UNIVERSITÀ DEGLI STUDI DI TRENTO

Project report

GRAPH-BASED KEYWORDS TRACKING FROM DIGITAL NEWS

Instructor: *Dr. Denis Paperno*

Student: *Nguyen Duc-Duy*

UNITN, January 2016

CONTENTS

SUMMARY AND ACKNOWLEDGEMENT

Graph-Based Keywords Tracking from Digital News is the project belongs to course of Computational skills for text analysis. The primary objective is collect, process and visualize trending keywords published in Digital News websites over time, based one co-occurrences of words. Indeed, a system was constructed to (1) receive data from 6 famous online newspapers in United Stated; then (2) processing text with natural language processing methods; and finally (3) visualize the result as a graph in web environment. Furthermore, the evolution of keyword's graph allow us to track the trending topics overtime.

The project would have not been finished without the knowledge and supervision, provided by Dr. Denis Paperno (CLIC lab, a unit of the Center for Mind/Brain Sciences, University of Trento). The source and tests were operated on UIT Cloud system, provided by University of Information Technology (Vietnam National University Ho Chi Minh City).

I. OBJECTIVE AND OVERALL IDEA

1. OBJECTIVE

The project focus on its primary and secondary objectives:

- **Primary objective:** construct a system that allow user to visually tracking the change of trending topic in digital news.
- **Secondary objective:** the system is able to operate in real-time. In another words, the whole process could be carried out automatically and continuously.

There are several characteristics that distinguish digital news headline – the subject of processing phrase from typical text document. For example, a headline from The New York Times in November 6, 2015: *"The Last Place on Earth With Ebola: Getting Guinea to Zero"*

- **Short expression and Compact meaning**. With a very limited total number of words, the headline transfers plenty of information, included metaphorical ideas.
- **Weak syntactic constrain**. This characteristic make most parsing effort became incapable to perform correctly.
- **Abnormal text capitalization**. The characteristic is an obstacle to part-of-speech tagging and named entities detection.

Accompany with the difficulties come from text characteristics, there are technical issues:

- **Real-time processing requirement**. The smaller interval between two times of getting data, the bigger challenge of timing for data processing. Indeed, the smaller interval of data collecting require faster data processing technique.
- **Data visualization and animation.** As a large number of nodes and edges come in and out the graph space in a short period of time, data visualization and animation technique need to be capable, quick enough to avoid delay, and slow enough so that user can be in touch with animation (graph evolution).

2. OVERALL IDEA

The primary objective focus on three main points:

- **Data collection**. Nowadays, digital news become a cheap and convenience channel to approach information about latest events happening worldwide. Compare to conventional newspapers, the ultimate advantage of digital news is that it is frequently updated, which makes up a stream of new topics stretching over time. Therefore, it is a valuable source of data for tracking trending keywords, and furthermore, trending topics.

In order to exploit this advantage, the system crawl approximately 300 headlines every hour. The data is stored in database, waiting for processing phrase.

- **Data processing.** The approach for data processing based on text co-occurrence. The approach assume that, the co-occurrence of a pair of words a document (headline) indicates that they have a relationship. There are a several questions raised along with the approach:
    - **Word as a unit of meaning.** A word could typically consider as a unit of meaning, or a node in the graph. However, there are cases that would should be understand in group, or more exact, in sequence. A typical example of this case is proper name. To solve this problem, the program provide an algorithm to join words into groups, using named-entities recognition and machine learning.
    - **Time and frequency -oriented graph building.** Merely bases on co-occurrence lead to overcrowding of the graph with a huge number of nodes and edges. To tackle this problem, created time and frequency of nodes and edges is taken into action. Specifically, after the raw graph being constructed using word co-occurrence, it be pruned in several iterations, where old and low frequency node and edge are removed, until its size satisfy thresholds.
- **Graph visualization.** As text processing produced the result as a graph which is stored in database as a buffer, these information will continuously queried and visualize in web environment.

