Grid Computing und Peer-to-Peer Systeme

Seminar SS 2004

Herausgeber:
Florin Isaila, Guido Malpohl,
Vlad Olaru, Jürgen Reuter

Universität Karlsruhe
Institut für Programmstrukturen
und Datenorganisation

Autoren:
Hannes Kälber, Kirill Müller,
Martin Scheerer, Alexander Dieterle,
Johanner Lieder, Thomas Blattmann,
Mehmet Soysal, Timo Reimann,
Christian Würdig, Martin Heine

Vorwort

Im Sommersemester 2004 wurde im Seminar „Grid Computing und Peer-to-Peer Systeme“ eine Reihe aktueller Themen aus den Grid Computing, Peer-to-Peer Systeme und Ad-Hoc Netzwerkem Gebieten angeboten.

Jeder Teilnehmer wählte hieraus ein Thema, um darüber in der Form eines medial gestützten Vortrages zu referieren. Um allen Teilnehmern die Gelegenheit zu geben, aus diesem Seminar nachhaltig etwas mitzunehmen, fertigte jeder Vortragende eine allen zugängliche schriftliche Ausarbeitung an. Die Ausarbeitungen finden sich in leicht redigierter Fassung durch die Editoren im vorliegenden technischen Bericht wieder.

Inhaltsverzeichnis

Kapitel 1

Distributed Hash Tables

Seminarbeitrag von **Hannes Kälber**

Dieser Beitrag stellt verschiedene Verfahren vor, die verwendet werden, um verteilte Hashtables in einem Rechnernetzwerk zu realisieren und vergleicht diese bezüglich ihrer Eigenschaften im regulären Betrieb, wie auch ihr Verhalten im Fehlerfall. Abschließend wird der Versuch einer Bewertung der vorgestellten Verfahren unternommen.

1.1 Einleitung

Die in den letzten Jahren rapide zugenommene Vernetzung sowohl auf lokal begrenzter als auch auf globaler Ebene hat dazu geführt, dass man anfing, sich Gedanken über geeignete Datenstrukturen zu machen, die von den Vorteilen eines Rechnerverbundes gegenüber einem einzelnen Rechnersystem profitieren. Wesentlich hierbei ist, dass in einem solchen Rechnerverbund – verglichen mit einem klassischen Großrechner – sehr viel Rechenleistung und Speicherplatz zu geringen Kosten zur Verfügung steht. Des Weiteren kommt noch der Vorteil der geografischen Unabhängigkeit hinzu: Die beteiligten Systeme können physikalisch an einem beliebigen Ort stehen, solange dieser über eine Datenverbindung zu anderen Standorten verfügt. Ebenso können einem solchen Verbund dynamisch Rechner hinzugefügt und wieder entzogen werden, so dass sich eine bessere Auslastung der existierenden Hardware ergibt.

Bezüglich der Organisation eines solchen Rechnerverbundes gibt es im Wesentlichen zwei Möglichkeiten: Zum einen kann es eine zentrale Instanz geben, zum anderen können sich die beteiligten Knoten des Rechnernetzes auch gleichberechtigt verhalten. Bei den hier vorgestellten Distributed Hash Tables (DHT) liegt der zweite Fall vor. Doch was ist eigentlich ein DHT?

Ein DHT ist zunächst nichts weiter als eine normale Hash-Tabelle, wie sie allgemein bekannt ist: Von den zu speichernden Daten wird mittels einer Hash-Funktion

ein Hash-Wert ermittelt, der dann als Index darüber entscheidet, an welcher Position in der Hash-Tabelle die entsprechenden Daten hinterlegt werden. Also sind auf jeder Hash-Tabelle zwei Funktionen `store(key, value)` und `value=lookup(key)` verfügbar.

Somit ist der wesentliche Unterschied zwischen einer verteilten und einer normalen Hash-Tabelle deutlich: Da der Hashtable auf mehreren Rechnerknoten verteilt liegt, muss zusätzlich noch ermittelt werden, welcher Knoten für diesen Eintrag zuständig ist. Die verschiedenen Mechanismen werden im nächsten Abschnitt vorgestellt werden.

1.2 Verschiedene DHTs

Nachfolgend wird ein Überblick über die existierenden Verfahren für DHTs gegeben. Gezeigt werden die Operationen *Auffinden des zuständigen Knotens* (und damit das Routing im DHT), *Verhalten bei hinzukommenden Knoten* und *Verhalten bei verschwindenden Knoten*. Ebenso wird das *Verhalten bei unerwartet verschwindenen Knoten* beschrieben.

Nicht eingegangen wird darauf, wie die Daten letztendlich auf einem Knoten abgelegt werden, da dies ein Problem ist, dass unabhängig von der Struktur des verwendeten DHT ist. Ebenso nicht betrachtet werden hier Aspekte der Sicherheit eines DHT, also die Frage, ob man Daten böswillig verändern kann oder das Auffinden von Schlüsseln stören kann.

1.2.1 CAN

CAN (Content-Addresseable Network) [RFH+01] ist ein im Jahre 2002 in Berkeley entwickelter DHT, der einen d-dimensionalen Torus verwendet. Für den Fall $d = 2$ bedeutet dies also, dass der Koordinatenraum die Oberfläche eines gewöhnlichen Torus ist. Dieser Koordinatenraum ist virtuell; er ist also unabhängig von der physikalischen Anordnung der Knoten.

Zu jedem Zeitpunkt ist der Raum vollständig in d-dimensionale rechteckige Zonen partitioniert, d.h. jedem Punkt im Raum wird genau eine Zone zugeordnet und jeder Knoten übernimmt die Kontrolle für eine Zone. Da die Anzahl der Knoten im DHT dynamisch sein soll, muss die Zerlegung des Koordinatenraumes ebenfalls dynamisch sein.

Der virtuelle Koordinatenraum wird verwendet, um Informationen der Form `(key, value)` im DHT zu speichern. Dafür wird über eine Hash-Funktion jeder `key` einem Punkt P im virtuellen Raum zugeordnet und das `(key, value)`-Tupel wird auf demjenigen Knoten gespeichert, der für die Zone zuständig ist, in der der Punkt P liegt. Soll nun ein Eintrag im DHT gesucht werden, so wird wieder über die Hashfunktion der zugehörige Punkt ermittelt und die Daten können vom zugehörigen Knoten gelesen werden.

Es bleibt nun noch zu klären, wie die Abbildung von einem Punkt auf einen entsprechenden Knoten abläuft. Hierzu benötigt man ein geeignetes Routingverfahren, um von einem beliebigen Knoten zum Zielknoten zu gelangen.

Dieses Routing basiert auf den Informationen, die jeder Knoten über seine benachbarten Zonen hat. Dies sind neben der Netzwerkadresse des Nachbarknotens auch die virtuellen Koordinaten der zugehörigen Zone. Dabei sind zwei Zonen benachbart, wenn sich ihre Koordinatenbereiche in $d-1$ Dimensionen überlappen und in einer Dimension berühren. Jede Zone hat bei gleichgroßen Zonen somit $2d$ Nachbarn; wie später noch gezeigt wird, kann sich dieser Wert durch das Teilen von Zonen erhöhen. Wird nun von einem Knoten eine Nachricht (Auffinden oder Hinzufügen

eines (`key, value`)-Tupels) gesendet, so wird dieses an die jeweils passendste Nachbarzone weitergeleitet, bis die Nachricht bei der zuständigen Zone angekommen ist. Die passendste Zone ist diejenige, die zum Zielpunkt P den geringsten Abstand hat.

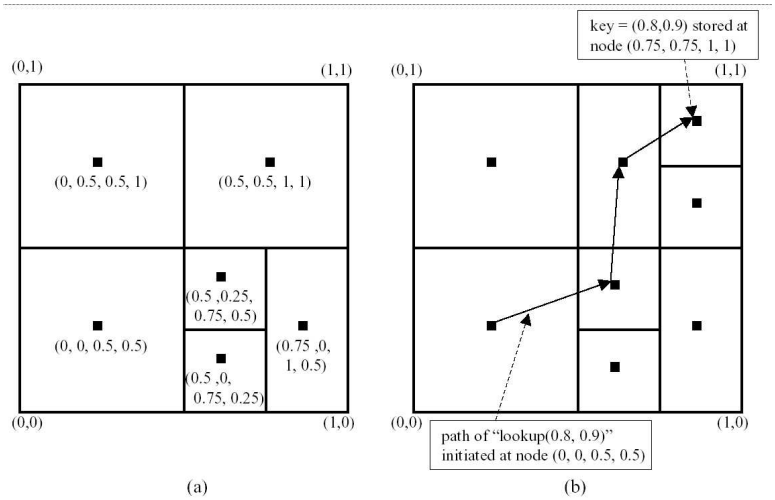

Abbildung 1.1: Unterteilung des zweidimensionalen Adressraumes in sechs Zonen. In (b) wird die Suche nach Schlüssel (0.8, 0.9) gezeigt.

Die Anzahl der Knoten im DHT soll dynamisch sein, also müssen neue Knoten eingebunden werden und andere Knoten den DHT verlassen können. Zunächst soll das Hinzufügen eines neuen Knotens betrachtet werden. Da der gesamte virtuelle Adressraum vollständig zerlegt ist, muss eine bereits existierende Zone geteilt werden. Welche Zone geteilt wird, wird zufällig vom neuen Knoten bestimmt. Hierfür wählt sich dieser einen beliebigen Punkt im Adressraum aus; die zugehörige Zone wird dann geteilt. Zunächst einmal muss jedoch ein Knoten bekannt sein, der bereits im DHT eingebunden ist. Auf welche Weise dieses geschieht, ist grundsätzlich irrelevant.

Für CAN existieren z.B. sogenannte Bootstrap-Nodes, auf die über DNS-Einträge verwiesen wird. Diese Bootstrap-Nodes halten dann eine Liste von zufällig gewählten Knoten vor, die an den neuen Knoten zurückgegeben wird. Ausgehend von einem solchen Start-Knoten wird dann über die oben beschriebenen Routing-Mechanismen der Zielknoten aufgesucht. Der Zielknoten halbiert nun seine Zone und ordnet die so neu entstandene Zone dem neuen Knoten zu. Um später Knoten das Verlassen des Verbundes zu erlauben, muss die Teilung in einer bestimmten Richtung erfolgen. So wird üblicherweise abwechselnd in X-Richtung und Y-Richtung geteilt.

Anschließend müssen noch die Routing-Einträge bei den betroffenen Knoten erneuert werden. Der neue Knoten kann diese Informationen vom alten Knoten bekommen. Sowohl neuer als auch alter Knoten müssen diejenigen Knoten aus ihren Routing-Tabellen entfernen, die nun nicht mehr Nachbarn sind. Danach müssen die beiden Knoten ihre jeweiligen Nachbarn über die Änderung in der Zoneneinteilung informieren.

Wie zu erkennen ist, hängt der Aufwand für die Aktualisierung der Routinginformationen beim Hinzufügen weiterer Knoten nur von der Anzahl der Dimensionen des virtuellen Adressraumes ab, nicht aber von der Anzahl der Knoten im System insgesamt.

Es bleibt nun noch zu betrachten, wie das Verlassen eines Knotens stattfindet. Hierbei tritt nun das Problem auf, dass die Zone, für die der Knoten zuständig

ist, danach keinem Knoten mehr zugeordnet wäre. Da dies nicht sein darf, muss einer der Nachbarknoten die Zuständigkeit für diese Zone übernehmen und die bisher auf dem verlassenden Knoten gespeicherten Daten zu sich übertragen. Wenn es eine Nachbarzone gibt, die mit der Zone des verlassenden Knotens zusammen eine größere gültige Zone bilden kann, so wird diese Verschmelzung durchgeführt. Ist eine Verschmelzung nicht möglich, so übernimmt der Knoten mit der *kleinsten* Zone vorübergehend die Verantwortung für die verlassene Zone.

Im Falle eines Fehlers, also dem Ausfall eines Knotens oder einer Netzwerkverbindung, müssen geeignete Maßnahmen ergriffen werden, um die Konsistenz des DHTs weiterhin sicherzustellen. Die Strategie hierfür ist die, dass einer der Nachbarn des ausgefallenen Knotens verübergehend die Verantwortung für den ausgefallenen Bereich übernimmt. Allerdings sind sämtliche auf dem ausgefallenem Knoten gespeicherte Daten nicht mehr verfügbar. Aus diesem Grund sollten die Daten im DHT regelmäßig von derjenigen Stelle aktualisiert werden, die sie ursprünglich in den DHT eingebracht hat.

Um den Ausfall eines Knotens zu erkennen, wird auf die Tatsache zurückgegriffen, dass im regulären Betrieb jeder Knoten Update-Nachrichten an seine Nachbarn sendet, in denen er die Koordinaten seiner Zone und eine Liste seiner Nachbarn mit den dazugehörigen Zonenkoordinaten bekannt gibt. Wird von einem Knoten für einen bestimmten Zeitraum eine solche Nachricht nicht empfangen, so wird angenommen, dass der zugehörige Absender ausgefallen sein muss. Wenn ein Knoten festgestellt hat, dass sein Nachbar ausgefallen zu sein scheint, so setzt er einen Übernahme-Machanismus in Gang, der zunächst einen Übernahme-Timer startet. Dieser erhält einen Startwert, der proportional zur Größe der eigenen Zone ist. Sobald dieser Timer abgelaufen ist, sendet der übernehmende Knoten eine entsprechende TAKEOVER-Nachricht an die übrigen Nachbarn des ausgefallenen Knotens; diese sind ihm durch die regelmäßigen Routing-Updates bekannt. Empfängt ein Knoten eine TAKEOVER-Nachricht, so vergleicht er die Größe seiner Zone mit der Zonengröße des Absenders der TAKEOVER-Nachricht, so dass derjenige Knoten, dessen Zone am kleinsten ist, zuständig für die Zone des ausgefallenen Knotens wird.

Da dieser Mechanismus nur kleinere Ausfälle beheben kann und es zudem dazu kommen kann, dass ein Knoten für mehrere Zonen zuständig ist, kann ein Knoten mit Hilfe seiner Routing-Tabelle und denen seiner Nachbarn Zonen ausfindig machen, die sich günstig zu einer größeren vereinigen lassen. Durch eine anschließende Umordnung der Zuordnung von Zonen zu Knoten kann so wieder der Zustand erreicht werden, in dem jeder Knoten für genau eine Zone zuständig ist.

1.2.2 Chord

Chord ist ebenfalls im Jahre 2002 am MIT entwickelt worden [SMK+01]. Es stellt ein Verfahren zur Organisation eines DHT dar und bietet somit die bekannten Funktionen auf DHTs an.

Wie schon bei CAN, so ist auch bei Chord die Zuordnung eines Schlüssels zu einem Knoten essentiell. Aus Gründen einer möglichst guten Lastverteilung muss diese Zuordnung eine gleichmäßige Verteilung liefern. Diese Funktion bildet sowohl eine Knoten-ID (wie z.B. die IP-Adresse) als auch die Schlüssel, die gespeichert werden sollen, in einen zirkulären Adressraum mit Adressen der Länge m bit ab. Für diese Abbildung wird üblicherweise ein Hash wie SHA-1 verwendet. Ähnlich wie bei CAN bekommt auch bei Chord jeder Knoten einen Teil des Adressraumes zugewiesen; auch hier ist die Zerlegung vollständig. Der Bereich, für den ein Knoten zuständig ist, wird folgendermaßen festgelegt: Von einer Adresse i an wird der in aufsteigender Adressrichtung nächste Knoten gesucht, dessen Adresse größer oder gleich i ist. Auf diese Art ist eindeutig festgelegt, auf welchem Knoten ein bestimmtes `(key, value)`-Tupel gespeichert wird.

Um jedoch eine effiziente Suche nach dem richtigen Knoten (bzw. seiner Netzwerkadresse) durchführen zu können, benötigt man ein geeignetes Routingverfahren. Im Fall von Chord sieht dieses so aus, dass in einem Raum mit 2^m Adressen jeder Knoten m Einträge in seiner Routingtabelle hat. Diese Einträge werden als *Finger* bezeichnet. Dabei wird der i-te *Finger* des Knoten n folgendermaßen bestimmt: Es wird zunächst $f_i(n) = (n + 2^{i-1}) \bmod 2^m$ berechnet, anschließend der für die Adresse $f_i(n)$ zuständige Knoten ermittelt. Der entsprechende i-te-Routingeintrag auf n zeigt nun auf diesen Knoten. Hierbei ist offensichtlich, dass der erste *Finger* ($i = 1$) von n der direkt auf n folgende Knoten ist. Ebenso fällt auf, dass sehr viele Routing-Einträge auf Knoten in relativer Nähe zu n zeigen, während die Anzahl von Routing-Einträgen auf weit entfernte Knoten nur sehr gering ist. Genauer: Die Anzahl an Verweisen auf andere Knoten nimmt exponentiell mit deren Entfernung zu n im Adressraum ab.

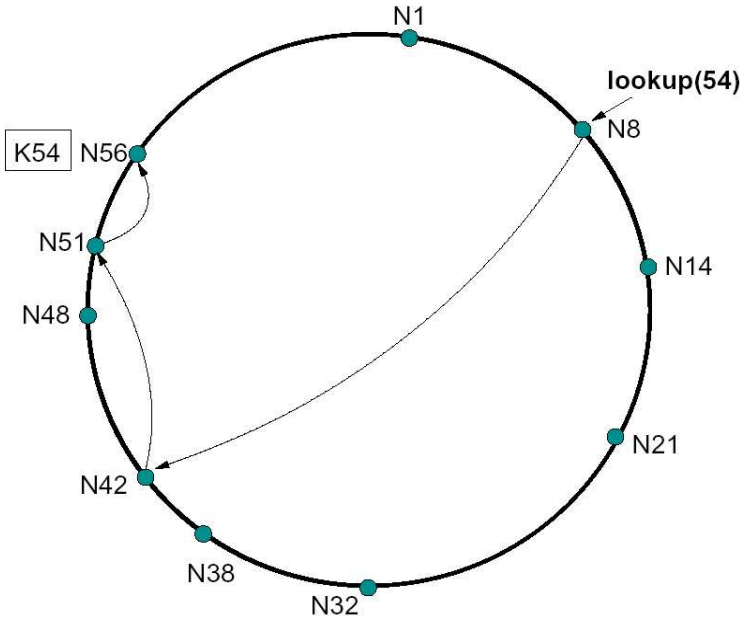

Abbildung 1.2: Suche nach dem Schlüssel mit der Adresse 54. Die Pfeile zeigen den Weg des lookups.

Soll nun der zu einem Schlüssel k zugehörige Knoten gefunden werden, geht Chord wie folgt vor. Zunächst wird an einen beliebigen Knoten n_0 die Anfrage nach k gerichtet. Höchstwahrscheinlich ist in der finger-Tabelle von n kein Eintrag, der direkt zum Zielknoten führt, gespeichert. Also muss n_0 die Anfrage weiterleiten, wozu er aus der finger-Tabelle denjenigen Knoten n_1 sucht, der numerisch unmittelbar vor k liegt[1]. Die Anfrage wird nun von n_1 bearbeitet und von dort ggf. an einen Knoten n_2 weitergeleitet. Dieses setzt sich solange fort, bis der Knoten n_i erreicht ist, der tatsächlich zuständig für k ist. Der Knoten n_i erkennt dies daran, dass k zwischen n_i und $f_i(n)$, also zwischen dem Knoten selber und dem nächsten Knoten im Adressraum, liegt. Da näher an k gelegene Knoten detailliertere Routinginformationen haben, findet die Annäherung an k sehr schnell statt.

Nun bleibt noch zu betrachten, wie Chord bei der Aufnahme eines neuen Knotens verfährt. Um diesen Vorgang effizienter zu gestalten und Chord auch stabiler zu machen, speichert Chord neben der Finger-Tabelle auch noch einen Verweis auf seinen direkten Vorgänger-Knoten. Soll nun ein neuer Knoten n hinzugefügt werden,

[1] Hier werden aus Gründen der Einfachkeit der Schlüssel k und seine ID synonym verwandt.

so wird der neue Knoten zunächst mit den Informationen aus einem bereits existierendem Knoten e initialisiert, d.h. die Finger-Tabelle wird ebenso übernommen wie der Zeiger auf den Vorgänger-Knoten. Um nun das Routing wiederherzustellen, müssen die Finger-Tabellen auf n und e an die neuen Gegebenheiten angepasst werden. Zusätzlich müssen die Finger-Tabellen aller anderen Knoten ebenfalls überprüft und ggf. aktualisiert werden, da n ja ggf. in einer der Finger-Tabellen auftauchen kann. Der Fall, dass $f_i(p)$ auf n zeigt, also n der i-te Finger von p ist, tritt genau dann ein, wenn der Abstand von n zu p mindestens 2^{i-1} beträgt und $f_i(p)$ momentan auf einen Wert größer n zeigt.

Der Algorithmus zur Anpassung der Finger-Tabellen startet beim i-ten Finger von n und wandert dann über den Vorgänger-Verweis den Chord-Ring rückwärts entlang. Nun wird bei jedem Knoten geprüft, ob die Finger-Tabelle korrekt ist und bei Bedarf korrigiert. Der Algorithmus bricht ab, sobald ein Knoten erreicht wird, dessen i-ter Finger auf einen Knoten zeigt, der vor n liegt. Dieser Algorithmus wird dann für das nächste i erneut gestartet. Auch wenn es den Anschein erweckt, dass dieses Verfahren einen sehr hohen Aufwand hat, liegt die Anzahl der Einträge, die tatsächlich angepasst werden müssen, im Mittel nur in $O(m * \log N)$, wobei m die Länge der verwendeten Adressen in Bit ist und N die Anzahl der Knoten im DHT angibt. Als letzter Schritt verbleibt nun noch die Aufgabe, von dem Nachfolger von n diejenigen Schlüssel auf n zu übertragen, für die n nun verantwortlich ist. Diese Wartungsoperation wird jedoch nicht bei jedem Einfügen erreicht, da die Funktionstüchtigkeit von Chord durch die direkten Verweise auf die Nachfolgerknoten immer gewährleistet ist.

Das Entfernen eines Knotens aus einem Chord-Ring verläuft ähnlich. Hier müssen sämtliche Schlüssel, die auf dem verlassenden Knoten v liegen, auf den Nachfolger von v übertragen werden. Anschließend bleibt noch die Anpassung der Finger-Tabellen auf den übrigen Knoten, die aber grundsätzlich genauso verläuft wie beim Hinzufügen eines Knotens.

Im Falle eines Ausfalls eines Knotens übernimmt sein Nachfolger die Zuständigkeit für diesen Knoten. Daher müssen alle Knoten, die einen Verweis auf diesen Knoten haben, seinen Nachfolger finden können. Um einen Ausfall entdecken zu können, senden alle Knoten ihren Nachfolgern regelmäßig Nachrichten. Bleibt eine solche Nachricht vom Vorgänger aus, musss ein Knoten aus seiner Finger-Tabelle den ersten ansprechbaren Knoten entnehmen und diesen als seinen Nachfolger eintragen. Allgemein wird so auch verfahren, wenn weitere Knoten aus der Finger-Tabelle nicht ansprechbar sind.

Um im Falle eines Ausfalls dennoch keine Datenverluste zu erleiden, ist in Chord eine Zahl r definiert, die die Redundanz der Tupel angibt. Diese Redundanz bewirkt, dass bei einer Speicheroperation auf einem Knoten die r Nachfolger des speichernden Knotens ebenfalls die Daten speichern. Da bei einem Ausfall auf einen der nächsten Knoten zugegriffen wird, ist sichergestellt, dass es trotzdem nicht zu Datenverlust kommt.

1.2.3 Pastry

Pastry ist im Jahre 2001 veröffentlicht worden [RD01] und stellt einen weiteren Ansatz für DHTs dar. Das Design von Pastry sieht für jeden Knoten eine eindeutige, 128 Bit lange Knoten-ID vor. Grundsätzlich ist es auch bei Pastry unerheblich, wie diese Knoten-ID gewonnen wird. Von den Entwicklern vorgeschlagen wird die Erzeugung dieser ID auf Grundlage eines Hashwertes des öffentlichen Schlüssels eines Knotens oder seiner IP-Adresse. Wichtig ist auch hier, dass die tatsächlich existierenden IDs über den gesamten Adressraum gleichmäßig verteilt sind. Die Schlüssel werden auch hier wieder in den gleichen Adressraum abgebildet wie die Knoten-IDs. Derjenige Knoten, der den geringsten Abstand zu einem bestimmten

Schlüssel hat, übernimmt die Speicherung desselben.

Pastry fasst die Adressen als eine Folge von Ziffern mit der Basis 2^b auf. Der Parameter b ist ein Konfigurationsparameter und hat typischerweise den Wert 4. Auf der Basis dieser Darstellung erfolgt bei Pastry das Routing. Eine Anfrage wird immer so weitergeleitet, dass zunehmend mehr Stellen von Beginn der Knotenadresse mit dem Schlüssel übereinstimmen. Die Routingtabelle eines jeden Knotens enthält $(2^b - 1) * \text{ceil}(\log_{2^b} N) + l$ Einträge, wobei l ein weiterer Konfigurationsparameter ist, der ganzzahlig und gerade sein muss und meistens den Wert 16 hat. Dieser Wert wird weiter unten noch erläutert. N steht hier für die Anzahl der maximal möglichen Adressen, typischerweise ist $N = 2^160$.

Jeder Eintrag in dieser Routingtabelle ordnet einem bestimmten Präfix einen entsprechenden Knoten zu. Die Routingtabelle ist in $\text{ceil}(\log_2^b N)$ Zeilen und $2^b - 1$ Spalten eingeteilt. In der n-ten Zeile der Routingtabelle stehen Verweise auf diejenigen Knoten, deren Knoten-ID in den ersten n Bit mit der des aktuellen Knotens übereinstimmt, aber in der $n + 1$-ten Stelle abweicht. Die Nachricht kann dann an einen Knoten weitergeleitet werden, dessen Präfix in einer weiteren Stelle mit dem gewünschten Ziel übereinstimmt.

Da es für ein bestimmtes Präfix mehr als einen geeigneten Knoten geben kann, muss Pastry entscheiden, welcher Knoten eingetragen wird. Die Funktionalität des Routings ist – unabhängig von der Wahl des genauen Knotens – immer gewährleistet, so dass sich hier die Möglichkeit zur Optimierung bietet. Tatsächlich verwendet Pastry zu genau diesem Zweck eine Metrik, die helfen soll, aus einer Liste von geeigneten Knoten den besten auszusuchen. Häufig kommen hier Verfahren zum Einsatz, die die Struktur des zugrundeliegenden Netzwerkes berücksichtigen, wie z.B. die Round-Trip-Time. Aber auch andere Bewertungskriterien sind denkbar, wie z.B. die Leistungsfähigkeit eines Knotens und die des Weges zu ihm hin.

Zusätzlich zu dieser Routingtabelle hält jeder Knoten auch noch $l/2$ Verweise auf die nächsten Knoten mit größerer ID und ebenso viele Verweise auf die nächsten Knoten mit kleinerer ID. Diese zusätzlichen Verweise werden als *leaf set* bezeichnet. Der zuvor bereits erwähnte Parameter l gibt also die Größe des *leaf sets* vor. Tritt beim Routing der Fall auf, dass zu einem Knoten mit einem bestimmtem Präfix geroutet werden müsste, ein solcher Knoten jedoch nicht existiert, schaut der aktuelle Knoten in seinem leaf set nach demjenigen Knoten, der ein gleichlanges Präfix hat, aber numerisch dichter am gesuchten Ziel liegt. Ein solcher Knoten existiert im leaf set, solange nicht der aktuelle Knoten oder sein direkter Nachbar der zuständige Knoten für das gesuchte Ziel ist.

Soll ein neuer Knoten eingefügt werden, so wird für diesen nach dem eingangs erwähnten Verfahren eine Knoten-ID bestimmt. Nun kontaktiert der neue Knoten zu Initialisierungszwecken einen nahegelegenen[2] Pastry-Knoten. Dieser leitet die Nachricht an denjenigen Knoten k weiter, der numerisch am dichtesten am neuen Knoten liegt. Erreicht wird dieses dadurch, dass die Knoten-ID des neuen Knotens wie ein normaler Schlüssel aufgefasst wird und somit die normalen Routingmechanismen greifen können. Der neue Knoten übernimmt dann das *leaf set* vom Knoten k und trägt in seine Routingtabelle diejenigen Informationen ein, die auf dem Weg vom ursprünglich kontaktierten Knoten bis zu k gesammelt wurden; präziser wird die i-te Zeile des i-ten Knotens auf dem Weg zu k übernommen. Diese Informationen reichen aus, um den neuen Knoten korrekt zu initialisieren und ihn in die Lage zu versetzen, diejenigen Knoten über seine Präsenz zu informieren, für deren Routing er eine Rolle spielen könnte.

Nun bleibt noch zu untersuchen, wie sich Pastry verhält, wenn ein Knoten verschwindet. Doch zunächst muss detektiert werden, dass ein Knoten nicht mehr existiert. Es gibt mehrere Möglichkeiten, wie festgestellt werden kann, dass ein Knoten

[2] Im Sinne der zuvor erwähnten Metrik

ausgefallen ist. Zunächst einmal kann eine Reaktion ausbleiben, wenn ein Knoten k einen anderen aus seinem *leaf set* kontaktieren möchte. In diesem Fall sendet k an den in Richtung des ausgefallenen Knotens am weitesten entferneten Knoten eine Anfrage nach dessen *leaf set*, in dem auf jeden Fall nur Informationen stehen, die k noch nicht hat, so dass k weiter entfernte Knoten kennenlernt. Auf diese Art und Weise kann k sein *leaf set* aktualisieren. Dann kann ein ausgefallener Knoten im Rahmen des regulären Routings bemerkt werden.

Um nun den entsprechenden Eintrag aus Zeile i seiner Routingtabelle durch einen anderen ersetzen zu können, sendet k an einen beliebigen anderen Host aus der Zeile i seiner Routingtabelle einer Anfrage nach einem lebendigen Knoten der für den zu aktualisierenden Routingeintrag zuständig ist. Sollte keiner der befragten Knoten aus der Zeile i ein gültiges Routingziel kennen, so fragt er einen der Knoten aus der Zeile $i-1$ seiner Routingtabelle, also aus der Zeile, deren Präfix um ein Zeichen kürzer ist. Wichtig hierbei ist, dass die Anfrage nicht an einen Knoten aus der gleichen Spalte wie der ungültige Routingeintrag gestellt werden darf, da sonst die Routinganfrage kein gültiges Ergebnis liefern kann. Durch die Anfrage an Knoten aus der Zeile $i-1$ wird eine deutlich größere Zahl an Knoten befragt. Kommt auch hier noch kein Ergebnis zustande, wird weiter mit der Zeile $i-2$ verfahren. Auf diese Weise wird zwangsweise ein Knoten gefunden, der den ausgefallenen Knoten im Routing von k ersetzen kann.

1.2.4 Viceroy

Viceroy ist in Berkeley und Israel im Jahr 2002 entwickelt worden [MNR02] und ist auf den ersten Blick sehr stark an Chord angelehnt. Ebenso wie dort werden bei Viceroy Schlüssel und Knoten in den selben Adressraum abgebildet. Der Adressraum hat eine Ringstruktur, die Knoten werden ebenfalls auf diesen Ring abgebildet. Jeder Knoten hat somit genau einen Vorgänger und einen Nachfolger und speichert jeweils einen Verweis auf sie. Bis hierher kann alles für Chord gesagte auch auf Viceroy übertragen werden. Viceroy kombiniert allerdings die von Chord bekannte Ringstruktur mit der Struktur eines sog. Butterfly-Networks[3].

Der Unterschied besteht nun darin, wie Viceroy das Routing vornimmt. Hierzu werden fünf weitere Verweise auf andere Knoten zusätzlich gespeichert. Ziel ist es, sowohl Verweise auf weit entfernte[4] als auch auf nahe gelegene Knoten zu erhalten, um das Routing besser als mit linearem Aufwand abwickeln zu können. Dabei werden $\log N$ Level gebildet, wobei N wieder die Anzahl der insgesamt im System befindlichen Knoten ist. Jeder Knoten wählt einen Level, wobei die Wahl so gestaltet wird, dass alle Level in etwa die gleiche Anzahl an Knoten erhalten. Nun wird ein Knoten k auf dem Level l betrachtet. Zunächst trägt k zwei Verweise auf zwei Knoten im Level $l+1$ ein; der eine Verweis (*long-range contact*) zeigt auf einen Knoten, der ungefähr den Abstand[5] $\frac{1}{2^l}$ zu k hat, der andere Verweis (*close-by contact*) zeigt auf einen Knoten, der möglichst nahe an k liegt. Zusätzlich werden noch zwei *level-ring links* eingetragen. Dies sind Vorgänger und Nachfolger auf einem Ring, der nur Knoten vom Level l enthält. Der letzte Verweis (*up-link*) erfolgt nun auf einen Knoten im Level $l-1$, der wiederum nahe an k liegt. Bei allen Knoten des Levels 1 erfolgen keine Eintragungen für den up-link.

Das eigentliche Routing erfolgt in drei Schritten. Im ersten Schritt wird die Anfrage über die up-links zu einem Knoten des nächsthöheren Levels $l-1$ weitergeleitet.

Im zweiten Schritt, wenn ein Verringern des Levels nicht mehr sinnvoll erscheint, werden die long-range und close-by contacts verwendet. Es wird genau dann der

[3] Was dieses genau ist, wird z.B. in [Sie79] beschrieben.
[4] In diesem Abschnitt ist immer der Abstand im virtuellen Adressraum gemeint.
[5] Zur Vereinfachung liegen die numerischen Werte des Adressraums hier in $[0, 1)$.

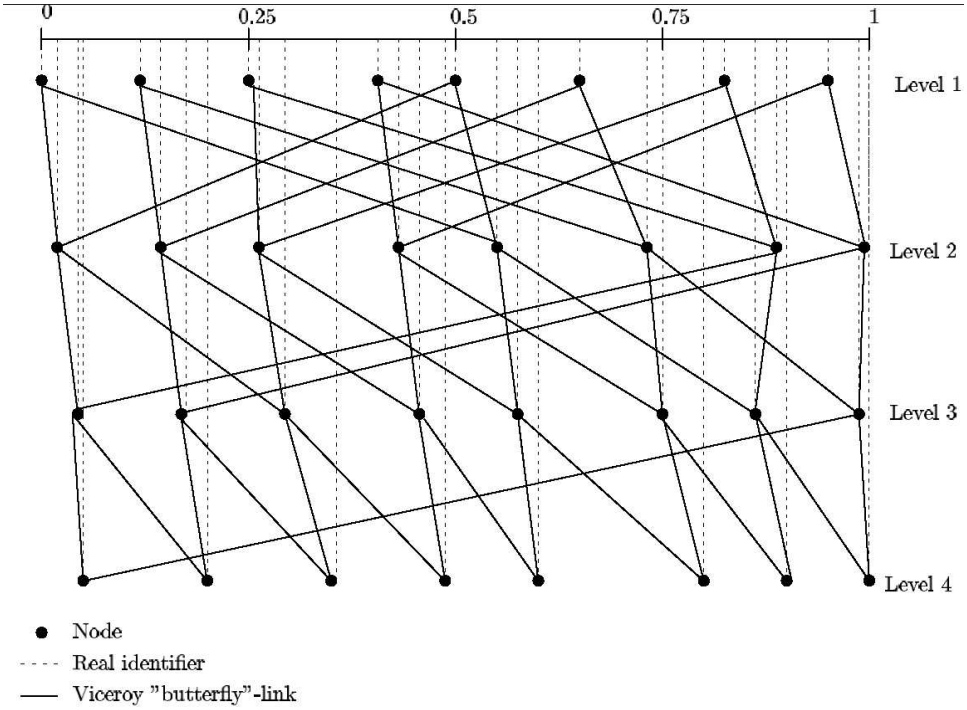

Abbildung 1.3: Darstellung eines idealen Viceroy-Netzwerkes. Dargestellt sind nur die long-range- und close-by-Verweise.

long-range contact verwendet, wenn das gesuchte Ziel weiter als $\frac{1}{2^l}$ vom aktuellen Knoten entfernt ist, andernfalls wird der close-by contact angesprochen. Schritt zwei wiederholt sich solange, bis kein Verweis auf einen höheren Level existiert. Nun kann man davon ausgehen, dass man sich relativer Nähe zum gesuchten Ziel befindet. Somit wird nun der letzte Schritt ausgeführt, in dem über die level-ring- und normalen Ringverweise der tatsächlich zuständige Knoten gefunden wird.

Nachdem das Routing in einem Viceroy-Netz beschrieben wurde, soll geklärt werden, wie das Betreten des Netzes sowie das spätere Verlassen abgewickelt wird. Zunächst muss der neue Knoten k eine ID aus dem Adressraum bekommen. Dabei müssen die IDs so gewählt werden, dass sie gleichmäßig über den Adressraum verteilt liegen. Mit Hilfe des Routings wird der Nachfolger s von k auf dem Ring ausfindig gemacht. Der bisherige Vorgänger p von s ist nun der Vorgänger von k, s ist Nachfolger von k, k wird Nachfolger von p und Vorgänger von s. Nun ist k in den Ring integriert und kann von s alle diejenigen Schlüssel übernehmen, für die er nun zuständig ist.

Jetzt muss k aber noch in das Butterfly-Netz eingebunden werden. Dazu wählt k seinen Level und sucht über die Vorgänger- und Nachfolger-Verknüpfungen auf dem Ring die jeweils nächsten Knoten, die auf dem gleichen Level liegen wie k. Nun kann sich k in den *level-ring* integrieren, indem analog vorgegangen wird, wie es schon für das Einfügen in den Ring beschrieben wurde. Damit sind zwei der fünf zusätzlichen Routingeinträge gefüllt. Um den *close-by contact* zu finden, geht k auf dem Ring in aufsteigender Richtung solange weiter, bis er einen Knoten findet, dessen Level um eins höher ist als der Level von k. Für den *long-range contact* muss k zu seiner ID $\frac{1}{2^l}$ addieren, wobei l der Level von k ist, und denjenigen Knoten finden, der für diesen Wert zuständig ist. Von diesem Knoten aus wird dann wieder auf dem Ring in aufsteigender Richtung der erste Knoten mit dem Level $l + 1$ gesucht. Dieser ist dann der *long-range contact* von k. Der *up-link* wird schließlich dadurch

gefunden, dass von k ausgehend auf dem Ring in aufsteigender Richtung der erste Knoten, dessen Level um eins niedriger ist, gesucht wird. Es müssen also neben k nur noch höchstens vier weitere Knoten ihre Routinginformationen ändern, nämlich Vorgänger und Nachfolger auf dem normalen Ring und Vorgänger und Nachfolger auf dem *level-ring*.

Soll ein Knoten das Netz verlassen, so muss er seine gespeicherten Schlüssel an seinen Nachfolger auf dem Ring übergeben. Anschließend kann der Knoten das Netz verlassen. Vorgänger und Nachfolger auf dem Ring und dem level-ring sollten nun ihre jeweiligen Vorgänger- und Nachfolger-Verweise aufeinander zeigen lassen. Stößt im weiteren Verlauf eine Anfrage über einen Verweis auf diesen nicht existenten Knoten, so wird über die lookup-Funktion der Nachfolger des fehlenden Knotens ermittelt und der fehlerhafte Eintrag korrigiert. Dies geschieht ohne gesonderte Wartungsoperation im laufenden Betrieb.

1.2.5 Kademlia

Kademlia [MM02] wurde im Jahre 2002 an der New York University entwickelt. Ebenso wie die vier vorhergeangenen DHTs verwendet auch Kademlia einen gemeinsamen, virtuellen Adressraum für Knoten und Schlüssel. Adressen haben 160 Bit und sind typischerweise über einen Hash wie SHA-1 erzeugt worden. `(key, value)`-Tupel werden auf demjenigen Knoten gespeichert, dessen ID der ID von `key` am nächsten kommt und k weiteren Knoten, um eine gewisse Redundanz zu erhalten.

Ein zentraler Bestandteil von Kademlia ist die XOR-Metrik, die verwendet wird, um den Abstand zweier Punkte im Adressraum zu bestimmen. Der Abstand zwischen zwei Punkten p_1 und p_2 ist definiert durch $d(p_1, p_2) = p_1$ XOR p_2. Es lässt sich zeigen, dass die bitweise exklusive-oder Verknüpfung tatsächlich alle Eigenschaften einer Metrik erfüllt: Wegen x XOR $x = 0$; x XOR $y > 0$, für $x \neq y$ ist XOR positiv definit, aus x XOR $y = y$ XOR x $\forall x, y$ ergibt sich Symmetrie und aus (x XOR y) XOR (y XOR z) = x XOR z und $a + b \geq a$ XOR b folgt die Dreiecksungleichung (x XOR y) + (y XOR z) $\geq x$ XOR z.

In Kademlia speichert jeder Knoten Informationen über andere Knoten im Verbund. Konkret heißt das, dass ein Knoten für jedes $0 \leq i < 160$ eine Liste anlegt, in der er bis zu k Knoten, die von ihm aus einen Abstand zwischen 2^i und 2^{i+1} haben, speichert. Ein solche Liste wird als *k-Bucket* bezeichnet und ist sortiert nach der Zeit, die vergangen ist, seit der entsprechende Knoten zum letzten Mal gesehen wurde; die am längsten nicht mehr gesehenen Knoten bilden dabei den Kopf der Liste. Jede Liste kann maximal k Elemente speichern, allerdings wird dieser Wert für kleine i nicht erreicht. Der Parameter k ist ein Replikationswert, der auf allen Knoten gleich ist und so gewählt wird, dass die Wahrscheinlichkeit, dass k Knoten innerhalb einer Stunde ausfallen, äußerst gering ist. Ein Richtwert für k ist 20. k gibt an, auf wievielen Knoten ein Schlüssel tatsächlich gespeichert wird.

Diese k-Buckets können als Blätter eines Binärbaumes angesehen werden. Die Knoten in einem k-Bucket haben alle ein gemeinsames Präfix, dass darüber Auskunft gibt, an welcher Stelle im Baum der jeweilge k-Bucket zu finden ist. Dies bedeutet, dass die k-Buckets jeweils einen bestimmten Teil des Adressraumes abdecken; zusammen decken alle k-Buckets den gesamten Adressraum ab, ohne dass es zu Überschneidungen kommt.

Empfängt ein Kademlia-Knoten eine beliebige Nachricht von einem anderen Knoten, so aktualisiert er den zugehörigen k-Bucket. Steht der andere Knoten schon in dieser Liste, so wird der Eintrag nur ans Ende verschoben. Existiert er hingegen noch nicht, so wird er dann ans Ende der Liste angehängt, wenn der k-Bucket noch weniger als k Einträge hat. Ist dies nicht der Fall, so wird geprüft, ob der erste Knoten in der Liste noch reagiert; falls er dies tut, so wird sein Eintrag ans

Ende geschoben und kein Eintrag für den neu entdeckten Knoten angelegt. Kommt keine Antwort, so wird dieser Knoten entfernt und der neu hinzugekommene an das Listenende angehängt.

Kademlia unterscheidet zwischen der Suche nach einem Knoten und der Suche nach einem Schlüssel. Zunächst soll die Suche nach einem Knoten erläutert werden. Wird eine Anfrage nach einem Knoten mit einer bestimmten ID gestellt, so wird auf diese Anfrage hin eine Liste der k am nächsten zu der gesuchten ID gelegenen Knoten zurückgegeben. Der Inhalt dieser Liste stammt entweder nur aus einem k-Bucket oder aber aus mehreren, wenn der k-Bucket mit dem geringsten Abstand[6] zur ID weniger als k Elemente enthält. In jedem Fall muss derjenige Knoten, der eine solche Knoten-Suche empfangen hat, den Inhalt seines k-Buckets, der zum gesuchten Schlüssel gehört, zurückliefern; sollten tatsächlich nur weniger als k Einträge auf diesem Knoten im entsprechenden k-Bucket vorhanden sein, so wird die Antwort mit Knoten aus Nachbarbuckets aufgefüllt, bis k Verweise vorhanden sind. Im Detail läuft dieser lookup wie folgt ab: Der Initiator einer Anfrage wählt i Knoten aus demjenigen k-Bucket aus, der den geringsten Abstand zur gesuchten ID hat. Sind in dem gewählten k-Bucket weniger als i Einträge, so werden aus den nächstnäheren k-Buckets soviele Knoten entnommen, bis i erreicht ist. Nun werden i Anfragen parallel an die ausgewählten Knoten gesendet. Der Parameter i ist ein Systemparameter, der angibt, an wieviele Knoten gleichzeitig Anfragen gesendet werden sollen; ein typischer Wert ist 3. Nun beginnt der Initiator erneut i Anfragen zu senden, diesmal aber an Knoten, die im ersten Durchgang nicht berücksichtigt wurden. Antwortet ein Knoten nicht schnell genung, wird er vom weiteren Verfahren ausgeschlossen, bis wieder eine Nachricht von ihm eintrifft. Durch diese Anfragen lernt der Initiator weitere Knoten kennen und sortiert diese in seine k-Buckets ein. Wird während dieser Anfragen kein Knoten mehr zurückgegeben, so sendet der Initiator seine Knoten-Suche erneut an all diejenigen der k nächsten Knoten, die noch keine Anfrage erhalten haben. Sind nun Antworten von allen k Knoten eingetroffen, so ist die Knoten-Suche beendet und der Initiator kennt nun die k nächsten Knoten zur gesuchten ID [7].

Die Schlüsselsuche läuft zunächst genauso ab wie die Knotensuche. Hat allerdings einer der bei der Knotensuche angefragten Knoten tatsächlich den gesuchten Schlüssel gespeichert, so gibt er nicht die k nächsten Knoten zurück, sondern den Wert des gesuchten Schlüssels. Sobald ein Wert für den gesuchten Schlüssel zurückgegeben wurde, bricht die Suche ab. Stellt der suchende Knoten fest, dass einer der k zum gesuchten Schlüssel nächsten Knoten den Schlüssel nicht besessen hat, so sendet er den Schlüssel zusammen mit dem dafür gefundenem Wert an jenen Knoten zur Speicherung.

Um einen bestimmtes `(key, value)`-Tupel zu speichern, wird zunächst eine Knotensuche durchgeführt und an die daraus erhaltenen k am nächsten zum Schlüssel gelegenen Knoten das Tupel zur Speicherung gesendet. Um zu verhindern, dass der DHT mit veralteten Einträgen überfüllt wird, muss derjenige, der das Tupel in den DHT eingebracht hat, in bestimmten Intervallen dieses neu speichern lassen. Zusätzlich senden auch die Knoten, die den Schlüssel gespeichert haben, diesen zur Speicherung an den DHT. Damit werden eventuell ausfallende Knoten ausgeglichen und sichergestellt, dass ein bestimmter Schlüssel mit sehr hoher Wahrscheinlichkeit tatsächlich auf k Knoten verfügbar ist.

Soll ein Knoten h dem Netzwerk hinzugefügt werden, so muss er einen bereits

[6] Mit dem Abstand eines k-Bucket zu einer ID ist die Differenz zwischen dem Abstand des akuellen Knotens zu den Elementen des k-Bucket und dem Abstand des aktuellen Knotens zur gesuchten ID gemeint. Dieser Abstand ist nicht als exakt zu sehen, sondern als „Abstandsklasse“, die sich über ein gemeinsames Präfix des Abstandes definiert.

[7] Bei der Knotensuche werden zwar insgesamt k^2 Verweise zurückgeliefert, aber nur k auch tatsächlich gespeichert.

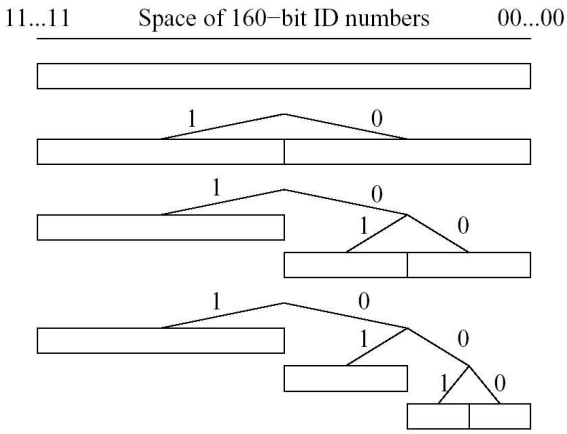

Abbildung 1.4: Aufteilung der k-Buckets im Laufe der Zeit. Es wird immer nur derjenige Bucket geteilt, in dem auch die ID des zugehörigen Knotens liegt.

ins Netzwerk eingegliederten Knoten e kennen und sich eine ID generiert haben. Der Knoten h fügt nun den Knoten e in seine Routingtabelle ein. Hierfür fügt h e in den passenden k-Bucket ein. Da h aber zu diesem Zeitpunkt nur einen einzigen Knoten e kennt, hat h auch nur einen einzigen k-Bucket. Der neue Knoten h führt nun eine Knotensuche nach seiner eigenen ID durch, hierdurch wird h den anderen Knoten bekannt gemacht und die k-Buckets von h füllen sich. Soll ein Knoten bei h in einen der k-Buckets eingetragen werden, so wird dieses getan, solange der entsprechende k-Bucket noch nicht voll ist. Ist dies jedoch der Fall und die Knoten-ID von h liegt in dem Adressbereich, für den auch der ausgewählte k-Bucket zuständig ist, so wird der k-Bucket in zwei k-Buckets geteilt, wobei beide Buckets nun ein um eine Stelle längeres Präfix haben. Liegt ein neuer Verweis jedoch nicht im gleichen Adressbereich wie h, so wird der neue Verweis verworfen. Dadurch ist eines der Präfixe auch Präfix von h, das andere jedoch nicht. Abbildung 1.4 illustriert dies.

Dass die k-Buckets tatsächlich für ein funktionierendes Routing ausreichen, wird anhand der Abbildung 1.5 deutlich. Für das Routing ist ja entscheident, dass jeder k-Bucket mindestens einen Knoten enthält. Durch das sukzessive Teilen der k-Buckets ist dieses aber auch gewährleistet.

1.3 Vergleich und Bewertung

In diesem Kapitel sollen die zuvor vorgestellten DHTs miteinander verglichen werden und es soll eine qualitative Bewertung der einzelnen DHTs versucht werden.

Die wichtigste Aufgabe, die ein DHT erfüllen muss, ist das Auffinden eines bestimmten Schlüssels. Grundsätzlich ist die benötigte Zeit für eine Suche nach einem bestimmten Schlüssel abhängig von der Anzahl der Knoten n. Von CAN abgesehen, liegt die Anzahl der „durchlaufenen“ Knoten für eine Suche immer in $O(\log n)$. Für CAN ergibt sich bei dieser Betrachtung das Problem, dass neben der Anzahl der Knoten im Netz noch eine weitere Größe, nämlich die Anzahl d der Dimensionen als veränderliche Größe hinzukommt. Der Aufwand bei CAN wird mit $d\sqrt[d]{n}$ angegeben. Wählt man aber d so, dass $d = \frac{1}{2}\mathrm{ld}\ n$ gilt, erreicht auch CAN einen logarithmischen Aufwand beim Auffinden von Schlüsseln.

Ebenso interessant ist es, zu betrachten, wie stark die Belastung durch Suchen mit zunehmender Anzahl der Knoten im Netz ansteigt, also wie gut der DHT skaliert. Hier ist festzustellen, dass die Belastung eines Knotens durch Routing bei allen

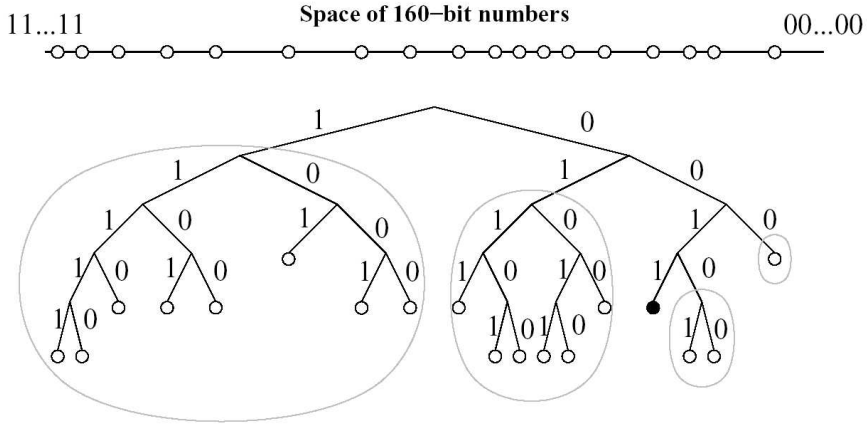

Abbildung 1.5: Binärbaum bei Kademlia. Der schwarze Knoten hat die Adresse 0011... Die grauen Bereiche stellen die k-Buckets des Knotens dar. Deutlich wird hier auch, dass die k-Buckets tatsächlich den gesamten Adressraum überscheidungsfrei füllen.

DHTs logarithmisch ansteigt; bei CAN ist wieder zu beachten, dass die Anzahl der verwendeten Dimensionen für die erwartete Anzahl von Knoten gut gewählt sein muss.

Es soll nun die Größe der Routingtabellen auf jedem einzelnen Knoten verglichen werden, um zu prüfen, ob auch das Routing annehmbar skaliert. Bei Chord und Pastry liegt die Größe der Routingtabelle in $O(m)$, wobei m die Länge der Adressen in Bit angibt. Damit liegt bei diesen DHTs eine quasi-konstante Größe der Routingtabelle vor, ebenso wie bei Kademlia, wo auf jedem Knoten $m * k$ Verweise gespeichert werden. Viceroy hat eine echt konstante Größe der Routingtabelle, die bei 7 Einträgen liegt. CAN nimmt auch hier wieder eine Sonderrolle ein; die Anzahl der Routingeinträge ist einerseits von d abhängig und andererseits auch davon, wie sehr die Zonengrößen ausgewogen sind. Da CAN aber durch Neuorganisation seiner Zonen die Zonengröße verhältnismäßig ausgewogen hält, liegt die Anzahl der Routingeinträge in $O(d)$ und hat damit ebenfalls eine relativ konstante Größe der Routingtabelle.

Bezüglich des Verhaltens im Falle eines Ausfalls sind bis auf CAN alle DHTs flexibel, insbesondere Kademlia leidet wegen der parallelen Anfragen bei in einem solchen Fall auch nicht an vergrößerter Latenz beim Auffinden von Schlüsseln. Das Thema der Redundanz ist bei Chord und Kademlia schon im ursprünglichen Design berücksichtigt worde, aber für alle vorgestellten DHTs gibt es Mechanismen, die für die redundante Speicherung von Schlüsseln sorgen.

Die DHTs nutzen alle einen virtuellen Adressraum, der kaum oder gar nicht mit dem zugrundeliegendem Netz korreliert. Dies bedeutet, dass im ungünstigsten Fall zwei virtuell benachbarte Knoten netzwerktechnisch sehr weit auseinander liegen können. Der einzige DHT, der diesem Aspekt Rechnung trägt, ist Pastry, das, wenn es die Wahl zwischen mehreren Knoten hat, denjenigen bevorzugt, der netzwerktechnisch am nächsten liegt.

Interessant ist bei der Betrachtung aller DHTs, dass, obwohl teilweise sehr verschiedene geometrische Strukturen zugrunde liegen, die Leistungsdaten doch in der gleichen Größenordnung liegen. Dies bedeutet, dass die Wahl eines geeigneten DHT für ein konkretes Ziel nicht so sehr von den theoretischen Leistungsdaten abhängen kann, sondern bestimmte, individuelle Aspekte eines DHT über seine Eignung entscheiden müssen. So wäre CAN durchaus gut geeignet für Netze, in denen mit einer

geringen Knotenfluktuation zu rechnen ist und ihre Zahl verhältnismäßig konstant bleibt, aber absolut ungeeignet für sehr dynamische Netze, da es sehr leicht zu unausgewogenen Zonen kommen kann, die bei der Suche zu zusätzlichen Schritten führen. Im letzteren Fall erscheinen Kademlia und Viceroy deutlich besser geeignet, da sich die Routingtabellen dort im laufenden Betrieb anpassen. Hat man es hingegen mit Netzen zu tun, die zwar eine große Knotenzahl haben, aber deren Knoten sehr unterschiedliche Leistungsdaten bezüglich ihrer Bandbreite und Speicherkapazität aufweisen, dann dürfte Pastry, dass ja über eine entsprechend zu definierende Metrik diese Aspekte berücksichtigen kann (genügend Auswahl an Knoten vorrausgesetzt), in die engere Wahl kommen. Aber auch Kademlia erscheint hier sehr attraktiv, da es bereits im Basisdesign für ein Caching von Schlüsseln sorgt. Tatsächlich wird Kademlia als einiziger der vorgestellten DHTs als Option in der P2P-Software eMule angeboten.

Ist hingegen ein einfaches, robustes Design gefragt, dass keinen besonderen Eigenschaften genügen muss, ist Chord der am besten geeignete DHT.

Möglicherweise ließe sich aber auch durch eine beeinflusste Wahl der Knoten-IDs das Problem der unterschiedlichen Leistungsfähigkeiten bei den Knoten lösen. So wäre es denkbar, dass man jedem Knoten einen Leistungswert zuordnet und dann die IDs so vergibt, dass sich eine ausgewogene Leistungsdichte über den virtuellen Adressraum ergibt, also schwächere Knoten immer kleinere Zonen haben als leistungsstarke Knoten. Selbstverständlich kann man dann keine Hash-Funktionen wie SHA nutzen, die diesem Bestreben entgegenstehen.

Abschließend bleibt also festzustellen, dass es bei den ausgewählten DHTs keine schlechten Designs gibt, jedoch bessere und schlechtere Einsatzgebiete. Letztendlich erscheinen zwei DHTs besonders interessant: Chord wegen seines einfachen Designs und einer dennoch erstaunlich hohen Fehlertoleranz und Kademlia, dass nun als erster der vorgestellten DHTs auf sehr breiter und globaler Basis in eMule verwendet werden kann. Die ersten Eindrücke hiervon sind sehr vielversprechend.

Literaturverzeichnis

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[MNR02] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly, 2002.

[RD01] 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, November 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[Sie79] H. J. Siegel. Interconnection networks for simd machines. *Computer*, 12(6):57–65, 1979.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Francs Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

Kapitel 2

Overlay-Anwendungen: Chord und CFS

Seminarbeitrag von **Kirill Müller**

Die meisten Peer-to-Peer-Anwendungen, unabhängig vom Einsatzgebiet, benötigen eine Antwort auf folgende Frage: Wie kann effizient der Rechnerknoten gefunden werden, der ein bestimmtes Datum speichert? Dieser Beitrag widmet sich den verschiedenen Aspekten der Auffindung von Schlüsseln.

Im ersten Teil wird ein verteiltes dezentrales Protokoll mit dem Namen *Chord* [SMK+01] vorgestellt. Das Protokoll stellt eine injektive Abbildung von Schlüsseln auf Rechnerknoten her. Chord definiert ein Overlay-Netzwerk, um strukturierte Peer-to-Peer-Kommunikation zu ermöglichen. Robustheit und Skalierbarkeit kann wegen der Einfachheit des Protokolls sogar bewiesen werden.

Im zweiten Teil wird das *Cooperative File System* [DKK+01] vorgestellt. CFS ist ein Nur-Lese-Dateisystem, bei dem die Daten auf mehrere Knoten verteilt werden. Als Schlüssel-Auffindungs-Protokoll wird Chord benutzt. Durch die beweisbaren Eigenschaften von Chord kann auch für CFS Robustheit und Skalierbarkeit garantiert werden. Weitere Eigenschaften sind Effizienz und Lastausgleich, die sogar bei einer großen Anzahl von Einzelausfällen bewahrt werden.

2.1 Einleitung

Das Wort „Overlay“ bedeutet soviel wie „Überlagerung“. Gemeint ist, daß in einer Netzstruktur wie dem Internet die Kommunikation zwischen den Knoten nicht beliebig erfolgt. Vielmehr wird das Netzwerk nur benutzt, um virtuelle Netzwerke herzustellen, die einem selbstdefinierten Muster folgen. Dies ist erforderlich, wenn die Topologie des zugrundeliegenden Netzwerks nicht die Kommunikationsbedürfnisse der Anwendung wiederspiegelt.

Overlay-Netzwerke sind also keine neuartige Technologie, sondern vielmehr ergeben sie sich aus dem Anwendungsfall. Prominente Beispiele sind DNS und IP-Tunnelling. Durch eine geschickte Wahl der Netztopologie können Dienstmerkmale wie Robustheit, Redundanz und Bandbreite zur Verfügung gestellt werden, selbst wenn das zugrundeliegende Netzwerk diese nicht in ausreichendem Umfang bietet.

