Smart Air Traffic Control System

Rohit Khanwani, Harshwardhan Jain, Sanjana Kaundinya

Overview

We implemented an application for airports and airlines that uses flight schedules and real time data from airborne aircrafts that offers real time collision detection and gives routes based on this service, to prevent aircrafts in the air from crashing into one another. Our architecture is built in such a way that in the future, we can add more microservices for things such as fuel optimization and weather detection. We foresee our clients to be the sensors that communicate with the aircrafts and signaling how to go on with their route.

Types of Data Involved

Static: Data that doesn't change much, for example flight numbers, airports, possible routes and long-term flight schedules.

Dynamic: Data that flows in through the sensors in real-time, for example aircraft altitude, location and weather updates.

Interface

General idea of the query and response of our application:

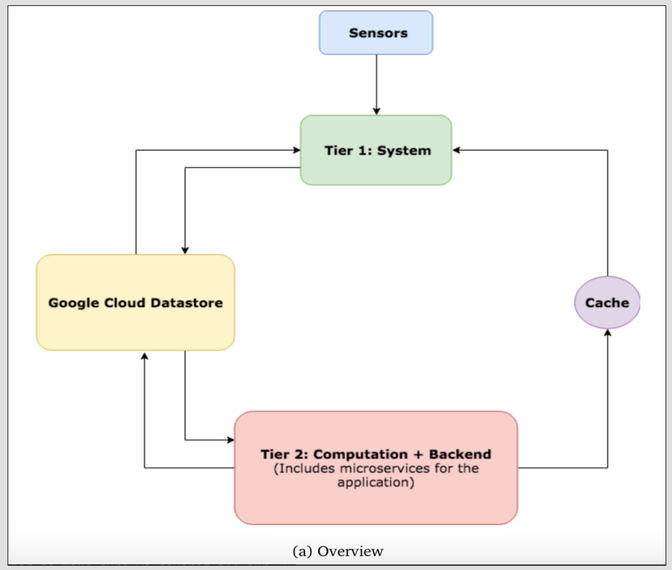
Input:

```
{[flight number], [location], [altitude], [speed], [weather conditions]}
```

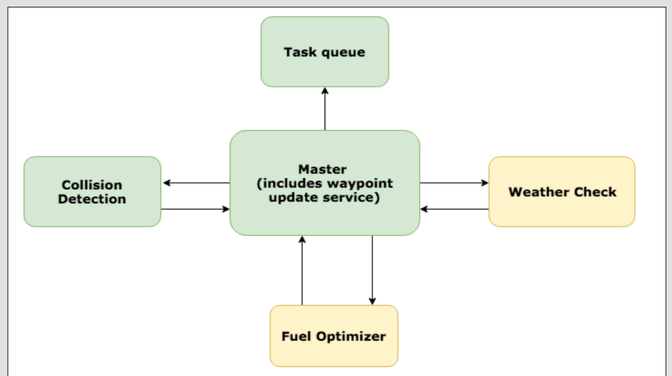
Output:

```
{[next optimal waypoint], [next optimal speed], [next optimal altitude]}
```

Architecture/Implementation



(a) Overview



(b) Tier 2

Figure 1: Application Architecture

Key Points:

- Tier 2 can host several different microservices
- All calls to and from Datastore are asynchronous
- Fault tolerance through request and data replication
- Consistency constraint is strictly enforced

Evaluation

For testing, we had a two part strategy. The first part dealt with the overall correctness of the system, as well as ensuring that extreme edge cases would be taken care of in this testing. The second part looked more at the infrastructure we had built around the application as a whole: load balancing, stress testing, scalability, performance with several requests, and more. To do the tests we had a two part method to it, first to write tests on our own and test correctness, and then to stress test it with running several instances on different computers to mimic many flights going through the system.

Figure 3: Overall Statistics for our System

App Engine Release	Total Instances	Average QPS	Average Latency	Average Memory
1.9.54	12	4.138	346 ms	49.83 MB

Figure 4: Memcache Details

Memcache service level	Hit ratio	Items in cache	Oldest item age	Total cache size
Shared Best effort. Change	90.7% 938,436 hit / 96,248 miss	2,640	2 sec	1.49 MB

Figure 5: Graph of Load Increase

(a) The graph below shows how the load was increased from 0 to over 700 flights in the span of 7 minutes

(b) The horizontal axis is the time lapse, and the vertical axis denotes the rate of increase