II. SYSTEM ORGANIZATION

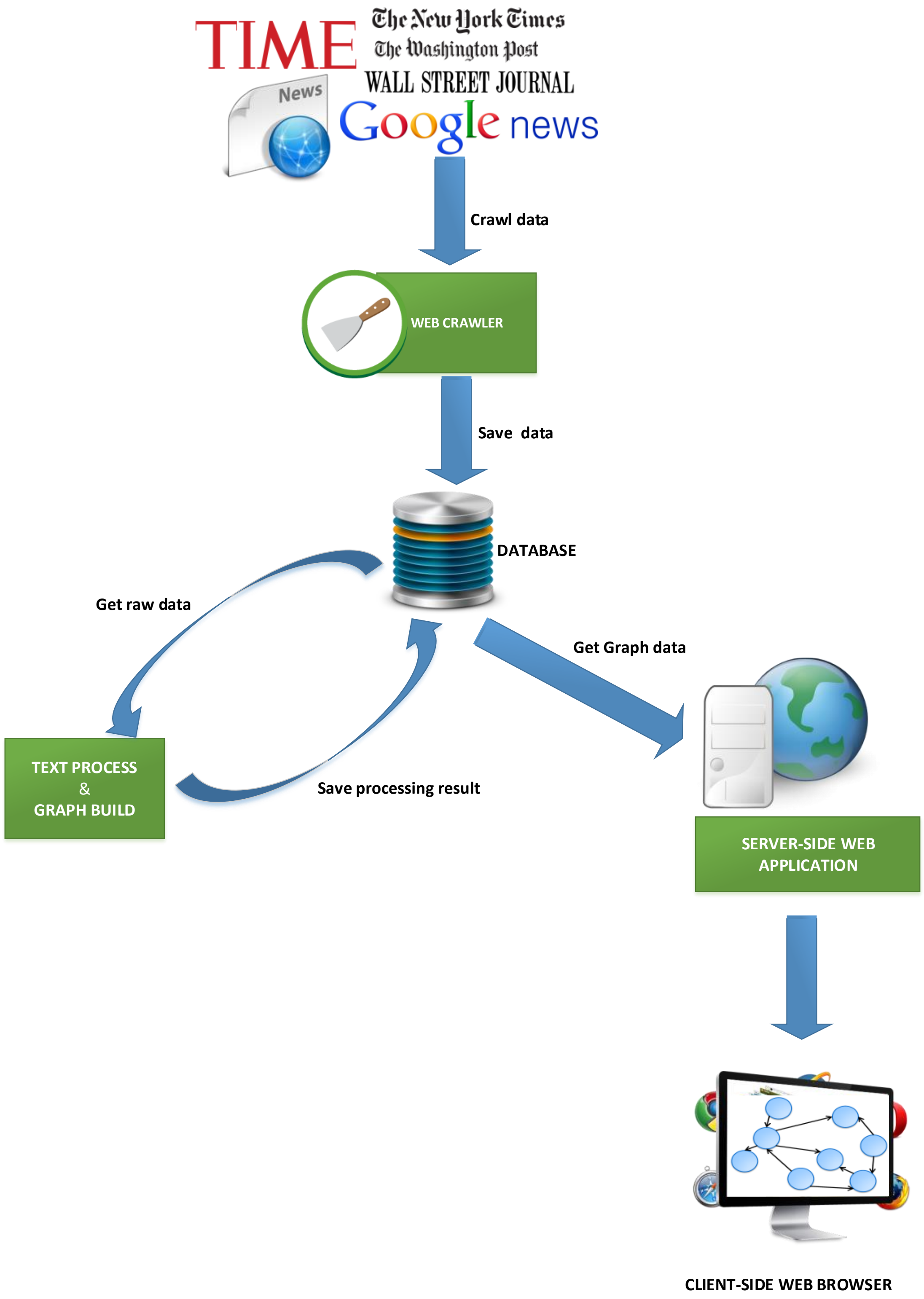

Figure 1 Organization of the system

Figure 1 illustrates the organization of the proposed system. At the beginning of the process, **Crawler** hourly download data from 5 well-known digital news in Unites States. After preprocessing, these data will be store in a **Database**. These data will then being queried and load as input for the process of **Text Process and Graph Build**, which produces the keywords graph and save it to the Database. A **Web Application** was established in Webserver to receive request from **Client-side Web Browser**. After a predefined interval, the application update new result from Database and manipulate changes to current graph and visualize the new graph in web browser interface.

