Apache Solr 4.x with RankingAlgorithm 1.5.x

By

Nagendra Nagarajayya
transaxtions llc
http://solr-ra.tgels.org

Table of Contents

Apache Solr 4.x with RankingAlgorithm 1.5.x

By

Nagendra Nagarajayya
http://solr-ra.tgels.org
updated 2013/11/16

1. Introduction

Apache Solr (a lightning fast, open source search platform) can now work with a new search library RankingAlgorithm instead of Lucene. Solr with RankingAlgorithm search seems to be comparable to Google site search (see Perl index comparison results) for certain queries and much better than Lucene. Solr with RankingAlgorithm with complex-lsa also simulates human language acquisition and recognition (see demo).

RankingAlgorithm 1.4.x enables Solr to rank product searches very accurately and also enables Near Real Time Search.

RankingAlgorithm 1.5.x adds complex-lsa algorithm with multiple similarities simulating human language acquisition and recognition.

Multiple algorithms are available SIMPLE, COMPLEX, COMPLEX1, COMPLEX-LSA. SIMPLE is a very fast algorithm and can return queries in <100ms on a 10m wikipedia index (complete index). It can also scale to 100m docs or maybe more. COMPLEX is a more complex algorithm so is a little slower compared to the SIMPLE, but can also still return queries in < 100ms on a 10m wikipedia index (complete index). COMPLEX is more accurate and should be able to give you the best rankings as compared to SIMPLE.

COMPLEX-LSA simulates human language acquisition and recognition and can retrieve semantically related terms, sentences, paragraphs, chapters, books, images, etc. COMPLEX-LSA has three similarities, TERM_SIMILARITY, DOCUMENT_SIMILARITY

TERM_DOCUMENT_SIMILARITY. With TERM_SIMILARITY similar and relevant terms can be retrieved using a term to query. With DOCUMENT_SIMILARITY, similar and relevant documents can be retrieved using a document to query. With TERM_DOCUMENT_SIMILARITY, similar and relevant documents can be retrieved using terms to query. Complex boolean expressions are supported including fuzzy/regular expression/wild card/prefix/suffix/glob queries, etc. See examples under examples/lsa for usage.

COMPLEX-LSA needs a model to be created. The model uses the existing Solr index to build the model. The model is a read-only model and expects the Solr index to be ready only also. If any documents are added or deleted from the Solr index, the model needs to be rebuilt. Once the model is created, it needs to be opened for further use. The model is associated with multiple similarities allowing a similarity to be specified for each query. Complex boolean expressions are supported including fuzzy/regular expression/wild card/prefix/suffix/glob queries, etc.

RankingAlgorithm has been integrated into Solr in such a way that either Lucene or the RankingAlgorithm can be used to do the search. RankingAlgorithm scoring does not break any of the existing functionality. So shard, faceting, highlighting, replication, etc. still work as before. Faceting with complex-lsa works with documentsimilarity and termdocumentsimilarity but not with termsimilarity.

2. Using Solr with RankingAlgorithm

There is no change in the way you access Solr. All searches work the same as before.

So a Solr search such as:

http://localhost:8983/solr/?q=california gold rush&fl=score

should still work as before. The only difference is that Solr instead of using Lucene library for search, uses the RankingAlogrithm library to search and rank the documents. The returned scores are different from Lucene and reflects the relevancy of a document.

As said above, RankingAlgorithm scores in two different modes, Document mode and Product mode. In Document mode, the scoring is for relevance and in Product mode, scoring is for occurrence. Document mode is suitable for general purpose searches such as Wikipedia docs, HTML, Word/PDF or similar docs. The Document mode is the default. Product mode is for searches found on retail stores, online store/shopping/comparison/auction websites, etc, including short text sites like tweeter.

Product mode takes a term occurrence into account and scores accordingly. For eg. a search for "wii console" will show titles starting with "wii console" are first, and the others rank lower as the occurrence of "wii console" shifts in the title or gets reversed, see below:

Wii Console and Wii Fit Plus with Balance Board Bundle (Nintendo Wii)
Wii Console System with Wii Sports Resort Game with TWO MotionPlus Attachments
Nintendo **Wii Console** w/ Bonus Wii Sports Resort Bundle, Black
Pelican Accessories **Wii Console** Stand - Nintendo Wii
Grafitti Skin for Nintendo **Wii Console**
NEW AC Adapter Cable Cord Power Supply For NINTENDO **WII** Gaming **Console**
Wii Remote Charging **Console** Stand
Nintendo **Wii** Skin - System **Console** Skin and two Wii Remote Skins - Blue Vortex
CET Domain 10301901 **Console** Stand Station for Nintendo **Wii**

There is also a scan attribute, where the scan can be a fast scan, medium scan or a full scan. Scan is the depth of the search so can be fast, slower or slow. The default is fast scan.

