



北京师范大学 联合国国际学院  
香港浸会大学

BEIJING NORMAL UNIVERSITY · HONG KONG BAPTIST UNIVERSITY  
UNITED INTERNATIONAL COLLEGE

## **Text Search Based on Spark for Children's Books**

[Group 5]

[Frank 谢欣言 1930026136]

[Tony 焦海旭 1930026057]

[Connor 徐贺 1930026139]

A report submitted to

Dr. Canhao Xu

for

DS 3003 Data Processing Workshop II

Data Science (DS)

Division of Science and Technology (DST)

[2021.12.19]

## Contents

|   |           |
|---|-----------|
| <b>1. Introduction .....</b>                                  | <b>3</b>  |
| <b>2. Text Files Collection .....</b>                         | <b>3</b>  |
| <b>3. Indexing and Ranking .....</b>                          | <b>10</b> |
| <b>4. Web Server and Interface of the Search Engine .....</b> | <b>30</b> |
| <b>5. Conclusion and Reflection .....</b>                     | <b>33</b> |
| <b>6. Reference .....</b>                                     | <b>34</b> |

## 1. Introduction

### 1) Goal of the project:

The goal of this project is to implement a system that can retrieve text contents related to the query and order them according to relevance which are stored at different nodes in a Hadoop/HDFS cluster.

To achieve that, we worked on text files collection, indexing and ranking, and finally developed a web server with an interface of the search engine.

### 2) Background and our motivation:

Nowadays, there is a huge amount of book resources on the Internet, and for readers, especially for children, it is extremely difficult for them to get the book or content, they want to read.

Therefore, we use the HDFS and Spark analysis and computing methods to build a search engine which allows them to find the books containing the content they want to read quickly. By accessing the web server, they can also download and read these books from the site immediately.

## 2. Text Files Collection

### 1) Source of our file set:

Our text file resources are collected from Kaggle, which are children's books.

The URL is: <https://www.kaggle.com/amoghjrules/babi-childrens-books-facebook-ai>.

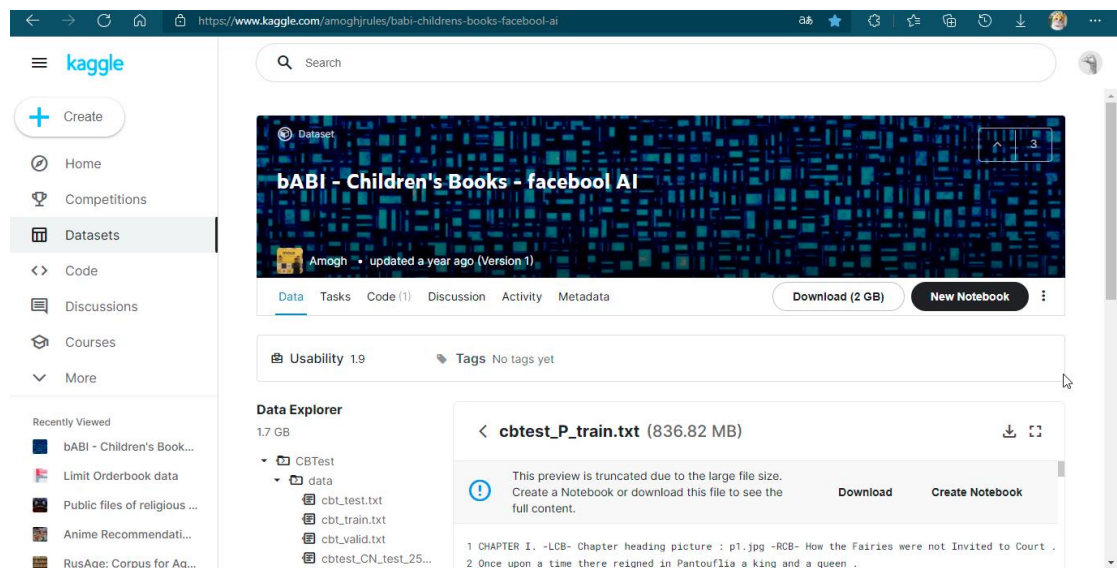


Figure 1 Source of Our File Set

## 2) Data preprocessing (Splitting):

Our file set is consisted of 2 files which are over 800MB and 200MB separately. But it was not easy to find the specific part we want to read, thus, because we found that about every 5000 lines were relevant to one book, we split these large files into 2000 small files so that the users could only download the one they need.



```
# -*- coding:utf-8 -*-
from datetime import datetime

def Main():
    source_dir = r'C:\Users\lenovo\Desktop\[DPW2] Group Project\BigText\cbtest_P_train.txt'
    target_dir = 'split/' #to store the txts after splitting

    #counter
    flag = 0

    #name index
    name = 1

    #to store the data
    dataList = []

    print("Start splitting...")
    print(datetime.now().strftime('%Y-%m-%d %H:%M:%S')) #start time

    with open(source_dir, 'r') as f_source:
        for line in f_source:
            flag+=1
            dataList.append(line)
            if flag == 5000: #split each 5000 lines into a file
                with open(target_dir+"ChildrenBook_"+str(name)+".txt", 'w+') as f_target: #use name counter to name the file
                    for data in dataList:
                        f_target.write(data)
                    name+=1
                    flag = 0
                    dataList = []

    #process the remaining data which are less than 5000 lines:
    with open(target_dir+"ChildrenBook_"+str(name)+".txt", 'w+') as f_target:
        for data in dataList:
            f_target.write(data)

    print("Finish splitting!")
    print(datetime.now().strftime('%Y-%m-%d %H:%M:%S')) #end time

if __name__ == "__main__":
    Main()
```

Start splitting...  
2021-12-03 10:17:34  
Finish splitting!  
2021-12-03 10:18:12

Figure 2 Codes of Data Preprocessing (Splitting)

| 名称                  | 日期时间            | 类型   | 大小       |
|---------------------|-----------------|------|----------|
| ChildrenBook_1.txt  | 2021/12/2 22:23 | 文本文档 | 1,514 KB |
| ChildrenBook_2.txt  | 2021/12/2 22:23 | 文本文档 | 1,563 KB |
| ChildrenBook_3.txt  | 2021/12/2 22:23 | 文本文档 | 1,567 KB |
| ChildrenBook_4.txt  | 2021/12/2 22:23 | 文本文档 | 1,584 KB |
| ChildrenBook_5.txt  | 2021/12/2 22:23 | 文本文档 | 1,447 KB |
| ChildrenBook_6.txt  | 2021/12/2 22:23 | 文本文档 | 1,364 KB |
| ChildrenBook_7.txt  | 2021/12/2 22:23 | 文本文档 | 1,744 KB |
| ChildrenBook_8.txt  | 2021/12/2 22:23 | 文本文档 | 1,956 KB |
| ChildrenBook_9.txt  | 2021/12/2 22:23 | 文本文档 | 2,062 KB |
| ChildrenBook_10.txt | 2021/12/2 22:23 | 文本文档 | 2,068 KB |
| ChildrenBook_11.txt | 2021/12/2 22:23 | 文本文档 | 1,743 KB |
| ChildrenBook_12.txt | 2021/12/2 22:23 | 文本文档 | 1,851 KB |
| ChildrenBook_13.txt | 2021/12/2 22:23 | 文本文档 | 2,071 KB |
| ChildrenBook_14.txt | 2021/12/2 22:23 | 文本文档 | 2,103 KB |
| ChildrenBook_15.txt | 2021/12/2 22:23 | 文本文档 | 2,143 KB |
| ChildrenBook_16.txt | 2021/12/2 22:23 | 文本文档 | 2,226 KB |
| ChildrenBook_17.txt | 2021/12/2 22:23 | 文本文档 | 1,970 KB |
| ChildrenBook_18.txt | 2021/12/2 22:23 | 文本文档 | 1,919 KB |

Figure 3 File Set after Splitting

That's also why we didn't remove the Stopwords, because we wanted to provide the users with readable results but not only for the convenience of our research.

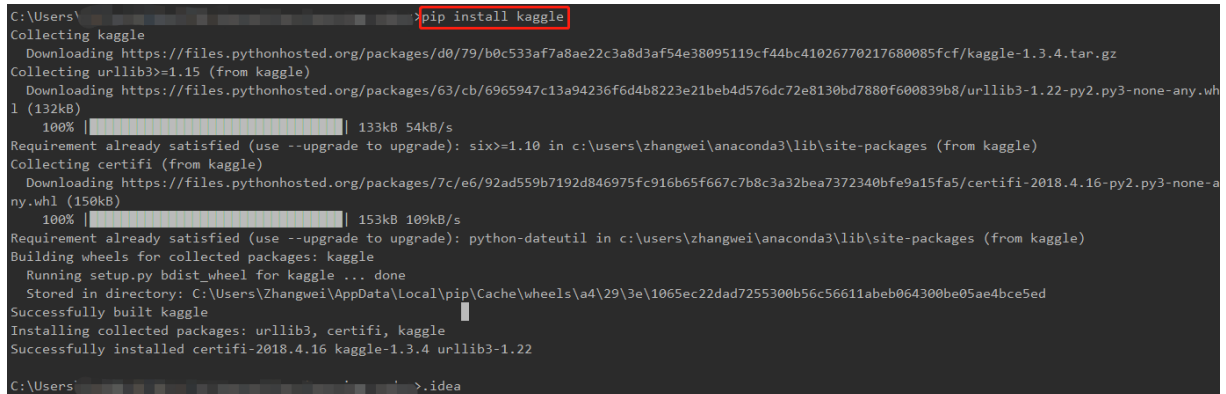
However, this also made our data set have lots of characters which are not in English and some redundant words in the indexing and ranking step. They might mislead the calculated results, which was really confusing and not “good” enough.

### 3) **Discuss:** Methods to speed up the data collection process:

In our collection process, we found that, if there are many small files on the internet, we can use web crawler to download them one by one quickly.

If the data set is in just one or few big files, then we can use the split method after downloading, which is also very time-saving.

If we want to download files from Kaggle, we can even use a Kaggle package in python to download automatically and efficiently:



```
C:\Users\...>pip install kaggle
Collecting kaggle
  Downloading https://files.pythonhosted.org/packages/d0/79/b0c533af7a8ae22c3a8d3af54e38095119cf44bc41026770217680085fcf/kaggle-1.3.4.tar.gz
Collecting urllib3>=1.15 (from kaggle)
  Downloading https://files.pythonhosted.org/packages/63/cb/6965947c13a94236f6d4b8223e21beb4d576dc72e8130bd7880f600839b8/urllib3-1.22-py2.py3-none-any.whl (132kB)
100% |#####| 133kB 54kB/s
Requirement already satisfied (use --upgrade to upgrade): six>=1.10 in c:\users\zhangwei\anaconda3\lib\site-packages (from kaggle)
Collecting certifi (from kaggle)
  Downloading https://files.pythonhosted.org/packages/7c/e6/92ad559b7192d846975fc916b65f667c7b8c3a32bea7372340bfe9a15fa5/certifi-2018.4.16-py2.py3-none-any.whl (150kB)
100% |#####| 153kB 109kB/s
Requirement already satisfied (use --upgrade to upgrade): python-dateutil in c:\users\zhangwei\anaconda3\lib\site-packages (from kaggle)
Building wheels for collected packages: kaggle
  Running setup.py bdist_wheel for kaggle ... done
  Stored in directory: C:\Users\Zhangwei\AppData\Local\pip\Cache\wheels\4\29\3e\1065ec22dad7255300b56c56611abeb064300be05ae4bce5ed
Successfully built kaggle
Installing collected packages: urllib3, certifi, kaggle
Successfully installed certifi-2018.4.16 kaggle-1.3.4 urllib3-1.22
C:\Users\...>.idea
```

Figure 4 Kaggle Package in Python

### 4) **Small file set for test:**

Due to that it's really time-consuming to process the big data set, we also prepared a 100MB small set chosen from the original large set:

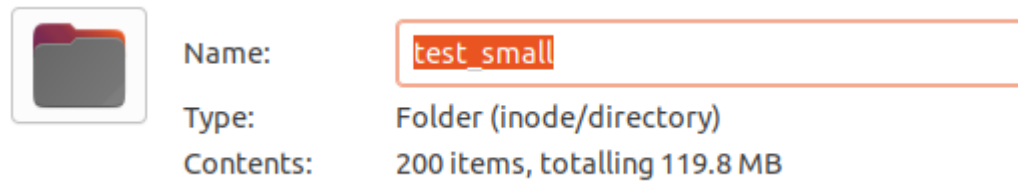


Figure 5 Small File Set

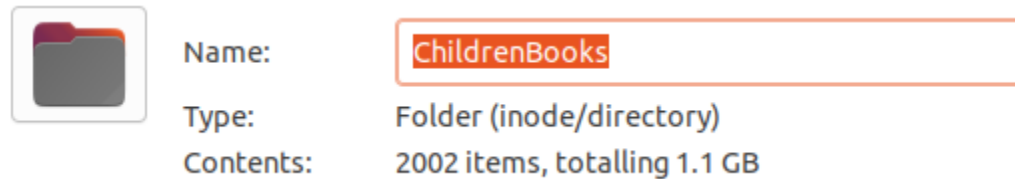


Figure 6 Large File Set

### 5) Put the data set to HDFS:

To use the data for calculating and as the resource of the web server, we need to put them to the HDFS.

