Nicht Plaxtonbasierte DHTs und Bereichsabfragen

Jörg Meltzer

23. Mai 2003

Zusammenfassung

Ziel dieses Dokumentes ist es, einen allgemeinen Einblick in CAN-basierte DHTs zu verschaffen und Erweiterungen des CAN Designs für Bereichsabfragen zu erklären. Es wird zunächst das grundlegende Design des P2P Overlaynetzes erklärt, sowie anschliessend Optimierungen hinsichtlich Performance, Fehlertoleranz und Robustheit vorgestellt.
Der zweite Teil dieses Dokuments befaßt sich mit den für Bereichsabfragen nötigen Änderungen und Erweiterungen am CAN Design. Es werden weitere speziell für Bereichsabfragen vorgestellte Optimierungsverfahren analysiert.

Inhaltsverzeichnis

1 Motivation

Suchen von Objekten in einem Peer2Peer Netzwerk wird immer raffinierter. Von erster Generation P2P Netzen wie Napster und Gnutella ging es weiter über intelligent strukturierte verteilte Hashtabellen die auf Plaxtons Algorithmus aufbauen. Pastry und Tapestry lockerten Plaxtons einfaches statisches Design auf und schufen eine selbstorganisierende aber komplexere Infrastruktur. Sei n die Anzahl der Knoten in dem P2P Netzwerk, so benötigten alle der bisher vorgestellten Algorithmen $O(\log n)$ Hops im Overlay Network, jedoch ebenfalls $O(\log n)$ Speicherplatz für ihre Nachbarlisten.
Fragen, die zu beantworten sind: Gibt es ein Design, das mit einer konstanten Anzahl von zu merkenden Nachbarn für Knoten im P2P Netz auskommt? Sind effiziente Routingergebnisse und eine robuste Infrastruktur mit einer solch losen Kopplung der Knoten möglich? Wir suchen eine effiziente Indexierungsinfrastruktur, die die Bereichsabfragen ermöglicht. Möglichst wenig Netzlast soll bei Bereichsabfragen entstehen. Wie werden Replikations und Cachingstrategien sinnvoll eingesetzt?

2 CAN

2.1 Begriffsklärung

Der Begriff Content Adressable Network (CAN) steht für ein Design, mit dem verteilte Hashtabellen im Internetgrößenformat realisiert werden können. Elementare Methoden wie das Suchen und Speichern von Daten bei einem gegebenen Schlüsselwort werden durch ein solches System implementiert.

2.2 Design

2.2.1 Overlaynetz

Das Overlaynetz benutzt einen *d-dimensionalen Torus*, also ein d-dimensionales Koordinatensystem, das toroid verbunden ist. Jeder Knoten ist in diesem Netz eindeutig über seine Koordinate definiert. Ein Schlüssel wird von einer uniformen Hashfunktion auf einen Punkt in diesem d-Torus abgebildet und definiert so den Speicherort für das ihm zugehörige Datum. Auf diese Art und Weise realisiert das CAN eine verteilte Hashtabelle. Der virtuelle Raum wird von den CAN Knoten partitioniert, so ist jeder CAN Knoten ist für einen Teil des d-Toruses zuständig, dieser Teil wird als *Zone* des Knotens bezeichnet. Objekte werden bei dem Knoten gespeichert, dessen Zone den dem Objekt zugeordneten Punkt beinhaltet. Aus gründen der Lastverteilung und der Fehlertoleranz speichern alle unmittelbaren Nachbarn des Knoten (Knoten mit d-1 dimensionaler Berührungsfläche, einer an jeder Begrenzungsfläche des Knotens) ebenfalls die Zone des Knotens. Das Besitzerrecht an der Zone hält jedoch der ursprüngliche Knoten.

2.2.2 Routing

Jeder Knoten verfügt über eine *Routingtabelle*, die Tupel (IP, virtuelle Koordinatenzone) speichert. In der Tabelle gibt es einen Eintrag für jeden Nachbar (Nachbarknoten wie beschrieben in → 2.2.1), dies beschränkt die Tabellengröße auf maximal $2 \cdot d$ (dies sichert uns den konstanten Speicherbedarf, denn die Dimension des Toruses verändert sich nicht).
Die Knoten routen Insert-, Delete- und Lookup-Nachrichten, sowie diverse Steuernachrichten. Dabei routen sie zunächst mit einem *greedy Algorithmus*, d.h. zu dem Knoten, dessen euklidscher Abstand der Kleinste zum Zielpunkt ist. Ist dieser Knoten ausgefallen, so wird versucht zu den anderen in der Richtung gelegenen Nachbarn (sortiert nach steigendem euklidischem Abstand) zu routen, bei Ausfall aller dieser Knoten wird als alternative Wegewahl eine *expandierende Ringsuche* angewandt.
Die durchschnittliche Pfadlänge beträgt $\frac{d}{4} \cdot n^{\frac{1}{d}}$. Dies läßt sich leicht aus der Formel für den Durchmesser des d-Kubus herleiten.

2.2.3 Knotenbeitritt

1. Anfangs muss ein neu eintretender Knoten über einen Bootstrap Mechanismus (*YOID* →2.2.4) Zugang zu einem CAN Knoten finden.

2. Der neue Knoten wählt einen zufälligen Punkt im virtuellen Raum.

3. Mittels CAN Routing wird eine join Anfrage zu dem zuständigen Knoten geroutet.

4. Die Zielzone wird halbiert, eine Hälfte behält der alte Knoten die andere der Neue. Die Wahl der Dimension bei der Teilung geschieht über Ordnung der Dimensionen der Größe des alten Knotens. Im 2-d Raum wird erst nach der x-Achse, dann y-Achse usw. geteilt, dies ermöglicht bei Knotenaustritt eine leichtere Zonenzusammenführung.