2.1 Enabling RankingAlgorithm

Add

```
<library>rankingalgorithm</library>
```

to solrconfig.xml

RankingAlgorithm can be enabled by adding the above line to solrconfig.xml. To use the SIMPLE algorithm, use:

```
<library>rankingalgorithm</library>
<rankingalgorithm>
```

```
        <algorithm>simple</algorithm>
    </rankingalgorithm>
```

To use the COMPLEX algorithm, use:

```
      <library>rankingalgorithm</library>
    <rankingalgorithm>
        <algorithm>COMPLEX</algorithm>
    </rankingalgorithm>
```

Default is SIMPLE algorithm.

To enable document mode use:

```
      <library>rankingalgorithm</library>
    <rankingalgorithm>
        <algorithm>COMPLEX</algorithm>
        <mode>document</document>
    </rankingalgorithm>
```

To enable product mode, enable:

```
      <library>rankingalgorithm</library>
    <rankingalgorithm>
        <algorithm>COMPLEX</algorithm>
        <mode>product</document>
    </rankingalgorithm>
```

Default is document mode.

To enable scan,

```
      <library>rankingalgorithm</library>
    <rankingalgorithm>
        <algorithm>complex</algorithm>
        <mode>product</document>
        <scan>medium</document>
```

```
</rankingalgorithm>
```

Default is fast scan.

SIMPLE algorithm functions only in document mode. COMPLEX algorithm is more accurate but a little slow compared to SIMPLE algorithm. SIMPLE is very fast and can return queries on the wikipedia index in < 100 ms and can also scale to 100 documents. SIMPLE algorithm is also very good and may be well suited than COMPLEX for some type of queries.

2.1.1 enabling complex-lsa algorithm

complex-lsa algorithm can also be enabled similar to the other algorithms in the solrconfig.xml file as below:

```
<rankingalgorithm>
     <algorithm>complex-lsa</algorithm>
     <mode>termsimilarity</document>
</rankingalgorithm>
```

The valid values for mode with algorithm=complex-lsa are:
termsimilarity, documentsimilarity, termdocumentsimilarity.

complex-lsa algorithm needs a model to be created or open. A model is associated with every query. Since a model is associated with every query, it may be easier to specify the algorithm and similarity along with the query for complex-lsa aglorithm as below:

2.1.1.1 To create a model

http://localhost:8983/solr/collection1/select/?q=abraham&fl=score,*&**model=create**&algorithm=complex-lsa&**dimension=300**&mode=termsimilarity

Note: ** very important * only text field is used to create the model now. So you will need to have a field named "text" for the model to be created.**

You need to include model=create in the query. This may take sometime to create, but once the model is created, it is also opened and the query q=abraham is executed with term similarity and algorithm=complex-lsa.

Dimension=300 (used for the bible corpus demo), but can be any value between 1 and Integer max value. This is one of the most critical values that enables the model to find hidden and semantic relations. This may need to be experimented with to get good results. The number of dimensions also affects performance.

2.1.1.2 To open a model:

http://localhost:8983/solr/collection1/select/?q=abraham&fl=score,*&**model=open**&algorithm=complex-lsa&mode=termsimilarity

You need to include mode=open in the query. This may take sometime to open the first time, but once the model is open the next query is much faster as it does not have to reopen the model. Once a model is open you don't need specify it in the query..

Note: ** very important * only text field is used in complex-lsa. So you will need to have a field named "text" for the algorithm to work.**

2.1.1.3 TermSimilarity

To use termsimilarity when the model is open:

http://localhost:8983/solr/collection1/select/?q=abraham&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=termsimilarity**

to use a boolean AND query with termsimilarity:

http://localhost:8983/solr/collection1/select/?q=holy+AND+ghost&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=termsimilarity**

2.1.1.4 DocumentSimilarity

To use documentsimilarity:

http://localhost:8983/solr/collection1/select/?q=0&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=documentsimilarity**

Note: The query value is now a document id such as 0, 1, 2 … max doc id.

