CSE535:INFORMATION RETRIEVAL

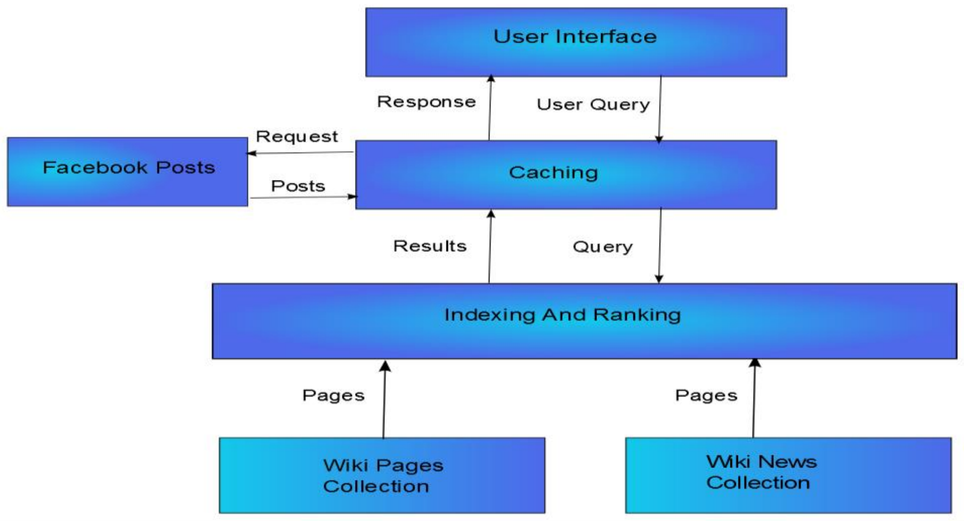
Related news articles and social media posts for Wikipedia

FINAL REPORT

Team: The Invincible

Abdul Muqtadir Mohammed 50135042
Abhineet Kumar Sinha 50130415
Imran Sayyed Adam Bijapuri 50133118
Satya Chandu Dheeraj Balakavi 50134723

System diagram and description:



Features / USPs for your implementation:

We have come up with 3 USPs for our project and they are described as follows,

1. Similarity Model:

We have defined our custom implementation for finding related Wikipedia Articles and WikiNews Articles. Solr, by default, uses tf-idf model which we would to be flawed given our dataset. It was preferring documents with shorter field length (Title field, Description field). Therefore, we found a research paper with SweetSpot Similarity Model, through which we were able to achieve greater control over varied lengthNorm values of different fields which was then found to be supported by Solr. Using this, we defined per field similarity for Title/Description in Wikipedia Articles and Title/Description in WikiNews Articles. We defined the steepness as 0.5, so that the documents are not penalized more for their field length.

Example (**Wikipedia Articles**)
Title:

- lengthNormMax =10 (Tokens) and lengthNormMin = 2
- lengthNorm steepness = 0.5

Description:

- lengthNormMax =350 and lengthNormMin = 75

Example (**WikiNews Articles**)
Title:

- lengthNormMax =10 (Tokens) and lengthNormMin = 2
- lengthNorm steepness = 0.5

Description:

- lengthNormMax =50 and lengthNormMin = 500

2. Scoring Model:

We were successful to implement the scoring logic using what we learnt in IR classes where we were taught to give the highest importance to phrase matches. Our scoring logic uses shingles and gives higher weightage to the following order,

1. Phrase Matches in Title: If all the words of the user query match the title of the document exactly in same order. [Highest Weightage]
2. Phrase Matches in Description: If the phrases appears in the articles description
3. Contains All Words in Title: If all the words of user query are present in the Title (Need not be present together)
4. Recent TimeStamp: This was done only for WikiNews articles (Recent documents were given boost)
5. Word Present in Title: Individual word matches in title
6. Word Present in Description: Individual word present in Description

