Martin Eisenhardt, Andreas Henrich, Wolfgang Müller:
Inhaltsbasierte Suche in P2P-Systemen.
Datenbank-Spektrum, Heft 12: S. 5-15 (2005)

Inhaltsbasierte Suche in P2P-Systemen

Martin Eisenhardt Andreas Henrich
Wolfgang Müller

{martin.eisenhardt|andreas.henrich|wolfgang.mueller}@wiai.uni-bamberg.de

LS Medieninformatik, Otto-Friedrich-Universität Bamberg

Februar 2005

Peer-to-Peer-Netze (P2P-Netze) erfreuen sich zunehmender Beliebtheit zur verteilten und administrationsarmen Verwaltung größerer Dokumentenkollektionen. Ein wichtiges Problem ist dabei die Suche und insbesondere die inhaltsbasierte Suche nach Dokumenten in P2P-Netzen. Der vorliegende Beitrag gibt einen Überblick über entsprechende Ansätze und zeigt interessante Forschungsaspekte auf.

1 Einführung

Die Verwaltung großer Dokumentenkollektionen ohne zentrale Instanz und mit geringem Administrationsaufwand ist ein primäres Ziel von P2P-Systemen. So könnten z.B. Freiberufler Know-How-Dokumente zu ihrem Arbeitsgebiet über ein P2P-Netz gemeinsam nutzen. Auch der Aufbau verteilter Datensammlungen mit Multimediadaten z.B. für die Entwicklung von eLearning-Angeboten ist denkbar. Den meisten Nutzungsszenarien ist gemein, dass eine effiziente und effektive Suchfunktionalität erforderlich ist.

1.1 Ein Klassifikationsschema

Um eine systematische Betrachtung der verschiedenen Ansätze zur Suche in P2P-Netzen zu ermöglichen, nutzen wir die in Abbildung 1 aufgeführten Klassifikationsdimensionen.

Diese Klassifikationsdimensionen werden im Folgenden zunächst kurz erläutert. Dabei sei darauf hingewiesen, dass die angegebenen Dimensionen keineswegs orthogonal sind. Sie geben aber verschiedene sinnvolle Blickwinkel wieder:

- *Basis der Suche:* Zahlreiche Ansätze zielen auf die effiziente Suche nach Objekten mit Hilfe eines eindeutigen Schlüssels. Dieser Schlüssel kann aus dem Anwendungsgebiet entstammen (z.B. bei einer ISBN) oder er kann durch eine Hashfunktion erzeugt werden. Bei dieser Art der Suche muss der Anfragesteller offensichtlich genau wissen, wonach er sucht.

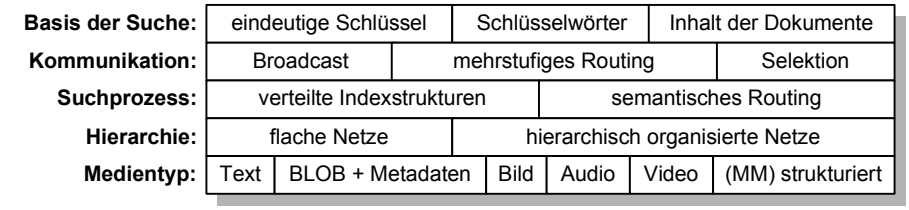

Abbildung 1 — Ein Klassifikationsschema für Ansätze zur Suche in P2P-Netzen

Eine Alternative bietet die Suche auf Basis von Schlüsselwörtern. Dabei werden den verwalteten Objekten manuell Schlüsselwörter zugeordnet, die in Suchanfragen adressiert werden können.

Die dritte Möglichkeit bildet schließlich die inhaltsbasierte Suche, die in der Regel auf automatisch generierten Feature-Vektoren zu den Objekten basiert. Dabei kann es sich im Falle von Textdokumenten um Beschreibungsvektoren nach dem Vektorraummodell handeln (vgl. z.B. [9]), in denen die Komponenten ausdrücken, wie gut das Dokument durch einzelne Begriffe beschrieben wird. Im Falle von Bildern ist beispielsweise an Feature-Vektoren zu denken, die die Farbverteilung oder Textureigenschaften repräsentieren. Für die Vektoren wird dann im Rahmen einer Anfrage die Ähnlichkeit zu einem gegebenen Anfragevektor bestimmt. Aufgrund der sich ergebenden Ähnlichkeitswerte kann eine Rangordnung der Dokumente aufgestellt werden, die (hoffentlich) deren inhaltliche Relevanz im Hinblick auf die Anfrage wiedergibt.

- *Kommunikation:* Werden die Objekte im P2P-Netz quasi am Ort ihrer Entstehung verwaltet, so können zu einer Anfrage gesuchte Objekte potentiell auf beliebigen Peers im Netz abgelegt sein. Um alle Peers zu erreichen, die potentiell zum Ergebnis der Anfrage beitragen können, kann man die Anfrage dann in einem Broadcast im P2P-Netz verteilen. Der sich ergebende Kommunikationsaufwand ist aber erheblich.

  Alternativ kann ein mehrstufiges Routing der Anfrage erfolgen, bei dem die Anfrage gezielt über einen bestimmten Pfad zum möglichen Speicherungsort angefragter Dokumente geleitet wird. Um dies zu ermöglichen, müssen die Dokumente oder zumindest Referenzen auf die Dokumente in der Regel auf bestimmten Peers im Netz (die z.B. über eine Hashfunktion zu ermitteln sind) abgelegt werden.

  Im Falle inhaltsbasierter Anfragen kann das mehrstufige Routing im Allgemeinen nicht exakt erfolgen. Statt dessen werden bei der Verteilung einer Anfrage im Netz jeweils nur solche Peers angesprochen, die mit hoher Wahrscheinlichkeit zu guten Treffern führen. Dies erfordert zusätzliche Informationen über die auf den erreichbaren Peers verwalteten Dokumente, die eine solche Selektion erlauben.

- *Suchprozess:* Zur Suche wurden verschiedene Ansätze vorgeschlagen, die auf der Verteilung einer Indexstruktur im P2P-Netz basieren (Hashtabelle, Suchbaum, invertierte Listen, ...). Im Falle invertierter Listen können die Listen einzelnen Peers zugeordnet werden. Sucht man dann mit zwei Anfragewörtern nach Dokumenten, so kann man zunächst durch eine schlüsselbasierte Suche die Peers ermitteln, die die entsprechenden Listen zu den Anfragewörtern enthalten. Diese Listen müssen dann übertragen und gemischt werden, um das Anfrageergebnis zu bestimmen.

Beim *semantischen Routing* verwaltet dagegen im einfachsten Fall jeder Peer seine Daten selbst (mit einer lokalen Indexstruktur). Eine Anfrage wird dann in der Regel nach dem Kommunikationsmuster der *Selektion* zu den Peers geleitet, die mit hoher Wahrscheinlichkeit gute Treffer aufweisen.

- *Hierarchie:* Hier werden Netze, bei denen alle Peers völlig identische Aufgaben übernehmen, von Netzen unterschieden, in denen ausgewählte Peers z.B. im Rahmen des Routing spezielle Aufgaben übernehmen. Dabei nutzt man in vielen P2P-Systemen Multihierarchien, in denen die Peers zum Teil dynamisch und gleichzeitig verschiedene Rollen übernehmen.
- *Medientyp:* Schließlich ist zwischen Systemen zu differenzieren, die Dokumente als *binary large objects* (BLOBs) ohne Kenntnis der Inhalte verwalten, und solchen, die die Medieninhalte gezielt adressieren. In der zweiten Kategorie ist dann zwischen den verschiedenen Medientypen zu unterscheiden.

Neben den angesprochenen Dimensionen sind natürlich weitere Charaktersitika von P2P-Systemen zu nennen, die ebenfalls zu deren Klassifikation herangezogen werden können. Hier ist z.B. an die Nutzung von Replikation und Caching zur Beschleunigung von Suchanfragen zu denken. Die oben genannten fünf Aspekte erscheinen den Autoren aber am bedeutendsten im Hinblick auf die Einordnung von Systemen zur Suche in P2P-Netzen. Bei der Vorstellung einzelner Systeme in den folgenden Kapiteln werden wir daher zur Charakterisierung der Ansätze jeweils die Einordnung bezüglich der für die Ansätze wichtigsten Dimensionen angeben.

Im Folgenden sollen nun verschiedene repräsentative Systeme zur Suche in P2P-Systemen vorgestellt werden. Obwohl unser primäres Augenmerk hier auf der inhaltsbasierten Suche liegt, werden wir zunächst Ansätze für die schlüsselbasierte Suche betrachten, da viele Verfahren zur inhaltsbasierten Suche letztlich Weiterentwicklungen solcher Verfahren sind oder auf diesen aufsetzen.

