Comparison of indexers

Beagle, JIndex, metaTracker, Strigi

Michal Pryc, Xusheng Hou
Sun Microsystems Ltd., Ireland

November, 2006

Updated: December, 2006

Table of Contents

1. Introduction

There has been a great deal of interest in providing desktop search to users in recent years. You only have to look at Microsoft's Vista and Mac OS X to see how popular such features are in commercial desktop environments. The reason is obvious, with the increase in hard drive space on the users machines, the proliferation of email, web browsing, the use of on line chat and so on, desktop users are drowning in a sea of information. We need to support our users by providing them with powerful desktop search facilities, capable of indexing and searching all these information sources.

A number of search engines are available for the Gnome and KDE desktop environments, many based around the open source Lucene search engine. It would be tremendous if we could adopt one of these search engines for the Gnome platform, so we can provide the type of integrated search experience for our users that they really need, irrespective of which distort they are using. So to help in this assessment we have carried out an comparison of four different Unix based indexers. Some of the key features we are looking at assessing are:

Performance – indexes should be small, indexing itself should have minimal impact on CPU, memory footprint should be as small as possible, searching should be very fast.

Usable – search clients should be easy to use and intuitive, at the same time should allow powerful search.

Extensible – must be able to easily add in new information sources and document filters to extend the indexer.

Shareable – should be possible to share indices if users want to search across multiple indices on a network, both local and remote.

Integrated– should support appropriate API's to allow it to be fully integrated into the desktop and key desktop applications.

Note: Some of the indexers we tested are in very active development and are changing rapidly. We have included an update section to reflect these changes.

2. Indexers

Index search tools, that were tested

- **Beagle** - version 0.2.7

  Homepage: http://beagle-project.org

  IRC: #dashboard on irc.gimp.org

  Mailing list: http://mail.gnome.org/mailman/listinfo/dashboard-hackers

  License: A mix of the X11/MIT License and the Apache License

- **JIndex** - Internally (Sun Microsystems) modified version 0.1

  Homepage: http://jindex.berlios.de,

  Modified version: is available on SWAN internal network:

  http://jdswiki.ireland.sun.com/twiki/bin/view/JDS/JIndexProject

  License: LGPL

- **Meta Tracker** - version 0.5.0 - CVS from 08 November 2006

  Homepage: http://www.gnome.org/projects/tracker/

  IRC: #tracker on irc.gnome.org

  Mailing list: http://mail.gnome.org/mailman/subscribe/tracker-list

  License: GPL

- **Strigi** - version 0.3.8

  Homepage: http://www.vandenoever.info/software/strigi

  IRC: #strigi on irc.freenode.net

  Wiki page: http://strigi.sf.net

  License: LGPL

  Mailing lists:

  - http://lists.sourceforge.net/mailman/listinfo/strigi-user
  - http://lists.sourceforge.net/mailman/listinfo/strigi-devel

3. Test environment

3.1 Machine

- Vendor/Model: **IBM / Thinkpad T23**

3.2 CPU

- Vendor: **INTEL**
- Model: **Intel(R) Pentium(R) III Mobile CPU 1133MHz**
- Core Clock: **1130.500 MHz**
- Motherboard vendor: **IBM**
- Mbd. model: **2647-8RU**
- Mbd. chipset: **Intel 830MP**
- Bus type / clock: **PCI / 133 MHz**
- Cache total: **512 KB**
- SMP (number of processors): **1**

3.3 RAM

- Total: **256 MB**
- Type: **SDRAM - non-ECC - 133MHz**

3.4 Disk

- Vendor/Model: **SAMSUNG / MP0603H 60.0 GB**
- Interface: **IDE / EIDE**
- Driver/Settings: **udma5**
- Timing cached reads: **864 MB in 2.00 seconds = 432 MB/sec**
- Timing buffered disk reads: **74 MB in 3.06 seconds = 24.18 MB/sec**

3.5 Kernel

- Version: **2.6.17-gentoo-r8**
- Swap size: **1028.16 MB**

3.6 GCC

- Version: **(Gentoo 4.1.1)**
- Options:

```
/var/tmp/portage/gcc-4.1.1/work/gcc-4.1.1/configure --prefix=/usr --bindir=/usr/i686-pc-linux-gnu/gcc\
-bin/4.1.1 --includedir=/usr/lib/gcc/i686-pc-linux-gnu/4.1.1/include --datadir=/usr/share/gcc-data/i686-pc-\
linux-gnu/4.1.1 --mandir=/usr/share/gcc-data/i686-pc-linux-gnu/4.1.1/man --infodir=/usr/share/gcc-data/i686\
-pc-linux-gnu/4.1.1/info --with-gxx-include-dir=/usr/lib/gcc/i686-pc-linux-gnu/4.1.1/include/g++-v4 --host=\
i686-pc-linux-gnu --build=i686-pc-linux-gnu --disable-altivec --enable-nls --without-included-gettext --with\
-system-zlib --disable-checking --disable-werror --disable-libunwind-exceptions --disable-multilib --disable\
-libmudflap --disable-libssp --disable-libgcj --enable-languages=c,c++,fortran --enable-shared --enable\
-threads=posix --enable-__cxa_atexit --enable-clocale=gnu
```

3.7 libc

- Version: **2.4-r3**

4. Supported data sources

The table is based on the information taken from the project home pages and from the mailing lists. It shows, what can be indexed and which type of the data types are supported. The *comments* column, contains more precise information about data source.

Y = yes; **P** = in progress; **F** = planned in the next few months

data source	*comments*	*Beagle*	*JIndex*	*Tracker*	*Strigi*
Plain text		Y	Y	Y	Y
File system		Y			Y
Evolution	mail, calendar, and addressbook	Y		P	
Thunderbird	mail, news, RSS feeds, and addressbook	Y		P	
Gaim and Kopete	Instant messaging	Y	Y	F	Y
Firefox and Epiphany	web pages	Y		F	Y
Konqueror	web pages	Y			Y
Blam, Liferea and Akregator	RSS Feeds	Y			
Tomboy	notes	Y	Y	F	Y
KMail	mail	Y			Y
KNotes	notes	Y			Y
OpenOffice.org	documents, presentations, spreadsheets	Y		Y- (v1.4)	Y
OpenDocument	odt, ods, odp	Y			Y
Microsoft Office	doc, xls, ppt	Y		Y	Y
AbiWord	abw	Y		Y	Y
Rich Text Format	rtf	Y		Y	Y
PDF	pdf	Y	Y	Y	Y
HTML	xhtml, html, htm	Y	Y	Y	Y
Source code	C, C++, C#, Fortran, Java, JavaScript, Lisp, Matlab, Pascal, Perl, PHP, Python, Ruby, Scilab and Shell scripts	Y	Y (java only)	Y	Y
Texinfo		Y			Y
Man pages		Y			Y
Docbook		Y		F	Y
Monodoc		Y		F	Y
Windows help files	chm	Y			
Images	jpeg, png, bmp, tiff, gif	Y	Y	Y	Y (png only)
Audio	mp3, ogg, flac	Y	Y (mp3 only)	Y	Y
Video	mpeg, asf, wmv, mng, mp4, quicktime and other formats supported by mplayer	Y		Y	
Application launchers		Y	Y	F	Y
Linux packages	ebuild, rpm	Y		F	Y
Generic XSLT files		Y		Y	Y

Table 1: Supported data sources

The architecture of Beagle, Strigi and metaTracker allows one to use plug ins for additional sources. It would be great to see all of the indexers agree on a common plug-in API to allow plug-ins to be shared across the different indexers. This would allow the number of supported data sources to grow much more quickly for all the indexers. The first discussion on the common searching API over DBUS was started by the maintainer of the Strigi project[1], but maybe now is the time to start discussion on the common data source plugin API.

