The Impact of Covid-19 on Air Traffic: Spatiotemporal Forecasting and Benchmarking

Adelle Driker, Bo Yan, Yuan Hu

Advisor: Professor Rose Yu

Scalability and Robustness Requirements

Explanation of Scalability

More Data

We are currently using our dataset at the level of the Country of where the flights are departing from. To scale this and be able to provide data at a more granular level, we have the ability to reprocess the raw data to provide information about the numbers of flights from specific airports around the world.

More Storage

Our data currently comprises a size of several gigabytes - we are storing the raw data in an S3 bucket and then transfer it to an InfluxDB database after processing. We are able to flexibly expand storage in the InfluxDB database when incorporating new data.

More Processing Power

Our Python Notebooks currently utilize CPU power, and we have tested out using a single GPU using Google Collab. By converting our notebooks to run on a Sagemaker instance in AWS, we will be able to utilize more powerful processors of our choice to improve performance in both data preprocessing and modeling aspects. Additionally, this improvement will allow us to handle more users, whose increased number of requests would impact the model and dashboard.

Summary of Case Studies that Include Robustness

In order to be better equipped to resist failures within our product, we have researched ways to incorporate robustness into our work. In our review, we have noted that a robust data pipeline should be able to only process the required data, have checkpoints, and be very well-documented[1]. Not only does this help us build a resilient and reliable pipeline, but it also is able to set up a foundation for our model to incorporate data in a stable condition to provide the end-user a reasonable set of results. With regard to the end-user, the pipeline (including the final model and dashboard) must be thoroughly tested to ensure that it will be able to handle a variety of common user requests. If a certain request, which proves to be common, cannot be handled by the pipeline, we will ensure that the proper update to the pipeline will be added to assure robustness.

Illustration of Data Pipelines and End-to-End Process

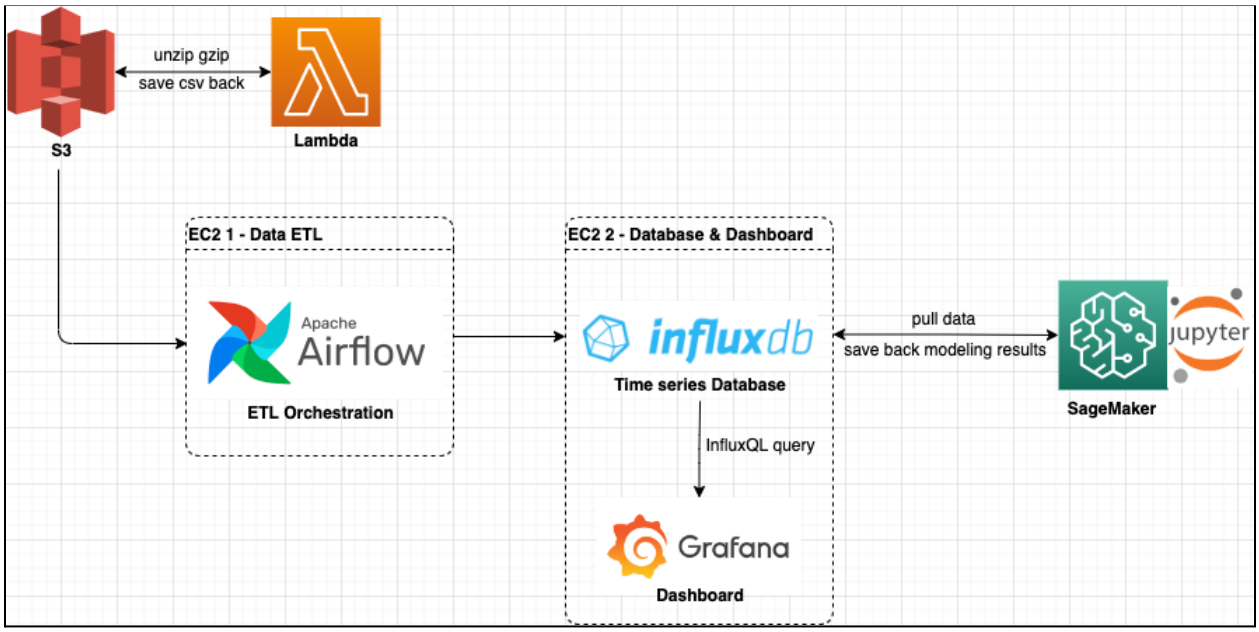

Figure 1. Project Architecture

Major architecture components

Choice of major architecture components includes AWS S3, lambda function, Airflow orchestrated ETL process, InfluxDB database, AWS SageMaker and Grafana.

