Peer-to-Peer-Systeme

Teil V: Gradminimierte Netze

Björn Scheuermann

Humboldt-Universität zu Berlin
Wintersemester 2015/16

Reflexion: Kriterien zum Vergleich von DHTs

Anhand welcher Kriterien können wir DHTs bewerten oder vergleichen?

Reflexion: Kriterien zum Vergleich von DHTs

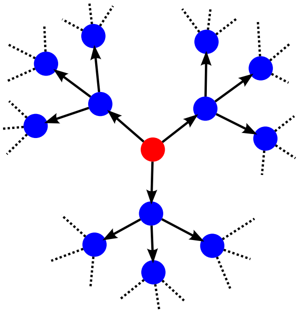
- Kommunikationsaufwand
  - für das Routing (Lookup-Pfadlänge)
  - für die Verwaltung der Datenstruktur (Join/Leave, Reparaturen,...)
- Robustheit, Zuverlässigkeit
  - gegen Ausfälle
  - gegen Angreifer
- Fairness
  - bezüglich der Schlüsselverteilung
  - bezüglich der Routing-Last
- Zustandsgröße
  - insbes. die Zahl der Verbindungen im Overlay (Grad)
- ...

Wir betrachten nun speziell das Zusammenspiel von Pfadlängen und Grad der Knoten im Overlay etwas genauer

Grad vs. Durchmesser – Prinzipielle Schranken

- Stellen wir uns einen beliebig konstruierten Overlay-Graphen vor, in dem jeder Knoten genau d ausgehende Kanten hat
- In einem Schritt können d Knoten, mit genau zwei Schritten höchstens $d \cdot d$ Knoten erreicht werden, usw.
- In genau h Schritten höchstens d^h Knoten
- In *höchstens h* Schritten also maximal

$$\sum_{i=0}^{h} d^i = \frac{d^{h+1} - 1}{d - 1} < d^{h+1}$$



Grad vs. Durchmesser – Prinzipielle Schranken

Da wir in h Schritten auf keinen Fall mehr als d^{h+1} Knoten erreichen können, gibt es einen *Tradeoff zwischen Grad und Durchmesser*

- Um bei n Peers von Grad d einen Durchmesser von h erreichen zu können, muss $d^{h+1} > n$ sein
- Wenn wir den Grad konstant halten wollen, müssen wir einen Durchmesser von $\Omega(\log n)$ akzeptieren, denn

$$\begin{aligned} & d^{h+1} > n \\ \iff \quad & h > \log_d n - 1 \end{aligned}$$

Grad vs. Durchmesser in CAN und Chord

- CAN hat konstanten Grad, aber polynomiellen Durchmesser
- Das ist – aus dieser Perspektive – nicht wirklich optimal!

- Chord hat logarithmischen Durchmesser, dafür aber auch einen logarithmischen Grad
- Auch nicht optimal...

- Jetzt: *Gradminimierte* Netzwerke – konstanter Grad, logarithmischer Durchmesser

Kontinuierliche Graphen

- Wir besprechen nun eine DHT namens „Distance Halving“
- Distance Halving beruht auf dem Prinzip von *kontinuierlichen Graphen*; das sind Graphen, in denen die Knotenmenge V kontinuierlich ist
- Im Falle von Distance Halving ist V das Intervall $[0, 1)$
- Die Kantenmenge E in einem kontinuierlichen Graphen ist (wie in einem „normalen“, diskreten Graphen) eine Teilmenge von $V \times V$

[Naor, Wieder: Novel Architectures for P2P Applications: the Continuous-Discrete Approach, Transactions on Algorithms, 2006]

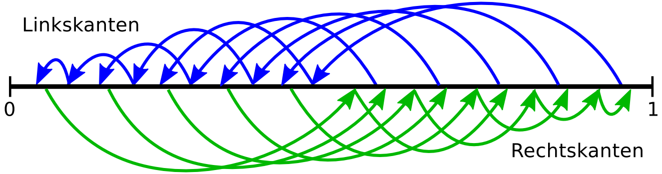
Der Distance-Halving-Graph

Auf der Knotenmenge $V = [0, 1)$ werden zwei Funktionen definiert:

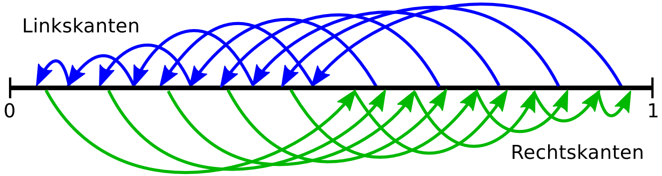
$$l(x) = \frac{x}{2} \qquad\qquad r(x) = \frac{x}{2} + \frac{1}{2}$$

Diese Funktionen definieren *Linkskanten* und *Rechtskanten* ausgehend von jedem Punkt aus V:

- Linkskante: $(x, l(x)) \in E$
- Rechtskante: $(x, r(x)) \in E$



Der Distance-Halving-Graph

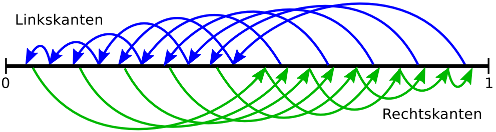


$$l(x) = \frac{x}{2} \qquad r(x) = \frac{x}{2} + \frac{1}{2}$$

Wie viele ausgehende Kanten hat jeder Punkt?
Wie viele eingehende Kanten hat jeder Punkt?
Wo beginnen die eingehenden Kanten?

- Jeder Knoten hat genau zwei ausgehende Kanten und eine eingehende Kante (!)

Der Distance-Halving-Graph



- Die eingehende Kante in Punkt x beginnt im Punkt

$$b(x) = 2x \mod 1$$

  Sie wird auch die *Rückwärtskante* von x genannt
- Achtung: Die Rückwärtskanten existieren nicht „zusätzlich", sondern die Rückwärtskante von x ist Links- oder Rechtskante eines anderen Punktes $b(x)$
- Betrachtet man diesen Graphen als *ungerichteten* kontinuierlichen Graphen, hat jeder Knoten x Grad drei:
  - eine Linkskante $(x, \frac{x}{2})$
  - eine Rechtskante $(x, \frac{x}{2} + \frac{1}{2})$
  - eine Rückwärtskante $(x, 2x \mod 1)$

Diskretisierung kontinuierlicher Graphen

Aus dem kontinuierlichen Graphen lässt sich in der folgenden Weise ein diskreter Graph mit n Knoten erzeugen:

- Teile V in n Partitionen $V_1, \ldots, V_n$ auf, sodass

$$\bigcup_{i=1}^{n} V_i = V \qquad \text{und} \qquad i \neq j \;\Rightarrow\; V_i \cap V_j = \emptyset$$

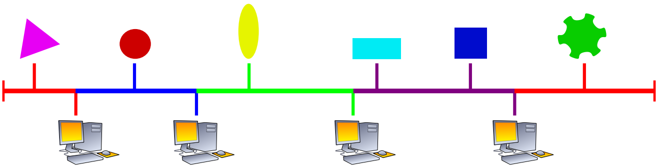
- Die Knotenmenge des diskreten Graphen ist $V_1, \ldots, V_n$
- Der diskrete Graph enthält die Kante (V_i, V_j) genau dann, wenn

$$\exists v_i \in V_i, v_j \in V_j : \quad (v_i, v_j) \in E$$

- Achtung:
  - Im kontinuierlichen Graphen hatte jeder Punkt genau je eine Links-, Rechts- und Rückwärtskante
  - Im diskretisierten Graphen hat ein Knoten (= Intervall) im Allgemeinen mehrere Links-/Rechts-/Rückwärtskanten!

