Problemseminar "Peer-to-Peer (P2P) Data Management" Thema:Verarbeitung komplexer Queries in schemabasierten P2P-Netzwerken

Sebastian Stoll
Betreuer: Timo Böhme

6. Januar 2004

Inhaltsverzeichnis

1 Einleitung

In den letzten Jahren haben Peer-to-Peer (P2P) Systeme eine rasante Entwicklung vollzogen. Angefangen bei einfachen Architekturen wie Napster und Gnutella sind, unter anderem durch die Verbesserungen von Routing und Topologie, die P2P-Systeme aus den Bereichen Content-Sharing und Distributed Computing nicht mehr wegzudenken. Unter einem Peer-to-Peer Netzwerk versteht man dabei einen Verbund von Rechnern/Peers die sowohl die Funktionen eines Client, als auch die eines Servers ausüben. Dabei sind die einzelnen Peers autonom und tauschen Daten direkt untereinander aus. Aufgrund dieser Autonomie ist allerdings nicht sichergestellt, das jeder Knoten zu jedem Zeitpunkt dem Netz zur Verfügung steht.

Im Bereich der P2P-Systeme ist bis jetzt ein Hauptgebiet der Datenbankforschung nicht ausreichend bzw. nur in Ansätzen berührt wurden - die Beschreibung von Daten mit Schemabeschreibungssprachen und die Verarbeitung komplexer Queries. In solch einem Szenario wäre es möglich, auf weltweit verteilten Datenbeständen, die mit heterogenen Schemas oder verschiedenen Schemabeschreibungssprachen ausgezeichnet sind, Anfragen durchzuführen. Die Ergebnisse dieser Anfragen könnten dann wiederum verteilt aggregiert oder zwecks Distributed Computing zwischen den einzelnen Knoten bzw. Peers ausgetauscht werden. Dabei können diese Daten nicht nur textueller Form oder File-Sharing Inhalte sein, sondern auch Webseiten, Bilder oder Kollektionen aller Art.

In dieser Ausarbeitung, welche im Rahmen des Problemseminars "Peer-to-Peer(P2P) Data Management", in der Abteilung Datenbanken, an der Universität Leipzig entstanden ist, möchte ich auf Anforderungen an und Techniken für schemabasierte P2P-Systeme eingehen. Weiterhin werden aktuelle Forschungsarbeiten, die einen Focus auf schemabasierte P2P-Systeme setzen, vorgestellt. Dabei sollen die Systeme in ihren Grundzügen erläutert und im speziellen die Query-Verarbeitung besprochen werden. Abschliessend wird ein Vergleich dieser Systeme gezogen und ein Ausblick auf zukünftige Entwicklungen gegeben.

2 Abgrenzung zu herkömmlichen Ansätzen

Da im Grunde genommen alle P2P-Systeme ihre Daten auszeichnen [1] und zum Teil ein globales Schema besitzen, ist es notwendig eine Abgrenzung zwischen diesen herkömmlichen und wirklich schemabasierten P2P-Systemen, wie ich sie in dieser Arbeit erläutere, vorzunehmen. Der wichtigste Unterschied ist das Vorhandensein eines komplexen Schemas. Unter komplex ist zu verstehen, das eine Schemabeschreibungssprache [2] existiert, mit der eigene Schemas definiert und bestehende Schemas beliebig erweitert werden können. Bei herkömmlichen Systemen wie Gnutella oder eMule sind dagegen die Schemas nicht einsehbar oder können nicht durch den Nutzer verändert werden. Durch die Möglichkeit zur Definition eigener Schemas ist ein schemabasiertes System heterogen, das heißt verschiedene Peers verfügen über verschiedene Sichten auf ihre Daten. Diese können auch identisch sein.

Bedingt durch das Vorhandensein eines komplexen Schemas, steigt auch die Fähigkeit eines System seine Inhalte mit Informationen (Metadaten) auszuzeichnen. Eine Datei beziehungsweise beliebiger Content kann, anstatt wie bei File-Sharing Systemen nur mit Name und ID, auch mit anderen Informationen wie zum Beispiel Erzeuger, Inhalt, Fachbereich und so weiter ausgezeichnet werden.

[1] wenn auch zum Teil nur mit dem Filenamen

[2] zum Beispiel RDFS

Die Verknüpfung von komplexen Schemas und Auszeichnung von Content ergibt bei schemabasierten Systemen erheblich mehr Spielraum bei der Query-Formulierung. Wenn man bei einem bestehenden Filesharing-System nach etwas sucht, gibt man ein Keyword ein. *Linux* als Suchanfrage gibt jeglichen Content wieder, der *Linux* im Namen enthält. Teilweise werden Restriktionen erlaubt, wie zum Beispiel einen Bereich für die Dateigröße anzugeben [3]. Dies führt aber immer noch zu großen Antwortmengen. Will man nun ein spezielles Buch zu Linux, von einem bestimmten Autor, suchen, kann man in einem schemabasierten System den Autor mit angeben und zum Beispiel wie bei PIER [22] die Anfrage im SQL-Format stellen[4]. Viele Systeme und Ansätze stellen dabei neben Selektion und Projektion noch weitere Operatoren zur Verfügung. Dies sind Range, Gruppierungs und Joinoperatoren sowie Aggregat -und Sortierfunktionen. Darüber hinaus werden für eine Query-Abarbeitung oft mehrere Peers involviert, so das zum Beispiel Join-Operationen theoretisch über den gesamten verteilten Datenbestand möglich sind [14].

Eine weitere Möglichkeit der Abgrenzung besteht in der Betrachtung der verwendeten Topologien in Ansätzen zu schemabasierten P2P-Systemen. Dort handelt es sich meist um mehrschichtige Architekturen [22, 20] die als unterste Schichte eine Distributed Hash Table-Technolgie [23, 17] einsetzen um dem darüberliegenden P2P-Netz eine Struktur, was Daten und Informationsverteilung betrifft, zu geben. Man spricht dann auch von strukturierten P2P-Netzen. Dies in Verbindung gebracht mit Routing Indizes, die Informationen über verwendete Schemas speichern [14], vermeidet das *fluten* des gesamten Netzes mit Anfragen. Eine Weiterleitung erfolgt nur an die Peers, die für eine positive Antwort auch in Frage kommen.

Systeme dieser Art, die relationale Anfragen und das Erzeugen eigener Schemas unterstüzen, sowie eine Struktur besitzen, klingen nach verteilten Datenbanksystemen, verlassen den Kontext der P2P-Systeme aber nicht. Auch schemabasierte P2P-Systeme verfügen über kein globales Wissen und die Peers besitzen weiterhin ihre Autonomie. Die sich dynamisch verändernden Datenbestände sowie das ständige Kommen und Gehen von Peers verhindert statische Query-Optimierung und einen Quality of Service. Nichtsdestotrotz besitzen schemabasierte P2P-Systeme Vorteile gegenüber herkömmlichen Ansätzen, die vor allem aus der oben geführten Abgrenzung hervorgehen. Dies sind:

- Unterstützung heterogener komplexer Schemas
- erweiterbare Auszeichnung von Ressourcen
- umfangreichere Anfragemöglichkeiten
- besseres Query-Routing

3 Anforderungen an schemabasierte P2P-Systeme

Um eine Infrastruktur für ein effizientes schemabasiertes P2P-System aufzubauen, müssen bestimmte Anforderungen erfüllt sein. Dabei sind mehrere Grundbausteine eines solchen Systems zu betrachten. Man muss sich mit der Netzwerktopologie und der Ressourcenallokation innerhalb des Netzes bzw. dem Query-Routing befassen. Weiterhin wird eine Sprache zur Beschreibung der Schemas benötigt, um damit die Daten die im P2P-Netzwerk vorhanden sind auszuzeichnen. Eine Anfragesprache mit der Queries formuliert werden und Algorithmen um

[3] wie im MLDonkey-Netzwerk

[4] Das Schema muss natürlich dem Kontext gerecht werden

diese zu optimieren stehen ebenfalls zur Disposition. Dabei entstehen zwangläufig Probleme die Performance und Anfrageergebnisse beeinträchtigen. Die hier vorgenommene Einteilung in Netzwerktopologie, Ressourcenallokation, Schemabeschreibungssprachen und Queryverarbeitung soll nicht als strikte Trennung von Anforderungsbereichen zu verstehen sein, es bestehen natürlich Überlappungen zwischen diesen.

