

Universität des Saarlandes, Fachbereich Informatik

# **Analysis of the Evolution of Peer-to-Peer Systems**

David Liben-Nowell, Hari Balakrishnan, David Karger

Jan Conrad

Ausarbeitung

im Rahmen des Seminars

"Peer-to-peer Systems"

Wintersemester 2003/04

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>S. 3</b>
<b>2. Background zu Chord</b>	<b>S. 5</b>
<b>2.1. Grundlagen</b>	<b>S. 5</b>
<b>2.2. Successoren und finger-Pointer</b>	<b>S. 5</b>
<b>3. Analyse zu Chord</b>	<b>S. 7</b>
<b>3.1. Mögliche Fehler</b>	<b>S. 7</b>
<b>3.2. Idealer Zustand</b>	<b>S. 8</b>
<b>3.2.1. Definition</b>	<b>S. 8</b>
<b>3.3. Ausfallmodell</b>	<b>S. 9</b>
<b>3.3.1. Definition</b>	<b>S. 9</b>
<b>3.3.2. Lemma</b>	<b>S. 10</b>
<b>3.4. "Join"-Modell</b>	<b>S. 10</b>
<b>3.4.1. Definition</b>	<b>S. 11</b>
<b>3.4.2. Lemma</b>	<b>S. 12</b>
<b>3.5. Dynamisches Modell</b>	<b>S. 13</b>
<b>3.5.1. Definition</b>	<b>S. 13</b>
<b>3.5.2. Theorem</b>	<b>S. 14</b>
<b>3.5.3. Beweis</b>	<b>S. 14</b>
<b>4. Zusammenfassung und Ausblick</b>	<b>S. 16</b>

# 1. Einleitung

Diese Ausarbeitung entstand aus dem Seminar "Peer-to-Peer-Systems" im Fachbereich Informatik an der Universität des Saarlandes im Wintersemester 2003/2004. Sie behandelt den, für den PODC 2002 von David Liben-Nowell, Hari Balakrishnan und David Karger erstellten Text "Analysis of the Evolution of Peer-to-Peer Systems"[1].

Es wird hierin eine theoretische Analyse von Peer-to-peer Netzwerken unter besonderer Berücksichtigung von gleichzeitiges Verlassen bzw. Hinzukommen von mehreren Knoten angestellt.

## **Definition Peer-to-peer Netzwerke:**

*Als Peer-to-peer (P2P) Systeme bezeichnet man verteilte Systeme ohne zentrale Kontrollstelle, oder hierarchischen Aufbau. Jeder Knoten des Netzwerks besitzt die gleiche Software und gleichwertige Funktionalität.*

Aufgrund der Tatsache, dass P2P-Netzwerke Systeme sind, die sich ständig verändern ist es sehr kompliziert Knoten zu adressieren, sodass sie später wieder leicht zu finden sind. Die Analysen von Protokollen wie CAN [6], Chord [7], oder Pastry [2] gehen hierbei von einem idealen "Overlay"<sup>1</sup> aus, indem die Knoten so angeordnet sind, dass sie in schnellstmöglicher Zeit gefunden werden können. Jeder Knoten ist dann in der Lage nach "joins", oder "departures" von anderen Knoten das Netzwerk wieder in einen optimalen Zustand zurück zu versetzen, in dem Suchanfragen effizient gehandhabt werden können. Somit verfügen solche Netzwerke über eine gewisse Fehlertoleranz [4,5,7].

Bei den dazugehörigen Analysen wird jedoch von sequentiellen "joins" und "departures" im Netzwerk ausgegangen. Gleichzeitiges Hinzukommen oder Ausfallen von Knoten ist in realistischen P2P-Netzwerken sehr wahrscheinlich. Dies kann jedoch dazu führen, dass das Netzwerk nicht mehr im optimalen Zustand ist. So ist nicht mehr gewährleistet, dass Suchanfragen effizient durchgeführt werden oder überhaupt das richtige Ergebnis liefern.

Um mit diesem Problem umgehen zu können, muss eine Art Wartungsprotokoll den Index des Netzwerkes ständig reparieren.

Dieses Wartungsprotokoll muss ständig laufen und somit müssen wir in unseren Analysen von P2P-Netzwerken davon ausgehen, dass das Netzwerk fast nie einen idealen Status besitzt.

---

<sup>1</sup> Mit "Overlay" ist der Index des Netzwerkes gemeint. Ein idealer Index wird im folgenden auch mit idealem Status des Netzwerkes bezeichnet.

Deshalb ist es sinnvoll, das Wartungsprotokoll selbst zu analysieren, um etwas über die Performance eines sich ständig verändernden P2P-Netzwerkes aussagen zu können. Doch wie kann man ein geeignetes Maß für die Evaluierung des Wartungsprotokolls finden? Die Laufzeit kommt als Maß nicht in Frage, da das Wartungsprotokoll genau so lange wie das gesamte Netzwerk läuft. Auch die Bandbreite des gesamten Netzwerkes ist unendlich. Stattdessen erweist sich die Netzwerkbandbreite, die ein Knoten zum Durchführen der Wartung benötigt, da diese Ressource für die eigentliche Aufgabe des Netzwerkes nicht mehr zur Verfügung steht. Die Bandbreite mit der jeder einzelne Knoten mit dem Netzwerk verbunden ist (und nicht Speicherplatz oder Rechenzeit), ist momentan die am meisten limitierte Ressource in P2P-Netzwerken.

