

Hadoop 培训练习

一 实验环境搭建

本实验环境采用 Cloudera quickvm 来实现，大家有条件也可以自己从 cloudera 网站下载。

1.1 请采用 ssh 客户端登录 cloudera 所在的服务器：

- Ip 地址：
- 端口号：
- 用户名：root/Intel1234

1.2 登录完成后请依次启动 mysql,clouder-scm-server,cloudera-scm-agent，在 shell 下依次执行：

```
service ntpd start
service mysqld start
service cloudera-scm-server start
service cloudera-scm-agent start
检查服务的状态
service cloudera-scm-server status
service cloudera-scm-agent status
```

Cloudera 管理进程的监听端口默认是 7180，cloudera-scm-server 的启动过程比较慢，需要 1 分钟左右，请检查 7180 端口是否已经处于监听状态：

```
#netstat -an |grep 7180
```

当出现如下状态的显示，代表 hadoop 已经正常启动。

```
[root@quickstart ~]# service cloudera-scm-server status
```

```
cloudera-scm-server (pid 822) is running...
```

```
[root@quickstart ~]# service cloudera-scm-agent status
```

```
cloudera-scm-agent (pid 2067) is running...
```

```
[root@quickstart ~]# netstat -an |grep 7180
```

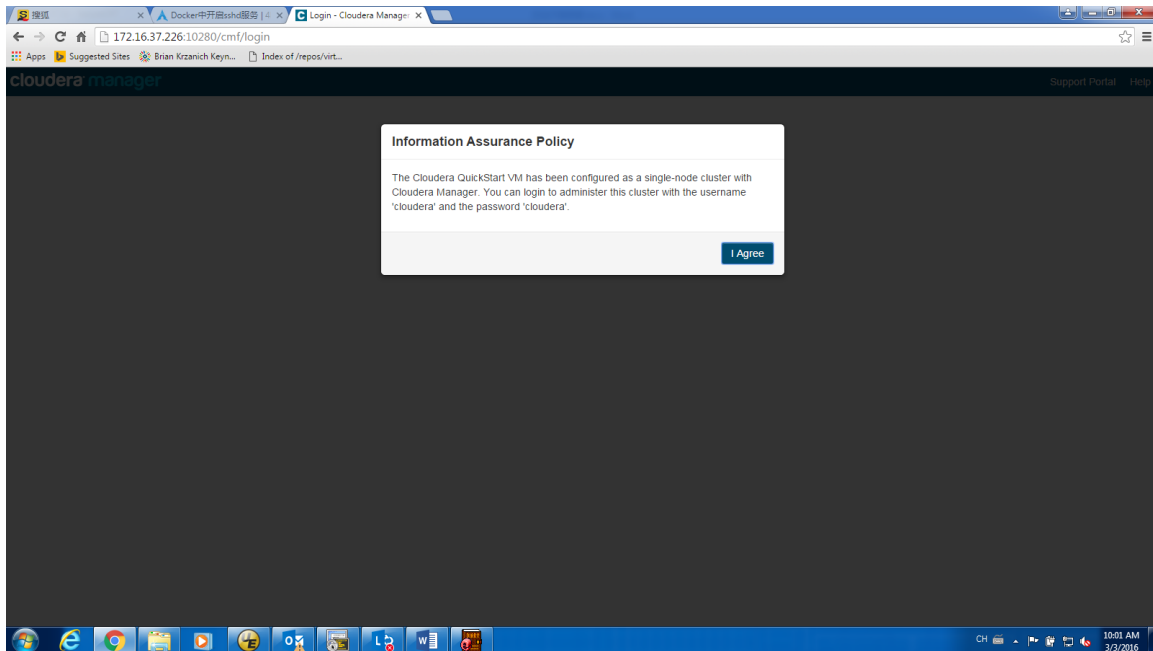
```
tcp    0    0.0.0.0:7180        0.0.0.0:*        LISTEN
```

1.3 请通过您客户端的浏览器访问 cloudera manager 的管理界面

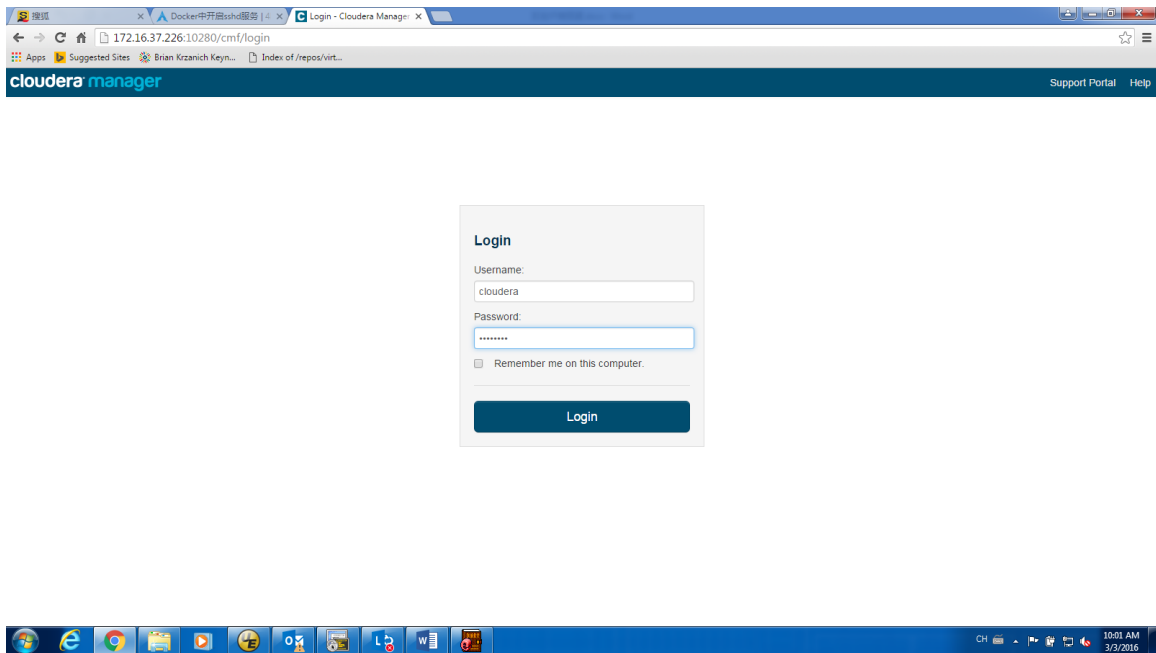
Ip:

Port:

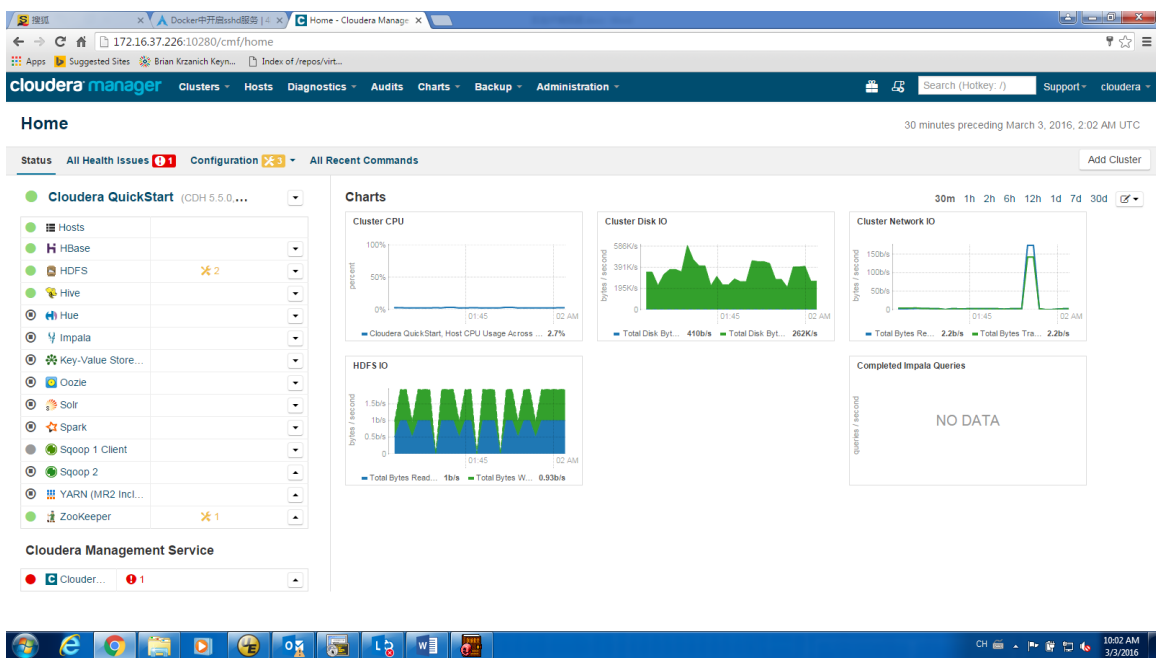
Username: cloudera/cloudera



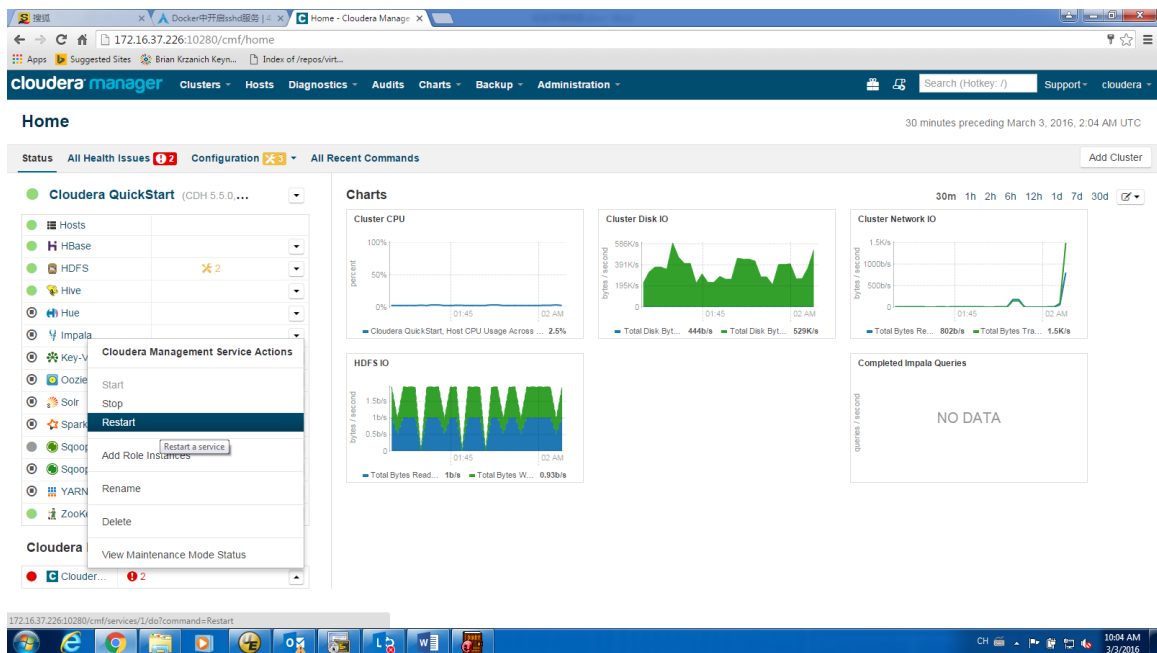
点击 I agree



用 cloudera/cloudera 登录



首先请重新启动最左下方的 cloudera management service



然后依次启动 hdfs,yarn,hive,zookeeper,hbase,spark(由于实验环境内存限制, 请不要启动全部服务)。

1.4 请用 ssh 登录 cloudera 服务器

```
#su - hdfs
```

```
[root@quickstart ~]# su - hdfs
```

```
-bash-4.1$ hadoop fs -ls /
```

```
Found 5 items
```

```
drwxrwxrwx - hdfs supergroup 0 2015-11-19 01:45 /benchmarks
```

```
drwxr-xr-x - hbase supergroup 0 2016-03-03 01:17 /hbase
```

```
drwxrwxrwt - hdfs supergroup 0 2016-03-03 01:14 /tmp
```

```
drwxr-xr-x - hdfs supergroup 0 2015-11-19 01:47 /user
```

```
drwxr-xr-x - hdfs supergroup 0 2015-11-19 01:47 /var
```

```
-bash-4.1$ hadoop fs -mkdir /mydata
```

```
-bash-4.1$ hadoop fs -ls /
```

```
Found 6 items
```

```
drwxrwxrwx - hdfs supergroup    0 2015-11-19 01:45 /benchmarks
drwxr-xr-x - hbase supergroup    0 2016-03-03 01:17 /hbase
drwxr-xr-x - hdfs supergroup    0 2016-03-03 02:14 /mydata
drwxrwxrwt - hdfs supergroup    0 2016-03-03 01:14 /tmp
drwxr-xr-x - hdfs supergroup    0 2015-11-19 01:47 /user
```

jps 查看机器上的进程。

jps

创建一个自己的目录，浏览 hdfs 根目录下文件夹，至此实验环境搭建完成，如果大家时间比较富裕，可以浏览熟悉一下 cloudera manager 的功能（此内容不属于本次培训，本次培训还是以开源软件为核心）。

二 HDFS 文件系统

2.1 ssh 登录到您的 cloudera 虚拟机环境，su 成 hdfs

su - hdfs，hdfs 对 hadoop 相当于 linux 的 root，在本次实验中，由于是独占式环境，我们的实验都采用 hdfs 用户。

hdfs dfs (老版本采用 hadoop fs)

hdfs dfs -ls /（老版本采用 hadoop fs -ls /）

2.2 首先我们将 testdata 测试文件 copy 到 hadoop 文件系统中，如果上一步的 hadoop 文件系统/mydata 没有建立，请首先创建该文件夹：

