Sqoop 及 Hbase 体验报告

李易非 熊苗 林新迪

一、Mysql 安装及数据导入

最终Hbase需要5个节点，我们在网易云上建立5个节点，并在 `host0` `slave1` 上配置 MySQL 环境，将Imdb和烂番茄数据分别存在两个节点上。

1 环境搭建

1. 输入以下命令更新系统安装源：

```
sudo apt-get update
```

如果不更新，下面使用sudo apt-get命令在线安装软件时可能会出现错误：Unable to locate package. 因为软件源可能被修改。

2.输入下面命令依次在线安装mysql服务器、客户端和运行库：

```
sudo apt-get install mysql-server
sudo apt-get install mysql-client
sudo apt-get install libmysqlclient-dev
```

3.登录进入MySQL

```
mysql -h localhost -u root -p
# 退出用exit-
# h数据库主机
# -u用户
# -p密码
# -P端口号（大写P）
```

2 导入数据

1.创建数据库

```
create database DB_IM;
use DB_IM;
```

2.创建表格

对于imdb数据：

```
 create table TB_IM (Id varchar(100),
 Name varchar(100),
 YearRange varchar(100),
```

```
  ReleaseDate varchar(30),
  Director varchar(100),
  Creator varchar(100),
  Cast varchar(100),
  Duration varchar(20),
  RatingValue varchar(10),
  ContentRating varchar(10),
  Genre varchar(100),
  Url varchar(300),
  Description varchar(3000),
  primary key(Id));
```

对于烂番茄数据

```
  create table TB_badtomato (Id varchar(200),
  Name varchar(300),
  YearRange varchar(100),
  ReleaseDate varchar(100),
  Director varchar(500),
  Creator varchar(300),
  Actors varchar(200),
  Cast varchar(5000),
  Language varchar(100),
  Country varchar(200),
  Duration varchar(20),
  RatingValue varchar(50),
  RatingCount varchar(20),
  ReviewCount varchar(50),
  Genre varchar(300),
  FilmingLocations varchar(200),
  Description varchar(10000),
  primary key(Id));
```

3.导入数据

```
  load data local infile '~/imdb.csv'
  into table TB_IM
  fields terminated by ','
  optionally enclosed by '"'
  escaped by '"'
  lines terminated by '\n';
```

由于我们存储信息使用的是变长字符串，有可能在读入的时候字符串长度超过我们的类型设置，会导致data truncated。此时我们需要使用 `show warnings` 命令查看导入信息的warning，判断是否发生data truncated。

3 设置host

一般来说，Mysql默认为只允许 localhost访问，如果我们想要使用Sqoop 就必须允许通过ip地址从其他服务器节点访问当前Mysql系统。

想让**局域网**中的所有机器都能连接**MySQL数据库**，首先要给MySQL开启远程连接的功能。想要开启这个权限需要两方面的设置，一方面是通过user table修改host这一栏。

1. 在localhost的服务器节点登入mysql后
2. 更改 "mysql" 数据库里的 "user" 表里的 "host" 项，从"localhost"改称"%" 或添加一个用户为“%” 。

```
grant all privileges on *.* to root@"%" identified by 'root' with grant option;
flush privileges;
```

上面两行代码的意思是给从任意ip地址连接的用户名为root，密码为root的用户赋予所有的权限。其中的"%"为任意的ip地址，如果想设为特定的值也可以设定为特定的值（以通配符%的内容增加主机/IP地址，也可以直接增加IP地址）。

同时我们还需要进入 `/etc/mysql/mysql.conf.d/mysqld.cnf` 文件修改，将 `bind address` 注释掉。

通过上述设置后我们可通过如下语句来测试:

```
mysql -hlocalhost -uroot -proot # 保证本地可以连进去
mysql -hhost0 -uroot -proot # 保证通过 host0 可以连进去
mysql -hslave1 -uroot -proot # 保证 host0 可以访问 slave1 的 mysql 服务器
```

