Effizientes Routing in verteilten skalierbaren Datenstrukturen

Erik Buchmann, Klemens Böhm
{buchmann|kboehm}@iti.cs.uni-magdeburg.de
Arbeitsgruppe *Data and Knowledge Engineering*
Otto-von-Guericke Universität, Magdeburg

Abstract: Verteilte skalierbare Datenstrukturen (SDDS) besitzen große Bedeutung, insbesondere als Grundlage der Realisierung von innovativen Web-Diensten. Die Knoten einer SDDS verwalten (Schlüssel, Wert)-Paare sowie Kontaktinformation über andere Knoten. Diese Kontaktinformationen werden für das Routing von Nachrichten zwischen den SDDS-Knoten benötigt. Dieser Artikel untersucht, wie sich das Caching von Kontaktinformation und die Auswahl der Schlüsselabbildung, d.h. der Abbildung der Datenobjekte auf den Schlüsselraum der SDDS, auf das Routing auswirkt. Unser Hauptergebnis ist die Erkenntnis, dass Caching insbesondere in Verbindung mit einer nachbarschaftserhaltenden Schlüsselabbildung vorteilhaft ist.

1 Einführung

Verteilte skalierbare Datenstrukturen (SDDS) [LNS96, KW94] dienen zur verteilten, selbstorganisierenden Verwaltung von Dictionary-Datentypen, also von Mengen aus (Schlüssel, Wert)-Paaren. Wir sehen viele Anwendungen insbesondere im Web-Kontext, beispielsweise verteiltes Web-Caching [IRD02], Annotations- [RMW94] oder Backlink-Services [CGM99]. Unser neues Projekt *Fairnet* zielt auf die Weiterentwicklung von SDDS ab. Wir erhoffen uns mittelfristig Antworten auf die folgenden Fragen: Wie sieht eine SDDS-Realisierung aus, die Caching, Lastbalancierung, Fehlertoleranz etc. beinhaltet? Wie sehen Self-Tuning Mechanismen in diesem Kontext aus? Was ändert sich, wenn sich einzelne Knoten unkooperativ verhalten? Dieser Artikel beschreibt nun einige Voruntersuchungen, die wir mit einem von uns entwickelten Prototypen durchgeführt haben, um die Leistungseigenschaften von SDDS besser zu verstehen und ihre Performanz zu steigern.[1]

Mit SDDS verwaltet jeder Knoten eine Menge von Schlüsseln. Er leitet eine Operation – z.B. eine Anfrage oder eine Einfüge-Operation – die er nicht selbst bearbeiten kann, an einen aus seiner Sicht geeigneteren Knoten weiter. Jeder Knoten besitzt daher eine *Kontaktliste* in Form einer Menge von Knoten zusammen mit der möglicherweise veralteten Information, welchen Teil des Schlüsselraumes jeder dieser Knoten verwaltet.

[1] Die durchgeführten Experimente sind teilweise vorläufig, da unser Cluster zur Simulation eines großen Network of Workstations noch im Aufbau begriffen ist, wir daher nur mit relativ kleinen Datenmengen experimentieren konnten. Unser experimenteller Aufbau ist aber in Größenordnungen durchaus vergleichbar mit dem in [LNS96, Abe01].

