Anfragebearbeitung und Routing in Schema-basierten P2P-Systemen

Marcel Karnstedt
marcel.karnstedt@tu-ilmenau.de
Fakultät für Informatik und Automatisierung, TU Ilmenau
Postfach 100565, D-98684 Ilmenau

Zusammenfassung

Im Zusammenhang mit Filesharing-Anwendungen und skalierbaren verteilten Datenstrukturen hat sich das Peer-to-Peer (P2P) Paradigma in jüngster Zeit immer stärker verbreitet. Aufgrund ihres dezentralen Charakters versprechen P2P-Systeme erhöhte Robustheit und Skalierbarkeit und eröffnen dadurch neue Möglichkeiten für Datenintegrationsanwendungen. In solchen Schema-basierten P2P-Systemen kann ein neuer Teilnehmer bereits im Netzwerk existierende Korrespondenzen nutzen und Zugang zur vollständigen im Netz verfügbaren Information erhalten, indem er zu nur einem einzigen bereits integrierten Peer entsprechende Abbildungen definiert. Mit den neuen Möglichkeiten entstehen auch neue Anforderungen an das Design und die technische Realisierung solcher Systeme. Die vorliegende Arbeit nimmt Bezug auf eine dieser Anforderungen: das Problem der verteilten Anfragebearbeitung. Wir untersuchen verschiedene Bearbeitungs- und Routingstrategien basierend auf verschiedenen Arten von Routing Indexen und präsentieren erste experimentelle Ergebnisse.

1 Einleitung

Dezentralisierung verspricht bei der verteilten Anfragebearbeitung in Datenintegrationsszenarien eine wesentlich verbesserte Skalierbarkeit und Effizienz gegenüber klassischen zentralisierten Ansätzen. Peer-to-Peer-Systeme (P2P-Systeme) stellen eine konsequente Realisierung des dezentralen Gedankens dar. Es existiert kein globales Wissen, weder in Form eines Schemas noch über die Verteilung der Daten. Ein Peer hat nur das Wissen über die lokalen Daten und die Daten der direkt benachbarten Peers. In darauf basierenden Integrationsansätzen agiert jedes Peer als Quelle und zwischen den Peers werden Schema-Korrespondenzen definiert ([GHI+01]).

In dieser Arbeit beschäftigen wir uns mit einem Problem das in Verbindung mit P2P-Systemen auftritt, dem Problem der effizienten verteilten Anfragebearbeitung. Im Zusammenhang damit untersuchen wir speziell das Problem des Routings von Anfragen. In Abschnitt 2 wird kurz das genutzte Datenmodell beschrieben, woraufhin dann in Abschnitt 3 mögliche Anfragestrategien beschrieben werden. Auf die Einführung einer möglichen Routingstrategie in Abschnitt 4 beschreiben wir erste Tests zur Evaluation unserer Ansätze in Abschnitt 5. Ein kurzer Ausblick schließt die Arbeit ab.

2 Datenmodell

Im weiteren Verlauf gehen wir davon aus, dass alle Peers ihre Daten in Form von XML bereitstellen und jedes Peer-Schema durch ein DTD oder XML-Schema repräsentiert wird. Außer Schema-Definitionen benötigen wir Korrespondenzen zwischen verschiedenen Schemata. Wir schränken mögliche Ansätze in dieser Arbeit auf zwei Kernoperationen ein, die wir im Folgenden kurz umreißen. Seien D_1 und D_2 zwei XML-Dokumente und e_1 und e_2 Pfade in diesen Dokumenten. Wir definieren:

- *equivalence* dargestellt durch $(D_1)//e_1 \equiv (D_2)//e_2$: besagt, dass das Element e_1 in D_1 semantisch identisch zu dem Objekt beschrieben von e_2 in D_2 ist. Dadurch wird eine *horizontale* Fragmentierung realisiert. Die Beziehung kann genauer definiert werden durch erweiterte Beziehungstypen (overlap, disjoint, inclusion etc.).

