About me...

• Co-author of “Lucene in Action” as well as co-author of “Java Development with Ant”
• Active member of the Lucene community – a leading Lucene and Solr committer, member of the Lucene Project Management Committee, member of the Apache Software Foundation
• Frequent speaker at various industry events.
• Co-founder of LucidWorks, dedicated to Lucene/Solr support services.
Agenda

- Introducing Solr and Lucene
- Installing and running Solr
- Indexing your data
- Configuring the schema
- Surveying Solr’s features
- Tuning and scaling
- Integrating into your applications
Introducing Lucene/Solr
Relevance

Vector Space Model (VSM) for relevance
• Common across many search engines
• Lucene is a highly optimized implementation of the VSM

Indexing

Finds and maps terms and documents
• Conceptually similar to a book index
• At the heart of fast search/retrieve

$d_j = <w_{1,j}, w_{2,j}, \ldots, w_{n,j}>$
$q = <w_{1,q}, w_{2,q}, \ldots, w_{n,q}>$

$w =$ weight assigned to term
Apache Lucene

- [http://lucene.apache.org/java](http://lucene.apache.org/java)
- Java based Application Programming Interface (API) for adding search and indexing functionality to applications
- Fast and efficient scoring and indexing algorithms
- Lots of contributions to make common tasks easier:
  - Highlighting, spatial, Query Parsers, Benchmarking tools, etc.
- Most widely deployed search library on the planet
Lucene Basics

• Content is modeled via Documents and Fields
  - Content can be text, integers, floats, dates, custom
  - Analysis can be employed to alter content before indexing

• Searches are supported through a wide range of Query options
  - Keyword
  - Terms
  - Phrases
  - Wildcards
  - Many, many more
Apache Solr

- http://lucene.apache.org/solr
- Lucene-based Search Server + other features and functionality
- Access Lucene over HTTP:
  - Java, XML, Ruby, Python, .NET, JSON, PHP, etc.
- Most programming tasks in Lucene are configuration tasks in Solr
- Faceting (guided navigation, filters, etc.)
- Replication and distributed search support
- Lucene Best Practices
Lucene/Solr-Powered (small sample)
Example - /browse

Find:  

Boost by Price

Field Facets

cat

Electronics (14)  
Memory (3)  
Connector (2)  
Graphics Card (2)  
Hard Drive (2)  
Monitor (2)  
Search (2)  
Software (2)  
Camera (1)  
Copier (1)  
Multifunction Printer (1)  
Music (1)

Test with some GB18030 encoded characters More Like This

Price: $0.00

Features: No accents here 这是一个功能 This is a feature (translated) 这份文件是很有光泽 This document is very shiny
In Stock: true

Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133 More Like This

Price: $92.00

Features: 7200RPM, 8MB cache, IDE Ultra ATA-133 NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB)
In Stock: true
Other Features

• Data Import Handler
  - Database, Mail, RSS, etc.

• Rich document support via Apache Tika
  - PDF, MS Office, Images, etc.

• Replication for high query volume

• Distributed search for large indexes
  - Production systems with 1B+ documents

• Configurable Analysis chain and other extension points
  - Total control over tokenization, stemming, etc.
Why Solr?

• Specialized tools do the job better
  - Solr performs much better, for text search, than a relational database
  - Solr knows about languages
    » E.g. lowercasing ΟΔΥΣΕ´ΥΣ produces οδυσεύς
  - Solr has features specific to text search such as highlighting
Where does Solr fit?

Front End (Ruby, PHP, Java, ...)

Solr

Data Source

Data Source 2

Data Source 3

Relational database

web, net, filesystems
Solr block diagram

- Query
- Response
- Request Handler
- Response Writer
- Index
- UpdateHandler
- Data Source

Solr
Lucene
Installing and running Solr
Installing and running Solr

• Download Solr
  - from http://lucene.apache.org/solr
• Unzip (un-tar) distribution
• cd solr/example
• java -jar start.jar
• http://localhost:8983/solr
Tutorial and Reference Guide

- solr/site/html/tutorial.html
- and refer to the new Apache Solr Reference Guide
Indexing your data
Indexing Content

- Incoming data
- External pipeline
- Update handler
- Analysis
- Index
Update Handling

Update Handler

UpdateRequestProcessorChain

UpdateRequestProcessor

UpdateRequestProcessor

UpdateRequestProcessor

UpdateRequestProcessor
UpdateRequestProcessors

• Can modify incoming documents
  - Add fields
  - Modify fields
  - Delete fields