Dieser Artikel untersucht nun, wie einfache Maßnahmen, die die Lokalität in Zugriffsmustern besser berücksichtigen als bisherige SDDS-Implementierungen, zu einer verbesserten Performance führen. Wir konzentrieren uns zwei Aspekte: (1) Auswahl der Knoten in den Kontaktlisten, (2) Einfluß der Abbildung der zu verwaltenden Datenobjekte auf den Raum der möglichen Schlüssel. Unser Kostenmaß ist ebenso wie in [LNS96, KW94] die Anzahl der Message Hops; die Topologie des Netzwerks bleibt außen vor. Hinsichtlich (1) lassen sich existierende SDDS-Implementierungen in zwei Kategorien aufteilen. In der ersten Kategorie (z.B. CAN, P-Grid) ist fest vorgegeben, welche Knoten in der Kontaktliste enthalten sind; das Anfragemuster bleibt unberücksichtigt. Im zweiten Fall (z.B. [KW94]) ist die Länge der Kontaktliste nicht begrenzt. Das ist insbesondere dann unrealistisch, wenn Knoten nur einen kleinen Teil ihrer Ressourcen einbringen wollen. Wir untersuchen, inwieweit existierende Caching-Strategien für die Verwaltung von Kontaktlisten von SDDS geeignet sind, und wir entwickeln eine neue Strategie speziell für diesen Kontext. Es zeigt sich allerdings, dass ausgefeilte Caching-Strategien hier keinen deutlichen Vorteil bringen. Bezüglich (2) haben wir beobachtet, dass bisherige Abbildungen von Datenobjekten in den Schlüsselraum darauf abzielen, den Wertebereich möglichst gleichmäßig auszunutzen. In unserem Kontext ist es dagegen wichtig – und der Artikel wird dies erklären und experimentell nachweisen –, ähnliche Datenobjekte auf ähnliche Schlüssel abzubilden.

2 Content-Adressable Networks

Content-Addressable Networks (CAN) [RFH+01] sind eine Ausprägung von SDDS, bei der jeder Knoten ein multidimensionales Schlüsselintervall verwaltet. Abbildung 1 zeigt beispielhaft den Aufbau eines CAN aus 14 Knoten und die Zuordnung von Ausschnitten des Schlüsselraums zu Knoten. Dies ist unabhängig von der Netzwerktopologie; das CAN etabliert ein virtuelles overlay-Netzwerk. Jeder CAN-Knoten kennt seine Nachbar-Knoten und weiß, welchen Ausschnitt des Raums sie verwalten. In Abbildung 1 hält Knoten 8 beispielsweise Informationen über die Knoten 0, 3, 4, 7 und 9. Die Schlüssel der SDDS sind das Ergebnis der Abbildung von anwendungsspezifischen Datenobjekten auf den SDDS-Schlüsselraum, der *Schlüsselabbildung*. Beispielsweise bildet eine Hash-Funktion URI auf Vektoren von Integer-Werten ab, gegeben den Schlüsselraum $\mathbf{Z}^n$.

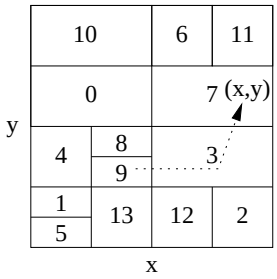

Abbildung 1: Schlüsselraum eines CAN und Beispiel-Anfrage.

Jeder CAN-Knoten kann Operationen wie Einfügen oder Anfragen absetzen. Jede SDDS-Operation spezifiziert dabei einen Zielpunkt im SDDS-Schlüsselraum. Eine Operation löst mindestens eine Nachricht aus, die häufig erst über mehrere Knoten (Hops) zum Ziel gelangt. Das Weiterleiten dieser Nachrichten vom Ausgangsknoten der Operation zum Zielpunkt, der möglicherweise zunächst unbekannt ist, wird allgemein als *Routing* bezeichnet. Abbildung 1 zeigt eine Beispiel-Anfrage des Knotens 9 zum Ziel (x,y) mit zwei Hops.

Caching von Kontaktinformationen in CAN. Im ursprünglichen Entwurf [RFH+01] führt eine Operation in einem d-dimensionalen CAN mit n Knoten zu $O(n^{1/d})$ Message Hops. Bei kleinem d ist diese Zahl zu groß. Bei großem d dagegen ist die Anzahl der zu verwaltenden Nachbarn sehr hoch, die dann bei Änderungen umgehend informiert werden müssen. Um die Anzahl der Nachrichten zu verringern, untersuchen wir im folgenden den Fall, dass die Kontaktliste eines CAN-Knotens nicht nur dessen Nachbarn, sondern zusätzlich weitere Knoten enthält, die der Knoten beim Routing mit einbezieht. Es wird also ein Cache mit limitierter Größe für Kontaktinformation angelegt. Existierende SDDS verfügen zwar über einen Cache, allerdings ist seine Größe entweder nicht begrenzt (vgl. z.B. [KW94]), oder der Cache enthält unabhängig vom Anfragemuster [Abe01] eine fest vorgegebene Auswahl von Knoten. Da beide Fälle nicht optimal sind, wollen wir in diesem Artikel Caching-Strategien und Eigenschaften des Caches untersuchen. Wir orientieren uns dabei an existierenden SDDS-Implementierungen, indem wir veraltete Kontaktinformationen im Cache zulassen, und den Versand zusätzlicher Nachrichten für Aufbau und Verwaltung des Caches vermeiden.

