Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 365–372

ISBN 978-83-60810-27-9
ISSN 1896-7094

Quality Benchmarking Relational Databases and Lucene in the TREC4 Adhoc Task Environment

Ahmet Arslan
Anadolu University
Computer Engineering Department
Eskisehir, Turkey
Email: aarslan2@anadolu.edu.tr

Ozgur Yilmazel
Anadolu University
Computer Engineering Department
Eskisehir, Turkey
Email: oyilmazel@anadolu.edu.tr

***Abstract*—The present work covers a comparison of the text retrieval qualities of open source relational databases and Lucene, which is a full text search engine library, over English documents. TREC-4 adhoc task is completed to compare both search effectiveness and search efficiency. Two relational database management systems and four different well-known English stemming algorithms have been tried. It has been found that language specific preprocessing improves retrieval quality for all systems. The results of the English text retrieval experiments by using Lucene are at par with top six results presented at TREC-4 automatic adhoc. Although open source relational databases integrated full text retrieval technology, their relevancy ranking mechanisms are not as good as Lucene's.**

I. Introduction

RELATIONAL database management systems (RDBMS) have been the preferred way of managing data for the past two decades. In recent years, data that many applications manage are moving more from structured to unstructured (free form text). Although relational databases are designed to handle structured data, many web applications use databases to manage and query unstructured data. With the increase in unstructured text, development of search engine libraries - which are specifically designed to quickly and effectively search large volumes of unstructured text - has been gaining momentum. There are several open source search engine libraries available with different features [1]. Many database vendors (IBM DB2[1], Microsoft SQL Server[2], MySQL[3], Oracle[4], PostgreSQL[5]) have recognized the need for free form text search and started implementing features that would support full-text search capabilities. Search engine libraries and relational databases each have unique advantages but also they have overlapping capabilities in common. In our previous work [2], we compared the full text search capabilities of different open source relational databases and Lucene on Turkish text documents. In this work we are extending our experiments by using English data set and deeply exploring full text search configuration parameters of relational databases.

[1]http://www-01.ibm.com/support/docview.wss?uid=swg27004103
[2]http://msdn.microsoft.com/en-us/library/ms142571.aspx
[3]http://dev.mysql.com/doc/refman/5.5/en/fulltext-search.html
[4]http://www.oracle.com/technology/products/text/index.html
[5]http://www.postgresql.org/docs/8.4/static/textsearch-intro.html

The paper is organized as follows. Section II briefly summarizes the related work, Section III explains the retrieval systems and stemming algorithms that we used, Section IV shows experimental results and our analysis on them, Section V gives summary of our observations about relational databases' full text search, and Section VI provides concluding remarks.

II. Related Work

There are many studies which evaluated the performance of search engine libraries over Text REtrieval Conference (TREC) test collections. TREC[6] is an annual conference aiming to encourage research in information retrieval (IR) based on large test collections. Database vendors also have evaluated their full-text search capabilities by participating in TREC competitions [3], [4]. Earlier papers [5], [6], [7] have described full text search capabilities and features of different relational databases, focusing specifically on the integration of free form text and structured data. The studies on comparison of IR systems are common, but there are no studies on Relational Database Management Systems' information retrieval qualities. The other studies have focused on hybrid IR-DB system solutions and integration of IR and databases.

Some unpublished articles discuss and compare different aspects of relational databases and search engines libraries. In [8], Marc Krellenstein explores the benefits of a full text search engine in comparison to a database. The article by David Smiley [9], addresses a scenario in which a web application needs to have a full text search capability. It discusses using the text search features of relational database versus using Apache Solr[7] - an open source search platform built on top of Lucene.

In a recent paper Yinan Jing and Chunwang Zhang [10] compared Lucene and a relational database in terms of query time. The data set used in their work was composed of auto-generated numeric and alphanumeric fields. They performed their test on these structured fields which are not tokenized therefore they didn't use full-text search but rather structured queries. A systematic comparison of text retrieval quality of relational databases and search engine libraries has not been done.

[6]http://trec.nist.gov/
[7]http://lucene.apache.org/solr/

III. Experimental Setup

In this study TREC-4 adhoc task was completed by querying forty nine topics (numbers 202-250) over 567,529 documents. Topic 201 was ignored since it retrieved no relevant documents and wasn't used in the actual evaluation of TREC-4 adhoc competition. DOCNO field - which is common for all documents - used as unique identifier in our experiments and the rest of the document - except DOCNO and DOCID fields - is taken as single textual field named contents. Topics are queried over this single field. The details of the document set and topics used in TREC-4 can be found here [11].

We used one open source information retrieval library (Apache Lucene 3.0.1) and two open source Relational Database Management Systems with full-text search capabilities (MySQL 5.5.3, PostgreSQL 8.4.4) for retrieval experiments. The latest versions of each system were used in our study in order to take account into latest features and improvements. We would like to experiment with other popular relational databases such as IBM DB2, Microsoft SQL Server and Oracle; however they do not permit disclosure results of any program benchmark tests without their prior consent.

