Chord

Till Janetzki
(till@literaturport.de)

Seminar „Internet Routing“ ,
Technische Universität Berlin

WS 2007/2008 (Version vom 18. Januar 2008)

Zusammenfassung

Dieses Papier behandelt Chord, ein Protokoll zum Aufbau, Betrieb und effektiven Durchsuchen großer Peer-to-Peer (P2P) Netzwerke. Trotz seines vergleichsweise einfachen Aufbaus, garantiert Chord dabei die Robustheit, Korrektheit und Geschwindigkeit des Netzes. Chord ist kein fertiges Produkt sondern in stetiger Entwicklung. Es hat damit eher experimentelle und pädagogische als praktische Bedeutung.

1 Einleitung

Die Organisation eines kleinen P2P Netzwerkes mit wenigen dutzend Teilnehmern (Knoten) läßt sich auch mit trivialen Algorithmen sehr effektiv bewältigen. Beispielsweise kann jeder Knoten die Liste aller anderen Knoten zwischenspeichern und so in kürzester Zeit Adressen und Informationen der anderen Teilnehmer ermitteln. Ein solcher Ansatz funktioniert nur für ausgesprochen kleine Netze und skaliert quasi überhaupt nicht. Chord stellt einen Ansatz vor, mit dem auch extrem große P2P Netze realisierbar werden, ohne dabei viele schmerzhafte Kompromisse eingehen zu müssen.

1.1 Motivation

Eine zentrale Herausforderung von P2P Netzen ist das schnelle aber dennoch zuverlässige Auffinden von Informationen obwohl lediglich ein verschwindend geringer Teil der anderen Knoten bekannt ist. Eine zentrale Koordinierungsstelle gibt es nicht und bedingt durch die Struktur des Internets können Knoten jederzeit vorübergehend oder permanent ausfallen. Chord bietet hierfür eine elegante Lösung an, deren Komplexität lediglich logarithmisch mit der Anzahl der Knoten wächst.

1.2 Aufbau dieses Papiers

Im Folgenden soll zunächst der grundlegende Aufbau sowie die Basisoperationen des Suchens und Einfügens von Knoten gezeigt werden. Die Algorithmen werden grob in ihrer Funktionsweise erklärt ohne dabei auf Beweise oder Detailinformationen einzugehen. Anschliessend wird diskutiert welche Vor- und Nachteile aus der Architektur eines Chord Netzes resultieren und welche Anwendungsfälle für ein solches Netz bestehen. Im vierten Abschnitt werden einige Probleme in der internen Verwaltung eines Netzes vorgestellt und die jeweilige Loesungsidee kurz skizziert. Der letzte Abschnitt bewertet kurz Chord als Ganzes und stellt einige darauf basierende Applikationen vor.

2 Funktionsweise von Chord

Chord ist modular, als untere Schicht in einem Softwaresystem, konzipiert [DBK+01]. Im Wesentlichen bildet es frei wählbare Schlüssel auf einen Wertebereich ab und liefert zu einem solchen beliebigen Schlüssel s die ID des jeweiligen, verantwortlichen Knotens zurück. Die Verwaltung der zu speichernden oder abzurufenden Daten muss das Softwaresystem leisten. Chords API bietet dazu lediglich Hilfsfunktionen an. Jeder Knoten hat eine exakt definierte Zuständigkeit für Informationen und kennt nur einen (minimal) kleinen Teil des gesamten Netzes.

2.1 Aufbau

Chord ordnet dazu die Knoten auf einem virtuellen Ring an. Jeder Knoten kennt seinen direkten Vorfahren und Nachfolger und erhält außerdem dazu eine Nummer (ID) die seine Position auf dem Ring kenntlich macht. Der Ring hat eine Größe von 2^m möglichen IDs und wird von 0 bis $2^m - 1$ durchnummeriert.

