Klassifikation von Namens- und Lokalisierungsdiensten in dezentralen verteilten Systemen

Thorsten Strufe

Technische Universität Ilmenau
thorsten.strufe@tu-ilmenau.de

Zusammenfassung Bei Namens- und Lokalisierungsdiensten handelt es sich um essentielle Funktionen dezentraler verteilter Systeme. Obwohl eine große Anzahl von Forschungsprojekten zu Peer-to-Peer-Systemen existiert, deren Arbeit sich gerade auf die Identifikation und Lokalisierung von Objekten konzentriert, finden sich wenig Ansätze, die diese Dienste grundsätzlich analysieren und klassifizieren. Der Versuch einer solchen Untersuchung und Einteilung soll mit dieser Arbeit unternommen werden, um zukünftigen Projekten eine Hilfestellung bei der Auswahl eines passenden Verfahrens zu geben.

1 Einleitung

Bedeutende Kernfunktionen in verteilten Systemen sind die Namens- und Lokalisierungsdienste. Insbesondere in dezentralen verteilten Systemen, vor allem in den Peer-to-Peer-Systemen, stellen sie einen Großteil des Entwicklungsaufwandes dar: Ist eine Ressource (etwa aktive verarbeitende Komponenten, Speicher, Datenobjekte etc.) in einem Suchraum erst einmal identifiziert und lokalisiert, ist es in der Regel verhältnismäßig einfach, auf diese zuzugreifen.

Soll ein Lookup-Mechanismus für dezentrale verteilte Systeme ausgewählt werden, so ist das Ausmaß existierender Ansätze auf den ersten Blick erschlagend. Im Umfeld der für den Austausch von Audiodaten entwickelten Filesharing-Systeme gibt es eine Vielzahl unterschiedlichster Projekte, die an der Entwicklung der Suche und der Nachrichtenweiterleitung arbeiten. Bei genauerer Betrachtung kann jedoch festgestellt werden, daß sich die Änderungen anhand charakteristischer Merkmale in wenige Gruppen einteilen lassen.

In diesem Beitrag werden die einzelnen Projekte vorgestellt und untersucht um anschließend eine Klassifikation möglicher Verbesserungen und ihrer Auswirkungen aufzustellen.

Im zweiten Teil der Arbeit werden zunächst einige grundlegende Begriffe vorgestellt. Der dritte Teil enthält eine Einführung in die Problematik der Namensauflösung in verteilten Systemen. Im vierten Teil werden einige mögliche Klassifizierungsmerkmale vorgestellt, anhand derer die einzelnen Entwicklungen im fünften Teil gruppiert werden, bevor der sechste Teil nach einer kurzen Zusammenfassung einen Ausblick auf zukünftige Forschungsarbeiten gibt.

2 Grundlegende Begriffe

Zunächst sollen einige bekannte Begriffe, die für die vorliegende Arbeit von besonderer Bedeutung sind, eingeführt werden.

2.1 Namen

Adressen sind Zugriffspunkte für Ressourcen und dienen der direkten örtlichen (physischen) Lokalisierung derselben. Da Adressen sich häufig ändern können, hat eine Ressource in der Regel zusätzlich einen Bezeichner (auch „pure names“[12] oder „identifier“), über den die aktuelle Adresse aufgelöst werden kann. Ein Bezeichner ist genau einer Ressource zugeordnet und dient als erste Indirektionsstufe zur Abstraktion von der Adresse, um Adresstransparenz zu erreichen. Bezeichner haben folgende Eigenschaften[19]:

1. Ein Bezeichner ist höchstens einer Ressource zugeordnet.
2. Jeder Ressource ist genau ein Bezeichner zugeordnet.
3. Ein Bezeichner ist immer derselben Ressource zugeordnet (er wird kein zweites mal genutzt).

Die Relation zwischen Bezeichnern und Ressourcen ist folglich eine injektive Abbildung.

Der Begriff des Namens ist grundsätzlich eine Generalisierung für alle Verweise auf Ressourcen, worunter sowohl Adressen als auch Bezeichner fallen. Da die strikten Einschränkungen für Bezeichner und ihr in der Regel maschinennahes Format die Nutzbarkeit stark einschränken, werden als weitere Indirektionsstufe „other names“[12] eingeführt, für die im weiteren der Begriff „Name“ stehen soll. Zum einen können zusätzlich zum ursprünglichen, weitere, als Alias-Namen bezeichnete Namen auf die gleiche Ressource verweisen. Zum anderen können in getrennten Kontexten unterschiedliche Ressourcen gleiche Namen haben.

