Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

Lokalisierung von Ressourcen

Naming + Lookup in unstrukturierten und strukturierten P2P-Datenverbünden

Problemseminar "Peer-to-Peer (P2P) Data Management"

Bearbeiter: Tobias Peitzsch

Betreuer: Toralf Kirsten

Leipzig, Dezember 2003

Inhaltsverzeichnis

1. Motivation, Zielstellung und Abgrenzung

In den letzten Jahren ist die Anzahl der privaten Rechner und Rechnerarbeitsplätze sehr stark angestiegen. Gleichzeitig wuchs deren Leistungsfähigkeit enorm. Viele dieser Rechner verfügen über ein größeres Leistungsvermögen als frühere Großrechner. Dies ermöglicht mit der fortschreitenden Bandbreitenvergrößerung verbesserte Kommunikationsmöglichkeiten, welche sehr viele Nutzer in Anspruch nehmen können. Derzeitige Client- Server- Lösungen bieten hierfür eine unzureichende Unterstützung. Die Kommunikation der Clients mit dem Server wird bei sehr vielen Nutzern zum Nadelöhr. Auch ist das Betreiben und Unterhaltung der Server sehr kostspielig. Zur Lösung dieses Problems wurde die Peer- to- Peer Technik entwickelt.
Die Herausforderung derzeitiger Peer- to- Peer Systeme bestehen in der Verbesserung der Zuverlässigkeit, Interoperabilität und Skalierbarkeit der Systeme [GB03]. Auf Grund dessen wurden verschiedene Architekturen entwickelt, die versuchen diesen Herausforderungen gerecht zu werden. Die zentrale Fragestellung bei der Nutzung dieser Architekturen betrifft die Minimierung des Aufwands der Ressourcenverwaltung und der Ressourcensuche. Werden die Ressourcen zentral verwaltet, so ist deren Suche am effizientesten. Allerdings erfordert dies einen hohen Aufwand für den Aufbau und Pflege einer solchen zentralen Ressourcenverwaltung. Dem gegenüber ist die dezentrale Ressourcenverwaltung mit einem minimalen Aufwand in Hinsicht auf die Ressourcenverwaltung verbunden, führt aber zu einer ineffizienten Ressourcensuche.
Diese Arbeit stellt verschiedene Lokalisierungsverfahren vor und diskutiert deren Vor- und Nachteile. Ein besonderer Schwerpunkt liegt dabei auf den Methoden der Ressourcensuche in dezentralen Systemen. Hier wird eine Auswahl von Methoden, die eine Tiefen- bzw. eine Breitensuche durchführen, sowie Methoden, die auf das Hashen aufbauen, präsentiert und diskutiert. Bei der Vorstellung der Lokalisierungsmethoden wird kurz auf den Aufbau der Systeme eingegangen. Wie die einzelnen Teilnehmer die Systeme betreten oder verlassen, kann in der angegebenen Fachliteratur nachgelesen werden.
Kapitel 2 stellt ausgewählte Grundlagen von Peer- to- Peer- Systemen, deren Namensgebung sowie eine Einteilung der Lokalisierungsverfahren vor. Kapitel 3 illustriert verschiedene Suchmethoden in dezentralen Peer- to- Peer Systemen, die auf dem Fluten aufbauen, Routing Indizes verwenden oder verteilt Hashtabellen benutzen. Das Kapitel 4 gibt einen Ausblick auf P2P Datenbanken.

2. Grundlagen

2.1 Peer- to- Peer Systeme und deren Klassifikation

Ein Peer- to- Peer System (P2PS) ist eine Netzwerkarchitektur, die aus vielen verschiedenen Knoten besteht, wobei die Knoten untereinander verbunden sind. Jedem Knoten sind dabei unterschiedliche Ressourcen (in Typ und Menge) zugeordnet. Als Ressourcen können sowohl Rechenleistung, Speicherkapazität, als auch Daten in unterschiedlicher Form verstanden werden. Ziel ist es, den Mangel einer Ressource in einem Knoten durch deren Bereitstellung auf einen anderen Knoten zu beheben. Dabei dienen die Verbindungen zwischen den Knoten einerseits zur Lokalisierung von Ressourcen mittels Suchnachrichten und andererseits zum Zugriff auf die gefundenen Ressourcen. In Abbildung 2 – 1 ist ein Beispiel für ein P2PS gegeben. Die Kanten im Graph stellen die Verbindungen zwischen den Knoten dar. Darüber werden die Ressourcen oder die Suchnachrichten ausgetauscht.

Nach [SCH02] werden Netzwerkarchitekturen als Peer- to- Peer Systeme bezeichnet, "... wenn die Teilnehmer einen Teil ihrer eigenen Ressourcen ... zur Verfügung stellen. Die Teilung der Ressourcen ist notwendig, damit das Netzwerk Dienste oder Inhalte zur Verfügung stellen kann (Bsp.: gemeinsame Dateinutzung, gemeinsame Arbeitsdokumente). Auf die Ressourcen kann von jedem Peer direkt zugegriffen werden, ohne auf die zwischenliegenden Peers zuzugreifen. Die Teilnehmer eines solchen Netzwerkes sind daher zugleich Ressourcenanbieter und Ressourcennutzer."

Zusammenfassend ergeben sich folgende grundlegende Eigenschaften [SCH02]:

- Existenz von direkten Verbindungen zwischen den Peers
- Peers als Anbieter und Nutzer von Ressourcen
- teilweise Bereitstellung von Diensten durch zentrale Server

Zur Klassifizierung von Peer- to- Peer Systemen existieren verschiedene Ansätze. Klassifizierungsmerkmale betreffen beispielsweise die zugrunde liegende Architektur, die Struktur der Ressourcenverteilung und die Struktur von datengebundenen Ressourcen.

Hinsichtlich ihrer Architektur werden P2PS nach [SCH02] in dezentrale (pure, reine) und zentrale (hybride) Peer- to- Peer Systeme unterteilt. In hybriden Peer- to- Peer Systemen wird ein Suchindex durch ein oder mehrere zentrale Server bereitgestellt. Alle anderen Aufgaben werden von den

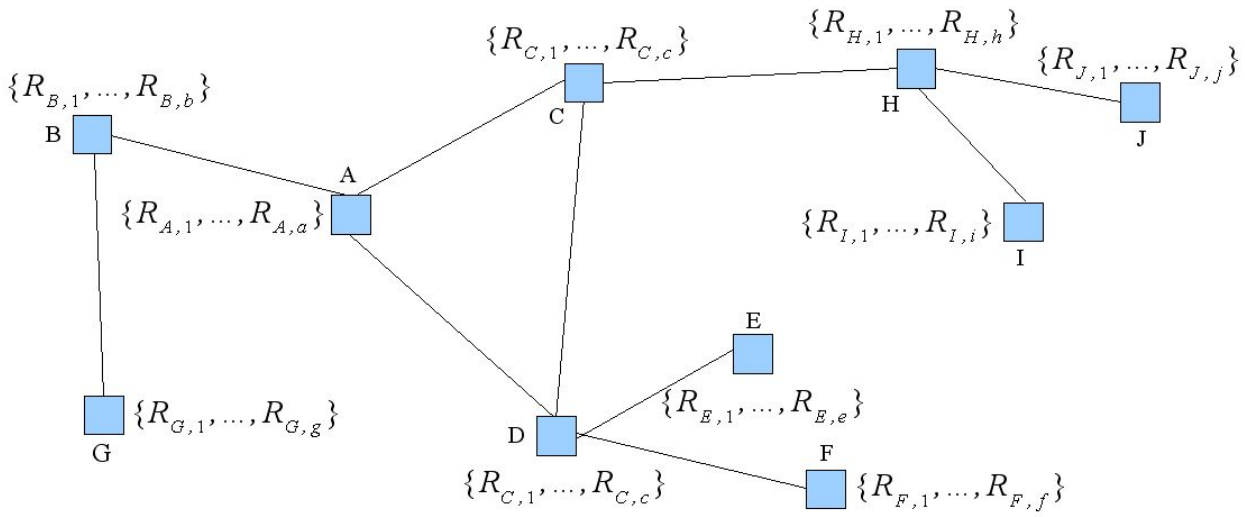


Abbildung 2 – 1

Knoten (oder Peers) erledigt. Der Vorteil einer solchen Architektur liegt in der Effizienz der Suche. Die Knoten greifen auf einen zentral verwalteten Suchindex zu, welcher gut durch eine Datenbank realisiert werden kann. In puren Peer- to- Peer Systemen wird komplett auf eine oder mehrere zentrale Einheiten verzichtet. Es ist vollkommen dezentralisiert. Dadurch bedingt, ist es sehr schwierig Ressourcen effizient zu finden.

Strukturen von datengebundenen Ressourcen werden nach strukturiert, semistrukturiert und unstrukturiert klassifiziert. Dabei entsprechen strukturierte Ressourcen einem oder mehreren komplexen Schemen. Der Inhalt einer Ressource ist feingranular und getypt. Im Gegensatz dazu stehen die unstrukturierten Ressourcen. Hier ist der Inhalt grobgranular und unterliegt keinem expliziten Schema. Die kleinste sinnvoll zu verarbeitende Einheit ist meist die Ressource selbst. Vermischen sich Merkmale von strukturierten und unstrukturierten Ressourcen, spricht man von semistrukturierten Ressourcen

Strukturen bei der Verteilung von Ressourcen teilt man ebenfalls in strukturiert, semistrukturiert und unstrukturiert ein. Die Platzierung von Ressourcen in der strukturierten Ressourcenverteilung folgt einem festgelegten Plan. Gegensätzlich dazu verhält sich die unstrukturierte Ressourcenverteilung. Hier ist die Platzierung von Ressourcen im System nicht vorgegeben. Sind beide Merkmale der Ressourcenplatzierung im System vorhanden, spricht man von semistrukturierter Ressourcenverteilung. In Abbildung 2 – 2 sind die Zusammenhänge dargestellt.

