LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTERARBEIT

Organisation von Bloom-Filtern zur effizienten k-nächste-Nachbarn-Suche in kontextzentrischen sozialen Netzen

Judith Greif

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTERARBEIT

Organisation von Bloom-Filtern zur effizienten k-nächste-Nachbarn-Suche in kontextzentrischen sozialen Netzen

Judith Greif

Aufgabenstellerin:	Prof. Dr. Claudia Linnhoff-Popien
Betreuer:	Mirco Schönfeld Dr. Martin Werner
Abgabetermin:	26. Juli 2016

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 26. Juli 2016

. . . . . .
(Unterschrift der Kandidatin)

Abstract

Hier kommt der Abstract hin. Hier kommt der Abstract hin.

Inhaltsverzeichnis

1 Einleitung

Die digitale Kommunikation hat im letzten Jahrzehnt einen rapiden Wandel erlebt. Soziale Online-Netzwerke[1] haben an Bedeutung gewonnen und zu neuen Kommunikationsmustern im Internet geführt. Sofortnachrichtendienste und Instant-Messenger ersetzen zunehmend Kommunikationsformen wie SMS und Telefonie. Das Smartphone hat das Mobiltelefon als mobiles Endgerät fast vollständig abgelöst. Feststehende Desktop-Rechner mit einem gleich bleibenden Netzzugang sind außerhalb von Firmen und Bildungseinrichtungen rückläufig. Dagegen sind Notebooks und Tablets weiterhin auf Erfolgszug[2].

Die Kommunikation im Internet ist jedoch nach wie vor Ende-zu-Ende- beziehungsweise adressbasiert. Das spiegelt sich im Aufbau der bestehenden sozialen Online-Netzwerke wider: Kommunikation basiert darin auf Online-Freundschaft. Mobilität und spezifischer Kontext der Mitglieder werden kaum berücksichtigt. In der Realität verlieren die Webbrowser-Schnittstellen der sozialen Netzwerke jedoch an Bedeutung. So veröffentlichte Facebook 2014 eine Studie zum Nutzerverhalten von US-Bürgern in einer Multigeräte-Welt[3]. Danach nutzten 60% der Erwachsenen in den USA täglich mindestens zwei Endgeräte, knapp 25% sogar drei Geräte. Mehr als 40% begannen den Tag mit einem Gerät und beendeten ihn mit einem anderen. Das Smartphone ist dabei das Gerät, das am häufigsten mitgenommen wird und eine zentrale Rolle in der Kommunikation per E-Mail und in sozialen Netzen einnimmt. Das Szenario, in dem Alice vor ihrem Desktop-Rechner zu Hause oder im Büro sitzt und über die Webbrowser-Schnittstelle Nachrichten an ihren Facebook-Freund Bob schreibt, gehört demnach der Vergangenheit an. Stattdessen verwendet Alice wohl eher ein Tablet, ein Notebook und ein Smartphone und kommuniziert mit Bob je nach Aufenthaltsort und Kontext ganz unterschiedlich.

So stellt sich die Frage nach einer Neuorientierung der sozialen Online-Netze: Nicht nur in der praktischen Umsetzung, also durch Schaffung unterschiedlicher Schnittstellen und neuer Funktionalitäten, sondern im Sinne eines tatsächlichen Paradigmenwechsels. Ein kontextzentrisches soziales Netz basiert in seiner Struktur nicht auf Ende-zu-Ende-Kommunikation, adressbasiertem Routing und einem gleich bleibenden Netzzugang. Kommunikation beruht allein auf Kontext-Ähnlichkeit statt auf virtueller Freundschaft. Diese Überlegungen sind z.B. in das soziale Online-Netz AMBIENCE eingeflossen. Nachrichten werden darin auf Grund von zeitlicher und räumlicher Ähnlichkeit ausgetauscht, Sender und Empfänger bleiben weitgehend anonym. Ein solches Netz erfordert eine neue Kommunikationsstruktur. Nachrichten werden nicht aktiv von einem Sender für einen spezifischen Empfänger verfasst und an ihn verschickt. Stattdessen kann ein Sender eine Nachricht verfassen und z.B. an einem WiFi-Access Point hinterlegen. Mitglieder des Netzwerks, die sich in der Nähe des Access Points aufhalten, können die dort vorhan-

[1] Der englische Begriff hierfür lautet *Online Social Network (OSN)*. Die Begriffe *Soziales Netz*, *Soziales Online-Netz* und *Soziales Online-Netzwerk* werden im Folgenden synonym verwendet.

[2] Laut Analysen der Marktforschungsinstitute Gartner und IDC, vgl. z.B. `http://www.golem.de/news/pc-markt-absatz-von-pcs-geht-weiter-erheblich-zurueck-1601-118505.html`.

[3] Vgl. `https://www.facebook.com/business/news/Finding-simplicity-in-a-multi-device-world`.

denen Nachrichten mit gezielten Anfragen durchsuchen und die zum jeweiligen Kontext ähnlichsten Nachrichten abrufen.

Damit stellt sich die Frage: Wie lassen sich die Nachrichten an einem Host, also z.B. an einem WiFi-Access Point, so organisieren, dass die k ähnlichsten Nachrichten möglichst schnell und effizient gefunden werden? Wenn das soziale Netz wachsen und über den Status eines Prototypen hinaus erfolgreich sein soll, ist das von entscheidender Bedeutung. Mengentheoretisch betrachtet handelt es sich dabei um das Problem der k-nächsten-Nachbarn-Suche, die zu einer Anfrage die k ähnlichsten Elemente einer Menge, hier bestehend aus den Nachrichten an einem Host, finden soll.