Firstly, we put them on the Namenode:

```
root@p930026136master:~/Downloads# docker cp ~/Downloads/ChildrenBooks 73bb8fbd6b25:ChildrenBooks
root@p930026136master:~/Downloads# docker exec -it namenode bash
root@73bb8fbd6b25:/# ls
ChildrenBooks KEYS boot entrypoint.sh hadoop home lib64 mnt proc run sbin sys usr
ChildrenBooks.zip bin dev etc hadoop-data lib media opt root run.sh srv tmp var
```

Figure 7 Put on the Namenode 1

```
root@73bb8fbd6b25:/ChildrenBooks# ls
ChildrenBook_1.txt ChildrenBook_20.txt ChildrenBook_300.txt ChildrenBook_401.txt ChildrenBook_502.txt ChildrenBook_603.txt
ChildrenBook_10.txt ChildrenBook_200.txt ChildrenBook_301.txt ChildrenBook_402.txt ChildrenBook_503.txt ChildrenBook_604.txt
ChildrenBook_100.txt ChildrenBook_201.txt ChildrenBook_302.txt ChildrenBook_403.txt ChildrenBook_504.txt ChildrenBook_605.txt
ChildrenBook_101.txt ChildrenBook_202.txt ChildrenBook_303.txt ChildrenBook_404.txt ChildrenBook_505.txt ChildrenBook_606.txt
ChildrenBook_102.txt ChildrenBook_203.txt ChildrenBook_304.txt ChildrenBook_405.txt ChildrenBook_506.txt ChildrenBook_607.txt
ChildrenBook_103.txt ChildrenBook_204.txt ChildrenBook_305.txt ChildrenBook_406.txt ChildrenBook_507.txt ChildrenBook_608.txt
ChildrenBook_104.txt ChildrenBook_205.txt ChildrenBook_306.txt ChildrenBook_407.txt ChildrenBook_508.txt ChildrenBook_609.txt
ChildrenBook_105.txt ChildrenBook_206.txt ChildrenBook_307.txt ChildrenBook_408.txt ChildrenBook_509.txt ChildrenBook_610.txt
ChildrenBook_106.txt ChildrenBook_207.txt ChildrenBook_308.txt ChildrenBook_409.txt ChildrenBook_510.txt ChildrenBook_611.txt
ChildrenBook_107.txt ChildrenBook_208.txt ChildrenBook_309.txt ChildrenBook_410.txt ChildrenBook_511.txt ChildrenBook_612.txt
ChildrenBook_108.txt ChildrenBook_209.txt ChildrenBook_310.txt ChildrenBook_411.txt ChildrenBook_512.txt ChildrenBook_613.txt
ChildrenBook_109.txt ChildrenBook_210.txt ChildrenBook_311.txt ChildrenBook_412.txt ChildrenBook_513.txt ChildrenBook_614.txt
ChildrenBook_11.txt ChildrenBook_211.txt ChildrenBook_312.txt ChildrenBook_413.txt ChildrenBook_514.txt ChildrenBook_615.txt
ChildrenBook_110.txt ChildrenBook_212.txt ChildrenBook_313.txt ChildrenBook_414.txt ChildrenBook_515.txt ChildrenBook_616.txt
ChildrenBook_111.txt ChildrenBook_213.txt ChildrenBook_314.txt ChildrenBook_415.txt ChildrenBook_516.txt ChildrenBook_617.txt
ChildrenBook_112.txt ChildrenBook_214.txt ChildrenBook_315.txt ChildrenBook_416.txt ChildrenBook_517.txt ChildrenBook_618.txt
ChildrenBook_113.txt ChildrenBook_215.txt ChildrenBook_316.txt ChildrenBook_417.txt ChildrenBook_518.txt ChildrenBook_619.txt
ChildrenBook_114.txt ChildrenBook_216.txt ChildrenBook_317.txt ChildrenBook_418.txt ChildrenBook_519.txt ChildrenBook_620.txt
ChildrenBook_115.txt ChildrenBook_217.txt ChildrenBook_318.txt ChildrenBook_419.txt ChildrenBook_520.txt ChildrenBook_621.txt
ChildrenBook_116.txt ChildrenBook_218.txt ChildrenBook_319.txt ChildrenBook_420.txt ChildrenBook_521.txt ChildrenBook_622.txt
ChildrenBook_117.txt ChildrenBook_219.txt ChildrenBook_320.txt ChildrenBook_421.txt ChildrenBook_522.txt ChildrenBook_623.txt
ChildrenBook_118.txt ChildrenBook_220.txt ChildrenBook_321.txt ChildrenBook_422.txt ChildrenBook_523.txt ChildrenBook_624.txt
ChildrenBook_119.txt ChildrenBook_221.txt ChildrenBook_322.txt ChildrenBook_423.txt ChildrenBook_524.txt ChildrenBook_625.txt
ChildrenBook_12.txt ChildrenBook_222.txt ChildrenBook_323.txt ChildrenBook_424.txt ChildrenBook_525.txt ChildrenBook_626.txt
ChildrenBook_120.txt ChildrenBook_223.txt ChildrenBook_324.txt ChildrenBook_425.txt ChildrenBook_526.txt ChildrenBook_627.txt
ChildrenBook_121.txt ChildrenBook_224.txt ChildrenBook_325.txt ChildrenBook_426.txt ChildrenBook_527.txt ChildrenBook_628.txt
ChildrenBook_122.txt ChildrenBook_225.txt ChildrenBook_326.txt ChildrenBook_427.txt ChildrenBook_528.txt ChildrenBook_629.txt
ChildrenBook_123.txt ChildrenBook_226.txt ChildrenBook_327.txt ChildrenBook_428.txt ChildrenBook_529.txt ChildrenBook_630.txt
ChildrenBook_124.txt ChildrenBook_227.txt ChildrenBook_328.txt ChildrenBook_429.txt ChildrenBook_530.txt ChildrenBook_631.txt
```