1.2 Operationen

Zur Beschreibung der Ansätze in den folgenden Kapiteln ist ein Verständnis der von den einzelnen Peers anzubietenden Operationen erforderlich. Für ein P2P-System ist grundsätzlich zu klären, wie alle diese Operationen realisiert werden:

- *Initial Introduction* bzw. *Join:* Durch die *Initial Introduction* wird eine erste Verbindung eines neuen Peers mit dem Netz hergestellt. Dabei ist primär zu klären, wie Informationen über den neuen Peer im Netz verbreitet werden und wie der Peer mit Informationen über andere Peers im Netz versorgt wird. Jeder Peer kennt in der Regel nur einen Ausschnitt des Netzes, der durch seine so genannten *Nachbarn* definiert ist. Verbindet sich ein Peer nach einer zwischenzeitlichen Unterbrechung erneut mit dem Netz, so verwendet er hierzu die *Join*-Operation. Dabei sind vor allem Informationen über zwischenzeitliche Veränderungen im Netz auszutauschen. Initial Introduction und Join beruhen oft auf den gleichen Algorithmen.
- *Leave:* Wenn ein Peer das Netz temporär verlassen möchte, sollte er eventuell übernommene Aufgaben in einem geordneten Prozess an seine Nachbarn übergeben, damit im Betrieb keine Einschränkungen entstehen. Leider verlassen Peers in der Praxis das Netz häufig ohne eine entsprechende geordnete Abmeldung (man spricht dann von *crash failure*).

- *Suche:* Während die drei ersten Operationen dem Aufbau und der Aufrechterhaltung der Struktur dienen, ist bei der Suche zu klären, wie die Strukturen des P2P-Netzes für eine praktikable Suche genutzt werden können.
- *Einfügen und Löschen von Dokumenten:* Schließlich ist davon auszugehen, dass die Datenbestände auf den einzelnen Peers dynamisch sind. Informationen über neue Dokumente, gelöschte Dokumente und veränderte Dokumente sind dabei ggf. im Netz zu verbreiten.

Da einige der im Weiteren vorgestellten Systeme nicht alle Operationen behandeln, werden wir uns jeweils auf die Beschreibung der Operationen beschränken, die zum Verständnis erforderlich sind.

2 Suche nach Schlüsseln

2.1 Napster

Einordnung: *hierarchisch (zentraler Server), schlüsselbasiert (mit Wildcards)*

Der P2P-Ansatz wurde einer breiten Öffentlichkeit durch Napster bekannt. Dies entbehrt nicht einer gewissen Ironie, da Napster selbst keine reine P2P-Anwendung ist: in Napster verwaltet ein zentraler Index-Server verteilten Plattenplatz. Der Index ermöglicht das Auffinden von Dateien über ihre Dateinamen. Mittels impliziter Wildcards wird die Bearbeitung einfacher Ähnlichkeitsanfragen realisiert, bei denen der Dateiname nur partiell angegeben werden muss. Um bei Napster eine Datei herunterzuladen gibt der Nutzer zunächst eine Anfrage ein, und Napster vermittelt ihm über den Index-Server Referenzen auf Peers, die die gewünschten Dateien enthalten. Zwischen den Peers findet dann ein direkter Datenaustausch statt, um die Datei herunterzuladen. Der zentrale Server stellt dabei einen *single point of failure* dar.

Initial Introduction, Join: Aufgrund der hybriden Struktur sind Initial Introduction und Join sehr einfach: ein aktiver Peer meldet sich beim Server an. Da Napster eine sternförmige Struktur hat, sind keinerlei weitere Schritte für die Aufrechterhaltung der Struktur zu unternehmen.

2.2 Gnutella

Einordnung: *Broadcast, schlüsselbasiert (mit Wildcards), flaches Netz, BLOB + Metadaten*

Gnutella [5] vermeidet zentrale Komponenten. Das Ur-Gnutella ist konsequent als P2P-System angelegt, Peers in Gnutella sind gleichzeitig Clients und Server, so genannte *Servents*.

`query`-Messages in Gnutella erlauben die Suche nach Begriffen in Dateinamen und haben eine so genannte time to live (TTL). Ferner trägt jede Message die Information, wie oft sie schon weitergeleitet worden ist, d.h. wie viele *hops* sie gemacht hat. Ein Peer, der eine `query` empfängt, bearbeitet die Anfrage lokal und leitet sie gleichzeitig an alle seine Nachbarn weiter, sofern die Anzahl der Hops die TTL nicht übersteigt.

Dieses als *flooding* bekannte Verfahren kann als verteilte Variante einer Breitensuche in einem Graphen verstanden werden. Hierbei entsteht ein Spannbaum, entlang dessen die Botschaften weitergegeben werden. Alle Peers des Netzes, die mit *TTL* Hops zu erreichen sind, erhalten und bearbeiten die Anfrage. Die Resultate der Anfrage werden entlang des Spannbaums zum Ausgangspunkt der Anfrage zurückgegeben.

Das Gnutella-Protokoll in seiner Reinform ist überholt. Dies liegt daran, dass für $TTL \rightarrow \infty$ jede *Verbindung* des Gnutella-Netzes betrachtet wird. Potentiell erhält jeder Peer eine gegebene Anfrage über jede eingehende Kante. Hieraus folgt, dass Gnutella für große Netze zu einer hohen Kommunkations- und Anfragebearbeitungslast führen würde.

Gnutella hat weitere Schwächen: Aufgrund der fehlenden Strukturierung des Gnutella-Netzes und der begrenzten TTL kann nicht garantiert werden, dass jeder Knoten die Botschaft erhält. Schlimmer noch, es ist nicht garantiert, dass die *besten* Peers eine Anfrage erhalten.

Initial Introduction, Join: Um sich einem Gnutella-Netz anzuschließen, benötigt der neue Peer die Adresse eines Servents. Dann baut er eine TCP-Verbindung mit dem Servent auf und sendet eine `connect`-Message. Empfängt er als Antwort eine `ok`-Message, ist die Initial Introduction damit abgeschlossen. Nun kennt der neue Peer bis jetzt nur denjenigen Peer, mit dem er sich initial verbunden hat. Jedoch enthält das Gnutella-Protokoll Messages, die es Peers ermöglichen, von der Existenz anderer Peers zu erfahren, mit denen man sich verbinden kann.

Ziel verschiedener Verbesserungen von Gnutella ist die Überwindung dieser Schwächen sowie die Erweiterung der Funktionalität von bzw. gegenüber Gnutella. Ein wichtiger Ansatz hierzu ist die Einrichtung von so genannten Super-Peers [20]. Hierbei wird vom Netz dynamisch festgestellt, welche Peers häufig und zuverlässig online sind sowie eine hohe Bandbreite haben. Diese Peers werden dann zu Super-Peers. Super-Peers verhalten sich gegenüber normalen Peers wie Napster-Server gegenüber Napster-Peers. Ein Super-Peer enthält also die Indexdaten aller angeschlossenen *normalen* Peers und agiert diesen Peers gegenüber als ein Index-Server. Untereinander verhalten sich Super-Peers wie Gnutella-Peers, geben also jede Anfrage an alle Nachbarn weiter.

Ein anderer Ansatz [7] strukturiert das Netz als Hyperwürfel und erreicht damit, dass beim Broadcast nur die minimal notwendige Anzahl von Messages verschickt wird. Gleichzeitig wird erreicht, dass ein Auseinanderbrechen des Netzes (z.b. Aufgrund von Ausfällen oder Angriffen) schwieriger wird.

2.3 Freenet: Suche bei gleichzeitiger Anonymität

Einordnung: *schlüsselbasiert, mehrstufiges Routing, flaches Netz, BLOB*

Freenet [4] hat eine gegenüber den anderen vorgestellten Systemen veränderte Zielsetzung. Hauptziel ist die anonyme Publikation von Inhalten im Internet. Hierbei soll nicht nur die Anonymität des *Publishers* und des *Readers* gewährleistet sein, sondern auch *Server*- und *Dokument*-Anonymität. Server-Anonymität bedeutet, dass für ein gegebenes Dokument nicht vorhergesagt werden kann, auf welchem Server es zu finden ist. Dokument-Anonymität bedeutet, dass ein Server nicht weiß, welche Dokumente er bereitstellt.

Diese harten Anforderungen bedingen, dass Freenet sich in vielerlei Hinsicht von anderen P2P-Systemen unterscheidet. Freenet ist letztlich ein verteilter Cache. Dokumente werden von den Publishern dem Netz hinzugefügt, bewegen sich im Netz und werden ggf. auch aus dem Netz verdrängt, wenn andere Dokumente mehr nachgefragt werden.

Jeder Peer verwaltet eine Routing-Tabelle, in der für eine Menge von Nachbarn eingetragen ist, welche Schlüssel diese enthalten. Für jeden Nachbarn wird so eine kleine Menge repräsentativer Schlüssel gespeichert. Eine Anfrage wird nun immer in Richtung des Peers weitergeleitet, für den der ähnlichste Schlüssel in der Routing-Tabelle gespeichert ist. Auf diese Weise wird eine verteilte Tiefensuche realisiert. Findet ein Peer das gesuchte Dokument in seinem lokalen Datenspeicher, so gibt er als Teil des Anfrageresultats das Dokument an den Nachbarn zurück, von dem er die Anfrage erhalten hat. Dabei vermerkt er in seiner Antwort, dass er das gesuchte Dokument enthielt. Jeder Peer auf dem Rückweg zum Anfragesteller wird dann das gesuchte Dokument weiterleiten, zusammen mit einem Vermerk, von wem das Dokument geliefert wurde. Nun wird jeder Peer auf dem Rückweg das Dokument in seinen Cache einfügen. Um die Herkunft des Dokuments zu verschleiern, kann außerdem jeder Peer auf dem Rückweg entscheiden, sich selbst als Quelle des Dokuments auszugeben.