We used edismax Query Parser, and created a separate RequestHandler (/better) and the mainQuery formation was done using magic fields and functions. In order to optimize the computation for timestamp, we have considered only the day/month/year as the parameter avoiding processing of Hour/Seconds as they take more time and we found them of little value.
Sample:

```
<requestHandler name="/better" class="solr.StandardRequestHandler">
  <lst name="defaults">
    <str name="indent">true</str>
    <str name="q.op">AND</str>
    <str name="wt">json</str>
    <str name="q">
      _query_:"{!edismax qf=$qfQuery mm=$mmQuery pf=$pfQuery bq=$boostQuery
v=$mainQuery}"
    </str>
    <str name="qfQuery">name^10 contents^5</str>
    <str name="mmQuery">1</str>
    <str name="pfQuery">name^1000 contents^500</str>
    <str name="ps">3</str>
    <str name="bf">recip(ms(NOW/HOUR,timestamp),3.16e-11,1,1)^100</str>
    <str name="boostQuery">
      _query_:"{!edismaxqf=$boostQueryQf mm=100% v=$mainQuery}"^9999
    </str>
    <!--The phrase is present in the title i.e NAME field-->
    <str name="boostQueryQf">name content</str>
    <str name="fl">*,score</str>
  </lst>
</requestHandler>

<str name="parsedquery_toString">
Sample Parsed Query for “India China” in WikiNews Articles
+(((name:india^10.0 | contents:india^5.0) (name:china^10.0 |
contents:china^5.0))~1) (name:"india china"~3^1000.0 | contents:"india
china"~3^500.0) ((+name:india +name:china)^9999.0) (1.0/(3.16E-
11*float(ms(const(1417906800000),date(timestamp)))+1.0))^100.0
</str>
```

3. Relatedness of News Articles

Our implementation of retrieving the related news stories is inspired from an idea that was mentioned in the thesis – "Extracting Named Entities and Synonyms from Wikipedia for use in News Search." by "Christian Bøhn".

Summary of thesis: Each wikipedia page has internal links, redirects and disambiguation which make it very easy for the users to find more information about a specific keyword mentioned in the article text.
This research work intend to collect all links pointing to the same article and then aggregate them based on the label to find synonyms as well as the popularity of that synonym.
Then a simple prototype is made up to two components:

- Named entity mining component that is used to extract entities from Wikipedia in order to build a dictionary of named entities,
- List of common synonyms for each entity.

Adaptation of this concept in project:
1. Terms of query could be used as entities.
2. Links in user selected wikipedia article are extracted. These links can be used as list of terms related to entities defined in step 1 .
4. These terms are ranked on basis of term frequency in selected wikipedia article, to find top three relevant terms.
5. Query formulation for news article = terms from user query and top three terms found in previous step.

A high-level overview of our implementation is given below,

1. Extract all the internal Wikipedia links that are present in the user-selected Wikipedia article. These will be the links that have to be scored on the basis of the term frequency in the article.
2. Create a local in-memory index of the Wikipedia article content, so we can make use of Lucene core APIs to compute scoring of the links against the wiki article content.
3. Save the link and it's in a map, and reverse sort the same to obtain the top 3 results.
4. Perform string comparisons and concatenations, where necessary, to generate a single search query string.
5. Using SolrJ API trigger a request to Solr to retrieve the news stories in a ranked order for the search query we have formulated in the above steps.
6. Format the Solr response from JSON string to Java object, which can be used to render the related news stories, title and link, onto the UI.

Social Media Posts

The project required fetching of Social Media Posts (Facebook) for the user query term. The fetched posts are ranked according to the tf-idf and a small boost is also given to the date parameter to fetch the latest articles. Used Graph API v 2.2 for the purpose.
It also lets you define new objects and actions in the social graph from your website, and the way that you create new instances of those actions and objects is by using the Graph API.
In order to make calls to the Graph API on behalf of a user, either to retrieve some user information or to post on the user's behalf, you will need the user to grant the necessary permissions to your website.
The query fed to the API after a click on a specific Wikipedia article was generated in a similar fashion as it was generated for fetching news articles.
The format for search Facebook API GET request is
graph.facebook.com/search?q={your-query}&[type={object-type}]&access_token=<AccessToken>

Configuration details and tweaks:

We have indexed following fields from the enWikipediaArticles and WikiNews dataset using DataImport Handler,

```
<!--DataImport Handler-->
<dataConfig>
<dataSource type="FileDataSource" encoding="UTF-8" />
<document>
<entity name="page" processor="XPathEntityProcessor"
stream="true" forEach="/mediawiki/page/" url="C:\Users\Amair\Desktop\solr-
4.10.1\example\solr\wiki\data\wikiCopy.xml"
transformer="DateFormatTransformer,TemplateTransformer,RegexTransformer">
<field column="id" xpath="/mediawiki/page/id" />
<field column="name" xpath="/mediawiki/page/title" />
<field column="contents" xpath="/mediawiki/page/revision/text" />
<field column="timestamp" xpath="/mediawiki/page/revision/timestamp"
dateTimeFormat="yyyy-MM-dd" />
<field column="$skipDoc" regex="#REDIRECT .*" replaceWith="true"
sourceColName="contents"/>
</entity>
</document>
</dataConfig>
```

