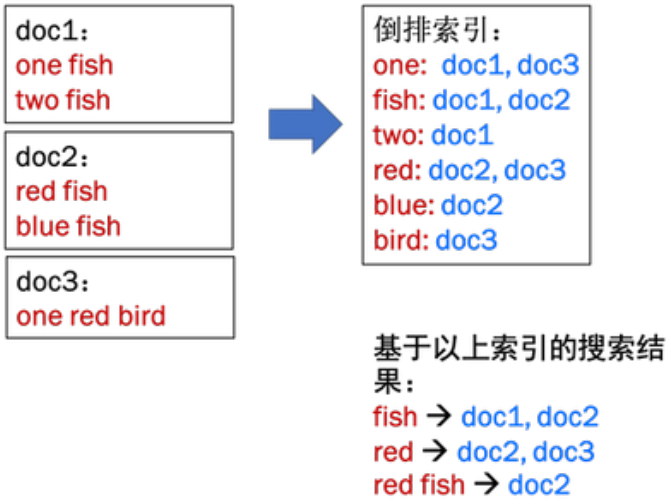
Author: Zhang Jiwen Student ID: 16307110435

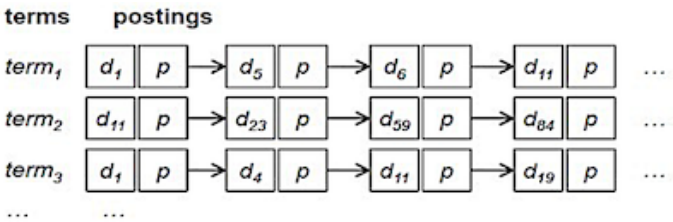
1.具有payload的倒排文档索引

Inverted Index(倒排索引)是几乎所有全文检索搜索引擎都在使用的一种数据索引技术。
采用倒排索引，对于给定的查询词(term)，能快速或取包含有该term的文档列表(the list of documents)。

简单的倒排文档索引示意图:



如果考虑单词在每个文档中出现的词频、位置、对应Web文档的URL等诸多属性，则简单的倒排算法就需要进一步的改进。我们把这些词频、位置等诸多属性称为有效负载（Payload）。



一个倒排索引由大量的postings list构成，每个posting list与一个term相关联。一个posting 包含一个document id和一个payload。payload上载有term在document中出现情况相关的信息（例如：出现位置，词频等等）。

前述提到，MapReduce的核心是Map和Reduce两个函数。Map用于生成会被Reduce使用的中间结果。由于MR系统是运行在分布式集群上的，所以中间结果会被Partition和Combine(可选的)。在Parition过程中，系统自动按照map的输出键进行排序，因此，进入Reduce节点的(key, {value})对将保证是按照key进行排序的，而{value}则不保证是排好序的。

那如果想要对value进行排序，应该怎么办呢?

- 一种办法是在Reduce任务中，对{value}表中的各个value进行排序。但当{value}列表数据量巨大、无法在本地内存中进行排序时，将出现问题。
- 更好的办法是利用MapReduce过程中的Partition过程会对Map输出key进行排序的效应，在分发过程中就完成对value的排序。想要做到这一点，就需要将value中需要排序的部分加入到key中，使key成为复合键。

举例来说，本来Map任务输出的键值对是（word, (doc_titile, word_frequency)）。现在为了实现对word_frequency的排序，我们将Map的输出重置为（(word, word_frequency), doc_titile）。

但这样会带来一个问题：我们仍然想要将同一个word对应的key-value pair放到同一个reduce任务下，但是现在的key是一个复合键。为此，需要实现一个新的Partitioner：从（word, word_frequency）中取出word，以word作为key进行分区。

了解到这些以后，就可以进行MR倒排文档生成了。

2.本次任务

- 倒排文档生成函数：带有payload的倒排文档索引
- 文档查询函数：以contenttile为文本名。实现根据查询关键词，返回相关文档名（序列）
- 提交查询返回结果的截屏

