Kademlia

A Peer-to-peer Information System based on the XOR Metric

Martynas Ausra

12.5.2007

Zusammenfassung

Das Kademlia-Protokoll wurde im Jahr 2002 an New York University von Petar Maymounkov und David Mazieres entwickelt. Es handelt sich um ein *peer-to-peer* System *Kademlia*, das auf einer verteilten Hash-Tabelle (DHT) basiert. Aus dem Einsatz der XOR-Metrik-Topologie als Distanzfunktion zwischen Knoten im virtuellen Schlüsselraum resultiert die effiziente Lösung des Wörterbuch-Problems. Durch die rekursive (parallele) Implementierung des *lookup*-Algorithmus wird die Komplexität des Systems noch deutlich verbessert. Durch die spezielle Ordnung der Routing-Tabellen (k-buckets) wird auch die Konsistenz des Systems gesichert.

Inhaltsverzeichnis

1 Art und Aufbau des Netzes

Über den bestehenden *local area network* wird eine dezentrale virtuelle Netzwerkstruktur (DHT) aufgebaut. Die Indexierung der Informationen geschieht hier im Gegensatz zu Systemen mit zentralen Indexierungsserver (wie Napster, Gnutella) durch *clients*, die einen bestimmten Inhalt der Informationen (Hashwert) darstellen und vorteilhaft im virtuellen Raum verteilt werden. Mit Hilfe von lookup-Anfragen ist somit möglich alle gesuchten Informationen im Netz zu finden, sobald sie da vorhanden sind.

Nach Quelle [2] ist die Grundlage von Kademlia das *User Datagram Protocol* (UDP): minimales, verbindungsloses Netzprotokoll, das zur Transportschicht der Internetprotokollfamilie gehört. Aufgabe von UDP ist es, Daten, die über das Internet übertragen werden, der richtigen Anwendung zukommen zu lassen. Um die Daten, die mit UDP versendet werden, dem richtigen Programm auf dem Ziel-Rechner zukommen zu lassen, werden bei UDP so genannte Ports verwendet. Dazu wird bei UDP die Port-Nummer des Dienstes mitgesendet, der die Daten erhalten soll. Dies nennt man eine Erweiterung der *host-to-host*- auf eine Prozess-zu-Prozess-Übertragung. Zusätzlich bietet UDP die Möglichkeit einer Integritätsüberprüfung an, indem eine Prüfsumme mitgesendet wird. Dadurch kann eine mögliche fehlerhafte Übertragung erkannt werden.

2 Kademlia-Knoten

Kademlia besitzt über gemeinsamen virtuellen Adressraum für Knoten-IDs und Schlüssel mit 160 Bit Breite. Jeder am Netz beteiligte Client besitzt eine eindeutige, anfangs zufällig generierte 160 Bit Knoten-ID. Jedem Informationsinhalt (value), der im Netzwerk abgelegt oder gesucht wird, wird ein eindeutiger 160 Bit Hash-Wert (key) durch DHT zugewiesen. Somit ist jeder Schlüssel ein eindeutiger Repräsentant eines bestimmten Informationsinhalts. Um eine Information im Netzwerk zu finden, wird ihr durch DHT zugewiesene Hashwert (key) als Schlüssel-Wert-Paar auf mehreren Knoten mit einer zum Schlüssel ähnlichen ID gespeichert. Die Anonymität der Knoten geht

hiermit verloren, da jeder Empfänger einer Anfrage, sobald er von einem anderen Knoten im Netzwerk (Sender) kontaktiert wird, sofort seine Daten übermittelt bekommt. Um die durch DHT im Netz gebildeten *(key,value)* Paare zu publizieren oder zu finden, benutzt Kademlia den Begriff der Distanz zwischen zwei Bitsequenzen.

3 XOR-Metrik

Im Kademlia-Paper ([4] Abs. 2.1) beschreiben die Autoren die Eigenschaften der XOR-Metrik.

