Verteilte Anfrageverarbeitung in DHT-basierten P2P-Systemen

Philipp Rösch[1] Christian von der Weth[2] Kai-Uwe Sattler[1] Erik Buchmann[3]
[1]Fakultät Informatik und Automatisierung, TU Ilmenau
[2]Fakultät Informatik, Universität Karlsruhe (TH)
[3]Fakultät Informatik, Universität Magdeburg

Abstract: Peer-to-Peer (P2P)-Systeme und hierbei speziell verteilte Datenstrukturen versprechen Skalierbarkeit bis auf Internet-Größe bei fairer Verteilung der Infrastrukturkosten und hoher Robustheit. Für viele potenzielle Anwendungen sind die von derartigen Systemen unterstützten einfachen Schlüsselzugriffe nicht ausreichend – vielmehr ist die Verwaltung und Anfrage komplex strukturierter Daten notwendig. In diesem Beitrag untersuchen wir daher, ob und wie sich Operationen und Strategien für die verteilte Verarbeitung SQL-artiger Anfragen in P2P-Systemen auf Basis verteilter Hashtabellen effizient realisieren lassen.

1 Einleitung

Eine der aktuellen Herausforderungen im Bereich Datenmanagement ist die Verwaltung riesiger Datenbestände in massiv verteilten Systemen, die bis zur Internet-Größe reichen können, bei einer großen Anzahl paralleler Anfragen. Eine derartige Skalierbarkeit ist jedoch nur dann erreichbar, wenn weitgehend auf zentrale Komponenten und Strategien, die globales Wissen erfordern, verzichtet wird. P2P-Systeme – und hier speziell Systeme in Form verteilter Datenstrukturen – versprechen durch ihre Charakteristika diese Skalierbarkeit bei gleichzeitiger hoher Robustheit gegenüber Ausfällen und Angriffen. Solche Systeme sind daher insbesondere für folgende Einsatzfälle prädestiniert: *(1)* Es sind extrem große Datenbestände für viele Nutzer zu verwalten. Gleichzeitig trägt ein großer Personenkreis zur Sammlung, Erweiterung und Pflege der Daten bei. *(2)* Die faire Verteilung der Infrastrukturkosten hat eine große Bedeutung, da einzelne Personen oder Organisationen meist nicht bereit oder in der Lage sind, diese Kosten allein zu tragen. *(3)* Über die Möglichkeiten der Anfrageausführung hinaus sind weitere datenbanktyische Eigenschaften wie transaktionale Garantien oder globale Änderbarkeit meist sekundär, da Aufbau und Pflege einzelner Bestände meist dezentral erfolgen.

Konkrete Beispiele für derartige Anwendungsszenarien sind u.a. die Verwaltung *öffentlicher Daten* – d.h. Datensammlungen, die für einen breiten Nutzerkreis von Interesse sind – wie etwa Daten der Genomanalyse, Metadaten für das Semantic Web, wissenschaftliche Daten oder auch typische File-Sharing-Daten [THH+04]. So lassen sich P2P-Systeme als strukturierter Index für derartige Datensammlungen einsetzen.

Einen vielversprechender Ansatz für die genannten Herausforderungen stellen verteilte Hashtabellen (engl. distributed hash tables – DHTs) dar. Allerdings sind in vielen Anwendungen die von diesen Systemen angebotenen schlüsselbasierten Zugriffe auf unstrukturierte Datenbestände nicht ausreichend. Unser Ziel ist es daher, (relational) strukturierte Daten in einem DHT-System zu verwalten und Anfragen mit Such-, Verknüpfungs- und Transformationsoperationen zu ermöglichen. In diesem Beitrag werden wir daher ein verteiltes Anfragesystem für ein DHT-basiertes Datenmanagementsystem vorstellen, das SQL-artige Anfragen verarbeiten kann.

