OPAL: LEVERAGING OPEN SOURCE

EDDGE Ogden
309th SWEG
Kenneth Call, Isaac J. Myers, Zayd L. Ma
Air Force Materiel Command
Hill AFB, UT
309sweg.eddge.ogden@us.af.mil

Daniel Lowe
Avionics Test and Analysis Corporation
dlowe@avtest.com

ABSTRACT

The current explosion of test and evaluation data being collected from various systems has exposed a strong need for low-cost digital infrastructure to facilitate scalable analytics across all available data. Private industry and academic research have built such systems utilizing Open-Source Software (OSS) with tremendous success. The 309th Software Egineering Group (SWEG) developed OPAL (Open Platform for Advanced Learning) platform is a government owned and developed solution to address this gap and provide data discovery, analytics, and warehousing all license free. OPAL leverages best-in-breed Open-Source Software including JupyterLab (Python analysis environment), MinIO (S3-compliant, redundant and object-versioning data backend), Postgres (Data cataloging), and Dask (scalable compute), among others. In addition to Open-Source tooling, custom integration and software piping are used to further lower the analysts' barrier to available data: custom Chapter 10 parsing and translating at high speed (10GB/min) into Apache Parquet format, a web-based data catalog for discovery, and lightweight arbitrary object storage organization. This paper will delineate design choices, our DevOps paradigm, benchmarking numbers, and results against a publicly available commercial flight dataset.

INTRODUCTION

Test and evaluation data are generated throughout the lifecycle of a system. As both the cost of system instrumentation and data archiving have decreased, the amount of collected data has exploded in volume. Although the cost of computer resources has also fallen at a similar pace, it yet remains a challenge to effectively use the data resources that have become available. The 309th SWEG is a subdivision of the US Air Force Materiel Command with the mission to provide sustainment for various software systems. Such sustainment efforts produce and interact with various large collections of legacy data and ongoing test and evaluation. Some of the challenges facing the 309th include accessing and interpreting legacy data, running analyses over extended

datasets, documenting useful provenance information, and leveraging advances in data analysis and big data approaches. Further complicating 309th challenges is the need to operate leanly in fully disconnected secure environments. OPAL (Open Platform for Advanced Learning) is a government owned and developed solution that integrates Open-Source Software into a single secure package that leverages OSS to address these challenges. The objective of this paper is to introduce OPAL architecture, simplified deployment, and analytics workflows developed on real world data. Our experience is that data can be used more effectively when organized in an on-line repository and novel analysis results can be obtained by pairing such an on-line repository with a modern analysis framework.

INITIAL TARGET DATA AND REQUIREMENTS

IRIG 106 [1] is a telemetry standard maintained by the Telemetry Group of the Range Commands Council. Chapter 10 and 11 of IRIG 106 defines the standard for On-board Recorders of telemetry data. These telemetry files are often known as "Chapter 10" files. Many flight platforms use the IRIG 106 standard for test and evaluation data recording. Chapter 10 files produced by an on-board recorder encapsulate the various data sources that may exist on the platform, which may include, but are not limited to, 1553 bus traffic, ethernet traffic, and flight video. A publicly accessible example of such a dataset can be built from NASA data after conversion using tools developed by ATAC (Avionics Test and Analysis Corporation).

IRIG 106 is an open standard and various tools exist to interpret Chapter 10 files. The OPAL team elected to develop a custom Chapter 10 interpreter named TIP (Translate, Ingest, Parse). This decision was driven by the desire for a light-weight and multi-threaded tool that can transform Chapter 10 into formats accessible by standard analysis tools. In the context of OPAL, TIP is employed to produce Apache Parquet [2] files that have strong library support, have built in statistics for efficient searching, and minimize size on disk through configurable compression.

Historically, flight telemetry data has been organized with human intensive processes that have resulted in a fragmented data provenance landscape which is costly to maintain due to institutional knowledge that might span decades. The result is data becomes difficult to search for specific flights by characteristics, to filter by particular features or events within flights, to guarantee that potentially desirable flight metadata is associated with each flight, or to access telemetry without deep institutional knowledge. The opportunity facing the 309th SWEG and the OPAL team is to tackle all these challenges. Figure 1 sketches out the legacy process. All analyses are driven by a user need. In the legacy case, data, often off-line, would need to be located, data of interest extracted using custom tools, then the data would be exported to other systems for analysis.

