

Open source search for the enterprise: Success factors

Open source search for the enterprise: Success factors

By Brian Pinkerton

© 2010 Lucid Imagination

BIOS

Brian Pinkerton is Vice President of Product Management and Chief Architect at Lucid Imagination. He started his career as a senior software engineer at NeXT. He then developed WebCrawler, the Web's first comprehensive search engine. He has since been the Technical Architect at AOL, VP of Engineering and Chief Scientist at Excite, Principal Architect at Ag (Amazon), Director of Search at Technorati, co-founder/President of Minimal Loop and VP Engineering at Scout Labs.

This IT Briefing is based on a Lucid Imagination/TechTarget Webcast, "Open source search for the enterprise: Success factors."

This TechTarget IT Briefing covers the following topics:

• Introduction to Solr	4
• Introduction to LucidWorks Enterprise	5
• Search within reach	6
• Lowest cost of growth	8
• Introduction to Solr Cloud	9
• LucidWorks Enterprise: Key concepts	10
• The user interface	10
• LucidWorks Enterprise ReST API	12
• Conclusion	16
• Questions and Answers	17

Copyright © 2010 Lucid Imagination. All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

About TechTarget IT Briefings

TechTarget IT Briefings provide the pertinent information that senior-level IT executives and managers need to make educated purchasing decisions. Originating from our industry-leading Vendor Connection and Expert Webcasts, TechTarget-produced IT Briefings turn Webcasts into easy-to-follow technical briefs, similar to white papers.

Design Copyright © 2004–2010 TechTarget. All Rights Reserved.

For inquiries and additional information, contact:

Mark Lewis
Director of Product Management
mlewis@techtarg.com

Open source search for the enterprise: Success factors

Today's voluminous data growth demands a scalable platform for search application developers – without it, the cost of data retrieval will increase. In this hands-on technical workshop, Lucid Imagination Chief Architect Brian Pinkerton presents a hands-on, pragmatic approach essential to unlocking the potential of open source search.

Companies of all sizes look to employ a search application. A discussion on enterprise search covers a huge spectrum in terms of the size of the company and industry, but there are some search-related issues that they all care about. The most important issue is the quality of the search results. Folks differ in what they consider high quality results, but they all want high quality results in their search application. That's why they've come to search in the first place. The second issue is performance. Everybody wants timely results. And they all want high throughput; a lot of requests per second. And finally, everybody is concerned about costs, especially in

today's economy. They're not just concerned about the cost of the software itself. They want to know: How many people do you need to develop it? How much does it cost to develop? How many developer workstations do you need? What does it cost to deploy? How many servers do you need? And then, what's the ongoing cost to maintain it? Everybody's concerned about that whole pipeline of costs.

Folks differ in their need for search in two specific areas: scale and complexity. Figure 1 illustrates the range of scale. There are two axes on this graph. One is the number of documents, which is what we think of as the unit in full text search application. The second axis is the number of queries per second or the query rate. These two axes each span about six orders of magnitude from the lower left hand corner, which might be a simple internal knowledge-based application with only a few tens of thousands of documents and maybe a query every minute or two, all

