Stadelmann, Thomas; Blank, Daniel; Henrich, Andreas

Implementierung von IR-Modellen auf Basis spaltenorientierter Datenbanken oder invertierter Listen

Datum der Zweitveröffentlichung: 20.11.2023

Verlagsversion (Version of Record), Konferenzveröffentlichung

Persistenter Identifikator: urn:nbn:de:bvb:473-irb-918796

Erstveröffentlichung

Stadelmann, Thomas; Blank, Daniel; Henrich, Andreas (2015): „Implementierung von IR-Modellen auf Basis spaltenorientierter Datenbanken oder invertierter Listen“. In: Thomas Seidl (Hrsg.), Datenbanksysteme für Business, Technologie und Web (BTW 2015), Hauptband, Bonn: Gesellschaft für Informatik e. V., S. 299–308.

Implementierung von IR-Modellen auf Basis spaltenorientierter Datenbanken oder invertierter Listen

Thomas Stadelmann, Daniel Blank, Andreas Henrich*

Lehrstuhl für Medieninformatik
Universität Bamberg, An der Weberei 5, 96047 Bamberg
thomas.stadelmann@stud.uni-bamberg.de
{daniel.blank|andreas.henrich}@uni-bamberg.de

Abstract: Im Information Retrieval (IR) wird die Anwendung spaltenorientierter Datenbankmanagementsysteme (DBMS) diskutiert, um u.a. durch die Trennung von Datenhaltung und Suchlogik zusätzliche Flexibilität zu gewinnen. Es stellt sich die Frage, ob sich solche Systeme für den praktischen Einsatz eignen, oder ob deren Einsatz auf das Prototyping beschränkt ist. Ziel dieser Arbeit ist es daher, IR-Systeme auf Basis spaltenorientierter DBMS mit konventionellen IR-Bibliotheken auf Basis invertierter Listen bzgl. ihrer Effektivität und Effizienz unter Verwendung des weit verbreiteten Okapi BM25 Retrieval-Modells zu vergleichen. Dabei werden bisherige Arbeiten insbesondere im Hinblick auf die Anzahl und den Typ der untersuchten Anfragen sowie die durchgängige Verwendung von Kompressionsmöglichkeiten erweitert.

1 Einleitung

Für die Implementierung von IR-Systemen wurden bisher hauptsächlich auf deren Einsatzzweck speziell zugeschnittene Datenstrukturen verwendet. Vor allem invertierte Listen haben sich dabei bewährt [MB11]. In letzter Zeit wurden jedoch vermehrt alternative Ansätze für die Umsetzung von IR-Systemen präsentiert. Einen Schwerpunkt bilden dabei IR-Systeme auf Basis spaltenorientierter DBMS, sog. Column Stores. Aufgrund einer sauberen Trennung von Daten und Suchlogik lässt sich durch die Verwendung spaltenorientierter DBMS zusätzliche Flexibilität gewinnen [MSLd14]. So kann das Retrieval-Modell mittels deklarativer Sprachen (z.B. SQL) ohne genauere Kenntnis der Datenrepräsentation sehr einfach angepasst werden. Im praktischen Einsatz stellen neben der Flexibilität allerdings vor allem Effektivitäts- und Effizienzcharakteristika die wesentlichen Aspekte bei der Beurteilung eines IR-Systems dar. Diese Arbeit beleuchtet daher die Frage, inwiefern datenbankbasierte Ansätze die Leistungsfähigkeit konventioneller IR-Systeme im Hinblick auf deren Effektivität und Effizienz erreichen können.

Wir vergleichen fünf frei verfügbare Suchbibliotheken, um jeweils das Okapi BM25 Retrieval-Modell [JWR00] zu unterstützen. Dabei liegt der Fokus auf einer disjunktiven An-

*Unser Dank gilt Isabel Kahre und Sven Peter für ihre Unterstützung im Zuge dieser Arbeit.

fragebearbeitung mit BM25, bei der in einem Ergebnisdokument nicht alle Anfrageterme zwingend vorkommen müssen. Als spaltenorientiertes DBMS wird MonetDB[1] verwendet. Außerdem werden mit Lucene[2] und MG4J[3] zwei in Java geschriebene IR-Bibliotheken, sowie mit Indri[4] und Xapian[5] eine C- bzw. C++-Bibliothek untersucht. Zunächst wird mit Hilfe einer Testkollektion eine Effektivitätsbewertung der Systeme vorgenommen, bevor im Anschluss daran die Systeme im Kontext einer Wikipedia-Suche auf ihre Effizienz hin analysiert werden.