There are three primary components developed in the project: Web Crawler, Text Process & Graph Build and Server-side Web Application.

1. WEB CRAWLER

The web crawler collects approximately 300 new headlines every hours from 5 popular news channels, included:

- Time (http://time.com)
- Google News (https://news.google.com/)
- Wall Street Journal Europe (http://www.wsj.com/europe)
- Washington Post (https://www.washingtonpost.com/)
- New York Time (http://www.nytimes.com/)

The data crawled are stored in the database for further usage in graph construction.

Specifically, **184954** news headings were collected continuously from September 16th to November 6 2015.

2. TEXT PROCESS & GRAPH BUILD

The phrase of Text Process and graph build operate parallel with web Crawler, showed in Figure 2.

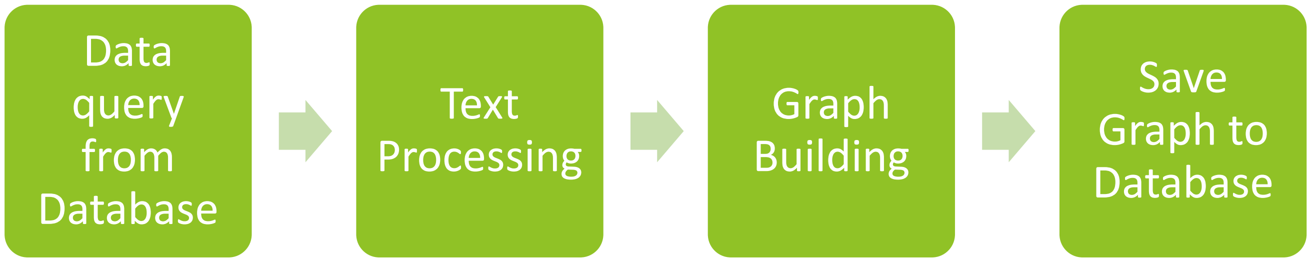

Figure 2 Text Processing & Graph Build

By default, starting from a predefined time point, the program queries all data within a time window. The time window then moves to the next position to get new data.

TEXT PROCESSING

The text corpus consists of various number of individual document, each of them contain a headline and metadata about its origin. Each of these document is processing with the following procedure:

- **Text clean up**. Due to the variety of data source, there are many differences an inconsistency in expressions. A typical example for this problem is quotation mark. While some newspaper prefer ' '(0x22 in ASCII) and " "(0x27 in ASCII) which can easily be recognized by many text processing library, the others use ‘ ’(U+2018 and U+2019 in Unicode) and “ ” (U+201C and U+201D in Unicode); hence, they need to be preprocessed beforehand.
- **Part-of-speech tagging**. Part-of-speech tagging is performed in order to support filtering preferred part-of-speech tags to enter the keyword graph; in this case, they are nouns and proper-nouns. To deal with problem of *Abnormal text capitalization*, a tagger was trained using NLTK Brown to adapt this characteristics.
- **Named-entities recognition.** The technique is applied to detect possible sequences of words that represent human names, organizations, locations, facilities. These named-entities are treated as one word and being tag as “NE” will not be eliminate during words filtering.
- **Words filtering.** Word filtering eliminate words whose part-of-speech is not in the preferred list, or not satisfy orthographic characteristics.

The outcome of text processing a corpus is a set of raw nodes (each of them represents a keyword) and raw edges (each of them represent a co-occurrence of two keywords).

Every node and edge is stored with two accompanied information:

- **Frequency**: The total number of time its keyword(s) appeared in the corpus.
- **DateAdded**: The latest time point that the node/edge appeared in the corpus.

GRAPH BUILDING

Although the result of text processing step could be considered as a graph structure, it is usually be highly populated due to merely base on keyword co-occurrence. Beside, this graph is temporally structured only, since it is built from an individual corpus, with no connection to its ancestors, or the graphs built in previous time windows. Therefore, the graph building step aims to solve the above problem, bases on nodes/edges frequency and the time that they are added to the current keyword graph.