Definition: Gegeben seien zwei B=160 Bit Sequenzen x und y. Um die Distanz zwischen x und y zu berechnen, wird in Kademlia die XOR-Funktion bitweise eingesetzt, deren Ergebnis dezimal ausgewertet wird:

$$d(x, y) = x \oplus y$$

XOR ist eine gültige, nicht-euklidische Metrik, für die folgende Eigenschaften gelten:

$$0 \leq \{x, y, z\} \leq (B - 1) :$$

Identität: $d(x, x) = 0$
Die Distanz von einem Knoten im Netzwerk zu sich selbst beträgt Null.

Existenz: $d(x, y) > 0, x \neq y$
Die Eigenschaft der Existenz besagt, dass, wenn es zwei verschiedene Knoten im Netzwerk existieren, so ist die Distanz zwischen den Knoten positiv.

Symmetrie: $\forall x, y : d(x, y) = d(y, x)$
Die Eigenschaft der Symmetrie sorgt für die Konsistenz der Routingtabellen. Ein beliebiger Knoten im Netzwerk ist zu seinen Kontakt-Einträgen bezüglich deren Distanzen symmetrisch.

Dreiecksungleichung: $d(x, y) + d(y, z) \geq d(x, z)$
Die Dreiecksungleichung garantiert eine effizientere Wahl der *lookup*-Routen und sorgt damit für die Steigerung der Latenz.

Transitivität: $\exists y : d(x,z) = d(x,y) \oplus d(y,z)$

Die Eigenschaft der Transitivität bietet eine Akkumulierung der Teildistanzen zur Gesamtdistanz zwischen zwei beliebigen Knoten im Netzwerk an.

Unidirektionalität: Für jeden gegebenen Punkt x und Distanz $\epsilon > 0$, existiert genau ein Punkt y, so dass $d(x,y) = \epsilon$. Diese Eigenschaft garantiert, dass alle *lookups* für den selben Schlüssel gegen einen und immer den selben Pfad konvergieren, unabhängig davon, welcher Knoten die *lookup*-Anfrage gesendet hat. Somit werden die entlang eines *lookup*-Pfades liegenden *(key,value)* Paare zwischengespeichert und können bei einer identischen *lookup*-Anfrage wieder aufgerufen werden.

4 Routing-Tabellen

Kademlia-Knoten beinhalten Informationen übereinander, um effizient (O(log n)) Nachrichten untereinander auszutauschen. Um den ganzen Adressraum zu erfassen, führt jeder Knoten $n \in \{2^0, \ldots, 2^{B-1}\}$ genau B (=160) viele Listen von Tupeln *(IP-Adresse, UDP-Port, Knoten-ID)*. Es wird angenommen, dass in der $i-ten$ $(0 \leq i \leq B-1)$ Liste jeweils Platz für k viele Tupel aus *(IP,UDP,ID)* gibt, wobei jedes Tupel genau einen Kontakt-Knoten im Kademlia-Netzwerk beschreibt. Nach den Autoren des Kademlia-Systems, wird die Konstante k so gewählt, dass ein Ausfall aller Knoten einer solchen Liste innerhalb einer Stunde sehr unwahrscheinlich ist. Nach Quelle [1] (Abs. 2) ist $k = 20$. Diese Listen werden also *20-buckets* genannt.

4.1 20-buckets

Im $i-ten$ $(0 \leq i \leq B-1)$ 20-bucket speichert ein Knoten $n \in \{2^0, \ldots, 2^{B-1}\}$ diejenigen Kontakte, die um 2^i bis 2^{i+1} von ihm (n) entfernt sind. Für kleinere Werte i sind generell die *20-buckets* fast leer, da $|\{2^i, \ldots, 2^{i+1}\}| < |\{2^{i+1}, \ldots, 2^{i+2}\}|$ mit $0 \leq i \leq B-1$. Für große Werte i können die 20-buckets voll sein. Die Anordnung der *20-buckets* illustriert die **Abbildung 1**.

