Cross-instance Search System

Search Engine Comparison

Martin Haye
Email: martin@snyder-haye.com
January 2004

1. Introduction

The cross-instance search system requires an underlying full-text indexing and search engine. Since CDL envisions a sophisticated query system, writing such an engine from scratch would be prohibitively time-consuming. Thus, a search has been undertaken to locate a suitable existing engine.

First we undertook an initial survey of a large number of full-text engines. From these the field was limited to three candidates for further testing, on the basis of the following essential requirements:

- ❑ Open-source
- ❑ Free (as in beer)
- ❑ Relevance ranking
- ❑ Boolean operators
- ❑ Proximity searching

Four engines met all these requirements: Lucene, OpenFTS, Xapian, and Zebra. Initial index runs and query tests were performed on all four. All except OpenFTS performed well enough to make the finals, but OpenFTS showed long index times and very poor query speed (roughly an order of magnitude worse than the other engines). Given this disappointing performance, further rigorous tests seemed pointless, and I eliminated OpenFTS.

The remainder of this paper details the rigorous comparison and testing of the remaining engines: **Lucene**, **Xapian**, and **Zebra**.

For reference, the next six runners-up are given below with a comprehensive feature matrix.

	Amberfish	Guilda	XQEngine	Swish-E	ASPseek	OpenFTS
Owner	Etymon Systems	CDL	FatDog / SourceForge	swish-e .org	aspseek .org	XWare / SourceForge
Language	C	Perl	Java	C	C++	Perl/C
API	cmdline	Perl	Java	C	CGI	Perl
Proximity search	No	No	No	No	No	Yes
Relevance ranking	Yes	Yes	No	Yes	Yes	Yes
Range	No	No	Yes	No	No	No

operators						
UNICODE	No	Maybe	Yes	No	Yes	Partial
Wildcards	No	Yes	No	Yes	Yes	No
Fuzzy search	No	No	No	Yes	No	No
Arbitrary fields	No	Yes	Yes	No	Partial	Yes

2. Feature Comparison

2.1 Community Support, Documentation

	Lucene	**Xapian**	**Zebra**
Owner	Apache Jakarta	xapian.org	IndexData Corp.
Developer community	Formal open group	Informal closed group	Formal closed group
Current developers	Doug Cutting, and 14 others	Former employees of BrightStation PLC	Employees and partners of IndexData Corp.
Documentation	Extensive, high quality.	Sparse, adequate.	Huge, dense, high quality
Outside articles	Many	No	Many

All three engines appear to be in active development (with several releases in 2003), so it seems unlikely CDL would become stuck with an un-maintained tool.

It's harder to put a finger on the level of activity, but it seems that Lucene has the highest level of outside interest in improving the engine itself, while Lucene and Zebra both have lots of people asking deployment questions. By contrast, Xapian has garnered relatively little interest, at least judged by its paucity of Google hits.

Win: Lucene and Zebra

2.2 Platform

	Lucene	**Xapian**	**Zebra**
Platforms	All	All	All
Native language	Java	C++	C
Java API	Yes	No	No

Lucene has the advantage here, in that interfacing it to the Java servlet/query system will be quite simple. Java code also has the advantage of being relatively "safe", in the sense that memory leaks are uncommon, and crashes are less likely. (By contrast, during the indexing process, Xapian leaked memory at a significant rate.)

All of the engines claim to run on all major UNIX-variants in addition to Windows. However, the C/C++ engines will inevitably be somewhat difficult to get running on any specific combination of GCC/make/autoconf, etc.

Win: Lucene

2.3 *Query Features*

	Lucene	Xapian	Zebra
Wildcards (e.g. `fishe*`, `m?re`)	Yes	No	Yes
Range operators (e.g. `12 ≤ x ≤ 18`)	Yes	No	Yes
Fuzzy searching (based on edit dist.)	Yes	No	Yes
Term boosting (boost relevance of a term)	Yes	No	No
Arbitrary fields (date, author, ARK, etc.)	Yes	No	Yes
Stemming (search for `help` retrieves `helper`, `helping`, `helps`, etc.)	English + 11 other lang.	English + 12 other lang.	No
Thesaurus expansion (search for `cook` retrieves `cook`, `chef`, `prepare`, etc.)	No	No	No

All of the engines support:

- Boolean operators (i.e. “and”, “or”, “not”, “+”, “-“)
- Phrase queries (e.g. “tom thumb” – all words must appear together, in order)
- Proximity searching with adjustable range
- Relevance ranking
- Partial result sets (e.g. returning results 10 through 20, instead of all results)

Win: Lucene, with Zebra close behind

2.4 *Unique Features*

Proximity ranking (Lucene only): When performing a proximity search, Lucene increases the ranking of hits where the words are closer together.

