Full-Text with PHP and Sphinx

Vladimir Fedorkov @ NYPHP.ORG
September 25\textsuperscript{th}, 2012
About me

• Performance geek
  – blog http://astellar.com
  – Twitter @vfedorkov

• Enjoy LAMP stack tuning
  – Especially MySQL

• Enjoy speaking on the conferences

• Use Sphinx in production from 2006
Search is important

• Keep customer **satisfied**
  – Let visitor find what he need

• Search is the way to **explore** the website
  – Find something that your customer doesn’t know
  – Show your customer what **you want** to show
Good search is important

• And it’s more than one side:
  – Speed
    • Rule 0.1 ... 1 ... 10
  – Relevance
  – Convenience
  – Flexibility
  – Simple maintenance
  – Fault tolerance
Available solutions

• Most databases have integrated FT engines
  – MySQL: MyISAM FT index
    • Recently released FT support in InnoDB

• Standalone solutions
  – Solr, Lucene, Sphinx.

• External services
  – IndexDen, SearchBox, Flying Sphinx, WebSolr, ...
Sphinx records

- Standalone open source search server
- Searching though **Billions** of documents
  - Over 30,000,000,000 at Infegy
  - Over 26,000,000,000 at boardreader.com
    - over 8.6Tb indexed data across 40+ boxes
- Serves 200,000,000+ queries per day
  - craigslist.org 2,000+ QPS against 15 Sphinx boxes
- 10-1000x **faster** than MySQL on full-text searches
  - Even faster on faceted search queries
- List is not complete
Example. Search against 8M rows.

MySQL

mysql> SELECT id, ...
   --> FROM myisam_table
   --> WHERE MATCH(title, content_ft)
   --> AGAINST ('I love sphinx') LIMIT 10;
...
10 rows in set (1.18 sec)

Sphinx

mysql> SELECT * FROM sphinx_index
   --> WHERE MATCH('I love Sphinx') LIMIT 10;
...
10 rows in set (0.05 sec)
mysql> SELECT * 
    -> FROM sphinx_index
    -> WHERE MATCH('I love Sphinx')
    -> LIMIT 5
    -> OPTION field_weights=(title=100, content=1);

+-----------------+--------+------------------+------------+
| id              | weight | channel_id       | ts         |
+-----------------+--------+------------------+------------+
| 7637682         | 101652 | 358842           | 1112905663 |
| 6598265         | 101612 | 454928           | 1102858275 |
| 6941386         | 101612 | 424983           | 1076253605 |
| 6913297         | 101584 | 419235           | 1087685912 |
| 7139957         | 1667   | 403287           | 1078242789 |
+-----------------+--------+------------------+------------+
5 rows in set (0.05 sec)
Key differences

• Meta fields @weight, @group, @count
• No full-text fields in output
  – Requires additional lookup to fetch data
• MySQL query become primary key lookup
  – WHERE id IN (33, 9, 12, ..., 17, 5)
  – Good for caching
  – Good compatibility with NoSQL date storages
• Scaling is transparent for the application
SQL & SphinxQL

- WITHIN GROUP ORDER BY
- OPTION support for fine tuning
  - weights, matches and query time control
- SHOW META query information
- CALL SNIPPETS let you create snippets
- CALL KEYWORDS for statistics
Full-Text functions

• And, Or
  – hello | world, hello & world
• Not
  – hello -world
• Per-field search
  – @title hello @body world
• Field combination
  – @(title, body) hello world
• Search within first N
  – @body[50] hello
• Phrase search
  – “hello world”
• Per-field weights
• Proximity search
  – “hello world”~10
• Distance support
  – hello NEAR/10 world
• Quorum matching
  – "the world is a wonderful place"/3
• Exact form modifier
  – “raining =cats and =dogs”
• Strict order
• Sentence / Zone / Paragraph
• Custom documents weighting & ranking
Non Full-Text searches

• GEO-distance search support
• Faceted search support
  – Date and time segments
  – Price ranges and other
• Built in one – to – many attributes
  – For page tags
  – For multi category items
• Numeric, timestamps and string support
Integration ways

• Indexing
  – MySQL, PostgreSQL, MSSQL and any ODBC source
  – Insert/Update/Delete for Real-Time engine
    • Via SphinxQL

