Zhenfeng Shi | Report

SJTU – Shanghai – China
✉ jack.shi2013@gmail.com • Report for EE327

Search Engine Group June 18 2016
IIoT, Lab.
1-434
SJTU

Work Overview

Search Engine

In the first few months of this semester, our group developed a search engine without any open-source packages like *Lucene* or *Solr*.

The work distribution is mainly as follows:

- Index Creation
- Ranking and Evaluation
- Text acquisition
- Text transformation
- User interaction

I participated the work of Index Creation, doing research and implementing index construction and index compression. Also, I learned how to use NLTK (Natural Language Toolkit) to do natural language processing and gave a speech on a Monday group meeting.

Web Crawler

In the last few weeks of this semester, our group is assigned to crawl data from the website.

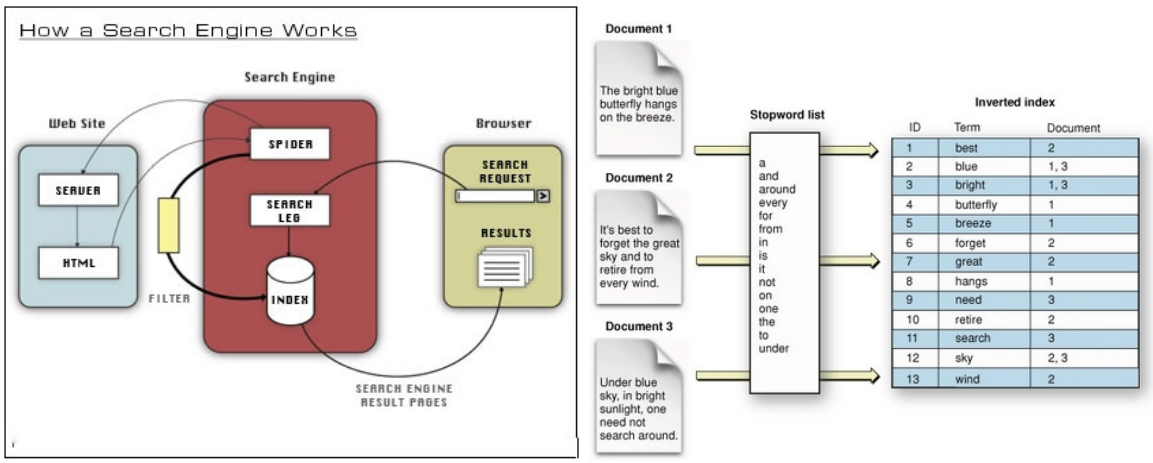
Each member was in change of two website. For me, it was *CiNii Articles (http://ci.nii.ac.jp/en)* and *ERIC (https://eric.ed.gov/)*.

In these few weeks, I designed a Hadoop-based distributed web crawler which can crawl data faster than the traditional way.

Detail Description

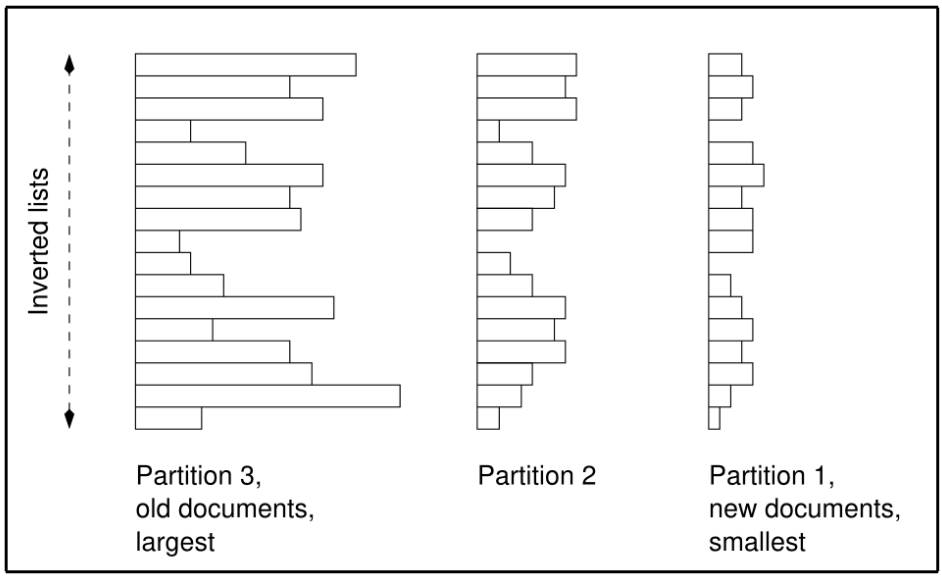
Search Engine

Search engine indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. The inverted index data structure is a central component of a search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs.



Nan Zuo and I are in change of the Index Creation. We mainly implements the inverted index on HDFS, which is a Java-based distributed file system provided by Hadoop.

An inverted file index contains two main parts: a vocabulary, listing all the terms that appear in the document collection; and a set of inverted lists, one per term. Each inverted list contains a sequence of pointers (also sometimes known as postings), together with a range of ancillary information, which can include within-document frequencies and a subsidiary list of positions within each document at which that term appears. A range of compression techniques have been developed for inverted lists, and, even if an index contains word positional information, it can typically be stored. Index compression also reduces the time required for query evaluation. The standard form of inverted index stores the pointers in each inverted list in document order, and is referred to as being *document sorted*. Unlike the traditional partitioning, we used a new partitioning method name *Geometric Partitioning*, which is proposed by N. Lester, A. Moffat and J. Zobel[1].



