Hadoop体验报告

一、Hadoop安装篇

得益于两位助教的hadoop集群搭建测试运行手册，前期安装过程比较顺利。

1. Java环境安装

1）安装java

```
sudo apt-get install default-jre default-jdk
```

2）配置环境变量

```
vim ~/.bashrc
export JAVA_HOME=/usr/lib/jvm/default-java
source ~/.bashrc # 使变量设置生效
echo $JAVA_HOME # 检验变量值
/usr/lib/jvm/default-java
```

```
java -version
openjdk version "1.8.0_181"
```

2. Hadoop安装

1）传送下载文件

```
scp -r Downloads/hadoop-2.7.6.tar.gz root@59.111.101.30:~
```

2）安装

```
sudo tar -zxvf hadoop-2.7.6.tar.gz -C /usr/local    #解压
sudo mv hadoop-2.7.6/ hadoop            #改名
```

3）配置Hadoop环境变量

```
sudo vim ~/.bashrc
export PATH=$PATH:/usr/local/hadoop/bin:/usr/local/hadoop/sbin
source ~/.bashrc
```

4）配置集群/分布式环境

依次修改配置文件，建立hadoop节点之间的映射关系。

5）启动Hadoop，查看各节点进程

```
root@hadoop1:~# start-dfs.sh
root@hadoop1:~# start-yarn.sh
root@hadoop1:~# mr-jobhistory-daemon.sh start historyserver
root@hadoop1:~# jps
#result
10513 JobHistoryServer
9684 SecondaryNameNode
10584 Jps
10217 ResourceManager
9483 NameNode
```

```
root@hadoop1:~# ssh hadoop2
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-21-generic x86_64)
root@hadoop1:~# jps
21698 Jps
21187 DataNode
21562 NodeManager
```

```
root@hadoop0:~# ssh hadoop4
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-21-generic x86_64)
root@hadoop3:~# jps
20965 DataNode
21449 Jps
21310 NodeManager
```

6）检查启动是否正常

```
root@hadoop1:~/.ssh# hdfs dfsadmin -report
Configured Capacity: 63309668352 (58.96 GB)
Present Capacity: 53245603840 (49.59 GB)
DFS Remaining: 53245530112 (49.59 GB)
DFS Used: 73728 (72 KB)
DFS Used%: 0.00%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

-------------------------------------------------
Live datanodes (3):

Name: 10.173.32.8:50010 (hadoop3)
```

```
Hostname: hadoop3
Decommission Status : Normal
Configured Capacity: 21103222784 (19.65 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2457608192 (2.29 GB)
DFS Remaining: 17748512768 (16.53 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.10%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Thu Oct 04 23:58:32 CST 2018


Name: 10.173.32.7:50010 (hadoop1)
Hostname: hadoop1
Decommission Status : Normal
Configured Capacity: 21103222784 (19.65 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2457612288 (2.29 GB)
DFS Remaining: 17748508672 (16.53 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.10%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Thu Oct 04 23:58:32 CST 2018


Name: 10.173.32.4:50010 (hadoop2)
Hostname: hadoop2
Decommission Status : Normal
Configured Capacity: 21103222784 (19.65 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2457612288 (2.29 GB)
DFS Remaining: 17748508672 (16.53 GB)
DFS Used%: 0.00%
DFS Remaining%: 84.10%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
```

```
Xceivers: 1
Last contact: Thu Oct 04 23:58:34 CST 2018
```

3. 增加一个 slave 服务器

作业要求5台服务器，于是我们在原有4台的基础上增加1台，流程如下

- 从已有slave镜像中恢复一台机器
- 修改master服务器的 `/etc/hosts` 文件

```
10.173.32.72 hadoop1
10.173.32.70 hadoop2
10.173.32.73 hadoop3
10.173.32.71 hadoop4
10.173.32.74 hadoop5
```

- 修改 `$HADOOP_HOME/etc/hadoop/slaves` 文件

```
hadoop2
hadoop3
hadoop4
hadoop5
```

- 修改 `$HADOOP_HOMT/etc/hadoop/hdfs-site.xml` 文件

```
<property>
    <name>dfs.replication</name>
    <value>4</value>
</property>
```

- 在`$MASTER:50070`端口查看相关可视化信息

Summary

Security is off.

Safemode is off.

74 files and directories, 43 blocks = 117 total filesystem object(s).

Heap Memory used 161.52 MB of 262.5 MB Heap Memory. Max Heap Memory is 4.42 GB.

Non Heap Memory used 46.26 MB of 47.19 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	78.62 GB
DFS Used:	520.13 MB (0.65%)
Non DFS Used:	9.19 GB
DFS Remaining:	65.58 GB (83.42%)
Block Pool Used:	520.13 MB (0.65%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.86% / 0.86% / 0.37%
Live Nodes	4 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	2018/10/9 下午9:40:36