The Table 2 presents external libraries or programs that are used to index the data

data source	***Beagle***	***JIndex***	***Tracker***	***Strigi***
Emails	gmime-sharp			(attachments and nested files) using jstream
MS Word	wv1, optionally gsf-sharp		(ole2) based on code from libgsf	wvWare
MS Excel	ssindex - external program from gnumeric		(ole2) based on code from libgsf	
MS Powerpoint	gsf-sharp		(ole2) based on code from libgsf	
PDF	pdfinfo, pdftotex -external program from xpdf	PDFBox-0.7.2.jar itext-1.4.4.jar	With code from xpdf 0.93 www.foolabs.com/xpdf/	pdftotext Poppler-utils (based on libpoppler)
HTML	modified HtmlAgilityPack included in the Beagle source tree	htmlparser.jar itext-1.4.4.jar	Code from libhtmlparse 0.1.13 http://msalem.translator.cx/libhtmlparse.html	
Windows help files	chmlib			
Image files	custom code, mostly copied from F-Spot			
Audio files	entagged-sharp, included in the Beagle source tree	(mp3) sabercat.jar	(ogg) using ogg-vorbis	
Video files	MPlayer or Totem - external programs		(asp) based on code from xine (avi,mpeg) based on code from avinfo 1.0.0 alpha 11 and bitcollider 0.6.0	
RPM	rpm			
Compressed files			(tar, tar.gz,deb)Using zlib	libz, libbz2

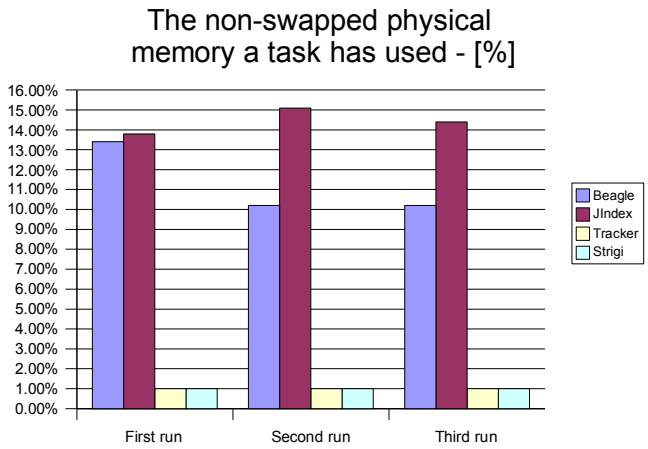
Table 2: External libraries used to index data

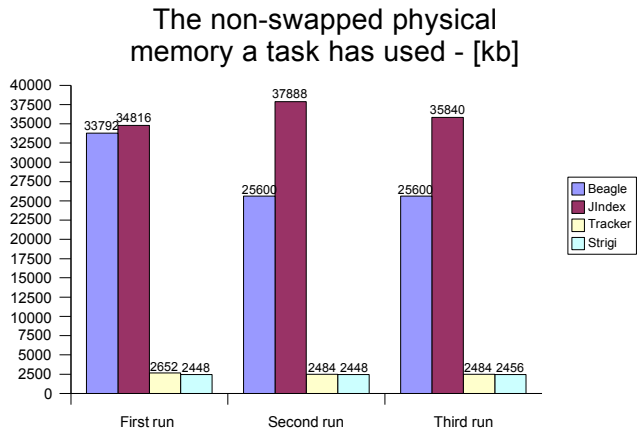
1 http://lists.kde.org/?l=kde-core-devel&m=116163130325537&w=2

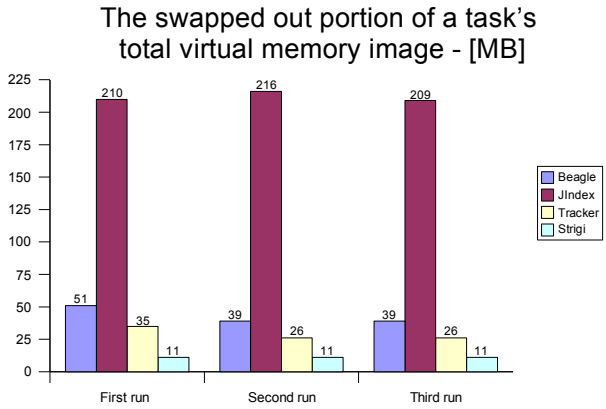
5. IDLE Daemon

5.1 Daemon memory

The Linux *top* command was used to produce the charts below. For each daemon, three measurements were made. First was the first start of the daemon after system startup, second was made just after stopping the daemon and third, after the second stop of the daemon.



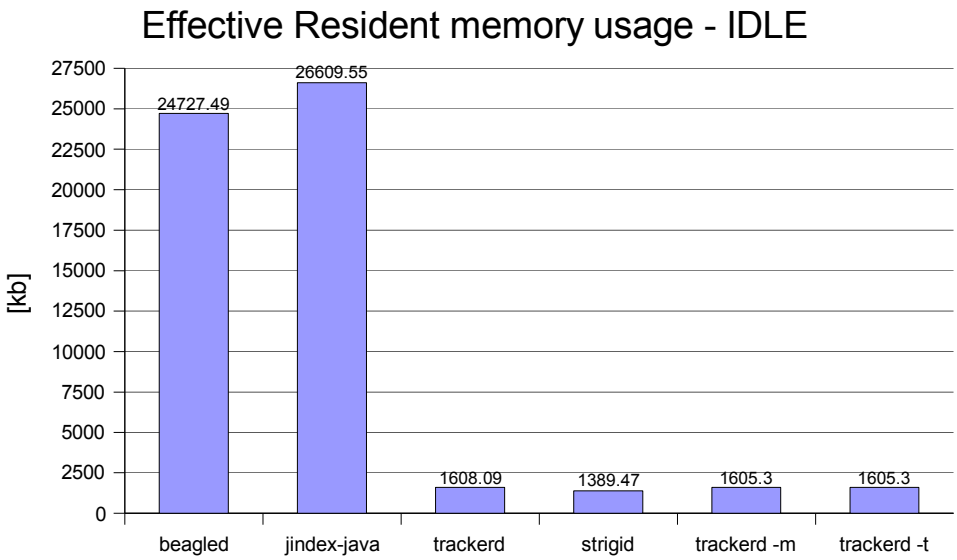




Those charts shows that Strigi uses the smallest amount of memory, quite the same is Tracker. JIndex is written in Java, so running JVM consumes a lot more memory, but Beagle which is running on top of Mono is not much better.

