Proseminar
Efficient Peer-to-Peer Keyword Searching
Patrick Reynolds and Amin Vahdat

Volker Kudelko

Matrikelnummer: 2026459

Prof. Dr.-Ing. Gerhard Weikum
Database and Information Systems
Universität des Saarlandes

25.01.2005

Inhaltsverzeichnis

1 Einführung **2**

2 Vorstellen der einzelnen Techniken **3**
2.1 Partitionierung 3
2.1.1 Horizontales Partitionieren 3
2.1.2 Vertikalales Partitionieren 3
2.1.3 Wahl der geeigneten Partitionerung 4
2.2 Zentralisiert oder Verteilt 4
2.3 Bloom Filter 5
2.3.1 Einführung und Vergleich 5
2.3.2 Detaillierte Betrachtung 6
2.3.3 Überschüssige gesendete Bytes 6
2.4 Caches 7
2.5 Inkrementelle Ergebnisse 8
2.6 Virtuelle Hosts 9

3 Simulator Design **10**

4 Analyse der Techniken beim Einsatz im Simulator **11**
4.1 Skalierbarkeit und Virtuelle Hosts 11
4.2 Bloom Filter 12
4.3 Caching 12
4.4 Analyse unter meheren Rahmenbedingungen 13

5 Zusammenfassung **15**

Literaturverzeichnis **16**

Kapitel 1

Einführung

Kürzlich entwickelte Datei-Archivierungs Systeme, (z.B. Chord[4]) welche auf verteilten Peer-to-Peer Hash Tabellen aufbauten, beinhalteten keine guten, bzw. effizienten Such-Möglichkeiten. Die Autoren verteten aber die Ansicht, dass solche effizienten Such-Möglichkeiten sehr wichtig für ein solches System sind, und daher auch nicht weggelassen werden sollten.

Ziel ist es nun eine verteilte Such-Engine auf Basis einer verteilten Hash-Tabelle zu entwickeln und auf ihre Effizienz hin zu analysieren. Zu Anfangs stellt sich die Frage, was besonders wichtig für eine effizientes Suchsystem ist. Die Antwort hierauf lautet eindeutig: **End User Latenz**, denn gerade bei langsamen Internet Anbindungen (z.B. Modem) kann diese Zeit durchaus recht lange sein. Daher ist es essentiell, diese Verzögerungen möglichst gering zu halten. Eine solche Verringerung kann durch eine Reduzierung der Bytes, die pro Anfrage gesendet werden, erreicht werden. Dies kann sich sehr leicht gestalten, falls die Suchanfrage aus nur einem Schlüsselwort besteht, hierbei wird nämlich die Bearbeitung der Anfrage nur von genau einem Host durchgeführt und es ist nicht notwendig, dass noch mehere Hosts sich der Bearbeitung dieser Anfrage anschließen. Es hat sich jedoch gezeigt, dass der überwiegende Teil der Anfragen aus mehreren Schlüsselwörtern besteht[1]. Daher versuchen wir den Netzwerk Verkehr für diese Suchanfragen zu minimieren.

[1] Eine Analyse der Logfiles des IRCache Proxy mit 99.405 Suchanfragen zeigte, dass 71,5 Prozent der Anfragen aus zwei oder mehreren Schlüsselwörtern besteht.

Kapitel 2

Vorstellen der einzelnen Techniken

2.1 Partitionierung

Da unser System aus einer sehr großen Menge von Schlüsselwörtern und Dokumenten besteht, ist es nicht möglich, dass ein zentraler Rechner diese Datenmenge speichert und bei Suchanfragen bearbeitet. Es ist daher notwendig, die Datenmenge auf mehrere Rechner zu verteilen.

Dies kann durch zwei Arten geschehen: mit dem **horizontalen** und dem **vertikalen** Partitionieren.

2.1.1 Horizontales Partitionieren

Beim horizontalen Partitionieren wird ein Dokument genau einem Knoten zugeteilt, welcher eine Liste mit den in den Dokumenten enthaltenen Schlüsselwörtern speichert.