All of the experiments were completed on Apple Mac Pro with two 2.8 GHz Quad-Core Intel Xeon processors and 6 GB 800 MHz DDR2 memory running Mac OS X Version 10.5.7. All test programs were implemented with the Java programming language, running on JDK 1.6.

A. Stemming Algorithms for English

Stemming is the most employed technique in IR to enhance retrieval quality in terms of recall. We investigated publicly available stemmers for the English language in our experiments. Probably the two most well-known stemming algorithms for English language are the Porter [12] stemming algorithm and the Lovins [13] stemming algorithm. Lovins is the first ever published stemming algorithm and removes 297 different endings using longest-match algorithm. Porter stemming algorithm created and maintained by Dr. Martin Porter. The algorithm can be best defined by its author's own words:

"The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflectional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems."[8]

English (Porter2)[9] stemming algorithm is a result of Dr. Martins attempt to improve the structure of the original Porter algorithm.

These algorithms are rule based and do not use dictionary or lexicon and preferred for their linear running time complexity since dictionary based stemmers can be sometimes slow for real-world web applications.

[8]http://tartarus.org/\textasciitildemartin/PorterStemmer/

[9]http://snowball.tartarus.org/algorithms/english/stemmer.html

TABLE I
Analyzer Building Blocks

ootb	Porter	Lovins, English	KStem
LetterTokenizer	LetterTokenizer	LetterTokenizer	LetterTokenizer
LowerCaseFilter	LowerCaseFilter	LowerCaseFilter	LowerCaseFilter
StopFilter	StopFilter	StopFilter	StopFilter
	PorterStemFilter	SnowballFilter	KStemFilter

KStem [14] stemming algorithm is a dictionary based inflectional stemming algorithm which uses human readable dictionary. It is a less aggressive stemmer than the standard Porter stemmer. It was written by Bob Krovetz, ported to Lucene by Sergio Guzman-Lara (UMASS Amherst).

We used these four different stemming algorithms for English to improve text retrieval performances of each system. To measure this improvement we included out-of-the-box settings of each system. In out-of-the-box option no preprocessing is done on the documents and queries. Built-in English support and stop-word list of each system is used.

B. Lucene

Lucene[10] is a powerful, free, open source IR library written entirely in Java. It is suitable for nearly any application that requires full-text search and its popularity is increasing because of simplicity, high performance, maturity and scalability.

Analysis, in Lucene, is the process where free form text is converted into tokens by tokenization, lowercasing, stemming and etc. Analysis process begins with a Tokenizer which breaks free form text into tokens. And then, the created token stream is fed into nested TokenFilters. TokenFilters can add, modify or delete its input token stream. For example lowercasing, removing common words, reducing words to a common base form or injecting synonyms occurs in TokenFilters. Lucene has several Tokenizer and TokenFilter implementations. An analyzer is an encapsulation of the analysis process which is an essential part of Lucene. Custom Analyzers can be built from a Tokenizer and a TokenFilter chaining pattern.

Five different analyzers are used in our runs and Table I shows the Analyzer building blocks used to create them. First column of Table I represents StopAnalyzer that comes with out-of-the-box Lucene. StopAnalyzer's default English stopword set contains 33 common words.

LetterTokenizer divides text at non-letters by capturing tokens as maximal strings of adjacent letters, as defined by `java.lang.Character.isLetter()` method. We have used this tokenizer because TREC-4 topics do not have any alphanumeric or numeric words. All tokens are composed of letters. (Except three acronyms U.S., U.K. and e.g. which are taken as stop words in our runs) Therefore we didn't index any numeric or alphanumeric tokens.

LowercaseFilter normalizes the token by lowercasing its text.

[10]http://lucene.apache.org

StopFilter removes tokens that exist in a provided list of stop words. In custom analyzers, we used 70 stopwords which are superset of StopAnalyzer's default stopword list.

SnowballFilter - that comes in contrib package of Lucene -, stems words using a Snowball-generated stemmer. Snowball[11], which is created by Dr. Martin, is a small string manipulation language specifically designed for creating stemming algorithms for use in Information Retrieval. A range of non-English stemmers are implemented by using snowball script, including Danish, Dutch, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish and Turkish [15]. There exists three English-specific stemmers (named English, Lovins, and Porter) implemented by using snowball script and available at Dr. Martins web site. These exact names can be passed as parameter to constructor of SnowballFilter to initialize a stem filter for that language in Lucene. In our experiments we used English and Lovins.

PorterStemFilter is java implementation of Porter algorithm and has the same behavior as SnowballFilter when Porter is used for the name argument to the SnowballFilter constructor. PorterStemFilter is much faster since it does not use Snowball Program.

KStemFilter stems words according to Bob Krovetz's kstem algorithm. Lucene does not have KStemmer implementation with the out-of-the-box settings. Source code of the stemmer is downloaded from web site of Center for Intelligent Information Retrieval - University of Massachusetts Amherst[12].

StopAnalyzer and four different custom Lucene analyzers—representing each stemming option—are used to create five Lucene indices. The same analyzers are used to search topics over each index.