The process of graph building consist of two actions:

- **Node merging:**
    - **Duplicate nodes merge** with *MergeDuplicate* procedure. This action aims to merge words that are orthographically similar.
    - **Join frequented bigrams** into one with *BigramNodeMerge* procedure.
- **Graph construction and prettify**:
    - **A graph structure** is constructed. As new nodes and edges are coming, the graph itself adjusted to retain the integrity.
    - **Graph prettify** with *GraphPrunning* procedure. This step aims to remove old and low frequency nodes and edges, in order to control the size of the graph. This process also the phrase where the evolution of keyword graph is carried out.

SAVE GRAPH TO DATABASE

The outcome of the processing every text corpus is a graph, under the form of two sets:

- Set of nodes;
- Set of edges.

The graph will be saved to the database as one record, accompanied with its metadata.

3. SERVER-SIDE WEB APPLICATION

Server-side web application visualizes the graph structure and its evolution over time and send these visualization to Client-side web browser.

Figure 3 show the working of the webserver. Firstly, a client computer send a request using HTTP protocol to server. This request is transferred in internet environment and forward to web server. Then, provide parameters will help webserver depart the iterations of requesting data from database. As new data arrived to webserver, it is processed and the server then send response with the result via network environment back to client computer. The process of querying new data from database, process them and send result to user is repeated until the user close the website from browser or time out occurred.

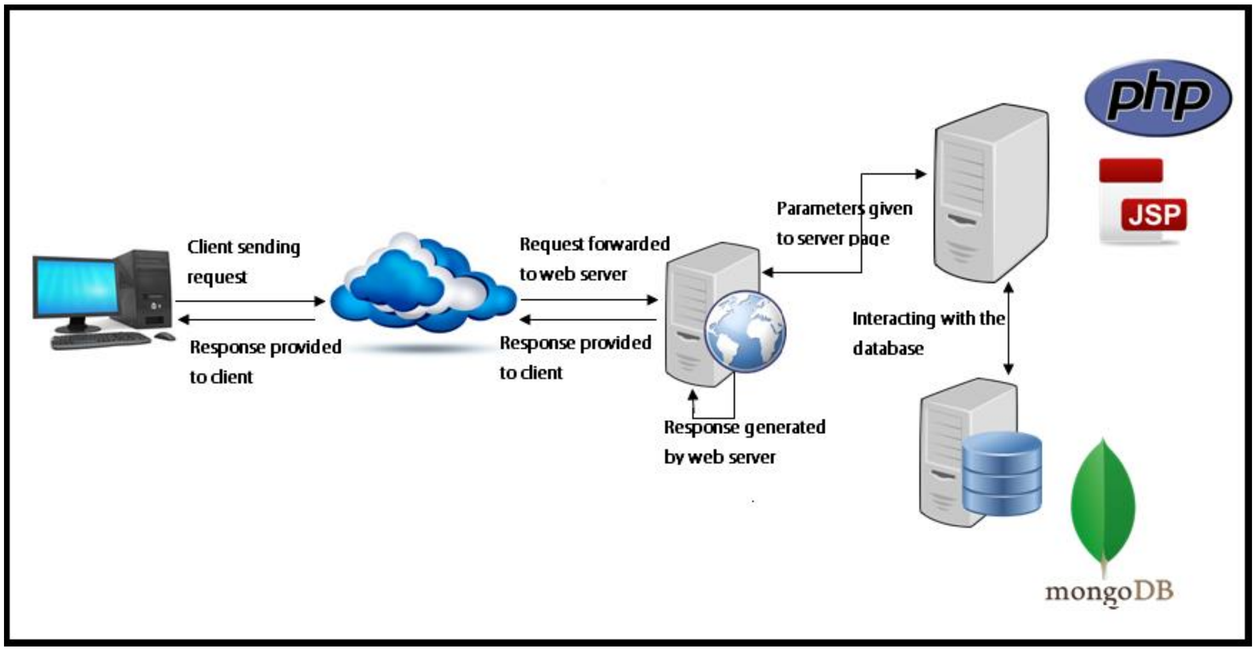
Client sending request
Response provided to client
Request forwarded to web server
Response provided to client
Parameters given to server page
Response generated by web server
Interacting with the database
php
JSP
mongoDB

Figure 3 How the web-application works

As a result, users can see the result in graphical interface, as showed in Figure 4.

