Sphinx: Leveraging Scalable Search in Drupal

Mike Cantelon
http://straight.com
http://mikecantelon.com

Drupal Camp
Vancouver,
May 16 - 17, 2008

This Talk

- Aims to orient you to the Sphinx realm
- Code provided on mikecantelon.com to give you basic Drupal Sphinx-driven search

What Is Sphinx?

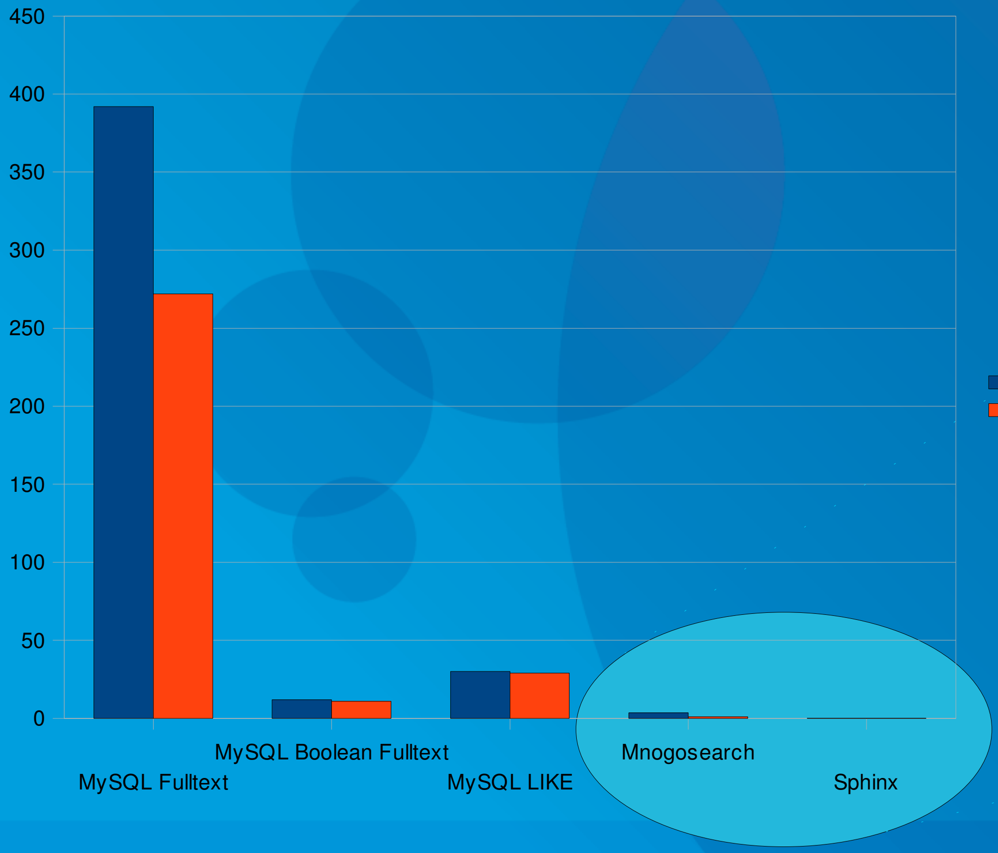
- Open source search engine
- Developed by Andrew Aksyonoff
- No Java required
  (written in C++)
- Mostly cross-platform
  (Windows version not yet production ready)
- Fast
- Scalable
- Flexible
- Straightforward

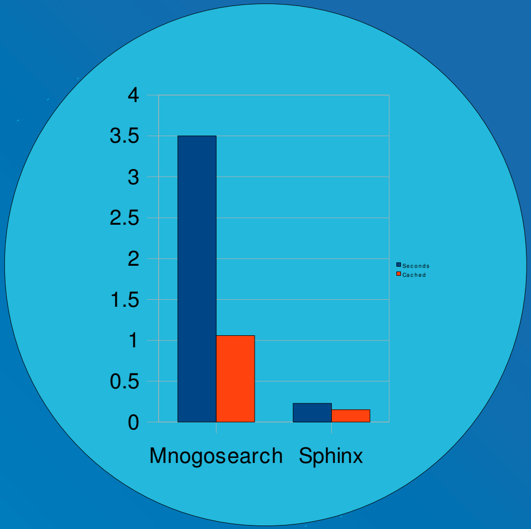
Who Uses It?