3.1 Netzwerktopologie

Einer der wichtigsten Punkte von denen die Effiziens eines P2P-Systems abhängt ist die Organisation der Peers im Netzwerk. Dabei sind mehrere Herausforderungen zu meistern, die gleichermaßen für schemabasierte wie nichtschemabasierte P2P-Systeme gelten. Von größter Wichtigkeit ist dabei die Skalierbarkeit des Systems. Es muss möglich sein, ein Netzwerk mit einer großen Anzahl von Peers aufzubauen, ohne das dabei die Rechenlast beziehungsweise der Traffic für einen Rechners ins Unendliche steigt. Der Nachrichtenaufwand für Aufbau und Maintenance des P2P-Netzwerkes sollte dabei möglichst gering ausfallen. Dabei ist anzunehmen das die Teilnehmer des Netzwerkes geographisch verteilt und über Datenleitungen verbunden sind. Des weiteren muss die Architektur mit der Heterogenität bezüglich Software, Betriebssystem und verfügbarer Bandbreite der Peers zurechtkommen. Es ist zum Beispiel möglich das sich sowohl normale PC's als auch PDA's oder andere mobile Einheiten im Netzwerk befinden. Aufgrund der Autonomie der Peers muss außerdem mit periodisch das P2P-System verlassenden und betretenden Teilnehmern umgegangen werden. Diese Zu- und Abgänge müssen den anderen Netzteilnehmern bekannt gemacht werden und es muss sich um eine eventuelle Replikation von Ressourcen gekümmert werden. Dies könnten zum Beispiel Indizes für das Routing sein. Sollte es Peers geben die Schemas anderer Peers verwalten, müssen Mechanismen vorhanden sein um die Schemas der hinzugekommener Peers zu verarbeiten und zu integrieren. Wenn ein Peer das Netzwerk wieder verläßt sollten seine Schema-Einträge sofort oder nach einer bestimmten Zeit entfernt werden.

3.2 Ressourcenallokation

Das Auffinden von Daten beziehungsweise die Propagierung von Wissen über Daten in einem P2P-Netzwerk ist von größter Wichtigkeit. Nutzer, die einander unbekannt sind, möchten Content austauschen beziehungsweise miteinander kommunizieren. Dafür benötigen sie Informationen über Nutzer, die sich gerade im Netzwerk befinden und über die Daten welche diese besitzen. Dies geschieht über Lookup-Funktionen innerhalb des P2P-Netzwerkes. Dabei muss das P2P-System über effiziente Suchmechanismen verfügen um dem Suchenden ein schnelles und zufriedenstellendes Ergebnis zu liefern. Im weiteren muss es dem dynamischen Character des Netzes Rechnung tragen und die sich verändernden Positionen von Daten aktualisieren und nicht mehr vorhandene Daten aus Suchdatenbanken, oder ähnlichem, entfernen. Darüberhinaus sollten die Daten jedes Peers von jedem anderem Peer aus erfragbar sein. Damit schließt sich auch die Forderung nach Zugang zu diesen Daten an, sofern der Teilnehmer, der diese besitzt, bekannt ist. Da die Nutzer unter Umständen einander unbekannt waren, muss das P2P-System für den Austausch der Verbindungsdaten sorgen. Die Ressourcenallokation sollte schnell vonstatten gehen und der durch Such -oder Informationsdaten verursachte Traffic sollte möglichst gering gehalten werden.

3.3 Schemabeschreibungssprachen und Schemaverarbeitung

Als grundlegender Bestandteil der schemabasierten P2P-Systeme fungiert die Schemabeschreibungssprache. Sie dient zum Auszeichnen von Content und damit als Schnittstelle zu den Informationsbedürfnissen der anderen Peers. Die Schemas sind der Ausgangspunkt für die Query-Erstellung der P2P-Anwendung. Die verwendete Beschreibungssprache sollte dabei zur Darstellung von Abhängigkeiten und hierarchischen Beziehungen der ausgezeichneten Objekte untereinander fähig sein. Die Schemas sollten nachträglich erweiterbar und auch verteilt unter den Peers abspeicherbar sein. Somit wäre es zum Beispiel möglich verteilte Repositories aufzubauen. Weiterhin sollte ein P2P-System mehrere Schemabeschreibungssprachen unterstützen, damit Peers, die unterschiedliche Sprachen benutzen, kommunizieren und in einem Netzwerk integriert werden können. Dies erfordert wiederum Algorithmen für Mapping beziehungsweise Transformationen. Die Integration mehrerer Schemas in ein Schema sollte möglich sein, um die Datenbestände mehrerer Peers zu aggregieren.

3.4 Query-Verarbeitung

Um Daten innerhalb des P2P-Netzes zu finden muss Query-Funktionalität zur Verfügung gestellt werden. Anfragen sollen in möglichst kurzer Zeit alle Peers erreichen die für die Query in Frage kommen. Dabei wird neben einem leistungsfähigen Routing-Algorithmus auch ein Query Processing benötigt, das für die Anfragen einen möglichst effizienten Ausführungsplan erstellt. Damit soll die Anzahl der nötigen Nachrichten und der Netztraffic der bei den Peers entsteht gering gehalten werden. Anfragemöglichkeiten einfacher P2P File-Sharing Systeme erschöpfen sich bei Keyword-Anfragen und erlauben, abgesehen von Dateigröße oder Dateierweiterung, keine großen Möglichkeiten der Selektion. Ein schemabasiertes P2P-System sollte daher den gleichen Spielraum wie relationale Anfragesprachen bieten und Queries über mehrere Attribute, Attribut-Wertebereiche sowie Joins, Selektionen und Aggregatfunktionen unterstützen. Wenn ein schemabasiertes P2P-System mehrere Schemasprachen beinhaltet, muss auch die Anfragesprache diese unterstützen beziehungsweise eine entkoppelte Anfragesprache zur Verfügung stellen, die dann Anfragen in die jeweiligen Zielsprachen transformiert.

3.5 Probleme

Bei der Realisierung dieser Anforderungen kommt es zu Problemen, die auch in nicht schemabasierten Systemen auftreten und durch Merkmale der P2P-Systeme bedingt sind. Dies sind der verursachte Traffic, der durch die Übertragung von Daten, Steuerungsinformationen und Queries entsteht. Des Weiteren kann es bei der Allokation von Daten dazu kommen, dass bestimmte Peers bei einer Suche nicht erreicht werden. Dies kann durch TimeToLive-Einstellungen der Queries bedingt sein, oder mit der Netzwerktopologie zusammenhängen. Speziell aus der Schemabasiertheit ergeben sich Probleme bei der Query-Optimierung, bei Transformationen und der Integration der Schemas. Man stelle sich in diesem Zusammenhang ein P2P-System vor, in dem eine Schemabeschreibungsprache jedem Peer erlaubt, seine Daten mit mit eigenen Schemas zu beschreiben. Will nun ein Peer A an einen Peer B eine Anfrage stellen, wird er diese auf seinem eigenem Schema formulieren. Das P2P-System muss diese Anfrage nun so transformieren, das sie auf das Schema von B angewendet werden kann. Mit den Ergebnissen die B zurückliefert, muss das Gleiche geschehen. Sofern es Peers gibt, die Schemas von verschiedenen Peers zusammenfassen und eine Informationsschnittstelle für andere Peers/Clients zur Verfügung zu stellen, muss es Integrationsmechanismen geben. Damit

nicht unnötig Bandbreite verschenkt wird, müssen die Anfragen zudem so optimiert werden, das die Daten möglichst von schnellen, zuverlässigen Peers bezogen werden und eine Lastverteilung berücksichtigt wird. Aufgrund von nicht vorhandenen globalen Informationen über Bandbreiten, verwendete Schemas und Verfügbarkeit von Peers, stellen sowohl die Transformationen als auch die Integration und Optimierung nicht triviale Probleme dar. Abhängig davon müssen Entscheidungen beim Design eines Systems bezüglich Topologie, verwendeten Schemasprachen und Ausprägung der Schemabasiertheit getroffen werden.

4 Techniken für schemabasierte P2P-Systeme

Im folgendem werden Techniken, die bei den zu besprechenden Forschungsansätzen Verwendung finden, erläutert. Es handelt sich dabei um die Schemabeschreibungssprachen RDF/RDFS und OIL. Im weiteren werden Distributed Hashtables und die P2P-Protokolle CAN und CHORD, die darauf basieren, besprochen. Sie stellen für viele Ansätze eine Basis dar, auf der weitere Architekturschichten aufgesetzt werden. Es ist darauf hinzuweisen das dies nur einen Teil des aktuellen Standes der Technik wiederspiegelt und damit keineswegs vollständig ist.

