大数据系统基础大作业

一、数据源

数据流格式

给定 2 路 TCP 数据流，格式为**< topic, timestamp, int_value >**

- **topic**: 流标识，在整个流过程中不变。
- **timestamp**: 时间戳，递增，但可能与当前时间不一致。
- **int_value**: 整数型数据(小于等于 2^{10})

数据流生成

数据流生成原理参照助教发放的demo，但是进行了一些改动来配合整个分布式程序

起始时间戳timestamp采用当前时间加上一个随机偏移量（不超过1天）生成，数据流时间戳递增

数据流速度为[0.1,level*0.4]秒/条，level为1到4的随机整数

value为0到2^{10}中的随机整数

核心代码：

```
random.seed(time.time())
offset = random.randint(0, 86400)   # initial time difference (less than 1
day)
speed_level = random.randint(1, 4)   # stream speed level
topic = 'topic ' + str(1) + ' '
t = time.time() + offset
x = random.randint(0, 2**10)    # int_val range [0,1024]
data = topic + str(t) + " " + str(x)
```

二、流处理

实验要求

实现接收 2 路 TCP 数据流的处理程序(processor)，并输出递增文件(或者数据流)

格式为:**< topic1, timestamp1, topic2, timestamp2, int_value >**

topic1:数据流 1 的标识

timestamp1:数据流 1 的数据元组的时间戳

topic2:数据流 2 的标识

timestamp2:数据流 2 的数据元组的时间戳

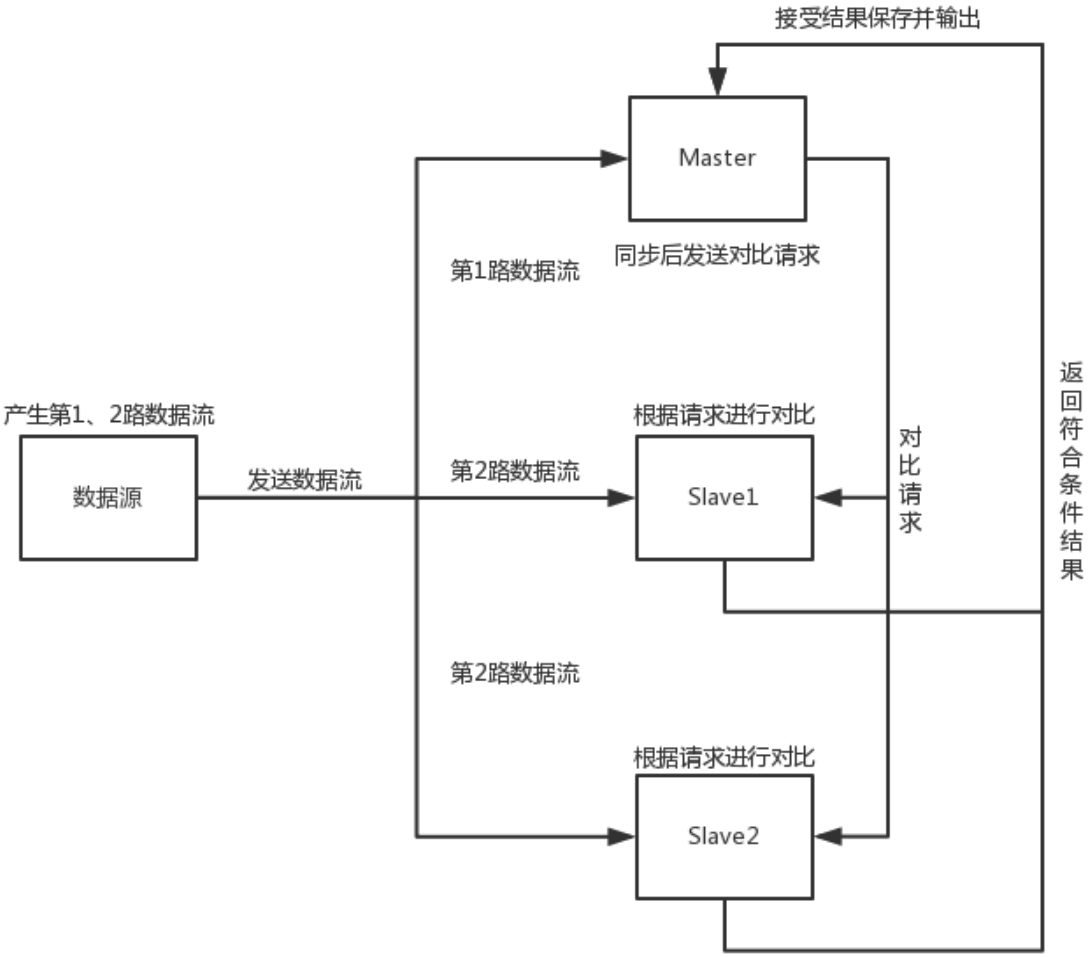
int_value:当数据流 1 与数据流 2 中时间戳 timestamp1 和 timestamp2 距离不超过 30 秒，且两者的 int_value相同时输出至递增文件

