YouSearch – Suche auf einem Netzwerk privater Webserver

Ausarbeitung

Simon Georg Pinkel

1. Einleitung

Die Reife des Internets (DNS) und die Einfachheit von WWW-protokollen (HTTP) haben in den vergangenen Jahren das Web als Netzwerk zum Austausch von Informationen attraktiv gemacht. Die Verbreitung von Personal Computern hat dazu beigetragen, Webserver also content-sharing Plattform auch im privaten Gebrauch einzusetzen. So verwenden beispielsweise über 1500 Leute bei IBM den Webserver YouServ [5].
Suche auf diesen Daten ist dabei offensichtlich erwünscht, allerdings gibt es diesbezüglich gerade auf Netzwerken privater Webserver erfahrungsgemäss besondere Probleme:

1) Daten auf privaten Webservern sind in der Regel sehr flüchtig, Links auf solche Sites werden schnell unbrauchbar
2) Suche durch Navigation erweist sich oft als unzureichend, da Daten auf privaten Webservern in der Regel schlecht angeordnet und nicht ausreichend lokal verlinkt sind
3) Erfahrungsgemäss handelt es ich bei einem Grossteil der Daten um keine HTML-dateien, die oft mit dem Rest ungenügend (oder gar nicht) verlinkt sind, wodurch ebenfalls die Effizienz von Suche durch Navigation leidet

Unzulänglichkeit von Websuchmaschinen

Im Kontext von Webservern stellt sich die Frage nach der Verwendung von Websuchmaschinen zur Datenlokalisierung. IR Systeme wie Google sind in der Regel crawl-basiert, und genau darin liegt das Problem: der schnell evolvierende Inhalt privater Webserver kollidiert mit einer zu niedrigen Crawl Frequenz der Suchmaschine. Falls ein Host zur Crawlzeit offline ist, werden seine Daten nicht indiziert, eine entsprechende Suche erzeugt ein unvollständiges Resultat (false negatives). Wenn ein Host Crawlzeit online, aber zur Zeit der Anfrage offline ist, sind seine Daten unerreichbar, obwohl sie im Suchresultat auftauchen (false positives). Im letzten Fall wurden die Informationen zwar indiziert, und der Host ist zur Queryzeit online, aber die Daten haben sich inzwischen geändert, was falsches Ranking oder im schlimmsten Fall wieder false positives oder false negatives zur Folge hat. Insgesamt sind die Resultate also veraltet und unvollständig.
Als Lösung wird im folgenden YouSearch[1] vorgestellt, eine Anwendung, die sich nahtlos in ein Webserver-system integrieren lässt und Suche auf schnell evolvierenden, flüchtigen Daten privater Webserver ermöglicht und dabei schnelle, frische und vollständige Resultate liefert. YouSearch verwendet im wesentlichen eine

Peer-to-Peer Architektur, um die Arbeit, die Aktualität und Vollständigkeit der Suchresultate zum Ziel hat, auf die verschiedenen Webserver/Peers zu verteilen.

Probleme bestehender P2P file sharing Systeme

P2P Systeme wie Kazaa/Gnutella oder Napster unterstützen, ebenso wie YouSearch, Suche und File Sharing auf einem Netzwerk entsprechender Hosts. Kazaa/Gnutella überfluten dabei das Netzwerk mit Anfragen, die mit einer Lebensdauer (ttl) versehen sind. Neben dem unnötigen Traffic hat dies insbesondere unvollständige Suchresultate zur Folge; Nichtsdestotrotz ist Kazaa sehr verbreitet, was sich auf den grossen Anteil von Musik- und Videodateien zurückführen lässt: Diese Files werden oft heruntergeladen und somit in vielen Replikationen auf dem gesamten Netzwerk verteilt. YouSearch hingegen geht von einem Corpus aus, der nur wenige Replikate aufweist, und hat trotzdem zum Ziel, vollständig und aktuell zu sein.
Napster verwendet einen zentralen Index wie bei einer Suchmaschine; Dies schränkt die Skalierbarkeit des Netzwerks deutlich ein, ausserdem hat diese Architektur eine zentrale Schwachstelle, fällt sie aus, so fällt die Suche auf dem Netzwerk aus (single point of failure). Diese Architektur ist insbesondere dann schon nicht mehr umsetzbar, wenn nicht nur Dateinamen, sondern auch Dateiinhalt wie bei YouSearch (text/html) indiziert wird.