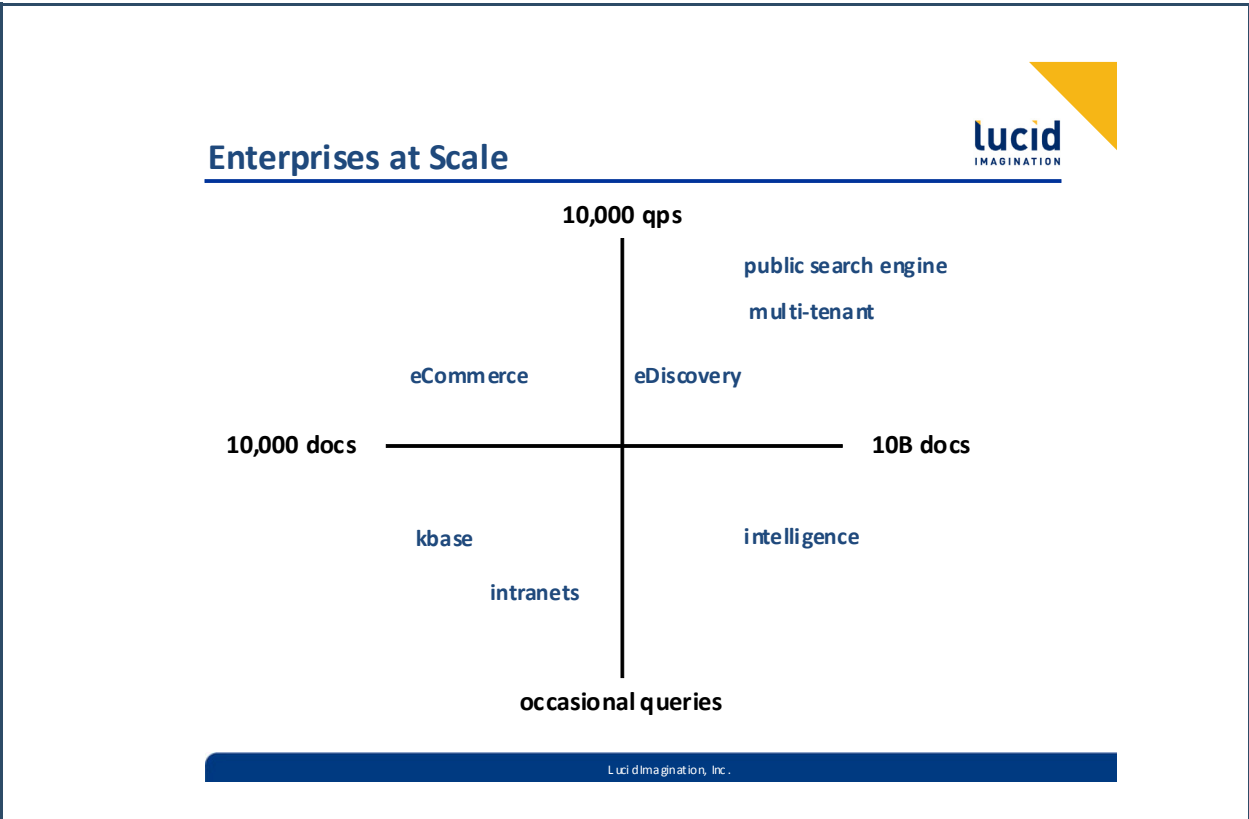


Figure 1

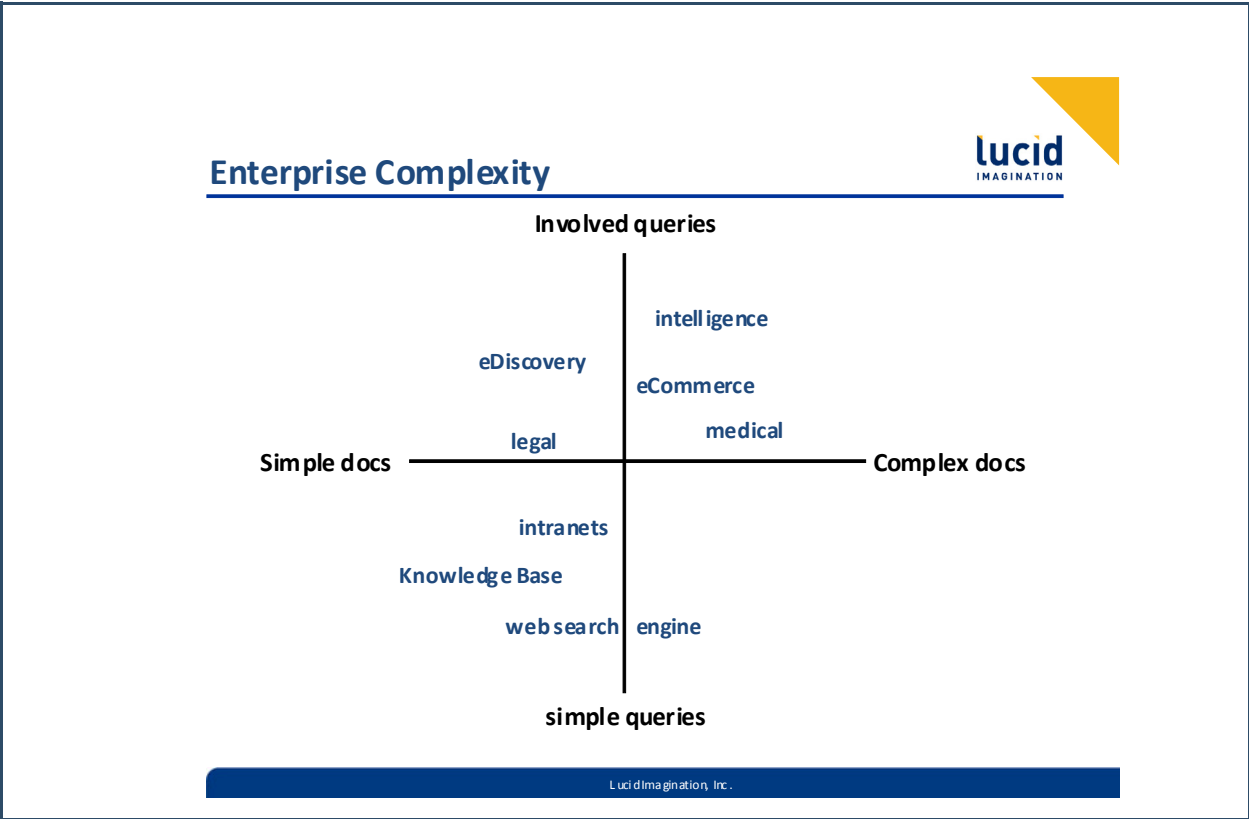


Figure 2

the way up to the right corner with one of four public search engines that have billions of documents and tens of thousands of queries per second. You can see that scale varies widely. In an e-commerce environment, for example, documents typically equal products, and there are only a finite number of products. Maybe they have 10,000-100,000 documents in their search application, but they serve a relatively high query rate because they have high volume sales. Or maybe the enterprise is in the intelligence community and they have a lot of documents but few analysts, so the query rate is low.