实验设计和实现

对于本次实验，采取将处理程序processor分布到多台服务器上的方案

数据分布式处理流程

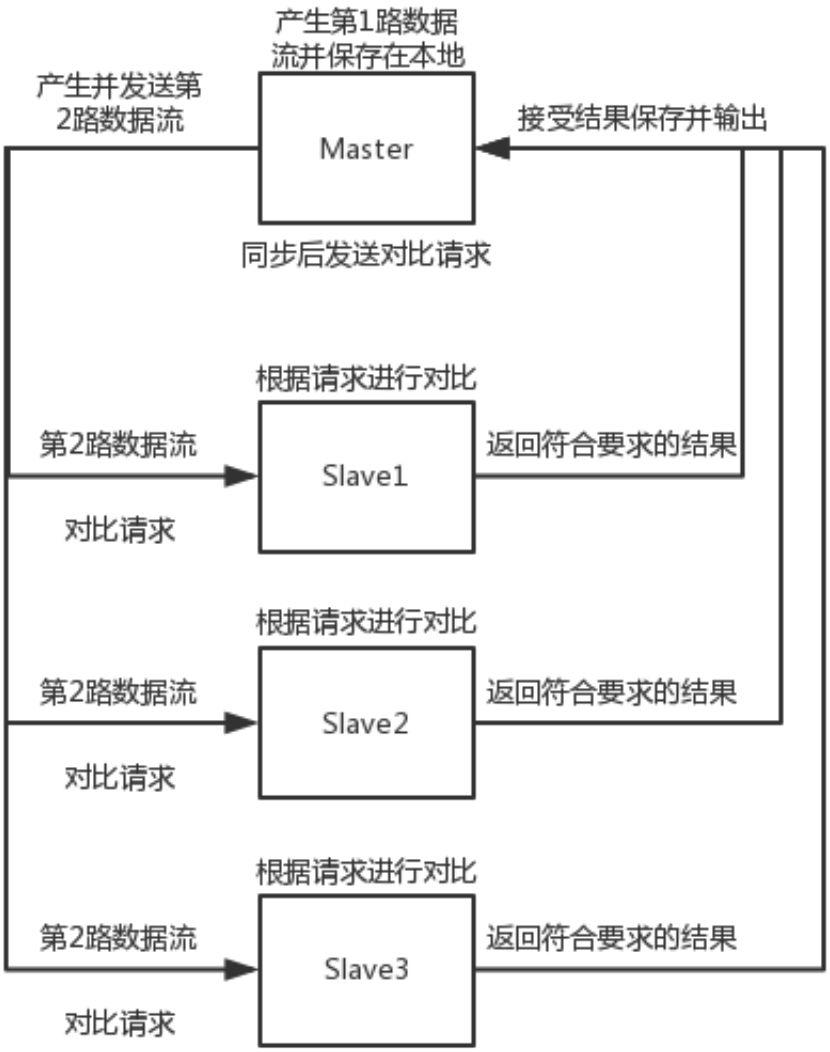
原先打算的处理流程:



由于本次实验我只打算使用4台服务器，所以单独拿出一台服务器当数据源很浪费性能

经过思考后作出的改变:

将数据源和Master整合到一台服务器上，这样其余三台都作为Slave，提高了程序的运行效率，而且在实现上都差不多。



分布式架构

根据需要将整个处理程序分为两个部分：

Master（数据源+任务调度）：

1台服务器

线程1: 产生第1路数据流保存在本地

线程2: 产生并发送第2路数据流给所有Slave

线程3: 将第1路数据流按照一定规则均分后，分别向所有Slave发送该条数据在第二路数据流中的对比请求

线程4: 接收从所有Slave传回的处理结果，把结果保存到递增文件并输出

Slave（数据对比）：

多台服务器（本次实验中为3台）

线程1: 从Master接收第2路数据流

线程2:接受Master的处理请求，以请求中第一路数据流的时间戳前后30秒为范围，在第2路数据流中对比查找符合要求的数据，并传回结果给Master。

服务器间通信

使用python的socket来传送数据，每条数据使用python的struct对数据进行打包后传送

包格式定义:

其中i表示整形存放sign，128s表示128字符长度存放数据data

```
fileinfo_size = struct.calcsize('i128s')
```

封包:

```
message = struct.pack('i128s', sign, data)
```

拆包:

```
from_recv = client_sock.recv(fileinfo_size)
sign, data  = struct.unpack('i128s', from_recv)
```

信号sign含义:

对于Master收到的数据：由于自身产生数据流，所以只接收Slave的结果，sign信号无所谓

对于Slave收到的数据：sign分为1 (数据流)，2 (对比请求)

发送数据函数

使用socket可靠地发送数据，与前几次实验相同

```
# 发送数据函数
def sendmessage(IP, data):
    sk_slave = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sk_slave.settimeout(10)
    sk_slave.connect((IP, 6666))
    message = struct.pack('i128s', 1, data)
    sk_slave.send(message)
    sk_slave.close()
```

数据对比请求分配

设置一个下标i，从已接收的第一路数据流中递增地取数据，根据i除以3的余数来决定分配给哪个Slave

当余数为0时，此条请求发送给Slave1

当余数为1时，此条请求发送给Slave2

当余数为2时，此条请求发送给Slave3

```
tmp = datastream[i]
if i%3 == 0:
    sendmessage(slave1_IP, tmp)
    i += 1
elif i%3 == 1:
    sendmessage(slave2_IP, tmp)
    i += 1
else:
    sendmessage(slave3_IP, tmp)
    i += 1
```

数据处理同步问题

因为两路数据时间戳有偏移，如果不考虑时间戳同步问题，有可能会出现需要处理的第1路数据的时间戳远超出了已接收的所有第2路数据流的时间戳，这时需要对比的符合时间范围的第2路数据流还没接收到，会导致数据对比不完整，可能会漏掉符合条件的数据。

为了避免此问题发生，在Master发出数据对比请求之前，需要检查当前Slave接收的最新的第2路数据时间戳是否大于等于当前需要对比的第1路数据时间戳加上30秒，如果满足则发送请求，否则等待满足条件后发送请求。

具体做法为：

首先获得第二路数据流时间戳的偏移值**offset_2**

在每次发送对比请求的时候，都可以根据当前时间加偏移值知道第2路数据流当前的时间为**time.time()+offset_2**

再判断这条对比请求中的第一路数据时间戳加上30秒**timestamp_1+30**是否小于等于第2路数据流当前时间戳**time.time()+offset_2**，满足则发送，否则等待直至满足再发送

```
#同步等待后分配数据处理任务
def assignment():
    i = 0
    while True:
        try:
            #从第一路数据流中取出一条数据
            tmp = datastream[i]
            #根据差来判断是否发送或等待
            diff = float(tmp.split()[2]) - time.time() - offset_2 + 30
            if(diff <= 0):
                #设置的发送速度
                time.sleep(0.1)
                if i%3 == 0:
                    sendmessage(slave1_IP, tmp)
                    i += 1
                elif i%3 == 1:
```

```
                    sendmessage(slave2_IP, tmp)
                    i += 1
                else:
                    sendmessage(slave3_IP, tmp)
                    i += 1
            else:
                time.sleep(diff)
        except IndexError:
            pass
```

Slave中的数据对比

1. Slave在接受到数据对比请求后，取得了第1路数据的时间戳**timestamp_1**和**value_1**值
2. 遍历已接收的第2路数据流的每条数据，取出其时间戳**timestamp_2**和**value_2**值
3. 计算其时间戳的差值**diff**
4. 如果差值小于-30，那么中断遍历，因为时间戳是递增的，往后数据的时间戳都不满足对比的范围
5. 如果差值在-30到30之间，则继续对比value值，如果相等则满足条件，按照要求的格式回传给Master

```
    tmp  = data.split()
    timestamp_1 = tmp[2]
    value_1 = tmp[3]
    for each in datastream:
        tmp = each.split()
        timestamp_2 = tmp[2]
        value_2 = tmp[3]

        time_diff = float(timestamp_1)- float(timestamp_2)

        #第1路数据比第2路数据时间戳小超过30秒，那么后面的没有继续比较的意义
        if time_diff < -30:
            break
        if ((time_diff <= 30)  and (value_1 == value_2)):
            data = "topic1 "+timestamp_1+" topic2 "+timestamp_2+" "+value_1
            #结果发回Master
            sendmessage(data,1)
            print "回传结果"
```

