CS 5412 Intermediate Report

Harshwardhan Jain (hj364), Sanjana Kaundinya (ssk228), Rohit Khanwani (rk632)

March 28th, 2018

1 Overview

We are currently implementing an application for airports and airlines that uses flight schedules and real time feed from airborne aircrafts to give routes that minimize traffic, drag, and flight costs using machine learning models trained on historical data. We foresee our clients to be the air traffic controllers in the air traffic watch towers monitoring flights and talking to the pilots about flight take-off and landing.

2 Data

We visualize our system, on a higher level, dealing with two kinds of data:

- **Static Data**: Data that doesn't change much, for example flight numbers, airports, possible routes and long-term flight schedules.
- **Dynamic Data**: Data that flows in through the sensors in real-time, for example aircraft altitude, location and weather updates.

We researched free APIs that could provide us with the data we need to test our application. Using the Bureau of Transportation Statistics' Transtats portal, we were able to obtain most of the static data that we require. This included flight information and flight schedules. However, obtaining route data has been a challenge, possibly because of the scale and security reasons. Therefore, we plan on constructing data that somewhat represents real-world scenarios for routes, and the dynamic data. We are considering drawing the dynamic data from the possible routes, at least for flight co-ordinates, and constructing values for altitude and other fields that we may need. This data will essentially mimic real time data coming in from sensors on the ground.

3 Interface

The interface that we are working on is going to have three different components with which the client will be able to interact with. The first component is the monitoring of ongoing arrivals and departures of aircrafts. Our system will be doing computations to figure out the most optimal sequence in which the aircrafts should take out and land such that it can reduce drag and also overall fuel usage. We are currently working on the static data that we have acquired from our research to use as factors when making these computations. Other factors that will be at play are things such as aircraft size (which we have researched and plan to use the average commercial plane for our example data) as well as the real-time data of incoming flights to land and outgoing flights to take off, which as mentioned before we are considering creating manually as our research has not proved to find us much on this font. We are in the process to come up with an algorithm that will take all these constraints into account and give a plan that will optimize for all these conditions.

The next component that we will have to work on will be to reduce overall flight traffic. This will also provide data to our clients working in the air traffic control tower and make recommendations to also time landings and take-offs to reduce traffic as much as possible on the ground, and in the air. Reducing flight traffic will help reduce delay times and allow for passengers to board and get off their plane faster, instead

of having to wait on the plane while it taxis and tries to find an open gate; or wait on the runway trying to leave the airport. This will be based on the flight schedules for the day, which we presume can be grabbed from airport websites as JSON strings and broken into the components we need. The goal of this component is to optimally sequence flights in order to reduce traffic.

The final component (which we still have to plan out and implement) and probably the most important component, will be to significantly reduce delays and missed flights for passengers. This will be based on active real time delays due to weather, aircraft maintenance, and other miscellaneous reasons. Our system will apply a machine learning model that will look at passengers' connecting flights and try to delay these flights so that a large number of passengers won't miss their flight. Essentially it will optimize for a large group of people with the same connecting flight, effectively causing a delay for these passengers such that they won't miss their connection and can get to their destination on time. These calculations will have to be done in real time, and will have to be fast and accurate, so the recommendation that the system makes to the air traffic controller will be the best one at the time.

Here is a pseudo-code example of the kind of query interface we are trying to build and implement for this project:

```
{[route], [flight schedule], [weather conditions]} ==>

{[optimized arrivals/departures], [fuel reduction plan], [flight traffic reduction protocol]}
```

So given all these inputs, we are trying to output the above following features.