Hdfs dfs -mkdir /mydata

然后 copy 文件，检查是否成功。

hadoop fs -copyFromLocal /home/hdfs/testdata /mydata/

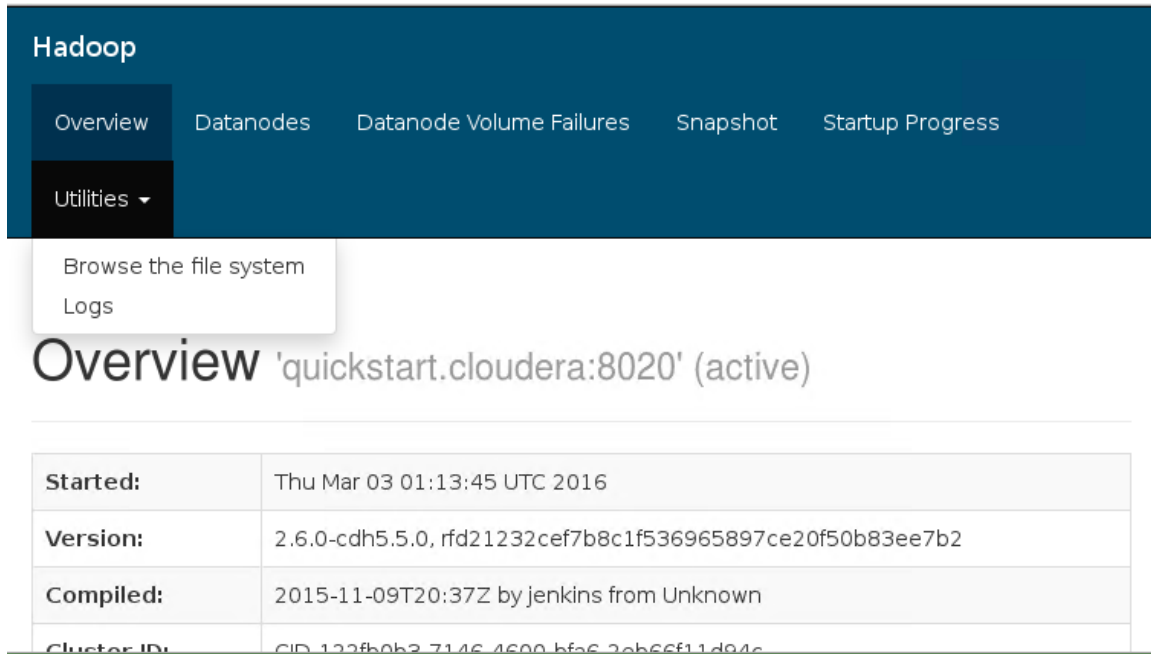
hadoop fs -ls /mydata/

hadoop fs -tail /mydata/testdata

2.3 通过 namenode 的 web UI 界面检查我们的 testdata 信息。

2.3.1 启动浏览器，输入：http://ip:50070/

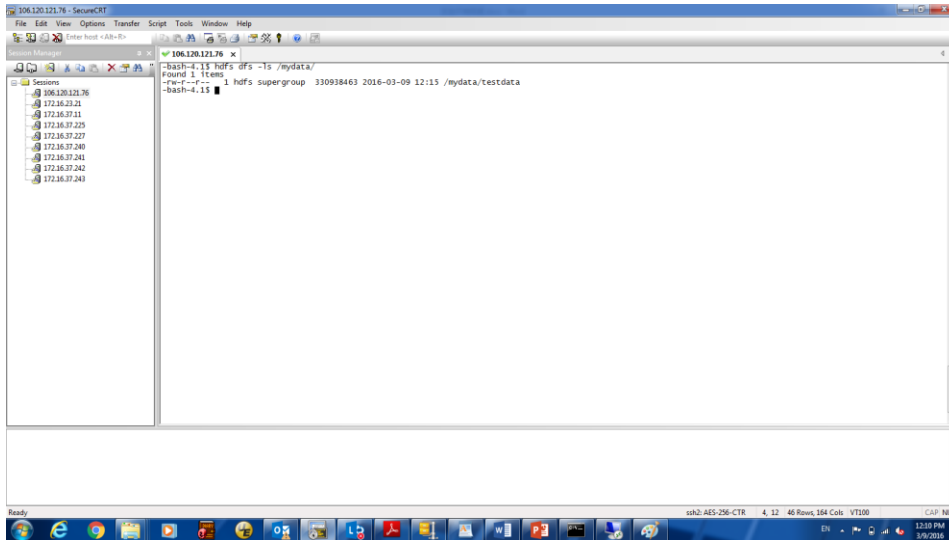
2.3.2. Click Utilities > “Browse the file system,” then navigate to the /user/training/weblog directory.



Started:	Thu Mar 03 01:13:45 UTC 2016
Version:	2.6.0-cdh5.5.0, rfd21232cef7b8c1f536965897ce20f50b83ee7b2
Compiled:	2015-11-09T20:37Z by jenkins from Unknown
Cluster ID:	CID-122fb0b3-7146-4600-bfa6-2eb66f11d84c

2.3.3 Verify that the permissions for the testdata file are identical to the permissions you observed when you ran the `hdfs dfs -ls` command.

2.3.5 Now select the testdata file in the NameNode Web UI file browser to bring up the File Information window.



2.3.6 Observe that for each block in the file, only a single host quickstart.cloudera is listed under Availability. This indicates that the file is replicated to a single host. CDH sets the replication number to 1 for a pseudo distributed cluster, When you build a cluster with multiple nodes later in this course, the replication number will be 3, and you will see three hosts listed under Availability.

File information - testdata

[Download](#)

Block information -- Block 0

Block ID: 1073743068

Block Pool ID: BP-501095374-10.0.0.16-1447897441387

Generation Stamp: 2244

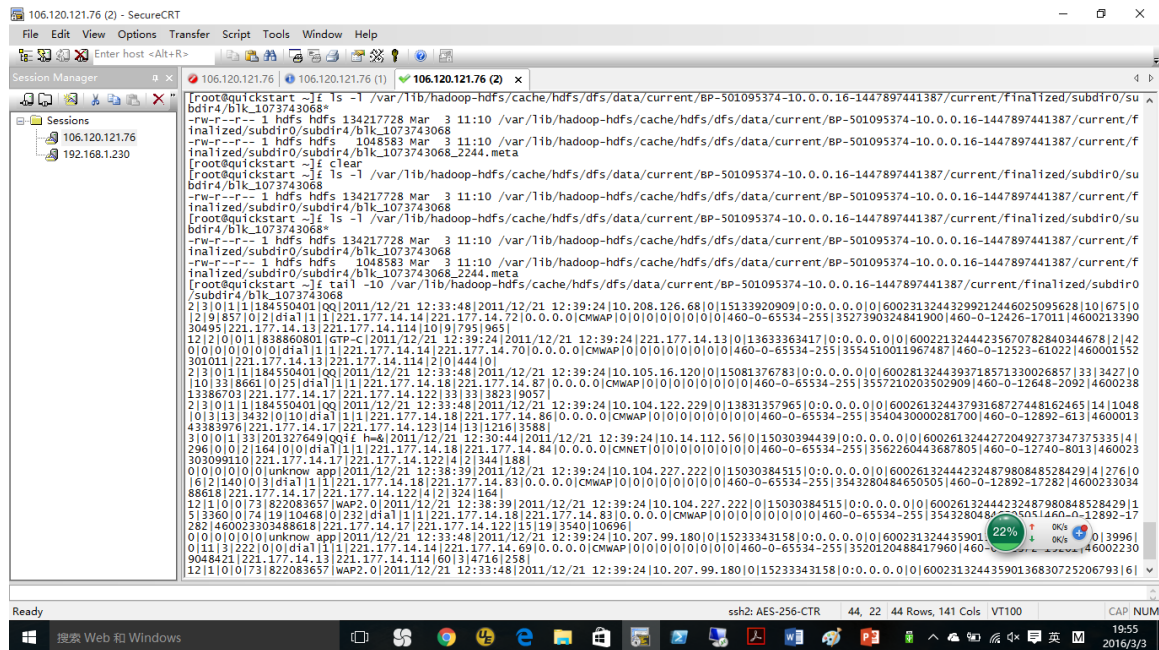
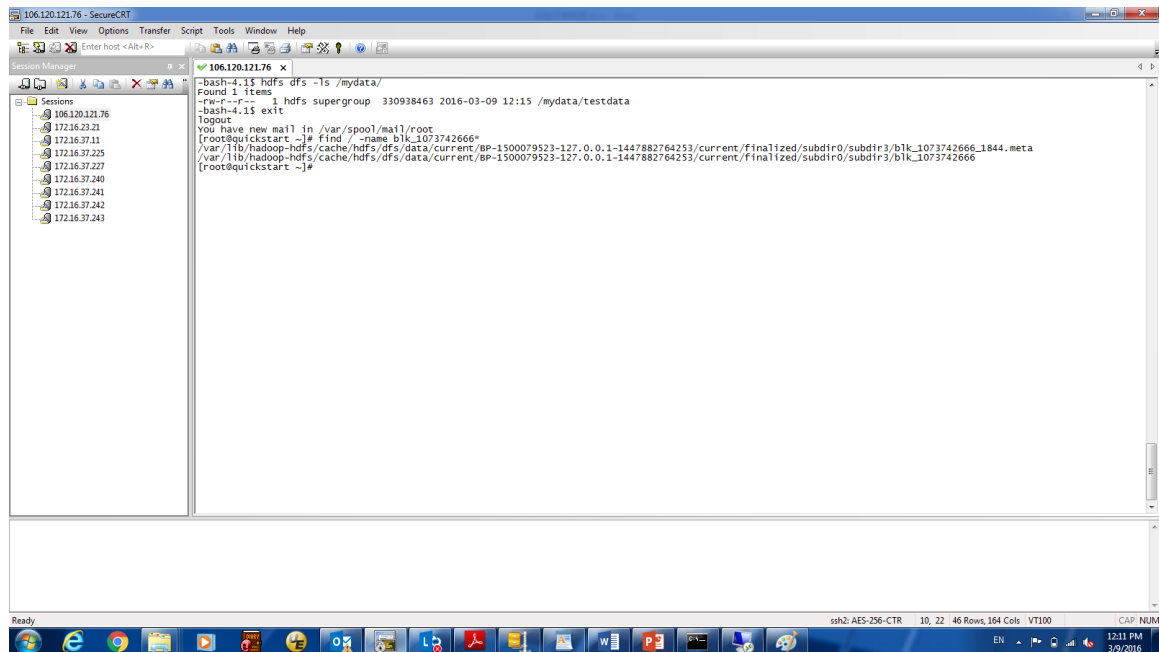
Size: 134217728

Availability:

- quickstart.cloudera

2.3.7 选择 block 0 并记录下 block id 字段的值。

2.3.8 回到 ssh 界面，按照下图去查看 testdata 的底层存储状况。

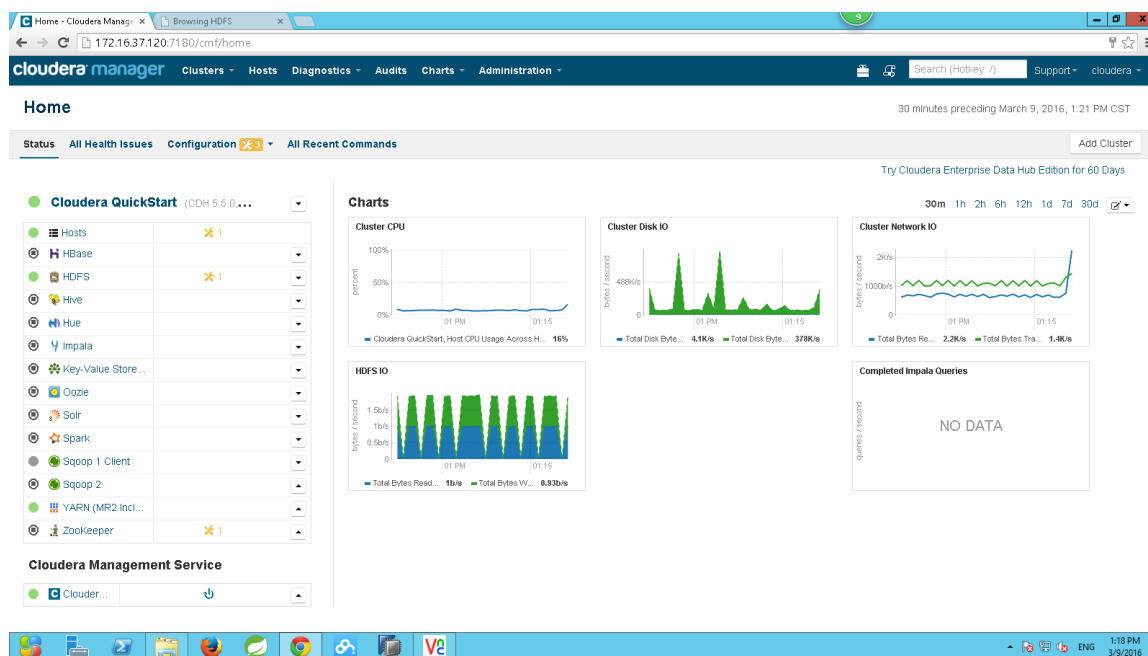


可以看到，hadoop 文件系统每一块的大小，以及元数据的信息，每个 block 当中的内容（大家可以考虑如何在 hadoop 中对文件进行加密）。

三 运行 map-reduce job

hadoop 会将需要的代码和配置文件压缩成 jar 包,通过 `hadoop jar` 命令发布到 hadoop 集群, `hadoop jar` 命令最少包含两个参数, 一个是 jar 包的全路径名称, 另外一个执行程序的名字 (一个 jar 包中可能包含多个可执行程序)。在本实验中, 我们运行两个 map reduce 任务, 一个是传统的 wordcount, 另外一个是一个传统的 ETL 任务用 map/reduce 生成。

3.1 确保 hdfs 和 yarn 服务已经启动, 如果没有启动, 请从图形界面启动。



3.2 `su - hdfs` 如果用其它用户, 需要手动设置环境变量和 hdfs 权限

3.3 `hadoop jar /usr/jars/hadoop-mapreduce-examples-2.6.0-cdh5.5.0.jar wordcount /input /output/wordcount`

3.4 如果中途执行错误, 需要手动删除 hdfs 中/output 目录下生成的目录, 否则可能报错, 当采用不同用户的时候, 特别注意权限问题。

3.5 观察系统执行结果

The image displays two screenshots of a SecureCRT terminal window, showing the execution of a Hadoop wordcount job on a cluster.

Top Screenshot: The terminal shows the execution of the command `hadoop jar /usr/jars/hadoop-mapreduce-examples-2.6.0-cdh5.5.0.jar wordcount /input /output/wordcount`. The logs indicate that the job was submitted successfully and is running. The output shows the following statistics:

- File System Counters:
 - FILE: Number of bytes read=712
 - FILE: Number of bytes written=232157
 - FILE: Number of read operations=0
 - FILE: Number of large read operations=0
 - FILE: Number of write operations=0
 - HDFS: Number of bytes read=664
 - HDFS: Number of bytes written=613
 - HDFS: Number of read operations=0
 - HDFS: Number of large read operations=0
 - HDFS: Number of write operations=2
- Job Counters:
 - Launched map tasks=1
 - Launched reduce tasks=1
 - Data-local map tasks=1
 - Total time spent by all maps in occupied slots (ms)=428672
 - Total time spent by all reduces in occupied slots (ms)=514432
 - Total time spent by all map tasks (ms)=3349
 - Total time spent by all reduce tasks (ms)=4019
 - Total vcore-seconds taken by all map tasks=3349
 - Total vcore-seconds taken by all reduce tasks=4019
 - Total megabyte-seconds taken by all map tasks=428672
 - Total megabyte-seconds taken by all reduce tasks=514432
- Map-Reduce Framework:
 - Map input records=1
 - Map output records=87
 - Map output bytes=901
 - Map output materialized bytes=708
 - Input split bytes=111
 - Combine input records=87

Bottom Screenshot: The terminal shows the execution of the command `hadoop fs -ls /output`. The output shows the following file listing:

Permissions	Owner	Group	Size	Modification Time	Path
drwxr-xr-x	hdfs	super	0	2016-03-03 12:44	/output/wordcount
-rw-r--r--	hdfs	super	0	2016-03-03 12:44	/output/wordcount/_SUCCESS
-rw-r--r--	hdfs	super	613	2016-03-03 12:44	/output/wordcount/part-r-00000

采用浏览器登录到 <http://ip:8088>,查看任务执行

3.6 wordcount 源码

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	...
1	0	0	1	0	0 B	3 GB	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	...
0	0	0	1	0	0

23% progress indicator

Show 20 entries

ID	User	Name	Application Type	Queue	StartT

```

import java.io.IOException;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.MapReduceBase;

```

```

import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
/**
 *
 * 描述: WordCount explains by Felix
 * @author Hadoop Dev Group
 */
public class WordCount
{
    /**
     * MapReduceBase 类:实现了 Mapper 和 Reducer 接口的基类（其中的
     方法只是实现接口，而未作任何事情）
     * Mapper 接口：
     * WritableComparable 接口：实现 WritableComparable 的类可以相互
     比较。所有被用作 key 的类应该实现此接口。
     * Reporter 则可用于报告整个应用的运行进度，本例中未使用。
     *
     */
    public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, Text, IntWritable>
    {

```

```

/**
 * LongWritable, IntWritable, Text 均是 Hadoop 中实现的用于封装
Java 数据类型的类，这些类实现了 WritableComparable 接口，
 * 都能够被串行化从而便于在分布式环境中进行数据交换，你可以
将它们分别视为 long,int,String 的替代品。
 */
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

/**
 * Mapper 接口中的 map 方法：
 * void map(K1 key, V1 value, OutputCollector<K2,V2> output, Reporter
reporter)
 * 映射一个单个的输入 k/v 对到一个中间的 k/v 对
 * 输出对不需要和输入对是相同的类型，输入对可以映射到 0 个或
多个输出对。
 * OutputCollector 接口：收集 Mapper 和 Reducer 输出的<k,v>对。
 * OutputCollector 接口的 collect(k, v)方法:增加一个(k,v)对到 output
 */
public void map(LongWritable key, Text value,
                OutputCollector<Text, IntWritable> output, Reporter reporter)
                throws IOException
{
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);

```

```

while (tokenizer.hasMoreTokens())
{
    word.set(tokenizer.nextToken());
    output.collect(word, one);
}
}
}

public static class Reduce extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable>
{
    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException
    {
        int sum = 0;
        while (values.hasNext())
        {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}

public static void main(String[] args) throws Exception
{

```

```

/**
 * JobConf: map/reduce 的 job 配置类，向 hadoop 框架描述 map-
reduce 执行的工作
 * 构造方法：JobConf()、JobConf(Class exampleClass)、
JobConf(Configuration conf)等
 */
JobConf conf = new JobConf(WordCount.class);

conf.setJobName("wordcount");    //设置一个用户定义的 job 名称
conf.setOutputKeyClass(Text.class); //为 job 的输出数据设置 Key 类
conf.setOutputValueClass(IntWritable.class); //为 job 输出设置 value
类

conf.setMapperClass(Map.class);    //为 job 设置 Mapper 类
conf.setCombinerClass(Reduce.class); //为 job 设置 Combiner 类
conf.setReducerClass(Reduce.class); //为 job 设置 Reduce 类

conf.setInputFormat(TextInputFormat.class); //为 map-reduce 任务
设置 InputFormat 实现类

conf.setOutputFormat(TextOutputFormat.class); //为 map-reduce 任
务设置 OutputFormat 实现类

/**
 * InputFormat 描述 map-reduce 中对 job 的输入定义
 * setInputPaths():为 map-reduce job 设置路径数组作为输入列表
 * setInputPath(): 为 map-reduce job 设置路径数组作为输出列表
 */
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));