Während man davon ausgeht, dass gerade häufig nachgefragte Dokumente in vielen Knoten gecacht sind, kann man wie bei Gnutella nach einer Freenet-Suche nie sicher sein, dass ein nicht gefundenes Dokument auch wirklich nicht im Netz vorhanden ist. Diesen Mangel beheben die weiter unten vorgestellten DHTs, jedoch unter Verzicht auf die Anonymitätsfunktionen von Freenet.

Initial Introduction, Join: Ein Peer, der mit einem Freenet-Netz in Verbindung treten will, benötigt hierzu nur die Adresse eines Freenet-Peers. Die größte Schwierigkeit bei einer solchen Verbindung ist, auf konsistente Weise dem neuen Freenet-Peer einen Schlüssel zuzuordnen. Dieser Schlüssel entscheidet, welche Anfragen der neue Peer bevorzugt erhält, und somit auch welche Daten er cachen wird. Dass der Peer den Schlüssel selbst wählt, steht im Widerspruch mit der Server-Anonymität. Clarke stellt in [4] ein kryptographisches Protokoll vor, bei dem mehrere Peers konsistent einen zufälligen Schlüssel für einen Peer wählen.

Leave: Mit einem Knoten, der das Netz verlässt, verschwinden ggf. Dokumente aus dem Netz. Häufig verwendete Dokumente werden jedoch mit großer Wahrscheinlichkeit gecacht sein und somit dem Netz nicht verloren gehen.

2.4 Chord

Einordnung: *verteilte Indexstruktur, schlüsselbasiert, mehrstufiges Routing, flaches Netz*

Chord [18] ist eine verteilte Hashtabelle oder *distributed hash table* (DHT, vgl. hierzu [1]). In Chord wird jeder Peer und jeder verwaltete Eintrag mittels einer 160 Bit langen Zahl identifiziert, die über eine Hashfunktion gewonnen werden kann. Die Peers werden in einem logischen Ring angeordnet, so dass jeder Peer einen linken Nachbarn hat, dessen ID kleiner ist, und einen rechten Nachbarn, dessen ID größer ist (Additionen erfolgen dabei modulo der Ringgröße). Jeder Eintrag, also jedes zu verwaltende Objekt, wird auf dem Peer verwaltet, der die kleinste ID hat, die größer oder gleich der ID des Objekts ist.

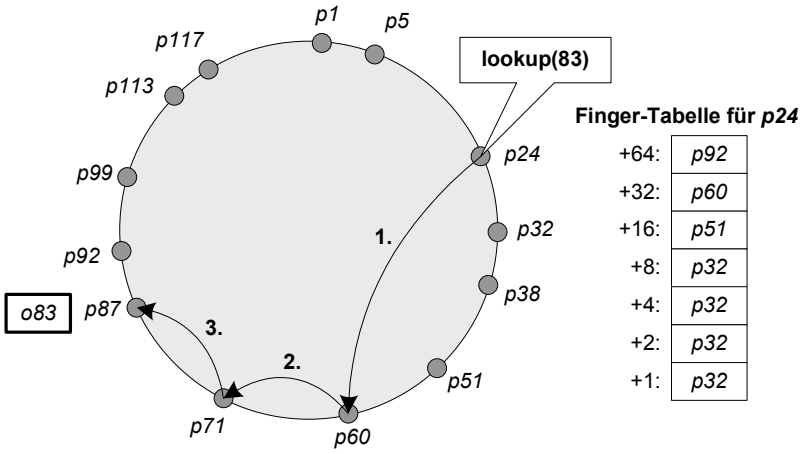

Abbildung 2 — Ein Chord-Ring und der Ablauf einer Anfrage in diesem Ring

In Abbildung 2 ist ein solcher Ring für 7-Bit IDs gegeben (IDs entstammen also der Menge $\{0, 1, \ldots, 127\}$). Peers sind durch dunkelgraue Kreise auf dem Ring dargestellt. In unserem Beispiel wird das Objekt mit ID *o83* auf dem Peer mit ID *p87* verwaltet (die vorangestellten Buchstaben p und o dienen lediglich der Unterscheidung zwischen Peers und Objekten).

Der so aufgebaute Ring von Peers ermöglicht es, Objekte durch lineare Suche nach IDs zu finden. Um die Effizienz zu steigern, verwaltet jeder Peer eine so genannte *Finger-Tabelle*. Diese Tabelle enthält Adresse und ID eines Peers auf der Hälfte, einem Viertel, Achtel *etc.* des Kreisumfangs. In unserem Beispiel ist die Finger-Tabelle für Peer *p24* angegeben. Bei einer Suche leitet nun jeder Peer eine Anfrage zu seinem direkten Nachfolger im Ring weiter, es sei denn, die ID des gesuchten Objekts ist größer als die ID eines Peers in der Finger-Tabelle. In diesem Fall wird die Anfrage an den Peer in der Finger-Tabelle geschickt, dessen ID gerade noch kleiner als die ID des gesuchten Objekts ist. Die Abbildung zeigt den Ablauf einer Suche nach dem Objekt mit der ID *o83* ausgehend von Peer *p24*. Es lässt sich zeigen, dass die Anzahl der Hops über die eine Anfrage weitergeleitet wird, logarithmisch mit der Zahl der Peers im Chord-Ring ansteigt.

In der hier vorgestellten Form geht Chord davon aus, dass jeder Peer seinen Nachfolger kennt. Wenn mehrere Nachbarn im Ring gleichzeitig ausfallen, ist dies nicht mehr gegeben. Eine Lösungsmöglichkeit ist, jeden Peer zusätzlich zur Finger-Tabelle eine Liste speichern zu lassen, die m direkte Nachfolger enthält.

Initial Introduction, Join: Initial Introduction und Join werden dadurch bewerkstelligt, dass ein Peer mit einer gegebenen ID p_{neu} einen beliebigen Peer kontaktiert. Dieser findet nun mittels des beschriebenen Chord-Lookup-Mechanismus seinen Nachfolger und seine *Finger*.

Zusätzlich ist dann ein Austausch von ID/Wert-Paaren erforderlich. Wenn wir den Vorgänger von p_{neu} mit p_{pred} bezeichnen und den Nachfolger mit p_{succ}, dann wird p_{neu} für alle ID/Wert-Paare zuständig, für die $p_{neu} \geq ID > p_{pred}$ gilt. Für diese ID/Wert-Paare war vorher p_{succ} zuständig, sie müssen also von p_{succ} zu p_{neu} transferiert werden.

Leave: Das Protokoll, das beim Join von Peers zum Auffinden korrekter Finger benutzt wird, kann auch verwendet werden, um nach dem Ausstieg von Peers die Finger-

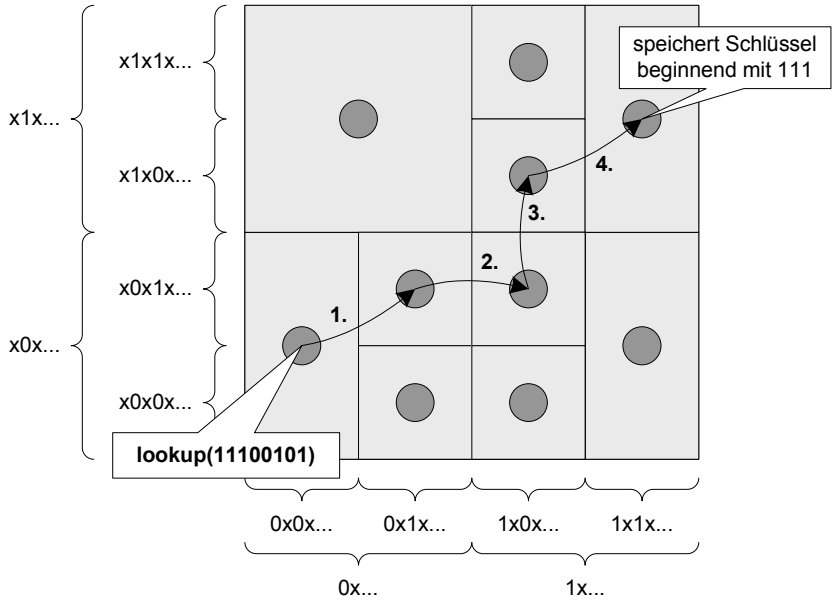

Abbildung 3 — 2-dimensionaler Koordinatenraum eines CANs

Tabellen auf einem korrekten Stand zu halten. Hierzu wird die oben beschriebene Suche nach Fingern von allen Peers periodisch durchgeführt. Der Transfer von ID/Wert-Paaren ist jedoch der eigentlich problematische Teil beim Leave von Peers. Hier muss durch geeignete Replikation dem Verlust von ID/Wert-Paaren vorgebeugt werden.