The documents in the full text search application and the queries that people run at them also vary tremendously. On the one hand you have a simple document as in a Web search engine where you're searching a bunch of Web pages. Web search engines actually make this quite a bit more complicated than it seems, but at first blush they're just indexing a piece of HTML. Then there are enterprises that have more complicated documents to search across like, for example, medical records or other highly structured documents that have attachments and many

different fields with rules about what goes in those fields. So documents vary from very simple to very complicated.

The queries vary as well. The simplest query is the Web search engine. Public search engines provide a search box, and if you write a complicated search string it will overflow that search box in a second. The average search string is about 2-1/2 words, all the way up to much more involved queries. Legal folks, for example, often will paste an entire paragraph into a search box. The intelligence folks build complicated queries with complicated Boolean compressions. But regardless of the complexity of the query, everybody still wants instantaneous response time.

Another interesting thing with respect to this complexity issue is the kinds of applications that people build with search. Most people are use to seeing search in its simplest form: the user types a query into the search box, and they get some results. The query is simple because the user types it. The software might exploit the document structure but users don't. If an application is built around search then the application may add data to

What's Great about Solr?



- ▶ Scales to large document collections
- ▶ High performance
- ▶ Transparent Configuration
- ▶ Flexible: can be adapted to nearly anything
- ▶ Solid out-of-the-box relevance
- ▶ Deploys on nearly any hardware
- ▶ Large developer community

Lucid Imagination, Inc.

Figure 3

the query, and the users don't know how complicated it is. Oftentimes users will use a powerful interface to generate hard queries. Folks may want more analytics out of the search results, so the software produces a histogram over time and runs simple analytics on the results. Still others might want full-blown document conversion, somewhat bare metal access to the index. Increasingly, search is a component of a larger distributed system. Programs are consuming the results instead of people. Sometimes search is used as a simple matching engine, and sometimes it's in a more complicated scenario.

Introduction to Solr

Solr is a search server that's built on top of a search library called Lucene. Solr is used widely in the enterprise for small to huge document collections. You can run it on a few documents, which we often do in testing, or you can run it on millions of documents, and it will maintain high performance.

Solr has a number of benefits. Configuration is straight forward. Solr is flexible, so you can adapt it to just about any scenario. It has solid out-of-the-box relevance, so you don't

need to do tons of tuning. Solr's greatest asset is its large developer community. A lot of people have deployed Solr and are running it currently. Chances are good that somebody's done something close to what you want to do. You can go to the community and get some very relevant help.

But Solr doesn't do everything well. It is a little hard to approach if you just want to index documents and get running. One reason for that is that Solr has no basic connectors to basic data sources. For example, you can't just crawl the Web and tell it to index that stuff or index these things out of my file system or database. Secondly, Solr, especially when it comes to the enterprise, has no security. It just indexes everything and anybody who queries them can access everything. Finally, scaling Solr to accommodate large collections is possible, but it's a manual process.

Introduction to LucidWorks Enterprise

At Lucid Imagination we help people build search applications that are built on Lucene and Solr. Our job has been to help compa-

nies get started with their applications, introduce them to Solr, help them with the pitfalls and help them once they're up and running in production mode. Having dealt with the weak areas of Solr ourselves, we decided to build on top of it to eliminate some of the problem areas and provide value to enterprise customers who need software that's adapted for their world.

LucidWorks Enterprise is a search platform that's built on top of Apache Solr and Lucene. The goal of LucidWorks Enterprise is to extend Solr and make its deployment in the enterprise more seamless. To achieve that we use all of the features inside Solr and we built some stuff around it to make it more accessible and to make it fit in better in the enterprise.

It's important to note that we built on top of Solr. We are fully committed to Solr and its future, and when we have components of LucidWorks Enterprise that belong in Solr we contribute those back to Solr and right into open source. A couple of examples of that kind of change are distributed field collapsing and the more architectural components that

relate to deploying Solr in the cloud. Those are core changes that belong in Solr, so that's where we put them. They go back to the open source.