2.1.1.5 TermDocumentSimilarity

To use termdocumentsimilarity:

http://localhost:8983/solr/collection1/select/?q=abraham&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=termdocumentsimilarity**

to use a boolean AND query with termdocumentsimilarity:

http://localhost:8983/solr/collection1/select/?q=holy+AND+ghost&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=termdocumentsimilarity**

to use a faceting query with termdocumentsimilarity:

http://localhost:8983/solr/collection1/select/?q=holy+AND+ghost&fl=score,*&model=open&algorithm=complex-lsa&mode=termdocumentsimilarity&facet=true&facet.field=text

2.2 Enabling mode and library dynamically

Document and Product mode can be enabled by adding “mode=product” to the search query. For eg. If the search is for “wii console”:

http://localhost:8983/solr/?q=wii console&fl=score&mode=*product*

To use document mode:

http://localhost:8983/solr/?q=wii console&fl=score&mode**=***document*

To use complex-lsa algorithm similarities:

To use termsimilarity:

http://localhost:8983/solr/collection1/select/?q=abraham&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=termsimilarity**

To use documentsimilarity:

http://localhost:8983/solr/collection1/select/?q=0&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=documentsimilarity**

To use termdocumentsimilarity:

http://localhost:8983/solr/collection1/select/?q=abraham&fl=score,*&model=open&algorithm=complex-lsa&mode=termdocumentsimilarity

To change scan:

`http://localhost:8983/solr/?q=wii console&fl=score&scan=`*`medium`*

To change library to Lucene:

`http://localhost:8983/solr/?q=wii console&fl=score&library=lucene`

2.3 realtime-search (a very fast NRT)

Solr with RankingAlgorithm includes realtime-search a very fast nrt. With realtime-search, any document added immediately becomes searchable without a commit. So documents can be added in real-time with searches in parallel. As there is no commit, the indexing is very fast. A 70,000 TPS (documents added per sec) has been seen on a quad core intel x86_64 system Linux with 48GB heap, see http://solr-ra.tgels.com/wiki/en/Near_Real_Time_Search_ver_4.x for more info.

Steps to enable RT

Add

```
<realtime visible="200" granularity="request">true</realtime>
<library>rankingalgorithm</library>
```

to solrconfig.xml

The visible attribute controls the max milli-seconds before a newly added document becomes visible in search results. So setting this to 0 means newly added documents are visible immediately but can affect update performance. Setting this to about 150-200ms seems to offer the most optimum performance. With visible set to "200", a performance as high as 70000 docs/sec has been seen with Solr-RA 4.0.

The granularity attribute controls the NRT behavior. With the default granularity="request", all search components like search, faceting, highlighting, etc. will see a consistent view of the index and will all report the same of number of documents. With granularity="intrarequest", the components may each report the most recent changes to the index.

2.4 Steps to use complex-lsa

Step1:

Index all your documents with Solr as before or if you already have an existing index that will work also.

For eg. if you want to start from scratch and index the bible corpus (download corpus here).

a. Download Solr 4.x with RankingAlgorithm 1.5.x

b. Download bible corpus

c. Install Solr with RankingAlgorithm as in section 3.

d. unzip bible corpus to the exampledocs directory under [installation]/example directory

e. edit solr/collection1/conf/schema.xml

change name=”text” field to stored=”true” from stored=”false”

Note: Important, name=”text” field is required and needs to be stored=”true” as the model uses this to access the index and creates it files.

f. Run the extract handler as below:

```
for I in `ls -1 bible_chapters/*/*`;  echo $i;  curl
“http://localhost:8983/solr/update/extract?literal.id=${i}&defaultFie
ld=text&fmap.content=text” --data-binary @${i} -H 'Content-
type:text/text'
```

Step 2:

Check that the indexing worked by counting the number of docs indexed as below:

http://localhost:8983/solr/collection1/select/?q=*:*&fl=score,*

Step 3:

Create the complex-lsa model as below:

http://localhost:8983/solr/collection1/select/?q=abraham&fl=score,*&model=**create**&dimension=300&algorithm=complex-lsa&**mode=termsimilarity**

(This step creates the model, opens it and retrieves all results relevant to abraham)

Step 4:

Execute your queries with the different similarities against the model:

http://localhost:8983/solr/collection1/select/?q=holy+AND+ghost&fl=score,*&**model=open**&algorithm=complex-lsa&**mode=termsimilarity**