5. Der neue Knoten kopiert die Nachbarliste des Alten.

6. Die Nachbarn werden über die neue Zonenaufteilung benachrichtigt (diese halbieren deren lokale Sekundärkopie der Zone).

7. In periodischem Abstand werden update Nachrichten von den Zonenbesitzern an die Nachbarn gesendet.

Wir sehen, daß die Hauptaktivität nur in der unmittelbaren Nachbarschaft des Zielknoten (bei den $2 \cdot d$ Nachbarknoten) stattfindet, d.h. der Knotenbeitrittsalgorithmus skaliert gut, da nur konstant viele Knoten am Zonenteilungsmechanismus beteiligt sind.

2.2.4 Bootstrap

Angenommen, einem CAN ist ein DNS Name zugeordnet, dann kann der Knoten dort eine Liste von CAN Bootstrapknoten erreichen, über die er seine Join-Anfrage schicken kann. Als mögliche Implementierung dieser Bootstrapdienste könnten *Yallcast und YOID* [6] verwendet werden.

2.2.5 Knotenaustritt

CAN unterstützt zwei Arten des Knotenaustritts, einen sauberen Handovermodus sowie eine Knotenausfallbehandlung.

1. **Handover:**
   Im Normalfall wird ein Knoten seinen Austritt bei seinen Nachbarn vorher ankündigen, in diesem Falle übernimmt dann einer der Nachbarn die Zone des austretenden Knoten (dies wird idealerweise der Knoten mit höchster Bandbreite oder niedrigstem Datenbestand sein), es findet eine Zonenverschmelzung statt. Bisher nicht erreichte Nachbarn werden der Nachbarschaftsliste des neu zuständigen Knoten aufgenommen und Sekundärkopien der Zonen werden angeglichen.

2. **Ausfall:**

   (a) Zonenbesitzer schicken in regelmäßigen Abständen *Update-Nachrichten* an ihre Nachbarn. Fällt diese Nachricht für längere Zeit aus, dann geht der Nachbarknoten von einem Ausfall dieses Knotens aus und startet bei sich einen *Takeover Timer*.

(b) Meldet sich der Knoten wieder, so wird der Timer abgebrochen.

(c) Ist der Timer abgelaufen dann schickt der Nachbarknoten an die anderen Nachbarknoten (CAN Routing) eine Takeover Nachricht.

(d) Empfängt einer der Nachbarknoten die Takeovernachricht dann bricht er den Timer ab, falls der andere ein kleineres Zonenvolumen verwaltet, oder er überstimmt diesen und er sendet wiederum an alle Nachbarn eine Takeovernachricht.

(e) Wird der Knoten nicht wieder überstimmt und sind mehr als die Hälfte der Nachbarknoten des Ausgefallenen erreichbar, so dehnt der Knoten seine Zone auf die des Ausgefallenen aus (Sekundärkopie wird zu einer Primärkopie).

(f) Bei größeren Ausfallszenarien nutzt der Knoten eine expandierende Ringsuche, um die anderen Nachbarn zu finden.

CAN verfügt über spezielle Zonenneuzuweisungsalgorithmen (CAN → [1]), die dem entstehenden Ungleichgewicht, das bei Zonenverschmelzung entsteht, entgegenwirkt. Ähnlich dem Knoteneintritt ist auch der Knotenaustritt ein lokaler Vorgang. Nur bei größeren Ausfallszenarien ist mit etwas mehr Aufwand zu rechnen.

2.3 Designoptimierungen

In einer Standard CAN Infrastruktur werden kürzeste Wege der Wege mit durchschnittlich $O(n^{\frac{1}{d}})$ Overlaynetz Hops erreicht. Es existiert jedoch keine Abbildung des Overlaynetzes auf das zugrunde liegende IP Netz. Ein CAN Hop kann daher sehr viele IP Level Hops beinhalten. Das Standarddesign hat also erhebliche Probleme mit der Latenz bei Sprüngen zwischen den CAN Knoten. Tapestry bietet die Möglichkeit Daten in mehreren Root Knoten zu speichern, es wäre sinnvoll ähnliche Erweiterungen in das CAN System zu integrieren. Diese Maßnahmen dienen der allgemein besseren Verfügbarkeit, wünschenswert wären zusätzlich ein Hotspot-Management, also Lastenverteilung bei besonders häufig gestellten Anfragen.

2.3.1 Multidimensionale Räume

Die einfachste Designoptimierung ist es an dem Dimensionsparameter zu schrauben. Overlaynetze höherer Dimension bieten zusätzliche Wege für das gesamte CAN Routing. Dadurch erhöht sich die Robustheit; ebenso wird der Hypertorus immer kompakter, weswegen die Anzahl der nötigen CAN Hops beim Routing sinkt. Der Dimension ist prinzipiell keine Grenze gesetzt allerdings erhöht sich der Aufwand linear mit dem Ansteigen der Dimension beim Austausch von Daten bei Knoten Ein- und Austritt, sowie der Routingtabellengröße. In Abbildung 1 sieht man das zweidimensionale Overlaynetze nicht skalieren. Ab dreidimensionalem Overlaynetz ist allerdings ebenso zu sehen, daß die durchschnittliche Anzahl der Hops stark zurückgeht und effizientes Routing auch für eine Million Knoten gewährleistet ist.