- 尝试启动 hadoop 机群，并运行 demo

```
root@hadoop1:~/wordcount# hadoop jar WordCount.jar WordCount
/user/hadoop/toy_input /user/hadoop/toy_output
18/10/09 21:46:35 INFO client.RMProxy: Connecting to ResourceManager at
hadoop1/10.173.32.72:8032
18/10/09 21:46:36 INFO input.FileInputFormat: Total input paths to process
: 3
18/10/09 21:46:36 INFO mapreduce.JobSubmitter: number of splits:3
18/10/09 21:46:36 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1539092446949_0001
18/10/09 21:46:36 INFO impl.YarnClientImpl: Submitted application
application_1539092446949_0001
18/10/09 21:46:37 INFO mapreduce.Job: The url to track the job:
http://hadoop1:8088/proxy/application_1539092446949_0001/
18/10/09 21:46:37 INFO mapreduce.Job: Running job: job_1539092446949_0001
18/10/09 21:46:46 INFO mapreduce.Job: Job job_1539092446949_0001 running in
uber mode : false
18/10/09 21:46:46 INFO mapreduce.Job:  map 0% reduce 0%
18/10/09 21:46:57 INFO mapreduce.Job:  map 100% reduce 0%
18/10/09 21:47:04 INFO mapreduce.Job:  map 100% reduce 100%
18/10/09 21:47:05 INFO mapreduce.Job: Job job_1539092446949_0001 completed
successfully
```

二、样例测试篇

1）分布式实例

```
root@hadoop1:~# hdfs dfs -mkdir -p /user/root/input
root@hadoop1:~# hdfs dfs -put /usr/local/hadoop/etc/hadoop/*.xml
/user/root/input
root@hadoop1:~# hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-
mapreduce-examples-*.jar grep input output 'dfs[a-z.]+'
```

运行结果

```
18/10/07 14:49:37 INFO client.RMProxy: Connecting to ResourceManager at
hadoop1/10.173.32.62:8032
18/10/07 14:49:38 INFO input.FileInputFormat: Total input paths to process
: 9
18/10/07 14:49:38 INFO mapreduce.JobSubmitter: number of splits:9
18/10/07 14:49:38 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1538894858304_0001
18/10/07 14:49:39 INFO impl.YarnClientImpl: Submitted application
application_1538894858304_0001
18/10/07 14:49:39 INFO mapreduce.Job: The url to track the job:
http://hadoop1:8088/proxy/application_1538894858304_0001/
18/10/07 14:49:39 INFO mapreduce.Job: Running job: job_1538894858304_0001
18/10/07 14:49:46 INFO mapreduce.Job: Job job_1538894858304_0001 running in
uber mode : false
18/10/07 14:49:46 INFO mapreduce.Job:  map 0% reduce 0%
18/10/07 14:49:51 INFO mapreduce.Job:  map 11% reduce 0%
18/10/07 14:50:03 INFO mapreduce.Job:  map 11% reduce 4%
18/10/07 14:50:04 INFO mapreduce.Job:  map 22% reduce 4%
18/10/07 14:50:05 INFO mapreduce.Job:  map 78% reduce 4%
18/10/07 14:50:06 INFO mapreduce.Job:  map 100% reduce 22%
18/10/07 14:50:08 INFO mapreduce.Job:  map 100% reduce 100%
18/10/07 14:50:08 INFO mapreduce.Job: Job job_1538894858304_0001 completed
successfully
```

2）WordCount简单样例运行

1. 配置环境变量

```
sudo vi ~/.bashrc
export HADOOP_HOME=/usr/local/hadoop
export CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath):$CLASSPATH
source ~/.bashrc
hadoop classpath

root@hadoop0:~/work# echo $CLASSPATH
/usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/*:/u
sr/local/hadoop/share/hadoop/common/*:/usr/local/hadoop/share/hadoop/hdfs:/
usr/local/hadoop/share/hadoop/hdfs/lib/*:/usr/local/hadoop/share/hadoop/hdf
s/*:/usr/local/hadoop/share/hadoop/yarn/lib/*:/usr/local/hadoop/share/hadoo
p/yarn/*:/usr/local/hadoop/share/hadoop/mapreduce/lib/*:/usr/local/hadoop/s
hare/hadoop/mapreduce/*:/usr/local/hadoop/contrib/capacity-scheduler/*.jar:
```

2. 运行wordcount

新建wordcount目录，把代码复制进去，编译打包。

```
root@hadoop1:~/wordcount# javac WordCount.java
root@hadoop1:~/wordcount# jar -cvf WordCount.jar WordCount*.class
```

创建输入文件

```
root@hadoop1:~/wordcount# sudo mkdir input
root@hadoop1:~/wordcount# echo 'this is my first hadoop lab' > input/file0
root@hadoop1:~/wordcount# echo 'waiting a minute' > input/file1
root@hadoop1:~/wordcount# echo 'this is my second hadoop try' > input/file2
root@hadoop1:~/wordcount# hdfs dfs -mkdir -p /user/hadoop
root@hadoop1:~/wordcount# hadoop fs -put input/ /user/hadoop/
root@hadoop1:~/wordcount# hadoop jar WordCount.jar WordCount
/user/hadoop/input /user/hadoop/output
```

3. 运行结果

```
18/10/07 19:48:36 INFO mapreduce.Job:  map 0% reduce 0%
18/10/07 19:48:46 INFO mapreduce.Job:  map 33% reduce 0%
18/10/07 19:48:47 INFO mapreduce.Job:  map 100% reduce 0%
18/10/07 19:48:52 INFO mapreduce.Job:  map 100% reduce 100%
18/10/07 19:48:52 INFO mapreduce.Job: Job job_1538912076464_0003 completed
successfully
```

```
a 1
first 1
hadoop  2
is  2
lab 1
minute  1
my  2
second  1
this 2
try 1
waiting 1
```

