Indexing Arbitrary Data with SWISH-E

Josh Rabinowitz • joshr@joshr.com • *SkateboardDirectory.com*

From The Proceedings Of The 2004 USENIX Technical Conference

Abstract

Fast lookups are crucial to many computer applications and operations. The general problem of indexing and searching on arbitrary data is not a simple one, with many semantic, linguistic, and technical issues to iron out. In this paper we present swish-e, a descendent of Kevin Hughes' SWISH project from 1994. Swish-e provides a full-featured and useful toolkit to index and query 8-bit ASCII data. This paper discusses the structure, features, and usage of swish-e, with mentions of possible directions for further development and interesting related work. We also compare swish-e to MySQL's full-text search feature in terms of features and speed, and discuss two real-world swish-e applications, Sman and Swished.

1. Introduction

This paper discusses the features and limitations of swish-e[1], and to a lesser extent, MySQL's fulltext search feature[2]. This paper loosely builds on information presented in the Author's article in Linux Journal entitled "How To Index Anything"[3]. We at SkateboardDirectory.com discovered swish-e when researching indexing toolkits and were attracted by its feature set, perl interface, quality documentation, and lively and informative discussion list.

2 SWISH-E Overview

The three most common data storage techniques (flat files, Berkeley DB[4]-like binary files, and SQL databases) each give rise to their own particular data search and retrieval features, strengths and weaknesses. While each technique allows some form of easy and/or fast lookups, none is inherently optimized for searching collections of human language text.

Designed not for storage but for quick retrieval of data from prebuilt indices, swish-e fills that need. Swish-e provides a native C interface and command line tools to build and query indices, and a perl interface for searching as well. Indices consist of a pair of binary files and are built using the `swish-e` binary and one of swish-e's three indexing methods. We have been using swish-e at SkateboardDirectory.com since 2002.

2.1 Building SWISH-E

Currently, installing swish-e on a unix-like system means building from source. You can find tarballs for the source code on the swish-e website, and swish-e is built through a typical install process using a `./configure` script. The Perl SWISH::API can be found in the /perl subdirectory and is likewise installed through the typical perl '`perl Makefile.PL; make; make test; sudo make install`' process.

2.2 Configuration Files

Swish-e will typically depend on a single configuration file while creating a index. These files follow a familiar line-oriented name/value syntax. Blank lines and lines beginning with a # are ignored, remaining lines are expected to be single, named directives. For example, this is a valid swish-e configuration file:

```
# example1.conf
IndexFile example1.index
DefaultContents HTML2
```

2.3 SWISH-E Parsers

Swish-e directly supports indexing of text, html, and XML files, converting HTML or XML entities where appropriate, and the ability to index data based on the tags it resides within. The XML2, HTML2, and TXT2 parsing engines, which require the libxml2 libraries, are preferable to the original counterparts (called XML1, HTML1 and TXT1), especially when handling HTML.

2.4 Properties and MetaNames

MetaNames are the fields in a swish-e index that are searched on. Properties are the fields returned from a swish-e search describing the particular documents. By default, text is indexed under the MetaName `swishdefault`, and Properties returned indicate information about each relevant document.

Several so-called Auto Properties are always present in search results returned from the swish-e APIs. The table below summarizes some of the most important ones.

Table 1: some of swish-e's Auto Properties

swishrank	Normalized integer between 1 and 1000 representing relevance of hit to query
swishtitle	Title of the document. Either the filename, or (by default) if parsed from HTML, the text between <title> tags
swishdocpath	URL or filepath of indexed document
swishdocsize	Length of indexed document, in bytes

2.4.1 String Properties

Any non-numeric properties are internally stored as strings. By default, any properties longer than 100 characters are compressed using zlib before storage, which helps keep the index sizes down. Note that each string is compressed on its own, so redundancy between properties is not exploited in the compression process.

2.4.2 Numeric Properties

Using the `PropertyNamesNumeric` directive, swish-e has the ability to store properties as unsigned integers, which allows for proper sorting numerically at search time. Unfortunately, the swish-e engine is not particularly efficient in its methods of sorting by numeric properties: currently the whole result list is simply sorted by integer after the index is searched and before the results are returned to the client.

2.5 Indexing Methods

Here we discuss three different ways to index data with swish-e. The first is to index files using one of the built-in parsers alone, as shown in the next section. The other two methods, FileFilters and external programs, allow for conversion of data from other sources or file formats.

