML-Seite, wie in Listing 3.1 dargestellt ist. Dem Leser mag die Autodiscovery in Form eines in der Regel orangen Symbols bekannt sein, welches Webbrowser in der Adresszeile einblenden, wenn die entsprechende Seite über einen oder mehrere Feeds verfügt.

```
1 <link rel="alternate" type="application/rss+xml" title="CNN - Top
    Stories [RSS]" href="http://rss.cnn.com/rss/edition.rss">
```

Listing 3.1: Link zu einem Feed im Header einer Webseite.

Obwohl dieses Verfahren bisher nicht offiziell standardisiert wurde, hat es sich als Industriestandard etabliert, sodass die meisten Webseiten, die einen Feed anbieten, auch die Autodiscovery-Funktion unterstützen. Ausnahmen bilden u. a. die Seiten der British Broadcasting Corporation (BBC) und Cable News Network (CNN). Während die BBC auf ihrer Startseite[6] keine Feeds anbietet, stellt CNN nur einen Feed auf der Startseite zur Verfügung, hält jedoch eine separate Übersichtsseite[7] mit den angebotenen Feeds bereit. Feeds, welche nicht mit der Autodiscovery gefunden werden können, sind nicht in dem erstellten Datensatz enthalten.

Das Resultat der Autodiscovery auf den ca. acht Millionen Webseiten ist Liste 3 mit knapp vier Millionen Feed URLs. Interessanterweise enthält die Liste auch mehrere URLs von BBC und CNN, woraus geschlossen werden kann, dass diese Feeds von verschiedenen Webseiten verlinkt sind. Aus den Zahlen kann jedoch nicht geschlossen werden, dass jede zweite Webseite einen Feed bereitstellt. Sowohl in der Liste der Webseiten als auch in den Feed URLs sind Duplikate enthalten. Dies ist der Tatsache geschuldet, dass sich die Ergebnisse der Suchanfragen aus Abschnitt 3.4.2 überlappen.

3.4.4. Bereinigung der Feed URLs

In Liste 3 der gewonnenen vier Millionen Feed URLs konnten zwei Typen von Duplikaten identifiziert werden: *tatsächliche Duplikate* und *Format-Duplikate*. Neben Duplikaten enthielt die Liste noch einen Anteil „defekter“ URLs, die auf nicht (mehr) verfügbare Feeds verweisen. Abbildung 3.2 zeigt die Bereinigung der URLs als einen verfeinerten Ausschnitt

[6] `http://www.bbc.com`, letzter Zugriff Mai 2011
[7] `http://edition.cnn.com/services/rss/`, letzter Zugriff Mai 2011

aus Abbildung 3.1 (Seite 79). Zuerst wurden die tatsächlichen Duplikate entfernt (Liste 4), anschließend alle erreichbaren Feeds ermittelt (Liste 5) und abschließend die Format-Duplikate entfernt und die verbleibenden URLs in Liste 6 zwischengespeichert.

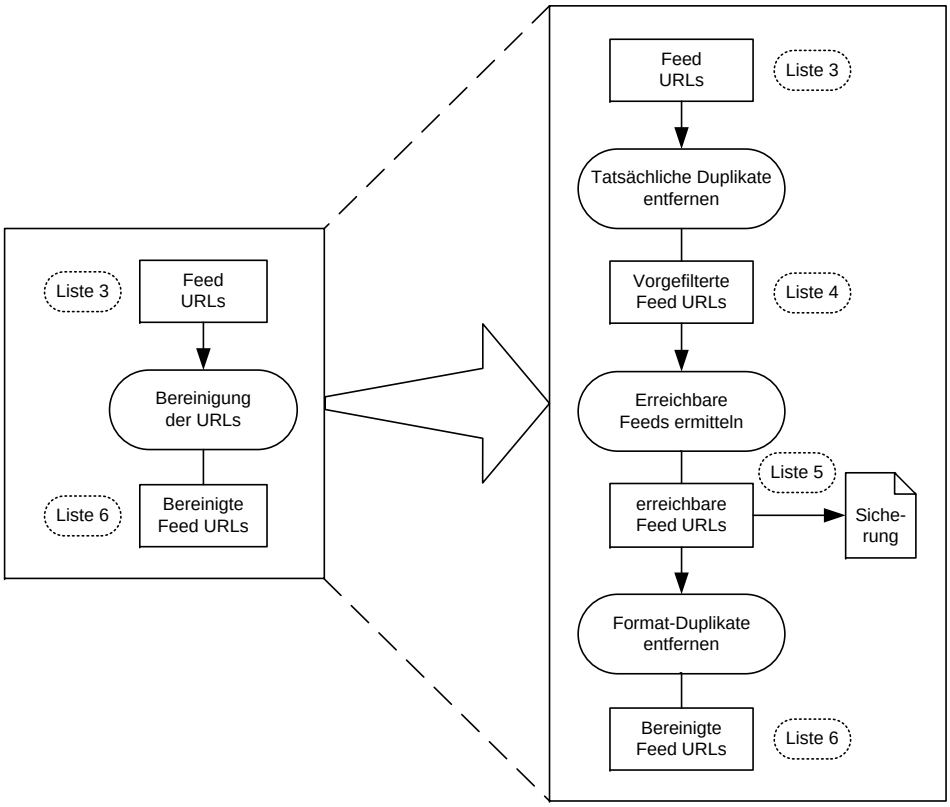

Abbildung 3.2.: Bereinigung der Feed URLs im Detail (Ausschnitt aus Abblidung 3.1).

Tatsächliche Duplikate sind identische Feed URLs, welche sich aus überlappenden Suchergebnissen und Webseiten ergeben, die auf die selbe Feed URL verweisen. Die Identifikation tatsächlicher Duplikate ist trivial, nach ihrer Eliminierung beträgt die Anzahl verschiedener Feed URLs in Liste 4 noch 1,5 Millionen. Im Zuge der Entfernung identischer URLs wurden auch jene entfernt, die eine SessionID der Form *PHPSESSID* oder *sessionid* enthielten, da viele der mit SessionIDs versehenen Feeds identisch waren und sich nur im Identifikator unterschieden. Ein weiterer Grund für die Entfernung ist, dass einige Systeme Fehlerseiten zurückgeben, wenn ein Aufruf mit einer abgelaufenen SessionID erfolgt. Der Anteil Feeds mit SessionID in Liste 3 betrug jedoch nur 0,1 %. Insgesamt ist der Schwund von 3,8 auf 1,5 Millionen URLs beachtlich, jedoch nicht überraschend, da Liste 2 mit den von Bing gelieferten Webseiten keiner Filterung unterzogen wurde und populäre Webseiten mehrfach vertreten waren.

Liste 4 der vorgefilterten Feed URLs enthält noch eine Vielzahl defekter URLs, welche nicht (mehr) auf Feeds verweisen, illegale Zeichen enthalten oder in sonstiger Weise fehlerhaft sind. Daher wurden als Nächstes alle URLs einmal angefragt und jene entfernt,

unter denen kein Feed erreichbar war. In diesem Schritt schrumpfte die Anzahl der URLs um ca. 12 %, sodass reichlich 1,3 Millionen tatsächlich erreichbarer Feeds als Zwischenergebnis der Suche gefunden wurden. Im Vergleich zu Statistiken aus dem Jahr 1999 ist der Anteil fehlerhafter Links deutlich größer als vor einem Jahrzehnt, damals lag er bei rund 5 %, gemittelt über elf Suchmaschinen [LG99].

Nachdem die tatsächlichen Duplikate und fehlerhaften Links entfernt wurden, ist es an der Zeit, die Format-Duplikate zu identifizieren und zu entfernen. Ein Format-Duplikat eines Feeds liegt vor, wenn ein Provider den gleichen Inhalt in verschiedenen Formaten anbietet, typischerweise um dem Leser die Wahl zwischen RSS 2.0 und Atom 1.0 zu ermöglichen. Für die Erstellung des Datensatzes wurde entschieden, dass im Falle multipler Feed-Formate Atom 1.0 den Vorzug erhält. Hintergrund der Entscheidung ist zum einen, dass Atom 1.0 ein standardisiertes Format ist. Zum anderen bietet RSS 2.0 zwar theoretisch mehr Elemente an, die einer Vorhersage von Aktualisierungen dienlich sein können, die Analyse eines vorab erstellten Datensatzes hat jedoch ergeben, dass dieser Mehrwert nicht existiert, da Elemente wie skipHours oder ttl kaum verbreitet sind und zudem beinahe immer Standardwerte enthalten, die nicht dem tatsächlichen Aktualisierungsmuster der jeweiligen Feeds entsprechen. Darüber hinaus wird in Abschnitt 3.7 gezeigt, dass RSS 2.0 das mit Abstand dominierende Format ist. Die Eliminierung der Format-Duplikate unter Beibehaltung der Atom 1.0 Versionen der Feeds verschiebt die Verteilung der Formate im Datensatz ein wenig in Richtung Atom 1.0, RSS 2.0 bleibt dennoch das vorherrschende Format. Nicht zuletzt ist das Ziel der Arbeit die Vorhersage von Aktualisierungen basierend auf der Historie der Feeds, welche von der Publikation neuer Inhalte und nicht dem verwendeten Format abhängig ist. Querbeziehungen zwischen den Formaten, etwa ob ein Inhalt zuerst in der Atom- oder der RSS-Variante eines Feeds veröffentlicht wurde, sind nicht von Interesse.

Zur Eliminierung der Format-Duplikate wurde die aktuelle Liste 5 der 1,3 Millionen URLs zunächst auf typische Hinweisreize in den URLs untersucht, die auf Duplikate hinweisen. Listing 3.2 zeigt einige exemplarische Format-Duplikate.

```
1  http://www.organicfacts.net/feed/atom.html
2  http://www.organicfacts.net/feed/rss.html
3
4  http://kidneydiettips.davitablogs.com/?feed=atom
5  http://kidneydiettips.davitablogs.com/?feed=rss2
6
7  http://www.espeleovalencia.com/index.php/feed/
8  http://www.espeleovalencia.com/index.php/feed/atom/
9
10 http://www.essenzadicannella.com/feeds/posts/default
11 http://www.essenzadicannella.com/feeds/posts/default?alt=rss
12
13 http://www.br-online.de/br/jsp/global/funktion/rssexport/rssExport.jsp?
       rssType=atom&contentId=/index.xml&bereich=HP
14 http://www.br-online.de/br/jsp/global/funktion/rssexport/rssExport.jsp?
       rssType=rss20&contentId=/index.xml&bereich=HP
```

Listing 3.2: Fünf Beispiele für Format-Duplikate.

Bei den ersten beiden Feeds ist das Format im Dateinamen kodiert, in den Zeilen 4 und 5 wird es als GET-Parameter übergeben. Den Feed URLs in den Zeilen 7 und 8 sowie 10 und 11 ist es nicht direkt anzusehen, dass es sich um verschiedene Formate handelt, da in Zeile 7 respektive 10 keine Angabe eines Formats enthalten ist. Das fünfte Paar Feed URLs weist die Besonderheit auf, dass die Hinweisreize auf das Vorhandensein eines Formates auch mehrfach in einer URL vorkommen können, im gezeigten Beispiel ist jeweils drei mal die Zeichenkette „rss“ enthalten, in beiden kommt zusätzlich noch das in dem Fall entscheidende Unterscheidungsmerkmal „rss20“ bzw. „atom“ vor.

Aus den gewonnenen Erkenntnissen wurde der im Folgenden beschriebene Algorithmus zur Eliminierung der Format-Duplikate entwickelt. Grundlage bilden 13 im Datensatz identifizierte Muster, mit denen die Feed-Formate kodiert sind. Diese Muster sind in Listing 3.3 dargestellt, ihr Aufbau wird nachfolgend dargelegt. Aufgrund der sehr geringen Verbreitung anderer Unterformate bzw. Vorgängerversionen beschränkt sich die Erkennung auf die sehr stark verbreiteten Formate RSS 2.0 und Atom 1.0.

```
1 ATOM   = (atom10|atom1.0|atom_1.0|atom_10|atom)
2 RSS    = (rss_2.0|rss_200|rss_20|rss2.0|rss200|rss20|rss2|rss)
3 FORMAT = (?<![a-zA-Z0-9]) ( ATOM | RSS ) (?![a-zA-Z0-9])
```

Listing 3.3: Muster zum Auffinden von Format-Duplikaten.

Zunächst wurden alle Feed URLs anhand ihrer Domain gruppiert. Die erhaltenen Gruppen wurden anschließend sequentiell per Pattern-Matching auf das Muster FORMAT aus Zeile 3 hin untersucht. Unterschieden sich in einer Gruppe zwei oder mehr Feed URLs ausschließlich in dem gesuchten Muster, so wurde ein Format-Duplikat erkannt. Mit diesem Schritt wurden die ersten beiden sowie das fünfte Feed-Paar in Listing 3.2 als Format-Duplikate identifiziert. Konnte eine URL als Präfix einer anderen ermittelt werden, so wurde ebenfalls ein Duplikat gefunden. Da in diesem Fall nicht alle Formate ermittelt werden konnten, wurde die kürzere URL verworfen. Mit dieser Regel wurden das dritte und vierte Feed-Paar des aktuellen Beispiels ermittelt und somit alle vorliegenden Duplikate eliminiert.

Zwei Details des Algorithmus sollen nicht unerwähnt bleiben. Die Erkennung von Duplikaten in URLs wie dem fünften Feed-Paar gestaltete sich am aufwendigsten, da die Zeichenkette „rss“, welche bei anderen Feeds (siehe viertes Paar) zur Identifikation des Formates dient, hier ein normaler Bestandteil der URL ist. Zur Vermeidung dieser als *false positives* bezeichneten Treffer wurden die beiden Muster RSS und ATOM in Zeile 3 (Listing 3.3) mit zwei Ausdrücken gekapselt, die Wortgrenzen darstellen. Eine Format-Zeichenkette wird nur dann als Identifikator betrachtet, wenn sie kein Bestandteil eines sie umgebenden Wortes ist, d. h. weder vor noch nach dem Muster ein Buchstabe oder eine Ziffer steht.

Das zweite entscheidende Detail ist die Reihenfolge der Muster in den Zeilen 1 und 2. Hier war darauf zu achten, dass keines der Muster als Präfix eines Nachfolgers verwendet werden kann, da beim Pattern-Matching und der verwendeten oder-Verknüpfung der Formate stets das erste Muster als Treffer zurückgegeben wird, welches in der betrachteten URL gefunden wird. Eine Invertierung der Musterreihenfolge aus Listing 3.3 Zeile 2 und eine anschließende Anwendung des Algorithmus auf das zweite Feed-Paar hätten zur

Folge, dass anstatt „rss2“ nur „rss“ identifiziert werden würde. Beim darauf folgenden Vergleich der beiden URLs würden sich diese jedoch um das Zeichen „2“ unterscheiden[8] und das Duplikat somit nicht identifiziert.

Nach der Entfernung der Format-Duplikate verbleiben rund eine Million Feed URLs, welche in Liste 6 abgelegt werden. Wenngleich eine Vielzahl von Duplikaten entfernt werden konnte, so enthält die Liste 6 weiterhin einige Duplikate. Der vorgestellte Algorithmus wurde ausführlich am vorhandenen Datensatz getestet, auf eine Evaluierung mittels Goldstandard Verfahren wird jedoch verzichtet, da er kein zentraler Bestandteil der Arbeit ist. Abschließend sei darauf hingewiesen, dass der Datensatz noch einen dritten Typ von Duplikaten enthalten kann. Ressourcen-Duplikate sind Feeds, welche über komplett verschiedene URLs erreicht werden können, die auf exakt die gleiche Ressource zeigen. Um diese Duplikate zu ermitteln, müssten die Inhalte der Feeds miteinander verglichen werden. Nach eigener Erfahrung ist dieser Typ von Duplikaten äußerst selten, sodass der Aufwand der Eliminierung in keinem sinnvollen Verhältnis zu dessen Nutzen stehen würde. Die Anzahl der in Liste 6 möglicherweise verbleibenden, unentdeckten Duplikate wird als sehr gering eingestuft.

3.4.5. Zufällige Auswahl einer Teilmenge

Der letzte Schritt zum Auffinden der zu verwendenden Feed URLs besteht in der zufälligen Auswahl einer Teilmenge von 200.000 Feed URLs, welche als Liste 7 der nächsten Phase – der lückenlosen Aufzeichnung der Historie – übergeben werden. Vorangegangene Experimente haben ergeben, dass diese Anzahl mit den zur Verfügung stehenden technischen Ressourcen[9] noch handhabbar ist. Zudem sind 200.000 Feeds ein sehr umfassender Datensatz, sodass die in Abschnitt 3.1 aufgestellte Anforderung eines breitgefächerten Datensatzes als erfüllt betrachtet wird.

3.4.6. Zusammenfassung der Teilschritte zum Auffinden der Feed URLs

Die vorangegangenen Teilschritte zum Auffinden der Feed URLs werden in Tabelle 3.2 noch einmal übersichtlich zusammengefasst.

Schritt	**Liste**	**Resultat**
3.4.1: Auswahl von Kategorien aus dmoz.org	Liste 1	56.918 Kategorien
3.4.2: Auffinden von URLs von Webseiten	Liste 2	8.262.000 URLs
3.4.3: Untersuchung der Webseiten auf Feed-URLs	Liste 3	3.814.955 URLs
3.4.4: Entfernung der URL-Duplikate	Liste 4	1.529.086 URLs
3.4.4: Ermittlung tatsächlich erreichbarer Feeds	Liste 5	1.341.661 URLs
3.4.4: Entfernung von Format-Duplikaten der Feeds	Liste 6	994.413 URLs
3.4.5: Zufällige Auswahl einer Teilmenge der Feeds	Liste 7	200.000 URLs

Tabelle 3.2.: Zusammenfassung der Teilschritte zum Auffinden der Feed URLs.

[8] http://kidneydiettips.davitablogs.com/?feed=atom ⇒ http://kidneydiettips.davitablogs.com/?feed=
http://kidneydiettips.davitablogs.com/?feed=rss2 ⇒ http://kidneydiettips.davitablogs.com/?feed=2

[9] Die zur Verfügung stehenden technischen Ressourcen werden in Abschnitt 4.3.1 beschrieben.

3.5. Lückenlose Aufzeichnung der Historie

Die zweite Phase der Erstellung des Datensatzes war die lückenlose Aufzeichnung der Aktualisierungen der 200.000 Feeds aus Liste 7 über einen Beobachtungszeitraum von vier Wochen. Die in Abbildung 3.1 (Seite 79) bereits skizzierten Schritte seien nun in Erinnerung gerufen. Jeder Feed aus Liste 7 wurde zunächst angefragt und das empfangene Dokument in einer komprimierten Datei abgelegt. Anschließend wurde das XML-Dokument geparst, um alle Einträge zu extrahieren. Alle bisher nicht bekannten Einträge wurden fortlaufend in einer csv-Datei gespeichert, in welcher der Strom der Feed-Einträge über den kompletten Beobachtungszeitraum zusammengesetzt wurde. Anschließend wurde das Aktualisierungsintervall des Feeds ermittelt und mit diesem die nächste Überprüfung geplant. Dieser Zyklus wurde für jeden der 200.000 Feeds regelmäßig wiederholt. Die Teilschritte werden im Folgenden im Detail betrachtet. Primäres Ziel war die Erfüllung der in Abschnitt 3.1 aufgestellten Anforderung der Vollständigkeit des Datensatzes. Sie gilt als erfüllt, wenn die Historie der Feed-Einträge lückenlos aufgezeichnet wurde oder dies erkennbar nicht der Fall ist. Im selben Schritt wird die Anforderung eines realen Datensatzes erfüllt.

3.5.1. Rekapitulation der grundlegenden Eigenschaften von Feeds

In den Abschnitten 2.1.2 und 2.3.4 wurden einige wichtige Eigenschaften von Feeds herausgearbeitet, die für die Aufzeichnung der Historie von zentraler Bedeutung sind.

Feeds sind üblicherweise nach dem FIFO-Prinzip aufgebaut und haben eine durchschnittliche Fenstergröße von 15 Einträgen, die meist konstant ist. Dieser Aufbau der XML-Dokumente kann jedoch nicht als gegeben angenommen werden, da einige Feeds nicht diesem Muster entsprechen, wie vorangegangene Experimente gezeigt haben. In einzelnen Fällen wurde sogar beobachtet, dass Einträge mehrere Tage später unter Verwendung eines falschen Publikationsdatums in das den Feed repräsentierende Dokument eingeschoben wurden. Datumsangaben können daher fehlerhaft sein und repräsentieren nicht immer das tatsächliche Veröffentlichungsdatum eines Eintrages. Laut der Feed Spezifikationen müssen Einträge über einen eindeutigen Identifikator verfügen.

Das wichtigste Unterscheidungsmerkmal ist die hohe Diversität der Aktualisierungsmuster der Feeds. Während bei einzelnen Feeds mehrere neue Einträge pro Sekunde zu verzeichnen sind, werden andere über Jahre nicht geändert. Das Intervall zwischen zwei Änderungen ist zudem meist variabel.

3.5.2. Erkennung neuer, bekannter und fehlender Einträge

Der erste Schritt des Zyklus bestand aus dem Abfragen und Herunterladen des Feeds. Anschließend wurde er geparst, um die enthaltenen Einträge zu extrahieren. Abbildung 3.3 illustriert 13 Feed-Einträge entlang einer Zeitachse, vier Abfragen des Feeds sowie die entsprechenden Fenster W, die pro Abfrage gesehen werden. Der Feed hat eine konstante Fenstergröße von fünf Einträgen.

Abfrage $\mathcal{A}$ stellte den ersten Zugriff auf diesen Feed dar, sodass per Definition alle im

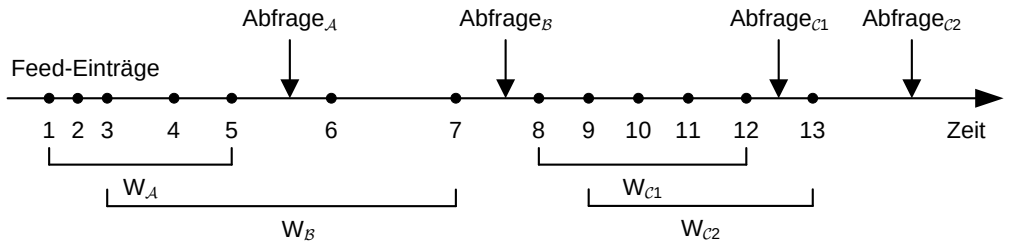

Abbildung 3.3.: Feed-Einträge entlang der Zeitachse, vier Abfragen des Feeds sowie die entsprechenden Fenster, die pro Abfrage gesehen werden.

Fenster $W_{\mathcal{A}}$ sichtbaren Einträge 1–5 neu waren. In diesem Fall wurde der komplette Feed gemeinsam mit dem empfangenen HTTP-Header in einer komprimierten Datei abgelegt. Zusätzlich wurde eine csv-Datei angelegt, in der alle extrahierten Einträge in verkürzter Form abgelegt wurden. Das Format der beiden Dateien wird später erläutert.

In jedem nachfolgenden Durchgang des Zyklus musste für jeden empfangenen Eintrag überprüft werden, ob dieser bereits im vorangegangenen Durchlauf erfasst wurde oder ob er neu ist. Der Algorithmus zur Identifikation und dem Vergleich von Einträgen ist aufwendig und wird im Anschluss eruiert, im Moment genügt es zu wissen, dass die Einträge identifiziert und verglichen werden können.

Angenommen, die als $\mathcal{B}$ gekennzeichnete Abfrage war die zweite Abfrage des Feeds. Die im korrespondierenden Fenster $W_{\mathcal{B}}$ enthaltenen Einträge 3–7 wurden mit den bereits vorhandenen verglichen, wobei die Einträge 3–5 als Duplikate bereits bekannter und die Einträge 6 und 7 als neu identifiziert wurden. Die neuen Einträge wurden der bestehenden csv-Datei hinzugefügt, die sodann die Einträge 1–7 in der Reihenfolge ihrer Veröffentlichung enthielt.

Für die Rekonstruktion eines Stroms aus allen Einträgen eines Feeds wurde zunächst die Annahme getroffen, dass Feeds dem Muster eines FIFO-Puffers entsprechen. Ist dies der Fall, so müssen sich die in den Fenstern zweier sequentieller Abfragen enthaltenen Einträge in mindestens einem überschneiden, um eine lückenlose Historie zu gewährleisten. Im aktuellen Beispiel überschnitten sich die Fenster $W_{\mathcal{A}}$ und $W_{\mathcal{B}}$ in den Einträgen 3–5.

Die Abfragen $\mathcal{C}1$ und $\mathcal{C}2$ stellen zwei Möglichkeiten dar, bei denen nicht festgestellt werden kann, ob Einträge verpasst wurden. Folgte Abfrage $\mathcal{C}1$ auf $\mathcal{B}$, so ist die Schnittmenge der in den Fenstern $W_{\mathcal{B}}$ und $W_{\mathcal{C}1}$ enthaltenen Einträge die leere Menge. Es kann nicht festgestellt werden, ob ein Eintrag verpasst wurde oder nicht. Diese potentielle Lücke im Datensatz wird als *Miss* bezeichnet und entsprechend in der csv-Datei markiert. Die Datei enthält somit zeilenweise die Einträge 1–7, den speziellen Miss-Eintrag sowie die Einträge 8–12. Die Bedingung für das Vorliegen eines Miss ist in Gleichung 3.1 dargestellt.

$$(W_n \cap W_{n+1} == \emptyset) \Rightarrow \text{Miss} \tag{3.1}$$

Betrachtet man statt $\mathcal{C}1$ die Abfrage $\mathcal{C}2$ als direkten Nachfolger von $\mathcal{B}$, so überlappen sich die Fenster $W_{\mathcal{B}}$ und $W_{\mathcal{C}2}$ ebenfalls nicht und es wird gleichermaßen ein Miss festgestellt

und in der csv-Datei vermerkt. In diesem Fall wurde tatsächlich ein Eintrag verpasst. Wie das Beispiel der Abfragereihenfolge $\mathcal{B}$–$\mathcal{C}1$ zeigt, folgt aus dem Vorhandensein eines Miss nicht zwangsläufig, dass der Datensatz eine Lücke enthält – es konnte nur nicht festgestellt werden, dass keine Lücke vorhanden ist. Ein Miss steht somit für null oder mehr verpasste Einträge.

Bisher wurde vorausgesetzt, dass Feeds dem FIFO-Muster entsprechen. In Abbildung 3.4 ist ein Feed dargestellt, der nicht diesem Muster entspricht. Zur Unterscheidung der Einträge sei ein Identifikator eingeführt. Bis zum Eintrag h an Position 8 entspricht der Feed dem in Abbildung 3.3 vorgestellten, an Position 9 wird jedoch der Eintrag e erneut veröffentlicht. Aus Sicht eines Nutzers, der bei Abfrage $\mathcal{G}$ die Einträge a–e heruntergeladen hat und in Abfrage $\mathcal{H}$ die Einträge e, i, j, k, l zurückgegeben bekommt, ergibt sich eine Überschneidung der Fenster $W_\mathcal{G}$ und $W_\mathcal{H}$ im Eintrag e. Nach Gleichung 3.1 liegt kein Miss vor und der Nutzer hat in diesem Fall auch keine Chance festzustellen, dass er Einträge verpasst hat.

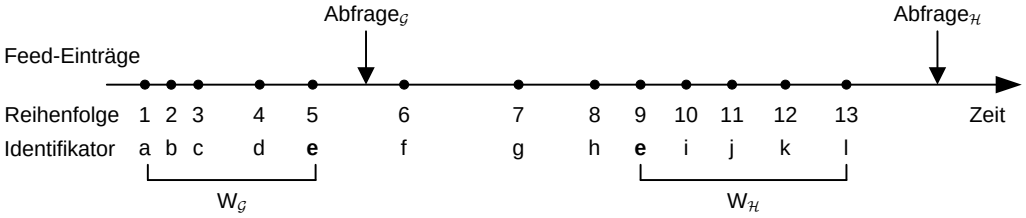

Abbildung 3.4.: Feed-Einträge entlang der Zeitachse mit Wiederholung des bereits früher veröffentlichten Eintrags e.

Für den aktuellen Schritt der Erkennung neuer, bekannter und fehlender Einträge kann zusammengefasst werden, dass jeder neue Eintrag in der Reihenfolge seiner Veröffentlichung gespeichert wird. Überlappen sich die Fenster zweiter sequentieller Abfragen nicht, so wird die potentielle Lücke mittels eines speziellen Miss-Eintrags dokumentiert. Entspricht ein Feed nicht dem FIFO-Muster, so kann bei der Erstellung des Datensatzes weder garantiert werden, dass dieser Feed vollständig erfasst wird, noch dass die Lücken gekennzeichnet sind.

3.5.3. Identifikation von Einträgen

Im vorangegangenen Abschnitt wurde angenommen, jeden Eintrag eines Feeds eindeutig identifizieren zu können, um darauf aufbauend neue von bereits bekannten Einträgen zu unterscheiden. Die Identifikation von Einträgen ist jedoch komplizierter als zunächst vermutet und wird nun im Detail vorgestellt.

Laut der Feed Spezifikationen von RSS 2.0 und Atom 1.0 (vgl. Abschnitt 2.3.1) müssen Einträge über einen eindeutigen Identifikator verfügen, der genau diesem Verwendungszweck dient. Untersuchungen der 200.000 zufällig ausgewählten Feeds in Liste 7 zeigen jedoch, dass nicht alle Feeds standardkonform sind. Jeder Feed wurde einmal angefragt und anschließend das feed- respektive channel-Element sowie der erste Eintrag überprüft.

Bei 11 % der Feeds konnte kein Identifikator gefunden werden, sodass für die Identifikation von Einträgen weitere Attribute wie der Titel des Eintrags, der Link zum Artikel sowie das Datum der Veröffentlichung auf ihre Eignung hin überprüft werden.

Wie bei realen Daten zu erwarten, bringen alle Attribute Besonderheiten mit sich. Zunächst kann jedes der Attribute einen leeren Inhalt aufweisen bzw. nicht präsent sein. Bei einigen Feeds konnte beobachtet werden, dass sich verschiedene Einträge eines Fensters in einem oder mehreren der Attribute Titel, Link und Datum gleichen, obwohl es sich um verschiedene Einträge handelt. Zudem konnte in der manuellen Analyse einiger Feeds beobachtet werden, dass die Links versteckte SessionIDs enthalten, die z. B. als Unterordner im Pfad „getarnt“ waren und von ihrem Muster her nur beim Vergleich der Links mehrerer Fenster zu identifizieren waren. Fließen derart aufgebaute Links in einen Identifikator ein, können eigentlich gleiche Einträge nicht mehr einander zugeordnet werden. Diese Besonderheit trat nur in Einzelfällen auf, sodass sie zwar erwähnt sei, jedoch bei der Erstellung des Datensatzes vernachlässigt werden kann.

Bei Untersuchungen der Datumsfelder pubDate, updated und published fiel auf, dass die Werte bei einigen Feeds dynamisch zur Abfragezeit gesetzt werden. Dies hat zur Folge, dass alle Einträge einer Abfrage den gleichen Zeitstempel aufweisen und sonst identische Einträge bei jeder Abfrage verschiedene Zeitstempel besitzen. Für Feeds dieser Klasse ist das Datum als Teil eines kombinierten Identifikators kontraproduktiv, Duplikate würden nicht erkannt werden und die Qualität des Datensatzes würde sinken. Aus diesem Grund fließt das Datum nicht in den eigenen Identifikator ein.

Aus den gewonnenen Erkenntnissen wird Algorithmus 1 zur Generierung des Identifikators eines Eintrags vorgeschlagen, der auf einer Kombination der Attribute des Eintrags aufbaut. Bis auf den komplexen Typ *feedEntry* enthalten alle Variablen der Zeilen 1–8 Zeichenketten, der Operator + dient zur Verkettung zweier Zeichenketten. Die beiden aufgerufenen Methoden geben jeweils einen Booleschen Wert zurück und bestimmen das Feed-Format. Unterformate werden nicht unterschieden, ist ein Attribut in einem Feed-Eintrag nicht enthalten, so wird als Standard die leere Zeichenkette verwendet.

Der erstellte und zurückgegebene Identifikator setzt sich aus dem Titel, Link und Identifikator des Feed-Formats zusammen. Abschließend wird aus pragmatischen Gründen die Hash-Funktion sha1 verwendet, um Identifikatoren gleicher Länge zu erhalten und somit die im weiteren Verlauf benötigte Persistierung der Identifikatoren in einer Datenbank zu optimieren.

Algorithmus 1 Generierung eines eindeutigen Identifikators eins Feed-Eintrags.

```
1: generateID(feedEntry) {
2: newID ← feedEntry.title + feedEntry.link
3: if isAtomFeed() then
4:     newID ← newID + feedEntry.id
5: else if isRSSFeed() then
6:     newID ← newID + feedEntry.guid
7: end if
8: return sha1(newID)
9: }
```

3.5.4. Bestimmung des Abfrageintervalls

Der finale Schritt beim Durchlaufen eines Zyklus zur Aufzeichnung der Historie besteht in der Berechnung des *Abfrageintervalls* $u_{Abfrage}$, d. h. wann der Feed das nächste Mal auf Aktualisierungen zu überprüfen ist. Hierfür sei zunächst an das wichtigste Unterscheidungsmerkmal der Feeds erinnert: die hohe Diversität der Aktualisierungsmuster.

Der in Abschnitt 2.8 eingeführte, einfachste Ansatz ist, alle Feeds in dem gleichen, konstanten Intervall anzufragen. Feed-Reader-Programme verwenden üblicherweise ein Intervall von einer Stunde: $u_{Abfrage} = 60\ min$, dies wird als Fix1h bezeichnet. Für die Erstellung des Datensatzes ist Fix 1h nicht praktikabel. Bei einem Intervall von einer Stunde ist zu erwarten, dass bei vielen Feeds regelmäßig Einträge verpasst werden. Das Herabsetzen des Intervalls auf einen sehr kleinen Wert wie $u_{Abfrage} = 1\ sek$, bei dem bei fast keinem der Feeds ein Eintrag verpasst wird, ist zwar theoretisch möglich, würde jedoch einen ökonomisch und ökologisch nicht vertretbaren Umfang an technischen Ressourcen beanspruchen und ist daher nicht praktisch umzusetzen.

Die naheliegende Lösung ist, Feeds mit unterschiedlichen Intervallen anzufragen. Für die Erstellung des Datensatzes wurde $u_{Abfrage}$ an jeden Feed angepasst. Hierzu seien zwei Ansätze vorgestellt, die auf dem Veröffentlichungsdatum der Einträge aufbauen. Zunächst sei der Idealfall angenommen, dass alle Einträge einen solchen Wert aufweisen, dieser dem tatsächlichen Veröffentlichungsdatum entspricht und zusätzlich die Systemzeiten des Feed Anbieters und des Abfragenden synchronisiert sind.

Nach jeder Abfrage eines Feeds werden die Daten der Veröffentlichung aller Einträge zunächst chronologisch geordnet, um das durchschnittliche *beobachtete Aktualisierungsintervall* $u_{beobachtet}$ des Feeds nach Gleichung 3.2 zu berechnen, wobei $|W|$ die aktuelle Fenstergröße und t_n der Veröffentlichungszeitpunkt des n-ten Eintrags ist. Für Feeds, die nur einen einzelnen oder gar keinen Eintrag enthalten, kann kein Intervall bestimmt werden. In diesem Fall wird ein Wert β angenommen, dessen Ermittlung und Bedeutung in Abschnitt 3.5.5 nachgereicht werden.

$$u_{beobachtet} = \begin{cases} \frac{1}{|W|-1} \sum_{n=1}^{|W|-1} (t_{n+1} - t_n) & \text{wenn } |W| > 1 \\ \beta & \text{sonst} \end{cases} \tag{3.2}$$

$u_{beobachtet}$ wird immer aus dem Fenster der aktuellen Abfrage berechnet, Werte vorangegangener Abfragen fließen nicht ein. Setzt man $u_{Abfrage} = u_{beobachtet}$, so wird der Feed im Durchschnitt einmal pro neuem Eintrag angefragt. Zur Vermeidung zu häufiger oder zu seltener Abfragen wird der Wertebereich von $u_{Abfrage}$ durch die untere Schranke α und die obere Schranke β begrenzt (Gleichung 3.3).

$$u_{Abfrage} = min\left(\beta,\ max\left(\alpha, u_{beobachtet}\right)\right) \tag{3.3}$$

Für die Einführung der Schranken α und β gab es mehrere Gründe. Die untere Schranke wird benötigt, um ein zu häufiges Abfragen der Feeds zu vermeiden. Zu häufige Abfragen führten in vorangegangenen Experimenten dazu, dass einige Server die wiederholten Abfragen als Angriff klassifizierten und blockten. Ein Ausweg wäre, die Abfragen über dynamisch wechselnde Proxy-Server zu leiten. Hiervon wurde aufgrund der geringen

Anzahl betroffener Feeds (ca. 0,04 % des Datensatzes) und der steigenden Komplexität des Gesamtsystems abgesehen. Bei einigen Feeds, wie den letzten Änderungen der Wikipedia[10], konnte beobachtet werden, dass der Feed in der Regel nur alle 60 Sekunden neu erstellt wurde. Es ist zu vermuten, dass der Feed von einem Cron Job[11] gebaut wird, dessen Intervall auf 60 Sekunden eingestellt ist. Bei diesem und allen ähnlich generierten Feeds liefert ein kürzeres Abfrageintervall keinen Mehrwert. Werden innerhalb einer Minute mehr Einträge erstellt als in einem Fenster zu sehen sind, verpassen alle Leser des Feeds die jeweils ältesten Einträge. In diesem Fall würde ein publish-subscribe-System, das auf dem Feed aufsetzt, keinen Vorteil bringen. Aus technischer Sicht musste zudem beachtet werden, dass zu kurze Intervalle eine sehr hohe Last erzeugten, die mit den zur Verfügung stehenden Ressourcen nicht mehr verarbeitet werden konnten. Als guter Kompromiss stellte sich eine untere Schranke von $\alpha = 1\ min$ heraus, bei der nur noch bei extrem häufig aktualisierenden Feeds wie Yahoo Answers[12] wiederholt Einträge verpasst wurden.

Die obere Schranke β wurde verwendet, um möglichst keine Aktualisierungen jener Feeds zu verpassen, welche verhältnismäßig nur sehr selten neue Einträge aufweisen. Eine solche obere Schranke ist bereits aus Abschnitt 2.5.3 von der Berechnung der adaptiven TTL beim Web Caching bekannt [CL98, RS02, INST04]. Als Beispiel seien sogenannte travel-Blogs herangezogen, auf denen Nutzer primär dann neue Inhalte veröffentlichen, wenn sie auf Reisen sind. Die Feeds dieser Blogs werden häufig über Monate nicht aktualisiert und ab einem gewissen Zeitpunkt wieder beinahe täglich. Um keine Einträge zu verpassen, kann in Gleichung 3.3 ein „kleiner“ Wert für die obere Schranke β eingesetzt werden. Hierdurch werden selektiv nur Feeds mit großem $u_{beobachtet}$ beeinflusst. Als guter Kompromiss zwischen den zur Verarbeitung benötigten Ressourcen und dem Risiko, Einträge zu verpassen, stellte sich eine obere Schranke von $\beta = 6\ h$ heraus. Die Auswirkungen der Parameter sind in Abbildung 3.5 an einem Beispiel mit $\alpha = 1\ h$ und $\beta = 3\ h$ illustriert.

Die Verwendung der oberen Schranke brachte aus technischer Sicht einen Nachteil mit sich: Mehr als 90 % der Feeds des Datensatzes hatten ein beobachtetes Aktualisierungsintervall $u_{beobachtet} > 6\ h$, sodass die Eingrenzung des Abfrageintervalls $u_{Abfrage}$ nach Gleichung 3.3 dazu führen würde, dass exakt alle sechs Stunden eine Lastspitze von 180.000 zu verarbeitenden Feeds auftritt. Das Problem wurde pragmatisch gelöst, indem $u_{Abfrage}$ abschließend selektiv angepasst wurde. Dieser Schritt ist in Gleichung 3.4 dargestellt. Auf diese Weise wurde die Lastspitze der 180.000 Feeds im ersten Zyklus gleichmäßig auf einen Zeitraum von drei Stunden verteilt, spätere Zyklen brachten eine weitere Glättung und Verteilung auf die durch die obere Schranke vorgegebene Zeitspanne von sechs Stunden.

$$u'_{Abfrage} = u_{Abfrage} - \gamma \qquad \text{mit } \gamma = \begin{cases} random\left(0, \frac{\beta}{2}\right) & \text{wenn } u_{Abfrage} = \beta \\ 0 & \text{sonst} \end{cases} \tag{3.4}$$

Der Durchlauf des aktuellen Zyklus z zur Aufzeichnung der Historie des Feeds endet

[10] `http://ja.wikipedia.org/w/index.php?title=%E7%89%B9%E5%88%A5:%E6%9C%80%E8%BF%91%E3%81%AE%E6%9B%B4%E6%96%B0&feed=atom`, Zugriff 25.07.2011

[11] `http://unixhelp.ed.ac.uk/CGI/man-cgi?cron+8`, Zugriff 25.07.2011

[12] `http://answers.yahoo.com/rss/allq`

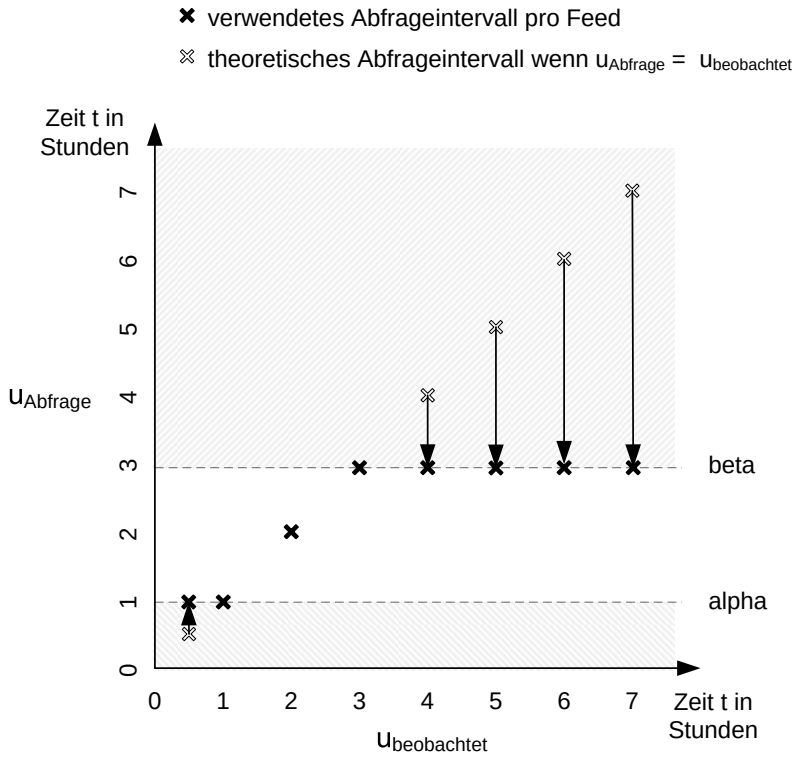

Abbildung 3.5.: Einfluss der Parameter auf das Abfrageintervall am Beispiel $\alpha = 1\ h$ und $\beta = 3\ h$.

mit der Planung des Zeitpunktes τ_{z+1} der nächsten Abfrage. Dieser ergibt sich aus dem Zeitpunkt τ_z sowie $u'_{Abfrage}$. Gleichung 3.5 stellt dies dar. Mit dem beschriebenen Verfahren wurde eine durchschnittliche Last von 1.000 Abfragen je Minute erreicht, die nur gering variierte und mit den zur Verfügung stehenden Ressourcen bewerkstelligt werden konnte.

$$\tau_{z+1} = \tau_z + u'_{Abfrage} \tag{3.5}$$

3.5.5. Behandlung von Zeitstempeln

Zur Berechnung des Abfrageintervalls wurde bisher der Idealfall angenommen, dass 1) alle Einträge ein Attribut mit dem Veröffentlichungsdatum aufweisen, 2) dessen Wert dem tatsächlichen Veröffentlichungsdatum entspricht und 3) die Systemzeiten des Feed Anbieters und Abfragenden synchronisiert sind.

Die in Abschnitt 3.7.7 nachgereichte Analyse des Datensatzes ergab, dass knapp 11 % der 200.000 Feeds in Liste 7 über keine Angabe eines Zeitstempels verfügen. Für diese Feeds wurde das Aktualisierungsintervall approximiert, indem bei jeder Abfrage den *neuen* Einträgen der aktuelle Zeitstempel zugeordnet und die Berechnung von τ_{z+1} mit diesen Werten vorgenommen wurde. Die Verwendung des aktuellen Zeitstempels führte

zunächst zu dem Effekt, dass der Feed zu häufig abgefragt wurde. Bei allen nachfolgenden Abfragen wurden jedoch nur die Zeitstempel der neuen Einträge korrigiert, sodass das Abfrageintervall über die Zeit an die tatsächliche durchschnittliche Änderungsrate des Feeds angepasst werden konnte.

Die Berechnung der Zeitdifferenz zwischen den Systemuhren des Feed Anbieters und des Abfragenden erfolgte unter Verwendung des Date-Attributes im HTTP-Header der erhaltenen Antwort. Ist die Zeitdifferenz bekannt, so kann bei Feeds, deren Einträge alle den gleichen Zeitstempel aufweisen, ermittelt werden, ob diese zur Abfragezeit oder einem früheren Zeitpunkt generiert wurden. Im ersten Fall dürfen die Zeitstempel nicht zur Berechnung des Abfrageintervalls eingesetzt werden. Die Zeitstempel werden zwar gespeichert, zur Ermittlung von τ_{z+1} wird der Feed jedoch so verarbeitet, als würde er keine Zeitstempel enthalten.

Die verbleibende Voraussetzung, dass Einträge das tatsächliche Veröffentlichungsdatum mit sich führen, ist am schwersten zu überprüfen. Bei dem Datum eines als neu identifizierten Eintrags kann nur dann festgestellt werden, dass es nicht dem tatsächlichen Veröffentlichungsdatum des Eintrags entspricht, wenn es in der Zukunft liegt oder älter ist als der Zeitpunkt der vorangegangenen Abfrage. In beiden Fällen wird es gespeichert, fließt aber nicht in die Berechnung der Intervalle ein. Stattdessen wird der aktuelle Zeitstempel der Abfrage verwendet. Die bereits geschilderte Beobachtung, dass Feeds per Cron Job erstellt werden und somit kurze Verzögerungen zwischen dem Veröffentlichungsdatum des Eintrags und der tatsächlichen Veröffentlichung im Feed liegen, wurde nicht berücksichtigt. Zur Erstellung des Datensatzes ist dieses Verhalten nicht relevant, da es dazu führt, dass ein Feed häufiger abgefragt wird als neue Einträge sichtbar sind. Dies geht zu Lasten der Systemressourcen, erhöht jedoch nicht das Risiko, Einträge zu verpassen und genügt somit der Anforderung nach Vollständigkeit des Datensatzes (Abschnitt 3.1).