2 Verwandte Arbeiten

Die vorliegende Arbeit motiviert sich maßgeblich durch die Arbeiten von Mühleisen et al. [MSLd14]. Dort wird vorgeschlagen, Dokumentrepräsentationen in einer spaltenorientierten Datenbank zu speichern und Retrieval-Modelle wie BM25 mittels SQL zu formulieren, um IR-Funktionalitäten zu realisieren. Besondere Relevanz in Bezug auf die Anwendung dieses Vorgehens sehen die Autoren im Bereich des Prototyping. So kann durch die Verwendung der deklarativen Sprache SQL von aufwendigerem, imperativem Code für die Realisierung des jeweiligen Retrieval-Modells abgesehen werden. Auf die Anpassung spezieller Datenstrukturen und Algorithmen kann so bei der Implementierung eines neuen Retrieval-Modells verzichtet werden.

Die Machbarkeit des auf einem spaltenorientierten DBMS beruhenden Ansatzes für das IR demonstrieren Mühleisen et al. durch die Verwendung des quelloffenen DBMS MonetDB und des kommerziellen DBMS VectorWise[6]. Auf beiden Systemen wird das konjunktive BM25 Retrieval-Modell mittels SQL-Statements realisiert. Zur Optimierung der konjunktiven BM25 Anfragebearbeitung wird die Vorberechnung der Termfrequenzen für die Terme der einzelnen Dokumente vorgeschlagen, da diese Termfrequenzen statisch sind. Eine weitere Möglichkeit zur Optimierung sehen Mühleisen et al. in der Sortierung der Term-IDs in der Term-Tabelle, um somit die binäre Suche auf den Term-IDs zu ermöglichen. Mühleisen et al. vergleichen die beiden relationalen spaltenorientierten DBMS MonetDB und VectorWise mit den drei quelloffenen Suchmaschinen Lucene, Indri und Terrier. Als Dokumentenkollektion werden ca. 45 Millionen Dokumente einer Web-Kollektion verwendet. Es werden stets 50 Anfragen evaluiert. Dabei wird jeweils zwischen dem ersten Durchlauf (Cold Run) und dem nachfolgenden Durchlauf (Warm Run) unterschieden. Mühleisen et al. kommen in ihren Messungen zu dem Ergebnis, dass die auf spaltenorientierten Datenbanken realisierten IR-Systeme sowohl in der Effektivität als auch in der Effizienz auf einem vergleichbaren Niveau wie konventionelle Ansätze liegen.

Bereits zuvor wurde untersucht, wie sich Retrieval-Modelle mit relationalen DBMS umsetzen lassen (z.B. [GFHR97]). Außerdem existieren Ansätze zur Volltextsuche in vielen

[1] vgl. MonetDB: *https://www.monetdb.org/Home* (letzter Abruf: 12.9.2014)
[2] vgl. Lucene: *http://lucene.apache.org/core/* (letzter Abruf: 12.9.2014)
[3] vgl. MG4J: *http://mg4j.di.unimi.it/* (letzter Abruf: 12.9.2014)
[4] vgl. Indri: *http://www.lemurproject.org/indri.php* (letzter Abruf: 12.9.2014)
[5] vgl. Xapian: *http://xapian.org/* (letzter Abruf: 12.9.2014)
[6] vgl. VectorWise: *http://www.actian.com/* (letzter Abruf: 12.9.2014)

gängigen relationalen DBMS, die meist das Boole'sche Retrieval unterstützen. IR-Systeme auf Basis invertierter Listen werden z.B. in [MB11] verglichen. Ein Alleinstellungsmerkmal der Arbeit von Mühleisen et al. ist jedoch die Verwendung spaltenorientierter DBMS.

Zur genannten Arbeit von Mühleisen et al. unterscheidet sich unser Vergleich in folgenden wichtigen Punkten: Zunächst betrachten wir im Gegensatz zu Mühleisen et al. nicht nur 50 Anfragen bei der Effizienzevaluation, sondern über 15000, wodurch ein wesentlich realistischeres Bild gewährleistet wird. Zum Zweiten betrachten wir über alle Verfahren hinweg den Effekt der komprimierten Indexablage und schließlich legen wir den Schwerpunkt auf ein disjunktives Anfragemodell, bei dem auch Dokumente für eine Anfrage relevant sein können, die nicht alle Anfrageterme enthalten. Eine solche Anfragebearbeitung ist insbesondere für spezielle Anfragen wichtig, bei denen in der Regel nur wenige relevante Dokumente existieren. Die Betrachtung disjunktiver Anfragen erscheint auch deshalb wichtig, weil eine konjunktive Anfrageverarbeitung ein deutlich höheres Optimierungspotential für die Algorithmen bietet. Ferner analysieren wir mit Xapian und MG4J mit dessen zwei Indexlayouts IR-Bibliotheken, die von Mühleisen et al. nicht betrachtet werden.