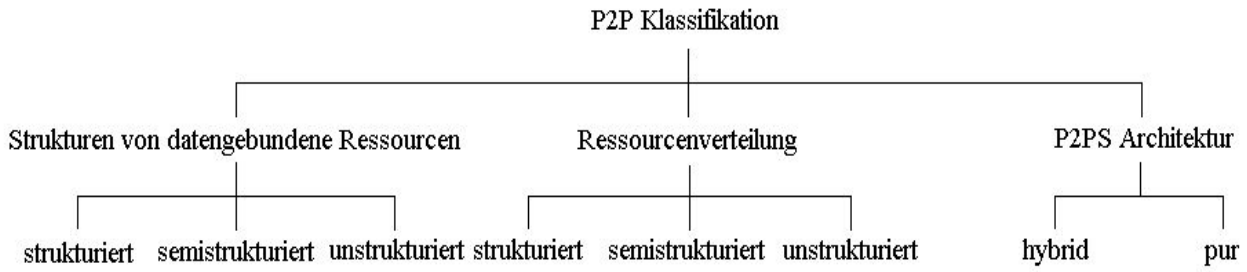


Abbildung 2 – 2

2.2 Namensgebung in Peer- to- Peer Systemen

Zur direkten (physischen) Lokalisierung von Knoten oder Ressourcen dienen Adressen, welche denen eindeutig zugeordnet sind. Wegen häufigen Adressänderungen erhält jeder Knoten oder jede Ressource zusätzlich einen sogenannten Bezeichner. Über diesen Bezeichner kann die Adresse aufgelöst werden. Dabei haben die Bezeichner folgende Eigenschaften[WiJ95]:

- Ein Bezeichner gehört höchstens zu einen Knoten oder einer Ressource
- Jedem Knoten oder jeder Ressource ist genau ein Bezeichner zugeordnet
- Ein Bezeichner gehört immer zu dem selben Knoten oder zu der selben Ressource.

Unter dem Begriff des Namens werden die Adressen und die Bezeichner von den Knoten oder Ressourcen zusammengefasst. Dabei können mehrere Namen, sogenannte Alias- Namen, zu einem Bezeichner oder einer Adresse gehören.

In getrennten Kontexten können verschiedene Knoten oder Ressourcen gleiche Namen haben. Der entstehende Konflikt mit der Eindeutigkeit wird durch die Einführung von Namensräumen aufgelöst. Der Zugriff auf einen Knoten oder eine Ressource wird durch einen zusammengesetzten Verweis auf den Namen der Ressource und den Namensraum geregelt und als Referenz bezeichnet.

In Abbildung 2 – 1 ist ein Beispiel für die Namensgebung aufgeführt. Die einzelnen Knoten sind eindeutig durch ihren physikalischen Ort bestimmt. Damit bei einem Bezug auf die Knoten nicht jedes Mal der Ort angegeben werden muss, erhalten die einzelnen Knoten die Bezeichner ”A“- ”J“. Der Einfachheit halber seien die Namen der Knoten mit den Bezeichnern identisch. Die in den Knoten gespeicherten Ressourcen bekommen ebenfalls einen Bezeichner. In der Abbildung sind sie mit $R_{j,i}$ abgekürzt. Innerhalb eines Knotens sind die Namen der Ressourcen eindeutig.

Es kann vorkommen, dass zwei Ressourcen in verschiedenen Knoten gleiche Namen haben. Der entstehende Konflikt mit der Eindeutigkeit der Namen wird durch Angabe des Knotennamens aufgelöst. Der Knoten, der die Ressource beinhaltet, stellt somit einen Namensraum dar.

2.3. Ressourcenlokalisierung im Überblick

Da ein Knoten i. a. nicht alle verfügbaren Ressourcen besitzt, müssen nicht vorhandene Ressourcen bei einen anderen Knoten im Peer- to Peer System lokalisiert und genutzt werden. Dabei wird der Name der Ressource von einem Lokalisierungsdienst auf eine Referenz abgebildet. Mit dieser Referenz kann nun der Knoten die gewünschte Ressource nutzen.
Für die Lokalisierung gibt es verschiedene Strategien in Peer- to- Peer Systemen:

- Durchsuchen des gesamten Systems (angewendet in pure P2PS)
- Ablage und Nachschlag in einem zentralen Register (angewendet in hybride P2PS)

Diese Suchstrategien können in Suchverfahren unterteilt werden. Bei puren Peer- to- Peer Systemen sind Verfahren mit und ohne Routing Indizes sowie Verfahren, die auf verteilten Hashtabellen aufbauen, vorhanden.
Zu dem Suchverfahren ohne Routing Indizes werden die Methoden Non- Indexed- Flooding, Iterative Deeping, Directed BFS und Local Indices im nächsten Kapitel vorgestellt. Außerdem wird auf die Methoden Compound Routing Indizes und Hop Count Routing Indizes als Vertreter der Suchverfahren mit Routing Indizes eingegangen. Als Abschluss der Lokalisierung in puren Peer- to-Peer Systemen werden die auf verteilten Hashtabellen aufbauenden Methoden CAN und CHORD besprochen.

3. Ausgewählte Methoden der Lokalisierung

3.1 Die naive Suchmethode: Fluten

In dezentralen P2P Systemen besitzt jedes teilnehmende Peer eine gewisse Anzahl von Nachbarn. Mit diesen werden Suchnachrichten ausgetauscht. Der Ressourcenaustausch selbst findet direkt zwischen den betreffenden Peers statt.
Die naivste Möglichkeit der Suche nach Ressourcen wird als das ”Fluten“ (oder Schneeballprinzip)

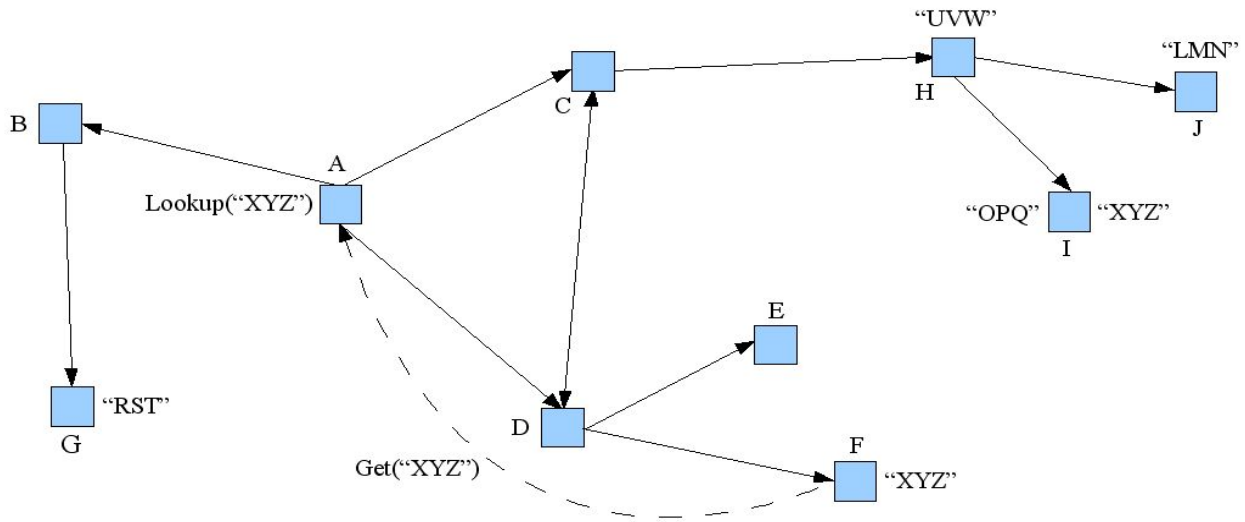


Abbildung 3 – 1

bezeichnet. Dabei prüft jeder Knoten eingehende Anfragen und leitet sie an seine Nachbarn weiter. Ist zusätzlich die gesuchte Ressource auf dem Knoten vorhanden, so wird eine Erfolgsnachricht entsprechend des Weges der Suchnachricht zum Initiator der Anfrage zurückgesendet. Die Suchnachricht wird an die Nachbarn weitergeleitet, auch wenn die Ressource vorhanden ist. Damit wird das ganze P2P System abgesucht und es kann sicher entschieden werden, ob die gesuchte Ressource grundsätzlich vorhanden ist und deren Anzahl festgestellt werden.

Die dezentrale Struktur des Systems hat dabei den Vorteil, dass sich das Netz selbst verwaltet. Das heißt, die Verwaltung der Knoten sowie die Aufnahme neuer Knoten ist mit geringem Aufwand möglich. Nachteilig wirkt sich das auf die Performanz der Suche aus. Durch die Vielzahl von Suchanfragen können Knoten unter der Kommunikationslast zusammenbrechen und so möglicherweise das P2P System in getrennte Subnetzwerke partitionieren. Ebenso nachteilig wirkt sich die Möglichkeit aus, dass Knoten mehrfach besucht werden, wodurch nicht endende Zyklen entstehen. Es kann auch nicht auf Netzwerkanomalien, wie zum Beispiel gezielte Angriffe auf Knoten, reagiert werden. Diese Suchmethode ist für eine effiziente Ressourcenallokation nicht praktikabel.

Abbildung 3 – 1 konkretisiert das in Abbildung 2 – 1 vorgestellte Beispiel für das Fluten. Es sind die Ressourcen mit den Bezeichnern ”LMN“, ”OPQ“, ”RST“, ”UVW“ und ”XYZ“ vorhanden, wobei ”XYZ“ in zwei verschiedenen Knoten präsent ist. Die Namen der Ressourcen seien mit deren Bezeichnern identisch.