二、集群环境搭建

环境一览：

- hadoop 2.7.6
- Hbase: 1.2.11
- Sqoop: 1.4.7

Hbase 集群安装

HBase是一种构建在HDFS之上的分布式、面向列的存储系统，它适用于需要实时读写、随机访问超大规模数据集的情形。不同于一般的关系数据库是基于行的存储, Hbase是基于列的存储，数据本身就是索引，这样的存储方式非常适合于非结构化数据存储。
HBase是介于Map Entry(key & value)和DB Row之间的一种数据存储方式。它设计目标是支持稀疏表。稀疏表通常会有很多列，但每一行有值的列较少。按照传统的设计，我们可以有两种设计方法。

1. 按行存储，把所有列的数据合在一起放在一个文件中。当我们想要访问少数几个列的数据时，需要遍历每一行，读取整个表的数据，较为低效。
2. 按列存储，把每一列的数据单独分开存在一个文件中。当我们想要访问少数几个列的数据时，只需要读取对应的文件，不用读取整个表的数据，读取效率很高。然而，由于稀疏表通常会有很多列，这会导致文件数量特别多，会影响文件系统的效率。

而Column Family的提出就是为了在上面两种方案中做一个折中。HBase中将一个Column Family中的列存在一起，而不同Column Family的数据则分开，同时可以把经常一起访问的比较类似的列放在同一个Column Family中，这样就可以在访问少数几个列时，读取尽量少的数据。

简单来说，HBase中创建的表由RowKey，Column Family 组成，而 Column Family 又由 Column qualifier 数组组成。表的 Column Family 可以根据需求动态增加，针对某个特定数据，我们只需要把它存储在 Hbase的某个 column family 的 byte[i] 字符串中即可，无需指定它的具体类型是char还是varchar等。

但这样的设计也有一些trade-off，Hbase并不支持多行事务操作，也仅仅支持对于Key的线性索引。

针对我们的两个表格，我们发现两个表的 column 高度相关且有许多的overlap。

```
IMDB:
"Id","Name","YearRange","ReleaseDate","Director","Creator","Cast","Duration","RatingValue","ContentRating","Genre","Url","Description"

烂番茄:
"Id","Name","Year","Release Date","Director","Creator","Actors","Cast","Language","Country","Duration","RatingValue","RatingCount","ReviewCount","Genre","Filming Locations","Description"

```

针对这样的情况，我们将两个表相同的 column qualifier 修改为相同名字，并将所有属性存放在同一个 Column Family `movieFeature` 里面。

在这一部分中，我们基本参照官方文档 **HBase QuickStart** 完成了集群的设置，需要特别注意的是 HBase 与 Hadoop, Sqoop 的版本兼容问题，尽管在官方文档中声明 HBase-2.0.5 与 Hadoop 2.7.6 相容，但在 Sqoop 导入过程中会出现问题，我们会在错误总结中详述，最终我们采取的 HBase 版本为 1.2.11

	HBase-1.2.x, HBase-1.3.x	HBase-1.4.x	HBase-2.0.x	HBase-2.1.x
Hadoop-2.4.x	✔	✘	✘	✘
Hadoop-2.5.x	✔	✘	✘	✘
Hadoop-2.6.0	✘	✘	✘	✘
Hadoop-2.6.1+	✔	✘	✔	✘
Hadoop-2.7.0	✘	✘	✘	✘
Hadoop-2.7.1+	✔	✔	✔	✔
Hadoop-2.8.[0-1]	✘	✘	✘	✘
Hadoop-2.8.2	!	!	!	!
Hadoop-2.8.3+	!	!	✔	✔
Hadoop-2.9.0	✘	✘	✘	✘
Hadoop-2.9.1+	!	!	!	!
Hadoop-3.0.[0-2]	✘	✘	✘	✘
Hadoop-3.0.3+	✘	✘	✔	✔
Hadoop-3.1.0	✘	✘	✘	✘
Hadoop-3.1.1+	✘	✘	✔	✔

