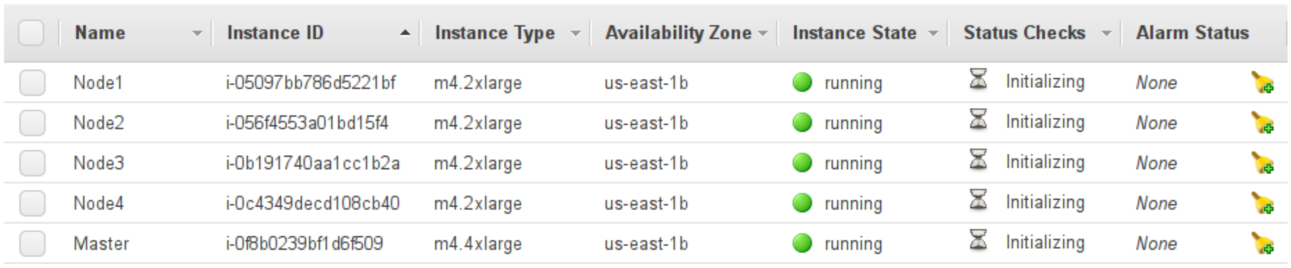
Cloud Computing Capstone Project - Part 1

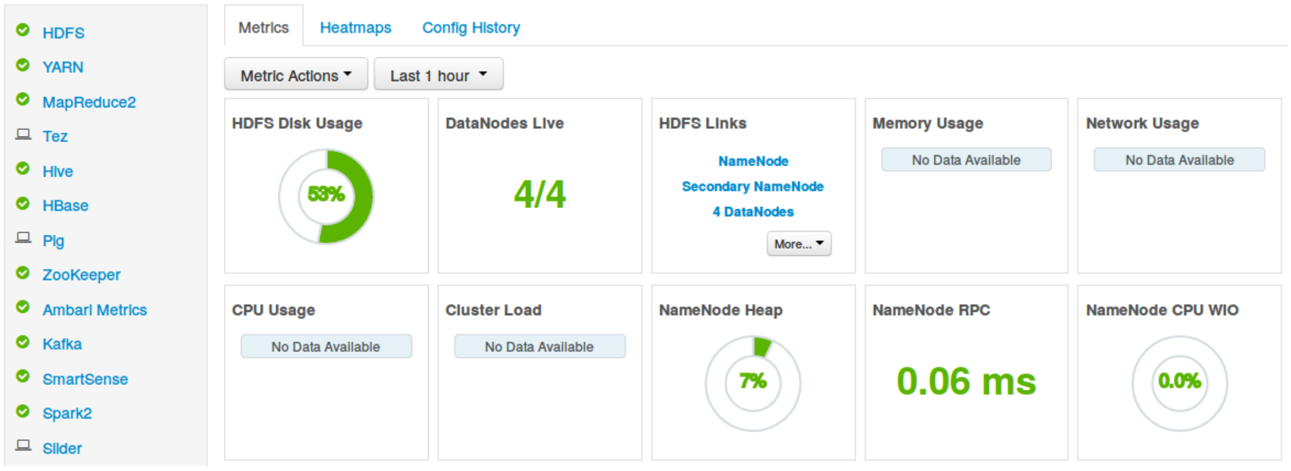
System integration

To perform our analysis of the transportation dataset we deployed a 4 node cluster on Amazon EC2 using general purpose instances (m4.2xlarge) with 50GB of EBS storage and an additional node (m4.4xlarge) to manage the cluster.



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
Node1	i-05097bb786d5221bf	m4.2xlarge	us-east-1b	● running	Initializing	None
Node2	i-056f4553a01bd15f4	m4.2xlarge	us-east-1b	● running	Initializing	None
Node3	i-0b191740aa1cc1b2a	m4.2xlarge	us-east-1b	● running	Initializing	None
Node4	i-0c4349decd108cb40	m4.2xlarge	us-east-1b	● running	Initializing	None
Master	i-0f8b0239bf1d6f509	m4.4xlarge	us-east-1b	● running	Initializing	None

We deployed Hortonwork's solution using Apache Ambari with HDFS, Hadoop, HBase, Spark and Kafka. We used Yarn to schedule our jobs and ZooKeeper to integrate our systems.