3.代码示例

环境配置

- 阿里云ECS Ubuntu 16.04
- java: open jdk-7
- hadoop: 2.9.2
- python: 3.5

3.1 解决中文乱码问题

由于下载的文件不是utf-8编码，而是gbk编码的。为了后续编程方便和一致性(jieba的官方文档说如果输入的字符串是gbk编码可能会出现未知错误)，建议将文件先转码为utf-8编码。具体操作是：首先在MacOS文本编辑器的偏好设置里，将文件的打开编码设置为gbk，保存编码设置为utf-8，然后打开下载的文件，另存为...即可。然后要记得把文本编辑器的偏好设置改回来（即打开/关闭编码都最好是utf-8）。

然后，将下载的文件scp到已经配好环境的ECS服务器上。

```
scp local_file_path username@server:host_file_path
```

这里建议不管是不是使用云服务器，都检查一下当前使用终端的字符集是否支持中文。如果不支持，在服务器上还是会出现中文乱码问题，可以参照这个链接：ubuntu16.04解决文件中文乱码问题。

3.2 mapper.py

这次选用hadoop streaming来编程。相比mrjob，hadoop streaming更为灵活，用户能够自定义更多特性，而且输入输出都是字符串的特性也比较简单好操作。具体的一些工作原理可以参考官方文档：Hadoop Streaming。

mapper.py从stdin中读入文件，它要做的事是从输入的字符串中：

1. 识别出文档内容和文档名称
2. 对文档内容进行分词（需要安装python第三方库jieba）
3. 返回(单词，所在文档，词频)这样的key-value pair

```
import sys
import re
import jieba
from collections import Counter

pattern1 = r"<[a-zA-Z]*>"  # <>
pattern2 = r">[^a-z]*"  # ><
pattern3 = r"\".+?\""

contents = []
titles = []

for line in sys.stdin:
    result = re.search(pattern1, line)
    if result is not None:
        typo = result.group()[1:-1]
        _text = re.search(pattern2, line).group()[1:-2] # omit "\n" and ">"
        if typo == "content":
            contents.append(_text)
        elif typo == "contenttitle":
            _text = _text[:-1] if (_text != "" and _text[-1] == " (") else
_text
            titles.append(_text)

for i in range(len(titles)):
    title, doc = titles[i], contents[i]
    seg_list = list(jieba.cut(doc, cut_all=True)) # precise mode
    counter = Counter(seg_list)
    for pos, word in enumerate(set(seg_list)):
        print("{}@{}\t{}".format(word, counter[word], title))
```

3.3 reducer.py

reducer要做的事情，是从stdin中读入mapper的输出，按照（单词，所有包含该单词的文档）形式组织好数据并输出。**注意：partitioner在输出key/value pair的时候，会在key与value之间加一个“\t”**

```
import sys
import re
from collections import defaultdict

out_dict = defaultdict(list)

for line in sys.stdin:
    key, value = line.split("\t")
    word, count = key.split("@")
    out_dict[word].append(value[:-1]+"@"+count)

for word, value_list in out_dict.items():
```

```
    titles = "<SEP>".join(value_list)
    print("{}\t{}".format(word, titles))
```

3.4 run.sh

在前文中我们有提到，为了将同一个word对应的key-value pair放到同一个reduce任务下，需要实现一个新的Partitioner：从（word, word_frequency）中取出word，以word作为key进行分区。在Hadoop Streaming中这个具体应该怎么实现呢?

很简单。Hadoop有一个工具类org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner， 它在应用程序中很有用。Map/reduce框架用这个类切分map的输出， 切分是基于key值的前缀。

因此，我们只需要在执行程序的时候指明所用的partitioner和分区，就可以实现“按照一部分key的值进行排序”的功能。这里，为了避免每次启动hadoop的时候都要反复手动输入shell脚本，需要一个.sh文件来保存会在终端进行的输入。

```
#!/usr/bin/env bash
rm ./part-00000
hdfs dfs -rm -r bigdata/output/

# sprcify -D <options> at the beginning
# otherwise hadoopstreaming will fail to recognize these options
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.9.2.jar \
-D map.output.key.field.separator=@ \
-D mapreduce.partition.keypartitioner.options=-k1,1 \
-D
mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapreduce.lib.part
ition.KeyFieldBasedComparator \
-D mapreduce.partition.keycomparator.options=-k2nr \
-input bigdata/inputs/news_tensite_xml.smarty.dat \
-output bigdata/output/ \
-mapper "python3 mapper.py" \
-reducer "python3 reducer.py" \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \

# get file from HDFS to local and visualize them
hdfs dfs -get bigdata/output/part-00000 ./
head -100 part-00000
```