We had several goals for LucidWorks Enterprise. Our primary goal was ease of deployment, so we made the basic configuration easy and the basic ingestion of documents straight forward. You can build a search application with just a few clicks. We're cognizant of the fact that oftentimes when you make something easy you give up power, and we didn't want to do that. We wanted to retain the power and flexibility of Solr. We took Solr-trunk, and we built around it. You can still use Solr as you've been using it. In fact, if you wanted a commercially supported version of Solr, you could get LucidWorks Enterprise and use it just like you use Solr today.

We also wanted to enable scale, because customers say that scaling search is expensive. We looked at that from two points of view: technology and pricing. First of all, what do we do to the product to make it easier to scale up? When you build and distribute the

search application how do you add nodes? How can you flexibility adapt to changing sizes in either document collection or in query rate? These are scale-related issues from a technical perspective. A lot of vendors of commercial search software charge a lot of money as you scale up. We wanted to introduce a pricing model that made scaling up affordable.

Search within reach

The first thing we wanted to do was offer a search solution that is better tuned for relevance and features out of the box. In other words, it is tuned for the typical introductory user. Secondly, we've got a simple user interface to help you get started. You don't have to edit XML configuration or anything like that. With just a few clicks of the mouse you can have a search application up and running. However, we recognized that while the simple user interface is great, it doesn't help people who want to build around the search platform. Folks typically want to integrate it with their existing applications. They'll probably ignore our user interface over time but in order to build a powerful application simply,

Search Within Reach



- ▼ Smart Defaults
 - ▼ better relevance out of the box
 - ▼ tuned for the typical intro user
- ▼ Simple UI
 - ▼ Makes getting started incredibly easy
- ▼ ReST API
 - ▼ configure it on the fly
 - ▼ easily integrate with other applications
- ▼ Integrated Doc Acquisition
 - ▼ File system, web, and database

LucidImagination, Inc.

Figure 4

Cloud



- ▼ ZooKeeper maintains config state
 - ▼ solrconfig.xml
 - ▼ schema.xml
 - ▼ stop words
- ▼ Failover uses shared ZooKeeper node status
- ▼ Load balancing decisions made locally
- ▼ Based on the SolrCloud patch
 - ▼ <http://wiki.apache.org/solr/SolrCloud>

LucidImagination, Inc.

Figure 5

these days you need some kind of ReST API. So we included with LucidWorks Enterprise a ReST API that lets people administer and configure the system from afar.

And then lastly, we simplified and integrated some document acquisition. You can acquire documents from the file system, network share, the Web and a database – again, with just a few clicks of the mouse. But if you have your own document acquisition or an export from something that you want to index, you can continue to use Solr just like you did before.

All of these functions extend Solr 4.0 trunk. We've patched it a little bit; we've fixed some of the bugs. We include the source code of the built in Solr that we're using in LucidWorks Enterprise. It's there in binary form. We are fully committed to the open course nature of the Solr core.

It's still fully accessible and configurable. We built LucidWorks Enterprise around Solr, but you can still add your own RequestHandlers. You can still insert your own data by sending it to the update handlers. You can still install

and use your own analyzers or query parcelers.

We've also added some interesting stuff around Solr to take advantage of its features; one of those is Click Scoring. When you deploy search into your enterprise, one of the things you often give up is the great link structure of the Web, things that enable page rank. What Click Scoring does instead is count how often people click on a particular result, and then boost those results most often selected, automatically incorporating feedback directly from users on which results are best.

Lowest cost of growth

LucidWorks Enterprise is focused on ease of scaling. Everybody wants to know that their search solution will scale. We introduced a new component called Solr Cloud that has been just recently committed back to Solr. It allows you to use a single shared configuration for a cluster of Solr machines. It includes built-in load balancing using that information and automatic failover. We will also include distributed support for field col-

lapsing and grouping and collection-wide IDF, which is a technical term that addresses the problem that you get into when you scale collections to big sizes. And lastly we're pricing this for growth. You pay a small amount for the upfront license to LucidWorks Enterprise and licenses for additional servers are just \$1,000 per additional server.

Introduction to Solr Cloud

Solr Cloud is most useful when you need to scale beyond a couple of machines – if you need to maintain a cluster of ten machines or more. You might need to do this for a couple of reasons. For example, you might have a very high query rate and you need ten replicas of the same thing. Or maybe you have too many documents to fit on one box and you need 10 boxes or 100 boxes or 1,000 boxes to hold all that data. To ease scaling we've implemented Solr Cloud, which uses a technology called ZooKeeper to maintain the shared configuration state. That's the configuration state that Solr usually uses to keep track of everything, such as the solrconfig, which configures the query parsers and things like that; the schema that says what

are your fields and what are their types and how are they indexed and things like stop words and synonyms.

The Solr Cloud handles failover automatically so you don't need to use a hard load balancer, for example, when you add more boxes. If one of your boxes goes down, Solr will notice that and route around it until the box comes back up. Solr also handles the load balancing so that you can partition requests across different servers.

Solr Cloud is an area of active development right now. We just committed the patch to Solr and will enhance this more and more over the coming year.