Aus der Verarbeitung der eruierten Sonderfälle folgt häufig die Verwendung des Zeitstempels der Abfrage. Dies kann zur Folge haben, dass alle Einträge eines Fensters den gleichen Zeitstempel aufweisen. Wie bei Feeds mit einer Fenstergröße $|W| < 2$ ist es in diesen Fällen nicht möglich, ein Intervall zu berechnen, sodass in Gleichung 3.2 $u_{beobachtet} = \beta$ als Heuristik gesetzt wird. Ausschlaggebend für die Entscheidung ist, dass für 90 % der Feeds des Datensatzes $u_{beobachtet} > \beta$ gilt. Wird ein kleinerer Wert oder gar null als Standard gesetzt, so erzeugen Feeds mit fehlerhaften Zeitstempeln eine unnötig hohe Last bei der Erstellung des Datensatzes.

3.5.6. Einschränkungen bei der Verarbeitung von Feeds

Insgesamt konnten vier Ursachen identifiziert werden, die zu Einschränkungen in der Verarbeitung der Feeds führten.

In seltenen Fällen kam es vor, dass ein Feed temporär oder gar nicht verarbeitet werden konnte. Die nachfolgende Auswertung hat ergeben, dass bei 0,6 % der rund 40 Millionen Abfragen der Feed nicht (mehr) erreichbar war oder ein sonstiger Fehler vorlag und der Server einen entsprechenden HTTP-Status-Code zurückgab.

Eine weitere Fehlerquelle sind XML-Dokumente, die unzulässige Zeichen oder syntaktische Fehler in einem Eintrag enthalten. Der Feed Parser der zur Erstellung des

Datensatzes umgesetzten Implementierung baut auf der Bibliothek *Rome*[13] auf, welche alle gängigen Feed-Formate unterstützt. Konnte der Parser ein Dokument nicht verarbeiten, standen keine auswertbaren Daten zur Verfügung. Die Auswertung ergab, dass dieser Fall bei rund 0,7 % der Abfragen eintrat. Das XML-Dokument wurde unverarbeitet persistiert und kann z. B. von den Entwicklern von Parsern verwendet werden, um ihre Software fehlertoleranter zu gestalten.

Aus Gründen der Umsetzbarkeit wurden zudem zwei Einschränkungen getroffen: Das Herunterladen eines Feeds wurde gestoppt, sobald die Größe des bereits übertragenen Anteils des Dokuments ein Megabyte (MB) überschritt. Zudem wurden bestehende Verbindungen nach drei Minuten beendet. In Abschnitt 3.7.3 wird gezeigt, dass die durchschnittliche Größe eines Feeds 43 kB beträgt, Größen über einem MB sind äußerst selten, benötigen jedoch sehr viele Ressourcen zur Verarbeitung, sodass Verzögerungen im Betrieb des Crawlers auftreten können, die zu verpassten Einträgen in häufig aktualisierenden Feeds führen. Das Limit von drei Minuten für geöffnete HTTP-Verbindungen wurde eingeführt, da es in Ausnahmefällen vorkommen kann, dass ein Server die Verbindung abbricht, jedoch nicht explizit beendet. Die durchschnittliche Verbindungsdauer betrug eine Sekunde, sodass der erzwungene Abbruch nach drei Minuten den Normalbetrieb nicht beeinflusste.

In allen vier beschriebenen Fällen erfolgte die Erkennung von Misses bei der nächsten Abfrage wie bereits in Abschnitt 3.5.2 beschrieben. Aufeinander folgende Fehler in der Verarbeitung eines Feeds führten nur dann zu einem Miss, wenn potentiell ein Eintrag verpasst wurde. Somit konnte vermieden werden, dass jeder Fehler in der Verarbeitung in einem Miss resultierte.

Insgesamt traten bei knapp 0,5 % der Feeds sehr häufig Fehler auf, sodass das Verhältnis zwischen fehlerhaften und erfolgreichen Abfragen dieser Feeds 10:1 überschritt. In diesem Fall wurde der Feed als fehlerhaft markiert und die Verarbeitung des Feeds beendet.

3.5.7. Struktur der persistierten Daten

Als Persistenzschicht wurden sowohl das Dateisystem als auch eine Datenbank eingesetzt. Im Dateisystem wurden je Feed eine csv-Datei mit dem rekonstruierten Strom der Einträge sowie pro Abfrage mit mindestens einem neuen Eintrag eine gz-komprimierte Datei mit dem HTTP-Header und dem unverarbeiteten Feed angelegt.

Die Datenbank besteht aus den Tabellen *feeds* und *feed_polls*. Die Tabelle feeds enthält zu jedem der 200.000 Feeds des Datensatzes eine Zusammenfassung aller Abfragen. Hierzu gehören u. a. die Anzahl der erfolgreichen Abfragen, bei denen der Feed erreicht und der Inhalt geparst werden konnte, die Anzahlen der Abfragen, bei denen der Feed entweder nicht erreichbar war oder nicht geparst werden konnte. Weitere wichtige Kenndaten eines Feeds sind die Anzahl der Abfragen, bei denen potentiell Einträge verpasst wurden, d. h. ein Miss verzeichnet wurde, sowie die Summe aller heruntergeladenen Einträge. Neben diesen Daten, die nach jeder Abfrage aktualisiert wurden, erfolgte die Erfassung einer Reihe von Metadaten wie dem Feed-Format sowie, ob verschiedene Attribute wie

[13] `http://java.net/projects/rome/`, Zugriff August 2011

pubDate, cloud oder skipHours und skipDays unterstützt werden. Die Tabelle feeds bildet die Grundlage der statistischen Auswertung des Datensatzes in Abschnitt 3.7.

Die Tabelle feed_polls enthält zu jeder der rund 40 Millionen Abfragen, die während der Erstellung des Datensatzes erfolgten, den Zeitpunkt der Abfrage, Informationen aus dem HTTP-Header sowie die Anzahl der neuen Einträge der Abfrage, die aktuelle Fenstergröße sowie das übertragene Datenvolumen. Aus dem HTTP-Header wurden die Attribute Status Code, ETag, Date, Last-Modified und Expires persistiert. Das übertragene Datenvolumen wurde im Anschluss an die eigentliche Erstellung des Datensatzes aus den gz-Dateien approximiert. Lieferte der Server den Status-Code 304 „not modified“ zurück, so wurde die Größe des HTTP-Headers verwendet, andernfalls die Größe des zuletzt übertragenen Feeds. Im Falle von Fehlermeldungen wie 404 „Not Found“ stellt dies eine minimale Verfälschung dar, die jedoch vernachlässigt werden kann, da in 99,6 % aller Abfragen entweder ein Feed oder der Code 304 zurückgegeben wurde. Die Tabelle feed_polls ist einer der Bausteine für die in den Kapitel 4 und 5 folgenden Evaluationen.

Einen weiteren wichtigen Baustein bilden die csv-Dateien mit den rekonstruierten Strömen der Feed-Einträge. Pro neuem Eintrag wurde eine Zeile mit dem Zeitstempel der Veröffentlichung (3.5.5), dem Zeitstempel der Abfrage, dem eigenen Identifikator des Eintrags (3.5.3), dem Titel, dem Link zum Beitrag sowie der aktuellen Fenstergröße geschrieben. Wurde bei einer Abfrage festgestellt, dass sich das aktuelle und das zuletzt empfangene Fenster des Feeds nicht überlappen, so wurde das potentielle Fehlen von Einträgen in Form einer speziellen Miss-Zeile markiert.

Die pro Abfrage mit mindestens einem neuen Eintrag angelegte, komprimierte Datei mit dem HTTP-Header sowie dem XML-Dokument des Feeds wurde unter Verwendung des vom Server empfangenen Zeichensatzes (engl. encoding) gespeichert. Die Dateien werden im weiteren Verlauf der Arbeit unter anderem verwendet, um die Feeds im Detail zu analysieren und Statistiken zu erstellen. Darüber hinaus enthalten sie zahlreiche Informationen, die anderen Wissenschaftlern und Entwicklern dienlich sein können. Eine Auswertung der Dateien kann dazu beitragen, inhaltliche Analysen der Feeds zu erstellen oder bestehende Feed-Parser zu verbessern – der eingesetzte Rome-Parser konnte z. B. die zurückgegebenen Dokumente bei 0,7 % der Abfragen nicht verarbeiten. Als Primärschlüssel zur Zusammenführung der Datenbank und der Dateien werden eine intern generierte FeedID sowie der Zeitstempel der Abfrage eingesetzt.

3.6. Veröffentlichung des Datensatzes

Die vierte Anforderung an den aufwendig gewonnenen Datensatz ist, dass dieser nicht nur der eigenen Arbeit dienlich sein, sondern auch Anderen zur Verfügung gestellt werden soll (siehe 3.1). Um diesem Anspruch gerecht zu werden, wurde der Datensatz anderen Interessenten aus Forschung und Entwicklung auf der Plattform Areca [UMKS11] als Feed-Item-Dataset-TUDCS6[14] frei zur Verfügung gestellt. Interessenten haben freien Zugriff auf die in Abschnitt 3.5.7 beschriebene Datenbank sowie die csv-Dateien mit den rekonstruierten Strömen der Einträge. Die komprimierten Dateien mit den kompletten

[14] `http://areca.co/10/Feed-Item-Dataset-TUDCS6`, letzter Zugriff August 2011

Inhalten der Feeds konnten aufgrund des Umfangs von rund 100 Gigabyte (GB) nicht direkt zur Verfügung gestellt werden. Als Vorschau wurde ein kleiner Ausschnitt der Dateien des Datensatzes bereitgestellt, der komplette Datensatz ist auf Anfrage verfügbar. Dieses Vorgehen ist üblich, wie andere Datensätze zeigen [G2006, G2008]. Die im aktuellen Kapitel beschriebene Methodologie zur Gewinnung des Datensatzes sowie ein Ausschnitt der Auswertung wurden publiziert [RUM+11].

3.7. Statistische Auswertung und Fehlerbetrachtung

Die Aufzeichnung der Historie erfolgte für die zufällig ausgewählten 200.000 Feeds (Liste 7, S. 85) über einen Zeitraum von vier Wochen vom 08.07.2011 bis 05.08.2011. Es erfolgten rund 40 Millionen Abfragen, die rekonstruierten Ströme der Einträge umfassen in der Summe ca. 54 Millionen Einträge. Bei mehr als 6 Millionen Abfragen wurden neue Einträge gefunden und die empfangenen XML-Dokumente sowie die HTTP-Header in einer gz-Datei gespeichert. Der Datensatz hat eine unkomprimierte Größe von ca. 400 GB respektive 100 GB in komprimierter Form. Inklusive der Vorarbeiten und Durchführung nahm die Erstellung des Datensatzes rund sechs Monate in Anspruch, wobei ein Großteil des entwickelten Quellcodes in der Evaluation in Kapitel 4 wiederverwendet wird.

Alle in den nachfolgenden Abschnitten betrachteten Auswertungen beziehen sich auf diese Daten. Im Fokus der Betrachtung stehen verschiedene Verteilungen wie die der Feed-Formate, Fenstergrößen, Anzahl von Einträgen pro Feed sowie die Ermittlung und Verteilung der Aktualisierungsmuster. Eine Analyse der Grenzen des Datensatzes sowie des Grads der Erfüllung der Anforderungen runden das Kapitel ab.

Zum besseren Verständnis der Abschnitte sei auf zwei Dinge hingewiesen: die Verwendung von Begrifflichkeiten und der Einfluss des Zeitraums, in dem die Erstellung des Datensatzes erfolgte. In den Auswertungen werden häufig sowohl das arithmetische Mittel als auch der Median betrachtet. Die Begriffe arithmetisches Mittel und Durchschnitt werden synonym verwendet. Ist der Median gemeint, so wird dies explizit angegeben. Der in Abschnitt 3.4 dargelegte Ablauf zum Auffinden der Feed URLs hat einige Wochen in Anspruch genommen, sodass zwischen der Ermittlung der rund 1,3 Millionen erreichbaren Feeds (Liste 5), der zufälligen Auswahl der 200.000 Feeds (Liste 7) und dem tatsächlichen Start der Aufzeichnung eine gewisse Zeit vergangen ist. Aus diesem Grund waren einige Hundert Feeds zum Start der Aufzeichnung nicht mehr erreichbar. Für diese konnten keine Daten erhoben werden, sodass die Statistiken stets einen sehr kleinen Anteil Feeds mit unbekannten Merkmalsausprägungen enthalten.

3.7.1. Verteilung der Feed-Formate

Abbildung 3.6 zeigt die Verteilung der Feed-Formate des gewonnenen Datensatzes. RSS 2.0 ist mit einem Anteil von 75 % ganz eindeutig das dominierende Format, gefolgt von Atom 1.0 mit einem Anteil von 21 %. RSS 1.0 liegt mit weniger als 4 % auf Platz 3, die anderen Formate haben jeweilige Anteile von maximal 0,5 % und sind somit kaum eine praktische Relevanz. Bei 43 Feeds (0,02 %) konnte der eingesetzte Rome-Parser kein Format bestimmen.

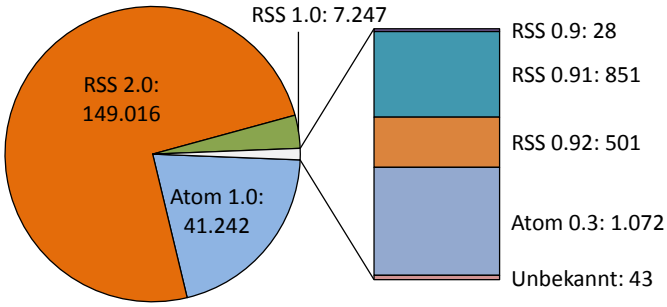

Abbildung 3.6.: Verteilung der Feed-Formate.

3.7.2. Verteilung der Fenstergrößen

Abbildung 3.7 zeigt die Verteilung der Fenstergrößen aller Feeds. Die Werte wurden nach der Aufzeichnung der Feeds ermittelt, sodass Feeds mit veränderlichen Größen mit ihrem letzten Wert eingegangen sind. Es ist eine markante Spitze bei einer Fenstergröße von zehn Einträgen zu erkennen, die zugleich zwei verschiedene Verteilungsmuster gegeneinander abgrenzt. Oberhalb von zehn Einträgen pro Feed sind weitere Spitzen bei 15, 20 und 25 Einträgen zu erkennen, die sich deutlich von den dazwischen liegenden Größen unterscheiden. Dieses Muster setzt sich mit zunehmender Fenstergröße fort, weitere Spitzen liegen bei 50, 100 und 1.000 Einträgen. Diese wurden in der Abbildung aggregiert und als „>25“ dargestellt. Bei den Fenstergrößen unterhalb von zehn Einträgen dominieren leere Feeds mit keinem Eintrag sowie Feeds mit nur einem einzigen Beitrag, die zusammen 18 % des Datensatzes ausmachen. Die durchschnittliche Fenstergröße beträgt 17, der Median liegt bei zehn, das größte beobachtete Fenster umfasste 4.564 Einträge.

Die zunächst ungewöhnlich erscheinende Verteilung kann wie folgt interpretiert werden. Die Fenstergröße ist ein Parameter, der von der Konfiguration des Feed-Anbieters abhängig

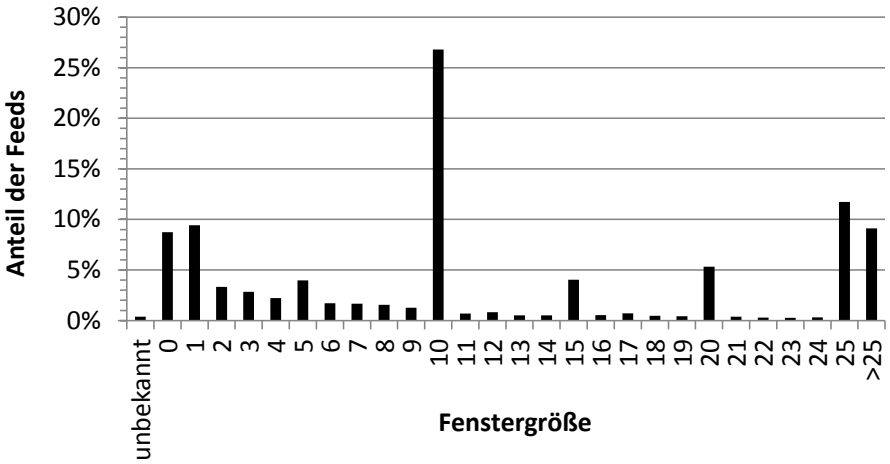

Abbildung 3.7.: Verteilung der Fenstergrößen.

ist. Diese erfolgt entweder manuell oder es wird ein Standardwert verwendet. In beiden Fällen wurde sie von einem Menschen gesetzt, weshalb häufig runde oder anderweitig markante Werte gewählt werden.

Der hohe Anteil Feeds mit keinem oder nur einem Eintrag ist auf den Umstand zurückzuführen, dass Systeme zur Gestaltung von Weblogs sowie Content-Management-Systeme eine Unterstützung für Feeds bieten, die in der Standardkonfiguration aktiviert ist und einen leeren Feed bzw. einen „Hello World"-Eintrag enthält. Eine zufällige Auswahl von Feeds dieser Fenstergröße und eine manuelle Überprüfung des Inhalts bestätigte diese Annahme.

Bei der Analyse des Datensatzes konnte festgestellt werden, dass ca. 10 % der Feeds variierende Fenstergrößen aufwiesen. Da die Fenstergröße ein beobachteter Parameter ist und nicht direkt als Attribut im Feed enthalten ist, muss geschlussfolgert werden, dass der Wert eine untere Schranke für die Anzahl von Feeds mit variabler Fenstergröße darstellt. Es ist davon auszugehen, dass der Wert steigt, je größer der Beobachtungszeitraum gewählt wird. Innerhalb der vier Wochen, in denen der Datensatz erstellt wurde, wiesen rund 40 % der Feeds neue Einträge auf, die nicht in der ersten Abfrage enthalten waren. Extrapoliert man den Anteil Feeds mit variabler Fenstergröße, so kann geschätzt werden, dass rund ein Viertel aller Feeds eine variable Fenstergröße aufweist.

Die Variationen weisen zudem drei verschiedene Ausprägungen auf. Bei einigen Feeds ist die Fenstergröße direkt proportional zur Anzahl veröffentlichter Einträge – mit jedem neuen Eintrag wird sie um eins erhöht, bestehende Einträge werden nicht entfernt. Das zweite Muster sind Feeds, deren Fenstergröße zwischen einer Vielzahl von Werten schwankt. Dieses Verhalten ist zu beobachten, wenn der Feed nur Einträge enthält, die in einem bestimmten Zeitraum wie der aktuellen Woche oder den letzten 24 Stunden veröffentlicht wurden. Das dritte Muster sind Feeds, deren Fenstergröße zwischen zwei festen Werten alterniert, wobei einer der Werte meist Null oder Eins ist und der andere Wert die tatsächliche Fenstergröße. Die Ursache für dieses Verhalten konnte nicht erschlossen werden.

3.7.3. Verteilung der Datengrößen

Die Betrachtung der Datengrößen ist ein Anhaltspunkt für die im System des Abfragenden entstehende Last zur Verarbeitung der Feeds. Es werden sowohl die Feeds selbst als auch die HTTP-Header betrachtet. Bei der Ermittlung der Datengrößen wurde nur die Größe des zurückgegebenen XML-Dokuments gemessen, eingebundene externe Skripte oder Medienobjekte wie Bilder sind nicht in die Berechnung eingeflossen, da diese nicht bei jeder Abfrage, sondern nur dann übertragen werden müssen, wenn der Feed tatsächlich einen neuen Eintrag aufweist. Bei der Ermittlung, ob ein Feed über einen neuen Eintrag verfügt, können diese ignoriert werden. Diese Einschränkung beruht auf der Annahme, dass sich die Änderung eines Feeds oder gar eines Eintrags direkt im XML-Dokument widerspiegelt und nicht nur ein per Referenz eingebundenes Objekt geändert wird. Um unverfälschte Werte zu ermitteln, wurden alle Feeds exakt einmal abgefragt, die in Abschnitt 3.5.6 eingeführte Begrenzung der Größe auf ein MB wurde nicht angewendet.

Abbildung 3.8 zeigt das Histogramm der Datengrößen des Datensatzes. Die Größen variieren zwischen 40 Bytes und drei MB und ähneln einer Potenzfunktion mit negativem Exponenten. Knapp ein Viertel der Feeds hat eine Größe von maximal fünf kB, die Hälfte

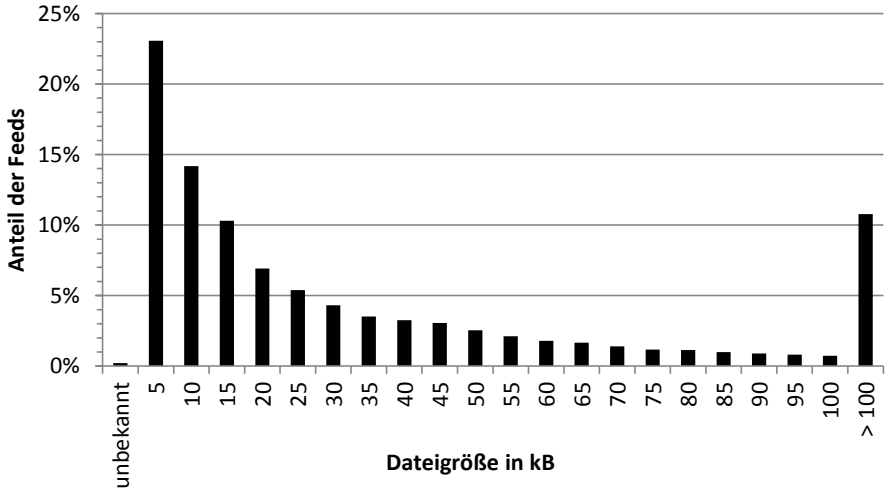

Abbildung 3.8.: Verteilung der Datengrößen.

der Feeds (Median) hat eine maximale Größe von 17 kB und ca. elf Prozent sind größer als 100 kB, sodass die durchschnittliche Größe 43 kB beträgt. Vergleicht man diese Werte mit Messungen[15] aus dem Jahr 2004, so hat sich die durchschnittliche Größe innerhalb von sieben Jahren verdoppelt. Der zum Vergleich herangezogene Datensatz ist jedoch deutlich kleiner, sodass der Vergleich nur als Anhaltspunkt verwendet werden kann.

Interessanterweise sind Feeds im Mittel größer als Webseiten. Eine durchschnittliche Webseite hatte im Jahr 2008 eine Größe von 25 kB [Kin08], 1999 waren es im arithmetischen Mittel 19 kB (Median 3 kB) [LG99], wobei auch hier jeweils nur die HTML-Dokumente ohne Berücksichtigung eingebetteter Objekte betrachtet wurden.

Die HTTP-Header sind erwartungsgemäß um mehrere Größenordnungen kleiner, betragen im arithmetischen Mittel 369 Byte und weisen eine deutlich kleinere Streuung als die Datengrößen der Feeds auf, sodass Werte über 700 Byte sehr selten sind. Unterstützt ein Server bedingte GET-Abfragen (siehe Abschnitt 2.3.5), so wird ein unverändertes Dokument nicht erneut übertragen, sondern nur ein entsprechender Header mit dem Status-Code 304 „not modified“ zurückgegeben. Bei der Erstellung des Datensatzes enthielten 44 % der empfangenen Antworten diesen Status-Code.

3.7.4. Verteilung der Anzahl von Einträgen

Abbildung 3.9 zeigt die Korrelation zwischen der Anzahl von Einträgen und der Anzahl von Feeds. Auf der Abszisse ist die Summe aller Einträge eines Feeds dargestellt, dem gegenüber stehen auf der Ordinate die jeweilige Anzahl von Feeds, welche diese Summe von Einträgen innerhalb von vier Wochen aufwiesen. Die Verteilung zeigt typische Merkmale einer Potenzfunktion mit negativem Exponenten. Wie im unteren Bereich der Grafik erkennbar ist, weist die Anzahl von Einträgen je Feed eine sehr hohe Streuung auf. Im

[15] `http://www.allthingsdistributed.com/historical/archives/000435.html`

arithmetischen Mittel hat ein Feed 289 Einträge, der Median liegt hingegen bei nur 12 Einträgen, der am häufigsten aktualisierte Feed hatte mehr als 12 Millionen neue Einträge innerhalb von vier Wochen.

Die Analyse der Ausreißer zeigt, dass diese zumeist bei markanten Fenstergrößen von 1, 10, 25, 50 bzw. 100 Einträgen liegen. Diese Feeds wiesen innerhalb des Beobachtungszeitraums keine neuen Einträge auf. Aufgrund der logarithmischen Darstellung fehlen in Abbildung 3.9 zwei Datenpunkte. 17.040 Feeds hatten eine konstante Fenstergröße von null und wiesen im kompletten Zeitraum keinen einzigen Eintrag auf, für 405 nicht erreichbare Feeds konnte kein Wert ermittelt werden.

Der Vergleich des Datensatzes mit dem knapp 10.000 Feeds umfassenden Datensatz in [SCC07] zeigt, dass beide eine äquivalente Verteilung aufweisen. Die Autoren ermittelten $y = ax^b$, $a \simeq 376$ und $b \simeq -0.78$ als Approximation ihrer Verteilung. Um die Verteilung der Anzahl von Einträgen in Abbildung 3.9 ebenfalls mit dieser Potenzfunktion zu beschreiben, wurden die Werte $a \simeq 677$ und $b \simeq -0.74$ ermittelt. Die Exponenten sind nahezu identisch, der in etwa doppelt so große Wert des Parameters a ist auf den unterschiedlichen Umfang der Datensätze zurückzuführen, sodass sich die Verteilungen in ihren absoluten Werten unterscheiden.

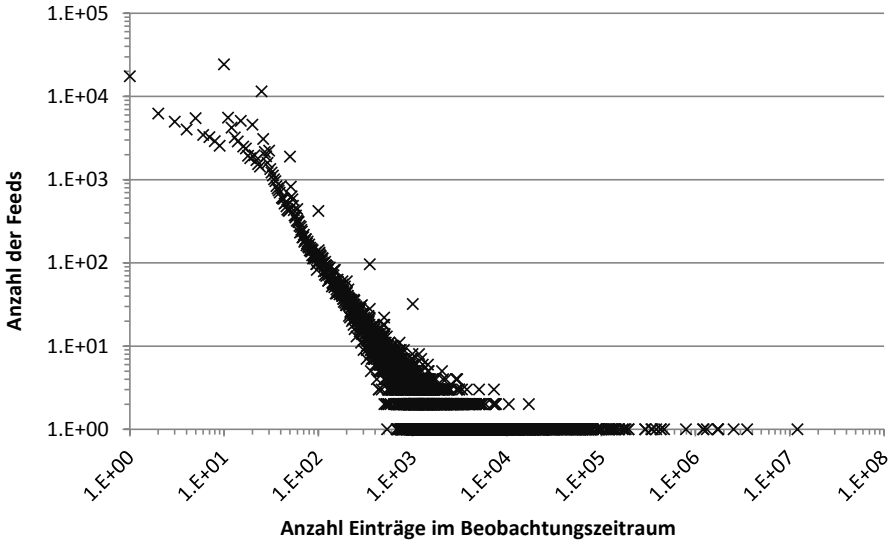

Abbildung 3.9.: Verteilung der Anzahl von Einträgen.

3.7.5. Ermittlung der Aktualisierungsmuster

Aus den vorangegangenen Abschnitten wird erkennbar, dass sich Feeds in den betrachteten Kenngrößen deutlich unterscheiden. Im Hinblick auf die Vorhersage von Aktualisierungen wird nun untersucht, inwiefern sich unterschiedliche Aktualisierungsmuster identifizieren lassen. Hierfür wurde zunächst eine zufällige Auswahl der Feeds manuell betrachtet und aus den Erkenntnissen acht Muster abgeleitet. Die Aktualisierungsmuster weisen partielle

Überlappungen auf, sodass es nicht immer möglich ist, einen Feed klar einem Muster zuzuordnen. Zudem können Feeds ihr Verhalten im Laufe der Zeit ändern. Die Muster tragen dennoch zum besseren Verständnis der Feeds bei und begünstigen die Entwicklung von Strategien zur Vorhersage von Änderungen [URM+11]. Diese Muster sind orthogonal zu den in [FMNW03] vorgestellten Klassen, denen ein Dokument in Abhängigkeit von der anteiligen Änderung seines Inhalts zugeordnet wird (siehe Abschnitt 2.4.2). Zur Vorhersage werden die Muster der Änderungsraten betrachtet, die bei Webseiten zu beobachtenden Muster partieller Änderungen des Inhalts werden bei Feeds aufgrund des FIFO-Verhaltens als nicht relevant betrachtet.

Die acht Muster sind in Abbildung 3.10 dargestellt. Auf jeder der Zeitachsen repräsentieren die Markierungen die Zeitpunkte, zu denen ein neuer Eintrag im Feed erscheint. Die Zuordnung von Feeds zu Mustern erfolgte regelbasiert auf Basis der Zeiträume zwischen den Einträgen. Hierzu wurden klassische Größen wie das durchschnittliche und das Median Intervall, die Standardabweichung, die Zeitdifferenz zwischen letzter Abfrage und neustem Eintrag sowie die durchschnittliche Anzahl neuer Einträge pro Tag verwendet.

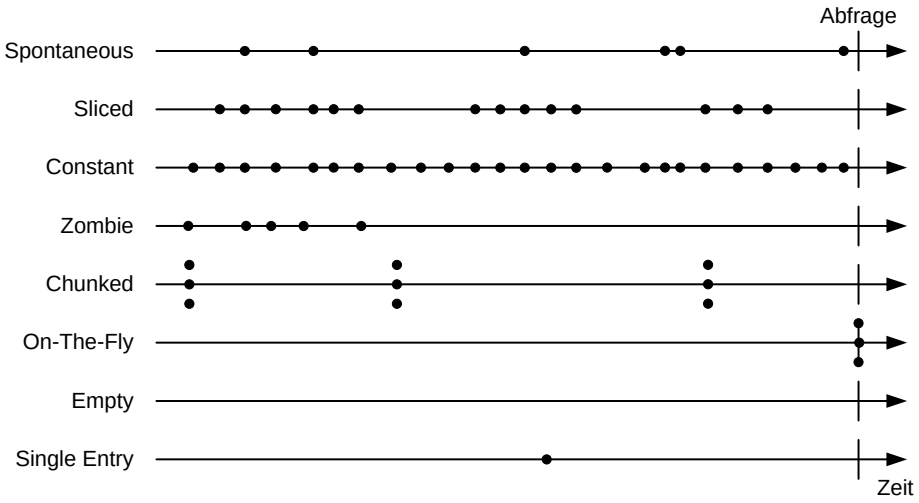

Abbildung 3.10.: Aktualisierungsmuster der Feeds.

Spontaneous In diesem Muster weist die Länge der Zeiträume zwischen zwei Einträgen eine hohe Streuung auf und neue Einträge sind im Mittel (Median) täglich oder seltener zu verzeichnen. Diese Feeds gehören häufig zu selten aktualisierten Blogs mit nur einem Autor.

Sliced Diese Feeds weisen zu bestimmten Zeiten ein erhöhtes Vorkommen an neuen Einträgen auf, während zu anderen Zeiten nur sehr geringe Aktivitäten zu verzeichnen sind. Dieses Verhalten ist z. B. bei kleineren Nachrichtenagenturen zu beobachten, die tagsüber ein relativ hohes Aufkommen neuer Einträge aufweisen, jedoch über Nacht nur wenige oder keine neuen Einträge veröffentlichen.

Constant Werden in einem Feed zu jeder Tages- und Nachtzeit nahezu konstant viele Einträge veröffentlicht, so wird dieser Feed als Constant bezeichnet. Dieses Verhalten ist unter anderem bei international agierenden Nachrichtenagenturen wie BBC und Reuters und den Feeds zu den letzten Änderungen der Wikipedia zu beobachten. Die Zeitdifferenz zwischen zwei Einträgen darf zwei Stunden nicht überschreiten.

Zombie Der Feed ist noch erreichbar, die letzte Aktivität liegt jedoch weit in der Vergangenheit. Einem Feed wird dieses Muster zugewiesen, wenn der Zeitpunkt des neuesten Eintrags mindestens acht Wochen vor dem der letzten Abfrage liegt und dieser Zeitraum mindestens um den Faktor acht größer ist als der Median des Aktualisierungsintervalls des Feeds. Angenommen, ein Feed wird alle zwei Wochen aktualisiert, so muss der letzte Eintrag mindestens 16 Wochen in der Vergangenheit liegen, damit der Feed als Zombie bezeichnet wird.

Chunked Einige Feeds veröffentlichen alle Einträge eines Fensters zum selben Zeitpunkt, sodass alle Einträge den gleichen Zeitstempel aufweisen. Der Zeitpunkt der Veröffentlichung kann in der Vergangenheit liegen.

On-The-Fly Feeds dieses Musters generieren den Inhalt dynamisch zur Abfragezeit und setzten das Veröffentlichungsdatum aller Einträge auf den aktuellen Zeitpunkt. Bei der Identifikation muss beachtet werden, dass die serverseitige Generierung des Dokuments unter Umständen mehrere hundert Millisekunden – bei großen Dokumenten gar mehrere Sekunden – dauern kann, sodass sich die Zeitstempel um wenige Sekunden unterscheiden können. Zudem kann es vorkommen, dass die Systemuhren des Abfragenden und des Servers nicht synchron laufen. Um einem Feed das Muster On-The-Fly zuzuordnen, muss der Median der Zeiträume zwischen allen Einträgen eines Fensters Null sein, zudem darf die Differenz zwischen dem Zeitstempel des neuesten Eintrags und dem Wert des Date-Feld des HTTP-Headers maximal eine Sekunde betragen. Steht das Date-Feld nicht zur Verfügung, wird alternativ mit dem Zeitpunkt der Abfrage verglichen.

Empty Hat ein Feed keinen Eintrag, so wird er diesem Muster zugeordnet.

Single Entry Enthält ein Feed nur einen einzelnen Eintrag, so kann – wie bei Feeds ohne Eintrag – kein richtiges Aktualisierungsmuster ermittelt werden, sodass diese beiden Muster eingeführt wurden.

3.7.6. Verteilung der Aktualisierungsmuster

Abbildung 3.11 zeigt die Verteilung der Aktualisierungsmuster des erstellten Datensatzes. Zur Identifikation der Muster wurden alle Einträge des Beobachtungszeitraums von vier Wochen berücksichtigt. Einem Drittel der Feeds kann das Muster *Spontaneous* zugeordnet werden, da sie unregelmäßige Aktualisierungen aufweisen. Ein weiteres Viertel wurde als *Zombie* identifiziert. Abgesehen von der ersten Abfrage, in der per Definition der komplette Fensterinhalt neu ist, wies keiner der Feeds innerhalb dieses Zeitraums einen neuen Eintrag auf. Das Muster *Sliced* konnte 18 % der Feeds zugeordnet werden. Sie

zeigen zyklische Unterschiede zwischen Tag- und Nachtzeiten. Die beiden Muster *Single Entry* und *Empty* sind beinahe gleich häufig vertreten. Da die beiden Muster anhand der Fenstergröße der Feeds identifiziert werden, sind sie auch in Abbildung 3.7 zu erkennen. Die Anteile der Feeds, in denen neue Einträge 24 Stunden am Tag und 7 Tage die Woche *(Constant)* veröffentlicht werden, ist relativ gering. Selbiges trifft auf Feeds des Musters *Chunked* zu. On-The-Fly Feeds, deren Einträge und Zeitstempel zur Abfragezeit generiert werden, sind nur sehr selten. 0,2 % der Feeds waren nie erreichbar, sodass kein Muster zugeordnet werden konnte.

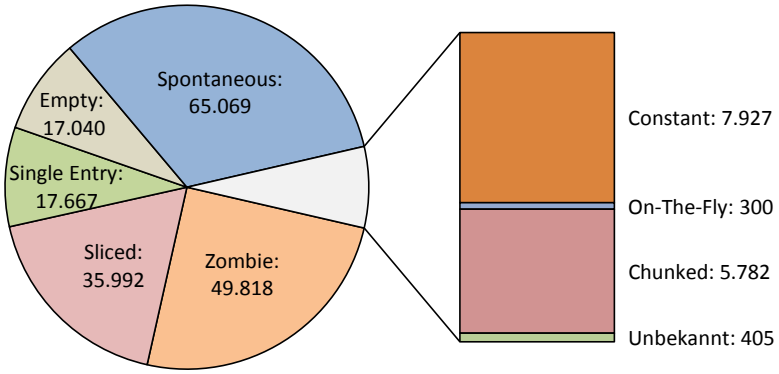

Abbildung 3.11.: Verteilung der Aktualisierungsmuster.

Die Betrachtung der Verteilung der Aktualisierungsmuster zeigt, dass nur ein kleiner Anteil der Feeds konstant und häufig neue Einträge aufweist. Konstante Abfrageintervalle von einer Stunde, wie sie häufig in Feed Readern eingesetzt werden, führen zu einer Vielzahl von Antworten, die keinen neuen Inhalt enthalten. Kann ein Feed z. B. als *Zombie* identifiziert werden, so können das Abfrageintervall und folglich der Bedarf an Ressourcen signifikant gesenkt werden.

3.7.7. Untersuchung einzelner Feed-Attribute

In Abschnitt 2.3.1 wurden die Feed-Formate und deren Eigenschaften vorgestellt. In Abschnitt 3.5 wurde einigen Attributen wie pubDate, updated und id besondere Aufmerksamkeit geschenkt, da diese zur Erstellung des Datensatzes ausgewertet wurden, um Duplikate zu erkennen oder das Aktualisierungsintervall eines Feeds zu ermitteln. Es wurde bereits vorweg genommen, dass einige obligatorische Attribute nicht von allen Feeds unterstützt werden. Die detaillierte Analyse des Datensatzes wird an dieser Stelle nachgeholt.

Identifikatoren In 80 % der Feeds konnte einer der Identifikatioren id oder guid gefunden werden, die einen Eintrag eindeutig identifizieren. Für den Umkehrschluss muss beachtet werden, dass knapp 9 % der Feeds dem Muster *Empty* entsprechen und keine Einträge aufweisen, sodass für diese Feeds nicht bestimmt werden kann, ob sie Identifikatoren bereitstellen. Reichlich 11 % der Feeds enthalten Einträge ohne Identifikatoren. Aus diesem Grund wurde zur Erkennung von Duplikaten auf weitere Attribute zugegriffen.

Veröffentlichungsdatum Das Veröffentlichungsdatum bzw. Datum der letzten Aktualisierung eines Eintrags steht je nach Format in einem der Attribute pubDate, updated oder published. Knapp 11 % der Feeds enthalten Einträge, die kein Datum enthalten bzw. deren Datum keinem konventionellen Format entspricht und somit nicht auswertbar ist. Für diese Feeds wurde als Heuristik der Zeitpunkt der Abfrage als Veröffentlichungsdatum angenommen und für die Vorhersage neuer Einträge eingesetzt.

Wenngleich nur knapp 11 % der Feeds kein Datum aufweisen, so muss festgestellt werden, dass rund 40 % aller Einträge des Datensatzes kein Datum aufweisen. Dies ist auf die in Abschnitt 3.7.4 untersuchte Verteilung der Anzahl Einträge zurückzuführen. Einige der Feeds mit der größten Anzahl neuer Einträge verfügen über keine Angabe eines Veröffentlichungsdatums. Dieser Verschiebung wird in Kapitel 4 mit verschiedenen Modi der Evaluation Rechnung getragen.

ttl (RSS 2.0) Das Attribut ttl sowie die nachfolgend betrachteten Attribute sind RSS 2.0-spezifisch und können für adaptive Abfragen ausgewertet werden. Die Verteilung der ttl-Werte ist in Abbildung 3.12 dargestellt. Es zeigt den ttl-Wert auf der Abszisse und die Anzahl der Feeds auf der Ordinate. Auf den ersten Blick weist das Diagramm eine ähnliche Charakteristik wie die Verteilung der Fenstergrößen in Abbildung 3.7 auf.

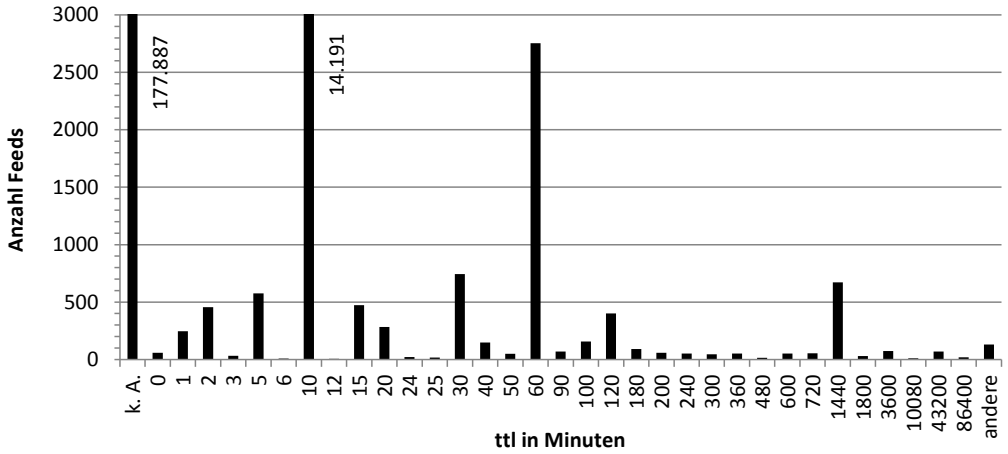

Abbildung 3.12.: Verteilung der time-to-live-Werte.

Um eine übersichtliche Darstellung zu erreichen, wurden zwei „Ausreißer“ verkürzt dargestellt und die entsprechenden Werte in der Grafik ergänzt. 89 % der Feeds enthalten kein ttl-Attribut. Rund 7 % aller Feeds enthalten einen Wert von 10 Minuten, für ca. 1,4 % aller Feeds ist eine Lebensdauer von einer Stunde angegeben. Die verbleibenden Werte entsprechen typischen runden Werten wie 15 und 30 Minuten sowie Vielfachen einer Stunde mit einer letzten Spitze bei einem Tag (1440 Minuten). Sehr selten vorkommende ttl-Werte wurden als Rauschen unter dem Punkt „andere“ zusammengefasst.

Werden die ermittelten Werte nicht zu allen 200.000 Feeds in Relation gestellt, sondern ausschließlich die Untermenge der rund 22.000 Feeds betrachtet, die das

Attribut ttl aufweisen, ist festzustellen, dass knapp zwei Drittel dieser Feeds einen Wert von zehn Minuten und ein weiteres Achtel einen Wert von einer Stunde enthalten. Diese Erkenntnis sowie die auffällig häufig vorkommenden runden Werte unterstreichen die Vermutung, dass der Parameter händisch von den Diensteanbietern eingestellt wird und kein Bezug zum tatsächlichen Aktualisierungsintervall des jeweiligen Feeds besteht. Diese Vermutung wurde in der dritten These dieser Arbeit (auf Seite 16) geäußert und wird in Abschnitt 5.2 im Detail untersucht.

cloud Das Attribut cloud wird nur von ca. 3,7 % der Feeds unterstützt. Es sei darauf hingewiesen, dass aus dem Vorhandensein des Attributs nicht geschlossen werden darf, dass der referenzierte Web-Service tatsächlich existiert und registrierte Nutzer über Aktualisierungen informiert. Aufgrund der geringen Verbreitung wurde von einer aufwendigen Überprüfung abgesehen, ob die Web-Services tatsächlich existieren und die erwartete Funktionalität bereitstellen.

PubSubHubbub Zur Untersuchung der Verbreitung des in Abschnitt 2.7.2 vorgestellten PubSubHubbub-Protokolls [Pub11] wurde jeder Feed auf die Existenz des PubSubHubbub-spezifischen Link-Attributs überprüft. Das Attribut konnte nur in knapp 7 % der Feeds des Datensatzes gefunden werden. Die bezüglich des cloud-Attributs getroffene Aussage gilt analog.

skipHours und skipDays Die Attribute skipHours und skipDays sind mit 0,05 % respektive 0,04 % Verbreitung in realen Feeds faktisch nicht relevant. Der potentielle Mehrwert der Attribute zur Vorhersage von Aktualisierungen wird in Kapitel 5 untersucht.

Jeweils ca. 11 % der Feeds enthalten keine Identifikatoren bzw. kein Veröffentlichungsdatum ihrer Einträge. Der Anteil der Feeds, die das Attribut ttl unterstützen, beträgt ebenfalls 11 %, wobei die Werte zehn Minuten und eine Stunde am häufigsten sind. cloud, skipHours und skipDays werden nur sehr selten verwendet.

3.7.8. Analyse verpasster Einträge

Die Anzahl der potentiellen Lücken im Datensatz ist eines der Qualitätskriterien, die in Abschnitt 3.1 aufgestellt wurden. An dieser Stelle sei an Gleichung 3.1 (auf Seite 86) erinnert. Eine potentielle Lücke wird immer genau dann festgestellt, wenn die Schnittmenge zweier aufeinanderfolgender, nicht leerer Fenster keinen Eintrag enthält, sodass nicht ermittelt werden kann, ob etwas verpasst wurde oder nicht.

In Abbildung 3.13 ist die Anzahl potentieller Lücken auf der Abszisse der Anzahl von Feeds auf der Ordinate gegenübergestellt, die höchstens so viele Lücken aufweisen. Für 94 % der Feeds wurde kein einziger Eintrag verpasst, 2 % der Feeds weisen genau eine und ein weiteres Prozent der Feeds zwei potentielle Lücken auf, sodass 97 % der Feeds maximal je zwei Lücken enthalten. Vergleicht man dies mit der hohen Anzahl neuer Einträge, dem Abfrageintervall von maximal sechs Stunden und der Länge des Beobachtungszeitraums, so unterstreichen diese Werte die hohe Qualität des Datensatzes.

