Implementation of a High-Performance Distributed Web Crawler and Big Data Applications with Husky

Franklin Lee
The Chinese University of Hong Kong

Abstract

Husky is a distributed computing system, achieving outstanding results in large scale data mining and designing efficient distributed algorithms. Considering low efficiency of systems like Hadoop and Spark, etc., it is a motivation for us to start the Husky open-source project, which attempts to strike a better balance between high performance and low development cost. Web crawler is a well-known method of collecting huge amount of data for implementations relating to big data. However, simple compiler system shows low efficiency in crawling billions of data. With the help of Husky, it is a motivation to create a high-performance distributed web crawler which can complete a great quantity of crawl assignments in a high speed.

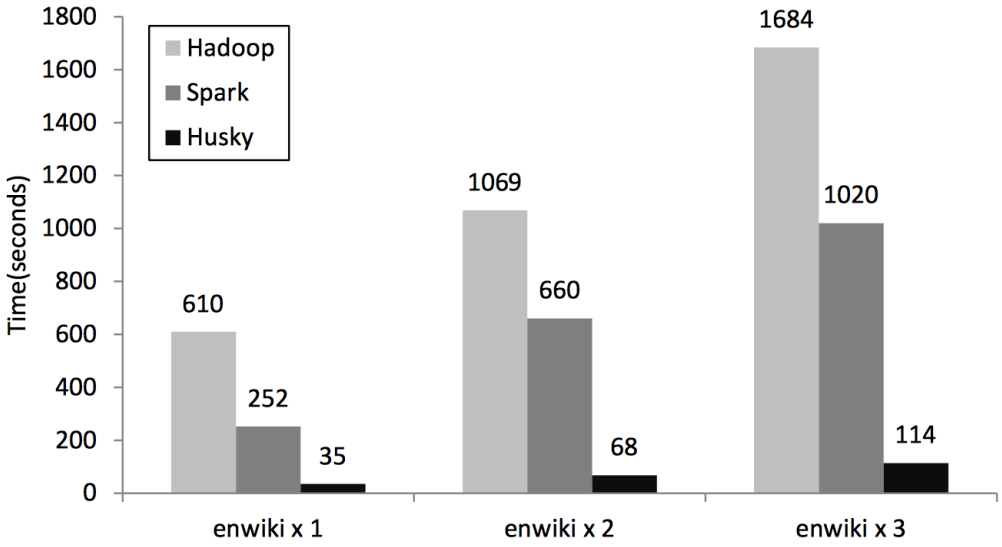


Figure 1. Performance on TF-IDF

1. Introduction

In order to improve and perfect the performance of Husky[2], division of labor assigned to different task groups (e.g. Pyhusky group, machine learning group, crawler group). I am belonging to crawler group; whose task aims at providing plenty of raw data to build a big data foundation so that other group members (e.g. machine learning group) can train their specific mode or test their existing algorithms. Meanwhile, some attempts related with big data can be efficiently analyzed by utilizing crawled data (e.g. customer impression analysis, prices of houses analysis, conference topic trend analysis). With the help of high efficient system Husky, billions of data can be downloaded

in few minutes.

2. Basic Crawler Implementations

A crawler is a program that automatically collects Web pages to create a local index and/or a local collection of web pages. Roughly, a crawler starts off with an initial set of URLs, called *seed URLs.* It first retrieves the pages identified by seed URLs, extracts any URLs in the pages, and adds the new URLs to a queue of URLs to be scanned. Then the crawler gets URLs from the queue (in some order), and repeats the process.

In my approach, basic crawler code is written in python, which has powerful library like RE, BeautifulSoup. When facing different websites, crawling methods need changing in order to fit different situations. In general, two kinds of crawling methods are developed and implemented:

The first method is simple but useful, which only needs few lines of code and can crawl thousands of pages. This method makes use of URL to distinguish pages' differences by number changes. We can slightly change the number so that we can link to adjacent webpages:

```
def trade_price(max_pages):
    page=1
    while page<=max_pages:
        url='http://sz.lianjia.com/ershoufang/luohu/pg' + str(page)
+ '/'
     page += 1
```

Listing 1: Core Code Method 1

Websites like stocks information, house prices, customer comments are usually fit the previous style mentioned above.