实验结果

由于程序随机生成的时间戳偏移值不确定，为了避免等待过长的同步时间，在运行代码的时候人为设定第1路数据流的时间戳偏移为0，第2路数据流时间戳偏移为33

将Master.py，Slave1.py，Slave2.py，Slave3.py分别部署到thumm01，thumm02，thumm03，thumm04上，先运行Slave1.py，Slave2.py，Slave3.py，最后运行Master.py，程序开始运行

Master显示自己产生的第1路数据流并保存到datastream1.txt中

```
-bash-4.1$ python Master.py
topic 1 1515486594.39 349
topic 1 1515486594.59 168
topic 1 1515486594.96 818
topic 1 1515486595.19 207
topic 1 1515486595.39 932
topic 1 1515486595.65 492
topic 1 1515486595.82 335
topic 1 1515486595.94 1000
topic 1 1515486596.07 785
topic 1 1515486596.4 938
topic 1 1515486596.79 488
topic 1 1515486596.98 423
topic 1 1515486597.23 87
topic 1 1515486597.54 254
topic 1 1515486597.73 28
topic 1 1515486597.97 463
topic 1 1515486598.14 666
topic 1 1515486598.24 125
topic 1 1515486598.56 556
topic 1 1515486598.95 864
topic 1 1515486599.07 652
topic 1 1515486599.17 525
topic 1 1515486599.59 750
topic 1 1515486599.71 517
```

Slave1、2、3同时显示接收到的第2路数据流，其中Slave1会将第2路数据流保存到datastream2.txt中

```
[2017210879@thumm02 ~]$ python slave1.py
Listening on thumm02:6666
topic 2 1515486627.4 859
topic 2 1515486627.62 895
topic 2 1515486628.55 776
topic 2 1515486629.79 715
topic 2 1515486631.28 331
topic 2 1515486632.32 848
topic 2 1515486633.23 873
topic 2 1515486633.84 857
topic 2 1515486634.82 482
topic 2 1515486636.1 180
topic 2 1515486636.33 641
topic 2 1515486637.52 677
topic 2 1515486637.97 305
topic 2 1515486638.33 888
topic 2 1515486639.79 76
topic 2 1515486641.2 149
topic 2 1515486642.53 646
topic 2 1515486642.75 319
topic 2 1515486642.98 715
topic 2 1515486643.18 643
topic 2 1515486643.86 743
topic 2 1515486645.06 320
topic 2 1515486646.2 734
```

```
[2017210879@thumm03 ~]$ python slave2.py
Listening on thumm03:6666
topic 2 1515486627.4 859
topic 2 1515486627.62 895
topic 2 1515486628.55 776
topic 2 1515486629.79 715
topic 2 1515486631.28 331
topic 2 1515486632.32 848
topic 2 1515486633.23 873
topic 2 1515486633.84 857
topic 2 1515486634.82 482
topic 2 1515486636.1 180
topic 2 1515486636.33 641
topic 2 1515486637.52 677
topic 2 1515486637.97 305
topic 2 1515486638.33 888
topic 2 1515486639.79 76
topic 2 1515486641.2 149
topic 2 1515486642.53 646
topic 2 1515486642.75 319
topic 2 1515486642.98 715
topic 2 1515486643.18 643
topic 2 1515486643.86 743
topic 2 1515486645.06 320
topic 2 1515486646.2 734
```

```
[2017210879@thumm04 ~]$ python slave3.py
Listening on thumm04:6666
topic 2 1515486627.4 859
topic 2 1515486627.62 895
topic 2 1515486628.55 776
topic 2 1515486629.79 715
topic 2 1515486631.28 331
topic 2 1515486632.32 848
topic 2 1515486633.23 873
topic 2 1515486633.84 857
topic 2 1515486634.82 482
topic 2 1515486636.1 180
topic 2 1515486636.33 641
topic 2 1515486637.52 677
topic 2 1515486637.97 305
topic 2 1515486638.33 888
topic 2 1515486639.79 76
topic 2 1515486641.2 149
topic 2 1515486642.53 646
topic 2 1515486642.75 319
topic 2 1515486642.98 715
topic 2 1515486643.18 643
topic 2 1515486643.86 743
topic 2 1515486645.06 320
topic 2 1515486646.2 734
```