Der diskrete Distance-Halving-Graph

- Um den (diskreten) Overlay-Graphen des Distance-Halving-Netzwerks zu generieren, wird das Intervall $[0, 1)$ auf die Peers aufgeteilt
- Dies geschieht ähnlich wie in Chord:
  - Jedem Peer wird ein Punkt des Intervalls zugeordnet
  - Dem Peer gehört das Intervall links von seinem Punkt (modulo 1, also wieder als gedachter Ring)
  - Peers sind für die Schlüssel in ihrem Intervall zuständig
- Zusätzlich zu den Rechts- und Linkskanten werden *Ring-Kanten* eingefügt, die benachbarte Peers verbinden



Grad im diskreten Distance-Halving-Graph

Wir betrachten nun den Durchschnittsgrad in diesem Graphen

Der durchschnittliche Knotengrad im diskreten Distance-Halving-Graphen ist kleiner als 8, also *konstant*

Zeige per Induktion, dass bei n Peers insgesamt höchstens $4n - 1$ Kanten existieren.

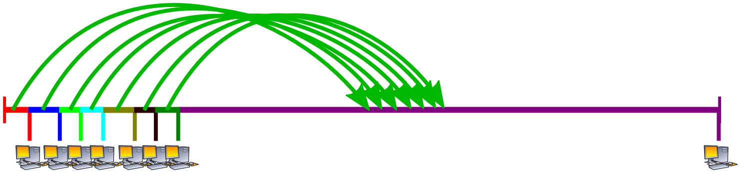
Induktionsanfang: Für $n = 1$ gibt es drei Kanten (eine Linkskante, eine Rechtskante, eine Ringkante).

Induktionsschritt: Durch Hinzufügen des $n + 1$-ten Peers wird ein Peer-Intervall $[x, z)$ an einem Punkt y geteilt. Das Segment, das $l(y) = \frac{y}{2}$ enthält, ist das einzige, in das sowohl $[x, y)$ als auch $[y, z)$ eine Linkskante haben. Deshalb entsteht durch das Einfügen höchstens eine neue Linkskante im diskreten Graphen.

Analog entstehen auch maximal eine neue Rechts- und eine Rückwärtskante, außerdem eine Ringkante, zusammen also höchstens vier neue Kanten.

Grad im diskreten Distance-Halving-Graph

- Das sagt aber noch nichts aus über den maximalen Grad einzelner Knoten – einzelne Peers können dennoch eine sehr hohe Zahl von Nachbarn haben:



- Offensichtlich ist es wichtig, dass das Intervall *gleichmäßig* zwischen den Peers aufgeteilt wird
- Sei s_i die Länge des Intervalls des i-ten Peers; definiere die *Gleichmäßigkeit* (engl. *smoothness*) der Aufteilung als

$$\rho = \max_{i,j} \frac{s_i}{s_j} \qquad \left(= \frac{\max_i s_i}{\min_i s_i} \right)$$

Grad im diskreten Distance-Halving-Graph

Der Ausgrad jedes Knotens ist maximal $\rho + 4$, der Eingrad jedes Knotens höchstens $\lceil 2\rho \rceil + 1$

Beweis: Sei $[a, b)$ das längste, $[c, d)$ das kürzeste Intervall. Dann ist $|d - c| = |b - a|/\rho$. Außerdem gilt, dass $|r(b) - r(a)| = |b - a|/2$. Deshalb schneiden höchstens

$$\lceil\, |r(b) - r(a)| \;/\; |d - c|\, \rceil + 1 = \left\lceil \frac{|b-a|}{2} \;/\; \frac{|b-a|}{\rho} \right\rceil + 1 = \left\lceil \frac{\rho}{2} \right\rceil + 1$$

verschiedene Peer-Intervalle das Intervall $[r(a), r(b))$. Analog für $[l(a), l(b))$; deshalb ist der Ausgrad beschränkt durch

$$2\left(\left\lceil \frac{\rho}{2} \right\rceil + 1\right) \leq \rho + 4.$$

Der Eingrad folgt entsprechend.

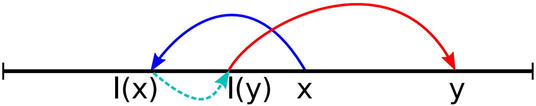
Für eine gleichmäßige Intervallaufteilung (d. h. beschränkte Gleichmäßigkeit ρ) ist also auch der Worst-Case-Grad in $O(1)$

(To-Do für später: Wie erreicht man Gleichmäßigkeit?)

