IR- Project 3

Question Answering System

Team Name: Pasanga

Agrawal, Saurabh- 50096545

Manivasagam, Niranjan- 50097714

Soundara Rajan, ShiyamSundar-50097590

Thruvas Jeyakumar, Sabharinath Babu- 50098129

Table of Contents

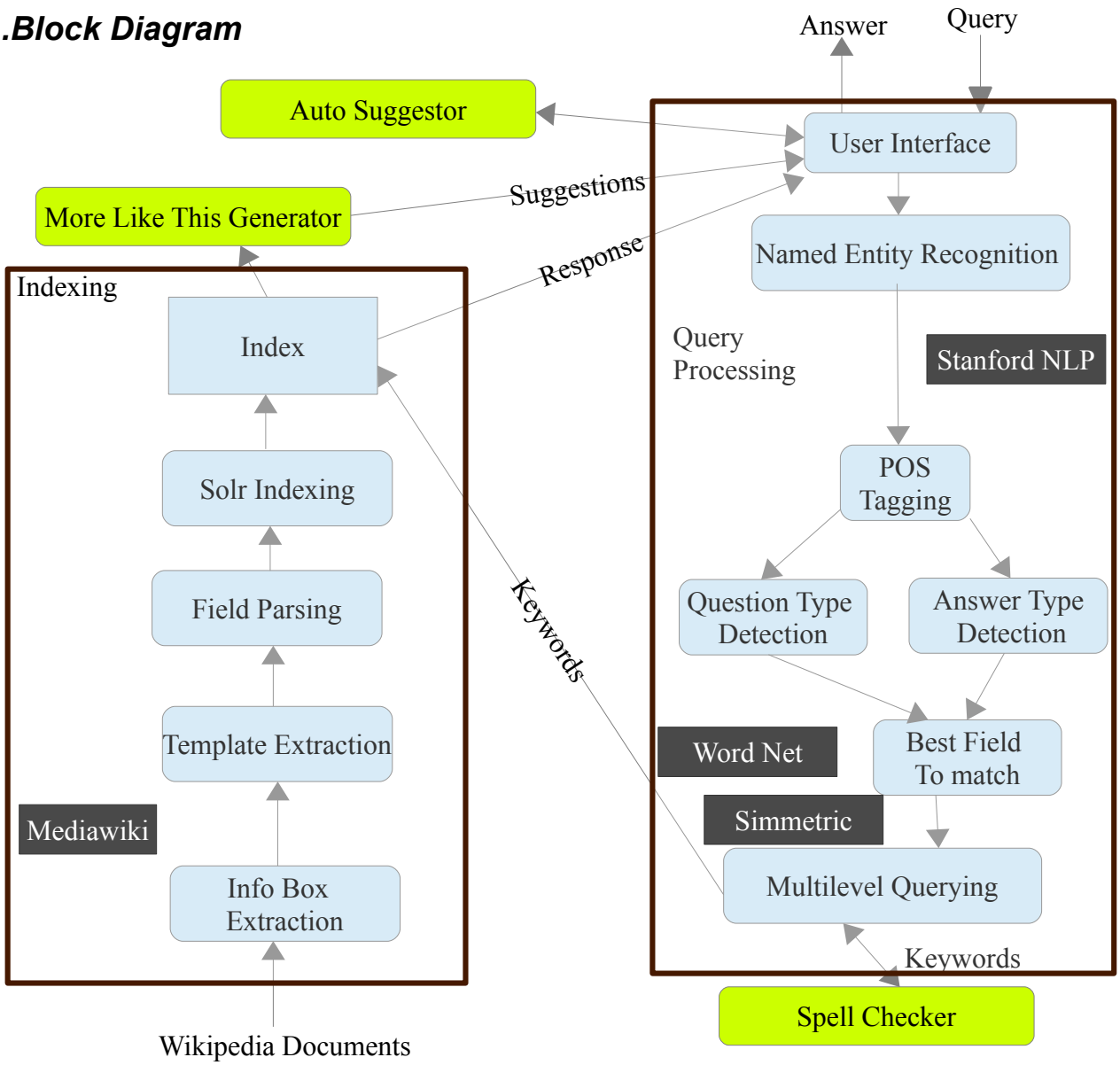
1.Introduction

The QA system developed aims to robustly answer the user's queries. Wikipedia documents are indexed using SolrJ. The query that the user enters into the system is then processed using NLP and key words are extracted. These key words are then sent to the index and queried. The possible answers are then retrieved and ranked using Solr's inbuilt ranking features and the top results are displayed.