Zur Strukturierung von Namen werden Namensräume in Form gerichteter Graphen mit attributierten Knoten aufgespannt. Innerhalb dieser Graphen wird zwischen Kontext- und Namensknoten unterschieden, wobei durch Kontextknoten ein gerichteter Graph aufgebaut und in den Attributen der Namensknoten der Name und gegebenenfalls die Adresse einer Ressource abgelegt wird. Innerhalb von Namensräumen werden Namen in der Regel nach bestimmten Vereinbarungen, einem Namenssystem, vergeben. Enthält das Namenssystem die strikten für Bezeichner geltenden Regeln, so hat der Namensraum eine Baumstruktur. Sind Alias-Namen erlaubt, so kann es zu Schleifen kommen, und aus der Baumstruktur wird ein einfacher gerichteter Graph.

Die Beschreibung durch Eigenschaften und Meta-Informationen, mittels derer mehrere Ressourcen gleicher oder ähnlicher Art zusammen gefasst werden können, bildet eine letzte Abstraktionsstufe.

Gibt es innerhalb eines Systems mehrere unterschiedliche Adressräume, wird für den Zugriff auf eine Ressource ein zusammengesetzter Verweis auf Ressource und Adressraum, in welchem sie angesprochen werden kann, benötigt. Dieser wird als Referenz bezeichnet.

2.2 Verteilung

Dezentrale verteilte Systeme bestehen in der Regel aus lose gekoppelten Komponenten, deren Hinzukommen und Verlassen nicht vorhersehbar, im besonderen Falle sehr dynamisch ist. In dezentralen verteilten Systemen kennt jede Komponente nur eine eingeschränkte Anzahl anderer Komponenten. Diese können Anfragen und Aufgaben an weitere, der Quelle unter Umständen unbekannte, Komponenten delegieren und bei Bedarf Nachrichten an diese weiterleiten. Innerhalb dieser Systeme ist es nicht möglich, eine vollständige Kenntnis über enthaltene Komponenten und deren Lokalität, respektive über die Struktur des Systems zu erhalten. Aufgrund der Dynamik kann es keine dedizierten Instanzen geben, in welchen Informationen endgültig abgelegt werden können.

Um eine höhere Verfügbarkeit, geringere Belastung einzelner Ressourcen und kurze Kommunikationswege in grossen Systemen zu erreichen, können Replikate von Ressourcen erzeugt werden. Hierbei handelt es sich um Kopien von Ressourcen, also der Applikation identisch erscheinende Datenobjekte, die sich für diese nur durch den Ort ihrer Speicherung unterscheiden. Replikate einer Ressource sind in der Regel durch gleiche Bezeichner referenzierbar. Die Tatsache, daß eine Ressource in mehreren Replikaten vorliegt, bleibt für die Applikation transparent. Für sie ist das

Replikat nicht als solches erkennbar, sie greift lediglich auf die benötigte Ressource zu.

2.3 Namens- und Lokalisierungsdienste

Um Objekte mit bekannten Namen zu finden, muss die zugehörige Abbildung in dafür zuständigen, als Lookup-Dienst bezeichneten, Komponenten abgespeichert sein. Ausgehend von der eingeführten Abstraktionshierarchie ist es einleuchtend, daß es zur Auflösung jeder der Indirektionsstufen spezialisierter, den jeweiligen Namensraum implementierender, Lookup-Dienste bedarf:

- Lokalisierungsdienste bilden Bezeichner auf Referenzen ab;
- Namensdienste bilden Namen auf Bezeichner ab.

Um Objekte zu finden, von denen bestimmte Eigenschaften, aber weder Name noch Bezeichner bekannt sind, können zusätzlich „Discovery-“ oder „Trading-Services“ eingesetzt werden.

Grundsätzlich gibt es für die Realisierung von Namens- und Lokalisierungsdiensten zwei Suchstrategien:

1. Ablage und Nachschlag in einer zentralen Registrierung,
2. Durchsuchen des gesamten Systems.

Im ersten Fall wird der Namensraum an einer zentralen Stelle implementiert. Alle Informationen werden an dieser Stelle registriert und können dort daraufhin abgerufen werden.