Mater会在等待同步后向Slave发送数据对比请求，此时没有显示

Slave接收到请求后开始对比，如果对比到符合条件的数据会显示回传结果（如图中右侧红线框所示）

Master接收到回传的结果会显示出来并且保存到result.txt中（如图中左侧红线框所示）

```
topic 1 1515486632.97 340
topic 1 1515486633.22 628
topic 1 1515486633.52 271
topic 1 1515486633.74 408
topic 1 1515486634.22 926
topic 1 1515486634.55 188
topic 1 1515486634.91 386
topic 1 1515486635.01 392
topic 1 1515486635.11 610
topic 1 1515486635.46 781
topic 1 1515486635.61 978
topic 1 1515486635.83 208
topic 1 1515486636.38 181
topic 1 1515486636.53 196
topic1 1515486636.53 topic2 1515486652.05 196
topic 1 1515486636.86 324
topic 1 1515486637.18 129
topic 1 1515486637.55 675
topic 1 1515486637.83 215
topic 1 1515486638.04 555
topic 1 1515486638.21 72
topic 1 1515486638.57 366
topic 1 1515486638.88 668
topic 1 1515486639.0 129
topic 1 1515486639.11 339
```

```
topic 2 1515486659.25 106
topic 2 1515486660.34 650
topic 2 1515486660.73 328
topic 2 1515486661.59 993
topic 2 1515486663.04 782
topic 2 1515486663.5 751
topic 2 1515486664.63 376
topic 2 1515486665.65 284
topic 2 1515486666.34 219
topic 2 1515486667.12 865
topic 2 1515486667.75 341
topic 2 1515486668.51 663
topic 2 1515486669.53 215
回传结果
发送topic1 1515486636.53 topic2 1515486652.05 196给thumm01完毕
topic 2 1515486670.07 952
topic 2 1515486671.05 601
topic 2 1515486672.12 784
topic 2 1515486673.51 738
topic 2 1515486674.19 705
topic 2 1515486674.67 538
topic 2 1515486675.84 714
topic 2 1515486676.15 126
topic 2 1515486677.52 768
topic 2 1515486679.06 762
```

运行五分钟后用Ctrl+C依次中断Master和其余三个Slave，查看result.txt

```
-bash-4.1$ more result.txt
topic1 1515486621.43 topic2 1515486649.06 956
topic1 1515486636.53 topic2 1515486652.05 196
topic1 1515486646.04 topic2 1515486657.67 322
topic1 1515486654.1 topic2 1515486654.93 33
topic1 1515486656.9 topic2 1515486686.04 466
topic1 1515486658.38 topic2 1515486663.5 751
topic1 1515486665.9 topic2 1515486694.72 179
topic1 1515486666.69 topic2 1515486684.29 557
topic1 1515486667.52 topic2 1515486688.61 336
topic1 1515486668.88 topic2 1515486671.05 601
topic1 1515486675.42 topic2 1515486672.12 784
topic1 1515486681.67 topic2 1515486677.52 768
topic1 1515486685.5 topic2 1515486715.29 520
topic1 1515486685.74 topic2 1515486670.07 952
topic1 1515486686.14 topic2 1515486703.92 744
topic1 1515486687.01 topic2 1515486674.19 705
topic1 1515486687.25 topic2 1515486713.37 687
topic1 1515486696.18 topic2 1515486674.67 538
topic1 1515486697.03 topic2 1515486670.07 952
topic1 1515486697.81 topic2 1515486708.02 39
topic1 1515486704.79 topic2 1515486694.72 179
topic1 1515486706.17 topic2 1515486718.52 542
topic1 1515486711.21 topic2 1515486723.94 87
topic1 1515486712.01 topic2 1515486700.02 53
```