Der Vorteil hierbei ist, dass beim Einfügen oder Ändern eines Dokumentes, welches n Schlüsselwörter beinhaltet, nur genau ein Knoten kontaktiert werden muss (nämlich genau der Knoten, der für das Schlüsselwort zuständig ist). Siehe Abbildung 2.1 auf Seite 4. Diese Art der Partitionierung wird von der Suchmaschine Google[1] benutzt.

2.1.2 Vertikalales Partitionieren

Beim vertikalen Partitionieren werden ein oder mehrere Schlüsselwörter genau einem Knoten zugeordnet. Dieser speichert eine Liste mit all den Dokumenten, die diese Schlüsselwörter enthalten.

Der Vorteil hierbei ist, dass bei einer Suche, die aus k Schlüsselwörtern besteht, sichergestellt ist, dass nicht mehr als k Server kontaktiert werden müssen.

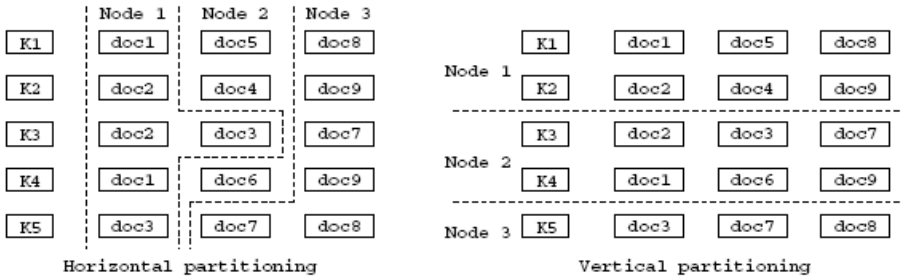

Abbildung 2.1: Vergleich zwischen Horizontalem und Vertikalem Partitionieren

(Im Gegensatz zum horizontalen Partitionieren - hier müssten alle Server kontaktiert werden). Siehe Abbildung 2.1 auf Seite 4.

2.1.3 Wahl der geeigneten Partitionerung

Beide Partitionierungen haben jeweils ihre Vor- und Nachteile. Da wir aber ein Archivierungs System entwickeln wollen, gehen wir davon aus, dass sich die meisten Dateien nur sehr selten ändern, und diejenigen, die sich ändern, tun dies sehr oft. Ebenfalls nehmen wir an, dass die Anzahl der Suchanfragen die Anzahl der Änderungen deutlich übersteigen wird.

Aus diesem Grund wählen wir die Vorzüge einer vertikal partitionierten Such Engine.

2.2 Zentralisiert oder Verteilt

Es stellt sich nun die Frage, ob das System zentralisiert oder verteilt verwaltet werden soll. Die Wahl fällt hierbei eindeutig auf ein verteiltes System, da dieses weit mehr Vorteile gegenüber einem zentralen System bieten kann.

Beispielsweise besteht nicht das Risiko eines Totalausfalls, wenn ein Rechner vom Netz geht. Wenn bei einem zentralen System der Server ausfällen würde, wäre das ganze Netzwerk davon betroffen und es könnten keine Suchanfragen mehr gestellt werden. Ebenfalls verringert sich das Risiko für Denial-of-Service Attacken und die Zensur durch inländische oder ausländische Regierungen. Desweiteren wird durch die verteilte Struktur eine höhere Replikation erreicht, wodurch eine bessere Verfügbarkeit der Daten erreicht wird. Durch eine verteilte Struktur, ist es auch möglich, dass eine Antwort auf eine Suchanfrage von einem Host erledigt wird, dessen Standort am nächsten zum suchenden Host liegt, was zu einer Verringerung der End-zu-End Latenz führt.

2.3 Bloom Filter

2.3.1 Einführung und Vergleich

Ein Bloom Filter[2,3] ist eine platzsparende Methode um eine Menge darzustellen, wobei sich diese Art der Darstellung besonders gut dafür eignet um Anfragen ob ein Element in der Menge ist oder nicht zu verarbeiten. Vergleiche dazu Abbildung 2.2.

