International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 15 (2018) pp. 11707-11714
© Research India Publications. http://www.ripublication.com

An Approach to Semantic Search using Domain Ontology and GraphDB.

Vinayak Prasad Ramavath
Department of Electronics and Communication Engineering, NIT Warangal, India.

Anjali Sudhir Moharir
Department of Electronics and Communication Engineering, NIT Warangal, India.

Abstract

Various search engines are used in every field throughout the world every day. They are the fastest way to gather data in a short time. It is clear that the quicker and more efficient the results, the more helpful it is to the end users. This in turn, increases the capacity to push forward science and technology, architecture, medicine, and many other fields. One efficient way of retrieving optimal data from a search is by using the semantic search technique. Semantic search is a methodology used to query data, with emphasis on the intent behind the query, instead of only the words contained within it. While this approach is being adopted and implemented by some organizations, the architectures for indexing and querying documents and data differ greatly. Some major difficulties are that the search experience is dependent on a number of elements. These include a query language processor strong enough to handle billions of different queries, a user-friendly interface, result ranking. It also requires the use of appropriate data structures (graph, document-based) to store data. Semantic search is preferred over keyword-based searched as it works towards understanding the same questions posed multiple ways. It accounts for language styles and textures, and reduces the number of false-equivalence results. The purpose of this work is to create an architecture that can use the aims of semantic search to both store data in the form of documents, and query them to attract optimal results. Used properly, this architecture can shape the way searches are carried out, resulting in efficient and optimized retrieval of data.

Keywords: Semantic Search, Information Retrieval, Natural Language Processing, Entity Extraction.

INTRODUCTION

Search Engines are one of the most popular tools for access to Internet and data that people use on a daily basis. These Web Search Engines return billions of responses to billions of different queries every day. Most modern search engines aim to go beyond retrieving relevant documents. In order to satisfy the needs of the common user, they try to understand the user's intent in order to provide the most relevant results to user's query. The most crucial step towards reaching this goal is identifying the entity and intent behind the user's specific query. This helps in returning semantically accurate results.

Online search in many sites remains very much keyword-based to this day. This means whenever you enter words describing what you are looking for, the retrieved results will contain the search terms you used exactly as they are. With keyboard based search, if you type in "copy machine", you will see pages having the words "copy" and "machine" on them, but not a "multi-purpose printer". Although a multipurpose printer might be the thing you were actually looking for. Keyword-based search recognizes the form of the words, and ignores their meaning. From a language based perspective, this is not the optimal way of retrieving data. Users are expected to have full knowledge of the terminologies of their domain of study, which is not always possible. A remedy for such Search Engine Fatigue comes in the form of Semantic Search. In contrast to keyword-based search, the goal of semantic search is to understand the intent behind a user's query and find information based not just on the presence of the words, but also on their meaning.

Natural Language Processing (NLP) is broadly defined as the automatic manipulation of natural language, like speech and text, by software. As humans, we can attempt to understand various language styles and textures, provided we know the basic language being spoken. These are features of our world that our brain processes instantly. For example, our brain knows that the sentences, 'I need a place to work out of.' and 'I need a place out of which to work.' We can do it with such ease, that we end up taking this ability for granted. However, in the world of software, this is a much bigger project. The study of natural language processing grew out of the field of linguistics and was motivated with the rise of computers. It has been around for more than 50 years. NLP basically sits at the intersection of computer science, artificial intelligence and computational linguistics. In this paper, we see a software architecture for not only understanding simple sentences in multiple forms, but also using these sentences as queries to retrieve data.

The remainder of the paper is organized as follows: Prior Art Section provides background knowledge on Semantic Search, which form the foundation of the framework. Ontology Graph and Domain Model Section discusses the graphs involved in data storage and creation of domain ontology. Workflow

Management System Section describes the workflow management, advantages of the Microservice based Architecture and the message broker Kafka which is used for communication between the microservices. Indexing and Query Processing Section shows the implemented system using the indexing and query processing pipelines. Results Section displays the results and the last section before the Acknowledgement provides the conclusions along with future perspectives.