• Search
  – API
    • PHP, Python, Java, Ruby, C is included in distro
    • .NET, Rails (via Thinking Sphinx) via third party libs
  – MySQL-compatible protocol
<?php

require ( "sphinxapi.php" ); //included in distro

$cl = new SphinxClient();

$cl->SetServer ( "127.0.0.1", 9312 );

$res = $cl->Query ( "iphone", "shop_items" );

//some error processing

var_dump($res)
?>

Via PHP API
Sphinx components

- Indexer
- Indexes
- Daemon
Architecture sample

- Database
- Indexing
- Sphinx index
Architecture sample

Diagram:
- Application
- Sphinx daemon
- Database
- Indexing
- Sphinx index

Connections:
- Application to Sphinx daemon with query
- Application to Database
- Database to Indexing
- Indexing to Sphinx index
- Additional data to Database
Sphinx applications

• Find relevant documents
  – Items in store(s)
  – Articles in blog/forum/news/etc website(s)
  – Pictures or photos
    • By text, description, GEO-data, publish time, etc
  – Friends
    • In social networks or dating websites

• Offload main database from heavy queries
• Build advanced search and search-based services
From search to facets

• Search drill-down
  – By category
  – By document (item) date
    • Today / Week / Month / Year / Others
  – By price range
  – By distance

• Show best documents
  – For front page
  – For category/brand/etc pages
Faceted search support

• Usually more than one facet
  – Multiquery support
    • Common part calculates just once

• Aggregation function support
  – MIN(), MAX(), COUNT(), COUNT(DISTINCT ...)

• WITHIN GROUP ORDER BY
  – Best items in each subgroup
Faceted search: drill-down by years

mysql> SELECT ..., YEAR(ts) as yr
    -> FROM sphinx_index
    -> WHERE MATCH('I love Sphinx')
    -> GROUP BY yr
    -> WITHIN GROUP ORDER BY rating DESC
    -> ORDER BY yr DESC
    -> LIMIT 5
    -> OPTION field_weights=(title=100, content=1);

<table>
<thead>
<tr>
<th>id</th>
<th>weight</th>
<th>channel_id</th>
<th>ts</th>
<th>yr</th>
<th>@groupby</th>
<th>@count</th>
</tr>
</thead>
<tbody>
<tr>
<td>7637682</td>
<td>101652</td>
<td>358842</td>
<td>1112905663</td>
<td>2005</td>
<td>2005</td>
<td>14</td>
</tr>
<tr>
<td>6598265</td>
<td>101612</td>
<td>454928</td>
<td>1102858275</td>
<td>2004</td>
<td>2004</td>
<td>27</td>
</tr>
<tr>
<td>7139960</td>
<td>1642</td>
<td>403287</td>
<td>1070220903</td>
<td>2003</td>
<td>2003</td>
<td>8</td>
</tr>
<tr>
<td>5340114</td>
<td>1612</td>
<td>537694</td>
<td>1020213442</td>
<td>2002</td>
<td>2002</td>
<td>1</td>
</tr>
<tr>
<td>5744405</td>
<td>1588</td>
<td>507895</td>
<td>995415111</td>
<td>2001</td>
<td>2001</td>
<td>1</td>
</tr>
</tbody>
</table>

5 rows in set (0.00 sec)
Misspells correction service

• Provides correct search phrase
  – “Did you mean” service

• Allows to replace user’s search on the fly
  – if we’re sure it’s a typo
    • “ophone”, “uphone”, etc
  – Saves time and makes website look smart

• Based on your actual database
  – Effective if you DO have correct words in index
Bundled solution

- Helper script is located in /misc/suggest/
  - suggest.conf includes required Sphinx index
  - suggest.php is an actual implementation
- Requires PHP and MySQL to work
- Based on the tri-grams & levenshtein function
Limitations and features

• Provided as a showcase, not a complete service
• Doesn’t work with UTF8
  – PHP function limitation
• Based on your actual database
  – Index required rebuild as you have new data
• Script is only provides you word-by-word correction
• Works better in combination with autocompletion service
Autocompletion service

• Suggest search queries as user types
  – Show most popular queries
  – Promote searches that leads to desired pages
  – Might include misspells correction
Implementation