2 Verteilte Hashtabellen und CAN

Verteilte Hashtabellen sind eine Ausprägung von verteilten, skalierbaren Datenstrukturen, die dem Peer-to-Peer (P2P) Paradigma folgen. Sie verwalten große Mengen von (Schlüssel,Wert)-Paaren über die Schnittstelle einer Hashtabelle. Hierbei werden Lookup-Operationen angeboten, die typischerweise in einem Netzwerk aus n Knoten nur $\log n$ Schritte erfordern. Eine konkrete Variante von DHTs sind u.a. Content Addressable Network (CAN) [RFH$^+$01], die wir auch für unsere Arbeit als Grundlage gewählt haben. Die vorgestellten Techniken sind jedoch vollständig auf andere DHTs übertragbar, einzig das Fragmentierungsschema ist auf CANs zugeschnitten und müsste für den Einsatz mit anderen DHTs modifiziert werden. Ein CAN ist ein verteiltes System zur Verwaltung von (Schlüssel,Wert)-Paaren, das aus vielen unabhängigen, gleichberechtigten Knoten (Peers) besteht. Diese Peers sind üblicherweise PCs oder Workstations, die von den Teilnehmern betrieben werden, die am CAN partizipieren wollen. Der Raum aller gültigen Schlüssel K im CAN ist die Abbildung eines d-dimensionalen kartesischen Schlüsselraums $K \subset \mathbb{N}^d$ auf einen d-Torus. Die Schlüssel der Anwendung werden dabei mit einer Hashfunktion in den Schlüsselraum des CAN transformiert. Der Schlüsselraum ist unabhängig vom darunterliegenden physikalischen Netzwerk, d.h. das CAN formt ein virtuelles Overlay-Netzwerk. Ein Schlüssel im CAN ist eine Liste von d Koordinaten: $k = \{c_1, c_2, ...c_d\}, c_i \in \mathbb{N}$. Jeder Peer $p \in P$ ist für einen bestimmten Ausschnitt $(k_p^{\min}, k_p^{\max})$ aus dem Schlüsselraum (Zone) verantwortlich, d.h., er speichert alle Datensätze, für die gilt: $k \in (k_p^{\min}, k_p^{\max})$. Daneben verwaltet jeder Peer eine Liste aller Peers, deren Zone zu seiner unmittelbar benachbart ist.

Das CAN bietet im Wesentlichen die Operationen **`put`** und **`get`**. Dabei speichert die **`put`**-Operation einen Datensatz auf dem Peer, in dessen Zone der Schlüssel gehört; **`get`** wiederum fragt diesen Datensatz ab. Die Operationen müssen daher an den Peer weitergeleitet werden, der für die jeweils passende Zone verantwortlich ist, um jedem Teilnehmer im CAN den Zugriff auf alle (Schlüssel,Wert)-Paare zu ermöglichen. Dafür kommt eine Variante des *Greedy Forward Routing* zum Einsatz: Erhält ein Peer eine Operation, die nicht für seine Zone bestimmt ist, so bestimmt er den euklidischen Abstand zwischen dem Schlüssel der Operation und den ihm bekannten Nachbarn. Die Operation wird nun an den Nachbarn weitergeleitet, dessen Abstand zum Schlüsselwert am geringsten ist. Der Empfänger verfährt ebenso, bis der Peer erreicht ist, der für die passende Zone verantwortlich ist. Für ein Netzwerk aus n Peers werden dabei im Durchschnitt $\frac{d}{4} \cdot n^{\frac{1}{d}}$ Schritte benötigt.

3 Datenverteilung

Sollen nun relationale Daten in einem CAN verwaltet werden, so stellt sich die Frage, wie diese auf den einzelnen Peers verteilt werden, d.h., ob und wie Relationen fragmentiert werden, auf welche Attribute die Hashfunktion(en) angewendet werden, ob eine geclusterte Speicherung sinnvoll ist und welche Hashfunktionen überhaupt verwendet werden sollen. Hierbei bestehen zwei konträre Ziele: Einerseits sollen die Daten im Interesse einer fairen Lastverteilung möglichst gleichmäßig auf alle Peers verteilt werden, andererseits ist für eine effiziente Verarbeitung von Anfragen (insbesondere Bereichs- und Verbundanfragen) eine möglichst zusammenhängende Speicherung anzustreben. Hierfür erscheinen prinzipiell Ansätze auf Basis raumfüllender Kurven geeignet [AX02].

Unter Berücksichtigung dieser Anforderungen und mit dem Ziel einer fairen Lastverteilung haben wir bisher mit einem einfachen lokalitätssensitiven Fragmentierungsansatz experimentiert, der auf einer Reverse-Bit-Interleaving-Technik basiert. Hierbei wird eine Relation R mit dem Relationenschema $\mathcal{R}(A_K, A_1, A_2, \ldots, A_n)$ – wobei A_K ein frei gewähltes Schlüsselattribut bezeichnet – durch zwei Hashfunktionen h_R und h_K in den Schlüsselraum des CAN abgebildet. Die Hashfunktion h_R wird dabei auf die Relationen-ID (etwa den Relationennamen), die Hashfunktion h_K auf den Schlüsselwert eines Tupels der Relation angewendet. Durch Konkatenieren der resultierenden Bitwerte ergibt sich somit für ein Tupel $t \in R$ der Schlüsselwert $k'_t = h(\mathcal{R}, t(A_K)) = h_R(\mathcal{R}) \circ h_K(t(A_K))$. Die Bitfolge von k'_t wird anschließend in die d Koordinaten des CAN aufgeteilt, indem Bit $0, d, 2d, \ldots$ den Wert der ersten Koordinate, Bit $1, (d+1), (2d+1), \ldots$ den Wert der zweiten Koordinate usw. bilden, sodass ein d-dimensionaler CAN-Schlüssel entsteht (Abb. 1 links). Somit werden die Tupel einer Relation entlang einer Z-Kurve im CAN verteilt, wobei k'_t dem Z-Wert des Tupels t entspricht (Abb. 1 rechts). Die kombinierte Anwendung von zwei Hashfunktionen ermöglicht dabei eine flexible Kontrolle der Verteilung: Mit der ersten Hashfunktion h_R wird der Startpunkt der Relation festgelegt, die zweite Hashfunktion h_K bestimmt den Verlauf. Die verwendeten Hashfunktionen müssen weiterhin die spezielle Eigenschaft der Ordnungserhaltung erfüllen, sodass sichergestellt ist, dass die für einen gegebenen Wertebereich zuständigen Peers direkt über den Z-Wert und somit den CAN-Schlüssel angesprochen werden können.