2.5 CAN

Einordnung: *verteilte Indexstruktur, mehrstufiges Routing, ggf. content-basiert*

CANs [17] bieten eine alternative Implementierung von DHTs. Schlüssel in einem CAN sind Koordinaten in einem d-dimensionalen kartesischen Raum, genauer auf einem d-dimensionalen Hypertorus. Ein in einem CAN gespeicherter Vektor kann in jeder Komponente Werte zwischen 0 und 1 annehmen. Der Torus ist nun in Hyperquader unterteilt, so genannte Zonen. Ähnlich Chord ist jeder Peer für eine Zone des Schlüsselraums zuständig. Jeder Peer verwaltet eine Routing-Tabelle mit den Adressen derjenigen Peers, die für die jeweils benachbarten Zonen zuständig sind.

Im Beispiel in Abbildung 3 sind die Schlüssel Punkte in einem zweidimensionalen Raum. Sie werden hier durch *Interleaving* aus normalen binären Schlüsseln gewonnen. Die Bits an den geraden Positionen werden verwendet um den x-Wert zu errechnen und die Bits an den ungeraden Positionen werden verwendet um den y-Wert zu errechnen. Ein Schlüssel wird dann in der Zone gespeichert, die für den gewonnenen (x, y)-Wert zuständig ist.

Bei der Bearbeitung einer Anfrage (q_x, q_y) wird die Anfrage jeweils so weitergeleitet, dass die Zone des Empfängers dem Anfragepunkt möglichst nahe ist. Hierbei wird die Anfrage in einem N Peers großen Netz $\mathcal{O}(d \cdot N^{\frac{1}{d}})$ Male weitergeleitet. In unserem Beispiel ergibt sich für $d = 2$ somit $O(\sqrt{N})$.

Initial Introduction, Join: Um dem Netz einen neuen Knoten hinzuzufügen, wird ähnlich Chord zunächst die Zone bestimmt, für die der Knoten zuständig sein wird. Nehmen wir an, der neue Peer habe eine ID p_{neu}, die in der aktuell p_z zugeordneten Zone liegt. In diesem Fall wird die Zone von p_z so geteilt, dass eine Hälfte der Schlüssel in der neu zu bildenden Zone von p_{neu} und eine Hälfte der Schlüssel in der neuen, verkleinerten Zone von p_z liegt. Danach werden die Schlüssel/Wert-Paare von p_z nach p_{neu} transferiert und die Verbindungen zu den Nachbarn der neuen Situation angepasst. Etwaige, in p_{neu} bereits enthaltene Schlüssel/Wert-Paare, die nicht in der Zone von p_{neu} liegen, sind natürlich in das CAN einzufügen.

Leave: Wenn ein Knoten das CAN geregelt verlässt, gibt er seine Zone und die entsprechenden Daten explizit an einen Nachbarn weiter. Fällt ein Knoten aus, dann erkennen die Nachbarn dies dadurch, dass der ausfallende Peer keine periodischen Updates mehr verschickt. Diese Nachbarn ermitteln durch ein einfaches Protokoll, welcher von ihnen die kleinste Zone hat, und derjenige mit der kleinsten Zone wird die Zone des ausgefallenen Peers übernehmen.

Das beschriebene Verfahren lässt CANs für eine Ähnlichkeitssuche auf Vektoren geeignet erscheinen, sofern man die Komponenten der Vektoren direkt als Koordinaten im CAN verwendet. Für geringe Dimensionalitäten d erscheint ein solcher Ansatz durchaus viel versprechend, jedoch wird es bei hoher Dimensionalität der zu indexierenden Vektoren ($d > 10$) aufgrund des *Curse of Dimensionality* notwendig für eine exakte Bearbeitung einer Ähnlichkeitsanfrage praktisch alle Peers zu kontaktieren.

Ein allgemeines Problem von DHTs ist, dass ein Peer beim Eintritt in das Netz einen Anteil von $\frac{N-1}{N}$ seiner Schlüssel/Wert-Paare an andere Peers weiterleiten muss und seinerseits einen Anteil von $\frac{1}{N}$ der gesamten im Netz enthaltenen Schlüssel/Wert-Paare zugesandt bekommt. Dieses Problem gewinnt bei CANs mit hochdimensionalen Daten durch den Umfang der Schlüssel (Vektoren) noch an Bedeutung. Trotz der zu erwartenden Kommunikationslast wählen [19, 8] den Ansatz zur Indexierung von Vektoren. Wir gehen auf [19] weiter unten ein.

2.6 Canon: hierarchische DHTs

Einordnung: *hierarchisches Netz*

In den letzten Abschnitten wurden verschiedene Implementierungen von DHTs vorgestellt. Sollen diese Ansätze im Internet-Maßstab angewendet werden (d.h. mit $\gg$ 10.000 Knoten), sind Modifikationen an den Verfahren notwendig. Einige Ziele solcher Modifikationen können Fehlerisolation, effizientes Caching, effiziente Bandbreitennutzung, Anpassung des P2P-Netzes an das gegebene physikalische Netz sowie eine hierarchisch organisierte Zugriffskontrolle sein. Alle diese Punkte lassen sich in einer *flachen* DHT nicht oder nur mit unverhältnismäßigem Aufwand realisieren, sind in hierarchischen DHTs jedoch relativ einfach zu implementieren.

In [16] stellen die Autoren einen *Canon* genannten Ansatz vor, mit dem sich existierende Implementierungen von DHTs in eine hierarchische Struktur überführen lassen.

Im Folgenden wird eine solche Transformation anhand von Chord beschrieben, dessen hierarchische Variante in [16] *Crescendo* genannt wird. Voraussetzung für die Transformation ist eine Hierarchie, deren Blätter die zu vereinenden Chord-Ringe sind. Dabei ist es nicht notwendig, dass die einzelnen Peers die gesamte Hierarchie kennen.

Vielmehr muss jeder Peer p_a nur seine Position in der Hierarchie kennen und zu jedem anderen Peer p_b den engsten gemeinsamen Vorfahren in der Hierarchie berechnen können, also denjenigen inneren Knoten der Hierarchie, der beide Peers unter seinen Nachfahren hat und dabei am weitesten unten in der Hierarchie angeordnet ist. Hierzu reicht es aus, jedem Peer einen Namen wie etwa im Domain Name Service (DNS) zu geben; in einem solchen Ansatz können zwei Peers leicht ermitteln, welche Domain beiden gemeinsam ist.

Die einzelnen Chord-Ringe an den Blättern der Hierarchie werden nun sukzessive miteinander verschmolzen, so dass jeder innere Knoten der Hierarchie einen Chord-Ring repräsentiert, der eine Verschmelzung aller Chord-Ringe im von diesem inneren Knoten ausgehenden Teilbaum darstellt. Eine Verschmelzung mehrerer Ringe lässt sich als Folge einer Verschmelzung von jeweils zwei Ringen darstellen.

Betrachten wir die beiden Chord-Ringe R_1 und R_2. Diese sollen miteinander verschmolzen werden. Jeder Ring beinhaltet eine Anzahl von Peers, deren IDs aus dem Bereich $\left[0, 2^{160} - 1\right]$ stammen. Alle Peers der beiden Ringe werden nun in einen gemeinsamen Ring aufgenommen. Dabei stellt ein Peer p aus R_1 eine Verbindung zu einem Peer p' aus R_2 her, falls die folgenden Bedingungen zutreffen: (1) p' ist der nächste Knoten im gemeinsamen Chord-Ring, dessen Entfernung mindestens 2^k beträgt (für $0 \leq k < 160$), und (2) p' ist näher an p als alle Knoten aus R_1.

Dies bedeutet, dass jeder Peer p in R_1 zunächst eine normale Finger-Tabelle für diesen Ring R_1 verwaltet. Falls zwischen p und dem nächsten Peer in R_1 (der auf jeden Fall in der Finger-Tabelle von p referenziert wird) ein Peer aus R_2 liegt (Bedingung 2), wird dieser in die Finger-Tabelle aufgenommen falls er auch Bedingung 1 erfüllt. Nehmen wir unser Beispiel aus Abbildung 2 an und betrachten den dort abgebildeten Ring als R_1. Wenn nun in R_2 Peers $p27$, $p29$ und $p30$ enthalten wären, würde der Eintrag für $+1$ in der Finger-Tabelle von $p24$ zu $p27$, der Eintrag für $+2$ ebenfalls zu $p27$ und der Eintrag für $+4$ zu $p29$, während alle anderen Einträge unverändert blieben.