Query expansion (Xapian only): This interesting feature kicks in *after* a query has been performed. The user selects a few of the documents that are most relevant to their query. Then Xapian can suggest additional terms to supplement the query, to get more results “like these”. Alternately Xapian can simply fetch a new set of documents “like these”.

Standards-based approach (Zebra only): Zebra implements a large subset of the ANSI/NSIO Z39.50 Protocol. Going with this approach would provide three benefits: (1) we could theoretically replace the underlying search engine with any of several other Z39.50 implementations; (2) since the standard is a client/server protocol, indexes could be easily distributed over several machines, or even conceivably across a network; and (3) we might be able to interface very easily with other libraries which run Z39.50 servers.

Win: All

3. Performance Comparison

3.1 *Test Methodology*

For each engine, we want to know:

- How fast is the indexer?
- How efficient (both in disk access time and space) are the resultant indexes?
- How fast is the query engine?

The eventual system we build will need to provide results at two levels:
(1) the top N (e.g. 10) documents matching the query, and
(2) for each document, the top M (e.g. 3) hits within that document (where a hit would be all the words in the query, near one another).

One strategy would be to keep a single index, where an index unit would be a "chunk" of, say, 50 words. Then one could query for all the in-document hits, and then synthesize the overall document scores. However, it seems unlikely that this would yield a quality score for each document, since it would essentially ignore most of that document's chunks. To avoid this, the system would have to spend time fetching essentially all the hits, instead of just the top M or N.

A better strategy is to maintain two types of indexes. The first index would contain each document as a single indexable unit, and this would provide the M document hits. The second index would contain chunked versions of the documents to provide the N hits within each of the M documents. These tests attempt to simulate this latter strategy.

The set of input documents was derived from a full drop of the 2,809 texts served by dynaXML as of January 5. To eliminate differences in stop-word lists and vagaries of parsing XML, I extracted just the text nodes from each document, converted them to lower case, removed all punctuation except periods, and replaced all stop words with "-". As a final step, each file was broken into 50-word chunks separated by newlines.

Then I configured each engine to produce the two indexes mentioned above: (1) a per-document index, and (2) a per-chunk index.

To test query speed, I developed a set of "typical" queries involving one to several words, using combinations of Boolean operators, proximity, and phrase searches. Additionally I added a few "pathological" cases to test the inner loops of the engines.

I tested queries with a "cold" cache, simulating a query on a term that hasn't been seen in a while, and with a "warm" cache, simulating repeated or similar queries. The cold cache essentially tests how quickly the index data can be located and read from disk, whereas the warm cache tests the speed of in-memory calculations.

3.2 Indexing Speed

As outlined above, each engine was configured to index whole document units (2,809 documents) and 50-word units (2,863,095 "chunks".) The start and end times were recorded, and the elapsed times in hours and minutes are given in the table below.

*Note: The Xapian chunked index ran all night (581 minutes) and only completed 300 of the 2809 documents. Several experiments have failed to speed up the process, so the time given here was extrapolated from the partial run.

	Lucene	Xapian	Zebra
Whole-doc index	00:52 (52 min)	04:28 (268 min)	01:00 (60 min)
Chunked index	09:14 (554 min)	*90:40 (5440 min)	01:11 (71 min)

It is interesting to note that Lucene is a little faster at adding entire documents, while Zebra is much faster at adding chunked documents. This implies Lucene spends quite a bit of time updating the structures for a single chunk, while its process for adding a term is quite efficient.

Win: Zebra overall, Lucene for whole-doc, Zebra for chunked

3.3 Index Size

I calculated the size of each index and divided it by the size of the input documents; the resulting ratios are given in the table below. The chunk indexes are all significantly larger than whole-doc indexes for two reasons: (1) they contain many more index units; and (2) the actual document text is recorded verbatim to simulate being able to show hits in context, whereas the whole-doc indexes do not need to store the document data.

*Note: Again, the Xapian chunked index number has been extrapolated from the run which completed 300 documents (the truncated output size was 1182M.)