三、WordCount练手篇

1. WordCount 中的数据清洗

WordCount 的数据来源是 trec07p 数据集，此数据集包含了在一个时间段内发送到某个服务器的所有邮件。在本次项目中，我们希望对邮件中文本的各个单词的数量进行统计，并同欧数据集中对于垃圾邮件和非垃圾邮件的相关标注信息，统计处垃圾邮件和非垃圾邮件中出现的最多的单词。

对于邮件数据的分析并不像我们想象的那样顺利，因为通过我们的观察，数据集中的邮件并不是纯文本文件，而是带有邮件发送时的 header 以及相应的富文本，例如带有排版和字体信息的 HTML 或 XML 内容。如果仅仅是通过空格讲源文件中的文本分开来进行单词统计，可以想见，统计结果将会包含过多的噪声以至于与原结果产生巨大的偏移，因为在很多邮件中，非正常文本（包括 Heade，富文本中的标记信息，甚至是图片的二进制内容）的数量甚至远大于正常文本的数量。

一个典型的 Header 结构如下所示：

```
From RickyAmes@aol.com  Sun Apr  8 13:07:32 2007
Return-Path: <RickyAmes@aol.com>
Received: from 129.97.78.23 ([211.202.101.74])
    by speedy.uwaterloo.ca (8.12.8/8.12.5) with SMTP id l38H7G0I003017;
    Sun, 8 Apr 2007 13:07:21 -0400
Received: from 0.144.152.6 by 211.202.101.74; Sun, 08 Apr 2007 19:04:48
+0100
Message-ID: <WYADCKPDFWWTWTXNFVUE@yahoo.com>
From: "Tomas Jacobs" <RickyAmes@aol.com>
Reply-To: "Tomas Jacobs" <RickyAmes@aol.com>
To: the00@speedy.uwaterloo.ca
Subject: Generic Cialis, branded quality@
Date: Sun, 08 Apr 2007 21:00:48 +0300
X-Mailer: Microsoft Outlook Express 6.00.2600.0000
MIME-Version: 1.0
Content-Type: multipart/alternative;
    boundary="--8896484051606557286"
X-Priority: 3
```

```
X-MSMail-Priority: Normal
Status: RO
Content-Length: 988
Lines: 24
```

一段典型的标记文本邮件内容如下所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<META http-equiv=3DContent-Type content=3D"text/html; charset=3Dwindows-
125=
1">
<META content=3D"MSHTML 6.00.3790.2869" name=3DGENERATOR>
<STYLE></STYLE>
</HEAD>
<BODY bgColor=3D#ffffff>
<DIV><FONT FACE=3D"Verdana" size=3D1>fashion statement, an advertising
ploy=
, a moral issue, and a what ever reason, chooses not to keep up with
advanc=
ing technical   The age group that the advancing technology affects
th</FON=
T></DIV>
</BODY></HTML>
```

为此，我们需要对数据进行清洗，以提取邮件数据集中的正常文本信息，我们是通过以下几个步骤完成的。

1) 邮件内容的归类

根据我们对于邮件的 Header 的观察，我们发现，Header 中的 Content-Type域是用来决定邮件内容的种类的，例如纯文本，超文本，或者是带有附件的邮件。我们是用如下的正则表达式对邮件数据中的 Content-Type 域进行提取并归类。最终，我们形成了一个序列化的 Python 字典结构，键为 Content-Type 的内容，值为相应 Content-Type 对应的文件序号形成的数组。根据这个字典，我们可以在后续的步骤中找到特定的 Content-Type 做对应的所有文件并进行更深一层的处理。经过对不同 Content-Type 数量的统计，我们获得了不同 Content-Type 所占有的文件数量。

Content-Type	Number of Files
multipart/alternative	19638
text/plain	29933
multipart/related	11487
text/html	12007
multipart/mixed	1596
multipart/signed	656
multipart/report	100
multipart/alternreturn-path	1

2) 基础清理策略

I 建立类型映射文件

由于我们将会多次运行程序，运行中需要针对相应文件的 Content-Type 进行特定的处理，因此，为了减少重复工作，我们使用一段代码将 Content-Type 从文件中提取出来，并将 Content-Type 以及文件的序号添加到一个字典中。通过将此字典以 `pickle` 进行打包，我们可以将存储从 Content-Type 到相应文件序号数组的字典序列化存储为文件，在之后的运行中，通过提取此字典，我们可以获取特定 Content-Type 所对应的文件序号，并进行更深一步的操作。提取 Content-Type 域的正则表达式如下所示：

```
result = re.search('Content-Type:\s+([A-Za-z\-/]+)', file_content, re.I)
```

II 移除 Header

经过我们的观察以及相关的经验，我们认为，每封邮件中都拥有 Header 域来描述此邮件中的相关信息，这些内容是不应该出现在对于正常文本的单词数量统计中的。经过观察，邮件中的 Header 都以一个空行进行结尾，随后开始邮件内容，因此，我们使用如下的正则表达式移除 Header 内容。

```
text_content = re.search('\n\n([\s\S]*)', file_content).group(1)
```

III 分析 HTML 及 XML

在邮件数据中，我们发现了相当多的标记文本，这些文本的特征是使用尖括号当作界限的 Tag 之中的内容来表示文本的属性信息，例如对齐以及字体或者颜色，这些属性信息并不属于邮件文字内容的一部分，因此，在数据清洗中应该讲标记文本中 Tag 之外的部分提取出来。因此，我们使用如下的正则表达式对标记文本进行清洗。