Nach der beschriebenen Ergänzung der Finger-Tabellen können Anfragen genauso geroutet werden wie zuvor: Nachrichten für einen bestimmten Peer werden immer zu demjenigen Peer weitergereicht, dessen ID möglichst nahe an der Ziel-ID, aber nicht größer als diese ist. Dabei wird durch die Struktur der Finger-Tabellen die Anfrage immer zunächst im lokalen Ring bis zu der Stelle geleitet, wo sie entweder beantwortet werden kann oder wo ein Übergang in andere Ringe erfolgen muss.

Der beschriebene Ablauf geht dabei davon aus, dass ID/Wert-Paare im vereinigten Ring übergreifend den jeweils passenden Peers zugeordnet werden. Mit geringen Modifikationen bei der Suche ist aber auch eine Verwaltung der ID/Wert-Paare in den jeweils lokalen Ringen möglich. Schließlich lassen sich in das System auch Sichtbarkeitsregeln und Zugriffsschutzmechanismen auf Basis der Hierarchie natürlich integrieren.

Neben der beschriebenen Transformation einzelner Chord-Ringe in eine Hierarchie von Chord-Ringen, die auf jeder Ebene der Hierarchie voll funktionsfähig sind und bleiben, existieren ebenfalls Transformationen für CANs (*Can Can*), Symphony (*Cacophony*) und Kademlia (*Kandy*).

3 Verteiltes IR: Indexstrukturen

Im vorangegangenen Kapitel haben wir einige wesentliche Ansätze zur schlüsselbasierten Suche kennen gelernt. Nun werden wir darauf aufbauend Ansätze zur inhaltsbasierten

Suche betrachten. Einen ersten Block bilden verteilte Indexstrukturen, die die Verbindungen zwischen den Peers strukturieren und die Daten so auf die Peers verteilen, dass die resultierende Datenstruktur die effiziente Verarbeitung von Ähnlichkeitsanfragen erlaubt.

3.1 pVSM: Verteilte Invertierte Listen

Einordnung: *verteilte Indexstruktur, inhaltsbasiert, Text*

Tang *et al.* präsentieren in [19] zwei Ansätze zum Retrieval in P2P-Systemen. Einer von ihnen (pVSM, *Peer Vector Space Model*) verwendet DHTs, um verteilte invertierte Listen zu erzeugen.

Eine invertierte Liste enthält für einen gegebenen Term t_i (ein gegebenes Wort) eine Liste mit Verweisen auf alle Dokumente D_j, die t_i mindestens einmal enthalten. Bei einer Ähnlichkeitsanfrage, die aus Anfragetermen $t_{q_1}, \ldots, t_{q_m}$ besteht, werden für jedes t_{q_ℓ} diejenigen D_j aufgefunden, die t_{q_ℓ} mindestens einmal enthalten. Für diese Dokumente wird auf Basis der Häufigkeit des Anfrageterms in D_j und der Häufigkeit des Anfrageterms in der Datensammlung ein Score $S(D_j, t_{q_\ell})$ für D_j bezüglich t_{q_ℓ} berechnet. Die Summe der Scores über die einzelnen Anfrageterme für ein Dokument D_j ist dann der gesamte Score des Dokuments bezüglich der Anfrage $S(D_j, \{t_{q_1}, \ldots, t_{q_m}\}) = \sum_{\ell=1}^{m} S(D_j, t_{q_\ell})$.

Dieses Prinzip ist einfach auf DHTs übertragbar: Jeder Knoten in der DHT wird für einige Terme t_i zuständig und enthält somit eine Liste mit Referenzen auf alle Dokumente, die t_i enthalten. Eine Referenz besteht jedoch hier aus Dokument-ID und Peer-Adresse. Würde man Listen für hochfrequente Begriffe mit in die Betrachtung einbeziehen, ergäben sich durch den Umfang der Listen Probleme hinsichtlich der Speicherplatzanforderungen und des Kommunikationsaufwands. Tang *et al.* betrachten daher diese häufigen Begriffe nicht und zeigen für ihren Ansatz gute empirische Resultate. Li *et al.* [12] geben jedoch eine pessimistische Prognose für die Skalierbarkeit des Ansatzes.

Initial Introduction, Join: Ein Peer, der seine Daten dem Netz bereitstellen will, muss zunächst lokale invertierte Listen berechnen. Dann findet er mithilfe der DHT für jeden Term, der in seinen Dokumenten vorkommt, den zuständigen Knoten und schickt diesem zuständigen Knoten die entsprechende Liste. Gleichzeitig wird er als Teil des DHT-Protokolls für einige Schlüssel (also Terme) zuständig und bekommt von seinen Nachbarn einige Terme sowie die zugehörigen invertierten Listen zugesandt.

Leave: Beim geordneten Verlassen des Netzes wird ein Peer alle Peers, die Indexdaten von ihm enthalten, im DHT kontaktieren und eine Löschung dieser Indexdaten veranlassen. Ebenso wird er die invertierten Listen, die er enthält, gemäß dem DHT-Protokoll an einen oder mehrere Nachbarn weitergeben.

Bei einem ungeordneten Verlassen des Netzes werden zweierlei Dinge passieren: erstens wird ein Teil der im Netz enthaltenen Indexdaten unbrauchbar, da sie auf einen nicht existierenden Peer verweisen. Zweitens wird der Peer, der das Netz verlässt, eine oder mehrere invertierte Listen mit sich aus dem Netz nehmen. Dem muss durch Redundanz im DHT begegnet werden.

3.2 pLSI verteiltes Latent Semantic Indexing im CAN

Einordnung: *verteilte Indexstruktur, inhaltsbasiert, mehrstufiges Routing*

Latent Semantic Indexing (LSI) ist eine Methode, die Dimensionalität der Term/Dokument-Matrix zu reduzieren. Eine Datensammlung wird hier als eine Matrix von Gewichten w_{td} gesehen, in der einem Term t das Dokument D mit dem Gewicht w_{td} zugeordnet wird. Der Gedanke bei LSI ist nun, dass die Matrix $W = w_{td}$ ein Modell der Semantik der Datensammlung ist, das jedoch zu detailliert ist. Die Hoffnung ist, dass eine zu W ähnliche Matrix W' mit einer niedrigeren Dimensionalität (also niedrigerem Detailgrad) die Semantik der Datensammlung besser ausdrückt.

Vor der Transformation sind die Dokumente typischerweise durch dünn besetzte Vektoren repräsentiert, danach durch dicht besetzte Vektoren der Länge 1. Die Dimension dieser Vektoren ist niedriger, aber immer noch im Bereich von 100.

Bei einer Anfrage wird die Anfrage als ein Punkt im Raum der transformierten LSI-Vektoren gesehen. Bei der Anfragebearbeitung wird der Winkel zwischen dem Anfragevektor und den Dokumentvektoren berechnet. Diejenigen Dokumente, die der Anfrage am ähnlichsten sind, haben einen LSI-Vektor mit möglichst kleinem Winkel zum Anfragevektor.

Da die LSI-Vektoren alle auf der Einheitskugel liegen, ist die Datenverteilung schlecht an die Indexierung in CANs angepasst. pLSI transformiert die in kartesischen Koordinaten gegebenen LSI-Vektoren in Polarkoordinaten, um sie einfacher für CANs indexierbar zu machen.

Eine weitere Eigenschaft der LSI-Vektoren ist, dass die Dimensionen nach Wichtigkeit geordnet sind. pLSI gruppiert die Dimensionen und indexiert verschiedene Dimensionengruppen in verschiedenen CANs. Die Ähnlichkeitssuche nach hochdimensionalen Vektoren wird so in mehrere niederdimensionale Suchen aufgeteilt. Die so gewonnenen Resultate sind eine Näherung der Resultate, die man mit einer zentralen LSI erhalten würde.

Tang *et al.* berichten von beeindruckenden Resultaten auf Textdaten. Nur wenige Knoten müssen für hohe Präzision (relativ zur zentralisierten Verarbeitung von Anfragen) kontaktiert werden.

Initial Introduction, Join: pLSI erwartet, dass eine LSI bereits gegeben ist. Tatsächlich ist dies jedoch einer der Schwachpunkte von pLSI, weil die Berechnung einer LSI für sehr viele Dokumente extrem aufwendig ist. Ebenso ist das konsistente Nachführen der Daten bei dynamischen Datenbeständen eine herausfordernde Aufgabe. Ferner ist der Kommunikationsaufwand bei einem Eintritt in das Netz hoch. In einem Netz mit 100.000 Peers und 1.000 Dokumenten pro Peer würde der Eintritt eines Peers Nachrichten im Umfang von ca. 20 MB verursachen. Kritisch ist dabei insbesondere die Tatsache, dass ein Nutzer, der dem Netz keine Daten zur Verfügung stellt, sofort Anfragen an das Netz stellen kann. Dem Netz Dokumente zur Verfügung zu stellen, wird also in Form von Kommunikationsaufwand bestraft.

Einen anderen Ansatz zur Ähnlichkeitssuche auf Beschreibungsvektoren mithilfe von CANs beschreibt [8]. Dabei werden in den einzelnen Peers neben Verweisen auf angrenzende Nachbarn auch Verweise auf einige zufällig gewählte oder gezielt ausgesuchte weiter entfernt liegende Peers verwaltet, um die Suche zu beschleunigen. Einen Ansatz, bei dem eine Baumstruktur im P2P-Netz verteilt wird findet sich z.B. bei [2].