Die Grafik weist eine auffällige Stufe im Bereich von 28 potentiellen Lücken auf. Es wird angenommen, dass ca. 150 Feeds des Datensatzes genau einmal täglich generiert

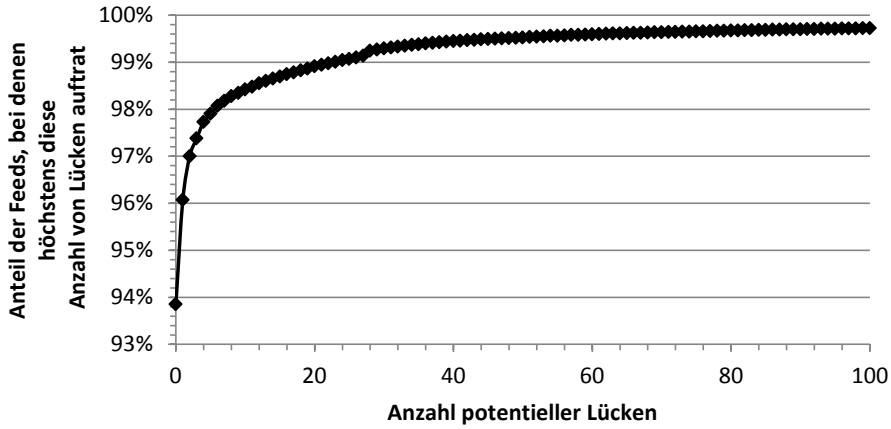

Abbildung 3.13.: (kumulative) Verteilungsfunktion der Anzahl potentieller Lücken über den Datensatz.

werden, die den Feeds zugrunde liegenden Daten sich jedoch häufiger ändern, sodass aufeinanderfolgende Fenster disjunkt sind. Rund 50 dieser Feeds kann das Muster *Single Entry* zugeordnet werden, von denen viele einen „Tipp des Tages“ enthalten. Diese potentiellen Lücken sind nicht vermeidbar.

Die Ursachen für potentielle Lücken sind auf die Problematik fehlerhafter Datumsangaben (3.5.5), das gewählte Abfrageintervall und dessen Schranken (3.5.4) sowie die aufwendige Erkennung von duplizierten Einträgen (3.5.3) zurückzuführen. Diese wurden im jeweiligen Abschnitt im Detail ergründet.

3.8. Grad der Erfüllung der Anforderungen

In Abschnitt 3.1 wurden vier Anforderungen an den zu erstellenden Datensatz aufgestellt, diese werden nun rekapituliert und auf ihre Erfüllung überprüft.

Real Diese Anforderung gilt als erfüllt, wenn die Daten tatsächlich existierender Feeds aufgezeichnet werden. Aus diesem Grund wurden URLs realer Feeds ermittelt (3.4) und anschließend deren Historie aufgezeichnet (3.5). Der gewonnene Datensatz ist real und die Anforderung ist erfüllt.

Breitgefächert Der erstellte Datensatz ist breitgefächert, da bei der Ermittlung der Feeds (3.4) keine einzelne Domäne bevorzugt wurde. Der Datensatz enthält zudem eine Vielzahl von Sprachen. Auf Grund der sehr breit angelegten Ermittlung der Feeds und der Größe des Ausschnitts von 200.000 Feeds wird davon ausgegangen, dass der Datensatz einen realistischen Ausschnitt der vorkommenden Aktualisierungsmuster umfasst. In der Phase der Aufzeichnung der Historie (3.5) wurden alle Feeds einheitlich verarbeitet, womit die Anforderung nach einem breitgefächerten Datensatz als erfüllt gilt.

Vollständig Aufgrund der zahlreichen Faktoren, die zur Erfüllung dieser Anforderung berücksichtigt werden müssen, ist diese am schwierigsten umzusetzen. Der Datensatz gilt als vollständig, wenn entweder pro Feed eine lückenlose Historie der Einträge vorliegt oder potentielle Lücken als solche erkennbar sind. Zur Umsetzung der vollständigen Historie wurden alle Feeds regelmäßig abgefragt, wobei das Intervall adaptiv an das beobachtete Aktualisierungsintervall des Feeds angepasst wurde. Potentielle Lücken wurden in csv-Dateien mit den rekonstruierten Historien als solche markiert. Die Voraussetzung, Lücken zu erkennen, ist ein FIFO-Verhalten der Feeds (3.5.2). Dies ist nicht generell gegeben, es kann jedoch nicht immer erkannt werden, wenn der Feed diese Voraussetzung nicht erfüllt. Unter Berücksichtigung dieser Einschränkung gilt die Anforderung der Vollständigkeit dennoch als erfüllt.

Nachnutzbar Der Datensatz kann von anderen genutzt werden, wenn er gut dokumentiert, in einem gebräuchlichen Format persistiert und öffentlich zugänglich ist. Zur Erfüllung dieser Anforderung wurden die Daten in dem in Abschnitt 3.5.7 beschriebenen Format gespeichert und anschließend auf einer Plattform für wissenschaftliche Datensätze zur Verfügung gestellt. Die Dokumentation erfolgte im Rahmen von [RUM+11], sodass die vierte Anforderung ebenfalls als erfüllt betrachtet wird.

Aus der Summe der einzelnen Betrachtungen folgt, dass alle vier aufgestellten Anforderungen erfüllt wurden. Der in diesem Kapitel gewonnene Datensatz sowie die Analysen der Daten bilden die Grundlage für die folgenden Kapitel, in denen der Zeitpunkt des Auftretens eines neuen Eintrags auf Basis der Historie des Feeds vorhergesagt wird.

4. Vorhersage von Änderungen anhand der Historie

Aufbauend auf den in Kapitel 2 herausgearbeiteten Algorithmen und dem in Kapitel 3 beschriebenen Datensatz erfolgt in diesem Kapitel die Vorhersage von Änderungen anhand der Historie. Das Kapitel umfasst drei wissenschaftliche Beiträge: Aus den in Kapitel 2 analysierten Algorithmen wird ein neuer Algorithmus entwickelt, der die Stärken mehrerer Algorithmen kombiniert. Anschließend werden eine Methodik und Metriken entwickelt, mit denen verschiedene Aspekte der Vorhersage wie die Anzahl benötigter Abfragen und der Anteil verpasster Einträge beurteilt werden können. Diese werden im dritten Schwerpunkt eingesetzt, um den eigenen Algorithmus mit einer Vielzahl gängiger Algorithmen aus verwandten Arbeiten anhand eines realen, breitgefächerten Datensatzes zu vergleichen. Die erzielten Ergebnisse werden abschließend im Hinblick auf das betrachtete Szenario der Integration von Produktinformationen in ein PIS diskutiert. Das identifizierte Optimierungspotential wird im nachfolgenden Kapitel 5 untersucht.

4.1. Architektur eines Feed Data Providers

Das Ziel der Integration von Feeds in ein PIS ist, die in den Feeds enthaltenen Produktinformationen zeitnah dem integrierenden System zur Verfügung zu stellen und dabei möglichst keine Einträge zu verpassen. Hierfür wird die in Abbildung 4.1 dargestellte Architektur eines *Feed Data Providers* vorgeschlagen.

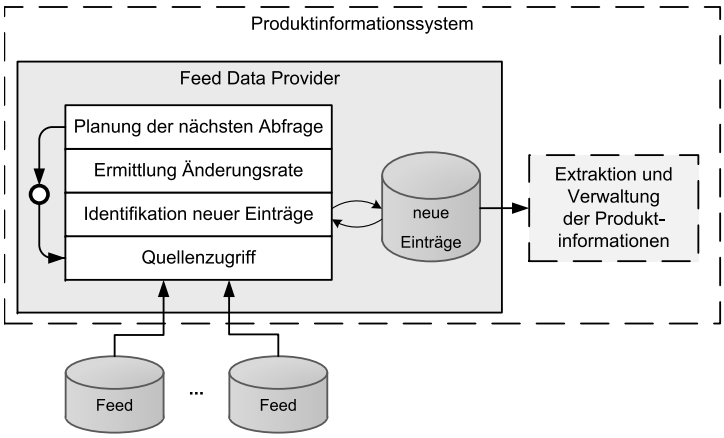

Abbildung 4.1.: Integration eines Feed Data Providers in ein Produktinformationssystem.

Ein Feed Data Provider besteht aus vier Schichten, die dem Quellenzugriff, der Identifikation neuer Einträge, der Ermittlung der Änderungsrate sowie der Planung der nächsten Abfrage dienen. Aus Sicht des PIS sind nur die neuen Einträge von Interesse, die dem PIS in geeigneter Weise vom Feed Data Provider zur Verfügung gestellt werden. Das PIS hat jederzeit Zugriff auf die neuen Einträge, sodass bereits vorhandene Mechanismen zur Integration der neuen Informationen eingesetzt werden können, um zu entscheiden, wann welche Einträge in welcher Reihenfolge in das PIS integriert werden. Der Fokus der Arbeit liegt auf dem Feed Data Provider, die in Abbildung 4.1 zusammengefassten Komponenten des PIS wurden bereits in Abbildung 1.1 eingeführt, sie sind nicht Gegenstand der Arbeit.

4.2. Algorithmen zur Vorhersage von Änderungen anhand der Historie

Im Folgenden werden die in Kapitel 2 vorgestellten Algorithmen aufgegriffen und um den eigenen, MAVSync genannten Algorithmus ergänzt, in dem die Vorteile verschiedener Verfahren kombiniert werden. Die Algorithmen werden in drei Gruppen unterteilt: jene mit statischem Intervall (4.2.2), adaptive Algorithmen mit begrenzter Historie (4.2.3) und auf dem Poisson-Modell basierende Verfahren (4.2.4).

Da einige Algorithmen aus anderen Domänen übernommen werden und daher nicht für die Vorhersage neuer Feed-Einträge konzipiert wurden, müssen diese für feedspezifische Sonderfälle angepasst werden. Ein Sonderfall liegt z. B. vor, wenn aus den Zeitstempeln der Einträge eines Fensters kein Intervall berechnet werden kann. Darüber hinaus werden alle Algorithmen um die in Abschnitt 3.5.4 eingeführten Schranken α und β erweitert. Um den Fokus der nachfolgenden Betrachtungen auf den jeweiligen Kern des Algorithmus zu lenken, werden die eigenen Schranken vorab erläutert und in den jeweiligen Betrachtungen der Algorithmen nur erwähnt.

4.2.1. Einheitliche Modifikationen der Algorithmen

Es sei Φ die Menge aller betrachteten Vorhersagealgorithmen. Für jedes $\phi \in \Phi$ wird nach jeder Abfrage zum Zeitpunkt τ_z eine Zeitspanne $u_{\phi,z}$ ermittelt und anschließend auf Basis von $u_{\phi,z}$ der Zeitpunkt der nächsten Abfrage τ_{z+1} errechnet.

Um zu vermeiden, dass einige Feeds in zu kleinen bzw. großen Abständen abgefragt werden, wurden in Abschnitt 3.5.4 die Schranken α und β eingeführt. Verfügt ein Feed über zu wenige Einträge oder tragen alle Einträge des Feeds den gleichen Zeitstempel als Veröffentlichungsdatum, so können Algorithmen wie FixLearned kein geeignetes Intervall ermitteln. In diesem Fall wird stattdessen ein Standardwert η mit $\alpha \leq \eta \leq \beta$ gewählt. In der Evaluation in Abschnitt 4.4 wird $\eta = 60$ Minuten gesetzt: dies ist der Standardwert einiger Feed-Reader-Programme. Korrigierte Intervalle $\widetilde{u}_{\phi,z}$ werden mit einer Tilde symbolisiert und nach Gleichung 4.1 gebildet.

$$\widetilde{u}_{\phi,z} = \begin{cases} \eta & \text{wenn } u_{\phi,z} \text{ nicht definiert ist,} \\ \min\left(\beta, \max\left(\alpha, (u_{\phi,z})\right)\right) & \text{sonst} \end{cases} \tag{4.1}$$

Theoretisch könnte die Behandlung des Sonderfalls, dass $u_{\phi,z}$ nicht bestimmbar ist, auf $u_{\phi,z} = 0$ abgebildet werden, sodass $u_{\phi,z}$ auf den Wert der unteren Schranke α gesetzt wird. Der Vorteil der expliziten Behandlung nicht bestimmbarer Intervalle $u_{\phi,z}$ ist, dass bei der Wahl einer sehr kleinen unteren Schranke wie $\alpha = 1\,\text{Minute}$ zwischen sich tatsächlich sehr häufig ändernden Feeds und solchen unterschieden werden kann, für die kein Intervall ermittelt werden kann.

4.2.2. Algorithmen mit statischem Intervall

Die am einfachsten aufgebauten Algorithmen besitzen ein statisches Intervall, dass initial festgelegt und anschließend nicht mehr verändert wird, wobei zwischen vom Nutzer vorgegebenen und gelernten Intervallen zu unterscheiden ist.

Fix1h, Fix1d und Fix7d

Die Wahl eines statischen Abfrageintervalls ist ein trivialer Algorithmus, indem jeder Feed mit dem gleichen, statischen Intervall abgefragt wird. Ein Abfrageintervall von einer Stunde, kurz Fix1h, ist ein typischer Wert von Feed-Reader-Programmen (siehe Abschnitt 2.7) und wird daher als *Baseline* der Evaluation verwendet. Als weitere universelle Algorithmen werden Fix1d mit einem statischen Intervall von einem Tag und der in [CG03b] erwähnte Fix7d mit einem Intervall von einer Woche betrachtet.

FixLearned

In den Abschnitten 2.4.1 und 2.4.3 wurde vorgeschlagen, Quellen mit einem statischen Intervall abzufragen, wobei die Periodendauer *einmalig* an die jeweilige Quelle angepasst wird. Für Feeds lässt sich dies auf zwei verschiedene Varianten umsetzen: Einerseits können ausschließlich die im Fenster der ersten Abfrage $\mathcal{A}$ erhaltenen Einträge zur Berechnung des Intervalls verwendet werden, andererseits kann zusätzlich der Zeitpunkt $\tau_{\mathcal{A}}$ mit in die Berechnung einbezogen werden. Beide Varianten sind in Abbildung 4.2 dargestellt und werden im Folgenden vorgestellt.

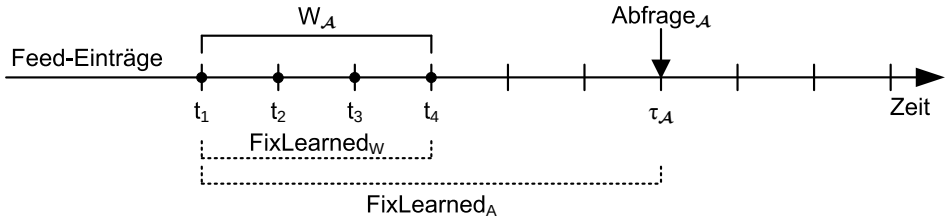

Abbildung 4.2.: Ermittlung des Abfrageintervalls u für die verschiedenen Varianten des Algorithmus FixLearned.

Bei dem als FixLearned_W bezeichneten Algorithmus wird das Intervall ausschließlich anhand des ersten Fensters mit der Fenstergröße $|W_{\mathcal{A}}|$ ermittelt. Es entspricht dem

Durchschnitt aller Intervalle zweier aufeinanderfolgender Einträge und ist in Gleichung 4.2 am Beispiel mit $|W_{\mathcal{A}}| = 4$ dargestellt.

$$u_{FixLearned_W,\mathcal{A}} = \frac{1}{3} * \sum_{n=1}^{3}(t_{n+1} - t_n) = \frac{t_4 - t_1}{3} \tag{4.2}$$

Verallgemeinert für beliebige Abfragen z ergibt sich Gleichung 4.3, wobei $t_{W_{\mathcal{A}},start}$ und $t_{W_{\mathcal{A}},end}$ dem Veröffentlichungsdatum des *ältesten* (t_1) respektive *neuesten* (t_4) Eintrags entsprechen, der im ersten Fenster $W_{\mathcal{A}}$ enthalten ist.

$$u_{FixLearned_W,z} = \frac{t_{W_{\mathcal{A}},end} - t_{W_{\mathcal{A}},start}}{|W_{\mathcal{A}}| - 1}; \quad |W_{\mathcal{A}}| \geq 2 \tag{4.3}$$

Eine Alternative ist, neben dem ersten Fenster den Zeitpukt $\tau_{\mathcal{A}}$ der Abfrage $\mathcal{A}$ mit in die Rechnung einzubeziehen. Diese Variante sei als $\mathsf{FixLearned}_A$ bezeichnet, für die Berechnung des Intervalls wird Gleichung 4.3 zu 4.4 erweitert. Das Intervall entspricht somit dem Quotient aus der Länge des Beobachtungszeitraums und der Anzahl der Einträge, die in diesem Zeitraum veröffentlicht wurden.

$$\begin{aligned} u_{FixLearned_A,z} &= \frac{(t_{W_{\mathcal{A}},end} - t_{W_{\mathcal{A}},start}) + (\tau_z - t_{W_{\mathcal{A}},end})}{|W_{\mathcal{A}}| - 1 + 1}; \quad |W_{\mathcal{A}}| \geq 1 \\ &= \frac{\tau_{\mathcal{A}} - t_{W_{\mathcal{A}},start}}{|W_{\mathcal{A}}|} \end{aligned} \tag{4.4}$$

Der Datensatz enthält einen hohen Anteil von $\mathsf{Zombies}$ mit $t_{W_{\mathcal{A}},end} \ll \tau_{\mathcal{A}}$, sodass zu erwarten ist, dass $\mathsf{FixLearned}_A$ im Durchschnitt weniger Abfragen stellt, als $\mathsf{FixLearned}_W$. Liegen $t_{W_{\mathcal{A}},end}$ und $\tau_{\mathcal{A}}$ jedoch dicht beieinander mit $(\tau_{\mathcal{A}} - t_{W_{\mathcal{A}},end}) < u_{FixLearned_W}$, so gilt $u_{FixLearned_A} < u_{FixLearned_W}$ und $\mathsf{FixLearned}_A$ fragt den entsprechenden Feed häufiger ab als $\mathsf{FixLearned}_W$.

Sowohl bei $\mathsf{FixLearned}_W$ als auch $\mathsf{FixLearned}_A$ wird das anhand der ersten Abfrage gelernte Modell nie aktualisiert. Für beide Alternativen ergibt sich der Zeitpunkt der nächsten Abfrage $\tau_{\phi,z+1}$ rekursiv aus dem aktuellen Zeitpunkt $\tau_{\phi,z}$ und dem korrigierten Intervall $\widetilde{u}_{\phi,z}$.

$$\tau_{\phi,z+1} = \tau_{\phi,z} + \widetilde{u}_{\phi,z} \tag{4.5}$$

Der Zusammenhang sei an einem Beispiel des Algorithmus $\mathsf{FixLearned}_A$ veranschaulicht. Aus den Gleichungen 4.1, 4.4 und 4.5 ergibt sich mit $\phi = FixLearned_A$ der Zeitpunkt der nächsten Abfrage $\tau_{FixLearned_A,z+1}$ wie folgt:

$$\tau_{FixLearned_A,z+1} = \tau_{FixLearned_A,z} + \begin{cases} \eta & \text{wenn } |W_{\mathcal{A}}| = 0 \text{ oder } t_{W_{\mathcal{A}},start} = \tau_{\mathcal{A}} \\ \min\left(\beta, \max\left(\alpha, \left(\frac{\tau_{\mathcal{A}} - t_{W_{\mathcal{A}},start}}{|W_{\mathcal{A}}|}\right)\right)\right) & \text{sonst} \end{cases} \tag{4.6}$$

Eine dritte Alternative zur Berechnung wäre eine Kombination der vorgenannten Ansätze. Das erste empfangene Fenster wird als Beobachtungszeitraum aufgefasst, das Intervall nach dem Grundsatz Beobachtungszeitraum durch Anzahl Einträge berechnet und folglich

in Gleichung 4.3 der Divisor $W_{\mathcal{A}} - 1$ durch $W_{\mathcal{A}}$ ersetzt. Wie eine einfaches Beispiel mit vier Einträgen und einer tatsächlichen Änderungsrate von einer Stunde zeigt, führt dies zu einem falschen Ergebnis: der Algorithmus würde den Feed alle 45 Minuten abfragen. Der Fehler des Ansatzes ergibt sich aus der Ermittlung des Beobachtungszeitraumes, da auf beide Grenzen jeweils ein Eintrag fällt.

4.2.3. Adaptive Algorithmen mit begrenzter Historie

Die im Folgenden aufgegriffenen Algorithmen AdaptiveTTL und LRU-2 sowie der aus den Vorteilen mehrerer Algorithmen entwickelte MAVSync werten einen begrenzten Abschnitt der Historie der Feeds aus, um das Abfrageintervall adaptiv an den jeweiligen Feed anzupassen.

AdaptiveTTL

Der aus der Domäne Web Caching bekannte Algorithmus AdaptiveTTL wurde in Abschnitt 2.5.3 vorgestellt. Die in Gleichung 2.10 verwendeten Parameter date und last-modified werden in der Simulation durch die bereits bekannten Parameter τ und t ersetzt. Die last-modified Zeitstempel der Server können nicht aus dem Datensatz simuliert werden, da einige Server diese Zeitstempel zur Anfragezeit aktualisieren, andere generieren das Feed-Dokument in regelmäßigen Abständen neu, sodass unter Umständen zwar last-modified, nicht aber der Feed selbst geändert wird. Mit den Modifikationen ergibt sich Gleichung 4.7, die Bildung von $\widetilde{u}_{AdaptiveTTL,z}$ erfolgt wie bereits bekannt nach Gleichung 4.1. Auf die Darstellung der oberen Schranke β aus Gleichung 2.10 wurde verzichtet – die Schranke ist bereits in 4.1 enthalten.

$$u_{AdaptiveTTL,z} = M * (\tau_z - t_{W_z,end}); \quad |W_z| \geq 1 \tag{4.7}$$

In der Domäne Web Caching wird für M häufig einer der Werte 0,1 oder 0,2 eingesetzt. Um ein geeignetes M für den betrachteten Feed-Datensatz zu ermitteln, wird M in der Evaluation in Abschnitt 4.4.1 zunächst zwischen 0,1 und 1,0 variiert und die Evaluationsergebnisse des am besten geeigneten Wertes anschließend mit denen der anderen Algorithmen verglichen. Auf diese Weise wird neben dem Vergleich aller Algorithmen untersucht, ob sich die geläufigen Werte 0,1 und 0,2 für die Vorhersage von Änderungen von Feeds eignen. Die Berechnung des Zeitpunktes der nächsten Abfrage erfolgt nach Gleichung 4.5.

LRU-2

LRU-2 und die verallgemeinerte Form LRU-K [OOW93] wurden ebenfalls bereits in Abschnitt 2.5.3 vorgestellt. Überträgt man LRU-2 auf die Domäne Feeds, so wird das Intervall zwischen den Zeitpunkten $t_{F,new-1}$ und $t_{F,new}$ der beiden neuesten Einträge des Feeds betrachtet, es ergibt sich Gleichung 4.8. Im Unterschied zu den vorangegangen Algorithmen wird die Betrachtung nicht auf das aktuelle Fenster beschränkt, sondern der komplette Strom der Einträge betrachtet. Dieser Unterschied ist für alle Feeds relevant, die dem Muster Single Entry zugeordnet werden können.

Eine weitere, erwähnenswerte Besonderheit ist die Definition, welcher Eintrag als Zweit-Neuester erachtet wird. Angenommen ein Feed hat eine Fenstergröße von drei und alle Einträge wurden am selben Tag veröffentlicht: a um 0:00 Uhr und b, c um 1:00 Uhr. Streng genommen sind b und c die Neuesten. Aufgrund des Ursprungs LRU-2 aus der Domäne Caching, in der nur eine Änderung eines Objekts pro Zeitpunkt betrachtet wird, erfolgt die Definition, dass $t_{F,new-1} < t_{F,new}$ gilt und kein $t_{F,x}$ mit $t_{F,new-1} < t_{F,x} < t_{F,new}$ existiert. Es ergibt sich a als Zweit-Neuester. Wurde insgesamt nur ein oder gar kein Eintrag empfangen, so wird das Intervall in Gleichung 4.1 zu η korrigiert. Der Zeitpunkt der nächsten Abfrage ergibt sich anschließend nach Gleichung 4.5:

$$u_{LRU2,z} = t_{F,new} - t_{F,new-1} \tag{4.8}$$

Erweiterter Moving Average mit virtuellem Eintrag

Die mit Fix* zusammengefassten Algorithmen mit statischem Intervall haben den entscheidenden Nachteil, dass sie nicht adaptiv auf Variationen der Änderungsrate der Feeds reagieren können. AdaptiveTTL verwendet ausschließlich den Zeitraum zwischen dem neuesten Eintrag und der aktuellen Abfrage, sodass die übrigen Einträge des Fensters und somit wertvolle Informationen nicht beachtet werden. Wird ein Feed z. B. regelmäßig am Ersten des Monats aktualisiert und erstmalig am dritten Tag des Monats abgefragt, so erfolgt die nächste Abfrage bereits nach wenigen Tagen. LRU-2 wäre in diesem Fall besser geeignet. Weist die Änderungsrate hingegen eine hohe Varianz auf, so ist die Betrachtung des letzten bekannten Intervalls ungeeignet.

Aus den in der Analyse des Datensatzes gewonnenen Erkenntnissen sowie den Stärken und Schwächen der betrachteten Verfahren wird der eigene Algorithmus MAVSync entwickelt. Das Ziel ist, durch geschickte Kombination der beschriebenen Verfahren die Schwächen eines Algorithmus mit den Stärken eines anderen auszugleichen.

Die Grundidee des MAVSync ist, einen virtuellen Eintrag einzusetzen, um $u_{MAVSync}$ in Zeiten geringer Aktivität des Feeds kontinuierlich zu vergrößern. Dies ist ein Unterschied zu dem als Gleitender Mittelwert (engl. Moving Average) bekannten Algorithmus, bei dem nur tatsächliche Werte einer Datenreihe berücksichtigt werden. Bei einer erkannten Änderung wird $u_{MAVSync}$ wieder an die aktuelle Änderungsrate des Feeds angepasst. In diesem Schritt wird zudem eine Synchronisation zwischen dem vorhergesagten Zeitpunkt des nächsten Eintrags und dem Zeitpunkt der nächsten Abfrage angestrebt.

Die Idee der Synchronisation zwischen dem abfragenden Feed Data Provider und dem Feed Server sei an einem Beispiel in Abbildung 4.3 veranschaulicht. Der Feed hat eine Fenstergröße von vier Einträgen, neue Einträge erscheinen gleichmäßig zu jeder vollen Stunde. Die Berechnung der Änderungsrate erfolgt nach Gleichung 4.9 auf Grundlage von Gleichung 4.3 (FixLearned_W), jedoch mit der Modifikation, dass das Intervall $u_{MAVSync_W,z}$ jeweils anhand des aktuellen Fensters W_z ermittelt wird. Mögliche Variationen der Änderungsrate werden aufgrund der zyklischen Neuberechnung des Intervalls berücksichtigt.

$$u_{MAVSync_W,z} = \frac{t_{W_z,end} - t_{W_z,start}}{|W_z| - 1}; \quad |W_z| \geq 2 \tag{4.9}$$

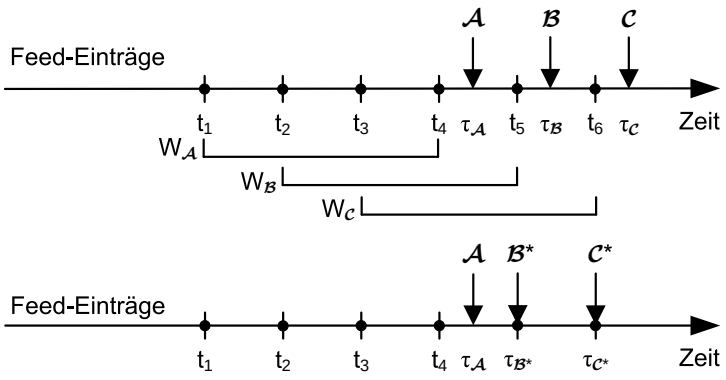

Abbildung 4.3.: Vorhersage der Zeitpunkte neuer Einträge. Oben: ohne Synchronisation, unten: mit Synchronisation.

Im oberen Bereich der Abbildung ergibt sich $\tau_{\mathcal{B}}$ anhand der Gleichungen 4.9 und 4.5 (mit $z = \mathcal{A}$ und $z{+}1 = \mathcal{B}$), sodass der Feed nach einer Stunde erneut abgefragt wird. Diese Folge setzt sich fort, wobei der Abstand zwischen dem jeweils neuesten Eintrag und der aktuellen Abfrage konstant ist und generell ein Delay von z. B. 20 Minuten vorliegt. Als naheliegende Lösung bietet es sich an, den Zeitpunkt der nächsten Abfrage basierend auf dem jeweils neuesten Eintrag der aktuellen Abfrage zu berechnen. Nach Gleichung 4.10 ergeben sich die im unteren Bereich der Abbildung dargestellten Abfragen $\tau_{\mathcal{B}}* = t_4 + u_{\mathcal{A}}$ und $\tau_{\mathcal{C}}* = t_5 + u_{\mathcal{B}}*$, die jeweils mit den Zeitpunkten t_5 und t_6 der nächsten Einträge übereinstimmen, woraus ein Delay von Null resultiert. Der Schritt in Gleichung 4.10 wird als *Synchronisationsschritt* bezeichnet.

$$\tau_{z+1} = t_{W_z,end} + u_{MAVSync_W,z} \tag{4.10}$$

Die Berechnung erscheint zunächst optimal, hat jedoch zwei Nachteile, die in Abbildung 4.4 dargestellt sind: 1) Folgen nach dem Zeitpunkt t_4 zunächst keine weiteren Einträge, so kann $u_{MAVSync_W,z}$ bei der alleinigen Betrachtung des Fensters $W_{\mathcal{A}1}$ nicht adaptiv vergrößert werden. Dies hat zur Folge, dass der Feed in Zeitabschnitten ohne Aktivität (beispielsweise 3:00–10:00 Uhr) unnötig häufig abgefragt wird, z. B. stündlich. 2) Liegen t_1–t_4 dicht beieinander, jedoch insgesamt weit in der Vergangenheit von $\mathcal{A}$, so ergibt sich nach Gleichung 4.10 ein $\tau_{\mathcal{B}}$ mit $\tau_{\mathcal{B}} < \tau_{\mathcal{A}}$: der vorhergesagte Zeitpunkt liegt in der Vergangenheit. Dieser Effekt ist vor allem bei Zombies zu erwarten.

In beiden Fällen erfolgt daher eine alternative Berechnung, die auf $\mathsf{FixLearned}_A$ basiert (Gleichung 4.4). Zum Zeitpunkt τ_z wird ein virtueller Eintrag eingefügt und in die Berechnung des Intervalls einbezogen, sodass sich Gleichung 4.11 ergibt. Der Zeitpunkt der nächsten Abfrage τ_{z+1} ergibt sich nach Gleichung 4.5 aus dem ermittelten Intervall $u_{MAVSync_A,z}$ sowie dem dem aktuellen Zeitpunkt τ_z:

$$u_{MAVSync_A,z} = \frac{\tau_z - t_{W_z,start}}{|W_z|}; \quad |W_z| \geq 1 \tag{4.11}$$

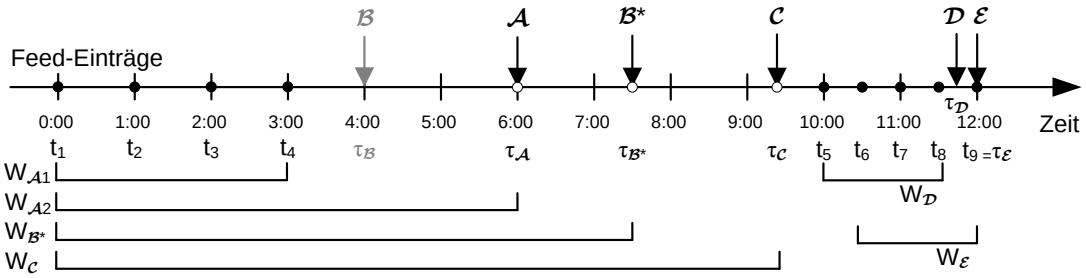

Abbildung 4.4.: MAVSync: Bestimmung der Zeitpunkte neuer Abfragen unter Einbeziehung virtueller Einträge.

Die beiden grundlegenden Eigenschaften des MAVSync lassen sich wie folgt zusammenfassen: Für die Berechnung des Zeitpunktes der nächsten Abfrage wird zuerst geprüft, ob der Synchronisationsschritt ausgeführt werden kann. Ist dies nicht der Fall, erfolgt die Berechnung auf Basis eines virtuellen Eintrags. Solange in aufeinander folgenden Abfragen kein neuer Eintrag enthalten ist, wird der virtuelle Eintrag auf den Zeitpunkt der jeweiligen Abfrage verschoben und das Abfrageintervall iterativ vergrößert. Sobald ein neuer Eintrag identifiziert ist, erfolgt eine Anpassung des Intervalls an dieses Fenster und es wird geprüft, ob ein Synchronisationsschritt ausgeführt werden kann.

Im aktuellen Beispiel in Abbildung 4.4 sind die ersten vier Einträge jeweils zur vollen Stunde veröffentlicht, $t_1 = 0{:}00\,\text{Uhr}$. Die erste Abfrage $\mathcal{A}$ erfolgt um $\tau_{\mathcal{A}} = 6{:}00\,\text{Uhr}$. Nach den Gleichungen 4.9 und 4.10 ergibt sich ein Intervall $u_{MAVSync_W,\mathcal{A}1} = 1\,\text{Stunde}$ und somit $\tau_{\mathcal{B}} = 4{:}00\,\text{Uhr}$. Da $\tau_{\mathcal{B}} < \tau_{\mathcal{A}}$ kann kein Synchronisationsschritt ausgeführt werden. Stattdessen wird zum Zeitpunkt $\tau_{\mathcal{A}} = 6{:}00\,\text{Uhr}$ ein virtueller Eintrag eingefügt und das Intervall $u_{MAVSync_W,\mathcal{A}2} = 90\,\text{Minuten}$ nach Gleichung 4.11 ermittelt. Es ergibt sich $\tau_{\mathcal{B}}* = 7{:}30\,\text{Uhr}$ als Zeitpunkt der nächsten Abfrage $\mathcal{B}*$. Es wird kein neuer Eintrag in $W_{\mathcal{B}}*$ gefunden, sodass der virtuelle Eintrag auf $\tau_{\mathcal{B}}* = 7{:}30\,\text{Uhr}$ verschoben und das Intervall auf $u_{MAVSync_W,\mathcal{B}*} = 113\,\text{Minuten}$ vergrößert wird. Die nächste Abfrage $\mathcal{C}$ erfolgt um $\tau_{\mathcal{C}} = 9{:}23\,\text{Uhr}$. Um $\tau_{\mathcal{D}} = 11{:}44\,\text{Uhr}$ werden vier neue Einträge vorgefunden, das Intervall $u_{MAVSync_W,\mathcal{D}} = 30\,\text{Minuten}$ bestimmt und ein Synchronisationsschritt durchgeführt, sodass sich $\tau_{\mathcal{D}} = 12{:}00\,\text{Uhr}$ ergibt. Der Eintrag E_9 wird in diesem Idealfall ohne eine Verzögerung erlangt.

Aus der Analyse des Datensatzes ergab sich, dass die Erstellung des XML-Dokuments großer Feeds einige Sekunden benötigen kann. Um bei einer geringfügigen Verzögerung in der Bereitstellung des Feeds und der resultierenden Verschiebung von t_9 zu vermeiden, dass $\tau_{\mathcal{E}}$ z. B. zwei Sekunden vor t_9 liegt, wird für den Einsatz des Algorithmus unter realen Bedingungen empfohlen, einen zusätzlichen Puffer κ von wenigen Sekunden in Gleichung 4.10 aufzunehmen: $\tau_{z+1} = t_{W_z,end} + u_{MAVSync_W,z} + \kappa$. In dieser Arbeit wird $\kappa = 0$ gesetzt, etwaige Verzögerungen der Server werden nicht simuliert.

Algorithmus 2 ist eine prägnante Zusammenfassung des MAVSync. Er besteht aus den Methoden $update(W_z, \tau_z)$ in den Zeilen 1–18 zur Berechnung des Zeitpunktes der nächsten Abfrage τ_{z+1} sowie $limitUpdateInterval(u_{check})$ zur Beschränkung des ermittelten

Intervalls durch α und β (Zeilen 20–27). Nach der Initialisierung von τ_{sync} in Zeile 2 erfolgt in Zeile 4 die Umsetzung der Gleichung 4.9, sodass das Intervall $u_{MAVSync_W,z}$ aus dem aktuellen Fenster berechnet wird. Enthält das Fenster W_z mindestens zwei Einträge, so ergibt sich der potentielle Zeitpunkt der nächsten Abfrage τ_{sync} aus dem Zeitpunkt des neuesten Eintrags $t_{W_z,end}$ und dem Intervall des letzten Fensters (Zeilen 5 und 7 entsprechen Gleichung 4.10). Liegt das resultierende Intervall u_{check} innerhalb der Schranken α und β, wird der Synchronisationsschritt durchgeführt und die Verarbeitung beendet (Zeilen 7–9). In diesem Schritt werden auch etwaige negative Intervalle erfasst, z. B. das sich im vorangegangenen Beispiel ergebende Intervall $u_{check} = \tau_{\mathcal{A}} - \tau_{\mathcal{B}}$.

Erfolgt keine Synchronisation, wird $u_{MAVSync_A,z}$ anhand des aktuellen Zeitpunktes τ_z und dem des ältesten Eintrags $t_{W_z,start}$ ermittelt (Zeilen 12–14, entspricht Gleichung 4.11), durch die Schranken begrenzt und die nächste Abfrage τ_{z+1} auf Basis des aktuellen Zeitpunktes τ_z berechnet (Zeile 15). Enthält W_z keinen Eintrag, so wird das Standardintervall η eingesetzt (Zeile 11).

Algorithmus 2 Adaptive Ermittlung des Abfrageintervalls mittels MAVSync.

```
1:  update(W_z, τ_z) {
2:  τ_sync ← 0
3:  if |W_z| ≥ 2 then
4:      u_MAVSync_W,z ← (t_Wz,end − t_Wz,start) / (|W_z| − 1)
5:      τ_sync ← t_Wz,end + u_MAVSync_W,z
6:  end if
7:  u_check ← τ_sync − τ_z
8:  if u_check == limitUpdateInterval(u_check) then
9:      τ_z+1 = τ_sync
10: else
11:     u_MAVSync_A,z ← η
12:     if |W_z| ≥ 1 then
13:         u_MAVSync_A,z ← (τ_z − t_Wz,start) / |W_z|
14:     end if
15:     τ_z+1 ← τ_z + limitUpdateInterval(u_MAVSync_A,z)
16: end if
17: return τ_z+1
18: }
19:
20: limitUpdateInterval(u_check) {
21: if u_check < α then
22:     return α
23: else if u_check > β then
24:     return β
25: end if
26: return u_check
27: }
```

4.2.4. Auf dem Poisson-Modell basierende Algorithmen

Die Algorithmen dieser Gruppe werden auch als PostRate-Algorithmen bezeichnet. Die Grundidee ist, dass die Änderungsrate von Feeds zyklischen Mustern unterliegt. Diese wird geeignet modelliert, in einer Trainingsphase gelernt und Vorhersagen anschließend anhand des Modells getroffen.

IndHist

In IndHist wird die Änderungsrate λ eines Feeds als nicht-homogener Poisson-Prozess modelliert. Ein Zyklus entspricht einem in 24 Stunden unterteilten Tag, die jeweilige Änderungsrate wird in einer Trainingsphase gelernt. IndHist wurde in Abschnitt 2.5.4 ausführlich betrachtet, die Funktionsweise sei an dieser Stelle noch einmal kurz skizziert: Basis der Ermittlung des Zeitpunktes der nächsten Abfrage τ_{z+1} ist τ_z. Von diesem Zeitpunkt werden iterativ die Änderungsraten der nachfolgenden Stunden (anteilig) kumuliert, bis ein Schwellwert θ erreicht ist, sodass z. B. nach 4h:12m $\theta = 0,7$ Änderungen erwartet werden und der Feed erneut abgefragt wird. Der Schwellwert θ ist global für alle Feeds zu ermitteln.

IndHist/TTL

IndHist/TTL ist eine Kombination der Algorithmen IndHist und AdaptiveTTL, um bursts verarbeiten zu können, die nicht im Modell von IndHist gelernt wurden. Beide Algorithmen sind dem Leser bereits vertraut. Wie in Abschnitt 2.5.4 (auf Seite 57) bereits erläutert, wird zu jedem Zeitpunkt τ_z zunächst geprüft, ob die Anzahl der in einem Zeitfenster V[1] beobachteten Änderungen mit dem Modell übereinstimmt. Überschreitet das Verhältnis beobachteter zu modellierten Änderungen einen Schwellwert T_{burst}, so erfolgt die Berechnung von τ_{z+1} mittels AdaptiveTTL, andernfalls mittels IndHist. Die Autoren ermittelten $T_{burst} = 2$ als geeignet und führten Experimente mit den Fenstergrößen $V = 1$ und $V = 24$ Stunden durch. Hierbei variierten sie die Parameter M für AdaptiveTTL sowie θ für IndHist.

Für den Vergleich aller Algorithmen in Abschnitt 4.4 werden die beiden Parameter M und θ zunächst separat anhand der jeweiligen Basisalgorithmen bestimmt. Um Synergieeffekte in der Evaluation zu nutzen werden die einzeln bestimmten Bestwerte anschließend für IndHist/TTL verwendet.

LIHZ

Der in [LIHZ08] beschriebene und im Folgenden als LIHZ bezeichnete Algorithmus wurde in Abschnitt 2.7.3 vorgestellt. Die Änderungsrate eines Feeds wird als inhomogener Poisson-Prozess modelliert, wobei ein Zyklus von einer Woche mit taggenauer Granularität in einer Trainingsphase gelernt wird. Im Unterschied zu IndHist, bei dem die Anzahl Änderungen pro Zeiteinheit erfasst wurde, erfolgt in LIHZ ausschließlich eine binäre

[1] Der von den Autoren in [BGR06] gewählte Parameter W wird im Folgenden als V bezeichnet, um eindeutig zwischen dem Zeitfenster V und dem Fenster W eines Feeds zu differenzieren.

Erfassung, ob an einem Tag eine Änderung festgestellt wurde. Die Anzahl erfolgter Änderungen wird nicht beachtet.

LIHZ wird wie in Abschnitt 2.7.3 beschrieben implementiert, die in [LIHZ08] identifizierten, jedoch von den Autoren nicht beantworteten Fragen werden wie folgt umgesetzt: Erkannte Änderungen werden dem Tag zugeschrieben, an dem sie bei einer Abfrage identifiziert wurden, eine rückwirkende Aktualisierung des Modells anhand der im Fenster enthaltenen Einträge findet nicht statt. Der in Gleichung 2.11 (Seite 68) aufgeführte Parameter α wird wie von den Autoren vorgeschlagen mit dem fixen Wert $\alpha = 0,9$ übernommen. Zur Entscheidung, wann die nächste Abfrage erfolgt, summieren die Autoren die sich aus 2.11 ergebenden Wahrscheinlichkeiten, dass an einem Tag t eine Änderung auftritt. Beim Erreichen eines in [LIHZ08] verbal eingeführten Schwellwertes von 0,5 erfolgt die Abfrage des Feeds. Dieser Schwellwert wird in der eigenen Arbeit als σ bezeichnet und in der Evaluation variiert.

4.3. Test Aufbau

Zur Durchführung der Evaluation wurde die nachfolgend beschriebene Simulationsumgebung aufgebaut, in der jeder Algorithmus separat ausgeführt und anhand der im Anschluss in Abschnitt 4.3.3 eruierten Metriken bewertet wird.

4.3.1. Aufbau der Simulationsumgebung

Der Aufbau der Simulationsumgebung basiert auf der zur Erstellung des Datensatzes eingesetzten Implementierung und ist in Abbildung 4.5 skizziert. Die für jeden Feed zyklisch durchzuführenden Schritte wurden bereits in den Abschnitten 3.3 und 3.5 sowie in Abbildung 3.1 eingeführt. Zur Simulation der Algorithmen in den Kapiteln 4 und 5 wurden die folgenden Modifikationen durchgeführt: Anstatt Feeds von den Servern abzufragen, wurde über die Simulationsumgebung auf den erstellten Datensatz zugegriffen und die zum simulierten Zeitpunkt aktuellen Einträge zurückgegeben. Als Simulationsdauer wurden 27 Tage festgelegt, deren Beginn am 09.07.2011 07:00:00 CEST[2] lag. Die Notwendigkeit der leichten Verkürzung des Simulationszeitraums im Vergleich zur Erstellung des Datensatzes wird im Anschluss erläutert. Die Simulationsdauer wurde zudem in eine 7-tägige Trainingsphase und eine 20-tägige Testphase unterteilt, um den Anforderungen einiger Algorithmen zu genügen, die zwischen diesen beiden Phasen unterscheiden. Der verwendete Feed Datensatz bestand aus den beiden in Abschnitt 3.5.7 beschriebenen Datenbanktabellen feeds und feed_polls sowie einer dritten Tabelle mit allen Feed-Einträgen.

Die Schritte zur Verarbeitung des Feeds erfolgten wie bereits bekannt: Anhand des simulierten Fensters werden zunächst die neuen Einträge identifiziert. Anschließend folgen die Ermittlung der Änderungsrate des Feeds und die Planung der nächsten Überprüfung. Diese beiden Schritte erfolgen in Abhängigkeit vom eingesetzten Algorithmus zur Vorhersage. Mit dem letzten Schritt schließt sich der Verarbeitungszyklus einer Abfrage. Liegt der

[2] Central European Summer Time (CEST)

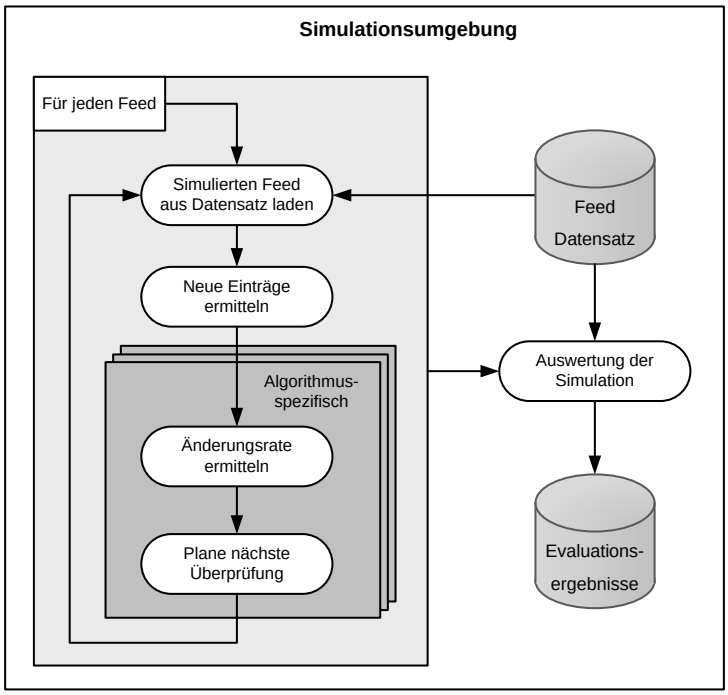

Abbildung 4.5.: Vereinfachte Darstellung der Simulationsumgebung.

simulierte Zeitpunkt der nächsten Abfrage des Feeds innerhalb des simulierten Zeitraums, so erfolgt eine weitere Iteration, andernfalls ist die Verarbeitung für den Feed beendet.