Another interesting feature of LucidWorks Enterprise is the data acquisition. Solr has no real built in acquisition for documents. Documents have to come to Solr through an update handler. So we built in acquisition for file system documents. You can just index your file from somewhere on a network share, from the Web through Web crawl, or from a database. Using this functionality makes this first step of acquiring and indexing data a little easier.

We didn't build this functionality from scratch. We integrated other proven open source packages. File system and Web connectors both use a system called Aperture, and they work with document conversion that's also based on an open source project called Tika. That handles conversion of basic file types and it also maps data in those documents to a set of built-in fields we have in LucidWorks Enterprise, such as title, author and modification date. So a lot of that built-in plumbing is just handled. You don't have to worry about doing it yourself.

LucidWorks Enterprise: Key concepts

Everything in LucidWorks Enterprise is based on a collection, which is a logical group of documents that are related either in format, schema or by use type. It could be a book and the books might come from both the Web and from the file system. It could be a set of e-commerce products that come from a database. A collection is set of stuff that you want to search. It might be a set of data coming from a couple of different data sources, both the file system and the data for example, or the file system and the Web, or a

collection of them.

Collections typically have a set of fields that you're interested in searching: title, body, document, date. And they come with a set of settings that define how you want search to work with respect to this collection. For example, do you want auto complete turned on? Do you want unsupervised feedback turned on? The further you get into search, the more you begin to appreciate these options. And then finally, the iconical representation of a collection is the index on disc. That's something that Solr uses Lucene to handle.

The user interface

The goal with the LucidWorks Enterprise user interface is to get started easily, have simple operations be really simple, and we wanted people to be able to try different options and schemas really quickly and also to manage some security.

The first thing that you see when you start up LucidWorks Enterprise is what we call Quick Start. At the heart of Quick Start is putting

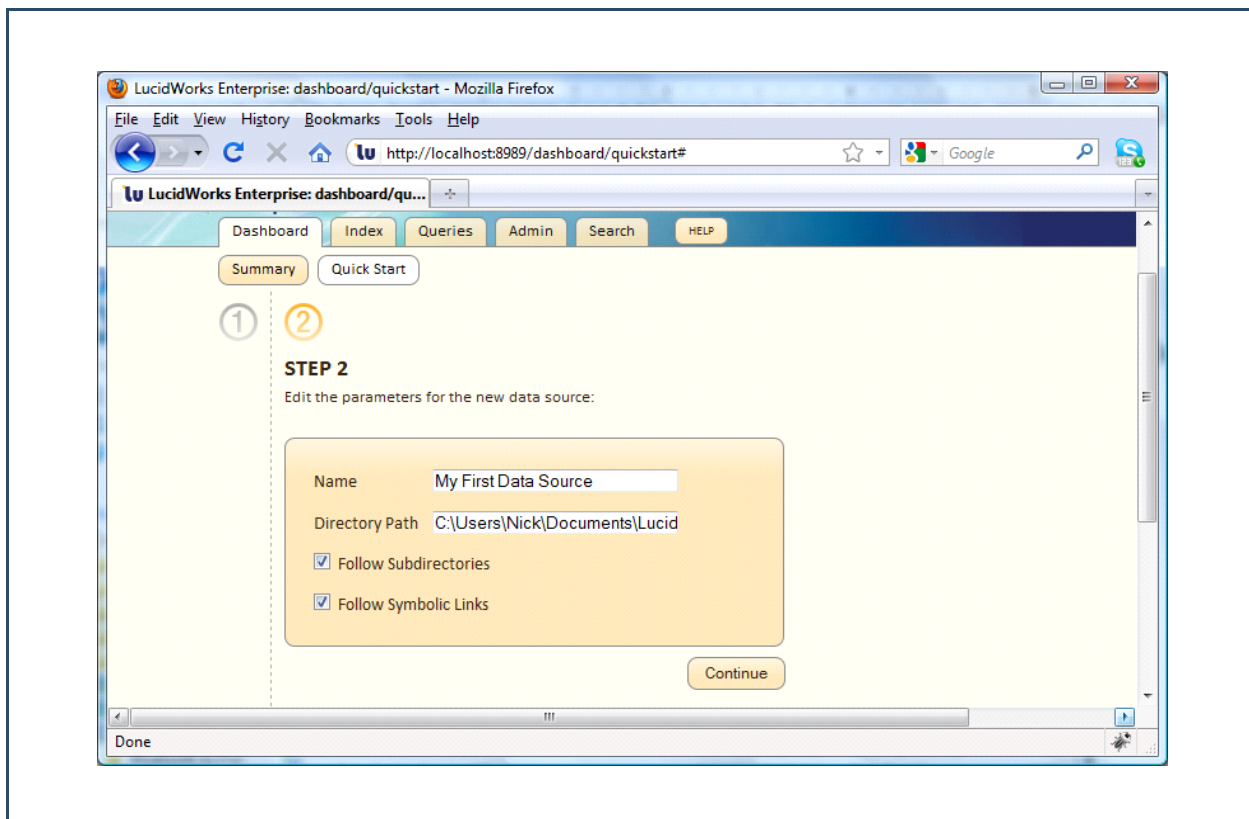


Figure 6

some data into the system. In the example in Figure 6 the user is importing data from a file system. So he says, here's my first data source, and here's the path to the data I want to import. He clicks continue and that data is imported into LucidWorks Enterprise. It's indexed, and it's available for searching.