Note: You don't need model=open once the model has been opened. It does not impact performance if you add it to every query

2.5 age feature

For users who want to query in realtime and just want to get changes in the index, the age feature can be made use to query a 2 billion document index in ms. The age works well documents are being add/updated consistently. It can be used as below:

http://localhost:8983/solr/?q=wii
console&fl=score,*,age=latest&docs=10

```
age=latest : enables searching only the most recently added documents
docs=n     : limit the number of results returned
```

2.6 Configuring options in solrconfig.xml

```
<!-- realtime-search tag
        Enables realtime-seach, a very fast nrt based search. Works with both
RankingAlgorithm and Lucene. Does not close the searcher object. No need for commit
(enable autocommit and set to an hr or as needed). The caches are not destroyed.
Please disable the queryResultsCache as needed. The visible attribute controls the
max milli seconds before a newly added document becomes visble in search results.
So setting this to 0 means newly added documents are visible immediatelybut can
affect update performance. Setting this to about 150-200ms seems to offer the most
optimum performance. The granularity attribute controls the NRT behavior. With the
default granularity="request", all search components like search, faceting,
highlighting, etc. will see a consistent view of the index and will all report the
same of number of documents. With granularity="intrarequest", the componets may
each report the most recent changes to the index.).
                attributes:
                        visible: numeric [ in ms. from 0 - any, default 200 ms ]
```

```
                        granularity: [request | intrarequest]    ; provides a
consistent view of the index across a request or make available changes as they
occur with intrarequest
  attributes:
                        visible: numeric [ in ms. from 0 - any, default 200 ms ]
                        granularity: [request | intrarequest]    ; provides a
consistent view of the index across a request or make available changes as they
occur with intrarequest

                values:
                        true    ; enable realtime-search
                        false   ; turns off realtime-search
    -->


    <realtime visible="200" granularity="request">true</realtime>
    <!-- library tag
        Choose the library to use with Solr.

                Values:
 rankingalgorithm        ; enables rankingalgorithm library
                        lucene                  ; enables lucene library
    -->
    <library>rankingalgorithm</library>
    <!-- RankingAlgorithm tag
        RankingAlgorithm specific tags. Choose between different algorithms like
SIMPLE, SIMPLE1, COMPLEX. Choose the mode for the search, DOCUMENT or PRODUCT.
Choose the scan fast, medium, full.
                elements:
                        algorithm
                                values:
                                        simple  [default]
                                        simple1
                                        complex
                                        complex1
                                        complex-lsa
                        mode
                                values:
                                        document [default]
                                        product
                                        if algorithm=complex-lsa
                                               termsimilarity
                                               documentsimilarity
                                               termdocumentsimilarity
                        scan    [optional]
                                values:
                                        fast [default]
                                        medium
                                        full
    -->
    <rankingalgorithm>
        <algorithm>simple</algorithm>
        <mode>document</mode>
    </rankingalgorithm>^M
```

3. Installing Solr with the RankingAlgorithm

You can install Solr with RA in two different ways. You can download Solr4.x with RA.zip a bundle of Apache Solr 4.x and Ranking Algorithm (a big download) or just download the solr-ra 4.x.war.zip which is a web archive file and copy it into an existing or new Solr 4.x installation. Below the are steps for both:

3.1 Download Solr 4.x with RA.zip (bundle)

Installation is the same as Solr. Download Solr 4.x with RA.zip (Solr version 4.x with the RankingAlgorithm) from solr-ra.tgels.com. Unzip or untar the file, change to examples directory and start Solr as before, java -jar start.jar

Step1:

Download Solr4.x with RA.zip from http://solr-ra.tgels.com

Step2:

Unzip it to a directory

Step3:

cd unzip directory/apache-solr-ra-4.x/examples

Step4:

bash -x start_solr.sh

or

java -Xmx 2gb -jar start.jar.

Step 5:

Configuring options in solrconfig.xml:

```
<realtime visible="200" granularity="request">true</realtime> <!-- true to
   enable near real time or false -->
<library>rankingalgorithm</library> <!--rankingalgorithm or lucene -->
<rankingalgorithm>
<algorithm>simple</algorithm> <!-- simple or complex -->
    <mode>document</mode> <!-- document or product mode -->
     <scan>fast</scan> <!-- fast, medium, full works in product mode -->
</rankingalgorithm>
```

3.2 Download Solr 4.x with RA.war (war file)

Instead of downloading the Solr 4.x with RA (a huge file), you can download just the solr_ra.war file. You will still need to download the Solr 4.0 from the Solr website as below. Unzip that first, and then change to the examples directory and follow the steps as below.

