CLOUD COMPUTING CAPSTONE

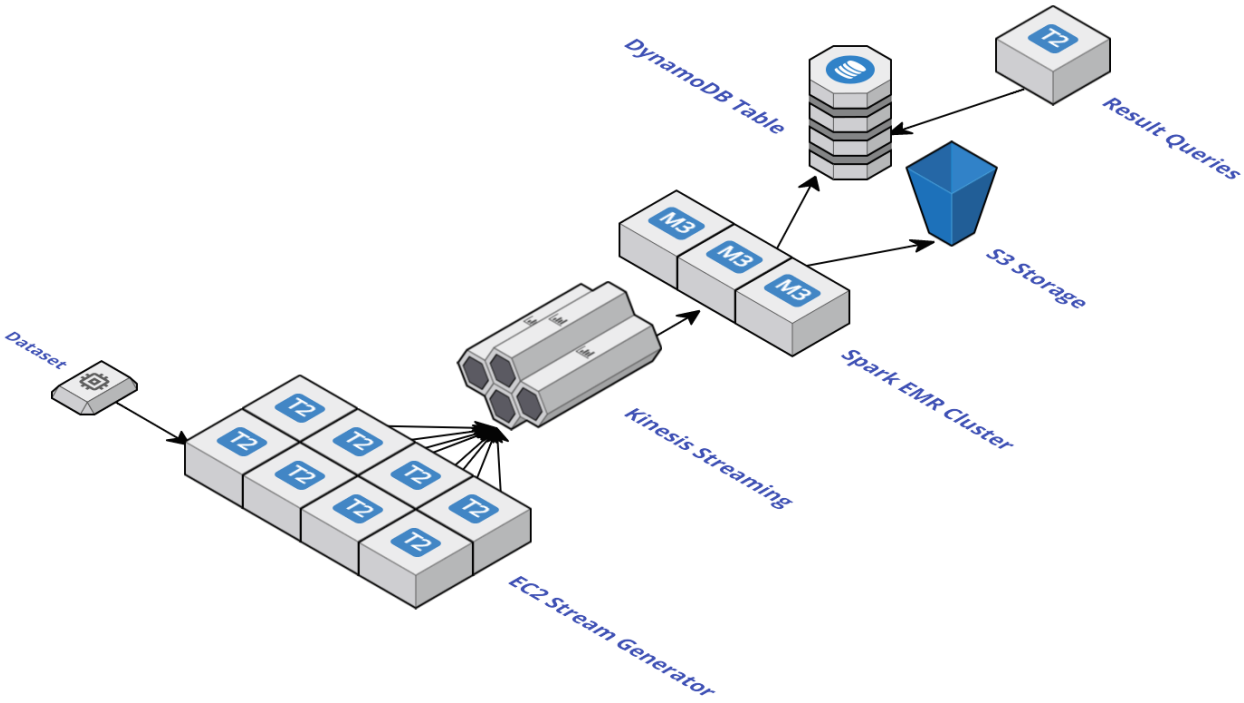
University of Illinois at Urbana-Champaign
Filipe Barretto

VIDEO DEMONSTRATION

https://youtu.be/XjEBAKqhBYw

SYSTEM INTEGRATION

The system is composed by the public transportation dataset, stored in EBS volumes, AWS EC2 instances to generate the stream that will be processed, AWS Kinesis integrated with AWS EMR running a Spark Cluster to process the stream, *AWS S3,* and *AWS DynamoDB* to store the results. The dataset was attached to many EC2 instances, generating a stream into Kinesis, processed by the spark application developed in Scala, and writing some results in S3 and the results that will be queried later on DynamoDB. The query applications were developed in Python.



APPROACHES AND ALGORITHMS

To answer the questions, a the Spark application ran Hive queries to get the desired results. For the first set of questions, a simple SQL code to select origin, destination or carrier, grouping the results was implemented.
For the second set of questions, a little bit more of complexity was implemented in the Hive queries. Each group has two columns and the results are added to *AWS DynamoDB* with a primary key as an id based on the flight date, a secondary key based on the flight route, some sort keys to help the queries and a field for the value desired in each question.
Finally, for the third set of questions, to view the distribution, Tableau (http://www.tableau.com/) was used. For the query, the data was stored on *DynamoDB*, with fields such as date and delay time, for future use, and some information was stored straight to AWS S3.

RESULTS

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

FROM	FLIGHTS
DFW	12,670,400
ORD	8,309,200
IAH	8,265,200
EWR	5,926,400
LAX	5,345,200
PHX	5,243,600
DEN	5,023,600
LAS	4,671,600
SFO	4,638,800
SEA	3,639,600

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

AIRLINE	DELAYS
CO	20.1229163906
DL	19.9960944891
UA	19.7470263526
US	17.7809600635
HP	13.0000855131
WN	12.7638023573
TW	10.6300884956
AA	9.48194022958
NW	9.4642270351
AS	6.63950087736