Im zweiten Fall ist der Namensraum verteilt implementiert. Jede Komponente ist damit für die Namensauflösung der durch sie angebotenen Objekte zuständig. Bei der Suche wird das gesamte System mit der Anfrage geflutet, woraufhin jede Komponenten prüft, ob ihre Objekte den gewünschten Eigenschaften, respektive der Name oder Bezeichner den jeweils gesuchten entspricht. Ist die Struktur eines Systems feststellbar, so können Pfade hindurch gelegt werden, entlang derer die Suchanfragen weitergeleitet werden.

3 Lookup in dezentralen, verteilten Systemen

Die Implementierung der vorgestellten Suchstrategien in Namens- und Lokalisierungsdiensten ist in dezentralen verteilten Systemen ungleich schwerer. Mangels vollständiger Kenntnis über Struktur und Umfang ist es weder möglich, eine dedizierte Instanz zur Registrierung aller Ressourcen festzulegen und diese Information an alle Komponenten zu verteilen.

Noch können eindeutige Suchpfade, entlang derer alle Ressourcen erreichbar sind, durch das System gelegt werden.

Für das zweite Problem bestehen zwei Lösungsmöglichkeiten. Entweder wird eine Struktur anhand von außen vorgegebener Informationen künstlich erzeugt, oder es muß durch hohe Redundanz der Nachrichten (etwa die Weiterleitung an jeweils alle bekannten Nachbarkomponenten) sichergestellt werden, daß die Anfrage bei jeder Komponente zumindest einmal ankommt.

Ein weiteres Problem ist die Tatsache, daß es in dezentralen Systemen zu uneinheitlicher Namensgebung kommen kann, und daher unterschiedliche Namenssysteme existieren können. Dies ist insbesondere im Hinblick auf die Existenz von Replikaten problematisch, da diese den gleichen Bezeichner besitzen müssen, wenn auf die Einführung von Replika-Managern verzichtet werden soll.

3.1 Fluten

Die einfachste Möglichkeit zur Flutung des Systems mit Anfragen, wie sie etwa in der Version 0.4 von Gnutella[2] realisiert ist, besteht darin, jede bei einer Komponente eingehende Anfrage lokal zu prüfen und an alle bekannten Komponenten, mit Ausnahme der anfragenden, weiterzuleiten. Auf diese Weise wird der gesamte Suchraum abgedeckt und die gesuchte Ressource gefunden.

Vorteil dieser Lösung ist, daß durch die jeweils lokale Bearbeitung der Anfragen recht einfach auch eine unscharfe Suchen realisiert und so neben einer Abbildung von Namen sowohl auf Bezeichner und Referenzen auch eine erweiterte Namensauflösung stattfinden kann.

Ohne gemeinsames Namenssystem ist eine eindeutige Identifikation und somit die Detektion von Replikaten allerdings nicht möglich. Zusätzlich führt diese Vorgehensweise bei einer steigenden Anzahl von Komponenten zu einem in Abhängigkeit von der Anzahl bekannter Komponenten exponentiell steigendem Kommunikationsaufkommen[17] und damit in großen Systemen zwangsläufig zum Kollaps.

3.2 Registrierung

Um eine Registrierung von Ressourcen zu ermöglichen, muß zunächst eine Komponente ausgewählt werden, die den Namensraum implementiert. Denkbare Lösungen wären externe, manuelle Eingriffe oder die Zuhilfenahme eines Wahlalgorithmus'. In sehr dynamischen oder sehr großen

Systemen sind diese Verfahren allerdings nicht praktikabel: Systemexterne Eingriffe, da sie einen sehr hohen und unbefriedigenden Wartungsbedarf nach sich ziehen und Wahlalgorithmen, da sie zu einem immensen Kommunikationsaufkommen führen und die erlangte Information zu schnell veraltet.

Da eine zentrale Vergabe folglich keine Lösung darstellt, wird ein generisches und verteilt implementierbares Namenssystem für Ressourcen und Komponenten eingeführt, welches ohne zentrale Koordination auskommt. Hierbei handelt es sich in der Regel um die Berechnung von Hash-Werten als Abbildung. Daraufhin wird die Tatsache, daß auch Komponenten Bezeichner haben und diese nach dem gleichen Namenssystem erstellt werden, für eine Abbildung der Ressourcen auf die für sie zuständigen Komponenten genutzt. Um eine Anfrage zur Komponente mit dem benötigten Bezeichner weiterleiten zu können, wird anhand der Bezeichner eine abstrakte Struktur über das System gelegt, mittels derer die Nachricht geroutet wird.