Dieses Fragmentierungsschema besitzt noch weitere Freiheitsgrade, die die Anfrageperformanz beeinflussen. So ist beispielsweise in Abhängigkeit vom Wertebereich des Koordinatenraums des CAN eine Spreizung der gehashten Schlüsselwerte möglich, welche die Anzahl der Peers bestimmt, über die die Relation verteilt wird.

4 Anfrageoperatoren

Anfragepläne werden üblicherweise als Graphen von Planoperatoren (POP) repräsentiert. Diese Planoperatoren unterstützen eine gemeinsame Schnittstelle, wobei wir jedoch nicht das bekannte Iterator-Modell [Gr93] einsetzen sondern einen Push-Ansatz verfolgen. Dies bedeutet, dass jeder Operator seine produzierten Ergebnistupel an seinen Elternknoten im

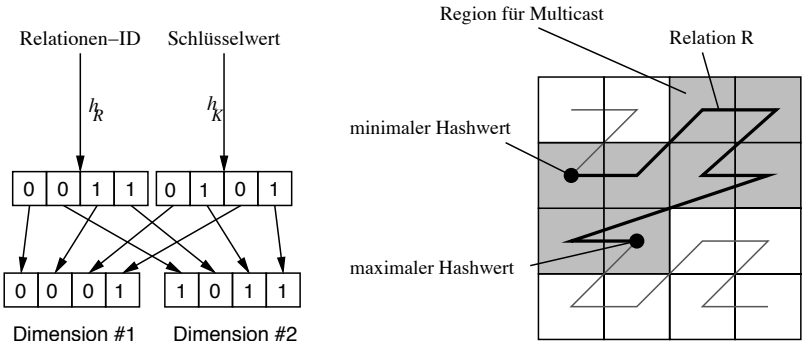

Abbildung 1: Verteilung der Daten im CAN

Plan sendet. Der Peer (oder genauer der Punkt im Schlüsselraum), der diesen Operator verarbeitet, wird dabei anhand der Tupelwerte (in Abhängigkeit von der Art der nachfolgenden Operation) bestimmt. Zu beachten ist, dass im Normalfall mehrere Instanzen eines Operators zur gleichen Zeit existieren, d.h. es wird Intra-Operator-Parallelität realisiert, indem die Operation parallel von mehreren Peers, die Fragmente der gesuchten Relation verwalten, ausgeführt wird.

Für die Realisierung der verschiedenen Planoperatoren werden ergänzend zu den CAN-Basisoperationen noch weitere Primitive benötigt: **`temp_put(`**k**`,`** v**`,`** tm**`)`** als eine Variante von **`put`**, die das (k, v)-Paar nur temporär (für die Zeitdauer tm) im CAN einfügt und insbesondere für das Rehashing zur Unterstützung von Verbundberechnungen hilfreich ist, **`send_message(`**z**`,`** m**`)`** zum Senden der Nachricht m (z.B. einen Anfrageplan) an den Peer, in dessen Zone der Punkt z fällt und **`send_multicast(`**$z_{\min}$**`,`** $z_{\max}$**`,`** m**`)`** als Multicast-Variante, die die Nachricht m an alle Peers sendet, die Zonen im Z-Kurvenintervall $\langle z_{\min}, z_{\max} \rangle$ verwalten. Auf der Basis dieser Operationen haben wir Planoperatoren für Selektion, Verbund und Gruppierung/Aggregation entwickelt, die im Folgenden kurz vorgestellt werden. Grundsätzlich kann dabei auf Ideen aus dem Bereich verteilter und paralleler Datenbanksysteme sowie speziell zu hashbasierten Operationen [Br84] zurückgegriffen werden. Allerdings müssen dabei die Besonderheiten von DHTs wie Routing und das Fehlen einer globalen Koordination berücksichtigt werden.