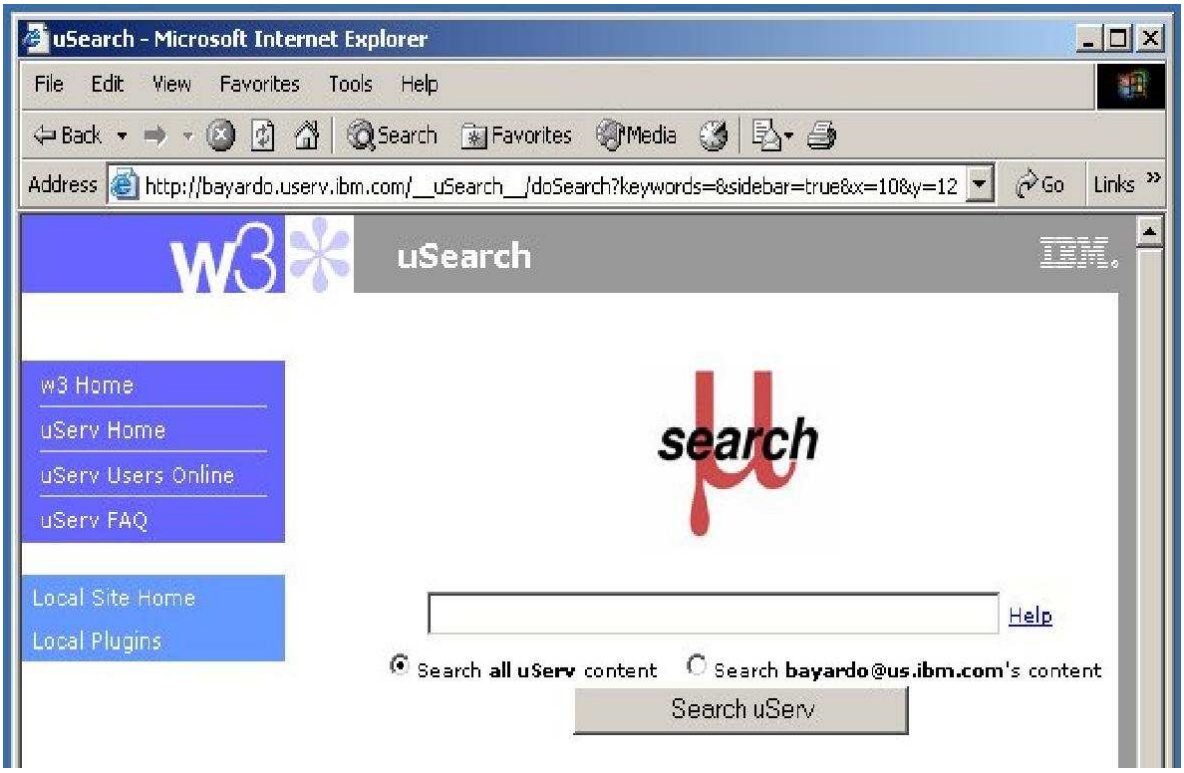

Abbildung 1: Das Webinterface von YouSearch

Insgesamt haben die meisten P2P Systeme gemein, dass sie ein proprietäres Protokoll verwenden, wodurch die indizierten Daten ohne die Installation eines entsprechendes Clients nicht erreichbar sind. YouSearch hingegen ist im Kern web-kompatibel, Benutzer können von aussen über ein Webinterface im Suchmaschinen-design (Abbildung 1) auf den angebotenen Inhalt zugreifen.

2. Implementierung

In diesem Abschnitt wird die Implementierung von YouSearch vorgestellt. In 2.1 wird eine grobe Übersicht über die Systemarchitektur gegeben, in 2.2 – 2.5 wird die Rolle der in 2.1 vorgestellten Komponenten anhand der Prozesse Indexing, Querying, Caching sowie Fehlerbehandlung erläutert.

2.1 Übersicht

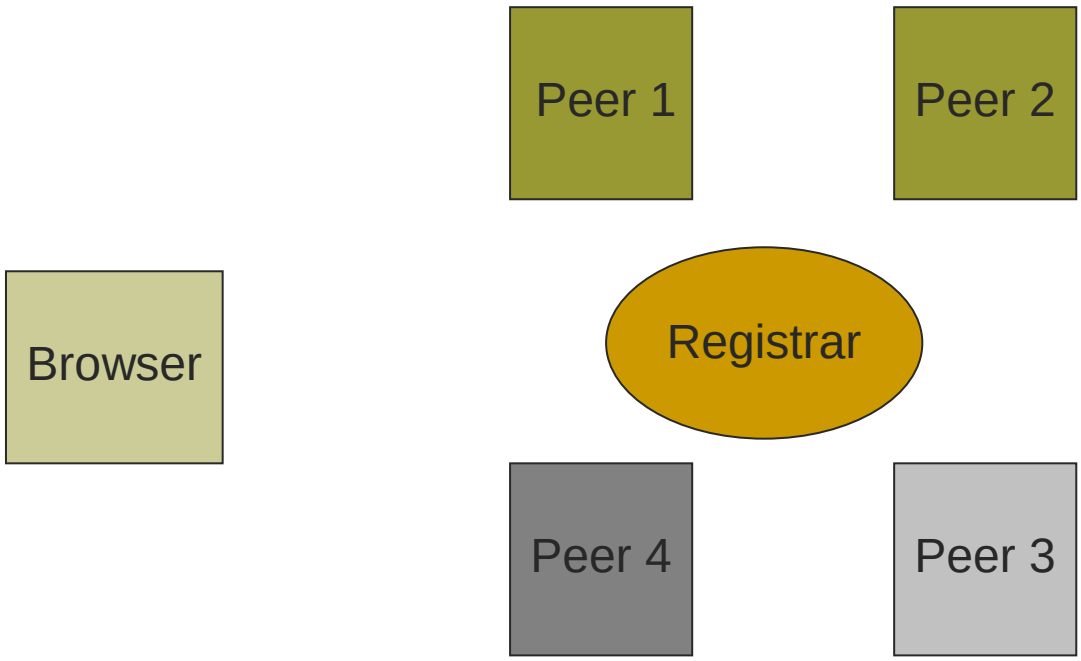

Abbildung 2: Übersicht über YouSearch (■ = Gruppe YouSearchTeam)

YouSearch setzt sich aus folgenden Komponenten zusammen (siehe Abb. 2):

1) Peers, auf denen ein mit YouSearch erweiterter Webserver läuft,
2) Browser, die auf dem freigegebenen Inhalt suchen, und
3) eine leichte, zentrale Komponente genannt „Registrar“, die den Zustand des Netzwerks speichert

Das Registrar dient als eine Art „schwarzes Brett“ für die Peers. Auf den Peers selbst können von den Benutzern der Peers überlappende Gruppen (siehe Abbildung 2) definiert werden, um einen gemeinsamen Kontext auf dem Inhalt zu unterstützen. Eine Anfrage an das Netzwerk kann auf eine oder mehrere solcher Gruppen eingeschränkt werden (z.B.: „decidability group:TheoreticalComputerScience“).