• Gives you a common point across multiple sources
Indexing options

• **Solr XML, JSON, or CSV**
  - POST to /update
  - Set Content-Type header appropriately
  - Arbitrary XML can be transformed via XSLT

• **Rich documents**
  - Word, PDF, HTML, and many other formats
  - Uses Apache Tika for parsing/extraction
  - Again, set Content-Type header appropriately

• **DataImportHandler**
  - Relational databases (via JDBC)
  - RSS, Atom, XML
  - Plain text
  - E-mail, IMAP
  - Allows scripting, transformations, blending multiple data sources
post.jar (SimplePostTool)

• Under example/exampledocs
• Easy way to start indexing content of all varieties
  - For details: java -jar post.jar -help
• Solr XML
  - java -jar post.jar *.xml
• Solr JSON
  - java -Dtype=application/json -jar post.jar *.json
• CSV
  - java -Dtype=text/csv -jar post.jar books.csv
• File system
  - java -Dauto -Drecursive -jar post.jar ../..site/html/
• Web crawl
  - java -Ddata=web -Drecursive=1 -Ddelay=5
    -jar post.jar http://www.lucidworks.com
Adding documents to Solr

• Solr can process XML documents in its own format
  - and many other formats too

```xml
<add>
  <doc>
    <field name="id">SP2514N</field>
    <field name="name">Samsung SpinPoint</field>
  </doc>
  <doc>
    <field name="id">NG123</field>
    <field name="name">iPod Touch</field>
  </doc>
</add>
```
Configuring the schema
Documents and fields

• Documents
  - What you find when you search
  - A collection of fields

• Field
  - A name
  - A value
    » Can have multiple values, or no value
  - A type
Field attributes

- **Type**
  - int, float, date, location, string, text

- **Required**
  - Force every document to have this field

- **Indexed**
  - Can be searched (also sorted and faceted)

- **Stored**
  - Can be returned to client
  - Highlighting requires field to be stored

- **Multi-valued**
string versus text

• String is one term, even if multiple words
  - "Portland, Oregon"
  - Not changed from what Solr received

• Text is usually split up into multiple terms, and manipulated
  - portland
  - oregon
  - May be considerably changed from what Solr received
    » The Running Shoes => run shoe
Dates

• Date format (UTC, always, everywhere)
  - 1995-12-31T23:59:59Z

• Date math
  - NOW
  - <field name="timestamp" type="date" indexed="true" stored="true" default="NOW" />
  - NOW/TIMESTAMP
    » 2010-07-28T23:34:45Z/YEAR -> 2010-01-01T00:00:00Z
  - NOW/HOUR, NOW/SECOND
  - NOW/YEAR-1YEAR+2DAYS
Dynamic fields

• Allow indexing of content with fields not mentioned explicitly in the schema
• Field names must match a pattern
• Pattern has a wildcard at the start or end:
  - `attribute_*`
  - `*_s`
• Can be used to implement a somewhat "schema-less" index
  - Using the dynamic field definitions found in the default schema.xml
Schema used during indexing and querying

Schema controls analysis

Index

Query

Data

Analysis

Analysis
Analyzers

• An analyzer processes the text for a field
• Each TextField type has its own analyzer
• An analyzer is a combination of other classes
  - CharFilter
  -Tokenizer
  - TokenFilter
Character filters

```xml
<fieldType name="text_ws" class="solr.TextField">
  <analyzer>
    <charFilter
      class="solr.MappingCharFilterFactory"
      mapping="mapping-ISO8859_1Accent.txt"/>
    <tokenizer class="solrWhitespaceTokenizerFactory"/>
    <filter class="solrLowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

He went to the café.

He went to the cafe.
He went to the cafe.
He went to the cafe.

Token filters

```xml
<fieldType name="text_ws" class="solr.TextField">
  <analyzer>
    <charFilter
      class="solr.MappingCharFilterFactory"
      mapping="mapping-ISO88591Accent.txt"/>
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```
Language handling

• What if you have more than one language?
  - One field
  - Multiple fields

• Stemming
  - built-in Snowball and other stemming algorithms

• Whitespace
  - Not all languages use it
  - Bi-grams
    » お元気ですか  becomes
    » お元気  気で  ですか  ですか
  - Dictionary based tokenization
    » お  元気  ですか
The copyField directive

• Take data for one field and also put it in another
• Allows the same input data to be indexed and handled different ways
  - stemmed / unstemmed
  - string / text (facet / full text search)
  - aggregate multiple fields into single searchable field

```xml
<copyField source="name" dest="text"/>
<copyField source="author" dest="text"/>
<copyField source="summary" dest="text"/>
```
"schema-less"?!

