

实验4

报告较长，建议借助书签查看

实验4-第1题

分别编写MapReduce程序和Spark程序统计双十一最热门的商品和最受年轻人(age<30)关注的商家（“添加购物车+购买+添加收藏夹”前100名）；

MapReduce编程

统计双十一最热门的商品（“添加购物车+购买+添加收藏夹”前100名）

代码：

EX4-Hadoop\src\main\java\topcount\TopItems.java

输出文件：

output/mapreduce-output/output1

实现过程

问题可以转化为Word Count任务，输入是user_log_format1.csv文件，输出是<item_id, count>的key-value对。只需进行2次MapReduce任务，第一次统计商品的重要值，第二次按重要值排序。比较像第一次MapReduce编程的作业。

第1次map阶段：

按行划分Mapper任务，对每一行按符号','分割，如果分割后的第5个字符串（timestamp）等于'1111'，同时第6个字符串（action_type）不等于'0'，则生成<item_id,1>的key-value对。

第1次reduce阶段：

将相同key值得value加总，得到<item_id,count>。

第2次map阶段：

key-value对将上一一次的输出结果对调，即<频数>-<单词>

第2次reduce阶段：

设置reducer的数量为1即可。

运行的命令为：


```
hadoop jar EX4-Hadoop-1.0-SNAPSHOT.jar topcount.TopItems
exp4/data_format1/user_log_format1.csv exp4/output
```

结果存放在了mapreduce_output/output1中

→ ↺ ⚠ 不安全 | node1:8088/cluster

🔍 ☆ 📄 ⚙️ 👤 ⋮

Logged in as: dr.who



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	0	2	0	0 B	16 GB	0 B	0	16	0	2	0	0	0	0

Scheduler Metrics

Scheduler Type		Scheduling Resource Type		Minimum Allocation		Maximum Allocation	
Capacity Scheduler		[MEMORY]		<memory:1024, vCores:1>		<memory:8192, vCores:8>	

Show 20 entries

Search:


ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1607772341611_0002	root	sort	MAPREDUCE	default	Sat Dec 12 19:31:16 +0800 2020	Sat Dec 12 19:31:35 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1607772341611_0001	root	items count	MAPREDUCE	default	Sat Dec 12 19:28:48 +0800 2020	Sat Dec 12 19:31:14 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A

Showing 1 to 2 of 2 entries

First Previous 1 Next Last

安全 | node1:19888/jobhistory/job/job_1607772341611_0001

🔍 ☆ 📄 ⚙️ 👤 ⋮



MapReduce Job job_1607772341611_0001

Job Overview

Job Name:

items count

User Name:

root

Queue:

default

State:

SUCCEEDED

Uberized:

false

Submitted:

Sat Dec 12 19:28:48 CST 2020

Started:

Sat Dec 12 19:28:59 CST 2020

Finished:

Sat Dec 12 19:31:14 CST 2020

Elapsed:

2mins, 15sec

Diagnostics:

Average Map Time

1mins, 3sec

Average Shuffle Time

1mins, 16sec

Average Merge Time

0sec

Average Reduce Time

2sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Dec 12 19:28:53 CST 2020	node2:8042	logs

Task Type	Total	Complete	
Map	15	15	
Reduce	1	1	
Attempt Type	Failed	Killed	Successful
Maps	4	4	15
Reduces	0	0	1

安全 | node1:19888/jobhistory/job/job_1607772341611_0002

🔍 ☆ 🚫 ⚙️ 👤 ⋮



MapReduce Job job_1607772341611_0002

Job Overview

Job Name: sort
User Name: root
Queue: default
State: SUCCEEDED
Uberized: true
Submitted: Sat Dec 12 19:31:16 CST 2020
Started: Sat Dec 12 19:31:30 CST 2020
Finished: Sat Dec 12 19:31:35 CST 2020
Elapsed: 4sec
Diagnostics:
Average Map Time 2sec
Average Shuffle Time 0sec
Average Merge Time 0sec
Average Reduce Time 1sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Dec 12 19:31:23 CST 2020	node1:8042	logs

Task Type	Total		Complete	
Map	1		1	
Reduce	1		1	
Attempt Type	Failed		Killed	Successful
Maps	0	0		1
Reduces	0	0		1

```

[root@node1 exp4]# hadoop jar EX4-Hadoop-1.0-SNAPSHOT.jar topcount.TopItems exp4/data_format1/user_log_format1.csv exp4/output
20/12/12 19:28:45 INFO client.RMPProxy: Connecting to ResourceManager at node1/172.18.174.44:8032
20/12/12 19:28:46 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool inter
r application with ToolRunner to remedy this.
20/12/12 19:28:47 INFO input.FileInputFormat: Total input paths to process : 1
20/12/12 19:28:47 INFO mapreduce.JobSubmitter: number of splits:15
20/12/12 19:28:47 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1607772341611_0001
20/12/12 19:28:48 INFO impl.YarnClientImpl: Submitted application application_1607772341611_0001
20/12/12 19:28:48 INFO mapreduce.Job: The url to track the job: http://node1:8088/proxy/application_1607772341611_0001/
20/12/12 19:28:48 INFO mapreduce.Job: Running job: job_1607772341611_0001
20/12/12 19:29:00 INFO mapreduce.Job: Job job_1607772341611_0001 running in uber mode : false
20/12/12 19:29:00 INFO mapreduce.Job: map 0% reduce 0%
20/12/12 19:29:17 INFO mapreduce.Job: map 1% reduce 0%
20/12/12 19:29:25 INFO mapreduce.Job: map 2% reduce 0%
20/12/12 19:29:29 INFO mapreduce.Job: map 3% reduce 0%
20/12/12 19:29:33 INFO mapreduce.Job: map 4% reduce 0%
20/12/12 19:29:37 INFO mapreduce.Job: map 6% reduce 0%
20/12/12 19:29:40 INFO mapreduce.Job: map 7% reduce 0%
20/12/12 19:29:42 INFO mapreduce.Job: map 8% reduce 0%
20/12/12 19:29:43 INFO mapreduce.Job: map 10% reduce 0%
  
```

```
[root@node1 exp4]# hdfs dfs -cat exp4/output/*
1: 191499, 2494
2: 353560, 2250
3: 1059899, 1917
4: 713695, 1754
5: 655904, 1674
6: 67897, 1572
7: 221663, 1547
8: 1039919, 1511
9: 454937, 1387
10: 81360, 1361
11: 514725, 1356
12: 783997, 1351
13: 823766, 1343
14: 107407, 1319
15: 889095, 1272
16: 936203, 1270
17: 770668, 1257
18: 698879, 1235
19: 349999, 1218
20: 671759, 1167
21: 186456, 1162
```

统计最受年轻人(age<30)关注的商家 (“添加购物车+购买+添加收藏夹”前100名)

代码：

EX4-Hadoop\src\main\java\topcount\TopMerchants.java

输出文件：

output/mapreduce-output/output2

实现过程

由于购买记录和用户信息存放在两张表中，所以这道题的重点是两张表的合并，然后是和上面一样思路的WordCount+排序。

如何利用MapReduce排序呢？第一次map的数据来自于两个表，即user_log_format1和user_info_format1，通过筛选之后各自产生<user_id,seller_id>和<user_id,age_range>的key-value对。第一个问题是如何判断数据是来自哪个文件？我的方法是在map中从context上下文获取文件的路径：

```
String filepath = ((FileSplit)context.getInputSplit()).getPath().toString();
```

然后产生key-value对时先判断数据来自哪个文件，如果来自user_log_format，则产生<user_id,seller_id>，来自user_info_format，则产生<user_id,age_range>，这里还需要对value值进行标记，因为reduce并不知道某个key-value对来自哪个表。

```

if (filepath.indexOf("user_log_format1") != -1) {
    generate <user_id,seller_id+tag:"a">
} else if(filepath.indexOf("user_info_format1") != -1) {
    generate <user_id,age_range+tag:"b">
}