PRIOR ART

There has been some prior art in the field of semantic search. For example, Google has come a long way in making its search semantic. A researcher can frame a question in many ways to get nearly the same documents as search results. Individual contributors, such as Thomas Lukasiewicz (University of Oxford, UK) dedicate their time to understanding this querying process and optimizing it. Researchers, Qazi Mudassar Ilyas, Yang Zong Kai and Muhammad Adeel Talib (Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, China) came up with an architecture to serve this same purpose. Over time, one thing has become clear. We cannot rely on keyword-based searches anymore. As David Amerland said, “I am a firm believer that knowledge is power but only if it leads to comprehension.”

Since 1990, many researchers and innovative people have devoted time, energy, and resources to creating useful search engines. This is because we know that the faster we get answers to our questions, the quicker we can push any and all fields of research and development forward. The faster and more accurate our results, the easier it is for students to learn, write a paper, or share their thoughts. With each step the search engines take forward, we can make it more comfortable and convenient for people to get their queries answered.

After considering the prior art, we have come up with the semantic search architecture discussed in this paper. We believe this architecture of query-processing dominates over other architectures in terms of the appropriateness of results obtained. With proper training, the model can potentially understand all dialects of English spoken throughout the world.

Ontology Graph and Domain Model

Ontology is defined as an explicit formal specification of the terms in the domain and relations among them. It is primarily used to capture knowledge of any particular domain. The major advantage of use of an ontology is that it will provide a globally unique identifier for all concepts. Advantages of using an ontology is that it:

- Helps avoid ambiguity of terms
- Helps share common understanding of the structure of information among the users
- Enables reuse
- Analyzes the domain knowledge.
- Enables the merging of already existing knowledge, thereby expanding it further.

To develop an ontology for any domain, a set of questions is formulated. These are the questions that the envisioned knowledge-based agent should be able to answer. This is also known as schema. Based on these questions, some of the concepts, sub concepts, relationships, features and instances that are defined as the part of the ontology can be identified. These questions are formed, taking into consideration the different ways people all around the world would ask that particular question. In any domain, controlled vocabulary of words from that domain is taken for knowledge representation. When representing knowledge for the domain, controlled vocabulary helps avoid the use of duplicate, arbitrary or perplex words which would lead to inconsistent knowledge. It also prevents misspelling of words. Ontology is implemented in almost all fields of study (medical, space, food, aviation, commerce, linguistics, agriculture etc.).

Here, we picked out a domain (Java) as we had expertise in Java and created an ontology for the “Java” domain. We can easily replace the domain by collecting enough information about it from domain experts and creating an ontology out of it. We stored the entire ontology using a graph and we used neo4j for this purpose. Neo4j is a graph database management system developed by neo4j, Inc. Neo4j has an easy-to-learn and easy-to-use query language and a web based, graphical interface which allows users to easily browse and explore the graph. Also, it is fast for querying and scales very well to handle larger datasets.

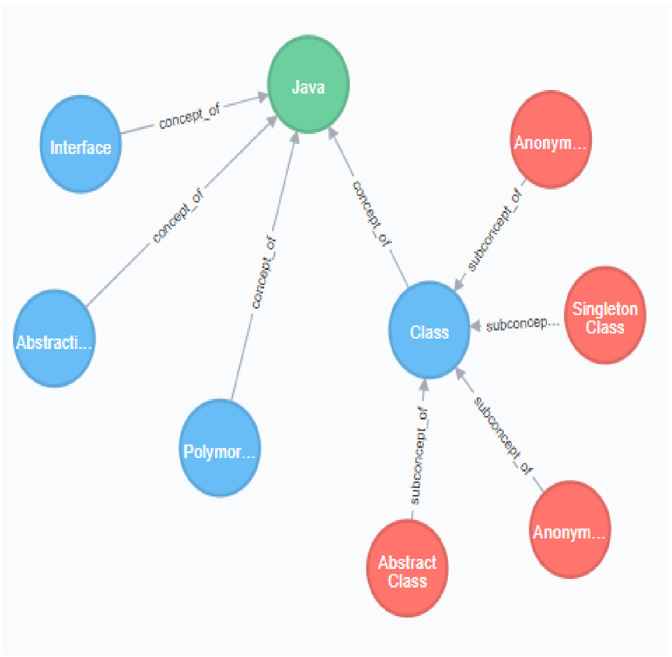


Figure-1: Sample Ontology for the Domain “Java”