Zur Bewertung der Algorithmen verfügt die Simulationsumgebung über eine Monitoring-Komponente. Diese hat sowohl wahlfreien Zugriff auf den Datensatz als auch auf die Schritte des Verarbeitungszyklus der Feeds und persistiert die Ergebnisse der Evaluationen. Die pro Zyklus aufgezeichneten Evaluationsergebnisse sowie die Kriterien zur Auswertung und dem Vergleich der Algorithmen werden in Abschnitt 4.3.3 vorgestellt. Zunächst wird in Abschnitt 4.3.2 der Zugriff auf den simulierten Feed untersucht.

Für die technische Umsetzung der rund zwei Milliarden simulierten Feed-Abfragen standen temporär zwei Server zur Verfügung, von denen einer nativ und der andere mittels einer virtuellen Maschine genutzt wurde. Die Hardware des nativ genutzten Gerätes basiert auf 4 Dual Core AMD Opteron MP 880 (2,4 GHz, 2 MB Cache), 32 GB DDR RAM und einem internen Speichersystem aus 5 Ultra-320 SCSI Festplatten (RAID 5). Der mittels vmWare virtualisierte Server basiert auf 2 Quad-Core Intel Xeon E5620 (2,4 GHz, 12 MB Cache), 48 GB DDR3 RAM sowie einem per 8 Gbit/s Fibre Channel angebundenen Speichersystem aus 6 Nearline-SAS Festplatten (RAID 5), der VM standen hiervon 4 CPU Kerne sowie 30 GB RAM zur Verfügung. Auf beiden Systemen erfolgten identische Installationen von Ubuntu 10.10 Server 64bit mit MySQL Server 5.1.49. Zur Optimierung der pro simulierter Feed-Abfrage notwendigen drei Datenbankzugriffe wurden die ca. 6 GB großen Tabellen mit rund 32 bzw. 40 Millionen Einträgen geeignet partitioniert und indiziert.

4.3.2. Zusammenhänge zwischen der Erstellung des Datensatzes und den simulierten Zugriffen

Die Zusammenhänge zwischen der Erstellung des Datensatzes, der Bestimmung des Simulationszeitraums und der Bereitstellung simulierter Fenster sind in Abbildung 4.6 dargestellt und werden im Folgenden eruiert.

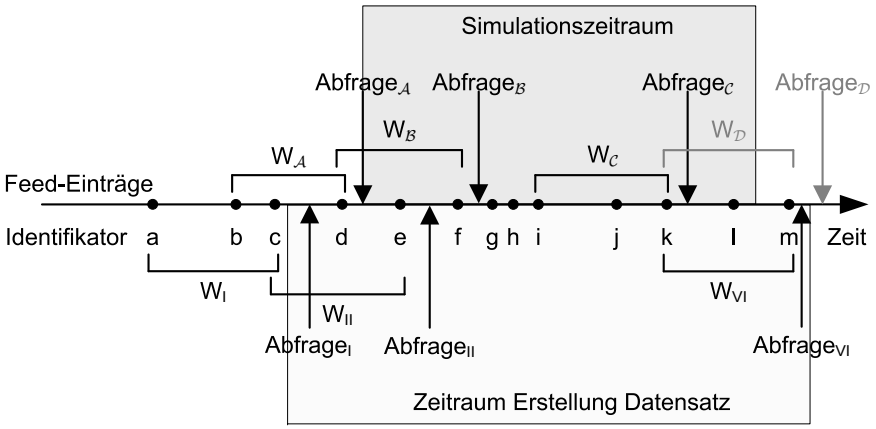

Abbildung 4.6.: Bereitstellung der simulierten Fenster.

Auf einer Zeitachse sind 13 Einträge a–m eines fiktiven Feeds verzeichnet. Diese sind anhand ihres korrigierten Veröffentlichungsdatums t (siehe Abschnitt 3.5.5) sortiert, sodass für zwei Einträge a, b stets $t_a \leq t_b$ gilt. Unterhalb der Zeitachse ist der Zeitraum der Erstellung des Datensatzes markiert sowie drei der sechs zur Erstellung des Datensatzes ausgeführte Abfragen I, II und VI mit den entsprechenden Fenstern W_I, W_{II} und W_{VI}. Auf die Darstellung der Abfragen III–V mit W_{III}–W_V wurde aus Gründen der Übersichtlichkeit, jedoch ohne Beschränkung der Allgemeinheit, verzichtet. Im oberen Bereich der Grafik sind der Simulationszeitraum sowie vier simulierte Abfragen $\mathcal{A}$–$\mathcal{D}$ und die entsprechenden Fenster $W_\mathcal{A}$–$W_\mathcal{D}$ dargestellt. Die Einträge der Fenster W_I und $W_\mathcal{A}$ liegen zwar nicht in den entsprechenden Zeiträumen, sie konnten jedoch bei den Abfragen $\mathcal{A}$ und I innerhalb der Zeiträume ermittelt werden. Je nach Fenstergröße und Änderungsrate des Feeds können die Einträge a–c mehrere Jahre in der Vergangenheit liegen.

Ermittlung des Simulationszeitraums

Zum Beginn der Erstellung des Datensatzes mussten alle 200.000 Feeds abgefragt werden. Dies geschah innerhalb eines Zeitraums von einer reichlichen Stunde, sodass der Zeitpunkt der ersten realen Abfrage I zwischen den Feeds variiert. Um sicherzustellen, dass zum Start der Simulation zu jedem Feed bereits Daten vorliegen, wurde der Zeitpunkt um einige Stunden hinter den Startzeitpunkt der Erstellung des Datensatzes geschoben.

Für die Wahl des Simulationsendes war zu beachten, dass bis zu diesem Zeitpunkt zu jedem Feed vollständige Informationen vorliegen müssen. Wie in Abschnitt 3.5.4 dargelegt, wurde jeder Feed mindestens alle sechs Stunden abgefragt. Das Ende des Simulations-

zeitraums wurde daher um sechs Stunden vor das Ende des Zeitraums der Erstellung des Datensatzes gelegt. Somit wurde sichergestellt, dass jeder Feed nach dem Ende des Simulationszeitraums noch mindestens einmal abgefragt wurde (Abfrage VI im Beispiel).

Unterteilung in Trainings- und Testphase

Der Simulationszeitraum wurde anschließend in eine Trainings- und Testphase geteilt, um den Anforderungen einiger Algorithmen wie IndHist zu genügen, die eine explizite Trainingsphase vorsehen. Die Trainingsphase beträgt 7 Tage vom 09.07.2011 07:00 CEST bis zum 16.07.2011 06:59:59 CEST, die resultierende Testphase umfasst 20 Tage vom 16.07.2011 07:00:00 CEST bis zum 05.08.2011 07:00:00 CEST.

Beziehen sich die in den nachfolgenden Abschnitten getroffenen Aussagen gleichermaßen auf beide Phasen, so wird der beide Phasen zusammenfassende Begriff Simulationszeitraum verwendet, andernfalls wird konkret zwischen den Phasen differenziert.

Bereitstellung der simulierten Feeds

Für jeden zu evaluierenden Algorithmus und jeden Feed erfolgt die erste Abfrage $\mathcal{A}$ exakt zum Beginn des Simulationszeitraums, die Zeitpunkte aller weiteren Abfragen $\mathcal{B}$–$\mathcal{D}$ werden vom jeweiligen Algorithmus berechnet.

Im Beispiel in Abbildung 4.6 hat der Feed eine konstante Fenstergröße von drei. Anhand des simulierten Zeitpunktes $\tau_{\mathcal{A}}$ werden von der Simulationsumgebung die drei nächstgelegenen, vergangenen Einträge b–d als simuliertes Fenster zurückgegeben. Der Eintrag a ist bereits nicht mehr in $W_{\mathcal{A}}$ enthalten und wird ignoriert.

Einen Sonderfall stellen Feeds dar, deren Fenstergröße variabel ist, da die Größen der simulierten Fenster aus den Werten der Erstellung des Datensatzes hergeleitet werden müssen. In Abschnitt 3.7.3 wurden drei Muster identifiziert: stetig wachsende Fenstergrößen, alternierende Fenster, die zwischenzeitlich den Wert Null annehmen sowie variierende Fenster, deren Größe schwankt. Die Simulation dieser Fenster ist nicht trivial und kann dazu führen, dass ein Algorithmus vermeintlich mehr neue Einträge auffindet als im Datensatz vorhanden sind. Dies sei an einem Beispiel in Abbildung 4.7 erläutert.

Der Feed hatte zu den Zeitpunkten τ_X und τ_{XI} eine Fenstergröße von vier und zu τ_{XII} von sechs Einträgen. In der Simulation wurde der Eintrag a bereits in einer vorangegangenen Abfrage erfasst. Für die Ermittlung der Größe von $W_{\mathcal{F}}$ zum Zeitpunkt $\tau_{\mathcal{F}}$ spielt es keine Rolle, ob die Fenstergröße von der zeitlich unmittelbar zurückliegenden oder zukünftigen realen Abfrage X oder XI übernommen wird, denn beide hatten eine Größe von Vier.

Zur Ermittlung von $W_{\mathcal{G}}$ kann entweder W_{XI} oder W_{XII} verwendet werden. Wird die Fenstergröße der vorangegangenen realen Abfrage übernommen, so wird zunächst das Fenster $W_{\mathcal{G}1}$ simuliert, dem Algorithmus die Einträge c–f übergeben und f als einziger neuer Eintrag identifiziert. In der darauffolgenden Abfrage $\mathcal{H}$ werden die Größe von $W_{\mathcal{H}}$ von W_{XII} übernommen und die Einträge b–g als simuliertes Fenster zurückgegeben. Die Schnittmenge von $W_{\mathcal{G}1}$ und $W_{\mathcal{H}}$ sind c–f, sodass b und g als neu identifiziert werden.

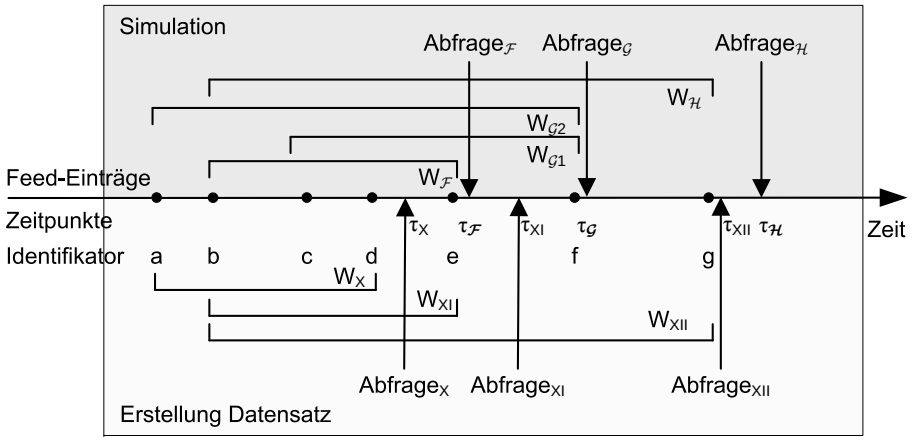

Abbildung 4.7.: Simulation variabler Fenstergrößen.

Der Eintrag b wird daher zwei Mal als neuer Eintrag identifiziert, sodass insgesamt acht statt sieben neue Einträge beobachtet werden.

Die alternative Wahl für die Fenstergröße von $\mathcal{G}$ ist als $W_{\mathcal{G}2}$ dargestellt, in diesem Fall wird a ein zweites Mal als neu identifiziert. Wie das Beispiel zeigt, kann es in beiden Fällen zu Verfälschungen kommen. Umgekehrt kann es ebenso vorkommen, dass Einträge fälschlicherweise verpasst werden. Dies ist der Fall, wenn entweder die vorangegangene Fenstergröße gewählt wird und die reale Fenstergröße anstieg oder die zukünftige Fenstergröße kleiner als die zuletzt gewählte ist. Es kann daher keiner der Alternativen ein Vorzug gegeben werden. Für die Simulation wird festgelegt, immer die Fenstergröße der vorangegangenen realen Abfrage zu übernehmen. Werte zwischen den beobachteten Größen wie $W_\mathcal{G}$=5 sind spekulativ und werden nicht in Betracht gezogen.

Selektion fehlerhafter Feeds

Zur Evaluation der Algorithmen wurden von den 200.000 Feeds des Datensatzes die Feeds ausgeschlossen, deren Verarbeitung während der Erstellung aufgrund einer zu hohen Fehlerquote vorzeitig beendet wurde (siehe Abschnitt 3.5.6). Zudem wurden Feeds ausgeschlossen, zu denen zu Beginn des Simulationszeitraums noch keine Daten vorlagen. Dieser Fall trat bei knapp 200 Feeds ein, die in den ersten Stunden bzw. Tagen nicht verfügbar waren.

Darüber hinaus wurden alle Feeds ausgeschlossen, deren Einträge *generell* keine Zeitstempel aufwiesen. In Abschnitt 3.7.7 wurde ermittelt, dass dies knapp 11 % der Feeds betraf. Bei der Erstellung des Datensatzes wurde den Einträgen jeweils der Zeitstempel der Abfrage zugewiesen (Abschnitt 3.5.5), um die Änderungsrate zu approximieren und diese Feeds geeignet abfragen zu können. Zur Erstellung des Datensatzes war dies ausreichend, für eine realistische Evaluation ist dies jedoch ungeeignet: Die Berechnung des Delays, mit dem Einträge in der Simulation empfangen werden, würde auf bereits geschätzen Werten basieren und eine Verzerrung der Ergebnisse bedingen.

Der Ausschluss betrifft alle Feeds, deren Einträge generell keine Zeitstempel aufweisen. Feeds, bei denen Korrekturen der Zeitstempel nur in Einzelfällen auftraten, werden in der Evaluation berücksichtigt. Bezogen auf die Gesamtheit aller Einträge des Datensatzes führt dies zu einer Reduktion von rund einem Drittel. Die drei Feeds mit den meisten Einträgen haben keine Zeitstempel, jedoch 12, 3,5 und 2,5 Millionen Einträge. Eine Analyse ergab zudem, dass diese drei dem On-The-Fly Muster zugeordnet werden können und einer dieser Feeds als Spam zu klassifizieren ist. Die Erkennung von Spam ist nicht Gegenstand der Arbeit. Eine Verzerrung hinsichtlich der Änderungsraten ist dennoch nicht zu erwarten, der in der Simulation eingesetzte Ausschnitt des Datensatzes enthält nach wie vor Feeds mit sehr hohen Änderungsraten wie Yahoo Answers[3] mit mehr als 1,7 Millionen Einträgen.

Für die Evaluation verbleiben somit 89 % der Feeds des Datensatzes, diese haben in der Summe knapp 33 Millionen Einträge.

Behandlung von Misses im Datensatz

In Abschnitt 3.7.8 wurden bei der Erstellung des Datensatzes verpasste Einträge untersucht. Die Analyse ergab, dass 94 % der Feeds keine Lücken aufweisen. Für die Simulation wurde entschieden, die verbleibenden 6 % *nicht* auszuschließen, sondern so zu verarbeiten als ob sie eine vollständige Historie hätten. Der Hintergrund der Entscheidung ist, dass vor allem Feeds mit einer sehr hohen Änderungsrate zu diesen 6 % gehören. Ein Ausschluss dieser Feeds hätte eine Verschiebung des Datensatzes hin zu seltener aktualisierten Feeds zur Folge und würde die realen Verteilungen nicht mehr adäquat repräsentieren. Es sei darauf hingewiesen, dass ein etwaiges burst-Verhalten eines Feeds, das bereits bei der Erstellung des Datensatzes verpasst wurde, in der Evaluation nicht simuliert werden kann.

4.3.3. Metriken zur Evaluation

Grundsätzlich werden die Algorithmen anhand von drei Kriterien beurteilt: 1) der durchschnittlichen Verzögerung Delay, mit der neue Einträge gefunden werden, 2) dem durchschnittlichen Recall, d. h. dem Verhältnis zwischen gefundenen Einträgen und 3) allen Einträgen sowie der durchschnittlichen Anzahl Abfragen, die zum Auffinden eines neuen Eintrags benötigt werden. Diese drei Kriterien werden anschließend zu einem gemeinsamen Wert kombiniert. Neben der Erfassung der Anzahl von Abfragen wird zudem das zu übertragenden Datenvolumen betrachtet. Abschließend erfolgt eine differenzierte Betrachtung, auf welche Art die Durchschnittswerte berechnet werden.

Für ein besseres Verständnis der Metriken seien Einträge zunächst bezüglich ihres Status klassifiziert, ob sie in der Testphase eines Algorithmus gefunden, verpasst oder ignoriert wurden bzw. ob ihr Status offen ist. Dies ist in Abbildung 4.8 beispielhaft dargestellt. Alle Einträge, deren Veröffentlichungsdatum nach dem Ende der Testphase bzw. vor dem ersten simulierten Fenster liegen (a und m), werden ignoriert und gehen nicht in die nachfolgenden Berechnungen ein. Die Einträge b–d wurden bei der ersten Abfrage gefunden, die jeder Algorithmus zum exakt gleichen Zeitpunkt ausführt. Die

[3] `http://answers.yahoo.com/rss/allq`, letzter Zugriff November 2011.

Einträge e, f sowie i–k wurden in der Testphase vom Algorithmus gefunden, g und h sind Misses. Für alle Einträge (l), die nach der letzten Abfrage $\mathcal{C}$, jedoch noch innerhalb der Testphase liegen, kann nicht entschieden werden, ob sie bei der nächsten Abfrage $\mathcal{D}$ gefunden oder verpasst würden, da $\mathcal{D}$ außerhalb der Testphase liegt und daher nicht ausgeführt wird.

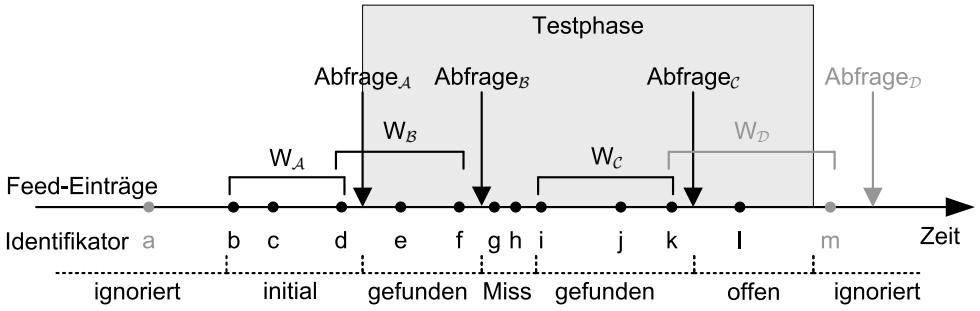

Abbildung 4.8.: Klassifikation von Einträgen innerhalb des Simulationszeitraums.

4.3.4. Herleitung der Metriken

Delay

Die Berechnung des Delay basiert auf den in Abschnitt 2.7.4 (Seite 69) aufgeführten Gleichungen 2.13 – 2.15, die aus [SCC07] abgeleitet wurden. Der Delay eines Eintrags E_i ist die Zeitdifferenz zwischen seinem Veröffentlichungsdatum t_i und dem Zeitpunkt der Abfrage τ_z, in der er als neuer Eintrag identifiziert wurde. Für die erste Abfrage $\mathcal{A}$, bei der Einträge gefunden werden, wird ein Delay von Null definiert. Der Delay wird daher nur für alle als *gefunden* markierten Einträge e, f und i–k jeweils einmalig berechnet.

Im betrachteten Szenario ist der Delay idealerweise (nahe) Null, sodass neue Einträge mit minimaler Verzögerung beim Feed Data Provider vorliegen und dem PIS zur Verfügung gestellt werden. Die Berechnung eines kumulativen Delay birgt den Nachteil, dass dieser unabhängig von der Anzahl gefundener Einträge ist. Ein Algorithmus, der Feeds nur sehr selten abfragt und z. B. nur $\mathcal{A}$ und $\mathcal{C}$ ausführt, erzielt einen geringeren Delay als das gezeigte Beispiel. Aus diesem Grund wird ein durchschnittlicher Delay $\overline{D}(F)$ eingeführt, bei dem der Durchschnitt über alle gefundenen Einträge eines Feeds F gebildet wird. Der Nachteil von $\overline{D}(F)$ ist, dass dieser nur für $|\text{gefunden}| > 0$, d. h. Feeds mit mindestens einem in der Testphase gefundenen Eintrag, definiert ist. Diese Einschränkung wird im Anschluss separat betrachtet.

$$\overline{D}(F) = \frac{1}{|\text{gefunden}|} \sum_{i=1}^{|\text{gefunden}|} D(E_i); \quad |\text{gefunden}| > 0 \tag{4.12}$$

Misses und Recall

Das zweite Evaluationskriterium wird aus der Anzahl verpasster Einträge abgeleitet. Die Bezeichnung Miss wurde bereits in Abschnitt 2.7.4 aus [HLKK08] übernommen und anschließend in Abschnitt 3.5.2 im Rahmen der Erstellung des Datensatzes verfeinert.

Wie bereits aus Abbildung 3.3 bekannt, konnte bei der Erstellung des Datensatzes bei zwei aufeinander folgenden, disjunkten Fenstern $W_{\mathcal{B}}$, $W_{\mathcal{C}}$ nicht zweifelsfrei ermittelt werden, ob ein Eintrag verpasst wurde. Dieser Nachteil besteht in den simulierten Läufen der Algorithmen nicht. Anhand des Datensatzes wird die korrekte Anzahl verpasster Einträge bestimmt und aufgezeichnet, im aktuellen Beispiel sind dies g und h. Hierbei ist es wichtig, zwischen den Sichtweisen der Simulationsumgebung und den zu evaluierenden Algorithmen zur Vorhersage zu differenzieren: Die tatsächliche Anzahl Misses steht *ausschließlich* der Simulationsumgebung zur Auswertung zur Verfügung. Die Algorithmen selbst haben nur Kenntnis von den Feed-Einträgen der Fenster $W_{\mathcal{A}}$–$W_{\mathcal{C}}$. Aus Sicht der Algorithmen kann daher – wie zur Erstellung des Datensatzes – pro Abfrage nur binär bestimmt werden, ob ein potentieller Miss vorlag. Der resultierende, bedeutende Unterschied ist, dass die Anzahl Misses der Evaluation exakt der Anzahl Einträge entspricht, die in der Simulation verpasst wurden, während die Anzahl Misses bei der Erstellung des Datensatzes die Anzahl potentieller Lücken im Datensatz ist. Das Optimum liegt vor, wenn kein Eintrag verpasst wird.

Die Berechnung des Recall R basiert auf den Misses sowie den gefundenen und den als offen klassifizierten Einträgen. Der Recall ist das Verhältnis der gefundenen Einträge zu allen Einträgen, die ein perfekter Algorithmus gefunden hätte. Die initiale Abfrage $\mathcal{A}$ wird nicht in die Berechnung einbezogen. Analog zu Gleichung 4.12 gilt für 4.13, dass der Recall nur für Feeds definiert ist, für die mindestens ein Eintrag gefunden werden kann. Die Metrik wurde in ähnlicher Form in [Sin07] als *percentage observed* verwendet.

$$R(F) = \frac{|\text{gefunden}|}{|\text{gefunden}| + |\text{Miss}| + |\text{offen}|}; \quad (|\text{gefunden}| + |\text{Miss}| + |\text{offen}|) > 0 \tag{4.13}$$

Anzahl Abfragen pro Eintrag

Für jede Abfrage werden Systemressourcen wie CPU und Speicher benötigt, um mit dem Server des Anbieters zu kommunizieren. Unterstützt der Anbieter bedingte GET-Anfragen, so kann zwar das Volumen der zu übertragenden Daten minimiert werden, der Aufbau der Verbindung, die Auswertung der Antwort sowie die Planung der nächsten Abfrage müssen dennoch durchgeführt werden. Aus diesem Grund wird als drittes Kriterium die Anzahl von Abfragen pro Eintrag (APE) eingeführt. APE ist das Verhältnis aus der Anzahl vom jeweiligen Algorithmus vorhergesagter Abfragen ($\mathcal{B}$, $\mathcal{C}$) und allen neuen Einträgen, die bei diesen Abfragen gefunden wurden. Wurde kein neuer Eintrag gefunden, so ist APE nicht definiert. Wie bereits bekannt, wird die initiale Abfrage $\mathcal{A}$ nicht in die Berechnung einbezogen.

$$APE(F) = \frac{|\text{Abfragen}| - 1}{|\text{gefunden}|}; \quad |\text{gefunden}| > 0 \tag{4.14}$$

Mit den beiden Zielen, jeden neuen Eintrag schnellstmöglich zu erlangen, jedoch keine unnötigen Abfragen durchzuführen, ist APE = 1 der Idealwert. Eine Ausnahme bilden Feeds, bei denen mehrere Einträge zum gleichen Zeitpunkt veröffentlicht werden – für diese liegt der Idealwert unter Eins.

Der Delay wird genau dann von APE beeinflusst, wenn durch eine gesteigerte/verringerte Anzahl Abfragen Einträge eher/später gefunden werden. Analog kann APE den Recall beeinflussen, sodass durch eine gesteigerte/verringerte Anzahl Abfragen mehr/weniger Einträge gefunden werden können. Delay und Recall werden jedoch nur dann beeinflusst, wenn in einer Abfrage tatsächlich ein neuer Eintrag gefunden wird. Wird ein Feed hingegen in Zeiten ohne Aktivitäten vermehrt/vermindert abgefragt, hat dies keinen Einfluss auf die beiden anderen Metriken.

Übertragungsvolumen

Die Bestimmung des Übertragungsvolumens ergibt sich aus der Betrachtung der APE und vermittelt einen guten Eindruck von den zu übertragenden Daten. Während Delay, Recall und APE in der Simulation exakt aus dem Datensatz bestimmt werden können, stellt das ermittelte Übertragungsvolumen nur eine Näherung dar. Dies hat zwei Ursachen. Zum einen werden die bei der tatsächlichen Übertragung eingesetzten Komprimierungsverfahren nicht berücksichtigt, sondern nur die tatsächlichen Größen der XML-Dokumente der Feeds betrachtet. Die zweite Ursache erschließt sich aus der Berechnung des simulierten Übertragungsvolumens. Statt für jede simulierte Abfrage die Größe aus allen einzelnen Einträgen zu rekonstruieren, wird die Größe der vorangegangenen realen Abfrage zur Erstellung des Datensatzes verwendet. Dies kann dazu führen, dass das Übertragungsvolumen der simulierten Abfrage in der Realität sowohl größer als auch kleiner gewesen wäre. Aufgrund der großen Anzahl von Einträgen und Abfragen ist anzunehmen, dass die resultierende Abweichung im Mittel nahe Null liegt.

4.3.5. Zwei Modi zur Berechnung des Durchschnitts über einen Datensatz

Die Metriken Delay, Recall und APE wurden zunächst für die Bewertung eines Algorithmus bezüglich eines einzelnen Feeds definiert. Bei den Definitionen wurde zudem auf etwaige Einschränkungen hingewiesen, dass z. B. keine der Metriken für die Bewertung von Feeds der Muster Zombie und Empty verwendet werden kann, da diese keinen einzigen neuen Eintrag innerhalb der Testphase aufweisen. Dies ist ein entscheidender Nachteil. Zur Lösung des Problems werden zwei Modi vorgeschlagen, die zur Berechnung der Durchschnittswerte für einen aus mehreren Feeds bestehenden Datensatz eingesetzt werden können. Formelzeichen zur Bildung von Durchschnittswerten werden mit einer Linie gekennzeichnet, z. B. $\overline{R}$.

Modus Feeds Im Modus Feeds wird jede Metrik zunächst für jeden Feed separat ermittelt und anschließend das arithmetische Mittel einer Metrik über alle l Feeds gebildet, für die diese definiert ist. Der Vorteil des Modus Feeds ist der faire Vergleich aller Feeds, da jeder Feed gleich gewichtet eingeht. Dem gegenüber stehen jedoch mehrere Nachteile. Die Berechnung ist nicht für alle Feeds des Datensatzes möglich,

sodass bestimmte Aktualisierungsmuster wie Zombie und Empty nicht bewertet werden können. Aus Sicht der Einträge werden Feeds mit einer hohen Anzahl neuer Einträge benachteiligt. Dies ließe sich theoretisch durch Bildung eines gewichteten Durchschnitts vermeiden, bei dem die Anzahl der Einträge einbezogen wird. Die Kehrseite eines gewichteten Durchschnitts wäre jedoch, dass Feeds mit sehr vielen Einträgen dominieren und sehr geringe Werte einzelner Feeds nicht erfasst werden. Aus diesem Grund wird im Modus Feeds kein gewichteter Durchschnitt gebildet. Der Modus Feeds wird als Index F an den Metriken geführt, wie in Gleichung 4.15 exemplarisch dargestellt ist. Der Durchschnitt bezieht sich generell auf einen kompletten Datensatz sowie einen Algorithmus, diese beiden Parameter werden jedoch aus Gründen der Übersichtlichkeit nicht aufgeführt. In der Arbeit wird nur ein einziger Datensatz betrachtet und die Zugehörigkeit zu einem Algorithmus ergibt sich aus dem Kontext.

$$\overline{R}_F = \frac{1}{l} \sum_{i=1}^{l} R(F_i); \quad l > 0 \tag{4.15}$$

Modus Einträge Die alternative Ermittlung der Durchschnittswerte erfolgt direkt über die Anzahl aller Einträge aller Feeds des Datensatzes, sodass die Zuordnung von Einträgen zu Feeds nicht beachtet werden muss. Der Vorteil dieses Modus ist neben der einfacheren Berechnung, dass die Metriken für Feeds aller Aktualisierungsmuster verwendet werden können. Einzige Voraussetzung ist, dass in mindestens einem der Feeds ein neuer Eintrag gefunden wird. Analog zum Modus Feeds wird der Modus *Einträge* mit dem Index E gekennzeichnet: z. B. $\overline{R}_E$. Im Gegensatz zum Modus Feeds wird der Modus *Einträge* von Feeds mit einer sehr hohen Anzahl von Einträgen dominiert.

4.3.6. Güte als zusammenfassendes Kriterium

Die Durchschnittswerte für Delay, Recall und APE werden nun zu dem gemeinsamen Kriterium *Güte* G zusammengefasst, welches (innerhalb eines Modus) einen direkten Vergleich der Algorithmen anhand eines einzigen Wertes ermöglicht und somit die in Abschnitt 1.5 aufgestellte These zur Bewertung von Vorhersagen erfüllt. Hierzu werden zunächst Anforderungen an ein solches Kriterium aufgestellt, anschließend verschiedene Möglichkeiten der Realisierung eruiert und schließlich eine Variante gewählt. Die Berechnungen erfolgen für beide Modi Feeds und Einträge analog, weshalb in den Ausführungen dieses Abschnitts auf die Indices E und F verzichtet wird.

Anforderungen an die Güte

In den Evaluationen in Abschnitt 4.4 sowie Kapitel 5 werden die in Abschnitt 4.2 aufgeführten Algorithmen zunächst einzeln betrachtet, um etwaige Parameter zu bestimmen, und anschließend die jeweils Besten miteinander verglichen.

Ausgangspunkt sind die drei Kriterien $\overline{D}$, $\overline{R}$ und $\overline{APE}$. $\overline{R}$ liegt generell im Intervall [0;1], $\overline{D}$ und $\overline{APE}$ haben theoretische Wertebereiche von [0;∞] und (0;∞]. In den in der Arbeit

betrachteten Evaluationen beträgt der Maximalwert des durchschnittlichen Delays wenige Tage und der höchste Wert für $\overline{APE}$ liegt bei mehreren Hundert. Die Güte ist aus diesen drei zu bilden.

Es sei Φ die Menge *aller* insgesamt zu betrachtenden Algorithmen und Ω die Menge aller in *eine* Evaluation einzubeziehenden Algorithmen mit $\Omega \subseteq \Phi$. Die erste Bedingung an die Güte ist, dass alle Algorithmen $\omega \in \Omega$ anhand der Güte geordnet werden können. Angenommen, für zwei Algorithmen ω_1 und ω_2 gilt: ω_1 hat einen doppelt so hohen Delay wie ω_2, jedoch einen nur halb so hohen APE-Wert und der Recall beider ist gleich; so ist beiden Algorithmen die gleiche Güte zuzuweisen. Auf diese Weise kann z. B. der Parameter M des Algorithmus AdaptiveTTL anhand der Güte empirisch bestimmt werden, indem der Wert gewählt wird, mit welchem die höchste Güte erzielt werden kann.

Um den in einer Evaluation als besten ermittelten Algorithmus in anderen Evaluationen mit weiteren Algorithmen $\phi \in \Phi, \phi \notin \Omega$ zu vergleichen, ergibt sich die zweite Anforderung, dass sich die anhand der Güte ermittelte Reihenfolge der Algorithmen ω nicht ändert, wenn weitere Algorithmen ϕ in die Betrachtung einbezogen werden.

Der Idealfall liegt vor, wenn die Betrachtung der weiteren Algorithmen ϕ gar keinen Einfluss auf die Güte der Algorithmen ω hat, sodass nicht nur die Reihenfolge sondern auch die konkreten Werte unverändert bleiben. Dies sei Anforderung drei.

Herleitung der Güte

Ein simpler Ansatz ist, die Güte direkt aus den drei Werten $\overline{D}$, $\overline{R}$ und $\overline{APE}$ zu bilden. Dies ist jedoch nicht zielführend, da ein Algorithmus umso besser ist, je geringer $\overline{D}$ und $\overline{APE}$ sind; $\overline{R}$ ist hingegen zu maximieren. Aus diesem Grund wird vorgeschlagen, die Werte zunächst zu normieren und die Güte aus den normierten Werten zu berechnen.

Eine Normierung kann u. a. auf den besten, schlechtesten oder einen als Baseline gewählten Algorithmus stattfinden. Der Bezug kann zudem lokal, global oder dem theoretisch Besten bzw. Schlechtesten erfolgen und wird am Beispiel Delay sowie einigen exemplarischen Algorithmen eruiert. Für die anschließende Bildung der Güte werden das arithmetische, geometrische und harmonische Mittel betrachtet.

Die Normierung als Relation zu einem anderen Algorithmus sei am Beispiel der Normierung auf den Besten erläutert und im Anschluss zur Normierung auf einen beliebigen Algorithmus generalisiert.

Bei einer Normierung auf den Besten wird diesem Algorithmus der Wert 1 zugeteilt und die Werte aller übrigen Algorithmen in Relation zum Besten ermittelt. Die Berechnung des normierten Delays (sowie auch APE) erfolgen nach Gleichung 4.16, die des Recalls nach 4.17. Die resultierenden normierten Werte sind den Spalten 5–7 in Tabelle 4.1 zu entnehmen. In den Spalten 8–10 folgen die Werte der drei verschiedenen Möglichkeiten, einen Mittelwert aus den normierten Werten zu bilden.

$$\widehat{D}(\omega) = \frac{\operatorname{argmin}_{x \in \Omega}(\overline{D}(x))}{\overline{D}(\omega)}; \quad \omega \in \Omega, \overline{D}(\omega) > 0 \tag{4.16}$$

$$\widehat{R}(\omega) = \frac{\overline{R}(\omega)}{\text{argmax}_{x\in\Omega}(\overline{R}(x))}; \quad \omega \in \Omega, \underset{x\in\Omega}{\text{argmax}}(\overline{R}(x)) > 0 \tag{4.17}$$

Algorithmus	Durchschnittswerte			auf Besten normiert			Mittelwerte		
	$\overline{D}$	$\overline{APE}$	$\overline{R}$	$\widehat{D}$	$\widehat{APE}$	$\widehat{R}$	arith.	geom.	harm.
A1	30min	3,200	0,950	1,000	0,375	0,969	0,781	0,714	0,639
A2	100min	2,100	0,960	0,300	0,571	0,980	0,617	0,552	0,491
A3	200min	1,200	0,980	0,150	1,000	1,000	0,717	0,531	0,346

Tabelle 4.1.: Vergleich verschiedener Mittelwerte bei Normierung auf den Besten.

Im Beispiel hat ω=$A1$ mit $\overline{D}_E(A1)$=30 min den geringsten Delay, es ergibt sich der normierte Wert Eins. $\widehat{D}_E(A2) = \frac{\overline{D}_E(A1)}{\overline{D}_E(A2)} = 0{,}300$; $\widehat{D}_E(A3) = \frac{\overline{D}_E(A3)}{\overline{D}_E(A1)} = \frac{30\,min}{200\,min} = 0{,}150$.

Wie am Beispiel der Algorithmen $A2$ und $A3$ gut zu erkennen ist, nimmt die Normierung keinen Einfluss auf die Verhältnisse der Algorithmen zueinander. $A3$ hat mit 200 Minuten einen doppelt so hohen durchschnittlichen Delay $\overline{D}$, sodass sich ein nur halb so hoher, normierter Wert $\widehat{D}$ im Vergleich zu $A2$ ergibt. Dies ist eine bekannte Eigenschaft der Potenzfunktion $f(x) = cx^{-1}$, die sich bei genauer Betrachtung hinter Gleichung 4.16 verbirgt. Neben der Abbildung des Wertebereichs $(0;\infty) \to (0;1]$ erzielt diese Variante der Normierung den Effekt, besonders gute Werte feiner zu unterscheiden als relativ schlechte Werte.

Bei der anschließenden Bildung des Mittelwerts sind klare Unterschiede in der sich ergebenden Reihenfolge zu erkennen: Während das arithmetische Mittel die Reihenfolge $A1$, $A3$, $A2$ ergibt, ist diese bei den anderen beiden Mittelwerten $A1$, $A2$, $A3$. Zur Betrachtung der Ursachen und Auswirkungen bezüglich der Eignung als Güte wird das aktuelle Beispiel zunächst um einige Algorithmen zu Tabelle 4.2 erweitert.

Durch die Einführung eines neuen besten Algorithmus $A4$ für den Delay $\overline{D}$ müssen alle normierten Werte $\widehat{D}$ neu berechnet werden. $A4$ hat einen um den Faktor zehn geringeren Delay $\overline{D}$ als $A1$, sodass die normierten Werte $\widehat{D}$ für die Algorithmen $A1$–$A3$ in

Algorithmus	Durchschnittswerte			auf Besten normiert			Mittelwerte		
	$\overline{D}$	$\overline{APE}$	$\overline{R}$	$\widehat{D}$	$\widehat{APE}$	$\widehat{R}$	arith.	geom.	harm.
A1	30min	3,200	0,950	0,100	0,375	0,969	0,481	0,331	0,219
A2	100min	2,100	0,960	0,030	0,571	0,980	0,527	0,256	0,083
A3	200min	1,200	0,980	0,015	1,000	1,000	0,672	0,247	0,044
A4	3min	3,200	0,980	1,000	0,375	1,000	0,792	0,721	0,643
A5	6min	1,600	0,980	0,500	0,750	1,000	0,750	0,721	0,692
A6	6min	6,400	0,490	0,500	0,188	0,500	0,396	0,361	0,321
A7	24min	3,200	0,980	0,125	0,375	1,000	0,500	0,361	0,257
A8	2000min	1,200	0,980	0,002	1,000	1,000	0,667	0,114	0,004

Tabelle 4.2.: Erweiterung des Vergleichs verschiedener Mittelwerte bei Normierung auf den Besten.

Tabelle 4.2 auf ein Zehntel des ursprünglichen Wertes in Tabelle 4.1 fallen. Das Verhältnis untereinander bleibt erhalten.

Vergleicht man nun die Mittelwerte von $A1$–$A3$ beider Tabellen 4.1 und 4.2, so stellt man fest, dass sich die aus dem arithmetischen Mittel ergebende Reihenfolge ändert (von $A1$, $A3$, $A2$ zu $A3$, $A2$, $A1$), sodass der bisherige Beste der drei nun der schlechteste ist. Die Reihenfolge der anderen beiden Mittelwerte bleibt gleich. Die Ursache des Vertauschens der Reihenfolge beim arithmetischen Mittel ist, dass durch die neue Normierung des Delays alle drei Algorithmen sehr kleine Werte für $\overline{D}$ erhalten und somit $\widehat{APE}$ und $\widehat{R}$ das Ergebnis stärker beeinflussen. Die Kombination einer multiplikativen Normierung und eines additiven Mittelwerts ist somit ungeeignet.

Obwohl die sich für $A1$–$A3$ ergebende Reihenfolge bei Verwendung des geometrischen und harmonischen Mittels gleich bleibt ist dennoch ersichtlich, dass sich beim harmonischen Mittel die Verhältnisse der Algorithmen untereinander deutlich ändern und nur beim geometrischen Mittel exakt gleich bleiben[4]. Die Ursache der Abweichung des harmonischen Mittels ist dessen Eigenschaft, Ausreißer zu „bestrafen“, sodass die sehr niedrigen Werte $\widehat{D}$ von $A2$ und $A3$ betont werden und der Mittelwert entsprechend niedrig ausfällt. Diese Sensibilität gegenüber Ausreißern ist in einigen Anwendungsgebieten wie der Bildung des *F1-Score* im Information Retrieval gewünscht: Der F1-Score wird als harmonisches Mittel aus Precision und Recall gebildet [MRS08]. Für die Bewertung der Güte ist dies jedoch nicht zielführend, wie die Betrachtung von $A4$–$A7$ zeigt.

Basis der folgenden Betrachtungen sei $A4$. $A5$ weist einen doppelt so hohen Delay und halb so viele Abfragen auf, der Recall ist gleich. Sollen alle drei Metriken gleich gewichtet werden, so haben beide Algorithmen intuitiv die selbe Güte – wie in Anforderung 1 formuliert wurde. Das geometrische Mittel genügt dieser Anforderung, während beim harmonischen Mittel der recht kleine Wert $\widehat{APE} = 0,375$ von $A4$ einen größeren Einfluss als die anderen Werte hat und ein kleinerer (harmonischer) Mittelwert resultiert.

$A6$ ist in allen drei Kriterien genau halb so gut wie $A4$. $A7$ und $A4$ unterscheiden sich nur in $\overline{D}$, $A7$ hat den achtfachen Delay. Betrachtet man die Mittelwerte, so bilden alle drei die Halbierung der Werte von $A6$ korrekt ab, unterscheiden sich jedoch deutlich für $A7$. Einzig das geometrische Mittel liefert das gewünschte Resultat und weist $A7$ einen halb so hohen Wert wie $A4$ zu. $A7$ und $A6$ sind somit im geometrischen Mittel identisch. Umgekehrt betrachtet ist $A4$ jeweils doppelt so gut wie $A6$ und $A7$, unabhängig davon, ob $A4$ in allen drei Kategorien doppelt so gut ist $(2 \cdot 2 \cdot 2)$, oder in einer Kategorie acht mal und in den anderen beiden gleich gut $(2^3 \cdot 1 \cdot 1)$.

Zusammenfassend ergibt sich, dass bei der Normierung auf den besten Algorithmus nur das geometrische Mittel den Anforderungen 1 und 2 an die Güte entspricht. Erfolgen mehrere Evaluationen sequentiell, so kann die Normierung entweder auf den jeweiligen (lokal) Besten, den Besten aller Evaluationen oder einen als Baseline gewählten Algorithmus stattfinden. Die Varianten weisen verschiedene Vor- und Nachteile auf. Bei Normierung auf den lokal Besten unterscheiden sich die Gütewerte zwischen verschiedenen Evaluationen, wie das Beispiel $A1$–$A3$ in den Tabellen 4.1 und 4.2 zeigte. Die Reihenfolge ist dennoch

[4] Beispiel Verhältnis $A1$ zu $A3$: geometrisches Mittel $\frac{0,714}{0,531} \simeq \frac{0,331}{0,247} \simeq 1,340$, die Ungleichheit ist die Folge von Rundungsfehlern der normierten Werte. Harmonisches Mittel: $\frac{0,639}{0,346} = 1,847$, $\frac{0,219}{0,044} = 4,977$

stabil. Bei der Normierung auf den global Besten, d. h. den Besten aller Evaluationen sind neben der Reihenfolge auch die Werte der Güte stabil und können direkt miteinander verglichen werden. Der entstehende Nachteil ist, dass der Beste bereits vorab bekannt sein muss, sowie in jeder Evaluation mit aufzuführen ist. Eine Normierung auf einen als Baseline gewählten Algorithmus ist ebenfalls möglich, hat jedoch den selben Nachteil, dass dieser in allen Evaluationen aufzuführen ist. Die Wahl der Baseline hat weder Einfluss auf die Reihenfolge der Algorithmen noch deren Verhältnisse der geometrischen Mittel untereinander. Ein weiterer Effekt der Normierung auf eine Baseline ist, dass die normierten Werte nicht mehr im Intervall (0;1] liegen, sondern einen Wertebereich von (0;∞) aufweisen: Algorithmen die in einer Metrik besser sind als die Baseline erhalten einen normierten Wert größer Eins und schlechtere Algorithmen einen Wert kleiner Eins. Eine Normierung auf einen theoretisch besten Algorithmus ist nicht möglich, da dieser einen Delay von Null hat und es daher nicht möglich ist, die betrachteten Algorithmen zu diesem ins Verhältnis zu setzen.

Die Normierung auf den jeweiligen (lokal) besten Algorithmus einer Evaluation und die Verwendung des geometrischen Mittels ist somit der aussichtsreichste Kandidat zur Berechnung der Güte.