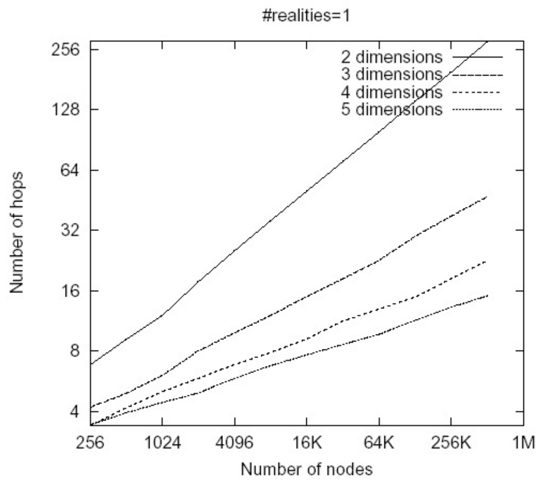


Abbildung 1: Effekt der Dimension auf die durchschnittliche Pfadlänge

2.3.2 Realities - Multiple Koordinatenräume

Nicht zu verwechseln mit der Erhöhung der Dimension eines Koordinatenraums ist die Bildung mehrer Koordinatenräume. Ein Knoten speichert nun für jeden Koordinatenraum eine primäre Zone und ebenso Nachbarzonen. Dies impliziert eine Replikation der Daten, da für jeden Koordinatenraum ein Knoten Besitzer der Zone ist. Sei r die Anzahl der Realities, so stehen einem Knoten nun r Schlüssel für die Formulierung einer Lookup-Anfrage zur Verfügung. Abbildung 2 zeigt, daß es bei einer Änderung der Anzahl der Realities ähnlich wie bei der Änderung der Anzahl der Dimensionen (Abbildung 1) bei niedrigen Werten zu sprunghaften Pfadlängenreduktionen kommt.
Denkbar ist auch das parallele Routing einer Anfrage zu allen Zielknoten aller Realities, bei zufälliger Verteilung der Knoten in den Overlaynetzen wäre der erwartete Nutzen groß, allerdings steigt auch die Netzlast proportional zu r.

2.3.3 Multidimensionale Räume ⇔ Realities

Analysen (CAN → [1]) haben gezeigt, daß die Multidimensionalität des Raumes zu kürzeren Pfadlängen führt als ein äquivalenter Fokus auf die Anzahl der Koordinatenräume. Allerdings haben Realities nützliche Nebeneffekte hinsichtlich Verfügbarkeit und Robustheit.

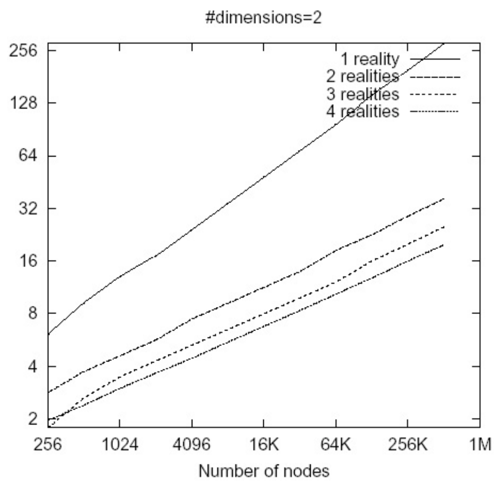


Abbildung 2: Effekt der Realities auf die durchschnittliche Pfadlänge

2.3.4 Bessere CAN Routing Mechanismen

Bisher orientierten wir uns einzig und allein an dem euklidischen Abstand im virtuellen Raum bei der Entscheidung bei welchem Knoten eine Nachricht weitergeroutet wird. Um besser auf die IP-Netz Topologie abzubilden, ist es nützlich die Round-Trip-Time RTT mit in die Entscheidungsmetrik mitaufzunehmen. Ein Tradeoff zwischen Nähe im virtuellen Raum und RTT ist nötig.
Metrik ist eine *Per-Hop-Latenz*, die als Quotient aus gesamter Pfadlatenz und Pfadlänge berechnet wird. Eine effektive Pfadlänge sollte genutzt werden, denn die Knoten liegen nicht auf einer Gerade zum Zielpunkt, denkbar ist Wahl der Pfadlänge als Differnz aus Eigen- und Nachbarabstand zum Zielpunkt mit euklidscher Metrik.
Bei steigernder Dimension des Hypertorus sind Synergieeffekte zu erwarten, denn die euklidischen Abstände differieren weniger als bei niedrigen Dimensionen, wohingegen uns linear mehr Knoten für die RTT gewichtete Berechnung der Kosten zur Verfügung steht.

Simulationen (CAN → [1]) mit 2^8 - 2^{18} Knoten und einer zufälligen Auswahl der Latenz zwischen den Knoten (1, 10, 100ms Verbindungen) haben ergeben, daß RTT gewichtetes Routing 24% (d=2) bis 40% (d=4) effizienter ist als ungewichtetes CAN Routing.

2.3.5 Zonenüberladung

Bisher war ein Knoten für eine Zone zuständig (oder bei mehreren Realities für eine Zone in dem entsprechenden Koordinatenraum). Dies führt jedoch

bei Knoten Ein- und Austritt zu erhöhtem Wartungsverkehr und macht das System fehleranfällig.
Die Idee der *Zonenüberladung* ist es mehreren Knoten die primäre Eigentümerschaft zu übertragen. Tritt ein neuer Knoten dem System bei, so wird er wie gehabt zu seinem Zielpunkt geroutet, dort erwartet ihn ein Konglomerat an Knoten, das für die Zone zuständig ist. Bis zu einer maximalen Anzahl von Knoten $MAXPEERS$ verschmilzt der Knoten mit dem Konglomerat, er kopiert sich die Routingtabellen von einem der alten Knoten und benachrichtigt verwandte und benachbarte Knoten über seine Existenz. Bei überschreiten von $MAXPEERS$ wird die Gruppe halbiert und die Nachbargruppen entsprechend informiert (Knotenbeitritt → 2.2.3).
Der Knotenaustritt ist unproblematisch so lange mehr als 1 Knoten noch in der Gruppe verweilt, sonst wird ein Handover vollzogen.
Ausserdem sind Performanceverbesserungen möglich, denn ein Knoten wählt bei beim Routing den Nachbarn aus der Nachbargruppe der die niedrigste RTT aufweist.
Der Nachteil ist das Zonenüberladung dem Verteilungsaspekt wiederspricht, Knoten müssen mehr Daten halten und über mehr Nachbarn Informationen speichern. In (CAN → [1]) konnte bei gleichem Simulationsaufbau (Bessere Routing Mechanismen → 2.3.4), eine Reduktion der Per-Hop Latenz um 45% erreicht werden.