HBase 的安装过程可以总结如下：

1. 下载文件并解压，移动到相关位置，添加相关路径

```
cd
wget http://mirrors.tuna.tsinghua.edu.cn/apache/hbase/1.2.11/hbase-1.2.11-bin.tar.gz
tar xzvf hbase-1.2.11-bin.tar.gz
mv hbase-1.2.11 /usr/local/hbase
export HBASE_HOME=/usr/local/hbase  # 可以直接加入 ~/.bashrc
export PATH=$PATH:$HBASE_HOME/bin # 可以直接加入 ~/.bashrc
```

2. 修改 `$HBASE_HOME/conf` 下的相关配置文件，在本次作业中，我们在5个节点 host0, slave[1-4] 上搭建集群环境，将 host0 作为 HMaster, 其余四个 slave 服务器作为 RegionServer, 考虑到任务相对简单暂时没有设置 Backup server

    1. 修改 `hbase-env.sh` , 加入 `export JAVA_HOME=/usr/lib/jvm/default-java`
    2. 修改 `hbase-site.xml`

```
<configuration>
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://host0:9000/hbase</value>
    </property>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>host0,slave1,slave2,slave3,slave4</value>
    </property>
    <property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>/usr/local/zookeeper</value>
    </property>
</configuration>
```

    其中， `hbase.rootdir` 需要设置 HDFS 文件系统对应的端口（在我们的设置中：9000）下的文件目录，需要注意的是，这个目录不需要我们自己建立，若自己建立可能会产生诸如文件迁移的副作用。

    `hbase.cluster.distributed` 设为 `true` 之后表明 HBase 搭建在分布式集群上

    `hbase.zookeeper.quorum` `hbase.zookeeper.property.dataDir` 为 zookeeper 相关设置，这里采取了简单粗暴的设置，对于效率要求更高的系统可以参考 **zookeeper 设置** 进行精细设置

    3. 修改 `regionservers` , 改为 slave1, slave2, slave3, slave4（换行符分离） 即可；
3. 将修改后的 `$HBASE_HOME` 发送给所有 slave 节点

```
scp /usr/local/hbase root@slave1:/usr/local/
scp /usr/local/hbase root@slave2:/usr/local/
scp /usr/local/hbase root@slave3:/usr/local/
scp /usr/local/hbase root@slave4:/usr/local/
```

4. 启动 Hbase shell，并进行简单命令测试

```
[root@host0 ~]$ start-hbase.sh
host0: starting zookeeper, logging to /usr/local/hbase/logs/hbase-root-zookeeper-host0.out
slave1: starting zookeeper, logging to /usr/local/hbase/logs/hbase-root-zookeeper-slave1.out
slave4: starting zookeeper, logging to /usr/local/hbase/bin/../logs/hbase-root-zookeeper-slave4.out
slave2: starting zookeeper, logging to /usr/local/hbase/bin/../logs/hbase-root-zookeeper-slave2.out
slave3: starting zookeeper, logging to /usr/local/hbase/bin/../logs/hbase-root-zookeeper-slave3.out
starting master, logging to /usr/local/hbase/logs/hbase-root-master-host0.out
slave2: starting regionserver, logging to /usr/local/hbase/bin/../logs/hbase-root-regionserver-slave2.out
slave4: starting regionserver, logging to /usr/local/hbase/bin/../logs/hbase-root-regionserver-slave4.out
slave3: starting regionserver, logging to /usr/local/hbase/bin/../logs/hbase-root-regionserver-slave3.out
slave1: starting regionserver, logging to /usr/local/hbase/logs/hbase-root-regionserver-slave1.out
```

此时，在 host0 (master) 上 jps, 我们可以看到

```
[root@host0 ~]$ jps
31632 HQuorumPeer
31728 HMaster
31137 ResourceManager
30771 NameNode
32020 Jps
30984 SecondaryNameNode
15370 JobHistoryServer
```