Die lineare Normierung sei der Vollständigkeit halber als Alternative aufgeführt. Die Idee ist, dem schlechtesten Algorithmus Null als normierten Wert zuzuweisen und die Durchschnittswerte der übrigen Algorithmen linear auf das Intervall [0;1] abzubilden. Die Berechnung des normierten Delays und der Anzahl Abfragen pro neuem Eintrag erfolgt nach Gleichung 4.18, für den Recall nach Gleichung 4.19. Die sich für das bereits bekannte Beispiel $A1$–$A8$ ergebenden normierten und Durchschnittswerte sind Tabelle 4.3 zu entnehmen.

$$\widehat{D}'(\omega) = 1 - \frac{\overline{D}(\omega)}{\operatorname{argmax}_{x\in\Omega}(\overline{D}(x))}; \quad \omega \in \Omega, \operatorname*{argmax}_{x\in\Omega}(\overline{D}(x)) > 0 \tag{4.18}$$

$$\widehat{R}'(\omega) = 1 - \frac{\overline{R}(\omega)}{\operatorname{argmax}_{x\in\Omega}(1-\overline{R}(x))}; \quad \omega \in \Omega, \operatorname*{argmax}_{x\in\Omega}(1-\overline{R}(x)) > 0 \tag{4.19}$$

Die Normierung des Delay $\widehat{D}'$ erfolgt anhand des Ausreißers $A8$, sodass den übrigen Algorithmen ausnahmslos sehr hohe Werte mit $\widehat{D}' \geq 0{,}9$ zugeordnet werden. Bei der Betrachtung der Mittelwerte fällt zunächst auf, dass das harmonische Mittel für $A6$ und $A8$ nicht definiert ist. Formal betrachtet ist das harmonische Mittel nur für Algorithmen definiert, die in keiner der Metriken den insgesamt schlechtesten Wert aufweisen. Diese Eigenschaft ist besonders schlecht, wenn nur sehr wenige Algorithmen betrachtet werden und kann dazu führen, dass für gar keinen Algorithmus ein Mittelwert bestimmbar ist.

Das geometrische Mittel wird in diesen Fällen generell Null und ist ebenfalls nicht zum Vergleich der Algorithmen geeignet. Der verbleibende Kandidat – das arithmetische Mittel – erweist sich ebenfalls als ungeeignet, wie der Vergleich der Werte von $A4$ und $A5$ bzw. $A6$ und $A7$ zeigt. Diese müssten paarweise gleich sein. Die lineare Normierung wird daher als ungeeignet betrachtet.

Algorithmus	Durchschnittswerte			lineare Normierung			Mittelwerte		
	$\overline{D}$	$\overline{APE}$	$\overline{R}$	$\widehat{D}'$	$\widehat{APE}'$	$\widehat{R}'$	arith.	geom.	harm.
A1	30min	3,200	0,950	0,985	0,500	0,902	0,796	0,763	0,727
A2	100min	2,100	0,960	0,950	0,672	0,922	0,848	0,838	0,827
A3	200min	1,200	0,980	0,900	0,813	0,961	0,891	0,889	0,887
A4	3min	3,200	0,980	0,999	0,500	0,961	0,820	0,783	0,742
A5	6min	1,600	0,980	0,997	0,750	0,961	0,903	0,896	0,888
A6	6min	6,400	0,490	0,997	0,000	0,000	0,332	0,000	-
A7	24min	3,200	0,980	0,988	0,500	0,961	0,816	0,780	0,740
A8	2000min	1,200	0,980	0,000	0,813	0,961	0,591	0,000	-

Tabelle 4.3.: Vergleich verschiedener Mittelwerte bei linearer Normierung auf den Schlechtesten.

Definition der Güte $\widehat{G}$

Zur Berechnung der Güte eines Algorithmus erfolgt zuerst eine Normierung der Werte der drei Metriken $\overline{D}$, $\overline{APE}$ und $\overline{R}$. Die Berechnung erfolgt anhand der Gleichungen 4.16 und 4.17, die normierten Werte werden mit $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ bezeichnet. Zur Ermittlung der Güte wird zunächst das (gleich gewichtete) geometrische Mittel nach Gleichung 4.20 aus $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ gebildet. Die Betrachtung eines gewichteten Gütemaßes folgt in Abschnitt 5.3.1.

$$G = \sqrt[3]{\widehat{D} \cdot \widehat{APE} \cdot \widehat{R}} \tag{4.20}$$

Wenngleich dies bereits als Gütemaß geeignet wäre, erfolgt dennoch eine abschließende Normierung von G zu $\widehat{G}$ nach Gleichung 4.21. Dies hat keinen Einfluss auf die Relationen der Algorithmen untereinander, sondern dient einzig dem Ziel ein leichter verständliches Maß zu erhalten. Dem besten Algorithmus wird eine Güte von Eins zugewiesen, allen anderen Algorithmen ein anteiliger Wert. Ist ein Algorithmus $A5$ z. B. in allen drei Kriterien doppelt so gut wie $A6$, so hat er eine doppelt so hohe Güte. Dies sei abschließend in Tabelle 4.4 gezeigt, die einen Ausschnitt des aktuellen Beispiels enthält. Ferner sei mit dem Begriff Güte generell der normierte Wert $\widehat{G}$ gemeint.

Mit der gewählten Berechnung genügt die Güte den ersten beiden aufgestellten Anforderungen. Die dritte Anforderung kann erfüllt werden, indem in jeder Evaluation die global besten Algorithmen (aller Evaluationen) mit aufgeführt werden. Hiervon wird jedoch aus Gründen der Übersichtlichkeit abgesehen, sodass die Ermittlung des besten Algorithmus generell innerhalb einer Evaluation stattfindet und der Beste anschließend in einer weiteren Evaluation betrachtet werden kann.

Mit der Definition der Güte ist die in Abschnitt 1.5 aufgestellte These zur Bewertung von Vorhersagen erfüllt.

$$\widehat{G}(\omega) = \frac{G(\omega)}{\operatorname{argmax}_{x\in\Omega}(G(x))}; \quad \omega \in \Omega, \operatorname*{argmax}_{x\in\Omega}(G(x)) > 0 \tag{4.21}$$

Algorithmus	Durchschnittswerte			auf Besten normiert			G	$\widehat{G}$
	$\overline{D}$	$\overline{APE}$	$\overline{R}$	$\widehat{D}$	$\widehat{APE}$	$\widehat{R}$		
A3	200min	1,200	0,980	0,015	1,000	1,000	0,247	0,342
A4	3min	3,200	0,980	1,000	0,375	1,000	0,721	1,000
A5	6min	1,600	0,980	0,500	0,750	1,000	0,721	1,000
A6	6min	6,400	0,490	0,500	0,188	0,500	0,361	0,500
A7	24min	3,200	0,980	0,125	0,375	1,000	0,361	0,500

Tabelle 4.4.: Beispiel zur Berechnung der Güte aus den normierten Werten.

Abschließend sei noch einmal in Erinnerung gerufen, dass die Berechnung der normierten Werte und der Güte in beiden Modi Feeds und Einträge (siehe Abschnitt 4.3.5) analog erfolgt und mit den bekannten Indizes F und E als $\widehat{D}_F$, $\widehat{APE}_F$, $\widehat{R}_F$ und $\widehat{G}_F$ bzw. $\widehat{D}_E$, $\widehat{APE}_E$, $\widehat{R}_E$ und $\widehat{G}_E$ dargestellt wird. Zudem sei bemerkt, dass die Werte der Tabellen im nachfolgenden Abschnitt 4.4 ebenfalls gerundet werden, die interne Rechnung jedoch mit einer höheren Genauigkeit wie z. B. Sekunden erfolgt, sodass in den Tabellen marginale Rundungsfehler vorkommen können.

4.3.7. Zusammenfassung

Die Evaluation der Algorithmen erfolgt anhand eines Ausschnitts des in Kapitel 3 beschriebenen Datensatzes, der knapp 179.000 Feeds umfasst. Mittels einer Simulationsumgebung wird eine Testphase von 20 Tagen durchgeführt, der eine optionale Trainingsphase von 7 Tagen vorgelagert ist. Bei der Erstellung des Datensatzes ermittelte Misses werden in der Simulation nicht betrachtet.

Die Bewertung der Algorithmen basiert auf dem Delay, Recall und der Anzahl Abfragen pro neuem Eintrag. Diese Durchschnittswerte werden in den Modi Feeds und Einträge ermittelt, anschließend auf den jeweils besten Algorithmus normiert und daraus eine Güte abgeleitet.

4.4. Evaluation der Algorithmen

Mit dem in den vorangegangenen Kapiteln gewonnenen Wissen über den Datensatz, die Algorithmen, die Simulationsumgebung sowie die zur Bewertung eingesetzten Metriken kann in diesem Abschnitt die Evaluation der Algorithmen erfolgen. Hierfür werden zunächst in Abschnitt 4.4.1 die für einige Algorithmen benötigten Parameter wie M für AdaptiveTTL bestimmt, um die jeweils bestmöglichen Werte für den Datensatz zu ermitteln. Anschließend werden die jeweiligen Gewinner mit allen anderen Algorithmen verglichen. Die Bewertung erfolgt anhand der Güte sowie anhand der Metriken Delay, APE und Recall. Darüber hinaus wird das zu übertragende Datenvolumen untersucht, welches die jeweiligen Algorithmen verursachen.

In allen Evaluationen dieses Abschnittes wird der Einfluss der Schranken gering gehalten. Ein Wert von $\alpha = 1$ Minute wird als realistische untere Schranke für das betrachtete

Szenario erachtet. Der oberen Schranke β wird ein Wert von vier Wochen zugewiesen, sodass sie keinen Einfluss auf die Ergebnisse hat. Optimierungen der Algorithmen auf Basis der Schranken werden in Kapitel 5 separat betrachtet, in diesem Kapitel wird zunächst das „natürliche Verhalten“ der Algorithmen untersucht.

4.4.1. Bestimmung der Parameter für AdaptiveTTL, IndHist und LIHZ

Die Bestimmung der Parameter der Algorithmen AdaptiveTTL, IndHist, IndHist/TTL und LIHZ erfolgte empirisch, wobei umfangreiche Experimente mit bis zu 20 Iterationen pro zu bestimmendem Parameter vollzogen wurden. Die Ergebnisse werden im Folgenden auf die wesentlichen Resultate verkürzt dargestellt und analysiert. An den vollständigen Evaluationsergebnissen interessierte Leser seien auf die jeweiligen Abschnitte im Anhang verwiesen. Diese enthalten eine tabellarische und grafische Darstellung. Für das allgemeine Verständnis der Parameterbestimmung werden diese jedoch nicht benötigt.

AdaptiveTTL

Wie in Abschnitt 2.5.3 bereits eingeführt, wird in der Domäne Web Caching häufig ein Wert $M = 0{,}1$ oder $M = 0{,}2$ als geeignet angegeben. Um zu untersuchen, inwiefern sich dies für die Vorhersage der Zeitpunkte neuer Einträge eignet, wurde M zunächst zwischen 0,1 und 1,0 in Schritten von 0,1 variiert und anschließend Iterationen mit der Schrittweite 1 zwischen 1 und 10 durchgeführt. Ein repräsentativer Ausschnitt der Ergebnisse ist den Tabellen 4.5 und 4.6 zu entnehmen, die vollständigen Resultate befinden sich im Anhang in den Tabellen A.1 und A.2. Die jeweils besten Werte einer Spalte sind hervorgehoben. Eine Visualisierung der normierten Werte sowie der Güte ist in Abbildung 4.9 dargestellt, eine vergrößerte Darstellung befindet sich im Anhang als Abbildung A.1.

Beim Vergleich der *durchschnittlichen* Werte fällt zunächst auf, dass $\overline{D}_F$ und $\overline{D}_E$ mit steigendem M kontinuierlich wachsen, $\overline{R}_F$ und $\overline{R}_E$ kontinuierlich fallen und $\overline{APE}_F$ und $\overline{APE}_E$ nur bis zu einem Scheitelpunkt sinken und anschließend wieder steigen. Die entsprechenden normierten Werte $\widehat{D}_F$, $\widehat{D}_E$ und $\widehat{\overline{APE}}_F$, $\widehat{\overline{APE}}_E$ verhalten sich gegenläufig. Abgesehen von den Scheitelpunkten von $\overline{APE}_F$ und $\overline{APE}_E$ war dieses Ergebnis zu erwarten: Der Parameter M ist indirekt proportional zur Frequenz, mit der die Feeds abgefragt werden. Je kleiner M, desto häufiger wird abgefragt, sodass mit einer sinkenden Anzahl Abfragen einerseits die Verzögerung steigt, mit der neue Einträge gefunden werden, und andererseits die Anzahl verpasster Einträge zunimmt. Umgekehrt folgt, dass mit kleinerem M der Delay und der Recall verbessert werden.

Die beiden nicht erwarteten Scheitelpunkte galt es zu analysieren. Die Untersuchungen ergaben, dass sie auf zwei Ursachen zurückzuführen sind: die begrenzte Simulationsdauer und das Standardintervall η, welches in Abschnitt 4.2.1 eingeführt wurde.

Mit wachsendem M wurden alle Feeds immer seltener abgefragt. Die Anomalie, dass bei sinkender Anzahl Abfragen der Wert APE steigt, sei in Abbildung 4.10 anhand eines realen Feeds und den Werten $M = 8{,}0$ und $M = 9{,}0$ skizziert. In beiden Varianten wird die initiale Abfrage $\mathcal{A}$ ausgeführt. Für $M = 8{,}0$ ergeben sich die beiden Abfragen $\mathcal{B}$ und $\mathcal{C}$, sodass mit zwei Abfragen sechs neue Einträge gefunden werden und $\text{APE}_{M=8} = 0{,}\overline{3}$

AdaptiveTTL	Durchschnittswerte			auf Besten normiert			
M	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
0,1	12h:24min	26,061	0,940	1,000	0,017	1,000	0,653
0,5	33h:25min	3,778	0,813	0,371	0,118	0,864	0,851
1,0	46h:03min	1,788	0,708	0,269	0,250	0,754	0,938
2,0	61h:08min	0,948	0,586	0,203	0,472	0,623	0,989
3,0	66h:10min	0,707	0,488	0,187	0,632	0,520	1,000
4,0	73h:36min	0,620	0,444	0,168	0,721	0,472	0,977
5,0	75h:16min	0,538	0,396	0,165	0,831	0,421	0,979
6,0	75h:05min	0,485	0,352	0,165	0,923	0,374	0,975
7,0	76h:15min	0,447	0,326	0,163	1,000	0,347	0,971
8,0	74h:29min	0,469	0,298	0,166	0,953	0,317	0,934
9,0	74h:17min	0,471	0,274	0,167	0,949	0,291	0,908
10,0	76h:01min	0,468	0,260	0,163	0,956	0,277	0,888

Tabelle 4.5.: Parameterbestimmung AdaptiveTTL, Durchschnitt über alle Feeds.

AdaptiveTTL	Durchschnittswerte			auf Besten normiert			
M	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
0,1	17min	4,147	0,844	1,000	0,179	1,000	1,000
0,2	29min	2,504	0,834	0,593	0,297	0,989	0,990
0,4	50min	1,505	0,819	0,344	0,493	0,970	0,972
0,6	1h:08min	1,155	0,802	0,255	0,643	0,950	0,954
0,8	1h:25min	0,986	0,790	0,202	0,753	0,966	0,927
1,0	1h:39min	0,897	0,779	0,174	0,828	0,923	0,905
2,0	2h:33min	0,743	0,710	0,113	1,000	0,842	0,809
3,0	3h:18min	0,764	0,618	0,087	0,972	0,733	0,703
10,0	5h:48min	1,118	0,363	0,050	0,664	0,430	0,429

Tabelle 4.6.: Parameterbestimmung AdaptiveTTL, Durchschnitt über alle Einträge.

folgt. Für $M = 9{,}0$ wird nur $\mathcal{E}$ ausgeführt, die zweite Abfrage $\mathcal{F}$ liegt bereits außerhalb der Testphase, sodass $\mathrm{APE}_{M=9,0} = 0{,}5$ folgt.

Das Standardintervall η wurde eingeführt, um Feeds abzufragen, bei denen es nicht möglich ist ein Intervall anhand der Einträge zu ermitteln. Je seltener alle übrigen Feeds abgefragt werden, desto stärker fallen diese ca. 10 % der Feeds ins Gewicht. Theoretisch könnte η ebenfalls mit M bzw. den jeweiligen Parametern der anderen Algorithmen multipliziert werden, um es zu variieren. Dies wurde jedoch nicht umgesetzt, da der Effekt einerseits erst in einem Bereich auftritt, in dem die Algorithmen bereits ohnehin schlechte Ergebnisse liefern und andererseits alle zu evaluierenden Algorithmen gleichermaßen beeinflusst werden.

Ein bedeutender Unterschied zwischen den beiden Modi Feeds und Einträge ist beim durchschnittlichen Delay $\overline{D}_F$ und $\overline{D}_E$ erkennbar. Der kleinste Wert von $\overline{D}_F$ ist in etwa um den Faktor 30 größer als $\overline{D}_E$. Dies ist auf die Zipf-Verteilung der Einträge zurückzuführen, die bereits in Abbildung 3.9 (Seite 100) untersucht wurde. Im Modus Feeds wird zuerst über die Einträge der Feeds gemittelt und anschließend ein Durchschnittswert über alle Feeds gebildet. Aufgrund der Zipf-Verteilung der Einträge und Änderungsraten ist

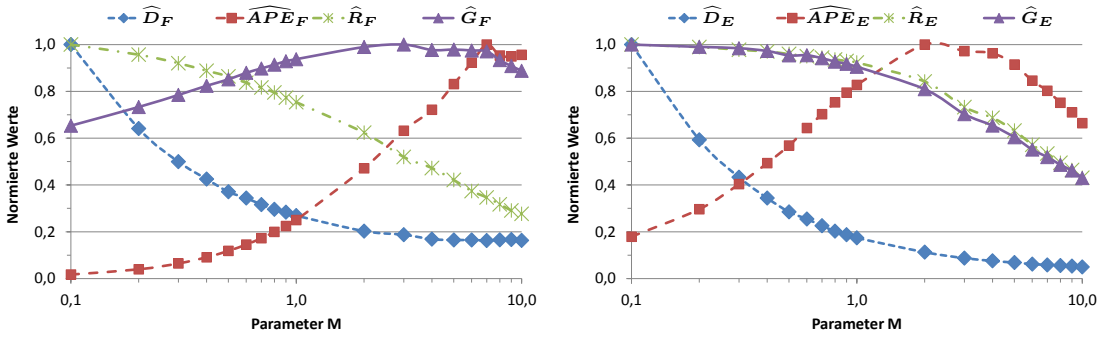

Abbildung 4.9.: Parameterbestimmung AdaptiveTTL, Betrachtung der normierten Größen $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ sowie der Güte $\widehat{G}$. Vergleich des Durchschnitts über alle Feeds (links) und Einträge (rechts).

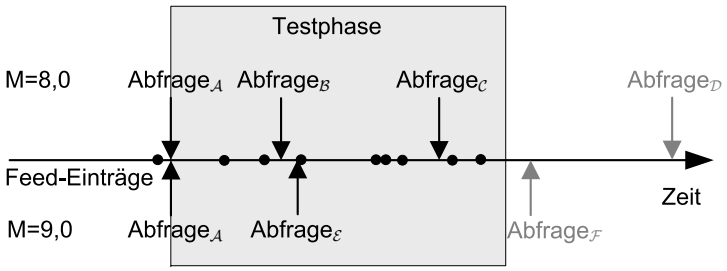

Abbildung 4.10.: Anomalie steigender APE bei sinkender Anzahl Abfragen.

bei einem Großteil der Feeds nur selten ein neuer Eintrag zu verzeichnen, sodass diese Einträge mit durchschnittlichen Verzögerungen zwischen 12 Stunden und mehreren Tagen gefunden werden. Im Modus Einträge hingegen werden alle Einträge gleich gewichtet und der Durchschnittswert von sich sehr häufig ändernden Feeds mit hohen Änderungsraten dominiert. Als Ausreißer zu bezeichnende einzelne Einträge mit einem Delay von mehreren Tagen oder gar Wochen haben nahezu keinen Einfluss.

Die bisherigen Evaluationsergebnisse zeigen bereits, dass dieser Unterschied eine hohe Bedeutung hat. Im Modus Feeds erzielt $M = 3{,}0$ die höchste Güte, im Modus Einträge der Wert $M = 0{,}1$. Die Besten eines Modus erzielen im anderen Modus jeweils rund 30 % schlechtere Resultate. Aufgrund der unterschiedlichen Berechnung beider Modi ist die Bildung des Mittelwerts aus $\widehat{G}_F$ und $\widehat{G}_E$ ungeeignet. Um AdaptiveTTL dennoch fair mit den anderen Algorithmen zu vergleichen, werden beide „Gewinner“ in Abschnitt 4.4.2 als $\text{AdaptiveTTL}_{0,1}$ und $\text{AdaptiveTTL}_{3,0}$ aufgegriffen.

IndHist und IndHist/TTL

Zur Evaluation von IndHist sind zwei Schritte nötig. Zum einen muss das Modell der Änderungsrate gelernt werden, zum anderen weist IndHist einen Parameter θ auf, der geeignet zu wählen ist [BGR06]. Um das Modell zu trainieren, stand den Autoren von

IndHist	Durchschnittswerte			auf Besten normiert			
θ	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
0,1	11h:39min	15,522	0,952	1,000	0,017	1,000	0,749
0,7	37h:47min	2,736	0,809	0,308	0,095	0,851	0,854
5,0	69h:49min	0,523	0,447	0,167	0,497	0,470	0,992
8,0	72h:19min	0,384	0,348	0,161	0,676	0,366	0,999
10,0	71h:07min	0,339	0,303	0,164	0,765	0,318	1,000
30,0	52h:15min	0,260	0,116	0,223	1,000	0,122	0,879
50,0	42h:11min	0,268	0,066	0,276	0,971	0,069	0,776
100,0	28h:31min	0,312	0,028	0,408	0,834	0,029	0,630

Tabelle 4.7.: Parameterbestimmung IndHist, Durchschnitt über alle Feeds.

[BGR06] eine Log-Datei zu Verfügung, aus der sie Änderungen der betrachteten Objekte ableiten konnten. Dieses Verfahren wird übernommen, indem das Modell anhand aller Einträge des Datensatzes gelernt wird, deren (korrigiertes) Veröffentlichungsdatum vor dem Ende der Trainingsphase liegt. Es ist zu beachten, dass auf diese Weise ein „perfektes“ Modell gelernt wird, das bei einem realistischen Einsatz des Algorithmus nur schwer zu erstellen wäre. Für den realen Einsatz müssten die Feeds bereits in der Trainingsphase geeignet abgefragt werden, wobei keine Einträge verpasst werden dürfen. Hierfür ist ein alternativer Algorithmus nötig – in [BGR06] konnten jedoch keine Hinweise auf ein geeignetes Vorgehen gefunden werden. Es ist naheliegend, dass die im Folgenden mit IndHist erzielten Ergebnisse den Idealfall darstellen und der Algorithmus im realen Einsatz aufgrund eines schlechteren Modells auch schlechtere Vorhersagen trifft.

Für Bestimmung des Parameters θ wurden zunächst die von den Autoren gewählten Werte zwischen 0,1 und 0,7 analysiert. In beiden Modi stieg die Güte mit wachsendem θ, sodass weitere Iterationen mit variierenden Schrittweiten durchgeführt wurden. Ausgewählte Ergebnisse sind den Tabellen 4.7 und 4.8 sowie der Abbildung 4.11 zu entnehmen, die vollständige Darstellung aller Messergebnisse ist in Anhang A.2 zu finden.

Die Kurvenverläufe sind wie folgt zu interpretieren: $\widehat{APE}_F$ und $\widehat{APE}_E$ zeigen die

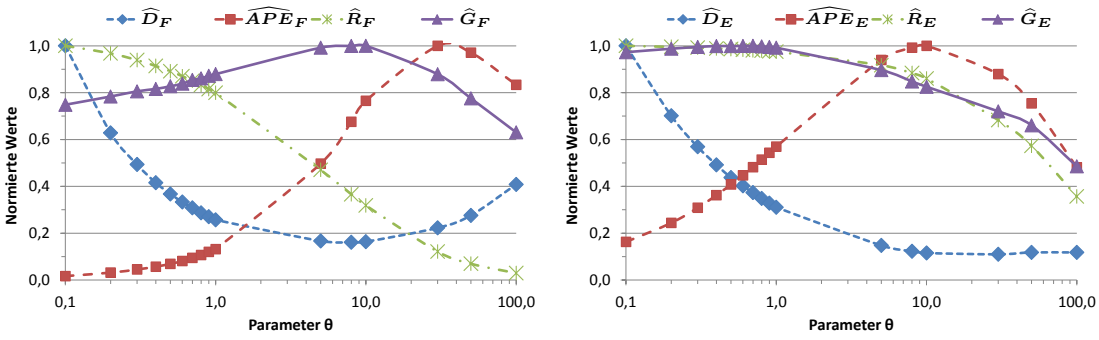

Abbildung 4.11.: Parameterbestimmung IndHist, Betrachtung der normierten Größen $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ sowie der Güte $\widehat{G}$. Vergleich des Durchschnitts über alle Feeds (links) und Einträge (rechts).

IndHist	Durchschnittswerte			auf Besten normiert			
θ	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
0,1	18min	3,284	0,979	1,000	0,163	1,000	0,9729
0,2	25min	2,191	0,974	0,702	0,244	0,996	0,9879
0,3	31min	1,731	0,971	0,569	0,309	0,992	0,9952
0,4	36min	1,475	0,968	0,492	0,362	0,989	0,9992
0,5	40min	1,309	0,965	0,437	0,408	0,986	0,9988
0,6	44min	1,195	0,962	0,402	0,447	0,983	0,9998
0,7	47min	1,108	0,960	0,373	0,482	0,981	1,0000
0,8	50min	1,038	0,958	0,348	0,514	0,979	0,9976
0,9	53min	0,983	0,956	0,328	0,544	0,977	0,9953
1,0	56min	0,938	0,954	0,311	0,570	0,975	0,9921
5,0	1h:59min	0,568	0,898	0,148	0,940	0,917	0,8964
10,0	2h:31min	0,534	0,842	0,115	1,000	0,861	0,8253
100,0	2h:29min	1,110	0,349	0,118	0,481	0,357	0,4855

Tabelle 4.8.: Parameterbestimmung IndHist, Durchschnitt über alle Einträge.

bereits bekannten Scheitelpunkte bei der Wahl sehr großer Werte für θ. Beim Delay ist dieser Scheitelpunkt ebenfalls erkennbar, im Modus Feeds ist er sogar recht deutlich ausgeprägt. Die Ursache liegt auch hier in der begrenzten Testphase, sodass beim Überschreiten eines Schwellwertes für θ ein Großteil der selten aktualisierten Feeds des Datensatzes kein zweites Mal innerhalb der Testphase abgefragt wird und sich $\overline{D}_E$ beinahe ausschließlich aus den wenigen sehr häufig aktualisierten Feeds ergibt, bei denen nur geringere Verzögerungen pro Eintrag auftreten. Im Modus Einträge ist dieser Effekt nur sehr schwach ausgeprägt. Es wird geschlussfolgert, dass der Delay eines Feeds im realen Einsatz der Algorithmen (ohne zeitliche Beschränkung der Beobachtungsdauer) bei weiterer Vergrößerung von θ gegen einen Grenzwert konvergiert, der dem Produkt aus der Änderungsrate und der halben Fenstergröße des Feeds entspricht. Dies ist der Erwartungswert des Delays für ein Fenster, das ausschließlich neue Einträge enthält. Der Grenzwert der Abfragen pro neuem Eintrag eines Feeds ist das Reziproke der Fenstergröße. Der Recall sinkt bei steigendem θ stetig und erreicht den Wert Null, wenn keiner der Feeds erneut abgefragt wird.

Im Durchschnitt über alle Einträge konnten mit $\theta = 0{,}7$ die besten Resultate erzielt werden, wobei die Werte zwischen 0,3 und 1,0 nahezu eine identische Güte ergaben. Die verringerte Anzahl Abfragen wurde vom ansteigenden Delay ausgeglichen. Im Modus Feeds stellte sich $\theta = 10{,}0$ als am Besten geeignet heraus. Im jeweils anderen Modus wurden mit diesen Werten rund 15–20 Prozent schlechtere Ergebnisse erzielt, sodass auch hier beide Werte für den Vergleich aller Algorithmen vorgeschlagen werden.

Abschließend wurde der kombinierte Algorithmus IndHist/TTL untersucht. AdaptiveTTL wies im Modus Feeds für $M = 3{,}0$ die höchste Güte auf, im Modus Einträge erzielte $M = 0{,}1$ die besten Resultate. Die beiden Werte wurden mit den besten Werten von IndHist kombiniert, sodass sich $\text{IndHist/TTL}_{10{,}0/3{,}0}$ ($\theta = 10{,}0$; $M = 3{,}0$) und $\text{IndHist/TTL}_{0{,}7/0{,}1}$ ($\theta = 0{,}7$; $M = 0{,}1$) ergaben. Diese beiden Varianten wurden mit den von den Autoren in [BGR06] vorgeschlagenen Fenstern $V = 1$ und $V = 24$ Stunden evaluiert, wobei der ebenfalls von den Autoren gewählte Wert $T_{burst} = 2$ übernommen wurde.

In beiden Modi ergaben sich bei der Wahl des kürzeren Fensters marginal bessere Werte, der Unterschied betrug jedoch nur 1–2 %, sodass geschlussfolgert wird, dass entweder die Erkennung von bursts ungeeignet ist oder der Datensatz nur sehr wenige bursts enthält. Die für $V = 1$ erzielten Werte werden in Abschnitt 4.4.2 präsentiert.

LIHZ

Der letzte separat zu betrachtende Algorithmus ist LIHZ. Im Unterschied zu den bislang evaluierten Algorithmen erfolgt die Vorhersage in LIHZ sehr grobgranular. In der Trainingsphase werden alle Algorithmen exakt alle 24 Stunden abgefragt, in der anschließenden Testphase entspricht das Abfrageintervall dem Vielfachen von einem Tag. Die Evaluationsergebnisse werden in den Tabellen 4.9 und 4.10 wie gewohnt verkürzt dargestellt, die vollständige Darstellung befindet sich in Anhang A.3.

Abbildung 4.12 zeigt das sehr auffällige Verhaltensmuster von LIHZ. Im Modus Feeds weisen Delay, APE, Recall und somit die Güte für $\sigma < 0{,}6$ nahezu konstante Werte auf. Während $\widehat{APE}_F$ mit steigendem σ beinahe linear steigt, beginnen $\widehat{D}_F$ und $\widehat{R}_F$ ab diesem Wert zu schwingen. Aufgrund des recht einfachen Modells des Algorithmus, in dem Änderungen der Feeds nur binär gespeichert werden (statt die Anzahl neuer Einträge zu betrachten), ist der Einfluss der zeitlich beschränkten Testphase bereits bei diesen vergleichsweise kleinen Werten für σ erkennbar. Die höchste Güte $\widehat{G}_F$ wird bei $\sigma = 1{,}0$ erzielt. Dies bedeutet, dass Feeds genau dann erneut abgefragt werden, wenn laut trainiertem Modell genau eine Änderung zu erwarten ist. Insgesamt zeigt LIHZ im Modus Feeds für $0{,}6 \leq \sigma \leq 2{,}5$ eine vergleichbare Güte.

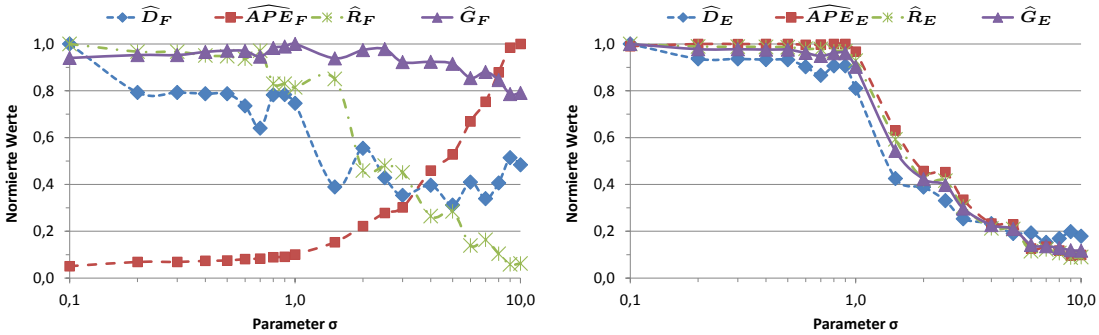

Abbildung 4.12.: Parameterbestimmung LIHZ, Betrachtung der normierten Größen $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ sowie der Güte $\widehat{G}$. Vergleich des Durchschnitts über alle Feeds (links) und Einträge (rechts).

Im Modus Feeds verlaufen alle vier Kurven annähernd gleichförmig. Für $\sigma \leq 0{,}9$ sind die Werte stabil, einzig der Delay nimmt geringfügig zu. Die höchste Güte wird für $\sigma = 0{,}1$ erzielt. Im Bereich $1{,}0 \leq \sigma \leq 4{,}0$ brechen die normierten Werte ein, wobei eine Verdopplung von σ einer Verdopplung von $\overline{D}_E$ und $\overline{APE}_E$ sowie einer Reduktion von $\overline{R}_E$ um die Hälfte entsprechen. Der Verlauf von $\widehat{APE}_E$ erscheint auf den ersten Blick nicht auffällig, beim Vergleich mit den vorangegangenen Evaluationen muss jedoch festgestellt

LIHZ	Durchschnittswerte			auf Besten normiert			
σ	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
0,1	32h:00min	1,729	0,835	1,000	0,051	1,000	0,940
0,6	43h:30min	1,085	0,782	0,736	0,081	0,937	0,970
0,7	49h:57min	1,057	0,811	0,641	0,083	0,971	0,946
0,8	40h:53min	0,983	0,693	0,783	0,089	0,830	0,983
0,9	40h:51min	0,966	0,693	0,783	0,091	0,830	0,989
1,0	42h:53min	0,874	0,680	0,746	0,101	0,815	1,000
1,5	82h:16min	0,575	0,710	0,389	0,153	0,850	0,939
2,0	57h:43min	0,396	0,384	0,555	0,222	0,459	0,974
2,5	74h:40min	0,316	0,402	0,429	0,278	0,481	0,978
3,0	90h:37min	0,291	0,377	0,353	0,302	0,451	0,923
10,0	66h:10min	0,088	0,052	0,484	1,000	0,063	0,791

Tabelle 4.9.: Parameterbestimmung LIHZ, Durchschnitt über alle Feeds.

LIHZ	Durchschnittswerte			auf Besten normiert			
σ	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
0,1	12h:04min	2,594	0,163	1,000	0,991	1,000	1,000
0,2	12h:53min	2,570	0,161	0,936	1,000	0,989	0,978
0,3	12h:53min	2,570	0,161	0,936	1,000	0,989	0,978
0,4	12h:56min	2,570	0,161	0,933	1,000	0,987	0,976
0,5	12h:56min	2,569	0,161	0,932	1,000	0,986	0,975
0,6	13h:22min	2,578	0,160	0,902	0,997	0,979	0,962
0,7	13h:56min	2,575	0,160	0,866	0,998	0,979	0,949
0,8	13h:19min	2,569	0,158	0,906	1,000	0,969	0,961
0,9	13h:18min	2,570	0,158	0,907	1,000	0,969	0,961
1,0	14h:53min	2,657	0,152	0,811	0,967	0,929	0,903
2,0	31h:04min	5,615	0,069	0,388	0,458	0,422	0,423
4,0	51h:53min	11,028	0,035	0,233	0,233	0,212	0,226
6,0	62h:36min	20,286	0,019	0,193	0,127	0,115	0,142
10,0	67h:23min	25,946	0,015	0,179	0,099	0,090	0,117

Tabelle 4.10.: Parameterbestimmung LIHZ, Durchschnitt über alle Einträge.

werden, dass $\widehat{APE}_E$ nicht wie bisher beobachtet mit einer sinkenden (durchschnittlichen) Anzahl Abfragen steigt, sondern sinkt. Dieser Effekt wird vom Standardintervall η verursacht, dessen Einfluss um so größer ist, je seltener alle übrigen Feeds abgefragt werden. Betrachtet man jedoch die bereits sehr schlechten Werte $\overline{D}_F$ und $\overline{R}_F$, so kann der Effekt vernachlässigt werden: Selbst wenn $\overline{APE}_F$ bei entsprechendem σ halbiert statt verdoppelt würde, so hätte dies aufgrund der anderen beiden Metriken Delay und Recall dennoch ein Sinken der Güte zur Folge.

In dem im Anschluss folgenden Vergleich aller Algorithmen werden die jeweiligen Besten – $\mathsf{LIHZ}_{0,1}$ und $\mathsf{LIHZ}_{1,0}$ – übernommen.

4.4.2. Vergleich aller Algorithmen

Mit der Bestimmung der Parameter im vorangegangenen Abschnitt 4.4.1 wurde die letzte Voraussetzung erfüllt, um alle Algorithmen anhand Delay, APE, Recall und Güte zu vergleichen. Hierfür werden die Modi Feeds und Einträge zunächst separat und anschließend gemeinsam betrachtet.

Durchschnitt über alle Feeds

Die Evaluationsergebnisse sind in Tabelle 4.11 und Abbildung 4.13 dargestellt. Die Reihenfolge der Algorithmen entspricht der ihrer Einführung in Abschnitt 4.2. Da die normierten Werte dieser Evaluation nicht von einem auf der Abszisse abgetragenen Parameter abhängen, wurde eine alternative Darstellung der separat zu betrachtenden Algorithmen gewählt.

Bei der Betrachtung von $\overline{D}_F$ und $\overline{APE}_F$ sowie den resultierenden normierten Werten $\widehat{D}_F$ und $\widehat{APE}_F$ fällt auf, dass diese sehr stark divergieren. Fix1h weist mit großem Abstand den geringsten Delay auf, der nächstbeste Algorithmus Fix1d erzielt einen um den Faktor 25 schlechteren Wert von 12 Stunden. Der Maximalwert liegt bei knapp drei Tagen und ist um den Faktor 150 schlechter als der beste Wert. Der dominierende Wert ergibt sich aus dem festen Intervall von einer Stunde, mit dem Fix1h jeden Feed abfragt. $\overline{D}_F$ entspricht daher rund dem Erwartungswert. Selbiges gilt für Fix1d. Fix7d hingegen weist einen Delay auf, der unterhalb des Erwartungswertes von 84 Stunden liegt. Die Ursache liegt in der durchschnittlichen Änderungsrate der Feeds des Datensatzes. Hat ein Feed z. B. eine Fenstergröße von Zehn und eine durchschnittliche Änderungsrate von 2,4 Stunden, werden die Einträge nach einem Tag durch neuere ersetzt. Für die Berechnung des Delays ergibt sich daher kein Unterschied, ob der Feed täglich oder wöchentlich abgefragt wird: Der Delay wird nicht für verpasste Einträge berechnet, der Erwartungswert beträgt bei zufälliger Wahl des Zeitpunktes der Abfrage für jeden Eintrag 12 Stunden.

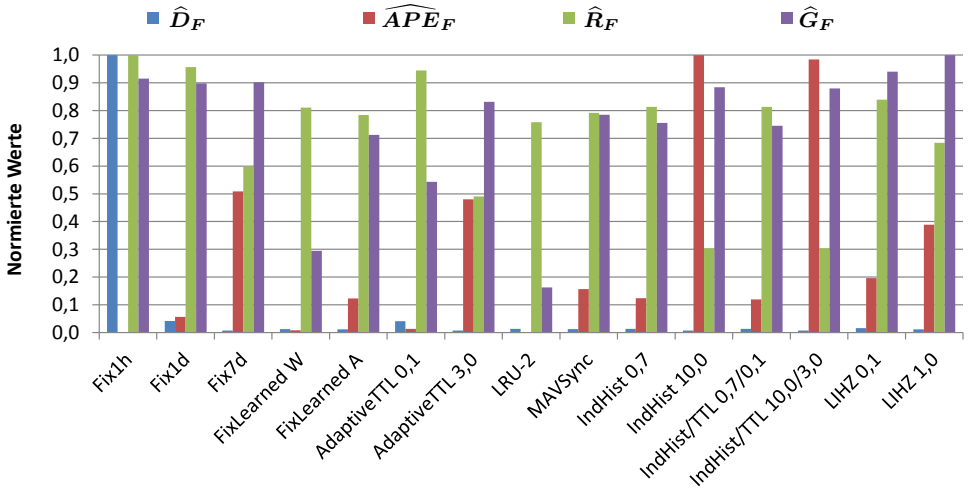

Abbildung 4.13.: Vergleich der Güte aller Algorithmen im Modus Feeds.

Algorithmus	Durchschnittswerte			auf Besten normiert			
	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
Fix1h	31min	141,334	0,995	1,000	0,002	1,000	0,915
Fix1d	12h:05min	5,998	0,952	0,042	0,057	0,956	0,898
Fix7d	66h:56min	0,667	0,595	0,008	0,509	0,598	0,902
FixLearned$_W$	40h:41min	42,882	0,807	0,012	0,008	0,811	0,294
FixLearned$_A$	43h:14min	2,753	0,780	0,012	0,123	0,784	0,712
AdaptiveTTL$_{0,1}$	12h:24min	26,061	0,940	0,041	0,013	0,945	0,543
AdaptiveTTL$_{3,0}$	66h:10min	0,707	0,488	0,008	0,480	0,491	0,831
LRU-2	37h:24min	258,870	0,754	0,014	0,001	0,758	0,162
MAVSync	41h:21min	2,173	0,788	0,012	0,156	0,792	0,785
IndHist$_{0,7}$	37h:47min	2,736	0,809	0,013	0,124	0,813	0,755
IndHist$_{10,0}$	71h:07min	0,339	0,303	0,007	1,000	0,304	0,884
IndHist/TTL$_{0,7/0,1}$	37h:47min	2,848	0,809	0,013	0,119	0,813	0,745
IndHist/TTL$_{10,0/3,0}$	70h:59min	0,345	0,303	0,007	0,984	0,304	0,880
LIHZ$_{0,1}$	32h:00min	1,729	0,835	0,016	0,196	0,839	0,940
LIHZ$_{1,0}$	42h:53min	0,874	0,680	0,012	0,388	0,684	1,000

Tabelle 4.11.: Vergleich der Güte aller Algorithmen im Modus Feeds.

Die Streuung der Werte fällt bei $\overline{APE}_F$ sogar noch stärker aus. Der hohe Wert von Fix1h ist der „Preis“ für den sehr geringen Delay und den höchsten Recall im aktuellen Vergleich. Der im Vergleich zu Fix1h knapp doppelt so hohe Wert von LRU-2 überrascht jedoch, vor allem in Anbetracht des hohen Delays und geringeren Recalls. Eine Analyse ergab, dass LRU-2 sehr schlecht zur Vorhersage geeignet ist, wenn die Abstände zwischen den Einträgen eines Feeds stark variieren. Werden in einem Feed zwei Einträge innerhalb weniger Minuten veröffentlicht, so verwendet LRU-2 dieses Intervall im Anschluss. Folgt nun über einen langen Zeitraum kein neuer Eintrag, wird der Feed dennoch mit einer hohen Frequenz abgefragt. Im umgekehrten Fall ist der Effekt ebenso von Nachteil. Bei Feeds mit Tag- und Nachtzyklen kann es somit vorkommen, dass der Feed nachts in kurzen Intervallen abgefragt wird, ohne neue Einträge zu erlangen. Beim ersten neuen Eintrag am Morgen wird das sich aus der Nacht ergebende Intervall von z. B. 10 Stunden gelernt und der Feed somit erst abends erneut abgefragt. Es resultiert ein hoher Delay, unter Umständen werden zudem einige Einträge verpasst.

Der Vergleich von FixLearned$_W$ und FixLearned$_A$ zeigt, dass beide einen ähnlichen Delay haben, sich jedoch bei vergleichbarem Recall drastisch in der Anzahl benötigter Abfragen unterschieden. Die Ursache ist, dass FixLearned$_A$ aufgrund der Einbeziehung des Zeitpunktes der ersten Abfrage Zombies viel besser behandeln kann als FixLearned$_W$. FixLearned$_A$ ist daher deutlich besser geeignet als FixLearned$_W$ und zudem breiter einsetzbar, da auch für SingleItem und Chunked Feeds ein Intervall berechnet werden kann. Dies ist bei FixLearned$_W$ nicht möglich, da der Algorithmus mindestens zwei verschiedene Zeitstempel von Einträgen zur Ermittlung des Intervalls benötigt.

Im Modus Feeds stellt sich im Vergleich aller Algorithmen anhand der Güte LIHZ$_{1,0}$ als am Besten geeignet heraus. Auf den nächsten Plätzen folgen mit rund 10 % Abstand die Algorithmen Fix1h, Fix7d, Fix1d sowie IndHist$_{10,0}$, die jeweils eine vergleichbare Güte erreichen. Der eigene Algorithmus MAVSync schafft es zunächst nur ins Mittelfeld und

liegt reichlich 20 % hinter dem Besten zurück. Er erzielt dennoch bessere Ergebnisse als FixLearned$_A$, auf dem MAVSync basiert. IndHist/TTL$_{10,0/3,0}$ erzielt marginal schlechtere Ergebnisse als IndHist$_{10,0}$, ist jedoch um rund fünf Prozent besser als AdaptiveTTL$_{3,0}$. Dieses Resultat unterstreicht die in Abschnitt 4.4.1 getätigte Aussage, dass die von den Autoren in [BGR06] gewählte Erkennung von bursts ungeeignet ist oder der Datensatz nur über sehr wenige bursts verfügt.

Zusammenfassend wird festgestellt, dass die sehr einfach umzusetzenden Algorithmen Fix1h, Fix1d, Fix7d recht gut zur Abfrage von Feeds geeignet sind, wenn diese gleich gewichtet werden. Mit der Verwendung eines einfachen Poisson-Modells der Änderungsrate in Form des Algorithmus LIHZ$_{1,0}$ konnten die Ergebnisse noch verbessert werden. Dieses zunächst ein wenig enttäuschende Resultat von MAVSync ist jedoch ohne die Betrachtung aller Algorithmen im Modus Einträge nicht aussagekräftig, wie im Folgenden sowie bei dem Vergleich beider Modi gezeigt wird.

