iSearch+: A Smart News Search Engine

EEL6761 Cloud Computing and Storage - Midterm Report

Yifan Wang
dept. Computer & Information Science & Engineering
University of Florida
Gainesville, United States
wangyifan@ufl.edu

Jingyang Guo
dept. Computer & Information Science & Engineering
University of Florida
Gainesville, United States
jingyang.guo@ufl.edu

Qi Jiang
dept. Computer & Information Science & Engineering
University of Florida
Gainesville, United States
q.jiang@ufl.edu

***Abstract*—This is the final report of the iSearch+: A Smart News Search Engine project. As we proposed in the proposal paper, iSearch+ is a software system that integrates data miner into search engine. It is aiming to search information from news articles from major news websites and conduct data mining analysis to extract hidden information in the news articles. In section 1 and 2 We will further propose the project and illustrate the motivation. In section 3 we will introduce the system architecture, including basic components and relationships among them. Section 4 shows our system design as well as implementation, illustrating the main ideas and fundamental tools or algorithms we are using. Our deployment details on AWS are illustrated in section 5. After introducing the system, we performed experiment on the system and the evaluation results are presented in section 6. With all these implement and experiments, we drew some conclusion, and determined some future work in section 7. The team coordination, along with further plans on the project will be presented in section 8.**

***Index Terms*—search engine, fuzzy search, Spark, data crawling, data mining, inverted index, parallel computing**

I. Introduction and Motivation

Nowadays, we are in a information explosion era. The amount of information on world wide web is growing faster and faster. Huge demands for getting needed information as soon as possible have emerged. In order to get information efficiently, users need to use search engine. Search engine [1] refers to a system that automatically collects information from the Internet and, after some sorting, provides the user with a result of query. By nature, information on the Internet is vast and unordered. They are like many small islands connected by bridges. The bridges are web links. Every web page we access includes some web links pointing to other web pages. People can visit other pages by clicking these links from current page. For search engines, they visit as many pages as possible by following these links. then draw a whole picture of the Internet for users.

A search engine consists of crawler, web server, searcher and storage basically. Crawler is the program for fetching web page content. Normally a crawler will visit a web page, get its content, clean what it gets, then continue to visit another page through web links. Crawler is a critical component of search engine. It must be optimized perfectly such that it is able to crawl tens of billions of items in several seconds. In addition, single thread and single computing node are not enough for crawler of modern search engine. Multi-threaded and distributed crawler is the main stream now.

High-performance web servers are key to support high concurrent queries from users. As there are thousands of millions of users on the Internet today, a search engine like Google will receive countless queries even in one second. There are several popular ways to improve server's performance. First, also the basic way, is to use high end hardware, like 256 core CPU. But this makes it too expensive. Then with distributed computing becoming popular, distributed server cluster is the main stream now. Basically a server cluster consists of multiple physical machines. Each machine in the cluster is called a node. There is normally one master node and several worker nodes in one cluster. The master node works on scheduling and assign jobs for worker nodes as well as managing worker nodes, like monitoring their status and restarting unexpectedly interrupted nodes. In the distributed case, data stream can be separated into different worker nodes to reduce the load on each single machine. Load balance is the research topic on how to separate the data traffic in order to balance workload on each node. In industry, load balance is done by a reversed proxy, like NginX. Reversed proxy is a proxy server in front of web server cluster. All data traffic will get through it first and be separated and redirected to different worker nodes. Reversed proxy also has the functions of isolating front and back end files and filtering data stream.

Searcher is undoubtedly core of search engine. One successful searching includes two main processes, inverted index searching and documents searching. Inverted index searching will return corresponding document identifiers and then documents searching will fetch the actual documents using these identifiers. Earlier search engines would search index

rather than inverted index in the first phase. Index refers to the mapping from documents to words contained in these documents. In the mapping, each key is a document identifier while value is collection of words existing in this document. But this approach is way too inefficient because you have to search all keys to check which documents contain the query words. So inverted index was proposed. Inverted index is a inverted mapping comparing to index, that is, it is a mapping from words to the documents containing them. Each key is a word and value is collection of identifiers of documents containing the word. In this case, you just need to look up query words in keys then the corresponding values would be identifiers of documents including those words. This is much more efficient than index searching. So inverted index is a huge progress in search engine field.

Storage is important infrastructure of search engine and normally implemented using databases. There are two major requirements on it. One is storing huge amount of data and the other is supporting high-speed looking up. Single database instance is not enough for storing data today. So developers prefer to use database clusters in large search engines. The inverted index and documents are stored in multiple servers. This feature, along with the independence between different documents, makes it easy to apply parallel techniques when looking up to provide high performance.

By nature, search engine can only search information seen by humans. It is not able to find out more valuable information hidden in what we see. But most of time extracting the hidden information is people's major demand. So it would be much better if the search engine has the ability to mine in data. That is why we would like to integrate a data miner into the search engine.