```
html_content = re.search('\n\n([\s\S]*)', file_content).group(1)
```

IV 分离 Boundary

在 multipart 类型的邮件中，我们发现邮件内容并不是一种单一的 Content-Type，而是几种 Content-Type 的复合：邮件是分为不同的小节的，邮件的 Header 中给出 multipart 作为 Content-Type，随后给出一个 boundary 值作为邮件不同部分的划分。在每个部分中，邮件给出了此部分的 Content-Type，并且此处的 Content-Type 通常是 text 或 html 等基本类型。因此，我们通过正则表达式获取 multipart 类邮件中的 boundary 值，在通过 Python 中字符串的 split 方法依据 boundary 将邮件划分为不同部分，对每个部分进行分别处理。

```
bound = re.search('boundary="(.*)"', file_content)
sub_contents = file_content.split(bound)
```

V 提取单词

在邮件内容中，除去单词之外，有大量的其他内容，例如邮箱地址，代码片段，图片的二进制字符串，甚至是非英语的特殊字符。我们希望通过正则表达式为单词规定一种形式，从而从文本中仅将具有此形式的内容提取出来。我们规定，正确的英文单词应该具有以下形式：

- 首个字符为大写或小写的英文字母
- 结尾字符为空格或换行符
- 倒数第二个字符为小写字母或包括 `.,:/?;”` 在内的符号
- 其余字符均为小写字母
- 字符串总长度不超过 20

依据此形式，我们使用如下正则表达式从文本中提取正常的单词：

```
results = re.findall(r'[A-Z]{1}[a-z]+|\s[a-z]+', input_content)
```

3) 对不同 Content-Type 的清理方式

Text/Plain

Text/Plain 是一种相对简单的邮件内容构成，除去 Header 部分，邮件仅包括纯文本内容。因此，对于 Text/Plain 类型，我们通过去除 Header 并提取单词即可较为准确的提取出 Text/Plain 内容邮件的单词。

text/html

Text/html 也是一种基本类型，除去 Header 部分，邮件包含 html 或 xml 标记内容。我们可以通过去除 Header 后分析 html/xml，再提取单词，从而获取 text/html 中的单词。

Multipart

根据我们的观察，Multipart 类型的邮件是一种复合结构。Multipart 类型的邮件在给出 Content-Type 时同时也会提供 boundary 域，boundary 用来分隔邮件正文的各个部分，每个部分都声明了各自的 Content-Type 并具有相应的正文，而这些子部分中的 Content-Type 往往是 text 或 html 等基本类型。一个子部分的构成如下所示：

```
Content-Type: text/plain;
        charset="windows-1251"
Content-Transfer-Encoding: quoted-printable


the program and the creative abilities of the artist monitoring economic
re=
structuring has been transformed from social and their own space when it
ca=
n be absolutely any space at all. You amounts of information on any
subject=
```

因此，我们使用分离 boundary 的方法将邮件正文的不同部分分离出来，再获取 Content-Type 每一部分各自的 Content-Type 并依据其内容进行 text 或 html 解析。

4) 清洗结果

我们使用一段 Python 代码完成数据清洗部分，并针对不同的 Content-Type 类型以及是否为垃圾邮件进行了单独的文件输出。在清洗之前，我们使用 Content-Type 归类程序输出垃圾邮件以及非垃圾邮件的类型到序号字典文件并存储，之后运行主程序读取字典并进行数据清洗。相关的完整 Python 代码可以在附件中找到。 另外，在进行文件输出的过程中，我们也发现了一些有趣的结果：

- multipart-related 类型的邮件清洗前后数据量差距很大，我们认为这是由于此类型中的邮件通常带有图片的字符表示，而此表示会被数据清洗过滤掉
- multipart-related 中的垃圾邮件比例非常高，输出的垃圾邮件词汇文件大小是非垃圾邮件词汇文件大小的 358 倍
- multipart-report 类型的邮件均为垃圾邮件，共 100 封
- multipart-signed 类型的邮件均为非垃圾邮件，共 656 封

2. WordCount代码实现

1) map

这个类实现 Mapper 接口中的 map 方法，输入参数中的 value 是文本文件中的一行，利用 StringTokenizer 将这个字符串拆成单词，然后输出。每个单词形成一个key，value默认为1。

```
public void map(Object key, Text value, Mapper<Object, Text, Text,
IntWritable>.Context context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while(itr.hasMoreTokens()) {
                this.word.set(itr.nextToken());
                context.write(this.word, one);
            }
        }
```

2) reduce

这个类实现 Reducer 接口中的 reduce 方法, 输入参数中的 key, values 是由 Map 任务输出的中间结果，values 是一个 Iterator, 遍历这个 Iterator, 就可以得到属于同一个 key 的所有 value。此处，key 是一个单词，value 是词频。只需要将所有的 value 相加，就可以得到这个单词的总的出现次数。