Auswirkungen des Kontaktinformationen-Cachings in CAN. Wir untersuchen in einem ersten Schritt, ob das Caching von Kontaktinfomationen überhaupt sinnvoll ist, und quantifizieren den Einfluß der Cache-Größe auf das Routing-Verhalten. Die Cache-Größe wurde in sechs Schritten von 'keinem Cache' auf 'unbegrenzte Cache-Größe' gesteigert. Als Verdrängungsstrategie wurde Random gewählt. Mit *Random* entscheidet ein gleichverteilter Zufallszahlenstrom über die zu verdrängenden Einträge. Das Zugriffsmuster bleibt unberücksichtigt.

Die Ablaufumgebung ist ein CAN von 1.000 Knoten, in dem 50.000 gleichverteilte Abfragen über gleichverteilte Daten abgesetzt werden. Diese Gleichverteilungen sind der für den Cache ungünstigste Fall, da keine Cache-Einträge bevorzugt nachgefragt werden.

Abbildung 2 visualisiert die Ergebnisse dieses Experiments. Die Darstellung ist durch Durchschnittswertbildung über ein Intervall von jeweils 1.000 Operationen geglättet. Wenig überraschend ist der fallende Grenznutzen der Cache-Größe. Der Graph der unbegrenzten Cache-Größe überschneidet sich bereits vollständig mit dem Graph für 500 Einträge. Ebenfalls zu erwarten war die verlängerte Zeitspanne bis zum Erreichen eines stabilen Zustands bei größeren Caches, da ein kleiner Cache schneller gefüllt ist.

Beachtenswert sind jedoch zwei Aspekte: Zum Einen ist auch ein kleiner Cache bereits sehr wirksam. Schon eine Cache-Größe von 2% der Gesamtzahl der Knoten reduziert die Zahl der Hops um $1/3$. Zum Anderen tritt diese Verbesserung schon nach wenigen ausgeführten Operationen pro Knoten ein. Auch bei einer kurzen Verweildauer der Knoten im

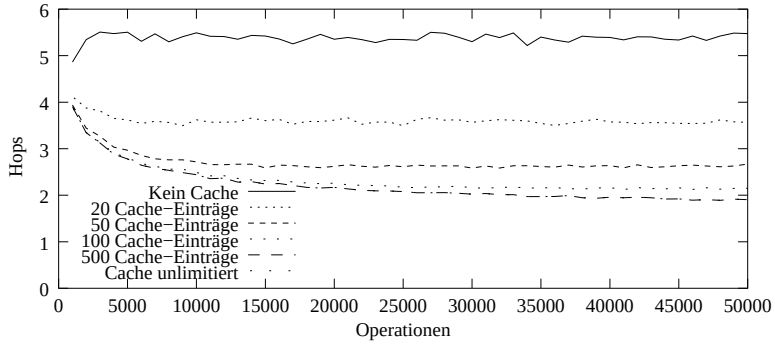

Abbildung 2: Zusammenhang von Cache-Größe zur Anzahl der Message-Hops.

Netz sind also signifikante Verbesserungen zu verzeichnen.

3 Verdrängungsstrategien und lokalitätserhaltende Abbildungen

Nachdem der Nutzen eines Caches für Kontaktinformationen grundsätzlich gezeigt wurde, versuchen wir im Folgenden, dessen Effizienz zu steigern. Naheliegend ist dabei der Einsatz einer besseren Verdrängungsstrategie. Dazu vergleichen wir *Least Frequently Used* (LFU), *Least Recently Used* (LRU), *Least Reference Density* (LRD)[2] mit Random als Referenzstrategie und der speziell auf Routing zugeschnittenen Strategie ISD.