- *child-of/part-of* dargestellt durch $(D_1)//e_1 \prec_{id_1=id_2} (D_2)//e_2$: bedeutet, dass das Element e_2 in D_2 ein Sibling des Elementes e_1 ist. Die Bedingung $id_1 = id_2$ ist die Verbundbedingung, wobei id_i selbst ein Pfad in D_i ist. Diese Beziehung entspricht einer *vertikalen* Fragmentierung.

Zusätzlich zu den aufgelisteten Operationen ist eine *Transformation* τ nötig. $\tau((D_1)//e_1)$ transformiert den von e_1 referenzierten Teilbaum D_1 auf bestimmte Art und Weise, z.B. durch Restrukturierung oder Extraktion von Elementen. Realisiert werden können solche Transformationen mit Hilfe von Stylesheets. Im Weiteren gehen wir davon aus, dass eine solche Transformation und die inverse Operation dargestellt und für die Anfragebildung verwendet werden können, ohne näher darauf einzugehen.

Mit diesen Beziehungen lassen sich nun Korrespondenzen definieren. Als Beispiel nehmen wir ein einfaches Szenario an, in dem 4 Peers $P_1 \dots P_4$ Daten über Kunstwerke vereinen. Die in Abbildung 1 dargestellten Schemata der Peers sollen die Beziehungen untereinander verdeutlichen. Hier werden Verbindungen zwischen P_1 und P_2, P_1 und P_3 und schließlich zwischen P_2 und P_4 etabliert. Die jeweiligen Korrespondenzen zwischen den Peers werden mit den eingeführten Operationen entsprechend definiert.

P_1:
```
<gallery>
  <painting id="...≫
    <title>...</title>
    <artist>...</artist>
  </painting>
  ...
</gallery>
```

P_2:
```
<sculptures>
  <object id="...≫
    <title>...</title>
    <artist>...</artist>
  </object>
  ...
</sculptures>
```

P_3:
```
<artists>
  <person>
    <name>...</name>
    <birth date= />
  </person>
   ...
</artists>
```

P_4:
```
<catalog>
  <item>
    <title>...</title>
    <exhibition>...</exhibition>
  </item>
  ...
</catalog>
```

Abbildung 1: Beispielszenario

Zur Anfrageformulierung nutzen wir eine Teilmenge von XQuery, entsprechend der Verwendung von XPath mit Verbunden. Zur Darstellung der Anfragen nutzen wir eine Menge von XML Algebraoperatoren (ähnlich [VGD+02]). Auf eine Auflistung und genauere Beschreibung wird hierverzichtet.

Ein wesentlicher Aspekt eines verteilten Anfrageprozessors ist eine effiziente Anfragebearbeitung. Simples fluten des Netzwerkes indem jedes Peer alle zur Verfügung stehenden Verbindungen zu anderen Peers nutzt, ist aufgrund der Größe der untersuchten Netzwerke nicht realisierbar. In Verbindung mit einem *Time-To-Live Parameter (TTL)* kann diese Strategie zu unvollständigen Resultaten führen. Globales Wissen steht nicht zur Verfügung. Im nächsten Abschnitt beschäftigen wir uns daher mit einem möglichen Ansatz zur Lösung dieses Problems unter Verwendung von lokal beschränkter Information.

3 Anfragebearbeitung

In unserer Implementierung werden Anfragen in Form von *Plan-Operatoren* repräsentiert. Bei Verwendung eines solchen graphbasierten Modells werden Anfragen als Bäume repräsentiert. Jeder Knoten des Baumes stellt eine bestimmte Operation dar. Die einzelnen Operationen werden als Implementierung einer allgemeinen Schnittstelle *POP* realisiert.

Die wesentlichen Ansätze für die verteilte Anfragebearbeitung bilden *Data Shipping* und *Query Shipping*. Beim Data Shipping werden alle benötigten Daten zum initialisierenden Peer gesendet und die Ausführung sämtlicher Operationen erfolgt bei diesem Peer. Beim Query Shipping Ansatz wird das im Netz transportierte Datenvolumen im Vergleich stark reduziert. Bei diesem Verfahren werden alle Operationen so tief wie möglich in der Hierarchie der Peers ausgeführt, d.h. nur Daten, die lokal nicht

