Informatik Forsch. Entw. (2005) 20: 152–166
DOI 10.1007/s00450-005-0205-9

REGULÄRE BEITRÄGE

Matthias Bender • Sebastian Michel • Gerhard Weikum • Christian Zimmer

Das MINERVA-Projekt: Datenbankselektion für Peer-to-Peer-Websuche

Eingegangen am 20. September 2004 / Angenommen am 12. Mai 2005 / Online publiziert am 26. August 2005

Zusammenfassung In diesem Artikel wird MINERVA[1] präsentiert, eine prototypische Implementierung einer verteilten Suchmaschine basierend auf einer Peer-to-Peer (P2P)-Architektur. MINERVA setzt auf die in der P2P-Welt verbreitete Technik verteilter Hash-Tabellen auf und benutzt diese zum Aufbau eines verteilten Verzeichnisses. *Peers* in unserem Ansatz entsprechen völlig autonomen Benutzern mit ihren lokalen Suchmöglichkeiten, die bereit sind, ihr lokales Wissen und ihre lokalen Suchmöglichkeiten im Rahmen einer Kollaboration zur Verfügung zu stellen. Wir formalisieren unsere Systemarchitektur und beschreiben das zentrale Problem einer effizienten Suche nach vielversprechenden Peers für eine konkrete Anfrage innerhalb des Verbundes[2]. Wir greifen dabei auf existierende Methoden zurück and passen diese an unseren Systemkontext an. Wir präsentieren Experimente auf realen Daten, die verschiedene dieser Ansätze vergleichen. Diese Experimente zeigen, dass die Qualität der Ansätze variiert und untermauern damit die Wichtigkeit und den Einfluss einer leistungsstarken Methode zur Auswahl guter Datenbanken. Unsere Experimente deuten an, dass eine geringe Anzahl sorgfältig ausgewählter Datenbanken typischerweise bereits einen Großteil aller relevanten Ergebnisse des Gesamtsystems liefert.

Schlüsselwörter Peer-to-Peer · Datenbankselektion · Websuche

Abstract This paper presents the MINERVA[1] project that protoypes a distributed search engine based on P2P techniques. MINERVA is layered on top of a Chord-style overlay network and uses a powerful crawling, indexing, and search engine on every autonomous peer. We formalize our system model and identify the problem of efficiently selecting promising peers for a query as a pivotal issue. We revisit existing approaches to the database selection problem and adapt them to our system environment. Measurements are performed to compare different selection strategies using real-world data. The experiments show significant performance differences between the strategies and prove the importance of a judicious peer selection strategy. The experiments also present first evidence that a small number of carefully selected peers already provide the vast majority of all relevant results.

Keywords Peer-to-Peer · Query Routing · Web Search

CR Subject Classification H.4 · H.3.3 · H3.4

1 Einführung

Peer-to-Peer (P2P)-Architekturen sind insbesondere im Kontext von Musiktauschbörsen wie Napster oder Kazaa bekannt geworden. Sie erlauben den Umgang mit großen Datenmengen in einer verteilten und selbstorganisierenden Weise. In einem solchen kollaborativen System sind alle Peers völlig autonom und die gesamte Funktionalität wird über alle Peers verteilt. Somit wird die Systemlast über alle Teilnehmer verteilt, und der Ausfall eines beliebigen Peers kann niemals zum Ausfall des Systems führen. Diese Eigenschaften bieten ein enormes Potenzial für skalierbare, effiziente, fehlerresistente und dynamische verteilte Systeme. Zusätzlich kann ein solches System vom „intellekuellen" Input einer großen Teilnehmerzahl profitieren, z.B. in Form von Lesezeichen (Bookmarks) oder Click-Streams der Benutzer. Eines der zentralen Probleme ist jedoch das Auffinden guter Peers für ein konkretes Informationsbedürfnis.

Obwohl die Peer-to-Peer-Technologie vergleichsweise noch jung ist, so zeigen sich deutliche Berührungspunkte mit der „traditionellen" Forschung im Information-Retrieval und Datenbankbereich; Peer-to-Peer-Technologie kann daher von bestehenden Vorarbeiten profitieren. Die Besonderheiten einer verteilten Architektur erfordern jedoch einen

Diese Arbeit wurde teilweise gefördert durch das integierte EU-Projekt DELIS (Dynamically Evolving, Large Scale Information System).

M. Bender · S. Michel · G. Weikum · C. Zimmer
Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken

[1] `http://www.minerva-project.org`

[2] Wir verwenden die Begriffe Kollaboration, Verbund und P2P-System synonym.

neuen Blickwinkel auf bestimmte Aspekte. Das Fehlen einer zentralen Indexierung beispielsweise behindert den Einsatz bewährter Methoden zur Auswahl von Datenbanken, insbesondere auch durch die Schwierigkeiten, globale Maße für solch einen großen und dynamischen Verbund zu berechnen.

Dieser Artikel stellt die Architektur des Forschungsprototyps MINERVA vor und zeigt verschiedene Verfahren, mit deren Hilfe Anfragen an ausgewählte Peers delegiert werden können. Jeder Peer ist autonom und besitzt eine eigene lokale Suchmaschine mit dem dazugehörigen lokalen Index. Die Peers teilen ihr Wissen (oder eine selbstgewählte Teilmenge davon), indem sie Meta-Informationen publizieren. Als Meta-Informationen wählen wir kompakte Statistiken sowie Quality-of-Service-Informationen und formen so ein konzeptionell globales Verzeichnis. Dieses Verzeichnis wird jedoch in einer vollständig dezentralisierten und weitestgehend selbstorganisierenden Weise implementiert. Wir benutzen dazu die Technik verteilter Hash-Tabellen am Beispiel des Chord-Protokolls [38], das wir für unsere Zwecke angepasst und re-implementiert haben. MINERVA benutzt dieses konzeptionell globale Verzeichnis, um solche Peers zu identifizieren, die mit höchster Wahrscheinlichkeit die qualitativ hochwertigsten Ergebnisse zu einer Anfrage liefern. Eine Anfrage wird dabei standardmäßig zunächst mit der lokalen Suchmaschine des Anfragenden ausgewertet, kann danach aber zur Verbesserung der Resultatsgüte an andere Peers weitergeleitet werden. Die jeweiligen Resultate dieser Peers werden vom Anfragenden letztlich zu einem gemeinsamen Resultat zusammengeführt.

Der wissenschaftliche Beitrag unserer Arbeit liegt in der systemorientierten Herangehensweise an Websuche im Kontext von P2P-Systemen. Wir präsentieren Messergebnisse unseres vollständig lauffähigen Systems auf realen Daten und berücksichtigen dabei sowohl die Resultatsgüte als auch die Effizienz und den Overhead unserer Architektur. Zu den neuen Aspekten gehören die Art und Weise, wie wir die zugrunde liegende Chord-artige Basisarchitektur zum effizienten Umgang mit den Meta-Informationen der einzelnen Peers nutzen und darauf die Güte verschiedenster Verfahren zur Auswahl geeigneter Datenbanken evaluieren. Im Gegensatz zu früheren Arbeiten auf dem Gebiet des verteilten Information-Retrieval (z.B. [13, 19, 31]) wurde unsere Architektur von Beginn an konsequent und explizit für die Größenmaßstäbe und die Dynamik eines P2P-Verbundes entworfen.

Nach einer Übersicht über verwandte Arbeiten in Abschn. 2 stellt Abschn. 3 den Chord-Algoritmus vor. Abschnitt 4 liefert eine kurze Einführung in die Grundlagen des Information-Retrievals, die für den weiteren Verlauf des Artikels benötigt werden. Abschnitt 5 stellt die Systemarchitektur vor; Abschnitt 6 formalisiert diese. Abschnitt 7 skizziert die Implementierung von MINERVA, die als Grundlage der Evaluation der verschiedenen Strategien zur Auswahl der Datenbanken dient, die Abschn. 8 vorstellt. Abschnitt 9 zeigt erste Messergebnisse, die mit der prototypischen Implementierung auf realen Daten erzielt wurden. Abschnitt 10 geht kurz auf das Problemfeld bezüglich der Zusammenführung von Resultatslisten ein. Abschnitt 11 schließt mit einem Ausblick auf zukünftige Forschungsaspekte diesen Artikel ab.

2 Verwandte Arbeiten

Die aktuelle Forschung auf dem Gebiet von P2P-Systemen (z.B. Chord [38], CAN [35], Pastry [34], P2P-Net [4] oder P-Grid [3]) baut zumeist auf verteilten Hash-Tabellen auf, die eine Zuordnung von Schlüsseln (z.B. Titeln oder Autoren) zu Peers in einer verteilten Weise erlauben und gleichzeitig gut mit der Gesamtgröße des Verbundes skalieren. Die Suche nach den jeweils zuständigen Peers für einen solchen Schlüssel in einem Verbund aus n Peers kann typischerweise in $O(\log n)$ Schritten ausgeführt werden, während gleichzeitig kein Peer mehr als $O(\log n)$ Verwaltungsinformationen speichern muss. Diese Systeme können auch mit einer hohen Dynamik umgehen, d.h. mit dem häufigen Kommen und Gehen von Peers in unvorhersehbarer Weise und ohne Vorankündigung. Allerdings sind diese Ansätze auf exakte Anfragen nach einzelnen Schlüsseln beschränkt. Dies ist unzureichend für Websuche, bei der Anfragen nach Relevanz sortierte Listen von Ergebnissen liefern sollen [11].

Im Folgenden gehen wir kurz auf bestehende Ansätze im Bereich P2P-Websuche ein. Galanx [42] ist eine P2P-Suchmaschine, die auf dem Webserver Apache sowie auf der Datenbank BerkeleyDB aufsetzt. Dabei werden Anfragen ähnlich wie in unserem Ansatz zu potenziell relevanten Peers weitergeleitet. Da Webserver jedoch typischerweise weniger volatil sind als autonome Peers, treten die Aspekte der Dynamik innerhalb des Verbundes in den Hintergrund.

PlanetP [7] ist eine Publish-Subscribe-Architektur für P2P-Communities und die erste ihrer Art, die eine Suche mit Ranking nach Relevanz der Inhalte unterstützt. Dabei unterscheidet PlanetP lokale Indexe und ein globales Verzeichnis, das alle Peers und ihre verfügbaren Informationen beschreibt. Zur Verbreitung dieses Verzeichnisses wird ein Gossiping-Algorithmus verwendet, d.h. jeder Peer verbreitet seine Informationen großflächig und unstrukturiert über das gesamte Netzwerk. Das System skaliert daher nur bis zu einer Größe von wenigen tausend Peers.

Odissea [39] benutzt eine zweischichtige Architektur, bei der globale Indexlisten über die Peers verteilt werden. Die Anfrageausführung benutzt eine verteilte Variante des Schwellwertalgorithmus, der u.a. von Fagin [16] eingeführt wurde. Das System erzeugt jedoch eine hohe Netzwerkauslastung, insbesondere beim Verteilen der Indexlisten. Außerdem scheinen Anfragen derzeit auf maximal zwei Suchterme beschränkt zu sein.