C. MySQL

MySQL has support for full-text indexing and searching based on a space-vector model. Full-text indices can be created only on CHAR, VARCHAR, or TEXT columns of MyISAM tables. Full-text searching is performed using MATCH()...AGAINST() syntax which is introduced on June 2000. However MySQL has no linguistic support (stemming) for English or any other language. MySQL has a default stopword list for English[13] and removes them during indexing and searching. Like words included in the built-in stop word list, also words that are less than four or greater than 84 characters are also ignored by default in full-text searches. Table II shows MySQL's user-overridable full-text search parameters including their default values and descriptions. These parameters are defined by the system variables and can be obtained by executing

```
SHOW VARIABLES LIKE 'ft_%';
```

SQL statement. The other modifications such as disabling 50% threshold and changing tokenization behavior are non-trivial tasks that require source code modification and recompilation of MySQL.

1) Indexing: Five different MySQL tables with two columns (docno, contents) are created. For out-of-the-box option documents are inserted directly into the table. For the remaining stemming options, first they passed through respective Lucene Analyzer and then inserted into their tables. Full-text indices are built on tables thus:

```
ALTER TABLE docs ADD FULLTEXT (contents);
```

The indices are created after loading all data to tables because for large data sets, it is much faster than loading data into tables that have an existing FULLTEXT index.

2) Searching: MySQL full text search have basically three modes: natural language mode, Boolean mode and with query expansion.

Natural Language Full-Text Search[14] performs a natural language search for a query against a text collection. There are no operators in this mode and the stop-word list applies. Another interesting property of this mode is the elimination of query words that occur in more than or equal to half of the collection. In another words if a query word is present in at least 50% of the documents, it is treated as a stop-word. This mode is the default mode in MySQL and automatically sorts search results in order of decreasing relevance. Run using this mode is executed as follows:

```
SELECT docno, MATCH (contents) AGAINST ('<topic>')
AS score FROM docs WHERE MATCH (contents)
AGAINST ('<topic>') LIMIT 1000;
```

Relevance ranking algorithm of Natural Language mode uses Vector Space Model where rows and queries are represented as weighted vectors. MySQL uses a variant of the classic tf-idf (term frequency-inverse document frequency) weighting scheme along with pivoted document length normalization. Details of the ranking algorithm can be found here [16].

Boolean Full-Text Search[15] allows usage of implied Boolean operators ([no operator], +, -) and various advanced search methods like wildcard (*) and phrase search. A leading plus sign means required or mandatory operator. The word after the plus sign must exist in every row returned. A leading minus sign means prohibited operator. The word after the minus sign must not exist in any row returned. When no operator is specified, it means that this word is optional and should exist in returned rows. Rows that contain optional words will get higher scores. Complete list of supported operators are shown at the first row of Table II. Stop-word list applies but 50% threshold limitation does not apply with this mode. This mode does not automatically sort search results in order of decreasing relevance therefore 'ORDER BY score DESC' clause is added to SQL sentence in two runs using this mode:

[11] http://snowball.tartarus.org
[12] http://ciir.cs.umass.edu/
[13] http://dev.mysql.com/doc/refman/5.5/en/fulltext-stopwords.html
[14] http://dev.mysql.com/doc/refman/5.5/en/fulltext-natural-language.html
[15] http://dev.mysql.com/doc/refman/5.5/en/fulltext-boolean.html

TABLE II
MYSQL VARIABLES ASSOCIATED WITH FULLTEXT SEARCHING

Variable Name	Default Value	Description
ft_boolean_syntax	+ - >< () ~ * : "" & |	List of operators supported by boolean full-text searches performed using IN BOOLEAN MODE.
ft_max_word_len	84	Maximum length of the word to be included in a FULLTEXT index.
ft_min_word_len	4	Minimum length of the word to be included in a FULLTEXT index.
ft_query_expansion_limit	20	Number of top matches to use for full-text searches performed using WITH QUERY EXPANSION
ft_stopword_file	(built-in)	File from which to read the list of stopwords for full-text searches.

SELECT docno, MATCH (contents) AGAINST ('<topic>' IN BOOLEAN MODE) AS score FROM docs WHERE MATCH (contents) AGAINST ('<topic>' IN BOOLEAN MODE) ORDER BY score DESC LIMIT 1000;

Relevance ranking algorithm of Boolean Mode is quite different from Natural Language Mode. This mode provides only simplistic relevance ranking [17]. It is defined as the sum of weights of matched words in query string. Weights are defined by boolean operators. This ranking mechanism produces always 1 when + operator is used before query terms. For example in the pure required type query '+term1 +term2 +term3 +term4' weights of each term will be 1/4. When no operator is used, score is equal to count of matching query terms. For example in the pure optional type query 'term1 term2 term3 term4' weights of each term will be 1. If a document contains three of these terms, it will get score of three. This ranking algorithm does not use collection-wide statistics (inverse document frequency) therefore Boolean Mode full text searches does not require FULLTEXT indices. Other interesting property of this ranking is that calculated scores are always greater than or equal to one.