```

在reduce阶段，根据标签判断数据来自哪个表，然后将同一个user_id对应的seller_id和age_range合并即可。值得一提的是，在合并任务的map阶段，就可以对log进行筛选，根据timespan和action_type筛选出在1111当天有效的行为记录，这样map产生的key-value值可以大大减少，在reduce阶段又可以进一步筛选，只对age_range在(1,2,3)中的user_id输出<null,seller_id>即可，将所有无效数据剔除，下一步的MapReduce只需要对seller_id做wordcount，工作量大大减小。

运行代码的命令为：

```

hadoop jar EX4-Hadoop-1.0-SNAPSHOT.jar topcount.TopMerchants \
-Dinput_data=exp4/data_format1/user_log_format1.csv,\
exp4/data_format1/user_info_format1.csv \
-Doutput_dir=exp4/output

```

结果存放在了mapreduce_output/output2中

The screenshot shows the Hadoop YARN web interface. At the top, there's a navigation bar with '不安全' (Insecure) and 'node1:8088/cluster'. Below it is the 'All Applications' header. The main content area is divided into 'Cluster Metrics' and 'Scheduler Metrics'.

Cluster Metrics:

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
3	0	0	3	0	0 B	16 GB	0 B	0	16	0	2	0	0	0	0

Scheduler Metrics:

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Below the scheduler metrics, there's a table showing application details. The table has columns: ID, User, Name, Application Type, Queue, StartTime, FinishTime, State, FinalStatus, Progress, Tracking UI, and Blacklisted Nodes.

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1607830155379_0003	root	sort	MAPREDUCE	default	Sun Dec 13 11:33:19 +0800 2020	Sun Dec 13 11:33:37 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1607830155379_0002	root	word count	MAPREDUCE	default	Sun Dec 13 11:32:59 +0800 2020	Sun Dec 13 11:33:16 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A
application_1607830155379_0001	root	Reduce side join	MAPREDUCE	default	Sun Dec 13 11:30:16 +0800 2020	Sun Dec 13 11:32:57 +0800 2020	FINISHED	SUCCEEDED	<div></div>	History	N/A

At the bottom, it says 'Showing 1 to 3 of 3 entries' and has navigation links: 'First', 'Previous', '1', 'Next', 'Last'.

node1:19888/jobhistory/job/job_1607830155379_0001



MapReduce Job job_1607830155379_0001

Job Overview

Job Name: Reduce side join
User Name: root
Queue: default
State: SUCCEEDED
Uberized: false
Submitted: Sun Dec 13 11:30:16 CST 2020
Started: Sun Dec 13 11:30:28 CST 2020
Finished: Sun Dec 13 11:32:56 CST 2020
Elapsed: 2mins, 28sec
Diagnostics:
Average Map Time 1mins, 7sec
Average Shuffle Time 1mins, 4sec
Average Merge Time 1sec
Average Reduce Time 3sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Dec 13 11:30:22 CST 2020	node1:8042	logs

Task Type	Total		Complete	
Map	16		16	
Reduce	1		1	
Attempt Type	Failed		Killed	Successful
Maps	7	3		16
Reduces	0	0		1

node1:19888/jobhistory/job/job_1607830155379_0002



MapReduce Job job_1607830155379_0002

Job Overview

Job Name: word count
User Name: root
Queue: default
State: SUCCEEDED
Uberized: true
Submitted: Sun Dec 13 11:32:59 CST 2020
Started: Sun Dec 13 11:33:11 CST 2020
Finished: Sun Dec 13 11:33:16 CST 2020
Elapsed: 4sec
Diagnostics:
Average Map Time 2sec
Average Shuffle Time 0sec
Average Merge Time 0sec
Average Reduce Time 1sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Dec 13 11:33:05 CST 2020	node2:8042	logs

Task Type	Total		Complete	
Map	1		1	
Reduce	1		1	
Attempt Type	Failed		Killed	Successful
Maps	0	0		1
Reduces	0	0		1

全 | node1:19888/jobhistory/job/job_1607830155379_0003

Logged in as: dr.wno



MapReduce Job job_1607830155379_0003

Job Overview

Job Name: sort
User Name: root
Queue: default
State: SUCCEEDED
Uberized: true
Submitted: Sun Dec 13 11:33:19 CST 2020
Started: Sun Dec 13 11:33:33 CST 2020
Finished: Sun Dec 13 11:33:37 CST 2020
Elapsed: 3sec
Diagnostics:
Average Map Time 1sec
Average Shuffle Time 0sec
Average Merge Time 0sec
Average Reduce Time 1sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Dec 13 11:33:26 CST 2020	node1:8042	logs

Task Type	Total		Complete	
Map	1		1	
Reduce	1		1	
Attempt Type	Failed		Killed	Successful
Maps	0	0		1
Reduces	0	0		1

```

[root@node1 exp4]# hadoop jar EX4-Hadoop-1.0-SNAPSHOT.jar topcount.TopMerchants \
> -Dinput_data=exp4/data_format1/user_log_format1.csv,exp4/data_format1/user_info_format1.csv \
> -Doutput_dir=exp4/output
20/12/13 11:30:14 INFO client.RMProxy: Connecting to ResourceManager at node1/172.18.174.44:8032
20/12/13 11:30:15 INFO input.FileInputFormat: Total input paths to process : 2
20/12/13 11:30:16 INFO mapreduce.JobSubmitter: number of splits:16
20/12/13 11:30:16 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1607830155379_0001
20/12/13 11:30:17 INFO impl.YarnClientImpl: Submitted application application_1607830155379_0001
20/12/13 11:30:17 INFO mapreduce.Job: The url to track the job: http://node1:8088/proxy/application_1607830155379_0001
20/12/13 11:30:17 INFO mapreduce.Job: Running job: job_1607830155379_0001
20/12/13 11:30:29 INFO mapreduce.Job: Job job_1607830155379_0001 running in uber mode : false
20/12/13 11:30:29 INFO mapreduce.Job: map 0% reduce 0%
20/12/13 11:31:00 INFO mapreduce.Job: map 1% reduce 0%
  
```

```

[root@node1 exp4]# hdfs dfs -cat exp4/output/*
1: 4044, 7278
2: 3491, 3661
3: 1102, 3588
4: 3828, 3434
5: 4173, 3348
6: 3734, 3303
7: 2385, 3214
8: 4976, 3064
9: 798, 2997
10: 422, 2893
11: 1892, 2792
12: 1393, 2774
13: 4282, 2737
14: 1535, 2720
  
```

Spark程序

统计双十一最热门的商品（“添加购物车+购买+添加收藏夹”前100名））

代码：

EX4-Spark\src\main\java\topcount\TopItems.java

输出文件：

output/spark-output/TopItems

实现过程

思路和Hadoop MapReduce一样，在Spark上实现起来也比较方便，只需要调用特定的方法，利用java的匿名函数定义操作即可。

文件->read()方法->

Rdd->map(将一行字符串转化成数组)->

filter(符合条件的log)->

mapToPair(产生<item_id,1>)-> reduceByKey(产生<item_id,count>)-> map(反转key-value)->sortByKey(false)-