Section 2 describes the overall flow of the system.Section 3 elaborates the unique features of the system. Section 4 explains the details about the schema. Section 5 explains the performance details. Section 6 has the screen shots of the User Interface showing the system at work.

2.System Description

2.1.Block Diagram



2.2.Functional Description

In the above block diagram, the black boxes represent the external libraries that are used. The green boxes are additional Solr features that were implemented in the system.

2.2.1.Indexing

The Wikipedia xml pages are are fed into the system to build the index. We are indexing wikipedia documents comprising of three categories: person, organization and places. So naturally the first step would be to categorize the documents based on the info box type. The next step is to extract the templates for the several fields present in the info box, for example: the date, country flags, name etc. One peculiar API we use here is the media wiki API to extract the country names from the flag codes that are available in the info boxes.

Once the fields have been extracted, we parse it out and feed it into Solr directly for indexing using SolrJ. It is to be noted that we directly pass the parsed text into Solr environment for indexing. This is done by altering the Solr schema so that it identifies the Wikipedia document directly. The details about the schema is provided in a later chapter.

2.2.2.Query Processing

Once the index has been built, the user queries on the index to obtain the response. In order identify the user's information need, we process the query provided by the user.

Named Entity Recognition

The first step in query processing is to identify the document that contains the answer to the user's query. This is achieved by using the Stanford NLP which has a pre trained deployable Named Entity Recognizer. This identifies if the query has any words which fall under the categories: person, location, date, organization, money. With this information, the system can narrow down its search substantially.

Parts of Speech Tagger

The system utilizes, a Parts of Speech(POS) tagger, in the next step. This serves two purposes, one is that it identifies the question words (why, what, how, where, when,which) and secondly it identifies the keywords required to query the index. We presume that, these words will be the nouns and verbs. Using the question words, we can make an estimate on the type of answer the user is expecting. For example, a query with the word when could point to a date, while a query with where points to a location.

Determining the Field to match

Using the POS tagger and the Named Entity recognizer to narrow down the search we arrive at the keyword. Now the next step is to find the exact field to match in-order to extract the answer that the user seeks. Since our system does not restrict the user to a certain set of queries, there is a situation that the same result can be fetched by formulating the query in different ways. For example, the wikipedia markup contains the field spouse. Suppose the user query is:

"Who is Kathie Kay's husband?"

The answer that the system should return is similar to the answer returned when the user replaces

husband by the word spouse in the above query. It is easy to spot that husband and spouse are synonyms of each other. So to make the system understand that husband is equivalent to spouse we use the WordNet API developed my the students of Stanford to retrieve similar words of all the words similar to all of the available fields in a map. We use this Map data structure to then identify alternate queries that the user can give and hence find the best field to retrieve.

One other issue that may arise is that the user may make a spelling error while query. This is resolved by finding the edit distance of the obtained keyword with the every field available and with the synonym map that was constructed as said before. The field that retrieves the lowest distance is chosen as the best field to match.

Multi Level Querying

While the system works well when the POS tagger and the Named Entity recognizer fires properly, for a few cases they fail to process the query terms properly. In such a case the search may fail. In order to over come this issue, the system employs multiple levels of querying. The situation in which each level will be activated is discussed below:

Level 1: This level is successful if the Named Entity recognizer is able to correctly recognize the entity.In this case a simple category filter is applied to the info-box type(Person, Organization, Location).This narrows the search to a single info box whose value is the recognized named entity.

Level 2: Suppose Level 1 fails, this then the execution comes to this level. This level succeeds when the Named entity is recognized wrongly in this case the category filter is removed and the recognized entity is queried against all the document names .