2.5.1 Indexing Files Directly

Assume that our working directory contains the above `example1.conf` file and a set of HTML files we want to index in ./html, we can build the index `example1.index` using `swish-e`'s `-f` and `-i` options to specify the configuration file to use and which directory to index:

```
swish-e -f example1.conf -i ./html
```

2.5.2 Using FileFilters

For data in formats other than HTML, XML or TXT, you need to arrange to have the files converted to one of the formats that swish-e directly supports.

The most straightforward way to convert files for swish-e is via the `FileFilter` method. This is engaged by including lines like the following in your configuration file:

```
# example2.conf
FileFilter .pdf pdftotext "'%p' -"
IndexContents TXT .pdf
```

This specifies that files ending with .pdf are to be converted using the pdftotext executable (part of the xpdf package[5]), and then indexed using swish-e's `TXT` parser. This configuration file would be used similarly to the one shown in the previous example.

The downside to the FileFilter method is that swish-e invokes a child process for each document to be converted. This can present a performance issue during index time. On the other hand, assuming a program exists to perform the translation required, it can be easy to support additional file types by adding a pair of lines like the ones above to your configuration file.

2.5.3 Using External Programs

The most flexible mechanism for getting your data into a swish-e index is through External Programs. Essentially, external programs convert documents to a supported format as needed, wrap the result with appropriate swish-e headers, and pass that to `swish-e`. Again, swish-e will only interpret data as TXT, HTML, or XML, so if you have special needs, target your external program's output for a specific one of the parsers.

Here's an example external program that takes all the XML files in the apache 2.0 docs/manual tree, which we've copied to ./manual, and prepares them for indexing with swish-e.

```
#!/usr/bin/perl -w
# example3-prog.pl
# appends data to Path-Name: header

my @files =
`find ./manual -name '*.xml' -print`;

chomp(@files);
my $cnt = 1;
for my $f (@files) {
    open(FILE, "< " . $f);
    my $xml = join("", <FILE>);
    close(FILE);
    my $size = length $xml;
    # note: Fails if UTF
    print "Path-Name: $f $cnt\n",
       "Document-Type: XML*\n",
       "Content-Length: $size\n\n",
        $xml;
    $cnt++;
}
```

We didn't have to use an external program to index the XML files, but doing so allows us to easily introduce how to use some of swish-e's special XML handling features, and to show how to easily add MetaNames and Properties using swish-e's `ExtractPath` and `ReplacePath` directives.

2.6 XML, HTML, MetaNames and Properties

Swish-e lets you easily index any text within an HTML or XML tag as a MetaName and/or Property through use of the `MetaNames` and `PropertyNames` directives.

The following configuration file shows use of these directives, along with the `ExtractPath` and `ReplacePath` directives described below:

```
# example3.conf
MetaNames summary docnum swishtitle
PropertyNames summary
PropertyNamesNumeric docnum

# expect path like "/file/path 123"
#act like the 123 was in <docnum> tags
ExtractPath docnum regex \
     '!^.*( [0-9]+)$!$1!'
# remove the " 123" for indexing
ReplaceRules       regex \
      '!^(.*) [0-9]+$!$1!'
```

Note that although our example above uses backslashes to denote continued lines in our configuration file, `swish-e` does not support this feature, so make sure to enter each directive on its own line when writing configuration files.

The `swish-e` executable can be used as shown below to create an index from `example3.conf` and `example3-prog.pl`:

```
swish-e -f example3.index \
 -c example3.conf  \
 -i ./example3-prog.pl \
 -S prog
```

Here, -f sets the path of the index to be created, -c specifies the configuration file to use during the indexing process, and -i sets the program to be used to convert documents for `swish-e`. Lastly, the `-S prog` option denotes that the -i option specifies a program to be executed describing the documents to be indexed. If you forget the `-S prog` option, swish-e will index the file `example3-prog.pl` itself, and not the documents it describes when executed.

2.6.1 ExtractPath and ReplacePath

Using `ExtractPath` and `ReplacePath` can be useful for adding meta data to documents to be indexed. In this case, the `ExtractPath` directive serves to let the integer appended to the document path be indexed as though it had appeared inside the document in `<docnum>` tags. The `ReplaceRules` directive then removes the appended integer from the pathname so that it won't appear in the `swishdocpath` when retrieved from a swish-e index.

2.7 Searching

There are two main approaches to searching on a swish–e index: using the `swish-e` executable directly, or using one of the APIs to do so.

2.7.1 The SWISH-E Query Language

Swish-e supports logical grouping via parenthesis as well as AND, OR and NOT Boolean logic that behave predictably.