Damit verknüpft ist die Frage nach der Nachrichtenform. Wie können Multimedia-Nachrichten wie Bilder, Textdateien oder Links effizient, einheitlich und sicher vor unbefugtem Zugriff hinterlegt und verschickt werden? Zudem muss die Ähnlichkeit oder Unähnlichkeit von Nachrichten ermittelt werden können, d.h. der k-nächste-Nachbarn-Suche muss ein Ähnlichkeitsmaß zu Grunde liegen. AMBIENCE verwendet dazu eine Bloom-Filter-Konstruktion. Eine Nachricht wird als Menge von Zeichenketten aufgefasst, die mit geeigneten Hashfunktionen in ein Bit-Array fester Länge eingefügt werden. Ähnlichkeit zwischen Nachrichten ist damit als Ähnlichkeit zwischen Bloom-Filtern definiert. Nachrichten werden in Form von Bloom-Filtern kodiert, gespeichert und verglichen. Als Ähnlichkeitsmaß dient die Jaccard-Distanz, mit der sich die Ähnlichkeit von Mengen beschreiben lässt.

Die folgende Arbeit behandelt daher die Organisation von Bloom-Filtern zur effizienten k-nächste-Nachbarn-Suche in kontextzentrischen sozialen Netzen. Das folgende Kapitel 2 gibt einen Überblick über mengentheoretische und probabilistische Grundlagen, verwendete Datenstrukturen und ihre Nutzung in AMBIENCE. Kapitel 3 gibt einen Überblick über verwandte Arbeiten und Fragestellungen. Das entwickelte Verfahren wird anschließend in Kapitel 4 dargestellt. Kapitel 5 vergleicht die Implementierung mit dem bisherigen, nicht optimierten Ansatz. Im abschließenden Kapitel 6 wird ein Fazit gezogen und auf mögliche zukünftige Arbeiten eingegangen.

2 Hintergrund

Im Folgenden wird das Konzept des *kontextzentrischen sozialen Netzes* erläutert, das dieser Arbeit zu Grunde liegt. Anschließend werden mengentheoretische und probabilistische Grundlagen und Verfahren sowie Daten- und Indexstrukturen dargestellt, die Eingang in diese Arbeit gefunden haben. Es wird erläutert, inwiefern sie für das soziale Online-Netz AMBIENCE relevant sind oder darin verwendet werden.

2.1 Kontextzentrische soziale Netze

Wie eingangs dargestellt lässt sich mit Zunahme der mobilen Endgeräte und dem Erfolgszug des Smartphones eine Tendenz beobachten, die in bestehenden sozialen Online-Netzwerken wenig abgebildet wird: Weg von adressbasiertem Routing und Ende-zu-Ende-Kommunikation hin zu *context-awareness* und *information-centric networking*.

Der Begriff context-awareness wurde von Schilit et al. 1994 geprägt und bezeichnet die Nutzung von Kontextinformationen als Informationsquelle für Anwendungen und Netzwerke[1]. Information-centric networking (ICN) bezeichnet ein neuartiges Konzept für Netzwerke, die nicht auf Ende-zu-Ende- oder Sender/Empfänger-Kommunikation basieren, sondern auf den im Netzwerk vorhandenen Informationen[2]. Der Begriff *Kontext* im Zusammenhang mit interaktiven Anwendungen wurde bereits in den 90er Jahren geprägt. In der klassischen Definition von Dey und Abowd bedeutet Kontext

> [...] any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application including the user and application themselves[3].

Diese Definition wird für soziale Online-Netze eingegrenzt:

> Context is any information that can be used to infer aspects of the surroundings of an entity in a way, in which some applications may have interest. Surroundings include all information that could possibly impact the behaviour of the entity[4].

Ein kontextzentrisches soziales Netz ist also ein soziales Online-Netz, das auf Kontext-Variablen wie Ort und Zeit als Informationsquellen basiert und in der Regel dezentral organisiert ist (z.B. mit einer Peer-to-Peer- statt einer Client/Server-Architektur). Kommunikation beruht darin allein auf Kontext-Ähnlichkeit, nicht auf Online-Freundschaft:

[1] Vgl. [SAW94].
[2] Vgl. [ADI+12] für eine ausführliche Darstellung.
[3] [DA99]: 306f..
[4] [WDS15]: 2.

> A context-centric online social network is an online social network in which the edges of the social graph are defined from context information and context matching algorithms. An edge between two profiles exists for a fixed information object if and only if the two profiles share the relevant context as defined by the publisher[5].

2.1.1 AMBIENCE

AMBIENCE ist ein soziales Online-Netzwerk, das 2015 als Prototyp implementiert wurde und sich an diesem neuen Paradigma orientiert. Die vorliegende Arbeit hat das Ziel, einen spezifischen Aspekt des Netzwerks zu optimieren, nämlich die Organisation der Nachrichten an einem Host für k-nächste-Nachbarn-Anfragen. Nachrichten und Anfragen werden in AMBIENCE als Bloom-Filter kodiert. Bei einer Anfrage wird also ein Anfrage-Filter mit einer Menge von Bloom-Filtern verglichen, die an einem Host gespeichert sind. Aktuell werden die Filter dort einfach als unsortierte Liste gespeichert. Die Laufzeit für k-nächste-Nachbarn-Anfragen an einen Host mit n Filtern liegt damit in $O(n^2)$. Dieses Laufzeitverhalten gilt es zu optimieren, wenn das Netzwerk wachsen und über den Status eines Prototypen hinaus erfolgreich sein soll.

2.2 Bloom-Filter

Ein Bloom-Filter ist eine probabilistische Datenstruktur zur Beschreibung von Mengen, die in der ursprünglichen From 1970 von Burton H. Bloom eingeführt wurde[6]. Er besteht aus einem Bit-Array der festen Länge m, dessen Elemente zunächst alle auf 0 gesetzt sind. Das Einfügen von Informationsobjekten basiert auf der Berechnung einer festen Anzahl k unabhängiger Hashfunktionen, die positive Werte kleiner als m annehmen. Soll ein Objekt in den Filter eingefügt werden, werden seine Hashwerte berechnet und die die entsprechenden Bits im Filter gesetzt[7].