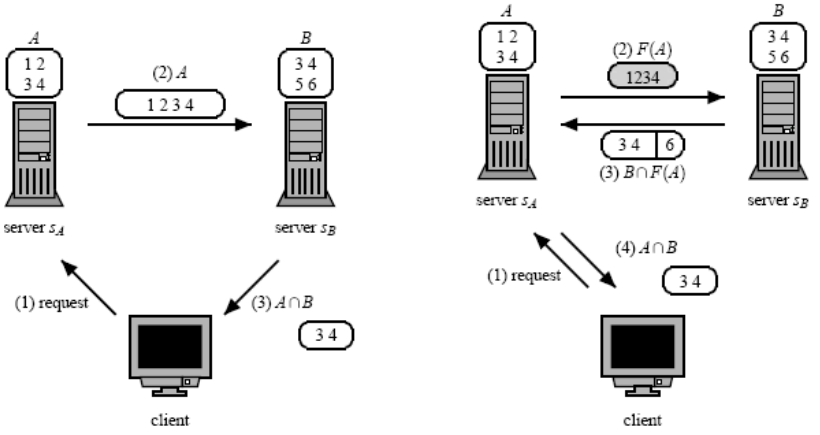

Abbildung 2.2: Vergleich einer simplen AND Anfrage ohne und mit Bloom Filter

Bei Verzicht auf den Einsatz von Bloom Filter sendet zunächst der Client seine Anfrage an Server S_A, dieser ist zuständig für das Schlüsselwort A. Er sendet nun eine Liste mit den Dokumenten, die das Schlüsselwort A enthalten, an Server S_B, dieser ist zuständig für das Schlüsselwort B. S_B bildet nun die Schnittmenge von seinen eigenen Ergebnissen zum Schlüsselwort B und den Ergebnissen, die er von S_A erhalten hat. Diese werden dann zurück zum Client gesendet.

Wird nun jedoch ein Bloom Filter eingesetzt, so sendet Server S_A anstatt einer Liste mit allen Dokumenten, die das Schlüsselwort A enthält, einen Bloom Filter $F(A)$. Durch den Einsatz eines Bloom Filters kann es nun aber auch vorkommen, dass sogenannte *False Positives* auftreten. Dies bedeutet, dass in der Menge $F(A)$ unglücklicherweise Elemente auftreten können, die nicht in A sind. Aus diesem Grund sendet Server S_B seine Intersection nicht direkt zum Client sondern zunächst wieder an Server S_A. Dieser bildet dann die Schnittmenge $A \cap (F(A) \cap B)$

Durch diesen zusätzlichen Schritt werden die False Positives eliminiert. Erst jetzt werden die Ergebnisse zum Client gesendet.

Der Vorteil des Einsatzes von Bloom Filter liegt in der Reduzierung der gesendeten überschüssigen Bytes. Dadurch dass nicht die vollständige Liste mit Dokumenten an Server S_B gesendet wird, sondern nur ein wesentlich kleinerer

Bloom Filter, ergibt sich eine deutliche Einsparung an gesendeten überschüssigen Bytes.

Der Nachteil hierbei ist, das mögliche Auftreten von False Positives, die eventuell durch zusätzliche Schritte eliminert werden müssen. Bei einer Suchanfrage, bei der k Server teilgenommen haben, bedeutet dies, dass alle k Server in Betracht kommen, um diese False Positives zu entfernen.

2.3.2 Detaillierte Betrachtung

Bei der Erstellung eines Bloom Filters wird ein Vector v bestehend aus m bits initialisert, wobei alle Bits auf 0 gestetzt sind. Nun wählt man k voneinander unabhängige Hashfunktionen mit Wertebereich 1 bis m. Möchte man nun ein Element a mit Hilfe eines Bloom Filters darstellen, so setzt man die Bits an den Positionen $h_1(a)$ bis $h_k(a)$ auf 1. Möchte man nun testen ob ein Element b in der Menge ist, so genügt es die Bits an den Postitionen $h_1(b)$ bis $h_k(b)$ zu testen. Sind alle diese Bits 0, so ist b definitiv nicht in der Menge. Andernfalls nehmen wir an, dass b in der Menge ist, obwohl diese Annahme falsch sein könnte.

