



Sphinx

Full-text search in 2010

Andrew Aksyonoff // Sphinx Technologies Inc. // 2010



The Sphinx?

- Stand-alone full-text search engine
- Free, open-source, GPL



<http://sphinxsearch.com>

The talk?

- ...is NOT a quick introduction to Sphinx
 - There is a bunch of tutorials online
 - Getting started is as simple as
`./configure && make install && indexer test1 && search test`

The talk?



- ...going to cover new and shiny stuff
 - “new” as in added since last UC

The features?

- Tier-1
 - You **definitely** want to know about these!
- Tier-2
 - Still killer, but in some scenarios only
- Tier-3
 - Minor niceties



And lets begin with...



And lets begin with... tier-3 actually

- Better stats (eg. per-agent counters)
- Better expression constant optimizer
- Better dupe/zero/... document ID reporting
- FlushAttributes() API call
- Index precaching progress bar
- ...and many many more (SINT(), payload indexing max_batch_queries, searchd -logdebug, indextool --htmlstrip, force_all_words, etc etc etc)

Tier-2 (things **some** care about...)

- `sql_joined_field`
 - When your DB does not have `GROUP_CONCAT`
 - When your DB crawls doing `GROUP_CONCAT`
- `sql_joined_field = tags from query; \`
`SELECT docid, CONCAT('tag',tagid) \`
`FROM tags ORDER BY docid ASC`
 - `ORDER` clause required (for now)

Tier-2 (things **some care about...)**

- Query-based highlighting in snippets
 - Controlled with `query_mode` option in `BuildExcerpts()` API call
 - Switch it on and “John Doe” does not highlight Jane Doe anymore
 - Quirks with fields

Tier-2 (things **some** care about...)



Désirée Palmen / *Zebra* / C-print / 2002 / 30 x 59 inches

Tier-2 (things **some** care about...)

- Blended characters
 - The ones we aren't really sure about
 - AT&T, OS/2, C++, U.S.A.F.
 - Previously, exceptions (case-sens, token-level)
 - Now, `blend_chars` directive (char-level)
 - Caveat, positions (they blend)!

Tier-2 (things **some** care about...)

- Keyword expansion
 - hello → (hello | *hello* | =hello)
 - Internal, does not mess up query-positions
- SPH04 ranker
 - SetRankingMode(SPH_RANK_SPH04)
 - (phrase proximity+BM25) + head/tail match boost
 - Ranks exact field match higher (“podunk” vs “podunk drive” vs “e podunk dr”)

Tier-2 (things **some care about...)**

- Hitless indexing
 - When you don't need keyword positions at all, `hitless_words = all`
 - When you want to speedup some keywords, `hitless_words = mykeywords.txt`
 - Smaller indexes, faster searches
 - No phrase/proximity/... matching, no phrase ranking

Tier-2 (things **some** care about...)

- Common subquery cache
 - q1: "barack obama" california
 - q2: "barack obama" washington
 - q3: "barack obama" texas
- Send these in a batch (using multi-queries)
 - + enable subquery cache (subtree_xxx)
 - = "barack obama" gets reused
 - = PROFIT!!!

Tier-2 (things *some* care about...)



Tier-2 (things **some** care about...)

- Better parallel searches with `dist_threads`
 - Before: `agent = localhost:9312:chunkX`
 - Now: `local = chunkX + dist_threads = 4`
 - Eliminates redundant network roundtrip from searched to itself
 - Works in fork/prefork modes (best of two worlds)
 - Got implicit affinity, explicit TBD

Tier-2 recap

- `sql_joined_field` indexing
- `query_mode` snippet highlighting
- Blended characters (`blend_chars`)
- Keyword expansion (`expand_keywords`)
- SPH04 ranker
- Hitless indexing (`hitless_words`)
- Subquery cache (`subtree_xxx`)
- Better parallel searching (`dist_threads`)

Tier-2 appendix A

- I could probably go on for hours
 - Preindexed attributes (way faster startup)
 - Automatic phantom killer on merge
 - New sorting (from 2-3% to 100x better)
 - NEAR syntax (hello NEAR/3 world)
 - SQL+FS indexing (`sql_file_field = myfilename`)
 - Optimizations (1kw queries, attr copying, Mysql..)
- But we need to cover tier-1 too!

Tier-1 features

- Bad news, there still are 5 of those ahead



- Good news, 4 of 5 are pretty quick

1. MPM

- MPM as in Multi Processing Models
- We only had fork before
- We've added prefork and threads now
 - Helps sys time in cases with many tiny queries
 - Prefork is safer
 - Threads come with a watchdog (work in progress)

2. `indextool --check`

- Index consistency check
- Verifies every single property we could think of and then some
 - Dictionary and doclist/hitlist consistency
 - Attributes consistency and range checks
 - MVA and K-list checks...
- Fun fact, some values are 3x «redundant»

3. New index format

- It's only 70% the previous size now
- This is a feature
- This is **not** a bug (had actual reports...)
- Positions are now stored more efficiently
- Seamless and backwards compatible, as usual
 - Just reindex the data, and you are set
 - Fun fact: we're at index format v.20 actually

4. String attributes

- `sql_attr_string` + `sql_field_string` directive
- Storage only (for now)
 - ie. no `WHERE/GROUP/ORDER` yet, that's TBD
- Kept in RAM, same as other attributes
- Still, let you avoid hitting the database at all!