3 Beschreibungen der eingesetzten Techniken und Suchsysteme

Ebenso wie Mühleisen et al. [MSLd14] unterstellen wir die Verwendung des Okapi BM25 Retrieval-Modells. Dieses lässt sich anhand von Formel (1) beschreiben [JWR00, Roe13]:

$$Sim(Q,D) = \sum_{i=1}^{t} \underbrace{\ln \frac{N - n_i + 0{,}5}{n_i + 0{,}5}}_{\text{IDF-Teil}} \cdot \underbrace{\frac{(k_1+1)tf_{di}}{k_1((1-b)+b\frac{dl}{avdl})+tf_{di}}}_{\text{DTF-Teil}} \cdot \underbrace{\frac{(k_3+1)tf_{qi}}{k_3+tf_{qi}}}_{\text{QTF-Teil}} \tag{1}$$

Die Ähnlichkeit zwischen Anfragerepräsentation Q und Dokumentrepräsentation D wird durch Summenbildung über alle t Anfrageterme ermittelt. Einzelne Summanden setzen sich aus drei Teilen zusammen: dem IDF-Teil (inverse Dokumentfrequenz), dem DTF-Teil (Dokumenttermfrequenz) sowie dem QTF-Teil (Anfragetermfrequenz).

Der IDF-Teil modifiziert geringfügig die inverse Dokumentfrequenz $\frac{N}{n_i}$, bei der N für die Gesamtzahl der Dokumente im Korpus steht und n_i die Dokumentfrequenz (*df*) von Term i erfasst. Durch die Verwendung des Logarithmus können negative Werte resultieren. Die von uns evaluierten IR-Systeme gehen auf unterschiedliche Art damit um.

Der DTF-Teil wird maßgeblich von der Termfrequenz tf eines Terms i in einem Dokument d beeinflusst; dl bezeichnet die Länge eines Dokuments (entspricht i.d.R. der Anzahl der Terme), $avdl$ die durchschnittliche Dokumentlänge in der Kollektion. Die beiden Parameter b und k_1 können i.d.R. vom Anwender der IR-Bibliothek konfiguriert werden. Standardwerte sind entsprechend als Vorschläge vorgegeben. Auch hierbei unterscheiden sich die verschiedenen Systeme.

	Version	b	k_1	k_3	MAP	P@10	P@20
Indri	5.6	0,75	1,2	7	0,1904	0,3830	0,3184
Lucene	4.7.2	0,75	1,2	0	0,1837	0,3726	0,3198
MG4J	5.2.1	0,75*	1,2	0	0,1904	0,3802	0,3156
MonetDB	Jan2014-SP2	0,75	1,2	0	0,1904	0,3783	0,3142
Xapian	1.2.17	0,5	1	1	0,1897	0,3783	0,3170

Tabelle 1: Links: Okapi BM25 Parameter der evaluierten Suchsysteme; * zeigt an, dass $b = 0{,}5$ verändert und an den Wert von Lucene angeglichen wurde. Rechts: MAP, P@10 und P@20 bei disjunktivem BM25 auf der Ohsumed-Kollektion.

	docid	*name*	*dl*
docs:	1	Doc1	234
	..	..	..

	termid	*docid*	*tf*
terms:	1	1	2
	..	..	..

	termid	*term*	*df*
dict:	1	hamburg	23
	..	..	..

Abbildung 1: Schema des datenbankbasierten IR-Systems

Der QTF-Teil betrachtet die Vorkommenshäufigkeit eines Anfrageterms tf_{qi}. Da bei vielen Systemen initial $k_3 = 0$ gilt, ergibt sich für den QTF-Teil in diesen Fällen ein Wert von 1, sodass auf eine Anfragetermgewichtung bei Berechnung der Ähnlichkeit verzichtet wird.

Im Folgenden beschreiben wir kurz die wesentlichen Charakteristika und Unterschiede der analysierten IR-Systeme. Diese unterscheiden sich u.a. bei der BM25 Parametrisierung sowie bei den zugrunde liegenden Datenstrukturen. Die voreingestellten BM25-Parameter der Programme sowie die eingesetzte Programmversion fasst Tabelle 1 zusammen.

Datenbankbasierte Lösungen mit MonetDB
Das verwendete Datenbankschema zur Umsetzung des BM25 Retrieval-Modells ist in Abb. 1 dargestellt. Dabei handelt es sich um ein gemäß den verbalen Erläuterungen von [MSLd14] optimiertes Modell mit vorberechneten Termfrequenzen (*tf*). Der SQL-Code aus [MSLd14] für konjunktives BM25 wird von uns übernommen und für das disjunktive Modell entsprechend angepasst (vgl. Abb. 2). Wir verwenden die in Tab. 1 angegebenen Parameter. Außerdem operiert der SQL-Code bei Mühleisen et al. schon auf *termid*-Ebene und nicht, den Anfragen entsprechend, auf *term*-Ebene. Diese Auflösung muss zusätzlich durchgeführt werden. Wir beschränken uns hier auf zwei Varianten. Variante a) legt für jede Abfrage jeweils pro *term* einen SQL-Parameter fest. Dieser wird vor dem eigentlichen SQL-Statement durch ein zusätzliches SQL-Statement mit der *termid* befüllt und in der Abfrage verwendet. Variante b) lädt die komplette *term-termid* Zuordnung (vgl. die rechte Tabelle in Abb. 1) a priori in den Hauptspeicher.