4.1 RDF/RDFS

Das Web besteht aus Millionen von Seiten, Links und Daten auf die man, wenn man die Adresse weiss, zugreifen kann. Das Problem hier und in den bisherigen P2P-System war, dass kaum Metadaten, das heißt Information über Information, existierten. Das W3C stellt mit dem RDF-*Resource Description Framework* [6] ein Framework zur Verfügung, um mit genau dieser Problematik umgehen zu können. Es soll die automatische Verarbeitung von Web-Ressourcen ermöglichen und kann in vielen Bereichen benutzt werden [7]. Dies sind zum Beispiel die Unterstützung von Search-Engines, die Beschreibung von Content und Beziehungen von Content innerhalb einer Webseite oder digitalen Bibliothek. Weiterhin besteht die Möglichkeit, RDF mit digitalen Signaturen einzusetzen und damit zum Beispiel e-commerce oder e-collaboration zu unterstützen. Das Datenmodell von RDF [8] ermöglicht die Beschreibung von Resourcen über Properties, wobei ein Property einen Typ und einen Wert besitzt. Für diese Beschreibung werden dann sogenannte RDF-Triple benutzt, die jeweils aus $< property, subject, value >$ bestehen. *subject* steht für die Ressource die beschrieben werden soll, *property* für die zugeornete Eigenschaft und *value* für den Wert. Es kann jede Ressource beschrieben werden, solange sie über ein URI-*unique resource identifier* identifizierbar ist. Die Werte der Properties können wiederum Ressourcen sein und somit Beziehungen zwischen Ressourcen ausdrücken. Allerdings verfügt das RDF-Datenmodell über keine Mechanismen diese Beziehungen zwischen Werten von Properties und Ressourcen zu definieren. Dies ist die Aufgabe der *RDFS* - RDF Schemas welche die Deklarationen von Properties und der damit zusammenhängenden Semantik übernimmt. Ein Schema beschreibt damit nicht nur die Properties einer Ressource, sondern unter Umständen auch die Art der zu definierenden Ressource.
Ein Beispiel für eine RDF-Auszeichnung in XML:
<rdf:Description about='http://www.Hans-Mueller.de'>
<Creator>Hans Müller</Creator>
</rdf:Description>
Und das gleiche Beispiel als RDF-Triple:

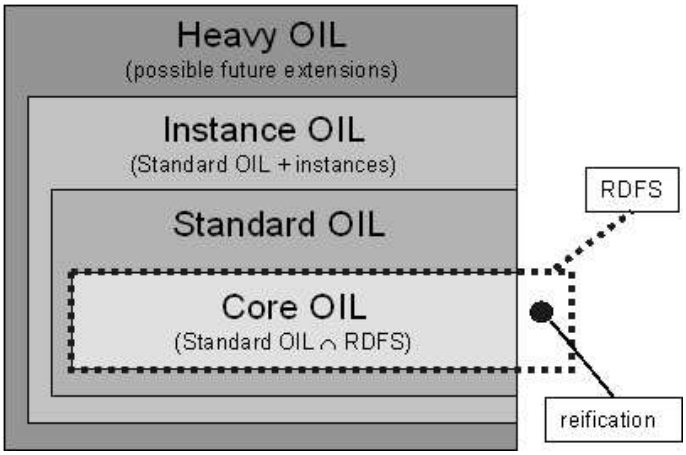

Abbildung 1: OIL-Architektur

{Creator,[http://www.Hans-Mueller.de],Hans Müller}
Hier wäre das *subject*:http://www.Hans-Mueller.de, die *property*:Creator und *value*: Hans Müller. Zu den Eigenschaften von RDF zählt, das RDF-Descpriptions in XML konvertiert werden können und damit leicht austauschbar sind. Darüber hinaus kann jeder Properties, Ressourcen und Schemas erzeugen.

4.2 OIL

Auf dem Weg zu einem Semantic Web muss es Möglichkeiten geben Ontologien zu beschreiben. Unter einer Ontologie versteht man dabei eine Sammlung von Konzepten und den Relationen zwischen eben diesen. Eine Ontologie muß außerdem formal definiert sein, damit sie im Rahmen eines wissensbasierten Systems computergestützt verarbeitet werden kann [13]. Eine Möglichkeit dafür bietet sich mit OIL-*Ontology Interchange Language* [9]. Diese baut auf RDF auf und bietet eine Schichtenarchitektur an, wobei jede Schicht aus einer Untersprache besteht(siehe Abbildung). Je höher die Schicht, desto mehr Funktionen werden zur Verfügung gestellt und desto höher ist auch die Komplexität. Dabei ist es Menschen und Maschinen, die nur eine niedrige Schicht verstehen, möglich Ontologien teilweise zu verstehen, die aus höheren Schichten stammen. Dies würde bedeuten das zum Beispiel Systeme die nur RDF verstehen, fähig wären OIL-Ontologien zu verarbeiten, zumindest im Bereich ihrer beschränkten Möglichkeiten.

Core OIL [10] überschneidet sich zum größten Teil mit RDFS, was RDF Schema Agents ermöglicht diese Ontologien zu verarbeiten. **Standard OIL** [11] umfaßt die Grundfunktionen um adequate Ausdruckskraft, präzise Beschreibung der Semantik und vollständige Deduktion zu ermöglichen. **Instance OIL** [12] Ermöglicht Integration über Datenbankfähigkeiten. Es besitzt das selbe Schema wie Core OIL. **Heavy OIL** ist noch nicht spezifiziert wurden und soll Möglichkeiten für Erweiterungen bietet. Eine OIL-Definition besteht aus Klassen und Beziehungen. Die Klassen werden durch Eigenschaften beschrieben und Beziehungen stellen das Verhalten von Objekten die zu einer Klasse gehören dar.

Ein Beispiel wäre die Klasse *animal* und deren Subklassen $giraffe$ und *lion* [24].
class-def animal
class-def giraffe
    subclass-of animal
class-def lion
    subclass-of animal

4.3 Distributed Hash Tables

Aufgrund der nicht vorhandenen Kenntnis über die komplette Struktur und den Umfangs eines P2P-Netzes, ist es nicht möglich bei jedem Peer alle Information über die im Netz vorhandene Ressourcen zu speichern. Diese Problematik wird durch den ständigen Ein- und Austritt von Peers im Netz noch verschärft. Eine Möglichkeit damit umzugehen, und einen effizienten Lookup-Dienst für Ressourcen und Peers zur Verfügung zu stellen, bieten Distributed Hash Tables(DHT). Dabei werden mit Hilfe eines Hashing Algorithmus Key-Value Paare erzeugt, wobei sowohl die Identifier der Knoten, zum Beispiel IP-Adresse, als auch Identifier für die Ressourcen, zum Beispiel der Name, gehasht wird. Da es natürlich in einem dezentralen, verteilten Netz nicht möglich ist diese Hashtable bei jedem Peer komplett zu speichern, wird sie verteilt gespeichert. Jeder Peer verfügt dann über eine Menge von Nachbarn, deren Key-Value Paare er abgespeichert hat. Aufgrund der Struktur des Netzes werden bestimmte Objekte bestimmten Peers zugeteilt, wobei diese dort physisch oder als Referenz auf einen anderen Peer abgelegt werden. Wenn eine Anfrage für eine Ressource bei einem Knoten eintrifft, wird anhand einer Ähnlichkeits bzw. Entfernungsfunktion bestimmt welcher von der Nachbarn des angefragten Knotens näher an der gesuchten Ressource liegt. An diesen wird die Anfrage dann weitergeleitet. Wenn der Peer mit der gesuchten Ressource gefunden wurde, wird eine Referenz, das Objekt selber oder anderweitige Information an den Suchenden zurückgegeben. Wie die Ähnlichkeitsfunktion arbeitet und das Routing erfolgt, hängt dabei von der Dimension und Struktur des Netzes ab. Es gibt eindimensionale Ringstrukturen wie CHORD [17] und zweidimesionales Netz-Routing wie CAN [23], die im Anschluss kurz vorgestellt werden. Da es bei diesen Ansätzen zu langen Suchpfaden kommen kann, wurden weiterhin Systeme für den n-dimensionalen Raum wie PASTRY [2], TAPESTRY [3] und PLAXTON [4] entwickelt.

4.4 CHORD

Die eindimensionale Struktur von Chord besteht aus einem Ring in dem sowohl die Identifier der Peers als auch die der Ressourcen angeordnet sind. Als Key fungiert ein m-bit Identifier, der über eine Consistent Hashing Funktion erzeugt wird.Dabei muss m so groß sein, dass verhindert wird das zwei Peers oder Ressourcen auf den selben Key gehasht werden. Bei einem m-bit Key sind demnach 0-2^m-1 Werte möglich. Die Identifier der Peers sind kreisförmig modulo 2^m angeordnet und werden durch hashen der IP-Adresse berechnet. Der Schlüssel k einer Ressource wird aus dem Ressourcen-Key erzeugt und im Uhrzeigersinn dem ersten Peer zugeordnet,dessen Identifier größer oder gleich dem der Ressource ist. Wenn dann ein Peer eine Suche startet, wird die Anfrage im Uhrzeigersinn weitergeleitet bis der entsprechende Peer, der die Ressource oder eine Referenz darauf besitzt, gefunden ist. Für dem Fall, dass ein Peer den Ring verlässt, werden alle seine Ressourcen dem nachfolgendem Peer zugeordnet. Wenn ein Peer hinzukommt, übernimmt er alle Ressourcen, die zwischen ihm und seinem

Vorgänger liegen. Da ein Suchpfad bei vielen Peers unter Umständen sehr groß werden kann, sind Optimierungen möglich. Zum Beispiel muss ein Peer nicht nur seinen direkten Nachfolger führen, sondern kann in logarithmisch ansteigenden Intervallen weitere Nachfolger führen. Eine Anfrage kann dann an den Nachfolger weitergeleitet werden, der dem Identifier der Anfrage am nächsten und kleiner ist. Die Komplexität für einen *exakt match lookup* beträgt somit O($logN$).