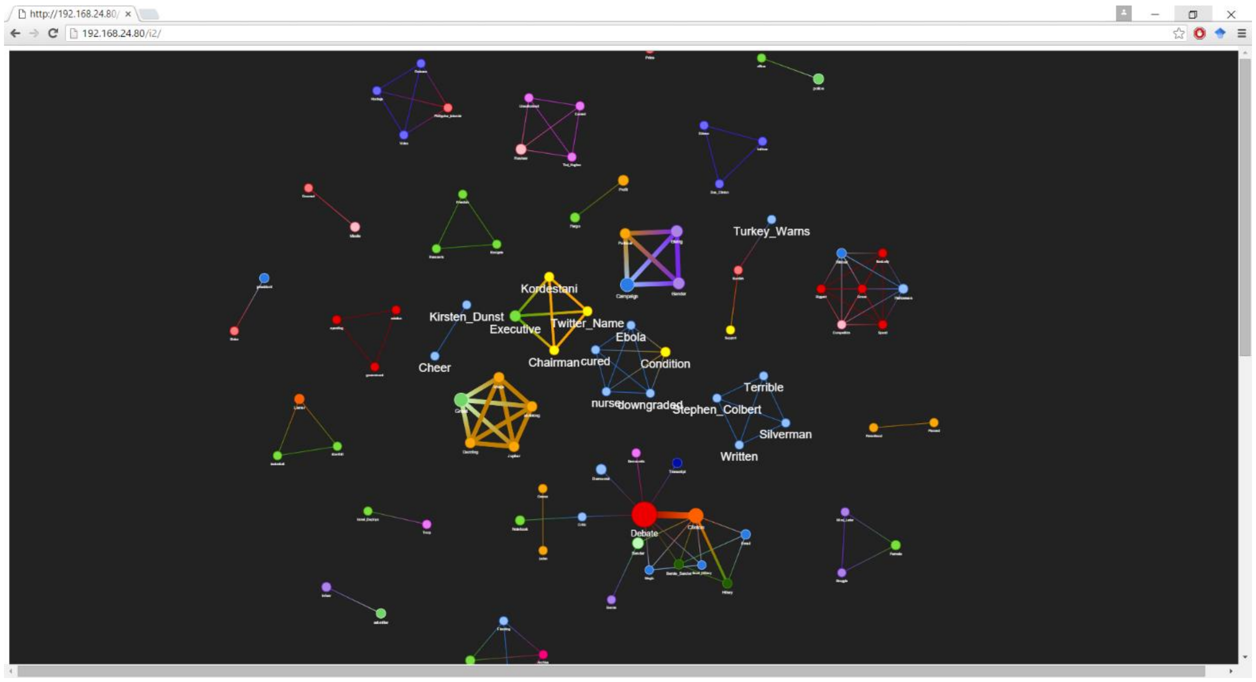

Figure 4 Web Application interface from users' browser

Color and size of nodes and edges are used to show their characteristics. Nodes and edges with high frequency have a bigger size, compare to the lower ones; while sharing the same color indicates that they have the same frequency. Besides, new nodes are labeled with the bigger size than the old ones.

From the users' perspective, the application support interactive actions, such as drag-and-drop, zoom-in, zoom-out, change edge stroke…

III. SOURCE CODE COMPONENTS

The source code of the system are deployed in webserver, whose acts as showed in Figure 5.

The code was separated into 5 modules.

1. WEBCRAWLER.ZIP

Descriptions: code for crawler which download data from new paper website. The crawler then connect to an Mongodb instance running in local host and save data to database, under default database name: “rskills” and collection “crawleddata”.

Behavior: the program call python script from command prompt environment; when the python program finish it works, the program sleep 1 hours and then start calling the script again.

Files:

- **Run.bat**: executive command file to run the crawler
- **Main.py**: python main program
- **Functions.py**: source code of 6 crawlers

Dependencies:

- Scrapy and its dependencies (http://scrapy.org/)
- PYWin32 (http://sourceforge.net/projects/pywin32/)
- Pymongo and its dependencies (https://api.mongodb.org/python/current/)

Deployment: this component could be deploy in any computer which has stable and consecutive connection to the internet, as well as MongoDB database during its operation. In case the crawler run in different machine to the one which mongod instance is running, the `HOST` and `PORT` parameters need to be reassigned accordingly.