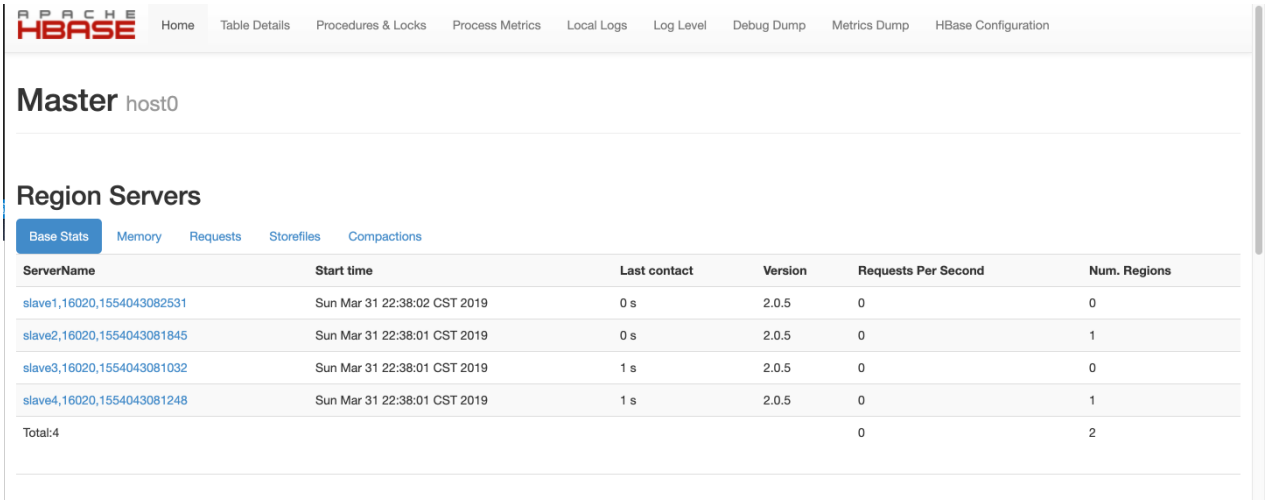
其中，`HMaster`，`HQuorumPeer` 为 HBase 与 zookeeper 相关进程

在 slave 上 jps, 我们可以看到

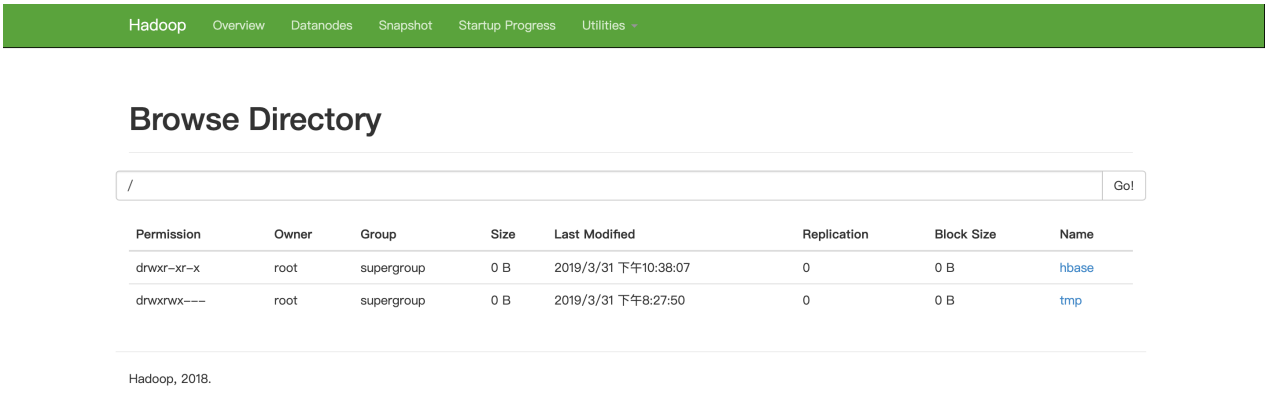
```
root@slave1:~# jps
9601 Jps
9046 NodeManager
9223 HQuorumPeer
8922 DataNode
9358 HRegionServer
```