2.3.6 Landmarkierungen

Eine Möglichkeit Verbindungen im Overlaynetz besser auf IP Ebene zu projezieren, ist die Konstruktion des CAN mittels Landmarkierungen. Die Knoten, die bei Initialisierung des Systems mitwirken, bestimmen die RTT zueinander und partitionieren den Torus so, daß die RTT zwischen den Knoten minimiert wird. Ein neuer Knoten erhält initial eine Liste von Knoten über das Bootstrapsystem, er wählt sich nun den Join-Zielpunkt nicht mehr zufällig, sondern nahe dem Knoten mit minimaler RTT aus dieser Liste. Lange Latenzzeiten bei Applikationsebene sind nun sehr unwahrscheinlich.

Bei gleichem Simulationsaufbau (Bessere Routing Mechanismen → 2.3.4) konnte beobachtet werden, daß in einem vierdimensionalen CAN Overlaynetz mit 4 Landmarkierungen, die jeweils mehr als 5 IP Hops auseinander liegen mußten ein konstanter IP/CAN Latenzfaktor nahe 1 bei variierender Knotenzahl gegeben ist.

2.3.7 Mehrere Hashfunktionen

Mehrere Hashfunktionen bilden den Schlüssel auf verschiedene Punkte im Koordinatenraum ab. Bei k Hashfunktionen entspricht dies einer k-fachen Replikation des Objekts zu den k Zielpunkten der Hashfunktionen. Im Un-

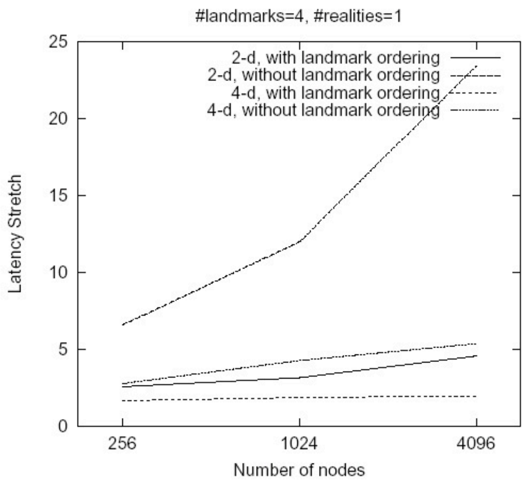


Abbildung 3: Latenzreduktion durch die Einbeziehung von Landmarkierungen während der CAN Konstruktionsphase.

terschied zu Realities ist es hier nicht nötig zusätzlich Nachbarn zu speichern, da sich die Knoten in nur einem Overlaynetz befinden. Denkbar ist es, wieder parallel an alle Zielknoten zu routen.

Abbildung 4 zeigt, daß man unter Verwendung von 5 Hashfunktionen bei zweidimensionalen Overlaynetzen etwa eine Halbierung der Latenz bei Abfragen erreichen kann.

2.3.8 Hotspotdaten

Bei häufigen Anfragen an eine bestimmte Zone haben wir bisher auf die Kapazität des Peers vertraut, Zonenüberladung und Realities haben nur eine beschränkte Wirksamkeit bei massiven Anfragen.
CAN Hotspot Management erweitert die Funktionalität bei Knoten, die auf den Routingwegen liegen. Jeder Knoten hält nunmehr seine Zone und in einem Cache temporär geroutete Abfragen. Knoten praktizieren ein Lookup-While-Routing, bei einem Cache Hit schickt der Knoten die Antwort ohne weiterzurouten (mit einer gewissen Wahrscheinlichkeit routet er nicht, um gelegentlich Updates zu erhalten). Knoten können aktiv ihren Nachbarn Daten in den Cache schieben, um auf eine Überlast aufmerksam zu machen. Wenn die Nachfrage wieder sinkt, veralten die Cache Werte und werden verworfen.

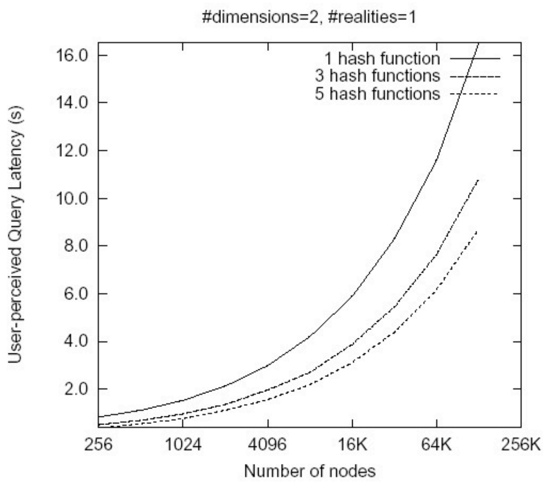


Abbildung 4: Reduktion der vom Benutzer wahrgenommenen Abfragelatenzzeiten unter der Verwendung mehrerer Hashfunktionen.