Level 3: If both Level 1 and Level 2 fails the query comes to this level. This level succeeds when the entity type is correctly detected but none of the info-box names matches with it. For example consider the query containing the named entity 'India'. The system correctly identifies it as a Location and applies a category filter. But the info-box name for the document containing information about India is Republic of India. So in this case we fire a wild card query on the info box name to obtain the result.

Level 4: It is similar to Level 3 but it does not narrow down the search using the category filter. It searches all of the available info-boxes.

Level 5: Suppose Level 2 fails, but the Named Entity type is correctly recognized then a category filter is applied to retrieve documents of that type. Now instead of searching the info-box name, we check a default text field that contains all the fields in the info box and its corresponding value. The top documents that have the keyword located in any field is retrieved.

Level 6: This level is similar to the pervious level but it does not have the category filter applied. So this level searches all the fields in all of the available info-boxes.

Now once the document is identified, then the field to match is then identified by comparing it with the available field list and using the synonym map obtained using word-net as explained before. This field to match is then queried on the index and the response is obtained for the user.

Handling Reverse Queries and How Queries

Two peculiar cases of queries are the how queries and the reverse query.There are 2 types of how queries, we will look at them with examples. Suppose the user asks

"How many spouse does Kathie Kay have?"

The info-boxes contain the name of the people, and not the count so in this case we must deviate from the normal execution and give a count of the entries in that field.Suppose he asks:

"How did Gandhi die?"

The answer is the value in the death_cause field. So we have a list of countable field types and if the question type is detected as "how" then the count of the entries in the field is taken, else the field value is returned as the result.

Reverse queries are queries that return the info-box name as the answer. An example of a reverse query is:

"Who died on 27- Jun -1989 ?"

This case the result will be an info-box name. To identify wether the user's query is a reverse query, Name entity recognizer is applied on the query. If the entity type that is recognized is Money or Date then we presume that it is a reverse query. The POS tagging is used to identify the best field to query as usual and a named query <Named entity:best field to query> is issued to the index and the answer is retrieved.

3.Universal Strong Points and Features

3.1.Strong Points

- The system does not limit the number of queries the user can choose from. The system also allows the user to give the query in natural language. This robustness makes the sets the system apart.
- The System supports queries of type how many, what,which, when, where, how,who.
- The system is able to answer reverse queries and queries that have some analytical value. For example the query:

  "Which companies had foundation in 1982"

  This will return the list of all companies that where founded in 1982.
- The system returns multiple results when required.
- It has a simplistic user interface which provides a pleasant experience to the users. It also responds to the query instantly.

3.2.Features

3.2.1.More Like This

The QA system apart from displaying the user the response to the query, is also able to give the user a list of wikipedia pages that are similar to the named entity in the given query.

3.2.3.Spell Checking

The system employs an n-gram based spelling checker that tries to find any errors that the user may have made while querying.

3.2.4.Edit Distance (Simitrics)

This external jar is used to find the best matched field to query. For example, consider the query:

"When was Apple formed?"

The info-box of Apple contains a field named 'formation'. A simple match of the obtained keyword would try to match formation with formed and this would fail. Hence the edit distance between the obtained keyword and all the available fields is made. The field that gives the minimum distance is identified as the field to match.

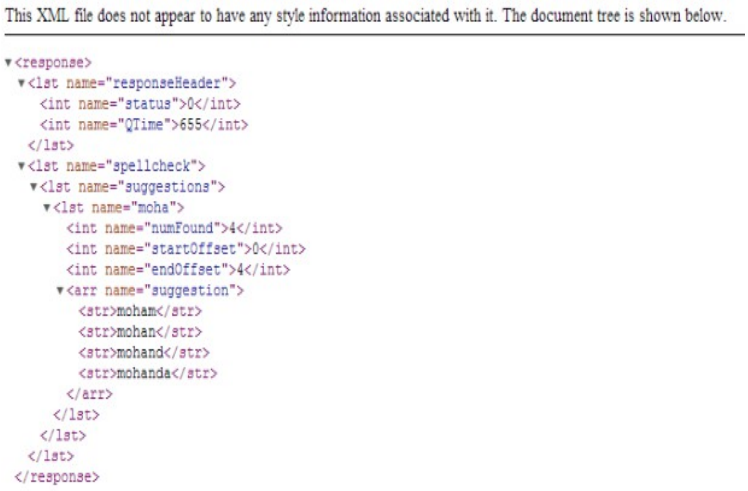
3.2.5.WordNet

Since natural language queries are permitted the user is free to give the same query in different words. We use the getSimilar feature of WordNet to extract the synonyms of each keyword and equate it to its original keyword.