>map(反转key-value)->

take(100)->输出

运行代码的命令为：

```
spark-submit --class "topcount.TopItems" --master local EX4-Spark-1.0-SNAPSHOT.jar \
hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv \
hdfs://node1:9000/user/root/exp4/output
```

```
[root@node1 exp4]# spark-submit --class "topcount.TopItems" --master local EX4-Spark-1.0-SNAPSHOT.jar hdfs://node1:9000/user/root/exp4/data_format1/
user_log_format1.csv hdfs://node1:9000/user/root/exp4/output
20/12/13 14:04:45 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
20/12/13 14:04:45 INFO SparkContext: Running Spark version 2.4.7
20/12/13 14:04:45 INFO SparkContext: Submitted application: TopItemCount
20/12/13 14:04:45 INFO SecurityManager: Changing view acls to: root
20/12/13 14:04:45 INFO SecurityManager: Changing modify acls to: root
20/12/13 14:04:45 INFO SecurityManager: Changing view acls groups to:
20/12/13 14:04:45 INFO SecurityManager: Changing modify acls groups to:
20/12/13 14:04:45 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(root); groups
with view permissions: Set(); users with modify permissions: Set(root); groups with modify permissions: Set()
20/12/13 14:04:46 INFO Utils: Successfully started service 'sparkDriver' on port 41570.
20/12/13 14:04:46 INFO SparkEnv: Registering MapOutputTracker
```

```
20/12/13 15:15:21 INFO ShutdownHookManager: Detecting
[root@node1 exp4]# hdfs dfs -cat exp4/output/*
(1,(191499,2494))
(2,(353560,2250))
(3,(1059899,1917))
(4,(713695,1754))
(5,(655904,1674))
(6,(67897,1572))
(7,(221663,1547))
(8,(1039919,1511))
(9,(454937,1387))
(10,(81360,1361))
(11,(514725,1356))
(12,(783997,1351))
(13,(823766,1343))
(14,(107407,1319))
(15,(889095,1272))
```

结果存放在了spark-output/TopItems中

统计最受年轻人(age<30)关注的商家（“添加购物车+购买+添加收藏夹”前100名）

代码：

EX4-Spark\src\main\java\topcount\TopMerchants.java

输出文件：

output/spark-output/TopMerchants

实现过程

思路也和MapReduce程序一样，在Spark中更好操作，分别读入两个表形成Rdd，先筛选出1111当天action_type!=0的log，转为PairRdd：<user_id,seller_id>，然后对user_info_format表操作产生PairRdd: <user_id,age_range>，最后对两个PairRdd进行join操作，最后筛选出age_range in (1,2,3)的单位，进行wordcount并排序。

运行代码的命令为：

```
spark-submit --class "topcount.TopMerchants" --master local EX4-Spark-1.0-  
SNAPSHOT.jar \  
hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv,\  
hdfs://node1:9000/user/root/exp4/data_format1/user_info_format1.csv \  
hdfs://node1:9000/user/root/exp4/output
```

```

t 15 tasks are for partitions Vector(0))
20/12/13 15:52:55 INFO TaskSchedulerImpl: Adding task set 8.0 with 1 tasks
20/12/13 15:52:55 INFO TaskSetManager: Starting task 0.0 in stage 8.0 (TID 61, localhost, executor driver, partition
20/12/13 15:52:55 INFO Executor: Running task 0.0 in stage 8.0 (TID 61)
20/12/13 15:52:55 INFO ShuffleBlockFetcherIterator: Getting 15 non-empty blocks including 15 local blocks and 0 remote
20/12/13 15:52:55 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
20/12/13 15:52:55 INFO Executor: Finished task 0.0 in stage 8.0 (TID 61). 3552 bytes result sent to driver
20/12/13 15:52:55 INFO TaskSetManager: Finished task 0.0 in stage 8.0 (TID 61) in 94 ms on localhost (executor driver)
20/12/13 15:52:55 INFO TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have all completed, from pool
20/12/13 15:52:55 INFO DAGScheduler: ResultStage 8 (take at TopMerchants.java:85) finished in 0.100 s
20/12/13 15:52:55 INFO DAGScheduler: Job 1 finished: take at TopMerchants.java:85, took 0.795489 s
1: 4044, 7248
2: 3491, 3634
3: 1102, 3565
4: 3828, 3416
5: 4173, 3333
6: 3734, 3277
7: 2385, 3209
8: 4976, 3048
9: 798, 2988
10: 422, 2874
11: 1892, 2779
12: 1393, 2756

```

```

[root@node1 exp4]# hdfs dfs -cat exp4/output/*
(1,(4044,7248))
(2,(3491,3634))
(3,(1102,3565))
(4,(3828,3416))
(5,(4173,3333))
(6,(3734,3277))
(7,(2385,3209))
(8,(4976,3048))
(9,(798,2988))
(10,(422,2874))
(11,(1892,2779))
(12,(1393,2756))
(13,(4282,2713))
(14,(1535,2706))
(15,(4760,2643))
(16,(3760,2595))
(17,(4644,2582))

```

结果存放在了spark-output/TopMerchants中

实验4-第2题

编写Spark程序统计双十一购买了商品的男女比例，以及购买了商品的买家年龄段的比例；

统计双十一购买了商品的男女比例

代码：

EX4-Spark\src\main\java\topcount\CountGender.java

输出文件：

output/spark-output/CountGender

实现过程

也是两个表的连接，然后筛选。如果从SQL的角度来看，查询可以看成是：user_log表左连接user_info表，日期=1111，action_type=2，按年龄分组，统计不重复的user_id数，还要考虑排除异常的age_range值。

表的连接在第1题的编程中已经实现了，这里就不再重述，这题的关键是去除重复的user_id，然后按照年龄分组。

合并之后的PairRdd形式为：<user_id,(item_id,gender)>，首先转换为<user_id,gender>，然后去除重复的key值：

```
JavaPairRDD<String,String> user_gender = joined_logs.mapToPair(
    s -> {
        return new Tuple2<String,String>(s._1,s._2._2);
    }
);
//去除重复的user_id
user_gender = user_gender.reduceByKey((x,y) -> x);
```

分组的思路有很多，我还是把这个任务当成WordCount来做：

再将性别编号gender变为key，value是1，相当于对gender做WordCount，最后获得gender=0和1的count数，计算比例即可。

运行命令：

```
spark-submit --class "topcount.CountGender" --master local EX4-Spark-1.0-
SNAPSHOT.jar \
hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv,\
hdfs://node1:9000/user/root/exp4/data_format1/user_info_format1.csv \
hdfs://node1:9000/user/root/exp4/output
```

输出结果存放在了spark-output/CountGender中，格式为 (性别代号，总数)

```
[root@node1 exp4]# hdfs dfs -cat exp4/output/*
(0,285638)
(1,121670)
(2,10426)
```

```
20/12/13 16:22:01 INFO TaskSchedulerImpl: Finished task 5.0 in stage 4.0 (TID 60) in 12 ms on localhost (executor d
20/12/13 16:22:01 INFO ShuffleBlockFetcherIterator: Getting 15 non-empty blocks including 15 local blocks and 0 re
20/12/13 16:22:01 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
20/12/13 16:22:01 INFO Executor: Finished task 5.0 in stage 4.0 (TID 60). 1280 bytes result sent to driver
20/12/13 16:22:01 INFO TaskSetManager: Finished task 5.0 in stage 4.0 (TID 60) in 13 ms on localhost (executor dri
20/12/13 16:22:01 INFO TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
20/12/13 16:22:01 INFO DAGScheduler: ResultStage 4 (collect at CountGender.java:82) finished in 0.176 s
20/12/13 16:22:01 INFO DAGScheduler: Job 0 finished: collect at CountGender.java:82, took 73.077025 s
0: 285638
1: 121670
2: 10426
0: 70.1%
1: 29.9%
```