Data miner is a primary component for iSearch+, which makes it stand out from common search engines. As the quantity of the data that we can get from the Internet increases in exploding speed, extracting valuable information efficiently becomes significantly important, especially when the data source is news, articles or journals, which holds dense information. Searching for specific topic or data and receiving search results along with relative analysis in real time gives user more detailed and accurate information, thus making the information retrieve process much more efficient and fluent.

In our search engine, we introduced sentiment analysis, heat map (mapper) and hot topic trend analysis functions. It is highly significant to conduct sentiment analysis on searched results, say if one needs to know the media or the public's attitude towards commercial products, public figures, literature or movie production or a policy, by merely searching articles from the Internet is a possible way, but a well-constructed chart showing the positive/negative polarity of the articles, along with their proportion, could be a large assistant and save time.

Heat map collects news articles that are mentioning specific locations, marks the city locations on a visualized map and also gives URLs, which are linked to the original articles that mentioned the city. One can search for a topic and check the heat of that topic by location and be aware of how it is spread.

By implementing hot trend analysis and the curve chart, iSearch+ data miner gives user a direct view, like to what extent the topic is concerned as time passes by, with a time axis in the curve chart.

We will illustrate and reference some past relative works and a brief comparison of iSearch+ with previous search engines. The system architecture is illustrated in section 3. Then in section 4.A - 4.B, we introduce the framework design and functions of front end UI and web server; in 4.C - 4.E, we give out a structure of back end architecture, including computing and storage layer, where data miner implementation is concluded, along with the working flow chart.

II. Related Work

Starting from 1998, when Sergey Brin and Larry Page published their paper "The anatomy of a large-scale hypertextual web search engine.", and introduced Google to the world, an era of search engine has lifted its curtain. Search engine has always been a hot topic because of the high requirement for it, and plenty of search engine products has come out since then.

In 2006, Yuangui Lei and his team proposed SemSearch, which is a search engine for the semantic web. Until the paper was published, most existing tools are mainly focused on enhancing the performance of traditional search technologies, aiming to serve those users who are familiar with semantic data, while to some extend ignoring common users who only care about using search results. So they developed the search engine which has such architecture: take a Google-like user interface as the outermost layer, collecting user input, put the input through the text search layer (semantic search), then query for the processed objects to be searched from the World Wide Web or databases.

The iSearch+ system is partly most like SemSearch system, however, the SemSearch is a purely search engine, lacking the functions of data mining. While our project, iSearch+, adds a data mining layer which can deal with the information queried from the news webs, and is able to present those information after analysis to the user, along with semantic searching results.

In 2012, Google proposed Google Trends, which uses search engine data to forecast near-term values of economic indicators, and showed examples including auto-mobile sales, unemployment claims, travel destination planning and consumer confidence. Also Baidu Index provides the function to analysis Internet information and the ability to report the total search volume for a group of terms. These can be seen as a motivation of our project.

While both Google Trends and Baidu Index offers data analysis methods, they only provide a couple of them, and systematically combined data mining modules are the strongest feature of iSearch+.

In section 3, we will further describe system architecture of iSearch+.

III. System Architecture

iSearch+ consists of four main layers Fig. 1: front end pages, web server (back end), parallel computing layer and storage layer.

Front end is UI for users to input what they want to search and get responses. That is what displayed to users.

Web server, or back end, is pivot of the whole system. It receives input from front end, preprocesses the input and sends it to lower level components. Then it retrieves search results from the lower level components and respond to front end with the results. Web server is deployed on AWS EC2.

Parallel computing layer has three main parts, searcher, crawler, and data miner. Searcher plays the main role in searching while crawler takes responsibility of retrieving up-to-date news materials periodically from the Internet. And data miner, as the last process of the data flow, takes news materials that are crawled and stored into the database, apply the algorithms respectly and hand over to the front end. This layer is deployed on AWS EMR.

Storage layer is implemented using MySQL database located on AWS RDS, which stores news articles, inverted index and other materials we need.

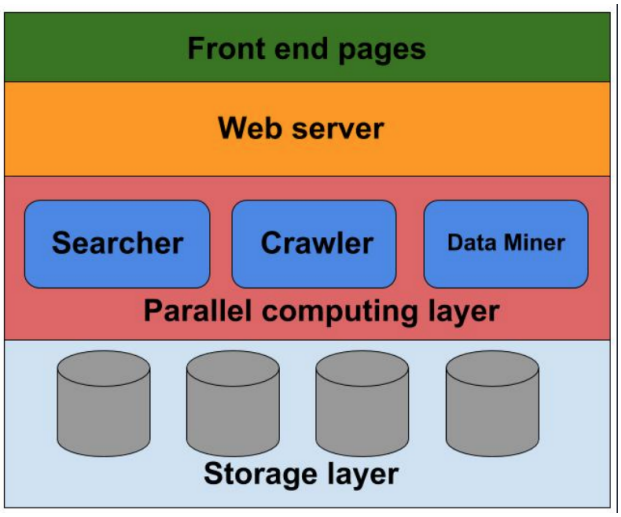

Fig. 1. Main Layers

Basic relationships of each search system component is shown in Fig. 2.