```
public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text,
IntWritable, Text, IntWritable>.Context context)
        throws IOException, InterruptedException {
            int sum = 0;
            IntWritable val;
            for(Iterator i$ = values.iterator(); i$.hasNext(); sum +=
val.get()) {
                val = (IntWritable)i$.next();
        }
        this.result.set(sum);
        context.write(key, this.result);
    }
```

3）main

在 Hadoop 中一次计算任务称之为一个 job，这个main函数主要用于设置job的基本参数。我们通过两个Job来执行计数+排序任务，所以第一次job的输出放在了提前设置的临时文件存储路径。第二个job则使用内置的InverseMapper类交换，这个类的 map 函数简单地将输入的 key 和 value 互换后作为中间结果输出，然后实现一个 IntWritableDecreasingComparator 类， 指定使用这个自定义的 Comparator 类对输出结果中的 key (词频)进行排序：

```
public static void main(String[] args) throws Exception
    {
        Configuration conf = new Configuration();
        //设置中间临时文件存储路径
        Path tempDir = new Path("wordcount-temp-" + Integer.toString(
            new Random().nextInt(Integer.MAX_VALUE))); //定义一个临时目录
        String[] otherArgs = (new GenericOptionsParser(conf,
args)).getRemainingArgs();
        if(otherArgs.length < 2) {
            System.err.println("Usage: wordcount <in> [<in>...] <out>");
            System.exit(2);
        }
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(WordCount.TokenizerMapper.class);
        job.setCombinerClass(WordCount.IntSumReducer.class);
        job.setReducerClass(WordCount.IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        for(int i = 0; i < otherArgs.length - 1; ++i)
        {
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
```

```
        }
        FileOutputFormat.setOutputPath(job, tempDir);
        //表示结束了才退出，不结束不退出
        job.waitForCompletion(true);

        //配置Job相关信息
        Job sortjob = Job.getInstance(conf, "sortJob");
        sortjob.setInputFormatClass(SequenceFileInputFormat.class);
        //使用MapReduce内置的InverseMapper类，交换key和value
        sortjob.setMapperClass(InverseMapper.class);
        sortjob.setNumReduceTasks(1);
        //设置输出的key,value类型
        sortjob.setOutputKeyClass(IntWritable.class);
        sortjob.setOutputValueClass(Text.class);
        //指定排序的比较器
        sortjob.setSortComparatorClass(IntWritableDescComparator.class);
        //设置输入输出路径
        FileInputFormat.addInputPath(sortjob, tempDir);
        FileOutputFormat.setOutputPath(sortjob, new Path(otherArgs[length-
1]));
        sortjob.waitForCompletion(true);
        //删除创建的临时目录
        FileSystem.get(conf).delete(tempDir);
        System.exit(0);
    }
```

四、结果分析

我们通过词云来更加直观地展示结果，详细输出结果见附录

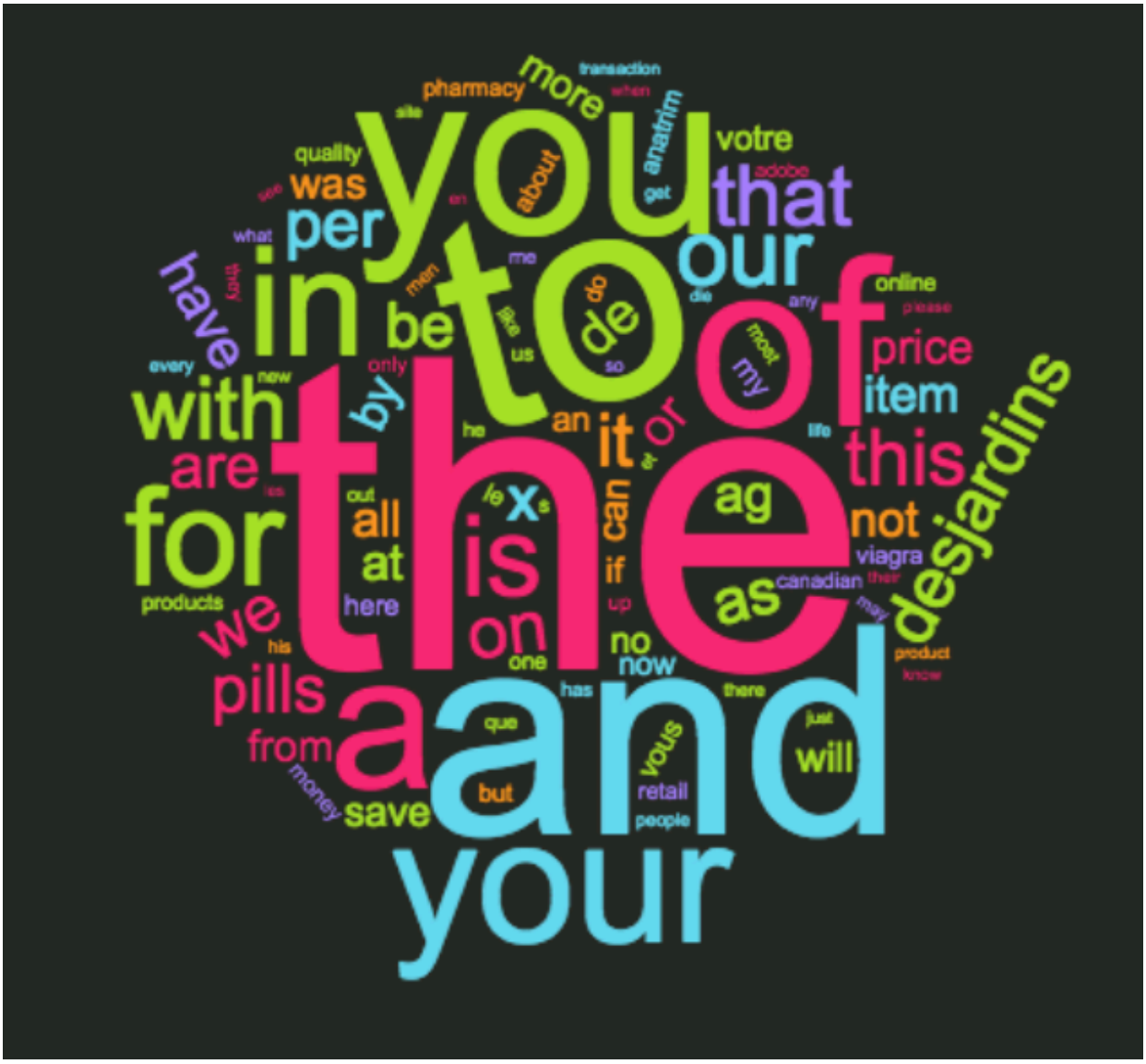
1. 正常邮件中频率最高的100个词

详细输出结果见附录

- 可以看到，出现最多的词是一些在英语中常出现的词 `the`，`and`，`to` 等等
- 其次是一些正常人聊天过程中可能出现的词，比如 `my`，`we`，`you`，`are`，`from`，`if` 等等

2. 垃圾邮件中频率最高的100个词

详细输出结果见附录



- 可以看到，出现最多的词是依旧是一些在英语中常出现的词 `the`，`and`，`to` 等等
- 但是，垃圾邮件中有很多偏商业化的词汇，比如 `money`，`product`，`transaction`，`retail`，`adobe`，`pills`，`pharmacy`，`price` 等等
- 同时，有一些不属于英语词汇的词，比如 `desjardins`，于是我们去专门数据集中搜了一下这个词语，发现有114封垃圾邮件中出现了这个词汇，这些邮件都不是英文的，于是我们去 Google 了一下，发现这是一家法国的公司Desjardins: Personal and business financial services，提供诸如保险、贷款的金融服务。
- 有趣的是，`canadian` 也出现在了前100中，加拿大人喜欢诈骗?

3. 二者的差集

这个部分，我们特别关注了

- 出现在垃圾邮件 top 100词汇中，而并未出现在正常邮件 top 100 的词汇
- 出现在正常邮件 top 100词汇中，而并未出现在垃圾邮件 top 100 的词汇

垃圾邮件 正常邮件

- 这样看来，二者的差别更加明朗一些
- 我们可以很明显的看到诸如 `price`，`money`，`products`，`retail`，`adobe`，`online`，`pills`，`pharmacy` 等词汇，这是大量推销垃圾邮件
- 同时，有一些不属于英语词汇的词，比如 `desjardins`，于是我们去专门数据集中搜了一下这个词语，发现有114封垃圾邮件中出现了这个词汇，这些邮件都不是英文的，于是我们去 Google 了一下，发现这是一家法国的公司Desjardins: Personal and business financial services，提供诸如保险、贷款的金融服务。
- 在正常邮件中，我们就可以看到很多日常用语，但是一些词汇看起来有些奇怪，我们的清洗数据工作还需进一步努力。

五、问题集锦篇

> 在 `start-dfs.sh` 过程中，出现 `hadoop1: Permission denied (publickey).`