4.2 Interne Organisation der 20-buckets

In diesem Abschnitt wird die von den Autoren des Kademlia-Systems (Quelle [4] Abs. 2.1) beschriebene Organisation innerhalb der 20-buckets, die die Kon-

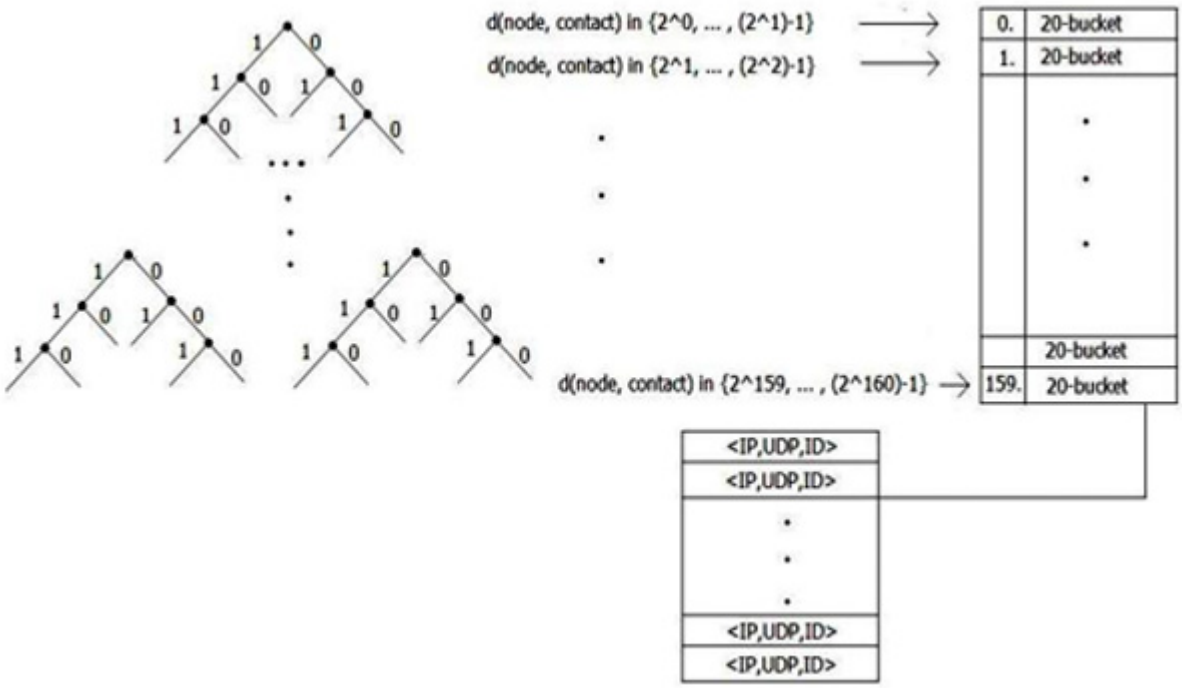

Abbildung 1: k-buckets

sistenz des Informationsflusses innerhalb des Kademlia-Netzes maximiert, wiedergegeben.

Jeder *20-bucket* innerhalb eines Knotens n wird nach LS-Strategie (*last seen*) sortiert. Knoten, mit denen am längsten nicht kontaktiert wurde, befinden sich ganz am Anfang eines *k-buckets* (*head*), und umgekehrt: Knoten, mit denen neulich kontaktiert wurde, stehen ganz am Ende eines *k-buckets* (*tail*).

Aus diesem Aufbau ergibt sich, dass ein Knoten über ein hohes Wissen über seine Nachbar verfügt, aber nur wenige Knoten kennt, die sehr weit von ihm entfernt sind.

Sobald ein Kademlia-Knoten eine Nachricht (Anfrage oder Antwort) von einem anderen Knoten erhält, wird der entsprechende *20-bucket* des Empfängers mit der ID-Nummer des Senders aktualisiert.

1. Falls der ID-Eintrag des Senders im *20-bucket* des Empfängers schon existiert, wird er von dem Empfänger am *tail* der Liste verschoben.

2. Falls der ID-Eintrag des Senders noch nicht im *20-bucket* des Empfängers eingetragen ist, und der *20-bucket* noch nicht voll ist, wird er von dem Empfänger am *tail* der Liste generiert.

3. Falls der ID-Eintrag des Senders noch nicht im *20-bucket* des Empfängers eingetragen ist, und der *20-bucket* bereits voll ist, versucht der Empfänger dem aktuellen *head*-Knoten aus seiner Liste ein *ping* Signal zu senden:

   - Der Knoten am *head* antwortet nicht. Er wird aus der Liste eliminiert und der neue Sender-Knoten wird in die Liste am *tail* eingefügt.
   - Der Knoten am *head* antwortet. Ihm wird die neue Position *tail* in der Liste zugewiesen. Die Anfrage des neuen Sender-Knotens wird dann abgelehnt.