3 Bereichsabfragen CAN-basierter DHTs

Bereichsabfragen können mittels verschiedener Verfahren realisiert werden. Eine naive Herangehensweise ist jeden diskreten Wert im Abfragebereich einzeln abzufragen. Ene andere Möglichkeit ist es die Bereiche selbst mit in die Hashfunktion einzubeziehen → *Approximate Range Selection Queries in Peer-to-Peer Systems* [5]. Es muß dann dafür gesorgt werden, daß Knoten, die mit diesem Hashwert referenziert werden, die Anfrage mittels lokalem Cache beantworten können oder die Anfrage in mehreren kleinen Teilbereichen rekursiv propagieren. Problematisch ist in diesem Ansatz, daß ein Approximierungsmechanismus nötig ist, um die Vielzahl der möglichen Bereichsabfragen auf wenige Bereichsabfragen zu beschränken. Knoten die einen Bereich speichern sollten auch bei Anfragen ihren lokalen Cache benutzen können.
Im folgenden wird eine dritte Möglichkeit für Bereichsabfragen vorgestellt, die dieses Problem mit Lokalität erhaltenden Hashfunktionen löst.

3.1 Motivation

Die bisher vorgestellten Verfahren dienen der Optimierung der Suche nach einem Objekt. Wie aber kann man DHTs erweitern, um komplexe Informationsinfrastrukturen zu realisieren?
Motivation für *Efficient Range Queries for Grid Information Services* [2] ist eine P2P-basierte Umsetzung des Grid Index Information Service *GIIS* zu schaffen. GIIS gibt ein kohärentes Abbild von den im Grid verteilten Ressourcen und erlaubt es speziellen Ressourcen zu suchen. Eine Anfrage im

GIIS ist zum Beispiel die Suche des CONDOR Batchsystems nach Knoten für Ausführung eines Jobs mit folgenden Anforderungen:

Requirements = Arch == „SGI“ && OpSys == „IRIX6“ && Memory $\geq$ 256.

Knoten mit den Attributen Arch & OpSys können wir über traditionelle CAN Routing Mechanismen suchen, für Attribut Memory benötigen wir Bereichsabfragen, sonst müßten wir jeden diskreten Wert einzeln testen.

3.2 Design

Eine Eigenschaft der Hashfunktion im CAN Konzept [1] ist es, daß Schlüssel zufällig auf Punkte im Overlaynetz abgebildet werden. Wir wählen deshalb raumfüllende Kurven wie die Hilbertkurve, um eine Lokalität bei ähnlichen Schlüsseln zuzusichern. Knoten speichern wie gehabt Schlüssel-Werte Paare, jedoch propagieren sie die Schlüssel-Werte Paare in regelmäßigen Abständen zu einem Intervall Keeper (IK), der für Schlüssel-Werte Paare in einem festen Wertebereich zuständig ist. Ändert sich der Wert eines Attributes, so wird entsprechend das Datum bei einem anderen IK gespeichert. Es gibt, um auf das Beispiel zurückzukommen, in einem sehr groben Modell z.B. 4 IKs, die für die Bereiche 0 - 255, 256 - 374 Mb, 375 - 512, 513 - Max zuständig sind. Bei einer Anfrage wird wegen der Bedingung Memory $\geq$ 256 zu den drei letzten IK geroutet. Jeder IK sucht in seinem Cache nach Werten, die mit der Spezifikation übereinstimmen und routet diese an die Quelle der Bereichsabfrage. In der tatsächlichen Implementierung ist dies ein komplexerer Vorgang (Abfragen → 3.2.2).

3.2.1 Intervallabbildung auf Hilbertkurven

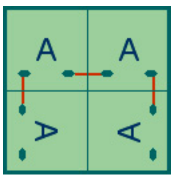
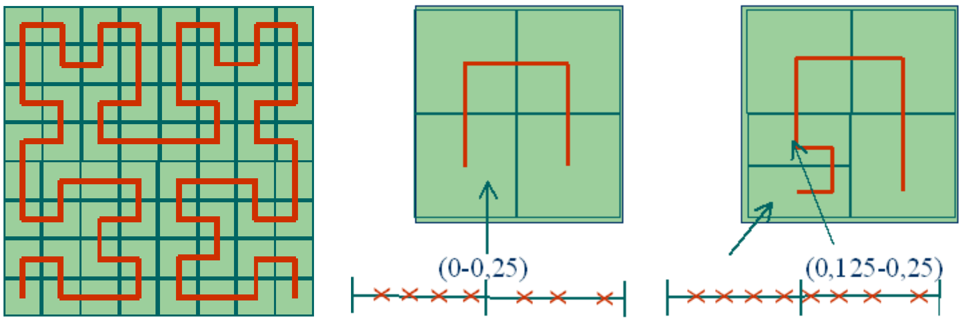


Abbildung 5: Hilbertkurven Konstruktion - Nächste Approximationsstufe

Die Hilbertkurve 2D und ähnliche Kurven in höheren Dimensionen [4], bilden Kurvensegmente auf Blöcke eines d-dimensionalen Raumes eineindeutig ab. Dies ist genau die Hashfunktion, die wir benötigen, denn ein Attributwert-Intervall (Hilbertkurvensegmente) läßt sich direkt auf Blöcke im Overlaynetz abbilden. Die IKs werden entlang der Kurvensegmente aufgereiht.



Hilbert Kurve Level 3 Intervallteilung bei Überlast

Abbildung 6: Weitere Hilbertkurven

Die Konstruktion der Hilbertkurve beginnt mit einer geviertelten Fläche. Die Kurve besucht jeden der Quadranten, vom Dritten über den Zweiten über den Ersten hin zum Vierten. Die Teilung läßt sich nun für jeden Quadranten beliebig weiterführen 3.2.1. Die Basiskurve (L1 Approximierung) wird in diesem Vorgang entsprechend gespiegelt und gedreht. Die Anordnung muß nicht notwendigerweise statisch sein, ist ein Attributwertebereich mit weniger Objekten vertreten, dann übernimmt ein IK einen größeren Hyperkubus im Overlaynetz und entsprechend ein größeren Wertebereich. In diesem Abschnitt wird ein niedrigeres Approximationslevel der Hilbertkurve gewählt. Finden sich in einem Attributwertebereich sehr viele Objekte, so wird ein höheres Approximationslevel der Hilbertkurve für diesen Bereich gewählt. Dieser Attributwertebereich wird nun feiner aufgeteilt.
Ein IK speichert eine Zone mit maximaler Anzahl von L Objekten, wird diese überschritten, so wird das Intervall (und die Zone) geteilt (Abbildung 3.2.1).