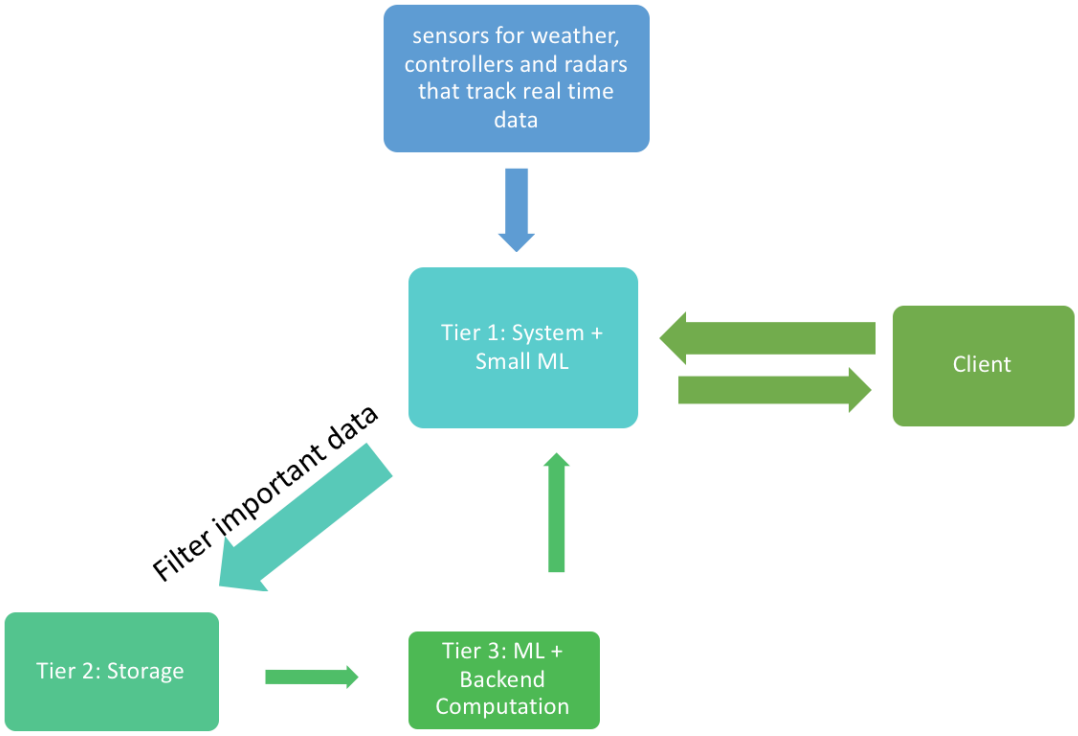
4 Implementation/Architecture

Our system is going to have three main tiers through which the data from sensors, radars, and controllers will pass through. They are the following:

- **Sensors**: The aircrafts are going to act as sensors, sending real time feed like temperature, position, direction, traffic, drag, wind speed, etc. This information is going to be used to update routes for existing flights/flights departing soon based on the output of machine learning models from the first tier.
- **Tier 1**: The sensors are going to send real time feed to tier 1 of the system. The first tier will be running local machine learning models (updated periodically by the central machine learning model) and communicate any changes/information with the client over the network. The client will then be getting in touch with pilots to communicate appropriate changes. The first tier will be the only point of contact between the sensors and the client and all communication will be done over a secure network with encrypted messages. We are imagining our clients to be the air traffic controllers that are in constant touch with pilots. Additionally, the first tier will also be filtering and sending relevant data to tier 2 (discussed below) for storage and for updating the central machine learning algorithm.
- **Tier 2**: This is going to be the storage unit for the application. All data (historical, new data sent from tier 1, schedules, etc.) reside on this part of the stack. This tier is a central data unit for the machine learning model to train on. Every unit in the first tier sends data to this tier.
- **Tier 3 - ML**: This is the last layer of the application. This layer contains the central machine learning model(s) that is locally cached (with some local changes) at tier 1. It gets new data from tier 2 and updates the machine learning model. It pushes the updated models to the first tier periodically.

The Tier 1 system in a real world scenario would be the edge in this situation, that will be modeled as multiple points of presence in the real world. For our project, it's going to be a service that runs on a laptop that will constantly filter in the real time data coming in from the sensors (which we will model as active data being fed in from the open source flight data we find). In addition, this Tier is the one that interfaces with the client so we are thinking of trying to make some sort of GUI that will make recommendations based

Figure 1: Architectural Diagram for our System



on the three components that we outlined and are currently working on previously in our Interface section.