Routing im Distance-Halving-Overlay

- Es gibt gleich mehrere Ansätze, um im Distance-Halving-Graphen in $O(\log n)$ Schritten zu routen
- Wir besprechen hier einen der einfacheren (aber dennoch effektiven und eleganten!)

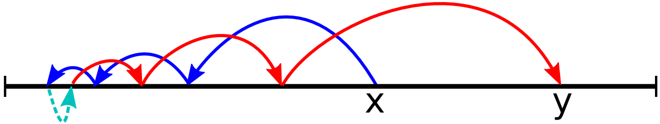
- Beobachtung:
  - Wenn wir von x nach y routen wollen, können wir
    - zunächst von x nach $l(x)$ über die Linkskante,
    - dann von $l(x)$ (mit irgendeinem noch zu definiernden Verfahren) nach $l(y)$
    - und schließlich von $l(y)$ über die Rückwärtskante nach y



  - Die Distanz zwischen $l(x)$ und $l(y)$ ist nur halb so groß wie die zwischen x und y

Routing im Distance-Halving-Overlay

- Idee für ein einfaches Routingverfahren:
  Schachtele solche Routingschritte rekursiv ineinander!
- In jedem Schritt halbiert sich die verbleibende Distanz



- Nach t Schritten ist der Abstand also höchstens noch 2^{-t}
- Spätestens wenn 2^{-t} kleiner geworden ist als die kleinste Intervalllänge, liegen $I^t(x)$ und $I^t(y)$ entweder im selben Peer oder in zwei direkt benachbarten Peers
- Von dort wird (ggf. nach einem Ringkanten-Schritt) dann über die Rückwärtskanten zu y geroutet

Routing-Algorithmus

Algorithmisch lässt sich Routing von x nach y dann folgendermaßen formulieren:

Algorithmus Links-Routing(x, y)

if x und y benachbart **then**
  sende die Anfrage von x nach y
else
  $x' \leftarrow l(x)$
  $y' \leftarrow l(y)$
  sende die Anfrage von x nach x'
  Links-Routing(x', y') *// rekursiver Aufruf*
  sende die Anfrage von y' nach y
end if

(Wichtig zu beachten: Dieser Algorithmus wird *verteilt im Netzwerk* ausgeführt, nicht von einem einzelnen Knoten! Das nächste Kommando führt immer der Knoten aus, bei dem die Nachricht gerade liegt.)

Routing-Komplexität

Für eine feste Gleichmäßigkeit $\rho \geq 1$ bei n Peers sind für das Routing nur $O(\log n)$ Schritte notwendig

Beweis: Es muss mindestens ein Intervall der Länge $\geq 1/n$ geben; also ist das kleinste Intervall mindestens $1/(n\rho)$ lang.

Der verbleibende Abstand nach dem t-ten rekursiv geschachtelten Routingschritt ist höchstens noch 2^{-t}, also wird die Länge des kleinsten Intervalls erreicht, sobald

$$\begin{aligned} 2^{-t} &= \frac{1}{n\rho} \\ \Leftrightarrow \quad t &= \log_2 n\rho \end{aligned}$$

Es sind also höchstens $\lceil \log_2 n\rho \rceil$ Rekursionsschritte nötig und die Anfrage muss insgesamt maximal

$$\underbrace{\lceil \log_2 n\rho \rceil}_{\text{Hin}} + \underbrace{1}_{\text{Ringkante}} + \underbrace{\lceil \log_2 n\rho \rceil}_{\text{Zurück}} \in O(\log n)$$

Mal weitergeleitet werden.