```

```
        JobClient.runJob(conf);    //运行一个 job
    }
}
```

3.7 针对 testdata 做数据汇总

该程序主要是对 hdfs 下/testdata 按照维度进行汇总的一个程序

3.7.1 请用 ssh 登录服务器

cd /home/hdfs/group,请查看 groupmapper.java,groupreducer.java 和 group.java 三个程序。

```
cat groupmapper.java
```

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
```

```
public class groupmapper extends MapReduceBase implements
Mapper<Object,Text,Text,Text> {
    private Text k = new Text();
    private Text v = new Text();
    private String sline=new String("");
    private String av[];
```



```

    public void map(Object key, Text value,OutputCollector<Text,Text>
output,Reporter reporter) throws IOException {

        sline = value.toString();

        av = sline.split("&");

        k.set(av[0]+"| "+av[19]);

v.set(av[21]+"| "+av[22]+"| "+av[23]+"| "+av[25]+"| "+av[26]+"| "+av[27]+"| "
+av[28]+"| "+av[29]+"| "+av[30]
+"| "+av[31]+"| "+av[32]);

        output.collect(k,v);

    }
}

```

```

-bash-4.1$ cat groupreducer.java

import java.io.IOException;

import java.util.StringTokenizer;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

```

```

import org.apache.hadoop.mapred.Reporter;

public class groupreducer extends MapReduceBase implements
Reducer<Text,Text,NullWritable,Text> {

    private Text sline = new Text();

    public void reduce(Text key,Iterator<Text>
values,OutputCollector<NullWritable,Text> output,Reporter reporter)
throws IOException {

        String[] splita={"","","","","","","","","","","",""},

        int call_dur =0;

        int cfee_bill_dur = 0;

        int long_fee_bill_dur = 0;

        int cfee = 0;

        int lfee = 0;

        int lfee2 = 0;

        int info_fee = 0;

        int disc_cfee = 0;

        int disc_lfee = 0;

        int disc_lfee2 = 0;

        int disc_info_fee = 0;

        int count = 0;

        while (values.hasNext()){

            splita=values.next().toString().split("\\|");

            for(int i=0;i<splita.length;i++)

                if (splita[i]==null)

```

```

        splita[i]="0";
        count +=1;
        call_dur+= Integer.parseInt(splita[0]);
        cfee_bill_dur+= Integer.parseInt(splita[1]);
        long_fee_bill_dur += Integer.parseInt(splita[2]);
        cfee += Integer.parseInt(splita[3]);
        lfee += Integer.parseInt(splita[4]);
        lfee2 += Integer.parseInt(splita[5]);
        info_fee += Integer.parseInt(splita[6]);
        disc_cfee += Integer.parseInt(splita[7]);
        disc_lfee += Integer.parseInt(splita[8]);
        disc_lfee2 += Integer.parseInt(splita[9]);
        disc_info_fee += Integer.parseInt(splita[10]);
    }

    sline.set(key
+"|"+count+"|"+call_dur+"|"+cfee_bill_dur+"|"+long_fee_bill_dur+"|"+
cfee+"|"+lfee+"|"+lfee2+"|"+info_fee+"|"+disc_cfee+"|"+disc_lfee+"|"+
disc_lfee2+"|"+disc_info_fee);

    output.collect(null,sline);
}
}

```

-bash-4.1\$ cat groupby.java

```

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;

```

```

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;

public class groupby {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.out.println("usage: [input] [output]");
            System.exit(-1);
        }
        JobConf conf = new JobConf(groupby.class);
        conf.setJobName("groupCount");
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        conf.setMapperClass(groupmapper.class);
        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(Text.class);
        conf.setReducerClass(groupreducer.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);
        JobClient.runJob(conf);
    }
}

```

}

3.7.2 编译该程序

su – hdfs

cd /home/hdfs/group

javac -classpath `hadoop classpath` *.java

3.7.3 将编译的程序打成 jar 包

Jar cvf group.jar *.class

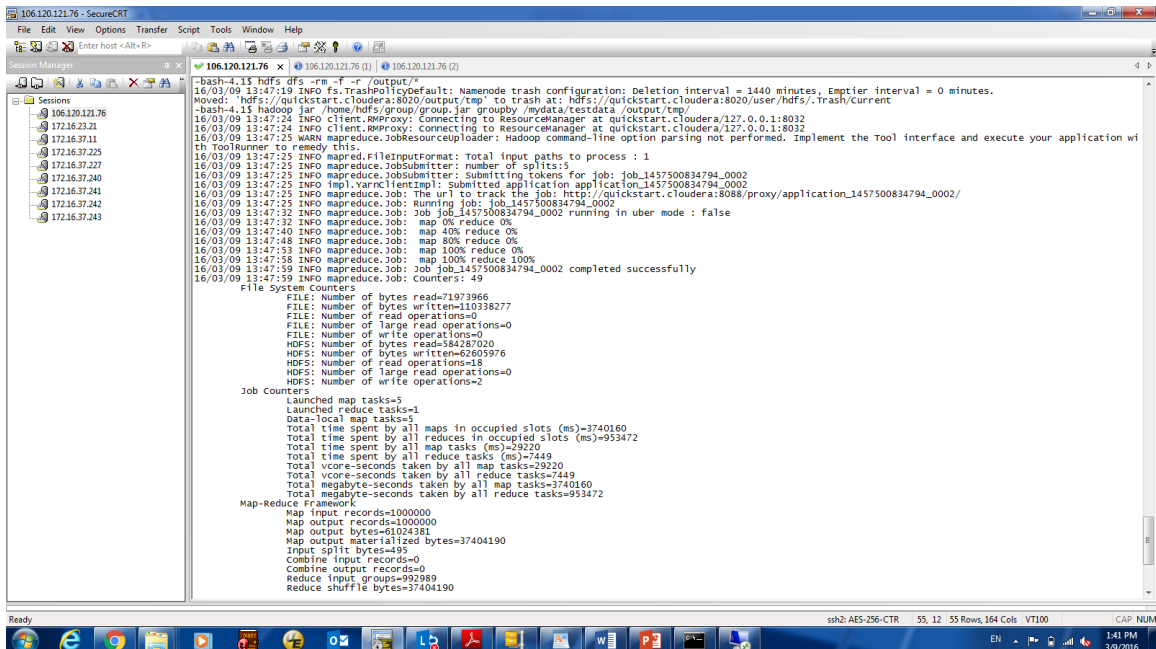
3.8.3 运行该程序

首先将输出目录清空

hdfs dfs -rm -f -r /output/*

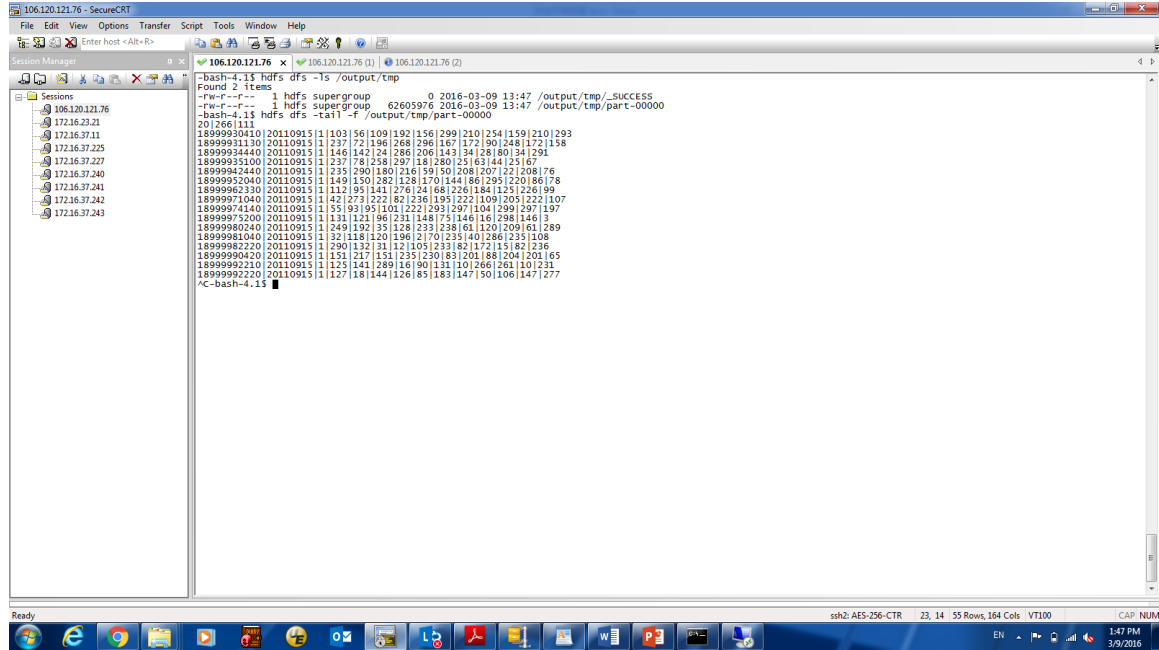
然后运行该程序

hadoop jar /home/hdfs/group/group.jar groupby /mydata/testdata /output/tmp/