Here is a sample screenshot from our ontology in the chosen domain ‘Java’. We can see that ‘class’, ‘interface’, ‘polymorphism’, etc. are the concepts of ‘Java’. Also, ‘abstract class’ ‘anonymous class’ and ‘inner class’ are sub-concepts of ‘class’ as indicated by the relationship between them. In this way, we modelled the entire concept graph based on our knowledge in Java.

Similar to the concept graph, there is an intent graph where most of the words anticipated from the search query are classified into 6 intents and put under their respective intent nodes, i.e. Basic, Tutorial, Example, Getting Started, Complete Reference and Troubleshoot. Words like define, explain, why, list, what, etc. indicate that the user is looking to learn basics of a concept in java. So, all such words are included under the Basic node in the intent graph. Words like code, solution, example, etc. are kept under Example node. Here we divided the user query into tokens using NLP techniques. We search for each token in the intent graph and figure out the intent of the user in order to achieve better results.

This provides a huge advantage, as we can then display results truly based on the intent of the users’ queries. For example, a query with ‘Basic’ intent will give results containing basic definitions and theory. A query with the intent as ‘Tutorial’ will give more videos on how to accomplish the task at hand. It will also give step by step instructions and algorithms for completing the task.

Table-1: List of Six Intents and their Child Nodes in the Intent Graph

Node ID (INT)	Intent (Parent Node)	Example Child Nodes
1	Basic	What, Describe, Explain, Define, Fundamental, Elementary, List.
2	Tutorial	How, Tutorial, Learn, Course.
3	Example	Example, Code, Solution.
4	Getting Started	Install, Download, Get Started
5	Complete Reference	Documentation, Material
6	Troubleshoot	Troubleshoot, Rectify, Solve

WORKFLOW MANAGEMENT SYSTEM

A. Microservice Based Architecture

This paper proposes an architecture for implementing the Semantic Search using microservices. Microservices, also known as microservice based architecture is an architectural style that structures an application as a collection of loosely coupled services, enabling the continuous deployment of large/complex applications. One of the greatest advantages of this architectural style is that we make sure each microservice is performing only one operation. This is known as the Single Responsibility Principle. Each of these services can easily be deployed and then redeployed independently without compromising the integrity of the application. This is very effective during debugging, and during testing and verification.

B. Message Transfer System

We used the message broker Apache Kafka for the communication between the microservices. Kafka is preferred as it does not require large hardware, and is capable of handling high-velocity and high-volume data. Kafka is able to support message throughput of thousands of messages per second, perfect for search engines. It has the capability to handle these messages with a very **low latency (**in the **range** of milliseconds). We use different Kafka topics to handle each transaction between two services. For example, messages going from Crawler to Parser (refer architecture below) have a topic of ‘crawled’. Messages transferred from Parser to Indexer have a topic of ‘parsed’. This allows the message transfer to be simple and concise.

Indexing and Query Processing

This paper section describes our work in the semantic search engine application. The implementation is divided into two pipelines which performs the following tasks.

- Indexing huge number of documents related to our domain and storing them in the graph database
- Implementing a query processing pipeline which takes in a user query as an input and fetches the related documents using the concept and intent, which are extracted using various NLP methods.

This diagram shows the NLP pipeline, used to better understand the intent of the user’s query. It also shows the indexing pipeline, used to parse documents and categorize them according to their metadata, contents, etc. These two pipelines, working simultaneously, give us semantically optimized search results.

