Day 1 - Lucene Solr Workshop/Conference October 7-8 2010 Boston

History

- Lucene developed by Doug Cutting 1999 Lisp programmer
- become Apache open source project
- Solr emergent from a CNET development
- Lucene/Solr merged
- Netflix uses Solr

Stick to version 3.1 of Lucene until 4.0 it comes out; 4.0 is still being developed

Definition: reversed-index (inverse of a document) switch key and values, change [terms -> documents] to [documents -> terms]

Important Classes

- DataImportHandler
- RequestHandler
- SearchHandler
- ResponseHandler

QueryHandler

- Standard QueryHandler uses the default Lucene Query Language
- can be changed to use a different Query handler, e.g. DISMAX, see http://wiki.apache.org/solr/DisMaxRequestHandler
- Multiple RequestHandlers can be defined.

ResponseHandler

- */select* in the URL indicates request that the *select* RequestHandler is being used
- returns nested key value pairs, supported formats include JSON, PHP, RUBY, and XML
- the *query time field* only shows the search time. To fetch the total end-to end time, get it from tomcat log files
- version field shows the version of the XML protocol used, e.g. <str name="version">2.2</str>
- only stored fields are returned back from the result set
- for PHP call EVAL to get PHP objects back. Here is an example

```
$code =
file_get_contents('http://localhost:8983/solr/select?q=iPod&wt=php')
;
eval("\$result = " . $code . ";");
print_r($result);
```

Searching Data

- *fq =* filter query only boolean (does not score results)
- filter queries are executed before the standard query *q=* is being executed
- filter queries can be cached and cache size can be adjusted using *filterCache*
- specify *fs* to specify/restrict fields to return
- use *start*/*rows* for paging (beware of crawlers that page through)
- use *indent* option to make XML output more readable
- *sort =* sorting (works only for indexed fields)
- *debugQuery=true* shows detailed scoring information for each hit
- *q:apple pie* translates to apple OR pie (depending on the default operator)
- use parenthesis to group text, e. g. *fruit:(banana AND pineapple)*
- single characters*?*
- multiple characters ***
- fuzzy searches ~
- dynamic term boosting *q=solr^4.0 OR lucene* (default score 1.0)
- *q=title:"crizzly bear"~10* searches for both words within distance 10
- *filterQueries* are cached
- *rangeQueries* use [] for inclusive {} for exclusive range, e.g. [10 TO 12] to fetch 10,11,12 and {10 TO 13} to fetch values 11 and 12

Fields

- use *trieFields* instead of sortable fields (sint,sfloat,etc.), e.g. tfloat, tdouble (null values are not allowed)
- date format: yyy-MM-ddTHH:mm Z=designates UTC time

Query Parser

- use *defType* to specify the *QueryParser* to use
- extend *QParserPlugin*
- two main query parsers
- *defType=lucene* (if it does not like it throws exception, very powerful)
- defType=dismax (the idea is to not bother the user with complex query syntax)
    - dismax ranks fields individually and search across all of them
    - dismax uses max rather than sum for scoring

DISMAX Configuration

- Specify URL parameters in the solconfig.xml XML
- set arg to *invariant* to allow no overrides
- combined weighted search *text^0.5 title^2.0 filename=^0.1*
- use function query to bias the scoring function

- boost factor *bf* generally is based on text string but may be used for numeric types e.g. to integrate product margin, popularity, etc.
- phrase score *ps=10* (within distance of 10 words)
- defType=boost (BoostQParser)

Constant-Score Searches

- *:* sort documents by ascending id order
- range based query returns the same score
- sorting, tiebreaking is done by number of document IDs

Indexing

- use UpdateRequestHandler to import data from XML, CSV files
- Use TIKAT to import Word/PDF documents
- use DataImportHandler for getting data from RDBMS, RSS, Email Server
- use curl or use stream.body=<ENTER-XML> ; works only if on localhost ,e. g. http://localhost:8983/solr/update?stream.body=%3Ccommit%3E%3C/commit%3E
- each segment has multiple files
- segments are not changed until merges or optimized
- adds segments after segment (named 0, 1 ,etc...)
- Optimize is merging segments into single ones
- Use default merge policy (is less critical for Lucene 2.9)
- Segments are never updated (deleted or added)
- You may set the merger policy to merge segments if larger than xyz documents etc.
- Important Classes: *IndexReader*/*IndexWriter*
- Merges might be extensive in terms of disk/IO processing. Avoid this by doing merges/commits at night or use slave servers for production search and master servers as merger/committers.
- Use *RamBufferSize* to specify the size of a single segment (size to hold in memory before flushing)
- run *commit* to detect/activate new segments/data (re-initializes *IndexSearchers* that hold a view of the index in memory)
- commit causes a last flush from memory to disk
- optimize *maxSegments* to specify the number of segments after an *optimize*
- optimize squeezes out empty space (if deletions have occurred)
- boost values can be specified for each document within the XML file, e. g. *<doc boost="3.0">.* Score is transferred to all fields in the document.
- boost values can be specified for a field: *<field name="content_type" boost="2.0"*
- import csv files *curl '<host>:<port>/solr/update/csv --data-binary @data.csv -H 'Content-type:text/plain; charset=utf-8'*
- use TIKA/Solr Cell to import documents
- use SolrJ for parsing TIKA/Solr cell results

- Lucene doc IDs are not persistent
- no fields updates only document updates (partial updates in preparation)
- to delete entry use curl/stream body or java client, e. g. http://localhost:8983/solr/update?stream.body=<delete><id>05991</id></delete>
- after add/delete/update do a commit

Requirement Analysis

- How fast does the stuff need to show up ?
- Who are the users ?
- What languages do they speak ?
- How many people will be using it?
- How large will your index be?
- Replication for data protection?
- Store user queries?