Das in [36] beschriebene System benutzt einen invertierten Index, der über die Peers verteilt wird. Besonderes Augenmerk wird dabei auf Techniken zur Verringerung der benötigten Bandbreite bei der Ausführung von Anfragen gelegt. [25] betrachtet hybride P2P-Systeme, in denen ein Peer entweder ein einfacher Knoten oder ein Verzeichnisknoten sein kann. Die Verzeichnisknoten arbeiten dabei als sogenannte Super-Peers, was möglicherweise die Skalierbarkeit

und die Fähigkeit zur Selbstorganisation des Gesamtsystems einschränkt. Die Auswahl geeigneter Datenbanken basiert auf der Kullback-Leibler-Divergenz zwischen den Termverteilungen der einzelnen Peers.

Verschiedene Strategien zur Verteilung von Anfragen über einfache Schlüsselwortsuchen hinaus wurden vielfach diskutiert [8, 9, 45], unterstützen aber kein Ranking nach Relevanz und sind daher für unsere Zwecke nicht geeignet. Die Konstruktion semantischer virtueller Netzwerke (Semantic Overlay Networks) mit Hilfe von Clustering- bzw. Klassifikationstechniken wird in [10, 26] betrachtet und ist orthogonal zu unserem Ansatz. [2] diskutiert den Aufbau skalierbarer Formen von semantischen virtuellen Netzen und identifiziert Strategien zu ihrer Nutzung bei der Anfrageausführung. [41] verteilt einen auf LSI-Techniken (Latent Semantic Indexing) beruhenden globalen Index auf die Peers, indem LSI-Dimensionen mit Hilfe der verteilten Hash-Tabelle CAN verteilt werden. In diesem Ansatz geben die Peers ihre Autonomie auf und sind zur Zusammenarbeit verpflichtet.

Neben der aktuellen Forschung im Bereich der P2P-Websuche sind auch frühere Arbeiten zu verteilter Informationssuche und Meta-Suchmaschinen relevant. [6] gibt einen Überblick über Methoden zur Kombination verteilter Anfrageergebnisse. [19] stellt ein formales Entscheidungsmodell für die Auswahl von Datenbanken im Kontext verteilter Informationssuche vor. [32] untersucht verschiedene Qualitätsmaße für die Auswahl von Datenbanken. [21,29] untersuchen die Skalierbarkeit verteilter Indexe.

Einen guten Überblick über existierende Techniken von Meta-Suchmaschinen gibt [31]. [43] evaluiert Strategien zur effizienten Bestimmung guter Suchmaschinen für eine konkrete Anfrage. Ungeachtet der Relevanz dieser früheren Arbeiten stellt die kollaborative Suche in P2P-Netzwerken allerdings eine noch größere Herausforderung dar, da die vorherigen Arbeiten sich typischerweise auf einen stabilen Verbund von recht wenigen Suchmaschinen beziehen – im Gegensatz zur gewünschten Skalierbarkeit und Dynamik in P2P-Netzwerken.

3 Chord – Ein skalierbares Lookup-Protokoll für P2P-Systeme

Das effiziente Auffinden von Knoten in einer P2P-Architektur ist ein fundamentales Problem, das von verschiedensten Seiten angegangen worden ist. Frühe (aber nichtsdestotrotz verbreitete) Systeme wie Gnutella verlassen sich dabei auf eine unstrukturierte Topologie, in der ein Peer alle Nachrichten an alle ihm bekannten Nachbarn (oder eine zufällige Auswahl von Nachbarn) weiterleitet. Typischerweise enthalten diese Nachrichten einen Zähler, der bei jeder Weiterleitung um eins vermindert wird. Auch wenn experimentelle Ergebnisse zeigen, dass dieses *Message Flooding* (oder *Gossiping*) in den meisten Fällen erstaunlich gut funktioniert, gibt es keine Garantien, dass alle betroffenen Knoten letztlich die Nachricht erhalten. Außerdem widerspricht die Vielzahl unnötiger Nachrichten unserem Ziel einer effizienten Architektur.

Chord [38] ist ein verteiltes Lookup-Protokoll, das dieses Problem angeht. Es stellt die Funktionalität einer verteilten Hash-Tabelle (*Distributed Hash Table*, DHT) zur Verfügung, indem es Schlüssel jeweils eindeutig bestimmten Knoten innerhalb des Netzwerkes zuweist. Zu diesem Zweck benutzt Chord konsistentes Hashing [24]. Konsistentes Hashing verteilt mit hoher Wahrscheinlichkeit die Last gleichmäßig über das Netzwerk, da jeder Knoten in etwa für die gleiche Anzahl von Schlüsseln verantwortlich ist. Diese Lastbalancierung arbeitet insbesondere auch bei einer sich dynamisch verändernden Zielmenge an Peers, d.h. wenn Knoten den Verbund verlassen oder ihm beitreten.

Chord kann nicht nur garantieren, den für einen bestimmten Schlüssel zuständigen Knoten zu finden, sondern tut dies auch sehr effizient. In einem stabilen Verbund aus N Knoten speichert jeder Knoten nur Informationen über $O(\log N)$ andere Knoten, und jede beliebige Anfrage kann durch $O(\log N)$ Nachrichten innerhalb des Verbundes beantwortet werden. Diese Eigenschaften erlauben potenziell hochgradig skalierende Anwendungen.

Das Konzept hinter Chord lässt sich einfach veranschaulichen: Alle Knoten p_i und alle Schlüssel k_i werden mit Hilfe einer Hashfunktion auf denselben Bereich eindeutiger numerischer Bezeichner abgebildet. Da alle Operationen modulo einer fest vorgegebenen großen Konstante (der maximalen Anzahl von Peers) durchgeführt werden, ergibt sich ein zyklischer Raum von Bezeichnern, der sogenannte *Chord-Ring*. Im Folgenden verwenden wir diese numerischen Bezeichner synonym zu ihren ursprünglichen Knoten bzw. Schlüsseln, d.h. wir verzichten aus Gründen der Übersichtlichkeit auf die explizite Darstellung der Hashfunktion. Jeder Schlüssel k_i wird seinem nächsten nachfolgenden Knoten auf dem Chord-Ring zugeordnet, d.h. ein Knoten ist zuständig für den Bereich zwischen seinem Vorgängerknoten und sich selbst. In Abb. 1 verteilen sich 10 Knoten über den Chord-Ring. Für den Schlüssel k_{54}, zum Beispiel, ist der Knoten p_{56} zuständig, da er der nächste folgende Knoten auf dem Chord-Ring ist.

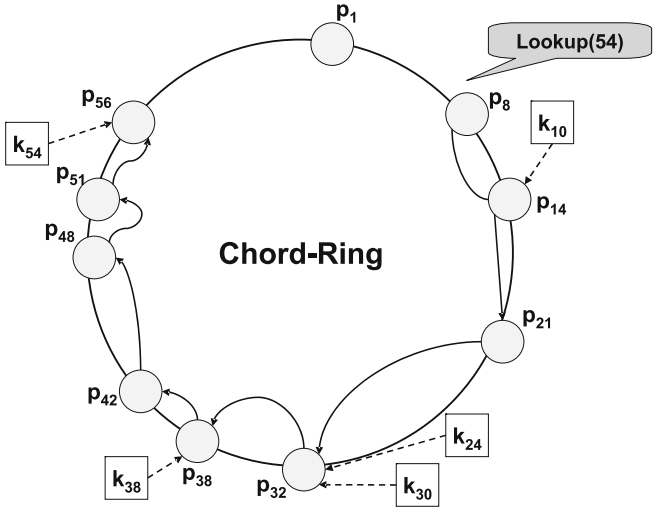

Abb. 1 Chord-Architektur

Abbildung 1 illustriert auch ein naives Verfahren zum Auffinden des für einen Schlüssel zuständigen Knotens. Wenn jeder Knoten seinen Nachfolger auf dem Chord-Ring kennt, kann eine Anfrage nach k_{54} solange um den Ring herum weitergeleitet werden, bis ein Paar von Knoten gefunden wird, zwischen denen sich der Schlüssel k_{54} befindet. Der zweite Knoten dieses Paares (p_{56}) ist der derzeit zuständige Knoten für diesen Schlüssel. Dieses Verfahren ähnelt der Suche in einer linearen Liste, welche den zuständigen Knoten in erwarteten O(N) Schritten finden kann, wobei jeder Knoten lediglich O(1) Informationen über andere Knoten (nämlich über seinen Nachfolger) benötigt.

Um die Suche zu beschleunigen bedient sich Chord zusätzlicher Informationen über die aktuelle Netzwerk-Topologie. Jeder Knoten p_i unterhält eine sogenannte *Fingertabelle*. Der m-te Eintrag der Fingertabelle eines Knotens p_i enthält einen Zeiger auf den ersten Knoten p_j, der von p_i auf dem Chord-Ring mindestens 2^{m-1} entfernt liegt. Dieses Schema besitzt folgende wichtige Eigenschaften: Jeder Knoten speichert nur wenige Zeiger und kennt dabei seine eigene Umgebung besser als entfernte Bereiche auf dem Chord-Ring. Gleichzeitig enthält die Fingertabelle jedoch typischerweise nicht genug Informationen, um den zuständigen Knoten für einen beliebigen Schlüssel *direkt* bestimmen zu können. Da jedoch jeder Knoten diese Zeiger in Zweierpotenz-Intervallen um den Chord-Ring herum unterhält, kann eine Nachricht mit jeder Weiterleitung mindestens die Hälfte der verbleibenden Distanz überbrücken. Diese Eigenschaft ist in Abb. 2 für Knoten p_8 dargestellt. Es folgt unmittelbar, dass die erwartete Zahl der Nachrichten, um den zuständigen Knoten für einen beliebigen Schlüssel in einem Netzwerk aus N Knoten zu finden, $O(logN)$ beträgt.

Chord beinhaltet ein Stabilisierungsprotokoll, das jeder Knoten periodisch im Hintergrund ausführt. Es aktualisiert die Einträge der Fingertabelle sowie die Zeiger auf die direkten Nachfolger, so dass Chord die Korrektheit seiner Lookup-Operation auch bei einem sich dynamisch ändernden Verbund garantieren kann. Insbesondere lässt sich zeigen, dass selbst mit veralteten Einträgen in der Fingertabelle die Leistung des System nur sehr langsam nachlässt.

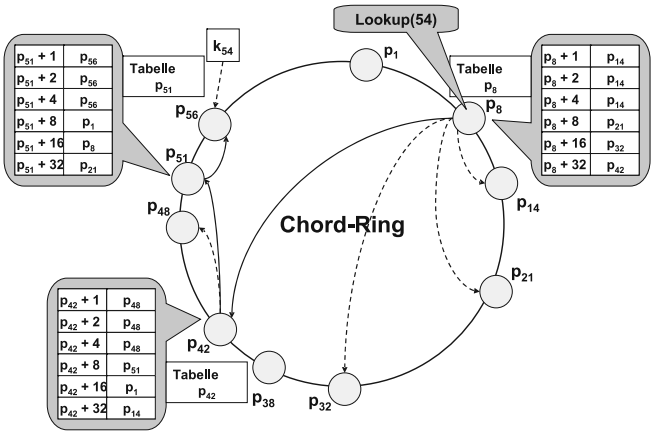

Abb. 2 Skalierbare Lookup-Operationen mit Hilfe von Finger-Tabellen

Chord kann seine Lookup-Operation den verschiedensten Anwendungen zur Verfügung stellen, z.B. verteilten Dateisystemen. Chord alleine ist jedoch noch keine Suchmaschine, da es nur exakte Anfragen auf einzelnen Schlüsseln erlaubt und kein Ranking von Ergebnissen unterstützt.