Trotz der Vielfalt der bislang entwickelten Peer-to-Peer-Systeme haben viele dieser Systeme eine Gemeinsamkeit: Die Kernoperation ist das Auffinden von Daten. Mit Chord wurde ein verteiltes Protokoll entworfen, das ein effizientes Auffinden von Schlüsseln in einem sich dynamisch ändernden Peer-to-Peer-System ermöglicht.

Das englische Wort „Chord“ kann in diesem Kontext mit „Kreissehne“ übersetzt werden. Das hat etwas mit der Topologie des Overlay-Netzwerks zu tun: Chord benutzt einen chordalen Ring.

Das Chord-Protokoll unterstützt lediglich eine einzige Operation: Finde zu einem gegebenen Schlüssel den ihm zugeordneten Rechnerknoten. Je nach Anwendung kann dieser Rechnerknoten die Speicherung eines dem Schlüssel zugeordneten Wertes übernehmen. Als Basis für die Bereitstellung dieser Funktionalität genügen herkömmliche Netzwerk-Protokolle wie UDP.

Chord benutzt *konsistentes Hashing* [KLL+97], um jedem Knoten einen Schlüssel zuzuordnen. Die Benutzung von konsistentem Hashing führt automatisch zu einem Lastausgleich, da jeder Knoten in etwa dieselbe Anzahl Schlüssel erhält. Ferner ist die Zuordnung zwischen Rechnerknoten und Schlüsseln relativ statisch, so daß bei Veränderungen der Netzstruktur nur wenige Schlüssel zu einem anderen Knoten umziehen müssen.

Frühere Arbeiten zu konsistentem Hashing gingen davon aus, daß jeder Knoten Kenntnis über die meisten anderen Knoten im System hat. Diese Vorgehensweise funktioniert allerdings nur für kleinere Netzwerke und skaliert nicht gut mit wachsender Knotenanzahl. Im Gegensatz dazu braucht Chord für effizientes Routing nur $O(\log N)$ andere Knoten zu kennen.

Die Knoten müssen miteinander kommunizieren, um eine Schlüsselsuche zu ermöglichen. Für jede Anfrage werden $O(\log N)$ Nachrichten ausgetauscht. Verändert sich das Netzwerk, werden die für das Routing notwendigen Datenstrukturen aktualisiert.

Ist die Routing-Information teilweise veraltet, sinkt die Performance nur sanft. Dies ist insofern wichtig, da in der Praxis Knoten beliebig zum Netzwerk hinzukommen und dieses auch wieder verlassen. Es wird sogar für nur $O(\log N)$ Knoten nicht immer möglich sein, stets vollständig korrekte Information aufrechtzuerhalten.

Es kann bewiesen werden, daß der Chord-Algorithmus korrekt ist. Im Unterschied zu anderen Verfahren lassen sich hier sogar qualitative Aussagen über die Leistungsfähigkeit bei sich verändernden Netzwerken beweisen.

Im verteilten Dateisystem CFS kommt das Chord-Protokoll zum Einsatz. Erstmals konnte so ein System entwickelt werden, das folgenden Anforderungen gerecht wird:

- Symmetrie, Dezentralisierung
- Zuverlässigkeit auch bei unbeaufsichtigten freiwilligen Teilnehmern
- Schnelle Auffindung der gesuchten Daten
- Robustheit und Effizienz auch bei häufig hinzukommenden und ausfallenden Knoten
- Lastausgleich über alle Knoten auch bei Dateien verschiedener Größe

Im Gegensatz dazu leisten bisherige Entwürfe immer nur einen Teil der oben aufgeführten Anforderungen.

CFS besteht aus drei Schichten. Chord bildet dabei die unterste Schicht. Performance und Fehlertoleranz wird durch eine Mittelschicht erreicht. Die oberste Schicht interpretiert die von der Mittelschicht auf Anfrage bereitgestellten Rohdaten-Blöcke und stellt auf Anwendungsseite eine konventionelle Dateisystem-Schnittstelle zum lesenden Zugriff bereit.

Der Rest des Beitrags ist wie folgt gegliedert. Im Abschnitt 2.2 werden verwandte Arbeiten vorgestellt. In den Abschnitten 2.3 und 2.4 wird das Chord-Protokoll unter statischen und dynamischen Aspekten erläutert. Das verteilte Dateisystem CFS wird in Abschnitt 2.5 vorgestellt. Eine Zusammenfassung ist in Abschnitt 2.6 zu finden.

2.2 Verwandte Arbeiten

Das Chord-Projekt [Cho] befindet sich noch in Entwicklung. Auf der Webseite sind unter anderem die Quellen und ein häufig aktualisierter technischer Bericht [SMK+02] verfügbar. Dieser Bericht ist im Wesentlichen eine Erweiterung der Originalarbeit [SMK+01].

Ein detaillierter Vergleich mit verwandten Arbeiten ist in [SMK+01] zu finden. Daraus geht unter anderem hervor, daß Tapestry [ZKJ01], Pastry [RD01] (siehe auch folgender Beitrag) und CAN [RFH+01] am ehesten mit Chord verglichen werden können. Der entscheidende Vorteil von Chord ist jedoch seine Einfachheit und die daraus resultierende Robustheit auch bei störungsanfälligen Netzwerken und Knoten.

Als Hauptnachteil von Chord gegenüber anderen Protokollen muß erwähnt werden, daß die Nachbarschaft von Knoten im Overlay in keiner Weise mit der physischen Nachbarschaft korreliert. Dadurch kann es passieren, daß eine Suchanfrage ein Vielfaches der physikalischen Entfernung zwischen den Knoten zurücklegen muß. Das wird teilweise durch die geringe durchschnittliche Anzahl notwendiger Hops ausgeglichen. Die Einbringung von Information über die zugrundeliegende Netzwerk-Topologie beim Aufbau des Overlay-Netzwerks [CDHR02] ist ein entscheidender Vorteil von Pastry gegenüber Chord.

Eine Suche nach Schlüsselwörtern wie beispielsweise in Gnutella kann durch Aufsetzen einer verteilten Suchmaschine wie FASD [Kro02] implementiert werden. Die Chord FAQ [Cho] gibt an, daß ein naiver Ansatz nicht gut zu funktionieren scheint.

Der Bericht von Androutsellis-Theotokis [AT02] stellt einen schönen Überblick über die existierenden Overlay-Technologien dar.

Unter den neuesten Systemen ist insbesondere YAPPERS [GSG03] hervorzuheben. Hier wurde versucht, einen Hybridansatz zwischen lose (Gnutella) und eng (DHT-basiert) gekoppelten Systemen herzustellen.

Chord wurde von seinen Entwicklern als Unterbau für das verteilte Dateisystem CFS benutzt. Inzwischen ist Chord von der Forschungsgemeinde als leistungsfähige Basis zum Aufbau von Overlay-Netzwerken anerkannt. Meist wird es in einem Atemzug mit CAN, Pastry und Tapestry erwähnt. Einige Systeme wie ConChord [ACMR02], Web Service Discovery [SP04] und INS/Twine [BBK02] benutzen bereits das Chord-Protokoll. Das Resource Management Framework [SR03] stellt sogar eine eigene Implementation bereit.

CFS übernimmt verschiedene Entwurfs-Ideen des Nur-Lese-Dateisystems SFS-RO [FKM02]. Bei CFS ist jedoch neu, daß dynamisch der Aufenthaltsort des gewünschten Datums gefunden wird.

Volltextsuche und Anonymität bietet CFS nicht. Beides kann jedoch als darüberliegende Schicht implementiert werden. Änderungen von Dateien können nur vom Besitzer vorgenommen werden, und nur durch Austausch der kompletten Datei.

Gnutella [Gnu] leistet Volltextsuche, ist aber bei weitem nicht so performant wie CFS. OceanStore [KBC+00] bietet Anonymität und Änderbarkeit, jedoch zum Preis einer wesentlich komplexeren Systemstruktur.

2.3 Das Chord-Protokoll: Statik

2.3.1 Bezeichner

Ein *Bezeichner* stellt eine m-Bit-Zahl dar. Jedem Knoten und jedem Schlüssel wird ein Bezeichner zugeordnet. Dadurch werden Knoten und Daten auf einen gemeinsamen Datentyp abgebildet.

Die Abbildung wird durch eine Hash-Funktion realisiert. Für Knoten wird der Bezeichner aus IP-Adresse und Port-Nummer erzeugt, bei den Schlüsseln wird der gesamte Schlüssel gehasht. Sind Daten zu einem Schlüssel zugeordnet, kann auch der Hash-Wert der Daten als Schlüssel verwendet werden.

Zur Berechnung der Zuordnung wird in der Praxis SHA-1 [Nat95] mit $m := 160$ verwendet. Die Funktion hat die schöne Eigenschaft, daß bei beliebig verteilten Argumenten die Funktionsergebnisse mit höchster Wahrscheinlichkeit gleichverteilt sind. Das senkt für diesen Anwendungsfall die Wahrscheinlichkeit einer Kollision auf das theoretisch mögliche Minimum von 2^{-m}. Für $m = 160$ ist das vernachlässigbar. Ferner ist es praktisch unmöglich, zu einem gegebenen Funktionswert $sha(x)$ ein inverses y mit $sha(x) = sha(y)$ und $x \neq y$ zu finden.

Prinzipiell sind jedoch auch andere Hash-Funktionen und andere Werte für m denkbar.

2.3.2 Der chordale Ring

Die Bezeichner bilden einen zyklischen Ring modulo 2^m. Die Zuordnung von Schlüsseln zu Rechnerknoten geschieht, indem der Knoten mit dem nächsthöheren Hash-Wert gewählt wird. Ein solcher Knoten $nf(k)$ heißt *Nachfolger* des Schlüssels k. Sind die Knoten und Datenwerte bezüglich der Hash-Funktion in einem Kreis angeordnet, ist der Nachfolger eines Datenwerts der nächste Knoten im Uhrzeigersinn.

Im folgenden werden zur Kennzeichnung von Knoten und Schlüsseln ausschließlich deren Bezeichner verwendet.

Für ein Netzwerk mit $m = 6$ und einer Knotenmenge $N = \{ 1, 8, 14, 21, 32, 38, 42, 48, 51, 56 \}$ wäre $nf(10) = 14$ und $nf(54) = 56$, siehe auch Abbildung 2.1. Ein neuer Schlüssel mit 58 wird zum Knoten $nf(58) = 1$ zugeordnet. (Das Beispiel und die dazugehörigen Abbildungen wurden aus [SMK+01] entnommen.)

2.3.3 Beweisbare Schranken

In [KLL+97] wird folgender Satz bewiesen:

Für N beliebige Knoten und K beliebige Schlüssel gilt mit höchster Wahrscheinlichkeit:

1. Jeder Knoten ist höchstens für $(1 + \epsilon) \cdot K/N$ Schlüssel verantwortlich.
2. Bei einem neu hinzukommenden oder das Netzwerk verlassenden Knoten n ändert sich die Zuordnung für höchstens $O(K/N)$ Schlüssel. Es sind nur Schlüssel betroffen, die zu n zugeordnet werden oder wurden.

Bei der oben angegebenen Implementation des konsistenten Hashings läßt sich eine Obergrenze von $\epsilon = O(\log N)$ beweisen. In der Originalarbeit [KLL+97] wird vorgeschlagen, auf einen physikalischen Knoten eine Anzahl v *virtueller Knoten*

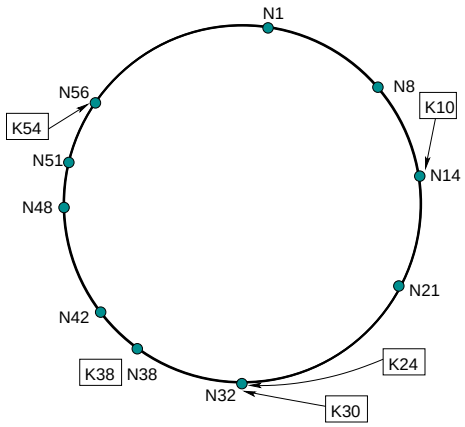

Abbildung 2.1: Ein Beispiel-Netzwerk mit 10 Knoten und 5 Schlüsseln

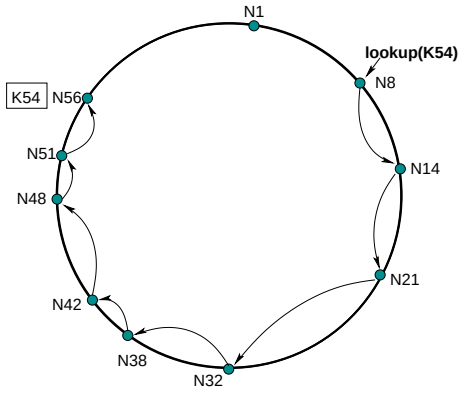

Abbildung 2.2: Primitive Suche nach einem Schlüssel in einem Chord-Netzwerk

aufzusetzen. Mit einer solchen Strategie kann N beliebig erhöht werden. Da der Faktor $(1 + \epsilon)/N$ mit steigendem N fällt, kann durch künstliche Erhöhung der Knotenanzahl bei konstantem K ein geringerer Wert für die Schranke $(1 + \epsilon)/N$ und somit eine bessere Gleichverteilung erreicht werden.

Beim Einsatz mehrerer virtueller Knoten auf einem physikalischen Knoten werden kurzzeitige Belastungsspitzen einzelner virtueller Knoten besser abgefangen. Der Ausfall eines physikalischen Knotens hat jedoch stets den gleichzeitigen Ausfall mehrerer virtueller Knoten zur Folge.

2.3.4 Auffinden eines Schlüssels – naiv

Der einfache Algorithmus für das Auffinden eines Schlüssel k im Ring ist ganz einfach: Durchlaufe alle Knoten in der Reihenfolge der Bezeichner, bis der Schlüssel bei einem dieser Knoten gefunden wird. Gilt für einen Knoten n die Ungleichung $nf^{-1}(n) < nf(k) < n$, kann das Verfahren abgebrochen werden. (Die Suche ist über das Ziel hinausgeschossen.)

Natürlich muß hier beachtet werden, daß die Knoten selbst keine Datenstrukturen, sondern aktive Objekte sind. Insofern würde man den Durchlauf so implementieren, daß jeder Knoten seinen Nachfolger fragt, falls er selbst nicht Eigentümer des Schlüssels ist. Bemerkenswert ist, daß alle Knoten gleichberechtigt sind und denselben Algorithmus ausführen.

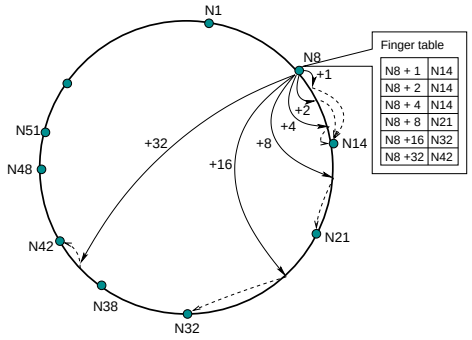

Abbildung 2.3: Eine Finger-Tabelle

Abbildung 2.2 zeigt die Suche nach Schlüssel 54 ausgehend von Knoten 8. Falls sich der Schlüssel 54 nicht auf dem Knoten 56 befindet, kann die Suche bei Knoten 56 abgebrochen werden.

Unter der Voraussetzung, daß jeder Knoten seinen Nachfolger kennt, funktioniert dieses Verfahren immer. Diese Garantie erkauft man sich mit einer hohen durchschnittlichen Anzahl Hops von $O(N)$.

2.3.5 Die Finger-Tabelle

Ein Aufwand von $O(N)$ Hops pro Schlüssel-Suche ist nicht tragbar. Um die Suche zu beschleunigen, wird bei Chord zusätzliche Routing-Information verwaltet.

Jeder Knoten n speichert eine Routing-Tabelle mit bis zu m Einträgen, die *Finger-Tabelle*. Der i-te Eintrag $n.finger[i]$ mit $0 \leq i < m$ enthält den Knoten, dessen Entfernung auf dem chordalen Ring mindestens 2^i beträgt. Es gilt also $n.finger[i] = nf(n+2^i)$. Man beachte, daß $n.finger[0]$ stets der direkte Nachfolger von n im Ring ist.

Abbildung 2.3 zeigt die Finger-Tabelle für Knoten 8. Der 5. Finger ($i = 4$) ist Knoten $nf(8 + 2^4) = nf(24) = 32$.

Jeder Eintrag in der Finger-Tabelle besteht aus dem Knoten-Bezeichner und der IP-Adresse mit Port-Nummer.

Jeder Knoten speichert nur Information über wenige Knoten. Dabei ist das Wissen über (im Sinn der Ring-Topologie) naheliegende Knoten umfangreicher als über weiter entfernte. Diese lückenhafte Information insbesondere im Hinblick auf weit entfernte Knoten genügt im Normalfall nicht, um eine Suchanfrage direkt zum richtigen Knoten weiterzuleiten. In unserem Beispiel kann Knoten 8 nicht selbst den Speicherort für Schlüssel 34 ermitteln, da $nf(34) = 38$ nicht in der Finger-Tabelle auftritt.

2.3.6 Auffinden eines Schlüssels – skalierbar

Im folgenden wird der Algorithmus skizziert, der das Auffinden eines Schlüssels in $O(\log N)$ Hops ermöglicht. Es ist im wesentlichen eine Erweiterung des primitiven Algorithmus: Auch hier wird der Ring durchlaufen, und jeder betroffene Knoten führt denselben Algorithmus aus. Jedoch ermöglicht es die Finger-Tabelle, abzukürzen.

Die Grundidee ist folgende: Wenn ein Finger auf einen Knoten zeigt, der kleiner als der Suchschlüssel ist, brauchen die Knoten vor diesem Finger nicht einzeln durchgegangen werden – sie sind ebenfalls kleiner als der Suchschlüssel und können somit nicht dessen Nachfolger sein. Die Suche kann also bei einem Finger-Knoten

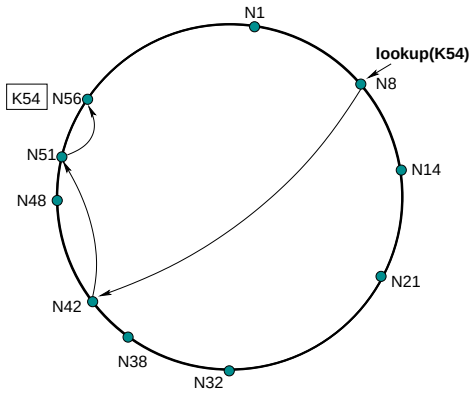

Abbildung 2.4: Effiziente Suche nach einem Schlüssel in einem Chord-Netzwerk

beginnen, der echt kleiner als der Suchschlüssel ist. Am effizientesten wird die Suche, wenn der größte Finger gewählt wird, der gerade noch kleiner als der Suchschlüssel ist: Dann wird die größtmögliche Anzahl Knoten übersprungen.

Abbildung 2.4 zeigt den Weg, den die Suchanfrage nach Schlüssel 54 ausgehend von Knoten 8 einschlagen wird. Knoten 8 wählt den größten Finger kleiner 54, das ist in diesem Fall der 6. Finger ($i = 5$), der auf Knoten 42 zeigt. Der 4. Finger von Knoten 42 zeigt auf Knoten 51, der 5. Finger zeigt bereits auf Knoten 1, also wird als nächstes der 4. Finger-Knoten kontaktiert. Mit einem weiteren Hop ist der korrekte Knoten gefunden.

Es gilt eine obere Schranke von $O(\log N)$ für die Anzahl Hops beim oben angegebenen Algorithmus. Intuitiv ist das klar: In jedem Schritt wird die Distanz zum Schlüssel im Erwartungswert um mindestens die Hälfte verkürzt, und die Maximaldistanz kann nicht größer als N sein. Den relativ technischen, allerdings nicht schweren Beweis findet man in der Originalarbeit zu Chord [SMK+01].

Im Rahmen dieser Arbeit wurden auch Simulationen durchgeführt, bei denen eine mittlere Anzahl Hops von $\frac{1}{2} \log N$ beobachtet wurde. Bei einer etwas genaueren Formulierung der intuitiven Argumentation aus dem letzen Absatz werden diese Ergebnisse verständlich: In jedem Schritt wird die Distanz zum Schlüssel um mindestens die Hälfte, im Mittel jedoch um drei Viertel verkürzt.

2.4 Das Chord-Protokoll: Dynamik

2.4.1 Zuordnung von Schlüsseln zu Knoten

Bei neu hinzukommenden oder das Netzwerk verlassenden Knoten ändert sich die Zuordnung von Schlüsseln zu Rechnerknoten. Die verwendete Zuordnungsstrategie sorgt jedoch dafür, daß die Änderungen minimal und nur lokaler Natur sind. Würde in unserem Beispiel ein neuer Knoten 26 dem Netzwerk beitreten, würde lediglich der Schlüssel 24 von Knoten 32 zum neuen Knoten 26 umziehen. Bei Ausfall von Knoten 14 ändert sich nur für Knoten 21 etwas: Er bekommt den Schlüssel 10 zugeordnet.

Eine veränderte Zuordnung hat zur Folge, daß die Schlüssel zu ihren neuen Knoten umziehen müssen. Das kann besonders aufwendig werden, wenn an den Schlüsseln Nutzdaten hängen.

2.4.2 Ankunft neuer Knoten

Bisher wurde beim Suchalgorithmus nur der statische Fall betrachtet. Sollen neue Knoten hinzukommen, muß der Ring erweitert werden, und eventuell müssen Schlüssel migriert werden. Hier soll nur ein grober Überblick über die Vorgehensweise gegeben werden, der interessierte Leser sei auf [SMK+01] verwiesen.

Um die Erweiterung des Rings um neue Knoten zu unterstützen, speichert jeder Knoten neben dem Nachfolger-Knoten und der Finger-Tabelle auch den Vorgänger-Knoten. Das Erweitern des Rings geschieht dann im wesentlichen wie das Hinzufügen eines neuen Eintrags in eine doppelt verkettete lineare Liste.

Das Erzeugen eines neuen Chord-Rings geschieht durch das Eintreffen des ersten Knotens. In diesem Fall zeigt der Nachfolger-Verweis auf den (bisher) einzigen Knoten.

Kommt ein neuer Knoten n' dazu, benötigt dieser nur eine Referenz auf einen beliebigen Knoten n. Dieser wird befragt, welcher Knoten der Nachfolger von n' ist, entsprechend wird der Nachfolger-Verweis in n' gesetzt.

Dies sind die einzigen Aktionen, die unmittelbar nach Einteffen eines neuen Knotens durchgeführt werden. So kann jedoch kein Ring aufgebaut werden: Ein neuer Knoten zeiget zwar auf den Ring, allerdings zeigt kein Knoten aus dem Ring auf den neuen Knoten.

Um neue Knoten „einzubürgern“, werden in regelmäßigen Abständen Wartungsarbeiten am Ring durchgeführt. Dabei wird der Vorgänger des Nachfolgers geprüft und entsprechend der Nachfolger-Zeiger korrigiert. Ferner wird der Nachfolger unter Angabe des eigenen Bezeichners benachrichtigt, so daß dieser bei Bedarf seinen Vorgänger-Zeiger korrigieren kann.

Die oben dargestellten Aktionen sorgen für einen stabilen Ring. Die Konstruktion der Finger-Tabelle erfolgt ebenfalls durch regelmäßige Auffrischung: Jeder Knoten n setzt regelmäßig $n.finger[i] := nf(n + 2^i)$ für alle i. Die Berechnung von nf wird dabei mit Hilfe der oben beschriebenen effizienten Suchprozedur durchgeführt, mit der auch der Speicherort eines Schlüssels ermittelt wird.

Bemerkenswert hierbei ist, daß ein Knoten immer nur seinen Nachfolger, nie seinen Vorgänger kontaktiert: Die Information über den Vorgänger kann inkorrekt sein, zum Beispiel ist das direkt nach dem Hinzufügen eines neuen Knotens der Fall.

Der technische Bericht [SMK+02] enthält einen Beweis, daß die oben dargestellten Wartungsarbeiten am Ring bei jeder beliebigen Reihenfolge von Knotenankunft und Stabilisierung irgendwann einen stabilen Ring erzeugen. Die Stabilisierung darf also im laufenden Betrieb erfolgen, ohne die normale Arbeit des Netzwerks zu beeinträchtigen und ohne auf Synchronisation jedweder Art angewiesen zu sein.

2.4.3 Auswirkung neuer Knoten auf die Korrektheit der Suche

Da der Ring nicht sofort nach Ankunft eines neuen Knotens in einem stabilen Zustand ist, können Suchanfragen auch einen instabilen Ring antreffen.

Probleme sind beispielsweise dann zu erwarten, wenn die Nachfolger-Zeiger nicht korrekt sind oder bei Schlüssel-Migration noch nicht alle Schlüssel zum neuen Knoten kopiert wurden: Dann schlägt der Suchvorgang fehl. Die über Chord liegende Anwendungsschicht bekommt einen Fehler zurückgeliefert und hat die Möglichkeit, den Suchvorgang nach einer Pause zu wiederholen: Nachfolger-Verweise sind irgendwann korrigiert, Migrationsvorgänge sind irgendwann abgeschlossen.

Die exakte Position der Finger hat auf die Funktionalität keinen Einfluß, sie dient nur der Sicherstellung der Schranken für die Anzahl Hops bei Suchanfragen. Insofern spielt es keine Rolle, ob durch das Hinzufügen eines Knotens ein Finger eigentlich auf den neuen Knoten zeigen müßte.

2.4.4 Auswirkung neuer Knoten auf die Anzahl Hops bei der Suche

Die Schranke von $O(\log N)$ Hops wurde nur für einen stabilen Ring bewiesen. Durch veraltete Finger-Information kann die Schranke überschritten werden. Die folgende Überlegung soll darstellen, wie viele Knoten hinzugefügt werden müßten, um die Anzahl notwendiger Hops wesentlich zu erhöhen.

Zu einem stabilen Netzwerk seien Knoten hinzugefügt worden, ohne daß eine Aktualisierung der Finger erfolgte. Die Vorgänger-Verweise seien jedoch aktuell. Unter diesen Bedingungen erfolgt ein Routing zu einem neuen Knoten mit Hilfe der Finger-Tabellen über die vor dem Hinzufügen vorhandenen Knoten. Nur in den letzten Schritten wird ein linearer Scan durchgeführt, bei dem nur neue Knoten besucht werden.

Die Erhöhung der Anzahl Hops ergibt sich aus der Anzahl Hops beim linearen Scan. Diese wiederum hängt von der Anzahl der neu hinzugekommenen Knoten ab. Durch die Uniformität der Hash-Funktion kann davon ausgegangen werden, daß bei N' neu hinzugekommenen Knoten im Mittel N'/N neue Knoten zwischen zwei alten Knoten liegen. Die Länge des Scans ergibt sich somit im Mittel aus $N'/(2N)$, die Gesamtzahl Hops ist gleich $O(\log N) + N'/(2N)$.

Schon um die Anzahl Hops für die Suche um einen Faktor zu erhöhen, müßten also $O(N \log N)$ Knoten neu hinzukommen! Selbst das Hinzufügen von N Knoten kostet im Mittel nur einen einzigen Hop mehr.

Diese Überlegungen berechtigen zu folgender Aussage: Solange die Finger schneller aktualisiert werden als sich die Netzwerkgröße verdoppelt, können Suchanfragen in $O(\log N)$ Hops durchgeführt werden.

2.4.5 Ausfall bestehender Knoten

Während das Eintreffen neuer Knoten ein freiwilliger Prozeß ist, kann das Ausscheiden bestehender Knoten auf unfreiwilliger Basis geschehen: Netzwerk- oder Stromausfälle, Hardware- oder Softwarefehler sind nur einige der zahlreichen möglichen Gründe. Die Qualität eines Systems läßt sich auch daran messen, wie gut es mit solchen Umständen umgehen kann.

Die Korrektheitsaussage des Chord-Protokolls setzt voraus, daß jeder Knoten seinen Nachfolger kennt. Inkorrekte Information über Nachfolger führt zu inkorrekten Suchläufen. Wenn in unserem Beispiel die Knoten 14, 21 und 32 gleichzeitig ausfallen, weiß Knoten 8 nichts über seinen Nachfolger 38, da kein Finger darauf zeigt. Knoten 8 geht davon aus, daß sein Nachfolger Knoten 42 ist. Eine Suche nach dem Schlüssel 30 würde also 42 liefern, obwohl 38 die richtige Antwort wäre.

Eine Erhöhung der Robustheit gegenüber Knotenausfällen wird erreicht, indem eine Nachfolgerliste verwaltet wird, die die r nächsten Nachfolger enthält. Wenn der direkte Nachfolger nicht antwortet, wird einfach der nächste Nachfolger laut Liste probiert. Um den Ring aufzutrennen, müßten alle r Nachfolger-Knoten gleichzeitig ausfallen. Bei einer Einzelausfallwahrscheinlichkeit von p wäre das eine Wahrscheinlichkeit von p^r, was bereits mit moderaten r sehr unwahrscheinlich ist.

Die Verwaltung der Nachfolger-Liste benötigt nur kleine Änderungen in den Stabilisierungs-Routinen. Zusätzlich muß die Such-Routine angepaßt werden, um auf Knotenausfälle adäquat reagieren zu können.

Mit einer Nachfolger-Liste der Länge $r = \Omega(\log N)$ können erstaunliche Resultate bewiesen werden [SMK+02]: Mit höchster Wahrscheinlichkeit werden weder Funktionalität noch Performance beeinflußt, wenn die Hälfte der Knoten ausfällt! Die Rechnung geht davon aus, daß die ausfallenden Knoten zufällig verteilt sind.

Hier wird ein Vorteil davon deutlich, daß beim Aufbau eines Chord-Netzwerks die darunterliegende Netztopologie nicht auf das Overlay abgebildet wird. Selbst

wenn ein Angreifer in der Lage wäre, ein IP-Subnetz oder eine ganze geographische Region vom Chord-Netzwerk zu trennen, hätte das wahrscheinlich keinen Einfluß auf die Funktionsfähigkeit des Netzwerks.

Selbstverständlich muß das System durch geeignete Maßnahmen wie beispielsweise Replikation gegen Verlust der den Schlüsseln zugeordneten Nutzdaten geschützt werden. Dieser Aspekt soll hier jedoch nicht behandelt werden, der interessierte Leser sei auf den technischen Bericht [SMK+02] verwiesen.

Unbenommen bleibt natürlich die Möglichkeit, daß Knoten das Netzwerk geplant verlassen. Das kann wie ein Fehlschlagen behandelt werden. Die Performance von Chord verbessert sich jedoch, wenn man das Protokoll für diesen Fall erweitert. Ein sich verabschiedender Knoten kann seine Schlüssel und die verbundenen Nutzdaten zu seinem Nachfolger übertragen. Außerdem kann eine kurze Nachricht an den Vorgänger und an den Nachfolger helfen, die Ringstruktur nach Ausscheiden des Knotens schnell wiederherzustellen.

2.5 Cooperative File System

2.5.1 Ein Dateisystem basierend auf Chord?

Wie oben beschrieben, leistet Chord nur eine einzige Operation – das Auffinden des Rechnerknotens, der für einen Schlüssel verantwortlich ist. An diesem Schlüssel können weitere Nutzdaten hängen. Aber wie wird daraus ein Dateisystem, selbst wenn nur lesend zugegriffen werden soll?

Ein Dateisystem ist verantwortlich für das Verwalten von Dateien. Üblicherweise verfügen Dateisysteme über die Möglichkeit, Verzeichnisse anzulegen, die selbst wieder Verzeichnisse und Dateien enthalten können. Eine Datei:

- kann eine Größe von einigen Bytes bis zu inzwischen einigen Terabytes haben
- kann einmal im Jahr oder tausendmal pro Sekunde zugegriffen werden
- liegt an der Wurzel oder in einem beliebig tief geschachtelten Verzeichnis
- hat keine oder hunderttausend andere Dateien im selben Verzeichnis
- hat einen Besitzer und Zugriffseinschränkungen.

Auf den ersten Blick fällt es schwer, ein verteiltes System zu konstruieren, daß das alles leistet. Der naive Ansatz, den vollständig qualifizierten Dateinamen als Schlüssel zu verwenden, ist zumindest problematisch. Durch die flache Struktur der Schlüssel sind Operationen wie das Auflisten aller Dateien in einem Verzeichnis teuer, da sie das ganze Netzwerk involvieren.

Die Vielfalt der heute existierenden Dateisysteme ist riesig. Die meisten davon legen ein blockorientiertes Gerät wie z.B. eine Festplatte oder eine optische Platte zugrunde. Blöcke definieren eine Struktur über dem ansonsten nicht effizient verarbeitbaren Bitstrom. Blöcke ermöglichen es, Datenobjekte auf der Festplatte zu adressieren. Ein Block kann gleichermaßen Nutzdaten oder Metadaten speichern.

Die Wahl des Dateisystems und die Wahl des zugrundeliegenden blockorientierten Geräts sind orthogonale Entwurfsentscheidungen. Beispielsweise läßt sich ein Dateisystem genausogut auf einer Partition, auf einer ganzen Festplatte und auf einem RAID-Verbund aufsetzen.

Insofern genügt es, mit Hilfe von Chord ein verteiltes blockorientiertes Gerät zu implementieren und darauf ein passendes Dateisystem aufzusetzen. Im folgenden soll gezeigt werden, wie dieses Konzept in CFS verwirklicht wurde.

2.5.2 Systemstruktur

Jeder CFS-Dienstnehmer enthält drei Software-Schichten: Eine Dateisystem-Schicht, eine Datenhaltungs-Schicht basierend auf DHash und eine Schlüsselfindungs-Schicht basierend auf Chord. Das Dateisystem greift auf DHash zu, um Blöcke abzurufen. DHash benutzt Chord, um den aktuellen Aufenthaltsort eines Blocks aufzuspüren.

Beim CFS-Dienstgeber fehlt die Dateisystem-Schicht. Es ist ja auch nicht notwendig, daß eine Festplatte Kenntnis über die Struktur der Daten hat, die auf ihr gespeichert sind. Die Datenhaltungs-Schicht speichert die Datenblöcke und kümmert sich um Replikation und Caching. DHash und Chord arbeiten zusammen, um eine Blocksuche mit dem Auffinden gecachter Kopien zu verbinden.

Die von DHash gelieferten Blöcke werden von der Dateisystem-Schicht in einem Format interpretiert, daß ähnlich dem in SFSRO [FKM02] verwendeten ist. Das Dateisystem-Format besitzt Ähnlichkeiten zu Unix V7. Hauptunterschied ist, daß Block-Bezeichner anstelle von Block-Nummern verwendet werden. Der Block-Bezeichner wird durch einen SHA-1-Hash [Nat95] der Blockdaten generiert: Dadurch wird Unfälschbarkeit der Daten und Integrität des Dateisystem-Baums garantiert.

Ein weiterer Unterschied zu vielen konventionellen Dateisystemen ist, daß hier mehrere Dateisysteme auf einem blockorientierten Gerät Platz haben. Partitionierung ist nicht notwendig. Jedes Dateisystem wird durch seine Wurzel identifiziert, die ebenfalls einen DHash-Block darstellt. Von der Wurzel aus gelangt man durch Verweise zu den Meta- und Nutzdaten. Insofern braucht CFS auch nicht „formatiert“ zu werden: Ein System ohne Wurzel-Blöcke (insbesondere ein System ohne jedwede Art von Blöcken) ist leer.

2.5.3 Hinzufügen, Ändern und Löschen von Dateisystemen

Das Hinzufügen eines neuen Dateisystems geschieht wie folgt. Zunächst werden die Meta- und Nutzdatenblöcke dem Dateisystem hinzugefügt. Die Verwendung von SHA-1 zur Generierung der Bezeichner garantiert Kollisionsfreiheit. Erst zum Schluß wird der Wurzel-Block signiert und eingefügt. Diese Vorgehensweise stellt sicher, daß den Klienten jederzeit eine konsistente Sicht auf das Dateisystem präsentiert wird. Die Sicht auf das Dateiystem könnte jedoch unter gewissen Umständen veraltet sein.

Der Schreibschutz eines CFS-Dateisystems bezieht sich nur auf die Klienten. Jedoch kann ein Dateisystem vom Besitzer aktualisiert werden. Dazu müssen die geänderten Blöcke eingespielt werden. Bei Änderung eines Blocks ändert sich auch sein Bezeichner, dadurch müssen alle Vorgänger mit eingespielt werden.

In letzter Instanz ändert sich bei der Aktualisierung die Wurzel, diese wird wie beim Erzeugen des Dateisystems zuletzt eingespielt. Nach Einspielen der Wurzel ist die Änderung des Dateisystems abgeschlossen. CFS gestattet Aktualisierungen der Wurzel nur dann, wenn der Besitzer seine Identität anhand der Signatur der alten Wurzel nachweisen kann. Der Bezeichner der Wurzel ändert sich bei Aktualisierung nicht.

Eine Löschoperation wird nicht unterstützt. Stattdessen werden Blöcke immer nur bis zu einer zugesicherten Aufbewahrungszeit vorgehalten. Ein Besitzer kann um Aufschiebung der Aufbewahrungszeit bitten. Tut er das immer wieder, wird eine dauerhafte Datenhaltung ermöglicht. Hört er damit auf, wird das Dateisystem irgendwann gelöscht.

2.5.4 Änderungen in Chord: Auswahl des nächsten Knotens

Die Originalarbeit zu Chord [SMK+01] wählte den als nächstes zu kontaktierenden Knoten ausschließlich anhand der Finger-Liste aus. Im Rahmen der Arbeit an CFS wurde eine Heuristik eingeführt, die die Auswahl von Knoten mit einer geringen erwarteten Latenzzeit bevorzugt. Die Heuristik basiert unter anderem auf Latenzzeitmessungen im laufenden Betrieb. Der interessierte Leser sei auf die Originalarbeit [DKK+01] verwiesen.

2.5.5 Authentifizierung von Knoten

Bei einem verteilten Dateisystem stellt sich die Frage, inwieweit den Teilnehmern Vertrauen geschenkt werden kann.

Ein möglicher Angriff sieht so aus: Der Angreifer wählt einen Knoten-Bezeichner direkt nach dem Bezeichner des zu zerstörenden Datenblocks. Er erhält vom System die Verantwortung für diesen Block und verneint anschließend dessen Existenz.

Die Knoten-Bezeichner berechnen sich aus einem SHA-1-Hash der IP-Adresse und einer kleinen Zahl, die den Index des virtuellen Knotens angibt. Dadurch sinkt die Fähigkeit potentieller Angreifer, den Knotenbezeichner selbst zu wählen.

Andere Angriffe unter Vortäuschung einer falschen IP-Adresse wären denkbar. Zum Schutz dagegen wird bei einem neuen Knoten geprüft, ob unter der behaupteten IP-Adresse mit dem behaupteten Knoten-Index wirklich ein gültiger Knoten verbirgt und ob der Knoten-Bezeichner korrekt gebildet wurde.

Mit diesen Schutzmaßnahmen müßte ein Angreifer etwa so viele Knoten kontrollieren, wie es bereits Knoten im System gibt, um eine gute Chance für einen Angriff auf einen beliebigen Datenblock zu haben. Je größer das Netz wächst, desto geringer ist diese Chance.

2.5.6 DHash – die Schicht über Chord

DHash ist verantwortlich für folgende Aspekte:

- Caching
- Bereitstellung von Daten auf Verdacht
- Replikation
- Kontingentverwaltung

Ohne Zweifel sind das wichtige Eigenschaften eines verteilten Dateisystems. Aus Platzgründen wird jedoch verzichtet, darauf einzugehen, wie das alles erreicht wird. Der interessierte Leser findet in der Originalarbeit [DKK+01] eine detaillierte Beschreibung.

An dieser Stelle sei nur erwähnt, daß DHash an das darüberliegende Dateisystem nur drei Schnittstellen-Methoden bereitstellt: Block lesen, Block schreiben, Wurzel aktualisieren.

2.6 Zusammenfassung

Die Ermittlung des Aufenthaltsorts eines Datenobjekts ist eine zentrale Fragestellung bei verteilten Systemen. Chord löst dieses Problem mit einem dezentralen Ansatz in nur $O(\log N)$ Speicher pro Knoten und $O(\log N)$ Hops pro Anfrage. Seine Einfachheit, beweisbare Robustheit und beweisbare Performance sind besonders

hervorstechende Eigenschaften. Das Erweitern eines Netzwerks geschieht problemlos. Es arbeitet sogar bei veralteter Routing-Information und bei Ausfall vieler Teilnehmer immer noch korrekt.

Chord kann als Unterbau für verteilte Applikationen verschiedenster Bauart verwendet werden. Die Entwickler von Chord haben es als Basis für ein verteiltes Dateisystem genutzt, dabei stellt Chord die Funktionalität eines blockorientierten Geräts bereit. CFS erbt somit die guten Eigenschaften von Chord und erhöht Performance, Zuverlässigkeit und Verfügbarkeit mit Hilfe einer Mittelschicht, die eng mit Chord zusammenarbeitet.

Die Ansätze von Chord und CFS faszinieren mit ihrer puristischen Schönheit. Offenbar hat Chord das Potential, auch im planetarischen Maßstab eingesetzt zu werden. Es bleibt abzuwarten, inwiefern die fehlende Lokalität der Kommunikation beim ursprünglichen Entwurf von Chord durch Weiterentwicklungen ausgeglichen werden kann.

Literaturverzeichnis

[ACMR02] Sameer Ajmani, Dwaine E. Clarke, Chuang-Hue Moh, and Steven Richman. ConChord: Cooperative SDSI certificate storage and name resolution. In *Peer-to-Peer Systems, First International Workshop, IPTPS 2002*, number 2429 in Lecture Notes in Computer Science, pages 141–154, March 2002.

[AT02] Stephanos Androutsellis-Theotokis. A survey of peer-to-peer file sharing technologies. ELTRUN Athens University of Economics and Business web site, 2002. `http://www.eltrun.gr/whitepapers/`.

[BBK02] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Proceedings of Pervasive 2002*, volume 2414/2002. Springer-Verlag Heidelberg, 2002.

[CDHR02] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. Technical Report 82, Microsoft Research, 2002.

[Cho] The Chord project. `http://www.pdos.lcs.mit.edu/chord/`.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215. ACM Press, 2001.

[FKM02] Kevin Fu, M. Frans Kaashoek, and David Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Syst.*, 20(1):1–24, 2002.

[Gnu] Gnutella. `http://www.gnutella.com/`.

[GSG03] P. Ganesan, Q. Sun, and H. Garcia-Molina. YAPPERS: A peer-to-peer lookup service over arbitrary topology. In *Proceedings of IEEE INFOCOM*, 2003.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[KLL+97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM Press, 1997.

[Kro02] Amr Z. Kronfol. *FASD: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine.* PhD thesis, Princeton University, 2002.

[Nat95] National Institute of Standards and Technology. Secure hash standard, April 1995. FIPS 180-1.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[SMK+02] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Technical Report TR-819, MIT, January 2002. `http://www.pdos.lcs.mit.edu/chord/papers/`.

[SP04] Cristina Schmidt and M. Parashar. A Peer-to-Peer Approach to Web Service Discovery. *World Wide Web*, 7(2):211–229, June 2004.

[SR03] Alan Southall and Steffen Rusitschka. *The Resource Management Framework: A System for Managing Metadata in Decentralized Networks Using Peer to Peer Technology*, volume 2530/2003, pages 144–149. Springer-Verlag Heidelberg, 2003.

[ZKJ01] 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.

Kapitel 3

Pastry, Squirrel und SplitStream

Seminarbeitrag von **Martin Scheerer**

3.1 Einführung

Diese Seminararbeit beschäftigt sich mit zwei Anwendungen, Squirrel und SplitStream, die auf Pastry aufbauen. Pastry ist ein Peer-to-Peer (P2P) Framework.

P2P Systeme sind verteilte Systeme in denen alle Knoten die selben Fähigkeiten und Aufgaben haben. Die Kommunikation zwischen den einzelnen Knoten ist symmetrisch. Im Besonderen gibt es daher in reinen P2P Systemen keine Server/Client Struktur. Ausserdem sind keine zentralen Kontroll und Steuereinheiten nötig.

Oft erzeugen P2P Systeme ein Overlaynetzwerk. Charakteristisch für solche Netzwerke ist, dass sie auf einer Menge andere Netzwerke definiert werden. Dadurch entsteht eine Schichtung von virtuellen Netzwerken. Verbindungen zwischen zwei Knoten in einem Overlaynetzwerk werden durch ein ganzes Segment im darunterliegenden Netzwerk realisiert.

Zum besseren Verständniss von Squirrel, ein verteilter Webcache, und SplitStream, ein baumbasiertes Multicast System, wird zuerst Pastry beschrieben.

3.2 Pastry

3.2.1 Einführung und Einordnung

Bei Pastry handelt sich es um ein Framework für große (Größenordnung 100.000 Knoten) P2P Anwendungen. Pastry übernimmt das Routing und die Lokalisierung von Objekten. Das Overlaynetzwerk, das von Pastry dafür erzeugt wird, ist für

heterogene Netzwerke wie das Internet ausgelegt. Pastry ermöglicht eine komplett dezentralisierte Anwendung. Somit gibt es keine Hierarchie oder Spezialisierung der Knoten. Die Objekte werden strukturiert abgelegt. D.h. für jedes Objekt ist eindeutig bestimmt an welcher Stelle im Netzwerk es abgelegt ist.

Eingebaute Mechanismen der Fehlerbehandlung und Selbstorganisation für den Ausfall einzelner Knoten führen dazu, dass Pastry ein Resilient Overlay Network bildet.

3.2.2 Design

Knoten

Jedem Knoten in einem Pastrynetzwerk ist eine 128-bit ID zugeordnet. Diese IDs werden zufällig zugeteilt. Möglichkeiten zur Bestimmung der KnotenID sind zum Beispiel ein Hash der IP oder des öffentlichen Schlüssels. Wichtig ist das die KnotenIDs gleichverteilt sind. Man kann sich den Raum der Knoten als zirkulären Raum vorstellen.

Aufgrund der zufälligen Verteilung der KnotenIDs ist es wahrscheinlich, dass Nachbarn im physikalischen Netz keine Nachbarn im Overlaynetzwerk sind. Das Overlaynetzwerk orientiert sich also in seinem Aufbau nicht am Aufbau des darunterliegenden physikalischen Netzes.

Wir nehmen an ein Netzwerk besteht aus N Knoten. Jetzt ist der Schlüssel eines Objekts gegeben. Die Schlüssel für Objekte stammen auch aus dem Raum der KnotenIDs. Natürlich gibt es daher nicht für jedes Objekt einen Knoten. Normalerweise wird der Knoten mit der numerisch nächste KnotenID zum Objektschlüssel gesucht. Das Routing erfolgt in mehreren Schritten, die immer näher zum Zielknoten heranführen. Man kann Aussagen über die maximale Anzahl solcher Schritte treffen. Pastry braucht, um einen Objekschlüssel von einem beliebigen anderen Knoten aus zu finden, maximal $log(N)$ solcher Schritte.

Bis zu einer Anzahl von $|L|/2$ ausgefallenen Knoten mit benachbarter KnotenID kann das Erreichen des gewünschten Knotens garantiert werden. L ist die Blattmenge. Auf die Blattmenge wird weiter unten näher eingegangen.

Knoten Zustandsmenge

Jeder Knoten hat eine Routingtabelle, eine Nachbarschaftsmenge und eine Blattmenge. Dieser Abschnitt beschreibt den Inhalt dieser Zustandsmengen. Hierfür werden die KnotenIDs und Objektschlüssel als eine Sequenz von Ziffern zur Basis 2^b betrachtet. b ist ein Konfigurationsparameter von Pastry.

Routingtabelle Jeder Eintrag der Routingtabelle besteht aus einer KnotenID und der dazugehörigen IP. Die n-te Zeile enthält nur KontenIDs die mit der lokalen KnotenID n Ziffern gemeinsam haben. Die $n+1$. Ziffer ist immer ungleich der $n+1$. Ziffer der lokalen KnotenID.

Da es im Allgemeinen mehrere Knoten gibt, die diese Bedingung erfüllen, ist das nicht eindeutig. Es wird jedoch ein bestimmter Knoten ausgewählt, nämlich einer, der im physikalischen Netzwerk möglichst nahe ist. In Abschnitt 3.2.5 gibt es mehr zu den Lokalitätseigenschaften dieser Auswahl.

Wenn überhauptkein Knoten mit dieser Bedingung bekannt ist, dann wird dieser Eintrag leer gelassen.

Nachbarschaftsmenge Hier werden KnotenIDs und IPs von nahen Knoten (bzgl. der Entfernungsmetrik) gespeichert. Im normalen Routingprozess werden diese Ein-

träge nicht benutzt. Diese Menge wird für die Verwaltung der Entfernungsmetrik gebraucht. Die Anzahl der Einträge ist normalerweise zwischen 2^b und $2 \cdot 2^b$.

Blattmenge Die Blattmenge besteht zur einen Hälfte aus numerisch nächsten Knoten mit IDs die kleiner sind als die des lokalen Knotens, zur anderen Hälfte aus den IDs, die größer sind. Hier wird also die Nachbarschaft im Overlaynetzwerk abgebildet, im Gegensatz zur Nachbarschaftsmenge, die sich ja auf die Nachbarschaft im physikalischen Netz bezieht. Die Größe dieser Menge ist vergleichbar mit der Nachbarschaftsmenge.

Routing

Zu Beginn wird geprüft ob der Schlüssel innerhalb der Blattmenge liegt. D.h. numerisch relativ nahe an der eigenen KnotenID. Dann wird die Anfrage an den numerisch nächsten Knoten weitergeleitet.

Wenn der Schlüssel nicht innerhalb der Blattmenge liegt so wird die Routingtabelle herangezogen.

Wenn dieser Eintrag in der Routingtabelle leer ist, so wird die Nachricht an einen Knoten weitergeleitet, der ein mindestens genauso langes gemeinsames Prefix wie der lokale Knoten hat, zusätzlich aber noch numerisch näher am Schlüssel liegt.

Das hier beschriebene Vorgehen konvergiert immer, da in jedem Schritt die Nachricht dem Zielknoten näher kommt.

3.2.3 Pastry API

Für Applikationen, die auf Pastry aufbauen, werden folgende Methoden angeboten:

- **nodeID = pastryInit(Credentials, Application)** meldet den lokalen Knoten in einem Pastrynetzwerk an. Der erste Parameter dient zur Authentifizierung des Knotens im Netzwerk. Der zweite ist eine Referenz auf die Applikation um ein Callbackmuster für einige der folgenden Methoden zu realisieren.
- **route(msg, key)** leitet die Nachricht zu dem Knoten mit der numerisch nächsten KnotenID zum Schlüssel weiter.

Um von Pastry bei bestimmten Ereignissen benachrichtigt zu werden, muss die Applikation folgende Methoden implementieren:

- **deliver(msg, key)** wird von Pastry auf dem Knoten aufgerufen, der die numerisch nächste KnotenID zum Schlüssel hat.
- **forward(msg, key, nextID)** Über diese Methode hat die Applikation vor dem Weiterleiten einer Nachricht durch Pastry noch die Möglichkeit korrigierend einzugreifen. Zum Beispiel kann die nextID auf NULL gesetzt werden, dadurch wird die Nachricht nicht mehr weitergeleitet.
- **newLeafs(leafSet)** wird von Pastry bei jeder Änderung an der Blattmenge aufgerufen um der Applikation die Möglichkeit zu geben darauf zu reagieren.

3.2.4 Selbstorganisation und Anpassung

Beim Hinzufügen, Löschen oder beim Ausfall eines Knotens versucht Pastry die Auswirkungen auf den laufenden Betrieb gering zu halten. Dies wird durch das automatische Reparieren des Netzwerks beim Ausscheiden eines Knotens erreicht. Neue Knoten werden in das Netzwerk integriert. All diese Fälle werden lokal behandelt, d.h. nur Knoten innerhalb einer Umgebung müssen bei solchen Änderungen benachrichtigt werden. Die Protokolle hierzu werden in [RD01] erläutert.

3.2.5 Lokalität

Bisher wurde gezeigt wie Pastry Nachrichten versendet und weiterleitet. Man konnte den Aufwand dafür in Anzahl der Hops angeben. Was dabei nicht beachtet wurde ist die tatsächliche Dauer einer Zustellung. In einem Overlaynetzwerk können zwei Knoten zwar direkt nebeneinander liegen (Raum der KnotenIDs) und trotzdem nur sehr langsam miteinander verbunden sein, da sie in ganz verschiedenen physikalischen Netzen existieren. Daher ist es für Pastry wichtig auch beim Weiterleiten in seinem Overlaynetzwerk diese Lokalitätseigenschaften mit zu berücksichtigen. Da es meist nicht eindeutig ist an welchen Knoten eine Nachricht weitergeleitet werden muss, kann es also unter Berücksichtigung einer Entfernungsmetrik schlechte und gute Wege zum Zielknoten geben.

Pastry benutz als Maß für die Entfernungen im Netzwerk eine skalare Metrik, wie zum Beispiel die Anzahl der IP-Routing Hops oder die geographische Entfernung. Die Applikation muss eine Funktion anbieten, die für eine gegebene IP-Adresse eine Entfernung zurückgibt. Ein Knoten mit einer kleineren Entfernung wird bervorzugt. Die Applikation ist verantortlich mit Caching und ähnlichen Mechanismen dafür zu sorgen, dass die Entfernungsmessung keinen zu großen Overhead erzeugt. Hier wird davon ausgegangen, dass die gewählte Metrik euklidisch ist, d.h. die Dreiecksungleichung gilt für alle Distanzen zwischen Pastryknoten.

In der Praxis ist dies schon für IP-Routing Hops nicht gegeben. Dies führt aber höchstens zu einer Verschlechterung der Lokalitätseigenschaften von Pastry.

3.2.6 Effiziensbetrachtung

Pastry skaliert mit der Anzahl der Knoten. D.h. auch bei vielen Knoten bleibt die Anzahl der Routingschritte pro Anfrage bei $log(N)$.

Wenn man die tatsächlich zurückgelegte Distanz mit der bei einer globalen Routingtabelle optimalen Distanz vergleicht, verlängert sich der durchschnittliche Weg um ungefähr 30% bis 40%. Dies ist der Preis, der für den dezentralen Ansatz zu zahlen ist.

Der Durchsatz an Routingnachrichten für eine unoptimierte Java Implementierung beträgt 3000 Nachrichten pro Sekunde.

3.3 Squirrel

3.3.1 Einführung

Webcaches sind eine weit verbreitete Technik um die Benutzungs des Webs schneller und billiger zu machen. Dies wird durch das Zwischenspeichern von aus dem Web abgefragten Dokumenten erreicht. Ein Nebeneffekt davon ist die Entlastung der Webserver, da Objekte jetzt auch von diesen Zwischenspeichern geliefert werden können.

Webcaches sind daher oft an der Grenze von Unternehmensnetzen ins Internet oder bei Internet Service Providern installiert. Squirrel bietet hier eine Alternative an, die alle Clients zu einem großen Webcache zusammenschaltet und somit die zentralen Webcaches überflüssig werden.

Idee

Die grundsätzliche Idee von Squirrel ist die Nutzung der lokalen Caches von jedem Browser. Bisher hält jeder Browser die Objekte, die er abgerufen hat, in seinem lokalen Cache um sie schnell wieder anzeigen zu können. Möchte ein anderer Knoten aus dem selben Netz auf die selben Objekte zugreifen, so geht diese Anfrage an den

zentralen Webcache, der diese Objekte hoffentlich noch enthält. Von den lokalen Caches der übrigen Knoten wird kein Gebrauch gemacht.

Dies gewinnt an Bedeutung, wenn man betrachtet welchen Aufwand Unternehmen für die Anschaffung, Wartung und Verfügbarkeit eines zentralen Webcaches betreiben. Dieser zentrale Webcache würde wegfallen, wenn man einen mindestens äquivalenten Webcache auf Basis der lokalen Caches realisieren könnte. Dieser dezentrale Cache hat gegenüber einem zentralen Webcache noch weitere Vorteile. Zum einen fällt ein *single point of failure* weg, zum anderen skaliert der dezentrale Webcache mit der Anzahl der Knoten, da mit der Anzahl der Knoten auch die Anzahl der lokalen Caches zunimmt.

Squirrel nutzt das selbstorganisierende P2P-Framework Pastry, dass in 3.2 besprochen wurde. Pastry findet die Routen zu Knoten, die das gewünschte Objekt beinhalten. Squirrel ist durch die Benutzung von Pastry auch bei Ausfall einiger Knoten noch benutzbar, man muss nur den Inhalt der lokalen Caches der ausgefallenen Knoten nocheinmal vom ursprünglichen Server holen. Die Herausforderung ist es diese Vorteile zu nutzen und mit diesem verteilten Ansatz eine Leistung, die mit einem zentralen Webcache vergleichbar ist, zu erreichen. Man muss hier vom Benutzer wahrgenommene Latenz, Trefferverhältnis (*hit ratio*) und externe Bandbreitenbenutzung betrachten. Weitere interresante Themen sind die Mehrbelastung der Knoten und des lokalen Netzwerks.

Hintergrund

Browser generieren HTTP GET Anfragen für Internetobjekte wie HTML Seiten, Bilder und andere Inhalte. Diese Anfragen werden entweder vom lokalen Cache, vom Webcache oder vom ursprünglichen Webserver beantwortet. Wer die Anfrage beantwortet, hängt davon ab wo eine aktuelle Version des Objekts liegt, und ob das Objekt überhaupt geeignet ist in einem Cache gespeichert zu werden. Für Objekte die dynamisch, also erst zum Anfragezeitpunkt erzeugt werden, trifft dies zum Beispiel nicht zu. Solche Objekte werden also in jedem Fall weiterhin vom ursprünglichen Webserver geliefert.

Ist das Objekt im Cache, so wird überprüft ob es noch aktuell ist. Ist es aktuell wird es an den Browser gesendet. Ist es nicht aktuell wird eine bedingte GET (cGET) Anfrage gestellt. Diese cGET Anfrage kann von einem Cache auf dem Weg zum ursprünglichen Server beantwortet werden, oder vom Ursprungsserver selbst. Die Antwort enthält entweder das Objekt oder eine *not-modified* Nachricht, wenn das Objekt nicht verändert wurde.

3.3.2 Design

Zuerst werden die Voraussetzungen für den Ansatz von Squirrel behandelt. Im nächsten Abschnitt wird gezeigt wie die Browser und Squirrel zusammenarbeiten. Danach werden zwei verschiedene Ansätze für das Vorgehen von Squirrel besprochen.

Voraussetzungen

Die Zielumgebung von Squirrel sind Netzwerke von 100 bis 100.000 Knoten (meistens Desktoprechner) in üblichen Unternehmensnetzen. Es wird davon ausgegangen, dass jeder dieser Knoten Zugang zum Internet hat, entweder direkt oder durch eine Firewall. Weitere Bedingung ist das alle Knoten geographisch in einer Region liegen. Daraus folgt das Kommunikation zwischen den Knoten schnell und billig ist, also um mindestens eine Größenordnung besser als die externe Anbindung.

Jeder teilnehmende Knoten führt eine Instanz von Squirrel aus. Alle diese Instanzen haben die selbe Funktion zur Bestimmung der Aktualität von Objekten.

Aufbau

Um Squirrel mit einem üblichen Browser benutzen zu können, wird etwas von der Idee abgewichen. Es wird nicht der lokale Cache der Browser genutzt, dieser wird deaktiviert, bzw. auf null gesetzt. Dafür wird in jedem Browser die lokale Squirrel Instanz als Proxyserver eingestellt. Somit gehen alle Anfragen des Browsers an die lokale Squirellinstanz und können dort weiterbearbeitet werden. Der lokale Cache wird jetzt also von Squirrel und nicht vom Browser verwaltet. Dies macht es auch möglich die selbe Squirrelinstanz für verschiedene Browser zu nutzen.

Ob es nicht vielleicht doch sinnvoll ist dem Browser etwas lokalen Cache zu lassen, damit er eventuelle Optimierungen durchführen kann, die zum Beispiel beim häufigen Benutzen der Zurück-Funktion nützlich sein könnten, sollte Gegenstand weiterer Untersuchungen sein.

Von diesem Aufbau ausgehend werden zwei verschiedene Algorithmen zur Verwaltung der Objekte entwickelt.

