



Sphinx 101

Andrew Aksyonoff // Sphinx Technologies Inc. // 2009



Who are you?

- Sphinx – FOSS full-text search engine



<http://sphinxsearch.com>

Who are you?

- Sphinx – FOSS full-text search engine
- Why Sphinx?
 - Quick indexing and searching
 - Good relevance (we think)
 - Scales well
 - Can improve non-fulltext (!) queries
 - Lots of other features

But we have MySQL, Lucene/Solr, etc?

- Fine – as long as they **do** work for you, but...
- MySQL fulltext – can be fine for small DBs, but won't really scale even to 100K-1M rows
- Lucene/Solr – perform and scale better, but
 - Sphinx still indexes and searches faster
 - Sphinx yields better relevance (we think)
 - Sphinx is better at SQL style queries
 - They have RT Updates though (but we will too)

Faster as in?

- 10-1000x (!!!) faster than MySQL on fulltext
 - Just because MySQL degrades heavily when fulltext indexes start to hit the disk
- 2-3x faster than MySQL on scans
 - On a single core
 - Less overheads, but less dynamic
- 2-4x faster than Lucene on fulltext
 - At least on our internal benchmarks

Relevance as in?

- Actually, it was why I started Sphinx
- Run-of-the-mill BM25 does not care about keyword positions
- OK for rare keywords, but kills searches like "to be or not to be", "i feel you", etc
- Sphinx boosts (sub) phrase matches
- Perfect match is guaranteed to be ranked #1

Scales as in?

- Over 2,000,000,000 (yes billion) documents
- Over 10,000,000 queries/day
- Powers Craigslist, Meetup, WikiMapia, etc

Still interested?

- Lets see how it works
- Chapter 1 – workflow overview
- Chapter 2 – two big perf-related features
- Chapter 3 – war story
 - If time permits...

Chapter 1. A bit of engine insides



Sphinx workflow

- Indexing first – using indexer program
- Searching second – using searchd program
- There are data sources (what and where?)
- There are indexes
 - What data sources to index
 - How to process the incoming text
 - Where in the FS to put the index files

So how do I index?

- Define a source – basically an SQL query
- Define an index – basically FS path + misc text processing settings
- Run indexer
- ...
- **PROFIT!!!**

Show me yours, I'll show you mine (config file, that is)

```
source test1
{
    sql_query = SELECT id, title, descr, added, price \
        FROM products
    sql_attr_timestamp = added
    sql_attr_float = price
}

index test1
{
    source = test1
    path = /my/index/store/test1
}
```

Indexing for the real curious

- Typical inverted index
- In two acts, with an intermission
- Act 1 – collect documents
 - Fetch, tokenize, normalize, partially sort keywords
- Act 2 – final sort
- Intermission – attributes
 - Fetch, sort, store

And then how do I search?

- Run searchd
- Query it
 - Through native API (PHP, Perl, Python, Ruby, Java, C#, Haskell...)
 - Through SphinxSE
 - Through MySQL interface

And then how do I search?

- Native API?

```
<?php

require ( "sphinxapi.php" );
$client = new SphinxClient ();
$res = $client->Query ( "my first query", "test1" );
var_dump ( $res );

// wham, bam, searching kinda done
```

And then how do I search?

- SphinxSE?
- Something you compile into MySQL, and then

```
SELECT *
FROM sphinxsetable s
JOIN products p ON p.id=s.id
WHERE s.query='@title ipod'
ORDER BY p.price ASC

// or better!
... WHERE s.query='@title ipod;sort=attr_asc:price';
```



And then how do I search?

- MySQL interface? (aka SphinxQL)
- 1-week fresh killer feature
- Drumroll...