Fields:

- DocId
- Title
- Description
- Timestamp
- Copy field: phrase_suggest for optimization of Wildcard queries

```
<!-- Schema field -->
<field name="id" type="text_general" indexed="true" stored="true" required="true"/>
<field name="name" type="text_title" indexed="true" stored="true"/>
<field name="contents" type="text_general" indexed="true" stored="true"/>
<field name="timestamp" type="tdate" indexed="true" stored="true"/>
<field name="text" type="text_general" indexed="true" stored="true"/>
<copyField source="name" dest="phrase_suggest"/>
<field name="phrase_suggest" type="text_suggest" indexed="true" stored="false"/>
```

Title/Description Field:

```
<!-- TITLE FIELD Analyzer and Similarity for WikiNews -->
<fieldType name="text_title" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <charFilter class="solr.HTMLStripCharFilterFactory"/>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.PorterStemFilterFactory" />
    <!-- We will only use synonyms at query time -->
    <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true"
expand="false"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.TrimFilterFactory" />
  </analyzer>

  <similarity class="solr.SweetSpotSimilarityFactory">
    <!-- TF -->
    <float name="baselineTfMin">1.0</float>
    <float name="baselineTfBase">1.5</float>
    <!-- plateau norm differ for Descritption-->
    <int name="lengthNormMin">2</int>
    <int name="lengthNormMax">10</int>
    <float name="lengthNormSteepness">0.5</float>
  </similarity>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.EnglishMinimalStemFilterFactory" />
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

TimeStamp:

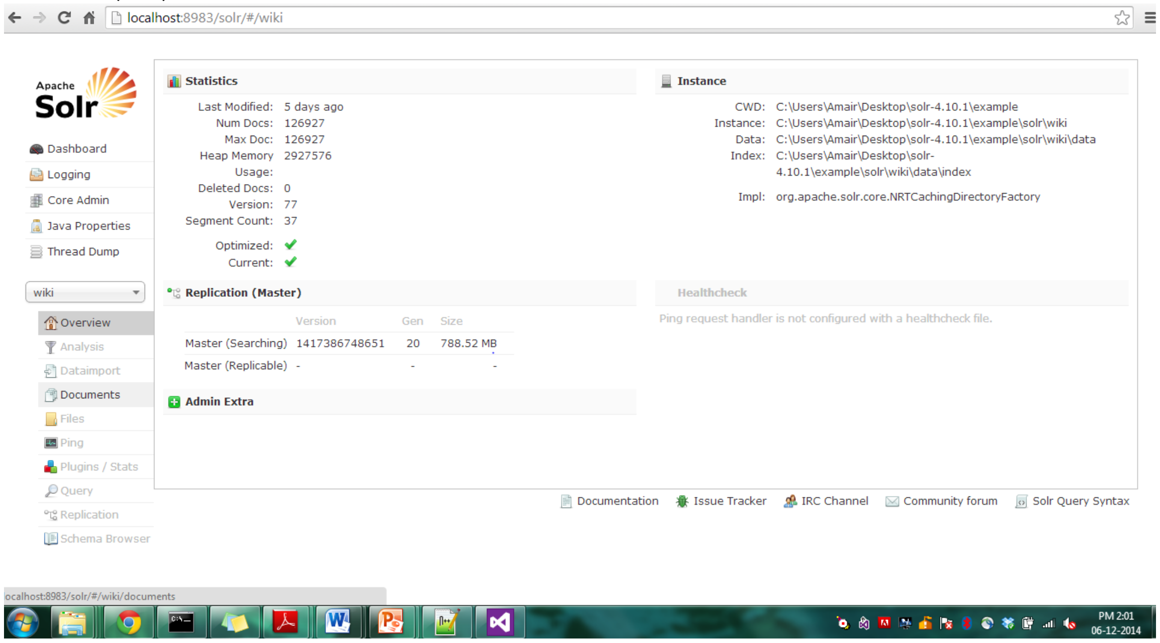
```
<fieldType name="tdate" class="solr.TrieDateField" precisionStep="6" positionIncrementGap="0"/>
```

Phrase_suggest:

```
<fieldType name="text_suggest" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <charFilter class="solr.HTMLStripCharFilterFactory"/>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.EnglishMinimalStemFilterFactory" />
    <filter class="solr.EdgeNGramFilterFactory" minGramSize="1" maxGramSize="25" side="front"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.TrimFilterFactory" />
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.EnglishMinimalStemFilterFactory" />
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Solr Stats:

Index Size (Wikipedia Articles):
Total: 1,26,927


localhost:8983/solr/#/wiki

Statistics

Last Modified: 5 days ago
Num Docs: 126927
Max Doc: 126927
Heap Memory Usage: 2927576
Deleted Docs: 0
Version: 77
Segment Count: 37
Optimized: ✓
Current: ✓

Instance

CWD: C:\Users\Amair\Desktop\solr-4.10.1\example
Instance: C:\Users\Amair\Desktop\solr-4.10.1\example\solr\wiki
Data: C:\Users\Amair\Desktop\solr-4.10.1\example\solr\wiki\data
Index: C:\Users\Amair\Desktop\solr-4.10.1\example\solr\wiki\data\index
Impl: org.apache.solr.core.NRTCachingDirectoryFactory

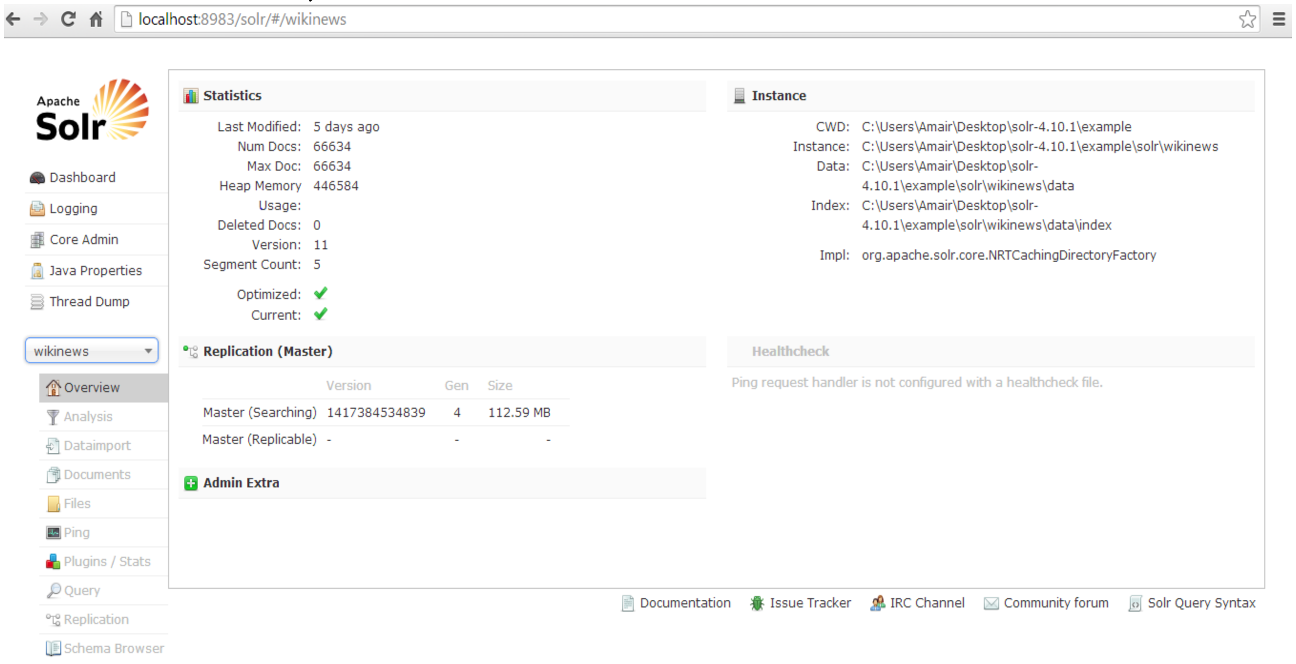
Replication (Master)

	Version	Gen	Size
Master (Searching)	1417386748651	20	788.52 MB
Master (Replicable)	-	-	-

Healthcheck

Ping request handler is not configured with a healthcheck file.