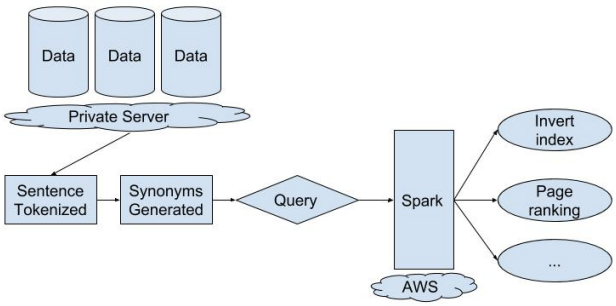

Fig. 2. Architecture of Search System

IV. System Design and Implementation

A. Front End Pages

As a complete search engine, iSearch+ has impressive UI, which is implemented using HTML, JQuery (JavaScript) and Bootstrap (CSS). It gives users a surprising experience when searching and reading the news they need. In this section we will introduce the main front end pages.

1) Main Search Board: First of all, we designed a main search board for our application, check Fig. 3. The search box and button are located at the center of this page. Users search news by typing keywords into the search box and clicking the search button beside it. For example, if a user is concerned about news of stocks, he just need to type a few keywords like ”stock” or ”NASDAQ” and click search button, then he will be directed to the search result page. We also design sign-in button located at upper right corner. Users could sign-in for private search history and personal favorite collection. For users' convenience, we place a clock on this page to help users make their schedules. Daily greeting is shown below the clock to improve user experience.

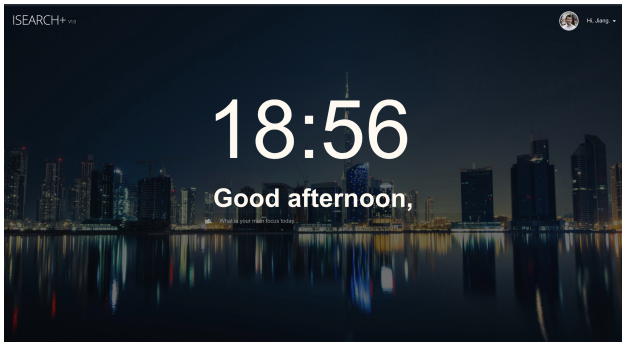

Fig. 3. Main Search Board of Search System

2) Sign-in and Sign-up Page: User need to register at the first and have account for the better usage of our application. As it mentioned above, after registration and sign-in, user is accessible to their personal search history and interested news management. Consequently, sign-in page is designed and implemented as shown in Fig. 4.

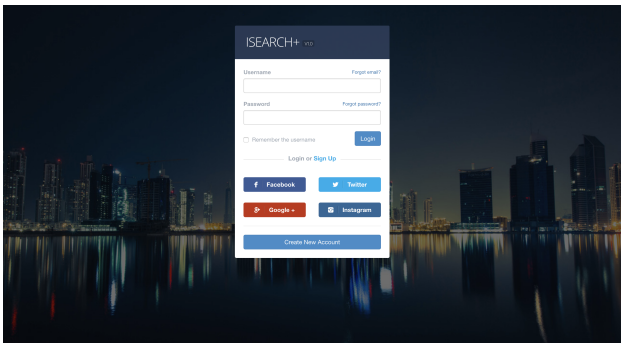

Fig. 4. Sign-In Page of Search System

Also, a sign-up page is made for easy registration like Fig. 5. After clicking register button on the Sign-In Page, it will direct users to Register Page. Users have to enter their e-mail, password and confirm password while registering. Used e-mail cant not be approved for registering new account. After users click register button, if registration fails, the page will display what the problem is and how to fix it. If users register successfully, it will redirect users to Sign-In page.

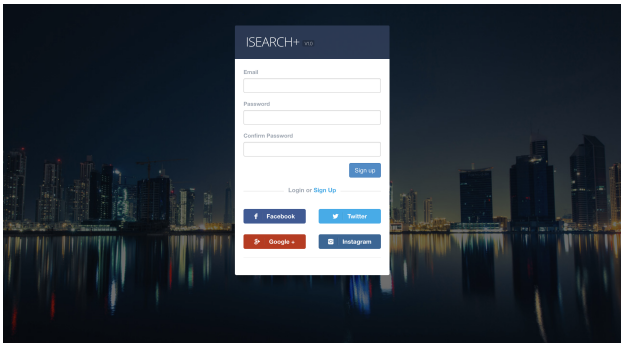

Fig. 5. Sign-Up Page of Search System

3) Search Result Page: Now, we will introduce the most important part of UI in our application. After typing search keywords in search box, web page will direct users to result page. This page consists of several crucial components. In the left side menu, users could get access to searching and viewing history by clicking ”History” icon. There is also some statistical information shown in chart by clicking ”Chart” icon. Search result news are listed at body of the page, user can go to next or previous page to check all the results.