• No such thing really
  - A field is always of a certain type

• Solr now has capabilities making it easier to start from scratch though:
  - REST API to add fields and copyFields
  - Update processing can optionally create new fields, and guess their type
Surveying Solr’s features
Update operations

• Documents can be "updated" (replaced)
  - entirely or atomically: currently same thing underneath

• Documents can be deleted
  - by unique key
  - or by query

• Commits
  - soft commits
  - hard commits
Deleting content

• POST to /update

```xml
<delete>
  <id>ID01</id>
</delete>

<delete>
  <query>category:electronics</query>
</delete>

<delete>
  <id>ID01</id>
  <id>ID02</id>
</delete>
```
Simplified JSON delete syntax

• Single delete by id
  - {"delete":"book1"}

• Multiple delete by id
  - {"delete":["book1","book2","book3"]}

• Delete with optimistic concurrency
  - {"delete":{"id":"book1","_version_":123456}}

• Delete by query
  - {"delete":{"query":"tag:category1"}}
Updates in Solr

• Replace, not in-place update
  - It marks the old version, based on unique key, of the document as deleted
  - It adds the new version of the document
Atomic updates

• Since 4.0, Solr can "update/modify" fields in documents
• For this to work, all fields in your document must be stored, not just indexed
• Underneath, this is a delete and re-add using existing values

```bash
   {
      "id" : "book1",
      "pubyear_i" : { "add" : 2001 },
      "ISBN_s" : { "add" : "0-380-97365-1"}
   }
]
   {
      "id" : "book1",
      "copies_i" : { "inc" : 1},
      "cat" : { "add" : "fantasy"},
      "ISBN_s" : { "set" : "0-380-97365-0"}
      "remove_s" : { "set" : null } } ]
```
Solr search request response

• When you send a request, Solr sends back a response. Search requests respond with:
  - Success / failure
  - Time to process query
  - Number of matching documents
  - The first N documents
  - …and more, depending on the options specified
Response writers

• Response writers format the output from requests
• Many types
  - XML, JSON, Ruby, PHP, CSV, Java (binary)
  - …and more
• &wt=json
The Solr Search URL

  - q=ipod
  - &fl=id,name (only return specified fields)

- Keep URL encoding in mind!
Filter queries

• Limit the possible responses to the main query
• Do not change ordering or scoring
• Can be based on any query type
  - &fq=category:music
  - &fq=price:[0 TO 100]
  - &fq=rating:[3 TO *]
Sorting results

• Solr can sort by
  - Score
  - A value in a field
  - A function
• In ascending or descending order
• Multiple fields
  - &sort=price asc,manu_exact desc
Query Parsing
Query parsers

• How do query objects get created from text?
  - Lucene Query objects are created by a query parser
  - Solr has many query parsers available, and allows new ones to plug in

• Query parsers can generate simple queries or complex ones

• See also:
  - http://www.slideshare.net/erikhatcher/solr-query-parsing

• Default query parser: "lucene"
  - Uses the historical Lucene query parser syntax
Lucene queries

• Boolean queries
  - +this –that
  - this AND that

• Field queries
  - title:this
  - description:that

• Fuzzier queries
  - It starts with pro: pro*
  - It ends with tion: *tion
  - I'm not sure of the second letter: c?t
  - It's something like steve:
    » steve~
  - It matches a regular expression:
    » /Ap.*e/ matches Apache and Apple

• Range queries
  - Ranges with [ ] are inclusive at both sides
    » price:[0 TO 92]
    » price:[0 TO *]
    » rating:[3 TO *]
  - Range queries with { } are exclusive (4.0+)
    » price:{0 TO 92}
    » Can use both { and ]
      • price:[0 TO 92]

• Phrase/proximity queries
  - "Harry Potter" matches only Harry Potter
  - "Harry Potter"~2 matches Harry Potter and Harry James Potter
• Simple constrained syntax
  - "supports phrases" +requiredTerms -prohibitedTerms loose terms
• Spreads terms across specified query fields (qf) and entire query string across phrase fields (pf)
  - with field-specific boosting
  - and explicit and implicit phrase slop
  - scores each document with the maximum score for that document as produced by any subquery; primary score associated with the highest boost, not the sum of the field scores (as BooleanQuery would give)
• Minimum match (mm) allows query fields gradient between AND and OR
  - some number of terms must match, but not all necessarily, and can vary depending on number of actual query terms
• Additive boost queries (bq) and boost functions (bf)
• Debug output includes parsed boost and function queries
edismax: extended dismax

