Cloud Computing Capstone Task II

Baoshi Sun, February 17, 2016

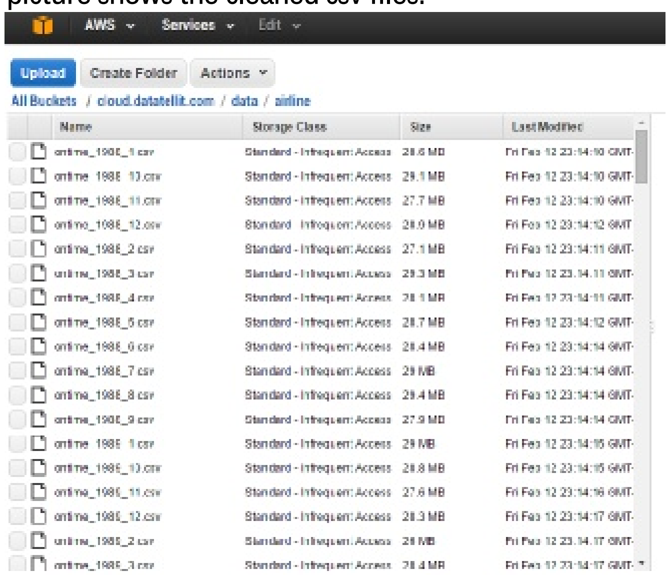
Experiment Environment

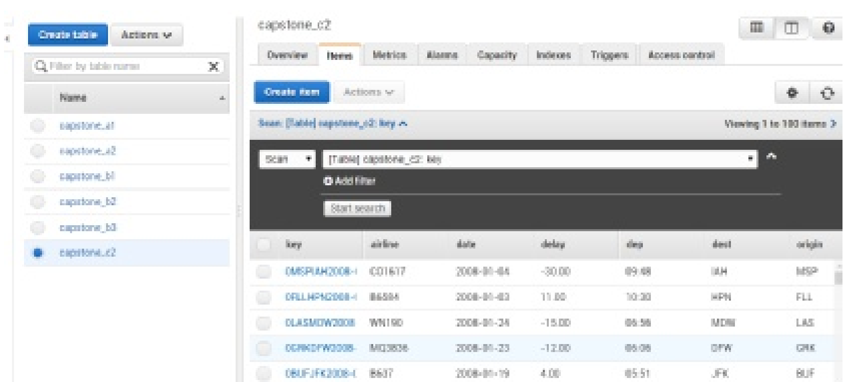
Instead of using Hadoop & Cassandra on AWS EC2 IaaS like what I did in task I, for this task I turned to EMR and DynamoDB.

- AWS Resources
  - 3 m3.xlarge EC instances compose an EMR luster
  - Volume: total 80G
  - S3 Storage
  - DynamoDB
- 3rd Party Components
  - Kafka 0.9.0.x
- Development Tools
  - Python 2.7 + pyspark
  - Boto 3

Data L oading & Cleaning

The methods are almost the same as that in task I, expect that a few Hadoop file operation commands are replaced by AWS S3 CLI. The picture shows the cleaned csv files.

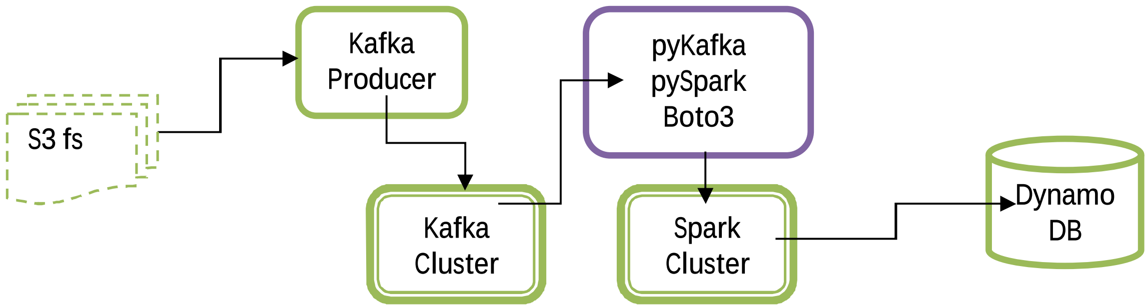




Methodology

Although I tried S3 direct file streaming and Kafka consumer streaming approach, eventually I decided to use Kafka direct streaming, and it works very well.