最后统计得到的女性比例为70.1%，男性比例为29.9%。

统计购买了商品的买家年龄段的比例

代码：

EX4-Spark\src\main\java\topcount\CountAge.java

输出文件：

output/spark-output/CountAge

实现过程

思路和统计年龄比例是一样的，只是筛选和分组的属性不同，还要注意年龄段7和8都表示年龄 ≥ 50 。

运行命令：

```
spark-submit --class "topcount.CountAge" --master local EX4-Spark-1.0-SNAPSHOT.jar \
\
hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv,\
hdfs://node1:9000/user/root/exp4/data_format1/user_info_format1.csv \
hdfs://node1:9000/user/root/exp4/output
```

```
[root@node1 exp4]# spark-submit --class "topcount.CountAge" --master local EX4-Spark-1.0-SNAPSHOT.jar \
> hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv,\
> hdfs://node1:9000/user/root/exp4/data_format1/user_info_format1.csv \
> hdfs://node1:9000/user/root/exp4/output
20/12/13 16:40:24 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
20/12/13 16:40:25 INFO SparkContext: Running Spark version 2.4.7
20/12/13 16:40:25 INFO SparkContext: Submitted application: CountAge
20/12/13 16:40:25 INFO SecurityManager: Changing view acls to: root
20/12/13 16:40:25 INFO SecurityManager: Changing modify acls to: root
20/12/13 16:40:25 INFO SecurityManager: Changing view acls groups to:
20/12/13 16:40:25 INFO SecurityManager: Changing modify acls groups to:
20/12/13 16:40:25 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view p
20/12/13 16:41:46 INFO TaskSetManager: Finished task 11.0 in stage 4.0 (TID 60) in 21 ms on localhost (exe
20/12/13 16:41:46 INFO TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
20/12/13 16:41:46 INFO DAGScheduler: ResultStage 4 (collect at CountAge.java:86) finished in 0.293 s
20/12/13 16:41:46 INFO DAGScheduler: Job 0 finished: collect at CountAge.java:86, took 71.057173 s
: 26
0: 90638
1: 24
2: 52420
3: 110952
4: 79649
5: 40601
6: 35257
7: 6924
8: 1243
1: 0.0%
2: 16.0%
3: 33.9%
4: 24.4%
5: 12.4%
6: 10.8%
7: 2.1%
8: 0.4%
```

```
[root@node1 exp4]# hdfs dfs -cat exp4/output/*
(,26)
(0,90638)
(1,24)
(2,52420)
(3,110952)
(4,79649)
(5,40601)
(6,35257)
(7,6924)
(8,1243)
```

输出结果存放在了spark-output/CountAge中，格式为 (年龄代号，总数)

统计得到各年龄段比例分别为：

1: 0.0%
2: 16.0%
3: 33.9%
4: 24.4%
5: 12.4%
6: 10.8%
7+8: 2.5%

实验4-第3题

基于Hive或Spark SQL查询双十一购买了商品的男女比例，以及购买了商品的买家年龄段的比例；

查询双十一购买了商品的男女比例

代码：

EX4-Spark\src\main\java\topcount\SqlCountGender.java

输出文件：

output/spark-output/SqlCountGender

实现过程

首先利用源数据创建Dataset，即两个表user_log和user_info，由于后面使用SQL语句要用到表的名称，所以可以利用Dataset注册一个临时视图：

```
user_log.createTempView("user_log");  
user_info.createTempView("user_info");
```

使用SQL语句查询双十一当天购买了商品的男女人数，思路是：user_log表左连接user_info表，日期=1111，action_type=2，按年龄分组，统计不重复的user_id数，还要考虑排除异常的age_range值。

SQL语句如下：

```
select gender,count(distinct ul.user_id) buy_num  
from user_log ul  
left join user_info ui  
on ul.user_id = ui.user_id  
where action_type=2 and time_stamp=1111 and gender in (0,1)  
group by gender;
```

使用SparkSession的sql()方法执行上述语句，就得到了各性别在1111发生购买行为的人数。然后还要求各人数所占的比例，可以调用spark sql的各种方法来处理：

```
result = result.withColumn("percent",format_number(col("buy_num").divide(
sum("buy_num").over()).multiply(100),5));
```

运行命令：

```
spark-submit --class "topcount.SqlCountGender" --master local EX4-Spark-1.0-
SNAPSHOT.jar \
hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv,\
hdfs://node1:9000/user/root/exp4/data_format1/user_info_format1.csv \
hdfs://node1:9000/user/root/exp4/output
```

输出结果存放在了spark-output/SqlCountGender中，格式为 [性别代号，总数，比例]

```
20/12/13 18:02:32 INFO TaskSchedulerImpl: Removed taskset 0.0, whose tasks have all completed, from pool
20/12/13 18:02:32 INFO DAGScheduler: ResultStage 6 (show at SqlCountGender.java:53) finished in 0.185 s
20/12/13 18:02:32 INFO DAGScheduler: Job 3 finished: show at SqlCountGender.java:53, took 144.393136 s
+-----+-----+-----+
|gender|buy_num| percent|
+-----+-----+-----+
|      0| 285638|70.12826|
|      1| 121670|29.87174|
+-----+-----+-----+

[root@node1 exp4]# hdfs dfs -cat exp4/output/*
[0,285638,70.12826]
[1,121670,29.87174]
```

统计得到女性的比例为70.1%，男性比例为29.9%

查询购买了商品的买家年龄段的比例

代码：

EX4-Spark\src\main\java\topcount\SqlCountAge.java

输出文件：

output/spark-output/SqlCountAge

实现过程

操作和上述任务同理，实现逻辑可用SQL语句表示：

```
select age_range, count(distinct ul.user_id) buy_num, time_stamp
from user_log ul
left join user_info ui
on ul.user_id = ui.user_id
where action_type=2 and time_stamp=1111 and age_range>=1 and age_range<=8
group by age_range;
```

运行命令：

```
spark-submit --class "topcount.SqlCountAge" --master local EX4-Spark-1.0-
SNAPSHOT.jar \
hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv,\
hdfs://node1:9000/user/root/exp4/data_format1/user_info_format1.csv \
hdfs://node1:9000/user/root/exp4/output
```

输出结果存放在了spark-output/SqlCountAge中，格式为 [年龄代号，总数，比例]

```
20/12/13 18:14:36 INFO BlockManagerMaster: BlockManagerMaster stopped
20/12/13 18:14:36 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
20/12/13 18:14:36 INFO SparkContext: Successfully stopped SparkContext
20/12/13 18:14:36 INFO ShutdownHookManager: Shutdown hook called
20/12/13 18:14:36 INFO ShutdownHookManager: Deleting directory /tmp/spark-3f6557fa-4c72-421e-9a85-e1d6013507ad
20/12/13 18:14:36 INFO ShutdownHookManager: Deleting directory /tmp/spark-9d2783d1-ed3c-43bd-b4ac-5104db582f42
[root@node1 exp4]# hdfs dfs -cat exp4/output/*
[7,6992,2.12498]
[3,111654,33.93336]
[8,1266,0.38476]
[5,40777,12.39276]
[6,35464,10.77805]
[1,24,0.00729]
[4,79991,24.31049]
[2,52871,16.06831]
```

统计得到各年龄段比例分别为：

1: 0.0%
 2: 16.0%
 3: 33.9%
 4: 24.3%
 5: 12.4%
 6: 10.8%
 7+8: 2.5%

实验4-第4题

预测给定的商家中，哪些新消费者在未来会成为忠实客户，即需要预测这些新消费者在6个月月内再次购买的概率。基于Spark MLlib编写程序预测回头客，评估实验结果的准确率。

构建特征

代码：

pyspark-mllib/data_proc.py

Notebook：

pyspark-mllib/data_proc.ipynb

输出文件：

LibSVMFile：output/procd_train_real

实现过程

特征的构建是模仿天池上该比赛的一篇notebook来实施的(<https://tianchi.aliyun.com/notebook-ai/detail?spm=5176.12586969.1002.3.1f1b1d8a8ezV94&postId=143593>)