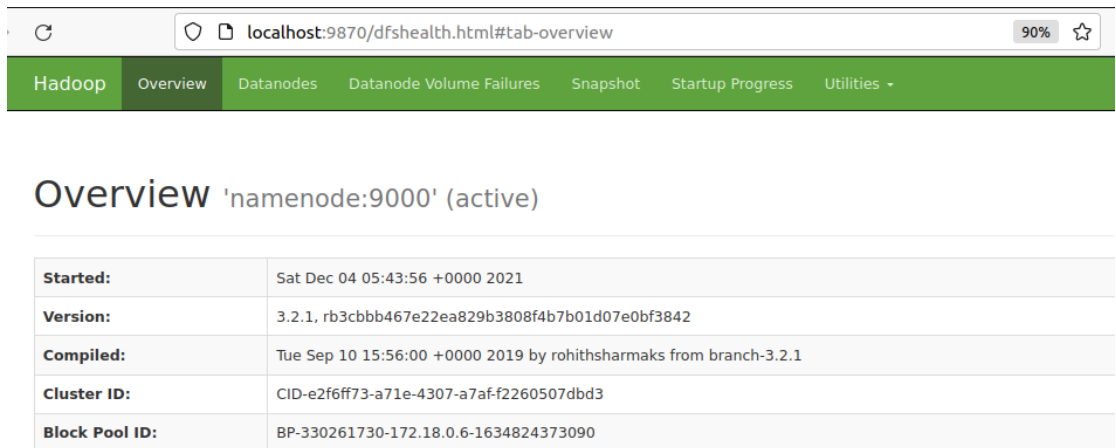
Figure 8 Put on the Namenode 2

Then, we entered the Namenode and move the files to the HDFS:

```
root@73bb8fbd6b25:/# hadoop fs -mkdir -p ChildrenBooks
root@73bb8fbd6b25:/# hdfs dfs -put ./ChildrenBooks/* ChildrenBooks
2021-12-02 14:57:21,794 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2021-12-02 14:57:22,344 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2021-12-02 14:57:22,382 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2021-12-02 14:57:22,405 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2021-12-02 14:57:22,843 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```

Figure 9 Move to the HDFS

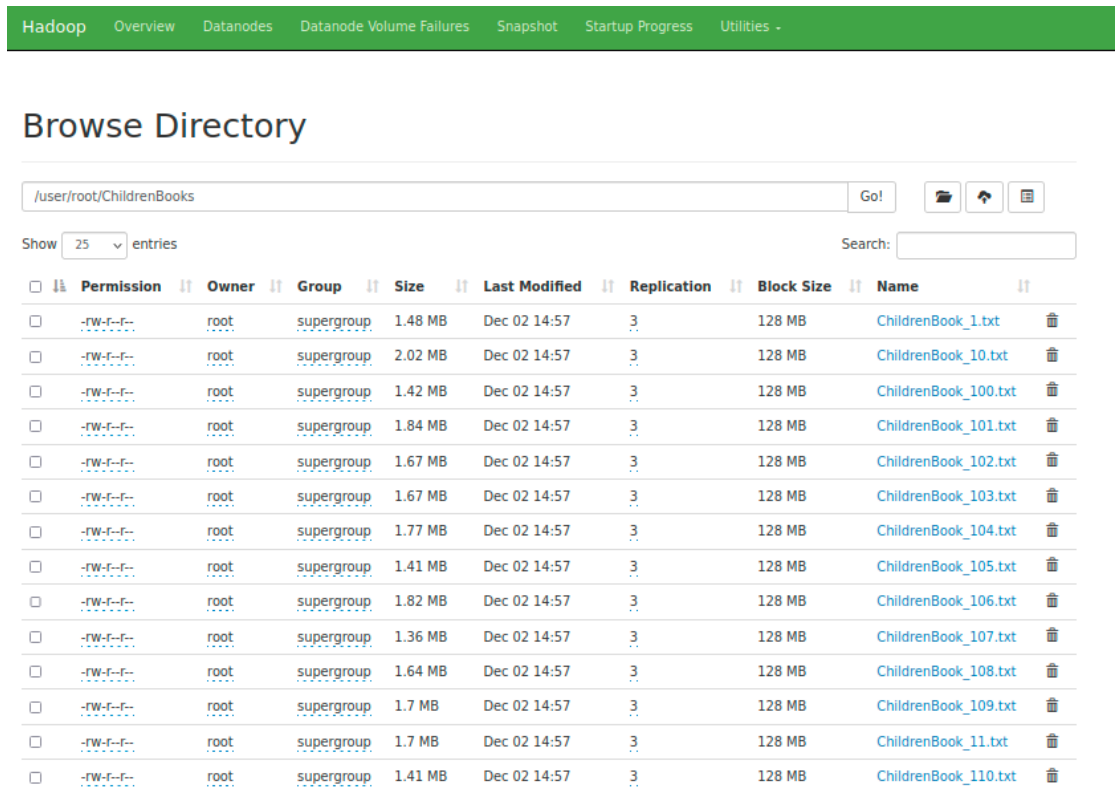
Finally, the data is on our HDFS cluster:



The screenshot shows the Hadoop Overview page for the namenode:9000. The page has a green header with navigation tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The main content area is titled "Overview 'namenode:9000' (active)". Below the title is a table with the following information:

|                       |  |
|-----------------------|--|
| <b>Started:</b>       | Sat Dec 04 05:43:56 +0000 2021                                     |
| <b>Version:</b>       | 3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842                   |
| <b>Compiled:</b>      | Tue Sep 10 15:56:00 +0000 2019 by rohithsharmaks from branch-3.2.1 |
| <b>Cluster ID:</b>    | CID-e2f6ff73-a71e-4307-a7af-f2260507dbd3                           |
| <b>Block Pool ID:</b> | BP-330261730-172.18.0.6-1634824373090                              |

Figure 10 The HDFS Cluster



The screenshot shows the Hadoop Browse Directory page for the path /user/root/ChildrenBooks. The page has a green header with navigation tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The main content area is titled "Browse Directory". Below the title is a search bar with the text "/user/root/ChildrenBooks" and a "Go!" button. Below the search bar is a table with the following columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. The table lists 11 files in the ChildrenBooks directory, all owned by root and supergroup, with sizes ranging from 1.41 MB to 2.02 MB and last modified dates of Dec 02 14:57. The files are named ChildrenBook\_1.txt through ChildrenBook\_110.txt.

| Permission | Owner | Group      | Size    | Last Modified | Replication | Block Size | Name                 |
|------------|-------|------------|---------|---------------|-------------|------------|----------------------|
| -rw-r--r-- | root  | supergroup | 1.48 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_1.txt   |
| -rw-r--r-- | root  | supergroup | 2.02 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_10.txt  |
| -rw-r--r-- | root  | supergroup | 1.42 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_100.txt |
| -rw-r--r-- | root  | supergroup | 1.84 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_101.txt |
| -rw-r--r-- | root  | supergroup | 1.67 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_102.txt |
| -rw-r--r-- | root  | supergroup | 1.67 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_103.txt |
| -rw-r--r-- | root  | supergroup | 1.77 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_104.txt |
| -rw-r--r-- | root  | supergroup | 1.41 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_105.txt |
| -rw-r--r-- | root  | supergroup | 1.82 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_106.txt |
| -rw-r--r-- | root  | supergroup | 1.36 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_107.txt |
| -rw-r--r-- | root  | supergroup | 1.64 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_108.txt |
| -rw-r--r-- | root  | supergroup | 1.7 MB  | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_109.txt |
| -rw-r--r-- | root  | supergroup | 1.7 MB  | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_11.txt  |
| -rw-r--r-- | root  | supergroup | 1.41 MB | Dec 02 14:57  | 3           | 128 MB     | ChildrenBook_110.txt |

Figure 11 Data Set on the HDFS



## 6) **Discuss:** How to update the contents and results of our system dynamically?

In the research process, we thought about that: how to design the system if it is required to collect and analyze text data from websites or social media contents in real time or update the results dynamically as time goes by, just like Google?

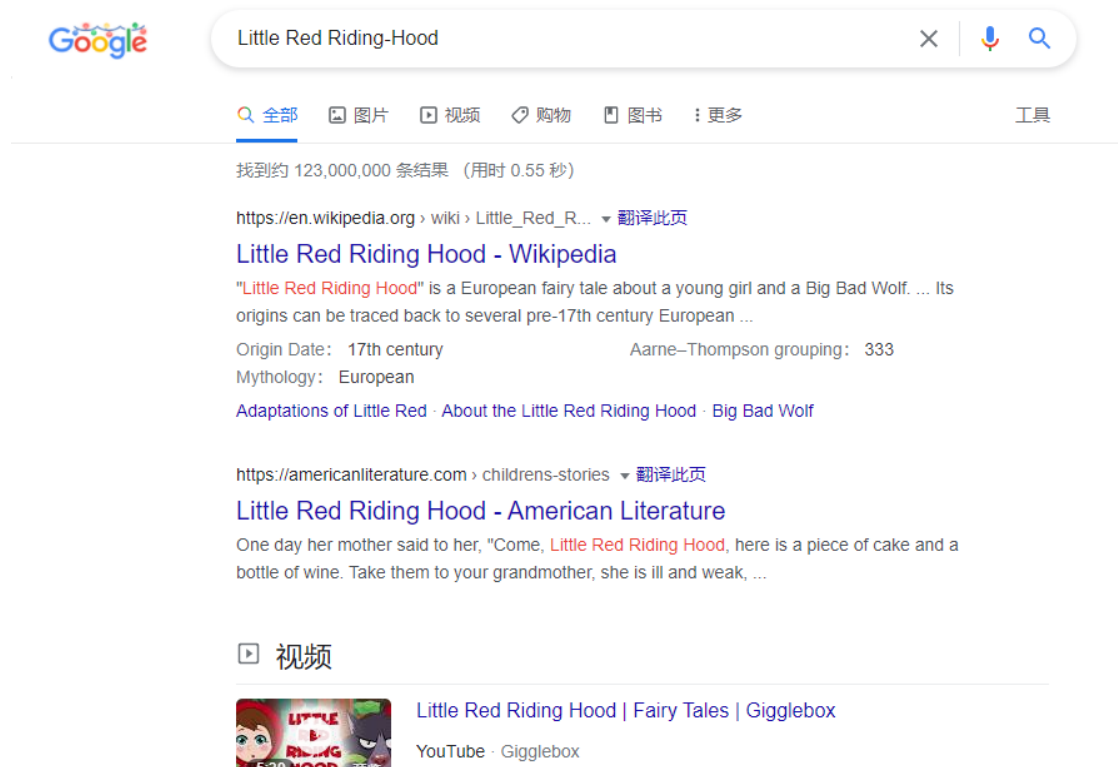


Figure 12 Dynamic Update on Google

If we got our data from websites or social media, we can firstly crawl the new data from the source page, then fetch and update the new content every hour (or 24h for slow updates), reconstruct the Inverted Index and recalculate the sorting to update our search results. In that case, the searched results on the web interface are always the newest periodically.

However, for our system, we need to add new files or update files on HDFS manually, instead of fetching it automatically and then recalculating the index. Due to that children's books need to be found, read and filtered by people, that's also one point that is not "good" enough.



### 3. Indexing and Ranking

#### 1) Indexing:

##### ■ On single node and no MapReduce:

Firstly, we implemented the no MapReduce codes by ourselves using two dictionaries to do that, one stores key - file name, and value - content, and another: key - word, value - the file it appears.

```
#create a dict with key: file name, and value: content
content = []
docu_dict = {}
for i in range(11):
    if i == 0:
        continue
    with open('./ChildrenBooks/ChildrenBook_'+str(i)+'.txt') as f:
        content = f.read()
        docu_dict['ChildrenBook_'+str(i)+'.txt'] = content
#print(docu_dict)

from time import *

begin_time = time() #to get the running time

#to split all words into tokens:
words = []

for i in docu_dict.values():
    cut = i.split()
    words.extend(cut)

set_words = set(words)
#print(set_all_words)

#construct the inverted index, when traversing a file,
#if a word is in the dict, append the file name to its value,
#else, append that word as a new key
inverted_index = dict() #use a dict to store: key - word, value - the file it appears
for b in set_words:
    temp = []

    for j in docu_dict.keys():
        field = docu_dict[j]

        split_field = field.split()

        if b in split_field:
            temp.append(j)
        inverted_index[b] = temp

end_time = time()
run_time = end_time - begin_time #to get the running time
print(int(run_time))
print(inverted_index)
```

Figure 13 Codes of the InvIndex on Single node and No MapReduce

And the results of Inverted Index are as following:

```
{'Where': ['ChildrenBook_2.txt', 'ChildrenBook_3.txt', 'ChildrenBook_4.txt', 'ChildrenBook_6.txt', 'ChildrenBook_7.txt', 'ChildrenBook_8.txt'], 'presenting': ['ChildrenBook_2.txt', 'ChildrenBook_5.txt', 'ChildrenBook_7.txt'], 'rat-catcher': ['ChildrenBook_1.txt', 'ChildrenBook_3.txt'], 'Will': ['ChildrenBook_2.txt', 'ChildrenBook_4.txt', 'ChildrenBook_5.txt', 'ChildrenBook_9.txt', 'ChildrenBook_10.txt'], 'happen': ['ChildrenBook_2.txt', 'ChildrenBook_5.txt', 'ChildrenBook_6.txt', 'ChildrenBook_7.txt', 'ChildrenBook_9.txt', 'ChildrenBook_10.txt'], 'Majesty|age|book|mother|notion|princess|sentiment|someone|tire|to-morrow': ['ChildrenBook_6.txt'], 'also': ['ChildrenBook_1.txt', 'ChildrenBook_2.txt', 'ChildrenBook_3.txt', 'ChildrenBook_4.txt', 'ChildrenBook_5.txt', 'ChildrenBook_6.txt', 'ChildrenBook_7.txt', 'ChildrenBook_8.txt', 'ChildrenBook_9.txt', 'ChildrenBook_10.txt'], 'miss': ['ChildrenBook_2.txt', 'ChildrenBook_5.txt', 'ChildrenBook_6.txt'], 'marked': ['ChildrenBook_1.txt', 'ChildrenBook_4.txt'], 'live': ['ChildrenBook_3.txt', 'ChildrenBook_5.txt', 'ChildrenBook_6.txt', 'ChildrenBook_7.txt'], 'Page|Water|bottle|flame|heaven|light|love|place|prince|youth': ['ChildrenBook_6.txt'], 'XXXXX': ['ChildrenBook_1.txt', 'ChildrenBook_2.txt', 'ChildrenBook_3.txt', 'ChildrenBook_4.txt', 'ChildrenBook_5.txt', 'ChildrenBook_6.txt', 'ChildrenBook_7.txt', 'ChildrenBook_8.txt', 'ChildrenBook_9.txt', 'ChildrenBook_10.txt'], 'EAST': ['ChildrenBook_10.txt'], 'altar': ['ChildrenBook_3.txt', 'ChildrenBook_6.txt'], 'soil': ['ChildrenBook_9.txt'], 'twentieth': ['ChildrenBook_10.txt'], 'winning': ['ChildrenBook_2.txt', 'ChildrenBook_5.txt', 'ChildrenBook_8.txt'], 'defeated': ['ChildrenBook_8.txt'], 'Majesty|blade|bow|fact|fairy|gentlemen|objects|pleasure|scene|things': ['ChildrenBook_7.txt'], 'doublez': ['ChildrenBook_7.txt'], 'cruel': ['ChildrenBook_1.txt', 'ChildrenBook_2.txt', 'ChildrenBook_3.txt', 'ChildrenBook_4.txt', 'ChildrenBook_5.txt', 'ChildrenBook_6.txt'], 'mice': ['ChildrenBook_3.txt', 'ChildrenBook_6.txt', 'ChildrenBook_9.txt'], 'ChildrenBook_10.txt'], 'sixty-three': ['ChildrenBook_1.txt', 'ChildrenBook_4.txt'], 'hastened': ['ChildrenBook_7.txt'], 'skin': ['ChildrenBook_1.txt', 'ChildrenBook_3.txt'], 'showers': ['ChildrenBook_8.txt'], 'Make': ['ChildrenBook_9.txt'], 'smooth': ['ChildrenBook_2.txt', 'ChildrenBook_4.txt'], 'alighted': ['ChildrenBook_8.txt'], 'air|conditions|duty|hand|king|kings|prince|sons|space|word': ['ChildrenBook_1.txt', 'ChildrenBook_2.txt', 'ChildrenBook_3.txt', 'ChildrenBook_4.txt', 'ChildrenBook_5.txt', 'ChildrenBook_6.txt', 'ChildrenBook_7.txt', 'ChildrenBook_8.txt', 'ChildrenBook_9.txt', 'ChildrenBook_10.txt']}]
```

Figure 14 Results of the InvIndex on Single node and No MapReduce

For small files, the running times are as following:

For 2 files, it took 16s:

```
print(int(run_time))
print(inverted_index)

16
```

Figure 15 2 Files

For 3 files, it took 43s:

```
print(int(run_time))
print(inverted_index)

43
```

Figure 16 3 Files

For 10 files, it took 13.5min:

```
print(int(run_time))
print(inverted_index)

810
```

Figure 17 10 Files

However, we found that if it processes more files, it will be slower and even crash for large files (1GB):

```
Kernel Restarting

The kernel appears to have died. It will restart automatically.
```

Figure 18 For 1GB File Set

In that case, it can be found that indexing using single node and no MapReduce is extremely slow.

- On single node and using MapReduce.



Then, we used the Hadoop MapReduce and HDFS to help us calculate the Inverted Index:

```
#Inverted Index using mapreduce but not using cluster:
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession
import os, re
from time import *

begin_time = time()    #to get the running time

#spark = SparkSession.builder.master('spark://dpw2frankm:7077').appName('SparkInvIndex').getOrCreate()
#sc = spark.sparkContext
sc = SparkContext.getOrCreate(SparkConf())

inverted_index = sc.wholeTextFiles('hdfs://localhost:9000/user/root/test_small/')\
    .flatMap(lambda x: [(os.path.basename(x[0]).split(".")[0],1),1] \
        for i in re.split('\W', x[1]))\
    .reduceByKey(lambda a, b: a + b)\
    .map(lambda x: (x[0][1],(x[0][0],x[1])))

#we skip the collect and print step because it's too slow:
output = inverted_index.collect()
for i in range(inverted_index.count()):

    print(output[i])
    #if i>10: #limit number to print
    #    break

end_time = time()    #to get the running time
run_time = end_time - begin_time
print('Running time: ', run_time)
```

Figure 19 Codes of the InvIndex using Mapreduce

The results are as following:

```
('presently', ('ChildrenBook_187', 29))
('foul', ('ChildrenBook_187', 31))
('trussing', ('ChildrenBook_187', 19))
('nine', ('ChildrenBook_187', 19))
('pieces', ('ChildrenBook_187', 28))
('parcels', ('ChildrenBook_187', 24))
('string', ('ChildrenBook_187', 44))
('tie', ('ChildrenBook_187', 20))
('parcel', ('ChildrenBook_187', 25))
('fairness', ('ChildrenBook_187', 24))
('belay', ('ChildrenBook_187', 20))
('violence', ('ChildrenBook_187', 20))
('His', ('ChildrenBook_187', 54))
('curled', ('ChildrenBook_187', 18))
('malicious', ('ChildrenBook_187', 18))
('triumph', ('ChildrenBook_187', 30))
('sweating', ('ChildrenBook_187', 20))
('lad', ('ChildrenBook_187', 23))
```

Figure 20 Codes of the InvIndex using MapReduce

For small files (100MB), it took 52s:



```
end_time = time()      #to get the running time
run_time = end_time - begin_time
print('Running time: ', run_time)

('twice', ('ChildrenBook_99', 10))
('crawl', ('ChildrenBook_99', 8))
('How', ('ChildrenBook_99', 8))
('shrunk', ('ChildrenBook_99', 8))
('grew', ('ChildrenBook_99', 8))
('bounding', ('ChildrenBook_99', 3))
('swam', ('ChildrenBook_99', 2))
('given', ('ChildrenBook_99', 2))
('staying', ('ChildrenBook_99', 2))
('colours', ('ChildrenBook_99', 2))
('upper', ('ChildrenBook_99', 2))
('difficult', ('ChildrenBook_99', 2))
('remarked', ('ChildrenBook_99', 2))
('flies', ('ChildrenBook_99', 1))
('coming', ('ChildrenBook_99', 1))
('eaten', ('ChildrenBook_99', 1))
('moving', ('ChildrenBook_99', 1))
('excited', ('ChildrenBook_99', 1))
Running time: 52.0172278881073
```

Figure 21 Results for Small Files

And for large files (1GB), it took 12min:

```
('stealing', ('ChildrenBook_1022', 43))
('listen', ('ChildrenBook_1022', 97))
('Elves', ('ChildrenBook_1022', 338))
('by', ('ChildrenBook_1022', 134))
('hand', ('ChildrenBook_1022', 40))
('sadly', ('ChildrenBook_1022', 97))
('asked', ('ChildrenBook_1022', 40))
('whence', ('ChildrenBook_1022', 40))
('We', ('ChildrenBook_1022', 151))
('Land', ('ChildrenBook_1022', 315))
('worthy', ('ChildrenBook_1022', 46))
('fair', ('ChildrenBook_1022', 376))
('forth', ('ChildrenBook_1022', 203))
('Look', ('ChildrenBook_1022', 48))
('withered', ('ChildrenBook_1022', 138))
('crowns', ('ChildrenBook_1022', 254))
('gone', ('ChildrenBook_1022', 90))
('lead', ('ChildrenBook_1022', 48))
('lives', ('ChildrenBook_1022', 48))
('stay', ('ChildrenBook_1022', 171))

print('Running time: ', run_time)
Running time: 716.1954402923584
```

Figure 22 Results for Large Files

Thus, we can find that it's much faster than no Mapreduce.

#### ■ Use MapReduce and Spark cluster:

Finally, we use MapReduce and Spark Cluster, which is consisted of 1 master 2 workers.

The information of our Spark cluster is as following:

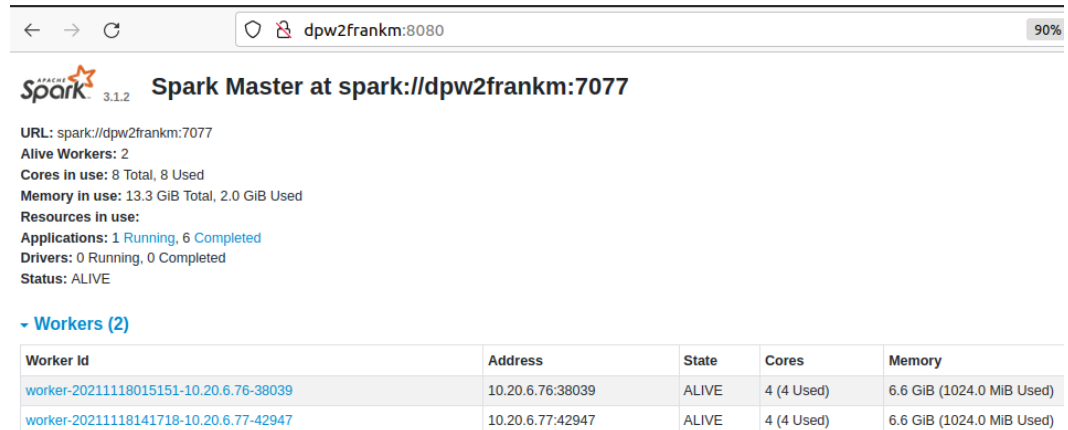


Figure 23 Our Spark Cluster 1

|              |        |                  |            |
|--------------|--------|------------------|------------|
| dpw2Frank-w2 | active | UbuntuD20.04_60G | 10.20.6.77 |
| dpw2Frank-w1 | active | UbuntuD20.04_60G | 10.20.6.76 |
| dpw2Frank-m  | active | UbuntuD20.04_60G | 10.20.6.75 |

Figure 24 Our Spark Cluster 2

The codes using Spark:

```
#Inverted Index using cluster and mapreduce:
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession
import os, re
from time import *

begin_time = time() #to get the running time

spark = SparkSession.builder.master('spark://dpw2frankm:7077').appName('SparkInvIndex').getOrCreate()
sc = spark.sparkContext
#sc = SparkContext.getOrCreate(SparkConf())

inverted_index = sc.wholeTextFiles('hdfs://localhost:9000/user/root/ChildrenBooks/')\
    .flatMap(lambda x: [(os.path.basename(x[0]).split(".")[0], i), i], 1) \
    .reduceByKey(lambda a, b: a + b)\
    .map(lambda x: (x[0][1], (x[0][0], x[1])))

#output = inverted_index.collect()
#for i in range(inverted_index.count()):

#    print(output[i])
#    if i>100: #limit number to print
#        break

end_time = time() #to get the running time
run_time = end_time - begin_time
print('Running time: ', run_time)
```

our Spark cluster

Figure 25 Codes of InvIndex Using Mapreduce and Spark

For small files (100MB), it only took 11s:

```
end_time = time()    #to get the running time
run_time = end_time - begin_time
print('Running time: ', run_time)

('necks', ('ChildrenBook_114', 26))
('sure', ('ChildrenBook_114', 21))
('indeed', ('ChildrenBook_114', 27))
('stones', ('ChildrenBook_114', 13))
('drove', ('ChildrenBook_114', 31))
('end', ('ChildrenBook_114', 47))
('days', ('ChildrenBook_114', 107))
('Beautiful', ('ChildrenBook_114', 10))
('Sicilianische', ('ChildrenBook_114', 48))
('Fifty', ('ChildrenBook_114', 10))
('years', ('ChildrenBook_114', 113))
('ago', ('ChildrenBook_114', 49))
('lived', ('ChildrenBook_114', 84))
('anxious', ('ChildrenBook_114', 10))
('married', ('ChildrenBook_114', 10))
('determined', ('ChildrenBook_114', 10))
('seemed', ('ChildrenBook_114', 46))
('maiden', ('ChildrenBook_114', 100))
('commanded', ('ChildrenBook_114', 11))

print('Running time: ', run_time)

11.9827483409587
```

Figure 26 Results for Small Files

And for large files (1GB), it only took 10min:

```
end_time = time()    #to get the running time
run_time = end_time - begin_time
print('Running time: ', run_time)

('Matthew', ('ChildrenBook_1317', 148))
('ministers', ('ChildrenBook_1317', 84))
('shape', ('ChildrenBook_1317', 37))
('form', ('ChildrenBook_1317', 37))
('Smith', ('ChildrenBook_1317', 29))
('summing', ('ChildrenBook_1317', 29))
('delivery', ('ChildrenBook_1317', 29))
('poor', ('ChildrenBook_1317', 29))
('fault', ('ChildrenBook_1317', 40))
('Terry', ('ChildrenBook_1317', 29))
('let', ('ChildrenBook_1317', 48))
('run', ('ChildrenBook_1317', 47))
('matter', ('ChildrenBook_1317', 77))
('Wood', ('ChildrenBook_1317', 43))
('theology', ('ChildrenBook_1317', 47))
('sound', ('ChildrenBook_1317', 64))
('told', ('ChildrenBook_1317', 41))
('laugh', ('ChildrenBook_1317', 26))
('undignified', ('ChildrenBook_1317', 26))

print('Running time: ', run_time)