After a significant review of existing government off the self (GOTS) as well as commercial off the shelf (COTS) tooling and pricing, we decided to build our own minimal, but extensible, solution to meet the 309th SWEG need. Namely we need a system that meets the following requirements:

- Completely license free so that smaller siloed organizations can adopt and benefit immediately
- Operational and security requirements easily satisfied with existing IT infrastructure and minimal personnel skill increase

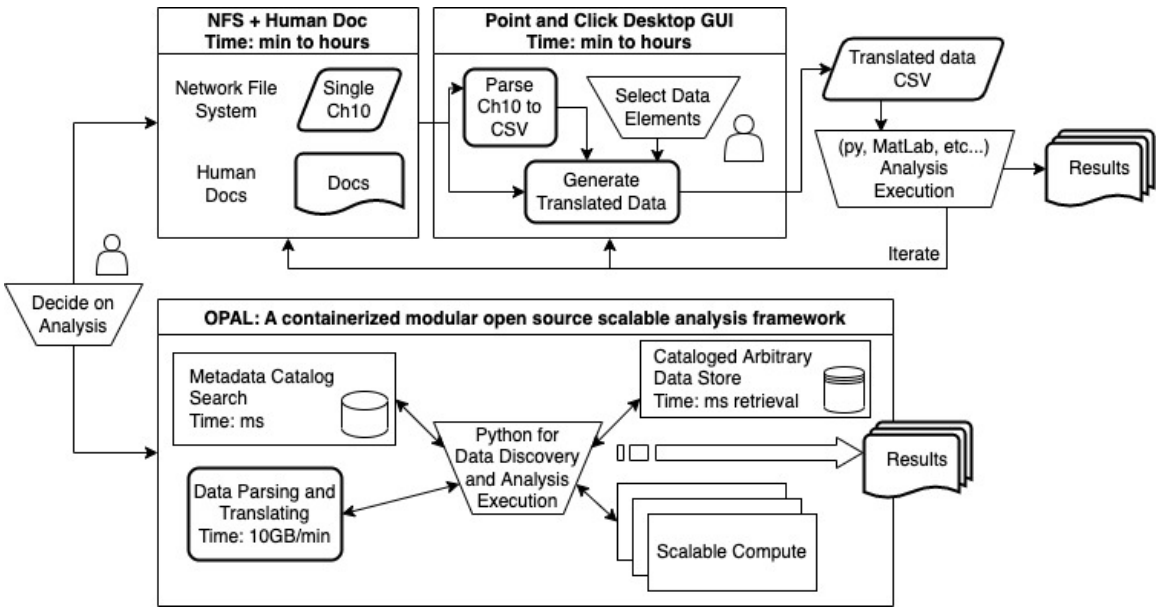

Figure 1: Analyses are driven by user needs. OPAL improves on previous proceses by (1) making data accessible and searchable, (2) Tightening the analysis iteration loop, and (3) Providing a unified environment

- Searching and feature filtering across thousands of flights' meta and all available time-series data at interactive-speed
- Low friction combination of data from disparate flight platforms and data sources
- Scalable storage and compute framework which enables analysis requiring machine learning
- Maximal use of OSS to limit duplication of existing technologies and to ensure performance and modernity

OPAL ARCHITECTURE AND DEPLOYMENT

OPAL includes a code editing and execution environment with a large collection of software packages chosen to enable access and perform analyses on telemetry data not previously possible. The OPAL software is deployed as an orchestrated set of container images with all OS files and configuration, shared libraries, and applications pre-configured. Currently Docker Compose is supported but Kubernetes is on the near term roadmap. The process to stand up OPAL is fully automated once air-gap transfer has been complete. The DevSecOps paradigm used by the OPAL development team is enabled by Platform One [3]. All software artifacts ported to air-gapped systems are outputs from continuous-integration pipelines being executed on IronBank or 'PartyBus'. This digital infrastructure that enables full pipeline visibility into common and approved security baselines for review by interested parties is critical. This visibility has paid large dividends by helping security offices understand how to securely leverage modern tooling on some of our unique datasets.

Once deployed and integrated with network authentication protocols, OPAL has the capabilities to interpret Chapter 10 data (See Section), save data to a common data store (through Minio),

```
tip_parse NASA.ch10 -o parsed_data -L warn
tip_translate_1553 parsed_data/NASA_1553.parquet
 ↪  DTS/DTS1553_Synth_Nasa.yaml -o translated_data
tip_translate_arinc429 parsed_data/NASA_arinc429.parquet
 ↪  DTS/DTS429_Synth_NASA.yaml -o translated_data
```

Figure 2: OPAL have built in Chapter 10 parsing and translating tools that are executable with a single command. Typical computation time with a standard desktop number of cores is 10GB/min.

catalog data (PostgresSQL and custom web interface), run online analyses (Juypter and associated packages), run big data (Dask), train and use neural networks, and run computer vision tasks. Section gives an example of this workflow.