Die Hashwerte werden verwendet, um Anfragen auf dem Filter auszuführen. Ist ein Objekt im Filter enthalten, sind seine Bits gesetzt worden, d.h. man kann eine Anfrage nach seinen Hashwerten durchführen und so mit großer Wahrscheinlichkeit ermitteln, ob es in den Filter eingefügt wurde: Sind ein oder mehrere Bits des Anfrageobjekts nicht gesetzt, so ist das Element mit Sicherheit nicht vorhanden. Es gibt also keine falsch negativen Antworten. Allerdings kann es sein, dass ein Element nicht in den Filter eingefügt wurde, obwohl alle seine Bits gesetzt sind (falsch positive Antworten). Grund dafür ist die Kollisionseigenschaft von Hashfunktionen, die ein großes Universum von Elementen auf einen sehr viel kleineren Wertebereich, hier die Länge des Bloom-Filters, abbilden. Es kann somit zu Kollisionen zwischen unterschiedlichen Informationsobjekten bzw. ihren charakteristischen Hashwerten kommen. Die Falsch-Positiv-Rate eines Bloom-Filters ist abhängig von m, k und der Anzahl der eingefügten Elemente n und lässt sich berechnen als[8]

$$f = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k.$$

[5] [WDS15]: 4.
[6] Vgl. [Blo70].
[7] Vgl. [BM04]: 487.
[8] Vgl. ebd. für eine umfassende Darstellung.

Aus der Kollisionseigenschaft folgt auch, dass ein einmal eingefügtes Objekt nicht mehr aus einem Bloom-Filter gelöscht werden kann, weil das offensichtlich zu falsch negativen Ergebnissen für andere Objekte führen könnte.

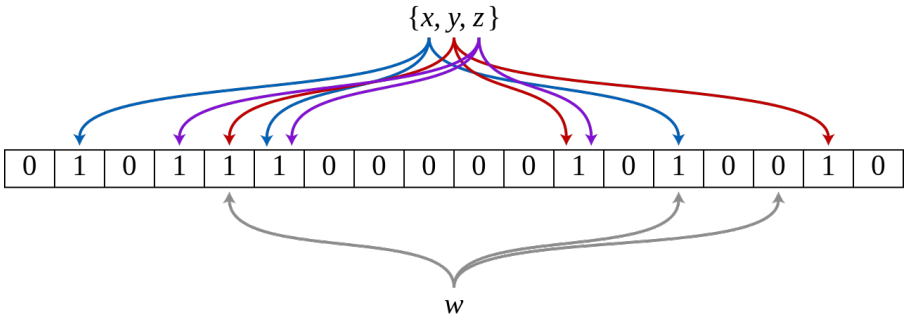

Abbildung 2.1: Beispiel für einen Bloomfilter, in den die Objekte x, y und z eingefügt wurden. Das Objekt w ist nicht im Filter vorhanden.

2.2.1 Distanzmaße

Um die Ähnlichkeit zweier Mengen zu ermitteln, werden verschiedene Distanzmetriken oder Ähnlichkeitsmaße eingesetzt. Um Bloom-Filter miteinander zu vergleichen und insbesondere für die k-nächste-Nachbarn-Suche muss ein geeignetes Ähnlichkeitsmaß zur Anwendung kommen. Bayardo et al. verwenden dazu die *Kosinus-Ähnlichkeit*[9]. Sakuma und Sato definieren die Ähnlichkeit von Bloom-Filtern über die Anzahl gleicher 1-Bits[10]. Ist diese für zwei Anfragefilter identisch, werden die Bit-Arrays negiert und die Anzahl gleicher 0-Bits verglichen.

AMBIENCE verwendet eine Abschätzung der *Jaccard-Distanz* zur Ermittlung der Ähnlichkeit von Bloom-Filtern. Die Jaccard-Distanz zwischen zwei Mengen A und B ist definiert als

$$J_\delta(A,B) = 1 - J(A,B) = \frac{|A \cap B| - |A \cup B|}{|A \cup B|},$$

wobei

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

den *Jaccard-Koeffizienten* bezeichnet. Die Jaccard-Distanz nimmt Werte im Bereich $[0,1]$ an. Identische Mengen haben eine Jaccard-Distanz von 0, Mengen ohne gemeinsame Elemente haben eine Jaccard-Distanz von 1. Für Bloom-Filter lässt sich die Jaccard-Distanz analog berechnen. Die Vereinigungsmenge zweier Bloom-Filter F und G lässt sich als bitweises logisches Oder, die Schnittmenge als bitweises logisches Und repräsentieren. Auch hier nimmt die Jaccard-Distanz offensichtlich Werte zwischen 0 und 1 an. Je ähnlicher die Filter sind, desto kleiner ist ihre Jaccard-Distanz. Die Jaccard-Distanz zwischen Bloom-Filtern ist, anders als die von Sakuma und Sato verwendete Distanzmetrik, nicht transitiv in dem Sinne, dass zwei Filter, die beide eine geringe Jaccard-Distanz zu einem dritten Filter aufweisen, untereinander nicht ähnlich sein müssen. Man betrachte z.B. die Filter

[9] Vgl. [BMS07].
[10] Vgl. [SS11]: 321.

F1, *F2* und *F3* mit folgenden Werten: Die Jaccard-Distanzen zwischen *F1*, *F2* und *F3* betragen somit:

$$J_\delta(F1, F3) = 1 - J(F1, F3) = 1 - \frac{|F1 \cap F3|}{|F1 \cup F3|} = 1 - \frac{5}{10} = 0.5$$

$$J_\delta(F2, F3) = 1 - J(F2, F3) = 1 - \frac{|F2 \cap F3|}{|F2 \cup F3|} = 1 - \frac{5}{10} = 0.5$$

$$J_\delta(F1, F2) = 1 - J(F1, F2) = 1 - \frac{|F1 \cap F2|}{|F1 \cup F2|} = 1 - 0 = 1$$

Daran wird deutlich: Obwohl *F1* und *F2* jeweils die Hälfte der Elemente mit *F3* gemeinsam haben, lässt sich daraus kein Wert für die Ähnlichkeit zwischen *F1* und *F2* ableiten. Sie sind sich im Gegenteil maximal unähnlich.