Full-Text Searches with Query Expansion[16] applies classic blind relevance feedback which is also known as Pseudo relevance feedback. It performs natural language mode search twice and assumes top few (controlled by ft_query_expansion_limit variable which has a default value of 20) results of first search are relevant. And then, it appends these documents to the original query to perform second search. This mode can be activated by adding WITH QUERY EXPANSION or IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION modifiers to the query. Functions of these two modifiers are exactly the same and they yield same results. Since this mode is modified version of a natural language search, it automatically sorts search results in order of decreasing relevance. Runs using this mode are executed as follows:

SELECT docno, MATCH (contents) AGAINST ('<topic>' WITH QUERY EXPANSION) AS score FROM docs WHERE MATCH (contents) AGAINST ('<topic>' WITH QUERY EXPANSION) LIMIT 1000;

[16]http://dev.mysql.com/doc/refman/5.5/en/fulltext-query-expansion.html

D. PostgreSQL

PostgreSQL supports full text indexing of textual documents and relevance ranking for full text database searching.

In PostgreSQL, dictionaries[17] allow fine-grained control over how tokens are normalized. PostgreSQL provides several predefined dictionaries for linguistic support, available for many languages, and English is one of them.

PostgreSQL have two special data type tsvector and tsquery representing preprocessed documents and processed queries with support of boolean operators respectively. These data types are vector representation of documents and queries like in vector space model. There are some functions to convert documents or queries into these data types. to_tsvector is used to transform a document to tsvector data type while to_tsquery and plainto_tsquery are used for converting a query to the proper tsquery data type. All of these transformation functions take a language specific configuration parameter. Full text searching is done using the match operator @@, which returns true if a tsvector (document) matches a tsquery (query).

1) Indexing: Five different PostgreSQL tables with two columns (docno, contents) are created. An additional tsvector type column named ts_col is added to the table.

ALTER TABLE docs ADD COLUMN ts_col tsvector;

For out-of-the-box option, documents are inserted into the tables without any preprocessing. Then ts_col column is populated from contents column by invoking to_tsvector function with the configuration parameter for the English language that comes with out-of-the-box settings of PostgreSQL.

UPDATE docs SET ts_col = to_tsvector('english', contents);

For the remaining stemming options, documents are first analyzed by respective Lucene Analyzer and then inserted into their tables. The ts_col column is populated with the output of to_tsvector function, but this time using the simple template parameter which behaves like no language is specified because data in the table are already preprocessed.

UPDATE docs SET ts_col = to_tsvector ('pg_catalog.simple', contents);

PostgreSQL offers two kinds of indices that can be used to speed up full text searches.

- GiST (Generalized Search Tree) based index
- GIN (Generalized Inverted Index) based index

[17]http://www.postgresql.org/docs/8.4/static/textsearch-dictionaries.html

TABLE III
POSTGRESQL DOCUMENT LENGTH NORMALIZATION OPTIONS

Mode	Meaning
0	(the default) ignores the document length
1	divides the rank by 1 + the logarithm of the document length
2	divides the rank by the document length
4	divides the rank by the mean harmonic distance between extents (this is implemented only by ts_rank_cd)
8	divides the rank by the number of unique words in document
16	divides the rank by 1 + the logarithm of the number of unique words in document
32	divides the rank by itself + 1

Since GIN index is best for static data and searches are about three times faster than GiST, GIN index was created to speed up the search as follows:

CREATE INDEX text_index ON docs USING gin (ts_col);

2) Searching: PostgreSQL provides two functions to_tsquery and plainto_tsquery for converting a query to the tsquery data type. plainto_tsquery transforms unformatted text querytext to tsquery by parsing and normalizing text, then inserting the & (AND) Boolean operator between surviving words. to_tsquery creates a tsquery value from querytext, which must consist of single tokens separated by the Boolean operators & (AND), | (OR) and ! (NOT). Note that to_tsquery with AND operator is identical to plainto_tsquery.

To rank search results, PostgreSQL provides two predefined ranking functions to calculate similarity between a tsvector (document) and a tsquery (query). These are standard ranking function (ts_rank) and cover density ranking function (ts_rank_cd) [18]. While the ts_rank does not consider term position proximity, the ts_rand_cd ranking function punishes documents where the search terms are further apart. PostgreSQL's both ranking functions do not use any global information (inverse document frequency); therefore indices are not mandatory but can be used to speed up full text searching.

Both ranking functions take an integer normalization option[18] that specifies whether and how document length normalization will be done. Table III shows the document length normalization options and their effect on ranking mechanism that PosgreSQL supports with the out-of-the-box settings.

Note that mode 32 is used just to scale ranks between zero and one. The ordering of search results does not change in this mode.