Development Suggestions

- it is really hard to find something in crappy data
- acquiring and cleaning the data is the hardest part
- get inside their head/level of sophistication
- use Google as example (user behavior) one simple search textbox
- advanced search pages are almost never used
- most search interfaces are overly rich, overdeveloped, too complex for average user, don't spend too much time before having a use case, use parenthesis wisely
- profile data/searches to optimize your system
- stemming increases recall but may reduce precision
- interesting terms can be adjusted with payloads
- always check the logs for issues

Open Source Contribution

- do a *svn --diff* to generate a patch an upload to JIRA
- recently a IntelliJ project has been created that integrates Lucene and Solr
- if you write patches, make patches upward compatible (working for all versions)
- download source code and compile with ant, add print statement to see response "break the ice"
- don't wait to be an expert to join in
- join mailing list dev@lucene.apache.org

Day 2 - Lucene Solr Workshop/Conference October 7-8 2010 Boston

Analyzer Configuration

- use analyzer to specify indexing and querying methods
- add <analyzer>XYZ</analyzer> to solrconfig.xml
- <analyzer type="index">XYZ</analyzer>
- <analyzer type="query">XYZ</analyzer>

Data Import Handler (RequestHandler)

- pull instead of push
- http://localhost:8983/solr/admin/dataimport.jsp?handler=/dataimport
- full import or delta import
- extendable to other sources
- multiple data import handlers available
- configure *Data Sources*, *Transformers*, *Entities*
- declarative configuration file (instead of logic embedded in code)
- ideal for rapid prototyping; pull data from client SQL database into Solr for demonstration "Let me show how to do stuff in Solr"

Faceting

- can be fairly memory expenses (depending on the size)
- incrementally build query (add filter queries)
- Ideal for exploring data (what is the most popular, etc.)
- user expects to see all facets
- it is too challenging to show all unique values if many
    - show an "other" category that bins the remaining facets
    - show pop up with searchable paginated results
- *facet.sort=lex* for sorting alphabetically, e.g. cities
- you can specify facet params for fields separately, e.g. <fieldName><facetParam>=<value> *f.price.facet.method=fc* (use *fieldCache*)

Cache

- Field Cache Memory Footprint
    - *[nTerms*(56+(avg_string_length*2)+4] + nDocs*4*
- ENUM Cache
    - bit set (spanning all documents) for each unique value
    - unique_value = bitmap[0][1][0][...]
    - nTerms*(nDocs/8) + nTerms+56+(avg_string_length*2)+4

Other Features

Spell Checking

    - configure in solrconfig.xml
- Highlighting
    - http://localhost:8983/solr/select/?q=NASA&version=2.2&start=0&rows=10&hl=true&hl.snippets=2
- Auto-Completion (get terms from text)
    - http://localhost:8983/solr/terms/?terms.fl=title&terms=true&terms.prefix=b
- Multi-Select Faceting
    - use as filter query (Lucene) after clicking
    - fq={!tag=myTag}project:lucene&facet=on&facet.field{!tag=myTag}=project

Scoring

- Term Frequency (frequency in the document) x Inverse Document Frequency (inverse frequency of the term across the index collection) [TF x IDF]
- Set debugQuery=T to get information about how each document has been scored

Relevance

- Create sets of queries with known results for evaluating relevance
- do A/B testing (compare two distinct implementations/user groups with one another)
- use Luke to poke into existing index files
- use analysis.jsp to see term frequencies, etc.

Replication

- example set-up 1x master for indexing 2x slaves (load balanced) for querying (no resource interference between indexing and querying)
- if you index and query on the same machine, use separate disk for indexing and writing
- if many users index/modify data use-auto-commit instead of explicit commit
- replication is logged

Sharding

- for queries that take a long time, load balancing does not help. Split data into shards.
- score is affected as document frequencies (IDF) are calculated separately for each shard
- distribute data randomly to avoid bias
- each server can talk to any other
- set up one dedicated shard aggregation machine that fetches data from all others
- to split existing index, you need to re-index
- use latest Sun Java Virtual Machine
- monitor time on the JConsole
- avoid using NTFs for production (locking issues). Lucene does many seeks - the faster seek and read the better)

Cores

- used for datasets with varying schemas
- alternatively set-up multiple Solr web applications
- cores > 1000 can be slow (improvement is on the way)
- search across cores can be done on the fly like a shard
- date faceting does not work across cores

Schema

- avoid schema.xml changes (new fields are OK)
- try to be more inclusive in the beginning
- specify required fields in the schema.xml (required=true)

Diverse

- Microsoft FAST discontinued (for Linux)
- each shard gets own VM
- only 8 threads per CPU due to memory controller limitation
- consider Texas Memory Systems (RamSan) for in-memory solution
- stemming has 6-steps (first 2 are good, removes plurals, no parameter yet to use only a subset)
- geo-spatial search not yet ready for prime