So bilden durch verteilte Hash-Tabellen (distributed hashtables, DHT) realisierte *Inhalts-Adressierbare-Netzwerke*[16] eine Lösung.

Der ursprüngliche DHT-Ansatz war für die Veröffentlichung von Dokumenten gedacht und sah vor, diese vollständig bei der zuständigen Komponente zu speichern. Da es in multimedialen Systemen zu hohem Datenaufkommen kommt und eine Suche nach allgemeinen Ressourcen, also beispielsweise auch nach weiterverarbeitenden Komponenten wünschenswert ist, wird das Verfahren hier abgewandelt und anstelle eines Dokumentes eine Referenz auf die Ressource abgelegt.

Vorteile dieses Ansatzes sind die inhärente Lastverteilung durch die Aufteilung der Registrierung auf alle Komponenten, sowie das durch die Möglichkeit, die Anfragen direkt zu der zuständigen Komponente zu routen, geringere Kommunikationsaufkommen.

Es ergeben sich allerdings auch einige Nachteile. Angesichts der Tatsache, daß eine nicht ähnlichkeitserhaltende Namenstransformation stattfindet, ist keine erweiterte Suche möglich, da ähnliche Namen auf vollkommen unterschiedliche Hash-Werte abgebildet werden. Problematisch ist auch die Auswahl des zu hashenden Attributes: werden die Namen gehasht, so ist das Suchergebnis von der unkoordinierten Namensgebung abhängig. Durch hashen inhaltsspezifischer Merkmale wird es jedoch zum reinen Lokalisierungsdienst. Ein weiteres Problem ergibt sich, wenn Komponenten das System ungeordnet verlassen. In diesem Fall kann es dazu kommen, daß sie die bei ihnen registrierten Informationen nicht zur weiteren Verfügung bei anderen Komponenten registrieren, was dazu führt,

daß diese daraufhin verloren sind. Darüberhinaus können im Falle eines ungünstigen Routings durch die Strukur sehr lange Suchwege notwendig sein.

4 Klassifizierungsmerkmale

Die Nachteile des reinen Flutens und der einfachen DHT haben auf beiden Strecken zu einer Vielzahl an Überarbeitungen geführt.

Ein Großteil der resultierenden Vorschläge können dabei auf einige wenige Änderungen reduziert werden. Diese lassen sich in grundlegende Gruppen einteilen, anhand derer die zur Zeit prominenten Systeme klassifiziert werden sollen. Ausgehend von den Flutmechanismen kommt es dabei häufig zu Hybridlösungen, in denen an zu bestimmenden Stellen registriert[5][6], oder wie bei Projekt jxta[7] und Futella[20], eine Struktur eingeführt wird. Auf Basis der DHT wird vor allem versucht, durch eine bessere Strukturierung höhere Suchfreiheitsgrade zu erreichen.

Über die Erweiterungen hinaus ist jedoch vor allem die Einteilung nach der Funktionalität ein grundlegendes Klassifizierungsmerkmal.

4.1 Namens- vs. Lokalisierungsmechanismen

Wie in Abschnitt 3 bereits beschrieben, eignen sich flutende Mechanismen sowohl für die erweiterte Auflösung von Namen als auch von Bezeichnern gut. Zur Einführung eines erweiterten Namensdienstes, welcher, um die Benutzbarkeit zu steigern, über die exakte Suche hinaus ähnliche Namen als Ergebnis liefert, muss in jeder Lookup-Komponente eine unscharfe Suche realisiert sein.

DHT eignen sich nicht für die Implementierung solcher erweiterter Namensdienste. Sollen sie als Namensdienste genutzt werden, müssen die Namen der Ressourcen gehasht werden. In diesem Fall besteht bei erweiterter Suche die Ergebnismenge aufgrund der Tatsache, daß das Hashen als gewählte Abbildung nicht ähnlichkeitserhaltend ist, aus Verweisen auf vollkommen unterschiedliche Ressourcen, deren Namen ähnliche Hash-Werte ergeben. Replikate mit ähnlichen, aber unterschiedlichen Namen sind in diesem Fall in der Ergebnismenge nicht enthalten. Die DHT eignen sich folglich lediglich für die Realisierung von exakten Namens- oder besser von Lokalisierungsdiensten.