SphinxQL FTW

```
$ mysql -P 3307
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 0.9.9-dev (r1734)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT * FROM test1 WHERE MATCH('test')
      -> ORDER BY group_id ASC OPTION ranker=bm25;
+-----+-----+-----+-----+
| id    | weight | group_id | date_added |
+-----+-----+-----+-----+
|     4 |    1442 |         2 | 1231721236 |
|     2 |    2421 |        123 | 1231721236 |
|     1 |    2421 |        456 | 1231721236 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

What the searcher can do

- SphinxQL explains it pretty good now
- It supports arbitrary expressions in SELECT
- It supports WHERE, ORDER BY, GROUP BY, COUNT, AVG/MIN/MAX/SUM
- It supports our own extensions
 - Such as OPTION ranker=bm25
 - Such as WITHIN GROUP ORDER BY ...

What's beyond basic searching?

- Many features of different caliber, actually
 - MVAs, tokenizing settings, wordforms, 1-grams, HTML processing, throttling, geosearching, arbitrary expressions, prefix/infix indexing, tuning the ranking...
- Let's quickly cover two frequently used ones
- Both optimization related

Chapter 2. A bit of optimizations



How to optimize queries

- Partitioning + distributed search
- Multi-queries

- More out-of-scope keywords
 - Choosing ranking vs. sorting mode
 - Filters vs. keywords
 - Filters vs. manual MTF
- Three Big Magic Buttons

Partitioning

- Swiss army knife, for different tasks
- Bound by indexing time?
 - Partition, re-index the recent changes only
- Bound by filtering?
 - Partition, search the needed indexes only
- Bound by CPU/HDD?
 - Partition, move out to different cores/HDDs/boxes

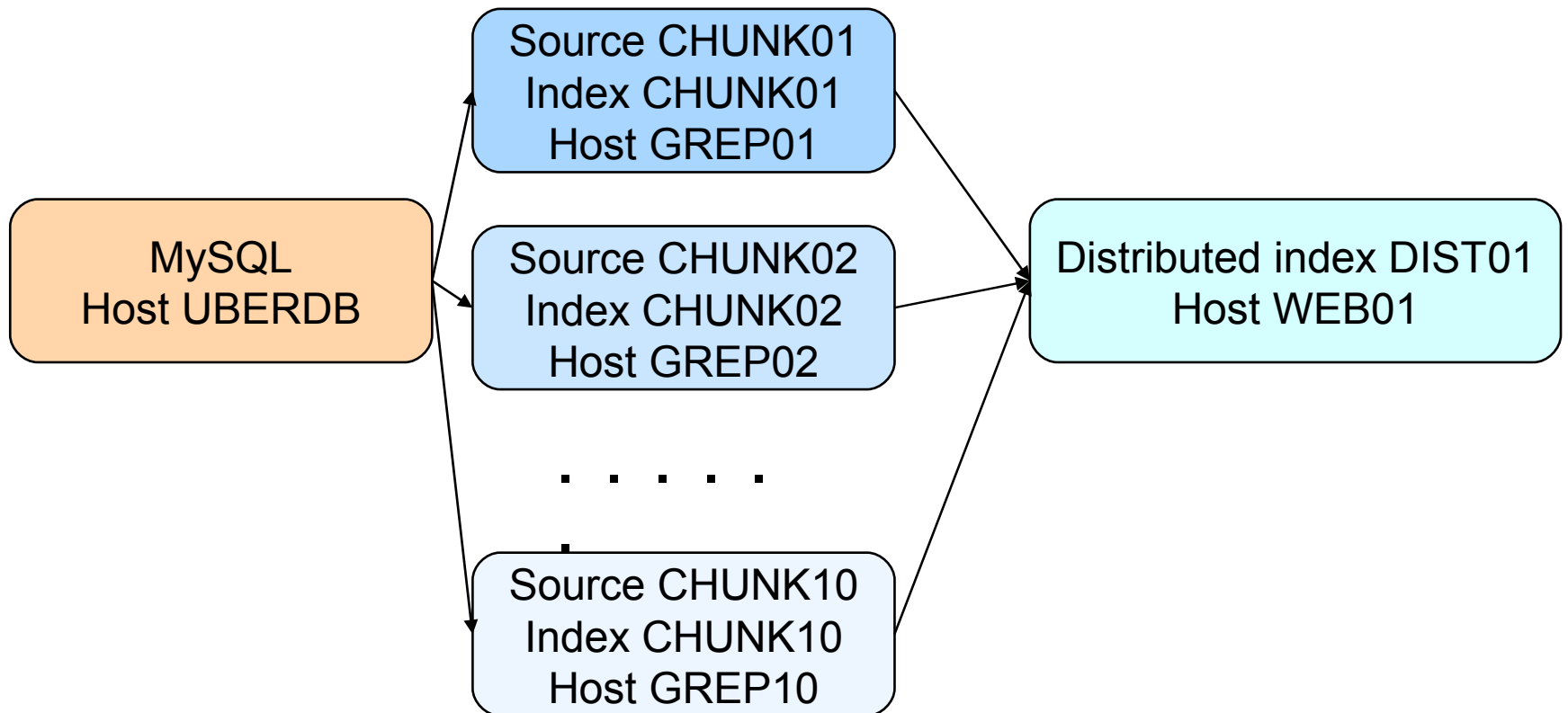
Distributed indexes

- Essentially, just lists of local+**remote** indexes