• Enable prefix indexing
  – Set min_prefix_len and prefix_fields

• Use pre-built index with search phrases
  – Based on user’s input
  – Based on document statistics

• Use star search: MATCH (‘ipho*’)  
  – It’s sometimes wise to delay search until 3-4 letters has typed
Related search

• Improving visitor experience
  – Providing easier access to useful pages
  – Keep customer on the website
  – Increasing sales and server’s load average

• Based on documents similarity
  – Different for shopping items and texts
  – Ends up in data mining
Implementation

• Uses main Sphinx index
• Basic implementation uses quorum operator
  – “Sony NEX-5N”/2
  – “Mitt Romney wonders why airplane windows don’t open”/2
• Next step: use custom ranking
• Next step: enable statistics
  – Keywords/Phrases
  – Shopping experience
• Next step: use internal information
Quick summary

• Basic search
• Facets
• Search-based services
  — Misspells
  — Autocompletion
  — Related
• Speeding up search
  — Advanced tricks
  — Scaling & clouds
Non Full Text Search

• Offloading database from bad queries
  – Heavy & Long running
  – That can’t be efficiently handled
    • Flags enabled = 1

• Can be combined with full-text queries
  – On plain queries
  – In faceted search
GEO-Distance support

• Geographical distance is the distance measured along the surface of the earth
  – Two pairs of float values (Latitude, Longitude)
• GEODIST(Lat, Long, Lat2, Long2) in Sphinx

SELECT *, GEODIST(docs_lat, doc_long, %d1, %d2) as dist,
FROM sphinx_index
ORDER BY dist DESC
LIMIT 0, 20
Search within range

• Grouping results by
  – Price ranges (items, offers)
  – Date range (blog posts and news articles)
  – Ratings (product reviews)
• INTERVAL(field, x0, x1, ..., xN)

SELECT
  INTERVAL(item_price, 0, 20, 50, 90) as range,
  @count
FROM my_sphinx_products
GROUP BY range
ORDER BY range ASC;
Text search for integers

• Meta keywords search sometimes faster
  – __META_AUTHOR_ID_3235
  – __META_AUTHOR_NAME_Name_Kelby

• First letter search
  – __ARTIST_A, __ARTIST_B, __ARTIST_C, ...

• Static ranges emulation with meta keywords
  – __MY_RANGE_0, __MY_RANGE_1, ...

• Not flexible, but fast
Another way to speed up is scaling

• Combine different indexes
  – Main + Delta
  – Ondisk + RT
  – Distributed and local
    • Don't forget about dist_threads!

• Use parallel indexing
OnDisk indexes

Application

Sphinx daemon

Database

Indexing

Sphinx index

Query

Search results

Additional data
On disk vs Real-time indexes

- Database
  - SELECT
  - Indexing
    - OnDisk index
- Application
  - INSERT, UPDATE, DELETE
  - Sphinx daemon
    - Real-time RAM chunk
    - Disk chunk(s)
Bright side of scaling

• Faster search
• Better load control
• Hardware utilization
Dark side

• Hardware faults
• Network issues
• Balancing issues
  – Search time related to slowest search chunk
• Complicated operations
How to survive

• Compact indexes
  – Remove stopwords
  – Use bitmasks

• Set max_matches to an appropriate value

• Emergency controls
  – cutoff
  – max_query_time
How to survive II

• Tune distributed indexes
  – Concurrency control
    • dist_threads
    • max_children
  – Network & wait timeouts
    • agent_connect_timeout
    • agent_query_timeout
Use redundant indexes

• Sphinx will remove duplications automatically
  – You will have complete results event if node fails

• Use Sphinx HA
  – Not yet public
    • Could be found in trunk!
More info

• [http://sphinxsearch.com/docs](http://sphinxsearch.com/docs)
• Conferences
  – I’ll be doing Sphinx tutorial Oct 1st at Percona Live NY
  – Sphinx team at Oracle Open World & MySQL connect in San Francisco
• Invite me to speak
  – Ping me via email vlad@astellar.com
• Follow me on twitter @vfedorkov
• [http://astellar.com](http://astellar.com)
  – Upcoming webinars
• Ask questions here 😊
Thank you!