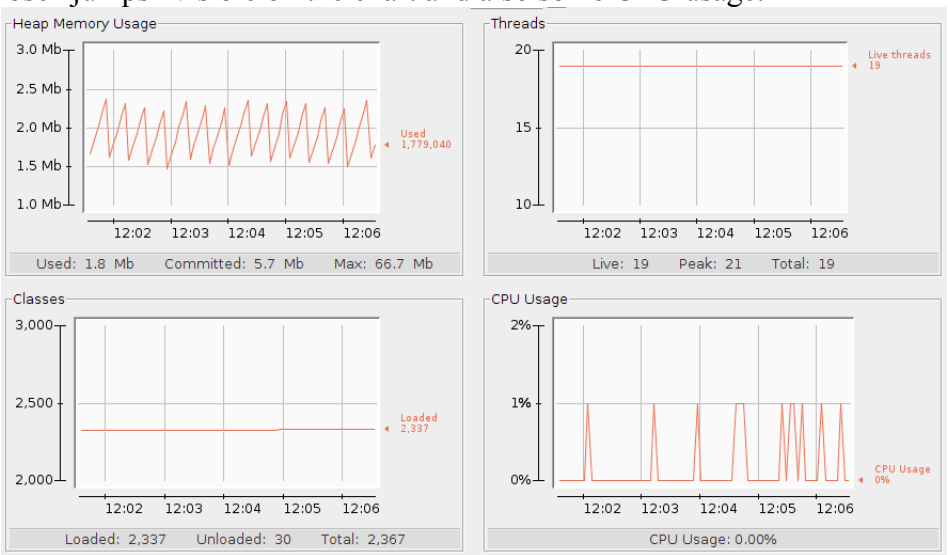
The *top* command is not very good for those types of comparison, cause it takes data from the /proc. Exmap is a tool that allows accurately determine how much physical memory is used by individual process and shared libraries. It counts number of processes that uses shared libraries and with this information can calculate the effective memory usage of the process. In the memory tests swap partition was not used. The chart below shows effective resident size of the processes, which means that this is the "corrected" version of the resident memory size.

We can see that Strigi(strigid) uses the smallest amount of physical memory. Tracker,which has three mode Normal mode(trackerd), Turbo mode(trackerd-t) and low-memory mode(trackerd-m), also use very small amount of the memory, but JIndex and Beagle are really consuming a lot of it.

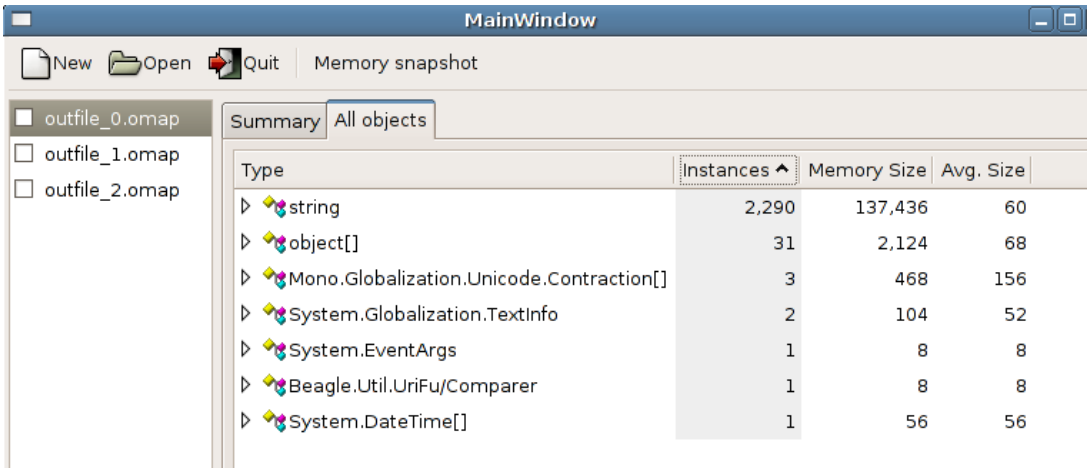


It is very hard to compare JIndex and Beagle with the other two indexers, because JIndex uses java virtual machine while Beagle uses the mono VM. This means that many system resources are taken just to create running environment for those daemons, however we will try to take shots from memory heap for both of them.

For Jindex, JConsole from Java Development Kit was used. To connect to the process we need to specify variable "-Dcom.sun.management.jmxremote". The results of the modified version are presented in the screen-shots below. We can see, that in Idle the daemon is using between 1.5Mb to 2.3Mb of the memory. It uses pooling method instead of FAM for getting information about the files, which causes those "jumps" visible on the chart and also some CPU usage.

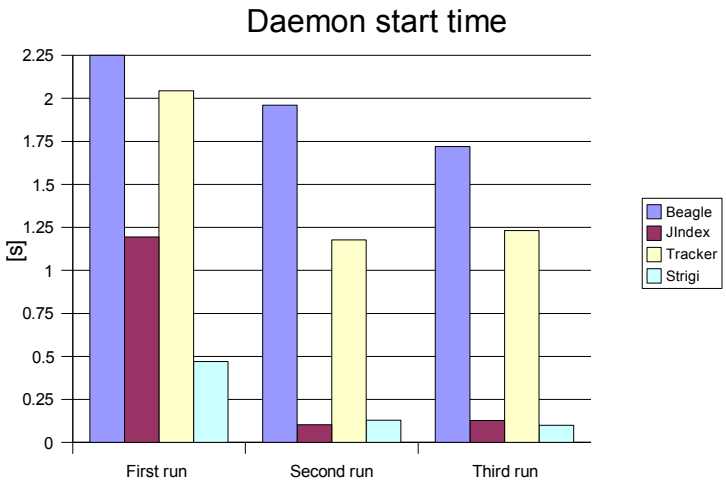


Tracking down memory for Beagle was done with heap-shot application. To use it we have to start mono application with heap-shot profile “mono --profile=heap-shot application.exe” and than voila, ready. We can see that for all objects around 0.137 Mb was used, which is less than JIndex, but of course we can not compare it like this, because for the user, system resources used by all processes required to run application are valuable, which means that we should look at the chart presenting Effective Resident memory usage.



5.2 Daemon startup times

To determine daemon start times, strace was used. Traced output was reviewed and calculations resulted in the chart below.



6. Indexing

This section shows, how indexers behave during indexing different type of the data and how they influence the system resources.

6.1 Data set

Table 3 lists the Data set used for testing. It consists of different types of files that all of the indexers can handle. The results and tests are described below.

	HTML	***JAVA***	***PNG***	***MP3***	***PDF***	***TXT***	***TOTAL***
Number of files	205	208	6	180	79	164	**842**
Size of the files	7.9M	1.9M	6.4M	761M	60M	2.8M	**834M**

Table 3: Data set for testing

Table 4 Shows the results for times and index sizes after indexing the entire data set.

	Beagle	***JIndex***	***Tracker***[2]	***Strigi***
Time of indexing [min:sec]	16:47	19:12	*Normal mode:* 12:37 *Turbo mode:* 9:34	5:18
Size of the index database	5.7M	6.1M	6.3M	6.3M