Im folgenden analysieren wir das Wartungsprotokoll, das von Chord verwendet wird, die sogenannte "Idealisation". Wir analysieren einen Chord-Ring, wenn es zu Ausfällen, bzw. zum Dazukommen von neuen Knoten unter bestimmten Bedingungen kommt und betrachten die Korrektheit und die Laufzeit von Suchanfragen. Um den Aufwand der Wartung messen zu können, werden wir das Kommen und Gehen von Knoten in Bezug zu der Anzahl an Ausführungen der Idealisation setzen die nötig sind, damit Anfragen an das Netzwerk wieder korrekte Ergebnisse liefern.

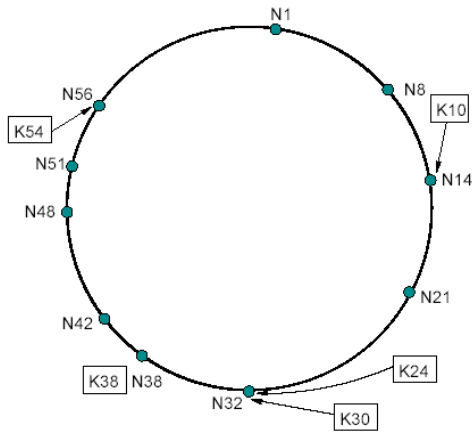


Abbildung 1

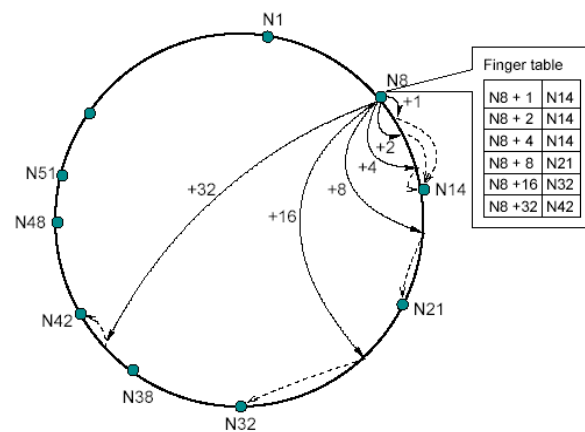


Abbildung 2

## 2. Background zu Chord

### 2.1 Grundlagen

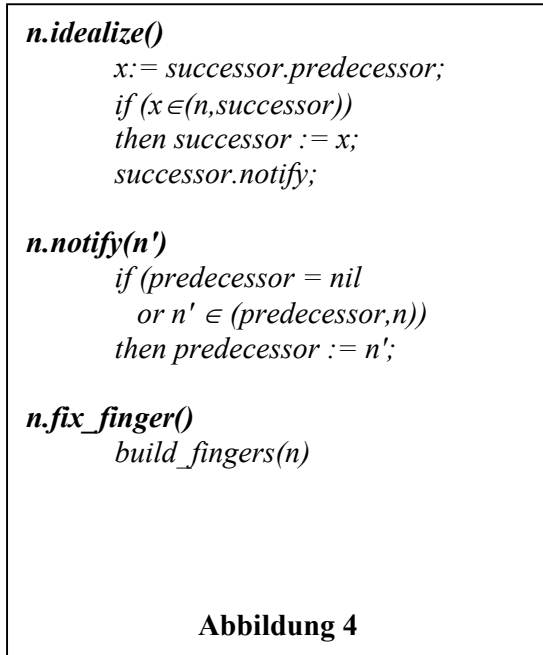
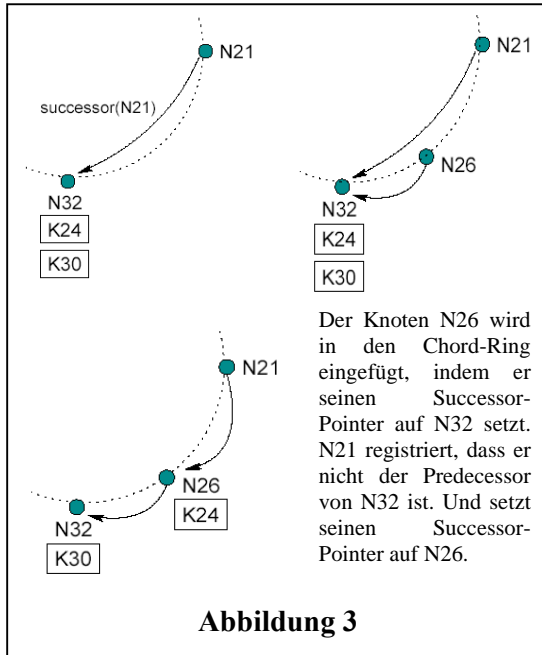
Das Protokoll von Chord beherrscht nur eine einzige Operation: Zu einem gegebenen Schlüssel den dazugehörigen Knoten finden. In Chord ist dies mittels consistent hashing<sup>2</sup> implementiert. Die IP-Adressen der Knoten und der Schlüssel werden als m-bit Integer durch die Hash-Funktion SHA-1 im selben Hashtable gespeichert. Jeder Schlüssel ist zu mit seinem Successor-Knoten<sup>3</sup> verbunden. Dies ist der Knoten mit der nächst höheren IP. Dadurch, dass Consistent Hashing die Knoten im IP-Adressraum gleichverteilt, kann man das so entstehende Netzwerk grafisch gut als Ring darstellen [Abbildung 1].