2.7.2 Searching With SWISH-E

Searching directly with `swish-e` is straightforward but not as flexible as the API. It is nevertheless very valuable for quick tests against indices to see that they work as expected.

For example, we can conduct searches on our `example3.index` like so:

```
swish-e -f example3.index -w restart
```

which returns results like (abridged and reformatted):

```
  1000 manual/stopping.XML
         "stopping.XML" 10577
  608 manual/platform/windows.XML
         "windows.XML" 30773
  608 manual/programs/apachectl.XML
         "apachectl.XML" 5818
  544 manual/mod/mpm_common.XML
         "mpm_common.XML" 39171
```

By default, each result contains a rank, the pathname of the indexed file, the title, and the byte count of the indexed data.

2.7.3 Using SWISH::API.pm

Here at SkateboardDirectory.com, one of the features that attracted us to swish-e was its perl interface, provided through the included SWISH::API. Most of the important features available for searching through the `swish-e` executable are also accessible through the perl API. Here's a short example that can perform searches similar to the one shown above:

```
  #!/usr/bin/perl -w
  #search4.pl
  use SWISH::API;
  my ($max, $cnt) = (10,0);
  my $index = "./example3.index";
  my $query = join(" ", @ARGV);
  my $handle = SWISH::API->new($index);
  my $results = $handle->Query($query);
  while ( ($cnt++ < $max) &&
  (my $res = $results->NextResult)) {
      printf "%d '%s' %d\n",
      $res->Property("swishrank"),
      $res->Property("swishdocpath"),
      $res->Property("swishdocsize");
  }
```

Of course, in a real-world application you should probably `use strict` and perform error checking.

2.7.4 The SWISH-E C API

There is also a C API for querying swish-e indices. It is similar to the perl API, but allows access to more of the low-level details in a swish-e index. For examples on use of the swish-e C API, see the swish-e documentation.

2.8 Other Notable SWISH-E Features

Some other features that warrant explanation are described in this section.

2.8.1 Merging indices

Swish-e has the ability to search on multiple swish-e indices simultaneously and merge the results meaningfully. This enables searching on groups of indices that may be collectively larger than the current per-index limit of about 2GB.

2.8.2 PHP Interface

Many people have expressed interest in a PHP interface to swish-e, and one is in development.[6]

2.9 SWISH-E Ranking

The ranking algorithm used in swish–e does not bear easy explanation, but does take into account factors including the size of the documents, the frequency of each word in the document, and which tags the given text resides in. Maintainer Moseley has repeatedly expressed his desire for someone to clean up the ranking code used in swish-e.

3 Real-World Examples

Here we examine two real-world uses of swish-e: sman, the Searcher for Man Pages; and swished, a concurrent, persistent swish-e deamon based on mod_perl.

3.1 Sman – Searcher For Man Pages

Sman is Searcher for Man Pages (written by the Author) which uses swish–e to offer ranked, fulltext searches on your system's manpages. The Sman package, which is currently available at http://joshr.com/src/sman, is likely to appear on CPAN,

Like most well-designed software, no single part of the sman package is particularly complex. However, due to the wide range of differences in the ways man pages are written, displayed and presented in various software packages and operating systems, there are relatively large amount of moving parts in sman.

As a high-level overview, sman consists of two programs: `sman` and `sman-update`. `Sman` performs the searches on the index of man pages, and `sman-update` updates that index. Sman is broken into a series of perl modules. By far most of the complexity is employed in `sman-update`.

`Sman-update`, which is intended to be run nightly, does everything necessary to parse your manpages and create a swish-e index which allows freetext searching on the compete text of all the manpages, as well as the ability to search on man pages by text in their command, section, pathname, or description. In a little more detail, `sman-update`:

- reads a configuration file (by default /usr/local/etc/sman-default.conf) which specifies options including where to store the final index, (by default /var/lib/sman/sman.index)
- finds your manpages
- figures out how to best convert your man pages to ASCII
- creates a temporary swish-e configuration file for use while indexing the man pages
- converts each manpage and parses the result to ascertain the title, description, section, and complete text.
- outputs XML to swish-e to parse, using the temporary configuration file

Sman, the tool that actually performs searches on the index, is essentially a highly enhanced version of `search4.pl`, shown above. We've skipped the details of the object-oriented design used. For more details, see the Sman source code and documentation.[7]

3.2 swished – A SWISH-E Daemon

For some time, one of the items on the swish-e to-do list has been for someone to write a persistent, concurrent server for swish-e indices. Here we present the overall design of such a daemon, written as an apache mod_perl handler.