In out-of-the-box option to obtain pure OR queries, description parts of topics are tokenized at white spaces and then OR operator ("|") is inserted between each token. English language configuration is used in this mode. SQL queries are submitted as follows:

SELECT docno, *ranking_function*(ts_col, query, *normalization*) AS rank FROM docs, to_tsquery('english','<topic>') query WHERE query @@ ts_col ORDER BY rank DESC LIMIT 1000;

In remaining stemming options to obtain pure OR queries, description parts of topics are first passed through respective Lucene Analyzer and then OR operator ("|") is inserted between surviving words. Simple template configuration is used because queries are already analyzed. SQL queries are submitted as follows:

SELECT docno, *ranking_function*(ts_col, query, *normalization*) AS rank FROM docs, to_tsquery('pg_catalog.simple','<topic>') query WHERE query @@ ts_col ORDER BY rank DESC LIMIT 1000;

IV. EXPERIMENTAL RESULTS

Since the overview of TREC-4 paper presents the precision/recall curves for the groups with the highest non-interpolated average precision (**MAP**) and the runs are ranked by the average precision, in this work the same metric is used in global evaluation. While evaluating each system in itself we also presented precision at 5 (**P@5**) and precision at 10 (**P@10**) values as well as search time (**sec/q**) per query. Additionally results of citri2 run of Royal Melbourne Institute of Technology [19] (one of the best TREC-4 automatic adhoc) were included in experimental result for comparison.

The evaluation measures presented in this paper are calculated by using Chris Buckley's `trec_eval`[19] package (version 8.1) which is the standard tool used by the TREC community for evaluating an adhoc retrieval run, given the results file and a standard set of judged results.

In our calculations a cut-off level of 1000 is used, which defines the retrieved set as the top 1000 documents in the ranked list which is similar to official TREC usage:

trec_eval -c -M1000 official_qrels submitted_results

All retrieval systems are designed in a similar fashion to standard TREC-type adhoc runs that retrieve maximum 1000 documents per topic.

Topics created for the TREC-4 adhoc task consist of only one field (description). Therefore we ran retrieval experiments over each index, by using description-only queries. In this work completely automatic query construction is used. Description only queries had on average 16 terms with stop words, 9 terms without stop words.

A. Lucene

Lucene's scoring[20] mechanism uses both the Vector Space Model and the Boolean Model. The Boolean model is used to first filter the documents to be used in score calculation. Lucene's scoring algorithm implements cosine similarity between tf-idf weighted documents and queries. It adds several factors to cosine similarity including document length normalization. Default length normalization function divides the score by square root of the number of words in the document.

[18]http://www.postgresql.org/docs/8.4/static/textsearch-controls.html

[19]http://trec.nist.gov/trec_eval/

[20]http://lucene.apache.org/java/3_0_1/scoring.html

TABLE IV
LUCENE SEARCH QUALITY COMPARISON

Run	MAP	P@5	P@10	sec/q
StopAnalyzer OR operator	**0.1645**	**0.4939**	**0.4163**	0.0452
StopAnalyzer AND operator	0.0011	0.0490	0.0286	0.0067

TABLE V
MYSQL SEARCH QUALITY COMPARISON

Run	MAP	P@5	P@10	sec/q
natural language mode	**0.1182**	**0.3388**	**0.3204**	1.2244
boolean mode AND operator	0.0023	0.0612	0.0449	0.0816
boolean mode OR operator	0.0318	0.1429	0.1041	3.4693
with query expansion limit 3	0.0249	0.0939	0.0755	8.1428
with query expansion limit 5	0.0405	0.1426	0.1122	10.4285
with query expansion limit 10	0.0326	0.1102	0.0939	14.7346
with query expansion limit 15	0.0229	0.0735	0.0633	18.0612
with query expansion limit 20	0.0331	0.1265	0.0898	20.6530

Details of the Lucene scoring can be found in chapter 3.3 of [15].

Lucene allows selecting between two Boolean operators (AND, OR) when performing search. Search quality results of these two operators are given in Table IV. Result set of OR operator is superset of the result set of AND operator. In other words, result set of OR operator already contains result set of AND operator. Moreover Lucene gives higher scores to the documents that contain more query terms. This fact implies that highest ranked documents will usually have the most ORed query terms among documents returned. OR operator used in remaining Lucene runs since it yields better results than AND operator.

B. MySQL

Total eight runs performed (with the out-of-the-box settings) to determine which type of MySQL full-text search is superior. Search quality results of our MySQL runs are given in Table V. It includes different boolean operators and search options described in section III-C2.

Boolean mode with pure required and pure optional queries performed badly due to its simplistic ranking mechanism which is described in section III-C2.

Blind relevance feedback (BRF) is used in TREC competitions and usually improves performance in TREC adhoc tasks. For example Cornell SMART system [20] at TREC 4 applied BRF (with good success) by adding the most frequently occurring 50 single terms and 10 phrases from the top 20 documents to initial query. However in MySQL it didn't perform as expected. To investigate this behavior we ran experiments with query expansion with five different ft_query_expansion_limit values. However results were still lower than natural language mode. We compared individual MAP values of natural language mode run and best with query expansion mode (limit 5) run and found that in 4 topics query expansion performed better. MySQL Reference Manual does not explain details of the query expansion mechanism but we suspect that this is due to appending whole document text to the initial query. Appending whole document to the initial query increases noise significantly and returns nonrelevant documents.

