lucid IMAGINATION

Programmer's Guide to what's new in Apache Solr / Lucene 3.4

A Lucid Imagination
Technical White Paper

Abstract

With the merging of Apache Lucene and Apache Solr into a single codebase, the release of the new Solr 3.x branch – represented at the time of this writing by Solr 3.4 – brings with it a host of welcome improvements over Solr 1.4 that make search-related applications more powerful, faster, and easier to build. Some new features are completely under the hood, such as performance improvements and improved analysis capabilities. Others will help you improve the usability and presentation of your search application, such as new highlighting capabilities that enable you to visually indicate various levels of relevance, or more easily accessible auto-complete functionality. But perhaps the most exciting are those that open up entirely new doors in terms of what your application can do, such as spatial search and integration with the Unstructured Information Management Architecture, or UIMA.

In this paper, you will learn about:

New search capabilities: Solr 3.4 includes improved query support, including the ability to sort by function queries, as well as improved analysis and new input and output formats.

Performance improvements: Under the hood, Lucene and Solr 3.4 provide better index segment management, as well as providing previously unavailable distributed support for spellchecking.

New options for building search applications: In addition to making things easier for you with features such as range facets and a new Velocity-driven search UI, Solr 3.4 includes a number of exciting new features that open up whole new areas of exploration in terms of building search applications. These include spatial search and the ability to search unstructured information using Apache UIMA.

What to expect in Solr 4: Even as Solr 3.4 hits the net, new features destined for Solr 4 are making waves where they're available, and we'll look at a few of them and how they'll affect you.

Table of Contents

Introduction

Apache Solr, if you're not familiar with it, is a web-application-friendly, enterprise-level search platform built on the powerful Apache Lucene libraries.

The releases of Solr 3.1, 3.2, 3.3 and now Solr 3.4 have brought with them a host of new capabilities, some of which were built for Solr itself, and some of which were built for Lucene, the engine behind Solr. But with the merging of code bases for the two projects, the joint release brings a slew of new features, including performance improvements, new analysis capabilities, and entire functions and features that open up a new world of possibilities for you to consider when building applications based on Solr.

So what are these new features? Well, starting with searching, new capabilities include features such as the ability to sort on functions rather than just fields or relevance, new input and output formats and a host of new analysis capabilities. For example, it is now possible to specify a complex field such as `LatLonType`, which holds latitude and longitude values that can be separately analyzed for spatial search. (More about that momentarily.)

Solr 3.4 also includes a number of performance improvements. Many of these improvements are in areas developers using Solr typically don't touch directly, but still notice, such as faster querying and analysis, and improvements due to Lucene's new segment management techniques. Developers will also find that improvements in 3.4 mean that they can now take advantage of features that might have been unavailable before, such as the ability to use spellchecking in a distributed installation.

But perhaps the most exciting areas of improvement in Apache Solr 3.4 are new features that open up new avenues for development. Some provide a faster and easier way of doing things you might already be doing, such as the new Velocity-driven user interface that enables easier rapid prototyping, or the `Suggester` component, which makes it easier to create autocomplete functionality. Others, such as Spatial Search or integration with UIMA, make available the types of exciting functionality that has previously been out of reach for most developers.

Let's take a more in-depth look at some of these features, starting with those that directly impact the user's ability to search for content.

New search capabilities

The whole point of using Apache Solr is enabling users to find your content, so it's natural that some of the new features in Apache Solr 3.4 focus on new ways to search. These features include new query support, and new ways to analyze the content once the search has been initiated. Let's look at some of them.

Enhanced query support, including sorting by function query

From the user's standpoint, a search begins when he or she enters a query, but from a developer's standpoint, that process can't begin until a parser processes the request.

Solr works with several parsers, and one of the more powerful is the Dismax parser. Unfortunately, the Dismax parser doesn't support some of the more useful and more commonly used features, such as wildcard, fuzzy, or range queries, for example. Solr 3.4 includes the Extended Dismax query parser, or edismax, which not only includes these types of queries, but also supports all of the Lucene syntax users are used to.

Once the user submits the search request, it's up to the developer to decide how to present the data Solr ultimately returns. By default, Solr returns results based on relevance. Because that's not always what you want, it also enables you to specify the sort order based on a specified field, such as a date or a title. But one feature that developers have wanted for some time is the ability to base the sort order not on a specific field, but on the results of a function. In Solr 3.4, that is now possible. For example, you can now return results based on sales figures – i.e., best-selling items – as in:

```
...&sort=product(price, numSold) desc
```

In this case, results will be returned based on the product of the `price` and `numSold` fields, so items that have brought in more revenue will appear higher in the search results. In this case, we are using the built-in `product` function, but you can use any function, whether it comes standard with Solr or you have created it yourself.