Ziel ist es, ausgehend von Knoten ”A“ die Ressource mit dem Namen ”XYZ“ zu finden. Da die Ressource lokal nicht vorhanden ist, sendet er Suchanfragen an seine Nachbarn ”B“, ”C“ und ”D“. Diese suchen wiederum lokal nach der Ressource und leiten die Suchanfrage an ihre Nachbarn weiter. Dies wird so lange fortgeführt, bis die Knoten ”E“, “F“, ”G“, ”I“ und ”J“ erreicht sind und die Suche abbricht.

In den Knoten ”F“ und ”I“ wird die gesuchte Ressource gefunden und die Referenz an den anfragenden Knoten ”A“ entsprechend des Hinweges der Suchanfrage gesendet.

Eine der oben genannten Schwierigkeiten zeigt auch dieses Beispiel. Es entsteht ein Zyklus zwischen den Knoten ”A“, ”C“ und ”D“. Wird dieser nicht erkannt, so findet die Suchanfrage kein Ende.

Zur Beseitigung der Schwächen des Flutens wurden weitere Verfahren entwickelt, die Gegenstand der folgenden Abschnitte sind. Diese Verfahren lassen sich in die Lokalisierung ohne Routing Indizes, in die Lokalisierung mit Routing Indizes und in die Lokalisierung mit verteilten Hashtabellen unterteilen. Bei dem Verfahren ohne Routing Indizes werden keine Informationen über die vorhandenen Ressourcen zur Lokalisierung benutzt. Dadurch sind die auf diesem Verfahren aufbauenden Methoden nur eingeschränkt nutzbar. Diesen Mangel versucht man durch das Führen von Routing Indizes zu beheben. Man benutzt Strukturen in Ressourcen sowie Strukturen in der Verteilung der Ressourcen zur effizienteren Suche. Danach wird auf das Verfahren der verteilten Hashtabellen und die Lokalisierung in hybriden Peer-to- Peer Systemen eingegangen.

3.2 Lokalisierung ohne Routing Indizes

3.2.1 Non- Indexed- Flooding

Beim Non- Indexed- Flooding wird das Peer- to- Peer System durch Fluten bis zu einer vorgegebenen Tiefe durchsucht. Die Suchanfragen werden von einem die Anfrage bearbeitenden Knoten zu seinen Nachbarn gesendet. Diese wiederum senden die Anfrage an ihre Nachbarn. Die Ergebnisse der Suchanfrage werden über den Herkunftsweg zum anfragenden Client zurückgesendet. Damit nicht das gesamte Netzwerk geflutet wird, erfolgt die Suchanfragen mit einem Time- to- Live Flag

(TTL Flag). Das TTL Flag dient der Terminierung der Suchanfrage nach einer festgelegten Anzahl

von Sprüngen. Jeder Knoten dekrementiert das TTL Flag um den Wert ”1“ und leitet die Suchanfrage so lange weiter, bis das TTL Flag den Wert ”0“ erreicht. Dies hat zur Folge, dass nur ein Teil des gesamten Netzwerkes durchsucht wird, was zu einer Reduzierung der Anzahl der Nachrichten führt. Dauerhafte Zyklen können durch die beschränkte Anzahl von Weiterleitungen der Suchanfrage nicht entstehen. Trotzdem ist die Anzahl der bei einer Suche erzeugten Nachrichten sehr hoch. Darunter leidet wie in der naiven Methode die Performanz der Lokalisierung. Auch können auf Grund der hohen Kommunikationslast Knoten ausfallen und das System partitionieren.
Ein weiterer Schwachpunkt ist, dass bei einer negativen Rückmeldung nicht entschieden werden kann, ob die Ressource im System existiert, oder nur die begrenzte Suchtiefe erreicht wurde.
In dem in Abbildung 3 – 1 gegebenen Beispiel kann bei einer Suchtiefe von ”1“ keine Ressource mit dem Namen “XYZ” gefunden werden bzw. bei einer Suchtiefe von ”2“ wird nur die Ressource im Knoten ”F“ gefunden. Nach dem Auffinden der Ressource wird die Referenz entsprechend des Suchweges zum Initiator der Suchanfrage zurückgeleitet. Daraufhin stellt Knoten ”A“ eine direkte Verbindung zu Knoten ”F“ her, auf dem die gesuchte Ressource ”XYZ“ verfügbar ist.
Ein Anwendungsbeispiel für diese Suchmethode ist das Gnutella- Netzwerk. Hier hat jedes Peer vier Nachbarn. Die Suchtiefe beträgt ”7“. Ist eine Ressource gefunden, so wird deren Ort als URL an den Urheber der Suchanfrage gesendet. Dieser lädt mittels des HTTP-Protokolls die Ressource.
Wie oben dargestellt, ist diese Suchmethode nicht effektiv. Deshalb werden in [YM02] drei Methoden zur Effizienzsteigerung vorgeschlagen, die im Folgenden vorgestellt werden:

(1) Iterative Deeping
(2) Directed BFS
(3) Local Indices.

3.2.2.Iterative Deeping

Beim Iterative Deeping werden statt einer festen Suchtiefe verschiedene Tiefen (r, t, s, ...) benutzt, so dass sich die Anzahl der erreichbaren Knoten sukzessive erhöht. Die Suche erfolgt analog zum Non- Indexed- Flooding. Der Initiator sendet eine Suchanfrage mit der Suchtiefe r. Werden die Anfragen von Knoten auf der r- ten Suchebene (Suchhorizont) empfangen, so werden sie als *frozen Q*uery temporär gespeichert. Erhält der Initiator nicht genügend Resultate, so sendet dieser eine *resend message* mit neuer Tiefe t. Die schon beim ersten Suchlauf besuchten Knoten werden nicht nochmals durchsucht, sondern sie leiten die Anfrage direkt an ihre Nachbarn weiter. Werden diese

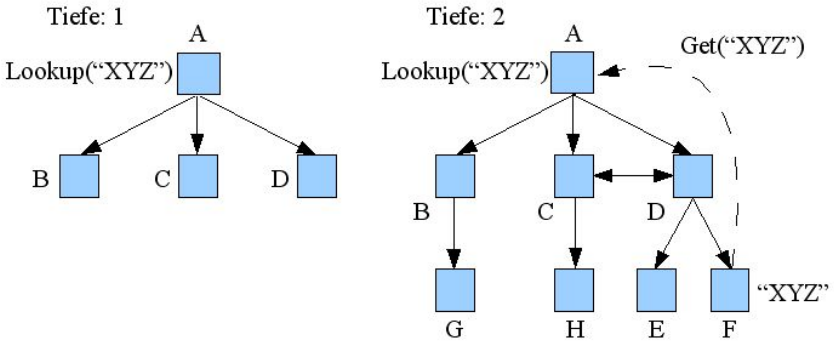


Abbildung 3 – 2

Nachrichten von einem Knoten auf der r-ten Suchebene empfangen, so wird die *frozen* Query als normale Suchanfrage mit einer neuen Suchtiefe von t - r weitergeleitet. Dieser Vorgang kann bis zu einem Maximum beliebig oft wiederholt werden. Der Vorteil dieser Methode liegt in der geringeren Netzwerk- und Knotenbelastung. Werden bei einer geringen Suchtiefe genügend Resultate erzielt, so wird die weitere Suche in der Tiefe eingespart. Nachteilig wirkt sich die erhöhte Suchdauer bei wenig vorhandenen Ressourcen aus, da auf die Resultate der Suche gewartet werden muss, bevor eine neue Suche mit einer größeren Suchtiefe initiiert werden kann. Abbildung 3 – 2 setzt das aus dem vorherigen Abschnitt bekannte Beispiel fort. Knoten ”A“ sucht wieder die Ressource mit dem Namen ”XYZ“. Er benötigt nur ein Resultat zur Befriedigung der Anfrage. Die verschiedenen Tiefen seien ”1“, ”2“ und ”3“. Da die Ressource nicht lokal vorhanden ist, sendet Knoten ”A“ die Suchanfrage mit der Tiefe ”1“ an seine Nachbarn ”B“, ”C“ und ”D“. Da die maximale Suchtiefe erreicht ist, ohne die gesuchte Ressource gefunden zu haben, wird diese eingefroren und an Knoten ”A“ ein negatives Resultat gesendet. Dieser sendet eine neue Anfrage mit der Tiefe ”2“. Dadurch wird der Knoten ”F“ erreicht und ”XYZ“ gefunden. Das Suchen in den Knoten mit der Tiefe ”3“ kann entfallen, da nur eine Ressource gefordert war.

3.2.3 Directed Breadth First Search (Directed BFS)

Diese Methode führt wie das Fluten eine Breitensuche durch. Zur Vermeidung der hohen Belastung des Netzwerkes wird hier eine Suchanfrage nur an ausgewählte Nachbarn versendet. Um einen Qualitätsverlust zu vermeiden, wird über jeden Nachbarn eine Statistik geführt und anhand dieser die besten Nachbarn ausgewählt (Ranking). Mögliche Kriterien zum Ordnen der Nachbarn sind:

(1) die Anzahl der erfolgreich beantworteten Suchanfragen

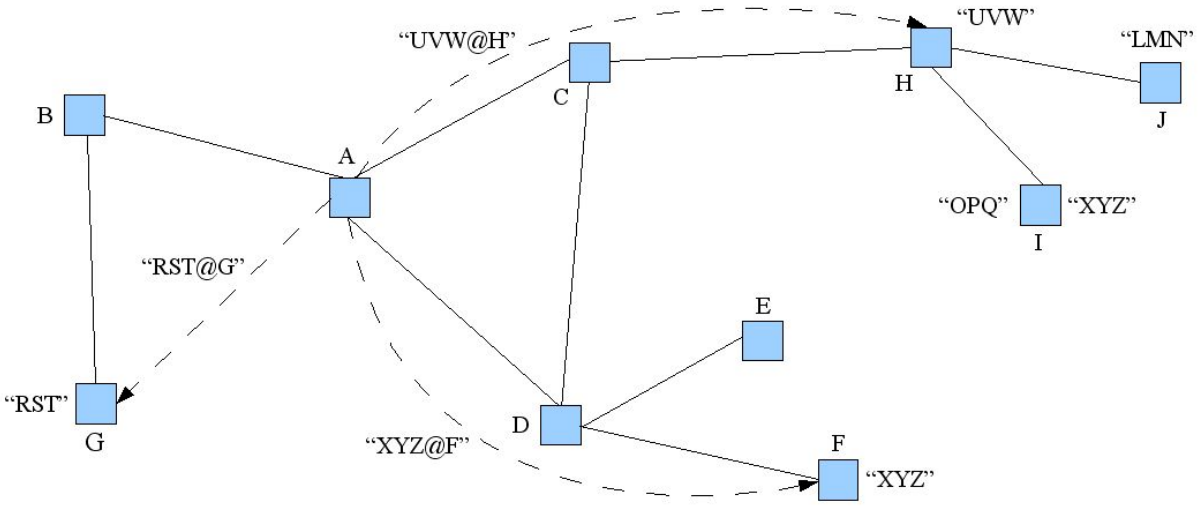


Abbildung 3 – 3

(2) die Anzahl der Knoten, über welche die Suchanfrage geroutet wird (Hop Count)

(3)die Anzahl der Nachrichten, die der Nachbar verschickt hat. Eine hohe Anzahl verweist auf eine gute Anbindung.