The second method is powerful and comprehensive, which utilizes distributed script files which makes each process of crawling clear and obvious. First insert the main websites of crawling object, and define the limit requesting time for each page, then assign proxy servers and make arrangements for RE search rules in order to link to wanted resource webpages, finally download related data to HDFS files and repeat the whole process. Main functions are defined in hcrawl, and it is quite simple for user to import it in program and follow steps to crawl:

```
__init__.py
__init__.pyc
crawler_config.py
crawler_config.pyc
crawler_dist.py
crawler_dist.pyc
crawler_utils.py
crawler_utils.pyc
```

Listing 2: Core Code Method 2

3. The High-Performance Distributed Web Crawler Approach

Husky is an efficient and expressive distributed computing framework; It has been shown that many existing programs can be easily implemented and bridged together inside Husky, and Husky is able to achieve similar or even better performance compared with domain-specific systems. Considering Husky system's powerful computing framework, it is a motivation to combine Husky with crawler to make a high-performance distributed web crawler.

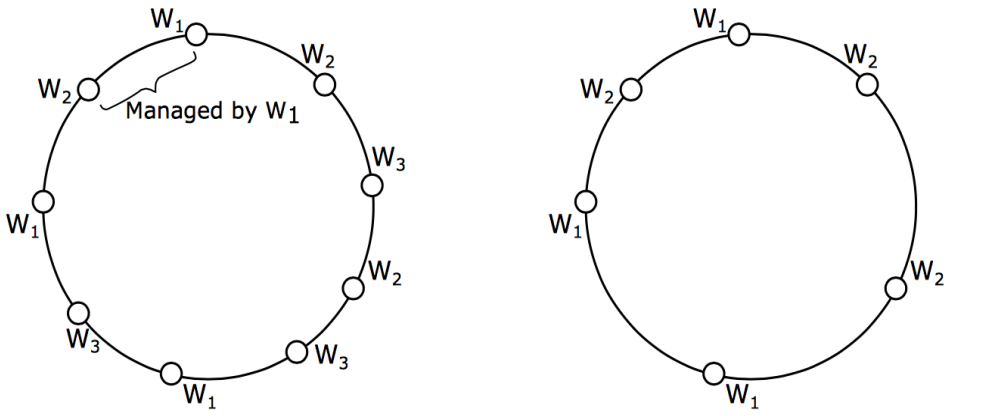
3.1 Workers

A worker can be regarded as a special type of objects that can read from external sources, e.g., *Hadoop distributed file system (HDFS)*, and create objects. Husky allows users to specify a partition function to assign objects to different workers. An application can achieve better performance if the partition function groups objects that frequently interact with each other in the same partition.

3.2 Master-Worker Architecture

A Husky cluster consists of one master and multiple workers. The master is responsible to coordinate the workers, and workers perform actual computations. To perform a round computation (by calling list execute), a worker first receives all incoming communications (if any) from other workers, dispatches messages to objects and invokes their execute function, and finally flushes the outgoing communications that object have generated.

Usually a worker flushes outgoing communications only after all workers on its same host finishes the current execution round. The purpose is to exploit the effective shuffle combiner technique to maximize message reduction. However, when no combiner is provided, Husky will batch and flush message right after they are generated, in order to interleave CPU usage and networking IO.



(a) Normal hash ring (b) Hash ring after W_3 removed

3.3 Crawler Distribution with Husky

Web crawlers are always dealing with massive data processing, which usually consume much time and CPU. Generally, a web crawler starts with a list of URLs to visit. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies. If the crawler is performing archiving of websites it copies and saves the information as it goes.