4.2 Weiterleitung

Aufgrund der Tatsache, daß das bei dem ursprünglichen Gnutella notwendige vollständige Fluten des Systems zur Nachrichtenexplosion und da-

mit zum Kollaps führt, sind einige Einschränkungen notwendig, um soviel überflüssige Kommunikation wie möglich zu vermeiden. Bei den auf DHT beruhenden Systemen wird durch ein besseres Routing der Suchanfragen versucht, eine Verkürzung der Suchpfade zu erreichen.

Tiefe der Suche Existieren in großen Systemen Replikate, so muss eine Suchanfrage unter Umständen nicht das gesamte System durchlaufen, um eines davon zu entdecken. Zur Realisierung eines eingschränkten Suchhorizontes, welcher zur Verminderung des Kommunikationsbedarfs führt, werden mittels *Time-To-Live-Feldern* (TTL) maximale Lebenszeiten von Nachrichten realisiert. Findet sich innerhalb des gegebenen Suchhorizontes kein befriedigendes Ergebnis, kann die Suche mit einem erweiterten Suchhorizont wiederholt werden („Expanding Ring“).

Auch in verteilten Hash-Tabellen kann die Anzahl der notwendigen Sprünge bis zur Zielkomponente durch eine Verbesserung des Routing der Anfragen gesenkt werden.

Da das reine Ring-Routing, welches etwa CHORD[8] zugrunde liegt, ebenso wie das zweidimensionale Netz-Routing von CAN[16] zu sehr langen Suchpfaden führen kann, wurde mit Plaxton[15], Pastry[18] und Tapestry[21] ein Prä-, beziehungsweise Suffix-Routing eingeführt. Die einzelnen Komponenten der resultierenden Systeme bauen sich Routing-Tabellen mit Empfängerkomponenten für jeden möglichen Wert einer Stelle des Hash-Wertes auf. Eine Suchanfrage wird daraufhin Stelle für Stelle von Komponente zu Komponente weitergeleitet, um am Ende bei der zuständigen Instanz anzukommen.

Ein anderes Routingverfahren, basierend auf einer XOR-Metrik, verfolgt Kademlia[11], auf dem „Overnet“[14] aufbaut. Hierbei sind alle Komponenten abstrakt in einem globalen binären Baum angeordnet. Jede muß zumindest eine Komponente aus den parallelen Ästen dieses Baumes kennen. Anfragen werden an die Komponente mit dem per XOR-Operation ermitteltem ähnlichsten Bezeichner, der also die geringste Anzahl abweichender Stellen zum Suchstring aufweist, weitergeleitet, bis die zuständige Komponente erreicht wird.

Breite der Suche In einem System, in welchem jede Anfrage von jeder empfangenden Komponente an alle bekannten Komponenten weitergeleitet wird, erhöht sich mit der Anzahl der Verbindungen die Anzahl der Nachrichten exponentiell. Zusätzlich kann es zur Maschenbildung und damit zu dem mehrfachen Eintreffen einer Anfrage bei derselben Komponente kommen. Wird die Anzahl der Verbindungen auf eine zu ermit-

telnde Mindestanzahl eingeschränkt, so kann davon ausgegangen werden, daß bei weiterhin nahezu vollständiger Abdeckung des Suchraums die Maschenbildung abnimmt.

Eine naheliegende Lösung für die Nachrichtenexplosion ist somit die manuelle Einschränkung aktiver Verbindungen und die Einführung maximaler Lebenszeiten (siehe Abschnitt 4.2) für Nachrichten. Diese hat bei Gnutella 0.4 zu einer drastischen Verringerung des Kommunikationsbedarfs, allerdings auch zu einer deutlichen Verschlechterung der Trefferqualität (14% Trefferwahrscheinlichkeit[20]) geführt.

Da die Einschränkung aktiver Verbindungen zusätzlich zu einer Verringerung der Ausfallsicherheit und Broadcasts zu redundanten Nachrichten führen, wird in vielen Arbeiten versucht, dadurch Verbesserungen zu erzielen, daß Anfragen nur an ausgewählte, erfolgversprechende Komponenten weitergeleitet werden.