Um ein performantes Kommen und Gehen von Knoten zu ermöglichen nutzt Chord eine Form des konsistenten Hashings [HT03] um Knoten und Schlüssel im selben Zahlenraum, also auf dem selben virtuellen Ring abzulegen. Die ID eines Knotens berechnet sich aus dem Hashwert seiner IP-Adresse, seines Ports und einiger weiterer Daten modulo 2^m. Das konsistente Hashing sorgt dafür das Knoten ohne Restrukturierung des gesamten Netzes eingefügt oder entfernt werden können. Die eigentliche Idee besteht darin, die zu speichernden oder abzurufenden Informationen auf den gleichen Zahlenraum zu hashen wie die Knoten. Jeder Information wird eine ID auf dem Ring zugewiesen und der jeweils nachfolgende Knoten für verantwortlich erklärt. Kommt ein neuer Knoten dazu, so nimmt er seinem nachfolgenden Knoten die Verantwortung für alle die Schlüssel ab, deren ID kleiner oder gleich seiner eigenen ID ist (Nachfolger). Verlässt ein Knoten den Ring, ist es Aufgabe des Systems dafür zu sorgen, dass der Nachfolgerknoten seine nunmehr gestiegene Verantwortung wahrnehmen kann.

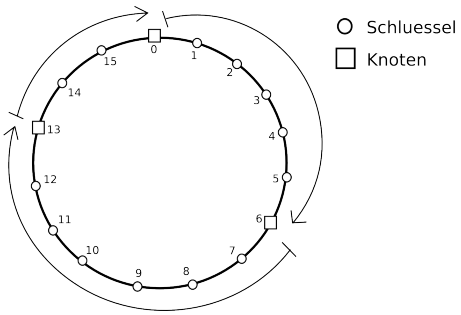

Abbildung 1: Chord Netz mit drei Knoten (0,6,13) und $m = 4$. Die Knoten sind jeweils für die IDs der Schlüssel vom Vorgängerknoten bis zu Ihrer eigenen ID zuständig.

2.2 Fingertabelle

Ein solcher Ring liesse sich bereits verwenden, nur müssten für jede Suche im schlimmsten Fall alle Knoten des Rings einmal durchlaufen werden. Um dies zu beschleunigen speichert Chord in jedem Knoten eine sogenannte Fingertabelle. Finger eines Knotens k werden die Knoten genannt, die der ID von k an 1, 2, 4, 8, 16, ..., 2^{m-1} Stelle im Ring folgen. Gibt es nicht an jeder dieser Stellen einen Knoten, so wird der jeweils nachfolgende

und damit verantwortliche Knoten in die Tabelle eingetragen. Gibt es im Ring größere Lücken, so haben verschiedene Startwerte dennoch den selben Finger. [1]

Definition 1 *Sei n die ID eines Knotens k. Der i-te Startwert von k berechnet sich dann aus* $Start_i = n + 2^{i-1} \mod 2^m$ *für alle i:* $0 < i < m$.

I	KNOTEN 0	Start	Finger	KNOTEN 13	Start	Finger
1		1	6		14	0
2		2	6		15	0
3		4	6		1	6
4		8	13		5	6

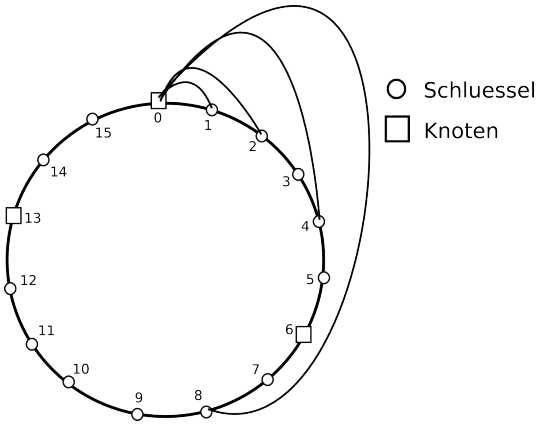

Abbildung 2: Die Start-Einträge der Fingertabelle von Knoten 0. Die Finger sind die der ID des Startwerts nachfolgenden Knoten (hier 6 und 13).

2.3 Suchen mittels der Fingertabelle

Mit den Fingern ist Chord nun in der Lage die Wege duch den Ring extrem zu verkürzen. Liegt eine gesuchte Information (und damit die ID) weit entfernt auf dem Ring, so bittet der Knoten einfach seinen größtmöglichen Finger um Hilfe, dessen ID gerade noch Unterhalb der gesuchten liegt. Da jeder Knoten mehr über seine nah gelegenen Nachfolger als über entfernte Knoten weiss , ist der neue Knoten besser in der Lage die Suchanfrage weiter zu bearbeiten.

Auf diese Weise bewegt man sich sehr schnell durch den Ring. Da die Fingerabstände quadratisch wachsen, legt man mit jedem Schritt mittels eines Fingers mindestens die halbe Distanz zum gesuchten Ziel zurück.