3.5 query.py

文档查询函数最简单的实现是：以contenttile为文本名。实现根据查询关键词，返回相关文档名（序列）。但是这里，我们想要实现的是：根据输入的word序列，查找与该序列最相关的文档，并输出文档名。

因此，我们需要使用TF-IDF模型来构建文档向量，

$$docVec[i,:] = (w_{i1}, \ldots, w_{ij}, \ldots, w_{iN})$$

其中

$$w_{ij} = TF_{ij} * IDF_j$$

$$TF_{ij} = \frac{\text{单词}j\text{在文档}i\text{中出现频率}}{\text{文档}i\text{的词数}}, IDF_j = \frac{\text{总文档数目}}{\text{含有单词}j\text{的文档数目}}$$

由此，我们只需要按照输入单词序列构建查询向量，然后计算查询向量与文档向量的cosine相似度即可。

具体代码实现如下：(这里设置了early break=5，也就是只print前5个最相关的文档名)

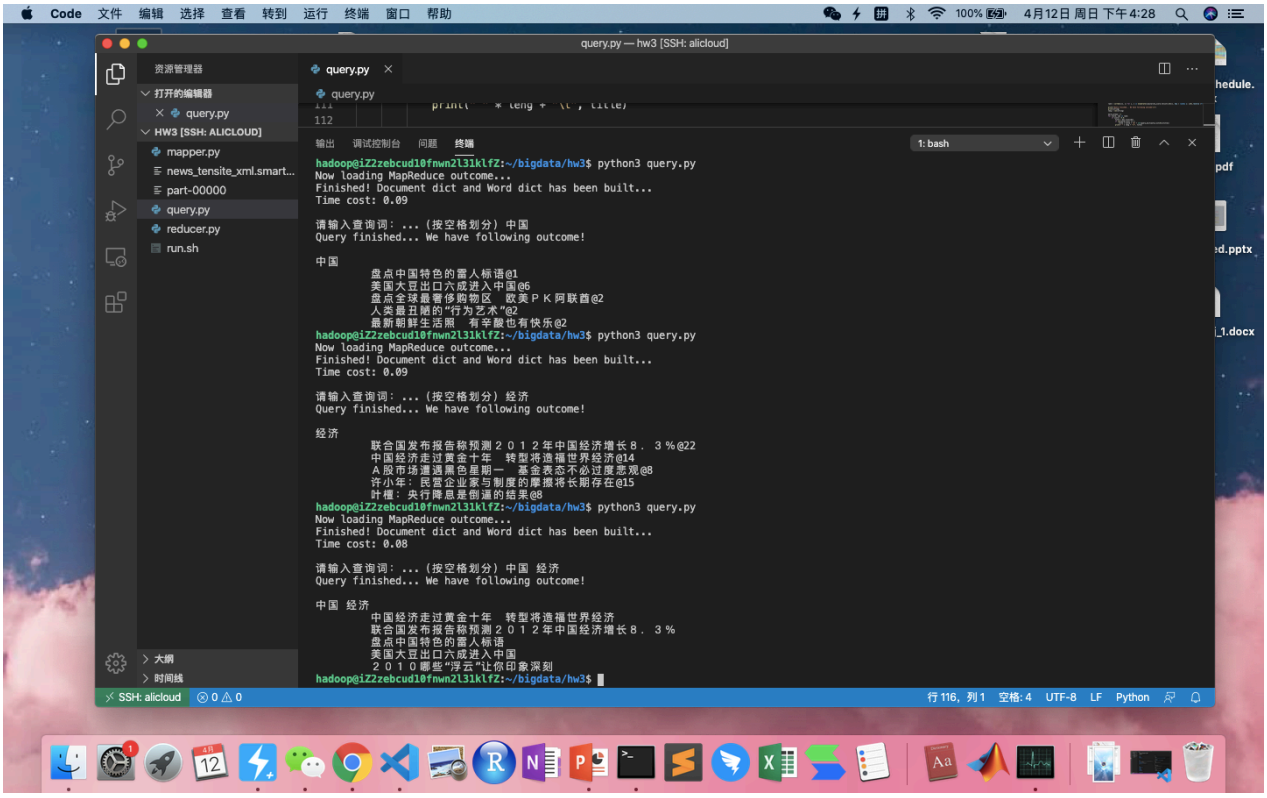
```
import numpy as np
from time import time

start = time()
print("Now loading MapReduce outcome...")
query_dict = {} # word->[doc_names : freq]
doc_count = {} # 文档次数统计

with open("./part-00000", "r", encoding="utf-8") as f:
    lines = f.readlines()

    for line in lines:
        word, rest = line.split("\t")
        titles = rest.split("<SEP>")

        word2doc = {}
        for title in titles:
            doc, freq = title.strip().split("@")
            word2doc[doc] = int(freq)
            doc_count[doc] = int(freq) + doc_count.get(doc, 0)
        query_dict[word] = word2doc

doc_list = list(doc_count.keys())
doc_idx = {doc:i for i, doc in enumerate(doc_list)}

word_list = list(query_dict.keys())
word_idx = {word:i for i, word in enumerate(word_list)}

M, N = len(doc_list), len(word_list)
print("Finished! Document dict and Word dict has been built...")
print("Time cost: {:.2f}\n".format(time()-start))


def construct_TFIDF(M, N, doc_dict, doc_idx, word_dict, word_idx):
    tf_shape = (M, N)
    idf_shape = (1, N)

    def TF_matrix():
        m = np.zeros(tf_shape, dtype=float)
```

```
        for word in word_dict:
            for doc in word_dict[word]:
                m[doc_idx[doc], word_idx[word]] = word_dict[word][doc] /
doc_dict[doc]

        return m

    def IDF_matrix():
        m = np.zeros(idf_shape, dtype=float)

        for word in word_dict:
            c = len(list(word_dict[word].keys()))
            m[0, word_idx[word]] = np.log(N/c)