Lösungen auf Basis invertierter Listen
Lucene ist eine häufig verwendete IR-Bibliothek, die seit Version 4 das BM25 Modell unterstützt. In der von uns verwendeten Version 4.7.2 sind die Parameter des BM25 Modells wie folgt gesetzt: $b = 0{,}75$, $k_1 = 1{,}2$ und $k_3 = 0$. Lucene verzichtet demnach auf den QTF-Teil des BM25 Retrieval-Modells und damit auf die Berücksichtigung der Anfragetermfrequenzen. Zudem vermeidet Lucene negative IDF-Teile, indem $ln(\frac{N-n_i+0{,}5}{n_i+0{,}5}+1)$ verwendet wird. Eine weitere Besonderheit ist die Speicherung der Dokumentlänge mit verminderter Präzision. Die Dokumentlänge wird in eine 1 Byte große Gleitkommazahl mit 3 Bit Mantisse transformiert.

```
WITH subscores AS(
  SELECT terms.docid, terms.termid,
          (log((@N-df+0.5)/(df+0.5))*((tf*(1.2+1)/(tf+1.2*(1-0.75+0.75*(dl/@avdl)))))) AS subscore
  FROM terms
  JOIN docs ON terms.docid=docs.docid
  JOIN dict ON terms.termid=dict.termid
  WHERE terms.termid IN (/*list of term IDs*/)
)
SELECT name, score FROM(
  SELECT docid, sum(subscore) AS score
  FROM subscores GROUP BY docid
) AS scores
JOIN docs ON scores.docid=docs.docid ORDER BY score DESC LIMIT 1000;
```

Abbildung 2: SQL-Code für die Bearbeitung disjunktiver Anfragen mittels Variante b).

MG4J verwenden wir mit den gleichen BM25 Parametern wie Lucene. Diese Anpassung erfolgt, um bei der Effektivitätsevaluierung den Einfluss der reduzierten Genauigkeit bei Speicherung der Dokumentlänge in Verbindung mit dem unterschiedlichen Umgang mit negativen IDF-Teilen quantifizieren zu können. Resultiert ein IDF-Teil mit einem Wert kleiner als der Schwellwert 10^{-6}, wird der Schwellwert in den weiteren Berechnungen verwendet. Bei MG4J wird zudem zwischen dem ab Version 5 neu eingeführten Quasi-Succinct-Indexformat [Vig13] und dem traditionellen Format unterschieden.

Indri ist ein IR-System, das von der University of Massachusetts entwickelt wird und häufig im Zusammenhang mit Sprachmodellen Anwendung findet. Es unterstützt jedoch auch eine Anfragebearbeitung mit BM25. Die beiden Parameter b und k_1 bleiben gegenüber Lucene unverändert. Allerdings unterscheidet sich Indri bei Auslieferung mit einem $k_3 = 7$ von den anderen Systemen, wobei die Anfrageterme in den in Abschnitt 4 bei der Evaluierung betrachteten Fällen in der Regel ohnehin jeweils nur 1-mal vorkommen, sodass die QTF-Komponente ohne Einfluss bleibt.

Xapian ist hinsichtlich der BM25 Konfiguration in mehrerlei Hinsicht speziell – zum einen bei der Parameterfestlegung mit $k_1 = 1$, $k_3 = 1$ bzw. $b = 0{,}5$ und zum anderen im IDF-Teil. Sofern $\frac{N-n_i+0{,}5}{n_i+0{,}5} < 2$, wird $ln(\frac{N-n_i+0{,}5}{n_i+0{,}5} \cdot 0{,}5 + 1)$ verwendet.

4 Evaluierung

In unseren Experimenten verwenden wir die in Abschnitt 3 genannten Systeme. Das kommerzielle VectorWise wird nicht evaluiert. Auf eine Evaluation von Terrier verzichten wir ebenso, weil dieses System in [MSLd14] schlechtere Effizienzwerte als bspw. Lucene und Indri aufweist. Wir evaluieren aus diversen Gründen vorwiegend das disjunktive BM25-Modell, u.a. aufgrund der verwendeten Testkollektion, die sonst viele Anfragen mit leerer Ergebnismenge liefern würde, sowie der Eingeschränktheit der Konsolenanwendung von Indri auf diese Art der Anfragebearbeitung. Ferner ist eine disjunktive Anfragebearbeitung insbesondere für spezielle Anfragen mit wenigen relevanten Dokumenten wichtig und in der Bearbeitung in der Regel aufwändiger als die konjunktive Verarbeitung.