Die von uns realisierten Planoperatoren sind immer zweigeteilt, d.h. sie arbeiten in zwei Schritten: Zunächst wird festgestellt, *wo* der Operator ausgeführt werden soll, d.h. es wird anhand der Selektionsbedingung oder der Tupel des Zwischenergebnisses die Zone(n) im CAN identifiziert, die Ergebnis- oder Verbundkandidaten enthält. An den Peer dieser Zone (den Ausführungspeer) wird dann der Plan mit dem Verweis auf den nächsten Schritt gesendet. Im zweiten Schritt führt dieser Peer genau diese Operation auf den lokalen Daten aus. Bei der Vorstellung der Operatoren gehen wir daher davon aus, dass der *Initiatorpeer* jeweils der Peer ist, der die Entscheidung über den nächsten Ausführungsort trifft. Dieser Peer p_i hat den aktuellen Plan soweit abgearbeitet, dass ein Zwischenergebnis vorliegt, das lokal nicht weiter bearbeitet werden kann.

Der *SelectionPOP* stellt eine Scan-Operation auf einer Relation R mit einer optionalen Selektionsbedingung F der Form $A_K \theta c$ dar. Folgende Fälle müssen dabei berücksichtigt werden: *(1)* Punktanfrage auf dem Schlüsselattribut, wobei der für diese Anfrage zuständige Peer einfach durch die Anwendung der Hashfunktion auf dem Relationnamen $\mathcal{R}$ und dem Schlüsselwert ($h(\mathcal{R}, c)$) ermittelt werden kann, *(2)* Bereichsanfrage auf dem Schlüsselattribut, bei der die Selektionsbedingung wie beschrieben auf ein Intervall der Z-Kurve abgebildet und der Plan über Multicast an alle Peers dieses Intervalls geschickt wird, *(3)* Selektion auf einem Nichtschlüsselattribut, die durch einen Multicast an alle Peers der betreffenden Relation verarbeitet wird.

Realisierungen von Verbundoperationen in DHTs wurden bereits in [HHL+03, TP03] untersucht. Wir haben zwei konkrete Varianten implementiert: als partitionierter Nested Loops Join (*PartitionedNLJoinPOP*) und als Symmetric Hash Join (*SHJoinPOP*). Beim *PartitionedNLJoinPOP* senden Peers, die einen der beiden Verbundpartner verwalten, ihre Tupel an die Peers der anderen Relation, wobei die jeweiligen Empfängerpeers durch Anwendung der Hashfunktion auf die Tupel identifiziert werden. Für den *SHJoinPOP* werden die Tupel beider Relationen anhand der Werte der Verbundattribute durch den Initiator neu verteilt, d.h. im Schlüsselraum der Verbundrelation eingefügt. Die verantwortlichen Peers bilden gleichzeitig die Verbundpeers, die den eigentlichen Verbund über einen Symmetric Hash Join [WA93] berechnen.

Gruppierung wird im Relationenmodell immer zusammen mit einer Aggregation durchgeführt. Für diesen Zweck wurden ebenfalls zwei Varianten implementiert. Der naive *CentralGroupingPOP* – der immer am Anfragepeer als Wurzel des Plans ausgeführt wird – sammelt und gruppiert die empfangenen Tupel und wendet darauf die entsprechende Aggregationsfunktion an. Der *HashGroupingPOP* nutzt dagegen die Eigenschaften des CAN, indem die zu gruppierenden Tupel entsprechend der Werte ihrer Gruppierungsattribute temporär neu verteilt werden. So entstehen an den Gruppierungspeers automatisch die Gruppen, auf die anschließend die Aggregation angewendet werden kann.

Da die Zwischenergebnisse im Normalfall auf verschiedenen Peers verteilt produziert werden, wird auf dem Anfragepeer noch ein spezieller Kollektoroperator benötigt, der die Teile des Anfrageergebnisses einsammelt und schließlich ausgibt. Auf die damit verbundenen Besonderheiten werden wir im folgenden Abschnitt eingehen.

5 Anfrageplanung und -ausführung

Wie in Abschnitt 4 bereits dargestellt, werden Anfragen zur Abarbeitung in Pläne in Form von Bäumen aus Planoperatoren überführt. Aufgrund des fehlenden globalen Wissens ist eine Optimierung hierbei nur sehr eingeschränkt möglich, etwa durch Abschätzung der benötigten Hops, wofür jedoch zumindest Annahmen über die Datenverteilung getroffen werden müssen. So werden bisher nur links-orientierte Anfragebäume erstellt, wobei versucht wird, eine hashbasierte Selektion als den am weitesten links stehenden Operator im Baum anzuordnen. Sofern die Anfrage eine Gruppierung enthält, wird der Gruppierungsoperator inklusive der Aggregationen als Wurzel im Baum gesetzt.