4.5 Content Addressable Network (CAN)

Einen zwei beziehungsweise mehrdimensionalen Ansatz verfolgt CAN. Hier wird ein d-dimensionaler virtueller Raum aufgespannt der in Zonen unterteilt ist. Jeder Peer verwaltet eine Zone und speichert die Ressourcen oder Referenzen die in seine Zone fallen. Jeder Peer speichert weiterhin Routing-Informationen für O(d) andere Peers. Wenn ein Peer eine Anfrage macht, wird diese zu einem Punkt im d-dimensionalen Raum geroutet. Aufgrund der Raumaufteilung wird die Anfrage immer an die Nachbarn weitergeleitet die dem Zielpunkt am nächsten liegen. Wenn der Peer erreicht ist, in dessen Zone der Punkt liegt, wird abgebrochen und die Ressource oder anderweitige Information zurückgegeben. Je höher die Dimension ist, desto kürzer ist auch der Suchpfad und die Kosten für einen *exact match lookup* liegen bei O($dN^{\frac{1}{d}}$).

5 Aktuelle Forschungsansätze

5.1 Verteilte Queries in schemabasierten Super - Peer - Technologien

In ihrem Edutella Projekt [14] stellen Brunkhorst, Dhraief, Kemper, Nejdl und Wiesner eine, auf einer schemabasierten P2P-Infrastruktur aufbauende, Architektur für das Semantic Web vor [16]. Diese basiert auf den W3C Standards RDF und RDFS um die verteilten Ressourcen zu beschreiben und nutzt das JXTA-Framework [18]. Im Anschluss wird der Aufbau des in Edutella verwendeten Super-Peer Netzwerkes und die Methodik für das Query-Routing erläutert. Die Aufstellung von Query-Plänen und deren Unterstützung durch Routing Indizes beziehungsweise Optimierung ist ebenfalls Gegenstand der Betrachtung.

5.1.1 Edutella Super-Peer Topologie

Super-Peer Netzwerke tragen der Tatsache Rechnung das die Bandbreite und Rechenleistung der einzelnen Netzteilnehmer stark schwanken kann. Daher werden diese in Peers (niedrige Leistungsfähigkeit) und Super-Peers (hohe Leistungsfähigkeit) eingeteilt. Ein Super-Peers verwaltet dabei eine bestimmte Anzahl von Peers und übernimmt Verantwortung für Peer-Integration, Query-Routing sowie die Vermittlung von Informationen. Die Super-Peers von Edutella sind in der HyperCuP Topology [19] organisiert. Damit kann für das Routing ein rekursiver Graph-*Hypercube*-aufgebaut werden (siehe Abbildung), der bei N Super-Peers eine Pfadlänge von maximal log_2N besitzt. Dadurch werden effiziente und nicht redundante Query-Broadcasts ermöglicht. Anfragen fluten dabei nicht das Netz, sondern werden von einem Peer an seinen Super-Peer weitergegeben und über ein Super-Peer-Backbone gemäß des Hypercube an die anderen Super-Peers und deren angeschlossene Peers weitergeleitet. Ob die Weiterleitung an alle Super-Peers, oder nur eine Auswahl dieser erfolgt, wird durch Routing Indizes bestimmt.

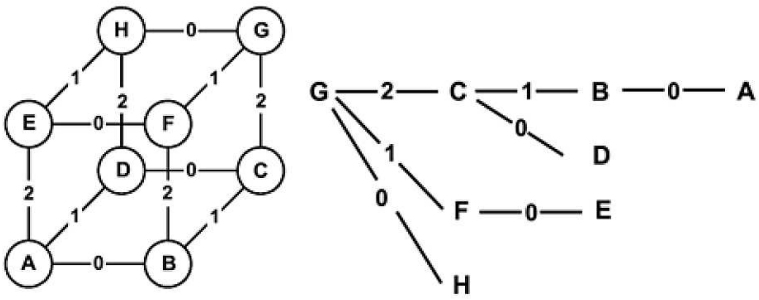

Abbildung 2: HyperCup und Routing Graph [16]

5.1.2 Routing Indizes

Bei Edutella wird zwischen zwei Formen von Routing Indizes unterschieden. Dies sind einmal Super-Peer/Peer Indizes (SP/P Indizes) und Super-Peer/Super-Peer Indizes (SP/SP Indizes). Die SP/P Indizes verwalten Informationen über die an den einzelnen Peers abgespeicherten Metadaten und werden bei der Registrierung eines Peers bei einem Super-Peer angelegt(Beispiel siehe Abbildung). Sollten die Schemas oder Metadaten eines zu registrierenden Peers schon vorhanden sein, wird von diesen nur eine Referenz auf ihn erzeugt.

Metadaten können verschiedene Granularitäten haben, wobei hier in Schemas, Schema Properties, Property Value Ranges und Property Values unterschieden wird. Damit die Queries nur an die Super-Peers weitergeben werden, welche überhaupt die passende Schemas unterstützen, wurden die SP/SP Indizes eingeführt. Sie enthalten eine Zusammenfassung der SP/P Indizes aller Super-Peers, anhand derer dann entschieden wird ob ein Super-Peer für eine Query-Weiterleitung in Frage kommt. Wenn ein Peer eine Anfrage stellt wird diese zuerst an seinen Super-Peer weitergeleitet. Dieser entscheidet dann anhand seiner SP/P Indizes an welche seiner eigenen Peers die Anfrage geht. Dann wird aufgrund der Information in den SP/SP Indizes die Anfrage an die Super-Peers weitergeleitet, deren Peers passende Schemen unterstützen. Wenn die Anfrage dann bei einem Super-Peer vorliegt wird der SP/P Indize genutzt um an die untergeordneten Peers weiterzuleiten. Weil die Möglichkeit besteht, dass ein Peer das Netzwerk ohne Rückmeldung verlässt, ist die Registrierung nur temporär. Sie muss daher in bestimmten Zeitabständen von den Peers erneuert werden.

5.1.3 Query-Verarbeitung

Im Gegensatz zu Systemen wie Gnutella, wo das Netzwerk mit Anfragen geflutet wird und die Query-Verarbeitung vollständig beim Client stattfindet, schlägt man für das Edutella Pro-

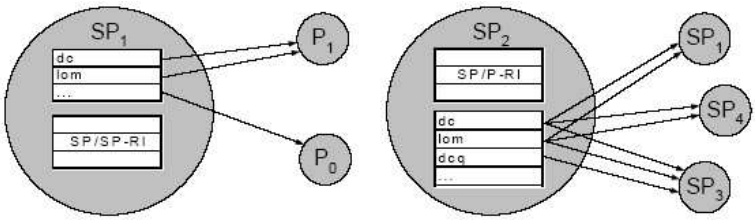

Abbildung 3: SP/P und SP/SP Routing Indizes auf Schemaebene [16]

jekt eine Verteilung der Query-Verarbeitung auf die (Super-)Peers vor. Dabei verhindert man zum einen das Fluten des Netzwerkes mit Nachrichten, da nur die notwendigen Peers angefragt werden. Zum anderen können nutzerdefinierte Filter und komplexe Operatoren schon auf der Ebene der (Super-)Peers ausgeführt werden, und verringern so die Datenmenge, die an Peers übertragen werden muss. Denn wenn diese das Query-Processing alleine ausführen müssten, würden sie auch alle Anfrageergebnisse selber verarbeiten müssen, was zum Beispiel für mobile Clients mit beschränkter Rechenleistung zu Problemen führen kann. Super-Peers stellen somit Fähigkeiten für die Query-Verarbeitung, Query-Optimierung und Management der Indexstrukturen zur Verfügung. Zusätzlich wird angenommen, dass Peers, die über ein bestimmtes Mindestmaß an Rechenleistung verfügen, Query-Verarbeitung durchführen. Das Hauptaugenmerk der Query-Verarbeitung liegt beim verteilten Durchführen von Code-Carrying Query Evaluation Plans (QEP). Damit ist das datennahe Ausführen von Teilen des QEP gemeint, die nutzerdefinierten Code, wie zum Beispiel Joins oder Selektionen, enthalten. Dazu stellt der Client eine Anfrage, die nutzerdefinierten Code enthält, an einen (Super-)Peer. Wenn die Anfrage an einen Peer gestellt wurde, wird sie von diesem erst an seinen Super-Peer weitergeleitet. Dessen Query-Optimierer stellt dann fest wo am besten welche Teile des QEP ausgeführt werden und mit welchen Operatoren die Ergebnisse kombiniert werden müssen. Im Anschluss werden Teile des QEP an die entsprechenden Super-Peers versandt und dort wiederum optimiert und an die (Super-)Peers weitergeleitet, die nach Query-Plan dafür in Frage kommen. Der Vorteil dabei ist, dass die Rechenleistung die für die Auswertung des QEP nötig ist auf die (Super-)Peers verteilt wird und der Traffic sich auf das Versenden von Anfrageergebnissen beschränkt. Müßten erst alle Ergnisse an einem Knoten zusammengetragen werden, bevor zum Beispiel eine Join Operation oder eine Selektion stattfinden kann, wäre der Traffic bedeutend höher. Weiterhin können auch Thin-Clients wie zum Beispiel PDA's oder WAP-Handies das P2P-Netzwerk anfragen, weil sie nur Anfrageergebnisse anzeigen müssen und dafür wenig Ressourcen benötigen. Der Verteilungsprozess ist dynamisch da er auf den Routing Indizes der Super-Peers basiert. Deshalb ist auch keine statische Query-Optimierung möglich und wird dynamisch anhand der Daten durchgeführt, die den Super-Peers jeweils momentan bekannt sind.