Lastbalanciertes Routing

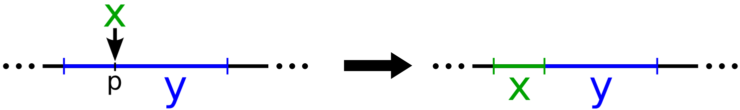
- Der vorgestellte Algorithmus verwendet *nur* die Linkskanten!
- Funktioniert völlig analog natürlich auch mit den Rechts- statt den Linkskanten
- Problem in beiden Fällen: Die Knoten am linken (bzw. rechten) Rand werden sehr stark belastet
- Dies lässt sich verbessern, indem in jedem Rekursionsschritt zufällig gewählt wird, ob Links- oder Rechtskanten benutzt werden sollen

Erzeugen gleichmäßiger Intervallaufteilungen

- Noch offenes Problem: Wie können wir eine gleichmäßige Aufteilung des Intervalls auf die Peers (und damit ein beschränktes ρ) erreichen?
- Dies ist auch für andere Overlays als Distance Halving interessant (z. B. für Chord!)
- Wir fangen mit einem sehr einfachen Verfahren an, und werden es dann schrittweise verbessern

Aufteilen durch zufälliges Einfügen

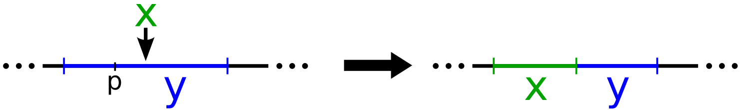
- Ganz einfaches Verfahren (analog Chord):
  1. Neuer Peer x wählt einen zufälligen Punkt $p \in [0, 1)$
  2. Sucht den Peer y, der gegenwärtig für p zuständig ist
  3. Dessen Intervall wird im Punkt p geteilt, der neue Peer übernimmt einen Teil



- Bei diesem Verfahren ist
  - das längste Segment m. h. W. $\Theta\left(\frac{\log n}{n}\right)$ lang
  - das kürzeste Segment m. h. W. nicht kürzer als $\Theta\left(\frac{1}{n^2}\right)$
- Das macht uns also nicht wirklich glücklich, denn ρ steigt dann mit wachsendem n!

Aufteilen in der Intervallmitte

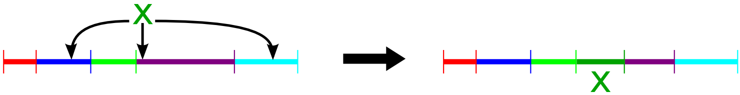
- Verbessertes Verfahren:
  1. Neuer Peer x wählt einen zufälligen Punkt $p \in [0, 1)$
  2. Sucht den Peer y, der gegenwärtig für p zuständig ist
  3. Dessen Intervall wird *in der Mitte* geteilt, der neue Peer übernimmt eine Hälfte



- Hierbei ist
  - das längste Segment noch immer $O\left(\frac{\log n}{n}\right)$
  - das kürzeste Segment nun m. h. W. $\Theta\left(\frac{1}{n \log n}\right)$
- ρ steigt also auch hier noch mit wachsendem n ...

Einfügen durch Mehrfachauswahl

- Nochmals verbessertes Verfahren:
  1. Neuer Peer schätzt $\log n$ (dafür gibt es Verfahren)
  2. Wählt dann $t \log n$ Punkte zufällig (für richtig gewählte Konstante t)
  3. Überprüft alle Segmente, in die diese Punkte fallen
  4. Fügt sich in der Mitte des längsten der getroffenen Intervalle ein



- Nach n Einfügeoperationen sind m. h. W. die Längen aller Segmente in $\Theta(1/n)$
- Gilt sogar unabhängig vom Zustand davor – das Verfahren ist also „selbstreparierend“
- Es existieren weitere Techniken, um Gleichmäßigkeit auch beim Wegfall von Peers zu erhalten

Distance Halving – Zusammenfassung

- Wir haben nun also eine verteilte Hashtabelle kennengelernt, die asymptotisch *gradoptimal* ist:
  - Nachbarschaftsgröße $O(1)$ ...
  - ... und dennoch Routing in $O(\log n)$ Schritten
- Distance Halving ist mittels kontinuierlicher Graphen konstruiert
- Die Eigenschaften hängen wesentlich von der Gleichmäßigkeit der Intervallaufteilung ab
- Das bedeutet natürlich *nicht*, dass dies in allen Situationen die beste Wahl wäre – die Welt ist komplizierter als Grad und Durchmesser!