TIP

In the workflow described for OPAL, the first step is parsing and translating flight files using TIP (TIP is included with an OPAL deployment). This is a multi-step process whereby a Chapter 10 file is processed, data are translated to appropriate engineering units or other formats such as video, and results are output in logical groupings of timeseries structured parquet files.

The initial step in this process, tip_parse, receives input of a Chapter 10 file and parses the file's packets along with their payloads. The Chapter 10 packet data are organized into structured formats that are grouped by bus type (e.g. Ethernet, Mil-Std-1553, etc.), and output to a set of parquet files along with relevant metadata produced during parsing.

At the time of writing, translation is implemented for both Mil-Std-1553 and ARINC 429 parsed bus data, and embedded video. The translation process requires inputs of a directory of parquet files generated by the parsing process, as well as a Data Translation Specification (DTS) in yaml format. The purpose of a DTS is to provide information about how the fields in parsed data are to be translated to engineering units, as well as to provide analysts information about data which are potentially present in the flight recording. An example of the command line arguments to perform the parse then translate steps are given in Figure 2.

TIP is designed to allow further development of parsing and translating capabilities. Examples of potential new capabilities would be supporting additional video formats or adding PCM data parsing and translating capabilities. Furthermore, there are DTS schemas that have been developed which are unique to both Mil-Std-553 and ARINC 429 translation. Any future bus translation implementations will require the development of its own unique DTS schema.

AN EXAMPLE WORKFLOW

A simple example workflow enabled by OPAL is delineated below with supporting example code snippets to illustrate the low barrier between all of the various steps to properly proccess data. The example was executed on a running intance of OPAL with data converted from NASA MatLab files to Chapter 10 using a tool developed by ATAC. It is important to note that a workflow does not always start with TIP parsing and translating. As shown in Figure 1 OPAL is a modular analysis

environment that is limited only by authentication controls and an analyst's creativity with python.

- Parse and translate single or mutiple ch10 files using TIP (Figure 2)
- Publish the resulting parquet files in the data catalog (Figure 3)
- Query the data catalog for the flights and data of interest (Figure 4)
- Analyze and plot data (Figure 4 and Figure 5)
- Scale as needed (Figure 6)
- As desired, save new data products to the data catalog
- Iterate

The value of this type of analysis framework has already been proven out by several groups both at a test range and within the 309th. Previous workflow times have been reduced by several orders of magnitude and analyses that were simply infeasbile on a meaningful timescale are now possible within days or weeks of starting. For parties interested in learning more about some of these results the reader is encouraged to reach out to the authors and supporting teams.

OSS MODEL

OPAL is Government Off-the-shelf software and as such is free to use for all government entities. However, this software was not developed solely by federal employees. The software development approach, in addition to leaning on the OSS community, is novel in the DoD and has led to rapid feature engineering and operational viability–from conception to production in about a year. Development of the platform source code, TIP (Chapter 10 parsing tool), and operations workflows have been a highly collaborative effort. The list of contributors includes federal employees, Metrostar Systems, and Avionics Test and Evaluation Corporation (ATAC). Other plans exist for future collaboration with other government and private groups.

CONCLUSIONS

OPAL has leveraged open source packages and modern analysis techniques to tackle the problem of analyzing a large set of legacy flight data. Custom scripts written in OPAL have been used to correlate telemetry data saved in Chapter 10 format across multiple flights, and we have demonstrated the ability to perform analyses across flight platforms. We have demonstrated that analysis tasks can be written and executed much faster than through existing workflows. Example data have shown analyses specific to flight data which originate in Chapter 10 format. OPAL is not solely a flight data analytics platform, nor is it intended to process only Chapter 10 files. The OPAL data model is custom-configurable by scripts which can be written within the OPAL development environment. It was designed to support the creation, upload, and publishing of arbitrary data types which are made searchable and exposed to the Catalog UI. We have used this arbitrary data model

```
from opal import kinds
kinds = kinds.load()

translated_kind = kinds.lookup("tip_translated")

1553_metadata = translated_kind.upload("translated_data/1553_translated",
↪  parent = "NASA.ch10")

arinc429_metadata =
↪  translated_kind.upload("translated_data/arinc429_translated", parent =
↪  "NASA.ch10")

from upload_many import publish_instance

publish_instance(1553_metadata["instance_id"], "tip_translated")
publish_instance(arinc429_metadata["instance_id"], "tip_translated")
```