Eine einfache, aber sehr effiziente Methode für diese selektive Weiterleitung ist die Einführung der „Random Walker“[10]. Hier sendet die suchende Komponente ihre Anfrage nur an eine beschränkte Auswahl an Knoten, die diese an jeweils nur eine Komponente, die zufällig ausgewählt wird, weiterleiten. Da die Anzahl der Nachrichen pro Anfrage im Verlauf nicht steigt, kann die Lebensdauer und damit der Suchhorizont erhöht werden. Um die Trefferwahrscheinlichkeit zu erhöhen, werden alle Walker mit Bezeichnern versehen und diese in jeder weiterleitenden Komponente gespeichert, um gleiche Anfragen bei erneutem Empfang an eine andere Komponente senden zu können.

Um einen anderen Ansatz handelt es sich bei der Einführung von Reputation[4, 13, 3]. Die Ergebnisse aus Suchanfragen werden als Indiz für die Kompetenz der benachbarten Komponenten, respektive der durch sie erreichbaren Systemteile, gespeichert und bei zukünftigen Suchanfragen berücksichtigt.

Als letzter Ansatz der selektiven Weiterleitung muss die Einführung von „Supernodes“, die in 4.3 detailliert beschrieben sind, betrachtet werden.

4.3 Struktur

Eine grundlegende Verbesserungsmöglichkeit ist die Verkürzung der Suchwege durch Einführung oder Veränderung logischer Strukturen. Dies kann durch dynamische Hierarchien, manuell anzulegende Gruppen oder die Erhöhung der Suchfreiheitsgrade durch Verbesserung der Routing-Mechanismen innerhalb der DHT erreicht werden.

Hierarchie Eine Möglichkeit, viele Komponenten zu entlasten, ist die Einführung zentraler Supernodes, welche sowohl Knotenpunkte für die Kommunikation bilden, als auch von den anderen Komponenten für Look-up-Operationen und als Registrierungsinstanzen genutzt werden. Jede Suchanfrage wird direkt an einen der Supernodes gestellt. Dieser leitet sie an die anderen Supernodes weiter, welche sie mit den bei ihnen gespeicherten Informationen vergleichen und gegebenenfalls mit Erfolgsnachrichten antworten.

Auf diese Weise wird lediglich das Teilsystem der Supernodes geflutet, während die restlichen Komponenten einfache Registrierungs- und Nachschlageoperationen ausführen können.

Da zu Beginn in dezentralen Systemen keine Supernodes existieren, müssen diese mittels lokal begrenzter Wahlalgorithmen oder aufgrund ihrer Ressourcen dynamisch bestimmt werden. Ausgewählte Systeme, welche auf Supernodes basieren, sind etwa der Clip2Reflector für Gnutella 0.4[1], Fasttrack (KaZaA)[6] und Gnutella 0.6[9]

Gruppen Eine andere Möglichkeit, dezentrale Systeme zu strukturieren, ist die interessenbasierte Gruppenbildung, wie sie etwa bei jxta[7] eingesetzt wird. Interessengruppen können zum Zeitpunkt des Eintritts von Ressourcen manuell oder auf Basis von Eigenschaften der Ressourcen (Meta-Informationen) gebildet werden. Grundsätzlich wird die Suche in gruppenbasierten Systemen in zwei Schritten realisiert: Als erstes wird eine Komponente lokalisiert, die Mitglied der zur gesuchten Ressource passenden Gruppe ist. Über diese Komponente wird im zweiten Schritt die gesamte Gruppe mit der Anfrage geflutet.

Durch diese Hierachie wird auch in diesem Ansatz das Fluten auf einen abgeschlossenen Teil des Systems beschränkt, was zu einer deutlich besseren Skalierbarkeit führt. Ein Problem ist die Tatsache, daß es zunächst zu sinnvoller Gruppenbildung (und damit in der Regel systemexternen Eingriffen) kommen muß. Da auch hierbei keine zentrale Koordination stattfindet, kann es aufgrund unterschiedlicher Namenssysteme zu disjunkten Gruppen der gleichen Spezialisierung kommen. Probleme bei Suchoperationen treten auch dann auf, wenn etwa aufgrund abweichender Namenssysteme oder der relativen Entfernung der suchenden Komponente vorhandene Gruppen nicht gefunden werden.

Suchgraphen In enger Verbindung mit dem gewählten Routing-Mechanismus ergibt sich für verteilte Hash-Tabellen ein zugehöriger Graph.

Ausgehend vom eindimensionale Ring bei CHORD, über ein zweidimensionales Netz bei CAN und den binären Baum von Kademlia steigt die Anzahl der Suchfreiheitsgrade bis zum n-dimensionalen Raum von Pastry, Tapestry und Plaxton, wodurch die Länge der Suchpfade stetig gesenkt wird.