4 Kurze Einführung in Information-Retrieval

In diesem Abschnitt geben wir einen sehr kurzen Überblick über Information-Retrieval-Techniken, auf denen die weiteren Abschnitte dieses Artikels aufbauen.

Information-Retrieval-Systeme speichern große Mengen an schwach struktrierten oder unstrukturierten Daten, wie z.B. Textdokumente oder Webseiten in HTML. Sie stellen Suchfunktionen zur Verfügung, mit deren Hilfe sich relevante Dokumente zu einer Anfrage berechnen lassen. Typische Vertreter solcher Systeme sind Websuchmaschinen oder Digitale Bibliotheken. In der jüngsten Vergangenheit integrieren auch immer mehr relationale Datenbanksysteme solche Funktionen.

Die Suchfunktionalität wird durch Ähnlichkeitsmaße implementiert, die die Ähnlichkeit von Anfragen zu den Dokumenten berechnen. Im Rahmen von textbasierter Informationssuche mit einfachen Schlüsselwortanfragen repräsentieren diese Ähnlichkeitsmaße typischerweise die Häufigkeit des Auftretens der Suchterme in einem Dokument sowie die relative Lage der Suchterme zueinander innerhalb eines Dokuments. Abschnitt 4.1 stellt das Konzept der *invertierten Indexlisten* vor, welche eine effiziente Anfrageausführung ermöglichen. Abschnitt 4.2 gibt eine kurze Einführung in das populärste Ähnlichkeitsmaß, das sogenannte *TF*IDF*-Maß. Für nähere Details verweisen wir den Leser auf [11, 30].

4.1 Invertierte Indexlisten

Das Konzept invertierter Indexlisten wurde entwickelt, um effizient solche Dokumente innerhalb einer Dokumentmenge zu identifizieren, die einen konkreten Suchterm enthalten. Zu diesem Zweck bilden alle Terme, die innerhalb der Dokumentmenge vorkommen, eine indexartige Baumstruktur (oftmals einen B*-Baum), bei der die Blätter jeweils eine Liste von eindeutigen Bezeichnern genau jener Dokumente beinhalten, die einen Term enthalten (vgl. Abb. 3). Über die Listen aller Terme einer Anfrage wird konzeptionell ein Schnitt oder eine Vereinigung gebildet, um mögliche Treffer für die Anfrage zu identifizieren. In Abhängigkeit von der konkreten Ausführungsstrategie werden die Dokumente innerhalb der Listen nach ihren Bezeichnern oder nach ihrer Ähnlichkeit zum jeweiligen Term geordnet, was eine noch effizientere Abarbeitung der Indexlisten ermöglicht.

4.2 *TF*IDF*-Maß

Die Häufigkeit des Vorkommens eines Terms t in einem Dokument d wird als *Termhäufigkeit* ($tf_{t,d}$) bezeichnet. Intuitiv

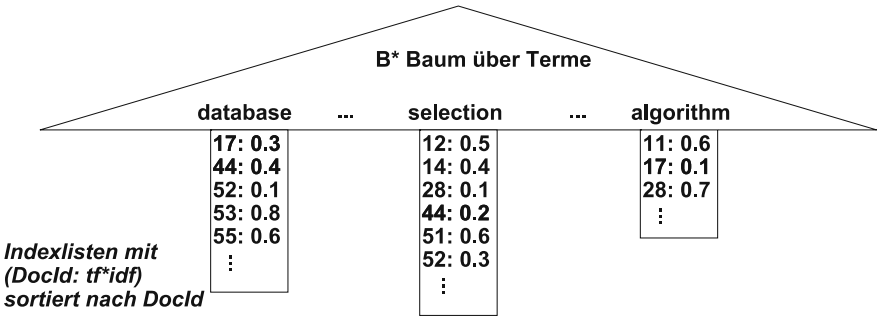

Abb. 3 B*-Baum mit invertierten Indexlisten

steigt die Relevanz eines Dokumentes mit steigender Termhäufigkeit für einen Suchterm. Die Anzahl der Dokumente innerhalb einer Dokumentmenge, die den Term t enthalten, wird als *Dokumenthäufigkeit* (df_t) bezeichnet; die *inverse Dokumenthäufigkeit* (idf_t) eines Terms ist das Verhältnis der Gesamtzahl aller Dokumente zur Dokumenthäufigkeit. Intuitiv gesprochen sinkt die relative Wichtigkeit eines Suchterms innerhalb einer Anfrage, je mehr Dokumente diesen Term enthalten, d.h. der Suchterm bietet ein geringeres Unterscheidungspotential zwischen den Dokumenten. In der Praxis werden diese beiden Maße normalisiert (z.B. auf das Intervall (0,1]) und/oder logarithmisch gedämpft. Ein typischer Vertreter dieser Familie von *TF*IDF*-Maßen berechnet das Gewicht $w_{i,f}$ des i-ten Suchterms im j-ten Dokument wie folgt:

$$w_{i,j} := \frac{tf_{i,j}}{\max_t\{tf_{i,j}\}} \times \log\left(\frac{N}{df_i}\right) \quad (1)$$

wobei N die Gesamtanzahl der Dokumente in der Dokumentensammlung ist.

In jüngerer Zeit haben weitere Relevanzmaße basierend auf statistischen Sprachmodellen und probabilistischer Informationssuche sehr große Beachtung gefunden [12, 19]. Im Rahmen dieses Artikels beschränken wir uns jedoch auf das verbreitete *TF*IDF*-Maß.

4.3 Top-k Anfrageausführung

Ein guter Algorithmus zur Anfrageausführung sollte es vermeiden, die Indexlisten für alle Suchterme *komplett* zu lesen; vielmehr sollte er idealerweise die investierte Arbeit auf O(k) begrenzen, wobei k die Anzahl der gewünschten Resultate beschreibt. In der Literatur wurden verschiedene Algorithmen vorgeschlagen, die diesem Ziel nahekommen. Der bekannteste seiner Art ist der sogenannte Schwellwert-Algorithmus (*Threshold Algorithm*, TA) von Fagin [18], der unabhängig auch von Nepal et al. [33] sowie von Güntzer et al. [20] entwickelt wurde. Er benutzt absteigend nach Termgewicht sortierte invertierte Indexlisten unter der zusätzlichen Annahme, dass die endgültige Relevanz des Dokuments mit Hilfe einer monotonen Aggregationsfunktion (z.B. einer gewichtete Summe) berechnet wird. TA traversiert alle invertierten Indexlisten abwechselnd, d.h. die Listen werden in der Regel durch (schnelle) sortierte Zugriffe gelesen. Für jedes zuvor noch nicht gesehene Dokument d benutzt TA einen Direktzugriff auf die übrigen Indexlisten, um die endgütige Relevanz des Dokumentes zu bestimmen. Diese Informationen werden in einer Kandidatenmenge gespeichert. Zusätzlich berechnet der Algorithmus eine obere Schranke für das maximale Gesamtgewicht noch nicht gesehener Dokumente und kann daher das Traversieren der Indexlisten abbrechen, sobald keines dieser unbekannten Dokumente mehr seinen Weg in die Kandidatenmenge finden kann. Probabilistische Methoden, die dieses Verfahren noch weiter beschleunigen können, werden in [40] vorgestellt.

5 Systemdesign

Abbildung 4 illustriert unseren Ansatz, der auf Chord aufbaut und einer Publish-Subscribe-Architektur ähnelt. Wir setzen eigene lokale Indexe voraus, welche von externen Crawlern importiert werden können. Ein konzeptionell zentrales, aber physisch verteiltes Verzeichnis speichert kompakte, aggregierte Meta-Informationen über diese lokalen Indexe der Peers (bzw. über von den Peers selbst gewählte Teilmengen davon). Wir benutzen eine verteilte Hash-Tabelle zur Verteilung der Terme, so dass jeder Peer für eine randomisierte Teilmenge aller Terme innerhalb des Verzeichnisses verantwortlich ist.

Jeder Peer veröffentlicht nun eine Zusammenfassung (*Post*) zu jedem Term in seinem lokalen Index (Abb. 4, Schritt 0). Diese wird jeweils an denjenigen Peer weitergeleitet, der gerade für diesen Term verantwortlich ist. Dieser Peer verwaltet eine *PeerListe* aller Posts zu diesem Term aus dem gesamten Verbund. Zur Erhöhung der Ausfallsicherheit können diese Listen auch über mehrere Peers repliziert werden. Posts enthalten Kontaktinformationen des absendenden Peers sowie statistische Angaben über den Term innerhalb des lokalen Indexes.

Der Ablauf einer Anfrage mit mehreren Suchtermen sieht wie folgt aus. Zunächst besorgt sich der anfragende Peer die PeerListen für alle Suchterme (Abb. 4, Schritt 1). Durch die Anwendung von Methoden zur Auswahl geeigneter Datenbanken ermittelt er die vielversprechendsten Peers für die Anfrage (*Datenbankselektion*)[3]. Danach wird die Anfrage an die ausgewählten Peers weitergeleitet, die die Anfrage auf

[3] Abschnitt 8 erläutert nähere Details zur Datenbankselektion

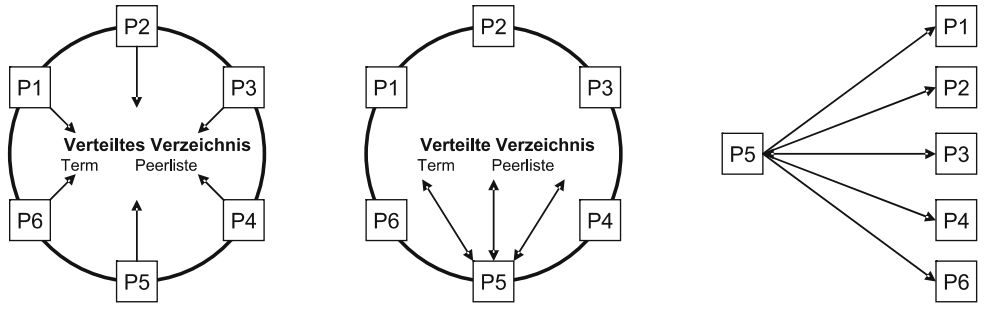

Abb. 4 P2P-Anfrage-Routing

der Basis ihrer lokalen Indexe ausführen (Abb. 4, Schritt 2). Diese Kommunikation findet in dieser Phase durch Punkt-zu-Punkt-Verbindungen statt. Dies erlaubt eine effiziente Nachrichtenübermittlung und verringert die Last auf dem verteilten Verzeichnis. Schließlich werden die Resultate der verschiedenen Peers zu einer gemeinsamen Resultatsliste zusammengeführt (*Query Result Merging*[4]).

Das Ziel, qualitativ hochwertige Ergebnisse in Bezug auf Präzision und Ausbeute zu erzielen, lässt sich nur schwer mit dem Wunsch nach a priori unbegrenzter Skalierbarkeit in Einklang bringen, da die besten Methoden zur Informationssuche umfangreiche Statistiken und globale Indexe benötigen. Durch die Beschränkung auf kompakte, aggregierte Zusammenfassungen und durch die Begrenzung der tatsächlich an der Anfrageausführung beteiligten Peers durch die Datenbankselektion halten wir die Größe des globalen Verzeichnisses und den auftretenden Netzwerkverkehr gering. Gleichzeitig können wir bei der eigentlichen Anfrageausführung von den lokalen Indexen profitieren. Dieser Ansatz lässt eine gute Skalierbarkeit des Gesamtsystems erwarten.