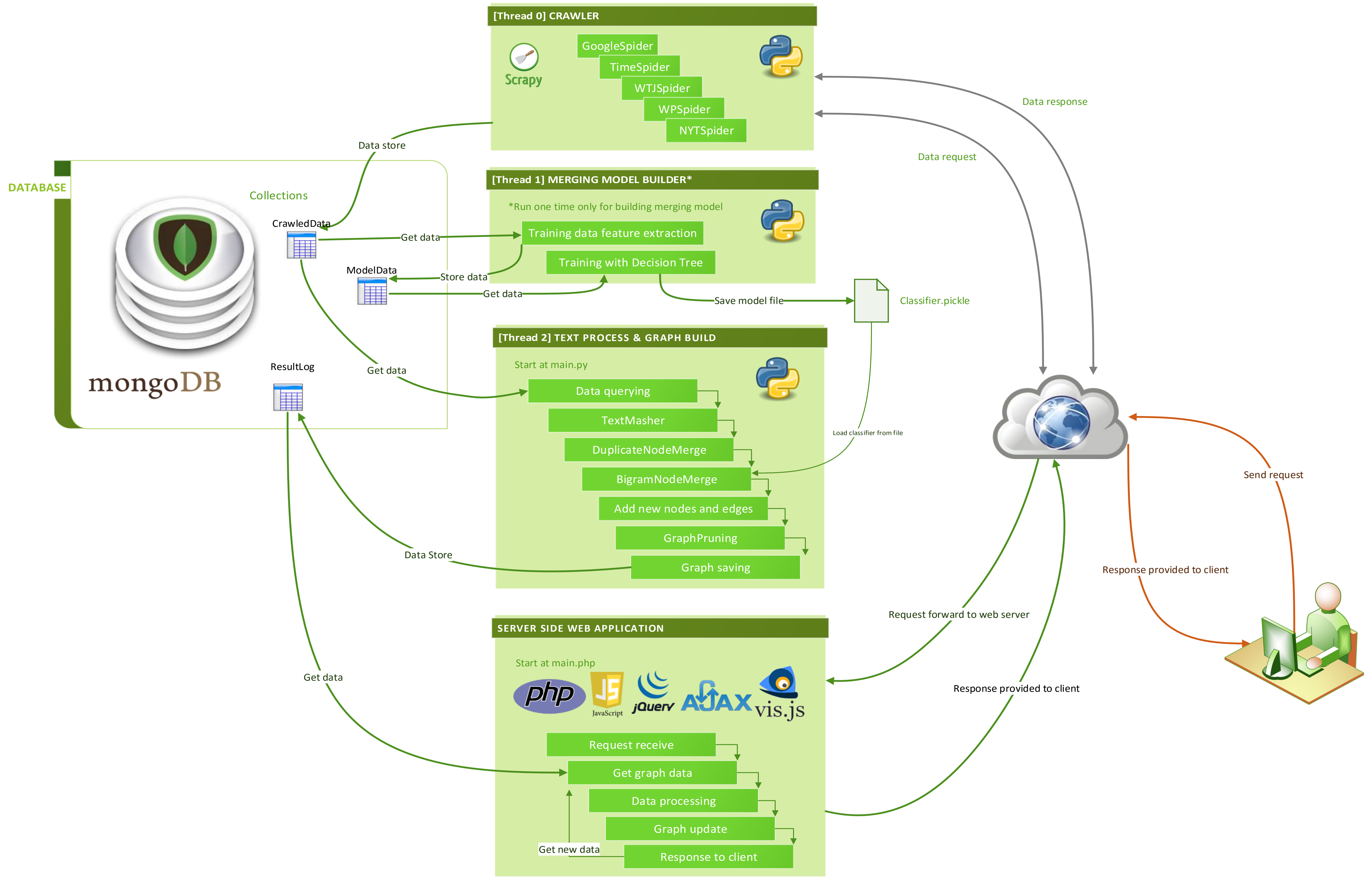

Figure 5 System design and working

2. MODELBUILDER.ZIP

Descriptions: code for extract bigram from data into a training set which will then be used to train a bigram-based classifier for node merging.