Step1:
Download Solr 4.x from the Apache Solr website, as in here:
http://wiki.apache.org/solr/NightlyBuilds

Step2:
Install Solr 4.x by unzip it to a directory.

Step3:
Download the solr_ra war file from solr-ra.tgels.com, as in here:
http://solr-ra.tgels.com/solr-ra.jsp
(click on download war file link at the bottom of the page)

Step4:
cp solr_ra.war.zip file to the example directory under unzip directory/apache-solr-ra-4.x/examples.

Step5:
unzip solr_ra.war.zip

Step 6:
cp solr.war webapps

Step 7:
bash -x start_solr.sh

or

java -Xmx 2gb -jar start.jar.

Step 8:

Configuring options in solrconfig.xml

```
   <realtime visible="150" granularity="request">true</realtime> <!-- true to
      enable near real time or false -->
   <library>rankingalgorithm</library> <!--rankingalgorithm or lucene -->
   <rankingalgorithm>
 <algorithm>simple</algorithm> <!-- simple or complex -->
      <mode>document</mode> <!-- document or product mode -->
       <scan>fast</scan> <!-- fast, medium, full works in product mode -->
   </rankingalgorithm>
```

3.5 Installing on Glassfish/Tomcat/JBoss/WebLogic

If you want to deploy Solr on a different container than the default Jetty container, then deploy as before (ie. Deploy examples/webapps/solr.war on Tomcat or Glassfish or Weblogic or Jboss).

3.5.1 Installing with Tomcat:

Tomcat is very simple. Download SolrRA.zip as in step 3.1, untar the file in a folder, for eg: /solr and copy the war file under /solr/apache-solr-ra-4.x/example/webapps/solr.war to tomcat/webapps directory. Add the below options to catalina.sh or set the JAVA_OPTS as below:

JAVA_OPTS="-Dsolr.solr.home=/eneeds/fs/solr/apache-solr-ra-4.x/example/solr"

Now restart tomcat as before, and you should be able to access the admin page as below:

http://localhost:port/solr

3.5.1.1 Configuring multi-cores with tomcat

Download SolrRA.zip as in step 3.1, untar the file in a folder, for eg: /solr and copy the war file under /solr/apache-solr-ra-4.x/example/webapps/solr.war to tomcat/webapps directory. Copy the default core configuration ie. /solr/apache-solr-ra-4.x/examples/solr/collection1 folder to the /solr/apache-solr-ra-4.x/example/multicore folder as below:

cp -r solr/collection1 multicore/solr

Edit multicore/solr.xml and the add a line so that solr.xml now looks like this:

```
<solr persistent="false">

  <!--
  adminPath: RequestHandler path to manage cores.
    If 'null' (or absent), cores will not be manageable via request handler
  -->
  <cores adminPath="/admin/cores" host="${host:}" hostPort="${jetty.port:}">
    <core name="core0" instanceDir="core0" />
    <core name="core1" instanceDir="core1" />

 <core name="solr" instanceDir="solr" /><-- new line added -->
  </cores>
```

Now add the below line to JAVA_OPTS in catalina.sh as below:
JAVA_OPTS="-Dsolr.solr.home=/solr/apache-solr-ra-4.x/example/multicore"

Restart tomcat and you should be able to access the core as below:

http://localhost:8983/solr/core1/select/?q=*:*

or

http://localhost:8983/solr/solr/select/?q=*:*

4. Using the RankingAlgorithm library

Download the RankingAlgorithm 1.4.x jar file from here:

http://solr-ra.tgels.com/rankingalgorithm.jsp

(Click on download link)

Add the rankingalgorithm_1.4.x.jar file to your classpath.