```
16/03/09 13:47:19 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 1440 minutes, Empty interval = 0 minutes.
Moved: 'hdfs://quickstart.cloudera:8020/output/tmp' to trash at: hdfs://quickstart.cloudera:8020/user/hdfs/.Trash/current
-bash-4.1$ hadoop jar /home/hdfs/group/group.jar groupby /mydata/testdata /output/tmp/
16/03/09 13:47:24 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera:127.0.0.1:8032
16/03/09 13:47:25 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with
ToolRunner to remedy this.
16/03/09 13:47:25 INFO mapred.fileinputformat: Total input paths to process : 1
16/03/09 13:47:25 INFO mapreduce.JobSubmitter: number of splits:5
16/03/09 13:47:25 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1457500834794_0002
16/03/09 13:47:25 INFO impl.YarnClientImpl: Submitted application application_1457500834794_0002
16/03/09 13:47:25 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1457500834794_0002/
16/03/09 13:47:32 INFO mapreduce.Job: Job job_1457500834794_0002 running in uber mode : false
16/03/09 13:47:32 INFO mapreduce.Job: map 0% reduce 0%
16/03/09 13:47:40 INFO mapreduce.Job: map 40% reduce 0%
16/03/09 13:47:48 INFO mapreduce.Job: map 80% reduce 0%
16/03/09 13:47:53 INFO mapreduce.Job: map 100% reduce 0%
16/03/09 13:47:58 INFO mapreduce.Job: map 100% reduce 100%
16/03/09 13:47:59 INFO mapreduce.Job: Job job_1457500834794_0002 completed successfully
16/03/09 13:47:59 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=71973966
FILE: Number of bytes written=110338277
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=384287020
HDFS: Number of bytes written=62605976
HDFS: Number of read operations=19
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=5
Launched reduce tasks=1
Data-local map tasks=5
Total time spent by all maps in occupied slots (ms)=3740160
Total time spent by all reduces in occupied slots (ms)=953472
Total time spent by all map tasks (ms)=29220
Total time spent by all reduce tasks (ms)=7449
Total vcore-seconds taken by all map tasks=29220
Total vcore-seconds taken by all reduce tasks=7449
Total megabyte-seconds taken by all map tasks=3740160
Total megabyte-seconds taken by all reduce tasks=953472
Map-Reduce Framework
Map input records=1000000
Map output records=1000000
Map output bytes=61024381
Map output materialized bytes=37404190
Input split bytes=485
Combine input records=0
Combine output records=0
Reduce input groups=992989
Reduce shuffle bytes=37404190
```

3.8.4 检查运行结果



至此 mapreduce 部分结束。

四 采用 hive 管理数据

首先在 hdfs 文件系统上创建一个 hive 的用户数据目录。

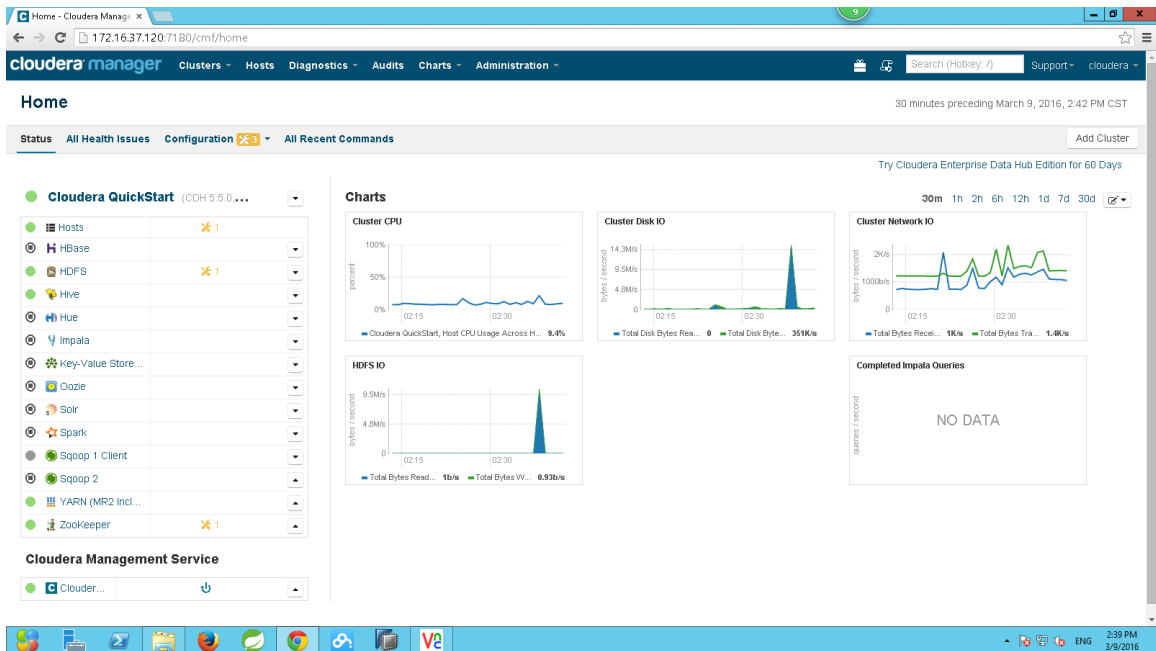
```
Su - hdfs
```

```
Cd /home/hdfs
```

```
Hdfs dfs -mkdir /hivedata
```

```
Hdfs dfs -put hivedata /hivedata
```

4.1 启动 hive 服务器，如果您的 hive 服务器没有启动，请首先启动 hive 服务（hive 依赖底层的 hdfs 和 yarn，在 hive 服务启动前，请首先启动 hdfs 和 yarn）。



4.2 启动 hive 客户端

-bash-4.1\$ beeline

Beeline version 1.1.0-cdh5.5.0 by Apache Hive

**beeline> !connect jdbc:hive2://localhost:10000 hive hive
org.apache.hive.jdbc.HiveDriver**

Connecting to jdbc:hive2://localhost:10000

Connected to: Apache Hive (version 1.1.0-cdh5.5.0)

Driver: Hive JDBC (version 1.1.0-cdh5.5.0)

4.3 创建外部表

create external table http_cdr(

key string,

ywlx string,

chuanshuxieyi string,

zhuxieyleixing string,

chanpinbianma string,

yingyongxuhao string,
yewurizhileixing string,
yingyongleixing string,
chanshengshijian string,
jieshushijian string,
laiyuanip string,
xieyiduankou string,
guanliandianhua string,
dslamport string,
status string,
id string,
fasongshujubao bigint,
fasongshujuzijie bigint,
fasongdaikuan int,
yewufasongdaikuan int,
jishoushujubao bigint,
yewujieshouzijieshu bigint,
jieshoudaikuan int,
yewujieshoudaikuan int,
yonghuleixing string,
jieruleixing string,
bangdingleixing string,
sgsnip string,
ggsip string,


```
giip string,  
jierudianmingcheng string,  
shangxingchongchuanliuliang bigint,  
xiaxingchongchuanliuliang bigint,  
shangxingchongchuanshujubao bigint,  
xiaxingchongchuanshujubao bigint,  
shangxingsui pianliuliang bigint,  
xiaxingsui pianliuliang bigint,  
shangxingsui pianbaoshu bigint,  
xiaxingsui pianbaoshu bigint,  
rai string,  
imei string,  
msisdn_location string,  
imsi string,  
sgsn_ctrl_ip string,  
ggsn_ctrl_ip string,  
lianlushangxingbaoshu bigint,  
lianluxiaxingbaoshu bigint,  
lianlushangxingliuliang bigint,  
lianluxianxingliuliang bigint  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '|'   
LOCATION '/hivedata/';
```

4.4 检查表是否创建:

0: jdbc:hive2://localhost:10000> show tables;