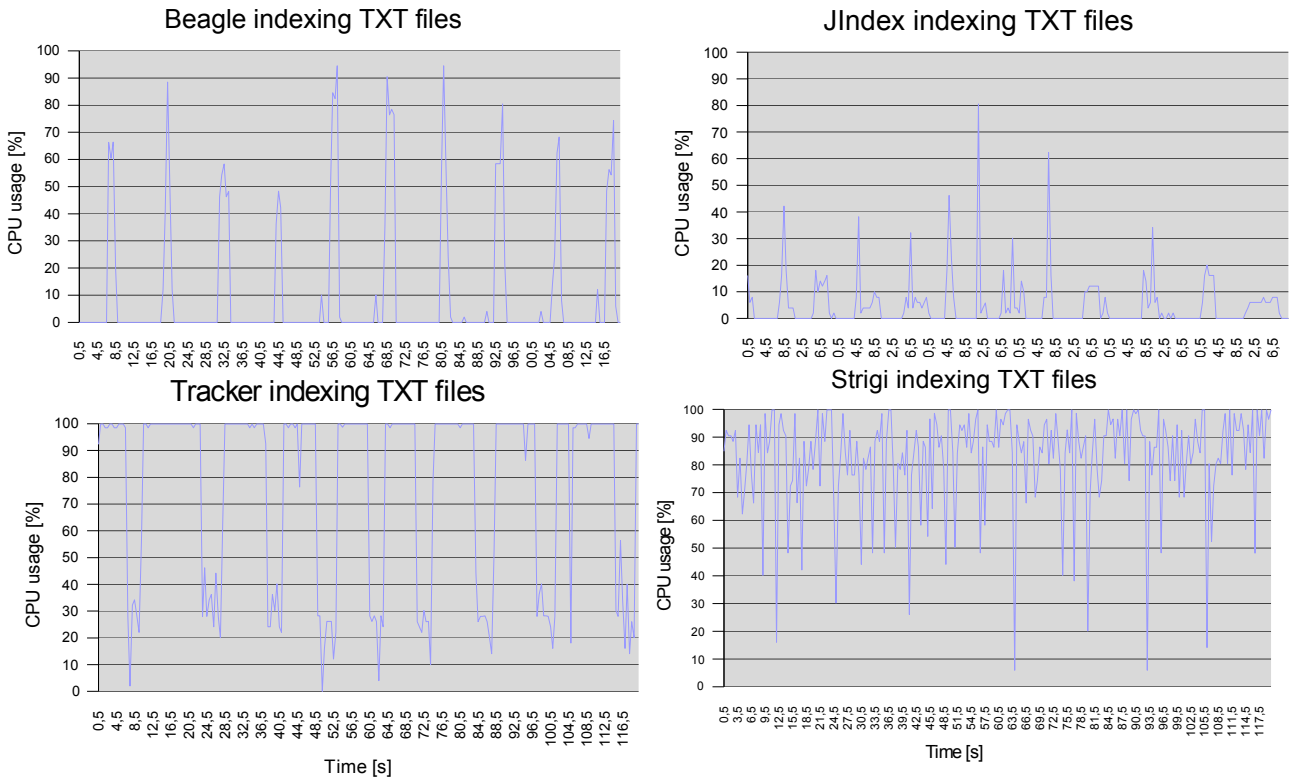
Table 4: Times and index size for after indexing all data set

6.2 Indexing TXT files

This test shows how the indexers indexed a set of 10,000 text files. For each indexer the same set of files was provided. A small script to measure average CPU usage was written. It is attached in the appendices section. Every 0.5 seconds it retrieved the current CPU usage for the running indexer. The list that was generated from the script was used to calculate average CPU usage. For Beagle both beagled and beagled-helper processes were measured, since both are running while indexing.

The charts below represent CPU usage over a period of 2 minutes, while daemons were indexing text files. We can see that every few seconds beagled-helper daemon was sleeping, Strigi performed the indexing much faster, but the average CPU usage was the highest. JIndex allows the user to perform other system tasks almost without noticing that the indexer is running, however it's memory usage is very high. Tracker can run in three different modes, allowing you to save some resources. Turbo mode really kills our CPU, while low memory mode does not seem to be much better in managing system resources than normal mode.

2 Tracker daemon allows to use “-t” or “-m” flag, which enables turbo or low memory mode.



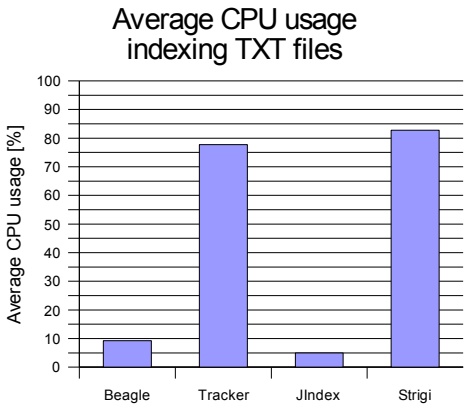
The size of the indexed database for text files differs for all indexers. Beagle produced the smallest database, 50% bigger was JIndex, Strigi 92% bigger and the biggest produced Tracker, which was almost 126% bigger than beagle database (Table 5).

	Beagle	***JIndex***	***Tracker***	***Strigi***
Number/size of TXT files	10 000 / 168MB	10 000 / 168MB	10 000 / 168MB	10 000 / 168MB
Size of the index database	62MB	93MB	140MB	119MB
Time of indexing [hr:min:sec]	02:18:05	03:02:55	03:03:14	00:04:26
CPU TIME [hr:min:sec]	00:12:05	00:09:15	02:22:40	00:03:44
Average CPU usage	8.79% (beagled process) 0.46% (beagled-helper process)	5%	77.73%	82.75%

Table 5: Indexing 10 000 text files

Average CPU usage, was calculated, while indexers were performing text indexing. Beagle uses two processes while indexing text files, so the average CPU usage is for both processes.

The CPU TIME, which is taken from the **top** command, shows total CPU time that indexer has used since it started. In fact CPU TIME should equals the time, which indexing process used for the task, multiplied by average CPU usage. It differs, but differences are very small, and might be caused by human factor. This tests shows that Tracker have some serious problems with text files, because CPU TIME was really huge comparing to other indexers (Table 5).

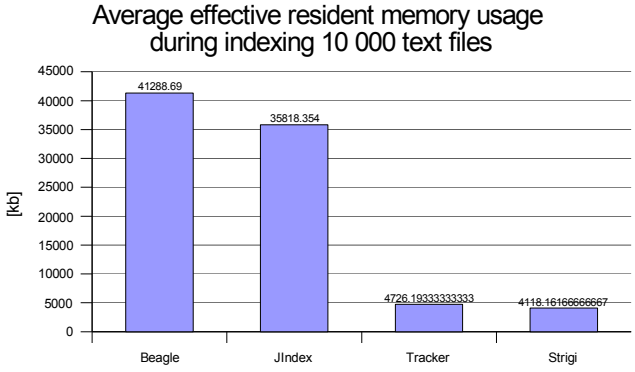


All tests for the Tracker were made using normal mode. However, the Tracker daemon can run in three different modes (normal, turbo, low-memory): turbo, which enables faster indexing that may degrade performance of the rest of the system; or low-memory which uses less memory, but results in slower indexing. Tests showed that the modes only affect flushing data to the inverted word index, which is made during indexing. The table below presents tests of these modes on a set of 500 text files (8.5MB). The size of the index database after indexing was the same for all modes, as one would expect (7MB).The turbo mode can index text files about 22% faster than the normal mode and 27% faster than low memory mode (Table 6). Of course nothing is for free and average CPU usage increase dramatically in turbo mode. This is more suitable for indexing files when the user is not using computer, such as over night.