weiterverarbeitet werden können, werden an andere Peers gesendet. Es existieren einige Arbeiten in denen gezeigt wird, dass der Query Shipping Ansatz wesentlich effizienter in Hinsicht der Anzahl generierter Nachrichten und des gesendeten Datenvolumens ist, wenn kein Daten-Caching genutzt wird (z.B. [FJK96]). Da wir uns auf sehr große P2P-Netzwerke konzentrieren, in denen die einzelnen Peers, zur Zeit, kein Caching nutzen, beschränken wir uns in dieser Arbeit auf den Ansatz des Query Shippings. Beim Query Shipping kann man die Anfragen zum Einen zerlegen und die Teilanfragen separat im Netz bearbeiten. Wir nutzen einen anderen Ansatz, bei dem mutierende Anfragepläne im Netz versendet und von den Peers bearbeitet werden, z.B. indem sie Antwortdaten in den Plan einfügen (ähnlich [PM02]).

Der gesamte Query Shipping-Algorithmus wird hier nicht gelistet. Wir beschränken uns auf seinen Kern, die Prozedur *process-POP()*, in der die eigentliche Anfragebearbeitung bei jedem Peer abläuft. Eine stark gekürzte Version der Prozedur ist in Abbildung 2 dargestellt. Zur parallelen Anfragebearbeitung werden Nachrichten versendet, nach deren Empfang ein Peer die Prozedur *process-POP()* startet.

Input: POP q
Output: Nachricht zurück zum Initiator, enthält q gefüllt mit Daten

```
1   found-one = false;
2   /* Korrespondenzen auf q anwenden und lokale Daten eintragen */
3   forall Nachbar-Peers P do
4       if data-known-to-peer(q, P) then /* starte process-POP(q) bei P um parallel q zu bearbeiten*/
5   od
6   if found-one = false then /* frage alle Nachbarn */
7   /* Mische erhaltene Pläne und verarbeite Joins u.a. Operationen */
8   send-POP-back-to-initiator(q);
```

Abbildung 2: Prozedur *process-POP*

In der Prozedur *data-known-to-peer()* (Zeile 4) wenden wir Routing Strategien an, die in Abschnitt 4 kurz beschrieben werden. Wir fragen nur Peers an, die uns auch Daten liefern können. Finden wir kein solches Peer, nutzen wir wie beim *Flooding* alle lokal etablierten Verbindungen (Zeile 6).

4 Routing von Anfragen

Wenn ein Peer eine Anfrage lokal nicht (vollständig) beantworten kann, so benötigt es Informationen über die Qualität der Daten, die über die Verbindungen zu den benachbarten Peers (Peers mit einer direkten Verbindung im Netzwerk) angefragt werden können. Dies umfasst auch die Daten von Peers, die mehrere Verbindungsschritte (*hops*) vom aktuellen Peer entfernt liegen. Die Daten dieser Peers werden nicht durch die lokal definierten Korrespondenzen beschrieben, wie die der Nachbarn, sondern müssen durch andere Datenstrukturen erfasst werden. Die Frage, welche Verbindungen nun am besten geeignet sind, um eine aktuelle Anfrage möglichst effizient zu beantworten, wird allgemein mit dem Problem des *Routings* von Anfragen beschrieben. Bei der von uns gewählten Datenstruktur, die wie beschrieben die Daten der weiter entfernt liegenden Peers erfassen soll, handelt es sich um eine Form von Routing Tabellen, sogenannte *Routing Indexe* (genauer *Compound Routing Indexes* [CGM02]). Da eine Erfassung der Daten aller im Netzwerk integrierten Peers schlichtweg unmöglich ist, beschränken wir die Indexe durch einen *hop count*. Es werden nur die Daten erfasst, die von Peers geliefert werden können, die maximal hop count Schritte vom Index haltenden Peer entfernt liegen. Bei einem hop count von 1 entspricht der Index den lokal definierten Korrespondenzen. Würde man ihn zum Maximum erhöhen, so würde er wiederum globales Wissen reflektieren, was wie erläutert aber nicht realisierbar ist. Als Bezeichner auf Schemaebene nutzen wir die Pfade der in den Anfragen enthaltenen XPath-Anfragen. Zusätzlich indexieren wir die Daten auf Instanzebene, d.h. durch entsprechende Filterprädikate. Diese werden definiert als Informationen über die Verteilung der Daten und unter den Peers entsprechend ausgetauscht. Auf nähere Ausführungen zu den verwendeten Indexen wird hier aus Platzgründen verzichtet. Der in Tabel-