Unser Ansatz lässt sich leicht durch zusätzliche Verzeichnisse erweitern, die neben den statistischen Zusammenfassungen weitere Daten bereithalten, z.B. lokale Lesezeichen (Bookmarks) der Peers [5], Informationen über vorherige Suchen in Form von Query-Logs und Click-Streams [28], oder explizite Dokumentbewertungen von Benutzern. Diese Informationen lassen sich zur weiteren Verbesserung der Datenbankselektion nutzen.

6 Formalisiertes Systemmodell

In diesem Abschnitt formalisieren wir unsere Systemarchitektur. Sei $P := \{p_i | 1 \leq i \leq r\}$ die Menge der aktuell im System befindlichen Peers. Sei $D := \{d_i | 1 \leq i \leq n\}$ die Menge aller im System indexierten Dokumente und sei $T := \{t_i | 1 \leq i \leq m\}$ dazu analog die Menge aller Terme.

Jeder Peer p_i besitzt einen lokalen Index für eine Teilmenge T_i aller Terme (üblicherweise ist $|T_i| \ll |T|$). Dieser lokale Index enthält Statistiken $s_t \in S$ für jeden Term t aus der Menge der indexierten Dokumente $D_i \subseteq D$ (gewöhnlich gilt $|D_i| \ll |D|$).

Eine Hash-Funktion $hash : T \rightarrow Id$ wird benutzt, um die Terme auf die verfügbaren Peers zu verteilen. Dies geschieht durch die Zuordnung eines Bezeichners (id) pro Term. Die zugrunde liegende Hash-Tabelle besitzt eine Funktion $lookup : Id \rightarrow P$, die denjenigen Peer p_i zurückliefert, der momentan für einen Bezeichner zuständig ist.

Ein *PeerListRequest* (plr) an einen Peer p_i bezüglich eines Terms t ist wie folgt definiert: Die Funktion $plr : T \times P \rightarrow 2^{P \times S}$ liefert diejenigen Tupel (p_j, s_t) für alle p_j, die zuvor lokale Statistiken zu dem Term t an p_i gesendet haben. Ein Funktionsaufruf $lookup(hash(t))$ wird benutzt, um den derzeit zuständigen Peer für Statistiken bezüglich des Terms $t \in T$ zu bestimmen.

Zum Aufbau des globalen Verzeichnisses publiziert jeder Peer p_i Statistiken s_t für alle $t \in T_i' \subseteq T_i$ (T_i' ist dabei diejenige selbstgewählte Teilmenge von T_i, über die Peer p_i Informationen veröffentlichen möchte), die das Verzeichnis wie folgt erzeugen:

$$systerms : T \rightarrow 2^{P \times S}$$
$$systerms(t) := plr(t, lookup(hash(t)))$$

Das Verzeichnis stellt also eine Abbildung von Termen nach PeerListen zur Verfügung.

Wir betrachten eine Anfrage q als eine Menge von (*Term, Termgewicht*)-Paaren und folglich die Menge aller möglichen Anfragen als $Q := 2^{T \times \mathbb{R}}$. Um nun eine Anfrage q auszuführen muss im Schritt der Datenbankselektion zunächst eine Menge vielversprechender Peers identifiziert werden. Dies geschieht mit Hilfe einer Funktion

$$selection : Q \rightarrow 2^P$$
$$selection(q) := comb(\bigcup_{(t,w) \in q} systerms(t), q)$$

die eine Teilmenge aller Peers dadurch bestimmt, dass sie die Resultate von *systerms* mit Hilfe einer geeigneten Funktion $comb : 2^{P \times S} \times Q \rightarrow 2^P$ verknüpft.

[4] siehe auch Abschn. 10

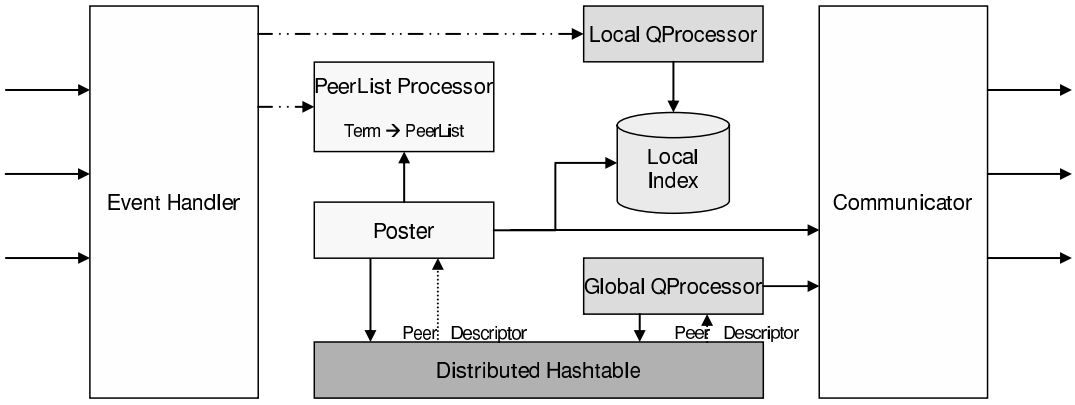

Abb. 5 Systemarchitektur

Die Ausführung einer Anfrage q beschreibt die Funktion $exec : Q \times 2^P \rightarrow \langle D \rangle$ ($\langle D \rangle$ ist eine Liste von Elementen $d \in D$), die die Anfrage an die zuvor durch $selection(q)$ ausgewählten Peers weiterleitet und schließlich die jeweiligen lokalen Resultate zu einer gemeinsamen Resultatsliste zusammenführt. Schließlich definieren wir die Funktion zur globalen Ausführung einer Anfrage q als $result : Q \rightarrow \langle D \rangle$ wie folgt:

$$result(q) := exec(q, selection(q))$$

7 Implementierung

Abbildung 5 zeigt den Aufbau eines Peers. Er basiert auf einer verteilten Hash-Tabelle, die das globale Verzeichnis implementiert. Dieses Verzeichnis liefert eine Beschreibung des zuständigen Peers für einen Term zurück, einen sogenannten *PeerDescriptor*. Ein *Communicator* wird instanziiert, um Nachrichten an andere Peers zu verschicken. Auf der Gegenseite besitzt jeder Peer einen *EventHandler*, der eingehende Nachrichten empfängt und an die jeweilige lokale Komponente weitergibt.

Jeder Peer besitzt einen lokalen Index. Dieser wird von der eigenen Suchkomponente (*Local QueryProcessor*) benutzt, um Anfragen lokal zu bearbeiten, sowie vom *Poster*, der für das Versenden der aggregierten und termspezifischen Zusammenfassungen an das globale Verzeichnis zuständig ist. Dazu benutzt der Poster wiederum die verteilte Hash-Tabelle zum Auffinden des derzeit für einen Term zuständigen Peers. Der *PeerListProcessor* jenes Peers verwaltet die eingehenden Posts für diesen Term aus dem gesamten Verbund. Stellt ein Benutzer eine Anfrage, dann benutzt die Komponente zur globalen Suche (*Global QueryProcessor*) erneut die verteilte Hash-Tabelle, um mit Hilfe einer Instanz des Communicators für jeden Suchterm die derzeitige PeerListe zu erhalten. Nach der Datenbankselektion leitet der GlobalQueryProcessor die Anfrage an die ausgewählten Peers weiter, die wiederum die Anfrage mit Hilfe ihrer LocalQueryProcessors ausführen. Schließlich führt der GlobalQueryProcessor diese Ergebnisse zusammen und präsentiert sie dem Benutzer.

MINERVA benutzt eine Java-basierte Re-Implementierung von Chord als verteilte Hash-Tabelle, kann jedoch auch generell mit jeder beliebigen verteilten Hash-Tabelle benutzt werden, die eine $lookup(key)$-Methode unterstützt. Die Kommunikation unserer Implementierung basiert auf Sockets, aber andere Kommunikationsformen (z.B. Web-Services [1]) könnten einfach integriert werden. Abbildung 6 zeigt die grafische Benutzeroberfläche unseres Prototyps. Der Benutzer startet das System, indem er entweder einen neuen Chord-Ring erzeugt oder einem bestehenden Chord-Ring beitritt. Beide Operationen benötigen die Angabe eines lokalen Ports für die Chord-orientierte Kommunikation sowie eines weiteren Ports für die applikations-orientierte Kommunikation. Das Beitreten zu einem bestehenden Chord-Ring erfordert die Kenntnis eines beliebigen vorhandenen Peers, der sich bereits im Chord-Ring befindet. Statusinformationen über den Chord-Ring werden angezeigt und kontinuierlich aktualisiert. Der Bereich *Posts* zeigt Informationen über die Terme, für die der Peer derzeit zuständig ist, d.h. für die er Posts von anderen Peers empfangen hat. Die Schaltfläche Post startet das Publizieren der Zusammenfassungen über den lokalen Index an das verteilte Verzeichnis. Der Bereich *Queries* startet Anfragen mit beliebig vielen Suchtermen, die in ein Formularfeld eingegeben werden können. Die endgültigen Resultate werden sortiert nach ihrer Relevanz ausgegeben.

8 Methoden zur Datenbankselektion

Datenbankselektion ist das Problem, Peers zu finden, die Anfragen eines gegebenen Peers mit hoher Resultatsgüte bei geringen Kosten ausführen können. Unser Ansatz besteht dabei aus zwei Schritten:

- Die Identifizierung *möglicher* Kandidaten
- Die Ermittlung der *vielversprechendsten* Kandidaten durch Berechnung eines Gütemaßes, das die erwartete Resultatsgüte in Zusammenhang mit den zu erwartenden Kosten bringt

Der erste Schritt wird erreicht, indem alle veröffentlichten Posts zu den Suchtermen vom Verzeichnis angefordert