2.2 Indexing

Jeder Peer in YouSearch verwendet einen Webserver [5], um die Daten zugänglich zu machen, und einige Komponenten, wie man sich auch in Websuchmaschinen findet, um die Daten lokal zu indizieren (dabei kommt die Technologie aus [4] zum Einsatz).

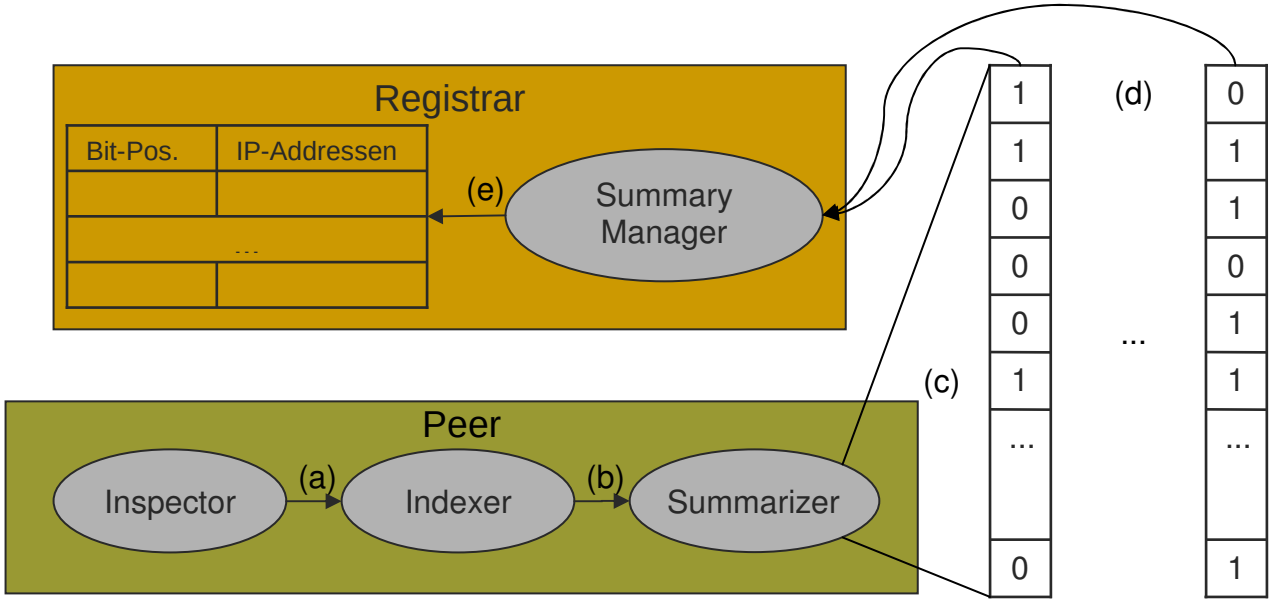

Abbildung 3: Indexing

Im folgenden wird beschrieben, wie eine Zusammenfassung dieses Indexes für alle anderen Peers zugänglich gemacht wird (siehe Abbildung 3):
Der **Inspector** untersucht freigegebene Dateien auf Änderungen; Falls nötig, informiert (a) er den **Indexer**, der den lokalen Index aktualisiert. Anschliessend schickt (b) dieser eine Liste aller auftretenden Terme *T* auf dem jeweiligen Peer an den **Summarizer**, der daraus einen *Bloom Filter* erstellt (c) (sei *Term* die Menge aller Terme):

1) Ein Bit Vektor *V* der Länge *L* wird mit 0 initialisiert.
2) Mithilfe einer Hashfunktion $H : Term \rightarrow \{1,...,L\}$ werden alle Terme *t* in *T* gehasht, indem die Stelle *V(H(t))* auf 1 gesetzt wird für alle *t* in *T*

V wird dann zum Registrar geschickt (d), dass den Bloom Filter in eine Struktur aggregiert, die Bitpositionen auf Listen von IP-Addressen der jeweiligen Hosts abbildet.
Um Hashkollisionen zu vermeiden, werden statt einer *k* unabhängige Hashfunktionen *H[i]* verwendet, und dementsprechend *k* Bit Vektoren *V[i]*. Dies bedeutet, dass genau dann ein Peer Daten zu einem Term *t* aufweist, wenn gilt:

$$\text{für alle Bitvektoren } V[i]: V[i](H[i](t)) = 1$$

Tatsächlich senkt die Verwendung eines Bit Vektors mit *k* Hashfunktionen die Anzahl an false positives geringfügig (siehe [3], Seiten 2-3), jedoch erlaubt die erstgenannte Methode die Dezentralisierung des Registrars auf verschiedene physikalische Hosts durch Verteilung der *k* Bloom Filter-zusammenfassungen auf diese Hosts; Dadurch kann man leicht Arbeitsaufwand und Traffic delegieren.