It is easy to understand TREC-like runs; natural language mode is more suitable due to its sophisticated ranking algorithm and the 50% threshold limitation. Query terms that occur in half of the documents in the collection have no distinctive property. Such words alone would return at least half of the documents in the collection. Natural language mode that yields best results in terms of retrieval quality is selected as best representative of MySQL.

C. PostgreSQL

In PostgreSQL, both matching operator (@@) and ranking functions takes same two parameters, tsquery and tsvector. We observed that when AND is used as matching operator, some topics did not return any documents while the others returned a few documents. Pure AND queries returned 13 documents per topic on the average. This behavior of AND operator yield very poor retrieval quality. Therefore - as described in section III-D2 - initially we used OR for both matching operator and ranking functions in our runs. By doing so, we obtained better results than AND operator.

When we examined our submitted qrels file, we observed that many documents ranked exactly with the same float value for a particular topic. Further investigations revealed that one of the ranking functions (ts_rank) does not play well with OR operator. As it can be understood from first two rows of Table VI ts_rank does not take account into how many ORed terms match when all query terms occur in a document. When it is used with AND operator, this behavior reverses and score increases with the number of matched terms. Interestingly ts_rank with AND operator (last row), does not yield zero for the document that does not contain all of the query terms. On the other hand ts_rank_cd function yields zero in this scenario. Also score produced by ts_rank_cd with OR operator increases as the number of the query terms found in the specified document increases.

After these observations we concluded that it is more convenient to use ts_rank_cd with OR operator while rank_cd with AND operator in our score calculations. We keep using OR operator for matching function in our remaining runs. Note that if we were to use ts_rank_cd with AND operator, the documents that do not contain all of the query terms would get a rank of zero. To obtain pure AND tsquery for use with ts_rank function, topics are fed into plainto_tsquery function. SQL sentence of runs using ts_rank is modified as follows:

SELECT docno, ts_rank(ts_col,
plainto_tsquery('english','<topic>'), *normalization*) AS
rank FROM docs WHERE to_tsquery('english','<topic>')
@@ ts_col ORDER BY rank DESC LIMIT 1000;

Total eleven different runs performed (with the out-of-the-box settings) to determine which ranking function and doc-

TABLE VI
POSTGRESQL RANKING FUNCTIONS WITH BOOLEAN OPERATORS

SQL: SELECT	ts_rank	ts_rank_cd
to_tsvector('t1 t2 t3'), to_tsquery('t1 | t2')	0.0607	0.2
to_tsvector('t1 t2 t3'), to_tsquery('t1 | t2 | t3')	0.0607	0.3
to_tsvector('t1 t2 t3'), to_tsquery('t1 | t2 | t4')	0.0405	0.2
to_tsvector('t1 t2 t3'), to_tsquery('t1 & t2')	0.0991	0.1
to_tsvector('t1 t2 t3'), to_tsquery('t1 & t2 & t3')	0.2683	0.1
to_tsvector('t1 t2 t3'), to_tsquery('t1 & t2 & t4')	0.0991	0

TABLE VII
POSTGRESQL SEARCH QUALITY COMPARISON

Run	MAP	P@5	P@10	sec/q
ts_rank mode 0	0.0818	0.1878	0.1755	19.0408
ts_rank mode 1	**0.1071**	**0.3102**	**0.2898**	19.5918
ts_rank mode 2	0.0441	0.1102	0.1184	19.5306
ts_rank mode 8	0.0505	0.1143	0.1265	19.0204
ts_rank mode 16	0.0987	0.2612	0.2714	19.0408
ts_rank_cd mode 0	0.0065	0.0163	0.0122	21.3673
ts_rank_cd mode 1	0.0111	0.0204	0.0163	21.6122
ts_rank_cd mode 2	0.0267	0.0735	0.0837	21.5510
ts_rank_cd mode 4	0.0465	0.0980	0.0735	21.0816
ts_rank_cd mode 8	0.0169	0.0245	0.0306	21.0612
ts_rank_cd mode 16	0.0094	0.0122	0.0143	21.1224

ument length normalization combination yields best results. Search quality results of PostgreSQL runs are given in Table VII. It includes six different document length normalization options and two ranking functions described in section III-D2. Note that option 4 is supported only by ts_rank_cd and option 32 is just a cosmetic change so that it is not included in our runs.

In all of runs ts_rank performed better than ts_rank_cd in terms of both search quality and search time. ts_rank_cd yielded highest MAP value with mode 4 which is implemented just for it. Among two predefined ranking functions and six length normalization options, standard ranking (ts_rank) function with the normalization option 1, yielded highest MAP value. Therefore this combination is selected as best representative of PostgreSQL.