3.2.2 Abfrageimplementierung

Eine Bereichsabfrage wird durch Routen zu dem IK, der den Mittelpunkt $m = \frac{l+u}{2}$ des Intervalls der Abfrage beinhaltet, begonnen.

Es gibt 3 verschiedene Implementierungen von Bereichsabfragen.

1. **Brute Force**
   Ein Knoten berechnet den kleinsten Hyperkubus, der alle Zonen der IKs, die sich mit der Abfrage überschneiden, enthält. Dann flutet er die Anfrage an alle Knoten in dem Hyperkubus. Nachteil dieser Methode ist, daß sich manche Zonen einiger Knoten, die am Rand des Hyperkubus liegen, sich nicht mit dem Abfragebereich überschneiden.

2. **Kontrolliertes Fluten**
   Wir lassen den aktuellen Knoten die Nachricht nur weiterleiten, wenn die Nachbarknoten in dem Abfragebereich liegen.

3. **Gerichetes Kontrolliertes Fluten**
   Funktioniert wie Kontolliertes Fluten, nur propagieren wir die Anfrage in zwei Wellen. In der Ersten werden nur Knoten benachrichtigt, die ein höheres Intervall besitzen und in der zweiten Welle Knoten mit niedrigeren Intervallwerten. Dies dient der Reduzierung der Nachrichtenduplikation, die bei allen drei Implementierungen auftritt.

3.2.3 Updating Strategien

Serverknoten propagieren in regelmässigen Abständen ihre Attributwerte an den zuständigen Intervallkeeper. In einem naiven Ansatz könnten sie jede Update Nachricht mittels CAN Routing übertragen (ähnlich wie CAN-Insert Nachricht). Intervallkeeper speichern nun diese Objekte für einen festgelegten Zeitraum und löschen sie, wenn die Updatenachricht im nächsten Reportzyklus ausbleibt.
Vorteil dieser Methode ist die Entkopplung der Datenhaltung von dem Bereichsabfragemechanismus. Fällt ein IK aus, so übernimmt im nächsten Reportzyklus ein anderer IK den Wert des Serversknoten.
Allerdings wird jede Updatenachricht über das P2P Routing übertragen. Wenn sich der Attributwert beim Server nicht ändert, dann könnte die nächste Übertragung des Serverknoten an den IK auf IP Weg stattfinden. Vorausgesetzt der IK teilt sein Intervall nicht, fällt nicht aus und bestätigt dem Serverknoten das Update mit seiner IP Adresse.

3.3 Optimierte Updatestrategien

In den Updatemechanismus wird nun ein IP Caching miteinbezogen. IK, die eine Updatenachricht erhalten, senden dem Serverknoten ihre IP mit der Hoffnung auf einen gleichbleibenden Attributwert beim Serverknoten. Dieser hat nun zusätzlich einen Cache mit maximaler Cachegröße (s) in der die IKs, zu denen er schon einmal eine Updatenachricht gesendet hat, gespeichert sind. Liegt in der neuen Periode ein Attributwert vor dem ein bekannter IK zuordenbar ist, dann wird nun mittels IP Routing die Nachricht zu ihm geroutet. Fällt der Attributwert auf ein unbekanntes Intervall oder antwortet der IK bei IP Übertragung nicht, dann kann über CAN Routing die Nachricht übertragen werden.

So elegant wie diese Methode ist, so weniger effizient wird sie, wenn die Hashtabelle viele ähnliche Werte speichert und sich die Anzahl der IKs in diesem Bereich stark ändern. Knoten würden nun stetig zu IKs, deren Wertebereich fast mit dem des alten IK übereinstimmt, über CAN Routing Updates senden. Wenn es auf exakte Werte nicht ankommt, dann kann der Serverknoten versuchen bei jedem gecachten IK ein Toleranzwert t vom Intervallbegin abzuziehen, bzw. Intervallende hinzuzuaddieren. Es besteht nun die Möglichkeit einem der IKs ein Objekt mit leicht ausserhalb liegenden Attributwert unterzuschieben und weiterhin performant mit IP Updatenachrichten zu kommunizieren.

Da CAN Routing seine garantierte Routingperformance nur bei gleichgroßen Zonen erhält, ist es notwendig auf eine Lastenverteilung zu achten. Eine bessere Lastenverteilung läßt sich mit Attributwertrandomisierung erreichen. Der Wert des Attributes wird vom Server randomisiert, wenn er diesen über CAN Routing übertragen muss. Dazu wird ein Intervall [-r,r] (r ist Randomisierungslevel) bei den niederwertigesten Stellen des Attributwertes hinzuaddiert. Updateinformationen werden nun bei einem IK in der Nähe des eigentlichen IK gespeichert. Die Chance in ein kleines Intervall mit dieser Methode zu treffen ist geringer, deswegen sind Intervalle gleichmäßiger verteilt.

3.4 Analyse

Im folgenden Abschnitt werden Routing- und Updatestrategien an drei verschiedenen Attributwertdichteverteilungen (Abbildung 7) analysiert. Uniforme Verteilung, Pareto Verteilung (a=100) und einem Data-Center mit 41 Serverknoten, das 34 Tage lang die CPU Auslastung überwacht hat.