You can also define more complicated data sources, for example, using a database connector. We made this as simple as possible, but you still have to tell LucidWorks Enterprise where the database is and how to get

the data out of the database. Once you have data in the system, it's a simple matter to search it and we have a built in search user interface that includes auto complete by default. It is incredibly easy. You can get this running in just a couple of minutes and it enables you to iterate on your search application quickly.

As you're iterating you might discover that you need to tweak things. For example, you might decide you need to edit how a field is stored or you might need to add fields. We

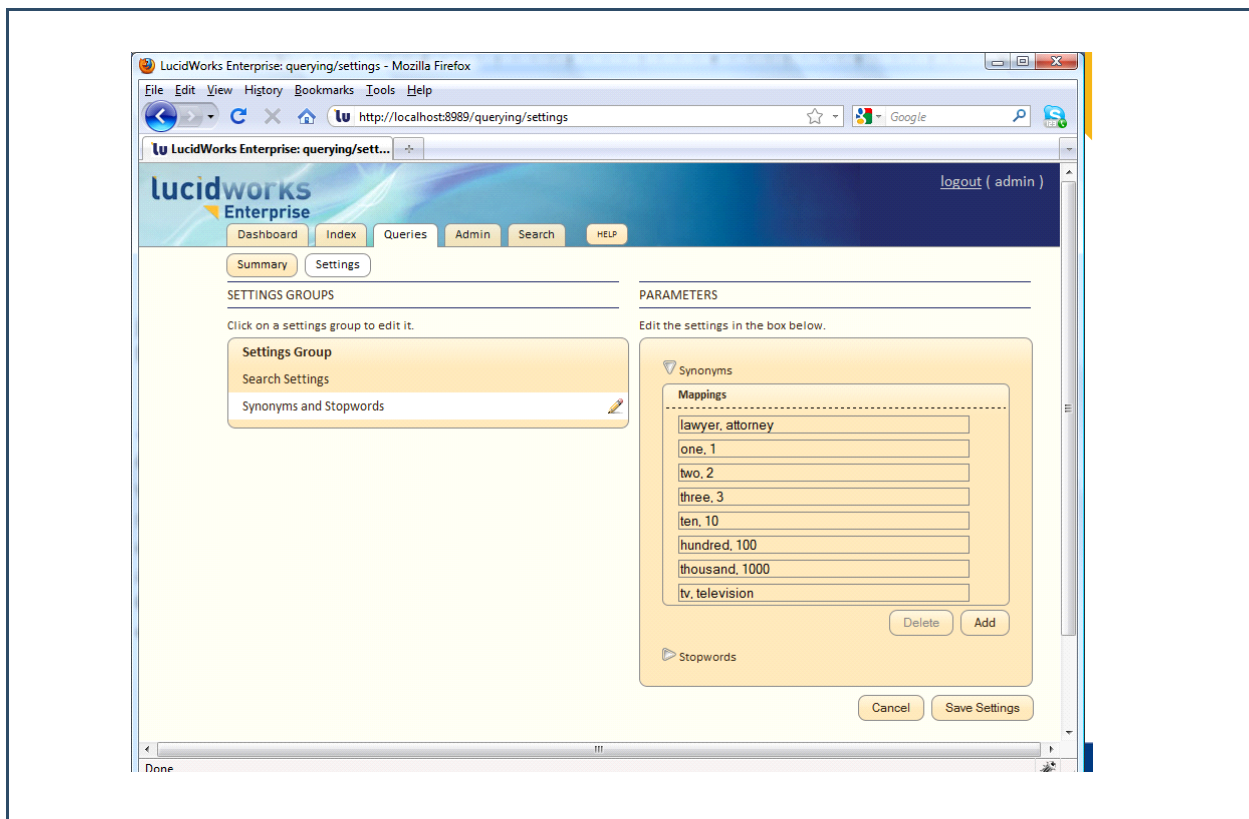


Figure 7

provide a big list of default fields and the way to change how they're indexed or change their language, things like that.

A variety of settings are also available at the search level that limits how the search user interface works. For example, you can turn on unsupervised feedback to emphasize relevance or recency. You can turn on and off stop words at query time, auto complete. All the kinds of things you would expect to find in a fully featured search application are available with the click of a button.