Algorithmen

Die hier vorgeschlagenen Algorithmen sollen den Raum der Möglichkeiten etwas verständlicher machen. Daher wird eine sehr einfache Variante vorgestellt. Sie wird als *Home-Store* bezeichnet. Die zweite Variante ist komplexer und wird als *Directory* bezeichnet.

Beiden gemeinsam ist, dass in der lokalen Squirrel Instanz der Schlüssel für ein gewünschtes Objekt berechnet wird. Man benutzt hierfür einen SHA-1 Hash der URL. Entweder ist das Objekt lokal vorhanden und gültig, dann wird es sofort an den Browser gesendet. Oder es müssen andere Knoten kontaktiert werden. Auch hier ist der erste Schritt noch beidesmal der selbe. Man benutzt das Routing von Pastry um den Knoten im Netz zu finden, der dem Objektschlüssel am nächsten ist. Dies ist der *Home* Knoten. Ab hier unterscheiden sich die beiden vorgestellten Verfahren.

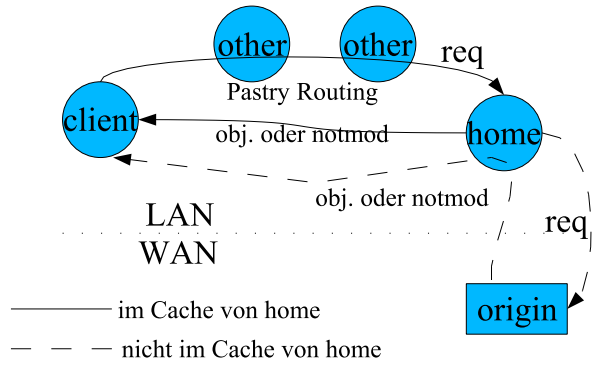

Abbildung 3.1: *Home-Store* Ansatz

Home-Store In diesem Fall speichert Squirrel das Objekt sowohl im Clientcache als auch auf dem *Home* Knoten. Das Protokoll in Abbildung 3.1 läuft folgenderma-

ßen ab: Wenn der Client keine gültige Kopie des Objekts hat, so hat er entweder eine veraltete Kopie, oder das Objekt ist nicht im lokalen Cache. Dementsprechend sendet er eine cGET oder ein GET Anfrage an den *Home* Knoten. Wenn der *Home* Knoten eine gültige Kopie besitzt, wird diese an den anfragenden Client zurückgesendet und der Vorgang ist beendet. Entsprechendes gilt für den Fall, dass das Objekt nicht verändert wurde.

Findet der *Home* Knoten dagegen eine veraltete Kopie vor, oder hat das Objekt nicht im Cache, kontaktiert er den ursprünglichen Server mit einer cGET bzw. GET Anfrage. Antwortet der Server wird der Cache des *Home* Knotens aktuallisiert, und diese Antwort an den Client zurückgesendet.

Alle externen Anfragen müssen also durch den *Home* Knoten geleitet werden. So kann man davon ausgehen, dass der *Home* Knoten immer die aktuellste Version des Squirrelnetzwerks besitzt. Somit ist keine weitere Suche nach diesem Objekt nötig.

Directory Dieser Ansatz basiert auf der Idee, dass derjenige Knoten der kürzlich auf ein Objekt zugegriffen hat, weitere Knoten mit diesem Objekt beliefern kann, da es noch im lokalen Cache liegt. Dafür pflegt der *Home* Knoten für jedes Objekt ein kleines Verzeichnis mit Verweisen auf Knoten die als letztes auf diese Objekt zugegriffen haben. Vorgeschlagen werden vier Verweise pro Objekt. Die Grundidee ist es bei Anfragen aus diesen Verweisen zufällig einen Knoten auszuwählen. Ein solcher Knoten wird *Delegate* genannt. Das Protokoll sorgt dafür, dass alle *Delegates* die selbe Version des Objekts enthalten.

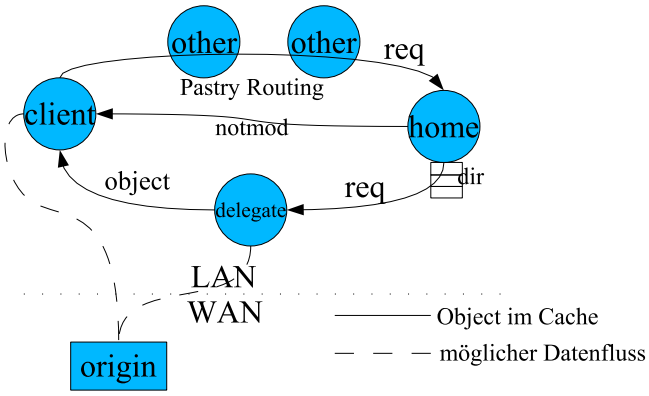

Abbildung 3.2: *Directory* Ansatz

Dieses Protokoll ist in Abbildung 3.2 beschrieben. Jeder Knoten unterhält ein Verzeichnis für Objekte die dort ihren *Home* Knoten haben. In diesem Verzeichnis werden auch Metadaten wie der Zeitpunkt der letzten Änderung und explizite Informationen wie *time-to-live* und Cachekontroll-Informationen gehalten. Daher kann der *Home* Knoten cGET Anfragen bearbeiten ohne das Objekt selbst gespeichert zu haben. Die Verwaltung der Gültigkeit eines Objekts findet also auf dem *Home* Knoten statt.

Wie beim *Home-Store* Ansatz wird die HTTP Anfrage zuerst zum *Home* Knoten geleitet. Hat der *Home* Knoten dieses Objekt vorher noch nicht gesehen, so besitzt er auch kein Verzeichnis für dieses Objekt. Dies teilt er dem Client durch eine kurze Nachricht mit. Daraufhin führt der Client die Anfrage zum Ursprungsserver aus.

Der *Home* Knoten behandelt den Client als möglichen *Delegate* und schreibt ihn in das neu erstellte Verzeichnis. Die Metadaten bleiben uninitialisiert. Normalerweise erhält der Client bald eine Antwort vom Ursprungsserver und sendet dem *Home* Knoten eine Nachricht, die alle nötigen Metadaten enthält. Ist das Objekt nicht zum Cachen geeignet, wird der *Home* Knoten aufgefordert das ganze Verzeichnis zu löschen.

Im zweiten Fall ist das Objekt schon bekannt, d.h. es existieren Einträge im Verzeichnis. Handelt es sich um eine cGET Anfrage, kann diese sofort beantwortet werden. Andernfalls wird die Anfrage an einen zufällig ausgewählten *Delegate* weitergeleitet und der Client wird zum Verzeichnis hinzugenommen.

Weitere Fälle, wie veraltete Verzeichniseinträge werden in [SID02] näher erläutert.

Knoten Ankunft, Abschied und Ausfall

Beide oben vorgestellten Verfahren müssen mit häufigen Veränderungen bei den teilnehmenden Knoten rechnen. Neben den von Pastry ausgeführten Routinen für diese Fälle muss Squirrel folgende, zusätzliche Operationen ausführen.

Wenn ein Knoten neu hinzukommt, so wird er automatisch *Home* Knoten für eine Anzahl von Objekten. Aber er hält noch keine Objekte oder Verzeichnisse für die er zuständig ist, d.h. bei einer Anfrage werden unnötige Kontakte zum Ursprungsserver aufgebaut. Das Squirrelnetzwerk funktioniert aber weiterhin. Um den Verlust an Leistung zu vermeiden, der durch unnötige externe Kontakte auftritt, nutzt Squirrel eine Möglichkeit der Pastry API, nämlich über Änderungen in der Blattmenge von Pastry informiert zu werden. Dies tritt immer bei den Nachbarn, die von einer Veränderung der Knoten betroffen sind, auf.

Im *Home-Store* Ansatz werden die Objekte auf den neuen *Home* Knoten übertragen. Ein Vorteil hat hier das *Directory* Modell, hier müssen nur die Verzeichnisinformationen übertragen werde, was im Allgemeinen weniger ist als das Objekt selbst. Man kann auch Kompromisse eingehen, so dass zum Beispiel nur häufig besuchte Objekte umkopiert werden. Geht man von einer Zipf-Verteilung aus, hat man immer noch ein sehr gutes Caching Verhalten.

Knoten, die einfach ausfallen, benötigen keine Nachbearbeitung von Squirrel, die Objekte für die sie zuständig waren, werden eben vom jetzt zuständigen Knoten neu angefordert. Wenn im *Directory* Modell ein *Delegate* ausfällt, so stellt dies der *Home* Knoten schon bei der Kontakaufnahme fest, und streicht ihn aus dem Verzeichnis.

Eine weitere Möglichkeit stellt das ordnungsgemäße Beenden eines Knotens dar. Hier kann der Knoten noch Objekte oder Verzeichniseinträge auf seine Nachbarn verteilen, so dass keine Information verloren geht.

3.3.3 Messergebnisse

In diesem Abschnitt werden einige Messungen vorgestellt, die zum einen die Leistungsfähigkeit von Squirrel demonstrieren, zum anderen die beiden oben besprochenen Modelle gegeneinander vergleichen.

Messumgebung

Für die Messungen wurde ein Simulation genutzt. Diese Simulation erzeugt pro Knoten einen Client im simulierten Netzwerk. Um dennoch möglichst realistisch zu bleiben, wurde auf die Logfiles von zentralen Webcaches zurückgegriffen. In [SID02] werden zwei verschieden Logfiles behandelt, hier wird ein Logfile, der in Cambridge entstanden ist behandelt. Der Logfile hat folgende Eigenschaften:

Zeitdauer	31 Tage
Anzahl HTTP Anfragen	0,971 Millionen
Durchschnittl. Anfragerate	0,362 req/s
max. Anfragerate	186 req/s
Anzahl Objekte	0,469 Millionen
Anzahl Clients	105
Größe statischer Objekte	2,21 GB
totale ex. Bandbreite	5,7 GB

Da über die Cachefähigkeit der Objekte aus dem Logfile wenig zu entnehmen ist, wird ein einfaches Verfahren angewendet. Alle Objekte die über HTTP GET Anfragen (ohne SSL) und ohne Zeichen wie '?', '=' oder das Wort 'cgi' in der URL aufgerufen wurden, werden als statisch betrachtet. Alle anderen Anfragen werden in der Simulation sofort an den Ursprungsserver weitergeleitet.

Zu Beginn der Simulation werden die Caches wie folgt aufgewärmt. Wurde laut Logfile die erste Anfrage nach einem Objekt erfolgreich beantwortet, so wird dieses Objekt auch im Squirrelcache als schon bekannt vorrausgesetzt. Während des Experiments werden dann Anfragen in der zeitlichen Abfolge gestellt, wie sie dem Logfile zu entnehmen sind. Auch die Verdrängungsstrategie wird aus dem Logfile übernommen.

Externe Bandbreite

Die externe Bandbreite wird als Anzahl der Bytes, die zwischen Squirrel und den Webservern übertragen wird, definiert. In diesem Abschnitt wird betrachtet wie die externe Bandbreite durch den Einsatz der beiden Modelle minimiert wird, und wie diese Veränderung der externen Bandbreite von der Größe der lokalen Caches abhängt.

In Abbildung 3.3 wird die externe Bandbreite in GB über den gesammten Zeitraum aufgetragen. Auf den einzelnen Knoten wird eine LRU Verdrängungsstrategie gefahren. Die zwei gepunkteten Linien zeigen die Extremwerte, einmal für keinen Cache, die andere zeigt die externe Bandbreite für einen zentralen Cache.

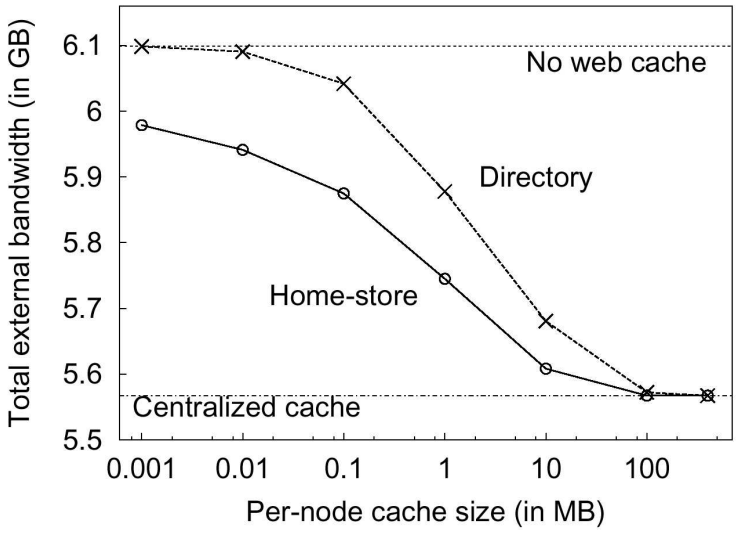

Abbildung 3.3: Externe Bandbreite bei variabler lokaler Cachegröße

Man sieht deutlich, dass schon zwischen 0,01 und 1 MB Cachegröße die externe

Bandbreite reduziert wird. Dies gilt für beide Modelle, allerdings schneidet der *Home-Store* Ansatz besser ab, da hier schon bei weniger lokalem Speicher eine Reduzierung erreicht wird.

Eine Erklärung für diesen Unterschied ist das unterschliedliche Browseverhalten von verschiedenen Benutzern. Während im *Home-Store* Ansatz alle Objekte mittels der Hashfunktionen gleichmäßig über alle Knoten verteilt werden, wird im *Directory* Modell auf Knoten, die viel browsen, schneller ein Objekt aus dem Cache gestrichen um für neue Platz zu machen.

Fehlertoleranz

Wenn die gesamte Anbindung ans Internet ausfällt, so liefern beide, Squirrel und ein zentraler Webcache weiterhin die gecachten Inhalte.

Ein anderer Fehlerfall ist der Ausfall eines internen Routers, der zu einer Aufteilung des internen Netzwerks führt. Im Fall der zentralen Webcaches nutzt nur noch ein Teil des Netzes den Webcache. Squirrel organisiert sich Dank der Eigenschaften von Pastry so um, dass in jedem Teil des zerfallenen Netzes ein neues funktionsfähiges Squirrel läuft.

Ein dritter häufigerer Fehlerfall tritt ein, wenn zwar noch Verbindung zum Internet besteht, aber einzelne Knoten im Netz ausgefallen sind. Das Squirrelnetzwerk funktioniert weiter, nur die Information auf dem jeweiligen Knoten muss von den jeweiligen Webservern neu angefordert werden. Wenn dagegen der zentrale Webcache ausfällt, ist der gesamte gecachte Inhalt nicht benutzbar.

Messungen für den Ausfall eines einzelnen Knotens haben für den Logfile aus Cambridge ergeben, dass im Durchschnitt 0,95% Leistungseinbußen entstehen. Im schlechtesten Fall können es bis zu 3,34% sein. Diese Werte gelten für den *Home-Store* Ansatz. 1,68% beziehungsweise 12,4% sind es für den *Directory* Ansatz.

Es liegt auf der Hand, dass für größere Netzwerke mit noch mehr Clients die Auswirkungen eines einzelnen Ausfalls weiter abnehmen.

Über Nacht abgeschaltete Rechner können in den meisten Fällen vorher noch ihre populärsten Inhalte an Nachbarn weitergeben, so dass der Verlust nicht zu sehr ins Gewicht fällt.

3.3.4 Diskussion der Ergebnisse

Zwei Faktoren wurden bei den Messungen noch nicht betrachtet. Es ist mit einem weiteren Vorteil für Squirrel zu rechnen, wenn man die Festplattenbelastung mit berücksichtigt. Die Leistung eines zentralen Webcaches kann durch seine Festplatte beschränkt sein.

Der zweite Faktor ist der Hauptspeicher. Auf einem zentralen Webserver steht nur eine feste Größe zur Verfügung um Inhalte zwischen zu speichern. Squirrel kann ungenutzen Speicher aller Clients nutzen um schnell auf Anfragen antworten zu können.

Die Ergebnisse zeigen, dass die Verteilung der Last mittels der Hashfunktion gut funktioniert.

Unter den hier angenommenen Voraussetzungen war der *Home-Store* Ansatz klar der bessere. Allerdings könnte der *Directory* Ansatz bei veränderten Randbedingungen, wie geographisch verteilte Squirrelnetzwerke oder höhere interne Latenz wieder interessanter werden.

3.3.5 Vergleichebare Arbeiten

Squirrel kann als Verschmelzung von kooperativen Webcaches [CDN+96, GCR98] und P2P Routingsystemen gesehen werden.

Kooperative Webcaches arbeiten selbständig und tauschen untereinander Information aus. Dies kann zum Beispiel die Trefferrate für eine Gruppe kleinerer Netzwerke verbessern.

Problematisch bei den kooperativen Ansätzen ist die Integration der verteilten Caches. Daher gibt es Versuche Hierarchien und Anpassung an lokale Anforderungen mittels eines Clusters von Webcaches zu realisieren [KSW98].

Unterschiedlich zu den oben genannten Alternativen benötigt Squirrel keine weitere Infrastruktur.

Ein kommerzielles Produkt, MangoSofts CacheLink [Man04], erlaubt den verteilten Zugriff auf die lokalen Caches wie bei Squirrel. Allerdings werden maximal 250 Clients unterstützt und über die Funktionsweise ist nichts bekannt.

3.3.6 Fazit

Der Einsatz eines P2P basierten Systems zum Cachen von Websites scheint sinnvoll. Die Ergebnisse der Messungen versprechen eine ähnliche Leistung wie ein zentraler Webcache. Die Implementierung kann mittels des *Home-Store* Ansatzes einfach gehalten werden.

Probleme treten vielleicht im echten Einsatz auf, die den Gewinn des verteilten Caches schmälern könnten. Zum Beispiel könnte es wenig verfügbarer Cache an Wochenenden nötig machen, doch einige Squirrel Instanzen auf dezidierten Rechnern laufen zu lassen, damit die Leistung für die restlichen Benutzer noch erträglich bleibt.

Um ein abschließendes Urteil zu fällen, sollte man auf die ersten Erfahrungen eines praktischen Einsatzes warten.

3.4 SplitStream

3.4.1 Einführung

SplitStream ist ein System zur Verbreitung von Daten mit hoher Bandbreite. Unter den Daten kann man sich beispielsweise eine Live-Übertragung in Bild und Ton vorstellen. Dabei gibt es viele Empfänger aber nur eine Quelle, daher auch die Bezeichnung Multicast für ein solches System. Herkömmliche Systeme verteilen die Daten an die Empfänger unter Benutzung einer Baumstruktur. Abbildung 3.4 zeigt eine solche typische Baumstruktur.

Beim Betrachten der Abbildung 3.4 fällt auf, dass eine kleine Anzahl innerer Knoten die Last trägt, die Daten weiterzuleiten. Diese Last kann man durch speziell dafür ausgelegte Router in den Griff bekommen. Aber in einem kooperativen Multicast System, in dem ein Multicast auf Anwendungsschicht (im Gegensatz zu einem Multicast, der in Hardware implementiert wird) durchgeführt wird, ergeben sich Probleme, da ja ein Knoten freiwillig die Position eines inneren Knotens übernehmen müsste. Die Blattposition würde dagegen jeder anstreben. Das hätte zur Folge, dass die wenigen inneren Knoten überlastet werden.

Gewünscht wäre, dass jeder Knoten so viel zur Verteilung der Daten beiträgt, wie er selbst von der Verteilung profitiert. Zusätzlich kommt hinzu, dass einige Knoten überhaupt nicht die Bandbreite besitzen, die für einen inneren Knoten benötigt wird.

SplitStream benutzt Multicasts auf Anwendungsschicht. Der Aufwand wird möglichst gleichmäßig über alle teilnehmenden Knoten verteilt. Ausserdem ist SplitStream in der Lage unterschiedliche Einschränkungen der Knoten bezüglich ihrer Bandbreite einzuhalten. SplitStream baut auf Pastry und Scribe [BCMR02] auf. In diesem Teil werden das Design und einige vorläufige Ergebnisse präsentiert.

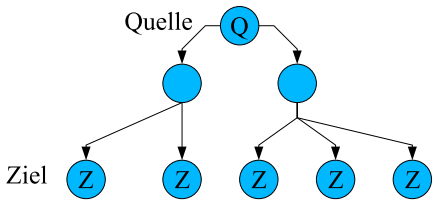

Abbildung 3.4: Multicast Baum

3.4.2 Idee

Zuerst scheint es bei Betrachtung der Abbildung 3.4 unmöglich den Empfängern auch einen Teil der Last für die Verteilung der Daten aufzubürden. Um aber dennoch die Ziele der Lastverteilung und Berücksichtigung individueller Bandbreitenbeschränkungen zu verwirklichen, wird die Baumstruktur aus Abbildung 3.4 verändert. Der Grundgedanke hierbei ist es anstatt einem Baum (herkömmliche Lösungen), mehrere Bäume zur Verteilung der Daten aufzubauen. Hierzu wird der Datenstrom in sogenannte *Stripes* aufgespalten. Und für jeden dieser *Stripes* wird ein eigener Baum erzeugt. Man kann also sagen, dass an Stelle des Baums ein ganzer Wald tritt.

Ziel ist es jetzt natürlich diesen Baum so zu konstruieren, dass zum einen jeder Knoten entsprechend seiner Nutzung des Systems auch an der Last seinen Anteil trägt, zum anderen die Bandbreitenbeschränkungen jedes Knotens nicht überschritten werde. Zusätzlich gilt es natürlich die Bäume so zu optimieren, dass die Verzögerung von der Quelle der Daten zu den Empfängern minimiert wird.

Zur Lastverteilung ist es wichtig, dass jeder Knoten in einem Baum innerer Knoten ist, in allen anderen dagegen Blattknoten. Somit wird erreicht, dass die Belastung gleichmäßig verteilt wird. Abbildung 3.5 zeigt einen Wald aus zwei *Stripes* in

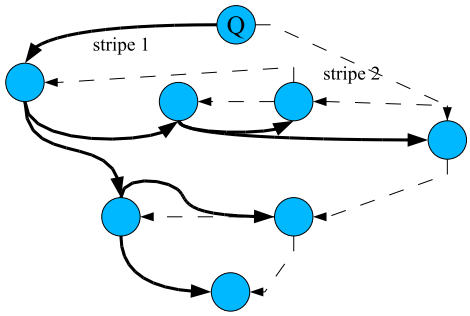

Abbildung 3.5: Multicast Bäume für zwei *Stripes*

dem diese Bedingung erfüllt ist. Hatte also der ursprüngliche Baum die Bandbreite B pro Verbindung hat man bei k *Stripes* die Bandbreite B/k pro *Stripe*.

Da idealerweise jeder Knoten nur in einem Baum innerer Knoten ist, betrifft der Ausfall dieses Knotens auch nur einen *Stripe*. Mit einer Kodierung der Daten wie zum Beispiel mit Erasure Coding [BBK02] oder Multiple Description Coding [Goy01] kann der Empfänger die restlichen *Stripes* trotzdem noch nutzen.

Die größte Herausforderung für SplitStream ist die effiziente Konstruktion des Multicast Waldes um die Last zu verteilen. Und dabei die Bandbreitenbeschränkung

und die Verzögerung zu berücksichtigen.

Hierbei hilft Scribe, ein anderes Framework, welches von SplitStream dazu benutzt wird die einzelnen Bäume aufzubauen. Ein Großteil der Aufgabe, die Konstruktion der einzelnen Bäume, wird von Scribe übernommen.

3.4.3 Scribe

Scribe ist ein System welches auf Pastry aufbaut. Um einen Baum aus den teilnehmenden Knoten zu konstruieren, wählt Scribe zuerst einen zufälligen Pastry Schlüssel. Dieser Schlüssel wird hier als GuppenID bezeichnet. Der Trick, den Scribe jetzt benutzt, ist einfach. Mit Pastry wird die Route jedes Knotens zum Knoten, der der GruppenID am nächsten liegt, gefunden. Der Baum besteht jetzt aus allen Knoten und den Routen, die Pastry gefunden hat.

Die Eigenschaften von Pastry gelten somit auch für die Kanten in diesem Baum. Das bedeutet im Besonderen, dass die Routen effizient gewählt werden.

Um neue Knoten zu einem Baum hinzuzufügen wird von diesem Knoten aus so lange in die Richtung der GruppenID geroutet, bis man auf einen Knoten trifft, der schon Teil des Baumes ist. So kann man mit Scribe einfach eine große Menge an Bäumen und eine ausreichende Menge von Knoten pro Baum bewältigen.

Mittels einer *Anycast*-Erweiterung von Scribe [CDHR03] ist es auch möglich nach Resourcen im Netz zu suchen. Dies wird vorallem für das Auffinden von freien Kapazitäten genutzt.

3.4.4 Design

SplitStream stelle ein Framework für Anwendungen dar, die Daten mit hoher Bandbreite an mehrere Knoten verteilen wollen. Daher beginnen wir diesen Abschnitt mit der Schnittstelle, die SplitStream den Anwendungen anbietet.

Aufgaben der Anwendungen

Damit eine Anwendung SplitStream benutzen kann, muss sie zuerst den Datenstrom, der an der Quelle vorliegt, aufspalten. Diese Aufgabe muss bei der Anwendung liegen, da hier anwendungsspezifisches Wissen einfließen muss. Bei der Kodierung muss folgendes eingehalten werden:

1. jeder *Stripe* sollte ungefähr die selbe Bandbreite benötigen
2. jeder *Stripe* sollte die selbe Menge an Information enthalten und es dürfen keine Hierarchien innerhalb der *Stripes* existieren
3. fehlertollerante Kodierung der *Stripes*, so dass eine Teilmenge der *Stripes* ausfallen kann und trotzdem noch Information rekonstruiert werden kann

Um den dritten Punkt zu erfüllen können Anwendungen explizit Informationen aus anderen Peers anfordern. Ein anderer Weg sind redundante Informationen in den *Stripes*. Mit der MDC Kodierung [Goy01] wäre es möglich bei einem Film die Qualität zu verringern, wenn einige *Stripes* ausfallen. Ein weiterer Vorteil dieser Kodierung besteht darin, dass Clients mit niederer Bandbreite auch weniger *Stripes* abonieren können, und somit eben eine geringere Qualität erhalten.

Innere Knoten disjunkte Bäume

Für jeden der k *Stripes* wird ein anderer Multicast Baum benutzt. SplitStream nutzt Eigenschaften von Pastry um Bäume zu realisieren, die in ihren inneren Knoten disjunkt sind. Da der Scribe Baum aus Routen der Mitglieder zum Wurzelknoten

besteht, haben alle inneren Knoten mindestens ein Präfix der Länge eins mit der GruppenID gemeinsam. Dadurch kann man sicherstellen, dass k *Scribe*-Bäume disjunkte innere Knoten haben, wenn man GruppenIDs wählt die sich alle in der ersten Stelle unterscheiden.

Hier gehen wir davon aus, dass der Parameter b von Pastry so gewählt wird, dass $k = 2^b$ gilt.

Individuelle Ein- und Ausgangsbandbreite

Der entstandene Wald aus Scribe Bäumen entspricht dem Ziel innere Knoten disjunkte Bäume zu haben. Ausserdem sind die Bedingungen, die an die Eingangsbandbreite gestellt werden, automatisch erfüllt. Dies liegt daran, dass jeder Knoten selbst entscheidet wie viele *Stripes* er abonieren will.

Ein Problem gibt es allerdings noch, nämlich die Ausgangsbandbreite. Diese kann überschritten werden, sobald sich zu viele Kinder an einem Knoten anmelden. Hierfür hat Scribe einen Mechanismus die Anzahl der Kinder für einen Knoten zu beschränken. Wenn ein Knoten seine maximale Outputbandbreite erreicht hat, übergibt er an den anfragenden Knoten eine Liste seiner Kinder im Baum. Das angehende Kind versucht es nun bei den Knoten dieser Liste. Wenn hier wiederum nichts frei ist, so wird dieses Verfahren rekursiv bis an die Blätter des Baums fortgeführt. Man sollte meinen spätestens hier muss die Anfrage Erfolg haben, und der Knoten endlich aufgenommen werden, da die Blätter ja keine Kinder haben. Dies ist nur richtig für einen Baum, betrachtet man den ganzen Wald, so sollte ja jeder Knoten in einem Baum innerer Knoten sein. Es ist also möglich, dass alle Blätter eines Baums keine Kapazitäten mehr frei haben.

Dieses Problem wird mittels eines Tricks gelöst. Ein anderer Knoten muss gehen, so hat der neue Knoten Platz. Über die Auswahl des abgestoßenen Knotens kann man den Wald optimieren.

Da es nun wieder einen suchenden Knoten gibt, muss eine spezielle Gruppe benutzt werden. In dieser Gruppe werden alle Knoten mit freien Kapazitäten gespeichert.

Wird dort keine freie Kapazität gefunden, so wird der Anwendung mitgeteilt, dass momentan keine Kapazitäten zur Verfügung stehen.

Details zu diesem Verfahren beschreibt [CDK+03].

Unabhängige Pfade

Durch die Einschränkungen, die sich beim Aufbau des Waldes durch den vorhergehenden Abschnitt ergeben, kann man in der Praxis nicht garantieren, dass jeder Knoten für jeden seiner abonierten Stripes unabhängige Pfade durch das Netzwerk benutzt. Da diese Eigenschaft aber wichtig für die Ausfallsicherheit und Fehlertolleranz von SplitStream ist, muss man abwägen zwischen Leistung und Ausfallsicherheit. SplitStream erlaubt es der Anwendung hier Prioritäten zu setzen.

3.4.5 Messergebnisse

Für vorläufige Tests wurden 40.000 SplitStream Knoten in einem emulierten Netzwerk erzeugt. Benutzt wurde der Georgia Tech Netzwerktopologie Generator. Es wurde ein Scribe Wald für 16 *Stripes* erzeugt. Jedem Knoten wurden Bandbreitenbeschränkungen entsprechend einer Verteilung unter Gnutella Knoten im May 2001 zugeteilt.

Während der Waldkonstruktion wurden von jedem Knoten durchschnittlich 56 Nachrichten bearbeitet. Die Verzögerung war im Vergleich zu IP Multicast 1,35 mal

so groß. Der maximale Delay 1,8 mal so groß. Diese Steigerung spiegelt den größeren Aufwand wieder, die Last gleichmäßig auf allen Knoten zu verteilen.

Man beobachtete außerdem den Grad der Unabhängigkeit in diesem SplitStream Wald. Ohne das die Ausgangsbandbreite die Eingangsbandbreite in einem Knoten überstieg, hatten über 95% der Knoten 12 oder mehr *Stripes* mit unabhängigen Pfaden.

3.4.6 Vergleichbare Arbeiten

Es gibt viele Multicast Systeme, die auf der Anwendungsschicht multicasten. Aber im Gegensatz zu SplitStream basieren sie alle auf einem einzigen Multicast Baum.

Overcast [GSG00] schaltet spezielle Server zu einem Multicast Baum zusammen und versucht durch Messungen und Schätzungen der Bandbreiten den Baum zu optimieren. Die größten Unterschiede zwischen Overcast und SplitStream sind:

1. Overcast nutzt spezielle Server, SplitStream nutzt alle Teilnehmer
2. Overcast erzeugt einen bandbreitenopimierten Multicast Baum, SplitStream erzeugt einen Wald von optimierten Mutlicast Bäumen

CoopNet [MRL99] betreibt einen zentralen Server zur Verteilung der Inhalte. Anfragende Knoten werden direkt vom Server bedient, solange dieser nicht überlastet ist. Tritt dieser Fall ein, leitet der Server Anfragen an die schon teilnehmenden Knoten weiter. Da auch hier die Daten gestriped sind, entstehen so mehrere Multicast Bäume. Es gibt allerdings zwei grundsätzliche Unterschiede.

1. CoopNet nutzt einen zentralen Algorithmus um die Bäume aufzubauen. SplitStream ist dezentral und daher besser skalierbar.
2. CoopNet versucht nicht jeden Knoten in die Last mit einzubeziehen, allerdings könnte diese Eigenschaft noch nachgerüstet werden.

3.4.7 Fazit

SplitStream bietet eine interessante Möglichkeit breitbandige Inhalte zu verteilen. In der Simulation wurde das Hauptziel der Lastverteilung erreicht. Zusätzlich gewinnt man noch Ausfallsicherheit und Anpassungsfähigkeit.

Aufgrund der vorliegenden Ergebnisse sollte man SplitStream für das Verteilen von großen Datenmengen benutzen.

Allerdings gilt hier, wie auch bei Squirrel, dass Erfahrungen in der Praxis noch ausstehen. Probleme könnten unter anderem zu große Delays in manchen Pfaden bereiten.

Literaturverzeichnis

[BBK02] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast, 2002.

[BCMR02] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks, 2002.

[CDHR03] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical report, 2003.

[CDK+03] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment, 2003.

[CDN+96] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, pages 153–164, 1996.

[GCR98] Syam Gadde, Jeff Chase, and Michael Rabinovich. A taste of crispy Squid. In *Proceedings of the Workshop on Internet Server Performance (WISP'98)*, 1998.

[Goy01] V.K. Goyal. Multiple description coding: Compression meet the network., 2001.

[GSG00] J. Gemmell, E. Schooler, and J. Gray. Fcast multicast file distribution, 2000.

[KSW98] Michal Kurcewicz, Wojtek Sylwestrzak, and Adam Wierzbicki. A distributed www cache. *Comput. Netw.*, 30(22-23):2261–2267, 1998.

[Man04] Mangosoft. Mangosoft homepage, 2004.

[MRL99] A. Mohr, E. Riskin, and R. Ladner. Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction, 1999.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[SID02] Antony I. T. Rowstron Sitaram Iyer and Peter Druschel. Squirrel: a decentralized peer-to-peer web cache. pages 213–222, 2002.

Kapitel 4

Infrastruktur und Methoden zur Entwicklung von Overlays

Seminarbeitrag von **Alexander Dieterle**

In diesem Kapitel werden zwei Systeme vorgestellt, die sich mit verschiedenen Schwerpunkten der Entwicklung verteilter Systeme befassen. *PlanetLab* ist ein weltweit verteiltes Netzwerk zur Entwicklung und zum Test von Netzdiensten und Infrastrukturdiensten. *MACEDON* stellt eine Infrastruktur bereit, um verteilte Algorithmen schneller und einfacher zu entwickeln und zu evaluieren. Des Weiteren wird ein Vergleich der Ressourcenverwaltung von *PlanetLab* und dem *Globus Toolkit* vorgenommen.

4.1 Einleitung

Das Interesse an verteilten Computersystemen hat in den letzten Jahren exponentiell zugenommen. Einerseits trägt der günstige Erwerb von Heimcomputern dazu bei, dass teure Hochleistungsrechner durch Rechnerbündel (Cluster) ersetzt werden, die aus vernetzten Heimcomputern bestehen und sich am gleichen Standort befinden. Andererseits bietet das Internet ein weltweites Netz, durch das mittlerweile fast alle Rechner auf unserem Planeten miteinander verbunden sind. Ein solches Netz, bei dem die zugehörigen Rechner geographisch weit voneinander entfernt sind, wird auch als Raster (Grid) bezeichnet. Die große Anzahl an Rechnern und die Tatsache, dass die Infrastruktur des Internet immer besser ausgebaut ist und somit die Vernetzung der einzelnen Rechner immer leistungsfähiger wird, machen die Nutzung des Internet als „Großrechner“ zunehmend attraktiver.

Um die Rechner über das Internet systematisch miteinander zu verbinden, werden semantische Überlagerungsnetzwerke (Overlay Networks) benutzt. Damit ist es möglich, die große Menge an Ressourcen miteinander zu verbinden. Den größten Anwendungsbereich solcher Overlays sieht man momentan bei Peer-to-Peer Systemen wie Kazaa oder Gnutella, mit denen Heimanwender Dateien austauschen können. Ein Beispiel für die gemeinsame Nutzung einer anderen Ressource, hier der Rechenleistung, ist Seti@Home, bei dem Datenpakete zu Heimrechnern geschickt werden. Wenn sich diese im Leerlauf befinden, was zu 95% der Zeit der Fall ist, werden die Setidaten berechnet und wieder zurückgeschickt.

4.2 Motivation

Da sich Overlayneztwerke in der Anfangsphase ihrer Entwicklung befinden, gibt es noch wenige Standards, auf denen man aufbauen kann. Bis jetzt setzen alle Anwendungen direkt auf dem Internet auf. Viele Forschungsgruppen versuchen nun, weitere Abstraktionsschichten zu etablieren. Dies soll verhindern, dass das Rad jedes Mal neu erfunden wird, wie es bei aktuellen Overlayanwendungen der Fall ist. Die hier vorgestellten Systeme beschäftigen sich mit verschiedenen Abstraktionsebenen. Die *PlanetLab* Vereinigung forscht im Bereich der Netzwerk- und Infrastrukturdienste, also auf einer unteren Ebene, die auf dem Internet aufsetzt, um höherwertige Dienste für darüber liegende Schichten zu bieten. *MACEDON* dagegen ist ein System, das alle unteren Schichten schon fertig bereitstellt, um Forschern eine Schnittstelle auf hoher Ebene zu bieten, die eine schnellere und einfachere Entwicklung verteilter Anwendungen ermöglichen soll.

4.3 Überblick

In den folgenden Abschnitten werden zwei Overlayprojekte genauer untersucht, *PlanetLab* [BBC+04] und *MACEDON* [RKB+04]. Außerdem wird *PlanetLab* mit einem dritten Projekt namens *Globus* [RBC+04] in Bezug auf die Ressourcenverwaltung verglichen.

4.3.1 PlanetLab

PlanetLab ist ein weltweit verteiltes Netzwerk zur Einrichtung und Bewertung von Netzwerkdiensten, die den Anforderungen eines global verteilten Netzes gerecht werden. Die Plattform besteht im April 2004 aus ungefähr 400 Rechnern an 160 Standorten in 25 Ländern. Zur Teilnahme an der im Juli 2002 ins Leben gerufenen Plattform genügt es ein paar Rechner zur Verfügung zu stellen, die ans Internet angeschlossen sind.

PlanetLab bietet zwei Konzepte, welche die Forschungsmöglichkeiten im Bereich von Overlaynetzwerken unterstützen sollen:

Verteilte Virtualisierung (distributed virtualisation) Abstraktion von Hardware in Form von virtuellen Maschinen stellt ein bekanntes Konzept dar. Jedoch sind bestehenden Systeme nur für den lokalen Einsatz ausgelegt. *PlanetLab* erweitert dieses Konzept, in dem virtuelle Maschinen auf verschiedenen Rechner über virtuelle Verbünde, so genannte *Slices*, logisch miteinander in Zusammenhang gebracht werden. In diesen *Slices* können nun auf einer einheitlichen Schnittstelle Netzwerkdienste entwickelt werden.

Entbündelte Verwaltung (unbundled management) Hinter diesem Begriff verbirgt sich die Frage, wie man die Infrastruktur von *PlanetLab* selbst vom zugrund liegenden Betriebssystem abtrennt. Es soll möglich sein neben Netzdiensten auch Infrastrukturdienste von *PlanetLab* selbst über die Plattform zu entwickeln und zu testen ohne das Betriebssystem zu modifizieren. Am besten ist dieses Konzept mit dem Aufbau von Micro-Kernels zu vergleichen, bei denen auch versucht wird, möglichst viel Funktionalität aus dem Kern in höhere Schichten zu verlagern.

Im Abschnitt über *PlanetLab* sind die Anforderungen an die Infrastruktur, die diese Konzepte mit sich bringen und deren Umsetzung im *PlanetLab* Betriebssystem beschrieben.

4.3.2 MACEDON

Das Kürzel *MACEDON* steht für „Methodology for Automatically Creating, Evaluating and Designing Overlay Networks“. *MACEDON* ist ein System, dessen Ziel es ist, das Implementieren und Testen von Overlayalgorithmen zu vereinfachen. Außerdem soll es möglich sein Algorithmen, die mit *MACEDON* implementiert wurden, durch einheitliche Evaluationswerkzeuge besser miteinander zu vergleichen als es bisher möglich war.

4.4 PlanetLab

In diesem Abschnitt wird zuerst die Zielsetzung und Ausrichtung von *PlanetLab* erläutert. Dann werden die Anforderungen an ein solches System vorgestellt und die konkrete Umsetzung durch *PlanetLab* beschrieben.

4.4.1 Ziele

Der Entwurf von *PlanetLab* wurde durch einige grundlegende Ziele geprägt, die sich auf die Wahl der Benutzergemeinde sowie auf die Entwicklung des Internet beziehen.

Die *PlanetLab* Benutzergemeinde lässt sich in zwei große Gruppen unterteilen, die sehr verschiedene Anforderungen stellen. Einerseits soll eine Plattform zur Verfügung gestellt werden, auf der Forscher mit globalen Netzdiensten experimentieren können. Gleichzeitig soll Endbenutzern eine reale Plattform zur Verfügung gestellt werden, auf der sie neue Netzwerkdienste einrichten und benutzen können, um auf höheren Schichten zu experimentieren.

Weiterhin soll die *PlanetLab* Plattform dazu dienen den Wettbewerb im Bereich multipler Netzwerkdienste zu fördern und durch deren Entwicklung auch die Entwicklung des Internet hin zu einer dienstorientierten Architektur zu fördern.

4.4.2 Beziehungen

Ein verteiltes System wie *PlanetLab* ist durch eine Reihe von Beziehungen geprägt, die dessen Entwurf entscheidend beeinflussen.

Eine Beziehung besteht zwischen *PlanetLab* und den Institutionen, die ihre Rechnerknoten zur Verfügung stellen. *PlanetLab* muss auf die Knoten zugreifen, um das *PlanetLab* Betriebssystem und die darauf laufenden Dienste zu administrieren. Die lokalen Administratoren der Knoten müssen Einfluss auf die Benutzung der Knoten und den Netzwerkdurchsatz nehmen können. Diese Beziehung erfordert eine geeignete Teilung der Kontrolle über die Knoten.

Eine weitere Beziehung bilden *PlanetLab* und seine Benutzer. Ein Benutzer bekommt von *PlanetLab* einen Zugang zu einem *Slice*. Dadurch werden mehrere Knoten vereint, auf denen die Forscher mit ihren Diensten experimentieren können.

Dieser Beziehung begegnet *PlanetLab* mit seinem Konzept der *verteilten Virtualisierung.*

Die dritte Beziehung besteht zwischen *PlanetLab* und denjenigen Forschern, die an den Infrastrukturdiensten von *PlanetLab* arbeiten. Um dies zu ermöglichen bedarf es des Konzepts der *entbündelten Verwaltung.*

Die letzte Beziehung existiert zwischen *PlanetLab* und dem Internet an sich. Die Erfahrung hat gezeigt, dass Netzüberwachungsdienste von Knotenstandorten empfindlich auf Experimente mit Netzdiensten über *PlanetLab* reagieren. Hier müssen Policen eingeführt werden, die den Netzverkehr regeln und eine „Überflutung“ von Standorten verhindern, um das Internet vor *PlanetLab* zu „schützen“.

4.4.3 Anforderungen

a. Verteilte Virtualisierung (Distributed Virtualization)

Im *PlanetLab* System wird jeder Rechnerknoten durch eine oder mehrere virtuelle Maschinen (*VM*) repräsentiert. Die Ressourcen der Rechnerknoten, die durch die *VM* zur Verfügung gestellt werden, werden zu *Slices* zusammengefasst und den Benutzern zur Verfügung gestellt. Es können beliebig viele *Slices* nebeneinander auf einem Knoten laufen und natürlich können verschiedene *Slices* auch unterschiedliche Knoten beinhalten. Dies verdeutlicht Abbildung 1. Die Graphik zeigt miteinander verbundene Rechnerknoten und zwei Beispieldienste (Chord, Pastry) die auf verschiednen Knoten laufen.

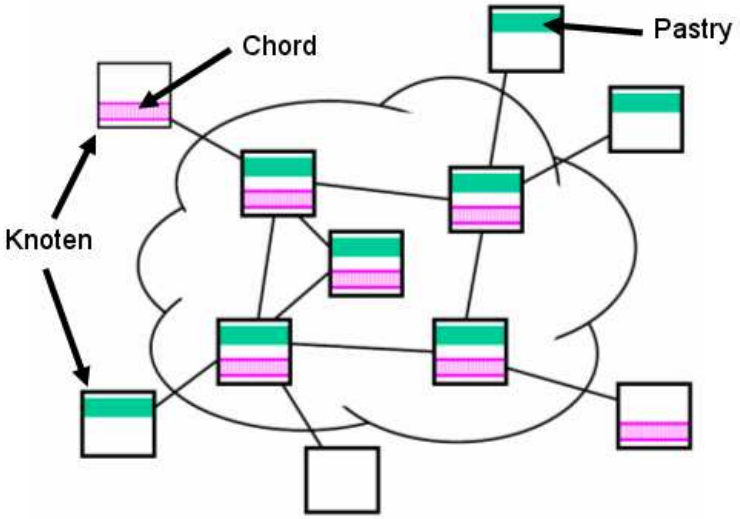

Abbildung 4.1: Verteilte Dienste werden zu Slices zusammengefasst

PlanetLab stellt hier Dienste zur Verfügung zur Erstellung und Initialisierung von *Slices*, und zur Bindung von *Slices* an Knotenressourcen. Dabei wird darauf geachtet, dass *Slices* nur die nötigste Funktionalität aufweisen, um größtmöglichen Spielraum für die Dienste zu lassen, die später in den *Slices* laufen. Es wird angestrebt, dass *Slices* eine unabhängige Schnittstelle bieten, die es erlaubt jede gewünschte Softwareumgebung zu laden und somit uneingeschränkt benutzbar ist. Dieses Ziel liegt allerdings noch in der Ferne. Momentan wird die Schnittstelle, die ein *Slice* zu Verfügung stellt, durch eine UNIX API repräsentiert und führt somit nur Software aus, die mit UNIX Systemen kompatibel ist.

Isolation der Slices Virtuelle Maschinen auf einem einzelnen Rechner vermitteln der darüber liegenden Software eine isolierte Sicht auf den Rechner. Dadurch „sieht“ die höherschichtige Software die anderen Prozesse oder *VMs* auf diesem Rechner nicht. Genauso müssen auch *Slices* voneinander isoliert werden. Der Unterschied besteht lediglich darin, dass mehrere isolierte *VMs* miteinander gekoppelt werden. Um bestimmten Dienstanforderungen gerecht zu werden, müssen einzelne Knoten bestimmte Zusicherungen in Bezug auf ihre Ressourcen machen können, z.B. für Messungen oder Softrealtimeanwendungen.

Diese müssen durch das *PlanetLab* OS folgendermaßen bereitgestellt werden:

Die Zuordnung und zeitliche Koordination von Ressourcen in Bezug auf Taktzyklen, Netzwerkbandbreite, Arbeits- und Plattenspeicher muss so bewerkstelligt werden, dass *Slices* auf demselben Knoten laufen können, ohne sich gegenseitig in Bezug auf die Leistung stark zu beeinflussen.

Es muss eine Aufteilung und Verwaltung der Namensräume stattfinden, um die einzelnen *Slices* auf einem Knoten zu unterscheiden und gegeneinander abzusichern. Dazu müssen z.B. Netzwerkadressen, Datei- und Verzeichnisnamen in den Kontext der jeweiligen *VM* gebracht werden.

Den *VMs* muss eine stabile Programmierbasis zugrunde gelegt werden, damit in *Slices* ausgeführter Kode das Betriebssystem des Knotens nicht korrumpieren kann. Dazu bedarf es der Einschränkung der Zugriffsrechte, die einem *Slice* bereitgestellt werden.

Isolation von PlanetLab gegenüber dem Internet Neben den Knoten muss auch das Internet vor *Slices* geschützt werden, da kaputte oder bösartige Dienste großen Schaden anrichten könnten. Hieraus ergeben sich folgende Anforderungen an die Ressourcenvergabe von *PlanetLab*:

Die Begrenzung der Ressourcen einzelner Knoten muss von *PlanetLab* und den Knoteninhabern ausgehandelt werden. Zur Begrenzung des Schadens, den ein Dienst z.B. durch Konsum der vollen Bandbreite in einem lokalen Uni-Netzwerk anrichten könnte, müssen Ressourcenbelegungen einzelner Dienste isoliert voneinander limitiert werden können.

Die Ressourcenbelegung muss außerdem einfach zu überwachen sein. Zusätzlich zur „normalen“ Überwachung durch Messinstrumente wird hier ein Mechanismus gefordert, mit dessen Hilfe man auch die Dienste identifizieren kann, die Schaden durch ungewollte Ressourcennutzung anrichten.

b. Entbündelte Verwaltung (Unbundled Management)

Globale Netzdienste sind ein relativ neues und attraktives Forschungsgebiet. Dies betrifft vor allem Dienste, die benötigt werden, ein System wie *PlanetLab* zu verwalten. Neben den Diensten, die für Endbenutzer entwickelt werden, sind für ein System wie *PlanetLab* natürlich auch die Dienste interessant, welche die eigene Infrastruktur bilden. Diese sind entscheidend für die Leistung des Systems. Deshalb versucht *PlanetLab* die eigenen Dienste so gut wie möglich zugänglich und so leicht wie möglich ersetzbar zu machen. Außerdem sollen Infrastrukturdienste parallel nebeneinander ausgeführt werden können. Dadurch können Forscher das System ständig um neue Dienste erweitern und diese gegeneinander testen. Solche Dienste sind z.B. *Slice* Erzeugung, Ressourcen- und Topologieerkennung, Leistungsüberwachung und Softwareverteilung. *Entbündelte Verwaltung* bedeutet in diesem Zusammenhang, dass Infrastrukturdienste in eigenen *Slices* und somit auch ausgeführt werden können. Diese neue Architektur soll das Problem klassischer Architekturen, die hinsichtlich der Einführung neuer grundlegender Dienste unflexibel sind, auf ein Minimum reduzieren.

Die daraus folgenden Anforderungen an das *PlanetLab* OS sind:

- Minimierung der Funktionalität im *PlanetLab* Kern. Es sollte nur die Funktionen im Kern implementiert werden, die absolut notwendig sind. Alle anderen Funktionen sollten in Form von Infrastrukturdiensten implementiert werden.

- Maximierung des Wettbewerbs zwischen Diensten. Dies bedingt einen gemeinsamen Zugriff auf die Schnittstelle zwischen dem Betriebssystem und den Diensten. Zudem muss dieser Zugriff ohne spezielle Privilegien, die z.B. nur ein Superuser hat, ausgeführt werden können.

- Obwohl die Erzeugung von *Slices* in die Dienstebene verlagert ist, bedarf es einer initialen Urladerfunktion, die den ersten *Slice* produziert, indem dann die anderen Erzeugerdienste laufen können. Weiterhin muss eine Schnittstelle vorhanden sein, mit der *Slice* Erzeuger neue *VMs* erstellen können.

c. Architektur

Da die Forschergemeinde bereit war *PlanetLab* zu nutzen, als die ersten Maschinen damit eingerichtet wurden, stand es außer Frage, auf ein Betriebssystem zu warten, das alle Vorraussetzungen erfüllt, die ein flächendeckendes Overlaysystem wie *PlanetLab* stellt. Programmierer portieren ihre Applikationen nur ungern auf andere APIs. Außerdem ziehen sie ein etabliertes und umfangreiches Betriebssystem einer speziell für eine Forschungsrichtung geschriebenen API vor. So wurde als zugrunde liegendes Betriebssystem Linux gewählt, welches schon vom Großteil der Forscher benutzt wurde. Um Linux an die Bedürfnisse von *PlanetLab* anzupassen, mussten einige Modifikationen vorgenommen werden. Die Architektur von *PlanetLab* sieht folgendes Schema vor:

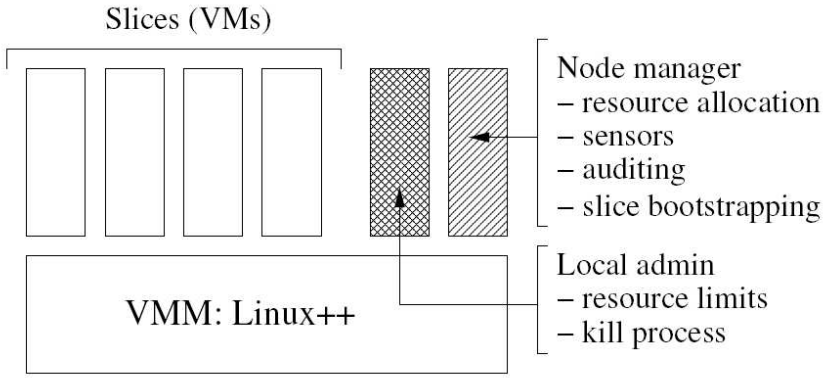

Abbildung 4.2: Architektur des PlanetLab OS

Die unterste Ebene wird von einem Überwachungsdienst für virtuelle Maschinen, dem *Virtual Maschine Monitor (VMM)* repräsentiert. Dieser *VMM* implementiert und isoliert die darüber liegenden *virtuellen Maschinen (VMs)*. Momentan wird der *VMM* durch den Linux-Kernel und einige Erweiterungen desselben. Es gibt zwei privilegierte *Root-VMs*, die spezielle Funktionen erfüllen. Das sind der *Knotenverwalter (Node Manager)* und der *lokale Administrator (Local Admin)*. Der *Knotenverwalter* überwacht und verwaltet alle *VMs*, die auf dem Knoten laufen. Er ist für die Erzeugung von *Slices* und deren Ressourcenzuordnung verantwortlich. Jegliche Kommunikation zwischen *VMs* und dem *VMM* geht über den *Knotenverwalter*. Aufrufe des *Knotenverwalters* sind immer lokal, d.h. ein verteilter Dienst kann nur über eine lokale Instanz auf den *Knotenverwalter* zugreifen. Des Weiteren stellt der

Knotenverwalter Policen für die Erstellung und Ressourcenzuteilung von *VMs* bereit. Diese Policen sind bisher hart implementiert, sollen aber künftig vom *lokalen Administrator* einstellbar sein. Der *lokale Administrator* ermöglicht dem Knoteninhaber die Restriktion der lokalen Ressourcen, die *PlanetLab* zur Verfügung stehen, wie z.B. die maximale Bandbreite, mit der *PlanetLab* den Rechner betreiben darf. Der *lokale Administrator* ist auch berechtigt Prozesse zu beenden. Dies ist notwendig, wenn ein kaputter Dienst nicht mehr über entfernten Zugriff gesteuert werden kann.

Man unterscheidet auch bei den *Slices* und den Diensten, die darin laufen, zwischen Privilegierten und Unprivilegierten. Privilegierte Dienste haben die Möglichkeit, über bestimmte Aufrufe beim *Knotenverwalter* z.B. Ressourcen für ihre *VM* anzufordern. Dienste, die für Endbenutzer sichtbar sind, sollen immer im unprivilegierten Modus laufen. Somit soll eine Trennung zwischen Infrastrukturdiensten und Anwendungsdiensten geschaffen werden.

Der Frage, wo und wie neue Funktionalität implementiert werden soll, begegnet *PlanetLab* mit zwei Richtlinien:

- Neue Funktionen sollten immer auf der höchst möglichen Ebene implementiert werden. Ist es also machbar, eine Funktion in einen Dienst auf *Slice* Ebene zu packen, sollte man dies tun, auch wenn vielleicht kleine Nachteile z.B. in Bezug auf Leistung entstehen, die bei einer Implementierung im *Knotenverwalter* nicht aufgetreten wären.

- Außerdem sollte sparsam mit der Privilegienvergabe umgegangen werden, d.h. einem Dienst sollten immer nur die Privilegien vergeben werden, die unbedingt notwendig sind. Es dürfen keine „root“-Freischeine an Dienste verteilt werden. Dies würde die Isolation und damit die Sicherheit der Systemkomponenten schwächen.

4.4.4 Umsetzung & OS (Schnittstelle und Implementierung)

a. Knoten Virtualisierung

Bei der Umsetzung der Knotenvirtualisierung stellt sich nun die Frage, auf welchem Level abstrahiert werden soll. Die Möglichkeiten werden durch die Forderung eingeschränkt, dass sehr viele *Slices* und damit *VMs* auf einem Knoten laufen sollen. Es gibt bekannte Systeme wie *VM*Ware, die den Rechner komplett in einer *VM* emulieren und nur die blanken Ressourcen bereitstellen. Dies verschafft einem 100% der Flexibilität, welche die darunter liegende Maschine bietet. Allerdings auf Kosten der Leistung, da man diese *VM* von Grund auf neu einrichten muss, das Betriebssystem inklusive. Auf einem Standartrechner kann man so nicht mehr als eine oder zwei *VMs* laufen lassen. Für *PlanetLab* suchte man also eine andere Lösung. Eine noch nicht ausgereifte Lösung verbirgt sich hinter dem Begriff „Paravirtualisierung“. Hier wird nicht die komplette Maschine instanziiert, sondern nur ein Teil der Prozessorbefehle und einige virtuelle Geräte.

Momentan findet die Virtualisierung des *PlanetLab* OS auf der Ebene der Systemaufrufe statt. Dafür bedient man sich einer Kernelerweiterung für *VServer*. Jede *VM* in *PlanetLab* wird durch einen *VServer* repräsentiert. Diese *VServer* laufen nebeneinander und können unabhängig verwaltet werden. Außerdem ermöglichen sie die Trennung der Namensräume, Benutzerzugänge und Dateien für die einzelnen *VMs*. Damit *Slice* Benutzer die zugeordneten *VMs* einrichten und verwalten können ist jeder *VServer* mit eingeschränkten Root Rechten ausgestattet. Diese erlauben ihm die volle Kontrolle über die Ressourcen der eigenen *VM*, jedoch hat er keinen direkten Zugriff auf die Systemressourcen oder das Betriebssystem.

Um die Skalierbarkeit der *VServer* zu erhöhen, wurde die Speicherzuteilung verbessert. Durch die Nutzung von „nur lesen“ (read-only) und „kopieren beim Schreibzugriff“ (copy-on-write) -Speichersegmenten wird die Effizienz der Speicherverwaltung von *VServern* gesteigert. Durch Anwendung von copy-on-write auf die Basisdateien, die alle *VServer* gemeinsam haben, werden nur 5,7 % des Speicherplatzes benötigt, den komplette Kopien belegt hätten. Dies erhöht bei einer großen Anzahl von *VMs* erheblich die Skalierbarkeit des Systems. Da copy-on-write nur Vorteile bringt, solange nicht viele Dateien geschrieben werden, muss hier noch der tatsächliche Nutzen im laufenden Betrieb getestet werden.

b. Isolation und Ressourcen Allokation

Die Isolation der *VMs* und *Slices* stellt eine wichtige Funktionalität dar. Dadurch werden Dienste voneinander abgetrennt und es kann eine Steuerung der Ressourcenzuordnung stattfinden. Das Hauptproblem in diesem Zusammenhang ist der Streit um gemeinsame Ressourcen. *PlanetLab* bedient sich hier dem Isolationsmechanismus von „Scout“. Der *Knotenverwalter* übernimmt die Ressourcenzuteilung. Die Kriterien zur Vergabe der Ressourcen sind jedoch ausgelagert. D.h. die Frage, wer von welcher Ressource wie viel bekommt, wird nicht vom *Knotenverwalter* beantwortet. Dafür läuft ein initialer Vermittlungsdienst in einem privilegierten *Slice*, der wiederum Ressourcen an andere Vermittlungsdienste vergibt. Diese können die Ressourcen dann frei an Dienste vergeben. Im Moment sind die Vergabekriterien noch zentral von *PlanetLab* gesteuert. Sie sollen den Knoteninhabern, also den *lokalen Administratoren*, übergeben werden. Diese können bis jetzt nur die Gesamtbandbreite des Netzanschlusses konfigurieren. Durch mehr Konfigurationsfreiheit der Knoteninhaber können Maschinen individueller angepasst werden. Das ist wünschenswert, da auf jedem Rechner andere *Slices* laufen und somit verschiedene Ressourcenbelastungen vorliegen.

Zur Vergabe von erneuerbaren Ressourcen wie Netzbandbreite und Taktzyklen geht *PlanetLab* nach dem Prinzip der Gleichberechtigung und der Garantie vor. So bekommt jeder der N *Slices*, die auf einem Rechner laufen auch 1/N Taktzyklen zugeteilt im Fall voller Auslastung. Allerdings werden auch Ressourcenkontingente zugesichert, wie z.B. eine bestimmte Bandbreite.