本次实验结果datastream1.txt，datastream2.txt和result.txt见附件

三、batch数据处理

数据文件

两个数据流每个流每小时形成1个 batch，共生成六个小时一共12个batch

文件命名为topic1_1, topic1_2 ….topic1_6和topic2_1, topic2_2 ….topic2_6

每行格式：**< topic, timestamp, int_value >**

数据生成使用2个线程分别生成第1路和第2路数据流

生成batch函数：

```
def genBatch(stream_index):
    random.seed(time.time())
    offset = random.randint(0, 86400)  # initial time difference (less than 1
day)
    speed_level = random.randint(1, 4)  # stream speed level

    start = time.time()
    topic = 'topic ' + str(stream_index) + ' '

    f = open("topic"+str(stream_index)+"_1",'w')

    i = 1
    while i < 7:
        while time.time()-start < 3600:
            t = time.time() + offset
            x = random.randint(0, 2 ** 10)  # int_val range [0,1024]
            data = topic + str(t) + " " + str(x) + "\n"
            f.write(data)
            time.sleep(
                max(0.1, random.randint(0, speed_level * 400) / 1000.0))  #
0.1 <= stream interval <= level * 0.4 (seconds)
        start = time.time()
        i += 1
        f.close()
        if i<7:
            f = open("topic"+str(stream_index)+"_"+str(i),'w')
```

实验设计与实现

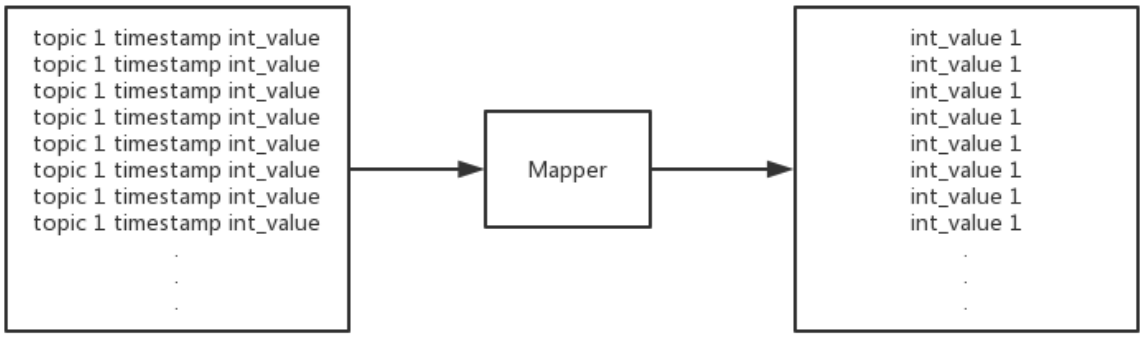
本次实验利用HDFS和Hadoop的MapReduce框架来实现，只需要设计实现map和reduce函数即可

Mapper

Mapper的工作就是提取原始数据的关键信息，本实验只要求输出在两个流中共同出现的所有int_value，而不关心时间戳，所以关键信息只有topic号和int_value

Mapper的作用具体为：

< topic, timestamp, int_value > ⇒ < int_value, topic >



具体实现如下：

```
import sys
def read_input(file):
    for line in file:
        yield line.split()

def main(separator=' '):
    data = read_input(sys.stdin)
    for words in data:
       print "%s%s%s" % (words[3], separator, words[1])

if __name__ == "__main__":
    main()
```

Reducer

Reducer工作流程：

1. 接收Mapper产生的已经按照int_value排好序的中间结果
2. 由于本实验要求输出在两路数据流中**共同**出现的int_value，所以对于同一路数据流中重复的int_value可以去除
3. 去重后如果相邻两行的int_value相同而topic不同，则输出int_value

具体实现如下：

```
import sys
if __name__ == "__main__":
    current_line = None

    for line in sys.stdin:
        line = line.rstrip()

        #不重复继续判断，重复则跳过，相当于去重
        if line != current_line:
            if current_line:
                current_value, current_topic = current_line.split()
                value, topic = line.split()
```

```
                if (current_topic != topic) and (current_value == value):
                    print value

            current_line = line
```

实验结果

先将生成的batch存到HDFS上

```
hadoop fs -put topic* input
```

输入命令调用MapReduce框架

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.5.2.jar   \
-file ./mapper.py -mapper ./mapper.py   \
-file ./reducer.py -reducer ./reducer.py   \
-input input/*   \
-output output
```

从HDFS取回运行结果part-00000，并查看内容

```
hadoop fs -get output/part-00000
more part-00000
```

```
-bash-4.1$ more part-00000
0
1
10
100
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
101
1010
1011
1012
1013
1014
1015
1016
1017
1018
```

因为不关心数据时间戳，而且int_value范围只有[0,1024]的整数，所以6个小时的两路数据流中极大概率都会生成0到1024的所有int_value，程序的结果就包括0到1024的所有整数。