Definition 2 *Eine beliebige Anfrage braucht für* $n = 2^m$ *mit hoher Wahrscheinlichkeit höchstens* $O(logn)$ *Schritte um zum Ziel zu kommen.*

Durch z.B. ausfallende Knoten und damit inkorrekte Fingertabellen kann die Suche verzögert werden. Es läßt sich dennoch zeigen, dass die Anzahl von benötigten Schritten im Mittel $\frac{1}{2}logn$ nicht übersteigt [SMK+01].

[1] In der Implementierung von Chord werden solche Finger nicht gespeichert sondern erst ggf. später ergänzt. Im der folgenden Tabelle sind sie aber dennoch aufgeführt.

Beispiel In Abbildung 3 sucht Knoten 0 den Nachfolger von Schlüssel 11. In seiner Fingertabelle stehen nur die Knoten 6 und 9. Da beide IDs kleiner sind als 11 wählt Knoten 0 Knoten 9 als den größeren der beiden, um mit der Suche fortzufahren. Knoten 9 wiederholt jetzt die gleichen Schritte, ist aber deutlich näher am Ziel. Er hat nur einen Finger dessen ID kleiner als die gesuchte ist, der also damit in Frage kommt: Knoten 10. Knoten 10 kennt keine weiteren kleinen Knoten. Der Nachfolgerknoten muss also der Nachfolger von 11 sein und ist damit verantwortlich für die dort abgelegten Informationen. Knoten 10 informiert also Knoten 0, dass der Nachfolger von Schlüssel 11 Knoten 13 ist.

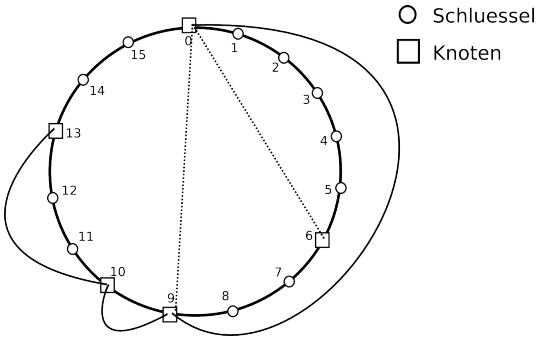

Abbildung 3: Knoten 0 sucht den verantwortlichen Knoten (Nachfolger) von ID 11. Die gepunkteten Linien zeigen nochmals die Finger des Knotens 0.

2.4 Einfügen von neuen Knoten

Um das Netz in der Praxis zu benutzen muss es möglich sein, jederzeit weitere Knoten aufzunehmen ohne die Korrektheit der Anfragen zu gefährden. Eine Anfrage kann dann korrekt beantwortet werden wenn zumindest die Vorgänger und Nachfolger jedes Knotens korrekt sind. Zwar müssen auch die Fingertabellen anderer Knoten angepasst werden, da jedoch immer der direkte Vorgänger Auskunft über den gesuchten Nachfolger gibt, verlangsamt das lediglich die Suche, gefährdet aber nicht die Korrektheit. Es genügt also zunächst den beiden Nachbarknoten den neuen Knoten bekanntzumachen und ebenso die Nachbarknoten im neuen Knoten einzutragen.

Ein neuer Knoten k braucht zuerst einen beliebigen Knoten aus dem bestehenden Netz, wenn er nicht selbst den erste Knoten bilden will. Mittels der Hashfunktion errechnet er seine eigene ID und fragt dann den bekannten Knoten nach dem Nachfolger dieser ID, Knoten z. Ist dieser gefunden, so setzt Knoten k Knoten z als seinen Nachfolger ein und informiert z darüber, dass er der neue Vorgängerknoten ist. Das umgebende Softwaresystem muss nun dafür sorgen, dass alle Schlüssel für die k nun verantwortlich ist übertragen werden, bevor k Anfragen beantworten kann.

Der neue Knoten k kann nun bereits Suchanfragen an das Netz stellen, ist aber selbst noch keinem anderen Knoten bekannt. Dies geschieht durch eine von jedem Knoten periodisch aufgerufene Stabilisierungsfunktion (siehe auch Abschnitt 4). Dabei fragt jeder Knoten seinen Nachfolger, ob er selbst immernoch sein Vorgänger ist. Ist das nicht der Fall, aktualisiert er seinen Nachfolger und informiert diesen über den neuen Vorgänger.

