Distributed QA System

Saurabh Pujar
M. S. in Computer Science
New York University
ssp437@nyu.edu

Bharathi Priyaa T.
M. S. in Computer Science
New York University
bt978@nyu.edu

Kshitiz Sethia
M. S. in Computer Science
New York University
kshitiz.sethia@nyu.edu

Abstract

Question Answering (QA) is a computer science discipline within the fields of information retrieval and natural language processing (NLP), which is concerned with building systems that automatically answer questions posed by humans in a natural language. We are trying to build a basic distributed version of this system which will give intelligent, meaningful and relevant answers at runtime by processing the information stored in a large corpus. The question will be processed and information will be retrieved by using Hadoop and MapReduce. Our current aim is to build such a system for user-specified topics.

Keywords—analytics, NLP, AI, map reduce, Hadoop

I. INTRODUCTION

The project aims to use large amount of data in a natural language text corpus in order to develop a system, which can understand and answer questions in natural language.

A question answer system will require access to a large amount of factual information. We plan to use three text corpora as our source of information. The corpora are Wikipedia, Wiki News and Reuters News Corpus. We plan to use Hadoop in order to query and process the information provided by text corpora. The question will be processed by an NLP component which will generate keywords from the question and use it to query the text corpus.

II. MOTIVATION

A system capable of answering questions has been an AI dream for many decades now. After widespread initial interest in the 1950's, this field saw a loss in popularity for many years which was later known as the AI winter. With the advent of Big Data and distributed computing technologies like Hadoop and MapReduce, it became possible to processing large amounts of information in reasonable time. This combined with statistical techniques like Machine Learning has effectively ended the AI winter and lead to a lot of advances in the field in recent years.

The amount of data being generated currently is enormous. A system which can interact with a user and intelligently responding using the large amounts of data as a knowledge base would be a very effective way to make data useful for ordinary users. Anyone who needs specific, meaningful information retrieved from a particular dataset in simple language can benefit from such a system. The same algorithm can be used to answer legal questions by going over legal data. Or health questions by going over medical data. For example, the user starts with a question on a topic and continues to ask questions on the same topic, our system will load data only based on his initial topic.

With every passing day more and more knowledge is added to the web and to build better and more accurate systems, it is important that we factor in this data. To build systems which respond quickly to user's questions, we can make use of big data technologies like Hadoop and MapReduce. These technologies allow us to process the large quantities of data in parallel and come with real-time answers to questions. We saw that what we were studying in class can be directly used to build modern QA systems.

III. RELATED WORK

There are two schools of thought on Question Answering. The earlier one operates on structured data. Structured data is commonly represented as RDF (Resource Description Framework) with triples of the form (subject, relation, object). For example: Barack Obama (subject), president of(relation), United States(object). The second school of thought operates on raw text from various sources. Currently, most of the systems are a hybrid approach, with question type classification deciding which underlying data source to use.

According to recent surveys of QA based on structured data [11], the different systems support different kinds of RDF data sources. There are various RDF data sources (and ontologies): DBPedia, Freebase amongst the major ones, MuzicBrainz, Mooney Geography and many small ones. The QA systems are two types, ontology specific and ontology independent. The major problem being that each ontology has its own vocabulary for relations and its own schema. This lack of standardization although makes RDF more usable, makes the use of QA systems on top of it more difficult. Amongst the ontology independent QA systems is FREyA [12], which beats all other such systems and is actively supported on Github by its creators. FREyA uses sophisticated NLP techniques to decipher the question, understand the underlying ontology and can answer questions with high accuracy.

A more recent trend, since 2002 majorly has been to use publicly available text on the internet to answer questions. Our system follows this school of thought.

Among modern QA systems, the most advanced is IBM's Watson and the systems abilities were on display when it won the game show Jeopardy competing with expert human players. Watson uses a combination of search engines and domain specific knowledge bases to come with natural language answers to compound questions. Building a system comparable in performance is beyond the scope of this project.

IV. DESIGN

The project will basically have three components and many subcomponents. The main components are as follows:

1. Question processing (ProcessQueryJson.py)
   a. Query Formulation
   b. Answer Type Detection
2. Answer retrieval (Map-reduce, searcher. Pig)
   a. Document Retrieval
   b. Document Ranking
3. Answer processing (ProcessAnswer.py)

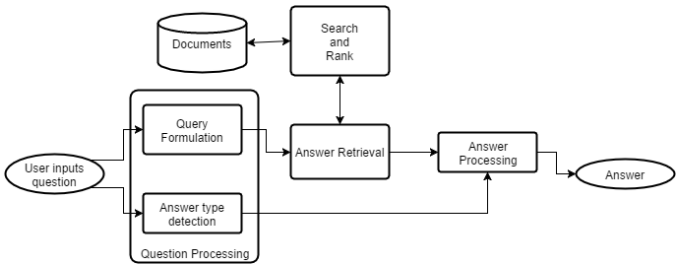

Figure 1: Design of our QA system

1) Question Processing:

The given question is used to generate keywords which are then used to narrow down the answer. Python programming language was used to code this component. Python was chosen because of the abundance of NLP and machine learning libraries which were critical for our project. Question processing involves query formulation and answer type detection. For query formulation it is essential to determine which words in the question will ultimately be the deciding factors for a correct answer. We can ignore the stop-words like "is, the, and" and focus more on the nouns, adjectives and verbs. To identify these components we use a part of speech tagger. Once we have the nouns we can use named entity recognition to determine which named entities are referred to in the question. Based on the type of question, we can also determine what named entity, if any, is expected as an answer. For example, if the question starts with a "who" the expected answer is a PERSON and if the question starts with a "where" the answer would be a location.