	Lucene	Xapian	Zebra
Whole-doc index	26% (293M)	94% (1044M)	38% (418.1M)
Chunked index	133% (1479M)	*995% (11060M)	51% (562.7M)

One interesting note is that size ratio (whole-doc index size vs. chunked index size) for a given engine is roughly in proportion to the time ratio for that same engine. It makes sense.

Also of note is that Zebra is very efficient at storing raw document text (for the chunked index) on disk, probably using some sort of compression.

Win: Zebra overall, Lucene for whole-doc, Zebra for chunked

3.4 Basic query speed

For each engine I performed an identical set of roughly "typical" queries, drawn in part from old dynaXML log files, and representing a mix of single-term, phrase, Boolean, and proximity searches. Here are the 20 queries:

- literature

- "fertility rate"
- "susan v gallagher coetzee"
- montagne
- critical *or* essays *or* j *or* m *or* coetzee
- indigenous
- copy
- adler
- teen *and* center
- "functional genomics"~5
- ahmad *and* shah
- salt
- lyric *and* poet *and* era *and* high *and* capitali
- china *and* population
- ernest *and* hemingway
- "man apartheid"~20
- wynder
- sullum
- photo
- rich *and* man *and* poor
- habsburg

Proximity searches are shown above using the notation "word1 word2"~5. In this case, word1 and word2 must appear (in any order) within a window of 5 words.

First, each test was performed with a "cold" disk cache (nothing in the operating system cache.)

*Note: Once again, the Xapian chunked index numbers have been extrapolated.

"Cold" cache results:

	Lucene	Xapian	Zebra
Whole-doc index	3.245	51.766	14.519
Chunked index	6.619	*74.261	10.610

These results indicate that Lucene's on-disk structures are the most efficient to access.

Next, the tests were immediately performed again. I call this the "warm" cache test, since all the disk blocks needed by the queries were now in the operating system cache (and any cache maintained by the index system). Because the times measured are small, each test was actually performed 10 times and the results averaged.

"Warm" cache results:

	Lucene	Xapian	Zebra
Whole-doc index	0.056	0.892	0.999
Chunked index	0.251	*1.289	0.744

Interestingly, Lucene is significantly faster on these basic queries, despite being written in Java. One could speculate that Zebra's client/server architecture and fancy transaction processing and rollback mechanisms could be slowing it down. It's hard to explain why Xapian would be so slow.

Win: Lucene

3.4 Pathological query speed

Also of interest is how the engines perform in the case of "pathological" queries, designed to be time-consuming to process. I determined the most commonly used words in the document set (not including the default set of stop-words) so that the engines would have to iterate through many entries to compute the query results. These test the outer limits rather than "typical" cases.

Here are the five queries used:

- from *or* i *or* were
- "he an his"~6
- "had have which"~20
- she *or* one *or* those
- from *or* i *or* were *or* he *or* an *or* his *or* had *or* have *or* which *or* one
- from *and* i *and* were *and* he *and* an *and* his *and* had *and* have *and* which *and* one

*Note: Once again, the Xapian chunked index numbers have been extrapolated.

"Cold" cache results:

	Lucene	Xapian	Zebra
Whole-doc index	3.245	2.302	19.086
Chunked index	10.155	*144.701	12.976

"Warm" cache results:

	Lucene	Xapian	Zebra
Whole-doc index	1.316	0.393	17.125
Chunked index	4.334	*23.315	10.557

Xapian performs so well on the whole-document queries that one wonders if the extrapolated chunked index times are completely off. Perhaps Xapian would have excellent overall query performance, if only it could index the chunked documents in a reasonable amount of time. We may never know.

Also, we see again that Lucene is much faster than Zebra, probably indicating some inefficiency in Zebra's algorithms or architecture.

Win: Lucene

4. CONCLUSIONS

Unfortunately, Xapian does not appear ready for prime-time yet. Its query speed shows promise, and the query expansion feature is interesting, but clearly the engine needs more work before it could be used here.

Based on the results above, it's clear that either Lucene or Zebra would be a reasonable foundation upon which to build the Cross-Instance Search System. A quick summary follows.

Factors in favor of Lucene:

- Java
- Proximity ranking
- Query speed

Factors in favor of Zebra:

- Standards-based
- Index speed
- Index size

The final choice, of course, rests with CDL staff.