Durchschnitt über alle Einträge

Die zweite Betrachtung aller Algorithmen erfolgt aus dem Blickwinkel des Modus Einträge. Die in Abbildung 4.14 und Tabelle 4.12 präsentierten Evaluationsergebnisse werden wie folgt interpretiert: AdaptiveTTL$_{0,1}$ weist mit 17 Minuten den geringsten durchschnittlichen Delay $\overline{D}_E$ (und somit den besten Wert $\widehat{D}_E$) auf, der mit Abstand schlechteste Wert von reichlich zwei Tagen wird von Fix7d erzielt. AdaptiveTTL$_{0,1}$ erreicht diesen sehr guten Wert durch eine vergleichsweise hohe Anzahl Abfragen pro neuem Eintrag $\overline{APE}_E$. Den zweitbesten Wert für $\widehat{D}_E$ erzielt Fix1h, gefolgt von MAVSync, FixLearned$_A$, FixLearned$_W$ und IndHist$_{0,7}$ auf den Plätzen 3–6. Der sehr geringe $\overline{APE}_E$-Wert von Fix7d führt diesen zwar in dieser Bewertung an die Spitze, resultiert jedoch im geringsten Recall und der geringsten Güte aller betrachteter Algorithmen.

Im Vergleich des Recalls erzielen FixLearned$_A$, MAVSync und IndHist$_{0,7}$ annähernd

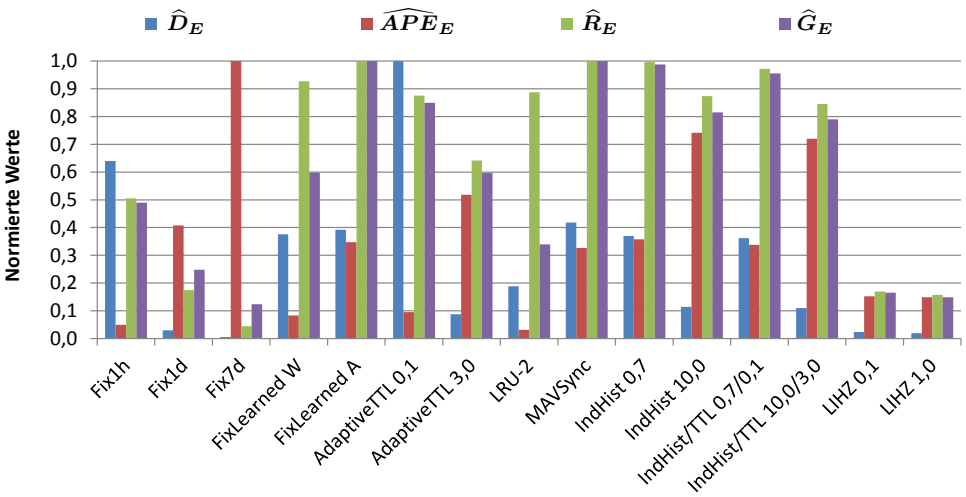

Abbildung 4.14.: Vergleich der Güte aller Algorithmen im Modus Einträge.

Algorithmus	Durchschnittswerte			auf Besten normiert			
	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
Fix1h	27min	7,984	0,487	0,640	0,050	0,505	0,490
Fix1d	9h:47min	0,972	0,168	0,029	0,408	0,175	0,248
Fix7d	49h:38min	0,396	0,043	0,006	1,000	0,045	0,124
FixLearned$_W$	46min	4,739	0,893	0,376	0,084	0,927	0,597
FixLearned$_A$	44min	1,141	0,964	0,392	0,347	1,000	0,999
AdaptiveTTL$_{0,1}$	17min	4,147	0,844	1,000	0,095	0,875	0,849
AdaptiveTTL$_{3,0}$	3h:18min	0,764	0,618	0,087	0,518	0,641	0,597
LRU-2	1h:31min	12,414	0,855	0,189	0,032	0,887	0,340
MAVSync	42min	1,212	0,963	0,418	0,327	0,999	1,000
IndHist$_{0,7}$	47min	1,108	0,960	0,369	0,357	0,996	0,988
IndHist$_{10,0}$	2h:31min	0,534	0,842	0,114	0,741	0,874	0,815
IndHist/TTL$_{0,7/0,1}$	48min	1,172	0,937	0,362	0,338	0,972	0,955
IndHist/TTL$_{10,0/3,0}$	2h:36min	0,550	0,815	0,110	0,720	0,845	0,790
LIHZ$_{0,1}$	12h:04min	2,594	0,163	0,024	0,153	0,169	0,165
LIHZ$_{1,0}$	14h:53min	2,657	0,152	0,019	0,149	0,157	0,149

Tabelle 4.12.: Vergleich der Güte aller Algorithmen im Modus Einträge.

gleiche Werte und führen diese Bewertung an. Insgesamt können die Algorithmen anhand des Recalls grob in zwei Klassen unterteilt werden. In der Ersten befinden sich alle Algorithmen, welche das zur Abfrage gewählte Intervall frei anhand der Feeds berechnen können. Die zweite Klasse enthält die übrigen Algorithmen mit fixen statischen Intervallen, sowie LIHZ$_{0,1}$ und LIHZ$_{1,0}$, die nur Vielfache eines Tages als Intervall wählen. Wie bereits in Abschnitt 4.3.5 eingeführt, wird der Modus Einträge von Feeds mit sehr hohen Änderungsraten dominiert. Für diese Feeds ist ein Intervall von einem oder mehreren Tagen sehr schlecht geeignet und führt zu einer hohen Zahl an Misses, wie die Ergebnisse bestätigen. Fix7d ist daher am schlechtesten geeignet: Nur rund 4 % aller Einträge werden gefunden. Bei einem fixen Intervall von einer Stunde wird dennoch die Hälfte aller Einträge verpasst, woraus geschlussfolgert wird, dass alle Feeds in noch kürzeren Intervallen abzufragen wären, um einen den adaptiven Algorithmen gleichwertigen Recall zu erzielen. Dies hätte jedoch den Nachteil, dass sehr selten aktualisierte Feeds ebenfalls unnötig häufiger abgefragt würden.

Im Mittel – der Güte $\widehat{G}_E$ – ist MAVSync mit sehr geringem Vorsprung der beste Algorithmus, dicht gefolgt von FixLearned$_A$ und IndHist$_{0,7}$. IndHist/TTL$_{0,7/0,1}$ reiht sich zwischen den beiden ihm zugrunde liegenden Algorithmen IndHist$_{0,7}$ und AdaptiveTTL$_{0,1}$ ein. Die selten abfragenden Algorithmen Fix1d und Fix7d sowie LIHZ$_{0,1}$ und LIHZ$_{1,0}$ sind im Modus Einträge am schlechtesten geeignet. Dieser Unterschied basiert unter anderem auf dem teilweise sehr geringen Recall $\overline{R}_E$, den diese Algorithmen im Modus Einträge aufweisen. Beim Vergleich der Werte $\overline{R}_F$ und $\overline{R}_E$ von Fix7d fällt auf, dass nicht 60 % der Einträge gefundenen werden, sondern nur 4 %. Dies zeigt, dass bei der Auswertung und Interpretation der Ergebnisse immer das Bezugssystem, d. h. der Modus beachtet werden muss. Wie am Beispiel IndHist$_{10,0}$ zu erkennen ist, kann auch $\overline{R}_F \ll \overline{R}_E$ gelten: Der Algorithmus weist für die Mehrzahl der Feeds einen schlechten Recall auf, obwohl insgesamt rund 84 % der Einträge gefunden werden.

Es wird das Resümee gezogen, dass die Betrachtung der Einträge eines Fensters und des Zeitpunktes der ersten bzw. aktuellen Abfrage bereits genügend Informationen liefern, um die Änderungsrate eines Feeds zu bestimmen und den Zeitpunkt der nächsten Änderung vorherzusagen. Das von $\mathsf{IndHist}_{0,7}$ trainierte Modell führt zu keinen besseren Ergebnissen. Aus den nahezu identischen Resultaten von MAVSync und $\mathsf{FixLearned}_A$ wird geschlossen, dass sich die Änderungsraten der Feeds in der Testphase von drei Wochen kaum geändert haben oder die Änderungen im Durchschnitt ausgeglichen wurden. Es wird die Vermutung aufgestellt, dass der Vorteil von MAVSync gegenüber $\mathsf{FixLearned}_A$, adaptiv auf Änderungen zu reagieren und z. B. nicht mehr aktualisierte Feeds immer seltener abzufragen, nur über einen langen Beobachtungszeitraum von mehreren Monaten ersichtlich wird. Ein solcher Datensatz konnte jedoch mit den zur Verfügung stehenden Ressourcen nicht erstellt werden.

Mit dem Resümee wird die in Abschnitt 1.5 aufgestellte These bezüglich der Änderungshistorie für den Modus Einträge bestätigt: Die in einem Fenster enthaltene partielle Historie eines Feeds enthält genügend Informationen, um eine Vorhersage zu tätigen, deren Qualität äquivalent zu aufwendigeren Verfahren ist, in denen vorab ein Modell trainiert werden muss. Für eine Generalisierung der Aussage sind die Resultate beider Modi gegenüberzustellen.

Betrachtung beider Modi

Aus dem Vergleich der Evaluationsergebnisse beider Modi ergibt sich, dass die im Modus Feeds am besten abschneidenden Algorithmen $\mathsf{LIHZ}_{1,0}$ und $\mathsf{LIHZ}_{0,1}$, sowie Fix1h, Fix1d und Fix7d im Modus Einträge die schlechtesten Resultate erzielen. $\mathsf{FixLearned}_A$ ist in beiden Modi deutlich besser als $\mathsf{FixLearned}_W$, der Vorsprung beruht jeweils auf der geringeren Anzahl erfolgter Abfragen. MAVSync, $\mathsf{FixLearned}_A$ und $\mathsf{IndHist}_{0,7}$ erzielen im Modus Einträge eine äquivalente Güte, im Modus Feeds ist der eigene Algorithmus diesen beiden jedoch überlegen. Im Modus Feeds erzielen $\mathsf{AdaptiveTTL}_{3,0}$ und $\mathsf{IndHist}_{10,0}$ eine rund 6 % bzw. 13 % höhere Güte als MAVSync, büßen diesen Vorsprung jedoch im Modus Einträge komplett ein, sodass MAVSync hier rund 63 % bzw. 23 % besser ist, als diese beiden. Die in Abschnitt 5.3.2 berechnete durchschnittliche Güte über beide Modi wird dies belegen.

Es ergibt sich folgendes Gesamtbild: 1.) MAVSync erzielt bei der Bildung der Durchschnittswerte über alle Einträge mit marginalem Vorsprung die höchste Güte. Die im Modus Einträge ebenbürtigen Algorithmen scheiden im Modus Feeds schlechter als MAVSync ab. 2.) Die im Modus Feeds besseren Algorithmen erreichen dies nur durch eine aufwendige, vorgelagerte Optimierung entsprechender Parameter wie M, θ und σ oder die Wahl eines fix vorgegebenen Intervalls. Werden diese beiden Aspekte berücksichtigt, so rückt MAVSync auch in der Gesamtwertung mit an die Spitze, sodass die bezüglich der Änderungshistorie aufgestellte These als erfüllt gilt. Die vergleichsweise einfache Berechnung von MAVSync erzielt eine den aufwendigeren Berechnungen des Algorithmus IndHist äquivalente Güte und im Mittel sogar bessere Resultate als LIHZ.

4.4.3. Übertragungsvolumen

Neben den im vorangegangenen Abschnitt betrachteten Metriken vermittelt die Betrachtung des Übertragungsvolumens ein klassisches Beispiel des Ressourcenbedarfs der Algorithmen. Das Übertragungsvolumen ist unabhängig von den Modi Feeds oder Einträge. Abbildung 4.15 zeigt das kumulierte Übertragungsvolumen der Algorithmen für die Testphase von 20 Tagen. Zur besseren Unterscheidbarkeit der Algorithmen im Mittelfeld wurde die Darstellung auf zwei Terabyte (TB) begrenzt. Die in der Legende enthaltene Reihenfolge entspricht der des Übertragungsvolumens der Algorithmen.

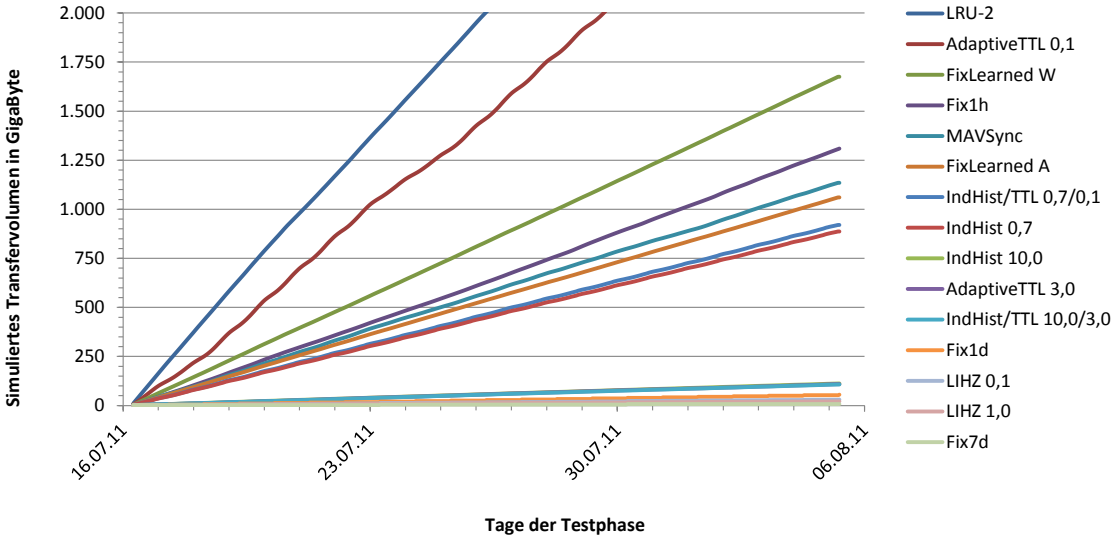

Abbildung 4.15.: Simuliertes Übertragungsvolumen der Algorithmen innerhalb der Testphase vom 16.07.2011 07:00 bis 05.08.2011 07:00.

Anhand der Abbildung ist zu erkennen, dass sich drei Cluster von Algorithmen ergeben. Wie bereits aus den in Abschnitt 4.4.2 gewonnenen Erkenntnissen wie der Anzahl Abfragen pro neuem Eintrag erwartet, bedingt LRU-2 das mit Abstand größte Übertragungsvolumen und erreicht innerhalb von 20 Tagen einen beachtlichen Wert von rund 4 TB. Dass zwischen $\overline{APE}_E$ und Übertragungsvolumen jedoch kein direkter Zusammenhang besteht, zeigt der Unterschied zwischen $\mathsf{AdaptiveTTL}_{0,1}$ und Fix1h: Fix1h führt rund doppelt so viele Abfragen pro neuem Eintrag aus wie $\mathsf{AdaptiveTTL}_{0,1}$, das resultierende Übertragungsvolumen beträgt rund 1,3 TB und somit nur knapp die Hälfte des Werts von $\mathsf{AdaptiveTTL}_{0,1}$, der bei 3 TB liegt. Dies hat zwei Ursachen. Das Übertragungsvolumen eines Feeds ergibt sich aus dem Produkt seiner Größe und der Anzahl getätigter Abfragen, sodass bei gleicher Anzahl Abfragen zweier Feeds komplett unterschiedliche Volumen resultieren können, wenn diese unterschiedliche Größen aufweisen. Dies kann sowohl aus verschiedenen Fenstergrößen als auch verschiedenen Längen der in den Einträgen enthaltenen Beschreibungen resultieren. Die zweite Ursache ist, dass $\overline{APE}_E$ nicht der Anzahl getätigter Abfragen, sondern der Anzahl getätigter Abfragen *pro neuem Eintrag* entspricht. Hat ein Feed A die doppelte Änderungsrate wie ein Feed B und wird A

doppelt so häufig abgefragt und jeweils bei jeder Abfrage ein neuer Eintrag gefunden, so resultieren für beide Feeds die gleichen $\overline{APE}_E$-Werte.

Insgesamt zeigt die Abbildung, dass die im Modus Feeds recht gut abschneidenden Algorithmen ein relativ geringes Übertragungsvolumen bedingen. Das mit 7,5 GB geringste Übertragungsvolumen ist Fix7d zuzuordnen, IndHist/$\text{TTL}_{10,0/3,0}$, $\text{AdaptiveTTL}_{3,0}$ und $\text{IndHist}_{10,0}$ erzielen beinahe identische Volumen zwischen 107 und 112 GB. Die besten Algorithmen im Modus Einträge sind ebenfalls recht nahe beieinander, ihre Werte differieren um 20 %.

Abschließend sei der auffällige Verlauf von $\text{AdaptiveTTL}_{0,1}$ analysiert. Die Variationen des Anstiegs sind auf unterschiedliche Aktivitäten der Feeds zu Tag- und Nachtzeiten zurückzuführen. Dieses Verhalten tritt auch bei anderen Algorithmen wie MAVSync und IndHist auf, was jedoch in der Darstellung nur schwer zu erkennen ist. Neben den Tag- und Nachtzyklen weist $\text{AdaptiveTTL}_{0,1}$ zudem wöchentliche Zyklen auf, die bei genauer Betrachtung zu erkennen sind: An den ersten beiden Tagen der Testphase ist der Anstieg geringer als an den darauffolgenden fünf Tagen. Dieses Muster wiederholt sich ab dem 23.07.2011. Der Start der Testphase erfolgte an einem Samstag, sodass der Schluss gezogen wird, dass Feeds im Durchschnitt an den Wochenenden weniger Einträge aufweisen, als an den Tagen Montag bis Freitag.

4.5. Zusammenfassung

Die in Kapitel 2 untersuchten Algorithmen wurden in diesem Kapitel aufgegriffen, an die Domäne Feeds angepasst, um den eigenen Algorithmus MAVSync erweitert und anschließend miteinander verglichen. Hierfür wurden zunächst die drei Kriterien Delay, APE und Recall aufgestellt und aus diesen anschließend ein Gütekriterium hergeleitet. In der Evaluation wurde zwischen den zwei Modi Feeds und Einträge unterschieden, um einerseits alle Feeds und andererseits alle Einträge mit dem gleichen Gewicht in die Resultate eingehen zu lassen.

Die Ergebnisse zeigten, dass zwischen den beiden Modi beachtliche Unterschiede in der Güte der Algorithmen auftreten. Die im Modus Feeds am besten abschneidenden Algorithmen wie Fix1d, Fix7d und $\text{LIHZ}_{1,0}$ weisen die Gemeinsamkeit auf, Feeds nur täglich oder seltener abzufragen und somit im Modus Einträge schlechte Ergebnisse zu erzielen. Aus diesem Grund wird in Kapitel 5 untersucht, inwieweit ausgewählte adaptive Algorithmen mit den in Abschnitt 4.2.1 eingeführten Schranken verbessert werden können, um im Modus Feeds bessere Ergebnisse zu erzielen, ohne die Güte im Modus Einträge negativ zu beeinträchtigen.

Eine weitere sich aus den Evaluationsergebnissen ergebende Idee ist eine alternative, gewichtete Berechnung der Güte. Ziel soll es sein, Anwendern mit der Güte ein Werkzeug zur Hand zu geben, mit dem sie die Algorithmen nach ihren Bedürfnissen bewerten können und auf diese Weise zum Beispiel den Delay oder Recall höher zu gewichten, wenn sie an einer zeitnahen Erkennung neuer Einträge interessiert sind oder möglichst wenige Einträge verpassen möchten.

5. Verfeinerung der Vorhersage

Einige ausgewählte Algorithmen werden in diesem Kapitel vertieft betrachtet und anschließend der Bogen zur Generalisierung der Betrachtungen geschlagen, um die Anwendbarkeit sowie Erweiterungsmöglichkeiten zu diskutieren.

Zuerst werden die Schranken α und β variiert und die Auswirkungen auf einige Algorithmen untersucht. Im Anschluss erfolgt die Auswertung feedspezifischer Attribute zur Vorhersage von Änderungen.

Auf diese beiden Verfeinerungen folgt in 5.3 eine generalisierte Betrachtung der Güte, wobei einzelne Metriken höher gewichtet werden und die Arbeit im Kontext der Anwendbarkeit diskutiert wird. Den Abschluss bildet die Betrachtung von Erweiterungsmöglichkeiten in Abschnitt 5.4.

5.1. Variation der Schranken des Abfrageintervalls

Die untere Schranke α und die obere Schranke β wurden bereits zur Erstellung des Datensatzes in Abschnitt 3.5.4 eingeführt und in Abschnitt 4.2.1 auf die zu evaluierenden Algorithmen übertragen. Um den Einfluss der Schranken zunächst zu minimieren und das „natürliche“ Verhalten aller Algorithmen zu vergleichen, wurde in den Evaluationen des vorangegangenen Kapitels die untere Schranke α auf einen Wert von einer Minute gesetzt und keine Beschränkung[1] des Intervalls durch eine obere Schranke β vorgenommen.

Im Folgenden werden die Auswirkungen beider Schranken untersucht. Die Untersuchung erfolgt getrennt für die Modi Feeds und Einträge anhand der Algorithmen, die im Vergleich aller Algorithmen in Abschnitt 4.4.2 die besten Resultate aufwiesen.

Die Variationen der Schranken erfolgt in allen Evaluationen gleich. Für α werden die Werte 1, 5 und 15 Minuten sowie eine Stunde gewählt, seltenere Abfragen erscheinen anhand der im Datensatz analysierten Änderungsraten ungeeignet. Als obere Schranke β werden ein Tag und eine Woche untersucht. Um die Schranken unabhängig von einander zu analysieren, erfolgen zudem Evaluationen ohne den Einfluss der oberen Schranke. Dies sei als $\beta = \infty$ dargestellt. Mit den vier unteren und drei oberen Schranken ergeben sich pro betrachtetem Algorithmus zwölf Evaluationen.

5.1.1. Durchschnitt über alle Feeds

Im Modus Feeds erzielte LIHZ in den beiden Varianten $\mathsf{LIHZ}_{1,0}$ und $\mathsf{LIHZ}_{0,1}$ die höchste Güte, gefolgt von den Algorithmen Fix1h, Fix7d und Fix1d mit festen Intervallen sowie

[1] Es wurde $\beta = 4$ Wochen gewählt, sodass die Schranke keinen Einfluss auf die Abfrageintervalle der Trainings- und Testphase hatte.

IndHist$_{10,0}$. Zur Analyse der Auswirkungen der Schranken werden LIHZ$_{1,0}$, IndHist$_{10,0}$ sowie der eigene Algorithmus MAVSync betrachtet. Der Vergleich verschiedener Versionen von LIHZ erscheint nicht erstrebenswert; eine Beschränkung der drei Algorithmen mit fixem Intervall ist nicht möglich, da diese alle Feeds mit dem gleichen Intervall abfragen, sodass die Anwendung der Schranken eine Überführung von einem Algorithmus in den andern bedingt.

Das in Abschnitt 4.4 eingesetzte Vorgehen, zunächst die Algorithmen einzeln zu betrachten um die Parameter auszuwählen, welche die höchste Güte erzielen und anschließend die jeweiligen Besten miteinander zu vergleichen, wird aufgegriffen.

LIHZ$_{1,0}$

Die Auswirkungen der Variation von α und β auf LIHZ$_{1,0}$ sind Abbildung 5.1 zu entnehmen. Für jeweils einen Wert α wird β variiert, sodass sich vier Blöcke ergeben. Zudem erfolgt von links nach rechts eine schrittweise Einschränkung des Abfrageintervalls, wobei α zunehmend größer und β kleiner wird. Das Muster wird in allen Evaluationen dieses Abschnitts beibehalten. Die in allen Abbildungen ganz links dargestellten Werte der Kombination $\alpha = 1$ Minute und $\beta = \infty$ entsprechen somit denen in Abschnitt 4.4.2. Es sei darauf hingewiesen, dass die Normierung der Werte je Vergleich anhand des Besten erfolgt, sodass sich $\widehat{D}_F$, $\widehat{APE}_F$ und $\widehat{R}_F$ für $\alpha = 1$ Minute und $\beta = \infty$ von den normierten Werten in den Abschnitten 4.4.1 und 4.4.2 unterscheiden.

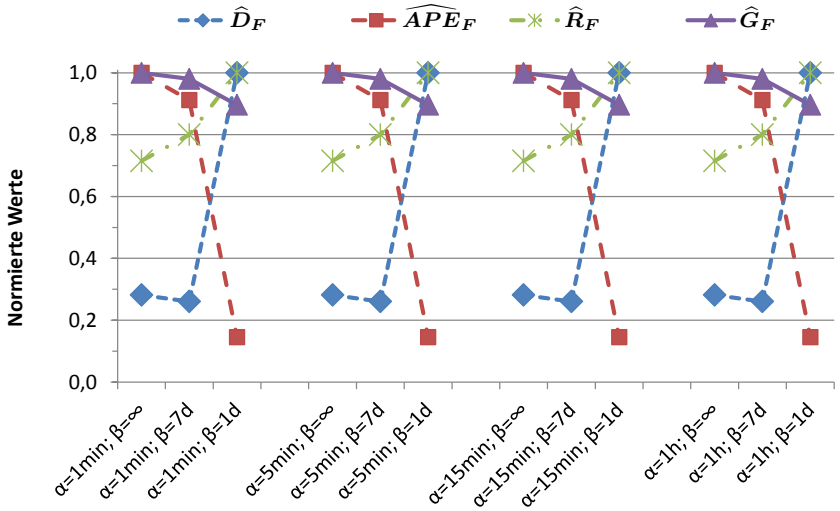

Abbildung 5.1.: LIHZ$_{1,0}$, Variation der Schranken α und β im Modus Feeds.

In Abbildung 5.1 ist zu erkennen, dass die Variation von α keinen Einfluss auf den Algorithmus hat. Aus den hier nicht tabellarisch dargestellten Werten geht hervor, dass die Werte jeweils exakt gleich sind. Dies war zu erwarten, da das von LIHZ$_{1,0}$ gewählte Abfrageintervall dem Vielfachen eines Tages entspricht und die gewählten unteren Schranken somit keinen Einfluss auf das Intervall nehmen.

Die Verläufe innerhalb eines Blocks zeigen einen Rückgang der Güte, bedingt durch die Einschränkung des Intervalls. Je kürzer das Intervall ist, desto häufiger werden die Feeds abgefragt. Somit sinkt $\widehat{APE}_F$, während $\widehat{R}_F$ und $\widehat{D}_F$ steigen. Bei $\beta = 7$ Tage ist die in Abschnitt 4.4.1 untersuchte Anomalie des Delays zu erkennen, die sich aus der Beschränkung der Testphase ergibt. Der Einfluss ist jedoch nur marginal. Durch eine weitere Einschränkung des Intervalls auf $\beta = 1$ Tag sinkt der durchschnittliche Delay und bewirkt einen Anstieg des abgebildeten normierten Delays. Im Gegenzug werden mehr Abfragen ausgeführt, was in einem schlechteren $\widehat{APE}_F$ resultiert. $\widehat{APE}_F$ sinkt im Verhältnis stärker als $\widehat{D}_F$ und $\widehat{R}_F$ steigen, sodass die Güte ebenfalls sinkt.

In der durchgeführten Evaluation wird die höchste Güte für $\mathsf{LIHZ}_{1,0}$ erzielt, wenn die Länge des Intervalls nicht durch β begrenzt wird. Die gewählten Werte der unteren Schranke beeinflussen die Ergebnisse nicht. Somit ergibt sich gegenüber den in Abschnitt 4.4.2 untersuchten Werten keine Steigerung der Güte durch den Einsatz der Schranken.

$\mathsf{IndHist}_{10,0}$

Abbildung 5.2 zeigt die Evaluationsergebnisse des Algorithmus $\mathsf{IndHist}_{10,0}$. Die untere Schranke nimmt nahezu keinen Einfluss auf die Ergebnisse im Modus Feeds, der Unterschied der normierten Werte zwischen verschiedenen β bei konstantem α beträgt weniger als 0,5 %, größere Werte von α führen zu jeweils besseren Ergebnissen.

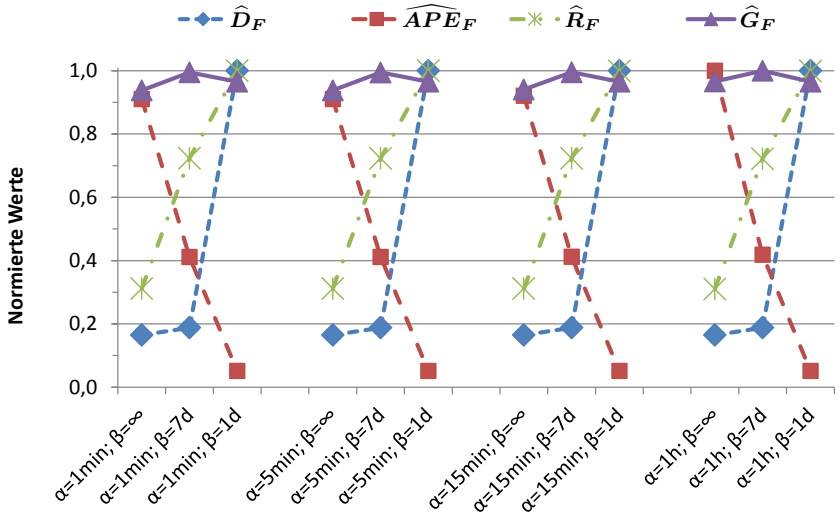

Abbildung 5.2.: $\mathsf{IndHist}_{10,0}$, Variation der Schranken α und β im Modus Feeds.

Durch die Begrenzung des Intervalls auf maximal sieben Tage kann der Recall mehr als verdoppelt werden, insgesamt ergibt sich ein leichter Anstieg der Güte um 7 %. Eine weitere Beschränkung des Intervalls bewirkt eine Reduktion des Delays in beachtlichem Maße (und folglich eine Steigerung des normierten Wertes in Abbildung 5.2). Dem gegenüber steht jedoch ein deutlich schlechterer Wert für $\widehat{APE}_F$. Dies hat einen Rückgang der Güte zur Folge.

Die Beobachtungen können folgendermaßen interpretiert werden: Für $\mathsf{IndHist}_{10,0}$ wurde der Parameter θ dahingehend optimiert, im hier betrachteten Modus Feeds die besten Resultate zu erzielen. Im Gegensatz zum Modus Einträge stellte sich der Wert $\theta = 10{,}0$ als am Geeignetsten heraus. Dies führte im Mittel zu einer geringen Anzahl von Abfragen und somit zu vergleichsweise großen Intervallen zwischen zwei Abfragen. Die obere Schranke β wirkt diesem entgegen, sodass nach einer anfänglichen Steigerung der Güte bei weiterer Reduktion des Intervalls die Güte wieder rückläufig ist. Der Idealwert von β kann sowohl zwischen einem und sieben Tagen liegen, als auch kleiner als ein Tag sein, was jedoch aus den Ergebnissen nicht zu bestimmen ist. Aufgrund der beschränkten zur Verfügung stehenden technischen Ressourcen konnten keine weiteren Schranken betrachtet werden. Die höchste Güte wird mit $\alpha = 1$ Stunde und $\beta = 7$ Tage erzielt.

MAVSync

Der dritte betrachtete Algorithmus ist MAVSync, dessen Resultate im Modus Feeds in Abbildung 5.3 dargestellt sind. Auf den ersten Blick ist das bekannte Muster zu erkennen, welches sich aus den Schranken ergibt. Der Einfluss von α ist marginal, kleinere Werte für β führen hingegen zu einer Steigerung der Güte.

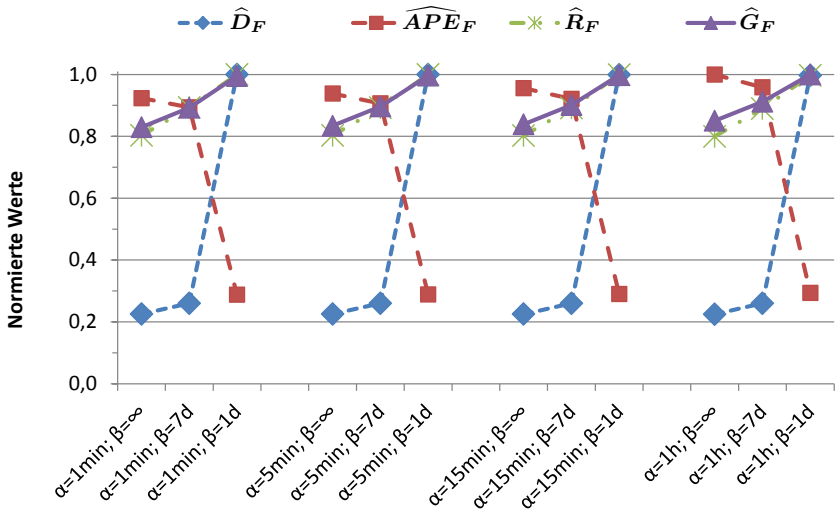

Abbildung 5.3.: MAVSync, Variation der Schranken α und β im Modus Feeds.

Anhand der Resultate ist anzunehmen, dass die Güte bei einer weiteren Reduktion der oberen Schranke weiter steigt. Hierbei muss beachtet werden, dass der durchschnittliche Recall von 0,79 bei $\beta = \infty$ auf 0,98 bei $\beta = 1$ Tag steigt. Eine weitere Steigerung ist somit kaum möglich und es ist zu vermuten, dass die höchste Güte wenige Prozent über dem ermittelten besten Wert liegt, der sich bei $\alpha = 1$ Stunde und $\beta = 1$ Tag ergibt.

Vergleich der besten Algorithmen im Modus Feeds

Die drei untersuchten Algorithmen werden in Abbildung 5.4 gegenübergestellt. Im linken Bereich sind die Ausgangswerte aus Abschnitt 4.4.2 dargestellt, gegenüber die soeben ermittelten Besten der betrachteten Schranken. Die genauen Werte sind in selbiger Reihenfolge der Tabelle 5.1 zu entnehmen.

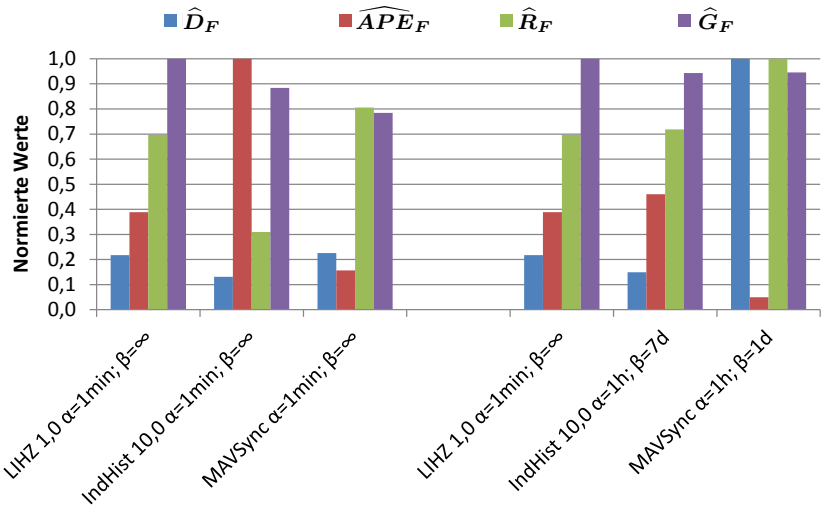

Abbildung 5.4.: Vergleich der besten Algorithmen im Modus Feeds unter Berücksichtigung der Schranken α und β.

Abs.	Algorithmus	Schranken		Durchschnittswerte			auf Besten normiert			
		α	β	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
4.4.2	$\text{LIHZ}_{1,0}$	1min	∞	42h:53min	0,874	0,680	0,218	0,388	0,696	1,000
	$\text{IndHist}_{10,0}$	1min	∞	71h:07min	0,339	0,303	0,131	1,000	0,310	0,884
	MavSync	1min	∞	41h:21min	2,173	0,788	0,226	0,156	0,806	0,785
5.1.1	$\text{LIHZ}_{1,0}$	1min	∞	42h:53min	0,874	0,680	0,218	0,388	0,696	1,000
	$\text{IndHist}_{10,0}$	1h	7d	62h:31min	0,738	0,702	0,149	0,460	0,718	0,943
	MavSync	1h	1d	9h:20min	6,822	0,978	1,000	0,050	1,000	0,945

Tabelle 5.1.: Vergleich der besten Algorithmen im Modus Feeds unter Berücksichtigung der Schranken α und β.

Es ist gut zu erkennen, dass MAVSync am meisten von der Variation der Schranken profitiert. Der durchschnittliche Delay $\overline{D}_F$ kann um mehr als den Faktor vier gesenkt und $\overline{R}_F$ beinahe auf den theoretischen Maximalwert erhöht werden, wobei $\overline{APE}_F$ nur reichlich um den Faktor drei erhöht wird. $\text{MAVSync}_{\alpha=1h;\beta=1d}$ erzielt neben dem besten Recall den mit Abstand besten Delay der hier betrachteten Algorithmen und belegt mit marginalem Vorsprung den zweiten Platz vor $\text{IndHist}_{10,0;\alpha=1h;\beta=7d}$. $\text{LIHZ}_{1,0;\alpha=1m;\beta=\infty}$ erzielt die höchste Güte des Vergleichs, der Vorsprung ist jedoch auf gut 5 % gesunken.

5.1.2. Durchschnitt über alle Einträge

Im Modus Einträge werden die drei Algorithmen MAVSync, FixLearned$_A$ und IndHist$_{0,7}$ analysiert. Diese erzielten in Abschnitt 4.4.2 die höchste Güte. Die Auswertung der Simulationen mit unterschiedlichen Schranken ergab, dass sich α und β auf alle drei Algorithmen in gleichem Maße auswirken, weshalb hier nur ein Algorithmus detailliert dargestellt wird.

MAVSync

Die Evaluationsergebnisse sind in der bekannten Weise in Abbildung 5.5 dargestellt. In der Grafik sind zwei sich aus den Schranken ergebende Muster zu erkennen. Die Variation von β bewirkt eine Reduktion des durchschnittlichen Delays und folglich einen Anstieg des normierten Delays $\widehat{D}_E$ in Abbildung 5.5. Im Gegenzug ist $\widehat{APE}_E$ leicht rückläufig, es ergibt sich ein Anstieg der Güte. Der Recall wird von β nur marginal beeinträchtigt, die Unterschiede betragen weniger als ein Prozent. Die Variation der unteren Schranke bewirkt eine Verschlechterung von $\widehat{D}_E$, $\widehat{R}_E$ und folglich $\widehat{G}_E$, während $\widehat{APE}_E$ zunächst sogar bessere Werte annimmt.

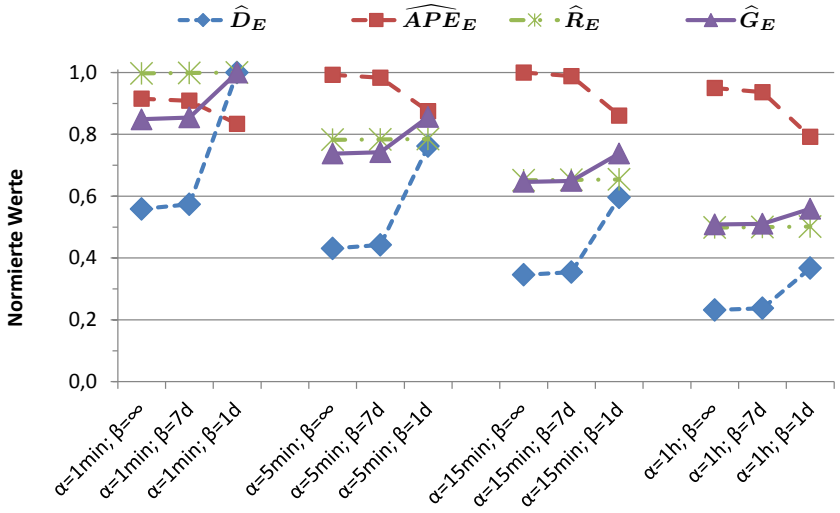

Abbildung 5.5.: MAVSync, Variation der Schranken α und β im Modus Einträge.

Diese Resultate sind wie folgt zu erklären: Im Modus Einträge dominieren Feeds, die im Zeitraum der Erstellung des Datensatzes sehr viele Einträge und somit eine sehr hohe Änderungsrate aufwiesen. Der Maximalwert lag bei rund 42 Einträgen pro Minute im Durchschnitt über vier Wochen. Auf diese Feeds hat ausschließlich die untere Schranke einen Einfluss. Eine Erhöhung der Schranke bewirkt seltenere Abfragen dieser Feeds, sodass die Wahrscheinlichkeit Einträge zu verpassen steigt. Dies ist am Rückgang des Recalls und Delays zu erkennen.

Die anfängliche Verbesserung und der anschließende Rückgang von $\widehat{APE}_E$ sind ein interessanter Effekt. Das seltenere Abfragen der Feeds mit den Werten $\alpha = 5$ Minuten

und $\alpha = 15$ Minuten führt bei sich entsprechend oft ändernden Feeds dazu, dass bei jeder Abfrage ein Fenster mit ausschließlich neuen Einträgen erlangt wird, sodass die Anzahl pro neuem Eintrag benötigter Abfragen sinkt und $\widehat{APE}_E$ steigt. Bei einem Wert von $\alpha = 1$ Stunde ist $\widehat{APE}_E$ aufgrund eines (nicht dargestellten) steigenden Wertes $\overline{APE}_E$ rückläufig. Auch dieser Effekt kann anhand der Eigenschaften des Datensatzes und eines Beispiel-Feeds mit einer Fenstergröße von 50 nachvollzogen werden.

Beim kleinsten evaluierten Wert von $\alpha = 1$ Minute werden vor allem bei Feeds mit sehr hohen Änderungsraten häufig mehrere neue Einträge pro Abfrage gefunden, im Beispiel seien dies 42 neue Einträge pro Abfrage. MAVSync ermittelt diese Änderungsrate und ein Abfrageintervall von 1–2 Sekunden, welches von $\alpha = 1$ Minute auf diesen Wert angehoben wird. In die Bildung des Durchschnitts $\overline{APE}_E$ gehen für diesen Feed 28.800 Abfragen (1 pro Minute im Testzeitraum von 20 Tagen) und rund 1,2 Mio. neue Einträge ein, für den Feed ergeben sich 0,024 Abfragen pro neuem Eintrag. Wird eine höhere untere Schranke gewählt, so werden bei jeder Abfrage 50 neue Einträge gefunden und es ergeben sich 0,020 Abfragen pro Eintrag. Über die Testphase nehmen die Anzahl getätigter Abfragen und die Anzahl insgesamt gefundener neuer Einträge jedoch ab. In die Berechnung des Durchschnitts über alle Einträge aller Feeds gehen somit deutlich weniger Einträge dieses Feeds in die Berechnung ein, sodass der sehr gute APE-Wert dieses Feeds im Vergleich zu anderen Feeds an Gewicht verliert. Selten geänderte Feeds werden von der unteren Schranke nicht beeinflusst, die Anzahl Abfragen und gefundener Einträge dieser Feeds ändert sich nicht. Aus diesem Grund wird $\widehat{APE}_E$ bei einer weiteren Erhöhung der unteren Schranke zunehmend schlechter. Es wird vermutet, dass sich $\overline{APE}_E$ bei weiterer Erhöhung von α dem Reziproken der durchschnittlichen Fenstergröße nähert, wenn bei jedem Feed und jeder Abfrage ausschließlich neue Einträge gefunden werden. Diese Hochrechnung ist jedoch für den praktischen Einsatz der Algorithmen von geringer Bedeutung, da eine deutliche Erhöhung von α zu einem inakzeptablen Recall führen wird, der gegen Null geht.

Insgesamt ergibt sich die höchste Güte des Algorithmus MAVSync bei $\alpha = 1$ Minute und $\beta = 1$ Tag. Dies trifft ebenso auf die nicht im Detail vorgestellten Algorithmen $\mathsf{FixLearned}_A$ und $\mathsf{IndHist}_{0,7}$ zu. Diese drei werden nun im direkten Vergleich betrachtet.

Vergleich der besten Algorithmen im Modus Einträge

Die von den Algorithmen $\mathsf{FixLearned}_A$, $\mathsf{IndHist}_{0,7}$ und MAVSync erzielten Bestwerte werden in Tabelle 5.2 und Abbildung 5.6 präsentiert. Der Vergleich zeigt den unterschiedlichen Einfluss der Schranken auf die drei Algorithmen, die in Abschnitt 4.4.2 nahezu identische Gütewerte erzielten.

MAVSync und $\mathsf{FixLearned}_A$ profitieren in etwa gleichermaßen von einer oberen Schranke von einem Tag. Die Güte kann um 18 % bzw. 17 % gesteigert werden, sodass MAVSync die Führung verteidigt und marginal ausbaut. Der PostRate-Algorithmus $\mathsf{IndHist}_{0,7}$ profitiert von den Einschränkungen des Intervalls weniger als die anderen beiden und fällt im Vergleich anhand der Gütewerte zurück.

Abs.	Algorithmus	Schranken		Durchschnittswerte			auf Besten normiert			
		α	β	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
4.4.2	FixLearned$_A$	1min	∞	44min	1,141	0,964	0,525	0,971	0,997	0,848
	IndHist$_{0,7}$	1min	∞	47min	1,108	0,960	0,494	1,000	0,993	0,838
	MAVSync	1min	∞	42min	1,212	0,963	0,559	0,914	0,996	0,849
5.1.2	FixLearned$_A$	1min	1d	26min	1,255	0,967	0,912	0,883	1,000	0,989
	IndHist$_{0,7}$	1min	1d	31min	1,216	0,965	0,750	0,911	0,998	0,936
	MAVSync	1min	1d	23min	1,330	0,966	1,000	0,833	0,999	1,000

Tabelle 5.2.: Vergleich der besten Algorithmen im Modus Einträge unter Berücksichtigung der Schranken α und β.

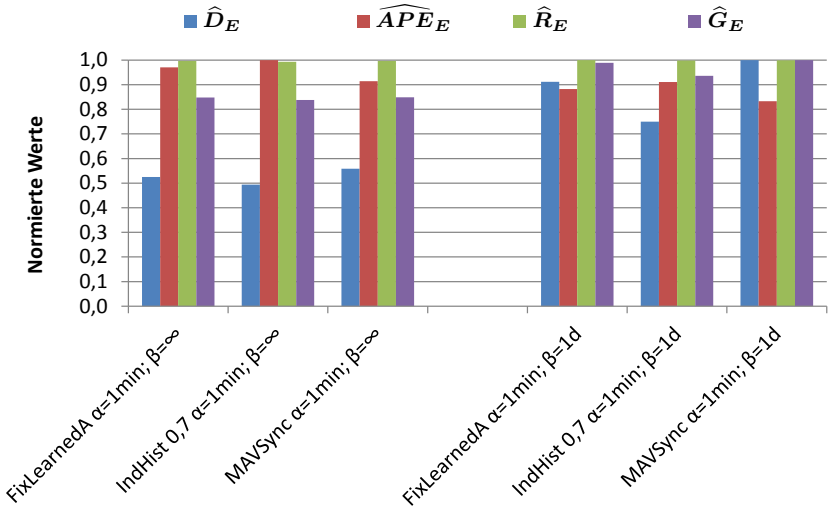

Abbildung 5.6.: Vergleich der besten Algorithmen im Modus Einträge unter Berücksichtigung der Schranken α und β.