2.2.2 Teil- und Obermengenbeziehung

Will man Bloom-Filter z.B. nach Ähnlichkeit gruppieren, kann man stattdessen Teil- und Obermengenbeziehungen zwischen ihnen betrachten. Die Teilmengenbeziehung zwischen zwei Bloom-Filtern sei hier wie folgt definiert:

> Ein Bloom-Filter F ist *Teilmenge* eines Bloom-Filters G, wenn darin mindestens die gleichen 0-Bits gesetzt sind wie in G (und möglicherweise weitere, zusätzliche 0-Bits).

Die Obermengenbeziehung zwischen zwei Bloom-Filtern sei hier wie folgt definiert:

> Ein Bloom-Filter F ist *Obermenge* eines Bloom-Filters G, wenn darin mindestens die gleichen 1-Bits gesetzt sind wie in G (und möglicherweise weitere, zusätzliche 1-Bits).

Der maximal gefüllte Filter, in dem alle Bits gesetzt sind, ist damit Obermenge aller Filter derselben Länge (auch von sich selbst). Der leere Filter ist die (triviale) Teilmenge aller Filter derselben Länge (auch von sich selbst). Teil- und Obermengen sind Umkehrungen

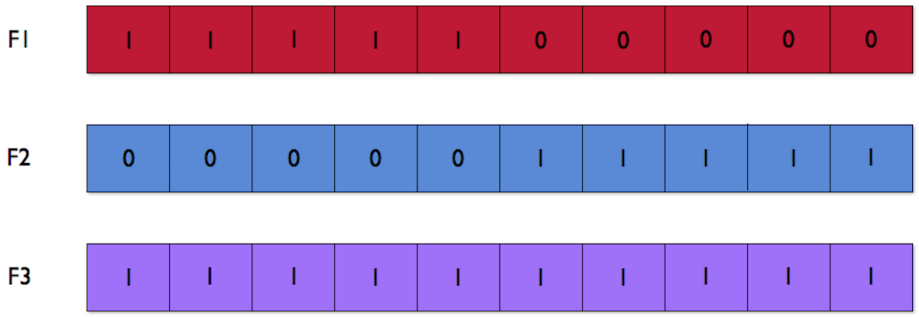

Abbildung 2.2: Jaccard-Distanzen zwischen Bloom-Filtern.

voneinander, d.h. wenn F Teilmenge von G ist, folgt daraus, dass G Obermenge von F ist, und umgekehrt.

Zwischen den Bloom-Filtern in Abb. 2.2 bestehen folgende Teil- und Obermengenbeziehungen: *F1* und *F2* sind Teilmengen von *F3*. *F3* ist Obermenge von *F1* und *F2*. Zwischen den maximal unähnlichen Filtern *F1* und *F2* bestehen keine Teil- und Obermengenbeziehungen.

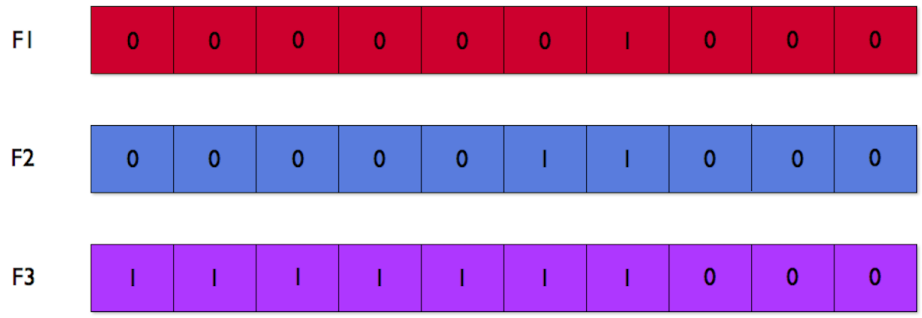

Abbildung 2.3: Teil- und Obermengenbeziehungen zwischen Bloom-Filtern.

Teil- und Obermengenbeziehung sind außerdem transitiv, was an Abb. 2.3 deutlich wird: *F1* ist Teilmenge von *F2*, damit auch Teilmenge von *F3*. *F3* ist Obermenge von *F2*, damit auch Obermenge von *F1*.

Teil- und Obermengenbeziehungen sind also im Gegensatz zur Jaccard-Distanz dazu geeignet, transitive Ähnlichkeitsbeziehungen zwischen Bloom-Filtern abzubilden. Diese Eigenschaft spielt eine zentrale Rolle im hier entwickelten Verfahren, das in Kap. 4 ausführlich dargestellt wird.

2.2.3 Hashfunktionen

Die Frage nach den idealen Hashfunktionen für einen Bloom-Filter ist nicht eindeutig zu beantworten[11]. Grundsätzlich muss zwischen kryptografischen Hashfunktionen wie MD5 und SHA und gewöhnlichen Hashfunktionen wie Murmur- oder Jenkins-Hashfunktionen unterschieden werden. Die Berechnung von kryptografischen Hashfunktionen dauert in der Regel länger, dafür haben sie bestimmte Eigenschaften wie eine hohe Kollisionsresistenz und Gleichverteilung der Ergebniswerte.