Figure 3: OPAL has built in arbitrary object storage via the 'kinds' library that is configurable upon data upload. The kinds storage management system enables data lineage, fixed and arbitrary metadata, and is completely agnostic to datatype. After successful upload, in this example, the kind metadata are published to a searchable catalog for near instantaneous metadata search.

```
import matplotlib.pyplot as plt

df_1553 = translated_kind.read(1553_id, "NAV")
where_valid = df_1553[df_1553["NAV-0110"] & df_1553["NAV-0111"]]

fig, ax = plt.subplots()
ax = where_valid.plot(
    kind="scatter", title="Aircraft Position (1553)",
    x="NAV-23", y="NAV-21", c="NAV-25", s=1,
    cmap="viridis", figsize=(15, 10), ax=ax)

ax.set_xlabel("Longitude [deg]")
ax.set_ylabel("Latitude [deg]")
plt.gcf().get_axes()[1].set_ylabel("Altitude [ft]")
```

Figure 4: Read published data. This example plots latitude, longitude, and altitude telemetry that was translated by TIP. Result of plots in Figure 7

```
df_turbine_speed = translated_kind.read(t429_id, "Engine_Turbine_RPM_40")
df_exhaust_temp = translated_kind.read(t429_id,
↪  "Exhaust_Gas_Temperature_40")
df_1553 = translated_kind.read(1553_id, "NAV")
where_valid = df_1553[df_1553["NAV-0110"] & df_1553["NAV-0111"]]
ax = df_turbine_speed.plot(kind="scatter", x="time", y="N2_RPM",
↪  figsize=(13, 10), c='k', label="Turbine Speed [RPM, percent max.]")
ax.set_ylabel(r'Turbine Speed AND Exhaust Temp', c='k')
ax.tick_params(axis='y', colors='k')
df_exhaust_temp.plot(kind="line", x="time", y="Exhaust_Gas_Temperature",
↪  ax=ax, c='k', label="Exhaust Temp [deg C]")
ax2 = where_valid.plot(kind="line", x="time", y="NAV-25",
↪  secondary_y=True, c='b', ax=ax)
ax2.set_ylabel("Altitude [ft]", c='b')
ax2.yaxis.label.set_color('blue')
ax2.tick_params(axis='y', colors='blue')
ax.set_xlabel("Time [Epoch, ns]")
ax.legend()
title = ax.set_title("Engine Data (ARINC429) as Function of Altitude
↪  (1553)")
```

Figure 5: In this example, 1553 and ARINC 429 are combined into a single analyses and plot. However, using OPAL the commands are nearly identical after upload in the user configurable data storage.

```
from dask.distributed import Client, progress
import dask.bag
import dask.dataframe
import os

with Client(n_workers=16, processes=True) as client:
    ds_bag = dask.bag.from_sequence(datasets.values())
    ds_bag = ds_bag.filter(lambda ds: '1553' in ds and 'arinc429' in ds)
    ds_bag = ds_bag.map(lambda ds: get_rpm_at_cruise_altitude(ds["1553"],
    ↪  ds["arinc429"]))
    future = client.compute(ds_bag)
    progress(future, notebook=False)
    df = pd.concat(future.result())

    df.plot(
        kind="scatter", x=altitude, y=rpm,
        xlabel="Cruise Altitude [ft]", ylabel="Engine Turbine Speed [RPM,
        ↪  percent max.]",
        title="Turbine Speed at Cruise Altitude (all flights)",
        figsize=(15, 10)
```

Figure 6: Single analyses are easily scaled out to available cores using Dask. After cluster overhead, computation time decreases lenearly with resources as expected. OPAL attemps to make this type of light weight HPC easily accessible so large analyses approach interactive speeds.