Improved analysis options

Once the user submits a query, it needs to be analyzed, and the developers of Lucene and Solr have put a lot of work into improving that process, particularly when it comes to stemming and language support.

Stemming is the process by which a term is converted to its "root" word. For example, "running" becomes "run", or "sailed" becomes "sail". Stemming is important in a search because it would be impractical for users to search for all different forms of a word; better to convert both the query and the indexed content to their root words, and compare that.

Solr has always provided the opportunity to use custom stemmers, but many people use the standard Snowball stemmers, which provide generally good results, but can be very aggressive. The problem there is that often, you wind up with the same stem for very different words, such as "organization," "organized", and "organs", all of which become "organ", leading to inaccurate search results.

To get around this problem, Solr 3.4 provides a set of less aggressive stemmers specifically for various European languages, rather than a single stemmer for multiple languages. What's more, Solr 3.4 also includes `KStemFilterFactory`, a less aggressive stemmer for English words.

Interestingly, another improvement in Solr 3.4 goes in the other direction. Since its inception, Solr has provided the opportunity to use custom tokenizers for each language you anticipate indexing. Tokenizers are important because they provide the means by which you break down a block of text into words or terms to be indexed or searched. Unfortunately, this means that there were then a huge number of tokenizers to keep up with, and you needed to make sure that the appropriate content was indexed and searched with the appropriate tokenizer.

In Solr 3.4, the standard tokenizer includes significant improvements, making it more suitable for use with virtually any language, reducing the complexity of your search application accordingly.

Complex field types

Apache Solr has always allowed you to specify the type of data to be included in a field so that you can control how it's analyzed. For example, a string field is treated differently from a text field; whereas a string is taken as a complete unit, text field data is broken up into tokens, and subjected to further manipulation and analysis. Similarly, a number field is treated differently from a date field. Both can be subjected to arithmetic, but the results are very different.

So as you can see, the field type is very important in how that data is treated, and how it behaves.

With Solr 3.4, you now have the option for defining complex field types -- that is, a type that includes subtypes. For example, another new Solr feature is Spatial Search. In spatial search, in order to determine the distance between two points, you need to know what those two points are. A point generally consists of a latitude value and a longitude value, but separately those values aren't as useful as they are together.

Enter the `LatLonType`. This type actually has two values. For example, we might add data that indicates a product is available at the Buffalo, NY store:

```
<field name="store">45.17614,-93.87341</field>
```

In this case, `store` is defined as a field of type `LatLonType`, and `45.17614,-93.87341` is a single field value – as far as the user is concerned. Complex fields allow your application to analyze the individual elements of such a field.

For example, you can now specify a complex field type such as `LatLonType` in a range query, as in:

```
...&q=*:*&fq=store:[45,-94 TO 46,-93]
```

In this case, the server knows to break down the complex values into their components and analyze them properly. (More on spatial search in the next section.)

JSON document indexing

Another area where Solr 3.4 provides new capabilities is in terms of indexing data. While it has always been easy to retrieve data from Solr in the JSON format, indexing it has been another matter altogether.

JSON is a compact format for representing data, similar to XML but much less verbose. For example:

```
"doc": {
  "id" : "THX-1138",
  "title" : "Blue Harvest"
}
```

Solr 3.4 makes it a relatively straightforward task to index JSON data, much the same way you normally index XML.

Besides the fact that JSON takes up less bandwidth, the ability to index it directly is also important because many web services and APIs use it as their delivery format. For example, Twitter returns JSON data as the result of an API call.

Indexing XML transformed with XLST

As handy as JSON is, there are still times when you want to deal with XML. For example, you might be indexing Docbook-based documents, or RSS feeds. Both of these sources are valid XML, but unfortunately, they're not in a format in which Solr can easily ingest them.

In earlier versions of Solr, you could get around this problem using the DataImportHandler, but that required creating a new request handler and a new DIH configuration – definitely not for the faint of heart. Enter the new XSLT handler, cleverly called XsltUpdateRequestHandler, which takes the pain out of this process. This new request handler enables you to specify an XSLT stylesheet Solr can use to pre-process your data and put it into the standard Solr XML vocabulary.