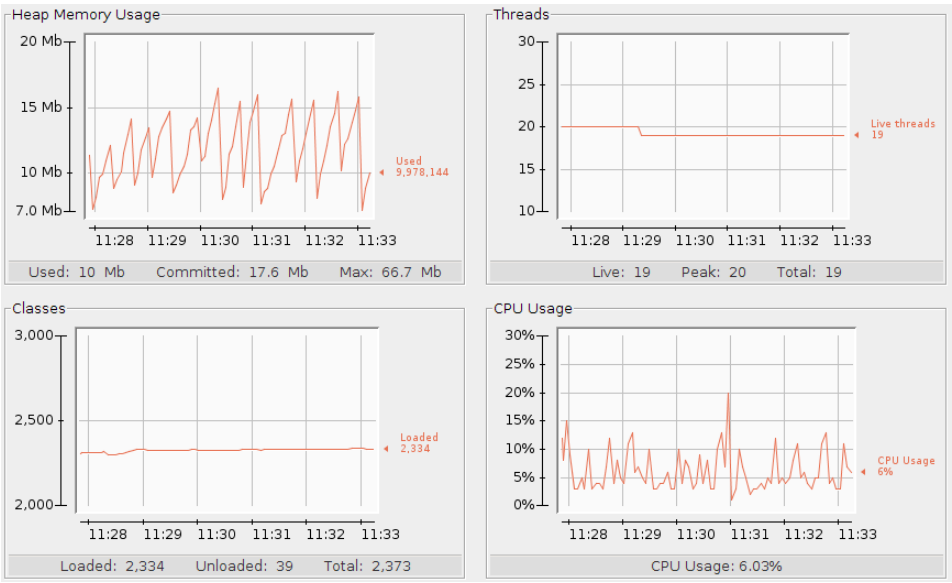
	trackerd	***trackerd -t (turbo)***	***trackerd -m (low mem)***
Time of indexing [min:sec]	8:35	6:42	9:08
CPU TIME [min:sec]	6:36	6:39	6:31
Average CPU usage	76,71%	99,38%	70,56%

Table 6: Three Tracker modes - indexing 500 text files

To see how indexers were using memory during indexing txt files, again exmap tool was used. From the 10 memory snapshots average was calculated. For Beagle, both processes beagled and beagle-helper were used for calculations.



JIndex is running in a JVM. To see what memory impact the JIndex program is having on memory, separated from the effect of having to run a JVM instance on the system, we used JConsole. The average Heap Memory size for JIndex alone [excluding the memory requirements for the JVM] is around 12MB, and it shows very low CPU usage.



6.3 CPU usage - indexing PDF

CPU usage is more difficult to measure here, because a lot of child processes like pdftotext, pdfinfo, are open. So Average CPU usage was calculated for each process and child running. The calculations were done by measuring each process for some period of time and then the percentage for each process calculated over this time period. The results, shows that Strigi is the fastest indexing PDF files and the resultant database index is quite small. However, does it contain the same amount of information that Tracker and Beagle? Section 7 of this document tries to get answers for this question.

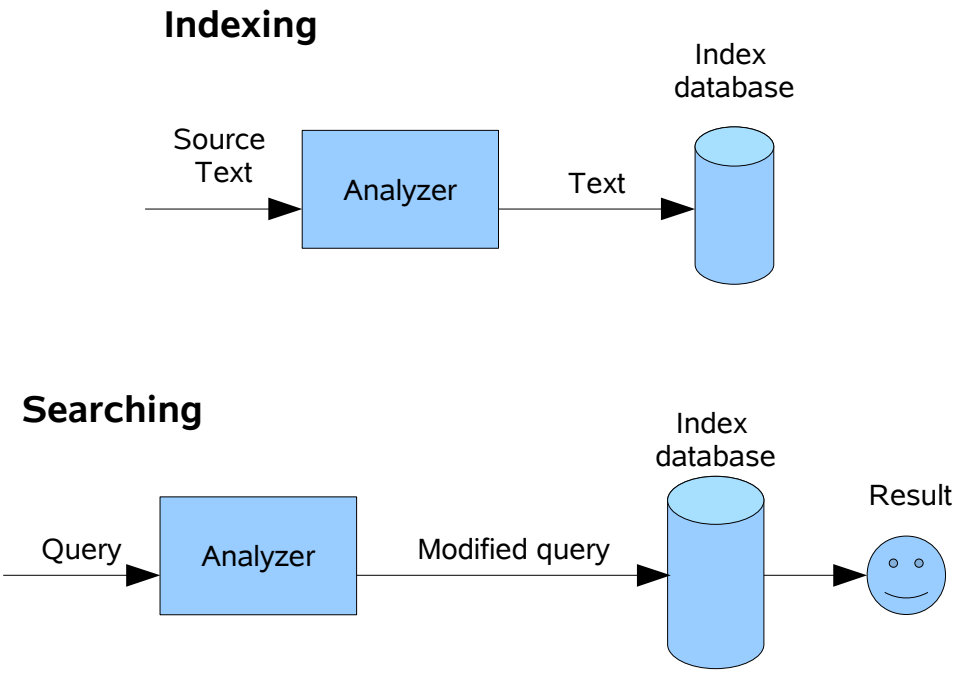
Again CPU usage is the smallest for JIndex (Table 7) and allows one to perform other tasks on the system at the same time. Tracker (88,06%+10,51%=98,57%) does not allow anything except indexing. We can see that effective resident memory usage is the best for tracker and the second best for Strigi which are much better than Beagle and JIndex. The process pdftotext uses almost the same effective resident memory for all processes, it is obvious, because it is the same application. The same is for the Average CPU usage per process for pdftotext, the differences are rather small and can be caused by some flag that the indexer is using while invoking pdftotext or just measurement inaccuracy.

	Beagle	***JIndex***	***Tracker***	***Strigi***
Number/size of PDF files	50 / 61MB	50 / 61MB	50 / 61MB	50 / 61MB
Size of the index database	1.6MB	0.8MB	2.6MB	1.2MB
Time of indexing [min:sec]	4:24	3:56	4:29	3:20
Average CPU usage per process	beagled **0.13%** beagled-helper **2.41%** pdftotext **85.77%** pdfinfo **0.47%**	JIndex (java) **70.02%**	pdftotext **88.06%** trackerd **10.51%**	pdftotext **91.11%** strigid **1.95%**
Effective resident memory usage per process	beagled-helper **20969.28 kb** beagled **19419.80 kb** pdftotext **2940.03 kb**	JIndex (java) **55245.35 kb**	trackerd **2214.65 kb** pdftotext **2954.12 kb**	strigid **4086.3 kb** pdftotext **2934.33 kb**

Table 7: Indexing PDF files

7. Searching

The result from the search is not simple comparison of the strings. The way of indexing and querying the index database affect the result. Two drawings below, presents simplified way of indexing and searching using the analyzers. In the indexing process those analyzers are responsible for parsing and tokenizing the input text. During searching analyzer is parsing the query and modifying it to allow better result. This chapter will compare indexers regarding to different queries and searching algorithms.



7.1 Terms

Term queries consists of a single word or a single phase. Those type of queries are the typical for daily use, and should give the best results. As a default Beagle and Tracker are using stemming algorithms during search, which basically means that morphological variants of the terms are searched[3]. It reduces the words for their stem form of the word – the base.

Beagle have a results limit of 100 which can not be overridden by any flag. So all the queries were specified to gives results < 100. The table presents the number of files found by each client.

Query	*Beagle*	*JIndex*	*Tracker*	*Strigi*
A Single Term in quote: “bug”	7	8	6	1
A Single Term with big letter in quote: “Bug”	7	8	6	0
A Phrase in quote: “big bug”	1	1	3	0
A Phrase with big letters: “Big bug”	1	1	3	0
A Phase without quote: big bug	3	14	3	1
A Single Term in quote: “acknowledge”	9	8	9	1

Table 8: Sample terms query results - number results found [test set contained 164 text files].

So where those differences comes from? After some research it was found that Strigi does not have encoding detection and it only indexes files with UTF-8 character encoding.

JIndex had the best results searching for a single term in quotes, it found one file more than Beagle, that contained an embedded word in a line with different characters “ BUG ”.

The Tracker does not support some encoding like Chinese one, that is why there is difference in search results. For the Tracker searching phrases in quotes gives more results, because it uses stemming as the default dropping quotes, so asking for “big bug” will search for stemmed terms “big” AND stemmed term “bug”. Tracker will not search for exact phrases such as “big bug”, while it simply does not care about quotes in queries.