5.1.4 Generierung und Optimierung von Query-Plänen

Die QEP-Erzeugung verläuft in 5 Hauptschritten. Zu Beginn wird die SQL-Query geparst und in eine interne Darstellung umgewandelt. Es erfolgt außerdem eine Vorbereitung der nachfolgenden Schritte, indem zum Beispiel die benutzerdefinierten Operatoren und Properties identifiziert werden. Es schließt sich das Resource binding an, in dem die lokalen Indizes benutzt werden, um die Speicherorte der benötigten Ressourcen auszumachen. Dafür sieht man in Edutella eine Einteilung in **resource directions** (RD), **physical resources** (PR) und **logical resources** (LR) vor. Eine LR wird durch vom Nutzer spezifizierte properties oder property-values eingeschränkt und unter Benutzung des lokalen SP/P Index an PR's gebunden. Wenn kein Lokaler Eintrag vorliegt, wird unter Zuhilfenahme des SP/SP Index versucht, die Bindung an eine RD zu vollziehen, die auf einen anderen Super-Peer zeigt. Wenn die Bindungen erfolgt sind wird in einem dritten Schritt ein lokaler Query-Plan erzeugt. In diesem werden die Unterpläne ermittelt, welche bei den benachbarten Super-Peers ausgeführt werden müssen. Da nur ein Teil des Super-Peer-Backbones eingesehen werden kann, ist keine statische Query-Optimierung möglich. Jeder Super-Peer optimiert seinen Teil des Planes unter Zuhilfenahme von statistischen Daten über die Input-Daten, die Netzwerk-Topology und die

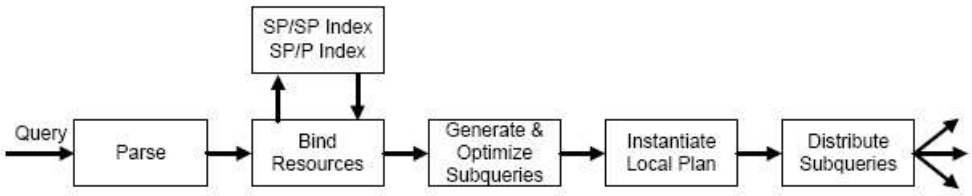

Abbildung 4: Query-Plan Generierung an einem Super-Peer [16]

verwalteten Peers. Dabei sollen zum Beispiel die Anzahl gebunder Ressourcen, Antwortzeiten der Peers und deren Transfer-Raten eine Rolle spielen. Nachdem ein lokaler Plan generiert wurde, wird er am Super-Peer instanziiert. Sollte es Super-Peers geben, die Ergebnisse des lokalen Plans als Input-benötigen, wird mit diesen eine Verbindung aufgebaut. Wenn dann die Verbindungen mit den Super-Peers aufgebaut wurden, denen die Unterpläne zugeteilt werden sollen, kann der lokale Plan ausgeführt werden. In einem letzen Schritt werden die Unterpläne an die entsprechenden Super-Peers verteilt und die eigentliche Query-Verarbeitung kann beginnen. Zum Abschluss sammelt der Super-Peer die Results der Unterpläne, führt eventuell noch Operationen auf diesen aus und sendet dann die Ergebnisse an den anfragenden Peer/Client.

5.1.5 Zusammenfassung

Brunkhorst, Dhraief, Kemper, Nejdl und Wiesner stellen eine auf RDF und JXTA aufbauende Architektur für ein P2P-System vor. Die Super-Peer Topology und indizebasiertes Routing sollen den Traffic-verringern und die Skalierbarkeit des P2P-Systems erhöhen. Weiterhin sollen die besprochenen Konzepte für verteilte und datennahe Queryausführung in die fortschreitende Entwicklung von Edutella einfließen. Weitere Möglichkeiten für Optimierungen sehen sie in der erweiterten Einbindung von statistischen Daten. Darüber hinaus sollen alternative Strategien für die Erstellung des Query-Plans und des dynamischen Plazierens von Operatoren untersucht werden.

5.2 Ein Framework für komplexe Queries in P2P-Systemen

In ihrem Research-Paper diskutieren Triantafillou und Pitoura einen auf Distributed Hash Tables (DHT) basierenden Ansatz zu schemabasierten P2P-Systemen [21]. Das Ziel der Forschungsarbeit ist ein umfassendes Query-Processing-System für P2P-Netzwerke, dass komplexe Queries effizient verarbeitet. Dabei stellen sie sich ein durch DHT strukturiertes Netzwerk vor, in dem Anfragen nicht nur über ein Attribut und eine Relation erfolgen können, sondern auch über mehrere Attribute und Relationen. Der Einsatz von Aggregatfunktionen, sowie die Gruppierung und Sortierung von Anfrageergebnissen ist ebenfalls vorgesehen. Im Anschluß möchte ich die Architektur dieses Frameworks erläutern und Lösungen für das Query-Processing aufzeigen, welche im Rahmen dieser Arbeit gegeben werden.

5.2.1 Framework-Architektur

Das P2P-Netzwerk baut auf der der Chord-DHT Lösung auf, das heißt die Anordnung der Peers erfolgt in einem modulo 2^m Ring - dem Chord-Ring. Von jedem Peer wird erwartet, dass er Daten veröffentlicht, abspeichert und weitergibt. Er wird innerhalb des Chord-Rings

über einen m-bit Identifier identifiziert. Die Daten werden in Relationen abgelegt, pro Daten-Objekt ein Tupel mit Attributen. Jedes Tupel verfügt außerdem über einen Primary-key t. Zur Vereinfachung wird angenommen, dass es nur eine Relation gibt $R(DA_1, DA_2, .., DA_k)$. Die DA_i sind die Wertebereiche der Attribute, A_i die Attributnamen für $i \in \{1, 2, ..k\}$. Die Wertebereiche sind im Moment noch auf Integer-Werte begrenzt, sollen aber in der weiteren Entwicklung auf andere Datentypen wie Strings oder Boolean ausgeweitet werden. Der Identifier des Peers wird über eine Hash-Funktion wie zum Beispiel SHA-1 [5] aus der IP-Adresse berechnet. Zusätzlich wird mit einer ähnlichen Hash-Funktion ein Wert für $t \in \{0, 1, ..2^m - 1\}$ berechnet. Die Anordnung der Tupel im Chord-Ring erfolgt über die Funktion $succ()$. Dazu wird jedes Tupel $R(a_1, a_2, .., a_k)$ mit Schlüssel t dem Peer zugeordnet, dessen Identifier gleich dem von t oder der Nächstgrößere innerhalb des Chord-Ringes ist. Dieser Peer wird dann als Successor-Peer bezeichnet und durch $succ(t)$ angegeben. Die Routing Informationen jedes Peers umfassen seinen Vorgänger im Chord-Ring und, logarithmisch ansteigend, seine Nachfolger. Die Komponente, welche die Nachfolger abspeichert, wird dabei als *finger-table* bezeichnet. Um Range-Queries zu ermöglichen werden zusätzlich die Attribute der Tupel mit einer reihenfolgeerhaltenden Hash-Funktion (order-preserving) gehasht. Die k-Hash-Funktion ergeben sich dabei wie folgt. Es wird die Differenz aus dem jeweils höchsten und niedrigsten Werten eines Wertebereiches $DA_i = \{low_i, high_i\}$ gebildet. Dann wird der Identifier-Raum in $\frac{2^m}{s_i}$ Bereiche aufgeteilt wobei sich s_i ergibt durch:

$$s_i = \frac{2^m}{high_i - low_i + 1}$$

Der Hashwert eines Attributes $a_i \in DA_i$ wird ermittelt über

$$h_i : DA_i \rightarrow \{0, 1, ..., 2^m - 1\}, \quad h_i(a_i) = ((a_i - low_i) * s_i)$$

Für jedes Attribut eines Tupels wird ein entsprechendes Replikat in aufsteigender Reihenfolge im Chord-Ring abgespeichert. Bei k Attributen existieren dann maximal $(k+1)$ Replikate des Tupels innerhalb des P2P-Netzwerkes. Triantafillou und Pitoura geben selber zu bedenken, dass dies ineffizient sein kann, aber meinen, dies sei durch gesteigerte Verfügbarkeit, die Verringerung von Routing-Hops und Ermöglichung von Range- und Join-Queries gerechtfertigt. Eine mögliche Verbesserung, um Speicherplatz zu sparen, sehen sie darin, für jedes Replikat jeweils nur das Attribut und einen Verweis zu hinterlegen. Dieser zeigt dann auf den Peer der über $succ(t)$ ermittelt wurde und das vollständige Tupel speichert. Das würde andererseits die Anzahl der Einträge in den Routingtabellen stark erhöhen.