To limit the scope of the project, we decided to restrict the type of questions to simple factoid questions. A fact based questions will have a simple fact as an answer. The nature of the question is such that it need not be broken down into sub questions which have to be resolved separately. For example "Where is the Louvre located?" The answer is "Paris". The answer to this question is a straight forward fact. A more complex question would need to be broken down into two or more factoid questions to be answered. Consider the question "William Wilkinson's "An account of the principalities of Wallachia and Moldova" inspired this author's most famous novel". The answer is "Bram Stoker". In the example below, answer to the first part of the question is a clue for the second part of the question. This question was correctly answered by IBM's Watson super computer in the game show Jeopardy. This limiting of question space was done given the limited time frame we had to complete the project.

One of the primary tasks once given a question is to determine what type of an answer is expected. Many types of factual questions will expect a named entity as a response. Named entities, which are usually proper nouns, pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities etc. For example, the question "Who founded Virgin Airlines?" has a "PERSON" as the answer. The question "Which country has the largest population?" has a "GPE" or a Geo-political entity as the answer. Both PERSON and GPE are examples of NLTK named entity tags. The answer type can be determined using some rule based techniques or machine learning or a combination of both. Currently we are using rule based techniques because the question types we are handling are limited. So a "who" type question will result in an answer type "PERSON" and a where type question will result in an answer type "LOCATION" or "GPE".

However not all answers will return a named entity as an answer. For example, a question like "Who is Narendra Modi?" will not return a PERSON as the answer. Similarly "what" questions, "how" questions and any "Who PERSON?" questions will have more descriptive answers and will not return any named entities. In such cases emphasis is on the "headword" which is the first noun after the "Wh-" word. For example, "What is the **capital** of Texas?" will have emphasis on "capital" and we will look for sentences which have the word capital in them.

In query formulation we will pick out certain key words from the question which we will use to query the database and retrieve documents. For the purposes of the project, we will ignore the frequently occurring stop words like "is", "the", "of", "in", "his" etc. We are only considering nouns, adjectives and in some cases verbs to formulate our queries. Given a question, our first task would be to determine the parts of speech of each word in the question. This is achieved using the NLTK library which is open source.

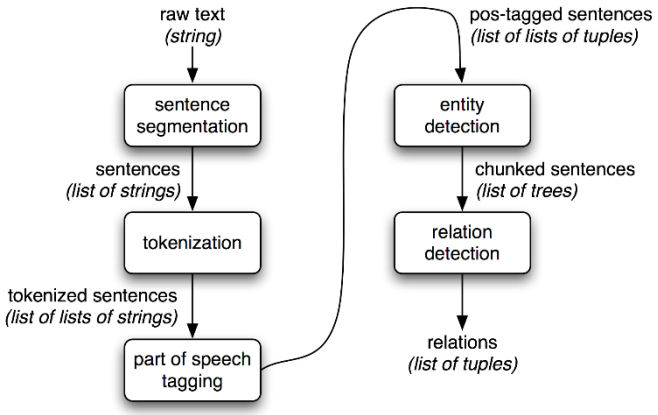

Figure 2: Flow of Question Processing component

Given a question we first break it down into a list of words and then try and determine the part of speech of each word. The standard way to do this is using Hidden Markov Model to train over a large data set of human annotated part of speech tags. Once this is done we get a set of possible part of speech tags for each word in the sentence and the probability of each tag. NLTK also considers the probability of the tag given the probability of the preceding word(s). Once we have this, we determine the most likely tag sequence using the Viterbi algorithm. Given a good enough data set, we can get accuracy of around 95% which is close to state of the art. The part of speech tagging component is built on top of pre-trained Stanford NLP parser and gives state of the art performance.

Consider the below example:

Q: Who is the President of the United States?

After running part of speech tagger we get the following output:

[('Who', 'WP'), ('is', 'VBZ'), ('the', 'DT'), ('President', 'NNP'), ('of', 'IN'), ('the', 'DT'), ('United', 'NNP'), ('States', 'NNPS'), ('?', '.')]

We then use this output to determine if any named entities exist in the POS sequence. Given below is the output:

(S Who/WP is/VBZ the/DT President/NNP of/IN the/DT
(GPE United/NNP States/NNPS)
?/.)

As you can see, United States being a geo political entity (GPE) was correctly recognized. Now that we know that this is a "Who" question and the named entity in it is not a PERSON and hence we can conclude that the response is a PERSON. The other case in "Who" questions can be like the one described below.