其中，`HRegionServer`，`HQuorumPeer` 为 HBase 与 zookeeper 相关进程

同样，我们在 `host0:16010` 我们可以看到 HBase 运行状态



在 `host0:50070` (HDFS) 页面，我们也可以看到我们刚刚指定的文件夹 `hbase.root.dir`



接着，进行几条简单命令的测试，我们就能确定 HBase 的环境搭建基本无误了，更多的 shell 命令详见 **hbase shell 命令一览**

```
hbase(main):001:0> create 'hbase_test', 'cf'
0 row(s) in 1.9070 seconds

=> Hbase::Table - hbase_test
hbase(main):002:0> list 'hbase_test'
TABLE
hbase_test
1 row(s) in 0.0270 seconds

=> ["hbase_test"]
hbase(main):003:0> put 'hbase_test', 'row1', 'cf:a', 1
0 row(s) in 0.1950 seconds

hbase(main):004:0> scan 'hbase_test'
ROW                           COLUMN+CELL
 row1                         column=cf:a, timestamp=1554302412769, value=1
1 row(s) in 0.0610 seconds
```

Sqoop 安装篇

sqoop 的官方文档对于安装部分的教程很少，所以在本次作业中我们参考了 **sqoop 安装教程** 进行 sqoop 的安装配置

1. 下载，解压，修改相关路径

```
tar zxvf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
mv sqoop-1.4.7.bin__hadoop-2.6.0 /usr/local/sqoop
export SQOOP_HOME=/usr/local/sqoop # 加入 ~/.bashrc 中
export PATH=$PATH:/usr/local/sqoop/bin
```

2. 修改 `$SQOOP_HOME/conf` 中的 `sqoop-env.sh` 文件，加入如下信息

```
export HADOOP_COMMON_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

3. 配置 java 的 mysql 连接器 mysql-connector-java

```
tar zxvf mysql-connector-java-5.1.32.tar.gz
cd mysql-connector-java-5.1.32
mv mysql-connector-java-5.1.32-bin.jar $SQOOP_HOME/lib/
```

4. 验证 sqoop 版本

```
[root@host0 ~]$ sqoop-version
Warning: /usr/local/sqoop/../hcatalog does not exist! HCatalog jobs will fail.
Please set $HCAT_HOME to the root of your HCatalog installation.
Warning: /usr/local/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
19/04/03 22:53:02 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
Sqoop 1.4.7
git commit id 2328971411f57f0cb683dfb79d19d4d19d185dd8
Compiled by maugli on Thu Dec 21 15:59:58 STD 2017
```

通过 sqoop 导入 mysql 数据到 HBase

在这一部分中，我们遇到了诸多问题，问题描述详见错误总结，在这一部分中我们将着重讲解我们摸索的一套疑难排查流程；

首先，mysql 默认是不允许除了 localhost 之外的服务器登陆的，所以我们首先需要如 *Mysql 安装* 部分中叙述的步骤解决权限问题，接着尝试一下几个命令

```
mysql -hlocalhost -uroot -proot # 保证本地可以连进去
mysql -hhost0 -uroot -proot # 保证通过 host0 可以连进去
mysql -hslave1 -uroot -proot # 保证 host0 可以访问 slave1 的 mysql 服务器
```

若可以，则出现问题不会是 mysql 连接的问题，接着我们可以尝试在 host0 (sqoop 安装服务器) 进行基础测试

```
sqoop list-databases --connect jdbc:mysql://localhost/ --username root -P
sqoop list-databases --connect jdbc:mysql://host0/ --username root -P
sqoop list-databases --connect jdbc:mysql://slave1/ --username root -P
```

若上述语句出错，请仔细检查 mysql 相关权限设置是否正确；

我们确保了 sqoop 可以连接到 mysql 并展示所有 databases，下一步确保可以正确连接特定的 database