Extracting and cleaning the data (ETL)

Of all the datasets in the transportation dataset, we used the airline on-time performance data to answer the questions.

We parallelized the ETL process by loading a subset of the data from each node. We processed the data in memory and piped it to HDFS to avoid having to touch the local disks, using standard Linux command line utilities (unzip, tail, sed, cut...).

These are the steps we followed:

- Uncompressed the dataset ZIP files.
- Removed commas from one of the columns.
- Filtered the columns we needed to answer the questions.
- Removed any trailing HTML code.
- Piped the output to HDFS.

For example:

```
$ cat etl.sh
#/!bin/bash
BASE_DIR=/home/ec2-user/airline_ontime
for dir in `ls $BASE_DIR`; do
        echo "> Processing directory $BASE_DIR/$dir..."
                { find "$BASE_DIR/$dir" -name \*.zip -exec unzip -p "{}" \;; } | \
                tail -n+2 | sed -e 's/, / /g' | cut -d"," -f7,12,18,26,37 | tr -d '"' | \
                awk '!/<.*>/ { print $0 }' | hdfs dfs -put - /capstone/airline_ontime/$dir.csv
done
```

Optimization: We only needed 5 columns to answer questions 1.1 to 3.1, but we needed 8 to answer question 3.2, which only required data from 2008. To reduce the ETL time and the size of our data we used a reduced dataset containing data from 1988 to 2008 to answer questions 1.1 to 3.1, and another dataset containing data from 2008 to answer question 3.2.

Questions

Group 1 (Answer any 2):

Q1.1 - Rank the top 10 most popular airports by numbers of flights to/from the airport.

- At each mapper: For each entry in the dataset we read the departure and arrival airports and computed partial sums of arrivals and departures for each airport. We output these partials sums using the airport identifier as the key.
- At a single reducer: We computed the total number of departures and arrivals for each airport, sorted and output the global top 10.

Optimization: To reduce the amount of data transferred to the reducer we moved part of the computation to the mappers by computing partial sums, since a single mapper would read data from many flights belonging to the same airport.

- Result:

```
# hdfs dfs -cat /capstone/q1.1.out/part-00000
1. ORD: 12449354
2. ATL: 11540422
3. DFW: 10799303
4. LAX: 7723596
5. PHX: 6585534
6. DEN: 6273787
7. DTW: 5636622
8. IAH: 5480734
9. MSP: 5199213
10. SFO: 5171023
```

Q1.2 - Rank the top 10 airlines by on-time arrival performance.

- At each mapper: We output partial sums of arrival delays for each airline at each mapper, using the airline identifier as the key.
- At a single reducer: We computed the total arrival delay for each airline, sorted and output the global top 10.
- Result:

```
# hdfs dfs -cat /capstone/q1.2.out/part-00000
1. HA: -1.01180434575
2. AQ: 1.15692344248
3. PS: 1.45063851278
4. ML (1): 4.74760919573
5. PA (1): 5.32243099993
6. F9: 5.46588114882
7. NW: 5.55778339267
8. WN: 5.56077425988
```

```
9. OO: 5.73631246366
10. 9E: 5.8671846617
```

Group 2 (Answer any 3):

Q2.1 - For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.

- At each mapper: We output partial sums of departure delays for each departure airport/airline pair, using the departure airport identifier as the key.
- At each reducer: For each departure airport/airline pair, we computed the total departure delay, sorted and the top 10 were written to the database in batches, indexed by the departure airport identifier.