4.4 Skalierbarkeit

Die durch die Einführung von Hierarchie erreichte bessere Skalierbarkeit bei flutenden Mechanismen[6, 9] ist lediglich als Abschwächung des Grundproblems zu betrachten. Bei sehr großen Systemen mit sehr vielen Anfragen muss es in der oberen Hierarchiestufe zwischen den Supernodes zu einem Kollaps kommen. Das gleiche gilt bei der Einführung von Interessengruppen[7] für die Kommunikation innerhalb derselben.

Anders ist die Einführung der selektiven Weiterleitung einzuschätzen. Bei dem Einsatz der Random Walker kommt es zu keiner Vervielfältigung von Nachrichten, die Lösung ist so als skalierbar anzusehen.

Ähnliches gilt für die reputationsbasierende Mechanismen: In stabilen Systemen sinkt die Anzahl der notwendigen Nachrichten, sobald sich die Reputation durch das System propagiert hat.

DHT-Mechanismen können grundsätzlich als skalierend angenommen werden, da sie in der Regel mit nur einer Nachricht und ebenfalls ohne Nachrichtenvervielfältigung auskommen. Zudem beruht das Design der verteilten Registrierung grundlegend auf Lastverteilung

4.5 Trefferwahrscheinlichkeit

Die Trefferwahrscheinlichkeit unterscheidet sich bei den registrierenden und den flutenden Mechanismen deutlich. Registrierende Mechanismen liefern zu jeder Anfrage nach einer existierenden und registrierten Ressource ein richtiges Ergebnis, da immer eine zuständige Komponente gefunden wird.

Flutende Mechanismen haben in großen Systemen eine deutlich geringere Trefferwahrscheinlichkeit. Wie in 4.2 beschrieben finden sie nur Ressourcen, die sich innerhalb ihres Suchhorizontes befinden. Obwohl die Einführung von Strukturen in einer deutlich erhöhten Trefferwahrscheinlichkeit resultiert (etwa 96% bei der Einführung von Interessengruppen [20]) gilt auch hier: Befindet sich die zuständige Gruppe oder der registrierende Supernode nicht innerhalb des Suchhorizontes, so bleibt die Suche erfolglos.

5 Einteilung bestehender Systeme

Eine Übersicht der Verfahren und Projekte, nach den in Abschnitt 4 beschriebenen Merkmalen klassifiziert, zeigt Tabelle 1.

Merkmal Verfahren	Basis, Auflösung	Struktur	Weiterleitung	Skalierung, Treffer
Gnutella V. 0.4	Fluten(bruteforce), erweiterter Name	keine	Broadcast, nicht selektiv, ungeroutet	schlecht, schlecht
Fasttrack Gnuttella 0.6	Fluten (Supernodes) erweiterter Name	hierarchisch (Supernodes)	Supernodes: nicht selektiv, ungeroutet	mittel, gut
jxta	Fluten (in Gruppe) erweiterter Name	hierarchisch (Gruppen)	in Gruppen selektiv, ungeroutet	mittel gut
Random Walker	Fluten (lange Trails) erweiterter Name	keine	selektiv ungeroutet	gut gut
PlanetP, Routing indices	Fluten erweiterter Name	keine	selektiv ungeroutet	gut gut
CAN	Registrieren Bezeichner	Netz	geroutet	gut sehr gut
CHORD	Registrieren Bezeichner	Ring	geroutet	gut sehr gut
Kademlia (Overnet)	Registrieren Bezeichner	Baum	geroutet xor-metric	gut, schnell sehr gut
Pastry, Tapestry, Plaxton	Registrieren Bezeichner	Baum	geroutet (prä-, suffix)	gut sehr gut

Tabelle 1. Einteilung der untersuchten Systeme

Hierbei ist gut erkennbar, daß sie in Abhängigkeit der Strukturierung, der Skalierbarkeit und der Funktionalität grob in vier Gruppen eingeteilt werden können:

1. Simple flutende Verfahren (Gnutella 0.4: Keine Struktur, skaliert nicht, erweiterte Namensauflösung)
2. Hierarchische flutende Verfahren (Fasttrack, Gnutella 0.6, jxta: strukturiert, skalieren besser, erweiterte Namensauflösung)
3. Selektiv weiterleitende Verfahren (Random Walker, PlanetP, Routing Indices: keine Struktur, skalieren gut, erweiterte Namensauflösung)
4. Verteilte Hashtable (skalieren gut, strukturiert, keine erweiterte Namensauflösung)

6 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden die aktuell prominenten Lookup-Mechanismen für dezentrale verteilte Systeme vorgestellt und klassifiziert.