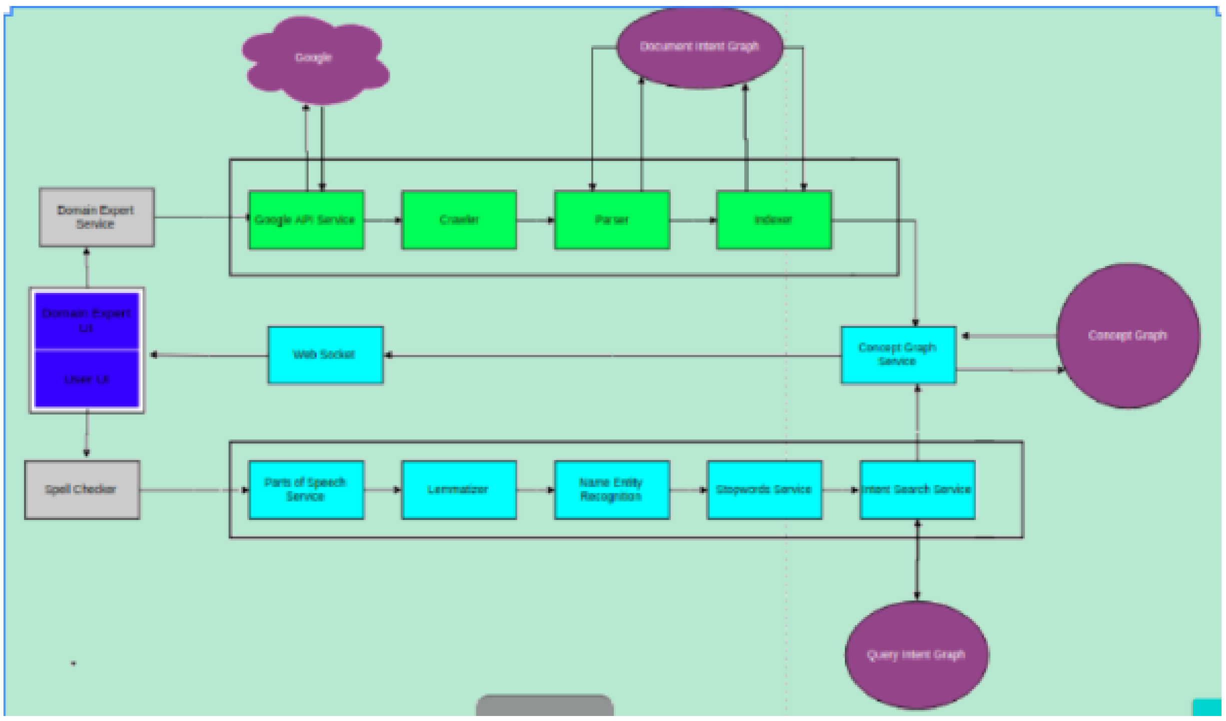


Figure-2: System Architecture Indicating Indexing Pipeline, Query Processing Pipeline and Knowledge Graphs

A. Indexing Pipeline

Indexing of web documents is divided into three components: Crawler, Parser and Indexer. The Crawler, also called the Spider, traverses the web collecting information. It stores the information into a huge repository after being compressed. The Parser follows hyperlinks given by the crawler across the web collecting information in the form of HTML web pages. The indexing module takes these pages and assigns an index to each of them based on their intents (explained in detail later.) It also pushes the pages into the database according to their indices. Every Web document has an associated ID number called document identifier, which is assigned whenever a new URL is parsed out of a web page.

Crawler: Crawler starts with the list of seed URLs as the initial input. These URLs can be obtained from any search engine API available. Here, Google's Custom Search API was used for this purpose. All we need to provide Google's API is the domain and a concept name and the API returns a bunch of URLs for the crawler to start its work. Crawler traverses the web to download the corresponding webpage in XML format, using the java library Jsoup. It sends the downloaded XML document as input to the Parser.