Result:

```
# ./query.py "CMI" "BWI" "MIA" "LAX" "IAH" "SFO"
CMI
   * (ABI, -7.0)
   * (PIT, 1.10243055556)
   * (CVG, 1.89476168004)
   * (DAY, 3.11623529412)
   * (STL, 3.98167330677)
   * (PIA, 4.59189189189)
   * (DFW, 5.94414274631)
   * (ATL, 6.66513761468)
   * (ORD, 8.19409814324)
BWI
   * (SAV, -7.0)
   * (MLB, 1.15536723164)
   * (DAB, 1.46959459459)
   * (SRQ, 1.58848388801)
   * (IAD, 1.79094076655)
   * (UCA, 3.65416985463)
   * (CHO, 3.74492753623)
   * (GSP, 4.19768664564)
   * (SJU, 4.44465842287)
   * (OAJ, 4.47111111111)
MIA
   * (SHV, 0.0)
   * (BUF, 1.0)
   * (SAN, 1.71038251366)
   * (SLC, 2.53719008264)
   * (HOU, 2.91219912473)
   * (ISP, 3.64739884393)
   * (MEM, 3.74510662248)
   * (PSE, 3.97584541063)
   * (TLH, 4.26148447469)
   * (MCI, 4.61224489796)
LAX
   * (SDF, -16.0)
   * (IDA, -7.0)
   * (DRO, -6.0)
   * (RSW, -3.0)
   * (LAX, -2.0)
   * (BZN, -0.727272727273)
   * (MAF, 0.0)
   * (PIH, 0.0)
   * (IYK, 1.26982474406)
   * (MFE, 1.37647058824)
IAH
   * (MSN, -2.0)
   * (AGS, -0.61879049676)
   * (MLI, -0.5)
   * (EFD, 1.88770821367)
   * (HOU, 2.17203698515)
   * (JAC, 2.57058823529)
   * (MTJ, 2.95015698587)
   * (RNO, 3.22158438576)
   * (BPT, 3.59953252824)
   * (VCT, 3.61190878378)
SFO
   * (SDF, -10.0)
   * (MSO, -4.0)
   * (PIH, -3.0)
   * (LGA, -1.75757575758)
   * (PIE, -1.34104046243)
   * (OAK, -0.813200498132)
   * (FAR, 0.0)
   * (BNA, 2.42596644785)
```

Q2.2 - For each airport X, rank the top-10 airports in decreasing order of on-time departure performance from X.

- At each mapper: We output partial sums of departure delays for each departure/arrival airport pair, using the departure airport identifier as the key.
- At each reducer: For each departure/arrival airport pair, we computed the total departure delay, sorted and the top 10 were written to the database in batches, indexed by the departure airport identifier.
- Result:

```
# ./query.py "CMI" "BWI" "MIA" "LAX" "IAH" "SFO"
CMI
   * (ABI, -7.0)
   * (PIT, 1.10243055556)
   * (CVG, 1.89476168004)
   * (DAY, 3.11623529412)
   * (STL, 3.98167330677)
   * (PIA, 4.59189189189)
   * (DFW, 5.94414274631)
   * (ATL, 6.66513761468)
   * (ORD, 8.19409814324)
BWI
   * (SAV, -7.0)
   * (MLB, 1.15536723164)
   * (DAB, 1.46959459459)
   * (SRQ, 1.58848388801)
   * (IAD, 1.79094076655)
   * (UCA, 3.65416985463)
   * (CHO, 3.74492753623)
   * (GSP, 4.19768664564)
   * (SJU, 4.44465842287)
   * (OAJ, 4.47111111111)
MIA
   * (SHV, 0.0)
   * (BUF, 1.0)
   * (SAN, 1.71038251366)
   * (SLC, 2.53719008264)
   * (HOU, 2.91219912473)
   * (ISP, 3.64739884393)
   * (MEM, 3.74510662248)
   * (PSE, 3.97584541063)
   * (TLH, 4.26148447469)
   * (MCI, 4.61224489796)
LAX
   * (SDF, -16.0)
   * (IDA, -7.0)
   * (DRO, -6.0)
   * (RSW, -3.0)
   * (LAX, -2.0)
   * (BZN, -0.727272727273)
   * (MAF, 0.0)
   * (PIH, 0.0)
   * (IYK, 1.26982474406)
   * (MFE, 1.37647058824)
IAH
```

```
   * (MSN, -2.0)
   * (AGS, -0.61879049676)
   * (MLI, -0.5)
   * (EFD, 1.88770821367)
   * (HOU, 2.17203698515)
   * (JAC, 2.57058823529)
   * (MTJ, 2.95015698587)
   * (RNO, 3.22158438576)
   * (BPT, 3.59953252824)
   * (VCT, 3.61190878378)
SFO
   * (SDF, -10.0)
   * (MSO, -4.0)
   * (PIH, -3.0)
   * (LGA, -1.75757575758)
   * (PIE, -1.34104046243)
   * (OAK, -0.813200498132)
   * (FAR, 0.0)
   * (BNA, 2.42596644785)
   * (MEM, 3.30248229975)
   * (SCK, 4.0)
```