Behavior: the program process the text from collection crawleddata. Pair of words that occurred as a bigram will be marked as True (positive) and add to the training set, otherwise, it will be mark as False (negative) before adding to training set. The training set will then being stored in database name: "*rskills*" and collection "*modeldata*". Then another python script will read *modeldata* collection to train a classifier with Decision Tree learning algorithm. The classifier then will be save in file, named "classifier.pickle"

Files:

- **fullfeatureextraction.py**: python script for feature extraction and save to database
- **modelbuilder.py**: python script to load training set from *modeldata*, train the classifier and save to file.

Dependencies:

- NLTK and its dependencies (http://www.nltk.org)
- Pymongo and its dependencies (https://api.mongodb.org/python/current/)

Deployment: the component could run in any machine that have connection to the MongoDB database in which crawled data are stored. Similar to the first component, it requires `HOST` and `PORT` to be correctly assigned.

3. WORDGAME.ZIP

Descriptions: the primary component for code processing. It build a graph structure according to co-occurrence of keywords in text, and then update the graph as new text corpus comes.

Behavior: The program gets data continuously from database until there are no new data coming to the system, then perform text processing activities. Early in the text processing phrase, the text records are normalized, broken in to tokens, then lemmatize. The token sequences then construct a graph structure (at the graph initialization step) or being add into the graph (in case the current graph is not empty). The later period of the process focus on merging pair of words that frequently appeared together and treat them as one compound word. The outcomes – consist of a node set and an edge set accompany with timestamp – are saved to database with the connection through mongod instance.

Files:

- **Graph structure definition**: consist of three files

- **Node.py:** the class defines a node in the keyword graph. Each node store the keyword, its frequency and the latest accessing time point.
- **Edge.py**: the class defines an edge in the graph. Each edge store two keywords that it connected, its frequency and the latest accessing time point.
- **Graph.py**: the class defines a keyword graph, which included a set of Node class and a set of Edge class. For convenience in data access, two reference sets are also stored in the graph structure: set of all keywords in Node set and set of all keyword pairs in edge set. Besides, there are 21 methods was implemented to access data, normalize and maintain the integrity of the graph.

- **functions.py**: contains functions involved in text processing, building part-of-speech tagging model, graph pruning, graph traverse…
- **main.py:** contain the main program. The program must start from this file.

Dependencies:

- NLTK and its dependencies (http://www.nltk.org)
- Pymongo and its dependencies (https://api.mongodb.org/python/current/)

Thresholds: A several threshold are used to control the size and the evolution of graph. They are mostly located at the beginning of **main.py** and **functions.py**. They come with descriptions right inside the code.

Deployment: similar to the previous components, this component can be deployed in any machine chat can connect to the database, and connection parameters should be well configured. Additionally, the program consist of multiple processing steps, hence it require sufficient resources. Another solution is modifying thresholds regarding to the processing environment (such as `DELTA_TIME`, `NUM_MAX_NODES`, `NUM_MAX_EDGES`, `MAXIMUM_PRINNING_ITERATIONS`, `MIN_WORD_LENGTH`, `preferedTags…`)

4. WEBBED.ZIP

Descriptions: the code of a web application that visualize the evolution of the graph over time. Every second, new data are load from database and display under the form of colored weighted acrylic graph.

Behavior: Once receive a request from server, the application start listen to the database. Every second, the program executes a query to the database for new graph data. The response is collect using `$_REQUEST` in php, and processed in Javascript. As a result, new nodes and edges are added to the graph visualization.

Files:

- **Libraries**:
    - **jquery-1.11.3.min.js** and **jquery-ui.js**: JQuery AJAX library, which is used to consecutively update graph without reloading the page.
    - **Vis.js** and **vis.css**: VIS JavaScript library for graph visualizing.
- **getMongoDB.php**: the file that perform graph data query, and the decode the result (which is in JSON format) into array.
- **myjavascrip.js**: the script that initializes and maintains the graph (which is called by index.php). This file also the scripts that make request for data by calling getMongoDB.php.
- **index.php**: the default php file where people send request to. It is the central points that call libraries and myjavascrip.js.

Dependencies:

- PHP (http://www.php.net/)
- JQuery AJAX (http://jquery.com/)
- VisJS (http://visjs.org/)
- PHP MongoDB Driver (https://docs.mongodb.org/ecosystem/drivers/php/) installed to PHP server.