- NowPublic
- MySQL
- The Pirate Bay
- Joomla
- MiniNova
- Etc...
  http://www.sphinxsearch.com/powered.html

How Fast is It?

http://peter-zaitsev.livejournal.com/12809.html

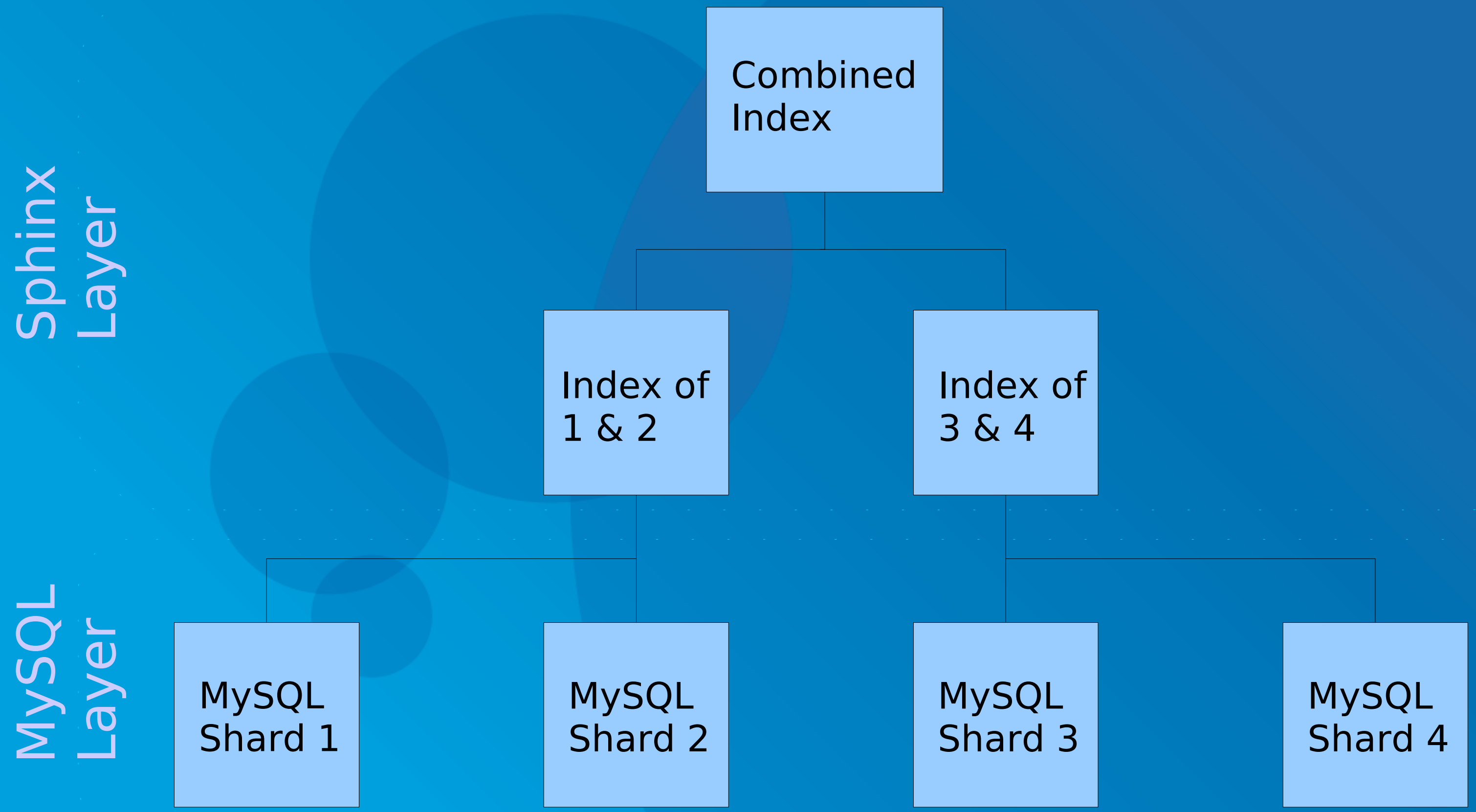




How Does Sphinx Scale?