Parser: The XML document obtained from crawler and the webpage is now stored in the form of keywords present in them. These keywords are fetched from the ontology created for this particular domain. These keywords alone are not sufficient for retrieving information about the webpage. The presence of keyword in various HTML tags of web documents should be considered for indexing the web pages. The proposed parsing technique has considered the presence of keywords in various HTML tags of web documents such as head, title, body and link. A certain weight is assigned to each of these tags and the XML document is traversed for these keywords. Each keyword is also given a value. The score of each keyword is determined based on where it occurred in the HTML page and the number of times it occurred in that place. A map containing all the keywords as keys and their respective scores as values is sent as input to the indexer service.

$$\mathbf{KS}=\sum_{j=1}^{ntag}(\boldsymbol{TS}*\boldsymbol{K}*\boldsymbol{N})$$

Where,

KS=Keyword Score

TS=Tag Score

K=Individual Keyword Score

N=Frequency of Keyword in tag

ntag =Number of tags

Indexer: Indexing refers to the organization of data according to a specific schema or plan, thereby making information more presentable and accessible. In this service we calculate the total score of the URL (webpage) for each intent.

The Indexer takes the concept from the Crawler and the Keyword Score for each keyword form the Parser. From here, it goes to the intent graph and finds the parent node (true intent) of each keyword. The sum of all KS's under each parent node is the total intent score for that intent. The indexed URL is now attached to its respective concept in the ontology (concept) graph along with its scores.

E.g. the final scores can be (Basic: 140, Tutorial: 423, Troubleshooting: 411, etc.). This indicates that the document is primarily a tutorial, but can also be shown as a result for a troubleshooting intent, towards the end of the list of results.

B. Query Processing Pipeline

Spell Checker: The first block of the NLP pipeline is the spellchecker. It checks the spelling of each word of the user query, by checking the words against the dictionary it knows. This is a RESTful spellcheck web service provider created using Hunspell dictionaries. The database that this service uses can be easily modified by the developer in order to add or subtract words. Once the query has been checked for spelling errors and modified (if needed), it is sent to the PoS Tagger.

Part-Of-Speech Tagger: The Part-Of-Speech Tagger (PoS Tagger) is a piece of software that reads text in some language (here, English) and assigns parts of speech to each word (noun, verb, adjective, etc.). Part-of-Speech tagging is not as simple as having a list of words and their parts of speech, because some words can represent more than one part of speech at different times (e.g. Alert can be both a noun and verb). Also, some parts of speech are complex and unspoken. The PoS Tagger takes care of all these scenarios, by referring to its dictionary. This dictionary can be modified according to the users' needs, if every this becomes necessary. The PoS Tagger then sends the original query along with all the parts of speeches to the Lemmatizer.

Lemmatizer: The goal of lemmatization is to reduce derivationally related forms of a word to a common base form. Unlike stemming, which refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes, lemmatization does things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as lemma. If "saw" is passed through lemmatizer, it would return either "see" or "saw" based on whether the use of token was a verb or a noun. The part of speech of the token is already found when passed through the Part-of-Speech tagger. The lemmatizer sends the lemmatized query to the NER service for further processing.

NER Service: The process of finding names, people, places, and other entities, from a given text is known as Named Entity Recognition. Here, the domain being Java, we used the NER Service to separate out tokens with words related to Java. We identify this as the 'concept' of the query. This is done using the opennlp libraries TokenNameFinderModel and NameFinderME.

The tokens coming from the NER Service can be either single word or multi-word, formed based on the rules extracted mainly from the trained data set. Data is fed to the service in the format:

E.g. Tell me about <START: keyword> Abstract Class <END>

Here, "\t" is used to denote a tab in the query, and CR is used to signify a new line.

Building the NER Service can be tricky. For our purposes, we need the NER Service to identify any and all intents related to the domain 'Java'. This means, there need to be enough sample queries (data points) for it to understand multiple intents, sometimes more than one in one user query. The NER Service does not only learn what the concept can be, it also learns what it should not be. For example, in the example given above, the NER Service learns that 'Abstract Class' can be a query, and all that 'Tell me about' is not a concept. Also, we need the micro-service to understand that queries can be of many lengths. Until we give it a surplus of data points with three-letter concepts, it will not understand that three letter concepts can exist. As a results, it will try to find a one or two letter concept, thus leading to the wrong results. A very large number of data points must be given to get accurate results. Once this is taken care of, our query processing pipeline can extract the concept of any query.

When ample data is entered into the data set, taking into considerations the various ways the same query can be expressed, this service learns to identify the 'concept' of the query. (Here, 'Abstract Class'). The NER Service then sends the query along with the identified concept to the Stop Words Eliminator.