Unsere neue Strategie ISD *(Inverse Square Distribution)* basiert auf einem gleichnamigen Vorschlag zum Aufbau der Weltsicht von Knoten in verteilten Datenstrukturen [Kle00]. Dabei steht p für die Wahrscheinlichkeit zur Aufnahme eines Knotens v in den Cache eines Knotens u. Für die Distanz zwischen zwei Knoten u, v in der Metrik des Schlüsselraums verwenden wir die Notation $d(u, v)$. Der Wert r ist eine Konstante. Es gilt jetzt, dass $p \sim [d(u, v)]^{-r}$. Diese Strategie berücksichtigt das Anfragemuster also nicht. Für $r = d$ mit d als der Dimensionalität des Schlüsselraumes hat Kleinberg unter bestimmten Annahmen gezeigt [Kle00], dass ein dezentraler Algorithmus Pfade von $O(\log n)$ Länge konstruieren kann, wenn die Weltsicht eines Knotens zu o.g. Formel konform ist. Es muss aber noch untersucht werden, ob jene Formel auch zu einer sinnvollen Verdrängungsstrategie führt.

Auswahl der Abbildung auf Schlüsselwerte. Die uns bekannten Schlüsselabbildungen zielen darauf, die zu verwaltenden Daten gleichmäßig über den Schlüsselraum zu verteilen. Ein Ziel dabei ist die gleichmäßige Auslastung der Knoten. In vielen Fällen greifen aufeinanderfolgende Operationen aber auf benachbarte Datenbereiche zu. Die Erhal-

[2]Die *Least Reference Density* (LRD)-Strategie verwendet als Maß für den Nutzen der Cache-Position i die Referenzdichte d_i. Diese wird nach der Gleichung $d_i = \frac{r_i}{g - e_i}$ ermittelt, indem der Referenzzähler r_i durch die Anzahl aller seit dem Einlagerungspunkt e_i des Cache-Elements i registrierten Referenzen g geteilt wird. Damit berücksichtigt diese Strategie zugleich das Alter und die Anzahl der Referenzen eines Eintrags.

tung dieser Lokalität im Schlüsselraum in Verbindung mit Caching von Kontaktdaten führt möglicherweise zu einer deutlichen Verbesserung des Anfrageverhaltens. Wir haben einen Algorithmus erstellt, der Zeichenketten wie URL in einen mehrdimensionalen Schlüsselraum abbildet und diese Lokalität erhalten soll (Abbildung 3). Damit lassen sich Testdaten beispielsweise aus Web-Logs generieren. Der Algorithmus summiert die Zeichencodes eines Strings abwechselnd für jede Dimension auf. Ein mit der Länge der Zeichenkette abnehmender Wichtungsfaktor sorgt dafür, dass Zeichen am Anfang stärker in den resultierenden Schlüsselwert eingehen als die Zeichen am Ende. Der Algorithmus gibt ein Array mit einem n-dimensionalen Punkt – hier im Intervall der Integer-Ganzzahlen – zurück.

```
function {$line} {
  set $faktor = 1000000
  while {[stringlength $line]>0} {
    foreach $i in $dimension {
      set $val($i) = [firstchar $line]*$faktor + $val($i)
      set $line = removefirstchar $line }
    set $faktor = $faktor / 2 }
  return $val() }
```

Abbildung 3: Lokalitätserhaltende Abbildung von Strings in den Schlüsselraum des CAN.

Abbildung 4 stellt exemplarisch den Weg eines Web-Surfers im Schlüsselraum eines CAN dar. Die verwendeten URL sind auf der rechten Seite der Abbildung zu sehen. Das Mapping der URL in den Schlüsselraum erfolgte zum einen gleichverteilt mit einer CRC32-Prüfsumme und zum anderen mit dem Algorithmus gemäß Abbildung 3. Im zweiten Fall werden nur zwei Anfragen langwierig zum Zielknoten geroutet, alle anderen betreffen entweder den letzten Zielknoten oder einen seiner unmittelbaren Nachbarn. Sektion 4 wird quantifizieren, wie sehr die Zahl der Hops durch die Kombination aus lokalitätserhaltender Abbildung und Caching zurückgeht.