After researching on various cloud providers, we have decided to go with the free tier on Google Cloud Platform. In addition to a few credits, the GCP free tier offers some of the Google Cloud products for free up to a given quota. Some of these are very useful, and provide automatic scalability benefits that we believe will be vital to our application. More specifically, we plan on using the App Engine platform to develop and deploy our application, and the Google Datastore for storing our data. Previously we had considered using the Google Cloud Storage platform, with the Unified Object Storage paradigm, however we quickly realized that the Cloud Storage platform is good for storing unstructured data such as images. Since most of our data is structured, we switched over to Datastore, which is a NoSQL document database that offers a high availability of reads and writes and massive scalability. This is ideal for our use-case. Additionally, the free tier offers services that we may use in the future such as Compute Engine, and Cloud Pub/Sub. We also believe that App Engine applications integrate well with TensorFlow applications, which would allow for a clean integration of our ML component.

5 Evaluation

When we come around to evaluation and testing of our system we will do the following. To simulate a real world environment, we will start off by running n (with values of n from 1 to 100) instances of the application on different virtual machines on our desktop and then we will deploy it on the cloud to run more instances. For around 7-8 sensors, we will have 1 process that acts as the tier 1 for those sensors and communicates with them. We will vary the number of processes for each tier to account for different traffic situations and depict the busier areas like Atlanta, etc. We will also have some tests for the machine learning model to

analyze its correctness and soundness. We will log the latency for different scenarios to make sure acceptable standards of performance are met.

To check if the machine learning model is updated periodically, we will train the model on datasets drastically different from the live feed and check if the machine learning classifier is updated as planned. This will ensure that information flows accurately among different tiers of the application. Another aspect we aim on testing is whether or not a sensor/flight is sending data to the correct ATC airport, as per proximity to departure and arrival airports.

In addition to the data that has to be retained in tier 2, we will add additional useless data that has to be filtered by tier 1 to simulate real-world situations. This would allow us to test if our first tier is only retaining necessary data accurately.

We believe that evaluating the application using these methods would allow us to exhaustively test if our application is scalable and high-performing in terms of both number of clients and number of sensors/flights supported. The operating costs of the application can be assessed by analyzing bandwidth. We will be adding more evaluation methods as needed while we continue to implement the application.

6 Milestones and Group Assignments

- March 7th, Sanjana and Harsh: Research appropriate server side languages and start laying foundation of detailed implementation for the tiered system
- March 14th, Rohit and Harsh: Research ways to get all the data needed. If required, look for appropriate ways to construct data points.
- March 18th, Rohit and Sanjana: Look into Google Cloud storage and ML services. Decide on a provider and follow online tutorials by that provider to get a feel of the development environment
- March 21st, Sanjana, Harsh and Rohit: Start coding up server side
- March 21st, Harsh and Rohit: Research ML models we can potentially use for our system
- April 4th, Sanjana and Rohit: Hook up server that's been coded up and Google Cloud services
- April 4th, Harsh and Rohit: Finalize on an ML model and being implementation
- April 12th, Rohit and Harsh: Integrate ML model into Google Cloud services
- April 18th, Sanjana and Rohit: Test server side code and its filtering properties with data from openflight
- April 18th, Harsh: Train ML model on data
- May 2nd, Sanjana and Rohit: Start creating a testing suit to rigorously test the Tier 2 and Tier 3 services
- May 2nd, Harsh: Create testing suite and try to further optimize ML model
- May 9th, everyone: Work together to create proper real time data use cases and rigorously test the system for fault tolerance, race conditions, latency, etc.

We added some milestones that were not originally included in the plan - such as the collection of data and exploring cloud providers and their tutorials. These milestones became clearer as we started thinking about the practical aspects of the development process. We plan to use convolutional neural networks for our machine learning models. Neural networks are known to learn pattern and differentiating features. We also considered using Support Vector Machines for their simplicity and (les) training resource requirement, but given the nature of the predictions and the requirement for highly accurate data, we have decided to use deep learning. We will train them on GPU's on the backend and ship the weight vectors with periodic updates.