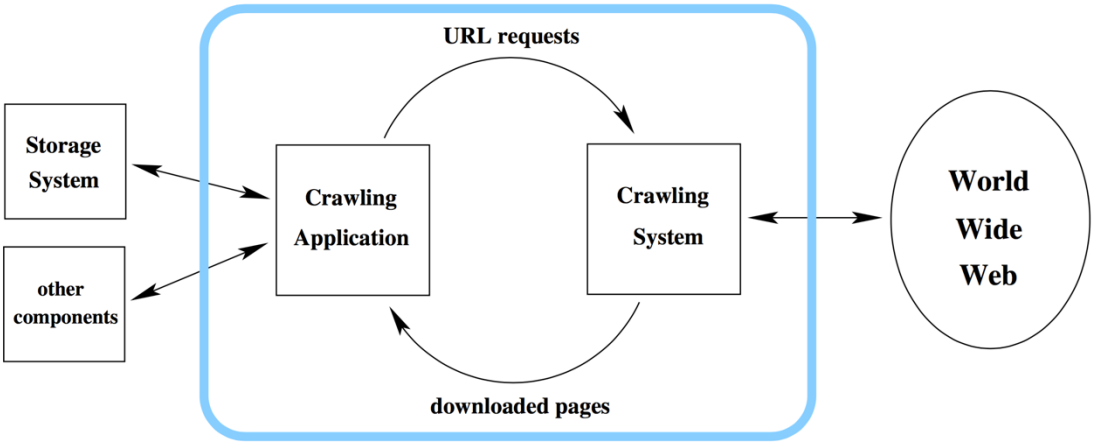


Figure 2: Basic Two Components of the Crawler

With the help of Husky, the task of crawler can be distributed to workers so that the speed can be greatly enhanced. After simple commands in dialog window:

- `./Master conf/<user>.conf`
- `./Daemon conf/sg.conf`
- `python <file_name> --host <master_host> --port <master_port>`

Then add Pyhusky command in order to utilize Pyhusky library which make program clear and fast, e.g.:

- `ph.env.pyhusky_start()`
- `husky_list=ph.env.parallelize(data_cloud)`
- `husky_list.write_to_hdfs("/datasets/stock_company_information/")`

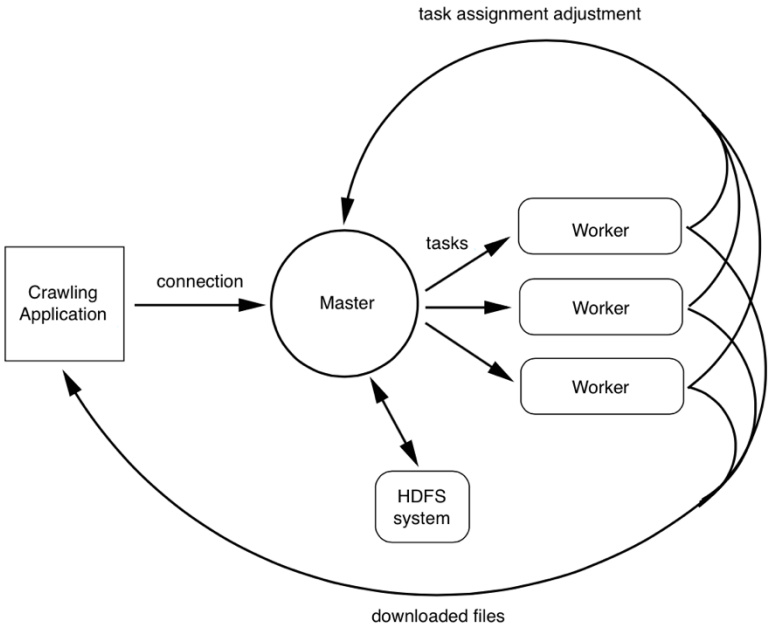


Figure 3: Crawler Configuration with Husky

With the enhancement of Husky system, crawling speed will skyrocket up. In single local server, tested program can reach approximately 5 to 10 HTML pages per second. By comparison, a configuration as the one in Figure 3 would require between 3 to 5 workstations, and would achieve an estimated peek rate of 250 to 400 HTML pages per second.

4. Big Data Applications with Husky

As the super advantage in crawler with Husky, it is very fast to gain a large amount of data. With huge amount data, it is interesting to make analysis to obtain accurate insights. Also, by analyzing specific data in certain industry, we can even comprehend and forecast the future trend, and make corresponding adjustment with the support of evidence-based data.

4.1 City Super Customer Impression Analysis

City Super is a supermarket in Hong Kong, and it is famous for imported product and high-quality service, whose target group is main high-income people. For a company, it is very important to investigate customer impression and feedback to make better management and further progress. However, it is very difficult to know the facts by real questionnaires, with big data, we can have the aid of data visualization to figure out customer impression by retrieving core words from customer comments.

By web crawler with Husky, customer comments from Da Zhong Dian Ping website can be easily downloaded. We specify one store of City Super located in IAPM, retrieving its comments from 2014.5.1 to 2016.5.25. Totally three data sets with 42,955 Chinese words. Then

convert the comments into English, and design algorithm to realize word count and word selection (e.g. exclude common words “is”, “the” etc.). Finally, we can apply Word Cloud Data Visualization technique to figure out the customer impression.