Messungen werden so nah wie möglich an der jeweiligen IR-Bibliothek durchgeführt, um mögliche Störungen durch weitere Softwareschichten zu vermeiden. Dies bedeutet, dass für die Java-Bibliotheken jeweils ein Java-Programm geschrieben wurde. Die C-bzw. C++-Bibliotheken werden über deren mitgelieferte Kommandozeilenprogramme direkt mittels Shell-Skript ausgeführt. Datenbanken werden mittels JDBC aus Java angesprochen.

Als Testsystem wird ein Rechner mit Intel Core-i7 860 Vierkern-CPU, 4GB Hauptspeicher und einer 250GB HDD mit 7200 rpm verwendet. Die Festplatte wurde in eine ext4-Partition und eine btrfs-Partition mit aktivierter zlib Kompression aufgeteilt. Dadurch können Messungen sowohl mit als auch ohne Kompression durchgeführt werden. Anders als in [MSLd14] beschränken wir uns bei der Kompression nicht nur auf die spaltenorientierten Datenbanken. Auch die konventionellen IR-Bibliotheken werden mit und ohne Kompression getestet. Als Betriebssystem wird Ubuntu Server 14.04 LTS verwendet. Neben dem OpenJDK wurden auf dem System lediglich die für das Kompilieren der Kommandozeilenanwendungen benötigten Pakete installiert.

Effektivitätsevaluierung auf Basis der Ohsumed-Kollektion
Für die Effektivitätsevaluierung wird die Ohsumed-Kollektion[7] verwendet. Sie besteht aus einer Untermenge der bibliografischen Angaben zu medizinischer Fachliteratur, die in der MEDLINE-Datenbank des US-amerikanischen *National Center for Biotechnology Information* vorgehalten werden. Diese besteht aus 348566 Referenzen aus 270 medizinischen Fachzeitschriften der Jahre 1987 bis 1991. Jede Referenz beinhaltet eine textuelle Kurzbeschreibung, die wir indexieren. Des Weiteren wurden von Ärzten 106 Anfragen an die Testkollektion formuliert. Von einer anderen Ärztegruppe wurden jeweils Dokumente als definitiv und möglicherweise relevant auf eine Anfrage eingestuft. Da die von uns verwendeten Effektivitätsmaße eine binäre Definition von Relevanz erwarten, werden auch die möglicherweise relevanten Dokumente als relevant eingestuft.

Zur weiteren Verarbeitung werden die Kurzbeschreibungen und Anfragen mittels der in der Programmbibliothek von Lucene enthaltenen Werkzeuge vorverarbeitet, sodass alle verwendeten Bibliotheken diesbezüglich einheitlich arbeiten. Die Vorverarbeitung besteht aus Tokenisierung, Entfernung von Stoppwörtern und einem Porter Stemming. Die Tokenisierung kann bei den Kommandozeilensystemen nicht beeinflusst werden, sodass wir diese den jeweiligen Systemen überlassen. Beim datenbankgetriebenen Ansatz kommt die Default-Tokenisierung von Lucene zum Einsatz.

Es werden stets für jedes System die jeweils ersten 1000 Ergebnisse zu einer Anfrage angefordert. Tabelle 1 in Abschnitt 3 zeigt die Effektivitätsmaße Mean Average Precision (MAP), P@10 und P@20. Die insgesamt sehr geringen Abweichungen zwischen den einzelnen Suchtechnologien resultieren vor allem aus einer unterschiedlichen Parametrisierung (vgl. Tabelle 1) und einer unterschiedlichen Umsetzung der IDF-Komponente des BM25 Modells. Zudem trägt bei Lucene die bereits erwähnte verringerte Präzision bei Speicherung der Dokumentlänge zu Effektivitätsverlusten – gemessen mit MAP und P@10 – bei. Interessanterweise werden diese zumindest im Falle der Ohsumed-Kollektion bei P@20 kompensiert.

[7] *http://ir.ohsu.edu/ohsumed/ohsumed.html* (letzter Abruf: 15.9.2014)

	Indri	Lucene	MG4J.S	MG4J.I	MonetDB	Xapian
Indexierungszeit	4:31 Std	6:06 Std	5:01 Std	4:56 Std	7:07 Std	5:13 Std
Indexgröße	4,2 GB	2,8 GB	0,9 GB	1,1 GB	2,9 GB	7,8 GB

Tabelle 2: Indexierungszeiten und unkomprimierte Indexgrößen in Verbindung mit der Wikipedia-Kollektion. Komprimierte Größen lassen sich im btrfs-Dateisystem nicht exakt bestimmen.