Stop Words Eliminator: Some extremely common words (a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, to, was, were, will, etc.) are extracted from the query. This is because they are clearly not the concept or intent of the query. These words are called stop words. Stop word elimination is a simple but important aspect of many text mining applications as it has the following advantages:

- ➢ Reduces memory overhead (since words in consideration are eliminated)
- ➢ Reduces noise and false positives (since the focus is on the important terms)
- ➢ Can potentially improve power of prediction (dependent on application).

In our service we used an array obtained from various NLP sources containing the stop words. Like the spellchecker, the stop words database can be modified easily by the developer as and when needed.

Intent Search Service: This service takes the tokens from the SWE service and searches for these tokens in the intent graph, present in Neo4j graph DB. Once a token is found in the intent graph, its parent is identified as the intent of the user's query.

When the user types in the query "Tell me something about interfaces", it goes through the NLP pipeline. Here, NER and Intent Search Service identify the concept and intent of the query respectively as 'interface' and 'Basic'. The concept graph is searched for the concept of the query, and the required URLs are returned. The documents are displayed in descending order of the scores given by the indexer for that particular intent.

IMPLEMENTATION AND RESULTS

A. Theoretical Implementation and Expected Results

The figure below shows the processing of the query "Tell me something about interface". This is taken as an example to show how the query is altered in each service of the query processing pipeline. The inputs and outputs of each block (micro-service) are given on the arrows leading in and out of the micro-service respectively.

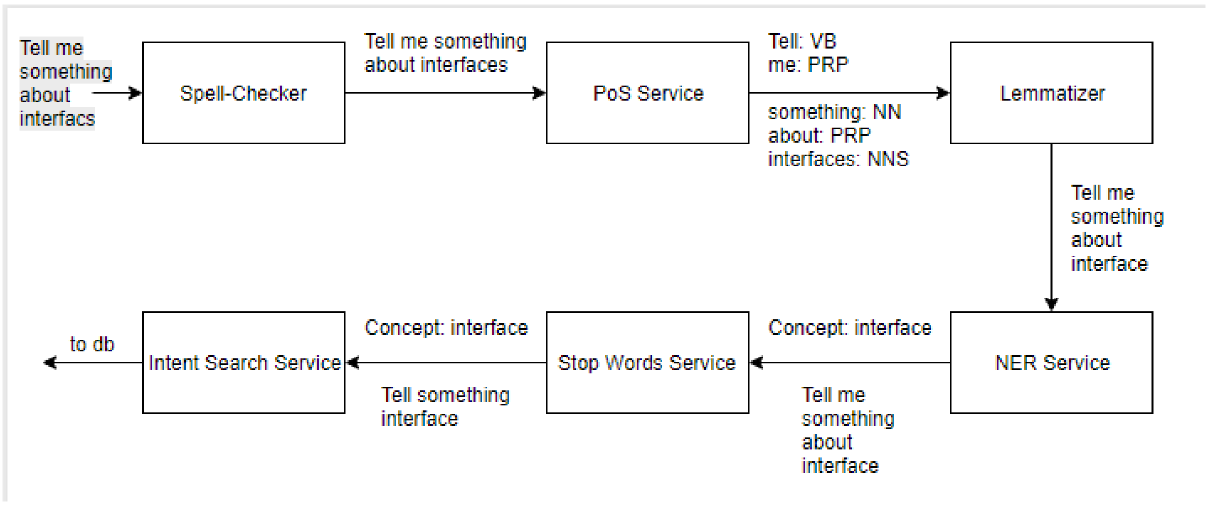
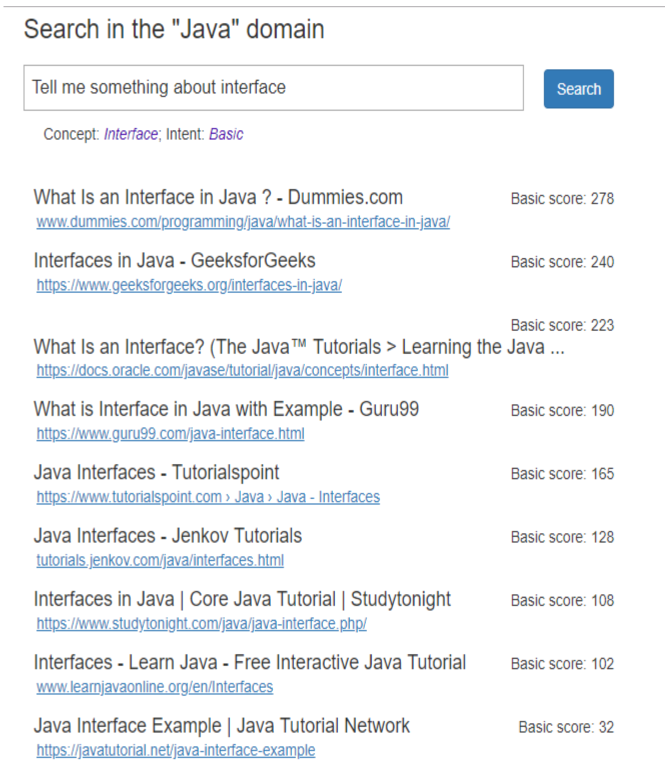