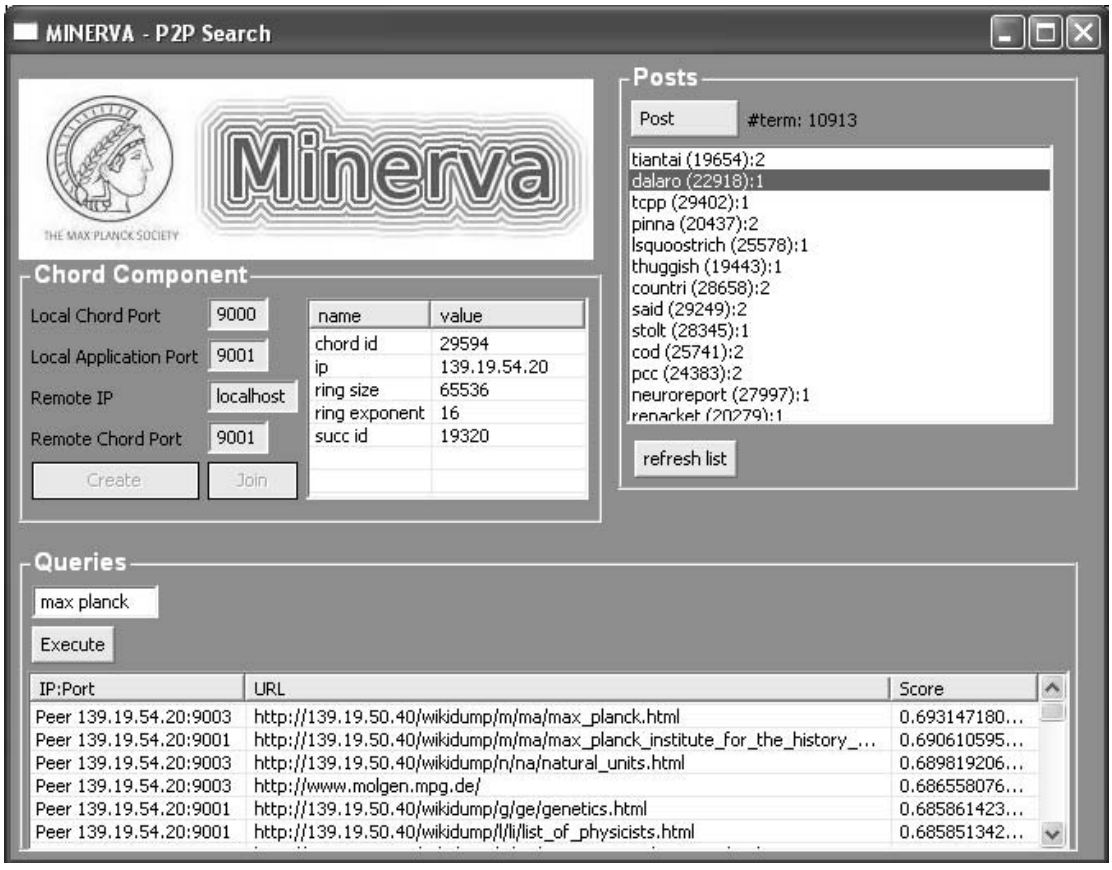

Abb. 6 Benutzeroberfläche des MINERVA-Prototyps

werden. Aus Effizienzgründen kann die Anzahl der zurückgelieferten Posts begrenzt werden, so dass nur die besten Posts (basierend auf einem beliebigen Gütemaß) zurückgeliefert werden. Im zweiten Schritt werden alle Posts eines Peers kombiniert, um die erwartete Resultatsgüte des Peers für diese konkrete Anfrage abzuschätzen (Datenbankselektion). Das eingangs erwähnte Gütemaß beschreibt dabei den erhofften Beitrag dieses Peers zu den endgültigen Resultaten, während die erwarteten Kosten den Ressourcenbedarf, die aktuelle Lastsituation des Peers oder seine Netzwerkanbindung berücksichtigen. Für diese dynamische Kostenabschätzung gibt es diverse Ansätze in der Literatur (siehe z.B. [27]). Die einfachste Herangehensweise ist es, die spezifischen Kapazitäten und Auslastungen der Peers sowie des Netzwerkes zu ignorieren und statt dessen anzunehmen, dass die gesamten Kosten der Anfrageausführung proportional mit der Anzahl der beteiligten Peers steigen. In diesem Artikel wählen wir diese vereinfachte Herangehensweise und konzentrieren uns auf die Betrachtung der erwarteten Resultatsgüte.

In den folgenden Abschnitten werden verschiedene Strategien vorgestellt, die sich zur Datenbankselektion eignen. Weitere Strategien wurden in der Literatur vorgeschlagen; zu nennen ist hier insbesondere auch der entscheidungstheoretische Ansatz von Fuhr [19]. In Analogie zur bestehenden Literatur bezeichnen wir die verschiedenen statistischen Maße als kollektions-spezifisch; in unserem Kontext ist dabei eine Kollektion der lokale Index eines Peers. Weiterhin betrachten wir im Folgenden nur Anfragen mit gleichgewichteten Termen, d.h. die Anfragen sind Mengen von Termen.

8.1 $cdf - ctf^{\max}$ Ansatz

Dieser einfache ad-hoc Ansatz kombiniert die *Dokumenthäufigkeit innerhalb einer Kollektion* (*cdf*) mit der *maximalen Termhäufigkeit einer Kollektion* ($ctf^{\max}$) (der maximalen Häufigkeit des Terms in einem Dokument in der Kollektion) und summiert diese Werte für alle Suchterme. *cdf* ist also für einen Term t bzgl. eines Peers p_i die Anzahl der Dokumente im lokalen Index von p_i, die den Term t enthalten; $ctf^{\max}$ ist für einen Term t bzgl. eines Peers p_i die Termhäufigkeit (*tf*) von t in genau jenem Dokument d des lokalen Indexes von p_i, welches die höchste *tf* für diesen Term t unter allen Dokumenten von p_i besitzt.

Die Ähnlichkeit s_i des i-ten Peers bzgl. einer Anfrage q wird demnach wie folgt berechnet:

$$s_i = \sum_{t \in Q} \alpha \cdot \log cdf_{i,t} + (1 - \alpha) \cdot \log ctf_{i,t}^{\max}$$

Der Parameter α kann dabei Werte zwischen 0 und 1 annehmen und bestimmt den relativen Einfluss von *cdf* bzw.

$ctf^{\max}$. Die Werte s_i werden für alle betrachteten Kollektionen berechnet; in absteigender Reihenfolge sortiert ergibt sich so die mit diesem Ansatz berechnete Kollektionsrangfolge.

8.2 CORI-artige Ansätze

In diesem Abschnitt betrachten wir zwei Ansätze, die auf den Arbeiten über das CORI-System [6, 13] basieren. Wir bezeichnen diese Ansätze als CORI1 bzw. CORI2.

*CORI*1

Dieser Ansatz berechnet die Ähnlichkeit s_i des i-ten Peers bzgl. einer Anfrage q wie folgt:

$$s_i = \sum_{t \in q} \frac{s_{i,t}}{|q|}$$

$$s_{i,t} = \alpha + (1 - \alpha) \cdot T_{i,t} \cdot I_{i,t}$$

Die Berechnungen von $T_{i,t}$ und $I_{i,t}$ benutzen dabei die maximale Größe des Chord-Rings als obere Schranke zur Abschätzung der *Anzahl der Peers* im System (np), die *Dokumenthäufigkeit innerhalb einer Kollektion* (cdf) und die *maximale Termhäufigkeit einer Kollektion* ($cdf^{\max}$):

$$T_{i,t} = \beta + (1 - \beta) \cdot \frac{\log(cdf_{i,t} + 0.5)}{\log(cdf_{i,t}^{\max} + 1.0)}$$

$$I_{i,t} = \frac{\frac{\log(np + 0.5)}{cf_t}}{\log(np + 1)}$$

wobei die *Kollektionshäufigkeit* cf_t die Anzahl der Kollektionen beschreibt, die den Term t enthalten. Wir schätzen diesen Wert durch die Länge der entsprechenden PeerListe für diesen Term t ab. Die Parameter α und β werden analog zu [13] als $\alpha = \beta = 0.4$ gewählt.

*CORI*2

Dieser Ansatz beruht auf der in [6] vorgestellten Methode und unterscheidet sich von CORI1 durch die Berechnung von $T_{i,t}$. Wir betrachten die Größe V_i des Termraums einer Kollektion (d.h. die Anzahl verschiedener Terme innerhalb ihres Indexes) und bilden den Durchschnitt V^{avg} über alle Kollektionen, die diesen Term t enthalten:

$$T_{i,t} = \frac{cdf_{i,t}}{cdf_{i,t} + 50 + 150 \cdot \frac{|V_i|}{|V^{avg}|}}$$

In der Praxis ist es schwierig, den Wert V^{avg} exakt zu bestimmen. Wir wählen daher als Abschätzung den Durchschnitt über alle Kollektionen, die wir bei der Ausführung einer Datenbankselektion in den PeerListen vorfinden.

8.3 GlOSS-artiger Ansatz

Diese Strategie basiert auf der Arbeit über das GlOSS-System [22]; wir bezeichnen diesen Ansatz als GlOSS-artig. Zunächst werden die Suchterme aufsteigend nach ihren cdf-Werten sortiert (der Einfachheit wegen nummerieren wir die Terme $t_1, t_2, \ldots, t_q$ entsprechend dieser Sortierung), d.h. für jedes Paar t_t, t_{t+1} gilt $cdf_t \leq cdf_{t+1}$.

In einem zweiten Schritt wird die durchschnittliche Termhäufigkeit (ctf_t^{avg}) pro Dokument innerhalb einer Kollektion berechnet, indem die *Häufigkeit eines Terms in einer Kollektion* (ctf) in Verhältnis zur *Dokumenthäufigkeit innerhalb einer Kollektion* (cdf_t) gesetzt wird:

$$ctf_t^{avg} = \frac{ctf_t}{cdf_t}$$

Nun berechnet sich das Ähnlichkeitsmaß des i-ten Peers bzgl. einer Anfrage q wie folgt:

$$s_i = \sum_{t=1}^{q} \left[\left(cdf_{i,t} - cdf_{i,t-1} \right) \cdot \sum_{u=t}^{q} \left(ctf_{i,u}^{avg} \cdot \log \left(\frac{|C_i|}{cdf_{i,u}} \right) \right) \right]$$

mit $cdf_{i,0} := 0$, wobei $|C_i|$ die Anzahl der Dokumente in der i-ten Kollektion darstellt.

8.4 Ansätze basierend auf statistischen Sprachmodellen

Die beiden nächsten Ansätze beruhen auf statistischen Sprachmodellen (Language Models, LM).

LM nach Callan

Basierend auf [37] berechnen wir das Ähnlichkeitsmaß einer Kollektion wie folgt:

$$s_i = \sum_{t \in Q} \log(\lambda \cdot s_{i,t} + (1 - \lambda) \cdot s_{GE,t})$$

wobei $s_{GE,t}$ das statistische Modell der englischen Sprache (*General English*) darstellt und λ einen Parameter zur Kalibrierung. Wir approximieren dieses Modell dadurch, dass wir die Häufigkeit eines Terms t innerhalb einer Kollektion durch die Gesamtanzahl der verschiedenen Terme in der Kollektion teilen. Weiterhin wählen wir $\lambda = 0.7$ und berechnen das Ähnlichkeitsmaß $s_{i,t}$ wie folgt:

$$s_{i,t} = \frac{ctf_{i,t}}{cs_i}$$

wobei cs_i die Größe der i-ten Kollektion als Summe der Termhäufigkeiten (*nicht*: die Anzahl verschiedener Terme!) darstellt.

LM nach Xu & Croft

Der zweite Ansatz basiert auf [44]. Wir berechnen die Distanz zwischen einer Anfrage und einer Kollektion wie folgt:

$$dist_i = \sum_{t \in Q} \left[\frac{1}{|q|} \cdot \log \left(\frac{1}{|q| \cdot s_{i,t}} \right) \right]$$

$$wobei\ s_{i,t} = \frac{ctf_{i,t} + 0.01}{c_{s,i} + 0.01 \cdot |V_i|}$$

Die Kollektionsrangfolge ergibt aus der Sortierung der Kollektionen in aufsteigender Reihenfolge dieser Distanzen.

9 Experimente

9.1 Aufbau

Für unsere Experimente werden 10 thematisch fokussierte Kollektionen benutzt, die aus Webcrawls ausgehend von manuell ausgewählten Startseiten zu den Themen Sport, Informatik, Unterhaltung und Gemischtes entstanden sind. Jede dieser Kollektionen wird im Rahmen der Experimente jeweils genau einem Peer zugeordnet. Zusätzlich wird eine Referenzkollektion erstellt, indem die einzelnen Kollektionen vereinigt (und Duplikate eliminiert) werden. Tabelle 1 zeigt Details der Kollektionen, insbesondere auch den Grad ihrer Überlappung.