Effizienzevaluierung auf Basis der Wikipedia

Da zur Messung der Effizienz keine Relevanzurteile benötigt werden, haben wir uns hier für einen größeren Korpus mit mehr Anfragen entschieden. Die Anfragen stammen aus dem bekannten AOL Query Log[8]. Indexiert wird die englische Wikipedia. Da die Anfragen aus einem Query Log des Jahres 2006 extrahiert werden, wählen wir einen Wikipedia Dump[9] aus dem November 2006. Tabellen und Bildreferenzen werden aus den Dokumenten entfernt. Dadurch entstehende zu kurze Dokumente ohne jeglichen Informationsgehalt werden mittels Schwellenwert entfernt. Es werden nur die Dokumente verwendet, die mindestens acht Wörter enthalten. Als Resultat erhalten wir 1490557 Dokumente. Von den Anfragen des AOL Query Logs werden nur diejenigen verwendet, bei denen unmittelbar nach der Suche auf einen Link der englischen Wikipedia geklickt wurde. Somit passen die Anfragen sowohl inhaltlich als auch zeitlich zur Dokumentkollektion. Ferner wurden die Suchanfragen durch einfache Heuristiken weiter bereinigt (mindestens drei Zeichen, mindestens einen Vokal, keine Internetadresse). Weiter wurden Anfragen vernachlässigt, die Wikipedia als Begriff enthalten. Insgesamt konnten so 15284 Anfragen extrahiert werden.

Während bei der Ohsumed-Kollektion die Entfernung von Stoppwörtern und eine Stammformreduktion von Vorteil ist, um die Effektivität zu steigern, werden solche Verfahren bei Webkollektionen wie unserem Wikipedia Dump kontrovers diskutiert [BR11]. Da Websuchmaschinen durchaus andere Ergebnisse auf unterschiedliche Anfragen des gleichen Wortstammes liefern und das *to-be-or-not-to-be*-Problem [BR11, *p*. 226] gegen eine Stoppwortentfernung spricht, verzichten wir hier auf diese Vorverarbeitungsmaßnahmen. Für die Tokenisierung verwenden wir die gleichen Mechanismen wie zuvor. Tabelle 2 gibt Aufschluss über die Indexierungszeiten und Indexgrößen.

Wir verwenden zunächst das disjunktive BM25 Retrieval-Modell, das bei nahezu allen Anfragen ein Ergebnis liefert (bei 99,7% bzw. 97,5% mit bzw. ohne Verwendung einer Stammformreduktion und Stoppworteliminierung). Bei konjunktivem Modell liefern nur noch 46,2% der Anfragen ohne Stammformreduktion und Stoppworteliminierung ein Ergebnis (mit Stammformreduktion und Stoppworteliminierung sind dies 92,0%).

Bei den Anfragedurchläufen findet wie bei Mühleisen et al. nach dem Cold Run unmittelbar folgend ein Warm Run statt. Vor jedem Cold Run werden Dateisystemcaches geleert. Die Anfragen werden immer in der gleichen Reihenfolge an das jeweilige System gestellt.

Statistiken über die Anfragezeiten aller 15284 Anfragen sind in Tabelle 3 dargestellt. Abbildung 3 (links) zeigt den Warm Run, der sich in der Interpretation der Ergebnisse und Beurteilung der Leistungsfähigkeit der einzelnen Systeme nur bei den Ausreißern vom

[8] *http://www.gregsadetsky.com/aol-data/* (letzter Abruf: 13.10.2014)

[9] *https://dumps.wikimedia.org/archive/2006/2006-12/enwiki/20061130/enwiki-20061130-pages-articles.xml.bz2* (letzter Abruf: 15.9.2014)

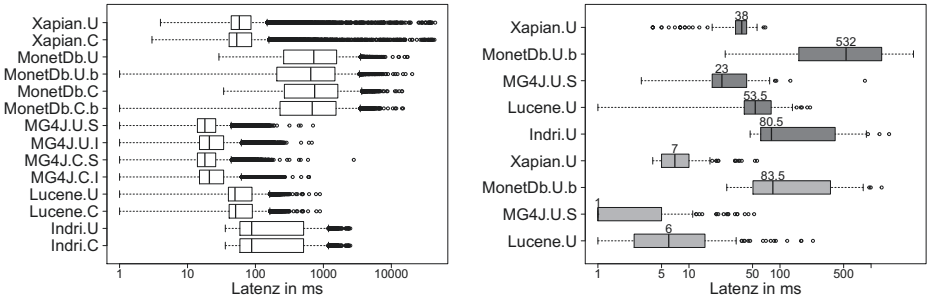

Abbildung 3: Links: Durchläufe über alle 15284 Anfragen für den Warm Run; b steht für Variante b) gemäß Abschnitt 3; U für das ext4 Dateisystem, C für das btrfs Dateisystem mit aktivierter zlib Kompression, I für das alte Indexformat von MG4J, S für das neue Indexformat von MG4J. Rechts: Disjunktive (dunkelgrau) und konjunktive Durchläufe (hellgrau) über 100 Anfragen im Warm Run.