It's a simple matter to edit your system file.

You make these changes at this level and LucidWorks Enterprise applies them internally. In Figure 7 we're mapping 'lawyer' to 'attorney' and some common numbers to their numeric representations, and that makes search a lot fuller, and people get documents they expect in the result.

LucidWorks Enterprise ReST API

When folks use a product like this they want to know how they can easily integrate it into

their other applications. You can use our own user interface or you can build your own. If you build your own, you can use ReST API. It's how we built our user interface. Anything that you see in our user interface is built on top of the ReST API and is easily configurable at that level.

Not only can you integrate this with your own applications, and it enables other functionality as well. One of the things that we found very powerful is that it lets you set configuration parameters on the fly. Where Solr would require you to edit the Solrconfig.xml text file directly, here with LucidWorks Enterprise you can set your config parameters on the fly and some of them will take effect immediately. Some of them, because of the nature of Solr, still require the core to be reloaded and essentially restarting Solr.


ReST API is also great for easily scripting application builds. It's easy to build a shell script that will build your application, index some content and release it. Or you can do things from cron. So if you want to optimize every midnight or add new documents every midnight, you can handle that using the tools

you're used to using.

ReST is just another way of saying we use HTTP to enable the configuration of the system. It uses some HTTP that you're used to, like 'get', and some other HTTP that you might not be used to, like 'post', 'put' and 'delete' that were primitives added a little later to HTTP. ReST is very simple and understandable. It's easy to use and we use a format called JSON which is an alternative to XML that structures data for network applications, but in a more human readable format.

All of the following examples are built around using ReST with JSON to do things to LucidWorks Enterprise. The first thing you might want to ask is, "What collections are there?" You can ask for /API/collections, and it will return a little blob of JSON that says you have one collection. And then ask if there are any data sources associated with that collection, and in Figure 8 you can see there are two data sources: a file system data source and a Web data source. It shows all the parameters that we're tracking for each data source. Each parameter is settable and will cause LucidWorks Enterprise to do something.

ReST API



- ▶ **What collections are there?**
`# curl -s http://localhost:8888/api/collections ["collection1"]`
- ▶ **Any data sources for that collection?**
`# curl -s http://localhost:8888/api/collections/collection1/datasources`

```

[
  {
    "crawl_depth": 0,
    "follow_links": true,
    "name": "books",
    "include_subdirectories": true,
    "id": 5,
    "type": "FileSystemDataSource",
    "path": "/Users/bp/li/content/short_books"
  },
  {
    "crawl_depth": 2,
    "name": "thinkpink",
    "exclude_paths": null,
    "url": "http://www.thinkpink.com/bp/",
    "id": 8,
    "type": "WebDataSource",
    "collect_links": false,
    "include_paths": null
  }
]

```

LucidImagination, Inc.

Figure 8

Figure 9 shows what happens if you ask it for the configuration information for a particular field. You can get all of the data for all the fields at once, but as you can see, there's a lot of data for each particular field. Here we're asking for just information about this one field body. The first thing to notice here is that I've asked for the fields relative to our particular collections. That says to us that collections are the locus for the schema. Everything in the collection has one schema. And all of the body fields in this collection have the attributes shown. We're indexing

this field, we're doing synonym expansion on this piece, for example.

We decided not to use this field if somebody chooses to use the 'find similar feature', for example. What if we wanted to change that? What if we wanted to use this field in 'find similar'? In that case, we can just use the ReST API to change just this one field. Use a put command and change find similar to true. If you subsequently ask for the body field information again, you can see that use and find similar near the ball is in fact set to

How about some field configuration?

```
# curl -s http://localhost:8888/api/collections/collection1/fields/body
{
  "name": "body",
  "term_vectors": false,
  "default_value": null,
  "index_for_autocomplete": true,
  "use_for_deduplication": false,
  "highlight": true,
  "multi_valued": true,
  "stored": true,
  "indexed": true,
  "copy_fields": [
    "text_all"
  ],
  "search_by_default": false,
  "facet": false,
  "default_boost": 1.0,
  "user_field": false,
  "editable": true,
  "index_for_spellcheck": true,
  "synonym_expansion": true,
  "short_field_boost": "moderate",
  "include_in_results": false,
  "use_in_find_similar": false,
  "query_time_stopword_handling": true,
  "field_type": "text_en",
  "omit_tf": false
}
```

Lucid Imagination, Inc.

Figure 9

true. So that's another example of changing the schema on the fly and using it from a scriptable medium.

ReST is great, but it's an API for applications; it's not good for scripting. So we built some code that exercises the API and lets you script things. We built four sets of them out-of-the-box, and we'll be introducing more. We built ones for perl, csharp, php and python. So let's see how you might use those to build an application.