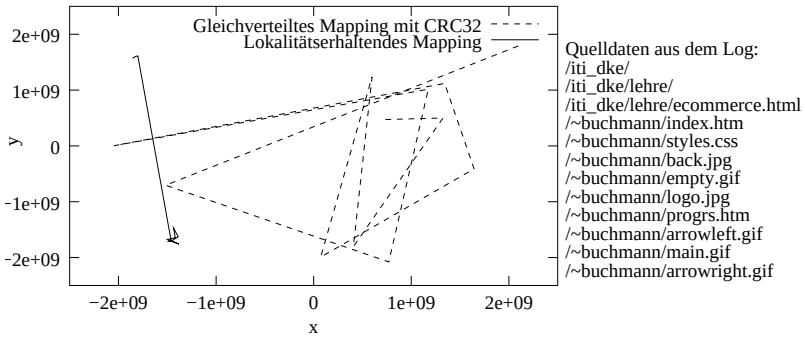

Abbildung 4: Darstellung des Pfades eines Surfers im Schlüsselraum des CAN.

Aktion	Wahrsch.	Abbildung auf das CAN
Link anwählen	52%	Mit 15% Wahrscheinlichkeit wird ein zur letzten Abfrage benachbarter Punkt generiert, mit 85% Wahrscheinlichkeit ein entfernter Punkt.
Back-Button	36%	Es wird der zuvorletzt abgefragte Punkt in der Historie erneut abgefragt.
History oder Bookmark	5%	Ein Punkt aus der Menge der bereits abgefragten wird mit Gleichverteilung zufällig gewählt und erneut abgerufen.
URL eingeben	3%	Ein entfernter Punkt wird zufällig generiert.
Home-Button	2%	Der erste Punkt der Surf-Sitzung wird erneut abgefragt und die Sitzungshistorie gelöscht.
Reload	2%	Es wird der zuletzt abgefragte Punkt in der Historie erneut abgefragt.

Tabelle 1: Verhalten des künstlichen Surfers

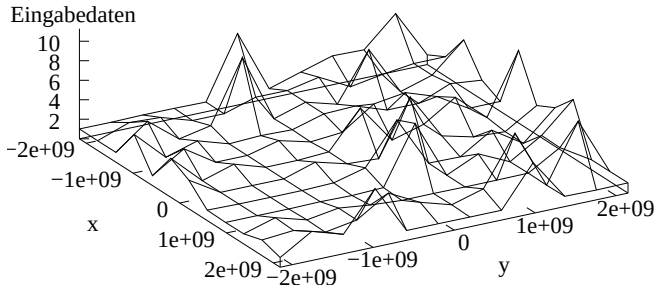

Abbildung 5: Verteilung der künstlich generierten Anfragepunkte im zweidimensionalen Schlüsselraum.

4 Experimente

Experimenteller Aufbau. Es kommen drei Testdatensätze zur Anwendung. Der erste Datensatz enthält gleichverteilte Anfragepunkte und dient als Referenz. Diese Verteilung ist natürlich unrealistisch. Der zweite Datensatz simuliert das Nutzerverhalten von vielen unterschiedlichen Websurfern. Kern der Testdaten-Generierung ist ein künstlicher Surfer zur Erzeugung statistisch relevanter Daten, der das in [TG97] beschriebene Verhalten zeigt. Zunächst wird dabei eine Liste mit über den Schlüsselraum gleichverteilten Startpunkten – entsprechend den Servern im Web – erzeugt. Der nächste Schritt weist jedem Knoten mit Hilfe einer normalverteilten[3] Zufallsfunktion einen dieser Punkte als Startwert zu. Daraufhin werden die Punkte, die jeder einzelne Surfer abfragt, wie in Tabelle 1 dargestellt, sukzessive ermittelt. Abbildung 5 zeigt die Verteilung der Anfragepunkte für ein zweidimensionales CAN. Zu erkennen sind mehrere 'beliebte' Bereiche mit