### 2.2 Successoren und Finger-Pointer

Jeder Knoten speichert seinen "Successor-Knoten" - der Knoten mit der nächst höheren IP. Somit kann der Successor eines Schlüssels durch auffinden eines Knotens  $n$  mit  $n < k < n.successor$  gefunden werden. Um dies effizienter zu handhaben, verwaltet jeder Knoten eine finger-Liste [Abbildung 2]. Für alle  $i$  in  $\{1, 2, \dots, m\}$  gibt die Funktion  $n.finger(i)$  den Knoten

<sup>2</sup> Frühere Arbeiten [4,5] haben gezeigt, dass consistent hashing eine zufällige und unabhängige Verteilung gewährleistet. Diese Eigenschaft wird später noch sehr wichtig sein.

<sup>3</sup> Der Begriff Successor bezeichnet in diesem Zusammenhang den Knoten, der die nächst höhere IP-Adresse besitzt. Predecessor ist analog der Knoten, der im Netzwerk die nächst niedrigere IP erhalten hat.



zurück, dessen IP die nächst größere zu  $n + 2^{i-1}$  ist. Mit einer finger-Liste der Länge  $\log(N)$  erreichen wir von jedem Knoten aus jeden Key in  $\log(N)$  Schritten. Wenn ein neuer Knoten  $n$  hinzukommt muss er an mit seiner IP an die richtige Stelle im Chord-Ring eingefügt werden.  $N$ .successor muss auf den im Uhrzeigersinn nächsten Knoten auf dem Ring zeigen, und der Zeiger des Predecessors von  $n$  muss seinen Successor-Pointer auf  $n$  setzen. Außerdem müssen die finger-Listen aller Knoten durch die Funktion `fix_finger` [Abbildung 3] aktualisiert werden. Da wir ja in dieser Analyse davon ausgehen, dass Knoten zufällig und unabhängig zum System hinzukommen können, muss in periodischen Abständen eine `idealize`-Prozedur [Abbildung 4] ausgeführt werden, damit die gewünschten Eigenschaften des Netzwerks erhalten bleiben. Dafür benötigen wir einen Predecessor-Pointer, der den nächsten Vorgänger des Knotens speichert. Außerdem muss gewährleistet sein, dass das System auch weiterhin funktioniert, wenn der Successor eines Knotens ausfällt. Deshalb verwaltet jeder Knoten eine Successor-Liste mit den nächsten  $k$  Successoren, sodass er diese abarbeiten kann, falls Knoten ausfallen.

### 3. Analyse von Chord

#### 3.1 Mögliche Fehler

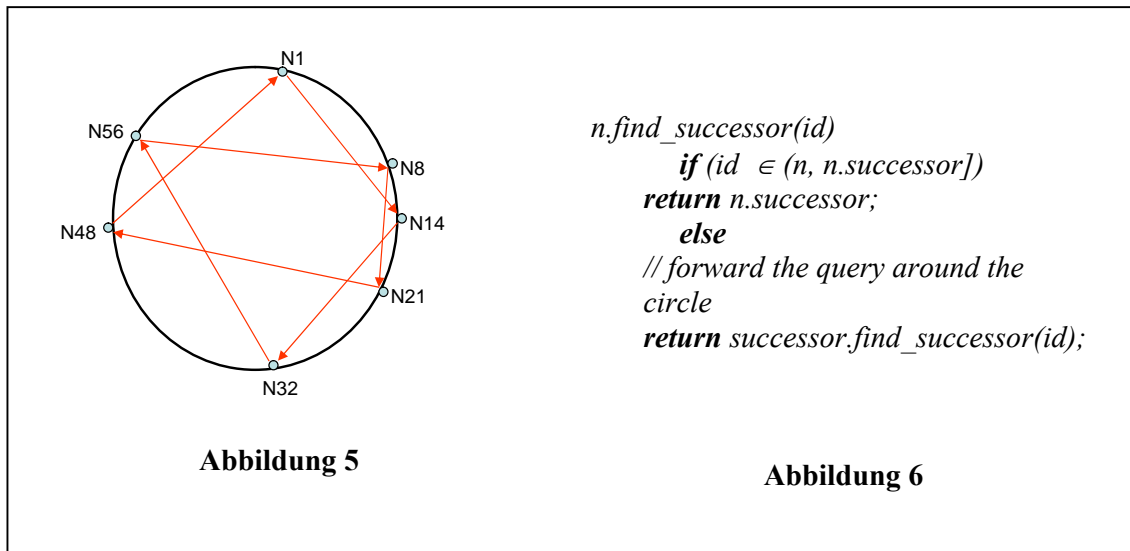
Die Funktion `idealize` [Abbildung 4] versucht, das Netzwerk in einen "idealen Status" zu versetzen um die Funktion des Systems mit einer gewissen Fehlertoleranz zu gewährleisten. Dies ist allerdings nicht unter allen Umständen möglich. Uns interessiert, inwieweit Chord noch korrekte Ergebnisse liefern kann, obwohl es sich nicht mehr im optimalen Zustand befindet. Nun stellt sich die Frage, welche Fehler auftreten können, sodass das Netzwerk nicht mehr korrekt funktioniert?