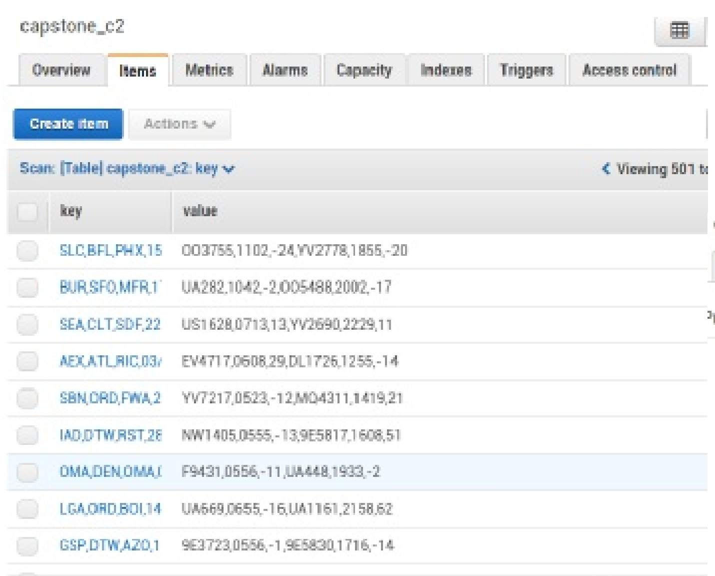
Results of all calculations are stored into DynamoDB. As can be seen in above picture, one table corresponds to one question.

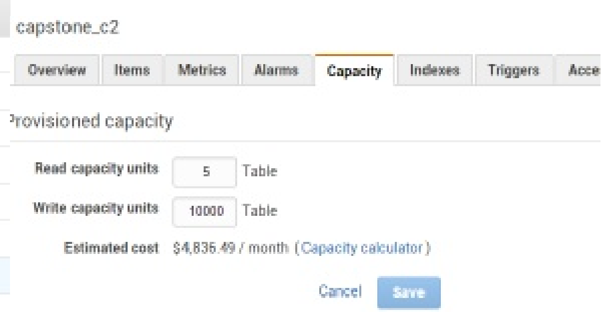


One single spark streaming program is deployed, which processes all 6 tasks (Q1.1, Q1.2, Q2.1, Q.2.2, Q2.3 and Q3.2) in a row, so that the data stream needs to be played only once.

I also wrote a small data feeding tool to generate stream to data consumers. The tool now supports reading data from S3, HDFS and local files.

As to the outputs, at first I wanted to write everything directly to DynamoDB via Boto interface. However, it turns out that the throughput of DynamoDB is a big problem, even if batch writing is applied. I have to increase the write units, especially when write data of question 3.2, which cost an arm and a leg.





To balance the stream feeding speed and the program processing efficiency, a 10-second sleep is inserted between two feedings of data file. On the other hand, after test I set the streaming query interval to 6 seconds and check points is updated at every 60 seconds.

In addition, considering the program should execute for a long time but it can't stop even if there is no more data coming in, I added a simple self-check mechanism. If the number of total processed records stay unchanged for two minutes, the program will save necessary data and make a graceful exit.

Check the source code here:

- [Main spark streaming program]
- [Data feeding scripts]

Results Report (Video link)

Question 1.1 & Question 1.2

```
16/02/16 04:32:01 INFO DAGScheduler: ResultStage 12775 (runJob
16/02/16 04:32:01 INFO DAGScheduler: Job 1746 finished: runJob
Total: 233503980
ORD: 12449288
ATL: 11539676
DFW: 10799262
LAX: 7723452
PHX: 6585495
DEN: 6273780
DTW: 5636591
IAH: 5480672
MSP: 5199211
SFO: 5171014
16/02/16 04:32:01 INFO JobScheduler: Finished job streaming job
16/02/16 04:32:01 INFO JobScheduler: Total delay: 81.738 s for
```

```
16/02/16 13:08:51 INFO
16/02/16 13:08:51 INFO
HA: -1.01
AQ: 1.14
PS: 1.44
ML: 4.65
PA: 5.24
NW: 5.43
F9: 5.43
WN: 5.50
OO: 5.61
9E: 5.69
16/02/16 13:08:51 INFO
```

Question 2.1

```
Result to question b1 on SRQ:
{u'value': u'TZ(-0.38%),RU(-0.09%),YV( 3.34%),AA( 3.61%),UA( 3.91%),US( 3.94%),TW( 4.27%),NW( 4.81%),DL( 4.81%),XE( 4.97%)
Result to question b1 on CMH:
{u'value': u'DH( 3.39%),AA( 3.47%),NW( 3.95%),ML( 4.30%),DL( 4.66%),PI( 5.14%),EA( 5.78%),US( 5.86%),RU( 5.96%),AL( 5.98%)
Result to question b1 on JFK:
{u'value': u'RU( 4.91%),UA( 5.85%),CO( 8.08%),DH( 8.32%),AA( 9.91%),B6(10.99%),NW(11.20%),PA(11.42%),DL(11.77%),MQ(12.22%)
Result to question b1 on SEA:
{u'value': u'OO( 2.65%),PS( 4.70%),YV( 4.97%),AL( 6.00%),TZ( 6.31%),US( 6.36%),NW( 6.41%),DL( 6.47%),HA( 6.85%),AA( 6.86%)
Result to question b1 on BOS:
{u'value': u'RU( 2.07%),TZ( 3.02%),PA( 4.37%),ML( 5.63%),EV( 6.83%),NW( 7.03%),DL( 7.23%),US( 8.33%),AA( 8.43%),AL( 8.44%)
```