Figure 4: City Super Customer Comments 2014-2016

By carefully observing the data visualization results, it is obvious to figure out the main impression of customer toward IAPM store: Good, Fresh, Imported, Price, etc. And this just accords with City Super’s core ideas. Also, by comparing words through different years, it is clear to find out the trend of customers’ impression (e.g. word “Japanese” becomes smaller and smaller, it means the attention of customers towards Japanese product decreased). Store manager may adjust the product model to comply with trend.

4.2 Topic Modeling and Data Visualization for Conference Analytics

INTERSPEECH is an academic conference, which usually calls for papers referring to a list of tracks with a high-level description regarding the scope of the conference. Tracks can offer effective approaches where authors can submit papers to their preferred tracks (Table 1), reviewers can acquire related papers in preferred tracks, readers can search their favorite papers in corresponding tracks.

1. Speech Perception and Production
2. Prosody, Phonetics, Phonology, and Para-/Non- Linguistic Information
3. Analysis of Speech and Audio Signals
4. Speech Coding and Enhancement
5. Speaker and Language Identification
6. Speech Synthesis and Spoken Language Generation
7. Speech Recognition - Signal Processing, Acoustic Modelling, Robustness, and Adaptation
8. Speech Recognition - Architecture, Search & Linguistic Components
9. LVCSR and Its Applications, Technologies and Systems for New Applications
10. Spoken Language Processing - Dialogue, Summarization, Understanding
11. Spoken Language Processing -Translation, Info Retrieval
12. Spoken Language Evaluation, Standardization and Resources

Table 1: Interspeech Conference Tracks

By powerful crawler with Husky, we can collect Interspeech (2014)’s conference papers. After arranging, Interspeech 2014 conference released 12 main tracks, 1,078 paper abstracts. The total number of paper abstracts is 1,078, words of which are 101,312. The average words in a paper abstract are around 94. After pre-processing in terms of stemming and removal, the vocabulary contains 5,732 unique words. Each data set contains twenty documents with detailed statistics of latent topics retrieved from papers in format of .tsv.

Each document contains 200 words with specific probabilities both in percentage form and exponential form varying from 2000 to 2015.

In approach to analytics of topic modeling, we can apply Latent Dirichlet Allocation (LDA[1]) technique. LDA is a topic probability model producing distinct sets of data like text corpora. The formula (1) of LDA is illustrated by referring to the notions which α and β are hyper-parameters, M is a set of documents with discrete probability from a corpus D, θd is a topic mixture for document d, ωdn is the nth word in document d, and zdn is the underlying topic assignment for word wdn with θd.

$$p(D|\alpha,\beta)=\prod_{d=1}^{M}\int p(\theta_d|\alpha)\left(\prod_{n=1}^{N_d}\sum_{z_{dn}}p(z_{dn}|\theta_d)p(w_{dn}|z_{dn},\beta)\right)d\theta_d.$$

In procedure of LDA, each document is generated by sampling each single document-specific topic in proportion of θd form a Dirichlet distribution, and then each word is separated from a topic-specific Multinomial distribution $p(w_{dn}|z_{dn},\beta)$. A low-dimension of data is generated with a word distribution of $P(w|z)$, which represents the probability of a word w in topic z and a topic distribution in certain document $P(z|d)$, which are the mixture of topics in document d. The main interest is on $P(w|z)$ because it is an important criterion between latent topics and given tracks.

Basically, both an assumption that a 1: N relationship between track and topic and a definition that a match is the combination of a topic with its highest-rank track according to F-score are prerequisite. Based on the information framework which each document is a track and each underlying topic is a topic, each topic can be generated by means of LDA by selecting its top N = 200 words in descending probability order. This parameter's coverage is around 80% of the probability space in each single topic. Around 20-50 words are pre-processed according to track descriptions. The processing system of topic-track matching system is represented (Figure 5) below.