2.3 Querying

Jeder Peer bietet ein Webinterface zur Suche an; In Abbildung 4 wird veranschaulicht, wie YouSearch eine solche Anfrage verarbeitet.

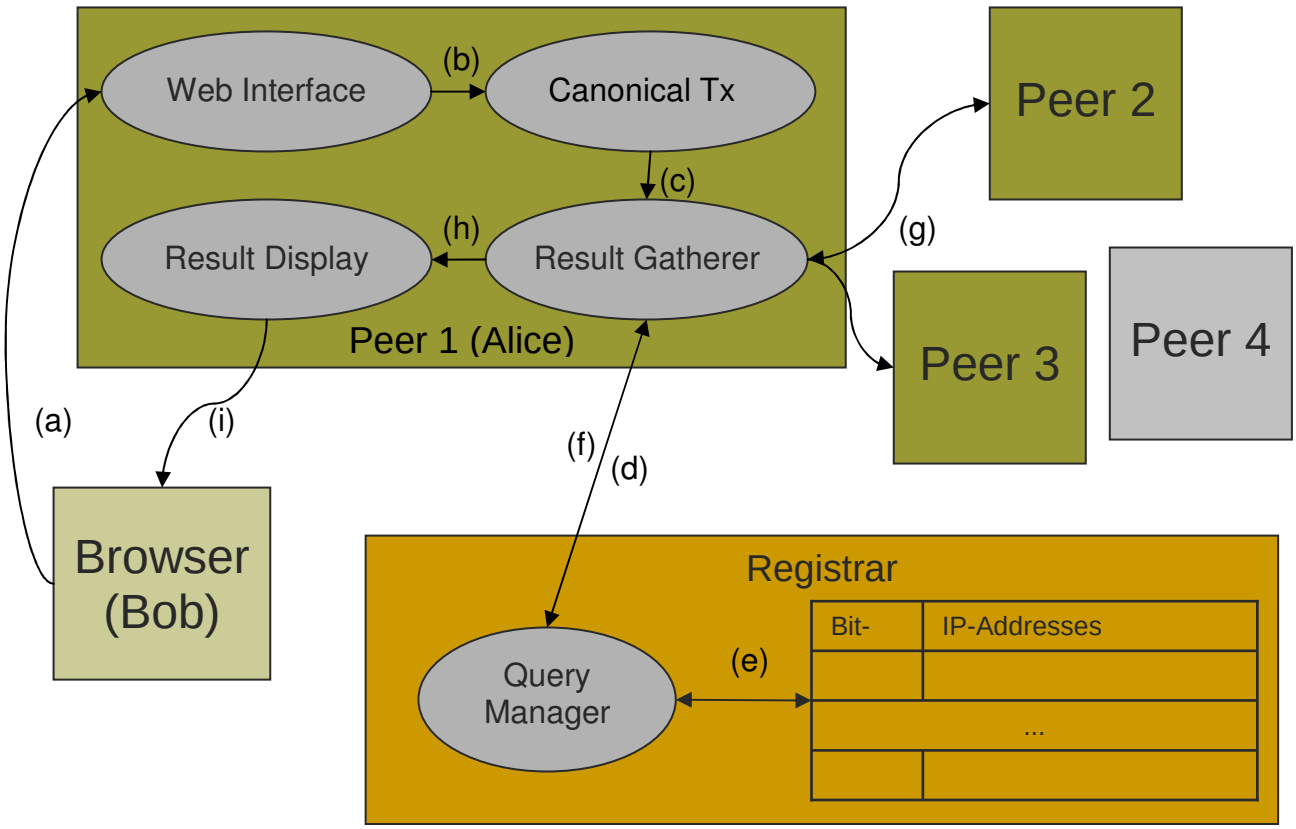

Abbildung 4: Querying (■ = Gruppe YouSearchTeam)

Angenommen, Browser Bob stellt die Anfrage „pdf group:YouSearchTeam“ an Peer Alice (a); Alice's **Web Interface** schickt die Anfrage weiter an den **Canonical Transformer** (b), der die Query in die kanonische Form {(keywords,{pdf}), (group,{YouSearchTeam})} umwandelt und an den **Result Gatherer** weitergibt (c). Dieser sendet die Query an das Registrar (d), dessen **Query Manager** aus den Bloom Filter-daten die Menge *R* aller zu dieser Query relevanten Peers berechnet (e). *R* wird an Alice zurückgeschickt (f). Falls die Query wie im Beispiel Anwendungen eines `group`- oder `site`-Operators enthielt, werden die entsprechenden Peers aus *R* gefiltert. Dies könnte bereits beim Registrar geschehen, was den ausgehenden Traffic reduzieren würde. Diese Methode würde jedoch weiteren Berechnungsaufwand für das Registrar implizieren. Ausserdem ist gerade der `group`-Operator teuer, da die entsprechende Filterung einen remote procedure call beim Gruppenserver impliziert. Hinzu kommt, dass das Registrar auf diese Weise gruppenunabhängig agieren kann, und somit von Änderungen auf der Gruppenstruktur nicht betroffen ist.
Alice kontaktiert nach der Verfeinerung von *R* alle darin noch verbleibenden Peers direkt, diese erstellen eine Antwort zur Query basierend auf ihrem lokalen Index und schicken dieses an Alice (g), dessen Result Gatherer sammelt alle Ergebnisse, diese werden vom **Result Display** visualisert (h) und an Bob gesendet (i). Bob erhält hierbei bereits Resultate, während der Result Gatherer noch sammelt, damit keine Verzögerung wahrgenommen wird.