Index Size (Wikipedia Articles):
Total Documents: 66,634


localhost:8983/solr/#/wikinews

Statistics

Last Modified: 5 days ago
Num Docs: 66634
Max Doc: 66634
Heap Memory Usage: 446584
Deleted Docs: 0
Version: 11
Segment Count: 5
Optimized: ✓
Current: ✓

Instance

CWD: C:\Users\Amair\Desktop\solr-4.10.1\example
Instance: C:\Users\Amair\Desktop\solr-4.10.1\example\solr\wikinews
Data: C:\Users\Amair\Desktop\solr-4.10.1\example\solr\wikinews\data
Index: C:\Users\Amair\Desktop\solr-4.10.1\example\solr\wikinews\data\index
Impl: org.apache.solr.core.NRTCachingDirectoryFactory

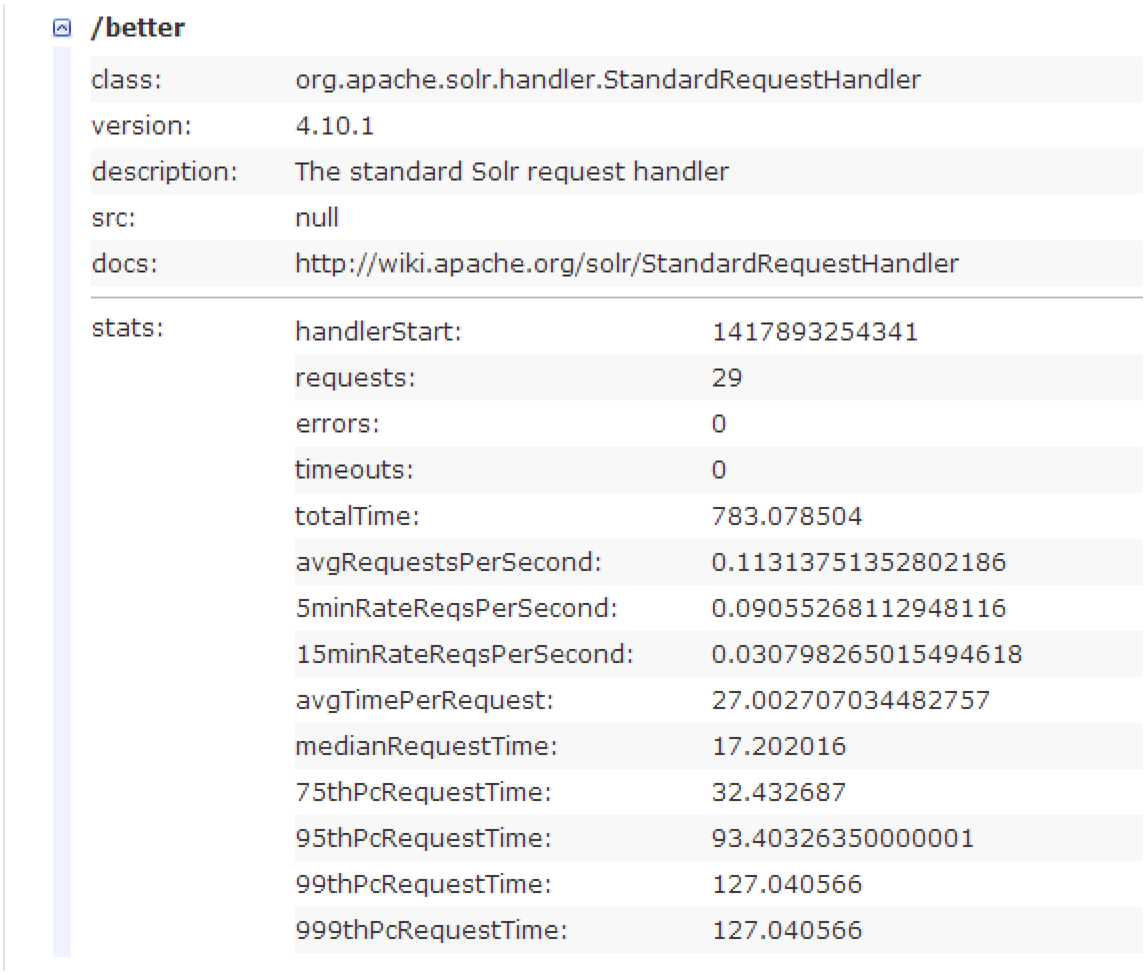
Replication (Master)

	Version	Gen	Size
Master (Searching)	1417384534839	4	112.59 MB
Master (Replicable)	-	-	-

Healthcheck

Ping request handler is not configured with a healthcheck file.