[3]Da auch im WWW einige Seiten erfolgreicher sind und häufiger abgefragt werden als andere.

hoher Referenzdichte, zahlreiche mäßig und einige schwach nachgefragte Bereiche im Schlüsselraum. Der dritte Datensatz schließlich bildet die Einträge aus dem Access-Log unseres Instituts-Webservers mit einem Algorithmus nach Abbildung 3 auf Testdaten ab. Die Verweildauer und die Zahl der abgesetzten Anfragen der Teilnehmer sind – wahrscheinlich in Folge der unterschiedlichen Interessen von Studenten und Mitarbeitern – verglichen mit dem zweiten Testdatensatz sehr inhomogen: 2.5% aller Teilnehmer generieren ca. 20% aller Anfragen. Als Simulationsumgebung dient wiederum ein CAN mit 1.000 Knoten, in dem die Knoten insgesamt 50.000 Abfragen absetzen.

Auswirkungen der Anfragelokalität. Nachdem Abschnitt 2 die Vorteilhaftigkeit von Caching in unserem Kontext bereits demonstriert hat, soll nun der Nutzen der Lokalitätserhaltung quantifiziert werden. Wir haben Experimente mit unterschiedlichen Cache-Größen durchgeführt, beschränken uns in der Darstellung aber auf die Cache-Größe von 20 Einträgen. Es wird sich herausstellen, dass auch bei sehr kleinen Caches eine lokalitätserhaltende Schlüsselabbildung die Zahl der Hops signifikant verringert. Ohne Caching jedoch würde der Erhalt der Lokalität keinen Nutzen bieten, da auch in diesem Falle stets alle Nachrichten von Nachbar zu Nachbar weitergereicht werden.

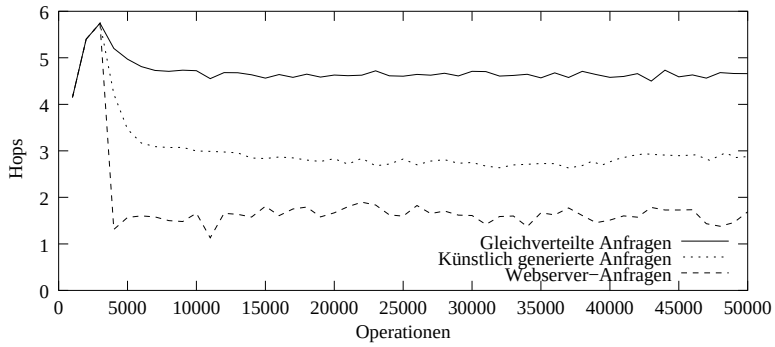

Abbildung 6: Zusammenhang von der Datenverteilung zur Anzahl der Message-Hops.

Abbildung 6 zeigt, dass sich die Effektivität des Caches durch Erhalt der Anfragelokalität erheblich steigern lässt. Diese Steigerung ist jedoch letztendlich trotz unseres recht ausgefeilten experimentellen Aufbaus nur schwer allgemeingültig quantifizierbar, da sie im sehr hohen Maße von den Anfragedaten abhängt. Weil bei den Simulationsdaten, die aus dem Webserver-Log gewonnen wurden, wenige Teilnehmer eine Vielzahl von benachbarten Anfragen stellen, wirkt sich das Caching der Kontaktinformationen erheblich stärker aus als bei einer großen Zahl von aktiven Teilnehmern, wie sie mit dem künstlichen Surfer erzeugt werden.