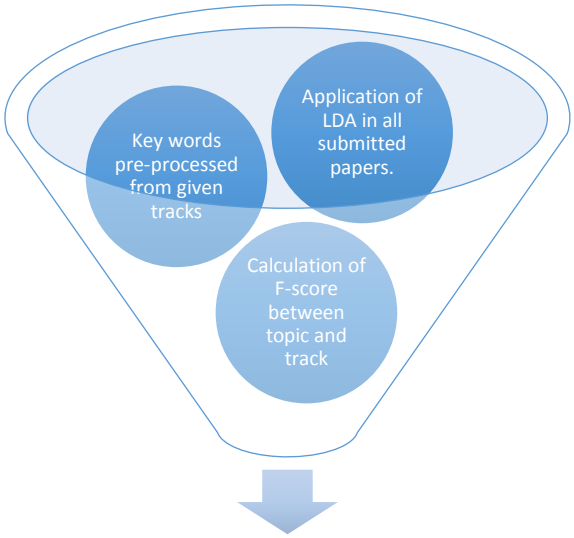


Figure 5: The Process of LDA and Topic-Track Matching

First, implement pre-processing step in both papers and conference tracks. Next, retrieve latent topics with a set of words in descending order of probability $P(w|z)$. And then match every topic with the highest F-score track by computing corresponding overlapping words, where Wk represents the key words from a track and Wt represents top-ranking words from a topic. Finally, calculate Precision, Recall and F-score in sequence:

$$\text{Precision} = \frac{|W_k \cap W_t|}{|W_k|} \quad (2)$$

$$\text{Recall} = \frac{|W_k \cap W_t|}{|W_t|} \quad (3)$$

$$\text{F-score} = \frac{|W_k \cap W_t|}{\frac{|W_k|+|W_t|}{2}} \quad (4)$$

With further implementation of matching system [1], analytics data sets can be obtained.

Then we can gain results and make observations. Above all, in order to check the correlation degree between expert-defined tracks and latent topics, we may implement word clouds (Figure 6) to visualize each topic's concentrations:

Figure 6: A Set of Word Clouds Representing Concentrations of Different topics; The bigger the words are, the more popular the concentrations are.

We can figure out the dominating words in each topic from the figures above (e.g. algorithm for the last figure), and then by comparison of the key words of expert-defined tracks we can judge whether the expert-defined tracks are comprehensive enough to cover the latent topics.

If we want to dig further on specific topics to see whether there is predictable trend in certain field. We may implement several data visualization steps to gain concrete answer. Here is a set of data visualization graphs (Figure 7) which represent a topic' s concentration varying from 2000-2009:

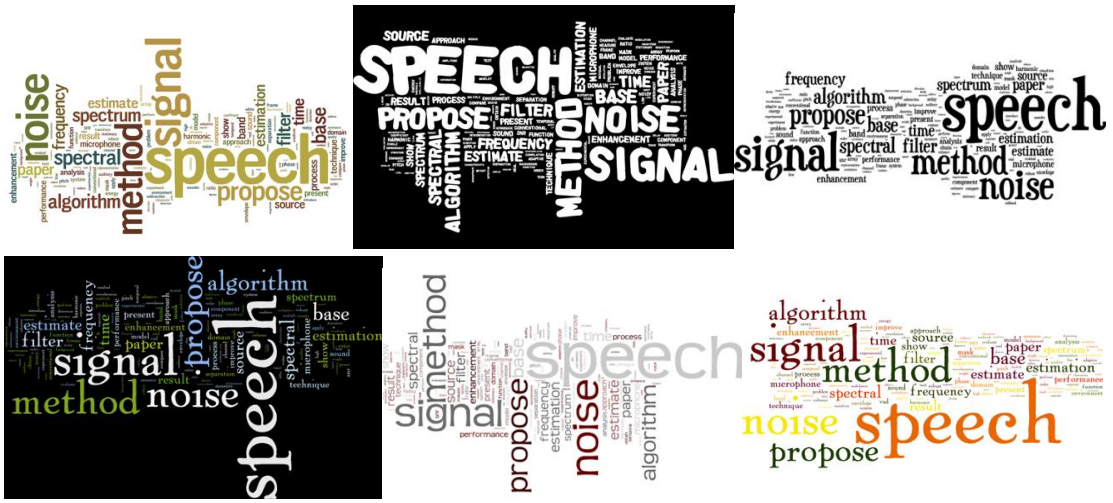


Figure 7: A Set of Words Clouds Representing Concentrations of a Topic 2000-2009

After observations we can figure out that the concentrations of this topic are mainly in aspects of “Speech”, “Noise”, “Signal”, “Method”, etc. If we want to figure out the trends in detail about the dominating concentrations, heat maps (Figure 8) can offer practical approach as it can easily spot the dominating words among over 1000 words by color difference:

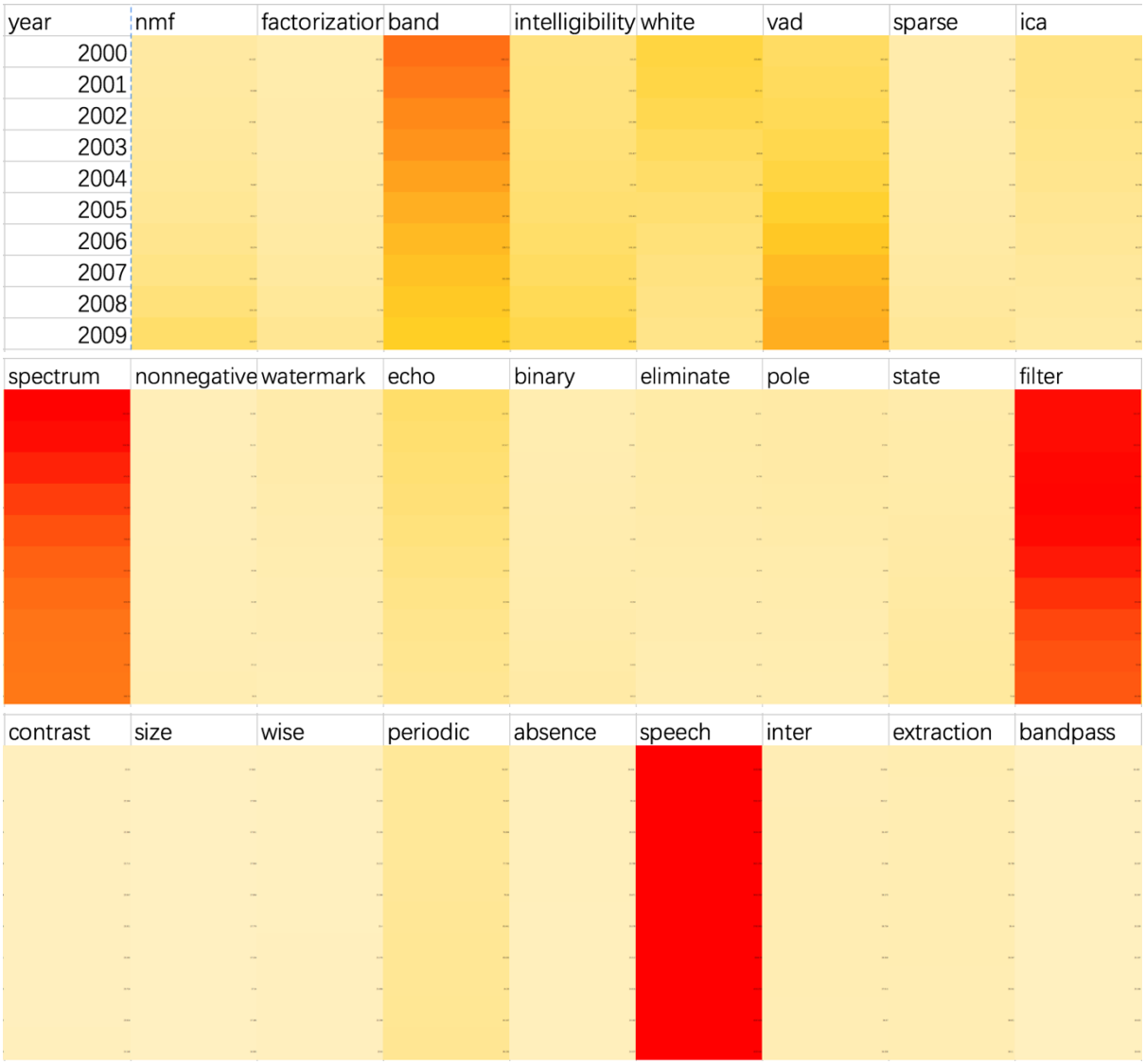


Figure 8: By implementing heat map to realize data visualization, we can specify the important data immediately.

And if we want to know more about the dominating words' trends in order to make further evidence-based predictions about the popular tracks, which can help conference organizers define more explicit and comprehensive track descriptions in the future, we may implement several charts (Figure 9) representing the dominating words.

Figure 9: Several Representations of Dominating Words

By observations we can figure out that "Speech" keeps rising and has the highest value, so further interest should be paid; "Signal" has high value but keeps decreasing, so the concentration for next year's conference tracks regarding "Signal" may reduce. Each topic can be analyzed in this method and evidence-based predictions can be extracted from data ultimately.

5. References

[1] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[2] Fan Yang, Jinfeng Li, and James Cheng. Husky: Towards a more efficient and expressive distributed computing framework. *PVLDB*, 9(5):420–431, 2016.

Summer Research Internship
Franklin Lee
SID: 1155077143