AWS S3

S3 provides us elastic scalability data storage. First, S3 has no storage limit. Individual objects can be up to 5TB in size. Data is stored in buckets. Each bucket can store an unlimited amount of unstructured data. Additionally, each object is identified using a unique key, and we can use metadata to flexibly organize data.

Lambda function

Lambda is engineered to provide managed scaling in a way that does not rely upon threading or any custom engineering in our code. As traffic increases, Lambda increases the number of concurrent executions of our functions.

Airflow Orchestrated ETL process

The ETL process is orchestrated by Airflow. More specifically, celery executors use standing workers to run tasks. Using the Celery executor we can configure both the number of worker nodes and their size. More the number of worker nodes we have available in our environment, or larger the size of our workers, the more scaling we can achieve. Hence, this huge amount of resources will have more than enough capacity to run hundreds of tasks concurrently.

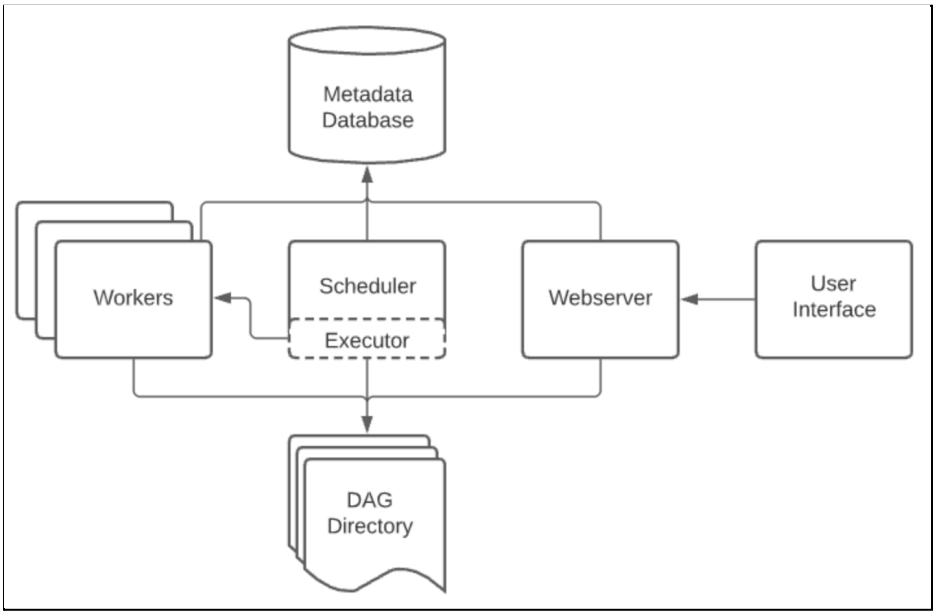

Figure 2. Airflow Celery executor Architecture

InfluxDB database

InfluxDB stores time-series data with a clustered instance, which consists of meta nodes and data nodes. Meta nodes hold all the metadata including all nodes in the cluster and their role, all databases and retention policies that exist in the cluster, cluster users and their permissions and all shards and shard groups. Data nodes hold all of the raw time-series data including measurements, tag keys and values, field keys and values. Based on scalability requirements, we can bring more/reduce the number of data nodes to meet our needs.

AWS Sagemaker

Amazon SageMaker supports automatic scaling (autoscaling) for our hosted models.
Autoscaling dynamically adjusts the number of instances provisioned for a model in response to changes in our workload. When the workload increases, autoscaling brings more instances online. When the workload decreases, autoscaling removes unnecessary instances so that we don't pay for provisioned instances that we aren't using.