5.2.2 Rudimentary Queries

Die Queries werden von Triantafillou und Pitoura in rudimentary und non-rudimentary Queries eingeteilt. Die nicht rudimentären Anfragen bauen dabei auf den rudimentären Anfragen auf. Sie werden, was die verwendeteten Mechanismen angeht, im Query-Processing aus den einfachen Anfragen zusammengesetzt. Der erste Anfragentyp $[SR, SA, =]$ bezieht sich auf eine Relation (SR=single relation). Es wird nur ein Attribut in Betracht gezogen (SA=single attribute) und dieses durch eine Bedingung über den Gleichheitsoperator eingeschränkt. Ein Beispiel in SQL:

```
SELECT *
FROM R
WHERE R.a=10
```

Der Input für diese Anfrage ist ein Attribut der Relation, $a_i \in DA_i$, und als Ergebnis wird eine Liste von Tupeln zurückgeliefert, die dieses Attribut mit einem bestimmten Wert enthalten. Der anfragende Peer berechnet dazu, über den Chord-Lookup Algorithmus mit Hilfe der Hashfunktion h_i, die Peers, die passende Tupel enthalten. Dann sendet er seine Anfrage an diese Peers, welche ihre lokal gespeicherten Tupel durchsuchen und die Ergebnisse zurücksenden. Die Kosten dafür betragen $O(logN)$ Routing-Operationen, gemäß der Komplexität des Lookup-Algorithmus.
Der zweite Anfragetyp $[SR, SA, <>]$ entspricht der ersten Anfrage, allerdings ist der Operator für die Bedingung ein Range-Operator, das heißt er gibt einen Wertebereich für das Attribut an. SQL-Beispiel:

```
SELECT *
FROM R
WHERE R.a between '10' and '100'
```

Durch die schon besprochenen reihenfolgeerhaltenden Hash-Funktionen h_i werden nun die Peers berechnet, welche den geforderten Wertebereich $(low, high)$ eingrenzen. Dann wird die Query an den Peer geroutet, der die Tupel ab der unteren Grenze enthält $succ(h_i(low))$. Dieser sendet die geforderten Tupel zurück und leitet die Anfrage an seinen Nachfolger im Chord-Ring weiter. Aufgrund der Reihenfolgeerhaltung enhält dieser nun ebenfalls Tupel mit Attributen aus dem entsprechenden Wertebereich. Die Anfrage wird dann solange an den Nachfolger weitergegeben, bis die obere Grenze des Wertebereichs erreicht ist und der Algorithmus abbricht. Die Komplexität der Routing-Operationen entspricht der Anzahl der Peers auf den der geforderte Wertebereich des Attributes verteilt ist. Im besten Fall ist das ein Peer mit $O(logN)$. Sollte der Wertebereich über alle Peers verteilt sein, entspricht das dem schlechtesten Fall mit $O(N)$.
Um die Performanz zu verbessern wurden das Konzept der Range Guards (RG) eingeführt. Einer bestimmten Anzahl von leistungsfähigen Peers werden jeweils Replicate der Tupel zugeteilt, wo ein bestimmtes Attribute in einen bestimmten Wertebereich fällt. Jeder RG steht somit für einen Wertebereich und enthält Routing-Informationen für die benachbarten RGs. Auch besitzt jeder Peer eine Verknüpfung für seinen entsprechenden RG. Um Lastverteilung zwischen den RGs sicher zu stellen, wird der Attributwertebereich DA eines Attributes A in l Teile partitioniert. Die Größe s einer Partition wird über

$$s = \frac{Betrag\ von\ DA}{l}$$

bestimmt. Sollte nun eine Anfrage für einen bestimmten Wertebereich $(low, high)$ von einem Peer gestellt werden, wird über $succ(h_i(low))$ der Peer ausfindig gemacht, welcher die Tupel ab der unteren Grenze low enthält. Dieser Peer leitet die Anfrage an den entsprechenden RG weiter, welcher seine Daten durchsucht und die Ergebnistupel an den Anfrager sendet. Dann wird überprüft, ob der RG die obere Grenze des Wertebereiches schon abdeckt. Wenn dies nicht der Fall ist, wird die Query an den nachfolgenden RG weitergeben. Der Algorithmus bricht ab, wenn der Wertebereich eines RG die obere Grenze beinhaltet. Er benötigt $O(logN)$ Routing-Operationen um, $succ(h_i(low))$ zu finden und $O(l)$ um die Anfrage von einem RG zum nächsten zu geben. Da es signifikant weniger RG als Peers gibt, $l << N$, ist diese Methode für Range Queries viel effizienter als die Vorhergehende. Triantafillou und Pitoura schlagen für l den Wert $logN$ vor, was dann einer Gesamtkomplexität von $O(logN)$ entspräche. $O(logN)$ für den Lookup von $succ(h_i(low))$, plus $O(logN)$ für die Routing-Operationen der

RGs. Nachteile dieses Ansatzes sind die hohen Leistungsanforderungen an die RGs. Diese müssen sowohl über große Speicherkapazitäten als auch hohe Rechenleistung und Bandbreite verfügen. Für alle Peers erhöht sich zudem die Anzahl von Routing-Informationen die abgespeichert werden müssen. Die Anzahl der abgespeicherten Tupel kann, wie schon bei der reihenfolgeerhaltenden Abspeicherung der Attribute, über die Speicherung von Verweisen statt kompletter Tupel verringert werden. Dies erhöht allerdings nochmals die Menge notwendigen Routing-Informationen.
Die letzte rudimentäre Anfrage ist $[MR, MA, join]$. Damit werden Queries klassifiziert, die eine Join-Operation über mehrere Relationen und Attribute ausführen. Zum Beispiel über zwei Relationen R und S:

```
SELECT *
FROM R, S
WHERE R.a=S.a
```

Aufgrund der Reihenfolgeerhaltenden Abspeicherung befinden sich Attribute mit gleichen Werten auch an den gleichen Peers. Das heißt:

$$R.a = S.a \Rightarrow h(R.a) = h(S.a) \Rightarrow succ(h(R.a)) = succ(h(S.b))$$

Damit müssen für einen Join keine Tupel zwischen Peers übertragen werden, sondern er kann lokal erfolgen und das Ergebnis wird an den Anfrager gesendet. Dies entspricht im Grunde einem Hash-Join von RDBMS. Da alle Peers einbezogen werden müssen, ensteht hoher Rechenaufwand und die Routing-Kosten betragen $O(n)$. Eine Verbesserung kann abermals durch den Einsatz von RGs erzielt werden. Dazu müssen für alle Relationen und deren Attribute RGs bestimmt werden. Der Wertebereich wird wie schon beschrieben auf l RGs aufgeteilt. Der Join bezieht dann nur diese l Peers ein, was verhindert das sich gesamte P2P-Netzwerk mit der Join-Query befassen muss. Die Komplexität beträgt für $l = logN$, wie schon oben, $O(logN)$. Diese Vorgehensweise entspräche dann einem partitionierten Hash-Join von RDBMS.

5.2.3 Non-rudimentary Queries

Die nicht rudimentären Anfragen werden von Triantafillou und Pitoura nur kurz besprochen und im Grundsatz über eine Mehrfachanwendung der Techniken für rudimentäre Anfragen bearbeitet. Der Anfragen-Typ $[SR, MA, =]$ (MA=Multible Attributes) funktioniert wie $[SR, SA, =]$ nur das dabei der Lookup-Algorithmus mehrfach, eben für jedes Attribut durchläuft. Dann werden die Ergebnisse an den Anfrager gesendet. Beispielanfrage:

```
SELECT *
FROM R
WHERE R.a=10 and R.b=100 and R.c=1000
```

Ähnlich wird $[SR, MA, <>]$ bearbeitet, wobei hier $[SR, SA, <>]$ einfach mehrmals durchlaufen wird. Eine Anfrage dazu in SQL:

```
SELECT *
FROM R
WHERE (R.a between '10' and '100') and (R.b between '20' and '200')
```

Auf genau die gleiche Weise wird $[MR, MA, =]$ verarbeitet. Es wird einfach $[SR, MA, =]$ mehrfach angewendet. Dazu ein mögliche SQl-Anfrage:

```
SELECT *
FROM R, S
WHERE R.a=10 and R.b=20 and S.c=30 R.d=S.d
```

Bei $[MR, MA, <>]$ wird $[SR, MA, <>]$ mehrfach ausgeführt. Eine Beispielanfrage in SQL:

```
SELECT *
FROM R, S
WHERE (R.a between '10' and '100') and (R.b between '20' and '200') and (S.c
between '30' and '300') and (S.d between '40' and '400')
```

Die letzte nicht rudimentäre Anfrage $[MR, MA, =, sf]$, wobei sf für special functions steht, wird unter Zuhilfenahme aller vorangegangenen Query-Methoden bearbeitet. Funktionen wie min, max, average usw. werden auf die Ergebnistupel angewandt. Der Einsatz von Range Guards verhindert bei diesem wie auch den vorangegangenen Anfragen einen zu großen Routing-Aufwand.