Einfluß der Verdrängungsstrategie des Cache. Der Cache ist wieder auf 20 Einträge beschränkt. Die Zahl der Knoten im CAN beträgt 3.000. Wir vermuten, dass die Unterschiede zwischen den Verdrängungsstrategien unter beschränkten Verhältnissen besonders

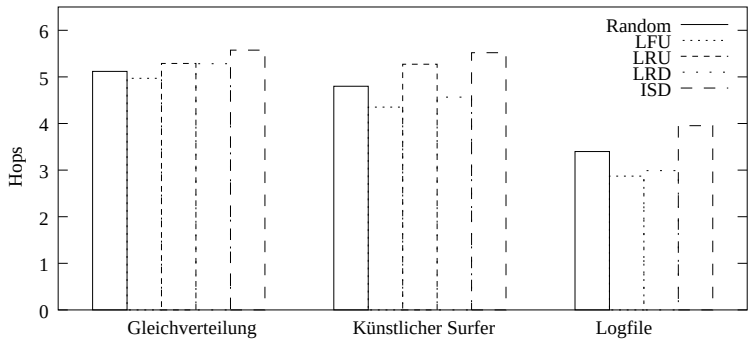

Abbildung 7: Vergleich der Caching-Strategien.

deutlich zu Tage treten. Es werden wieder 50.000 Abfragen abgesetzt, und wir ermitteln den Durchschnitt der Anzahl der Hops aller Nachrichten.

Abbildung 7 zeigt uns, dass die Verdrängungsstrategien nur marginal unterschiedliche Resultate erzielen. Weitere Messungen haben gezeigt, dass bei größeren Caches die Unterschiede weiter verschwinden.

Der Grund für dieses überraschende Resultat ist unseres Erachtens darin zu sehen, wie die Caches mit Daten gefüllt werden. Da die mit einer Weiterleitung überbrückte Distanz mit zunehmender Anzahl der Hops bis zum Erreichen des Ziels asymptotisch gegen Null verläuft, existieren sehr viele Kontakte über kurze Distanz und wenige über lange. Je größer der mit einem Kontakt im Cache überbrückte Schlüsselraum, desto seltener sein Auftreten. Aus diesem Grund erzeugen sämtliche untersuchten Caching-Strategien sehr ähnliche Caching-Inhalte, wenn auch aus völlig unterschiedlichen Positionen. Mit Random kommen häufig vorkommende Kontakte – also diejenigen kurzer Distanz – auch häufiger in den Caches vor. ISD erzielt denselben Effekt durch die Bildung einer expliziten Abhängigkeit von der Distanz der Kontaktinformationen. Da schließlich die häufig vorkommenden Kontaktinformationen kurzer Reichweite auch die von LFU, LRU und LRD geführten Statistiken dominieren, entstehen auch dort ähnliche Verteilungen.

5 Schlußbemerkungen

Verwandte Arbeiten. Dieser Abschnitt geht kurz auf verwandte Arbeiten ein, sofern die Abgrenzung in diesem Artikel noch nicht erfolgt ist.

Vorschläge für verteilte Datenstrukturen existieren in vielen Ausprägungen von verteilten Suchbäumen wie dem Distributed Random Tree [KW94] oder P-Grid [Abe01] bis zu varianten verteilter Hash-Tabellen wie CAN [RFH$^+$01], Chord [SMK$^+$01], LH* [LNS96], Pastry [RD01] oder Tapestry [ZKJ01]. Eine prominente Anwendung dieser verteilten Datenstrukturen sind moderne Peer-to-Peer Netze [MKL$^+$02]. [GHea01] skizziert die Imple-

mentierung von Query-Processing Operatoren in Peer-to-Peer Umgebungen. Auch wenn dieser Artikel im engeren Sinne zu unserem orthogonal ist, zeigt er, dass verteilte, Koordinator-freie Verwaltung großer Datenbestände auch in den Augen anderer wichtig ist.

Das zur Nachrichtenweiterleitung genutzte Verfahren *Greedy Routing* [ADS02] folgt dem *Small-World* Phänomen [Mil67] Milgrams, der durch Experimente zur Weiterleitung von Briefen Eigenschaften sozialer Netze sichtbar machte. Trotz der langen Zeitspanne seit der ersten Publikation Milgrams bestehen jedoch neben den in dieser Arbeit untersuchten Fragen zahlreiche weitere offene Punkte [RSS02] bei der Nachrichtenweiterleitung.