Nachbar ID	Kategorie (Schemaebene)	Prädikat (Instanzebene)	Kardinalität	Anzahl Peers
1	painting	-	520	4
	painting/title	-	520	3
	...	...	...	...
	painting/person	-	210	2
	painting/person/name	-	210	2
	painting/person/birth	[@date<'1800']	132	1
	painting/person/birth	[@date≥'1800']	78	1
4	item	-	112	5
...	...	...	...	...

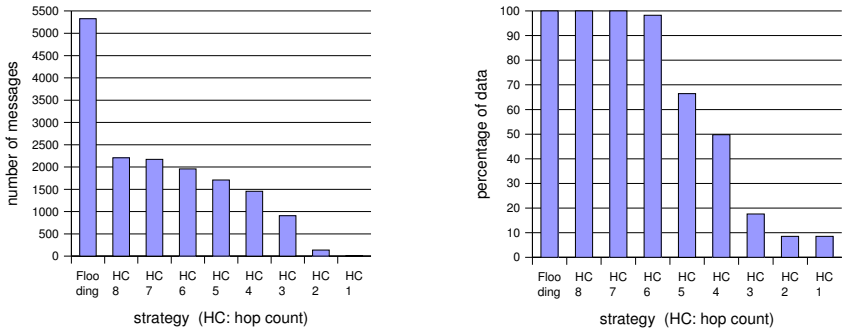
Tabelle 1: Beispiel eines Compound Routing Indexes bei Peer P_2

le 1 dargestellte Ausschnitt eines Routing Indexes soll zur Veranschaulichung dienen. Abgebildet ist ein an Peer P_2 gebildeter Index. P_2 entspricht hier dem Peer P_2 aus dem in Abschnitt 2 angeführten Beispiel. Es wird ersichtlich, wie die Beschreibung der von P_3 gelieferten Daten, die nur über die Verbindung zu P_1 abfragbar sind, in den Index integriert wird.

Zu Zeit von uns untersuchte Probleme sind vor allem der effiziente Aufbau und die Pflege der Routing Indexe. Im Moment finden auch keine Methoden zur Wahrung der Kohärenz, also zur Unterstützung von Veränderungen der im Netzwerk erfassten Daten, Anwendung. Ein weiterer, aktuell untersuchter, interessanter Aspekt dieser Indexe ist die mögliche Nutzung für ein *dynamisches Kostenmodell*, das zur Anfragebearbeitung genutzt werden kann. Dies umfasst unter anderem die dynamische Generierung von Anfrageplänen, die Definition von neuen, am verteilten Modell orientierten Kostenfaktoren und Entscheidungen basierend auf Aspekten der Datenqualität.

5 Evaluation

Um die beschriebenen Anfrage- und Routingstrategien vorläufig bewerten zu können, haben wir ein kleines Testszenario entwickelt. In einem Netzwerk aus 40 Peers wurden Daten von Shakespeare-Stücken ([Bos99]) auf die einzelnen Peers verteilt. Die Verbindungen zwischen den Peers wurden zufällig gewählt, die bidirektionalen Korrespondenzen zunächst manuell definiert. Wir untersuchen zur Zeit auch Möglichkeiten und Ansätze, diese Mappings mit Hilfe eines (semi)-automatischen regelbasierten Algorithmus zu bilden. Wir generierten ein Anfragemix aus 17 Anfragen, initiiert an 4 verschiedenen Peers, insgesamt also 68 Anfragen. Implementiert wurde das Szenario in Java, das Verhalten der Peers wird durch Threads simuliert. In diesem Abschnitt werden kurz die Ergebnisse eines der wichtigsten Tests vorgestellt. Die Ergebnisse weiterer Tests sowie eine erweiterte Auswertung können in [KHS04] nachgelesen werden.