4.3 Vorteile

1. Die Anzahl der Konfigurationsnachrichten, die zwischen Knoten gesendet werden müssen, um Kontakt-Informationen übereinander zu aktualisieren, wird durch den Einsatz der 20-buckets minimiert. Die Konfigurationsinformationen werden somit automatisch als Seiteneffekt von *key-lookups* aktualisiert.

2. *Uptime function* (Quelle [4] Abs. 2.1): Je länger ein Knoten aktiv bleibt, desto wahrscheinlicher wird es, dass er sich noch eine Stunde länger online aufhält. Die Wahrscheinlichkeit, dass die in *20-buckets* eines Knotens enthaltene aktive Kontakte auch weiterhin aktiv bleiben, wird maximiert, indem man die ältesten online Kontakte durch *Cashing* immer zugriffsbereit hält.

5 Austausch von Informationen

Das Kademlia-Protokoll kommt nach Quelle [4] (Abs. 2.2) mit vier RPC's: PING, STORE, FINDNODE, FINDVALUE aus.

5.1 Instruktionen

PING RPC Instruktion ermittelt, ob ein Knoten online ist.

STORE RPC Instruktion speichert ein Schlüssel-Wert-Paar auf einem Knoten für eventuell spätere Zugriffe ab.

FINDNODE RPC Instruktion erhält eine 160 Bit Knoten-ID als Argument. Der Empfänger dieser Instruktion gibt k (20) Tupel *(IP,UDP,ID)* zurück, die diejenigen k (20) Knoten repräsentieren, die am nächsten an der gesuchten Knoten-ID liegen.

FINDVALUE RPC Instruktion erhält eine 160 Bit *value* als Argument. Falls bei dem Empfänger dieser Instruktion kein STORE zuvor ausgeführt wurde, gibt er auch die k (20) Tupel *(IP,UDP,ID)* zurück, die diejenigen k (20) Knoten repräsentieren, die am nächsten am gesuchten *value* liegen. Ansonsten (falls bei dem Empfänger zuvor ein value abgelegt wurde) gibt er lediglich den value zurück.

5.2 Knoten-lookup Algorithmus

Mit Hilfe der in Quelle [1] (Abs. 4.5) beschrieben *lookup*-Prozedur wird für eine gegebene Knoten-ID maximal k (20) viele PING-fähige Knoten lokalisiert, deren Knoten-IDs am nächsten an der gesuchten Knoten-ID liegen. Kademlia benutzt einen rekursiven *lookup*-Algorithmus (parallele Version), der auf die in der **Abbildung 2** dargestellte iterative Version basiert:

- Befrage α viele Knoten aus eigenem Ziel-ID-nächsten nicht leeren *k-bucket*: schicke jedem parallel eine FINDNODE RPC. Falls der befragte *k-bucket* weniger als α Knoten beinhaltet, so werden Knoten von anderen in der Nähe liegenden *k-buckets* ausgewählt, allerdings max α.
- Im Rekursionsschritt gibt jeder der α PING-fähigen (befragten) Knoten genau k viele Ziel-ID-nächste Tupel aus eigenen *k-buckets* zurück.
- Aus diesen $\alpha * k$ vielen Tupeln wähle wieder genau α viele Ziel-ID-nächste Knoten.
- Wiederhole diese Prozedur solange, bis entweder in der aktuellen $\alpha * k$ Menge kein Knoten näher am Ziel liegt als der bisher Betrachtete oder es wurden bereits k (20) Ziel-ID-nächste aktive Kontakte ermittelt.
- k viele Ziel-ID-nächste Tupel *(IP,UDP,ID)* werden schließlich zurückgegeben.
- Nach Quelle [1] (Abs. 4.5.1) ist $\alpha = 3$ ein optimaler Richtwert.