(4)die Größe der Warteschlange, wobei eine lange Warteschlange auf eine Auslastung des Zielknotens oder sogar dessen Ausfall hindeutet.

3.2.4 Local Indices

Eine andere Verbesserung ergibt sich, wenn lokale Indizes verwendet werden. Beim Non- Indexed-Flooding verfügt jeder Knoten über einen Index, der die Ressourcen aller innerhalb eines Radius (oder Tiefe) befindlichen Knoten enthält. Erhält ein Knoten eine Suchanfrage, so durchsucht er diesen Index. Wird die gewünschte Ressource nicht gefunden, wird die Anfrage an die Nachbarn gesendet. Diese leiten die Suchanfragen an ihre Nachbarn weiter, ohne den lokalen Index zu benutzen. Dieses Prozedere geschieht so lange, bis ein Knoten erreicht wird, der noch nicht für eine Suche herangezogen wurde. Diese durchforsten ihren lokalen Index nach der gesuchten Ressource und leiten die Anfrage bei Nichtvorhandensein der Ressource an ihre Nachbarn weiter. Dieser vorgegebene Prozess wiederholt sich so lange, bis die Ressource gefunden wird oder das TTL- Flag den Wert ”0“ erreicht hat. Bei dieser Methode der Lokalisierung muss sichergestellt sein, dass die Indizes über die Nachbarn beim Betreten oder Verlassen von Knoten aktualisiert werden. Dazu existieren die Aktionen *join*, *leave*, *update*, auf die hier nicht weiter eingehen wird. In der Praxis hat

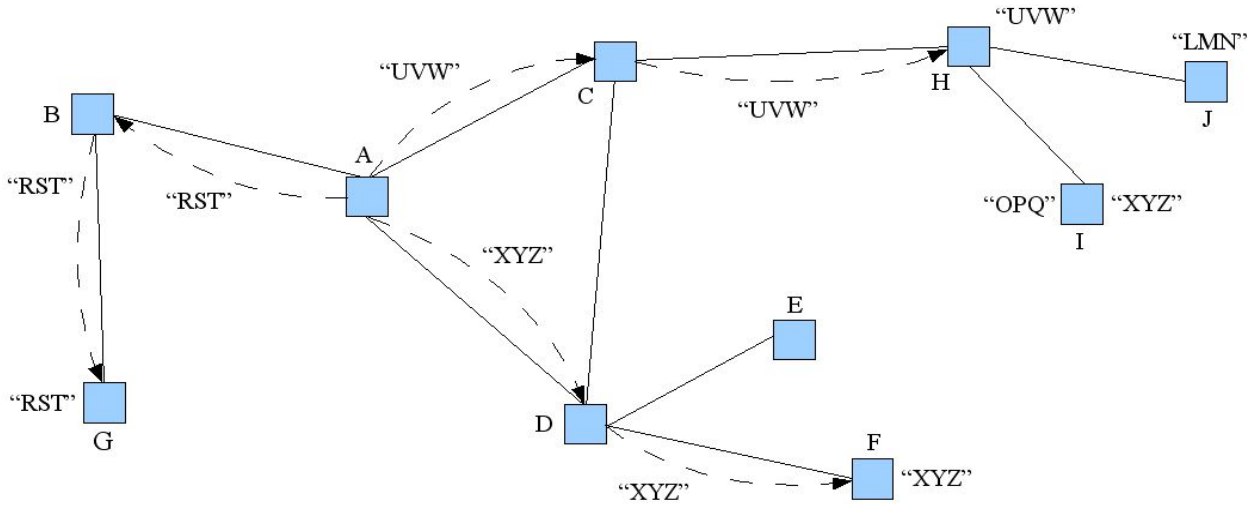


Abbildung 3 – 4

sich herausgestellt, dass das Führen von lokalen Indizes zu einen erhöhten Speicher- und Änderungsaufwand führt, aber die Netzwerkbelastung und die eingesetzte Rechenleistung stark reduziert. In Abbildung 3 – 3 ist ein Beispiel für einen lokalen Index in Knoten ”A“ gegeben. Der Radius hat den Wert ”2“. Knoten ”A“ speichert in diesem Index Verweise auf die Ressourcen ”UVW“ im Knoten ”H“, ”XYZ“ im Knoten ”F“ und ”RST“ im Knoten ”G“. Sobald der Knoten ”A“ eine der genannten Ressourcen sucht, findet er sie in seinen Index und braucht nicht die Nachbarn kontaktieren. Dagegen ist die Ressource ”OPQ“ nicht im Index von Knoten ”A“ vorhanden. Deshalb sendet er die Anfrage an seine Nachbarn ”B“, ”C“ und ”D“, welche diese ohne sie zu bearbeiten an ihre Nachbarn weiterleiten. Dort wird ebenso verfahren. Damit ist die Suchtiefe ”3“ erreicht und alle Ressourcen sind nicht mehr Inhalt des Index von ”A“. Die Knoten auf dieser Ebene suchen in ihren lokalen Indizes nach der Ressource ”OPQ“, wobei ”I“ sie findet.

3.3 Lokalisierung mit Routing Indizes

3.3.1 Überblick

Im Gegensatz zu lokalen Indizes enthalten Routing Indizes keine direkten Referenzen auf Ressourcen. Vielmehr werden mit ihnen anhand verschiedener Methoden Vorschläge für die zu besuchenden Knoten generiert, auf denen die Suche weitergeführt werden kann.

Wird eine Suchanfrage an einen Knoten gestellt, so bearbeitet dieser die Anfrage, sucht mittels seiner Routingtabelle einen geeigneten Nachbarknoten und leitet die Anfrage an diesen weiter. Der Prozess wird so lange fortgesetzt, bis die gewünschte Ressource gefunden ist.
Die Vorteile gegenüber den lokalen Indizes liegen in dem verringerten Speicherplatz der Indizes sowie in der Reduzierung der Netzwerkbelastung, da bei Veränderungen weniger Änderungsinformationen versendet werden müssen.
In [CM02] werden verschiedene Methoden zur Bestimmung der zu besuchenden Knoten gezeigt, von denen hier

(1)Compound Routing Indizes

(2)Hop- Count Routing Indizes

näher vorgestellt werden.

3.3.2 Compound Routing Indizes

Das Compound Routing (CRI) verwendet Routing Indizes in denen Ressourcen in Kategorien zusammengefasst werden. Jeder Knoten besitzt neben einem Index, der die lokalen Ressourcen verwaltet, eine Routingtabelle, welche die Anzahl der Ressourcen, die über Nachbarknoten erreichbar sind, aufgeteilt nach den Kategorien der Ressourcen, enthält. Wird eine nicht lokal vorhandene Ressource gesucht, so wird diese den Kategorien zugeordnet. Die Suchanfrage wird an den Nachbarn geleitet, der über die größte Güte der zugeordneten Kategorien verfügt. Die Güte G eines Nachbarn berechnet sich mit der Formel $\#Res * \prod_s \frac{CRI(s)}{\#Res}$, wobei $CRI(s)$ die Anzahl der Ressourcen in der Kategorie s ist. Hierbei werden nur die Kategorien verwendet, zu denen die Ressource zugeordnet wurde. Ist die gesuchte Ressource nicht im Nachbarknoten vorhanden, verfährt dieser analog.
Vorteilhaft bei dieser Suchmethode ist neben dem reduzierten Platzbedarf der Routingtabellen, dass viel weniger Suchanfragen in eine Richtung geleitet werden, in welcher überhaupt keine passende Resultate existieren.
Durch die fehlende Gewichtung der Entfernung der Ressourcen wird eine Änderung in der Anzahl der Ressourcen in einer Kategorie zu jedem Knoten propagiert. Das führt zu einen enormen Änderungsaufwand, wobei zu berücksichtigen ist, dass eventuell vorhandene Zyklen aufgelöst werden müssen. Schwierigkeiten bereitet die Kategorisierung der Ressourcen. Oft ist es nicht

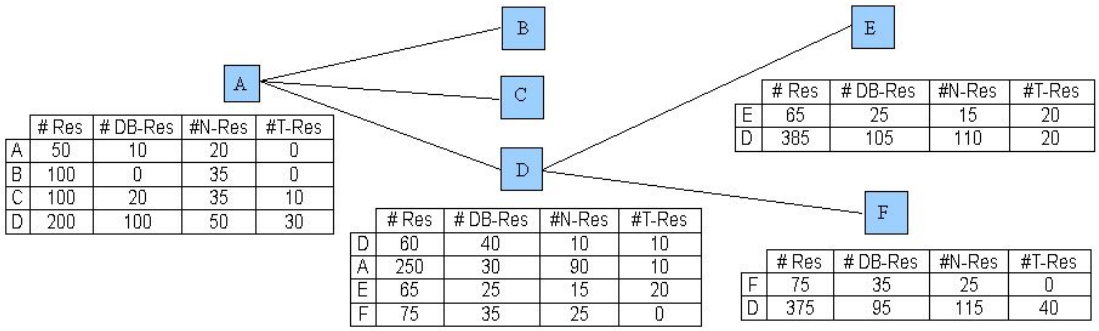


Abbildung 3 – 5

eindeutig, in welche Kategorie eine Ressource eingeteilt werden soll. Auch kann der Eindruck entstehen, dass beim Vorhandensein von vielen Ressourcen in einer Kategorie viele spezielle Ressourcen vorhanden sind.