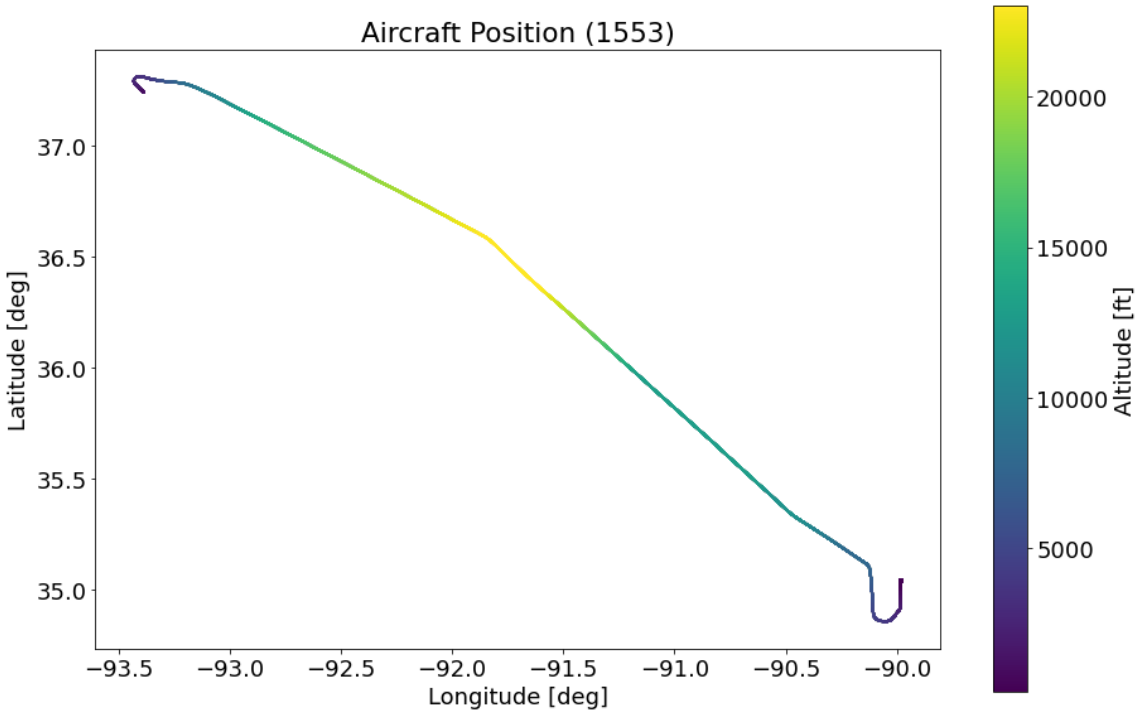

Figure 7: Result of code in Figure 4 This is a plot of latitude and longitude with the track colored by elevation.

to make OPAL conform to the data access patterns that we expect will be of most use to analysts within our domain. Our intention is that other groups will configure OPAL in their own way to maximize utility. The code base and customizations of OPAL are publicly available.

This work was only possible with the entire 309th EDDGE team as well as colloaborators and support organizations at all locations.

REFERENCES

[1] *IRIG 106 Chapter 10 Draft 06 Release*. Range Commanders Council U.S. Army White Sands Missile Range New Mexico 88002-5110.

[2] "Apache parquet." `https://parquet.apache.org/`.

[3] "Platform one." `https://software.af.mil/team/platformone/`.