In Figure 10 we run `cd examples/perl`, and we just say `ds.pl show`. It shows a file system data source that's indexing some books, and it has included extra information about this data source because it's actually two calls to the API. It says the schedule for this data source is that it's not running. If it were running – if we were actively indexing from this collection – then it would show that it was running. That's an example of using code at the command line level to make your job of building applications easier.

Using the Example Scripts

lucid
IMAGINATION

▼ A simple example: list the data sources

```
# cd examples/perl
# ds.pl show
Data Sources: http://localhost:8888/api/collections/collection1/datasources
Data Source #5:
  Info: http://localhost:8888/api/collections/collection1/datasources/5 => {
    "follow_links" : true,
    "crawl_depth" : 0,
    "name" : "books",
    "id" : 5,
    "include_subdirectories" : true,
    "path" : "/Users/bp/li/content/short_books",
    "type" : "FileSystemDataSource"
  }
  Status:
  http://localhost:8888/api/collections/collection1/datasources/5/status => {
    "description" : "Not Running",
    "status" : {},
    "id" : 5,
    "running" : false
  }
```

LucidImagination, Inc.

Figure 10

You can also simplify long Perl commands with a straight forward implementation from the command line. You can use the Perl scripts or PHP to write a script to build your application and if you make a mistake, you can drop the index and start over again. You never have to restart LucidWorks Enterprise, you don't have to mess with anything; you just build a script that does it.

Conclusion

LucidWorks Enterprise is in limited distribution and soon will be widely available. It is available free for developers at <http://bit.ly/lucidworks-developer>. LucidWorks Enterprise is a great development platform for search. It helps people streamline their search application development and it's making search a lot more accessible.

Common Questions

Question: Is there a circumstance where I should use Solr but not LucidWorks Enterprise?

Answer: We have taken a version of Solr trunk and we've embedded that in LucidWorks Enterprise. We made a few extra bug fixes and things like that, but it has all gone back into the open source work. Our goal is to never fork Solr. We don't want to be making changes to Solr that don't go back into open source. It just becomes a maintenance headache. So we're fully committed to pushing bug fixes and new features back into Solr, especially when they belong there. Solr Cloud is a good example. That's a place where we added new functionality to Solr, a lot of new code actually, and we've pushed that right back into Solr because it belongs at that fundamental low level.

Question: The price for a one year subscription is \$68,000 for two servers running LucidWorks Enterprise. How many documents can it search with that and how fast

will that configuration index the documents?

Answer: LucidWorks Enterprise is almost indistinguishable from Solr in the size of collections that you can maintain on a server. We don't add a lot of overhead; we don't subtract a lot of overhead. On a typical server you could expect to get a collection of about 10 million typical sized documents with a decent query rate. This is not a crippled version of Solr. This is a fully functional, fully scalable version on a typical modern server. You could do a 10 million document search and get a decent query response time. With a rack of machines you're up to search engine scale.

Question: I have a Lucene application and the guy who built it left. Should I go to Solr, should I go to LucidWorks Enterprise or should I stay with Lucene and try to find someone who can help me with that?

Answer: That's a common scenario amongst our customers. Lucene is a search library and it's designed to be embedded in a software

program or application. As such there's not room for higher level management of indexes, distribution and things like that. Lucene provides a lot of great basic functionality. Most of the power of Solr derives from Lucene. Because Lucene and Solr have been recently integrated as projects at the open source level, a lot of that's getting reorganized and some of the more powerful features from Solr are actually coming back into Lucene.

There are many great reasons to keep Lucene embedded in your application, but if you are fanning it out across many nodes for distri-

bution or trying to talk to it with HTTP or something like that, then you should consider using LucidWorks Enterprise. LucidWorks Enterprise implements the server functionality around Lucene and reduces the integration burden with powerful features like security and faceting. So if you need those features, you should definitely consider Solr. A lot of people can upgrade from a Lucene application to a Solr application or a LucidWorks Enterprise application with a little bit of work. It actually simplifies their lives.



About TechTarget

We deliver the information IT pros need to be successful.

TechTarget publishes targeted media that address your need for information and resources. Our network of technology-specific Web sites gives enterprise IT professionals access to experts and peers, original content, and links to relevant information from across the Internet. Our events give you access to vendor-neutral, expert commentary and advice on the issues and challenges you face daily. Our magazines give you in-depth analysis and guidance on the critical IT decisions you face. Practical technical advice and expert insights are distributed via specialized e-Newsletters, video TechTalks, podcasts, blogs, and wikis. Our Webcasts allow IT pros to ask questions of technical experts.

What makes TechTarget unique?

TechTarget is squarely focused on the enterprise IT space. Our team of editors and network of industry experts provide the richest, most relevant content to IT professionals. We leverage the immediacy of the Web, the networking and face-to-face opportunities of events, the expert interaction of Webcasts, the laser-targeting of e-Newsletters, and the richness and depth of our print media to create compelling and actionable information for enterprise IT professionals.

Lucid Imagination_02_2010_2005