Der Grund für das bei der Realisierung der Planoperatoren verfolgte Push-Prinzip ist das Ziel einer weitgehend *zustandslosen* Anfrageverarbeitung. Bei einem Pull-basierten Ansatz muss ein Operator auf die Bereitstellung der Eingabe durch seine(n) Kindknoten im Plan warten. Dies setzt voraus, dass der Ausführungszustand gespeichert wird, solange die Verarbeitung auf einem anderen Peer fortgesetzt wird. Darüber hinaus kann es bei Knotenausfällen zu Fehlern oder zumindest Verzögerungen durch Zeitüberschreitungen kommen. Im hier beschriebenen Ansatz wird das vermieden, indem Tupel bzw. Tupelmengen zusammen mit dem jeweils noch verbleibenden Plan durch das Netzwerk von Operatoren und Peers „fließen“.

In Abbildung 2 ist dieses Prinzip anhand eines Beispiels illustriert. Gegeben seien die Relationen R und S mit dem Schema $\mathcal{R}(A, B)$ bzw. $\mathcal{S}(C, D)$ und der in der Abbildung gegebenen Verteilung sowie der Plan $\sigma_{2\leq A\leq 5}(R) \bowtie_{B=C} S$. Der Anfragepeer p_0 identifiziert anhand der Bedingung $2 \leq A \leq 5$ das Z-Kurvenintervall und damit die korrespondierenden Peers (hier p_1, p_2 und p_3). An diese Peers wird nun der Plan gesendet und dort die Selektion auf den lokal verfügbaren Daten ausgeführt. Die jeweils selektierten Tupel t werden dann zusammen mit dem verbleibenden Plan an die durch $h(\mathcal{S}, t(B))$ adressierten Peers mit Fragmenten der S-Relation (hier p_4 und p_5) gesendet, da nur hier die passenden Verbundkandidaten vorhanden sein können. Anschließend wird der Verbund jeweils lokal ausgeführt und – da der Verbund gleichzeitig die Wurzel im Plan dargestellt – die Ergebnisse an den Anfragepeer übergeben.

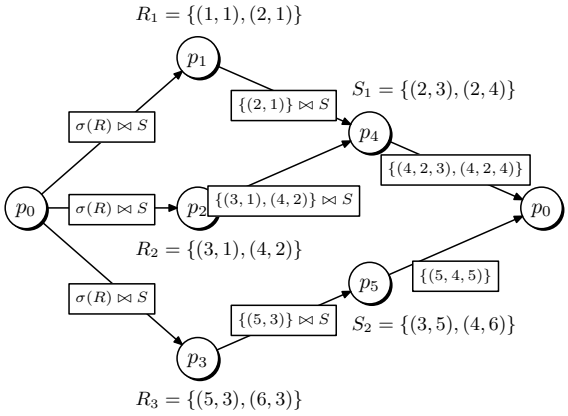

Abbildung 2: Beispiel einer Anfrageverarbeitung

Ein Problem dieser zustandslosen Verarbeitung ist jedoch das Erkennen des Endes einer Eingabe. Da Operatoren die Eingabe von ihren Kindern im Plan nicht anfordern sondern „unaufgefordert“ oder sogar tupelweise zugesendet bekommen, ist das Ende einer (Zwischen-)Relation nicht bekannt – schließlich können Tupel aufgrund von Kommunikationsproblemen auch verzögert eintreffen. Besonders deutlich wird dieses Problem bei blockierenden Operatoren wie Sortierung oder Gruppierung. Speziell für die Gruppierung haben wir daher die Variante einer „vorzeitigen Aggregation“ [WZ99] mit schrittweiser Verfeinerung des Ergebnisses gewählt. Dies bedeutet, dass ein Gruppierungspeer sein aktuelles Zwischenergebnis nach einer bestimmten Anzahl gesammelter Tupel oder nach einer bestimmten Zeit weiterleitet. Das Endergebnis (etwa eine Aggregation) wird somit durch schrittweise eintreffende Zwischenergebnisse ergänzt bzw. erweitert. Die Anfrageausführung kann dann mit einem Zeitlimit abgebrochen werden, wobei jedoch keine Garantie für die Vollständigkeit zu diesem Zeitpunkt gegeben werden kann.

6 Adaptive Anfrageoperatoren

Die im vorigen Abschnitt vorgestellte Ausführungsstrategie erfordert eine Vorabplanung der Anfrage. Dies kann gerade in einem großen P2P-System zu Problemen führen. So sind die für einen Kostenvergleich äquivalenter Pläne notwendigen Statistiken und Kostenparameter ohne globales Wissen nur schwer zu ermitteln bzw. zu pflegen – auch wenn man eher statische Datenbestände annimmt. Weiterhin kann sich ein zuvor als optimal gewählter Plan aufgrund der Dynamik des Systems während der Ausführung als suboptimal herausstellen, wenn die der Planung zugrunde liegenden Annahmen nicht mehr zutreffen.