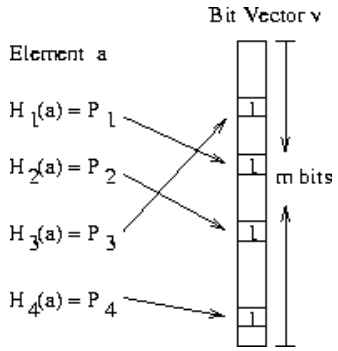

Abbildung 2.3: Aufbau eines Bloom Filters

2.3.3 Überschüssige gesendete Bytes

Situation ohne den Einsatz von Bloom Filter Betrachtet man die überschüssigen gesendeten Bytes ohne den Einsatz von Bloom Filter bei einer simplen AND Anfrage, bestehend aus den Schlüsselwörtern A und B, wobei es jeweils 10.000 Dokumente gibt, die das Schlüsselwort A bzw. B enthalten und bei Verwendung eines 128 bit großen Dokument Identifier, ergibt sich $10.000 \cdot 128\ bit = 1.280.000\ bit$.

Situation mit dem Einsatz von Bloom Filter Betrachtet man nun die Situation mit Bloom Filter ergibt sich folgendes:

Die Wahrscheinlichkeit, dass ein bestimmtes Bit auf 0 gesetzt ist, beträgt

$$p_{1bit0} = \left(1 - \frac{1}{m}\right)^{kn} \tag{2.1}$$

wobei k die Anzahl der Hashfunktionen, m die bits, also die Größe des Bloom Filters, und n die Anzahl der Elemente in der Menge darstellt.

Daraus ergibt sich die Wahrscheinlichkeit für das Auftreten eines False Positives von

$$p_{fp} = 1 - \left(1 - \left(\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{\frac{kn}{m}}\right)^{k} \tag{2.2}$$

Bildet man nun die erste Ableitung und löst nach k auf, ergibt sich die Wahrscheinlichkeit für ein False Positive bei optimaler Anzahl von Hashfunktionen von

$$p_{fp} = 0,6185^{m/n} \tag{2.3}$$

Nun ergibt sich die Anzahl der überschüssig gesendeten Bytes (excess.bits) durch:

$$excess.bits = m \cdot p_{fp} \cdot |B| \cdot j \tag{2.4}$$

Berechnet man nun die optimale Größe des Bloom Filters ergibt sich durch Bilden der ersten Ableitung und Auflösen nach m:

$$m = |A| \log_{0,6185}\left(2,081 \frac{|A|}{|B|\, j}\right) \tag{2.5}$$

Setzt man nun in die Gleichungen 2.4 und 2.5 die Daten für A,B und j ein, erhält man 106,544 als Anzahl überschüssiger gesendeter Bytes. Dies entspricht einer Kompression von etwa 12:1 gegenüber dem Senden ohne Bloom Filter.

Reihenfolge der Suchanfrage Desweiteren ist zu beachten, dass bei Suchanfragen, die aus mehr als zwei Schlüsselwörtern bestehen, es nicht egal ist, mit welchem Schlüsselwort begonnen wird. Das System beginnt stets mit dem Schlüsselwort, das in den wenigsten Dokumenten auftaucht, und sendet dann einen Bloom Filter oder die Menge selbst an den Host, der die zweitwenigsten Ergebnisse zu einem Schlüsselwort der Anfrage liefert. Dies wird solange wiederholt, bis alle Server, die die zu suchenden Schlüsselwörter enthalten, kontaktiert wurden.

2.4 Caches

Empfängt ein Server Daten, wie z.B. eine Liste mit Dokumenten, die ein bestimmtes Schlüsselwort enthalten, oder einen Bloom Filter, so kann er ihn lokal

speichern. Falls nun die gleiche Suchanfrage wieder zu bearbeiten ist, kann der Server direkt, also ohne mit anderen Servern in Kontakt zu treten, diese Anfrage bearbeiten und die Ergebnisse direkt zum Client senden.

Es stellt sich hierbei heraus, dass es effizienter ist, Bloom Filter zu speichern, da diese - wie in Gleichung 2.4 und 2.5 berechnet - wesentlich kleiner sind als die ganze Dokumentliste. So ist es möglich bei gegebenem Platz mehr Daten abzuspeichern. Ebenso wird beim Einsatz eines Caches die Trefferrate erhöht, was dazu führt dass sich das Minimum der Überschuss Bits verringert.