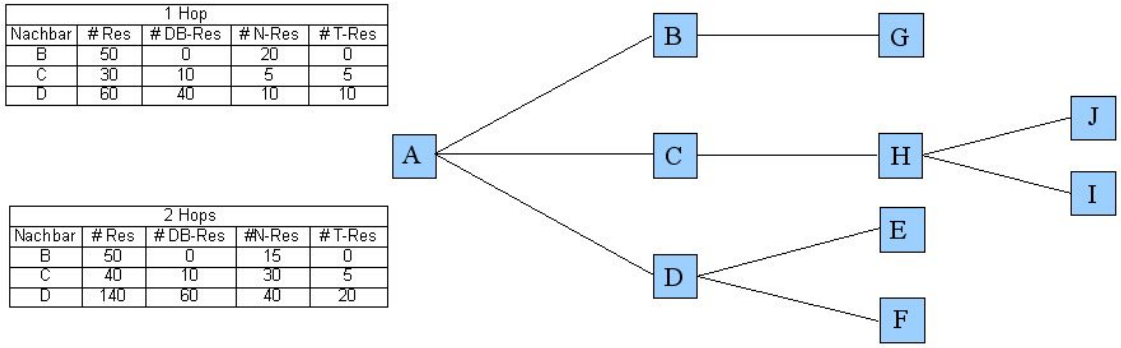
Die Abbildung 3 - 5 erweitert das gegebene Beispiel und illustriert diese Methode. Der Knoten ”A“ stellt eine Suchanfrage nach der Ressource ”XYZ“. Da die Ressource nicht im lokalen Index verzeichnet ist, wird sie den vorhandenen Kategorien im Routing Index zugeordnet. Beispielsweise seien die Kategorien ”Datenbanken“ (DB), ”Netzwerke“ (N) und ”Theorie“ (T) vorhanden, wobei ”XYZ“ den Kategorien ”DB“ und ”T“ zugeordnet wird. Die Güte G der Nachbarknoten berechnet sich durch

$$G(B)=200*\frac{0}{200}*\frac{0}{200}=0\ ,\ G(C)=200*\frac{20}{200}*\frac{10}{200}=1 \text{ und } G(D)=200*\frac{100}{200}*\frac{30}{200}=15.\ .$$

Knoten ”A“ wählt ”D“ zum Nachfolger aus, da dessen Güte maximal ist. ”D“ hält ebenfalls nicht die Ressource lokal und berechnet deshalb die Güte seiner Nachbarn ”E“ und “F“ (G(E) = 7, G(F)= 0). Damit wird die Anfrage an den Knoten ”E“ geleitet. Dieser besitzt nicht die Ressource und hat auch keine weiteren Nachbarn. Deshalb sendet er die Suchanfrage an Knoten ”D“ zurück, der anhand der Güte den nächstbesten Knoten auswählt. Damit wird die Suchnachricht an Knoten ”F“ gesendet, auf dem die Ressource gefunden wird und die Suche terminiert.

3.3.3 Hop-Count Routing Indizes

Eine verbesserte Version der Suche mit Compound Routing Indizes ist das Suchen mit Hop- Count Routing Indizes. Hier werden die Kategorien zusätzlich mit einem Maß für die Entfernung der Ressourcen gewichtet. Zur Berechnung der Güte der einzelnen Knoten wird auf [CM02]



1 Hop

Nachbar	# Res	# DB-Res	# N-Res	# T-Res
B	50	0	20	0
C	30	10	5	5
D	60	40	10	10

2 Hops

Nachbar	# Res	# DB-Res	#N-Res	# T-Res
B	50	0	15	0
C	40	10	30	5
D	140	60	40	20

Abbildung 3 – 6

verwiesen.In jedem Knoten sind Routing Indizes für jeden ”Hop“ (Sprung) bis zu einer gewissen Tiefe gespeichert. Die Tiefe wird Horizont genannt. Informationen über den Horizont hinaus sind nicht bekannt. Die Suche nach Ressourcen erfolgt analog wie das Suchen mit Compound Routing Indizes. Die Lokalisierung mit Hop-Count Routing Indizes vergrößert die Routingtabelle zum Speichern der Routing Indizes, löst aber das Problem, dass Suchanfragen an Nachbarn geschickt werden, die Zugriff auf eine große Anzahl von Ressourcen haben, diese aber sehr weit entfernt sind. Außerdem wird durch den beschränkten Horizont der Änderungsaufwand sehr stark reduziert.
Abbildung 3 – 6 zeigt einen solchen Hop-Count Routing Index im Knoten ”A“ mit der Tiefe ”2“ (siehe Tabelle ”1 Hop“). Mit einem Sprung zu Nachbar ”D“ können 60 Ressourcen erreicht werden, wovon 40 aus dem Bereich ”Datenbanken“, 10 aus dem Bereich ”Netzwerk“ und 10 aus dem Bereich ”Theorie“ stammen. Der Knoten ”A“ kann über ”D“ weitere 140 Ressourcen mit ”2“ Sprüngen (die Nachbarn von ”D“) erreichen.
Die bisher vorgestellten Methoden haben einen linearen Suchaufwand (gemessen an der versendeten Suchnachrichtenanzahl) zur Anzahl der besuchten Knoten im Peer- to- Peer System.
Nachfolgend werden zwei Suchmethoden erklärt, bei denen sich der Suchaufwand auf $O(\log N)$

bzw. $O(\frac{D}{4} * N^{\frac{1}{D}})$ reduziert.

3.4 Lokalisierung mit verteilten Hashtabellen

Bei diesem Lokalisierungsverfahren werden Hashwerte von Ressourcen erzeugt, über die die Lokalisierung stattfindet. Unterschiede in den Hashfunktionen führen zu verschiedenen Methoden.

Bei allen Methoden werden jedem Knoten bestimmte Wertebereich der Hashfunktion zur Verwaltung der Ressourcen zugeordnet. Dabei kann die Ressource selbst oder eine Referenz auf die Ressource vom verwaltenden Knoten gehalten werden. Zur Lokalisierung einer Ressource wird der Hashwert verwaltende Knoten gesucht.
Die auf diesem Verfahren aufbauenden Methoden haben bei einer günstigen Wahl der Hashfunktion Vorteile bezüglich der Skalierbarkeit, Lastbalancierung, Verfügbarkeit und Namensgebung.

3.4.1 CHORD

Die in [SM02] beschriebene Methode CHORD verwendet einen logischen Ring, um eine Ressource nach $O(\log N)$ Schritten zu finden. Die einzelnen Knoten werden mittels einer gleichmäßig verteilenden Hashfunktion (beispielsweise SHA-1) jeweils auf einen n- Bit langen Hashwert abgebildet. Die Hashwerte werden als Punkte auf einen Ring $modulo\, 2^n$ aufgefasst. Ein Wert k auf dem Ring wird von dem Knoten mit dem kleinsten Hashwert verwaltet, der größer oder gleich k ist. Diesen Knoten bezeichnet man als successor(k). Die auf den Knoten vorhandenen Ressourcen werden wie die Knoten mit der Hashfunktion auf Hashwerte t im Ring verteilt. Die Schlüssel-Ressourcen Paare (bzw. Schlüssel- Referenz Paare) verwaltet der t nachfolgende Knoten (successor (t)).
Zur Realisierung der Suche speichert jeder Knoten den auf dem Ring nachfolgenden Knoten. Zusätzlich wird zur schnellen Navigation eine *finger table* geführt. Der i- te Eintrag in der Tabelle des Knotens mit dem Hashwert j speichert den Knoten mit dem Hashwert $j+2^i \bmod 2^n, i \in \{0,...,\text{n-1}\}$. Existiert dieser Knoten nicht, so wird der im Uhrzeigersinn nächste Knoten gespeichert.
Trifft eine Suchanfrage an einem Knoten d ein, so wird der Schlüssel der gesuchten Ressource mit der Hashfunktion auf einen Wert k abgebildet. Die Suche liefert nicht den Knoten, der die Ressource hält, sondern den Knoten der den Hashwert verwaltet. Dies hat den Vorteil, dass beim Einfügen von Knoten in das System der Suchalgorithmus zur Bestimmung des Nachfolgeknotens und der Einträge in den *finger tables* benutzt werden kann.
Bei der einfacheren Variante der Suche prüft der Knoten, ob er k verwaltet und sendet die Suchanfrage bei nicht Vorhandensein des Hashwertes an den Nachfolger von d.

```
//ask node n to find the successor of id
n.find_successor(id)
    if id in (n, n.successor]
        return n.successor;
    else
    //forward the query around the circle
        return successor.find_succesor(id);
```

Abbildung 3 – 7

```
//ask node n to find the successor of id
n.find_successor(id)
        if id in (n,n.successor]
                return n.successor
        else
                n'= clossest_preceding_node(id)
                return n'.find_successor(id)
```

```
//search the local table for the highest predecessor of id
n.clossest_preceding_node(id)
        for i = 0 to n-1
                if finger[i] in (n,id)
                        return finger[i]
        return n
```