Dabei werden die Buckets und die inneren Knoten des Baumes mit einer kontrollierten Redundanz auf die Peers verteilt.

3.3 Distributed PageRank

Einordnung: *inhaltsbasiert, verteilte Indexstrukturen, mehrstufiges Routing, Text*

PageRank (siehe [11]) ist ein von der Suchmaschine Google her bekanntes Verfahren zur Indexierung und Bewertung von Dokumenten im WWW. Zentrales Element hierbei ist der ebenfalls als *PageRank* bezeichnete Wert, der einem Dokument zugeordnet wird und dessen Position in der Antwort auf eine Suchanfrage bestimmt.

Der PageRank eines Dokuments ist abhängig von der Zahl der *Inlinks*, also der Anzahl der Hyperlinks, die von anderen Dokumenten auf dieses Dokument zeigen. Gleichzeitig erhöht ein Inlink, der von einem Dokument mit hohem PageRank kommt, den PageRank des betrachteten Dokuments stärker als ein Inlink von einem niedrig bewerteten Dokument. Dokumente geben ihren PageRank über ihre *Outlinks*, also die von ihnen ausgehenden Hyperlinks, an die referenzierten Dokumente weiter. Dabei wird der PageRank gleichmäßig zwischen allen Outlinks aufgeteilt. Diese Struktur kann man auch als Gleichungssystem auffassen, das einer iterativen Lösung zugänglich ist.

Distributed PageRank [10] verteilt die Berechnung des PageRank in einem P2P-Netz. Typischerweise wird es sich bei dem P2P-Netz um eine Implementierung einer DHT handeln, wie etwa Chord oder CAN. Jeder Knoten im Netz ist dann für diejenigen Dokumente im WWW zuständig, deren ID (z.B. ein *cryptographic hash* der URL) in seinen Schlüsselraum fällt. Liegen auf den Peers nur Referenzen (URLs) auf die eigentlichen Dokumente vor, so muss ein Peer vor der Indexierung eines Dokuments dieses erst anhand der vorliegenden Referenz beschaffen. Gleichzeitig unterhalten die Peers invertierte Listen im Stile von pVSM, um die Anfragebearbeitung durchführen zu können. In diesen invertierten Listen wird zu einzelnen Begriffen neben den IDs der Dokumente, die diesen Begriff enthalten, auch der PageRank dieser Dokumente verwaltet. Zusätzlich kann auch noch die Termfrequenz mitgeführt werden. Eine besondere Stärke des Ansatzes ist, dass Änderungen (z.B. neu publizierte oder gelöschte Dokumente) nicht wie beim original PageRank-Verfahren zu einer vollständigen Neuberechnung des PageRanks führen, sondern durch eine begrenzte Anzahl von Update-Nachrichten behandelt werden können. Das verteilte Gleichungssystem erreicht sehr schnell wieder einen stabilen Zustand.

Anfragen in einem P2P-Netz mit Distributed PageRank werden inkrementell bearbeitet. Der anfragende Peer p kreiert die aus m Suchbegriffen $t_{q_1}, t_{q_2}, \ldots, t_{q_m}$ bestehende Anfrage und ermittelt, welcher Peer im Netz die invertierte Liste für den ersten Suchbegriff t_{q_1} verwaltet. An diesen Peer p_1 verschickt er die Anfrage. p_1 bestimmt anhand der vorliegenden invertierten Liste zu t_{q_1}, welche Dokumente t_{q_1} enthalten und sortiert diese absteigend nach ihrem PageRank. p_1 entfernt den Begriff t_{q_1} aus der Suchanfrage und sendet die verkürzte Anfrage $t_{q_2}, \ldots, t_{q_m}$ zusammen mit den ersten $X\%$ der so gerankten Dokumente an den Peer p_2 weiter, der die invertierte Liste für den Begriff t_{q_2} enthält. p_2 entfernt aus der erhaltenen Liste von Dokumenten alle diejenigen, die den Term t_{q_2} nicht enthalten, entfernt t_{q_2} aus der Liste der Suchbegriffe und schickt die verkürzte Suchanfrage mit der Liste der bisher matchenden Dokumente an den Peer p_3 weiter, der für den Begriff t_{q_3} zuständig ist. Durch dieses inkrementelle Vorgehen wird der Netzverkehr gegenüber anderen Modellen zur Anfrageverarbeitung stark verringert.

Ein mögliches Anwendungsszenario nennen die Entwickler von Distributed PageRank in [10]: Die Web-Server im WWW (oder zumindest eines Teils davon) bilden die Knoten eines P2P-Netzes. Die Peers verwenden Distributed PageRank, um alle erreichbaren Dokumente im WWW zu indexieren. Laut Berechnungen in [10] ist für die initiale Berechnung des PageRanks für alle erreichbaren WWW-Dokumente eine Zeit von etwa 35 Tagen anzusetzen. Danach werden Änderungen (neue, geänderte und gelöschte Dokumente) über Update-Nachrichten behandelt, so dass der verteilte Index in diesem P2P-Netz immer auf einem aktuellen Stand ist – eine Leistung, die keine aktuelle Suchmaschine bieten kann.

4 Verteiltes IR: Routing

Die im vorherigen Abschnitt vorgestellten Systeme versuchen Daten geschickt im Netz zu verteilen und das Netz zu strukturieren, um schnell und effizient Ähnlichkeitsanfragen durchführen zu können. Die jetzt vorzustellenden Systeme gehen einen anderen Weg: die Indexdaten verbleiben im Peer, jedoch wird versucht, die zur Abfrageverarbeitung versandten Messages so zu routen, dass sie bevorzugt Peers erreichen, die nützliche Daten enthalten. Diese Systeme zeichnen sich vor allem dadurch aus, dass Joins billiger als Anfragen sind. Dies ist ein wichtiger Faktor, um Peers dazu zu bewegen, nicht nur Dienste vom P2P-Netz nachzufragen, sondern auch eigene Daten bereitzustellen.

4.1 DISCOVIR

Einordnung: *inhaltsbasiert, semantisches Routing*

DISCOVIR ist ein Ansatz für Ähnlichkeitsanfragen in P2P-Netzen. Ng and Sia [15] modifizieren den Flooding-Ansatz von Gnutella behutsam, um Ähnlichkeitsanfragen durchzuführen. Ihr Modell der Anfragebearbeitung heißt Firework Query Model (FQM).

Die erste Veränderung ist, dass jeder Knoten eine Kurzzusammenfassung seiner eigenen Daten enthält. Ein Peer kann aufgrund dieser Zusammenfassung schnell entscheiden, ob er die Anfrage lokal bearbeiten oder nur weiterleiten wird. Ferner gibt es in DISCOVIR zwei verschiedene Arten von Links, die so genannten *random links* (normale Gnutella-artige Links) sowie bevorzugte Links, die so genannten *attractive links*. Peers werden über attraktive Links verlinkt, wenn ihre Zusammenfassungen ähnlich sind.

Erhält ein Peer eine Anfrage, die seiner lokalen Zusammenfassung *ähnlich* ist, bearbeitet er diese Anfrage lokal und leitet sie über attraktive Links an andere Peers weiter. Eine Anfrage, die der lokalen Zusammenfassung *unähnlich* ist, wird *nicht* lokal bearbeitet und nur über normale Links an Nachbarn weitergeleitet, wobei die TTL verringert wird.

Dieses Modell verwendet die normalen Links um Cluster von Peers zu erreichen, deren Zusammenfassung der Anfrage ähnlich ist. Diese Cluster werden dann mithilfe der attraktiven Links erschöpfend besucht.

Zwei Dinge führen also zu einer Effizienzsteigerung: Erstens werden gegenüber vollständigem Flooding des Netzes weniger Botschaften verschickt und zweitens wird nicht jede Anfrage auch lokal verarbeitet. Dies ist wichtig, da gerade Ähnlichkeitsabfragen nach hochdimensionalen Daten aufwändig im Hinblick auf Festplattenzugriffe und Prozessorlast sind.

4.2 PlanetP: Source Selection und Rumor Spreading

Einordnung: *inhaltsbasiert, Selektion, flaches Netz*

Ebenso wie das Firework Query Model wird im PlanetP-System [6] versucht, die Anzahl der zu bearbeitenden Anfragen je Peer zu minimieren. Zusätzlich wird erreicht, dass Anfragen ausschließlich an diejenigen Peers versandt werden, bei denen davon ausgegangen wird, dass sie zum Anfrageresultat etwas beizutragen haben. Um diese Peers auswählen zu können, enthält jeder Peer im Netz Zusammenfassungen zu allen anderen Peers.