2.4 Caching von Resultaten

YouSearch unterstützt Speicherung von Resultaten bei Peers, um diese später komplett abrufen zu können, anstatt die Indizes der relevanten Peer wieder einzeln bemühen zu müssen. Dabei wird nach einer Anfrage das erstellte Resultat automatisch in einem lokalen Puffer abgelegt, Abbildung 5 beschreibt diesen Vorgang. Alternativ können Benutzer bestimmte Urls in einem Resultat als „besonders relevant“ markieren, und dadurch diese Url zu einem persistenten Bestandteil des Resultats auf eine bestimmte Anfrage zu machen. Auf diesen Sachverhalt wird in 2.4.2 genauer eingegangen.

2.4.1 Automatisches Caching

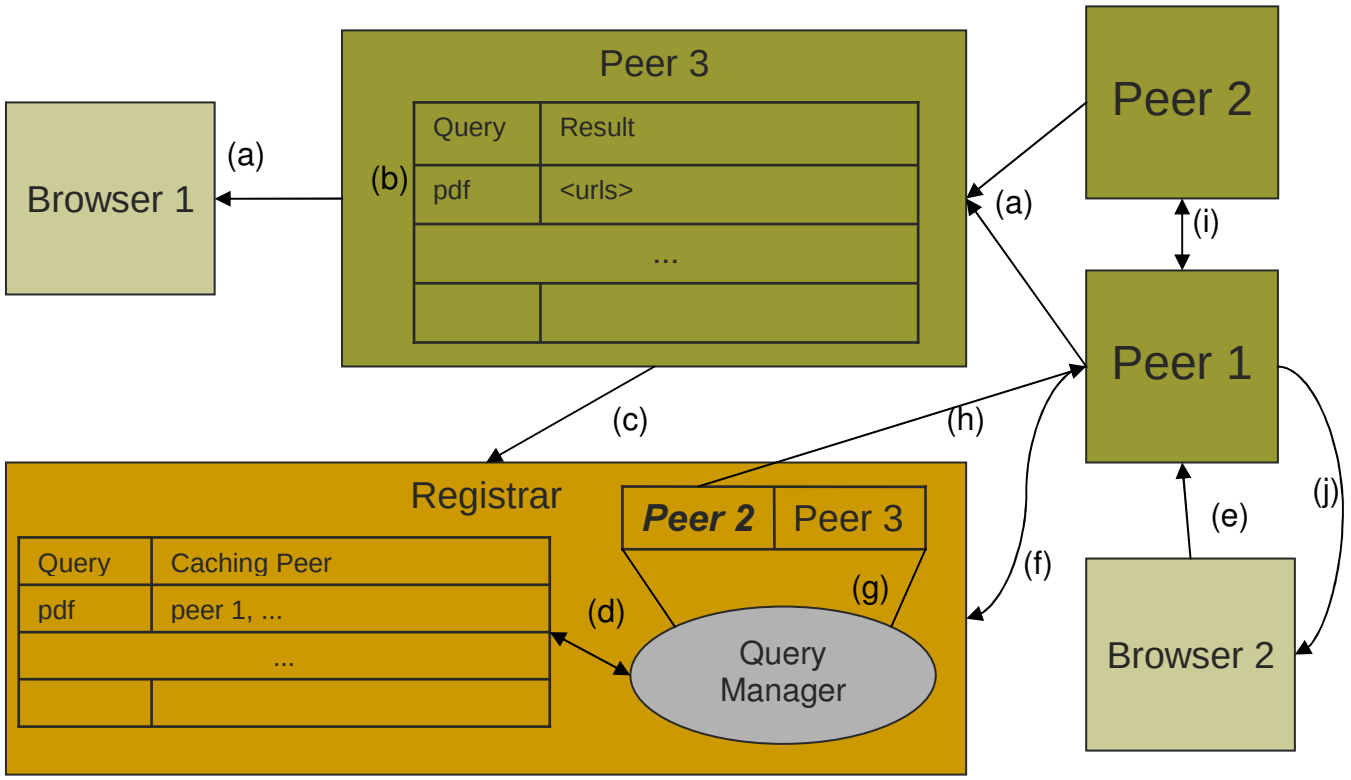

Abbildung 5: Caching

Wenn ein Peer (Peer 3) die Anfrage „pdf“ eines Browsers (Browser 1) beantwortet (a), wird das (query,resultat)-paar bei Peer 3 gespeichert (b). Ausserdem wird das Registar darüber informiert (c), dass sich Peer 3 als einen Caching Peer zu der Anfrage „pdf“ vormerkt (d). Falls nun ein Browser (Browser 2) dieselbe Anfrage an einem anderen Peer (Peer 1) stellt (e), und dieser wie in 2.3 beschrieben die Anfrage „pdf“ an das Registrar weiterleitet (f), schlägt das Registrar zunächst in seiner Cachedatenbank nach, ob Peers existieren, die das Resultat speichern (g). Handelt es sich dabei um mehrere Peers (hier Peer 2 und Peer 3), wird einer zufällig gewählt (hier Peer 2) und dessen IP-addresse an den anfragenden Peer, hier Peer 1, geschickt. Dieser muss sich anschliessend nur noch das gespeicherte Resultat von Peer 2 beschaffen (i), und es an Browser 2 weiterleiten (j).