2.5 Inkrementelle Ergebnisse

Ein nicht zu verachtendes Problem bei sehr großen Netzwerken ist die große Anzahl an Ergebnissen, die man pro Suchanfrage erhält. Es besteht hier ein proportionaler Zusammenhang zwischen der Anzahl der Ergebisse und der Anzahl der Dokumente in Netzwerk. Dies bedeutet, dass bei derart umfassenden Netzwerken eine Suchanfrage mehr Bandbreite beansprucht als die gleiche Anfrage in einem kleineren Netzwerk.

Zwar können durch den Einsatz von Bloom Filter und Caches diese Bandbreitenkosten wesentlich reduziert werden, sie können aber nicht den linearen Anstieg dieser Kosten verhindern.

Abhilfe schafft hier der Einsatz von inkrementellen Ergebnissen[5]. Vgl dazu Abbildung 2.4 auf Seite 9. Die Menge der Dokumente, die das Schlüsselwort A beinhaltet, wird nun in meherer kleinerer Mengen aufgeteilt. Es wird nun anstatt eines Bloom Filters für A nur ein Bloom Filter für $F(A_1)$ gesendet. Im Anschluss daran bildet Server S_B die Schnittmege $F(A_1) \cap B$ und sendet diese zum Eliminieren der False Positives zurück an S_A, welcher dann die Ergebnisse zum Client sendet. Entspricht die Anzahl der Ergebnisse, die der Client bei diesem Schritt erhält, nicht der gewünschten Anzahl, so wiederholt sich dieser Vorgang mit den übrigen Teilmengen von A (A_2, A_3) bis die gewünschte Anzahl an Ergebnissen erreicht ist.

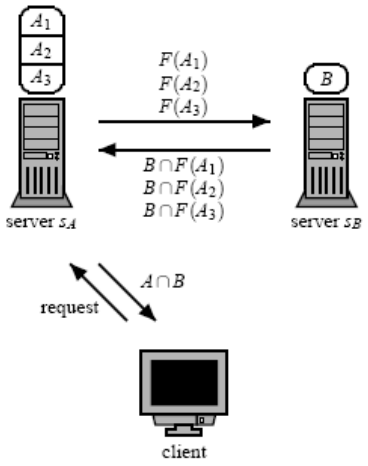

Abbildung 2.4: Einsatz inkrementeller Ergebnisse

Ein weiterer Vorteil, der durch den Einsatz inkrementeller Ergebnisse entsteht, ist dass ein Servercache nun nicht einen Bloom Filter über einer ganzen Menge speichern kann, sondern auch über einer Teilmenge. Er kann sich somit die Teilmengen heraussuchen, die am häufigsten angefragt werden und die weniger populären Teilmengen nicht speichern. Dadurch dass es jetzt möglich ist Teilmengen anstatt ganzer Menger zu speichern, reduziert sich der Speicherplatzbedarf.

Es ergibt sich jedoch auch einen Nachteil durch den Einsatz inkrementeller Ergebnisse: Es entsteht eine höhere CPU Auslastung beim Verarbeiten der Suchanfrage, da die Rechenzeit, die für Schnittmengenbildung von $B \cap F(A_i)$ notwendig ist, identisch mit $B \cap F(A)$ ist.

2.6 Virtuelle Hosts

Ein Problem von Peer-to-Peer Netzwerken ist deren Heterogenität, das heißt, es gibt in solchen Netzwerken viele veschiedenartig leistungsfähige Rechner. Es kann nun passieren, dass ein bestimmter Host im Netzwerk eine unangebrachte Menge [1] von Schlüsselwörtern zugeteilt bekommt[2].

[1] bespielsweise zu viele, so dass ein Host durch die Suchanfragen überlastet wird, oder zu wenige, so dass es ohne weiteres für ihn möglich wäre, noch zusätzliche Suchanfragen zu beabeiten

[2] Dieses Zuteilen geschiet mit Hilfe einer Hash Funktion: Jeder Host im Netzwerk erhält eine eindeutige Nummer (ID), soll nun das Schlüsselwort *key* einem Host zugeteilt werden, so wird der Hashwert dieses Schlüsselwortes berechnet und der Host dessen ID am nächsten zu diesem Hash Wert liegt, dem wird das Schlüsselwört *key* zugeteilt

Kapitel 3

Simulator Design

Es handelt sich beim hier vorgestellten Simulator um eine Java Applikation, welche 1,85 GB an Daten mit 1,17 Millionen unterschiedlichen Schlüsselwörtern in 105.953 Dokumenten beinhaltet.