• "An advanced multi-field query parser based on the dismax parser"
  - Handles "lucene" syntax as well as dismax features
• Fields available to user may be limited (uf)
  - including negations and dynamic fields, e.g. uf=* -cost -timestamp
• Shingles query into 2 and 3 term phrases
  - Improves quality of results when query contains terms across multiple fields
  - pf2/pf3 and ps2/ps3
  - removes stop words from shingled phrase queries
• Multiplicative "boost" functions
• Additional features
  - Query comprised entirely of "stopwords" optionally allowed
    » if indexed, but query analyzer is set to remove them
  - Allow "lowercaseOperators" by default; or/OR, and/AND
Faceting
What is faceting?

“Men Shoes” We found 14504 items!

NARROW YOUR CHOICES

YOUR SELECTIONS: SHOES  MEN

NEAREST  MOST POPULAR  NAME  LOWEST PRICE  HIGHEST PRICE  CUSTOMER RATING

CATEGOR

Sneakers & Athletic Shoes (7120)
Boots (1875)
Loafers (1750)
Oxfords (1455)
Sandals (1384)
Boat Shoes (398)
Clogs & Mules (231)
Slippers (231)
Insoles & Accessories

OCCASION

Casual (8803)
Athletic (3709)
Outdoor (1655)
Dress (1489)
Office & Career (1155)
Action Sports (1067)
Work & Duty (406)
Wedding (3)
Other (272)
What can you facet on?

• Values in fields
  - Strings
  - Date and number ranges, automatically
  - Can have one value, or multiple
    » A book might be both mystery and historical fiction
    » It will be counted in each category
  - Pivot multiple fields across one another

• Queries
  - Range queries are most common for faceting by query:
    » This year (Jan 1 to Dec 31), last year, the year before...
  - Or saved searches could become facets, for example

• You may need the data in a particular format
  - You don't want the words in the field separated
  - "string versus text"
Numeric and date range faceting

• Range faceting
  - Range facets divide a range into equal sized buckets*.
  - facet.range.start=100
  - facet.range.end=900
  - facet.range.gap=200
  - facet.range.other=before
  - facet.range.other=after
Geospatial

• Solr provides a number of geospatial features
  - Limit search to a region
  - Affect score by distance
Geospatial examples

• Index shapes other than points: circles, polygons, etc.

• Indexing:
  - "geo":"43.17614,-90.57341"
  - "geo":"Circle(4.56,1.23 d=0.0710)"
  - "geo":"POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30))"

• Searching:
  - fq=geo:"Intersects(-74.9 41.4 -69.4 44.5)"
  - fq=geo:"Intersects(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30)))"
More-like-this

• Finds similar documents, based on
  - The contents of document(s) in the index
  - Data provided as a parameter

• Builds & runs a query
Grouping

• Collapses a group of results with the same field value down to a single (or fixed number) of entries
  - only show one result per "site"
  - group search results by "category", showing 3 or so results per category
Spell checking

- Improves findability
- Can suggest from indexed content or external file

Start new search

Search Results for luceene

Found 21 results in 0.081 seconds. Displaying page 1 of 3, sorted by relevancy

Did you mean: lucene
Suggestions

- Improves findability
  - Shares code with spellcheck
Highlighting

Search Results for hit highlighting

Found 578 results in 0.043 seconds. Displaying page 1 of 58, sorted by relevancy.

[lucene-java-user] Hits Highlighting

Sent 2003-11-25 by "Pleasant, Tracy" <tracy.pleasant@...>

Are there any hit highlighting functions? I have a simple one, but it gets complicated with searching multiple words, having tokens, etc.