End-to-end process

Firstly, raw gzip files will be uploaded to the S3 bucket <air-traffic-raw>, simultaneously, a lambda function will be automatically triggered by the S3PUT event, gzip file will be unzipped and saved back to the S3 bucket <air-traffic-csv>.
Then, csv files will be fed into the ETL pipeline for processing and transformation. The main stages of ETL were scripted with relevant Airflow operators and saved in the DAG directory. Transformation mainly aims at output processed data for generating EDA results and preparing data for modeling. The last stage of the ETL process is to load processed data into the influxdb database as measurements.

Once measurements are stored, we run notebooks with the influxdb-python API to pull required measurements for constructing and evaluating our models; the results are loaded back into influxdb.
As the last stop of data flow, the grafana dashboard pulls data stored in databases via influxQL query language and renders visualizations.

Approach

More Data

The method to adjust this within our preprocessing script is straightforward, and we also have the means to impute a large amount of missing data.The COVID data will need to be mapped to each airport's country, which can be sourced from a mapping file. The resulting structure of the data that will be ingested by the model will be the same as that at a country level, and further adjustments will need to be made to the model.

More Storage

InfluxDB and S3 would be able to be flexibly upgraded to hold more data; however, we would incur higher costs with S3 as more data is housed there. It will be necessary to eventually find another storage space for the raw data once our AWS allowance is reached.

More Processing Power

Once implemented in a Sagemaker instance, the notebook will be able to be run with a processor of our choice. Since the dataset is constantly growing as new data become available, having more processing power available will help future-proof our product as well.

Evaluation Strategy and Plan

Our evaluation strategy will consist of making sure that the same quality of results is achieved. We define our success as being able to match results before and after the scalability update and observing that the system is displaying statistically significant improvements in performance. We plan to implement each scalability improvement independently of the others to ensure that one enhancement does not negatively impact the others.

Results and Next Steps

With the above changes, we should be able to see improvements in performance while keeping the quality of results the same as before the change. Most of the updates are straightforward and should require little effort to put into place. Others, like increasing the dataset size, will require a small amount of code modification and the creation of the mapping file between airports and countries. Next steps include implementing these changes according to the plan

described above, which is not completely set in stone and may be adjusted in case roadblocks occur.

Major Updates to Steps 1-7

Further progress on the final dashboard has been made - we are testing the ability to present the modeling results and be able to filter them down. We have generated some results that focus on the forecast for the airlines from the United States as a prototype of the final product. We are also in the process of implementing our own version of a DCRNN model based on Professor Yu's original version in PyTorch. This model is better able to incorporate location as an additional attribute and has outperformed several high-performing models on a set of traffic data according to one of Professor Yu's papers[2].

Team Member Contributions

Bo:

- Wrote and debug scripts for DCRNN model, continued debugging Seq2Seq model
- Generated data against all models for global flights prediction results to support Yuan's setting up of grafana to visualize global flights
- Performed coding and performance tuning against all models for a specific country's flights prediction to support Yuan's setting up of grafana to visualize US flights
- Maintained Capstone Project Planning spreadsheet, GitHub, and Rose's Documentation

Yuan:

- Set up and configured EC2 instances for Airflow, influxDB and grafana.
- Scripted lambda function, Airflow DAGs for ETL process
- Established panels within grafana dashboard
- Contributed to "Illustration of Data Pipelines and End-to-End Process" of the report

Adelle:

- Contributed to the "Explanation of Scalability", "Summary of Case Studies that Include Robustness", "Approach", "Evaluation Strategy and Plan", "Results and Next Steps", and "Updates" sections of the report
- Researched and experimented with DCRNN model based on Professor Yu's demo version and TorchTS documentation

Citations

(1) https://medium.com/@gohitvaranasi/how-to-build-robust-data-pipelines-in-the-big-data-ecosystem-806f84d9009f

(2) https://paperswithcode.com/paper/diffusion-convolutional-recurrent-neural