Eine Alternative zur Verbesserung des Routing-Verhaltens besteht darin, explizit leistungsfähige Routen als *Expressways* zu bestimmen. In [XZ02] wird ein Algorithmus vorgestellt, der in einer verteilten Umgebung automatisch Rechner mit hoher freier Kapazität als Endpunkte von Expressrouten bestimmt.

Das Thema von [GBHC00] ist die effiziente Implementierung von SDDS, und die Autoren warten mit eindrucksvollen Performance-Zahlen auf. Allerdings ist die zugrundeliegende Architektur ein zentral administrierter PC-Cluster, also nicht mit der mittelfristigen Ausrichtung unseres Projekts vergleichbar. Außerdem bleibt das Thema Caching bei [GBHC00] explizit außen vor.

Zusammenfassung und Ausblick. Im Rahmen unseres Projekts Fairnet beschäftigen wir uns mit der Weiterentwicklung verteilter skalierbarer Datenstrukturen. Die Voruntersuchungen, die dieser Artikel beschreibt, konzentrieren sich auf zwei Aspekte: Caching der Kontaktinformation und Lokalitätserhaltung der Abbildung der Datenobjekte in den Schlüsselraum der SDDS. Es stellt sich heraus, dass Caching zwar vorteilhaft ist, eine ausgefeilte Caching-Strategie im SDDS-Kontext aber keine Verbesserung mit sich bringt. Die Verwendung einer lokalitätserhaltenden Schlüsselabbildung in Kombination mit Caching hat sich dagegen als sehr vorteilhaft erwiesen. – Zukünftige Untersuchungen werden auch Ausfälle von Knoten betrachten, die die Wirksamkeit des Cachings beeinflussen.

Wir danken Stefan Knöfel, Kathleen Krebs und Mirko Tikalsky für ihre Hilfe bei dieser Studie.

Literaturverzeichnis

[Abe01] Karl Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. *Lecture Notes in Computer Science*, 2172:179–??, 2001.

[ADS02] James Aspnes, Zoë Diamadi, and Gauri Shah. Fault-tolerant routing in peer-to-peer systems. *21. ACM Symposium on Principles of Distributed Computing*, pages 223–232, July 2002.

[CGM99] Soumen Chakrabarti, David A. Gibson, and Kevin S. McCurley. Surfing the Web backwards. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1679–1693, May 1999.

[GBHC00] Steven D. Gribble, Eric A. Brewer, Joseph M. Hellerstein, and David Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *Proceedings of the*

4th Symposium on Operating Systems Design and Implementation (OSDI-00), pages 319–332, Berkeley, CA, October 23–25 2000. The USENIX Association.

[GHea01] Steven Gribble, Alon Halevy, and Zachary Ives et al. What Can Peer-to-Peer Do for Databases, and Vice Versa? In *Proceedings of the Fourth International Workshop on the Web and Databases*, 2001.

[IRD02] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A decentralized peer-to-peer web cache. *21th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, 2002.

[Kle00] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[KW94] Brigitte Kröll and Peter Widmayer. Distributing a Search Tree Among a Growing Number of Processors. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994*, pages 265–276. ACM Press, 1994.

[LNS96] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. LH* - A Scalable, Distributed Data Structure. *TODS*, 21(4):480–525, 1996.

[Mil67] Stanley Milgram. The small world problem. *Psychology Today*, 1(1):60–67, 1967.

[MKL+02] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, March 2002.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329 ff., 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In Roch Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 161–172, New York, August 2001. ACM Press.

[RMW94] M. Roscheisen, C. Mogensen, and T. Winograd. Shared web annotations as a platform for third-party value-added information providers: Architecture, 1994.

[RSS02] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing Algorithms for DHTs: Some Open Questions. 2002.

[SMK+01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In Roch Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, August 27–31 2001. ACM Press.

[TG97] Linda Tauscher and Saul Greenberg. How people revisit web pages: empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies*, 47(1):97–137, 1997.

[XZ02] Zhichen Xu and Zheng Zhang. Building Low-maintenance Expressways for P2P Systems. Tech Report HPL-2002-41, HP Laboratories, Palo Alto, March 2002.

[ZKJ01] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: an infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley, April 2001.