To unsubscribe, e-mail: lucene-user-unsubscribe@jakarta.apache.org For additional commands, e-mail: lucene-user-help@jakarta.apache.org

http://www.lucidimagination.com/search/document/ac8734ce2b559fc
Pseudo-fields

- Returns other info along with document stored fields
  - Function queries
    » fl=name,location,geodist(),add(myfield,10)
  - Fieldname globs
    » fl=id,attr_*
  - Multiple "fl" (field list) values
    » &fl=id,attr_*
    » &fl=geodist()
    » &fl=termfreq(text,’solr’)
  - Aliasing
    » fl=id,location:loc,_dist_:geodist()
    » fl=id,[explain],[shard]
$ curl http://localhost:8983/solr/query?
q=solr
&fl=id,apache_mentions:termfreq(text,'apache')
&fl=my_constant:"this is cool!"
&fl=inStock, not(inStock)
&fl=other_query_score:query($qq)
&qq=text:search

{ "response":{ "numFound":1,"start":0,"docs":[
{
 "id":"SOLR1000",
 "apache_mentions":1,
 "my_constant":"this is cool!",
 "inStock":true,
 "not(inStock)":false,
 "other_query_score":0.84178084
 }]
}}
Pseudo-join

- Restrict to blogs mentioning netflix:
  - fq={!join from=blog_id to=id}body:netflix

id: blog1
name: Solr ‘n Stuff
owner: Yonik Seeley
started: 2007-10-26

id: blog2
name: lifehacker
owner: Gawker Media
started: 2005-1-31

id: post1
blog_id: blog1
author: Yonik Seeley
title: Solr relevancy function queries
body: Lucene’s default ranking [...]

id: post2
blog_id: blog1
author: Yonik Seeley
title: Solr result grouping
body: Result Grouping, also called [...]

id: post3
blog_id: blog2
author: Whitson Gordon
title: How to Install Netflix on Android
Adding more data

• You may want to have:
  - More than one configuration for Solr
  - More than one data set
Solr Cloud
SolrCloud

• Distributed/sharded indexing & search
  - Auto distributes updates and queries to appropriate shards
  - Near Real Time (NRT) indexing capable

• Dynamically scalable
  - New SolrCloud instances add indexing and query capacity

• Reliable
  - No single point of failure
  - Transactions logged
  - Robust, automatic recover
SolrCloud: Assigning machines

- `-DnumShards=3`
- `-Dbootstrap_confdir=./solr/conf`
- `-DzkHost=<host>:<port> [,<host>:<port>]`
SolrCloud: Assigning machines

- Leader shard1
- Leader shard2
- ZK Host(s)
- `DzkHost=<host>:<port>[,<host>:<port>]`
- At this point you can index and search, you have one machine/shard
SolrCloud: Assigning machines

- Leader shard1
- Leader shard2
- Leader shard3
- Replica shard1

ZK Host(s)
SolrCloud: Assigning machines

ZK Host(s)

Leader shard1
Leader shard2
Leader shard3
Replica shard1
Replica shard2
SolrCloud: Assigning machines

- Leader shard1
- Leader shard2
- Leader shard3
- Replica shard1
- Replica shard2
- Replica shard3

ZK Host(s)
SolrCloud cluster visualization/navigation

How SolrCloud works

• Using intelligent clients
  - SolrJ is currently the only such client
  - Tell SolrJ the address of ZooKeeper, not Solr
  - As a new node is added to shard, ZooKeeper automatically starts sending requests to it
  - When a server fails, ZooKeeper directs clients to other nodes for that shard
How SolrCloud works

• Server failures during indexing
  - When a replica goes down
    » Leader stops sending it updates
  - When a leader goes down
    » A replica is elected to take over the leader role
  - When it a replica recovers
    » If differences are small: do a simple sync
      ▪ Send missing update commands from leader to replica
    » Otherwise: replicate the whole index
      ▪ Send full index from leader to replica
SolrCloud’s capabilities

- **Transaction log**
  - All updates are added to the transaction log. The tlog provides support for: durability for updates that have not yet been committed, peer syncing, real-time get (retrieve documents by unique id) always up to date because it checks the tlog first, does not require opening a new searcher to see changes

- **Near Real Time (NRT) indexing**
  - Soft commits make updates visible
  - Hard commits make updates durable

- **Durability**
  - Updates to Solr may be in several different states: buffered in memory, flushed, but not committed or viewable, soft committed (flushed and viewable), committed (durable)
  - The transaction log ensures data is not lost in any of these states if Solr crashes.