As in the above figure, The oldest index pointers are in the lowest, largest partition, which in this example is at level 3. The vocabulary, not shown in the figure, includes three pointers with each term's entry.

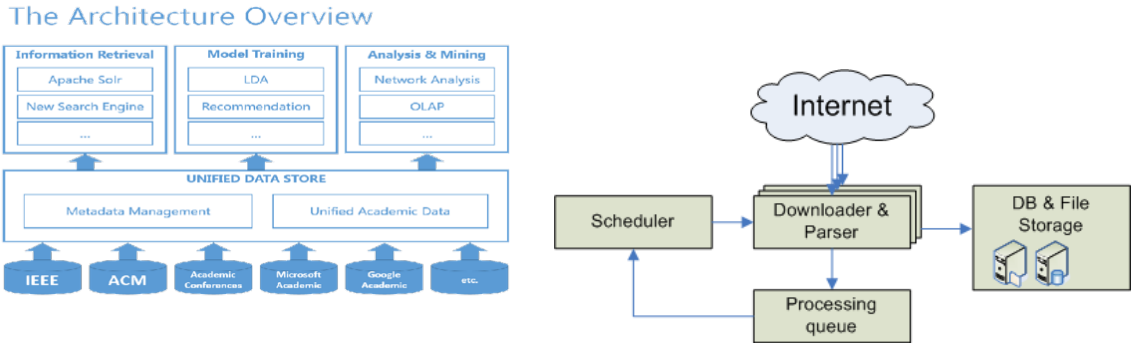
By implementing this method, the disk access cost is approximately as follows:

$$\frac{r-1}{r}(0.5+\frac{log(n/b)}{logr})$$

which is significantly reduced compared to standard contiguous representation of inverted indexes.

Web Crawler

Web crawler plays a very essential role in our lab project, academic search engine. We need paper information from the website to be stored on our local server to support our upper layer analysis.



Basically, we got two different methods to crawl the data from the web, which is listed as follows:

1. Crawl data from *Google Scholar, Microsoft Bing*
2. Crawl data from organization website like *IEEE, ACM*, etc.

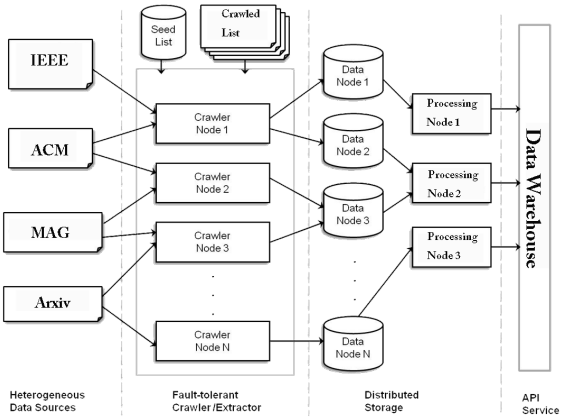
The former choice is a dead end because companies like Google or Microsoft have more advanced technology, experience and equipment than us. They know how to crawl and how to prevent crawling better than all of us. Any kind of robotic behavior on these websites will be detected and blocked.

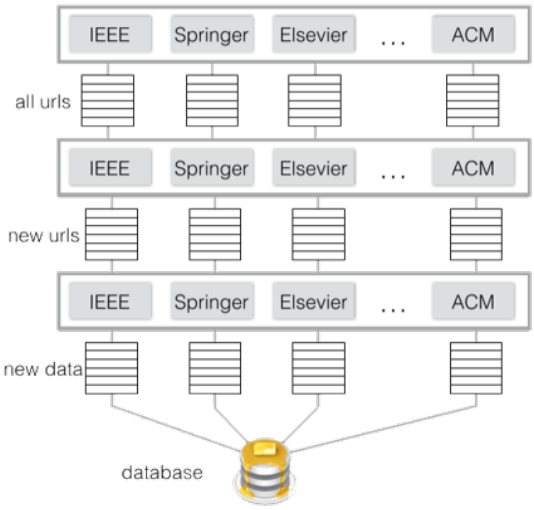
For the latter solution, if we use a polite crawling method (which means in a relatively low requesting speed), such crawling behavior on these organization websites might not be detected and blocked. But there are also drawbacks on this method:

1. Different organization websites have different HTML format and taggers. Therefore, we have to design specific crawling programs for specific websites. This will cause redundant effort.
2. Some websites like *ACM* may still have secure mechanisms. Therefore, our offensive fast speed crawling behavior might also be detected, warned or blocked.

A systematic web crawler platform is in need to deal with the situation. Such system should have the following features:

1. Automating crawling, processing and storing data from website to local server database.
2. Long-term monitoring on each websites, detect new papers and crawl them automatically.
3. Automatically switching VPNs.





Hadoop-based web crawler should be a possible answer. Hadoop is open-source software which provides reliable, scalable and distributed computing. It mainly has four modules:

- *Hadoop Common* provides the common utilities that support the other Hadoop modules
- *HDFS* is a distributed file system that provides high-throughput access to application data
- *YARN* is a framework for job scheduling and cluster resource management

- *MapReduce* is based on YARN, for parallel processing of large data sets.

I have implemented a simplified system on Hadoop using 3 servers in the lab. In pratice, I found that using distributed crawling system can reached a far more swift speed than single task or multi-threading task in Python. Also, since the resources and nodes in Hadoop is automatically managed by the software, the automation and performance optimization should not bother. For each node, a specific program designed for specific websites can be attached using different VPN.

Reference

1. Fast On-Line Index Construction by Geometric Partitionin, Nicholas Lester, Alistair Moffat, Justin Zobel, CIKM'05, October 31–November 5, 2005