593.4163906574249
```

Figure 27 Results for Large Files

Therefore, it's obviously that Spark and MapReduce made it significantly faster.

#### ■ Results comparison and analysis:

After comparing the performance of them, we can find that, if we use

MapReduce, the calculation of the Inverted Index can be dramatically faster. And Spark can also accelerate the calculation significantly. In conclusion, while we doing big data analysis, Spark and MapReduce are very helpful and even necessary.

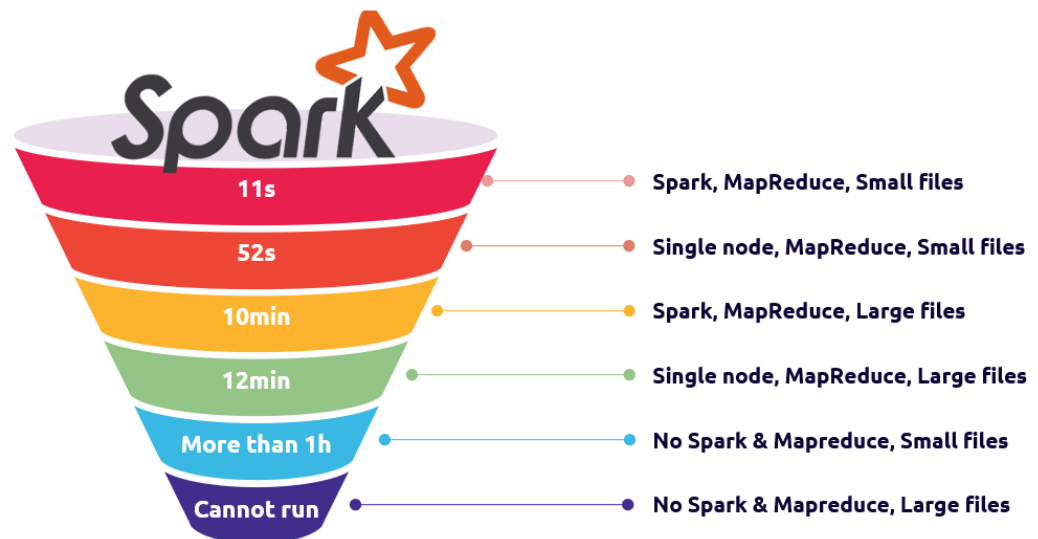


Figure 28 Results Analysis of InvIndex

### ■ Third-party library for indexing: Elasticsearch:



Figure 29 elasticsearch

Elasticsearch is an open source distributed full-text search and analysis engine. It supports RESTful operations and allows you to store, search and analyze large amounts of data in real time. Elasticsearch is one of the most popular search engines that power applications with complex search requirements, such as large e-commerce stores and analytics applications. Elasticsearch is a distributed document store. Instead of storing information as rows of columnar data, Elasticsearch stores complex data



structures that have been serialized as JSON documents. When you have multiple Elasticsearch nodes in a cluster, stored documents are distributed across the cluster and can be accessed immediately from any node.

When a document is stored, it is indexed and fully searchable in near real-time--within 1 second. Elasticsearch uses a data structure called an inverted index that supports very fast full-text searches. An inverted index lists every unique word that appears in any document and identifies all of the documents each word occurs in.

- Step 1:

First of all, Elasticsearch can only receive formatted data, so we need to convert the text file into formatted data (**json**).

The picture below shows the unprocessed text file.

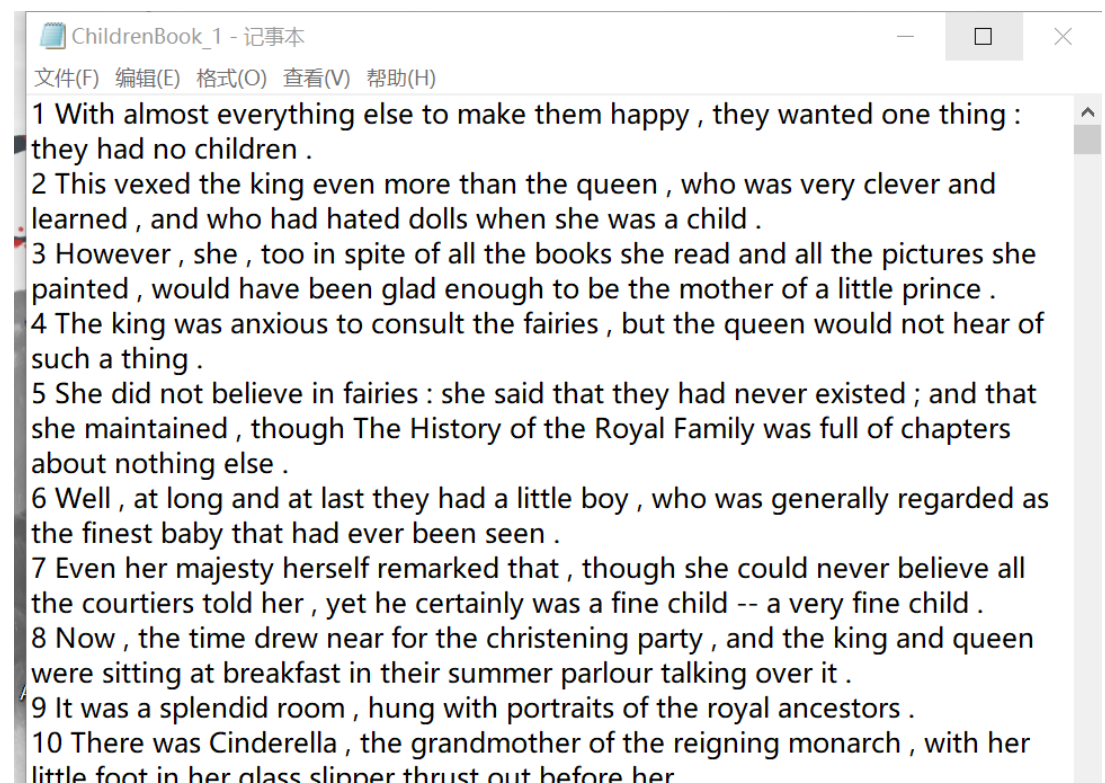
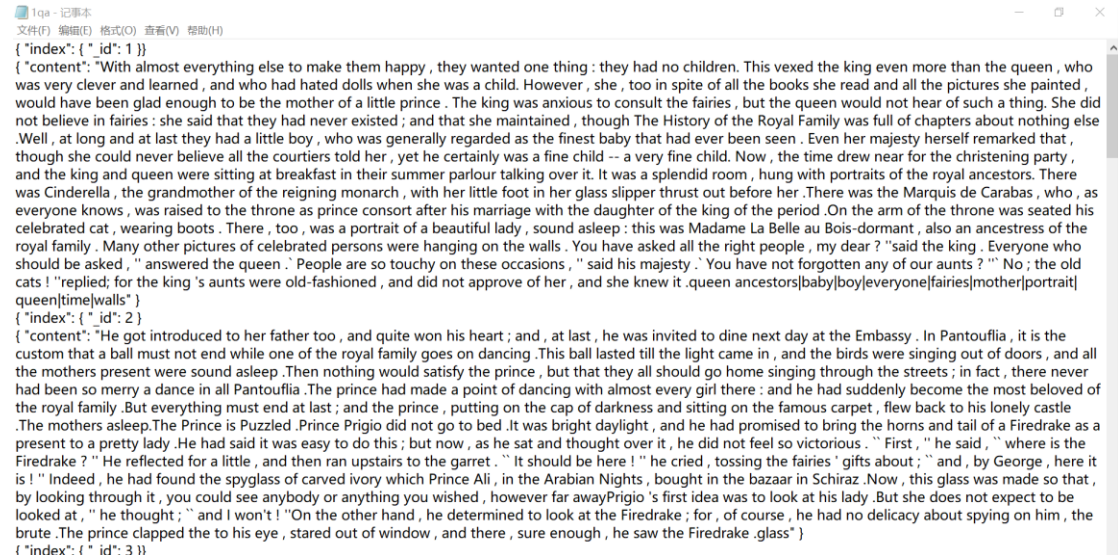


Figure 30 Unprocessed Text File

Since after we done the transform job on python code, it cannot totally achieve what we want in the json file. In the planned json file in our dataset, we want every index referring to a Children's book of

our dataset. Thus, we planned to rewrite the json file on Windows

The picture below shows the processed text file by ourselves.

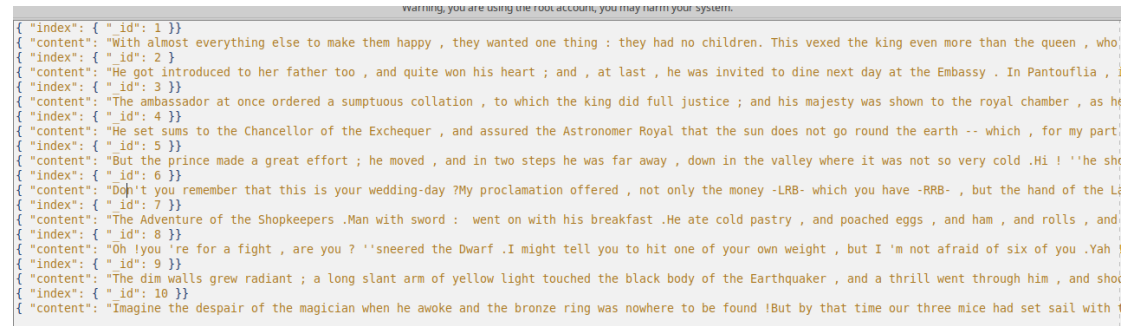


```

Tqa - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
{"index": {"id": 1 }}
{"content": "With almost everything else to make them happy , they wanted one thing : they had no children. This vexed the king even more than the queen , who was very clever and learned , and who had hated dolls when she was a child. However , she , too in spite of all the books she read and all the pictures she painted , would have been glad enough to be the mother of a little prince . The king was anxious to consult the fairies , but the queen would not hear of such a thing. She did not believe in fairies : she said that they had never existed ; and that she maintained , though The History of the Royal Family was full of chapters about nothing else .Well , at long and at last they had a little boy , who was generally regarded as the finest baby that had ever been seen . Even her majesty herself remarked that , though she could never believe all the courtiers told her , yet he certainly was a fine child -- a very fine child. Now , the time drew near for the christening party , and the king and queen were sitting at breakfast in their summer parlour talking over it. It was a splendid room , hung with portraits of the royal ancestors. There was Cinderella , the grandmother of the reigning monarch , with her little foot in her glass slipper thrust out before her. There was the Marquis de Carabas , who , as everyone knows , was raised to the throne as prince consort after his marriage with the daughter of the king of the period .On the arm of the throne was seated his celebrated cat , wearing boots . There , too , was a portrait of a beautiful lady , sound asleep : this was Madame La Belle au Bois-dormant , also an ancestress of the royal family . Many other pictures of celebrated persons were hanging on the walls . You have asked all the right people , my dear ? "said the king . Everyone who should be asked , " answered the queen . People are so touchy on these occasions , " said his majesty . You have not forgotten any of our aunts ? " " No ; the old cats ! "replied; for the king 's aunts were old-fashioned , and did not approve of her , and she knew it .queen ancestors[baby]boy[everyone]fairies[mother]portrait queen[time]walls }
{"index": {"id": 2 }}
{"content": "He got introduced to her father too , and quite won his heart ; and , at last , he was invited to dine next day at the Embassy . In Pantouflia , it is the custom that a ball must not end while one of the royal family goes on dancing .This ball lasted till the light came in , and the birds were singing out of doors , and all the mothers present were sound asleep .Then nothing would satisfy the prince , but that they all should go home singing through the streets ; in fact , there never had been so merry a dance in all Pantouflia .The prince had made a point of dancing with almost every girl there : and he had suddenly become the most beloved of the royal family .But everything must end at last ; and the prince , putting on the cap of darkness and sitting on the famous carpet , flew back to his lonely castle .The mothers asleep.The Prince is Puzzled .Prince Prigio did not go to bed .It was bright daylight , and he had promised to bring the horns and tail of a Firedrake as a present to a pretty lady .He had said it was easy to do this ; but now , as he sat and thought over it , he did not feel so victorious . " First , " he said , " where is the Firedrake ? " He reflected for a little , and then ran upstairs to the garret . " It should be here ! " he cried , tossing the fairies ' gifts about ; " and , by George , here it is ! " Indeed , he had found the spyglass of carved ivory which Prince Ali , in the Arabian Nights , bought in the bazaar in Schiraz .Now , this glass was made so that , by looking through it , you could see anybody or anything you wished , however far away.Prigio 's first idea was to look at his lady .But she does not expect to be looked at , " he thought ; " and I won't ! "On the other hand , he determined to look at the Firedrake ; for , of course , he had no delicacy about spying on him , the brute .The prince clapped the to his eye , stared out of window , and there , sure enough , he saw the Firedrake .glass" }
{"index": {"id": 3 }}
  
```

Figure 31 Processed Text File

And this one is on Ubuntu, it has a very long row in every index:



```

warning: you are using the root account, you may harm your system.
{"index": {"id": 1 }}
{"content": "With almost everything else to make them happy , they wanted one thing : they had no children. This vexed the king even more than the queen , who was very clever and learned , and who had hated dolls when she was a child. However , she , too in spite of all the books she read and all the pictures she painted , would have been glad enough to be the mother of a little prince . The king was anxious to consult the fairies , but the queen would not hear of such a thing. She did not believe in fairies : she said that they had never existed ; and that she maintained , though The History of the Royal Family was full of chapters about nothing else .Well , at long and at last they had a little boy , who was generally regarded as the finest baby that had ever been seen . Even her majesty herself remarked that , though she could never believe all the courtiers told her , yet he certainly was a fine child -- a very fine child. Now , the time drew near for the christening party , and the king and queen were sitting at breakfast in their summer parlour talking over it. It was a splendid room , hung with portraits of the royal ancestors. There was Cinderella , the grandmother of the reigning monarch , with her little foot in her glass slipper thrust out before her. There was the Marquis de Carabas , who , as everyone knows , was raised to the throne as prince consort after his marriage with the daughter of the king of the period .On the arm of the throne was seated his celebrated cat , wearing boots . There , too , was a portrait of a beautiful lady , sound asleep : this was Madame La Belle au Bois-dormant , also an ancestress of the royal family . Many other pictures of celebrated persons were hanging on the walls . You have asked all the right people , my dear ? "said the king . Everyone who should be asked , " answered the queen . People are so touchy on these occasions , " said his majesty . You have not forgotten any of our aunts ? " " No ; the old cats ! "replied; for the king 's aunts were old-fashioned , and did not approve of her , and she knew it .queen ancestors[baby]boy[everyone]fairies[mother]portrait queen[time]walls }
{"index": {"id": 2 }}
{"content": "He got introduced to her father too , and quite won his heart ; and , at last , he was invited to dine next day at the Embassy . In Pantouflia , it is the custom that a ball must not end while one of the royal family goes on dancing .This ball lasted till the light came in , and the birds were singing out of doors , and all the mothers present were sound asleep .Then nothing would satisfy the prince , but that they all should go home singing through the streets ; in fact , there never had been so merry a dance in all Pantouflia .The prince had made a point of dancing with almost every girl there : and he had suddenly become the most beloved of the royal family .But everything must end at last ; and the prince , putting on the cap of darkness and sitting on the famous carpet , flew back to his lonely castle .The mothers asleep.The Prince is Puzzled .Prince Prigio did not go to bed .It was bright daylight , and he had promised to bring the horns and tail of a Firedrake as a present to a pretty lady .He had said it was easy to do this ; but now , as he sat and thought over it , he did not feel so victorious . " First , " he said , " where is the Firedrake ? " He reflected for a little , and then ran upstairs to the garret . " It should be here ! " he cried , tossing the fairies ' gifts about ; " and , by George , here it is ! " Indeed , he had found the spyglass of carved ivory which Prince Ali , in the Arabian Nights , bought in the bazaar in Schiraz .Now , this glass was made so that , by looking through it , you could see anybody or anything you wished , however far away.Prigio 's first idea was to look at his lady .But she does not expect to be looked at , " he thought ; " and I won't ! "On the other hand , he determined to look at the Firedrake ; for , of course , he had no delicacy about spying on him , the brute .The prince clapped the to his eye , stared out of window , and there , sure enough , he saw the Firedrake .glass" }
{"index": {"id": 3 }}
{"content": "The ambassador at once ordered a sumptuous collation , to which the king did full justice ; and his majesty was shown to the royal chamber , as he }
{"index": {"id": 4 }}
{"content": "He set sums to the Chancellor of the Exchequer , and assured the Astronomer Royal that the sun does not go round the earth -- which , for my part }
{"index": {"id": 5 }}
{"content": "But the prince made a great effort ; he moved , and in two steps he was far away , down in the valley where it was not so very cold .Hi ! "he sh }
{"index": {"id": 6 }}
{"content": "Don't you remember that this is your wedding-day ?My proclamation offered , not only the money -LRB- which you have -RRB- , but the hand of the Li }
{"index": {"id": 7 }}
{"content": "The Adventure of the Shopkeepers .Man with sword : went on with his breakfast .He ate cold pastry , and poached eggs , and ham , and rolls , and }
{"index": {"id": 8 }}
{"content": "Oh !you 're for a fight , are you ? 'sneered the Dwarf .I might tell you to hit one of your own weight , but I 'm not afraid of six of you .Yah }
{"index": {"id": 9 }}
{"content": "The dim walls grew radiant ; a long slant arm of yellow light touched the black body of the Earthquaker , and a thrill went through him , and sho }
{"index": {"id": 10 }}
{"content": "Imagine the despair of the magician when he awoke and the bronze ring was nowhere to be found !But by that time our three mice had set sail with t }
  
```

Figure 32 in Ubuntu

### Remark:

Since we found that it is really hard to using more than 100MB json file on Elasticsearch engine, basically we assume that there are some long blank in our txt file as well as some reading problems like hard to read and system will break down. Thus, we simply read 10 books file into our test json file.

### ● Step 2:

Before we start step 2, we need to firstly install Elasticsearch on our ubuntu 20.04.

Firstly, update the package index and install the necessary dependencies to add a new HTTPS repository:

```
sudo apt update
```

```
sudo apt install apt-transport-https ca-certificates wget
```

Import the GPG key of the repository:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo  
apt-key add -
```

The above command should output OK, which means that the key has been successfully imported and packages from this repository will be considered trusted.

Next, add the Elasticsearch repository to the system by issuing the following command:

```
sudo sh -c 'echo "deb https://artifacts.elastic.co/packages/7.x/apt  
stable main" > /etc/apt/sources.list.d/elasticsearch-7.x.list'
```

If you want to install a previous version of Elasticsearch, change 7.x to the desired version in the above command.

After enabling the repository, enter the following to install Elasticsearch:

```
sudo apt update
```

```
sudo apt install elasticsearch
```

After the installation process is complete, the Elasticsearch service will not start automatically. To start the service and enable it, run:

```
sudo systemctl enable --now elasticsearch.service
```

To verify that Elasticsearch is running, use curl to send an HTTP request to port 9200 on the local host:

```
curl -X GET "localhost:9200/"
```

You should see something like:

```
{
```

```
"name": "vagrant",
"cluster_name": "elasticsearch",
"cluster_uuid": "IJqDxPfXSrmFQ27KbXbRIg",
"version": {
  "number": "7.8.0",
  "build_flavor": "default",
  "build_type": "deb",
  "build_hash":
"757314695644ea9a1dc2fec26d1a43856725e65",
  "build_date": "2020-06-14T19:35:50.234439Z",
  "build_snapshot": false,
  "lucene_version": "8.5.1",
  "minimum_wire_compatibility_version": "6.8.0",
  "minimum_index_compatibility_version": "6.0.0-beta1"
},
"tagline": "You Know, for Search"
}
```

Elasticsearch has been installed on your Ubuntu server.

And after that, we can input our json file into Elasticsearch

Using code on terminal:

```
curl -H 'Content-Type: application/x-ndjson' -XPOST
'10.0.0.19:9200/es/doc/_bulk?pretty' --data-binary @es.json
```

```
root@dpw1:~# curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/es/_bulk?pretty' --data-binary @es_bonus.json
{
  "took" : 557,
  "errors" : false,
  "items" : [
    {
      "index" : {
        "_index" : "es",
        "_type" : "doc",
        "_id" : "1",
        "_version" : 1,
        "result" : "created",
        "shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        }
      },
      "seq no" : 0,

```

Figure 33 Terminal 1

From this picture you can see that, we successfully input file es, which has total 10 number of documents, and its store about 35.6kb.

And if we want to check the condition of each instance on Elasticsearch engine, we can use this code on terminal:

***curl 'localhost:9200/\_cat/indices?v'***

```
root@dpw1:~# curl 'localhost:9200/_cat/indices?v'
health status index      uuid                               pri rep docs.count docs.deleted store.size pri.store.size
green open   .geoip_databases gmFlnzSAQC-onfnFVbkQ-w 1 0 43 36 40.9mb 40.9mb
yellow open   es                8_vrqUJXRswilBShvXIDXw 1 1 10 0 35.6kb 35.6kb
root@dpw1:~#
```

Figure 34 Terminal 2

### ● Step 3:

In Step 3 we need to use Jupyter Notebook and python code to finish inverted index job.

Before we using python code, we need to firstly install a new library **–inelastic**.

inelastic builds an approximation of how an inverted index would look like for a particular index and document field, using the Multi termvectors API on all stored documents.

To install inelastic, run the following command:

***\$ pip3 install --upgrade inelastic***

inelastic currently only supports Elasticsearch versions 6.X and 7.X.

The inelastic module exposes the **InvertedIndex class**.

Our python code:



```
In [1]: '''
        Bonus part:
        The Inverted Index based on Elasticsearch and inelastic
        '''
        from inelastic import InvertedIndex
        from elasticsearch import Elasticsearch
        import time # Calculate running time.

        es = Elasticsearch()
        ii = InvertedIndex(search_size = 2500, scroll_time = '10s')

        start = time.time()

        n_docs, errors = ii.read_index(es, 'es', 'content')
        print('# docs: {}, #errors: {}'.format(n_docs, errors))

        for entry in ii.to_list():
            print(entry)

        end = time.time()
        print('\n\n\n')
        print("Consume %.3f s, Inverted Index done successfully!"%(end-start))
```

Figure 35 inelastic Codes

And the result:

```
print("Consume %.3f s, Inverted Index done successfully!"%(end-start))

# docs: 10, #errors: 0
('a', <IndexEntry IDs: ['1', '10', '2', '3', '4', '5', '6', '7', '8', '9']>)
('abolished', <IndexEntry IDs: ['7']>)
('about', <IndexEntry IDs: ['1', '10', '2', '4', '6', '7']>)
('abyss', <IndexEntry IDs: ['9']>)
('accent', <IndexEntry IDs: ['4']>)
('accompanied', <IndexEntry IDs: ['3']>)
('advanced', <IndexEntry IDs: ['5']>)
('adventure', <IndexEntry IDs: ['7']>)
('affair', <IndexEntry IDs: ['9']>)
('affection', <IndexEntry IDs: ['6']>)
('afraid', <IndexEntry IDs: ['4', '8']>)
('after', <IndexEntry IDs: ['1', '3', '7']>)
('afterwards', <IndexEntry IDs: ['9']>)
('again', <IndexEntry IDs: ['3', '9']>)
('air', <IndexEntry IDs: ['3', '5']>)
```

Figure 36 inelastic Result 1



```
print("Consume %.3f s, Inverted Index done successfully!"%(end-start))
('work', <IndexEntry IDs: ['7']>)
('working', <IndexEntry IDs: ['6']>)
('world', <IndexEntry IDs: ['4', '9']>)
('worst', <IndexEntry IDs: ['4']>)
('worth', <IndexEntry IDs: ['7']>)
('would', <IndexEntry IDs: ['1', '10', '2', '3', '4', '7', '9']>)
('wrong', <IndexEntry IDs: ['4']>)
('yah', <IndexEntry IDs: ['8']>)
('yellow', <IndexEntry IDs: ['9']>)
('yet', <IndexEntry IDs: ['1', '3', '5', '9']>)
('you', <IndexEntry IDs: ['1', '2', '3', '4', '5', '6', '7', '8', '9']>)
('young', <IndexEntry IDs: ['4', '8']>)
('younger', <IndexEntry IDs: ['4']>)
('your', <IndexEntry IDs: ['6', '7', '8']>)
```

```
Consume 0.156 s, Inverted Index done successfully!
```

Figure 37 inelastic Result 2

You can see that every word has a relevant row which includes the file that it exists.

And additionally, I want to refer another way to achieve inverted index, which also using the application of inelastic. But now, we can run it on terminal, code:

```
$ inelastic -i es -f content | column -t -s,
```

```
root@dawl:~# inelastic -i es -f content | column -t -s ,
```

Figure 38 Terminal 3

And the result:



[illegible]

Figure 39 Terminal Result 1

|            |     |    |  |    |    |   |   |   |   |   |   |   |   |
|------------|-----|----|--|----|----|---|---|---|---|---|---|---|---|
| temple     | 1   | 1  |  | 9  |    |   |   |   |   |   |   |   |   |
| ten        | 1   | 1  |  | 4  |    |   |   |   |   |   |   |   |   |
| terribly   | 1   | 1  |  | 5  |    |   |   |   |   |   |   |   |   |
| than       | 5   | 5  |  | 1  | 10 | 4 | 6 | 8 |   |   |   |   |   |
| that       | 31  | 9  |  | 1  | 10 | 2 | 3 | 4 | 5 | 6 | 7 | 9 |   |
| the        | 289 | 10 |  | 1  | 10 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| their      | 8   | 6  |  | 1  | 10 | 3 | 4 | 5 | 8 |   |   |   |   |
| them       | 9   | 6  |  | 1  | 10 | 3 | 4 | 7 | 8 |   |   |   |   |
| themselves | 1   | 1  |  | 3  |    |   |   |   |   |   |   |   |   |
| then       | 11  | 4  |  | 2  | 3  | 5 | 9 |   |   |   |   |   |   |
| there      | 15  | 7  |  | 1  | 10 | 2 | 3 | 5 | 6 | 7 |   |   |   |
| these      | 4   | 4  |  | 1  | 3  | 7 | 8 |   |   |   |   |   |   |
| they       | 25  | 8  |  | 1  | 10 | 2 | 3 | 4 | 5 | 7 | 9 | I |   |
| thin       | 1   | 1  |  | 4  |    |   |   |   |   |   |   |   |   |
| thing      | 4   | 3  |  | 1  | 4  | 7 |   |   |   |   |   |   |   |
| things     | 1   | 1  |  | 3  |    |   |   |   |   |   |   |   |   |
| think      | 1   | 1  |  | 9  |    |   |   |   |   |   |   |   |   |
| third      | 1   | 1  |  | 4  |    |   |   |   |   |   |   |   |   |
| this       | 12  | 7  |  | 1  | 10 | 2 | 3 | 4 | 6 | 7 |   |   |   |
| though     | 2   | 1  |  | 1  |    |   |   |   |   |   |   |   |   |
| thought    | 2   | 1  |  | 2  |    |   |   |   |   |   |   |   |   |
| thousands  | 1   | 1  |  | 9  |    |   |   |   |   |   |   |   |   |
| three      | 3   | 2  |  | 10 | 4  |   |   |   |   |   |   |   |   |
| thrill     | 1   | 1  |  | 9  |    |   |   |   |   |   |   |   |   |
| throne     | 3   | 2  |  | 1  | 6  |   |   |   |   |   |   |   |   |
| through    | 6   | 4  |  | 2  | 3  | 5 | 9 |   |   |   |   |   |   |
| throwing   | 1   | 1  |  | 9  |    |   |   |   |   |   |   |   |   |
| thrust     | 1   | 1  |  | 1  |    |   |   |   |   |   |   |   |   |
| till       | 4   | 2  |  | 2  | 5  |   |   |   |   |   |   |   |   |
| time       | 5   | 4  |  | 1  | 10 | 4 | 5 |   |   |   |   |   |   |
| to         | 89  | 10 |  | 1  | 10 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| told       | 2   | 2  |  | 1  | 9  |   |   |   |   |   |   |   |   |

Figure 40 Terminal Result 2

The first column is the frequency of each word, and the second





column is the total count of appearance of the number the file, and the last few columns are the actual file for the word has appeared overall.