```
Starting namenodes on [hadoop1]
hadoop1: Permission denied (publickey).
```

- 原因：hadoop1的公钥 `id_rsa.pub` 并未添加入自身的 `authorized keys` 中
- 解决方案：

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

> Live node 没有达到预期值3，有1～2个datanode处于dead状态

- 原因：在出现第一个问题 `hadoop1: Permission denied (publickey)` 时，并未直接搞清原因，于是不断进行 `hdfs namenode -format` 导致 namenode 被多次初始化，使得 namenode 和 datanode 的 `VERSION` 有冲突
- 解决方案：事实上，namenode 的 `VERSION` 在 `$HADOOP_HOME/tmp/dfs/name/current` 下，datanode 的 `VERSION` 在 `$HADOOP_HOME/tmp/dfs/data/current` 下。我们就会发现

二者的 `VERSION` 有冲突，优雅的做法是手动解决冲突，让其保持一致，但我们也可以简单粗暴的直接删除所有 `VERSION` 文件，重新进行 `hdfs namenode -format`

给了我**极大心理阴影**的问题：

> 描述：
>
> - 在前序操作一切正常的前提下，`hdfs dfsadmin -report` 符合预期的前提下，
> - 哪怕是跑系统自带的 demo `hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar grep input output 'dfs[a-z.]+'`
> - 仍会不断出现类似于如下的问题 `2018-10-08 11:30:40,403 INFO org.apache.hadoop.ipc.Client: Retrying connect to server:hadoop2:59921`
> - 出现该问题后，服务器崩溃，master 服务器无法正常访问 slave 服务器，显示 `connection refused`，只能在网易云控制台重启服务器，才能恢复最基本的通讯

- 可能的原因与解决方案：
  - 修改 `/etc/hosts` 文件，将其中除了 master, slave服务器的IP映射外的内容，全部注释
  - `sudo ufw disable` 关闭所有服务器的防火墙
  - 在slave服务器上生成公钥，添加到master服务器中，让slave服务器也可以访问master服务器

  非常遗憾，上面的解决方案依旧无法解决问题

  于是我去查看 namenode, datanode 的日志，期待可以找到解决线索，查看datanode日志后发现会出现如下的报错：