Werden Bloom-Filter z.B. zum schnellen Nachschlagen in großen, verteilten Datenbanken eingesetzt, wird auf die kryptografischen Eigenschaften zu Gunsten des verminderten Rechenaufwandes verzichtet. Das NoSQL-Datenbanksystem Cassandra und das Hadoop-Framework für skalierte, verteilt arbeitende Software verwenden beispielsweise Bloom-Filter in Kombination mit Murmur- und Jenkins-Hashfunktionen. Darüber hinaus ist MD5 für Bloom-Filter weit verbreitet. Die Murmur-Hashfunktionen weisen gleichzeitig gute Verteilungseigenschaften auf und lassen sich vergleichsweise schnell berechnen, wes-

[11] Vgl. [BM04]: 487.

wegen sie generell für den Einsatz in Bloom-Filtern empfohlen werden[12]. AMBIENCE verwendet Murmur-Hashfunktionen zur Generierung der Bloom-Filter. Für die eigene Implementierung wurde der Murmur2-Hash verwendet[13].

2.2.4 Bloom-Filter-Varianten und Anwendungen

Wegen ihres geringen Speicherbedarfs und einfachen Implementierung erfreuen sich Bloom-Filter in unterschiedlichsten Versionen großer Beliebtheit. Wichtige Varianten sind z.B. *Attenuated Bloom Filter*, *Counting Bloom-Filter* und *Compressed Bloom Filter*. Ein Counting Bloom-Filter benötigt mehr Speicherplatz als ein klassischer Bloom-Filter, dafür können Objekte wieder daraus entfernt werden[14]. Komprimierte Bloom-Filter werden eingesetzt, wenn Bloom-Filter als Nachrichten mit begrenzter Länge versendet werden oder die übertragene Datenmenge minimiert werden soll[15]. Attenuated Bloom-Filter[16] können als Array von Bloom-Filtern betrachtet werden und können z.B. in einem Netzwerk Informationen darüber enthalten, welche Dienste an einem anderen Knotenverfügbar sind. Besonders häufig kommen Bloom-Filter in verteilten Anwendungen und Netzwerkdiensten zum Einsatz[17].

Neben Hadoop und Cassandra werden Bloom-Filter in unzähligen, zum Teil hoch skalierenden Anwendungen eingesetzt. Weitere Beispiele sind der quelloffene Webproxy Squid und der Chrome-Browser, wo Bloom-Filter zum schnellen Nachschlagen als bösartig eingestufter Webseiten verwendet werden. Broder und Mitzenmacher formulieren das Bloom-Filter-Prinzip wie folgt:

> Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated[18].

2.3 Indexstrukturen

Zur effizienten Bearbeitung von Anfragen und Operationen kommen in der internen Schicht von Datenbanksystemen spezielle Datenstrukturen und Speicherverfahren zum Einsatz. Sie werden *Indexstrukturen* genannt und organisieren die Daten an Hand von Indizes, um die gewünschten Operationen zu unterstützen[19].

Ein *Index* oder *Verzeichnis* einer Datei enthält Informationen über ihre Struktur, wobei mit „Datei“ in diesem Zusammenhang eine komplette Datenstruktur gemeint ist, also z.B. ein Suchbaum oder ein Array. Indexstrukturen lassen sich danach klassifizieren, wie sie die Daten organisieren:

1. *Daten-organisierende Indexstrukturen* werden zur Organisation der tatsächlich anfallenden Daten eingesetzt – meist in Form von *Suchbäumen*.

[12] Vgl. `http://spyced.blogspot.de/2009/01/all-you-ever-wanted-to-know-about.html`.
[13] Vgl. `https://sites.google.com/site/murmurhash/MurmurHash2.cpp` für den Quellcode.
[14] Vgl. [FCAB00].
[15] Vgl. [Mit02].
[16] Vgl. [SS11]: 316 und 318.
[17] Vgl. [BM04] für eine ausführliche Darstellung.
[18] [BM04]: 486.
[19] Trotz der großen Verbreitung von Indexstrukturen in Datenbanksystemen scheinen in der Literatur keine überblicksartigen Darstellungen zu existieren. Der aktuelle Abschnitt stützt sich daher im Wesentlichen auf das Skript zur Vorlesung *Anfragebearbeitung und Indexstrukturen in Datenbanksystemen* im Wintersemester 2013/2014 an der Ludwig-Maximilians-Universität München (vgl. [Kri14]).

2. *Raum-organisierende Indexstrukturen* werden zur Organisation des Speichers eingesetzt, in dem die Daten gehalten werden. Sie verwenden vor allem *dynamische Hashverfahren.*
3. *Hybride Indexstrukturen* sind eine Kombination der vorgenannten Klassen und basieren auf *Hashbäumen.*

Eine gute bzw. effektive Indexstruktur sollte folgenden Anforderungen genügen:

1. *Effiziente Suche:* Eine Suchanfrage auf der Indexstruktur soll in optimaler Zeit ein Ergebnis liefern. D.h. die Anfrage soll in möglichst wenig Schritten an die Seite oder Seiten weiter geleitet werden, die die angefragten Daten enthalten.
2. *Dynamisches Einfügen, Modifizieren und Löschen von Datensätzen:* Die zu organisierende Datenmenge verändert sich möglicherweise über die Zeit, was durch die Indexstruktur widergespiegelt und unterstützt werden muss.
3. *Erhalt der lokalen Ordnung:* Falls es Datensätze gibt, deren Schlüssel in der angewandten Ordnungsrelation (z.B. die Kleiner-Gleich-Ordnung) aufeinander folgen, sollte die Indexstruktur diese Ordnung übernehmen. Suchbäumen erfüllen diese Eigenschaft, nicht aber lineare Hashverfahren. Die Wahl bzw. Implementierung der Indexstruktur muss also zum Anwendungsfall passen.
4. *Speichereffizienz:* Effiziente Speichernutzung ist für real existierende und hoch skalierende Anwendungen von zentraler Bedeutung.

Weitere mögliche Anforderungen sind *Machbarkeit* und *Implementierungskosten.* Sie sind für die vorgestellte Implementierung nachrangig in dem Sinne, dass der Nachweis der Machbarkeit durch die Implementierung selbst erfolgt. Da AMBIENCE ein Prototyp ist und im akademischen Umfeld entwickelt wurde, kann die wirtschaftliche Kalkulation der Kosten, wie sie ein Unternehmen vornehmen würde, außer Acht gelassen werden.