Der Linux-Kernel ist zwar in der Lage Prozesse zu koordinieren, jedoch kann er die Anforderungen an die zeitliche Koordination von *VServer* nicht erfüllen. Dafür wurde das die Zeitkoordination des Linux Kernels durch das *SILK-Modul* ersetzt, welches zusätzlich die zeitliche Ressourcenzuteilung von *VServern* regelt. Die Taktzyklen werden hier proportional zum Verhältnis an „Shares“, welche ein *VServer* besitzt, verteilt. Die Anzahl an Shares pro Knoten ist auf 1000 begrenzt. Die Shares werden Ressourcen Containern zugeteilt, in denen sich die *VServer* und ihre Prozesse befinden. Wenn ein *VServer* in seinem Container 10 Shares hält und die Anzahl der insgesamt vergebenen Shares 50 beträt, bekommt er 10/50 = 20% der Taktzyklen zugeteilt. Durch die Begrenzung der Shares auf 1000 können jedem *VServer* 1/1000 = 0,1% der Taktzyklen zugesichert werden. Durch einen Schalter kann festgelegt werden, ob die Shares eines *VServers* in Relation zu den vergebenen Shares oder der Gesamtzahl der Shares gesetzt wird. Dies würde in unserem Beispiel bedeuten, dass der *VServer* nicht mehr 20% der Taktzyklen, sondern nur noch 10/1000=1% der Taktzyklen benutzen darf. Auf diese Weise kann man leicht bestimmen, welchen *VServern* der Überschuss an Ressourcen zugeteilt wird und welche nur das garantierte Ressourcenkontingent erhalten.

c. Netzwerk Virtualisierung

Zur Virtualisierung des Netzwerks benutzt *PlanetLab* eine *sichere* Version der *Raw Socket* Schnittstelle, mit der Benutzer IP Pakete verschicken und empfangen können ohne administrative Rechte zu besitzen. Sicher bedeutet in diesem Zusammenhang zwei Dinge: Die Sockets werden an bestimmte TCP- oder UDP- Ports gebunden und es wird sichergestellt, dass nur ein Socket über einen bestimmten Port kommuniziert. Außerdem werden ausgehende Pakete daraufhin überprüft, ob deren Adressen auch zum Socket passen, der an den entsprechenden Port gebunden ist.

Die *sicheren Raw Sockets* werden wie das spezielle Zeitzuteilungsverfahren für *VServers* durch das *SILK-Modul* bereitgestellt.

Die *sicheren Raw Sockets* sind gut geeignet für Netzwerkmessungen und zur Entwicklung von Netzprotokollen. Man kann auch den Netzverkehr eines *Slice* damit überwachen, indem man einen passiven *Sniffer Socket* an einen Port bindet, über den Daten geschickt werden. Kopien dieser Daten werden dann vom *Sniffer Socket* erzeugt und können ausgewertet werden. Ein spezieller *Sniffer Socket* mit administrativen Rechten kann benutzt werden, um die Pakete des gesamten Knotens zu überwachen.

d. Überwachung

Für die Überwachung eines verteilten Systems wie *PlanetLab* sind leistungsfähige Werkzeuge absolut notwendig. Um auch hier dem Prinzip der *entbündelten Verwaltung* gerecht zu werden, stellt *PlanetLab* auf unterer Ebene eine Sensorschnittstelle bereit, die hauptsächlich Daten transferiert, die schon in irgendeiner Form vorliegen, wie z.B. die CPU-Auslastung der einzelnen *VServers*. Diese Daten können nun wieder durch Dienste auf *Slice* Ebene ausgewertet und weiterverarbeitet werden. So ist es auch hier möglich alternative Überwachungsdienste gleichzeitig zu testen.

Es gibt nun verschiedene Sensoren, die gewisse Teilinformationen über das System bereitstellen. Durch die parallel laufenden Überwachungsdienste entsteht hier nun wieder ein Ressourcenkonflikt beim Zugriff auf die Sensoren. Dafür wird ein *Sensor Server* eingesetzt, der die einzelnen Sensoren aggregiert. Will ein Dienst nun auf einen Sensor zugreifen, muss er zuerst eine Anfrage an den *Sensor Server* stellen, der ihn dann mit dem Sensor verbindet.

Zur Kommunikation mit dem *Sensor Server* und den Sensoren wir das HTTP-Protokoll verwendet. Eine Sensoranfrage kann beispielsweise so aussehen:

`http://localhost:33080/nodes/ip/name`

Der *Sensor Server* wird über den Port identifiziert. Der gewünschte Sensor heißt „nodes“. Diese Information ist für den Server bestimmt und leitet den Anfrager weiter zum Sensor. Die Angaben nach der Sensoridentifikation sind für den Sensor bestimmt. Hier wird der Sensor „nodes“ die IP-Adresse (IP) und den DNS-Namen (name) aller registrierten *PlanetLab* Knoten zurückgeben.

4.4.5 Bewertung

Drei Aspekte der Umsetzung des *PlanetLab* Systems sollen nun bewertet werden: Die Skalierbarkeit der *VServers*, die *Slice* Erzeugung und die Dienstinitialisierung.

Die Skalierbarkeit von *VServern* ist abhängig vom Verfügbaren Plattenspeicher und von den Kernelressourcen. Der Plattenspeicher wurde um ca. 95% reduziert. Eine *VServer* Instanz verbraucht durch copy-on-write anstatt 500 MB nur noch ungefähr 30 MB. Dies ermöglicht es, 1000 *VServer* auf einem einzigen Knoten laufen zu lassen. Dieser Skalierungsfaktor lässt sich durch neue Techniken noch weiter verbessern. Eine weitere Beschränkung der Skalierbarkeit stellt die Verfügbarkeit von Kernelressourcen wie z.B. Dateideskriptoren dar. Durch Erweitern der Kernelressourcen kann man diese Probleme kurzfristig beseitigen. Allerdings erfordert

die Nutzung von mehr Ressourcen auch effizientere Kernel Algorithmen. Eine O(n) Suche skaliert bei 1000 *VServern* wahrscheinlich ungenügend.

Die Hauptausrichtung von *PlanetLab* zielt auf langlebige Dienste ab, bei denen die Zeit für eine *Slice* Erzeugung relativ unkritisch ist. Allerdings sollen mit *PlanetLab* auch kurzlebige Dienste unterstützt werden. Für einen schnellen Verbindungsaufbau beispielsweise spielt die Zeit der *Slice* Erzeugung eine entschiedene Rolle. Die meiste Zeit wird hierbei benötigt, um die *VServer* auf den einzelnen Knoten zu initialisieren und dann die gewünschte Software zu laden. Die *VServer* Initialisierung dauert je nach Cacheverhalten zwischen zehn Sekunden und einer Minute.

Wie lange es dauert, einen Dienst in einem betriebsbereiten *Slice* zu starten, zeigt der Beispieldienst Sophia, welcher Software Updates mit dem RPM Mechanismus durchführt. Es dauert ca. zehn Sekunden, um ein „leeres“ Update durchzuführen, das nur den aktuellen Stand erkennt. Müssen Pakete installiert werden beträgt die durchschnittliche Zeit ca. 25 Sekunden pro Knoten. Betrachtet man allerdings die Gesamtdurchschnittszeit des *Slice*, dann dauert es schon fast fünf Minuten ein Update durchzuführen. Diese Zeit wird durch den langsamsten Knoten bestimmt. Hier besteht noch ein Verbesserungsbedarf der Verteilungsmechanismen des Overlays.

4.5 PlanetLab und Globus – ein Vergleich

Globus und *PlanetLab* sind zwei Systeme zur Entwicklung großer verteilter Netzwerksysteme mit verschiedenen Zielsetzungen, jedoch haben sie mit teilweise ähnlichen Problemstellungen zu kämpfen.

PlanetLab will eine Testumgebung für Netzdienste bereitstellen, mit der es möglich ist, einerseits Anwendungs- aber vor allem Infrastrukturdienste zu entwickeln und zu testen. *Globus* dagegen stellt eine allgemeine, standardisierte Software als Basis für verteilte Anwendungen über gemeinsame Ressourcen bereit. Das *Globus Toolkit* ist ein aus bestehenden Technologien zusammengesetzter Werkzeugkasten. *Globus* arbeitet eng mit Standardisierungsorganisationen zusammen, um Standards wie die Open Grid Service Resource Architecture (OGSA) und das Web Service Resource Framework (WSRF) zu etablieren.

Beide Systeme arbeiten auf der gleichen Grundlage, einem geographisch verteilten Netz von Rechnern. Dadurch werden sie bei der Verwaltung der Ressourcen mit ähnlichen Problemen konfrontiert. Sie entwickeln Mechanismen, die ein effizientes Erkennen, Überwachen und Zuordnen von Ressourcen ermöglichen.

4.5.1 Voraussetzungen

Die Benutzergemeinden von *PlanetLab* und *Globus* sind unterschiedlich. Während *PlanetLab* vor allem durch Forscher betrieben wird, die mit Netzdiensten experimentieren, wird *Globus* von Programmierern genutzt, um verteilte Anwendungen für Endbenutzer zu entwickeln. Darüber hinaus besteht die *Globus* Gemeinde nicht nur aus Forschern, sondern hat auch in der Industrie schon viele Benutzer.

Die durch die Zielgruppen bestimmte Funktionalität kann folgendermaßen unterschieden werden:

PlanetLab versucht den Forschern durch eine minimale Beschränkung der Dienstentwicklung durch sein Prinzip der *entbündelten Verwaltung* soviel Freiheit wie möglich zu bieten. Der so entstehende Wettbewerb zwischen entwickelten Netzdiensten bietet deren Benutzern eine breite Auswahl. So können dann Anwendungsprogrammierer über die beste oder passendste Lösung entscheiden. *Globus* dagegen versucht eine standardisierte und funktionsreiche Lösung anzubieten, die den speziellen Anforderungen von Anwendungen gerecht werden und direkt eingesetzt werden

können.

Bei der Betrachtung der Anwendungen, die auf beiden Systemen laufen sollen, kann man zwischen zwei Gruppen unterscheiden:

Es gibt einerseits Gridanwendungen, die sehr rechenintensiv sind und denen es daher vor allem auf die Verfügbarkeit von Rechenleistung auf den einzelnen Knoten ankommt. Dies schließt natürlich nicht aus, dass solche Anwendungen auch viel Netzverkehr verursachen. Bei *PlanetLab* jedoch kann man sagen, dass der Fokus in Bezug auf die Ressourcen eher auf die Netzwerkbandbreite gerichtet ist, da das Testen von Netzdiensten mehr die Bandbreite als die CPU-Auslastung als kritische Ressource ansieht. Dies spiegeln auch die Architekturen der Systeme wieder. Ein durch viele *Slices* belegter *PlanetLab*-Knoten wird bei rechenintensiven Diensten schnell in die Knie gehen. *Globus* dagegen ist eher für rechenintensive Anwendungen ausgelegt als für eine gute Skalierung, der auf einem Knoten laufende virtuellen Organisationen.

Ein großer Unterschied zwischen *PlanetLab* und *Globus* liegt bei der Flexibilität der darunter liegenden Plattform. Da *PlanetLab* hauptsächlich als reale Testumgebung dient, ist die Portierbarkeit auf eine Vielzahl von Systemen zweitrangig. *PlanetLab* läuft in seiner bisherigen Form nur auf intelkompatiblen PCs mit Linux als Betriebssystem. *Globus* dagegen unterstützt alle gängigen Betriebssysteme und eine Vielfalt von Geräten, darunter Rechnerbündel, PCs, PDAs, Dateisysteme, Datenbanken, Sensoren. Diese Geräte können alle zu einer virtuellen Organisation im Sinne von *Globus* semantisch verknüpft werden.

PlanetLab sowie *Globus* laufen auf einer großen Anzahl von Knoten. Die Kontrolle über die Ressourcen der einzelnen Knoten wird zwischen den Konteninhabern und den Systemen geteilt. *Globus* und *PlanetLab* haben auch hier unterschiedliche Ausrichtungen. *PlanetLab* läuft ausschließlich auf dedizierten Maschinen. Die Administratoren von *PlanetLab* haben vollen Zugriff auf die Rechner, im Fall von Linux einen Root Zugang, der ihnen auch erlaubt die Rechner auszuschalten und über Netzwerk wieder zu starten. Diese große Einschränkung der Standortautonomie bedeutet gleichzeitig einen großen Vorteil für *PlanetLab*, da der Forschergemeinde nichts bei der Nutzung der Ressourcen im Weg steht und damit ein schnelles Vorankommen der Experimente gesichert ist.

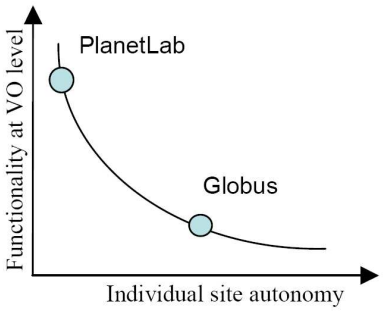

Abbildung 4.3: Funktionalität vs. Standortautonomie

Globus Benutzer dagegen verlieren weniger von ihrer Eigenständigkeit, wenn sie einer virtuellen Organisation beitreten. Bei *Globus* müssen alle Ressourcenzugriffe explizit vom lokalen Administrator gewährt werden, was auch beinhaltet, dass alle privilegierten Dienste unter lokaler Kontrolle sind.

4.5.2 PlanetLab und Globus gemeinsam

PlanetLab und *Globus* sind keine konkurrierenden Systeme. Sie sind eher unterschiedlich aber gleichzeitig kompatibel. Kompatibel in der Hinsicht, dass *PlanetLab* eine Schicht bietet auf der *Globus* installiert werden kann. Die beiden Systeme können durch eine gemeinsame Installation in vielerlei Hinsicht voneinander profitieren. *Globus* Benutzern wird eine Testplattform mit einer starken Infrastruktur geboten, die durch eine Installation von *Globus* auf *PlanetLab* als Dienst in *Globus* bereitgestellt werden kann. Die Gemeinden von *PlanetLab* und *Globus* können ihre Ideen und Erfahrungen austauschen und damit gegenseitig ihre Forschungen vorantreiben. Für *PlanetLab* bietet *Globus* eine Möglichkeit, die entwickelten Netzdienste in einem größeren Kontext zu testen und weiterzuentwickeln.

4.6 MACEDON

4.6.1 Motivation

Da sich das Forschungsgebiet im Bereich Overlays noch im Anfangsstadium befindet, gibt es noch nicht viele Werkzeuge, welche eine einfache Umsetzung von Overlayalgorithmen ermöglichen. Um mit verteilten Algorithmen experimentieren zu können, erarbeiten Forscher viele verschiedene Techniken in den einzelnen Bereichen zum des Entstehungsprozesses: Entwurf, Implementierung, Test und Bewertung. Diese Vorgehensweise ist mühsam und in gewisser Hinsicht uneffektiv, da durch die verschiedenen Techniken ein fairer Vergleich zwischen den Algorithmen schwierig wird. Außerdem wird bei jedem Entstehungsprozess das Rad mehrmals neu erfunden, da viele Schichten implementiert werden müssen, auf denen dann der eigentlich zu testende Algorithmus aufsetzt. Die Arbeit am eigentlichen Problem beschränkt sich dann nur noch auf einen kleinen Teil des Gesamtprojekts, wohingegen die Schaffung der Infrastruktur für die Testumgebung den größten Teil der Arbeit einnimmt. Ist ein Algorithmus dann erfolgreich in einer Umgebung gestestet, soll er natürlich auch gegen Konkurrenten getestet werden, die meistens auf einer anderen Umgebung aufsetzen. Dies macht den Vergleich unfair und verzerrt meistens die Ergebnisse. Also müssen die Implementierungen zwischen den Systemen portiert werden, was wieder erheblichen Aufwand bedeutet. Die Erfahrung zeigt, dass durch diese Hürden das eigentliche Ziel des Testens verschiedener Algorithmen ins Hintertreffen gerät und stattdessen die einzelnen Implementierungslösungen miteinander verglichen werden.

Um diesen Problemen entgegenzuwirken stellt *MACEDON* nun eine Infrastruktur bereit die das Entwickeln, Testen und Bewerten neuer Algorithmen in vielerlei Hinsicht vereinfachen soll:

- Verteilte Algorithmen sollen in einer übersichtlichen und prägnanten domänenabhängigen Sprache spezifiziert werden können.
- Der generierte Code soll in etablierten Testumgebungen und in existierenden Netzwerken ausführbar sein.
- Es soll eine allgemeine Overlay-API bereitgestellt werden, welche die Kompatibilität zwischen Implementierungen von Algorithmen und Anwendungen verbessert.
- Die Bewertung von Experimenten soll fair und konsistent sein.

4.6.2 Overlay Abstraktion

Eine vorteilhafte Repräsentation von verteilten Algorithmen muss zwei Kriterien gleichzeitig erfüllen: Sie muss ausdruckstark genug sein, um die Komplexität verschiedner Protokolle darstellen zu können und gleichzeitig einfach zu benutzen, um die Implementierung der Algorithmen zu erleichtern. *MACEDON* begegnet diesen Anforderungen mit einem Modell, das sich als *ereignisgesteuerten endlichen Zustandsautomaten* beschreiben lässt, mit dem die Forscher ihre Algorithmen spezifizieren können.

Die Zustandsautomaten werden durch Knoten-Zustände, Ereignisse und Aktionen charakterisiert. Jeder Knoten eines *MACEDON*-Overlays wird durch einen lokalen Zustandsautomaten repräsentiert. In den Zustandsvariablen eines Knotens sind verschiedene Informationen gespeichert, die Auskunft über Beziehungen zu anderen Knoten geben, z.B. über die Nachbarknoten, aber auch über die Eigenschaften der eigenen Position des Knotens im Overlay, beispielsweise die Bandbreite der Verbindung zu Nachbarknoten oder Routingtabellen. Die lokalen Knotenzustände bilden zusammen den Systemzustand des Overlaynetzwerks. Die Zustandsübergänge werden durch Ereignisse ausgelöst. Ereignisse können z.B. durch abgelaufene Zeitgeber, Nachrichten von anderen Knoten oder durch Funktionsaufrufe über die Schnittstelle ausgelöst werden. Je nachdem, in welchem Systemzustand sich das Netzwerk befindet, werden durch eintretende Ereignisse verschiedene Zustandsübergänge und Aktionen ausgelöst. Diese Aktionen können das Ändern von Zuständen und Zeitgebern und das Senden von Nachrichten beinhalten.

4.6.3 Architektur

Das *MACEDON* System ist in drei Schichten unterteilt: Die Anwendungsschicht, die Multiprotokollschicht und die Netzträgerschicht. In diesem Protokollstapel werden tiefere Schichten von darüber liegenden Schichten benutzt. Auf der Anwendungsschicht befinden sich die die Implementierungen von Multicast- oder DHT (Distributed Hash Tables)-Anwendungen. Die verschiedenen Overlayprotokolle in der Schicht darunter können wie in der Abbildung dargestellt aufeinander aufsetzen. Diese Schicht wird nach unten mit der Netzträgerschicht verbunden, die ein reales Netzwerk sein kann, das über TCP/IP läuft oder auch ein Testnetz wie der „Network Simulator“ (ns).

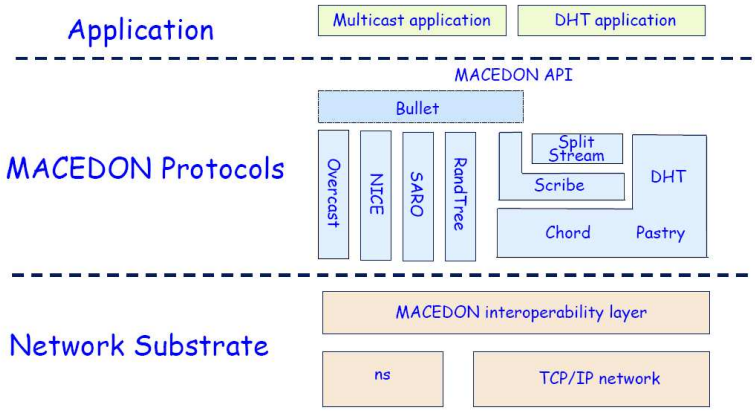

Abbildung 4.4: Abbildung 4: MACEDON Protokollstapel

Für die interne Kommunikation stellt *MACEDON* einen „Up- und Downcall“ Mechanismus bereit, mit dem sich Protokolle verschiedener Schichten gegenseitig

aufrufen können. Aufrufe sind z.B. `forward()`, `deliver()` und `notify()`. Der `notify()` Upcall ermöglicht es beispielsweise unteren Schichten höhere Schichten über eine Änderung in Nachbarlisten zu informieren.

Will man nun mit Hilfe von *MACEDON* einen verteilten Algorithmus implementieren, muss man zuerst eine Protokollspezifikation erstellen, die den Algorithmus im Zustandsmodell von *MACEDON* beschreibt. Diese Protokollspezifikation wird dann von *MACEDON* in ausführbaren C++ Kode übersetzt. Es gibt drei wichtige Headerdateien bei der Protokollspezifikation, die zusätzliche Möglichkeiten zum Testen von Algorithmen bieten. Im Protokollheader wird spezifiziert, welche Protokolle der mittleren Schicht des Protokollstapels verwendet werden. So kann man einen Algorithmus z.B. mit Scribe und darunter liegendem Chord testen oder durch einfaches Ändern der Headerdatei Pastry als Basis für Scribe benutzen. Die Adressheaderdatei bietet die Möglichkeit zwischen IP- oder Hashadressen zu wählen. Durch diese Option könnte man leicht verschiedene Hashverfahren für Adressen testen. In der dritten Headerdatei kann eine vierstufige Ablaufprotokollierung (Trace) aktiviert werden.

4.6.4 Bewertung

Die Untersuchungen von *MACEDON* zeigen Stärken des Systems in dreierlei Hinsicht. Es ist gelungen ein System zu entwickeln, das eine starke Vereinfachung des Entstehungsprozesses sowohl beim Entwurf, bei der Implementierung als auch bei der Bewertung von verteilten Algorithmen bietet.

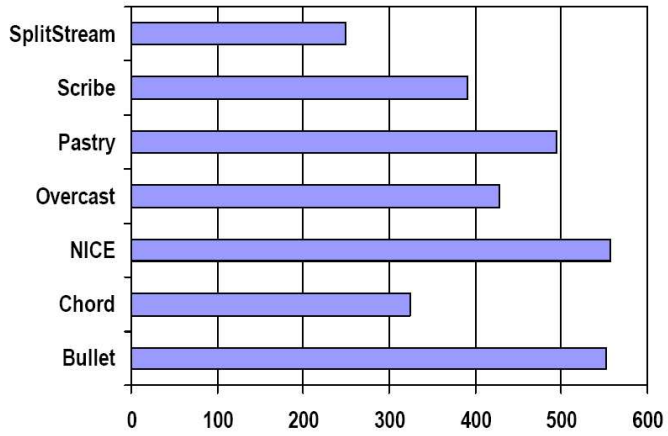

Abbildung 4.5: Anzahl der Kodezeilen von MACEDON Spezifikationen verschiedener Algorithmen

Abbildung 5 zeigt, wie viel Zeilen Kode die Spezifikation das jeweilige Protokoll hat. Die Implementierung von NICE beispielsweise erzeugt aus den ca. 500 Zeilen Spezifikation 2500 Zeilen C++ Code. Mit dem *MACEDON* Betriebssystem zusammen sind es sogar 6000 Zeilen.

In Bezug auf die Skalierbarkeit und Leistungsfähigkeit hat sich *MACEDON* ebenfalls bewährt. Ein Experiment, bei dem eine *MACEDON* Pastry Implementierung mit einer FreePastry Java Implementierung verglichen wurde, zeigte Vorteile von *MACEDON* in beiden Belangen. Während FreePastry maximal zwei Instanzen pro Knoten zuließ konnten 20 *MACEDON* Instanzen auf einem Rechner gleichzeitig laufen. Auch die Latenzzeit der *MACEDON* Implementierung war um 80% geringer als die von FreePastry.

Dadurch, dass *MACEDON* eine Schnittstelle zur TCP/IP- und ns-Schicht bietet, ist dessen System leicht auf andere internetweite Overlaysysteme portierbar. So ist es möglich die spezifizierten Algorithmen auf jeder *MACEDON* Installation zu testen. Beispielsweise wurde auf der *PlanetLab* Plattform eine *MACEDON* Installation mit 50 Knoten erfolgreich eingerichtet.

4.7 Zusammenfassung

PlanetLab und *MACEDON* sind zwei aktuelle Forschungsprojekte im Bereich der Overlaynetzwerke, die beide noch sehr jung sind. Dadurch unterliegen sie noch einer starken Veränderung und ständiger Weiterentwicklung. Trotzdem haben sich die Ausrichtungen der Systeme schon gefestigt. *PlanetLab* stellt durch seine Konzepte ein System zur Infrastrukturentwicklung bereit, während sich *MACEDON* auf die Vereinfachung des Entstehungsprozesses von verteilten Algorithmen konzentriert. Beide haben zum Ziel, die Vergleichbarkeit und damit den Wettbewerb von Entwicklungen zu fördern. Im Fall von *PlanetLab* sind es die Infrastrukturdienste und bei *MACEDON* die verteilten Algorithmen.

Die Entwicklung der Projekte deutet auf eine immer stärkere Integration der Systeme hin. *MACEDON* hat schon eine Installation auf *PlanetLab* laufen. Auch *Globus* und *PlanetLab* streben eine engere Zusammenarbeit an. Dies lässt auch vermuten, dass die Forschergemeinden von Overlaynetzen mit der Zeit immer weiter zusammenwachsen werden.

Literaturverzeichnis

[BBC+04] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating system support for planetary-scale network services. In *In Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (March 2004)*, March 2004.

[RBC+04] Matei Ripeanu, Mic Bowman, Jeffrey S. Chase, Ian Foster, and Milan Milenkovic. Globus and planetlab resource management solutions compared. In *Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-13), Honolulu, Hawaii, June 2004*, February 2004.

[RKB+04] Adolfo Rodriguez, Charles Killian, Sooraj Bhat, Dejan Kostic, and Amin Vahdat. Macedon: Methodology for automatically creating, evaluating, and designing overlay networks. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004), March 2004*, March 2004.

Kapitel 5

Grid Measurements

Seminarbeitrag von **Johannes Lieder**

5.1 Einführung

Das Internet unterliegt seit geraumer Zeit einem starken Wandel, der Anlass dazu gibt, die bisher anerkannten Modelle über die Zusammensetzung von Nutzdaten in zentralen Netzwerkknoten und Spitzenbelastungen der Netzwerkinfrastruktur neu zu bewerten und eventuelle neue Erkenntnisse in die Überlegungen einzubeziehen.

Neben den klassischen Diensten, wie E-Mail und HTTP (*world wide web*) besteht für den Internet-Anwender durch neuerliche Entwicklungen auch der Bedarf an Austauschmöglichkeiten für größere Datenmengen, in Form von verschiedensten Multimediainhalten und Binärdaten. Dies stellt eine grundlegende Veränderung im Belastungsprofil der vorhandenen Netze dar, da typische Dateigrößen für diese Objekte nicht wie in der Vergangenheit im Bereich von einigen Kilobyte liegen, sondern sie um bis zu 5 Größenordnungen übertreffen (Megabyte oder gar Gigabyte).

Erst durch das Vorhandensein eines globalen Datennetzes ergeben sich Möglichkeiten verteilte Computerressourcen gemeinsam zu nutzen und somit Aufgaben zu bewältigen, die mit konservativen Systemen nicht realisierbar wären. Ein Modellparadigma, welches diese Gegebenheiten nutzt, ist das Konzept des Grid Computing, bei dem vorhandene Rechenleistung einer großen Anzahl dezentral verteilter Knoten für eine gemeinsame Aufgabe genutzt wird. In Anlehnung an ein Power Grid (das Stromversorgungsnetz eines Landes), in dem Energie omnipräsent und ununterbrochen zur Verfügung gestellt wird, hat das Grid Computing seine Bezeichnung erhalten. Eine zentrale Eigenschaft des Grids ist die zentrale Steuerung, die dafür sorgt, dass eine koordinierte Nutzung der Ressourcen für eine gemeinsame Aufgabe gewährleistet ist.

Ein sehr populäres und anschauliches Beispiel für solch ein Szenario ist das Projekt SETI@home welches einen Screensaver als Client für Heimrechner zur Verfügung

stellt. Vom Gast-Betriebssystem ungenutzte CPU-Zyklen werden für die eigenen Berechnungen verwendet und somit wird allein durch die immense Zahl an teilnehmenden Knoten eine Rechenleistung gebündelt, die selbst dedizierte Supercomputer bei weitem übertrifft.

Neben dem Konzept des Grid Computing, welches eher für wissenschaftliche Projekte und Forschung von Bedeutung ist, haben sich im alltäglichen Umgang mit dem Internet jedoch mehr sog. Peer-to-Peer Systeme (P2P-Systeme) etabliert, die eine abgewandelte Form des Grids darstellen. Peer-to-Peer Systeme weisen mehrere signifikante Unterschiede zum klassischen Grid auf, da bewusst die typischen Client- und Server-Rollen der teilnehmenden Rechner aufgegeben werden und daraus gleichberechtigte Peers entstehen (Symmetrie). Obwohl in einem reinen P2P-Netzwerk die Knoten laut Definition eine homogene Menge von Peers darstellen, kommen diese Systeme in der praktischen Realisierung jedoch meist nicht ohne eine gewisse zentrale Infrastruktur aus.

Die fortschreitende Breitbandvernetzung einer großen Anzahl von Haushalten zusammen mit den genannten neuen Anwendungsgebieten, erfordern eine genaue Untersuchung und Charakterisierung der Zusammensetzung des Netzwerkverkehrs und der Architekturen von P2P-Systemen - zusammengefasst unter dem Begriff *Grid Measurements* - um künftig für die wachsenden Anforderungen an das globale Datennetz gerüstet zu sein. Schon heute stellt die Übertragung von Multimedia-Daten einen größeren Anteil des Gesamttraffics von ISPs (Internet Service Provider) und Universitätsnetzen dar, als der Web-Traffic, der über das HTT-Protokoll transportiert wird. Diese Tatsache rechtfertigt so den Schwerpunkt, der mit dieser Arbeit und Grid Measurements im Allgemeinen gemacht wird, um zukünftige Grid- und P2P-Systeme noch leistungsfähiger entwerfen zu können.

5.2 Peer-to-Peer File Sharing

5.2.1 Peer-to-Peer Architekturen

Peer-to-Peer Architekturen sind durch ihre Maxime, Ressourcen der individuellen Peers einer höheren Aufgabe oder wieder der Allgemeinheit zukommen zu lassen, besonders geeignet die Infrastruktur für einen Datenaustausch zwischen einer großen Zahl von verteilten Peers zur Verfügung zu stellen. Das Ziel eines File Sharing-Netzwerks ist es also die Örtlichkeit und den Austausch von Dateien für eine große Gruppe von Benutzern über das Internet zu fördern und zu kumulieren.

Zu den bekanntesten P2P File Sharing Diensten zählen Napster, Kazaa und Gnutella. Einer der neueren Vertreter dieser Systeme ist BitTorrent, welches aufgrund seiner besonderen Konzeption an verschiedenen Stellen eine Sonderrolle einnimmt. Keiner der genannten Dienste verwirklicht dabei ein einzelnes abgeschlossenes P2P-Konzept, sondern stellt jeweils eine Mischform verschiedener Paradigmen dar.

Obwohl eine Peer-to-Peer Architektur streng genommen keine expliziten Serverknoten vorsieht, wird das File Sharing-Netzwerk Napster unter anderem durch einen zentralen Server-Cluster als Anlaufpunkt für die Peers realisiert. Es ergibt sich eine vergleichsweise einfache Netzwerktopologie (s. Abbildung 5.2.1), bei dem die Clients sternförmig mit dem Cluster verbunden sind. Ein Cluster bietet hierbei die Möglichkeit der Lastverteilung (*load balancing*), da möglicherweise eine sehr große Anzahl von Peers an dem P2P-Netzwerk teilnehmen und somit an diese zentrale Komponente hohe Ansprüche in Bezug auf Verfügbarkeit gestellt werden müssen (*single point of failure*). Andererseits ist es für einen Client vergleichsweise einfach an dem P2P-Netzwerk teilzunehmen. Nach dem Aufbau einer Verbindung zum zentralen Napster-Cluster werden die Liste der freigegebenen lokalen Dateien und die IP-Adresse hochgeladen und der Benutzer kann eine Suchanfrage an den zentralen

Verzeichnisdienst stellen. Der Server gibt Daten über eine Menge von Peers zurück, die möglicherweise über die gesuchten Daten verfügen und mit Hilfe einer Bewertung der Verbindungsgüte kann der P2P-Client die bestmögliche Transferquelle ermitteln. Somit verwirklicht also nur noch das letzte Stadium, der Datentransfer, ein richtiges Peer-to-Peer Konzept, da hier die Kommunikationspartner gleichberechtigt als Client oder Server fungieren können.

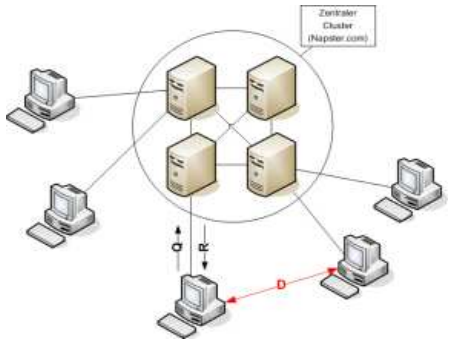

Abbildung 5.1: Napster Peer-to-Peer Architektur

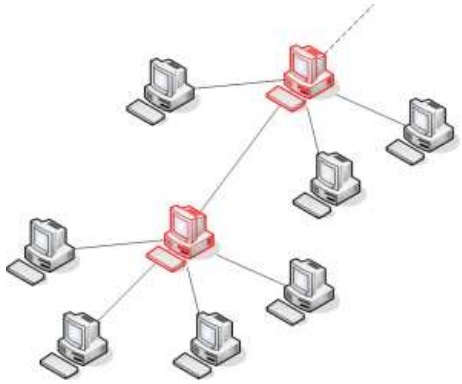

Abbildung 5.2: Kazaa Peer-Hierarchie

Eine Weiterentwicklung der Napster-Architektur ist das Kazaa P2P-Netzwerk, welches das Konzept der sog. Supernodes einführt. Peers sind also entweder Supernodes oder sind mit einem Supernode verbunden, die in gewisser Weise die Rolle des zentralen Napster-Clusters übernehmen und eine Dateiliste ihrer untergeordneten Peers unterhalten. Ein Vorteil dieser hierarchischen Struktur (Abbildung 5.1) ist die Dezentralisierung der Verwaltungsinstanzen und eine gleichzeitige Erhöhung der Ausfallsicherheit und Lastverteilung. Im Gegenzug erfordert dies vom Client einen größeren Aufwand beim Verbindungsaufbau, denn es muss zu Beginn erst ein geeigneter Superknoten gefunden werden. Weiterhin muss das Netzwerk infrastrukturelle Mechanismen zur Differenzierung der Peers im Hinblick auf Bandbreite und Aufrechterhaltung der Kommunikation beim Wegfall eines solchen Knotens zur Verfügung stellen. Auch Suchanfragen müssen nun gegebenenfalls an übergeordnete Superknoten in der Hierarchie weitergegeben werden, falls der erste Superknoten keine Resultate zurückliefern konnte. Wurden eine genügende Anzahl an Ergebnisse zurück zum anfragenden Client propagiert, so kann wie auch beim Napster-Netzwerk, schließlich eine direkte Verbindung zum entsprechenden Peer hergestellt werden.

Das Gnutella-Netzwerk als dritte P2P-Variante verfolgt wiederum ein reines

Peer-to-Peer Konzept (Abbildung 5.3) und ist daher nicht auf zentrale Rechnerknoten angewiesen. Bis auf einige veröffentlichte bekannte Knoten (sog. *well-known hosts*) ermitteln die Clients bei der Verbindungsaufnahme mit dem Netzwerk völlig autark benachbarte Peers mit Hilfe von Ping/Pong-Nachrichten. Das Gnutella-Protokoll sieht dabei vor, dass empfangene Ping-Nachrichten mit einem Pong-Paket beantwortet werden und weitere Ping-Pakete an die, dem jeweiligen Host noch zusätzlich bekannten Peers, abgesetzt werden. Zusammen mit einer vorgegebenen maximalen Lebensdauer (*hop count*) eines solchen Ping-Pakets entsteht ein Flooding-Mechanismus, der dynamisch – abhängig vom jeweiligen Zustand des Netzwerks – eine Discovery- und gleichzeitig eine verteilte Such-Infrastruktur zu induzieren vermag. Das Gnutella-Netzwerk kommt somit durch seine fast reine Peer-to-Peer Overlay-Struktur dem Ideal des völlig homogenen Peer-Netzwerks am nächsten.

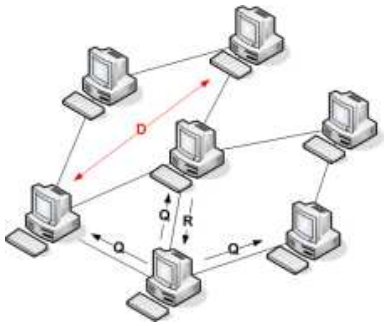

Abbildung 5.3: Gnutella Overlay-Netzwerk

Wie bereits anfänglich erwähnt, nimmt der File Sharing-Dienst BitTorrent in dieser Kategorisierung (vom zentralisierten P2P-File Sharing hin zu reinen P2P-Konzepten) eine Sonderstellung ein, da in dieser Architektur keine richtige Suchfunktion auf Datei- oder Verzeichnisebene, wie sonst üblich, vorgesehen ist. Stattdessen werden Downloads eindeutig über korrespondierende `.torrent`-Dateien (*meta data files*) identifiziert, die auf verschiedensten Web-Servern (*directories*) im Internet zugänglich abgelegt sind.

Ein exemplarischer Ausschnitt der Download-Liste des bekanntesten Directory Servers *www.suprnova.org* ist in Abbildung 5.4 zu sehen. Ein Objekt wird nicht wie bei den bisher vorgestellten File Sharing-Systemen einfach durch die Teilnahme eines Peers, der im Besitz der Datei ist, am Netzwerk hinzugefügt, sondern ein Benutzer muss das Objekt „explizit“ hochladen (die sog. *seed*). Besonders ausgezeichnete Benutzer (Moderatoren) überprüfen alle neuen Inhalte und veröffentlichen sie gegebenenfalls im Directory. Somit wird verhindert, dass falsches Material Einzug in das System erhält und die Pollution Rate kann, im Gegensatz zu anderen File Sharing-Diensten, auf einem sehr geringen Niveau gehalten werden. Neben diesem Konzept eines moderierten File Sharing Dienstes (*P2P moderation system*) verwirklicht BitTorrent weiterhin ein ausgeklügeltes Handelssystem von Dateiblöcken (*chunks*), so dass immer ein gerechtes Verhältnis zwischen Downloads und Uploads gewährleistet ist (*anti-leeching*).

Damit die Objekte jederzeit lokalisierbar sind, führen sog. Tracker Buch darüber welche Peers über eine vollständige Kopie einer Datei verfügen. Der zuständige Tracker wird in der `.torrent`-Datei vermerkt, so dass allein durch die Metadaten eine Verbindung zwischen anfragendem und bedienendem Peer zustande kommen kann.

Die BitTorrent-Philosophie garantiert also eine gerecht verteilte Nutzung der Ressourcen (Handels-Prinzip) und die Authentizität der Inhalte; auch bietet BitTorrent die Unabhängigkeit von einem zentralen Suchdienst, da die Metadaten alle

11:05	Reno 911 02x03 (XviD-LOL)	175 Mb	23	40	TV-rip	Resistance	link
10:38	The Weakest Link: - Star Trek Edition	78 Mb	1	5	TV-rip	riccoman	link
10:17	Reno 911 02x03 (XviD-LOL)	175 Mb	23	38	TV-rip	thegibson	link
10:01	Twin Peaks Season 2 Episode 3	224 Mb	2	2	TV-rip	smiler64	link
09:00	Euro 2004 Matchday 12 Holland Latvia...	700 Mb	1	56	TV-rip	maxxed	link
08:57	Aqua Teen Hunger Force - Season 3, D...	698 Mb	1	0	DVD-rip	cubbs	-
08:34	Euro 2004 Nederland-Letland 23 Juni ...	702 Mb	1	188	TV-rip	Arbor	link
08:30	The Simple Life 02x03 (PDTV-LOL)	174 Mb	112	425	TV-rip	Resistance	link
08:09	Newlyweds Nick and Jessica 03x02 (Xv...	175 Mb	67	309	TV-rip	Resistance	link

Abbildung 5.4: Aktuelle BitTorrent Downloads auf suprnova.org

wichtigen Informationen beinhalten. Zusammen bilden BitTorrent und Suprnova eine einzigartige Infrastruktur (daher auch oft die Bezeichnung BT/Suprnova) aus Verzeichnisdienst und Peer-to-Peer Netzwerk.

Bei der Betrachtung von Peer-to-Peer File Sharing Architekturen muss zwischen den beiden Teildiensten des Suchens und der anschließenden Datenübertragung getrennt werden. Zwar hat die Strategie nach der die Daten im weiteren Verlauf der Transaktion übertragen werden auch Auswirkungen auf die Effizienz eines File Sharing-Netzwerks, jedoch wird dies (mit Ausnahme von BitTorrent) bei den meisten Architekturen in ähnlicher Weise verwirklicht (konservative Client/Server-Verbindung zwischen den beiden Peers). Die ausschlaggebenden Unterschiede sind bei den klassischen File Sharing-Diensten in der P2P-Netzwerkarchitektur, die sie bilden – und somit auch in gleichem Maße in der Suchfunktionalität – zu finden. Im Gegensatz zu der zentral orientierten Struktur Napsters und der rein dezentral verteilten Struktur Gnutellas, stellt das Kazaa-Netzwerk einen Kompromiss dieser beiden Konzepte dar.

Obwohl Napster als Dienst in der Vergangenheit sehr erfolgreich war, widerspricht seine Architektur in gewissen Punkten dem Peer-to-Peer Konzept und bietet dadurch einen verwundbaren Punkt. Zwar ist es unwahrscheinlich, dass ein zentraler Cluster mit einer großen Anzahl von Knoten ausfällt, jedoch kann ein solches System prinzipiell schnell demontiert werden – ein sog. ,,pull the plug“ wäre also möglich.

Angesichts der Tatsache, dass theoretisch *mehrere* Gnutella-Netzwerke nebeneinander existieren können, wäre in diesem Fall ein langfristiger Fortbestand ohne Einschränkungen denkbar, da es hier keine zentralen Instanzen gibt, die mit einem Schlag entfernt werden könnten.

Die Gemeinsamkeiten der vorgestellten Systeme liegen also hauptsächlich im Objekttransfer, der strengen P2P-Konzepten folgt. Wurde ein passender Peer gefunden, der die gesuchten Daten bereitstellt, so wird schließlich der weitere Datenaustausch weitgehend einheitlich abgewickelt; d.h. die Verwaltung (das Einfügen von neuen Knoten und Objekten in das System) stellt einen der wichtigsten Dienste eines P2P-Systems dar. Folglich ist die Betrachtung eben dieser systemimmanenten Vorgänge eine der Kernaufgaben bei der Analyse und dem Verständnis von Peer-to-Peer Systemen.

5.2.2 Sammlung von Daten

Um die Charakteristika der verschiedenen Peer-to-Peer Architekturen untersuchen zu können, muss der Sprung von theoretischen Überlegungen zu praktischen Datenerfassungen gemacht werden. Alle hier näher betrachteten Dienste basieren auf dem Internet und ihren Technologien. Sie nutzen deshalb die nach dem ISO/OSI Referenzmodell vorgegeben Protokolle TCP/IP (Schicht 4, Transportschicht) um Ende-zu-Ende Verbindungen zwischen den Peers bzw. zum Server hin herstellen zu können. Dabei verwenden sie u.a. auf der Anwendungsschicht (Schicht 7) auch Stan-

```
Host: 207.46.156.188:1214
UserAgent: KazaaClient May 28 2002 00:23:52
X-Kazaa-Username: nikm
X-Kazaa-Network: KaZaA
X-Kazaa-IP: 193.45.14.143:1214
X-Kazaa-SupernodeIP: 64.40.102.44
```

Abbildung 5.5: Kazaa Protocol Query Header

dardprotokolle (z.B. HTTP) bzw. Protokolle mit eigenen Erweiterungen [Zah03]. Es liegt daher nahe die P2P-Mechanismen auf dieser oder einer der darunter liegenden Schichten zu untersuchen.

Abgeschlossene, unveränderliche Netze bieten hier im besonderen die Möglichkeit einer genauen Untersuchung durch die Aufzeichnung von Traffic-Daten an zentralen Netzwerk-Knoten (z.B. Subnetz-zu-Internet Gateways), wodurch die Erfassung der benötigten Daten vergleichsweise einfach zu bewerkstelligen ist. Hierbei bietet sich das Logging von Protokolldaten auf der Vermittlungsschicht (Schicht 3) als Vorgehensweise besonders an, da sie vollkommen transparent für die höheren Protokolle – und somit auch für den Benutzer – erscheint. Bei dieser passiven Messmethode ist jedoch zu beachten, dass bei einer begrenzten Zahl von teilnehmenden Client-Rechnern in einem Universitätsnetzwerk, oder einen Netzwerk einer vergleichbaren Einrichtung, nicht zwangsläufig repräsentative Daten zu erwarten sind.

Ist die Möglichkeit des Zugangs zu einer solchen zentralen Netzwerkkomponente nicht möglich, so muss mit Hilfe der (mitunter im Umfang sehr unterschiedlichen) Protokollfunktionalität eines P2P-Systems versucht werden, eine automatisierte Akquisition von relevanten Daten zu erreichen. Da die Bereitstellung zentraler Performance-Daten in diesen File Sharing-Protokollen nicht vorgesehen ist, wird das Protokoll mitunter für die Dauer einer Datenerfassung zweckentfremdet eingesetzt (*exploit*).

In der Praxis werden für diese File Sharing-Dienste meist recht einfache und unverschlüsselte Protokolle benutzt – als Basis dient hier bei den Diensten Napster, Kazaa und Gnutella das HyperText Transfer Protocol (HTTP). Im Falle von Kazaa werden beispielsweise zusätzliche Daten zum Protokoll-Header hinzugefügt (siehe Listing 5.5). Eingeordnet in die Schichtenhierarchie stellt diese zusätzliche Information somit eine Art SDU (*service data unit*) des Kazaa-Protokolls im darunterliegenden HTTP-Request dar (Kapselung).

Da die für eine erfolgreiche Datenübertragung nötigen `.torrent`-Dateien bei BitTorrent auf ganz gewöhnlichen Web-Servern abgelegt sind, setzt dieses System somit teilweise bestehende Internet-Technologien ein; de letztliche Handel mit Daten baut jedoch direkt auf der Transportschicht (TCP/IP) auf, um diese Mechanismen möglichst optimal (Vermeidung von Protokoll-Overhead) implementieren zu können.

Bei der Messung an der University of Washington [Zah03] wurde eine Auswertung des Netzwerkverkehrs an der Grenze zwischen dem Campusnetz und Internet, also eine passive Messung, über 200 Tage hinweg durchgeführt. Der beteiligte Benutzerkreis umfasste dabei ca. 60,000 Studenten, Mitarbeiter und sonstige Universitätsanghörige – im gemessenen Zeitraum kamen so insgesamt 20 Terabyte an eingehenden Daten zusammen. In der vorliegenden Messung wurden allerdings nur die ausgehenden Verbindungen weitergehend untersucht, also Verbindungen von universitätsinternen Rechnern zu universitätsexternen Peers.

Für die Untersuchungen an den verbleibenden File Sharing-Netzwerken Napster, Gnutella und BitTorrent wurden jeweils eigene Skripte und Programme entworfen,

um mit Hilfe von Protokoll-Requests vergleichbare Daten, wie in [Zah03] zu erhalten. Da der Zugriff auf die von Napster verwalteten Indices von außen nicht möglich ist und um eine Liste einer ausreichenden Zahl von Peers aufbauen zu können, stellte in [Gri02] ein „Crawler“ Suchanfragen nach populären Objekten an die Napster-Server.

Beim Measurement von Gnutella wurde, um vergleichbare Resultate erzielen zu können, der Ping/Pong-Mechanismus (Kapitel 5.2.1) des Systems (hohe TTLs) benutzt, so dass jeweils neue Pong-Pakete eines bis dahin unbekannten Peers registriert werden und die aufgestellte Peer-Liste so konstant aktualisiert wird. In den Pong-Antworten werden zudem automatisch Daten über Anzahl und Gesamtgröße der freigegebenen Dateien, die zur Aufstellung von Statistiken dienen können, mitgeliefert.

5.3 Untersuchung von Peer-to-Peer Systemen

5.3.1 Vorüberlegungen

Um ein tieferes Verständnis über die Eigenschaften von Peer-to-Peer Systemen neben den bisher angestellten theoretischen Überlegungen zu erlangen, müssen nun die durch die Auswertung ermittelten Daten untersucht werden. Dabei ist es wichtig schon vor der eigentlichen Analyse vermutlich wichtige Systembeziehungen zu identifizieren, um so später durch die Messdaten auf weitere Zusammenhänge schließen zu können.

Es soll nun also eine Liste von fünf wichtigen Charakteristika (vgl. Tabelle 5.3.1) aufgestellt werden, die mit Hilfe der Messdaten am konkreten Beispiel beobachtet werden können. Anhand dieser Daten können dann möglicherweise die „treibenden Kräfte“ bestimmt werden, die das System beeinflussen bzw. die Mechanismen entdeckt werden, die das System steuern.

Die erste wichtige Eigenschaft eines verteilten Systems – und somit auch eines Peer-to-Peer Systems – ist die Verfügbarkeit (Tabelle 5.3.1.1) des Gesamtsystems. Diese Größe wird wiederum direkt bedingt durch die Verfügbarkeit zentraler Komponenten und den Grad der Verteilung dieser zentralen Infrastruktur. Wie bereits in Kapitel 5.2.1 erwähnt, wäre ein ideales P2P-Netzwerk vollständig verteilt, jedoch ist dieser Ansatz im praktischen Anwendungsfall sehr schwer zu realisieren. Somit hat die mehr oder weniger ideale Architektur der zentralen Komponenten eine direkte Auswirkung auf Güte des gesamten Dienstes, woraus ggf. die Verlässlichkeit und die Flexibilität des Systems folgen. Im Falle eines fast vollständig verteilten P2P-Systems, rücken die Ansprüche an zentrale Netzwerkknoten mehr in den Hintergrund; hier werden den Peers infrastrukturelle Dienste durch Ressourcen der Allgemeinheit zur Verfügung gestellt, so dass sich die Qualität dieser Dienste für gewöhnlich notorisch auf einem niedrigen Niveau befindet. Also kann wiederum auch, je nach betrachteter Netzwerk-Architektur, die Verfügbarkeit der einzelnen Peers (in ihrer Gesamtheit betrachtet) direkte Auswirkungen auf unsere Analyse haben.

Eine weitere wichtige Eigenschaft (insbesondere für ein File Sharing-Netzwerk) sind die allgemeinen Download-Charakteristiken (Tabelle 5.3.1.2), wie z.B. die Download-Performance und die Gesamtdauer eines erfolgreichen Downloads, von der ersten Übertragung von Nutzdaten bis zum Abschluss der Anfrage. Auf natürliche Art und Weise stehen diese Größen im Zusammenhang mit den Eigenschaften der beteiligten Peers – der dritten großen Klasse von Eigenschaften (Tabelle 5.3.1.3). Abgesehen von der eigenen Download-Bandbreite (Downstream) hängt die Download-Performance von der oberen Grenze des Upstreams der Server-Peers (*upload/download bottleneck*) und den Latenzen der involvierten Netzwerkinfra-

struktur (*network latencies*) ab.

Neben den Auswirkungen auf das Peer-to-Peer System selbst, tendieren die erwähnten Download-Charakteristika dazu auch wichtige Faktoren für die Personen, die das System benutzen, zu sein – sie wirken sich daher auch auf die Benutzer-Charakteristiken (*user characteristics*) aus, die die nächste zu untersuchende Gruppe von Eigenschaften darstellt (Tabelle 5.3.1.4).

Als letzte Kategorie verbleibt nun noch eine bestimmende Größe. Es ist nahe liegend, dass die auszutauschenden Objekte, die die Grundlage des File Sharing-Systems ausmachen, einen großen Einfluss auf das Verhalten und die Leistungsfähigkeit haben. Hier gilt es genau zu untersuchen, in welchen Größenordnungen sich die Anzahl der Dateien im System und auf den einzelnen Peers befinden und wie die Häufigkeit eines Dateityps verteilt ist bzw. zwischen welchen verschiedenen Dateitypen überhaupt unterschieden werden kann. Auch die Dynamik der Objekte, die Beliebtheit (eine sehr dynamische Größe, die in einer reinen Abhängigkeit zum Benutzer/Client und seinen Eigentümlichkeiten steht) und die Lebensdauer von Inhalten sind nicht zu unterschätzende Faktoren in den komplexen Vorgängen, die die zu betrachtenden verteilten Netzwerke ausmachen und antreiben.

1. **availability**
2. **download characteristics**
3. **properties of participating peers**
4. **user/client characteristics**
5. **object characteristics**

Zusammen geben diese Punkte aus Tabelle 5.3.1 eine gewisse Auskunft über die Qualität eines File Sharing-Dienstes und ermöglichen somit auch eventuelle Rückschlüsse auf die zukünftige Popularität einer bestimmten Peer-to-Peer Architektur. Eine endgültige Einschätzung der gesammelten Daten ist jedoch erst nach dem Entwurf und der Verifikation eines adäquaten Modells in den folgenden Abschnitten möglich.

5.3.2 Verfügbarkeit

Die fortgeschrittene Architektur des Kazaa File Sharing-Netzwerks mit seinen Supernodes (Kapitel 5.2.1) erlaubt eine sehr gute Skalierbarkeit dieses Systems. Andere Systeme versuchen ihre Verfügbarkeit durch eine teilweise oder vollständige Verteilung (vgl. Gnutella) sicher zu stellen. Die zentrale Verzeichnis-Infrastruktur von BitTorrent mit seiner Internet-Seite *www.suprnova.org* kann hier keine Skalierbarkeit gewährleisten und ist somit eine Schwachstelle (*weak point*) dieser Architektur.

Wie bereits im vorigen Abschnitt begründet wurde, ist für eine Untersuchung der Verfügbarkeit das Gnutella-Netzwerks durch seine verteilte Natur eine Betrachtung der absoluten Anzahl der teilnehmenden Peers über die Zeit von Interesse. Durch die Verwendung des „Ping/Pong-Exploits“ konnten pro diskreter Messung in [Gri02] zwischen 8,000 und 10,000 verschiedene Peers ermittelt werden, was eine Erfassung von bis zur Hälfte der Gesamtpopulation von Gnutella darstellt.

Leider liegt in der vorliegenden Untersuchung nur eine Statistik über einen relativ kurzen Zeitraum von einer Woche vor, so dass ein direkter Vergleich mit weiteren Daten z.B. des BitTorrent-Dienstes nicht möglich ist. Durch die in Kapitel 5.3.1 angestellten Vorüberlegungen, ist jedoch anzunehmen, dass es bis auf saisonale und tageszeitabhängige Schwankungen keine signifikanten Einbrüche in der Population des Gnutella-Netzwerks gibt. Dies steht im Gegensatz zu den Beobachtungen in [Sip04] (Abbildung 5.7). Wie bereits vermutet, zeigt sich eine starke Abhängigkeit von BitTorrent zu seinem Verzeichnisdienst Suprnova und den „Trackern“, denen auch eine zentrale Bedeutung zukommt.

Die fehlende Verfügbarkeit dieses Portals und der gleichzeitige Ausfall der HTML Mirrors kann einen Zusammenbruch und das schrittweise Erliegen des gesamten Systems (erster rot bzw. grau schattierter Bereich in Abbildung 5.7) nach sich ziehen. Obwohl sich hier ganz eindeutig eine Schwachstelle des BitTorrent-Systems aufzeigt, bedeutet ein Versagen des Verzeichnisdienstes alleine nicht den Ausfall des gesamten Peer-to-Peer Netzwerks (siehe Abbildung 5.7). Wurden vom Benutzer bereits zu einem früheren Zeitpunkt die Metadaten einer `.torrent`-Datei auf den lokalen Computer heruntergeladen, so ist es immer noch möglich mit Hilfe dieser Daten den zuständigen Tracker ausfindig zu machen und so am File Sharing teilzunehmen. Der Ausfall eines großen Trackers, wie z.B. von *beowulf.mobilefrenzy.com* am 25.12.2003 (zweiter schattierter Bereich in Abbildung 5.7) hat im Gegensatz dazu jedoch eine drastische Reduktion der zur Verfügung stehenden Downloads zur Folge – in diesem Fall einen relativen Verlust von 40 [Sip04].

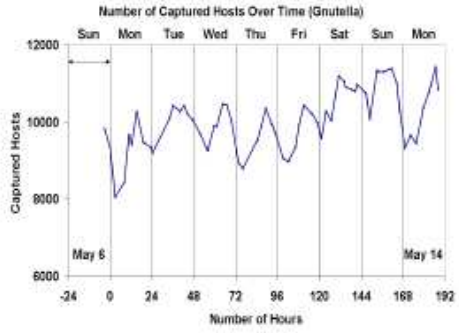

Abbildung 5.6: Verlauf der Peer-Population im Zeitraum einer Woche (Gnutella)

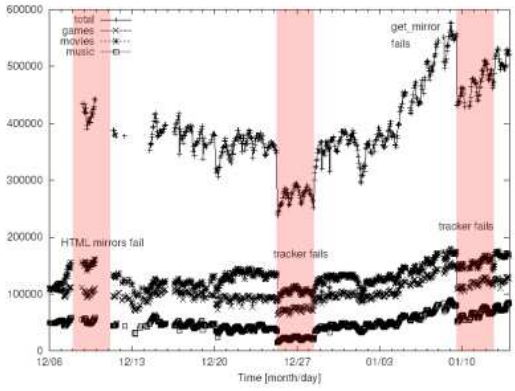

Abbildung 5.7: Einbrüche in der Zahl der Downloads beim Versagen zentraler Komponenten (BitTorrent)

Es ist weiterhin anzumerken, dass es hier (unter Last) innerhalb nur eines Monats zu mehreren Ausfällen kam, was eine gravierende Einschränkung der Benutzbarkeit und Verfügbarkeit dieses Dienstes darstellt. Die lückenhaften Aufzeichnungen in Abbildung 5.7 kommen dabei jedoch nur durch die Eigentümlichkeit des zur Messung benutzten Skripts zustande, da es an die zentralen `.torrent` Server und Mirrors in regelmäßigen Abständen Anfragen stellt, um eine Liste der verfügbaren Tracker aufstellen zu können. Stehen diese Dienste nicht mehr zur Verfügung, so können keine neuen Messdaten mehr gesammelt werden, obwohl das Peer-to-Peer Netzwerk immer noch funktionstüchtig ist.

5.3.3 Download Charakteristika

Abbildung 5.8 sowie eine Vielzahl der folgenden Diagramme sind als sogenannte kumulative Verteilungsfunktionen (CDF, *cumulative distribution function*) dargestellt. Die zu untersuchende Größe wird dabei auf die Abszisse aufgetragen; der entsprechende Ordinatenwert (zwischen 0 und 1 bzw. zwischen 0% und 100%) gibt den Anteil der Ausgangsmenge an, für den der Ausganswert zutrifft. Beispielsweise kann man so aus Abbildung 5.8 ermitteln, dass z.B. 84% (Ordinate) der betrachteten Größe (Downloads) eine Geschwindigkeit (Abszisse) von 50 KByte/s **oder weniger** erreichen. Umgekehrt kann man so schließen, dass 16% der Downloads eine höhere Geschiwndigkeit als 50 KByte/s erreichen müssen (restlicher Anteil der vertikalen Achse bis zum Wert von 100%). Da es sich um Verteilungsfunktionen handelt, steht somit (im Falle von linear skalierten Achsen) eine Diagonale CDF für eine vollkommene Gleichverteilung der vorliegenden Ausgangsdaten.