There are also some other pages for functions like news map. We will introduce them later in data miner section.

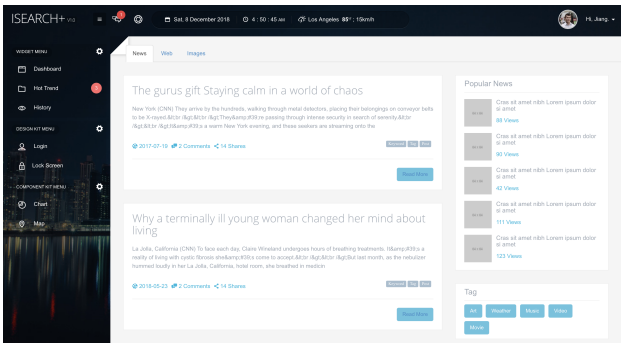

Fig. 6. Result page of Search System

B. Web Server

1) Web Framework: We choose Python Tornado as back end framework. The reason why we choose Python Tornado is that it is an asynchronous framework. The day it started to support asynchronous is even earlier than the time when Python started to support asynchronous officially. This means Tornado can support thousands of connections at the same time, which is a huge advantage comparing to other Python web framework like Flask and Django.

2) Reverse Proxy: We deploy a reverse proxy using Nginx. Nginx is a light weight web server comparing to Apache server. So it is more suitable to act as a reverse proxy.

A reverse proxy is very important for most websites, especially for those which need dense computing, because, as what we say above, reverse proxies can provide better isolation between front and back end files, easier maintenance, simpler management and higher level security.

First, we deploy front end page files on the reverse proxy so that the main server needs only to care about back end computing, which means we separate front and back end more clearly.

Second, since we isolate front end from back end, it is much simpler to maintain both ends now.

Third, another critical task of a reverse proxy is balancing load on web servers. When we have more than one server, we need to balance load on each of them to improve their performance. That is, we are supposed to move some work from heavy-loaded servers to light-loaded servers. At this time, the reverse proxy acts as a server cluster manager which monitors all nodes' load status and redirects data stream from those with heavier load to those with lighter load.

Finally, a reverse proxy can offer the system a higher level of security. The data stream from clients will always get through the proxy first and then arrive at actual servers. So proxy can monitor and filter dangerous data stream before it harms our servers. Even if the evil clients attack the proxy successfully, they cannot get access to any valuable data since the data are stored only on actual servers instead of on the proxy.

C. Parallel Computing Layer

This layer is the core of the whole system. Major works are done in this layer.

1) Crawler: Crawler is the tool for retrieving information from the Internet. Users can access online resources by web browser. But if we want to extract main information automatically from web, we need crawlers.

For fetching online information, a crawler will construct request just like what a browser does at the very beginning. But a crawler will not request for trivial files like CSS and JavaScript files. It only request the files containing the texts, images or other information that it needs.Then it parses the fetched files to extract what it needs and store the information.

Crawler is a very typical case of parallel computing. Most of time, the information we want to crawl has no relationship with each other. In our search engine, we need as many news articles as possible. Each news article is independent from other articles. Whether we have gotten one article has nothing to do with other articles.

As a result, multi-threaded or distributed techniques can improve crawlers' performance significantly. So we design our crawler as a multi-threaded crawler based on master-slave model developed using Python built-in thread library, "threading". One thread works as master while other threads act as slaves. The master is for monitoring and managing slave threads. In addition, it maintains a global queue of URLs (web links) for slaves to read from and update to. In each working round, slaves will read specific number of URLs from the global queue and crawl and save content from the URLs. Once a slave finds new links, it will put them to the queue. This model speeds up crawling significantly as well as guarantees synchronization among all slaves. Because of this reason, it has become the most popular model for large scale crawlers today.

After slaves crawl content successfully, they will save the content as text files on disk. In each file, the first line is section names separated by spaces, like "title", "url", "text" and so on. And the following multiple lines are corresponding content of these sections, separated by symbol "—". Then a data preprocesser will clean the files periodically according to our requirements. For example, it removes non-ascii characters because non-ascii characters would cause error when stored into MySQL database. Once cleaning has done, inverted index will be generated and saved in MySQL along with the original news articles.