3.2.6.AutoSuggest

The auto suggest request handler has been configured in SolrConfig. We have chosen the entity name field for the suggesting possible matches to the given query. Because our system supports, natural language queries providing dynamic suggestions using Ajax becomes complicated. We have however implemented the same in the Solr implementation. It can accessed via the URL:

http://localhost:8983/solr/suggest?q=moha



Here we can see that the auto suggest handler gives suggestions based on matching entity name for the string 'moha'.

4.Configuration and Solr Schema Tweaks

4.1.Solr Schema

This is the file that Solr uses to identify the fields in the info-boxes and index them. We have defined our own schema with field types that were extracted directly from the wiki markup pages. Additionally we have defined copy fields to club related fields. For example all of the name fields like native_name, full_name is set to a single name field this clears a lot of confusion while processing the user query because the edit distance for several of these fields turns out to be the same. The figure below shows a screen shot of the schema.xml file and the copy field entires in the schema.

```
<!-- catchall text field that indexes tokens both normally and in reverse for efficient
     leading wildcard queries. -->
 <field name="text_rev" type="text_general_rev" indexed="true" stored="false" multiValued="true"/>

 <field name="entity_name" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="birth" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="death" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="family" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="debut_info" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="work" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="mlt_place" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="player_info" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="politics_info" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="life_info" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="leader" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="coach" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="formed" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="income_info" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="finance_info" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="service_info" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="cricket_match_info" type="text_general" indexed="true" stored="false" multiValued="true"/>
 <field name="area_info" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="population_estimate_info" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="population_census_info" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="population_density" type="text_general" indexed="true" stored="true" multiValued="true"/>
 <field name="population_rank_info" type="text_general" indexed="true" stored="true" multiValued="true"/>

 <field name="id"    type="text_general"  indexed="true" stored="true" required="true"/>
 <field name="infobox_type" type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="honorific_prefix"       type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="name"        type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="notable_instruments"        type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="age" type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="net_worth" type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="president" type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="full_name"        type="text_general"  indexed="true" stored="true" multiValued="true" />
 <field name="honorific_suffix"        type="text_general"  indexed="true" stored="true" multiValued="true" />
```

```
  <copyField source="name" dest="entity_name"/>
  <copyField source="full_name"  dest="entity_name"/>
  <copyField source="honorific_prefix"  dest="entity_name"/>
  <copyField source="honorific_suffix"  dest="entity_name"/>
  <copyField source="native_name"  dest="entity_name"/>
  <copyField source="native_name_lang"  dest="entity_name"/>
  <copyField source="birth_name"  dest="entity_name"/>
  <copyField source="other_names"  dest="entity_name"/>
  <copyField source="nickname"  dest="entity_name"/>
  <copyField source="pseudonym"  dest="entity_name"/>

  <copyField source="birth_name"  dest="birth"/>
  <copyField source="birth_date"  dest="birth"/>
  <copyField source="birth_place"  dest="birth"/>

  <copyField source="death_date"  dest="death"/>
  <copyField source="death_place"  dest="death"/>
  <copyField source="death_cause"  dest="death"/>

  <copyField source="spouse"  dest="family"/>
  <copyField source="partner"  dest="family"/>
  <copyField source="children"  dest="family"/>
  <copyField source="parents"  dest="family"/>
  <copyField source="relatives"  dest="family"/>
```