Für Implementierung (vgl. Kap. 4) und Evaluation (vgl. 5) stehen daher die Anforderungen 1–4 im Mittelpunkt. Als weiteres Kriterium, das nicht zu den allgemeinen Anforderungen an Indexstrukturen zählt, wurden die Aufbaukosten der Indexstruktur betrachtet.

2.3.1 B-Bäume

B-Bäume sind eine weit verbreitete Form der Suchbäume und wurden zuerst 1972 von Rudolf Bayer und Edward M. McCreight vorgestellt. Sie erfüllen unter anderem folgende Eigenschaften:

1. *Aufbau:* B- und B^+-Bäume wachsen und schrumpfen von der Wurzel ausgehend.
2. *Balanciertheit:* Alle Blätter sind auf demselben Level.
3. *Minimaler Grad/Ordnung:* B- und B^+-Bäume sind definiert durch die Ordnung oder den minimalen Grad t, d.h. jeder Knoten außer der Wurzel enthält mindestens t Schlüssel.
4. *Suchbaumeigenschaft:* Schlüssel sind aufsteigend sortiert.
5. *Verzweigungsgrad:* Ein innerer Knoten mit k Schlüsseln hat genau $k+1$ Kinder[20].

[20] Vgl. [OW12]: 339–348. Den Grad bzw. die minimale Ordnung betreffend ist die Nomenklatur in der Literatur uneinheitlich. [OW12] sprechen in Anlehnung an [Knu99] von B-Bäumen der Ordnung m und fordern, dass jeder Knoten mit Ausnahme der Wurzel und der Blätter mindestens $\lceil \frac{m}{2} \rceil$ enthalte (vgl. ebd.: 342f.). Stein et al. definieren den minimalen Grad wie oben beschrieben (vgl. [SCLR09]: 489). Kriegel verwendet dasselbe Kriterium, bezeichnet es jedoch als Grad m (vgl. [Kri14]: 9).

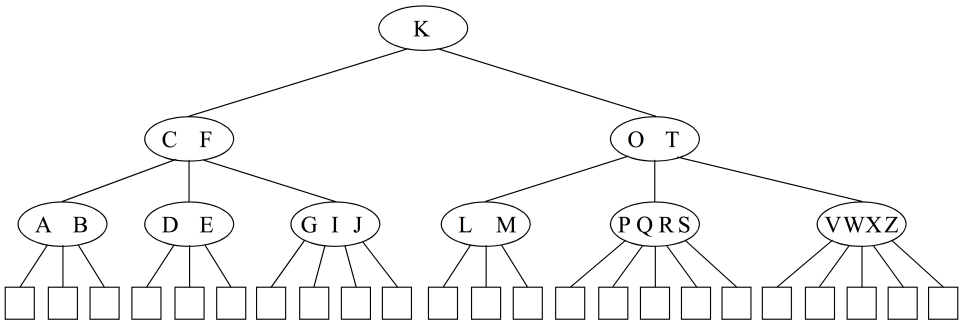

Abbildung 2.4: Ein B-Baum der Ordnung 2.

Unterstützte Operationen sind in der Regel Einfügen, Suchen und Löschen von Datensätzen. Dabei ist die Löschoperation am aufwendigsten zu implementieren, da auch das Löschen eines Schlüssels aus einem inneren Knoten betrachtet werden muss. Ein Schlüssel aus einem inneren Knoten kann nicht direkt gelöscht werden, da er zusätzlich als Separator für seine Kindknoten dient. Es muss daher ein neuer Separator aus dem rechten oder linken Kindknoten entnommen und in den Knoten eingefügt werden. Falls damit die B-Baum-Eigenschaften verletzt werden, d.h. der Kindknoten anschließend zu wenig Schlüssel enthält, muss eine Underflow-Behandlung eingeleitet werden, bei dem der Kindknoten mit einem Geschwisterknoten verschmolzen wird. Der Underflow kann sich rekursiv bis zur Wurzel fortsetzen und dafür sorgen, dass der Baum eine Ebene verliert und eine neue Wurzel erhält.

2.3.2 B^+-Bäume

Das entwickelte Verfahren stützt sich stark auf *B^+-Bäume*, eine Erweiterung der B-Bäume. Sie unterscheiden sich von ihnen in folgenden Eigenschaften:

1. *Minimale Schlüsselanzahl:* Jeder innere Knoten enthält mindestens einen Schlüssel.
2. *Ordnungsrelation:* Der Kindknoten zwischen den Schlüsseln *k1* und *k2* enthält alle Schlüssel $\geq$ *k1* und $<$ *k2*
3. *Speicherung der Datensätze:* Alle Schlüssel werden auch in den Blättern gespeichert. Die tatsächlichen Datensätze sind mit dem entsprechen Schlüssel im Blatt verknüpft.
4. *Sequentielle Verkettung:* Alle Blätter sind gemäß der Ordnung auf den Primärschlüsseln verkettet.

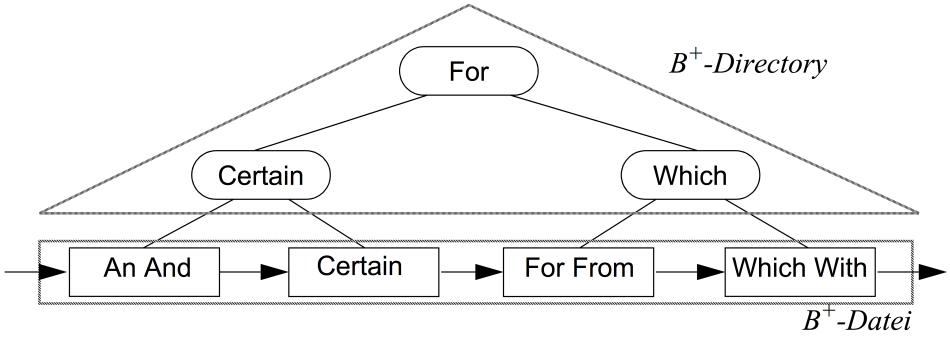

Abbildung 2.5: Ein B$^+$-Baum der Ordnung 2.