D. Global Evaluation

In this section we compare best representative of each system. In Table VIII, the retrieval qualities (in terms of mean average precision) of each system are compared. Each system performed its best with KStemmer while overall best performing one is Lucene (with KStemmer). It is observed that among four different stemming methods, best performing is the KStemmer for the English language. Also English (porter2) stemming algorithm performed slightly better than the original porter stemming algorithm in all systems. KStemmer with Lucene performed slightly (2.8%) better than citri2 [19] which was at the seventh seat at TREC-4 automatic adhoc competition with the MAP value of 0.1956. PosgreSQL's poor

TABLE VIII
MEAN AVERAGE PRECISION (MAP) VALUES

	ootb	Lovins	Porter	Porter2	KStem
MySQL	0.1182	0.1152	0.1314	0.1325	0.1394
PostgreSQL	0.1071	0.0973	0.1004	0.1010	0.1094
Lucene	0.1645	0.1726	0.1931	0.1941	**0.2012**

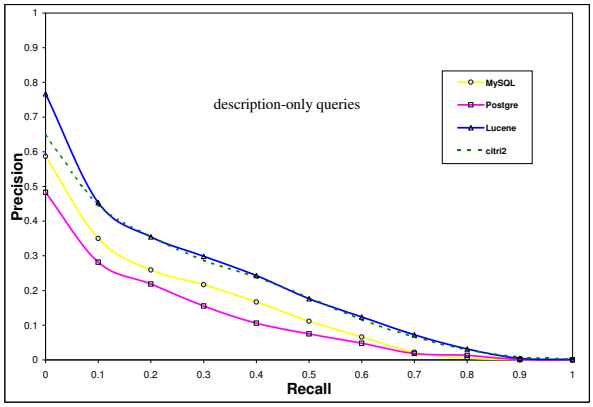

Fig. 1. Interpolated precision - recall graph

performance can be explained by lack of inverse document frequency component in its ranking mechanism.

Among the several metrics eleven point precision recall curves of each system are presented in the TREC-4 overview paper [11]. Therefore the curves of each system (with KStemmer) and citri2 are plotted on Figure 1 to compare information retrieval systems visually. Lucene whose curve is the closest to the upper right-hand corner of the graph (where recall and precision are maximized) indicates the best performance.

Although main focus of this study is text retrieval quality benchmarking, as a side note, indexing times of each system are presented on Figure 2. In our experiments relational database settings are optimized for static data. Note that in PostgreSQL, GIN index is recommended[21] for static data since lookups are about three times faster than GiST, on the other hand GIN index takes about three times longer to build than GiST.

Average searching times of each system's different runs are depicted separately in **sec/q** column of tables in previous subsections. Relational Databases' searching and indexing speed are much slower than Lucene. With the response time under 50 milliseconds per query, Lucene would perfectly satisfy online web users.

V. OBSERVATIONS

Many Relational Databases have full text search functionalities but all have different syntax so there is no real standard between different vendors. Details of their inner algorithms are not well-documented.

All three systems use document length normalization in order to prevent long documents taking over. Lucene and

[21] http://www.postgresql.org/docs/8.4/static/textsearch-indexes.html

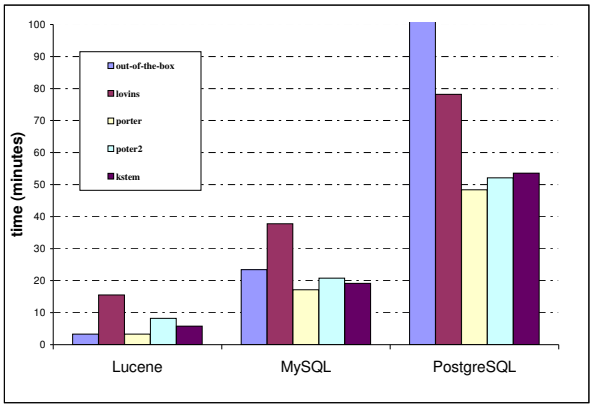

Fig. 2. Average Indexing Times

MySQL uses idf component in their score calculation while PostgreSQL does not. MySQL's full text search is limited to only MyISAM tables and has a few tuning parameters. PostgreSQL has more configurable parameters through dictionaries that provide stemming, synonym expansion, stop word removal etc. Available PosrgreSQL dictionaries are Stop Words, Simple, Synonym, Thesaurus, Ispell and Snowball. PosgreSQL's full text search, similar to Boolean mode in MySQL, does not require full text indices. In PostgreSQL, it is possible to immediately see the output of full text related functions and debug this way. For example executing SELECT to_tsvector('english', 'Testing the English configuration'); displays `'configur':4 'english':3 'test':1`

PostgreSQL and Lucene have highlighting feature that can generate snippets where query terms are highlighted. Generally users like to see which part of the document matches their queries.

VI. Conclusion

Our work in comparing the performances of relational databases and information retrieval libraries showed that; for TREC-4 adhoc collection, Lucene produced the best results in terms of efficiency and effectiveness. Lucene's out-of-the-box search quality reached to top six for TREC-4 adhoc evaluation. Although relational databases provide easy to use full text search capabilities that do not require an additional system installation and maintenance, without linguistic preprocessing their search quality is quite low. Due to the impractical response and indexing times, open source relational databases are unsuitable to be installed as a full text search solution for high traffic web applications.