```
+-----+--+
```

```
| tab_name |
```

```
+-----+--+
```

```
| http_cdr |
```

```
+-----+--+
```

1 row selected (0.086 seconds)

0: jdbc:hive2://localhost:10000>

0: jdbc:hive2://localhost:10000> describe http_cdr;

0: jdbc:hive2://localhost:10000>

0: jdbc:hive2://localhost:10000> select count(*) from http_cdr;

0: jdbc:hive2://localhost:10000> select * from http_cdr limit 10;

五 HBase 加载和查询数据

在该练习中，我们主要熟悉 HBase 的两个管理界面进行操作，hbase shell 和 GUI 管理界面。

5.1.hbase 当中要特别注意环境变量的设置，当运行 hbase 应用时，环境变量的设置和其它应用有一定差别。

ssh 登录到服务器

su - hdfs

运行 hbase classpath，检查输出

5.2 请确保 zookeeper 服务和 hbase 服务已经启动（先启动 zookeeper，再启动 hbase）。

运行 hbase shell, 进入 shell 管理界面

```
-bash-4.1$ hbase shell
```

```
hbase(main):001:0>
```

```
hbase(main):001:0> help
```

看 hbase 下可执行的命令

5.3 创建表 firsttable, 只有一个 family content

```
hbase(main):008:0> create 'firsttable','firstcolfamily'
```

```
0 row(s) in 0.4420 seconds
```

5.4 查看该表结构

```
hbase(main):009:0> describe 'firsttable'
```

```
Table firsttable is ENABLED
```

```
firsttable
```

```
COLUMN FAMILIES DESCRIPTION
```

```
{NAME => 'firstcolfamily', DATA_BLOCK_ENCODING => 'NONE',  
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1',  
COMPRESSION => 'NONE', MIN_VERSIONS => '0'
```

```
', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>  
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
```

```
1 row(s) in 0.0370 seconds
```

5.5 给 firsttable 增加一个 family

```
hbase(main):011:0> disable 'firsttable'
```

```
hbase(main):013:0> alter 'firsttable',NAME => 'secondfamily'
```

```
hbase(main):014:0> enable 'firsttable'
```

```
hbase(main):015:0> describe 'firsttable'
```

5.6 删除刚才增加的 family

```
hbase(main):015:0> disable 'firsttable'
```

```
hbase(main):016:0> alter 'firsttable','delete'=>'secondfamily'
```

```
hbase(main):017:0> enable 'firsttable'
```

```
hbase(main):018:0> describe 'firsttable'
```

5.7 删除表 firsttable

```
hbase(main):019:0> disable 'firsttable'
```

```
hbase(main):020:0> drop 'firsttable'
```

```
hbase(main):021:0> list
```

5.8 创建表 http_cdr

```
hbase(main):022:0> create 'http_cdr','content'
```

```
hbase(main):023:0> list
```

```
hbase(main):024:0> describe 'http_cdr'
```

5.9 hbase 需要加载的数据为我们前面用的 hive 的数据，如果您没有 copy，请 upload

```
Cd /home/hdfs
```

```
Hdfs dfs -rm -r -rf /mydata/*
```

```
Hdfs dfs -put hivedata /mydata/
```

5.10 启动加载程序

批量加载分为两个阶段，第一阶段是采用 **mapreduce** 进行数据准备，第二阶段是将准备好的数据加载到 **hbase** 数据库

5.10.1 利用 mapreduce 准备数据

```

-bash-4.1$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.separator='|' -
Dimporttsv.columns=HBASE_ROW_KEY,content:ywlx,content:chuansh
uxieyi,content:zhuxieyileixing,content:chanpinbianma,content:
yingyongxuhao,content:yewurizhileixing,content:yingyongleixin
g,content:chanshengshijian,content:jieshushijian,content:laiy
uanip,content:xieyiduankou,content:guanliandianhua,content:ds
lamport,content:status,content:id,content:fasongshujubao,cont
ent:fasongshujuzijie,content:fasongdaikuan,content:yewufasong
daikuan,content:jishoushujubao,content:yewujieshouzijiesshu,co
ntent:jieshoudaikuan,content:yewujieshoudaikuan,content:yongh
uleixing,content:jieruleixing,content:bangdingleixing,content
:sgsnip,content:ggsip,content:giip,content:jierudianmingcheng
,content:shangxingchongchuanliuliang,content:xiaxingchongchua
nliuliang,content:shangxingchongchuanshujubao,content:xiaxing
chongchuanshujubao,content:shangxingsuipianliuliang,content:x
iaxingsuipianliuliang,content:shangxingsuipianbaoshu,content:
xiaxingsuipianbaoshu,content:rai,content:imei,content:msisd_n
location,content:imsi,content:sgsn_ctrl_ip,content:ggsn_ctrl_
ip,content:lianlushangxingbaoshu,content:lianluxiaxingbaoshu,
content:lianlushangxingliuliang,content:lianluxianxingliulian
g -Dimporttsv.bulk.output=/output/httpcdr http_cdr
hdfs://localhost:8020/mydata

hdfs dfs -ls /output/raw/attr

```

可以看到目录下只有一个文件，这是因为目标表默认只有一个 regionserver，所以系统只启动了一个 reducer，当数据量比较大时，首先要对表 split，然后再准备数据，这样可以大大提高加载速度。

5.10.2 将准备好的数据挂载到 hbase

```
export HADOOP_CLASSPATH=`hbase classpath`
```

hdfs dfs -chmod -R 777 /output/httpcdr (这一步很重要, 很多错误都是由于权限问题引起, 在本实验中我们采取了简化处理的方式, 实际操作中请按照需求对给定用户授权即可)。

```
hadoop jar /usr/share/cmf/lib/cdh5/hbase-server-1.0.0-cdh5.5.0.jar completebulkload /output/httpcdr http_cdr
```

5.11 检查加载数据是否成功

```
-bash-4.1$ hbase shell
```

```
hbase(main):001:0> scan 'http_cdr'
```

然后可以休息一会儿了, 或者 ctrl+c 终止

5.12 利用 hbase shell 查询数据

```
hbase(main):001:0> get 'http_cdr','134733819332011/12/21  
16:59:12'
```

```
scan 'http_cdr',{STARTROW=> '134732012792011/12/21  
00:00:00',ENDROW =>'134732012792011/12/21 19:00:00'}
```

更多命令如 get, put, filter 等等请自己找本手册练习。

5.13 用浏览器打开 http://<HMaster_ip>:60010, 查看 HMaster 上的信息.

Region Servers

ServerName	Start time	Requests Per Second	Num. Regions
quickstart.cloudera,60020,1457527780297	Wed Mar 09 20:49:40 CST 2016	0	5
Total: 1		0	5

Backup Masters

ServerName	Port	Start Time
Total: 0		

Tables

Namespace	Table Name	Online Regions	Description
default	gsm	1	'gsm', (NAME => 'attr')
default	http_cdr	1	'http_cdr', (NAME => 'content')
default	test	1	'test', (NAME => 'content')

5.14 用浏览器打开 http://<regionserver_ip>:60030 查看 RegionServer 的信息和 regionserver 管理的 region 信息

Tasks

No tasks currently running on this node.

Block Cache

Attribute	Value	Description
Implementation	LruBlockCache	Block cache implementing class

See [block cache](#) in the HBase Reference Guide for help.

Regions

Region Name	Start Key	End Key	ReplicaID
hbase:meta,1.1588230740			0
http_cdr,1457530659301.d4c6deb2260a3e8b983f43a1f151bd6d			0
gsm,1457529214054.1a227b21569ab0abf7e13d69dd9cf70e			0

5.15 打开 http://<hdfs_ip>:50070 查看 hbase 底层存储的相关信息

5.16 如果大家有时间，可以用浏览器登录到 <http://<HMaster ip>:60010>，点击 http_cdr 表链接，对 http_cdr 表 split 一下，然后重新准备数据，看看这次能启动多少个 reducer，临时目录下准备的文件有多少。

六 Spark 数据加载和查询

6.1. 考虑到系统内存，请登录到 <http://<ip>:7180> 依次停止 hbase, zookeeper, 并确保 hive/yarn 和 hdfs 处于运行状态。

6.2 ssh 登录到服务器，su - hdfs，用 spark 登录运行 pyspark

Pyspark

系统会出现很多 info 和 warning 信息，忽略，出现三个>代表成功。

```
_____  
 /__/_  ____  __/_/  
 _\V _V _\' __/ \'/  
 /__/_ .__^_/_/_/_/_\ version 1.5.0-cdh5.5.0  
  /_/  
  _/
```

Using Python version 2.6.6 (r266:84292, Feb 22 2013 00:00:18)

SparkContext available as sc, HiveContext available as sqlContext.