由于特征数量比较多，调试比较麻烦，使用的语言改为python，与示例不同的是数据处理过程全部使用pyspark实现。

以下是需要构建的特征：

1. 用户的年龄(age_range)
2. 用户的性别(gender)
3. 某用户在该商家日志的总条数(total_logs)
4. 用户浏览的商品的数目，就是浏览了多少个商品(unique_item_ids)
5. 浏览的商品的种类的数目，就是浏览了多少种商品(categories)
6. 用户浏览的天数(browse_days)
7. 用户单击的次数(one_clicks)
8. 用户添加购物车的次数(shopping_carts)
9. 用户购买的次数(purchase_times)
10. 用户收藏的次数(favourite_times)

下面简要介绍一些涉及Spark DataFrame的操作：

(1)添加用户年龄和性别特征

使用join操作按照user_id将训练集和user_info表合并即可

```
df_train = df_train.join(user_info, ["user_id"], "left")
```

(2)添加总日志条目特征total_logs

将user_log表按照用户id和卖家id分组计数，将卖家id列的名称改为'merchant_id'，然后与训练集表进行连接：

```
total_logs_temp = user_log.groupby(["user_id", "seller_id"]).count()
total_logs_temp =
total_logs_temp.withColumnRenamed("seller_id", "merchant_id").withColumnRenamed("count", "total_logs")
df_train = df_train.join(total_logs_temp, ["user_id", "merchant_id"], "left")
df_train.limit(3).show()
```



```

+-----+-----+-----+-----+-----+-----+
|user_id|merchant_id|label|age_range|gender|total_logs|
+-----+-----+-----+-----+-----+-----+
|   464|      4718|   0|      6|   0|      4|
|   867|      3152|   0|      3|   0|     17|
|  1882|      4377|   0|      6|   1|      4|
+-----+-----+-----+-----+-----+-----+

```

(3)添加用户浏览的商品的数目unique_item_ids

这里需要筛选的是用户在每个卖家浏览的商品数目，商品不可以重复计数，需要进行去重操作。这里的思路是对user_log表先按照user_id,seller_id,item_id进行分组计数，这样就获得了去重后的item_id表，然后再按照user_id,seller_id进行分组计数，最后与训练集合并即可。

```

#unique_item_ids特征添加
unique_item_ids_temp = user_log.groupby(["user_id","seller_id","item_id"]).count()
unique_item_ids_temp =
unique_item_ids_temp.selectExpr("user_id","seller_id","item_id")
unique_item_ids_temp =
unique_item_ids_temp.groupBy(["user_id","seller_id"]).count()
unique_item_ids_temp =
unique_item_ids_temp.withColumnRenamed("seller_id","merchant_id").withColumnRenamed("count","unique_item_ids")
df_train = df_train.join(unique_item_ids_temp,["user_id","merchant_id"],"left")
df_train.limit(3).show()

```

```

+-----+-----+-----+-----+-----+-----+-----+
|user_id|merchant_id|label|age_range|gender|total_logs|unique_item_ids|
+-----+-----+-----+-----+-----+-----+-----+
|   464|      4718|   0|      6|   0|      4|      2|
|   867|      3152|   0|      3|   0|     17|      3|
|  1882|      4377|   0|      6|   1|      4|      1|
+-----+-----+-----+-----+-----+-----+-----+

```

商品类别categories，浏览天数browse_days的构建与此方法相似。

(4)对用户的行为类型构建one_clicks、shopping_carts、purchase_times、favourite_times等特征

首先还是将user_log表按照user_id,seller_id,action_type分组计数，然后根据不同的筛选条件定义udf函数，创建新的计数列，以one_clicks列的创建为例：

```

#按照user_id和seller_id分组
one_clicks_temp = user_log.groupby(["user_id","seller_id","action_type"]).count()
one_clicks_temp =
one_clicks_temp.withColumnRenamed("seller_id","merchant_id").withColumnRenamed("count","times")

```

```
#定义udf函数, 如果action_type为0, 返回相应计数, 否则返回0
def click_time(action,times):
    if action == 0:
        return 0
    else:
        return times
udf_click_time = functions.udf(click_time,IntegerType())
one_clicks_temp =
one_clicks_temp.withColumn("one_clicks",udf_click_time("action_type","times"))
```

最后按照user_id,seller_id分组合并, 使用聚合函数sum将计数相加

```
four_features = one_clicks_temp.groupby(["user_id","merchant_id"]).sum()
four_features =
four_features.selectExpr("user_id","merchant_id","`sum(one_clicks)` as
one_clicks",
"`sum(shopping_carts)` as shopping_carts","`sum(purchase_times)` as
purchase_times","`sum(favor_times)` as favor_times")
four_features.limit(3).show()
```

user_id	merchant_id	one_clicks	shopping_carts	purchase_times	favor_times
26535	3322	0	0	0	0
324592	1704	0	0	0	0
109128	3639	0	0	0	0

使用join合并到训练集即可:

```
df_train = df_train.join(four_features,["user_id","merchant_id"],"left")
```

(5)缺失值处理: 由于构建的特征值几乎都是条目或记录数, 缺失即可以认为是没有记录, 将值填充为0是一种简单可行的办法:

```
#填充缺失值
#策略是将后8个特征所有null值填充为0
df_train_filled = df_train.fillna(0)
```

(6)将处理好的训练集数据以LibSVMFile格式储存

为了方便训练阶段将数据输入模型, 先将训练集转化成LabeledPoint即(label,features)的格式, 然后以LibSVMFile格式存入到HDFS中, 这样在训练模型时可以直接将从文件读取的数据输入模型。

```
#先转成RDD
df_train_rdd = df_train_filled.rdd
#改成(label,features)的格式
df_train_rdd = df_train_rdd.map(lambda line:
LabeledPoint(line[2],Vectors.dense(line[3:])))
#保存为LibSVMFile格式, 方便后面训练使用
from pyspark.mllib.util import MLUtils
MLUtils.saveAsLibSVMFile(df_train_rdd,
"hdfs://node1:9000/user/root/exp4/procd_train_real")
```

使用训练集数据进行训练和预测

代码：

pyspark-mllib/train.py

Notebook：

pyspark-mllib/train.ipynb

实现过程

(1) 训练数据和测试数据的划分

导入特征构建阶段输出的LibSVM文件，将数据按照6：4划分为训练集和测试集，将训练集放入cache，提高运算速度。

```
data = MLUtils.loadLibSVMFile(sc,
"hdfs://node1:9000/user/root/exp4/procd_train_real")
# Split data into training (60%) and test (40%)
training, test = data.randomSplit([0.6, 0.4], seed=11)
training.cache()
```

(2) Logistic Regression

使用LogisticRegressionWithLBFGS，将测试集作为train方法的参数输入，即可训练一个Logistic Regression分类器。

```
# Logistic Regression
# Run training algorithm to build the model
model = LogisticRegressionWithLBFGS.train(training)
```

测试也非常简单，对test使用map操作，将每一行的features输入模型，获得预测的标签，然后计算错误率。

```
# Compute raw scores on the test set
predictionAndLabels = test.map(lambda lp: (float(model.predict(lp.features)),
lp.label))
testErr = predictionAndLabels.filter(lambda lp: lp[0] != lp[1]).count() /
float(test.count())
print("Test Error = " + str(testErr))
```

Test Error = 0.0625555970042182

(3) 支持向量机

SVM的训练和检验方法与Logistic Regression相似，只是调用的模型不同

```
# SVM
# Build the model
from pyspark.mllib.classification import SVMWithSGD, SVMModel
svm_model = SVMWithSGD.train(training, iterations=100)
# Evaluating the model on training data
labelsAndPreds = training.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda lp: lp[0] != lp[1]).count() /
float(training.count())
print("Training Error = " + str(trainErr))
```