Als Anfragen dienen uns die 7 häufigsten Anfragen der Websuchmaschine AltaVista für den 21. September 2004, welche wir von `http://www.wordtracker.com` übernommen haben, sowie 3 weitere ausgewählte Anfragen. Tabelle 2 zeigt alle Anfragen.

Tabelle 1 Statistiken zu den verwendeten Kollektionen

Kollektion	# Dokumente	Größe der Kollektionen in MB
Informatik 1	10459	137
Gemischtes 2	12400	144
Unterhaltung 2	11878	134
Sport 1	12536	190
Informatik 2	11493	223
Informatik 3	13703	239
Informatik 4	7453	695
Sport 2	33856	1086
Gemischtes 2	16874	809
Unterhaltung 2	18301	687
Σ	168953	4348
Referenzkollektion	142206	3808

Tabelle 2 Verwendete Anfragen (* kennzeichnet Anfragen, die *nicht* von WordTracker übernommen wurden)

Max Planck Light Wave Particle*	Einstein Relativity Theory*
Lauren Bacall	Nasa Genesis
Hainan Island	Carmen Electra
National Weather Service	Search Engines
John Kerry	George Bush Iraq*

Für jede Anfrage berechnen wir zunächst eine *ideale Kollektionsrangfolge* wie folgt: Die Anfrage wird zunächst auf der Referenzkollektion unter Anwendung der in Abschn. 4 vorgestellten Methoden ausgeführt und liefert ein *Referenzresultat*. Anschließend wird die Anfrage auf jeder Kollektion einzeln unter Anwendung der gleichen Strategie ausgeführt. Diese Ergebnisse werden jeweils mit Hilfe des in Abschn. 9.2 vorzustellenden Distanzmaßes mit dem Referenzergebnis verglichen. Die Kollektionen werden zur Erlangung der idealen Kollektionsrangfolge aufsteigend nach ihrer Distanz zum Referenzergebnis sortiert.

Wir evaluieren die Methoden zur Datenbankselektion, indem wir ihre Kollektionsrangfolgen mit dieser idealen Kollektionsrangfolge vergleichen (erneut durch Anwendung des Distanzmaßes aus Abschn. 9.2). Abbildung 7 illustriert den Versuchsaufbau.

Es gibt eine Reihe von Parametern, die die Ergebnisse der Experimente beeinflussen: Die Anzahl der Kollektionen in der idealen Kollektionsrangfolge, die Anzahl der Kollektionen, die die Methoden zur Datenbankselektion zurückliefern, die Anzahl der Dokumente, die von der Referenzkollektion zurückgeliefert werden, oder die Anzahl der Dokumente, die von den einzelnen Kollektionen zurückgeliefert werden.

In allen Experimenten werden 10 Peers (entsprechend der 10 Kollektionen) als separate Prozesse auf einem Notebook mit einem Pentium M 1,6 GHz Prozessor und 1 GB Hauptspeicher gestartet. Alle Peers greifen auf eine gemeinsame Oracle 10g Datenbank zurück, die auf einem separaten Server (Dual-Xenon, 3 GHz, 4 GB Hauptspeicher) installiert ist. Die Peers sind durch ein lokales 100 Mbit/s Netzwerk mit dem Server verbunden.

9.2 Distanzmaß

Formal betrachtet ist eine Rangfolge σ eine Bijektion von einer Definitionsmenge D_σ nach $[k]$, wobei $|D_\sigma| = k$ und $[k] := \{1, \ldots, k\}$. Metriken zum Vergleich solcher Rangfolgen sind schon lange Gegenstand intensiver Forschung [15, 23]. Eine der bekanntesten Metriken ist *Spearman's footrule metric* [14, 15], die die Differenz zwischen zwei Rangfolgen σ_1 und σ_2 mit $D = D_{\sigma_1} = D_{\sigma_2}$ wie folgt berechnet:

$$F(\sigma_1, \sigma_2) := \sum_{i \in D} |\sigma_2(i) - \sigma_1(i)| \quad (2)$$

Wir benötigen ein Distanzmaß zum einen beim (i) Vergleich der Resultatslisten der einzelnen Kollektionen jeweils mit der Resultatsliste der Referenzkollektion (zur Berechnung der idealen Kollektionsrangfolge) und zum anderen beim (ii) Vergleich verschiedener Kollektionsrangfolgen mit der idealen Kollektionsrangfolge. In unserem Fall sind die Definitionsmengen D_{σ_1} und D_{σ_2} also nicht notwendigerweise identisch; bezeichnen wir mit σ_1 jeweils die idealen Rangfolgen, so ist im Fall (ii) D_{σ_1} eine Obermenge von D_{σ_2}. Im

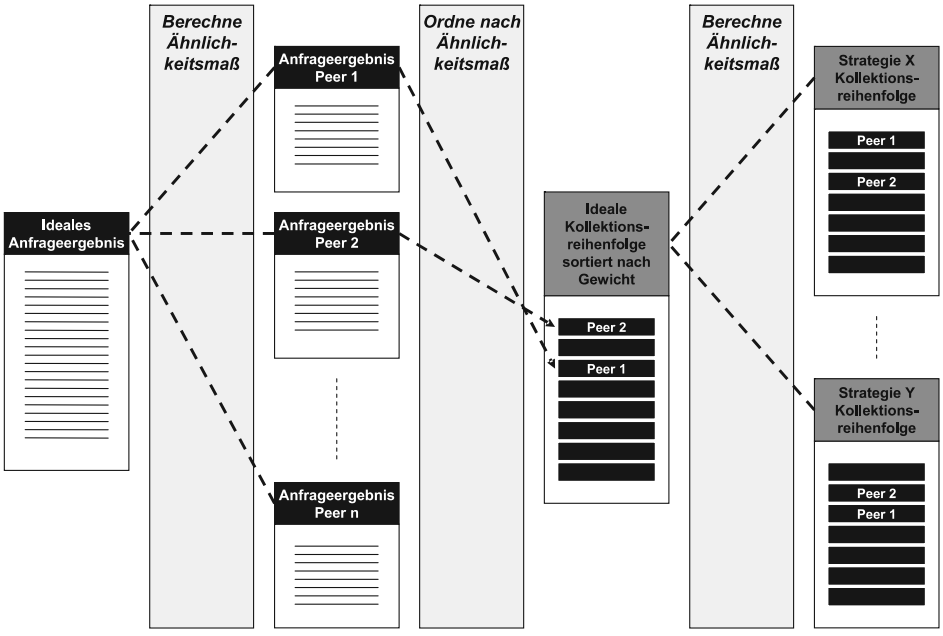

Abb. 7 Schematischer Aufbau der Experimente

Fall (i) kommt erschwerend hinzu, dass in der begrenzten Resultatsliste der Referenzkollektion nicht notwendigerweise alle Dokumente vorkommen, die von einzelnen Kollektionen in deren Resultatslisten geliefert werden. Daher können wir weder Spearman's footrule metric (2) noch andere bekannte Metriken (z.B. *Kendall's Tau* [23]) unmittelbar einsetzen. [17] liefert einen Überblick über Metriken, die Top-k-Listen als *unvollständige Rangfolgen* betrachten und Modifikationen verbreiteter Metriken vorstellen, die eine sinnvolle Anwendung erlauben. Eine Möglichkeit zur Anwendung der bekannten Metriken auf Top-k-Listen ist es, die Listen konzeptionell zu Listen über der *vereinigten* Definitionsmenge der beiden Rangfolgen zu erweitern.

Wir schlagen analog die folgende Formel zur Berechnung der Distanz zwischen zwei Rangfolgen σ_1 und σ_2 vor:

$$F'(\sigma_1, \sigma_2) := \sum_{i \in D_{\sigma_2}} |\sigma_2(i) - \sigma_1(i)| \qquad (3)$$

Auch wenn diese Formel auf den ersten Blick Spearman's footrule metric sehr ähnelt, so summiert sie im Unterschied zu ihr über alle Elemente aus D_{σ_2}. Allerdings ist (3) nur gültig, falls D_{σ_1} eine Obermenge von D_{σ_2} ist, da $\sigma_1(i)$ anderenfalls für alle $i \in D_{\sigma_2} \setminus D_{\sigma_1}$ undefiniert wäre. Daher benötigen wir die Definition einer Fortsetzung ${\sigma_1}'$ von σ_1 wie folgt, um die Obermengeneigenschaft herzustellen:

$${\sigma_1}'(i) := \begin{cases} \sigma_1(i) & i \in D_{\sigma_1} \\ |D_{\sigma_1}| + 1 & i \notin D_{\sigma_1} \end{cases}$$

Die Intuition hinter dieser Fortsetzung ist, dass sich alle Elemente in den Rangfolgen der Referenzrangfolgen befinden *müssen*. Wir wählen den kleinstmöglichen Rang $|D_{\sigma_1}|+1$ als Approximation des tatsächlichen Rangs. Zu beachten ist auch, dass wir keine Erweiterung von σ_2 benötigen, da wir nur über die Elemente aus D_{σ_2} summieren. Daher ist F' nicht symmetrisch.

Ein weiteres Problem ergibt sich, wenn *unterschiedlich große* Rangfolgen mit einer Referenzrangfolge verglichen werden sollen. Um Ungerechtigkeiten beim Vergleich von kurzen Rangfolgen (mit guten Resultaten) gegenüber langen Rangfolgen (mit weniger guten Resultaten im hinteren Bereich der Rangfolge) zu vermeiden, setzen wir eine Mindestlänge für jede Rangfolge σ_2 voraus und füllen kürzere Rangfolgen entsprechend mit künstlichen Elementen auf, die dem Rang $|D_{\sigma_1}| + 1$ in der Referenzrangfolge entsprechen.

9.3 Experimentelle Ergebnisse

Abbildung 8 stellt die Distanzen zwischen den von den verschiedenen Methoden zur Datenbankselektion ergebenen Kollektionsrangfolgen und der idealen Kollektionsrangfolge dar. Die Distanzen basieren auf dem in Abschn. 9.2 vorgestellten Distanzmaß und sind über unsere 10 Anfragen gemittelt. Für unsere Menge an Anfragen lieferte CORI2 die besten Ergebnisse und distanziert damit das GlOSS-artige Verfahren sowie die Ansätze basierend auf statistischen Sprachmodellen. Das Ad-hoc-Maß $cdf - ctf^{\max}$ hält sich erstaunlich gut. Das schlechte Abschneiden der Sprachmodelle widerspricht den Ergebnissen aus [37]; wir untersuchen derzeit die genauen Ursachen hierfür. Erste Ergebnisse deuten darauf hin, dass sich sowohl die von uns gewählten konkreten Anfragen als auch die zugrunde liegenden Kollektionen (durch ihre Überlappung) als ungünstig für diese Ansätze herausgestellt haben.