After searching for the “acknowledge” term, Beagle and Tracker found the same set of the files, both are using stemming and both found the file containing “acknowledgment”. What about exact phrases in quotes?? We attempted to have a precise query for an exact phrase, as indicated by quoting the phrase, but got back results that should only happen with a query without quotes. So Beagle like Tracker does not support exact phrase searches.

JIndex found exactly those files that contain the “acknowledge” term. As JIndex does not have stemming implemented it did not finding variants such as “acknowledgment”. JIndex does have some other algorithms for fuzzy search, but more about this later.

So now time for a handicap race! To even the odds we will index 100 text files, with a UTF-8 encoding

3 More about steeming: http://en.wikipedia.org/wiki/Stemming

that all the indexers including Strigi can handle. No doubt Strigi will have encoding detection in the next release, but for now lets simplify things a little.

Query	*Beagle*	*JIndex*	*Tracker*	*Strigi*
A Single Term in quote: “quick”	8	8	8	6
A Single Term in quote: “introduction”	13	12	13	11
A Single Term with big letter in quote: “Quick”	8	8	8	0
A Phrase in quote: “quick introduction”	1	1	1	0
A Phrase in quote: “introduction quick”	0	0	1	0
A Phrase with big letters: “Quick Introduction”	1	1	1	0
A Phase without quote: quick introduction	1	19	1	1
A Phase without quote: introduction quick	1	19	1	1
A Single Term in quote: “acknowledge”	11	3	11	2

Table 9: Sample terms query results - number results found [test set that all the indexers can handle].

So now we have really confusing set of results. After investigation it was found that Strigi has problems with indexing files with the terms “quick...” or “introduction.”, so a dot at the end of the word gives two results less for each query. Again Beagle and Tracker used word stemming to find exactly one more file (does Tracker uses the same algorithm that Beagle does??) than JIndex. The word in question was slightly misspelled which is awesome. Strigi does not like big letters in quotes while searching for two or more terms. Tracker simply ignores quotes that is why it gets a hit for “introduction quick”, with and without quotes, whereas none of the other indexers do. It is searching for “introduction” AND “quick”, irrespective of word order. For phrases without quotes JIndex got result 19, because it places OR instead of AND between terms which is different from the other search clients. Looking at the JIndex results one could say. “Hang on 8+12 != 19 or I had better go to school again!”, that is correct, but one file contains both words “quick” AND “introduction” and everyone knows that $f(A \vee B) = f(A) + f(B) - f(A \wedge B)$, which gives our combined result of 19 instead of 20.

7.1.1 Searching fields

So searching terms is a piece of cake (yes, I do like cakes), but what about searching for terms and metadata fields? Where metadata fields contain information about the file being indexed rather than the content terms, such as date of creation, file type and so on. For example say we want to search for all the text files that were indexed. We can do so using the Unix *find* command of course, but with a lot of data this can be rather time consuming. Searching metadata fields gives us an opportunity to do it in just a few seconds!

All clients have different syntax for searching these metadata fields, Unfortunately there is no standardized syntax nor do they have good documentation to explain how to search with these fields.

Beagle has a small howto on the web page, JIndex uses the Lucene syntax (this is documented), tracker is able to use RDF scripts, but again with no standardized name convention, and Strigi is using something that is similar to CLucene. Strigi and Tracker developers are talking about common query syntax standardization which would be great[4]!

To make it a little bit more complicated for the indexers, we carried out our test on 100 text files that had extensions some of which contained small and big letters. The command: **find ./ -name "*.[tT][xX][tT]"|wc -l** printed the result of 100, which should be the same for indexers. The table below shows the number of documents found, and below that the number query syntax that was used to get those results, so let's try to find all text files:

Beagle	***JIndex***	***Tracker***	***Strigi***
Result: **100** *beagle-query command:* beagle-query --mime text/plain	Result:**100** filemimetype:"text/plain"	Result:**100** *RDF FILE:* *<rdfq:Condition>* *<rdfq:inSet>* *<rdfq:Property* *name="File.Format" />* *<rdf:String>text/plain</rdf:String>* *</rdfq:inSet>* *</rdfq:Condition>*	Result:**87** mimetype:text/plain

Table 10: Sample fields query results , looking for text files in a test set of 100 text files.

Does 13 means bad luck for Strigi? Where are the missing 13 text/plain files, that were indexed? Strigi uses the Unix *file* command to check the mime type, so let's use the command:

for f in `ls` ; do file -i $f ; done

and see what happens. So, *file* checked all the files, and returned results of text/plain, image/x-3ds, text/x-c and text/x-java. The strange thing is that files with mime type image/x-3ds contained ASCII drawings, text/x-java in fact contained some source inside, etc. etc. So, after counting all text/plain files the result of 89 appeared. This is a little different than 87, but I will not go deeper in this problem. The Strigi developers should do something about it.

With regard to Tracker, result was good, but which end user would want to learn and write scripts for searching different types of files? I don't. But it can be a very powerful advanced feature. If they could wrap up canned field queries in a suitable GUI and allow users to write scripts if they wanted more complex queries then they could have the best of both worlds.

Other tests were performed using different fields. It showed up some issues with mp3 field data. Only JIndex could successfully perform searching for mp3 files through the song title field. Although Tracker have the fields in place, there was no data from mp3 file tags in the database. The same was true for Beagle and Strigi. These filters should be corrected for the various indexers. Also none of the indexers

4 http://www.mail-archive.com/xdg@lists.freedesktop.org/msg01765.html

could find pdf file through the title field.

Strigi's field searches are not case sensitive, which they should be. So searching for filename:*3 will give the same result as searching for filename:*MP3 or filename:*mp3 or filename:*mP3 or filename:*Mp3

This shows that Strigi distinguish case sensitive search through the terms, so why not through the fields?

7.1.2 Term modifiers

Term modifiers allows to make more flexible queries. If we don't quite remember the exact term then we might use one of these modifiers. In the computer world there is a standard regular expression syntax[5], which should be supported by the search clients. We will check if this is supported by the indexers.

The star "*", means that we want to search for the term that contains zero or multiple "unknown" characters. The query "old *man" should give results for the files that contain both "old man" and "old woman". Question mark "?" in the query replaces only one character, so you can use "te?t" to search for the "text" or "test".

Fuzzy search is more complicated. Only JIndex supports it and the search is based on the Levenshtein Distance, or Edit Distance algorithm[6]. So you need to add "~" at the end of the searching term. Searching returns results that might be relevant, even if the searched term does not appear anywhere in the text. So searching for "band~" will also find "sand", "sands" and other relevant words. Fuzzy search finds one or more substrings of a term or similar terms. On the other hand, stemming results are quite similar, but uses different types of algorithms and the concept is slightly different. It reduces word to the base form "stem" and then searches for it. There are many stemming algorithms like Paice/Husk, Porter, Lovins, Dawson or Krovetz, for more information please refer to the references.

Beagle and Tracker have support for the operator "**TO**". Beagle supports only date search, for Tracker we can write scripts that will allow this type of search. Strigi developers are working to support it.

Boosting, which is supported only by JIndex is important when query contains more than one terms. It is used to tell the search engine which term is more relevant. It uses "^" in the searching terms.