Q: Who is Barack Obama?

After part of speech tagging:

[('Who', 'WP'), ('is', 'VBZ'), ('Barack', 'NNP'), ('Obama', 'NNP'), ('?', '.')]

After running the named entity recognition component we get:

(S Who/WP is/VBZ (PERSON Barack/NNP Obama/NNP) ?/.)

We now know that the question is about a person Barack Obama. The answer in this case cannot be a PERSON type. It would be a description that can be anything. But the headword Barack and the noun Obama would be important.

For selecting words to build our query, we first begin by ignoring the "Wh-" word and the stop words. Considering the first example, we are left with the words "President", "United" and "States". We can club United and States together and only look for sentences which contain these two words one after the other. No other word can come between United and States and if it does, the words are being used in ways which will not be useful to us. But this is applicable to "United States" and not applicable to "Barack Obama". The answer sentence may state that "Barack Obama" or "Obama" or "Barack Hussain Obama" is the president of the United States. All are correct. Hence we look for "Barack" and "Obama" separately in the database.

In the query formulation, we assign the highest priority to the Named Entity or headword in the question. Followed by the nouns and then by the adjectives.

TYPE OF ENTITY	PRIORITY
NAMED ENTITY, HEAD WORD	10
NOUN	7
ADJECTIVE	5

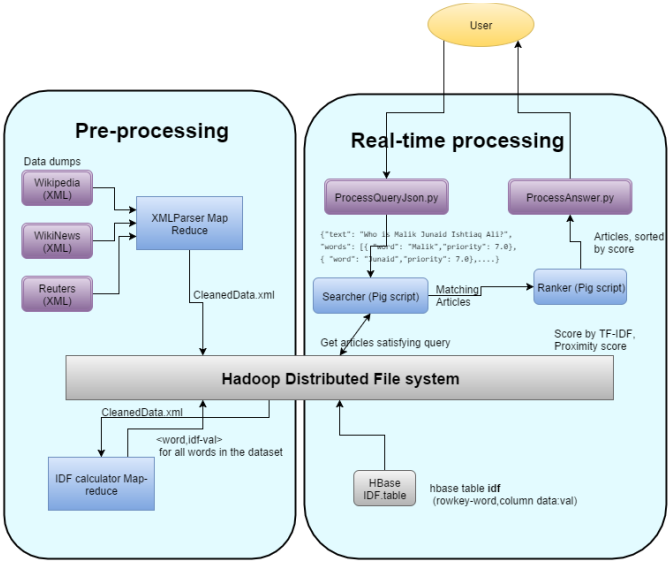

Figure 3: Detailed design of our QA system

2) Answer Retrieval:

a) Choice of dataset:

Our learnings from reading papers which used DBPedia like knowledge base is that, these systems are not scalable and are not up-to-date. DBPedia extracts structured information

from Wikipedia and curates them into RDF formatted data. RDF is a resource description framework, which has an annotation for each data word describing attributes of that data. For example the RDF of Michelle Obama [10] will have attributes like birthdate, nationality, etc. Although this dataset provides a plethora of information about a particular subject, it is not scalable. For e.g. If the spouse of Michelle Obama were to change, the entire RDF needs to be re-engineered by DBPedia and release. Thus the absence of live information does not make DBPedia a reliable knowledge base for a question answering system.

Thus we settled on choosing Wikipedia dataset which reflects up to date information stored as Wikipedia articles. The database dumps are updated every day. We also used WikiNews as a news data source for new events. For referring to important events in the past, we are referring to the Reuters News Corpus.

Wikipedia articles were conventionally stored as XML formatted files with the content of each articles stored within <Text><Title><ID> tags. We used a Hadoop MapReduce job to clean up the data. The cleaned-up data is stored in a Pig Table for further retrieval and processing. We used HBase to store IDF (Inverse Document Frequency) values which will help in scoring/ranking candidate answers to the question.

b) XMLParser Module:

Since the input data is an XML formatted data, we had to figure out a way to allow Mapper class take an XML input data type as input. Since each of the XML page data must be processed as a whole, to split and stream them as text bytes was not feasible in Native Hadoop. Hadoop MapReduce does not have built-in support for XML. Hence we used Apache Mahout's XMLInputFormat class, which provides support for serializing/deserializing XML documents. XMLInputFormat class extends the TextInputFormat class of Hadoop library and takes in two parameters namely the <start> and <end> tags. The text which is embedded between the <start> and <end> tags are processed as a whole.

To store our input data in a specific format and to enable transfer of data from Mapper to reducer, we wrote a custom-serialization object called WikiDataWritable. Our WikiMapper.java handles the conversion of XMLStreambytes and creates a WikiDataWritable Object. WikiDataWritable is a custom-serialization object with attributed defining our preferred XMLFormat. WikiDataWritable implements the Writable Interface to serialize and de-serialize XMLData when read/written as Input/Out Stream. We chose to have the output format from MapReduce as a common XML which will help us query multiple data sources seamlessly. Also XML provides a seamless integration into Pig, which is our second module.