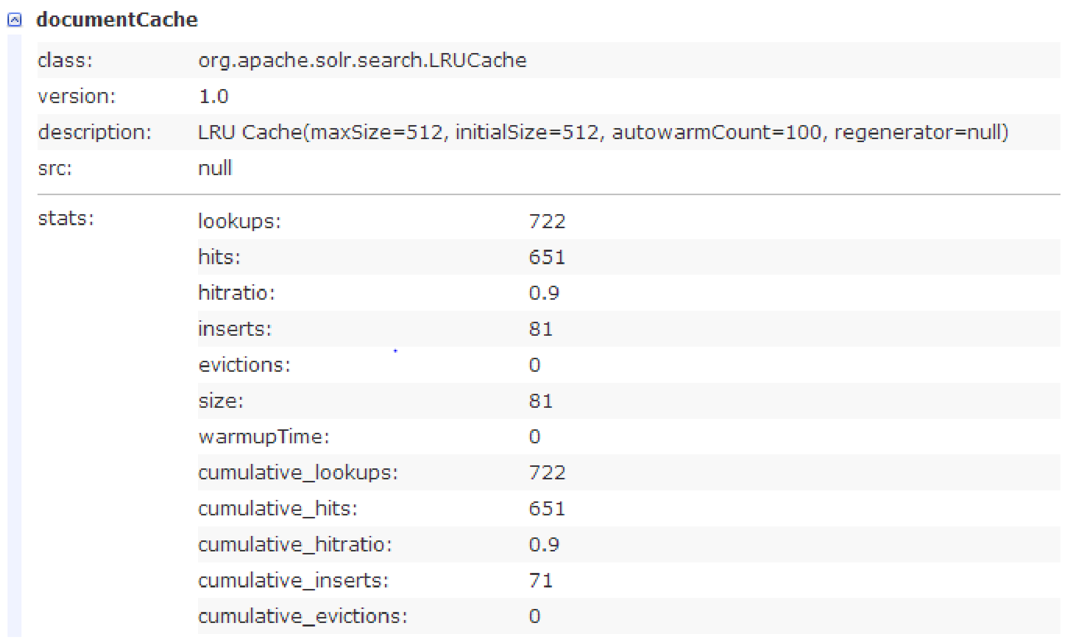
Performance Stats:
We have redefined our RequestHandler from /select to /better



/better

class:	org.apache.solr.handler.StandardRequestHandler	
version:	4.10.1	
description:	The standard Solr request handler	
src:	null	
docs:	http://wiki.apache.org/solr/StandardRequestHandler	
stats:	handlerStart:	1417893254341
	requests:	29
	errors:	0
	timeouts:	0
	totalTime:	783.078504
	avgRequestsPerSecond:	0.11313751352802186
	5minRateReqsPerSecond:	0.09055268112948116
	15minRateReqsPerSecond:	0.030798265015494618
	avgTimePerRequest:	27.002707034482757
	medianRequestTime:	17.202016
	75thPcRequestTime:	32.432687
	95thPcRequestTime:	93.40326350000001
	99thPcRequestTime:	127.040566
	999thPcRequestTime:	127.040566