Ein Peer p, der eine Anfrage seines Benutzers bearbeitet, wird zunächst die Anfrage auf die lokal in ihm gespeicherten Zusammenfassungen anwenden. Ergebnis ist ein Ranking der Peers nach der Wahrscheinlichkeit, mit der sie ein interessantes Ergebnis enthalten. p wird dann die am besten gerankten Peers kontaktieren und ihnen die Anfrage stellen. Die so befragten Peers geben die Anfrage *nicht* weiter, sondern bearbeiten sie nur lokal und geben das Ergebnis dieser Anfrage an p zurück.

Diese Zusammenfassungen werden durch so genanntes Rumor Spreading verteilt: Periodisch gibt jeder Peer neue Zusammenfassungen die er erhalten hat, an einige zufällig ausgewählte Nachbarn weiter. Hierbei wird jede neue Zusammenfassung nach relativ kurzer Zeit (für realistische Szenarien im Minutenbereich) in das gesamte Netz verteilt. Gleichzeitig ist die Kommunikationsbelastung sehr viel kleiner, als sie wäre, wenn jeder Knoten Veränderungen sofort per Broadcast im Netz verteilen würde.

Natürlich wird die Effizienz der Anfragebearbeitung in PlanetP entscheidend dadurch beeinflusst, welche Zusammenfassungen verwendet werden. Hierauf gehen wir im folgenden Abschnitt ein. Ein anderes wichtiges Problem ist, dass PlanetP in dieser Form natürlich nicht skalierbar ist: Die Zahl der zu befragenden Knoten ist ungefähr proportional zur Gesamtzahl der Knoten im Netz. Ferner ist die Zahl der Zusammenfassungen, die von jedem Peer verteilt, verwaltet, und gespeichert werden müssen, gleich der Zahl der Peers im Netz; PlanetP ist also nicht auf Internet-Maßstab skalierbar.

Initial Introduction, Join: Um einem PlanetP-Netz beizutreten, benötigt der neue Peer p nur die Adresse eines schon im Netz befindlichen PlanetP-Peers p_1. p teilt p_1 seine Zusammenfassung mit, und erhält von p_1 dessen Zusammenfassungen. Man beachte, dass p nur eine einzige Zusammenfassung versenden muss, um das Netz über seine Daten zu unterrichten, jedoch viele Zusammenfassungen empfangen muss, um sinnvoll anderen Knoten im Netz Anfragen stellen zu können.

Leave: Wenn ein Knoten das Netz verlässt, nimmt er nicht mehr am Rumor-Spreading-Prozess teil. Zusammenfassungen, die zu alt sind, werden von den einzelnen Knoten aus ihrer Liste gelöscht.

Zusammenfassungen für PlanetP

Wie weiter oben bereits angeführt, ist die Wahl des richtigen Typs von Zusammenfassungen bei PlanetP entscheidend. Effektivität und Effizienz des Ansatzes werden von dieser Wahl beeinflusst.

Die Entwickler von PlanetP gehen von einem Szenario aus, in dem Anwender Textdokumente über ein PlanetP-Netz austauschen wollen. Eine nahe liegende Zusammenfassung für diesen Zweck ist die Anwendung von Bloom-Filtern auf die lokale Kollektion

eines Peers. Bloom-Filter [3] sind eine komprimierte, verlustbehaftete Repräsentation von Mengen. Sie unterstützen effizient Abfragen, ob ein Element in einer Menge vorhanden ist. Sie sind verlustbehaftet: Anfragen können fälschlich positiv beschieden werden. Im Falle eines PlanetP-Netzes zur Suche nach Textdokumenten würde also ein Bloom-Filter für die lokale Kollektion eines Peers erzeugt und per Rumor Spreading an die anderen Peers verteilt. Diese könnten dann beim Bearbeiten einer suchbegriffbasierten Anfrage effizient feststellen, welche Peers im Netz Dokumente mit den gesuchten Begriffen enthalten. Allerdings kann es – neben den bereits erwähnten *false positives* – noch zu einem anderen Fehler kommen: Wird nach den Begriffen t_{q_1} und t_{q_2} gesucht und ein Bloom-Filter sagt aus, dass ein Peer p sowohl t_{q_1} als auch t_{q_2} enthält, so bedeutet dies noch nicht, dass p beide Begriffe im selben Dokument enthält. Statt dessen könnten auch zwei Dokumente bei p vorliegen, von denen eines t_{q_1} und das andere t_{q_2} enthält.

Sollen Dokumente im P2P-Netz verteilt werden, die neben Text auch andere Medientypen enthalten, muss auf andere Zusammenfassungen zurückgegriffen werden. In [14] wird eine Möglichkeit vorgestellt, wie in einem PlanetP-Netz multimediale Daten (insbesondere Bilder) gesucht werden können. Hierzu werden als Zusammenfassungen so genannte Cluster-Histogramme verwendet. Dabei werden mit einem verteilten k-means-Algorithmus die auf den Peers vorliegenden Bilder in k Cluster eingeteilt. Jeder Peer kann danach angeben, wie viele seiner Bilder in welchen Cluster fallen. Das Cluster-Histogramm kann dann als Zusammenfassung an andere Peers verteilt werden. Sollen Bilder gesucht werden, die einem Anfragebild ähnlich sind, so wird für das Anfragebild bestimmt, in welchen Cluster es fällt. Danach werden zunächst Peers kontaktiert, die zu diesem Cluster viele Bilder enthalten.

4.3 Rumorama: Skalierbares PlanetP

Einordnung: *inhaltsbasiert, Selektion, hierarchisches Netz*

Die bisher dargestellten Ansätze litten jeweils unter einem von zwei Problemen: Entweder war eine inhaltsbasierte Suche nicht oder nur mit großem Aufwand möglich (so bei den DHTs) oder das Verfahren skaliert nur sehr begrenzt (PlanetP). Im Folgenden wollen wir das Protokoll Rumorama [13] vorstellen, das auf skalierbare zusammenfassungsbasierte Suche in P2P-Netzen ausgelegt ist. Um dies zu erreichen, wird bei Rumorama ein robustes hierarchisches Netz aus PlanetP-Netzen aufgebaut. Die Blätter dieses Netzes sind PlanetP-Netze und werden Blattnetze (*leaf nets*) genannt. Über die inneren Knoten der Rumorama-Hierarchie kann auf die Blattnetze zugegriffen werden.

Ein typisches Rumorama-Netz ist in Abbildung 4 zu sehen. Wie in anderen Netzen wird auch in Rumorama jeder Peer über eine ID beschrieben. Der anfragende Peer in Abbildung 4 hat die ID 01011000 ... . Dieser Peer befindet sich in einem Blattnetz (angedeutet durch das graue Rechteck mit gestrichelter Begrenzung) und tauscht mit allen 18 anderen Peers in diesem Blattnetz Zusammenfassungen aus. Alle Peers in diesem Blattnetz haben eine ID, welche mit 01 beginnt. Genauer gesagt hat jeder Peer eine so genannte *friends mask*, welche das gemeinsame Präfix der IDs aller Freunde dieses Peers ist (d.h. aller Peers im selben Blattnetz). In unserem Beispiel ist die *friends mask* 01. Um auch mit Peers in anderen Blattnetzen kommunizieren zu können, hält ein Peer zusätzlich noch Informationen zu so genannten *Nachbarn* vor. Im Gegensatz zu Freunden werden für Nachbarn keinerlei Zusammenfassungen gespeichert. Im Beispiel hat der Peer zwei Nachbarn für jedes Bit seiner *friends mask*: für das erste Bit einen

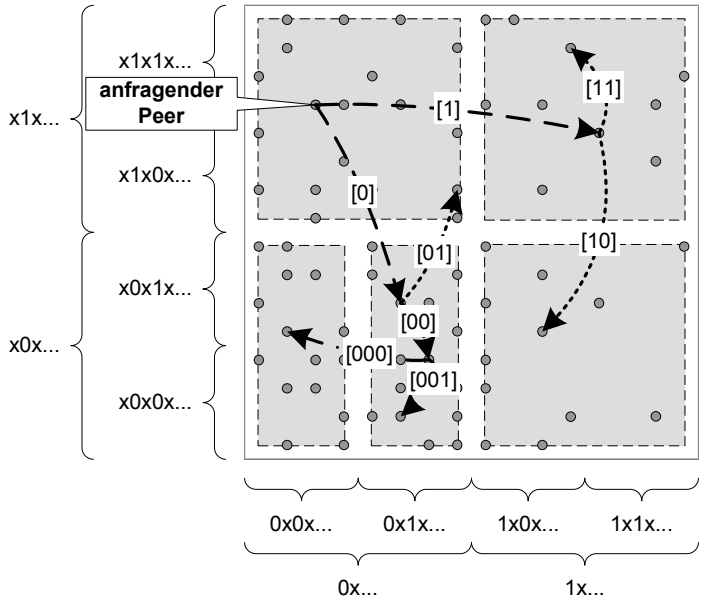

Abbildung 4 — Beispiel für die Verteilung einer Ähnlichkeitsanfrage zu allen Blattnetzen in Rumorama

Nachbarn, dessen ID mit 0 beginnt und einen, dessen ID mit 1 beginnt; für das zweite Bit einen Peer, dessen ID mit 00 beginnt und einen, dessen ID mit 01 beginnt. In einem realen System ist es notwendig, pro Bit Informationen zu mehreren Peers zu unterhalten, um in einem dynamischen Umfeld mit vielen Joins und Leaves operieren zu können. Darüber hinaus kann jeder Peer unabhängig von anderen Peers über die Länge seiner *friends mask* entscheiden und diese auch zur Laufzeit ändern. Hierdurch wird es möglich, die Anzahl der Freunde innerhalb gewisser Grenzen zu halten.