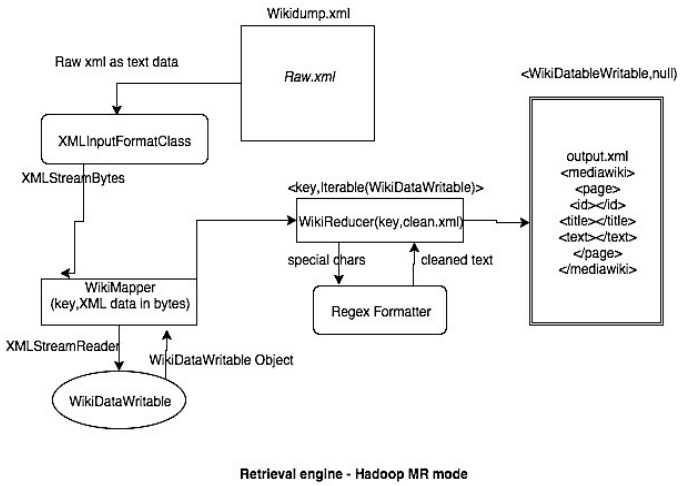

Figure 4: Retrieval engine – Hadoop MR mode

c) Inverse Document Frequency calculator:

To understand how important a word is in the Query, we need to calculate the *tf-idf* value for each term in the query. *Tf* refers to the term frequency (number of times a word 't' occurs in a document). *IDF* value is the inverse document frequency (how important is this term 't' among all the documents). *Tf-idf* weight is used as a measure to score/rank set of possible candidate answers. The formulae for both are given below:

$$TF - IDF_{t,n} = TF_{t,n} * IDF_t$$
$$t = term, n = num\ of\ documents$$
$$TF_{t,n} = \frac{a}{b}$$
$$a = number\ of\ times\ term\ t\ occurs\ in\ the\ document$$
$$b = total\ number\ of\ term\ in\ the\ document$$

$$IDF_t = \log_e\left(\frac{N}{df_i}\right)$$
$$N = Total\ number\ of\ documents$$
$$df_i = documents\ containing\ term\ t$$

Figure 5: TF-IDF score description

We write a Hadoop MapReduce job to scan all sets of documents (given by XMLParser Module) take frequency values of all possible words in the document. The output of this phase is stored as a CSV in hdfs to be later loaded into HBase table *idf*.

d) Searcher.pig:

We use Apache pig to load the cleaned wiki dataset as a table into Pig. Pig offers XML support though its XPath UDF in piggybank.jar. We load our cleaned.xml data as a table into Pig (WikiTable). The query output (as Json) from Question processing phase is also loaded into Pig using the native JsonLoader of Pig (QueryTable) The choice of pig over hive arrives from this aspect of easy integration of semi-structures data like XML and JSON. Term-frequencies as described above are calculated by a series of joins between the WikiTable and QueryTable. The idf-values are loaded from HBASE using HBaseStorage.

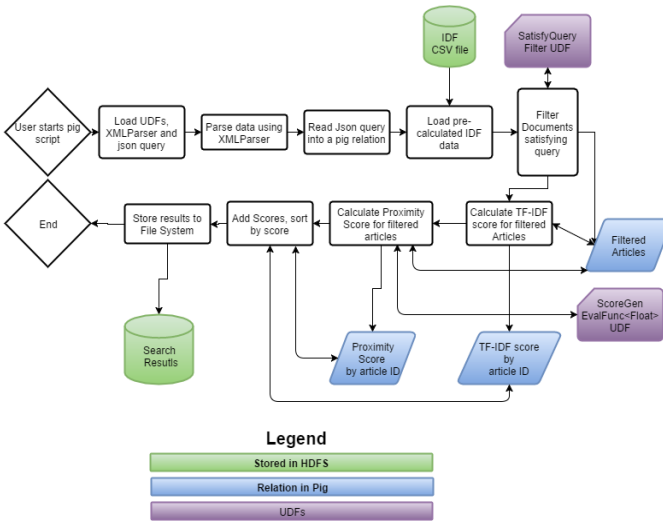

Figure 6: Data-pipeline in PIG of searching articles

e) Filtering articles:

The pig script for searching over the corpus takes as input argument a json that is structured as follows:

```
{
        "text": "<Actual question typed by user>",
        "answerType": "<The NLTK named entity tag of answer>",
        "words": [{
                "priority": <float value>,
                "word": "word 1"
                "alternatives":["alt1", "alt2"...]
        },...
        ]
}
```

Figure 7: Format of json received from Query Processing component

The pig script uses a Filter UDF, which returns *true* for articles that have any of the whole words (ignoring case) in the "words" part of the json, or their corresponding alternatives. This results in a lot of articles, but that's a problem for the ranking system.

f) Ranking Articles:

The ranking UDF called from the pig script returns a score for each article. This UDF is only called for the filtered articles, and the UDF scores each article independent of any other article. A higher score signifies that the rank is higher.