Abbildung 3 – 8

Dies wird so lange fortgesetzt, bis alle Knoten besucht worden sind oder ein Knoten gefunden ist, der k verwaltet. In Abbildung 3 – 7 ist der Algorithmus als Pseudocode dargestellt.
In der erweiterten Variante zur Suche von Ressourcen wird zur Lokalisierung die *finger table* benutzt. Trifft eine Suchanfrage bei einem Knoten ein, so wird geprüft, ob der Knoten den Hashwert verwaltet. Ist dies nicht der Fall, so wird die Anfrage an den Knoten aus *finger table* geleitet, dessen Hashwert vor dem gesuchten Wert liegt. Der Pseudocode ist in Abbildung 3 – 8 abgebildet.
Die Verwendung einer gleichmäßig verteilenden Hashfunktion bewirkt eine gute Verteilung der Knoten und deren Ressourcen. Sind einzelne Knoten überlastet, so können leistungsstarke Knoten mehrere so genannte virtuelle Knoten simulieren. Dadurch wird die mittlere Anzahl der von einen Knoten verwalteten Ressourcen gesenkt und eine Lastbalancierung erreicht. Bei der Suche hat CHORD Vorteile gegenüber den bisher vorgestellten Suchmethoden. Die gesuchte Ressource wird nach $O(\log N)$ Schritten gefunden oder als definitiv nicht vorhanden erkannt. Probleme sind aber bei einer unscharfer Suche gegeben. Durch die Eigenschaft der gleichmäßig verteilenden Hashfunktion werden ähnliche Schlüssel auf sehr unterschiedliche Hashwerte abgebildet. So sind ähnliche Ressourcen nur schwer auffindbar. Problematisch für die Performanz sind sehr dynamische Peer- to- Peer Systeme, bei denen häufig sehr viele neue Knoten hinzukommen oder das System verlassen. Dabei kann es vorkommen, dass die Nachfolger bzw. die Einträge in den Fingertabellen nicht richtig gepflegt werden können und es so zu Fehlern bei der Suche kommt. Ein dem entgegenwirkender Stabilisierungsalgorithmus sorgt dafür, dass nach einer gewissen Zeit die korrekte Funktionsweise wieder hergestellt wird. Eine genaue Beschreibung zum Einfügen, Verlassen von Knoten und den Stabilisierungsalgorithmus kann in [SM02] gefunden werden.
In Abbildung 3 – 9 wird der Aufbau eines solchen Systems erklärt. Es wird ein Ring mit n = 6 und 10 Knoten. Beispielsweise hat der Knoten mit dem Namen ”B“ den Hashwert ”32“. Die Verwaltung der auf dem Ring abgebildeten 5 Ressourcen übernehmen die den Hashwert der Ressource

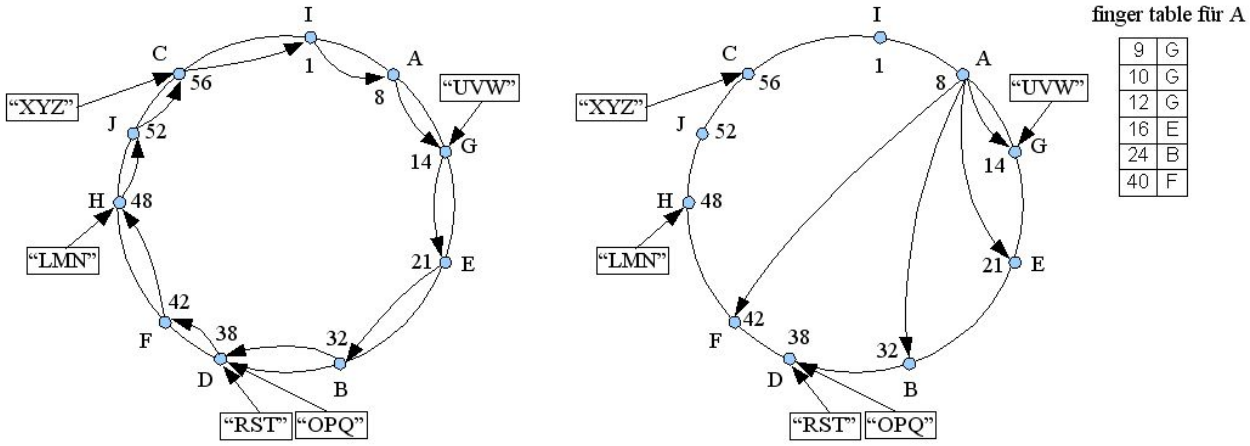


Abbildung 3 – 9

verwaltenden Knoten. Beispielsweise werden die Ressource ”RST” und ”OPQ“ mit den Hashwerten ”33“ und ”36“ vom Knoten ”D“ verwaltet. Stellt der Knoten ”A“ einen Suchanfrage nach der Ressource mit den Namen ”XYZ“, so wird der Hashwert von ”XYZ“ gebildet. Knoten ”A“ stellt fest, dass er den Hashwert nicht verwaltet. Bei der einfachen Suche leitet ”A“ eine Suchnachricht mit dem Hashwert zu seinen Nachbar ”G“. Dieser ist ebenfalls nicht für den Hashwert verantwortlich und leitet die Nachricht weiter. Die Nachricht wird solange weitergeleitet, bis der Knoten ”J“ erreicht ist. Dieser erkennt, dass nur ”C“ den Hashwert verwalten kann und liefert diesen Knoten als Ergebnis zurück. Nun kann durch Befragen des Knotens ”C“ durch Knoten ”A“ exakt festgestellt werden, ob er die Ressource verwaltet und gegebenenfalls die Ressource oder eine Referenz auf die Ressource liefern.

Bei der erweiterten Suche wird die *finger table* benutzt. Der Knoten ”A“ schaut in diese und wählt den Knoten ”F“ als den nächsten Knoten aus, dessen Hashwert der größte ist, der kleiner als der gesuchte ist. ”F“ verwaltet nicht den Hashwert und leitet die Suchnachricht mit dem Hashwert durch Nachschauen in seiner *finger table* an den Knoten ”J“ weiter. ”J“ stellt fest, dass ”C“ den gesuchten Hashwert verwaltet und sendet den Knoten ”C“ als Ergebnis der Suchanfrage zurück.

3.4.2 CAN

Das Content- Addressable Network (CAN) basiert wie CHORD auf verteilten Hashtabellen. Im Gegensatz dazu sind die Knoten und Ressourcen nicht auf einen Ring, sondern in einem virtuellen D-dimensionalen kartesischen Koordinatensystem organisiert. Jedem Knoten wird ein Teil des D-

dimensionalen Raumes zum Verwalten zugeteilt. Dabei wird die Position des Knotens im Raum durch eine gleichmäßig verteilende Hashfunktion bestimmt. Betritt ein neuer Knoten das System, so wird er mittels einer Hashfunktion auf einen Punkt des D- dimensionalen Raumes abgebildet. Der den Punkt verwaltende Knoten teilt seinen Raum und übergibt den Teilraum, der den Punkt enthält, an den neuen Knoten.

Die vorhandenen Ressourcen werden ebenfalls mit der Hashfunktion auf einen Punkt im D-dimensionalen Raum abgebildet. Die Schlüssel- Ressourcen Paare (bzw. Schlüssel- Referenz Paare) werden dem Knoten zugeordnet, der den zu dem Punkt gehörenden Teilraum verwaltet. Der exakte Vorgang zum Aufbau eines solchen Systems kann in [RF01] nachgelesen werden.

Weiter sind jedem Knoten nur seine Nachbarn (Adresse und Koordinaten im Raum) bekannt. Dabei sind zwei Knoten Nachbarn, wenn der Schnitt der von ihnen verwalteten Teilräume ein D - 1 dimensionales Objekt entlang vom D-1 Koordinatenachsen ergibt.

Die Suche folgt dem intuitiven Ansatz: Gehe den Weg mit der kürzesten Entfernung zur Ressource. Dabei wird die zu suchende Ressource mit der Hashfunktion auf einen Punkt abgebildet. Der direkte Weg ist eine Gerade im Koordinatenraum von der Koordinate des Initiatorknoten zu diesen Punkt. Dieser Weg ist aber i. a. nicht möglich, da im Initiatorknoten nur die in seinem Teilraum liegenden Punkte (mit den zugehörigen Schlüssel- Ressourcen- Paaren) verwaltet werden. Die Suche muss also über die Nachbarknoten erfolgen. Dazu wird der Nachbar, dessen Koordinate die kürzeste Entfernung zum gesuchten Punkt hat, ausgewählt und die Suchanfrage an diesen gesendet. Falls dieser den gesuchten Punkt verwaltet, liefert die Suche diesen Knoten zurück. Fällt der Punkt nicht in den Bereich des Nachbarknotens, so verfährt dieser wie der Initiatorknoten und wählt den Nachbarn mit der kürzesten Entfernung aus und sendet die Suchanfrage an diesen weiter. Dieser Vorgang wird so lange wiederholt, bis der den Punkt verwaltende Knoten gefunden ist.