Wenn sich beispielsweise gleichzeitig mehrere `Successor-Pointer` ändern, so kann es passieren, dass sich das Netzwerk in zwei oder mehr Teile aufspaltet und somit bestimmte Daten für bestimmte Knoten unzugänglich werden. Ein noch subtileres Problem stellen Schleifen dar. Schleifen (oder `Loops`) sind wie folgt definiert:

**Definition:** *Ein Netzwerk heißt*

- **Schwach ideal**, wenn für alle Knoten  $u$  im Netzwerk gilt:  
 $(u.successor).predecessor = u$
- **Stark ideal**, wenn es schwach ideal ist und wenn es für alle Knoten  $u$  im Netzwerk keinen Knoten  $v$  gibt, sodass  $u < v < u.successor$  gilt
- **Loopy**, wenn es schwach ideal aber nicht stark ideal ist.

Die vorher vorgestellten Protokolle für Chord sind schwach, aber nicht stark ideal. Wie man bei Abbildung 5 erkennt, entspricht dieser Chord-Ring den Anforderungen der Funktion `idealize` [Abbildung 4]. Es ist aber möglich, dass die Funktion `find_successor` [Abbildung 6] für eine Query zwei verschiedene Ergebnisse liefert. Beginnt `find_successor` für z.B. Key 45 und beginnt bei Knoten N8, so liefert die Funktion N48 als korrektes Ergebnis. Beginnt sie jedoch mit N14, so gibt sie N56 zurück. Da jedoch ein Key 45 mit N48 verknüpft wäre, würde N14 kein Ergebnis finden.



### 3.2. Der ideale Status

In diesem Abschnitt definieren wir, wie der bereits erwähnte ideale Status eines P2P-Netzwerkes aussehen kann. In anderen Arbeiten [7] hat sich gezeigt, dass der Chord-Ring durch die folgenden Punkte eine Reihe guter Eigenschaften erhält: Die Funktion `find_successor(q)` gibt in Zeit  $O(\log N)$  den richtigen Successor von  $q$  zurück, selbst wenn alle Knoten mit einer konstanten Wahrscheinlichkeit  $p < 1$  ausfallen. Dies ist eine beachtliche Fehlertoleranz und unterstützt schnelles und effizientes Suchen. Dabei gehen wir von einer Successor-Liste der Länge  $c \log(N)$  aus, wobei  $c$  eine Konstante darstellt.

**3.2.1 Definition:** Ein Chord-Netzwerk ist im idealen Zustand wenn:

1. **[Connectivity]:** Man kann mittels der Successor-list und der Finger-tables einen Pfad von jedem zu jedem Knoten finden
2. **[Randomness]:** Alle Knoten sind zufällig und gleichmäßig um den Chord-Ring verteilt
3. **[Cycle Sufficiency]:** Alle Knoten sind auf dem Chord-Ring<sup>4</sup>
4. **[Non-loopiness]:** Das Netzwerk besitzt keine Schleife
5. **[Successor list validity]:** Jede `u.successor_list` enthält die ersten  $c \log(n)$  Knoten, die  $u$  folgen.
6. **[Finger validity]:** Für alle Knoten  $u$  und alle  $i$  in  $1 \dots m$  ist der Successor von  $u + 2^{i-1}$  als `u.finger(i)` gespeichert

<sup>4</sup> Mit dem Ausdruck "auf dem Chord-Ring" ist die korrekte Integration des Knotens mit Successor-, Predecessor-Pointer, Successor-Liste, sowie Finger-Liste gemeint.

Da jedoch das Netzwerk durch "joins" und "leaves" von Knoten nicht im idealen Zustand bleiben muss, zeigen wir nun, dass leicht abgeschwächte Eigenschaften trotzdem einen "fast idealen" Zustand beibehalten.

### 3.3. Ein Ausfallmodell <sup>5</sup>

Wir betrachten ein Netzwerk mit  $N$  Knoten und nehmen an, dass  $N/2$  Knoten des Netzwerkes (zufällig verteilt) ausfallen. Für dieses Netzwerk wollen wir nun die gleichen guten Eigenschaften wie für ein Netzwerk im idealen Status erhalten. D.h., dass wir wenn wir ein Netzwerk im Ring-mit-Ausfällen-Status haben, er also die im folgenden definierten Eigenschaften erfüllt, erhalten wir mit geringen Wartungsaufwand wieder ein ideales Netzwerk.

**3.3.1 Definition:** Ein Chord-Netzwerk ist im **Ring-mit-Ausfällen-Status**, wenn:

1.,3.,4. aus Definition 3.2.1 bzw. 2.,5., wie folgt:

2. **[Randomness]:**

- a) Wie in der Definition des idealen Status
- b) Die Ausfälle von Knoten sind gleichverteilt

5. **[Successor list validity]:** Für alle Knoten  $u$  sei  $L(u)$  die Menge der noch lebenden Einträge in der Successor-Liste

- a) Alle  $|L(u)| \leq (c/3) \log N$
- b) Alle  $L(u)$  enthalten genau die ersten  $|L(u)|$  lebenden Knoten, die  $u$  folgen

6. **[Finger validity]:** Für alle Knoten  $u$  und Indices  $i$  gilt: Wenn  $u.finger[i]$  lebt, dann ist es der nächste lebendige Knoten der auf  $u + 2^{i-1}$  folgt

Da wir in unserem Modell davon ausgehen, dass die Ausfälle gleichverteilt sind, ist die Wahrscheinlichkeit der Gleichverteilung der Ausfälle in 2.b) gegeben.