Training Error = 0.06195103539602219

(4) 决策树

决策树模型有很多参数可以调整，我使用的是官方例子中的参数，决策树训练完之后还可以打印树的结构

```
# DecisionTree
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
# Train a DecisionTree model.
# Empty categoricalFeaturesInfo indicates all features are continuous.
model = DecisionTree.trainClassifier(training, numClasses=2,
categoricalFeaturesInfo={},
                                impurity='gini', maxDepth=5, maxBins=32)
# Evaluate model on test instances and compute test error
predictions = model.predict(test.map(lambda x: x.features))
labelsAndPredictions = test.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(
    lambda lp: lp[0] != lp[1]).count() / float(test.count())
print('Test Error = ' + str(testErr))
```

```
print('Learned classification tree model:')
print(model.toDebugString())
```

```
Test Error = 0.061589524328770795
Learned classification tree model:
DecisionTreeModel classifier of depth 5 with 21 nodes
If (feature 3 <= 6.5)
  If (feature 8 <= 1.5)
    If (feature 3 <= 2.5)
      If (feature 4 <= 3.5)
        Predict: 0.0
      Else (feature 4 > 3.5)
        Predict: 1.0
    Else (feature 3 > 2.5)
      Predict: 0.0
  Else (feature 8 > 1.5)
    Predict: 0.0
Else (feature 3 > 6.5)
  If (feature 3 <= 19.5)
    Predict: 0.0
  Else (feature 3 > 19.5)
    If (feature 4 <= 5.5)
      If (feature 8 <= 8.5)
        Predict: 0.0
      Else (feature 8 > 8.5)
        If (feature 5 <= 1.5)
          Predict: 0.0
        Else (feature 5 > 1.5)
          Predict: 1.0
    Else (feature 4 > 5.5)
      If (feature 8 <= 2.5)
        Predict: 0.0
      Else (feature 8 > 2.5)
        If (feature 5 <= 17.5)
          Predict: 0.0
        Else (feature 5 > 17.5)
          Predict: 1.0
```

我还使用了朴素贝叶斯，随机森林，GBDT等算法进行训练和测试，这些模型的使用方法和上面几个模型都差不多，训练的结果可以在对应的Notebook中查看，在训练集上的测试错误率差不多都在0.07左右，由于训练集是一个严重不平衡的数据集，所以这样的错误率没有意义，Logistic Regression结果的AUC值只有0.5左右，因此上述的模型在参数未经调整的情况下，即使错误率看上去很低，但实际上性能很差。。。

对测试数据进行预测，提交预测结果

代码：

pyspark-mllib/data_proc_testdata.py
pyspark-mllib/train_save.py
pyspark-mllib/test.py

Notebook :


pyspark-mllib/data_proc_testdata.ipynb
pyspark-mllib/train_save.ipynb
pyspark-mllib/test.ipynb

实现过程


用相同方法对测试集构建特征(data_proc_testdata.py) · 然后使用所有训练集数据训练模型并保存(train_save.py) · 最后将测试集特征输入模型 · 获得预测值 · 并保存(test.py)。

提交结果


Logistic Regression

 **日期:** 2020-12-20 14:08:52 **排名:** 无
score: 0.5015744

支持向量机

 **日期:** 2020-12-20 14:18:59 **排名:** 无
score: 0.5156678

GBDT

 **日期:** 2020-12-20 14:51:00 **排名:** 无
score: 0.5000562

分数都很低哈哈哈哈哈。

遇到的一些问题

1. 使用完整数据集运行MapReduce程序时出现内存溢出，系统自动杀死一些进程

问题描述

在用MapReduce操作完整的数据集时，会出现一些container被莫名其妙杀死，导致MapReduce进度回退的情况

```
20/12/09 19:21:01 INFO mapreduce.Job: map 51% reduce 0%
20/12/09 19:21:01 INFO mapreduce.Job: Task Id : attempt_1607512066198_0001_m_000013_1, Status : FAILED
Container killed on request. Exit code is 137
Container exited with a non-zero exit code 137
Killed by external signal

20/12/09 19:21:01 INFO mapreduce.Job: Task Id : attempt_1607512066198_0001_m_000014_0, Status : FAILED
Exception from container-launch.
Container id: container_1607512066198_0001_01_000016
Exit code: 1
Stack trace: ExitCodeException exitCode=1:
    at org.apache.hadoop.util.Shell.runCommand(Shell.java:585)
    at org.apache.hadoop.util.Shell.run(Shell.java:482)
    at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:776)
    at org.apache.hadoop.yarn.server.nodemanager.DefaultContainerExecutor.launchContainer(DefaultContainerExecutor.java:212)
    at org.apache.hadoop.yarn.server.nodemanager.containermanager.launcher.ContainerLaunch.call(ContainerLaunch.java:302)
    at org.apache.hadoop.yarn.server.nodemanager.containermanager.launcher.ContainerLaunch.call(ContainerLaunch.java:82)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:748)

Container exited with a non-zero exit code 1

20/12/09 19:21:02 INFO mapreduce.Job: map 52% reduce 0%
```

使用 `dmesg -T | grep -i memory` 命令查看关于内存的信息可以发现，在MapReduce运行过程中有一些java进程发生了OOM，系统自动杀死了这些进程，被杀死的大多是container的进程，但有时候还会杀死Namenode进程导致整个任务中断。

```
[Wed Dec 9 19:18:09 2020] [<ffffffff81186bd6>] out_of_memory+0x4b6/0x4f0
[Wed Dec 9 19:18:09 2020] Out of memory: Kill process 16767 (java) score 95 or sacrifice child
[Wed Dec 9 19:18:12 2020] [<ffffffff81186bd6>] out_of_memory+0x4b6/0x4f0
[Wed Dec 9 19:18:12 2020] Out of memory: Kill process 16766 (java) score 96 or sacrifice child
[Wed Dec 9 19:18:20 2020] [<ffffffff81186bd6>] out_of_memory+0x4b6/0x4f0
[Wed Dec 9 19:18:20 2020] Out of memory: Kill process 16827 (java) score 95 or sacrifice child
[Wed Dec 9 19:19:01 2020] [<ffffffff81186bd6>] out_of_memory+0x4b6/0x4f0
[Wed Dec 9 19:19:01 2020] Out of memory: Kill process 17121 (java) score 87 or sacrifice child
[Wed Dec 9 19:19:01 2020] [<ffffffff81186bd6>] out_of_memory+0x4b6/0x4f0
[Wed Dec 9 19:19:01 2020] Out of memory: Kill process 17122 (java) score 86 or sacrifice child
```

问题原因

Container使用的内存量超过约定值，触发产生KILL_CONTAINER事件，container被监控线程强制杀死

解决方案

需要调整下mapreduce的资源配置，修改配置文件：yarn-site.xml设置yarn.scheduler.minimum-allocation-mb参数和yarn.scheduler.maximum-allocation-mb，即单个容器可以申请的最小与最大内存

RM内存资源配置——两个参数 (yarn-site.xml)

```
<property>
<description>The minimum allocation for every container request at the RM,
in MBs. Memory requests lower than this won't take effect,
and the specified value will get allocated at minimum.</description>
<name>yarn.scheduler.minimum-allocation-mb</name>
<value>1024</value>
</property>

<property>
<description>The maximum allocation for every container request at the RM,
in MBs. Memory requests higher than this won't take effect,
and will get capped to this value.</description>
<name>yarn.scheduler.maximum-allocation-mb</name>
<value>8192</value>
</property>
```

它们表示单个容器可以申请的最小与最大内存。

2. 使用Hive对完整数据集执行查询操作失败

问题描述

使用Hive对完整数据集进行查询，MapReduce作业一直卡在0%，随后直接挂掉。
查看内存信息之后也发现了OOM killer事件。

问题原因

由于Hive使用的默认参数，没有自定义最大的map和reduce数，导致hive分配了过多的map数，使得内存超限，导致OOM。

解决方案

(1) 调整Hive的参数

1、合并输入文件

```
set mapred.max.split.size=256000000; #每个Map最大输入大小
set mapred.min.split.size.per.node=100000000; #一个节点上split的至少的大小
set mapred.min.split.size.per.rack=100000000; #一个交换机下split的至少的大小
set hive.input.format=org.apache.Hadoop.hive ql.io.CombineHiveInputFormat; #执行Map前进行小文件合并
```