- **Recovery**
  - Solr uses the transaction log for recovery; on startup Solr checks to see if the tlog is in a committed state, if not updates since the last commit are applied

- **Optimistic locking**
  - Solr maintains a document version (_version_ field); updates can now specify _version_; updates to incorrect version will fail
Solr as NoSQL

• Characteristics
  - Non-traditional data stores
  - Not designed for SQL type queries
  - Distributed fault tolerant architecture
  - Document oriented, data format agnostic (JSON, XML, CSV, binary)

• Updated durability via transaction log

• Real-time /get fetches latest version w/o hard commit

• Versioning and optimistic locking
  - w/ Real Time GET, allows read/write/update w/o conflicts

• Atomic updates
  - Can add/remove/change and increment a field in existing doc w/o re-indexing
Tuning and scaling
Relevance Tuning

• Art?
• Science?
• Magic?
• Often a never ending endeavor?
• All of the above
Improving relevance: field weighting

• What's most important?
  - title: potter
  - author: potter
  - description: potter
  - reviews: potter
Improving relevance: phrase queries

• Which is a better match?
  - Query: harry potter
  - Text:
    » Harry was a nice man. He lived on main street, next door to a potter.
    » Harry Potter was a wizard

• Why?
  • dismax and edismax parsers allow treating the query as phrase(s) to boost by proximity automatically
Improving relevance: function queries

• So far: text matching
• What about non-text factors?
  - price, distance, date
Scaling

• Collection size
  - Sharding using SolrCloud

• Query throughput
  - replication
  - load balancing
Integrating into your applications
Solr APIs

- **SolrJ**
  - Solr includes SolrJ (used internally for distributed operations)
  - When in Java/JVM-based environments, build upon SolrJ
- **PHP**
- **Ruby**
- **Python**
- **Perl**
- **.NET / C#**
require 'solr'
#
1.upto(max_pages) do |page|
  puts "Processing page #{page}"
  json = fetch_page(page)

  response = JSON.parse(json, :symbolize_names=>true)
  puts "Total products: #{response[:total]}" if page == 1

  mapping = {
    :id => :sku,
    :name_t => :name,
    :thumbnail_s => :thumbnailImage,
    :url_s => :url,
    :type_s => :type,
    :category_s => Proc.new{|prod|
      prod[:categoryPath].collect{|cat| cat[:name]}.join(' >> ')},
    :department_s => :department,
    :class_s => :class,
    :subclass_s => :subclass,
    :sale_price_f => :salePrice
  }

  Solr::Indexer.new(response[:products], mapping,
    {:debug => debug, :buffer_docs => 500}).index
end
SolrJ searching example

```java
SolrServer solrServer = new CommonsHttpSolrServer("http://localhost:8983/solr");
SolrQuery query = new SolrQuery();
query.setQuery(userQuery);
query.setFacet(true);
query.setFacetMinCount(1);
query.addFacetField("category");

QueryResponse queryResponse = solrServer.query(query);
```
User interfaces

• Built-in VelocityResponseWriter
  - powers /browse
  - for prototyping, proof-of-concepts
  - it's been used in production for producing dynamic e-mails

• Blacklight
  - http://projectblacklight.org
  - "Blacklight is an open source Solr user interface discovery platform. You can use Blacklight to enable searching and browsing of your collections. Blacklight uses the Apache Solr search engine to search full text and/or metadata. Blacklight has a highly configurable Ruby on Rails front-end."

• VUFind
  - PHP-based

• And several other freely available starting points
debug=true

• &debug=true
  - or debug=query|timing|results

• Parsed query
  - See which fields are used, effects of query analysis and boosting

• Search component timings
  - Is highlighting or faceting slowing things down?

• Results explanations
  - Scoring details for each result document
  - Can even provide score explanations for documents not seen in current page of results (explainOther)

• Cool tool provided by community members
  - http://explain.solr.pl/
  - Nice visualizations of Solr score explanations
Prototyping

• Don't overplan/overthink data ingestion and proving Solr out in your environment
• Just Do It
  - ingest your data in simplest possible way
  - fire up quick and not-so-dirty prototypical UI
  - iterate on config/ingestion/UI tweaks
  - go to scale: collection size, ingestion rate, query load
SolrMeter

Lucene Revolution!

• Lucene/Solr conference
• Coming this fall to Dublin, Ireland
• November 4-7, 2013

http://www.lucenerevolution.org
Search Smarter

- Dedicated to the Lucene/Solr ecosystem
- Support, training, and consulting
- Products:
  - LucidWorks Search
    » Connectors (SharePoint, web crawl, etc), security, administration/management
  - LucidWorks Big Data
    » Build on Hadoop and related technologies to provide analytics, machine learning
- http://www.lucidworks.com
- http://www.searchhub.org

© 2013 Search Smarter