References

[1] C. Middleton and R. Baeza-yates, "A comparison of open source search engines," Dec. 03 2008. [Online]. Available: http://wrg.upf.edu/WRG/dctos/Middleton-Baeza.pdf

[2] A. Arslan and O. Yilmazel, "A comparison of relational databases and information retrieval libraries on turkish text retrieval," in *Natural Language Processing and Knowledge Engineering, 2008. NLP-KE '08. International Conference on*, 19-22 2008, pp. 1 –8.

[3] K. Mahesh, J. Kud, and P. Dixon, "Oracle at trec8: A lexical approach," in *Text REtrieval Conference (TREC) TREC-8 Proceedings*. Department of Commerce, National Institute of Standards and Technology, 1999, pp. 207–??, nIST Special Publication 500-246: The Eighth Text REtrieval Conference (TREC 8). [Online]. Available: http://trec.nist.gov/pubs/trec8/papers/orcl99man.pdf

[4] S. Alpha, P. Dixon, C. Liao, and C. Yang, "Oracle at TREC 10: Filtering and question-answering," in *Text REtrieval Conference (TREC) TREC 2001 Proceedings*. Department of Commerce, National Institute of Standards and Technology, 2001, pp. 423–??, nIST Special Publication 500-250: The Tenth Text REtrieval Conference (TREC 2001). [Online]. Available: http://trec.nist.gov/pubs/trec10/papers/orcltrec10.pdf

[5] A. Maier and D. E. Simmen, "DB2 optimization in support of full text search," *IEEE Data Eng. Bull*, vol. 24, no. 4, pp. 3–6, 2001. [Online]. Available: http://sites.computer.org/debull/A01DEC-CD.pdf

[6] J. R. Hamilton and T. K. Nayak, "Microsoft SQL server full-text search," *IEEE Data Eng. Bull*, vol. 24, no. 4, pp. 7–10, 2001. [Online]. Available: http://sites.computer.org/debull/A01DEC-CD.pdf

[7] P. Dixon, "Basics of oracle text retrieval," *IEEE Data Eng. Bull*, vol. 24, no. 4, pp. 11–14, 2001. [Online]. Available: http://sites.computer.org/debull/A01DEC-CD.pdf

[8] M. Krellenstein, "Search engine versus dbms," Lucid Imagination. [Online]. Available: http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Search-Engine-versus-DBMS

[9] D. Smiley, "Text search, your database or solr," Packt Publishing, Nov. 2009. [Online]. Available: http://www.packtpub.com/article/text-search-your-database-or-solr

[10] Y. Jing, C. Zhang, and X. Wang, "An empirical study on performance comparison of lucene and relational database," in *Communication Software and Networks, 2009. ICCSN '09. International Conference on*, 27-28 2009, pp. 336 –340.

[11] D. Harman, "Overview of the fourth text retrieval conf. (TREC-4)," *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, 1996. [Online]. Available: http://trec.nist.gov/pubs/trec4/overview.ps.gz

[12] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[13] J. B. Lovins, "Development of a stemming algorithm," *Mechanical Translation and Computational Linguistics*, vol. 11, no. 1-2, pp. 22–31, 1968.

[14] R. Krovetz, "Viewing morphology as an inference process," in *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 1993, pp. 191–202. [Online]. Available: http://ciir.cs.umass.edu/pubfiles/ir-35.pdf

[15] O. G. Erik Hatcher and M. McCandless, *Lucene In Action*, 2nd ed. Manning Publications, 2010.

[16] *MySQL Internals Manual*, MySQL AB, Inc. [Online]. Available: http://forge.mysql.com/wiki/MySQL_Internals_Algorithms\#Full-text_Search

[17] S. Golubchik, "Mysql fulltext search," MySQL AB, Nov. 2004. [Online]. Available: http://forge.mysql.com/w/images/c/c5/Fulltext.pdf

[18] C. L. A. Clarke, G. V. Cormack, and E. A. Tudhope, "Relevance ranking for one to three term queries," *Inf. Process. Manage.*, vol. 36, no. 2, pp. 291–311, 2000.

[19] R. Wilkinson, J. Zobel, and R. Sacks-Davis, "Similarity measures for short queries," in *Text REtrieval Conference (TREC) TREC-4 Proceedings*. Department of Commerce, National Institute of Standards and Technology, 1995, pp. 277–285, nIST Special Publication 500-236: The Fourth Text REtrieval Conference (TREC-4). [Online]. Available: http://trec.nist.gov/pubs/trec4/papers/citri.ps.gz

[20] C. Buckley, A. Singhal, M. Mitra, and G. Salton, "New retrieval approaches using SMART: TREC 4," in *NIST Special Publication 500-236: The Fourth Text REtrieval Conference (TREC-4)*, D. Harman, Ed. Department of Commerce, National Institute of Standards and Technology, Nov. 1995. [Online]. Available: http://trec.nist.gov/pubs/trec4/papers/Cornell_trec4.ps.gz