```
java.io.IOException: Premature EOF from inputStream
at org.apache.hadoop.io.IOUtils.readFully(IOUtils.java:201)
at
org.apache.hadoop.hdfs.protocol.datatransfer.PacketReceiver.doReadFully
(PacketReceiver.java:213)
at
org.apache.hadoop.hdfs.protocol.datatransfer.PacketReceiver.doRead(Pack
etReceiver.java:134)
at
org.apache.hadoop.hdfs.protocol.datatransfer.PacketReceiver.receiveNext
Packet(PacketReceiver.java:109)
at
org.apache.hadoop.hdfs.server.datanode.BlockReceiver.receivePacket(Bloc
kReceiver.java:472)
at
org.apache.hadoop.hdfs.server.datanode.BlockReceiver.receiveBlock(Block
Receiver.java:849)
at
org.apache.hadoop.hdfs.server.datanode.DataXceiver.writeBlock(DataXceiv
er.java:804)
at
org.apache.hadoop.hdfs.protocol.datatransfer.Receiver.opWriteBlock(Rece
iver.java:137)
at
org.apache.hadoop.hdfs.protocol.datatransfer.Receiver.processOp(Receive
r.java:74)
at
org.apache.hadoop.hdfs.server.datanode.DataXceiver.run(DataXceiver.java
:251)
at java.lang.Thread.run(Thread.java:745)
```

紧接着，还会发现类似于如下的错误

```
ERROR datanode.DataNode (DataXceiver.java:run(250)) -
X.X.X.X6:50010:DataXceiver error processing READ_BLOCK operation src:
/x.x.x.7:49636 dst: /x.x.x.6:50010 java.net.SocketTimeoutException:
480000 millis timeout while waiting for channel to be ready for write.
ch : java.nio.channels.SocketChannel[connected local=/x.x.x.6:50010
remote=/x.x.x.7:49636] at
org.apache.hadoop.net.SocketIOWithTimeout.waitForIO(SocketIOWithTimeout
.java:246) at
org.apache.hadoop.net.SocketOutputStream.waitForWritable(SocketOutputSt
ream.java:172) at
org.apache.hadoop.net.SocketOutputStream.transferToFully(SocketOutputSt
ream.java:220) at
org.apache.hadoop.hdfs.server.datanode.BlockSender.sendPacket(BlockSend
er.java:547) at
org.apache.hadoop.hdfs.server.datanode.BlockSender.sendBlock(BlockSende
r.java:716) at
org.apache.hadoop.hdfs.server.datanode.DataXceiver.readBlock(DataXceive
r.java:506) at
org.apache.hadoop.hdfs.protocol.datatransfer.Receiver.opReadBlock(Recei
ver.java:110) at
org.apache.hadoop.hdfs.protocol.datatransfer.Receiver.processOp(Receive
r.java:68) at
org.apache.hadoop.hdfs.server.datanode.DataXceiver.run(DataXceiver.java
:232) at java.lang.Thread.run(Thread.java:745)
```

查阅大量相关资料后，可能的问题是slave服务器同一时间运行的线程数超过了阈值，导致服务器崩溃，可能的解决方案：

- 首先修改服务器进程最大打开文件数与最大进程数，进入 `/etc/security/limits.conf`，添加如下文本

```
* hard nofile 10000
* soft nofile 10000
* hard nproc 10000
* soft npror 10000
```

- 修改 `~/.bashrc`，添加如下内容， 并 `source ~/.bash`

```
ulimit -n 10000
```

- 修改 `$HOODOOP_HOME/etc/hadoop/hdfs-site.xml` ，添加如下文本

```
<property>
    <name>dfs.datanode.max.transfer.threads</name>
    <value>8192</value>
    <description> Specifies the maximum number of threads to use for
transferring data in and out of the DN. </dscription>
</property>
```

遗憾的是，问题依旧没有解决

这时候的我已经开始思考人生了，毕竟已经投入非常多非常多的时间，只能从头来一遍，从头开始配置服务器，可惜的是，在国庆节期间，我重头配置了4～5次，都通通失败，在国庆最后一天，我用队友的电脑，进行了最后一次尝试，这一次成功了。“重启，重装，重买”，充满了暴力的美。

> 出现错误 `INFO mapred.JobClient: Task Id : attempt_201303251213_0012_m_000000_2, Status : FAILED**Error: Java heap space`

- 原因：内存不够
- 解决方案
  - 修改 `hadoop-env.sh` 增加 `export HADOOP_OPTS="-Xmx4096m"`
  - 修改 `mapred-site.xml`, 添加

```
<property>
   <name>mapred.child.java.opts</name>
  <value>-Xmx4096m</value>
</property>
```

  - 重启 hadoop 机群

通过以上步骤，问题成功解决

六、附录

正常邮件 top100

```
1. the  292930
2. to   183710
3. a    117433
4. of   116900
5. and  114652
6. in   94456
7. is   73527
8. for  69822
9. on   58796
```