Take word ‘time’ as an example. ‘time’ has appeared 5 times in 4 different documents, they are file 1, 10, 4, 5.

I want to emphasize that the second way is clearer and simpler than the first way, both of them based on the library inelastic and Elasticsearch.

## 2) Ranking:

### ■ TF-IDF using Spark:

Firstly, for ranking, we used Spark to calculate the TF-IDF values which can decide the position at which a particular document appears in the results of a search engine query, and saved them in a directory (“tfidf-index”).

```
#Ranking with TF-IDF:
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession, SQLContext
import os, re
import math

#First we compute the TF-IDF of all files in ./itfidf, save as an RDD
#We need SQLContext for processing RDD->DF
#If you want to run the job on Spark cluster, you have to modify the following line, e.g.:
spark = SparkSession.builder.master('spark://dpw2tcxu:7077').appName('MyWordCount').getOrCreate()
sc = spark.sparkContext
sql = SQLContext(sc)

#If you want to run the job with Hadoop HDFS, you have to modify the following line:
data = sc.wholeTextFiles('hdfs://localhost:9000/user/root/ChildrenBooks/')

numFiles = data.count()

wordcount = data.flatMap(lambda x: [(os.path.basename(x[0]), i), 1] for i in re.split('\W', x[1]))\
    .reduceByKey(lambda a, b: a + b)

tf = wordcount.map(lambda x: (x[0][1], (x[0][0], x[1])))

idf = wordcount.map(lambda x: (x[0][1], (x[0][0], x[1], 1)))\
    .map(lambda x: (x[0], x[1][2]))\
    .reduceByKey(lambda x, y: x + y)\
    .map(lambda x: (x[0], math.log10(numFiles / x[1])))

#Slightly modified map output as (doc, (term, tfidf))
tfidf = tf.join(idf)\
    .map(lambda x: (x[1][0][0], (x[0], x[1][0][1] * x[1][1])))\
    .sortByKey()

#Then we convert the TF-IDF to an DF, and save to the disk
lines = tfidf.map(lambda x: (x[0], x[1][0], x[1][1])).toDF()
lines.write.save("tfidf-index")
```

Figure 41 Codes of Ranking Using TF-IDF

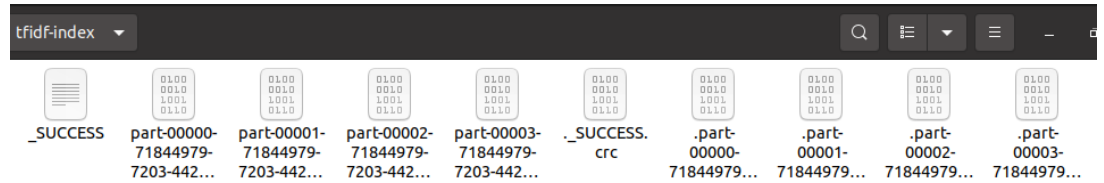


Figure 42 Results of Ranking TF-IDF

Then we do the search based on the result. We can get the search result if we search a word in children's books such as "little red riding-hood", then we can get the most related books.

```
#The RDD is mapped as map(lambda x: (x['term'],(x['doc'],x['tfidf']))), for example:
#tfidf_RDD = tfidf.map(lambda x: (x[1][0],(x[0],x[1][1])))

#Now we load the TF-IDF index from the disk, can convert it to an RDD, and map as x['term'],(x['doc'],x['tfidf'])
tfidf_RDD = sql.read.parquet("tfidf-index").rdd.map(lambda x: (x['_2'],(x['_1'],x['_3'])))

def tokenize(s):
    return re.split("\\W+", s.lower())

def search(query, topN):
    tokens = sc.parallelize(tokenize(query)).map(lambda x: (x, 1)).collectAsMap()
    bcTokens = sc.broadcast(tokens)

    #connect to documents with terms in the Query. to Limit the computation space
    #so that we don't attempt to compute similarity for docs that have no words in common with our query.
    joined_tfidf = tfidf_RDD.map(lambda x: (x[0], bcTokens.value.get(x[0], '-'), x[1])).filter(lambda x: x[1] != '-')

    #compute the score using aggregateByKey
    scount = joined_tfidf.map(lambda a: a[2]).aggregateByKey((0,0),
        (lambda acc, value: (acc[0] + value, acc[1] + 1)),
        (lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])))

    scores = scount.map(lambda x: (x[1][0]*x[1][1]/len(tokens), x[0])).top(topN)

    return scores

search("Little Red Riding-Hood", 5)
```

Do the search

```
[(282.6046739740165, 'ChildrenBook_793.txt'),
(269.6531691811077, 'ChildrenBook_594.txt'),
(171.44891901841405, 'ChildrenBook_1249.txt'),
(152.4123610489474, 'ChildrenBook_961.txt'),
(146.10517302768415, 'ChildrenBook_1218.txt')]
```

Figure 43 Search Results of Ranking with TF-IDF

### ■ Third-party library for ranking: **Elasticsearch**:

We also found other ways to do the ranking job, which is Elasticsearch.

Firstly, we use "query" to search for all the data according to the created index before. Here we use "match\_all" to show the all documents at first:



```
from elasticsearch import Elasticsearch # Only with ES 7.X
es = Elasticsearch()

body = {
    "query": {
        "match_all": {}
    }
}

es.search(index="es", _source="content", body=body)
```

Figure 44 Codes of Showing Documents

```
{
  'took': 3,
  'timed_out': False,
  '_shards': {'total': 1, 'successful': 1, 'skipped': 0, 'failed': 0},
  'hits': {'total': {'value': 10, 'relation': 'eq'},
    'max_score': 1.0,
    'hits': [
      {
        '_index': 'es',
        '_type': '_doc',
        '_id': '1',
        '_score': 1.0,
        '_ignored': ['content.keyword'],
        '_source': {
          'content': 'With almost everything else to make them had no children. This vexed the king even more than the queen , who had hated dolls when she was a child. However , she , too in spite o ictures she painted , would have been glad enough to be the mother o us to consult the fairies , but the queen would not hear of such a t she said that they had never existed ; and that she maintained , tho s full of chapters about nothing else .Well , at long and at last th y regarded as the finest baby that had ever been seen . Even her maj e could never believe all the courtiers told her , yet he certainly
```

Figure 45 Result of Showing Documents

In the showed documents, there are some attributed that need to be known:

- The first is “**took**”, it shows the time we spent in milliseconds;
- Then the “**hits.total**”, we can see it has ten example documents all;
- Then the “**hits.hits**”, it contains the detail data of the documents.

If we use “**match**” for searching, the searching for the file where the

matching word in the “content” is located:

```
In [3]: '''
        Search specific word in the file
        '''

        body = {

            "query": {

                "match": {

                    "content": "way"

                }

            }

        }

        es.search(index="es", _source="content", body=body)
```

Figure 46 Codes of Searching Using “match”

And the result after searching for the word “way” is as following. It shows the ID and the value of score:

```
Out[4]: {'took': 11,
        'timed_out': False,
        'shards': {'total': 1, 'successful': 1, 'skipped': 0, 'failed': 0},
        'hits': {'total': {'value': 3, 'relation': 'eq'},
        'max_score': 1.1261231,
        'hits': [{'_index': 'es',
        '_type': '_doc',
        '_id': '4',
        '_score': 1.1261231,
        '_ignored': ['content.keyword'],
        '_source': {'content': "He set sums to the Chancellor of the Exchequer , and assured the
Astronomer Royal that the sun does not go round the earth -- which , for my part , I believe
it does .The young ladies of the court disliked dancing with him , in spite of his good look
s , because he was always asking , `` Have you read this ? ''and `` Have you read that ? ''a
nd when they said they had n't , he sneered ; and when they said they had , he found them ou
t .He found out all his tutors and masters in the same horrid way ; correcting the accent of
his French teacher , and trying to get his German tutor not to eat peas with his knife .He a
lso endeavoured to teach the queen-dowager , his grandmother , an art with which she had lon
g been perfectly familiar !In fact , he knew everything better than anybody else ; and the w
orst of it was that he did : and he never was in the wrong , and he always said , `` Didn't
I tell you so ? '' And , what was more , he had !Illustration : page As time went on , Prin
ce Prigio had two younger brothers , whom everybody liked : They were not a bit clever , but
jolly . Prince Alphonse , the third son , was round , fat , good humoured , and as brave as a
```

Figure 47 Result of Searching Using “match”

It can be found that the outputs are sorted by the score:

```
'_id': '4',
'_score': 1.1261231,
'_id': '5',
'_score': 1.0873576,
```

```
'id': '3',
'score': 1.0511723,
```

Figure 48 Sorted ID and Score

And the sorted values are formed by **Term Frequency, Inverse Document Frequency** and the **normalized values of Document Frequency**.

- Term Frequency:

The higher the frequency, the higher the weight. Fields that mention the same word five times are more relevant than those that mention it only once.

```
tf(t in d) = √frequency
```

Figure 49 TF

- Inverse Document Frequency:

The higher the frequency, the lower the weight. Common words such as “and” or “the” make little contribution to relevance because they appear in most documents. Some uncommon words such as “elastic” or “hippopotamus” can help us quickly narrow down the scope and find documents of interest.

```
idf(t) = 1 + log ( numDocs / (docFreq + 1))
```

Figure 50 IDF

- The normalized values of Document Frequency:

The shorter the field, the higher the weight of the field. If a word appears in a field such as title, it is more relevant than it appears in a field such as body. The normalization formula of field length is as following:

```
norm(d) = 1 / √numTerms
```

Figure 51 Norm DF

Finally, the score is calculated by:

```

1 score(q,d) =
2     queryNorm(q)
3     · coord(q,d)
4     ·  $\sum$  (
5         tf(t in d)
6         · idf(t)2
7         · t.getBoost()
8         · norm(t,d)
9     ) (t in q)

```

Figure 52 Calculation of the Score

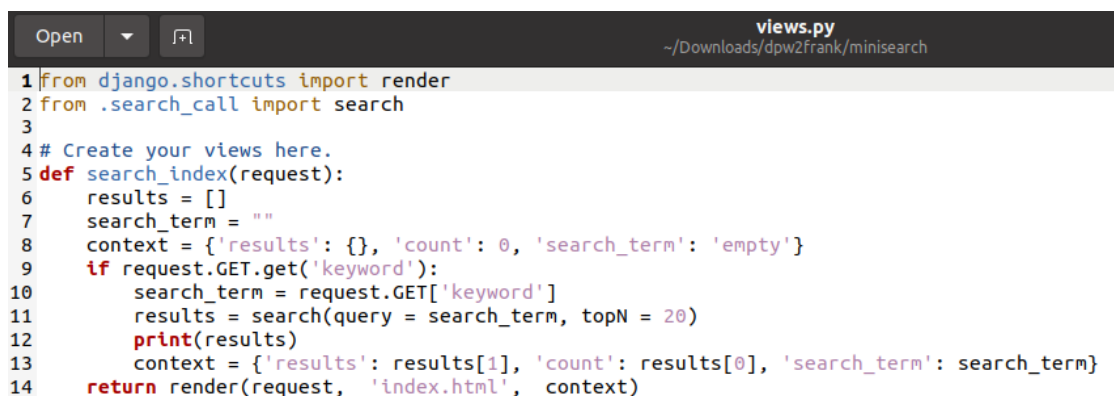
Thus, the value of “**\_score**” can be used as our standard of sorting. In other condition, when we searching for a word, **the normalized values of Document Frequency and Inverse Document Frequency** are similar, the **Term Frequency** will be used as the sorting standard.