Wenn nun die Hälfte aller Knoten ausfällt, so ist zu erwarten, dass auch die Hälfte der Knoten der Successor-Liste ausfällt. Somit bleiben mit großer Wahrscheinlichkeit<sup>6</sup>  $1/3$  der Knoten in der Successor-Liste am Leben. Dies muss auch für 5.a) erfüllt sein. Es ist offensichtlich, dass

---

<sup>5</sup> Da wir immer damit rechnen müssen, dass Knoten ausfallen können, ohne ein Logout-Protokoll durchzuführen, führen wir erst gar kein solches ein, sondern gehen immer davon aus, dass Knoten die das System verlassen einfach wegfallen.

<sup>6</sup> Vgl. Chernoff-Schranke

Bedingung 5.b) gelten muss, damit die Successor-Liste überhaupt korrekt produziert werden kann. Um diese nun für einen Knoten  $u$  zu bilden, betrachten wir die Successor-Liste des Successors von  $u$  usw. bis die Liste der  $c \log N$  Knoten für  $u$  zusammengetragen ist. Analog gilt dies für die finger-Liste und Punkt 6.

Mit dem folgenden Lemma wird deutlich, dass ein Netzwerk im Ring-mit-Ausfällen-Status in diesem Zustand bleibt, wenn nur genügend Ausführungen von idealize durchgeführt werden können.

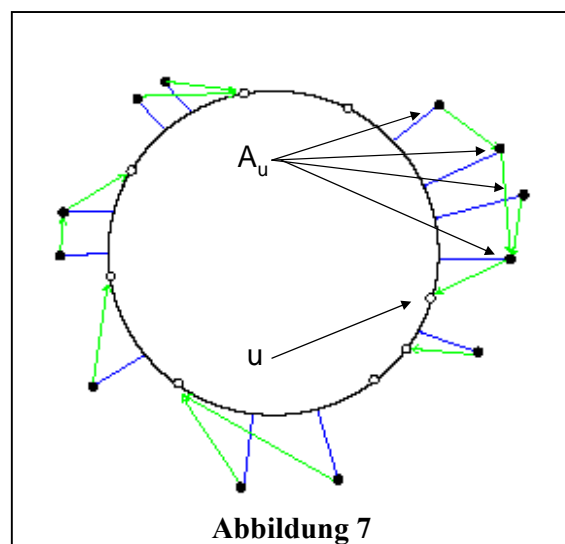
**3.3.2 Lemma:** *Man betrachte ein Chord-Netzwerk mit  $N$  Knoten, das den Ring-mit-Ausfällen-Status besitzt. Man nehme weiter an, dass  $N/2$  Knoten ausfallen und währenddessen mindestens  $\Omega(\log N)$  mal die Idealisation aufgerufen worden ist. Dann gilt mit hoher Wahrscheinlichkeit:*

1. *Zu jedem Zeitpunkt kann während diesem Prozess  $\text{find\_successor}(q)$  einen lebenden Successor von  $q$  in Zeit  $O(\log N)$  zurückliefern.*
2. *Das resultierende Netzwerk besitzt wieder den Ring-mit-Ausfällen-Status.*

Das Netzwerk bleibt mit hoher Wahrscheinlichkeit verbunden, da immer  $O(\log N)$  Knoten in der Successor-Liste vorhanden sind. Da jeder finger in der finger-Liste mit konstanter Wahrscheinlichkeit noch vorhanden ist, halbiert jedes "finger" die Entfernung zum Zielknoten mit konstanter Wahrscheinlichkeit. So bleibt die Anzahl der nötigen Sprünge bei Query-Prozess mit hoher Wahrscheinlichkeit in  $O(\log N)$ .

### 3.4 "Join"-Modell

Jetzt betrachten wir ein Modell bei dem Knoten zum Netzwerk hinzu kommen können, jedoch keine Knoten wegfallen. Hierbei müssen wir wieder einige der Bedingungen entspannen. Hierbei kann es jedoch dazu kommen, dass neu hinzugekommene Knoten eventuell noch nicht auf dem Chord-Ring integriert. Sie besitzen zwar bereits eine IP und kennen Ihren Successor, dieser kennt jedoch noch nicht den



richtigen Predecessor. Wenn nun weitere Knoten diesen nicht vollständig integrierten Knoten als Successor erkennen kann es zu Anhängen in Baumform kommen. Mit  $A_u$  bezeichnen wir den Anhang des Knotens  $u$ , den Baum also, der  $u$  als Wurzel besitzt [Abbildung 7].

**3.4.1. Definition:** Ein Chord-Netzwerk mit  $N$  Knoten ist im **Ring-mit-Anhängen-Status**, wenn:

1.u.2. wie in der Definition des 3.2.1, bzw 3.,4.,5. und 6 wie folgt

3. **[Cycle Sufficiency]:**

- a) Mindestens  $N/2$  der Knoten sind gleich und unabhängig auf dem Chord-Ring verteilt (also nicht in Anhängen)
- b) Für alle Knoten  $u$  gilt:  $|A_u| = O(\log N)$ .

4. **[Non-loopiness]:**

- a) Das Netzwerk besitzt keine Schleifen
- b) Für jeden Knoten  $v$  im Anhang  $A_u$  sind die IP-Adressen der Knoten von  $v$  zu  $u$  ansteigend.