We use two types of ranking heuristics:

1) tf-idf score of matched words in the article
2) proximity score of matched words in the article

Before we discuss the heuristics, we need to establish that we could also learn to rank, using user data, but we need heuristics to begin with. We can collect user data, by monitoring on a web UI the rank of the answer (in case of multiple answers) or correctness of answer. The inverse rank of chosen answer or the rejection of answer can then be used to tune the ranking algorithm internally by gradient descent. Back to our heuristics.

The proximity score is the measure of the closeness of the query words in a document. It is indicative of the relevance of the document to the actual answer. As an example, for the query words "president", "United", "States" should give higher priority to an article having them in the same sentence, versus an article having them in different paragraphs.

We use a common metric called "Slop Distance" for measuring distance between query words in a sentence. The slop distance is defined as the number of non-query words in the minimum length contiguous array of words containing all the matched query words of an article. Here are a few examples for the previously discussed query words:

Sentence	Slop Distance
The President of The United States	2
United States President, Barack Obama	0
The President of India visited The United States	4

Figure 8: Examples of Slop Distance

To measure this slop distance we designed an algorithm, which calculates the Slop distance in O(n) time complexity and O(number of query words) space complexity. It is present in "ScoreGen.java#minWindow()"

Given that we have our slop distance, we now need to generate a score which will be able to compare various documents with their own number of matched terms and slop distances. We need a function which takes (slop distance, number of matched words) and returns a score. Let's denote slop distance as "x", number of matched words as "n" and score as y = f(x,n).

The first solution is to vary each proximity score between the number of matched query words in the article and +1. For example for three matched terms, we get a score between three and four.

This solution however would fail in the following case. Article 1 has 4 matched terms with a slop distance of 100 (consider all the words being mentioned in two different paragraphs). Article 2 however has three terms matched, but with a slop distance of 5 (maybe, in the same sentence). Clearly, article 2 is more likely to contain the answer, our solution above is not usable.

We need a function, which till a certain threshold th, scores between [n, n+1] for an article and below that afterwards. This would choose an article with (n-1) matched terms over an article with (n) matched terms too far apart. We would also like that the threshold be a function of number of matched terms as we would need to keep a bigger slop window for 10 terms in comparison to 2 terms. So th needs to be th(n)

Let's consider three types of such functions in the diagram below. We are considering n = 2, and th(n) = 2n. That is the number of matched terms are 2, and if they have a slop distance more than 4, we score them below 2.

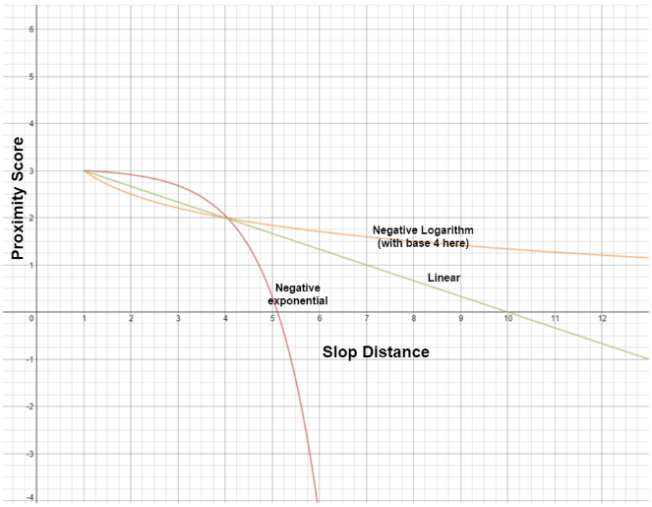

Figure 9: Comparison of choices for y=f(x,n) for n=2

The negative exponential penalizes less for small x, but grows very fast. The negative logarithm penalizes okay till x=2n, but never penalizes high slop a lot (goes very slowly to –infinity). The negative linear function, gets the best of both worlds and also reaches –infinity at an appropriate pace. We established this intuition after repeated search rankings with ambiguous examples of the kind discussed above. We choose linear as our preferred f(x,n), and fixed on th(n) = 2n after these experiments.

$$f(x,n) = \frac{-x}{2n-1} + \frac{2n^2+n}{2n-1}$$

Figure 10: Chosen proximity function

This is how our f(x,n) looks for $n \in \{1,2,3,4,5,6\}$ and $x \in [1,15]$. The black line denotes the threshold function below which the score is below n.

Figure 11: Chosen f(x,n)

n	1	2	3	4	5
Color	Blue	Red	Green	Purple	Orange

Figure 12: Colors of line for different "n" values

Now, we have an appropriate f(x,n) which scores within it's bound and penalizes appropriately. Let's see some toy results to see the output of our function.

Text	Proximity Score
Barack Obama is the President of the United States	3.8
The President of India visited the United States	3.4
President of India	2
Random text snippet	0

Figure 13: Sample text snippets and their proximity scores calculated by our heuristic.

3) Distributed system architecture:

In the below few sections, we investigate our choices of appropriate distributed systems technologies over their alternatives.

Hadoop for cleaning data: Although there was a lack of XML support in Native Hadoop, we used Hadoop's distributive capabilities to process the entire Wikidataset. MapReduce is much more scalable than existing file-indexing systems like solr.