2) Searcher: Searcher works on searching inverted index and sorting searching results.

First, searcher will look up inverted index table to find which documents the keywords are in. Then it sorts the documents and returns them to users.

This is also a typical case of parallel computing. Since different documents have nothing to do with each other, we can search them parallelly. So we choose Spark for the searcher. Spark SQL is a component of Spark which is designed for parallel operations on relational databases. The searcher shows a good performance by using Spark and Spark SQL.

3) Data Miner: Data miner lays in the parallel computing layer because it is applied to the searching results right after they are produced by searcher, so the data miner works serially with crawler and searcher. The final output of data miner will be submitted to the web server along with searcher's results.

Our data miner has three fundamental components: Sentiment Analysis, Heat Map (Mapper), and Hot Trend Analysis. More functions could be added dynamically due to further requirements. We deployed the algorithms onto Spark to make them process efficiently.

a) Sentiment Analysis:

Sentiment analysis is for calculating the positive / negative attitude polarity rate of the articles on one topic, given the set of articles that crawler and search engine provided.

For text mining, mandatory process include text summarization, keyword spotting, language tagging, syntactic parsing, named entity recognition, disambiguation, contrastive conjunction (for sentiment analysis) and coreference, etc.

For text summarization, we used TextRank algorithm, which was proposed by Rada Mihalcea, and Tarau Paul in 2004, and is a graph-based unsupervised ranking model for text processing relying on an algorithm derived from Google's PageRank. "Voting" and "recommendation" are two key concepts of TextRank, the core idea is based on processing a undirected / directed graph, and represent a vote from one vertex to another as edges, i.e. links between vertex. And the number of votes cast for one vertex is considered as a quantified importance index of the vertex, the higher the number of votes is, the higher importance the vertex has, and what's more, the importance of vertex A also influences the importance of its vote cast for other vertices, in other words, the higher importance of one vertex is, the more important its voting is. The voting importance is also took into consideration of the ranking model. So both the score of vertex V itself and all the vertices that votes V determine the final score of V. Note that a TextRank implementation needs numpy and NLTK python packages, and with the help of text processing toolkits, generate undirected and weighted graphs from text, each node representing a single sentence and start the voting and selecting process.

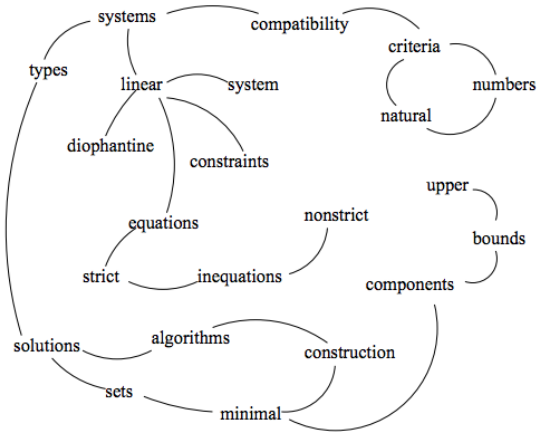

Fig. 7. TextRank Algorithm

The extracted important sentences are then applied to speech tagging algorithms, in order to tag on each word in the important sentences and retrieve valuable words that contributes to sentiment polarity decision. We use speech tagging methods provided by NLTK, and label the words into like ”CC” (coordination conjunction), ”RB” (adverbs) and ”NN” (none). We mainly focus on ”JJ” - adjectives, which can contribute most to sentiment polarity judgement, and select ”RB” - adverbs and ”CC” (coordination conjunction) as assistant words. What worth clarifying is that ”CC”, as a relatively important factor, when the CC is a contrastive conjunction, like ”but”, ”while”, ”however” or ”nevertheless”, the meaning of the whole sentence can be totally different; and further more, cases happen that sentences like ”I can not say it is bad”, the two negative words existing in one single sentence make it relatively positive. Such special cases should also be concluded into consideration, like detecting for contrastive conjunctions and reverse the sentence’s polarity.

TextBlob is a simplified text processing package for Python 2 and Python 3, and it provides some API for some common NLP tasks like speech tagging and clasification, so we made use of this package to implement our sentiment analysis function.

By using these algorithms, given a list of articles (which often belong to same topic since most time they are the search engine result), the sentiment module can return a dictionary labeling the number of articles holding a positive / negative / neutral attitudes towards the topic. And we present the result with a bar chart in the front end part, such that each time the user queries a key word, the web server shows all the results along with the polarity bar chart.

In the front end, the polarity bar chart is shown as Fig. 8.