Nach einer festen Zeitspanne (in der aktuellen Implementierung: 5 min) werden Cache-einträge bei einem Peer wieder gelöscht, und das Registrar darüber informiert, das seinerseits die jeweiligen Einträge aus seiner Cachingdatenbank entfernt.

2.4.2 benutzerseitige Empfehlung von Resultaten

Ein Benutzer kann in einer Antwortliste auf seine Query bestimmte Urls als „besonders relevant“ für seine Query markieren; Diese (query, url)-Paare werden beim Registrar gespeichert. Falls nun zu einem späteren Zeitpunkte dieselbe Query gestellt wird, wird das vorgeschlagene Resultat gesondert angezeigt. Ein Benutzer kann seine Anfrage auch nur von diesem manuell erstellten Index beantworten lassen, und dabei von ihm oder von anonymen Benutzern vorgeschlagene Urls modifizieren. Diese ermöglicht eine benutzerseitige Wartung von Resultaten.

2.5 Fehlerbehandlung[1]

Das Registrar hat sich in den in Abschnitt 2.2 bis 2.4 beschriebenen Prozessen immer komplett passiv verhalten, es diente als ein „schwarzes Brett“, auf dem Informationen über das gesamte Netzwerk notiert sind. Aktiv wird das Registrar nur zur Sicherstellung der Konsistenz der gespeicherten Daten, dass heisst im Fall der aktuellen Implementierung, festzustellen, ob sich ein Peer noch im Netzwerk befindet. Dazu kontaktiert das Registrar regelmässig die Peers (bis jetzt noch ein einfaches ping). Zusätzlich kann ein Peer Alice, der vermutet, dass ein Peer Carol offline ist, dies dem Registrar mitteilen. Peer Carol wandert dadurch an die erste Stelle in der Check-queue des Registrar. Wenn das Registrar bei der darauffolgenden Kontaktaufnahme zu Carol keine Antwort erhält, wird Carol aus dem Netzwerk entfernt.
Die Peers stellen ihre Verbindung zum Netzwerk ebenfalls aktiv sicher; Wenn ein Peer Carol lange Zeit keine Nachricht mehr vom Registrar oder einem anderen Peer erhalten hat, frägt es seinen Zustang beim Registrar ab. Lautet dieser „offline“, startet Carol eine neue Session.

3. Performance

Im folgenden wird die Performance von YouSearch diskutiert. Diese wurde in der Zeit vom neunten September 2002 bis zum neunten November 2002 gemessen. In dieser Zeit wurde YouSearch auf ungefähr 1500 Webservern im IBM Intranet verwendet.

[1] Abschnitt 2.5 basiert auf einer E-Mail von einem der Autoren von [1]

3.1 Antwortzeiten der Queries

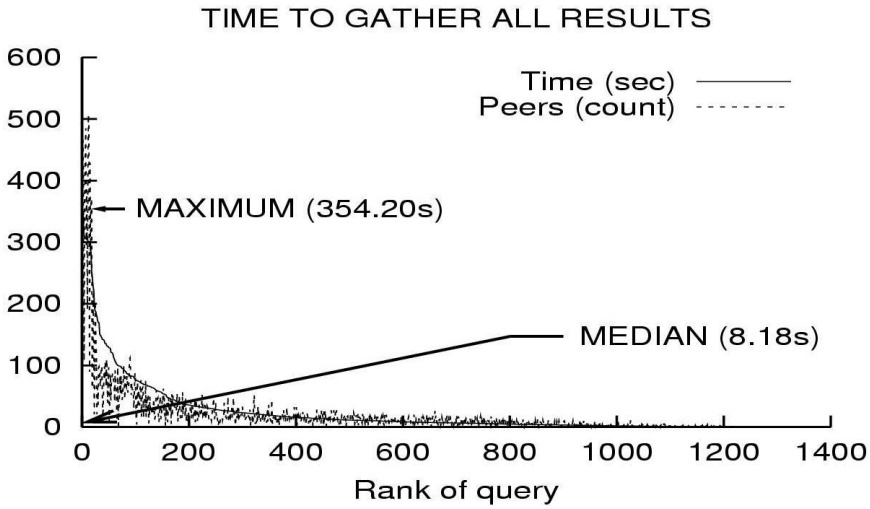

Abbildung 6: Antwortzeiten der Queries

Zwischen dem neunten September 2002 und dem neunten November 2002 wurden Queries auf dem Registrar mit ihren jeweiligen Antwortzeiten geloggt und daraus eine Menge von 1500 Queries geformt. Das Ergebnis dieser Untersuchung ist in Abbildung 6 veranschaulicht. Die gepunkteten vertikalen Linien stellen dabei die Anzahl an Peers der jeweiligen Query dar, wie zu erwarten folgen sie der Zeitlinie. Mehr als die Hälfte der Anfragen wurden somit in weniger als zehn Sekunden beantwortet, im schlimmsten Fall hat der Benutzer fast 6 Minuten auf ein vollständiges Resultat gewartet. Diese Zeiten können durch nebenläufige Anfragen auf die relevanten Peers von einem anfragenden Peer aus verbessert werden, in der aktuellen Implementierung wird dies noch sequentiell durchgeführt. Hierbei ist zu beachten, dass, wie schon in Abschnitt 2.3 erwähnt, der Benutzer bereits Resultate erhält, während diese noch gesammelt werden, und somit keine tatsächliche Verzögerung von 6 Minuten wahrnimmt.