开启org.apache.hadoop.hive ql.io.CombineHiveInputFormat后，一个data node节点上多个小文件会进行合并，合并文件数由mapred.max.split.size限制的大小决定，mapred.min.split.size.per.node决定了多个data node上的文件是否需要合并，mapred.min.split.size.per.rack决定了多个交换机上的文件是否需要合并。

2、合并输出文件

```
set hive.merge.mapfiles = true #在Map-only的任务结束时合并小文件
set hive.merge.mapredfiles = true #在Map-Reduce的任务结束时合并小文件
set hive.merge.size.per.task = 256*1000*1000 #合并文件的大小
set hive.merge.smallfiles.avgsize=16000000 #当输出文件的平均大小小于该值时，启动一个独立的map-reduce任务进行文件merge。
```

(2) 升级master节点的配置

最后我实在忍受不了master节点1核2GB的低端配置，直接将其升级成了2核8GB，之前的问题就没有了。

执行 `select count(*) from user_log;`

```
2020-12-25 13:52:49,386 Stage-1 map = 50%, reduce = 4%, Cumulative CPU 321.83 sec
2020-12-25 13:52:51,474 Stage-1 map = 50%, reduce = 8%, Cumulative CPU 325.76 sec
2020-12-25 13:52:53,600 Stage-1 map = 58%, reduce = 8%, Cumulative CPU 329.84 sec
2020-12-25 13:52:54,674 Stage-1 map = 75%, reduce = 13%, Cumulative CPU 331.11 sec
2020-12-25 13:52:55,721 Stage-1 map = 83%, reduce = 13%, Cumulative CPU 333.21 sec
2020-12-25 13:52:56,753 Stage-1 map = 92%, reduce = 13%, Cumulative CPU 333.47 sec
2020-12-25 13:52:57,777 Stage-1 map = 92%, reduce = 29%, Cumulative CPU 334.57 sec
2020-12-25 13:53:09,116 Stage-1 map = 100%, reduce = 29%, Cumulative CPU 346.27 sec
2020-12-25 13:53:11,172 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 348.11 sec
MapReduce Total cumulative CPU time: 5 minutes 48 seconds 110 msec
Ended Job = job_1608875338746_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 8 Reduce: 1 Cumulative CPU: 348.11 sec HDFS Read: 1910303918 B
Total MapReduce CPU Time Spent: 5 minutes 48 seconds 110 msec
OK
54925331
Time taken: 203.261 seconds, Fetched: 1 row(s)
```

3. 在Notebook上运行pyspark程序，SparkSubmit崩掉

问题描述

在Notebook上调试pyspark程序，导入了很多数据。同时为了直观输出每一步的操作结果，频繁执行了很多action方法，比如show()，导致最后JVM报错：

```
# A fatal error has been detected by the Java Runtime Environment:
#
# SIGBUS (0x7) at pc=0x00007f9d3d100c50, pid=29904, tid=0x00007f9ca2bfb700
#
.....
#
# Failed to write core dump. Core dumps have been disabled. To enable core
dumping, try "ulimit -c unlimited" before starting Java again
#
.....
```

问题原因

默认情况下Linux服务起的core file size设置为0，需要调整该参数，但是这个参数并不能解决问题；问题的根本原因在于服务器的运行应用程序的打开文件的最大数及最大进程数设置的相对较小默认为4096

需要修改如下配置：

/etc/security/limits.conf

解决方案

我最终没有选择修改配置，而是选择在python文件执行的中间阶段将已处理完的表先存到HDFS中，然后另起一个session，进行下一步操作，当需要将表合并时再读入相应的临时表。

```

[23] ▶ M4
#先将处理好的暂时写到文件中
df_train.write.options(header="true").csv("hdfs://node1:9000/user/root/exp4/procd_train_temp.csv", mode = 'overwrite')

[24] ▶ M4
#为了避免jvm崩溃，只能另起一个session
spark.stop()
spark = SparkSession\
    .builder\
    .master("local")\
    .appName("DataProcess2")\
    .config("spark.executor.memory", "3g")\
    .config("spark.executor.instances", "5")\
    .getOrCreate()

[25] ▶ M4
#导入user_log数据
user_log = spark.read.csv(r"hdfs://node1:9000/user/root/exp4/data_format1/user_log_format1.csv", encoding='utf8',
header=True, inferSchema=True)

[26] ▶ M4

```

同时减少不必要的action操作，尽量先做transformation，然后再做action，这样可以提高效率，减少不必要的开销。

4.遗留的SparkSubmit进程杀不死

问题描述

使用Notebook运行pyspark程序时，有时候忘记关掉SparkSession或者SparkContext，导致SparkSubmit进程一直没有关掉

```

[root@node1 ~]# jps
32497 DataNode
2211 SparkSubmit
32359 NameNode
32668 SecondaryNameNode
2286 Jps

```

而使用 kill -9 pid 命令依然不能杀死这个遗留进程

问题原因

kill -9发送SIGKILL信号将其终止，但是以下两种情况不起作用：

- a、该进程处于"Zombie"状态（使用ps命令返回defunct的进程）。此时进程已经释放所有资源，但还未得到其父进程的确认。"Zombie"进程要等到下次重启时才会消失，但它的存在不会影响系统性能。
- b、该进程处于"kernel mode"（核心态）且在等待不可获得的资源。处于核心态的进程忽略所有信号处理，因此对于这些一直处于核心态的进程只能通过重启系统实现。进程在AIX 中会处于两种状态，即用户态和核心态。只有处于用户态的进程才可以用"kill"命令将其终止

问题解决

输入命令 cat /proc/<pid>/status查看进程信息，找到父进程号ppid
然后杀死父进程 kill -9 ppid，遗留的SparkSubmit也就不见了。

环境搭建

Spark

单机模式

Spark的安装比较的简单，只需要将下载后的安装包解压，然后设置一下环境变量即可运行。

```
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

```
[root@node1 spark]# spark-shell
20/12/06 15:23:28 WARN NativeCodeLoader: Unable to load native-hadoop library for your p
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLev
Spark context Web UI available at http://node1:4040
Spark context available as 'sc' (master = local[*], app id = local-1607239421530).
Spark session available as 'spark'.
Welcome to

  ____      __
 / ___ |__ /  / /  ___
/  _ \|_ \|  /  /  / _ \
|  __|___) /  /  /  __/
|___|_____/  /  /___|___|
                    version 2.4.7

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_261)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

搭建集群

master节点——node1

编辑conf/slaves文件，在里面设置worker节点的主机名称

```
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

# A Spark Worker will be started on each of the machines listed here.
node2
~
```

编辑conf/spark-defaults.conf，设置主机名称，日志路径，executor的资源分配

```
spark.master                node1:7077
spark.eventLog.enabled      true
spark.eventLog.dir          hdfs://node1:9000/history
spark.driver.memory 1g
spark.executor.cores 1
spark.executor.memory 2g
```

编辑conf/spark-env.sh，设置history-server的ui，本机名称/IP，worker的资源分配

```
export SPARK_HISTORY_OPTS="-Dspark.history.ui.port=18080 -Dspark.history.retainedApplications=3 -Dspark.history.fs.logDi
story"
export SPARK_LOCAL_IP=node1
export SPARK_WORKER_MEMORY=2g
```

worker节点——node2

配置和master节点相同，将conf/spark-env.sh中的SPARK_LOCAL_IP改为本机名称/IP即可

启动集群

先启动HDFS，然后运行start-master.sh启动master节点，运行start-slaves.sh启动worker节点，运行start-history-server.sh启动历史记录server，方便查任务的运行记录

```
[root@node1 conf]# jps
29954 SecondaryNameNode
29783 DataNode
30106 Master
30220 HistoryServer
29645 NameNode
30301 Jps
```

```
[root@node2 conf]# jps
22563 DataNode
22709 Jps
22664 Worker
```