Eine mögliche Alternative stellt daher die adaptive Anfrageverarbeitung [GPFS02] dar, die die Aufhebung der Separierung von Optimierung und Ausführung verfolgt. Der Eddy [AH00] ist eine der „aggressivsten“ Umsetzungen einer adaptiven Anfrageverarbeitung. Er ermöglicht es, die Abarbeitungsreihenfolge der Operatoren einer Anfrage kontinuierlich neu zu ordnen. Die Granularität reicht dabei bis auf Tupelebene, d.h., für jedes Tupel kann die Operatorreihenfolge getrennt festgelegt werden. Der Eddy ist eine Art Verteiler oder Pipeline, der die Tupel mit Hilfe verschiedener Routing-Strategien zu den einzelnen Operatoren leitet. Das Routing basiert vor allem auf Laufzeitstatistiken wie der Warteschlangenlänge und der Selektivität.

Unsere nachfolgend vorgestellte Umsetzung einer adaptiven Anfrageverarbeitung passt den Eddy-Mechanismus an die Charakteristika von P2P-Systemen an. Wir versuchen mit dem P2P-Eddy soweit wie möglich, alle Peers gleichberechtigt in die Anfrageverarbeitung mit einzubeziehen und verzichten vollständig auf eine globale Koordination. Anstatt die Informationen, welche Operationen auf den Tupeln ausgeführt werden müssen, in einer zentralen Komponente oder dedizierten Peers zu hinterlegen, tragen hier die Tupel selbst diese Information. Diese Art Laufzettel für die Tupel wird in der weiteren Ausführung als Todo-Liste bezeichnet.

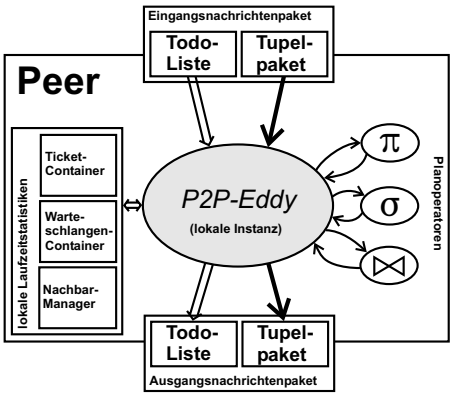

Abbildung 3: Prinzip des P2P-Eddy

Unter einem Nachrichtenpaket wird daher die Kombination aus Todo-Liste und Tupel verstanden. Eine Todo-Liste enthält immer nur die Operatoren einer Anfrage, die auch tatsächlich für das zugehörige Tupel gelten. Welche Operatoren das sind, wird durch den Anfragebaum vorgegeben. Alle Operatoren, die auf die Tupel einer Ursprungsrelation angewendet werden müssen, liegen auf dem Pfad zwischen Wurzel und der entsprechenden Relation. Pro Ursprungsrelation gibt es also eine Todo-Liste, die im Wesentlichen durch eine Tiefensuche im Anfragebaum erzeugt wird. Durch Verzweigungen im Baum aufgrund binärer Operatoren sind allerdings nicht unbedingt alle Operatoren auf diesem Pfad den Tupeln einer Ursprungsrelation zuzuordnen. Diese Operatoren müssen in einem Zwischenschritt herausgefiltert werden.

Um Verletzungen der Umformungsregeln der Relationenalgebra zu vermeiden, benötigt

jeder Operator zusätzlich ein Ready-Bit, welches angibt, ob ein Operator aus der Liste ausgeführt werden darf. Nach der Abarbeitung eines Operators wird dieser einfach aus der Todo-Liste entfernt und die Ready-Bits der restlichen Operatoren aktualisiert. Binäre Operatoren erfordern eine besondere Beachtung. Da hier Tupel unterschiedlicher Herkunft miteinander verknüpft werden, müssen auch deren Todo-Listen auf geeignete Weise verschmolzen werden, was einer Mengenvereinigung entspricht. Die Ergebnisliste wird dann möglichen Ergebnistupeln zugeordnet.

Für das Routing eines P2P-Eddy müssen zwei Fragen beantwortet werden: *(1)* Welcher Operator soll als nächstes ausgeführt werden (Operator-Routing)? *(2)* Zu welchem Peer soll das Nachrichtenpaket geschickt werden (Peer-Routing)? In beiden Fällen soll das Ziel verfolgt werden, möglichst wenig Overhead zu erzeugen.

Beim *Operator-Routing* sind verschiedene Strategien möglich, wovon wir die folgenden implementiert haben: *(1)* die zufällige Auswahl eines Operators, *(2)* die Auswahl nach Priorität der Operatortypen (womit sich Heuristiken wie „Selektion vor Join“ einfach umsetzen lassen, *(3)* die Auswahl nach Länge der Eingangswarteschlange als Maß für die Kosten eines Operators, *(4)* die Auswahl nach Selektivität auf Basis eines Ticket-Mechanismus sowie *(5)* die Suche nach dem „nächsten“ Join, d.h. bei dem die Distanz zum Ziel-Peer für die Neuverteilung des Tupels am geringsten ist. Durch die nahezu beliebige Verteilung der Operatoren im Netz sind auch Kenngrößen wie Warteschlangenlänge und Ticket auf den verschiedenen Peers verteilt. Die Entscheidungsfindung für das Operator-Routing ist also ein lokaler Prozess, sodass verschiedene Peers für gleiche Todo-Listen auch unterschiedliche Entscheidungen treffen können.