Pig for retrieving and ranking data: The inbuilt support for loaded semi-structured data like XML's and Json were better with Pig than with Hive. Also, Pig gives the user the possibility to write seamless User defined functions(UDF's) to work on bags/tuples of data in numerous ways. We also created a filtering UDF in Hive by sub classing GenericUDF. This achieved the same goals as SatisfyQuery UDF in Pigs. We chose Pig over Hive later as it had very good support for parsing XML data. This flexibility allowed us to use Pig instead of Hive.

HBase as word,idf value store : HBase is a distributed, scalable data store built on top of HDFS. In our case, Hbase table stores the idf values for all the words in the data set. This could run into millions of rows. HBase stores <word,val> pair as a dictionary in hdfs. This makes it more scalable and retrieval and lookups are faster. In future, if we would like to add attributes to a keyword(noun/ver/adjective), we just have to define new column family in existing columns.

Pig UDF's over Piglatin: We are using Regex to filter documents, which contain the target word. In the design we also support the use of alternatives to words, which are semantically appropriate synonyms. This complex processing is what prompted us to write our own UDF. The SatisfyQuery UDF subclasses FilterFunc from pig and returns a true for any document satisfying the json query.

For scoring the articles, we have created our own function for score calculation. This also required the use of UDFs in pig. ScoreGen subclasses EvalFunc<Float> to generate a float score for each article. Moreover, Pig UDFs are extremely easy to write and integrate with the pig scripts. They also give the added advantage of using Java. This facilitated us to take a complex json as input to the UDF and de-serialize it using Jackson libraries.

4) Answer Processing:

Once the passage retrieval engine has returned a set of paragraphs which may contain the answer to our query, the answer processing component goes through each sentence of each paragraph in order to determine if it contains the answer to the question. Once again, for the sake of simplicity given the time frame, we consider just one candidate answer and as soon as we find an appropriate sentence, we exit the application. Often there are many candidate answers and various machine learning techniques can be employed in order to choose the best one.

To find the correct answer, we consider one sentence of one paragraph at a time. We then run part of speech tagging and named entity recognition on this sentence the same way as above. Once we have the POS tags and the named entities, we then check and see if the sentence has everything we are looking for. Consider the example below:

Q: Where is the capital of England?

So we are looking for a sentence which contains a GPE (other than England) and the words "capital" and "England". Also consider a sample paragraph given below returned by the information retrieval component.

"*England's terrain mostly comprises low hills and plains, especially in central and southern England. However, there are uplands in the north (for example, the mountainous Lake District, Pennines, and Yorkshire Dales) and in the south west (for example, Dartmoor and the Cotswolds).* ***The capital of England is London****, which is the largest metropolitan area in the United Kingdom and the European Union.[nb 1] England's population of over 53 million comprises 84% of the population of the United Kingdom, largely concentrated around London, the South East, and conurbations in the Midlands, the North West, the North East and Yorkshire, which each developed as major industrial regions during the 19th century.[9]*"

Only one sentence "The capital of England is London, which is the largest metropolitan area in the United Kingdom and the European Union." qualifies all of our criteria and is extracted and printed as the answer. Consider the following question:

Q: Who was named the 2009 Nobel Peace Prize laureate?

In this case we are looking for a sentence which contains a PERSON and the words "Nobel", "Peace", "Prize" and "laurate". A sample paragraph as retrieved is given below.

"In 2004, Obama received national attention during his campaign to represent Illinois in the United States Senate with his victory in the March Democratic Party primary, his keynote address at the Democratic National Convention in July, and his election to the Senate in November. He began his presidential campaign in 2007 and, after a close primary campaign against Hillary Rodham Clinton in 2008, he won sufficient delegates in the Democratic Party primaries to receive the presidential nomination. He then defeated Republican nominee John McCain in the general election, and was inaugurated as president on January 20, 2009. ***Nine months after his inauguration, Obama was named the 2009 Nobel Peace Prize laureate.****"*

The last sentence contains the answer and is correctly selected and printed.

One obvious improvement would be to print only the answer and not any other part of the sentence. So in the above example, "Nine months after the inauguration" is additional information that need not be given and only "Obama" can be printed as the answer. In this example the solution is trivial and we simply print the named entity found in the sentence. But consider the answer in the previous example. In "The capital of England is London, which is the largest metropolitan area in the United Kingdom and the European Union." There are three location entities London, United Kingdom and European Union. Any one of these could be the answer and we would need to rank the potential answers using features like [8] Pattern Match, keyword distance novelty factor etc. This is something we hope to incorporate in our future endeavors.

V. RESULTS

(Future… In this section, you can describe: Your experimental setup/issues with data/performance/etc. Describe your experiments, describe what you learned. Did you prove or disprove your hypothesis? Were some results unexpected? Why?)

Experimental setup issues: We faced some hurdles in choosing the right distributed technology for retrieving documents. Although Hive and Pig were both possible systems, we settled on Pig based on the ease of setup and flexibility of writing UDF's to interact with Hive. The choice of Hbase as distributed key-value store although was intuitive we faced some hurdles when retrieving records from Hbase.