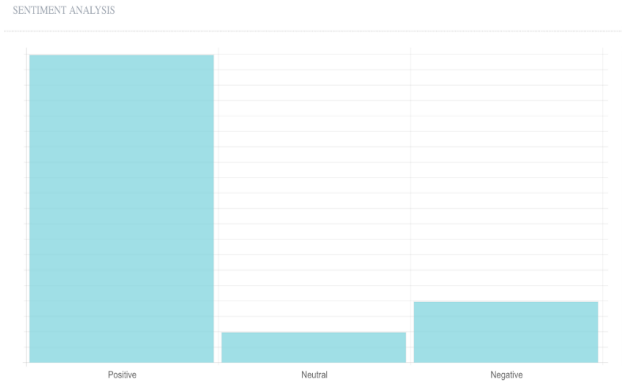

Fig. 8. Sentiment Analysis

b) Heat Map (Mapper)

Mapper is another critical component in data miner, showing the locations where target articles mentioned, and the URL links to the original news. Imagine such a situation, under which one wants to check hurricane status in Florida. Simply searching ”Florida hurricane” is one way, but in order to be more visualized and convenient, we can also refer to the generated heat map, where we can clearly see which cities in Florida showed up in the news about hurricane, further more, which specific news talked about it. What’s more, if we are just searching for ”hurricane”, we can even have a brief look at the spread of hurricane events all over US or even the world, getting a fuzzy acknowledge of the hurricane distribution in an indirect way.

We implemented the Mapper component on Spark, in order to ensure a high efficiency as a query is submitted to the search engine and the result articles are fetched back. We made to versions of Mapper implement to experiment the efficiency upgrade that Spark can bring us. At first, we implemented the mapper in a local method, with the help of pandas toolkit we keep a city name table, and look up for all the possible city names that could show up in each of those articles. We got the city names database from MaxMind, an IP Geolocation and online fraud prevention service provider, who provides a list of worldwide cities along with their latitude and longitude. However, it could make web server dangling too much time waiting for the results.

The Spark version behaves much better. We flatten and parallelize the article words, make every word that is contained in the article as its key, article itself and its URL as the value, thus forming RDDs; on the other hand, extract the city name from MaxMind’s database as the key, and the remaining information such as longitude, latitude, region, population and country as value. Then make join operation to the RDDs and leave the common keys, i.e. the city names, and finally get the city names and the articles that related those cities.

At the front end part, we present the Mapper result on a heat map chart, with markers spreading across the map to give a brief view of the presence of cities about one topic. The Mapper UI is shown as Fig. 9.

Fig. 9. Mapper Function: Categorize News with News Markers

The source news can be visited by accessing to the URL link in the info-window floating above the marks. Check Fig. 10.

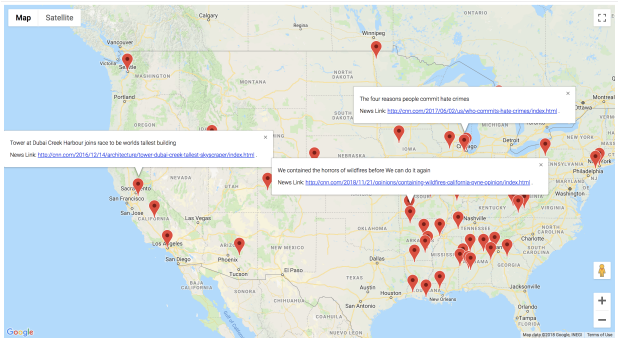

Fig. 10. Mapper Function: Read News with Info-window

c) Hot Trend Analysis

Hot trend news and popular news are always an excellent approach for users to learn about fresh information. Aiming at this need, we design this feature in our search system.

We define hot trend news as the news with hottest key words. More specific, the hot keyword in hot trend news should have a high frequency among all the recent news. In order to get these hot keywords, we have made a statistics of the keywords that appear frequently in the current news database and get article list that is related to these keywords using inverted index method. With article list result, more meta info of articles could be get like article title, article context, article URL and etc.

For better usage, we made a hot trend news scanning page. User could enter this page by just click on the icon button in the left side menu to get most useful news to read.

D. Storage Layer

This is a MySQL database deployed on AWS RDS. It is a data exchange center for the system, storing inverted index table, documents, crawled web data and users' data to be requested by the web server.

E. Working Flow

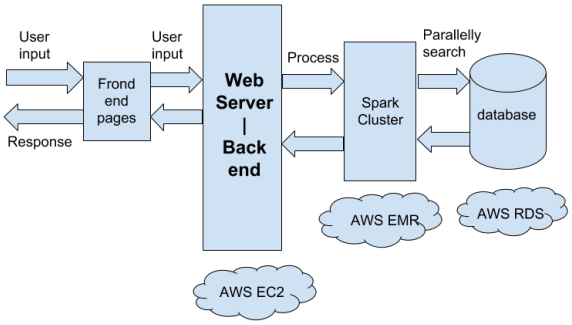

Fig. 11. Main Working Flow

As shown in Fig. 11, our search system mainly composed of two parts: front end and back end. Our system could run on both local host and cloud environment like Amazon Web Services or Google Cloud.