```
index dist1
{
    type = distributed
    local = chunk1
    agent = box02:3312:chunk02
    agent = box03:3312:chunk03
    agent = box04:3312:chunk03
}
```

- All remote ones are searched in parallel
- All results are merged together
- Agents can point to master itself (multi-core)

Distributed indexes: divide and conquer



Multi-queries

- **Any** queries can be sent together in a batch
- Always saves on network roundtrip
- Sometimes allows the optimizer to trigger

- Especially important and frequent case – different sorting/grouping modes
- 2x+ optimization for “faceted” searches

Multi-query sample

```
$client = new SphinxClient ();  
$q = "laptop"; // coming from website user  
  
$client->SetSortMode ( SPH_SORT_EXTENDED, "@weight desc" );  
$client->AddQuery ( $q, "products" );  
  
$client->SetGroupBy ( SPH_GROUPBY_ATTR, "vendor_id" );  
$client->AddQuery ( $q, "products" );  
  
$client->ResetGroupBy ();  
$client->SetSortMode ( SPH_SORT_EXTENDED, "price asc" );  
$client->SetLimit ( 0, 10 );  
$client->AddQuery ( $q, "products" );  
  
$result = $client->RunQueries ();
```

Three Big Magic Buttons

- If nothing else helps...
- Cutoff (cm. `SetLimits()`)
 - Forcibly stops searching after first N matches
 - Per-index, not overall
- MaxQueryTime (cm. `SetMaxQueryTime()`)
 - Forcibly stops searching after M milli-seconds
 - Per-index, not overall

Three Big Magic Buttons

- If nothing else helps...
- Consulting 😊
 - We can notice the unnoticed
 - We can implement the unimplemented



Chapter 3. Parallelization sample



Combat mission

- Got ~160M cross-links
- Needed misc reports (by domains→groupby)

```
***** 1. row *****
  domain_id: 440682
  link_id: 15
  url_from: http://www.insidegamer.nl/forum/viewtopic.php?t=40750
  url_to: http://xbox360achievements.org/content/view/101/114/
  anchor: NULL
  from_site_id: 9835
  from_forum_id: 1818
  from_author_id: 282
  from_message_id: 2586
message_published: 2006-09-30 00:00:00
  ...
```

Tackling – one

- Partitioned the data
- 8 boxes, 4x CPU, ~5M links per CPU
- Used Sphinx
- In theory, we could had used MySQL
- It practice, way too complicated
 - Would had resulted in 15-20M+ rows/CPU
 - Would had resulted in “manual” aggregation code

Tackling – two

- Extracted “interesting parts” of the URL when indexing, using an UDF
- Replaced the SELECT with full-text query

```
***** 1. row *****
      url_from: http://www.insidegamer.nl/forum/viewtopic.php?t=40750
urlize(url_from,0): www$insidegamer$nl
                   insidegamer$nl
                   insidegamer$nl$forum
                   insidegamer$nl$forum$viewtopic.php
                   insidegamer$nl$forum$viewtopic.php$t=40750
urlize(url_from,1): www$insidegamer$nl
                   insidegamer$nl
                   insidegamer$nl$forum
                   insidegamer$nl$forum$viewtopic.php
```

Tackling – three

- 64 indexes
 - 4 searchd instances per box, by CPU/HDD count
 - 2 indexes (main+delta) per CPU
- All searched in parallel
 - Web box queries the main instance at each box
 - Main instance queries itself and other 3 copies
 - Using 4 instances, because of startup/update
 - Using plain HDDs, because of IO stepping

Results

- The precision is acceptable
- “Rare” domains – precise results
- “Frequent” domains – precision within 0.5%

- Average query time – 0.125 sec
- 90% queries – under 0.227 sec
- 95% queries – under 0.352 sec
- 99% queries – under 2.888 sec



Questions?



<http://sphinxsearch.com>