We have also defined a reference to the ngram-Filter Factory for the spelling checker that we have implemented. Below is a figure showing the reference.

```
<types>
    <fieldtype name="ngram" stored="false" indexed="true" class="solr.TextField">
        <analyzer>
            <tokenizer class="solr.KeywordTokenizerFactory"/>
            <filter class="solr.LowerCaseFilterFactory"/>
            <filter class="solr.NGramFilterFactory" minGramSize="2"
                maxGramSize="10"/>
        </analyzer>
    </fieldtype>
    <fieldtype name="prefix" stored="false" indexed="true" class="solr.TextField">
    <analyzer type="index">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
     <filter class="solr.LowerCaseFilterFactory"/>
     <filter class="solr.EdgeNGramFilterFactory"
     minGramSize="2" maxGramSize="10"/>
     </analyzer>
<analyzer type="query">
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
<filter class="solr.LowerCaseFilterFactory"/>
</analyzer>
</fieldtype>

</types>
 <fields>

    <field name="wordNGram" type="ngram" multiValued="true"/>

    <copyField source="name" dest="wordNGram" />
```

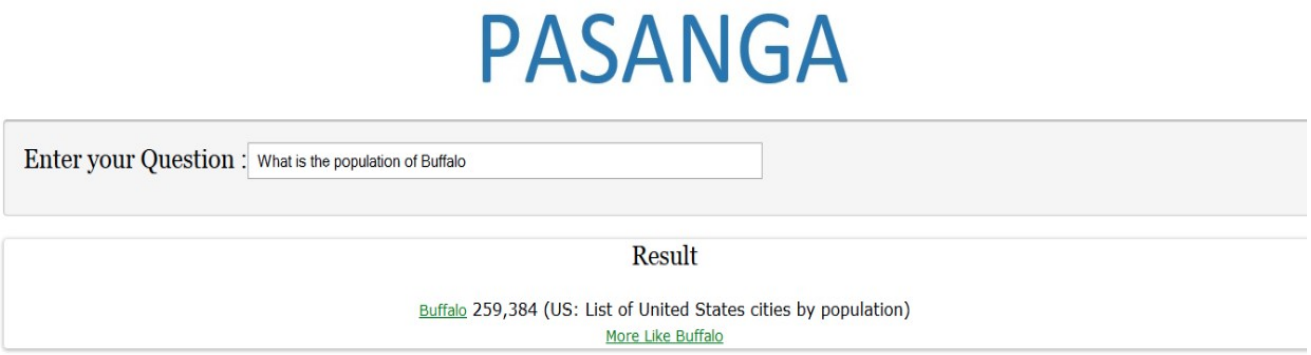
4.2.SolrConfig

These is the place were request handlers for several of Solr's features are listed. We have defined a request handler for the more like this feature that ships in with Solr.

```
    -->
  <requestHandler name="/tvrh" class="solr.SearchHandler" startup="lazy">
    <lst name="defaults">
      <str name="df">text</str>
      <bool name="tv">true</bool>
    </lst>
    <arr name="last-components">
      <str>tvComponent</str>
    </arr>
  </requestHandler>

  <!-- More like this request Handler-->
    <requestHandler name="/mlt" class="solr.MoreLikeThisHandler">
    <lst name="defaults">
      <str name="mlt.fl">name,nickname,contributions</str>
      <int name="mlt.mindf">1</int>
    </lst>
  </requestHandler>

  <requestHandler class="org.apache.solr.handler.component.SearchHandler" name="/suggest">
    <lst name="defaults">
      <str name="spellcheck">true</str>
      <str name="spellcheck.dictionary">suggest</str>
      <str name="spellcheck.count">5</str>

    </lst>
    <arr name="components">
      <str>suggest</str>
    </arr>
  </requestHandler>
```

5.User Interface Screen Shots

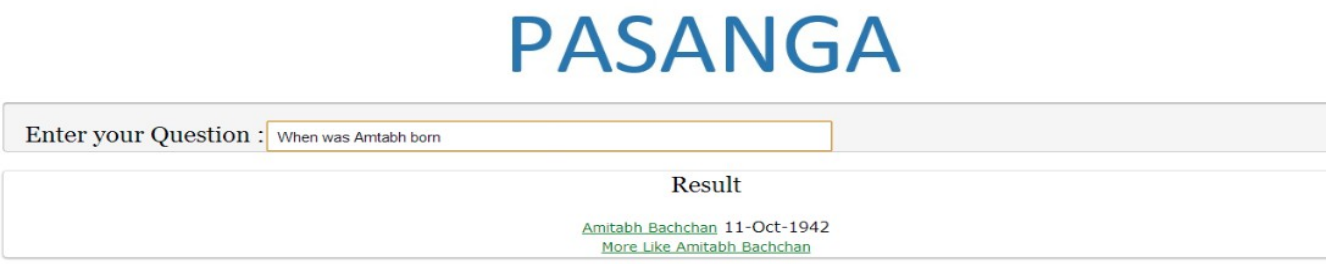
Below is a figure showing a simple query and a response