Eine der herausragenden Eigenschaften des BitTorrent-Systems ist die hohe durchschnittliche Geschwindigkeit, mit der ein Download fertiggestellt werden kann. Aufgrund des ausgeklügelten Handelsverfahrens wird immer garantiert, dass die zur Verfügung gestellte Download-Geschwindigkeit im Verhältnis mit der Upload-Bereitschaft bzw. der realen Upload-Geschwindigkeit der jeweiligen Peers steht. Diese Vereinbarung ermöglicht somit auch eine außergewöhnlich hohe durchschnittliche Dienstqualität des Gesamtsystems. Eine Messung mit 54,845 beteiligten Peers in [Sip04] zeigt, dass ein Großteil (90%) der untersuchten Rechner eine Download-Geschwindigkeit von weniger als 65 KB/s aufweist (Abbildung 5.8, CDF). Im konkreten Fall bedeutet dies eine *durchschnittliche* Transferrate von 30 KB/s – eine Geschwindigkeit bei der es ohne weiteres möglich ist, große Dateien innerhalb eines Tages zu übertragen.

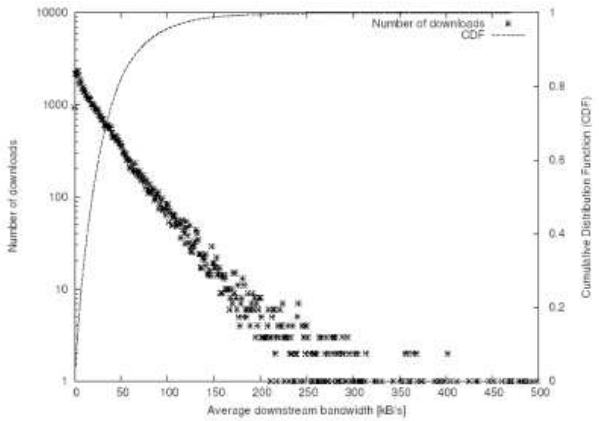

Abbildung 5.8: BitTorrent Download-Charakteristika

Der File Sharing-Dienst Gnutella kann durch seine weit entwickelte Peer-to-Peer Architektur eine ähnlich hohe Dienstqualität in Bezug auf die Downloadgeschwindigkeit eines Objekts bieten. Jedoch ist hier das Verhältnis zwischen Up- und Download nicht so strikt geregelt, wie im Falle von BitTorrent, weswegen hier weitere Dynamiken erkannt und erklärt werden müssen – Gleiches gilt für die Peer-to-Peer Netzwerke Napster und Kazaa.

Ein Ansatz Peers (und daraus folgend auch in gewissem Maße die Benutzer) in spezielle Profile zu unterteilen, bietet hier gute Erklärungsmöglichkeiten; dazu werden allerdings Schlussfolgerungen benötigt, die erst durch die Analyse von Eigenschaften der teilnehmenden Peers (Abschnitt 5.3.4) gezogen werden können.

Anhand von Abbildung 5.9 ist zu sehen, dass – am Beispiel von Gnutella – 22%

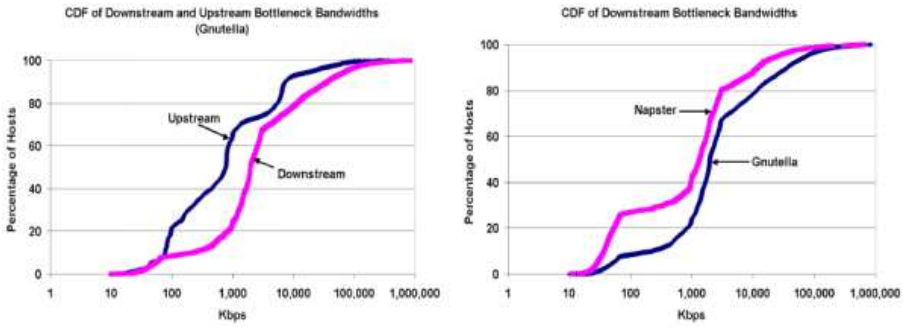

Abbildung 5.9: Up- und Download-Bandbreiten (Napster/Gnutella)

der gemessenen Peers höchstens eine geringe Upstream-Bandbreite von 100 KBit/s erreichen (analog dazu haben die restlichen 78% der Peers eine Anbindung schneller als 100 KBit/s). Eine vergleichsweise breitbandige Anbindung (mit Übertragungsraten größer als 10 MBit/s) besitzen dagegen nur 8% der untersuchten Peers.

Leider stehen für Napster und Gnutella keine genauen Datensätze über durchschnittliche Download-Geschwindigkeiten zur Verfügung, sodass hier im Gegensatz zu BitTorrent nur Aussagen über die allgemeine Verteilung der Maximalbandbreiten (*bottleneck bandwidths* gemacht werden können.

5.3.4 Eigenschaften teilnehmender Peers

Abbildung 5.10 zeigt die kumulative Verteilungsfunktion gemessener Netzwerklatenzen von Peers im Gnutella File Sharing-System (diese Daten sind relativ zum Ausgangspunkt, dem messenden Rechner, zu betrachten). Bei der Annahme einer homogenen geographischen Verteilung der Peers, ist die erhaltene Verteilungsfunktion jedoch auf exemplarische Weise repräsentativ für diese Untersuchung. Es ist zu beobachten, dass etwa 20% der Peers eine Mindestlatenz von 280ms haben, wobei ein vergleichbar hoher Anteil der Peers wiederum eine Maximallatenz von 70ms aufweist. Anschaulich bedeutet dies eine bis zu viermal größeren Distanz zwischen den entferntesten Peers der beiden beschriebenen Gruppen. Daraus lässt sich schließen, dass in einem P2P-Netzwerk, bei dem Verbindungen auf unstrukturierte Art eingegangen werden, ein großer Anteil der Verbindungen mit dem Problem hoher Latenzen belastet sein wird.

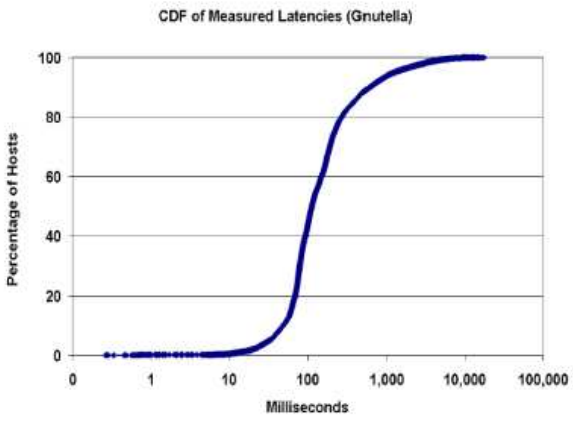

Abbildung 5.10: Gnutella Netzwerklatenzen

Neben den bisherigen Betrachtungen der Bandbreite und der Netzwerklatenzen spielt auch der Grad der Verfügbarkeit des einzelnen Hosts eine wichtige Rolle bei der angestrebten Kategorisierung. Die Kriterien für die Zuordnung in die Gruppe der serverartigen Peers sind also nicht nur eine hohe Bandbreite und eine niedrige Latenz, sondern auch eine hohe und somit auch dauerhafte Verfügbarkeit des ,,Servers“. Sollten solche Peers oft *nicht* verfügbar sein, so hat das gravierende Auswirkungen auf den Grad benötigter Replikationen im System, um Inhalte verfügbar zu halten – eine falsche Aufgabenzuteilung des P2P-Systems unter seinen teilnehmenden Hosts, sollte also immer vermieden werden, um diese Folgen ausschließen zu können.

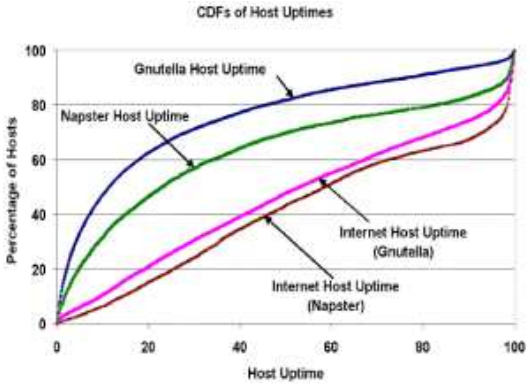

Abbildung 5.11: Host Uptimes (Napster/Gnutella)

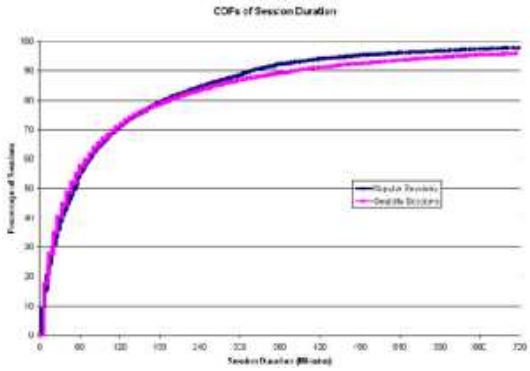

Abbildung 5.12: Verteilung der Sitzungsdauern

Betrachtet man weiterhin die Verfügbarkeitsverteilungen (*uptimes*) der Napster und Gnutella Peers (Abbildung 5.11), so kann man ähnliche Ausprägungen erkennen. Verglichen werden hier Application Host Uptimes (AHU, bezogen auf die Ausführungsdauer der File Sharing-Anwendung) und Internet Host Uptimes (IHU, reine IP-basierte Verfügbarkeit, d.h. der Peer ist im Internet mit seiner IP erreichbar). Die beobachteten IP-Uptimes sind dabei weitgehend identisch – nicht jedoch die anwendungsbezogene Verfügbarkeit. Die etwas flachere Kurve der Napster-Peers deutet auf eine häufigere Teilnahme dieser Benutzer im File Sharing-Netzwerk hin.

In absoluten Zahlen bedeutet dies, dass die ,,besten“ 20% der Napster-Clients eine Uptime von mehr als 83% (der gesamten Messdauer) aufweisen – im Falle von Gnutella hat das korrespondierende beste Fünftel nur eine Uptime 45% (oder mehr). Die mittlere Sitzungsdauer beträgt 60 Minuten, was im konkreten Anwendungsfall für das Herunterladen ein paar kleinerer Dateien ausreicht. Neben diesem

mittleren Wert zeigt Abbildung 5.12 die genaue Verteilung (CDF) der gemessenen Sitzungsdauern für die beiden Dienste Napster und Gnutella, wobei eine weitgehende Übereinstimmung der Messdaten zu beobachten ist.

Zusammen mit den Betrachtungen bezüglich der Download Charakteristika im letzten Abschnitt lässt sich somit die Gruppe der hochverfügbaren „Server"-Peers (*high availability profile*, *server-like*) modellieren.

Analog dazu gilt es nun ein clientähnliches Profil zu beschreiben – sie müssten sich dementsprechend gegenteilig zu den bereits untersuchten hochverfügbaren Peers verhalten. Im Zuge dieser Differenzierung soll nun auch die Anzahl der freigegebenen Objekte der jeweiligen Peers betrachtet werden. Schon die Bezeichnung clientähnlich gibt vor, dass diese Gruppe von Knoten vornehmlich Inhalte aus dem Netzwerk herunterladen (*always-downloading*) und aufgrund ihrer ausgesprochen serverunähnlichen Natur nur sehr wenige oder überhaupt keine Dateifreigaben besitzen (*no-files-to-share*).

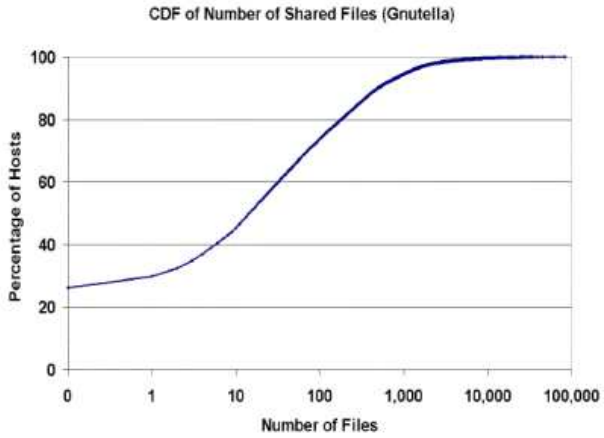

Abbildung 5.13: Verteilung der Dateifreigaben (Gnutella)

Insgesamt stellen somit also 75% der Knoten 100 Dateien oder weniger frei, wohingegen nur 7% der beteiligten Peers mehr als 1,000 Dateien freigeben – allein diese Rechner stellen mehr Daten zur Verfügung als alle anderen Peers zusammen (siehe Abbildung 5.13).

Der extremste Fall, dass ein Peer der Gemeinschaft überhaupt keine Dateien zur Verfügung stellt, kann in Abbildung 5.13 beobachtet werden. Der Beginn der Kurve (keine freigegebenen Dateien) zeigt, dass ein hoher Anteil von 25% der Clients bereits in diese Kategorie fällt – die korrespondierenden Benutzer werden oft auch als *free-riders* bezeichnet.

5.3.5 Anwendungsinstanzen und Benutzer

Eine der zentralen Fragen ist, ob die jeweiligen Anwendungsinstanzen (die Dienstprogramme mit denen sich ein Benutzer in ein File Sharing-Netzwerk einwählen kann) mit diesen Benutzern vollkommen identisch sind. Wie bereits in den vorigen Kapiteln erwähnt, kann eine Messung immer nur Statistiken über ein „laufendes Programm" erstellen; insofern muss bei allen weiteren Untersuchungen eine idealisierte Betrachtung in Form einer Identifizierung des Benutzers mit einer Anwendungsinstanz vorgenommen werden. Mit Sicherheit ist es denkbar, dass mehrere Personen (in welcher Art auch immer) eine einzige Anwendungsinstanz gemeinsam nutzen und somit die Messungen in gewisser Weise beeinflusst werden. Da die File Sharing-Anwendung IP-Verbindungen zu entfernten Rechnern aufbauen (vgl. Kapi-

tel 5.2.2), könnten sie also durch ihre assoziierte IP-Adresse unterschieden werden (der Verbindungsendpunkt wird immer mit dieser Adresse in Zusammenhang stehen). Dieses Unterscheidungskriterium kann jedoch in Verbindung mit häufig eingesetzten dynamischen Netzwerk-Konfigurationen (DHCP, *dynamic host configuration protocol*) in Unternehmens- und Universitätsnetzwerken problematisch sein. Ein weiteres eindeutiges Unterscheidungsmerkmal der Netzwerk-Peers sind die Benutzernamen, die während der Kommunikation untereinander übertragen werden (Kapitel 5.2.2, *kazaa protocol query header*); es wurde für die Messung in [Zah03] herangezogen.

Die während der Aufzeichnungen erschienene Lite-Version von Kazaa gibt hier standardmäßig bei der Installation einen Benutzernamen („KazaaLiteUser“) vor, der natürlich in dem größten Teil der Fälle später vom Benutzer aus Bequemlichkeit nicht mehr geändert wird. Diese Clients wurden in der Messung von [Zah03] nicht betrachtet, da hier keine eindeutige Unterscheidung der Peers mehr hätte vorgenommen werden können. Ähnliche Problematiken treten insbesondere auch bei Angaben auf, die freiwillig vom Benutzer gemacht werden können. Eine dieser Angaben ist die verfügbare Bandbreite, die ein File Sharing-Client, vorgegeben durch die reale Internetanbindung, konsumieren kann; sie wird meist bei der lokalen Installation der Client-Anwendung abgefragt. Allein 22% der Clients geben hier den Wert „Unknown“ an, was die Aussagekraft dieser gesammelten Informationen bereits aus diesem Grund infrage stellt. Sind die Benutzer nicht gewillt diese Angaben zu machen, so müssen also Lösungen gefunden werden, diese wichtigen Systemparameter in einer aktiven Art und Weise zu ermitteln.

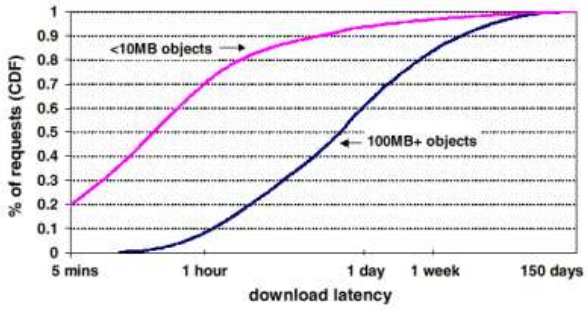

Abbildung 5.14: Download Latencies (Kazaa)

Eine weitere wichtige Beobachtung, die im Zusammenhang mit den Eigenschaften der File Sharing-Benutzer gemacht werden kann, ist ihre Geduldigkeit. Wie in Abbildung 5.14 zu sehen ist, benötigen 30% der kleinen Objekte (kleiner als 10 Megabyte) mehr als eine Stunde für den vollständigen Download. Für 40% der großen Objekte (größer als 100 Megabyte) beträgt die Dauer bereits mehr als einen Tag und schließlich auf 20% dieser Objekte muss der Benutzer mehr als eine Woche warten. Weiterhin ist aus den Messungen in Abbildung 5.15 abzuleiten, die Benutzer mit zunehmendem Alter „langsamer“ werden, d.h. ältere Clients konsumieren weniger Bytes in Bezug auf die Größe ihrer Downloads. Auch der Effekt der „Abnutzung“ zeigt, dass die Anzahl der Benutzer über die Zeit abnimmt bzw. die bisherigen Teilnehmer verlassen endgültig das Netzwerk. Entgegen dieser allgemeinen fallenden Tendenzen bleibt jedoch die Wahrscheinlichkeit einer Suchanfrage, die von einem Benutzer gestellt wird, über die Zeit gleich. Aus dieser unveränderlichen Wahrscheinlichkeit lässt sich ableiten, dass somit die Inhalte, die heruntergeladen werden, mit der Zeit kleiner bzw. weniger werden.

Neben der in Abschnitt 5.3.4 (Eigenschaften teilnehmender Peers) gemachten Einteilung der Knoten in zwei allgemeine Profile (*client-like* und *server-like*) ist für

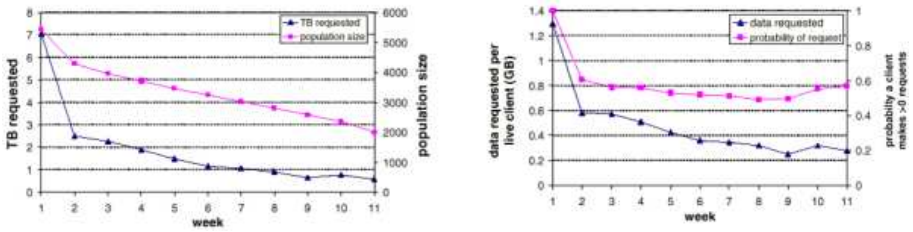

Abbildung 5.15: Benutzer Eigenschaften

die Analyse von File Sharing-Systemen nun auch die Betrachtung der Benutzereigenschaften (*no-files-to-share* und *always-downloading*) vorgenommen worden. Ein weiterer wichtiger Faktor, der die Effizienz eines File Sharing in sehr grundlegender Weise bedingt, sind die Objekte selbst, die über das System ausgetauscht werden.

5.3.6 Objekte und Inhalte

Sollen die Objekte und Inhalte untersucht werden, die über ein File Sharing-Netzwerk ausgetauscht werden, so muss die Frage gestellt werden zwischen welchen Objekt-Klassen überhaupt unterschieden werden kann und welche Dynamik diese Objekte besitzen. Dass heutige File Sharing-Netzwerke eine ganze Komposition aus verschiedensten Objekten verarbeiten müssen, macht Abbildung 5.16 deutlich. Im dem ermittelten Nutzlast-Profil des Kazaa-Netzwerks können drei Arten von Objekten unterschieden werden (kleine, mittlere und große Objekte).

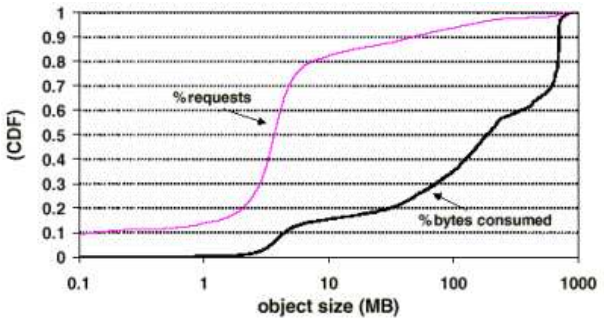

Abbildung 5.16: Nutzlast-Profil

Von besonderer Wichtigkeit für die weitere Analyse sind zwei dieser Gruppen und zwar die der kleinen Objekte mit einer Größe von mehreren Megabyte und die der großen Objekte jenseits von 100 Megabyte. Sie treten in Abbildung 5.16 durch die starken Steigungen im CDF-Diagramm in Erscheinung und stehen daher auch gleichzeitig für die das jeweilige File Sharing-System besonders beeinflussenden Objekte. Natürlich stellen diese beiden großen Klassen die Dateien mit Audio- und Videoinhalten dar, die in der überwältigenden Mehrheit der Fälle eine Größe von ca. 3 Megabyte bzw. ca. 600 bis 800 Megabyte besitzen. Ein Großteil der Requests in einem File Sharing-System entfallen dabei auf die kleinen Objekte (91%), dennoch wird der größte Teil der *übertragenen* Bytes durch die großen Inhalte verursacht (65%). In diesen Tatsachen spiegelt sich die dominierende Rolle der großen Dateien im Bezug auf den Verbrauch von Netzwerkbandbreiten wider.

Unabhängig von der Größe eines Objekts gelten für die Dateien in einem File Sharing-System jedoch noch weitere besondere Voraussetzungen. Im Gegensatz zu Web-Seiten, die im Zuge der Evolution des Internet immer mehr dynamische Inhalte erhalten haben, sind Multimedia-Dateien unveränderlich – ein Film z.B. ändert sich

nach seiner Bereitstellung nicht mehr. Dynamische Web-Seiten stellen den größten Einfluss auf das Verhalten des World Wide Web dar; bei File Sharing-Systemen müssen dafür aber andere Faktoren verantwortlich gemacht werden. Neben dieser Unveränderlichkeit werden Objekte auch meist nur einmal aus dem System heruntergeladen (*fetch-at-most-once*) – 94% aller Requests fordern eine Datei höchstens einmal an. Auch sind Tendenzen zu beobachten, dass die Popularität eines Inhaltes (eine Größe die direkt durch die Benutzerschaft vorgegeben wird) meist kurzlebig ist und dass beliebte Inhalte dazu neigen erst kürzlich in das System Einzug gehalten zu haben – die meisten Anfragen in einem System richten sich dabei jedoch auf ältere Objekte [Zah03].

5.3.7 Schlussfolgerungen

Abschließend muss also festgehalten werden, dass es sehr große Unterschiede zwischen den Anwendungen des File Sharings und dem World Wide Web gibt und somit File Sharing-Netzwerke nur sehr eingeschränkt damit verglichen werden können. Es bedeutet auch, dass frühere Deutungen und Zusammenhängen, die im Falle des WWW zutreffen nun nicht mehr ohne weiteres anwendbar sind – die gemachten Beobachtungen müssen modelliert und in eine konsistente Gesamtsicht eingearbeitet werden.

5.4 Modellierung

5.4.1 Zipf's Gesetz

Eine für die Modellierung des Internet wichtige Gesetzmäßigkeit ist das Zipf-Gesetz (*Zipf's law*), das eine Aussage über die Beliebtsheitsverteilung eines Objekts macht („Wie oft wird dass n-beliebteste Objekt innerhalb eines Zeitraums aufgerufen?“). Das Zipf-Gesetz wurde in bisherigen Veröffentlichungen oft zur Deutung von entsprechenden Beobachtungen im Fall des WWW zitiert und beschreibt dort auch sehr genau die allgemein anerkannten Zusammenhänge. Konkret verhält sich die Beliebtheit des i-ten Objekts proportional zu $i^{-\alpha}$, wobei α als Zipf-Koeffizient bezeichnet wird ($\alpha \in (0...2)$). Setzt man jedoch die Aufrufverteilung eines File Sharing-Systems in das Diagramm einer Zipf-Verteilung ein (die hier in Abbildung 5.17 durch die logarithmischen Skalen die Form einer Geraden annimmt), so ist zu erkennen, dass die Kurve im Bereich der populären Objekte viel flacher ist – so passt auch nicht angenähert in der Zipf-Profil.

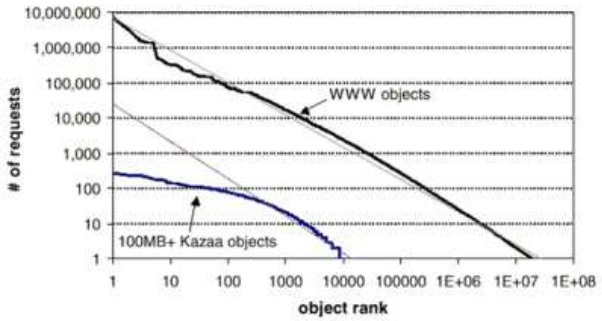

Abbildung 5.17: Kazaa Objekte im Zipf-Diagramm

Hier zeigt sich unmittelbar die vollkommen andere Natur der File Sharing-Netzwerke im Vergleich zum WWW, denn hier besteht die größte Dynamik nicht in

der Veränderlichkeit bereits existierender Objekte (wie z.B. Web-Seiten), sondern in der Ankunft vollkommen neuer Objekte. Der Grund liegt in den vorgestellten Eigenschaften – nocheinmal gegenübergestellt in Tabelle 5.4.1.

WWW:	*mutable objects*, *fetch-repeatedly clients*
File Sharing:	*immutable objects*, *fetch-at-most-once clients*

Im Falle von abgeschlossenen Netzwerken ist eine Anbindung an das Internet oft in Form eines Proxy-Servers (HTTP-Proxy) gegeben. Dieser fängt häufige Anfragen auf die selben Inhalte auf und sendet jeden *request* nur ein Mal, denn bei einem *cache hit* können die bereits bekannten Daten ohne weitere Nachfrage vom Proxy an den Client übermittelt werden. Diese Eigenschaft spiegelt genau den Übergang einer *fetch-repeatedly* zu einer *fetch-at-most-once* Charakteristik wider. Eine Zipf-verteilte Menge von Anfragen wird also in eine non-Zipf-Verteilung ,,transformiert".

5.4.2 Verbesserungsmöglichkeiten

Aufgrund der weiten Verbreitung von File Sharing-Anwendungen und somit auch ihrer wachsenden Bedeutung für Netzwerkadministratoren in Hinsicht auf Konsum von Bandbreite, werden Möglichkeiten zur Steuerung und Begrenzung speziell von P2P-Netzwerkverkehr immer wichtiger. Zwar ist es den Verantwortlichen möglich die ein- und ausgehende Netzwerkbandbreite gezielt zu beeinflussen (*traffic shaping*), jedoch kann dieser Ansatz nur eine vorübergehende Lösung darstellen.

Eine gravierende Schwachstelle bisheriger File Sharing-Konzepte ist die ungenutzte Örtlichkeit gespeicherter Objekte. Obwohl die Installation eines Proxys am Perimeter eines großen Netzwerks aufgrund von möglichen Copyright-Verletzungen und hohen Anforderungen an Ressourcen problematisch ist, könnte die Belastung für diese Netzwerke durch eine Ausnutzung der hier bestehenden Lokalität (*locality awareness*) verringert werden. Diese Lokalität kommt durch die möglicherweise hohe Anzahl von Benutzern in diesem abgeschlossenen Netzwerk zustande, die evtl. gesuchte Inhalte bereits heruntergeladen haben, die Suchalgorithmen des jeweiligen File Sharing-Netzwerks sich dieser ,,kostengünstigen" Verbindung innerhalb eines Subnetzwerks jedoch nicht bewusst sind. Solche Anfragen müssten also von einer zentralen Stelle in dem Netzwerk aus auf ,,lokale" Peers umgeleitet werden (*centralized request redirection*). Analysen des bisher aufgestellten non-Zipf-Modells in [Zah03] prognostizieren selbst bei einer konservativen Hochrechnung mögliche *hit rates* von bis zu 63% durch die Implementierung einer zentrale Umleitung für Objektanfragen – sollten sich diese Werte in der Realität bestätigen, so wäre dies ein sehr effektiver Ansatz, um signifikante Einsparungen in Bezug auf Bandbreite und die damit für die Einrichtungen verbundenen Kosten zu erreichen.

5.4.3 Zusammenfassung

Die erste Feststellung die zusammenfassend bei der Analyse von Peer-to-Peer File Sharing-Netzwerken gemacht werden muss ist, dass Web-Systeme und File Sharing-Systeme nur sehr bedingt vergleichbar sind. Allein zwei der wichtigsten Faktoren, die diese Systeme beeinflussen, wie die Benutzer und die durch das System übertragenen Objekte, verhalten sich gegensätzlich. Weiterhin werden bei dem Entwurf von Peer-to-Peer Architekturen oft implizite Annahmen, u.a. über die Eigenschaften der teilnehmenden Peers und das Verhalten bzw. die Motivationen der Benutzer gemacht. Die am System teilnehmenden Knoten sind entgegen der Peer-to-Peer Philosophien nicht uniforme Peers (beispielsweise in Hinsicht auf ihre Anbindung zum Netzwerk) und daher sollte statt der Homogenität die Heterogenität der Peers in den Mittelpunkt der Betrachtungen rücken, um eine effektive Aufgabenverteilung innerhalb

des Systems erreichen zu können. Auch im Falle des Benutzerkreises wird eine „Bereitschaft zur Kooperation“ angenommen, also dass in gleichem Maße alle Knoten Dateien zur Verfügung stellen ebenso wie sie sie konsumieren. Bei den Untersuchungen konnten den Knoten aufgrund ihres Verhaltens jedoch eindeutig verschiedene Rollen zugeordnet werden, die weitgehend dem klassischen Client-/Server-Modell entsprechen. Diese Tendenzen setzten sich auch bei kritischen Angaben wie z.B. der verfügbaren physikalischen Anbindung fort, die freiwillig von den Benutzern gemacht werden können – für verlässliche Informationen dürfen hier nur real gemessene Daten herangezogen werden, da sonst möglicherweise kritische Aufgaben im System falsch verteilt werden.

Eine bedeutende Möglichkeit den Ressourcenverbrauch heutiger File Sharing-Systeme zu verringern, könnte in der Berücksichtigung ungenutzter Dateilokalität liegen – hier haben die vorgestellten Modelle gezeigt, dass erhebliche Effizienzsteigerungen erzielt werden könnten; eine entsprechende Implementierung vorausgesetzt. Weiterhin ist das Verfügbarkeitskriterium für Peer-to-Peer Systeme von zentraler Bedeutung und daher sollten Komponenten, die einen einzelnen Verwundbarkeitspunkt darstellen, soweit wie möglich auf ein Minimum reduziert werden.

Alldies hängt jedoch davon ab, wie sich Peer-to-Peer Systeme in der Zukunft weiterentwickeln werden – ein ursprünglicher Antrieb diese Systeme zu entwerfen war ihre Notwendigkeit. Peer-to-Peer Systeme werden also weiterhin an den direkten Bedürfnissen der Anwender richten, weshalb sich die Frage stellt, ob sich neue theoretische Ansätze und Verbesserungsvorschläge im praktischen Einsatz durchsetzen werden. Betrachtet man jedoch die bisher erreichte Beliebtheit dieser Systeme bei der Anwenderschaft, so scheint es unwahrscheinlich, dass sie in Zukunft je wieder an Bedeutung verlieren werden.

Literaturverzeichnis

[Gri02] Stefan Saroiu; P. Krishna Gummadi; Steven D. Gribble. A measurement study of peer-to-peer file sharing systems, January 2002.

[Sip04] J.A. Pouwelse; P. Garbacki; D.H.J. Epema; H.J. Sips. A measurement study of the bittorrent peer-to-peer file-sharing system, May 2004.

[Zah03] P.K. Gummadi; Richard J. Dunn; Stefan Sroiu; Steven D. Gribble; Henry M. Levy; John Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload, October 2003.

Kapitel 6

Grid Security

Seminarbeitrag von **Thomas Blattmann**

6.1 Einleitung

Die hier vorliegende Ausarbeitung entstand im Rahmen des Grid-Computing Seminars im Sommersemester 2003/2004 an der Universität Karlsruhe. Es werden zwei unterschiedliche Themen behandelt: Im ersten Teil werden sicherheitsrelevante Aspekte zu verteilten Hashtabellen (Distributed Hash Tables) vorgestellt und diskutiert. Auf einzelne Angriffsmöglichkeiten wird genauer eingegangen und entsprechende Abwehrtechniken werden vorgestellt. Dabei wird Bezug genommen auf die bekanntesten Protokolle, die heute zur Realisierung einer DHT eingesetzt werden.

Der zweite Teil gibt eine Einführung in Spyware und die davon ausgehenden Gefahren. Es werden verschiedene Kategorien von Spyware vorgestellt und anhand konkreter Beispiele die Funktionsweise solcher Software genauer unter die Lupe genommen. Aktuelle Untersuchungen sollen die weite Verbreitung von Spyware aufzeigen. Abschliessend werden Möglichkeiten vorgestellt um entstehenden Gefahren vorzubeugen und bereits installierte Spyware entfernen zu können.

6.2 Sicherheit in Verteilten Hashtabellen

In Anlehnung an das elektrische Versorgungsnetz (electric power grid), das um 1910 in den USA errichtet wurde und für damalige Verhältnisse eine revolutionäre Entwicklung darstellte, steht der Ausdruck **Computational Grid** für eine Hard- und Software Infrastruktur die zuverlässigen, allgegenwärtigen und billigen Zugang zu verteilten Rechnerkapazitäten bereitstellt. Es soll beispielsweise möglich werden, global verteilte Computer zusammenzuschließen und so den Zugriff auf Rechenleistungen im Teraflop-Bereich zu erreichen. Durch den Zusammenschluss vieler

Rechner werden Anforderungen gedeckt, die selbst durch modernste Supercomputer nicht befriedigt werden können. Das dabei angestrebte Ziel ist, daß Rechnergrenzen nicht mehr wahrnehmbar sind, wenn die Gesamtheit der vernetzten Knoten zu einem logischen Rechner zusammengefasst werden.

Eine gewöhnliche Hash-Table ist eine leistungsfähige Datenstruktur in der Schlüssel und Objekte abgelegt und wiedergewonnen werden können. Nachdem ein Werte-Schlüssel-Paar in die Tabelle eingefügt wurde, kann der gesuchte Wert unter Angabe des Schlüssels relativ schnell wieder ausgelesen werden. Verteilte Hashtabellen stellen die gleiche Funktionalität wie eine gewöhnliche Hashtabelle bereit mit dem Unterschied, daß die gesamte Struktur verteilt und auf verschiedenen Knoten in einem Netzwerk abgelegt werden kann. Sie finden in unterschiedlichen Bereichen Anwendung - insbesondere in Peer-to-Peer Systemen wie Grid-Netzen. Um später auf die Sicherheitsproblematik in unterschiedlich organisierten DHTs eingehen zu können werden im Folgenden vier bekannte und verbreitete Protokolle kurz vorgestellt.

6.2.1 Lookup- und Storage-Protokolle in verteilten Hashtabellen

Chord

Chord ist eine als Ring organisierte verteilte Hashtabelle. Das Protokoll ist fehlertolerant gegenüber Ausfällen einzelner Knoten, es unterstützt den Anschluss weiterer Knoten während des Betriebs und verteilt die Daten gleichmäßig auf allen Rechnern (load balancing) womit eine Suchtiefe von O(log N) erreicht wird. Die folgende Graphik veranschaulicht die Arbeitsweise von Chord.

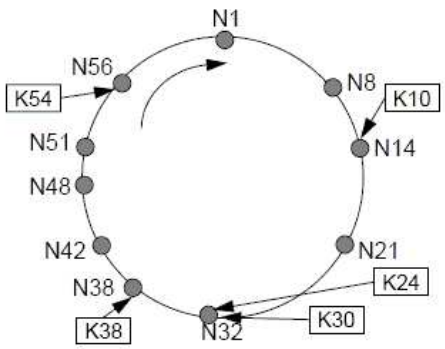

Abbildung 6.1: Chord

Abbildung 1.1 zeigt die im Ring angeordneten Knoten (N1, N8, ..., N56) mit einer eindeutigen, 160bit breiten ID sowie fünf einzufügende Datensätze mit den Schlüsseln K10, K24, K30, K38 und K54. Die Datensätze werden im Uhrzeigersinn von Knoten zu Knoten weitergereicht bis schliesslich ein Knoten erreicht wird, dessen ID größer oder gleich dem Schlüsselwert ist. In der Graphik ist beispielsweise Knoten N14 der Endpunkt für der Datensatz mit dem Schlüssel K10 und Knoten N56 Endpunkt für den Datensatz mit dem Schlüssel K54. An dieser Stelle wird der Datensatz dann abgespeichert.

Chord speichert selbst keine Schlüssel und Werte. Seine Funktionalität besteht einzig und allein darin, einen gegebenen Schlüssel auf den dafür zuständigen Knoten abzubilden (sog. Lookup Protokoll). Das eigentliche Speichern wird von einer höheren Anwendungsschicht (Storage Protokoll) implementiert.

Content Addressable Network (CAN)

CAN realisiert ähnlich zu Chord eine verteilte Hashtabelle. Dabei werden die Knoten in einem d-dimensionalen, virtuellen, kartesischen Koordinatensystem angeordnet, so daß jeder Knoten für einen bestimmten Teil des gesamten Koordinatenraums zuständig ist. Die Schlüssel werden über eine Hashfunktion auf einen Punkt im Raum abgebildet. Daten zu einem Schlüssel werden von dem Knoten gespeichert, zu dessen Bereich der Punkt gehört.

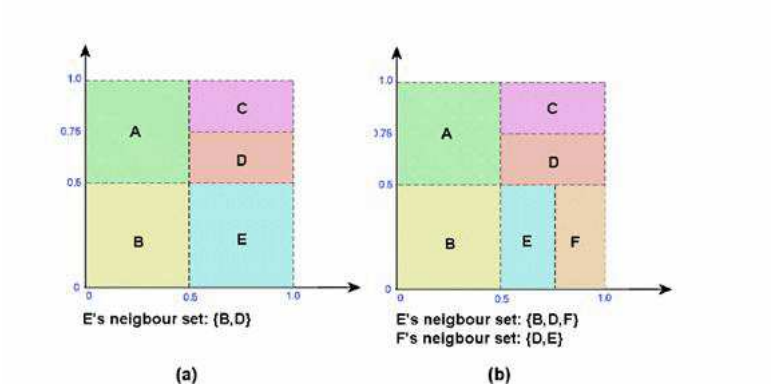

Abbildung 6.2: CAN

Abbildung 1.2 zeigt einen 2-dimensionalen Raum und veranschaulicht die Unterteilung in Koordinaten vor (a) und nachdem (b) sich eine neuer Knoten (F) der Hashtabelle angeschlossen hat. Das Einfügen von Daten erfolgt in drei Schritten. Zuerst wird der Abbildungspunkt des Schlüssels im Koordinatenraum berechnet. Es folgt der Transfer der Daten zum berechneten Punkt und schliesslich das Speichern der Daten durch den zuständigen Knoten. Analog erfolgt das Anfordern von Daten, die am Ende der Vorgangs zum anfragenden Knoten transferiert werden. Jeder Knoten in CAN kennt nur die unmittelbaren Nachbarn seines Teils des Schlüsselraums. Nachrichten werden an den Nachbarknoten mit dem geringsten euklidischen Abstand zum Zielpunkt weitergeleitet.

Pastry

In Pastry sind die Knoten in einer Baumstruktur organisiert wobei jedem Teilnehmer eine eindeutige 128 bit ID zugeordnet ist. Jeder Knoten hat 2ˆb ihm bekannte Nachfolger (in der Regel b = 4). Der Knoten kennt außerdem eine bestimmte Zahl von Rechnern deren IDs nahe an der eigenen liegen. Diese IDs werden im Leaf-Set gespeichert. Weiterhin unterhält jeder Knoten eine generelle Tabelle zur präfixbasierten Zustellung von Nachrichten (Routing Tabelle) sowie eine Tabelle mit direkten Nachbarn(Neighborhood-Set).Das eigentliche Routing läuft in drei Schritten ab:
Zuerst wird ermittelt, ob der Zielknoten im Leaf-Set liegt und die Nachricht gegebenenfalls dorthin geschickt. Falls nicht, wird die eigene Tiefe im Suchbaum ermittelt, die der Länge des gemeinsamen Präfixes von Nachrichten-ID und eigener Knoten-ID entspricht. Die Nachricht wird dann an den Knoten geschickt, dessen Präfix in einer weiteren Stelle mit der Nachrichten-ID übereinstimmt. Sollte kein passender Knoten gefunden werden, wird die Nachricht an den nummerischen nächsten Knoten aus allen bekannten Tabellen übergeben.

DHash

Chord ist ein Protokoll, das es ermöglicht anhand des Schlüssels (der sich aus dem Hashwert eines Datenblocks ergibt) den Knoten zu finden, der den Block speichert. Allerdings bleibt die Aufgabe des persistenten redundanten Speicherns der Datenblöcke sowie ein Caching derselben einer weiteren Schicht überlassen. DHash wird oft zusammen mit Chord eingesetzt und übernimmt die Aufgabe des Speicherns und Replizierens der Datenblöcke. Dabei wird jeder Satz in k aufeinanderfolgenden Rechnern repliziert. Zusätzlich verfügt jeder Knoten über einen Cache, der nach dem least-recently-used Verfahren aktualisiert wird.

6.2.2 Angriffe auf das Routing in verteilten Hastabellen

Distributed Hash Tables können offene Netzwerke sein die es anderen Knoten erlauben, jederzeit einzusteigen oder bereits integrierten Knoten auszuscheiden. An die abzuspeichernden Daten können unterschiedliche Anforderungen bzgl. Datenschutz/Datensicherheit, Langlebigkeit und Datenverfügbarkeit gestellt werden. Daher sollten die verwendeten Protokolle Mechanismen wie Reputationsmanagement, Replikation, Verschlüsselung usw. unterstützten, die es erlauben entsprechende Angriffe abzuwehren und einzelne Ausfälle zu kompensieren. Ein erster Ansatz die entstehenden Gefahren zu reduzieren, besteht darin, die Bedrohungen und das Ausmaß an Schäden einschätzen zu können um die Verfügbarkeit des Systems entsprechend anzupassen.

Angriffe auf verteile Hastabellen können prinzipiell drei verschiedenen Schichten zugeteilt werden, der Netzwerkschicht (network layer, TCP/IP), der darauf aufbauenden Schicht zur Zuordnung von Schlüsseln zu Knoten (lookup protocol) sowie der Schicht zum Abspeichern der Daten. In jeder Schicht sind andere Sicherheitsaspekte zu berücksichtigen wobei zusätzlich zu beachten ist, daß die Funktionalität der höheren Schichten vom korrekten Verhalten der unteren Schichten abhängt. So setzt beispielsweise das Lookup-Protokoll eine zuverlässige und sichere Übertragung seiner Nachrichten zum Zielknoten vom Netzwerk voraus.
Im Folgenden wird ein Überblick über Angriffe auf das Routing in verteilte Hashtabellen und deren Auswirkungen gegeben sowie mögliche Abwehrtechniken untersucht. Anschliessend werden Angriffe auf gespeicherte Daten und weitere Angriffsmöglichkeiten behandelt. An geeigneter Stelle wird dabei auf die vorgestellten Protokolle eingegangen und deren Stärken und Schwächen diskutiert. Weiterführende Informationen zum Thema finden sich auch in [SM02] sowie in [SL04].

Anfragen falsch weiterleiten

Angriffe auf das Routing haben das Ziel, die Kommunikation im Netzwerk zu stören um Anfragen und interne Steuerinformationen fehlzuleiten. Gelingt eine solche Attacke ist die Funktionalität des Systems erheblich beeinträchtigt da die Daten zwar noch vorhanden sind, der Zugriff darauf allerdings nicht mehr möglich ist.

Zunächst einmal kann ein im System integrierter Angreifer Anfragen nicht richtig weiterleiten und statt dessen an falsche oder nicht vorhandene Knoten adressieren. Der anfragende Knoten wird nach einiger Zeit noch einmal versuchen den Zielknoten zu erreichen. Ist die Wegwahlentscheidung jedoch derart implementiert, daß immer die selbe Route genommen wird, geht das Paket auch bei erneuten Versuchen verloren. Ein erfolgreicher Angriff resultiert letztlich darin, daß dem anfragenden Knoten die gewünschten Informationen vorenthalten werden (**denial of information**). Da der Angreifer nach Außen hin einen funktionalen Eindruck macht - die anderen Knoten können ja nicht wissen an wen die Pakete weitergeleitet werden - gibt es zunächst keinen Grund ihn vom System auszuschliessen.

Abhilfe schaffen Protokolle die es den Knoten ermöglichen, die einzelnen Hops der Pakete bis hin zum Empfänger mitzuverfolgen und gegebenenfalls alternative Routen zu wählen. In Pastry beispielsweise weiss der Sender, daß die ID des nächsten Hops in mindestens einer weiteren Stelle mit dem Schlüssel der Daten übereinstimmen muss. Fehlgeleitete Anfragen können erkannt und über einen anderen Teilnehmer zum Zielknoten geleitet werden. Unterstützt das Protokoll zusätzlich unabhängige Pfade (Pfade, die vom Quellknoten zum Zielknoten führen und außer diesen beiden keine weiteren Knoten gemeinsam haben), stehen immer Alternativen zur Auswahl. Allerdings garantiert die Mehrheit der verteilten Hashtabellen die Existenz solcher Pfade nicht. Insgesamt sind drei Faktoren für die Leistungsfähigkeit des Algorithmus im Falle eines Angriffs, wie er eben geschildert wurde, entscheidend:

- Die Existenz mehrerer alternativer Pfade zwischen zwei Knoten,
- die zusätzlichen Kosten, die durch alternative Routen entstehen sowie
- die Fähigkeit, falsche Routinginformationen erkennen und drauf reagieren zu können.

Falsche Schlüssel-Knoten Zuordnung

Ein weiterer bösartiger Knoten kann einen Schlüssel dem falschen Knoten zuordnen woraufhin die Daten an falscher Stelle abgespeichert werden und später nicht mehr auffindbar sind. Auch könnte der Angreifer bestimmte Daten bei sich abspeichern wollen und daher fälschlicherweise behaupten für beliebige Schlüssel verantwortlich zu sein. Diesen Angriffen kann in zwei Schritten begegnet werden:
Zunächst sollte der Sender beim Empfänger anfragen und sicherstellen, daß die Daten mit dem entsprechenden Schlüssel auch wirklich in den Zuständigkeitsbereich des Zielknotens fallen bevor die eigentlichen Daten übermittelt werden.
Zweitens sollten die Schlüssel den Knoten so zugeteilt werden, daß die Zuordnung auch jederzeit für alle Teilnehmer nachprüfbar ist. Chord berechnet die Schlüssel einzelner Teilnehmer als Hashfunktion über deren IP-Adresse. Da die IP-Adresse benötigt wird um einen Knoten zu kontaktieren, ist für den Sender leicht nachvollziehbar, ob er mit dem richtigen Empfänger spricht. CAN und andere Protokolle hingegen erlauben es Teilnehmern ihre ID's selbst festzulegen und machen es so unmöglich herauszufinden, ob ein Knoten auch wirklich für einen Schlüssel verantwortlich ist.

Routingtabellen beschädigen

Ein Angreifer kann auch versuchen, die Routingtabellen anderer Teilnehmer durch falsche Routinginformationen zu beschädigen und so unschuldige Knoten dazu verleiten, Pakete im Netzwerk fehlzuleiten. Daher sollte der aktualisierte Knoten stets nachprüfen ob die Rechner in seinen Tabellen erreichbar sind. Einige Protokolle wie Pastry stellen zusätzliche Anforderungen (bestimmte Präfixe) an Updateinformationen und verwerfen diese, sollten die Anforderungen nicht erfüllt sein. Letztlich wird jeder Knoten für sich entscheiden müssen, ob er dem anderen Knoten vertraut und die neuen Informationen übernimmt oder nicht.

Simulationen von Srivatsa und Liu

In verschiedenen Simulationen versuchen M. Srivatsa und L. Liu die Auswirkungen der bisher vorgestellten Mechanismen auf Leistung und Funktionalität zu zeigen. Im Folgenden wird auf das Ergebnis der Untersuchung kurz eingegangen. Detaillierte Informationen zu der Versuchsreihe findet sich dann in der im Anhang angegebenen Versuchsdokumentation [SL04].

Das erste Experiment zeigt, daß die Wahrscheinlichkeit von Fehlern im Lookup-Protokoll durch das Vorhandensein unabhängiger Pfade tatsächlich reduziert wird. Dazu wurde ein System mit insgesamt 1024 Knoten auf Basis eines Chord-Protokolls sowie vier unterschiedlich dimensionierten CAN-Protokollen simuliert.

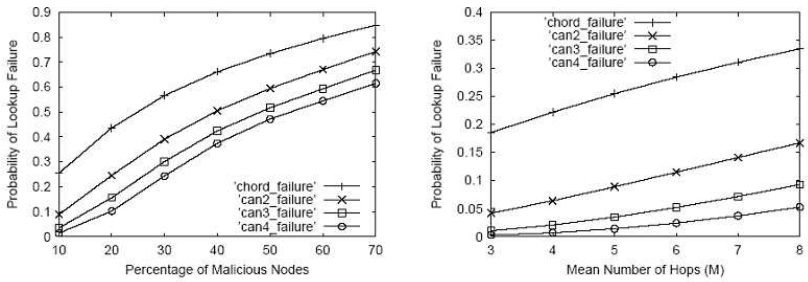

Abbildung 6.3: Versuch 1

Abbildung 1.3 ist zu entnehmen, daß in Chord 50% aller Anfragen verloren gehen falls ein Viertel der Knoten das Protokoll nicht korrekt umsetzt. Im 4-dimensionalen CAN hingegen erreichen bei gleicher Konstellation lediglich 10% der Anfragen ihr Ziel nicht. Weiterhin wird die Wahrscheinlichkeit von Fehlern in Abhängigkeit der Anzahl Hops, die durchschnittlich benötigt werden um vom anfragenden Knoten das Ziel zu erreichen, dargestellt. Dabei sind 10% aller beteiligten Knoten Angreifer.

Die zweite Simulation soll die Performanz eines Protokolls, das falsche Wegwahlentscheidung erkennt, mit einem Protokoll ohne diese Möglichkeit vergleichen.

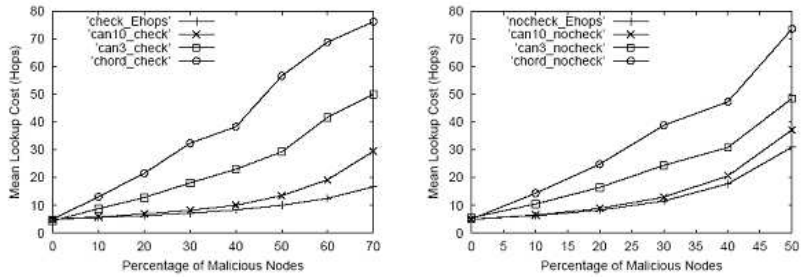

Abbildung 6.4: Versuch 2

Es stellte sich heraus, dass bei großen Werten für p (p = Anteil bösartiger Knoten) und M (M = Durchschnitt Hops von Sender zum Empfänger) ein Mitverfolgen der Route zu einer verbesserten Leistung führt wohingegen bei kleinen Werten für p keine nennenswerten Auswirkungen festzustellen waren. Tatsächlich waren die Kosten in einem 10-dimensionalen CAN mit p ohne Überprüfung nicht mehr als 5 - 8% über den Kosten des gleichen CAN-Protokolls mit Überprüfung.

6.2.3 Angriffe auf gespeicherte Daten

Ein ganz anderer Versuch, die Funktionalität einer verteilten Hashtabelle zu stören, besteht darin, abgespeicherte Daten zurückzuhalten bzw. deren Existenz zu leugnen. Weiterhin kann der Angreifer alle bei ihm abgespeicherten Daten verfälschen. Für den Fall, daß diese Daten nur in einem Knoten abgespeichert werden, ist ein Wiedergewinnen der ursprünglichen Informationen nicht mehr möglich. Daher sollten alle Protokolle durch Replikation der Daten auf mehreren Knoten dafür sorgen,

daß sie selbst dann noch zugänglich sind, wenn einzelne Teilnehmer ausfallen oder die Daten bewusst zurückgehalten werden. Leider folgen nicht alle Protokolle diesem Prinzip. DHash ermöglicht zwar das Replizieren von Daten - allerdings ist dafür ein einzelner Knoten verantwortlich. Um die Verantwortung auf mehrere Rechner zu verteilen wird in CAN der Ansatz gewählt, durch mehrere Hashfunktionen die Daten von Vornherein auf unterschiedlichen Rechnern abzulegen um Angriffe, wie oben geschildert, zu verhindern. Eine weitere Schwachstelle entsteht in Protokollen ohne verifizierbare und eindeutige Knoten-IDs. Der Angreifer hat dann die Möglichkeit dem System mehrmals nacheinander mit unterschiedlichen ID's beizutreten und auszusteigen, um verschiedene Daten zu manipulieren.

Die folgenden Abbildungen als Resultat weiterer Versuche von Srivatsa und Liu (mit 1024 Knoten und dem Chord-Protokoll) verdeutlichen die Problematik. In jeder Iteration schliesst sich der Angreifer dem Netz mit einer neuen ID an und manipuliert die ihm zugewiesenen Daten. Im ersten Versuch mit 7 Replikationen pro Datensatz war es jedem Knoten erlaubt maximal 20 unterschiedliche ID's anzunehmen. Ist jeder zehnte Knoten ein Angreifer, dann ist nach bereits 50 Iterationen fast die Hälfte der Daten im System beschädigt. Bei über 30 Prozent bösartiger Knoten sind nach gleich vielen Iterationen über 90 Prozent der Datensätze unbrauchbar. Die rechte Teilgraphik aus Abbildung 1.5 veranschaulicht den Anteil beschädigter Informationen für verschiedene Replikationsraten und einem Anteil bösartiger Knoten von zwei Prozent. Man kann entnehmen, daß sogar bei 11 Replikationen (R=11) pro Datensatz 15 Prozent der Informationen nach 125 Iterationen korrumpiert sind

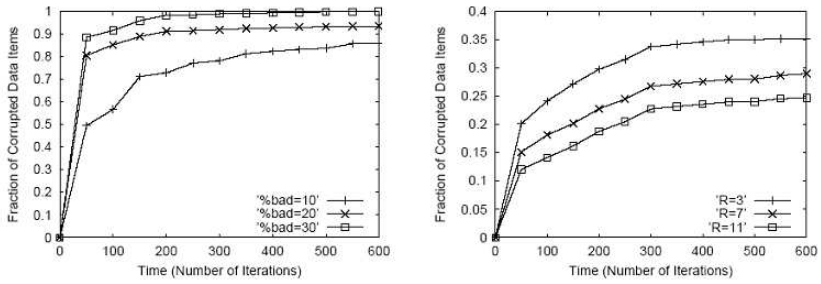

Abbildung 6.5: Versuch 3

Es bleibt festzuhalten, daß Replikation alleine nicht ausreicht um mit dem Verhalten angreifender Knoten fertig zu werden. Ein möglicher Ansatz ist, durch Verschlüsselung bzw. Signaturen die Echtheit und Richtigkeit der Daten zu garantieren. Daß sich solche Verfahren auch im praktischen Einsatz bewähren, wird durch weitere Experimente bestätigt. Allerdings ist zu beachten, daß dabei zusätzlicher Overhead entsteht. So dauert etwa eine 1024-Bit RSA Signatur auf einem 900MHz Pentium 3 Prozessor 8ms und die Verifikation einer signierten Nachricht immerhin noch 1ms.

6.2.4 Weitere Angriffstechniken

Inkonsistentes Verhalten

Angreifer, die sich teils korrekt verhalten und anderen Teilnehmern gegenüber inkorrekt, sind nur schwer ausfindig zu machen. Teilnehmer, die den störenden Knoten identifiziert haben, müssen also versuchen, alle anderen Teilnehmer davon zu überzeugen, ihn aus deren Routingtabellen zu entfernen. Durch gegenseitiges Bewerten (s. Abschnitt Reputationsmanagement) sollte dabei verhindert werden, daß fälschlicherweise sich korrekt verhaltende Knoten in Folge eines Angriffs ausgeschlossen

werden.

Denial of Service Attacken

Versucht ein Angreifer einen anderen Knoten mit zufällig erzeugten Paketen zu überfluten und so außer Funktion zu setzen, spricht man von einer Denial of Service Attacke. Aus Sicht des Gesamtsystems ist der betroffene Knoten ausgefallen und wird zwangsläufig aussortiert. Solche Angriffe sind besonders in Systemen mit keiner oder geringer Replikationsrate erfolgreich. Sind die Daten hingegen auf mehreren Rechner vorhanden, wird die Funktionalität durch den Ausfall einzelner Knoten nicht beeinträchtigt.

Reorganisationsmechanismus

Eine weitere Angriffsfläche bietet der Reorganisationsmechanismus vieler verteilter Hashtabellen. Schliessen sich neue Knoten dem System an oder steigen alte aus, dann müssen die Daten neu verteilt werden. Solch ein Vorgang kann einen erheblichen Netzwerkverkehr zur Folge haben. Für diesen Zeitraum ist die Leistungsfähigkeit im Vergleich zum normalen Arbeitsbetrieb stark reduziert. Angreifer zielen daher darauf ab die Notwendigkeit einer Reorganisation vorzutäuschen. Je weniger Daten jedoch auf einzelnen Knoten abgespeichert sind, desto schneller kann sich das System reorganisieren und so derartigen Angriffen entgegenwirken.

Antworten fälschen

In einer Situation in der eine Anfrage von Knoten Q über Knoten E nach A weitergeleitet wird, kann der vermittelnde Knoten E davon ausgehen, daß Q sich als nächstes mit A in Verbindung setzen wird. Daher kann er versuchen eine Antwort von A nachzustellen um so Q mit Fehlinformationen zu täuschen. Signierte Antworten machen derartige Angriffe unmöglich, sind aber aufwendig und führen zu Leistungseinbußen. Eine Alternative besteht darin, jeder Anfrage eine zufällig erzeugte Bitsequenz mitzugeben und die Antwort samt Bitsequenz nur zu akzeptieren wenn die empfangene Bitsequenz mit der gesendeten übereinstimmt.

Partitionen

Abschliessend sollen noch Partitionen (**engl. partitions**) vorgestellt werden. Diese werden dazu mißbraucht, Knoten von der eigentlichen DHT fernzuhalten, mit Fehlinformationen zu versorgen, Daten auszuspionieren und das Verhalten einzelner Knoten kennenzulernen. Unter einer Partition versteht man in diesem Kontext einen Zusammenschluss bösartiger Knoten zu einem Parallelnetzwerk. Dabei arbeitet dieses Netzwerk mit dem gleichen Protokoll wie das eigentliche System. Vereinzelte Knoten aus solch einer Partition können nebenbei als Teilnehmer in der verteilten Hashtabelle agieren. Ein neuer Knoten, der sich einem bestehenden System anschliessen will, läuft nun Gefahr, in eine solche Partition integriert zu werden. Durch bestimmte vertrauenswürdige Knoten welche die Integration neuer Teilnehmer übernehmen, sollte versucht werden derartige Angriffe abzuwehren.

6.2.5 Reputationsmanagement

Im letzten Abschnitt wurden verschiedene Angriffe auf verteilte Hashtabellen und Möglichkeiten zu deren Abwehr vorgestellt. Der Angreifer war dabei ein im System integrierter Knoten welche die Kooperation verweigerte oder andere Teilnehmer mit Fehlinformationen versorgte. An dieser Stelle soll noch kurz eine weitere Technik

vorgestellt werden, die es den Knoten erlaubt, ihre Arbeit gegenseitig zu bewerten um Angreifer zu erkennen und ausschliessen zu können.

Das Verfahren beruht auf Feedback für geleistete Arbeit. Knoten, die alle Anfragen korrekt beantworten, viele Nachrichten weiterleiten oder neue Peers korrekt ins Netz einbinden, erhalten ein positives Feedback für die geleistete Arbeit. Inkorrektes Verhalten hingegen wird negativ bewertet. Solche Feedbacks sollten allerdings nach einiger Zeit ihre Gültigkeit verlieren, da Angreifer ihr Verhalten ändern können. Die Bewertung wird von dem Knoten erzeugt, der die geleistete Arbeit feststellt und zunächst lokal gespeichert. Ausgewählte Feedback-Objekte können, wenn genügend Wissen über einen Knoten vorliegt, an ausgehende Nachrichten gehängt und so anderen Teilnehmern zugänglich gemacht werden. Diese übernehmen Feedbacks natürlich nur dann, wenn sie den Absender der Nachricht vertrauen, d.h. genug positives Feedback von ihm haben. Angreifer, die über einen längeren Zeitraum das Protokoll nicht korrekt umsetzen oder das Netzwerk auf andere Art zu schädigen versuchen können so erkannt und ausgeschlossen werden. Auf der anderen Seite müssen aber auch Vorsichtsmaßnahmen getroffen werden, die verhindern, daß korrekte Knoten infolge falscher Feedbacks irrtümlich von Netz entfernt werden.