5.1.3. Bewertung der Schranken

Insgesamt wurden im Rahmen des Abschnitts 5.1 die Daten von 60 Simulationen ausgewertet (fünf Algorithmen mit je zwölf Kombinationen α und β). Die Ergebnisse zeigen die teilweise gegenläufigen Auswirkungen der Schranken.

Bei der Bildung der Durchschnittswerte über alle Feeds nehmen die betrachteten Werte der unteren Schranke nur marginal Einfluss auf die erzielte Güte. Die Auswirkung der oberen Schranke ist hingegen verschieden: Jede weitere Verkürzung des Abfrageintervalls führt bei LIHZ$_{1,0}$ zu einer sinkenden Güte, bei IndHist$_{10,0}$ steigt die Güte zunächst und fällt anschließend wieder. Bei MAVSync steigt die Güte mit jedem weiteren Absenken von β.

Im Durchschnitt über alle Einträge sind die Auswirkungen auf alle drei Algorithmen gleich. Eine obere Beschränkung des Abfrageintervalls führt zu einer höheren Güte, während eine (zunehmende) Begrenzung des minimalen Intervalls zwischen zwei Abfragen stets zu schlechterer Güte führt. Es gilt zu beachten, dass die Aussage bezüglich α allgemeingültig ist, während eine weitere Reduktion von β dazu führen kann, dass beim

Erreichen eines Schwellwertes für β der normierte Delay $\widehat{D}_E$ langsamer steigt als $\widehat{APE}_E$ sinkt und somit $\widehat{G}_E$ ab diesem Wert für β wieder sinkt. Der Recall $\overline{R}_E$ hat hierauf kaum Einfluss, da alle Algorithmen bereits nahe dem theoretischen Maximum sind und $\overline{R}_E$ nur noch wenige Prozentpunkte steigen kann.

Insgesamt profitiert MAVSync am meisten vom Einsatz der Schranken. Diese Erkenntnis ist darauf zurückzuführen, dass MAVSync nicht an den Datensatz angepasst wurde – im Gegensatz zu den jeweils anderen Algorithmen, zu denen bereits in Kapitel 4.4.1 die geeignetsten Parameter bestimmt wurden. Die Ergebnisse zeigen, dass MAVSync im Modus Einträge das beste Ergebnis erzielt und im Modus Feeds nur wenige Prozentpunkte hinter dem Bestplatzierten $\mathsf{LIHZ}_{1,0}$ liegt. Zudem sei daran erinnert, dass $\mathsf{LIHZ}_{1,0}$ ausschließlich im Modus Feeds gute Ergebnisse erzielt und im Modus Einträge zu den schlechtesten Algorithmen zählt (vgl. Abbildung 4.14 auf Seite 144).

Zusammenfassend ergibt sich, dass an einer Reduktion des Delays oder Steigerung des Recalls interessierte Anwender die obere Schranke β verkleinern können, während eine Reduktion der Anzahl insgesamt getätigter Abfragen durch eine Erhöhung der unteren Schranke α erzielt werden kann. Diese unterschiedlichen Ziele werden in Abschnitt 5.3 im Detail beleuchtet, indem eine gewichtete Berechnung der Güte erfolgt. Die untere Schranke α hat primär Einfluss auf häufig aktualisierte Feeds, während sich β auf sich selten ändernde Feeds auswirkt. Darüber hinaus ermöglicht der Einsatz der oberen Schranke, den maximalen Delay zu begrenzen. Auf diese Weise ist es z. B. möglich, neue Einträge mit einer maximalen Verzögerung von einem Tag zu erlangen. Bezüglich APE und Recall können mit den Schranken keine garantierten Werte erzielt werden.

5.2. Auswertung abfragespezifischer Attribute

Die zweite Verfeinerung der Vorhersage ist die Auswertung abfragespezifischer Attribute. In Abschnitt 2.3.2 wurden im Rahmen der Vorstellung des Feed-Formats RSS 2.0 die vier XML-Elemente ttl, cloud, skipHours und skipDays identifiziert, die es Nutzern ermöglichen sollen, neue Einträge effizienter zu erlangen. Hierfür wird zunächst ihre Verbreitung im Datensatz rekapituliert und anschließend die Nutzung von ttl am Beispiel des Algorithmus MAVSync untersucht. In Abschnitt 1.5 wurde die These aufgestellt, dass die auf der puren Historie der Feeds basierende Vorhersage durchschnittlich bessere Ergebnisse erzielt, als mit der Auswertung dieser Attribute möglich ist.

Die XML-Elemente werden im Folgenden zumeist als Attribute des Feeds bezeichnet, da sich für den Einsatz zur Vorhersage kein Unterschied ergibt, ob die vier genannten im Kontext von XML als Elemente oder Attribute definiert sind.

5.2.1. Verbreitung der Attribute im Datensatz

Die Verbreitung der Attribute wurde initial in Abschnitt 3.7.7 vorgestellt. Im Hinblick auf die beiden Modi zur Bewertung der Algorithmen wird die Verbreitung der Attribute anhand des in allen Evaluationen eingesetzten Ausschnitts des Datensatzes von rund 179.000 Feeds und 33 Mio. Einträgen (vgl. Abschnitt 4.3.2) betrachtet.

Die time to live (ttl) wird in rund zwölf Prozent der Feeds angegeben, deren knapp 10 Mio. Einträge knapp ein Drittel aller betrachteten Einträge ausmachen. Die Verteilung der ttl-Werte wurde in Abbildung 3.12 (Seite 104) analysiert. Der Verteilung der Werte nach zu urteilen werden diese von den Diensteanbietern manuell konfiguriert oder Standardwerte übernommen, die nicht den tatsächlichen Änderungsraten der Feeds entsprechen. Eine Stichprobe bestätigt dies: Unter allen das ttl-Attribut unterstützenden Feeds weisen drei Feeds je rund 1,7 Mio. Einträge und eine Fenstergröße von 50 Einträgen auf, sodass ein Fenster im Durchschnitt nach einer reichlichen Minute ausschließlich neue Einträge enthält. Diese drei Feeds haben einen ttl-Wert von drei Minuten, der zu Misses und einem geringen Recall führen wird. Dies gilt es in einer anschließenden Evaluation in Abschnitt 5.2.2 empirisch zu belegen.

Das Attribut cloud wurde in knapp vier Prozent der Feeds des Datensatzes gefunden. Diese Feeds beinhalten nur 0,5 % der Einträge des Datensatzes. Aufgrund der geringen Verbreitung erscheint eine explizite Analyse nicht lohnenswert.

Die Attribute skipHours und skipDays werden nahezu von keinem Feed bereitgestellt. Weniger als 0,1 % der Feeds des Datensatzes weisen zumindest eines der Attribute auf. Der Anteil der Einträge dieser Feeds beträgt ebenfalls weniger als 0,1 % der Einträge des Datensatzes, sodass weitere Analysen als nicht aussichtsreich bezüglich des Gesamtergebnisses betrachtet werden.

5.2.2. Vorhersage von Änderungen unter Auswertung der ttl

Zur Auswertung der ttl wird MAVSync mit den in Kapitel 4.4 verwendeten Parametern $\alpha = 1$ Minute und $\beta = \infty$ eingesetzt. Feeds, die nicht über das ttl-Attribut verfügen, werden somit wie bereits bekannt abgefragt.

Die Definition des Attributs bezieht sich auf die Domäne Web-Caching. ttl ist die Zeitspanne in Minuten, die ein Feed in einem Cache verweilen kann, bevor er erneut von der Quelle abzurufen ist. Im Kontext der Vorhersage neuer Einträge sind zwei Interpretationen möglich: Einerseits kann die ttl als untere Schranke interpretiert werden, sodass ein Feed keinesfalls in kürzeren als dem in der ttl spezifizierten Intervall abgefragt wird, längere Intervalle jedoch möglich sind. Die Alternative ist, die ttl sowohl als untere als auch obere Schranke zu interpretieren und somit die ttl als Abfrageintervall zu wählen. Dies entspricht der Verwendung in einem Cache: Überschreitet die Verweildauer des Feeds im Cache den als ttl spezifizierten Wert und fragt ein Nutzer diesen Feed beim Cache an, so überprüft dieser zunächst die Quelle auf Änderungen, ehe die Antwort an den Nutzer übermittelt wird. Die Resultate der Alternativen sind in Abbildung 5.7 gegenübergestellt und den Tabellen 5.3 und 5.4 zu entnehmen.

Schranken		Durchschnittswerte			auf Besten normiert			
α	β	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
1min	∞	41h:21min	2,173	0,788	0,870	0,985	0,969	0,996
1min / ttl	∞	41h:23min	2,141	0,787	0,869	1,000	0,967	1,000
1min / ttl	∞ / ttl	35h:58min	119,187	0,813	1,000	0,018	1,000	0,277

Tabelle 5.3.: MAVSync, Auswertung des Attributs ttl, Durchschnitt über alle Feeds.

Schranken		Durchschnittswerte			auf Besten normiert			
α	β	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
1min	∞	42min	1,212	0,963	1,000	1,000	1,000	1,000
1min / ttl	∞	48min	1,248	0,858	0,857	0,971	0,890	0,905
1min / ttl	∞ / ttl	43min	4,391	0,861	0,959	0,276	0,895	0,619

Tabelle 5.4.: MAVSync, Auswertung des Attributs ttl, Durchschnitt über alle Einträge.

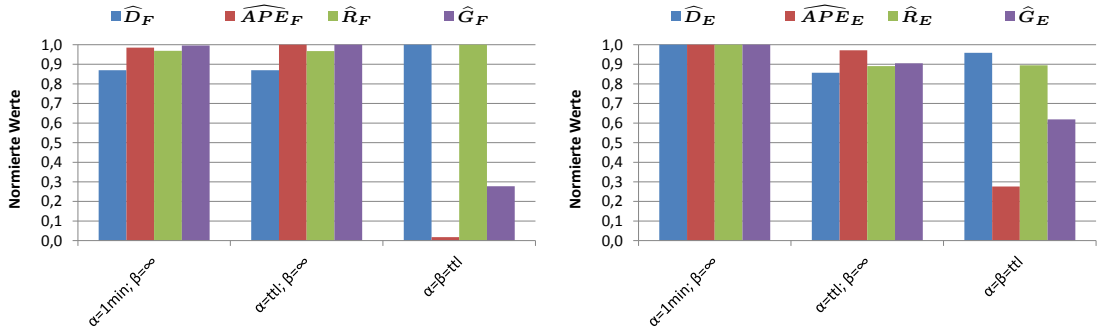

Abbildung 5.7.: MAVSync, Auswertung des abfragespezifischen Attributs ttl. Vergleich des Durchschnitts über alle Feeds (links) und Einträge (rechts).

Die jeweils erste Zeile enthält die Werte aus Abschnitt 4.4, die darauf folgenden entsprechen den genannten Alternativen der Interpretation der ttl. Die Ergebnisse bestätigen die aufgestellte These, wenn auch mit einer minimalen Einschränkung: Im Modus Feeds kann bei der Interpretation der ttl als untere Schranke ein marginal besserer Wert $\overline{APE}_E$ erzielt werden. Die sich ergebende Güte ist um wenige Tausendstel besser als der Vergleichswert aus Abschnitt 4.4.

Insgesamt werden die Resultate folgendermaßen interpretiert: Im Modus Feeds führt die Verwendung der ttl als Abfrageintervall ($\alpha = \beta = ttl$) zu einem geringeren Delay, da eine Vielzahl der Feeds den sehr kleinen ttl-Wert zehn Minuten aufweisen. Dies hat häufige Abfragen und somit einen drastischen Anstieg von $\overline{APE}_F$ um den Faktor 50 zur Folge. Der Recall steigt leicht. Im Modus Einträge führt keine der Alternativen $\alpha = \beta = ttl$ bzw. $\alpha = ttl; \beta = \infty$ zu besseren Resultaten. Die bereits in Abschnitt 5.2.1 beschriebenen Feeds mit sehr hoher Änderungsrate dominieren die Ergebnisse und bewirken ein Absenken von $\widehat{D}_E$ und $\widehat{R}_E$. Der Einfluss zu häufig abgefragter Feeds ist nicht so stark wie im Modus Feeds, der Rückgang von $\widehat{APE}_E$ ist schwächer als im Modus Feeds.

Unter der Maßgabe, das Attribut ttl als $\alpha = \beta = ttl$ zu interpretieren, gilt die These für dieses Attribut als erfüllt. Betrachtet man die Resultate der Alternative $\alpha = ttl$ in beiden Modi gemeinsam, so kann mit der als baseline verwendeten Konfiguration $\alpha = 1$ Minute und $\beta = \infty$ im Durchschnitt ebenfalls eine höhere Güte erzielt werden. Aufgrund der deutlich geringeren Verbreitung von cloud, skipHours und skipDays wird abgeschätzt, dass diese das schlechtere Ergebnis der ttl nicht ausgleichen, selbst wenn die betreffenden Feeds mit einem Recall von Eins sowie dem Wert Null für Delay und APE abgefragt werden.

5.3. Generalisierung der Güte

In diesem Abschnitt wird Anwendern ein Werkzeug zur Hand gegeben, mit dem sie die Algorithmen anhand ihrer Anforderungen bewerten und z. B. den Recall höher gewichten können, wenn sie möglichst wenige Einträge verpassen möchten.

Die Generalisierung erfolgt in zwei Schritten. Zunächst werden verschiedene Möglichkeiten der Gewichtung der einzelnen Kriterien miteinander verglichen. Im Anschluss werden die beiden separat berechneten Werte $\widehat{G}_F$ und $\widehat{G}_E$ zu einem Gesamtwert zusammengefasst.

5.3.1. Gewichtete Berechnung der Güte

Die Güte wurde in Abschnitt 4.3.6 definiert: Zunächst wird das geometrische Mittel der drei Kriterien $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ separat in den Modi Feeds und Einträge berechnet. Die berechneten Mittelwerte werden in Relation zum Besten normiert und das Ergebnis als Güte bezeichnet. Die Gewichtung der drei Metriken Delay, APE und Recall erfolgt gleichmäßig, sodass z. B. eine Halbierung des Delays bei gleichzeitiger Verdopplung der APE keine Änderung der Güte zur Folge hat.

Um einzelne Metriken hervorzuheben wird eine gewichtete Güte eingeführt, die auf dem gewichteten geometrischen Mittel basiert. Zur Berechnung wird Gleichung 4.20 (Seite 133) zu den Gleichungen 5.1 und 5.2 erweitert, die Berechnung im Modus Einträge erfolgt analog. Wird jedem der drei Gewichte w_D, w_{APE} und w_R der Wert 1 zugewiesen, so ergibt sich das gleichgewichtete geometrische Mittel aus Gleichung 4.20, das bereits in Kapitel 4 eingesetzt wurde.

$$G_F = \sqrt[w]{(\widehat{D}_F)^{w_D} \cdot (\widehat{APE}_F)^{w_{APE}} \cdot (\widehat{R}_F)^{w_R}} \tag{5.1}$$

$$w = w_D + w_{APE} + w_R \tag{5.2}$$

Um eines der Kriterien wie den Recall mit dem doppelten Gewicht in die Berechnung einfließen zu lassen, wird $w_R = 2$ gesetzt und den Gewichten der übrigen Kriterien der Wert 1 zugewiesen. Die sich nach anschließender Normierung ergebende Güte im Modus Feeds sei als $\widehat{G}_F^{R^2}$ notiert. Der bereits vertrauten Darstellung der Güte $\widehat{G}_F$ im Modus Feeds werden alle vom Wert 1 abweichenden Gewichte als oberer Index hinzugefügt, im Beispiel ist dies R^2 und zeigt die doppelte Wertigkeit des Recalls an. Der Normierungsschritt zur Überführung des gewichteten geometrischen Mittels G_F aus Gleichung 5.1 in die Güte erfolgt wie gehabt nach Gleichung 4.21.

Die Auswirkungen auf die sich aus der gewichteten Güte ergebende Reihenfolge der Algorithmen ist in Abbildung 5.8 dargestellt. In der oberen Grafik erfolgt die Berechnung im Modus Feeds, in der unteren Darstellung im Modus Einträge. In beiden Abbildungen werden Delay, APE und Recall jeweils doppelt in die Berechnung einbezogen. Zum Vergleich der gewichteten Güte mit den aus Abschnitt 4.4.2 bekannten (gleich gewichteten) Gütewerten werden diese ebenfalls aufgeführt. Die den Berechnungen zugrunde liegenden Werte $\overline{D}_F$, $\overline{APE}_F$, $\overline{R}_F$ sowie $\overline{D}_E$, $\overline{APE}_E$, $\overline{R}_E$ entsprechen denen aus dem Vergleich aller Algorithmen und können den Tabellen 4.11 und 4.12 entnommen werden.

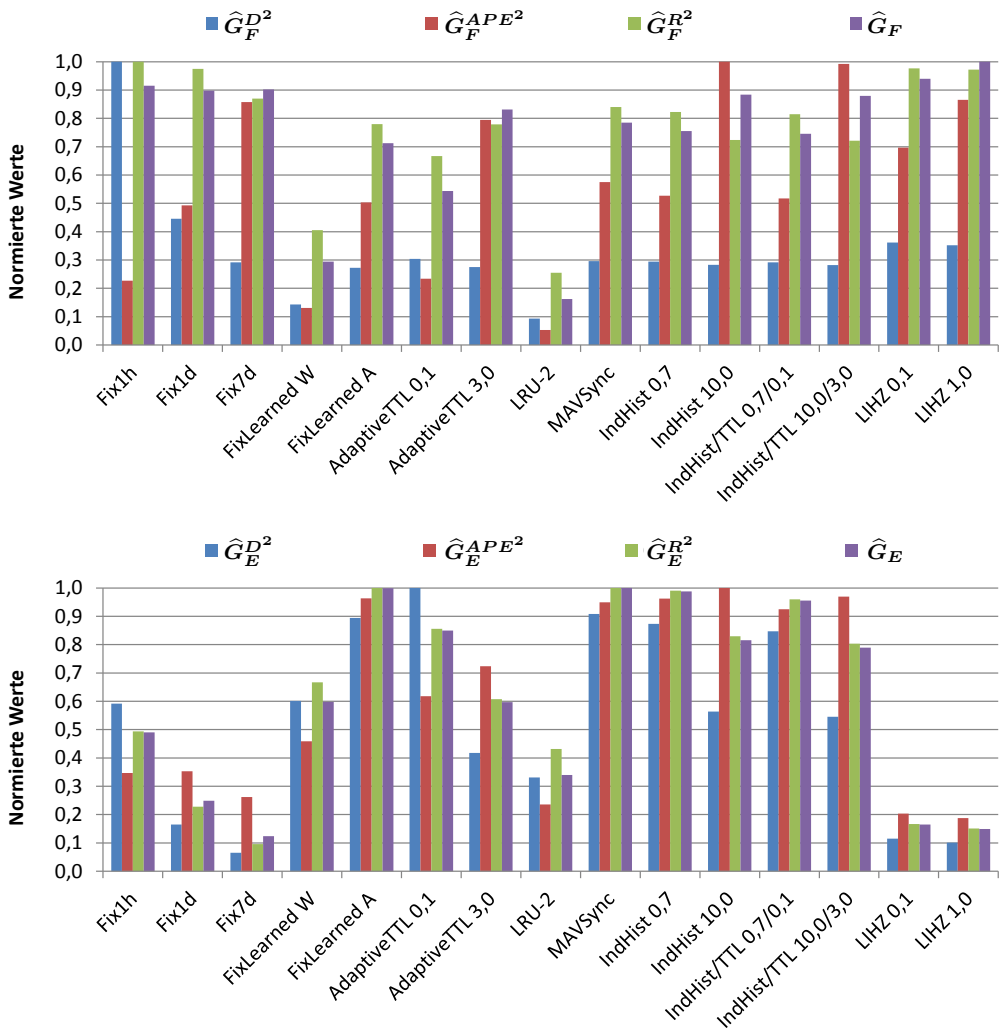

Abbildung 5.8.: Vergleich verschiedener Gewichtungen der Güte im Durchschnitt über alle Feeds (oben) und Einträge (unten).

Im Modus Feeds erzielt Fix1h den mit Abstand besten Wert für $\widehat{G}_F^{D^2}$. Dies ist auf den vergleichsweise sehr geringen durchschnittlichen Delay $\overline{D}_F$ von 31 Minuten zurückzuführen. Der Großteil der Algorithmen liegt im gering gestreuten Mittelfeld bei $\widehat{G}_F^{D^2} = 0{,}3$. Bei doppelter Gewichtung des Recalls führt Fix1h ebenfalls das Feld an, der Abstand der Algorithmen zueinander ist hier jedoch deutlich geringer. Die Algorithmen mit fixem Intervall sowie beide LIHZ-Varianten erzielen die besten Resultate, MAVSync liegt 16 % hinter dem Bestwert und somit im vorderen Mittelfeld. Bezüglich $\widehat{APE}_F$ erzielen die auf den Modus Feeds optimierten Algorithmen die besten Werte. Diese sind $\text{IndHist}_{10,0}$, $\text{IndHist/TTL}_{10,0/0,1}$, $\text{LIHZ}_{1,0}$ und $\text{AdaptiveTTL}_{3,0}$.

Insgesamt ist zu erkennen, dass die höhere Gewichtung einzelner Kriterien die aus der (gewichteten) Güte resultierende Reihenfolge der Algorithmen deutlich beeinflussen kann.

Die Auswirkungen sind umso größer, je breiter die zugrunde liegenden Durchschnittswerte $\overline{D}_F$, $\overline{APE}_F$ und $\overline{R}_F$ gestreut sind.

Im Modus Einträge fällt zunächst auf, dass die vier Gütewerte eines Algorithmus in der Regel eine deutlich geringere Streuung aufweisen. MAVSync zählt in den drei gewichteten Werten sowie der gleich gewichteten Güte zu den Besten und führt das Feld für $\widehat{G}_E^{R^2}$ und $\widehat{G}_E$ knapp an. AdaptiveTTL$_{0,1}$ profitiert von der hohen Anzahl getätigter Abfragen und erreicht den besten Wert für $\widehat{G}_E^{D^2}$, weist jedoch im Gegenzug einen deutlich schlechteren Wert für $\widehat{G}_E^{APE^2}$ auf. Die Werte der besten Algorithmen liegen allesamt sehr dicht beieinander, während die Algorithmen mit fixem Intervall sowie beide LIHZ-Varianten die schlechtesten Resultate erzielen. Dies entspricht den Beobachtungen aus Abschnitt 4.4.2.

Die sich aus den verschiedenen Möglichkeiten der Gewichtung ergebende Verschiebung der Reihenfolge ist im Modus Einträge weniger stark ausgeprägt als im Modus Feeds. Um die Beobachtungen in Korrelation zu betrachten, werden die Werte beider Modi im Anschluss zu einem zusammengeführt.

Bezüglich der in diesem und dem nachfolgenden Abschnitt betrachteten Evaluationsergebnisse muss beachtet werden, dass diese nur ein Beispiel darstellen. Um einen direkten Vergleich der hier erzielten Resultate mit den bereits aus Kapitel 4 bekannten Ergebnissen zu ermöglichen, wurden exakt die selben Varianten der Algorithmen AdaptiveTTL, IndHist, IndHist/TTL und LIHZ aufgegriffen, die in Abschnitt 4.4.2 vorgestellt wurden. Die in Abschnitt 4.4.1 ermittelten Parameter dieser vier Algorithmen wurden mit dem Ziel der Maximierung der gleich gewichteten Güte ermittelt, sodass sich für die alternativen Gewichtungen der Güte bessere Werte ergeben können, wenn die Parameter M, θ und σ erneut optimiert werden. Diese Spezialisierung wurde nicht durchgeführt, um stattdessen die Wechselwirkungen der gewichteten Gütewerte untereinander zu beleuchten.

5.3.2. Durchschnittliche Güte über beide Modi

Der letzte Schritt in der Generalisierung der Güte ist die Zusammenführung von $\widehat{G}_F$ und $\widehat{G}_E$ zu einem einzigen Gütewert $\widehat{G}_\emptyset$. Dieser ergibt sich aus dem (gleich gewichteten) geometrischen Mittel der Werte beider Modi $\widehat{G}_F$ und $\widehat{G}_E$, sowie einer anschließenden Normierung auf den Besten. Auf diese Weise ist es möglich, eine Menge zu betrachtender Algorithmen anhand eines einzigen Wertes zu vergleichen. Dies ist im Sinne eines Anwenders, der auf der Suche nach einem Algorithmus ist, der seinen Bedürfnissen am besten entspricht.

Ein Nutzer formuliert seine Ziele, indem er zuerst den drei Metriken Delay, APE und Recall die gewünschten Gewichte zuweist. Anschließend kann er beiden Modi zur Bildung der durchschnittlichen Güte $\widehat{G}_\emptyset$ ebenfalls verschiedene Gewichte zuordnen. Eine Betonung des Modus Feeds führt dazu, negative Ausreißer bezüglich der Feeds stärker zu gewichten, sodass z. B. Algorithmen schlechter abschneiden, die zwar für Feeds mit sehr vielen Einträgen gute Werte erzielen aber nur selten aktualisierte Feeds benachteiligen und für diese einen sehr hohen Delay erzielen. Verschiebt ein Anwender das Gewicht in Richtung des Modus Einträge, so ist es ihm wichtiger, für möglichst viele Einträge gute Werte zu erzielen. Die unter Umständen sehr schlechten Werte einzelner Feeds mit wenigen Einträgen nimmt er in Kauf. Für das betrachtete Szenario der Extraktion von

Produktinformationen aus Feeds mit dem Ziel der Integration in ein PIS erscheinen beide Modi wichtig, sodass ein Gleichgewicht zwischen den Feeds mit sehr vielen Einträgen auf der einen Seite und der Gesamtmenge der Feeds auf der anderen Seite erzielt werden soll. Dies wird anhand der in Abschnitt 5.3.1 erzielten Resultate untersucht. Abbildung 5.9 enthält die sich aus den beiden Modi Feeds und Einträge ergebenden durchschnittlichen Gütewerte $\widehat{G}_{\emptyset}$, wobei $\widehat{G}_F$ und $\widehat{G}_E$ (sowie $\widehat{G}_F^{D^2}$, $\widehat{G}_E^{D^2}$; $\widehat{G}_F^{APE^2}$, $\widehat{G}_E^{APE^2}$; $\widehat{G}_F^{R^2}$, $\widehat{G}_E^{R^2}$) paarweise gleich gewichtet werden.

Bei der Zusammenführung ist zu beachten, dass zur Berechnung von $\widehat{G}_F$ und $\widehat{G}_E$ zwar die selben Feeds und Einträge verwendet werden, es jedoch vorkommen kann, dass zur Berechnung der drei in $\widehat{G}_F$ eingehenden Werte $\overline{D}_F$, $\overline{APE}_F$ und $\overline{R}_F$ nur eine Teilmenge der Feeds eingeht. Dies geschieht genau dann, wenn einige Feeds im Beobachtungszeitraum keinen neuen Eintrag aufweisen. Der Effekt kann sowohl zu besseren als auch schlechteren durchschnittlichen Werten im Modus Feeds führen. Er ist umso geringer, je größer der Beobachtungszeitraum ist, je mehr Feeds betrachtet werden und je höher die Änderungsraten der Feeds sind.

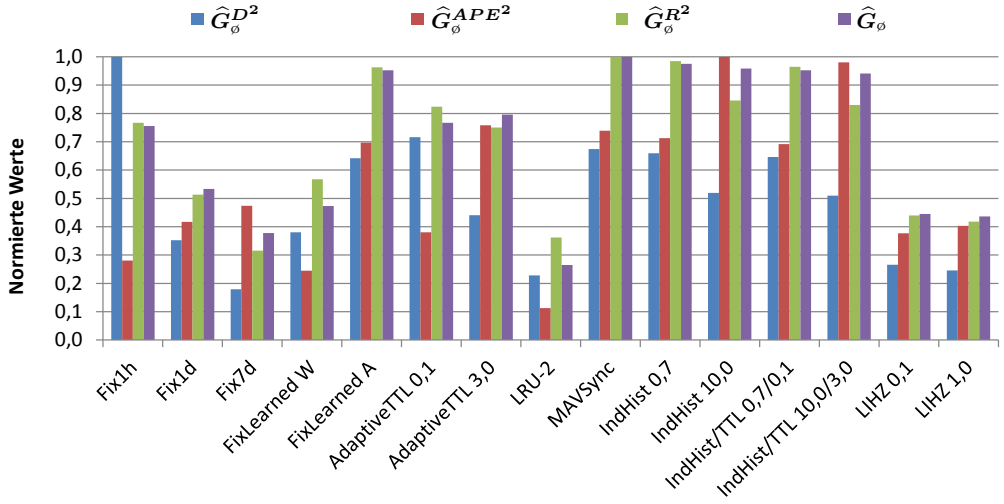

Abbildung 5.9.: Vergleich verschiedener Gewichtungen der Güte im Durchschnitt über beide Modi Feeds und Einträge.

Der eigene Algorithmus MAVSync erzielt die höchste Güte $\widehat{G}_{\emptyset}$ sowie ebenfalls den höchsten Wert für $\widehat{G}_{\emptyset}^{R^2}$ und setzt sich leicht von den übrigen Algorithmen ab. Dies spricht für den Einsatz von MAVSync in Szenarien wie dem vorgestellten PIS, in denen entweder keine Präferenzen bezüglich einer Metrik oder eines Modus vorliegen oder ein hoher Recall von besonderer Bedeutung ist. Eine weitere interessante Beobachtung ist, dass sich MAVSync um wenige Prozent von $\mathsf{FixLearned}_A$ abhebt, aus dem MAVSync hervorgegangen ist. Der Synchronisationsschritt und die kontinuierliche Anpassung des Abfrageintervalls an die momentane Änderungsrate des Feeds zeigen bereits in einer Testphase von drei Wochen einen Vorteil gegenüber einem initial gelernten, statischen Intervall.

Die geringe Anzahl erfolgter Abfragen führen $\mathsf{IndHist}_{10,0}$ und $\mathsf{IndHist/TTL}_{10,0/0,1}$ an die

Spitze, wenn APE höher gewichtet wird. Dies erfolgt jedoch zum Preis eines schlechteren Delays und Recalls. Der im Schnitt beste Delay wird mit Fix1h erzielt. Der Algorithmus weist mit Abstand den besten Wert $\widehat{G}_F^{D^2}$ auf, sodass sich auch im Durchschnitt über beide Modi der beste Wert $\widehat{G}_{\emptyset}^{D^2}$ ergibt. Die Spezialisierungen von Fix1h und $\mathsf{IndHist}_{10,0}$ führen jedoch zu deutlich schlechteren Werten der anderen Gewichtungen. Der Rückstand von MAVSync bezüglich $\widehat{G}_F^{D^2}$ und $\widehat{G}_F^{APE^2}$ kann durch Einsatz der Schranken $\alpha = 1$ Minute und $\beta = 1$ Tag reduziert werden. Die Auswirkungen der Schranken wurden in Abschnitt 5.1 eruiert. Die Evaluation zeigt, dass MAVSync am meisten vom Einsatz der Schranken profitiert und somit näher an den jeweiligen Besten ist, als ohne Verwendung der Schranken.

5.3.3. Gesamtbild der Algorithmen

Es ergibt sich folgendes Gesamtbild der Algorithmen, die in absteigender Reihenfolge der erzielten Gütewerte betrachtet werden. MAVSync ist mit leichtem Vorsprung der beste Algorithmus einer ausgewogenen Betrachtung. Dies ist von besonderer Bedeutung, da das Resultat ohne vorherige Optimierung von Parametern erzielt wird. Diese Erkenntnis bestätigt die zweite in Abschnitt 1.5 aufgestellte These, Änderungen von Feeds anhand der partiellen Historie eines Feeds vorherzusagen zu können, deren Qualität äquivalent zu aufwendigeren Verfahren wie der Modellierung der Änderungsrate als inhomogener Poisson-Prozess ist. Die empirische Untersuchung zeigt, dass die erzielte Güte leicht höher ist.

Das IndHist zugrunde liegende Modell der Änderungsrate wurde anhand eines lückenlosen Trainingsdatensatzes gelernt und kann in realen Umgebungsbedingungen ohne Zuhilfenahme anderer Algorithmen nicht reproduziert werden, da IndHist selbst nicht zum Trainieren des Modells konzipiert ist und auch keine Alternativen vorgeschlagen werden. Dennoch führt der Algorithmus im Mittel zu schlechteren Resultaten als MAVSync.

$\mathsf{FixLearned}_A$ erzielt generell um den Faktor zwei bessere Werte als $\mathsf{FixLearned}_W$. Dies verdeutlicht den Vorteil, den aktuellen Zeitpunkt einer Abfrage mit in die Berechnung des Intervalls einfließen zu lassen und nicht ausschließlich die in einem Fenster enthaltenen Werte zu berücksichtigen.

Die beiden Alternativen des AdaptiveTTL führen zu ähnlichen Gütewerten und unterscheiden sich untereinander in $\widehat{G}_{\emptyset}^{D^2}$ und $\widehat{G}_{\emptyset}^{APE^2}$. Dies zeigt sehr deutlich die Wechselwirkung dieser beiden Metriken.

Die Wahl eines fix vorgegeben Abfrageintervalls ist nur dann geeignet, wenn der Delay von besonderem Interesse ist *und* das Intervall zudem entsprechend an die Verteilung der Änderungsraten des Datensatzes angepasst wird. Der beachtliche Unterschied der Gütewerte $\widehat{G}_{\emptyset}^{D^2}$ von Fix1h, Fix1d und Fix7d verdeutlicht die Notwendigkeit der Anpassung.

Das grobgranulare Modell und das minimale Abfrageintervall von einem Tag führen zu unterdurchschnittlichen Resultaten von LIHZ. Der Algorithmus ist nicht geeignet, wenn Feeds mit hohen Änderungsraten zu verarbeiten sind.

Als am wenigsten geeignet stellte sich LRU-2 heraus, bei dem nur der neuste Eintrag eines Fensters in die Vorhersage der nächsten Änderung eingeht. Wie bereits in Abschnitt 4.4.2 untersucht wurde, ist dieser Ansatz besonders ungeeignet, wenn die Änderungsraten variieren.

5.4. Erweiterungsmöglichkeiten

Im Rahmen der Arbeit wurden einige Erweiterungsmöglichkeiten wie die Verfeinerung des Standardintervalls η und die Betrachtung begrenzter Ressourcen identifiziert. Diese werden im Folgenden kurz skizziert. Zudem wird das Szenario der Integration von Feeds in ein PIS aufgegriffen und Möglichkeiten aufgezeigt, Erkenntnisse aus der Extraktion und Verwaltung der Produktinformationen in die Vorhersage einfließen zu lassen.

5.4.1. Verfeinerung des Standardintervalls η

Im Bereich der Algorithmen wurde der Einfluss des Standardintervalls η anhand Abbildung 4.12 (Seite 140) analysiert: Je seltener alle Feeds abgefragt werden, umso stärker fällt der für η gewählte Wert ins Gewicht. Das Standardintervall wird genau dann zum Abfragen eines Feeds gewählt, wenn ein Algorithmus kein Intervall anhand der ihm zur Verfügung stehenden Informationen ermitteln kann (vgl. Abschnitt 4.2.1).

Das erwartete Optimierungspotential besteht in einer Differenzierung der Ursachen, die zur Verwendung des Standardintervalls führen. Die Änderungsrate eines Feeds kann nicht ermittelt werden, wenn weniger als zwei verschiedene Zeitstempel zur Verfügung stehen. Die Idee ist, zwischen Feeds zu unterscheiden, deren Einträge keine Zeitstempel aufweisen und jenen, die über gar keine Einträge verfügen. Selteneres Abfragen, z. B. täglich oder wöchentlich, der dem Muster Empty zuzuordnenden Feeds minimiert den APE-Wert und führt nur dann zu einem geringeren Recall und Delay, wenn der erste Eintrag in einem Feed gefunden wird.

Die Schwierigkeit besteht in der Unterscheidung von Feeds, die über einen längeren Zeitraum dem Muster Empty zuzuordnen sind und jenen, deren Fenstergröße nur temporär 0 beträgt. Betrachtungen in Abschnitt 3.7.2 ergaben, dass einige Feeds alternierende Fenstergrößen aufweisen, von denen ein Wert 0 ist. Weist ein solcher Feed beim ersten Aufruf eine Fenstergröße von 0 auf und wird folglich z. B. erst nach einer Woche erneut abgefragt, so resultieren ein sehr hoher Delay und potentiell ein geringer Recall. Diese sich aus der Verfeinerung des Standardintervalls ergebenden Risiken gilt es zu analysieren und gegeneinander abzuwägen.

5.4.2. Betrachtung begrenzter Ressourcen

In der Arbeit wurden die zur Verfügung stehenden Ressourcen nicht begrenzt, jedoch das Ziel der Sparsamkeit verfolgt, sodass ein Feed nur dann abzufragen ist, wenn neue Einträge erwartet werden[2].

Eine alternative Betrachtung ist, die zur Verarbeitung der Feeds zur Verfügung stehenden Ressourcen zu beschränken. In Kapitel 2 wurden hierzu einige Ansätze betrachtet. So wird z. B. im System advaRSS zyklisch für jeden Feed die Anzahl der zu erwartenden neuen Einträge ermittelt und anschließend die Feeds mit dem größten Wert abgefragt (siehe Abschnitt 2.7.5). Aus den in der eigenen Arbeit gewonnenen Erkenntnissen ist dies jedoch nur ein Schritt in die Richtung der Betrachtung begrenzter Ressourcen:

[2] Die drei Fix*-Algorithmen mit einem festen, einheitlichen Intervall sind hiervon ausgenommen.

Die benötigten Systemressourcen sind nicht einzig von der Anzahl getätigter Abfragen abhängig, sondern insbesondere vom Umfang der zu verarbeitenden Daten. Die Verarbeitung eines 1 MB großen XML-Dokuments ist um ein Vielfaches aufwendiger als die Auswertung eines HTTP-Status-Codes 304 „not modified“. Hiervon abgesehen führt das reine Aufsummieren noch nicht abgerufener Einträge dazu, alle Feeds gleich zu behandeln und ihre Fenstergröße zu missachten. Je höher der Schwellwert ausstehender Einträge gewählt wird und je kleiner die Fenstergröße eines Feeds ist, umso stärker sinkt der Recall dieses Feeds. Die Beispiele ermöglichen einen Einblick in die Problematik begrenzter Ressourcen und die sich ergebende Breite dieses Forschungsgebietes.

Eine weitere Alternative ist, die Anzahl getätigter Abfragen zu minimieren, jedoch nicht zu beschränken. Verfolgt ein Anwender das Ziel, die in den Feeds enthaltenen Informationen nur gelegentlich auszuwerten, jedoch möglichst nichts zu verpassen und die Quellen nur selten abzufragen, so ist der Recall zu maximieren und APE zu minimieren. Die Auswirkungen auf den Delay spielen keine Rolle. Zur Realisierung dieser Ziele ist es notwendig, die betrachteten Algorithmen dahingehend anzupassen, dass diese möglichst nur dann den Feed erneut abfragen, wenn die Anzahl erwarteter Einträge der Fenstergröße bzw. einem Anteil der Fenstergröße entspricht. Je nach Variabilität der Änderungsrate eines Feeds sowie dessen Fenstergröße ist ein entsprechender „Puffer“ an Einträgen notwendig, sodass ein Feed z. B. genau dann abgefragt wird, wenn 80 % neue Einträge eines Fensters erwartet werden oder das Fenster nur noch drei bereits bekannte Einträge enthält. Die Wahl der Art des Puffers (prozentual oder absolut) sowie der Wert sind zu analysieren.

5.4.3. Gewichtung von Feeds und Einträgen basierend auf Informationen aus dem PIS

Alle Feeds sowie deren Einträge wurden bisher als gleichwertig betrachtet. Zur Ermittlung der Änderungsrate der Feeds ist dies zielführend. Aus dem Szenario der Extraktion von Produktinformationen aus den Feeds und der anschließenden Integration dieser in ein PIS (siehe Abbildung 4.1, Seite 109) ist das folgende Optimierungspotential denkbar: Im Anschluss an die Verarbeitung des Feeds durch den Feed Data Provider werden die neuen Einträge dem PIS übergeben und dort einer Extraktion der potentiell enthaltenen Produktinformationen unterzogen. Hier könnte für jeden Eintrag entschieden werden, ob dieser relevante Informationen enthielt oder nicht und das Ergebnis der Entscheidung dem Feed Data Provider übermittelt wird.

Die Idee ist, die Änderungsrate des Feeds nicht anhand aller Einträge zu ermitteln, sondern nur die Einträge zu berücksichtigen, aus denen Produktinformationen extrahiert werden konnten. Es ist zu erwarten, dass Feeds um so seltener abgefragt werden, je seltener sie Produktinformationen enthalten. Seltenere Abfragen bedingen zumeist ein Ansteigen des Delays und Sinken des Recalls, sodass auch hier die Vor- und Nachteile zu untersuchen sind.

5.5. Zusammenfassung

In diesem Kapitel erfolgte eine Verfeinerung der Techniken zur Vorhersage, wobei in den Abschnitten 5.1 bis 5.3 drei Schwerpunkte im Detail betrachtet und anschließend in Abschnitt 5.4 einige Erweiterungsmöglichkeiten der Arbeit skizziert wurden.

Die Variation der Schranken α und β erfolgte am Beispiel der drei jeweils besten Algorithmen der Modi Feeds und Einträge, die vorab in Abschnitt 4.4.2 ermittelt wurden. Die Ergebnisse zeigten, dass der eigene Algorithmus MAVSync am meisten von der Optimierung profitiert. Im Modus Feeds holt er den Rückstand gegenüber $\mathsf{IndHist}_{10,0}$ auf und liegt nur noch wenige Prozent hinter $\mathsf{LIHZ}_{1,0}$, der weiterhin die höchste Güte erzielt. Im Modus Einträge verteidigt MAVSync die Führung und setzt sich leicht von den übrigen betrachteten Algorithmen ab.

In Abschnitt 5.2 wurden die abfragespezifischen Attribute des RSS 2.0-Formates untersucht. Die Simulationen bestätigten die These, dass die Vorhersage von Änderungen bessere Ergebnisse erzielt, wenn ausschließlich die in einem Fenster enthaltene partielle Historie der Feeds ausgewertet wird. Die Hinzunahme des abfragespezifischen Attributes ttl führte im Durchschnitt zu einer deutlich schlechteren Güte.

Die Generalisierung der Güte mit den Betrachtungen einer gewichteten Güte sowie der durchschnittlichen Güte über beide Modi Feeds und Einträge runden die Arbeit ab. Die Gewichtungen ermöglichen es Anwendern, die auf der Suche nach einem für ihren Einsatzzweck geeigneten Algorithmus sind, ihre Präferenzen bezüglich Delay, APE und Recall sowie den beiden Modi in Form von Gewichten auszudrücken und die Algorithmen anschließend zu vergleichen.

Die Zusammenfassung der Gütewerte beider Modi zu einem einzigen Wert bestätigt schließlich erneut die erste aufgestellte These, die drei Metriken zu einem Wert zusammenfassen zu können. Darüber hinaus zeigen die in Abbildung 5.9 dargestellten Evaluationsergebnisse, dass MAVSync in der Gesamtwertung, d. h. der gleich gewichteten Güte über beide Modi, den ersten Platz belegt. Dies untermauert die zweite aufgestellte These bezüglich der Änderungshistorie. Die von MAVSync getätigten Vorhersagen sind nicht nur von gleicher Qualität wie jene von aufwendigeren Algorithmen, sondern übertreffen diese sogar.

6. Zusammenfassung und Ausblick

Das übergeordnete Ziel dieser Dissertation ist im Web verfügbare Produktinformationen in ein PIS zu integrieren. Aus diesem breiten Forschungsfeld wurden Feeds als ein Aspekt herausgegriffen und im Detail untersucht. Feeds werden u. a. eingesetzt, um Nutzer in einem einheitlichen Format und aggregierter Form über Aktualisierungen oder neue Beiträge auf Webseiten zu informieren. Unglücklicherweise werden bei Feeds in der Regel keine Benachrichtigungsfunktionalitäten (push) angeboten, sodass Interessenten Feeds regelmäßig auf Aktualisierungen überprüfen müssen (pull). Die Betrachtung entsprechender Techniken bildet den Kern der Arbeit und führte zu vier wissenschaftlichen Beiträgen.

Bewertungskriterien Der erste Schritt zur Auswahl eines geeigneten Algorithmus ist die Definition der Kriterien, anhand derer die Techniken zur Vorhersage von Aktualisierungen miteinander verglichen werden. Hierfür erfolgte die Analyse und anschließende Definition verschiedener Metriken zur Beurteilung der Qualität von Vorhersagen sowie die Definition eines zusammenfassenden Gütemaßes, welches den Vergleich von Algorithmen anhand eines einzigen Wertes ermöglicht.

Algorithmen Die in verwandten Domänen eingesetzten Algorithmen zur Vorhersage der Zeitpunkte von Aktualisierungen wurden aufgearbeitet und an die spezifischen Anforderungen der Domäne Feeds angepasst. Anschließend wurde aus den Komponenten unterschiedlicher Techniken der Algorithmus MAVSync entwickelt, der bereits ohne den Einsatz spezieller Konfigurationsparameter und ohne Trainingsphase im Durchschnitt bessere Vorhersagen trifft als die übrigen betrachteten Algorithmen.

Abfragespezifische Attribute Das Feed-Format RSS 2.0 stellt einige optionale Elemente bereit, die es Diensteanbietern ermöglichen, Zeiträume ohne Aktivitäten zu spezifizieren oder eine Zeitdauer anzugeben, nach deren Ablauf ein Feed erneut von der Quelle abzurufen ist, wenn er in einem Cache vorgehalten wird. Die Verbreitung sowie die Werte dieser Attribute wurden untersucht und empirisch gezeigt, dass die auf der partiellen Historie der Feeds basierende Vorhersage von Änderungen bereits bessere Ergebnisse erzielt, als wenn die von den Diensteanbietern bereitgestellten Werte der abfragespezifischen Attribute in die Berechnung einfließen.