5.2.4 Zusammenfassung

Das von Triantafillou und Pitoura vorgeschlagenen Framework basiert auf Chord und soll als Basis für komplexe Anfragen in schemabasierten P2P-Systemen fungieren. Das Projekt befindet sich noch am Anfang und die komplexeren Queries, hier als non-rudimentary Queries bezeichnet, werden nur kurz abgehandelt. Auf Probleme wie die Kosten der Reorganisation des Chord-Ringes bei Ab-und Zugängen von Peers wird noch nicht eingegangen. Die Behandlung von leistungsschwachen Peers wie zum Beispiel PDA's wird auch nicht angesprochen. Daher bezeichnen sie ihre Arbeit auch als ersten Schritt und sehen weitere Herausforderungen zum Beispiel in dem Umgang mit sich dynamisch verändernden Schemas im P2P-Netz und der Verbesserung der Identifizierung von Range Guards.

5.3 PIER

Nach den theoretischen Betrachtungen zu DHTs, Netzarchitekturen und Query-Processing möchte ich mit *PIER* (Peer-to-Peer Information Exchange and Retrieval) [22] eine Forschungsarbeit von *Huebsch, Hellerstein, Lanham, Loo, Shenker und Stoica* vorstellen. Diese beinhaltet neben der Beschreibung des Systems auch eine Performanz-Studie von PIER, welche zeigt, das verteiltes Query-Processing in einem P2P-System auch bei tausenden von Peers effizient sein kann. Neben einer kurzen Systembeschreibung möchte ich das Simulationssetup vorstellen, sowie eine Zusammenfassung der Ergebnisse geben.

5.3.1 PIER Architecture

PIER ist ein Drei-Schichten-System(siehe Abbildung). Es besteht aus der Anwendungschicht welche über Schnittstellen mit der PIER-Schicht kommuniziert. Dort befindet sich der Query Processor (QP), welcher seine Aufgaben mit Hilfe der darunterliegenden DHT-Schicht ausführt. Hier wurde dafür das Content Adressable Network (CAN) implementiert. Dessen Routing Layer übernimmt die Aufgabe, den Schlüssel eines Datenobjektes in die IP-Adresse des Rechners umzuwandeln, der momentan für die Daten verantwortlich ist. Im Storage Manager werden temporäre Daten abgelegt, die im Zusammenhang mit der DHT stehen. Dies sind zum Beispiel Routing-Informationen. Im Moment benutzt der Storage Manager dafür

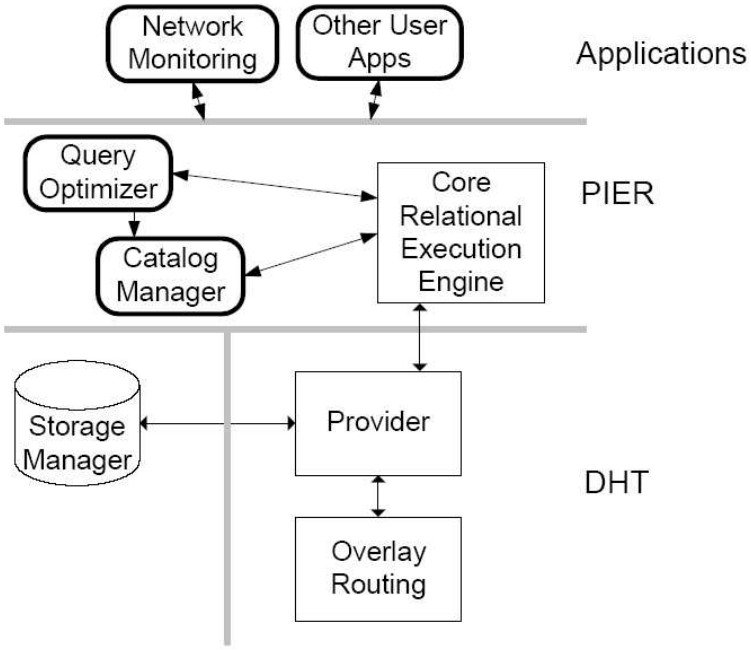

Abbildung 5: PIER Architecture [22]

noch ausschließlich Hauptspeicher, soll aber in Zukunft auch ein Disk/Filesystem verwenden können. Der Provider verknüpft die DHT mit dem Storage Manager und stellt eine Schnittstelle für Anwendungen dar, beziehungsweise die Verbindung zur PIER-Schicht her. Alle von der DHT-Schicht verwalteten Objekte werden einem Namespace zugeordnet, besitzen eine *resourceID* und eine *instanceID*. Aus dem Namespace und der *resourceID* wird der DHT-Schlüssel berechnet. Weiterhin gibt der Namespace die Applikation/Gruppe an, zu der das Datenobjekt gehört. Im Bezug auf das Query-Processing verhält er sich wie eine Relation. Der Query-Processor von PIER unterstützt Selektionen, Projektionen, Distributed Joins, Gruppierungen, sowie Aggregationen und ermöglicht deren verteilte Ausführung.

5.3.2 Join Berechnung

Der bei PIER verwendete Join-Algorithums ist eine DHT-basierte Version des pipelining *symmetric hash-join* [1]. Angenommen es existieren zwei Namespaces N_R und N_S die den Relationen R und S entsprechen. Die Grundidee ist, einen neuen Namespace N_Q zu schaffen, der die potentiellen Join-Tupel von R und S aufnimmt. Dafür führen alle Peers, die Teile von N_R und N_S besitzen, einen lokalen Scan ihrer Tabellen durch und identifizieren mit Hilfe der Selektionsprädikate die Tupel, welche für den Join in Frage kommen. Die Ergebnisse werden in N_Q eingefügt, wobei die verketteten Werte der Join-Attribute die ResourceID ergeben. Mit Hilfe der DHT können dann die Peers gefunden werden, die passende Join-Partner besitzen. Wenn bei einem Peer, der zum Namespace von N_Q gehört, ein Tupel ankommt, wird es mit den lokalen Tupeln verglichen und wenn ein passender Eintrag vorliegt, zu einem Antwort-Tupel verbunden. Dieses kann dann zum Anfrager oder zu der nächsten Stufe einer Query gesendet werden (wiederum ein anderer Namespace).

5.3.3 Simulationsetup

Für ihre Evaluierung haben die Entwickler von PIER einen Simulator programmiert, der bis zu 10000 Peers simuliert. Weiterhin wurden auf einem verteiltem Cluster von 64 Rechnern Messungen vorgenommen. Als Maße werden dabei nicht nur die Datengrößen betrachtet, sondern auch die Anzahl der Peers und die zugrunde liegenden Netzwerkcharakteristiken. Die Testanfrage war dabei folgende:

```
SELECT R.pkey, S.pkey, R.pad
FROM R, S
WHERE R.num1 = S.pkey
AND R.num2 > constant1
AND S.num2 > constant2
AND f(R.num3, S.num3) > constant3
```

R und S wurden künstlich erzeugt. R hat ca. 10 mal mehr Tupel als S. Die Konstanten wurden so gewählt, das die Selektivität 50% beträgt. Für die Funktion f, die nach dem Equi-Join ausgewertet werden muss, wurde die Verteilung der Tupel so gewählt, das 90% der Tupel aus R ein passendes Tupel in S haben. R.pad wurde eingeführt, um sicherzustellen, das jede Antwort eine Größe von 1KB hat. Für die Join-Operation wird das oben besprochene Verfahren benutzt. Eine weitere Annahme der Simulation ist eine Netzwerkverzögerung von 100ms zwischen zwei beliebigen Knoten und eine 10Mbps Anbindung. Der 64 Rechner-Cluster ist über ein 1-Gbps Netzwerk verbunden. Die Messungen wurden jeweils nach dem Einladen von R und S, sowie dem Stabilisieren des CAN vorgenommen. Wenn m von n Peers an einer verteilten Query-Berechnung teilnehmen beträgt die Datenlast t, die durch die Selektivität der Prädikate entsteht und an die rechnenden Knoten verteilt werden muss, jeweils $\frac{t}{m}$ mit $1 \leq m \leq n$. Bei einer Datenbank von 2GB und 50% Selektivität auf R und S wären dies t=1GB.