4. String attributes cont'd

- When it's possible to avoid the DB
 - String values can now be stored and fetched
 - Equality checks (WHERE col=X) and GROUP BY can be emulated with CRC32 trick

```
sql_query = SELECT ... col, CRC(col) AS colcrc ...  
sql_attr_string = col  
sql_attr_uint = colcrc
```
 - So unless you're sorting search results by strings and/or doing range checks, DB hit can be avoided



5. RT Indexes

- A new index backend
- Lets you update data on the fly
- Formally, they are “soft-realtime”
 - As in, most of the writes are very quick
 - But, not guaranteed to complete in fixed time
- Indexing-wise, just INSERT rows (literally)
- Searching-wise, transparent to the app

RT vs RAM

- Updates get accumulated in RAM
- Then they get flushed to disk (freeing RAM)
- Searches combine disk chunks + RAM chunk
- RAM use is **strictly controlled**
 - No data inserted, no RAM used
 - At most `rt_mem_limit` RAM user per index
 - Global limit (`rt_buffer_pool`?) also planned, TBD

RT indexing

- You ship the data to searchd over SphinxQL
 - `mysql_connect()` to Sphinx instead of MySQL
 - `mysql_query()` and do INSERT/REPLACE as usual
- INSERT-ed rows get into the ~~purgatory~~ temporary accumulator, lock-free
- And then into the RAM chunk on COMMIT (and yes, we have COMMIT and ROLLBACK)

RT searching

- Transparent to the calling application
- Any of the access methods work as usual
 - SphinxAPI
 - SphinxSE
 - SphinxQL

What about performance?



RT indexing performance

- Single-row INSERT averages at $\sim 0.5-2$ ms
 - Maxes out at a few seconds on flushes
 - Measured with ~ 1 KB row and no binlog, YMMV
- Multi-row INSERTs are faster
 - With 1000+ rows/batch, you get very close to regular disk based indexer speed
- We also have binlogs now, for crash safety

RT searching performance

- Theoretically – close to regular indexes
 - RAM chunk almost as quick on tests
 - Disk chunks **are** regular indexes
 - Fragmentation effects expected on bigger indexes (not seen in the wild just yet)
- Reportedly slower on write-heavy workloads
- This is alpha though... optimizations TBD



I'm all sold; where to get it?!

- Pull 1.10 from the SVN trunk
<http://code.google.com/p/sphinxsearch>
- Used in production, actually
- Mostly documented, even
- Packages (tarball etc) to follow, soon (tm)

The roadmap?



The roadmap?

- Mostly revolves around RT indexes
 - Finalizing 1.10 alpha release
 - More RT testing, profiling, and optimizations
- Next big thing? Most likely, that would be replications
 - Plural intended
 - From MySQL/Drizzle/...
 - Between Sphinx nodes

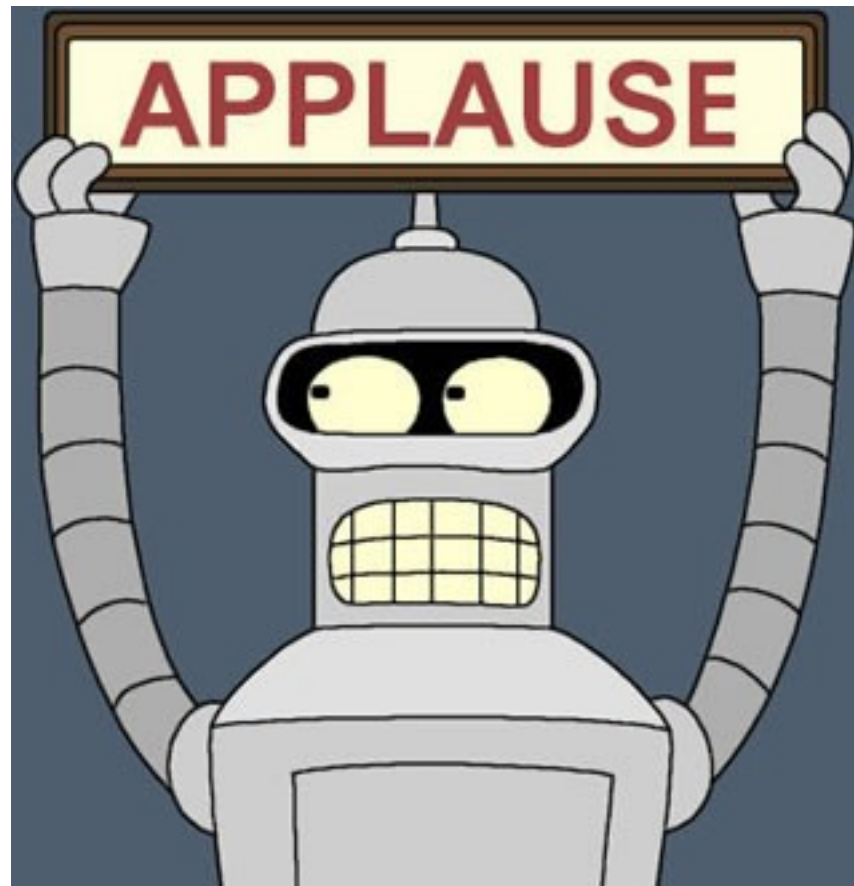
The roadmap?

- Generally turning into a database

ORAPHINX

- One that **can** still do full text search though

The end





Questions?