← → ↻ 不安全 | node1:8080



Spark Master at spark://node1:7077

URL: spark://node1:7077
 Alive Workers: 1
 Cores in use: 1 Total, 0 Used
 Memory in use: 2.0 GB Total, 0.0 B Used
 Applications: 0 Running, 0 Completed
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20201225105913-172.18.174.45-35163	172.18.174.45:35163	ALIVE	1 (0 Used)	2.0 GB (0.0 B Used)

← → ↻ 不安全 | node1:18080



History Server

Event log directory: hdfs://node1:9000/history

Last updated: 2020-12-25 11:06:14

Client local time zone: Asia/Shanghai

Show 20 entries

Search:

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
local-1608441273433	DataProcess	2020-12-20 13:14:32	2020-12-20 15:57:09	2.7 h	root	2020-12-20 15:57:09	Download
local-1608442652386	DataRead	2020-12-20 13:37:32	2020-12-20 15:56:41	2.3 h	root	2020-12-20 15:56:41	Download
local-1608450185220	miniProject	2020-12-20 15:43:04	2020-12-20 15:52:08	9.1 min	root	2020-12-20 15:52:08	Download
local-1608449165285	miniProject	2020-12-20 15:26:04	2020-12-20 15:33:19	7.2 min	root	2020-12-20 15:33:19	Download
local-1608442575482	miniProject	2020-12-20 13:36:14	2020-12-20 13:36:27	12 s	root	2020-12-20 13:36:27	Download
local-1608441215340	DataProcess	2020-12-20 13:13:34	2020-12-20 13:14:24	50 s	root	2020-12-20 13:14:24	Download

搭建Jupyter Notebook服务

第4题涉及到很多数据转换和机器学习代码的运行，我比较喜欢用python的Notebook来实现，我打算在运行spark的节点上部署Jupyter Notebook，并集成pyspark环境，这样就可以直接在我的浏览器或者vscode上实时编辑和运行pyspark程序了。

安装Anaconda

Anaconda集成了很多库，其中也有Jupyter，比较方便。依次输入以下命令即可

```
#安装bzip2
yum install -y bzip2
#根目录下创建一个文件夹用于存放Anaconda安装包,并进入文件夹
mkdir anaconda && cd anaconda
#使用wget下载Anaconda安装包
wget https://repo.continuum.io/archive/Anaconda3-4.4.0-Linux-x86_64.sh
#运行安装程序,开始安装Anaconda
./Anaconda3-4.4.0-Linux-x86_64.sh
```

安装成功后使用以下命令生成Jupyter Notebook配置文件

```
jupyter notebook --allow-root --generate-config
```

然后就会有相应的配置文件了

```
[root@node1 ~]# ls .jupyter/
jupyter_notebook_config.py  migrated
```

编辑jupyter_notebook_config.py, 设置all_root为True, 设置允许访问的IP, 启动时不打开浏览器, 监听的端口号

```
c.NotebookApp.allow_root=True
c.NotebookApp.ip='*'
c.NotebookApp.open_browser=False
c.NotebookApp.port=8888
```

配置pyspark的python driver

编辑环境变量, 在启动pyspark的时候以jupyter notebook作为python的driver

```
export PYSARK_DRIVER_PYTHON=jupyter-notebook
export PYSARK_DRIVER_PYTHON_OPTS=" --ip=0.0.0.0 --port=8888"
```

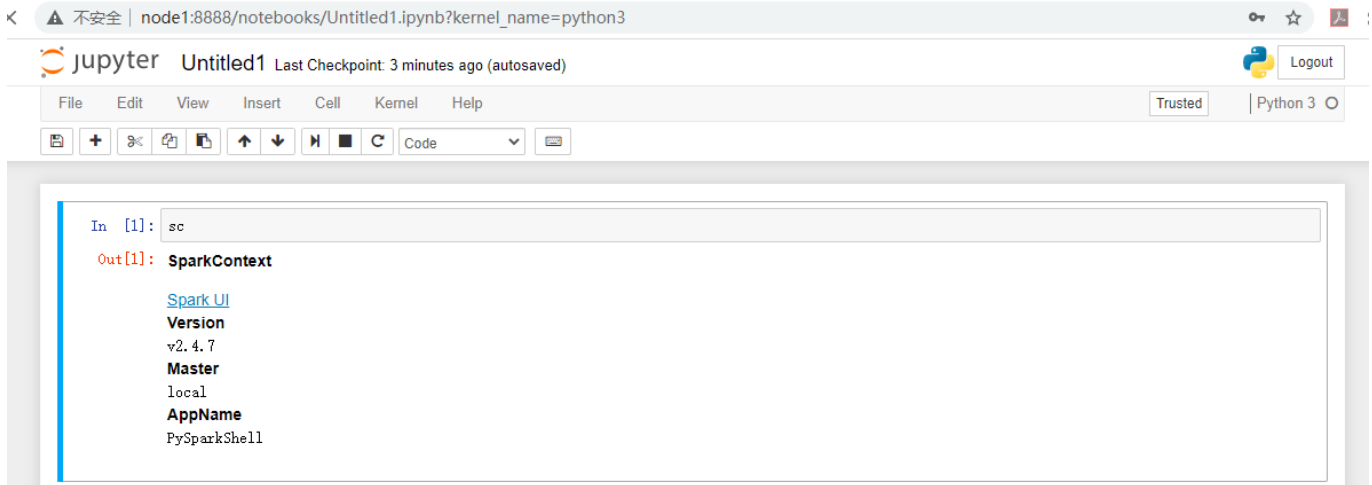
启动pyspark

启动pyspark时自动启动Jupyter Notebook, 显示服务的端口和token

```
[root@node1 ~]# pyspark --master local
[I 11:47:19.834 NotebookApp] Writing notebook server cookie secret to /run/user/0/jupyter/notebook_cookie_secret
[I 11:47:19.825 NotebookApp] Serving notebooks from local directory: /root
[I 11:47:19.825 NotebookApp] 0 active kernels
[I 11:47:19.825 NotebookApp] The Jupyter Notebook is running at: http://0.0.0.0:8888/?token=524b5891cd8b83f4cc3055bd388fb1614bdf6e648b
[I 11:47:19.825 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://0.0.0.0:8888/?token=524b5891cd8b83f4cc3055bd388fb1614bdf6e648b6a0a3b
```

使用给定的token访问主机的上述端口，就可以使用Jupyter Notebook编辑pyspark程序了



Hive

下载hive，然后解压文件

```
[root@node1 ~]# ls
apache-hive-2.3.6-bin.tar.gz
```

编辑环境变量

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
```

然后使用以下命令初始化数据库

```
schematool -dbType <db type> -initSchema
```

如果使用的是Hive默认的数据库，<db type>选项指定为derby即可。

当前目录下会产生数据库的元数据文件和log文件

```
metastore_db
```

```
derby.log
```

注意，任何情况下运行Hive时，需要在这两个文件夹所在的目录下启动Hive，不然会找不到数据库文件。

启动HDFS和yarn

```
[root@node1 hive]# jps
29954 SecondaryNameNode
31123 NodeManager
31015 ResourceManager
29783 DataNode
31433 Jps
29645 NameNode
```

然后启动hive即可

```
[root@node1 hive]# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-2.
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Co
  using Hive 1.X releases.
hive> show tables;
OK
invites
pokes
user_info
user_info_test
user_log
user_log_test
Time taken: 4.055 seconds, Fetched: 6 row(s)
hive> █
```