#### 4. Web Server and Interface of the Search Engine

Finally, we used Django to build a web interface for the search engine based on the TD-IDF results we calculated in the previous part.

We wrote a program called “minisearch” and ran it also with the help of Spark. Our codes are as following:

1) views.py:



```

Open  views.py
~/Downloads/dpw2frank/minisearch

1 from django.shortcuts import render
2 from .search_call import search
3
4 # Create your views here.
5 def search_index(request):
6     results = []
7     search_term = ""
8     context = {'results': {}, 'count': 0, 'search_term': 'empty'}
9     if request.GET.get('keyword'):
10         search_term = request.GET['keyword']
11         results = search(query = search_term, topN = 20)
12         print(results)
13         context = {'results': results[1], 'count': results[0], 'search_term': search_term}
14     return render(request, 'index.html', context)

```

Figure 53 views.py

2) search\_all.py: (the main application of the “minisearch”, and we use Spark to help the calculation)





```
1 from pyspark import SparkConf, SparkContext
2 from pyspark.sql import SparkSession, SQLContext
3 import os, re, sys, math
4
5 #sc = SparkContext.getOrCreate(SparkConf())
6 #sql = SQLContext(sc)
7
8 #You can also submit the job to a spark cluster, e.g.:
9 spark = SparkSession.builder.master('spark://dpw2frankm:7077').appName('searchcall').getOrCreate()
10 sc = spark.sparkContext
11 sql = SQLContext(sc)
12
13 #Be careful, in case of spark cluster, the tf-idf index should be in an HDFS, e.g.:
14 tfidf_RDD = sql.read.parquet('hdfs://localhost:9000/user/root/tfidf-index').rdd.map(lambda x: (x['_2'],(x['_1'],x['_3'])))
15
16 #tfidf_RDD = sql.read.parquet("tfidf-index").rdd.map(lambda x: (x['_2'],(x['_1'],x['_3'])))
17
18 def tokenize(s):
19     return re.split("\\W+", s.lower())
20
21 def search(query, topN):
22     tokens = sc.parallelize(tokenize(query)).map(lambda x: (x, 1)).collectAsMap()
23     bcTokens = sc.broadcast(tokens)
24
25     joined_tfidf = tfidf_RDD.map(lambda x: (x[0], bcTokens.value.get(x[0], '-'), x[1])).filter(lambda x: x[1] != '-')
26
27     scount = joined_tfidf.map(lambda a: a[2]).aggregateByKey((0,0),
28 (lambda acc, value: (acc[0] + value, acc[1] + 1)),
29 (lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])))
30
31     scores = scount.map(lambda x: (x[1][0]*x[1][1]/len(tokens), x[0])).top(topN)
32
33     return topN,scores
```

Figure 54 search\_all.py

### 3) index.html: (we implement the Download function here)

```
{% load static %}
<!doctype html>
<html>
  <body background="{% static 'img/background.jpg' %}">
<nav class="navbar navbar-light bg-light">
  <center>
    
    <h3><font color="white">Children Books Search Engine</font></h3>
    <form class="form-inline">
      <input
        class="form-control mr-sm-2"
        type="search" placeholder="Keyword"
        aria-label="Keyword"
        name = 'keyword'
        value = ""
      >
      <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
    </form>
  </center>
</nav>
{% if results %}
  <div class="textCentrado margenSup20 grisDC">
    <h3><font color="white">We found {{ count }} result{{ count|pluralize }} for your search "<i>{{ search_term }}</i>". s
    howing top hits according to TF-IDF score: </font></h3>
  </div>
  <div class="search results">
    {% for mytuple in results %}
      <div class="images">
        <font color="white">{{ mytuple.0 }}</font>
        <font color="white">{{ mytuple.1 }}</font>
        <a href='http://172.18.0.3:9864/webhdfs/v1/user/root/ChildrenBooks/{{
        mytuple.1 }}?op=OPEN&namenoderpcaddress=namenode:9000&offset=0'>Download</a>
        <!-- {{ mytuple.3 }}-->
      </div>
    </div>
  </div>
  <p></p>
</div>
```

Figure 55 index.html 1

```

    {% endfor %}
  </div>
  {%elif not search_term %}
  <p><font color="white">Insert your search here.</font></p>
  {%elif not results %}
  <p><font color="white">No books found now. Please input keywords such as "Little Red Riding-Hood" or "ugly duckling".</font></p>
  {% endif %}
</body>
</html>

```

Figure 56 index.html 2

#### 4) The interface:

Home page:

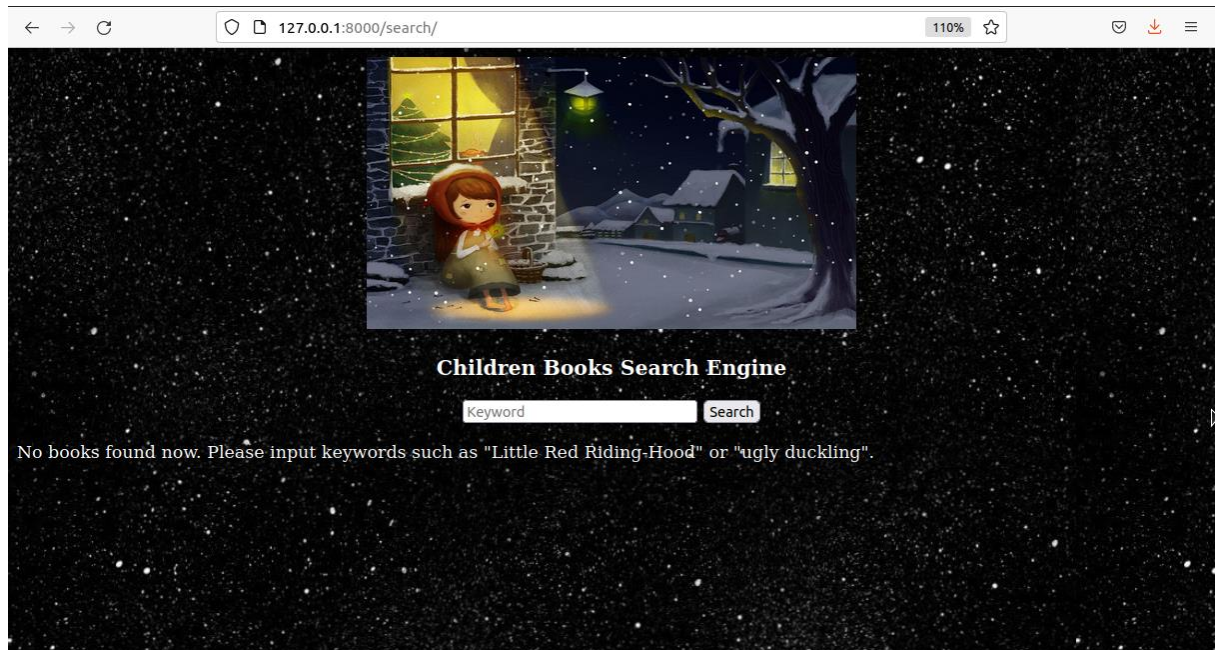


Figure 57 Home Page

Search results page:



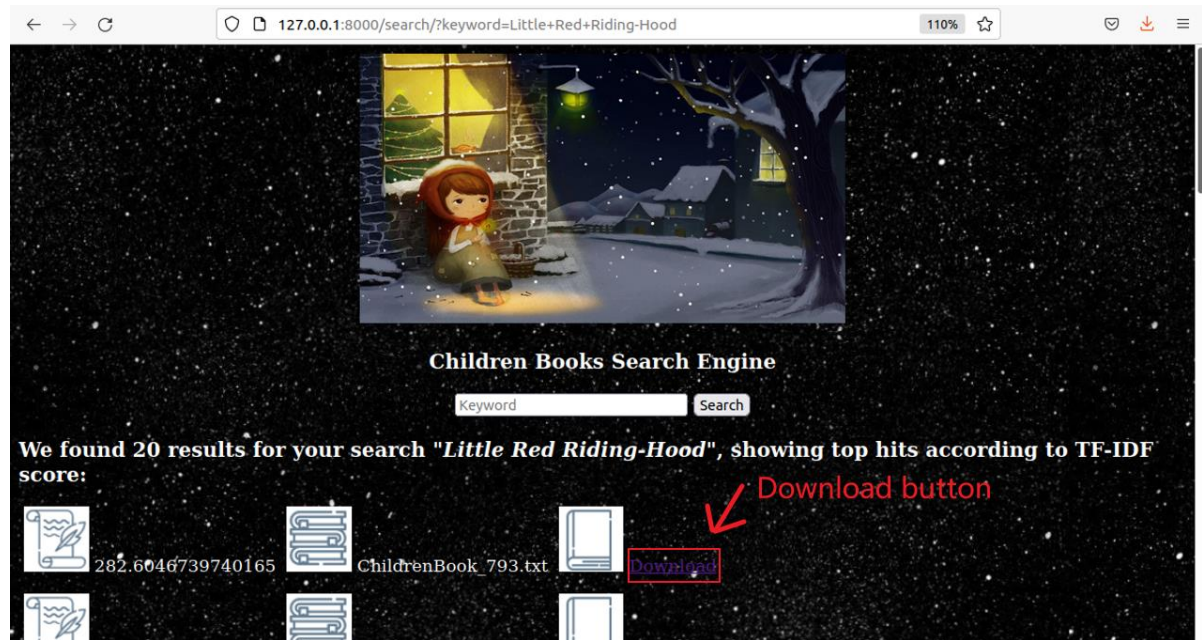


Figure 58 Search Results Page

## 5. Conclusion and Reflection

To achieve our goal that is to implement a system that can retrieve text contents related to the query and provide children a search engine to find, download and read the contents they want, we used the Spark cluster, HDFS, Hadoop MapReduce as well as Elasticsearch to calculate the Inverted Index and sorted them by the TF-IDF value. For a better user experience, we also built a Django web server for children to search and download their dream books.

In the development process we have come up with some “good” ideas to implement this system:

### 1) The splitting method:

For a faster data collection, we downloaded the big files and split them according to the contents, so that the readers can only find and download the specific book they want.

### 2) Readable results:

Our results (books with Stopwords) serve the user experience rather than the

developer's work, which means that they can be read by children easily and directly.

### 3) Help of Spark, Hadoop MapReduce and HDFS:

From the comparison in the research, we can find that, Spark cluster and MapReduce can dramatically accelerate the calculation process of big data, and the HDFS can help us to store and retrieve the big file set, which are really time-consuming and can avoid or recover from crash.

### 4) Download function:

For a convenient user experience, we implement the download function on the web interface, so that children can read the books directly on the web page but not from the HDFS.

However, we also realized some problems that are not "good" enough:

#### 1) The interference of Stopwords:

Because of that we did not remove the Stopwords, the characters not in English and some redundant words are in the results of Inverted Index and TF-IDF, which might influence the accuracy of the search.

#### 2) Hard to update in real time:

Due to the reason that our data are from a book file set but not crawled from a website or social media, if we want to update our results dynamically, we need to update new books or contents manually.

In the future researches and development, we will work hard on those problems to improve our system and try to provide a better service for children.

## 6. Reference

Gheorghe, R., Hinman, M. L., & Russo, R. (2015). *Elasticsearch in action*. Manning.

Dixit, B. (2016). *Elasticsearch Essentials*. Packt Publishing Ltd.

Divya, M. S., & Goyal, S. K. (2013). *ElasticSearch: An advanced and quick search technique to handle voluminous data*. *Compusoft*, 2(6), 171.