5.3.4 Skalierbarkeit

Unter der Annahme, dass jeder Knoten 1MB Daten verwaltet und die Query berechnet, wird jeweils die Zeit für das 30. Antworttupel gemessen. Die Performanz verringert sich dabei um einen Faktor von 4. Dies ist besser als das, was laut Komplexität von CAN herauskommen müßte. Dort würde die lookup-length um Faktor $n^{\frac{1}{4}}$ ansteigen, wobei n die Anzahl der Knoten/Peers ist. Bei einer Erhöhung der Knotenanzahl von 2 auf 10000 wäre das der Faktor 10. Dieser Unterschied wird damit erklärt, dass die Zeit für das Zurücksenden der Tupel an den anfragenden Knoten bei steigender Netzgröße nicht mit ansteigt. Die Autoren sehen die Skalierbarkeit ihres Ansatzes als bewiesen an, sofern die Anzahl der rechnenden Knoten nicht zu klein ist. Denn sollte dies der Fall sein, müßte jeder rechnende Knoten über eine hohe Bandbreite verfügen, weil er von vielen anderen Knoten Daten empfangen muss. Und dies würde zu Flaschenhälsen führen.

5.3.5 Zusammenfassung

Mit PIER wird die Implementierung eines schemabasierten P2P-Systems vorgestellt, dass auf DHT's aufbaut. Es wird anhand von Evaluierungen gezeigt, dass verteilte Query-Verarbeitung und Skalierbarkeit vereinbar sind und dient somit als Beispiel für zukunftsträchtige Entwicklungen im Bereich schemabasierter P2P-Systeme.

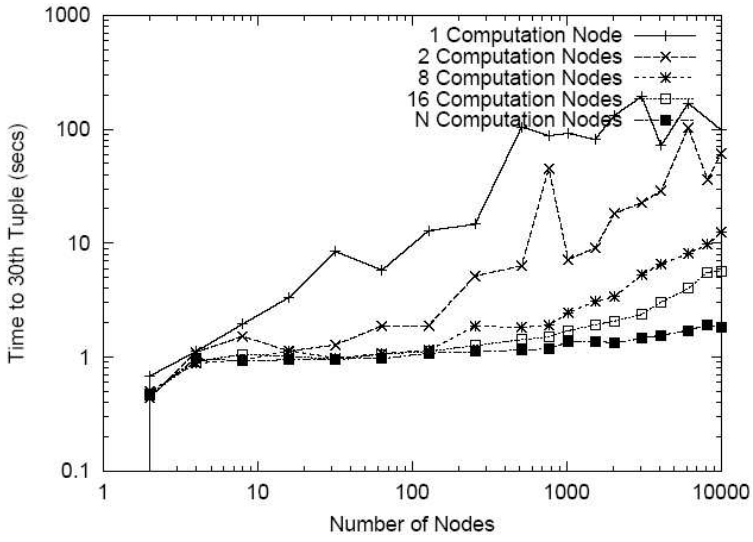

Abbildung 6: Durchschnittliche Antwortzeit für das 30. Tupel [22]

5.4 Taxonomy- und ontologiebasierte Systeme

Einen weiteren Ansatz zu schemabasierten verteilten Systemen stellen taxonomy- und ontologiebasierte Systeme dar. Dabei wird das Augenmerk auf eine gute Query-Verarbeitung und Modellierung von Sichten auf Daten gelegt. Die bekannten Problematiken wie Routing, dynamischer Charakter des Netzes oder zu hoher Netztraffic spielen in bisherigen Arbeiten keine große Rolle.

In ontologiebasierten Systemen beschreiben Peers ihre Daten mit Ontologien. Die wirklich vorhandenen Daten stellen dabei Instanzen dieser Ontologien dar. Daten werden ausgetauscht, indem Queries formuliert werden, die ein Vokabular benutzen, das ebenfalls in den Ontologien definiert ist. Der Peer welcher eine Query erhält, muss anhand seiner eigenen Daten und Informationen die Antwort ermitteln. Probleme die dabei auftreten können, sind zum Beispiel, dass nicht vorausgesetzt ist, dass der angefragte Peer die gleiche Ontologie benutzt.

Bei taxonomiebasierten Systemen verhält es sich ähnlich. Die Taxonomien beschreiben hierbei die Sicht der Peers auf ihre Daten. Die Arbeiten zu taxonomiebasierten Systemen von Tzitzikas, Meghini und Spyratos [25] und zur Query-Verarbeitung in Ontologiebasierten Systemen von Stuckenschmid, van Harmelen und Giunchiglia [15] seien dem interessierten Leser hierzu empfohlen.

6 Zusammenfassung

Mit der Entwicklung von schemabasierten P2P-Systemen wird ein neues Kapitel in der Geschichte der P2P-Systeme aufgeschlagen. Technolgien wie DHT und RDF ermöglichen Architekturen, die mehr leisten als nur File-Sharing mit Keyword-Suche. Komplexe Suchanfragen werden möglich und verbessertes Routing verringert die Wartezeit auf eine Antwort, beziehungsweise führt dazu, dass mehr Peers erreicht werden.

Dennoch stehen die Entwicklungen erst am Anfang und viele Probleme müssen noch gelöst werden. Diese werden vor allem durch den dynamischen Charakter der P2P-Netze verursacht,

wie zum Beispiel das Peers ständig das Netz betreten und verlassen können. Die Integration heterogener Schemas sowie eine Query-Evaluierung bei nicht vorhandenem globalem Wissen stellen ebenfalls große Herausforderungen dar. Forschungsarbeiten wie [16, 20] versuchen diese Herausforderungen zu meistern und Implementationen wie [22] zeigen, dass der Einsatz von Technologien wie DHT's Performanzvorteile gegenüber herkömmlichen Ansätzen hat. Ob sich schemabasierte Systeme durchsetzten und wie die Entwicklung weitergeht, wird die Zukunft zeigen.

Literatur

[1] A. N. Wilschut und P. M. G. Apers. Dataflow Query Execution in a Parallel Main-Memory Environment. In *Proc. First International Workshop on Peer-to-Peer Systems (TPTPS '01)*, Cambridge, MA, Mar. 2002.

[2] Antony Rowstron und Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pages 329–350, Nov 2001.

[3] B. Y. Zhao, J. D. Kubiatowicz, und A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01- 1141, UC Berkeley, April 2001. citeseer.nj.nec.com/zhao01tapestry.html.

[4] C. Greg Plaxton, Rajmohan Rajaraman, und Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In ACM Symposium on Parallel Algorithms and Architectures, pages 311–320, 1997.

[5] FIPS180-1. Secure hash Standard. U.S. Department of Commerce/NIST, National Technical Information Service, Springield, VA (1995)

[6] http://www.w3.org/RDF/

[7] http://www.w3.org/TR/1998/WD-rdf-schema/

[8] http://www.w3.org/TR/REC-rdf-syntax/

[9] http://www.ontoknowledge.org/oil/

[10] http://www.w3.org/2000/01/rdf-schema

[11] http://www.ontoknowledge.org/oil/syntax/Standard-OIL/
http://www.ontoknowledge.org/oil/rdf-schema/2000/11/10-oil-standard

[12] http://www.ontoknowledge.org/oil/syntax/Instance-OIL/
http://www.ontoknowledge.org/oil/rdf-schema/2000/11/10-oil-standard

[13] http://www.informatik.hu-berlin.de/⌣bien/ontologie.html

[14] http://edutella.jxta.org/,2002.

[15] H.Stuckenschmidt, F.van Harmelen, F.Giunchiglia: Query Processing in Ontology-Based Peer-to-peer Systems. Techn. Report, AI Department Vrije Univ. Amsterdam, Nov. 2002.

[16] I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, C. Wiesner: Distributed Queries and Query Optimization in Schema-Based P2P-Systems. Intern. Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2003.

[17] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, und Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet aplications. In Proceedings of ACM SIGCOMM'01, September 2001.

[18] L.Gong.Project JXTA:A technology overview. Technical report,SUN Microsystems, 2001. http://www.jxta.org/project/www/docs/TechOverview.pdf.

[19] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP-Hypercubes, Ontologies and Efficient Search on P2P Networks. In Intl. Workshop on Agents and P2P Computing, 2002.

[20] P. Triantafillou, T. Pitoura: Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. VLDB '03 Workshop on Databases, Information Systems, and Peer-to-Peer Computing, 2003.

[21] P.Triantafillou, T.Pitoura: Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. VLDB '03 Workshop on Databases, Information Systems, and Peer-to-Peer Computing, 2003 http://www.ceid.upatras.gr/faculty/peter/papers/dbisp2p.pdf.

[22] R. Huebsch et al.: Querying the Internet with PIER. Proc. VLDB 2003

[23] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, und Scott Schenker. A scalable content-addressable network. In Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pages 161–172. ACM Press, 2001.

[24] Ulf Schneider, Neue Ansätze der Künstlichen Intelligenz:Ontology Interference Layer, http://www.nef.wh.uni-dortmund.de/ ulf/KI

[25] Y. Tzitzikas, C. Meghini: Query Evaluation in Peer-to-Peer Networks of Taxonomy-based Sources. Proc. 10th Int. Conf. on Cooperative Information Systems (CoopIS), 2003.