Auf den gleichen Kollektionen beobachten wir die Ausbeute in Relation zu den Top-30-Dokumenten der Referenzkollektion. Dafür senden wir die Anfrage nacheinander an alle Kollektionen in absteigender Reihenfolge der idealen

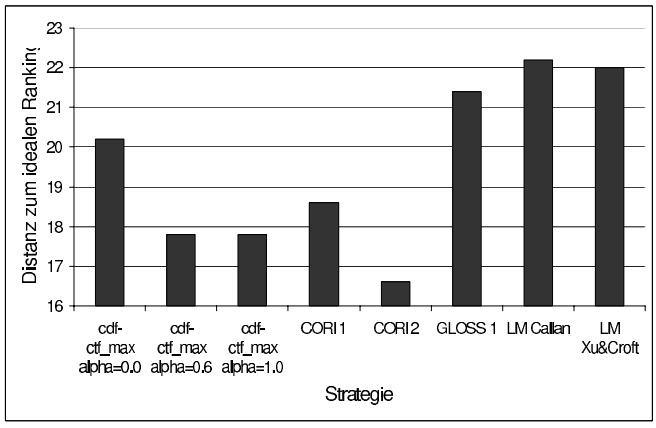

Abb. 8 Vergleich verschiedener Methoden zur Datenbankselektion

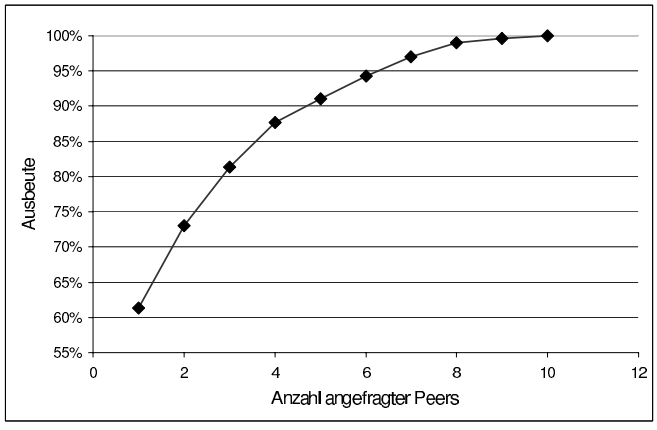

Abb. 9 Ausbeute (in %) in Abhängigkeit von der Anzahl befragter Peers

Kollektionsrangfolge. Abbildung 9 zeigt, dass bereits die beste Kollektion durchschnittlich etwa 60 % aller relevanten Ergebnisse liefert, während die schlechteren Kollektionen typischerweise nicht mehr viele relevante Dokumente beitragen. Dies bedeutet jedoch *nicht*, dass diese keine relevanten Dokumente enthalten, sondern lediglich, dass ihre relevanten Dokumente bereits zuvor von anderen Kollektionen beigesteuert worden sind. Dies unterstreicht die Wichtigkeit, die Überlappung der einzelnen Kollektionen bei der Datenbankselektion zu berücksichtigen.

Dies wird noch deutlicher, wenn man den Ressourcenverbrauch der Anfragen in Abhängigkeit von der Anzahl der erhaltenen relevanten Resultate bei steigender Anzahl an befragten Peers betrachtet. Abbildung 10 zeigt, dass bei unserem Versuchsaufbau auf einem einzelnen Notebook die Ausführungszeit annähernd linear zur Anzahl der befragten Peers verläuft. In der Realität erwarten wir, dass die Ausführungszeit nahezu konstant bleibt, da sich die Last über eine Anzahl von unabhängigen Prozessoren verteilt. Trotzdem sollte es in der Praxis vermieden werden, Peers zu befragen, die nur noch wenige neue Dokumente zur endgültigen Resultatsliste beitragen. Die dabei systemweit benutzen Ressourcen (CPU, Netzwerkverkehr) stehen in einem Missverhältnis zur erreichbaren Verbesserung des Resultats.

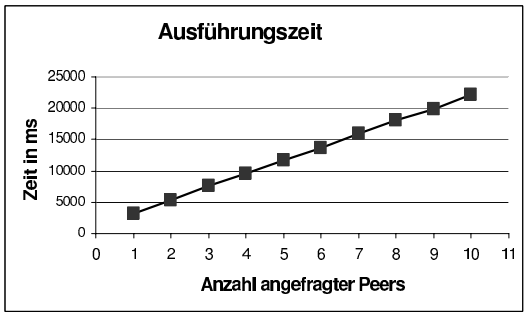

Abb. 10 Laufzeitverhalten

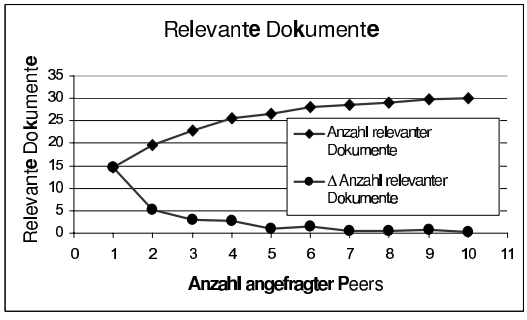

Abb. 11 Ausbeute (absolut und als Differenz) in Abhängigkeit von der Anzahl befragter Peers

In unserem Versuchsaufbau, der eine hohe Last auf dem verwendeten Notebook erzeugt, dauert die Ausführung einer Anfrage etwa 2 Sekunden pro angefragtem Peer. Die Ausführungszeit wird dabei klar dominiert von der Dauer der Anfrageausführung auf dem lokalen Index. Die von MINERVA benötigte Zeit zur Datenbankselektion ist vernachlässigbar. Abbildung 11 zeigt die Anzahl erhaltener relevanter Dokumente als Funktion der angefragten Peers. Wie erwartet tragen weniger relevante Kollektionen nur noch wenige relevante Dokumente zum endgültigen Ergebnis bei.

Wir betrachteten außerdem die Effizienz unseres Prototyps zur Laufzeit. Dazu beobachteten wir die folgenden Eigenschaften, die unabhängig von der gewählten Methode zur Datenbankselektion sind: Die Größe eines einzelnen Posts für einen Term beträgt etwa 20 Bytes. Die Gesamtmenge an Daten, die beim Publizieren der Posts über das Netzwerk gesendet werden muss, beläuft sich auf etwa 650 kB bei einer Kollektion mit 45 000 verschiedenen Termen. Dabei benutzen wir *gzip* zur Datenkompression.

Die Anfrage nach einer PeerListe benötigt etwa 150 Bytes. Die Größe der eigentlichen PeerListe ist abhängig von ihrer Länge und beträgt etwa 1000 Bytes für eine Liste von 10 Peers (wobei wir die maximale Länge der zurückgegebenen PeerListen zur Verbesserung der Systemleistung begrenzen können). Die eigentliche Suchanfrage (2 Suchterme) verursacht lediglich etwa 100 Bytes Netzwerkverkehr, die Rückgabe von 30 lokalen Resultaten jeweils etwa 2500 Bytes (darin sind allerdings noch umfangreiche Statistiken enthalten, die der anfragende Peer zum sinnvollen Kombinieren der lokalen

Ergebnisse bzw. zum Ranking der endgültigen Resultatsliste einsetzen kann).

Die Komplexität der Methoden zur Datenbankselektion kann grob mit $O(nl + m\log(m))$ angegeben werden, wobei m die Anzahl der Kollektionen darstellt, n die Anzahl der Suchterme in einer Anfrage und l die Länge der längsten PeerListe. nl ist dann eine obere Schranke für die Anzahl zu bearbeitender Posts, bevor die Kollektionsrangfolge (höchstens m Einträge) in $O(m\log(m))$ sortiert werden kann.

10 Zusammenführung der Resultatslisten

Neben dem Problem der Datenbankselektion (vgl. Abschn. 8) ergibt sich bei der Architektur von MINERVA ein weiterer wichtiger Aspekt, auf den in diesem Abschnitt kurz eingegangen werden soll. Die einzelnen angefragten Peers liefern ihre lokalen Resultate an den anfragenden Peer zurück. Dieser muss diese Listen zu einer gemeinsamen Resultatsliste zusammenführen. Dieser Vorgang wird auch als *Query Result Merging* bezeichnet. Beim Zusammenführen der Resultatslisten ergeben sich verschiedene Ansätze:

- Ansätze, die ausschließlich auf dokument- bzw. termspezifischen Statistiken der zurückgelieferten Dokumente aufbauen.
- Ansätze, die zusätzlich die im Rahmen der Datenbankselektion berechneten Gütemaße (z.B. basierend auf CORI oder statistischen Sprachmodellen) der einzelnen Kollektionen berücksichtigen [6, 37].

Wenn man voraussetzt, dass die einzelnen Peers im Rahmen der Anfrageausführung bereits ein (lokales) Gütemaß der von ihnen gelieferten Dokumente (z.B. basierend auf Termhäufigkeiten, vgl. Abschn. 4) berechnet haben, so besteht eine naive Möglichkeit zum Query Result Merging darin, alle Dokumente anhand dieses bereits vorhandenen Gütemaßes zu sortieren. In der Praxis wirft dieses Verfahren jedoch einige Probleme auf. Damit die einzelnen Werte auch über die Kollektionsgrenzen hinweg vergleichbar sind, würde dieses Verfahren eine systemweit einheitliche Strategie zur Berechnung der Gütemaße erfordern. Doch selbst dann wären die Gütemaße aufgrund verschiedener kollektions-spezifischer Statistiken innerhalb der Berechnung (z.B. lokaler Dokumenthäufigkeiten) nicht untereinander vergleichbar.

MINERVA bewertet Dokumente, indem es die Relevanz der Dokumente mit Hilfe der mitgelieferten Statistiken neu berechnet und sich nicht auf die von den Peers (u.U. mit verschiedenen Methoden) lokal berechneten Werte verlässt. Wir erreichen somit eine übereinstimmende Berechnung des Gütemaßes für alle Dokumente. Hierbei wird die *relative Termhäufigkeit* (*rtf*) eines Terms in einem Dokument mit der *Dokumenthäufigkeit innerhalb einer Kollektion* (*cdf*) und der *Anzahl der Dokumente in der Kollektion* ($|C|$) kombiniert. Die relative Termhäufigkeit ist hierbei der Quotient aus der Termhäufigkeit des Terms und der Gesamtzahl aller Wörter bezogen auf das jeweilige Dokument. MINERVA berechnet die Ähnlichkeit eines Dokumentes j, welches in der Resultatsliste von Peer p_i aufgeführt wird, zu einer Anfrage Q folgendermaßen:

$$s_{i,j} = \sum_{t \in Q} rtf_{t,j} \cdot \frac{\log \frac{|C_i|}{cdf_{i,t}}}{\log |C_i|}$$

Die Dokumente aus den verschiedenen Resultatslisten werden schließlich nach $s_{i,j}$ absteigend sortiert ausgegeben.

Dieses Vorgehen erfordert eine weitergehende Kooperation der einzelnen Peers, da sie zu jedem Dokument umfangreichere Statistiken liefern müssen. Die Verwendung der jeweiligen lokalen Statistiken (z.B. der Dokumenthäufigkeit innerhalb einer Kollektion) führt allerdings immer noch nicht notwendigerweise zu vergleichbaren Werten. Mögliche Lösungen dieses Problems bestehen darin, bei der Berechnung auf (ggf. approximierte) globale Statistiken oder auf die lokalen Statistiken des anfragenden Peers zurückzugreifen. Letzterer Ansatz verspricht, beim Query Result Merging das Interessenprofil des anfragenden Benutzers besonders zu berücksichtigen. Zusätzlich können diese Werte noch durch die im Rahmen der Datenbankselektion gewonnenen Bewertungen der einzelnen Peers angepasst werden.

Die Evaluierung der verschiedenen Ansätze ist Gegenstand aktueller Forschung im Rahmen des MINERVA Projekts, um auch in diesem Bereich Aussagen zur Qualität der verschiedenen Ansätze treffen zu können.