For the frond end part, at the beginning, system users could tap in some input query keywords. Front end will pass them to back end server when get these search keywords. Also, when search engine has searched for correct answer, the front end page can receive these results and give them to system users in various visualization approaches.

The back end part can be built on cloud with AWS services. Web server can preprocess original request such as splits query string into separated words and this can be accomplished on AWS EC2 instance. After preprocessing step, the Spark in cluster will search the keywords in AWS RDS database paralleling to ensure a high quality of searching speed. We will have more concrete introduction about combining our system with cloud service on the following section.

V. Deployment on AWS

Amazon Web Services is a subsidiary of Amazon.com that provides on-demand cloud computing platforms and it is very popular these years. It offers reliable, scalable, and convenient cloud computing services. Consequently, not only on local host, we also deploy our iSearch+ system on AWS environment.

In order to build the system on cloud, we should take some steps to make the cloud environment prepared. First of all, a EMR cluster on AWS need to be created in Fig. 12. This cluster is composed of one master and two slave nodes. After creating this cluster successfully, the cluster begin state running and we start configuring iSearch+ system dependencies on cloud environment.

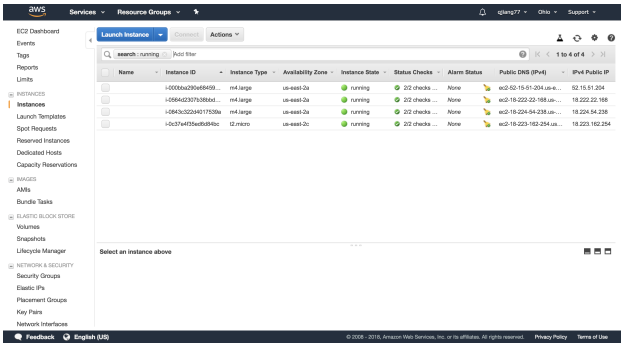

Fig. 12. Create a new AWS EMR cluster

The operating system of EMR is CentOS. Using ssh client, localhost could connect to EMR cluster on cloud like Fig. 13. Furthermore, several dependencies are needed to be configured, such as tornado and etc.

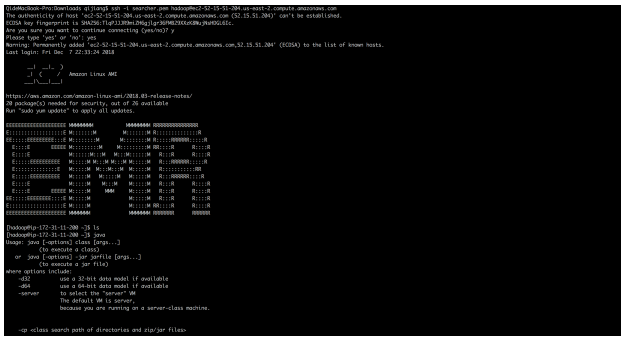

Fig. 13. Use ssh client to connect to EMR cluster

For the storage part, we make AWS S3 to store the news article which crawled by our crawler. Moreover, AWS RDS also plays an important role as MySQL database on cloud. With all these configuration done on AWS, our system could run normally on it.