Consider the RSS example. Suppose you had an RSS feed that you wanted to index, such as:

```
<rss>
  <channel>
     <title>Lucid Imagination</title>
     <link>http://www.lucidimagination.com/blog</link>
     <item>
       <title>Flexible ranking in Lucene 4</title>
       <link>http://www.lucidimagination.com/blog/?p=3951</link>
       <pubDate> 2011-09-12T18:51:40Z</pubDate>
       <guid isPermaLink="false">
         http://www.lucidimagination.com/blog/?p=3951
       </guid>
       <description><![CDATA[Over the summer I served as a Google Summer of
Code mentor for David Nemeskey. David proposed to improve Lucene&#8217;s
scoring architecture[...]]]></description>
     </item>
     <item>
       <title>Learn Lucene&#8230; deeper</title>
       <link>http://www.lucidimagination.com/blog/?p=3988</link>
       <pubDate>2011-09-12T18:35:45Z</pubDate>
       <guid isPermaLink="false">
         http://www.lucidimagination.com/blog/?p=3988
       </guid>
       <description><![CDATA[You&#8217;re using Solr, or some other Lucene-
based search solutions, &#8230; or you should and will be!  You are (or will
be) building your solutions on top of a top-notch search library, Apache
Lucene. Solr makes using Lucene easier &#8211; you can[...]]]></description>
       </item>
    </channel>
</rss>
```

This is well-formed XML, but while Solr does take XML as an input format, it needs to be use a specific XML-vocabulary, in which documents and fields are clearly defined. For example, we might split this feed of two entries into two separate documents, like so:

```
<add>
  <doc>
    <field name="id">http://www.lucidimagination.com/blog/?p=3951</field>
    <field name="title">Flexible ranking in Lucene 4</field>
    <field name="subject">Over the summer I served as a Google Summer of Code
mentor for David Nemeskey. David proposed to improve Lucene&#8217;s scoring
architecture[...]</field>
  </doc>
```

```
  <doc>
    <field name="id">http://www.lucidimagination.com/blog/?p=3988</field>
    <field name="title">Learn Lucene&#8230; deeper</field>
    <field name="subject">You&#8217;re using Solr, or some other Lucene-based
search solutions, &#8230; or you should and will be!  You are (or will be)
building your solutions on top of a top-notch search library, Apache Lucene.
Solr makes using Lucene easier &#8211; you can[...]</field>
  </doc>
</add>
```

Now, if you've done any work with XML, your first thought is probably, "That's easy, just use XSLT to convert it." And of course you could do that, pre-processing each file before you feed it to the indexer. But what if you don't have that opportunity? What if the XML is generated dynamically by some process and you need it to go directly into Solr?

Even in situations in which you could conceivably pre-process the data, it's not generally an ideal solution. Fortunately, Solr 3.4 has the ability to do this processing for you. In this case, you could simply add an XSL style sheet to the `solr/conf/xslt` directory and specify it when you do the update. The stylesheet simply does the conversion:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
    <add><xsl:apply-templates select="//item" /></add>
  </xsl:template>
  <xsl:template match="//item">
    <doc>
      <field name="id"><xsl:value-of select="guid"/></field>
      <field name="title"><xsl:value-of select="title"/></field>
      <field name="subject"><xsl:value-of select="description"/></field>
    </doc>
  </xsl:template>
</xsl:stylesheet>
```

You can then use the XsltUpdateRequestHandler, and specify that stylesheet when you index the document. For example:

```
curl "http://localhost:8983/solr/update/xslt?commit=true&tr=rss2solr.xsl"
-H "Content-Type: text/xml" --data-binary @blogrss.xml
```

Between the ability to index JSON and Solr 3.4's new abilities to index arbitrary XML, the task of getting data into Solr has never been easier.

Outputting CSV content

Another useful format for transporting data is that old standby, comma separated values. Again, this is a matter of reversed roles; just as it's been possible to retrieve JSON data but not index it, it has always been possible to read comma separated values, or CSV data, but not to output easily it as the result of a query.

Solr 3.4 changes that. Consider this example:

```
...q=*:*&fl=id,name,popularity,price,score&wt=csv
```

Using the CSV response writer means that Solr will return a response such as:

```
id,name,popularity,price,score
IW-02,iPod & iPod Mini USB 2.0 Cable,1,11.5,0.98867977
F8V7067-APL-KIT,Belkin Mobile Power Cord for iPod w/ Dock,1,19.95,0.6523595
MA147LL/A,Apple 60 GB iPod with Video Playback Black,10,399.0,0.2446348
```

Performance improvements

Perhaps the first thing people think of when it comes to any new version of a software package is performance improvements. This is particularly true for an application like Apache Solr, which frequently involves long running, processor intensive operations such as indexing or searching large amounts of content. Apache Solr 3.4 has made meaningful strides in terms of performance improvements.

General performance improvements

Because the engine behind Solr is Apache Lucene, many general performance improvements seen in Solr installations are due to changes that have been implemented in the Lucene libraries. These include improvements in speed for something as crucial as primary key lookups, as well as better performance for phrase queries. Solr 3.4 can also make use of Lucene's new, faster, Snowball analyzers.

Lucene has also improved the timing of some actions that can impact performance. For example, as your application adds and removes documents, index segments need to be merged in order to improve performance by reducing the number of places to look for data. Unfortunately, the merging process itself can be a load on your system, so the priority of merge threads will now be set more carefully by `ConcurrentMergeScheduler`.

The good news is that if you use Solr, you get these improvements, along with the deployment and development conveniences delivered by Solr's packaging of the low-level Lucene libraries.

Solr itself has also added its own improvements. For example, you can now add a commitWithin parameter to your updates, specifying the time within which you'd like your data committed. So if you had a window of, say, 10 minutes during which new data can be invisible to searches, rather than committing every single update or creating a new process, you can simply state that updates need to be committed within 10 minutes, and Solr 3.4 does the rest, saving resources and improving performance by committing only when necessary.

Solr also gives you get better performance by making better use of caching, even if you have situations in which you don't want cached data. Starting in Solr 3.4, you have the ability to disable the `queryCache` and the `filterCache` on a request-by-request basis, so you don't have to be afraid to use them where they'll do you some good.

Improved segment management

The heart of a Solr instance is the search index, so any improvements made to the process of managing the segments that make up that index will be beneficial for Solr users. To that end, the

developers have implemented various improvements to the `Directory` classes, including using memory mapped IO (via the `MMapDirectory` class) by default on systems for which it makes sense.

They've also implemented other improvements. For example, if all documents in a segment have been deleted, Lucene will drop the segment on commit rather than trying to merge it. Similarly, the `IndexWriter.addIndexes(Directory[])` method will now use file copy rather than merging.

On the other hand, when Lucene does perform a merge, it will favor segments with deletions, which tends to lead to a smaller number of segments and better performance.

Perhaps the biggest change is the debut of a new default merge policy, `TieredMergePolicy`. In situations in which you have frequent document deletions, `TieredMergePolicy` provides a much more efficient merge and better performance by allowing the merge of non-contiguous index segments.

Distributed search support for spellcheck and terms

In at least one case, improvements have led not to better performance directly, but to the ability to get better performance out of your Solr installation. In previous versions of Solr, both spellcheck and terms were architected in such a way that they could not be used in a distributed system. That meant that if you needed spellcheck, you didn't have the option to use distributed Solr, and vice versa.

In Solr 3.4, the developers have implemented distributed support for both spellcheck and terms, so if you have both a large index and the need for spell checking, you can finally get the performance you need by distributing your index over several servers.

New options for building search applications

Perhaps the most exciting new features in Solr 3.4 are those that will enable you to build new applications faster, or with capabilities that you might not have even considered. Just as Solr itself has put powerful search capabilities in the hands of developers with limited resources, Solr 3.4 has added the ability to use more advanced functions such as spatial search, or even UIMA, the Unstructured Information Management Architecture, which enables you to analyze content in a way that goes well beyond what we now consider "typical" search.

Let's start by looking at Spatial Search.

Spatial Search

Spatial Search literally adds a new dimension to search applications, enabling you to find the distance between two points in a space. Perhaps the most common example of Spatial Search is the one you've already seen many times on the web. You go to a web site looking for the nearest bank, or drugstore, or hardware store to your current location. You put in your location, and you back get a list of locations closest to you.

With the addition of Spatial Search to Solr, you now have the ability not only to perform such searches, but to also incorporate distance-type searches into your overall data in other ways. For example, the sample data that comes with Solr 3.4 includes a "store" field that contains co-ordinates. You could easily perform a search that looks for all DVD players within 5 miles of your location.

What's more, you can then sort results based on distance, so you might search for all DVD players, but list those in stores closest to the user at the top. You can even boost results based on the inverse of the distance, so that closer items will tend to appear higher in the search, but distance won't be the only factor considered.

Spatial Search also enables you to search based on distance in a variety of ways. For example, you can specify a "bounding box", either by radius, as in:

```
...&q=*:*&fq={!bbox}&sfield=store&pt=45.15,-93.85&d=5
```

or by specifying a range query, such as