- Distributed, parallel searching/indexing across multiple agents
- Agents can be machines (“horizontal scaling”) or processors/cores (“vertical”)
- Searches get first sent to remote agents
- Distribution is transparent to the application: any duplicate results are removed

One of Many Potential Sphinx Architectures



Sphinx Components and Installation

- Installation is (usually) straightforward
  (requires make and C++ compiler)
  - ./configure
  - make
  - make install
- Sphinx consists of three components:
  - Search daemon (*searchd* command)
  - Indexer (*indexer* command)
  - Search client (*search* command)

Sphinx Index Fundamentals

- Can have one or more data source
- Sources include MySQL, MySQL Sphinx storage engine, Postgres, xmlpipe2
- Each source must share schema
- Each document must have global ID
- Indexes can be rotated seamlessly
- Indexes end up taking *much more* disk space than source text: plan accordingly!

Sphinx Index Options

- UTF-8 encoding
- Prefix and infix indexing for wildcard searches
- HTML stripping
- N-gram support for Chinese, Japanese, etc.
- Multiple stemming schemes
- Stop words
- Word forms

Sphinx Stemming

- Good stemming support
- “Stemming” is the breaking down of words into their base form
- In addition to stemming, Sphinx supports indexing via Soundex and Metatone
- Multiple stemmers can be applied
- Optionally support for Snowball project for international stemming algorithms (http://snowball.tartarus.org)

Configuration

Sphinx Configuration Overview

- */usr/local/etc/sphinx.conf* is where Sphinx configuration normally lives
- Defines sources, indexes, and agents
- Configuration file is in custom format

Sphinx Source Configuration

- Source specifies:
  - Type
  - Connection info
  - Queries
  - Attributes
- Contains three kinds of queries:
  - Main query (what you want to index; first column must be document global ID)
  - Detail query (to fetch document details when using *search* command)
  - Optional “range” query

Source Range Query

- The range query is for pulling batches of rows, instead off all rows at once
- MyISAM has table-wide locking, so big queries can bog things down
- An example range query:

```
sql_query_range = SELECT MIN(nid),MAX(nid) FROM node
sql_range_step = 512
```

- *sql_range_step* defines how many rows to fetch on each pass

Sphinx Index And *searchd* Configuration

- The index will specify a source, path, and stemming/indexing options
- Make sure the path you specify for your index exists (for example: /var/opt/local/sphinx)
- The searchd section will require a directory to exist for logs, process ID file, etc.
- This directory will usually be */var/log/searchd*

Starting The Sphinx Search Daemon

- *searchd* can be started by simply typing “sudo searchd” into the command line
- Once *searchd* is started, index a source by entering “sudo indexer <name of source>”
- Once your index is created, enter “search -p <some search term>” to test
- “Seamless” indexing can be done by adding the “--rotate” option when using *indexer*

Implementation

Example of a Sphinx Implementation

- The “my_search” module, which will be available for download on mikecantelon.com, shows a simple use of Sphinx in Drupal

Sphinx Implementation Overview

- Sphinx data is accessible through multiple APIs: PHP, Python, Perl, Ruby, and Java
- APIs provides optional search snippet extraction, keyword highlighting, and result paging
- Results return only document IDs
- Hence, after searching returns results, you'll still likely need to fetch details of documents in result page

Sphinx Search Modes

- SPH_MATCH_ALL: match all keywords
- SPH_MATCH_ANY: match any keywords
- SPH_MATCH_BOOLEAN: no relevance, implicit boolean AND between keywords if not otherwise specified
- SPH_MATCH_PHRASE: treats query as a phrase and requires a perfect match
- SPH_MATCH_EXTENDED: allows for specification of phrases and/or keywords and allows boolean operators

Example of PHP Client Connect

- This example uses SPH_MATCH_EXTENDED:

```
// Note: searchd needs to be running
$cl = new SphinxClient();
$cl->SetServer('localhost', 3312 );
$cl->SetLimits($start_item, $items_per_page);
$cl->SetMatchMode(SPH_MATCH_EXTENDED);
```