5. **[Successor validity]:** Da keine Ausfälle von Knoten betrachtet werden kann man diesen Punkt auf das Betrachten des Successors vereinfachen. Für jeden Knoten  $v$  gilt dann:

- a) Falls  $v$  auf dem Ring ist, dann ist  $v$ .successor der erste Knoten auf dem Ring, der auf  $v$  folgt
- b) Falls  $v$  im Anhang  $A_u$  eines Knotens ist, so ist  $u$  der erste Knoten, der auf  $v$  folgt und auf dem Ring ist

6. **[Finger validity]:** Es existiert eine Teilmenge  $S$  aus  $N$  sodass alle Knoten aus  $S$  gleichförmig und unabhängig auf dem Chord-Ring verteilt sind und  $|S| = N/2$ . Für jedes  $u$  aus  $S$  und jedes  $i$  aus  $1..m$  muss gelten: Es gibt kein Element aus  $S$ , dass zwischen  $u + 2^{i-1}$  und  $\text{finger}(i)$  fällt.

Bedingung 3 aus Definition 3.4.1 garantiert, dass die Knoten im Netzwerk gleichverteilt sind. Die meisten Knoten sind auf dem Chord-Ring und höchstens ein konstanter Teil ist als Anhang verteilt. Dass es zu keinen Schleifen in den Anhängen kommen darf, stellt Bedingung 4.b) sicher. Dieses Kriterium ist erforderlich, damit `find_successor()` auch über den Anhängen korrekt funktioniert. Ist diese Bedingung nicht erfüllt, kann es passieren, dass `find_successor()` in der Schleife eines Anhangs stecken bleibt und einen falschen Knoten zurückgibt. Bedingung 5 garantiert die richtige Anordnung der Anhänge. In Bedingung 6 werden hinreichend genaue finger-Pointer gefordert.

Wenn man nun von einem Netzwerk mit  $N$  Knoten ausgeht und  $N$  zusätzliche Knoten in das Netzwerk eintreten, so ist das resultierende Netzwerk im Ring-mit-Anhängen-Status. Dies kann einfach gezeigt werden: Da die Funktion `find_successor` im idealen Zustand korrekt ist, ist klar, dass ein neu hinzukommender Knoten  $u$  seinen Successor-Pointer auf seinen Successor auf dem Ring oder einen anderen Knoten im Anhang korrekt setzt. Dies impliziert die Bedingungen 1,2,4,5. Bedingung 3.a) ist durch die  $N$  bereits auf dem Ring befindlichen Knoten erfüllt und 3.b) ist durch die Gleichverteilung der Knoten mit hoher Wahrscheinlichkeit erfüllt. Die Teilmenge  $S$  in Bedingung 6 seien alle  $N$  bereits existierende Knoten. Somit ist diese Bedingung auch erfüllt, da die `finger-tables` für diese  $N$  Knoten wegen des idealen Zustandes korrekt gebildet waren, und somit auch bezüglich  $S$  korrekt sind.

Nun betrachten wir ein Netzwerk im Ring-mit-Anhängen-Status und nehmen an, dass über  $\Omega(\log^2 N)$  Ausführungen der Idealisation keine neuen Knoten zum System hinzukommen. In jeder Runde, in der das Wartungsprotokoll ausgeführt wird, kommt ein Knoten aus einem Anhang zum Ring hinzu, solange bis ein Ringknoten  $v_r$  und mindestens ein Knoten  $v_a$  aus den Anhängen  $u$  für ihren Vorgänger halten. Das Idealisation-Protokoll wird daraufhin den Successor-Pointer des weiter entfernten Knotens richten. Dies ist  $v_r$ . Der Knoten  $v_a$  wird dann in den Ring aufgenommen. Auf diese Weise sind nach  $O(\log N)$  Runden alle Anhänge im Ring integriert. Nun müssen noch für alle Knoten die `finger-tables` erstellt werden.

Den zunächst eher intuitiven Ansatz kann man wie schon im vorausgegangenen Kapitel wieder zu einem Lemma zusammen fassen:

**3.4.2. Lemma:** *Man betrachte ein Chord-Netzwerk mit  $N$  Knoten im Ring-mit-Anhängen-Status. Man nehme weiter an, dass  $N$  Knoten zum System hinzukommen und währenddessen mindestens  $\Omega(\log^2 N)$  mal die Idealisation aufgerufen worden ist. Dann gilt mit hoher Wahrscheinlichkeit:*

1. *Zu jedem Zeitpunkt kann während diesem Prozess `find_successor(q)` den Successor  $s$  auf dem Ring von  $q$  oder einen Knoten  $u$  im Anhang  $A_s$  mit  $q < u < s$  in Zeit  $O(\log N)$  zurückliefern.*
2. *Das resultierende Netzwerk besitzt wieder den Ring-mit-Anhängen-Status.*

### 3.5. Dynamisches Modell

Schließlich vereinigen wir die beiden vorausgegangenen Ansätze um ein Modell mit "joins" und "leaves" zu erhalten. Man kann diese beinahe direkt zusammenstellen und muss nur einige feine Änderungen vornehmen.