```
...&q=*:*&fq=store:[45,-94 TO 46,-93]
```

This kind of query can be useful for specifying all items in a pre-defined area, such as all delicatessens within a particular neighborhood.

You can even use facet queries to create search facets that group results by ranges of distance, such as “less than 10 miles”, “10 to 50 miles”, and “more than 50 miles”.

Of course, “spatial” search doesn't have to be limited to actual geographic locations; you can also use spatial search to analyze data. For example, you can find all data points that are similar to a target data point, or all items within a particular range. You can even use it for analyzing the similarity of items that may have more than one dimension.

New Velocity-driven search UI

Perhaps less ground-breaking but more useful on an everyday basis, Solr's new Velocity-driven search UI provides a way to easily create and modify Solr applications for a demonstration or a proof of concept.

The new UI is installed as part of the Solr 3.4 example, so if you have Solr installed you can find the Velocity interface at

`http://localhost:8983/solr/browse`

Even if you don't want to build applications off of it, this UI can be extremely helpful in terms of visualizing your data, including your facets, clusters, and so on, so you can be sure you're building the application you think that you are.

The Velocity-driven UI is completely customizable, so you can create a version that suits your own look-and-feel.

The Velocity–driven search UI provides an easy way to visualize your data – and a starting point for creating your own search applications.

New range facets

One of the most popular features of Solr is that of faceting, or the ability to easily narrow results based on number of criteria, such as "category" or "author". While it's always been easy to create facets based on field values, range facets have always been a little more complicated.

Developers have made do by using facet queries, such as

```
...q=video&rows=0&facet=true&facet.field=inStock&facet.query=price:[*+TO+500]
&facet.query=price:[500+TO+1000]&facet.query=price:[1000+TO+1500]&facet.query
=price:[1500+TO+*]
```

In Solr 3.4, with the advent of range queries, that's no longer necessary. Instead, you can simply specify the fields to which you want apply a range, along with the size of the range, such as:

```
...q=video&rows=0&facet=true&facet.field=inStock&facet.range=price&
facet.gap=500&facet.range.end=1500
```

In Solr 3.4, you can specify facet range values on a per-field basis, so you can completely control the facets presented by your application.

Better, faster highlighting

Another feature that is a big draw for Solr is the ease with which you can create highlighted search results, in which the search term is immediately visible right there in the result listings so that users can judge for themselves whether the returned document is relevant.

One drawback of this feature, however, has been performance, particularly for large documents. Solr 3.4 includes Lucene's `FastVectorHighlighter`, which does two things for you as a developer.

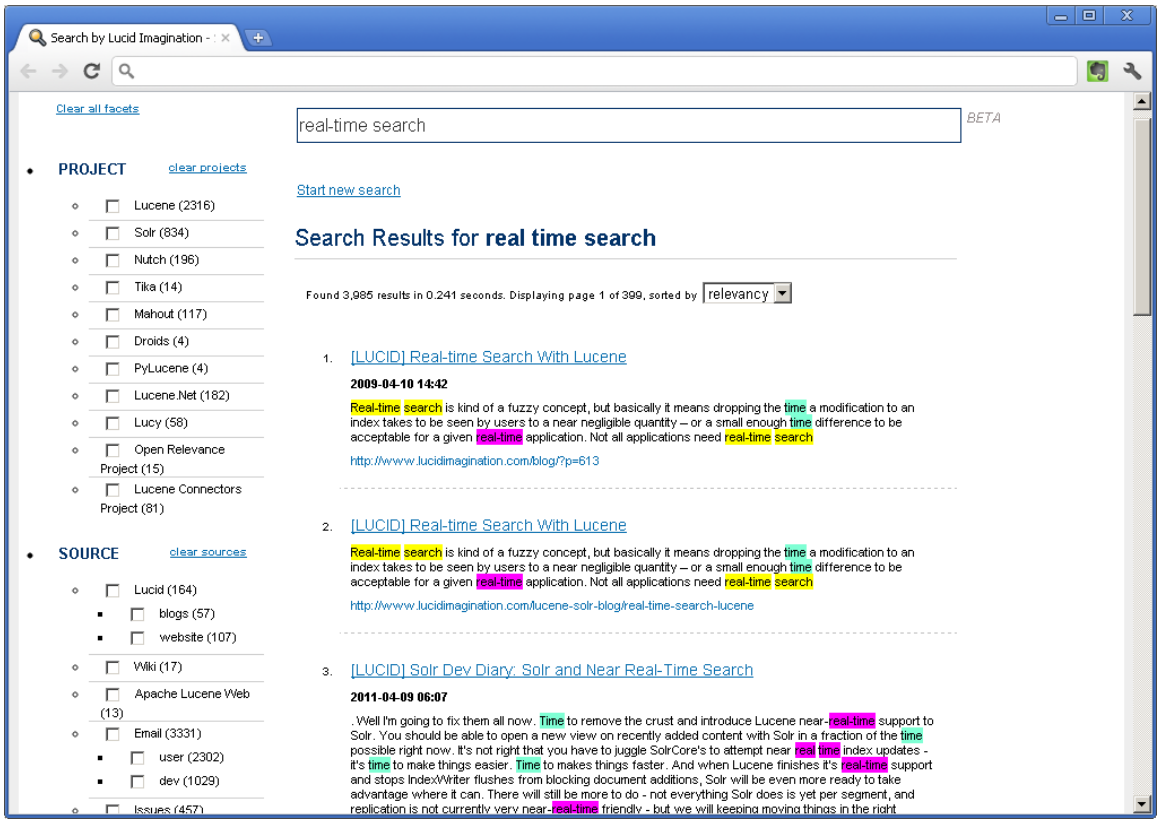
First, as the name implies, it's faster than the standard highlighter.

Second, it provides color-coded highlighting. What does this mean? It means that while the standard highlighter enables you to, say, display all found terms in bold, or in red, or in some other single presentation, the `FastVectorHighlighter` provides a series of options. For example, consider this configuration from the standard Solr example:

```
<fragmentsBuilder name="colored"
          class="solr.highlight.ScoreOrderFragmentsBuilder">
   <lst name="defaults">
```

```
      <str name="hl.tag.pre"><![CDATA[
          <b style="background:yellow">,
          <b style="background:lawgreen">,
          <b style="background:aquamarine">,
          <b style="background:magenta">,
          <b style="background:palegreen">]]></str>
      <str name="hl.tag.post"><![CDATA[</b>]]></str>
    </lst>
</fragmentsBuilder>
```

In this case, we're providing the highlighter with several levels of markup, starting with markup for the most relevant results and working our way down. So the results might look something like this:



The FastVectorHighlight component provides a visual indicator for multiple layers of relevance.

That's much more visually interesting than just bold, isn't it? In fact, because you can specify the opening tags, you can completely control the appearance of your highlighting, even providing different styles for different levels of relevance.

Faster autocomplete using Suggester

Another feature that's come to be expected from search engines is "autocomplete", in which the interface provides a drop-down menu of terms based on what the user has already typed. Prior to Solr 3.4, implementing this kind of feature has required developers to jump through a whole series of hoops in order to build the list of suggestions.

In Solr 3.4, you have the option to use the `Suggester` component, which takes the target characters and provides a list of suggestions that you can then integrate back into your application. Unlike the `TermsComponent`, which developers often used to create this feature, `Suggester` allows you to specify only the "most popular" results, or a threshhold, or percentage of documents in which a term or phrase must appear before it shows up in the list.

Searching unstructured information with Apache UIMA

Normally, when we talk about searching with Solr, We're talking about two different kinds of searches. One involves structured information, and the other involves unstructured information.

For example, you might index a document of:

```
<doc>
   <field name="vendor">Lucid Imagination</field>
   <field name="customer">AT&T</field>
   <field name="start_date">2010-12-31T23:59:59Z</field>
   …
</doc>
```

In this case, it'd be simple to do a search for customers of Lucid Imagination:

```
...&q=vendor:Lucid+Imagination
```

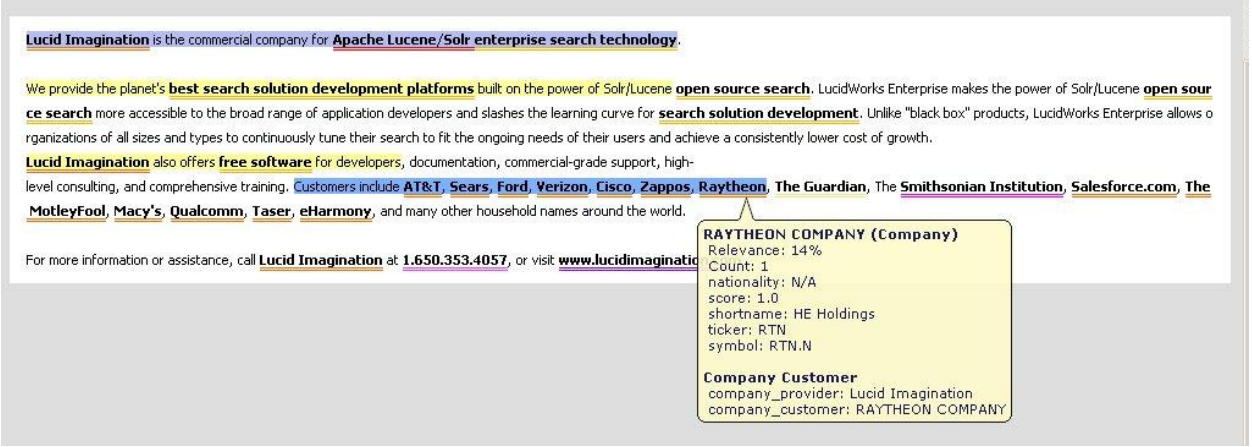
But what if all you had was a document with text such as:

```
Lucid Imagination also offers free software for developers, documentation,
commercial-grade support, high-level consulting, and comprehensive training.
Customers include AT&T, Sears, Ford, Verizon, Cisco, Zappos, Raytheon, The
Guardian, The Smithsonian Institution, Salesforce.com, The MotleyFool,
Macy's, Qualcomm, Taser, eHarmony, and many other household names around the
world.
```

How could you find that relationship? You could search for keywords and compare them to a dictionary then do analysis, but that's not very efficient.

The Unstructured Information Management Architecture (UIMA), which is now integrated with Solr 3.4, is a system that creates a "pipeline" of analyzers that annotate text with information so that you can start out with a block of text and wind up with something searchable.

More than that, the Solr 3.4 contrib/uima module includes annotators that link to services such as OpenCalais, which adds a whole variety of semantic information to content by recognizing information such as company names and relationships. For example, if we were to feed part of the summary of this paper to OpenCalais, the OpenCalais document viewer shows the type of information recognized.



Notice that this UIMA implementation not only recognizes tokens such as phone numbers, URLs, and company names, it can also add additional information about the entities it finds.

The integration of UIMA with Solr involves adding all of this information as additional fields when Solr indexes a document; that's what makes it searchable.

To configure Solr to use UIMA, you need to add information to the `solrconfig.xml` file that tells Solr what fields to analyze, and with what features. For example:

```
<uimaConfig>
  <runtimeParameters>...</runtimeParameters>
  <analysisEngine>com.example.AnalysysEngine</analysisEngine>
  <analyzeFields merge="true">body, summary, title</analyzeFields>
  <fieldMapping>
    <!-- mapping features of UIMA FeatureStructures to Solr fields  -->
```

```
    <type name="org.apache.uima.something.Annotation">
      <map feature="sampleNameFinder" field="person"/>
      <map feature="sampleCompanyFinder" field="corporation"/>
    </type>
    ...
  </fieldMapping>
</uimaConfig>
```

In this case, we're taking the content of the `body`, `summary`, and `title` fields of each document, merging them into a single block of text, and letting the `com.example.AnalysysEngine` process them. Names are added to new `people` fields, and companies are added to new `corporation` fields. Once the content's been added to the new fields, it's just as searchable as it would have been had it been part of the document all along.

Standard annotators also include those that identify sentences and parts of speech (such as nouns, verbs, and so on) and you could easily write your own. For example, you could find or create an annotator that looks for words such as "awful" and classifies text as a complaint, or looks for "great" and classifies it as a compliment. You're limited only by what you're willing to build and integrate into Solr.

The UIMA can be a powerful way to classify data; although Solr focuses on text, ultimately it can be used to search anything, if you can find or write the right analyzer. For more information on Solr's integration with Apache UIMA, see http://wiki.apache.org/solr/SolrUIMA .

Result Grouping/Field Collapsing

Another expected feature is that of result grouping, which is sort of a cross between Solr faceting and the "group by" clause in an SQL statement. This feature enables you to prevent displaying duplicate documents in returned results, or to show just a taste of what's available, enabling your users to decide just how detailed they want to get.

For example, suppose you wanted to search a set of documents representing the card catalog of your local library. You could do a complete search and facet on author names. This would give you a list of authors, and a set of documents, but the two wouldn't necessarily relate to each other. With result grouping, you could group by author and specify the number of results you wanted for each group, as follows:

```
…?q=*:*&wt=json&group=true&group.field=author_name&group.limit=2
```

In this case, you'd get back a list of results that showed each group, and the documents associated with it, such as:

```
{
  "responseHeader": {…},
  "grouped": {
    "author_name": {
      "matches": 170206,
      "groups": [
        {
          "groupValue": "Twain, Mark",
          "doclist": {
            "numFound": 77,
            "start": 0,
            "docs": [{"author_name": "Twain, Mark",
                      "title": "Adventures of Huckleberry Finn"},
                     {"author_name": "Twain, Mark",
                      "title": "The Adventures of Tom Sawyer"}]
          }
        },
        {
          "groupValue": "Dostoyevsky, Fyodor",
          "doclist": {
            "numFound": 33,
            "start": 0,
            "docs": [{"author_name": "Dostoyevsky, Fyodor",
                      "title": "The Idiot"},
                     {"author_name": "Dostoyevsky, Fyodor",
                      "title": "The Brothers Karamazov"}]
          }
        }
      ]
    }
  }
}
```


In this case, you get back a list of author names, just as you would with faceting, but you also get back a sample of documents for each author. Solr 3.4 lets you configure all of these options, from how many results you get back to how they're sorted. You can even group by query, just as you would with facets, or specify that facet counts should use only the most relevant document for each group, increasing the quality level of facet counts for large collections.

Looking Ahead: what to expect in Solr 4

As exciting as all of this is, the growing list of features that are coming down the pike in Solr 4 and beyond (beginning with 4.0 and its successive point releases) will make Solr even more exciting to work with.

Many of these improvements will make it even easier to create powerful search applications. For example, developers coming from relational databases often bemoan the lack of a "join" capability. In other words, to perform a query such as "show me all manufacturers of DVD players", you needed to either store the manufacturer's name in all product documents, or retrieve a list of manufacturer IDs and perform a second search on manufacturer documents. Solr 4 adds the ability to create a join condition, enabling you to get the results in a single step. This capability, along with others such as pseudo-fields, opens up a whole range of opportunities once thought to be rooted firmly in the RDBMS world.

Some of Solr's core features, such as faceting, will also get even more powerful. Solr 4 will see the addition of pivot faceting, in which a facet contains "sub-facets", enabling you to more easily provide your users the ability to "drill down" through search results. Power users will appreciate the ability to more precisely retrieve results by including regular expressions when performing a search, and industrial-strength users who have security or RDBMS mapping issues will be able to perform post-filtering on a set of results.

Other improvements are below the surface, such as the ability for SolrJ to process documents as they're found, rather than waiting for a single `SolrDocumentList`, or the addition of `DirectSolrSpellChecker`, which no longer requires a separate index for spell-checking. Solr 4 also includes better distributed performance through improvements to SolrCloud, as well as the ability to do date and numeric faceting across a distributed system.

But perhaps the most coveted improvement that will appear in Solr 4 is the addition of "near real time" searching, or NRT. Since the beginning, documents added to Solr have been available only after a full-index commit, leaving developers with frequently updated data facing a choice: they could deal with the fact that their data wouldn't be available immediately, or commit full index updates constantly, and deal with constant interruptions as their applications waited for the operation to complete. With the addition of NRT, developers can perform frequent "soft" commits, which make the data available, and less frequently run "hard" commits, which write the data out to persistent storage.

Can't wait for Solr 4 to be released? All of these features are available in the Solr nightly builds, or alternatively, as part of LucidWorks, the commercial grade Solr / Lucene open source platform from Lucid Imagination.

Summary

The release of Apache Solr 3.4 brings with it a host of improvements, including performance and analysis improvements and capabilities that provide new opportunities for building search applications. Some of these improvements, such as better segment management or general performance enhancements, are things that you won't need to do anything to take advantage of. Others, such as the ability to sort on functions or new range faceting, are new tools for your toolbox. Still others, such as field collapsing, spatial search or integration with Apache UIMA, provide entirely new avenues for development.

Add to that the improvements coming in Solr 4, such as Near Real-Time searching, joins, and pivot faceting, and Solr has become a force to be reckoned with.

About LucidWorks

Thousands of organizations around the world have turned to the power of Apache Solr/Lucene open source technology to drive their cutting-edge search applications. Now, Lucid Imagination, the commercial company behind Solr/Lucene, brings you LucidWorks, the search solution development platform.

LucidWorks leverages the cost-effective architecture of open source search, bundled with essential services and expert resources to help ensure you transform the constant, accelerating scope and scale of your content and data into powerful search solutions that scale economically.

You don't need to be familiar with Solr or Lucene in order to develop search applications with LucidWorks. Its power is in the convenience it delivers to let you build fast, flexible, scalable search applications.

- Speed development and deployment of your search application, slash the learning curve for your search solution development resources, and continuously deliver quality results to your end-users
- Powerful and innovative search functions built on of Solr, offer improved document acquisition, results quality, scalability, search experience that lower the cost of growth.
- Transform the disruptive power of open source search technology into a robust, stable, cost-effective search development platform to develop powerful, innovative search application for the most demanding enterprises, all backed by enterprise-grade support the from the world's leading experts in Solr/Lucene open source search technology

You can learn more about LucidWorks at www.lucidimagination.com.

Next Steps

Lucid Imagination is the commercial company for Apache Lucene/Solr enterprise search technology.

We provide the planet's best search solution development platforms built on the power of Solr/Lucene open source search. LucidWorks makes the power of Solr/Lucene open source search more accessible to the broad range of application developers and slashes the learning curve for search solution development. Unlike "black box" products, LucidWorks allows organizations of all sizes and types to continuously tune their search to fit the ongoing needs of their users and achieve a consistently lower cost of growth.

Lucid Imagination also offers free software for developers, documentation, commercial-grade support, high-level consulting, and comprehensive training. Customers include AT&T, Sears, Ford, Verizon, Cisco, Zappos, Raytheon, The Guardian, The Smithsonian Institution, Salesforce.com, The MotleyFool, Macy's, Qualcomm, Taser, eHarmony, and many other household names around the world.

For more information or assistance, call Lucid Imagination at 1.650.353.4057, or visit www.lucidimagination.com