6.2.6 Zusammenfassung und Schlussfolgerung

Es wurden verschiedene Schwachstellen verteilter Hashtabellen aufgezeigt und Vorschläge gemacht diese zu beheben. Dabei wurde gezeigt, daß diese Mechanismen sich durchweg positiv auf die Sicherheit und Leistungsfähigkeit auswirken. Es hat sich herausgestellt, daß sichere Protokolle bestimmten Entwurfsprinzipien folgen müssen. Dazu zählen verifizierbare Systeminvarianten (wie verifizierbare ID's), die Möglichkeit Anfragen auf ihrem Weg zum Zielknoten mitverfolgen zu können, das Verteilen von Verantwortlichkeiten auf mehrere unabhängige Teilnehmer, das Signieren von Nachrichten und gegenseitiges Bewerten der Teilnehmer. Sie ermöglichen angreifende Knoten zu erkennen und auszuschliessen sowie die Funktion der verteilten Hashtabelle trotz Anwesenheit einiger Angreifer garantieren zu können.

6.3 Spyware

6.3.1 Einführung

Programme, die das Verhalten und die Surfgewohnheiten von Anwendern überwachen und protokollieren um daraus gewonnene Informationen an die Hersteller der Software zurückzusenden, bezeichnet mal als Spyware. Dieser Vorgang geschieht im Allgemeinen ohne Wissen und Zustimmung des Anwenders. Meistens bemerkt der Anwender noch nicht einmal, daß das Spionage-Programm auf seinem Rechner installiert ist. Nach einer Aufzeigung der Gefahren, die von solcher Software ausgehen, werden verschiedene Klassen von Spyware genauer untersucht. Daran anschliessend werden vier bekannte und weit verbreitete Spyware-Programme vorgestellt. Abschliessend wird auf die Verbreitung von Spyware eingegangen und Maßnahmen vorgestellt um Rechner schützen zu können.

6.3.2 Was Spyware so gefährlich macht

Firmen, die über das Internet Werbung verbreiten, haben ein großes Interesse zu erfahren, wen sie eigentlich umwerben. Es macht beispielsweise wenig Sinn zwölfjährigen Schülern eine Lebensversicherung zu empfehlen, da diese nicht zur Zielgruppe gehören. Statt wahllos Werbebanner einzublenden, wird also zuerst überprüft, ob der Betroffene bereits bekannt ist. Falls bereits Daten über das Verhalten des Anwenders verfügbar sind, werden Werbebanner gezielt zu den Interessen des Benutzers eingeblendet. Neben Privatanwendern können auch öffentliche Einrichtungen und Firmen die Folgen von Spyware zu spüren bekommen. Spyware im Netzwerk bedeutet für betroffene Unternehmen, daß vertrauliche Firmeninformationen problemlos von unbekannten Außenstehenden eingesehen und erfasst werden können, ohne dass das Unternehmen etwas davon merkt. Dabei steht weniger das Ausspionieren des Surfverhaltens oder Werbebanner auf Internetseiten im Vordergrund. Deutlich gefährlicher sind die entstehenden Sicherheitslücken die es Konkurrenten ermöglichen, firmeninterne Daten auszuspionieren.

Eingriffe in die Privatsphäre durch Ausspionieren von Verhaltensgewohnheiten, von Spyware verbrauchte Rechenkapazitäten und die entstehenden Sicherheitslücken sind die Hauptgefahren die von solcher Software ausgehen.

6.3.3 Kategorien von Spyware

Im Folgenden werden verschiedene Kategorien von Spyware sowie Gefahren, die von den einzelnen Klassen ausgehen, vorgestellt. Anschliessend wird auf vier weit verbreitete Spyware-Programme genauer eingegangen.

Cookies und WebBugs

Cookies sind kleine Dateien die vom Browser auf der Festplatte abgelegt werden. Sie dienen der Speicherung von Informationen, welche die Webseite bei weiteren Besuchen des Surfers abrufen kann. Eine Webseite kann beispielsweise den Namen eines Besuchers ablegen. Kehrt der Benutzer nach einigen Tagen auf diese Seite zurück, so braucht er sich nicht erneut anzumelden - die Webseite begrüßt ihn bereits auf der Startseite mit den persönlichen Einstellungen. Um jetzt mehr das Surfverhalten eines Anwenders zu erfahren, können Datenjäger in jedem Cookie eine ID zur Benutzeridentifikation hinterlegen. So weiss der Werbende stets, welcher Banner auf welcher Webseite gesehen oder gar angeklickt wurde.

Browser-Hijacking

Beim Browser-Hijacking werden die Einstellungen des Browsers derart verändert, daß beim Start des Programms Werbeseiten angezeigt, oder eingegebene Adressen zunächste auf Werbeseiten weitergeleitet werden. Hijacking nutzt in erster Linie bestehende Sicherheitslücken der verwendeten Browser und versucht gleichzeitig Einstellungen am Betriebssystem zu verändern.

Keylogger

Keylogger wurden ursprünglich entworfen um alle Tastenanschläge des Benutzers in eine Logdatei zu schreiben und so an Passwörter, Kreditkartennummern oder andere vertrauliche Informationen zu kommen. Inzwischen sind sie derart weiterentwickelt, daß auch Besuche auf Internetseiten oder das Starten von Anwenderprogrammen mitverfolgt werden kann.

Tracks

Aufzeichnungen von zuletzt besuchten Webseiten, geöffneten Dateien oder zuletzt gestarteten Programmen bezeichnet man als Tracks. Solche Aufzeichnungen richten selbst keinen Schaden an, können jedoch von Spyware ausgelesen und an Interessierte weitergeleitet werden.

Malware

Malware ist der Überbegriff für Software, die den Rechner in irgendeiner Weise schädigt oder dessen normalen Arbeitsablauf stört. Dazu zählen beispielsweise Viren, Würmer, Trojaner oder Dialer.

Adware

Es gibt zahlreiche Freeware-Produkte, die im Gegensatz zu Shareware oder kommerzieller Software kostenlos sind. Will der Programmierer nicht ganz leer ausgehen kann er in seine Software Werbebanner platzieren. Ab diesem Zeitpunkt gilt die Freeware nicht mehr als Freeware sondern als Adware(Ad = Advertisement = Werbung). Die Werbung kann an das aktuelle Surfverhalten des Anwenders angepasst werden.

Spybots

Spybots sind genau die Programme, die man zunächst mit dem Begriff Spyware in Verbindung bringt. Sie überwachen das Verhalten des Benutzers, sammeln Informationen über besuchte Webseiten, eMail-Adressen, mit persönlichen Daten ausgefüllten Formulare usw. Die Informationen werden dann an Dritte weitergereicht und können je nach Bedarf weiterverarbeitet werden.

6.3.4 Gator, Cydoor, SaveNow und eZula

Nachdem im vorherigen Abschnitt verschiedene Kategorien aufgelistet und beschrieben wurden, sollen jetzt vier bekannte und weit verbreitete Spywareprogramme in ihrer Funktionsweise genauer untersucht werden.

Gator

Gator ist Spyware für Windows-Betriebssysteme. Das Programm führt Logbücher über besuchte Webseiten und leitet diese an den Gator-Server weiter. Claria Corporation - der Hersteller von Gator - verkauft diese Informationen und ermöglicht es Werbenden gezielt Werbung einzublenden. Weiterhin werden einige Daten in der Registrierungsdatenbank unter HKEY_CLASS_ROOTCLSID gespeichert. Das Programm versucht sich beim Besuch einiger Seiten selbst zu installieren. Verschiedene File-Sharing Programme wie Kazaa, Grokster oder iMesh sind an Gater gekoppelt. Claria selbst bietet auch einige kostenlose Tools zum Download bei deren Installation Gator mitinstalliert wird.

Cydoor

Cydoor ist ebenfalls ein Spyware-Tool für Windows. Sobald eine Internetverbindung besteht läd Cydoor verschiedene Werbebanner vom Server herunter. Diese werden eingeblendet sobald eine Anwendung mit Cydoor gestartet ist, unabhängig davon ob der Benutzer online ist oder nicht. Das Tool sammelt weiterhin Informationen über besuchte Webseiten sowie demographische Daten die dann in periodischen Abständen an einen Cydoor-Server verschickt werden. Der Hersteller des Tools - Cydoor Technologies - bietet ein kostenloses Software Development Kit (SDK) an, welches benutzt werden kann, um Cydoor in jedes Windows-Programm einbetten zu können.

SaveNow

SaveNow wurde mit der Absicht entwickelt den Anwender bei Einkäufen im Internet gezielt mit Werbung zu manipulieren. Es werden zwar keine gesammelten Daten an einen zentralen Server übermittelt; allerdings werden Werbebanner vom Server heruntergeladen und lokal gesammelte Informationen dazu verwendet diese zum passenden Zeitpunkt einzublenden.

eZula

eZula manipuliert einkommende Webseiten und versieht Schlagworte mit Links zu werbenden Firmen. Die Software ist an bekannte Programme wie Kazaa oder LimeWire gekoppelt.

6.3.5 Spyware der etwas anderen Art

Software, die Informationen über den Benutzer sammelt um diese anschliessend Dritten Personen zugänglich zu machen findet sich auch außerhalb der bisher vorgestellten Umgebung. Ein weit verbreitetes Beispiel hierfür sind die automatischen Updatemanager von Windows XP und die automatische Fehlerberichterstattung.

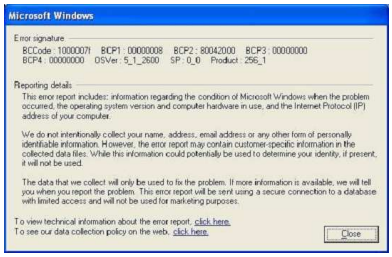

Abbildung 6.6: Windows-Fehlerreport

Abbildung 1.6 zeigt eine typische Fehlermeldung mit Informationen zu den übertragenden Daten. Mit dem Hinweis, daß die Informationen auf einer sicheren Datenbank mit eingeschränktem Zugriff gespeichert werden und daß der Bericht nicht zu Werbezwecken verwendet wird soll möglicherweise der datenschutzgerechte Umgang mit diesen Informationen suggeriert werden. Nachprüfbar ist jedoch keine dieser Aussagen.

Der über das Netz übertragene Fehlerbericht enthält auf jeden Fall folgende Informationen:

- Informationen über den Zustand der Datei zum Zeitpunkt, als das Problem auftrat
- Die Betriebssystemversion und die verwendete Computerhardware
- Die digitale Produkt-ID, die zum Identifizieren der Lizenz verwendet werden kann
- Die IP-Adresse

Es kann aber auch vorkommen, daß der Fehlerbericht kundenspezifische Informationen enthält, wie bspw. Daten aus geöffneten Dateien. Diese Informationen, falls vorhanden, können zum Feststellen der Identität verwendet werden. Microsoft gibt zwar an, diese Informationen keinen Dritten zugänglich zu machen; inwieweit dieser Garantie nachgekommen wird, ist allerdings nur schwer zu überprüfen.

6.3.6 Sicherheitslücken in Spyware

Spyware wurde mit dem Ziel entwickelt, Informationen über das Verhalten von Personen zu sammeln um so Werbenden zu ermöglichen ihre Werbung gezielt einzusetzen. Die von der Software ausgehenden Gefahren wurden bereits am Anfang dieses Kapitels kurz erwähnt. Die vielleicht größte Gefahr entsteht durch Sicherheitslücken in der Spyware selbst. Ist es einem Hacker gelungen eine solche ausfindig zu machen kann er diese dazu benutzen, Zugang zum System zu bekommen und gegebenenfalls immensen Schaden anzurichten.

Auf eine konkrete Sicherheitslücke in Gator und eZula soll im Folgenden genauer eingegangen werden. Sie wurde im Rahmen einer Untersuchung zur Verbreitung von Spyware an der University of Washington im September 2003 durchgeführt (vgl. [SSL04]). Gator hat das Problem nach Bekanntwerden inzwischen beseitigt, wohingegen die Gefahr in eZula auch weiterhin bestehen soll.

Die gefundene Schwachstelle entsteht während der Updates, die beide Programme in regelmäßigen Abständen unbemerkt durchführen. Dabei werden Daten und der Code in einem Archiv von einer zentralen Webseite heruntergeladen und installiert. Die Adresse des zu kontaktierenden Servers liegt der Spyware dabei in Form einer ausgeschriebenen URL vor, die zunächst via DNS-Anfrage in eine entsprechende IP-Adresse übersetzt werden muss. Nach dem Download wird das Archiv entpackt und im Dateisystem abgelegt. Dabei überprüft keines der genannten Programme die Echtheit der heruntergeladenen Daten. Gelingt es nun einem Außenstehenden Kontrolle über die aufgebaute TCP-Verbindung zu bekommen oder die Zuordnung zwischen dem Rechnernamen und der zugehörigen IP-Adresse zu verfälschen (sog. DNS-Spoofing), kann er ein Archiv seiner Wahl an den anfragenden Rechner schicken. Enthält dieses Archiv absolute oder relative Pfadnamen ist es leicht Dateien an irgendeiner Stelle im Dateisystem des Opfers zu platzieren. Beispielsweise könnte er eine ausführbare Datei ins Autostart - Verzeichnis des betroffenen Rechners speichern.

Eine weitere Sicherheitslücke in Gator erlaubt es Angreifern beliebige Software auf dem Zielrechner zu installieren und Kontrolle über das System zu bekommen. Dazu zählt ein Internet Explorer - Plugin zur Installation eines Passwortmanagers von Gator mit integrierter Spyware. Nähere Details hierzu finden sich auf den Internetseiten von EyeOnSecurity [1]. Es ist sehr wahrscheinlich, daß noch weitere - bisher unbekannte - Sicherheitslücken in Spyware vorhanden sind.

6.3.7 Verbreitung von Spyware

Der US-Internet-Provider EarthLink hat zusammen mit dem Software-Hersteller Webroot Software die Verbreitung von Spyware untersucht. Dabei wurde zwischen dem 1. Januar 2004 und 31. März 2004 rund eine Million Systeme überprüft und im Schnitt knapp 28 Spyware-Instanzen pro Rechner gefunden. Besonders viele Adware-Cookies wurden bei den Untersuchungen gefunden, insgesamt rund 23,8 Millionen. Hinzu kommen rund 5,3 Millionen Adware-Installationen, 184.919 Trojaner und 184.559 Tools zur System-Überwachung. Vor allem die recht hohe Zahl an Trojanern und Monitoring-Tools gibt dabei zu denken, erlauben sie doch unbefugten Zugriff auf den betroffenen Rechner oder zeichnen alle Aktionen des Nutzers auf. Eine im Selbsttest durchgeführte Statistik auf einem WindowsXP - Rechner mit Internetzugang bestätigt die Untersuchungsergebnisse von EarthLink. Mit Hilfe des Anti-Spyware-Tools Spybot Search&Destroy konnten insgesamt 29 Spywareinstanzen identifiziert werden. Ein weiterer Durchlauf auf einem Windows2000 Rechner erzielte insgesamt 39 Treffer, darunter 10 Instanzen von HitBox, 6 Instanzen von WebTrends live, 6 weitere von Advertising.com und jeweils drei von Adviva, DoubleClick und MediaPlex.

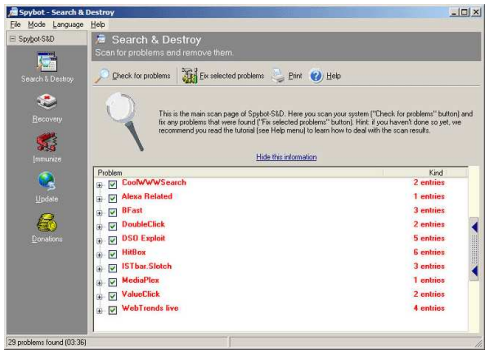

Abbildung 6.7: Spybot Auswertung

6.3.8 Was man gegen Spyware unternehmen kann

Befolgt man einige einfache Regeln, verringert sich die Gefahr, daß sich Spyware im System festsetzt. Es sollten grundsätzlich nur solche Programme installiert werden, die auch tatsächlich benötigt werden. Manche Software-Anbieter weisen in den Nutzungsbedingungen darauf hin, daß Spy- und Adware als zusätzliche Komponenten mit installiert werden. Es existieren auch ständig aktualisierte Datenbanken mit infizierter Software im Internet (www.spychecker.com). Weiterhin sind die Sicherheitseinstellungen im System den jeweiligen Gegebenheiten anzupassen. Firewalls verhindern zusätzlich, daß eine Verbindung mit dem Spyware - Hersteller aufgebaut werden kann. Es existiert eine Vielzahl an kostenlosen Tools zum Erkennen und Löschen von Spyware. Zu den bekanntesten Vertretern gehören Ad-aware

[1] http://eyeonsecurity.org/advisories

und SpyBot-Search&Destroy. Schliesslich läuf die überwiegende Mehrzahl der heute existierenden Spyware auf Microsoft Windows Betriebssysteme. Der Umstieg auf alternative Betriebssysteme bietet daher zusätzlichen Schutz.

Literaturverzeichnis

[SL04] Mudhakar Srivatsa and Ling Liu. Vulnerabilities and security threads in structured peer-to-peer systems: A quantitative analysis. 2004.

[SM02] Emit Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. 2002.

[SSL04] Steven D. Gribble Stefan Saroiu and Henry M. Levy. Measurement and analysis of spyware in a university environment. 2004.

Kapitel 7

Ad-Hoc Netzwerke

Seminarbeitrag von **Mehmet Soysal**

7.1 Einleitung

Safari ist ein Forschungsprojekt der Rice Universität Houston-Texas, und soll eine Synthese von P2P-Systemen und Ad-Hoc Netzen werden. Beide Bereiche, der Forschung, haben drastische Fortschritte in den letzten Jahren genossen.

7.2 Motivation

Das heutige Internet ist abhängig von fest verkabelten Leitungen. Die Abhängigkeit von fest verkabelter Infrastruktur macht es, technisch und ökonomisch, beinahe unmöglich alle Teile der Welt zu vernetzen. Ad-Hoc Netzwerke nutzen kabellose Technologien um einzelne Knoten miteinander zu verbinden. Als Knoten werden die einzelnen Rechner in einem Netzwerk bezeichnet, oder auch die Teilnehmer an einem Netz. Dadurch sind solche Netze kaum anfällig für Katastrophen oder gezielte Attacken, und sind ohne großen Aufwand in allen Teilen der Welt einsetzbar.

Genau hier setzt das Safari Projekt (siehe Kap. 7.6) an. Es verbindet Ad-Hoc Netze mit Peer-to-Peer (P2P) Techniken. Durch den Einsatz von P2P-Technologie wie Pastry [RD01] soll das Safari Netz auch bei sehr vielen Teilnehmern (> 10000 Knoten) gut skalieren, und soll zusätzlich grundlegende Netzwerkdienste (DNS, DHCP, Mail ...) bieten.

7.3 Übersicht über Peer-to-Peer Netze

Als "Peer-to-Peer" werden Netzwerke bezeichnet bei denen die Knoten/Teilnehmer gleichberechtigt miteinander kommunizieren. Das bedeutet, dass keiner der Knoten

die Rolle des Servers oder Klienten übernimmt.

7.3.1 Modelle von P2P Netzen

Zentrale P2P-Systeme

Zentrale P2P-Systeme, stellen keine P2P-Technologie im eigentlichen Sinne dar, denn sie verwenden einen zentralen Knoten, welcher das Netz verwaltet. Die "Arbeit" (z. B. der Datenaustausch) wird unter den einzelnen Knoten selbst erledigt. Diese Variante ist erstens nicht robust gegenüber Ausfällen, denn ohne den zentralen Knoten wäre das Netz nicht mehr funktionstüchtig, und zweitens hängt die Skalierbarkeit von der Leistungsfähigkeit des zentralen Knotens ab. Bekannte Beispiele für Zentrale Systeme wären ICQ (www.icq.com) oder Napster (www.napster.com). Ein Beispiel hierfür zeigt Abbildung 7.1. Der Austausch von Verwaltungsinformationen (rote Pfeile) geschiet zwsichen Knoten und zentraler Komponente. Der Datenaustausch (grüner Pfeil) wird ohne den zentralen Knoten abgewickelt.

Dezentrale Systeme

Alle Knoten in einem dezentralen System sind gleichberechtigt. Jeder Knoten kann die Rolle eines Servers (z. B. Daten zur Verfügung stellen) und die Rolle eines Klienten (z. B. Daten anfordern) einnehmen. Als Beispiel für ein dezentrales System wäre Gnutella [RF02] zu nennen. Jeder Knoten ist, mit einer Anzahl an anderen Knoten verbunden. Auf diese Weise wird das ganze Netzwerk aufspannt (Abb. 7.2). Der Vorteil an diesem Konzept ist, dass sich jeder Knoten entfernen lässt, ohne dass das gesamte Netz beeinflusst wird. Dieser Vorteil schafft aber ein Routingproblem, da durch Ausfall von einzeln Knoten neue Wege zum Ziel gesucht werden müssen.

Hybrid Systeme

Die hybriden Systeme sind eine Mischform von zentralen und dezentralen Systemen (Abb. 7.3), ein Beispiel eines hybriden Netzes ist z. B. Kazaa (www.kazaa.com). Die Knoten können in einem solchen System, mehrere Rollen einnehmen. Zum einen können die einzelnen Knoten eine zentrale Komponente in Bezug auf einen Teil des Netzes darstellen, oder sie nehmen nur die Rolle eines Klienten ein. Die Knoten stellen selbst fest, ob sie hinsichtlich Leistungsfähigkeit und Netzanbindung als Server geeignet sind, und ernennen sich selbst zu einem "Superknoten". Diese "Superknoten" übernehmen dann zusätzliche Aufgaben, wie z. B. Suchanfragen bearbeiten. Ein Beispiel ist in Abbildung 7.3 zu sehen. Einfache Knoten "verbinden" sich zu einem Superknoten, und senden ihre Anfragen (rote Pfeile) an zu diesem Superknoten. Die Superknoten senden die Anfragen an andere Superknoten weiter (blaue Pfeile), bis die Anfrage beantwortet werden kann. Sobald die Vermittlung abgeschlossen ist, wird die Arbeit (z. B. Datenaustausch) zwischen den Knoten direkt abgewickelt (grüne Pfeile).

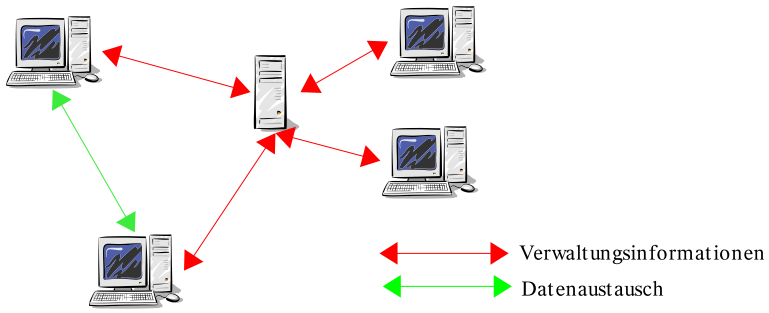

Abbildung 7.1: Zentrales P2P. Verwaltung (z. B. Vermittlung) wird von einem zentralen Knoten übernommen. Der eigentliche Datenaustausch geschieht direkt, von Knoten zu Knoten

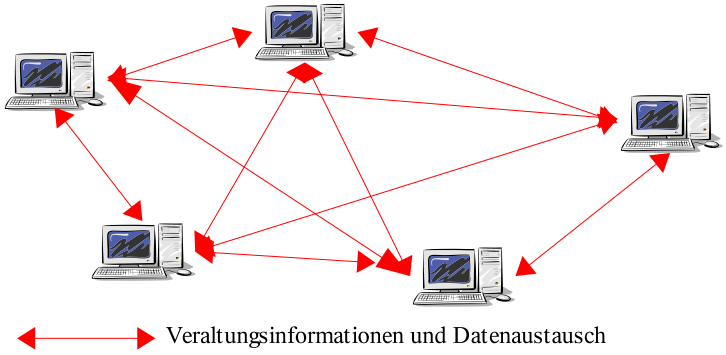

Abbildung 7.2: Dezentrales P2P. Hier werden Verwaltungsinformationen und Datenaustausch von den Knoten selbst übernommen

7.3.2 Routing mit Pastry

Wie schon erwähnt ist bei dynamisch dezentralen Netzen das Routing ein Hauptproblem. Pastry ist ein Lösungsansatz für Routingprobleme, bei dem allen Knoten dieselbe Rolle zugewiesen wird. Andere Lösungsansätze, wie sie z. B. Kazaa oder Napster verwendet werden, weisen leistungsfähigeren Knoten die Aufgabe für das Routing zu, oder das Routing wird komplett von zentralen Knoten übernommen.

Ziele von Pastry

- **Dezentralität**: Alle Knoten sind gleichberechtigt, d.?h., kein Knoten übernimmt die Rolle des Servers
- **Selbstorganisation**: Hinzufügen von Knoten muss effizient ausführbar sein. Der Ausfall einzelner Knoten darf nicht das ganze Netz beeinflussen.
- **Skalierbarkeit**: Das Netz darf, bei wachsender Anzahl an Teilnehmer, nicht überproportional langsamer werden.
- **Lokalität**: Damit das Netz effizient funktioniert, muss dabei auf Lokalität geachtet werden. Im folgenden Abschnitt wird näher auf dieses Ziel eingegangen.

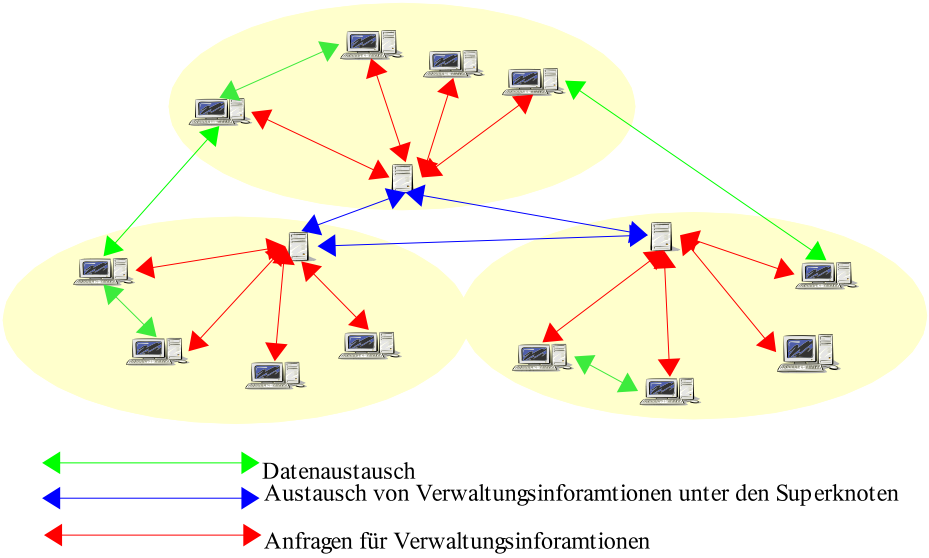

Abbildung 7.3: Hybrides P2P

Pastry-Topologie

In einem Pastry Netz haben die Knoten und Pakete eine 128 Bit lange ID. Dadurch gibt es genug Reserven um, jedem Knoten und Paket, eine eindeutige Kennung zuzuordnen. Mit Pastry werden Nachrichten gezielt verschickt, im Gegensatz zu ungezielten Versenden von Nachrichten wie es z. B. im Gnutella [RF02] Netz verwendet wird. Falls der Ziel-Knoten nicht mehr existieren sollte, so wird die Nachricht an den Knoten verschickt, dessen ID der Ziel-ID (numerisch) am nächsten ist.

Die Nachrichten werden, mittels präfixbasiertem Routing, anhand der numerischen Ziel-ID an den nächstbesseren Knoten weitergeschickt. Für die Strukturierung würden sich z. B. AVL-Bäume [Lar00] oder einfache Binär-Bäume [Fag96] eignen. Das Problem hierbei ist jedoch, dass bei AVL-Bäumen der Aufwand für das Ausbalancieren sehr aufwendig ist, dadurch würde das Netz sehr stark belastet werden. Binär-Bäume müssen nicht ausbalanciert werden, jedoch hätte der Baum, im schlechtesten Fall, eine maximale Tiefe von 128, und damit würde das Routing 128 Hops benötigen. Bei Pastry wurde eine Baumstruktur ausgewählt, bei der ein Pastry Knoten 16 Söhne hat, dadurch flacht der Baum auf eine Tiefe von 32 ab (vergl. [RD01]), und würde somit im schlechtesten Fall 32 Hops für das Routing benötigen.

Um **Lokalität** zu berücksichtigen, wird bei Pastry die “Nähe“ zwischen zwei Knoten unterschieden, 1. durch numerische “Nähe‘ der 128 Bit Adressierung und 2. durch geografische “Nähe“, welche durch die Anzahl der IP-Hops oder durch eine schnelle Anbindung gegeben ist. Diese Unterscheidung führt zu einer Aufspaltung der Routingtabelle (siehe Abb. 7.5). Die Routingtabelle besteht aus drei Teilen, der normalen Routingtabelle, der Blattmenge (im englischen Leafset genannt) und der Nachbarschaftsmenge (im englischen Neighborhoodset genannt). In der Blattmenge werden IDs gespeichert, welche mit der eigenen ID, einen langen gemeinsamen Präfix haben. In der Nachbarschaftsmenge werden IDs von verschiedenen Knoten gespeichert, welche “nah“ an den eigenen Knoten angebunden sind.

Routing mit Pastry

Das Routing mit Pastry geschieht folgendermaßen.

- Falls das Ziel in der eigenen Blattmenge enthalten ist, dann sende die Nachricht direkt an das Ziel weiter.

- Vergleiche die Länge des gemeinsamen Präfix (eigene ID und Ziel ID). Suche aus der Routingtabelle ein Knoten aus, dessen gemeinsames Präfix (ID des gespeicherten Knotens und Ziel ID) um mindestens eins länger ist, und sende Nachricht an den ausgesuchten Knoten aus der Routingtabelle.

- Falls kein passender Knoten gefunden wurde, suche einen Knoten aus der Vereinigung der Blattmenge, Routingtabelle und Nachbarschaftsmenge, dessen ID numerisch näher an der Ziel ID ist.

Ein Beispiel des Pastry Routing ist in Abbildung 7.4 zu sehen. Der Knoten, mit der ID 0112, hat eine Nachricht für Knoten 2004. Der erste Hop (Pfeil A) korrigiert die erste Stelle der Knoten ID. Also wird die Nachricht, von Knoten 0112 an einen Knoten geschickt, dessen ID numerisch näher an der Ziel ID. Dafür kommen alle Knoten in Frage, deren ID mit einer 2 anfangen, in diesem Fall wird der Knoten, mit der ID 2821 ausgewählt. Knoten 2821 korrigiert dann die 2. Stelle der ID (Pfeil B), also sendet der Knoten 2821 die Nachricht an einen Knoten mit einer ID aus dem Bereich 20xx, in diesem Fall an den Knoten 2032. Dieser korrigiert dann die 3. Stelle der ID (Pfeil C), und sendet die Nachricht an Knoten 2005. Dieser Vorgang wird solange wiederholt, bis die Nachricht am Ziel-Knoten angekommen ist oder falls der Ziel-Knoten nicht mehr existieren sollte, an den numerisch nächsten Knoten.

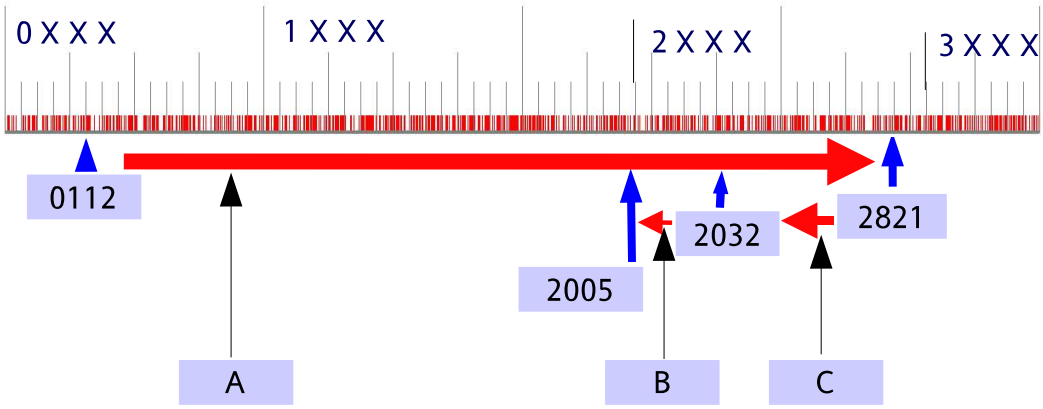

Abbildung 7.4: Pastry Routing Beispiel

7.4 Übersicht über Ad-Hoc Netze

Ad-Hoc Netze gehören zu den Netzen, welche mit kabellosen Technologien (z. B. Bluetooth [Mul01], WLAN [San01]) verbunden werden.

Es lassen sich zwei Varianten von kabellosen Netzwerken unterscheiden. Die erste Form ist die so genannte “Managed“ Form. Bei dieser Variante existieren feste Basisstationen, mit denen sich die Knoten verbinden. Unter den einzelnen Knoten gibt es keine direkte Verbindung, daher ist es kein Ad-Hoc in unserem Sinne [San01]. Als zweite Form von kabellosen Netzen gibt es dass sog. “Ad-Hoc“, bei der die Knoten untereinander kommunizieren über sog. “Multihop-Links“. Als Multihop werden Verbindungen bezeichnet, bei denen die Pakete mehrere Klienten überqueren bis sie am Ziel ankommen.

Node 10233102

Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Abbildung 7.5: Beispiel einer Pastry Tabelle

Ad-Hoc Netze werden meist selbstkonfigurierend ausgelegt, daher bedarf es keinem administrativem Eingriff bei Änderungen im Netz, sofern notwendige Informationen (Essid, MAC, Funktechnologie ...) bekannt sind. Um Technologien für Ad-Hoc Netze weiterzuentwickeln, hat sich eine Arbeitsgruppe aus der Internet Engineering Task Force (http://www.ietf.org) gebildet – MANET (http://www.ietf.org/html.charters/manet-charter.html). Die Hauptaufgabe von MANET ist es leistungsfähigere Routingalgorithmen zu entwickeln.

7.4.1 Eigenschaften von Ad-Hoc Netzen

Ad-Hoc Netze sind komplett dynamisch aufgebaut, da nicht nur die Knoten dem Netz beitreten oder es verlassen können, sondern auch sich die Knoten selbst bewegen können. Da solche mobilen Knoten von einer Batterie versorgt werden, müssen solche Netze möglichst energieeffizient sein. Der Sicherheit in solchen Netzen muss auch eine besondere Aufmerksamkeit gelten, da bei offenen Netzen fremde Knoten teilnehmen können, und die Funkstrecken anfällig gegenüber Angriffen sind.

Eines der Hauptprobleme bei Ad-Hoc Netzen bleibt aber das Routing. Bisherige Routing Verfahren, wie sie z. B. im Internet verwendet werden, sind für Ad-Hoc Netze ungeeignet. Solche Routing Verfahren sind davon abhängig, dass der “Nexthop“ (der nächste Knoten in der Routing Kette) funktionsfähig ist, und bleibt. Sollte der Nexthop nicht mehr da sein, so würde das Routing nicht mehr funktionieren, daher werden in Ad-Hoc Netzen Protokolle verwendet, welche an die Umgebung angepasst sind. Die bisher bekannten Ad-Hoc Routing-Verfahren lassen sich in 2 Kategorien einteilen, zu einem die Proaktiven und die Reaktiven Protokolle.

7.4.2 Routing Protokolle

Proaktive Routing Verfahren

Bei dem proaktiven (sog. Tabellenorientierten) Routing Verfahren haben die Knoten zu jedem Zeitpunkt die Netztopologie gespeichert (vergl. [PC97]). Die einzel-

nen Knoten verschicken periodisch so genannte Routingpakete an ihre Nachbarn. Die Routing Pakete enthalten die momentane Routingtabelle des Senders. Der Empfänger aktualisiert seine Routingtabelle und sendet wiederum seine aktualisierte Routingtabelle an seine Nachbarn. Bei vielen Änderungen in der Netztopologie werden sehr viele Routingpakete verschickt werden, um das Routing an die Änderungen anzupassen. Dadurch entstehen hohe Lasten wodurch die Skalierbarkeit des Netzes nicht mehr Gewährleistet ist. Daher sind solche Protokolle für etwas statischere Netze (mit unter 1000 Knoten), in denen sich die Knoten nicht sehr bewegen, gedacht.

Reaktive Routing Verfahren

Bei Reaktiven Routing-Verfahren wird das Routing in 2 Phasen aufgeteilt.

1. **Routenfindung:** In dieser Phase wird versucht eine Route zum Ziel aufzubauen

2. **Routenpflege:** In dieser Phase wird versucht die bestehende Route aufrecht zu erhalten

Auf diese Weise sind bei einer Änderung der Topologie nur die Rechner betroffen, welche gerade in Kommunikation miteinander sind, oder Knoten die gerade in Bewegung sind. Wie Reaktive Routing Verfahren funktionieren wird an "Dynamic Source Routing" erklärt.

Dynamic Source Routing

In der **Routenfindung** Phase sendet die Quelle eine "Routenanfrage" (in englischen Route Request oder kurz RREQ bezeichnet) an alle seine Nachbarn. Dieses RREQ ist eine Anfrage für einen Routen Weg. Im RREQ ist die Quelle enthalten, das Ziel und alle bisherigen Hops die passiert worden sind. Der RREQ hat eine eindeutige Kennung, um zu vermeiden das ein bestimmter Anfrage mehrmals weitergeleitet wird. Wenn ein RREQ bei einem Knoten ankommt, vergleicht dieser die Zieladresse mit der eigenen Adresse, falls diese übereinstimmt sendet er eine "Routenantwort" (im englischen Route Reply oder kurz RREP genannt) mit dem Pfad an die Quelle zurück. Falls der Empfänger, den Weg zum Zielknoten, in seinem Cache hat, schreibt er den restlichen Weg zum Ziel, in einen RREP und schickt ihn zur Quelle. Sollte ein Empfänger von einem RREQ, keinen Weg zum Ziel kennen so wird der RREQ an alle Nachbarn weitergeschickt. Die in dieser Kommunikation involvierten Knoten, aktualisieren dabei ihre Route Caches, da sie indirekt mithören. Dadurch werden die Routing Einträge der einzelnen Knoten auf dem aktuellen Stand gehalten, ohne dass die Knoten selbst irgendwelche RREQ starten müssen.

Sobald eine Route von der Quelle zum Ziel gefunden wurde, so wird mittels **Routenpflege** die Gültigkeit der Route zu überwacht. Wenn in einer Routingkette ein Knoten ausfällt (durch Bewegung oder Ausfall), so initiiert der Knoten, der den Ausfall bemerkt, ein "Routenfehler" (im englischen Route Error oder kurz RRER genannt) Paket. Dieser RRER wird dann zurück zur Quelle geschickt welche dann, die betreffende Route aus seinem Cache löscht, und eventuell einen neue Routenfindung startet. Dabei löschen auch die einzelnen Knoten welche von dem RRER passiert werden, die betreffenden Routingwege welche den ausgefallenen Knoten enthalten.

7.5 Vergleich zwischen P2P und Ad-hoc

7.5.1 Unterschiede zwischen P2P und Ad-Hoc

Der Hauptunterschied zwischen P2P und Ad-Hoc Netzwerken ist die Idee, die hinter beiden Netzwerken steckt. P2P Netze werden hauptsächlich benutzt um Daten, direkt von Knoten zu Knoten, auszutauschen. Ad-Hoc Netzwerke hingegen werden genutzt um mobile Netze aufzubauen, um ganze Gebiete mit Netz zu versorgen, und auf kostengünstigere Weise Netzwerke aufzubauen. Auch sind die Techniken auf denen beide Netze meist basieren unterschiedlich, Ad-Hoc Netze benutzen kabellose Techniken um die einzelnen Knoten miteinander zu verbinden, und P2P-Netze basieren auf bestehender, fester Infrastruktur. Der Austausch von Daten wird in P2P Netzen direkt zwischen den Knoten mittels Hop-by-Hop Verbindungen, abgewickelt. Hop-by-Hop Verbindungen sind direkte Verbindungen zwischen zwei Teilnehmern, ohne das andere Knoten als Zwischenstationen genutzt werden. In Ad-Hoc Netzen werden Daten über Multi-Hop Verbindungen ausgetauscht, dabei werden die Daten von Knoten zu Knoten weitergeleitet bis es am Ziel angekommen ist.

Unterschiede	P2P	Ad-Hoc
Idee dahinter	Datenaustausch	Versorgung mit Netz
Verbindungsart	direkt mit Hop-by-Hop	Indirekt über Multi-Hop
Reichweite	Um die ganze Welt	Soweit der Funk reicht
Proaktives Routing	Aufgrund der Größe nicht möglich	Bis zu einer bestimmten Größe möglich
Beweglichkeit	nicht möglich	möglich

7.5.2 Ähnlichkeiten zwischen P2P und Ad-Hoc

P2P und Ad-Hoc Netze sind selbstorganisierend und extrem robust, z. B. muss wegen einem Ausfall eines Knoten, nicht das ganze Netz neu organisiert werden. Außerdem haben beide Netze dasselbe Anfangsproblem. Wenn eine neuer Knoten dem Netz beitreten möchte, müssen diese erstmal ein Mitglied des Netzes finden. Bei P2P Netzen (z. B. Gnutella) werden solche Mitglieder über, in Listen gespeicherte IP Adressen gesucht. Sobald der neue Knoten, sich mit einem aktiven Knoten verbunden hat, ist der neue Knoten ein aktives Mitglied im Netz. Bei Ad-Hoc Netzen ist es ein wenig anders, hier muss ein Klient zuerst in die Reichweite des Netzes gelangen und danach seine Präsenz bekannt machen, und dies macht ihm zu einem aktiven Mitglied. Diese und andere Gemeinsamkeiten sind in der folgende Tabelle aufgelistet.

Gemeinsamkeiten	P2P	Ad-Hoc
Topologie	flach und alternierend	flach und alternierend zusätzlich mobil
Abhängigkeit von Knoten	nicht abhängig von einzelnen Knoten	nicht abhängig von einzelnen Knoten
Skalierbarkeit	sehr gut	Begrenzt durch die Bandbreite
Reaktives Routing	möglich	möglich

7.5.3 Fazit

Beide Netze haben Ähnlichkeiten da sie selbstorganisierend sind, jedoch basieren sie auf komplett unterschiedlichen Techniken. P2P-Netze basieren meist auf festverkabelten Leitungen, Ad-Hoc Netze hingegen basieren auf kabellosen Techniken wie z. B. WLAN oder Bluetooth. Daher eigenen sich die Netze nur für bestimmte Einsatzbereiche. Das im folgenden Kapitel vorgestellte Safari Projekt, will die Vorteile von P2P Techniken, wie z. B. Pastry, in Ad-Hoc Netze integrieren, um dadurch sehr große skalierbare Ad-Hoc Netze ($>$ 10000 Knoten) aufzubauen.

7.6 Safari Projekt

Das Safari Projekt soll eine Synthese vom P2P- und Ad-Hoc Techniken werden. Dazu müssen eine Reihe von neuen Protokollen und Techniken entwickelt werden. Noch ist nicht alles entwickelt worden und vieles existiert nur als Theorie. Zur Realisierung des Safari Projektes müssen 4 Kernprobleme gelöst werden.

1. Bisherige Ad-Hoc Routing Protokolle skalieren nur bis zu ein paar hundert Knoten. Und bieten nur einfache Netzwerkfähigkeiten. Safari soll auch bei weit über ein paar tausend Knoten, gut skalieren können, und normale Netzwerkfähigkeiten bieten (z. B. TCP/IP, DNS, Time, Mail ...)

2. Festverkabelte Netze sind abhängig von Diensten (wie z. B. Routern, DHCP, DNS). Bei Safari sollen diese Dienste über das ganze Netz verteilt werden.

3. P2P Technologien sollen integriert werden, um die Skalierbarkeit zu gewährleisten. Jedoch sind P2P Techniken nicht für Ad-Hoc Netze, und die damit verbundenen Gegebenheiten ausgelegt.

4. Es soll selbstständig feste Infrastruktur erkennen und benutzen, ohne davon abhängig zu sein.

7.6.1 Übersicht des Safari Netzes

Die Knoten in einem Safari Netz organisieren sich selbst in einer Zellenhierarchie. Jeder Knoten hat eine feste ID und eine dynamische Hierarchie ID (HID). Mit der HID wird die Position im Netz bestimmt. Die Zellen-Hierarchie wird durch das "Bojen Protokoll" aufgebaut. Durch Selbstbestimmung werden Knoten zu "Bojen" raufgestuft. Diese Bojen verschicken so genannte "Leuchtfeuer", mit denen das Netz organisiert wird. Der Begriff "Leuchtfeuer" ist aus dem englischen "Beacon" übersetzt (im weiteren Text werde ich den englischen Begriff Beacon nehmen, anstatt des deutschen Übersetzung "Leuchtfeuer"). Das Routing im Safari Netz wird in 2 Bereiche eingeteilt. In ein Proaktives Interzell Routing Protokoll um Pakete in eine andere Ziel-Zelle zu senden, und in ein Reaktives Intrazell Routing Protokoll, um die Pakete innerhalb einer Zelle an das Ziel zu schicken. (siehe Kap. 7.4.2) Die Position (HID) eines Knoten wird mit dem DHT bestimmt.

Eine Darstellung der Topologie zeigt Abb. 7.6. Die Zellen 11, 12, 13, 21, 22 sind so genannte fundamentale Zellen. In diesen fundamentalen Zellen befinden sich Level-0 Knoten (blaue Kreise) und genau eine Level-1 Boje. Die Level-0 Knoten, in einer fundamentalen Zelle, verbinden sich zur Level-1 Boje in dieser Zelle. Mehrere fundamentalen Zellen gruppieren sich zu einer so genannten "Superzelle" (Blaue Zelle 1 und rosa Zelle 2). In diesen Level-2 Zellen existieren jeweils genau eine Level-2 Boje (grüne Kreise). Alle Level-1 Bojen, der fundamentalen Zelle, verbinden sich zur Level-2 Boje in ihrer Superzelle.

7.6.2 Protokolle des Safari Projektes

7.6.3 Bojen Protokoll

Das "Bojen Protokoll" setzt sich aus zwei wichtigen Teilen zusammen. Als erstes die selbstorganisierende Hierarchie zwischen den einzelnen Knoten, welche durch "Bojen Stufen" bestimmt wird. Und als zweites die so genannten "Beacons", mit denen zunächst die Hierarchie aufgebaut, und danach die bestehende Hierarchie weiter optimiert wird.

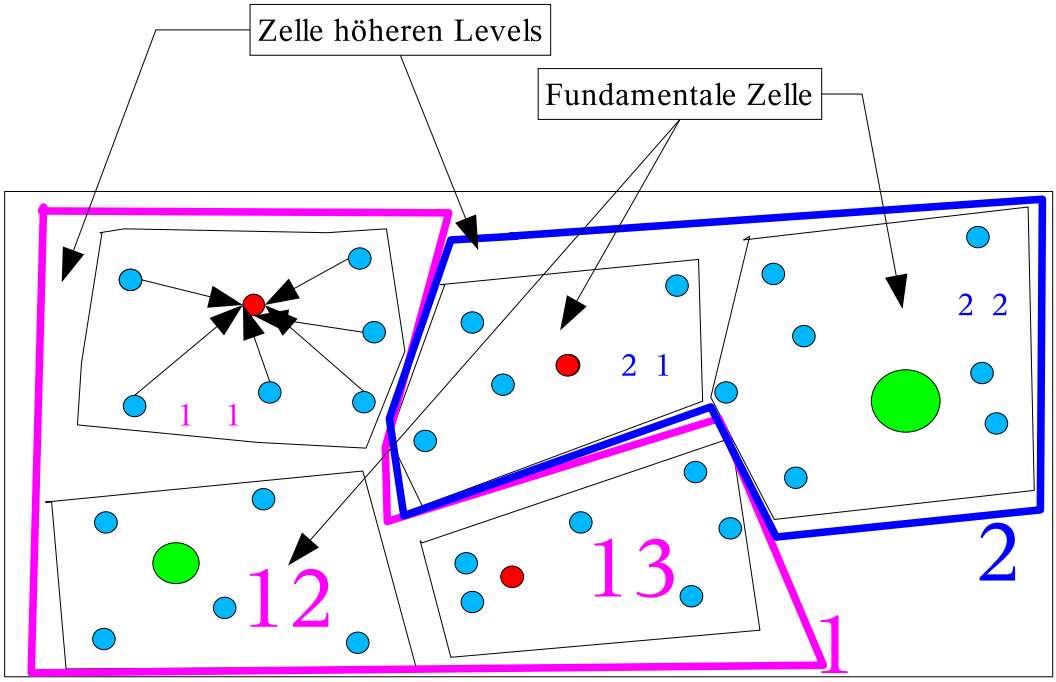

Abbildung 7.6: Darstellung der Netztopologie

Beacon

Die Beacons (Bojen Pakete) werden periodisch von den einzelnen Bojen, in einem vorgegebenen Radius, ausgesandt. Diese Bojen Pakete enthalten eine “Beacon Seriennummer“, um jedes Beacon eindeutig zu identifizieren. Des Weiteren enthalten die Beacons ID und Stufe der Sender-Boje, um sicherzustellen dass die einzelnen Empfänger Knoten die Beacons unterscheiden können. Zusätzlich werden anhand der Beacons auch Positions Informationen der einzelnen Knoten bestimmt (siehe Kap 7.6.4), und die Beacons liefern dabei auch nützliche Informationen über Intra- und Interzell Routing.

Bojen Stufen

Die Hierarchie selbst wird durch Stufen bestimmt, welche selbstständig durch die Knoten bestimmt werden. Sollte ein Knoten in das Netz eintreten, so hat er den Stufe-0. Knoten mit Stufe-0 lauschen, in einem vorgegebenen Radius, automatisch auf Beacon einer Stufe-1 Boje. Sollte ein Knoten nichts von einer Stufe-1 Boje hören, so steigt der Knoten um eine Stufe nach oben. Jeder Knoten “gehört“ zu einem Knoten (Boje) höherer Stufe, bis auf den “obersten“ Knoten, dieser gehört zu keinem Knoten/Boje. Alle Knoten gehören zu einem Knoten mit Stufe (1 .... k), wobei k die höchste Stufe im Netzwerk ist.

Alle Knoten warten auf Beacons. Für jede Stufe ist eine bestimmte Zeitspanne vorgegeben. Sollte ein Beacon empfangen werden, wird das Beacon in der Beacon Tabelle abgelegt und ersetzt den letzten Eintrag, welcher von derselben Boje geschickt wurde.

Sollte, bei einem Knoten A einer bestimmten Stufe n, eine bestimmte Zeitspanne einer höheren Stufe n+1 abgelaufen sein, so prüft der Knoten A in seiner Beacon

Tabelle ob von einem Knoten der Stufe n+1 ein Beacon empfangen wurde. Sollte ein neues Beacon vorhanden sein, so wird der Timer zurückgesetzt. Wenn nicht, erhöht Knoten A seine Stufe auf n+1.

Für den Fall dass ein Knoten A irrtümlich seine Stufe erhöht (z. B. weil er keinen Funkkontakt hat), speichert jeder Knoten noch den letzten Stand in seinem Cache, bevor er seine Stufe erhöht. Sollte Knoten A mit Stufe n ein Beacon von einem anderen Knoten B mit Stufe n erhalten, welcher in seinem Radius ist, so steigt Knoten A auf Stufe n-1 ab. Sollte ein Knoten auf Stufe n, nichts von einem anderen Knoten der Stufe n-1 hören, so steigt die Boje auch um 1 ab. Alle Knoten welche zu einer Boje mit Stufe-1 gehören, bilden die so genannte "fundamentale Zelle" (siehe Abb. 7.6).
Knoten mit höherer Stufe (Bojen) haben keine spezielle Aufgabe beim Routing von Paketen oder Auffinden von Ziel-Knoten, sie dienen nur zur Verwaltung des gesamten Netzes. Diese Bojen verschicken, mit den Beacon Informationen an alle Knoten im Netz, mit denen die Knoten ihre Arbeit verrichten können. Solche Informationen werden nur von den Bojen verschickt, ansonsten würden die Informationen, über das Netz, von jedem Knoten einzeln verschickt werden, was eine sehr hohe Last bedeuten würde. Sollte eine der Bojen ausfallen, so wird seine Stelle von einem anderen Knoten übernommen aus seiner Zelle übernommen, da alle Knoten, in der gleichen Zelle, dieselben Informationen haben.

Buoy Ad-Hoc Route Table (BART)

Jeder Knoten hat eine eigene "Buoy Ad-Hoc Route Table", die er durch die Bojen Broadcasts auf dem aktuellen Stand hält. Jeder Knoten wertet einen Beacon aus, auch wenn der Beacon aus einer fremden Zelle stammt. Die aus den Beacon gesammelten Routinginformationen werden im BART abgelegt.

7.6.4 Adressen Bestimmung

HID

Die HID ist eine Zahl, welche die Position eines Knoten in der Hierarchie bestimmt. Sollte die Hierarchie Stufe k haben, so hat die HID auch k Ziffern. Die erste Zahl (rechte Zahl) bestimmt die Stufe-1 Boje, und die zweite Zahl den Stufe-2 Boje .... Nehmen wir an, dass das Netz Stufe-3 hat, also hat die HID auch 3 stellen, z. B. HID = 124. Die 1 bestimmt die Stufe-3 Boje, die 2 bestimmt die Stufe-2 Boje und die 4 bestimmt die Stufe-1 Boje. Alle Knoten, die zur selben Stufe-1 Boje gehören, haben dieselbe HID.

Adress-Auflösung

Da beim Versenden von Paketen die ungefähre Position eines Ziel zu bestimmen ist, gibt es im Safari Netz einen "Verteilten Adress-Auflösungs" Dienst. Dieser Dienst ermittelt anhand der Knoten ID seine mögliche HID im Netz. Ein solcher Dienst muss sehr effizient und tolerant gegenüber Ausfällen sein. Um die Bedürfnisse eines Ad-Hoc Netzes zu befriedigen, muss dabei auf Lokalität geachtet werden, so dass ein Knoten A, zur Bestimmung eines benachbarten Knoten B, nicht einen entfernten Knoten C befragen muss.
Jeder Knoten hat eine feste ID und eine HID, welche seine Position in der Hierarchie bestimmt. Die Auflösung ID auf HID wird in einem DHT über dem Netzwerk auf folgende Weise gespeichert: Jeder Knoten A hascht das Tupel (ID, Koordinaten) k-mal mit k-verschiedenen Haschfunktionen, und ermittelt dadurch eine neue HID. Bei den Knoten B deren HID numerisch am nächsten bei der gehaschten HID ist,

werden dann das Tupel (ID, Koordinaten) abgelegt. Diese Knoten B sind dann die "Lokalisierungs Knoten" von A. Da die HID nur die fundamentale Zelle bestimmt, bekommen alle Knoten in dieser Fundamentalen Zelle, die nötigen Informationen um die Position des Knotens A zu bestimmen.
Dadurch ist aber nicht die Lokalität gegeben, denn wenn ein Knoten A die Position eines nahen Knotens B bestimmen will, so befragt Knoten A einen der Lokalisierungs Knoten von Knoten B. Da Lokalisierungs Knoten zufällig (per Hasch Funktion) über das ganze Netz verteilt werden, kann es passieren, dass Knoten A einen entfernten Knoten C befragen muss, obwohl Knoten B in der Nähe ist. Deshalb werden zusätzlich das Tupel (ID, Koordinaten), von Knoten B, bei nahen Knoten D abgelegt. Diese Knoten D werden, durch die HID des Knotens B bestimmt. Die letzten i (i = 1.....k-1) Stellen der eigenen HID werden durch zufällige Hasch-Werte ersetzt. An die Knoten D mit dieser HID wird zusätzlich das Tupel (ID, Koordinaten) abgelegt. Zum Beispiel, ein Knoten A hat HID=145, dann wird die letzte Stelle (die 5) durch zufällige Werte geändert, z. B. resultiert dann eine HID=149. Bei allen Knoten, die diese neue HID haben, werden dann das Tupel (ID, Koordinaten) gespeichert.

7.6.5 Routing bei Safari

Aus den Informationen welche von durch das Bojen Protokoll verbreitet werden, soll von dem Safari Team ein neuartiges Routing-Verfahren entwickelt werden, genannt Ad-Hoc Scalable Overlay Routing (ASOR). ASOR soll ein Mischsystem aus Proaktiven Interzell Routing und Reaktiven Intrazell Protokollen werden.

Proaktive Interzell Routing

Wenn ein Knoten S ein Paket an Knoten D zu verschicken hat, muss Knoten S zuerst die Position von D herausfinden. Dies geschieht mittels Adressauflösung (siehe Kapitel 7.6.4). Sollte das Ziel in derselben Zelle liegen, wird Intrazell Routing benutzt, andernfalls wird das Interzell Routing Verfahren verwendet.
Um ein Paket mittels Interzell Routing, zum Ziel zu senden, ermittelt der Sender zuerst die Ziel-HID. Sobald die Ziel-HID bestimmt wurde, schreibt der Sender den HID in den Kopf des Paketes. Danach untersucht der Sender seine BART, um nach einem Beacon zu suchen, welche von der Boje stammt und dessen HID an der höchsten (linken) Stelle mit der Ziel-HID übereinstimmt. Danach schickt der Sender das Paket in Richtung der betreffenden Boje. Sobald das Paket bei einem Knoten ankommt, dessen HID an der höchsten Stelle übereinstimmt, wird das Paket in Richtung der Boje geschickt dessen HID an der zweit höchsten Stelle übereinstimmt. Diese Prozedur wird solange wiederholt bis das Paket bei einem Knoten der fundamentalen Zelle ankommt. Sobald das Paket in der fundamentalen Zelle angekommen ist, wird das reaktive Routing Verfahren eingesetzt.

Dynamic P2P Source Routing

Innerhalb der Zelle wird ein Reaktives Routing Protokoll eingesetzt (siehe Kap. 7.4.2), jedoch integriert Safari das P2P Protokoll Pastry (siehe Kap. 7.3.2) um die Routensuche zu minimieren. Jedoch sind all diese Protokolle für das Internet ausgelegt, und daher nur schwer in Ad-Hoc Netzen einsetzbar. Deshalb wird vom Safari-Team **DPSR** [YCHD03] benutzt, welches Pastry schon nahtlos in "Dynamic Source Routing" integriert hat.
Das Routing bei DPSR ist dem Routing bei Pastry sehr Ähnlich. Zuerst wird in der Routing Tabelle oder Blattsatz nachgeschaut ob die Route zum Ziel enthalten ist. Sollte dies der Fall sein so wird die Route benutzt. Falls nicht wird eine

“Routenfindung“ (siehe 7.4.2) gestartet, welche den nächsten Hop ermitteln soll.

7.6.6 Netzwerk Dienste

Natürlich soll das Safari Netz auch grundlegende Netzwerkdienste bieten, (z. B. DNS, Authentifikation, E-Mail ...) wie sie aus klassischen Netzen bekannt sein sollten. Jedoch sind solche Dienste sind für Hochleistungsnetze des Internets ausgelegt, und sind ohne weiteres nicht in Ad-Hoc Netzen benutzbar. In Ad-Hoc Netzen ist die Bandbreite nicht mal annähernd so groß? wie im Internet, deshalb müssen diese Dienste den Bedingungen in Ad-Hoc Netzen angepasst werden.
Die Idee eines verteilten DNS Dienstes wurde kürzlich schon erforscht [CMM02], und wird vom Safari-Team in ihr Projekt integriert. Da Teilnehmer an so einem Netz auch gerne gemeinschaftliche Anwendungen nutzen möchten, wie z. B. E-Mail, Instant Messenger u.s.w ..., müssen auch solche Dienste den Bedingungen in Safari angepasst werden. Solche Anwendungen in einer verteiltem Form zu nutzen wurden schon teilweise entwickelt, z. B. POST [MPR+03]. POST ist in der Lage gemeinschaftliche Anwendungen, sicher und effizient anzubieten. Die Entwickler planen POST, auf 2 Arten, in Safari zu integrieren. Zu einem als verteilten Dienst, und zusätzlich soll es auch möglich sein, POST in einem klassisch Serverbasierten Dienst zu nutzen. Wie es genau implementiert werden soll, ist noch nicht genau genannt worden. Aber als Knoten für die Serverbasierte Form, sollen nur Knoten in Frage kommen, welche Verbindung zu fester Infrastruktur haben. Und damit kommen wir zum letzten Ziel des Projektes, dem Ausnutzen von bestehender fester Infrastruktur.