Es wurden insgesamt 95.409 Suchanfragen mit 43.433 unterschiedlichen Schlüsselwörtern durchgeführt.

Der Simulator wurde in drei verschiedenen Umgebungen getestet:

- Modem
- Backbone Verbindungen
- ein auf Gnutella basierendes Netzwerk (homogen)

Die Speicherkapazität der Hosts, welche auf einer Fläche von 4000 km^2 platziert wurden, reicht von einem bis hundert MB.

Ablauf einer Suchanfrage Es werden zunächst pro Schlüsselwort höchstens M Ergebnisse betrachtet und in die Suchanfrage involviert. Nun bildet jeder Host die Schnittmenge von seinen eigenen Ergebnissen mit den Ergebnissen, die an ihn gesendet wurden, und sendet diese weiter an den folgenden Host. Die Reihenfolge hierbei lautet wie in Abschnitt 2.3.3 auf Seite 7 betrachtet. Der Einsatz eines Bloom Filters hängt von der Bloom Filter Schranke ab, welche später in Abschnitt 4.2 auf Seite 12 behandelt und analysiert wird. Bei Einsatz eines Bloom Filter ist jeder Host, der einen Bloom Filter gesendet hat, ein potentieller Kandidat die False Positives entfernen. Erhält nun der anfragende Client zu wenige Ergebnisse, so wird M solange erhöht, bis die gewünschte Anzahl erreicht ist.

Kapitel 4

Analyse der Techniken beim Einsatz im Simulator

4.1 Skalierbarkeit und Virtuelle Hosts

Unser Ziel nach wie vor ist es die Bytes, die pro Suchanfrage übertragen werden, zu reduzieren, wodurch die End-to-End Latenz verringert wird.

Durch den Gebrauch von virtuellen Hosts konzentrieren sich populäre Schlüsselwörter auf leistungsfähigeren Hosts, womit die Wahrscheinlichkeit steigt, dass ganze Suchanfragen von einem einzigen Server verarbeitet werden können. Wie in Abbildung 4.1 (links) zu sehen ist, hat dies aber nur einen geringen Effekt auf die Anzahl der Bytes, die pro Anfrage gesendet werden.

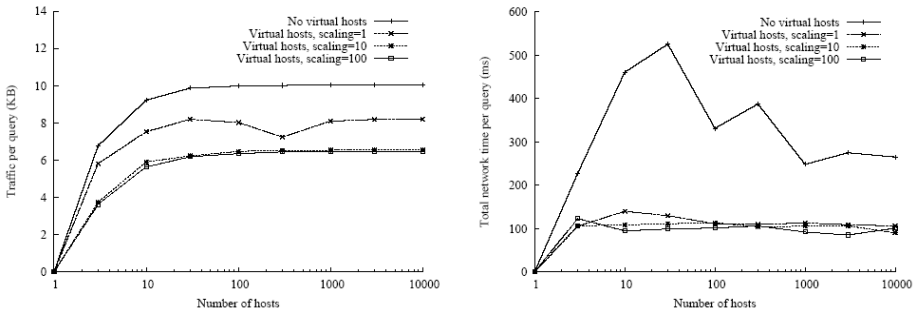

Abbildung 4.1: Einsatz von Virtuellen Hosts

Eine weitaus größere Verbesserung ergibt sich in der durchschnittlichen Zeit, die pro Anfrage benötigt wird. Durch den Einsatz virtueller Hosts verringert sich diese Zeit zu einem Fünftel, was anhand der Abbildung 4.1 (rechts) verdeutlicht wird. Dies lässt dich dadurch erklären, dass es keine Engpässe mehr gibt, da die populären Schlüsselwörter den leistungsstärkeren Hosts zugeteilt werden

und diese nicht überlastet werden. Ebenfalls wird ein besserer Lastenausgleich erzielt, so dass leistungsfähigere Hosts einen großen Anteil der Suchanfragen verarbeiten.