Der Knoten muss jetzt noch seine Fingertabelle füllen und ist dann vollwertiger Knoten im Netzwerk.

3 Eigenschaften von Chord

Die Architektur von Chord bringt einige wünschenswerte Eigenschaften mit sich, die aber zugleich andere Dinge erschweren.

3.1 Vorteile

Lastverteilung Durch die Nutzung der Hashfunktion werden die Knoten mehr oder weniger gleichmässig im Netz verteilt. Da der Abstand zum letzten Knoten die Menge an Informationen begrenzt, die ein Knoten speichern kann oder muss, sollten die Abstände möglichst gleichverteilt sein. Da das zwar im Mittel erreicht wird, es aber dennoch zu erheblichen Unterschieden kommen kann, ist es möglich auf jedem Knoten mehrere virtuelle Knoten laufen zu lassen. Damit steigt die Wahrscheinlichkeit für eine nur unwesentlich vom Mittelwert abweichende Informationsmenge zuständig zu sein. Knoten mit hoher Bandbreite, lassen einfach mehr virtuelle Knoten laufen.

Dezentralisierung Chord ist ein reines Peer-to-Peer Protokoll. Auf jedem Knoten läuft die identische Software, es gibt keinen Knoten dem höhere Bedeutung als einem anderen Knoten zukommt.

Skalierbarkeit Mit logarithmischen Kosten für eine Suchanfrage skaliert Chord ausgezeichnet auch bei extrem hohen Teilnehmerzahlen.

Verfügbarkeit Chord garantiert korrekte Suchanfragen für einen beliebigen Zustand des Netzes. Fällt ein Knoten aus oder kommt er dazu, reagiert Chord automatisch mit der Delegation der Verantwortlichkeiten. In einem System das Daten speichert ist Chord dabei allerdings auf Mithilfe der Replizierenden Softwareebene angewiesen um Daten redundant auf anderen Knoten verfügbar zu halten[Dab].

Flexible Namensgebung Durch die freie Wahl, wie Informationen gehasht werden, ist Chord sehr flexibel einzusetzen. Es lassen sich Indizes, wie Daten selbst oder beliebige andere Objekte auf dem Ring ablegen.

Robustheit Chord zeigt sich relativ unempfindlich gegenüber Angriffen und Netzausfällen. Durch die Verwendung der IP-Adresse in der Berechnung der ID ist es nicht möglich, eine beliebige Position im Ring einzunehmen um damit gezielt Verantwortung für einen zu manipulierenden Bereich zu übernehmen. Auch ist es nahezu unmöglich aufeinander folgende Positionen zu beziehen, um so den Ring zu partitionieren oder einzelne Knoten vom Netz zu trennen [SM02]. In [LNBK02] wird ein Algorithmus vorgestellt, der ein Chord Netz, das durch den Ausfall vieler Knoten in einen ungültigen oder zumindest suboptimalen Status gekommen ist, wieder repariert. Allerdings wird dazu kurzzeitig die Aufnahme neuer Knoten blockiert.

3.2 Probleme

fehlende Anonymität Die enge Bindung der ID an die IP-Adresse erschwert die oft wünschenswerte Anonymität erheblich. Zudem läuft eine Anfrage auf deterministischem Weg durch das System und lässt immer zumindest Rückschlüsse auf den Urheber zu. Anonymität ließe sich nur mit erheblichen Laufzeiterhöhungen erkaufen.

Latenz Chord garantiert zwar eine schnelle und skalierbare Suche, diese kann aber zu langsam und insbesondere spezialisierten Client/Server Systemen unterlegen sein. Da für jede Anfrage verschiedene Knoten befragt werden müssen und Chord keinerlei Optimierung hinsichtlich der Regionalität vornimmt, ist Chord für zeitkritische Systeme mitunter ungeeignet. Beispielsweise wird Chord in [CMM02] die Tauglichkeit als DNS-Server abgesprochen.

regionale Lastverteilung Chords Hashfunktion berücksichtigt keine regionalen Zusammenhänge. So kann es passieren, dass Daten denkbar ungünstige Wege zurücklegen. Dem kann mit Caching der Daten auf dem Weg begegnet werden, es aber nicht ausräumen.