Die wesentlichen Vorteile gegenüber dem B-Baum sind der höhere Verzweigungsgrad und die Bereichssuche. Tab. 2.1 vergleicht die durchschnittlichen Laufzeiten für Einfügen, Suchen und Löschen in B-Bäumen und B$^+$-Bäumen:

	Einfügen	**Suchen**	**Löschen**	**Bereichssuche**
B-Baum	$O(\log(2t-1))$	$O(\log(2t-1))$	$O(\log(2t-1))$	$\times$
B$^+$-Baum	$O(\log(2t-1))$	$O(\log(2t-1))$	$O(\log(2t-1))$	$O(\log 2t-1)$

Tabelle 2.1: Laufzeiten von Operationen auf Suchbäumen.

Daran wird deutlich, dass die Kosten für Einfügen, Suchen und Löschen in beiden Varianten vom Parameter t abhängen. Da die inneren Knoten im B$^+$-Baum mehr Schlüssel enthalten als im B-Baum, wird der Baum breiter und flacher. Geht man davon aus, dass ein Zugriff auf einen inneren Knoten einem Plattenzugriff entspricht, wird deutlich, dass sich damit die Kosten für Einfügen, Suchen und Löschen gegenüber dem B-Baum reduzieren lassen.

Die sequentielle Verkettung der Blätter im B$^+$-Baum ermöglicht zudem eine Bereichssuche. Das ist ein immenser Vorteil gegenüber dem B-Baum, bei dem jeder für eine Bereichssuche jeder einzelne Datensatz mit Kosten von $O(\log(2t-1))$ gesucht werden muss. Diese Kosten fallen beim B$^+$-Baum nur für den ersten Datensatz mit dem kleinsten Primärschlüssel an. Anschließend kann eine doppelt verkettete Liste traversiert werden.

3 Verwandte Themen

Wie in Abschnitt 2.2.4 erwähnt, sind Bloom-Filter eine häufig verwendete Datenstruktur. Auch in der Referenz-Implementierung AMBIENCE spielen sie eine wichtige Rolle. Somit stellte sich für die eigene Arbeit weder die Frage, ob überhaupt Bloom-Filter eingesetzt werden sollen, noch nach ihrer optimalen Implementierung. AMBIENCE diente vielmehr als Schnittstelle für den eigenen Entwurf.

Dennoch wurden auch in dieser Arbeit Bloom-Filter implementiert, um die eigenen Berechnungen zu überprüfen und mit realistischen Werten arbeiten zu können. Das folgende Kapitel stellt zunächst bewährte Techniken zur Implementierung von Bloom-Filtern dar. Die Darstellung des aktuellen Forschungsstandes beschränkt sich im Folgenden wegen der Fülle der Einsatzmöglichkeiten auf Bloom-Filter im Zusammenhang mit Indexstrukturen und die k-nächste-Nachbarn-Suche.

Ziel dieser Arbeit war eine optimale Lösung für das Anwendungsszenario von AMBIENCE. Eine solche wird vom aktuellen Stand der Forschung nicht abgedeckt. Überlegungen und Arbeiten, die Eingang in die eigene Implementierung gefunden haben, werden in den kommenden Abschnitten vorgestellt.

3.1 Implementierung von Bloom-Filtern

Wie in Abschnitt 2.2.4 dargestellt, bietet sich die Verwendung von Bloom-Filtern an, wenn Speicherplatz effektiv genutzt werden soll und die Auswirkungen von falsch Positiven zu verkraften sind. Die mathematischen Grundlagen wie Minimierung der Falsch-Positiv-Rate, Abschätzung der Anzahl eingefügter Elemente, Abschätzung der Jaccard-Distanz etc. werden von Bloom[1], Broder, Mitzenmacher [2] ausführlich dargestellt, um nur einige zu nennen.

Wie in den Abschnitten 2.2.3 und 2.2.4 erwähnt, werden Bloom-Filter im Apache-Projekt Cassandra eingesetzt. Sie dienen dort zum schnellen Nachschlagen in Tabellen, den so genannten *SSTables*. Die Cassandra-Entwickler, namentlich Jonathan Ellis, haben sich ausführlich mit der Implementierung von Bloom-Filtern und optimalen Hashfunktionen beschäftigt, ohne dass dies Eingang in wissenschaftliche Veröffentlichungen gefunden hätte. Die von ihnen getätigten Überlegungen sind dennoch sehr praxisrelevant[4]. Zur Wahl der Hashfunktionen schreibt Ellis:

[1] Vgl. [Blo70].

[2] Vgl. [BM04] und [Mit02] und Werner[3].

[4] `http://cassandra.apache.org/` ist die Hauptseite des Cassandra-Projekts. Der Cassandra-Quellcode ist frei verfügbar unter `https://github.com/apache/cassandra`. Datenmodell und Architektur werden z.B. auf `http://wiki.apache.org/cassandra/DataModel`, `http://wiki.apache.org/cassandra/ArchitectureOverview` und `http://prettyprint.me/prettyprint.me/2010/05/02/understanding-cassandra-code-base/index.html` beleuchtet. Für die vorliegende Arbeit wurden auch eine programmatische Rede (vgl. `https://youtu.be/WD1v6jr5fKY`) und Jonathan Ellis' Blog berücksichtigt (vgl. `http://spyced.blogspot.de/`).

> [I]t turns out that it's surprisingly hard to find good information on one part of the implementation: how do you generate an indefinite number of hashes? Even small filters will use three or four; a dozen or more is not unheard of[5].