3.2 Eigenschaften der Bloom Filter

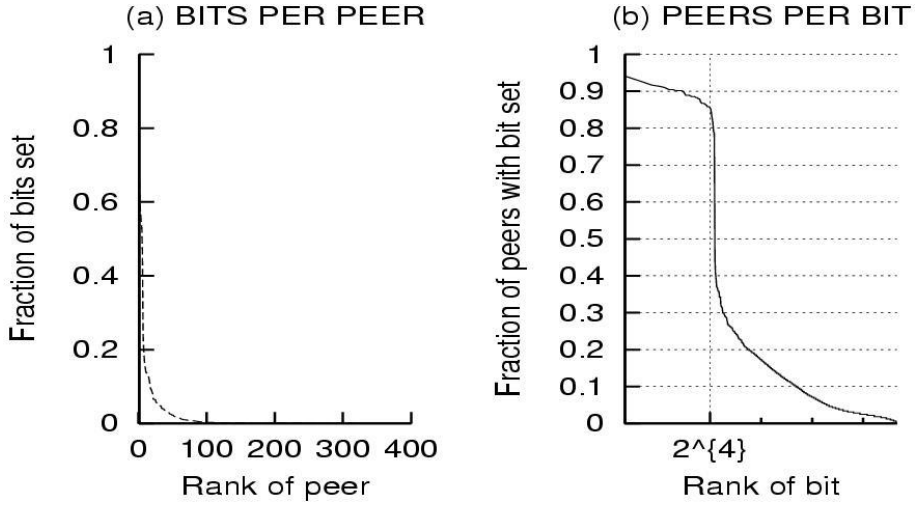

Abbildung 7: Eigenschaften der Bloom Filter

In der aktuellen Implementierung bei IBM werden *k*=3 Bloom Filter der Länge *L* von 64Kb verwendet, als Hashfunktion dient MD5, aus dessen Ergebnis 3 16b-Teile extrahiert werden.
Abbildung 7a zeigt, dass nur wenige Peers mehr als die Hälfte der Bits gesetzt haben, die so konfigurierten Bloom Filter eignen sich also für die meisten Peers als Zusammenfassung des Inhalts. Die Peers, in deren Bloom Filter mehr als 50% der Bits gesetzt sind, erführen ein hohen eingehenden Traffic und Berechnungsaufwand. Um Berechungen pro Anfrage zu verringern, könnte der Inhalt auf solchen Super-peers partioniert, und für jede Partition jeweils eine eigene Bloom Filter-zusammenfassung erstellt werden.
Abbildung 7b zufolge sind die Bits um Position 16 relativ beliebt(mehr als 80% der Peers haben diese Bits gesetzt), dies lässt durch Elimination von Stopwörten beim Indizierungsprozess reduzieren.

3.3 Auswirkungen von Caching

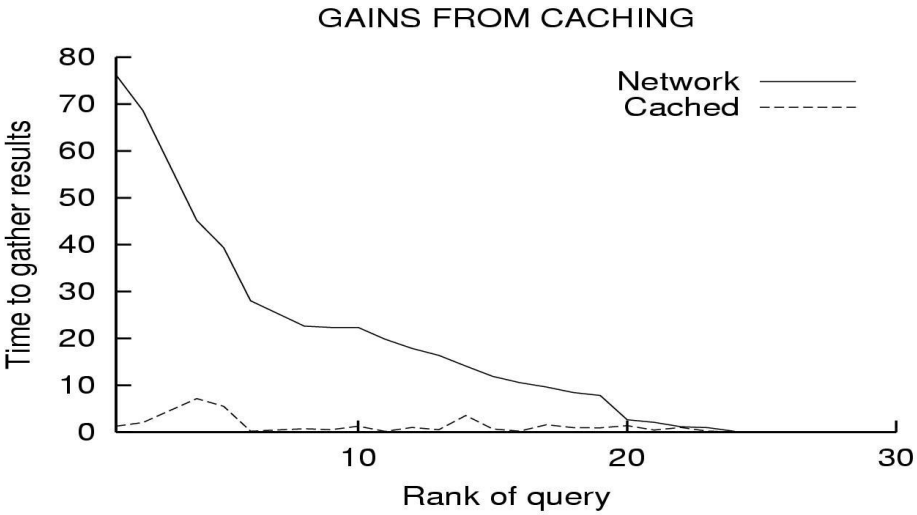

Abbildung 8: Auswirkungen des Caching

In einem Experiment wurden zunächst 25 Queries an einem Peer gestellt, anschliessend wurden dieselben 25 Queries an andere Peers im Netzwerk geschickt. Wie man in Abbildung 8 sehen kann, sind die Antwortzeiten im zweiten Fall sehr viel niedriger. Tatsächlich wurde von 1500 Queries nur 3,54% (ca. 53) aus den Peer Caches beantwortet. Dies steht 31,31% (ca.470) an wiederholt gestellten Queries gegenüber, hier ist also noch Raum zur Verbesserung. Die Entwickler begründen die geringe Nutzung der Caches mit der aktuell kurzen Lebensdauer der Einträge von 5 Minuten. Um zusätzlich Konsistenz zu gewährleisten, müssten die Cache-einträge nicht nur länger leben, sondern auch aktualisiert werden. Eine solche Wartung ist bis jetzt nur von Benutzerseite wie in 2.4.2 beschrieben möglich.