3.3 Einsatzmöglichkeiten

Die Entwickler von Chord skizzieren in ihrem Grundlagenpapier [SMK+01] mögliche Einsatzzwecke von Chord, die hier kurz wiedergegeben werden sollen:

Verteilte Indexsuche In Filesharing Netzen, die auf Peer-to-Peer Techniken setzen, muss es möglich sein, schnell und sicher diejenigen Nutzer zu finden, die eine bestimmt Datei anbieten. Hierbei wird einfach der Dateiname als Schlüssel für das Hashing verwendet und als Wert eine Liste eben dieser Nutzer abgelegt. Je nach Ausdehnung kann der Download dann direkt oder über ein paralleles, blockorientiertes Chord Netzwerk geschehen oder sogar im selben Ring lediglich mit einer anderen Hashingstrategie erfolgen.

Verteiltes Rechnen Chord könnte als Koordinator für große Rechenprojekte dienen. Dabei wird der Suchraum auf den Chord Adressbereich abgebildet. Ein Rechner arbeitet an den Daten, für die er der Nachfolger ist. Kommt ein Rechner dazu, werden Ergebnisse repliziert und ein Teil des Suchraums abgegeben. Insbesondere Communtity Projekte wie etwa seti@home, in denen es eine hohe Nutzerfluktuation gibt, lassen sich mit Chord elegant und effektiv umsetzen.

Kooperatives Spiegeln Chord könnte helfen beliebte Downloads besser verfügbar zu machen. Viele Dateien haben kurz nach ihrem Erscheinen eine hohe Beliebtheit, die anschließend asymptotisch abfällt (etwa Patches, neue Distributionen, ...). Kooperieren nun mehrere Unternehmen oder Personen, so kann die in den Zwischenzeiten brachliegende Bandbreite für die aktuell beliebte Datei verwendet werden. Gleichzeitig lässt sich damit die Verfügbarkeit erhöhen.

Bereitstellung von Daten trotz einer nicht immer bestehenden Leitung Anbieter die eine schlechte oder nur zweitweilige Anbindung an das Internet haben, könnten mittels Chord kooperieren und die Dateien des jeweils anderen transparent zur jeweiligen Onlinezeit bereitstellen.

3.4 Simulationsergebnisse

Um die gemachten Aussagen zu überprüfen, haben die Entwickler für [SMK+01] in 2001 bereits umfangreiche Tests durchgeführt. Dazu kamen sowohl ein Protokollsimulator mit bis zu 10^4 Knoten, als auch ein reales auf 10 US-amerikanische Städte verteiltes Netzwerk zum Einsatz. Die damals benutzen Algorithmen wurden seitdem stark verbessert, dennoch konnten im Wesentlichen alle zugesicherten Eigenschaften bestätigt werden.

Um die erwünschte Lastverteilung zu erreichen, müssen allerdings die schon erwähnten virtuellen Knoten eingesetzt werden, da ansonsten die Wahrscheinlichkeit keine oder überproportional viele Knoten verwalten zu müssen, zu gross wird. Beim Test des gleichzeitigen Ausfalls vieler Knoten, wie es etwa bei einem Netsplit passieren könnte, war die Anzahl der fehlschlagenden Suchanfragen in etwa identisch mit der Anzahl der tatsächlich ausgefallenen Knoten und damit für das Gesamtnetz nicht gefährlich. Eine replizierende Schicht vorrausgesetzt, ist das Netz in kurzer Zeit wiederhergestellt. Ohne eine solche lässt sich zumindest zu quasi jeder Zeit ein für eine beliebige ID verantwortlicher Knoten ermitteln.

4 Ausgewählte Algorithmen

Im vorletzten Kapitel wurde die grundlegende Funktionsweise von Chord erläutert. Im folgenden soll auf einige problematische Situationen genauer eingegangen werden. Nach Vorstellung der Problematik wird Chords Lösungsansatz kurz umrissen.

4.1 Einfügen von Knoten

Wie oben schon erwähnt, müssen nach dem Einfügen eines Knotens die Fingertabellen der anderen Knoten aktualisiert werden. Es ist aber nicht nötig sämtliche Knoten des Netzes zu überprüfen. Stattdessen reicht es für alle i: $0 < i < m$ zu überprüfen, ob es einen Knoten gibt, der mindestens im Abstand 2^{i-1} vor dem aktuellen Knoten liegt, dessen i.ter Finger größer ist, als der aktuelle Knoten. Also dann, wenn es beim Aufbau des Netzes zwischen dem Startwert eines früheren Knotens und der ID des aktuellen Knotens keine weiteren Knoten gab. Laut [SMK+01] gibt läßt sich zeigen, dass die Kosten für das Aktualisieren nach einem Einfügevorgang auf maximal $O(log^2 n)$ belaufen.