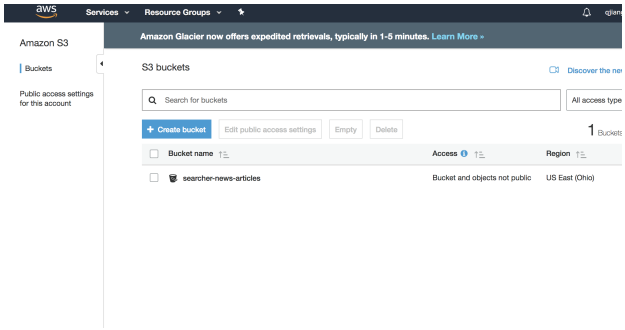

Fig. 14. Create AWS S3 for storage

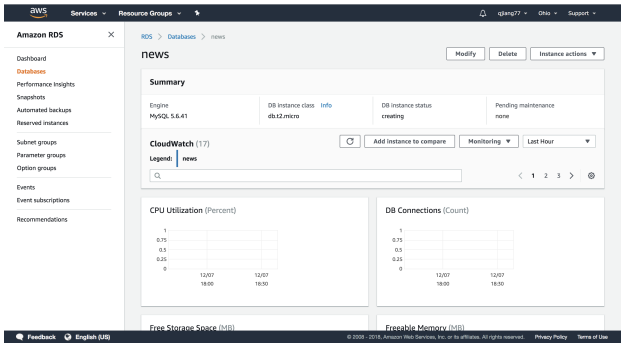

Fig. 15. Create AWS RDS as Database: MySQL

VI. Performance Evaluation

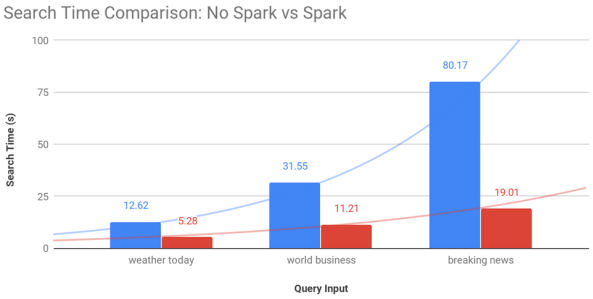

Fig. 16. Search Time Comparison: No Spark vs Spark

To evaluate performance of our system, we compared search time with and without Spark.

A. Experimental Setting

For convenience, our experiment is done on local PC. The hardware conditions are 2.4GHz Core i5 CPU with 256G SSD and 8G main memory. The software are Python 2.7, Spark 2.3.1, and MySQL 5.8.

B. Performance Metrics

The metrics is running time from clicking search button on the main page to search result page loaded completely.

C. Experimental Results

As shown in Fig. 16, searching with Spark is much faster than without Spark. It also shows that along with query word becoming more general, the difference between those two cases are larger and larger. It is because a more general word indicates larger amount of documents behind itand Spark can show more of its potential when used on a large dataset.

D. Discussions and Limitations

This experiment shows that using Spark gives our system a significant improvement on its performance, especially when we are searching some general words which are contained in huge amount of documents. Its limitation is that configuration of Spark is relatively complex. This makes the whole iSearch+ system more complex to be configured.

VII. Conclusion and Future Work

This paper introduces a search engine system which could search massive information data from a great number of news articles. Inverted index plays a very important role in searching articles with keywords. It is a popular approach in search engine which is widely used. To optimize this search system, we adopts data mining and parallel computing techniques. There are some different algorithms referred in data mining part, such as sentiment analysis, text rank and etc. All these algorithms help mine useful information for users. In order to get better performance on search speed, we use Spark to improve the whole system performance.

Not only efficient algorithms, our system also has a convenient user interface which is easy to handle for onboarded users.

There are also some features we expect to implement in the future, such as business analysis, the collection of hot key tags and stock price prediction. User can just go into popular news or focus on hot keywords to get fresh and important information. In addition, we expect the reinforcement of algorithms, such as text summarization, text classification and time series analysis.

We will also work on the performance improvement of searching speed. Maybe greater number of GPU and better parallel computing solutions would also be applied in our system in the future.

VIII. Project Management

A. Team Coordination

1) Yifan Wang: Data crawling, searcher development, back-end programming, AWS and MySQL configuration, documentation.

2) Jingyang Guo: Algorithms, data mining, documentation.

3) Qi Jiang: UI design, back-end programming, front-end programming, documentation.

References

[1] Brin S, Page L. The anatomy of a large-scale hypertextual web search engine[J]. Computer networks and ISDN systems, 1998, 30(1-7): 107-117.

[2] Thelwall M. A web crawler design for data mining[J]. Journal of Information Science, 2001, 27(5): 319-325.

[3] Paramkusham S. NLTK: THE NATURAL LANGUAGE TOOLKIT[J].

[4] Sauls C, Leithead T, Tokumi R. Synonym and similar word page search: U.S. Patent 7,822,763[P]. 2010-10-26.

[5] Zaharia, Matei, et al., "Spark: Cluster computing with working sets.", HotCloud 10.10-10 (2010): 95.

[6] Isele, Robert, et al. "LDspider: An open-source crawling framework for the Web of Linked Data." 9th International Semantic Web Conference (ISWC2010). 2010.

[7] Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." Proceedings of the NetDB. 2011.

[8] Ranjan, Rajiv. "Streaming big data processing in datacenter clouds." IEEE Cloud Computing 1.1 (2014): 78-83.

[9] Pang, Bo, and Lillian Lee. "Opinion mining and sentiment analysis." Foundations and Trends in Information Retrieval 2.12 (2008): 1-135.

[10] Alexander, Carol. Market models: A guide to financial data analysis. John Wiley & Sons, 2001.

[11] Brin, Sergey, and Lawrence Page. "The anatomy of a large-scale hypertextual web search engine." Computer networks and ISDN systems 30.1-7 (1998): 107-117.

[12] Lei, Yuangui, Victoria Uren, and Enrico Motta. "Semsearch: A search engine for the semantic web." International Conference on Knowledge Engineering and Knowledge Management. Springer, Berlin, Heidelberg, 2006.

[13] Mihalcea, Rada, and Paul Tarau. "Textrank: Bringing order into text." Proceedings of the 2004 conference on empirical methods in natural language processing. 2004.