There are at least three reasons that apache makes sense as an infrastructure for this purpose.

- http is a well defined protocol
- apache is stable, proven, widely deployable software that provides sophisticated logging, authentication, and extension mechanisms
- Perl, SWISH::API and mod_perl make it fairly easy

The plan is to provide a SWISH::API::Remote perl module that will access a swished daemon using an interface very similar to SWISH::API, but which is designed to communicate with swished over TCP/IP instead of directly reading the swish-e index.

4 SWISH-E vs MySQL

Just how fast is swish-e? Here we put swish-e and MySQL's fulltext search feature through the paces with some benchmarks against indices of different sizes and compare the results.

4.1 Differences and Similarities

While targeted at essentially the same problem of facilitating quick searches on larger amounts of textual data, the indexing models employed by MySQL's Full Text search and swish-e have significant differences. Some of these are outlined below.

Table 2: Some Pros and Cons of swish-e and MySQL

Engine	Pros	Cons
Swish-e	• more compact indices • indexes all words over 1 character	• not multibyte • no updates to indices* • only for indexing
MySQL	• updatable indices* • deep multibyte support • also a storage engine	• less compact indices • indexes only words over 3 characters** • ignores words appearing in over 50% of rows in a fulltext index***

* In swish-e, indexes must be rewritten to be modified. MySQL indices can be updated through normal sql queries.
** By default, MySQL will only index words four charaters or longer. This can be changed via MySQL's `ft_min_word_len` configuration option.
*** MySQL fulltext search does consider such words if searches are conducted in `boolean` mode, but then the results aren't ranked.

4.2 Benchmark Methodology

To compare swish-e with MySQL, a series of textual collections were created, varying in size from 1MB to 5GB. Each corpus was made of English words randomly chosen but based on the statistical frequencies of common English words. The following table summarizes the seven collections used in the benchmarks:

Table 3: breakdown of collections used for testing.

Approx Collection Size	Num Words/ Doc	Num Docs	Approx size of swish-e index	Approx size of MySQL index
1MB	15	8.3K	2.6MB	2.3MB
3MB	15	25K	5.9MB	6.8MB
10MB	54	25K	14MB	21MB
50MB	54	100K	56MB	109MB
250MB	273	100K	245MB	554MB
1GB	273	500K	970MB	2.0GB
5GB	1366	500K	4.6GB*	9.0GB

*The swish-e index for the 5GB collection was built in two equal-sized parts, as swish-e cannot yet support index files over about 2.1GB. The two indices used in the 5GB collection were searched simultaneously using swish-e's merge search feature.

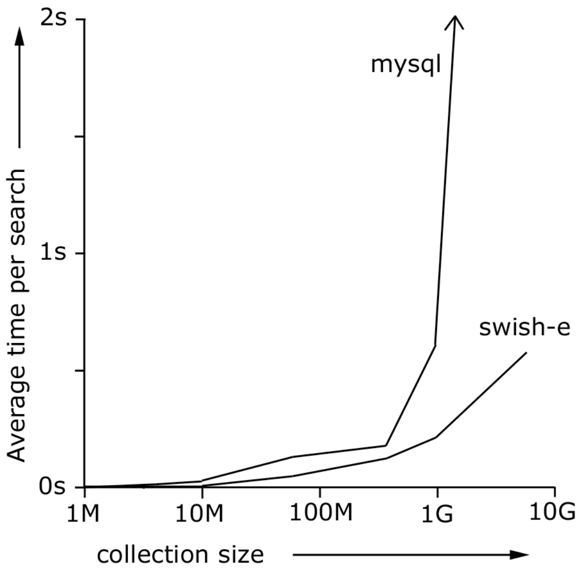
Tests were performed on a 3.06Ghz Hyper-Threading Pentium 4 with 2GB of RAM and an 80GB HD. The system was running a linux 2.6.5 kernel with SMP enabled. Swish-e version 2.5.1-2004-04-28 and MySQL 4.1.1 were used.

For MySQL, the provided `my-huge.cnf` configuration file was used, with the following modifications: `ft_min_word_len` was set to three instead of four, thus indexing words three or more letters (and not only words four letters or longer), and thread_concurrency was set to 4. Additionally, for the 1GB and 5GB indices, the MySQL configuration option MAX_ROWS was set to 10000000 and AVG_ROW_LENGTH was set to 13000 so as to allow enough data to be stored in each table.