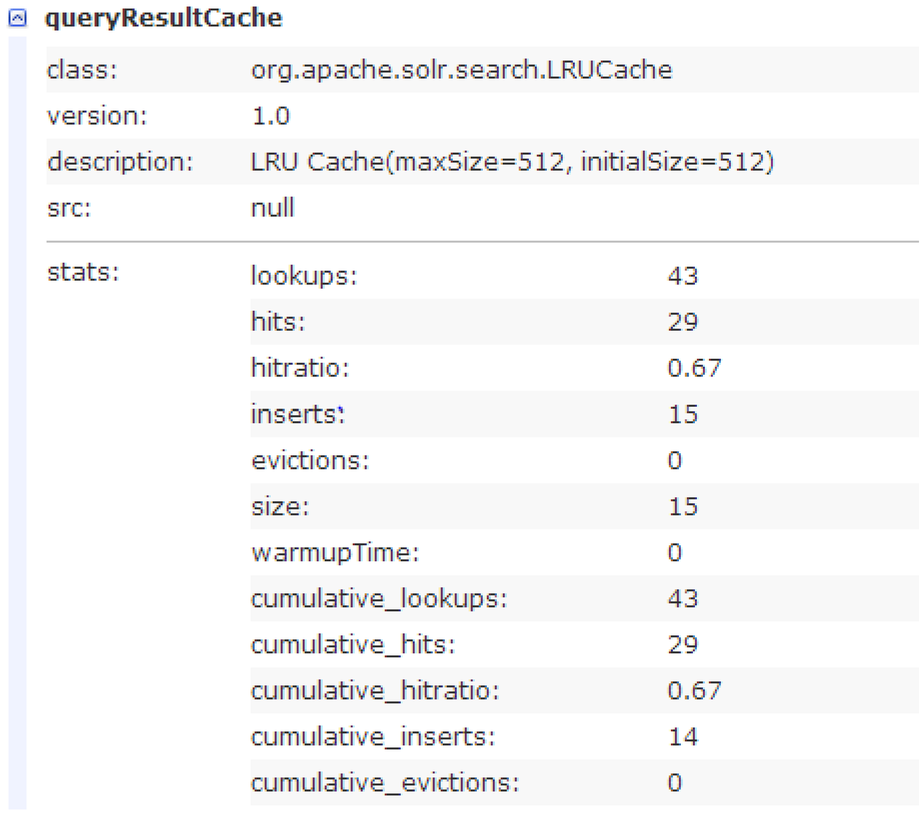
Document Cache Stats:



documentCache

class:	org.apache.solr.search.LRUCache	
version:	1.0	
description:	LRU Cache(maxSize=512, initialSize=512, autowarmCount=100, regenerator=null)	
src:	null	
stats:	lookups:	722
	hits:	651
	hitratio:	0.9
	inserts:	81
	evictions:	0
	size:	81
	warmupTime:	0
	cumulative_lookups:	722
	cumulative_hits:	651
	cumulative_hitratio:	0.9
	cumulative_inserts:	71
	cumulative_evictions:	0

Query Cache Stats:



queryResultCache

class:	org.apache.solr.search.LRUCache	
version:	1.0	
description:	LRU Cache(maxSize=512, initialSize=512)	
src:	null	
stats:	lookups:	43
	hits:	29
	hitratio:	0.67
	inserts:	15
	evictions:	0
	size:	15
	warmupTime:	0
	cumulative_lookups:	43
	cumulative_hits:	29
	cumulative_hitratio:	0.67
	cumulative_inserts:	14
	cumulative_evictions:	0

Future work:

- We wanted to incorporate a Click-Through Framework, where each query that the user has fired gets logged and then store the results on which the user has clicked. Once we have this score, we could incorporate this into our scoring mechanism, so that we have better ranking of documents. Normalizing of the score generated by Solr and adding Click-thru-Score (relative to the already-normalized-score) is the only challenge here.
- We have also tried to implement category based related news articles, but due to improper documents being fetched, we reverted back. In this case, there were a lot of irrelevant documents being fetched.
- We also wanted to implement autocomplete feature using the similar framework for spell-checker. We have also defined a handler for this purpose, but could not complete it due to shortage of time.

Member contributions:

Abdul Muqtadir Mohammed (5013-5042):

- Defined everything in schema.xml and solrconfig.xml.
- Defined a Data-Import Handler and data-config.xml, which also removed all the documents containing #REDIRECT as content along with defining of XPath.
- Created two separate cores for Wikipedia Articles and WikiNews Articles and indexed around 126927 + 66634 documents removing HTML/XML strips.
- Defined appropriate Analyzers for all the fields.
- Defined a new RequestHandler (/better) which was used by the entire application for querying. It uses shingles and assigns appropriate weightage for each match (such as Phrase Match etc.). This was the USP of our project and has been described in detail above under Scoring. For WikiNews Articles, I have also defined a separate weightage to TIMESTAMP.
- Improved relevance of Wikipedia articles and Wikinews articles for a given query.
- Query Formulation for fetching Wikipedia Articles and WikiNews Articles
- Implemented Per-Field Similarity and defined SweetSpot Similarity with appropriate range values after comparing the rankings with different similarity models such as DFRS, Tf-idf and different lengthNorm values, which is USP of our project.
- Implemented /spellcheck Handler which uses a separate index for suggesting words that we misspelled based on the values present in index.
- Implemented /wildcard Handler which was optimized for better performance (explained above)
- Implemented synonyms to be used only at query time, avoiding expanding.
- Implemented optimal DocumentCaching and QueryCaching.
- Tried implementing click-through scoring but failed to properly normalize the values (this could have been done if time permitted)
- Everything related to Solr and it's features has been implemented by me.