Für das *Peer-Routing* muss zunächst die Frage beantwortet werden, ob ein Nachrichtenpaket überhaupt an einen anderen Peer geschickt oder auf demselben Peer weiter abgearbeitet werden soll. Für einen dynamischen Ansatz wird dafür ein Parameter benötigt, der eine Aussage ermöglicht, welches Vorgehen sinnvoll ist. Ganz intuitiv bietet sich dafür die aktuelle Auslastung (z.B. als Summe aller Warteschlangenlängen) des Peers an. Überschreitet die Auslastung einen gewissen Wert, wird das Nachrichtenpaket verschickt.

Wird ein Nachrichtenpaket verschickt, muss noch das Ziel bestimmt werden. Da jeder Peer nur seine direkten Nachbarn und deren Bereiche im CAN-Koordinatenraum kennt, bietet sich am sinnvollsten ein Nachbar als Ziel-Peer an. Hierfür haben wir verschiedene Strategien implementiert, wie z.B. zufällige oder zyklische Auswahl sowie schnellste Verbindung. Da wir gegenwärtig aber noch keine dynamischen Lastszenarien berücksichtigen, werden wir das Peer-Routing an dieser Stelle nicht weiter vertiefen.

7 Evaluierung

In einer Reihe von Experimenten haben wir das Leistungsverhalten der vorgestellten Operatorimplementierungen und Anfragestrategien untersucht. Die wesentliche Frage war dabei, ob ein DHT-basiertes P2P-System tatsächlich für die Verwaltung und Anfrage relationaler Datenbestände geeignet ist und insbesondere die Erwartungen hinsichtlich der Skalierbarkeit erfüllt. Aus Platzgründen können hier jedoch nur ausgewählte Ergebnisse

präsentiert werden. Detaillierte Resultate können den Untersuchungen in [vdW04, Rö05] entnommen werden.

Die Experimente wurden mit einem im Java implementierten Anfrageprozessor auf Basis eines ebenfalls Java-basierten CAN-Prototyps durchgeführt. Die Netzgröße wurde dabei im Bereich von 100 bis 10.000 Peers gewählt. Als Daten wurden die TPC-H-Daten für 1 MB verwendet, die von den Peers jeweils in einfachen Hauptspeicherstrukturen gehalten wurden. Auf dem TPC-H-Schema wurde ein Mix von 10 verschiedenen Anfragen ausgeführt, die Punkt- und Bereichsanfragen, Verbunde von bis zu 4 Relationen sowie einfache Gruppierungen mit Aggregaten umfassen. Als Maß für den Aufwand wurde jeweils die Anzahl der Hops ermittelt.

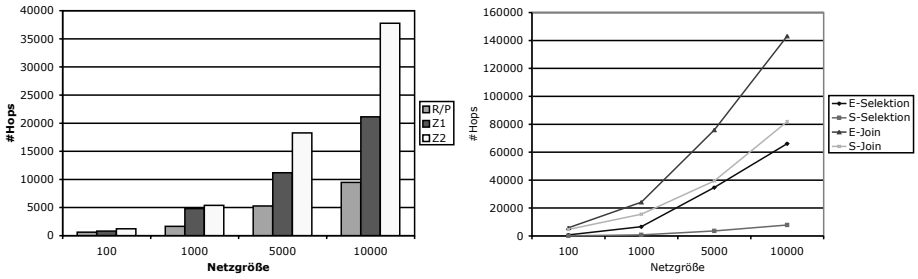

Abbildung 4: Evaluierungsergebnisse

Im ersten Experiment sollte die Frage beantwortet werden, wie sich eine Verteilung einzelner Relationen über mehrere Peers auswirkt. Dafür wurde der in Abschnitt 3 erwähnte Spreizfaktor für die Berechnung der Z-Kurve variiert, Z1 entspricht dabei einer Verteilung einer Relation auf 10-20 Peers, Z2 dagegen auf ca. 300-400 Peers. Auf den sich daraus ergebenden Datenverteilungen wurde jeweils der Anfragemix für unterschiedliche Netzgrößen ausgeführt (Abb. 4 links). Zum Vergleich wurde noch eine Verteilungsstrategie gewählt, bei der jede Relation komplett auf einem Peer gespeichert ist (R/P). Die R/P-Strategie ist natürlich keine realistische Annahme, da ja gerade das Ziel in einer verteilten Speicherung besteht. Bei großen Lasten (viele parallele Anfragen) würden Peers mit häufig verwendeten Relationen schnell überlastet.