```
sqoop list-tables --connect jdbc:mysql://localhost/DB_IM --username root -P
sqoop list-tables --connect jdbc:mysql://host0/DB_IM --username root -P
sqoop list-tables --connect jdbc:mysql://slave1/DB_Bad_tomatoes --username root -P
```

若上述语句出错，请检查数据库名称是否指定正确以及 mysql 权限是否设置正确；

确保了上述连接正常后，我们可以开始尝试 import 操作，在这一步操作中，我们预先在 HBase 中建立了表 `movies`，columnFamily 为 `movieFeature`

```
sqoop import \
--connect jdbc:mysql://host0/DB_IM \
--username root -P \
--table TB_IM \
--hbase-table movies --hbase-row-key Id --column-family movieFeature \
-m 1

sqoop import \
--connect jdbc:mysql://slave1/Bad_tomatoes \
--username root -P \
--table TB_badtomato \
--hbase-table movies --hbase-row-key Id --column-family movieFeature \
-m 1
```

上述命令中 `--hbase-table` 指定 HBase 中待导入的表，`--hbase-row-key` 指定每一行的 key，需要特别注意的是 `-m` 参数，在一般的导入过程中，我们需要设置 `-split-by` 参数，若不设定，则这个参数缺省为 `--hbase-row-key` 的值，他的作用在于根据 `-split-by` 指定的参数对数据进行 partition，从而利用多个服务器并行导入，所以这个值最好是分布较为均匀的数字类型的 column，但在我们的例子中所有的 column 均是 *VARCHAR* 类型，我们可以人为为所有数据添加一个关联的数字作为 `-split-by` 的 column，我们也可以直接指定 `-m 1`，仅让一个节点进行 mapreduce 任务，考虑到本次作业数据量较小，并行的优势也体现不出来，我们采取了后一种简单的解决方案。**官方对于 split 的解释在这里**

导入成功后，我们可以在 HBase shell 中查看导入是否成功

```
hbase(main):002:0> describe 'movies'
Table movies is ENABLED
movies
COLUMN FAMILIES DESCRIPTION
{NAME => 'movieFeature', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY =>
'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_E
NCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPL
ICATION_SCOPE => '0'}
1 row(s) in 0.1760 seconds
```

```
hbase(main):001:0> count 'movies'
Current count: 1000, row: anthar-linvincibiledevil-of-the-desert-against-the-son-of-hercules
Current count: 2000, row: invisible_man_returns
Current count: 3000, row: tall_texan
Current count: 4000, row: tt0042771
Current count: 5000, row: tt0080716
Current count: 6000, row: tt0104714
Current count: 7000, row: tt0175880
Current count: 8000, row: tt0435761
Current count: 9000, row: tt1389127
Current count: 10000, row: tt2442466
10964 row(s) in 1.7410 seconds

=> 10964
```

可以看到，我们成功导入了 10964 条来自两个数据库的数据，为了确定 schema matching 是否成功，我们还做了如下的测试

```
# Duration 为一个公共特征，共有 10964 数据
hbase(main):004:0> scan 'movies', {COLUMNS => 'movieFeature:Duration'}
...
 zapped_2014    column=movieFeature:Duration, timestamp=1554259069477, value=1 hr. 42 min.
 zapped_again   column=movieFeature:Duration, timestamp=1554259069477, value=
 zone_of_the_dead   column=movieFeature:Duration, timestamp=1554259069477, value=
 zwarte-zwanen-black-swans   column=movieFeature:Duration, timestamp=1554259069477, value=
10964 row(s) in 3.4690 seconds