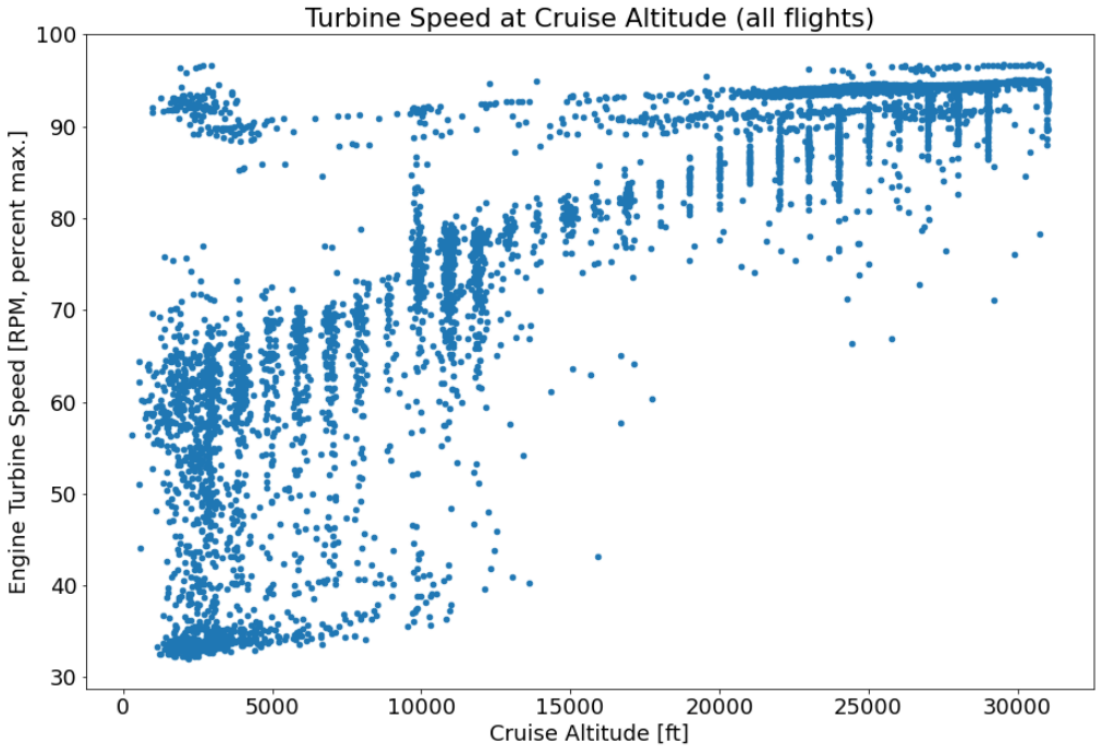

Figure 8: Result of looking across all 500 flights' 1553 and ARINC 429 data. Cruise Altitude is computed with a clustering algorithm on 1553 data and Engine Turbine Speed is extracted from ARINC 429 data.

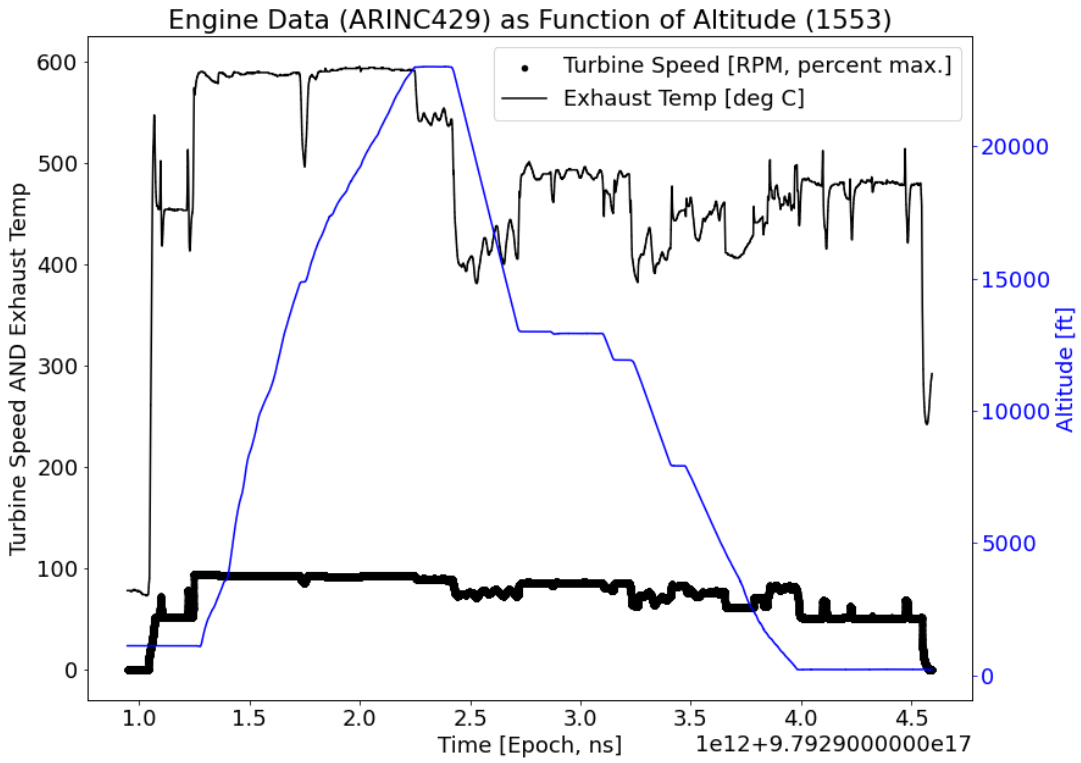

Figure 9: Data from a single flight used as a source for the multiple flight plot in figure 9. Time is in ns to highlight the fact TIP reports time as absolute time in nanoseconds since the epoch.