Using RankingAlgorithm to search is extremely simple since it make uses of the Lucene index to access the index. If you already have a Lucene Index, then you can use that as the first argument, see example code below or at http://solr-ra.tgels.com/downloads/code/Example.java:

Example 1:

```
            RankingQuery rq = new RankingQuery();
            IndexSearcher is = new IndexSearcher(index);
            StandardAnalyzer analyzer = new StandardAnalyzer();
            QueryParser parser = new QueryParser(field, analyzer);
            Query query = parser.parse(searchterms);
            RankingHits rh = rq.search(query, is); //is = Lucene IndexSearcher
object
            System.out.println("num hits=" + rh.getNumHits() + "--no docs=" +
is.maxDoc());
            for (int i=0; i<rh.getNumHits() && i<10; i++) {
                  System.out.println("i=" + i + "--" + rh.score(i) + "--docid=" +
rh.docid(i) + "--doc=" + rh.doc(i).get(title) );
            }
```

Example 2:

```
            IndexReader reader = IndexReader.open(FSDirectory.open(new
File(index)));
            RankingQuery rq = new RankingQuery();
            StandardAnalyzer analyzer = new StandardAnalyzer(Version.LUCENE_30);
            QueryParser parser = new QueryParser(Version.LUCENE_30, field,
analyzer);
            Query query = parser.parse(searchterms);
            TopScoreDocCollector tdc = TopScoreDocCollector.create(1000, true);
            rq.search(query, null, reader, tdc); //is = Lucene IndexSearcher object
            int hits = tdc.getTotalHits();
            ScoreDoc sda[] = null;
            if (hits > 0) {
                  sda = tdc.topDocs().scoreDocs;
            }
            System.out.println("num hits=" + hits + "--no docs=" +
reader.maxDoc());
            for (int i=0; i<hits && i<10; i++) {
                  ScoreDoc sd = sda[i];
                  System.out.println("i=" + i + "--" + sd.score + "--docid=" +
sd.doc + "--doc=" + reader.document(sd.doc).get(title) );
            }
            reader.close();
```

Make sure Lucene is also in the classpath since the RankingLibrary uses it to access the index. That is it. Very simple to use but gets you very accurate and relevant results. You can find more examples below:

- SimpleExample.java
- LuceneCollectorExample.java
- ComplexProductExample.java
- ComplexDocumentExample.java
- Simple1Example.java
- SimpleNRTExample.java
- AutoCompleteExample.java

- EdgeAutoCompleteExample.java
- InfixAutoCompleteExample.java
- GlobExpWithTermQueryExample.java
- RegExpWithTermQueryExample.java
- FuzzyQueryExample.java
- PrefixQueryExample.java
- WildcardExample.java

complex-lsa algoritm examples:

- Example1
- Example2
- Example3
- Example4
- Example5

5. Conclusion

Apache Solr with RankingAlgorithm makes accurate and relevant search very easy with ranking comparable to Google site search (see Perl index comparison results) and much better than Lucene.

Algorithm complex-lsa simulates human language acquisition and recognition, see demo, queries “holy AND ghost”, “japheth”. The query “holy AND ghost” returns “jesus” and “christ” at the top of the results. Nothing in the demo or Solr or index associates/links “jesus” , “christ” with “holy AND ghost”. The complex-lsa algorithm is able to discover the hidden semantic relationships modeling human knowledge to retrieve “jesus” and “christ” at the top of the results. Similarly a query for “japheth AND sons” not only returns all the sons of japheth -- ”gomer, magog, madai, java, tubal, meshech, tiras” -- but also associates such as “noah”, “shem”, etc.

realtime-search is very fast and can index 70,000 documents / sec. It also not only retrieve a document by id as in realtime-get but also seach for documents in realtime wihout a commit

or soft-commit. concurrent search, faceting, highlighting, etc. is supported as a single transaction. Realtime-search comes with multiple granularities enabling an application to get index changes intra request.

SIMPLE algorithm returns queries on a 10m wikipedia index in <50 ms. COMPLEX algorithm is more accurate but a little slower and can also return queries in <100ms. In document mode RankingAlgorithm ranks documents relevantly while ranking very accurately and precisely in the product mode. Document mode is very well suited for searching html, wikipedia, pdf/word type documents, while product works very well with short text as in retail websites, product comparison websites, short text messaging like twitter, etc. RankingAlgorithm with document and product mode is very well suited for the enterprise as well as the retail, ecommerce and websites.

6. References

1. Solr with RA, http://solr-ra.tgels.com/solr-ra.jsp
2. RankingAlgorithm, http://rankginalgorithm.tgels.com/rankingalgorithm.jsp
3. Apache Solr, http://projects.apache.org/projects/solr.html
4. Apache Lucene, http://projects.apache.org/projects/lucene_java.html