Cold Run unterscheidet, wie auch aus Tab. 3 ersichtlich ist. Dabei ist eine klare Tendenz zu erkennen: MG4J und Lucene stellen die effizientesten Systeme dar. MG4J ist in den meisten Fällen doppelt so schnell wie Lucene. MG4J kann sich mit dem neuen Quasi-Succinct-Indexformat weiter gegenüber Lucene absetzen. Xapian und Indri folgen. Während Xapian im Median noch nahe bei Lucene liegt, zeigen die arithmetischen Mittel und die Boxplots[10], dass Xapian unter Ausreißern leidet. Indri ist bei Betrachtung des Medians und des arithmetischen Mittels darüber hinaus noch etwas langsamer. MonetDB stellt mit Abstand das schlechteste System dar. Mit ca. 650 ms im Median ist das System mehr als 10-mal langsamer als Lucene und gar mehr als 30-mal langsamer als das beste System MG4J mit neuem Indexformat. Zudem verhält sich das MonetDB-System im Gegensatz zu den anderen Systemen so, dass bei der großen Zahl von Anfragen bei der wiederholten Durchführung (Warm Run) sogar geringfügige Effizienzverschlechterungen zu verzeichnen sind. Eine Analyse der Messergebnisse zeigt, dass die Anfragezeiten bei den Ausreißern ansteigen. Bei Mühleisen et al. führt ein Warm Run mit lediglich 50 Anfragen zu einer Beschleunigung der Laufzeit. Für eine geringe Anzahl an 100 Anfragen ist dies bei uns ebenso der Fall (sowohl bei konjunktiver als auch disjunktiver Anfragebearbeitung). Allerdings verschlechtert sich das Laufzeitverhalten des Warm Runs gegenüber des Cold Runs, sofern 15284 Anfragen untersucht werden. Durchweg positiv stellt sich Variante b) dar und damit das Halten der *term-termid* Zuordnung im Hauptspeicher. Damit arbeitet das System ca. 60 ms schneller.

Vergleicht man die Ergebnisse auf komprimierten und unkomprimierten Dateisystemen ergeben sich weitere Auffälligkeiten. Während alle Systeme außer MonetDB von der Kompression im Cold Run profitieren, sind beim Warm Run keine positiven Auswirkungen mehr sichtbar. Es zeigt sich, dass durch die Kompression vor allem Ausreißer vermieden werden. Durch die Tatsache, dass die Systeme zunächst Teile des Index von der Fest-

[10] Sofern keine Ausreißer eingezeichnet sind, handelt es sich bei den Whiskern um das Minimum bzw. Maximum; andernfalls um das 1,5-fache des Interquartilabstands.

	Median				arithmetisches Mittel			
Kompression	inaktiv		aktiv		inaktiv		aktiv	
Durchlauf	cold	warm	cold	warm	cold	warm	cold	warm
MG4J.S	22	18	18	18	31	23	27	23
MG4J.I	25	21	22	18	35	29	33	29
Lucene	52	50	50	51	75	64	63	64
Xapian	62	58	56	53	246	188	230	195
Indri	102	88	98	88	293	277	285	278
MonetDB.b	643	652	657.5	685	884	937	914	963
MonetDB	705	722	717	744	942	1009	973	1016

Tabelle 3: Median und arithmetisches Mittel in Millisekunden der Durchläufe bei allen 15284 Anfragen auf ext4 Dateisystem und btrfs Dateisystem mit aktiver zlib Kompression; b steht für Variante b) gemäß Abschnitt 3; S für das neue MG4J-Indexformat, I für das alte MG4J-Indexformat.

platte in den Hauptspeicher laden müssen, scheint es schlüssig, dass Ausreißer vor allem während des Startvorgangs auftreten.

Dieses Verhalten ist für Lucene am ausgeprägtesten. Ein ähnliches Verhalten ist bei den anderen konventionellen IR-Bibliotheken ebenfalls feststellbar. Lucene erreicht erst nach ca. 100 Anfragen ein konstantes Latenzniveau. Durch Dateisystemkompression kann der Start des Systems abgesehen von der ersten Anfrage stark beschleunigt werden. Die Evaluierung über alle 15284 Anfragen zeigt jedoch, dass sich die Kompression nach den ersten 100 Anfragen kaum noch positiv auswirkt.

Der rechte Plot in Abb. 3 stellt eine konjunktive und disjunktive Anfragebearbeitung bei 100 zufällig gewählten Anfragen im Warm Run gegenüber und verdeutlicht, dass die disjunktive Anfragebearbeitung bei allen Systemen deutlich mehr Aufwand verursacht als die konjunktive Verarbeitung. Während bei der disjunktiven Variante Lucene schlechter als Xapian abschneidet, ist diese Tendenz im linken Teil der Abbildung bei Betrachtung aller Anfragen nicht klar vorhanden. Abb. 3 (rechts) bestätigt außerdem für disjunktive Anfragen die bereits für konjunktive Anfragen getroffenen Aussagen von Mühleisen et al., die zeigen, dass bei Betrachtung weniger Anfragen im Warm Run Lucene besser als Indri arbeitet, letzteres auch besser als MonetDB. Die Abbildung zeigt auch, dass sowohl bei konjunktiver als auch bei disjunktiver Anfragebearbeitung MG4J das leistungsfähigste System ist, gefolgt von Xapian. Bei Betrachtung aller 15284 Anfragen und disjunktiver Anfragebearbeitung (vgl. Abb. 3, links) ist Xapian jedoch Lucene unterlegen.