The searches were performed with two groups of 200 searches made up of randomly selected words from the collections three characters or longer. In each group of 200 searches, 53 were 1-word searches, 115 were 2-word searches, 25 were 3-word searches, 4 were 4-word searches, 2 were 5-word searches, and 1 was a six-word search. (This approximates the distribution of numbers of words in the most popular searches on SkateboardDirectory.com.) All swish-e searches were conducted with the words ANDed, and MySQL searches were conducted with the + prefix, which has the effect of requiring relevant documents to contain all words in the query. Each index was tested by searching twice with two sets of 200 queries, retrieving the pathname and complete content of the first 20 documents found relevant, and computing the average response time. The MySQL server was not running when swish-e was being benchmarked, and the machine was rebooted between each test run.

4.3 Benchmark Results

Graph 1: MySQL against swish-e, showing average search time vs. collection size.



(The average search time for searches on the 5GB MySQL collection was 14 seconds, way off the graph above.)

As the graph above indicates, both engines are very fast, but swish-e is faster, especially for large indices. When searching on the 5GB collection using MySQL, it appears that performance suffers as the machine begins to thrash, as the index can no longer fit in memory and/or MySQL's caches. Adjusting MySQL's configuration may help narrow this gap but it appears that for now, swish-e is faster for raw searches. There is also likely more speed to be squeezed out of swish-e.

5 Research Ideas

As alluded to before in section 2.9, maintainer Moseley thinks that work on the ranking algorithm in swish-e is worthy of a graduate level thesis in computer science. Considering that Google is based on a 1998 Stanford hypertext retrieval project focused on ranking web pages[8], this is probably an understatement.

6 Limitations and Weaknesses

For all its strengths, swish-e has some weaknesses.

6.1 Size limits

There are various hard-coded and system-oriented size limits in swish-e, including limits on the maximum lengths of words, Properties, and indices as a whole.

6.2 Character Sets And Conversion Issues

Also, swish-e is admittedly an 8bit indexing system, and has no multibyte nor UTF support. Indices must be built and queried for a particular character set.

6.3 Occasionally Quirky Search Results

The ranking algorithm of swish-e occasionally leads to surprising, even erroneous search results, typically by failing to rank highly documents that one would expect to be relevant.

7 Future Plans for SWISH-E

These are some features the developers of swish-e feel are important.

7.1 UTF-8 support

This would enable a single index to hold data in languages that require multibyte characters, and would make swish-e indexes fully 8bit aware. According to Moseley, adding support for UTF-8, which he believes to be the best course of action, would require a near total rewrite of swish-e because of many assumptions that the size of one character is one byte.

7.2 Remove Two Gigabyte Limits

The files used in a swish-e index are currently limited to about 2GB in size, even on systems which support larger files. Support for indices larger than 2GB is in development.

7.3 Ranking Improvements

Mentioned above, there is room for improvement in the ranking algorithms.

8.0 Acknowledgements

Thanks to Kevin Hughes, Bill Moseley, Jóse Manuel Ruiz, David Norris, Roy Tennant, and Adam Souzis for their input and feedback on this paper. In addition, thanks to the entire current and past swish, swish-e, and MySQL development teams for great open source software products.

9.0 Conclusion

It is clear that swish-e is a powerful and fast engine with which to create and search on indices. The underlying speed of swish-e, coupled with its quality documentation, lively discussion list, perl, C, and PHP interfaces, provide a robust and flexible foundation upon which to build searching systems.

References

[1] Hughes, K., Moseley, B. et. Al.: *SWISH-E*. `www.swish-e.org`, 2004.

[2] MySQL AB: *MySQL*. `www.mysql.com`, 2004.

[3] Rabinowitz, J.: *How To Index Anything*; Linux Journal, July 2003, also at `www.linuxjournal.com/article.php?sid=6652`

[4] Sleepycat Software: *Berkeley DB*. `www.sleepycat.com`, 2004.

[5] Glyph & Cog, *Xpdf*; `www.foolabs.com/xpdf/`

[6] Ruiz, J. M., *php-swishe*; `http://prdownloads.sourceforge.net/php-swishe/`

[7] Rabinowitz, J.: *Sman: The Searcher for Man Pages;* `www.joshr.com/src/sman/`

[8] Page, L., Brin, S., Motwani, R., Winograd, T.: *The PageRank Citation Ranking: Bringing Order to the Web*; Stanford Digital Libraries Project, 1998.