STRATOLAUNCH High Speed Flight Test Data

Hayes Bounds, Nathan McKinley, Saachi Tamboli, Vihaan Khajanchi, Tanush Ashok, Dariush Mokhlesi, Kian Mokhlesi

INTRODUCTON & BACKGROUND

Stratolaunch is taking big steps in the high-speed flight industry using the Roc, the world's largest aircraft, as a mobile launch platform to deploy high speed vehicles into the hypersonic environment. In order to properly address our goals for the year, we decided to split up into two different teams. The visualization team is developing an interactive web app to replace a pre-existing dashboard, while the comparative analysis team is researching and testing databases to find the most performant database that matches Stratolaunch's needs.

VISUALIZATION

Goals:

- Aid flight testing progression by allowing complex flight data to be read easily
- Create a web application prioritizing security, efficiency, and interactivity
- Shift away from using Plotly Dash, a simplified plotting tool already in use by Stratolaunch

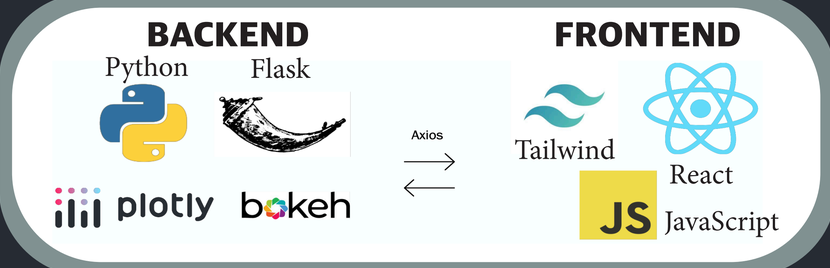
COMPARATIVE ANALYSIS

Goals:

- Evaluate the databases based on numerous metrics: query response times (how long it takes to receive queried data), disk utilization, etc.
- Ensure that the chosen database is powerful, robust, and scalable

TECH STACK

Depicts the web application's operating infrastructure



ACKNOWLEDGEMENTS

Special thanks to our Corporate Partner Mentors: Josh, Steven, and Conrad. Thank you to our TA Mahima, our Corporate Liason Lauren, and Dr. Ward and the whole of the Datamine.

RESEARCH METHODOLOGY

VISUALIZATION

- Investigated the performance of Bokeh and Plotly, two plotting packages, in tandem with React as a replacement for the Plotly Dash dashboard.
- Researched rendering methods: dynamically (render the plot on the frontend) or image generation (render an image on the backend and send it to the frontend)
- Utilized example code/documentation/tutorials to research React/Tailwind, two front-end web frameworks, in order to build our application

COMPARATIVE ANALYSIS

- Worked together to obtain a large list of database performance metrics: response times, query performance, and finding potential errors.
- Investigated how to properly test each of these metrics using specialized queries: finding a specific value, maximum, minimum, and average value from a large column of data
- Applied query testing to all databases

CONCLUSIONS

VISUALIZATION

Our project is a data visualization dashboard (below) with the following features:

- Rendering data from multiple CSV files simultaneously (side-by-side or overlaid)
- Specifying date and time range to plot

Bokeh/Plotly: Plotly WebGL is considerably faster than Bokeh on all tested samples

Dynamic/Image Rendering: Chose image rendering to ensure no data would be sent between client and server and for related performance reasons

Limits/Biases:

- Can only hold CSV data
- Selected React (because developers had prior experience with React), when other frameworks could have been more performant

COMPARATIVE ANALYSIS

InfluxDB

Ingest:
- Multiple Ingest Methods
  - InfluxDB Line Protocol for small data sets
  - Simple ingest from CSV

Querying:
- Utilizes Influx Query Language(IQL)
- Lots of documentation and easy translate from SQL and Python

Ease of Use:
- Appealing Web interface
- Built in query builder
- A lot of Documentation

QuestDB

Ingest:
- Can all be done within web console
- CSV ingest as easy as uploading file
- Automatic Data Type Detection and Conversion

Querying:
- Basic SQL
- Built in timestamp functions
- Easy translation to Python

Ease of Use:
- Easiest for any beginner
- A lot of documentation
- Best web console

TimescaleDB

Ingest:
- Manual ingest in terminal
- Requires all column data types to be pre-defined
- Unable to recognize dates in CSV
- Ingests random values

Querying:
- Basic SQL
- Easy translation to Python

Ease of Use:
- Requires additional download of PostgreSQL
- Little documentation regarding errors
- Poor web console

Apache Druid

Ingest:
- Can all be done within web console
- CSV ingest as easy as uploading file
- Database creates the code for the ingest with correct data types

Querying:
- Basic SQL
- Easy translation to Python

Ease of Use:
- Good Web Console(Similar to Quest)
- Very automated
- A lot of documentation

DolphinDB

Ingest:
- Simple ingest
- Unable to recognize dates in CSV and convert to timestamps

Querying:
- Basic SQL
- Easy to translate into Python
- Simple coding

Ease of Use:
- Has built in Python API to translate code into Python
- ODBC allows direct connection to Excel
- A lot of documentation

PLOTLY/BOKEH

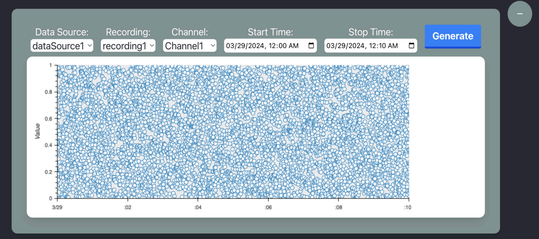
The below is a table depicting our plotting packages and their time-to-render in seconds. Utilized 1 sample and 10 sample testing (samples is the number of points plotted per second). SVG/WebGL/PNG indicates the rendering method used.

Performance Tests: bokeh vs plotly

Plot Tool	First Plot Time	Average Plot Time	Hardware	Samples	Length (seconds)
Plotly (WebGL)	2.635s	0.478s	MacBook Pro	1	3600
Plotly (WebGL)	1.138s	0.526s	MacBook Pro	10	3600
Bokeh (PNG)	2.5107s	0.853s	MacBook Pro	1	3600
Bokeh (PNG)	1.863s	1.004s	MacBook Pro	10	3600
Plotly (SVG)	0.957s	0.293s	MacBook Pro	1	3600
Plotly (SVG)	1.738s	1.402s	MacBook Pro	10	3600

APP

Depicts the current working version of the web application



FUTURE GOALS

VISUALIZATION

More interactivity:
- Currently researching zooming and panning features in two different ways: obtaining mouse coordinates, or using an existing package (ReactSVG)
- More interactive features can be added to allow for further inspection of the graph, like viewing the values of points

Dynamic graph modification:
- Allows engineers to rapidly modify graphs to plot complex data in simple ways
- Allows for one deployment of app, rather than multiple with different plotting functionalities
- For example, engineers could change axis names, point color, or even graph type (bar, line, box, etc.)

COMPARATIVE ANALYSIS

Our goal is to use the selected database to analyze data effectively:

- We must choose one of the following databases: DolphinDB, TimescaleDB, QuestDB, Apache ruid, InfluxDB
- The database we select must be capable of handling large-scale data efficiently and support real-time or near-real-time data processing and analytics
- Must be able to query metadata efficiently
- Flexible to allow for machine learning and predictive modeling
- Must be easy to use so we can analyze metadata provided by Stratolaunch