Q2.4 - For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.

- At each mapper: We output partial sums and counts (to calculate means later) of arrival delays for each arrival/departure airport pair, using both the departure and arrival airport identifiers as the key.
- At each reducer: For each departure/arrival airport pair, we computed the total arrival delay and count, divided them to calculate the mean, and wrote the result to the database in batches, indexed by both the departure and arrival airport identifiers.
- Result:

```
# ./query.py "CMI,ORD" "IND,CMH" "DFW,IAH" "LAX,SFO" "JFK,LAX" "ATL,PHX"
   * CMI -> ORD: 10.1436629064
   * IND -> CMH: 2.89990366089
   * DFW -> IAH: 7.65444252577
   * LAX -> SFO: 9.58928273111
   * JFK -> LAX: 6.63511915527
   * ATL -> PHX: 9.02134188151
```

Group 3 (Answer both questions using Hadoop.):

Q3.1 - Does the popularity distribution of airports follow a Zipf distribution? If not, what distribution does it follow?

We first generated a dataset of airport popularities (taken as the number of arrivals plus departures) for each airport, computing partial sums of arrivals and departures for each airport at the mappers and the totals at the reducers. The result was stored in HDFS.

To answer the first part of the question, we used Python's SciPy library to randomly sample from a Zipfian distribution and run a two-tailed Kolmogorov-Smirnov test against our data, which gave a p-value of 5.65869349497e-21. Assuming a standard 0.05 confidence level, we rejected the null hypothesis that the two samples were drawn from the same distribution.

For the second part, we then run the same test this time between a random sample from a log-normal distribution and our data, which gave a p-value of 0.0581649848755. This time we failed to reject the null hypothesis, which means the log-normal distribution is a reasonable fit for our dataset.

```
# ./kstest.sh
* Kolmogorov-Smirnov two-tailed p-value (zipf): 5.65869349497e-21
* Kolmogorov-Smirnov two-tailed p-value (lognorm): 0.0581649848755
```

Q3.2 - Tom wants to travel from airport X to airport Z. However, Tom also wants to stop at airport Y for some sightseeing on the way. More concretely, Tom has the following requirements (for specific queries, see the Task 1 Queries and Task 2 Queries): a) The second leg of the journey (flight Y-Z) must depart two days after the first leg (flight X-Y). For example, if X-Y departs on January 5, 2008, Y-Z must depart on January 7, 2008. b) Tom wants his flights scheduled to depart airport X before 12:00 PM local time and to depart airport Y after 12:00 PM local time. c) Tom wants to arrive at each destination with as little delay as possible. You can assume you know the actual delay of each flight.