        m[m < 1e-6] = 1

        return np.repeat(m, M, axis=0) # shape(M, N)

    return TF_matrix() * IDF_matrix()


def input2vec(word_list, word_idx, shape):
    out = np.zeros(shape, dtype=float)

    for word in word_list:
        if word in word_idx:
            out[word_idx[word]] = 1

    return out


def compute_Cosine(query_vec, matrix):
    assert matrix.shape[1] == query_vec.shape[0]

    inner_product = np.dot(matrix, query_vec) # shape(M,1)
    norm1 = np.linalg.norm(matrix, ord=2, axis=1, keepdims=True) *
np.linalg.norm(query_vec)

    cosine_similarity = inner_product / norm1
    return cosine_similarity.transpose()


if __name__ == "__main__":
    # get the query
    string = input("请输入查询词: ... (按空格划分) ")
    word_list = string.split(" ")
    query_vec = input2vec(word_list, word_idx, shape=(N, 1))

    # compute tf-idf
```

```
    tfidf_matrix = construct_TFIDF(M, N, doc_count, doc_idx, query_dict,
word_idx)

    # compute cosine relevance
    similarity_score = compute_Cosine(query_vec, tfidf_matrix) #shape(1, M)
    rank = sorted([(s, i) for i, s in enumerate(similarity_score.tolist()
[0])], key = lambda x: x[0],reverse =True)

    print("Query finished... We have following outcome!\n")
    print(string)
    leng = len(string)

    early_break = 5
    for score, idx in rank:
        if score > 0.0:
            title = doc_list[idx]
            if len(word_list) == 1:
                title = title + "@" + str(\
                    query_dict.get(word_list[0], {}).get(title, 0)
                    )
            print(" " * leng + "\t", title)

            if early_break is not None:
                early_break -= 1
                if early_break <= 0:break
```

3.5 Outcome

query结果：



我们可以看到，在分别输入“中国”和“经济”的时候，程序会分别返回前5个最相关的文档名称。

而同时输入“中国 经济”的时候，程序返回的第一个文档名称是“**中国经济走过黄金十年 转型将造福世界经济**”。充分说明我们实现的模型完成了**按照文档相似性查找和排序**。

4. 总结

在本次实践中，学习了如何避免linux出现中文乱码问题。学会了基础的python和hadoop streaming交互，了解了hadoop streaming的基本使用。学习了TF-IDF文档模型的使用。进一步加深了对MapReduce框架的理解。