```
>>>
```

spark 会默认创建一个上下文对象 sc，请确认该对象存在

```
>>> sc
```

```
<pyspark.context.SparkContext object at 0x11c3110>
```

```
>>>
```


运行 `exit()` 或者 `ctrl+d` 退出，注意 `exit` 后面的括号

6.3 我们也可以采用 `scala` 客户端登录

Spark-shell

检查出现的信息是否有错误，如果有，请 `debug` 原因，`scala` 客户端同样会默认建立一个上下文 `sc`，请确认

```
scala> sc
```

```
res0: org.apache.spark.SparkContext =  
org.apache.spark.SparkContext@51105d87
```

输入 `sc.`，然后按 `tab` 键，会显示 `sc` 中所有的变量

```
scala> sc.
```

<code>accumulable</code>	<code>accumulableCollection</code>	<code>accumulator</code>
<code>addFile</code>	<code>addJar</code>	
<code>addSparkListener</code>	<code>appName</code>	
<code>applicationAttemptId</code>	<code>applicationId</code>	<code>asInstanceOf</code>
<code>binaryFiles</code>	<code>binaryRecords</code>	<code>broadcast</code>
<code>cancelAllJobs</code>	<code>cancelJobGroup</code>	
<code>clearCallSite</code>	<code>clearFiles</code>	<code>clearJars</code>
<code>clearJobGroup</code>	<code>defaultMinPartitions</code>	
<code>defaultMinSplits</code>	<code>defaultParallelism</code>	<code>emptyRDD</code>
<code>externalBlockStoreFolderName</code>	<code>files</code>	
<code>getAllPools</code>	<code>getCheckpointDir</code>	<code>getConf</code>
<code>getExecutorMemoryStatus</code>	<code>getExecutorStorageStatus</code>	
<code>getLocalProperty</code>	<code>getPersistentRDDs</code>	
<code>getPoolForName</code>	<code>getRDDStorageInfo</code>	
<code>getSchedulingMode</code>		
<code>hadoopConfiguration</code>	<code>hadoopFile</code>	<code>hadoopRDD</code>
<code>initLocalProperties</code>	<code>isInstanceOf</code>	
<code>isLocal</code>	<code>jars</code>	<code>killExecutor</code>
<code>killExecutors</code>	<code>makeRDD</code>	

master	metricsSystem	newAPIHadoopFile
newAPIHadoopRDD	objectFile	
parallelize	range	requestExecutors
runApproximateJob	runJob	
sequenceFile	setCallSite	setCheckpointDir
setJobDescription	setJobGroup	
setLocalProperty	setLogLevel	sparkUser
startTime	statusTracker	
stop	submitJob	tachyonFolderName
textFile	toString	
union	version	wholeTextFiles

exit 或者 ctrl+d 退出。

6.4 我们加载一个简单的文件到 **spark** 中，并查看

启动 **pyspark**

Pyspark

```
>>> testdata=sc.textFile("file:/home/hdfs/hivedata");
```

```
>>> testdata.take(10);
```

用 **collect()**函数可以查看所有数据，但数据量比较大，输出时间比较长。

```
>>> testdata.collect();
```

6.5 我们用一组 **weblog** 来进行下面的练习

6.5.1 用 **ssh** 登录到服务器

```
-bash-4.1$ su - hdfs
```

```
-bash-4.1$ cd /home/hdfs
```

```
-bash-4.1$ tar xvf web.tar
```

```
-bash-4.1$ cd weblogs
```

```
-bash-4.1$ tail -10 2014-02-09.log
```

至此文件准备完成。

6.5.2 启动 pyspark

```
>>> logfile=sc.textFile("file:/home/hdfs/weblogs/2014-03-04.log");
```

6.5.3 创建一个 RDD，只包含访问 jpg 图片，然后查看前 10 行

```
>>> jpglogs=logfile.filter(lambda x:'jpg' in x);
```

```
>>> jpglogs.take(10);
```

6.5.4 用 map 函数查询每一行的长度，并取前 10 行直接输出

```
>>> logfile.map(lambda x:len(x)).take(10);
```

6.5.5 用 map 函数将每一行的每个单词生成一个数据

```
>>> logfile.map(lambda x:x.split()).take(2);
```

检查输出结果并观察 spark 输出的信息。

6.5.6 用 map 函数实现只取每一行的 ip 地址

```
>>> ip=logfile.map(lambda x:x.split()[0]);
```

```
>>> ip.take(10);
```

```
for x in ip.take(10):print x;
```

6.6 我们用 spark 对 weblogs 目录下的所有数据生成 key/value 的 pairs RDD

首先用 map 生成

```
(userid1,1)
```

```
(userid2,1)
```

```
(userid3,1)
```

```
...
```

```
(useridn,1)
```

然后用 reduce 汇总成

(userid1,15)

(userid2,6)

(userid3,2)

...

(useridn,8)

最后用 `sortByKey` 函数输出结果。

6.6.1 生成一个 RDD,该 RDD 用 `userid` 做 `key`, `value` 是所有实用的 `ip` 地址。

```
>>> logs=sc.textFile("file:/home/hdfs/weblogs/*");
```

```
>>> userregs = logs.map(lambda line: line.split()).map(lambda words:
(words[2],1)).reduceByKey(lambda count1,count2: count1 + count2);
```

```
>>> userregs.count();
```

```
>>> userregs.take(10);
```

6.7 spark 运行在 hdfs

首先将 `weblogs` 目录上传到 `hdfs`,然后将图片访问的记录过滤出来,并存储到 `hdfs` 的 `/output` 下。

Su - hdfs

```
hdfs dfs -put weblogs /weblogs
```

```
hdfs dfs -ls /weblogs
```

```
hdfs dfs -cat /weblogs/2014-03-08.log | tail -n 50
```

pyspark

```
>>> logs=sc.textFile("hdfs://localhost:8020/weblogs/*");
```

```
>>> logs.count();
```

```
>>> logs.filter(lambda x:".jpg" in
x).saveAsTextFile("hdfs://localhost:8020/output/jpgfile");
```

回到 `bash` 下检查文件存储是否成功

```
-bash-4.1$ hdfs dfs -cat /output/jpgfile/part-00181 |tail -n 50
```

（如果重新执行需要手动删除 hdfs /output/jpgfile 目录）

6.8 用 sparkSQL 做 ETL 作业

首先将 spark 数据上传到 hdfs

```
hdfs dfs -mkdir /sparkdata
```

```
hdfs dfs -put /home/hdfs/sparkdata /sparkdata
```

```
spark-shell
```

```
scala>val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

```
scala>import sqlContext.implicits._
```

```
scala>case class
```

```
cdrl(guanliandianhua:String,msisd_n_location:String,sgsnip:String,jierudianmingcheng:String,rai:
String,imei:String,chanshengshijian:String,jieshushijian:String,yonghuleixing:
String,jieruleixing:String,bangdingleixing:String,fasongshujubao:Int,jishoushujubao:
Int,shangxingchongchuanshujubao:Int,xiaxingchongchuanshujubao:Int,shangxingsuipianbaoshu:
Int,xiaxingsuipianbaoshu:Int,lianlushangxingbaoshu:Int,lianluxiaxingbaoshu:Int)
```

```
scala>val httpcdr2=sc.textFile("hdfs://localhost:8020/sparkdata/*").map(_.split('|')).map(p
=> new
cdrl(p(0),p(1),p(2),p(3),p(4),p(5),p(6),p(7),p(8),p(9),p(10),p(11).trim.toInt,p(12).trim.toInt,p
(13).trim.toInt,p(14).trim.toInt,p(15).trim.toInt,p(16).trim.toInt,p(17).trim.toInt,p(18).trim.t
oInt)).toDF();
```

```
scala>httpcdr2.registerTempTable("httpcdr2")
```

```
scala> val rs=sqlContext.sql("select
msisd_n_location,sum(fasongshujubao),sum(jishoushujubao),sum(shangxingchongchuanshujubao),sum(xia
xingchongchuanshujubao),sum(shangxingsuipianbaoshu),sum(xiaxingsuipianbaoshu),sum(lianlushangxin
gbaoshu),sum(lianluxiaxingbaoshu) from httpcdr2 group by msisd_n_location")
```

```
scala> rs.map(t=> t(0)+",("+t(1)+","+t(2) + ","+t(3) + "," +
t(4)+","+t(5)+","+t(6)+","+t(7)+","+t(8)+")").collect().foreach(println)
```

检查输出结果，和 hive 执行的 ETL 类似。

因时间原因 spark cluster 的运行和 spark mlb 请大家课后自己搭建环境联系。