Ausnutzen von fester Infrastruktur

Als letzten Punkt des Projektes soll noch feste Infrastruktur in das Safari Netz integriert werden. Z.B. könnten nach einer Naturkatastrophe, übergebliebene Bereiche mit fester Infrastruktur, durch kabellose Verbindungen miteinander verknüpft werden. Eine derartige Technik wurde schon entwickelt [MBJ99]. Jedoch ist diese Technik nur für kleinere Netze geeignet, und daher für das Safari Projekt ungeeignet. Daher will das Safari Team bessere Mechanismen entwickeln.

7.7 Fazit

Das Safari Team hat mit diesem Projekt ein Design vorgestellt, welches auch sehr große Netze problemlos Verwalten kann. Das Team hat eine selbstorganisierende Hierarchie entworfen und es mit der Hybriden Routing Protokoll gekoppelt, wobei das Routing Protokoll aus einem reaktiven und proaktiven Teil besteht. Diese Protokolle wurden vom Safari Team in Simulationen ausgewertet, und es hat sich herausgestellt, dass die Protokolle, eine große Anzahl von Knoten verwalten können. Ob das Projekt irgendwann verwirklicht werden kann, bleibt noch offen. Jedoch zeigt es neue Ideen und Techniken, welche sich bestimmt in anderen Projekten wiederfinden werden.

Literaturverzeichnis

[CMM02] Russ Cox, Athicha Muthitacharoen, and Robert Morris. Serving DNS using Chord. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.

[Fag96] Rolf Fagerberg. Optimal Rebalancing of Binary Search Trees. ALCOM-IT Technical Report TR-063-96, Aarhus, 1996.

[Lar00] Kim S. Larsen. AVL Trees with Relaxed Balance. *Journal of Computer and System Sciences*, 61(3):508–522, 2000.

[MBJ99] D. Maltz, J. Broch, and D. Johnson. Experiences designing and building a multi-hop wireless ad hoc network testbed, 1999. Experiences designing and building a multi-hop wireless ad hoc network testbed. Technical Report CMU-CS-99-116, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[MPR+03] Alan Mislove, Ansley Post, Charles Reis, Paul Willmann, Peter Druschel, Dan S. Wallach, Xavier Bonnaire, Pierre Sens, Jean-Michel Busca, and Luciana Arantes-Bezerra. Post: A secure, resilient, cooperative messaging system. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*, Lihue, Hawaii, May 2003.

[Mul01] Nathan J. Muller. Bluetooth : Die Referenz für den neuen Bluetooth-Standard ; Protokoll, Architektur, Datenaustausch ; Sicherheitsmodelle der Bluetooth-Spezifikation], 2001.

[PC97] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM (3)*, pages 1405–1413. IEEE Computer Society, Washington, DC, USA, 1997.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, Melbourne, Australia and St. Petersburg, Russia, 2001.

[RF02] M. Ripeanu and I. Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer systems, 2002. Matei Ripeanu and Ian Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems.

[San01] Asunción Santamaria. *Wireless LAN standards and applications*. Artec House INC., 2001.

[YCHD03] Himabindu Pucha Y Charlie Hu and Saumitra M Das. Exploiting the Synergy between Peer-to-Peer and Mobile Ad-Hoc Networks. In *In Proceedings of HotOS-IX: Ninth Workshop on Hot Topics in Operating Systems*, 2003.

Kapitel 8

Inhaltsbasierte Netzwerke

Seminarbeitrag von **Timo Reimann**

Diese Seminararbeit untersucht im ersten Abschnitt (8.1) das Konzept des inhaltsbasierten Netzwerkverkehrs (engl. *content-based networking*), das im Gegensatz zu konventionellen Techniken nicht mit expliziten numerischen Zieladressen arbeitet, sondern den Nachrichteninhalt in einer Prädikatendarstellung als Routinginformation verwendet. Neben der Präsentation der Grundprinzipien sollen die notwendigen Voraussetzungen für eine funktionierende Wegewahl erarbeitet werden [CW03].

Der zweite Ausarbeitungsabschnitt (8.2) betrachtet die Konstruktion eines P2P-Systems mithilfe eines semantischen Overlays, das eine effiziente Suche im Netzwerk durch eine enge Kopplung der Inhalte von Daten und ihrer Anordnung in einem kartesischen Raum ermöglichen soll. Die Suchfunktionalität wird dabei genauso wie die Transformation allgemeiner Information in ein für die Verarbeitung günstiges Format im Vordergrund stehen [TXD03].

8.1 Inhaltsbasierter Netzwerkverkehr mit Prädikatendarstellung

8.1.1 Einführung

Ein inhaltsbasiertes Netzwerk (kurz: *IBN*) wird durch ein Overlay auf Anwendungsebene, das aus einer Menge von verbundenen Klient- und Router-Knoten besteht, beschrieben. Der entscheidene Unterschied zu einem traditionellen Netzwerk ist das Kommunikationsmodell: Datenpackete in einem IBN finden ihren Weg von Sender zu Empfänger nicht auf herkömmliche Weise durch die Angabe einer numerischen Adresse (wie beispielsweise IP- oder MAC-Nummer) – stattdessen fungiert die Nachricht eines Pakets nicht nur als tatsächlich zu übermittelnde Information, sondern repräsentiert gleichzeitig die Adresse eines Knoten, womit der Netzwerkfluss durch

den Inhalt einer Nachricht bestimmt wird. Um dieses Dienstmodell ermöglichen zu können, muss jeder Empfänger seine möglichen „Interessen“ in Form sogenannter *Auswahlprädikate* angeben. Die tatsächliche Zustellung einer gesendeten Nachricht erfolgt an alle Empfänger des Netzwerkes, deren Auswahlprädikate mit der Nachricht korrespondieren.

Eine Nachricht wird als Menge von *(Attribut, Wert)*-Paaren gebildet. Beispielsweise könnte eine Nachricht, die eine Warnung über einen gravierenden Hardwarefehler auf einem Webserver bekannt gibt, folgendermaßen aussehen:

[class=„alert“, severity=6, device-type=„web-server“,
alert-type=„hardware-failure“]

Dagegen setzt sich ein Auswahlprädikat als logische *Disjunktion* von *Konjunktionen*[1] elementarer *Bedingungen* (engl. *constraints*) zusammen. Eine Bedingung wiederum ist ein Quadrupel der Form *(Typ, Name, Operator, Wert)*, und mehrere Konjunktionen von Bedingungen werden unter dem Begriff *Filter* zusammengefasst. Das folgende Beispiel für ein Prädikat, das obige Nachricht empfängt, soll die verwendete Struktur klarstellen:

[(alert-type=„intrusion“ $\wedge$ severity $>$ 2) $\vee$ (class=„alert“ $\wedge$
device-type=„webserver“)]

Offensichtlich korrespondieren die auf der linken Seite des ODER-Operator konjunktiv zusammengesetzten Bedingungen nicht mit der Nachricht, der rechte Teil hingegen trifft zu. Damit definiert man die Bedingung für die erfolgreiche Zustellung einer Nachricht: Sobald die Auswertung eines Filters innerhalb eines Prädikats den Wert *wahr* liefert, wird die Nachricht dem Empfänger zugestellt.

8.1.2 Anwendungsgebiete

Nach dem Überblick aus Abschnitt 8.1.1 stellt sich die Frage nach der Anwendbarkeit: in welchen Bereichen können IBN Vorteile gegenüber konventionellen Netzwerktypen auspielen?

Der offensichtliche Vorteil ist die Aggregation von Adressen und gewünschten Informationen: Durch die Verwendung von Prädikaten, die neben den „Interessen“ gleichzeitig eine Identifikationsrolle übernehmen, ist es für zwei Knoten, die in Kommunikation treten möchten, nicht länger notwendig, Adressen a priori in Erfahrung zu bringen. Die Mechanismen des IBN garantieren die korrekte Nachrichtenzustellung zwischen Sender und einem oder mehreren Empfängern.

Daher könnten einige sinnvolle Anwendungsbereiche lauten:

- File-Sharing Systeme
- „Intrusion Detection“-Systeme
- System-Monitoring/-Management
- elektronische Auktionen
- verteilte Computerspiele

[1] natürlich sind auch alternative Formen der Prädikatenbildung denkbar, wie z.B. die konjunktive Normalenform. Dies wird jedoch nicht Gegenstand dieser Ausarbeitung sein.

8.1.3 Wegewahlverfahren und Topologie

Um Informationen von einem Sender zu allen relevanten Empfängern verteilen zu können, ist es notwendig, ein angepasstes Wegewahlverfahren zu bestimmen, das die besondere Struktur eines IBN widerspiegelt. Dazu wird zunächst eine Methode, die logische Punkt-zu-Punkt Verbindungen über einem existierenden physischen Netzwerk realisiert (*Routing*), untersucht. Danach wird die Weiterleitung von Informationen zwischen direkt verbundenen Knoten, was auch als *Forwarding* bezeichnet wird, betrachtet. Genau wie in traditionellen Netzwerken sorgen ausgezeichnete Knoten, die *Router*, für die korrekte Weiterleitung von Nachrichten.

8.1.4 Routing

Ausgangspunkt des Routing-Algorithmus ist das sogenannte *Broadcast*-Verfahren. Darunter versteht man im Allgemeinen den Versand einer einzelnen Nachricht an sämtliche erreichbaren Empfänger innerhalb eines (Sub-)Netzes (wie z.B. das Streaming von Videodaten in einem LAN). Die Notwendigkeit dafür ergibt sich aus der Tatsache, dass die Menge der Zielknoten einer Nachricht erst zur Sendezeit durch einen Abgleich zwischen Nachricht und Prädikaten festgelegt wird, somit also potentiell alle Empfänger eines Netzes erreicht werden müssen. Natürlich wäre ein Broadcast viel zu allgemein für eine differenzierte Nachrichtenzustellung: wie später erklärt wird, dient eine zusätzliche Filterung dem Zweck, nur tatsächlich empfangsberechtigte Knoten zu ermitteln. Dieser Sachverhalt wird in Abbildung 8.1 grafisch verdeutlicht.

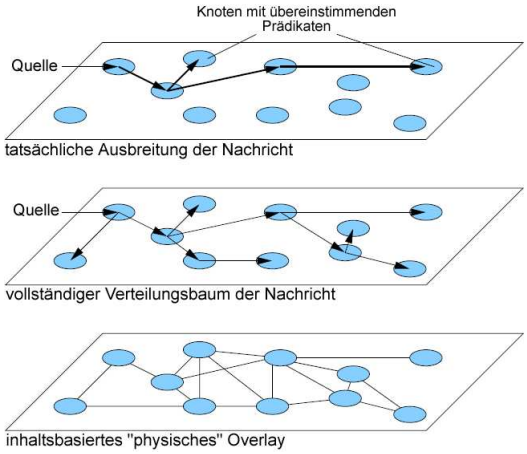

Abbildung 8.1: **Netzwerk-Overlay und Routing-Schema**

Router verwenden für das Routing-Schema zwei Protokolle: Ein *Broadcast Routing*-Protokoll sowie ein *inhaltsbasiertes Routing*-Protokoll. Ersteres enthält topologische Informationen über das physische Netzwerk und ist für die Umsetzung des oben beschriebenen Broadcast-Mechanismus verantwortlich. Letzteres sorgt dafür, dass nicht alle Netzwerkteilnehmer jede Nachricht erhalten, indem Router anhand ihrer Prädikate entscheidet, ob eine propagierte Nachricht sie passieren darf oder nicht. Eine Weiterleitung findet also genau dann nicht statt, falls es keinen Knoten „hinter“ dem Router gibt, der Interesse an der Nachricht zeigt. Die Verbindungen eines Routers zu seinen angrenzenden Knoten bezeichnen man als *Schnittstelle*.

Da Knoten flexibel einem Netzwerk hinzugefügt und entfernt werden sollen, benötigt das inhaltsbasierte Routing-Protokoll Methoden, um routingrelevante Informationen regelmäßig auf einen aktuellen Stand zu bringen. Zu diesem Zweck

wird ein „push“-Mechanismus, dem sogenannte (engl. *receiver advertisements*) zugrundeliegen, von einem „pull“-Mechanismus unterschieden, der mithilfe von *Senderanfragen* (engl. *sender requests*) und *Aktualisierungsmeldungen* (engl. *update repies*) arbeitet.

Empfängerankündigungen (EA) werden in regelmäßigen Abständen und/oder sobald neue inhaltsbasierte Adressen entstehen von Knoten ausgegeben. Dabei dient eine EA, die die inhaltsbasierte Absenderadresse enthält, dem Zweck, den Absender (als potentieller Empfängerknoten) allen Routern bekannt zu machen. Um dies zu ermöglichen, wird eine EA zunächst via Broadcast ausgegeben; jeder Router, bei dem diese eintrifft, fügt sie der Schnittstelle, die zum Absender zurückführt, durch eine logische Disjunktion hinzu. Falls es hierbei zur Bildung eines neuen Prädikats kommt, wird die EA weiterpropagiert. Anderenfalls bricht die Verteilung ab, da sämtliche Router im weiteren Verlauf bezüglich der Absenderadresse auf dem aktuellen Stand sein müssen. Eine ausgegebene Nachricht eines beliebigen Senders würde ab sofort den Absender der EA erreichen, falls er empfangsberechtigt wäre.

Durch das Verfahren der EA können ergänzende Routing-Informationen einem bestehenden Netzwerk im Betrieb hinzugefügt werden. Bricht jedoch ein Knoten aus dem Netzwerk aus, kann diese Information nicht durch eine EA bekanntgegeben werden, da die Informationsverknüpfung in Form einer logische Disjunktion stets nur erweiternd sein kann. Um auch einschränkende Informationen zu verteilen, werden regelmäßig Senderanfragen (SA) zwischen Routern ausgetauscht. Auch hierbei fließt zunächst eine SA eines Absenders (einem Router) durch ein Broadcast in das Netzwerk. Trifft diese auf einen weiteren Router, antwortet dieser mit einer Aktualisierungsmeldung (AM), die in umgekehrter Richtung zum Absender versandt wird und folgenden Inhalt hat: Handelt es sich bei dem antwortenden Router um einen Blattknoten des Netzwerks, umfasst die AM lediglich seine inhaltsbasierte Adresse. Mittelbare Router dagegen kombinieren ihre eigene inhaltsbasierte Adresse in einer logischen Disjunktion mit Adressen aus AM anderer Router, die sich „abwärts“ (downstream) dieses Routers befinden. Auf diese Art und Weise erhält der Absender der SA pro Schnittstelle eine AM mit den kombinierten Adressen aller Router, die durch diese Schnittstelle erreichbar sind.

8.1.5 Forwarding

Damit ein konkreter Router die Entscheidung treffen kann, an welche Schnittstellen eine bestimmte Nachricht weitergeleitet werden soll, bedarf es eines Weiterleitungsprozesses, der aus der sogenannten *Broadcast-* sowie der *inhaltsbasierten Weiterleitung* gebildet wird.

Ohne Beschränkung der Allgemeinheit treffen wir die Annahme, dass eine gegebene Broadcast-Weiterleitungsfunktion für eine Nachricht m von einem Absenderknoten s eine Menge von Ausgabeschnittstellen B zurückgibt. Weiterhin gehen wir von der Existenz einer *inhaltsbasierten Weiterleitungstabelle* aus, die im Rahmen des inhaltsbasierten Routing-Protokolls verwaltet wird. Diese Tabelle bildet von Schnittstellen auf Prädikate ab, indem sie für eine Schnittstelle i ein Prädikat p_i, das die Vereinigung aller Prädikate der Knoten hinter der Schnittstelle i repräsentiert, ermittelt. Analog zum Routing stützen wir für uns auch für das Forwarding auf den Broadcast-Mechanismus, um danach mit einer einschränkenden inhaltsbasierten Weiterleitung nur auf die zulässigen Schnittstellen verweisen zu können. Abbildung 8.2 skizziert beispielhaft eine inhaltsbasierte Weiterleitungstabelle.

Mithilfe dieser Komponenten ist es nun möglich, eine inhaltsbasierte Weiterleitungsfunktion IBF zu definieren, die als Parameter drei Werte entgegen nimmt: Eine Nachricht m, eine Menge von (Broadcast-)Ausgabeschnittstellen B sowie eine inhaltsbasierte Weiterleitungstabelle $T = \{p_1, p_2, \ldots, p_I\}$, wobei I die Gesamtzahl der Schnittstellen kennzeichnet. Die Funktion berechnet eine Teilmenge von B,

die aus Schnittstellen in T besteht, welche eine Prädikatenübereinstimmung mit m haben. Eine formale Darstellung lautet:

$$\text{IBF}(m, B, T) = \{i : i \in B \wedge matches(p_i, m)\}$$

Für die effiziente und performante Nutzung eines IBN gilt es, diese Funktion zu optimieren.

I_1	$f_{1.1}$	*string* $dest =$ *int* $price < 500$
	$f_{1.2}$	*string* $stock = DYS$ *int* $quantity > 1000$ *int* $price < 500$
I_2	$f_{2.1}$	*string* $airline = UA$ *string* $orig = Chicago$ *string* $dest = Atlanta$
	$f_{2.2}$	*string* $dest = New\ York$ *int* $price < 200$
	$f_{2.3}$	*string* $orig = Chicago$
	$f_{2.4}$	*string* $airline = UA$ *bool* $upgradeable = true$
I_3	$f_{3.1}$	*string* $stock = MSFT$ *int* $price < 200$

Abbildung 8.2: **Beispiel einer Weiterleitungstabelle**

8.1.6 Zusammenfassung: inhaltsbasierte Netzwerke

Dieser erste Abschnitt zeigte, wie sich Daten durch die Verwendung von Prädikaten in einem wohldefinierten Format derart strukturieren lassen, dass eine Adressierung anhand semantischer Informationen möglich wird. Notwendig dafür sind zusätzlich Modifikationen wie beispielsweise im Bereich der Wegewahl, die durch Bereitstellung von angepassten Router-Einheiten eine dynamische Teilnehmerstruktur ermöglicht. Somit lässt sich die inhaltsbasierte Netzwerktopologie in vielen Anwendungsfälle nutzen.

8.2 P2P-System mithilfe eines semantischen Overlays

8.2.1 Einführung

Im zweiten großen Abschnitt 8.2 dieser Ausarbeitung soll versucht werden, den Aufbau eines Peer-to-Peer-(P2P-)Systems unter Hinzunahme eines semantischen Overlays zu vermitteln. Eine mit dem Inhalt der abzuspeichernden Daten[2] korrelierende Verteilung derjenigen im Speicherraum wird ausschlaggebend für die Implementierbarkeit eines effizienten Suchalgorithmus sein.

Dieser Abschnitt wird ebenso wie der vorhergehende den Begriff des *inhaltsbasierten Netzwerkes* benutzen. Wichtig für den Leser ist, dass dieser in seiner Bedeutung absolut nichts mit dem zuerst erläuterten Konzept des inhaltsbasierten Netzwerkes zu tun hat, sondern allein Bezug auf das des sogenannten *Content-Addressable Network* nimmt. Nur aus Gründen der sinnvollen Übersetzbarkeit

[2]Grundsätzlich ist die Form der Daten beliebig. Beispielhaft soll im weiteren Verlauf jedoch von zu indizierenden Textdokumenten ausgegangen werden.

bleibt es jedoch bei dieser identischen Terminologie. Für weitere Informationen bezüglich Content-Addressable Networks siehe die entsprechende Ausarbeitung eines anderen Seminarteilnehmers sowie [RFH+01].

8.2.2 Traditionelle Leistungsschranken

Ein fundamentales Problem, das die Suche in traditionellen P2P-Systemen in Performanz beschränkt, ist die Tatsache, dass Dokumente nicht strukturiert, sondern beliebig in einem Netzwerk verteilt sind. Für eine effiziente Suche bedarf es in solchen Systemen daher einer Ausweitung auf möglichst viele Knoten, um das Risiko, relevante Dokumente zu übersehen, minimieren zu können. Andererseits entstehen mit dieser Strategie schnell Defizite in Bezug auf Skalierbarkeit, die gerade für dynamische P2P-Systeme einen entscheidenen Erfolgsfaktor darstellt.

Zur Überwindung dieser Problematik werden Dokumente in einem inhaltsbasierten Overlay derart abgebildet, dass sich zusammengehörige, semantisch ähnliche Dokumente stets in der gleichen Suchregion innerhalb des Netzwerkes gruppieren, oder anders formuliert: Je semantisch verschiedener zwei Dokumente sind, desto größer ist ihr Abstand im Netzwerk. Der sich daraus ergebene Vorteil ist, dass der Umfang des Suchraumes für eine gegebene Anfrage beschränkt ist: eine Ausweitung der Suche auf möglichst viele Knoten ist überflüssig, da sich ausserhalb der relevanten Suchregion nur relativ irrelevante Dokumente befinden.

8.2.3 Repräsentation von Dokumenten

Zur Implementierung einer effizienten Suche wird ein inhaltsbasiertes Netzwerk benutzt. Dazu stelle man sich den semantischen Suchraum als kartesischen Raum vor, dessen Punkte als Schlüssel oder semantische Vektoren, die wiederum direkt oder indirekt auf Dokumente verweisen, bezeichnet werden. Durch ein noch zu erläuterndes Verfahren können beliebige Dokumente (und damit auch Suchanfragen) auf Schlüssel abgebildet und in den kartesischen Raum eingeordnet werden, wodurch man den Suchprozess ableiten kann: Für eine Suchanfrage wird der zugehörige kartesische Punkt lokalisiert und innerhalb eines Radius r um diesen alle Dokumente als Ergebnis zurückgeliefert. Da außerhalb der Suchregion (aufgrund des Zusammenspiels zwischen Semantik und Distanz) die Relevanz der gefundenen Dokumente rapide abnimmt, bilden die innerhalb des konstanten Radius ermittelten Dokumente das Ergebnis. Abbildung 8.3 stellt die räumliche Vorstellung nochmals dar.

Bevor die Details des Suchalgorithmus untersucht werden, stellt sich zunächst die Frage nach der Transformation von Dokumenten zu Schlüsseln. Dies geschieht in Zwischenstufen über zwei Verfahren, dem *Vector Space Model* (VSM) und dem *Latent Semantic Indexing* (LSI).

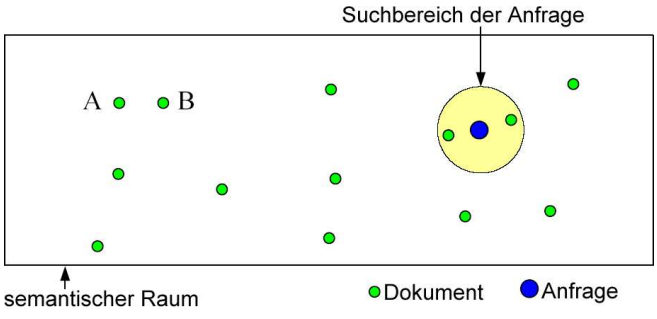

Abbildung 8.3: **Suche in einem semantischem Raum**

Vector Space Model

Im Folgenden wird die erste Umsetzung von Dokumenten (beziehungsweise Anfragen) in sogenannte *Termvektoren* betrachtet. Dabei steht jedes Element des Termvektors für die Relevanz eines Terms (Begriffes), der aus einem Dokument oder einer Suchanfrage stammt, und wird durch den Quotienten

$$\frac{Termfrequenz}{Dokumentfrequenz}$$

ausgedrückt, der einen wichtigen Zusammenhang einführen soll: Je häufiger ein Term in einem Dokument vorkommt, desto besser unterscheidet er dieses von anderen Dokumenten. Taucht der gleiche Term jedoch ebenfalls häufig in anderen Dokumenten auf, schmälert das seine Wichtigkeit durch die Division mit der Dokumentfrequenz. Jeder Termvektor spiegelt also die Relevanz eines Dokuments wider.

Die Ähnlichkeit zweier Termvektoren X und Y wird durch den Cosinus des Winkels dieser Vektoren bestimmt. X und Y sind dabei normiert, um den Längenunterschied verschiedener Dokumente zu kompensieren. Formal drückt sich der Ähnlichkeitswert von $X = (x_1, x_2, \ldots, x_n)$ und $Y = (y_1, y_2, \ldots, y_n)$ in Gleichung (8.1) aus:

$$\cos(X, Y) = \frac{X \cdot Y}{|X| \cdot |Y|} = \sum_{i=1}^{n} x_i y_i \tag{8.1}$$

Wie man erkennt, entspricht $\cos(X, Y)$ gerade dem Skalarprodukt von X und Y.

Latent Semantic Indexing

Da zumindest Textdokumente im Allgemeinen aus einer großen Anzahl von Wörtern besteht, leidet VSM unter dem Problem der Hochdimensionalität: Ein erzeugter Termvektor enthält sehr viele Elemente, was sowohl die Dokumentensuche wie auch die Transformation erschwert. Zur Lösung wurde LSI entwickelt, das einen hochdimensionalen Termvektor (durch eine Projektion in einen semantischen Untervektorraum) in einen niedrigdimensionalen semantischen Vektor verwandelt. Die Elemente solcher Vektoren stellen jetzt abstrakte Konzepte von Dokumenten anstatt Terme dar. Ermöglicht wird diese Transformation durch das mathematische Verfahren *SVD* (engl. singular value decomposition) [BDJ99], an dessen Ende tatsächlich nur die wichtigsten Begriffe eines Dokuments vektorisiert werden. Dafür wird eine Matrix A mit Rang r, dessen Zeilen (t) die verschiedenen Terme und Spalten (d) die verschiedenen Dokumente repräsentieren, mithilfe von SVD zunächst in ein Produkt von drei Matrizen zerlegt, also $A = U\Sigma V^T$. Dabei sind $U = (u_1, \ldots, u_r)$ eine $t \times r$ Matrix, $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$ eine $r \times r$ Diagonalmatrix und $V = (v_1, \ldots, v_r)$ eine $d \times r$ Matrix. Die σ_i sind die singulären Werte von A mit $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$. LSI approximiert A mit Rang r durch eine Matrix A_l mit Rang $l < r$, indem alle bis auf die l größten singulären Werte vernachlässigt werden. Seien $U_l = (u_1, \ldots, u_l)$, $\Sigma_l = \mathrm{diag}(\sigma_1, \ldots, \sigma_l)$ und $V_l = (v_1, \ldots, v_l)$, dann gilt:

$$A_l = U_l \Sigma_l V_l^T$$

Die Zeilen von $V_l \Sigma_l$ stellen die semantischen Vektoren der Dokumente dar. Mit gegebenen U_l, V_l und Σ_l ist es möglich, die semantischen Vektoren von Suchanfragen, Termen oder Dokumenten, die sich ursprünglich nicht in A befanden, durch Faltung in den semantischen Untervektorraum [BDJ99] zu generieren.

Wird ein verhältnismäßiges l für A_l verwendet (üblicherweise zwischen 50 und 350), bleibt durch SVD die relevante Struktur der Daten erhalten, während nichtrelevante Informationen nichtig werden. Darüber hinaus entsteht durch die Beobachtung von auftretenden Wortverwendungen der wünschenswerte Nebeneffekt der Verallgemeinerung: Eine Suche beispielsweise nach `PKW` kann ebenso Dokument mit dem nicht termisch, aber semantisch ähnlichen Begriff `Automobil` zurückliefern.

Wie VSM benutzt LSI den Cosinus des Winkels zweier Vektoren (Gleichung (8.1)) als Ähnlichkeitsmaß.

8.2.4 pSearch: Abbildung des semantischen Raumes auf ein IBN und Suche

Bisher gestatten es die vorgestellten Konzepte, Dokumente in eine Struktur zu überführen, die ein effizientes Wiederauffinden ermöglicht. Für ein P2P-System muss außerdem eine gleichmäßige Verteilung der Daten auf verschiedene Knoten des Netzwerkes gelten. Diese beiden Aufgaben, Abbildung des semantischen Raumes auf eine physische Topologie und die Dokumentensuche, wird vom *pSearch*-System übernommen.

Der semantische, kartesische Raum sei d-dimensional. Dieser wird in sogenannte *Zonen* aufgeteilt und für jede ein Knoten ausgewählt, der diese verwaltet. Wie schon erwähnt, besteht der semantische Raum aus Punkten (Schlüsseln), die auf Objekte (Indizes) verweisen, welche wiederum von dem Knoten gespeichert werden, der Zonenhalter des zugehörigen Schlüssels ist. Da jeder Knoten Indizes eines bestimmten Bereiches speichert und die Wegewahl bei Schlüsseln semantisch ähnlicher Dokumente stets die gleiche Zone ermittelt, wird automatisch garantiert, dass Indizes semantisch ähnlicher Dokumente auch tatsächlich an der gleichen Position im Netzwerk hinterlegt werden. Abbildung 8.4 beschreibt die Zonenaufteilung und insbesondere den Fall, dass ein weiterer Knoten dem IBN hinzugefügt und eine entsprechende Zone aufgeteilt wird. Im grafischen Beispiel tritt D dem Netzwerk bei und platziert seine Zone willkürlich im Bereich von C, woraufhin die Zone halbiert wird.

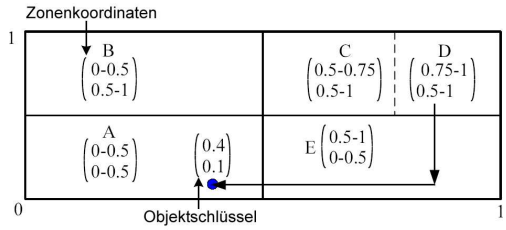

Abbildung 8.4: **Zonenaufteilung in einem 2D-IBN**

Jedoch müssen sich nicht notwendigerweise alle Knoten an der Suchverarbeitung beteiligen, sondern nur eine Teilmenge des Netzwerkes, die als *pSearch Engine* bezeichnet wird. In Abbildung 8.5 wird das Zusammenspiel der Knoten für die Veröffentlichung und Suche nach Dokumenten grafisch präsentiert: Knoten A publiziert ein Dokument bei Engine-Knoten B, der daraus einen Index erzeugt und diesen zum korrekten Bereich im Overlay anhand der Semantik weiterleitet, im Beispiel Knoten F. Im späteren Verlauf könnte E eine korrespondierende Suchanfrage absetzen, die — wiederum auf Grund der semantischen Eigenschaften — zu C weitergeleitet wird. Dieser Knoten übernimmt schließlich die Aufgabe, relevante Indizes in seiner lokalen Umgebung zu sammeln, wobei ein Ergebnis das von A publizierte und in F gespeicherte Objekt sein könnte.

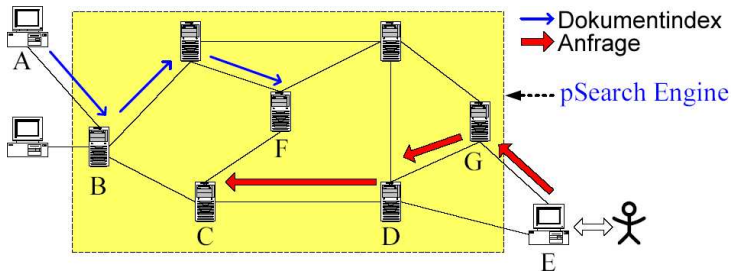

Abbildung 8.5: **Funktionsweise des pSearch-Systems**

Eine formale Definition der Eingliederung von Dokumenten in das semantische Overlay sowie der Suche kann wiefolgt beschrieben werden:

1. Für ein neues Dokument A wird von einem Engine-Knoten der zugehörige semantische Vektor V_a mittels LSI gebildet und als Schlüssel zur Speicherung des Index im IBN verwendet.

2. Auf eine Suchanfrage q hin wird von einem Engine-Knoten der zugehörige semantische Vektor V_q erzeugt und als Schlüssel durch das Overlay zum zuständigen Knoten weitergeleitet.

3. Sobald die Anfrage den zuständigen Knoten erreicht, wird diese innerhalb eines Radius r an alle Knoten geflutet. r bestimmt sich aus einem erwünschten Ähnlichkeitsschwellwert oder der Anzahl der verlangten Rückgabedokumente.

4. Alle empfangenen Knoten innerhalb des Radius r starten eine lokale Suche mittels LSI und geben Referenzen auf alle Dokumente, die am besten mit der Anfrage übereinstimmen, zurück.

Diese Verfahren, das LSI um die Funktionalität zur Beantwortung von Suchanfragen erweitert, lautet *pLSI* und kann anhand von Abbildung 8.6 nachvollzogen werden.

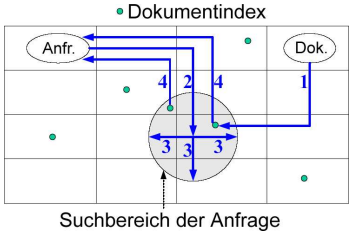

Abbildung 8.6: **pLSI für ein 2-dimensionales IBN**

8.2.5 Bekannte Probleme

Obwohl die oben erläuterten Konzepte, Algorithmen und Verfahren schon jetzt nutzen werden könnten, um ein inhaltsbasiertes P2P-System zu bauen, verbleiben einige Hindernisse, die die effiziente und skalierbare Nutzung des Netzwerkes einschränken. Dass diese Schwierigkeiten jedoch nicht unumgänglich sind, soll anhand der beiden folgenden Probleme — das der *Dimensionalitätsdifferenz* und jenes der *Suchraumgröße* — und deren Lösungen beispielhaft demonstriert werden.

Problem der Dimensionalitätsdifferenz

Im Abschnitt 8.2.4 über die Abbildung von Dokumentinformationen wurde die Annahme getroffen, dass die Dimensionalität des IBN gleich der des von LSI erzeugten semantischen Raumes ist, die üblicherweise zwischen 50 und 350 rangiert. Bezeichnet man letztere mit x, lautet die Mindestanzahl der produzierten Zonen im Rahmen der (gleichmäßigen) Partitionierung des semantischen Raumes 2^x. Weiterhin gilt, dass für ein Netzwerk mit n Knoten und einer Dimensionsgröße von $x > \log_2(n)$ mehr Zonen als Knoten existieren. Als Konsequenz daraus würden die Zonen nicht länger „gleichmäßig" verteilt und nur $\log_2(n)$ Dimensionen partitioniert (siehe Beispiele in Abbildung 8.7). Die Anzahl der tatsächlich partitionierten Dimensionen innerhalb eines IBN wird als *effektive Dimensionalität* bezeichnet.

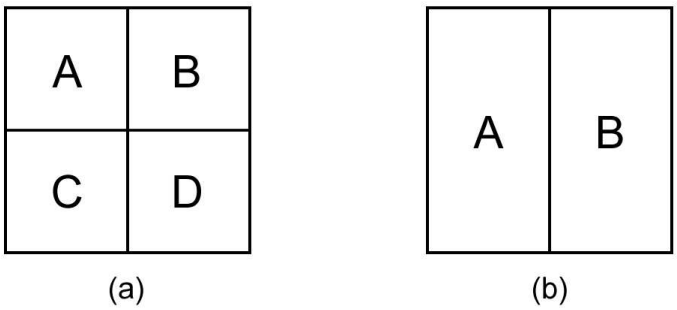

Abbildung 8.7: **Beispiel der Zonenbildung im zweidimensionalen Raum. (a) Mit ausreichend Knoten wird der Raum gleichmäßig in 2^x Zonen partitioniert. (b) Bei einer zu kleinen Knotenmenge wird die effektive Dimensionalität logarithmisch und zu klein für eine Gleichverteilung der Zonen**

Mit einer zu geringen Anzahl an Knoten entsteht ein Effizienzverlust für die Suche: Es wird nur der niedrigdimensionale semantische Raum korrekt partitioniert, der Suchraum für den unpartitionierten Bereich dagegen wird nicht reduziert, sondern Dokumente mit ähnlichem semantischen Inhalt über diesen Dimensionen über das gesamte Netzwerk verteilt.

Abbildung 8.8(a) verdeutlicht diese Situation. Hier besteht der semantische Raum aus den vier Dimensionen $v_0 - v_3$, die Suchanfrage Q und ein Dokument A besitzen die semantischen Vektoren $V_q = (0.55, -0.1, 0.6, -0.57)$ beziehungsweise $V_a = (-0.1, 0.55, 0.57, -0.6)$. Der Ähnlichkeitswert zwischen den beiden Vektoren beträgt nach Gleichung (8.1) 0.574, wobei auffällt, dass der Großteil dieses Wertes von den Vektorkomponenten v_2 und v_3 beigetragen wird.

Für die Speicherung der Vektoren V_a und V_q stehen uns vier Knoten $w - z$ zur Verfügung. Nach bisherigem Kenntnisstand dieses Abschnitts sollte klar werden, dass die Knotenanzahl für einen vierdimensionalen Raum zu gering ist: Für eine optimale Zonenzuteilung werden $2^4 = 16$ Knoten erwartet, ergo muss mit einer Partitionierung nur entlang der ersten $\log_2(4) = 2$ Dimensionen v_0 und v_1 gerechnet werden. Da sich V_a und V_q nicht in diesen Dimensionen ähneln und v_2 sowie v_3 nicht beachtet werden können, würde A mit einer deutlich geringeren Wahrscheinlichkeit auf Q hin gefunden werden.

Bevor eine Lösung des Problems vorgestellt wird, sollten zunächst einige allgemeine Beobachtungen gemacht werden:

- Obwohl die Anzahl der Dimensionen im semantischen Raum sehr groß ist, werden in der Praxis nur relativ wenige Dimensionen tatsächlich genutzt. Bei-

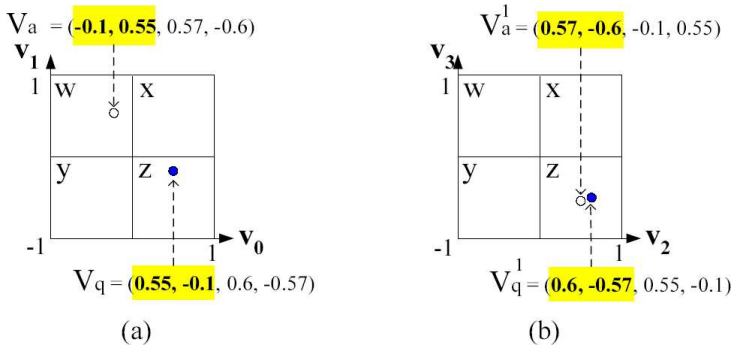

Abbildung 8.8: **Beispiel für rotierenden Index. Die Position des Vektors wird durch die ersten beiden Komponenten bestimmt. (a) Entlang der Dimensionen v_0 und v_1 partitioniertes IBN im ursprünglichen semantischen Raum. (b) Das selbe IBN partitioniert entlang den Dimensionen v_2 und v_3 nach der Rotation der semantischen Vektoren um zwei Dimensionen. Das relevante Dokument A wird auf Anfrage Q hin im Knoten z des rotierten Raumes gefunden.**

spielsweise finden sich beschriebene Konzepte in einer Dokument über Chemie nur mit geringer Wahrscheinlichkeit auch in einem Dokument über Informatik wieder.

- Üblicherweise beinhalten Suchanfrage nur kurze Stichwörter und werden im Allgemeinen von wenigen Konzepten korrekt erfasst, was bedeutet, dass schon eine geringe Anzahl an Elementen in den semantischen Vektoren einen ausreichenden Beitrag zum Ähnlichkeitswert liefern können.

- LSI (8.2.3) sortiert die Elemente von semantischen Vektoren absteigend gemäß dem Grad ihrer Wichtigkeit. Schon diese niedrigdimensionalen Elemente tragen daher mit ausreichendem Anteil zum Ähnlichkeitswert bei.

Diese Eigenschaften macht sich das sogenannte *Rotierender Index*-Verfahren zu Nutze, das zum einen die Dimensionalitätslücke schließt und zusätzlich den Suchraum reduziert. Die grundlegende Idee, die sich hinter diesem Verfahren verbirgt, ist die weitere Partitionierung von Dimensionen des semantischen Raumes durch Rotation (Verschiebung) der semantischen Vektoren. Was durch diesen Ansatz an Geschwindigkeit und Genauigkeit gewonnen wird, geht zu Kosten von Speicherplatz.

Um selbst in einem bezüglich der Anzahl der vorhandenen Knoten begrenzten IBN annähernd alle relevanten Dokumente finden zu können, wurde die Rotation von semantischen Vektoren eingeführt. Dabei wird aus einem Vektor $V = (v_0, v_1, \ldots, v_l)$ durch mehrmalige Verschiebung der Komponenten in die gleiche Richtung um m Dimensionen eine neue Menge von Vektoren gebildet (siehe Gleichung (8.2)), die als *rotierte semantische Vektoren* bezeichnet werden. m wird durch Gleichung (8.3) bestimmt und schätzt die effektive Dimensionalität des IBN ab. Der *rotierte Raum* i schließlich wird von Rotationsvektoren verschiedener Dokumente und Suchanfragen mit identischer Anzahl von i Rotationen aufgespannt.

$$V^i = (v_{i \cdot m}, \ldots, v_0, v_1, \cdots, v_{i \cdot m - 1}), \quad i = 0, \ldots, p-1 \tag{8.2}$$

$$m = 2.3 \cdot \ln(n) \tag{8.3}$$

Wie eingangs erwähnt, erkauft man sich einen Vorteil nur durch zusätzlichen Verbrauch von Speicherplatz: Der Index eines gegebenen semantischen Vektors V_a

wird ab sofort auf p Stellen im IBN gespeichert, wobei $V_a^i, i = 0, \ldots, p-1$ als jeweilige Schlüssel dienen. Erfolgt eine Suchanfrage V_q in dem manipulierten Netzwerk, muss der pLSI-Algorithmus nun für jeden Vektor V_q^i ausgeführt werden, insgesamt also p-mal. Für jede Anwendung wird eine separate Suchanfrage in den rotierten Raum i verschickt und individuell bearbeitet, d.h. jeder Suchraum liefert seine lokal optimalen Ergebnisse anhand V_q^i zurück. Dies entspricht letztendlich einem Zusammenschluss aller Rotationsräume, und es ist somit gelungen, trotz einer unzureichenden Anzahl an Knoten den vollständigen Suchraum ausreichend zu approximieren. Abbildung 8.8(b) benutzt das letzte Beispiel des zweidimensionalen IBN und zeigt, wie eine Rotation deutlich bessere Ergebnisse liefern kann.

Es ist noch zu erwähnen, dass die Rotation das Ähnlichkeitsmaß unbeeinflusst lässt. Der Ähnlichkeitswert zweier Vektoren wird bekanntermaßen durch das Skalarprodukt gemessen (Gleichung 8.1), womit auch rotierte Vektoren, die im Overlay räumlich nah beieinanderstehen, auch semantisch ähnlich bleiben. Nichtsdestotrotz werden trotz des Rotationsprinzips weiterhin komplette semantische Vektoren zur initialen Bestimmung des Ähnlichkeitswerts (mittels VSM und LSI) benutzt.

Große Suchräume

Trotz der Tatsache, dass das erläuterte IBN so konstruiert ist, dass semantisch zusammengehörige Dokumente auch räumlich nah platziert werden und daher bezüglich Suchanfragen lediglich innerhalb eines beschränkten Radius Knoten durchsucht werden müssen, verbleibt folgender effizienzhemmender Sachverhalt: Der Suchraum steigt überproportional mit der Dimensionalität der gegebenen Daten. Dieser Zusammenhang wird auch als *Fluch der Dimensionalität* bezeichnet.

Darüber hinaus wurden aus Untersuchungen zwei weitere wichtige Erkenntnisse gewonnen: (1) Hochdimensionale Datenräume sind spärlich popularisiert. (2) Die Distanz zwischen einer Suchanfrage und ihrem nächsten Nachbarn steigt mit der Dimensionalität des Raumes stetig an.

Um diesen sowohl unerwünschten wie auch unumgänglichen Systemeigenschaften entgegenzuwirken, ohne die Anzahl der anzufragenden Knoten zu erhöhen, muss der Suchalgorithmus pLSI in seiner letzten Stufe modifiziert werden. Zum besseren Verständnis sei erwähnt, dass selbst in einem weit verteiltem Raum semantisch ähnliche Dokumente dichte Anhäufungen bilden. Ein Mittel gegen den beschriebenen „Fluch“ müsste daher die Suche in Richtung der korrekten Anhäufung relevanter Dokumente dirigieren. Genau dies wird mit der *inhaltsorientierten Suche* angestrebt, die anhand der gespeicherten Inhalte der Knoten (den Indizes) und den Ergebnissen kürzlich angestoßener, lokaler Suchanfragen zur „richtigen“ Menge von Knoten, die die verlangten Dokumente am wahrscheinlichsten enthalten, führt.

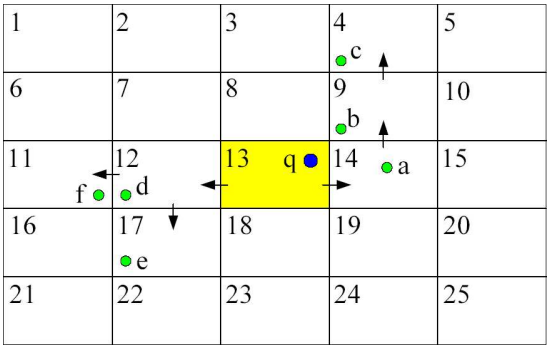

Abbildung 8.9: **Beispiel einer inhaltsorientierten Suche**

Abbildung 8.9 stellt ein Netzwerk aus 25 Knoten in einem zweidimensionalen IBN dar, das als Beispiel für die Funktionsweise des Algorithmus dienen soll. Als $a - f$ werden semantische Dokumentvektoren und mit q der Suchanfragevektor bezeichnet, und der Anfragesteller habe angegeben, dass er die drei am relevantesten Dokumente zurückgeliefert haben möchte. Dies seien im Beispiel a, b und c.

Zunächst wird die initiale Suchanfrage q durch die pSearch-Engine zu Knoten 13 weitergeleitet, der jedoch keine relevanten Dokumente besitzt. Daraufhin wird eine Liste $\mathcal{N}$ konstruiert, die alle zukünftig zu durchsuchenden Knoten enthält. Da zu Beginn keinerlei Vermutung darüber herrscht, in welchem Bereich sich relevante Dokumente am wahrscheinlichsten befinden, wird die Liste mit den unmittelbaren Nachbarn von Knoten 13 gefüllt, also $\mathcal{N} = \{8, 12, 14, 18\}$. Daraufhin prüft jeder dieser Nachbarknoten wiederum seine Nachbarn auf relevante Dokumente (Proben). Dieses Beispiel benutze eine sequentielle Suche, für die als nächstes Knoten 14 positive Ergebnisse von seinen Nachbarn erhält. In der Folge wird Knoten 14 durchsucht, Dokument a entdeckt und die Liste zu $\mathcal{N} = \{8, 12, 18, \mathbf{9}, \mathbf{15}, \mathbf{19}\}$ angepasst, wobei die fettgedruckten Knoten die weitere Tendenz der Suchrichtung angeben: Da in Knoten 14 ein erfolgreicher Suchtreffer festgestellt wurde, gilt mit der Annahme der Dokumentanhäufungen, dass in unmittelbarer Umgebung von Knoten 14 weitere relevante Ergebnisse zu finden sind. In analoger Weise sammelt Knoten 9 Proben seiner Nachbarn und wird in 4 fündig. Also durchsucht der Algorithmus als nächstes Knoten 9, wodurch ein weiteres Ergebnis, Dokument b, verzeichnet werden kann. Die Liste besitzt danach die Reihenfolge $\mathcal{N} = \{8, 12, 18, 15, 19, \mathbf{4}, \mathbf{10}\}$. Die Umgebung von Knoten 9 führt schließlich dazu, dass auch das letzte Dokument c auf Knoten 4 gefunden wird. Über weitere Probenermittlungen können wir im letzten Schritt noch ausschließen, dass auch die auf den Knoten 12, 11 und 17 gelagerten Dokumente nicht relevanter sind als $a - c$. Da die Trefferwahrscheinlichkeit für eine Ausweitung der Suche ab nun äußerst gering wird, terminiert die inhaltsorientierte Suche an dieser Stelle.

Natürlich ist es ebenfalls realisierbar, die Überprüfung der Nachbarknoten parallel auf mehreren Knoten durchzuführen. Der damit gewonnene Geschwindigkeitszuwachs erfolgt jedoch unter Umständen auf Kosten unnötiger Durchsuchungen von Knoten, die keine relevanten Ergebnisse zurückliefern können.

8.2.6 Zusammenfassung: P2P-System mithilfe eines semantischen Overlays

Mit den in diesem letzten Ausarbeitungsabschnitt präsentierten Erkenntnissen wurde gezeigt, dass das Modell des semantischen Overlay in Bezug auf P2P-Systeme Vorteile gegenüber konventionellen Netzwerkstrukturen aufweist. Durch die Konstruktion eines angepassten Wegewahlverfahrens ist es möglich, semantische Inhalte mittels VSM und LSI in einer Art und Weise verteilt abzuspeichern, dass ein Wiederauffinden durch den pLSI-Algorithmus entkoppelt von der Anzahl der Dokumente skalierbar erscheint. Der daraus resultierenden, gesteigerten Effizienz stehen zwar neuartige Probleme (wie das der Dimensionalitätsdifferenz und große Suchräume) gegenüber, die aus der Nutzung der verwendeten Konzepte gewachsen sind; durch entsprechende Modifikationen in Form von rotierenden semantischen Vektoren und der Entwicklung eines inhaltsorientierten Suchalgorithmus ist es jedoch weiterin möglich, sich die prinzipiellen Vorteile dieses Modells Nutze zu machen.

Literaturverzeichnis

[BDJ99] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[CW03] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '03)*, pages 163–174, Karlsruhe, Germany, August 2003.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[TXD03] Chunqiang Tang, Zhichen Xu, and Sandya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '03)*, Karlsruhe, Germany, August 2003.

Kapitel 9

Grid Middleware: GLOBUS

Seminarbeitrag von **Christian Würdig**

Dieses Kapitel beginnt mit der Erläuterung der Motivation, die hinter Grid-Computing[1] steckt und aus der heraus Globus entstand, sowie der zu lösenden Probleme, die bei der praktischen Realisierung zu beachten sind. Danach folgt ein Überblick über die zugrunde liegende Architektur von Globus, wobei die einzelnen Schichten, ihre Funktionen und Verbindungen dargestellt werden. Abgeschlossen wird das Kapitel mit einem Vergleich von Grid und zwei verbreiteten, ähnlichen Systemen, dem Internet und Peer-2-Peer Systemen, sowie einem Beispiel des praktischen Einsatzes von Grid.

9.1 Einleitung

Die Anforderungen an die Leistung und Verfügbarkeit von Rechnersystemen und den damit verbundenen Ressourcen steigen seit Jahren beständig an. Häufig werden jedoch Spitzenleistungen nur für einen relativ kurzen Zeitraum gefordert, so dass viele Firmen und auch öffentliche Einrichtungen, wie z.B. Universitäten, vor dem Dilemma stehen, entweder genug Kapazitäten für (kurzfristig) hohe Anforderungen bereit zu stellen und dabei in Gefahr zu laufen einen Großteil der Ressourcen die meiste Zeit ungenutzt herumstehen zu haben, oder eine Überlastung ihrer EDV-Infrastruktur zu riskieren.

Mit der zunehmend immer enger und zuverlässiger werdenden Vernetzung der Rechner weltweit scheint sich aber auch gleichzeitig eine Lösung für dieses Problem anzubieten. Die Idee ist es, Ressourcen anderer Rechnersysteme im Bedarfsfall zu nutzen bzw. im Gegenzug eigene Ressourcen (evtl. gegen Bezahlung) bereit zu stellen. Das ist der Grundgedanke des Grid-Computing.

Allerdings ergeben sich dabei auch vielfältige Probleme, wie z.B. das Auffinden und die Verwaltung von Ressourcen, sicherheitskritische Fragen („Wer darf was?“)

[1] siehe auch [Fos02]

und ähnliche. Mit dem Ziel diese Probleme zu lösen und das Grid-Computing voranzutreiben wurde 1995 *Globus Project* gegründet. Im Laufe der Zeit kamen immer mehr Mitglieder hinzu und 2003 wurde das Projekt in *Globus Alliance* umbenannt. Diese Allianz besteht zur Zeit aus mehreren Universitäten und Instituten (u.a. Argonne, ISI, Univ. von Chicago, Univ. Edinburgh, Schwedisches Zentrum für Parallelrechner), die über die ganze Welt verteilt sind. Zudem wird die Allianz von diversen Firmen aus der Industrie wie IBM, Microsoft und Cisco gesponsert.

9.2 Zielsetzungen und Motivation

Das Hauptziel der *Globus Alliance* ist die Unterstützung der einfachen orts- und plattformunabhängigen Nutzung verteilter Ressourcen. Im Zusammenhang damit wurde der Begriff *Virtuelle Organisation* (VO) geprägt. Darunter ist eine Gruppe von Personen, Firmen, Universitäten und/oder Dienstleistern zu verstehen, die gemeinsam an der Lösung eines Problems arbeiten [FKT01]. Die dabei auftretenden Anforderungen an Rechenleistung und Speicherkapazität übersteigen häufig die vorhandenen Kapazitäten einzelner Mitglieder.

Damit die koordinierte gemeinsame Nutzung der vorhandenen Ressourcen ermöglicht werden kann, ist ein Konzept nötig, das von den vielen verschiedenen Ressourcentypen abstrahiert und eine gemeinsame Plattform für die Bereitstellung, Verteilung und Nutzung bietet. Auch die Sicherung bestimmter Qualitätsmerkmale („Quality of Service“, QoS) wie Sicherheit, Geschwindigkeit, Verfügbarkeit, Problembehandlung und Bezahlung muss gewährleistet sein. Ein weiterer Aspekt ist die Integrationsfähigkeit in bzw. von bestehenden Produkten, damit bereits entwickelte und funktionierende Lösungen nicht „weggeworfen“ werden und das Rad neu erfunden werden muss. Das spielt vor allem eine Rolle bei der Nutzung vorhandener Strukturen, wie dem Internet und den zugehörigen Protokollen. Grid soll das Internet nicht ersetzen, sondern es nutzen.

Dabei stellt sich natürlich die Frage: Wenn es schon funktionierende Lösungen gibt, warum dann Globus? Die Antwort darauf ist relativ einfach: Bestehende Lösungen umfassen nur einen Teilaspekt, z.B. Dienstleister, die nur Speicherplatz zur Verfügung stellen oder Frameworks wie CORBA, wo nur die Ressourcen Speicher und Rechenkapazität geteilt werden und der Sicherheitsaspekt fehlt.

Gerade aus dem Blickwinkel der Vertrauenswürdigkeit fehlen vorhandenen Projekten wichtige Eigenschaften: Programme einer fremden Firma laufen in der eigenen Umgebung und die Kontrolle über diese Programme fehlt bzw. ist schwierig zu realisieren. Aber nicht nur aus dem Blickwinkel desjenigen, der die Ressourcen bereit stellt fehlen Kontrollmechanismen, sondern auch aus der Sicht des Nutzenden gibt es Defizite. So ist mit aktuellen Protokollen eine Kontrolle der Ressourcen („Wie viel kann ich nutzen?“, „Wann kann ich die Ressourcen nutzen?“) ebenso unmöglich, wie eine Koordination („Wann wird wem welche Ressource in welchem Umfang zur Verfügung gestellt?“) der selbigen. Es gibt auch Grid-Lösungen, die aber meist nur firmen- bzw. konzernintern realisiert sind und bei denen z.B. der Sicherheitsaspekt fehlen kann.

Auf Basis dieser Überlegungen entwickelte die *Globus Alliance* eine Serie von Protokollen und Diensten, die die Grundlage für das *Globus Toolkit* bilden. Dieses Toolkit mit seinen APIs und SDKs bildet zusammen mit der Grid-Architektur sowie ihren Protokollen und Diensten die *Globus Middleware*.

Dabei sei an dieser Stelle ein weiteres Projekt erwähnt – PlanetLab[2]. PlanetLab verfolgt ähnliche Ziele wie Globus, allerdings mit einem anderen Ansatz. Während Globus versucht, bestehende Protokolle soweit wie möglich zu nutzen und gemeinsame APIs zu definieren, stellt PlanetLab lediglich sogenannte virtuelle Container

[2]siehe Beitrag Alexander Dieterle

bereit, die ähnlich wie die virtuelle Maschine von Java funktionieren. Die Nutzer müssen dann darauf eigene Protokolle und Dienste implementieren. Damit eignet sich dieses Projekt vor allem als Testplattform für neue Protokoll- und Dienstentwicklungen, bei denen man mit möglichst wenig Aufwand feststellen will, wie gut sie sich in der Praxis bewähren.

9.3 Die Grid-Protokoll-Architektur basierend auf Globus

Es ist nicht das Ziel der Architekturspezifikation, alle benötigten Protokolle und APIs genau zu definieren und festzulegen, sondern generelle Funktionsklassen zu bestimmen, so dass die Universalität und Flexibilität der Idee des Grid-Computing erhalten bleibt.

Die von der *Globus Alliance* entworfene Grid-Protokoll-Architektur ist, wie in der Informatik üblich, als Schichtenmodell aufgebaut (siehe Abb. 9.1).

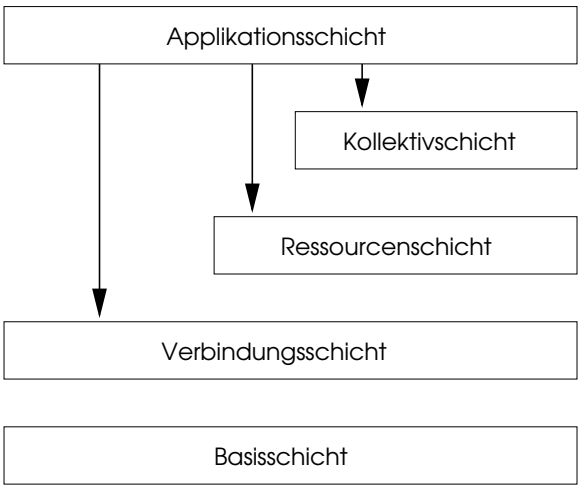

Abbildung 9.1: Modell der Grid-Protokoll-Architektur nach Globus [FKT01]

Bei der Bestimmung der Protokollklassen hat sich das „Sanduhrenmodell“(siehe Abb. 9.2) als sinnvoll erwiesen. Dabei repräsentiert die Form der Sanduhr die Vielfalt der Protokolle in den einzelnen Schichten.

In der Basisschicht existiert eine Menge an Low-Level-Protokollen, die für die vielen unterschiedlichen Technologien zuständig sind (im Vergleich dazu Ethernet, Tokenring, BNC, ATM, ... im Internet). Diese Schicht bildet den Boden der Sanduhr. Die Verjüngung bilden die Verbindungs- und Ressourcenschicht, die nur eine kleine Anzahl von Protokollen enthalten sollten, um die Komplexität im Rahmen zu halten (im Vergleich dazu TCP/IP und UDP im Internet). Diese Protokolle müssen allerdings so gewählt sein, dass es möglich ist, eine Vielzahl von High-Level-Protokollen darauf abzubilden und diese dann wiederum auf eine Vielzahl von Low-Level-Protokollen. Das Oberteil der Sanduhr wird von der Kollektiv- und der Anwendungsschicht gebildet. Diese beiden Schichten enthalten eine Menge an abstrakten und anwendungsspezifischen High-Level-Protokollen (im Vergleich dazu HTTP, FTP, SSH, SMTP, ... im Internet).