Query	***Beagle***	***JIndex***	***Tracker search client***	***Tracker RDF scripts***	***Strigi***
Support for: *	NO[7]	YES (* can not be at the beginning)	NO	NO	YES

5 http://en.wikipedia.org/wiki/Regular_expression
6 http://en.wikipedia.org/wiki/Levenshtein_distance
7 Beagle behaves a little strangely, when asking for "te*s" some files were found and some not, from 130 files which Strigi and JIndex found, Beagle found only 9. The same problem is with the "?" modifier.

Query	*Beagle*	*JIndex*	*Tracker search client*	*Tracker RDF scripts*	*Strigi*
Support for: **?**	NO[7]	YES	NO	NO	YES
Fuzzy search: ~	NO	YES	NO	NO	NO
Stemming search	YES (by default)	NO	YES (by default)	YES	NO
Support for: **TO** date:[20060101 TO 20060201] [anna TO annb]	date only (--start <date> --end <date>)	NO	NO	YES	NO
Support for boosting: ^ ski^5 italy	NO	YES	NO	NO	NO

Table 11: Support for query term modifiers

7.1.3 Boolean operators

If we want to go skiing and enjoy plenty of apes ski Guinness, we'll need to use Boolean operators or we'll get lots of hits for Ireland, which is a little short for snow! So, we can find all the documents that must have both “ski” AND “Guinness” terms. Or we might be happy with drinking Guinness if there's no skiing so we could use “ski” OR “Guinness”. Operator “+” tells us that this term must exists somewhere in the text as in the example where we know that we want go skiing to Italy and drink Guinness, we could use: +ski +Italy +Guinness. These operators help to give much more precise results. Each indexer supports at several boolean operators. By default, all indexers except JIndex are using AND between terms while JIndex uses OR which means that a query for *ski france* will give the same results as *ski OR france*.

Query	*Beagle*	*JIndex*	*Tracker search client*	*Tracker RDF scripts*	*Strigi*
Support for: **AND** (ski AND switzerland)	YES (by default)	YES	YES (by default)	YES (rdfq:and)	YES (by default)
Support for: **OR** (ski OR france)	YES	YES (by default)	NO	YES (rdfq:or)	NO
Support for: **NOT/-** (ski NOT denmark) (ski -ireland)	YES	YES	NO	NO	YES
Support for: + (ski +rent +italy)	YES	YES	NO	NO	NO

Table 12: Support for Boolean query operators

7.1.4 Grouping, Field Grouping and Term order

Sub queries are useful when we want to control many Boolean operators in a single query and are supported by JIndex. Tracker also can group fields, but only when using RDF queries.

Term order is only supported by JIndex. It means that first term in the query is the most important and it

sorts results in proper order, which is very nice feature.

Query	*Beagle*	*JIndex*	*Tracker search client*	*Tracker RDF scripts*	*Strigi*
Parentheses for expression grouping (spain AND party) OR siesta	NO	YES	NO	YES	NO
Parentheses for fields grouping filename:(png OR mp3)	NO	YES	NO	YES	NO
Term order	NO[8]	YES	NO	NO	NO

Table 13: Support for query grouping and query field grouping

7.1.5 Special characters

During the indexing and searching, applications are using analyzers to transform the query. Those transforms might use stop words and omit them. This simply means that the query "the big sandwich" will be transformed to "big sandwich". Those stop words might differ and it is important to use the same set of stop words for indexing and searching, otherwise it will be not possible to get proper result. All the indexers are using analyzers to support stop words. On the other hand none of the indexers is able to search through the stop word, so queries like "to", "or", "and" does not gives any results.

Sometimes we may want to search for certain special characters, such as those in a math equation, then the only choice is JIndex, which allows you to search for special characters.

Query	*Beagle*	*JIndex*	*Tracker search client*	*Tracker RDF scripts*	*Strigi*
Stop words: **a, the, and, or**	NO	NO	NO	NO	NO
Special characters: **-+ && || ! ( ) [ ] {} ^ ~ " * ? : **	NO	with the \ before the character	NO	NO	NO

Table 14: Support for special characters in the query

8 Beagle client allows to sort results by relevance, name or modification date

8. Install, build and support

8.1 Building & Installing

	Beagle	*JIndex*	*Tracker*	*Strigi*
Language	C#, which runs on top of the Mono frame-work	Java	C	C++
Requirements	- Mono >= 1.1.13.5 - gtk-sharp2 >= 2.4.0 - gmime >= 2.2.0 - Sqlite with mono-data-sqlite - libexif >= 0.5 - X header files (scrnsaver.h) - zip for mozilla extension - glib >= 2.6 and libxml >= 2.6.19 for libbeagle (C bindings) - libgeale and pygtk2 >= 2.6 for pybeagle (Python bindings)	- Java >= 1.5.0 - Java-Gnome bindings - Provided with JIndex jar libraries: commons-codec-1.3.jar commons-lang-2.1.jar db-4.3.jar gnumail.jar htmlparser.jar itext-1.4.4.jar jdic.jar jxl.jar log4j-1.2.13.jar lucene-1.4.3.jar PDFBox-0.7.2.jar sabercat.jar xalan.jar xstream-1.2.jar	- Sqlite >= 3.2 - libdbus >= 0.50 - dbus-glib bndings >= 0.50 - glib >= 2.6 - zlib - GMime	- CLucene >= 0.9.15 - CMake >= 2.4.2 - ZLib >= 1.2.3 - BZip2 >= 1.0.3 - OpenSSL
Optional	- kernel >= 2.6.13 (for inotify support and extended attributes) - evolution-sharp >= 0.9 - galago-sharp 0.5 - wv1 1.2.0 - pdfinfo - ssindex - gsf-sharp >= 0.6 - Firefox - MPlayer		- wv >= 1.0.2 - poppler (pdftotext) - libvorbis - GTK and Gnome stack for GUI tools	- Qt4 >= 4.1.2 - libxml2 - magic-dev - kernel >= 2.6.13 (for inotify support) - log4cxx >= 0.9.7 - Xerces-C >= 2.6.0
Supported database backends	- DotLucene	- Lucene	- Sqlite3	- CLucene - Hyper Estraier - Sqlite3 - Xapian

Table 15: Building & Installing indexers

8.2 Documentation

Documentation is very important for ordinary users as well for developers. Every programmer knows that it is painful, but few hours spend on documenting, can save many hours of others. During these indexer tests, especially when trying to find proper syntax, the author had no other option but to look into the source code. And we are not talking about having to write the book "Query syntax for dummies", because in many cases there was no documentation at all! Beagle seems to have the best on line help, which is not perfect. It does not describe how to write more complex queries nor does it give

you a list of possible keywords to use for example. Tracker uses RDF queries. There are five examples in the source code, but they are not explained enough though they do have a very good and active mailing list. JIndex uses standard Lucene queries, which were the most “user friendly” and are documented, but there was no help documentation at all from the JIndex side. Strigi, which uses Clucene as an database backend, does not support all of the Clucene query features, what was a bit confusing. The Strigi community uses mailing lists as well active IRC channels, where we could find a lot of answers, but some users don't want to be on IRC or on mailing lists just to find out how to use an application. You can imagine asking your granny to drop onto an IRC channel to find out how to search for her Christmas shopping list that she wrote on her home PC months ago, no I don't think so.

	Beagle	***JIndex***	***Tracker***	***Strigi***
Manual	Good	-	Average, some basic things explained	-
--help	Good	-	Good	-
Explanation of the query syntax	“How to search data” The Beagle homepage - average.	Lucene query syntax	RDF query syntax Complex for normal users, not explained enough	Poor – only in the source code

Table 16: Documentation

9. Summary

After reviewing those projects it is hard to say which is the best choice. If we would take all the bests from those indexers and put into one, then we would have great application. In this chapter we will try to summarize and point the strengthens and weakness of those indexers, which might help developers to improve indexers. The really good news are, that developers from two communities (strigi, metaTracker) started thinking on the common search API, which will rise the number of search clients, but a common plug-ins API would really speed up the developing process and functionality.