Wenn ein Peer eine Anfrage im Netz verteilen möchte, ist es wichtig, dass diese Anfrage jedes Blattnetz erreicht. Um dies zu garantieren, versendet der anfragende Peer seine Anfrage an einen Nachbarn, dessen ID mit 0 beginnt, und an einen Nachbarn, dessen ID mit 1 beginnt. In der Nachricht wird kodiert, wie lang das bereits im Verteilungsprozess berücksichtigte Präfix ist (im ersten Schritt ist dieser Parameter also 1). Ein empfangender Peer überprüft die Länge des bereits berücksichtigten Präfixes. Ist diese Länge $\geq$ der Länge seiner *friends mask*, so ist ein Blattnetz erreicht: diesem Peer liegen lokal alle Zusammenfassungen von Peers mit dem erforderlichen Präfix vor, und er kann — analog zu PlanetP — eine Anfragebearbeitung unter den Peers starten, die dem bereits berücksichtigten Präfix entsprechen. Ist hingegen die in der Anfrage kodierte Präfixlänge kürzer als die *friends mask* eines Peers, leitet dieser die Anfrage an zwei entsprechende Nachbarn weiter, indem er an das bereits berücksichtigte Präfix eine 0 bzw. eine 1 anhängt und den entsprechenden Parameter in der Anfrage um eins erhöht.

In Abbildung 4 ist eine solche Suchanfrage skizziert. Der anfragende Peer leitet die Anfrage an zwei seiner Nachbarn weiter. Bei beiden Nachbarn ist noch kein Blattnetz erreicht. Daher wird die Anfrage nochmals an jeweils zwei weitere Peers weitergeleitet. Die jeweils in den Nachrichten kodierten Präfixe sind in Abbildung 4 in eckigen Klammern dargestellt. Zu beachten ist, dass dieser zweite Schritt bereits parallel bearbeitet wird, genauso wie alle folgenden Schritte. Für einige Peers, nämlich diejenigen

mit den Präfixen 11, 10 und 01, ist in diesem zweiten Schritt ein Blattnetz erreicht. Diese Peers können die Anfrage aufgrund der bei ihnen lokal vorliegenden Zusammenfassungen der anderen Peers in ihrem jeweiligen Blattnetz in einem PlanetP-artigen Verfahren bearbeiten. Nur die Nachricht mit dem kodierten Präfix 00 hat noch kein Blattnetz erreicht, daher muss diese Nachricht an zwei Peers weitergeleitet werden, deren IDs mit 000 und 001 beginnen. Im nächsten Schritt ist auch dort ein Blattnetz erreicht, so dass dort ebenfalls die lokale Anfragebearbeitung beginnen kann.

Angenommen, der anfragende Peer ist an den k am besten zur Anfrage passenden Dokumenten interessiert, dann wird in jedem Blattnetz eine Liste der k besten Treffer berechnet. Die Ergebnisse eines Blattnetzes werden über den gleichen Pfad zurückgemeldet, über den die Suchanfrage das Blattnetz erreicht hat. Die inneren Knoten des durch die Suchanfrage aufgebauten Baums vereinen die bei ihnen eintreffenden Ergebnisse aus den Blattnetzen und melden ihrerseits wiederum k Treffer entlang des Pfads weiter.

Dieser Ansatz kann den *Curse of Dimensionality* nicht überwinden. Allerdings erlaubt er eine skalierbare Implementierung von PlanetP-artigen Netzen. Jeder Peer muss nur für eine begrenzte Anzahl von Peers (seine *Freunde*) Zusammenfassungen verwalten und kann die Anzahl dieser Freunde selbstständig und dynamisch über die Länge seiner *friends mask* anpassen. Die Verteilung von Anfragen geschieht durch die inhärente Parallelität in logarithmischer Zeit. Für die lokale Anfragebearbeitung in den Blattnetzen können verschiedene Heuristiken angewendet werden, die jeweils mehr Wert auf Ergebnisqualität oder Antwortzeit legen. Beispielsweise könnte nur ein begrenzter Anteil an Peers in jedem Blattnetz kontaktiert werden, etwa diejenigen, die aufgrund ihrer Zusammenfassung am wahrscheinlichsten relevante Dokumente enthalten.

5 Ausblick

In diesem Artikel haben wir einige wichtige Ansätze zur Indexierung von Daten in P2P-Systemen vorgestellt. Für eindimensionale Schlüssel wie für Vektordaten existieren effiziente Indexstrukturen. Jedoch bleiben auch viele offene Probleme.

Ein wichtiges Problem in P2P-Netzen sind Teilnehmer, die nur Services nachfragen, jedoch nicht oder nur wenig zur Bearbeitung von Anfragen beitragen, die so genannten *free riders*. Ein leistungsfähiges, skalierbares System sollte hiergegen gesichert sein. Ein Gegenstand der Forschung sind auch P2P-Systeme, die eine Abrechnung nachgefragter Services gestatten.

Ausfallsicherheit stellt ein weiteres Problemfeld dar. Die hier vorgestellten Verfahren sind sehr verschieden hinsichtlich ihrer Empfindlichkeit gegen Ausfälle von Peers. Dieses Problem lässt sich in der Regel durch gezielte Replikation abmildern.

Schließlich bilden die *Sicherheit gegen Angriffe* und die *Effizienz* der Systeme weitere bedeutende Forschungsbereiche.

Literatur

[1] Hari Balakrishnan, M. Frans Kasshoek, David Karger, Robert Morris, and Ion Stoica. Looking Up data in P2P Systems. *Communications of the ACM*, 46(2):43–48, February 2003.

[2] M. Batko, C. Gennaro, and P. Zezula. A Scalable Nearest Neighbor Search in P2P Systems. In *2nd International Workshop on Databases, Information Systems and Peer-to-Peer Computing*. LNCS, August 2004. To appear.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 1970.

[4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, 2000.

[5] Clip2. The Gnutella Protocol Specification v0.4.
URL: `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`, 2000.

[6] Francisco Matias Cuenca-Acuna and Thu D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. Technical Report DCS-TR-483, Department of Computer Science, Rutgers University, 2002.

[7] Stefan Decker, Mario Schlosser, Michael Sintek, and Wolfgang Nejdl. HyperCuP - Hypercubes, Ontologies and Efficient Search on P2P Networks. In *International Workshop on Agents and Peer-to-Peer Coputing*, Bologna, Italy, July 2002.

[8] Paresan Ganesan, Beverly Yang, and Hector Garcia-Molina. One torus to rule them all: multi-dimensional queries in P2P systems. In *7th International Workshop on the Web and Databases*. ACM, 2004.

[9] Sebastian Goeser. Relevanzranking. *Datenbank-Spektrum*, 10:46–49, August 2004.

[10] Karthikeyan Sankaralingam and Simha Sethumadhavan and James C. Browne. Distributed PageRank for P2P Systems. In *12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), 22-24 June 2003, Seattle, WA, USA*, pages 58–69. IEEE Computer Society, 2003.

[11] Lawrence Page and Sergey Brin and Rajeev Motwani and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[12] Jinyang Li, Boon Thau Loo, Joseph M. Hellerstein, M. Frans Kaashoek, David R. Karger, and Robert Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In M. Frans Kaashoek and Ion Stoica, editors, *IPTPS*, volume 2735 of *Lecture Notes in Computer Science*, pages 207–215. Springer, 2003.

[13] Wolfgang Müller, Martin Eisenhardt, and Andreas Henrich. Scalable summary-based search in P2P networks. zur Veröffentlichung eingereicht, 2004.

[14] Wolfgang Müller and Andreas Henrich. Fast Retrieval of High-Dimensional Feature Vectors in P2P Networks Using Compact Peer Data Summaries. In *ACM MIR'03 Workshop*, Berkeley, CA,USA, 2003.

[15] C. H. Ng and K. C. Sia. Peer clustering and firework query model. In *Poster Proc. of The 11th Interational World Wide Web Conf.*, Honululu, HI, USA, May 2002.

[16] Prasanna Ganesan and Krishna Gummadi and Hector Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *24th International Conference on Distributed Computing Systems (ICDCS 2004), 24-26 March 2004, Hachioji, Tokyo, Japan*, pages 263–272. IEEE Computer Society, 2004.

[17] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proc. 2001 Conf. on applications, technologies, architectures, and protocols for computer communications*, San Diego, CA, United States, 2001.

[18] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. ACM SIGCOMM Conf.*, San Diego, CA, USA, 2001.

[19] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. pSearch: Information retrieval in structured overlays. In *First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, 2002.

[20] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 49–58. IEEE Computer Society, 2003.