# Name 为一个公共特征，共有 10964 条数据
hbase(main):004:0> scan 'movies', {COLUMNS => 'movieFeature:Name'}
...
zapped_again    column=movieFeature:Name, timestamp=1554259069477, value=Zapped Again
 zone_of_the_dead  column=movieFeature:Name, timestamp=1554259069477, value=Zone of the Dead
 zwarte-zwanen-black-swans   column=movieFeature:Name, timestamp=1554259069477, value=Zwarte zwanen (Black Swans)
10964 row(s) in 3.1710 seconds

# Actors 为非公共特征，仅在烂番茄中有，共有 7390 条数据
hbase(main):004:0> scan 'movies', {COLUMNS => 'movieFeature:Actors'}
...
 tt5020372  column=movieFeature:Actors, timestamp=1554262715303, value=Tom Hopper,Amy Huberman,Nick Dunning
 tt5022680  column=movieFeature:Actors, timestamp=1554262715303, value=Landon Ackerman,April Adamson,Ali Adatia
 tt5039578  column=movieFeature:Actors, timestamp=1554262715303, value=Lea Thompson,Zoey Deutch,Avan Jogia
7390 row(s) in 2.7520 seconds
```

```
25
26  # ContentRating 为非公共特征，仅在 IMDB 中有，共有 3574 条数据
27  hbase(main):004:0> scan 'movies', {COLUMNS => 'movieFeature:ContentRating'}
28  ...
29  zapped_again  column=movieFeature:ContentRating, timestamp=1554259069477, value=R
30  zone_of_the_dead  column=movieFeature:ContentRating, timestamp=1554259069477,
    value=R
31  zwarte-zwanen-black-swans  column=movieFeature:ContentRating,
    timestamp=1554259069477, value=Unrated
32  3574 row(s) in 1.3650 seconds
```

在上述测试中，我们通过提取不同的列，确定我们的 schema matching 是成功的。

五、 错误记录

1. MySQL导入数据时使用 local infile

我们在导入数据时一开始使用

```
1  load data infile '~/imdb.csv'
2  into table TB_IM
3  fields terminated by ','
4  optionally enclosed by '"'
5  escaped by '"'
6  lines terminated by '\n';
```

没有local infile这个lable的时候，数据读入一直出错，报错如下

```
1  The MySQL server is running with the --secure-file-priv option so it cannot execute
   this statement
```

这个问题的原因是 MySQL 使用了 secure-file-priv 来保证读入数据的目录是安全的可信任的。如果没有预先设置的话，将会限制读入数据的目录在预先设置的特定目录。要解决这个问题，我们可以关闭 secure-file-priv 设置，但是更简单的方法是加上 Local 这个关键词，文件就是被 client 读取并发送到 server 上，巧妙避开了 secure-file-priv 这个设置。

2. 在 sqoop import 过程中爆出 Connection refused 的错误

通过环境配置中的 *通过 sqoop 导入 mysql 数据到 HBase* 部分的操作，我们逐步排查问题，解决了问题

3. sqoop import 过程中爆出 `java.lang.NoSuchMethodError: org.apache.hadoop.hbase.client.HBaseAdmin.<init>(Lorg/apache/hadoop/conf/Configuration;)V` 的错误

经过资料收集，是 HBase 与 Hadoop 一些版本不兼容性导致的，尽管官方声明 HBase 2.0.5 与 Hadoop 2.7.6 是兼容的。替换为 HBase 1.2.11 版本后 import 正常；

4. sqoop import 过程中爆出 `Generating splits for a textual index column allowed only in case of "-Dorg.apache.sqoop.splitter.allow_text_splitter=true"`

```
property passed as a parameter
```

这个错误正是我们在 *通过 sqoop 导入 mysql 数据到 HBase* 叙述的 `-split-by` 的作用以及注意事项，加入 `-m 1` 之后解决。

小组分工

- 李易非：HBase 集群搭建，Sqoop 安装，Sqoop 插入数据，报告撰写
- 熊苗：mysql 服务器安装，权限修改，Sqoop 安装，报告撰写
- 林新迪：报告整合撰写，错误查询，Sqoop 插入数据