Aus Platzgründen nicht aufgeführte Laufzeitmessungen auf Basis der Ohsumed-Kollektion, die sich durch mehr Terme pro Anfrage (5,44 in Relation zu 3,87) auszeichnet, bestätigen zudem die Überlegenheit der konventionellen IR-Systeme gegenüber MonetDB.

5 Fazit

In dieser Arbeit haben wir ein auf dem spaltenorientierten DBMS MonetDB basierendes IR-System mit vier verschiedenen konventionellen IR-Systemen verglichen. Dabei wurde als Retrieval-Modell Okapi BM25 verwendet. Als Dokumentkollektion wurde die Wiki-

pedia aus dem Jahr 2006 verwendet, sowie reale Anfragen des AOL Query Logs aus dem gleichen Jahr. Wir konnten zeigen, dass hinsichtlich der Effizienz bei der Anfragebearbeitung ein deutlicher Unterschied zwischen den performantesten IR-Bibliotheken und einem auf dem spaltenorientierten DBMS MonetDB basierenden IR-System existiert. Lucene ist etwa 10-mal so schnell wie das MonetDB-System, MG4J mit neuem Indexformat gar 30-mal schneller. Das schlechteste konventionelle System ist immer noch über 6-mal schneller als das MonetDB-System. Damit eignet sich das MonetDB-System gut für IR-Prototypen und Effektivitätsevaluierungen in moderatem Umfang. Für einen produktiven Einsatz wie etwa als Websuchmaschine oder zur Unternehmenssuche sind IR-Systeme auf Basis invertierter Listen jedoch besser geeignet.

Des Weiteren haben wir das Startverhalten von konventionellen IR-Bibliotheken analysiert. Dabei konnten wir feststellen, dass diese beim Start einige Anfragen benötigen, um ein performantes Anfragezeitniveau zu erreichen. Besonders auffällig ist dieses Verhalten bei Lucene. Dort werden ca. 100 Anfragen benötigt bis das System warm ist. Durch Dateisystemkompression kann der Startvorgang stark beschleunigt werden. Im weiteren Verlauf wirkt sich die Kompression allerdings nicht mehr positiv auf die meisten Systeme aus. Mit dieser Erkenntnis können die Messungen von [MSLd14] in deren ersten Lauf im Speziellen für Lucene relativiert werden. Denn dort wurden lediglich 50 Anfragen an das System gestellt, wodurch das System sich noch deutlich in der Startphase befand. Wir konnten zudem zeigen, dass Lucene und andere konventionelle IR-Bibliotheken von Caching-Mechanismen profitieren und unabhängig von der konkreten Anfrage anfangs stetig an Performanz gewinnen.

Literatur

[BR11] Ricardo A. Baeza-Yates und Berthier A. Ribeiro-Neto. *Modern Information Retrieval – the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[GFHR97] David A. Grossman, Ophir Frieder, David O. Holmes und David C. Roberts. Integrating structured data and text: A relational approach. *Journal of the American Society for Information Science*, 48(2):122–132, 1997.

[JWR00] K. Sparck Jones, S. Walker und S. E. Robertson. A Probabilistic Model of Information Retrieval: Development and Comparative Experiments. *Inf. Process. Manage.*, 36(6):779–808, November 2000.

[MB11] Christian Middleton und Ricardo A. Baeza-Yates. Open Source Search Engines. In Ricardo A. Baeza-Yates und Berthier A. Ribeiro-Neto, Hrsg., *Modern Information Retrieval – the concepts and technology behind search, Second edition*, Kapitel A. Pearson Education Ltd., Harlow, England, 2011.

[MSLd14] Hannes Mühleisen, Thaer Samar, Jimmy Lin und Arjen de Vries. Old Dogs Are Great at New Tricks: Column Stores for IR Prototyping. In *Proc. of the 37th Intl. SIGIR Conf. on Research and Development in Information Retrieval*, Seiten 863–866, Gold Coast, Queensland, Australia, 2014. ACM.

[Roe13] Thomas Roelleke. Information Retrieval Models: Foundations and Relationships. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 5(3):1–163, 2013.

[Vig13] Sebastiano Vigna. Quasi-succinct Indices. In *Proc. of the 6th Intl. Conf. on Web Search and Data Mining*, Seiten 83–92, Rom, Italien, 2013. ACM.