Figure 3: Theoretical Implementation of Query Processing Pipeline.

B. System Implementation and Experimental Results

When the query "Tell me something about interface" is entered, the NLP pipeline identifies the concept as "interface" and the intent of the user's query as "Basic. The documents (URLs) are fetched from the graph database and displayed here ranked according to their Basic scores. The documents relevant to the users' queries are returned accurately and efficiently.

The UI used can be changed according to the developers and users' styles and preferences. We created a UI keeping in mind that simplicity and ease of understand are key. This UI shows the URLs to the documents relevant to the query along with their titles and scores of its parent intent. This is shown in the figure below.


Search in the "Java" domain

Tell me something about interface

Concept: *Interface*; Intent: *Basic*

What Is an Interface in Java ? - Dummies.com
www.dummies.com/programming/java/what-is-an-interface-in-java/
Basic score: 278

Interfaces in Java - GeeksforGeeks
https://www.geeksforgeeks.org/interfaces-in-java/
Basic score: 240

What Is an Interface? (The Java™ Tutorials > Learning the Java ...
https://docs.oracle.com/javase/tutorial/java/concepts/interface.html
Basic score: 223

What is Interface in Java with Example - Guru99
https://www.guru99.com/java-interface.html
Basic score: 190

Java Interfaces - Tutorialspoint
https://www.tutorialspoint.com › Java › Java - Interfaces
Basic score: 165

Java Interfaces - Jenkov Tutorials
tutorials.jenkov.com/java/interfaces.html
Basic score: 128

Interfaces in Java | Core Java Tutorial | Studytonight
https://www.studytonight.com/java/java-interface.php/
Basic score: 108

Interfaces - Learn Java - Free Interactive Java Tutorial
www.learnjavaonline.org/en/Interfaces
Basic score: 102

Java Interface Example | Java Tutorial Network
https://javatutorial.net/java-interface-example
Basic score: 32

Figure-3: Obtained Search Results for the Query "Tell me something about interfaces"

The order of these results can be altered according to the needs of the organization. This can easily be done by changing the structure of the intent graph.

We have implemented this Semantic Search Engine using Java in a linux environment. Here, the testing was done first on localhost. The entire project was then deployed on AWS with proper results. Query Execution time is affected by the length of the query. The average execution time was 1.12 seconds on localhost for one-word concept queries and 1.6 seconds for two-word concept queries. When we executed the same on AWS, it was 0.74 seconds and 0.86 seconds respectively. However, more powerful servers are used by industry-scaled systems. This will reduce the time required to carry out the query significantly. Performance testing was done using JMeter. Load testing revealed a 9,998/10,000 hit success rate. We see that the architecture is stable and returns results efficiently.

CONCLUSION AND FUTURE WORK

The need for fast and accurate search results is becoming more and more important as the size of the Web has continually kept growing. The retrieval of relevant data is more complex as more people and thus more ontologies come into play each day. As shown above, this architecture of querying for properly indexed documents will have a significant impact on how we search for data on the Web.