Zusammenfassend ist zu bemerken, daß die Wahl eines dieser Mechanismen sehr stark von der Domäne und der benötigten Funktionalität abhängig ist.

Sollen in großen Systemen, verhältnismäßig einfach, Namen aufgelöst werden, so sind hierarchisch strukturierte, flutende Mechanismen zu wählen. Kommt hinzu, daß die Kommunikationskosten hoch sind (etwa in Systemen mit geringer Bandbreite), so sind selektiv weiterleitende Mechanismen sinnvoll. Dabei ist zu bedenken, daß sich diese beiden Ansätze problemlos kombinieren und so sehr effiziente Mechanismen realisieren lassen. Diese Verfahren eignen sich folglich sehr gut für den Einsatz auf Applikationsebene, da versehentliche Tippfehler oder die Unkenntnis über exakte Namen gut kompensiert werden und Antwortzeiten hier keine herausragende Bedeutung haben.

Sind allerdings Trefferwahrscheinlichkeit und Antwortzeiten von Interesse und sind die Bezeichner oder exakte Namen der Ressourcen bekannt, so bietet sich als Lösung der Einsatz verteilter Hashtables an. Sie eignen sich besser auf der Systemebene zum Auffinden bestimmter weiterverarbeitender Komponenten oder Dokumente, deren exakte Bezeichner bekannt oder berechenbar sind. Vorstellbar wäre bei diesen Systemen außerdem die Nutzung einer ähnlichkeitserhaltenden Abbildung anstelle des Hashings, um die Vorteile der DHT auch für eine sinnvolle erweiterte Namensauflösung nutzen zu können. Dabei ist darauf zu achten, daß die Auflösung weiterhin für die eindeutige Unterscheidung unterschiedlicher Ressourcen hoch genug ist.

Literatur

1. clip2. Clip2 reflector. `http://dss.clip2.com/reflector.html`.
2. clip2. The gnutella protocol specification v0.4. `http://rfc-gnutella.sourceforge.net/`.
3. Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 28th Conference on Distributed Computing Systems*, July 2002.
4. Francisco Matias Cuenca-Acuna, Richard P. Martin, and Thu D. Nguyen. PlanetP: Using Gossiping and Random Replication to Support Reliable Peer-to-Peer Content Search and Retrieval. Technical Report DCS-TR-494, Department of Computer Science, Rutgers University, July 2002.
5. eDonkey. `www.edonkey2000.com`.
6. FastTrack. `www.fasttrack.nu`.
7. The jxta Project. jxta. `http://overnet.com/documentation/how.html`.
8. Ion Stoica; Robert Morris; David Karger; Frans Kaashoek; and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet aplications. In *Proceedings of ACM SIGCOMM'01*, September 2001.

9. Tor Klingberg and Raphael Manfredi. The gnutella protocol specification v0.6. http://rfc-gnutella.sourceforge.net/.
10. C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks, 2001. citeseer.nj.nec.com/lv02search.html.
11. P. Maymounkov and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric, 2002. citeseer.nj.nec.com/maymounkov02kademlia.html.
12. Roger M. Needham. *Distributed Systems*, chapter Names, pages 315–327. Addison-Wesley, 1993.
13. Alpine Network. http://cubicmetercrystal.com/alpine/discovery.html.
14. Overnet. http://overnet.com/documentation/how.html.
15. C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.
16. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172. ACM Press, 2001.
17. J. Ritter. Why gnutella can't scale. no, really. http://www.darkridge.com/ jpr5/doc/gnutella.html, 2001.
18. Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov 2001.
19. Roel Wieringa and Wiebren de Jonge. Object identifiers, keys, and surrogates: Object identifiers revisited. *Theory and Practice of Object Systems*, 1(2):101–114, 1995.
20. T. Zahn, H. Ritter, J. Schiller, and H. Schweppe. Futella analysis and implementation of a content-based peer-to-peer network. In *Proceedings of the 8th International Netties Conference*, pages 15–22, September 2002.
21. B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001. citeseer.nj.nec.com/zhao01tapestry.html.