4.2 Bloom Filter

Bei einem permanenten Einsatz von Bloom Filter werden unnötige viele Daten übertragen, denn das Senden einer kleineren Menge ist effizienter ohne Bloom Filter, da so die Schritte zur Elimination der False Positives wegfallen. Die Lösung hierbei liegt darin, einen Bloom Filter nur zu benutzen, wenn die Zeit, die durch den Einsatz eines Blomm Filters ensteht, kleiner ist als die Zeit, die vergeht um die False Positives zu entfernen. Durch Tests mit dem Simulator wurde eine untere Schranke gefunden, ab der es sinnvoll ist, einen Bloom Filter einzusetzen. Als optimalen Wert ergab sich, dass wenn die Menge mehr als 300 Dokumente enthält, ein Bloom Filter eingesetz werden soll. Vergleiche dazu Abbildung 4.2.

Abbildung 4.2: Optimale Schranke für dein Einsatz eines Bloom Filters

4.3 Caching

Es stellt sich nun die Frage, was die wesentlichen Vorteile des Einsatzes eines Caches sind. Betrachtet man nun die gesendeten Bytes pro Anfrage mit und ohne Cache, wobei man False Positive Raten von 0,1 und 10 Prozent zulässt, ergibt sich nur ein geringes Einsparungspotential. Vergleiche dazu Abbildung 4.3 (links) auf Seite 13.

Lässt man nur geringe False Positive Raten zu, ist es notwendig, dass die False Positives entfernt werden, was zu zusätzlichen Übertragungen und somit zu einem höheren Netzwerkverkehr pro Anfrage führt. Dies wird ebenfalls in Abbildung 4.3 (links) auf Seite 13 verdeutlicht.

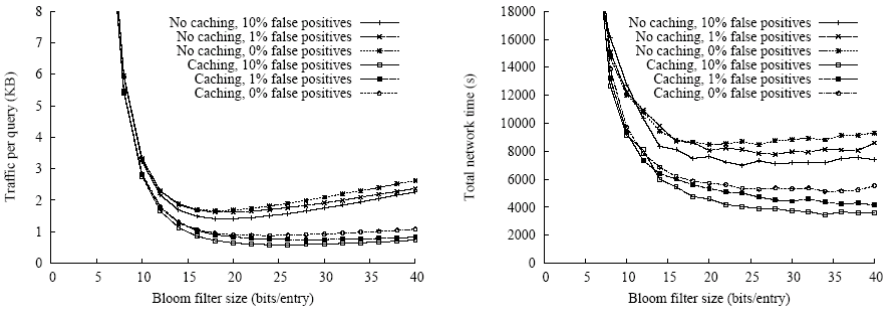

Abbildung 4.3: Vorteile beim Einsatz eines Caches

Betrachtet man nun die Zeit, die für eine Suchanfrage mit und ohne Caches benötigt wird, ergibt sich ein wesentlich größeres Einsparungspotential (etwa 45 Prozent). Vergleiche dazu Abbildung 4.3 (rechts).

4.4 Analyse unter meheren Rahmenbedingungen

Betrachtet wird nun der Einsatz aller Techniken unter drei Rahmenbedingungen:

- WAN
- heterogenes Netzwerk
- Modem

Ebenso teilen wir die End-to-End Zeit in drei Komponenten auf:

- Rechenzeit des Prozessors
- Netzwerk Übertragungszeit (= gesendete Bytes dividiert durch die Netzwerkanbindung)
- Latenz (die Zeit, die ein Elektron benötigt, um vom einem Host zum anderen zu gelangen)

Heterogenes Netzwerk In heterogenen Netzwerken zeigt sich, dass der Einsatz von virtuellen Hosts und Caches den stärksten Effekt auf die durchschnittliche Zeit, die pro Suchanfrage verstreicht, besitzt. Dies lässt sich dadurch erklären, dass Hosts mit hoher Bandbreite popoläre Schlüsselwörter beherbergen. Vergleiche dazu Abbilding 4.4 auf Seite 14.

WAN In einer WAN Umgebung lässt sich die Netzwerkübertragungszeit vernachlässigen, da alle hier Hosts über einer sehr schnelle Internetanbindung verfügen.

Durch den Einsatz von Caches reduziert sich die Rechenzeit, da so das Senden und berechnen von vielen Bloom Filtern nicht mehr notwendig ist. Ebenso wird die Rechenzeit durch das Aktivieren von virtuellen Hosts deutlich verringert, da viele Anfragen von leistungsfähigeren Hosts verarbeitet werden. Vergleiche dazu Abbilding 4.4 auf Seite 14.