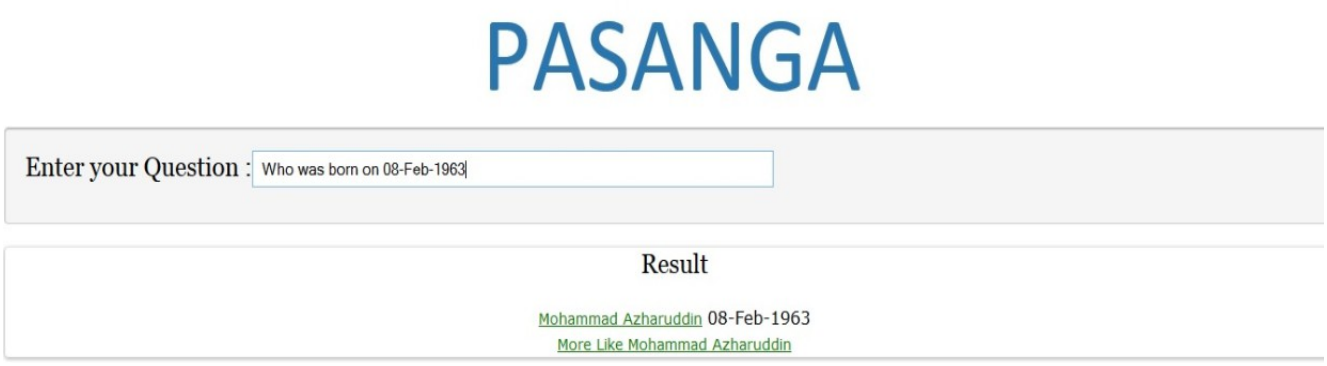
The figure below shows the state of the UI with the more like this feature.

Here the more like this handler suggests pages of similar interest, which on clicking redirects to the corresponding wikipedia page.

The below figure demonstrates the spelling check capabilities of the system.



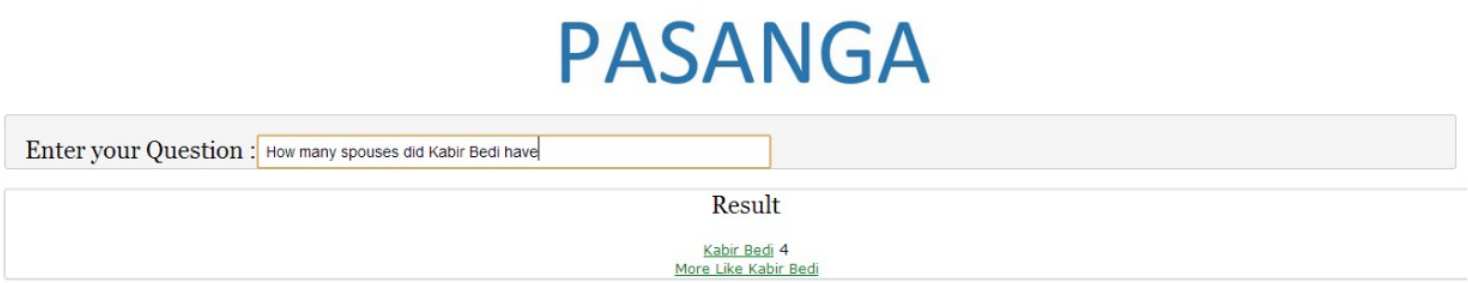
The below figure demonstrates a sample reverse query.



The figure shows another reverse query generating a list of results.

The image below is an example of a query returning multiple results.

Below is a an example of how query.



6.Solr Statistics

Index Size: 11,500.