**3.5.1.Definition:** Ein Chord-Netzwerk mit  $N$  Knoten ist im **Ring-mit-Ausfällen-und-Anhängen-Status**, wenn für ein konstantes  $D$  gilt:

1. **[Connectivity]:** Man kann mittels der Successor-list und der Finger-tables einen Pfad von jedem zu jedem Knoten finden
2. **[Randomness]:**
  - a) Alle Knoten sind zufällig und gleichmäßig um den Chord-Ring verteilt
  - b) Die Ausfälle von Knoten sind gleichverteilt
3. **[Cycle Sufficiency]:**
  - a) Mindestens eine  $N/3$  der Knoten auf dem Ring ist gleich und unabhängig auf dem Ring verteilt
  - b) Für alle aufeinander folgende Knoten auf dem Ring  $u_1, \dots, u_{\log N}$  gilt:
$$\sum_{i \in 1, \dots, \log N} |A_{u_i}| = O(\log N)$$
4. **[Non-loopiness]:**
  - a) Das Netzwerk ist Stark-ideal
  - b) Für jeden Knoten  $v$  im Anhang  $A_u$  sind die IP-Adressen der Knoten von  $v$  zu  $u$  ansteigend.
5. **[Successor list validity]:** Für alle Knoten  $u$  sei  $L(u)$  die Menge der noch lebendigen Einträge in der Successor-Liste
  - a) Alle  $|L(u)| \leq (c/3) \log N$
  - b) Wenn  $v$  auf dem Ring ist, so ist  $v$ .successor der erste lebendige Knoten der auf  $v$  folgt
  - c) Wenn  $v$  im Anhang  $A_u$  ist, dann ist  $u$  der erste lebendige Knoten der auf  $v$  folgt
  - d) Wenn die successor-Liste von  $u$ .successor einen lebendigen Knoten  $v$  überspringt, so ist  $v$  nicht in  $u$ .successor\_list
  - e) Keine Successor-Liste enthält Knoten, die schon länger als  $D \log^2 N$  Runden tot sind
  - f) Keine Successor-Liste überspringt lebendige Knoten, die schon länger als  $D \log^2 N$  Runden im System sind
6. **[Finger validity]:**

5. Es existiert eine Teilmenge  $S$  aus  $N$  sodass alle Knoten aus  $S$  gleichförmig und unabhängig auf dem Chord-Ring verteilt sind und  $|S| = N/2$ . Für jedes  $u$  aus  $S$  und jedes  $i$  aus  $1..m$  muss gelten: Es gibt kein Element aus  $S$ , das zwischen  $u + 2^{i-1}$  und  $\text{finger}(i)$  fällt.
6. Für alle  $i$  aus  $1..m$  gilt: Wenn  $u.\text{finger}(i)$  lebendig ist, dann ist er mindestens genau so nah an  $u + 2^{i-1}$  wie der erste lebendige Knoten aus  $S$  der auf  $u + 2^{i-1}$  folgt.

Das Netzwerk in [Abbildung 7] zeigt ein Netzwerk im Ring-mit-Ausfällen-und-Anhängen-Status.

Die Bedingung 5.d) garantiert, dass ein Knoten  $u$  nur neue Knoten in seine Successor-Liste aufnimmt, wenn er diese durch die Successor-Liste seines Successors kennenlernt. Ohne diese Bedingung könnte der Ring Loopy werden, falls zusätzlich Knoten ausfallen. 5.e) und 5.f) waren die Aktualität der Successor-Listen.

Diese intuitiven Bemerkungen, sowie die Lemmata zu den beiden vorherigen Zuständen führen zu folgendem Theorem:

**3.5.2.Theorem:** Wir betrachten ein Chord-Netzwerk im **Ring-mit-Ausfällen-und-Anhängen-Status** mit einer Successor-Liste der Länge  $c \log N$  und erlauben bis zu  $N$  beliebige "joins" und bis zu  $N/2$  beliebige "departures" zu einer beliebigen Zeit, wobei mindestens  $D \log^2 N$  Runden der Idealisierung durchlaufen worden sein müssen;  $D = O(1)$ . Dann gilt mit hoher Wahrscheinlichkeit:

1. Zu jedem Zeitpunkt kann während diesem Prozess  $\text{find\_successor}(q)$  den ersten lebenden Successor  $s$  von  $q$  oder einen Knoten  $u$  im Anhang  $A_s$  mit  $q < u < s$  in Zeit  $O(\log N)$  zurückliefern.
2. Das resultierende Netzwerk besitzt wieder den Ring-mit-Ausfällen-und-Anhängen-Status.

### **3.5.3. Beweis:**

Aus den gleichen Argumenten wie bei Lemma 3.1.2 und 3.2.2 folgt dass  $\text{find\_successor}()$  korrekte Ergebnisse in  $O(\log N)$  liefert.

Um den Beweis weiter formalisieren zu können, legen wir fest, dass ein Knoten *völlig in den Chord-Ring integriert* ist, genau dann wenn er bei einer Successor-Listen-Länge von  $c \log N$

länger als die Dauer von  $c^2 \log N$  durchgeführten Runden des Wartungsprotokolls auf dem Ring ist. Diese Bedingung ist erforderlich, da es nicht ausreicht, wenn ein Knoten  $v$  von einem anderen Knoten  $u$  der auf dem Ring liegt als Successor erkannt worden ist. Fällt nämlich  $u$  aus, so ist  $v$  auch wieder vom Netzwerk getrennt.