- At each mapper: First we classified flights into first leg and second leg. For second leg flights we subtracted two days from the flight's date to match them with the right first leg flights in the reducers. For each flight we output the departure, the date of departure, the arrival, the date of arrival, the airline, the flight number and an indicator of the leg number (one or two) using the date and month as the key.
- At each reducer: We received all possible flight combinations for the day and month indicated by the key. For each date/month/departure/arrival we wrote flight information to the database in batches, indexed by leg number, date, month, departure and arrival.
- Result:

```
# ./query.py "4,3,2008,CMI,ORD,LAX" "9,9,2008,JAX,DFW,CRP" "1,4,2008,SLC,BFL,LAX" \
             "12,7,2008,LAX,SFO,PHX" "10,6,2008,DFW,ORD,DFW" "1,1,2008,LAX,ORD,JFK"
CMI -> ORD -> LAX, 3/4/2008
  * Total arrival delay: -38.0
  1. First leg:
    * Origin: CMI
    * Destination: ORD
    * Airline/Flight Number: MQ/4278
    * Sched Depart: 07:10 3/4/2008
    * Arrival delay: -14.0
  2. Second leg:
    * Origin: ORD
    * Destination: LAX
    * Airline/Flight Number: AA/607
    * Sched Depart: 19:50 3/6/2008
    * Arrival delay: -24.0
JAX -> DFW -> CRP, 9/9/2008
  * Total arrival delay: -6.0
  1. First leg:
    * Origin: JAX
    * Destination: DFW
    * Airline/Flight Number: AA/845
    * Sched Depart: 07:25 9/9/2008
    * Arrival delay: 1.0
  2. Second leg:
    * Origin: DFW
    * Destination: CRP
    * Airline/Flight Number: MQ/3627
    * Sched Depart: 16:45 9/11/2008
    * Arrival delay: -7.0
SLC -> BFL -> LAX, 4/1/2008
  * Total arrival delay: 18.0
  1. First leg:
    * Origin: SLC
    * Destination: BFL
    * Airline/Flight Number: OO/3755
    * Sched Depart: 11:00 4/1/2008
    * Arrival delay: 12.0
  2. Second leg:
    * Origin: BFL
    * Destination: LAX
    * Airline/Flight Number: OO/5429
    * Sched Depart: 14:55 4/3/2008
    * Arrival delay: 6.0
LAX -> SFO -> PHX, 7/12/2008
  * Total arrival delay: -32.0
  1. First leg:
    * Origin: LAX
    * Destination: SFO
```

```
      * Airline/Flight Number: WN/3534
      * Sched Depart: 06:50 7/12/2008
      * Arrival delay: -13.0
    2. Second leg:
      * Origin: SFO
      * Destination: PHX
      * Airline/Flight Number: US/412
      * Sched Depart: 19:25 7/14/2008
      * Arrival delay: -19.0
  DFW -> ORD -> DFW, 6/10/2008
    * Total arrival delay: -31.0
    1. First leg:
      * Origin: DFW
      * Destination: ORD
      * Airline/Flight Number: UA/1104
      * Sched Depart: 07:00 6/10/2008
      * Arrival delay: -21.0
    2. Second leg:
      * Origin: ORD
      * Destination: DFW
      * Airline/Flight Number: AA/2341
      * Sched Depart: 16:45 6/12/2008
      * Arrival delay: -10.0
  LAX -> ORD -> JFK, 1/1/2008
    * Total arrival delay: -6.0
    1. First leg:
      * Origin: LAX
      * Destination: ORD
      * Airline/Flight Number: UA/944
      * Sched Depart: 07:05 1/1/2008
      * Arrival delay: 1.0
    2. Second leg:
      * Origin: ORD
      * Destination: JFK
      * Airline/Flight Number: B6/918
      * Sched Depart: 19:00 1/3/2008
      * Arrival delay: -7.0
```

Conclusions

This analysis could be used to further investigate problems in flight delays or calculate routes that minimize delays.

Despite the size of the dataset, the analysis was fast, and it could be made faster by horizontally scaling the cluster, although the maintenance costs should be taken into consideration too.