Um seine eigene Fingertabelle zu füllen, hat ein Knoten mehrere Möglichkeiten. Er könnte einfach für jeden Startwert in der Tabelle, das Netz nach dem zuständigen Nachfolger fragen. Das ist allerdings relativ teuer, hat doch der nächste Nachbar bereits eine ähnliche Fingertabelle aufgebaut. Ein Knoten fragt also zunächst seine beiden Nachbarn nach ihrer Fingertabelle und versucht diese soweit wie möglich zu übernehmen.

Definition 3 *Insgesamt belaufen sich die Kosten für das Einfügen von Knoten nach [DBK+01] auf* $O(log^2 n)$

4.2 Stabilisierung

Ohne weitere Maßnahmen, könnte der Ausfall von Knoten leicht das Netz belasten. Zwar sorgt die schon angesprochene Stabilisierungsfunktion, die auch zum Einfügen von Knoten aktiv wird dafür, dass zumindest die Nachfolger und Vorgänger aller Knoten in gewisser Zeit korrigiert werden. Es ist aber dennoch möglich das Netz zu zerstören oder empfindlich zu verlangsamen. Um schnell auf das Ausfallen eines Knotens reagieren zu können, speichert jeder Knoten, ausser seinem Nachfolger, noch eine ganze Liste weiterer Nachfolger. Fällt der erste Nachfolger aus, so ist der neue Nachfolger bereits bekannt und kann kontaktiert werden. Das wird allerdings problematisch, wenn einzelne Knoten vorrübergehend nicht erreichbar sind. Durch ungünstige Konstellationen kann es passieren, dass eine Art Ring im Ring aufgebaut wird. Auch Bugs in der Implementierung eines Clients oder unvorhergesehene Situationen sind bei der Größe eines Chord Netzes schon rein statistisch durchaus wahrscheinlich.

Zur Lösung solcher Mehrfachkreise versucht ein Knoten k sich regelmäßig selbst zu finden. Dabei nutzt er jedoch nur die Nachfolgerzeiger und ignoriert die Finger der anderen Knoten. Gibt es nun einen Ring im Ring, findet Knoten k irgendwann einen Knoten x für den gilt: $Vorgänger(x) < k < x$. Durch die Korrektur der Zeiger kann ein Ring auf diese Weise langsam stabilisiert werden.

Es gibt eine Vielzahl von verschiedenen Stabilisierungsalgorithmen mit ebenso unterschiedlichen Garantien. Leider gibt es bislang keinen Algorithmus der zugleich ein defektes Netz repariert und instand hält. Stattdessen muss auf den Instandsetzungsalgorithmus immer nur dann zugegriffen werden, wenn sich das Netz erkennbar in einem defekten Zustand befindet. In dieser Zeit können keine weiteren Knoten dem Netzwerk beitreten, bis das Netz wieder stabil ist. Praktische Beobachtungen der Entwickler von Chord lassen aber vermuten, dass das Netz in seinem jetzigen Stadium bereits sehr stabil läuft. Dennoch steht der Beweis der Korrektheit immer noch aus.

5 Zusammenfassung

Chord ist ein extrem flexibles und vielseitiges, vor allem aber skalierbares Netzwerk. Es garantiert den Erfolg von Suchanfragen sowie maximale Laufzeiten beim Einfügen und Löschen von Knoten mit jeweils $O(log^2 n)$, die Suche nach Informationen ist mit $O(log n)$ sogar noch schneller. Dabei ist Chord resistent gegen eine Vielzahl von Fehlerfällen und bietet auch nach einem massiven Netzausfall noch garantiert korrekte Antworten.

5.1 Projekte

Neben vielen Projekten, die mehr zu Forschungs und Demonstrationszwecken genutzt werden, gibt es auch einige Projekte die in der Praxis genutzt werden. Die verbreitetsten sind wahrscheinlich:

CFS CFS [Dab] ist ein kooperatives Dateisystem, dass im Wesentlichen auf Chord basiert. Es bietet dem Betriebsystem eine normale blockorientierte Schnittstelle auf Dateien. Ein Dateisystem wird durch einen Root-Block repräsentiert. Der Root-Block ist mit einem privaten Schlüssel des Autors signiert und enthält neben den Dateinamen die zugehörigen Chord IDs der einzelnen Blöcke. Das System erlaubt nur reine Lesezugriffe auf vorhandene Dateisysteme. Die Geschwindigkeit wird durch blockweises Cachen von häufig verwendeten Dateien gesteigert. Ebenso gibt es keine explizite Löschoperation, kommt es zu keinen Anfragen an das Dateisystem mehr. verschwindet es langsam aus dem Netz.