Standardized queries is another thing, that should take place. Most of the users wouldn't like to learn new query syntax just for the indexer, so why not to go for something that is widely used, something similar to google? It is good time for the developers to think about it and made the best choice having users in theirs mind.

The memory race, causing indexers to use as less memory as possible is quite exciting, but on the other hand we've got CPU usage, which is also very important. Brand new computers have lot's of memory, so if we want to do something else than just indexing, and we are not memory constrained, JIndex would seem to do the best job for us, because it have very good CPU resources usage.

All the indexers produced databases, with acceptable size and we should not put an effort to change it so far. At the date of updating this documentation, all indexers except Beagle were successfully compiled

on the SunOS 5.11 operating system with the forte compiler.

9.1 Beagle

Beagle is very mature application, which have the greatest amount of supported data types. Developers have also easier live, because of the clear documentation and the online API docs.

- ✔ Very mature project
- ✔ Documentation
- ✔ Amount of supported data types
- ✔ IMAP support (unfortunately only, when e-mail client have offline viewing switched on)
- ✔ Clear GUI client
- ✔ Stemming search algorithm

- ✗ CPU usage (for the PDF documents)
- ✗ Memory usage
- ✗ Limit for search results, that can not be overridden
- ✗ Programming language (C# that runs on top of the Mono VM)

9.2 JIndex

This is quite new project, that uses Java as a programming language, with a little of C native code, which is used with JNI. A lot of the things need to be changed in this project, but it shows how the CPU resources should be used, as well query was the most similar to the google.

- ✔ Query standards
- ✔ Query operators, term modifiers and fuzzy search
- ✔ CPU usage
- ✔ Clear GUI client

- ✗ Memory usage
- ✗ Pooling instead of notification system
- ✗ Threading problem

- ✗ No community and active developers

9.3 Tracker

The Tracker is not just indexer, as Tracker developers says, that it is metadata database and indexer framework, which have indexing and searching facilities. With the RDF queries it might be a powerful tool, but lack of documentation and standards neither good GUI, makes this tool hard to use by the users.

- ✔ RDF search scripts
- ✔ Stemming search algorithm
- ✔ Memory usage
- ✔ Common search API with Strigi in the future
- ✔ Manual pages / help

- ✗ CPU usage
- ✗ Problems indexing lot's of text files
- ✗ Query operators and term modifiers
- ✗ Not well documented RDF queries
- ✗ GUI without advanced search
- ✗ Tracker deleted all the *.java and *.txt files while indexing, because they were in the /tmp folder.
- ✗ Troubles with queries like “simple text” - it finds all the text files.

9.4 Strigi

All the indexers, except strigi are designed to index files, not streams. The new concept that uses Strigi allows to index data other than just the local files, which might be valuable in the future for example when indexing IMAP folders. Strigi comes also with two nice tools deepfind and deepgrep, that are tuned version of the find and grep commands. They allows to find the files, which contains specified text in the data sources supported by strigi (e.g. compressed sources).

- ✔ Uses streams not files as a data source
- ✔ Memory usage
- ✔ Time of indexing

- ✔ Calculating sha1sum for the files (finding duplicates on the system)
- ✔ Common API with Tracker in the future
- ✔ Indexing compressed files that contains other compressed files
- ✔ Includes deepgrep and deepfind tools

- ✗ CPU usage (for the PDF documents)
- ✗ CMake make system
- ✗ Zombie pdftotext processes after indexing
- ✗ Problems with some search queries
- ✗ Problems with small and big letters while searching
- ✗ Query standards
- ✗ Deletes database
- ✗ Fields standards
- ✗ Problems with some data types (mp3tags)
- ✗ Documentation
- ✗ Not clear ANSI C / POSIX source code
- ✗ Not well designed GUI

10. Update

Those projects are under the heavy development, which means that during writing this document, many things might change. Strigi is one of those that should be mentioned in this chapter. In the new release (0.3.11) the encoding problem seems to be solved, documentation is also slightly better and what is the most important, the patches were applied to allow building it on the Solaris platform with forte compiler.

11. Appendixes

11.1 CPU usage script

Example of usage: ***$ ./cpu.sh 46512 0.5 /tmp/output.cpu***

```
#!/bin/bash
#script for generating current cpu usage times for specified pid
if [ "$1" = --help ]; then
```

```
        echo Script for generating cpu usage for process
        echo
        echo $0 PID time filename
        echo
        exit 0
elif [ -z "$1" ] || [ -z "$2" ] || [ -z "$3" ]; then
        echo $0: missing argument
        echo Try \'$0 --help\' for more information.
        exit 1
elif [ `ps --pid $1|wc -l` -lt 2 ]; then
        echo There is no process with PID $1
        exit 1
fi
echo
echo Started, to stop, hit Ctrl^C
while :
 do
  top -b -n 1 -p $1|cat -|grep $1|cut -c 41-46 >> $3
  sleep $2
 done
```

12. References

Checked on the 3rd of the January 2007

Searchable data sources:

Mail, calendar and address book:
Evolution http://gnome.org/projects/evolution
Thunderbird http://www.mozilla.com/thunderbird
KMail http://kmail.kde.org

Instant messaging (IM):
Gaim http://gaim.sourceforge.net
Kopete http://kopete.kde.org

Web browser:
Epiphany http://www.gnome.org/projects/epiphany
Konqueror http://www.konqueror.org
Firefox http://www.mozilla.com/en-US/firefox/

News feeds:
Blam http://www.imendio.com/projects/blam
Liferea http://liferea.sourceforge.net
Akregator http://akregator.kde.org/

Note-taking:
Tomboy http://www.beatniksoftware.com/tomboy
KNotes http://kontact.kde.org/components.php#notes

Office suite:
OpenOffice.org http://www.openoffice.org
AbiWord http://www.abisource.com

Libraries:

PDF:
Xpdf http://www.foolabs.com/xpdf/about.html
PDFBox http://www.pdfbox.org/
itext http://www.lowagie.com/iText/
Poppler http://conference2005.kde.org/slides/poppler/index.html

E-mail:
gmime-sharp http://rpm.pbone.net/index.php3/stat/4/idpl/3509129/com/gmime-sharp-2.1.16-1mdk.noarch.rpm.html

Office suite:

wvWare http://wvware.sourceforge.net/
libgsf http://directory.fsf.org/All/libgsf.html

HTML:
htmlparser http://htmlparser.sourceforge.net/
HtmlAgilityPack http://www.codeplex.com/htmlagilitypack
libhtmlparse http://www.linux.org/apps/AppId_7266.html

Windows help files:
chmlib http://www.jedrea.com/chmlib/

Audio files:
ogg-vorbis http://www.vorbis.com/

Video files
xine http://xinehq.de/index.php/about

Compressed files
zlib http://www.zlib.net/

Xml
Xalan http://xml.apache.org/xalan-j/

Tools:

Exmap--A memory analysis tool to measure memory usage of processes and libraries:
http://www.berthels.co.uk/exmap/

heap-shot-- A memory profiler
http://primates.ximian.com/~lluis/blog/

Other:

Information Retrieval (IR) using stemming algorithms to reduce the word to its stem:
http://www.comp.lancs.ac.uk/computing/research/stemming/general/