- Note the use of **`SetLimits`** to allow us to implement paging/limiting

Sphinx Sort Modes

- Sphinx has a number of limited, specific search modes :
  - SPH_SORT_RELEVANCE
  - SPH_SORT_ATTR_DESC
  - SPH_SORT_ATTR_ASC
  - SPH_SORT_TIME_SEGMENTS
- SPH_SORT_EXTENDED provides SQL-like sorting, listing “columns” and specifying ascending/descending order
- SPH_SORT_EXPR allows sorting using a mathematical equation involving “columns”

Example of PHP Client Sort and Query

- This example sorts by relevance (most to least), then by creation date (reverse chronological order):

```
$cl->SetSortMode (SPH_SORT_EXTENDED,
  "@relevance DESC, created DESC");

// Do Sphinx query
$search_result = $cl->Query(
  $query,
  'stories'
);
```

Getting Fancier

Using the Sphinx PHP API Highlighter

- Once you've received search results from Sphinx, you can use API's highlighter to generate excerpts and highlight matches within the excerpts
- To do this you must *first retrieve the full text* of each of the documents shown on the current results page
- This will need to be done by database queries or some kind of cache lookup

PHP API Highlighter Example

- In this example $content is an array containing document IDs for keys and document text for values:

```
$highlighting_options = array(
   'before_match'     =>
   "<span style='background-color: yellow'>",
   'after_match'      => "</span>",
   'chunk_separator' => " ... ",
   'limit'            => 256,
   'around'           => 3,
);
$excerpts = $cl->BuildExcerpts($content,
'stories', $query, $highlighting_options);
```

Humanizing SPH_MATCH_EXTENDED

- SPH_MATCH_EXTENDED can be “humanized” by parsing a user's query and breaking it into phrases and keywords separated by a chosen operator
- For example: given the search query *“uwe boll” postal* and the chosen search type “match any”, we'd use boolean operator OR between the extracted phrase and keyword
- This would end up, internally, as *“uwe boll”* | *postal* (note the OR pipe character)

Implementing Humanization in PHP

- One way to humanize via clumsy regular expression brutality:
  1) Get keywords by using preg_split to split into array using phrases (designated by double-quotes) as splitting points
  2) Get phrases by using preg_match_all, matching anything in double-quotes
  3) Merge arrays
  4) Implode using search operator with spaces on either side (i.e. “ & “ or “ | “)

What About Paging?

- As Sphinx queries don't go through Drupal's database abstraction functions, built in paging won't work automagically, as when using Drupal's *pager_query* function
- Writing paging logic that looks/acts identical to Drupal's is tedious
- You can, however, leverage Drupal's paging functionality so *theme_pager* will do your bidding...

How to Leverage Drupal's Paging

- Manually set globals to leverage paging:

```
$element = 0;
$GLOBALS['pager_page_array'] =
  explode(',', $_GET['page']);
$GLOBALS['pager_total_items'][$element] =
  $found_count;
$GLOBALS['pager_total'][$element] =
  ceil($found_count / $items_per_page);
$GLOBALS['pager_page_array'][$element] =
max(0, min((int)$GLOBALS['pager_page_array']
[$element], ((int)$GLOBALS['pager_total']
[$element]) - 1));
```

Multi-Value Attributes (MVAs)

- Allow to you add Drupal taxonomy data to Sphinx index
- Implementation requires two things:
  - Addition of *sql_attr_multi* specification in the source section of your Sphinx configuration file
  - *sql_attr_multi* tells how to query for taxonomy
  - Addition of *SetFilter* call in PHP API before issuing query
- *SetFilter* example filtering by two taxonomy terms:
  - $cl->SetFilter('tag', array(114288, 4567));

MVA sql_attr_multi Example

```
sql_attr_multi = uint tag from ranged-query; \
  SELECT n.nid, td.tid FROM term_data td INNER
JOIN term_node tn \
    USING(tid) INNER JOIN node n ON
tn.nid=n.nid \
    WHERE td.vid=9 AND n.nid >= $start AND
n.nid <= $end; \
  SELECT MIN(n.nid), MAX(n.nid) FROM term_data
td INNER JOIN term_node tn \
    USING(tid) INNER JOIN node n ON
tn.nid=n.nid \
    WHERE td.vid=9
```

We're done!

Online Resources

- Official Sphinx site: http://www.sphinxsearch.com/
- These slides and example module: http://mikecantelon.com/talks/sphinx