Sphinx and search for DB

Egypt, Percona team meeting

Vadim Tkachenko,
Percona Inc

Originally by Andrew Aksenoff for HighLoad.ru

Search in DB – why?

- Popular RDBMSs are not so good in searching (in wide meaning of word “search”)
- In particular with FullText search
  - Performance of Search
  - Performance of UPDATE of tables
  - relevance
- In some cases there are problems with “usual” selects
  - Details later

Search in DB – how?

Search in DB – how?

- **Sphinx**
  - FullText engine
  - Free
  - OpenSource (GPL)
  - Originally designed to integrate with RDBMS
- What we will speak about?
  - Short review of internals
  - Short review of features
  - Short review of practical usages

Some numbers

Some numbers

- ~3.4 M rows, ~5 GB data (Wikipedia)
- MySQL
  - 286 ms/q “default”, 24 ms/q boolean, 3692 ms/q phrase
- Sphinx
  - 22 ms/q “default”, 13 ms/q boolean, 21 ms/q phrase
- where, order by, count, group by, limit, phrases – MySQL response - to 3-26 min(!)/q
- See details in Peter’s presentation on EuroOSCON’2006

Sphinx – general overview

PERCONA
Performance Consulting Experts

Sphinx – general overview

- Two main executables:
  - indexer, gets data from sources (DB) and builds FullText index
  - searchd, responses on FullText queries using built indexes
- Set of APIs for different languages
  - Lightweight native client to searchd
  - PHP, Python, Ruby, Perl, Java…
- SphinxSE
  - Client built in MySQL
  - Storage engine for MySQL 5.0.x and 5.1.x

Sphinx – Terms

- Sphinx documents are equal to records in DB
  - Document is set of text fields and number attributes + unique ID – similar to row in DB
  - Set of fields and attributes is constant for index – similar to table in DB
  - Fields are searchable for FullText queries
  - Attributes may be used for filtering, sorting, grouping

Sphinx – setup and usage

Sphinx – setup and usage

- Setup
  - ./configure && make && make install
  - vi sphinx.conf
  - ./indexer myindex
  - ./searchd
- Usage
  - include (“sphinxapi.php”);
  - $cl = new SphinxClient ();
  - $res = new $cl->Query (“test”, “myindex”);

Sphinx – indexes internals

Sphinx Moth

Sphinx – indexes internals

- FullText indexes are stored in File System
- Has special format
- Monolithic format, update is done only by built from “zero”
- Problem, but solvable
  - Fast speed of indexing (4-8 MB/sec)
  - Partitioning indexes to decrease “lag”
  - Indexes are not fragmented
  - Attributes are updatable in real-time

Partitioning in details

- Simple case: 2 indexes for indexing the same “table”
  - Rarely built “main” index, contains biggest part of data
  - Often built “delta” index, contains updates since time of last build of “main”
  - Often such scheme conforms to changes of data
  - Forums, blogs, mail/news archives…

Data sources

Data sources

- In general case “from everywhere”
  - Battery included “drivers” for MySQL, Postgres, XML with special formats
- Different types of sources can be combined in single index
- Data for indexing is fetching not from table, but from ***selects***
- That is it is possible to make additional handling in time of select – JOINs, subset of fields, etc.

Ranging by phrases

- Usual way – ranging only by word statistics
- On big collections or non-rare keywords – does not work
- Main factor of Sphinx ranging – max length of coincidence of part of phrase in query with document
- Exact quotations are ranged higher, that is quality of search “in general” better (subjective)
- Room for optimization – quorums, frequencies…

Attributes

- Unlimited number of numeric (1-32 bit unsigned integer, float) attributes
- Need for effective filtration, sorting, grouping of search results
- Otherwise (in case of storing inside DB), with10-100K found documents – very slow
- Extern, Inline, MVA

Sorting and filtering

- Sorting
  - By combination of attributes (“@weight desc, user_rank desc”)
  - Always optimized taking into account top-N found documents
- Filtering
  - 10+ times faster than on DB
  - 3 kind of filters: 1) set of values, 2) range of values, 3) geodistance
  - Unlimited count of filters
  - Exclude filters

Sorting and filtering

Grouping

- Aggregated information
- That is the reason for grouping
- Important difference – can be non-**exact**!
  - Performed in fixed memory
  - Between distributed nodes only aggregated information is passed
  - Practically exact value can be reached
- Often exact values are not so important

Distributed search

- Last but not least…
- Distributed indexes allow to run parallel search on several servers
  - Run remotes queries
  - Search on local indexes
  - Read remote results
  - Merge results and return to client
- Very useful for multi-core systems

Example 1.1 – “simple” search

- **Mininova.org**
- Sphinx replaced MySQL FT slaves
- 300-500K rows, 300-500 MB, 4-5 M q/day
- Prefix indexes are used, not full words
- Long query tail – top-N queries in cache on client side
- 2 servers with full mirror of indexes
- Loadavg (now) 0.3-0.4

Example 1.2 – “simple” search

- **Boardreader.com**
- 1B rows, 1.5 TB, 700K-1M q/day
- 6 servers, 4xCPU + 16 GB RAM + 0.5 TB
- 4 copies of searchd instead of signle (startup, HA)
- 4 “usual” HDD instead of RAID (faster!)
- Active partitioning by time
  - Different indexes 1-week, 3-month, all-time
- Distributed “hot” upgrade to new version of Sphinx
  - Back-compatibility is critical
  - In single point of time only ~1/24 of data is not available (P.S. only if new admin does not stop all servers at once)

Example 2 – “simple” select

- **Sahibinden.com**
- 400K rows, 500M, 3M q/day
- 99 fields, MySQL indexes are problematic
- Sphinx is used for FT search and for “usual” selects
  - Faster and handier than MySQL…
  - Frequent fast reindexing, 9-15 seconds per each 1 minutes on single of each 8+ CPUs
- PHP API overhead is noticeable
  - Two indexes, 34 and 99 attributes

Example 3. Grouping

- **Boardreader.com**
- Reports by links – top domains, links to/from, etc
- 150-200 Mrows, 50-100 GB data, 60-100 Kq
  - Queries using GROUP BY and SORT BY COUNT / COUNT DISTINCT
  - Big count of rows (1-10 M) to handle, small result set
  - Exact result is not required
- MySQL executed by 300 seconds
- Paralleled by Sphinx

Example 3. Grouping (cont.)

- Grouping on 6x4 CPU cluster
  - In parallel with search load
  - Distributed, fast, almost exact
- Pre-handling of indexing data
  - MySQL UDF to select all “interesting” substrings into different “words”
  - COUNT DISTINCT
  - 64-bit ids
- Room for optimization
  - Relevancy is not important…

Example 4. MVA selects

- **Grouply.com**
- N million of messages, M tags
- Interesting case for MVA

Conclusions

Conclusions

- Sphinx is not ideal ☺
- But, the wide area to use

- **http://sphinxsearch.com/**
- Author – **shodan@shodan.ru**