In the future, this architecture can be best used in businesses, schools, hospitals etc. with large data sets and documents to handle and organize. This application can be used with any domain just by creating an ontology from the data provided by the domain experts and modelling the intent graph according to that particular domain.

The algorithm for indexing documents can be further enhanced. This can be done by taking into account how many users have previously searched for a document, the age of the domain, the speed of loading the documents and many other such factors.

Machine learning techniques can be added in to enhance querying. Thus, the more the application is used, the better it will be at retrieving relevant documents. Also, the more input we add to our training data set, the more accurate the results will be. This can be done by conducting and considering speech surveys to understand how people from different parts of the world best communicate.

Overall, this architecture produces efficient and accurate results, and the project can be taken further in great leaps.

ACKNOWLEDGEMENTS

The authors wish to thank their colleagues at National Institute of Information Technology, Bangalore, India for their participation, insights, and patience.

REFERENCES

[1] P. Hema Priya, R. Ranga Raj (2013, Dec 13). An Improved Search Engine by Semantic Web Using Ontology. International Journal of Science and Research (IJSR), 2(12), 403-408.

[2] Sunny Lam, (2001, Feb 9).The Overview of Web Search Engines. Available: https://cs.uwaterloo.ca/~tozsu/courses/cs748t/surveys/sunny.pdf

[3] Er. Sugandha Sharma, Er. Seema Rani (2014, June 6).Survey on E-mail Spam Detection Using NLP. International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), 4(5) 55-61.

[4] Bikel, D., & Zitouni, I. (2012). Multilingual natural language processing applications: from theory to practice. IBM Press pp 286.

[5] Pascal Hitzler, Krzysztof Janowicz and Adila A. Krisnadhi (2015). Ontology Modelling with Domain Experts: The GeoVoCamp experience. Available: http://daselab.cs.wright.edu/pub2/2015-diversitypp-invited.pdf

[6] M. Song and Y. Wu, Handbook of Research on Text and Web Mining Technology, Hershey, PA, USA: IGI Global, 2009, pp. 228

[7] Bandyopadhyay, S. (Ed.). (2012). Emerging Applications of Natural Language Processing: Concepts and New Research: Concepts and New Research. IGI Global.

[8] De Laat, M., Lally, V., Lipponen, L., & Simons, R. J. (2007). Investigating patterns of interaction in networked learning and computer-supported collaborative learning: A role for Social Network Analysis. International Journal of Computer-Supported Collaborative Learning, 2(1), 87-103.

[9] Homa B. Hashemi, Amir Asiaee, Reiner Kraft. Query Intent Detection using Convolutional Neural Networks. Available: http://people.cs.pitt.edu/~hashemi/papers/QRUMS2016_HBHashemi.pdf

[10] The Stanford Natural Language Processing Group. (2012, Nov 15). Stanford Named Entity recognizer (NER) [Online]. Available: http://nlp.stanford.edu/software/CRF-NER.shtml#About

[11] Princeton University. (2012, Nov 10). WordNet: A Lexical database for English [Online]. Available: http://word net.princeton.edu/

[12] Louis, A. (2017). Natural Language Processing for Social Media.

[13] Liu, B. (2012). Sentiment analysis and opinion mining. Synthesis lectures on human language technologies, 5(1), 1-167.

[14] Pooja Mudgil, A. K. Sharma, Pooja Gupta. An Improved Indexing Mechanism to Index Web Documents. 5th International Conference on Computational Intelligence and Communication Networks. pp- 460-464,2013.

[15] Gupta, V., & Lehal, G. S. (2009). A survey of text mining techniques and applications. *Journal of emerging technologies in web intelligence*, *1*(1), 60-76.

[16] Kasemsap, K. (2016). Text Mining: Current Trends and Applications. Web Data Mining and the Development of Knowledge-Based Decision Support Systems, 338.

[17] Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB conference, pp 487–499

[18] Lin, J., & Dyer, C. (2010). Data-intensive text processing with MapReduce. Synthesis Lectures on Human Language Technologies, 3(1), 1-177.