Datensatz Mit dem Ziel aussagekräftiger Resultate erfolgten alle Simulationen auf einem breitgefächerten, realen Datensatz. Zur Erstellung wurden 200.000 Feeds über einen Zeitraum von vier Wochen regelmäßig abgefragt. In der Analyse des Datensatzes konnten wichtige Erkenntnisse über die Verteilung und Muster von Änderungsraten gewonnen werden, die bei der Generierung eines synthetischen Datensatzes verborgen geblieben wären. Der Datensatz wird der wissenschaftlichen Gemeinschaft frei zur Verfügung gestellt, um den Vergleich mit neuen Algorithmen zu erleichtern.

Abschließend werden die Einsatzszenarien der betrachteten Algorithmen rekapituliert. Anwender, die auf der Suche nach einem ihren Bedürfnissen entsprechenden Algorithmus sind, können von den Erkenntnissen der Arbeit folgendermaßen profitieren:

Entspricht ihr Ziel dem der Arbeit, d. h. die Verarbeitung einer großen Anzahl von Feeds mit breitgefächerten Änderungsraten, so ist MAVSync die beste Wahl, gefolgt von den Algorithmen IndHist und $\mathsf{FixLearned}_A$. Es gilt zu beachten, dass MAVSync adaptiv ist und die Zeitpunkte der Abfragen kontinuierlich an Variationen der Änderungsrate der Feeds anpasst, während $\mathsf{FixLearned}_A$ das Intervall nie anpasst und das von IndHist eingesetzte Modell regelmäßig neu zu trainieren ist, wobei die Autoren dies nicht näher spezifizieren [BGR06]. Es ist daher zu erwarten, dass MAVSync auch über lange Zeiträume mit variierenden Änderungsraten eine hohe Güte aufweist, während die von $\mathsf{FixLearned}_A$ und IndHist ohne erneutes Training sinkt.

Priorisiert ein Anwender eines der Kriterien Delay oder Recall, so kann er einerseits einen Algorithmus wählen, der aufgrund einer hohen Anzahl getätigter Abfragen hier sehr gute Resultate erreicht oder die obere Schranke entsprechend klein wählen und somit den maximal zulässigen Delay begrenzen. Das Ziel möglichst weniger Abfragen kann bei den betrachteten Algorithmen durch die Wahl eines verhältnismäßig großen Wertes der algorithmenspezifischen Parameter sowie durch Anhebung der unteren Schranke erreicht werden. Konkrete Werte sind von den Charakteristika der zu verarbeitenden Daten abhängig, sodass hier keine pauschalen Aussagen getroffen werden können.

Um die untersuchten Algorithmen auf andere Domänen anzuwenden sind mit hoher Wahrscheinlichkeit Anpassungen notwendig, da ein Großteil der betrachteten Techniken bereits für die Domäne Feeds modifiziert wurde. Als Beispiel seien die Domänen Web Crawling und Caching genannt. In beiden Domänen besteht die Notwendigkeit, die Zeitpunkte von Änderungen auf Basis einer beobachteten Historie eines Objekts vorherzusagen. Der entscheidende Unterschied generischer Web-Objekte zu Feeds ist die partielle Änderungshistorie, die ein Feed implizit mit sich führt. Die Auswertung der Historie ermöglicht das Erkennen von als Misses bezeichneten potentiellen Lücken in der Historie des Objekts. Ist nur der aktuelle Zustand eines Objekts beobachtbar und demzufolge ausschließlich eine binäre Entscheidung über erfolgte Änderungen möglich, so sind die auf die Verarbeitung von Feeds optimierten Algorithmen nur bedingt zur Vorhersage des nächsten Änderungszeitpunktes geeignet.

Im Gegenzug kann hingegen bestätigt werden, dass MAVSync die besten Resultate bei der Vorhersage der Änderungszeitpunkte von Feeds erzielt. Mit der Auswertung abfragespezifischer Feed-Attribute werden keine besseren Resultate erreicht. Die gewählten Metriken und die auf diesen basierende Güte sind gut geeignet, um die Algorithmen miteinander zu vergleichen, sodass alle drei aufgestellten Thesen erfüllt wurden.

Literaturverzeichnis

[AAB+09] R. Agrawal, A. Ailamaki, P. Bernstein, E. Brewer, M. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. Franklin, H. Garcia-Molina, J. Gehrke, L. Gruenwald, L. Haas, A. Halevy, J. Hellerstein, Y. Ioannidis, H. Korth, D. Kossmann, S. Madden, R. Magoulas, B. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. Szalay, und G. Weikum. The Claremont report on database research. *Communications of the ACM*, 52(6):56–65, June 2009.

[ABP10] G. Adam, C. Bouras, und V. Poulopoulos. Efficient extraction of news articles based on RSS crawling. In *International Conference on Machine and Web Intelligence (ICMWI)*, Algiers, Algeria, 2010.

[Ale11] Aletheia Konsortium. Aletheia – semantische Föderation umfassender Produktinformationen. `http://www.aletheia-projekt.de`, 2011. letzter Zugriff 15.12.2011.

[ATDE09] E. Adar, J. Teevan, S. Dumais, und J. Elsas. The web changes everything: understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 282–291, Barcelona, Spain, 2009. ACM.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, und D. Winer. Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`, 05 2000. letzter Zugriff 01.04.2011.

[BGR06] L. Bright, A. Gal, und L. Raschid. Adaptive pull-based policies for wide area data delivery. *ACM Transactions on Database Systems*, 31(2):631–671, 2006.

[BN97] P. Bernstein und E. Newcomer. *Principles of Transaction Processing, Second Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1997.

[Bra89] R. Braden. Requirements for Internet Hosts – Application and Support. `http://www.ietf.org/rfc/rfc1123.txt`, 1989. letzter Zugriff 01.04.2011.

[Bro02] J. v. d. Broecke. Pushlets - Whitepaper. `http://www.pushlets.com/doc/whitepaper-all.html`, August 2002. letzter Zugriff 27.09.2011.

[Cat92] V. Cate. Alex - a Global Filesystem. In *Proceedings of the 1992 USENIX File System Workshop*, Berkeley, CA, 1992. USENIX Assoc.

[CC09] W. Chen und P. Chundi. Trends Analysis of Topics Based on Temporal Segmentation. In *Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '09, pages 402–414, Linz, Austria, 2009. Springer-Verlag.

[CG99] J. Cho und H. Garcia-Molina. Synchronizing a database to Improve Freshness. Technical report, Department of Computer Science, Stanford, CA 94305, 10 1999.

[CG00] J. Cho und H. Garcia-Molina. Synchronizing a database to Improve Freshness. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 117–128, Dallas, Texas, USA, 2000. ACM.

[CG03a] J. Cho und H. Garcia-Molina. Effective Page Refresh Policies for Web Crawlers. *ACM Trans. Database Syst.*, 28:390–426, December 2003.

[CG03b] J. Cho und H. Garcia-Molina. Estimating Frequency of Change. *ACM Transactions on Internet Technology*, 3 (3):256–290, 2003.

[CH05] D. Chmielewski und G. Hu. A Distributed Platform for Archiving and Retrieving RSS Feeds. In *ICIS '05: Proc. of the Fourth Annual ACIS International Conference on Computer and Information Science*, pages 215–220, Washington, DC, USA, 2005. IEEE Computer Society.

[CHYZ09] J. Callan, M. Hoy, C. Yoo, und L. Zhao. The ClueWeb09 Dataset. `http://boston.lti.cs.cmu.edu/Data/clueweb09/`, 2009.

[CL98] P. Cao und C. Liu. Maintaining Strong Cache Consistency in the World Wide Web. *IEEE Trans. Comput.*, 47:445–457, April 1998.

[CN02] J. Cho und A. Ntoulas. Effective change detection using sampling. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 514–525, Hong Kong, China, 2002. Morgan Kaufmann Publishers Inc.

[Cro82] D. H. Crocker. Standard for the Format of ARPA Internet Text Messages. `http://www.ietf.org/rfc/rfc0822.txt`, 08 1982. letzter Zugriff 01.04.2011.

[Dad96] P. Dadam. *Verteilte Datenbanken und Client/Server-Systeme*. Springer, Heidelberg, 1996.

[DS05] M. Dürst und M. Suignard. Internationalized Resource Identifiers (IRIs). `http://www.ietf.org/rfc/rfc3987.txt`, 2005. letzter Zugriff 14.12.2011.

[DST99] V. Duvvuri, P. Shenoy, und R. Tewari. Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web. In *IEEE Transactions on Knowledge and Data Engineering*, pages 834–843, 1999.

[DTV09] A. Dasdan, K. Tsioutsiouliklis, und E. Velipasaoglu. Web Search Engine Metrics for Measuring User Satisfaction. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, Madrid, Spain, 2009. ACM. Tutorial slides.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, und T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, 1999. letzter Zugriff 01.04.2011.

[FGT08] D. Ford, C. Grimes, und E. Tassone. Keeping a Search Engine Index Fresh: Risk and optimality in estimating refresh rates for web pages. *Interface*, 08, 2008.

[FMNW03] D. Fetterly, M. Manasse, M. Najork, und J. Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 669–678, Budapest, Hungary, 2003. ACM.

[G2006] The BLOGS06 test collection. `http://ir.dcs.gla.ac.uk/test_collections/blogs06info.html`, 03 2006. letzter Zugriff 14.12.2011.

[G2008] The BLOGS08 test collection. `http://ir.dcs.gla.ac.uk/test_collections/blogs08info.html`, 04 2008. letzter Zugriff 14.12.2011.

[GC89] C. Gray und D. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. *SIGOPS Oper. Syst. Rev.*, 23:202–210, November 1989.

[GE01] A. Gal und J. Eckstein. Managing Periodically Updated Data in Relational Databases: A Stochastic Modeling Approach. *Journal of the ACM*, 48:2001, 2001.

[GF08] C. Grimes und D. Ford. Estimation of Web Page Change Rates. In *Proceedings of the Joint Statistical Meetings*, Denver, Colorado, USA, 2008.

[GKHK71] W. Gellert, H. Küstner, M. Hellwich, und H. Kästner. *Kleine Enzyklopädie Mathematik*. VEB Bibliographisches Institut Leipzig, 1971.

[GO08] C. Grimes und S. O'Brien. Microscale evolution of web pages. In *WWW '08: Proc. of the 17th international conference on World Wide Web*, pages 1149–1150, Beijing, China, 2008. ACM.

[Goe02] B. Goethals. Survey on frequent pattern mining. Technical report, University of Helsinki, 2002. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.791&rep=rep1&type=pdf`, letzter Zugriff 17.06.2011.

[GPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, und J. Widom. The TSIMMIS Approach to Mediation:

Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997. 10.1023/A:1008683107812.

[HC09] C. Hu und C. Chou. RSS Watchdog: An Instant Event Monitor on Real Online News Streams. In *CIKM '09: Proc. of the 18th ACM conference on Information and knowledge management*, pages 2097–2098, Hong Kong, China, 2009. ACM.

[HDD11] L. Hong, O. Dan, und B. Davison. Predicting popular messages in Twitter. In *Proceedings of the 20th international conference companion on World Wide Web*, WWW '11, pages 57–58, Hyderabad, India, 2011. ACM.

[HFAN98] G. Huck, P. Fankhauser, K. Aberer, und E. Neuhold. Jedi: Extracting and Synthesizing Information from the Web. In *COOPIS 98 Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, pages 32–43, New York, NY, USA, 1998. IEEE Computer Society.

[HGW+95] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, und Y. Zhuge. The Stanford Data Warehousing Project. Technical Report 1995-10, Stanford Infolab, 1995.

[HLKK08] Y. Han, S. Lee, J. Kim, und Y. Kim. A new aggregation policy for RSS services. In *CSSSIA '08: Proc. of the 2008 international workshop on Context enabled source and service selection, integration and adaptation*, pages 1–7, Beijing, China, 2008. ACM.

[INST04] A. Iyengar, E. Nahum, A. Shaikh, und R. Tewari. Web Caching, Consistency, and Content Distribution. In *[Sin04]*. Chapman & Hall CRC Press, 2004.

[JD08] T. Jörg und S. Deßloch. Towards generating ETL processes for incremental loading. In *Proceedings of the 2008 international symposium on Database engineering & applications*, IDEAS '08, pages 101–110, Coimbra, Portugal, 2008. ACM.

[JD09] T. Jörg und S. Deßloch. Formalizing ETL Jobs for Incremental Loading of Data Warehouses. In J. C. Freytag, T. Ruf, W. Lehner, und G. Vossen, editors, *Proceedings Datenbanksysteme in Business, Technologie und Web (BTW 2009), 13. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“*, volume 144 of *LNI*, pages 327–346. GI, 2009.

[Kin08] A. King. Average Web Page. `http://www.optimizationweek.com/reviews/average-web-page/`, May 2008. letzter Zugriff 08.03.2011.

[KN02] G. Klyne und C. Newman. Date and Time on the Internet: Timestamps. `http://www.ietf.org/rfc/rfc3339.txt`, 2002. letzter Zugriff 09.05.2011.

[Lan08] K. Lang. 20 Newsgroups Data Set. `http://people.csail.mit.edu/jrennie/20Newsgroups/`, 2008. letzter Zugriff 15.05.2011.

[LB08] J. Liu und L. Birnbaum. What Do They Think? Aggregating Local Views about News Events and Topics. In *WWW '08: Proc. of the 17th international conference on World Wide Web*, pages 1021–1022, Beijing, China, 2008. ACM.

[LCD01] D. Li, P. Cao, und M. Dahlin. WCIP: Web Cache Invalidation Protocol. Technical report, Cisco, Univ of Texas, http://tools.ietf.org/html/draft-danli-wrec-wcip-01, 2001. letzter Zugriff 25. Juni 2011.

[LG99] S. Lawrence und C. Giles. Accessibility of information on the web. *Nature*, 400:107–109, 7 1999.

[LH09] B. Lee und B. Hwang. An Efficient Method Predicting Update Probability on Blogs. In *Proceedings of the 2nd WSEAS International Conference on Engineering Mechanics, Structures and Engineering Geology*, Rodos Island, Greece, 2009. WSEAS.

[LIHZ08] B. Lee, J. Im, B. Hwang, und D. Zhang. Design of an RSS Crawler with Adaptive Revisit Manager. In *Proc. of 20th International Conference on Software Engineering and Knowledge Engineering–SEKE'08*, San Francisco, CA, USA, 2008.

[Lin11] J. Lindsay. Web Hooks. `http://wiki.webhooks.org`, 2011. letzter Zugriff 26.09.2011.

[LN07] U. Leser und F. Naumann. *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.* Dpunkt Verlag, 2007.

[LRS05] H. Liu, V. Ramasubramanian, und E. Sirer. Client behavior and feed characteristics of RSS, a publish- subscribe sysem for web micronews. In *Proc. of he 5th ACM SIGCOMM conference on Internet Measurement*, pages 29–34, Philadelphia, PA, USA, 2005. USENIX Association.

[LSC+07] Y. Lin, H. Sundaram, Y. Chi, J. Tatemura, und B. Tseng. Splog Detection Using Self-similarity Analysis on Blog Temporal Dynamics. In *Proceedings of the 3rd international workshop on Adversarial information retrieval on the web*, AIRWeb '07, pages 1–8, Banff, Alberta, Canada, 2007. ACM.

[LWC08] T. Lee, E. Wang, und A. Chen. Mining Serial Episode Rules with Time Lags over Multiple Data Streams. In *Proceedings of the 10th international conference on Data Warehousing and Knowledge Discovery*, DaWaK '08, pages 227–240, Turin, Italy, 2008. Springer-Verlag.

[LWLC02] J. Lee, K. Whang, B. Lee, und J. Chang. An Update-Risk Based Approach to TTL Estimation in Web Caching. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering*, WISE '02, pages 21–29, Singapore, 2002. IEEE Computer Society.

[MAAH09] J. Madhavan, L. Afanasiev, L. Antova, und A. Halevy. Harnessing the Deep Web: Present and Future. In *4th Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, January 2009.

[Mat05] N. Matloff. Estimation of internet file-access/modification rates from indirect data. *ACM Trans. Model. Comput. Simul.*, 15:233–253, July 2005.

[MCD+07] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, und C. Yu. Web-Scale Data Integration: You can afford to Pay as You Go. In *Third Biennial Conference on Innovative Data Systems Research*, pages 342–350, Asilomar, CA, USA, 01 2007.

[Mei06] J. O. Meiert. RSS 2.0 und Atom 1.0 im Vergleich (Sam Ruby et al.). `http://meiert.com/de/publications/translations/intertwingly.net/rss-2.0-and-atom-1.0/`, 11 2006. letzter Zugriff 01.04.2011.

[Mil92] D. L. Mills. Network Time Protocol (Version 3)—Specification, Implementation and Analysis. `http://www.ietf.org/rfc/rfc1305.txt`, 1992. letzter Zugriff 14.06.2011.

[MKD+02] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. Hoff, und D. Hellerstein. Delta encoding in HTTP. `http://www.ietf.org/rfc/rfc3229.txt`, 2002.

[MKK+08] J. Madhavan, D. Ko, Ł. Kot, V. Ganapathy, A. Rasmussen, und A. Halevy. Google's Deep Web crawl. In *Proceedings of the 34th international conference on Very Large Data Bases*, VLDB '08, pages 1241–1252, Auckland, New Zealand, August 2008.

[MO06] C. Macdonald und I. Ounis. The TREC Blogs06 Collection: Creating and Analysing a Blog Test Collection. Technical report, Department of Computing Science, University of Glasgow, 2006.

[MRS08] C. D. Manning, P. Raghavan, und H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 1 edition, July 2008.

[NS05] M. Nottingham und R. Sayre. The Atom Syndication Format. `http://www.ietf.org/rfc/rfc4287.txt`, 2005. letzter Zugriff 01.04.2011.

[OOW93] E. J. O'Neil, P. E. O'Neil, und G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, SIGMOD '93, pages 297–306, Washington, D.C., USA, 1993. ACM.

[OP08] C. Olston und S. Pandey. Recrawl scheduling based on information longevity. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 437–446, Beijing, China, 2008. ACM.

[OV11] M. Ozsu und P. Valduriez. *Principles of Distributed Database Systems (Third Edition)*. Prentice Hall, 3 edition, January 2011.

[PM10] R. Pantos und W. May. HTTP Live Streaming draft-pantos-http-live-streaming-04. `http://tools.ietf.org/html/draft-pantos-http-live-streaming-04`, June 2010. letzter Zugriff 27.09.2011.

[Pub11] PubSubHubbub Projektwebseite. `http://code.google.com/p/pubsubhubbub/`, 2011. letzter Zugriff 28.09.2011.

[RMP+07] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, und M. Welsh. Cobra: Content-based filtering and aggregation of blogs and RSS feeds. In *Proc. of the Symposium on Networked Systems Design and Implementation, Boston, MA*, 2007.

[RP08] C. Raïssi und M. Plantevit. Mining Multidimensional Sequential Patterns over Data Streams. In T. M. N. Il-Yeol Song, Johann Eder, editor, *Data Warehousing and Knowledge Discovery, 10th International Conference, DaWaK 2008*, Lecture Notes in Computer Science, pages 263–272, Turin, Italy, September 2008. Springer.

[RPS06] V. Ramasubramanian, R. Peterson, und E. Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, USA, 2006.

[RS02] M. Rabinovich und O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2002.

[RSW02] T. Rose, M. Stevenson, und M. Whitehead. The Reuters Corpus Volume 1 – from Yesterday's News to Tomorrow's Language Resources. In *Language Resources and Evaluation*, 2002.

[RUM+11] S. Reichert, D. Urbansky, K. Muthmann, P. Katz, M. Wauer, und A. Schill. Feeding the World: A Comprehensive Dataset and Analysis of a Real World Snapshot of Web Feeds. In *13th International Conference on Information Integration and Web-based Applications and Services (iiWAS2011)*, Ho Chi Minh City, Vietnam, 12 2011. ACM.

[Rus06] A. Russell. Comet: Low Latency Data for the Browser. http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/, March 2006. letzter Zugriff 27.09.2011.

[SA01] A. Sahuguet und F. Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowl. Eng.*, 36:283–316, March 2001.

[SCC07] K. Sia, J. Cho, und H. Cho. Efficient Monitoring Algorithm for Fast News Alerts. *IEEE Transactions on Knowledge and Data Engineering*, 19:950–961, July 2007.

[SCH+07] K. Sia, J. Cho, K. Hino, Y. Chi, S. Zhu, und B. Tseng. Monitoring RSS Feeds Based on User Browsing Pattern. In *International Conference on Weblogs and Social Media (ICWSM)*, Boulder, Colorado, USA, 2007.

[Sin04] M. P. Singh, editor. *Practical Handbook of Internet Computing*. Chapman & Hall CRC Press, Baton Rouge, 2004.

[Sin07] S. Singh. Estimating the rate of web page updates. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2874–2879, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[Six10a] Six Apart Ltd. Real-Time Updates. `http://developer.typepad.com/api/real-time-updates.html`, 2010. letzter Zugriff 27.09.2011.

[Six10b] Six Apart Ltd. Update Stream For Developers. `http://www.sixapart.com/labs/update/developers/`, 2010. letzter Zugriff 27.09.2011.

[SMPD05] D. Sandler, A. Mislove, A. Post, und P. Druschel. FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification. In M. Castro und R. van Renesse, editors, *4th Annual International Workshop on Peer-to-Peer Systems (IPTPS'05)*, volume 3640 of *Lecture Notes in Computer Science*, pages 141–151, Ithaca, NY, USA, 2005. Springer Berlin / Heidelberg.

[TK98] H. Taylor und S. Karlin. *An introduction to stochastic modeling*. Academic Press, Orlando, 3rd edition, 1998.

[TL09] M. Thiele und W. Lehner. Evaluation of Load Scheduling Strategies for Real-Time Data Warehouse Environments. In M. Castellanos, U. Dayal, und R. Miller, editors, *BIRTE*, volume 41 of *Lecture Notes in Business Information Processing*, pages 84–99. Springer, 2009.

[UMKS11] D. Urbansky, K. Muthmann, L. Kreisz, und A. Schill. Areca: Online Comparison of Research Results. In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media (ICWSM-11)*, Barcelona, Spain, 2011. The AAAI Press.

[URM+11] D. Urbansky, S. Reichert, K. Muthmann, D. Schuster, und A. Schill. An Optimized Web Feed Aggregation Approach for Generic Feed Types. In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media (ICWSM-11)*, Barcelona, Spain, 2011. The AAAI Press.

[Vog04] W. Vogels. Feed Analysis: The Age of Feeds. `http://www.allthingsdistributed.com/historical/archives/000447.html`, April 2004. letzter Zugriff November 2011.

[WHSS10] M. Walther, L. Hähne, D. Schuster, und A. Schill. Locating and Extracting Product Specifications from Producer Websites. In *ICEIS 2010 - Proceedings of the 12th International Conference on Enterprise Information Systems, SAIC*, volume 4, pages 13–22, Funchal, Madeira, Portugal, 2010.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25:38–49, March 1992.

[Win72] R. Winkler. *An introduction to Bayesian inference and decision.* Holt, Rinehart and Winston, Inc., 1972.

[Win99] D. Winer. XML-RPC Specification. `http://www.xmlrpc.com/spec`, 06 1999. letzter Zugriff 01.04.2011.

[Win03a] D. Winer. RSS 2.0 Specification. `http://cyber.law.harvard.edu/rss/rss.html`, 07 2003. letzter Zugriff 01.04.2011.

[Win03b] D. Winer. SOAP Meets RSS. `http://cyber.law.harvard.edu/rss/soapMeetsRss.html`, 07 2003. letzter Zugriff 01.04.2011.

[YCLL00] J. Yuen, E. Chan, K. Lam, und H. Leung. Cache invalidation scheme for mobile computing systems with real-time data. *SIGMOD Rec.*, 29:34–39, December 2000.

[YWLW10] M. Yang, H. Wang, L. Lim, und M. Wang. Optimizing content freshness of relations extracted from the web using keyword search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 819–830, Indianapolis, Indiana, USA, 2010. ACM.

[ZAL08] Y. Zhu, L. An, und S. Liu. Data Updating and Query in Real-Time Data Warehouse System. In *International Conference on Computer Science and Software Engineering*, volume 5, pages 1295–1297, Wuhan, China, December 2008.

[ZK01] T. Zurek und K. Kreplin. SAP Business Information Warehouse—from data warehousing to an e-business platform. In *17th International Conference on Data Engineering*, pages 388–390, Heidelberg, Germany, 2001. IEEE Computer Society.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

APE Abfragen pro Eintrag 126
API Application Programming Interface 21
BBC British Broadcasting Corporation 81
CDC Change Data Capture 25
CDN Content Distribution Network 50
CEST Central European Summer Time 119
CMS Content-Management-System 11
CNN Cable News Network 81
DSMS Data Stream Management Systeme 27
DWH Data Warehouse 12
FIFO First-In, First-Out 23
FPM Frequent Pattern Mining 27
GB Gigabyte 96
GUID Globally Unique Identifier 32
HTML Hyper Text Markup Language 31
HTTP Hyper Text Transfer Protocol 16
IoT Internet of Things 14
IRI Internationalized Resource Identifier 34
kB Kilobyte 42
LRU Least Recently Used 53
MB Megabyte 94
NER Named Entity Recognition 14
NLP Natural Language Processing 14
NTP Network Time Protocol 37
MLE Maximum Likelihood Estimator 45
MDM Master Data Management 13
P2P Peer-to-Peer 64
PC Primary Copy 27

A. Detaillierte Evaluationsergebnisse

Die in Abschnitt 4.4.1 verkürzt präsentierten Resultate der Parameterbestimmungen werden auf den nächsten Seiten noch einmal vollständig dargelegt. Die gewählte Darstellungsweise ermöglicht es dem Leser, einerseits die Durchschnittswerte $\overline{D}_F$, $\overline{APE}_F$ und $\overline{R}_F$ sowie $\overline{D}_E$, $\overline{APE}_E$ und $\overline{R}_E$ miteinander zu vergleichen und andererseits stets die sich ergebenden Verläufe der normierten Werte zu überblicken.

A.1. Adaptive TTL Parameterbestimmung

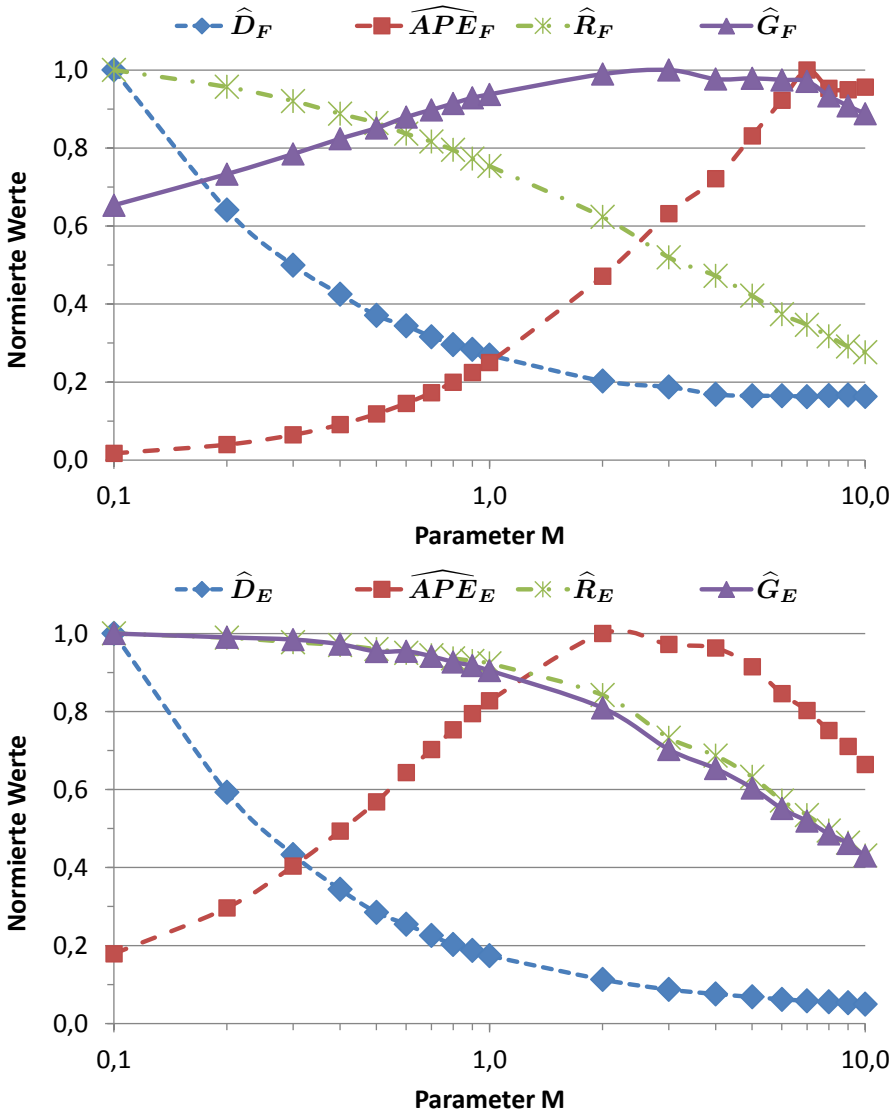

Abbildung A.1.: Parameterbestimmung Adaptive TTL, Betrachtung der normierten Größen $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ sowie der Güte $\widehat{G}$. Vergleich des Durchschnitts über alle Feeds (oben) und Einträge (unten).

AdaptiveTTL	absolute Werte			auf Besten normiert			
M	$\overline{D}_E$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
0,1	12h:24min	26,061	0,940	1,000	0,017	1,000	0,653
0,2	19h:20min	11,307	0,899	0,641	0,040	0,957	0,733
0,3	24h:49min	6,916	0,865	0,500	0,065	0,921	0,785
0,4	29h:09min	4,907	0,835	0,425	0,091	0,888	0,824
0,5	33h:25min	3,778	0,813	0,371	0,118	0,864	0,851
0,6	36h:03min	3,075	0,786	0,344	0,145	0,837	0,879
0,7	39h:11min	2,593	0,767	0,316	0,172	0,816	0,898
0,8	41h:52min	2,239	0,747	0,296	0,200	0,795	0,914
0,9	43h:39min	1,989	0,726	0,284	0,225	0,773	0,929
1,0	46h:03min	1,788	0,708	0,269	0,250	0,754	0,938
2,0	61h:08min	0,948	0,586	0,203	0,472	0,623	0,989
3,0	66h:10min	0,707	0,488	0,187	0,632	0,520	1,000
4,0	73h:36min	0,620	0,444	0,168	0,721	0,472	0,977
5,0	75h:16min	0,538	0,396	0,165	0,831	0,421	0,979
6,0	75h:05min	0,485	0,352	0,165	0,923	0,374	0,975
7,0	76h:15min	0,447	0,326	0,163	1,000	0,347	0,971
8,0	74h:29min	0,469	0,298	0,166	0,953	0,317	0,934
9,0	74h:17min	0,471	0,274	0,167	0,949	0,291	0,908
10,0	76h:01min	0,468	0,260	0,163	0,956	0,277	0,888

Tabelle A.1.: Parameterbestimmung Adaptive TTL, Durchschnitt über alle Feeds.

AdaptiveTTL	Durchschnittswerte			auf Besten normiert			
M	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
0,1	17min	4,147	0,844	1,000	0,179	1,000	1,000
0,2	29min	2,504	0,834	0,593	0,297	0,989	0,990
0,3	40min	1,840	0,825	0,433	0,404	0,978	0,984
0,4	50min	1,505	0,819	0,344	0,493	0,970	0,972
0,5	1h:00min	1,307	0,810	0,285	0,568	0,960	0,954
0,6	1h:08min	1,155	0,802	0,255	0,643	0,950	0,954
0,7	1h:17min	1,058	0,796	0,226	0,702	0,944	0,942
0,8	1h:25min	0,986	0,790	0,202	0,753	0,966	0,927
0,9	1h:32min	0,935	0,784	0,187	0,795	0,930	0,918
1,0	1h:39min	0,897	0,779	0,174	0,828	0,923	0,905
2,0	2h:33min	0,743	0,710	0,113	1,000	0,842	0,809
3,0	3h:18min	0,764	0,618	0,087	0,972	0,733	0,703
4,0	3h:48min	0,771	0,579	0,076	0,963	0,687	0,654
5,0	4h:13min	0,812	0,534	0,068	0,915	0,632	0,604
6,0	4h:38min	0,878	0,482	0,062	0,846	0,572	0,551
7,0	4h:56min	0,925	0,451	0,058	0,803	0,534	0,519
8,0	5h:12min	0,988	0,417	0,055	0,752	0,495	0,486
9,0	5h:23min	1,046	0,391	0,053	0,710	0,463	0,462
10,0	5h:48min	1,118	0,363	0,050	0,664	0,430	0,429

Tabelle A.2.: Parameterbestimmung Adaptive TTL, Durchschnitt über alle Einträge.

A.2. IndHist Parameterbestimmung

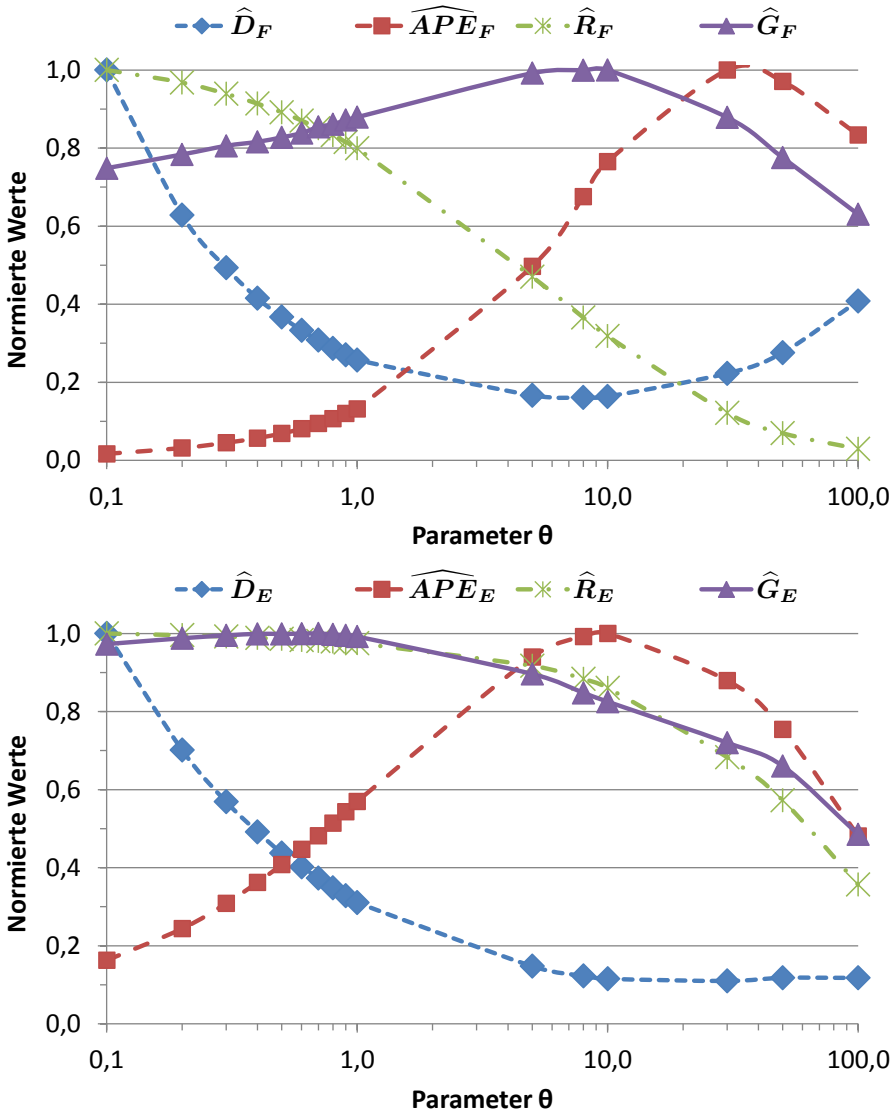

Abbildung A.2.: Parameterbestimmung IndHist, Betrachtung der normierten Größen $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ sowie der Güte $\widehat{G}$. Vergleich des Durchschnitts über alle Feeds (oben) und Einträge (unten).

IndHist	Durchschnittswerte			auf Besten normiert			
θ	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
0,1	11h:39min	15,522	0,952	1,000	0,017	1,000	0,749
0,2	18h:32min	8,225	0,921	0,628	0,032	0,968	0,784
0,3	23h:36min	5,762	0,893	0,493	0,045	0,939	0,806
0,4	28h:02min	4,547	0,870	0,415	0,057	0,914	0,816
0,5	31h:41min	3,764	0,848	0,367	0,069	0,892	0,828
0,6	34h:56min	3,206	0,828	0,333	0,081	0,871	0,838
0,7	37h:47min	2,736	0,809	0,308	0,095	0,851	0,854
0,8	40h:36min	2,437	0,793	0,287	0,107	0,833	0,861
0,9	43h:07min	2,164	0,776	0,270	0,120	0,816	0,872
1,0	45h:19min	1,968	0,761	0,257	0,132	0,800	0,879
5,0	69h:49min	0,523	0,447	0,167	0,497	0,470	0,992
8,0	72h:19min	0,384	0,348	0,161	0,676	0,366	0,999
10,0	71h:07min	0,339	0,303	0,164	0,765	0,318	1,000
30,0	52h:15min	0,260	0,116	0,223	1,000	0,122	0,879
50,0	42h:11min	0,268	0,066	0,276	0,971	0,069	0,776
100,0	28h:31min	0,312	0,028	0,408	0,834	0,029	0,630

Tabelle A.3.: Parameterbestimmung IndHist, Durchschnitt über alle Feeds.

IndHist	Durchschnittswerte			auf Besten normiert			
θ	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
0,1	18min	3,284	0,979	1,000	0,163	1,000	0,9729
0,2	25min	2,191	0,974	0,702	0,244	0,996	0,9879
0,3	31min	1,731	0,971	0,569	0,309	0,992	0,9952
0,4	36min	1,475	0,968	0,492	0,362	0,989	0,9992
0,5	40min	1,309	0,965	0,437	0,408	0,986	0,9988
0,6	44min	1,195	0,962	0,402	0,447	0,983	0,9998
0,7	47min	1,108	0,960	0,373	0,482	0,981	1,0000
0,8	50min	1,038	0,958	0,348	0,514	0,979	0,9976
0,9	53min	0,983	0,956	0,328	0,544	0,977	0,9953
1,0	56min	0,938	0,954	0,311	0,570	0,975	0,9921
5,0	1h:59min	0,568	0,898	0,148	0,940	0,917	0,8964
8,0	2h:22min	0,538	0,865	0,123	0,992	0,883	0,8480
10,0	2h:31min	0,534	0,842	0,115	1,000	0,861	0,8253
30,0	2h:39min	0,607	0,668	0,110	0,880	0,683	0,7203
50,0	2h:28min	0,708	0,560	0,118	0,754	0,573	0,6605
100,0	2h:29min	1,110	0,349	0,118	0,481	0,357	0,4855

Tabelle A.4.: Parameterbestimmung IndHist, Durchschnitt über alle Einträge.

A.3. LIHZ Parameterbestimmung

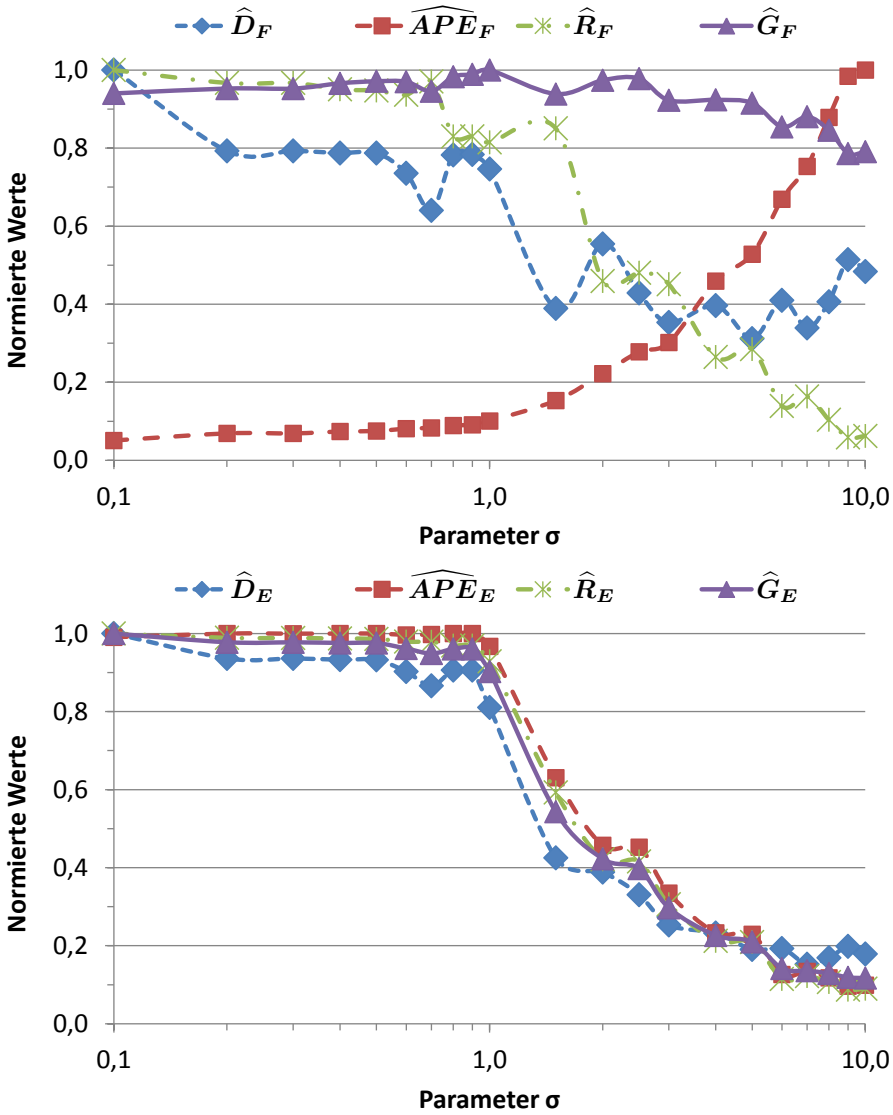

Abbildung A.3.: Parameterbestimmung LIHZ, Betrachtung der normierten Größen $\widehat{D}$, $\widehat{APE}$ und $\widehat{R}$ sowie der Güte $\widehat{G}$. Vergleich des Durchschnitts über alle Feeds (oben) und Einträge (unten).

LIHZ	Durchschnittswerte			auf Besten normiert			
σ	$\overline{D}_F$	$\overline{APE}_F$	$\overline{R}_F$	$\widehat{D}_F$	$\widehat{APE}_F$	$\widehat{R}_F$	$\widehat{G}_F$
0,1	32h:00min	1,729	0,835	1,000	0,051	1,000	0,940
0,2	40h:21min	1,274	0,808	0,793	0,069	0,967	0,953
0,3	40h:21min	1,274	0,807	0,793	0,069	0,967	0,953
0,4	40h:38min	1,193	0,794	0,788	0,074	0,950	0,966
0,5	40h:39min	1,169	0,791	0,787	0,075	0,947	0,971
0,6	43h:30min	1,085	0,782	0,736	0,081	0,937	0,970
0,7	49h:57min	1,057	0,811	0,641	0,083	0,971	0,946
0,8	40h:53min	0,983	0,693	0,783	0,089	0,830	0,983
0,9	40h:51min	0,966	0,693	0,783	0,091	0,830	0,989
1,0	42h:53min	0,874	0,680	0,746	0,101	0,815	1,000
1,5	82h:16min	0,575	0,710	0,389	0,153	0,850	0,939
2,0	57h:43min	0,396	0,384	0,555	0,222	0,459	0,974
2,5	74h:40min	0,316	0,402	0,429	0,278	0,481	0,978
3,0	90h:37min	0,291	0,377	0,353	0,302	0,451	0,923
4,0	80h:44min	0,191	0,221	0,396	0,459	0,265	0,924
5,0	102h:34min	0,166	0,238	0,312	0,528	0,285	0,915
6,0	78h:02min	0,131	0,116	0,410	0,669	0,139	0,855
7,0	94h:25min	0,117	0,137	0,339	0,753	0,164	0,880
8,0	78h:48min	0,100	0,087	0,406	0,879	0,104	0,846
9,0	62h:15min	0,089	0,049	0,514	0,985	0,059	0,786
10,0	66h:10min	0,088	0,052	0,484	1,000	0,063	0,791

Tabelle A.5.: Parameterbestimmung LIHZ, Durchschnitt über alle Feeds.

LIHZ	Durchschnittswerte			auf Besten normiert			
σ	$\overline{D}_E$	$\overline{APE}_E$	$\overline{R}_E$	$\widehat{D}_E$	$\widehat{APE}_E$	$\widehat{R}_E$	$\widehat{G}_E$
0,1	12h:04min	2,594	0,163	1,000	0,991	1,000	1,000
0,2	12h:53min	2,570	0,161	0,936	1,000	0,989	0,978
0,3	12h:53min	2,570	0,161	0,936	1,000	0,989	0,978
0,4	12h:56min	2,570	0,161	0,933	1,000	0,987	0,976
0,5	12h:56min	2,569	0,161	0,932	1,000	0,986	0,975
0,6	13h:22min	2,578	0,160	0,902	0,997	0,979	0,962
0,7	13h:56min	2,575	0,160	0,866	0,998	0,979	0,949
0,8	13h:19min	2,569	0,158	0,906	1,000	0,969	0,961
0,9	13h:18min	2,570	0,158	0,907	1,000	0,969	0,961
1,0	14h:53min	2,657	0,152	0,811	0,967	0,929	0,903
1,5	28h:23min	4,076	0,097	0,425	0,630	0,593	0,543
2,0	31h:04min	5,615	0,069	0,388	0,458	0,422	0,423
2,5	36h:29min	5,673	0,068	0,331	0,453	0,416	0,398
3,0	47h:37min	7,676	0,050	0,253	0,335	0,306	0,297
4,0	51h:53min	11,028	0,035	0,233	0,233	0,212	0,226
5,0	63h:23min	11,187	0,034	0,190	0,230	0,209	0,210
6,0	62h:36min	20,286	0,019	0,193	0,127	0,115	0,142
7,0	79h:08min	19,147	0,020	0,152	0,134	0,122	0,136
8,0	71h:25min	21,822	0,017	0,169	0,118	0,107	0,129
9,0	60h:50min	26,703	0,014	0,198	0,096	0,087	0,119
10,0	67h:23min	25,946	0,015	0,179	0,099	0,090	0,117

Tabelle A.6.: Parameterbestimmung LIHZ, Durchschnitt über alle Einträge.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Dresden, den 13.03.2012