Der Vergleich (Abbildung 8) bezieht sich auf das Routen zwischen zwei Knoten, bei einem Startwert mit 1 IK pro Parameter, L=20 und insgesamt 10*N Wiederholungen. Zu beobachten ist eine leicht höhere Anzahl der Hops bei der Paretoverteilung, zurückzuführen ist dies auf den höheren Routingaufwand bei ungleichen Intervallgrößen. Allerdings wird dieser Effekt in CAN Modellen höherer Dimension weiter gedämpft.

Abbildung 9 zeigt Charts für die durchschnittliche Pfadlänge, die Anzahl der Nachrichten und die Anzahl der Knotenbesuche für einen Versuchsaufbau mit 1000 IKs
Die Werte werden dargestellt in Abhängigkeit zur Größe des Abfrageintervalls (gemittelt über 10 Bereichsabfragen zufällig aus dem Bereich 0.002 - 0.6).
Gerichtetes und ungerichtetes kontrolliertes Fluten sind spürbar effektiver als Brute Force Fluten, besonders bemerkbar ist dies bei großen Berichsabfragen, da hier viele nicht an der Abfrage beteiligten Knoten kontaktiert werden.
Allerdings ist kaum ein Unterschied zwischen den beiden Kontrollierten Fluten Algorithmen bei d=2 zu sehen.

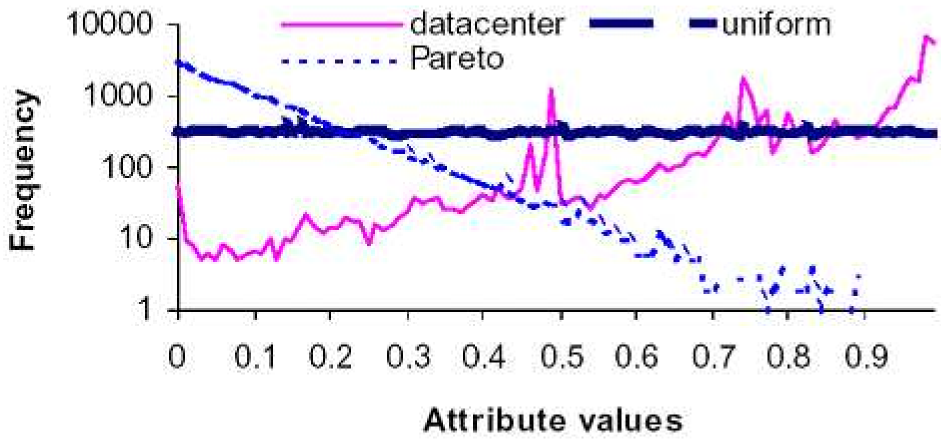


Abbildung 7: Dichteverteilung während des Experiments

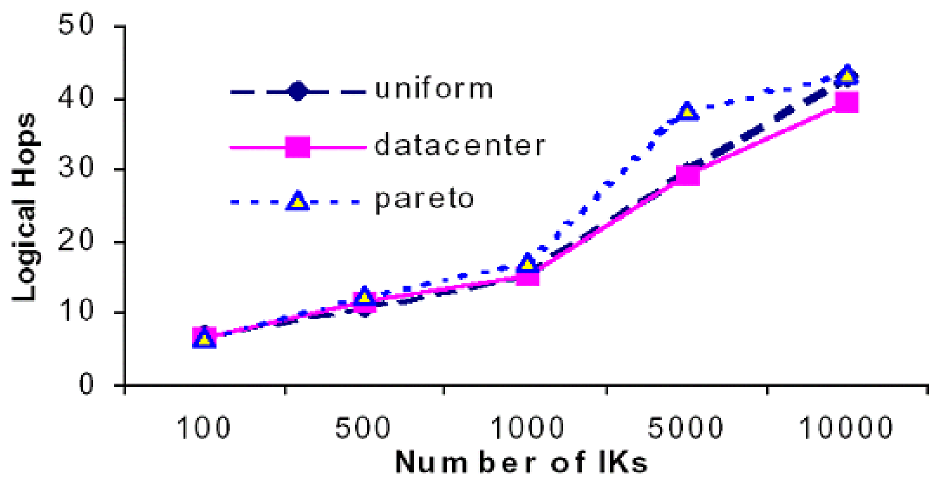
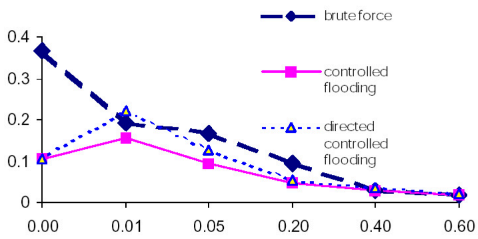
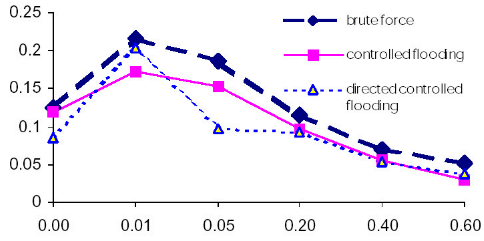


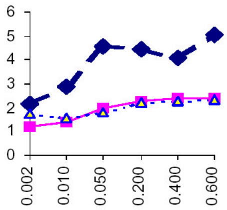
Abbildung 8: Vergleich der durchschnittlichen logischen Routingperformance (d=2)