Wir unterscheiden 3 Arten von Knoten:

- *"Alte" Knoten*: Knoten die schon vor mehr als  $D \log^2 N$  Ausführungen der Idealisation zum Netzwerk hinzugekommen sind
- *"Knoten mittleren Alters"*: Knoten die nicht vor mehr als  $D \log^2 N$  Ausführungen der Idealisation zum Netzwerk hinzugekommen sind
- *"Neue" Knoten*: Knoten die zum Zeitpunkt unserer Betrachtung hinzukommen

Nach der Definition des Ring-mit-Ausfällen-und-Anhängen-Status sind alle alten Knoten völlig in den Chord-Ring integriert (5.e) und 5.f)). Da die Knoten zufällig auf dem Chord-Ring verteilt sind, kommen mit hoher Wahrscheinlichkeit zwischen zwei alte Knoten nur  $O(\log N)$  neue. Somit behält der Anhang die geforderte Größe, bis er wieder vom Wartungsprotokoll analysiert wird. Knoten auf dem Ring können zwar ausfallen, aber mit hoher Wahrscheinlichkeit fallen nie mehr wie  $O(\log N)$  aufeinander folgende Knoten aus. Wenn nun ein Knoten des Rings ausfällt, müssen Anhänge zusammengelegt werden. Nach 3.b) wird dieser Anhang dann auch wieder nicht größer als  $O(\log N)$ .

Nachdem alle "Knoten mittleren Alters" zum Chord-Ring hinzugekommen sind, wird zusätzlicher Wartungsaufwand von  $O(\log^2 N)$  benötigt, um sicher zu stellen, dass alle Einträge in den finger-tables korrekt gebildet sind.

## Zusammenfassung und Ausblick

Unsere Betrachtungen zeigen, dass Chord auch im nicht ganz idealen Zustand mit hoher Wahrscheinlichkeit korrekte und effiziente Suchergebnisse liefert. Dabei haben wir den Wartungsaufwand bei einem Chord-Ring im Ring-mit-Ausfällen-und-Anhängen-Status betrachtet, der betrieben werden muss, wenn sich zum Zeitpunkt  $t_2$   $n/2$  der Knoten vom Zeitpunkt  $t_1$  im Netz befinden<sup>7</sup>. Wir können nach  $O(\log^2 N)$  Runden der Idealisation den Ring wieder in diesen Zustand zurück versetzen.

Um aus einem beliebigen Status zu einem stark idealen Netzwerk zu kommen, müsste man die idealize-Funktion modifizieren. David Liben-Nowell, Hari Balakrishnan und David Karger haben gezeigt, dass es mit Modifikation der Wartungsfunktionen in  $O(N^2)$  Runden möglich ist, jedes beliebige Chord-Netzwerk stark ideal zu machen. Das heißt auch, dass auf diese Weise Netzwerke mit Schleifen wieder ideal werden können.

Es bleiben jedoch für weitere Arbeiten noch viele Fragen offen:

Bei der hier gezeigten Analyse ist es jedoch erforderlich, dass die Zeit<sup>8</sup>  $\Delta t$  in der  $N/2$  Knoten des Netzwerkes verschwinden bekannt ist, um die Runden, die das Wartungsprotokoll bewerkstelligen muss, bestimmen zu können. Ist es für einen Knoten möglich, durch Beobachtung anderer Knoten, selbst zu bestimmen, welcher Wartungsaufwand notwendig ist, um nicht vom Netzwerk getrennt zu werden?

Außerdem sind wir davon ausgegangen, dass in unserem Netzwerk keine Nachrichten verloren gehen. Wie wirkt sich dies auf den nötigen Wartungsaufwand aus?

---

<sup>7</sup> Dies sagt nichts über die Gesamtzahl der Knoten aus.

<sup>8</sup> Dieser Zeitraum wird oft als "Halflife" des Netzwerkes bezeichnet[1]

## Quellen

### [1] DAVID LIBEN-NOWELL, HARI BALAKRISHNAN, DAVID KARGER

**Analysis of the Evolution of Peer-to-Peer Systems**  
Laboratory for Computer Science  
Massachusetts Institute of Technology  
200 Technology Square  
Cambridge, MA 02139 USA  
{dln,hari,karger}@lcs.mit.edu

[2] DRUSCHEL, P., AND ROWSTRON, A. Past: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (2001), pp. 65–70.

[3] FIAT, A., AND SAIA, J. Censorship resistant peer-to-peer content addressable networks. In *Proc. SODA* (2002).

[4] KARGER, D., LEHMAN, E., LEIGHTON, F., LEVINE, M., LEWIN, D., AND PANIGRAHY, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. STOC* (1997).

[5] LEWIN, D. Consistent hashing and random trees: Algorithms for caching in distributed networks. Master's thesis, Department of EECS, MIT, 1998. Available at the MIT Library, <http://thesis.mit.edu/>.

[6] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proc. SIGCOMM* (2001).

[7] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM* (2001).

## Literatur

DABEK, F., BRUNSKILL, E., KAASHOEK, M. F., KARGER, D., MORRIS, R., STOICA, I., AND BALAKRISHNAN, H. Building peer-to-peer systems with Chord, a distributed location service. In *Proc. IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (2001).

DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proc. SOSP* (2001).

KUEIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An architecture for global-scale persistent storage. In *Proc. ASPLOS* (2000).

PANDURANGAN, G., RAGHAVAN, P., AND UPFAL, E. Building low-diameter P2P networks. In *Proc. FOCS* (2001).

PLAXTON, C., RAJARAMAN, R., AND RICHA, A. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. SPAA* (1997).

SAIA, J., FIAT, A., GRIBBLE, S., KARLIN, A. R., AND SAROIU, S. Dynamically fault-tolerant content addressable networks. In *Proc. IPTPS* (2002).

STOICA, I., MORRIS, R., LIBEN-NOWELL, D., KARGER, D., KAASHOEK, M. F., DABEK, F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. Tech. Rep. TR-819, MIT LCS, 2001.  
<http://www.pdos.lcs.mit.edu/chord/papers/>.