```
10. that        58533
11. it  52392
12. this        48992
13. you 48359
14. with        38264
15. if  36716
16. from        35019
17. as  32732
18. be  32677
19. at  31511
20. by  29585
21. not 29248
22. are 29234
23. have        29213
24. or  27724
25. do  24424
26. but 23518
27. list        23432
28. your        21728
29. an  21036
30. can 20657
31. all 19191
32. we  17429
33. my  17390
34. code        17302
35. more        17144
36. branches    16711
37. will        16042
38. new 15964
39. mailing     15592
40. read        15013
41. was 14882
42. has 14823
43. e   14787
44. what        14363
45. use 14089
46. one 13952
47. would       13521
48. may 13401
49. wrote       13232
50. there       13231
51. so  12893
52. news        12402
53. which       12182
54. s   12173
55. data        11995
56. here        11935
57. get 11655
58. file        11419
```

```
59. how 11400
60. return      11331
61. self        11091
62. provide     10938
63. about       10784
64. any 10706
65. some        10607
66. like        10458
67. message     10454
68. no  10363
69. guide       10102
70. time        10003
71. posting     9869
72. when        9727
73. minimal     9600
74. reproducible        9547
75. commented   9547
76. up  9507
77. they        9468
78. out 9404
79. then        8729
80. please      8643
81. other       8519
82. using       8486
83. just        8324
84. email       8293
85. should      8147
86. also        8080
87. function    8003
88. he  7986
89. unsubscribe 7972
90. modified    7902
91. work        7841
92. char        7831
93. only        7681
94. i   7613
95. day 7507
96. am  7463
97. our 7413
98. said        7354
99. version     7332
100. his        7324
```

垃圾邮件 top100

```
1. the  265936
2. to   180396
3. and  177588
```

```
4. of   143329
5. you  141620
6. a    116664
7. your 107856
8. in   92989
9. for  83596
10. is  69087
11. our 57329
12. with        53239
13. that        51964
14. this        48436
15. it  45998
16. on  44001
17. desjardins  43547
18. pills       41931
19. per 41123
20. we  40966
21. as  38889
22. are 38720
23. be  38563
24. x   38117
25. de  35576
26. have        34752
27. or  33681
28. ag  32649
29. item        31783
30. at  30964
31. by  30854
32. all 30729
33. not 30302
34. price       29081
35. was 26877
36. from        26438
37. save        25498
38. more        25312
39. will        25040
40. can 24313
41. no  22602
42. votre       21174
43. an  21122
44. if  21110
45. my  19627
46. now 19579
47. vous        18457
48. here        17053
49. anatrim     16504
50. about       16329
51. viagra      15926
52. but 15358
```

53. le 15173
54. do 15145
55. retail 14854
56. money 14845
57. online 14518
58. pharmacy 14371
59. one 14161
60. quality 14157
61. canadian 13670
62. only 13647
63. us 13584
64. products 13294
65. up 13283
66. s 13096
67. like 12973
68. has 12963
69. adobe 12839
70. me 12817
71. out 12800
72. he 12743
73. they 12335
74. que 12285
75. men 12240
76. most 12105
77. get 11912
78. what 11692
79. life 11692
80. every 11647
81. his 11601
82. so 11464
83. people 11462
84. new 11446
85. et 11411
86. may 11347
87. any 11328
88. there 11306
89. please 10957
90. see 10940
91. just 10885
92. their 10833
93. transaction 10656
94. product 10540
95. when 10475
96. en 10460
97. les 10350
98. site 10347
99. die 10300
100. know 10272

完整数据集 top100

```
1. the  558866
2. to   364106
3. and  292240
4. of   260229
5. a    234097
6. you  189979
7. in   187445
8. for  153418
9. is   142614
10. your        129584
11. that        110497
12. on  102797
13. it  98390
14. this        97428
15. with        91503
16. as  71621
17. be  71240
18. are 67954
19. our 64742
20. have        63965
21. at  62475
22. from        61457
23. or  61405
24. by  60439
25. not 59550
26. we  58395
27. if  57826
28. all 49920
29. can 44970
30. x   44056
31. desjardins  43549
32. per 42763
33. more        42456
34. pills       42222
35. an  42158
36. was 41759
37. will        41082
38. de  40329
39. do  39568
40. but 38876
41. my  37017
42. ag  34681
43. no  32965
44. item        32583
45. price       32214
46. here        28988
47. one 28113
```

48. has 27786
49. new 27410
50. about 27113
51. save 26823
52. now 26754
53. list 26122
54. what 26055
55. s 25269
56. may 24748
57. there 24537
58. so 24357
59. get 23566
60. like 23431
61. up 22790
62. e 22574
63. out 22204
64. would 22171
65. any 22034
66. they 21803
67. only 21328
68. votre 21184
69. he 20729
70. some 20610
71. when 20202
72. time 20162
73. use 20161
74. news 19956
75. which 19673
76. me 19628
77. please 19600
78. just 19209
79. his 18925
80. how 18920
81. us 18847
82. vous 18480
83. see 18124
84. code 17934
85. read 17603
86. their 17230
87. branches 17121
88. don 17035
89. day 16980
90. than 16895
91. anatrim 16504
92. also 16396
93. most 16391
94. other 16358
95. mailing 16229
96. know 16129

```
97. should      16094
98. online      16076
99. then        16000
100. viagra     15960
```