Eine gesuchte Ressource wird nach $O(\frac{D}{4} * N^{\frac{1}{D}})$ Schritten gefunden oder definitiv als nicht vorhanden erkannt. Der Speicheraufwand für die Nachbarn ist im Gegensatz zu CHORD für beliebig große Peer- to- Peer Systeme konstant (O(D)). Die Suche hat wie das CHORD Protokoll den Nachteil, dass unscharfe Suchen nur schlecht unterstützt werden. Ähnliche Schlüssel werden durch die gleichmäßige Hashfunktion auf sehr unterschiedliche Bereiche im Koordinatenraum abgebildet. So sind Ressourcen mit ähnlichen Schlüsseln nicht in eng beieinander liegenden Knoten zu finden. Ebenfalls muss ein Stabilisierungsalgorithmus für die Korrektheit der Suche bei ausgefallenen Knoten sorgen. Dabei werden die Aufgaben des ausgefallenen Knotens von einem

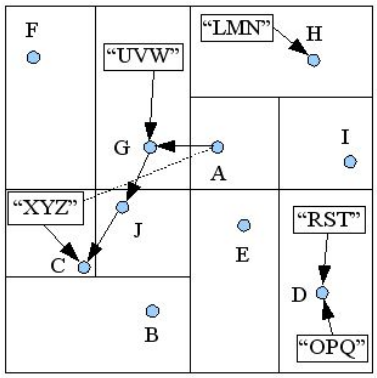


Abbildung 3 – 10

Nachbarknoten übernommen. Die genaue Funktionsweise wird in [RF01] dargestellt.
Abbildung 3 – 10 zeigt ein Beispiel für D = 2. Es sind 10 Knoten und 5 Ressourcen vorhanden. Der Knoten ”A“ kennt die Knoten ”E“, ”H“, ”I“ und ”G“ als Nachbarn. Die Knoten ”D“ oder ”J“ sind ihm nicht bekannt. Der Knoten ”A“ stellt eine Suchanfrage nach der Ressource mit den Namen “XYZ”. Entsprechend der Hashfunktion wird der Namen der Ressource auf den 2- dimensionalen Koordinatenraum abgebildet. Der kürzeste Weg zum gefundenen Hashwert wird in der Abbildung als gestrichelte Gerade dargestellt.
Knoten ”A“ stellt fest, dass er den Punkt nicht verwaltet. Er sucht den Nachbarn mit der kürzesten Entfernung zwischen den Koordinaten der Nachbarn und der des gesuchten Punktes. Dies trifft auf ”G“ zu. In Folge dessen sendet ”A“ die Suchanfrage an ”G“. Der Knoten ”G“ verwaltet ebenfalls nicht den gesuchten Punkt. Da die kürzeste Entfernung zwischen den Koordinaten zu ”J“ besteht, leitet ”G“ die Suchanfrage an ”J“ weiter. Auch ”J“ beinhaltet nicht den gesuchten Punkt und leitet die Anfrage an ”C“ weiter. Der Knoten ”C“ verwaltet den Punkt der gesuchten Ressource. Die Suche ist beendet. Als Ergebnis wird ”C“ an den Knoten ”A“ zurückgeliefert.

3.5 Lokalisierung in hybriden P2P Systemen

In hybriden P2P Systemen ist die Suche zentral organisiert. Jedes teilnehmende Peer meldet sich bei einem zentralen Server an und gibt eine Liste seiner Ressourcen (mit Metainformationen) dem Server bekannt. Wird eine Suchanfrage gestellt, so wird diese an den Server geleitet, von diesem bearbeitet und eine Liste von Knoten, welche die gewünschte Ressource bereitstellen, an den anfragenden Knoten gesendet. Dieser stellt dann eine Verbindung her. Der Vorteil der hybriden

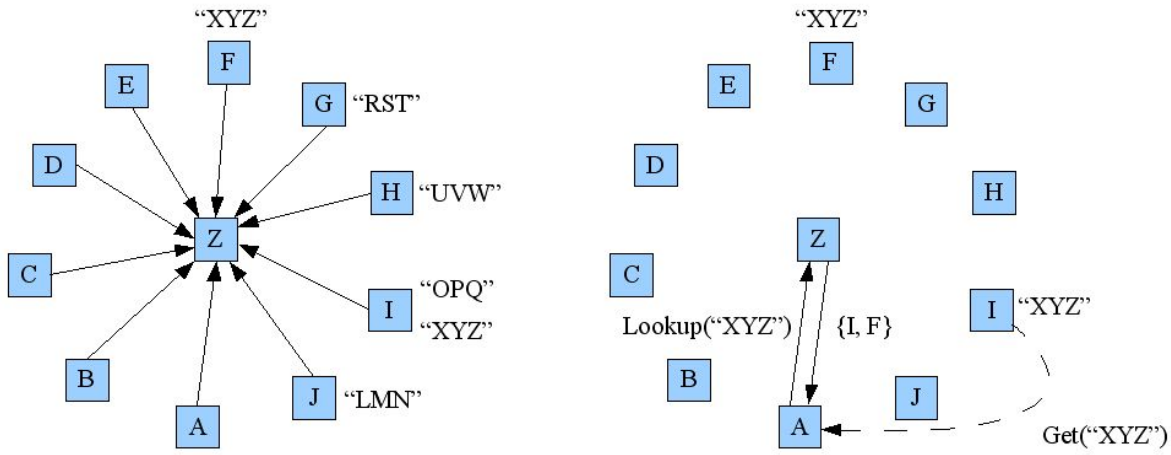


Abbildung 4 – 1

Peer- to- Peer Architektur ist die Effizienz der Suche. Eine zentral verwaltete Suchfunktion ist wesentlich leistungsfähiger als in puren Peer- to- Peer Systemen. Zum Skalieren von großen Systemen werden mehrere zentrale Server eingesetzt. Hier sind die Probleme, die mit der Kommunikation der Server entstehen, zu lösen. Die Skalierbarkeit ist aber weiterhin begrenzt. Lösungsvorschläge finden sich in [YM01]. Problematisch ist auch, dass die zentralen Server einen "single point of failure" darstellen. Sie sind durch gezielte Angriffe oder durch Zensur leicht auszuschalten. Dadurch kann das gesamte Peer-to- Peer System zusammenbrechen.
Abbildung 4 – 1 erklärt die Suche beispielhaft. Nachdem sich die teilnehmenden Knoten mit dem zentralen Server verbunden haben, senden sie vorhandene Metadaten über die Ressourcen an den Server. Knoten "A" stellt eine Suchanfrage nach der Ressource "XYZ". Diese leitet er an den zentralen Server weiter. Der Server bearbeitet die Anfrage und sendet das Ergebnis an Knoten "A" zurück, der nun eine direkte Verbindung zu Knoten "I" oder "F" aufbauen kann.

4. Ausblick auf P2P Datenbanken

Derzeitige populäre Peer- to- Peer Systeme arbeiten hauptsächlich mit Dateien und bieten nur unzureichende Anfragemöglichkeiten. Häufig kann nur nach Stichwörtern, die den Namen der Ressource wieder spiegeln, gesucht werden. Strukturen innerhalb von Ressourcen bleiben bei der Suche außen vor. Weiter ist die Konsistenz der Daten und deren Dauerhaftigkeit in einem Peer- to- Peer System nicht gewährleistet. Die Verschmelzung von Peer- to- Peer Systemen und

Datenbanktechnologie bietet völlig neue Möglichkeiten, stellt aber andererseits neue Herausforderungen. Insbesondere müssen Lösungen zu der in einem Peer- to- Peer nicht gegebene Dauerhaftigkeit der Daten, der möglichen Inkonsistenz durch unvollständige oder überlappende Daten, der fehlende Transaktionsunterstützung sowie in Hinsicht auf die Datensicherheit gefunden werden.

Anwendungsszenarien sind in der Medizin, zur Unterstützung von Workflows oder Geschäftsprozessen vorstellbar. Hier wäre eine ad- hoc Ressourcenintegration, eine ad- hoc Speichermöglichkeit oder die Nutzung verschiedener high- level Anfragesprachen denkbar.

In [RK03] wird eine P2P XML Datenbank vorgestellt, die XML- Dokumente verteilt speichert und verwaltet. Jeder Knoten im System speichert dabei nur einen Teil des Dokuments. Verteilte Hashtabellen werden zur Repräsentation der baumartigen Struktur des XML- Dokuments benutzt. Als Schnittstellen für Applikationen bietet das System eine Query Engine und das DOM Interface. Die Abbildung 4-1 zeigt die Architektur im Überblick.

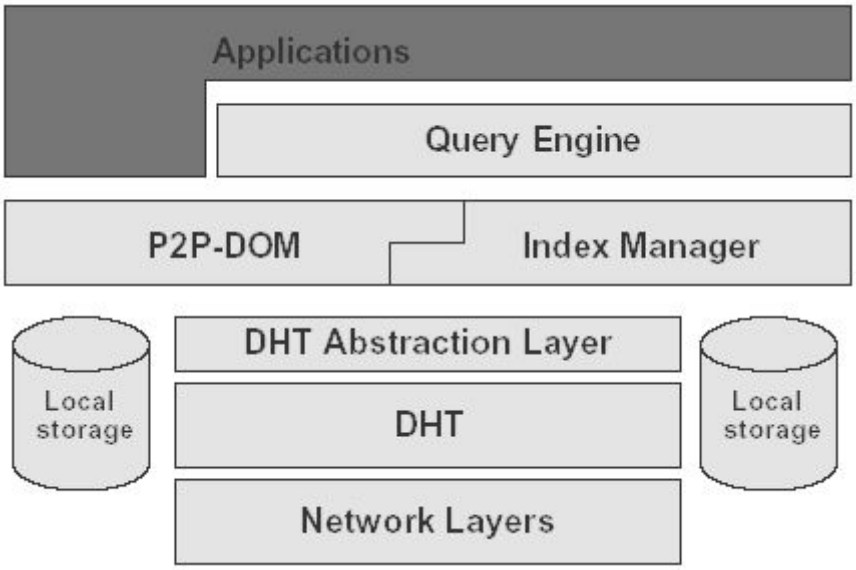
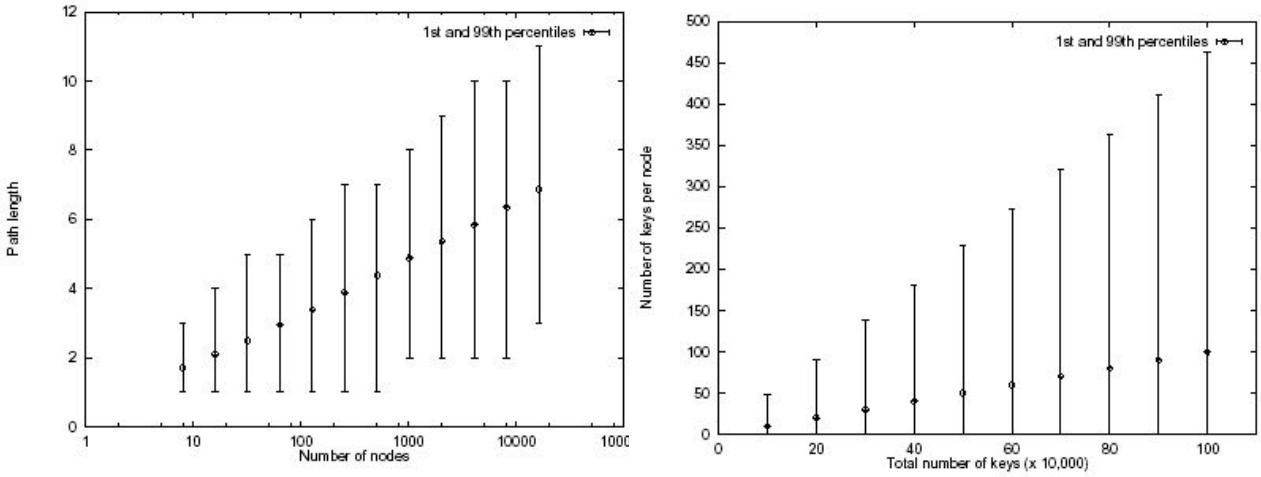