Abhineet Kumar Sinha (5013-0415):

1. This class overrides default similarity class. We developed different custom similarity class for each field.

Model for name field: Below parameters were overridden

- Coord^(15)
- termfreq^3
- Lengthnorm^0.5

Model for name field: Below parameters were overridden

- Coord^4
- termfreq^1.2
- Lengthnorm^0.8

Disadvantage:

- No factor to consider chronological order of articles.

2. Understanding research paper and creation of algorithm for relatedness of news article.
3. Research papers read:

"Extracting Named Entities and Synonyms from Wikipedia for use in News Search." by Christian Bøhn
"Effective Query Formulation with Multiple Information Sources" Michael Bendersky, Donald Metzler, W. Bruce Croft

Imran Sayyed Adam Bijapuri (5013-3118):

- **<u>Social Media Posts</u>**
  Implemented social media posts fetching (Facebook) by making use of Graph API v2.2. It's a low-level HTTP-based API that one can use to query data, post new stories and a variety of other tasks.

Implemented a ranking algorithm for sorting the social media posts based on their relevancy to user query.

- **<u>UI</u>**
  Worked with teammates for front end UI development. Used HTML, JavaScript, JQuery (for Ajax implementation), CSS, JSP, Scriptlets.Backend development for parsing the response returned from Solr using GSON and casting them into custom JAVA objects.
  Technologies used: Solrj, GSON.
- **<u>Solr Spell Checker</u>**
  Made use of Solr's SpellCheckComponent handler to implement Solr spell checking feature. The Spellcheck feature was called upon in two scenarios.

  1) When the number of returned results were less than a predetermined threshold (in our case).
  In this scenario the returned results were displayed on the screen along with a prompt to the user "**Did you mean <nearest word returned by Spellchecker?>**"
  2) When the number of results were zero
  In this scenario query fed is the top word returned by the Spellchecker component along with a prompt to the user "**Showing results for < nearest word returned by Spellchecker >**".

Satya Chandu Dheeraj Balakavi (5013-4723):

1. Designed the main user interface that is currently being used in the project. The UI screens I developed have been used as the baseline for building the front end application.

   <u>Technologies used</u>: HTML, Javascript, JQuery (for Ajax implementation), CSS, JSP, Scriptlets, Servlet
2. Implemented the query formation technique to retrieve the Related News stories after the click of a wiki article link in the search results page. The implementation includes,
   a. Understanding of the research paper "Extracting Named Entities and Synonyms from Wikipedia for use in News Search." and coming out with core implementation logic so it can be utilized by us during our implementation.
   b. Use of SolrJ API to establish a communication with Solr.
   c. Use of Google's GSON API for conversion from JSON string to Java object and vice versa.
   d. Use of a few Lucene core APIs, most notably MemoryIndex, on the process of implementing our core logic.
   e. Display of the most relevant 5 news stories on to the user interface.

   <u>APIs used</u>: SolrJ, Lucene core, Google's GSON
3. Implemented the logic to convert the Wikipedia content, with all the Wikipedia mark up, into normal (mark-up free) text so we can populate the wiki content abstract in the search results page. Used the two APIs – Java Wikipedia API (Bliki engine) and JSoup Java HTML Parser - first for conversion of Wikipedia marked-up content to HTML and later the HTML content to normal text. Reason for 2 step implementation was because the Wikipedia API's conversion to direct text seemed a bit more complex with the API we used, and other APIs were not providing desired results.

   <u>APIs used</u>: bliki-core and JSoup
4. Multiple UI enhancements and quick fixes, which includes a failed attempt in using JQuery and Ajax on one module wherein we had to incorporate a functionality to dynamically update the news stories on the same search results page rather than having the user to navigate to a new page.

Reference:

“Extracting Named Entities and Synonyms from Wikipedia for use in News Search.” by “Christian Bøhn”.
http://trec.nist.gov/pubs/trec16/papers/ibm-haifa.mq.final.pdf