3.4 Auslastung des Registrar

Gegeben ein Netzwerk bestehend aus *n* Peers, die regelmässig *k* Bloom Filter der Grösse *L* alle *T* Sekunden an das Registrar schicken, falls sich die lokalen Daten geändert haben. Sei *f* der Anteil an Peers, deren Daten sich in einer Zeitspanne *T* geändert haben, so ist der eingehende Traffic beim Registrar im Schnitt *f*n*k*L* alle *T*

Sekunden. In der aktuellen Implementierung ist *k*=3, *L*=65563b, T=300s. Mit einem konservativ gewählten *f* von 20% und einer T1-Netzwerkverbindung (1,45Mb/s) des Registrar mit einem Overhead von 20% ist die Anzahl an maximal möglichen Peers:

$$n = (80\% * 1{,}54\text{Mb/s} * 300\text{s}) / (20\% * 3 * 65536) = 9856$$

Das Netzwerk könnte also auf eine Grösse von 9856 Peers wachsen (aktuell bei IBM: um die 1500). Bei einer T3-leitung (44,736Mb/s) wären bis zu 286310 Peers möglich. Betrachtet man die recht hohe Änderungsfrequenz von 20% alle 5 Minuten, so stellen diese Zahlen relativ lose obere Grenzen dar. Der eingehende Traffic zum Registrar liese sich ausserdem durch Versendung ausschliesslich geänderter Bits verringern, momentan werden alle Bloom Filter vollständig an das Registrar geschickt.

4. Tuning

Gegeben folgende Situation: YouSearch wird mit einer Bloom Filter-grösse *L* von 512b veröffentlicht. Nun stellt sich heraus, dass die meisten Peers einen Grossteil dieser 512 bits auf 1 setzen. Die logische Konsequenz wäre die Erhöhung von *L*. Würde nun eine neue, entsprechend angepasste Version von YouSearch erscheinen, müsste jeder Benutzer diese Version installieren, damit die Änderung global wirksam wird. Im Übergangszeitraum müssten mehrere Versionen der Bloom Filter erzeugt werden, was zu einer Verkomplizierung von Code und dem einhergehenden Performanceverlust führt.
YouSearch bietet als Lösung für diese Problem den **Tuning Manager** an; Dieser ist auf jedem Peer installiert und reagiert auf Änderungen, die von einer weiteren zentralen Komponente (neben dem Registrar), dem **Administrator**, propagiert werden. Werte wie Bloom Filter-grösse *L,* Bloom Filter-anzahl *k*, Länge von Timeouts, Aktualisierungsfrequenz *T* etc. werden bei jedem Peer in einer Parameter-zustandsdatei gespeichert, vom Tuning Manager auf Anweisung des Administrators angepasst und von allen ausführenden Komponenten (Inspector, Indexer, Summarizer etc.) ausgelesen.

5. Zusammenfassung und mögliche Erweiterungen

YouSearch verwendet eine P2P-hybrid-architektur, um Suche auf flüchtigen, schnell evolvierenden Daten zu ermöglichen. Dabei werden auf effiziente Weise aktuelle und vollständige Resultate produziert. Eine leichte zentrale Komponente, das Registrar, speichert dabei den Zustand des Netzwerks in Form von Bloom Filter-zusammenfassungen der Daten der einzelnen Peers. Durch die Verwendung mehrerer Bloom Filter lässt sich das Registrar auf mehrere physikalische Hosts verteilen. Das primär passiv agierende Registrar verbraucht vergleichsweise wenig Platz und erzeugt nur geringen Berechnungsaufwand, bei den meisten Jobs handelt es ich um Nachschlageaktionen seiner Zustandsdatenbank.
Mögliche Erweiterungen schliessen die Partionierung der Daten in öffentlich und privat ein, letztere können nur von authentifizierten Benutzern durchsucht werden.

Weitere Änderungen könnten Textauschnitte im angezeigten Suchresultat implizieren. Das automatische Caching, das in 2.4.1 vorgestellt und in 3.3 bewertet wird, könnte Suchresultate für beliebte Anfragen länger aufrechterhalten und warten. Desweiteren unterstützt YouSearch noch kein globales Ranking auf den Suchresultaten, die Resultate der einzelnen Peers werden, wie sie eintreffen, hintereinander konkateniert. In der aktuellen Implementierung wäre zumindest ein hierarchisches Ranking möglich: Ein anfragender Peer bewertet alle relevanten Peers zu einer Query, und ordnet deren Resultate dementsprechend im Gesamtergebnis[2].

Bibliographie

[1] Mayank Bawa, Roberto J. Bayardo Jr., Sridhar Rajagopalan, Eugene J. Shekita. Make it Fresh, Make it Quick – Searching a Network of Personal Webservers
http://www-db.stanford.edu/~bawa/Pub/usearch.ps

[2] B. Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors
http://www.ovmj.org/GNUnet/papers/p422-bloom.pdf

[3] Andrei Broder, Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey
http://www.eecs.harvard.edu/~michaelm/NEWWORK/postscripts/BloomFilterSurvey.pdf

[4] D. Carmel, E. Amitay, M. Hersovici, Y. Mareek, Y. Petruschka, and A.Soffer. Juru at TREC 10 – Experiments with Index Pruning
http://trec.nist.gov/pubs/trec10/papers/JuruAtTrec.df

[5] R. J. Bayardo Jr., A.Somani, D. Gruhl, and R. Agrawal. YouServ: A Web Hosting and Content Sharing Tool for the Masses.
http://www.almaden.ibm.com/cs/people/bayardo/ps/www2002.pdf

[2] Vorschlag aus einer E-Mail eines der Autoren von [1]