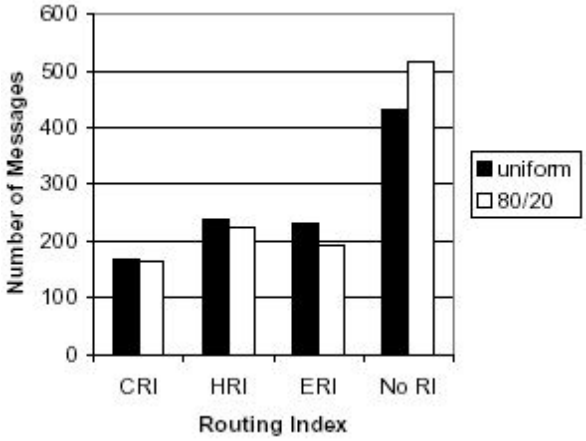
Abbildung 4 – 1 Quelle:[RK03]



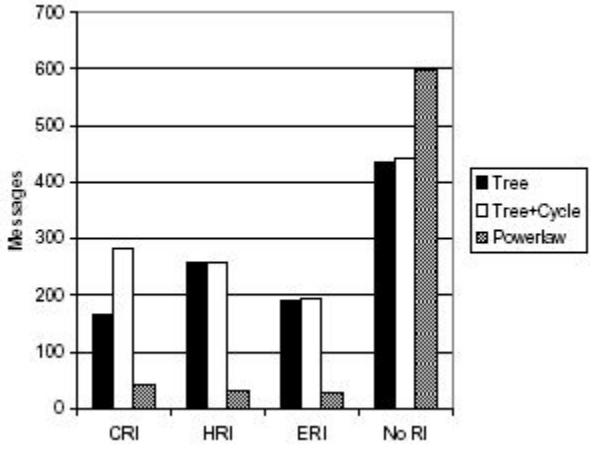
a) Abbildung 5-1 b)

5. Evaluierung

Beispielhaft werden Evaluierungen von CHORD und von den Verfahren mit Routing Indizes vorgestellt, um die in den jeweiligen Kapiteln gezeigten Vor- bzw. Nachteile nochmals aufzuzeigen. In Abbildung 5-1 ist eine Evaluierung von CHORD gezeigt (Quelle: [SM02]). In Abbildung 5-1 a) ist der Zusammenhang zwischen Pfadlänge, um eine Ressource zu finden, und die Anzahl der Knoten dargestellt. Es ist gut zu erkennen, dass die mittlere Pfadlänge logarithmisch steigt (Achtung: Exponentielle Skala bei der Knotenanzahl!). In Abbildung 5-1 b) ist die Anzahl der von einem Knoten verwalteten Ressourcen in Abhängigkeit von der Gesamtanzahl der Ressourcen gezeigt. Die Knotenanzahl beträgt dabei 10000. Es ist zu erkennen, das sich die Anzahl der verwalteten Ressourcen pro Knoten nahezu gleich verteilt. Dies ist der erwartete Zustand.
In Abbildung 5-2 ist eine Evaluierung von dem Verfahren mit Routing Indizes gegeben (Quelle: [CM02]). Es werden die vorgestellten Methoden Compound Routing Indizes, Hop- Count Rounting Indizes und die hier nicht vorgestellte Methoden Exponential Routing Indizes mit der naiven Suche verglichen. In Abbildung 5-2 a) ist die Anzahl der Nachrichten zum Finden einer Ressource bei einer Gleichverteilung der Ressourcen und bei einer Verteilung nach der 80/20 Regel in Abhängigkeit von der benutzten Suchmethode gezeigt. Die Knotenanzahl beträgt 60000. Es wird deutlich, dass das Verfahren mit Routing Indizes Vorteile bezüglich dem ohne Routing Indizes hat. Der Einfluss des Struktur auf die Nachrichtenanzahl ist in Abbildung 5-2 b) gegeben. Hier ist deutlich zu erkennen, dass das Verfahren mit Routing Indizes dem Verfahren ohne Routing Indizes überlegen ist. Besonders deutlich wird dies bei der Powerlaw- Topologie. Diese entspricht in etwa

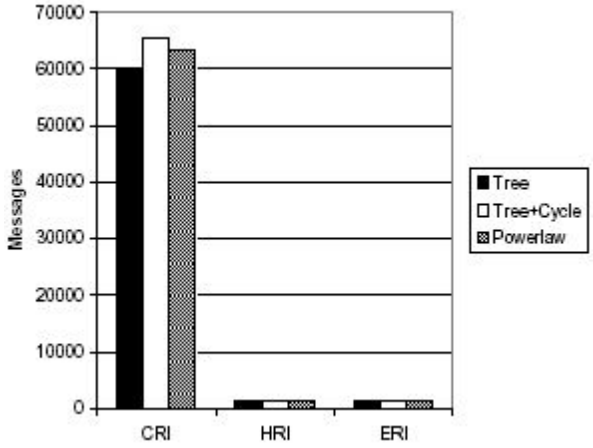


a)



b)

Abbildung 5-2



c)

einem realen P2P- System. In der Abbildung 5-2 c) ist der Änderungsaufwand bei den Methoden zum Verfahren mit Routing Indizes dargestellt. Hier ist deutlich zu sehen, dass bei der Methode Compound Routing Indizes eine Änderung im System zu jedem Knoten propagiert werden muss. Sind Zyklen im System vorhanden ist der Änderungsaufwand natürlich größer, da diese erkannt werden müssen.

Zusammenfassung

Zur Bewältigung der enormen Kommunikationsbedürfnisse in der heutigen Zeit wurden verschiedene Technologien entwickelt. Eine ist die Peer- to- Peer Technologie. Hier müssen die Probleme, die sich mit der Dezentraliesierung ergeben, gelöst werden. Die vorhandenen Verfahren

versuchen den Such- und Verwaltungsaufwand von Ressourcen zu minimieren. Es ist aber noch viel Entwicklungsaufwand nötig, damit sich die Technologie nicht nur im Filesharingbereich durchsetzen kann.

Bei den hier vorgestellten Verfahren bieten die auf verteilten Hashtabellen aufbauen Verfahren Vorteile. Sie erlauben eine schnelle Ressourcenlokalisierung und eine effiziente Ressourceverwaltung. Auch sind sie als Dienstanbieter für neue Anwendung geeignet. Ein Ansatz wäre die oben vorgestellte Verschmelzung mit der Datenbanktechnologie. Verfahren, die auf verteilten Hashtabellen aufbauen, haben den Nachteil, dass die Performanz bei sehr dynamischen Systemen leidet. Hier sind die anderen vorgestellten Verfahren besser geeignet. Der große Aufwand zum Verwalten der teilnehmenden Knoten ist hier nicht so leistungsentscheidend. Trotzdem wirkt sich der hohe Suchaufwand bzw. Verwaltungsaufwand negativ aus, weshalb diese Systeme nicht zu empfehlen sind.

Literaturverzeichnis

[CM02] Crespo, A., Garcia-Molina, H.: Routing Indices For Peer- to- Peer Systems. Proceedings of the International Conference on Distributed Computing Systems (ICDCS). Juli 2002.

[GB03] Gebhardt, S.: Einführung. Problemseminar: Peer- to- Peer (P2P) Data Management. Universität Leipzig. 2003.

[RF01] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. Proceedings of ACM SIGCOMM. 2001.

[RK03] Risse, T., Knezevic, P.: A Peer- to- Peer XML Database. Fraunhofer IPSI. 2003.

[SC02] Schollmeier, R.: A Definition of Peer-to-Peer Networking for the Classification of Peer- to-Peer Architectures and Applications. Proceedings of the First International Conference on Peer-to-Peer Computing (P2P.01). 2002. (Übersetzer: Autor)

[SM02] Stoica, I. et al.: Chord: A Scalable Peer- to peer Loopup Service for Internet Applications. Proc. Of the 2001 conference on applications, technologies, architectures, and protocols for computer communications. 2001.

[YM01] Yang, B., Garcia-Molina, H.: Comparing Hybrid Peer-to-Peer Systems. Proceedings of Very Large Databases(VLDB). 2001.

[YM02] Yang, B., Garcia-Molina, H.: Efficient Search in Peer-to-Peer Networks. Proceedings of the International Conference on Distributed Computing

Systems (ICDCS). Juli 2002.