UsenetDHT UsenetDHT [SDR04] ist ein Newsserver, der zum Ziel hat, Bandbreite durch gemeinsamen Speicher zu sparen. Er befindet sich noch in der Testphase, allerdings ist er nicht für einen vollwertigen Newsserver Ersatz geeignet, sondern kann bisher lediglich Server ersetzen die weit unten in der Server-Hierachie stehen und nur Endabnehmer beliefern.

OverCite OverCite [SCL+05] ist eine verteilte Version der Wissenschafts- und Zitationsdatenbank CiteSeer. Laut den Autoren ermöglicht OverCite eine vielfach höhere Performance als CiteSeer.

5.2 Abschlussbemerkung

Da Chord von seinen Entwicklern immer noch nicht in finaler Version vorliegt, kann nur ein Ausschnitt der Funktionalität und Algorithmen vorgestellt werden. Das originale Papier ist bereits 6 Jahre alt und hat zu einer großen Vielfalt von Varianten und Analysen geführt. Dennoch benennen die Autoren zahlreiche offenen Probleme, deren Lösung nocht aussteht.

Der von Chord verfolgte Ansatz, einen leichtgewichtigen Lokalisierungsservice anzubieten, ist sehr elegant und ermöglicht eine Vielzahl von Einsatzzwecken. Allerdings ist durch eben diese Flexibilität auch einiger Aufwand in der Implementierung der Kontroll- und Replikationsschichten nötig. Chord steht unter einer MIT-Lizenz und darf daher auch in kommerziellen Produkten kostenfrei verwendet werden.

Eine einfache Suche nach auf Chord basierenden Softwareprojekten liefert ein erstaunlich mageres Ergebnis. In [ABS07] behaupten die Autoren *"Chord is the most researched and best understood mechanism"* um sich anschließend ausschließlich dem Kadmelia Protokoll zu widmen. Chord wird als Ideengeber und Grundbaustein für neue Protokolle, seltener jedoch selbst eingesetzt. Ein Grund mag die grundlegende Maxime der Skalierbarkeit sein, die sich zwar für Netzwerke quasi beliebiger Ausdehnung eignet aber dafür bei kleinen Netzen, auf sie spezialisierten Lösungen, unterlegen zeigt.

Inwieweit sich Chord gegenüber seinen Konkurenten durchsetzt, muss abgewartet werden. So steht es aktuell zur Debatte Chord für einen neuen Standard für Audio- und

Videokonferenzen einzusetzen. Es ist aber auch gut möglich, dass Chord von seinen eigenen Weiterentwicklungen überholt wird und damit lediglich die Idee nicht aber die ursprüngliche Umsetzung erhalten bleibt.

Literatur

[ABS07] Phuoc Tran-Gia Andreas Binzenhfer and Holger Schnabel. Methods for performance improvement of kademlia-based overlay networks (methoden zur leistungsverbesserung eines kademlia-basierten overlay-netzes), 2007. Page start: 280.

[CMM02] R. Cox, A. Muthitacharoen, and R. Morris. Serving dns using a peer-to-peer lookup service, 2002.

[Dab] Frank Dabek. A cooperative file system.

[DBK+01] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with Chord, a distributed lookup service. pages 81–86, 2001.

[HT03] Aaron Harwood and Egemen Tanin. Hashing spatial content over peer-to-peer networks. In *Australian Telecommunications, Networks and Applications Conference (CD-ROM)*. ATNAC, 2003. 5 pages.

[LNBK02] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Observations on the dynamic evolution of peer-to-peer networks, 2002.

[SCL+05] Jeremy Stribling, Isaac G. Councill, Jinyang Li, M. Frans Kaashoek, David R. Karger, Robert Morris, and Scott Shenker. Overcite: A cooperative digital research library. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS05)*, Ithaca, NY, February 2005.

[SDR04] E. SIT, F. DABEK, and J. ROBERTSON. Usenetdht: A low overhead usenet server, 2004.

[SM02] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables, 2002.

[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.