Modem In einem Netzwerk, welches sich ausschließlich aus Hosts mit Modem Verbindungen zusammensetzt, spielt die Netzwerkübertragunszeit eine sehr dominate Rolle. Durch die Natur einer Modem Verbindung, welche nur sehr geringe Datenraten zur Verfügung stellt, ergibt sich eine lange Übertragunszeit.

Ebenso lässt sich beobachten, dass sich bei Einsatz von virtuellen Host kein Einsparpotential ergibt, da alle Hosts über die gleiche langsame Verbindung verfügen und somit keine bestimmten Hosts geeignet sind meherer oder populäre Anfragen zu bearbeiten. Vergleiche dazu Abbilding 4.4 auf Seite 14.

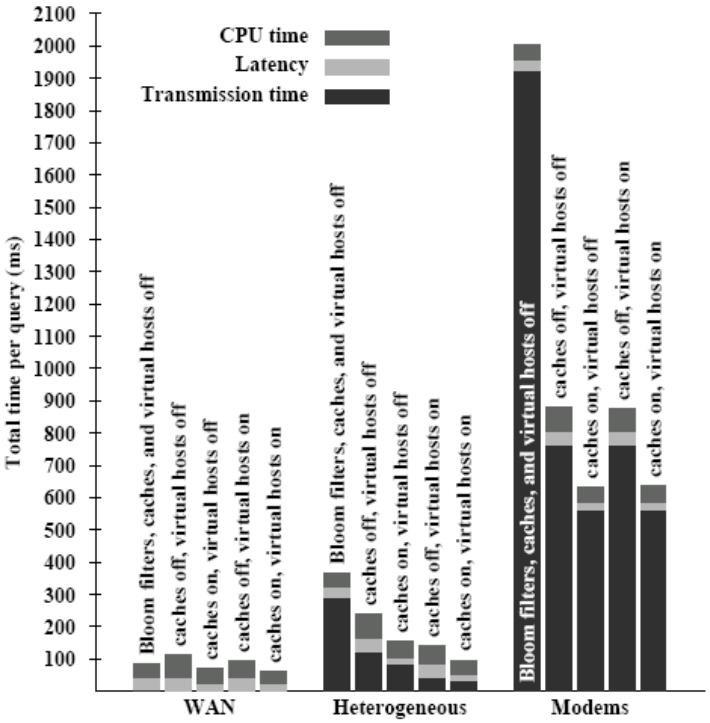

Abbildung 4.4: Ergebnisse des Einsatzes verschiedener Techniken unter unterschiedlichen Netzwerk Umgebungen mit einer Bloom Filter Schranke von 300 Elementen pro Menge

Kapitel 5

Zusammenfassung

Zusammenfassend lässt sich sagen, dass durch den gleichzeitigen Einsatz von virtuellen Hosts und Caches, sich das größte Einsparpotential in einem heterogenem Netzwerk ergibt. Es ist hiermit möglich, die durchschnittliche Antwortzeit einer Anfrage um 59 Prozent zu senken.

Vereinzelt kann der Gebrauch von virtuellen Hosts dazu führen, dass sich die durchschnittlichen übertragenen Daten pro Suchanfrage sich um 48 Prozent reduzieren. Dies lässt sich dadruch erklären, dass sich nun populäre Schlüsselwörter auf Hosts mit höherer Bandbreite konzentrieren.

Schließlich lässt sich beobachten, dass sich durch den Einsatz von Caches die durchschnittliche Latenz einer Anfrage um 47 Prozent verringert, da hierdurch ein größerer Teil der Netzwerk Übertragungen nicht mehr notwendig ist.

Literaturverzeichnis

[1] http://www.google.de

[2] http://www.cs.wisc.edu/%7Ecao/papers/summary-cache/node8.html

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, Page 422, 1970.

[4] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. *Chord: A scalable peer-to-peer lookup service for Internet applications.* 01, 2001.

[5] Jinyang Li and M. Frans Kaashoek. *On the Feasibility of Peer-to-PeerWeb Indexing and Search*, Page 6.