Die maththmatischen Grundlagen finden sich z.B. bei Kirsch und Mitzenmacher[6]. Viele Bloom-Filter-Implementierungen wie z.B. *PyBloom* verwenden jedoch Hashfunktionen, deren Ergebnisse nicht gleich verteilt sind. Das führt zu einer deutlich höheren Falsch-Positiv-Rate im Bloom-Filter als rechnerisch angenommen.

Der Grund dafür ist laut Ellis, dass die meisten Implementierungen von Hash-Funktionen auf schnelle Berechnung abzielten statt auf Gleichverteilung der Ergebnisse. Das ist aber für einen Bloom-Filter essentiell, wenn z.B. wie in Cassandra teure Eingabe/Ausgabe-Operationen durch den Einsatz von Bloom-Filtern minimiert werden sollen. Weist der Bloom-Filter eine erhöhte Falsch-Positiv-Rate auf, beispielsweise von 140% gegenüber dem erwarteten Wert, reduziert das die positiven Effekte des Bloom-Filters.

Ellis sieht hierfür zwei Lösungsansätze: Entweder die Verwendung von kryptografischen Hashfunktionen oder von Hashfunktionen mit guter Gleichverteilung der Ergebnisse wie Murmur- und Jenkins-Hashfunktionen. Kryptografische Hashfunktionen wurden bereits in Abschnitt 2.2.3 dargestellt. Der Nachteil daran ist, dass ihre Berechnung in der Regel aufwändiger sind als als von gewöhnlicher Hashfunktionen. Das spielt z.B. keine wichtige Rolle bei der einmaligen Berechnung eines Fingerabdrucks. Die Performance von Bloom-Filtern kann dadurch aber beeinträchtigt werden.

Beim Einsatz von Murmur- oder Jenkins-Hashfunktionen gibt es zwei Möglichkeiten, um eine beliebige Anzahl von Hashfunktionen mit guter Gleichverteilung der Ergebnisse zu erzeugen. Entweder berechnet man den i-ten Hashwert als $\text{hash0} + i * \text{hash1}$ wie von Kirsch und Mitzenmacher beschrieben[7] und in Cassandra angewendet. Alternativ nimmt man den i-ten Hashwert als Startwert für die Berechnung des $i+1$-ten Hashwerts. Dieser Ansatz wurde in Hadoop gewählt und auch in der eigenen Implementierung verwendet.

Zur Organisation der Bloom-Filter äußert sich Ellis ebenfalls, jedoch nur auf seinem Twitter-Account und ohne auf Details einzugehen. Er schreibt hierzu:

> Rather than naively checking every Bloom Filter for the element, organize the BF in a hierarchy akin to B+ tree[8].

3.2 k-nächste-Nachbarn-Suche

3.3 Bloom-Filter in Indexstrukturen

[5] Vgl. `http://spyced.blogspot.de/2009/01/all-you-ever-wanted-to-know-about.html`.
[6] Vgl. [KM06].
[7] Vgl. ebd..
[8] Vgl. `https://twitter.com/spyced/status/70726703751651328`.

4 Implementierung

5 Evaluation

6 Zusammenfassung und Ausblick

Abbildungsverzeichnis

Literaturverzeichnis

[ADI+12] Ahlgren, Bengt, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher und Börje Ohlman: *A Survey of Information-Centric Networking*. Communications Magazine, IEEE, 50(7):26–36, 2012.

[Blo70] Bloom, Burton H.: *Space/Time Trade-offs in Hash Coding with Allowable Errors*. Communications of the ACM, 13(7):422–426, 1970.

[BM04] Broder, Andrei und Michael Mitzenmacher: *Network Applications of Bloom Filters: A Survey*. Internet Mathematics, 1(4):485–509, 2004.

[BMS07] Bayardo, Roberto J., Yiming Ma und Ramakrishnan Srikant: *Scaling Up All Pairs Similarity Search*. In: *Proceedings of the 16th international conference on World Wide Web*, Seiten 131–140. ACM, 2007.

[DA99] Dey, Anind K. und Gregory D. Abowd: *Towards a Better Understanding of Context and Context-Awareness*. In: *Handheld and Ubiquitous Computing*, Seiten 304–307. Springer, 1999.

[FCAB00] Fan, Li, Pei Cao, Jussara Almeida und Andrei Broder: *Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol*. IEEE/ACM Transactions on Networking (TON), 8(3):281–293, 2000.

[KM06] Kirsch, Adam und Michael Mitzenmacher: *Less Hashing, Same Performance: Building a Better Bloom Filter*. In: *European Symposium on Algorithms*, Seiten 456–467. Springer, 2006.

[Knu99] Knuth, Donald E.: *The Art of Computer Programming*, Band 3. Addison Wesley Longman, 1999.

[Kri14] Kriegel, Hans-Peter: *Skript zur Vorlesung Anfragebearbeitung und Indexstrukturen in Datenbanksystemen*. 1994–2014.

[Mit02] Mitzenmacher, Michael: *Compressed Bloom Filters*. IIEEE/ACM Transactions on Networking (TON), 10(5):604–612, 2002.

[OW12] Ottmann, Thomas und Peter Widmayer: *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag, 5 Auflage, 2012.

[SAW94] Schilit, Bill, Norman Adams und Roy Want: *Context-aware Computing Applications*. In: *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, Seiten 85–90. IEEE, 1994.

[SCLR09] Stein, Clifford, Thomas H. Cormen, Charles E. Leiserson und Ronald L. Rivest: *Introduction to Algorithms*. MIT Press, 3rd Auflage, 2009.

[SS11] Sakuma, Hiroshi und Fumiako Sato: *Evaluation of the Structured Bloom Filters Based on Similarity*. In: *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on*, Seiten 316–323, März 2011.

[WDS15] Werner, Martin, Florian Dorfmeister und Mirco Schönfeld: *AMBIENCE: A Context-Centric Online Social Network*. In: *12th IEEE Workshop on Positioning, Navigation and Communications (WPNC '15)*, 2015.