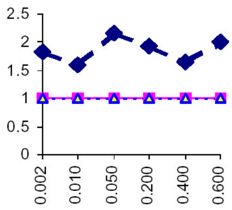
(a) Average path length



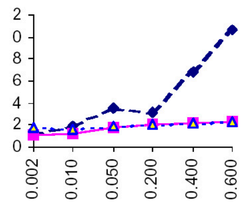
(a) Average path length



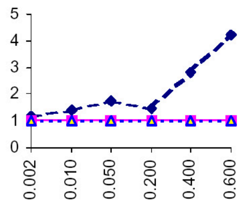
(b) Number of messages



(c) Nodes visited



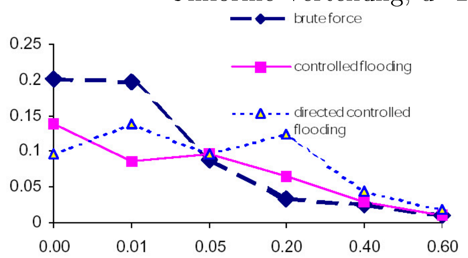
(b) Number of messages



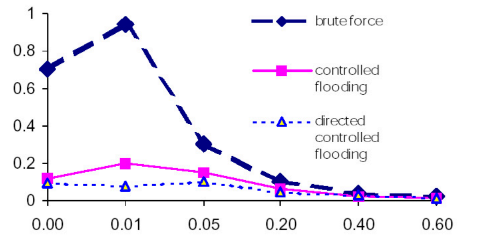
(c) Nodes visited

Uniforme Verteilung, d=2

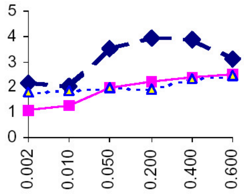
Data Center Verteilung, d=2



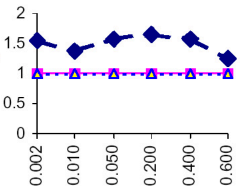
(a) Average path length



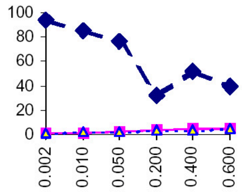
(a) Average path length



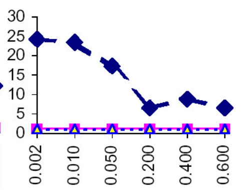
(b) Number of messages



(c) Nodes visited



(b) Number of messages



(c) Nodes visited

Pareto Verteilung, d=2

Data Center Verteilung, d=4

Abbildung 9: Vergleich der Attributwertverteilungen

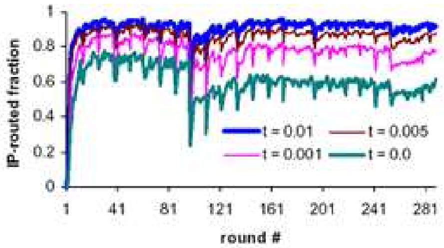


Figure 9: Example how the changes in update patterns affect the routing efficiency

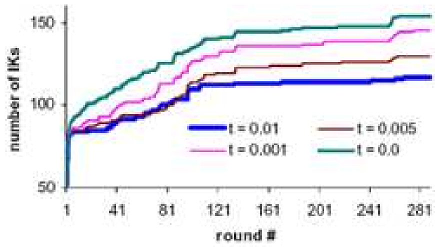


Figure 11: Effect of the tolerance level t on the number of IK

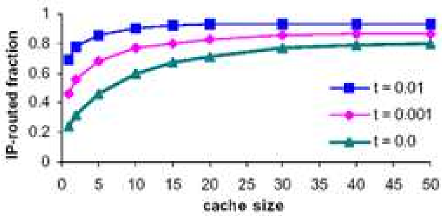


Figure 10: Influence of cache size and tolerance level t on the fraction on IP-routed updates

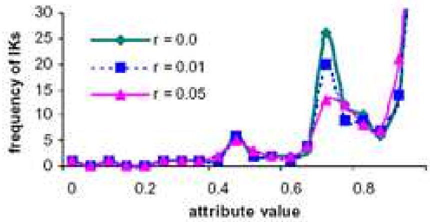


Figure 12: Effect of the randomization level r on the distribution of IKs density

In Figure 9 kann man in der 97. Runde Einbrüche des IP gerouteten Update-Verkehrs sehen, die auf ein neues Updatemuster beim Melden der Attributwerte zurückzuführen sind. Hervorgerufen wird dies durch Intervallneustrukturierungen der Intervallkeeper und IK-Neubildung.

In Figure 10 ist zu beobachten, daß Cachesize und der Toleranzfaktor in Kombination zu einem beachtlichen IP gerouteten Prozentsatz bei den Updatenachrichten führen. Eine Größe von s=10 und t=0.001 führt schon zu 75% IP gerouteten Verkehr.

In Figure 11 ist zu sehen, daß schon kleine Toleranzniveaus von 1% zu einer Drittelung der Anzahl von Intervallkeepern führen kann, dies spart eine Menge Routingoverhead.

In Figure 12 erkennt man, daß die Randomisierung nur begrenzt nützlich ist. Nur bei dem Peakwert von 0.7, wo die meisten Werte gespeichert sind nutzt es wirklich.

4 Literaturangaben

Literatur

[1] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker: A Scalable Content Addressable Network, Proc. ACM SIGCOMM 2001, August 2001. http://citeseer.nj.nec.com/ratnasamy01scalable.html

[2] Artur Andrzejak, Zhichen Xu: Scalable, Efficient Range Queries for Grid Information Services, Proc. Second IEEE International Conference on Peer-to-Peer Computing (P2P2002)

[3] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, „Grid Information Services for Distributed Resource Sharing“, 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, 2001.

[4] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier, „Space Filling Curves and Their Use in Geometric Data Structures“, Theoretical Computer Science, 181, 1997, pp. 3-15.

[5] Abhishek Gupta, Divyakant Agrawal, Amr El Abbadi „Approximate Range Selection Queries in Peer-to-Peer Systems“ University of California Santa Barbara

[6] http://www.icir.org/yoid/docs/index.html