Deployment: The application need to be deployed in webserver, which is sufficient to handle hundreds of nodes for visualization at the same time. In case the web application is running in a machine which is separated from the machine where mongod instance is running, the following variable in **getMongoDB.php** need to be modified:

```
$m = new MongoClient("mongodb://127.0.0.1");
```

IV. EXPORTED DATA

1. CRAWLEDDATA.JSON

Crawleddata.json contain all crawled headlines from news. There are 4 data fields for each record, included:

- `_id`: ID of the document/record
- `content`: content of the headline
- `crawlerid:` specific ID that indicate from which source, the data was collected.
- `Time:` collected time

The following JSON record indicate that a headline with content `"The Life of a Sheepherder"`, was collected at `2015-10-14T17:43:46.807Z` by from Wall Street Journal site.

```
{
    "_id" : ObjectId("561e31e27cae6a1af4eed296"),
    "content" : "The Life of a Sheepherder",
    "crawlerid:" : "WTJSpider",
    "time" : ISODate("2015-10-14T17:43:46.807Z")
}
```

2. RESULTLOG.JSON

Resultlog.json contain records representing the graph states over time.

There are 4 data fields for each record, included:

- `_id`: ID of the document/record
- `timestamp`: timestamp of the graph
- `nodes:` list of all node (represented as a keyword) , followed by its frequency..
- `edges:` list of all edge (represented as pair of keywords), followed by its frequency.

The following reduced JSON document represent a record a state of the graph at `2015-12-10T13:26:07.640Z` with some nodes – “`Biden`”,”`Winner`”,”`Grave`”… with the frequency are 2,3,3 respectively- and some edges – “`basketball - Lamar`” , ”`Spend Competition`”, ”`Hillary - Bernie_Sander`”… ”… with the frequency are 2,2,2 respectively.

```
{
    "_id" : ObjectId("56697d6f7cae6a26506566a8"),
    "timestamp" : ISODate("2015-12-10T13:26:07.640Z"),
    "nodes" : "[\"Biden:2\", \"Winner:3\",... \"Grave:2\"]",
    "edges" : "[\"basketball Lamar:2\", \"Spend Competition:2\",
\"Hillary Bernie_Sander:2\",... \"Halloween Basically:2\"]"
}
```

3. COMMANDLOG.JSON

Commandlog.json records all action of addition and removal edge/nodes in sample user browser. It reveals how the webserver works to provide visualization to user. There are 4 data fields for each record, included:

- `_id`: ID of the document/record
- `action`: Name of the action (`removeEdge`,`addEdge`,`removeNode`,`addNode`)
- `timestamp`: timestamp when the action was performed
- `data:` the target object of the action (node or edge)

The following document indicates that at `2015-12-11T23:43:42.010Z` an edge that connect keywords `Starve` and `Corruption_Current` was eliminated from the user graph visualization.

```
{
    "_id" : ObjectId("566b5fae7cae6a2650989593"),
    "action" : "removeEdge",
    "timestamp" : ISODate("2015-12-11T23:43:42.010Z"),
    "data" : "Starve Corruption_Current"
}
```

4. MODELDATA.JSON

Modeldata.json contains data extracted from the crawled data, which serve ad training data for node merging model. Every record is information related to a bigram occurred in crawled data. It contains 8 fields:

- `_id`: ID of the document/record
- `word1_title`: the value is true if the first character the first word is capitalized.
- `Word2_title`: the value is true if the first character the second word is capitalized.
- `word1`: original form of first word.
- `Word2`: original form of second word.
- `stem1`: stemmed form of first word.
- `Stem2`: stemmed form of first word.
- `tag`: the tag is true if this bigram do appeared in the current block of data, otherwise it is false.

The following document indicate that the bigram consists of two words: strong (whose stemmed from is strong and not in titled state) and dollar (whose stemmed from is doll and not in titled state) was not occurred in the current block of data.

```
{
    "_id" : ObjectId("5649e57d7cae6a0b70d769fb"),
    "word1_title" : false
    "word2_title" : false,
    "tag" : false,
    "word1" : "strong",
    "stem1" : "strong",
    "stem2" : "doll",
    "word2" : "dollar"
}
```