(a) Anzahl Nachrichten

(b) Real erzielter Anteil des Ergebnisses

Abbildung 3: Vergleich von Routing Indexen mit verschiedenen hop counts

Wir betrachten den Nutzen für die Anfragebearbeitung bei Unterstützung von Routing Indexen auf Schemaebene mit verschiedenen hop counts. Die Ergebnisse sind in Abbildung 3 zu sehen. *HC* x steht für die Strategie, die einen hop count von x nutzt. Zum Vergleich wurden zusätzlich die Ergebnisse des simplen Flooding-Ansatzes festgehalten. Als erstes Fazit halten wir fest, dass die Anfragebearbeitung mit Unterstützung von Indexen wesentlich effizienter abläuft als ohne jeden Index, selbst bei niedrigem hop count. Allerdings schränken zu niedrige Werte den Blick eines Peers auf das Netzwerk zu sehr ein und die Ergebnisse sind unbefriedigend unvollständig. In unserem Szenario erreichen wir mit einem hop count von 5 oder 6 sehr gute Ergebnisse im empfangenen Datenanteil und der Anzahl generierter Nachrichten. Diese Werte liegen etwa in der Mitte der beiden Extreme globales Wissen (hop count maximal) und nur Wissen über die Nachbarn (hop count 1).

In einem Satz zusammengefasst, reflektieren die Ergebnisse die erwartete Effizienzsteigerung bei Nutzung von Routing Indexen. Festzuhalten ist der hop count als wesentlicher Einflussfaktor auf die Beziehung zwischen Effizienz der Anfragebearbeitung und Umfang der erhaltenen Daten.

6 Schlussfolgerung

Eine der größten Herausforderungen in P2P-basierten Datenintegrationssystemen ist die effiziente Anfragebearbeitung. Die wesentlichen Probleme beruhen auf dem dezentralen Charakter dieser Systeme, durch den aus verteilten Datenbanksystemen bekannte Verfahren ineffizient werden. In dieser Arbeit wurden mögliche Ansätze zur Lösung dieser Probleme unter Verwendung von beschränktem lokalen Wissen basierend auf Routing Indexen beschrieben und evaluiert. Offene Fragen umfassen im Wesentlichen deren Aufbau und Pflege sowie die Frage nach einem optimalen Wissenshorizont, ausgedrückt durch den verwendeten hop count. Im Hinblick auf die Entwicklung eines verteilten Anfrageprozessors für P2P-Systeme verbleiben aber noch viele andere offene Schritte. So haben wir z.B. Anfragekosten und Möglichkeiten der kostenbasierten Optimierung vernachlässigt, aber auch in P2P-Systemen benötigte adaptive Anfragetechniken ([GPFS02]). Dies sind einige wesentliche Punkte unserer zukünftigen Arbeit.

Literatur

[Bos99] J. Bosak. Shakespeare 2.00, 1999. The plays of Shakespeare, marked up by Jon Bosak available at metalab.unc.edu/bosak/xml/eg/shaks200.zip.

[CGM02] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 28 tn Conference on Distributed Computing Systems*, July 2002.

[FJK96] M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *Proceedings of the SIGMOD Conference*, pages 149–160, 1996.

[GHI+01] S. D. Gribble, A. Y. Halevy, Z. G. Ives, M. Rodrig, and D. Suciu. What Can Database Do for Peer-to-Peer? In *WebDB 2001*, pages 31–36, 2001.

[GPFS02] A. Gounaris, N. W. Paton, A. A. A. Fernandes, and R. Sakellariou. Adaptive Query Processing: A Survey. In *BNCOD 2002*, pages 11–25, 2002.

[KHS04] M. Karnstedt, K. Hose, and K.-U. Sattler. Distributed Query Processing in P2P Systems with incomplete schema information. In *Proc. of the 3rd Int. Workshop on Data Integration over the Web (DIWeb2004), June 2004, Riga, Latvia, to appear*, 2004.

[PM02] V. Papadimos and D. Maier. Mutant Query Plans. *Information and Software Technology*, 44(4):197–206, April 2002.

[VGD+02] S. Viglas, L. Galanis, D. DeWitt, J. Naughton, and D. Maier. Putting XML Query Algebras into Context. submitted for publication, 2002.