Das Ergebnis gibt gleichzeitig auch einen groben Aufschluss über die Skalierbarkeit bis zu 10.000 Peers: Wie erwartet wächst der Aufwand logarithmisch mit der Netzgröße, wobei die einzelnen Operatoren teilweise jedoch unterschiedlich starken Einfluss haben.

Zur Bewertung der Eddy-Variante wurde ein weiteres Experiment durchgeführt. Hierzu wurde das gleiche Netz mit den gleichen Datenverteilungen und Anfragen eingesetzt und ein Vergleich zwischen statischer Anfragestrategie (S-Selektion bzw. S-Join) und dem Eddy-Ansatz vorgenommen (S-Selektion bzw. S-Join), jeweils getrennt für mehrere Selektions- und Verbundanfragen (Abb. 4 rechts). Der Aufwand für die Eddy-Variante wächst dabei im gleichen Maße wie die statische Strategie (jedoch mit größerem Overhead). Vorteile sind daher nur für dynamische Lastszenarien zu erwarten, die wir in aktuellen Arbeiten untersuchen.

8 Ausblick

In diesem Beitrag haben wir erste Ergebnisse unserer Arbeiten zu einem DHT-basierten Anfragesystem vorgestellt. Ziel ist es dabei, DHT-Techniken für die Verwaltung (zunächst) relationaler Daten einzusetzen und somit Anwendungen zu ermöglichen, die komplexe, deklarative Anfragen auf großen, Internet-weit verteilten Datenbeständen ausführen. Die bisherigen Ergebnisse illustrieren die grundsätzliche Machbarkeit eines derartigen Ansatzes, zeigen aber auch noch Verbesserungspotenzial. So hat – wie erwartet – die Datenverteilung einen großen Einfluss, insbesondere bei Bereichs- und Verbundanfragen. Auch hat sich gezeigt, dass Rehashing sehr aufwendig werden kann, so dass hier Alternativen untersucht werden müssen. Weiteres Optimierungspotenzial besteht auch bei Bereichsanfragen, indem eine engere Integration mit den Routing-Verfahren des DHTs erfolgt. Darüber hinaus erfordert eine exakte Beurteilung der vorgestellten Techniken auch noch eine adäquate Berücksichtigung realistischer Lastszenarien.

Literatur

[AH00] Avnur, R. und Hellerstein, J. M.: Eddies: Continuously Adaptive Query Processing. In: *Proc. ACM SIGMOD 2000, Dallas, Texas*. S. 261–272. 2000.

[AX02] Andrzejak, A. und Xu, Z.: Scalable, Efficient Range Queries for Grid Information Systems. In: *Proc. of the 2nd Int. Conf. on Peer-to-Peer*. 2002.

[Br84] Bratbergsengen, K.: Hashing Methods and Relational Algebra Operations. In: *VLDB'84, Singapore*. S. 323–333. 1984.

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

[Gr93] Graefe, G.: Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*. 25(2):73–170. June 1993.

[HHL+03] Huebsch, R., Hellerstein, J. M., Lanham, N., Thau Loo, B., Shenker, S., und Stoica, I.: Querying the Internet with PIER. In: *VLDB 2003, Berlin, Germany*. S. 321–332. 2003.

[RFH+01] Ratnasamy, S., Francis, P., Handley, M., Karp, R., und Shenker, S.: A Scalable Content Addressable Network. In: *ACM SIGCOMM 2001*. S. 161–172. 2001.

[Rö05] Rösch, P.: Ein Anfrageprozessor für CAN-basierte P2P-Systeme. Master's thesis. TU Ilmenau, Fakultät Informatik und Automatisierung. 2005.

[THH+04] Thau Loo, B., Hellerstein, J., Huebsch, R., Shenker, S., und Stoica, I.: Enhancing P2P File-Sharing with an Internet-Scale Query Processor. In: *VLDB 2004, Toronto, Canada*. S. 432–443. 2004.

[TP03] Triantafillou, P. und Pitoura, T.: Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In: *DBISP2P 2003, Berlin, Germany*. 2003.

[vdW04] von der Weth, C.: Dyamische Anfrageoperatoren für CAN-basierte P2P-Systeme. Master's thesis. TU Ilmenau, Fakultät Informatik und Automatisierung. 2004.

[WA93] Wilschut, A. und Apers, P.: Dataflow Query Execution in a Parallel Main-Memory Environment. *Distributed and Parallel Databases*. 1(1):103–128. 1993.

[WZ99] Wang, H. und Zaniolo, C.: User-Defined Aggregates in Database Languages. In: *DBLP'99*. LNCS. S. 43–60. Springer. 1999.