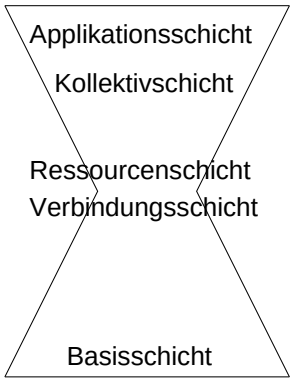

Abbildung 9.2: Sanduhrenmodell der Grid-Protokoll-Architektur

9.3.1 Basisschicht

In der Basisschicht erfolgt die Bereitstellung der einzelnen Ressourcen für den Zugriff über das Grid. Eine Ressource kann dabei z.B. ein Rechner, ein Cluster, ein verteiltes Dateisystem, etc. sein. Die Softwarekomponenten in dieser Schicht stellen demzufolge die lokalen, ressourcenspezifischen Operationen bereit, die durch Anforderungsoperationen aus einer höheren Schicht entstehen. Dabei gibt es einen Zusammenhang zwischen den in der Basisschicht implementierten Funktionen und den Anforderungsoperationen der anderen Schichten. Je größer der Funktionsumfang in dieser Schicht, desto differenzierter können die Operationen in den höheren Schichten ausfallen. Ein größerer Funktionsumfang stellt aber auch höhere Anforderungen an die einzelnen Elemente dieser Schicht und erschwert somit die Entwicklung der Infrastruktur. Des weiteren werden hier die Managementfunktionen zur Sicherung bestimmter Qualitätsmerkmale („Quality of Service“, QoS) bereitgestellt.

Das *Globus Toolkit* versucht soweit wie möglich vorhandene Operationen zu nutzen und implementiert nur die fehlenden. So wird u.a. Software mitgeliefert, die es ermöglicht Informationen[3] über den Zustand und die Struktur der verschiedenen Ressourcen zu erhalten. Außerdem werden diese Informationen zur Weiterverarbeitung für die nächste Schicht aufbereitet.

9.3.2 Verbindungsschicht

Die Verbindungsschicht definiert die Kommunikations- und Authentifizierungsprotokolle, die für die gridspezifischen Transaktionen, wie Datenaustausch zwischen Ressourcen oder Verifizierung der Identität von Benutzern und Ressourcen, benötigt werden. Die Protokolle haben dabei folgende Aufgaben:

- *Kommunikationsprotokolle.* Sie müssen den Austausch von Daten zwischen den Ressourcen der Basisschicht ermöglichen. Dafür werden im *Globus Toolkit* die vorhanden Protokolle für Internet, Transport, Routing und Namensvergabe (TCP/IP, UDP, DNS, BGP ...) genutzt.

- *Authentifizierungsprotokolle.* Diese Protokolle stellen kryptographische Mechanismen zur Authentifizierung von Nutzern und Ressourcen bereit. Das *Globus Toolkit* verwendet die Protokolle der auf public-key basierenden *Grid Security Infrastruktur* (GSI) [FKTT98, Hal04] für Authentifizierung, Kommunikationsverschlüsselung und Autorisierung. GSI erweitert dabei die TLS

[3] z.B. Hardware Konfiguration, CPU- und Netzwerkauslastung, verwendetes Betriebssystem, verfügbarer Speicherplatz

Protokolle, um wichtige Authentifizierungsmechanismen, wie Single Sign-On und Delegation [GM90]. Zudem wird die Kompatibilität zu lokalen Sicherheitslösungen wie Kerberos sichergestellt. Zur Identifikation werden X.509-Zertifikate benutzt.

9.3.3 Ressourcenschicht

Die Ressourcenschicht benutzt die Kommunikations- und Authentifizierungsprotokolle der Verbindungsschicht, um auf deren Basis effiziente und sichere Möglichkeiten zur Ausführung und Kontrolle verteilter Operationen auf den einzelnen Ressourcen zu schaffen. Dazu gehören u.a. Initialisierungs- und Beobachtungsmöglichkeiten aber auch die korrekte Abrechnung und Bezahlung der in Anspruch genommenen Leistungen.

Die Implementierungen der dazu nötigen Protokolle benutzen auch direkt Funktionen der Basisschicht, um möglichst effizient auf die Ressourcen zugreifen zu können. Dabei liegt der Fokus auf der Kontrolle der einzelnen Betriebsmittel, so wie sie in der Basisschicht vorhanden sind und nicht auf den komplexen, zusammengesetzten Abstraktionen. Diese werden in der Kollektivschicht behandelt.

Es sind an dieser Stelle zwei Hauptklassen von Protokollen zu unterscheiden, nämlich Informations- und Managementprotokolle, die folgende Aufgaben besitzen:

- *Informationsprotokolle.* Die Protokolle in dieser Klasse sind für die Ermittlung von Informationen über Struktur und Zustand der Betriebsmittel zuständig. Mit ihrer Hilfe lassen sich z.B. die Rechnerkonfiguration[4], CPU- und Netzwerk-Auslastung, Nutzungsbedingungen[5], etc. herausfinden.

- *Managementprotokolle.* Diese Protokolle dienen dem Aufbau von Beziehungen zwischen einzelnen Knoten innerhalb des Grid. Dazu gehören zum einen die Aushandlung der Zugriffsbedingungen auf die einzelnen Ressourcen und zum anderen die Kontrolle, dass die Zugriffsoperationen diese Bedingungen auch nicht verletzen. Zu den Zugriffsbedingungen gehören u.a. die Spezifizierung der Mindestansprüche in Bezug auf Qualität und Quantität aber auch Reservierungsanfragen, wie z.B. „Ich hätte gerne am xx.xx.xxxx um xx:xx Uhr folgende Ressourcen ...“ und die Abrechnungskontrolle der bereitgestellten Dienste.

In dieser Schicht sind, wie weiter oben schon erwähnt, möglichst wenig Protokolle erwünscht (siehe Abb. 9.2), da sie zusammen mit der Verbindungsschicht die Verjüngung der Sanduhr bildet. Dazu müssen sie elementare Mechanismen bereitstellen, die es erlauben eine große Palette verschiedener Ressource zu teilen und zu nutzen, ohne dabei jedoch die Art der Protokolle in den höheren und tieferen Schichten zu sehr einzuschränken.

Das *Globus Toolkit* nutzt ein auf LDAP basierendes Informationsprotokoll[6] zur Bereitstellung der Ressourceninformationen. Desweiteren gibt es zwei Managmentprotokolle. Eines, das HTTP als Basis hat[7] und zur Ressourcenbeschaffung sowie Beobachtung und Kontrolle der Berechnungen auf diesen Ressourcen dient, und ein anderes, das auf FTP aufbaut[8] und für für den Datenzugriff und -transfer zuständig ist. Dazu werden vom *Globus Toolkit* Client-APIs und -SDKs mitgeliefert um die Nutzung und Programmierung auf Basis dieser Protokolle zu erleichtern. Serverseitige SDKs vereinfachen dabei die Ressourcenintegration in das Grid.

[4] Betriebssystem, verwendete Hardware, ...
[5] Nutzungsdauer, Kosten, ...
[6] GRIP - Grid Ressource Information Protocol
[7] GRAM - Grid Resource Access and Management
[8] GridFTP

9.3.4 Kollektivschicht

Die Kollektivschicht beherbergt Protokolle und Dienste, die sich weniger mit einer einzelnen Ressource befassen als vielmehr mit Ressourcenansammlungen und den Interaktionen zwischen ihnen. Das Spektrum reicht dabei von „general purpose“ bis hin zu anwendungsspezifischen Protokollen. Beispiele dafür wären Verzeichnisdienste, die es erlauben Ressourcen anhand bestimmter Merkmale wie Ressourcentyp, Verfügbarkeit, Auslastung, etc. anzufordern, Diagnosedienste zur Erkennung von Überlastungen, Fehlern und Angriffen, oder das Bereitstellen einer ganzen Gruppe von (verschiedenen) Ressourcen[9] zur Lösung eines Problems.

Da diese Schicht auf der Kommunikations- und Ressourcenschicht aufbaut, die sich explizit um die Kommunikation und Kontrolle der einzelnen Ressourcen kümmern, können die Protokolle hier eine große Bandbreite verschiedener Operationen zur Abstraktion und Koordinierung ganzer Ressourcengruppen implementieren und bereitstellen, ohne neue Anforderungen an die individuellen Betriebsmittel stellen zu müssen.

Das *Globus Toolkit* stellt momentan folgende Dienste zur Verfügung:

- *Meta Directory Service.* Dienst zur Unterstützung verschiedener Sichten auf Ressourcengruppen mit Hilfe von Indexservern[10], die sich per LDAP Informationen von speziellen Diensten[11] über den Zustand der Ressourcen besorgen.
- *Registrierungsprotokoll.* Protokoll[12], um Ressourcen im Grid bekannt zu machen und zur Nutzung bereit zu stellen.
- *Replizierungsservice.* Zur Verwaltung von Datenspiegelungen im Grid gibt es spezielle Management- und Katalogisierungsdienste.
- *Co-Allocation.* Es werden SDKs und APIs[13] mitgeliefert, um ganze Gruppen, zusammengesetzt aus verschiedenen Ressourcen, zu reservieren.

9.3.5 Anwendungsschicht

Die Anwendungsschicht bildet den oberen Abschluß der Architektur. Hier sind die Anwendungen, die in einer Virtuellen Organisation laufen angesiedelt. Sie haben dabei Zugriff auf alle anderen Schichten der Grid-Architektur, wobei in jeder Schicht wohldefinierte Protokolle und APIs existieren, die genutzt werden können. Mit Anwendungen sind an dieser Stelle auch Frameworks wie CORBA gemeint, die man an die hier vorgestellte Architektur anpassen könnte, so dass vorhandene Programme die darauf basieren nicht umgeschrieben werden müssen.

9.4 Globus in der Praxis

9.4.1 Grid im Vergleich mit dem Internet

Wenn man Grid und Internet vergleicht, dann fällt auf, dass das Internet hauptsächlich dem reinen Austausch von Informationen dient und damit eine spezielle Ausprägung des Grid darstellt – ein Informationsgrid sozusagen. Darüber hinaus tendiert die Entwicklung in neuerer Zeit stark in Richtung des Teilens von Daten und Ausführen entfernter Anwendungen (Stichwort Java). Die beiden Technologien weißen demzufolge einige Gemeinsamkeiten auf:

[9] co-allocation
[10] GIIS - Grid Information Index Server
[11] GRIS - Grid Resource Information Service
[12] GRRP - Grid Resource Registration Protocol
[13] DUROC co-allocation Bibliothek

- großflächige Vernetzung vieler Teilnehmer
- Bereitstellung und Nutzung von verteilten Ressourcen

Allerdings gibt es auch Differenzen. So fehlen im Internet wichtige Features, wie Single Sign-On und Delegation und außerdem steht die Kommunikation mit einzelnen Ressourcen und nicht mit ganzen Ressourcengruppen im Vordergrund. Wobei Techniken wie Load-Balancing auch hier die Grenzen langsam verschwimmen lassen. Doch auch Grid hat im Vergleich mit dem Internet ein paar Defizite. Das Internet verfügt nämlich bereits über eine Infrastruktur und ausgereiftere und funktionierende Protokolle, während die Entwicklung bei Grid quasi noch in den Kinderschuhen steckt.

Mit dem Ziel, die beiden Technologien zu vereinen wurden aber auch einige gute Ansätze entwickelt. So könnte man Browsern mit Hilfe der GSI-Erweiterungen zu TLS Single Sign-On beibringen oder per GSI Delegation Befähigungen (capabilities) an Webserver weiterreichen, die dann im Auftrag des Clients agieren. Es existieren auch bereits Projekte, wie WebOS[14], die sich mit dieser Thematik beschäftigen.

Abschließend sei noch bemerkt, dass es nicht das Ziel von Grid ist, das Internet zu ersetzen, sondern dessen bereits existierende und funktionierende Infrastruktur zu nutzen.

9.4.2 Grid und Peer-to-Peer

Beim Vergleich von Grid mit Peer-to-Peer (P2P) fällt auf den ersten Blick sofort eine Gemeinsamkeit auf: Beide Strategien verfolgen die koordinierte Nutzung weltweit verteilter Ressourcen unabhängig von den Beziehungen der beteiligten Mitglieder. Es sind allerdings einige Unterschiede vorhanden, die sich durch die Entstehungsgeschichte erklären [FI03].

Der Hauptunterschied ist sicher durch das Umfeld begründet, aus dem heraus beide Technologien entstanden. Während Grid aus dem wissenschaftlichen Bereich (Universitäten, Forschungszentren) mit meist lediglich ein paar Hundert Teilnehmern kommt, wo ein gewisser Vertrauenslevel untereinander besteht und die Teilnehmer nicht völlig anonym sind, so entstand P2P „auf der Straße“ durch File-Sharing-Programme wie Napster oder Gnutella bzw. massiv parallele Anwendungen wie z.B. Seti@Home mit einigen Hunderttausend bis Millionen anonymen Teilnehmern, wo keinerlei Annahmen darüber getroffen werden, in wie weit man anderen vertrauen kann.

Auch in der Qualität und Quantität der gebotenen Dienste differieren beide. Die Ausfallquote einzelner Teilnehmer ist bei P2P Netzwerken in der Regel wesentlich höher als bei Grid Netzwerken. Zudem werden in P2P Netzwerken häufig nur einige wenige Dienste angeboten (Dateiaustausch, Prozessorzeit) während bei Grid durchaus komplexere Anforderungen entstehen (Zwischenspeichern größerer Datenmengen, Operationen auf verteilten Datenbanken).

Aus der unterschiedlichen Qualität und Teilnehmerzahl ergibt sich auch eine andere Struktur des Netzaufbaus. Die Netzwerke im P2P Bereich sind auf gute Skalierbarkeit (mehrere Millionen Teilnehmer), Fehlertoleranz und Selbstorganisation ausgelegt. Bei Grid Netzwerken kommen hingegen vermehrt zentrale Komponenten (zentrale Datenverwaltung, zentrales Management) zum Einsatz.

Anhand der Beispiele Seti@Home und Napster ist allerdings auch zu sehen, dass die Unterschiede mit der Zeit immer mehr verwischen. So kann man Seti@Home durchaus dem Grid Bereich zuordnen, da eine wesentliche P2P Eigenschaft fehlt: Der eigentlich Datenaustausch findet nicht zwischen den einzelnen Teilnehmern statt,

[14] http://www.cs.duke.edu/ari/issg/webos/

sondern mit einem zentralen Server. Auch bei Napster findet sich ein zentraler Verzeichnisdienst – ebenfalls ein gridspezifisches Merkmal.

Es ist zu beobachten, dass die Entwicklung von Grid Software immer mehr auf den Massenmarkt abzielt und versucht wesentliche Eigenschaften wie Fehlertoleranz und die gute Skalierbarkeit mit der Teilnehmerzahl aus dem P2P Bereich zu übernehmen. Die Entwickler von P2P Software hingegen zielen immer mehr auf mächtigere Anwendungen und versuchen sich an der Standardisierung verschiedener Dienste. So entwickeln sich Grid und P2P immer mehr aufeinander zu.

9.4.3 Beispiel GridFTP

Zunächst sollte man klären, was unter dem Begriff GridFTP [ABF+02] eigentlich zu verstehen ist. Da wäre zum einen das Protokoll an sich, welches aus dem bekannten FTP durch mehrere Erweiterungen entstanden ist. Zur Zeit gibt es noch keinen RFC[15], aber einen Draft [All03], der der IETF[16] zur Begutachtung vorgelegt werden soll. Dann gibt es noch den eigentlichen GridFTP Server, der auf einem, um die GridFTP-Protokollerweiterungen ergänzten, wu-ftpd[17] basiert. Schlussendlich liefert das Toolkit auch noch Clients, Bibliotheken und SDKs mit, die ebenfalls zu GridFTP gehören. In aller Regel wird der Begriff GridFTP für die komplette Softwaresammlung (Server, Clients, Bibliotheken, ...) benutzt.

Technisch gesehen ist das GridFTP-Protokoll in der Ressourcenschicht (siehe Abb. 9.1) angesiedelt. Das liegt daran, dass die eigentlichen Daten immer nur zwischen einzelnen Ressourcen ausgetauscht werden und nicht zwischen heterogenen Ressourcengruppen. Nichts desto trotz stellt GridFTP eine gutes Beispiel für Anwendungen im Grid dar, denn es existiert eine funktionierende Implementierung und es lassen sich relativ gut die einzelnen Zusammenhänge der Architektur darstellen.

GridFTP bietet folgende Features:

- *Authentifizierung.* Benutzer können sich per GSI und Kerberos authentifizieren. Mit Hilfe der GSI Erweiterungen ist somit auch Single Sign-On und Delegation möglich, so dass es reicht, wenn man sich bei einem GridFTP-Server authentifiziert um mit allen anderen FTP-Server in diesem Verbund kommunizieren zu können.

- *Third-party Transfer.* Dieses Feature wurde von FTP übernommen. Damit ist es möglich als Benutzer von außen den Transfer von Daten zwischen zwei Servern anzustoßen und zu kontrollieren. Das ist vor allem für den Transfer großer Datenmengen zwischen zwei Knoten gedacht.

- *Paralleler Datentransfer.* Um eine möglichst hohe Ausnutzung der Bandbreite zu erreichen, kann man die Daten mit Hilfe mehrerer paralleler Datenkanäle zwischen zwei Servern transferieren.

- *Verteilter Datentransfer.* Zur Bündelung von Bandbreiten ist es möglich, die Daten über mehrere Server, ähnlich dem RAID-0[18] Prinzip, zu verteilen. GridFTP erweitert FTP um neue Komandos zur Unterstützung des Transfer von, zu und zwischen auf diese Art organisierten Servern, sogenannten *striped Servern*. Diese Feature ist im *Globus Toolkit* allerding noch nicht enthalten, sondern erst für die Version 4.0 geplant.

- *Transfer von Dateifragmenten.* Diese Erweiterung von FTP ermöglicht es, nur einen Teil einer Datei zu übertragen. FTP unterstützt ursprünglich lediglich

[15] Request For Comment, siehe auch http://www.dodabo.de/netz/rfcs.html
[16] http://www.ietf.org/
[17] http://www.wuftpd.org/
[18] http://de.wikipedia.org/wiki/RAID

den kompletten Dateitransfer oder das Fortsetzen des Transfers ab einer bestimmten Stelle.

- *Zuverlässiger Datentransfer.* FTP ist von Haus aus robust, was Netzwerkfehler und Transferabbrüche betrifft. Bei einem Abbruch wird der Transfer an der entsprechenden Stelle einfach wieder aufgenommen. Fehler in der Übertragung werden dabei von darunter liegenden Transportprotokollen (TCP/IP) abgefangen. Diese Eigenschaft gilt natürlich auch für GridFTP.

- *Automatische Anpassung der TCP Puffergröße.* Dieses Feature ist im Protokoll vorgesehen, aber aktuell ebenfalls noch nicht implementiert. Die automatische Anpassung dient zur besseren Nutzung der Bandbreite bei TCP/IP. Die Puffergröße kann dann abhängig davon, ob wenige große Dateien oder viele kleine Dateien übertragen werden sollen, gesetzt werden.

Der Ablauf eines Datentransfers ist eigentlich relativ einfach. Als erstes muss sichergestellt, dass

a) der Client berechtigt ist, auf den Server zuzugreifen und

b) dass der Server auch der ist, der er zu sein vorgibt (Man in the Middle Attack)

Das kann z.B. dadurch geschehen, dass sich beide an zentraler Stelle authentifizieren. Single Sign-On per GSI machts möglich. Wenn die Identitäten geklärt sind und der Client die nötigen Rechte für den Zugriff auf den GridFTP-Server hat, dann baut der Client einen Kontrollkanal zum Server auf (im Falle des Third-party Transfers baut er zu zwei Servern je einen Kontrollkanal auf) und stößt den Datentransfer mit Hilfe der GridFTP-Kommandos an. Durch den Schichtaufbau der Archtitektur ist es dabei möglich, dem Server alle möglichen Datenquellen – angefangen bei lokalen, über verteilte und parallele Dateisysteme auf Einzelmaschinen oder Clustern bis hin zu anderen striped GridFTP Servern – zur Verfügung zu stellen. In Abb. 9.3 ist der Zustand der beteiligten Komponenten während des Datentransfers dargestellt.

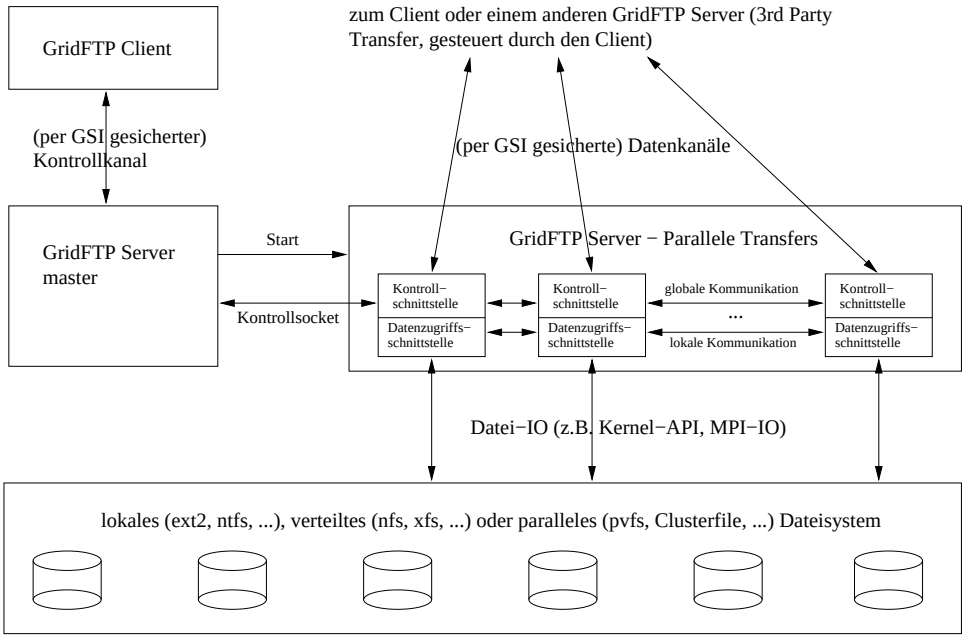

Abbildung 9.3: Schematische Darstellung der GridFTP Architektur [ACF+01]

9.5 Schlussbemerkung

Insgesamt bleibt festzuhalten, dass Grid-Computing sich noch im Anfangsstadium befindet, was die Verfügbarkeit von Tools und die Implementierung von Protokollen angeht. Das *Globus Toolkit* stellt bereits eine gute Ausgangsbasis dar, selbst wenn die Implementierung noch nicht vollständig ist. Auch in Sachen Benutzerfreundlichkeit, was die Installationsroutine und die Dokumentation angeht, gibt es ebenfalls noch einige Defizite.

Ein weiteres interessantes Projekt stellt PlanetLab dar, dass durch die Uniformität der Knoten, realisiert durch die virtuelle Container, eine gute Testplattform für neue Protokolle und Dienste darstellt (siehe auch Beitrag von Alexander Dieterle). Es steht dabei nicht in Konkurenz zu Globus, sondern die beiden Projekte können sich gut ergänzen [RBC+04].

Bei meinen Recherchen ist mir aufgefallen, dass sich die Publikationen momentan auf kollektive Ressourcenverwaltung konzentrieren. Die Forschung im Bereich Basis-, Verbindungs- und Ressourcenschicht scheint also nicht mehr soviel herzugeben und statt dessen werden jetzt die Möglichkeiten der oberen Schichten der Architektur vermehrt analysiert. Das bedeutet, dass die Konzepte im Bereich Infrastruktur den Status produktionstauglich besitzen, was schon mal einen sehr guten Schritt in die richtige Richtung darstellt. Es wird aber vermutlich noch ein paar Jahre dauern, bis das *Globus Toolkit* und andere Grid-Middleware soweit gereift sind, dass die meisten der gesteckten Ziele erreicht sind.

Literaturverzeichnis

[ABF+02] William Allcock, John Bresnahan, Ian Foster, Lee Liming, Joseph M. Link, and Pawel Plaszczac. Gridftp update january 2002. Technical report, January 2002.

[ACF+01] William Allcock, Ann L. Chervenak, Ian Foster, Laura Pearlman, Von Welch, and Michael Wilde. Globus toolkit support for distributed data-intensive science. In *Proceedings of Computing in High Energy Physics (CHEP '01)*, September 2001.

[All03] William Allcock. *GridFTP Protocol Specification (Global Grid Forum Recommendation GFD.20)*, March 2003.

[FI03] Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, February 2003.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. In *International J. Supercomputer Applications, 15(3)*, 2001.

[FKTT98] Ian Foster, Carl Kesselmann, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *ACM Conference on Computers and Security*, pages 83–91, 1998.

[Fos02] Ian Foster. What is the grid? a three point checklist. In *GRIDToday*, July 2002.

[GM90] Morrie Gasser and Ellen McDermott. An architecture for practical delegation in a distributed system. In *Proceedings 1990 IEEE Symposium in Research in Security and Privacy*, pages 20–30. IEEE Press, 1990.

[Hal04] Nicole Hallama. Präsentation: Grid security infrastructure - gsi, April 2004. Universität Potsdam.

[RBC+04] Matei Ripeanu, Michael Bowman, Jeffrey S. Chase, Ian Foster, and Milan Milenkovic. Globus and planetlab resource management solutions compared. In *13th Proceedings of the International IEEE Symposium on High Performance Distributed Computing*, June 2004.

Kapitel 10

Scheduling and Resource Management in Condor

Seminarbeitrag von **Martin Heine**

Bei Scheduling auf Rechnerbündeln oder Grids fällt es häufig schwer, guten Durchsatz, Robustheit, Fehlertoleranz und Anwendbarkeit auf möglichst vielen Anwendungsklassen gleichzeitig zu realisieren. In dieser Arbeit werden die Methoden und Lösungen vorgestellt, mit denen Condor versucht, genau diese teils gegensätzlichen Ziele in Einklang zu bringen.

10.1 Einführung

In der heutigen Zeit werden zur Bearbeitung größerer Aufgaben häufig parallele Systeme eingesetzt, in denen mehrere Prozessoren gleichzeitig die Aufgabe bearbeiten. Da einzelne Maschinen mit einer größeren Anzahl von Prozessoren sehr kostspielig sind, immer mehr verteilte Systeme eingesetzt. Diese *Rechnerbündel* (engl. *cluster*) oder *Grids* bestehen aus mehreren Rechnern mit jeweils einem oder mehreren Prozessoren, die über ein Netzwerk miteinander kommunizieren. Rechnerbündel sind im Gegensatz zu einzelnen Hochleistungssystemen durch das Anschließen weiterer Rechner erweiterbar. Die einzelnen Rechner unterscheiden sich häufig nur wenig von leistungsfähigen Arbeitsplatzrechnern.

In Rechnerbündeln gibt es meistens eine Instanz, die alle Knoten koordiniert. Der Zweck dieser Instanz ist die *Ressourcenverwaltung* (engl. *resource management*) und beinhaltet das Zuweisen von Aufgaben auf die einzelnen Ressourcen sowie das Kontrollieren der Abarbeitung der zugewiesenen Aufgaben. Ein Spezialfall von Ressourcenverwaltung ist die zeitliche Zuteilung von Aufgaben an den Prozessor, die als *Scheduling* bezeichnet wird. Im Falle von Ressourcenverwaltung in einem Rechnerbündel bilden die Knoten die zu verwaltenden Ressourcen.

Das vorrangige Ziel der Ressourcenverwaltung ist es, die zu bearbeitenden Aufgaben möglichst so im Rechnerbündel zu verteilen, dass alle Aufgaben möglichst schnell bearbeitet werden. Um die Rechenleistung des Rechnerbündels möglichst gut auszuschöpfen, versucht die Ressourcenverwaltung zu erreichen, dass jeder Knoten zu jedem Zeitpunkt von einem Prozess genutzt wird. Diese hohe Prozessorauslastung ist aber schwer zu bewerkstelligen, da die auf die einzelnen Prozessoren verteilten Teilaufgaben häufig zeitlich voneinander abhängen und dadurch in vielen Fällen aufeinander warten müssen. Zudem benötigen fast alle Prozesse mehr als eine Ressource und das Migirieren von Prozessen in verteilten Systemen ist nicht oder nur mit hohem Aufwand möglich.

Eine weitere Anforderung an die Ressourcenverwaltung ist Fairness. Das bedeutet, dass jeder Prozess jede Ressource tatsächlich auch irgendwann zugeteilt bekommt, so dass alle Prozesse gleichermaßen die Möglichkeit erhalten voranzuschreiten.

10.2 Scheduling

Zu Scheduling in parallelen Systemen gibt es verschiedene Ansätze. *Dedizierte Scheduler* gehen von einer gleich bleibenden Ressourcensituation aus, *opportunistische Scheduler* hingegen arbeiten auch mit nur zeitweise verfügbaren Ressourcen. Condor versucht die Vorteile beider Ansätze in einem System zu vereinigen.

10.2.1 Opportunistisches und dediziertes Scheduling

Charakteristisch für opportunistische Scheduling-Systeme ist der Einsatz von Ressourcen, die nicht ausschließlich dem Scheduling-System zur Verfügung stehen, sondern zu nicht vorhersagbaren Zeitpunkten für andere Zwecke verwendet werden. Meistens handelt es sich dabei um normale Arbeitsplatzrechner, die, wenn sie nicht gerade von ihrem jeweiligen Benutzer gebraucht werden, dem Scheduling-System zur Verfügung stehen. Wenn ein Benutzer seinen Rechner wieder benötigt, wird der Zustand des dort laufenden Prozesses gerettet und auf einen anderen Rechner migriert. Opportunistisches Scheduling basiert also auf der Nutzung zufällig ungenutzter Rechenzeit.

Dediziertes Scheduling setzt eine konstante Anzahl ständig verfügbarer Ressourcen voraus und kann aufgrund der Kenntnis der Ressourcenverfügbarkeit langfristige Pläne erstellen und im Vergleich zum opportunistischen Scheduling präzisere zeitliche Vorraussagen über die Abarbeitung von Aufgaben machen.

10.2.2 Schedulingarten im Vergleich

Der zentrale Nachteil von dediziertem Scheduling liegt in der Annahme begründet, dass jede Ressource jederzeit verfügbar sei. In der Praxis gibt es allerdings Wartungszeiten, Ausfälle und Prozesse, die auf das Fortschreiten anderer Prozesse warten müssen. Diese Phänomene haben zur Folge, dass Knoten nicht verfügbar sind, und treten mit steigender Knotenanzahl vermehrt auf. Diese Ereignisse werden bei Annahme dedizierter Ressourcen nicht berücksichtigt und sind daher auch nicht ohne weiteres kompensierbar.

Die meisten Systeme, die dediziertes Scheduling verwenden, sind auf spezielle Aufgabenstellungen zugeschnitten und generell wenig flexibel. Benutzer, die sowohl serielle als auch parallele Aufgaben zu erledigen haben, müssen dann häufig zwei verschiedene Systeme verwenden. Da unterschiedliche Systeme auch unterschiedliche Oberflächen und Kontrollmöglichkeiten bieten, ist der Benutzer dann gezwungen,

die Bedienung zweier Systeme zu lernen. Da die Administratoren dann zwei verschiedene Systeme installieren, warten und betreuen müssen, steigen auch dort Kosten und Aufwand. Zudem lassen sich verschiedene Scheduling-Systeme selten zur gleichen Zeit auf der gleichen Hardware betreiben, sodass häufig die Hardwareressourcen zwischen beiden Systemen aufgeteilt werden, wodurch die Leistungsfähigkeit des einzelnen Systems eingeschränkt wird.

Die feste Aufteilung der Hardwareressourcen zwischen zwei Systemen verschlechtert auch die Nutzung der Rechenleistung, da bei verschiedener Beanspruchung der beiden Systeme die nicht genutzte Leistung aus dem weniger belasteten nicht zur Entlastung des anderen, mehr belasteten Systems genutzt werden kann.

10.2.3 Scheduling bei Condor

Ursprünglich betrieb Condor ausschließlich opportunistisches Scheduling, die heutigen Versionen jedoch versuchen die Vorteile von dediziertem und opportunistischem Scheduling in einem System zu vereinen [**?**]. Opportunistisches Scheduling ermöglicht es, Hardwareressourcen, die nicht ausschließlich zum Rechnerbündel gehören, zu nutzen. Condor benutzt dabei ein Verfahren namens *Checkpointing*, das verhindert, dass beim Entzug eines Knotens aus dem Rechnerbündel die dort laufenden Prozesse verloren gehen. Checkpointing schreibt in diesem Fall den Zustand aller dort laufenden Prozesse in eine Datei. Diese Datei kann von Condor dann dazu benutzt werden, die Prozesse auf einem anderen Knoten wiederherzustellen und fortzuführen.

Um das gleichzeitige Betreiben verschiedener Scheduling-Systeme zu vermeiden, unterstützt Condor sowohl MPI (Message Passing Interface) als auch PVM (Parallel Virtual Machine). So können sowohl MPI- als auch PVM-Anwendungen mithilfe von Condor ausgeführt werden.

Zusätzlich zum opportunistischen Scheduling beherrscht Condor seit Version 6.3.0 auch dediziertes Scheduling und somit die effiziente Nutzung von dedizierten Ressourcen.

10.3 Der Aufbau eines Condor-Systems

Das eigentliche Condor-System besteht aus einer Menge von im Hintergrund arbeitenden Prozessen, die auf die verschiedenen Knoten des Rechnerbündels verteilt sind. Der zentrale Prozess, der Informationen über alle Knoten im Condor-System sammelt, nennt sich *Sammler* (engl. *collector*). Der Sammler-Prozess kommuniziert mit den anderen Prozessen durch den Empfang von *Klassenanzeigen* (engl. *ClassAds*). Klassenanzeigen sind Datenstrukturen, mit denen Aufgaben, Knoten und andere Ressourcen in Condor beschrieben werden. Sie bestehen aus einer Anzahl von Attributen, in denen der Zustand der beschriebenen Entität gespeichert ist. Mit Klassenanzeigen werden sowohl Meldungen über die Verfügbarkeit einzelner Entitäten als auch Anfragen nach Ressourcen an den Sammler-Prozess gesandt. Das Funktionsprinzip der Klassenanzeigen ist vergleichbar mit dem von Zeitungsanzeigen, denn sowohl Anfragen als auch Angebote werden zentral beim Sammler gespeichert.

Den Knoten, auf dem der Sammler-Prozess läuft, nennt man den *zentralen Verwalter* (engl. *central manager*). Ein weiterer Prozess, der nur auf dem zentralen Verwalter zu finden ist, ist der *Vermittler* (engl. *negotiator*). Er gleicht die beim Sammler-Prozess angekommenen Anfragen mit den Verfügbarkeitsmeldungen ab und sucht nach Übereinstimmungen. Bei einer gefundenen Übereinstimmung wird der Sender der übereinstimmenden Klassenanzeigen benachrichtigt. Auf jedem Knoten in Condor, der für die eigentliche Abarbeitung von Aufgaben zur Verfügung ste-

Klassenanzeige eines Rechners:

```
Type = "Machine";
OpSys = "Linux";
Arch = "INTEL";
Memory = 512M;
Constraint = true;
```

Klassenanzeige einer Aufgabe:

```
Type = "Job";
Owner = "run_sim";
Constraint =
other.Type == "Machine" &&
Arch == "INTEL" &&
Opsys == "Solaris251";
```

Abbildung 10.1: Beispiele von Klassenanzeigen

hen soll, läuft ein Prozess namens `Startd` . Dieser Prozess beobachtet den Status seines Knotens und sendet diesen gegebenenfalls als Klassenanzeige zum Sammler-Prozess auf dem zentralen Verwalter. Bei einer solchen Klassenanzeige wird aber nicht nur der Status des Knotens berücksichtigt, sondern noch eine ganze Reihe von Einstellungen und Präferenzen, die vom Besitzer des Knotens gesetzt werden können. So kann der Besitzer bestimmen, wer wann wie lange und unter welchen Umständen die Rechenzeit seines Systems in Anspruch nehmen darf. Weiterhin ist `Startd` auch für die Verwaltung und Ausführung der von Condor auf seinen Knoten verteilten Prozesse verantwortlich. Da Knoten, auf denen der Prozess `Startd` läuft, potentielle Knoten für die Abarbeitung von an Condor übergebene Aufgaben sind, nennt man sie *ausführende Maschinen* (engl. *execute machines*).

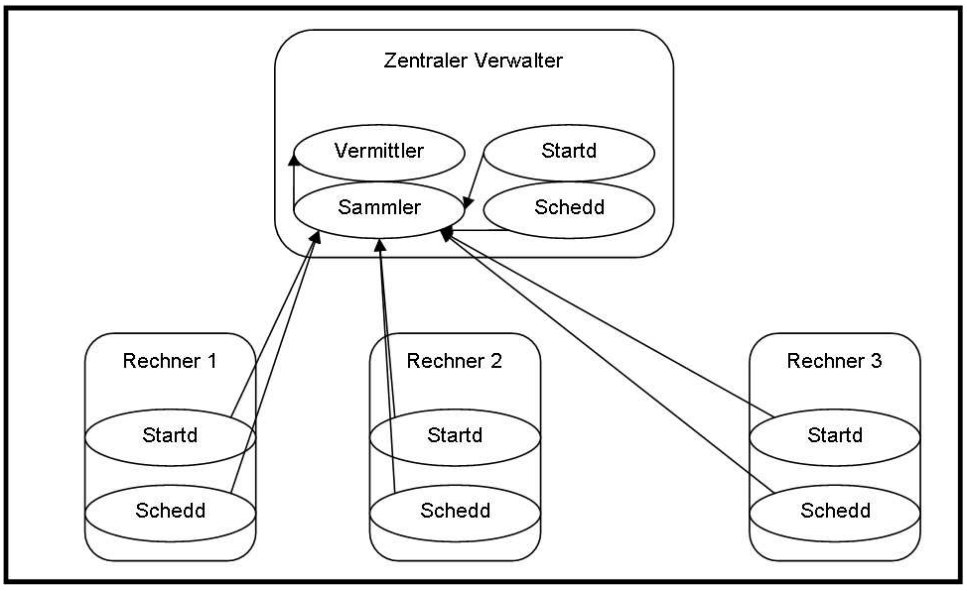

Abbildung 10.2: Aufbau von Condor

Wenn ein Benutzer eine Aufgabe zur Abarbeitung an Condor übergibt, wird sie dort in Form einer Klassenanzeige an den Sammler übergeben. Diese Klassenanzeige wird von einem Prozess `Schedd` erzeugt und verschickt. Wenn der Vermittler eine Übereinstimmung findet, wird der Prozess `Schedd` benachrichtigt. Dieser fordert die benötigten Ressourcen vom Prozess `Startd` an und führt auf den ihm vom Prozess `Startd` zugeteilten Ressourcen ein lokales Scheduling durch. Ein Knoten, auf dem der Prozess `Schedd` läuft, nennt man *Übergabemaschine* (engl. *submit machine*), da

man von diesem Knoten aus Aufgaben zur Abarbeitung an Condor übergeben kann.

Damit ein Prozess, wenn er von Condor auf einer ausführenden Maschine abläuft, genauso funktionieren kann, als ob er lokal auf der Übergabemaschine ausgeführt würde, wird auf der Übergabemaschine ein *Stellvertreterprozess* (engl. *shadow process*) erzeugt. Dieser gibt, um den abzuarbeitenden Prozess auf der ausführenden Maschine zu starten, dem dortigen `Startd`-Prozess die Anweisung, einen Starter zu erzeugen. Der Starter ist für die Ausführung des abzuarbeitenden Prozesses und für die Kommunikation mit dem Stellvertreterprozess auf der Übergabemaschine zuständig. So fängt der Starter alle Systemaufrufe des von ihm gestarteten Prozesses ab und leitet diese über das Netzwerk zur Übergabemaschine. Dort werden sie vom Stellvertreterprozess lokal ausgeführt, der die Ergebnisse wieder zurück zur ausführenden Maschine schickt. Dieses Verfahren zum entfernten Starten von Prozessen entspricht einem entfernten Funktions- oder Methodenaufruf, nur dass der ausgeführte Prozess durch den Starter und den Stellvertreterprozess transparent auf Ressourcen der Übergabemaschine zugreifen kann. Dadurch können auch Prozesse entfernt ausgeführt werden, die nicht explizit für Fernaufrufe konzipiert sind.

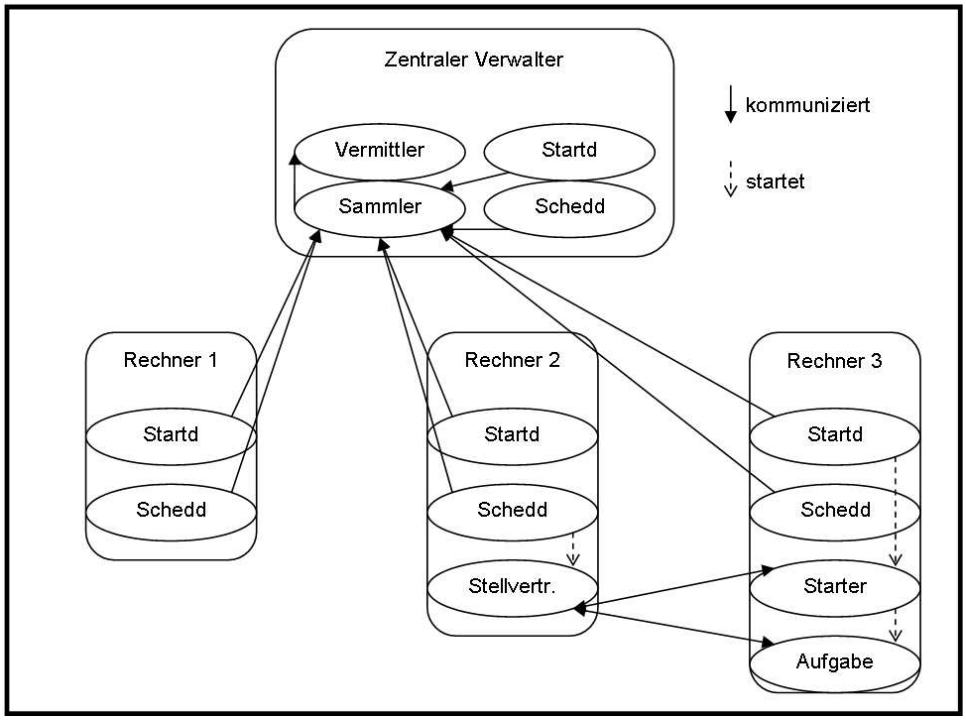

Abbildung 10.3: Bearbeitung von Aufgaben in Condor

Da auf den verschiedenen Knoten in Condor verschiedene und auch jeweils mehrere Condor-Prozesse laufen, gibt es auf jedem Knoten noch einen zusätzlichen Condor-Prozess, den *Master*, der alle Prozesse von Condor auf dem jeweiligen Knoten verwaltet. Neben dem Starten und Verwalten der lokalen Prozesse stellt der Master auch eine Schnittstelle dar, den Condor-Knoten über das Netzwerk zu administrieren.

10.4 Dediziertes Scheduling

Durch die Kombination von opportunistischem und dediziertem Scheduling läuft das dedizierte Scheduling bei Condor unter anderen Rahmenbedingungen als bei

herkömmlichen Scheduling-Systemen. Dieser Umstand führt zu einer Reihe neuer Ansätze zum Erreichen von hoher Ressourcenauslastung und hohem Durchsatz.

10.4.1 Dediziertes Scheduling bei Condor

Die in Abschnitt 10.3 beschriebene Funktionsweise von Condor geht nicht von der Planbarkeit des Einsatzes der einzelnen Ressourcen aus, sondern gleicht Bedarf und Verfügbarkeit dynamisch ab, betreibt also opportunistisches Scheduling. Um dediziertes Scheduling zu ermöglichen, kann man bei Condor Ressourcen als dediziert markieren. Solche Ressourcen werden einem dedizierten Scheduler zugeordnet. Die als dediziert gekennzeichneten Ressourcen müssen dem dedizierten Scheduler immer zur Verfügung stehen.

Dieser Umstand schließt die Verwendung der dedizierten Ressourcen im opportunistischen Scheduling nicht aus, gewährt aber den Prozessen des dedizierten Schedulers unbedingten Vorrang. Daher werden die auf dedizierten Ressourcen laufenden `Startd`-Prozesse so konfiguriert, dass Prozesse des dedizierten Schedulers nie vom restlichen Condor-System unterbrochen oder auf andere Knoten migriert werden. Prozesse, die vom opportunistischen Scheduler auf eine dedizierte Ressource verteilt wurden, werden bei Bedarf des dedizierten Schedulers sofort unterbrochen. Der Status als dedizierte Ressource wird vom Prozess `Startd` mittels einer Klassenanzeige dem Sammler bekannt gegeben.

Sobald der Prozess `Schedd` auf einer Übergabemaschine feststellt, dass ihm eine Aufgabe übergeben wurde, die dediziertes Scheduling verlangt, wird der dedizierte Scheduler benachrichtigt. Dieser fragt beim Sammler an, welche dedizierten Ressourcen ihm zugeordnet sind, und gleicht diese mit der Aufgabe ab. Danach fordert er die benötigten Ressourcen beim Vermittler an, der diese unverzüglich frei macht und übergibt. Die Ressourcen sind ab diesem Zeitpunkt direkt dem dedizierten Scheduler zugeordnet und stehen dem opportunistischen Scheduler nicht mehr zur Verfügung. Ist die dedizierte Aufgabe abgearbeitet worden, gibt der dedizierte Scheduler die Ressourcen an den opportunistischen Scheduler zurück. Wenn schon während der Abarbeitung einzelne Ressourcen nicht mehr benötigt werden, können diese auch schon früher zurückgegeben werden.

10.4.2 Vorteile von Condor bei dediziertem Scheduling

Bei der Verteilung von abzuarbeitenden Aufgaben durch einen dedizierten Scheduler wird ein *Zuordnungsplan* (engl. *Schedule*) erstellt, in dem die Aufgaben zeitlich über die verfügbaren Knoten verteilt werden. Da es häufig vorkommt, dass zur Abarbeitung einer Aufgabe nicht alle Knoten benötigt werden, gibt es immer Lücken in diesen Zuordnungsplänen. Diese Lücken stellen immer ungenutzte Ressourcen dar und verschlechtern somit die Ressourcenauslastung. Die meisten Scheduling-Verfahren versuchen Lücken mit kleinen Aufgaben niedriger Priorität zu füllen. Das funktioniert natürlich nur, wenn man genügend kleine niederpriore Aufgaben zu bearbeiten hat, und selbst dann wird es Lücken im Zuordnungsplan geben, die sich nicht so füllen lassen.

An dieser Stelle bringt die Kombination von opportunistischem und dediziertem Scheduling Condor einen beträchtlichen Vorteil. So können ungenutzte Ressourcen während der Lücken im Zuordnungsplan einfach dem opportunistischen Scheduler zurückgegeben werden, der diese dann bei Bedarf nutzt. Der opportunistische Scheduler muss diese Ressourcen nicht mit einer Aufgabe belegen, die in die Lücke im Zuordnungsplan passt, sondern er kann sie frei benutzen, da der erbrachte Fortschritt bei der Rückgabe an den dedizierten Scheduler mithilfe von Checkpointing gesichert wird und so die Aufgabe auf einer anderen Ressource fortgeführt werden kann.

Die Zeitdauer, die die Abarbeitung einer Aufgabe braucht, ist im Allgemeinen nicht berechenbar. Daher verlangen viele dedizierte Scheduling-Systeme vom Benutzer bei der Übergabe einer Aufgabe, diese Zeit anzugeben. Da der Benutzer diese Zeitdauer nicht immer kennt, muss er manchmal die Abarbeitungsdauer, die er dem Scheduling-System angibt, raten oder schätzen. Schlecht geschätzte Abarbeitungsdauern beeinflussen die Leistungsfähigkeit des Scheduling-Systems, das auf der Basis von diesen einen Zuordnungsplan erstellt, erheblich. Ein Umstand, der diese Schätzungen noch schlechter macht, ist, dass die meisten Scheduling-Systeme Aufgaben, die ihre geschätzte Zeit überschreiten, abbrechen und so den Benutzer dazu bringen, die benötigte Zeit recht großzügig einzuschätzen.

Der dedizierte Scheduler von Condor kann, da er nicht benötigte Ressourcen an den opportunistischen Scheduler zurückgibt, davon ausgehen, dass die Ressourcenauslastung nicht davon abhängt, ob er alle dedizierten Ressourcen benutzt oder nicht. Aus diesem Grund muss er nicht lange in die Zukunft reichende Zuordnungspläne machen, sondern kann immer aktuell mit den verfügbaren Ressourcen, den gerade laufenden Aufgaben und den wartenden Aufgaben mittels einer Heuristik die bestmögliche Entscheidung treffen. Die von laufenden dedizierten Aufgaben benutzen Ressourcen werden einfach als nicht verfügbar betrachtet. Wenn die nächste zu bearbeitende Aufgabe mehr Ressourcen braucht als verfügbar sind, fordert der dedizierte Scheduler diese vom opportunistischen Scheduler an oder wartet, bis eine laufende Aufgabe Ressourcen nicht mehr benötigt.

10.4.3 Zukünftige Scheduling-Merkmale in Condor

Obwohl Condor bereits, wie im vorhergehenden Abschnitt dargelegt, einige Vorteile gegenüber herkömmlichen Scheduling Systemen hat, gibt es eine Menge noch nicht realisierter Merkmale, die Condor vermutlich noch effektiver machen dürften.

- **Verschiedene Benutzerprioritäten bei dediziertem Scheduling.** Bisher verwendet Condor eine zentrale Warteschlange für wartende Aufgaben. Bei der Verwendung von verschiedenen Warteschlangen für verschiedene Benutzer und der Abarbeitung der Aufgaben anhand von Benutzerprioritäten würden zwar die Ressourcen nicht besser ausgelastet werden, aber die erreichte Fairness würde steigen.

- **Bessere Heuristik zur Ressourcenanforderung und -freigabe.** Die bisher benutzte Heuristik ist recht einfach und bietet Raum für Verbesserungen.

- **Hierarchien von dedizierten Schedulern.** Bisher kann jede dedizierte Ressource nur einem dedizierten Scheduler zugeordnet werden. Mit hierarchisch angeordneten Schedulern könnte man Prozesse zwischen den Ressourcenpools der einzelnen Scheduler migrieren. Die einzelnen Knoten können dann, wenn sie nicht von ihrem lokalen Scheduler gebraucht werden, von anderen Schedulern genutzt werden.

- **Unterstützung mehrerer verschiedener ausführbarer Dateien bei MPI-Anwendungen.** Momentan unterstützt Condor bei MPI-Anwendungen nur eine ausführbare Datei. Manche MPI-Implementierungen unterstützen aber verschiedene ausführbare Dateien auf den verschiedenen Knoten. Bei Condor muss aber auf allen Knoten, auf denen die Anwendung laufen soll, dieselbe Datei ausgeführt werden. Die Unterstützung von verschiedenen ausführbaren Dateien führt zu einfacherem Code.

- **Unterstützung für andere MPI-Implementierungen als MPICH.** Damit würde Condor noch mehr Anwendungen unterstützen.

- **Ressourcenreservierungen.** Dieses Merkmal würde helfen, interaktive Anwendungen mit Condor auszuführen, da der Benutzer dann weiß, wann seine Anwendung läuft.
- **Checkpointing bei parallelen Anwendungen.** Condor beherrscht Checkpointing bisher nur bei seriellen Anwendungen. Checkpointing bei parallelen Anwendungen würde den Durchsatz steigern und die Ressourcenverwaltung deutlich flexibler machen. Bisher werden parallele Anwendungen im Fehlerfall komplett neu gestartet.

10.5 Goodput

Der opportunistische Scheduler von Condor funktioniert über den Abgleich von Ressoucenanfragen und Ressourcenangeboten. In diesem Abschnitt wird aufgezeigt, wie dieser Abgleich vonstatten gehen muss, um die vorhandenen Ressourcen möglichst effizient zu nutzen.

10.5.1 Grundsätzliches

Goodput ist definiert als die Zeitdauer, in der eine Anwendung einen entfernten Prozessor für ihre eigentliche Bearbeitung nutzt [**?**]. Es gibt verschiedene Ursachen, die eine Anwendung davon abhalten, mithilfe des Prozessors Fortschritte zu erzielen, zum Beispiel Warten auf das Netzwerk oder Warten auf Ein- und Ausgabe. Bei Condor gibt es zusätzlich noch Migrationen und den Fall, dass eine Ressource nicht mehr verfügbar ist und die Anwendung auf einem anderen Rechner vom letzten Aufsetzpunkt an weiterbearbeitet wird und somit alle nach dem Ablegen des Aufsetzpunktes verrichtete Arbeit verloren geht.

Auch die Netzwerkkommunikation und deren Geschwindigkeit ist ein Einflussfaktor auf den Goodput, denn Netzwerkkommunikation tritt beim Ausführen von Anwendungen auf entfernten Systemen häufig auf. So wird schon vor dem eigentlichen Start der Anwendung die auszuführende Datei auf das entfernte System übertragen. Während sich die Größe dieser Dateien meist in Grenzen hält, werden die zu übermittelnden Daten im Falle von Aufsetzpunkten schnell mehrere hundert Megabyte groß, da sie zum Beispiel Hauptspeicherabzüge und zwischengespeicherte Ausgabedaten enthalten. Eine weitere Ursache für Netzwerkkommunikation sind Festplattenzugriffe, da diese von der ausführenden Maschine zur Übergabemaschine, auf der die Anwendung an Condor übergeben wurde, über das Netzwerk weitergeleitet werden. Diese Festplattenzugriffe sind nicht vorhersehbar, auch kann ihr Auftreten nicht zeitlich beeinflusst werden.

10.5.2 Co-matching

Wenn neue Aufgaben an das Condor-System übergeben werden, vergleicht der Vermittler, wie in Abschnitt 10.3 beschrieben, die Klassenanzeigen der Anfragen mit den Klassenanzeigen der Verfügbarkeitsmeldungen. Wenn bei diesem Vergleich von Ressourcenanfrage und Ressourcenangebot nur die Rechenleistung berücksichtigt wird, wird der Vermittler bei einer Anfrageserie zu allen Anfragen schnellstmöglich Ressourcenangebote finden. Daraufhin werden in einem kurzen Zeitraum alle ausführbaren Dateien der Anfragen an die entsprechenden entfernten Systeme geschickt, zusätzlich werden eventuell noch einige Prozesse migriert, die von den neu gestarteten Prozessen verdrängt wurden. Ohne zusätzliche Maßnahmen wäre die Folge eine plötzliche hohe Auslastung des Netzwerks, die wiederum höhere Transaktionsdauern verursachen würde. Dadurch stiegen die Wartezeiten der einzelnen Anwendungen, und die Prozessorauslastung der einzelnen Systeme würde sinken.

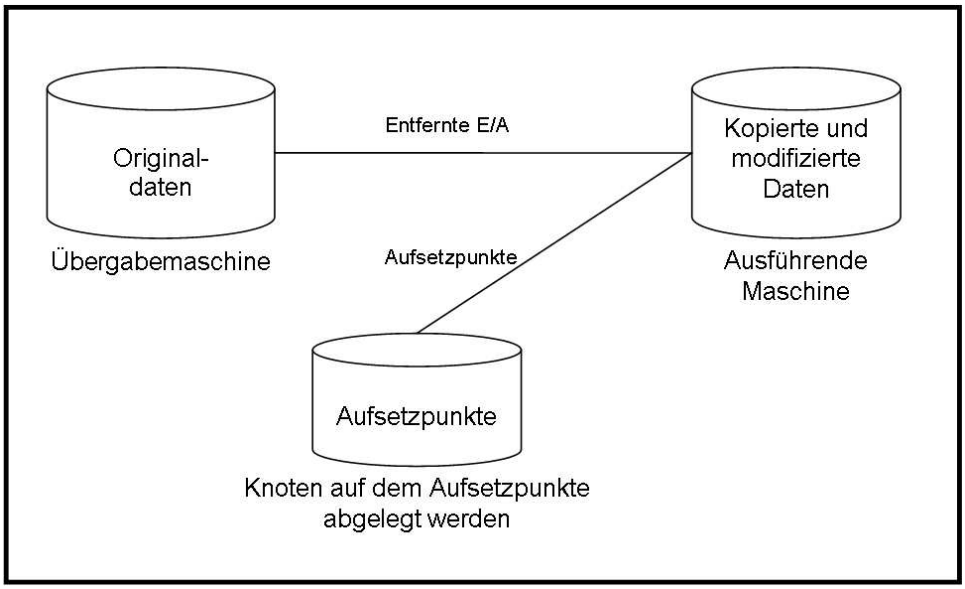

Abbildung 10.4: Netzwerkkommunikation in Condor

Um diesen Effekt zu vermeiden, wird beim Abgleich der Ressourcenanfragen und -angebote auch das Netzwerk als Ressource berücksichtigt. Die einzelnen Knoten senden hierzu in ihren Klassenanzeigen die aktuelle Auslastung ihres Subnetzes und die geschätzten Kosten einer Migration zum Sammler. In den Ressourcenanfragen wird die Größe der ausführbaren Datei und gegebenenfalls des zu ladenden Aufsetzpunktes übertragen, aus der sich der Bedarf an Netzwerkressourcen beim Start der Anwendung abschätzen lässt. Wenn der Vermittler nun eine Anfrage und ein Angebot vergleicht, überprüft er auch, ob die Netzwerkressourcen ausreichen, um die Anwendung zu starten. Ist dies nicht der Fall, wird nach einer anderen Übereinstimmung gesucht. Der Vermittler kennt die Netzwerkstruktur und bezieht die Netzwerkauslastung für jedes Subnetz mit ein. Der Abgleich einer Ressourcenanfrage mit einem Ressourcenangebot und der Netzwerkressource nennt man *co-matching*. Durch den Einbezug der Netzwerkressource in den Schedulingprozess wird verhindert, dass durch den Start neuer Anwendungen das Netzwerk überlastet wird.

10.5.3 Checkpoint Scheduling

Neben dem Start neuer Anwendungen ist auch das Übertragen eines Aufsetzpunktes zu dem zur Ablage vorgesehenen Knoten eine große Belastung für das Netzwerk. Um Netzwerküberlastung durch Checkpointing zu vermeiden, wird vor dem Anlegen eines Aufsetzpunktes der Scheduler um Erlaubnis gebeten, die nur dann erteilt wird, wenn es die Lastsituation im Netzwerk zulässt.

Der Knoten, auf dem die Aufsetzpunkte zentral abgelegt werden (engl. *checkpoint server*), ist an relativ vielen der Netzwerktransaktionen beteiligt, weshalb er einen guten Ansatzpunkt darstellt, um die Netzwerkbeanspruchung zu steuern. Im Gegensatz zum Lesen oder Erstellen eines Aufsetzpunktes, für das meistens keine feste zeitliche Grenze existiert, haben Migrationen oft eine feste Frist, bei deren Überschreitung der Migrationsversuch abgebrochen wird. Diesem Umstand kann mit einer höheren Priorität der kritischen Übertragungen Rechnung getragen werden. Auch können bei hohem Netzwerkverkehrsaufkommen alle Übertragungen nacheinander durchgeführt werden, um Bandbreite zu sparen.

Einen wesentlichen Einfluss auf die Netzwerkbelastung durch Checkpointing stellt die Häufigkeit der regelmäßig abgelegten Aufsetzpunkte dar. Bei dieser Problematik gilt es einen Kompromiss zwischen zum einen hoher Netzwerkbelastung durch häufiges Erstellen von Aufsetzpunkten und zum anderen seltenem Erstellen von Aufsetzpunkten mit hohem Verlust an Arbeit nach Fehlern oder Abbrüchen von Migrationsversuchen zu finden. Wo genau die optimale Lösung liegt, hängt stark von der Größe der einzelnen Aufsetzpunkte ab, da bei größeren Aufsetzpunkten bei deren Erstellung mehr Daten über das Netzwerk übertragen werden müssen, aber auch durch die längere Übertragungsdauer die Wahrscheinlichkeit von fehlschlagenden Migrationenversuchen steigt.

10.6 Zusammenfassung und Fazit

Der zentrale Vorteil von Condor ist seine Vielseitigkeit. So ist Condor ein System, mit dem man durch opportunistisches Scheduling die Leistungsfähigkeit vorhandener Systeme besser ausnutzen kann, aber gleichzeitig auch dediziertes Scheduling betreiben kann. Zudem unterstützt es sowohl PVM als auch MPI und kann sowohl serielle als auch parallele Anwendungen in einer verteilten Umgebung ausführen. Das alles gepaart mit der Robustheit durch Checkpointing und den Vorteilen aus der Kombination von opportunistischem und dediziertem Scheduling machen Condor für ein breites Feld an Nutzern interessant.

Index