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

SRQ

ORIGIN	DESTINATION
NW	0.0
UA	0.0
US	0.0
TW	0.0
DL	0.0
AA	0.0

CMH

ORIGIN	DESTINATION
CO	124.0
HP	22.0
US	14.0
TW	2.0
DL	1.0
NW	0.0
UA	0.0
AA	0.0

JFK

ORIGIN	DESTINATION
TW	15.0
DL	9.0
PI	6.0
US	4.0
NW	0.0
UA	0.0
HP	0.0
AA	0.0

SEA

ORIGIN	DESTINATION
AA	31.0
AS	15.0
NW	0.0
UA	0.0
US	0.0
HP	0.0
TW	0.0
DL	0.0
PS	0.0

BOS

ORIGIN	DESTINATION
CO	54.0
HP	29.0
TW	6.0
DL	6.0
US	4.0
UA	2.0
NW	0.0
EA	0.0
AA	0.0

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

SRQ

ORIGIN	DESTINATION
CLT	4.0
DFW	4.0
LGA	2.0
RDU	2.0
ATL	0.0
DTW	0.0
RSW	0.0
PBI	0.0
STL	0.0
PHL	0.0

CMH

ORIGIN	DESTINATION
DEN	124.0
LAS	77.0
PHL	14.0
DAY	4.0
PHX	3.0
STL	2.0
PIT	2.0
ATL	1.0
ORD	0.0
MDW	0.0

JFK

ORIGIN	DESTINATION
BWI	78.0
LAS	43.0
TPA	42.0
MSY	32.0
STL	28.0
DTW	15.0
PHL	9.0
UCA	6.0
PIT	4.0
IAH	2.0

SEA

ORIGIN	DESTINATION
SJC	99.0
ORD	68.0
JNU	20.0
PDX	15.0
HNL	12.0
PIT	4.0
MIA	4.0
LAX	3.0
LAS	0.0
PHX	0.0

BOS

ORIGIN	DESTINATION
SFO	90.0
SLC	90.0
MCO	83.0
FLL	60.0
BWI	59.0
EWR	54.0
PIT	36.0
SEA	31.0
LAS	29.0
CVG	25.0

For each source-destination pair X-Y, rank the top-10 carriers in decreasing order of on-time arrival performance at Y from X

LGA → BOS

CARRIER	DELAYS
DL	21.0
EA	4.0
NW	0.0

BOS → LGA

CARRIER	DELAYS
EA	10.0
DL	8.0
NW	3.0

OKC → DFW

CARRIER	DELAYS
DL	0.0
TW	0.0
AA	0.0

MSP → ATL

CARRIER	DELAYS
NW	9.0

Does the popularity distribution of airports follow a Zipf distribution?

As we can see from the image below and the data, that has the airport on the X Axis and the number of flights on the Y Axis, the popularity distribution of airports does not follow a Zipf distribution, since the number os flights of an airport n is not approximately twice as much as n-1. The distribution follows a Logarithmic distribution.

Image 01: Airport popularity distribution

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.

Queries did not return any valid route for the given inputs.

OPTIMIZATIONS

The optimizations made were in the parallel streaming of the data, using high performance instances streaming data to AWS Kinesis. This optimization, however, I believed compromised my results, since I did not configure DynamoDB write throughput accordingly and believe lost some data that was only realized after the final run. The entire system was run on AWS with high processing and low latency (EC2, EMR, DynamoDB, S3). In the application, some optimizations were made in the code. The Spark application was developed in Scala and Python for the queries on DynamoDB.

DISCUSSIONS ABOUT RESULTS

The results, in my opinion, make sense but are a little off target. I some some connectivity problems while setting up the system that compromised a little, but the major issue I believe was the write capacity on DynamoDB that I had set to a smaller number, compromising my results. For smaller dataset tests with only a single stream instead of multiple parallel streams, the results were as expected. Running with multiple streams to get the results faster did not work as expected. Since I didn't have the time nor AWS credits to rerun the entire system, I had to summit with some information missing. On the hurry to finish up, I even inverted one of the queries for the top airlines when I got the worst airlines performance.

The popularity distribution was also expected, since there are few hubs and many airports that fly to and from these main hubs, also expected from the results of task 1.
The results are useful and are used every day on flight ticket search, as on question 3.2, for example. The data becomes even more valuable when added more filters, such as price, number of seats. This kind of query is done millions of times every single day by major flight websites in the world.
Further analysis of this historical data could also help develop prediction algorithms of delays and other information, that can be of great value for airports and airlines to pursue optimizations.

HOW DID THE DIFFERENT STACKS (HADOOP AND SPARK) FROM TASK 1 AND TASK 2 COMPARE? WHICH STACK DID YOU FIND THE EASIEST TO USE? THE FASTED?

Overall, I believe its not simple to compare the stacks for task 1 and task 2 based only on the experience during this capstone. The changes in the requirements, such as canceling Storm and changing Spark from batch processing to streaming made it difficult to adapt and properly work with all stacks, specially with a short time frame.
Basically, I preferred using Hadoop, since I had a prior basic knowledge and had done some simple word count applications before this course. Using Spark I had a huge difficulty finding the documentation, that was mostly in Scala, a programming language that I had never worked with before and had a lot of trouble adapting in the short time we had for task 2. Besides I had some trouble setting up the streaming environment for task 2, while on task one was much simpler and straight forward for batch processing.
During my studies for task 2, I tried batch processing with Spark to start learning and was really impressed by the speed on with I got the results, that was much faster than my work on task 1.
By the end of the capstone, even with a greater knowledge on Hadoop, I became really interested in working more with Spark and exploring the possibilities to work with streaming.