Question 2.2

```
Result to question b2 on SRQ:
{u'value': u'EYW( 0.00%),SJU( 0.00%),TPA( 1.31%),IAH( 1.43%),MEM( 1.69%),FLL( 2.00%),BNA( 2.06%),MCO( 2.34%),RDU( 2.52%),MDW( 2.82%)
Result to question b2 on CMH:
{u'value': u'SYR(-5.00%),AUS(-5.00%),OMA(-5.00%),MSN( 1.00%),CLE( 1.09%),SDF( 1.35%),CAK( 3.69%),SLC( 3.93%),IAD( 4.02%),MEM( 4.07%)
Result to question b2 on JFK:
{u'value': u'SWF(-10.50%),ISP( 0.00%),ABQ( 0.00%),ANC( 0.00%),MYR( 0.00%),UCA( 1.89%),BGR( 3.18%),BQN( 3.57%),CHS( 4.24%),STT( 4.42%
Result to question b2 on SEA:
{u'value': u'EUG( 0.00%),PIH( 1.00%),PSC( 2.61%),CVG( 3.84%),MEM( 4.21%),BLI( 5.02%),CLE( 5.15%),YKM( 5.23%),SNA( 5.31%),LIH( 5.48%)
Result to question b2 on BOS:
{u'value': u'SWF(-5.00%),ONT(-3.00%),GGG( 1.00%),AUS( 1.20%),LGA( 2.92%),MSY( 3.14%),LGB( 5.12%),OAK( 5.75%),MDW( 5.80%),BDL( 5.89%)
ubuntu@ip-172-31-23-39:~/work$
```

Question 2.3

```
Result to question b3 on LGA,BOS:
{u'value': u'TW(-3.00%),US(-2.76%),PA(-0.41%),DL( 1.67%),EA( 4.69%),MQ( 9.25%),NW(13.82%),OH(24.96%),AA(28.50%),', u'key': u'LGA,BOS'}
Result to question b3 on BOS,LGA:
{u'value': u'TW(-11.00%),US( 1.04%),DL( 1.93%),PA( 5.95%),EA( 9.21%),MQ(11.93%),NW(14.48%),OH(24.53%),AA(28.00%),TZ(133.00%),', u'key': u'BOS,LGA'
Result to question b3 on OKC,DFW:
{u'value': u'TW( 0.10%),EV( 1.33%),MQ( 4.47%),AA( 4.50%),DL( 6.67%),OO(12.64%),OH(47.50%),', u'key': u'OKC,DFW'}
Result to question b3 on MSP,ATL:
{u'value': u'9E( 0.00%),EA( 4.08%),OO( 4.70%),DL( 6.24%),FL( 6.24%),NW( 6.88%),OH( 8.14%),EV( 9.76%),', u'key': u'MSP,ATL'}
```

Question 3.2

```
Result to question 3.2:
BOS,ATL,LAX,03/04/2008  FL270,0548,7,FL40,1857,-2
PHX,JFK,MSP,07/09/2008  B6178,1127,-25,NW609,1747,-17
DFW,STL,ORD,24/01/2008  AA1336,0657,-14,AA2245,1654,-5
LAX,MIA,LAX,16/05/2008  AA280,0817,10,AA456,1925,-19
ubuntu@ip-172-31-23-39:~/data$
```

-- Query: BOS,ATL,LAX,03/04/2008 -- Result: FL270,0548,7,FL40,1857,-2

The result indicates the route is taking FL270 which departures at 05:48 on 03/04/2008 from BOS to ATL with 7 minutes delay, and taking FL40 at 18:57 on 05/04/2008 from ATL to LAX with 2 minutes earlier than the schedule.

Conclusions

1. By compared the results between task I and task II, I found there are slight differences. One possible reason is the algorithms are not identical. For example, in task II the 'cancel' flights are totally ignored, while they were counted as 'delay' in task I. However, I suspect some messages might be lost in streaming mode. If time allows, I'd like to investigate in detail.
2. What I learned from the project regarding the differences between MR and streaming include:
    a. Streaming mode can perform many independent tasks upon one stream almost in one program space. But for MR, we need to launch different processes.
    b. Spark requires much more server resources, especially memory, than MR. With the equivalent settings, Task I ran smoothly. However, during Task II I encountered numerous 'insufficient memory' errors and had to reboot the cluster again and again.
    c. When dealing with streaming, it seems that more considerations should be put on optimization perspective.
    d. Although EMR is easy to use, it is expensive and limited in many aspects. I would suggest my company to build its cloud platform from scratch (native Hadoop and Spark) on EC2, rather than use EMR.