- **Goal:** Find the k nodes closest to a given target $T \in \{0,1\}^{160}$
- **RPC:** $\mathsf{find\_node}_n(T)$ returns all contacts from the (first non-empty) k-bucket in n's routing table that is closest to T
- **Lookup:**

  $n_o =$ ourselves (the node that is performing the lookup)

  $N_1 = \mathsf{find\_node}_{n_o}(T)$

  $N_2 = \mathsf{find\_node}_{n_1}(T)$

  $\ldots$

  $N_l = \mathsf{find\_node}_{n_{l-1}}(T)$,

  this completes when N_l contains no contacts that haven't been called already
- n_i is any contact in N_i

Abbildung 2: lookup

5.3 Beitreten des Netzes

Ein dem Kademlia-Netzwerk beitretender Knoten n hat am Anfang keine Einträge in seinen *20-buckets*. Um seine Nachbar-Knoten 'kennen zu lernen' und sich im Netzwerk 'bekannt zu machen' geht er nach folgendem Schema vor:

- n generiert eigene Knoten-ID mit einer konsistenten Hash-Funktion der DHT.
- n führt einen Knoten-lookup nach seiner eigenen Knoten-ID im Netzwerk aus.
- n aktualisiert eigene *20-buckets* mit den nach der Terminierung des *lookup*-Algorithmus zurückgelieferten (in der nahen Umgebung von n liegenden)(IP,UDP,ID) Tupeln. Gleichzeitig werden auch die *20-buckets* der eingetragenen Kontakte mit (IP,UDP,ID) von n aktualisiert.

5.4 Speicherung von (key,value)

Um ein Schlüssel-Wert-Paar im Kademlia-Netzwerk abzulegen, wird zuerst die FINDNODE Prozedur nach dem entsprechenden Schlüssel (key) ausgeführt, um die k (20) key-nächsten Knoten zu bestimmen. Anschließend wird mit Hilfe von STORE RPC auf diesen Knoten das entsprechende Schlüssel-Wert-Paar abgelegt.

Um die Speicherung der Informationen im Kademlia-System konsistent zu behalten, fügen die Autoren (Quelle [4] Abs. 2.2) folgende Bedingung ein: sobald ein Knoten n einen neuen Knoten n' im Netzwerk entdeckt, der näher an einigen Schlüssel-Wert-Paaren von n liegt als n selbst, so übergibt der Knoten n die entsprechenden Paare an Knoten n', ohne sie aus der Datenbank von n zu löschen.

5.5 Aktualität von (key,value)

Basierend auf die Daten in Quelle [1] (Abs. 2) werden bestimmte zeitliche Beschränkungen für die Aktualität der (key, value)-Paare im Kademlia-Netzwerk festgelegt.

- **tExpire:** Referenzen, die veränderte oder nicht mehr existierende Hash-Werte referenzieren, werden vermieden, da die (key, value)-Paare automatisch durch 24 Stunden-Stempel im Netzwerk verfallen.

- **tRepublish:** Der entsprechende Hash-Wert (ob aktualisiert oder nicht) muss somit alle 24 Stunden wieder neu vom Besitzer ins Netzwerk mit Hilfe von STORE RPC abgelegt werden.
- **tRefresh:** Knoten, auf denen (key,value)-Paare abgelegt sind, legen diese jede Stunde neu im Netzwerk ab. Somit wird die Verteilung der Paare auf mindestens k Knoten im Netzwerk sichergestellt.

6 Schlussfolgerung

6.1 Eigenschaften

- Die Steigerung der Resistenz gegen *DoS*-Attacken durch die Nutzung der dezentralen Indexierungsstruktur: dynamische Netzwerk-Struktur.
- Realisierung der lookup-Funktion mit logarithmischer Komplexität.
- Cashing der (key,value)-Paare: auch beim Ausfall vieler beteiligter Knoten ist keine wesentliche Latenzsteigerung zu erwarten.

6.2 Verbreitung

Diese Eigenschaften haben wahrscheinlich auch zu der derzeitigen Verbreitung geführt: Overnet, eMule, BitTorrent.

Literatur

[1] Kademlia: A Design Specification
http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

[2] Kademlia
http://de.wikipedia.org/wiki/Kademlia

[3] User Datagram Protocol
http://de.wikipedia.org/wiki/UDP

[4] Paper: a peer-to-peer Information System based on the XOR Metric
http://pdos.csail.mit.edu/ petar/papers/maymounkov-kademlia-lncs.pdf