The integration of Hbase with Pig is not possible with the current stable release(1.12) of Hbase. Hence we had to use an older version(0.98) to make Hbase work with Pig. Also, since Hbase also stores the timestamp data (at which a record was inserteD), It makes more sence to use Hase for range based queries which involve Timestamp. Our learning out of this experiment is that Hbase was a poor choice of Distributed key-value store and a better choice would've been Hive. Another issue we faced was in writing PIG UDF's which accepts both a tuple and a databag as arguments. The scenario arose when we wanted to send the QueryTable to a PIG UDF along with tuples(Title,Text). To workaround this issue, we had to write expensive PIG Latin joins which could otherwise have been handled in PIG Udf's.

We had to narrow down the scope of questions we were answering for this project because this was turning out to be a large undertaking. We did this by picking simple factoid based questions. We also made some simplifying assumptions in terms and used some heuristics for ranking and considering only one answer which matched our requirements.

Below are some of the questions we were able to answer correctly.

Q: Who is Barack Obama?

A: Barack Hussein Obama II (born August 4, 1961) is the 44th and current President of the United States, as well as the first African American to hold the office

Q: Who is the President of the United States?

A: Barack Hussein Obama II (born August 4, 1961) is the 44th and current President of the United States, as well as the first African American to hold the office

Q: Who was named the 2009 Nobel Peace Prize laureate?

A: Nine months after his inauguration, Obama was named the 2009 Nobel Peace Prize laureate.

Q: What is the capital of England?

A: The capital of England is London, which is the largest metropolitan area in the United Kingdom and the European Union

Q: Who is Malik Junaid Ishtiaq Ali?

A: Malik Junaid Ishtiaq Ali son of Shaheed Abdul Razaq Jaora who was the Journalist and martyred by the smugglers due to the expose off the crime of the smugglers in Mianwali

Not answered correctly:

Q: When was Wales included in The Kingdom of England?

<No Answer>

Initial hypothesis and results: Our hypothesis was that It is feasible to build a Text based Question answering system using distributed system technologies to achieve same or better performance over similar QA systems build using traditional technologies like solr/lucene or SQL relational databases on structured data like RDF's. Our experimental results show that we have indeed built a simple system, which by leveraging advantages of Hadoop & Mapreduce technologies performs much better in speed/scale over traditional file indexing systems like solr/lucene despite our choice of open text data which is a harder subset problem among QA systems.

VI. Future Work

The scope of this project is immense and given enough time, we can come with a more intelligent application that can still be used in real-time. One thing missing from our project is a user interface. A larger knowledge base deployed on a cluster will significantly cut down response time.

As of now we are posting grammatically correct questions with correctly spelled words. Checking grammatical and spelling correctness can easily be incorporated in Query processing and will make the application more user friendly. Another useful addition would be utilizing word2vec to handle tenses, synonyms and similarly used words. This would help us detect answers when they contain synonyms or plurals of the query words. We also need to move from a rule based answer type detection heuristic to a more data driven approach.

All modern Question Answering systems use a combination of information retrieval and knowledge based methods. Our system employs only Information Retrieval methods. This was chosen as it increases the usability of our system to openly available data on the web. Domain specific knowledge bases would be useful for answering questions which require in-depth knowledge of any specialized field. Like highly complex questions in medicine, math and science.

We would also like to handle some descriptive questions like "how" and "explain" type questions. Another good possibility to explore would be breaking down complex questions in many sub questions. Then answer each question individually leading up to a final answer. Additional handling of pronouns will help us answer a sequence of questions on the same topic so the continuous questions like "Who is Obama?" followed by the question "Who is his wife?" will be understood and answerable.

Using Pig, we were able to retrieve only the matching documents/articles for each query. This can further be refined by additional MapReduce or spark jobs to retrieve the exact passage/sentence which will answer the query.

Also, The existing system is run as a batch processing system. Searcher. Pig script is called manually after the completion of the MapReduce programs. HBase table is created and loaded separately into HDFS. Going further, we can integrate all these separate distributed processing jobs into a synchronized dataflow script using Oozie.

To speed up data processing, we can use an in-memory data processing framework like Apache Spark instead of Hadoop where intermediate-results are stored back in disk after each MapReduce task.

Extending the distributed system capabilities will give us a real time question answering system where users can ask questions and get immediate answers.

Incorporating all these functionalities will help us develop an expert system people can have a conversation with. Everything discussed so far is achievable with current

technology and we hope to continue working on this after the completion of the Real-time Big Data Analytics course.

VII. Conclusion

We have successfully built a text based Question Answering system on top of Hadoop. We have limited the scope of the questions to prioritize focus over the overall system architecture. The resulting system uses the Wikipedia data dump, the WikiNews data dump and Reuters News Corpus to answer a limited set natural language questions. The choice of a distributed system architecture in our case where the data corpus was huge(40GB), and there were multiple dataprocessing steps involved significantly sped up the processing time compared to a system which processes documents sequentially. Our QA system had many modules which were trivially partitionable and recombinable and by bringing in distributed approaches in these modules, we achieved an almost real time system with low latency execution. With improvements suggested in the previous section, we can suppose that Distributed technologies like Hadoop, Map reduce hints at a possibility of bringing a complex process intensive system to real time efficiencies.