Number of Fields indexed: 300+

Average Response time:

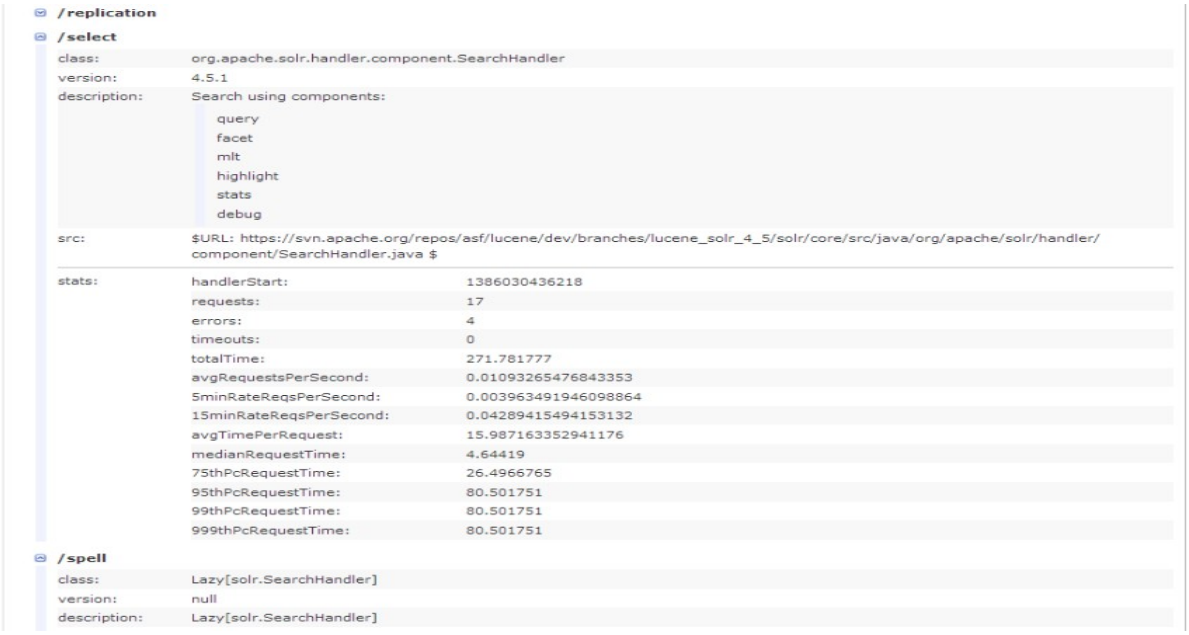


Image showing the Solr Statistics like the hit ratio:

documentCache

class:	org.apache.solr.search.LRUCache	
version:	1.0	
description:	LRU Cache(maxSize=512, initialSize=512)	
src:	$URL: https://svn.apache.org/repos/asf/lucene/dev/branches/lucene_solr_4_5/solr/core/src/java/org/apache/solr/search/LRUCache.java $	
stats:	lookups:	1265
	hits:	870
	hitratio:	0.69
	inserts:	395
	evictions:	0
	size:	395
	warmupTime:	0
	cumulative_lookups:	1265
	cumulative_hits:	870
	cumulative_hitratio:	0.69
	cumulative_inserts:	395
	cumulative_evictions:	0

7.Future Work

1. The system only responds to well built queries with a question words(what,how, when,where,which,who) and key words(nouns and verbs). If the user enters a raw query like “Steve Jobs” the system does not operate. We could implement clustering feature on the documents and display the clusters of related documents to the raw query.
2. The reverse query is partially implemented. We could improve on that front.
3. We could implement voice query.

8.Work Distribution

1. Agrawal, Saurabh- Multi Level Query Processing Algorithm, Reverse Query,More Like This,Spell Checking , Auto Suggest,Testing.
2. Manivasagam,Niranjan- WordNet Similarity, Stanford POS Tagger, Named Entity Recognition,Question Type Categorization,Documentation,.
3. Soundarajan, Shiyam Sundar- Parsing Wikipedia, JSP Implementation for UI, Basic AJAX code,Media Wiki for flags.
4. Thruvas Jeyakumar, Sabharinath Babu- Indexing Wikipedia, Schema Definition, Multilevel Query Processing, UI with AJAX.