11 Zusammenfasung und Ausblick

Dieser Artikel hat MINERVA vorgestellt, die prototypische Implementierung einer P2P-Suchmaschine. Wir haben unsere neuartige Architektur illustriert, bekannte Methoden zur Datenbankselektion aufgegriffen und sie an unser Umfeld angepasst. Unsere vorläufigen Ergebnisse zeigen die qualitativen Unterschiede der verschiedenen Methoden sowohl in Bezug auf die Kollektionsrangfolge als auch in Bezug auf die daraus resultierende Resultatsgüte. In unserem Versuchsaufbau überzeugte eine CORI-artige Methode zur Datenbankselektion und selbst eine einfach Ad-hoc-Methode schlug fortgeschrittene Modelle. Unsere Experimente zeigen außerdem, dass MINERVA zur Laufzeit kaum zusätzliche Ressourcen erfordert und dass eine geringe Anzahl befragter Peers typischerweise bereits einen Großteil der relevanten Dokumente liefert. All diese Resultate beziehen sich jedoch ausdrücklich derzeit nur auf unsere speziellen Versuchsbedingungen.

Wir bereiten derzeit Experimente auf deutlich größeren Dokumentsammlungen vor. Zusätzlich arbeiten wir an automatisierten Techniken zur Expansion der Anfragen. Die Methoden zur Datenbankselektion sollen außerdem durch Hinzunahme von Lesezeichen (Bookmarks) [5] sowie durch eine explizitere Betrachtung der Überlappung zwischen den einzelnen Kollektionen weiter verbessert werden. Idealerweise sollte eine Anfrage nur an solche Peers weitergeleitet werden, die ein Komplement zu den anderen befragten Peers darstellen (d.h., die eine geringe Überlappung untereinander aufweisen). Falls nämlich der befragte Peer die gleichen qualitativ hochwertigen Dokumente liefern kann, die der anfragende

Peer ohnehin schon selbst kennt, so ist die Weiterleitung der Anfrage an diesen Peer unnötig.

Literatur

1. Alonso G, Casati F, Kuno H (2004) Web Services – Concepts, Architectures and Applications. Springer, Berlin Heidelberg New York
2. Aberer K, Cudre-Mauroux P, Hauswirth M, Van Pelt T (2004) Gridvine: Building internet-scale semantic overlay networks. Technical report, EPFL
3. Aberer K, Hauswirth M, Punceva M, Schmidt R (2002) Improving data access in p2p systems. IEEE Internet Computing 6(1):58–67
4. Buchmann E, Böhm K (2004) How to Run Experiments with Large Peer-to-Peer Data Structures. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, USA, April 2004
5. Bender M, Michel S, Weikum G, Zimmer C (2004) Bookmark-driven routing in peer-to-peer web search. In: Callan J, Fuhr N, Nejdl W (eds) Proceedings of the SIGIR Workshop on Peer-to-Peer Information-Retrieval, pp 46–57
6. Callan J (2000) Distributed information retrieval. Advances in information retrieval, Kluwer Academic Publishers, pp 127–150
7. Cuenca-Acuna FM, Peery C, Martin RP, Nguyen TD (2002) PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. Technical Report DCS-TR-487, Rutgers University, September 2002
8. Cohen E, Fiat A, Kaplan H (2003) Associative search in peer to peer networks: Harnessing latent semantics. In: Proceedings of the IEEE INFOCOM'03 Conference, April 2003
9. Crespo A, Garcia-Molina H (2002) Routing indices for peer-to-peer systems. In: Proc. of the 28th Conference on Distributed Computing Systems, July 2002
10. Crespo A, Garcia-Molina H (2002) Semantic Overlay Networks for P2P Systems. Technical report, Stanford University, October 2002
11. Chakrabarti S (2002) Mining the Web: Discovering Knowledge from Hypertext Data. Morgan Kaufmann, San Francisco
12. Croft WB, Lafferty J (2003) Language Modeling for Information-Retrieval, vol 13. Kluwer International Series on Information-Retrieval
13. Callan JP, Lu Z, Croft WB (1995) Searching distributed collections with inference networks. In: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval. ACM Press, pp 21–28
14. Diaconis P, Graham R (1977) Spearman's footrule as a measure of disarray. Journal of the Royal Statistical Society, pp 262–268
15. Diaconis P, Graham R (1988) Group representation in probability and statistics. Institute of Mathematical Statistics
16. Fagin R (1999) Combining fuzzy information from multiple systems. J Comput Syst Sci 58(1):83–99
17. Fagin R, Kumar R, Sivakumar D (2003) Comparing top k lists. In: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, pp 28–36
18. Fagin R, Lotem A, Naor M (2001) Optimal aggregation algorithms for middleware. In: Symposium on Principles of Database Systems
19. Fuhr N (1999) A decision-theoretic approach to database selection in networked IR. ACM Transactions on Information Systems 17(3):229–249
20. Guntzer U, Balke W-T, Kiesling W (2000) Optimizing multi-feature queries for image databases. In: The VLDB Journal, pp 419–428
21. Grabs T, Böhm K, Schek H-J (2001) Powerdb-ir: information retrieval on top of a database cluster. In: Proceedings of the tenth international conference on Information and knowledge management. ACM Press, pp 411–418
22. Gravano L, Garcia-Molina H, Tomasic A (1999) Gloss: text-source discovery over the internet. ACM Trans Database Syst 24(2): 229–264
23. Kendall M, Gibbons JD (1990) Rank correlation methods. Edward Arnold, London
24. Karger D, Lehman E, Leighton T, Levine M, Lewin D, Panigrahy R (1997) Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: ACM Symposium on Theory of Computing, pp 654–663, May 1997
25. Lu J, Callan J (2003) Content-based retrieval in hybrid peer-to-peer networks. In: Proceedings of the twelfth international conference on Information and knowledge management. ACM Press, pp 199–206
26. Löser A, Siberski W, Naumann F, Nejdl W, Thaden U (2003) Semantic overlay clusters within super-peer networks. In: Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing, (DBISP2P), pp 33–47
27. Ludwig T (1993) Lastverwaltung für parallelrechner
28. Luxenburger J, Weikum G (2004) Query-log based authority analysis for web information search. In: WISE04
29. Melnik S, Garcia-Molina H, Raghavan S, Yang B (2001) Building a distributed full-text index for the web. ACM Trans Inf Syst 19(3):217–241
30. Manning CD, Schütze H (1999) Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, Massachusetts
31. Meng W, Yu CT, Liu K-L (2002) Building efficient and effective metasearch engines. ACM Computing Surveys 34(1):48–89
32. Nottelmann H, Fuhr N (2003) Evaluating different methods of estimating retrieval quality for resource selection. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. ACM Press, pp 290–297
33. Nepal S, Ramakrishna MV (1999) Query processing issues in image (multimedia) databases. In: ICDE, pp 22–29
34. Rowstron A, Druschel P (2001) Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp 329–350
35. Ratnasamy S, Francis P, Handley M, Karp R, Schenker S (2001) A scalable content-addressable network. In: Proceedings of ACM SIGCOMM 2001. ACM Press, pp 161–172
36. Reynolds P, Vahdat A (2003) Efficient peer-to-peer keyword searching. In: Proceedings of International Middleware Conference, pp 21–40, June 2003
37. Si L, Jin R, Callan J, Ogilvie P (2002) A language modeling framework for resource selection and results merging. In: Proceedings of the eleventh international conference on Information and knowledge management. ACM Press, pp 391–397
38. Stoica I, Karger D, Morris R, Kaashoek MF, Balakrishnan H (2001) Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications. ACM Press, pp 149–160
39. Suel T, Mathur C, Wu J, Zhang J, Delis A, Kharrazi M, Long X, Shanmugasunderam K (2003) Odissea: A peer-to-peer architecture for scalable web search and information retrieval. Technical report, Polytechnic Univ
40. Theobald M, Weikum G, Schenkel R (2004) Top-k query evaluation with probabilistic guarantees. In: VLDB, pp 648–659
41. Tang C, Xu Z, Dwarkadas S (2003) Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM Press, pp 175–186
42. Wang Y, Galanis L, de Witt DJ (2003) Galanx: An efficient peer-to-peer search engine system. Available at `http://www.cs.wisc.edu/~yuanwang`
43. Wu Z, Meng W, Yu CT, Li Z (2001) Towards a highly-scalable and effective metasearch engine. In: World Wide Web, pp 386–395
44. Xu J, Croft WB (1999) Cluster-based language models for distributed retrieval. In: Research and Development in Information-Retrieval, pp 254–261

45. B Yang, Garcia-Molina H (2002) Improving search in peer-to-peer networks. In: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02). IEEE Computer Society, pp 5–14

Matthias Bender studierte von 1999 bis 2003 Informatik an der Universität Saarbrücken. Nach Abschluss seiner Diplomarbeit (Data Mining Standards and their Interoperability) wechselte er im Oktober 2003 als wissenschaftlicher Mitarbeiter mit Gerhard Weikum an das Max-Planck-Institut für Informatik. Seine Forschungsgebiete sind Peer-to-Peer-Systeme, insbesondere Techniken zur effizienten Auswahl vielversprechender Peers im Rahmen von Peer-to-Peer-Websuche.

Sebastian Michel studierte von 1998 bis 2003 Informatik an der Philipps-Universität Marburg. Nach seinem Abschluss als Diplom-Informatiker (Thema der Diplomarbeit: Verteilte Anfrageverarbeitung mit Web Services) arbeitet er als wissenschaftlicher Mitarbeiter von Gerhard Weikum am Max-Planck-Institut für Informatik in Saarbrücken. Seine Forschungsgebiete sind Peer-to-Peer-Systeme, speziell Peer-to-Peer-Websuche und effiziente Algorithmen zur Top-k-Anfrageverarbeitung.

Gerhard Weikum war von 1994 bis 2003 Professor an der Universität des Saarlandes. Im Oktober 2003 wechselte er als Direktor an das Max-Planck-Institut für Informatik in Saarbrücken. Dort leitet er die Arbeitsgruppe 5 „Datenbanken und Informationssysteme". Seit 2004 ist er Sprecher der International Max Planck Research School (IMPRS). Zudem war er Vorsitzender des Programmkomitees der SIGMOD 2004 und ist seit 2004 Präsident des Board of Trustees der Very Large Database (VLDB) Endowment. Seine aktuellen Arbeitsschwerpunkte liegen in der automatischen Verwaltung von semistrukturierten Daten (z.B. in Intranets, digitalen Bibliotheken, Deep-Web-Portalen oder wissenschaftliche Datensammlungen) und der effektiven und effizienten Suche darauf. Darüber hinaus erforscht er selbstorganisierende und selbstoptimierende Peer-to-Peer-Systeme sowie ihren Nutzen, u.a. in den Bereichen Anfragebearbeitung, Websuche und Workflow Management.

Christian Zimmer begann im Wintersemester 1999 sein Informatik-Studium an der Universität des Saarlandes und beendete es 2003 mit der Diplomarbeit „COMPASS – Entwurf und Implementierung effizienter Strategien zum transparenten Crawling und Indexaufbau". Seit Oktober 2003 arbeitet er als wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Gerhard Weikum am Max-Planck-Institut für Informatik in Saarbrücken. Sein Forschungsschwerpunkt umfasst die Peer-to-Peer-Websuche.