Acknowledgment

We would like to thank Prof. Suzanne McIntosh for giving us the opportunity to work on this project. It was a great learning experience integrating distributed computing technologies with NLP and being able to extract relevant information from a large data set.

This project would not have been possible with freely available research papers, open source software and teaching material. We utilized a wide array resources to research and implement this project. We would like to thank all open source contributors who made this project possible.

References

[1] T. White. Hadoop: The Definitive Guide. O'Reilly Media Inc., Sebastopol, CA, May 2012.

[2] A. Gates. Programming Pig. O'Reilly Media Inc.,Sebastopol, CA, October 2011.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In proceedings of 6th Symposium on Operating Systems Design and Implemenation, 2004.

[4] S. Ghemawat, H. Gobioff, S. T. Leung. The Google File System. In Proceedings of the nineteenth ACM Symposium on Operating Systems Principles – SOSP '03, 2003.

[5] Adel Tahri and Okba Tibermacine: DB-Pedia Based Factoid Question Answering System

[6] Antoine Bordes, Jason Weston and Nicolas Usunier: Open Question Answering with Weakly Supervised Embedding Models

[7] Ben Hixon, Peter Clark and Hannaneh Hajishirzi: Learning Knowledge Graphs for Question Answering through Conversational Dialog.

[8] http://spark-public.s3.amazonaws.com/nlp/slides/qa.pdf.

[9] http://www.nltk.org/book/ch07.html

[10] http://dbpedia.org/page/Michelle_Obama.

[11] Lopez, Vanessa, et al. "Is question answering fit for the semantic web? a survey." Semantic Web 2.2 (2011): 125-155.

[12] Damljanovic, Danica, Milan Agatonovic, and Hamish Cunningham. "FREyA: An interactive way of querying Linked Data using natural language." The Semantic Web: ESWC 2011 Workshops. Springer Berlin Heidelberg, 2012.

[13] Andrenucci, Andrea, and Eriks Sneiders. "Automated question answering: Review of the main approaches." null. IEEE, 2005.

[14] Lin, Jimmy, and Boris Katz. "Question answering techniques for the World Wide Web." EACL-2003 Tutorial (2003).

[15] Liu, Chang, et al. "Hadoopsparql: a hadoop-based engine for multiple sparql query answering." The Semantic Web: ESWC 2012 Satellite Events. Springer Berlin Heidelberg, 2012. 474-479.

[16] Hirschman, Lynette, and Robert Gaizauskas. "Natural language question answering: the view from here." natural language engineering 7.04 (2001): 275-300.

[17] Katz, Boris, et al. "Omnibase: Uniform access to heterogeneous data for question answering." Natural Language Processing and Information Systems. Springer Berlin Heidelberg, 2002. 230-234.

[18] Bordes, Antoine, et al. "Large-scale simple question answering with memory networks." arXiv preprint arXiv:1506.02075 (2015).

[19] Zettlemoyer, Luke S., and Michael Collins. "Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars." arXiv preprint arXiv:1207.1420 (2012).

[20] Cooper, Richard J., and Stefan M. Rüger. "A Simple Question Answering System." TREC. 2000.

[21] Yahya, Mohamed, et al. "Natural language questions for the web of data." Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics, 2012.

[22] Prager, John, et al. "Question answering by predictive annotation." Advances in Open Domain Question Answering. Springer Netherlands, 2006. 307-347.

[23] Radev, Dragomir R., John Prager, and Valerie Samn. "Ranking suspected answers to natural language questions using predictive annotation." Proceedings of the sixth conference on Applied natural language processing. Association for Computational Linguistics, 2000.

[24] Kwok, Cody, Oren Etzioni, and Daniel S. Weld. "Scaling question answering to the web." ACM Transactions on Information Systems (TOIS) 19.3 (2001): 242-262.

[25] Croft, W. Bruce, Donald Metzler, and Trevor Strohman. Search engines: Information retrieval in practice. Reading: Addison-Wesley, 2010.

[26] Weston, Jason, et al. "Towards AI-complete question answering: a set of prerequisite toy tasks." arXiv preprint arXiv:1502.05698 (2015).

[27] Wang, Chong, et al. "Panto: A portable natural language interface to ontologies." The Semantic Web: Research and Applications. Springer Berlin Heidelberg, 2007. 473-487.

[28] Schätzle, Alexander, et al. "PigSPARQL: A SPARQL Query Processing Baseline for Big Data." International Semantic Web Conference (Posters & Demos). 2013.

[29] Unger, Christina, and Philipp Cimiano. "Pythia: Compositional meaning construction for ontology-based question answering on the semantic web." Natural Language Processing and Information Systems. Springer Berlin Heidelberg, 2011. 153-160.